Extending Greenstone for Institutional Repositories

David Bainbridge,[1] Wendy Osborn,[2] Ian H. Witten,[1] and David M. Nichols[1]

[1] Department of Computer Science
University of Waikato
Hamilton, New Zealand
{davidb, ihw, dmn}@cs.waikato.ac.nz

[2] Department of Mathematics and Computer Science
University of Lethbridge
Lethbridge, Canada
osborn@cs.uleth.ca

Abstract. We examine the problem of designing a generalized system for building institutional repositories. Widely used schemes such as DSpace are tailored to a particular set of requirements: fixed metadata set; standard view when searching and browsing; pre-determined sequence for depositing items; built-in workflow for vetting new items. In contrast, Fedora builds in flexibility: institutional repositories are just one possible instantiation—however generality incurs a high overhead and uptake has been sluggish. This paper shows how existing components of the Greenstone software can be repurposed to provide a generalized institutional repository that falls between these extremes.

1 Introduction

Institutional repositories are a popular form of digital library. Although many software systems exist to support them, widely used ones (such as DSpace [1]) are tailored to particular requirements. They assume a certain metadata set and present readers with a fixed view of the collection when searching and browsing the repository. Depositing an item involves a pre-determined sequence of steps; the presentation of the pages in the sequence is difficult to customize; and the workflow involved in reviewing new items is built-in. Although with sufficient programming effort one can circumvent such restrictions—existing institutional repository systems do provide some hooks to facilitate a limited degree of personalization—it is fair to say that they are not designed with flexibility in mind. For example, it would be hard to adapt them to use a radically different metadata set or a different sequence of operations when depositing new items.

The Fedora framework [2] is an interesting exception that has been designed expressly with flexibility in mind—an institutional repository is merely one possible instantiation. However working with such a generalized system incurs a high overhead and such manifestations have been slow to emerge. One promising

development in this area is Fez [3], which we review with other institutional repository software solutions in Section 6.

The paper is structured as follows. First we discuss what we mean by a "generalized institutional repository." Section 3 demonstrates a minimalist example to help convey the salient features of such a resource. Then we describe how existing components of Greenstone were repurposed to give it functionality comparable to existing repository systems. Section 5 presents a second worked example to show how the new system can be configured to emulate DSpace's submission workflow. We conclude by placing the work in the context of other repository software: DSpace, GNU EPrints and Fez.

2 Background

Greenstone is a suite of software for building and distributing digital library collections [4]. It is not a digital library but a tool for building digital libraries. It provides a flexible way of organizing information and publishing it on the Internet in the form of a fully-searchable, metadata-driven digital library. Using it, a rich set of different types of collections can be formed that reflect the nature of the source documents and metadata available.

In extending Greenstone for institutional repository use our aim was to develop a software solution that transcends the limitations imposed by current solutions specifically targeted towards institutional repositories, without triggering the high startup costs of shifting to a highly generalized framework.

We want to enable librarians to turn any Greenstone collection into a repository into which new items and metadata can be deposited by authorized personnel through an ordinary web interface. But different Greenstone collections have different metadata sets, and there is no restriction on how extensive—or minimalist—such metadata can be. So when metadata is entered through a sequence of web pages, the content of these pages, the number of pages in the sequence, and the metadata items that each one requests must all be customizable. For one collection a single web form may suffice; another may require a long sequence of different forms. When the depositing user goes back to an earlier to step to correct a metadata entry this variable amount of data—which is entirely dependent on the metadata set in use—must be remembered by the web browser.

We use the following notion of "generalized institutional repository":

- *The digital library collection can use any metadata set.*
- *Depositing an item can involve any number of steps.*
- *The stages involved in depositing an item can be designed individually.*
- *Flexible workflow.*

Depending on institutional procedures librarians may have roles such as 'reviewer', 'approver' or 'editor' for deposited items [1].

3 Example of Operation

To help illustrate the core business of an institutional repository, here is a minimalist example. Imagine a Faculty of Arts that has moved to a digital solution—couched as an institutional repository—that replaces the physical photographic color slide resource that the Faculty previously provided.

Figure 1 shows the submission process, which has in fact been developed using the newly extended version of Greenstone. A single page is used to gather salient facts before an item is deposited. Only four items of metadata are requested along with a picture of the artwork: title, artist, date and notes. A real-world version would most likely request many more fields than this.

