A New Framework for Building Digital Library Collections

George Buchanan
University College London Interaction Centre
31/32 Alfred Place
London, United Kingdom
g.buchanan@cs.ucl.ac.uk

David Bainbridge, Katherine J. Don and Ian H. Witten
University of Waikato
Hillcrest Road
Hamilton, New Zealand
davidb,kjdon,ihw@cs.waikato.ac.nz

ABSTRACT

This paper introduces a new framework for building digital library collections and contrasts it with existing systems. It describes a significant new step in the development of a widely-used open-source digital library system, Greenstone, which has evolved over many years. It is supported by a fresh implementation, which forced us to rethink the entire design rather than making incremental improvements. The redesign capitalizes on the best ideas from the existing system, which have been refined and developed to open new avenues through which digital librarians can tailor their collections. We demonstrate its flexibility by showing how digital library collections can be extended and altered to satisfy new requirements.

Categories and Subject Descriptors

H.3.7 [**Information storage and retrieval**]: Digital Libraries—*Systems issues*; H.3.7 [**Information storage and retrieval**]: Digital Libraries—*Collection*

Keywords: Digital Libraries, Architecture, Collection Building

1. INTRODUCTION

The trend towards increasingly open, flexible architectures can be traced in the development of digital library protocols [1]. The highly successful Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) has provided a simple base-line for metadata access, and subsequent work strives to base component-based, modular protocols upon it [10]. In a complementary realm, the emerging METS document framework [6] provides an open, extensible system for representing documents in digital repositories.

Set against the move towards standard protocols, today's digital library systems must confront an increasing range of document formats and media, architectural designs for browsing and classification, indexing requirements, and user interface techniques. This drives the demand for open architectures, but makes it hard to provide well-formed and supportive infrastructures that maximize reuse of individual components and the reliability of the system as a whole. This challenge is all the greater when a library system seeks to provide the option of full text indexation of documents.

Inspired by the spirit of initiatives such as OAI-PMH and METS, and motivated by the disparate uses of contemporary digital library systems, we have designed a new framework for building digital library collections, based on our own extensive and varied experience, and that of others [13]. The subject of this paper, we call the new framework "Greenstone 3" to distinguish it from the earlier system, "Greenstone 2". This framework is supported by a fresh implementation that is completely independent of the existing one. It capitalizes on the best ideas from the existing system, which we have further refined and developed to open new avenues through which collection designers can customize their digital library collections.

This paper proceeds as follows. To provide context to the work, we first give a brief background to the Greenstone project; then we introduce and describe the configuration structure and document representation used through our new building architecture. The third section describes the building process itself. This is followed by the model that we have adopted to ensure that the new architecture is extensible. Fifth, we compare this with the previous Greenstone 2 design and present an example use-case scenario, identifying the key advantages of the new open collection-building architecture. We conclude by relating the new architecture to existing collection-building systems.

2. BACKGROUND

The Greenstone digital library software provides a wide range of tools for building digital library collections [14]. Its origins can be traced back to 1998 when it was first used by the United Nations and other Non-Government Organizations. It runs on a wide range of operating systems, from a 486 PC running Windows 3.1 accessing the data and software from CD-ROM through to a modern workstation serving its collections over the Internet. Full installation instructions, documentation and the collection-designer interface is

JCDL'05, June 7–11, 2005, Denver, Colorado, USA.

```
<documenttypes>
  <document type="HTML">
  <document type="Text">
  <document type="MetadataXML">
</doctypes>

<search type="mg">
  <index name="dtx" level="document" field="text">
    <displayItem name="name" lang="en">entire documents</displayItem>
    <displayItem name="name" lang="fr">documents entiers</displayItem>
    <displayItem name="name" lang="es">documentos enteros</displayItem>
  </index>
  <index name="stx" level="section" field="text">
    <displayItem name="name" lang="en">chapters</displayItem>
    <displayItem name="name" lang="fr">chapitres</displayItem>
    <displayItem name="name" lang="es">capitulos</displayItem>
  </index>
  <index name="stt" level="section" field="title">
    <displayItem name="name" lang="en">section titles</displayItem>
    <displayItem name="name" lang="fr">titres des sections</displayItem>
    <displayItem name="name" lang="es">titulos de las secciones</displayItem>
  </index>
  <format>
    <gsf:template match="documentNode"><td valign="top"><gsf:link><gsf:icon/></gsf:link>
    </td><td><gsf:metadata name="Title"/></td></gsf:template>
  </format>
</search>

<browse>
  <classifier name="CLSubject" type="Hierarchy" file="subject.xml" field="Subject"/>
  <classifier name="CLTitle" type="AZList" />
  <classifier name="CLOrganization" type="Hierarchy" file="organization.xml"
              field="Organization"/>
  <classifier name="CLKeyword" type="List" file="howto.xml" field="HowTo">
    <displayItem name="name" lang="en">HowTo</displayItem>
  </classifier>
</browse>
```

Figure 1: A collection configuration file

available in English, French, Russian and Spanish; the end-user reader interface is in over 35 languages.

