javaJAM

SUPPORTING COLLABORATIVE REVIEW AND IMPROVEMENT OF OPEN SOURCE SOFTWARE

A THESIS SUBMITTED TO THE GRADUATE DIVISION OF THE
UNIVERSITY OF HAWAI'I IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

INFORMATION AND COMPUTER SCIENCES

AUGUST 2000

By
Monir Hodges

Thesis Committee:

Philip M. Johnson, Chairperson
Wesley Peterson
Daniel Suthers

We certify that we have read this thesis and that, in our opinion, it is satisfactory in scope and quality as a thesis for the degree of Master of Science in Information and Computer Sciences.

THESIS COMMITTEE

_____

Chairperson

_____

_____

To my family, Michael, Emily, and Sarah,
for their continuous moral support
throughout this project.

To my colleagues at
Honolulu Community College who
provided encouragement and freedom to experiment.

Acknowledgements

This research would not have been possible without the help of Philip Johnson, and the assistance of Robert Brewer. I also would like to thank all the members of the CSDL research group for providing initial feedback and sitting through several presentations of *javaJAM*. Your feedback has been valuable in helping me to provide a better interface.

I would like to thank Ty Kroll for doing the original investigation on enhancing the standard Javadoc for *javaJAM*.

Abstract

Development of Open Source Software is in many cases a collaborative effort, often by geographically dispersed team members. The problem for members is to efficiently review documentation and source code and to collect and share comments and annotations that will lead to improvements in performance, functionality, and quality.

javaJAM is a collaborative tool for assisting with the development of Open Source Software. It generates integrated documentation and source code presentations to be viewed over the web. More importantly, *javaJAM* provides an interactive environment for navigating documentation and source code and for posting annotations.

javaJAM creates relationships between sections of documentation, source, and related comments and annotations to provide the necessary cross-referencing to support quick and efficient reviews.

javaJAM was evaluated in a classroom setting. Student teams posted projects for team review using *javaJAM* and found it to be an easy way to review their projects and post their comments.

Table of Contents

List of Tables

List of Figures

1. Introduction

1.1 The Challenge of Open Source Software Development

There was a time when a lone software developer could sit in a corner for weeks or months or even years and quietly hack out a package and hope for some level of success and recognition. While it is possible for solo developers to create small, niche applications like the MemMax utility for freeing up RAM that is offered at AnalogX.com [1], much of the software development in the 21st century will be for large applications.

Applications are usually developed under intense circumstances. These circumstances have two salient features: competition and complexity. Whether an application is developed as Open Source, Shareware, or for shrink-wrapping, these two circumstances, competition and complexity, still apply since the first one to get well known will be the one most likely to gather a loyal following, find success and gain a competitive advantage.

The pressures of competition can greatly shorten the desired lapse time from product conception to product delivery. Whether it is a new idea, or just the next product cycle, there is often a compelling reason to keep the cycle as short as possible while keeping the overall quality of the software at a reasonable level.

Complexity requires in many cases that software is feature-complete and includes a rich user interface requiring minimal learning curves. While users may not take advantage of more than a small portion of the feature set, they are attracted to feature-rich software as can be seen by the success of PaintShop Pro version 6 [15] and the continual evolution of the Microsoft Office suite.

In response to competition and complexity, Open Source Software is rapidly gaining acceptance as a viable alternative model for high quality software development and innovation. The problem that developers face when working with this model is that they are often members of a team in which one or more members will have no or little opportunity for face-to-face meetings with other team members. This presents a hurdle for successful application development. Reviewing documentation and source code, and collecting and sharing notes and annotations that will lead to improvements in performance, functionality, and quality becomes the challenge if it is to be accomplished in a timely fashion.

1.2 Providing a Web-Enabled Collaboration Tool

One solution to supporting Open Source Software development is to create a tool that facilitates documentation and code review and provides for the collection of annotations as they relate to the section of documentation and code that the viewer has reviewed and has determined a need to make a comment or a suggestion. Such a tool would have to work behind the scenes to establish relationships between sections of documentation and source and also the annotations as they are posted and accumulated. Additionally, this tool would have to be easy to learn and use so that it would not get in the way of the overall software development task. This tool would be very easy to deploy if it could take advantage of the Internet and be accessible using any web browser.

javaJAM has been designed to accomplish exactly this. To get the most benefit from the Open Source approach and philosophy *javaJAM* provides a software collaboration tool that facilitates under an Open Source model the publication of software packages and the collection of review comments and notes for application documentation and source. These review comments are referred to, individually, as annotations. *javaJAM* relies on Javadoc for the generation of program documentation, and also incorporates the Java source code into the presentation.

Javadoc is Sun's implementation of an extensible documentation generation system that parses Java source code to extract and publish class, constructor, and method interface documentation in HTML format for the web. Javadoc does impose conventions in order for useful documentation to be generated. The Javadoc source code must contain in-line comments that follow these conventions.

The *javaJAM* extensions to Javadoc provide additional features for reviewing or creating annotations while reviewing source code and documentation. For developers already familiar with Javadoc, *javaJAM* is a familiar Javadoc environment that has been extended with carefully placed buttons that make it possible to also review source code and to post annotations.

Collectively, the annotations accumulated and reviewed for a software package can strongly influence the implementation of the next version of the software package. Such influence includes but is not limited to coding practices, accuracy and completeness of documentation, and software quality, functionality, and efficiency. Collecting and organizing these annotations is the key to creating a successful collaborative environment.

1.3 The *javaJAM* Implementation

javaJAM has been implemented to demonstrate that a web-enabled collaboration tool can improve the process of developing Open Source Software. The "*java*" portion of the name reflects the fact that it is a Java tool in part layered over Sun's Javadoc technology. The "*JAM*"

portion of the name is loosely based on the idea of “jamming” as musicians say. Jamming is a form of collaboration. The *javaJAM* tool is designed with three major parts. The first is the web-enabled presentation. This is what the user sees when navigating through documentation and source and creating annotations (see Section 4.3). The second is the *javaJAM* enhanced Javadoc that reads Java source and creates a web-enabled presentation of documentation and source (see Section 4.1). The third is the management tools used by an administrator to maintain a *javaJAM* server.

Manually posting Javadoc generated documentation on the web is simple. Simply reading code into an HTML editor and bracketing the code with <PRE> tags can easily accomplish posting Java source on the web. Once the code is published, collecting comments and annotations would require more work. The simplest implementation would be to have a form and the form would accept comments and annotations and email them to the author. By creating a hyper-linked page it would be possible to navigate between documentation and source and to email comments. A fair amount of manual labor is required to set this up, and the comments and annotations are not easily available for review. It can also be difficult for the reviewer to decide where exactly comments and annotations relate to documentation or source.

A better alternative is to automate the presentation of documentation and source; to provide a user-interface for navigating between the two with additional functionality for posting comments and annotations; and, most importantly, to create explicit relationships between sections of source and documentation and any comments and annotations posted in response to a review of these sections.

javaJAM stores all documentation, source, and annotations in HTML files. Each section (see Section 4.1.3 for details) contains anchors and hyperlinks. Each posted annotation also contains anchors and hyperlinks. These anchors and hyperlinks establish the relationships necessary to navigate from an annotation to the relevant section of document and to the relevant section of source. These relationships form triangles; from any one point navigating directly to either of the other two is possible.

The *javaJAM* interface is designed to look and feel like Javadoc so that it is immediately and easily accessible to everyone with Java experience. Javadoc is a standard documentation tool that is widely used because it is freely distributed with Sun's Java Development Kit (JDK) [18]. Javadoc is also designed to be extensible and *javaJAM* implements Javadoc classes. This first implementation of *javaJAM* is based on the following design goals:

1. Facilitate publication of Java documentation and source.
2. Maintain the standard Javadoc look and feel.
3. Implement linked "computer mediated communication" (CMC) relationships by establishing logical links between related documentation and source artifacts and

their annotations [7]. *javaJAM* implements the following types of artifacts: source overview, class overview, constructor, and method. These artifacts exist for both documentation and source.

4. Support efficient navigation between related documentation and source artifacts and their annotations.
5. Facilitate web-enabled asynchronous communications for the collection and organization of annotations related to documentation and source.
6. Provide for the moderation of annotations. A moderator acting on behalf of a development team can accept or reject annotations. Accepted annotations are expected to be implemented on the next or future release of the software.
7. Platform independence.
8. The initial implementation will be kept simple by not relying on a database, such as MS Access or MySQL for storage.

The scope of this first release of *javaJAM* does leave some important concepts and functionality outside of its focus. For example, an annotation's lifecycle is limited to a single version of the package. When a new version of the software is first published, it is a fresh start; there are no annotations. The previous version of the software can be visited, but none of the annotations automatically migrate to the new version. Also not in scope is the automation or facilitation of processes to assist the software development team with annotation review, establishment of consensus, acceptance/rejection of annotations and implementation of accepted annotations. For now it is left to the moderator to represent the team and to coordinate these activities.

1.4 Two Example Web-Enabled Application Reviews

Publishing on the web is easy. There are many HTML tools available, some free of charge. But creating a collaborative review environment takes coordination that is not possible simply by publishing documentation and source on the web and accepting comments by email. The following two examples illustrate the difference between simply publishing on the web vs. publishing with *javaJAM.* These examples demonstrate that *javaJAM* makes publishing applications, reviewing documentation and source, and collecting and organizing comments much simpler and more effective.

The following scenario is for the two examples below: Sophie is publishing the first iteration of her completed HeapSort project on the web. She hopes to publish her HeapSort application as Open Source Software , and she would like it to be as fast and as efficient as possible. She thinks it runs too slow after having tested it against 200 records. She decides to

publish it on the web so that programmers with more experience can review it and give her advice. She announces it in the newsgroup comp.lang.java.programmer.

1.4.1 Manually Publishing Documentation and Source

Assumptions:

Sophie has downloaded Sun's JDK and it is correctly installed.

- Sophie has HTML publishing skills.
- Sophie knows how to run Javadoc.
- Sophie has access to a web server for publishing her project.

Steps:

1. Sophie: completes coding and testing the HeapSort Java classes.
2. Sophie: decides to publish the project on the web so that others can review her project and provide improvement suggestions for the next version.
3. Sophie: runs Javadoc against her source to create the documentation.
4. Sophie: reads each Java source into an HTML editor and brackets the source with the <PRE> and </PRE> tags to preserve the source indentation and formatting.
5. Sophie: creates a HeapSort main web page with links to the main Javadoc generated documentation page and also to each of the HTML source pages.
6. Sophie: FTPs the HeapSort web page and related pages to her web site and updates her web site so that her home page links to her HeapSort page.
7. Sophie: posts a message about the help she needs (improving performance) with her HeapSort on the newsgroup comp.lang.java.programmer. In her posting she also includes the URL of the HeapSort web page.

. . . some time goes by . . .

8. Betty: sees Sophie's posting and visits Sophie's HeapSort page.
9. Betty: reads Sophie's HeapSort API and gets a sense of how the classes and methods are arranged.
10. Betty: decides to review the source related to the **heapSort** method in the **Heap** class because it appears to contain the core sorting logic.
11. Betty: navigates back to the HeapSort main page and selects the link to the **Heap** class.
12. Betty: scrolls down through the source until she locates the **heapSort** method and reviews the code.

13. Betty: returns to the comp.lang.java.programmer newsgroup and posts for Sophie suggestions on how to improve the **heapSort** method and also offers some general comments about the **Heap** class.
14. June: also sees Sophie's posting, visits Sophie's HeapSort page, and decides she would like to use the Heap class, but needs one additional feature.
15. June: posts to the comp.lang.java.programmer newsgroup a request for a method that will return the next 'n' elements starting from a specific element number.

. . . some time goes by . . .

16. Sophie: opens her email, finds Betty's and June's comments, reads them, and likes them. Sophie notes that June's comments are identical to one of Betty's.
17. Sophie: includes Betty's suggestions in the new version of HeapSort that she is working on.

. . . some time goes by . . .

18. Betty: wonders what happened to her suggestions and emails Sophie.
19. Sophie: looks through her old email and locates Betty's original suggestions.
20. Sophie: opens the Java source in her editor and determines that Betty's suggestions were all implemented.
21. Sophie: emails Betty her findings.

. . . some time goes by . . .

22. June: wonders what happened to her suggestions and emails Sophie.
23. Sophie: looks through her old email and locates June's original suggestions.
24. Sophie: opens the Java source in her editor and determines that June's suggestion was implemented.
25. Sophie: emails June her findings.

If Sophie's HeapSort becomes very popular she would have a hard time keeping up with the correspondence. In the scenario above, Betty and June will probably not know about each other's postings. They and others will be submitting redundant requests in some cases. Since Betty and June are probably unaware of each other's comments, the two of them are not in any sense collaborating. For them and any other contributors, the collaborative relationship is a simple one-to-one relationship between each contributor and Sophie. If HeapSort draws much

attention, Sophie is going to spend a lot of time managing these one-to-one relationships. *javaJAM* is designed to simplify the mechanics of collaboration, and to support a complex many-to-many relationship between contributors.

1.4.2 *javaJAM* Publishing Documentation and Source

Assumptions:

- Sophie has access to a *javaJAM* server, but has never accessed it.
- Sophie has access to WinZip or similar utility.

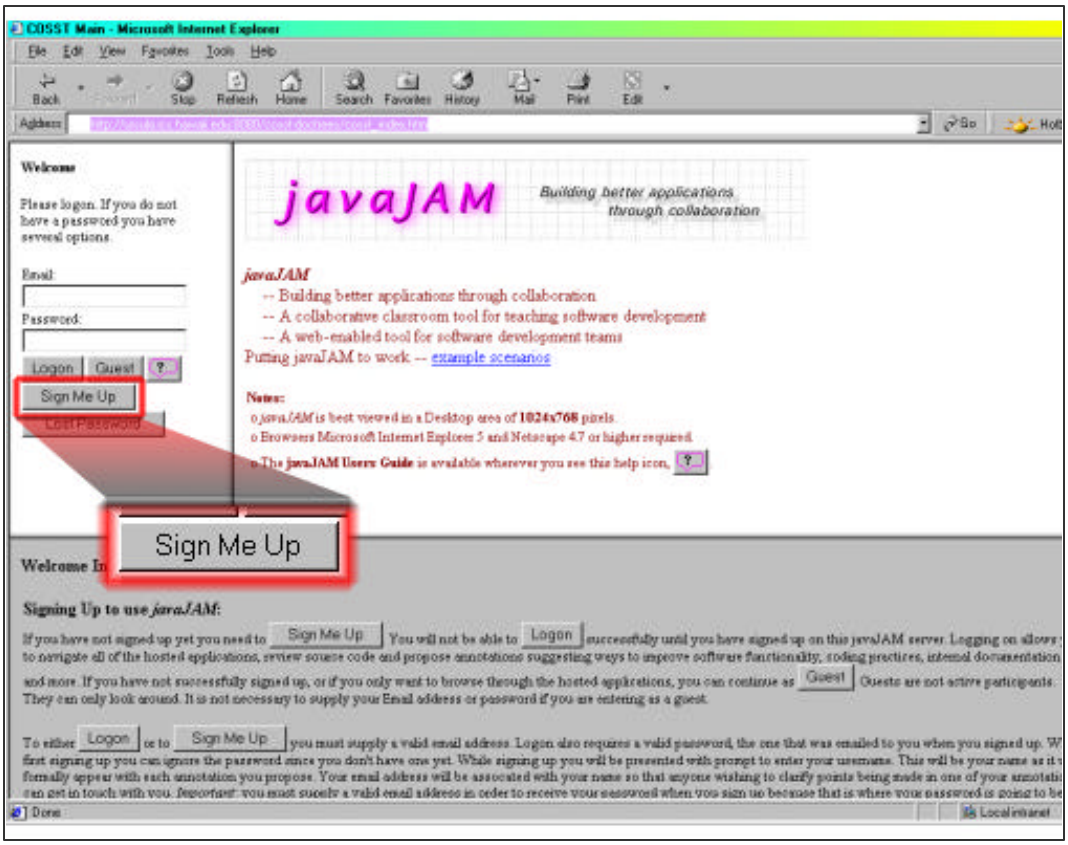
Steps:

1. Sophie: completes coding and testing the HeapSort Java classes.
2. Sophie: decides to publish the project on the web so that others can review her project and provide improvement suggestions for the next version.
3. Sophie: signs up on the *javaJAM* server and supplies her email address and her full name.
4. Sophie: receives an email from the server that provides her *javaJAM* password.
5. Sophie: zips her source and emails it as an attachment to the *javaJAM* server administrator for publishing on the *javaJAM* server.
6. Sophie: announces HeapSort on the newsgroup comp.lang.java.programmer. Her posting includes a request for help to make HeapSort run faster. In her posting she includes the *javaJAM* URL where her HeapSort is hosted so that the newsgroup users can review HeapSort and post suggestions.

   . . . some time goes by . . .

