Project Number. GFP0701

Webfoot VNC Collaboration

A Major Qualifying Project Report:

submitted to the faculty of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

by:

_____

Kenneth Jay Breeman III

_____

Eric Michael Griffel

April 23, 2008

Approved:

_____

Professor Gary F. Pollice, Major Advisor

1. Collaboration
2. VNC
3. Eclipse

Abstract

Although collaboration is a vital component of the modern software development process, it poses one of the greatest challenges to software developers. Many developers regularly find themselves struggling with tools that are either difficult to use or insufficient for their needs. To alleviate this obstacle we have built open source software to allow a user to share their user environment with another user in order to better communicate.

The result of the project is an Eclipse plug-in that implements Virtual Network Computing (VNC). By utilizing the plug-in, one user can instantiate a server that another user can connect to. Once this is done, both users will be able to see the environment of the server machine.

Acknowledgements

Kenneth Breeman – team member.

Eric Griffel – team member.

Prof. Gary Pollice – project advisor.

Tal Liron and the rest of developers who have contributed to the VNCj project – for providing us with a flexibly licensed open-source project to improve, extend, and build upon.

Table of Contents

List of Illustrations

List of Tables

1. Introduction

The Webfoot VNC Collaborator allows people to collaborate with each other over the Internet by sharing keyboard, mouse, and video in an integrated environment. Webfoot is WPI's environment built for object-oriented teams and is used to help people collaborate more effectively through Eclipse plug-ins. Eclipse is a popular integrated development environment from IBM. VNC is Virtual Network Computing, a protocol for sharing control of a computer system's keyboard, mouse, and display. Webfoot VNC Collaborator makes the process much more elegant through ease of use and integration.

The idea was initially conceived during a Software Engineering class with Professor Pollice and has grown into a complete project. Pair programming is a common practice in Software Engineering, but weather, schedules, and a variety of factors can infringe on time to pair program in the same physical location. To get around this, we used an existing tool, Virtual Network Computing (VNC), to work together. This solution was not elegant, but it was functional and we were able to collaborate without being in the same physical location.

VNC was originally designed as a piece of remote administration and remote control software. It involves a light weight client connecting to a server and passing messages back and forth over a single socket. Through the course of this project we became involved with people from the LGPL VNCj project, an open source VNC library written in Java[10]. The scope of our project widened to encompass improving VNCj as well as creating the Eclipse plug-in for Webfoot.

Eclipse is an open source development environment from IBM which has a rich plug-in interface. Its primary functionality is for writing programs but has been expanded to be

able to do much more. The Webfoot plug-in expands the collaboration capabilities of Eclipse in numerous ways[18]; the VNC Collaborator is just one of them.

Collaboration involves people working together towards a common goal. As the economy evolves and teams become geographically separated, distance becomes a major factor. Numerous tools exist to improve the collaboration process in a variety of ways, but no single tool does everything. In order to collaborate effectively over distances multiple tools are required. The Webfoot VNC Collaborator integrates several tools into one simple and easy to use interface and enables users to collaborate like never before.

The current state of collaboration software is discussed in the Background section. This is followed by how we went about creating this software and why we chose to do it that way in the Methodology section. The Results and Analysis section details the outcome of the project. Finally the Conclusions and Future work sections discusses the final state of the project and where it can continue to grow.

2. Background

There are numerous methods of collaborating and even more tools available which attempt to make the process easier. Previously, Webfoot has been a collection of plug-ins to interface with SourceForge[18]. SourceForge is based around source control and uses a task based approach to collaboration. The Webfoot VNC Collaborator breaks this mold by allowing users to collaborate directly without the need for a separate server in between.

Eclipse is an open source tool from IBM which includes support for synchronizing with CVS code repositories. Eclipse also supports plug-ins to work with SVN and other types of repositories. The biggest issue with this model is it lacks the ability to collaborate in real-time. One developer writes code and submits it, another developer writes code and submits, and whoever submits second needs to worry about merging any conflicting code. Commit comments are a nice feature, but they are not always used and may not convey enough information for any other developer to merge cleanly with it.

Eclipse is not always consistent across operating systems. Consistency is an important part of collaboration. Working with someone who is seeing something completely different while working on the same code, something we came across numerous times while working on this project, can be very frustrating. Eclipse does not have any available plug-ins for resolving these kinds of issues.

The Eclipse Platform is moving towards a more universal audience, projects are already underway to try and implement an e-mail client plug-in as well as plug-ins for

various instant messaging protocols[2]. Eclipse is actively trying to expand its functionality.

SourceForge provides a web interface as well as source control in a single centralized server based solution[12]. Clients can connect using a standard web browser and work with tasks, track issues, view source code, manage documents, discuss things in a forum, view and contribute to a wiki, generate reports, and create releases. SourceForge allows users to integrate these features closely with their source control. SourceForge is extremely useful for managing an open source project and is also what we chose to use while writing this project. Existing Webfoot projects are closely integrated with SourceForge as well[18].

Jazz is another development environment platform, similar to the Eclipse platform, but with much more focus on collaboration[19]. It focuses on team and process based collaboration and communication. At the time of the writing of this paper it is still in beta, but it appears to incorporate many of the features Eclipse and SourceForge already provide into one tool. The Webfoot plug-in for Eclipse helps to bridge the gap between Eclipse and SourceForge and provide some of the functionality Jazz is attempting to do.

Jupiter from the University of Hawaii's Collaborative Software Development Lab is a code review tool which aids users in reviewing code using checklists[20]. It helps coordinate and organize the process. However it lacks the ability to have multiple people review code at the same time without being in the same place or using additional tools.

Collab.net and Tigris.org both provide similar functionality to SourceForge and compete for users[21, 22]. These two focus mainly on further development of collaboration software. Currently neither of these has as many features as SourceForge.

