Working Paper Series
ISSN 1170-487X

Managing multiple collections, multiple languages, and multiple media in a distributed digital library

by Ian H Witten, Rodger McNab, Steve Jones, Sally Jo Cunningham, David Bainbridge, Mark Apperley

Working Paper 98/9
May 1998

Department of Computer Science
The University of Waikato
Private Bag 3105
Hamilton, New Zealand

Managing multiple collections, multiple languages, and multiple media in a distributed digital library

Ian H. Witten, Rodger McNab, Steve Jones,
Sally Jo Cunningham, David Bainbridge, Mark Apperley

Department of Computer Science, University of Waikato,
Hamilton, New Zealand.

ihw@cs.waikato.ac.nz

Phone (+64) 7 838–4246, fax (+64) 7 838–4155

Abstract Managing the organizational and software complexity of a comprehensive digital library presents a significant challenge. Different library collections each have their own distinctive features. Different presentation languages have structural implications such as left-to-right writing order and text-only interfaces for the visually impaired. Different media involve different file formats, and—more importantly—radically different search strategies are required for non-textual media. In a distributed library, new collections can appear asynchronously on servers in different parts of the world. And as searching interfaces mature from the command-line era exemplified by current Web search engines into the age of reactive visual interfaces, experimental new interfaces must be developed, supported, and tested. This paper describes our experience, gained from operating a substantial digital library service over several years, in solving these problems by designing an appropriate software architecture.

Keywords Digital library architecture, digital library interfaces, multimedia systems, multilingual interfaces, information management, distributed information retrieval

Managing multiple collections, multiple languages, and multiple media in a distributed digital library

As the capabilities of distributed digital libraries increase, managing organizational and software complexity becomes a key issue. How can collections and indexes be updated without impacting queries currently in progress? When users are offered a choice of different (natural) languages to communicate with the system, how can the software cope with all the various versions of the text that is displayed? With multi-media collections, how can the different kinds of search engines that are required to search various media types be accommodated within a uniform, extensible software framework? How can several experimental user interface clients for the same collections co-exist? How do these clients learn about new collections that are created in different physical locations?

The New Zealand Digital Library [1] currently offers about twenty separate collections of information (some are private but most are public); has interfaces in five different natural languages (English, French, German, Arabic, and Maori), offers the option of "text-only" Web pages as an alternative to the normal "text-and-image" presentation. In the library's present state this escalates to two hundred different combinations—twenty collections, five languages, two presentation versions—and the number continues to grow, multiplicatively. Moreover, two widely different search engines are incorporated, for different media types.

Managing and extending the library presents severe logistic challenges because of the wide variety of different facilities offered by the various collections. Although they share a family resemblance, each collection in the library is quite distinctive and involves different browsing and index structures. Individual collections can be served by computers in different locations, even different countries. Text and menu items must be stored separately in each languages, and text-only interfaces—designed principally for the visually impaired—require their own Web page structure. Finally, most collections contain graphics and/or audio in addition to text. Handling different output media is reasonably straightforward. Input queries, however, are a different matter, and we have a musical collection that is truly multi-media in that a different kind of retrieval engine is used to access tunes based on sound sequences.

This paper describes the software structure that we have developed to cope with the complexity sketched above. We begin by illustrating the scope and variety of collections and interfaces offered in the digital library. We then discuss the notion of a "collection" and describe how we handle the addition of new collections, in different locations, dynamically, without requiring any centralized list. Next we turn to the multiple language aspect and explain how a novel flexible macro language, where macros can be accessed associatively, is used to manage interfaces to dozens of collections, each with its own distinct characteristics, and each presented in many languages. Then we look at the radically different search engines that are incorporated, and describe how the architecture we have developed is flexible enough to accommodate these within a uniform structure and communication protocol. Finally, we show how the same structure facilitates the development of experimental user interfaces. The fundamental goal that underlies this work is to minimize the

maintenance effort that must be spent to keep the system operational and continue to expand its offerings.

THE NEW ZEALAND DIGITAL LIBRARY

Operational for several years now, the New Zealand Digital Library presents an evolving amalgam of diverse types of information. The basic access tool is full-text searching, although some collections allow traditional catalog searching based on author, title, and keywords, and full-text search within abstracts. Our experience is that while the user interface is considerably enhanced when traditional library cataloging information is available, it is prohibitively expensive to create formal cataloging information for many electronically-gathered collections. With appropriate indexes, full-text retrieval can be used to approximate the services provided by a formal catalog.

The core of any library is the collections it contains, and several examples will indicate the variety and scope of the services that are provided. Figure 1 shows part of a collection of *humanitarian development* information put together by the Global Help Project. At this point a particular book has been reached, by either browsing or full-text searching. At the top is a hierarchical browser, with a picture of the book's cover; the lower part gives the text of one subsection. Half of the collection, including this particular document, is in the French language; most of the rest is in English, with smaller amounts in Kisuaheli (an African language) and one book in Portuguese.