7. Betty: sees Sophie's newsgroup posting and visits the *javaJAM* server.

Figure 1, Initial *javaJAM* Screen


COSST Main - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Stop Refresh Home Search Favorites History Mail Print Edit
Address
Welcome
Please logon. If you do not have a password you have several options.
Email:
Password:
Logon Guest ?
Sign Me Up
Lost Password
javaJAM
Building better applications through collaboration
javaJAM
-- Building better applications through collaboration
-- A collaborative classroom tool for teaching software development
-- A web-enabled tool for software development teams
Putting javaJAM to work -- example scenarios
Notes:
o javaJAM is best viewed in a Desktop area of 1024x768 pixels
o Browsers Microsoft Internet Explorer 5 and Netscape 4.7 or higher required.
o The javaJAM Users Guide is available wherever you see this help icon, ?
Sign Me Up
Welcome In
Signing Up to use javaJAM:
If you have not signed up yet you need to Sign Me Up You will not be able to Logon successfully until you have signed up on this javaJAM server. Logging on allows
to navigate all of the hosted applications, review source code and propose annotations suggesting ways to improve software functionality, coding practices, internal documentation
and more. If you have not successfully signed up, or if you only want to browse through the hosted applications, you can continue as Guest Guests are not active participants.
They can only look around. It is not necessary to supply your Email address or password if you are entering as a guest.
To either Logon or to Sign Me Up you must supply a valid email address. Logon also requires a valid password, the one that was emailed to you when you signed up. W
first signing up you can ignore the password since you don't have one yet. While signing up you will be presented with prompt to enter your username. This will be your name as it
formally appear with each annotation you propose. Your email address will be associated with your name so that anyone wishing to clarify points being made in one of your annotati
can get in touch with you. Important: you must supply a valid email address in order to receive your password when you sign up because that is where your password is going to be
Done
Local intranet

8. Betty: signs up on the *javaJAM* server so that she may post annotations.

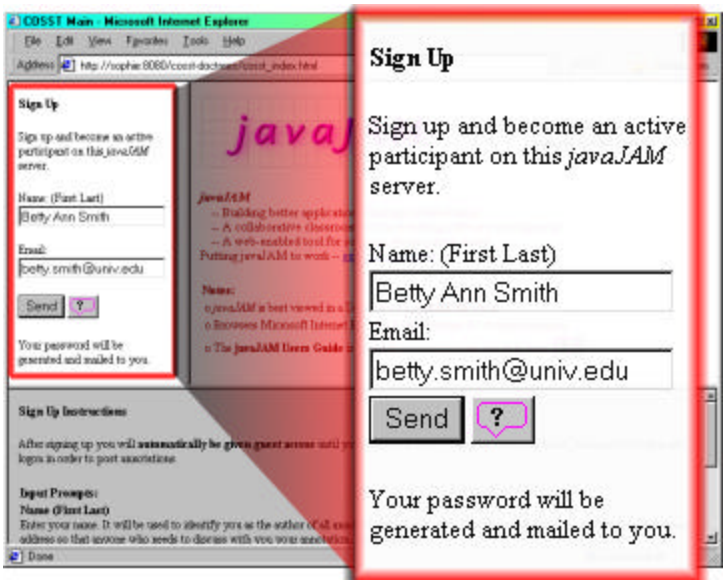


Figure 2, Sign Up Screen

9. Betty: receives an email message from the javaJAM server with her *javaJAM* password (see Appendix C).
10. Betty: logs on to the javaJAM server.

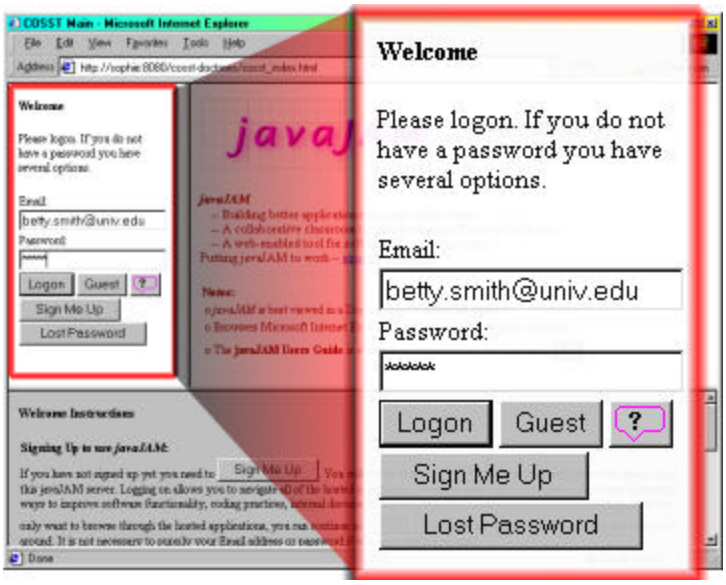


Figure 3, Welcome Screen

11. Betty: selects Sophie's HeapSort.

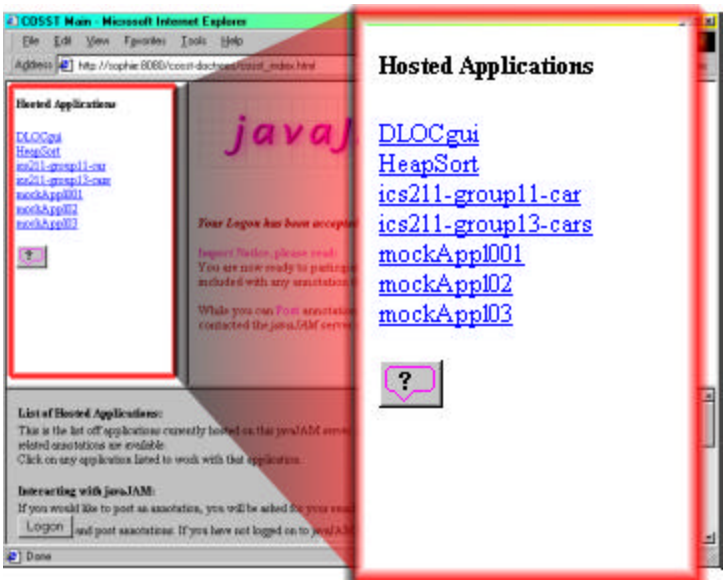


Figure 4, Hosted Applications Screen

12. Betty: reads Sophie's HeapSort API and gets a sense of how the classes and methods are arranged.
13. Betty: while at the top of the documentation for class **Heap** she clicks the Annotate button and posts her general comments about the class in the popup dialog box that appears. One of the comments suggests a new method for returning elements starting from any position.

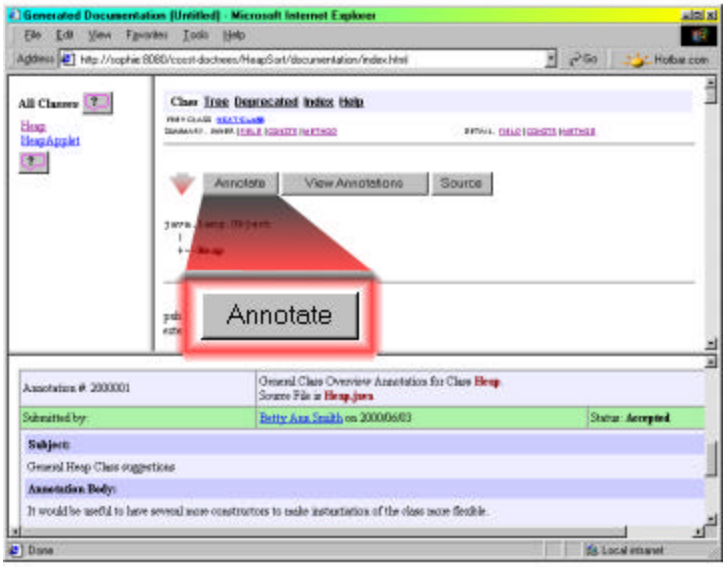


Figure 5, Annotate Button

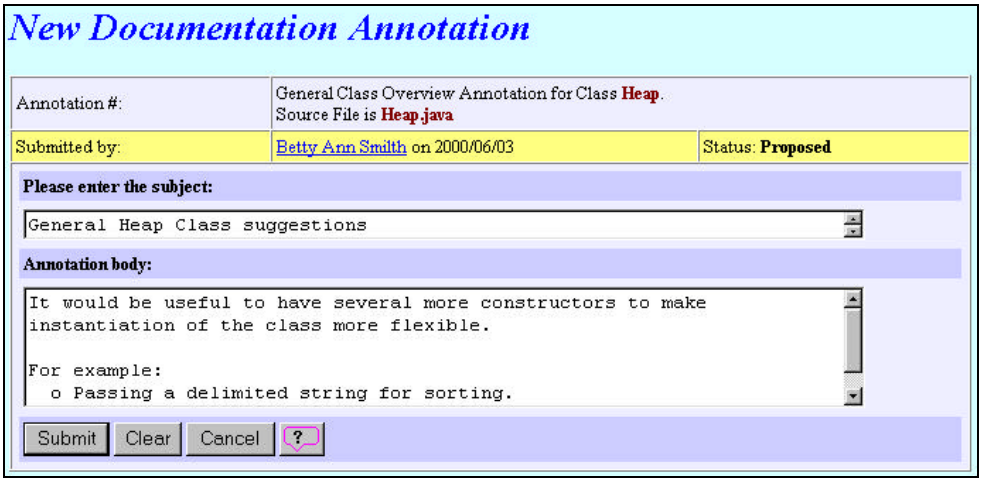


Figure 6, New Annotation Dialog

14. Betty: when done creating her class overview suggestions she will press the Submit button so that her suggestions are posted. A confirmation screen will appear.

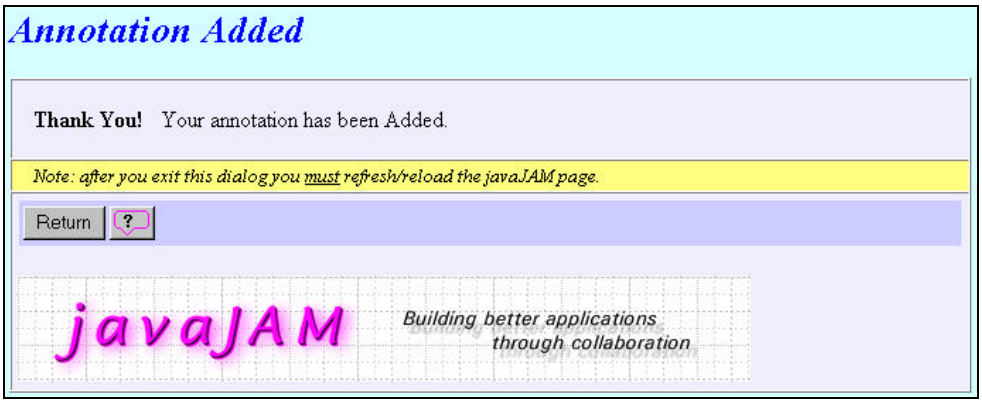


Figure 7, Add Annotation Confirmation Dialog

15. Betty: while reading the documentation for class **Heap**, method **heapSort** she clicks the Source button to review the related source. Betty: after reviewing the related source she clicks the annotate button and posts suggestions on how to improve the **heapSort** method.

    . . . some time goes by . . .

16. June: sees Sophie's newsgroup posting, visits the *javaJAM* server, signs up, and becomes an active participant so that she can look at HeapSort.
17. June: sees Betty's suggestion for a method that returns elements starting from any position and uses the "add related annotation" feature to state that she too is interested in this method.

    . . . some time goes by . . .

18. Sophie: logs on to the *javaJAM* server.
19. Sophie: from the list of available projects chooses HeapSort.
20. Sophie: reads through the annotations and finds Betty's two annotations with status **Proposed**.

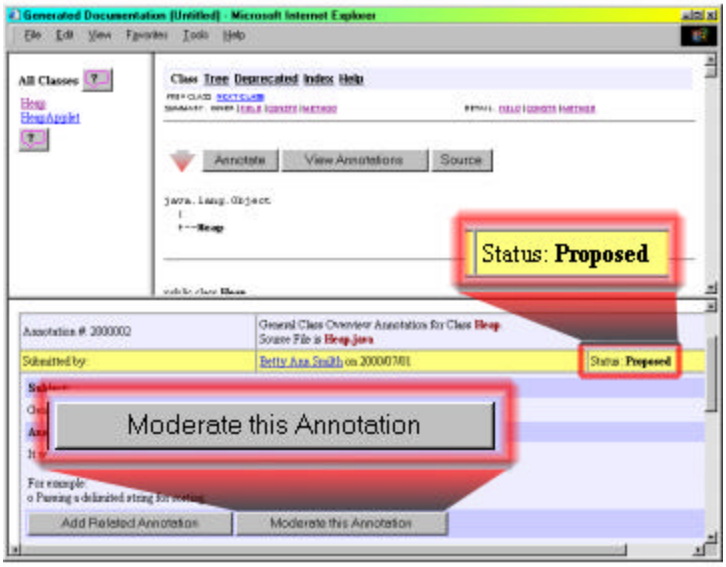


Figure 8, Proposed Annotation Status

21. Sophie: for each of the two annotations she presses the Moderate this Annotation button, reviews the comment, and presses the Accept button to change the annotation status to **Accepted**.

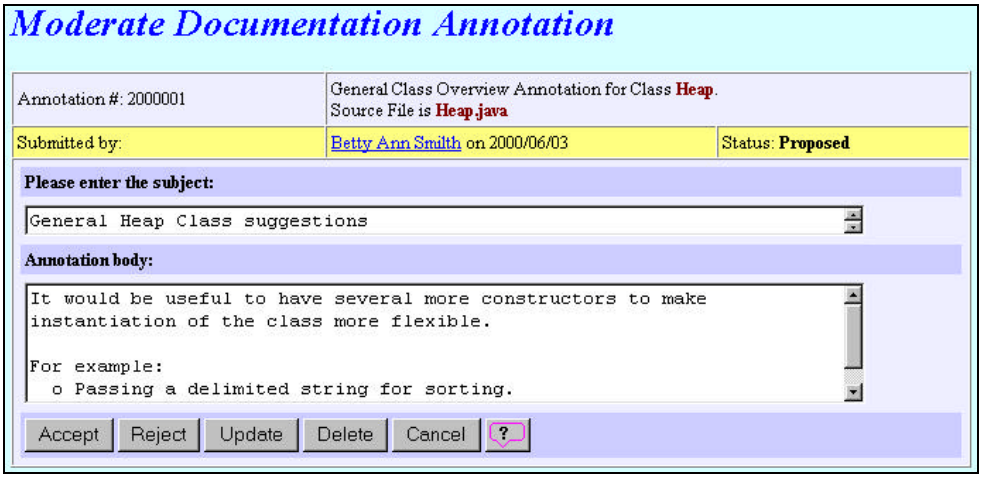


Figure 9, Accept Annotation Dialog

22. Sophie: includes Betty's suggestions in the new version of HeapSort that she is working on.
23. Sophie: posts the new version of HeapSort with Betty's suggestions.

. . . some time goes by . . .

24. Betty: wonders what happened to her suggestions and logs on to the *javaJAM* server.
25. Betty: from the list of available projects chooses HeapSort.
26. Betty: looks through the annotations and locates her original suggestions. She notes that they each have a status of **accepted**.

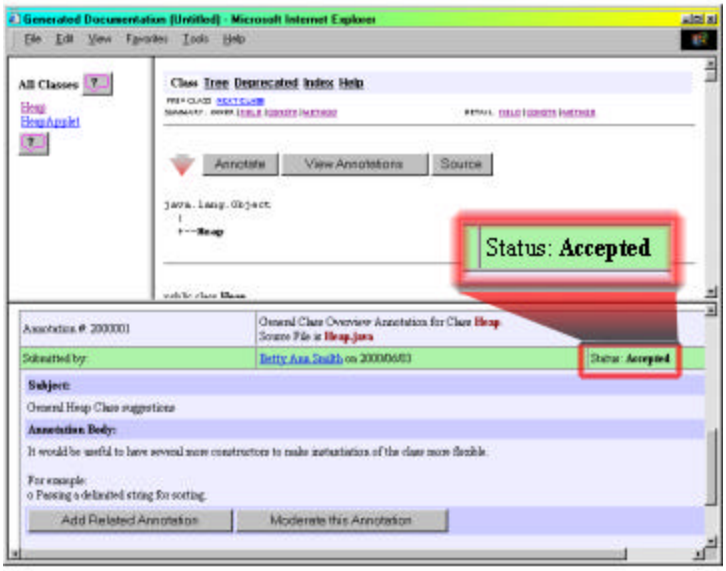


Figure 10, Accepted Annotation Status

. . . some time goes by . . .

27. June: wonders what happened to Betty's suggestion and logs on to the *javaJAM* server.
28. June: from the list of available projects chooses HeapSort.
29. June: looks through the annotations and locates Betty's original suggestion. She also notes that the suggestion for the new method has a status of **accepted**.

With *javaJAM* Sophie has accomplished much more than a simple static web page could offer. Her HeapSort posting on the *javaJAM* server is the center of activity and establishes a many-to-many relationship between the various contributors of comments and suggestions. This presents the opportunity for contributors to asynchronously interact with each other. Where

before there was the potential of many redundant suggestions, now annotations can result in an evolution of some of the suggestions because everyone can see all annotations.

1.5 Thesis Statement

The research for this thesis has demonstrated that:

1. Software development can be improved by using collaboration tools.
2. That the Javadoc tool can play an important role facilitating collaborative software development.
3. The *javaJAM* study group appreciated being able to easily publish their projects on the web.
4. The *javaJAM* study group appreciated being able to easily navigate between project source and documentation while reviewing their projects.

The statement "Software development can be improved by using collaboration tools" addresses the fact that a collaborative tool like *javaJAM* is helpful in some circumstances, but not others. In the classroom setting for example, a project team may have no interest in a project once it is completed if it is not a building block for a future assignment. *javaJAM* provides the strongest support for collecting reviews at the version level in preparation for the next version of a software package.

The statement "That the Javadoc tool can play an important role facilitating collaborative software development" discusses the importance of a documentation generation tool like Javadoc. Documentation is an important form of communication and is very beneficial to collaborative efforts. Languages like C and Visual Basic that do not have a standard documentation generation tool are at a disadvantage when it comes to creating a web-enabled collaborative environment. Javadoc gives the Java language a distinct advantage in collaborative software development efforts and makes Java a richer language.

The statement "The *javaJAM* study group appreciated being able to easily publish their projects on the web" relates the experiences of students in the study discovering and appreciating how easy it was to find their projects—documentation and source—published on the web and available for their team to review. It must be noted that *javaJAM* currently requires a site administrator, me in this case, to do the actual publishing of the students' projects. While some of the work of publishing can be pushed back to the students, the site administrator would still need to determine who would have privileges to moderate which packages.

The statement "The *javaJAM* study group appreciated being able to easily navigate between project source and documentation while reviewing their projects" relates the experiences of students in the study group navigating through their projects and creating annotations. The students with Javadoc experience—one of the two classes in the study—found the interface

intuitive. This experience is made possible because javaJAM creates explicit relationships between related sections of documentation and source and the annotations that are posted by reviewers.

It also should be noted that even those students without Javadoc experience who write Java code without considering Javadoc, still found the generated documentation for their programs to be useful. Javadoc, without the students' annotations, is still very capable of analyzing package layout and program source and generating meaningful documentation.

