Importing documents and metadata into digital libraries: Requirements analysis and an extensible architecture

Ian H. Witten,* David Bainbridge,* Gordon Paynter,† Stefan Boddie*

*Computer Science Department
University of Waikato
New Zealand
{ihw, davidb, sjboddie}@cs.waikato.ac.nz

† University of California Science Library
Riverside
California, U.S.
gordon.paynter@ucr.edu

ABSTRACT

Flexible digital library systems need to be able to accept, or "import," documents and metadata in a variety of forms, and associate metadata with the appropriate documents. This paper analyzes the requirements of the import process for general digital libraries. The requirements include (a) format conversion for source documents, (b) the ability to incorporate existing conversion utilities, (c) provision for metadata to be specified in the document files themselves and/or in separate metadata files, (d) format conversion for metadata files, (e) provision for metadata to be computed from the document content, and (f) flexible ways of associating metadata with documents or sets of documents. We argue that these requirements are so open-ended that they are best met by an extensible architecture that facilitates the addition of new document formats and metadata facilities to existing digital library systems. An example implementation of this architecture is briefly described.

Keywords

Software architecture, metadata, digital library architecture, Greenstone digital library software

INTRODUCTION

Flexible digital library systems need to be able to accept documents and metadata in a variety of different forms. Documents may be available in web-oriented formats such as HTML and XML, in word-processor formats such as Microsoft Word and RTF, in page description languages such as PostScript and PDF, or in media-rich formats such as JPEG images and MPEG video. Metadata may also be available in a variety of different forms: embedded in the documents, in separate metadata files, in spreadsheets, or even encoded into file naming conventions; or it may be computable from the documents themselves. Digital library designers must either insist that users adopt a particular prescribed scheme for document and metadata specification, or face the challenge of coming up with flexible, extensible, ways of allowing different formats to be accommodated.

Of course, there are standards for representing metadata, just as there are standards for representing documents. Embedded metadata might take the form of *meta* tags in HTML, *info* attributes in RTF, or *title* and *author* commands in LaTeX. Or it may be expressed as MARC records or in some standard encoding of Dublin Core metadata. While it might be desirable to enforce the use of a particular standard, as most digital library systems do, in practice there are many different standards to choose from! Furthermore, legacy data will always present conversion problems.

This paper explores a different approach that involves an extensible architecture.

REQUIREMENTS

There are two basic elements that must be considered when importing material into a digital library. The first comprises documents, where the term is interpreted in a suitably general way. These form the raw material of any library. The second is metadata, which is summary information, in a structured form, about the documents. This forms the basis for the organization of material: librarians are expert in creating metadata and using it to facilitate access to large information collections.

When a large collection of documents and metadata is made available for import into a digital library, questions arise as to the structure within which the documents are presented, and how the relationship between the documents and the metadata is expressed—which metadata pertains to which documents.

Documents

We use the term "document" to denote any information-bearing message in electronically recorded form. In a digital library, a document is a particular electronic encoding of what in library science is called a "work." A pressing practical problem is the wide variety of ways in which this encoding may be expressed. We focus in this paper on textual documents and media-rich formats supported by metadata; similar considerations arise when dealing with direct manipulation of multimedia content.

There are four principal styles of format in which electronic documents are expressed. The first comprises in web-oriented formats such as HTML, XHTML and XML document manifestations such as the Text Encoding Initiative and Open eBook format. The principal difficulty here is that such documents are not always self-contained:

they frequently include explicit links to other resources such as images or other documents and these raise questions about the identity of the document—where are its boundaries?

When such formats are designed with a strong notion of a work—for instance an Open eBook document contains a manifest of all external resources that constitute a single book—such decisions are straightforward to make. In less well defined situations (HTML being the archetypal example) images that a document refers to are generally considered as part of the document, and when documents are downloaded a subdirectory is created containing the images. In contrast, linked documents are considered as having separate identity.

When a set of documents are imported into a digital library, the question of what to do with links must be revisited. For example, when a link is to another document that is also being imported, it is often appropriate to replace it by a link to the library copy of the target document instead of a link to the original external resource. Such decisions will depend on the digital library's aim and context.

The second style of expression comprises word-processor formats such as Microsoft Word or RTF ("rich text format"). RTF is designed to allow word-processor documents to be transferred between applications, and uses ASCII text to describe page-based documents that contain a mixture of formatted text and graphics. In contrast, the native Word format is intended for use by a single word processor. Strictly speaking, it is inappropriate to use this format to convey documents to digital libraries; nevertheless, users often want to do that.

