A Fedora Librarian Interface

David Bainbridge and Ian H. Witten
University of Waikato
Hillcrest Road
Hamilton, NZ
{davidb,ihw}@cs.waikato.ac.nz

ABSTRACT

The Fedora content management system embodies a powerful and flexible digital object model. This paper describes a new open-source software front-end that enables end-user librarians to transfer documents and metadata in a variety of formats into a Fedora repository. The main graphical facility that Fedora itself provides for this task operates on one document at a time and is not librarian-friendly. A batch driven alternative is possible, but requires documents to be converted beforehand into the XML format used by the repository, necessitating a need for programming skills. In contrast, our new scheme allows arbitrary collections of documents residing on the user's computer (or the web at large) to be ingested into a Fedora repository in one operation, without a need for programming expertise. Provision is also made for editing existing documents and metadata, and adding new ones. The documents can be in a wide variety of different formats, and the user interface is suitable for practicing librarians. The design capitalizes on our experience in building the Greenstone librarian interface and participating in dozens of workshops with librarians worldwide.

Categories and Subject Descriptors

H.3.7 Digital Libraries, H.5.2 User Interfaces.

General Terms

Design, Human Factors.

Keywords

Graphical User Interface, End-user Collection Building, Fedora Digital Repository

1. INTRODUCTION

When selecting a digital library system for their online content needs, organizations such as libraries, museums and archives are faced with a potentially bewildering array of choices. Open source software is often preferred, because its aims and general philosophy are strongly aligned with the founding principles of public open-access organizations, but even here there is a wide selection. Cheshire, DSpace, ePrints, Fedora, and Greenstone are principal contenders.[1]

Each of these systems has its strengths and weaknesses. DSpace, for instance, is heavily optimized for institutional repository use. Ready to use out of the box, it has been widely adopted for this purpose. However, looking through the blogs of people associated with host institutions that use it, one encounters entries that describe some surprising limitations which can be as rudimentary as the metadata set being fixed at Dublin Core.

Fedora, in contrast, lies at the other end of the spectrum [5, 6]. It is founded upon a powerful digital object model, and is extremely flexible and configurable. However, there is a price: if institutions are to make use of Fedora, programming support by IT staff is a prerequisite. Unlike DSpace, establishing a useful Fedora-based digital library is not a simple matter of downloading the software, following the installation procedure and adjusting some entries in configuration files. All that must be done, of course, but much more is required. It is really necessary to undertake software development in order to be able to ingest existing documents and other content into the system and control how it is presented to users. This state of affairs, favoring extreme flexibility over a more targeted (and hence restricted) system, is a deliberate design decision by Fedora's architects.

Nevertheless, in most institutions it is librarians, not programmers, who must populate the digital library. Consequently we have created a front-end called the Fedora Librarian Interface (FLI) that is specifically designed to allow end-user librarians to place articles, existing metadata, and other material into Fedora repositories. Through an interactive GUI (graphical user interface), FLI allows users to drag documents into a Fedora repository, downloading them from the web if required: support exists for utilizing protocols such as OAI-PMH, Z30.50, SRW/U in addition to basic web mirroring. Users can add metadata to the documents through a convenient interactive interface, or associate full-text documents with already-existing metadata for example, a file of MARC records. They can configure the plugins that process the documents using a host of options that are amply documented by mouseover text (tool-tips). The system minimizes the work that a librarian must do for each document individually. Moreover, the interface is multilingual. In order to create this system we have capitalized extensively

[1]Coincidentally, the designers of these systems appear to be working their way through the alphabet!

JCDL'08, June 16–20, 2008, Pittsburgh, Pennsylvania, USA.

on our experience in producing Greenstone and interacting with librarians all over the world to refine and enhance its capabilities.[2]

The structure of the paper is as follows. First we give an overview of the Fedora software and review related third-party user interface based projects that extend its functionality. Next we give several examples of using the new Fedora Librarian Interface and contrast this with the approach required using the existing tools. Following this we broaden the description of the capabilities of the new librarian interface, as even the detailed examples provided do not cover its full capabilities. We conclude with a summary of the work.

2. OVERVIEW OF FEDORA

> *Fedora is open source software that gives organizations flexible tools for managing and delivering their digital content. At its core is a powerful digital object model that supports multiple views of each digital object and the relationships among digital objects. Digital objects can encapsulate locally-managed content or make reference to remote content. Dynamic views are possible by associating web services with object.*
>
> www.fedoracommons.org

Download and install Fedora and you will find yourself with a Java based system. Apache Axis and Tomcat are combined to provide a framework that supports servlets and web services. A simple web-based interface is provided that exemplifies the core functionality available. To start with the repository is of course empty and so included with the software are some demonstration objects that running a command line script will ingest to allow a user to experiment with the system.

Figure 1 shows a selection of snapshots using the default web interface.[3] Rather than use the demonstration objects, we have skipped ahead in terms of this paper and show the result of ingesting documents and metadata with the new librarian interface; this does not effect the general understanding of the default web interface but has the advantage of saving space. If Figure 1(a) the user has called up the main query page, and searched for documents that include the word "Canada" in the title. In Figure 1(b) they have clicked through to view the top-most item, the Fedora object for a digitized image, "a map of Canada and the north part of Louisiana ..." From the resulting object project view, the user can inspect basic properties about the object—id, label, content model, object type, creation date, last modified date, and owner ID—view datastreams and test-run disseminators. A datastream is MIME-encoded data (notionally static) associated with an object: for instance a source file such as an image or PDF file, or XML data. Disseminators are methods that act upon an object, and are linked to web-services that provide dynamic capabilities for the object, implemented by accessing the object's datastreams.

Figure 1(c) is called the index view, and is the result of running the default disseminator for the object, effectively listing the datastreams to the object. Presented as a web page, the datastream names are hyperlinked, and if one is clicked upon it serves up the underlying MIME-encoded data source. Clicking on *DC* yields the XML datastream representing Dublin Core metadata stored about the object. In Figure 1(d) the user has clicked upon the *url* datastream, which displays the digitized map.

Figure 2(a) shows the disseminators associated with the digital object. These are a direct consequence of this image being associated with the the content model UVA_STD_IMAGE—a content model provided with Fedora that supports a range of image manipulation, such as cropping, watermarking, and so forth—through the *RELS-EXT* datastream. In the default web interface the disseminators are listed, along with web form input components (radio buttons, text boxes and the like) that match to the parameters each disseminator takes. Taking *cropImage* as the example, the user has entered co-ordinate information (x, y, *width*, and *height*) and pressed the "run" button. The resulting excerpt is shown in Figure 2(b).

This is all well and good, but having experimented with some documents (such as the demonstration ones provided) what is the next step? Broadly speaking, there are two distinct IT roles to developing a Fedora digital repository: i) file processing, typically batch processed, to prepare documents for ingesting; and ii) the end-user interface, typically web-browser based. Leveraging off web-based technology to yield interface functionality and control presentation is quite a different skill set to the batch processing of file formats. One could even imagine two separate IT teams fulfilling these roles. The former is the focus of this paper, and ultimately we demonstrate the role a graphical interface tailored for librarians and other digital collection managers can play; but for now, we restrict ourselves to considering the possible options that utilize the standard building blocks provided by Fedora.

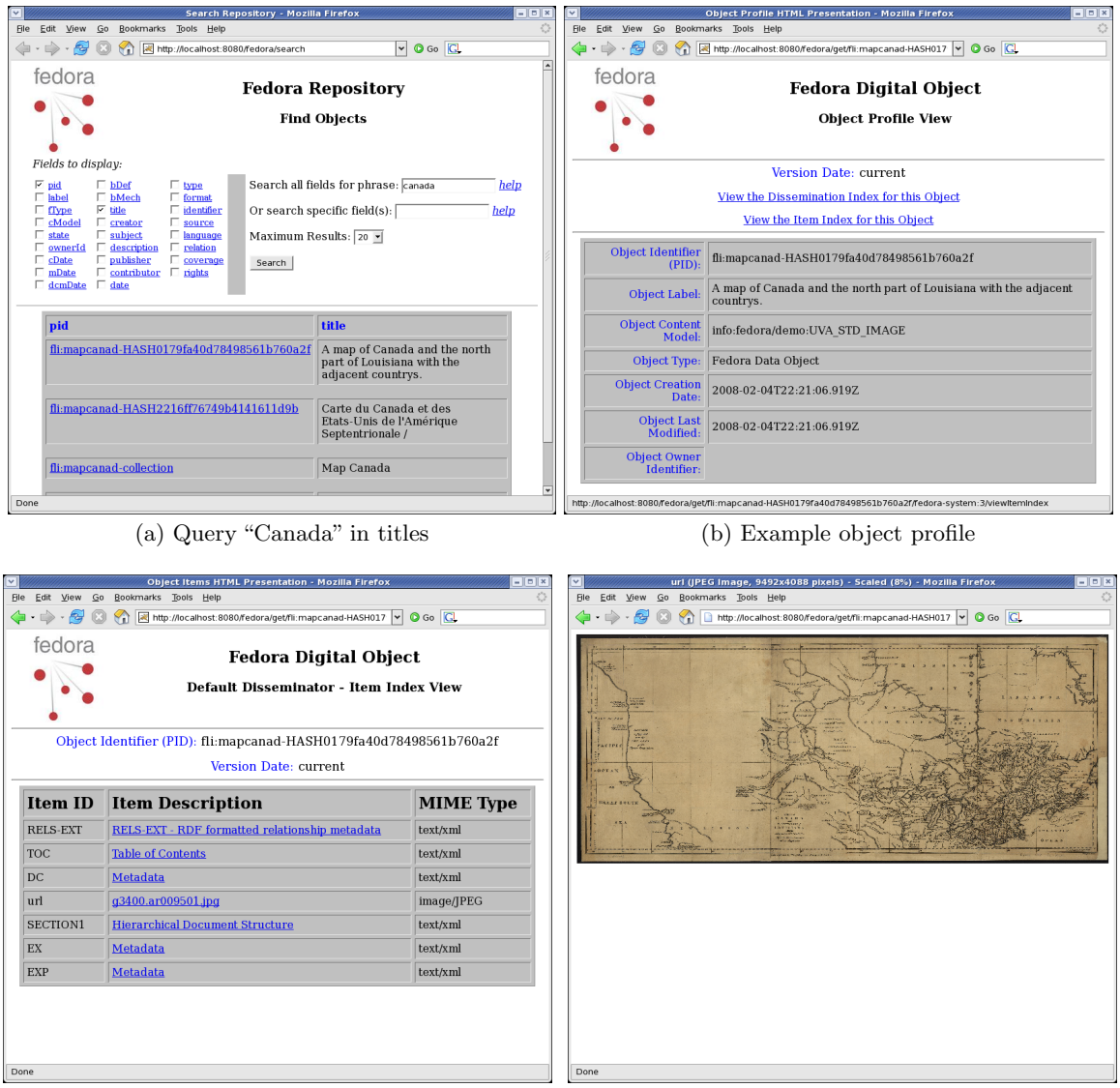
For ingesting content there are three main choices:

- Use the Fedora administration application (GUI-based).
- Write specific code for file processing and conversion (command-line).
- A GUI/Command-line hybrid.

A fourth category is document submission through a customized interface, the most popular form being through a web browser. This is usually part of a larger browser-based management interface: such systems are a blend of programming requirements to ingest and user interface design, with programming time spent developing the latter taking the lion's share. Unlike developing a handful of command-line scripts to help convert files in one format into another, the programming investment in such a hybrid is significant. Development projects tend to be clearly aligned with a particular community, and a trend in this type of work is to try and generalize the work and package it so others in the same line of work can make use of it. In Section 2.4 we review pertinent projects in this area.

[2]See *www.greenstone.org/map* for an overview of the number and location of workshops and tutorials given. Librarians are the dominant group in our e-mail discussion group, with members hailing from over 50 countries.

[3]The default web interface is not intended to be used as a final product, rather the be indicative of what is possible and to allow the developer to explore the repositories capabilities.