Our collection of *computer science technical reports* contains 46,000 reports—1.3 million pages, half a billion words—extracted automatically from 34 Gb of raw PostScript. There is no metadata information: we have only the contents of the reports (and the names of the FTP sites from which they were gathered). Many different languages are represented. Using the German-language interface option, Figure 2 shows the page received in response to a query for the word *Boolesche*; this returns several German documents. Accents are supported by our system and are included in searches when specified: the incorrect display of umlauted characters in Figure 2 is due to deficiencies in the PostScript extraction process.

While this provides a useful facility for German-speaking users, the retrieval facilities have not yet been engineered into a truly multilingual system. Further requirements include language-specific stemming, and restriction of queries to documents of one language—so that, for example, an English-speaking numatist seeking information on a particular *coin* does not receive documents matching the French word for "corner". This in turn involves identifying the language in which each document (or paragraph, for multilingual documents) is written. Translingual information retrieval, where a user's single-language query retrieves related documents in different languages, is another matter altogether, and is not addressed by our system.

One collection that is expressly bilingual contains forty *historic New Zealand newspaper* titles, published between 1842 and 1933 in the Maori language or for a Maori audience, sometimes in parallel Maori and English versions. Collected on microfiche, these 12 000 images represent a significant resource for historians, linguists and social scientists, but their riches remain largely untapped because of the inherent difficulty of accessing, searching and browsing material in unindexed microfiche form. We are presently preparing the images for incorporation into the

digital library. Figure 3 shows the parallel English–Maori text retrieved from the newspaper *Te Waka Maori* of August 1878 in response to the query *Rotorua*, a small town in New Zealand. Searching is carried out on electronic text produced using OCR; once the target is identified the corresponding page image can be displayed.

The interface can be presented in English, French, German, and Maori. Moreover, interfaces in each of these languages are available for text-only Web browsers. In order to further test the multilingual capability we have recently built a small *Arabic demonstration* collection, illustrated in Figure 4. A standard Web browser plugin is used for character display and input (which, for the Arabic language, is in right-to-left order). We have modified the text storage and search mechanisms to accommodate the non-ASCII Arabic alphabet and, while we do not yet use the Unicode character representation, that is an obvious next step.

A project on local oral history demonstrates multimedia capabilities. The source material is audio tapes of interviews, summary transcripts of each, and photographs that illustrate episodes mentioned in the tapes. Timing information that links the transcripts to the tapes is extracted manually, and a textual description of each photograph is included. Full-text retrieval operates on each of these sources, separately or together, to retrieve extracts and pictures that match a given query. From there the appropriate segment of tape can be played, and related pictures viewed. Figure 5 shows some results from a query about *VE day*; the small gray panel controls audio playback of a sound bite or relevant section from a participant's interview.

A technically more significant multimedia capability is demonstrated by a retrieval engine for music. Based on a novel scheme for searching musical melodies, this matches sung (or hummed) input to a database of tunes. Figure 6 shows the response when a user sang the first eight notes of *Auld Lang Syne* as a query. The transcribed input appears at the top left; titles of similar items, ranked according to matching score, appear below. Any of the tunes may be selected for audio replay or visual display; one appears in the front window. At present, just one collection is based on this retrieval system, a database of nearly ten thousand international folk tunes (half a million notes) from various countries: North American, Irish, British, German, and Chinese. Tunes from each country can be searched separately. This collection demonstrates the need to encompass radically different search regimes within a single unifying architecture.

ORGANIZING COLLECTIONS

The fundamental object in our design is a *collection*, a set of information that belongs together and is offered as a service to users [2]. Existing collections vary from small demonstrations holding just a few documents (like the Arabic one) up to repositories containing 3.5 Gb of text (computer science technical reports) and ten million documents (a library catalog collection). They vary from text-only information to multimedia collections with pictures, sound, and notated music; from collections where the only metadata is the source URL and creation date to ones with full bibliographic records; from collections containing bibliographic surrogates to ones containing documents in their entirety.

All collections are composed of *documents*. Documents come in a variety of formats: we presently accommodate plain ASCII, PostScript, PDF, HTML, SGML, and Microsoft

WORD for textual documents; REFER, BIBTEX, and USMARC for bibliographic information; several different image standards, and various internal formats for musical information. Collections invariably undergo some *building process* to make them suitable for display, browsing, search and retrieval. This often involves converting the documents to a different format, and identifying subparts that require their own indexes. Amongst the current collections are indexes for complete documents, individual pages, paragraphs, articles, abstracts, authors, titles, subjects, publication details, references, and motions in meetings. The building process also involves preparing some statistics about the collection which are incorporated into a descriptive paragraph.

When collections are rebuilt, active users continue unaffected. The building process takes place in a separate file directory and, when complete, the directory is moved, in a single atomic operation, into the live file area. Of course, the query engine does not read index information afresh for each query: that would be too slow. Its file handles to the index information become stale when the new collection supersedes the old, causing it to terminate on the next read operation—whereupon the query process is restarted automatically. Current users are unaware of what has happened, except perhaps for a brief delay while the query process re-initializes: the next query they issue is run on the new version (and may return different results from the same query issued previously). The worst that can happen is that references on a query results page being displayed to the user become changed when the update occurs, resulting in the wrong document being returned. Although we could arrange to warn users of this, it has not been found necessary in practice; instead we plan to use persistent document identifiers to overcome the problem.

Normally, several collections run on the same server machine. However, if desired different collections can operate on different computers—all that is necessary is a copy of the collection server software running on each one. Collections are active objects; they receive and process messages such as search requests and document display requests. In order to avoid any centralized list of collections (and thereby reduce maintenance), each collection can be instructed to look for others in the same directory. Adding a new collection merely involves inserting it into a directory that holds an already-known collection; by convention, only one directory is used on any given machine.

When a user interface client process starts up (and at periodic intervals thereafter) it sends messages to a small list of collections that are coded into it, requesting information on other collections on the same machine. The information that is returned about each collection includes its *identifier*, the *host machine* it runs on, and the *port* it uses for communication. When a collection is added, its existence will (eventually) be noticed by every interface client, providing it is aware of another collection on the same machine. The automatic nature of the notification process greatly simplifies the administration involved in collection creation.

Although the system is distributed in that collections can run on different computers, searching is not distributed—each collection is served by just one machine. Cross-collection searching is not at present enabled. While fairly simple for collections on a single server, cross-collection searching across different servers would involve a distributed search process—which is addressed by some information retrieval systems

(e.g. [3]). Our goal is not to further the state of the art in information retrieval *per se*, but to provide a flexible structure into which the latest advances can be slotted.

ACCOMMODATING DIFFERENT LANGUAGES

The digital library presents itself to the user in various languages, and new ones can be included merely by supplying translations of all phrases used in the interface, along with those used as menu items. The code that produces Web pages is divorced from the text strings that appear in them. Moreover, text-only versions of the interface are provided in each language. The principal reason for these is to provide a more accessible web page layout for blind and partially sighted users, and for this we follow standard guidelines [4].

Building a Web interface that is flexible enough for this purpose is not as simple as it first appears. The text that appears on a page, and most images, are language-dependent. It is convenient to use a default language (English) whenever an item occurs that has not yet been translated. Appropriate variations must be made in text positioning and justification for languages, such as Arabic, that stretch across the page from right to left. Text-only pages often have quite a different structure from the corresponding text-and-graphics pages. The notion of a default is useful when defining standard page headers and footers that are overridden by some of the collections. Finally, it is convenient to be able to reference directory names and URLs symbolically, increasing the ease with which the system can be configured to different environments.

As Figures 1–6 illustrate, the desire for a consistent "house style" leads to a layout in which headers and footers are the same, with collection- and page- specific variations. Although each collection is different and has its own indexes, menus, buttons, descriptive text, and icons, all collections have query-specification pages, query-response pages, and retrieved-information pages. For an easily-maintained, easily-augmented interface, it is vital to abstract out this commonality and represent it in just one place.

To provide an appropriate level of abstraction, we design Web pages using a model that comprises three parts: structure, content, and reusable items. The *structure* provides a framework that can incorporate different versions of the content—for example, a page might contain some sort of heading, a main information-bearing part, and a footer with navigation aids. The heading in turn might have its own structure with a masthead and navigation aids at the top and an advertisement underneath. The *content* includes the principal information-bearing elements of the page—text, graphics, sound clips, and movies—abstracted away from the page structure into separate resources. Many content items are language-specific, and all resources pertaining to a particular language are collected together for ease of maintenance. *Reusable items* include pathnames of certain directories and frequently-used icons. Isolating these items renders Web pages more portable.

These requirements are met using a specially-designed flexible macro language. Macro languages are often used to assist with the maintenance of large collections of Web pages [5]. However, while traditional languages such as *m4* [6] do permit text strings to be named, the only support that they provide for dealing with a plethora of

different versions of a page is the conditional inclusion of text. This quickly becomes cumbersome as the number of options increases.

The basic innovation in the macro language is that it allows many variants of the same macro, each with different parameters. Macro resolution is done at expansion time by choosing the variant that best suits the environment represented by the page being generated. For example, a particular macro may have image and text variants (the latter for text-only pages); the same name will be specified in the page description and the appropriate variant sought when the page is composed. The macro language allows any set of parameters with any range of enumerated values to be specified. Most macros used in our page specification involve a combination of three parameters: collection name, whether the page is text-only or text-and-graphics, and language. If a macro that matches all three is found in the database, it is used. Otherwise, the closest match is used, where the three parameters have assigned priorities. For example, if a text-only version of a particular macro is sought in German, but the database contains just two relevant macros, a text-only version in English and a text-and-graphics version in German, then the latter is preferred.

Incorporating the notion of macro variants into the language allows essential maintenance operations to be supported. For example, we have an automatic scheme for keeping translations up to date, in which certain people are designated maintainers for particular languages. Macros that require multilingual variants are flagged. Each week an automatic process scans all macros to find those that are missing translations, or need their translations updated, and mails the appropriate maintainer with the text that needs attention.

DIFFERENT SEARCH MECHANISMS

Although some of the "multimedia" collections are searchable through textual descriptions of the media objects that are stored, in the more general situation different search mechanisms are required for different media types. Presently, two search and retrieval methods are included, one for text and the other for music, and a uniform architecture has been developed to accommodate them, and the vastly different document types that they involve. This structure is flexible and will be able to incorporate other retrieval mechanisms in the future—for example, an image collection accessed by content-based queries (e.g. [7]).