There are two key difficulties with word-processor formats: they continually evolve to meet new demands, and they are often proprietary, with no public documentation. These problems are less severe with RTF: it is documented, and has an explicit and well-defined mechanism for adding new commands in a backwards-compatible manner that ensures that reasonable results can be obtained when new documents are processed by old software. However, with native Word the situation is different. Even Microsoft products sometimes can't read Word documents properly. Word is really a family of formats rather than a single one, and has nasty legacy problems. Although Microsoft have published "as is" their internal technical manual for the Word 97 version, the format continues to evolve. A serious complication is that documents can be written to disk in "fast save" mode, which no longer preserves the order of the text. Instead, new edits are appended, and whatever program reads the file must reconstruct its current state.

The third style of expression for documents comprises page description languages like PostScript and PDF. These combine text and graphics by treating the glyphs that express text as little pictures in their own right, and allowing them to be described, denoted, and placed on an electronic "page" alongside conventional illustrations. They portray finished documents, ones that are not intended to be edited, and are therefore more akin to traditional library documents than word-processor formats. Most of the time digital libraries can treat documents in these languages by processing them using standard "black boxes": generate this report in a particular page description language, display it here, transfer it there, and print. However, to build coherent collections out of the documents, it is beneficial to be able to extract the text for indexing purposes and some elements of document structure for browsing purposes, and these are challenging problems.

The fourth style of expression is media-rich documents such as sound, pictures and video. When accompanied by textual descriptions (the view taken here), their treatment becomes one of associating metadata with documents. This provides a baseline approach that unifies the different media types and permits all the metadata methods discussed below for the general case to be applied. Direct manipulation of content is also possible, but beyond the scope of this paper.

We have taken care to categorize these four groups as styles because in practice their boundaries overlap. For example, although PDF is formally a page description language it supports hyperlinks akin to web-based documents and annotations comparable with word processed documents. Proprietary eBook formats exhibit web characteristics, but are also bound up in issues resulting from closed specifications. This means the importing architecture should not be compartmentalized based on these categories, rather it should be flexible enough to respond to a range of conceptualized document features regardless of their origin.

Aside from document format, another key question for digital libraries is the librarian's traditional distinction between "work" and "document". This arises when we have to deal with different versions of a particular work. Digital representations of a work are far easier than printed ones to both copy and change. It is necessary to decide when two documents are to be considered the same and when they are different. Digital collections often contain many exact duplicates of documents; should duplicate copies be retained? When a new version of a document appears, should it supersede the old one, or should both be kept? The answers will depend on the purpose of the collection. Archival or historical records must not be allowed to change, but errors in collections of practical or educational information must be correctable.

A further complication that affects the identity of documents is that interactions with digital libraries are often sustained over time—for example, by keeping records of the interaction history of individual users to facilitate future interaction. When identifiers are allocated to documents, decisions must be made about whether duplicates are significant and when new versions of documents supersede old ones. For example, one way of assigning identifiers is to compute a signature from the

word sequence that makes up the document. This is attractive because exact copies receive the same identifier and are therefore mapped into the same object. However, sometimes it is necessary to make an updated version of a document supersede the original by giving it exactly the same identifier even though its content is slightly different, and in this case identifiers cannot simply be computed from the content.

Metadata

Metadata may be conveyed in three basically different ways. It may be *embedded* in the particular documents to which it pertains, contained in *auxiliary* metadata files, or *extracted* automatically from the textual content of the documents themselves. The final category extends the traditional definition of the term metadata, but it does so in a way that is conducive to our requirements.

Embedded metadata

Document formats such as HTML and RTF allow metadata to be specified explicitly: in the former case using <*meta*> tags and in the latter with an *info* statement. These provide a mechanism for specifying that certain "attributes" have certain "values". However, it is rather limited. HTML imposes no checks or constraints on the particular attributes that are used, while RTF restricts the attributes to a small fixed set. Metadata values are strings with no other structure.

XML, in contrast, is specifically designed for expressing both document structure and metadata in a very flexible way. Document Type Definitions (DTDs) can be created that enforce appropriate constraints over the metadata that is present in an entire family of documents, including rules that govern the syntactic nesting of nodes. Metadata can be expressed as an "enumerated" type with a particular set of valid values, particular attributes can be declared to be "unique" within the document to act as identifiers, and so on. Even more comprehensive facilities for defining data structures are provided in the related standard XML Schema. As well as describing what structure is allowed in an XML file, this provides a rich array of basic types including year, date, and URI, as well as textual patterns and ways of subtyping and defining new types.

Other document formats have their own particular way of embedding metadata. Microsoft Office documents have "Summary" metadata that comprises title, date, subject, author, manager, company, category, keywords, comments. E-mail documents have sender, recipient, date, subject, and so on. PDF documents have title, date, subject, author, keywords, and binding (left or right edge). Music files may have title, date, composer, copyright, description.

Auxiliary metadata

Metadata pertaining to a document collection is commonly expressed in the form of a separate metadata file. There are two widely-used standard methods for representing document metadata: the "machine-readable cataloging" (MARC) format and the Dublin Core. They represent opposite ends of the complexity spectrum. MARC is a comprehensive, well-developed, carefully-controlled scheme intended to be generated by professional catalogers for use in libraries. Dublin Core is an intentionally minimalist standard intended to be applied to a wide range of digital library materials by people who are not necessarily trained in library cataloging. These two schemes are of interest not only for their practical value but also to highlight diametrically opposed underlying philosophies. There are two other bibliographic metadata formats that are in common use amongst document authors in scientific and technical fields, namely BibTeX and Refer.

As well as being able to express complete, self-contained documents along with their metadata, the XML language is capable of representing metadata alone in the form of auxiliary files. The information in any of the above metadata formats could easily be expressed as an XML file.

Extracted metadata

Whereas explicit metadata is determined by a person after careful examination and analysis of the document, "extracted" metadata is obtained automatically from the document's contents. This is usually hard to do reliably, and although extracted metadata is cheap, it is often of questionable accuracy. Relatively few documents today contain explicitly-encoded metadata, although the balance will shift as authors recognize the added value of metadata, standards for its encoding become widespread, and improved interfaces reduce the mechanical effort required to supply it.

"Text mining" may be defined as the process of analyzing text to extract information that is useful for particular purposes, and is a hot research topic nowadays. The ready availability of huge amounts of textual information on the web has placed a high premium on automatic extraction techniques. In this area there is hardly any underlying theory, and existing methods use heuristics that are complex, detailed, and difficult to replicate and evaluate.

Associating metadata with documents

Several challenging practical issues arise when a large collection of documents and metadata is made available for import into a digital library. Any large collection of files is almost always stored in some kind of hierarchical directory space. And the structure of the file hierarchy is hardly likely to be completely random: it invariably represents some aspects of the structure of the document collection itself.

Large collections are usually created by amalgamating smaller subcollections, and the upper levels of the hierarchy are likely to reflect this structure. This may have important implications for metadata assignment. Ideally, one assumes that all metadata associated with a document

is explicitly coded into it, or into a separate metadata file along with an explicit link to the document itself (e.g., its source URL). However, this is often not the case in practice. Particular pieces of metadata may be associated with the position of documents in the directory hierarchy. For example, all works published by one organization may be grouped together into a subdirectory, and within this works may be grouped by author into subordinate directories. In a collection formed by amalgamating diverse subcollections, different parts of the hierarchy are likely to have their own special organization. And certain pieces of metadata—publisher and author, in this example—may be defined implicitly rather than being explicitly stated along with each document.

It may be that directory and file names have been chosen to reflect particular metadata values. Perhaps more likely is that auxiliary files will be available that state what metadata values are to be assigned to which parts of the hierarchy. In our experience with building digital library collections, metadata is more likely to be presented in the form of spreadsheet files than as MARC records. And these spreadsheets have an *ad hoc* structure that often relates to the directory hierarchy containing the documents.

The lower levels of the directory hierarchy may also have their own structure, in this case determined by the documents themselves. For example, image files associated with a particular HTML document are usually placed together in a subdirectory. Different types of files will call for different treatment. Some contain documents, others images, others metadata in various forms, and still others should be ignored completely.

Metadata at other levels

We have tacitly assumed that metadata is associated with individual documents, and this is certainly the normal case. However, metadata is frequently present at other levels.

At the subdocument level, metadata may be associated with particular parts of documents. Chapter and section headings are a kind of metadata, and different sections of a document frequently have different authors. In general, any kind of metadata may be associated with any logical level of a document. Unless explicitly stated otherwise, subparts of a document inherit the document-level metadata.

At the supra-document level, metadata may be associated with a collection of documents as a whole. This might comprise a general statement of the topic covered by a collection, principles that govern the inclusion or exclusion of documents, principles according to which the collection is organized, collection editor and creation date, as well as particular information relevant to the collection, such as time period covered by a historical collection.

ARCHITECTURE

The requirements that we have identified above are so varied that they can only be accommodated by a flexible architecture, one that is extensible in that new facilities can easily be added as new document and metadata formats arise. The architecture that we propose has three main components: an internal document format to which all documents are converted when they are imported into the system; a set of parsers that process document and metadata formats (and the directory structure in which they are presented) and whose functionality can be combined by cascading them, and a scheme for "designing" individual digital library collections by providing a configuration file that specifies what kind of documents and metadata they are to contain, and what searching and browsing facilities they provide.

Internal document format

Collections of disparate documents in assorted formats are best accommodated by converting them into some standard internal representation for processing. There are many reasons, amongst them:

- as a fallback for presentation
- to support full-text searching
- as a vessel for metadata
- to resolve the issue of document identity
- to facilitate rapid processing.

A standardized internal document format need not imply a uniform *presentation* format, because a link to the original document can be retained and that original presented when the user calls for the document. In a Web-based digital library, HTML documents can be presented directly; XML documents can be presented using an appropriate stylesheet; PDF documents can be presented through the Acrobat reader; PostScript ones by invoking a suitable viewer; Word and RTF documents by invoking Word itself. However, if the necessary viewing utility is not available (e.g., Word on a Linux platform), a convenient fallback position is to display the document's standard internal representation instead, with a concomitant reduction in display quality.

One purpose of the internal document format is to support full-text searching. It should be expressed in the form of as electronic text; preferably using Unicode (say UTF-8 or UTF-16 representation) to accommodate different languages and scripts. The requirements of full-text search may mean that the internal format must preserve certain document components. For example, if searching is required at a paragraph, section, or chapter level, those structural units must be represented in the internal document format. Search terms can easily be highlighted in found documents if they are presented using the internal representation; otherwise some word-by-word positional mapping back to the original may be needed. Note that search engine operations such as stemming and case-folding may preclude highlighting by re-scanning the found documents for the search terms.

The internal document format is a convenient vessel for storing the document's metadata. Whether metadata is embedded in the original document file, specified in some