The software supports full text indexing of documents, from which a number of ways to assist reader access is levered, including: date extraction, abbreviation expansion, author identification, keyphrase assignment and document summarization. This has been found to be a boon over the years, leading us to the conclusion that a comprehensive building framework should support and enable full-text processing during the build cycle to enable a greater support of readers and easier experimentation with such features. However, since the original design, several pertinent new open standards have emerged and a key objective is to incorporate them into the new design.

3. COLLECTION CONFIGURATION AND DOCUMENT REPRESENTATION

Before describing the process by which collections are built, we will show how the collection-building process is configured, and how documents are represented internally within the system. In [15] we described our "plugin" mechanism and showed how it provides great flexibility for such varied tasks as ingesting both document files and metadata files, extracting text and metadata from different file formats, and expanding compressed files. It has proved so successful that we have adopted it in the new framework. However, we depart radically from our earlier system by using the METS framework (which postdates that system) throughout for internal document representation.

3.1 Configuration

Collections are designed individually, and the structure of a collection is encapsulated in an XML file called the "collection configuration file." Its contents include build-time configuration options such as:

- The document types that the collection should recognize
- The metadata access structures or "classifiers" that are to be provided for users to browse the collection, such as by Titles A–Z
- The full-text indexes to build for searching the collection.

The configuration file also contains run-time information about the collections, for example display options.

The bulk of an example configuration file can be seen in Fig. 1—all that is omitted is the part concerning the purely runtime configuration. It defines a simple collection of HTML pages and plain text files, along with XML files that define metadata that is to be associated with these documents. A document may consist of several separate files—for example, in this collection any images associated with the HTML pages will be contained within the HTML document rather than being identified as documents in their own right. Or a document may comprise complementary pairs of files—like a Word file and matching PDF version.

The *search* block in Fig. 1 defines the full-text-searchable indexes that will be built. In this case the MG indexer[16]

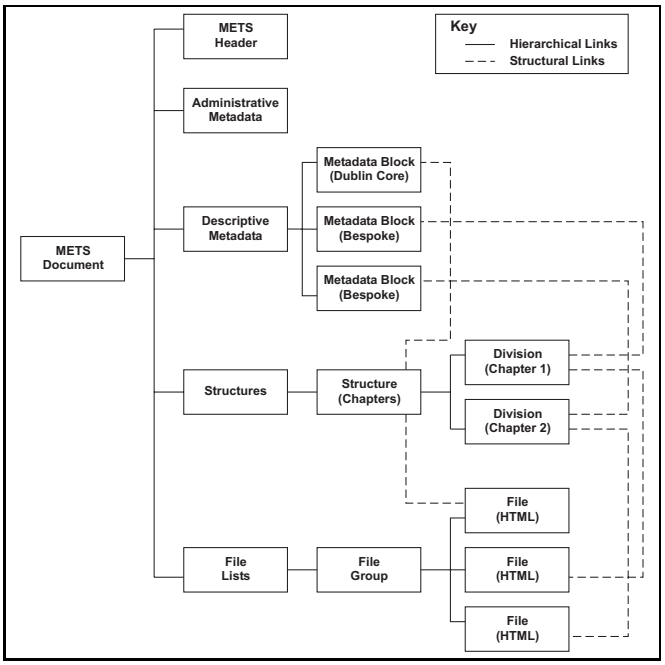

Figure 2: An example METS Document

will be used to build document-level and section-level indexes of the main text, and a section-level index of *Title* metadata. The *browse* block defines the metadata-based browsing mechanisms through which users can access the collection. This configuration file specifies a hierarchical subject browser by defining a Hierarchy-type classifier of each document's *Subject* metadata, and the file that defines that hierarchy is named. Other browsing classifiers are also defined. Run-time configuration items can be seen in both the classification and indexation parts of the file. For example. the "displayItem" nodes give names for the indexes (in English, French and Spanish) and for one of the browsers.

The Greenstone 3 building program reads this file and configures the components to be used by the collection-building process. Many default options are implied. For instance, this collection would automatically expand Zip files—because the Zip expansion plugin is enabled by default. The process followed during building will be described later.

