TeChat: A Handwritten Chat/Instant Message System

Writeup

Peter Na - naph@seas.upenn.edu

Faculty Advisor: Professor Zachary Ives
Abstract:

Instant messaging has been established as an essential method of communication over the internet. However, many aspects of person-to-person communication are lost through plain text messaging. Plain text removes many of the subtleties involved in a conversation such as emotion, and body language. Often times, even in face-to-face conversations, concepts cannot be conveyed through words alone, and the speakers may want to better “illustrate” their points. Drawings can be immensely more effective at communication than plain text (given a rudimentary amount of artistic ability), and can give interactions a “personal” touch.

This project is meant to expand upon the concept of “draw chatting” and bring it into a real-time environment, without sacrificing the features of an instant messaging client.

Related work:

This application will be similar to all chat software available, such as AIM (America On-line, 2008), GTalk (Google, 2008), and Windows Live Messenger (Microsoft, 2008), in that it is a chat program, and can send and receive messages. The inspiration for this project was initially derived from Windows Live Messenger, which has a very basic handwriting feature that can be used to send simple handwritten messages in addition to normal text messages. Browser-based web applications such as Paintchat (ShiChan, 2003) and Iscribble (Tyb, 2008) enable groups of users to draw on a shared canvas. Pictochat, a Nintendo DS application, allows users to send each other pictures they drew on the game system’s touch screen over an ad hoc wireless connection (IGN, 2004). The handwritten blog site Tegaki, where users draw blog entries and chat with drawings in the comments sections, is also an inspiration for this project. On Tegaki, users often use the handwritten commenting system to “chat” with friends and other users. Often times, they adopt original characters they created to represent themselves, and draw their characters performing certain actions or displaying a certain emotion to help emphasize what they are trying to say. The site is greatly popular among online artist communities, as well as roleplaying groups, and creatively inclined individuals who enjoy art and drawing (Pipa, 2008). However, since the sites were not specifically designed for chatting, it lacks many of the features characteristic of a chat application, and most importantly, one must constantly refresh the site to see if anyone has responded to their post.

Of the applications specifically designed for messaging, none explore the potential of handwriting as a medium for communication. Windows Live Messenger does include a simple implementation of a handwriting feature, with a pencil tool, eraser tool, and 16 colors. While drawing, one cannot send any text messages, and handwritten messages are not saved in the chat history. So, the application lacks many of the features that users who chat through images would desire. Group canvas applications like Paintchat and Iscribble are based more on collaboration than chatting, and thus it is difficult to tell who drew what picture, and there is no sense of when each drawing on the canvas was created. Tegaki is a site designed as a blog hosting system, so it focuses more on blog features than features characteristic of chat clients.

This project will focus on the concept of chatting through art, instead of text. So, the feature set will include better tools for artists, as well as provide easy ways of saving and sharing chat logs, along with other features to make draw-chatting a more satisfying experience.
The project relies heavily on the Extensible Messaging and Presence Protocol (XMPP) for sending and receiving messages to remote clients. Developed by the XMPP Standards Foundation (XSF), XMPP is a real time open-source XML-based protocol used for sending data between peers. Work has been done to make it a standardized and generalized framework for exchanging XML data, and make it more extensible and secure than its predecessor, Jabber (P. Saint-Andre, 2004). As it is quite simple, it is flexible as well, and so can be extended for more than plain text chatting, which is why the protocol is being used for this project.

The appeal of XMPP lies in its extensions, which allow it to accomplish more than simple presence and chatting. Since XMPP is a text based protocol, it cannot handle binary data on its own. Thus, file transfer extensions are available, such as SOCKS5 Bytestreams (XEP-0065) or In-Band Bytestreams (IBB) (XEP-0047). Based to the specification documents, IBB is a good match for this project, since it is reliable and robust, and because the image files that are produced by the application are not tremendously large. Also making things simpler is that fact that IBB file transfers are not affected by firewalls (Karneges, 2006).

The other half of this project is the user interface and drawing application. The Windows Presentation Foundation subsystem has fairly recently been added to .NET framework for version 3.0, to create a unified platform for GUI programming. Particularly useful is the InkCanvas class, which simplifies the process of producing an application for creating and saving images. WPF integrates the design aspect of GUIs with development through the eXtensible Application Markup Language (XAML), streamlining GUI development, and thus makes creating images and creating a GUI around logs of images somewhat simpler (Chappell, 2006).

**Technical approach:**

The chat system will be based on XMPP, known previously as Jabber, a real time open-source XML-based protocol used for sending data between peers. Images are drawn on a canvas through mouse or pen tablet input, which is encoded by the client into a form transportable through XMPP servers to the destination client. The destination client will display the image in a log along with a chat history.