Text retrieval is accomplished using MG [8], a freely-available full-text retrieval system that makes efficient use of disk resources by storing the index, and the text that is indexed, in compressed form. Typically text compresses to 25% of its original size, and the compressed index occupies around 7% of the size of the original text. This leads to a total storage requirement for the indexed collection of approximately one-third of the size of the original text alone.

Music retrieval is accomplished using MR [9], a novel scheme for searching musical melodies: it matches sung (or hummed) input to a database of tunes. Options are provided to restrict attention to subsets of the database, to choose one of two matching algorithms, to match anywhere within a tune or beginnings only, to match using musical intervals or up-down-same contour, to ignore or take account of note durations, to transcribe using fixed tuning or try to adapt to overall drift in the user's intonation, to specify minimum rest and note lengths, and to specify the speed of the

music. The reason for many of the options is that users differ in their musical ability and in the accuracy with which they remember tunes, and it is necessary to provide flexibility to cater for this variation.

Each collection uses a particular search engine (or engines, in the case of songs that are searchable by words or music). The process of building a collection, referred to earlier, is heavily dependent on the mechanisms used to search it, because much of building involves creating indexing structures that will be used later for searching. Consequently, for each collection a *builder process* and corresponding *collection server process* are defined. The former dictates how it should be built; the latter gives access to all document retrieval and searching functions.

To incorporate different index styles, often with widely differing needs, builders and servers use a flexible, object-oriented program structure. Object classes with all the basic features needed for building, document retrieval, and searching are defined, but the functionality they provide is selectively overridden by defining subclasses for particular search and retrieval systems, and these are finally overridden by collection-specific subclasses that accomplish any tasks peculiar to a collection.

EXPERIMENTAL USER INTERFACES

Internet search engines and library on-line public access catalogs commonly provide textual query languages such as that defined by Z39.58 [10], yet the problems such interfaces present for casual users have long been recognized. We are still in the "command-line" days of information retrieval and have not yet progressed to truly reactive interfaces. To rectify this, we are developing several experimental highly interactive, graphical, direct-manipulation interfaces to the Library, using various implementation platforms. Figure 7 shows two examples: a Java applet that provides a graphical language for Boolean querying, and updates itself immediately as the query is modified, and a Tcl/Tk application for visualizing clusters of documents resulting from a sequence of queries. A communication protocol [2] defines how user interface software communicates with collections through network connections established with collection servers. These interfaces have been developed without modifying the collection server software, and without close code integration of user interface and server.

Collection servers recognize three types of message: requests for general information such as details of available collections, requests for document information such as summaries or the full text of documents, and requests for documents matching specified search criteria. User interface clients do not necessarily operate on any collection's host machine, and our Java implementations are platform independent. We have developed an application programmer's interface to a software library that contains routines to form and send requests to collections and manage the resulting responses. This makes it easy to integrate new interfaces, without any impact on collection servers. Both the standard Web interface and our experimental interfaces exploit the protocol, and different interfaces can co-exist without any difficulty.

Dynamic query interfaces, where displayed document result sets change immediately whenever query parameters change, form the focus of our user interface research. Here, operations must be executed quickly on locally cached document information. Figure 8 shows the dynamic querying architecture that we have implemented. The

protocol allows close control over when information is downloaded from the collection server into the cache, and how much is transferred. For example, in bibliographic collections items such as author, title, and keywords can be retrieved individually for single documents. Only the information that is desired is retrieved over the network. This improves response time, minimizes the size of the local data store, and allows information to be displayed incrementally as it arrives. Users can continue interacting while information is still coming in.

CONCLUSIONS

This paper has described our approach to the management of organizational and software complexity in a large, fully-operational, widely-used digital library system. Our fundamental goal is to minimize the effort that is required to keep the system operational and continue to expand its offerings. New collections can be built, easily, with their own distinctive features and indexes. They can be added at any time, on any existing server, while the system is running. New languages can be accommodated by translating the phrases and images used in the interface and incorporating these into the collection server software. Multiple media are easily incorporated, and are present in many collections. Search engines for non-standard media, and indeed non-standard search engines for text, can be slotted into the system with minimal extra effort. New highly-interactive interfaces can be implemented on the client side, using a variety of support technologies, and brought on-line while the library system is operating. The structure we have developed makes it easy to keep a digital library working and growing.

ACKNOWLEDGMENTS

We gratefully acknowledge Carl Gutwin, who developed one of the applications depicted in Figure 7. Many thanks are due to Stefan Boddie, Te Taka Keegan, Craig Nevill-Manning, and Lloyd Smith, who contributed to this work in various ways.

REFERENCES

1. http://www.nzdl.org/

2. McNab, R.J., Witten, I.H. and Boddie, S.J. (1998) "A distributed digital library architecture incorporating different index styles." *Proc Advances in Digital Libraries '98*, 36–45.

