Assessing Repository Technology: Where Do We Go From Here?*

David Eichmann†

Software Reuse Repository Lab (SoRReL)
Dept. of Statistics and Computer Science
West Virginia University

Send correspondence to:
David Eichmann
SoRReL
Dept. of Statistics and Computer Science
West Virginia University
Morgantown, WV 26506
email: eichmann@cs.wvu.wvnet.edu

* to appear in the *International Journal of Software Engineering and Knowledge Engineering.*
† This work was supported in part by NASA as part of the Repository Based Software Engineering project, cooperative agreement NCC-9-16, project no. RICIS SE.43, subcontract no. 089 and in part by a grant from MountainNet Inc.

Abstract

Three sample information retrieval systems, archie, autoLib, and WAIS, are compared as to their expressiveness and usefulness, first in the general context of information retrieval, and then as prospective software reuse repositories. While the representational capabilities of these systems are limited, they provide a useful foundation for future repository efforts, particularly from the perspective of repository distribution and coherent user interface design.

1 – Introduction

As information becomes an increasingly important sector of the global economy, the way in which we access that information – and thereby the way in which we access and structure *knowledge* – becomes a critical concern. The engineering of knowledge is quickly becoming an area of research in its own right, independent of its parent disciplines of artificial intelligence, database systems, and information retrieval; consider the title of the journal that you now hold in your hands. Wegner recognized the value of knowledge engineering in his landmark article on the role of capital in software development:

> "Knowledge engineering is a body of techniques for managing the complexity of knowledge... it is capital-intensive in the sense that reusability is a primary consideration in the development of books, expert systems, and other structures for the management and use of knowledge." [10, p. 33]

Just as Wegner observed that the products of software engineering are capital, so are the products of knowledge engineering a form of capital. Identification, structure, and locatability are critical to the enabling of this knowledge capital. Innovation in this area is driven from two diverse perspectives, the traditional perspective of researchers and a not-so-traditional perspective of what might be referred to as an information underground.

The goal of this information underground is not necessarily an extension of the state of the art, but a rather more pragmatic development of an informational infrastructure [4]. The prototypes resulting from this type of work propagate quickly over the Internet, immediately generating large numbers of users. Even while still experimental, systems that provide distinct benefit frequently need to limit access in order to maintain reasonable system performance for other users of the underlying platforms.

My reference to this community as an underground is calculated, for even within the computer science community (let alone the academic or commercial communities as a whole), only a small percentage of individuals are aware of such information systems. This article was spurred by my interest in software repositories, a number of conversations that I've had in recent months, and the

benefit I think can be gained by widening the forum for such systems to a larger audience.

In particular, it is interesting to evaluate these systems as an enabling technology for software reuse repositories. Repositories, and by implication, information retrieval mechanisms, play a critical role in successful reuse. This statement disagrees with the conventional wisdom [9], that reuse is a social and managerial issue, and not a technical one. A closer examination of the conventional wisdom leads to a recognition that without a repository with substantial representational capability many of the social and managerial requirements cannot be supported.

This paper surveys a number of interesting information server projects, with an eye towards enabling technologies. Section 2 lays down a typical scenario in which such systems are used. Sample sessions for three systems appear in section 3, and an analysis appears in section 4. I conclude with remarks on the potential of future systems.

2 – A Scenario and User Profile

Consider a programmer involved in a research project in some reasonably sized university. I choose this context not only for its personal familiarity, but also because

- such projects typically take place in facilities with rich local and wide area network connectivity;
- programmers typically have a personal workstation with substantial display capabilities (e.g., X-Windows); and
- there are strong incentives in avoiding the redevelopment of capabilities available from other projects, either local or remote.

In effect, the development environment is one which is typical, or will be within the next few years. In addition, the social infrastructure and equipment infrastructure for a successful reuse program are present, if not an explicit charter for reuse, or a true repository.

Our programmer is now faced with a dilemma — aware that there is a strong likelihood that a

needed tool or component already exists somewhere out on the network, but uncertain as to where to begin the search in the thousands of systems that currently make up the Internet, or even how to identify the needed artifact. Until recently the only choices included asking acquaintances for advice (although the study by Schwartz and Wood [7] demonstrated the amazing potential for even ad hoc mechanisms such as this), poring over intermittently posted electronic digest news articles for likely sounding names, or manually searching a few sites maintained by volunteers and accessible through anonymous ftp. Obviously, our programmer is ripe for recruitment as a client of the services provided by the information underground.