![Diagram](image.png)

*Fig 1. Transportation of data using application and XMPP Protocol*

Jabber XML streams consist of `<presence/>`, `<message/>` and `<iq/>` (info/query). A basic connection would be as follows (From the XMPP core specifications):
Since at its core XMPP is used for basic text communications, and cannot transfer binary data, so
development will focus on using the available XMPP Extension Protocols (XEPs). The extension In-Band
Bytestreams (IBB) (XEP-0047) has been chosen for this project, as it is reliable, and is not affected by
firewalls. To implement this extension, the client has to encode the binary data as a Base64 string, which
is divided up into blocks (4KB blocks for this project), then sent to the receiver as a Jabber XML stream
(Karneges, 2006).

The application is currently being coded in C#, as there are robust network programming and XMPP
libraries available, as well as many drawing functions (specifically the System.Net and System.Drawing
namespaces). The third-party agsXMPP library has been very useful for implementing the chat
functionality of the application. The Windows Presentation Foundation (WPF) subsystem of the .NET
framework is being used to develop the user interface and canvas application. The drawing application is
being developed on top of the XMPP client (independent from XMPP standards, as the image is encoded
at the completion of the drawing). Currently, after an image is drawn (through an InkCanvas control), it is
saved locally for logging, and encoded as a base64 string, which is then encoded into an XMPP message.
It is sent to a Jabber server, where it is relayed to the receiver, where the message is decoded and displayed in the chat log.

The challenges for this project lie in learning and understanding the XMPP protocol, which is taking a substantial amount of time. However, using an established protocol for chatting and discovery will likely, in the long run, free up more time to develop the drawing application itself. Otherwise, a significant amount of time would be spent network programming and debugging a completely new protocol. Challenges will also include creating useful drawing tools, encoding/optimizing images so they are sent in a reasonable amount of time, being able to display many images without taking up too large of a memory footprint, and other features such as pen tablet sensitivity, as well as designing a user interface that is functional and intuitive.

For the early alpha public test release version of the project (and for future versions), a dedicated server will be set up for TeChat users.

**The interface:**

![The canvas](image)

**Fig. 2. The canvas**

The drawing application allows users to choose a color, and set brush width. Users may choose any color that can be displayed on a computer monitor from the color picker interface, which is orders of magnitude more than the 16 colors available to users of Windows Live Messenger, and the brush width picker provides more granularity. Users have the option to use the default cursor (the same arrow cursor for all other normal Windows applications), or a cursor that shows the color and width of the current brush. If the user is using a pen tablet for input, the application will automatically detect pressure, and adjust the brush strokes according to the pressure.
Fig 3. The chatlog

The chatlog displays all messages that have been exchanged in a conversation. When a message is received, the image is decoded, then stored locally on the disk, to log chat history.

Fig 4. The buddy list

The buddy list displays a list of all buddies online who are available to chat. From this window, one would be able to select a buddy to exchange messages with. When presence data is received, entries are updated. Advanced buddy list features are planned for a future release.

Resources required:
The resources required for this project will include a C# IDE (Microsoft Visual Studio) for the software coding aspect, and an XMPP server to handle authentication and friend discovery functions. Heavy use will be made of programming language manuals (C# books, online documentations for network programming and drawing libraries), network programming books, and the XMPP documentation. Once the project is released to the public, users, testers, and their computers will be needed to find bugs and provide input on ease of use issues.

**Conclusion:**

During the course of the project, an idea was formulated, and a system was designed to make chatting more interesting, interactive, and fun. A basic implementation was made, and will be ready for public testing soon. Unfortunately, some of the more advanced features present in modern day chatting clients are not present as of yet in TeChat, such as advanced buddy list customization. The drawing application, while definitely useable, is missing some convenient features such as copy, cut, and paste. Work on the project also brought to light the difficulty of effective user interface design, and while the current interface is the result of several reworkings, it still could use more work. Despite the shortcomings, however, TeChat is still unique, fun, and interesting to use, and will hopefully spark a deeper interest in the concept of “draw chatting” in the mainstream.
References:


This technical article highlights how the Windows Presentation Foundation subsystem can be used to design user interfaces. It provides many usage examples and explanations of features of WPF. The author speaks about the importance of powerful GUIs, and how WPF can link designers and developers to create innovative and appealing user interfaces. The author, as the head of a consulting company, generally speaks favorably of the system, but also provides examples and short tutorials to back up his claims.


This document specifies the features and implementation requirements of the In-Band Bytestreams extension for XMPP. As a Draft Standard of the XMPP Standards Foundation, it has been approved by the XMPP Council. Any file transfer implemented using In-Band Bytestreams must follow the specifications laid out in this document. Images created by the project are transferred to peers through In-Band Bytestreams, and so, the project makes heavy use of the document to stay standards compliant using XMPP.


This document is the core protocol specification of XMPP, and so is the main basis for the chat application portion of the project. As the core specification of XMPP, it is approved by the XMPP council and is the basis of all other XMPP documents. XMPP XML streams, how outgoing messages should be formatted, error handling, and general standards are all introduced and explained. The project is dependent on the information provided by this document to stay standards compliant, or to work at all.

Related work:


