Melding Structured Abstracts and the World Wide Web for Retrieval of Reusable Components

Jeffrey S. Poulin and Keith J. Werkman
Loral Federal Systems
Owego, NY
poulinj,keithw@lfs.loral.com

Abstract

Reusable Software Libraries (RSLs) often suffer from poor interfaces, too many formal standards, high levels of training required for their use, and most of all, a high cost to build and maintain. Hence, RSLs have largely failed to return the reuse benefits promised by their developers. This paper first describes an RSL implementation using the World Wide Web (WWW) browser Mosaic and shows how it meets most RSL needs, avoids most RSL pitfalls, and costs only a fraction of the cost for the average commercial RSL. Second, the paper describes a way to quickly assess the important aspects of a piece of software so programmers can decide whether or not to reuse it. Using the observation that when programmers discuss software they tend to convey the same key information in a somewhat predictable order, this paper describes a method to automatically mimic this activity using a Structured Abstract of reusable components. Structured Abstracts provide a natural, easy to use way for developers to (1) search for components, (2) quickly assess the component for use, and (3) submit components to the RSL.

1.0 Overview

The quest for ways to improve the software development process has led many organizations to pursue the substantial benefits available through software reuse. Many organizations focus their reuse initiatives on a reuse library where members of the organization can both store reusable assets and retrieve assets when they need them. Traditional RSLs use specialized methods for component classification, search, and retrieval. Unfortunately, these formal tools and techniques require both a large investment to implement and substantial training to use. For these reasons, many organizations have seen little benefit from their RSLs even though they may contain a large number of quality assets. This paper describes two initiatives aimed at remedying this situation.

First, we describe a software reuse library interface and search ability using the National Center for Super-Computer Research (NCSA) WWW browser Mosaic [2]. We developed this interface for the Loral Federal Systems RSL, which we call the *Federal Reuse Repository (FRR)*. Mosaic provides an simple, easy-to-use method to search for and extract reusable assets from the FRR. With large organizations investing as much as 80 to 130 person-years to develop a formal RSL, our Mosaic interface cost less than 1% of the cost to develop and maintain a standard, commercial-quality RSL [1], [12], [15]. Despite the minimal investment, our Mosaic-based RSL has quickly gained favor due to its intuitive interface and powerful features.

Second, we have developed a technique that provides the most needed reuse information quickly and naturally to the user. The method, called the *Structured Abstract (SA)*, works by always giving the same, important information in the same order. Specifying the information to give guarantees the user gets the information the user needs. Specifying the order allows the user's eye to train itself on the spot in the abstract containing the critical, decisive data. Finally, presenting the information in an natural-language abstract gives the information to the user in a familiar way, mimicking the manner the user would receive the information from a colleague over the phone or in a conversation.

SSR '95, Seattle, WA,USA

Our work with Structured Abstracts investigates how to best use the existing classification information residing in government and industrial reuse libraries. This allows us to leverage the large investment made in the original classification of components in these RSLs. Because much of the information in an SA already exists as part of a detailed classification, we can automatically create a fast on-line or hardcopy index of an RSL's contents for use in any environment.

By using a standard SA template, we have an easy-to-use method with which developers can search for needed components. The standard template helps users understand retrieved components so they can quickly assess them for use. The SA also provides a standard template for describing components that a developer wants to put *into* the RSL. Finally, anyone familiar with software and a pointing device can quickly grasp Structured Abstracts, thereby practically eliminating the need for any special reuse knowledge or training.

2.0 An RSL using the WWW

We implemented the LFS FRR using the WWW and an Mosaic interface to take advantage of the explosive growth of the WWW within the company [11]. The FRR contains several types of reusable components, including documents and software in a number of programming languages. The easy-to-use interface works with any WWW browser running on virtually any platform, including AIX,tm OS/2,tm VM,tm UNIX,tm DOS Windows,tm and Mac,tm and provides high-performance searching by subject, component source, and keyword search. Accessible from virtually any platform at any location in the company, the WWW implementation replaces the former mainframe-based RSL described in [12].