Virtual Network Computing (VNC) uses the Remote Frame Buffer (RFB) protocol to allow users to share keyboard, video, and mouse across a network connection. VNC and RFB have been around since 1999 and has become one of the defacto standards for desktop sharing. It has a variety of applications including high end KVM's and technical support solutions[1]. The current RFB specification is on revision 3.8, however some clients report version 4.0 and provide additional unofficial extensions to the protocol that are still backwards compatible[11]. Numerous open source and closed source solutions exist[8]. TightVNC and RealVNC are among the more popular implementations. One of the biggest issues with existing implementations is that they are OS specific and do not include hooks to extend functionality[8].

VNCj is an open source VNC implementation under the LGPL license. It is written in Java and prior to our contributions was mostly a dead project. It was written for an older version of Java and was not functionally complete. It did however include easy to use interfaces and hooks with which we could build upon and extend.

3. Methodology

Since there are numerous software tools to assist in collaboration, we started out this project by researching existing solutions to find a niche that had not been filled. The biggest gap we found was that most collaboration solutions were not real-time and interactive. The few that were, were limited to just a whiteboard and text chat, this is not sufficient for developers who use their entire system to develop software. VNC was the only tool we found that allowed for real-time collaboration.

VNC has been implemented on numerous platforms, but no single distribution was supported completely cross platform, and none of them included hooks to extend with. Licensing issues also posed a problem as very few solutions were completely open source and licensed flexibly. The protocol is published and supports numerous extensions while still maintaining backwards compatibility. This is done through a handshake process where the client and server determine how many different extensions they support.

Rather than re-inventing the wheel, we decided to look for open source solutions which we could build off of. Many of the distributions were not fully open source, and the few we found that were, were not built well and did not have a simple way to extend them. The one project we did find appeared to have been mostly abandoned and was not functionally complete. We decided that it was better than nothing and became contributors to the LGPL VNCj project. This allowed us to improve both projects at the same time while still maintaining compliance with the licensing restrictions.

As the first term of the project came to a close, we had completed most of our research and decided to begin working on the application. Our project had grown into

two projects now and we had a lot of work ahead of us. We decided to use an iterative incremental approach rather than waterfall. This allowed us to track our progress much better and prevent major design issues throughout the project. We held weekly team meetings as well as weekly check-ins with Professor Pollice and the rest of the Webfoot teams. This provided excellent feedback as we went and allowed us to steer our project more effectively.

To prevent ourselves from falling into a waterfall methodology, we began working on each part of the project so that we would always have working software. We did spikes to create an initial graphic user interface (GUI), we did some initial work on the client and server as well. With the groundwork done we decided to focus on improving the server side first. We planned on completing this portion by the end of the term.

We fell short of our goal to complete the backend code by the end of the second term. The VNCj project was simply too old and unmaintained and had caused a great deal of slow downs. We decided the best plan of action would be to continue working with VNCj since we had already invested so much time into it, but we acknowledged that our end result would be lacking some of the more unique features we had initially hoped to implement.

Once we had the server side working well, we tested it with all of the major VNC clients we could find including RealVNC, TightVNC, UltraVNC, and the integrated MAC client as well. Since the protocol is published and each of these clients followed it, we had a clear view of where our implementation succeeded and where it fell short. The biggest area we fell short in was that RFB supports incremental updates of small portions of the screen. This process saves bandwidth over the network, and depending on the

implementation can save CPU time as well. However since this was written in Java, we had a great deal of difficulty computing the image differences in a reasonable amount of time to provide any sort of speed up. We decided to work around the problem by sticking with full screen refreshes at a slightly reduced frame rate. This worked well and greatly reduced the CPU usage on the server end.

Now that the server end was complete we began work on the client end. Much to our dismay we found that VNCj had not fully implemented the encodings for image compression. The server side was able to encode into numerous encodings including Raw, CopyRect, Rise and Run, and Hextile; but was unable to decode them. This proved to be a time consuming endeavor to implement in Java efficiently.

The Hextile encoding is a variation of the 'Rise and Run' encoding which is very efficient in terms of memory, processor, and network usage. It is also the most difficult encoding we had to implement for the RFB protocol. It involves breaking an area of the display into a tile consisting of sixteen pixels by sixteen pixels. Each tile is then encoded using either Raw or RRE depending on which is more efficient. There are numerous external variables that need to be taken into account when encoding and decoding Hextile including placement of tiles within an area, previous background colors, and previous foreground colors. Hextile is now widely considered to be the standard encoding for most VNC implementations with many supporting further proprietary extensions such as Tight and ZLIB. Given more time this project could have encompassed those as well but the benefit of doing so versus the time required to do so was not reasonable.

As the end of the third term drew near, we had a break through and were able to improve the algorithmic efficiency by a factor of five. This reduced the CPU usage to a

reasonable ten percent, even on older computer hardware. With a fairly solid client and server implemented, we still had a GUI left to write and were running out of time.

We applied for an extension to the project and were granted it. We used this time to implement a GUI; we used the Eclipse Platform and the Standard Widget Toolkit (SWT) to write our GUI. This proved to be far more difficult than its counterpart the Abstract Windowing Toolkit (AWT) since SWT does not have a simple way to draw images to the display4. With the client, server, and GUI implemented, we only had a few bugs left to iron out. Due to delay in merging for the beta release of Webfoot, we were able to fix many of the bugs in our software including issues with the keyboard between Windows, Linux, and Mac platforms.

Memory leaks are a sign of poorly written software. When we discovered a linear increase in memory usage over time we knew something in our code was not being de-allocated properly and needed to be fixed immediately. Java has built in garbage collection so it had to be something graphical in nature which was not being disposed. Due to the complexity of SWT as well as its lack of basic components such as a displayable image, we learned a great deal about how SWT handles graphical resources. Our implementation of an image canvas had been replacing the front and back buffer images as refreshes occurred, however they were not being disposed from the SWT's point of view. This was what was causing the memory leak and was quickly corrected by calling the appropriate dispose method before overwriting the pointer.