1.6 Overview of this Document

The rest of this document explains *javaJAM* in detail and includes a study that was conducted to evaluate *javaJAM*. Chapter 2 describes related work such as the collaboration approach in Open Source environment and the need for tools like *javaJAM* to facilitate collaboration. Chapter 3 discusses the functionality of *javaJAM* and explains how it could facilitate collaborative software development. Chapter 4 details the implementation of *javaJAM*. In Chapter 5 the study is presented along with the results. In Chapter 6 *javaJAM* is compared to other collaboration tools. Chapter 6 discusses the contributions of this research and includes some general observations. Chapter 7 lists possible future enhancements and directions.

2. Related Work

The only way to meet the challenges of producing much of the software for the 21st century will be to collaborate. Assembling a team of software engineers, business analysts, and others will make it possible to bring a broad range of skills and perspectives to a development project, and will make it possible to develop portions of a project in parallel in order to shorten the overall development cycle while maintaining quality and effectiveness.

There are already a number of different tools to assist with team collaboration. There are project-planning tools to sequence and assign tasks. There are librarian tools to control access to modules and module versions under development. There are many tools for allowing geographically dispersed members of a team to communicate face-to-face and to share information and even desktops, but there are currently no tools to specifically assist with code and documentation review and the collection of comments.

2.1 Open Source Initiative

The Open Source Initiative (OSI) [20] offers distinct advantages for software development. With the rapid growth of the Internet collaboration between software engineers and others is now possible on a scale not previously imagined. There are many systems and much research related to the Open Source initiative; for example Linux, the Apache Web Server [2], and the next release of the Netscape browser. These packages are fully developed and supported by the Open Source initiative. The Open Source initiative also offers OSI Certification [21]. The OSI Certified mark may be used on software distributions only after an Open Source license has been obtained for it from the Open Source Organization. In order for the license to be obtained, the software must comply with the Open Source definition [6]. This definition defines Open Source software as having the following priorities:

Free Redistribution - there can be no license fees and no restrictions on using the software as a component of software that is distributed or sold.

Source Code - source code must be included in the distribution and no restrictions on distribution of binaries.

Derived Works - derived works must be permitted and allowed to be distributed under the same terms.

Integrity of The Author's Source Code - distribution of modified version may be prohibited if patch files for the source are permitted.

No Discrimination Against Persons or Groups - license must not discriminate against persons of groups.

No Discrimination Against Fields of Endeavor - license must not restrict any field of endeavor; genetic research for example.

Distribution of License - no additional licensing should be required for any parties.

License Must Not Be Specific to a Product - the license should not be conditional on the software being distributed with other software.

License Must Not Contaminate Other Software - license should not place restrictions on the use of other software.

Software that is distributed as Open Source software must be identified as being placed in the public domain and must contain the following notice:

"This software is OSI Certified Open Source Software.
OSI Certified is a certification mark of the Open Source Initiative."

javaJAM was designed in the spirit of the Open Source initiative and by default supports most of the above priorities and interferes with none of them. Packages hosted on a *javaJAM* server are available to everyone via the World Wide Web. Where the Open Source initiative's intention is to ensure that software is distributed freely with few if any limitations, *javaJAM*'s design goes one step further by creating an open environment for collecting comments and suggestions from anyone that is willing to take the time to provide them.

The Open Source initiative goes beyond the sharing of software by dedicated programmers that are working for the pleasure of making a good package or for establishing recognition. The business community is also becoming interested in the initiative because it is a good way to gain support for open standards. Businesses invest in developing software to support actual standards so that businesses can cooperate easier. Businesses also benefit from the initiative by increased security. Open Source software is exposed to extreme scrutiny and the problems found are reported and fixed, which is unlike proprietary software where problems can remain unreported and thus be exploited by the wrong person. An additional advantage to the business community is that Open Source software is "peer review" software. Mature, Open Source software is as reliable as software can get. Proprietary software does not have this advantage.

The best example of the success of the Open Source initiative is the Internet's infrastructure. DNS, sendmail, various TCP/IP stacks and utilities, and scripting languages such as Perl demonstrate that Open Source Software already plays a valuable role. In his book "The

Cathedral and the Bazaar," Eric S. Raymond compares traditional (Cathedral) software development methodologies to the methodology used for Open Source projects (Bazaar) such as Linux and Fetchmail [9].

> "Treating your users as co-developers is your least-hassle route to rapid code improvement and effective debugging."
>
> "Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone."
>
> "If you treat your beta-testers as if they're your most valuable resource, they will respond by becoming your most valuable resource."
>
> "The next best thing to having good ideas is recognizing good ideas from your users. Sometimes the latter is better."
>
> "Provided the development coordinator has a medium at least as good as the Internet, and knows how to lead without coercion, many heads are inevitably better than one."

javaJAM constructively supports each of these points. *javaJAM* gives users a voice and provides them with feedback so that they know their participation has impact. *javaJAM* makes a package available to a large base.

2.2 CSRS

CSRS is a computer-supported software review system (CSRS) that enables declarative definition of review processes and provides instrumented facilities for gathering and analyzing review data [8]. CSRS provides for formal technical review (FTR), a cornerstone of software quality assurance, which is typically under-utilized or inefficiently applied because it is generally a labor-intensive, manual process. CSRS has been implemented for Unix systems with an Emacs front-end. *javaJAM* represents a tradeoff between accessibility and formality since the intention is to appeal to the same general audience using Javadoc.

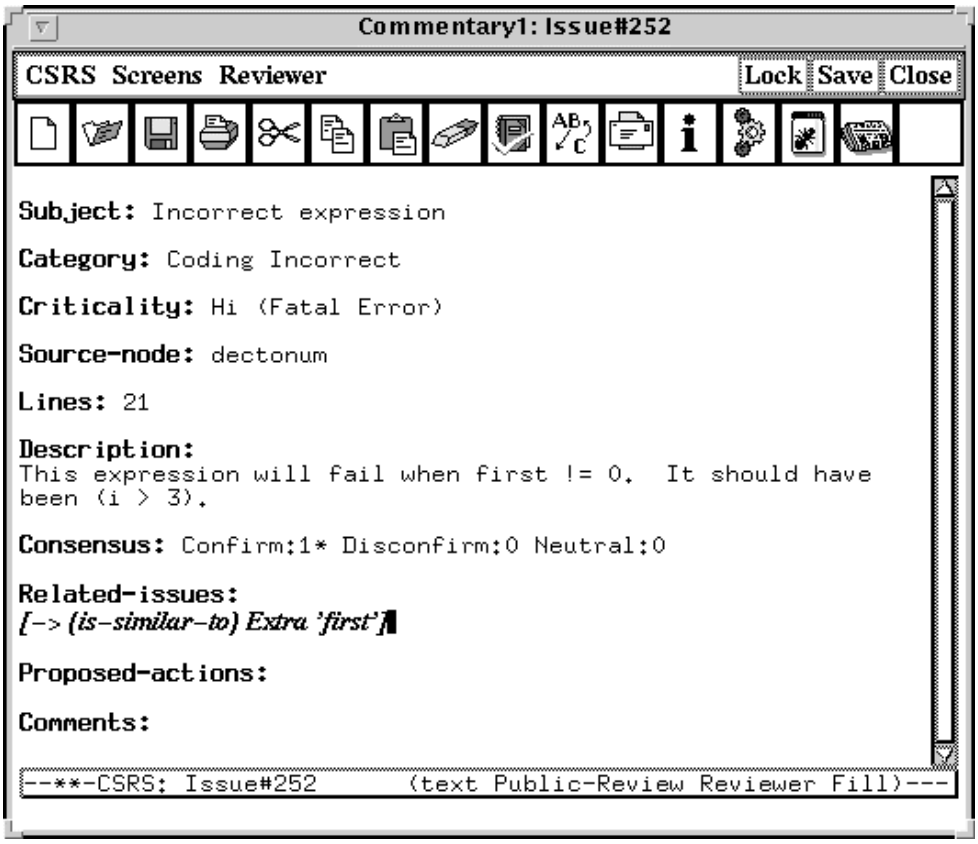


Figure 11, CSRS Sample Commentary

2.3 Giant Java Tree

The Giant Java Tree [10] is an Open Source project. This project creates a Java source tree consisting entirely of Open Source and permits browsing of the source. You can sign up as a developer and have access to participants' source code. It is free in a sense that you can share your source or use someone else's.

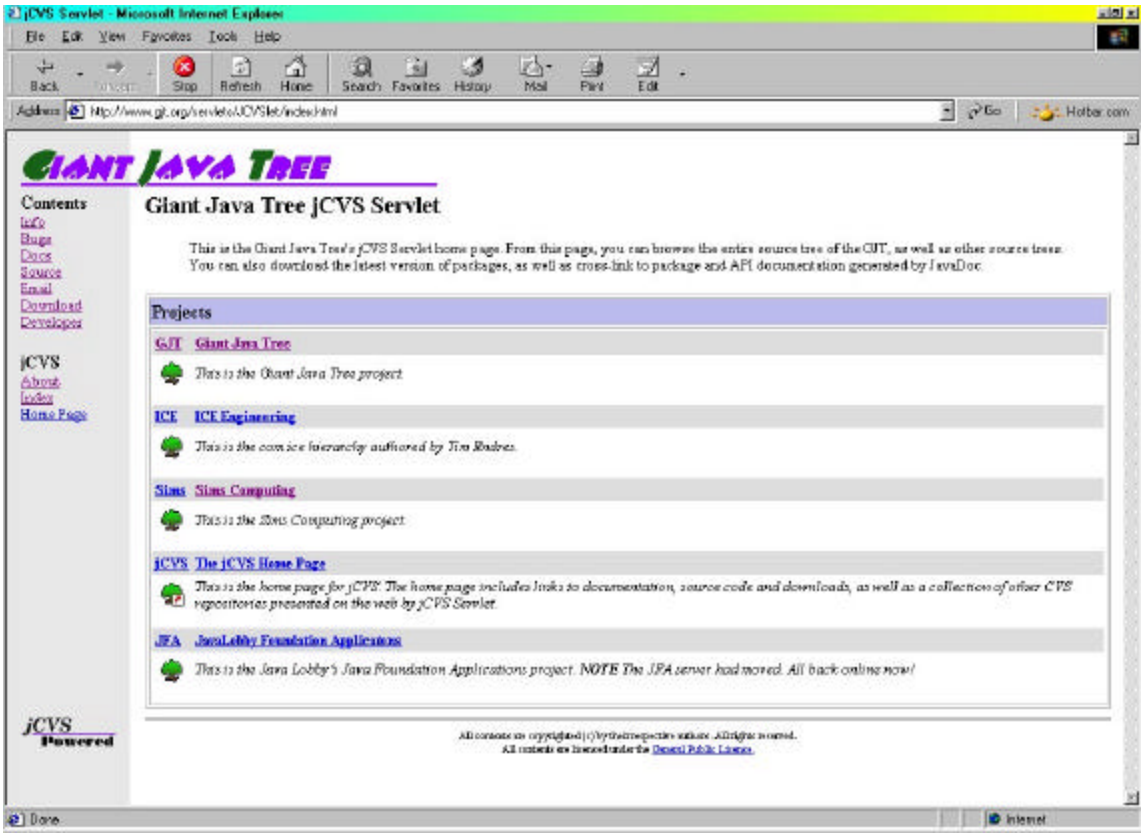


Figure 12, Giant Java Tree Project Selection Screen

Giant Java Tree provides automatic Javadoc generation, similar to *javaJAM*. For software engineers, Giant Java Tree provides an extensive source of Java examples and codes by providing a sharing environment that is strictly a code repository. The *javaJAM* environment could be used as a code repository, but it goes one step farther by collecting comments and suggestions and storing them along with their relationships. This allows participants to not only view others' source code, but also to communicate ideas towards improving the quality of the documentation and source. Where Giant Java Tree functions as a catalog, *javaJAM* functions as a catalog based collaboration tool.

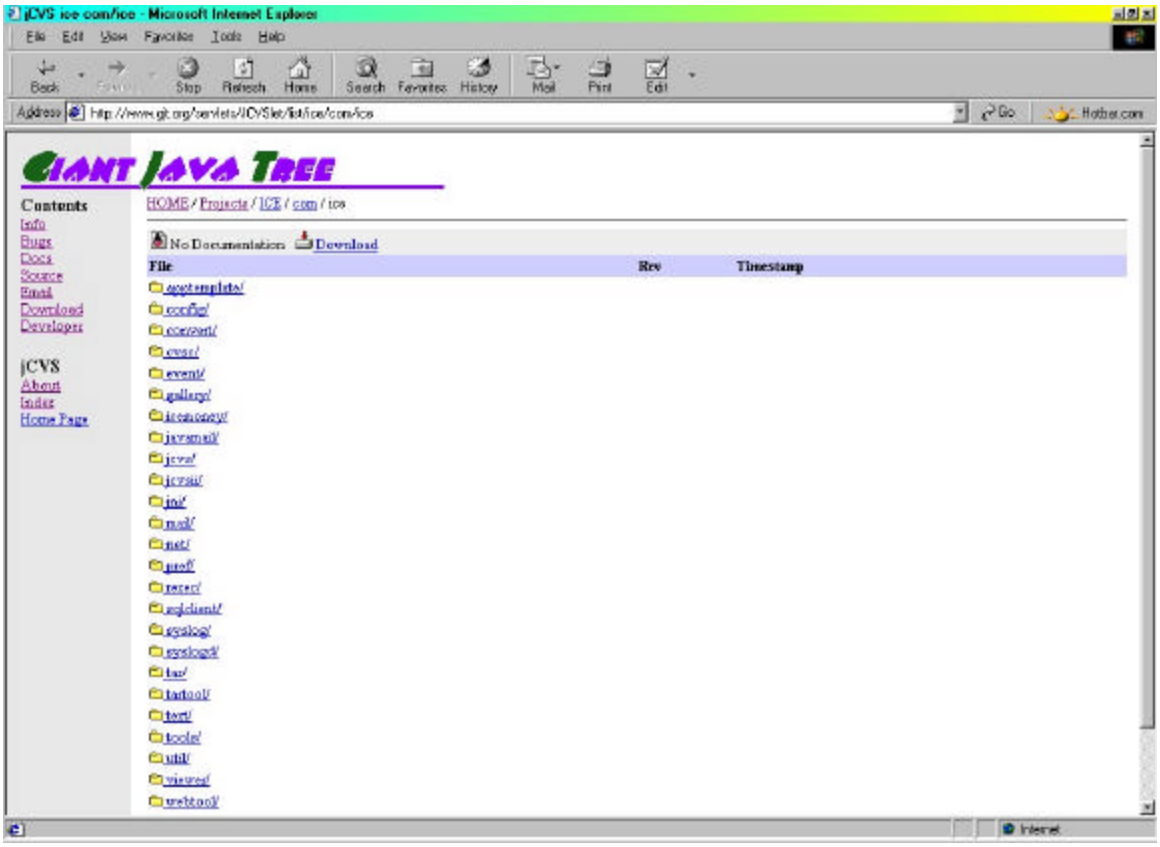


Figure 13, Giant Java Tree Sample Project Screen

2.4 SDM

SDM (Software Development Manager) [3] is a planned subsystem for supporting Open-Source Software via the online community. This subsystem will be an extension to the core ArsDigita Community System. The system will support those who use, extend, and fix bugs in a collaboratively developed and maintained software environment.

The SDM design contains the following core tables:

- **modules**: each row represents a collection of related source code; columns include module name, owner, current version, description, etc.
- **module_relationships:** each row records which modules interact significantly and who owns the interface between them.
- **module_releases**: each row represents a named version of a module, e.g., "3.7"; columns include release_date (null until done), anticipated_release_date, release_name, (just text; SDM doesn't care if 3.7 comes after 4.0) manager (person responsible for this particular release; typically the module owner).

- **module_log:** a linear log of dated information about progress on a given module.
- **bugs_and_features:** each row is one report from a user of a bug or a requested new feature; columns include who has been assigned to it, what priority it has been assigned, when it is expected to be fixed/added, whether it has in fact been fixed/added and, if so, in which release.
- **bug_release_map:** there will be a row if a bug is present in a particular release a module (covers the case where a bug is discovered in version 1.4 and we need the SDM to inform users that it is also present in 1.2 and 1.3).
- **tasks:** something to be done by a participant in SDM, typically a programmer. Testing is an example of an appropriate entry for the tasks table.
- **user_interest_map:** records the fact that a user of the SDM is interested in monitoring the progress of a bug, feature, or task.

The SDM design is strongly oriented to the life-cycle of the bug. It tracks the bug across software versions and identifies who is tasked to fix the bug. Requests for new features are treated much the same way that bugs are treated. While SDM has features that could be useful to *javaJAM*, problem assignment and tracking, it does not address the same situations as *javaJAM*. *javaJAM* is designed to collect comments and suggestions from anyone and automatically creates relationships between the information collected and the areas of source and documentation they relate to. Extending javaJAM functionality to include bug and new feature tracking in a way that makes the information available just as the source and annotations are now openly available would be beneficial.

The original date for Phase 1 of SDM was March 15, 1999. Development of Phase 2 was to be completed by August 15, 1999. There are currently no references to indicate that it has been implemented or is even still being considered for development other than the existence of the web site reference.

2.5 Tango Interactive

Syracuse University and WebWisdom.com completed Tango version 1 in April of 1998 [19]. The emphasis was to create a fully interactive collaborative tool. Tango was based on Microsoft's NetMeeting, but with a richer set of features:

- Supports any programming language
- Provides for live, interactive collaboration

- Supports live, application sharing via live desk-tops
- Creates transcriptions of chat sessions
- Supports video conferencing

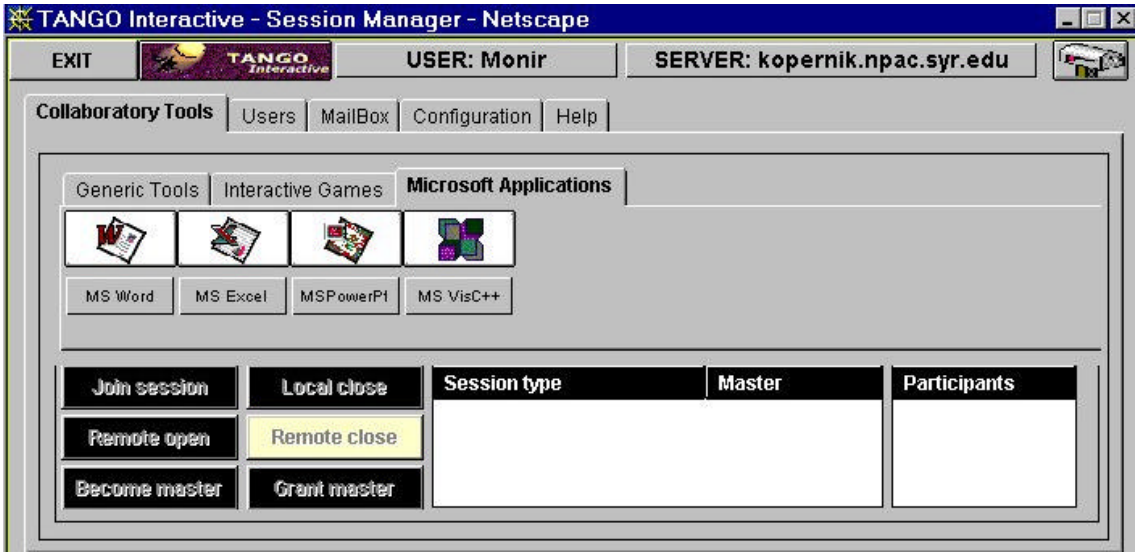


Figure 14, Tango Application Sharing Selection Screen

As an interactive collaboration tool Tango is very strong though having a fast Internet connection is an issue. What Tango lacks is a coordinated means of collecting and maintaining information that results from collaboration. The ability to create transcripts from chat sessions does not and probably is not meant to take care of this.

javaJAM has no live person-to-person interactive functionality at all. Adding this type of interactive functionality may not be beneficial in *javaJAM* since interactive functionality would not easily provide a mechanism to ensure that comments and suggestions are collected and the proper relations to documentation or source are established. To solve this problem would take *javaJAM* into an entirely new level of functionality. *javaJAM* does have an advantage in being faster, especially over dialup connections to the Internet than Tango because *javaJAM* works by sending only HTML files that are compatible with version 3 browsers. *javaJAM* is also easier to access; unlike Tango, no plug-ins are required for *javaJAM*.

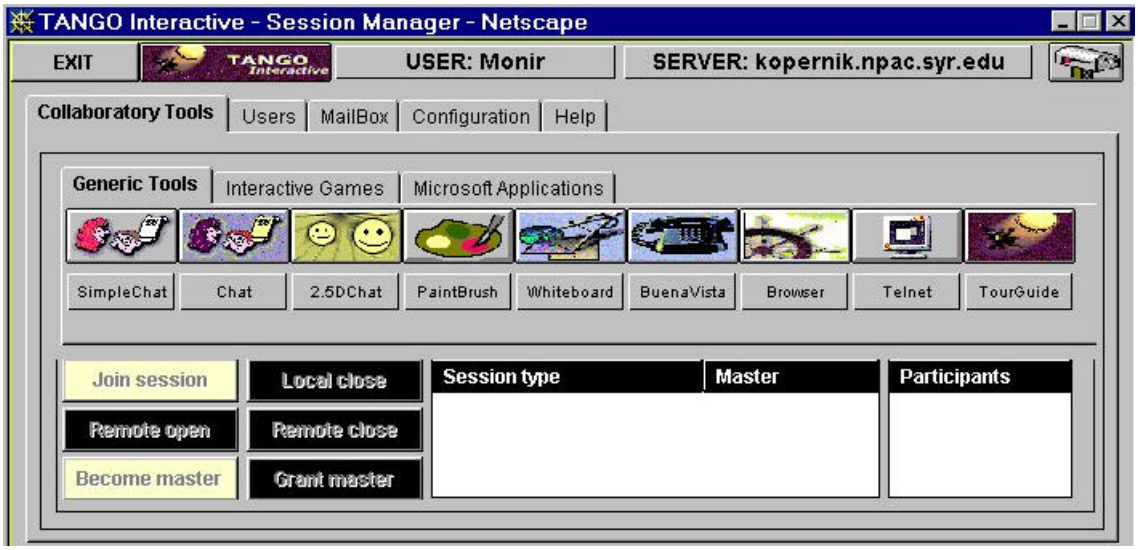


Figure 15, Tango Generic Tools Selection Screen

While Tango version 1.3 is still available as freeware from Syracuse University, the company WebWisdom.com sells Tango commercially as an interactive tool for e-businesses. It supports communication, collaboration, training, and knowledge management. The intent of Tango is to extend the physical office across Internet space. It is not at all about managing and distributing Open Source software projects

3. A New Collaboration Tool

javaJAM represents a new category of collaboration tools. It was originally conceived as an Internet enabled tool to build a community around a software development project for the exchanging of ideas. The original concept was proposed by Dr. Philip Johnson of the University of Hawaii at Manoa and evolved from there as I took the idea and worked to develop a tool which could be used by many different groups in an educational or professional setting [16].

The design goal for *javaJAM* was to assist with code and documentation review and the collection of comments. *javaJAM* was designed to provide a means to address this need for collaboration over the Internet between groups of software engineers or student programmers who work in a team to develop software.

javaJAM takes Sun's Javadoc standard and extends it to make it interactive. More than just providing for the Javadoc publication, *javaJAM* goes several steps farther to include the sharing of Java source and documentation in an Open Source environment and to also include the collection of annotations by reviewers.

javaJAM relies on HTML files to store the annotations it accumulates. *javaJAM* also partially implements the concept of HyperCode [13] and generates HTML versions of source files to facilitate code review. HyperCode is source code rendered in HTML with each line of source treated as a hyperlink and artifact. *javaJAM* manages relationships that make it possible to hyperlink specific artifacts (sections of source, documentation and comments) to each other for quick navigation. These relationships are formed by creating hyperlinks and bookmarks and embedding them throughout the documentation, source code, and comments. A scaleable alternative to relying on text files would be to implement *javaJAM* storage of annotations in a relational database using JDBC. This would greatly improve performance as well.

3.1 Requirements

The Open Source Software initiative brings with it new methodologies for software development. The Apache Software Foundation and sourceXchange [17] are two of a growing number of examples of new approaches to software development and collaboration. Both of these initiatives involve teams that work together to produce software without having to meet physically. The Apache web server is one of the most widely installed and widely used servers in the world. Existing and new tools will be developed to help these as well as more traditional team collaborate more effectively and more efficiently.

3.1.1 Documentation and Source Review

javaJAM has been designed to provide an application development team with a tool that makes it easy for members to review software documentation and source and to annotate specific portions of the documentation and source. Team members, no matter where they work or what hours they work, can contribute these annotations as long as they have Internet access.

A team that intends to use *javaJAM* must standardize on Sun's Javadoc for documentation. *javaJAM* extends Javadoc and therefore assumes that both documentation and source are to be made available for review. For teams that have already standardized on Javadoc, there is a distinct advantage. The *javaJAM* interface maintains nearly the same look and feel as the Javadoc documentation. This makes introducing and learning to use *javaJAM* in the collaborative environment much easier to accomplish. Access to *javaJAM* is browser-based to ensure that there are no barriers to access.

3.2 Functionality

javaJAM's primary function is to accept annotations and store a relationship between the annotation and the section of documentation or source to which it belongs. This is accomplished through the *javaJAM* extensions to the standard Javadoc. There are several major extensions to Javadoc. The first is the inclusion of annotation buttons throughout the standard Javadoc generated documentation. The second major extension to Javadoc is the generation of Java source rendered in HTML with annotation buttons throughout the source.

3.2.1 Documentation Annotation

Documentation annotation buttons occur throughout the Javadoc generated documentation. Javadoc generates one HTML file per class (and inner class). The structure of each Javadoc generated class documentation is as follows:

1. Package Navigation Bar
2. Package Tree Diagram
3. Class Documentation
4. Field Summary
5. Constructor Summary
6. Methods Summary
7. Field Detail
8. Constructor Detail
9. Methods Detail
10. Package Navigation Bar

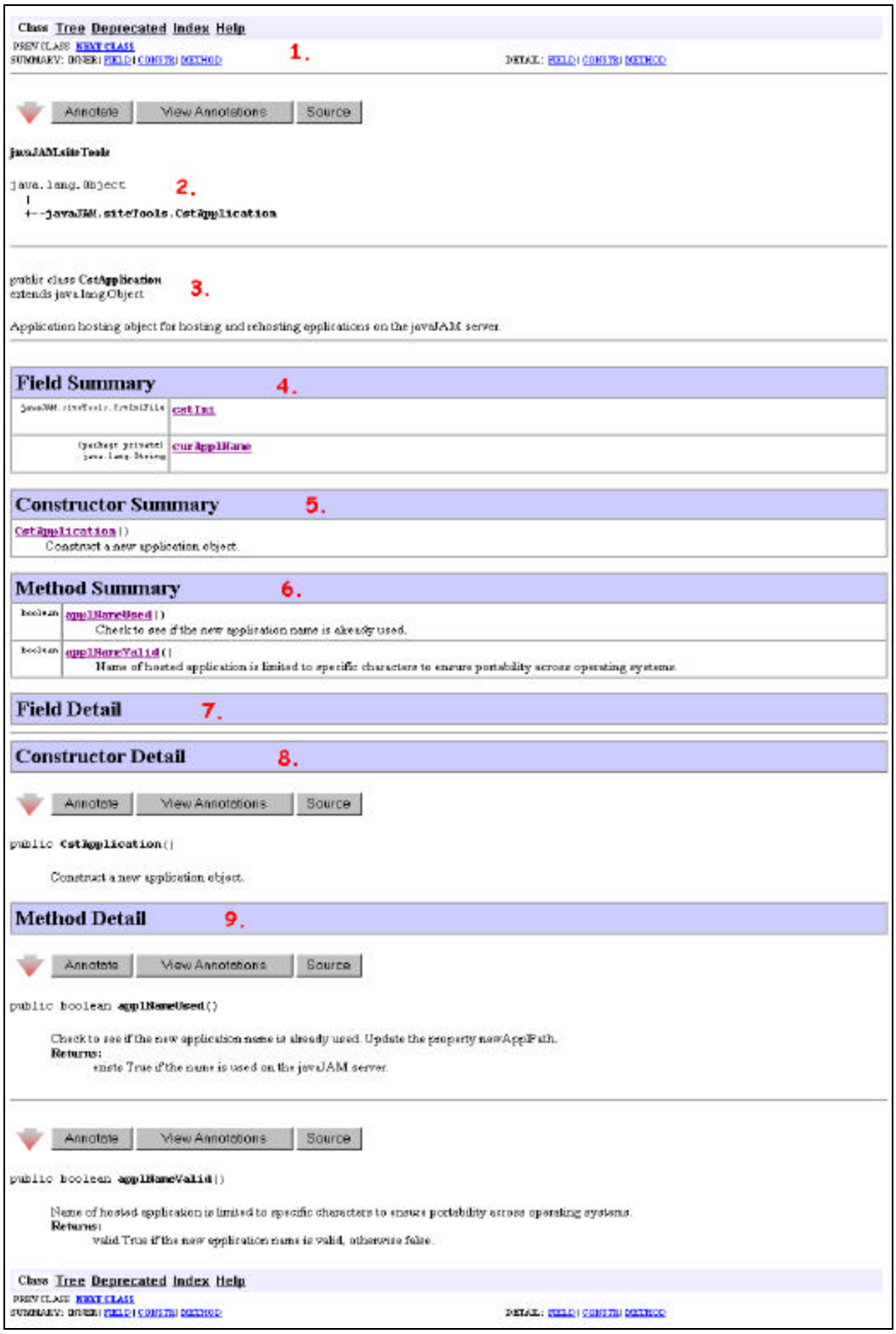


Figure 16, Sample Javadoc Class Structure

javaJAM imposes on this layout annotation buttons. These buttons appear as a triad in a single row and are imposed at the following locations:

- Just before the Package Navigation Bar (java.lang.object) above: the annotation button in this location provides for the creation of a "class overview" annotation.
- For each Constructor Detail above: there will be an annotation button leading each constructor. All annotations created here will be related to that constructor.
- For each Method Detail above: there will be an annotation button leading each method and method overload. All annotations created here will be related to that method or method overload.

These buttons are not only explicitly related to the annotations that they are used to create, but also to the source code that is specifically relevant to the section of documentation and related annotations. The "Source" button is an anchored hyperlink to the section of source related to the documentation. This makes it very easy to review a section of documentation and to quickly access related annotations and source.

3.2.2 Source Annotation

Javadoc has been extended to also render source in HTML. Source annotation buttons occur throughout the HTML rendered source. Javadoc generates one HTML file per class (and inner class). *javaJAM* does not change the layout of the source, but does insert annotation buttons at the following locations.

1. At the top of the source to provide a place for source-overview annotations.
2. Just before the class statement to provide a place for class-overview annotations.
3. Before each constructor definition.
4. Before each method and each method overload definition.

Similar to the documentation buttons, source buttons are not only explicitly related to the annotations that they are used to create, but also to the documentation that is specifically relevant to the section of source and related annotations. The "Doc's" button is a hyperlink to a bookmark in the section of documentation related to the source. Moving between source and related annotations and then on to related documentation and annotations is quick.

3.2.3 Line Numbering

javaJAM optionally inserts line numbers before each line. The intent of this is to make it easier to discuss source code by providing additional information for referencing. Line numbering can be enabled or disabled. Line numbering is added when a package is first hosted if the line number option is turned on (the default). Turning off the line numbering feature after a package has been hosted has no effect on the already hosted packages.

```
00085     protected void generateContents(Character unicode, List memberlist) {
00086         anchor("_" + unicode + "_");
00087         h2();
00088         bold(unicode.toString());
00089         h2End();
00090         dl();
00091         for (int i = 0; i < memberlist.size(); i++) {
00092             Doc element = (Doc)memberlist.get(i);
00093             if (element instanceof MemberDoc) {
00094                 printDescription((MemberDoc)element);
00095             } else if (element instanceof ClassDoc) {
00096                 printDescription((ClassDoc)element);
00097             } else if (element instanceof PackageDoc) {
00098                 printDescription((PackageDoc)element);
00099             }
00100         }
00101         dlEnd();
00102         hr();
00103     }
```

Figure 17, Line Numbering

3.2.4 Moderation

Documentation and source is published on a *javaJAM* server as a package. None, one, or many moderators can be assigned to each package. Moderators can manage the package's annotations. The following annotation moderation functions are available to a moderator:

- ***Accept***

  When an annotation is first posted its status is ***proposed***. This status indicates that the annotation has not been reviewed. The moderator can ***accept*** an annotation to indicate that the annotation will be incorporated into a future implementation of the package.

- ***Reject***

  Rejecting an annotation changes its status to ***rejected***, but the annotation remains available for review.

3.2.5 Actors and Roles

javaJAM is designed to be simple. There are only four categories of javaJAM participants. The four categories are listed below:

- **Guest:** Guests are not required to supply any information. Guests cannot participate however. They can only view the information available; source, documentation and related annotations.
- **Participant:** A participant is one who has signed up on the *javaJAM* server. Participants can post annotations. Sign up is required so that the author of an annotation can be authenticated and contacted should there be any follow-up necessary for an annotation**.**
- **Moderator:** Application moderators have responsibility for the applications hosted on a javaJAM server. Usually one moderator is assigned to one hosted application. It is possible to assign more than one moderator to a single application and to assign a moderator to more than one application.
- **Administrator:** The role of the administrator is to set up the hosted applications and provide access for moderators.

4. Implementation

javaJAM consists of three subsystems and has client and server components. The client component is implemented in HTML and works from any frames-capable, browser. The server component is implemented in Java and uses servlets according to version 2.0 of the Sun Servlet API [14]. The server piece can run on any platform that supports servlets.

javaJAM was developed with Sun's Java Web Server version 1.1.3 and later 2.0 on the server and with JDK 1.1.1 and later 1.2.2. *javaJAM* has been extensively tested with JavaWebServer 2.0, JDK 1.2.2, Internet Explorer 5.x and Netscape 4.x. (see Appendix F for the Quality Assurance Test Suite).

4.1 Documentation Tools Subsystem

The first *javaJAM* subsystem contains the extensions to Javadoc. It is used to generate all of the HTML source including the annotations buttons and also creates the four directories/folders (see Section 4.1.4) into which the HTML source for a published application resides. *javaJAM* is written in Java. This subsystem has 15 classes, 196 methods, and 2170 lines of code not including comments and blank lines.

It is necessary to execute javadoc.exe in order to generate the Javadoc documentation along with the *javaJAM* extensions. This requires a shell script (see appendix E for sample shell scripts).

4.1.1 Javadoc Frames Enhancement

javaJAM extensions to Javadoc contain several major enhancements. The first is that while the Javadoc look and feel has been maintained, there is an additional frame. This frame is placed at the bottom of the traditional two horizontal frames.