3.2 METS and Document Representation

We were eager to adopt an open, standard form for modeling the content and metadata of a document. The METS framework supports a free selection of metadata formats and avoids duplicating data that is already stored in constituent files [6]. Using this standard, open framework should facilitate interoperation with other digital library systems. Fig. 2 illustrates a simple METS document consisting of two chapters, each stored in an HTML file, linked by a central HTML file. Some bespoke metadata is kept on each chapter, whilst the document-level metadata is stored in Dublin Core format.

METS documents contain up to seven sections, of which five are illustrated in Fig. 2. These are the METS Header, Administrative Metadata, Descriptive Metadata, File lists and Structural lists. The structural content is the only required section, and the content of the remaining sections is mapped onto the encoded structure. For the sake of clarity, this example shows only the connections between sections and the descriptive metadata and files.

A document is encoded into METS form by a plugin that corresponds to the type of the document. Each such plugin extracts any appropriate metadata from the document files (e.g. title), catalogs the files that comprise the document, and extracts the document text in subsequent phases (extraction and indexation; see Section 4). In other words, it encapsulates the document content and acts as a standard gateway to its METS representation. This means that other phases of the building process are able to access data without having to understand the particulars of each file format—even if the data is stored as a binary object.

Document text can be provided in one of three ways, as the processing component requires. Plain Unicode is a simple baseline recognized throughout the system. At a higher level of sophistication, an XML representation is supported. Finally, indexers can use the underlying binary files if they so desire. A complete document plugin provides all these formats, though the XML representation is optional. In any case, a document may have no textual content whatsoever (in which case all forms are empty). For example, a PostScript file can be translated into an XML format that contains the document text, identifies the title and also preserves some of the original presentation. The PostScript plugin allows a plain-text reduction of this, but if an indexer wishes to treat PostScript files in a special way, it can access the original files directly.

METS documents are partitioned into several components, the most common elements being structure, files, administrative metadata and descriptive metadata. A given document may form part of a single file, or be an agglomeration of individual files and parts of files. Metadata can be associated with a document as a whole or with any part of

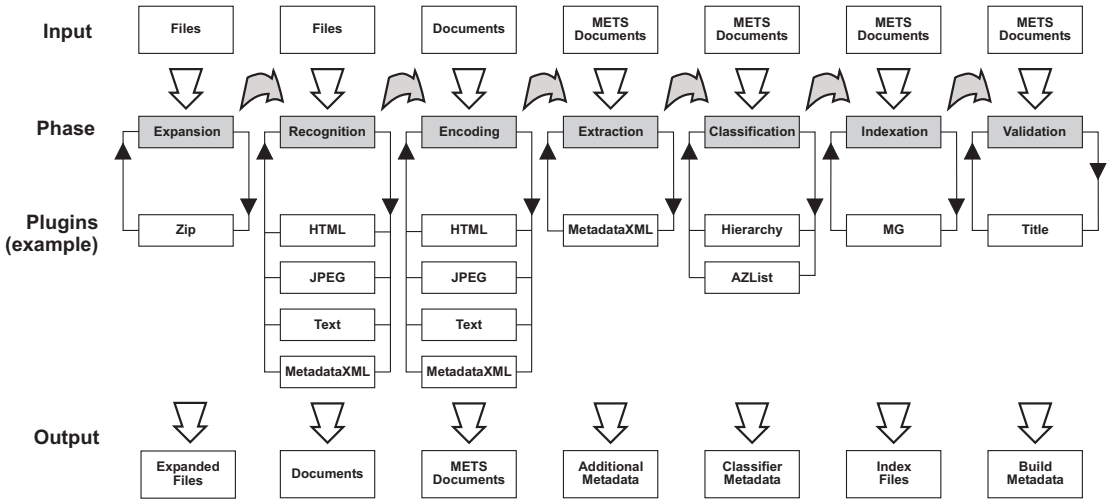

Figure 3: The Greenstone 3 Building Architecture

its structure. A document may also have more than one structural form. For instance, it may have both a page- and a chapter-and-section- based structure. Metadata may be provided in several different formats and standards, and this allows for the native coding of metadata on the document in different formats—for instance RFC1807 and Dublin Core.

When a document first enters the library, it is given a unique identifier. That identifier will remain with the document throughout any subsequent revisions, and is recorded within the METS framework.

The rest of this section describes each phase in greater detail.

Having described the configuration controls and document representation that underpin the new collection-building architecture, we now describe the building process itself.

4. BUILDING DL COLLECTIONS

In Greenstone 3, collections are built in a sequence of distinct phases:

Expansion. Compressed files such as Zip archives are expanded, and links to web sites are expanded into lists of constituent web pages.

Recognition. All files are sent to the Recognition Manager, which identifies groups of files as documents.

Encoding. All the recognized documents are cataloged for the subsequent phases of building. Inherent metadata that is marked up in the original files (e.g. the title of the document) is encoded along with the content of the document.

Extraction. Every document is passed through extractors, which use special processing algorithms to extract information from the document (e.g. title, keyphrases) or add metadata stored in special files.

Classification. Documents are assigned to classifiers (e.g. topical classifiers, ordered list classifiers) depending on their inherent and extracted metadata.

Indexation. Documents are indexed to support later searching.

Validation. Post-building checks on the collection and its documents.

4.1 Expansion

Before any individual documents are identified, any files that are compressed, or refer to external sources, are expanded. The case of a Zip archive has already been mentioned; other cases include OAI repositories, and links to web sites via a file of URLs. The purpose of this phase is to create a complete list of all files that are available for building the collection. These raw files are then passed to the next phase for grouping into documents.

4.2 Recognition

The relationship between files and documents is complex. A file of emails may be seen as one file that contains many documents. Conversely, a JPEG image and a plain text file that describes it may constitute a two-file document. Our architecture uses special "recognizer" plugins to address the problem of distinguishing documents from files.

When the building process is initialized, all plugins listed in the collection configuration file are registered by a "Recognition Manager." When the Expansion phase terminates, this Manager is handed each file in the collection in turn. It sends these files to each plugin. For some simple cases—like a plain text document comprising a single file—the plugins are trivial. However, in other circumstances conflicts can emerge, and the Recognition Manager includes a scheme for resolving these.

For example, suppose a collection includes both web pages and digital photographs. A single JPEG file may be a document in its own right, or form part of a web page. By default, JPEG files that are referred to by web pages, being part of a larger whole, are not treated as documents in their own right, while JPEGs that do not appear in the web pages are identified as separate, discrete documents. This default behavior can be adjusted, if desired, to allow *all* JPEG files to be treated as documents, even ones that appear in a web document. The same issue arises with any image file on a web page. Furthermore, such a file (e.g. a logo) may be part of more than one document. In this case it is treated as a subcomponent of all the documents in which it appears.

Some files may contain only metadata about other files (e.g. the XML metadata files supported in Greenstone 2) or may list other documents (e.g. as a list of html links). Such files are identified in the extraction phase and are set aside for processing by the Extraction phase later in the build process. Another special case can arise in the case of bibliographic files, where each reference in the file may be treated as a separate, single document. Our use of the METS framework permits us to identify parts of files as well as whole files. If the bibliographical entries contain links needing further expansion (e.g. to PDFs on the Internet), this will be handled in the Encoding phase.

Once the Recognition Manager has taken a pass through the entire inventory of files and identified the documents, it resolves any conflicts that occur and hands the output on to the Encoding phase. At this stage all that is known of a document is the list of files that constitute its content. No metadata whatsoever has yet been assigned. The content and representation of any metadata found within the document files is identified in the next phase.

4.3 Encoding

Documents appear in a wide variety of forms, textual encodings and file formats. The Encoding phase creates the document framework that is used throughout the remainder of the building process. This framework supports the retrieval of the full text of the document. It contains metadata defined within the document, and will also contain any metadata added to it. Finally, it binds the document, with its representation, into the classification and indexation structures. The METS document framework described in Section 3.2 is used to encapsulate the files, metadata and structure of each document.

Each type of document is encoded by its particular document plugin, which translates the recognised input files into marked up content and metadata within the METS framework. In the encoding phase, the full text of the document is identified, and metadata contained in the original files is also inserted into the METS representation of the document. The source files for the document are also recorded within its METS form. The document plugin can also follow links to other files and retrieve those before further processing. At the end of the Encoding phase, documents are ready to be processed by extracting further metadata, indexation and storage.

The METS encoding of each document is stored in an SQL database for rapid retrieval, and later access in the live digital library system. The actual database management system used can be readily altered - at present we use MySQL due to its conformance with the GNU Public License with which we also comply.

4.4 Extraction

The Extraction phase is controlled by the Extraction Manager, which filters the documents through a series of extraction plugins. These can be of two origins:

- Metadata-only files identified in the recognition phase above
- Explicit mention in the collection configuration file

The former method is typically used for files of descriptive metadata that are provided with the documents. The latter is used to invoke automatic extraction tools such as the KEA Keyphrase Extractor [7, 17].

At the conclusion of the Extraction phase, all metadata has been collected and stored in the METS document representation. The next two phases place the documents into the context of the collection as a whole.

4.5 Classification

The Classification phase uses each document's metadata to site it within the browsing structures, or "classifiers," that are supplied by the collection to access the documents it contains. Each classification is represented by a plugin specified in the collection configuration file. There may be more than one instance of any given classification type, achieved by initiating a separate copy of the plugin for each classification. Classifications are stored in a database, and when a document is matched to a node in a classification, this fact is recorded in both the document and the classification node.

