

APPLYING HYPERTEXT STRUCTURES TO SOFTWARE DOCUMENTATION

James C. French¹, John C. Knight, Allison L. Powell

{french | jck | alp4g}@cs.virginia.edu

Department of Computer Science², University of Virginia, Charlottesville, VA 22903

Abstract. Software documentation represents a critical resource to the successful functioning of many enterprises. However, because it is static, documentation often fails to meet the needs of the many diverse users who are required to consult it on a regular basis in the course of their daily work. Software documentation is a rich resource that has not been fully exploited.

Treatment of software documentation presents a number of interesting problems that require a blend of information retrieval and hypertext techniques for their successful solution. The evolving nature of a software project and the diverse demands on its documentation present an especially challenging environment. This is made even more challenging by the variety of information resources, ranging from formal specification languages to source code, that must be integrated into a coherent whole for the purpose of querying.

In this paper we discuss work in progress at the University of Virginia. We discuss the issues involved with automating the management of software documentation to better increase its utility. We describe a prototype system, SLEUTH, currently under investigation as a vehicle for software documentation management. The prototype maintains software documentation as a hypertext with typed links for the purpose of browsing by users with varying needs. These links are generated mechanically by the system and kept accurate under update. Appropriate authoring tools provide the system with the information it needs for this maintenance function. Ad hoc querying is provided over the documentation and hypertext documents are synthesized in response to these queries.

1. INTRODUCTION

Software documentation is a critical resource for many enterprises. A software system is a complicated artifact, and both building it and maintaining it require that engineers have accurate, easy-to-modify and easy-to-use documentation. This documentation has to include material about the systems specification, design, implementation, verification, installation, and so on, and for many large systems, it extends to dozens of volumes of text or the equivalent.

The documentation of a software system has to do more than serve as a source of accurate technical information—it has to do this for a variety of users. Engineers at all levels of experience need information ranging from high-level, conceptual material if they are new to a project, to low-

level detailed material if they are involved in development or maintenance. In addition, they need to know how different types of information are related—for example, how an aspect of design is realized in the implementation.

Present approaches to documentation often fail to meet the needs of the many diverse users who are required to consult it on a regular basis. There are many reasons for this, of course, but the thesis of the work described in this paper is that software documentation is *information* and that its creation and use is mainly a problem of *information retrieval*. The work described is an approach to software documentation that exploits modern information retrieval methods including extensive use of hypertext links.

Treating software documentation this way raises a number of interesting problems in information retrieval and hypertext techniques. The evolving nature of a software project and the diverse demands on its documentation present an especially challenging environment. This is made more challenging by the variety of information resources, ranging from mathematical specification languages to natural language to source code in a variety of high-level languages, that must be integrated into a coherent whole for the purpose of querying.

In this paper we discuss the issues involved with automating the management of software documentation to better increase its utility. We describe the mechanics of a prototype system, SLEUTH³, that is currently under investigation at the University of Virginia as a vehicle for software documentation management. Although it is a general-purpose system, many facets of its design are influenced by the goal of providing high quality and extremely accurate software documentation for safety-critical systems. Safety-critical systems are those systems where the consequences of failure are extreme, perhaps resulting in a large financial loss or endangering human life. Such systems require documentation that is especially precise, accurate, and easy to use. The latter property is particularly important because it is essential that engineers involved with the system be able to obtain *all* of the information that they need when they need it.

The SLEUTH system consists of both an authoring and a viewing environment. The prototype system maintains software documentation as a hypertext with typed links. Two unique features of SLEUTH are: (1) that these links are installed mechanically by the system and kept completely accurate under update, and (2) that links are provided between a variety of document types including current system source code. Ad hoc querying is provided over the documentation and a hypertext document is synthesized to hold the response to a query.

2. THE PROBLEM DOMAIN

The state of the art in software documentation includes desktop publishing systems, plain text, government standards and experimental systems. These approaches are generally not sufficient for the task at hand.

Commercial desktop publishing systems, such as FrameMaker and Interleaf, facilitate writing, viewing, and editing large documents and sets of documents. These systems allow documents to be collected into sets and automate the creation of indices and tables of contents. Most provide a WYSIWYG editing environment and tools for creating diagrams and mathematical formulae. They provide useful tools but do not offer document structure, search facilities, support for the integration of material in formal languages (such as source code), or guidance on what should be written.

Software-development standards, such as DoD-STD-2167A [4] and RTCA/DO-178A [17], generally prescribe the documents that have to be produced and sometimes their formats but seldom provide justification for the inclusion of particular documents. Many documents are required by DoD-STD-2167A but the standard merely states the name of the document; it does not prescribe the content, organization, or even the format. RTCA/DO-178A⁴ has a complete section on documentation and ties documentation in with regulatory approval. But it still prescribes nothing more than the types of document and a brief (usually one sentence) description of the required content. In support of the standards that are in widespread use, there are systems that generate documents automatically in required formats given a collection of system work products such as structured designs and source code. While there are some useful features in this approach, the documents produced often end up containing very little useful information. And they are certainly not easy to use.

The current approaches lead to software documents that do not serve the engineering community well for a number of reasons including the following:

- The preparation of documents requires enormous effort that is not viewed as commensurate with the benefits of having it. Engineers resent developing documentation.
- Software documents are notoriously inaccurate. Even if the documents are prepared carefully, they are rarely maintained. As a result, engineers do not trust documentation and argue that the only accurate documentation is the source code because that is what executes. However, all of the information needed by software engineers cannot be embodied in the source code. The importance of documentation in the software development process is discussed by Parnas *et al.* [16]. Inaccuracies or inconsistencies in documentation hinder that process.
- Software documentation is difficult to use because it is static and hence takes on a single form. This means that a single form has to meet the needs of the various interested parties. In practice, this compromise satisfies the needs of none very well. An important aspect of this static structure is that it rarely if ever contains a comprehensive index. This means that users are basically left to their own devices when searching for information.

A number of experimental systems have been developed in an effort to deal with the situation and to experiment with other approaches to managing software documentation. Four such systems are DIF[10], SODOS[11,12], HyperCASE[2] and LaSSIE[3]:

DIF. The Document Integration Facility is an environment to develop, maintain and browse the documentation associated with a large-scale software system. The documents produced in DIF are the documents associated with each phase in the software life cycle, from requirements analysis and specification to testing. Each segment of a document is viewed as an object and hypertext links between documents are relationships between objects. These relationships are stored in a relational database. Searching is allowed only on predefined keywords.

SODOS. The Software Documentation Support Environment integrates the ideas of an object-based model of the software life cycle with a database management system. The information gathered at each stage of the software life cycle is put into structured documents. The database consists of all documents associated with a project. SODOS provides a document interface that

allows users to modify and query documents. Possible query terms are defined by the author.

HyperCASE. HyperCASE integrates the two concepts of hypertext and CASE tools. HyperCASE uses hypertext to link related information in documents associated with the software life cycle. HyperCASE provides a suite of tools for creating, editing and presenting documents, a repository and a data dictionary. HyperCASE provides a number of browsing capabilities but is intended mostly for use by managers and software engineers currently working on a project. Little provision is made for alternate uses of the documents.

LaSSIE. LaSSIE focuses on the structure of a large software system and provides architectural, operational, feature and code views of the system. LaSSIE maintains a knowledge base to attempt to answer programmer questions. However, LaSSIE is specifically intended to represent structure and to answer questions about large software systems; it does not provide easy access to general descriptive overview information.

All of these systems are helpful and provide capabilities for the software engineer that are far superior to those available with traditional text documents. But they also have their limitations. For example, both DIF and SODOS integrate database management systems with the software documentation associated with the software life cycle. Queries are allowed on predefined keywords. This can be moderately helpful to users; however, it places an additional burden on the author. When predefining keywords, the author is forced to anticipate the needs of end users. If the documentation is used in a way that the author did not anticipate, desired terms might not be available for searching. This is similar to the problem often encountered with book indices. An individual may be certain that a key piece of information is included in the book, having read it once before, but be unable to locate the topic again because it was not included in the index. SLEUTH avoids this problem by indexing the full text of documents in the system.

With SLEUTH, the goal is to provide accurate documentation that addresses the needs of all of the users. This goal requires that the system provide a hypertext structure to permit smooth navigation through the material based on the users' needs and on the information content. It also requires that a query mechanism be provided to permit information to be retrieved based on queries that are not constrained by prescribed keywords or other static context limitations.

In considering the scope of the problem that software documentation presents, it is instructive to try to identify the set of potential users and to itemize their individual needs. The documentation for safety-critical systems, one of the systems that SLEUTH is designed to support, has many potential users including the following:

- The software engineer who must design a new system component. This engineer will need detailed information on all work products and all components to determine the interactions of the new component with the existing ones.
- The project manager preparing a status report. This user needs relatively high level yet very specific information on the areas of the project covered by the status report.
- The software engineer new to the project. This engineer will need high level and conceptual information presented in such a way that more detail is available when needed.
- A maintenance programmer charged with making an enhancement or change. This user needs detailed information on a specific system area, along with interactions with other components. As detailed by Soloway *et al.* [20], documentation should help the maintenance programmer

to have a full understanding of the system before making a change so that the system design can be consistently maintained.

- A regulator who must determine if a legal requirement has been implemented. The regulator may need information at varying degrees of detail to complete this task.

This is a very wide spectrum of needs. To address these needs, SLEUTH depends in large part on the hypertext structure of the documentation. This structure allows users to tailor both the degree of detail and the specific content to their needs. However, as discussed by Marchionini and Shneiderman [15], while hypertext is an excellent environment for browsing and providing varying levels of detail, it is not always best for providing quick answers to specific questions. To address those needs, SLEUTH provides a search engine, in addition to hypertext.

3. INFORMATION RETRIEVAL CHALLENGES WITH SOFTWARE DOCUMENTATION

The diversity of information contained in software documentation together with the myriad uses to which it is put conspire to make this an especially demanding arena in which to apply information retrieval techniques. It is also an arena in which there is a great deal of freedom to innovate. We are in a position to exploit fully any characteristic of software documents that can help improve retrieval effectiveness. The SLEUTH prototype provides an environment where we can conduct experiments, get user feedback, and refine our techniques. Particular areas of investigation are described below.

