UpLib: A Universal Personal Digital Library System

William Janssen, Kris Popat

22 November 2003

TR−03−16

UpLib: A Universal Personal Digital Library System

William C. Janssen and Kris Popat
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California, USA
{janssen,popat}@parc.com

ABSTRACT

We describe the design and use of a personal digital library system, UpLib. The system consists of a full-text indexed repository accessed through an active agent via a Web interface. It is suitable for personal collections comprising tens of thousands of documents (including papers, books, photos, receipts, email, etc.), and provides for ease of document entry and access as well as high levels of security and privacy. Unlike many other systems of the sort, user access to the document collection is assured even if the UpLib system is unavailable. It is "universal" in the sense that documents are canonically represented as *projections* into the text and image domains, and uses a predominantly visual user interface based on page images. UpLib can thus handle any document format which can be rendered as pages. Provision is made for alternative representations existing alongside the text-domain and image-domain representation, either stored or generated on demand. The system is highly extensible through user scripting, and is intended to be used as a platform for further work in document engineering. UpLib is assembled largely from open-source components (the current exception being the OCR engine, which is proprietary).

Keywords

Personal digital library, page image, thumbnail interfaces, web interfaces, document management, document repository

1. INTRODUCTION

Our Universal Personal Digital Library (UpLib) project addresses the capture, secure storage, organization, access, and use of documents involved in a person's day-to-day activities. Among those documents are credit card bills, family pictures, favorite books, letters of various kinds, receipts, tax forms, Web pages, and the other flotsam of modern life. As researchers, we also place particular emphasis on technical papers and notes, both ours and those of colleagues.
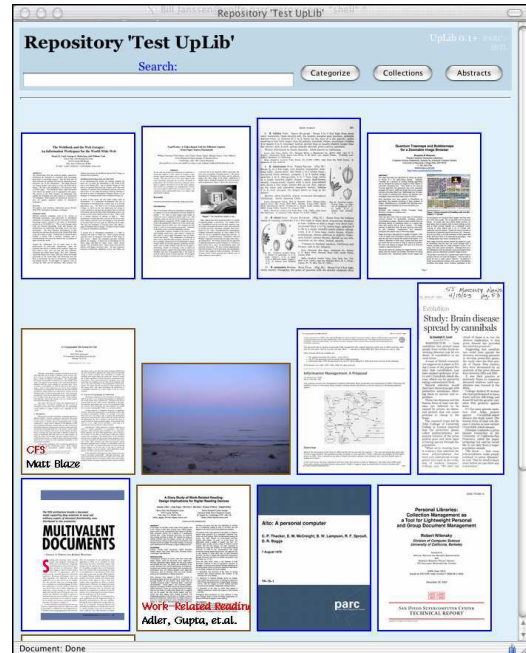
Figure 1: Document icons in the second version of our system. Each icon is generated using a *constant-area* algorithm, which allocates the same amount of area to each thumbnail, regardless of orientation. This approach allocates a fair amount of space to landscape-format documents, and makes it easy to distinguish between letter-size and A4-size documents by their relative heights. Clicking on an icon opens that document in a page-view mode.

Our intention is to deploy this library in multiple forms, including one that can be carried always in a small device like a PDA. Our approach uses both document images and document text but emphasizes images; we feel that the image domain presents certain unique opportunities not available in the symbolic domain, including a degree of universality and resistance to "bit-rot" or obsolescence of digital format.

It is likely that many if not most of the documents entered into the system will rarely be accessed, and some never accessed at all. It is nevertheless important that all documents entered into the system be retrievable so that the originals — paper or digital — may be discarded. Our personal information needs cannot be predicted perfectly in advance, so it
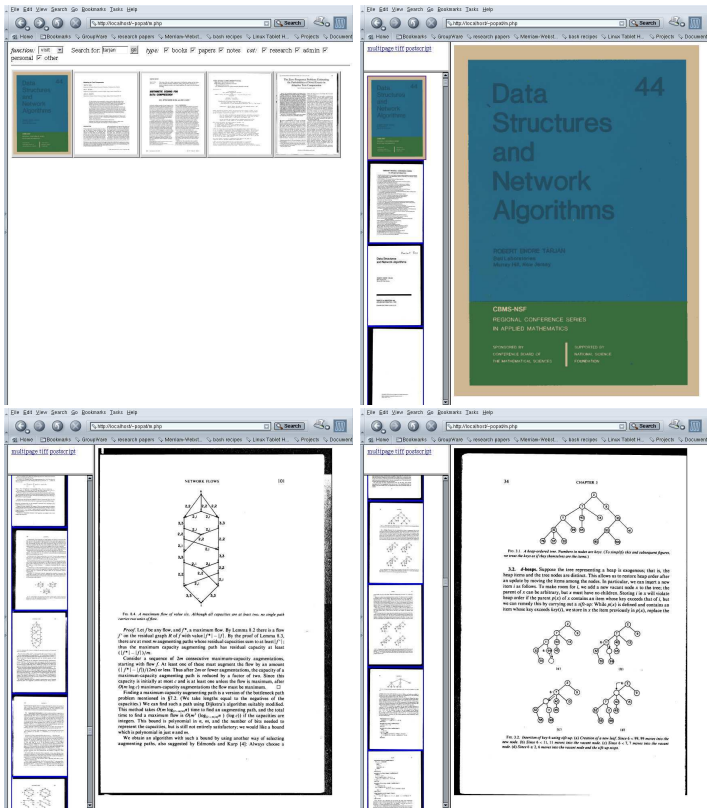
Figure 2: Retrieving and browsing a book in our first prototype. On a 600 MHz Intel Pentium III machine, each page displays in the order of 200 msec, without prior caching, irrespective of position in the book. Top left: search results for "Tarjan." Remaining images: page views as the thumbnail bar to the left is browsed. The functionality is similar to that provided by Adobe Acrobat Reader, but somewhat faster and without requiring a plug-in or separate viewing application.

is safer (and now feasible) to store virtually everything, taking advantage of plummeting hard disk costs, and shifting the burden of filing onto a well-designed retrieval process. So in addition to the day-to-day tasks of looking up research papers and notes, examples of unanticipated questions easily answerable with the UpLib include "what diameter contact lenses did my optometrist in Massachusetts prescribe?" and "how much did we pay in un-reimbursed medical expenses in 2002?" The potential effect on filing practices and information availability of adopting a digital personal document library into daily work-flow appears to be significant.

We feel that these considerations mandate the following design choices in a personal library system, and we attempt to fulfill them with the UpLib system:

- *Universality:* Documents can originate from any source and in any initial format;

- *Availability:* Documents – even large books – can be easily found, browsed, and read with a standard Web browser, and documents are still accessible when the personal library is not running;

- *Extensibility:* Functionality can be easily added to handle new types of search, conversion, organization, or access;

- *Searchability:* Documents can be retrieved using a natural combination of full-text retrieval over metadata and category information with visual search among initially retrieved thumbnail arrays to achieve a comfortable balance of precision, recall, and user involvement;

- *Scalability:* Tens of thousands of documents can be accommodated without reducing the system's responsiveness; and

- *Security:* Documents such as personal medical and financial records can be stored safely.

The next section briefly reviews other work in this field; those already familiar with this work may wish to skip directly to Section 3, where we begin the description of the two prototypes of an UpLib system that have been built, as part of the PARC Productive Reading effort. We describe these prototype systems in three functional stages: capture, document normalization and storage, and retrieval and use of the stored documents. Finally, we describe future work that we expect to do with the UpLib system of personal document management.

2. PRIOR WORK

The term "digital library" is broad enough to preclude a sensible exhaustive listing of *all* digital library systems; indeed the the World-Wide Web itself has emerged as a de-facto digital library of a general sort. It is therefore advantageous for our purposes to narrow the sense of the term *digital library system;* here we take it to mean: an integrated set of software, hardware, and protocol elements that together provide a means of storing, managing and accessing documents in digital form. All such digital library systems will have some degree of commonality among their functionalities; still, they can and do differ considerably in the specifics of their targeted end-use, their emphasis of which features are important, their detailed design choices, and their implementation. The main features and design choices behind several existing digital library systems were recently summarized by N. Fuhr et al. [10]. The list of systems presented there is by no means exhaustive, but the features and design choices appear to be representative of the current technology and practice.

Digital library systems can be categorized along several dimensions, e.g.: proprietary versus open; supporting large-scale versus limited collections; supporting many users simultaneously or few. Here we are primarily concerned with supporting the use and management of documents encountered in a person's or small group's day-to-day activities, in the absence of a system administrator. Therefore, the system must be both robust and lightweight. The system is also intended to be adaptable to evolving or newly recognized needs, and to serve as a research platform upon which to test new ideas or dovetail with complementary technologies in computer-enhanced reading and visualization. Therefore, extensibility, especially by non-guru users, is important to us as well. Proprietary document-management systems — both enterprise-level systems such as Documentum

(http://www.documentum.com/), and consumer-grade systems such as PaperPort (http://www.scansoft.com) — do not meet these requirements for ease of extensibility. There are few extant digital library systems we know of that are plausible fits to our requirements; the two that come closest are the *Personal Libraries* system by Wilensky at U.C. Berkeley [22] and the *Greenstone* open-source system by Witten et al [23].

The Berkeley *Personal Libraries* system is premised largely on a distinction between *collection* and *repository*; the former being a specification of some set of documents analogous to a list of pointers, and the latter embodying the "place" where individual documents are kept, or more precisely, an means of storing and retrieving individual documents. It provides cataloging and full-search capability through the *Cheshire-II* (http://cheshire.lib.berkeley.edu/) system. The *Personal Libraries* system is generally well-suited to the types of document storage and access tasks encountered in an academic setting, such as the provision of collections of research papers produced by or of interest to a research group, or the compilation of selected papers for course readers. It has provision for extensibility in at least two ways: by providing a customized style sheet that governs the display of search results, and by adding access functionality through a *Multivalent* client [17]. It also has a reasonable security/privacy mechanism, intended to address digital rights issues, but equally suitable to the protection of private personal data. However, from the descriptions we have of the system, it is not clear that its core functionality at the server end can be easily extended, particularly by non-expert developers. (Extensibility on the client side is provided through use of *Multivalent*). An instance of this system — to our knowledge, the main instance — can be accessed at **http://elib.cs.berkeley.edu/pl/**.

The *Greenstone* system is a public, extensible open-source project intended to grow in functionality as people contribute to it, to support interaction with large-scale multimedia collections. It seeks to provide a uniform framework for both searching and browsing, while providing for customizability via configuration files. It defines a specific format called *GML,*, a variant of HTML, into which documents must be converted in order to store them in a collection. *Greenstone* builds on the earlier *Managing Gigabytes* [24] programs, and like the Berkeley *Personal Libraries* system, takes advantage of ideas from earlier work such as *Dienst* [9] (see below) and *Harvest* [6]. Compared to the system we present here, the *Greenstone project* is relatively heavyweight — it takes pains to be scalable to very large scale collections, and its multimedia document model (and concomitant specialized GML format into which documents must be imported) are much more general than is needed for the types of use we are concerned with here. Moreover, we are concerned in our work that for people to adopt the system into their practice, it must allow them to retain any existing storage and access practices they may have individually evolved alongside the additional functionality provided by the system, rather than require them to commit wholly and irrevocably to an unfamiliar system whose future cannot be certain to them.