4.6 Indexation

To support the widest range of indexing systems, indexers are also treated as plugins. Again, the indexer—or indexers—used when building a collection are determined by the collection configuration file. For each indexer, parameters can be supplied that control its behavior. Indexers can build several different indexes, each with its own parameters. In the case of MG[16], for example, the structure level (e.g. document, section) and field (full text, title, etc.) can be controlled. In the example configuration file in Fig. 1, section- and document-level indexes of the full text and are built, plus a section-level title index.

4.7 Validation

Validation plugins are used to control the quality of the collection by checking the final form of each document, and the collection as a whole. For example, a collection may insist on a complete metadata set—that is, all compulsory fields must be present. Or the databases may be checked for empty classification nodes, which may be suppressed. Validation could be done in earlier phases, but we separate it into a final phase for architectural clarity.

4.8 Summary

Greenstone 3 uses a seven-phase build architecture. The first three phases identify and store each document. The next three phases enrich and index those documents. The final phase provides for quality control. The phases support different behaviors and activities that are required by different types of software, and a set of distinct functions that should assist programmers in extending any given implementation of the architecture.

Each document is recorded in the METS framework, providing a standard form that gives access to different metadata formats, file formats and document structures. The document plugins give the indexation and classification plugins access to content without having to decode or transcribe documents.

5. EXTENDIBILITY

The central role of plugins in each phase should now be apparent. However, we have made little mention of the structure of plugins themselves, or how they connect to the core

system. The architecture achieves extensibility through *Plugins* and *Managers*.

5.1 Managers and Plugins

Each phase of the building process is controlled by a Manager—e.g. the Recognition Manager, the Indexer Manager. Managers are configured through the collection configuration file when the building process starts; and in some cases further configuration occurs when special files—like metadata-only files—are found when building. Each manager coordinates the plugins for its phase of the build cycle.

Each manager works in the same way. It initializes and configures its plugins. Then, when its phase runs, it takes each plugin in turn and tells it to do any necessary preparatory work. Next it passes each document or file in turn to that plugin. Finally it sends the plugin a termination message.

Document plugins, which represent particular types of documents, are unique in that they apply to several different phases of the building operation, whereas other plugins appear only within one phase. Managers accept a list of plugins from the configuration file. New plugins are readily installed into the system by placing them into the plugin directory for their type, and adding them to the configuration file of one or more collections.

The standard interface for extraction, indexation and classification plugins includes:

getNumberOfPasses Return the number of passes required by the plugin

startPass(int) Begin a pass for this plugin

processDocument(Document) Process the given document

endPass(int) Terminate a given pass

tidyup Close the plugin

configure(XML Node) Pass configuration information from the collection configuration file to the plugin.

The *getNumberOfPasses* method allows plugins to request multiple passes over the documents. The MG indexation plugin, for example, requires a minimum of two passes plus a further two passes for each index it builds. Running each plugin separately and serially keeps memory overheads down.

5.2 Metadata Plugins

The METS framework allows for the incorporation of metadata in whatever format the person who encodes the document wishes. We extend the general concept of our plugin approach to the encoding of metadata. Whenever metadata is assigned to a document, be that due to automatic keyphrase extraction, information in a separate metadata file, in the document source or otherwise, our build framework passes the metadata to a plugin that handles the particular format. Thus, for example, we currently have Dublin Core and MODS metadata plugins.

A metadata plugin has a different structure to the process plugins described above. All that is supported is the assignment, adjustment or removal of metadata items. Metadata can be received in XML format, for encoding into the final built SQL database, or as data items (e.g. in the case of extraction). In either case, field names are checked and other validation steps taken as the metadata format demands. All this work is delegated to the metadata plugin. A generic plugin exists as a baseline for handling unknown metadata formats, but given the range of potential XML encodings even for simple label/value pairs, using different combinations of tags, text and tag attributes, it cannot yield a universal solution. The XML import facility of metadata plugins is important, as when a METS document is ingested directly into Greenstone, we delegate the decoding of the metadata to these plugins.

6. ADVANCE OVER EARLIER WORK

The architecture that we have described capitalizes on lessons learned from the existing Greenstone digital library system, and incorporates some significant improvements. At the time the earlier system was designed (1998) several important open standards did not exist, or were available only in draft form. For example, the METS Document Framework, a key component of the new architecture, was unborn.

The original Greenstone design adopted *ad hoc* file formats that became inconsistent with emerging norms. Collection configuration was viewed as a compilation process directed by a configuration file whose format was similar to the Makefile format [4] rather than by directives couched in a structured markup language. It used an HTML-like structure, the Greenstone Archive format, to store documents and metadata, which is less structured than modern XML. The internal formats were also inconsistent with each other—configuration files being flat files whereas document files were derived from HTML. These simple and apparently cosmetic differences magnified into large-scale divergence.