3 – Example Repositories

Early in the evolution of the Internet, system administrators began adapting file transfer facilities into what today is referred to as anonymous ftp, comprised of publicly accessible accounts, a limited file space, and a restricted command set. These facilities, while amazingly popular as a dissemination tool, presume a fair amount of user knowledge, not the least of which being where to look for the sought-after artifact. This section describes three information systems, archie, WAIS, and autoLib. Each of these systems has a distinct design focus, anonymous ftp access in archie, document retrieval/display in WAIS, and a limited form of electronic library in autoLib. However, the resulting systems have much in common, and their look and feel has several similarities. These systems were selected for discussion because they were designed primarily as *information retrieval systems*, rather than as software repository systems.

3.1 – archie

The archie system is "an on-line resource directory service for an internetworked environment" [3]. While archie isn't truly a repository per se, since it doesn't actually contain the artifacts that it classifies, when treated as a whole with the diverse anonymous ftp sites that it references, it does fit into our discussion. Archie grew out of the efforts of Emtage and Deutsch to automate the creation and referencing of previously hand-maintained lists of anonymous ftp sites. A demon peri-

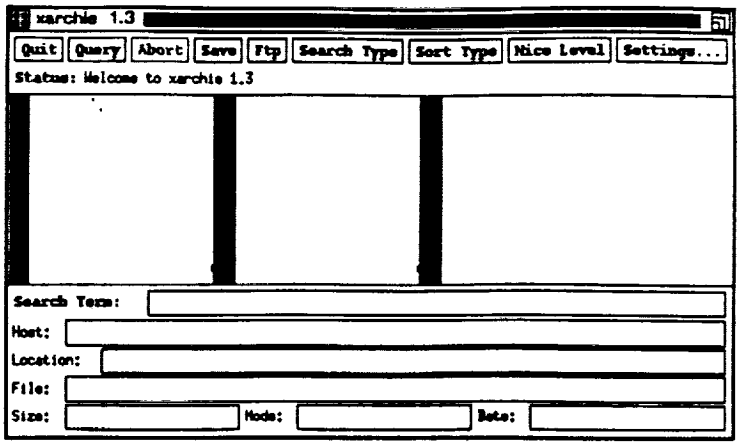

Figure 1: archie screen upon entry

odically sweeps through a list of known ftp sites, creating a list of artifacts accessible at each of them. This list of artifacts is then indexed for access by clients throughout the Internet seeking a site for some particular item.

I describe the xarchie user interface here, developed by Ferguson for the X-windows system from the ASCII user interface developed by Kehoe and the Prospero system developed by Neuman [5]. Xarchie and archie together form an example of a client/server application architecture, where the client application (xarchie) provides user-local support for commands, information display, and communication to the server application (archie), which provides access to a remove facility, in this case the archie database. Figure 1 shows xarchie's screen at entry. The series of buttons across the top of the window control the activity of the user's xarchie client and its interaction with an archie server and the ftp sites which the server indexes. Figure 2 shows the xarchie settings panel, including in particular the mode of search (exact, substring, regular expression, etc.), the order that hits are presented (sorted by name, modification date, etc.), and the archie server host to interrogate, in this case archie.sura.net.

Entering a search term for an artifact, say *xarchie.tar.Z*, a compressed Unix tar file of the xarchie source directory, and clicking the query button initiates the search, as shown in figure 3. As the search progresses, xarchie updates the status line, indicating establishment of connection, progress, and completion.

