ProCol

A jEdit Plugin for Online Project Collaboration

CS470 Final Report
Justin Dieters
Spring, 2004

Table of Contents

1.0 Introduction

ProCol is a project collaboration plugin for the jEdit text editor. The client operates through jEdit's plugin capability, using jEdit as a full-featured development environment. The server is able to be run from within jEdit as a plugin, or from the command line as a standalone server. This allows the server to be easily run on a developer's workstation, or on a standalone server machine. Configuration of the server and projects is done via properties files, which are able to be used with Java's Properties class.

2.0 Project Overview

2.1 Project Purpose

The target users for the ProCol plugin are small groups of programmers, from two to ten people, who are working together on a small to medium-sized project. It provides a common file repository, from which users are able to check out files for use. It also provides several communication and project management tools.

Current solutions to facilitate online collaboration between programmers, such as CVS, SourceForge, and gForge are often too complex to set up and too time-consuming to maintain for small projects that are being developed by a small number of users. ProCol is designed to provide a quick and easy set-up for short projects, while at the same time, providing enough functionality to be used over the full life of longer projects.

2.2 ProCol Services

ProCol provides several services to it's users. First and foremost, it provides a file repository with checkout functionality so that several people can work on the same project together without having to worry about changes by one user being overwritten by another user's changes to the same file. This includes versioning and changelog support. Secondly, it provides communication tools in the form of public and private messaging, and project management tools in the form of a todo list and a bug list.

2.3 Development Priorities

ProCol was developed with several priorities in mind. Usable file checkout functionality was of first and foremost importance. Private messaging, public messaging, todo list, and the bug list were also very important, however advanced functionality of these features was not implemented due to time constraints.

The software was developed with future development plans taken into consideration. Advanced functionality such as calendar and scheduling, concurrent editing, and web access for public viewing of the project are all highly desirable functions, but were not implemented for this class due to time constraints. Instead, the focus of the initial design and implementation was on creating a product with a solid framework and essential functionality implemented first, but the plan for additional functionality taken into consideration during the design.

3.0 Project Requirements

Since this this software was not built for a specific customer, there were no definite project requirements. Therefore, project requirements were developed by looking at existing solutions, at possible improvements to these solutions, and from my own experience with working with alternate solutions to the problems ProCol addresses.

Since the scope of this project was fairly large, the goal of the design and development of this software for this class was to bring the software to a level where it has the features and stability required for an initial version to be released and for development continued as an open source software project.

ProCol is made up of two major parts – the client and the server. Both the client and the server have their own unique requirements. There were also some general goals of the project, which helped define the direction the software development should take and what was important for the software to accomplish to be considered successful..

3.1 General Goals and Requirements

- Facilitate and encourage group collaboration on software programming projects over a local network and the Internet.
- Provide a complete and useful set of project development and communication tools to the developers.
- The client and server should both be portable, and able to run on many architectures. Main development was done on Linux, with additional testing on Windows XP, Windows 98, and Apple OS X.
- The programming language used is Java. Java 1.4.2 and Java 1.5.0-beta were used for the main development and testing environments, however compatibility with all 1.4.x or higher versions of Java was maintained. Specific features in jEdit 4.2 and ProCol both require Java 1.4.0 or higher to run, therefore compatibility with older versions of Java is not supported.
- This plugin requires jEdit 4.2 or higher. The plugin API has changed dramatically from jEdit 4.1 to 4.2 in order to support dynamic loading of plugins. The ProCol plugin takes advantage of this dynamic loading support, so it is not compatible with older versions of jEdit. At the time of this writing, the most currently released version is jEdit 4.2pre11. The estimated release date for jEdit 4.2final is at the end of May, 2004.
- jEdit does not have any official minimum system requirements, however the estimated recommended specifications for running jEdit with ProCol are a 233 Mhz Pentium II processor (or equivalent) with 96 MB of RAM. (Note, this may be higher depending on operating system requirements).