- Navigational aids for browsing and accessing the documents. This includes static system-defined navigational links and dynamic user-defined links. The system-defined links are those that are generated when the online version of the documentation is built. User-defined links result from queries.
- Flattening the hypertext for hardcopy. For the foreseeable future there will be a need for a hard copy of software documents. Although dynamic links are no longer possible, we would like to retain the utility of the static links as much as possible. We are approaching this by means of typed hypertext links, where the type is visually conveyed by typography or color or both.
- Full-text retrieval capability. We have expanded the notion of full-text to include source code text, requirement and specification language texts, as well as any other component of a software documentation library.

There have been some successful applications of IR technology to software engineering such as Frakes and Pole's work [7] with IR techniques to search for and retrieve components from a software reuse library. Wood and Somerville [21] have also explored this area. But for the most part, software documents have been relegated to hardcopy to languish largely unread.

The complete integration of all software documents—from requirements to source code—is necessary so that they are regarded as a coherent whole and treated as such for the purpose of information retrieval.

4. HYPERTEXT LINKS IN SLEUTH

There is an interesting and important difference between information in the form of software documentation and many other kinds of information—from the outset, software documentation can be *designed* for retrieval. Although legacy systems exist with voluminous, unstructured, and inaccurate documentation, new systems, and especially safety-critical systems, can be docu-

mented with a view to ensuring that the form and content of the documentation is optimized for retrieval that is easy, accurate and complete.

This is a novel situation that can be quite difficult to appreciate by both software engineers and information retrieval specialists. But it presents a tremendous opportunity because techniques that are useful yet have limitations in traditional circumstances can be used optimally in software documentation. An example is hypertext links—as noted above, in SLEUTH they are installed mechanically. The primary reason is to ensure the reader of completeness and accuracy rather than efficiency or cost savings. In safety-critical systems, the cost of failure is so high that cost savings during development are very much a secondary concern.

Completeness ensures that if the author determines that a link should be present to alert the reader to other information in the document, then it will be present. Accuracy ensures that all links throughout the document that are supposed to be to a particular point will indeed all be to that point. Once the installation is verified, the reader can be confident in each and every link. Because they are installed mechanically, any item that should be a link will be no matter where it appears and no matter how many times it appears.

A special case of link creation is the set of links to the *source code* of the software that is being documented. It is essential in the documentation of a safety-critical system that the source code be included and that it be the right source code. In many cases, documentation either does not include the source code or it contains a copy. A copy is actually worse than omitting the source code because the copy might be out of date.

SLEUTH generates hypertext links to the source code files actually used to build the software system so it is guaranteed to be the source code as seen by the compilers. Within SLEUTH the source text is actually reformatted when it is opened for viewing and so the appearance on the screen for the viewer is compatible with the remainder of the documentation.

As well as the links themselves, SLEUTH builds a “map” of the source-code hypertext links for users. This map is, of course, based on the directory structure of the files in which the source code is stored. The map is merely a convenient screen representation of the structure that facilitates navigation. The map is somewhat more than a presentation of the directory structure that would be displayed by a file manager, however. Included is a *table* of links for each software module rather than just one. This permits the collection of links to things such as a module’s interface (e.g. header file in C++), its implementation (e.g. implementation file in C++), and any other module-specific documentation into a single structure. When users select a filename, they are presented with a menu implemented as a FrameMaker scroll-box that offers the choice of source, header or descriptive files (see Figure 1). When users choose to view a header or source file, it is imported into a newly created FrameMaker document and automatically displayed. While this causes a slight increase in loading time, it helps to assure that users are viewing the current version of the source code rather than an out-of-date copy.

5. MECHANICS OF SLEUTH

An initial prototype toolset was created using a World Wide Web browser to investigate the merit of hypertext and searching capabilities in software documentation that was stored in HTML. The prototype was demonstrated and response to its capabilities was positive, but it was abandoned because the Web browser did not permit the necessary display flexibility (the paragraph formatting capabilities were limited; the number of different hypertext links that could be represented was limited by the number of possible character formats; and there was not full control

over the display mechanism).

In the current system, FrameMaker is used for both authoring and viewing. It provides a WYSIWYG editing environment and supports the creation of hypertext links. It also provides basic navigational features, a toolkit for customization, and it can be used to produce effective hardcopy versions of the documents precisely because it is a document-preparation system. Using FrameMaker to support authoring, viewing and printing eliminates any potential inconsistencies that might have occurred if different representations had been used for these three purposes.

In addition to the basis provided by FrameMaker, the major components of the current SLEUTH prototype system are as follows: configurable hypertext filter generators; a filter to produce a directory-structured index for source code (or any other notation); and a search engine and associated interface that allow full text searching on the documents in the collection.

The major components of SLEUTH are shown in Figure 2. The initial set of documents is authored using FrameMaker and the author configures the hypertext filters using the configuration file. Once the filters have been generated, they are used to create a hypertext containing the initial documents.

5.1 Document Creation

The author uses the standard FrameMaker WYSIWYG editing environment to compose the initial set of documents and any associated figures and tables. The SLEUTH system provides a document template that defines paragraph types, the formats to differentiate the typed hypertext links and other document formatting information. Authors are required to use these templates so that document styles and hypertext link types are consistent across documents in a collection. Of course, consistent terminology is also necessary for a coherent collection.

The author maintains a list of terms to become hypertext links while creating the documents together with the location to which the link should point if it is known at the time that the term is identified. We have mentioned the problems that can arise in other experimental systems when authors are required to identify keywords for searching. While the work required by authors to

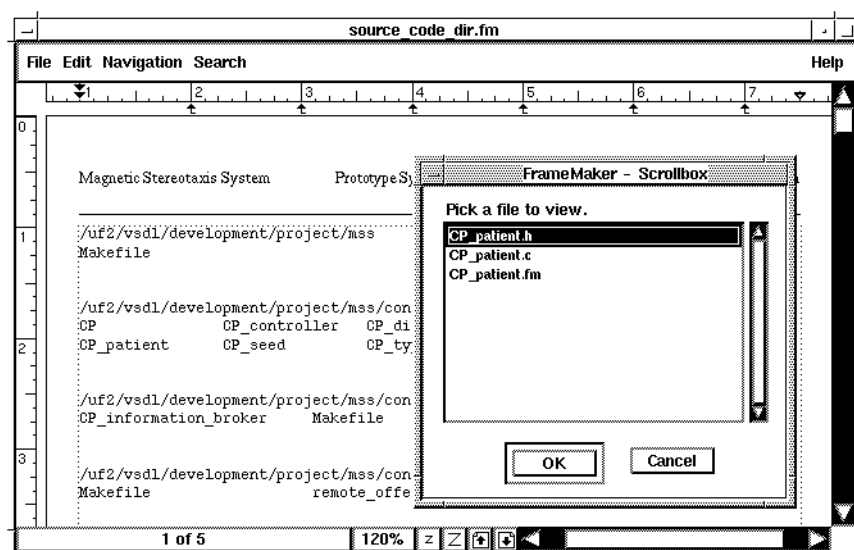


Figure 1 Interface to directory-structured index.

identify keywords and hypertext links is similar, SLEUTH avoids the problem of limited searching capability by using full-text searching. In addition, while requiring that the author maintain a list of terms to become hypertext links is a cognitively intensive task., it is not a labor-intensive or repetitive one because the links are installed mechanically. If new links are required for any reason at any time or if the set of links changes in some other way, the entire set of material seen in the viewing environment can be recreated *automatically* merely by executing the various filters again. In addition, maintaining this list while authoring documents encourages consistent term usage.

Clearly, a hardcopy version of the documentation will be desired in most cases. Other filters (also shown in Figure 2) that use the FrameMaker cross-reference facilities preserve some of the functionality of hypertext in the hardcopy version by displaying the page number in the collection of the online link destination. Once again, the author uses the recorded hypertext information to initialize the cross-reference filters (as well as the hypertext ones).

5.2 The Filters

The hypertext and cross-reference filters operate upon the MIF (Maker Interchange Format) representation of FrameMaker documents. MIF files consist of header information that defines the document format and the entire text of the document that has been marked up to include the formatting information. The filters locate specified terms of interest, and, given the appropriate information from the author, they insert the markup language fragments for the desired hypertext links. The filters ignore the header information and most of the markup tags—inserting a hypertext link in those areas would be an error.

The filters make it very simple to have multiple terms or multiple forms of the same term

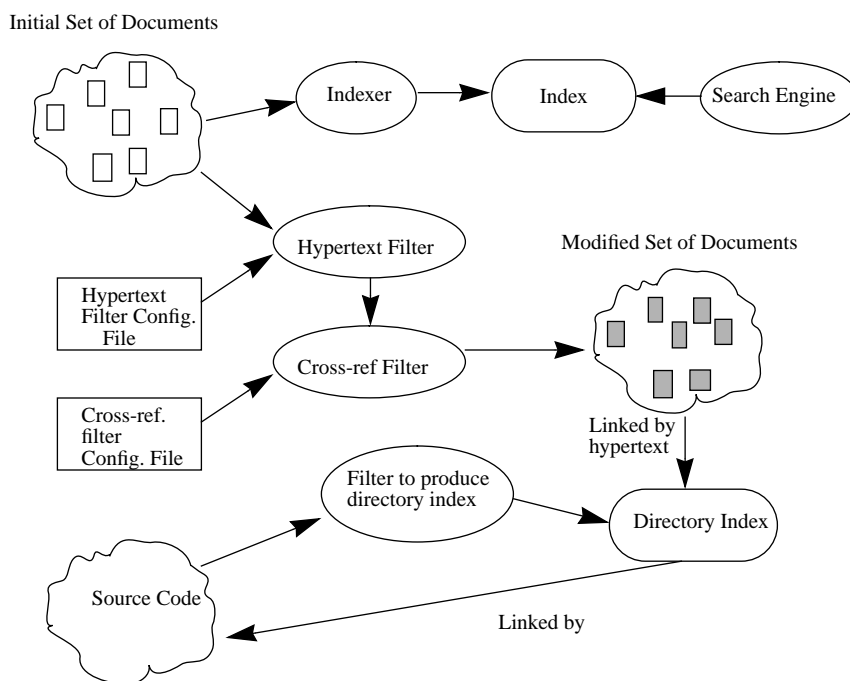


Figure 2 SLEUTH Authoring Architecture

point to a given destination. In some cases, however, the author might want a given term to point to multiple destinations. For example, a single term could have both a definition in the glossary and reference an appendix entry. This capability has not been implemented in the current prototype. Given a clear way to differentiate the multiple destinations, this capability could be implemented using a menu of choices similar to the menu used to choose different types of source code information.