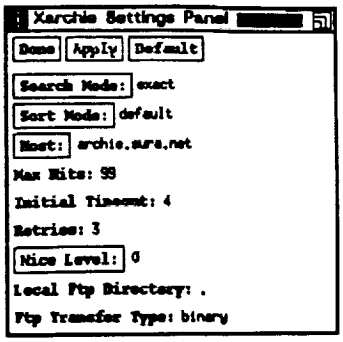

Figure 2: archie settings window

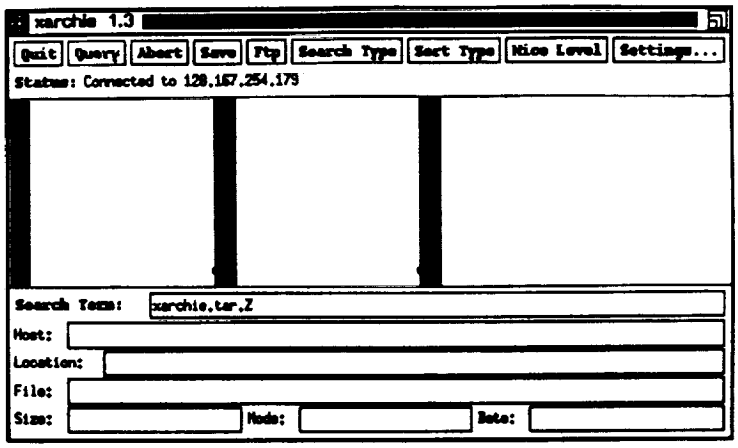

Figure 3: Initiating an archie search

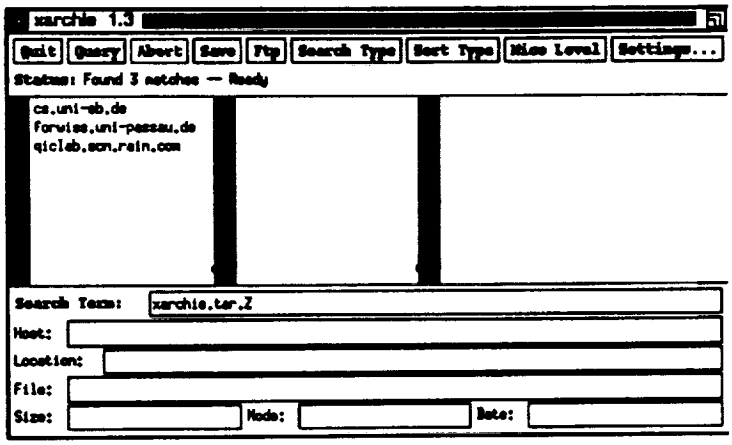

Figure 4: archie search results

Figure 4 shows the results of the search as a list of sites in the left scrolling region in the middle of the window. Selecting a particular site by clicking on it results in figure 5, with the location, size, and so on for this artifact on this site. A single instance of a match at the selected site automatically selects the middle scrolling region (corresponding to the directories) and the right scrolling region

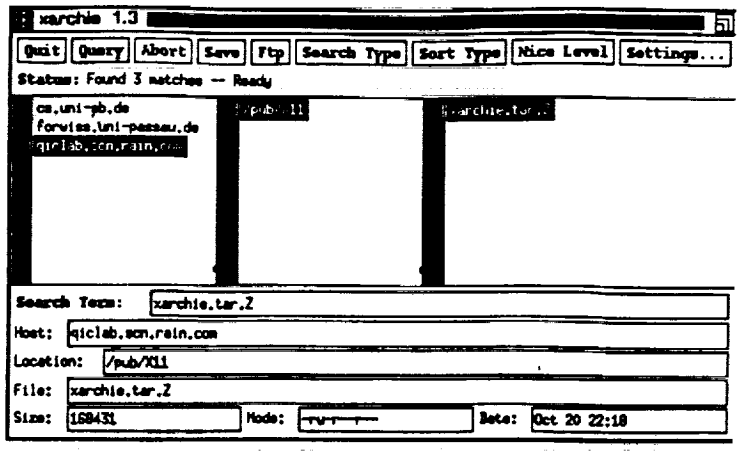

Figure 5: Selection of a site and copy

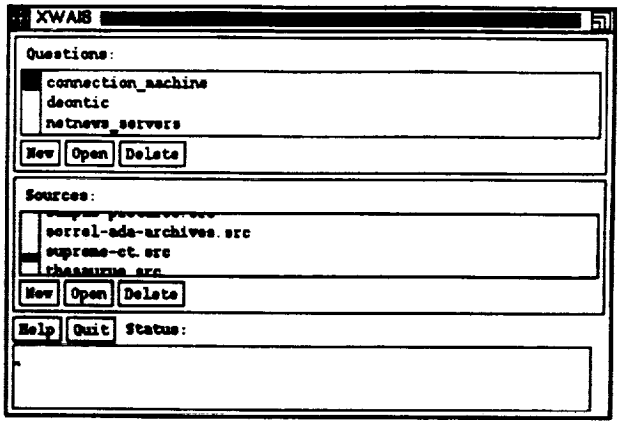

Figure 6: WAIS main window

(corresponding to the files). Multiple matches (typical with inexact matches) require the selection of both a directory and a file for the lower fields to be filled in. Clicking the ftp button establishes an anonymous ftp session to the archive site and retrieves a selected artifact into the local directory shown in the settings panel (shown in figure 2 as '.', the current directory).