We used Eclipse extensively throughout this project. Eclipse has tools for creating plug-ins and is a high quality IDE for developing Java in. We utilized the existing Webfoot and SVN plug-ins to work with SourceForge for our project. The Wiki feature

of SourceForge was especially useful for keeping a journal. As our plug-in became feature complete and stable, we were even able to use it to aid in further development. The ability to develop an Eclipse plug-in from within Eclipse was a great aid in the development process.

In addition to iterative testing after changing the software, we utilized a variety of debugging techniques to create our plug-in. We used the Eclipse Console and print statements for basic progress evaluation and value print outs. For more complex issues we used the Eclipse debugger which is able to debug an entire instance of Eclipse, from within another Eclipse session. The breakpoint feature was especially useful for debugging multiple threads at once. This proved useful on many occasions and allowed us to fix errors and find the source of problems quickly.

One issue we encountered while expanding the functionality of the plug-in was managing multiple clients on a single system. Each editor window included an instance of the VNC client it was utilizing, but multiple editor windows and multiple clients become cumbersome to manage. We utilized the Factory pattern to handle creation and management of VNC client sessions. This allowed us to prevent users from opening multiple client sessions to the same host session, a bug which could lead to plenty of confusion. Another advantage to this implementation is that when Eclipse is closing, the VNC Session Manager can very simply close all sessions in a simple and efficient manner.

Performance was another major issue we encountered. Each refresh cycle within our program requires the server to take a screenshot of the desktop, compress it using a Rectile Encoding, send it over the network, decode it on the client end, convert it to a

displayable image using SWT, and then display it to the user. Each of these steps takes a significant amount of computation time to run and required optimization to make the software useable. Since we were using a combination of old computer systems running AMD Athlon 2100+ as well as newer systems running the latest Intel Core 2 Duo processers we had a wide range to test with. We used the Eclipse Test & Performance Tools Platform Project (TPTP) to evaluate our software's performance. We found several bottlenecks, the most obvious being the complex image encoding process, as well some unexpected ones such as the amount of time spent blocked waiting for network resources to become available. We repeatedly tested these portions of the software and made improvements through algorithm optimizations as well as simple read ahead optimizations. This improved the performance of our software significantly. To allow users to tweak performance even further, we exposed a setting in the preferences dialog which allows users to specify the refresh rate per second.

To ensure our code actually worked, we performed a variety of tests periodically to find bugs and prove we had working software. The client and server portions of the code base were tested independently using numerous other implementations of VNC clients and servers. This tested whether or not our implementation complied with the RFB protocol. These tests were executed on Windows, Mac, and Linux platforms to ensure we had a solid back-end for our project. Once these tests were completed, we performed end to end testing of the entire software by running it in various version of Eclipse available to us through our personal systems as well as CCC lab machines here at WPI. Automated testing would have been a nice convenience throughout this project; however software such as this is difficult to test in an automated fashion. By following a strict code practice

of testing before committing code changes we were able to maintain working software and a clean, working build.

Before releasing a beta version of our software, we had to carefully consider licensing issues. We researched individual licenses as well as multi-license legal issues. We found that the LGPL and the EPL were compatible licenses and this has been supported in prior court cases. Since we were already contributors to the VNCj project, all code modifications we performed were automatically committed back to the VNCj project through a separate branch in their source control repository. To complete our license compliance we insured that all appropriate license statements were at the top of each file as needed and that proper credit was given where credit was due. A copy of both the LGPL and the EPL can be found in the appendices as well as packaged with our software.

Keeping future developers in mind as well as for our own sanity, we made use of the Java Documentation (JavaDoc) notation for our comments. Each class contains a comment detailing how the class should be used and what the idea was when writing it. Each method has an overview of what each of the arguments represents, how the method works, what it returns, and any exceptions that may occur. Our code also includes extra comments detailing specific ideas, algorithms, and variable meanings.

For project presentation day, we decided to take a different approach to the traditional Power Point presentation. We used a presentation style similar to the one used by Dick Hardt at OSCON 2005[7]. This involved using simple slides with only a few words on each, in rapid succession to get our point across. We substituted images in wherever possible to get the point across in a faster, clearer, and usually more humorous way.

Project presentation day arrived and we were well prepared. We delivered a decent presentation and included screenshot demonstrations of software from the entire Webfoot team being used together.

4. Results and Analysis

The goal of our project was to build an Eclipse plug-in that allows users to share their user environment with another system. We were able to achieve that goal and deliver a functional plug-in at the end of the term. The plug-in had all of the core features that we intended to include as well as being compatible with similar tools.

Our tool consisted of three significant components. The first component is the Virtual Network Computing (VNC) host. At the start of the project the development of the host was already started, however it was incomplete and still non-functional. We managed to complete the functionality of the host by implementing the remaining classes and methods.

The second component was the VNC client. We built the necessary classes and implemented the methods required for communication and decoding necessary for the client. Our results exceeded our goals for the number of functional image compression techniques.

The final component that we built was the Eclipse plug-in. We successfully managed to build an image viewer that displays in an editor window, allowing a client to connect to multiple hosts at the same time. Additionally, we managed to interface with Eclipse thoroughly and use the standard Eclipse framework to build the plug-in in lieu of constructing quantities of our own tools.

Our project turned out to be successful within the restrictions provided by the Java Virtual Machine (JVM). The JVM has a rigid security model that prevents developers from gaining direct access to the desktop environment. This provided significant speed

restrictions regarding the project. The VNC protocol allows for incremental updates to specific regions of the screen. Since the JVM only provides a generalized interface to the desktop, this feature is unavailable. As a result, there is no efficient means of determining which section of the screen has been updated and a full screen redraw is required each update. Additionally, since the JVM emulates a virtual system, less processing power is available to the system and causes the plug-in to utilize more resources and run more slowly.

