An Interactive User Management System for Multilingual Documents:

A Case Study of the Pei Jones Collection

A thesis

submitted in partial fulfilment

of the requirements for the degree

of

Master of Engineering

Computer Science

at

The University of Waikato

by

PAPITHA CADER

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

2012

Acknowledgements

People often wonder if they can find a supervisor with an extensive depth of knowledge and excellent communication skills while being transparent, enthusiastic and encouraging. I'm lucky to have found such a professor at Waikato University — Dr. David Bainbridge.

I take this opportunity to express my gratitude to the people who have been instrumental in the successful completion of this thesis.

First and foremost, I offer my sincere gratitude to my supervisor, Dr. David Bainbridge, who has supported me throughout my thesis. I attribute the level of my Master's degree to his encouragement and effort and without him this thesis would not have been completed or written. One simply could not wish for a better and friendlier supervisor.

I would like to thank the Ngā Pae o te Māramatanga (Centre of Research Excellence) for providing me with the scholarship which was a valuable support during this period.

I would also like to thank Dr. Hemi Whaanga for providing an expert view on my project.

A special thanks to Pat (Dharmendra Patwardhan) for being my source of inspiration.

I would like to thank my husband (Vivek) for supporting and encouraging me to pursue my dreams. I am truly indebted and thankful to my family for providing me moral support throughout my journey.

My sincere thanks go out to Sam McIntosh who has supported me a lot throughout the year.

Thank you God, for answering my prayers and giving me the strength to pursue my dream despite everyone thinking it was too late. Thanks to my daughter Tanya who has prayed for me all through the year and made me give it my best.

Table of Contents

Table of Figures

Chapter 1: Introduction

1.1. Introduction

The management of a digital library (DL) system and the end-user experience are often seen as two separate roles, and consequently implemented as two different software components. This can lead to unintentional inefficiencies in the maintenance of the software. The main alternative — which treats the two roles more uniformly — is to deliver the management interface through the web browser in addition to the end-user interface. In DL systems that adopt this approach, however, the web interface delivers a rigid workflow, tailored for a specific purpose, such as metadata assignment and document ingest for an institutional repository. While this provides a more efficient and streamlined system in terms of software maintenance, the overall functionality is more limited than with the former approach.

The hypothesis to this thesis is, can general purpose DL software architecture be designed to provide a flexible environment that supports both the management and end-user roles through a single unified web interface? In particular we take the collected

correspondence of the Māori scholar Dr. Pei Jones and undertake case study of how this approach can be applied in a practical way.

As inspiration, we take blogging software as an exemplar of a social media driven approach to content creation and curation that utilizes a single web interface. A traditional DL is designed in a manner where the end-user has read-only access to the repository (Figure 1). The Librarian or Curator uses a management console which is usually installed locally and the end-user is presented with a web interface to search and view the documents. This restricted access — in contrast with blogging software — prevents them from contributing or correcting content.

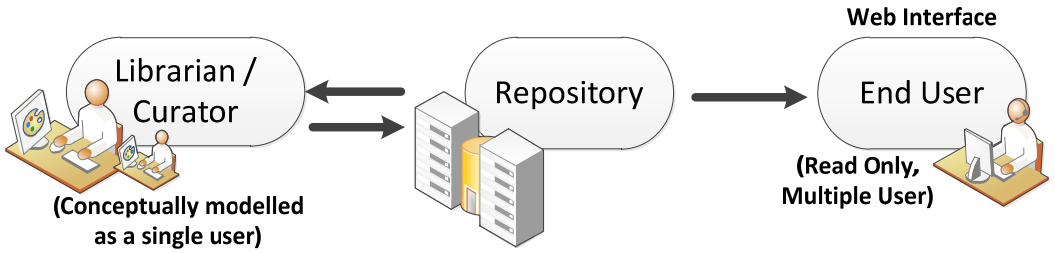


Figure 1: General Purpose Digital Library

Institutional repositories are characterised by fixed web interface workflows (Figure 2). Here the Librarian or Curator is again often conceptually modelled as a single user. However, this time end-users have access to contribute, making it a bidirectional relationship with the content stored in the repository.

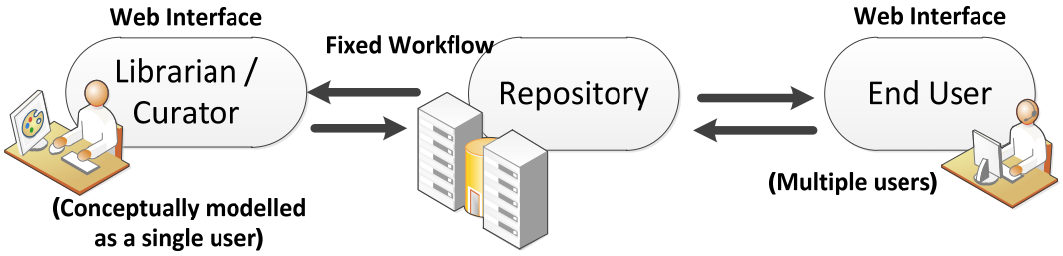


Figure 2: Institutional Repository

By combining a general purpose DL architecture with the web-interface approach of an institutional repository architecture — tempered by ideas of social media such as blogging software — this work seeks to derive a cohesive web based interface (Figure 3) for multiple users and librarians to access and share information simultaneously. This improves the flexibility of the various software components and provides a streamlined workflow.

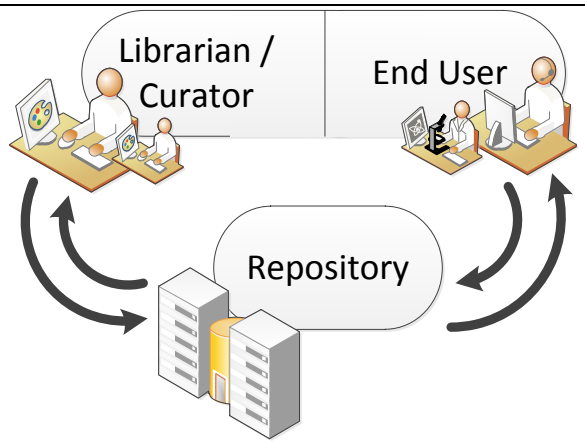


Figure 3: Proposed Flexible Web Based Digital Library

1.2. Requirements

A detailed consideration of the user requirements led to the development of a customized framework which was built to include the following features:

(i) *Batch-driven ingest*
(ii) *Web-browser activated ingest*
(iii) *Born-digital document creation*
(iv) *Document delete*
(v) *Document merge*
(vi) *Document metadata editing*
(vii) *Document structure editing*
(viii) *Workflow control of document creation and editing*

Batch-driven ingest – This is the process of ingesting typically large volumes of digital content and its corresponding metadata instigated from the command-line. Such a capability ensures a consistent structure is rapidly formed (repeatable on demand) across the collection and reduces the manual process which would otherwise be time consuming and labour intensive.

Web-browser activated ingest – This is the process of ingesting documents using the web-browser interface and enables the authorized user to have full control of the documents. Changes,

enhancements and corrections to individual documents can be made quickly.

Born-digital document creation – This is the process of creating and managing direct-to-digital content with a systematic method of maintaining this information. The increasing use of paperless content requires the continuous digital preservation, storage and archival of this content and the visibility of this information by searching and viewing from different platforms.

Document delete – The ability to remove a document in a collection directly from the web interface. This would enable the management of the collection from a unified front-end. For example, incorrect or older versions of a document have to be removed from the collection to keep it up-to-date. Thus, providing users the option to delete documents from a browser improves the ease of use and saves time.

Document merge – The flexibility in managing the collection by allowing the users to merge different sections and sub-sections of the document, allowing the content to be organized in a seamless manner. This feature will enable the assembly of documents in a structured manner and changes to this structure can be done easily by using a web interface. The merge action can be performed using drag and drop mouse actions.

Document metadata editing – This enhances the system by providing the ability to edit metadata. The system should also automatically populate the metadata for selected fields. For example, any inconsistencies in the collection can be easily identified by viewing the metadata elements grouped together from the web browser.

Document structure editing – This provides additional functionality to the system by enabling the structure of the document to be edited from the web browser. For example, by viewing and manipulating the structure of the document in a controlled manner provides a simplified process of editing the structure while maintaining the rules defined by the collection.

Workflow control of document creation and editing – This provides a system workflow based on a shared concept to actively contribute to the content and provide a mechanism for actioning on these requests. This framework would enable the creation, storage and distribution of collective data by implementing a workflow control.

The general aim of this work is to develop a digital library software infrastructure that meets these requirements. More specifically, a case study is undertaken based on the digitized correspondence of Dr. Pei Te Hurinui Jones, which comprise of manuscripts, photos and documents. This is done to test the design

of the software infrastructure in a practical setting. More details about this scholar and his prominence in Māoridom are presented in Chapter 4.

1.3. Approach

As remarked in the introduction, general purpose digital library software solutions implement the management of content through specially developed sub-systems. These sub-systems are typically separate from the implementation of the main end-user interface, which has the unfortunate side-effect of duplicating much functionality.

The premise for this project was to see if editable metadata and annotations embodied in the software architecture approach shown in Figure 3 used in combination with the searching and browsing capabilities normally associated with the end-user interface can be used internally by digital libraries, archivists and curators to manage and enrich content.

1.4. Synopsis

The structure of this thesis is as follows. In Chapter 2 we review widely used open source digital library and content management

software: DSpace,[1] EPrints,[2] Fedora,[3] Greenstone,[4] SilverStripe[5] and review their strengths and weaknesses in terms of the requirements of this project, leading to the decision to base the implementation work on Greenstone. In Chapter 3 we review the design and implementation with a detailed study of the Greenstone architecture. Chapter 4 explains the stages of developing the Pei Jones collection. Chapter 5 provides a summary of the evaluation and librarians' specific view on the developed Pei Jones collection. Chapter 6 provides the conclusions and ideas for future work.

[1] http://www.dspace.org/
[2] http://www.eprints.org/
[3] http://fedora-commons.org/
[4] http://www.greenstone.org/
[5] http://www.silverstripe.com/

Chapter 2: Background

In this chapter, we discuss the technology behind a representative selection of digital libraries software architectures. We elaborate on the studies comparing the common features and differences between these systems to help identify how well each aligns with the requirements of this project.

2.1. Walkthrough of a DSpace Digital Library

DSpace[6] is an open source software package that was designed to manage the published outputs of institutions and organizations. It was designed to collect and preserve different kinds of digital materials such as multimedia, videos and texts, with the data being arranged as groups of community collections. It was collaboratively developed by MIT and HP and was first released in 2002. DSpace has been used widely by universities and libraries.

Figure 4 shows the home page to one such DSpace repository.[7] On the left, there are four options for end-users to browse for information like issue date, author, title and subject in addition to the communities and collections page.

[6] https://wiki.duraspace.org/display/DSPACE/EndUserFaq
[7] http://demo.dspace.org

The document information page usually contains several metadata items to describe the resource and its contents.

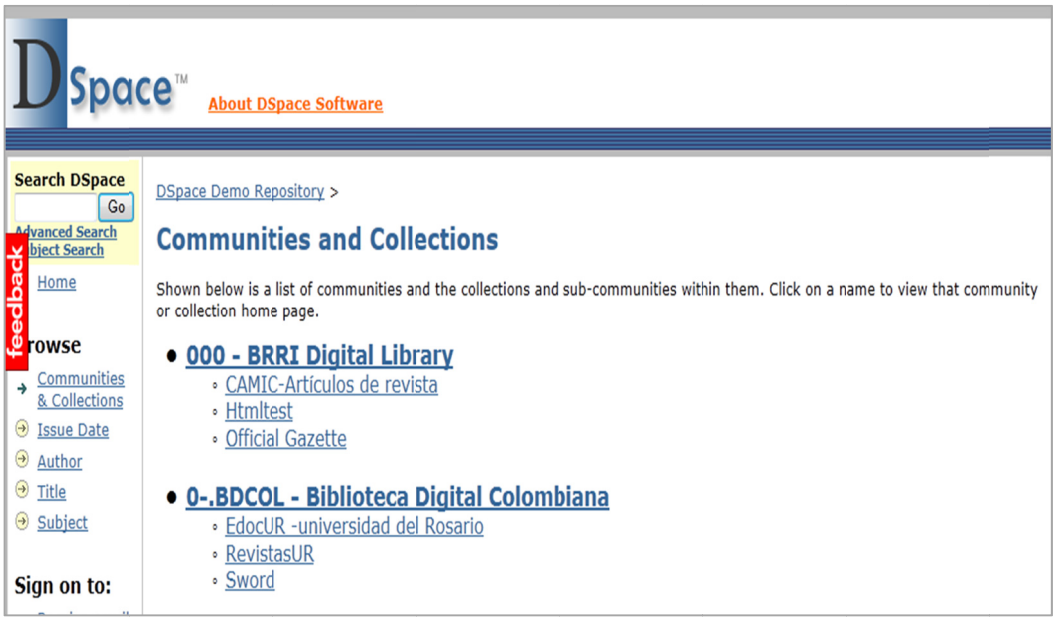


Figure 4: DSpace demonstration website

DSpace is aimed at the academic market and is positioned as a tool for electronic self-publication and storage of intellectual output from universities with capabilities for handling a wide range of digital works. DSpace is designed to capture digital content, store information securely, maintain indexes, preserve the information and enable redistribution of digital material in various formats. The structure of an example metadata record[8] within DSpace is shown in Figure 5.

[8] http://demo.dspace.org/xmlui/handle/10673/6177

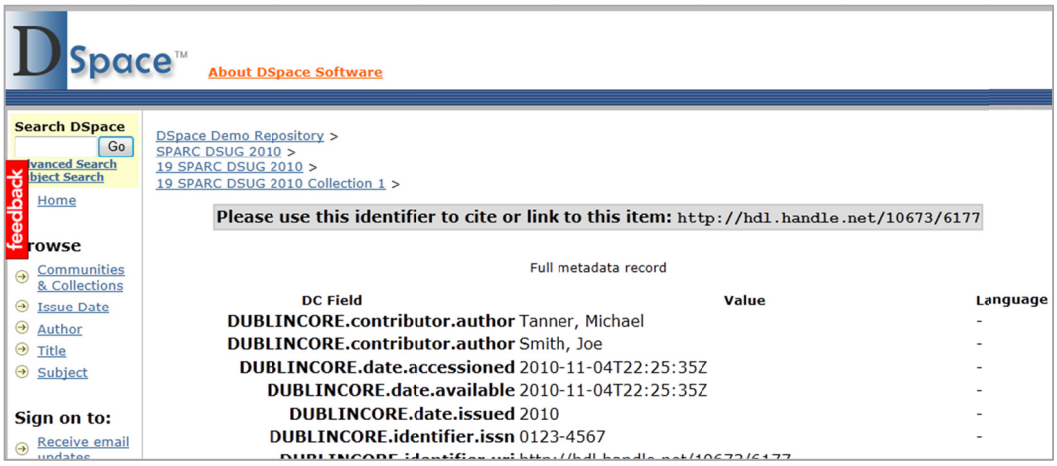


Figure 5: A Sample metadata record in DSpace

The widespread availability and usage of digital content over the past decade has meant that the storage, management and distribution of this information for research purposes has become a time consuming and labour intensive task. The burden of managing this has become increasingly tougher for the educational institutions and research centres to do on their own. DSpace is intended to provide a basic yet fully functional system to support these needs. This has led to the development of a system founded on support for the functions that a research organization needs while still preserving its simplicity. Even though DSpace emerged from a research background it has also been used in production and project environments. Figure 6 illustrates a sample workflow[9] of DSpace DL. The workflow receives a new submission and the user is prompted to provide a detailed description of the item. The user can then upload the related file which is then verified and added

[9] http://libraries.mit.edu/dspace-mit/build/index.html

into the collection by the curator of the collection. The user can then view the summary and details of the collection.

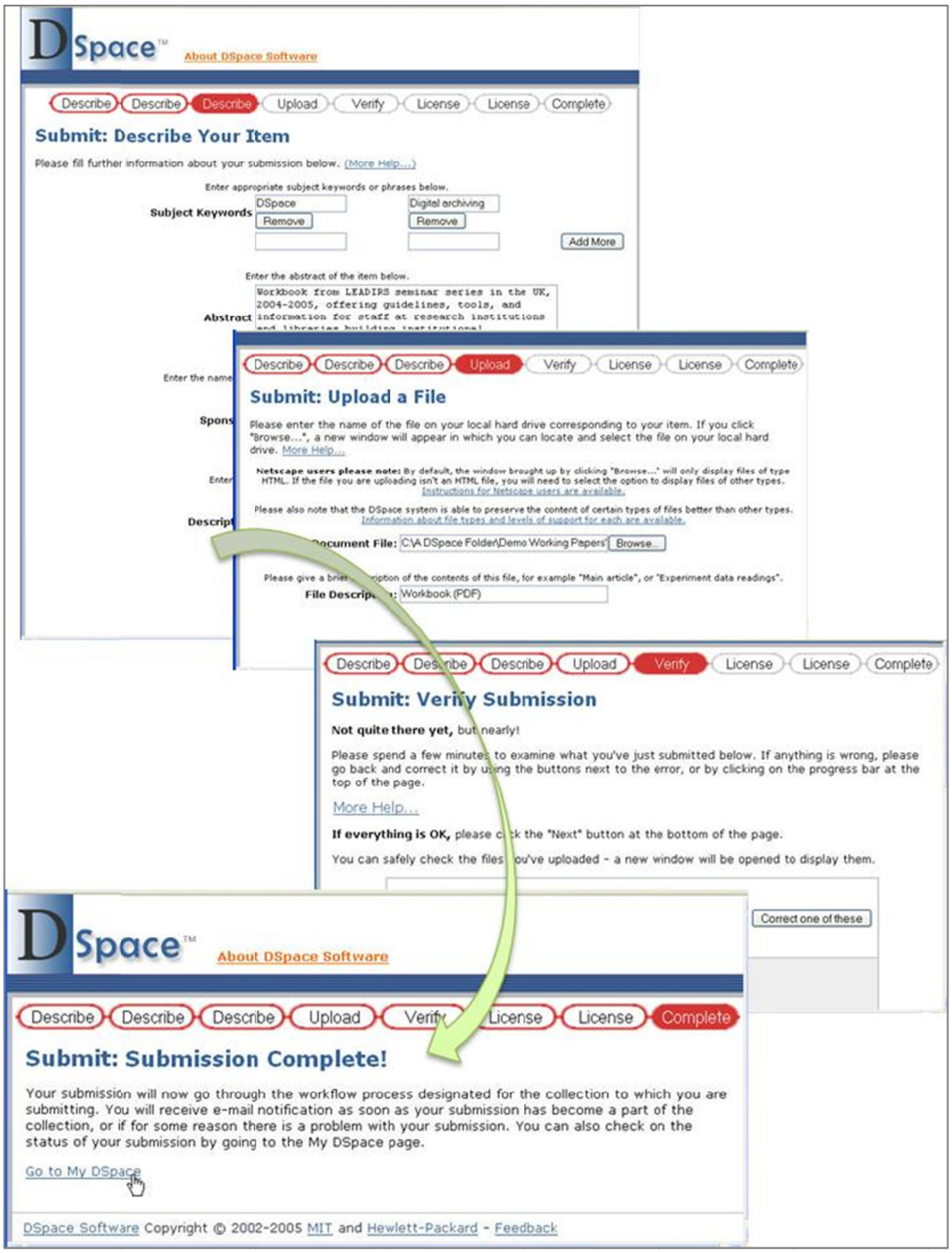


Figure 6: Workflow of DSpace DL

The main roles of DSpace are:

- The ability to seamlessly import digital content into the repository with metadata information.
- Enable centralized access to this content via browse lists, metadata based searching and in the case of HTML and PDF access through full text searching.
- Ensure that all the digital assets are securely preserved in the data repository

DSpace uses the Dublin Core standard [1] to store metadata and exchange information with other systems. DSpace uses a fixed metadata standard across all its collections.

2.2. Walkthrough of an EPrints Digital Library

The EPrints[10] digital library software is an open source application designed to manage digital repositories and (like DSpace) is mainly used for institutional repositories. EPrints was first released in 2000 and is used as a free open access application. EPrints is designed to preserve literature, scientific data and theses. The structure of EPrints encourages content creators to digitize their work and build digital repositories rather than using physical ones. This digital library software was developed at the School of

[10] http://www.eprints.org

Electronics and Computer Science, University of Southampton, UK. Figure 7 shows the EPrints repository where the user can browse for information by year, subject, division and author.

Figure 7: EPrints demonstration website

Documents can be uploaded to the EPrints repository using a web interface. Once the repository becomes a data provider, its digital materials can be shared on the Internet over Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH). [11] It supports any kind of metadata schema and allows users to define the hierarchical structures for searching and viewing documents. EPrints can store any kind of digital material. EPrints was developed as a Web based and command line based application written in Perl.

[11] http://wiki.eprints.org/w/Synchronize_your_repository_via_OAI-PMH

EPrints uses its own internal format to store the metadata which is different from most other DL systems. EPrints can store multiple archives within a single instance and also supports multiple DLs running on a single instance.

A demonstration of the application[12] from the EPrints website was used for comparing the feature sets. A sample workflow of the EPrints system is shown in Figure 8. The workflow consists of five stages. The first stage is to select the type of document, e.g. article, book, thesis etc. The document is then uploaded followed by the details about the document. The document is then categorised based on subject and finally deposited for review.

[12] http://www.eprints.org/software

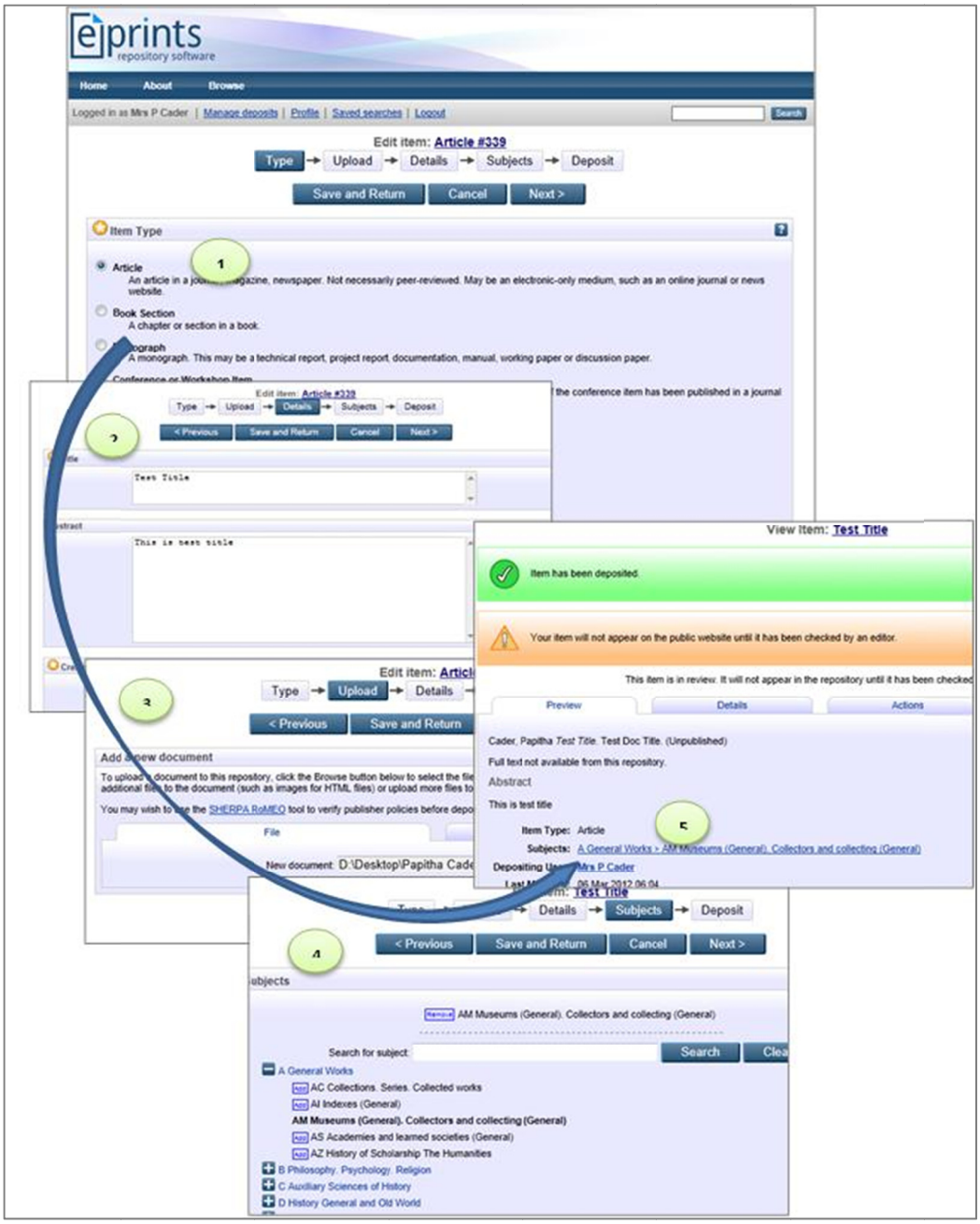


Figure 8: Workflow of EPrints System

One of the distinguishing features[13] of EPrints is its ability to be customized and configured, especially with respect to the subject hierarchy. This hierarchy creates a searchable archive which can

[13] http://wiki.eprints.org/w/EPrints_Manual

then be viewed on any platform. The EPrints workflow uses a powerful web based front end for the submission of digital content. Documents can be uploaded directly or as compressed zip files which are extracted on the server automatically. The content which is to be imported can also be mirrored directly from a website by specifying the source URL and save the custom metadata associated with it. The structure and integrity of the metadata fields is managed by supervisors who then tag this to the author's record. All documents and papers submitted through this process go through a moderation process which is then processed as per the pre-defined workflow. The workflow can also be triggered from the web interface.

