**Final Project:**

**ObjectDraw**

**Architecture Document**

May 7th 2009  
  
**Group One**  
Joseph Trapani  
Ben Helppi  
Satpreet Singh  
Mark D'Souza  
Amanda Parker

**Contents:**

1.    [Introduction](http://docs.google.com/Doc?docid=ajk73mnhm9vx_186crfz3jcd&hl=en#Introduction_09720386439369721)

2.    [Architecture](http://docs.google.com/Doc?docid=ajk73mnhm9vx_186crfz3jcd&hl=en#Architecture_2990002143420188_)

3.    [Choice of technology](http://docs.google.com/Doc?docid=ajk73mnhm9vx_186crfz3jcd&hl=en#Choice_of_technology_731165174)

4.    [Network Model:](http://docs.google.com/Doc?docid=ajk73mnhm9vx_186crfz3jcd&hl=en#Network_Model_3287089582625446)

5.    [Reflection](http://docs.google.com/Doc?docid=ajk73mnhm9vx_186crfz3jcd&hl=en#Reflection_24949863741915013_1)

**Introduction**

The purpose of this project was to develop a drawing tool that could draw objects on multiple computers, allow users to save and load a drawing session and transfer tools to use without recompiling the program.

MultiDraw is an extended version of ObjectDraw that permits multiple users on different computers to collaborate across a network on a shared drawing. MultiDraw permits remote users to view a session in real time and allows the owner of a drawing to transfer control to another user, thereby allowing the remote user to manipulate and modify the drawing. Other connected users can continue to view these changes in real time. Drawing canvas sessions can be saved and reopened for display and modification. MultiDraw allows the user to add and share drawing tools without recompiling the application.

**Choice of technology**

**Considered:**

The first technology considered was *Remote Method Invocation(RMI).*  This technology automatically handles the low-level network communication.  The con is that a lot of background work has to be done to set up an RMI connection.  The second technology considered was *Simple Object Access Protocol (SOAP).*  Soap web services need to be deployed and requires the PC to run Tomcat, thus making this choice infeasible.

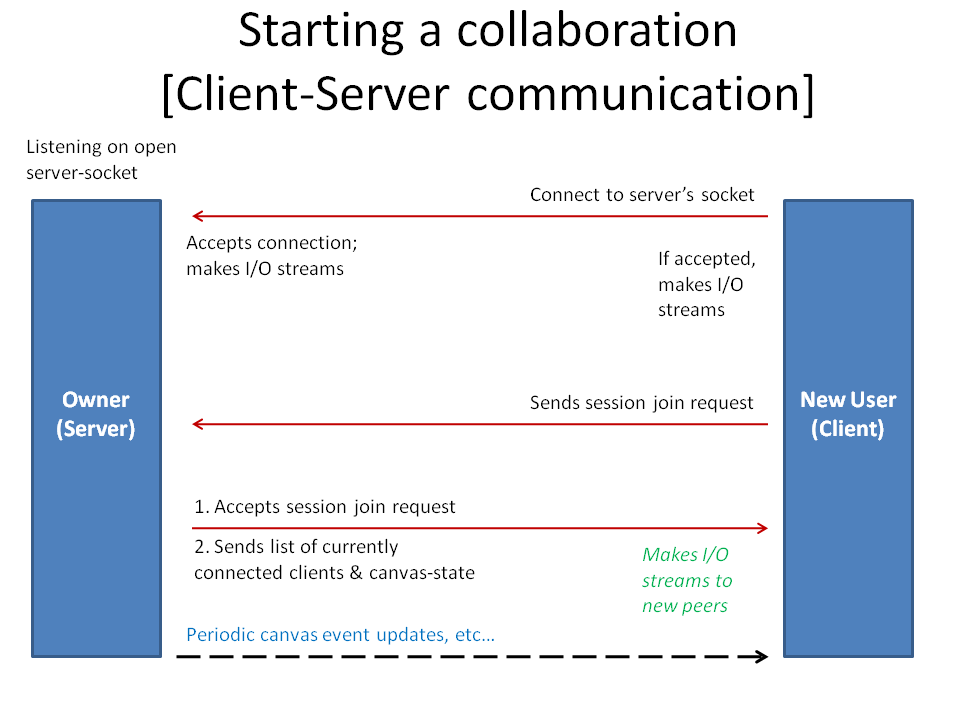
A peer-to-peer connection was used because it is decentralized in that each PC can act as a client to request data or as a server to provide data.  Specifically, one user would control all programs at any time using a Session.  When the user quits or another user wishes to take control, the session control instance is passed to the other user.  A "keep-alive" command is sent periodically to refresh the program to keep the program from timing out.