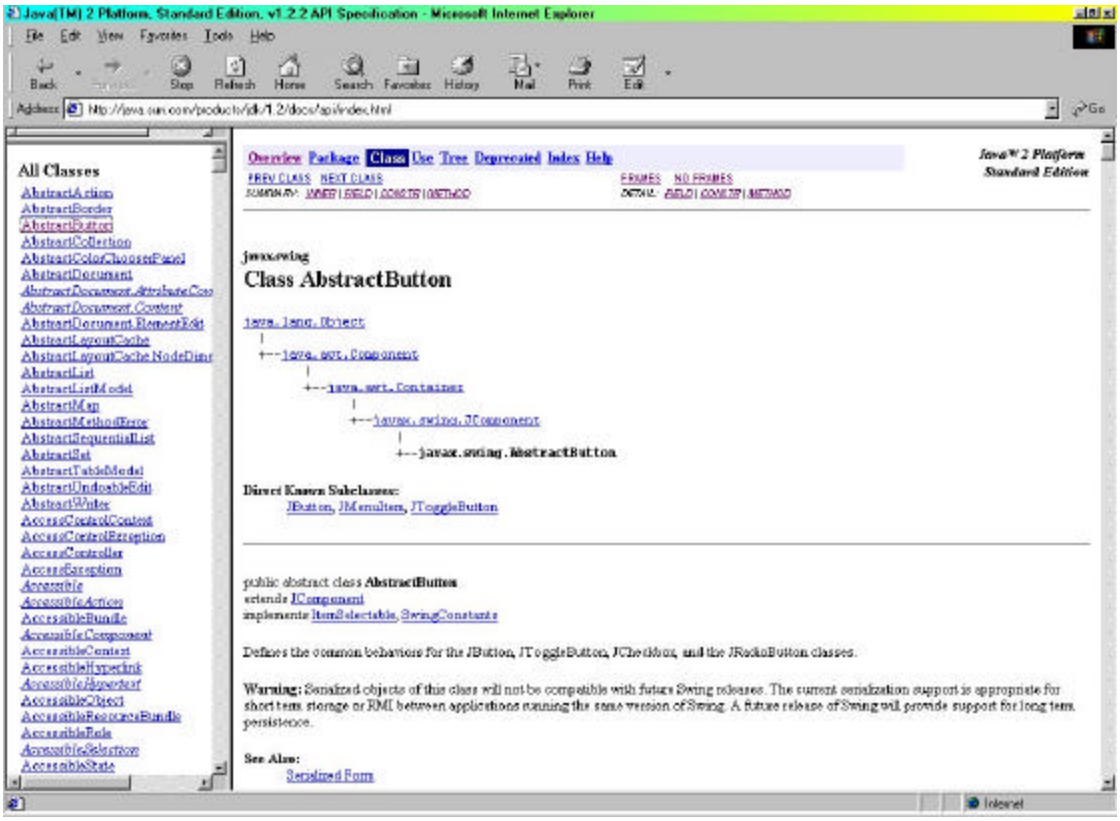


Figure 18, Standard Javadoc Frames

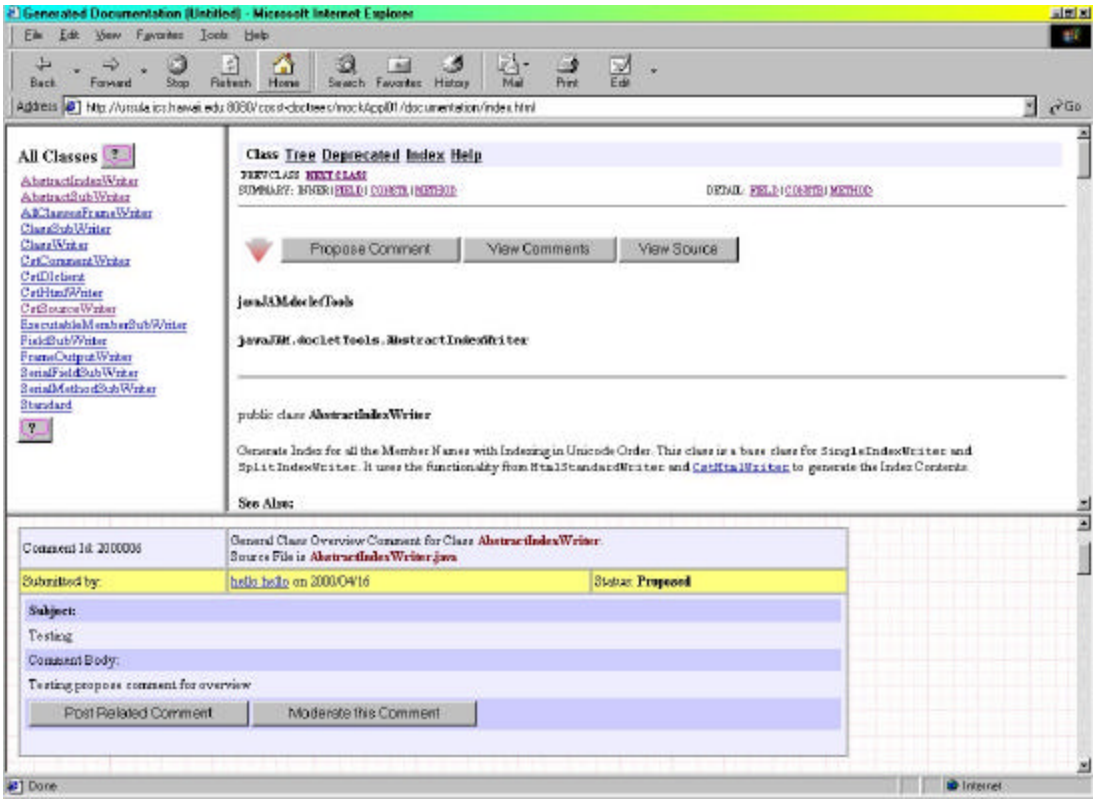


Figure 19, Enhanced Javadoc Frames

The new frame at the bottom is where *javaJAM* displays the annotations for documentation and for source. Using JavaScript the bottom frame is kept synchronized with the top-right frame. If the top-right frame is displaying documentation, as in the figure above, the bottom frame will show the related annotations. If the top-right frame is displaying source code, the bottom frame will show the annotations related to the source code. In the event that a browser does not support the JavaScript that makes this synchronization possible, the View Annotations button is another way to bring into view the correct annotations.

4.1.2 Javadoc Source Code Presentation Enhancement

If *javaJAM* is going to be a useful tool for the Open Source initiative it will have to include source code as well in the Javadoc generated documentation. The upper-right frame (see figure 14 above) by default displays Javadoc documentation to be consistent with the standard Javadoc behavior, but there is a Source button that can refresh the frame with the related source code for the class being displayed. When displaying source it is also possible to switch back to viewing documentation.

javaJAM by default inserts red line numbers into the source code presentation. The purpose of the line numbers is to ensure that it is simple to reference a particular line in the source when creating an annotation.

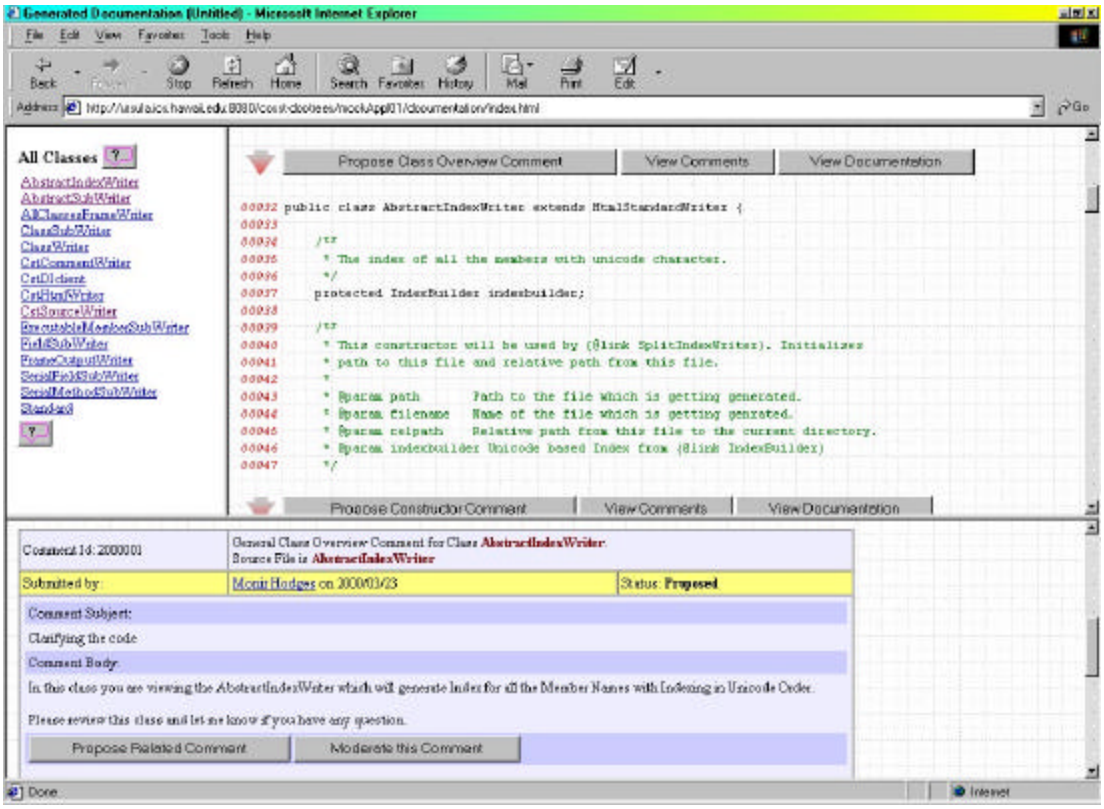


Figure 20, Enhanced Source Code Presentation

4.1.3 Javadoc Annotations Buttons Enhancement

There are a number of new buttons provided to assist with viewing source and creating and reviewing annotations. These buttons are dispersed throughout the documentation and source. The following types of buttons have been introduced:

- Create Annotation
- View Annotations
- View Source
- View Documentation
- Help

The Help buttons features context sensitive help. The on-line help appears in its own dialog box as a new instance of the browser. When help is invoked an anchor identifier is also passed. This anchor identifier is used by most browsers to position to a specific location inside

the file specified by the URL. Using this anchor identifier *javaJAM* provides context sensitive help.

4.1.4 Javadoc Output Files and Organization

Standard Javadoc output generates documentation to a folder named *documentation*. JavaJAM extends this by adding three more folders. One for the HTML rendered source and two for the annotations, documentation and source:

- documentation
- source
- doc_comments
- src_comments

All these folders contain HTML source files. Hierarchically, these folders are underneath the folder that is named to uniquely identify the hosted package. The structure of a collection of hosted applications might look like:

- public_html
    - cosst_doctrees
        - LLOCv1-10
            - documents
            - doc_comments
            - source
            - src_comments
        - javaJAMv0-80
            - documents
            - doc_comments
            - source
            - src_comments

The references to "cosst" and "comments" reflect an earlier design stage when *javaJAM* was referred to as the Collaborative Open Source Software Tool and when the term "annotations" was introduced in place of "comments" to refer to the information being collected.

4.2 Site Tools Subsystem

The second subsystem contains the *javaJAM* site tools. These tools assist the *javaJAM* administrator with tasks such as reviewing the installation and setup, reviewing the errors log, and, most importantly, with hosting and rehosting applications. Hosting an application requires generating the Javadoc HTML files and moving them into the public directory of Java Web Server 1.1.3 or higher. Those tools can be remotely accessed so that administration does not have to be performed from the server. This subsystem has 4 classes of which 3 are servlets, 27 methods, and 747 lines of code that are neither comment lines or blank lines. It also relies on a shell script to execute javadoc.exe.

Figure 21, Site Tools Menu

4.2.1 Server Settings Review Tool

The Server Settings Review tool is implemented as a servlet. This tool presents an HTML page that displays all of the *javaJAM* settings. These settings include the internal and also those stored in the javaJAM.ini file that is used for site-specific initializations. Of the internal settings that are displayed, the critical one is the *javaJAM* version number, which is stored in the class CstGlobal. The version identification appears as v11.22.33 where '11' is the major version, '22' is the minor version, and '33' is the version build. At the time of this writing javaJAM is at v00.80.60.

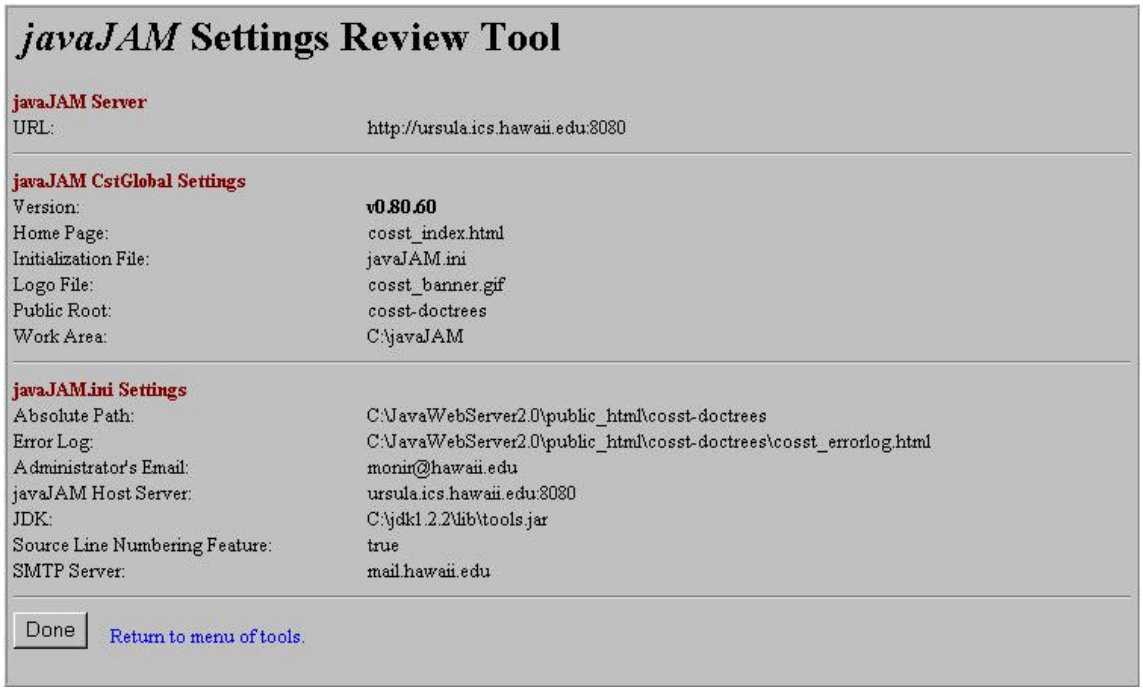


Figure 22, Settings Review Tool

4.2.2 Server Doclet Interface Tool

The Doclet Interface tool is implemented as a servlet and generates all of the directories and files necessary for hosting a package on a *javaJAM* web server. From the Site Tools menu the Doclet Interface tool is invoked from the option "File Upload and javaJAMdi Execute Tool." This option provides for uploading a .zip file to the *javaJAM* server, unzipping it using Classes in java.util.zip, and running the *javaJAM* doclet interface that generate Javadoc. The results of executing this tool are logged into a separate instance of the browse. See appendix G for a sample log.

4.2.3 Server Host/Rehost Tool

After the Doclet Interface Tool is executed against a package it is nearly ready to be hosted on any *javaJAM* server. The final stage of hosting involves a few steps:

1. Update all of the files' links to correctly reference the server and to also correctly reference the package's root directory.
2. Create a new package root on the javaJAM server.
3. Move the resulting output files and their folders to the *javaJAM* server so that they are available.

If it ever a hosted package must be moved to another server, it would be necessary to update all of the files' links in all of the HTML files, including the annotations. The Host/Rehost Tool can take care of this. If ever a hosted package must be renamed, to add version identification so that a package can be hosted more than once, the Host/Rehost Tool can take care of this too.

4.3 Review and Annotation Subsystem

The review and annotation subsystem presents applications for review and collects the annotations as they are posted. This is the core of *javaJAM*. This subsystem is written in Java and contains 12 classes of which 3 are servlets, 177 methods, and 2386 lines of code that are not in-line comments or blank lines. It also contains some static HTML pages and dynamically generates additional pages that include a minimal amount of JavaScript [12] to ensure that frames displaying source or documentation stay synchronized with their related annotations, which appear in the lower frame.

4.3.1 Authentication and Security Model

The security model implemented in *javaJAM* is mainly intended to ensure that the author of an annotation is authenticated. It helps to establish credibility and also makes it possible to communicate directly with the author by email where necessary. Authentication is handled by a combination of author's email address and password. The email address is assumed to be unique to the author. To acquire a password requires signing up on a *javaJAM* server and supplying a username (known as Participant name), and an email address. If the *javaJAM* server does not recognize the email address it is stored and a password is generated. This password is emailed to the new participant.

Passwords are a necessary evil. If authorship of annotations is misused or doubted, the collaborative environment can be compromised. Passwords help ensure authentication of authorship. To make it easier to remember *javaJAM* passwords, they are only five characters long and designed to be pronounceable. To help make them a little more secure, they also include several digits and an uppercase letter. Typical passwords might look like; cAt34 and 22Wit. Some effort has been made to help prevent naughty passwords.

The worst part about passwords is forgetting them. *javaJAM* includes a reminder feature that allows a participant to request that her email address be again emailed to the participant. Moderators are also authenticated using the same mechanisms. Guests are not authenticated because it is not necessary. They cannot participate in any way. They can only review documentation, source, and annotations.

4.3.2 Annotation Life Cycle

Annotations are the reason *javaJAM* exists. The life of an individual annotation begins with the Annotate action. A participant posts an annotation and the new annotation is stored.

The annotation life cycle:

1. Participant posts annotation - status is ***proposed***.
2. Annotation is reviewed by moderator and accepted, rejected, or deleted - status is ***accepted*** or ***rejected***.
3. Moderator forwards collection of accepted annotations to development team.
4. Team incorporates annotation into next release of application.

As an annotation move through its life cycle its status changes. The status of the annotation helps reviewers who post annotations to understand how their annotation might impact the package.

Annotations can have one of the following statuses:

- ***proposed*** - new annotation
- ***accepted*** - reviewed by moderator and accepted
- ***rejected*** - rejected by moderator

Annotations can be managed with the following options:

- Post Annotation - new annotation with status Proposed
- Accept Annotation - change status to Accepted
- Edit Annotation - edit annotation subject and body
- Reject Annotation - change status to Rejected
- Delete Annotation - annotation is deleted and no longer exists

When a moderator reviews an annotation the moderator has the privilege of either Accepting or Rejecting the annotation. Accepted annotations can be subsequently collected and passed on to the development team for review and action during the next product development cycle. Rejected annotations are retained. They may lead to additional annotations or they may be accepted at some point in the future. The moderator also has the capability of deleting annotations that are not appropriate at all.

4.3.3 *javaJAM* Dialogs

There are a number of dialogs that appear as new instances of the browser. These dialogs make it possible to perform related work in separate windows so that the review process

is not interrupted or the reviewers place lost. The following dialog instances of the browser are available:

- Help
- Create Annotation
- Annotation Creation Confirmation
- Maintain Annotation (for Moderators only)
- Annotation Maintenance Confirmation

4.4 Site Initialization File

The site initialization file (see sample in Appendix H) must exist in the javaJAM folder and be named javaJAM.ini. This file contains site-specific information that ensures that *javaJAM* can be easily ported from one site to the next without the need for source code modifications.