Dienst [9] was a relatively early web-based document storage, search, and access system intended for the dissemination of technical reports. *Dienst* saw fairly broad deployment through the mid-to-late 1990s, particularly among computer science departments in U.S. universities. Its strength was in its strong leveraging of the then-emerging World-Wide Web, with its standard protocols and its broadly deployed clients in the form of browsers. *Dienst* in turn had been influenced in part by an early document infrastructure system developed in our laboratory at PARC in the early 1990s called *System 33* [18]. *Dienst* relied on standard HTML forms for user input, the built-in cability of browsers to display images for its output, and the Common Gateway Interface ("cgi-bin") mechanism for executing its storage, search, and retrieval functions. As such it was even closer in its conception and design to UpLib (particularly to the first prototype of UpLib) than the two more "modern" systems reviewed above. But unlike UpLib, it was not designed as a *personal* digital library system, i.e., one to be easily administered and extended by its users.

Two other research efforts — *Haystack* [1] from MIT and *Presto* [11] from PARC — should be mentioned here as well, primarily for their conceptual relationship to our work with respect to personalization and the role of metadata. *Haystack* emphasizes personalization primarily by adapting the document search mechanism in a manner that is informed by the user's interests, as manifested by the contents of the documents that the user already has in his repository. The document collection and metadata are represented as a large graph, part of which is kept in fast memory, and the entirety of which is stored for use over multiple sessions through the use of persistent database technology. Relative to the system described here, *Haystack* is less image-centric and is aimed at handling a larger variety of personal documents including email, task lists, and appointments. Metadata plays a key role, and recent work on *Haystack* has emphasized its methods for the robust and effective representation and communication of metadata [13].

Presto [11] also emphasizes the role of personalized metadata in a filesystem, by eschewing a traditional location-in-hierarchy scheme for document organization and storage in favor of one based on a flexible combination of metadata elements. Collections of documents become dynamic, essentially defined by conditions satisfied by their metadata, and multiple categorization of documents becomes natural and automatic (a property it inherited from the earlier PARC project *Babar*). While our emphasis is on a repository and not a filesystem, our notion of *collection* is similar in spirit to its counterpart in *Presto*.

3. DOCUMENT CAPTURE

Document capture and retention is done for a number of different purposes. One study of information capture proposed 10 reasons why documents are captured [8]. Of these, half seem directly related to personal document libraries, while four of the remaining five are about collective document use in small groups. In addition, this study observed that 65% of the paper documents being captured were standard letter-size, and of those, 5/8 were only one page in length, while only 6% were longer than 3 pages in length. Many of the other documents were handwritten notes or newspaper clippings.

Documents are similarly captured by various means and in various formats. The proliferation of small cheap digital cameras suggest that they will also be used extensively for informal document capture of small paper documents, as well as for taking pictures and movies. Effective use of these
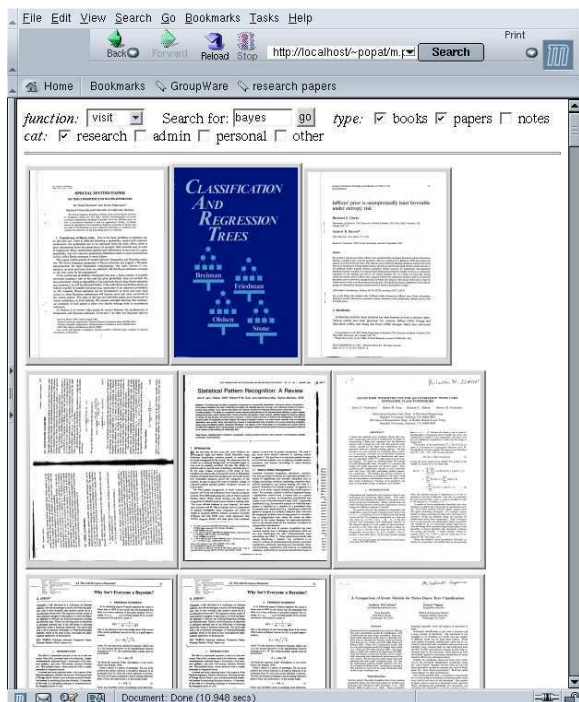
Figure 3: Results of a search in our first system, implemented in PHP4 and calling *htsearch* through the *cgi-bin* mechanism. The search results from *htdig* were parsed by the PHP script to form the names of the precomputed thumbnail images, which were then used to make HTML4 image buttons for the retrieved documents. Clicking on one of the retrieved-document buttons had an effect determined by *function*.
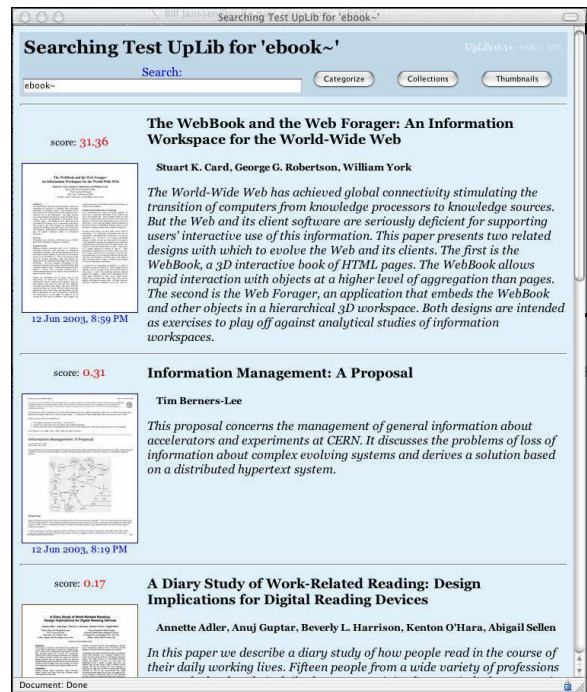