The filters are implemented as a series of *lex* [14] programs that are generated using the configuration files created by the author. The configuration files record the phrase to match, the link type, the destination document and the anchor name (if desired) within the document for each term that is to become a hypertext link or cross-reference. Available link and cross-reference types are those to appendices, source code, figures, tables, glossary entries, and other related documents. At present, it is not clear whether automatic determination of link type, such as that explored by Allan [1] is feasible for this application area.

Once the configuration files have been created, a rule file generator script is used to format them for input to *lex* (see Figure 3). One *lex* input file (i.e. specification) is created for each configuration file. SLEUTH provides *lex* header information and utility functions for both hypertext and cross-reference creation, and the appropriate headers and footers are concatenated with the rules files to create full *lex* specifications. Finally, the filters are produced by running the *lex* utility and compiling the resulting C programs.

The source-code index is generated using the same principles. Source-code file names that are formatted to represent their location in the directory tree are maintained in a FrameMaker document. Each directory name and source file name is a hypertext link that provides access to the source code.

Anchors are created at the destination for each hypertext link and cross reference. These anchors are inserted into the documents before the hypertext is generated for the first time—they only need to be inserted once.

Whether the standard hypertext and cross-reference facilities or the filters are used, it is necessary to maintain a set of original documents. To avoid inconsistencies, all modifications to the document collection are made to the original copies. The hypertext is then re-generated. If the author wants to expand the hypertext, the author updates the filter initialization files, re-generates the filters, and then re-creates the hypertext.

Since the filters are run on MIF files, not the standard binary files that FrameMaker produces, it is necessary to convert file formats as part of the filtering process. The hypertext is created when the filters are run on copies of the archived documents. The prototype system produces transformations such as those shown in Figure 4.

In the segment that has been modified, underlined terms designate hypertext links with the typeface denoting link type. In instances where color is available, link types are denoted using different colors instead of typeface. The page numbers are cross references inserted to facilitate the

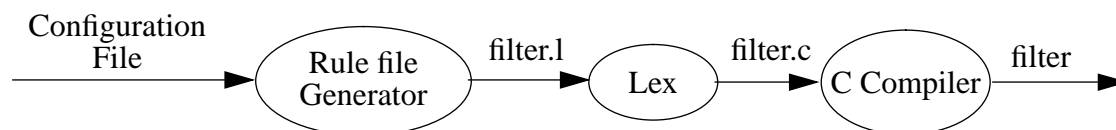


Figure 3 Filter creation.

use of the documents in hard copy form.

The documents currently being used to test the system are divided into chapters that discuss major system components, appendices which discuss subtopics, collections of figures and tables, sets of documents describing code libraries and classes, and a glossary. The current implementation provides a set of hypertext link types to indicate links to each of these document types.

In the example shown in Figure 4, italics represent links to the glossary page. No page numbers are necessary for this type of link. Block letters and the cross-reference format 'page F#' denote links to figures. Helvetica font and the cross-reference format 'page A#' denote links to appendices. The appendices contain supporting information which is important but not integral to understanding the system at large. Finally, boldface type and the cross-reference format 'page #' denote links to other main topic areas. In addition, there are link formats for source code and tables. The transformations shown in Figure 4 are the hardcopy transformations. The online version of the documents need not include 'page #', etc. as that would be redundant.

5.3 The Search Engine

The SLEUTH system currently utilizes a WAIS (Wide Area Information Server) [13, 5] search engine. WAIS is intended for distributed information retrieval and based on a client-server model of computation. We are using a variant of WAIS that allows searching based on simple Boolean keyword expressions. The search engine returns documents in a ranked list with scoring based upon the number of times a term occurs in a document, the location of the words, the frequency of those words in the collection of documents and the size of the document [5]. To