2.1 About the FRR

The FRR contains abstract data types, system utilities, Application Programming Interfaces (APIs), and other general-purpose functions that have wide application across LFS. Although the FRR contains some domain-specific collections, we consider the FRR a domain-independent RSL. Every software asset in the FRR comes with a complete set of supporting information to help users understand and integrate the software into their product. For example, if a user selects the Adatm implementation of the "POSIX_Signals" package from the *General Purpose Ada* library, the RSL returns an abstract, interface information, usage instructions, etc., in addition to the Ada body and Ada specification.

Loral FS Reuse – FRR

You've reached the home page for the Loral Federal Systems Reuse Repository (FRR). The FRR contains several hundred reusable software and document components; read more about the FRR, and see what's coming soon! Find out about external reuse sources.


FRR, hierarchical view (arranged by Language/Library)


FRR, arranged by Subject


Search the FRR (via WAIS)

Click here to contact the FRR Maintainer.

Allen Matheson owns this page.
Last updated: 22 Feb 95

Figure 1. The FRR Home Page

2.2 Searching the FRR

The FRR currently provides three ways to locate a needed component. As shown in Figure 1, a user can search the FRR using a:

1. Hierarchical view, arranged by Language or Library,
2. Subject listing, and
3. Keyword search.

The hierarchical view firsts narrows the search based on implementation language. If the user selects *Ada*, the Mosaic page shown in Figure 2 appears. This page lists the various collections of Ada software in the FRR; users will find it the fastest way to locate a particular component if they already know which collection it comes from; for example, that they want a component from the *Booch-Ada* collection.

However, a user often does not know (or care) where a component comes from and would rather

Ada Reuse Collections

Select the library you need:

- Advanced Automation System (AAS)
- Ada Run Time Environment (ARTE)
- Booch – Ada
- Booch – Ada Enhanced
- Common Ada Missile Packages (CAMP)
- Circuit Card Assembly and Processing System (CCAPS)
- General Purpose Ada
- Karlsruhe Abstract Data Types
- Realtime And Distributed Ada Services (RADAS)
- Sustaining Base Information Systems (SBIS) APIs NEW

Allen Matheson owns this page.
Last updated: 21 Feb 95

Figure 2. The Ada Language Home Page

simply search for components that perform a particular function. The FRR supports this need with the second search option: *The FRR, arranged by Subject.* Figure 3 shows a portion of the page produced if the user selects this option. For example, if the user needs a *monitor* or a *semaphore*, the user would click on *Synchronization Components* on this component listing.

FRR Component Listing by Subject

Select the type of component you need:

- Bit/String Manipulation Components
- Command Line Components
- Communication Components
- Data Structures Components (long list!)
- File Services Components
- Graphics Components
- Input/Output Components
- Miscellaneous Utilities
- Numerics and Math Packages
- OS (POSIX) Interfaces
- Real–Time Components
- Sorting and Searching Routines
- Synchronization Components

Figure 3. The Subject Listing Home Page

Finally, the third search option consists of a simple keyword search. The FRR keyword search, implemented using publicly available Wide Area Information Services (WAIS) software, supports Boolean queries and Porter Stemming (partial matches). The search runs very fast and rank-orders the search results to help indicate which components most closely meet a developer's needs. For example, the query "avl and tree" returned the following in less than 2 seconds:

AVL Tree (Ada)
  Score: 84, Total Bytes: 53323
Avl Tree Row Representation (Ada)
  Score: 62, Total Bytes: 35538
AVL Tree (Ada)
  Score: 60, Total Bytes: 29352
AVL Tree Repr Acc (Ada)
  Score: 53, Total Bytes: 40072

3.0 The Classification Problem

Large Reusable Software Libraries (RSLs) often use a formal, extensive classification mechanism to describe their contents. This mechanism provides detailed information upon which a user can search for components. The classification can greatly improve search precision, especially in large organizations or in situations where numerous RSLs interoperate by sharing access to components across a network or the Internet [5].

However, the up-front presentation of this information causes a number of problems because of the difficulty of quickly extracting the relevant reuse information from the plethora of other data [10]. To assist in reuse, the RSL must present the right level of information to the reuser. Unfortunately, using an extensive classification scheme requires the user to have a working knowledge of the issues and techniques surrounding software classification [13]. The reuser must especially have an understanding of the scheme used by the user's particular RSL [17]. Attempts to help casual or infrequent reusers locate components similar to those asked for, such as conceptual closeness graphs or synonym lists, aid in retrievability but do nothing for understandability and rapid reuse assessment. Mapping the user's idea of what the user needs to an existing component must happen quickly, efficiently, and painlessly [9].

Experience shows that reading a long list of component classifiers, many of which may not have immediate relevance, does not give an intuitive feel for the applicability of a particular module to a specific situation. In fact, most programmers typically assess the

applicability of a potentially reusable part in a matter of seconds, much as potential buyers of a house get their most lasting impression from it's "curb appeal." A reuser needs to make the most of those first precious seconds and make the best possible reuse decision.

4.0 Understanding Programs

Several program understanding techniques seek to help users deal with the flood of information provided by lists of facet and attribute-value classifiers. Central to these techniques lies the presentation of information in a standard format. For example, Input-Process-Output (IPO) diagrams provide an early method of standardizing component descriptions. A IPO diagram lists all program inputs, outputs, and a description of each process or function provided by the software [16].

Musser and Stepanov [8] adopted a standard set of information to describe the components in their Ada generic list package library. This information includes the interface specification, a short text description of the function, complexity metrics, examples, and dependencies on other routines. However, just as with long lists of classifiers, the method presents the information in a layout which forces the user's eye (and consciousness) to jump around the page to acquire and digest the data the user wants, in the order the user wants it.

Linn and Clancy present a standard module representation which they find useful in teaching programming to beginners [6]. By providing a schematic of a program and its function, they allow the student to picture the activities of the module and learn how to build other modules. From a programming understanding perspective, such "templates" make it easy to comprehend how a module works and how to integrate it into a product. However, the supplier must painstakingly construct the template by hand and although the template follows a standard format, the depth of detail does not make it a suitable "quick assessment" mechanism for a reuser.

Natural language methods seem uniquely absent from current approaches. In fact, a recent study by Maiden and Sutcliff examined the analytic and problem solving strategies used by analysts to understand and reuse program specifications [7]. They found that although results varied by individual, analysts preferred to assimilate and understand modules from a narrative describing the underlying reusable domain rather than from the formal specifications normally provided by a RSL. We believe that any effective rapid assessment method should make use of this preference.

5.0 Structured Abstracts

Structured Abstracts (SAs) help solve the component classification and program understanding problems by presenting software descriptions in a logical, narrative form and always providing the same information in the same order. Because the SA uses a natural text presentation, the reuser finds the SA easy and straightforward to understand. Most importantly, SAs transform the process of evaluating components from one of scanning a randomly-ordered list of *(facet,term), (attribute, value)* pairs to one of reading an orderly, concise, natural-language narrative. Finally, SAs have uses beyond searching for and assessing components; reusers can also use SAs to supply components to the RSL.

5.1 Concept

The method uses a paradigm analogous to the way programmers informally exchange information about software, such as when meeting in small groups or in discussions over the phone. The programmers tend to quickly convey the key aspects of the software in a few short sentences, thereby telling the interested party, in a very concise fashion, just enough information to judge whether or not there exists a component of interest, and if so, whether or not the component merits further investigation.[1]

The following key bits of information about components consistently appear in these conversations:

1. Programming language,
2. Environment,
3. Function performed by the routine,
4. Objects manipulated by the routine,

[1] In which case more detailed programming understanding methods may apply.

5. The platform or operating system upon which it runs, and,
6. Elements which make up the component.

And, if the party shows interest:

7. Where to get more information (a contact).

Further observation shows that the order of the above information tends to remain constant (e.g., language, function, object, platform, and contact). The SA implements these observations by using the following layout:

Items in the SA

1. Computer language and Component type
2. Domain
3. Function
4. Data
5. Operating System
6. (Element, ... , Element)
7. Contact

The basic SA template ties information from the RSL database (shown here underlined and in parenthesis) with default connecting text, shown here in italics. If a software component provides more than one function, an optional second sentence may give that function. The SA template looks like this:

> *A* (Computer Language) (Component Type) *for* (Domain) *that provide* (Function) *on* (Data) *data. Runs on* (Operating System). *Includes* (Element, ... , Element). *Contact* (Contact).

5.2 Sample Structured Abstracts

Extracting classification and descriptive element information into the template may result in Structured Abstracts similar to those below:

> C++ classes that provide text buttons and slide bars for the GUI domain. Runs on OS/2 and AIX. Includes documentation, abstract, and test cases. Contact John Smith (smithj@abc.lfs.loral.com).

> C functions that provide calendar operations on date data for the MIS domain. Calculates days between dates, date format conversions (Julian, Roman), day of month, and day of year. Runs on Zortech, Borland, and Top Speed compilers. Includes license agreement, documentation, and sample code. Contact Joan Brown (brownj@abc.lfs.loral.com).

> Ada packages that provide queues, stacks, sets, maps, and bags for the Abstract Data Type domain. Runs on Alsys, Verdix, and MILSPEC Ada compilers. Includes license agreement, documentation, examples, and test cases. Contact Bill Fried (friedw@abc.lfs.loral.com).

5.3 Implementation

We have implemented the Structured Abstract for component search and component submission. For the initial implementation we combined HyperText Markup Language (HTML) forms and the Common Gateway Interface (CGI) capability of the WWW with Perl scripts to route the selections to a WAIS-indexed database. A couple of points add merit to this approach. First, we continue to require only minimal investment in software and implementation. Second, when we indexed the entire FRR collection using WAIS, we used the formal classification of the original RSL. Therefore, the facets, attributes, and terms used in the HTML forms *exactly* match those used to index the components with WAIS. This leads to a high degree of exact matches, unlike that of most keyword search mechanisms [4].

The current implementation works for software components only. We have not attempted to use SAs for other types of reusable assets, such as documents.

5.3.1 Experiences

As reflected in the following scenarios and figures, we have made several modifications to our SA search and submittal forms based on user feedback. First, sophisticated users indicated a preference for the simple keyword search (the third FRR search option) because they could create very specific Boolean searches that they could not re-create using the SA. To address this, we added an additional "keyword" field so users could use the SA in conjunction with free text search.

Second, most users did not use the "domain" field when constructing queries. This follows from our view of the FRR as a domain-independent RSL.

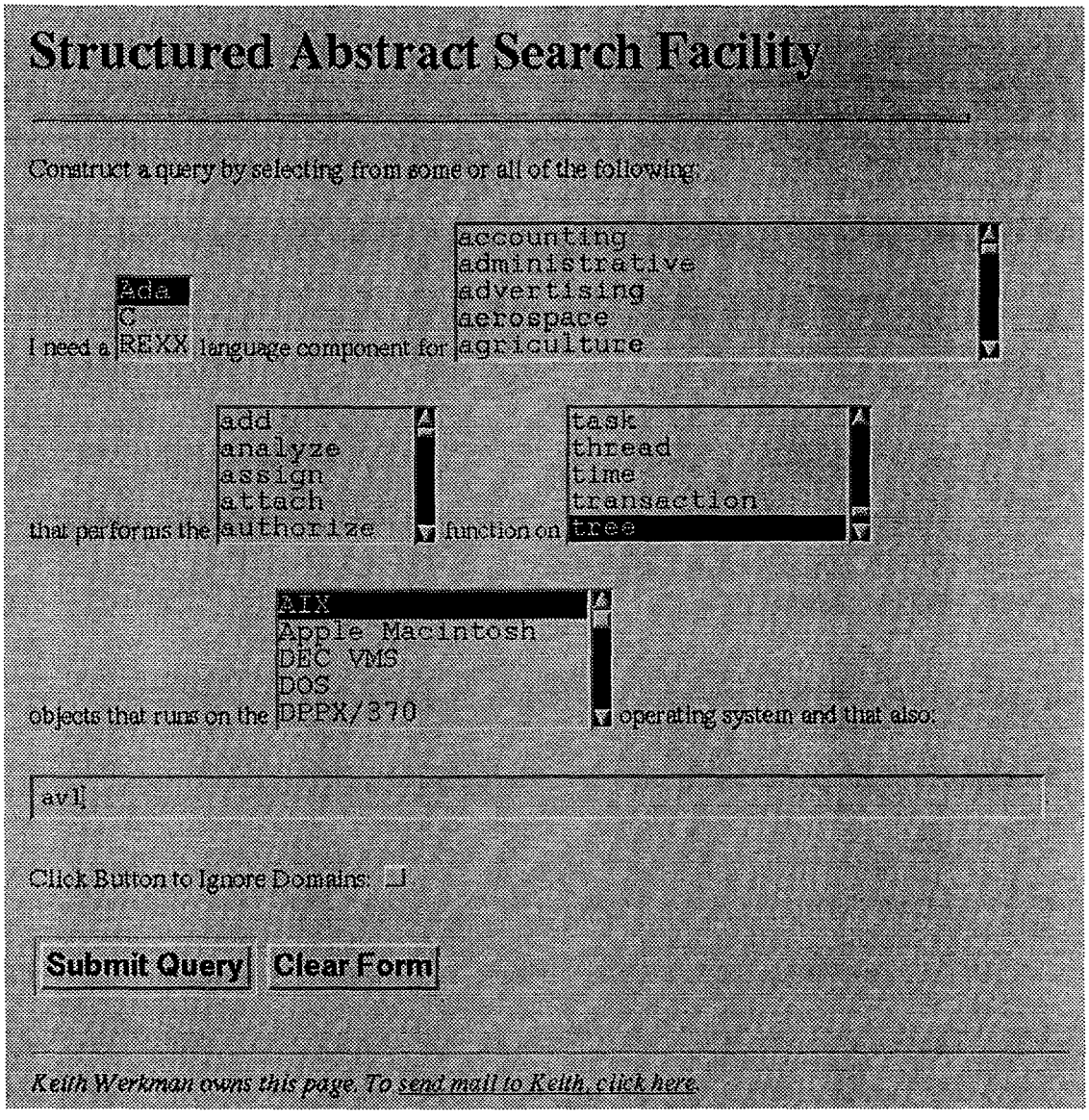

Figure 4. The Structured Abstract Search Facility

We considered deleting *domain* from our implementation of the SA; however, the existence of some domain-specific collections in the FRR and the fact that some users insisted on retaining the field caused us to keep it. The SA currently has a radio button which explicitly makes the SA domain-sensitive.

5.3.2 Query scenario

Figure 4 shows the SA search mechanism. Recall from Section 2.2 that a keyword query of the FRR for "avl and tree" returned four components. Suppose the user still needs software to establish an AVL tree data structure in the Ada language, but the routine must run on an AIX platform. The user does not care about the original domain intended for the component. To issue the query in the SA, the user goes to the WWW page containing the Structured Abstract. The user marks *Ada, tree,* and *AIX* in the appropriate boxes on the SA using the mouse and enters *avl* in the text entry box. When complete, the user clicks on the "Submit Query" icon. In this case, the search returns exactly one component from the FRR:

AVL Tree (Ada)
  Score: 47, Total Bytes: 53323

This component appeared as one of the four AVL trees in the original FRR query, but of the original four only this component has tested, compiled, and run on AIX. The ranking of 47 differs from the original (84) because of the addition of the terms *Ada* and *avl* to the query.

We dynamically generate this search result page in Mosaic, and present this information as HyperText links to the component information. The user would next click on the phrase *AVL Tree (Ada)* to see detailed information such as test cases, usage information, integration instructions, etc.

5.3.3 Submission scenario

Developers that supply reusable components to the RSL use exactly the same SA method as they do for searching; the SA works equally well in both roles. In fact, because the submission abstract looks almost identical to the search abstract we do not include it as a figure. In the submission abstract we changed some of the fixed text between SA fields so it targets a supplier rather than a searcher (i.e., rather than start with "I need a..." the submission SA starts with "I have a..."). When the supplier completes the form and clicks on the *submit* icon, we send the form by electronic mail to the RSL librarian, who manually verifies the information and ensures it meets FRR standards for quality and completeness.

Many RSLs require that component suppliers provide information which the SA does not contain. This information will vary greatly depending on the RSL and does not effect the contents or use of SAs. The RSL can solicite this additional information from the supplier using any convenient technique; as shown in Figure 5, we developed a companion HTML form which acquires the information which we happen need, such as supplier name, etc.

6.0 Related Work

Since 1994 numerous repositories for all types of information have emerged on the WWW. One of these, at the National Institute of Standards and Technology (NIST), provides an on-line cross-index of available mathematical modeling and statistical analysis software [3]. Called GAMS, it provides centralized access to such items as abstracts, documentation, and source code of the software modules that it catalogs. The GAMS WWW implementation uses a WAIS search across 10 attributes via WAIS-sf "structured fields." However, WAIS-sf structured fields only provide independent attribute-value pairs rather than a coordinated, textual description of a component such as provided by Structured Abstracts.

7.0 Future Work

We intend to show how to automatically generate SAs from classifier information already in an RSL database. This will involve completing the necessary CGI and Perl scripts necessary to extract formal classifier information from the RSL database, formatting the classifier information into Structured Abstracts, and generating output to soft and possibly hardcopy catalogs. The quality of the English text (subject-verb agreement, etc.) will depend on the quality of existing data. However, we can supply language rules to modify the standard SA template and control the text output so that the resulting SA sounds natural. This will allow us to create SAs at very little cost and recover investments made when classifying existing RSLs. We believe Structured Abstracts will work with most RSLs and proposed standard classification schemes [14].

8.0 Conclusion

This paper describes a Reusable Software Library (RSL) interface and search tool implemented using the WWW. In conjunction with a WWW browser, we provide a simple, easy-to-use method to find and extract reusable assets from a RSL and allow distributed access to RSL assets from a variety of platforms. Through the use of HTML forms, we implemented functions normally found in commercial-grade RSLs. Automatic generation of HTML pages and the use of command scripts further allows us to provide different views of the RSL, such as search by subject. Finally, integrating the RSL with WAIS provided a keyword search with minimal effort. Our WWW RSL cost less than 1% of the cost to develop a standard RSL and has quickly gained favor due to its intuitive interface and simple yet powerful information retrieval tools.

To help programmers search for and quickly assess the important aspects of retrieved software, this paper describes the *Structured Abstract*. SAs help

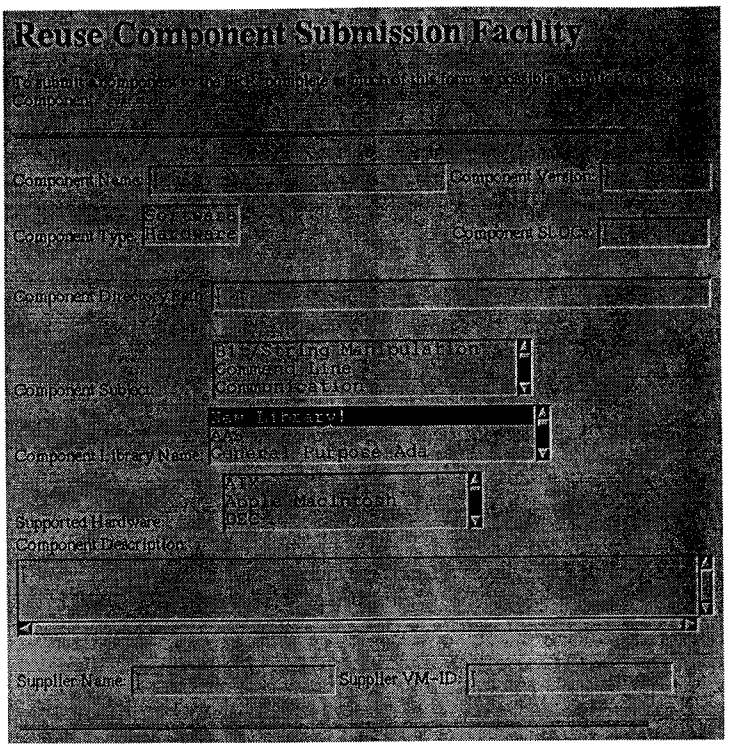

Figure 5. Additional Information Needed for Component Submission

solve the software classification problem by avoiding the long lists of randomly-presented data used in other techniques. SAs help solve the program understanding problem by providing a natural-language description of components in the same format and style that programmers use to convey key information about software.

We use the Structured Abstract in three ways. First, we use SAs to help developers search for components by describing what they need the same way they would communicate that need to a colleague. Second, we use SAs to help quickly assess the component for reuse because all the key information appears in a predictable order. Third, we use SAs to allow developers to describe components they submit to the RSL.