(a) Query "Canada" in titles

(b) Example object profile

(c) Example index view

(d) *url* datastream

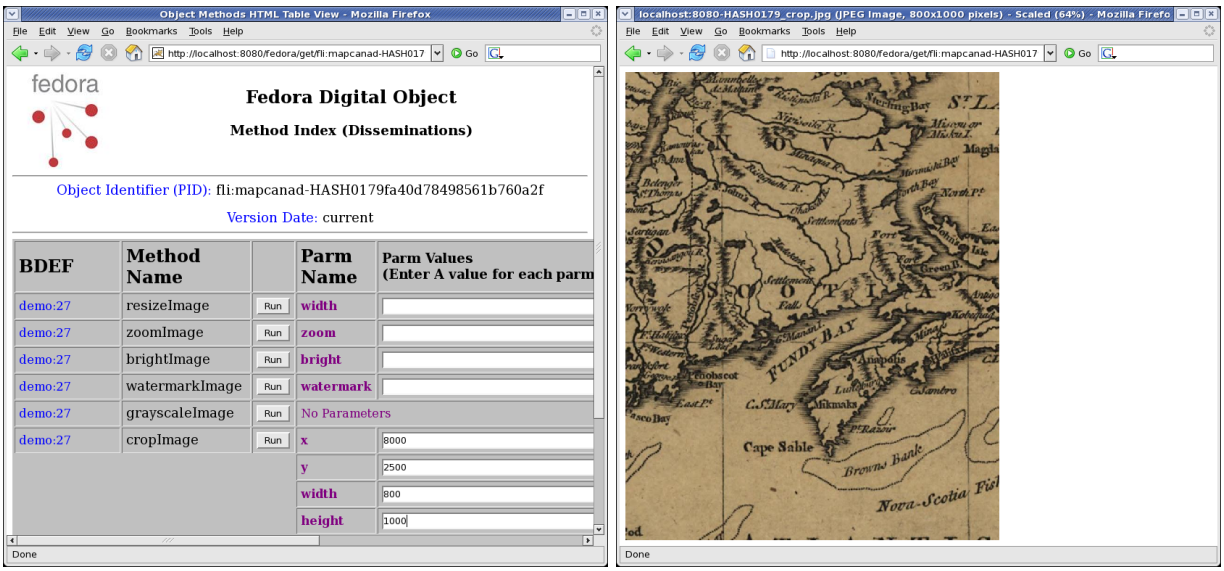
Figure 1: Default web interface to Fedora 3.

2.1 Administration application

Figure 3 is an illustrative snapshot of the Fedora administration interface. Having provided login details, the application presents a virtual desktop area, blank to begin with, within which windows appear as determined by the user's interaction. Using this application it is possible to access a wide range of capabilities. On a per item basis, interactively a new object can be created, or an existing one edited or removed (purged) from the repository. Alternatively batch driven commands can be initiated that process a set of files: build, ingest or modify. Files must either be in FOXML or FedoraMETS, or conformed to an object-specification that is combined with a templates to generate FOXML/FedoraMETS. Support is provided for searching the repository, from which items can be selectively manipulated. There is also access to consoles that connect with the Management and Access APIs to Fedora.

In Figure 3 the user has performed as series of operations. Near the bottom is the result list of a search (like before for titles matching "Canada"). Double clicking on the top-most item—the same document as in the web-browser example—produces a new window (located in the upper-left quadrant) through which datastreams and disseminators can be viewed and if desired altered. The *DC* datastream is currently selected, which presents this metadata formatted in XML. Options available for editing this are to edit the XML directly in a text box within the interface, or to launch an external application, for instance an generic XML editor. Exploring the datastreams for this object, the user has viewed the *RELS-EXT* datastream, and learned that the content model associated with this object is UVA_STD_IMAGE.

The Content Model Architecture is an enhanced feature in Fedora 3 over previous versions. It is a way of establishing fundamental characteristics that are shared across a set of digital objects. The building blocks for the content model are behavioural definitions (bDef) and behavioural Mecha-



(a) Disseminators for content model (b) Crop Method

Figure 2: Fedora Disseminators.

nisms (bMech),[4] which, like the content model, are merely additional objects stored in the repository. Their role is to provide an abstract set of of methods (bDef) and particular implementations (bMech). To take a more concrete example, a bDef tailored for image content might provide a method that generates thumbnails at runtime, suitable for display in a web browser. bMechs might then be provided that handle different image types: say one bMech that can handle JPEG, GIF and PNG formats as Java has standard support for these; a separate bMech could be written to process JPEG2000 files as this would require additional library support that might not mesh well with the built in ones. Given the mechanism is accessed through web services, it might even be that it is implemented in an entirely different programming language. Regardless of which bMech is used, the essential characteristic that an image can have a smaller web-browser friendly preview image can be set up as part of a content model and shared across a variety of source documents.

In our example, the user wishes to understand more about the content model associated with the map. They have called up the object that encapsulates its content model, displayed in the lower-right quadrant of Figure 3. From this they learn the object identifier of the corresponding bDef, call this object up and continue their exploration locating the bMech. For clarity in the figure these have been iconified, displayed along the bottom of the interface, and can be be opened again by clicking on them.

For comparison with later examples using the new librarian interface, a brief synopsis of creating a new object using the administration tool is now given. Creating a new object from scratch with the administration interface is very much a hand-crafting process. Upon creating a new object (file→New object...) the user is prompted, through a popup window, to provide a descriptive label and a customized identifier (or else elects to have one automatically generated by the system). Lets assume they wish to have a PDF document represented. After the popup window, next appears a window similar to the map object in Figure 3, only more sparsely populated. The *DC* datastream is there but with minimal content: the main item of note is the label entered when the object was created and is stored as *dc.Title*. Having set further Dublin Core metadata by editing the XML for this datastream, the user then introduces a new datastream for the object by clicking on the "new ..." tab, calling it *pdf* and filling out the fields appropriately, opting to have the datastream managed by the repository, and pressing the *import* button and navigating to the appropriate file. For *RELS-EXT* a similar interaction sets the content model. There is no particular restriction on the order of things, the user could have just as easily set up the *pdf* stream first and/or the *RELS-EXT* datastream before editing the Dublin Core metadata.

2.2 Command Line Script

At some stage command-line scripts become a necessity, for any non-trivial sized collections, as a way of getting the data in the format you have it in, into one of the two formats Fedora supports natively: FOXML or FedoraMETS. A common scenario would be one where data is exported from the existing database/catalog/repository in a format that system supports, say MARC or EAD from a catalog, OAI for metadata from a document repository, or a native format like the one used by DSpace for exporting so as to have the source documents included as well in the exported data.

The exported data then needs to be converted into one of Fedora's supported formats. A developer is essentially free to choose any programming language to do this, subject to any institutional policies or constraints. All the major programming languages include rich API/library support for processing XML and related technologies such as XSLT, for

[4]These terms make use of existing terminology from Fedora 2; the Fedora development team have indicated that these terms are likely to be changed in the near future to minimize confusion.