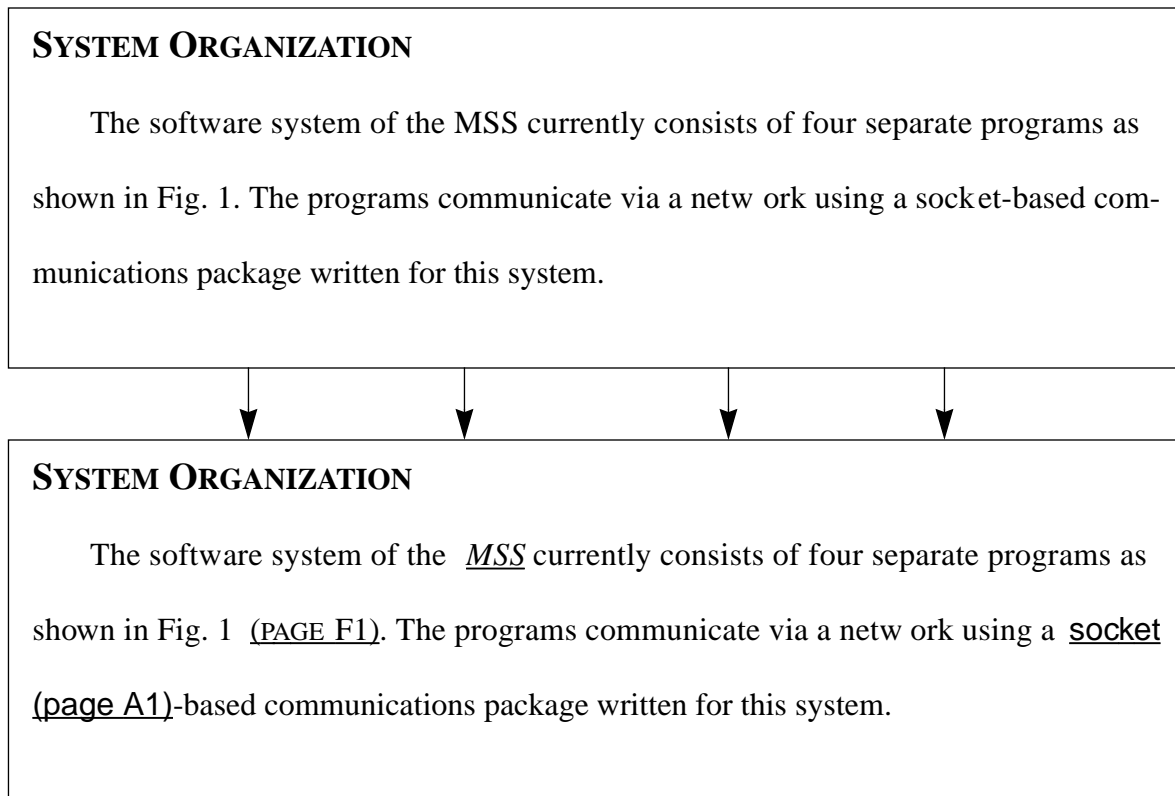


Figure 4 Document fragment before and after modification by hypertext and cross-reference filters

improve search granularity, each paragraph is treated as a document.

The *indexer* creates a table of the document and paragraph locations of terms in the documentation text. FrameMaker documents are stored either in binary or MIF format but neither of these formats can be indexed effectively, and so SLEUTH does not store the documents in the WAIS database. Instead, it provides utilities that extract the text from the documents (which are stored elsewhere), and then uses the WAIS indexer to index the text. After indexing, this text is discarded and only references to the original documents are retained in the WAIS index.

The documents in the SLEUTH system are indexed on a paragraph by paragraph basis, and so hypertext endpoints are added to all paragraphs in the documents. This allows SLEUTH to open documents for users at the nearest paragraph for any search response.

5.4 User Interface

The on-line interface presented to users is the standard FrameMaker viewing environment that has been enhanced using the FrameMaker Developer's Toolkit [8, 9]. This version can be navigated using the typed hypertext links in the traditional point and click fashion. An example viewing session is shown in Figure 5. Multiple windows can be viewed at the same time, allowing users to look up a definition and view a referenced figure while examining the document of interest. In this way, users can have a set of relevant documents arranged on the screen to explore a subject area in detail.

An intuitive and stable interface to the search engine was incorporated into the system using the FrameMaker Developer's Toolkit. If desired⁵, the WAIS search engine could be replaced without changing the user interface. Users begin a query by choosing the "Search" pull-down menu and enter details into a pop-up window. The response currently consists of a list of hypertext links that point to the documents and paragraphs in which the search terms are located. This list is returned in a generated FrameMaker document that is opened automatically.

6. INVESTIGATIONS AND CURRENT ISSUES

The most formidable challenge is to organize the online documents for maximum flexibility to answer unanticipated queries. Users will have varying backgrounds and will require access at different levels of specificity. A hierarchical hypertext provides the mechanism to gradually expose detail. The difficulty is summarizing the information returned by an ad hoc query to the level appropriate for the particular user. This implies the need to synthesize new documents on-the-fly, documents that meet the specific needs of the user issuing a query. Automatic abstracting will be crucial. Techniques for automatically abstracting documents have been around for over 35 years. Edmundson and Wyllys [6] survey their own work and that of three other researchers: Baxendale, Luhn, and Oswald. More recently Salton et al. [18,19] have discussed passage retrieval and theme generation to aid in the summarization and navigation of large complex texts. These and other techniques will be necessary to merge fragments of this multimedia (formal specification languages, narrative texts, figures and diagrams, tables, source code from multiple languages, etc.) environment into a cogent response to a query.