2.3. Walkthrough of a Fedora Digital Library

The Fedora[14] digital library (Flexible Extensible Digital Object Repository Architecture) was developed at Cornell University. The primary use of this digital library software tool is for the storage, management and publication of digital content. Fedora was developed based on the Kahn-Wilensky Framework [2]. In this framework, there are defined basic elements and components. The user interface and repositories have been separated in the architecture. The operations for storing and

[14] http://fedora-commons.org/download/2.1/userdocs/tutorials/tutorial1.pdf

accessing digital objects through a Repository Access Protocol (RAP) have been provided in each repository.

Figure 9 shows the browse collection feature of a Fedora repository. The user can browse for information by title or customized categories such as author, department, students etc.

Figure 9: Fedora demonstration website

The Fedora project was developed with a multi-tier service structure[15] as its primary focus and was designed to integrate into an open architecture framework. The various functions of the digital library were segregated into services with custom user interfaces. The core services [3] of the Fedora digital library software are:

[15] http://ecommons.library.cornell.edu

- Repository services
- Index services
- Collection services
- Naming services
- User-interface services

These core services are combined together as per the requirement to form a complete and usable digital library system. All content in the Fedora repository is managed as individual data objects. These data objects are made up of components of data streams which contain the digital content with the related metadata. There is no defined limit to the number of metadata items that can be stored in a data object. These objects are linked to each other in defined relationships which form an interconnected web of meaningful information. Figure 10 shows the administration tool console[16] which can be used to upload, search and retrieve, modify and purge data objects.

[16] http://fedora-commons.org/download/2.0/userdocs/client/admin-gui/index.html

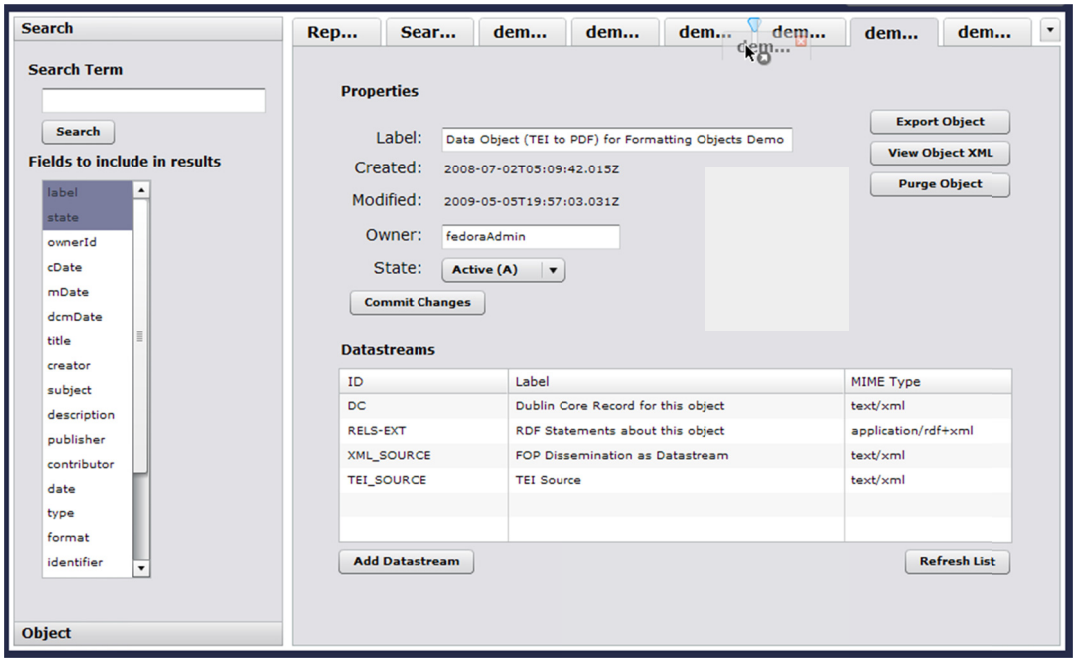


Figure 10: Fedora administration tool

Each data object is represented by an XML file containing information about the physical and virtual location. This ensures that the library consistently remains robust and reliable. All actions performed on the objects are logged systematically providing a complete audit trail of the XML files. At any point in time the repository can be rebuilt and reconstructed to its original form using the XML files and there is a requirement for a specific application or database system to manage this.

The data objects hold the metadata and content with intertwined relationships, the digital objects can be customized to support varied schemes with multiple contexts. These objects can

also exist independently without being dependant on each other or without any particular catalogue.

Figure 11 shows a customized implementation of the Fedora[17] system where the users can search or browse for collections using the left menu. The results and collection details are displayed on the right pane.

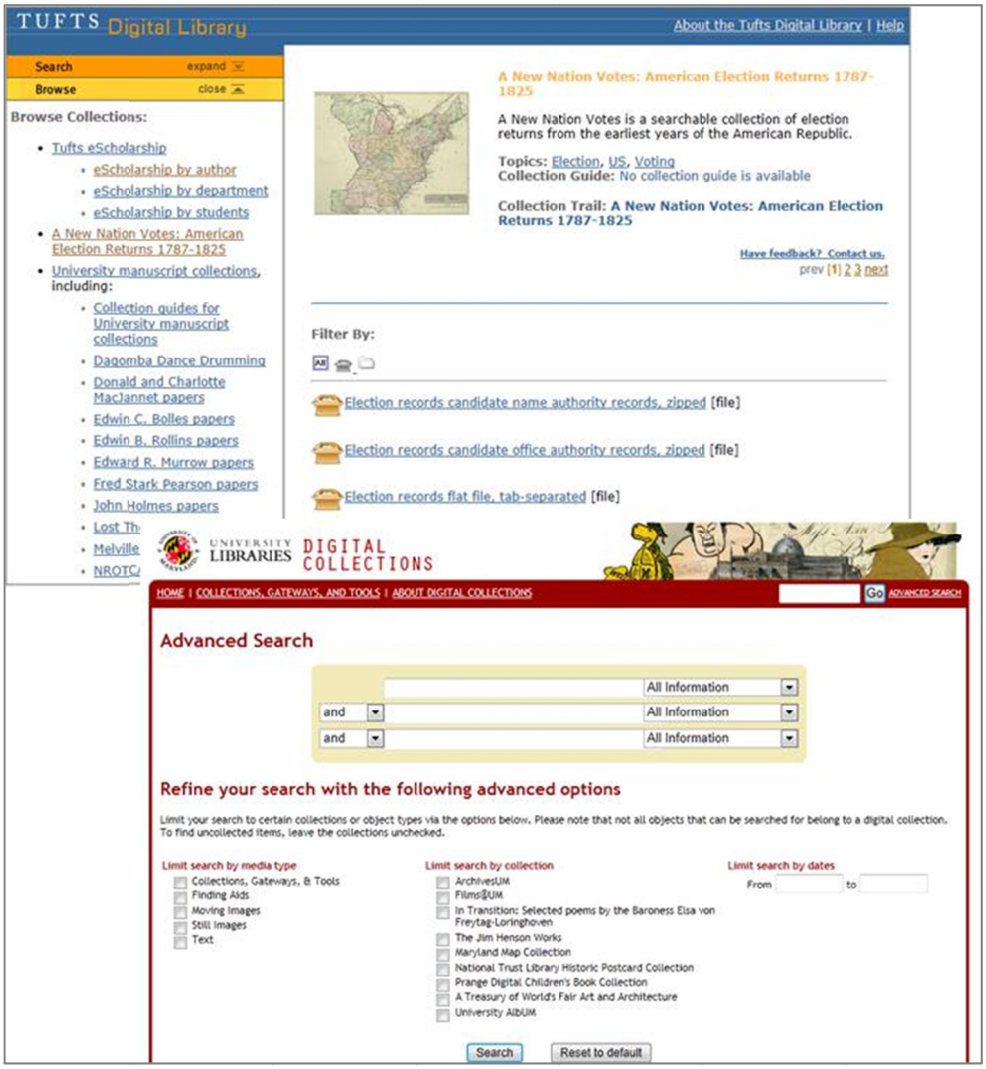


Figure 11: Sample Implementation of Fedora Digital Library

[17] http://dl.tufts.edu/

2.4. Walkthrough of SilverStripe CMS

SilverStripe[18] is an open source web based Content Management System [4]. It is being used by businesses, governments and non-profit organisations. SilverStripe can be used to build internet and intranet based websites as well as web applications with customized design, content workflows, and integration of social functions. SilverStripe is based on the PHP5 programming framework. The flexible framework enables the user to customize the site while still maintaining its security features, workflow and support for creating sub-sites.

SilverStripe uses a template based object oriented architecture and includes an external module to authenticate with external sources outside SilverStripe. The data-objects and the corresponding mapping to the underlying database are hard-coded since the developers have chosen a convention based system rather than a configuration based one by using logical and meaningful placement of files and directories. To manage files and images, SilverStripe recommends using the Data Object Module rather than the out-of-box system.

Figure 12 illustrates some of the features of SilverStripe which provide editing, uploading and blogging capabilities to the application. The screenshots show the web based content editor, file

[18] http://www.silverstripe.org/introduction/

management console and workflow management system. The user can create and upload the content using the content editor. The collections, files and folders can be managed using the file management console. Workflow around comments and approvals is managed using the workflow management system.

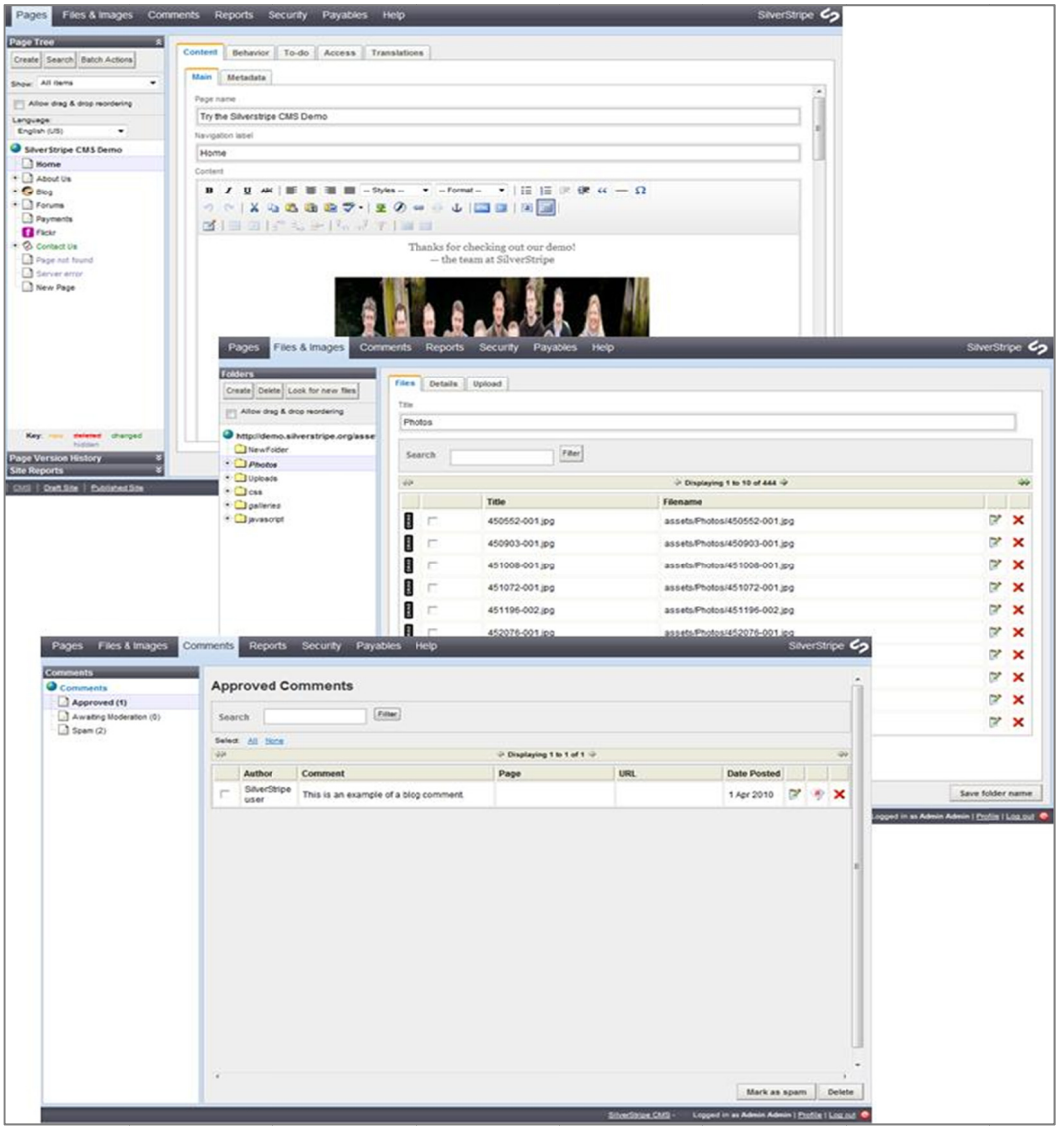


Figure 12: Sample Implementation of SilverStripe

2.5. Walkthrough of a Greenstone Digital Library

The Greenstone digital library is a software suite for building and distributing digital library collections [5]. Apart from being a digital repository, Greenstone is also commonly used as a tool for building customized digital libraries that hold different types of data. One of its key features is the ability to adapt the end-user interface depending on the type of information to be organized and published. All the digital content stored within Greenstone's data store is searchable using full-text indexes. A varied set of digital collections can be created and managed within Greenstone. The administrative functions build into Greenstone enables authenticated users to build, preserve and manage these digital collections.

The Greenstone Librarian Interface (GLI) is a graphical user interface application written in Java. Its primary use is to build and manage digital collections. There are three modes within GLI: Expert mode, Librarian mode and the Librarian Assistant mode, which control the level of detail displayed within the interface.

Figure 13 illustrates the workflow involved in building the MGDemo collection using GLI. There are six basic activities that can be performed using GLI, namely: Download, Gather, Enrich, Design, Create and Format [5]. The download tab is used to download documents from an external source into the collection and

the Gather tab is used to collect documents when creating or updating a collection. Metadata can be added to the collection from the Enrich panel. The full-text indexes and structure of browsing can be managed from the Design tab. When the collection is built the progress and status is displayed in the Create tab. The Format tab is used for formatting the collections appearance.

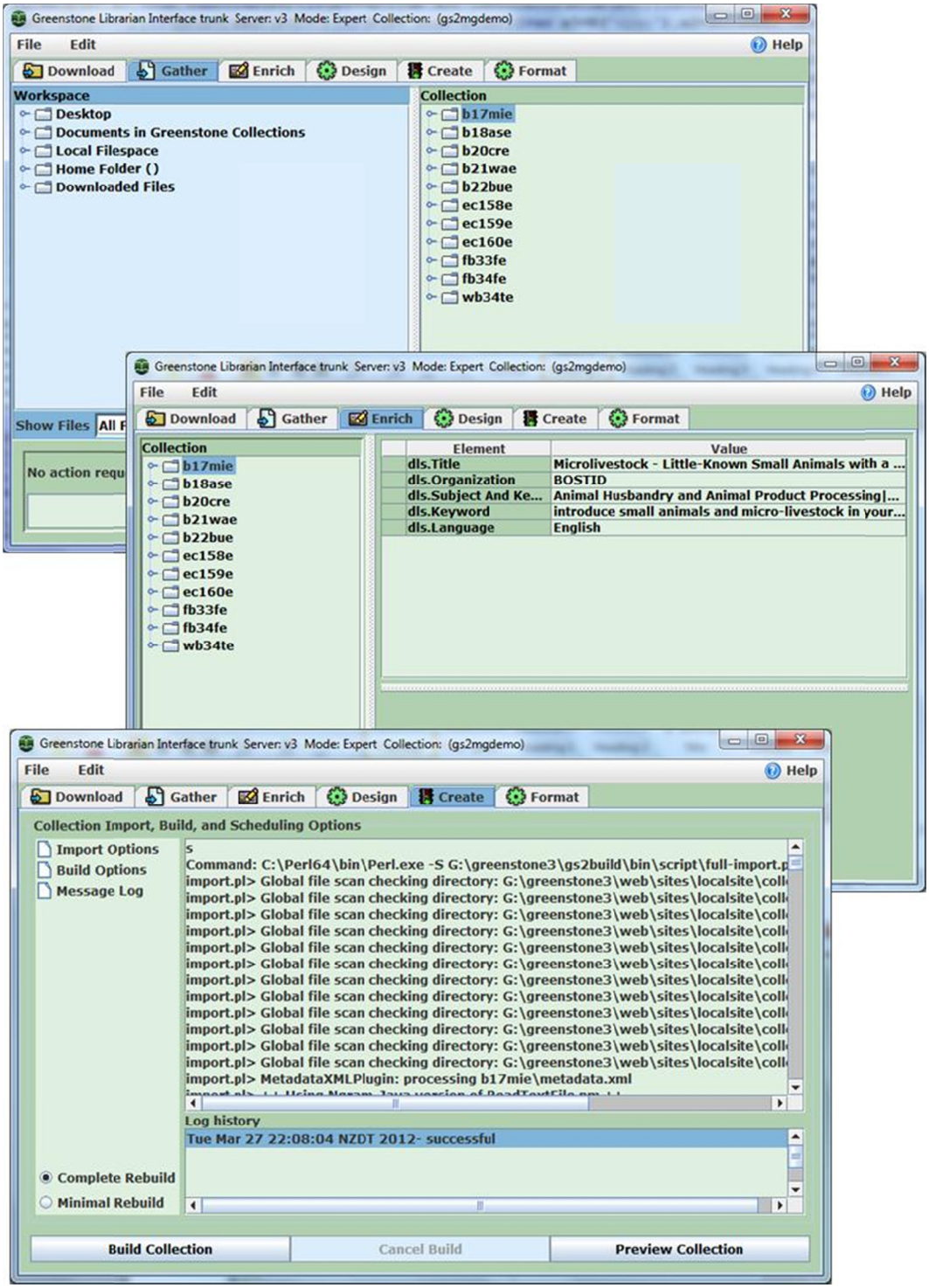


Figure 13: Building a demo collection in Greenstone using the Librarian Interface

The Greenstone digital library software contains a powerful and customizable searching and browsing module. The digital objects are segregated into sections and chapters for indexing and retrieval. When a document is added to the library, its metadata is extracted and processed automatically within the system. Greenstone comes pre-packaged with plugins for most commonly used document formats. Users with programming experience can also add additional plugins to Greenstone. Figure 14 shows the method of browsing collections in Greenstone. The user can customize the browsing options such as browse by title, subject, organisation etc.

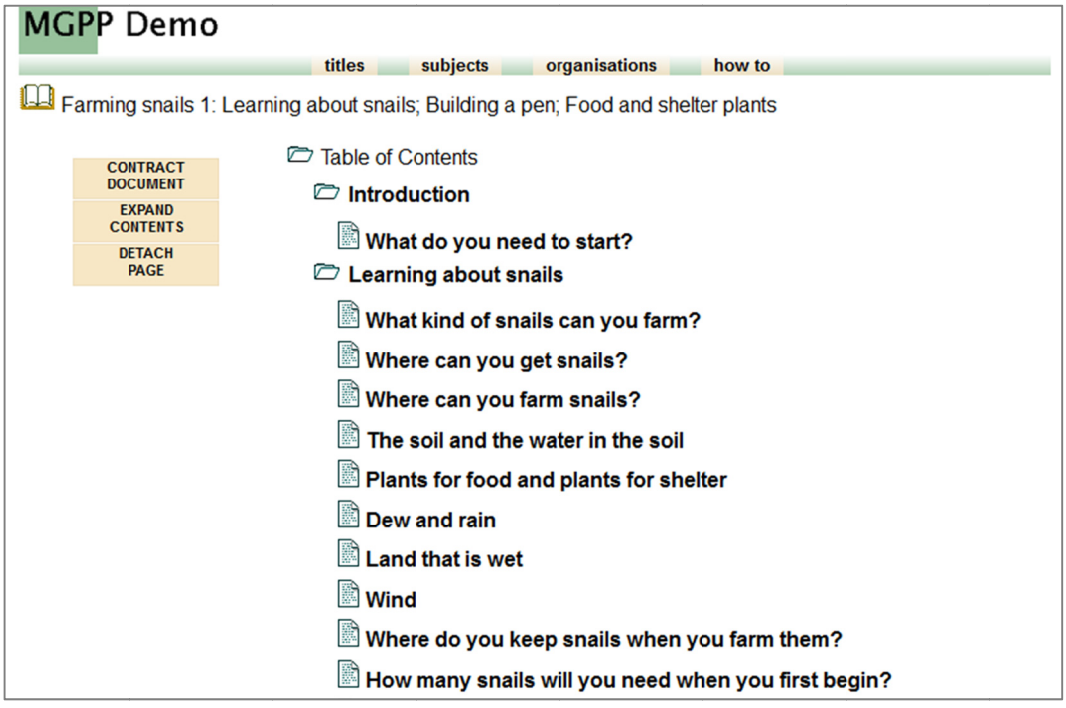


Figure 14: Greenstone demonstration website

Figure 15 shows the document view of the collection when the article is selected by clicking on it. The document view can contain text, images and other content as created by the user.

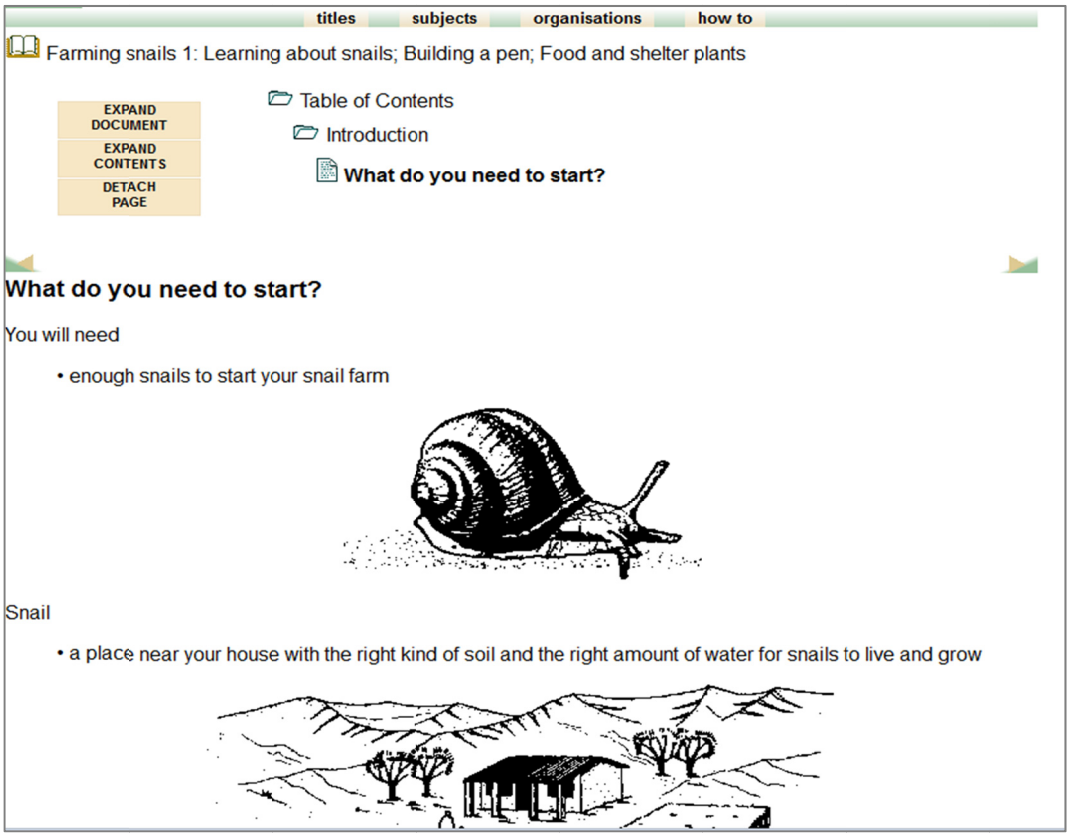


Figure 15: Greenstone collection document view

Greenstone 3 is the latest version of this software. The software is fully customizable and has a modular design structure. One of the key features of the new version is the ability to maintain its flexible functionality while at the same time remaining backward compatible with previous versions. The code base for the software has been fully rewritten in Java. The new software also uses Apache

Axis SOAP[19] (Simple Object Access Protocol) which enables servers located across the globe to serve the same content in a distributed manner.