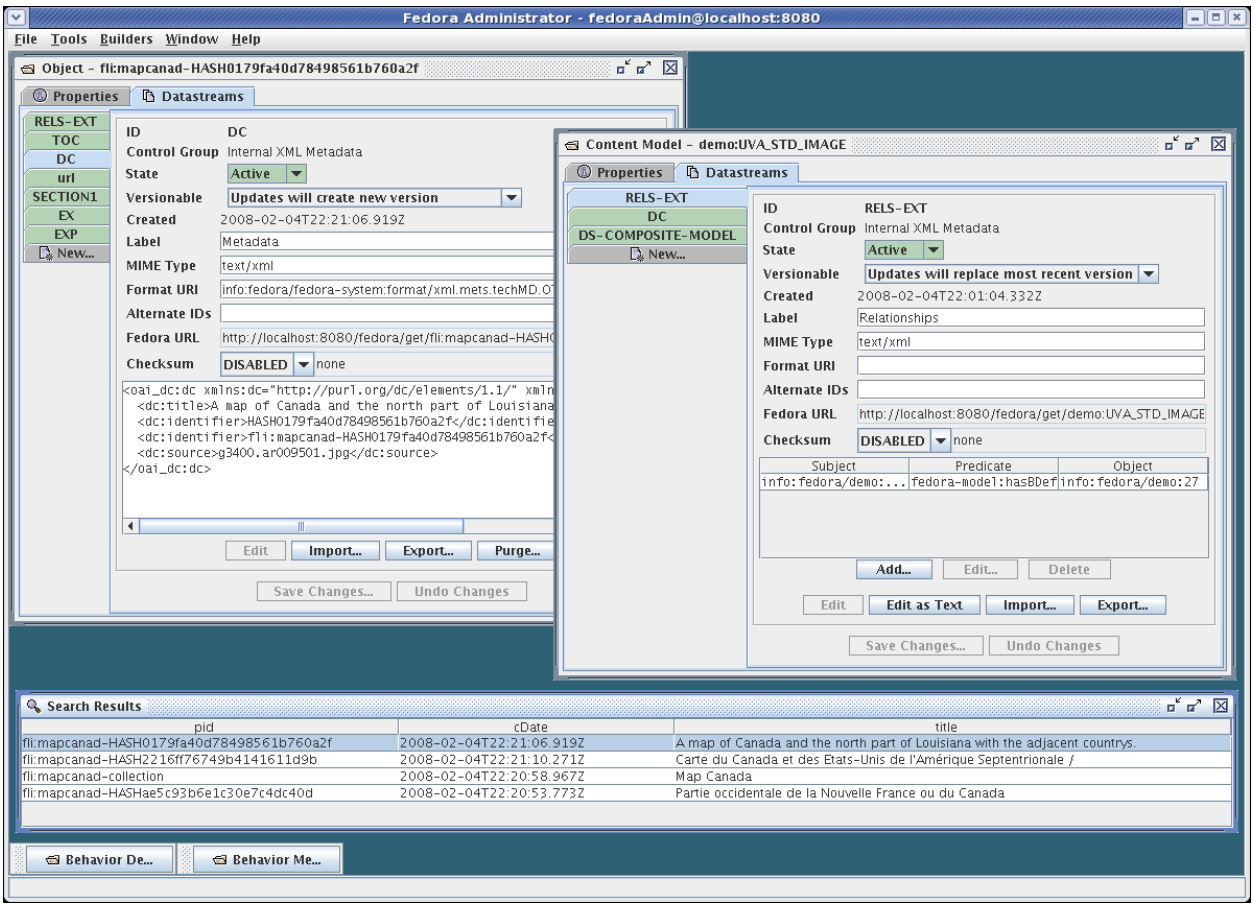


Figure 3: The Fedora Administration Application.

instance, and comparable mechanisms for manipulating files, meaning no one language is superior in this regard.

2.3 Hybrid

A hybrid approach combines command-line scripting and the administration GUI. While scripting is ideal for large scale processing, by necessity this is accomplished through automated rules. These do not perform so well when exceptions need to be handled, and left unchecked results in poor quality data overall in the repository. The graphical interface, therefore, is an ideally suited mechanism that can be used to counter this. In its most basic combination, the automated script is run first, and then the administration interface is used to locate erroneous items and correct them.

Alternatively, a more intricately spliced workflow would be to partially prepare the data to be ingested with command line scripts, and then shift to the graphical interface to oversee the final stages. HTML, for example, is a problematic document format to ingest through the Fedora administration interface due to the numerous associated files: images, style files, and the like. This would be tedious to handle through the interface. More attractive would be to have the HTML and associated files batch processed into self-contained folders, one per document. The librarian can then manually ingest these "packaged" items, maintaining high quality metadata control as they go utilizing the other features of the tool.