Despite the challenges, however, the plug-in has a refresh rate close to our desired goal on most systems. The resources utilized by the plug-in are lower than we were looking to achieve. Since we managed to implement more encodings and decodings than expected, the network utilization is also lower than we expected it to be.

There are only a few defects in the plug-in. Currently there is a chance of the plug-in terminating Eclipse when the host and the client are run on the same system. This defect is not known to occur when either the host or the client is run on another system, however. Additionally, on occasion the disconnect option from the menu fails to act and it must be selected again to function properly. This is not the same functionality as when there is no VNC Editor selected. The disconnect option should only terminate the active editor window if it is a VNC Editor. Otherwise it will have no effect.

Metric	Total	Mean	Standard Deviation	Maximum
Total Lines of Code	5316			
Number of Classes	51	5.667	4.19	16
Number of Methods	342	6.706	7.416	32
Depth of Inheritance Tree		1.627	1.171	6
McCabe Cyclomatic Complexity		3.046	10.08	180
Efferent Coupling		2.889	2.378	8
Instability		0.592	0.332	1
Lack of Cohesion of Methods		0.322	0.343	0.902
Abstractness		0.104	0.12	0.3
Weighted Methods per Class	1127	22.098	37.114	247

Table 1: Metrics

The majority of the metrics for our project[14], as demonstrated in **Error! Reference source not found.**, lie within reasonable bounds. The lines of code, number of classes and number of methods are not excessively large quantities. The averages for the number of classes (per package) and number of methods (per class) indicate a fair distribution among the various packages and classes.

The Depth of Inheritance Tree metric indicates objects' distance from the Object class in the inheritance tree. A high score is undesirable because any changes that need to be made to a class are likely to affect many other classes which are extended from it. in accordance with the KIS (Keep It Simple) philosophy, we avoided extending classes unnecessarily.

The McCabe Cyclomatic Complexity rating indicates the number of branches through sections of code. This is a difficult metric to keep low for our plug-in because of the necessity of passing through image arrays several times in order to initialize the images or compress them. We have, nonetheless, managed to keep this rating low throughout our

code. A substantial increase to this rating within our code is in a single method, the maximum case, which acts as a lookup table for key input.

The efferent coupling for a class is the number of methods within the class that depend upon methods from other classes. This is generally kept low to reduce the chance that changes to distinct classes can result in a change to the class. With a high coupling rating it becomes very likely that maintenance various sections of the project will have a significant effect upon unexpected sections of the code.

Instability of a class is the ratio between the efferent coupling of a class and its total coupling[15]. Afferent coupling, being a complimentary rating with regard to efferent coupling, is the number of external methods that depend upon methods within the class. By combining the two coupling ratings one can gain a general idea of not only how unpredictable changes to the class might be, but also of the likelihood that a change will occur to the class without modification of any of the source code of the class itself. This is aptly denoted "instability" since high ratings in a project indicate that even small changes to the code can have a profound impact throughout the rest of the project.

The Lack of Cohesion of Methods metric measures the Cohesiveness of a class, which is a gauge of how well a class adheres to one singular duty. If a class has a high cohesion rating, maxing at a rating of 1, then the class should likely be split into several other classes. This rating helps determine if programmers have split the tasks into appropriate classes or whether they simply massed all of the functionality into one singular class. This rating remains low for the majority of our classes and packages.

The Abstractness metric indicates the portion of interfaces and abstract methods declared in a package. Packages with high abstractness contain few implemented and

concrete classes. This rating, when combined with the instability metric, can provide a useful analysis of the project. It is undesirable to have both ratings very high or very low within a package. If a package is very instable as well as being very abstract, it is often a sign that it is serving a comparatively insignificant purpose. Likewise, if a package is very stable but not very abstract, then it means that it is being used heavily while not being easily maintainable. Throughout the majority of our code we managed to remain within the desirable region of highly abstract and highly stable or not very abstract and relatively instable.

The final metric, weighted methods per class, is the summation of the cyclomatic complexity of methods within the class. This value is generally kept low. As mentioned previously, the singular class, acting as a lookup table, possesses a significantly higher rating than the rest of the classes, skewing the results upwards. For the remaining classes the metric produced low values.

Another means of evaluating projects aside from the metrics already mentioned is an analysis of the dependencies of the package and classes within the project. This also relates back to several of the ratings including instability and coupling. The primary requirement for a program is that there should not be any cyclic dependencies in packages.