[19] http://axis.apache.org/axis/java/index.html

2.6. Properties of Digital Libraries

The table below compares the properties of the various DL and a representative content management system, which we now discuss.

	Command Line	Collection Management Software	DOM	Full Text Search	End-User Web Interface	Metadata Sets
DSpace	✓	Fixed workflow	Attached to top level	✓	Moderate	Restricted
EPrints	✓	Fixed workflow	Not Available	✓	Moderate	General
Fedora	✓	Desktop workflow supporting remote access	Uses Fedora object model	✓	No Interface	General
Greenstone	✓	Desktop & Flexible management, Remote access through GLI	Document object model	✓	Highly Configurable	General
SilverStripe	✘	Fixed workflow	Integrates into DOM	✘	Moderate	Restricted

Table 1: Properties of various DL

The Table 1 clearly identifies Greenstone DL as being the closest choice to the requirement stated in Section 1.2. We can also identify that DSpace, EPrints and SilverStripe use a fixed workflow management system while Greenstone and Fedora have a flexible management system with the ability to be remotely accessed. All the DLs except SilverStripe provide (sometimes requiring a third party extension) full text search capabilities and a command line interface. SilverStripe provides a good interface for content creation but lacks flexibility in managing metadata sets.

Among the DLs discussed above, the two that stand apart from the others based on the feature set are Fedora and Greenstone. However, Fedora (as provided) lacks a turn-key user interface and the effort involved to create one would be time-consuming.

2.7. Strengths and Weaknesses across all DL systems

Apart from the features of DSpace discussed in Section 2.6, its hard-coded reliance on Dublin core metadata is a severe restriction. Another limitation of DSpace is that it has been designed for organizations that have a centralized computing infrastructure.

The limitation of the EPrints digital library software on the other hand is its design. The digital library is primarily intended for scientific and academic publications where digital content is not added or modified often. This system uses statically generated pages even though it supports multiple archives under a single instance. When digital content is added or modified within a repository, the changes become visible only after the user interface is regenerated (static pages). Therefore there is a delay between editing and publishing of a digital object which may not be ideal in some situations.

While the Fedora digital library has many of the sought after features, as discussed in Section 2.6, it does not have a dedicated module to manage item level hierarchical security and a customizable user interface. Fedora provides a foundation to build customized schemes for different uses, but does not provide a complete singular solution, which was one of the key requirements for this project. Although Fedora can store and access any type of digital content it does not provide the level of flexibility across the whole information schema.

SilverStripe utilizes a fixed mapping between the underlying database and object-model which is not appropriate for all users. Setting up an interface layer to the different databases within SilverStripe's data-object design will enable it to connect to other

applications, however this method of configuring the system is still under development. One of the limitation of SilverStripe is that it does not support a command-line ingest process. SilverStripe provides a workflow based comments tracking system when comments posted by users go through an approval process before being published.

Greenstone is designed with simplicity at its core. It can be used by anyone with even basic computer skills and can be installed on personal computers and within organizations large or small. One of the main advantages of Greenstone that stands out is that while most of the other library systems need the user to identify the metadata information when adding digital content, Greenstone's architecture is based on a plugin based system which enables this to be automatically performed, in addition to manually assigned values making it dynamic and customizable.

Chapter 3: Design and Implementation

In this chapter, we discuss the various new features that were developed based on the requirements and how this was integrated with the base Greenstone 3 system. To set the screen for this discussion, we start with a review of the Greenstone architecture.

3.1. Review of Greenstone Architecture

3.1.1. Overview

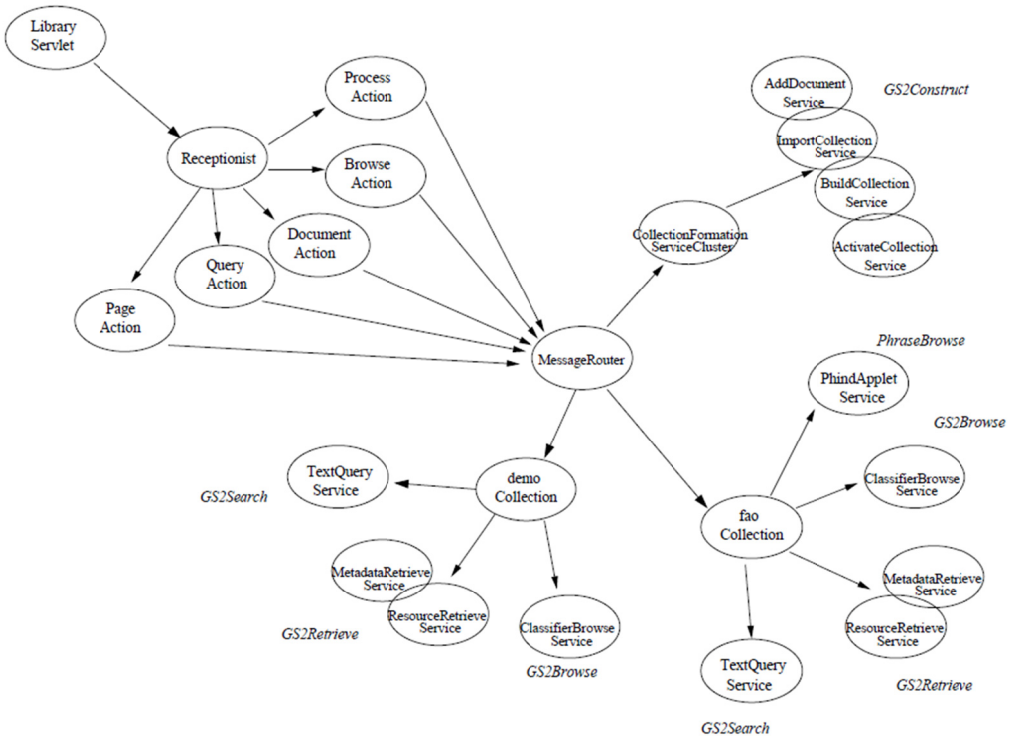
The architecture of Greenstone 3[20] has evolved over time using the experience gained from building the previous versions and by researching other open source software applications and projects. Greenstone 3 uses a combination of Java Servlet technology, XML[21] (Extensible Markup Language) and XSLT[22] (Extensible Stylesheet Language Transformations) to deliver the digital library interface to the end-users. Greenstone has a built-in access control mechanism which manages the authentication for the collections and individual documents. Users are provided appropriate authentication and can only access the services they are entitled to.

Between the version 2.x series and the version 3.x series, the software architecture was redesigned using a module topology

[20] http://wiki.greenstone.org/wiki/index.php/Manual
[21] http://www.w3.org/XML/
[22] http://www.w3.org/TR/xslt

method [6]. For portability reasons, Java was chosen as the implementation language, and messages encoded in XML were used to manage all communication between modules.

Although this top-level description above has many similarities with UMDL[23] (University of Michigan Digital Library) [7], on closer inspection Greenstone 3 has a very different infrastructure. The Greenstone 3 digital library system comprises of a digital library backend and a front end-user interface called the receptionist. An example of a simple standalone Greenstone implementation [8] is illustrated below in Figure 16.



[23] http://www.lib.umich.edu/digital-library-production-service-dlps

Figure 16: An example of a standalone Greenstone site, reproduced from "The design of Greenstone 3" [6]

The sample digital library shown in Figure 16 contains two collections, called *demo* and *fao*, which are linked to a group of services. All public functions provided by the digital library are "services." The `AddDocument` service adds a document to a specified collection while the `ImportCollection` service converts the document from its original format to the system specified format if required and imports it into the system. The `BuildCollection` service builds and manages the indexes and browsing structures of the collection. The `ActivateCollection` function actives a new collection in the system after which it can be viewed using the web interface. All these services work in conjunction to form the digital library collection.

From a user's point of view, a collection within Greenstone 3 is a focused group of documents which can be accessed through a uniform interface. From the system standpoint, the software can be defined as a cluster of related services which are grouped together by the data that encapsulates them. In the example shown in Figure 16, there are three modules providing a set of four services. The "Search" module manages the `TextQuery` service, the "Retrieve" module manages the retrieval of documents using the `ResourceRetrieve` service, the `MetadataRetrieve` service manages the

retrieval of metadata, and the `ClassifierBrowse` service handles the "Browse" function for the browsing of metadata. These combined services provide searching and browsing functionality for the Greenstone collection.

The Receptionist module shown in Figure 16 delivers a Web-based front end which acts as the main point of interaction with the DL. The top-level layer to the software implementation is a Java servlet which manages HTTP commands requests received via URLs and arguments which are translated into an XML output for the Receptionist. The Receptionist manages various types of requests made to the digital library by the `MessageRouter`. Figure 17 illustrates the "raw presentation" of collection metadata in Greenstone.

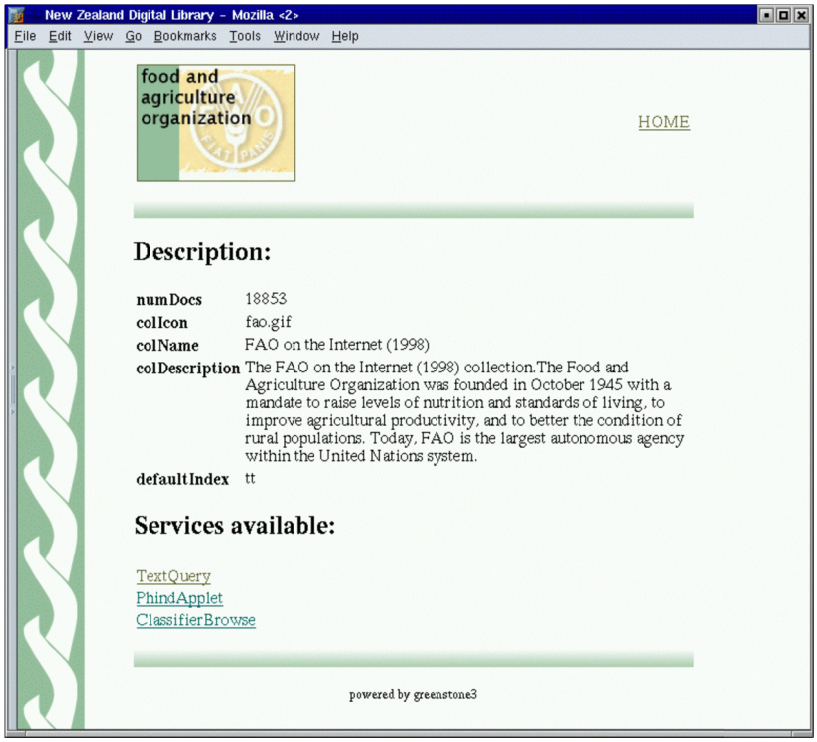


Figure 17: "Raw presentation" of collection metadata

Message Parsing

The system uses external and internal messages enclosed in message elements, which comprises of requests and responses. Requests received from outside, trigger corresponding actions in Greenstone 3. The requests that are received from a servlet are then forwarded to the Receptionist. CGI arguments are passed via this request which returns an XML page and is rendered in HTML. Internal messages within the system follow a standard format. Internal messages contain multiple requests and responses. The number of responses depends on the number of requests and is synchronous.

External requests received by the Receptionist are redirected to the appropriate Action module based on the action attribute. Based on the arguments the Action will trigger internal requests to the MessageRouter and returns the response to the Receptionist. The Greenstone servlet transforms the data using XSLT and renders HTML pages. The various types of internal requests are: describe, process, format, system and status. The system's main work is performed by process requests.

(i) **'describe'-type messages**

This type of message can be sent to any of the system modules. The response from the module is sent as a semi-predefined XML.

(ii) **'system'-type messages**

The cached information in MessageRouter, Collection or ServiceCluster can be updated by a System request. The MessageRouter comprises of a set of Collection module which communicate between each other. The MessageRouter also contains XML information about the collection and returns this upon request.

(iii) **'format'-type messages**

These messages are used to format the search results, document display structure, items classified in hierarchy. Format statements are written in XSLT or GSF (GreenStone Format). The receptionist converts the GSF to pure XSLT syntax.

(iv) **'status'-type messages**

These messages are used to display the status of process-type services. The requests to a process-service produce a status indicating if the process has started successfully or is still running.

(v) **'process'-type messages**

The main functionality of the system is handled by the process requests and responses. The service that the process is responding to determines the format of the message.

(vi) **'security'-type messages**

The maintenance of the collection relies on the underlying security of the library software. Users are authenticated and granted relevant permissions based on the username and password combination provided.

3.1.2. Collection Centric

Collections can be built within Greenstone using the Greenstone Librarian Interface (GLI). New collections can be replicated based on existing collections or built entirely as new collections. Each of the collections in Greenstone has a corresponding sub-folder with the `collect` folder. These collections are self-contained and hold all the configuration and data within.

In this section we discuss the directory structure of Greenstone and the various file formats used. Figure 18 shows the structure of the `GSDL3HOME` directory followed by a brief description of each of the directories, namely: `import`, `archives`, `index`, `transform` and `etc` with its contents.

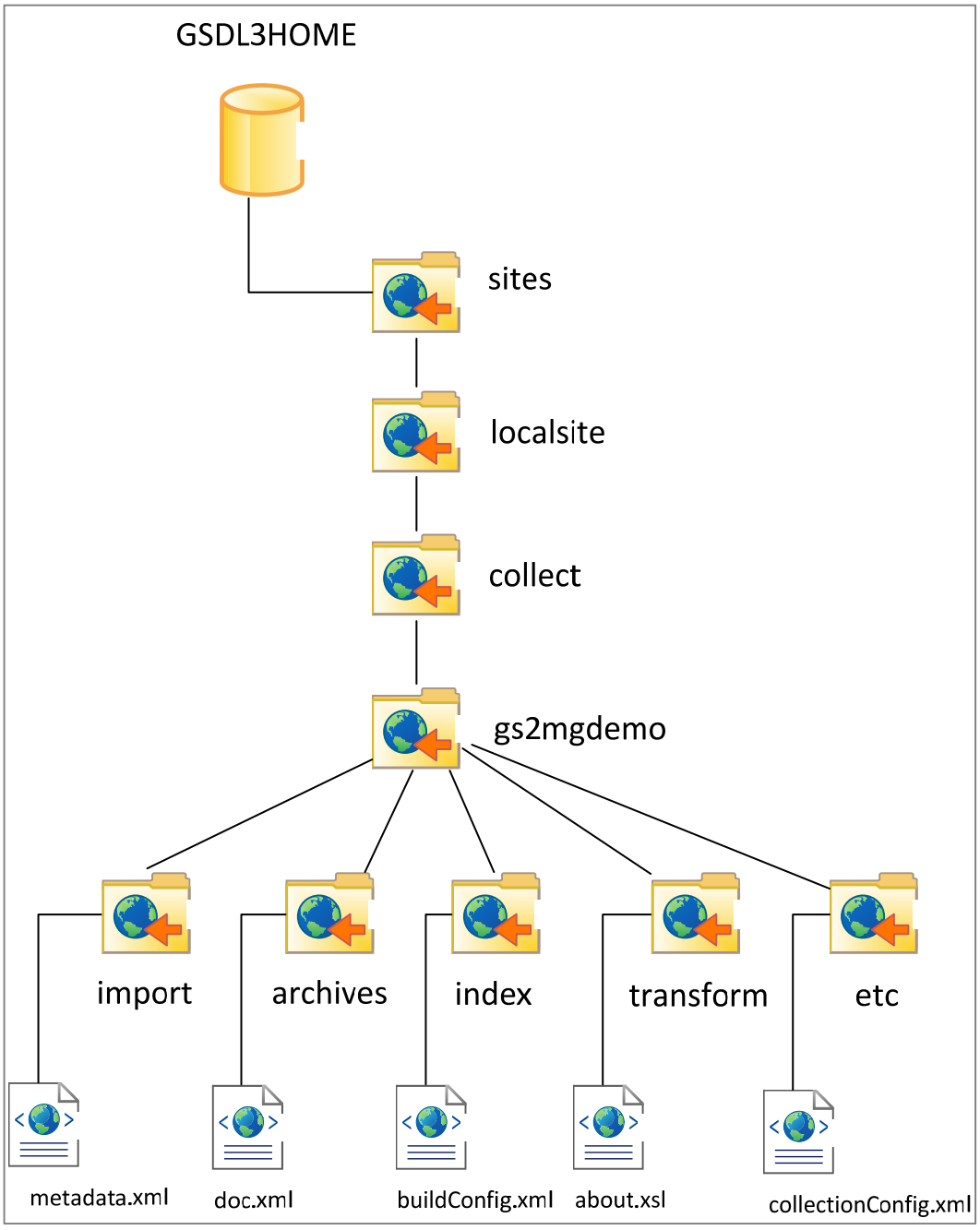


Figure 18: Directory structure of GSDL3HOME

(i) Import folder

The `import` folder contains the source material of the collection such as images, text, ocr, documents etc. Any

metadata that is assigned manually to the documents using the librarian interface are saved into this folder automatically as XML files called `metadata.xml`. Figure 19 shows the sample structure of `metadata.xml` file.

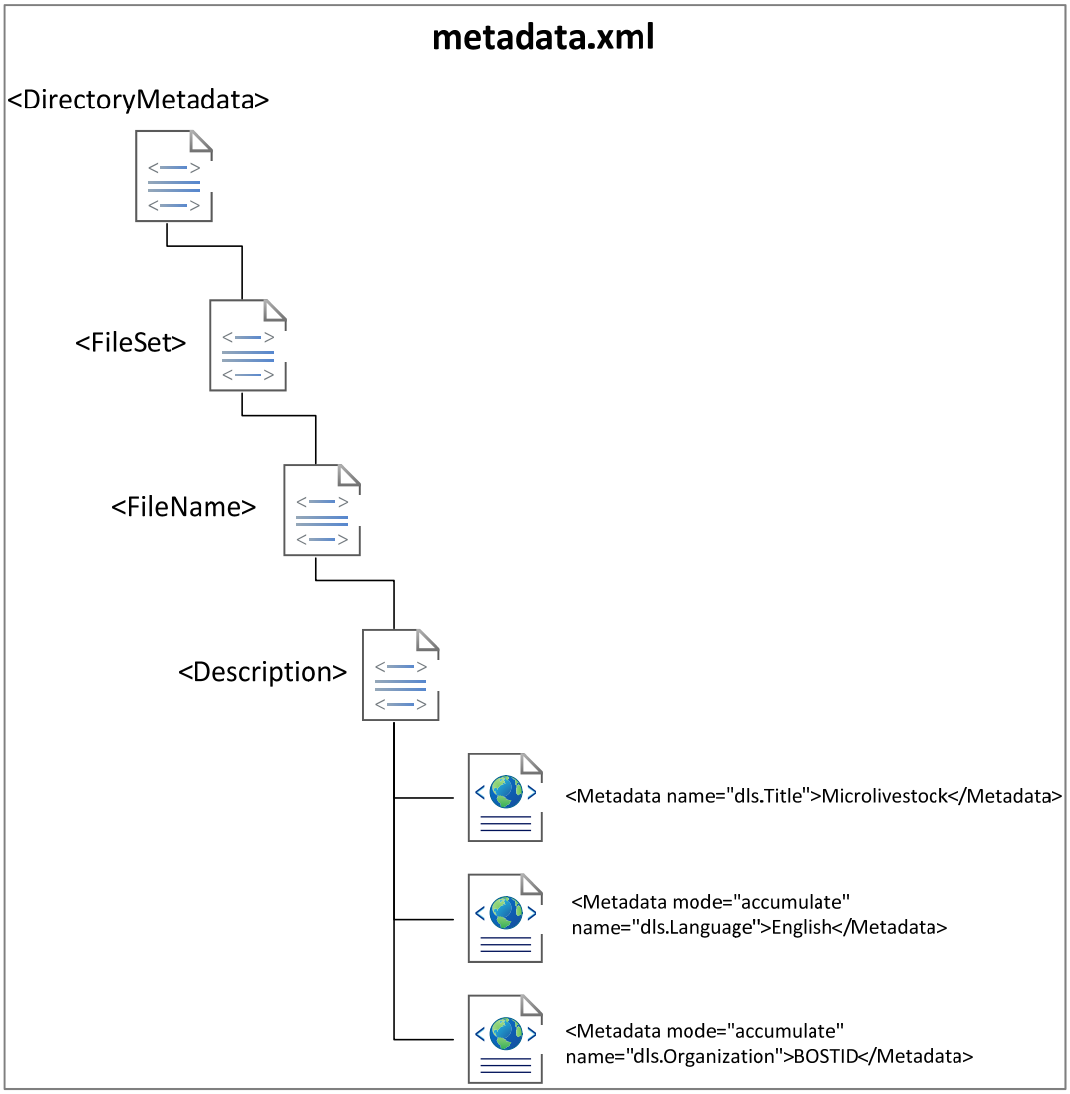


Figure 19: Structure of metadata.xml

The import process (`import.pl`) collects the documents from the import folder into the system and extracts the metadata from the `metadata.xml` file.

(ii) Archives folder

The `archives` folder stores the result of the import process. This folder also contains two important database files: `archiveinf-doc.gdb` and `archiveinf-src.gdb`, which are used for incremental building. The `archiveinf-doc.gdb` file contains the information about the files that constitute each document, and `archiveinf-src.gdb` records where these files came from in the `import` folder.

The `archives` file of each document has its own folder in the archives structure. The sub-folders within the archive folder are named according to the document's object identifier (OID). The default method for assigning OIDs is based on hashing to pseudo-random number depending on the document content. The OID's are character strings starting with the letters `HASH` as shown in the example below.

```
HASHe14e36cba08bd41c663237
```

Each of the sub folders within the `archives` folder contains a `doc.xml` file. This file stores document structure and content

details such as Section, Description and Content as shown in Figure 20.

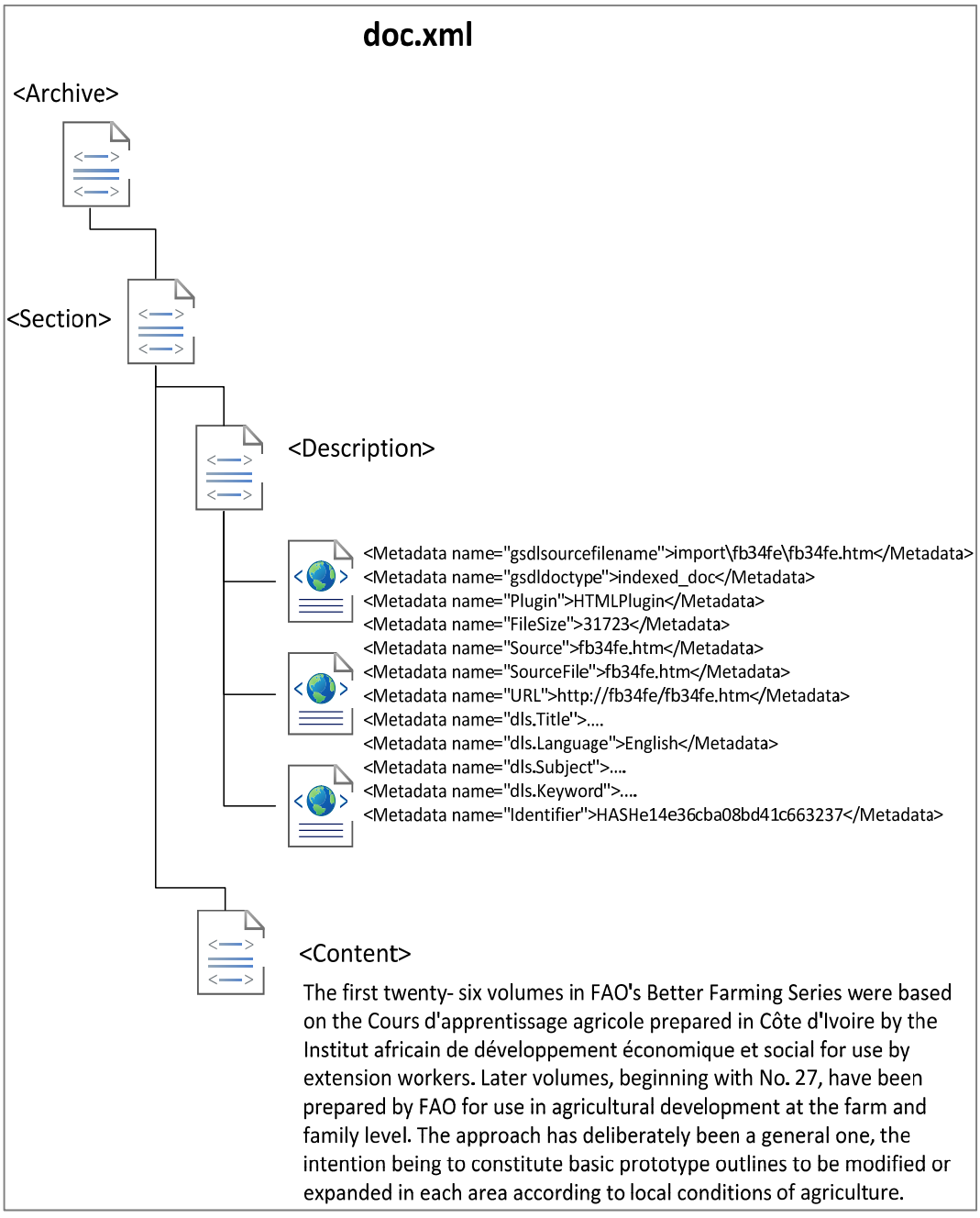


Figure 20: Structure of doc.xml

(i) Index folder

During the building process (buildcol.pl) a temporary `building` folder is created whose output is subsequently moved to the `index` folder. This folder stores all the information that is published to the end-user. The `index` folder contains the full-text index, database and hard linked if available on the operating system used, otherwise the file is copied.