4.5 Authentication Rationale

Authentication of participants is important for establishing trust and open communication. All annotations are posted with the email addresses and names of the participant. This makes it possible for discussions related to the annotation to be followed up on by email or off-line if necessary.

javaJAM authenticates all active participants. Authentication is established by providing an email address for user identification, a user name (first and last) to associate with the email address, and a *javaJAM* assigned password. Passwords (see Section 4.3.1 for more on authentication) are obtained by signing up on a *javaJAM* server. After signing up, the password is sent to the email address the user provided when signing up.

4.6 Client/Server Architecture

javaJAM contains client and server components. Some the client components are static HTML pages and others HTML pages that are dynamically generated by servlets on the server. The client components do all of the presentation. The server components do authentication and reading and writing of annotations to disk.

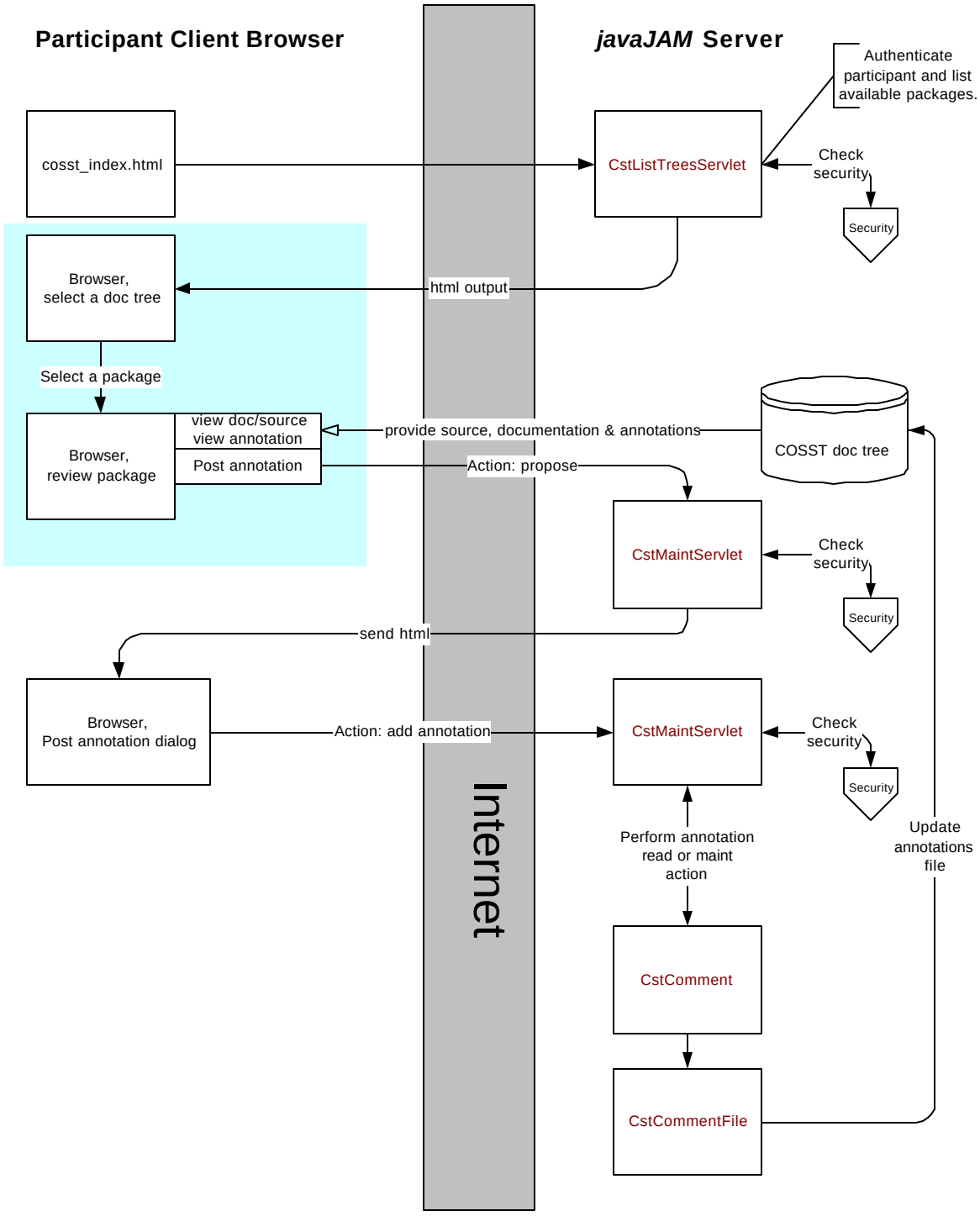


Figure 23, Participant Architecture

The moderator support design is nearly the same as that of the participant. The difference is that the moderator is able to perform more actions on annotations where the participant can only review and post.

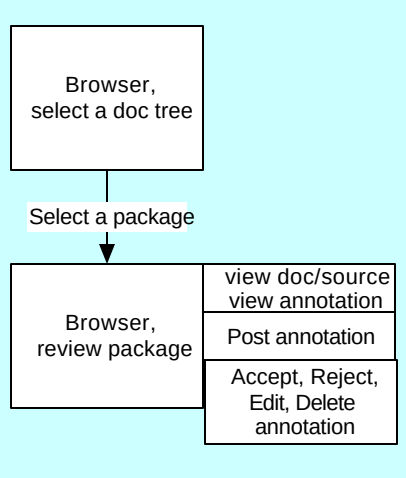


Figure 24, Moderator Architecture

The guest support design is nearly the same as that of the participant. The difference is that the guest cannot actively participate. The guest is only able to review annotations.

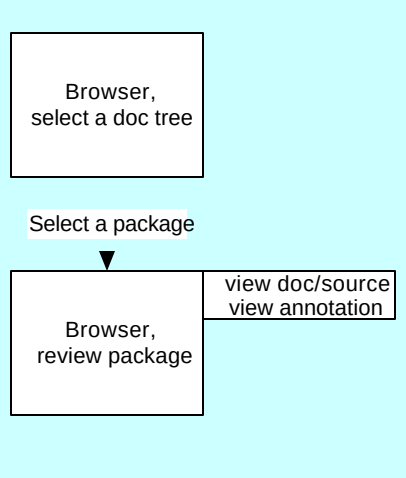


Figure 25, Guest Architecture

5. Case Study

5.1 Design and Goals

To evaluate the research hypothesis that *javaJAM* improves software quality through collaboration in a web-enabled environment, I designed a case study on the use of *javaJAM* and the effect of collaboration to improve software quality. The goal of the study was to answer the following research questions:

- Can students collaborate effectively and improve the quality of their code as a result of computer-mediated communication?
- Will *javaJAM* provide an effective environment to share source and collaborate?
- Will students find the annotation process of the *javaJAM* easily accessible and preferable to existing collaboration methods they might use such as email?
- If students are not familiar with collaborative tools, will they find them helpful?

5.2 Method

The study was implemented in the classroom environment since that was the easiest way to set up a study with many people involved. The study involved two ICS 211 (Introduction to Computer Science II) classes, one at the University of Hawaii at Manoa and one at Honolulu Community College. 50 students participated. The students were teamed in groups of three or four students. There were 13 groups and 37 projects were hosted. The students' backgrounds included two semesters of introductory programming in Java.

Instructors and students both supplied valuable information to help assess this thesis and the effectiveness of the *javaJAM* implementation. Student feedback was obtained by administering a questionnaire (see Appendix A) on the last day of class.

- **April 10, 2000:** *javaJAM* was introduced to the two instructors, Blanca Lopez and Samuel Rhoads. While both instructors responded favorably to the concepts, one of them required immediate changes. In order for the students to participate in a group project using *javaJAM* for collaboration, there would have to be a way to prevent student groups who were working on the same assignment to review code from other groups. *javaJAM* did not have provisions for restricting access by group identification. This feature had to

be implemented quickly so one of the instructors would permit the study to proceed.

- **April 11, 2000:** students were introduced on April 11th to *javaJAM* with a short PowerPoint presentation and a demonstration.
- **April 15, 2000:** *javaJAM* software was revised to accommodate for the controlled group interaction. This was accomplished by introducing the concept of team. If a package is assigned to team members then only those members will see the package listed on the *javaJAM* server.
- **April 18, 2000:** students signed up for a *javaJAM* password. Once they got their password they could be a participant and post annotations on their group's hosted application.
- **April 19, 2000:** the students began to send their Java source after the first clean compile to the *javaJAM* administrator to be hosted and assigned to their team. Each group could then see their project on the server and could participate in project review and annotation.
- **April 22, 2000:** *javaJAM* was revised in the process to allow for private methods. Javadoc by default does not include private methods in its processing. This required an additional setting to be specified on the command line used to execute javadoc.exe.
- **May 3, 2000:** the students are done hosting and reviewing applications.
- **May 6, 2000:** a questionnaire (see Appendix A) was distributed in both classes to a total of 50 students. The instructors encouraged the students to participate in the evaluation of the *javaJAM*. 38 students responded.

5.3 Results

5.3.1 Instructor Feedback

One of the instructors indicated that the students could not share program source between groups. In the ICS 211 course, students are graded based on their Java code and style of programming. Because of this, the sharing of source code as done in an open environment was not encouraged. *javaJAM* was quickly modified to also incorporate features for restricting packages to assigned teams.

5.3.2 Student Feedback

The student feedback (see figure 27 in Appendix B) indicated that the students found value in using *javaJAM* and could see it's potential. They found the *javaJAM* interface to be fairly

self-explanatory and easy to understand and navigate with some students finding it difficult and some finding it very easy to use.

From the results of the survey, students found *javaJAM* to have an intuitive layout (question #2), to be easy to use for reviewing projects (question #6), to be easy to use for finding related source and annotations (#7), and to be easy to use for adding related annotations (#8). *javaJAM* was moderately successful as an easy tool for navigation (#3) and for publishing (question #5). The on-line help was not well received (#1) and the perceived response time, while not found lacking by a third of the students, still managed to draw the most criticism.

5.4 Discussion

Student's expectations varied according to their programming skills. Those with less skill expected *javaJAM* to assist with debugging code for syntax errors. One of the two courses in the study required Javadoc, and the other did not. Some of the students with no Javadoc experience preferred to go back to the way they were comfortable communicating, email and face-to-face. In the course that did require Javadoc, the students were very comfortable with *javaJAM*.

Students complained about the *javaJAM* problems. One was the password mailing issue and the second was the delay caused by the manual process for hosting team projects. The email problem seems to be specific to the University of Hawaii mail servers. Testing from home using cable and dialup connections showed that RoadRunner and Lava.Net were compatible with *javaJAM*. Students also indicated that they would have preferred an automated process for hosting their projects. They wanted to host their own and have them published immediately. The students' desire for an automated process for hosting packages is very important. *javaJAM* will need to better embrace a self-service model if it is to facilitate and not interfere with or slow down collaboration. *javaJAM* requires a flexible hosting tool (see Section 8.2.1).

There was one other situation where the *javaJAM* application itself contributed to perceived performance problems. When a classroom full of students attempted to simultaneously authenticate on the *javaJAM* server, performance got very slow. This is because *javaJAM* data files are currently stored as sequential files and *javaJAM* uses a token to let only one user at a time access the authentication file. Implementing *javaJAM* data on a relational database will solve this as well as make *javaJAM* more easily scalable (see Section 8.1.1, Scalability Improvements).

From reading the student responses, impatience with *javaJAM* was aggravated by the end of the semester time pressures.

6. Conclusion

The goal of this research was to determine whether collaborative software development in the Open Source environment would be significantly facilitated by a web-enabled tool that provides for the collection of annotations and the creation of relationships between annotations and their corresponding sections of documentation and source code. To reach this goal the *javaJAM* tool was developed for collecting comments and suggestions and creating relationships between them and the sections of source and documentation that inspired them. This new web-enabled, collaborative tool was tested to determine whether new tools such as *javaJAM* could assist with collaborative software development in the Open Source environment. The case studies indicate that *javaJAM* if further developed could be a valuable tool.

6.1 Contributions of this Research

The primary contribution of this research is a web-enabled collaborative tool for software review and annotation. The strength of this tool is that it creates relationships between sections of documentation, source and posted annotations. These relationships facilitate software review and feedback. The purpose of this collaborative tool is to help teams improve software quality. This tool works towards that goal.

6.2 The Javadoc Advantage

The design of a web-enabled collaborative tool for software review and annotation requires the consideration of three fundamental elements: documentation, source, and review annotations. Fortunately, because the design assumed that Java would be the supported software language, it was obvious that Javadoc could play an important role, which it did. Javadoc proved to be a powerful tool. It was also not too difficult to extend Javadoc so that it not only rendered documentation in HTML, but also the source. With Javadoc not too much effort was required to quickly envision a solution for two of the three fundamental elements of this project.

Javadoc is unique to Java and provides for the Java language a unique advantage over other languages. Application development in Java is inherently richer than application development in other languages as a result of Javadoc. The study done for this thesis demonstrated that even students who did not annotate their programs to take advantage of the

capabilities of Javadoc nonetheless found the resultant documentation produced by Javadoc to be helpful for reviewing their projects.

Though Javadoc is extensible, only some of its source code has been made available. Members of the Open Source Initiative community have called upon Sun to publish Javadoc as Open Source Software. If that were to happen, maybe Javadoc functionality would be ported to other languages making it easier to provide tools like *javaJAM* for other languages too. Javadoc presents important technology and emphasizes the role of documentation. Greater consideration should be given to ensuring that all student assignments in Java require Javadoc generating documentation as being equally important to producing Java source code.

6.3 Observations on Teaching

javaJAM emphasizes the relationship between an annotation and the related areas of source and documentation. For example, if the third overload method of a specific class inspires annotations, it is easy to review those annotations and also to view the related documentation and the related source to quickly get a fuller picture of the discussion. Feedback from the study indicates that even second year students with no training or exposure to collaborative software development grasp the potential this offers as a learning tool and a tool that can improve the quality of the software that they develop.

It is interesting to observe that the art of teaching skills like drawing and painting have evolved over the last three centuries or more and have become very rigorous. Aspiring artists copy the works of the masters to learn and practice technique. The art of teaching programming is still very young. The methodology of teaching students does not emphasis collaboration or the studying of the practices and techniques of skilled programmers. Here too *javaJAM* can contribute, both by the sharing of well-written code and by the sharing of annotations that explain why it is well written.

7. Future Directions

7.1 Improved Support for Collaboration

7.1.1 Scalability Improvements

javaJAM was designed and implemented as a research and evaluation tool for the concepts that it proposes. Many design decisions were made in favor of speed and ease of implementation in order to test the thesis that a tool like *javaJAM* would be beneficial to the Open Source Software initiative. In order for *javaJAM* to successfully support large communities of software engineers, *javaJAM* storage of annotations will have to be redesigned. Currently, *javaJAM* uses a sequential flat-file storage structure. The advantage of the flat-file structure is that it is already in HTML format and so it is immediately available for browsing. To make *javaJAM* scalable will require that annotations be stored in a relational database. This would require changing several methods in the maintenance servlet.

7.1.2 Collaborative Teaching Tool

The requirements of the classroom can be much different than the requirements for producing Open Source software. In the classroom the teacher may want teams to learn to collaborate, but because multiple teams are working on the same assignment, it might be necessary to prevent teams from reviewing each other's work. Without imposing some security the entire class would probably function somewhat like a single team rather than as multiple teams as the teacher intended.

javaJAM has limited support for hosting the work of teams and hiding team projects from other teams, but the initial design for *javaJAM* did not fully embrace this requirement. To extend the security features would require functionality that would prevent teams from even being able to guess the URL's of other teams. Currently, javaJAM only hides the work of other teams when it lists the available projects.

In the new security model for *javaJAM* access to all hosted packages will need to be controlled. Also, the moderator, a teacher in this case, will have to be able to assign participants (students) to teams and assign teams to packages.

7.1.3 Moderator Comments

Annotations are mostly one-way communications. The annotation status provides some feedback. Since it is possible to create threads with annotations, that helps also to make discussion by annotation a richer experience, but the addition of moderator comments would also be very helpful in some scenarios. When an annotation is accepted or rejected there is currently

no accompanying explanation. If the moderator were able to comment on an annotation, this would provide for an additional way to communicate.