3.2 Client Requirements

- Users are able to connect to the server, select a project to work on, check out files, etc., all from the jEdit plugin interface.
- The plugin interface is compact and simple, and able to integrate well with the jEdit interface. Complex functionality is in the form of dialog windows separate from the main plugin interface.
- Common functions are quickly accessible from the plugin interface through the use of buttons. Less-used functions are put into drop-down menu items or context menus.
- The client maintains a tree of files which are able to be checked out. The tree indicates via bold text which files are currently checked out.
- The client is able to download copies of individual files or an entire tree snapshot without checking out the files.
- The client maintains a user list containing all the users currently logged in to the server. There are also be buttons representing the communication functionality of the server. These buttons 'light-up' in color when there is an unread message for the user.
- The client has a progress bar to indicate network activity. This is used in an indeterminate mode when the client is waiting for a response from the server, or a determinate mode when downloading data from the server.
- jEdit provides functionality for docking a plugin to the side, top, or bottom of the jEdit window. ProCol's user interface is designed to take advantage of this where applicable.

3.3 Server Requirements

- The server is able to run both in a command-line interface, in the case of servers without graphical capabilities, or from within jEdit, with output being displayed in a dockable status window.
- The server is able to be set up to allow a predefined set of users and passwords in order to preserve security of the project.
- Administration of the server is done by editing properties files, which define settings and are readable by the Java Properties class.

4.0 System Design

jEdit has some requirements for plugin design which had to be taken into consideration when designing ProCol. These requirements include interaction between the plugin interface and the rest of jEdit, implementing jEdit standards for the plugin interface and usage, and integrating user-selectable options, preferences, and help system with jEdit. However, overall the design of the plugin is largely separate from jEdit's requirements, so this did not present any major problems.

4.1 Client Architecture

The client is composed of three major parts: user interface, model, and networking facilities. The user interface shows the data on the screen. This includes the main ProCol window, showing the project files, user list, and various buttons. It also includes the various dialog windows, and windows which are displayed within jEdit's interface, such as the plugin options and the help interface.

The model section contains all the data used by the plugin. This includes the file tree, user list, messages, etc. The model and user interface communicate with each other using an observer pattern. The model contains all the data for the current session (project information, messages, etc.). The model can notify the interface of changes to itself, so the interface can update its representation of the information. jEdit allows the graphical interface to be unloaded and reloaded (either the client or jEdit's entire graphical interface), therefore the client can also query the model when it needs to be refreshed.

The networking section is the part of the client that communicates with the server. It handles messages received from the server, and creates messages and packets to send to the server and handled when received from the server. The networking section communicates with the model to send messages to the server, and update the client's model.

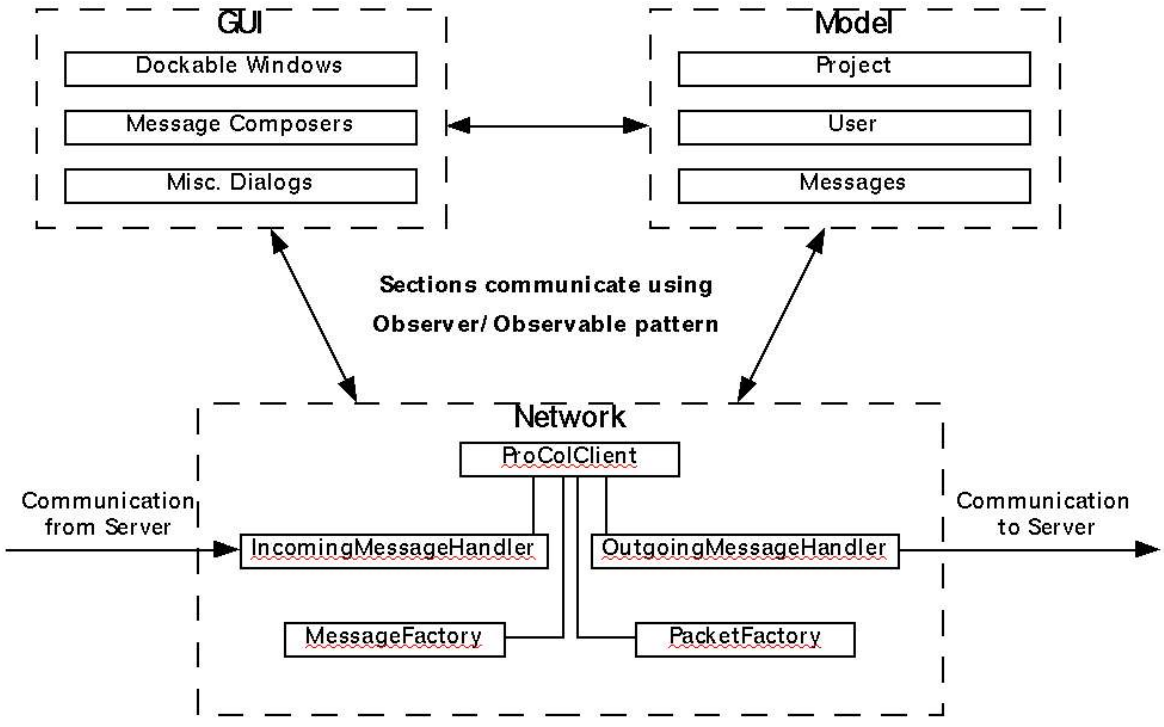

Figure 1 – Client Architecture Diagram

4.2 Server Architecture

The server is also composed of the same three major parts as the client – user interface, model, and network. Currently, the user interface is limited to displaying output of the server as it is running.