**Selected: Sockets**

A socket is one end-point of a two-way communication link between two programs running on the network. Socket classes are used to represent the connection between a client program and a server program.

**Architecture**

MultiDraw is based on the Model View Controller Architecture as was ObjectDraw.  A Graphical User Interface (GUI) contains the Drawing Canvas, Main Menu, Control Panel, Session Tab(s), Chat Window and Chat List from which the User selects inputs.  A Window Controller responds to the User's input.  The main program initializes the application, loads the GUI components and tools.

 Instances of MultiDraw *simultaneously participate in both a Server-Client and a Peer-to-Peer form of communication.*  A server-socket is active on every member, so any member start a session and can accept new clients. From that point the client is fully connected to the network, and operates as a peer. Conventionally the program manages event permissions. Thus every member is able to chat, but only one figure recognized as the master is able to create drawing event. Ownership/Mastership can be transferred, requested, auto-transferred if master dies, and even forcibly taken.   ****

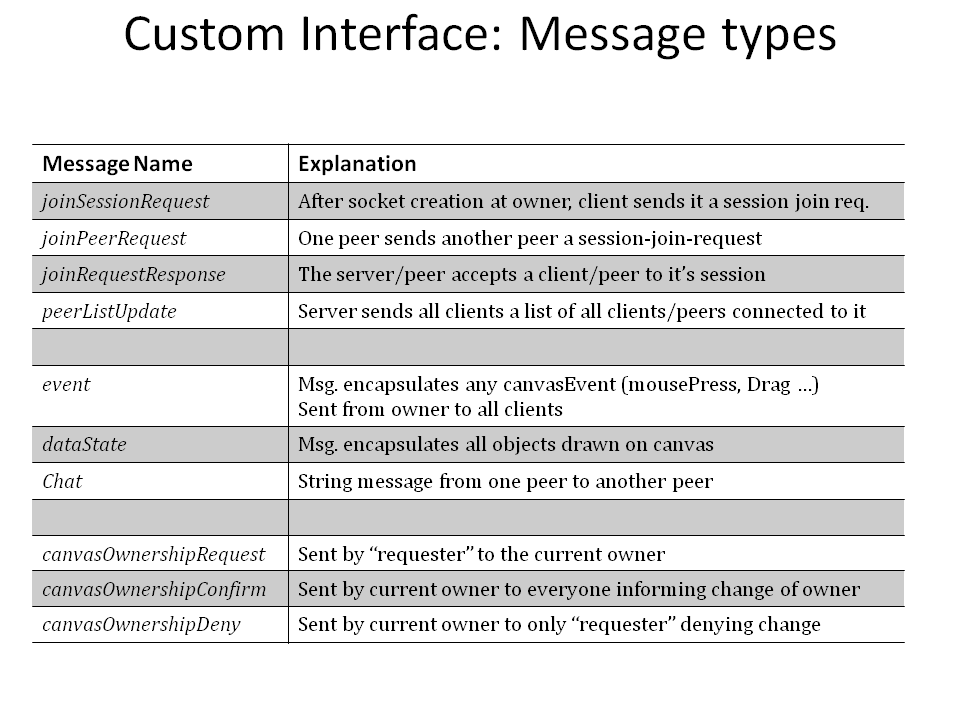
**Figure 1.**  MultiDraw Initialization

The user can either create or join a session.  Upon entering MultiDraw, the user can draw on a blank canvas or load a session to collaborate with others on a drawing.  If the user is allowed to draw (typically session master), the event to draw the object is sent over the socket(s) and received by each member.   The member draws the point on the canvas.

The GUI for the new program has more panels and options. A Tabbed Pane shows the canvases of the current sessions. A Chat Pane has a Chat Window to display the messages from the users and a panel lists the current users. In the new program, the applet is not used. A controller handles the user's actions for the File, Tool, Control, Session and Help Menus. The user's inputs for connection are validated. The user in incorporated as a member for a network session. A new session is created so that object drawing functionality will work and a new instance of the application is instantiated.