6.1 Architecture

In Greenstone 2, collections are constructed in two phases: *importing* and *building*. The first parallels the Expansion, Recognition and Encoding phases of Greenstone 3, while the second mirrors the Extraction, Classification and Indexation phases. Both phases use the same set of plugins, which are listed in the collection's configuration file and loaded separately in each phase—despite the fact that some plugins only pertain to one phase.

Document plugins, like *HTMLPlug*, played a rather different role from that described above. Documents were identified by a single key file—e.g. an HTML file—and complex document structures were poorly supported. A special generic plugin was later developed that recognized multi-component documents such as a Word file and its corresponding PDF version. However, the plugins had to be carefully ordered in the configuration file to recognize this compound document. Once a file was captured by a plugin, subsequent plugins could not see it. The Recognition phase described above facilitates complex document forms and arbitrates competing claims for a file by different document types.

6.2 Plugin details

Document plugins handle the encoding phase differently in the new design. Originally, all documents were encoded into the standard Greenstone Archive Format as soon as they were encountered. This duplicated content, and could lose, or render inaccessible, some information in the original

file. The benefit was that all documents were presented to subsequent phases in a standardized format.

In the new architecture, plugins convert document content to either plain text or well-formed XML, which allows formats such as TEI [9] to be retained verbatim. The METS Framework provides a standard container for metadata and structural information. This open public standard retains the benefits of the original Greenstone Archive Format, and goes further by allowing multiple structures within the same document.

Plugins in Greenstone 2 could be of three types—Expansion, Document, and Extraction. All were placed in the same pipeline. The new architecture makes a clearer distinction between phases and allows a richer variety of documents. For most tasks, it supports clearer, more compact, interfaces. In addition, monolithic parts of the old design, such as indexation, are now readily adjusted and extended in the flexible manner exhibited by other phases.

6.3 METS Ingestion

Since document ingest is built around the METS framework, the Greenstone 3 build process can directly receive material presented in a METS encoding. Though other DL systems, for example DSpace [8], are able to produce METS encodings of their documents, our system is to the best of our knowledge the first to present a generic METS ingestion facility. We have tested the interaction of our metadata plugins and direct METS ingestion with METS material from several different sources, and can successfully import a range of different encodings. Further work is required to extend the number of metadata formats supported in our new build process, and to test further METS encoded material.

6.4 Example: The Kids Digital Library

We briefly present an actual digital library that was difficult to accommodate within the earlier design, and show how it benefits from the new architecture. In the Kids DL [11] each document can belong to several collections. Some collections are private (e.g. a child's own documents), others public. Some documents are unchanging (accepted final essays); others are under continually revision by a restricted group of users. Students can annotate the work of others, and teachers provide feedback too. All collections support full-text searching and metadata-based browsing; some have additional browsing facilities (see below). Collections sometimes change rapidly over a short period—e.g. during class.

Using the new architecture, many of these requirements can be delivered far more easily. Consistent document identifiers and explicit change history allows revisions of a document, and comments made between changes to be tracked. Incremental indexing is supported by recording accession and revision data as METS administrative metadata. The METS framework allows files to be duplicated between collections yet reside in only one location.

The Kids DL featured unusual browsing classifiers such as the "Top ten" and "Latest" stories. These require simple support for feature extraction. In the earlier design, this could only be done by editing the building scripts—causing portability problems, and making it difficult to update the software. The new architecture circumvents these problems by modularizing feature extraction.

7. COMPARISON WITH OTHER DIGITAL LIBRARY SYSTEMS

A number of other DL systems support a variety of document formats (e.g. EPrints, Dienst, Alexandria). However, this is commonly achieved by indexing only metadata on the document, not the full text. Conversely, other systems (e.g. Perseus, Cheshire) index the full text but require standard document formats on input. Metadata formats are often adjustable, but at varying cost. The building architecture of the different systems is seldom documented in published papers. Cheshire II [5] emphasizes the construction of digital libraries from original scanned documents. The process described for collection building reveals a system of fixed metadata fields and a strict control of the format in which documents are presented to the system. Nowhere is support for feature extraction, expansion of compressed files, or novel indexes described.

The CORR[1] is built on the NCSTRL software [2]. The CORR documentation reports that the system requires documents to be submitted in a standard source format. No documentation on CORR or NCSTRL describes the parameters for collection configuration. In comparison, Greenstone 3's architecture permits flexible metadata and full text extraction, in addition to extendible indexation, classification and mining (extraction) of document content.

DSpace presents another digital library that can provide flexible building of full-text collections. Full text builds are achieved through the use of a series of *filters* and the execution of a *batchbuild*. This approach is broadly similar to that taken by Greenstone. As with Greenstone, the batch build process is run by the librarian as required. However, the updating of cataloged metadata is done at the time of the submission of a document to the library. Thus, DSpace combines the immediate update of a catalogue database with the periodic building of a full-text library. One consequence of this is that the database and full-text index can reflect the collection at different points of time and be inconsistent with each other. In terms of the DSpace batch build, its features are closer to that of Greenstone 2, with the focus of attention being on translating documents for indexation by a full-text indexer. Our new building framework widens this approach to other forms of processing and analysis of the document's full text such as the extraction of phrases.

8. DISCUSSION

Indexing tools have a significant effect upon the implementation of the building of a digital library collection. Where the electronic equivalent of a card index is maintained, the indexation features of a database system are entirely adequate. Such a database can be updated continuously without the need for an explicit build process.

However, as digital libraries provide more sophisticated access, and process the text of their constituent documents, continuous updates become problematic. Novel features of documents need to be identified and extracted—but often these very features are not identified on a document-by-document basis. For instance, keyphrase extraction [7] and full-text indexation often rely on information collected across many documents. Treating each new document entering the library separately may fail to recognize useful patterns, as

[1] http://arxiv.org/archive/cs/intro.html

that pattern may only be identified when also using documents already in the library and other new documents.

It is for this reason that Greenstone 2 introduced the presence of an explicit build process. Technical advances have reduced the need for parsing the entire collection for some purposes, but new techniques have increased this demand. An effective build process enables quick, small additions to the library at low cost, and equally supports entire reprocessing when required. Our new framework allows individual indexation plugins to decide upon the best strategy for them: all documents are sent to every plugin, but additional information indicating whether the document has been modified or is new is also given. Plugins can also maintain their own history. This allows the plugin to consider not only the immediate rebuild, but previous builds and the decisions it made then. Sophisticated and effective strategies can them be followed.

Lucene,[2] for example, allows incremental indexation, but for performance reasons it is wise to do an "optimize" rebuild on occasion, which in effect performs a complete rebuild. Using our framework, a Lucene indexation plugin can normally perform an incremental, rapid, rebuild. However, if the plugin identifies from its own history that a sequence of many incremental rebuilds has occurred, and an optimizing rebuild should now take place, it can do so immediately. As Greenstone 3 sends all documents to the plugin, it can process every document without additional interaction with the main build process.

Our earlier building process framework is mirrored in the recent DSpace system. Our new build process framework widens the use of interchangeable plugins, removing barriers we found when adding alternative indexers and new features to the earlier process. In addition, we move from using a proprietary document model to a standard, open model that can be readily exported for subsequent ingestion into other digital libraries and archives.

The same range of options is available for all plugins: thus, keyphrase extraction can be done using an incremental or a full-parse approach.

Using the configuration system within the framework, plugin authors can allow the digital librarian to tailor the behavior of their plugins. This allows such users to force quick rebuilds unless additional parameters are given at build time, or conversely to default to complete rebuilds when required.

8.1 Reference Models

The OAIS reference model [3] specifies a number of processes that can be applied to a document in an archive. Some of these are pertinent to the document $after$ indexation, so these are not immediately relevant to the build process of a library. However, three processes are highly relevant: ingestion, indexation and storage. The *Ingestion* process standardizes the framework in which information about the document is held. In our build process, this equates to the first three processes - expansion, recognition and encoding. The *Storage* process is fulfilled either by the storage of the original document on the host file system, or by compression through a full-text indexer such as MG in our indexation phase. Finally, the *Indexation* process we subdivide into extraction, indexation and classification.

[2]lucene.sourceforge.net

8.2 Quality Assurance

The EPrints system allows for the review of new metadata before a change is passed into public view. DSpace also has a verification step in the submission process for an individual documents. The presence of an explicit verification step supports good quality practice, and mirrors the behavior of the original Greenstone 2 system. In our old framework, a rebuilt collection's index files were placed in a "building" directory, whilst the live data was stored in an "index" directory. Similarly, a Greenstone 2 collection could be kept private before full release. The presence of a second directory and publication controls allowed the librarian to test a changed collection before release. However, in our redesign we believed that further work could be done to better support quality assurance.

Previous experience with authoring support has indicated that simple computational tools can achieve a significant contribution to quality control. For example, Thimbleby [12] demonstrated that many spelling errors within a document, or across several documents, can be identified without the use of a prepared dictionary. Broken links and missing titles of documents can also be readily identified automatically, and problems such as the mis-assignment of documents to non-existent subject headings, or to keyphrases not used by other documents, can be automatically detected. We have already identified the presence of validation plugins in our build process, and clearly automatic tools to identify these sorts of errors can be easily configured into the building process of a collection.