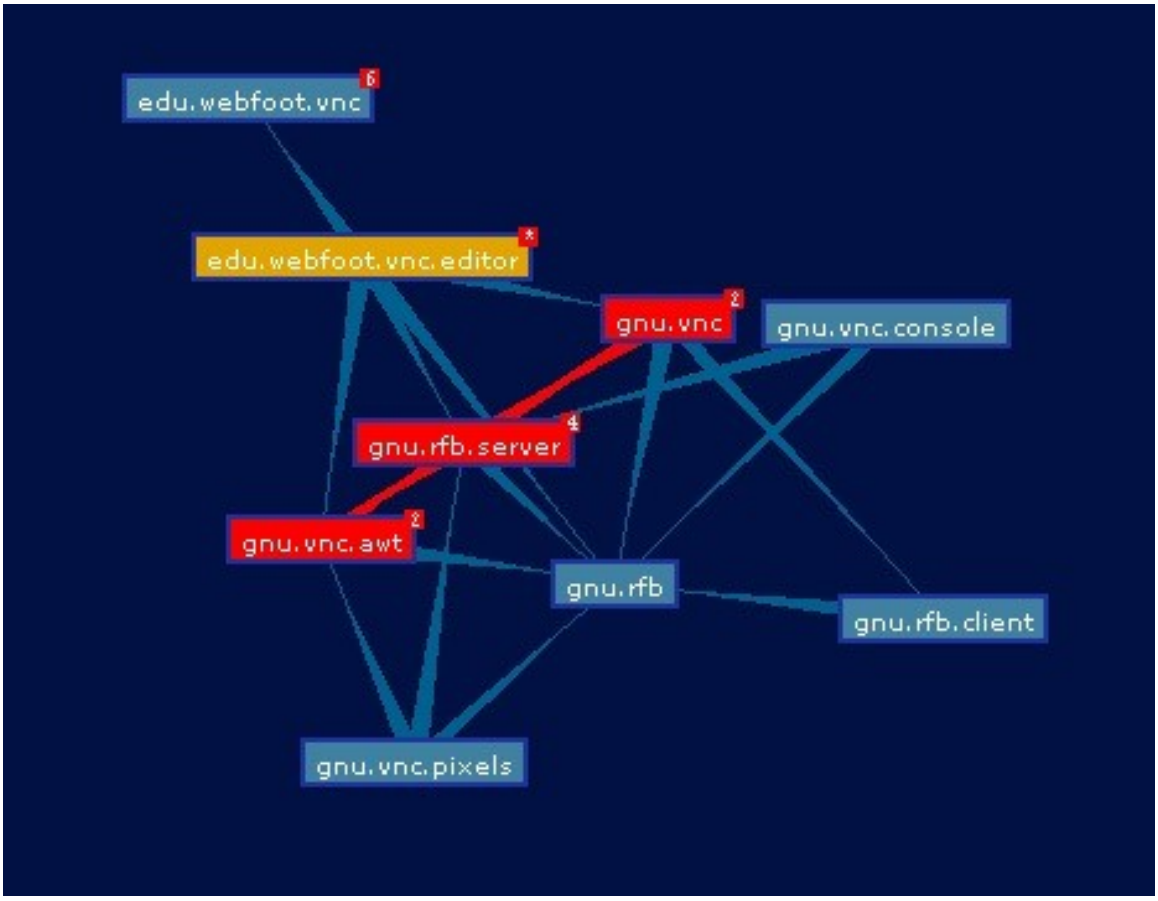

Figure 1: Internal Package Dependencies

Figure 1 outlines the package dependencies for packages within our project. The edges between nodes form triangles that point with the thinner end at a package that another package at the base of the triangle, the thicker end, starts at. The graph demonstrates that the package gnu.rfb.client depends upon the package gnu.rfb. This can be of use in understanding the architecture of the project. Figure 2 expands upon this by demonstrating the total dependencies of the project, both internal as well as external.

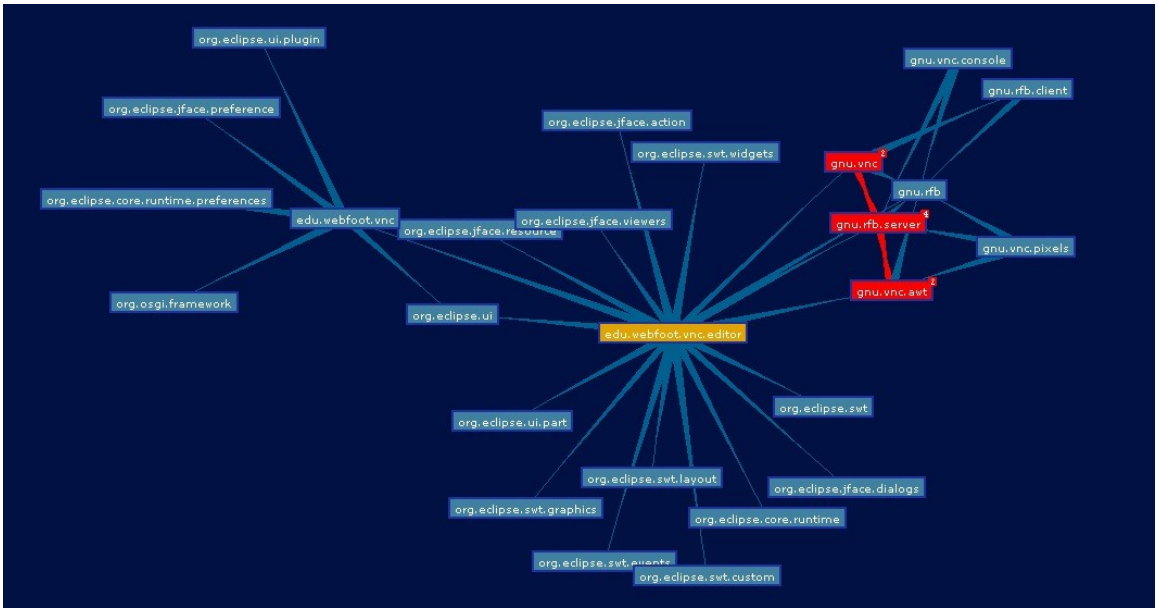

Figure 2: Complete Package Dependencies

The first two figures illustrate a cyclic dependency between three packages, as clarified in Figure 3.

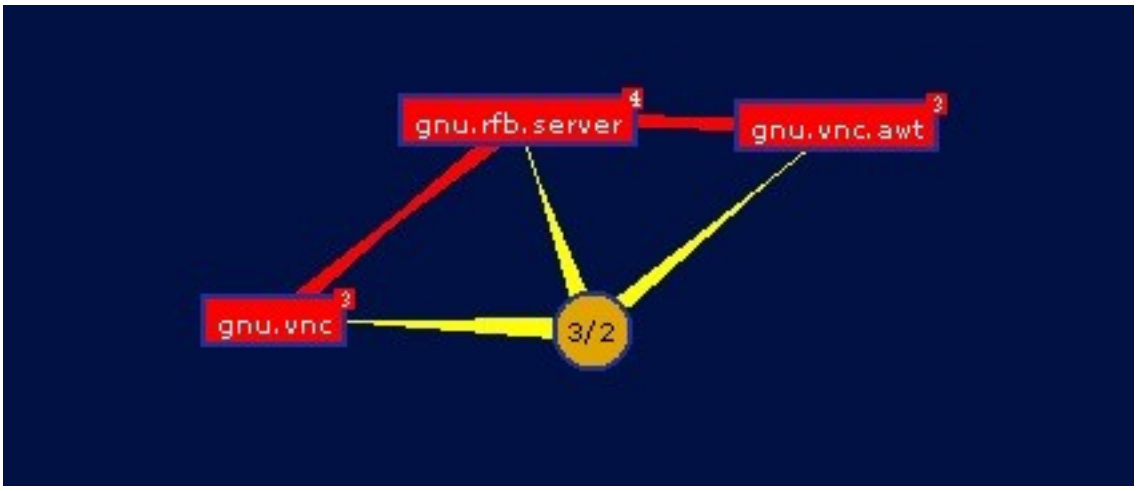

Figure 3: Cyclic Package Dependency

In order to determine the source of these we expanded one of the cycles as shown in Figure 4. The source of this problem is a section containing the Host, Authenticator, and Socket, which all need to interact in order to provide their service.

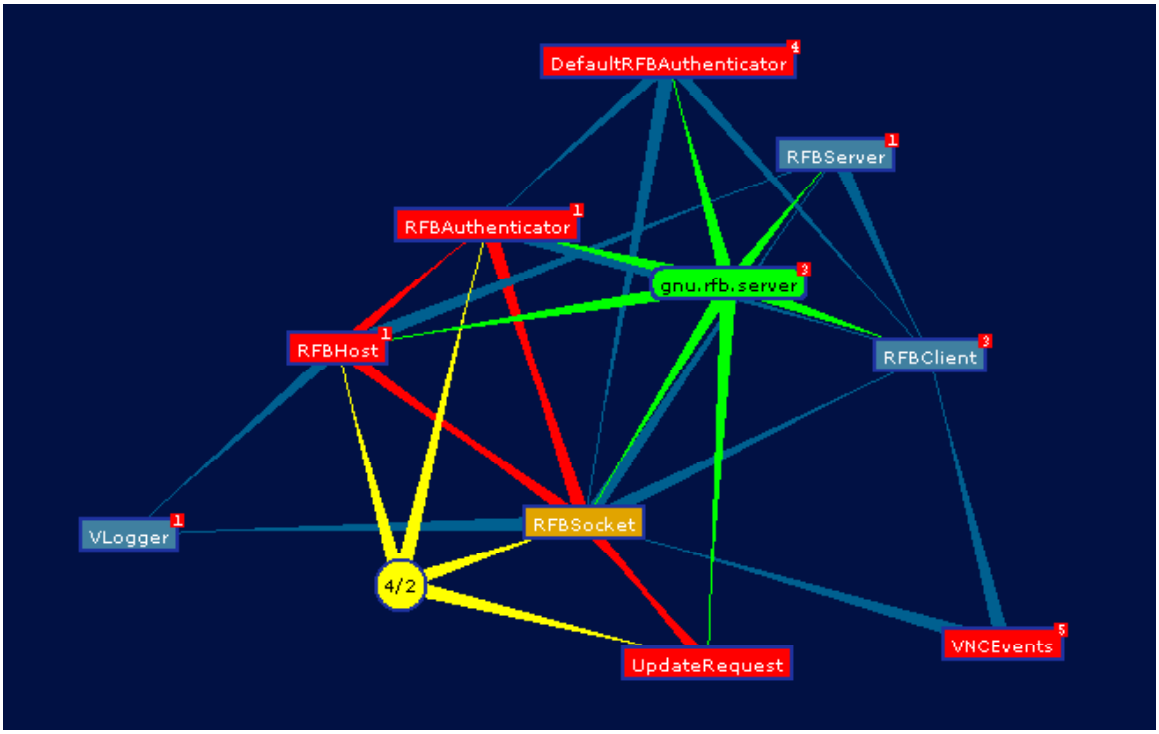

Figure 4: Class Dependencies within gnu.rfb.server Package

Figure 5 illustrates a hierarchical ordering of the packages based upon their dependency.

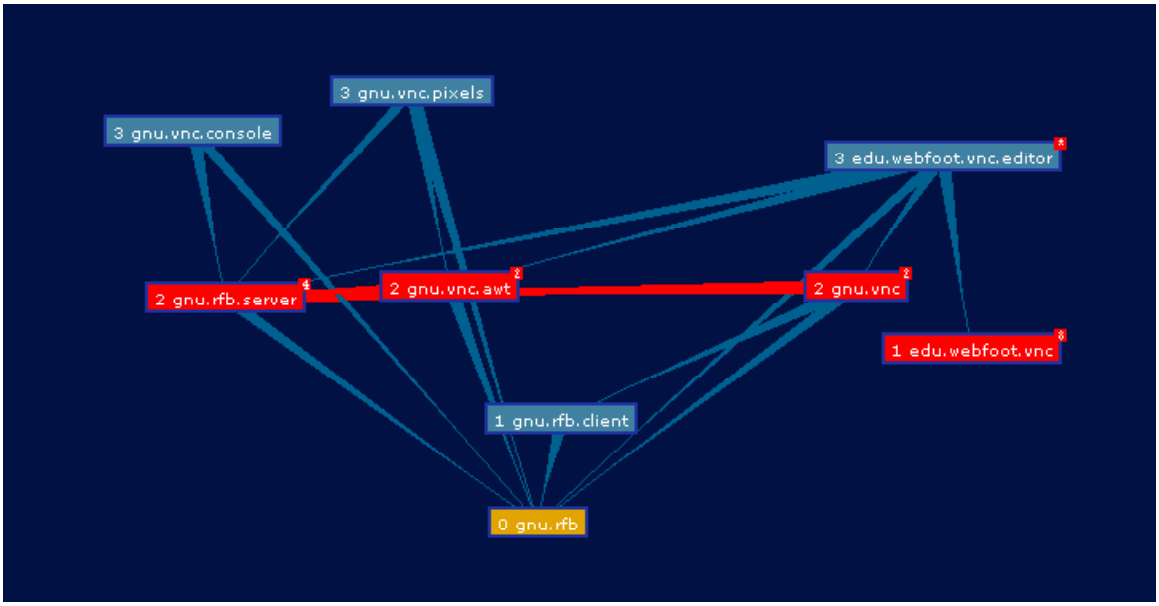

Figure 5: Package Layers

This more clearly illustrates what affect changes to classes might have upon the project. It also helps to illustrate stability of classes. Note that all of the external dependencies outlined in Figure 2 would be listed here at level 0.