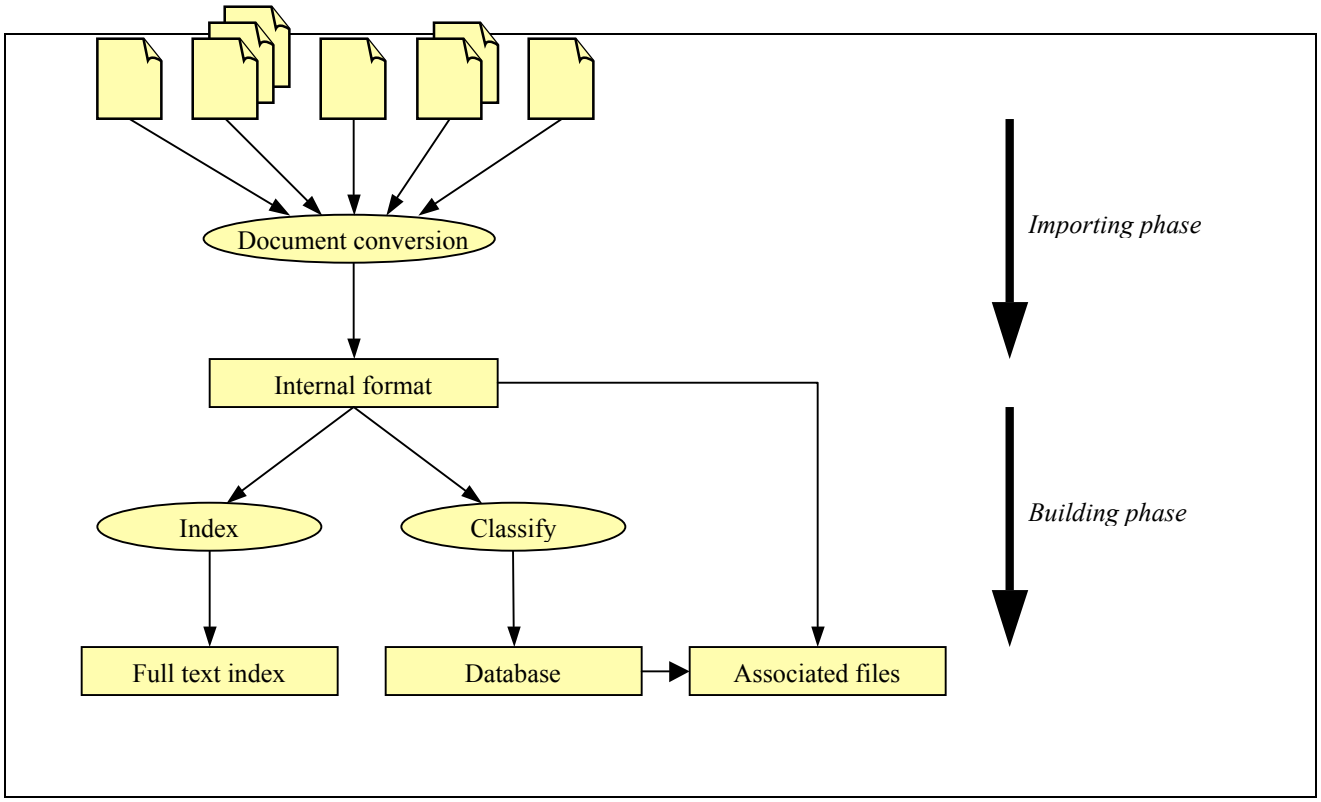

Figure 1 Creating a digital library collection within the proposed architecture

auxiliary file, or computed from the document text itself—or a mixture of all three—it can be pulled out into a standard form and saved as part of the internal document representation. This allows subsequent operations on document metadata to proceed in the same way irrespective of its source. XML is a suitable way of expressing document metadata along with its textual content and some structuring information.

Another purpose of the document format is to resolve issues of document identity. The simplest way to identify each document within the system is to assign it a unique label. It is important that labels persist over time, so that they can be used to record the history of individual users' interactions. Duplicate copies of documents can be eliminated by assigning them the same label. One way to do this, suggested above, is to compute a signature from the document's internal representation. Assuming that this representation is plain, unformatted text, this conveniently ignores differences that are merely formatting. However, for some collections, other ways of assigning document identifiers will be more appropriate—for example, when suitable labels pre-exist, differently-formatted versions of identical documents are to be distinguished, or there are special requirements for structured labels. Once computed, a document's identifier can be stored within it as metadata; then, the signature method simply becomes the default way of calculating this metadata value.

Finally, the internal format should facilitate rapid processing of documents. Most digital library collections are very large, and processing speed is often critical. Each document is converted to the internal format just once: subsequently all operations work on this representation.

Figure 1 gives an overview of how a digital library collection is created. The importing phase converts all documents to the internal format, which forms the basis for all subsequent operations. This improves efficiency when working with large collections (the normal case) by caching documents and metadata in a uniform format. We return to the remainder of the procedure shortly.

Plug-ins

The proposed architecture adopts a flexible scheme of parser modules, called plug-ins, to process document and metadata formats. Each collection may involve documents and metadata in several different formats, and a plug-in must be included for each type. Plug-ins fit together into a cascaded pipeline structure that provides a highly configurable workflow system.

Plug-in pipeline

Figure 2 shows the pipeline in generic terms. There are three types of plug-in: *structural*, *markup*, and *extraction*. The processing order in the pipeline is determined by the order in which they are listed in the collection's configuration file (see below).

The import process is initiated by feeding the name of the top-level directory that contains the source documents into the pipeline. This name is passed down the pipeline until one of the plug-ins signals that it is able to process it.

Figure 2 The generic plugin pipeline architecture (idealized: see text)

If an item travels to the end of the pipeline without being processed by any plug-in, the system generates a warning message and moves on to the next item. The process stops when no more items remain in the queue.

Structural plug-ins

Structural plug-ins operate on the generic file structure rather than on particular document formats. For example, under normal operating conditions, any filename that names a directory is processed by a structural plug-in that lists all files in the named directory and feeds their names, one by one, into the pipeline. In general this list includes further subdirectories, and they will be processed in the same way.

This is the default way in which a collection's directory structure is traversed. However, when certain files or directories need special treatment, or when directory names are significant for metadata assignment purposes it must also be possible to adapt this behavior. For example, it is often necessary to handle archived input formats in a digital library. Using this architecture, these can be expanded and their contents fed into the pipeline one by one. In other cases each source file may contain several documents (common E-mail formats do this) and need to be split into separate documents for the processing pipeline.

Markup plug-ins

Markup plug-ins process particular document or metadata types. To take one example of a document type, Microsoft Word documents require a markup plug-in that can parse this syntax. In this case, as in many others, external utilities will exist for document format conversion which the plug-

in calls to do the work. The result may be in a form such as HTML or text, which must be further transformed to the internal format using a subsequent plug-in.

To supply the necessary flexibility, plug-ins are designed to take various options that modify their behavior. One example is the input encoding used for the source files: in addition to ASCII, Unicode, and various other ISO standards, but special-purpose techniques used for encoding particular languages such as Chinese, Cyrillic, Greek, Hebrew, and standards accommodated by particular operating systems (e.g. Windows) are required if the architecture is to operate in an international arena. Another option is to specify which files a plug-in can process, in terms of a set of file extensions. The plug-in for HTML, for example, should accept filenames with the extension *.htm* or *.html*. It is also useful for a plug-in to block particular files and prevent them from being further down the pipeline—the same HTML plug-in will need to block files with such extensions as *.gif*, *.png* and *.jpg* because they do not contain any text or metadata but are embedded in documents when they are viewed.

Plug-ins for document formats that include metadata extract this information and transfer it to the document's internal-format file. One of the requirements identified above is to be able to assign metadata to documents from files that have been created manually (or automatically). For example, information pertaining to a document collection might be available in a standard form such as MARC records. A metadata markup plug-in processes this by placing the metadata into each individual document's internal file during the import process. Once there, it can be used to define searchable indexes and browsing structures.

Extraction plug-ins

The identified text and metadata are passed into a secondary pipeline of "extraction" plug-ins, again determined by the collection's configuration file (see below). These extract metadata from the plain text of the document and add it to the internal format. In this subsection we stretch the term "metadata" beyond its conventional usage to include any useful structured information about the contents of a document collection that can be extracted automatically.

One important piece of metadata that can be readily and reliably derived from a document's content is the language that it is written in. This can be added as *language* metadata. The same techniques can be used to identify the encoding scheme used for the document, which was mentioned above as an example of something that ought to be explicitly specifiable. This example illustrates that the sequential pipeline structure illustrated in Figure 2, while useful for conceptualizing what goes on, may be something of an idealization in implementation terms. We expand on this in the implementation section below.

A document's opening words are often used as a title substitute if *Title* metadata is unavailable, and so it is useful to be able to extract the first stretch of text and add it as metadata.

E-mail addresses are a good example of information that can be extracted automatically; they can be added to the document as *emailAddress* metadata. More challenging is to identify all dates (e.g., in years) relating to the content of historical documents and add them as *Coverage* metadata.

Technical, commercial and political documents make extensive use of acronyms. A list of acronyms and their definitions can assist document presentation by allowing users to click on an acronym to see its expansion, and help check whether acronyms are being used consistently in a document collection. Heuristic procedures can be used to extract acronyms and their definitions, and add them as *Acronym* metadata [1]. It is sometimes useful to annotate all occurrences of acronyms with links to their definitions.

In the scientific and technical literature, keywords and keyphrases are often attached to documents to provide a brief synopsis of what they are about. Keyphrases are a useful form of metadata because they condense documents into a few pithy phrases that can be interpreted individually and independently of each other. Again, heuristic procedures are available that obtain keyphrase metadata automatically from documents with a considerable degree of success [2, 3].

AN IMPLEMENTATION

The Greenstone digital library software is an implementation of large parts of the architecture that has been described above [5].

In our architecture, the design of a collection is encapsulated in a "collection configuration file" that directs both the import and build phases. This file contains the list of plug-ins for processing the source information, along with appropriate options for each, and this is used during the import procedure. The remainder of the configuration file includes information pertinent to the build phase—what searchable indexes the collection contains, what browsing facilities there should be, how the various pages should be formatted, and so on. We have described this mechanism in a previous paper [4] and will not dwell on it here.

Inheritance

Digital libraries process vast collections of documents, and one overriding requirement is that they operate reasonably efficiently. The processes that we are discussing are performed off-line, of course; and so do not affect the response to interactive library users. Nevertheless, even off-line collection building must take place expeditiously.

The pipeline architecture in Figure 2 has the potential to be rather sluggish. Each document must pass through many stages. It may be expanded from a compressed archive, converted to a different format (e.g., Word to HTML), reconverted (e.g. HTML to the internal format), have an identifier calculated (e.g. by hashing the full text), and the

```
<!DOCTYPE GreenstoneArchive [
  <!ELEMENT Section (Description,Content,Section*)>
  <!ELEMENT Description (Metadata*)>
  <!ELEMENT Content (#PCDATA)>
  <!ELEMENT Metadata (#PCDATA)>
  <ATTLIST Metadata name CDATA #REQUIRED>
]>
```

Figure 3 Greenstone internal document format

resulting document passed to a whole succession of extraction plug-ins that extract language and various other types of metadata. While the actual processing components are unavoidable, significant overhead will be incurred by repeatedly passing the document around. For example, if a Unix pipeline mechanism were used, the proposed architecture has the potential to be rather inefficient.