3. de Kretser, O., Moffat, A., Shimmin, T. and Zobel, J. (1998) "Methodologies for distributed information retrieval." *Proc Int Conf on Distributed Computing Systems*, Amsterdam; May.

4. http://trace.wisc.edu/docs/html_guidelines/htmlgide.htm

5. Peel, A. (1996) "HTML macros—Easing the construction and maintenance of Web texts." Technical Report 4-96, University of Kent, Canterbury, UK.

6. Kernighan, B.W. and Ritchie, D.M. (1979) "The m4 macro processor." In *Unix Programmer's Manual 2*, Technical Report, Bell Laboratories, Murray Hill, NJ.

7. Smith, J.R. and Chang, S.-F. (1996) "VisualSEEK: a fully automated content-based image query system." *ACM Multimedia Conference*, Boston, MA.

8. Witten, I.H., Moffat, A., and Bell, T.C. (1994) *Managing Gigabytes: Compressing and indexing documents and images*, Van Nostrand Reinhold, NY.

9. McNab, R.J., Smith, L.A., Witten, I.H., Henderson, C.L. and Cunningham, S.J. (1996) "Toward the digital music library: tune retrieval from acoustic input." *Proc Digital Libraries '96*, 11–18.

10. National Information Standards Organization (1995) *Z39.58: Common command language for online interactive information retrieval.* ANSI/NISO, Bethesda, MD.