However, human intervention is both technically hard to avoid and also, from a library perspective, highly desirable. We therefore need to not only have a structured build process, as described here, but also to structure testing of the built collection, and its final release for public access.

When the updated collection is finally made live, notifications can be sent to readers.

9. CONCLUSION

In the new collection building architecture we have described the building process is segmented into a number of distinct phases. Once documents are identified, they are encoded into a flexible, open framework (METS) and are passed in that form to the succeeding phases of the build process. Within each phase, the elements are componentized to support greater portability and simpler development. The build process is configured through a simple XML format file which is readily extensible for future components.

We have implemented this architecture in a practical digital library system. The use of a document framework rather than standardized document encoding has not created any obstacle to providing existing features. Rather, it supports our move to more complex document file structures and permits any processing of the document (e.g. by an indexer) to either refer to the original binary or to extracted content as it chooses. We also demonstrated that METS cannot only be used for encoding in a specific form for a specific library, but that a METS-centered build process can successfully ingest varied METS encodings of documents.

We believe that the challenge of providing effective, portable solutions to building digital library collections is an important one. As file formats diversify, the issue of digital preservation grows. Similarly, if extraction and indexation

technologies are not made more portable their adoption will slow. Our new architecture provides a step forward in meeting these challenges.

10. REFERENCES

[1] D. Bainbridge, G. Buchanan, J. McPherson, S. Jones, A. Mahoui, and I. Witten. Greenstone: A platform for distributed digital library applications. In *European Conference of Digital Libraries*, pages 137–148, Darmstadt, Germany, 2001.

[2] J. R. Davis and C. Lagoze. Ncstrl: design and deployment of a globally distributed digital library. *J. Am. Soc. Inf. Sci.*, 51(3):273–280, 2000.

[3] C. C. for Space Data Systems. *CCSDS 650.0-B-1: Reference Model for an Open Archival Information System (OAIS). Blue Book.* First edition, 2002.

[4] Free Software Foundation. *GNU make Manual (version 3.80)*, 2002.

[5] R. R. Larson and C. Carson. Information access for a digital library: Cheshire ii and the berkeley environmental digital library. In *Proceedings ASIS '99*, pages 515–535. Information Today, 1999.

[6] Library of Congress. *Metadata Encoding and Transmission Standard (METS).*

[7] G. W. Paynter, I. H. Witten, S. J. Cunningham, and G. Buchanan. Scalable browsing for large collections: A case study. In *Proc International Conference on Digital Libraries*, pages 215–218, 2000.

[8] M. Smith, M. Bass, G. McClella, R. Tansley, M. Barton, M. Branschofsky, D. Stuve, and J. Wakler. DSpace: An open source dynamic digital repository. *D-Lib Magazine*, 9(1), 2003.

[9] C. Sperberg-McQueen and L. Burnard, editors. *Guidelines for Electronic Text Encoding and Interchange.* TEI P3 Text Encoding Initiative, Oxford, 1999.

[10] H. Suleman and E. A. Fox. Designing protocols in support of digital library componentization. In *Proc European Conference on Digital Libraries*, pages 568–582, 2002.

[11] Y. L. Theng, N. Mohd-Nasir, G. Buchanan, B. Fields, H. Thimbleby, and N. Cassidy. Dynamic digital libraries for children. In *Proc Joint Conference on Digital libraries*, pages 406–415. ACM, 2001.

[12] H. W. Thimbleby. Basic user engineering principles for display editors. In M. B. Williams, editor, *Proceedings 6th. International Conference on Computer Communication*, pages 537–542. ICCC, 1982.

[13] I. H. Witten. Examples of practical digital libraries: collections built internationally using greenstone. *D-Lib Magazine*, 9(3), 2003.

[14] I. H. Witten and D. Bainbridge. *How to build a digital library.* Morgan Kaufmann, San Francisco, CA., 2003.

[15] I. H. Witten, D. Bainbridge, G. W. Paynter, and S. J. Boddie. Importing documents and metadata into digital libraries: Requirements analysis and an extensible architecture. In *Proc. of the European Conference on Digital Libraries*, pages 390–405, Sept. 2002.

[16] I. H. Witten, A. Moffat, and T. C. Bell. *Managing gigabytes: compressing and indexing documents and images. (second edition).* Morgan Kaufmann, San Francisco, CA., 1999.

[17] I. H. Witten, G. W. Paynter, E. Frank, C. Gutwin, and C. G. Nevill-Manning. KEA: Practical automatic keyphrase extraction. In *ACM DL*, pages 254–255, 1999.