For this reason, the pipeline metaphor of Figure 2 is not to be taken completely literally. The Greenstone implementation of the architecture utilizes an inheritance structure to provide the necessary flexibility while minimizing code duplication, without repeatedly passing document representations around. All plug-ins derive from the same basic code, which performs universally-required operations like creating a new internal document object to work with, assigning an object identifier, and handling a document's sections.

It is also more efficient to implement the automatic extraction plug-ins as part of the basic plug-in, rather than as separate steps in the pipeline. This simplifies the triggering of the cascading pipeline, when a particular markup plug-in needs to pass a document on to an automatic extraction plug-in for further processing. It also allows structural plug-ins to be enhanced with markup capabilities and *vice versa*. This is a useful ability when, for example, metadata and documents are represented separately in the file system. It allows more flexible dependencies to occur than the pipeline model, as when an extraction plug-in identifies both language and encoding and the latter is used in the early stages of converting the document to Unicode. Finally, it increases efficiency of operation because there is no need to physically pass large volumes of textual data from one extraction plug-in to the next.

Internal document format

The internal format divides documents into sections and stores metadata at the document or section level. One design requirement is to be able to represent any previously marked-up document that uses HTML tags, even if the markup is sloppy. Another is to be able to parse documents very rapidly. The internal format is an XML-compliant syntax that contains explicit markup for sectioning and metadata assignment, and can also embed HTML-style

first	Extract the first characters of text and add it as metadata.
email	Extract E-mail addresses.
date	Extract dates relating to the content of historical documents and add them as *Coverage* metadata.
language	Identify each document's language.
acronyms	Extract acronym definitions.
acronyms	Add acronym information into document text.
keyphrases	Extract keyphrases from the full text and add them as *Subject* metadata.

Table 2 Metadata extraction plugins

markup that is not interpreted at the top XML level.

In XML, tags are enclosed in angle brackets for markup, just like HTML tags. The internal format encodes documents that are already in HTML by escaping any embedded <, >, or " characters within the original text using the standard codes *<*, *>* and *"*.

An XML <*Section*> tag signals the start of each document section, and the corresponding closing tag marks the end of that section. Each section begins with a metadata block that defines pertinent metadata. There can be any number of metadata specifications; each gives the metadata name and its value. In addition to regular metadata, the file that contains the original document can be specified as *gsdlsourcefilename*, and files that are associated with the document, such as image files, can be specified as *gsdlassocfile*.

Figure 3 gives the XML Document Type Definition (DTD) for the internal document format. The basic document structure is preserved by allowing it to be split into *Sections*, which can be nested. Each *Section* has a *Description* that comprises zero or more *Metadata* items, and a *Content* part (which may be null)—this is where the actual document's content goes. A name attribute and some textual data are associated with each *Metadata* element (the name can be anything). Implementing both parts to be optional enables collections to be built purely on content or metadata: a useful detail that, for instance, allows for multimedia based content described by metadata, and means that the design encompasses traditional electronic library catalogue systems.

In XML, *PCDATA* stands for "parsed character data," that is, text that may involve further markup; to include characters such as '<' therefore, their XML entity form must be used, such as *<* for the less than symbol.

TEXTPlug	Plain text.
HTMLPlug	HTML, replacing hyperlinks appropriately.
WordPlug	Microsoft Word documents.
PDFPlug	PDF documents.
PSPlug	PostScript documents.
EMAILPlug	E-mail messages, recognizing author, subject, date, etc.
BibTexPlug	Bibliography files in *BibTex* format.
ReferPlug	Bibliography files in *refer* format.
SRCPlug	Source code files.
ImagePlug	Image files for creating a library of images.
SplitPlug	Splits a document file into parts.
ZIPPlug	Uncompresses files.
BookPlug	Specially marked-up HTML.
GBPlug	Project Gutenberg E-text.
TCCPlug	E-mail documents from Computists' Weekly.
PrePlug	HTML output from the PRESCRIPT program.