3.2 – WAIS

The Wide Area Information Service (WAIS) is an experiment in text-based distributed information systems by Thinking Machines and a number of collaborators [4]. WAIS supports the notion of multiple sources of information; a user selects one or more sources to respond to a question, phrased as a string of words which are deemed relevant to the question. Figure 6 shows the main window, containing a list of previously phrased questions and a list of already known sources.

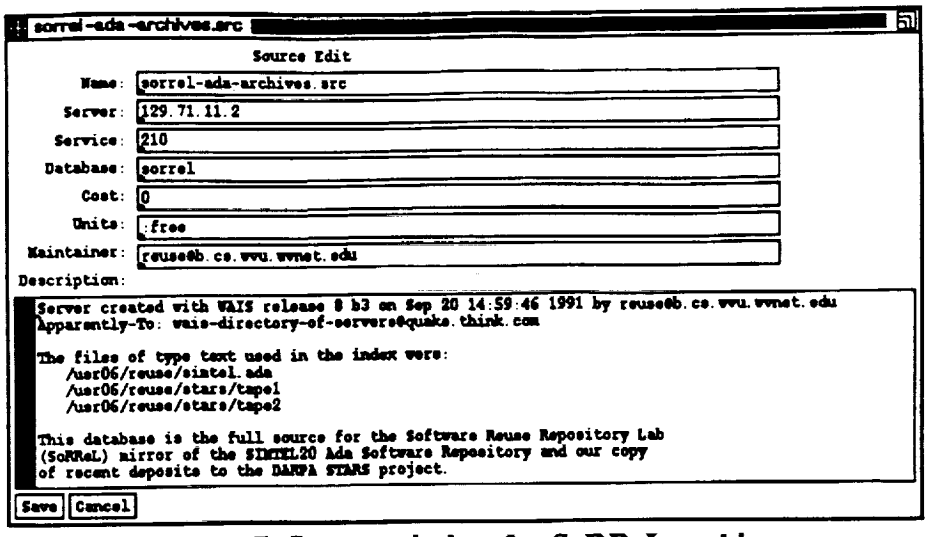

Figure 7: Source window for SoRReL archive

Opening a source displays a window containing information concerning the nature and location of that source, as shown in figure 7 for the Ada archive that the SoRReL group maintains. This information includes the Internet address and service port that the server for the source listens to, as well as unit and cost fields (as yet unused) and a textual description of the source. A single server can support multiple sources, each separately indexed and independently accessible. A distinguished source, maintained by Thinking Machines, acts as a directory to other sources by indexing source definitions such as the one shown in figure 7. These source definitions are retrievable using the same question mechanism employed for other questions. The sole distinction is in the saving of results; saving a source definition places it in the directory containing the user's known sources, making it accessible for subsequent questioning.

Figure 8 shows the question window following a successful search of the SoRReL source. Users select one or more already known sources to be consulted for this question by clicking the add source button and selecting from the resulting display of sources. The "Tell me about:" field accepts a collection of words to be used as a specification of the question. WAIS uses relevance feedback as its search mechanism; documents which match one or more of the words contained in the "Tell me about:" field are added to the collection of matching documents, and then presented to the user in the "Resulting Documents:" field ranked by a relevance metric, an indication of the fit to