Figure 4: Search results in the second system. Output from *Lucene* is mapped by the Python daemon to document descriptions consisting of the scores, thumbnails, and summaries of the documents; the icons similarly serve as HTML links to retrieve the associated document. A thumbnails-only display of results is also available.

documents requires a number of automated image enhancement steps, to de-warp and otherwise clean up the document image. A number of papers have already discussed these problems and potential solutions, such as the CamWorks paper [16]. In addition, the increasing availability of documents in electronic forms such as HTML and PDF makes it clear that these formats have to be supported as well in a personal document library. Indeed, Web "clippings" may come to replace conventional newspaper clippings.

Our current systems allow the use of digital cameras or scanners, for paper documents, and document converters, for electronic documents. Scanning can be done either manually or automatically with systems such as Xerox's Flowport, which mails a PDF file containing the scanned document to the user. Document converters such as PARC's System 33 [18] and documents.com allow us to produce TIFF or text versions of formats such as PDF, HTML, or Microsoft Word. In general, we feel that the system should be flexible enough to support any mode of user document capture, and we expect a large number of custom capture tools to aggregate around a successful document library technology, each supporting a particular workflow microculture. For instance, we should be able to "print to" an UpLib repository from an application, or drag a document onto the UpLib from a desktop view, or instruct an UpLib to accept a particular document from a command line. Tools to support each

of these modes of capture have in fact been constructed for our current system; they are not considered part of UpLib proper.

To encourage the proliferation of appropriate capture and input tools, we specify a *document folder* format which tools can deposit the captured document in. This consists of a directory containing three files, each of which is a *projection* of the document into a particular document space. The three projections that we are currently experimenting with are *text*, which is a file containing the symbolic text of the document, if it has any; *page image*, which is a multi-page TIFF file containing full-color page images of the document; and *metadata*, which is a file containing metadata properties of the document, represented as IETF RFC 822 "unstructured field" headers, and tagged for character set and language information as specified in RFCs 2231 and 2047. The actual metadata fields used are up to the user, though the system makes good use of certain fields such as "Title" and "Date" if they are present. In addition to these three files, the original digital version of the document is copied to a sub-folder of the directory, regardless of format. The directory is then passed to the UpLib repository daemon via an HTTP POST operation.

We also provide certain building blocks, which can be used to assemble particular capture tools. For example, some formats have simple and effective ways of obtaining the text of a document, such as the *pdftotext* tool for PDF. However, many scanned formats require the use of optical character recognition. We provide an OCR Web service, which when
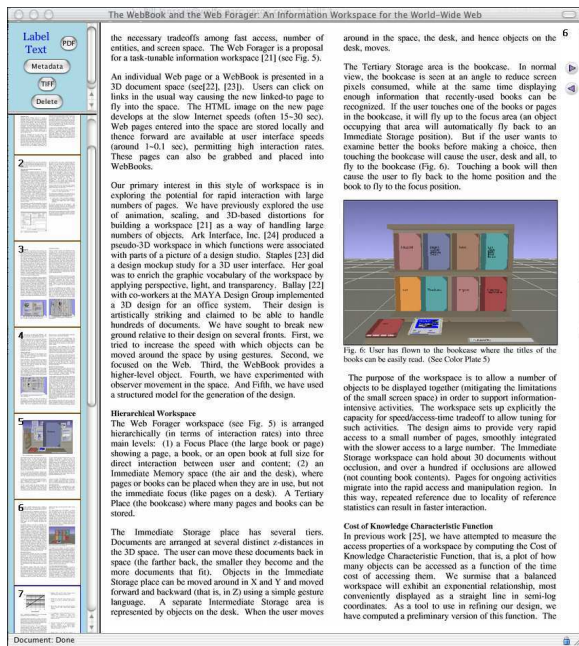
Figure 5: The page view is computed for a screen at least 1024 pixels high, and is quite readable as the image has been properly anti-aliased. The small page icons on the left-hand side are buttons providing direct access to that page; when used on the 768 × 1024 pixel screen of a Tablet PC, they are partially occluded, but the page numbers are still visible.
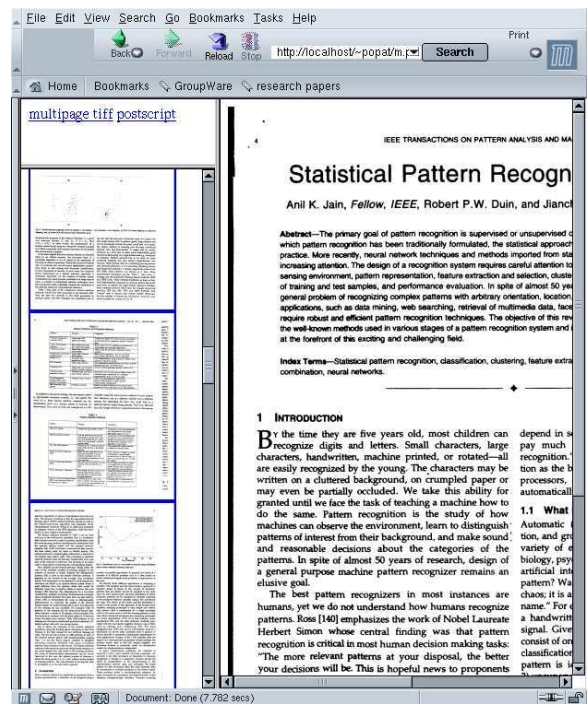

Figure 6: An example of insufficient window size, as would be the case on a low-to-moderate-resolution display. The need to scroll, especially horizontally, severely impedes the reading process. As mentioned in the text, the parallel PARC UbiText project may offer a solution, and may be adopted in UpLib for small-screen devices.

called with a set of images will try various OCR scans on the document. The output of these scans are compared to determine the best one by minimizing the empirical per-character entropy with respect to a character-5-gram language model (currently for English, but easily re-trainable on any other language representable in UTF-8). The best result is then returned to the caller. This service can be easily integrated into various image-oriented capture tools to produce the text projection of a document.

4. NORMALIZATION AND STORAGE

In many prior systems of this sort, a certain amount of user anxiety was caused by not allowing the user to control the location of the data saved by the systems; indeed, some of these systems did not allow the user to *know* where his documents were stored. We address this in UpLib by requiring the user to specify a directory to be used as the root of a particular repository. This allows the repository storage to be integrated with the user's normal backup procedures, and allows direct access to the document storage in case the UpLib server for some reason can't be run.

When a new document folder is passed to the UpLib system, it is assigned a unique name based on the current time, and placed in a "staging area" in the repository to undergo a process we call *normalization*, which involves producing other versions of the document, and generating more metadata about the document. Certain parts of this process are standard across all UpLib repositories, and others may be specific to a particular user or repository.

As part of the standard normalization, we require that each document be pre-rendered in several different forms for efficient search and presentation. Our current system requires that a file containing the symbolic text of the document be present, that thumbnails of each page of the document (in two different sizes) be present, and that a standard set of HTML files for accessing the document be present in a subdirectory of the document folder. To accomplish this, we use a sequence of "ripper" agents, which examine, in turn, the document in the staging area and produce alternate versions of it or augment the metadata information for it. Each ripper agent is an instance of a class which implements the *Ripper* interface; a default sequence of rippers is defined by the system, but this sequence can easily be augmented or modified on a per-repository basis through the standard extension mechanism.

One standard ripper generates the thumbnails for each page of the document; in our current system a small thumbnail used for document overview is generated along with a full-screen image used for actual page presentation. Another ripper adds an automatically generated document summary to the metadata file in the document folder. A third generates a set of HTML views of the document, drawing on the previously generated summary and on the thumbnails for page views. Still another generates an index entry for the document in a full-text indexing system; our first prototype used ht://Dig [21] for this purpose, and our second uses Jakarta Lucene [20].

Since the ripper agents are run sequentially, each can take

Figure 7: The metadata form allows some metadata about a document to be examined or updated. The full text of the document is available at the top of the form, for cutting and pasting into the other fields. Controls at the bottom of the form allow metadata to be shared with other UpLib users who may also have this document in their repositories.

advantage of data generated by earlier agents, or operate independently. In addition, rippers can be run outside the scope of the document addition task, at user request. We envision a library of possible analysis engines, each performing some metadata extraction or pre-use caching operation, that can continuously "rove" over the contents of the library, making opportunistic improvements. For example, as researchers, we'd like to integrate simple research paper metadata extraction similar to that used in CiteSeer [5], which would only run on documents that sufficiently resemble technical papers. Other transformations might perform URL extraction from Web pages, headline and dateline extraction from newspaper clippings, face recognition on family photos, or mailing-list classification on email messages; a number of these are currently being implemented.

When the document has been normalized, the document folder is automatically moved from the staging area to a storage area, and is then often manually verified. An important benefit of the UpLib is that it allows paper to be discarded. Before an important paper document that has been entered into the system can be thrown away or shredded, the digital version must be verified for quality and retrievability. This verification stage is by far the most time-consuming aspect of entering documents into the system, but our experience has shown that it is not prohibitively burdensome. For retrievability, an end-to-end check is made for each newly entered document by attempting to retrieve using the query terms that come to mind, and if the retrieval fails, then those query terms are added to the "keywords" metadata property of the document. Missing pages are checked for in large documents by inspecting every 10 or 20 pages, and making sure that the page numbers are incrementing by the

Figure 8: The result of a search. At the bottom, a form allows the search to be saved as a named persistent query.

expected amount. Scanning problems such as excessive skew or bad exposure are checked for by quickly scrolling through the thumbnail bar. In addition, our standard Web interface allows the user to examine, and if desired modify, the metadata about the document at this point.

5. RETRIEVAL AND USE

Once a document has been stored in the repository, most access to it is with a Web browser over SSL-encrypted connections. Currently, there are two major interfaces to the repository: either a visual presentation of all the documents in the repository, in one of three formats, or full-text search over the textual and metadata projection spaces.