To reach this page the user has already had to log in. In Figure 1a she is selecting the destination collection (the Art History repository). In the next step (Figure 1b) she has used the file browser that is launched by pressing the "Browse …" button to locate the artwork to submit, and entered metadata describing the items (Title: The Bower Meadow; Artist: Rossetti; Date: 1871–1872) along with notes about the painting. Along the bottom is a progress bar with a triangular marker showing the current position ("specify metadata").

Clicking on "deposit item" takes her to the next step (Figure 1c) where the new information is digested into the collection, which occurs in a matter of seconds. The final step is to view the collection, which is shown in Figure 1d where the user is browsing the Art History Repository by title. The repository is clearly in its early stages with only three items added so far, with the newest addition, *The Bower Meadow*, listed at the top.

4 Implementation

Only a modest amount of development work was necessary to extend Greenstone to support the notion of generalized institutional repository given earlier. The three enabling technologies were macros, runtime actions, and incremental building, all of which exist in Greenstone.

Greenstone macros are the key to controlling the generalized workflow. Checking form content and manipulations of form layout (adding in previous values etc.) are spliced into macros through JavaScript and DOM manipulation. To enable document submission, an existing runtime 'action' called The Collector [5], which supports the creation and building of collections through a web browser, was further abstracted and generalized. This 'action' was already able to provide a progress bar and used a database to store previously entered values from one page to the next. The new extension was to add support for multipart form file-upload with the new action called "the depositor." Incremental building using the Lucene indexer [6] is already a feature of Greenstone.

4.1 Macros

A Greenstone installation's look and feel, page structure and language interfaces, are

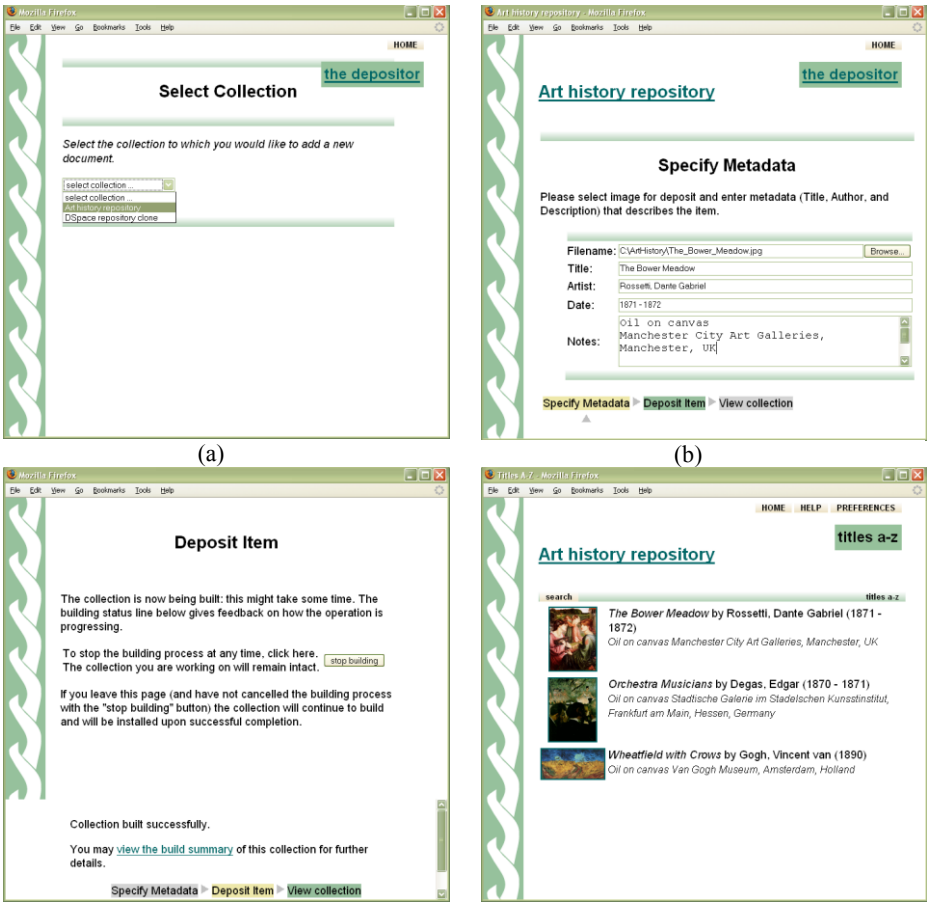

Figure 1: A simple example (a) selecting the Art History repository (b) selecting and image and entering metadata (c) depositing the item (d) browsing the collection

all achieved using a simple macro language. Figure 2 shows an artificial 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 2 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

```
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] \{D\'emarrer la recherche \}
_header_ [l=es] \{Iniciar la b\'usqueda\}

# ... and so on
```

Figure 2: Excerpt of macro file syntax to demonstrate main features.

content. These are known as "page parameters" because they 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 2 three versions of the macro *_header_* are defined within the "query" package, corresponding to the English, French and Spanish languages. They set the parameter *I* to the appropriate two-letter international standard abbreviation (ISO 639), enabling the system to present the appropriate version when the page is generated.