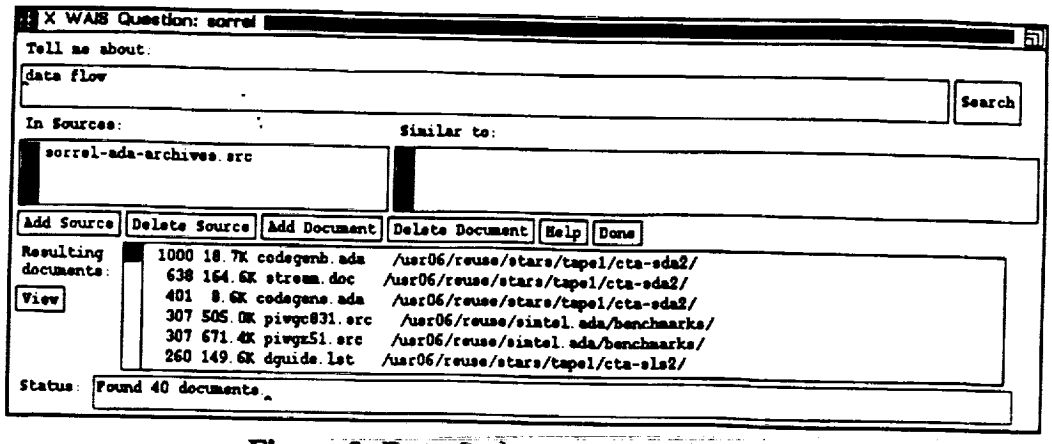

Figure 8: Data flow question and results

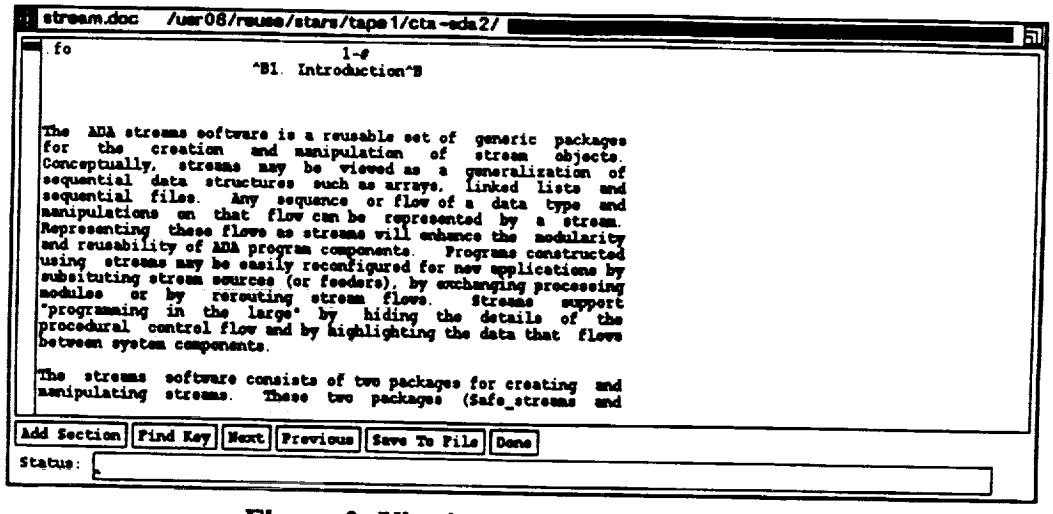

Figure 9: Viewing the streams document

the words occurring in the query string. Relevance feedback has been shown to be more effective than boolean expression as a search mechanism for textual information (a report of one such study appears in [6]).

Selecting a result document for viewing retrieves the document from its server and displays it in a window such as that shown in figure 9, which contains a portion of a document describing an Ada implementation of a stream package. The find key button scrolls the window and highlights in turn each occurrence of search words in the document. WAIS lets users specify an arbitrary program on the user's machine as the viewer to be invoked for a given class of documents, with the class defined using the suffix of the document's file name (for example, xgif is typically used to display images whose names end in '.gif').

Iterative refinement of a search that results in documents viewed with the text viewer is accomplished by selecting a salient portion of the document and clicking the add section button. An indication of the text selected is added to the "Similar to" field in the question window. Subsequent searches then append these refinements to the primary search phrase.

3.3 – autoLib

The autoLib system, under development by Barrios Technology and NASA's Johnson Space Center, is a monolithic application supported by a commercial relational database system (comprising the meta-information) and a UNIX file system (comprising the objects themselves). The structure of information provided by archie and WAIS is flat in the sense that there is little structure provided other that an indexing mechanism. The autoLib system, on the other hand, supports both a flexible single inheritance mechanism for definition of meta-information, and the definition of heterogenous collections of objects drawn from the inheritance scheme [1]. Figure 10 shows the main window for autoLib, including the topmost collection and its immediate sub-collections.

Clicking on an entry in the list moves the user down the hierarchy of collections to the corresponding subcollection, and that collection's subcollections are then displayed. The three buttons at the bottom of the window allow the user to step back up one level in the collection hierarchy, to move directly to the top of the hierarchy, or to view the objects associated with the current collection, respectively.

Figure 11 shows the object browser window, displaying the contents of one such collection. The three columns of information include the object's identifier, its filename, and a .short title.

