Multi-computer Virtual Whiteboard

Josh Job
Keith Koons
Nick Walker

Computer Science
South Dakota School of Mines and Technology
501 E St Joseph St
Rapid City, SD 57701

Abstract

The growth of the Internet has changed how teams do software development. Before the Internet, you needed to have your software development team all in one location to work together to produce software. With the Internet, it is now possible to connect with other developers from around to globe to produce software. There currently is not tool that allows developers to efficiently work together to solve development problems. The paper will discuss the requirements for such an application. The paper will then go into major problems discovered while designing and implementing the application, as well as solutions to those problems.

1. Introduction

1.1 Purpose of this document

The purpose of this document is to describe to the reader the difficulties that exist in holding design meetings when not all participants are present and how those difficulties are resolved by this virtual whiteboard project.

1.2 Scope

The Virtual Whiteboard has been assigned by Mr. Troy McVay of Stinger Ghaffarian Technologies (SGT), a contractor to the United States Geological Survey. Currently the offices of the software engineering department share information and progress via telephone call and email. This method, however, is insufficient for the employees as it requires a significant amount of time and organization. Therefore, Mr. McVay would like a Virtual Whiteboard program which allows the members of the software engineering department to schedule online meetings and edit shared documentation, diagrams, and software code in real-time. The program can, for example, connect the office in Rapid City with the office in Sioux Falls via broadband internet.

1.3 Definitions, Acronyms, and Abbreviations

API – Application Programming Interface
IRC – Internet Relay Chat
XML – Extensible Markup Language
GUI – Graphical User Interface

2. Problem

2.1 Virtual Whiteboard

To meet all of the projects requirements, three default window types have been developed: documentation, code, and diagram.

The window content has to be updated in real time so that all parties are constantly up to date. Any change made by one user will be immediately updated to all members of the session via the server. Because these changes can modify existing data, the user desiring to change the content of the window will have to lock the window while they are making changes and then release the lock to notify that they are done.

The documentation window has text formatting capabilities and can provide the meeting with the ability to create or to load files to help in the design process.

The code window is a specialized subset of the documentation window. Text formatting capabilities are specialized to handle specific key words and to highlight related objects.

Other window types may be needed in the future, thus our window engine is library based and can generate as many types of windows as there are libraries. The three default windows are also libraries and can be removed from the project should they not be needed.

2.2 GUI Requirements

Changes must be tracked to allow anyone to filter out the changes of any user at any time. The changes made by each user is also highlighted the color that is associated with the user.

The GUI is responsible for locating and importing any window libraries that is included with the application. The three window libraries that are included by default are the documentation, code, and diagram window. These libraries contain the necessary information for the GUI to create and handle window types.

The user also uses the GUI to set up the session parameters and select whether or not to use encryption. A password can be provided and then sent to all participants of the meeting to prevent unwanted users from gaining access to your session.

2.3 Server Requirements

The user that creates the session is set up automatically to be the server hosting the meeting. All updates go through the server and it keeps track of all users.
The meeting must be resilient enough to persist even though the server has left the session. If the server is no longer part of the session, the clients will check an internal list of users and determine who is to be the next server and they automatically switch to that server.

The server must ensure a secure connection and use encryption on all communications to prevent data from being compromised.

The server is required to behave much like an IRC or Instant messaging server. That is messages must reach their destination; unlike other network protocols where reliability must be added at a higher level. Where the Whiteboard application differs from others, is that any user needs to have the ability to act as a server. To do this, the only time the main application differentiates between local and remote servers is when it is establishing a connection. As with many designs, the ease of extending functionality later is important.

3. Implementation

3.1 Details

This project is being implemented using C++ with Qt [1] libraries to handle the GUI as well as the server. It is implemented under a Linux environment, but was written so as to be cross-platform with Windows.

Qt is a powerful cross platform library that enables threading, graphical content, and secure socket connections.

The server is written as a separate application that is spawned at need by the GUI. This can happen either when a user creates a meeting or when the current server leaves the session and a new user needs to be selected to act as the server.

3.2 GUI Design

The Qt application's main widget consists of an application control window. From this control window the user can create or join a session, open new windows (i.e. documentation and source code windows), and edit session member attributes (i.e. color).

Each window is implemented in its own class, containing only the properties specific to the windows functionality. The control window then creates and manages instances of its sub-windows. This was achieved with Qt by setting the control window as the parent window to each other generated window. By doing so, termination or closing of the control window causes all other window to be terminated or closed as well.

Although Qt provides a Designer Tool to create applications by dragging and dropping widgets, it has been found easier to generate the code "by hand" and also allows for easier management and overview. The control window is derived from the QMainWindow class, which provides a simple application window including minimize, maximize, and close features. Other windows are derived by the QWidget class and are tailored to a specific window type (i.e. dialog box, help window, etc.) by means of the QWindowFlags.

From these base classes widgets such as menu bars, buttons, text box, etc. can be added to the windows using the Qt layout classes. Layout classes such as the QHBoxLayout class and QVBoxLayout class will order widgets in a horizontal and vertical position, respectively.

In order to import window library classes, as mentioned earlier in section 2.2, Qt's plugin class is used.

3.3 **Server Design**

The Qt library provides two important features that influenced the overall design: 1) Each thread can have its own event queue and 2) Signals and Slots can cross threads using this event queue. To simplify early development, a post threaded design was chosen to connect remote clients. When a client connects, a class is created to control data specific to a client and a separate thread to handle events related to the client. A name is maintained for every client. A remote client also has a socket associated with it. The local connection does not wrap a socket, and instead, is wrapped in a standard class to communicate with the rest of the Whiteboard application. A global data structure contains information related to the state of the server. It maintains a list of connected clients, and links to code to handle a message type. Access to this structure is provided by Read Write Locks through the Qt API.

The XMPP [2] protocol heavily influenced the design of the communication protocol. XML is used to describe the stream. The whole stream is surrounded with a stream element. Within a stream, messages are sent in blocks surrounded by an element describing the message type. There are four types planned: authentication, presence, message and error. Authentication messages are used at the beginning of a stream for a client to prove they can connect. Presence message transmit information regarding the online status of a client; an example of this is the message sent to other clients when a newly connected client completes authentication. The message type is meant to convey the information sent by the Whiteboard application. Error messages are sent to inform the other side of the stream that the stream is currently in an undefined state; this is usually resolved by disconnecting the stream.

4. Conclusion

This virtual whiteboard project allows for feature rich and secure remote design meetings and facilitates the instant sharing of dynamic content. It provides a stable meeting platform that will remain alive until the meeting is completed.
The application is both portable and cross platform to allow a user to attend or host a meeting from anywhere they can find an internet connection and a Windows or compatible Linux platform.

Extensibility is insured by using a plugin system for the window types and by using a flexible packet system to keep users updated.

References

[1] Nokia Qt – http://www.qtsoftware.com/products; accessed 23 March 2009

[2] XSF XMPP – http://xmpp.org/; accessed 23 March 2009