5. Future Work and Conclusions

Virtual networking computing allows for shared control of a user's environment which, as mentioned in Section 2, enables a client to send keyboard and mouse inputs to a host system. Although a main feature of VNC, this can pose several issues when utilizing our tool. The first complication that arises as a result of this feature concerns input commands between the host and client systems. While collaborating, the host and client systems might send input to the environment that will produce undesirable results. One such example is if one user is typing in a text editor while the other user selects a new window sending the initial user's input to a new window and causing unexpected results. As a means of overcoming this problem future developers can implement restrictions to user input. Functionality can be added to restrict the input from a user so that only input from one user will be accepted to the environment. Which user can send input can be restricted using hotkey combinations, buttons placed in the user environment, or via other means that are chosen. Additionally, this will free up input from a user for various other purposes.

A significant drawback to using the VNC technology is the lack of effective communication between the host and the client computers. The tools built to implement virtual network computing rely on external tools to provide means of communicating between the host and client computers. Although the tool can be used to provide one-way communication depending upon which end has control of user input, it is inefficient and unreliable since control can be switched during runtime. Specifically, with our tool, control of input is given to both ends at once, which makes communication through direct use of the tool to be difficult at best. One feature that can be added to future versions of

the tool is messaging between the host and the client systems. Coupled with the restricted control mentioned previously in this section, the input of a user could be captured and displayed in a message box on the remote system, allowing for improved communications methods to be sent between partners.

As mentioned in Section 3, our plug-in was produced using an already existing tool and building off of it. By utilizing a tool that was already under production we were able to focus on other more significant concerns for the plug-in. Continuing with this approach, future developers can more fully integrate our VNC plug-in with the SolarFlare plug-in that was developed alongside ours for the Webfoot software suite. Our VNC plug-in would be able to make effective use of the audio communication component since neither the audio input nor the audio output is shared between the systems when using the VNC technology. Additionally, the messaging system already present in the SolarFlare plug-in can be utilized to implement the messaging system between the client and the host as mentioned previously in this section. In lieu of implementing a new messaging system, the SolarFlare messaging component can be used to add additional messaging functionality.

A significant drawback to our plug-in is the limited obtainable refresh rate. This is due to several factors. The first factor is the limited performance that can be derived from the Java Virtual Machine (JVM). The JVM is unable to produce the full efficiency and speed of the system at hand since it must devote resources to simulating and standardizing the system resources. This difference creates a reduction in the efficacy of our plug-in. Additionally, and more significantly, is the problem of the Java Security Architecture[17]. In order to protect the host system from malicious or insecure code

attempting to be executed by a program within the JVM, it provides a layer of abstraction between the host system and the virtual system that it presents to the program[16]. Several application programming interfaces (APIs) are provided to Java developers in order to pass through the abstraction layer to the native system; however the total number of interfaces remains limited. As a result, there is a restriction to the full capabilities outlined in the VNC Protocol that could be implemented. The VNC Protocol allows for incremental updates to take place, or, more specifically, for subsections of the screen to be updated instead of a full screen redraw. As a result of the Java Security Model, our plug-in is unable to implement this feature, losing a significant boost in the speed and reduction in network capacity used. If future developers manage to discern a means of overcoming this handicap and efficiently provide incremental updates, then vastly significant speed improvements would be realized.

Already implemented in our plug-in, as a means of improving communication between the client and the host, is the placement of a dot in the client's image, denoting the location of the host's mouse cursor. This allows the client to discern what the host is selecting or pointing at. This functionality is lacking in the other direction. The host currently will have no visual knowledge of where the client's mouse cursor is. This can provide difficulties anticipating where on the host's system the client might select with the mouse. This also increases the difficulty in communicating locations on the screen from the client system to the host system. In order to overcome this, future developers might seek to add functionality to allow a floating object to appear on the host system that will overlay all other windows and denote where the client's cursor currently is residing, if the client's cursor is within the screen region.

When connecting to a remote computer, one needs to specify an address for the host system. Future developers can implement a history or favorites feature for the connection address dialog box. This will ease the process of connecting to hosts for users, so that there is less for the client users to remember. Additionally, a developer might choose to implement a means of pinging a server in order to determine if a host is currently being run on a target system.

The current state of the plug-in requires that a client specify a password that matches the password specified on the host system prior to the initialization of the host. The password is encrypted using Data Encryption Standard and sent to the host for confirmation. Following that, the packets are sent between the client and host systems in clear text. Future developers might choose to add more robust security features to the plug-in. A significant challenge to the addition of a security model, however, is compatibility with other VNC client and host tools that are already in existence.

In conclusion, we set out to improve collaboration by creating and improving tools for developers to use. We researched and then implemented a tool that fills a unique niche which was previously unfilled in collaboration software. We designed and implemented our software using best practices that we learned throughout the WPI experience as Computer Science majors. We have contributed a small but useful piece of software to the open-source community which we hope to see grow in the future to become one of the few well recognized VNC implementations.