The model of the server contains all the information for all the projects hosted on the server. This includes the files, user messages, todo and bug lists, etc. The server will store information about projects on the server hard drive in a customized format. Since one of the goals of ProCol is to depend on as few external programs as possible, and be as quick and easy to set up and use as possible, it will not utilize a database such as MySQL for data storage.

The networking part of the server maintains the connections and communication to and from all the clients. It communicates with the model to get information from it or update it based on requests from the clients.

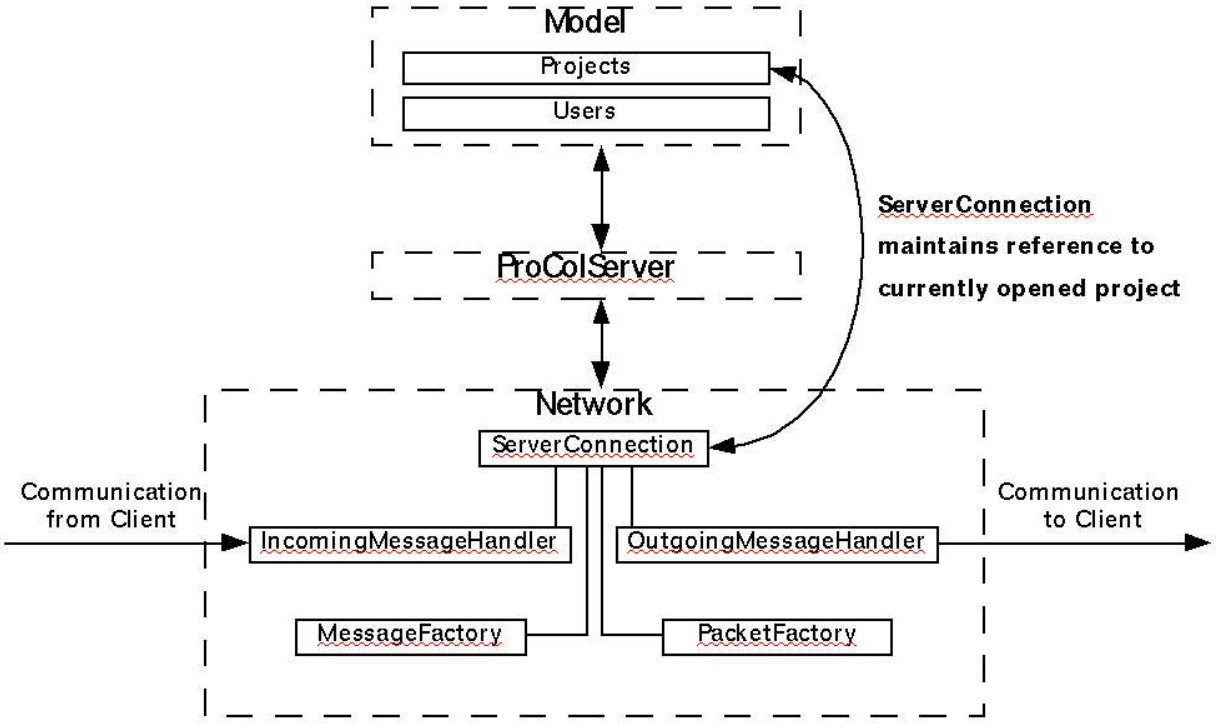

Figure 2 – Server Architecture Diagram

4.3 Client-Server Communication

4.3.1 Communication Overview

Network communication is a major part of the ProCol plugin. Networking is handled by several threads. There is a thread that watches for incoming connections, which then passes it off to a ServerConnection which contains a IncomingMessageHandler. There is also an ErrorHandler, so error messages are handled separately from the regular messages. This is mostly to separate and organize messages. This multithreaded design allows for concurrent connections to be handled efficiently.