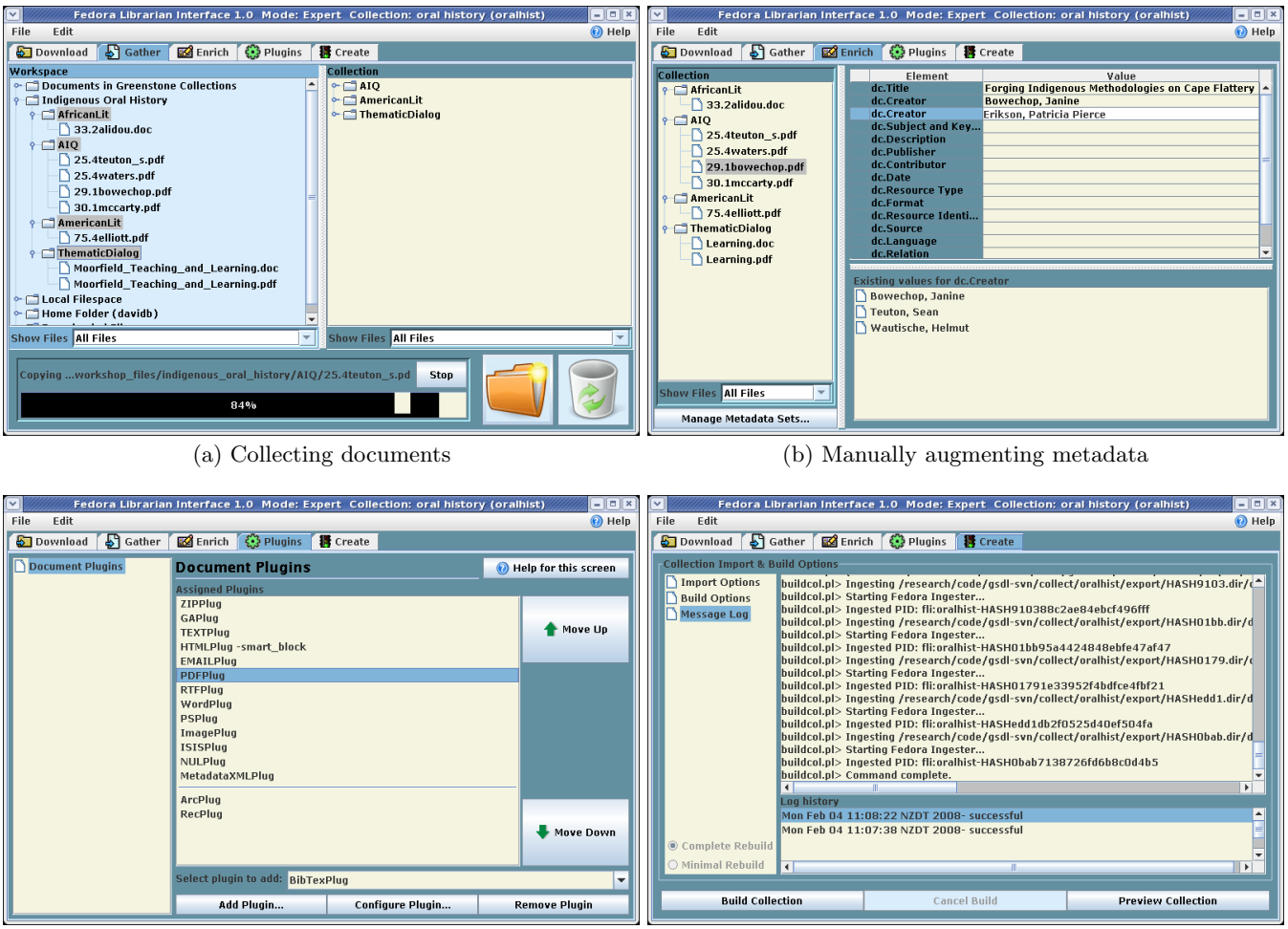
2.4 Closing the Gap

As our discussion above illustrates, getting things done with the tools provided by Fedora presupposes a high level of IT expertise, even when using the graphical user interface. Reiterating from before, this is the price to be paid for generality and is a deliberate design decision. But there is a danger of individual projects continually reinventing the wheel. Fortunately (and not entirely unpredictably) there are various projects that aim to minimize—for a particular domain—the IT development needed, and moreover can be reused—helping close the gap between what the user in such a domain wants to achieve and the technical steps that are necessary to accomplish this.

Elated [1] is produced by the Associated Colleges of the South (ACS), a consortium of 16 US liberal arts colleges and universities. It provides a lightweight, general-purpose web-based client for Fedora, primarily suitable for a digital assets management system, or institutional repository. It is servlet based, like the base Fedora system, but utilizes a separate database (mySQL) and consequently its own user authentication mechanism.

Fez [4] from the University of Queensland, Australia, provides a web interface to FEDORA targeted at library or similar institutions focusing on institutional repository use. It requires PHP and mySQL, and includes migration notes for those shifting from DSpace, ePrints or Elated.

Muradora [3] from Macquarie University, Australia, pro-



(a) Collecting documents

(b) Manually augmenting metadata

(c) Automatically associating documents together

(d) Combining metadata and source documents

Figure 4: Example snapshots of using Fli.

vides a web front-end to a Fedora repository and re-factors Fedora authentication and authorization into pluggable middleware components. While its primary focus is on authorization, providing a Shibboleth authentication module as well as extending XACML, it also delivers rich user-based searching and browsing capabilities. Installation at present, though, is quite complex: LDAP, DB XML, Orbeon XForms are just a few of the sub-components that are needed in addition to a base Fedora installation.

FABULOUS (Fedora/Arrow Batch Utility with Lots of user Services) augments standard Fedora command-line functionality, for instance allowing for batch activating and deactivating datastreams; of note, it can link batches of content files to existing metadata. It is controlled through a PHP based web interface.

Our contribution in this area is a graphical interface for Fedora focusing on the librarian as the primary end user that embodies a workflow centric ingest process. The work draws upon many years of experience in HCI design for digital library users, in particular from the Greenstone digital library project that has been guided and informed by hands-on use in the field by librarians and other digital content managers for over a decade [7].

Figure 4 shows a representative selection of snapshots of the application, taken from the first of our examples detailed below. Using a system of tabs across the top of the application—download, gather, enrich, plugins, create—the librarian controls the documents that are brought into the repository and configures the process for ingesting them. The interface supports the notion of collections, allowing for different workflows for different sets of documents. As we shall see in the worked examples below, there is a wealth of detail, at times subtle, that guides this process. But for now, using large brush-strokes, the interface lets a librarian:

- Bring in content from external sources through web mirroring or over protocols such as OAI-PMH, Z39.50 or SRW/U;
- Include the desired files in the collection through drag and drop, regardless of their file format;
- Manually assign any metadata (individually or by group) to the files using whatever metadata set they prefer;
- Automatically extract metadata that is either stored within source documents (such as document title, that is explicitly represented in formats such as PDF, HTML,

Word or MP3 files) or automatically derived values such as its file-size or character encoding;

- Control how the documents are processed (such as grouping files in the same folder together if they represent the same logical document); and
- Ingest then into the Fedora repository.

3. EXAMPLES

To illustrate the approach we provide two representative examples. In the first, the collection builder is working with Word and PDF files of authored papers. Sometimes they have just the Word document, sometimes just the PDF, and from time to time both. In the second example, the librarian is working with bibliographic data exported from an online catalog related to their specialist map collection and combining it with the recently digitized artifacts.

3.1 A scholar repository of indigenous oral history

Our first example is based around a scholarly repository for indigenous oral history—in particular manipulating a mixture of Word and PDF document and then in a later iteration (see Section 4) HTML web pages also—although anyone working in the area of institutional repositories will recognise the basic tenets and consequently its general applicability.

In Figure 4(a) the librarian has launched the new interface Fli and is in the process of selecting documents for inclusion in the repository. The *gather* tab provides access to the local file system through a traditional graphical tree hierarchy. Special short-cuts are provided for the user's home directory, other collections in the repository, and the download folder (used by the download tab and explained below). Further, customized shortcuts can be set up at will by right clicking on folders or files. In the case of documents being drawn from other collections within the repository, any existing metadata already assigned to it is transferred with it.