The object viewer window for object 2446 appears in figure 12. AutoLib employs a commercial relational database package for information storage, but the user model for autoLib is object-oriented, defined not only as a hierarchy of class definitions, where superclass names are prefixes of subclass names, but also as a hierarchy of collections, as mentioned earlier. AutoLib maps each

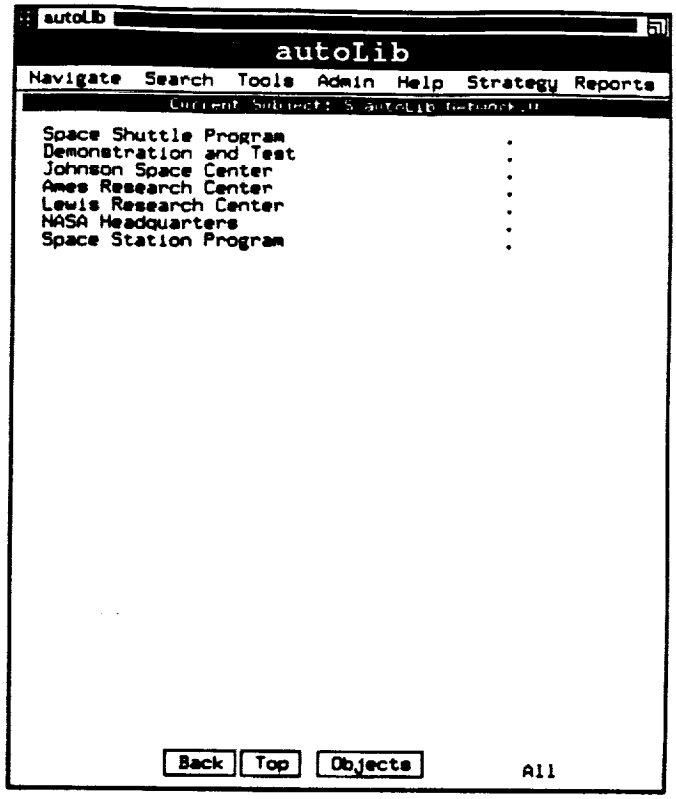

Figure 10: autoLib main window

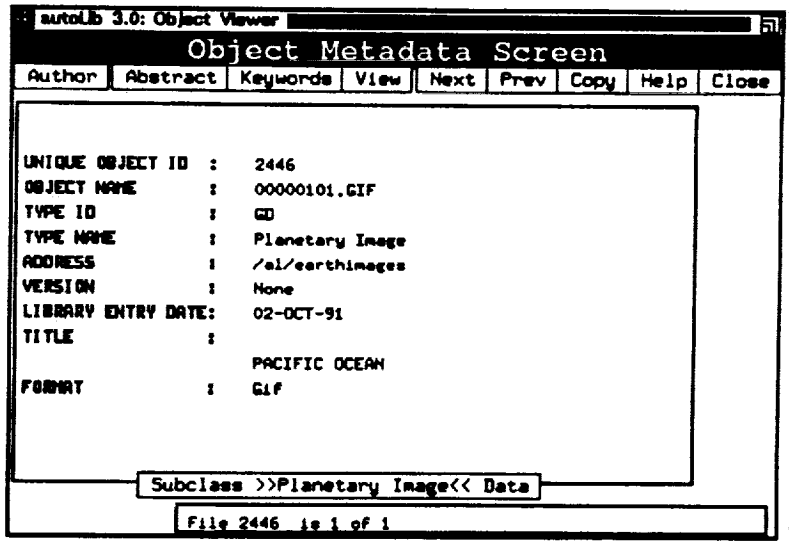

Figure 12: autoLib object viewer

concept (collection, class, object, etc.) into a corresponding database relation and maps each field in an autoLib window (e.g., the object filename, 00000101.GIF, for object 2446) to an attribute in the corresponding relation. The system derives the interpretation for a given object in the generic object relation from the field definitions stored by autoLib in the class field relation. While this is