When a client connects to the server, they first must authenticate with the server by logging in as a user. A client may only work with a project after being authenticated.

ProCol also utilizes anonymous SSL sockets for communication. This provides a quick and easy way to provide a moderate level of security for network transactions between the client and the server.

4.3.2 Communication Protocol

ProCol uses a custom network communication protocol that consists of a small header, and an optional data section. The default packet size is 2048 bytes. This may be set to any value (greater than the header size) on the server. Recommended sizes are between 1024 bytes and 8192 bytes, depending on the connection speed between the client and the server

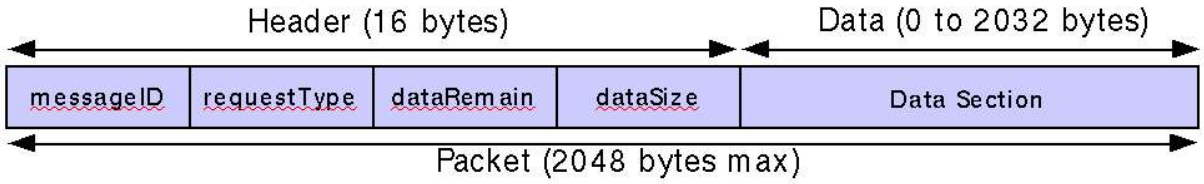

Figure 3 – ProCol Packet Diagram

The size of the header is fixed at 16 bytes, broken into four 4-byte ints.

The first header section contains the message ID, a unique identifier given to the message by the sender. This message ID is used to identify packets which belong to the same message.

The second header section contains the request code. The expected format for data section of the packet is dependent on the request code, and is defined with the request code in the ProCol API documentation for the RequestType class.

The third header section contains the data size that follows. This number can be any number from 0 (indicating there is no data with the packet) to MAX_PACKET_SIZE-HEADER_SIZE (indicating a full data section).

The last header section contains the number of bytes remaining in the message's data section, including the bytes in the current packet. In the case of a message that only requires one packet, or for the last packet in a message sequence, this number will be the same as the data size section.

Following this is the data section. The size of the data section can be any amount from 0 bytes to the remainder of the packet size. In the case of a 2048 byte packet with a 16 byte header, the maximum data section size is 2032 bytes. The particular format of a data section varies by request type. If the data size is equal to the maximum data size, the packet is part of a multi-packet message. If the data size is less than the maximum data size, then the packet is the last packet for the message series. If a message's data causes the last packet to be exactly the maximum data size, an additional packet with an empty data section will be sent to direct the client to terminate the message series.

Since ProCol communicates over TCP, packets are guaranteed to come in order. However, the messageID is needed because messages may be interrupted to send higher-priority messages, and these messages must be able to be distinguished from each other. By default, there are three priorities: low, normal, and high, and the order packets are sent in are determined by the server or client that is sending the packets.

4.4 Graphical Interface

Prototyping was used to develop all of the non-trivial graphical interface windows. Mock-ups were drawn on a whiteboard and design ideas worked out using NetBeans before the final interface was developed in jEdit. Since I had a clear idea of what I wanted the interface to look like, and because of my use of prototyping, I did not have to make many modifications to the interface components once they were created.

Figure 4 shows the final interface developed for the ProCol client. At the top are the client control buttons. Next is the project file tree and the file check-out and check-in buttons. Below this is the user list and communication buttons. Communication button icons are displayed black-and-while when there are no new messages, colorized icons indicate that new messages are available. Below this is the progress bar.

Figure 5 shows a typical jEdit session, with the ProCol interface docked to the right-hand side of the window and the Bug List docked to the top of the window.

Figure 6 shows the private message dialog, which is typical for the communication and project management dialogs.

Figure 7 shows the bug item composer. This was a particularly complex dialog, and relied heavily on prototyping to get the layout set before final implementation.

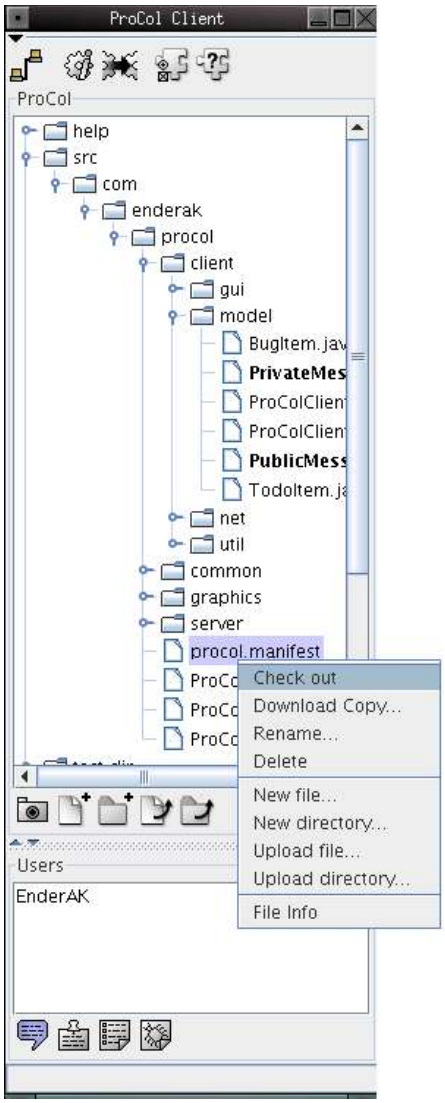

Figure 4 – ProCol GUI

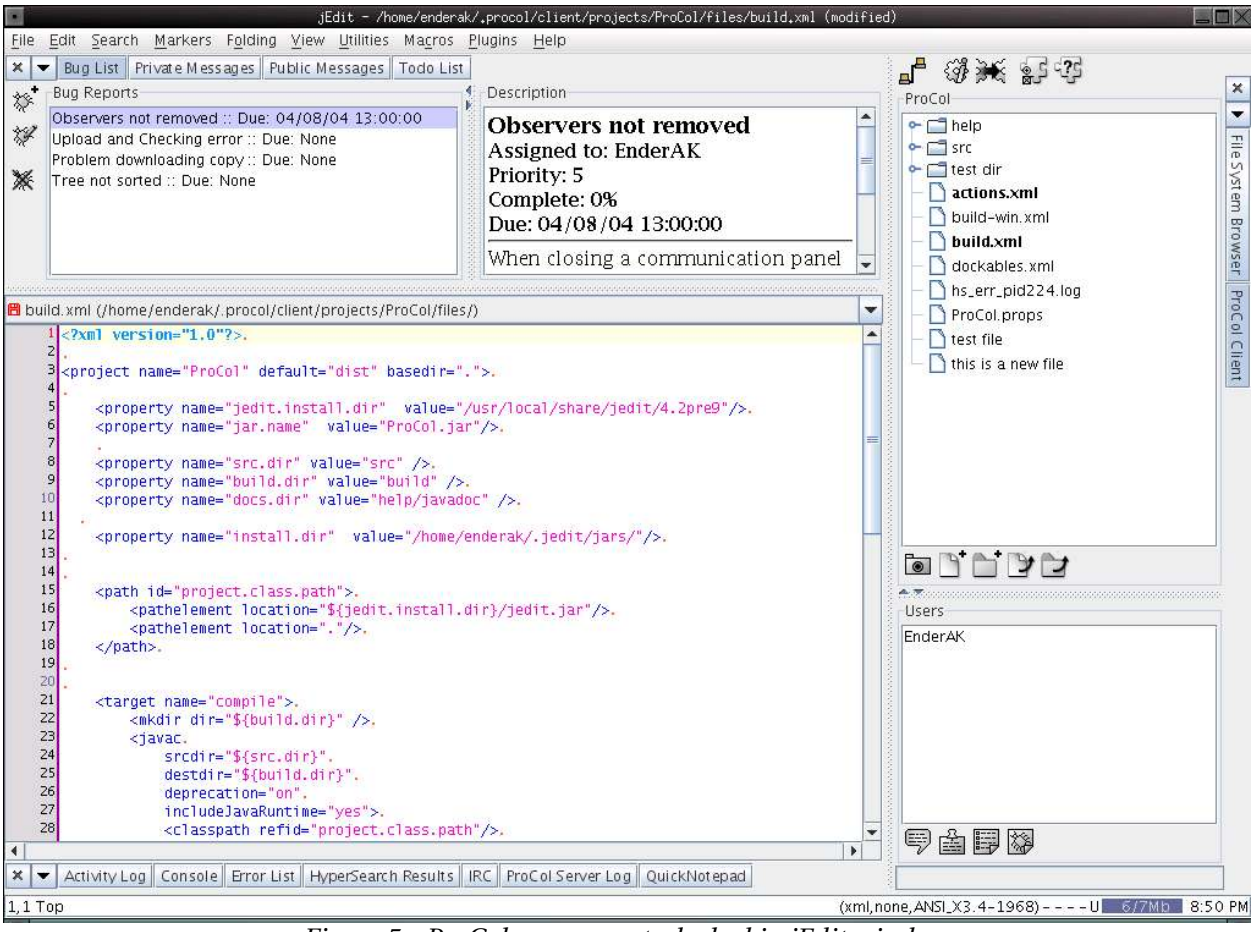

Figure 5 – ProCol components docked in jEdit window

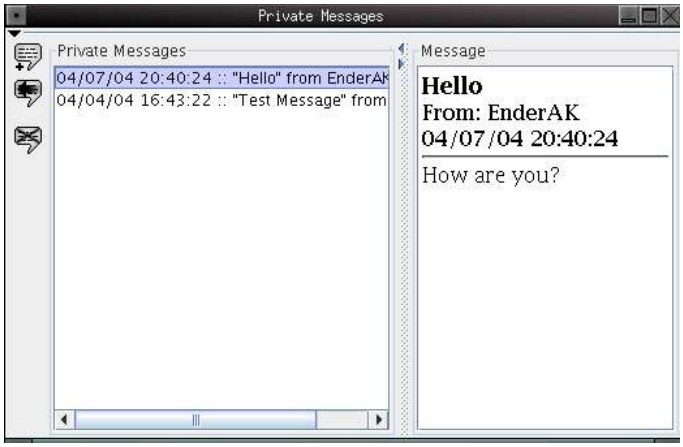

Figure 6 – Private Message Dialog

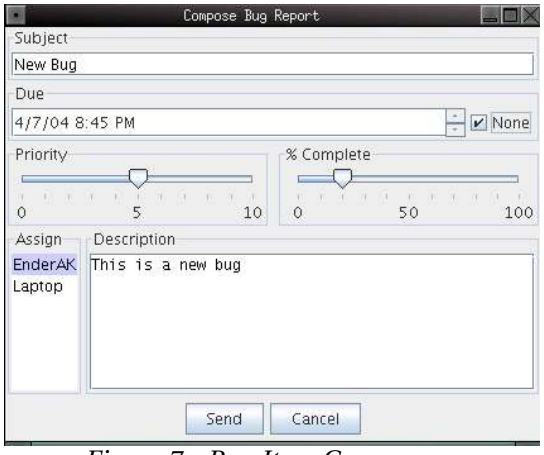

Figure 7 – Bug Item Composer

5.0 Development Process

Due to the short development time and large amount of work needed to bring this software to a usable level, the majority of time was spent in the actual implementation and testing of the software. The estimated time to be spent on the project is shown in the chart, Figure 8. Since much of the requirements, design, and prototyping was already completed in anticipation of this project, the numbers in the chart reflect the amount of work done during the duration of the semester, and does not take into account previous work.

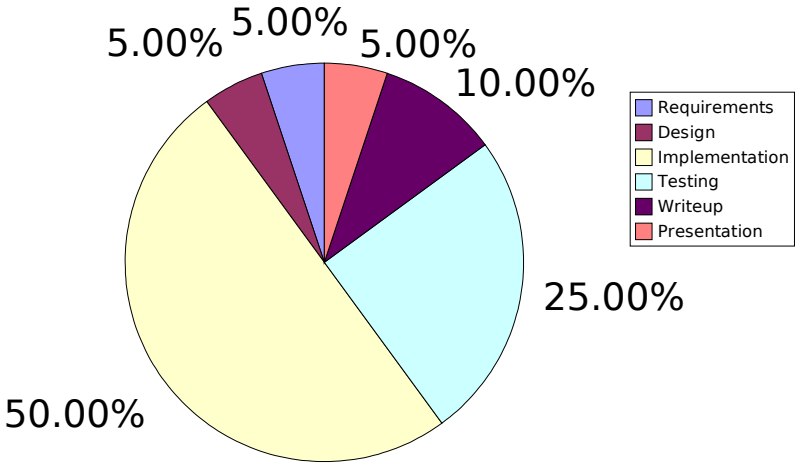

Figure 8 – Estimated Time Spent

Time dedicated to this project varied from week to week, but on average was about 2 hours per weekday and 8 hours per day on the weekend, for a total of approximately 26 hours per week, on average. The estimated total time spent designing, implementing, testing, and documenting this project is about 350 hours.