Examples of thumbnail overviews can be seen in Figures 1 and 3. A number of document icons, representing the most recently used N documents, are presented; N is a user-configurable parameter. This is convenient for a small number of documents, and tractable for a somewhat larger number. However, more sophisticated information management techniques need to be used for the number of documents we envision a typical repository as having. There is a great deal of work going on in this area which we expect to benefit from. We are currently exploring incorporation of a time-based organization scheme, such as a *perspective wall* [14], or perhaps a hierarchical clustering scheme [3], [12]. In addition to the thumbnail overviews, documents can be shown with their abstracts, in a fashion similar to that shown in Figure 4, or as just a list of document titles. In any of these displays, clicking on the document will take the user to the *page-reader view* of that document.

However, in a large database of documents, searching may be the primary way of finding documents. Our prototypes allow a search string to be specified by the user, using the particular query syntax supported by the underlying full-text indexing engine. This string generates a set of document matches, and the system then needs to indicate those matches to the user in some way. Figure 4 shows one way of doing this, implemented in our second prototype. Each result consists of three parts: the query relevance score, the thumbnail image of the first page of the document, and a summary of the text of the document. As with the thumbnail overviews, clicking on any result takes one to the actual document in the page-reader view.
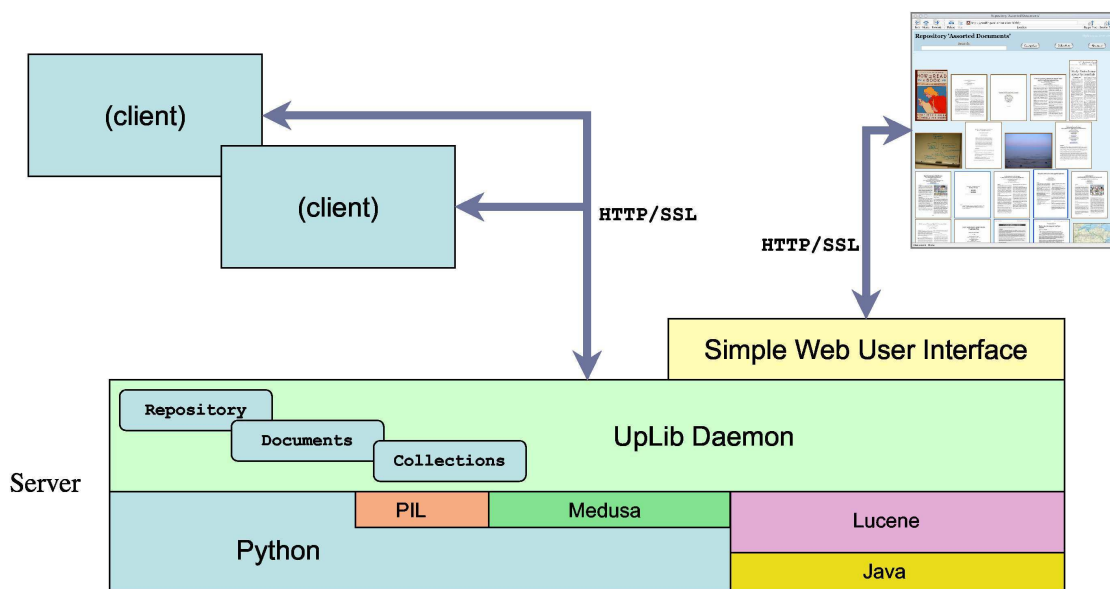
Figure 9: Using a client-server architecture with secure HTTP as the central communication paradigm enables a wide range of usages.

The Lucene system used for full-text indexing in the current version of UpLib treats each document as a cluster of text name-value pairs called *terms*, and can create an index for any number of the terms. This model is remarkably powerful; various metadata fields can be indexed separately from the main text of the document, which in turn allows Lucene to provide some of the capabilities of a full relational database. Searches can be run over a default set of terms, or over specific terms such as "Authors" or "Keywords". In addition, a number of advanced search features are available, such as phrase searching, fuzzy word matching, word colocation, range searches, and Boolean combination of search parts. We currently index a small number of standard metadata fields; this set is easily user-configurable, as is the set of default terms which unqualified searches are applied to.

Searches can be assigned a name and made persistent as a *collection* (see Figure 8). The set of persistent query-based collections can be accessed with a control button at the top of any of the repository overviews or search result forms. Opening a persistent query will cause the query to be rerun, and the current matching documents to be presented. We have found the speed of the Lucene engine we are currently using to perform searches to be more than acceptable for this purpose, even on collections comprising several thousand documents.

The page-reader view (see Figure 5) of a document is the other primary interface provided in the simple Web interface of UpLib. Accessing this view from a document icon or search result is very quick, as all components of the view are pre-computed. The view consists of three frames: a control area, a column of thumbnails of all the pages of the document, and a larger anti-aliased image of a particular page. The anti-aliased page image presentation is remarkably effective for reading, even for documents that are originally not in "virtual paper" form, i.e. Web pages. It provides simple page-turning controls at the right side of the page,

but any page can be accessed immediately from the column of page thumbnails. Above the column of page thumbnails is the control area, which normally includes buttons to retrieve PDF, TIFF, or text versions, delete the document (a confirmed delete, of course), and retrieve or edit document metadata; this control area is user-configurable, so that additional actions on the document can easily be added. In some cases, once it is determined that a document is of interest, it is printed out for more portable reading; the printed version is eventually discarded for recycling.

The second author has used the first prototype of this system to eliminate one large 4-drawer filing cabinet from his home, and one of about the same size from his office. He currently has about 2,000 documents in his UpLib, and accesses it daily. On average, he adds about 2.5 new documents to the system per day. We have also tried some scaling experiments with approximately 3500 documents, with no noticable decline in performance.