Acknowledgements

We would like to thank the FRR contractor, Allen Matheson of Cimarron (Houston, TX), for his key role in implementing the Mosaic interface to the FRR. We would like to thank Will Tracz for his valuable guidance and input, Tom Loggia of the LFS Reuse Steering Group for his support, and Gary Kennedy, manager of the Software Engineering department at LFS in Bethesda, MD, from whom the FRR receives its funding.

9.0 Cited References

[1] Berg, Klaus, "CLASSLIB - Class Management and Reuse Support on a MVS Mainframe," *Reusability Track of the 1994 ACM Symposium on Applied Computing (SAC'94)* Phoenix, Arizona, 6-8 March 1994, pp. 53-58.

[2] Berners-Lee, Tim, Robert Cailliau, Ari Luotonen, Henrik Frystyk Nielen, and Arthur Secret, "The World Wide Web," *Communications of the ACM,* Vol. 37, No. 8, August 1994, pp. 76-82.

[3] Boisvert, Ronald F., "A Web Gateway to a Virtual Mathematical Software Repository," *2nd International World Wide Web Conference: Mosaic and the Web,* Chicago, Illinois, 17-20 October 1994, *http://gams.nist.gov/*

[4] Furnas, G.W., "The Vocabulary Problem in Human-System Communication," *Communications of the ACM,* November 1987, pp. 964-971.

[5] Lillie, Charles W., "Distributed Network of Reuse Libraries Offers Best Approach to Successful Software Reuse," *Proceedings of the 3rd International Conference on Software Reuse,* Rio de Janeiro, Brazil, 1-4 November 1994, pp. 207-208.

[6] Linn, Marcia C. and Michael J. Clancy, "The Case for Case Studies of Programming Problems," *Communications of the ACM,* Vol. 35, No. 3, March 1992, p. 121.

[7] Maiden and Sutcliffe, "Analogously based reusability," *Behav. & Info. Technology,* Vol. 11, No. 2, March/April 1992, pp. 79-98.

[8] Musser, David R. and Alexander A. Stepanov, The Ada Generic Library. Springer-Verlag, New York, 1989.

[9] Poulin, Jeffrey S. and Martin Hardwick, "Adapting Object-Oriented CAD Database Concepts for Computer Aided Software Engineering," *Proceedings of the International Symposium on Database Systems for Advanced Applications,* Seoul, Korea, April 1989, pp. 201-208.

[10] Poulin, Jeffrey S., and Kathryn P. Yglesias, "Experiences with a Faceted Classification Scheme in a Large Reusable Software Library (RSL)," *Seventeenth Annual International Computer Software and Applications Conference,* Phoenix, AZ, 3-5 November 1993, pp. 90-99.

[11] Poulin, Jeffrey S. and Keith W. Werkman, "Software Reuse Libraries with Mosaic," *2nd International World Wide Web Conference: Mosaic and the Web,* Chicago, Illinois, 17-20 October 1994.

[12] Poulin, Jeffrey S., "Populating Software Repositories: Incentives and Domain-Specific Software," to appear, *Journal of Systems and Software,* fall 1995.

[13] Prieto-Diaz, Ruben and Peter Freeman, "Classifying Software for Reusability," *IEEE Software,* January 1987, pp. 6-16.

[14] RIG Technical Committee on Asset Exchange Interfaces, "A Basic Interoperability Data Model for Reuse Libraries (BIDM)," *Reuse Interoperability Group (RIG) Proposed Standard RPS-0001,* 1 April 1993.

[15] Sindre, G. Karlsson, E. Paul, P. "Heuristics for maintaining term structures for relaxed search," *Proceedings of the International Conference on Database and Expert Systems Applications (DEXA 92),* Valencia, Spain, 2-4 September 1992, pp. 20-25.

[16] Stay, J.F., "HIPO and Integrated Program Design," *IBM Systems Journal,* Vol. 15, No. 2, 1976, pp. 143-154.

[17] Yglesias, Kathryn P., "Limitations of Certification Standards in Achieving Successful Parts Retrieval," *Proceedings of the 5th International Workshop on Software Reuse,* Palo Alto, California, 26-29 October 1992.