LIST OF FIGURES

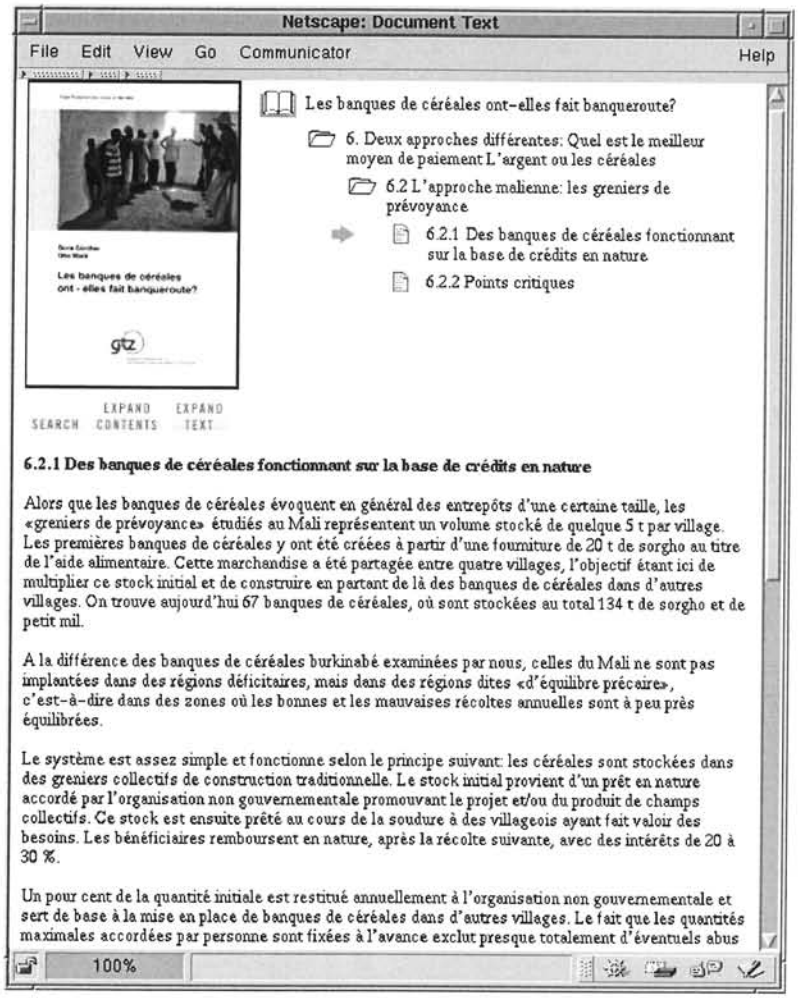

Figure 1 Browsing the *Humanitarian development* collection

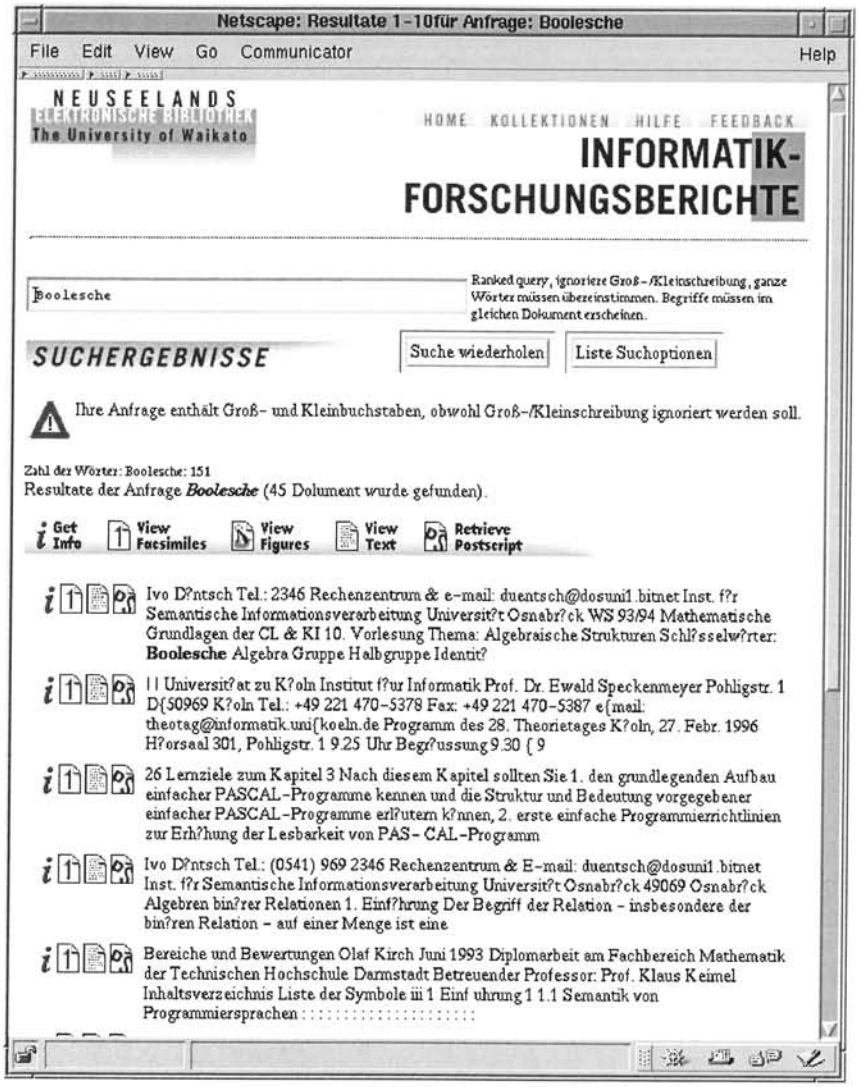

Figure 2 Result of a query to the *Computer Science Technical Reports* (with German interface option)