6. ARCHITECTURE

There are a number of ways to build systems of this type. Our current focus is on a client-server model, with a thin client built mainly with the "toolkit" defined by HTML and ECMAScript, communicating over a secure RPC channel to a server back-end written in Python that performs task-specific actions. The simple client will thus "run" in any modern Web browser, which, in the spirit of Dienst, we see as a significant advantage. The server is implemented as a daemon running on a user's computer with the user's identity, exporting functionality to the client via the standard HTTP/SSL protocol. In addition to servicing RPC calls from the client (or clients), it also performs the document normalization tasks described above, and supports metadata-mining agents implemented as separate threads of control, which continually work on improving the metadata for the repository.

User code can be loaded automatically at startup or while the daemon is running to introduce new ripper agents or UpLib "actions". Users can define new actions by adding Python modules to any directory on their "actions-path", and invoke any action by visiting an appropriate URL of the form "https://*host:port*/action/*module*/*action*". Parameters to the action may be passed as the query portion of the URL. The action code is loaded automatically when the URL is invoked in the server, and re-loaded automatically if the action module changes. Actions operate on three kinds of object, the *Repository*, *Document*, and *Collection* types. User code can create new subtypes of these object types and use them.

Additional information about the document is stored in either the document's metadata.txt file, or in subdirectories of the document folder. Metadata about the whole repository is stored in a top-level directory of the repository, devoted to that purpose. In general, information is stored in flat files as structured text of various sorts, to be easily accessible when the daemon is not running. Where appropriate, this text form is XML.

To enhance security, we use a variety of mechanisms. The "staging area" used in the normalization phase is within the document repository, so that partially-completed documents are not exposed to additional risk. All communication with the UpLib daemon is through SSL-encrypted HTTP connections, and appropriate HTTP caching directives are used to instruct Web browsers not to store pages fetched from the repository locally. Each repository may have a separate pass-phrase associated with it. The pass-phrase is never used as a command-line argument or stored on a filesystem; instead, a SHA-1 hash of the passphrase is used. The UpLib daemon issues short-lived secure cookies to the browser when the user logs into a repository, and these cookies are never stored to a filesystem. The repository data may be stored in an encrypted filesystem, and we recommend so doing for portable computer systems such as laptops.

We use a number of open-source packages in our latest implementation, including the **Python** language, the **Python Imaging Library** (PIL), the **Medusa** web server framework for Python, various tools from the **libtiff** package, the **Lucene** full-text indexing system from the Apache Jakarta project, and **Ghostscript**. Some of our capture tools use the **xpdf** package, and we've found the **curl** tool very useful in working on the system.

7. FUTURE WORK

At PARC there are currently a number of projects aimed at extracting various kinds of metadata from documents or document collections. They include generation of document co-reference graphs, measuring the "authority" of web pages, determining reading order of words in a document by page image analysis, and detection of topic shifts in the text of a document. We hope to incorporate results from most of these projects into the UpLib as rippers. In addition, we are looking at automatic generation of a *syntopicon* [2] for documents in a specific repository.

Extension of UpLib to a multi-user system for small groups or departments is an interesting possibility. The security framework used in UpLib means that every request is already authenticated. Multiple users could be supported simply by allowing multiple passwords and identities, instead of a single one. The major additional work would be in designing an access control language to allow different identities different levels of control over the system, and its extensions. Since all accesses and actions go through a single dispatching point in the UpLib daemon, implementing such an ACL would be straightforward.

There is always additional user interface work to pursue. An alternative to our current HTML/ECMAScript front end supported by a Web browser is one written specifically for our application, either as a stand-alone program, or perhaps as a Java applet. Our intention is to avoid this if possible, both to reduce barriers to adoption and to provide resistance to obsolescence. A more palatable variation on this approach is to use HTML/ECMAScript in conjunction with a custom Web browser, such as the highly portable *Multivalent Browser* [17] being developed at Berkeley, which supports a exceptional range of document interactions, such as annotation and "magic lenses" [4]. Direct annotation and snippet collection, as demonstrated in the FX-PAL XLibris system ([19]) are additional capabilities that seem appropriate to a Tablet PC version of the page-reader view, and we intend to incorporate such functionality.

We'd also like to support use of documents in smaller display contexts. The availability of small PDA devices with high-resolution color screens and Bluetooth automatic connectivity, such as the Palm Tungsten T, and other devices with very large storage capacity, such as the Apple iPod, suggest that pocket-size devices with both of these capabilities are likely in the near future. A pocket-size wirelessly available "container" holding all of your personal documents, as well as serving as an MP3 player and PDA, seems a likely outcome of these trends, and we intend to run our UpLib on these devices when they become available.

But interacting with documents on these devices presents special challenges [15]. Effective use of personal documents on these devices will require new ways of dealing with images, particularly images of text. One promising approach, PARC's UbiText [7], offers a way of re-flowing (essentially re-typesetting) images of pieces of the text (characters and words), and we hope to use it in future versions of our system.

In the longer term, we expect to release an open-source version of the system externally. Currently, the largest obstacle to doing this is the unavailability of a reliable open-source OCR engine, though some starts have been made. We would like to see more work done to remedy that problem.

8. CONCLUSION

We have described the objectives, design, and implementation of a universal personal digital library that emphasizes image-domain representation of documents, and is largely based on open-source components. Specifically, the architecture and functionality of two prototypes has been described. The first prototype of the UpLib system has been adopted into the daily work-flow of one of the authors over a period of about two years, suggesting that the system is both usable and useful (although a formal user study will be required to determine the effect of an UpLib on current work practices). Several colleagues have begun to adopt the system into their own work-flow. The second prototype greatly facilitates this; it is more powerful, more robust, more extensible, and easier to use by a novice. We expect that it will be adopted by a growing user community within PARC,

and that it will continue to improve in usability and functionality as a result.