A full-text index of the document collection contains the position of each word and the number of times it occurs in the collection. The default indexer for the collections is MGPP which operates at the word level. The database (GDBM by default) in the `index` folder is used to store and retrieve the metadata information. Figure 21 shows a sample of the `buildConfig.xml` which is located in the `index` folder.

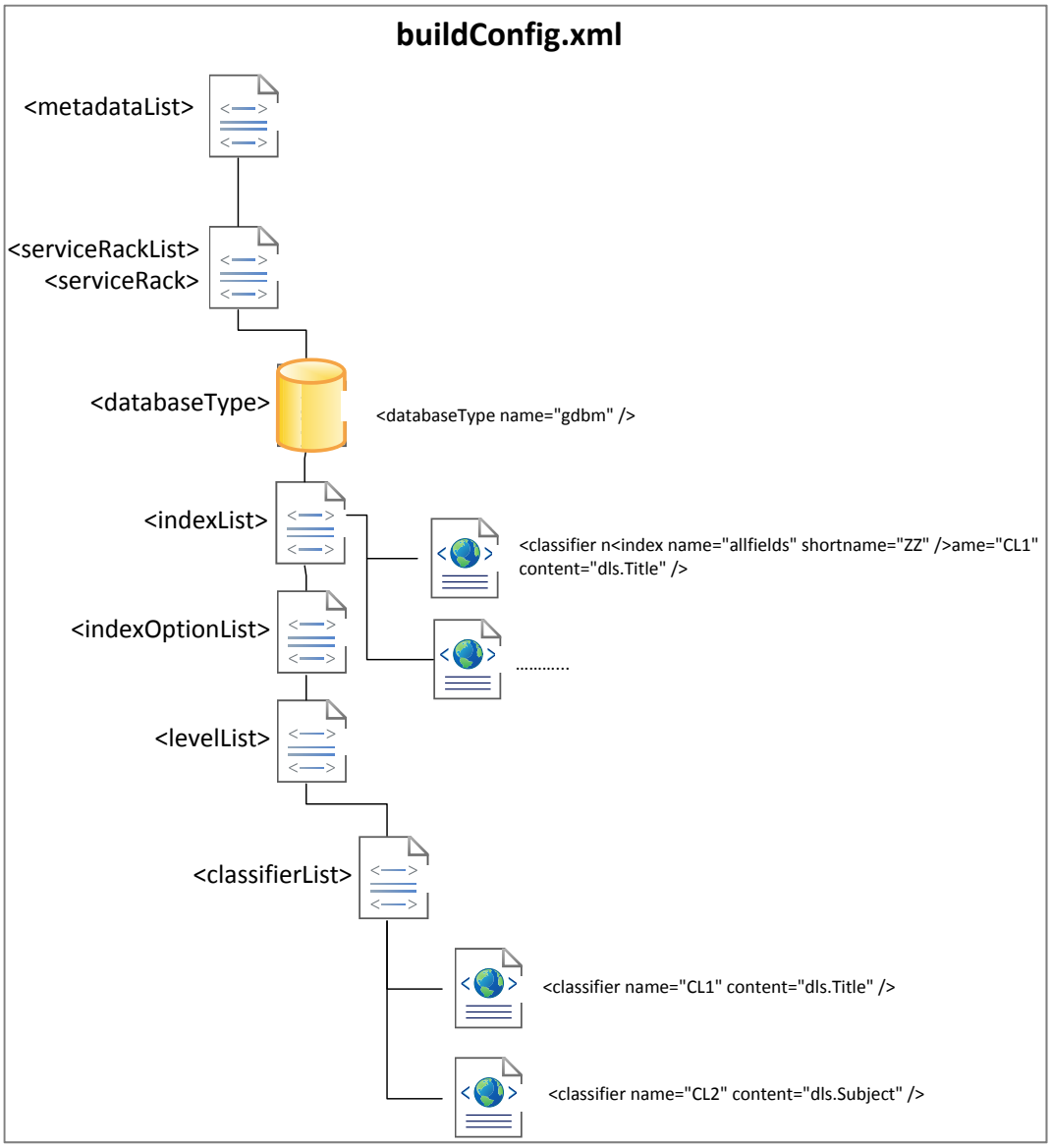


Figure 21: Structure of buildConfig.xml

3.1.3. Ingest

The data flow diagram in Figure 22 illustrates the various stages in batch driven collection building such as setting up the environment variables, loading documents into folders, importing documents into Greenstone and the final build process.

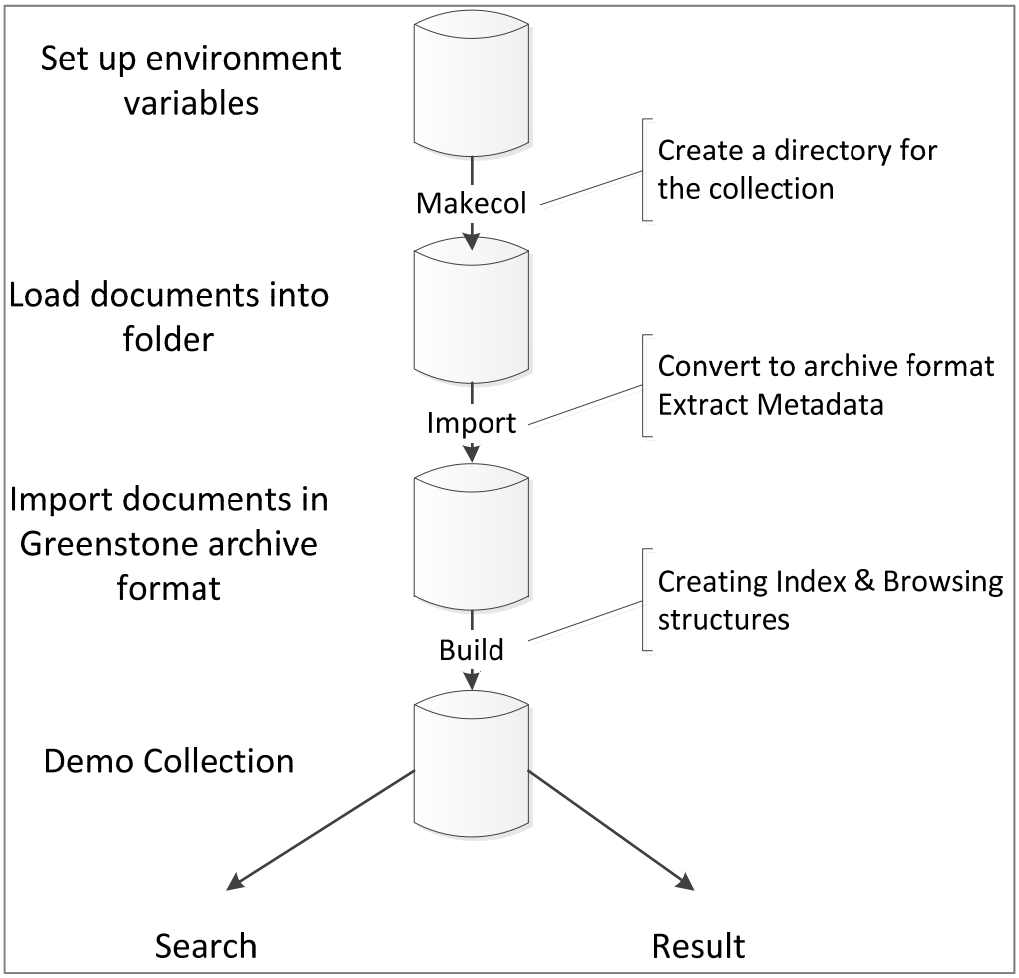


Figure 22: The ingest process in Greenstone 3

After setting up the environment, the source files in digital format and its text is stored as XML are used for the ingest process. The XML content is imported into Greenstone and indexed, after which the metadata is stored in GDBM databases. The Section 3.1.2 describes the import and build processes in Greenstone.

The Greenstone system provides the architecture and a flexible environment to manage the librarian interface and the end-

user web interface. The batch-driven ingest using GLI ensures a consistent structure is maintained across the collection and automates the repetitive processes, minimising the time consuming and labour intensive activities.

3.2. Greenstone relative to the requirements

We now turn our attention to a discussion of batch-driven ingest, web-browser activated ingest, document creation and deletion, editability of document metadata and annotations.

3.2.1. Batch-driven ingest process

Batch-driven ingest using command-line is adequately covered in Section 3.1.3 so we start our discussion at the second requirement.

3.2.2. Web-browser activated ingest process

Figure 23 illustrates the developed workflow of a web browser activated ingest process. The collection is built using GLI and available to be accessed from the web browser. Authenticated users can download a document, make changes or edits using locally

available resources and then ingest the file using the browser. A preview of the imported file is also displayed. The AJAX function saves the ingested document asynchronously in the background to the server. The manifest file maintains an inventory of all the individual files within the collection. The ingested document instantly replaces the existing version without any other user interaction.

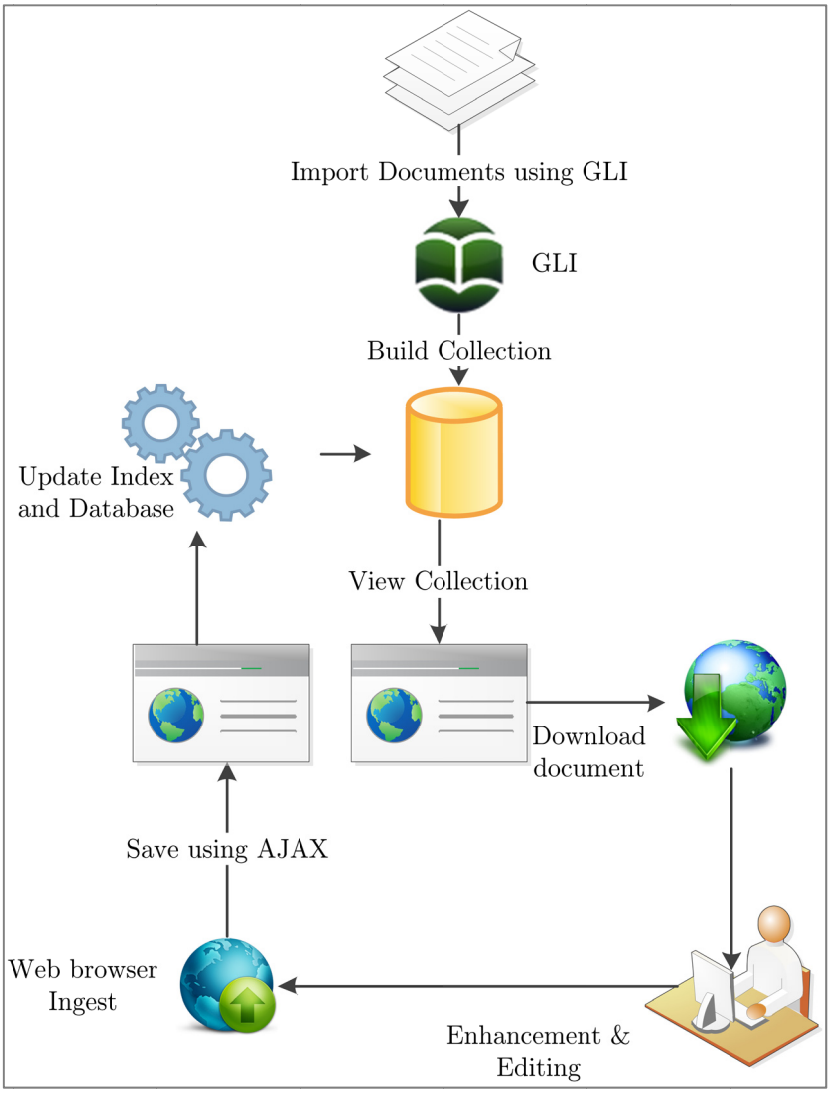


Figure 23: Workflow of web-browser activated ingest process

This drag and drop function is considered an important part of the front-end interface. Even though the various browser-versions offered today are very different in terms of platform support and usability, these differences can be overcome to produce a useful web application. As the properties and methods that the browsers offer evolve this function can be further enhanced in the future to improve the user experience.

3.2.3. Born Digital Document Creation and Deletion

The process of creating and managing direct-to-digital content that was added to Greenstone was built on top of the Seaweed framework [9] which enables the curator to interact with web content in a manner which resembles a word processor. This web based interface is used to create, edit or delete content based on the assigned privileges. The collection can also be customized to allow the editor to view the original image and the editable content side-by-side to enable ease of use. The benefits of using this approach to editing are:

- Decentralized maintenance which enables the system to be managed completely from a web browser which helps minimize errors within the workflow.

- The system is geared towards non-technical users. Curators with basic word processing knowledge will be able to create and manage the content just like a computer application. No knowledge of programming or HTML is required.
- The centralized authentication module ensures that only users with the relevant rights and permissions are allowed to edit content. All other users will only have read-only access to view content.
- The usage of style sheets and a redesigned interface ensures a consistent design is maintained across the library providing a seamless user experience. The content and design templates are stored separately to ensure that all sections are presented with the same consistent design.
- Navigation is automatically generated and managed by the Greenstone digital library software. All content is linked to the related paged image.
- Content is stored in a Greenstones database centrally and can be reused in any other section as per future requirements.
- Cooperation between the curators and content editors ensures that updates are done quickly and makes them accountable for the authored content.

A born digital document can be created from the web browser interface which uses the Seaweed framework. This framework makes

it a modeless system which supports the creation and publishing of content. Once the curator has created the content this is saved in serialized form to disk (see Figure 20). The save function triggers the rebuild and indexing of the content and makes it instantaneously available to the end-user.

Documents can be deleted directly from the web browser interface. The delete function removes the document and the associated files from the `archives` folder. This function also triggers the rebuild and indexing of the collection. Figure 24 illustrates the workflow of a born digital document creation and document deletion.

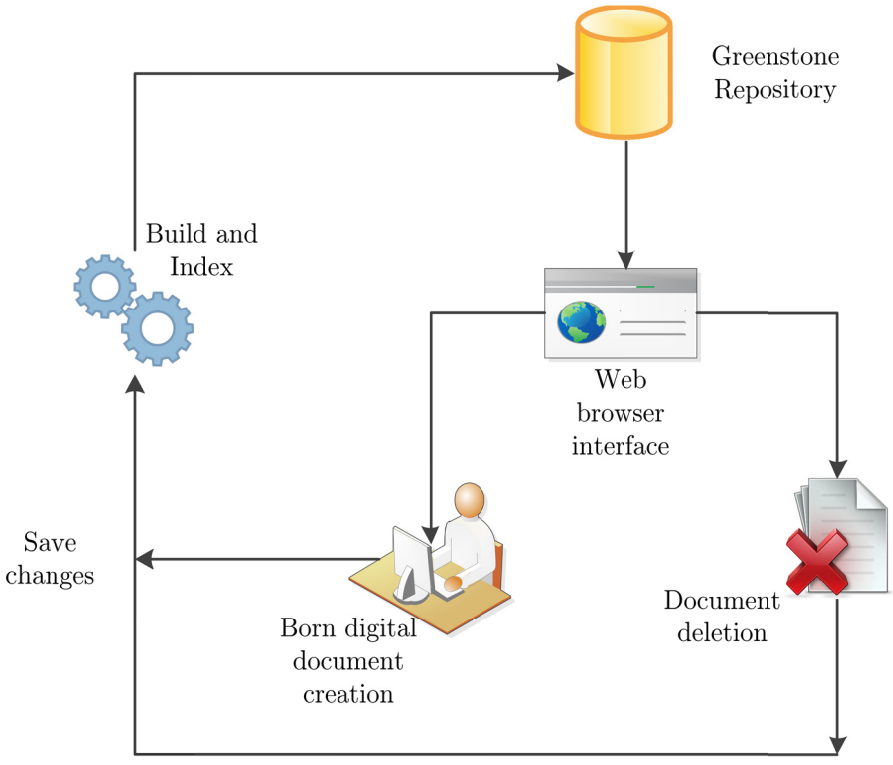


Figure 24: Workflow of born digital document creation and deletion

3.2.4. Editability of document metadata and structure

Document metadata and structure can be directly edited from the web pages with a standard web browser, without the need for switching between reading and editing modes. This technique simplifies the editing process.

The seaweed framework [9] bridges the gap between viewing, editing, and publishing making the content editable without the need to reload any page. The functions implemented within Seaweed mimic the actions of any common word processor like copy, cut, paste and undo to name a few and enable the editability of the metadata.

Figure 25 illustrates an overview of the Seaweed framework and how the DOM interacts with it. The figure contains three main sections, the web browser which displays the rendered output to the user, the DOM tree which is managed using the client-side JavaScript code and the Seaweed framework which presents the editable view.

The Seaweed framework separates the structure of the document and the content using CSS style sheets. The web page which displays content is made up of editable sections whose attributes are controlled using the HTML elements. The JavaScript

function of Seaweed capture all keyboard and mouse input triggers and performs the actions in real-time. The code is optimized to provide cross-browser functionality.

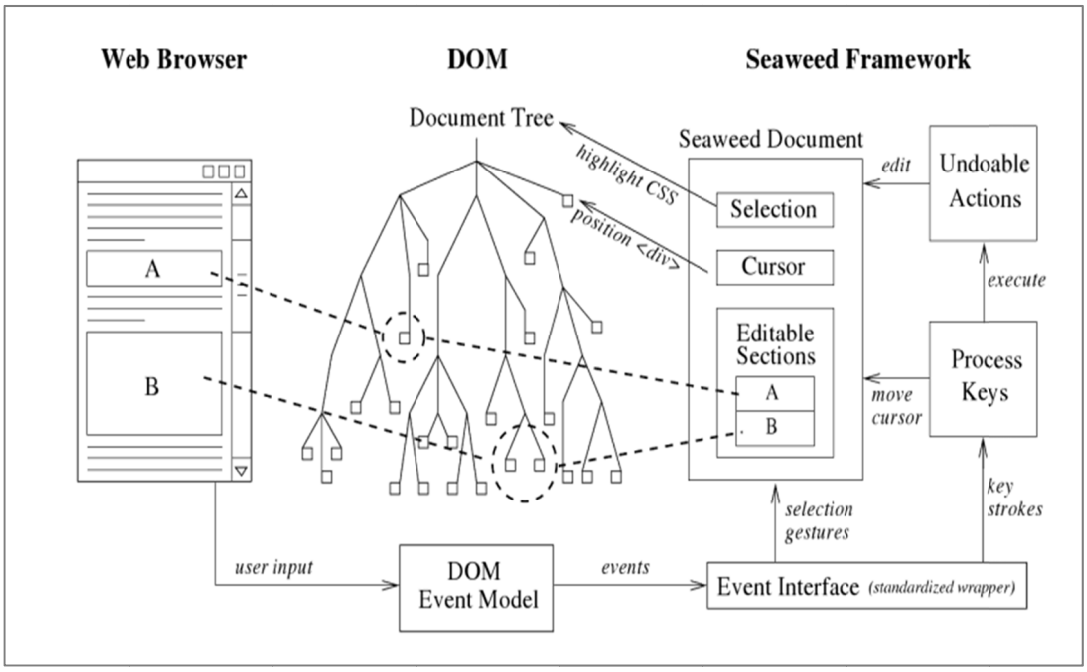


Figure 25: Overview of the Seaweed framework

In addition to this fairly pedestrian use of Seaweed, the structure of documents can now also be edited by dragging and dropping the structural elements as required. All of this is facilitated using a client-side framework which was developed entirely using JavaScript and conforms to the Document Object Model[24] (DOM) to completely manage the editing features. Figure 26 illustrates how the various components within Greenstone

[24] http://www.w3.org/DOM/

collaborate with each other to provide the seamless editability of document metadata and structure.

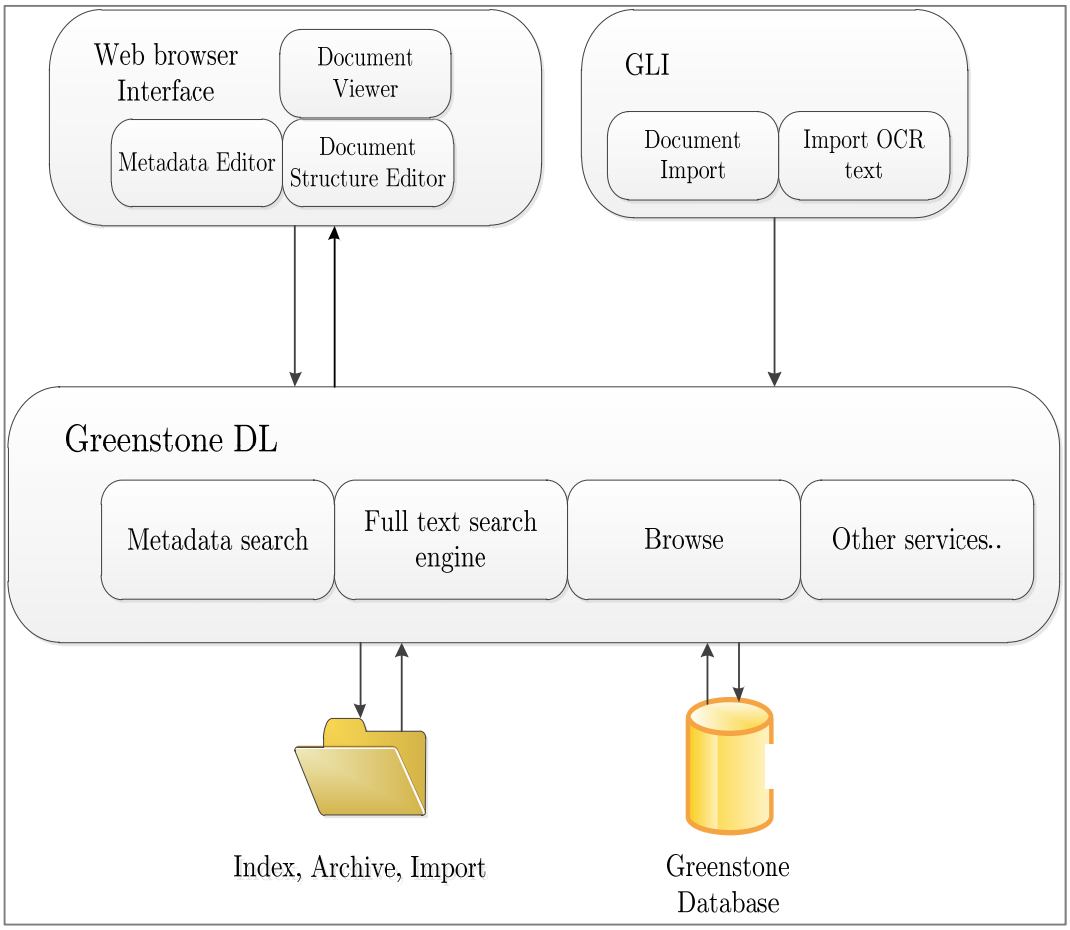


Figure 26: Workflow of document structure editor

Documents are ingested into Greenstone via a batch-driven ingest process or web-browser activated ingest as described in Section 3.2.2. The metadata and the structure are presented to the user on a web browser interface from the Greenstone DL. Metadata can be created, edited or removed using mouse gestures on the web

browser. See Section 1.4.8 for an example of it in use. Similarly, the structure of the collection which includes sections and sub-sections can be edited by dragging and dropping into a different location. Using this same drag and drop feature documents can also be merged into existing sections as required. All of these features provide the flexibility for the curators to manage the collections.

3.2.5. Annotations

The customized Greenstone DL software acts as a means of managing and sharing text and photo annotations. Annotations can be represented as comprehension, interpretation, cooperation and revision [10]. Documents and images serve as a medium to help us enhance our memory. A document or image can mean different things to different people. The annotation helps one interpret the meaning behind the document, adds more detail to the image and helps to remind the owner and reader of events, locations and people. From the advent of the digital age, the technology to preserve documents digitally meant that the ability to keep annotated visual records has become a necessity.