**Network Model:**

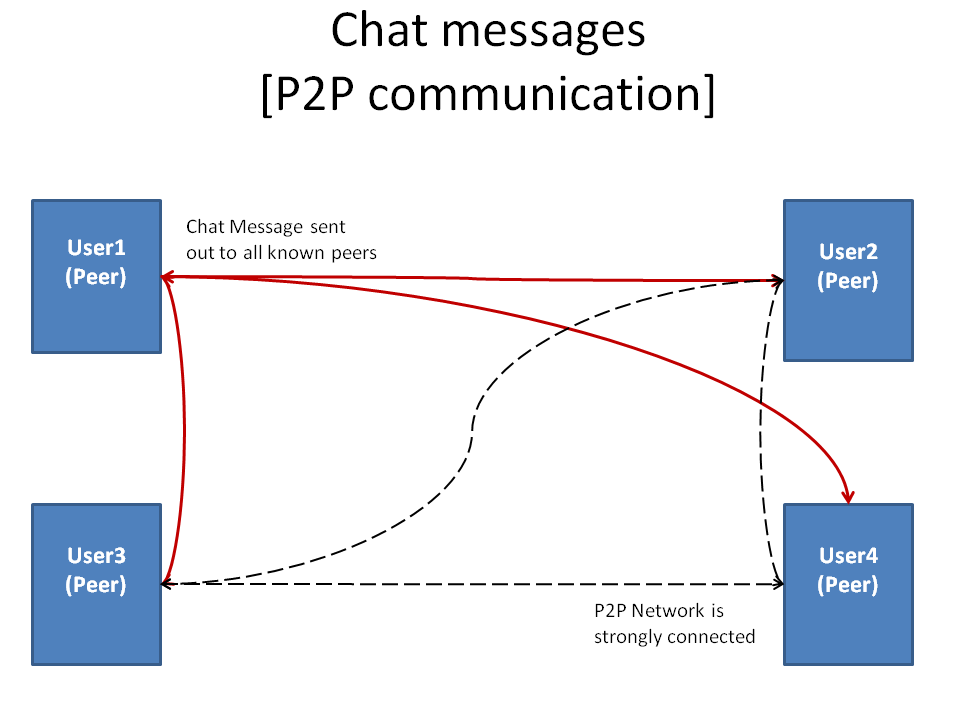
  The network model is very robust, and well formed to allow more functionality than is required for MultiDraw. For example the session manager can handle an arbitrary # of sessions. However it was only practical to draw one at a time. Constantly ran into defects in the java runtime. This project should have been able to be completed with one thread for sockets, and one thread for the Panel created. However it was found that Java Asynchronous methods were all defective, causing much frustration, and rework to accommodate these flaws. Final method was to use the # of threads required, which is 1 for Panel, 1 for a server socket, and 1 \* # of peers.



**Figure 2.** Messages in MultiDraw.

All events are handled by an instance of a Session. For example, the user selects the line tool, and clicks on the canvas. If permission allow, a mousePress event is fired. This interacts with the user’s local canvas, but also sends an event to every peer which does the same. Note that permission is determined by one function, so other modes are possible, such as everyone drawing at once, but is not implemented. The end result is that a line is created if the user is the valid owner of the session. Chat operates the same way, but since it is a unique event, it doesn’t check permissions, and everyone is allowed to chat.

A user can instantiate itself as a Master or a Client as shown in Figure 2. The user builds a network Session instance which sets up the local user and master objects or connects to an IP and Port.  The Session is responsible for handling all User actions that take place while the program is running.  It keeps track of the active members, who's in charge and manages the data. The network function connects the points on the canvas to the destination using the socket.



**Figure 3.**  Peer-to-Peer networking.

When the user generates an event, the object is assigned a unique ID and sent over the network as shown in Figure 3.   Upon receiving an event, the appropriate tool is called to draw the object.

A session manager manages all sessions in which the user participates. It creates a new network session. It automatically assumes that this will be an active session. The manager can update the canvas that it is controlling to switch focus to a particular session. Methods are provided to get the current session, all active sessions and remove a session from those being managed. The data for an active session is updated.   
  
A network bundle is created that contains the person to which it is associated, a stream to get objects from the person, a stream to send objects to the person and the socket number shared between the user and the person. A singular network object is used for all communication over the network and must be serialized. Network communication can be a canvas event, a transfer of ownership or a type of object drawn on the canvas. The object contains the original person who created the object, the recepient of the object, the data and the reason the object was made for deconstructing the object.   
  
A network message structure incorporates the Session which is sending the message, the Object which is being sent and the message type. This transfers mouse events to every client. Each user can send and receive data through the same port. A peer can issue a command to gain access to the drawing canvas. The server routine handles all requests using the message structure that is serialized and routes messages based on the type of request.

**Reflection:**

- Tool loading (from file/manual) 🡪 Done

- Tool loading (automatic/over network)  🡪 Not implemented

A user can load a new tool dynamically if he has the .class file of the tool available. On selecting the new tool using the Add-Tool function provided, the new tool becomes visible in the list of tools in the drop-down menu bar. Adding tools automatically over the network was not implemented.