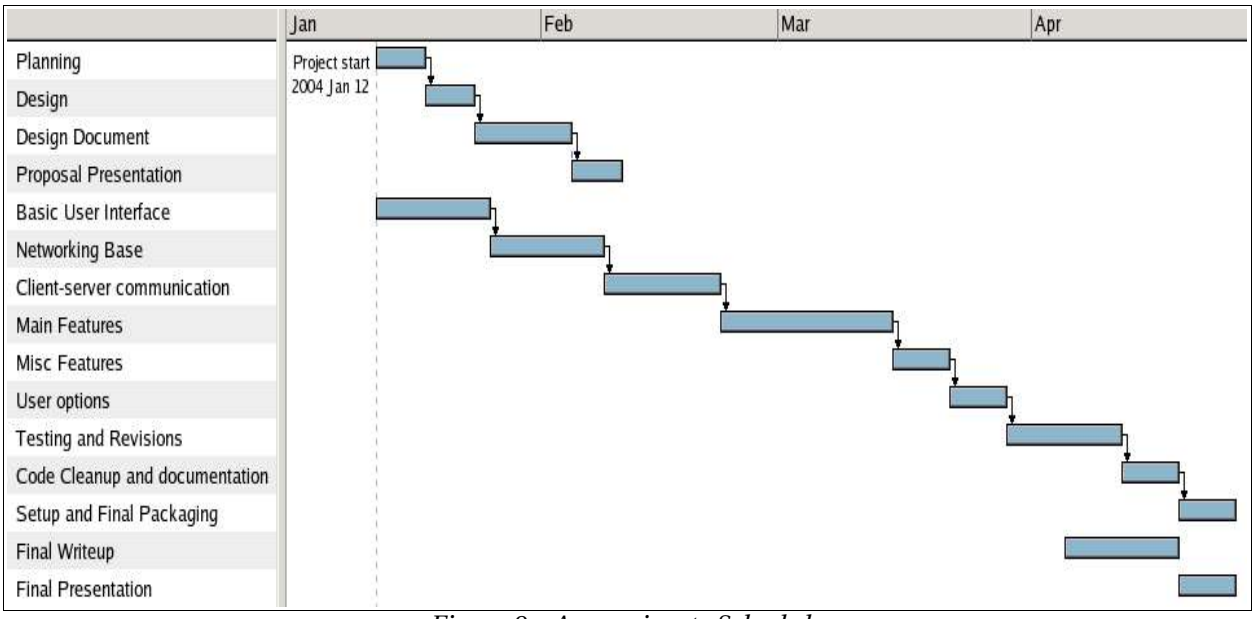

Figure 9 – Approximate Schedule

Figure 9 shows the schedule I set at the beginning of the semester. I was able to stick to the schedule fairly closely, though some tasks did take slightly shorter or longer, and some tasks were done in a more concurrent fashion instead of one after the other.

6.0 Results

6.1 What Worked?

I spent a large percentage of the development time working on the network communication between the client and the server. I used packets and priority queues to allow low-priority messages to be interrupted to send high-priority messages. Anonymous SSL security was used to encrypt network transmissions, so protect information such as passwords and source files to be intercepted by a third party. Anonymous SSL is still vulnerable to the "man-in-the-middle" attack if the third party intercepts the public key as it is first transmitted to the authorized person, however it provides a good level of security without having to store keys with a third-party certification authority.

Also on the topic of networking, there were several problems that came up communicating between Windows and Linux systems due to the different representation of files on disk. This was fixed by using URI's to represent the files when that information was transferred over the network. URI's were something I had not used before, but they proved to be very useful with respect to the network communication.

Secondly, jEdit's dockable windows were used heavily to provide a plugin that integrates well with jEdit. All major windows have the ability to be used as normal windows or to be docked within jEdit. Graphical components also configure themselves appropriately depending on if they are docked on the sides (vertically) or on the top or bottom (horizontally).

Lastly, jEdit makes heavy use of the Java Properties class to store settings and program information such as error messages and label text. I did this with ProCol as well, for everything from labels and buttons on the Client to storing messages and changelogs on the server. Java uses hash tables for working with Properties files, so the access time in searching for items is quite fast even for large sets of data. Making use of Properties files on the server for project information also alleviated the need to utilize a database, such as MySQL, which makes setting up the Server much easier, which was one of the goals set out for the server at the beginning of this project.