Annotations serve the end-users as a medium for unsupervised learning. Knowledge regarding facts, figures, dates, incidents etc. can be captured and saved randomly at various

instances by different people who have had experience and knowledge on the subject.

The scenarios from different social media illustrated in Figure 27 give a pictorial representation of the various types of annotations.

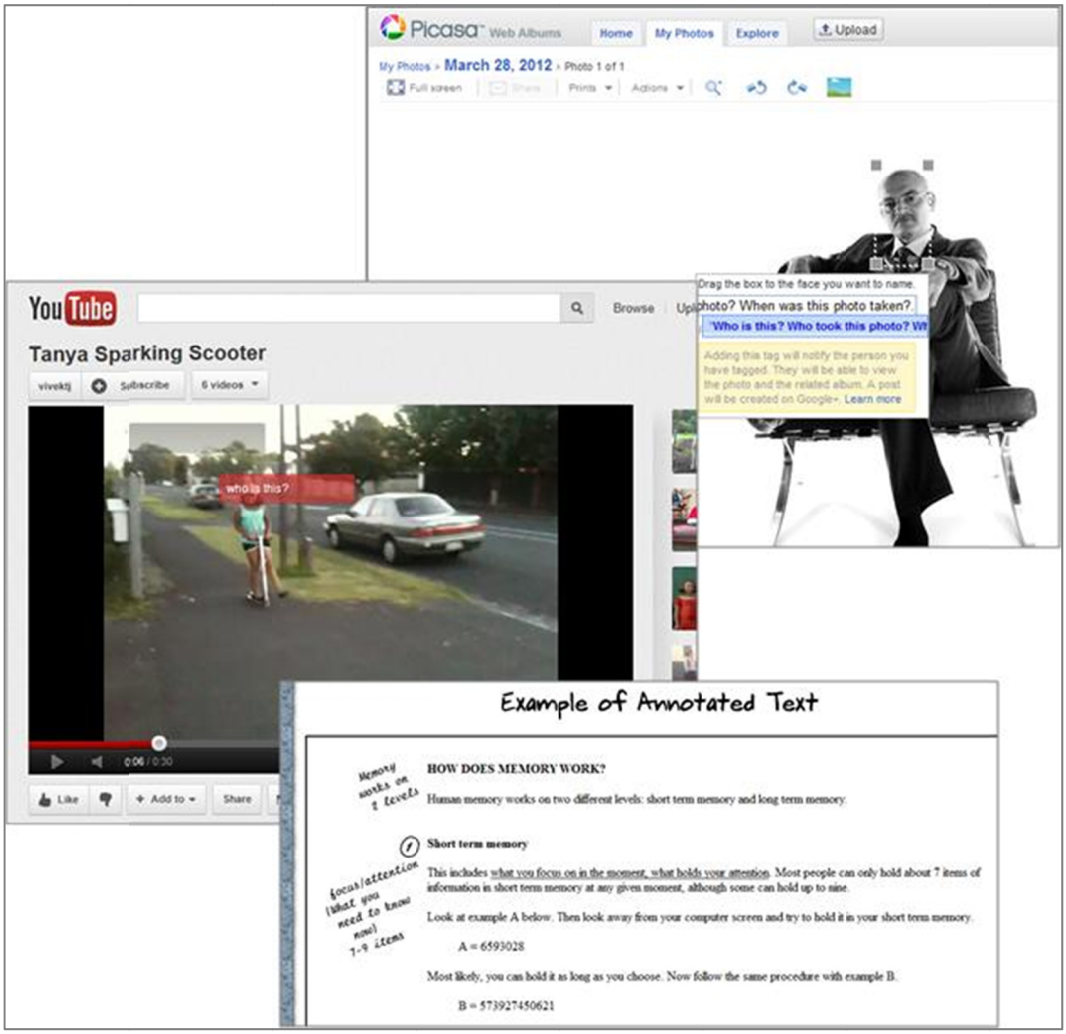


Figure 27: Pictorial scenarios from blogging/social media

Image Annotation:

Image annotations can be divided into text over image annotation and metadata.

Text over image annotation:

Descriptions in the form of comments or notes made either on an element or on specific parts of the image. These annotations are tagged based on the specific pixel location values and provides detailed information about the object or position it is pointing to.

Metadata:

The title, captions and metadata related to the image are mapped internally. These are made available to the user when required.

The advantages of using annotations are that they provide clear identification of notes and comments. Any time an annotation is tagged to an image, it appears beside the item. This identifies the object and the label that refers to it. The annotation tag identifies the section of the image which it refers to and marks it with defined borders around the annotated region.

Text Annotation

Annotating[25] is similar to marking notes on pages and is an excellent way of making the most out of the document. Integrating search within annotations makes it easy to find important information quickly for reviewing any text. It enables one to become familiar with the content and structure of the collection. It also provides a platform for engaging ideas and issues directly through comments, questions and associations. An annotation should serve as a medium to share the thoughts and ideas of the scholars.

Highlighting or underlining key words and phrases has been a common method to annotate books. Bringing this method to a web based platform makes it easier to review documents.

- The main uses of text annotations [10] are:
- Adding explanation
- Giving examples
- Providing factual evidence
- Considering an opposing view
- Comments and Responses

[25] https://github.com/okfn/annotator#readme

Figure 28 shows a summary of the lifecycle of an annotation and its functions as implemented in Greenstone for this project.

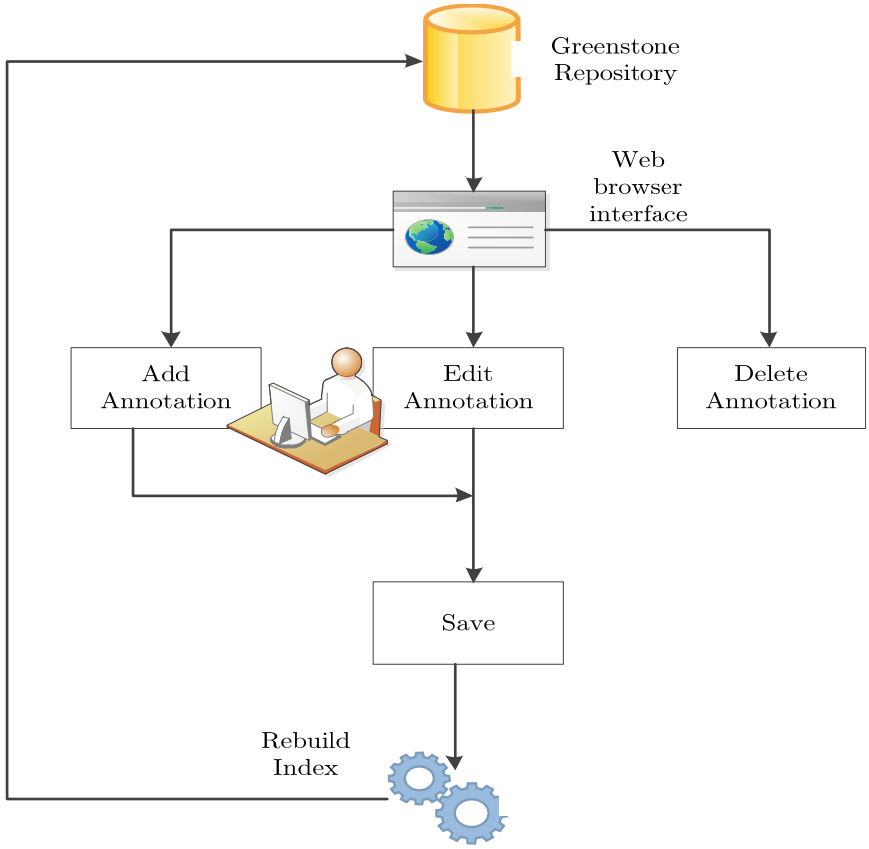


Figure 28: Annotator dataflow diagram

The annotations are saved as metadata and indexed either by the annotation text or its corresponding metadata elements using AJAX initiated 'ingest' capability as described in Section 3.2.2. The searching and indexing of the annotation text and metadata is managed independently from the application which is used to view the document.

Chapter 4: Developing the Pei Jones Collection

In this chapter, we discuss how the Pei Jones collection was developed and implemented, starting with building the collection followed by the various functionalities which were implemented as per the requirements stated in Section 1.2.

4.1. Introduction

The late Dr. Pei Te Hurinui Jones was one of the most prominent Māori scholars of the last century. An Interactive User Repository for his manuscripts has been developed as part this project. This invaluable content is currently preserved in physical form at the Mahi Mareikura, University of Waikato. This interactive system based on the analysis in Chapter 2, Greenstone digital library platform was designed to assist the scholars and librarians materials that the digital library is designed to store.

A digital version from the University of Waikato's library on Dr. Pei Te Hurinui Jones special collection inspires one to think about this indigenous language and its historical contribution. The knowledge gained from past generations need to be preserved for the future. Around the world, institutions are in the process or have already set up digital libraries which are being studied and developed by scholars and practitioners. A Case Study of Pei Jones

Collection aims to provide the scholars and users with an interactive management system that handles multilingual digitized documents. It is necessary to appreciate the complexity of documents and manuscripts, and the type of raw materials that the digital library is designed to store in order to develop a system that addresses all the requirements as stated in Section 1.2.

Figure 29 shows some representative samples taken from documents that are to be used as a case study for the collection: letters, manuscripts and photos.

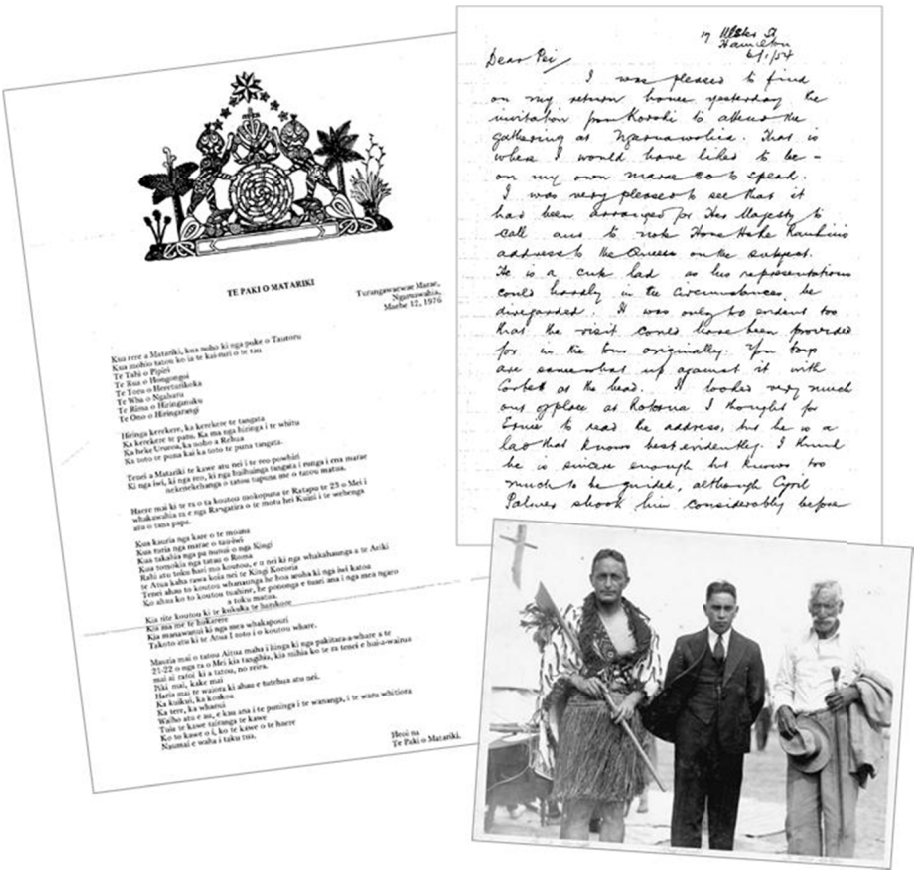


Figure 29: Sample documents from the collection

These manuscripts have (prior to this project) gone through a digitization process which involves scanning, Optical Character Recognition (OCR) and archival in digital format. The nature and type of manuscripts presented various challenges during the digitization process, making this a labour intensive process. These include,

1. Accuracy in the conversion of text by the OCR software.
2. Limitations in the ability to detect and convert hand written text.
3. Lack of a standardized method of indexing, storage and presenting metadata for the digitized multilingual manuscripts.

4.2. Building a Pei Jones Collection

The Pei Jones digital collection was created using the batch driven ingest process as described in Section 3.1.3 and the steps involved in importing the digital content into the collection were followed. Figure 30 shows the output of the Pei Jones collection as viewed through the web browser end-user interface.

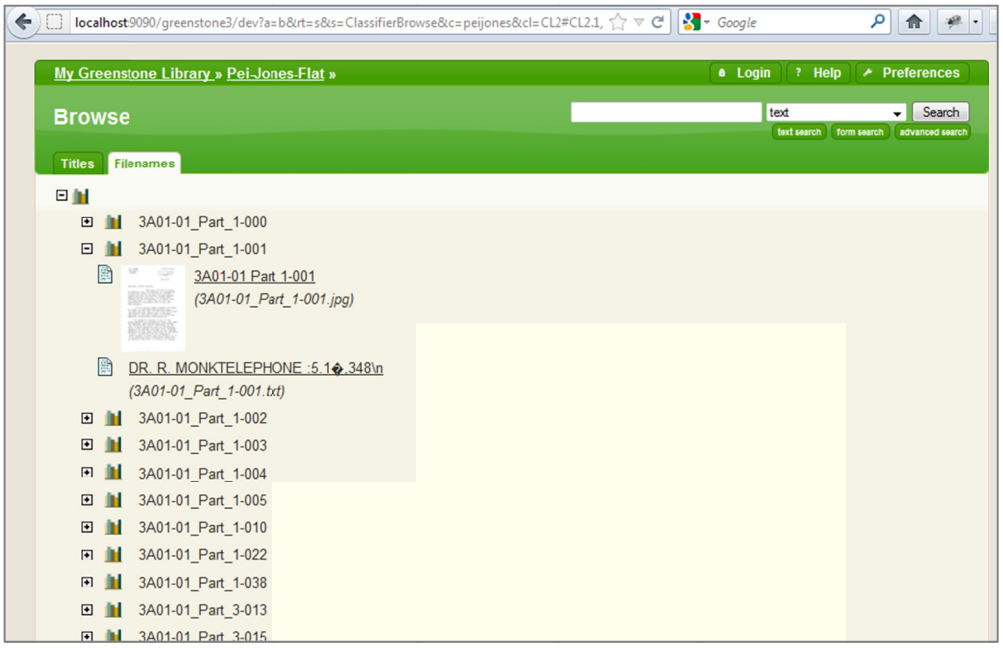


Figure 30: Output of the Pei Jones Collection

The following two configuration files are also created as part of the collection, namely: `collectionConfig.xml` and `buildConfig.xml`.[26] These files hold the metadata tags, design elements and all related information that are specific to the Pei Jones collection. The configuration files are encoded using standard UTF-8 format.

collectionConfig.xml

The configuration files [8] of the Pei Jones collection are made up of key metadata elements such as title, collection description, text to be displayed for indexes and format in which the search

[26] http://www.greenstone.org/docs/greenstone3/manual.pdf

results need to be displayed. The <metadataList> element specifies the collection metadata for Pei Jones collection. The <displayItemList> element specifies the collections language dependent information, such as the section details and document title. The information related to the display of the search index and results is managed by the <search> element. The <browse> element holds information on the classifiers and the <display> element managed the structure of the displayed document. The <displayItem> element is encapsulated within the <search> and <browse> element and provides title information which is used by the indexes or classifiers. All instructions related to the formatting of the nodes within the results are stored in the <format> element.

buildConfig.xml

The buildConfig.xml [8] file is generated during the build process of the collection. Information related to the collection such as metadata and the number of documents is stored in this file. Information related to the ServiceRack classes is also stored in this file. All information related to the built-in services is held in this ServiceRack.

4.3. Building a Māori Language N-Gram Profile

The Pei Jones collection comprises of a variety of multilingual documents, many of which are in Māori language. Therefore the automatic detection of Māori language metadata was a key requirement, however automatic detection of this language was absent from the basic Greenstone installation and consequently for the Pei Jones collection, steps were taken to rectify this.

4.3.1. N-Gram-Based Text Categorization

N-gram[27] is a "Contiguous sequence of *n* items from a given sequence." N-grams [11] can also be defined as a sequence of words or terms, where N represents the length of the word or term. Any word taken from a document can be represented as a sequence of overlapping N-grams. The definition of N-grams also considers spaces on either end of a word when the term is analysed. An example of this is shown below.

For example, the N-gram [12] sequence for the word "TEST" can be represented as:

bi-grams `_T, TE, ES, ST, T_`

tri-grams `_TE, TES, EST, ST_, T__`

[27] http://en.wikipedia.org/wiki/N-gram

quad-grams `_TES, TEST, EST_, ST__, T___`

In the example shown above the bi-gram consists of two character sequences while tri-gram takes three character sequences and so on. The N-gram language identification algorithm builds a frequency list of all possible sequences for every word in the whole document. This is then compared with known sequences from all possible languages and returns a probability based result.

This method of matching words based on its N-gram sequence is very accurate since it breaks every word into smaller parts and errors from mismatched words only affects a minimal number of N-grams terms. All the remaining words get an accurate match resulting in an overall high accuracy rate.

Other methods of language detection using principles of natural language parsing have proven to be more complex and yield less accurate results [13]. The accuracy of the detection process is consistent since it calculates the result based on statistical probability of N-gram occurrences and not the actual occurrence of the word.

4.3.2. Generating a Plain Text Corpus for Māori

NGramJ is a Java library used for automatic language recognition based on the N-gram technique. It uses language profiles

by counting the character sequences, which are then used for language detection. The application uses a command-line Java API to perform these tasks. Figure 31 illustrates the steps involved in creating a Māori language N-Gram profile for use in Greenstone.

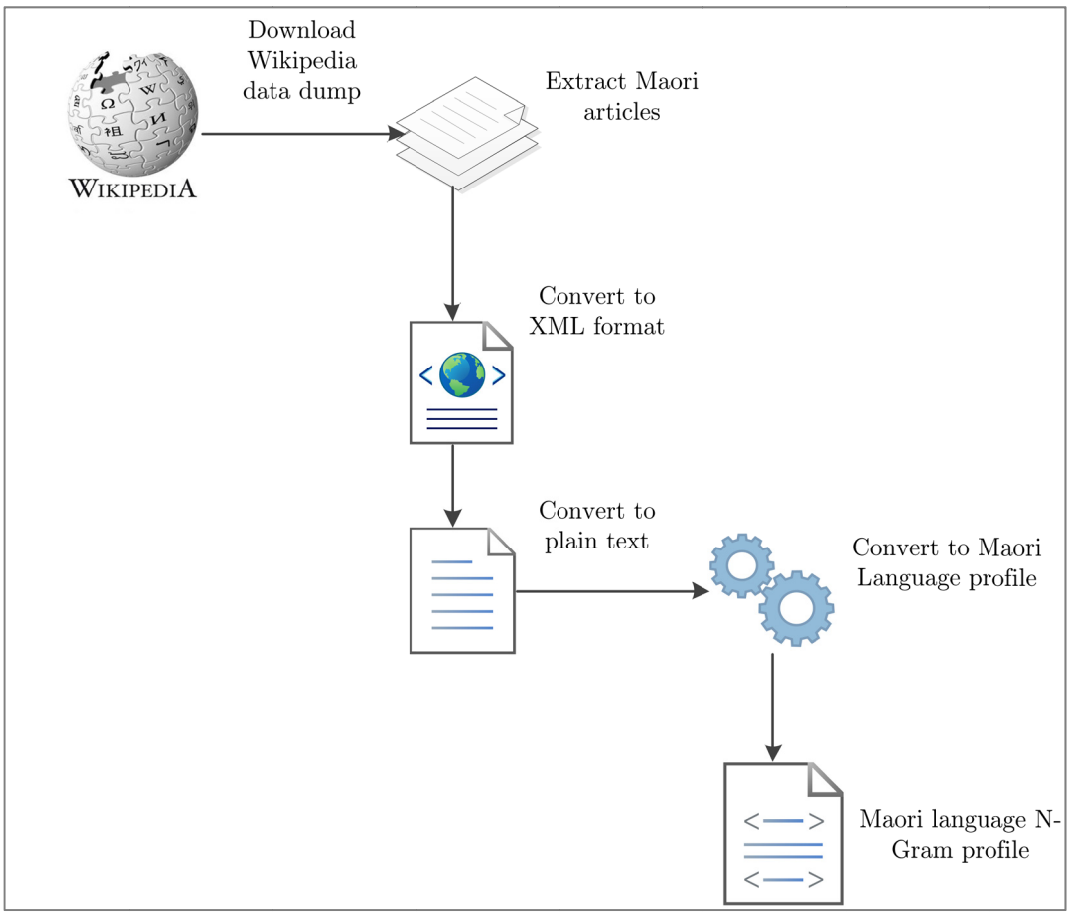


Figure 31: Steps involved in generating a Māori N-Gram Profile

The Wikipedia Extractors Toolkit[28] was used to analyse the Māori language Wikipedia dump[29] which comprises of the largest

[28] http://www.evanjones.ca/software/wikipedia2text.html

collection of Māori articles freely available on the web. The extracted Māori articles are converted into XML files and then to a plain text form. These plain text files contain the final language corpus extracted from Wikipedia. Running the `ngram.jar` against the text file created produces the `mi.ngp` (Māori language N-gram profile)

The functionality of the new Māori language profile can be tested by adding the `mi.ngp` file to the language folder. Figure 32 below shows the output of the language detection test on a sample Māori language file resulting in Māori (mi) 86.3%, Romanian (ro) 8.5% and Italian (it) 0.9% being the top three ranked choices.

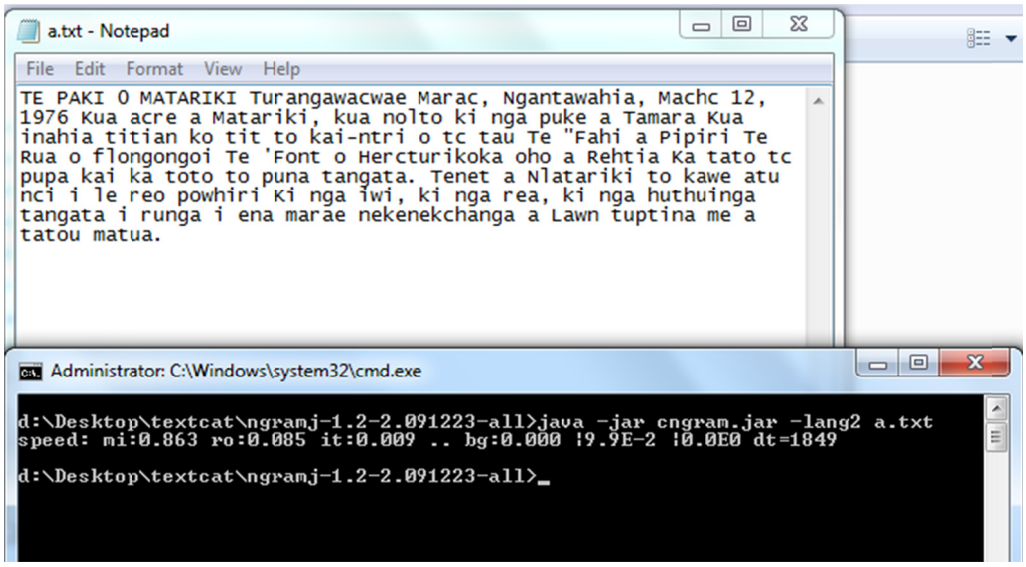


Figure 32: N-gram based Language Detection

[29] http://dumps.wikimedia.org/backup-index.html

4.3.3. Automatic detection of Māori metadata

The Māori language profile has been integrated into Greenstone by adding the N-gram Read Text Plugin via a Perl module. The Java N-gram solution then detects the Māori language metadata accurately.

The `ReadTextFile` contains all the functions required by the plugin. It reads the file and sets up the metadata which is saved in `doc_obj`, and then returns `(process_status,doc_obj)`. The function `get_language_encoding` removes all the HTML tags and returns an array containing the language and encoding information.

The `ngramj.pm` is a Perl module which includes the functions required to import the JAR files and parses the result and returns it into an array in the form of `'lang-encode'`.

The extracted metadata by the plugin, can be viewed via the Greenstone librarian interface when the collection has been built. Figure 33 below shows a snapshot of the Māori language detection plugin via the librarian interface.