For most purposes in software engineering, we would expect a query response to be a new

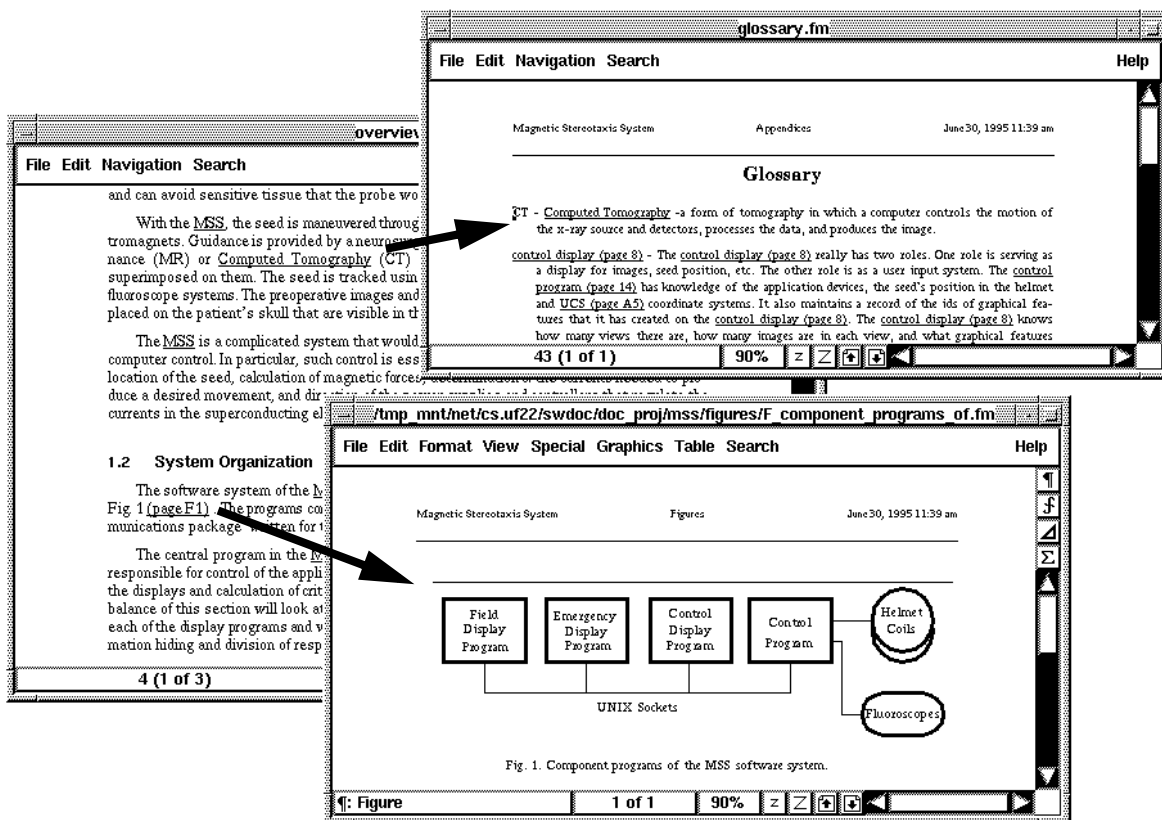


Figure 5 Example browsing session.

“fact dense” document on a specific user-defined topic. It must be comprehensible to the intended reader and it must provide sensible access into the documentation corpus when that reader requires further elaboration. Our initial studies have barely scratched the surface of what is possible. We have looked at paragraph level retrieval of narrative text where the document synthesized in response to a query consists of a concatenated list of sentences containing appropriate keywords. The links in this synthetic document are implicitly user-defined links.

In all cases our retrieval engine returns paragraphs. Our initial examination considered three ways to construct the summary response from the matching paragraphs returned by a query.

- The sentence containing the keywords was retained.
- The sentence containing the keyword plus the immediately preceding and immediately following sentences comprised the fragment.
- The whole paragraph containing a keyword match was retained.

In each case the fragments retained were concatenated to form the response document. Note that with sentence level retrieval it is possible to duplicate some sentences when using the second strategy above. Care was taken to elide all duplicates.

As an additional navigational aid, the concatenated fragments in the response document were delimited so that the reader could tell when contiguous chunks were from different underlying sections of software documents. The delimiter included a hypertext link to the beginning of the containing document section while the fragments were linked directly to their occurrence within

the document. Thus a reader finding a fragment useful but wanting some further elaboration on the point is provided access to the containing passage; in case the fragment suggests that the whole topic is useful, the reader has immediate access to the containing document section. In either case the normal intra-document hypertext links are available within the containing document.

Thus a response document is sufficiently differentiated so that users can immediately tell which underlying software documents contain appropriate material and relatively how much material each document contributes. Although our experience with this limited document abstraction mechanism is very preliminary at this point, it does seem clear the all three strategies above have a role to play. Much of the difference in strategies is simply a matter of preference. Different users seem to prefer different presentations and that is to be expected. It suggests that SLEUTH should provide a user profile mechanism so that the system can be configured by individual users for their specific preferences.

We intend to expand these experiments to include the other less standard “texts” available to us in the SLEUTH repository.

7. EVALUATION

It is, of course, important to evaluate systems such as the one described here. Is a system like SLEUTH really of any use? Does it improve the access to software documentation that we established as a goal? A complete evaluation of such a system requires an elaborate empirical evaluation in an industrial environment that would enable a statistically valid conclusion to be drawn. Such a rigorous evaluation has not yet been done.