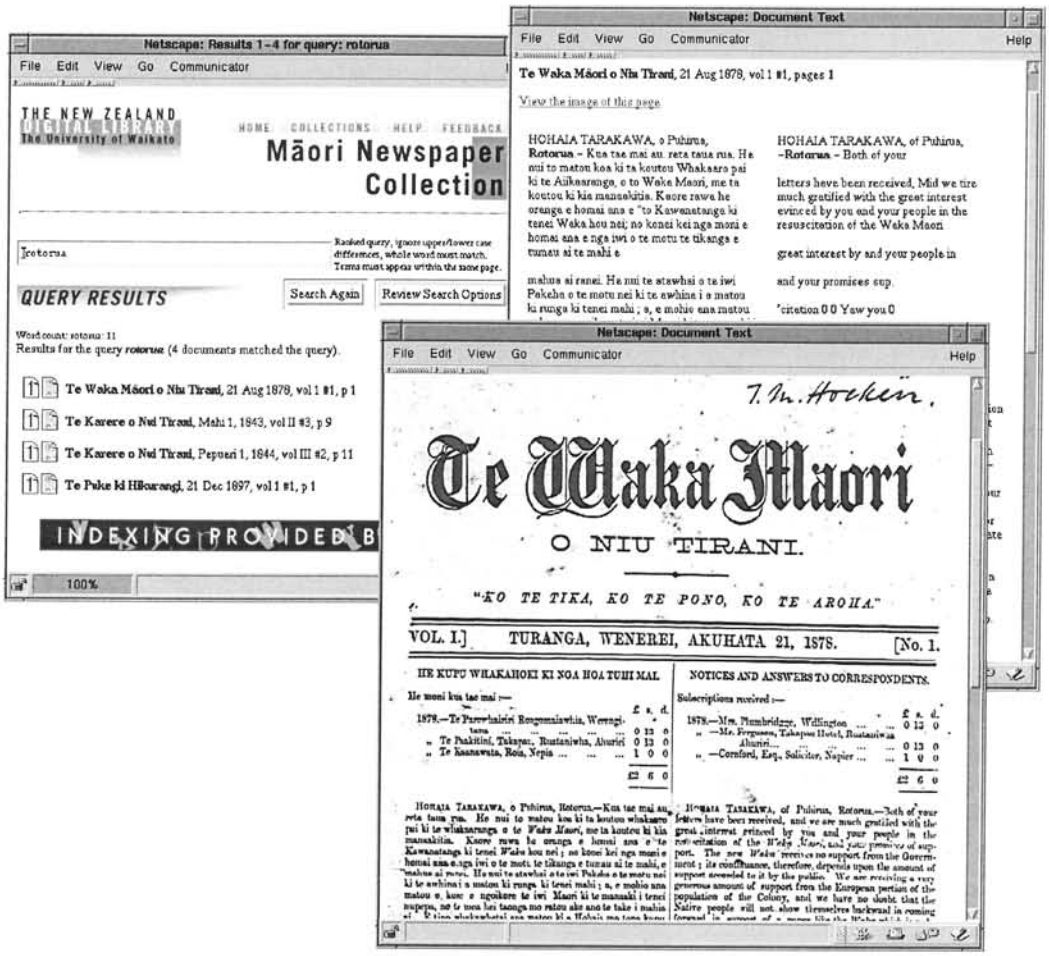

Figure 3 Searching the *Historic New Zealand newspapers* collection

Figure 4 Making a query to the *Arabic demonstration* collection

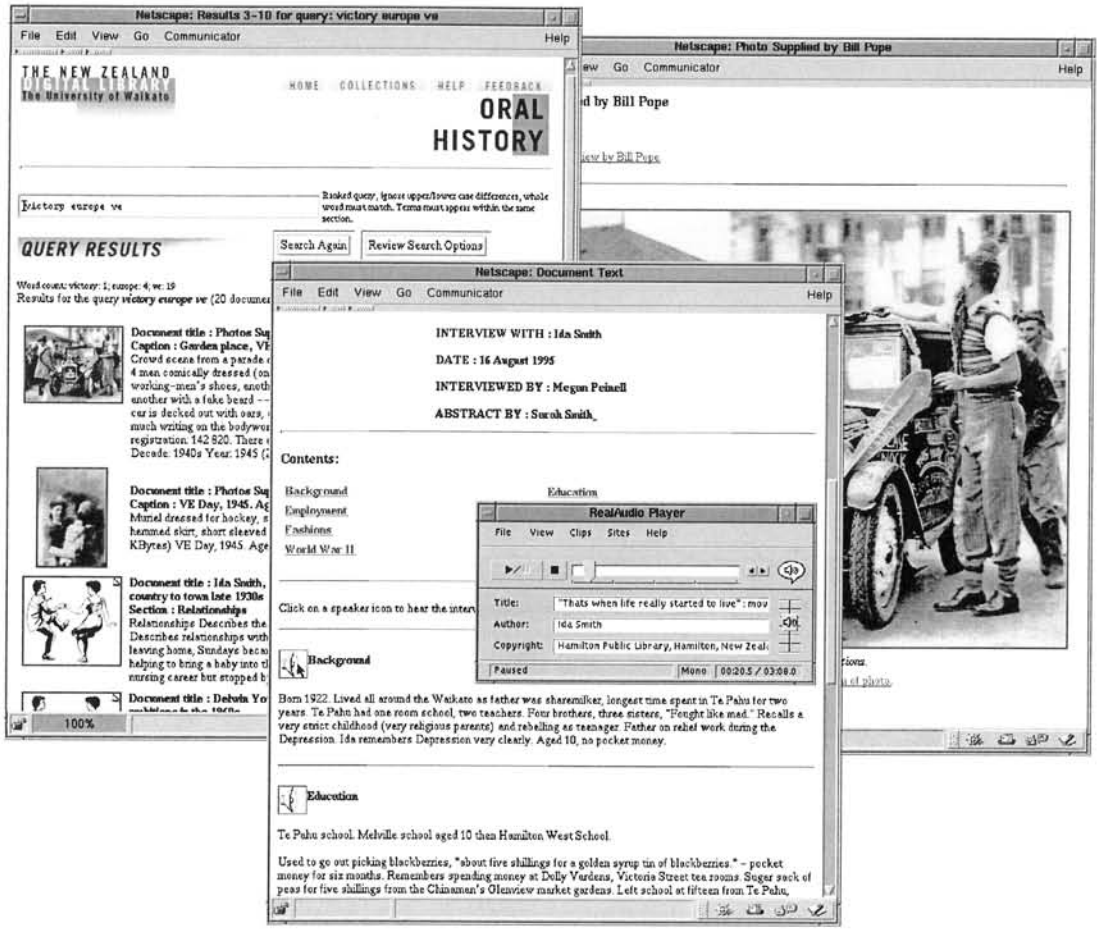

Figure 5 Learning about *VE day* from the *Oral history* collection

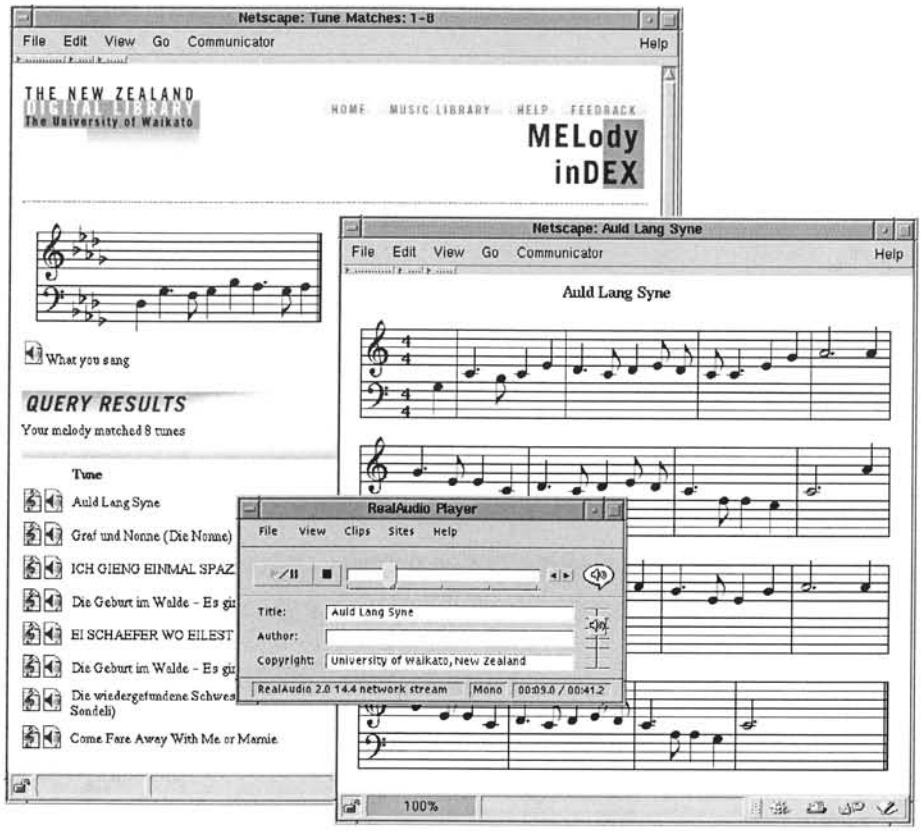