The current annotation layout only provides for annotation subject and body:

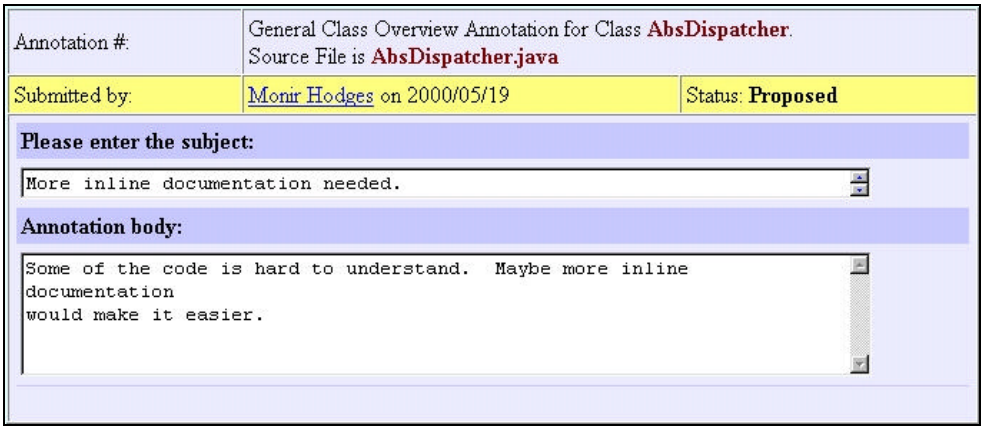


Figure 26, Post Annotation

The future annotation layout would also show moderator comments on the annotation if any exist:

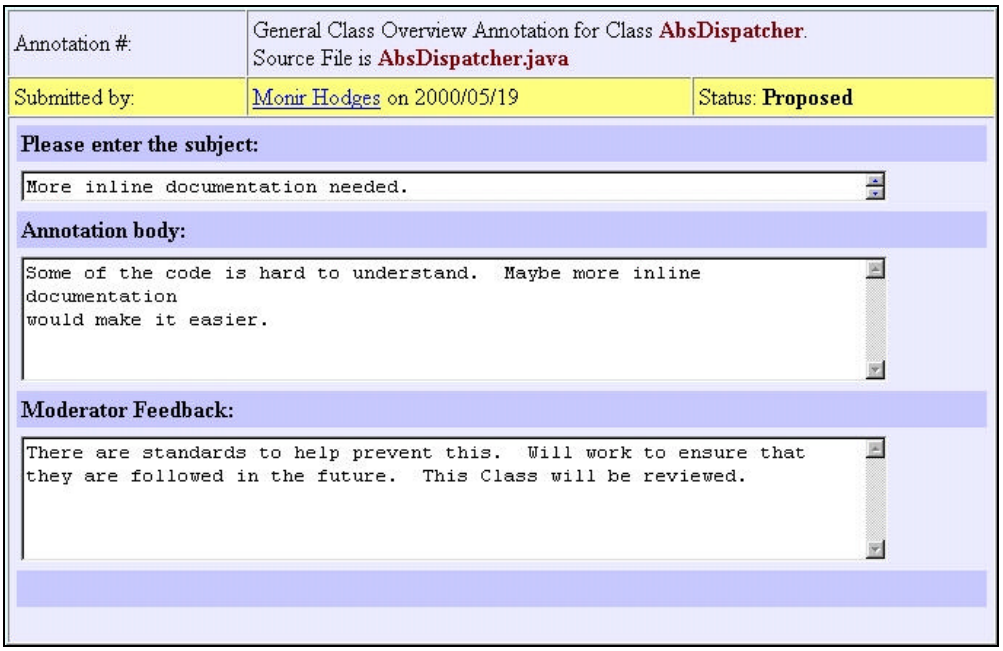


Figure 27, Future Post Annotation

7.1.4 Email Triggers

Some annotations may be considered extra sensitive or high priority by the author. An email trigger function would provide the author a way of automatically being informed that the status of the annotations has been changed. The updated annotation could be sent by email to the author at the time that the moderator changes the annotation status, or makes other changes.

7.1.5 Voting

Not all annotations should have the same weight. Some will have more support than others. Some annotations may even have detractors. Voting would help a developer to prioritize annotations. Unlike voting for politicians, it should be possible to vote against a particular annotation too. For example, it should be possible for three people to vote for a particular annotation and eight people to vote against it. To make this voting model collaborative, it would be necessary to thread a discussion for a particular annotation.

7.1.6 Threaded Discussions

Some annotations may benefit from having a related discussion thread. This would better organize the collaboration by reflecting the original annotation as the root of the threaded discussion. This would facilitate discussions related to annotation voting too.

7.1.7 Software Distribution

To more fully support the Open Source software development environment, *javaJAM* should also facilitate the distribution of source. This would not be difficult to implement since *javaJAM* already requires that a zip archive sent for hosting. By saving the archive and creating a link to it, it would be possible to make a hosted application's source available for download from a *javaJAM* server.

7.2 Improved Support for Application Hosting

7.2.1 Flexible Hosting tool

At this point the hosting tool is only available to the administrator. *javaJAM* would be much more flexible if the mechanism for authenticating moderators was much more flexible. The current mechanism requires the administrator to manually identify which applications a moderator can host and to manually host them for the moderator. The current tools support remote hosting of applications, so that simplifies the task of automation.

For the hosting tool to be flexible it would need to be changed to allow any authenticated moderator the option to host new packages on the *javaJAM* server. It would need to; 1) confirm that the package name is unique and able to be posted, 2) update the moderator record to indicate that the instructor owns the package, and 3) post the application on the server. If the

moderator is allowed to host packages without any intervention from the administrator, then it will be necessary to allow the moderator to also remove packages from the server once the package as hosted is no longer relevant. It would also be reasonable to allow the moderator to freeze the package before later removing it since the frozen version may help by providing a point of reference.

7.2.2 Package Version Control

Under some circumstances multiple versions of a package may be hosted at once and different versions may or may not be available for review or for annotating. This could be the case when an older version is kept as a reference version only and so should be only viewable, or it might be possible that a moderator would want to migrate specific annotations from one version to the next since they might represent ideas that will not appear in the next version, but are worth saving for some future version of the package.

7.2.3 Annotation Migration

Currently, the moderator can only change an annotation's status, or delete it. But annotations are valuable and may need to outlive a particular version of a hosted package. Functionality to provide for migrating annotations from one package or version to another package or version would provide great flexibility. If the annotations and their associated relationships were considered as the most valuable asset, then this function would protect that asset.

7.2.4 Automatic Clean Up

In order to simplify maintenance and prevent the accumulation of unused packages, teams, etc., *javaJAM* should use aging logic. The moderator could specify aging logic at the time the package is created. An "end-of-life" date could be specified for the package so that it would automatically be frozen or deleted, or both. Also, aging logic could be specified to determine how long a package could go without being accessed before being considered at the end of its life. A mechanism to provide the moderator an advanced email warning could help prevent an unintended deleting of a package.

7.3 Functional Enhancements

7.3.1 New Annotation Status

Individual annotations may have a complex life cycle. Additional statuses would help a moderator to move an annotation through its life cycle.

- ***Deferred***

  Currently, annotations are either accepted or reject by the moderator. It is also possible for the moderator to simply ignore a newly posted annotation, but that would not communicate a clear intention. A new "deferred" status would represent the situation where an annotation is beneficial, but cannot be implemented in the near future. Deferred annotations might possibly be migrated to some future version of the package.

- ***Hidden***

  In some circumstances it may be necessary to hide, temporarily or permanently, an annotation. This should not be a normal practice, but short of deleting a comment, this is a useful, nonpermanent alternative.

- ***Implemented***

  Some annotations will be suggestions for changes to documentation, source, or functionality. Having a status that indicates that annotation is implemented provides additional communication.

7.3.2 Extensibility

Besides supporting a range of standard statuses, also allowing a moderator to invent new status for a project or a site would create the most flexibility.

7.3.3 Annotations Reporting

Currently there is only one way to review annotations. Working with and managing annotations would be greatly enhanced by providing new ways to organize and review them. Annotations should be selectable and sort-able by a number of criteria including date posted, status, and author. The result would be a "collection" of annotations. Once a collection of annotations is created, it should be possible to manage a collection: accept, reject, delete, or migrate for example. Also functionality to allow these collections to be printed, emailed or even saved for future reference might help make *javaJAM* a powerful communication tool.

7.3.4 Password Encryption

The security model (Section 4.3.1) would be stronger if passwords were stored encrypted. Encryption could be done with a one-way hash function such as the Secure Hash Algorithm (SHA) developed by the NSA [5]. Java classes for SHA are available on the web.

One-way hashes are ideal for authentication. For authentication of a participant, the participant's password can be hashed and the hash compared against the database for a matching hash. Storing the hashed password for authentication has an additional advantage. Should someone obtain access to the data and the hashed passwords, having the password

hash is not the same as having the password itself. Also, password hashes are design to be irreversible. Discovering the original password from the hash requires cracking the SHA algorithm. This is not a casual activity.

7.4 User Interface Enhancements

7.4.1 Selective Viewing

While Javadoc already provides selective viewing of the entire package via the Index link at the top of each class and also selective viewing within a class in the summary section near the top of each class, additional selective viewing options would add more flexibility. The left frame currently lists the package's classes. If in front of the class name was a small plus sign, clicking on it could expand the class—like nested subfolders—so that links to constructors and methods would be presented for clicking.

7.4.2 IDE Integration

javaJAM would be a very strong and dynamic collaboration tool if it were integrated into an Integrated Development Environment (IDE) like Borland's JBuilder. After a project completes a successful compile an option to update the *javaJAM* hosted information is provided. If selected the updated documentation and source are replicated on javaJAM and all annotations with the status Accepted for the next version are updated to reflect the fact that they have been implemented in the current version. This status change would be automated and specific only to sections of the source that were modified. Annotations for areas of source that were not changed would not have their statuses change.

Appendix A, Sample Questionnaire

The following survey is being conducted in order to evaluate javaJAM. I am hoping that during the short time you have used this software tool, you have gathered enough insight to discuss javaJAM. Your insights and answers to the following questions will help me to find out if tools like javaJAM can help students to improve the quality of the software that they collaboratively develop.

Your completed survey will not be shared with your instructor.

Please answer each question by drawing a circle around your selection. For the essay questions please be as concise as possible. If the space provided is not enough, use the back of the survey sheet.

Thank you for taking the time to complete this survey.

javaJAM's on-line help was easy to use. **Strongly agree** **Agree** **Neutral** **Disagree** **Strongly disagree**	The layout and presentation of your documentation, source, and the posted comments/annotations easy to follow. **Strongly agree** **Agree** **Neutral** **Disagree** **Strongly disagree**
It was easy to move around the different parts of *javaJAM*. **Strongly agree** **Agree** **Neutral** **Disagree** **Strongly disagree**	*javaJAM*'s response time was reasonable? **Strongly agree** **Agree** **Neutral** **Disagree** **Strongly disagree**
javaJAM is an effective tool for publishing java program and generating javadoc on the Web. **Strongly agree** **Agree** **Neutral** **Disagree** **Strongly disagree**	It is easy to review the source and documentation of projects posted for your group. **Strongly agree** **Agree** **Neutral** **Disagree** **Strongly disagree**

When viewing documentation it is easy to find the related source and comments/annotations. (For example: viewing the documentation for a method, you could easily find the source.) **Agree** **Neutral** **Disagree** **Strongly disagree**	When reading comments it is easy to add related comments/annotations. **Strongly agree** **Agree** **Neutral** **Disagree** **Strongly disagree**

Did *javaJAM* allow you sufficient flexibility to work in the way you wanted? If not how would you improve it?

Was the feedback posted by your team members helpful in improving the quality of your code? If NO, please explain why.

YES
NO

Do you think *javaJAM* could be used as a good teaching/training tool?
(For example: To host sample projects that illustrate good programming style and practices.)
If NO, please explain why.

YES
NO

What problems did you have while using *javaJAM*?

What did you most like about *javaJAM*?

What did you least like about *javaJAM*?

Would you consider *javaJAM* suitable for sharing java source applications? Why?
YES
NO

Would you recommend using *javaJAM* in java programming classes as a review tool in team projects? Why?
YES
NO

Appendix B, Raw Questionnaire Results

Table 1, Raw Survey Results

	Strongly Agree	Agree	Neutral	Disagree	Strong Disagree
Q#1		7	15	11	
Q#2	1	15	7	9	1
Q#3		10	9	11	3
Q#4	1	10	5	9	4
Q#5	2	8	8	12	3
Q#6	1	16	10	5	1
Q#7		14	14	10	
Q#8	1	14	10	8	

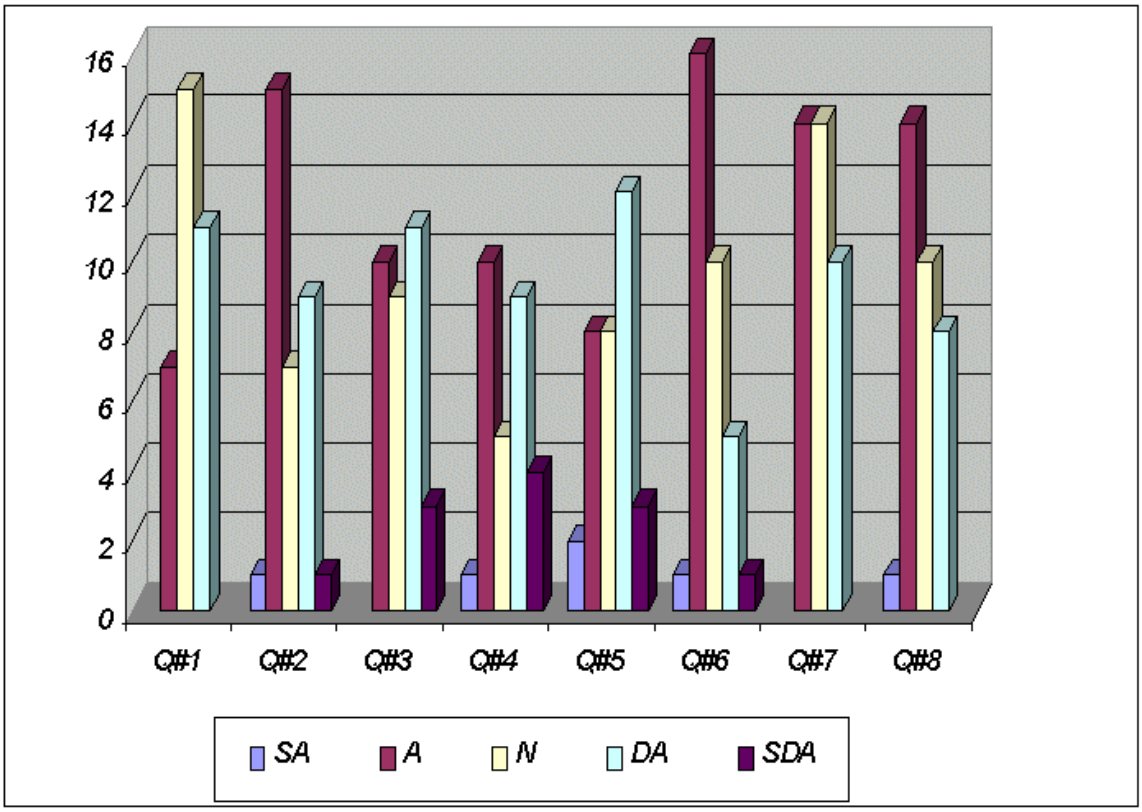


Figure 28, Survey Results Bar Chart

Appendix C, Sample *javaJAM* Sign Up Message

```
From: "Monir Hodges" <hodges@hawaii.edu>
To: <monir@hawaii.rr.com>
Subject: requested javaJAM reminder (fwd)
Date: Sunday, April 16, 2000 5:41 PM



Date: Sun, 16 Apr 2000 17:33:29 -1000 (HST)
From: hodges@hawaii.edu
To: monir@hawaii.rr.com
Subject: requested javaJAM reminder


Hello Monir Hodges,

Thank you for your request.  Your javaJAM information:

   Email...: monir@hawaii.rr.com
   Password: hiD84
   Username: Monir Hodges


NEW USERS:
Welcome to javaJAM.  You have indicated that you would like
participate in a javaJAM server.  You will need the password
and email address listed above to access this javaJAM server.

Please, never share your password.  Should you forgot your
password, use the [Lost Password] button on the opening
javaJAM page.

EXISTING USERS:
This message is being sent because of a request to send a
lost password.  The request was made for the password
associated with your email address.  This request can only
be generated from javaJAM by entering an email address
that is known to javaJAM and pressing the [Lost Password]
button.


Aloha!
```

Appendix D, *javaJAM* Data Structures

The supporting data structures are sequential files stored in C:\javaJAM on Win9x/NT systems. Records are stored one per line with fields and data stored as tuples. A sample record would look like: fld1=val1&fld2=val2&fld3=val3. It is possible that a field has no value, for example, field two has no value: fld1=val1&fld2=&fld3=val3

participant.jjd

Participants are authenticated against this file.

Layout:

email	Email address (participant ID).
password	Assigned password.
username	Name (First Last).
lastlogon	Last logon (yyyy/mm/dd). {not yet fully functional, shows date entry was added.
doctree	Packages that can be moderated. Delimit lists with "/". <all> permits moderation of all packages. <none> prevents moderation of all packages.

Example:

email=hodges@univ.edu&password=25nIc&username=Monir hodges&lastlogon=2000/05/04&doctree=<all>

team.jjd

Packages that appear here will be not be treated as Open Source. They will only be listed for participants assigned to the team that is allowed to review the package. This file was introduced during the Case Study to accommodate the classroom environment.

Layout:

doctree	Package that is assigned to a team.
email	Participant ID.

Example:

doctree=HeapSort-v01-00&email=hodges@univ.edu

userentry.jjd

This file works like a server-side cookie file. It provides persistent information during a *javaJAM* session. Entries expire automatically after 45 minutes of inactivity.

Layout:

clientIP	Participant's IP. Browser supplies this when it connects to the *javaJAM* server.
username	Name (First Last).
doctree	Package currently being reviewed.
status	This is set during initial authentication. "0" indicates unknown status, "1" indicates verified as participant. "2" indicates verified as moderator. "3" indicates guest.
email	Participant's ID.
timestamp	Timestamp. Updated during authentication and every time an annotation is posted or maintained.

Example:

```
clientIP=166.23.23.3&username=Monir Hodges&doctree=HeapSort-v01-00&status=2
        &email=hodges@univ.edu&timestamp=958874789230
```

Appendix E, Quality Assurance Test Suite

After each major revision of javaJAM a series of tests were performed to ensure that new features worked properly and that existing features were not broken. This series of tests is called a test suite. The table below illustrates the results of testing javaJAM version 0.80 with the test suite. The purpose of test suite is to help ensure quality software.