To gain insight into the performance of the approach that SLEUTH embodies, we have undertaken a feasibility study in which a significant fraction of the documentation for a single project, an experimental medical robotic system, has been prepared. The documentation provides the following major facilities:

- Over 100 pages of structured textual explanations of the various subsystems and appendices describing support features and theoretical background.
- A glossary of terms.
- A set of approximately 25 descriptive figures and tables.
- Hypertext links for over 40 words and phrases.

A variety of users have viewed this documentation and the preliminary subjective evaluation is positive. All of the authoring tools (link generators, etc.) have been used extensively and found to be very satisfactory both in terms of their execution-time performance and the resulting hypertext. A field evaluation of the utility of SLEUTH in a safety-critical application, the UVa nuclear reactor control program, is currently underway. A more detailed assessment of the system is planned.

8. CONCLUSION

Software documentation is information and can benefit from the application of information retrieval techniques. We have described a system that provides users with software documentation that has a much richer structure than is normally the case, that is equipped with extensive hypertext links that are installed mechanically, that incorporates the software source code as part of the

documentation, and that includes a general-purpose search facility.

Although we have no statistical data showing a significant performance improvement from the user's perspective, anecdotal evidence suggests that the SLEUTH environment is far superior to traditional static software documentation.

Acknowledgements. We would like to thank Greg Wohlford for his work with the WAIS server and the query-response fragment size investigation.

REFERENCES

1. Allan, J. (1995). *Automatic Hypertext Construction*. Ph. D. dissertation, Cornell University. Also technical report TR95-1484.
2. Cybulski, J. L., & Reed, K. (1992) A Hypertext Based Software Engineering Environment. *IEEE Software*, 9(2), 62-68.
3. Devanbu, P., Selfridge, P. G., Branchman, R. J. & Ballard, B. W. (1990). LaSSIE: a Knowledge-based Software Information System. *IEEE Proceedings of the 12th International Conference on Software Engineering*, 249-261.
4. DOD-STD-2167A Military Standard: Defense System Software Development (1988), U.S. Department of Defense, Washington, D. C.
5. Edguer, A. (1995). Frequently Asked Questions (FAQ) for FreeWAIS 0.5. Clearinghouse for Networked Information Discovery and Retrieval (CNIDR). <http://www.cnidr.org/Software/freewais.html>.
6. Edmundson, H. P. & Wyllys, R. E. (1961). Automatic Abstracting and Indexing—Survey and Recommendations. *Communications of the ACM*, 4(5), 226-234.
7. Frakes, W. B. & Pole, T. B. (1994). An Empirical Study of Representation Methods for Reusable Software Components. *IEEE Transactions on Software Engineering*, 20(8), 617-630.
8. Frame Technology Corporation (1993). *Frame Developer's Kit for Specific Platforms - UNIX*.
9. Frame Technology Corporation (1993). *Frame Developer's Kit Programmer's Guide*.
10. Garg, P. K. & Scacchi, W. (1990). A Hypertext System to Manage Software Life-Cycle Documents. *IEEE Software*, 7(3), 90-98.
11. Horowitz, E. & Williamson, R. C. (1986). SODOS: A Software Document Support Environment—Its Definition. *IEEE Transactions on Software Engineering*, SE-12(8), 849-859.
12. Horowitz, E. & Williamson, R. C. (1986). SODOS: A Software Document Support Environment—Its Use. *IEEE Transactions on Software Engineering*, SE-12(11), 1076-1087.
13. Kahle, B. & Medlar, A. (1991). An Information System for Corporate Users: Wide Area Information Servers. *Online Magazine*, 15(5), 56-60.
14. Lesk, M. E., & Schmidt, E. (1975). Lex - A Lexical Analyzer Generator. Computing Science Technical Report 39, AT&T Bell Laboratories, Murray Hill, N. J.
15. Marchionini, G. & Shneiderman, B. (1988). Finding Facts vs. Browsing Knowledge in Hypertext Systems. *IEEE Computer*, 21(1), 70-80.
16. Parnas, D. L., van Schouwen, A. J. & Kwan, S. P. (1990). Evaluation of Safety-Critical Software. *Communications of the ACM*, 33(6), 636-648.
17. RTCA/DO-178A (1985) Software Considerations in Airborne Systems and Equipment Certification, Radio Technical Commission for Aeronautics, Washington, D. C.
18. Salton, G., Allan, J., and Buckley, C. (1993). Approaches to Passage Retrieval in Full Text Information Systems. *Proceedings of the 16th Annual International ACM SIGIR Conference*

on Research and Development in Information Retrieval, 49-58.

19. Salton, G., Allan, J., Buckley, C. and Singhal, A. (1994). Automatic Analysis, Theme Generation, and Summarization of Machine-Readable Texts. *Science*, 264(5164), 1421-1426.
20. Soloway, E., Pinto, J., Letovsky, S., Littman, D., Lampert, R. (1988) Designing Documentation to Compensate for Delocalized Plans. *Communications of the ACM*, 31(11), 1259-1267.
21. Wood, M. and Somerville, I. (1988). An Information Retrieval System for Software Components. *SIGIR Forum*, Spring/Summer, 11-25.