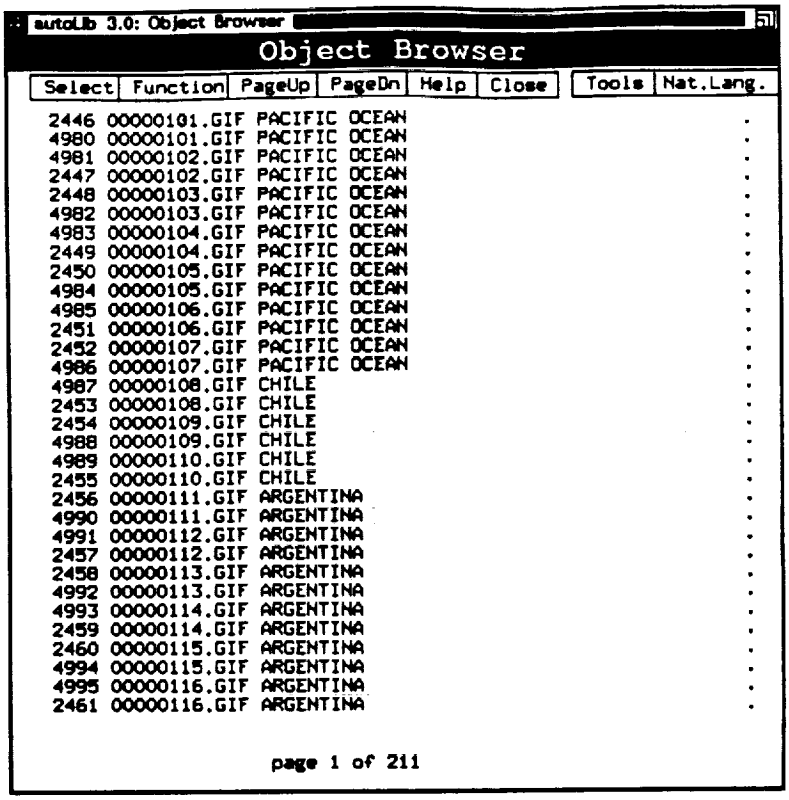

Figure 11: autoLib object browser

not a true object-oriented database, it provides much of the flexibility and rich structural mechanisms of a object-oriented database. The integration of objects and relations has been carried much further in work on extensible database systems such as POSTGRES [8].

In addition to the collection browsing mechanism described here, autoLib supports traditional boolean expression retrieval and a form of relevance feedback. Each object class has associated with it a *tool*, which is used to view the object itself, as opposed to the metadata characterizing that object, i.e., the fields presented in the object view window. Unlike WAIS, where tool execution occurred on the user workstation, tool execution in autoLib occurs on the autoLib server – the user workstation merely acts as an X-windows display.

4 – A Brief Comparison

Viewing these three systems as potential software repositories is interesting, and at the same time somewhat unfair to their designers, as none were created with that purpose in mind. However,

systems such as these are frequently called into service in such contexts, and the flexibility and adaptability exhibited provides interesting concepts and features for inclusion into systems specifically intended as repositories. Table 1 summarizes major aspects of the three systems. The popu-

Table 1: Systems Synopsis

	archie	WAIS	autoLib
architecture	client/server	client/server	monolithic
# server sites	~10	~100	1
interfaces	X-Windows, ASCII	X-Windows, ASCII	X-Windows (ASCII under development)
retrieval mechanisms	pattern-matching (on name only)	relevance feedback	browsing, boolean expression, relevance feedback (on abstract only, not full text)
information domain	material available by anonymous ftp	textual information	NASA flight center library materials
information stored	name, location, file attributes	word occurrence, headline, full text	full text / image, index terms, meta-information (administrator-defined)
archiving responsibility	decentralized	decentralized	centralized
indexing responsibility	centralized	decentralized	centralized
support required (archive)	none	moderate	high
support required (indexing)	moderate	low	high
promise as a repository	poor	limited	a potential framework
availability	public	public	private

larity of archie stems not from its rich representation scheme or novel search mechanisms, but rather from the low levels of effort required on the part of archive administrators and users to employ the system. It is an excellent example of how a limited purpose system implemented by volunteers can provide a valuable resource. Referring to archie as a software repository, however, stretches the definition of repository perhaps a little too far. Consideration of an artifact at a site as a candidate component requires that the user knows both the name and the purpose of that artifact, and the retrieval of the complete artifact (irrespective of the total size) before further consideration can be made.

The display facilities of WAIS alleviate the limitations of archie by presenting the user with a flexible means of query specification (without requiring classification by the archivist) and the opportunity to select from a variety of candidates and view portions of them prior to retrieving the complete text of the final selection. WAIS further increases flexibility in the nature of repositories by supporting interrogation of multiple sources for a given query and the generation of both public and private sources. (Note, however, that there is no technical impediment to doing this with archie as well – the archie designers simply chose global indexing rather than regional or local indexing.) The principle virtue of WAIS, its treatment of all material as text to be indexed, is also its principle failing from our perspective – there is no discrimination between code, supporting documents, and so forth – resulting in slightly more cumbersome search behavior.