Double clicking on a document in the tree view file hierarchy opens up the document: Fli maintains a list of applications to use associated by filename extension. Filenames at times can be rather cryptic (even when created by oneself!) and this feature allows a user a convenient way to clarify the what the content is.

In Figure 4(b) the librarian has progressed to the *enrich* tab where they can manually assign the documents with metadata, in a fielded form and multiple values for the same field can be entered. This can be done on a per file basis, or recursively by clicking on a folder higher up the file hierarchy. Source document have often undergone some initial organisation prior to inclusion into a digital library, and these two choices for metadata assignment work well in tandem. Unqualified Dublin Core is the default provided; other metadata sets can be switched in and out as needed.

In our example, all the articles from a particular journal have been stored in a single folder: clicking on the folder is an ideal level to set the journal title to the articles; clicking on the individual files, the appropriate level for labelling the title for each document. Clearly this model can be extrapolated in a larger example to a file hierarchy that represents the year and volume number of the journal edition. Furthermore, if additional articles are brought across into these folders at a later date they automatically inherit the folder-level metadata values.

In the enrich tab, the same double-click open document feature to access content is often used, this time to access metadata level information such as title, and authors, which they can then copy and paste into the metadata fields in Fli. The lower right area to this tabbed panel presents a history of values used for that particular metadata field, useful for example, with a set of documents where authors names recur.

New folders can be introduced into the file hierarchy of collected documents, and files moved around within this. It is also possible to rename files by right-clicking on them (chosen from the context menu that appears). We have found these abilities useful as experience in the field has shown us a clearer structure for organising source documents often emerges as the collection is being developed.

In Figure 4(c) the user has moved on to the *plugin* tab, and is reviewing the list of modules that will convert the source documents into FedoraMETS for ingest into the repository. A representative sample of formats processed is PDF, Word, PowerPoint, Excel, OpenOffice, HTML, MARC, MP3, QuickTime, DSpace, OAI, and CONTENTdm. At the time of writing, there were over 30 in all.

Document processing plugins are a powerful aspect to the design. It is the cornerstone to our solution of minimizing programming requirements for ingesting documents. It is not enough, however, for them to embody automated rules for processing various documents, for needs often change depending on the collection being developed and more particularly such control needs to be exerted from the user interface. For this reasons, plugins are designed to accept options that control their processing functionality; moreover, these plugins are programmatically self-describing, yielding XML messages that provide human-readable explanations as to what the option controls, along with machine-readable syntactic datatype information [8].

In Figure 4(c), having seen that PDFPlug is already in the list, the user has gone one step further and selected this plugin and pressed the "Configure plugin ..." button. The resulting popup up is dynamically configured and lists all the options the plugin supports, using whatever language the interface is currently displaying. The interface language is set through File→Preferences. Fli is currently available, fully translated in five languages (Arabic, Chinese, French, Spanish, and Russian); Eight others are at various stages of completion, utilizing a web-based management system. See [2] for more details.

Hovering over an option name brings up a tool-tip that explains what the option does and its current default value. Ticking the option's check-box activates it and makes the accompanying fielded information editable: for an option that takes a fixed list of values, a pull-down menu with those items in it is presented; for an option that requires an integer number, the default value is display in a text box and up and down arrows allow it to be change (optionally constrained to a minimum and maximum value); and so on.

Using default plugin values each Word and PDF document will enter the repository as its own digital object. For our example, the librarian wishes to take account of the situation where both a Word and PDF version is available. Using the convention of giving each version the same filename but different extension, in the popup window that appears as a

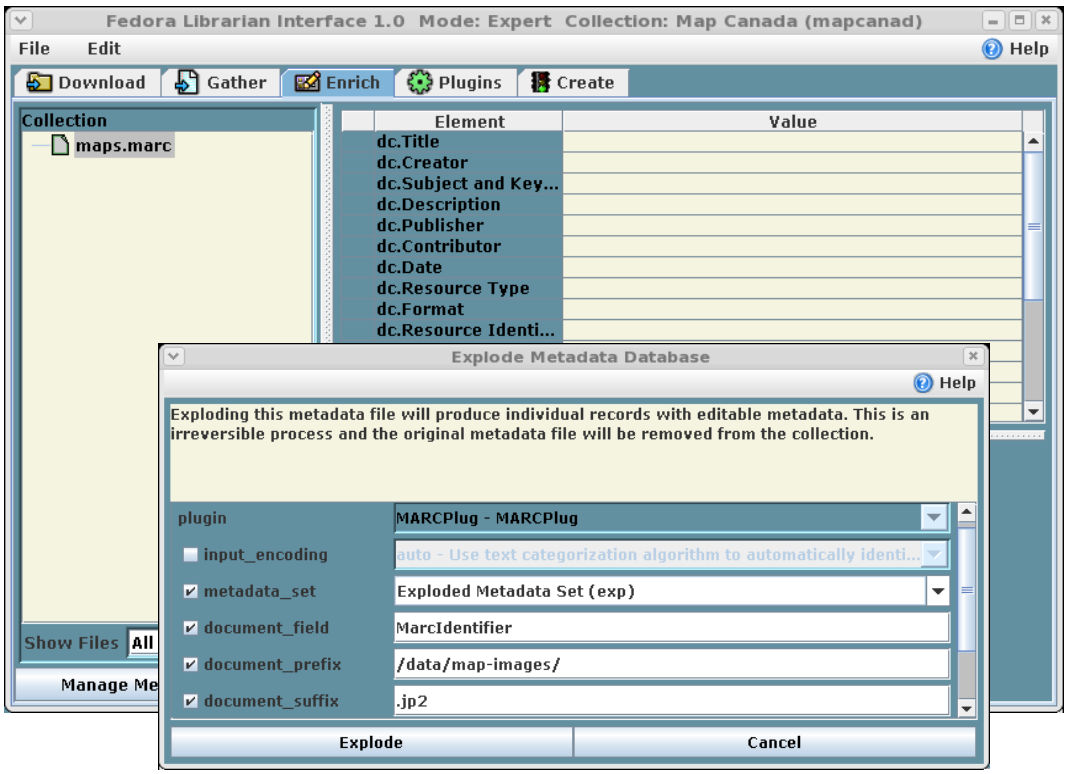


Figure 5: Ingesting documents into Fedora using the Librarian Interface.

result of clicking "Configure plugin ..." for PDFPlug, the librarian has activated the option *associate_ext* and entered "doc". This means that any file with the same root filename, but ends in ".doc" will now be bound into the same digital object in Fedora. If there is no PDF version, then the Word document will form its own object as before. In the event source documents do not follow this convention, the right-click rename feature mentioned above can assist is massaging the data into an appropriate form. Alternatively, a more powerful—but potentially harder to master—option *associate_tail_re* exists that allows a regular expression to be constructed that guides the document combining process.