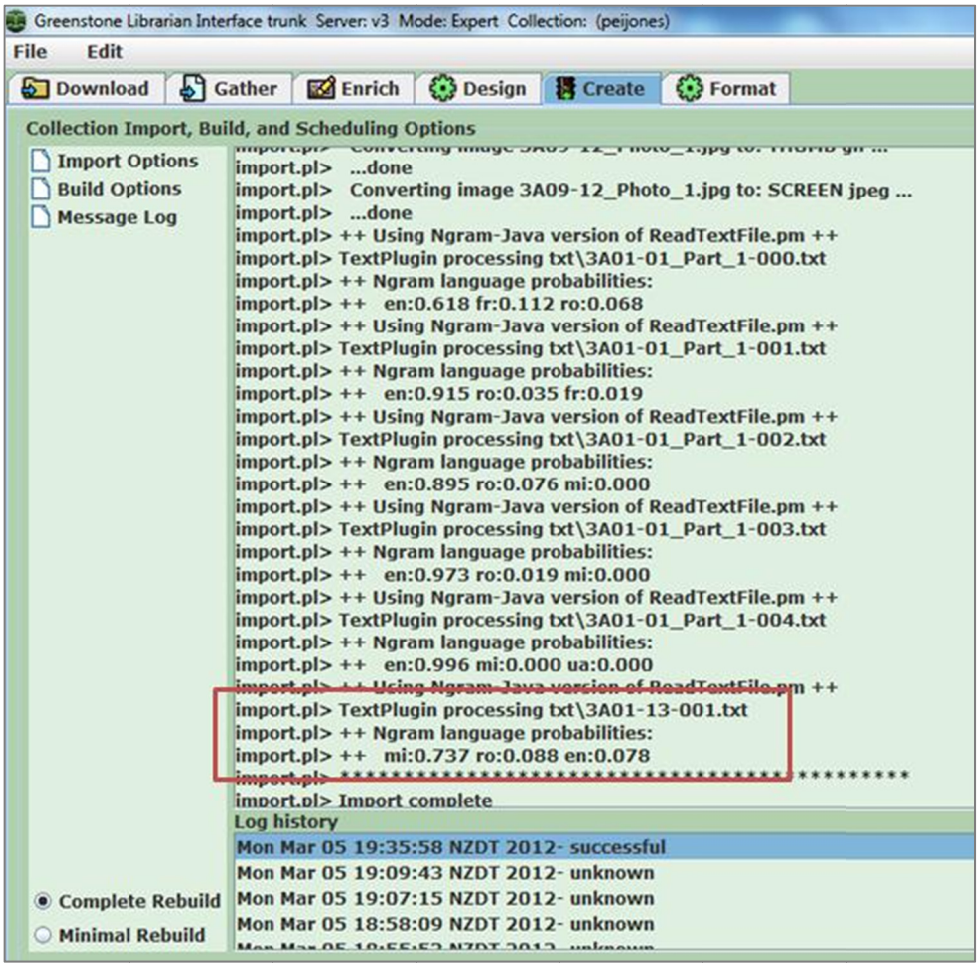


Figure 33: Māori language detection within the librarian interface

4.4. Zoom functionality

Legibility is a key aspect in this collection because of the volume of content that is handwritten. The ability to quickly magnify specific portions of the image for a detailed view from a web browser helps overcome the limitations of the original document.

The Zoomer JQuery plugin[30] is a useful open-source 3rd party JavaScript library, which was identified, that magnifies sections of the image as the mouse moves over the image and consequently was integrated into the document view to the Pei Jones collection. Since the Zoomer uses a jQuery[31] plugin, the jQuery library is loaded first followed by the plugin file which then invokes the new function on the area that needs to be zoomed. This function is flexible and fully customizable via CSS.

There are five parts to a Zoomer area. This allows flexibility in customizing each part, including the ability of having the "small" area with different content from the "large" zooming area.

- The Wrap – Wraps the text around the designated area.
- Small Image – The default viewable area that zooms when there is a "mouse over" action.
- Large Image – The content viewable through the Zoomer.
- Overlay – The zooming box that follows the mouse movement.
- Mover – A wrap for the large area and overlay.

The document view in Greenstone for this collection has been customized such that on loading the page a screen-view image is

[30] http://css-tricks.com/anythingzoomer-jquery-plugin/
[31] http://docs.jquery.com/Downloading_jQuery

initially displayed. The zoomer can be activated by clicking on any part of the image. A fixed area around the mouse gets magnified when clicked using the full size image. The movement of the mouse changes the zoomed area and it follows the position of the mouse movement. Figure 34 illustrated the steps involved in activating the zoom functionality in the Pei Jones collection.

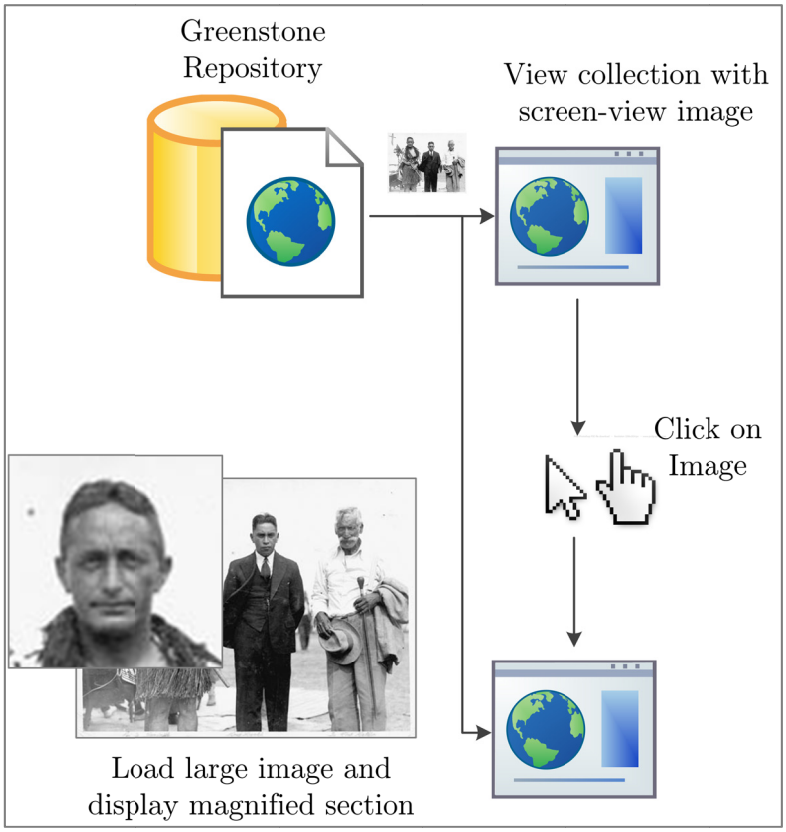


Figure 34: Activating the Zoom functionality

The zoom functionality uses a combination of JavaScript and Cascading Style Sheets (CSS) to achieve the result of zooming on a specific location. Within the Pei Jones collection the image plugin has been configured so that every image is saved as three sizes,

namely: thumbnail, screen-view image and large image. When a page is loaded the screen-view image is displayed by default. When the zoom function is activated, the script computes the relative position of the mouse on the image, using the X and Y coordinates and loads the large image with the corresponding coordinates. This gives an effect of zooming in on a specific location.

The Zoomer is loaded by calling the function `readyImagesForZoom()` when loading the page. The screen-view image is stored in the `screenViewImageDiv` variable. The `getElementById()` method accesses the elements `noZoomImage` and `wrap`. The expansion size is set to 50 as the default for the Pei Jones collection.

The `addzoom` function is triggered anytime an image is clicked. This activates the Zoomer function to loads the magnified image. On clicking the image and if the zoom is currently active or during exit of the function the `removezoom` function to loaded and the zoomer is cleared from view. Figure 35 shows us how a zoom function is used to magnify a specific location of the image.

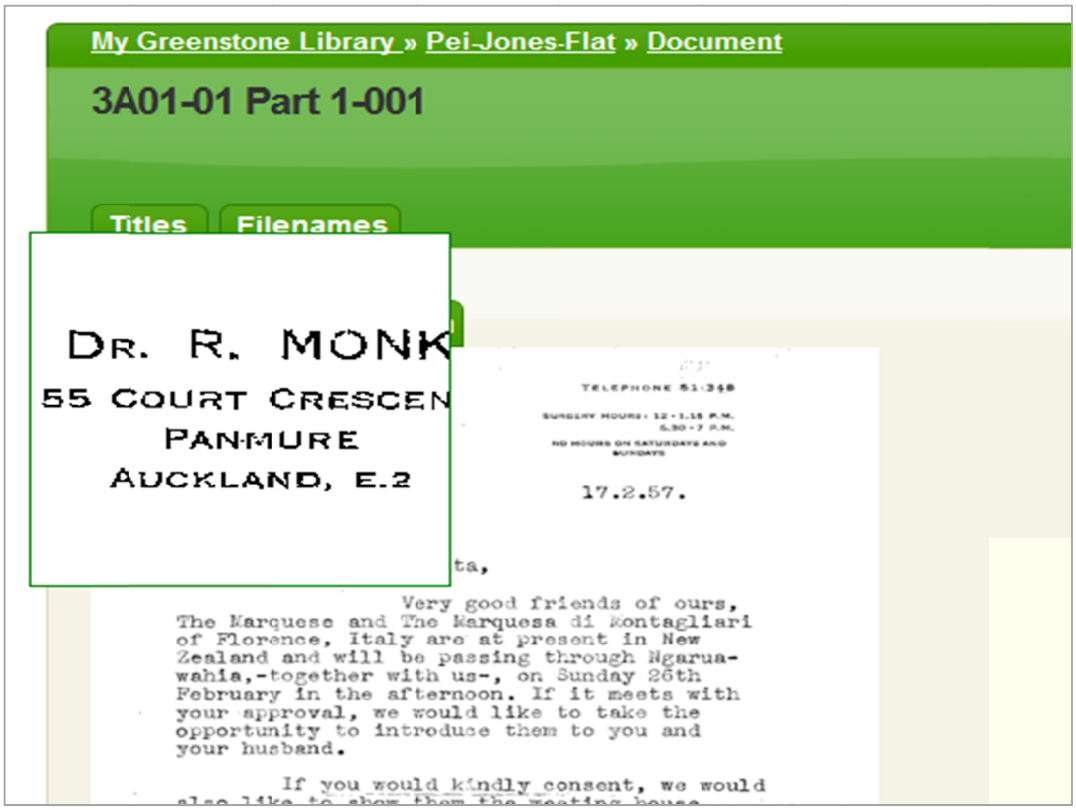


Figure 35: A sample of Zoom function

4.5. Text Annotation

Annotations are used to capture, track and search for text, tags and users of a document. By capturing user feedback and activity, a historic record of the activity is created which can be tagged to the lifecycle of the actual document. The flowchart in Figure 36 describes the data flow path when annotations are made or viewed on the Pei Jones collection.

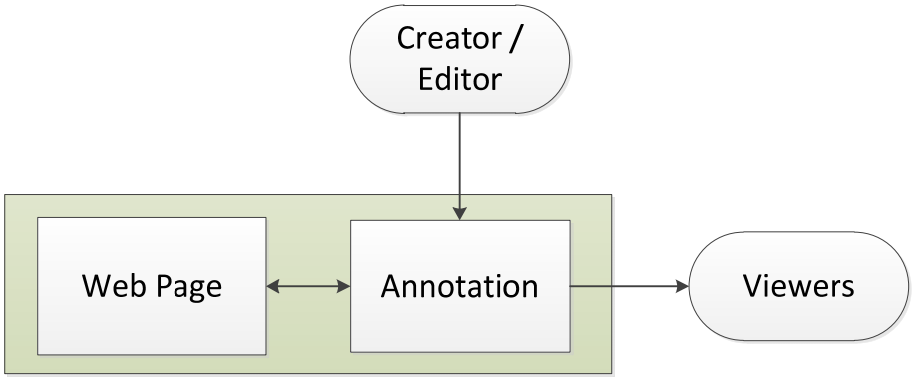


Figure 36: A Text Annotation dataflow diagram

The text annotator[32] integrated into the document view of Greenstone uses JQuery at its core and can be invoked by calling the `.annotator()` function. Including the `annotator-full.min.js` loads a collection of plugins which are called by running `.annotator("setupPlugins")`. The annotations are stored along with the metadata and can be indexed either by the annotation text or its corresponding metadata elements. This provides the application the ability to group documents with similar annotations. The searching and indexing of the annotation text and metadata is managed independently from the application used to view the document. The flowchart in Figure 37 lists the annotator functions.

[32] https://github.com/okfn/annotator

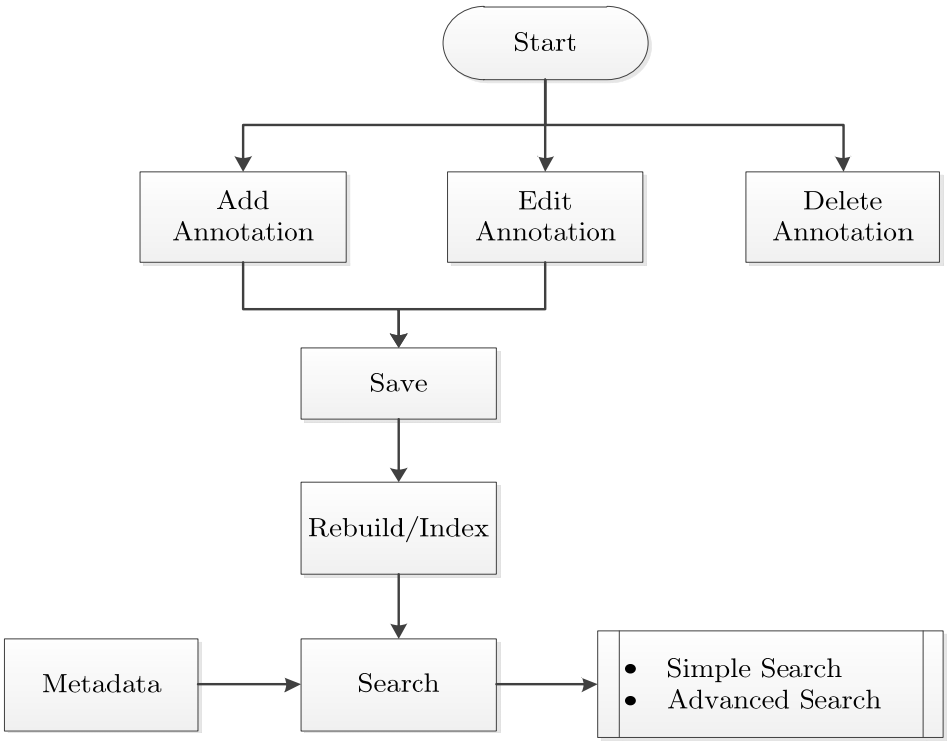


Figure 37: An annotator dataflow diagram

The annotator plugins that perform the various actions such as tagging, saving and authentication are listed below.

Tags Plugin:

This plugin allows the user to tag their annotations with keywords. The tags plugin uses two elements, `VIEWER` to add a section to a viewed annotation displaying any tags that have been added, and `EDITOR` which adds an input field to the editor allowing the user to enter a space separated a list of tags. This plugin can be initialised by calling `.annotator('addPlugin', 'Tags')`.

Store Plugin:

The store plugin sends annotations to the server to be saved in the database.

The following actions are performed by the store plugin

read: This GETs all annotations and is initialised by calling `.loadAnnotations()`. On receiving this request the Server returns an array of annotations serialised as JSON.

create: This POSTs an annotation to the server and is called when the annotator invokes the event `.annotationCreated()`. The annotation is updated with the data and saved in the database.

update: This PUTs an annotation on the server and is called when the annotator invokes the `.annotationUpdated()` event.

destroy: This plugin issues a DELETE request to the server for the respective annotation.

search: This calls the search function which is integrated into the Greenstone digital library and gets indexed automatically or when a rebuild is triggered.

Auth Plugin:

The Auth plugin compliments the Store Plugin by providing authentication for all requests. The plugin works by requesting an authentication token from the server and then provides this in all requests to the store.

The `authToken` comprises five parts:

`consumerKey`: the key issued by the administrator

`userId`: the unique identifier of the User and the browser

`authTokenIssueTime`: the formatted timestamp when the token was issued

`authTokenTTL`: the length of time in seconds for which the token is valid

`authToken`: the final auth token itself

The Pei Jones collection uses a customized Auth token setting which is stored in the database. The implementation of the token file is shown below. Figure 38 illustrates the output of the Annotator plugin after integration into the Pei Jones collection.

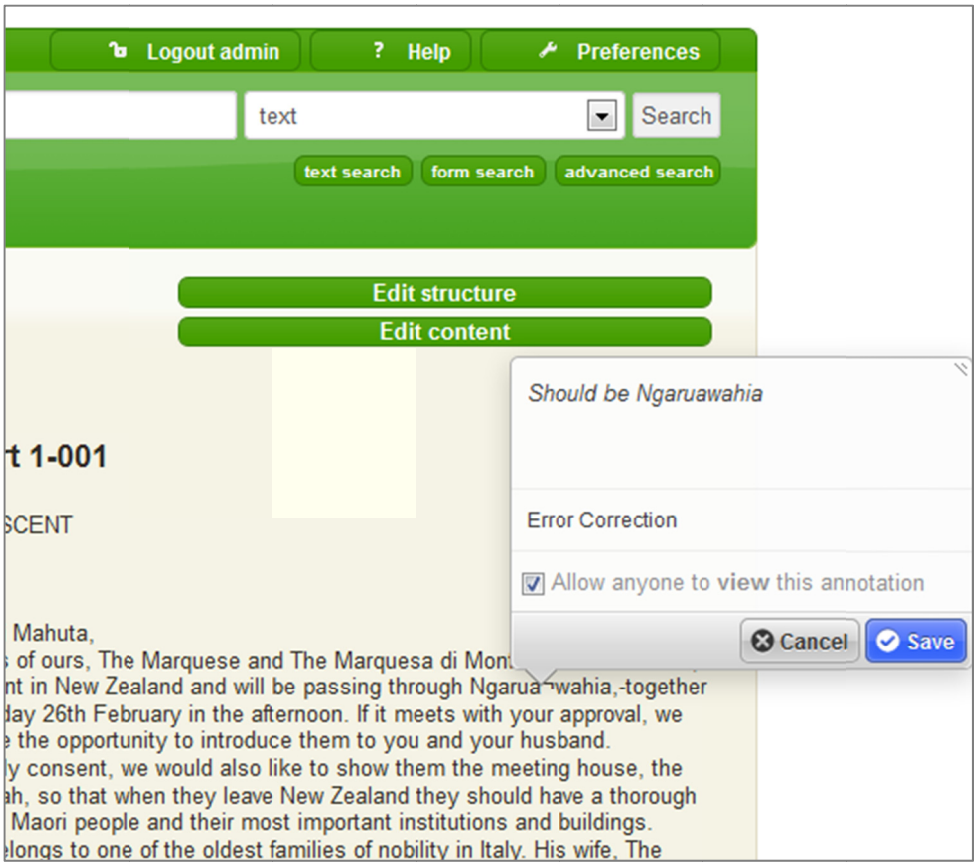


Figure 38: The output of the Annotator plugin

4.6. Image Annotation

Image Annotation[33] integrated into Greenstone uses a process of assigning metadata elements for captioning and tagging keywords to specific sections of an image. This implementation is a collection of JavaScript functions for annotating images.

A new photo annotation can be triggered on the web page by clicking on the "New photo annotation" button. When this button is clicked the function `AddNote()` is evoked. This function also defines

[33] http://css-tricks.com/examples/AnythingZoomer/text.php

the (default) start position via coordinates specified in the script and calls the respective function like SAVE, DELETE in response to the user's interaction with the post-it style widget that appears..

When the annotation has been entered clicking on the SAVE button calls the function `saveFunction()`. The saving of data occurs in three locations, namely the Index, Archive and Import with the `note.id` as the key field. The three parameters passed to the server are `metaname`, `metavalue` and `metapos`.

The saved annotations can be viewed by rolling over the mouse on the image. When the rollover annotation is displayed, clicking on it also brings up the controls to Save/Cancel/Delete the image annotation. An existing or new note can be deleted by clicking on the DELETE button which calls the function `deleteFunction(note)`. This removes the instance of the note from the Index, Archive and Import locations and finally synchronises the page with the server. Figure 39 illustrates an output of the Image annotator plugin after integration into the Pei Jones collection.

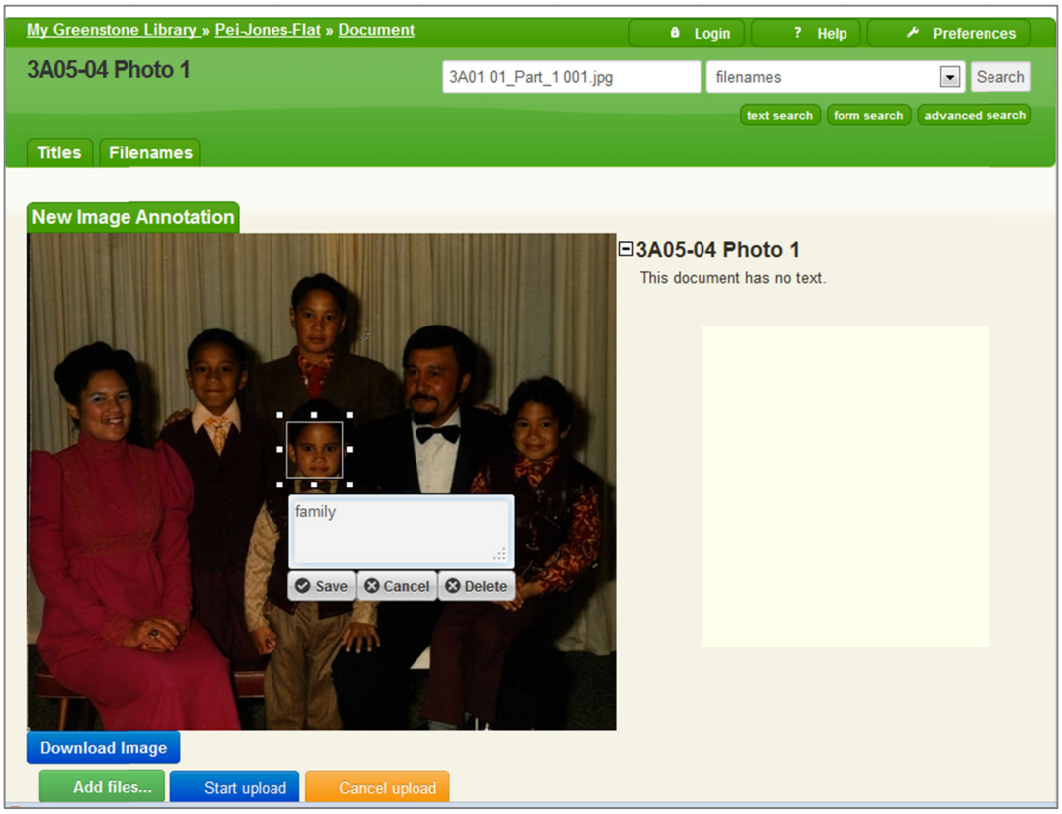


Figure 39: The output of Image annotation

4.7. Adding and searching for Text and Image metadata

The requirements for a workflow based around text and image annotations were primarily to address issues of ambiguity in the content. This may be noticed and corrected during the translation or transliteration stages by the librarians of content such as handwritten letters and photographs.. For example, in the document shown in Figure 40, the user has submitted a clarification regarding the ambiguity in the location names of Rotorua and Lake Rotorua.

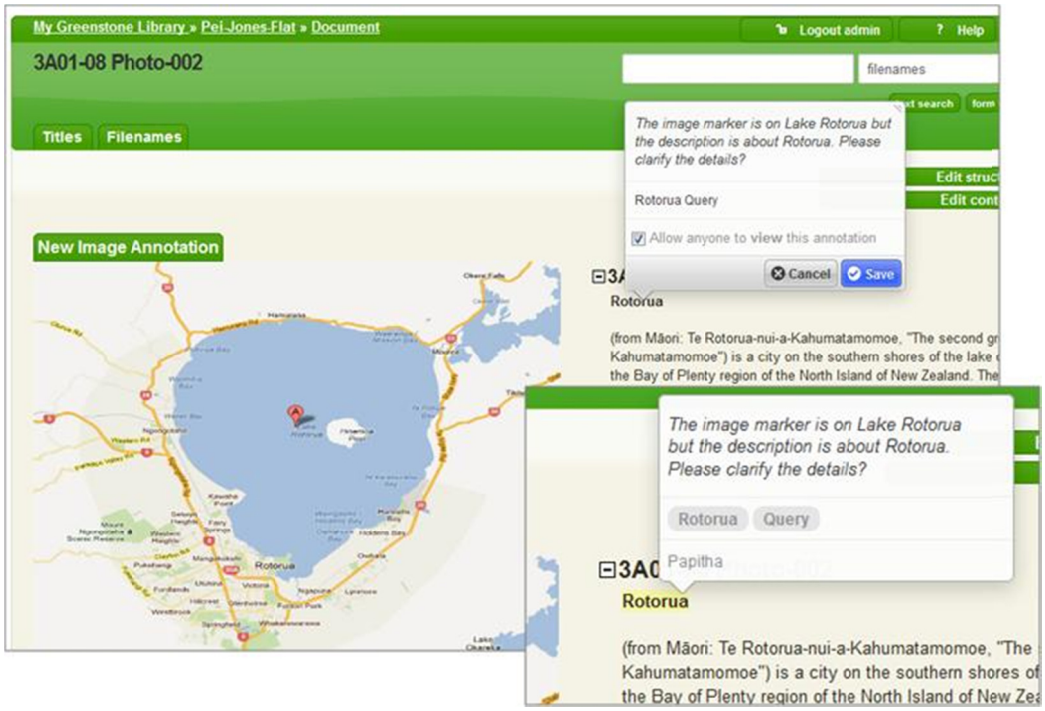


Figure 40: Example of annotation supporting the workflow

The Pei Jones collection includes a new metadata set which has been created using the Enrich view from GLI. The interface can be customized by name, namespace and description. This set inherits and propagates this across all its elements. Elements and sub-elements can also be specified. The lists are assigned metadata sets and can be edited or removed. Figure 41 shows the process of adding custom metadata to the Pei Jones collection.