Glossary

CVS – Concurrent Versions System, provides version control using open source code. (http://www.nongnu.org/cvs/)

Eclipse – An open source integrated development environment written in Java and open to extension through plug-ins. (http://www.eclipse.org/)

EPL – The Eclipse Public License, An open source software license from the Eclipse Foundation[3]. (http://www.eclipse.org/legal/epl-v10.html)

GPL – General Public License, A common open-source license from the Free Software Foundation[5]. (http://www.gnu.org/copyleft/gpl.html)

GUI – Graphical User Interface, A visual interface for viewing and interacting with a program.

IDE – Integrated Development Environment, a tool used to develop software by providing helpful tools to a developer wrapped into a single environment.

JavaDoc – Java Documentation, a tool to take verbosely written comments within the code a transform them into readable documentation in an automated fashion. (http://java.sun.com/j2se/javadoc/)

MVC – Model View Controller, A pattern for developing systems which include a graphical user interface which helps to prevent common pitfalls.

RealVNC - An implementation of VNC for Windows, Unix/Linux and Macintosh platforms. (http://www.realvnc.com/)

RFB – Remote Framebuffer, A message based binary protocol used by VNC to share control of a system[11]. (http://www.realvnc.com/docs/rfbproto.pdf)

SourceForge – A web based collaboration tool which is tied closely with source control[12]. (http://web.sourceforge.com/)

Spike – An agile development technique where a developer goes off on a directed tangent to learn about a particular concept or technology.

SVN – Subversion is a version control system similar to CVS but with a larger feature set. (http://subversion.tigris.org/)

SWT – Standard Widget Toolkit is an open source toolkit for Java which is used by Eclipse for graphics[4]. (http://www.eclipse.org/swt/)

TightVNC – An open-source implementation of VNC with file transfer capabilities as well as support for the 'Tight' encoding[8]. (http://www.tightvnc.com)

TPTP – The Eclipse Test and Performance Tools Platform, a performance testing suite to help find bottlenecks in programs, including Eclipse plug-ins. (http://www.eclipsecon.com/tptp/)

UltraVNC – An open-source implementation of VNC with support for encrypted communications[13]. (http://www.uvnc.com/)

LGPL – The Lesser General Public License is a less restrictive version of the General Public License and is written primarily with software libraries in mind[6]. (http://www.gnu.org/copyleft/lesser.html)

VNC – Virtual Network Computing is a method of remotely viewing and controlling a system over a network connection via a client and server utilizing the RFB protocol[8]. (http://www.realvnc.com/vnc/index.html)

VNCj – Is an open-source implementation of VNC using Java, it is licensed under the LGPL[10]. (http://sourceforge.net/projects/vncjlgpl)

Webfoot – WPI's Environment Built for object-oriented tools is a collection of software to enhance collaboration, especially for distributed teams[18]. (https://sourceforge.wpi.edu/sf/projects/webfoot)

References

1. Applications of VNC. RealVNC Ltd. 09 January 2008.
   (http://www.realvnc.com/vnc/how.html)

2. Eclipse Documentation. The Eclipse Foundation. 2008.
   (http://planeteclipse.net/documentation/)

3. Eclipse Public License. The Eclipse Foundation. May 2004.
   (http://www.eclipse.org/legal/epl-v10.html)

4. SWT: The Standard Widget Toolkit. The Eclipse Foundation. 22 February 2008.
   (http://www.eclipse.org/swt/)

5. GNU General Public License Version 3. Free Software Foundation, Inc.
   29 June 2007. (http://www.gnu.org/copyleft/gpl.html)

6. GNU Lesser General Public License Version 3. Free Software Foundation, Inc.
   29 June 2007. (http://www.gnu.org/copyleft/lesser.html)

7. Hardt, Dick. "OSCON 2005 Keynote - Identity 2.0" 2005.
   (http://identity20.com/media/OSCON2005/)

8. Kaplinsky, Constantin. TightVNC Documentation. April 2008.
   (http://www.tightvnc.com/docs.html)

9. Kaplinsky, Constantin. TightVNC Related Software. April 2008.
   (http://www.tightvnc.com/related.html)

10. Rajesh Patel and Shawn Rhoads. LGPL VNCj. 26 January 2007.
    (http://sourceforge.net/projects/vncjlgpl)

11. Richardson, Tristan. "The RFB Protocol Version 3.8". RealVNC Ltd.
    18 June 2007. (http://www.realvnc.com/docs/rfbproto.pdf)

12. SourceForge, Inc. What is SourceForge.net? April 2008
    (http://alexandria.wiki.sourceforge.net/What+is+SourceForge.net%3F)

13. UltraVNC Team. UltraVNC. April 2008. (http://www.uvnc.com/index.html)

14. Metrics 1.3.6 – Getting Started. April 2008 (http://metrics.sourceforge.net/)

15. Krols, Diederik. The Anti-Patterns Blog. April 2008
(http://dotbay.blogspot.com/2006_01_01_archive.html)

16. Java Security Architecture. April, 2008
(http://java.sun.com/javase/6/docs/technotes/guides/security/spec/security-spec.doc1.html#18313)

17. Java SE Security. April, 2008.
(http://java.sun.com/javase/technologies/security/index.jsp)

18. Pollice, Gary. The Webfoot Project. April 2008.
(https://sourceforge.wpi.edu/sf/projects/webfoot)

19. "What is Jazz?" The Jazz Project. IBM Rational. April 2008.
(http://jazz.net/pub/index.jsp)

20. Jupiter. University of Hawaii's Collaborative Software Development Lab. April 2008.
(http://csdl.ics.hawaii.edu/Tools/Jupiter/)

21. "Enabling Geographically Distributed Development." CollabNet, Inc. April 2008.
(http://www.collab.net/solutions/distributed_development.html)

22. "Open Source Software Engineering." CollabNet, Inc. April 2008.
(http://www.tigris.org/)