Satisfied with the selection of source documents, augmentation of metadata and customisation of the document plugin ingest process, the librarian now clicks on the rightmost tab, *create*. The principle activity in this panel is to click on the "build" button and observe the processing instructions to confirm the procedure for ingest is performing as expected. A cancel button is available if not. As a result of ingesting the documents, some metadata is automatically extracted or derived. This metadata is also displayed in the enrich tab, but in a read-only format.

Even in this panel there are options that allow for control, again retrieved from the under-lying implementation through self-describing options that are programmatically embedded into the software. Should the underlying commands ever be upgraded, the interface will automatically update itself and display the new features appropriately in the interface. One example of exerting control is utilising the "maxdocs" option, which controls how many documents are processed. Off by default, this is useful in the prototyping phase of a repository: it allows for a faster turn-around, ingesting a smaller number of documents which are then checked in the repository using a web browser. Any imperfection noted can be adjusted for using Fli, and then the collection rebuilt.

3.2 Combining bibliographic data and source

For our second example we demonstrate how metadata and source documents can be combined. Our motivating scenario is a library with a specialized collection of maps, in this case historic maps of Canada.[5] Bibliographic data is exported from the library's catalog in MARC format. This includes field 024, which stores identifier information for the source documents (the maps in this case). The identifier information is encapsulated in the file naming convention used for the digitized maps. They are placed in a folder *map-images* located in the top level *data* folder, on the file server.

When the librarian drags the MARC files across into the collection (*gather* panel), Fli pops up a window alerting the user to the fact that there is currently no plugin that is capable of processing this file. It suggests MARCPlug as potentially suitable, and ask if the user would like it to be added in. In the event more than one plugin registers the ability to process it, the popup window provides the list of possible choices, from which the user can choose to select one, or alternatively ignore all of them if desired. Plugins can also be added, and their order of priority changed if needed, at a later time through the *plugin* tab.

The next step is to "explode" the MARC metadata. This

[5]The example was artificially constructed, for pedagogical purposes, from an excerpt of the American Memory's Map Collection.

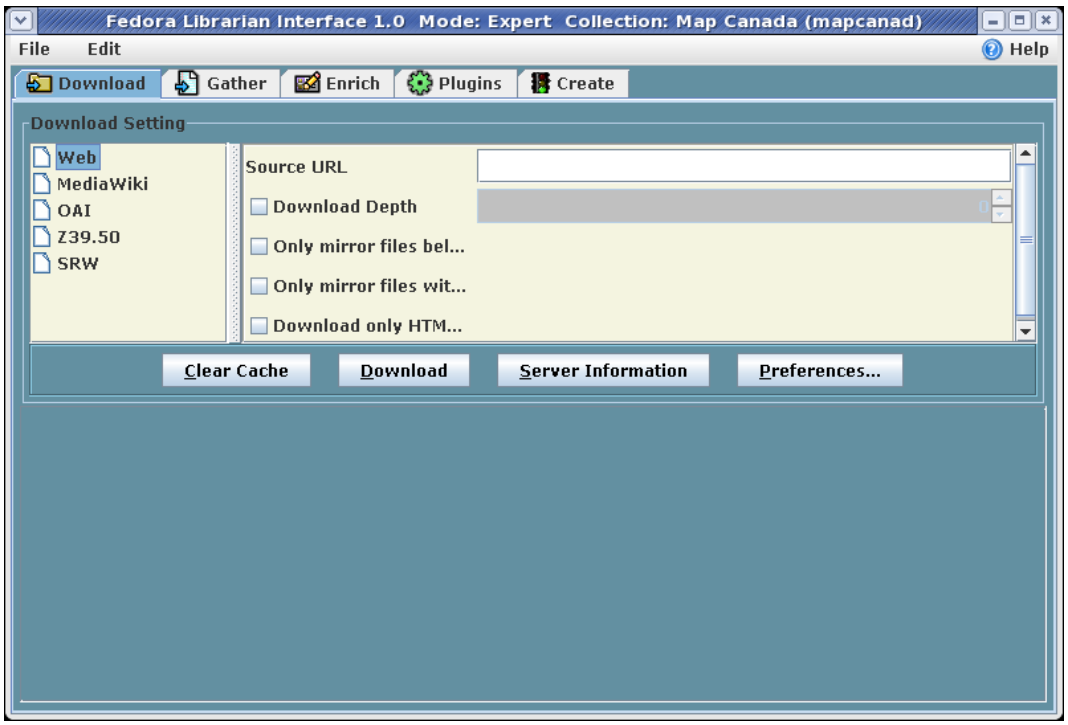


Figure 6: The download panel to FLI.

is a term used in the interface to mean decompose a single file, which may contain many records, into its constituent parts. Right-clicking on the MARC file in the enrich panel brings up the context menu that includes this additional feature. Selecting this results in the popup window featured in Figure 5. Various options are available (again deduced from FLI requesting machine-readable syntactic data-structure information and tool-tips). In our example the user has entered "MarcIdentifier" as the *document_field* (the field in the record that identifies the source image), set the *prefix* as */data/map-images/* where the files are located, and added the suffix ".jp2" as the maps are stored in the JPEG2000 format with this filename extension. Exploding the MARC file with these options and then building (ingesting) the collection yields a set of digital objects that express the metadata stored in each MARC record converted to English readable names through a mapping file (e.g. 024 maps to MarcIdentifier). In this case the default mapping file is used, however the user is free to devise their own, and specify it through the same popup window (the option is further down the page in the popup window and so not visible in the figure). In the event a matching source document is found, it is bound in to the digital object as a datastream. If no match is found, the digital object is still formed, however it contains only metadata.

The example documents used in Figure 1, which we looked at at the start of the article to show the default Fedora interface, are in fact the result of ingesting the map-based documents for this example. Plugins can optionally specify the content model they map to. In this case ImagePlug connects the digital objects it generates to Fedora's UVA_STD_IMAGE content model.

The MARC records were exported externally in this example; another option is to use the download panel, shown in Figure 6, that can retrieve metadata over protocols such as OAI-PMH, Z39.50, and SRW, and source documents through the web mirroring of sites in general, with a specialization for articles posted on wikis that use MediaWiki, of which Wikipedia is the best known example.

The librarian is not required to include a *document_field* during the exploding process. In this case individual files are produced for each record. They can still be combined with source documents post "explosion", even if there is not a systematic naming convention between the two: in the context menu brought up by right clicking on an item is the option "replace" through which the librarian can navigate on the file system to the source document for that metadata record. Alternatively the MARC file need not be exploded, in which case it the Fedora repository is populated with digital object that store the metadata only.

4. DISCUSSION

Returning to our first example, if we were to extend it to include web pages as well as PDF and Word documents, then, after including HTMLPlug (which would naturally be prompted by the application when the first HTML document was dragged across into the collection), the principle adjustments would be based around changing the settings to *associate_ext*. The option—which is located in BasPlug (a base plugin that all other plugins inherit from), and consequently available to all plugins—can take a list of file name extensions, separated by commas. Setting this option in HTML plug to "pdf,doc" and leaving the option in PDFPlug at "doc" would be one possible configuration that would achieve the right result: priority would be given to the HTML documents first, then PDF, then Word if present.

Anyone who has had to write code that parses HTML

syntax will know how messy it can be, given its evolution and and the laxity with which it is sometimes generated. HTMLPlug has a substantial list of options to help address these foibles, but regretfully due to the nature of the web will never be guaranteed to deal with all circumstances correctly. By comparison, while ingesting HTML through the Fedora administration tool might be tedious, one advantage of the more labour intensive setup is that all the peculiarities of the web pages, on a per document basis, can be studied and the appropriate action taken.

To undertake the second example with the Fedora administration tool alone is not feasible. The interface assumes a file represents either a single document or document-subcomponent (i.e. datastream). The notion of multiple items contained within one file is not well catered for, triggering the need for some pre-processing software that breaks up the files first. This would be an example of the command-line/GUI hybrid category we itemized in Section 2. With command-line driven pre-processing a requirement, it would be straightforward to include some additional logic—not dissimilar to that in Fli—that automatically combines the metadata in the records with source documents, perhaps leveraging off the work already implemented in the third party utility FABULOUS.

5. CONCLUSION

In conclusion, we have demonstrated, through two running examples, a variety of techniques a new interface for Fedora for librarians is capable of. These, however, should be seen as indicative as to the style of possible interaction rather than providing a full account of Fli's capability. The application embodies many years of HCI design and development work guided by working with practitioners in the field. For instance, there are literally hundreds of plugin options. Not only are these well documented in Greenstone self-help tutorials that have been developed over several years (and have also been translated into numerous other languages) they are equally applicable to Fli as the new interface shares the same document processing architecture.

In giving workshops centered around this type of digital library collection building application, we noticed that the sheer number of options can at times be overwhelming to users. Consequently we introduced a user mode which helps limit the number of options shown, given a particular mode. For instance, unless in expert mode, options that require a regular expression to be entered are suppressed.[6] We did not take this decision about introducing modes lightly, as there is evidence in the HCI literature that including modes can often be detrimental in a user interface, for instance when instruction in the documentation make no sense to the user because that part of the interface is not visible in the mode they are in; however in weighing up the pros and cons, in this case we decided having the mode was overall beneficial, and we took care to embed elements in the interface that act as hooks to lead the user to changing modes when options do not appear to be available.

The interface includes a comprehensive array of alerts and other general popups. These all come with a "don't show me this again" tick box in the popup window, and can additionally be controlled through the File→Preferences menu, as can proxying from behind a firewall with or without user authentication.

A companion application for creating new metadata sets and editing existing ones is bundled Fli. The ubiquitous Dublin Core (both qualified and unqualified) is shipped as a matter of course along with standards such as RFC1807 and the Australian and New Zealand Government Locator Service metadata sets (agls and nzgls respectively). As noted earlier, multiple metadata sets can be active with the application simultaneously, and the tool is also amenable to developing your own metadata sets for internal use.

Fli is available through the download page of the Greenstone web site, *www.greenstone.org*.

[6]For instance the expression $(?i)\backslash.(wav|aiff)\$$ used to match files that end the filename extension "wav" or "aiff" regardless of the letter-case used can be rather daunting.

6. REFERENCES

[1] P. Achananuparp and R. B. Allen. Developing a student-friendly repository for teaching principles of repository management. In *DigCCurr: International Symposium in Digital Curation*, Chapel Hill, NC, US, 2007.

[2] D. Bainbridge, K. D. Edgar, J. R. McPherson, and I. H. Witten. Managing change in a digital library system with many interface languages. In T. Koch and I. Sølvberg, editors, *Research and Advanced Technology for Digital Libraries, 7th European Conference (ECDL)*, volume 2769 of *Lecture Notes in Computer Science*, pages 350–361, Trondheim, Norway, 2003. Springer.

[3] R. Cover. Muradora GUI for Fedora repository uses SAML and XACML for federated identity. *Cover Pages*, October 2007. http://xml.coverpages.org/ni2007-10-26-a.html.

[4] C. G. Kortekaas. Making Fedora easier to implement with Fez. *Open Repositories*, January 2007. http://espace.library.uq.edu.au/view.php?pid=UQ:11924.

[5] C. Lagoze, S. Payette, E. Shin, and C. Wilper. Fedora: an architecture for complex objects and their relationships. *International Journal on Digital Libraries*, 6(2):124–138, April 2006.

[6] S. Payette and C. Lagoze. Flexible and extensible digital object and repository architecture (FEDORA). In *ECDL '98: Proceedings of the Second European Conference on Research and Advanced Technology for Digital Libraries*, volume 1513/1998, pages 41–59, Berlin, 1998. Springer.

[7] I. H. Witten and D. Bainbridge. A retrospective look at Greenstone: lessons from the first decade. In *JCDL '07: Proceedings of the 2007 conference on Digital libraries*, pages 147–156, New York, NY, USA, 2007. ACM Press.

[8] I. H. Witten, D. Bainbridge, G. W. Paynter, and S. Boddie. The Greenstone plugin architecture. In *Proceedings of the second ACM/IEEE-CS joint conference on Digital libraries*, pages 285–286. ACM Press, 2002.