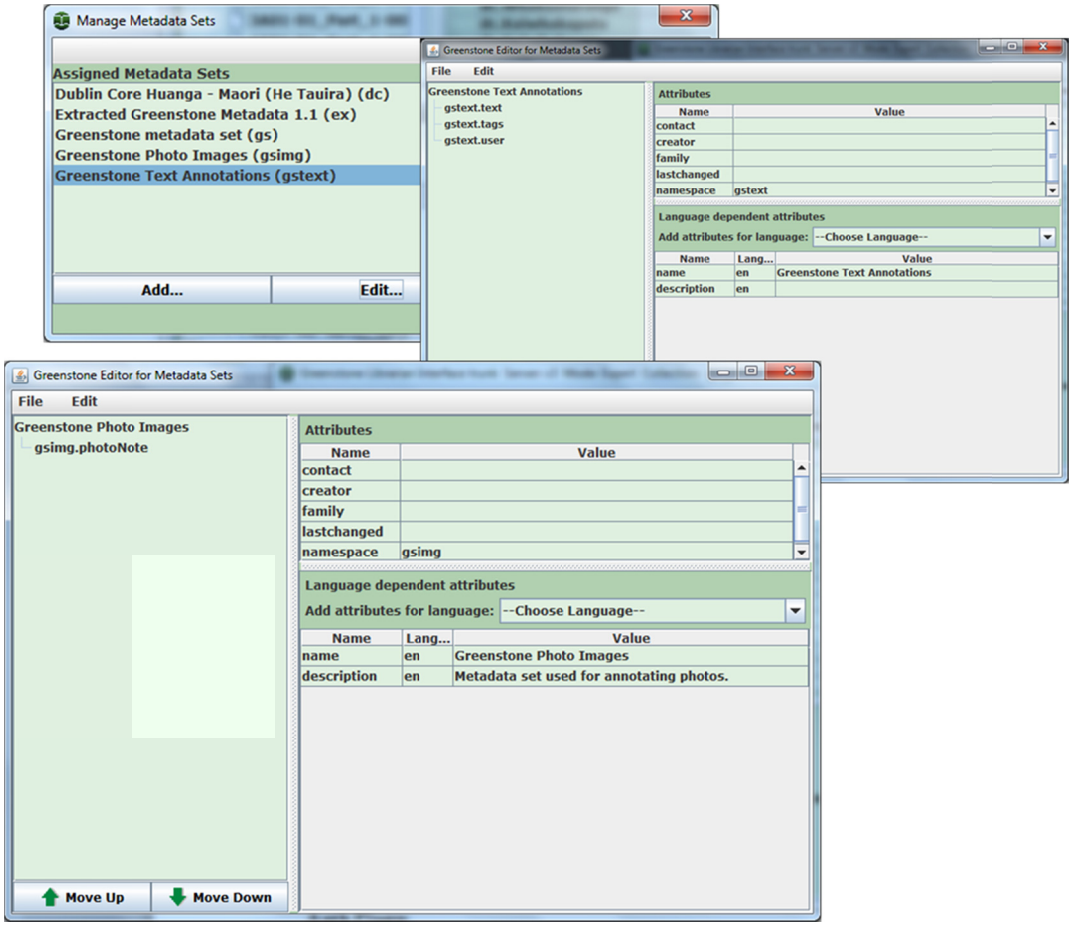


Figure 41: Adding custom Metadata using GLI

Building the collection creates two metadata files, namely `gstext.mds` and `gsimg.mds` in the Pei Jones metadata folder. This metadata file contains attributes like text, tags and user. The `indexList` such as `gstext.tags`, `gstext.text`, `gstext.user` are created within the `buildconfig.xml` file as listed below. Once the metadata is created the annotation's text, tags and user name are listed under the search filter. When the annotation is saved the collection needs to be rebuilt for indexing to enable the new content to be searchable.

The search functionality in Greenstone has the ability to search the contents of documents in a collection. Apart from the normal search, one of the key requirements was the ability to search within text and photo annotations. Searches can be performed using either the Simple Search or the Advanced Search feature. In order to enable search within annotated text and images all that was needed was the inclusion of appropriate (index) elements within the collection configuration file (see Section 4.2). This was done using GLI.

The Pei Jones collection uses MG++[34] a successor to MG "Managing Gigabytes" [11] developed to be more efficient at representing multiple indexes through word level indexes, with the added benefit of fielded and proximity based searching. The Pei Jones collection has a purpose built form based search interface which can be accessed from the preferences page of the collection. Fielded searched can be performed on any specified metadata element.

An example of the query operators supported are:

Boolean operators:

`& AND | OR ! NOT`, which can be encased in `()` for grouping and precedence

[34] http://www.greenstone.org/manuals/mgpp_user.pdf

Term modifiers:

```
#i    - case insensitive
#c    - case sensitive
#u    - unstemmed
#s    - stemmed
```

Proximity searching:

Searching for a phrase uses a strict proximity based method for matching the terms. The phrase must contain the query terms in the same order.

The maximum distance between the two query terms is specified using `NEARx`. Based on this predefined value a positive profile match is returned.

Fielded searching:

`[terms]:Field` specifies the particular field within which the search must be performed. This can also be a metadata element from the document which is to be searched. The metadata element in the collection is represented by the field name.

Figure 42 shows a sample search for a text annotation within the Pei Jones collection. A search for all annotations made by a specific user containing the selected text returns the result which takes the user to the page which has that specific annotation.

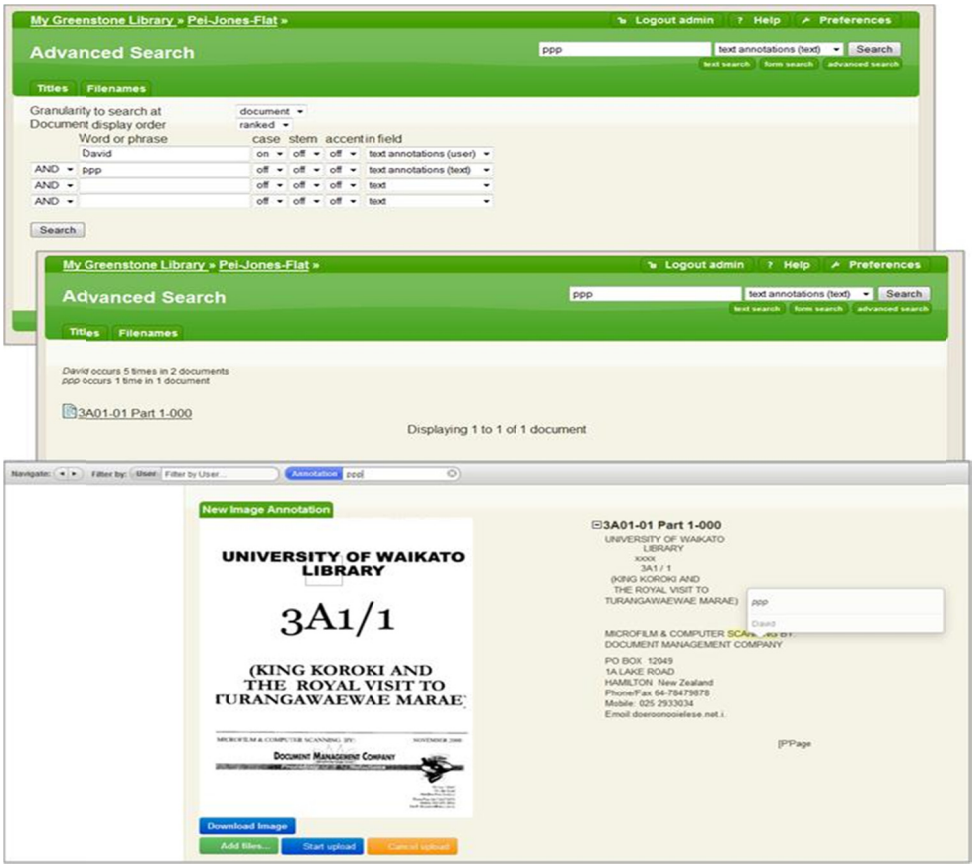


Figure 42: A Text Annotation Search within Greenstone

4.8. Browse, View and Edit content

Documents can be located in the Pei Jones collection either using browse by titles or filenames. The document can then be identified and viewed by navigating through the list and selecting the appropriate item. The original image and OCR text are displayed side by side. This layout has been used to facilitate the simplified editing process by allowing the curators to view the

image and the text on the same page. This enables the curators to compare and validate the content without having to navigate to different pages or open multiple pages. Figure 43 illustrates the browsing by filename to locate the documents in the Pei Jones collection and the side-by-side layout of image and text.

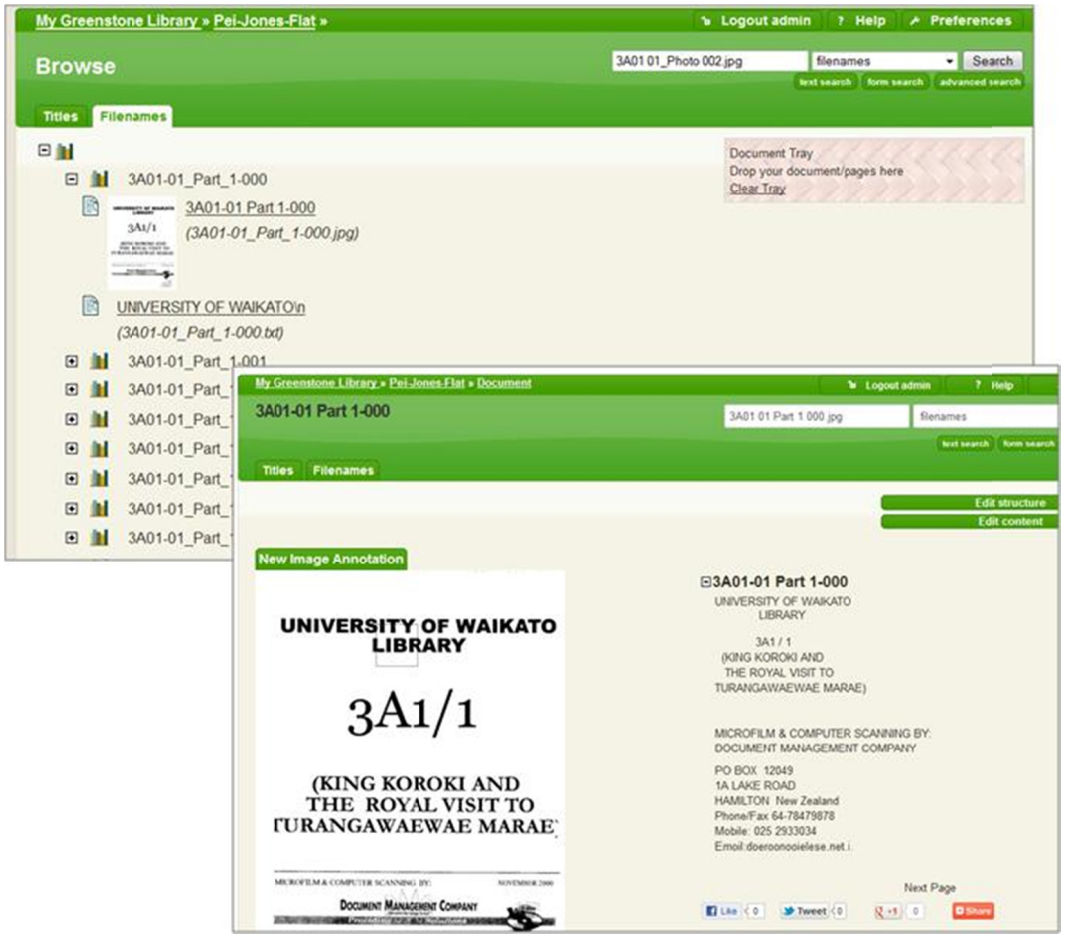


Figure 43: Browse by filename and the layout of output

The Pei Jones collection utilizes the integrated editing framework called Seaweed [9] (Seamless Web Editing) described in Section 3.2.4. The content editor can be activated using the edit content button and modifications can be made directly on the page. The edit structure button activates the structure edit mode where the metadata can the added, modified or deleted and the structure of the document can be modified.

The framework uses WYSIWYG in-place editing features. The document is split into editable sections which can be modified using a web browser. The elements can be categorized into five types: Zones, Boxes, Fields, Images and Links. Editable text areas which contain formatting are called Fields. A combination of links, images and fields are called boxes and are represented as a HTML template. A zone can contain within it a combination of zones and boxes. These zones can be re-arranged and inner boxes can be created and deleted. The main advantage of using this framework with AJAX is to save the content asynchronously; avoiding issues related to page reload. Figure 44 below illustrates the workflow of editing the content and metadata dynamically in the Pei Jones collection.

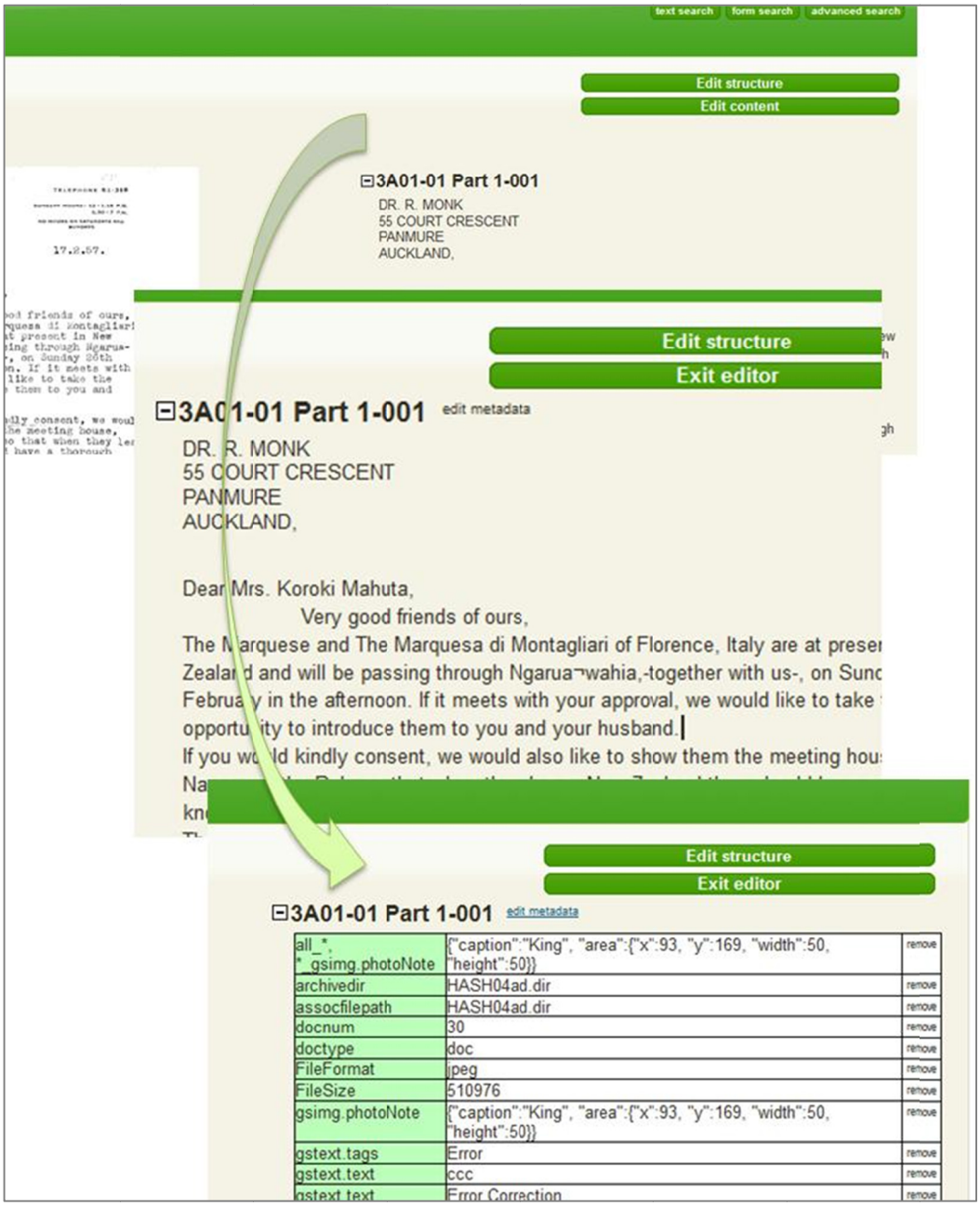


Figure 44: Editing content dynamically on the Pei Jones collection

4.9. Image upload and download process

The image upload and download process implemented in the Pei Jones collection uses the web-browser activated ingest process as described in Section 3.2.2. This ingest process uses the jQuery File Upload Plugin [35] which consists of a basic version of `jquery.fileupload.js` providing the File Upload API and `jquery.fileupload-ui.js` providing a complete user interface. This plugin uses the function `jQuery.ajax()` to perform the file upload requests.

The main parameters passed to the plugin are `url`, `type` and `dataType` as described below.

url:

This is the string which contains the URL to which the request is sent.

type:

This is the method of HTTP requests for the file uploads. This can either be "POST" or "PUT."

dataType:

[35] https://github.com/blueimp/jQuery-File-Upload

This is the type of data that the server sends back to the client. This is by default set to “json”

The integration of this plugin into the Pei Jones collection enables users to download an image, make changes or edits as required locally then upload the file using the browser. The image is uploaded asynchronously in the background using AJAX. The update can be activated using the browser button or drag and drop function.

Figure 45 illustrates the workflow of uploading and downloading images from the web browser. The image was downloaded and saved to a local folder as shown in the first image. The image was then enhanced using 3rd party software and uploaded to the web browser using drag-and-drop.

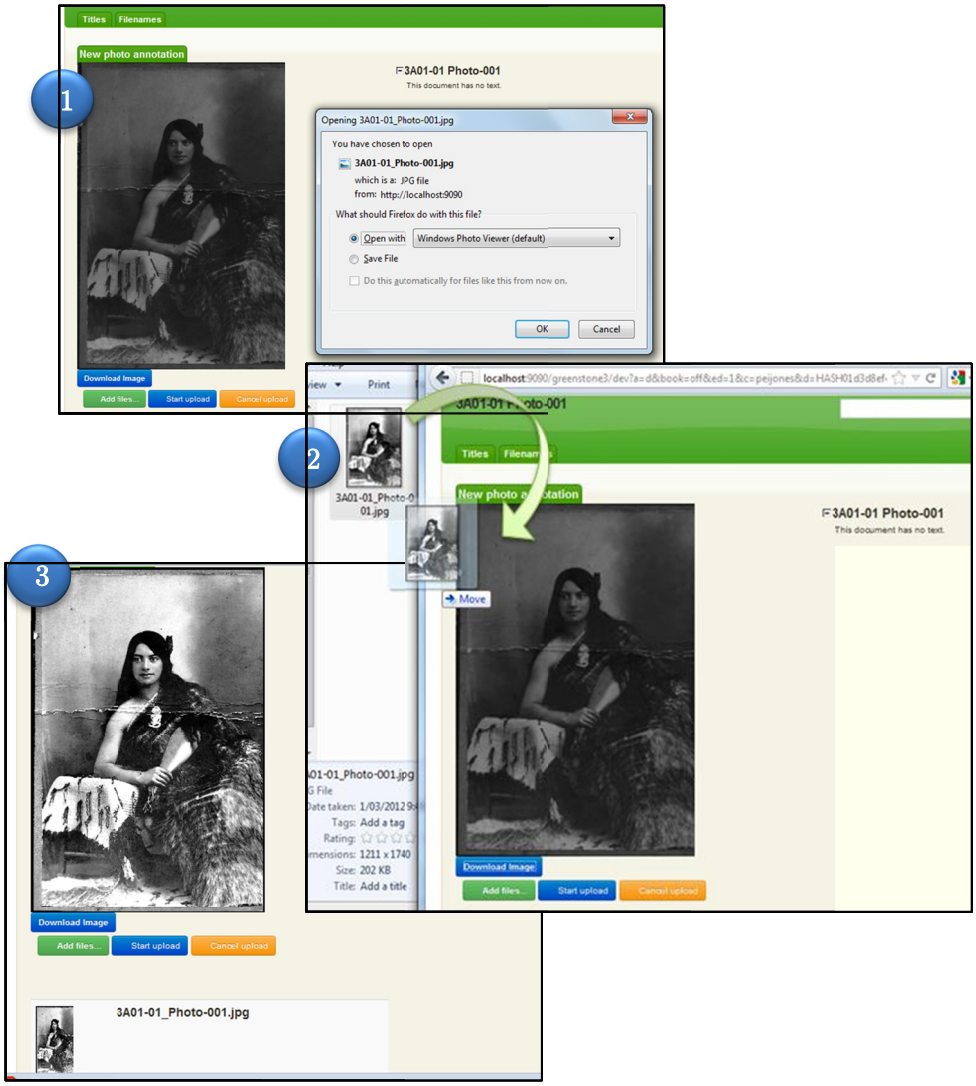


Figure 45: Image upload and download process

Chapter 5: Expert Evaluation

5.1. Introduction

In this chapter we cover the expert evaluation of the interactive user management system by the targeted digital librarians. This evaluation was aimed at assessing the functionality, users experience and identifying any problems with the system. The experts analysed the entire system from different perspectives and identified opportunities for improvement in the user interface and the inclusion of additional functionality.

The expert evaluation also aims to understand the system in its entirety and identify any potential bottlenecks and inconsistencies. The system should cater to different usage styles, easy navigation and the access of information in a logical manner. The system should be thoroughly checked for errors and provide audit logs of messages generated by the system.

After considering the target audience and the skills matrix of the curators, the Heuristic Evaluation [14] process was identified as the best solution to evaluate the user interface and workflows. Heuristic evaluation [14] is defined as the "usability engineering method" of identifying problems related to the design of the user

interface and its usability. This evaluation process is one of the most widely used methods of usability inspection due to the time to implement, cost and ease of us. The heuristic evaluation of the interactive user system was carried out by experts from the Computer Science and Māori departments.

Based on the guidelines for the classification of heuristics by Weinschenk and Barker and the Ten Usability Heuristics [36] by Jakob Nielsen [15] the following categories were studied and evaluated.

5.2. Visibility of System Status

Every time a document is edited or saved within the Pei Jones collection the indexing was done automatically in the background using the web interface. The only indication of this activity was the browser status, which showed that it was awaiting a response from the server. Based on the feedback, a notification mechanism was added into the system to give an indication of the system operation taking place at that instant.

The indexing process is triggered automatically when editing the structure, editing content and annotating an image. The status of the indexing process is displayed by notifying the user about the process. Figure 46 shows the indexing notification when editing the

[36] http://web.cs.wpi.edu/~kal/courses/hci/module6/neilsen10heuristics.htm

structure and Figure 47 shows the indexing notification when editing the content. Figure 48 shows the indexing notification which is displayed each time an image annotation is saved.