The use of an administrator-defined set of collection and class definitions provides autoLib a great deal of flexibility in organizing the information. In addition to the ability to organize the global structure of the information base, this definitional facility supports meta-descriptions of artifacts, a useful feature in our chosen context.

The structuring, classification, and retrieval mechanisms of autoLib are by far the richest of the three systems compared here. Much of this power obviously stems from the fact that autoLib is a proprietary system, whereas archie is a volunteer effort and WAIS is a research project. However, autoLib's look and feel suffers dramatically in our sample context. Unlike archie and WAIS, which use a client/server paradigm, autoLib executes solely on the server platform. In wide-area domains like the one in which our programmer operates, this results in slow display and update of windows, and an inability for a user to select alternative viewing tools without the intervention of the repository administrator.

5 – Conclusions

This paper reviewed three example information retrieval systems currently in use by a broad diversity of users. I focussed on computer-supported repositories for software artifacts (i.e., com-

ponents, documents, test suites, executable images, etc.) rather than addressing the more broadly-scoped notion of an information repository, which could easily encompass entities such as public libraries.

While these systems were not explicitly designed as software repositories, they do each provide some aspect of repository requirements. Each is a legitimate step forward in utility from early techniques for wide distribution of software. This analysis leads to the following proposal for perceiving the current state of software repository efforts from the standpoint of information systems.

Generation 1 – Program Libraries

This includes not only traditional compiler libraries, but also more distributed mechanisms such as the Ada Software Repository [2] and the various archives for news groups such as comp.sources.unix.

Generation 2 – Information Servers

Examples of this generation include archie, autoLib, and WAIS. The emphasis here is on the indexing and retrieval mechanisms, rather than upon deep representation.

Generation 3 – Component Bases

Fine-grain characterization of components and interrelationships distinguishes this generation. The nature of reuse in this generation is compositional, and is typified by the Department of Defense STARs efforts and the Japanese Software Factory projects.

Generation 4 – Software Knowledge Bases

This generation provides deep knowledge about representation, generation, and composition of components and design schemes and the process of software development.

My separation criteria for repository generations involves the nature and accessibility of the knowledge of each artifact that comprises the repository. Generations one and two provide wide access to artifacts, but little supporting infrastructure (although it might be argued that autoLib

could through the proper configuration efforts of a repository administrator be turned into a rudimentary generation 3 system). Generations three and four provide increasingly rich information concerning the nature of the artifacts contained within them. However, with this richness comes increasing specialization of domain, and increasing difficulty in supporting interoperability between repositories. The component base services of today and the software knowledge base services of tomorrow should not loose sight of the design goals of today's successful information servers.

References

1. Barrios Technology, *autoLib Automated Online Library Version 3 Product Overview*, March 1990.

2. R. Conn, "The Ada Software Repository and Software Reusability," *Proceedings of the Fifth Annual Joint Conference on Ada Technology and Washington Ada Symposium*, 1987, 45-53. (Also appears in *Tutorial: Software Reuse: Emerging Technology*, W. Tracz (ed.), IEEE Press, 1988, 238-246.)

3. A. Emtage and P. Deutsch, "archie – An Electronic Directory Service for the Internet," *Proceedings of USENIX*, San Francisco, CA, January 1992, 93-110.

4. B. Kahle, *Wide Area Information Server Concepts*, Thinking Machines Inc., November 1989.

5. C. Neuman, *The Virtual System Model for Large Distributed Operating Systems*, The University of Washington, 1989.

6. S. E. Robertson and K. Sparck Jones, "Relevance Weighting of Search Terms," *Journal of the American Society for Information Science*, **27** (1976), 129-146.

7. M. F. Schwartz and D. C. M. Wood, "A Measurement Study of Organizational Properties in the Global Electronic Mail Community," Technical Report CU-CS-482-90, University of Colorado, Boulder, August 1990.

8. M. Stonebraker, L. A. Rowe, and M. Hirohama, "The Implementation of POSTGRES," *IEEE Transactions on Knowledge and Data Engineering*, **2**, 1 (1990), 125-142.

9. W. Tracz, "Software Reuse Myths," *ACM SIGSOFT Software Engineering Notes* **13**, 1(1988) 17–21.

10. P. Wegner, "Capital–Intensive Software Technology," *IEEE Software* **1**, 3 (1984) 7–45.