9. REFERENCES

[1] E. Adar, D. Kargar, and L. A. Stein. Haystack: per-user information environments. In *Proceedings of the eighth international conference on Information and knowledge management*, pages 413–422. ACM Press, 1999.

[2] M. J. Adler and C. V. Doren. *How to Read a Book*. Touchstone Books, revised edition, 1972.

[3] B. B. Bederson. Photomesa: a zoomable image browser using quantum treemaps and bubblemaps. In *Proceedings of the 14th annual ACM Symposium on User Interface Software and Technology*, pages 71–80. ACM Press, 2001.

[4] E. A. Bier, M. C. Stone, K. Pier, W. Buxton, , and T. D. DeRose. Toolglass and magic lenses: The see-through interface. In *Proceedings of SIGGRAPH '93, ACM Computer Graphics Annual Conference Series*, pages 73–80, Anaheim, California, August 1993.

[5] K. Bollacker, S. Lawrence, and C. L. Giles. CiteSeer: An autonomous web agent for automatic retrieval and identification of interesting publications. In K. P. Sycara and M. Wooldridge, editors, *Proceedings of the Second International Conference on Autonomous Agents*, pages 116–123, New York, 1998. ACM Press.

[6] C. Bowman, P. Danzig, D. Hardy, U. Manber, M. Schwartz, , and D. Wessels. Harvest: A scalable, customizable discovery and access system. Technical Report CU-CS-732-94, University of Colorado, Boulder, Colorado, 1994.

[7] T. M. Breuel, W. C. Janssen, K. Popat, and H. S. Baird. Paper to PDA. In *Proceedings of the 16th IAPR Internation Conference on Pattern Recognition*, pages 467–479, Quebec City, Canada, August 2002. IAPR.

[8] B. A. T. Brown, A. J. Sellen, and K. P. O'Hara. A diary study of information capture in working life. In *Proceedings of 2000 ACM Special Interest Group on Computer-Human Interaction (CHI2000)*, pages 438–445, 2000.

[9] J. R. Davis, C. Lagoze, and D. B. Krafft. Dienst: Building a production technical report server. In *Proceedings of the 1995 Advances in Digital Libraries Conference*, pages 259–271, McClean, Virginia, May 1995. IEEE Computer Society, IEEE.

[10] DELOS Working Group 2.1. Survey on existing digital library systems, January 2001. http://www.sztaki.hu/delos_wg21.

[11] P. Dourish, W. K. Edwards, A. LaMarca, and M. Salisbury. Presto: an experimental architecture for fluid interactive document spaces. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 6(2):133–161, 1999.

[12] A. Graham, H. Garcia-Molina, A. Paepcke, and T. Winograd. Time as essence for photo browsing through personal digital libraries. In *Proceedings of the Second ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL'02)*, pages 326–335, Portland, Oregon, July 2002.

[13] D. Huynh, D. Karger, and D. Quan. Haystack: A platform for creating, organizing and visualizing information using rdf. In *Proceedings of the Semantic Web Workshop, The Eleventh World Wide Web Conference 2002*, 2002.

[14] J. D. Mackinlay, G. G. Robertson, and S. K. Card. The perspective wall: detail and context smoothly integrated. In *Proceedings of the SIGCHI conference on Human factors in computing systems: Reaching through technology*, pages 173–176, New Orleans, Louisiana, 1991. ACM.

[15] C. C. Marshall and C. Ruotolo. Reading-in-the-small: A study of reading on small form factor devices. In *Proceedings of the Second ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL'02)*, pages 56–64, Portland, Oregon, July 2002.

[16] W. M. Newman, C. R. Dance, A. S. Taylor, S. A. Taylor, M. Taylor, and T. Aldhous. Camworks: A video-based tool for efficient capture from paper source documents. In *Proceedings of the IEEE Conference on Multimedia Systems*, volume 2, pages 647–653, 1999.

[17] T. A. Phelps and R. Wilensky. The Multivalent browser: a platform for new ideas. In *Proceedings of the 2001 ACM Symposium on Document Engineering*, pages 58–67, Atlanta, Georgia, 2001. ACM. See also http://www.cs.berkeley.edu/ phelps/Multivalent/.

[18] S. Putz. Design and implementation of the system-33 document service. Technical Report ISTL-NLTT-93-07-01, Xerox Palo Alto Research Center, 3333 Coyote Hill Road – Palo Alto, CA 94304, 1993.

[19] B. N. Schilit, M. N. Price, and G. Golovchinsky. Digital library information appliances. In *Proceedings of Digital Libraries '98*, Pittsburgh, PA, June 1998. ACM.

[20] The Apache Project. Jakarta Lucene Overview, 2003. See http://jakarta.apache.org/lucene/docs/index.html.

[21] The ht://Dig Group. ht://Dig – Internet search engine software, 2003. See http://www.htdig.org/.

[22] R. Wilensky. Personal libraries: Collection management as a tool for lightweight personal and group document management. Technical Report SDSC TR-2001-9, San Diego Supercomputer Center, 9500 Gilman Drive – La Jolla, CA 92093-0505, 2001.

[23] I. H. Witten, R. J. McNab, S. J. Boddie, and D. Bainbridge. Greenstone: A comprehensive open-source digital library software system. In *Proceedings of the Fifth ACM International Conference on Digital Libraries*, 2000.

[24] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes*. Morgan Kaufmann, 2nd edition, 1999.