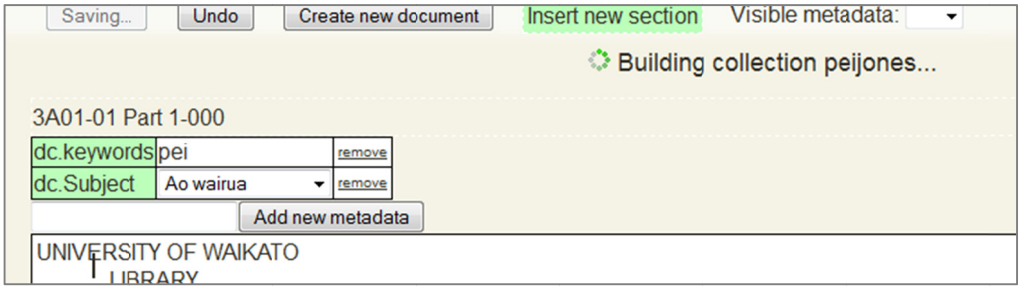


Figure 46: Build notification when editing the structure

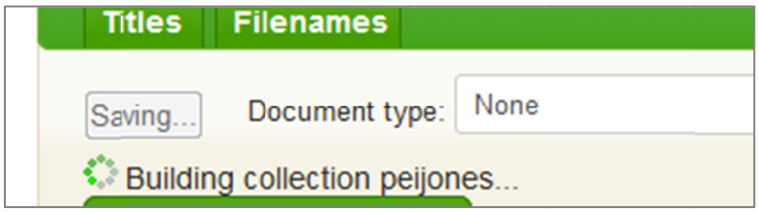


Figure 47: Build notification when editing the content

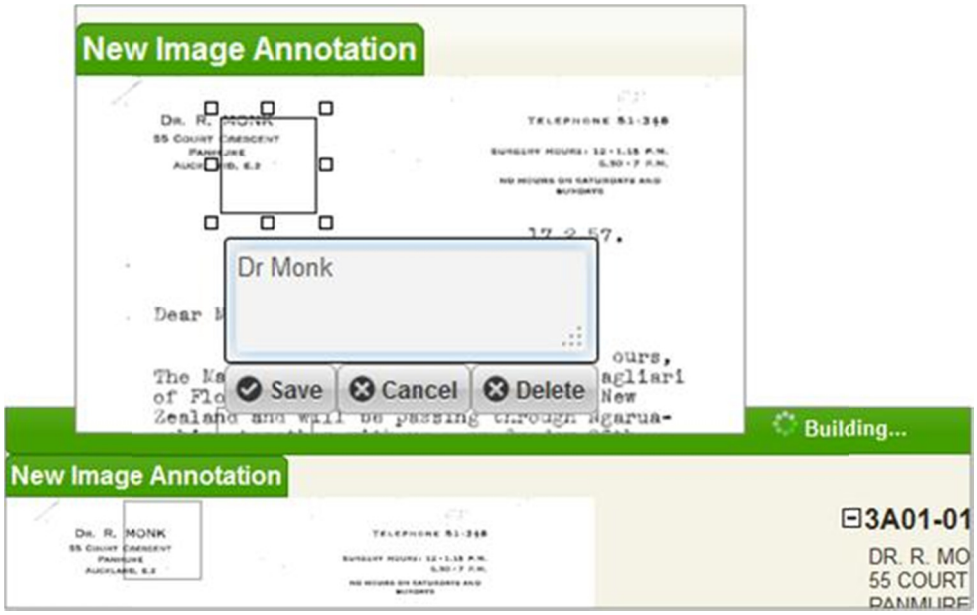


Figure 48: Build notification when saving an image annotation

Since the text annotation uses customized module, when a text annotation is saved to a document, the user is prompted with a message to rebuild the index. This module is designed in this manner so that the user can add multiple annotations on each page and when all annotations are completed the index can be rebuilt. On initiating the build, the user is notified about the rebuilding status as shown in Figure 49.

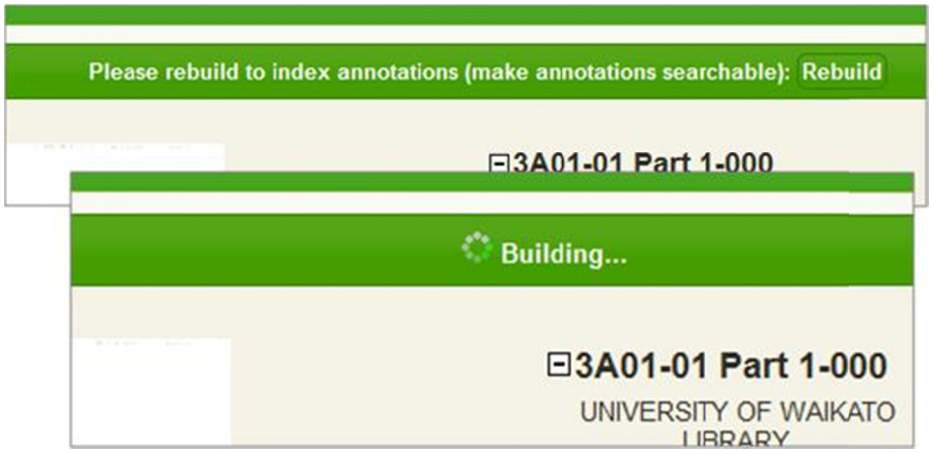


Figure 49: Build notification when saving a text annotation

5.3. User Control and Freedom

The editing of content is predominantly used by librarians who are familiar with word processing applications. Hence, providing the librarian with the flexibility to revert back to a previously edited state was considered a key feature. The undo feature has been implemented in two primary areas which are content undo and structure undo.

Content undo:

The undo feature during content editing is enabled using the edit action model provided by Seaweed. The undo editing action is executed by the undo manager which maintains a history of all transactions on the page. This history consists of a sequence of actions which are split into a series of operations and governed by

the operations manager. The undo action can be triggered by the windows undo shortcut command `Ctrl+Z`. Figure 50 illustrates the undo action as performed on the browser to revert text changes.

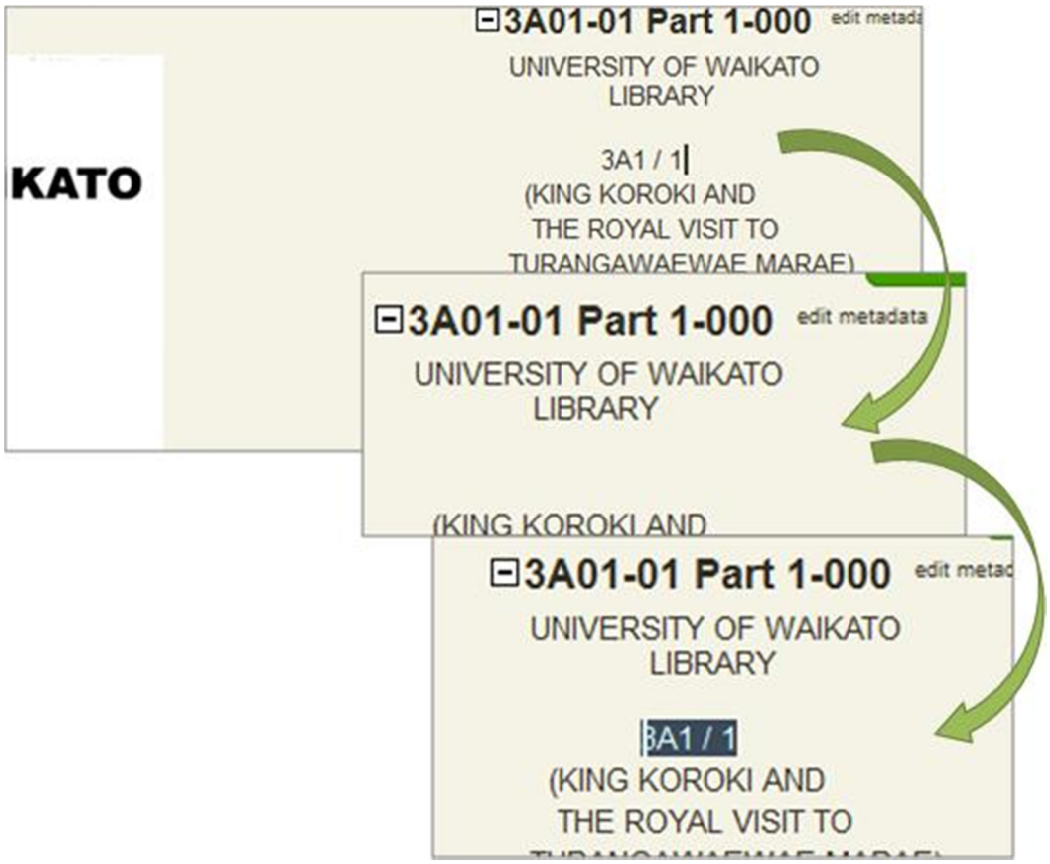


Figure 50: Content undo using Ctrl+Z

Structure undo:

Editing the structure involves adding, removing and editing metadata and structure. Prior to the user feedback, making changes to the structure was a rigid process.

The structure undo maintains a history of the actions from the time the page was rendered. By triggering the undo action the page reverts back to the initially rendered structure. The undo action can be activated by clicking on the undo button from the structure edit section. Figure 51 illustrates the undo action as performed on the browser to revert structural changes.

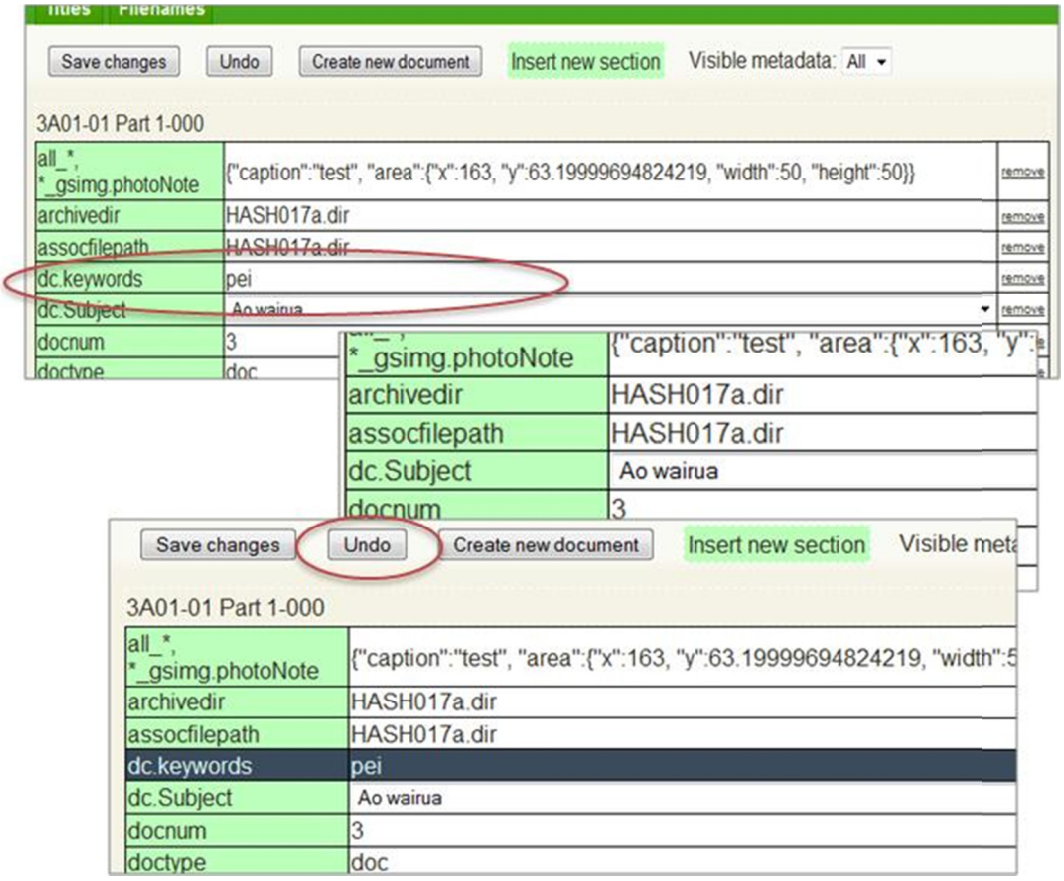


Figure 51: Structure undo activated on button click

5.4. Aesthetic Integrity

Evaluation of aesthetic integrity aimed to test the functionality and usability of the design while maintaining a unified standard. The user feedback regarding aesthetic integrity mandated that the collection should be visually appealing and consistently maintain the visual design principles. The proportions and alignment of the elements need to be clearly defined. This is vital since it can determine the users attitude and response to the collection.

Using standardised Cascading style sheets (CSS) the positioning, layout, font, colours and style of the entire web site is maintained. This separates the content from the HTML elements to maintain a coherent and well-structured web site with a standardised layout and navigation. Elements on the web page such as buttons and input boxes (which are used for specific operations) have also been standardised using CSS.

Figure 52 shows the Pei Jones collection with a consistent design model maintained across the web page. The look and feel of the buttons and navigation are also consistent across the collection.

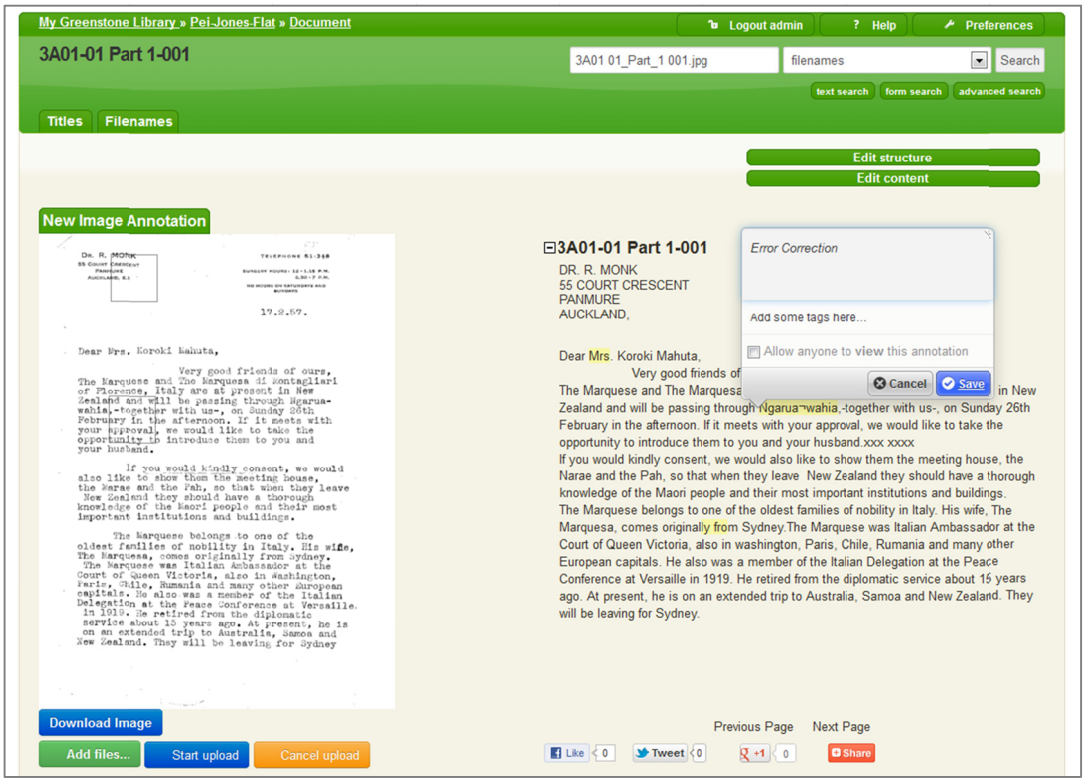


Figure 52: Aesthetic integrity maintained using CSS

5.5. Flexibility and efficiency of use

With frequent use of the text annotation facility in the collection, multiple annotations created by different users may appear on a single page which may lead to difficulty in searching for specific annotations. Based on the feedback received a filter plugin was included with the annotations to enable the user to navigate and filter specific annotations on a page.

This filter plugin adds a browser-like toolbar to the top of the page which contains the available filters that can be applied to the

current annotations. The toolbar has three main components, namely: navigation, filter by user and filter by annotation. When the user name is entered in the filter by field, the annotator only highlights annotations created by that user. The navigation arrows can be used to navigate sequentially from one annotation to another while highlighting the current annotation with a different colour. Filtering by user name and annotation only highlights the annotations which match both conditions. Figure 55 shows the filter plugin toolbar used in the Pei Jones collection.

Figure 53: Filter plugin toolbar for text annotations

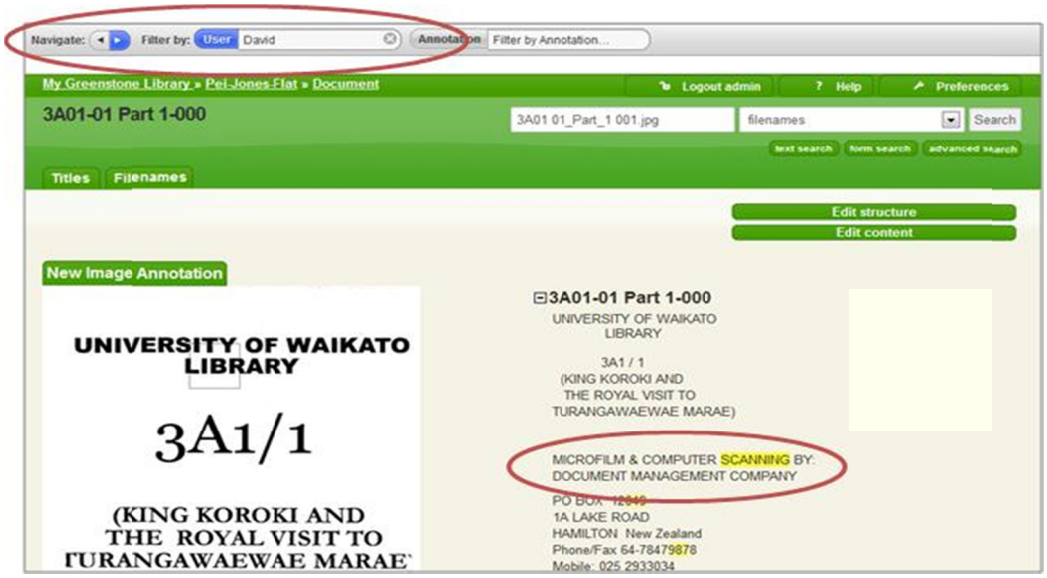


Figure 54: Using filter by user for text annotations

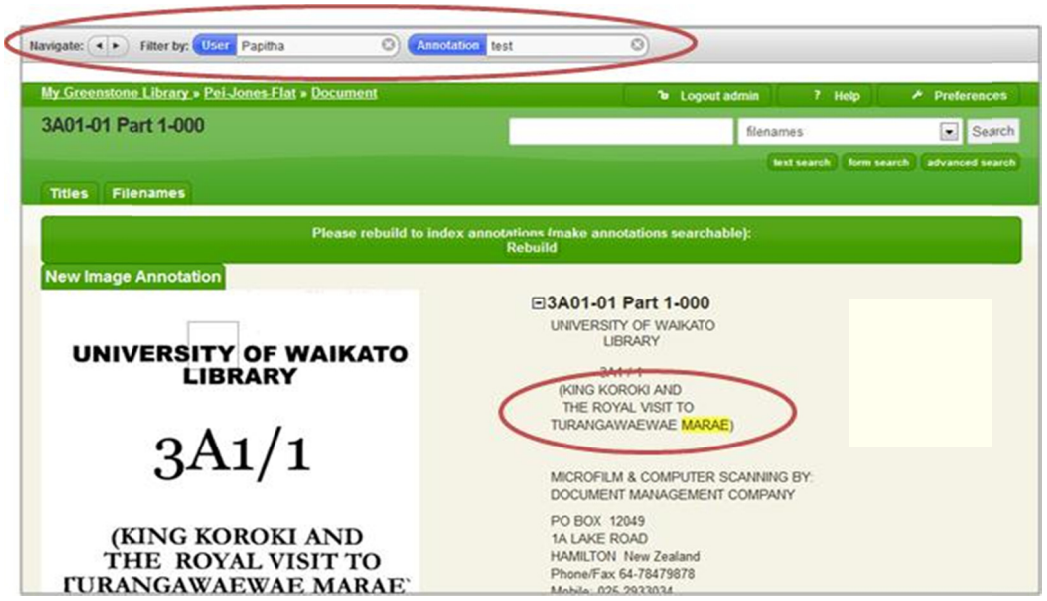


Figure 55: Using filter by user and filter by annotations

The expert evaluation of the interactive user management system assessed the functionality and users experience. Additional

functionalities such as visibility of system status, user control, aesthetic integrity, flexibility and efficiency were incorporated into the system based on the feedback received from the librarians.

Chapter 6: Conclusion

In this chapter we conclude this thesis with a discussion of how a general purpose DL has provided a flexible and cohesive environment through a single unified web based interface supporting management and end-user roles to access and share information simultaneously.

6.1. Contributions

Based on a detailed study of the user requirements, a customized framework has been developed that supports batch driven and web based ingest of content, a streamlined process to create, edit and delete documents from a web browser, improved accessibility and a workflow control for document creation and editing using annotations.

The end result brings the concept of sharing and collaboration from the social media spectrum into the digital library realm. The option of annotating provides a centralized platform where information can be shared between users and librarians as well as among each other's groups.

This newly developed system provides a static and dynamic method of editing the structure and content. The simplified user

interface and its similarity to common word processing applications means a shorter learning curve for users which in turn results in quicker adoption and an improved user experience.

The full text and metadata search capability provided by Greenstone with the addition of searching within annotations ensures that all the information can be easily located and accessed by the user irrespective of which part of the system it is stored in.

Specific to the needs of the Pei Jones collection, image content was stored in a structured hierarchical format within the system engineered in such a way that it could be easily accessed from the web interface or from the local file system. The web interface provides the user the ability to easily download (optionally adjust the image with locally installed applications) and upload images with a minimal number of clicks or by using common windows functions such as drag and drop.

The legibility of the image content is ensured with the option of dynamically zooming into specific sections of the image. This provides a quick method of viewing the detailed information without leaving the page or waiting to load a detailed image.

This developed work built on the Greenstone digital library platform and has provided a means for the scholars and librarians to manage and publish information easily using a common web based platform.

6.2. Future Work

The Pei Jones case study provides a useful blue-print for other collections of personal artefacts. The following functionality would be useful extensions to the developed work.

Approval based workflow System – Building a workflow based system developed on hierarchical approval will provide an ideal platform for vetting content before publically publishing. This will ensure accuracy and accountability of the content being published.

Audio and Video content with annotation – Development of a multimedia hosting platform will enable the preservation of audio and video data digitally. The inclusion of multimedia annotations will add variety to the collection and make it more meaningful to the user.

Bibliography

[1] T. Reese and K. Banerjee, Building Digital Libraries: A How-to-do-it Manual, vol. 153, K. Banerjee, Ed., Ann Arbor, Michigan: Neal Schuman Publishers, 2007, pp. 128-129.

[2] R. Kahn and RobertWilensky, "A framework for distributed digital object services," *International Journal on Digital Libraries,* vol. 6, no. 2, pp. 115-123, 2006.

[3] "Getting started with Fedora," 2011. [Online]. Available: https://wiki.duraspace.org/display/FEDORA/Home.

[4] I. Schommer and S. Broschart, SilverStripe: The Complete Guide to CMS Development, 1 ed., Auckland: John Wiley & Sons, 2010.

[5] I. H. Witten, D. Bainbridge and D. M. Nichols, How to Build a Digital Library: The Morgan Kaufmann Series in Multimedia Information and Systems, 2nd ed., Hamilton: Morgan Kaufmann, 2009.

[6] D. Bainbridge, K. J. Don and I. H. Witten, "The design of Greenstone 3: An agent based dynamic digital library," [Online].

Available: http://www.greenstone.org/manuals/gs3design.pdf.

[7] E. H. Durfee, D. L. Kiskis and W. P. Birmingham, "The agent architecture of the University of Michigan digital library," *IEEE Software Engineering,* 1997.

[8] K. Don, "Greenstone3 : A modular digital library," University of Waikato, 2010. [Online]. Available: http://www.greenstone.org/docs/greenstone3/manual.pdf.

[9] D. Bainbridge and B. J. Novak, Seamless Web Editing for Curated Content, Glasgow, UK: Research and Advanced Technology for Digital Libraries:14th European Conference, ECDL 2010, 2004, pp. 168-175.

[10] S. Serafin, "Writing Invention: Annotating a Text," 2004. [Online]. Available: http://rwc.hunter.cuny.edu/reading-writing/on-line/annotating-a-text.pdf.

[11] I. H. Witten, A. Moffat and T. C. Bell, Managing Gigabytes: Compressing and Indexing Documents and Images, 2nd ed., San Francisco: Morgan Kaufmann, 1999.

[12] W. Cavnar and J. M. Trenkle, "N-Gram-Based Text Categorization," in *SDAIR-94, 3rd Annual Symposium on*

Document Analysis and Information Retrieval, Las Vegas, 1994.

[13] T. Gottron and N. Lipka, "A Comparison of Language Identification Approaches," http://citeseerx.ist.psu.edu/viewdoc/ summary?doi=10.1.1.155.5553 2009. [Online].

[14] J. Nielsen and R. Molich, "Human factors in computing systems, pp. 249-256," in *CHI '90 Conference proceedings*, Seattle, April 1-5, 1990.

[15] J. Nielsen and R. L. Mack, Usability Inspection Methods, pp. 40-95, University of Michigan: Wiley, 1994.

[16] A. Dix, J. Finlay, G. D. Abowd and R. Beale, Human-Computer Interaction, Harlow: Prentice-Hall Europe, 1993.