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 2, 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.

4.2 Controlling the workflow

Figure 3 shows edited highlights of the macro file that produces the simple workflow shown in Figure 1. Key points in the file are:

- *_numsteps_,* a compulsory macro that defines the number *N* of stages in this submission process.
- *_step1content_, _step2content_, ... _stepNcontent_* is the convention used to define the page content that is displayed, along with *_stepNtext_* which controls what appears in the progress bar.

```
package depositor

_numsteps_ {1}

_textstep1_ {_textmeta_}
_laststep_ {build}
_textlaststep_ {_textbuild_}


_step1content_ {

<form name="depositorform" method=post action="_gwcgi_"
  enctype="multipart/form-data">
<input type=hidden name="p" value="_cgiargp_">

<center><h2>_textstep1_</h2></center>
<p>_textimagesimpledesc_</p>

<p><table>
<tr>
  <td>Filename:</td>
  <td> <input type=file name=dauserfile value="_userfile_" size=61></td>
</tr>

<tr>
  <td>Title:</td>
  <td> <input type=text name=damd.dc.Title value="_damd.dc.Title_"
         size=74></td>
</tr>
<!-- and so on … -->
</table></p>

<!-- … -->
<p>_depositorbar_</p>
</form>
}

_textmeta_ [l=en] {Specify Metadata}
```

Figure 3: Excerpt of macro file for producing the first step of the submission process for the Fine Arts Repository example.

- *_step1text_* in this example is defined to be *_textmeta_*, another macro (defined at the bottom of Figure 2) which resolves through the language independence feature to "specify metadata" when viewed in English
- *_laststep_* controls how the workflow ends: for example, automatic building the collection, or going to the collection's editor for review.
- *_depositorbar_* is defined by the runtime system (the depositor action). It is formed by composing the information represented in *_numsteps_* with *_step1text_*, *_step2text_* and so on. *_laststep_* specifies which of the predefined endings terminates the submission process (e.g. *_contentbuild_* and *_textbuild_*). Since these two are stored in the macro files they can be refined and extended as needed.

5 Extended example: emulating DSpace

To demonstrate the versatility of the design, a submission workflow in Greenstone was developed that closely emulates DSpace's [1]. Since both are open source systems, much of the HTML was transferred directly. The functionality is very similar, the difference being in how a submission involving multiple files is handled—as when submitting a web page including external resources such as images. In DSpace each file must be individually specified from within the form-based submission process. Since Greenstone can already handle archive formats such as Zip and Tar, we decided to ask the user to submit multiple-file works in this form. All files that make up the work must still be identified, but this happens outside the form-based submission, and is usually easier since the files can be multiply selected in one go.

In DSpace, runtime functionality for the submission process is handled by the server. If Title metadata is compulsory this is checked when the user proceeds to the next step of the submission process. In Greenstone the analogous functionality was embedded into each web page using JavaScript. This offers more flexibility to customize the workflow and more immediate feedback to the user.

Figure 4 shows snapshots of a faculty member working their way through the Greenstone adaptation of the DSpace submission procedure. To submit an item of work the user starts by logging in, and then selects the DSpace repository clone collection. Using Greenstone's collection macro override facility, this repository provides its own tailored workflow—eight steps in all, visible at the top of the snapshots. In the simple example of Figure 1 the progress bar was located at the bottom of the page, but it is easy to move the position of the macro *_depositorbar_* within the structured HTML to move it to at the top. In DSpace the progress bar is implemented as a series of images, and although we could have emulated this we chose not to because there is an existing Greenstone facility with the same function—furthermore it makes it easier to change the color scheme, fonts and wording used. (We tend to avoid textual images in Greenstone to facilitate multilingual operation.)

The scenario here is a university that uses DSpace-style submission to manage its staff's digital outputs. In Figure 4a the instructor for a Machine Learning course is at the first page of submitting a lecture on Bayesian networks. He has entered his and a colleague's name, the title of the talk and its type (a presentation). Other fields such as series, report number, and ISBN are not relevant and so he leaves them blank.

In Figure 4b the instructor has moved to the second step, which prompts for descriptive metadata: keywords, abstract, sponsors, and description. Again not every field is relevant. For each part of the form contextual help is available that describes the purpose of the field. In Figure 4c he has moved to the point where the file (in this case PowerPoint) is requested. Next (Figure 4d) information is displayed about the file transfer from submitter's computer to the server. The checksum is shown so he can check that no transmission errors occurred. This is accomplished using AJAX technology [7] to retrieve the information from the server in an extensible manner.

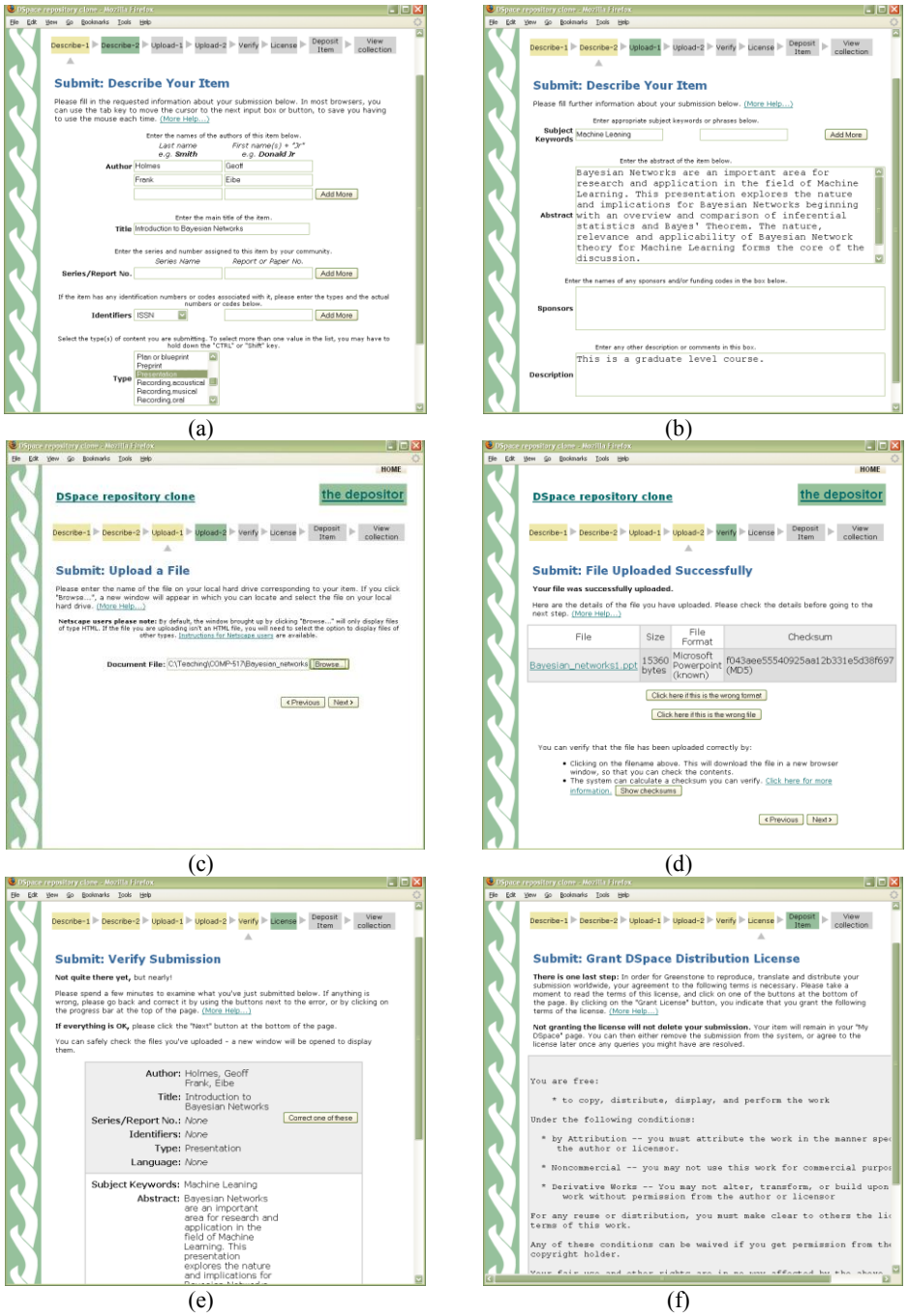

Figure 4: Emulating the submission workflow for DSpace (a) primary metadata (b) secondary metadata (c) select file (d) check file (e) review metadata (f) choose license.

The fifth step (Figure 4e) provides an opportunity to review and edit all information entered so far. It is also possible to return to any previous stage by clicking the progress bar. Making the system remember existing fields—even when they support an arbitrary number of values, as with authors—is tricky in JavaScript but possible. Figure 4f shows the final user input page, where the user decides whether to grant the distribution license. If he does, the PowerPoint presentation, along with its metadata, is time-stamped and deposited into the collection area. The collection's editor will be notified by email, and/or the collection will be incrementally rebuilt, depending on the settings in the collection's configuration file.

6 Discussion

We now discuss the context into which this work fits by summarizing the key points to software solutions being used as institutional repositories.

DSpace is specifically designed as an institutional repository. It is a popular choice by organizations to provide a digital repository that harnesses the output of their institution. It requires an IT specialist to install, which is commensurate with the typical organizational environment in which it is used. Some customization is possible but because runtime functionality is locked up in the server it is ostensibly a fixed workflow from a librarian's perspective. Full text indexing is possible, but only limited to a small number of native file formats.

GNU EPrints [8 is another popular choice with over 200 known installations worldwide**.** Rather than spanning an entire organization, many EPrints installations are deployed in a niche role by an entity within the organization, although it can and is deployed in a wider context. It is easy to install and it includes configuration files that control the metadata in use and the document types supported**.** Ironically enough, it has been the use by niche disciplines that has driven the need to support different metadata sets rather than the unified "one shoe fits all" approach seen in DSpace; however, it lacks the notation of communities and collections, which enables a repository to be used in different ways across an organization. EPrints supports full text indexing.

Fez [3] is an emerging software solution for institutional repository use. In beta form at the time of writing, its notion of generality and configurability is more ambitious than the above two systems. It is built on top of Fedora, and is exactly the sort of development the framework is aimed at. Fez utilizes the rich complexity of the Fedora framework to deliver a system tailored for institutional repository use. It includes the concept of communities and collections, configurable workflow and metadata. While Fedora can handle full text indexing, this ability is not exposed through Fez, and there are some compatibility issues with connecting Fez with a framework that is still itself under development.

7 Conclusion

We believe that Greenstone provides the following advantages for institutional repositories: trivial to install; configurable workflow that works with any metadata set and document type; variable number of steps; collection based with support for customization; incremental building that with full text indexing across a wide range of formats include HTML, PDF, Word, PPT, email, as well as automatic metadata extraction; and language independence.

All systems we have mentioned are open source, which means that anyone wishing to evaluate them can do so freely. In practice, however, considerable effort may be needed to do a trial—installation alone is often a major stumbling block [9]. (On more than one occasion we have met library staff who have spent months trying to get a trial installation up and running.) This would be easier if developers provided a sandbox for others to try their system out (one exists for GNU Eprints). Ours is at *www.greenstone.org/ir-sandbox/*

References

1. Tansley, R., Bass, M. and Smith, M. (2003) DSpace as an Open Archival Information System: Status and Future Directions. *Proc ECDL* pp. 446-460.
2. Lagoze, C., Payette, S., Shin, E. and Wilper, C. (2006) Fedora: an architecture for complex objects and their relationships. *Int J on Digital Libraries* 6(2) 124-138.
3. Fez. *http://sourceforge.net/projects/fez* [accessed 30 June 2006]
4. Witten, I.H. and Bainbridge, D. (2003) *How to build a digital library*. Morgan Kaufmann.
5. Witten, I.H., Bainbridge, D. and Boddie, S.J. (2001) Power to the people: end-user building of digital library collections. *Proc Joint Conf Digital Libraries*, pp. 94-103.
6. Lucene. *Apache Lucene*. *http://lucene.apache.org/* [accessed 30 June 2006]
7. Crane, D., Pascarello E. and James, D. (2005) *Ajax in Action*. Manning.
8. Eprints, *http://www.eprints.org/* [accessed 30 June 2006]
9. Nixon, W. DAEDALUS: Initial experiences with EPrints and DSpace at the Univ. of Glasgow, *Ariadne* 37, Oct 2003. *http://www.ariadne.ac.uk/issue37/nixon/* [accessed 30 June 2006].