6.2 What Did Not Work?

Soon after starting, I decided that I wanted to use Java's new I/O (NIO) API for networking. This seemed to be good in theory, but it seems that it is badly implemented by Sun at this point. There were many problems that I encountered that ended up being problems in Sun's code, and I was unable to work around many problems. Unfortunately, this resulted in approximately 2 weeks being wasted with trying to troubleshoot and rewriting code. Eventually, I had to rewrite large portions of code in order to revert back to using Java's classic networking.

One problem that I knew I was going to have from the onset was that I wanted to implement too many features, and I would not have enough time during the class to complete everything. Knowing this, I cut many features out while making my requirements. In the end, all my required features were implemented, but some ended up being rather basic.

6.3 Was the Project Successful?

There were several things that I considered when deciding if this project was successful or not. These included having a usable software program that satisfied all requirements.

I consider this project to be successful, because all requirements were satisfied and the program proved to be usable during testing. In addition, there has been lots of interest expressed by the jEdit community, and they are looking forward to seeing it when it is released. ProCol will be released under the GPL and development may be continued in the future.

7.0 Conclusion

The program I developed is a plugin for the Java-based jEdit text editor that facilitates network communication between users for collaborative project development. It was developed in Java with the goal of simplifying user collaboration for small to medium-sized projects.

7.1 What Did I Learn?

I learned several new things about programming in the process of developing this software. I learned about several areas of Java that I had little or no experience in before, including SSL, New IO (NIO), Threading, the Java Properties class, and using URI's to reference files in a cross-platform networked environment. I also learned many new techniques for the design and implementation of graphical interfaces using Swing.

I was able to make use of several design patterns that I had not used extensively before. These include using observers and observables to pass information around the different parts of the program. I also used a Model-View-Controller (MVC) design for the client plugin. Lastly, I made significant use of Factories for networking in both the client and server.

In working with jEdit, I learned a lot about programming a plugin, something I had never done before. In addition, I learned a bit about Beanshell programming, the scripting language jEdit uses. Also, I learned how to use Apache Ant to automate compiling and packaging a program. Apache Ant is used by jEdit and most plugins in order to automate build the code from source.

8.0 References

8.1 Other Contributors

ProCol was developed in whole by myself, however several members of the jEdit community gave me guidance with respect to building a jEdit plugin. In particular, the main developer of jEdit, Slava Pestov, gave me much guidance on how to properly interface ProCol with the jEdit plugin API. In addition, he has made several fixes and enhancements to jEdit itself that came up during the development of ProCol. The end result of this was that both ProCol and jEdit were able to be developed into better software than they would be otherwise.

8.2 Documents Referenced

8.2.1 Websites

- Pestov, Slava & Gellene, John. jEdit User's Guide – Writing Plugins, http://www.jedit.org/users-guide/writing-plugins-part.html
- Pestov, Slava. What's New in jEdit 4.2, http://www.jedit.org/42docs/news42/index.html
- Pestov, Slava, et al. jEdit 4.2 API Documentation, http://www.jedit.org/42docs/api/index.html
- Sun Microsystems, Inc. Java 1.4.2 API Documentation, http://java.sun.com/j2se/1.4.2/docs/api/
- Sun Microsystems, Inc. Java 1.5.0 API Documentation, http://java.sun.com/j2se/1.5.0/docs/api/
- Sun Microsystems, Inc. Java Secure Socket Extension Reference Guide, http://java.sun.com/j2se/1.4.2/docs/guide/security/jsse/JSSERefGuide.html

8.2.2 Books

- Cooper, James W. Java Design Patterns: A Tutorial. Addison Wesley, January 2000.
- Harold, Elliotte Rusty. Java Network Programming, 2nd Ed.. O'Reilly, August 2000.
- Hitchens, Ron. Java NIO. O'Reilly, August 2002.
- Oaks, Scott. Java Security, 2nd Ed.. O'Reilly, May 2001.

9.0 Other Documentation

There is other documentation for ProCol available online or included with ProCol. This includes the User Guides for the Client and the Server as well as the ProCol API documentation.

To open the help interface for ProCol from within jEdit, click on the help button in the top button bar of the ProCol Client window. This will display jEdit's help interface and load the ProCol help pages. Alternatively, you may go to `Help>jEdit Help` in the jEdit menu bar, or press F1, to load the jEdit help, and then select `Plugins>ProCol` in the contents tree.

For the latest information, downloads, and documentation, please see the ProCol homepage at http://www.enderak.com/procol/