Table 2, Test Suite

	Version 0.80	
	Servlet Environment	
Description	Result	Final Result
Programmer enters username, but not email	pass	
Programmer enters email, but not username	pass	
CstListTreesServlet lists all available doc trees, and nothing else	pass	
Programmer is given many chances to enter email and username	pass	
Selecting a doc tree correctly begins access to a cosst hosted doc tree	pass	
First comment adds correctly to new comment file.	pass	
Propose a new comment where comment seq num 001 already exists	pass	
Class overview comments properly inserted and positioned	pass	
File overview comments properly inserted and positioned.	pass	
Class constructor comments properly inserted and positioned.	pass	
Class/Method comments properly inserted and positioned.	pass	
Class overview comments properly added to the end	pass	
File overview comments properly added to the end	pass	
Class/Method comments properly added to the end	pass	
Insert new method comment between existing method comments.	pass	
Insert new class comment between file and class comments.	pass	
Insert new file comment in front of class and method comments.	pass	
Insert method overload, second occurrence	pass	
Adding a comment with no subject or text is not accepted	pass	
Second Method sorts properly against first Method's comments.	pass	
Only comment purges correctly from comment file.	pass	
Deleting File comment deletes correct comment from comment file.	pass	
Deleting class comment deletes correct comment from comment file.	pass	
Deleting method comment deletes correct comment from comment file.	pass	
Deleting last comment leaves correct counter	pass	
Correct method comment purges from comment file.	pass	
File comment accept without edits replaces comment with status change.	pass	
Class comment accept without edits replaces comment with status change.	pass	
Method comment accept without edits replaces comment with status change.	pass	
Comment accept with edits replaces comment along with status change.	pass	
Comments buttons change appropriately when a comment is accepted.	Obsolete	
Comment buttons appear properly for proposed comments.	Obsolete	

Two or more users can propose comments without problem.	pass	
Comment text, paragraphs are changed to tags for storage.	pass	
Comment text, paragraphs are changed to end of lines for editing.	pass	
Adding comments leaves accurate counter at bottom of comment file.	pass	
Accepting comments leaves accurate counter at bottom of comment file.	pass	
Rejecting comments leaves accurate counter at bottom of comment file.	pass	
Rejecting comment changes status to rejected	pass	
Updating an existing comment does not increment counter at bottom.	pass	
Updated comment replaces previous comment on disk.	pass	
Updating a non-existant comment does not crash.	Obsolete	
Deleting a non-existant comment does not crash.	Obsolete	
Moderating a non-existant comment does not crash.	pass	
File comment hyperlink correctly referenced from source.	pass	
Class comment hyperlink correctly referenced from source.	pass	
File comment hyperlink correctly referenced from source.	pass	
Create first userEntry with all fields	pass	
Add new userentry to empty list	pass	
Insert new userentry at top reserving rest of list	pass	
Update userEntry timestamp, in first position	pass	
Update userEntry timestamp, in middle position	pass	
Update userEntry timestamp, in last position	pass	
Read moderator from empty file	pass	
Read moderator from top of file	pass	
Read moderator from middle of file	pass	
Read moderator from end of file	pass	
Programmer cannot moderate, status changes to 2	pass	
Moderation privilege automatically stops when the user entry becomes expired.	pass	
Remove expired userentries	pass	
autosizing/positioning of dialogs	pass	
Help button appears in logon and reprompt forms and works	pass	
Help button appears in dialogs, and works	pass	
javaJAM buttons appear for source overview for all java source files	pass	
javaJAM buttons appear for class overview for all java source files	pass	
javaJAM buttons appear for class overview for all JavaDoc files	pass	
javaJAM buttons appear for all constructors for each java source file	pass	
javaJAM buttons appear for all constructors for each JavaDoc file	pass	
javaJAM buttons appear for all methods for each java source file	pass	
javaJAM buttons appear for all methods for each JavaDoc file	pass	
javaJAM cst_index assignments match between JavaDoc and source files	pass	
javaJAM hyperlink assignments match between JavaDoc and source files	pass	
javaJAM buttons have correct HIDDEN statements	pass	
New signups are stored in participants	pass	
New signups receive email	fail	fail
Participants cannot sign up more than once	fail	pass
Source comments code and line numbering presented in different color	pass	
Error trap and log email send failures	pass	
Team Security Tools		
Assign packages to team members	pass	

Restrict designated packages to team members	pass	
Moderator Tools		
upload zip file to javaJAM server	pass	
unzip uploaded files	pass	
execute javaJAM doclet interface to create javaJAM javadocs	pass	
host javaJAM javadocs	pass	
rehost javaJAM javadocs	pass	
Browser Tests		
All tests pass in IE 4x	pass	
All tests pass in IE 5x	pass	
All tests pass in Netscape 4x	pass	
All tests pass in Netscape 5x		

Appendix F, Win9x/NT javadoc.exe Shell Script

```
rem  title javaJAM Host/Rehost Tool
rem  javadoc !is run from directory containing the java source files.
rem  -doclet <doclet package/component/class>
rem  -docletpath (full path to doclet package root>
rem  -d <application package root>
rem  -classpath <full path to application package>
rem  *.java  !application source files
rem  ---- javadoc executes relative to the current default directory
rem  o For NT the first line, the title line, does not have to be
rem    suppressed.
rem  o For DOS the cd command is limited to C: drive.  Seems that this
rem    is a security feature imposed when command.com is executed as a
rem    child process.
rem  o The javaJAM host/rehost tool is also assuming that the default
rem    directory will be the javaJAM workarea, which is
rem    c:\javaJAM\tohost.
rem  o The path statement is required if the jdk is not already
rem    defined.
rem  o Javadoc requires the -private tag to prevent the private
rem    methods from being omitted from the documentation and the
rem    javaJAM button omitted from the source.
rem  o The final echo statement is required to work around a jdk bug
rem    that prevents a DOS child process from exiting properly.  The
rem    work around is to look for the echo and then to kill the child
rem    process.
cd c:\javaJAM\tohost
path c:\jdk1.2.2\bin;%path%
javadoc   -private   -doclet   javaJAM/docletTools/Standard   -docletpath
c:\JavaWebServer2.0\servlets\javaJAM.jar -d .\documentation *.java
echo BATCH DONE
```

Appendix G, CstUploadEtcServlet Log

Cleanup:

```
C:\JavaWebServer2.0>rem Clean up work area by removing previous
application entirely.

C:\JavaWebServer2.0>cd c:\javaJAM\tohost

C:\javaJAM\tohost>deltree /y *.*
Deleting documentation...
Deleting upload.zip...
Deleting AbsDispatcher.java...
Deleting AbstractTag.java...
Deleting LOCcli.java...
Deleting DLOCabsReader.java...
Deleting JavaAbstract.java...
Deleting DLOCcli.java...
Deleting LOCabsReader.java...
Deleting FileListFilter.java...
Deleting TextAbstract.java...
Deleting source...
Deleting src_comment...
Deleting doc_comment...

C:\javaJAM\tohost>rem Standard doclet needs the \documentation
directory in order to run.

C:\javaJAM\tohost>mkdir documentation

C:\javaJAM\tohost>echo BATCH DONE
BATCH DONE
```

Unzip:
Filename: \\Sophie\syswin98\javaJAM\tohost\upload.zip
Content-Type: application/x-zip-compressed
Contents:

File: AbsDispatcher.java	**Size:** 6583
File: AbstractTag.java	**Size:** 2363
File: LOCcli.java	**Size:** 18317
File: DLOCabsReader.java	**Size:** 12572
File: JavaAbstract.java	**Size:** 17831
File: DLOCcli.java	**Size:** 17816
File: LOCabsReader.java	**Size:** 16674
File: FileListFilter.java	**Size:** 1838
File: TextAbstract.java	**Size:** 4525

---

javaJAMdi:

```
C:\JavaWebServer2.0>title javaJAMdi -- Doclet Interface: Generate html
for hosting

C:\JavaWebServer2.0>rem

C:\JavaWebServer2.0>rem javadoc !is run from directory containing the
java source files.

C:\JavaWebServer2.0>rem -doclet

C:\JavaWebServer2.0>rem -docletpath (full path to doclet package root>

C:\JavaWebServer2.0>rem -d

C:\JavaWebServer2.0>rem -classpath

C:\JavaWebServer2.0>rem *.java !application source files

C:\JavaWebServer2.0>rem ---- javadoc executes relative to the current
default directory

C:\JavaWebServer2.0>rem

C:\JavaWebServer2.0>cd c:\javaJAM\tohost

C:\javaJAM\tohost>path
c:\jdk1.2.2\bin;C:\JAVAWEBSERVER2.0\BIN\..\jre\bin;C:\JAVAWEBSERVER2.0\
BIN\..\lib;C:\JAVAWEBSERVER2.0\BIN\..\native_lib;C:\WINDOWS;c:\windows;
c:\windows\COMMAND

C:\javaJAM\tohost>javadoc -private -doclet javaJAM/docletTools/Standard
-docletpath c:\JavaWebServer2.0\servlets\javaJAM.jar -d .\documentation
*.java
Loading source file AbsDispatcher.java...
Loading source file AbstractTag.java...
Loading source file DLOCabsReader.java...
```

```
Loading source file DLOCcli.java...
Loading source file FileListFilter.java...
Loading source file JavaAbstract.java...
Loading source file LOCabsReader.java...
Loading source file LOCcli.java...
Loading source file TextAbstract.java...
Constructing Javadoc information...
Building tree for all the packages and classes...
Building index for all the packages and classes...
Generating .\documentation\overview-tree.html...
Generating .\documentation\index-all.html...
Generating .\documentation\deprecated-list.html...
Building index for all classes...
Generating .\documentation\allclasses-frame.html...
Generating .\documentation\AAA-MainFrame.html...
Generating .\documentation\index.html...
Generating .\documentation\packages.html...
Generating .\documentation\../source\AbsDispatcher.html...
javaJAM: generating AbsDispatcher.html to ../source
Generating .\documentation\../src_comment\AbsDispatcher.html...
javaJAM: generating AbsDispatcher.html to ../src_comment
Generating .\documentation\../doc_comment\AbsDispatcher.html...
javaJAM: generating AbsDispatcher.html to ../doc_comment
Generating .\documentation\.\AbsDispatcher.html...
Generating .\documentation\../source\AbstractTag.html...
javaJAM: generating AbstractTag.html to ../source
Generating .\documentation\../src_comment\AbstractTag.html...
javaJAM: generating AbstractTag.html to ../src_comment
Generating .\documentation\../doc_comment\AbstractTag.html...
javaJAM: generating AbstractTag.html to ../doc_comment
Generating .\documentation\.\AbstractTag.html...
Generating .\documentation\../source\DLOCabsReader.html...
javaJAM: generating DLOCabsReader.html to ../source
Generating .\documentation\../src_comment\DLOCabsReader.html...
javaJAM: generating DLOCabsReader.html to ../src_comment
Generating .\documentation\../doc_comment\DLOCabsReader.html...
javaJAM: generating DLOCabsReader.html to ../doc_comment
Generating .\documentation\.\DLOCabsReader.html...
Generating .\documentation\../source\DLOCcli.html...
javaJAM: generating DLOCcli.html to ../source
Generating .\documentation\../src_comment\DLOCcli.html...
javaJAM: generating DLOCcli.html to ../src_comment
Generating .\documentation\../doc_comment\DLOCcli.html...
javaJAM: generating DLOCcli.html to ../doc_comment
Generating .\documentation\.\DLOCcli.html...
Generating .\documentation\../source\DLOCcli.ButtonEvent.html...
Generating .\documentation\../src_comment\DLOCcli.ButtonEvent.html...
javaJAM: generating DLOCcli.ButtonEvent.html to ../src_comment
Generating .\documentation\../doc_comment\DLOCcli.ButtonEvent.html...
javaJAM: generating DLOCcli.ButtonEvent.html to ../doc_comment
Generating .\documentation\.\DLOCcli.ButtonEvent.html...
Generating .\documentation\../source\FileListFilter.html...
javaJAM: generating FileListFilter.html to ../source
Generating .\documentation\../src_comment\FileListFilter.html...
javaJAM: generating FileListFilter.html to ../src_comment
Generating .\documentation\../doc_comment\FileListFilter.html...
javaJAM: generating FileListFilter.html to ../doc_comment
```

```
Generating .\documentation\.\FileListFilter.html...
Generating .\documentation\../source\JavaAbstract.html...
javaJAM: generating JavaAbstract.html to ../source
Generating .\documentation\../src_comment\JavaAbstract.html...
javaJAM: generating JavaAbstract.html to ../src_comment
Generating .\documentation\../doc_comment\JavaAbstract.html...
javaJAM: generating JavaAbstract.html to ../doc_comment
Generating .\documentation\.\JavaAbstract.html...
Generating .\documentation\../source\LOCabsReader.html...
javaJAM: generating LOCabsReader.html to ../source
Generating .\documentation\../src_comment\LOCabsReader.html...
javaJAM: generating LOCabsReader.html to ../src_comment
Generating .\documentation\../doc_comment\LOCabsReader.html...
javaJAM: generating LOCabsReader.html to ../doc_comment
Generating .\documentation\.\LOCabsReader.html...
Generating .\documentation\../source\LOCcli.html...
javaJAM: generating LOCcli.html to ../source
Generating .\documentation\../src_comment\LOCcli.html...
javaJAM: generating LOCcli.html to ../src_comment
Generating .\documentation\../doc_comment\LOCcli.html...
javaJAM: generating LOCcli.html to ../doc_comment
Generating .\documentation\.\LOCcli.html...
Generating .\documentation\../source\LOCcli.ButtonEvent.html...
Generating .\documentation\../src_comment\LOCcli.ButtonEvent.html...
javaJAM: generating LOCcli.ButtonEvent.html to ../src_comment
Generating .\documentation\../doc_comment\LOCcli.ButtonEvent.html...
javaJAM: generating LOCcli.ButtonEvent.html to ../doc_comment
Generating .\documentation\.\LOCcli.ButtonEvent.html...
Generating .\documentation\../source\TextAbstract.html...
javaJAM: generating TextAbstract.html to ../source
Generating .\documentation\../src_comment\TextAbstract.html...
javaJAM: generating TextAbstract.html to ../src_comment
Generating .\documentation\../doc_comment\TextAbstract.html...
javaJAM: generating TextAbstract.html to ../doc_comment
Generating .\documentation\.\TextAbstract.html...
Generating .\documentation\serialized-form.html...
Generating .\documentation\package-list...
Generating .\documentation\help-doc.html...
Generating .\documentation\stylesheet.css...

C:\javaJAM\tohost>rem echo ...warning messages are ok

C:\javaJAM\tohost>rem dir/S c:\javaJAM\upload > c:\javaJAM\dir.tmp

C:\javaJAM\tohost>rem type c:\javaJAM\dir.tmp

C:\javaJAM\tohost>echo BATCH DONE
BATCH DONE
```

Appendix H, Site Initialization File

Sample file with explanations for each line:

1. **[server]**

   This is a section label. It is for readability only and is ignored.

2. **hostname=ursula.ics.hawaii.edu:8080**

   This is the host portion of the URL. It is used when generating new packages for hosting. It is also used when creating new annotations.

3. **jdkTools=c:\jdk1.2.2\lib\tools.jar**

   The path to the JDK is required for accessing javadoc.exe while generating a new package for hosting.

4. **jwsPath=c:\JavaWebServer2.0\public_html**

   The path to the root of the public area on the web server is used when generating new packages for hosting. It is also used when creating new annotations.

5. **linnum=on**

   When generating a new package for hosting, by default the source is generated with line numbers. Setting this to "off" will suppress the line numbers. For best results this should be left on.

6. **mailserver=smtp.hawaii.edu**

   javaJAM needs to communicate to a friendly SMTP server so that it can send new participants their passwords and help old participants remember forgotten passwords.

7. **[scripts]**

   This is a section label. It is for readability only and is ignored.

8. **cleanup=_cleanup.bat**

   This is name of the shell script stored in the javaJAM folder that cleans up the remote hosting work area at the start of a new remote hosting action. This script can be eliminated in a future enhancement.

9. **doclet=_javaJAMdi.bat**

   This is the name of the shell script stored in the javaJAM folder that executes javadoc.exe. Javadoc 1.2.2 could not be run directly from Java which seems unreasonable.

10. **[admin]**

    This is a section label. It is for readability only and is ignored.

11. **admin=Monir Hodges**

    The name of the javaJAM administrator is included in any email sent by javaJAM.

12. **email=monir@hawaii.edu**

    The email address of the javaJAM administrator is included in any email sent by javaJAM.

Any line can be commented out by placing a semicolon in front of it. Lines that contain invalid keywords are ignored. The line numbering is for illustration only and should not be included in the initialization file.

Bibliography

[1] AnalogX web site. <http://www.analogx.com/contents/download/system/maxmem.htm>.

[2] Apache Software Foundation web site. <http://www.apache.org>.

[3] Ben Adida and Philip Greenspun. Supporting Open-Source Software via Online Community. <http://photo.net/wtr/acs/open-source.html>.

[4] Ben Shneiderman. Designing the User Interface, Strategies for Effective Human Computer Interaction. Chapter 4: Usability Testing, Surveys, and Continuing Assessments. Addison-Wesley Press, 1998, Third Edition.

[5] Bruce Schneier, Applied Cryptography, Section 18.13: Choosing a One-Way Hash Function. John Wiley & Sons, Inc., 1996, Second Edition.

[6] Bruce Perens. The Open Source Definition, June 1997. <http://www.opensource.org/osd.html>.

[7] Dan Suthers. Artifact-Centered Discussion Forum. <http://lilt.ics.hawaii.edu/lilt/opportunities/students/linked-cmc.html>

[8] Danu Tjahjono, CSRS Research Summary. <http://csdl.ics.hawaii.edu/Research/CSRS/CSRS.html>.

[9] Eric S. Raymond. The Cathedral and the Bazaar, May 2000. <http://www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-bazaar.html>.

[10] Giant Java Tree web site. <http://gjt.org>.

[11] James C. Luh. Open For Business. Internet World, September 15, 1999 <http://www.internetworld.com/print/1999/09/15/website/19990915-business.html>

[12] Jason Manger. JavaScript Essentials. Chapter 8: Manipulating Windows with JavaScript. McGraw-Hill, 1996.

[13] Jeremy Brown, et al. HyperCode, September 1994. <http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/DDay/brown/hypercode/hypercode.html>.

[14] Karl Moss. Java Servlets. Chapter 6: Server-Side Includes. McGraw-Hill Press, 1998.

[15] Paint Shop Pro 6.0 web site. <http://www.jasc.com>.

[16] Phillip Johnson, OpenJavaDoc: An Open Source Browser for Java. <http://csdl.ics.hawaii.edu/FAQ/FAQ/opportunities.html>.

[17] sourceXchange web site. <http://www.sourcexchange.com>.

[18] Sun Microsystems, Inc. Javadoc Tool Home Page. <http://java.sun.com/products/jdk/javadoc>.

[19] Tango Interactive web site. <http://www.webwisdom.com/tangointeractive>.

[20] The Open Source Initiative web site. <http://www.opensource.org>.

[21] The Open Source Initiative, The OSI Certification Mark and Program. <http://www.opensource.org/certification-mark.html>.