Figure 6 Using the *Melody index* collection

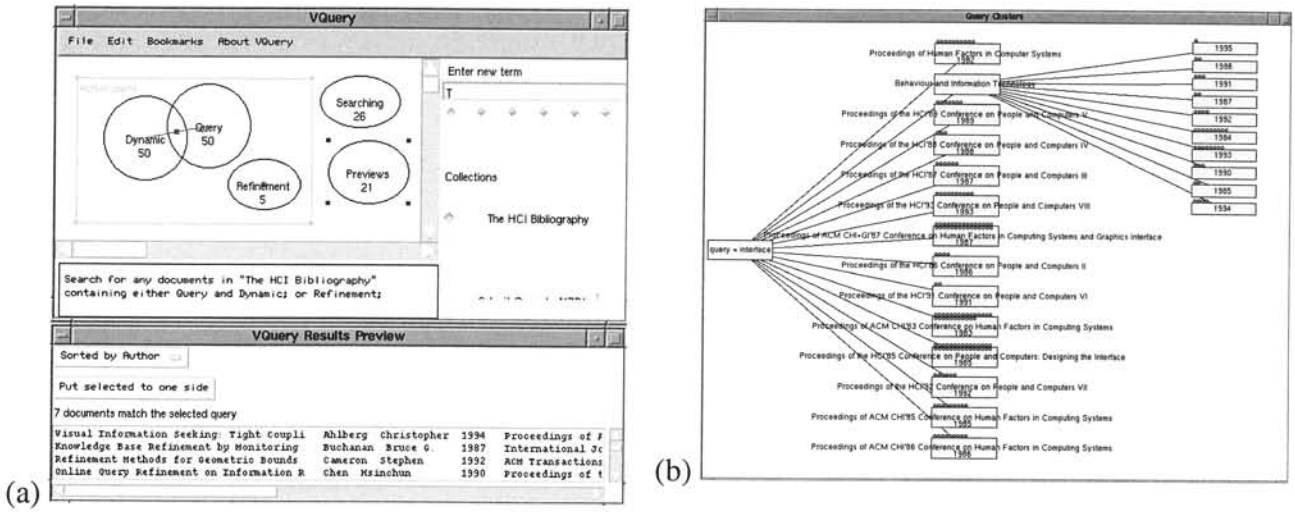

Figure 7 Experimental interfaces to the New Zealand Digital Library
(a) Interactive query specification
(b) Visualizing result set clusters

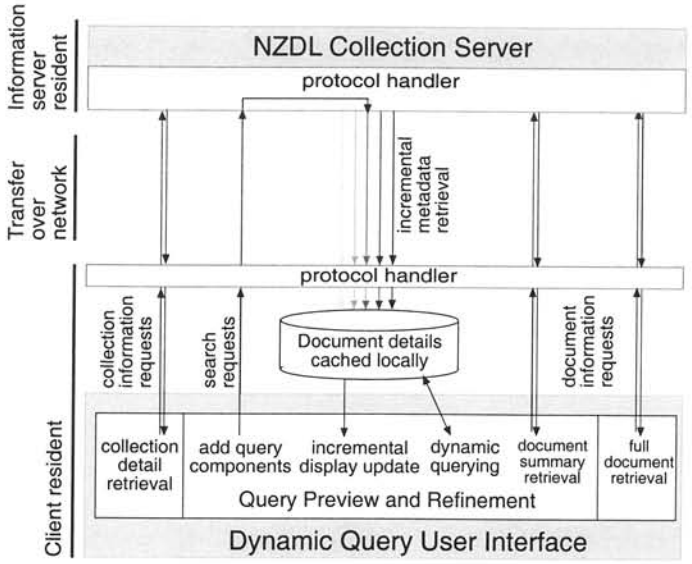

Figure 8 A dynamic query architecture that exploits the communication protocol