Table 1 Document processing plugins

(a)

```
<!DOCTYPE DirectoryMetadata [
  <!ELEMENT DirectoryMetadata (FileSet*)>
  <!ELEMENT FileSet (FileName+,Description)>
  <!ELEMENT FileName (#PCDATA)>
  <!ELEMENT Description (Metadata*)>
  <!ELEMENT Metadata (#PCDATA)>
  <ATTLIST Metadata name CDATA #REQUIRED>
  <ATTLIST Metadata mode (accumulate|override) "override">
]>
```

(b)

```
<?xml version="1.0" ?>
<!DOCTYPE DirectoryMetadata SYSTEM
"http://greenstone.org/dtd/DirectoryMetadata/1.0/DirectoryMe
tadata.dtd">
<DirectoryMetadata>
  <FileSet>
    <FileName>nugget.*</FileName>
    <Description>
      <Metadata name="Title">Nugget Point Lighthouse
        </Metadata>
      <Metadata name="Place" mode="accumulate">Nugget Point
        </Metadata>
    </Description>
  </FileSet>
  <FileSet>
    <FileName>nugget-point-1.jpg</FileName>
    <Description>
      <Metadata name="Title">Nugget Point Lighthouse
        </Metadata>
      <Metadata name="Subject">Lighthouse</Metadata>
    </Description>
  </FileSet>
</DirectoryMetadata>
```

Figure 4 XML metadata format: (a) DTD; (b) Example

Plug-ins

Table 1 lists the document processing plug-ins, while Table 2 shows the metadata extraction plug-ins. As previously mentioned, all plug-ins that extract metadata from full text are implemented as features of the basic plug-in object, and consequently all derived plug-ins inherit them. Extraction plug-ins, therefore, are specified as options to markup and structural plug-ins. This has the advantage that they can, if desired, be used selectively to extract information from certain types of document.

The structural plug-in that traverses directory hierarchies incorporates a way of assigning metadata to documents from XML files that contain auxiliary metadata. It checks each input directory for an XML file called *metadata.xml* and applies its contents to all the directory's files and subdirectories.

Figure 4a shows the XML Document Type Definition for the metadata file format, while Figure 4b shows an example *metadata.xml* file. The example contains two metadata structures. In each one, the *filename* element describes files to which the metadata applies, in the form of a regular expression. Thus *<FileName>nugget.* </FileName>* indicates that the first metadata record applies to every file whose name starts with "nugget". For these files (sourced from a collection of photos), *Title* metadata is set to "Nugget Point, The Catlins."

Metadata elements are processed in the order in which they appear. The second structure sets *Title* metadata for the file named *nugget-point-1.jpg* to "Nugget Point Lighthouse, The Catlins," overriding the previous specification. It also adds a *Subject* metadata field.

Sometimes metadata is multi-valued and new values should accumulate, rather than overriding previous ones. The *mode=accumulate* attribute does this. It is applied to *Place* metadata in the first specification above, which will therefore be multi-valued. To revert to a single metadata element, write *<Metadata name="Place" mode="override">New Zealand</Metadata>*. In fact, you could omit this mode specification because every element overrides unless otherwise specified. To accumulate metadata for some field, *mode=accumulate* must be specified in every occurrence.

CONCLUSIONS

This paper has analyzed the requirements for importing documents and metadata into digital libraries and described a new extensible architecture that satisfies these requirements. It also includes a brief sketch of the Greenstone digital library system as an example implementation of this architecture. The proposed structure converts heterogeneous document and metadata formats, organized in arbitrary ways on the file system, into a uniform XML-compliant file structure. This simplifies the construction of the indexes, browsing structures, and associated files that form the basis of the runtime digital library system. Object oriented design further enhances capabilities whilst maximizing code reuse. The result is a comprehensive, flexible and extensible design.

Further details, and many examples, can be obtained from *nzdl.org*. The software is available at *greenstone.org*.

REFERENCES

1. Yeates, S., Bainbridge, D. and Witten, I.H. (2000) "Using compression to identify acronyms in text." *Proc Data Compression Conference*, edited by J.A. Storer and M. Cohn. IEEE Press Los Alamitos, CA, p. 582.

2. Dumais, S.T., Platt, J., Heckerman, D. and Sahami, M. (1998) "Inductive learning algorithms and representations for text categorization." *Proc ACM Conf on Information and Knowledge Management*, pp. 148–155.

3. Frank, E., Paynter, G.W., Witten, I.H., Gutwin, C. and Nevill-Manning, C. (1999) "Domain-specific keyphrase extraction." *Proc Int Joint Conference on Artificial Intelligence*, Stockholm, Sweden. San Francisco, CA: Morgan Kaufmann Publishers, pp. 668–673.

4. Witten, I.H., Bainbridge, D. and Boddie, S.J. (2001) "Power to the people: end-user building of digital library collections." *Proc Joint Conference on Digital Libraries*, Roanoke, Virginia, pp. 94–103.

5. Witten, I.H., Bainbridge, D., Paynter, S. and Boddie, S.J. (2002) "The Greenstone plugin architecture." *Proc Joint Conference on Digital Libraries*, Portland, Oregon.