



Project Report and Technical Documentation

Zurich University of Applied Sciences Winterthur (ZHW)

Thomas Jund <info@jund.ch>
Andrew Mustun <andrew@mustun.com>
Laurent Cohn <info@cohn.ch>

1st January 2005

Version 1.0

Department: Engineering
Date of Delivery: 2nd July 2004
Research Mentor: Peter T. Früh
Prof. Dr. sc. techn., dipl. El.-Ing. ETH
University lecturer for Software Engineering

Abstract

This document describes the specification, design and implementation of TeamDraw, a chat client that features textual as well as graphical capabilities.

Some of the most popular services the Internet has to offer, have to do with the direct communication between its users. However, most of these services are purely text based which clearly hinders creativity. When it comes to brain storming and planning projects, using graphical elements to visualize ideas and thoughts can be crucial.

The software application that was developed for this project allows the users to exchange text messages as well as graphical elements such as lines or simple shapes. The implementation builds on existing solutions and reuses as much as possible from the concept of existing text based chat systems.

This project has shown that it is realistic to simultaneously work on drawings that are shared over the Internet.

Contents

1	Introduction	1
1.1	Scope Of This Document	1
1.2	Motivation	1
1.3	Example Applications	2
1.4	Task Description	2
2	Instant Communication Over The Internet	4
2.1	How IRC Works	4
2.2	Who Uses IRC?	4
3	From IRC To TeamDraw	5
3.1	Messages vs. Documents	5
3.2	Stateful IRC And Persistence	5
3.2.1	Clients With Equal Status	5
3.2.2	Centralized System With A Bot	6
3.3	Performance	6
4	Application Requirements	8
4.1	Functional Requirements	8
4.1.1	Basic Chat Functionality	8
4.1.2	Graphical Extensions	8
4.1.3	Managing Drawings	9
4.1.4	Platforms	9
4.2	Interface Requirements	10
4.2.1	Graphical User Interface Of The Client	10
5	Design	11
5.1	Architecture Overview	11
5.2	Overall Design	11
5.3	Design Of The TeamDraw Modules	12
5.3.1	Graphics Library	12
5.3.2	IRC Library	15
5.3.3	Core Library	16

5.3.4	Application GUI	17
6	The TeamDraw Client	19
6.1	The Main Application Window	19
6.2	Connecting To An IRC Server	19
6.3	Channel Dialog	20
6.4	Joining And Leaving A Channel	20
6.5	Text Based Communication	21
6.6	Graphical Communication	22
6.6.1	Tools	23
6.6.2	Highlighting	24
6.6.3	Selection	24
7	The TeamDraw Bot	25
7.1	Locking Graphical Entities For Modification	25
7.2	Persistence Of Drawings	26
7.3	Receiving And Forwarding Graphical Messages	26
7.4	Optimized Distribution Of Messages	26
7.5	Distinguishing Users	28
7.6	Configuration And Usage Of The Bot	30
7.7	IRC Commands For The Bot	30
7.8	Settings File	31
8	Project Management	32
8.1	Project Initiation	32
8.1.1	Choosing A License	32
8.2	Documentation Tools	32
8.2.1	Source Code Documentation	32
8.2.2	Technical Documentation	32
8.3	Development Platform	33
8.4	Programming Language	33
8.5	GUI Toolkit	33
8.6	Version Control System	34
8.7	Generation Of Executables	34

8.8	Graphics Format	34
8.9	Project Organization	35
8.9.1	Project Responsibilities and Deliverables	35
8.9.2	Project Schedule	36
9	Conclusion	37
9.1	Room For Improvements And Additional Features	37
9.2	Significance Of The Project	37
9.3	Learning Process	37
A	Protocol Specification	38
A.1	Message Details	38
A.2	Message Overview	38
A.2.1	Syntax Format	39
A.2.2	ABNF Notation	39
A.3	Message Details of TeamDraw Messages	40
A.3.1	Overview	40
A.3.2	Acquire Request Message	40
A.3.3	Acquire Reply Message	40
A.3.4	Choose Drawing Message	41
A.3.5	Drawing Completed Message	41
A.3.6	Registration Message	41
A.3.7	Registration Reply	41
A.3.8	Join Message	42
A.3.9	Drawing list Request Message	42
A.3.10	Drawing list Reply Message	42
A.3.11	Graphical Message	42
A.3.12	Graphical Message Start	43
A.3.13	Graphical Message Continue	43
A.3.14	Graphical Message End	43
B	Supported SVG Elements	44
C	Glossary	45

D	References	46
E	CD ROM Contents	47
F	About the Authors	48

List of Figures

1	The basic idea	2
2	An IRC network with clients.	4
3	Clients with equal status	5
4	Centralized system with a bot	6
5	GUI requirements	10
6	Architecture overview.	11
7	Graphics library design: model and view classes.	13
8	Graphics library design: controller classes.	14
9	Sequence diagram for drawing a line.	14
10	IRC library design.	15
11	Core library design.	16
12	The classes of the TeamDraw GUI in client mode.	17
13	The classes of the TeamDraw GUI in the standalone mode.	18
14	Application GUI design.	18
15	Server connections dialog	19
16	Channel dialog	21
17	Text area	21
18	Graphics area	22
19	Highlighting	24
20	Selection	24
21	Locking and modification sequence.	25
22	Drawing sequence	26
23	Round robin.	27
24	Login sequence	28
25	Bot joining sequence	29
26	Planned schedule.	36

27	Measured progress by module.	36
28	Message overview.	38

List of Tables

1	Server connection parameters.	20
2	TeamDraw tools.	23
3	Overview of bot commands.	30
4	Bot options, settings file.	31
5	Value benefit analysis for the technical documentation.	32
6	Value benefit analysis for the development platform.	33
7	Value benefit analysis for the programming language.	33
8	Overview of message details.	40

1 Introduction

This chapter briefly introduces project TeamDraw and defines the scope of this documentation. Further it should provide sufficient background information to understand the context and goal of this project.

1.1 Scope Of This Document

This document should give the reader an idea about how instant communication over the Internet (especially in the case of IRC) works. It then describes what has to be done to allow graphical communication on top of a text based service.

Reading this document does not require any knowledge of the IRC service. However, the reader should have a good idea of how the Internet works and understand the basics of communication and server / client models.

The more technical part of the document provides a reference of the protocol that was developed to extend IRC and briefly looks at the design and implementation of the TeamDraw modules.

1.2 Motivation

The Internet offers a place to not only find information but also communicate with other people without having to meet them in person. Services like E-mail have shown that communicating over the Internet makes sense and can be a real alternative to other communication means. However, while E-mail does a great job in replacing regular mail, it fails when it comes to holding virtual meetings over the Internet. For truly interactive communication, the information needs to be exchanged almost instantly between the users.

One service that allows such a direct communication is called IRC (Internet Relay Chat). IRC makes conversations possible that are similar to phone calls but based on text messages rather than voice. While this is much closer to a possible solution for an online meeting, there is still one important component missing: the white board or over-head projection which is used in meetings to draw sketches or diagrams or simply to be creative.

The goal of this project is the design and implementation of TeamDraw – an IRC client that is not limited to text based communication. The existing IRC protocol is extended by graphical functionality so users can talk to each other but also collaborate on creating simple drawings as shown in Figure 1.

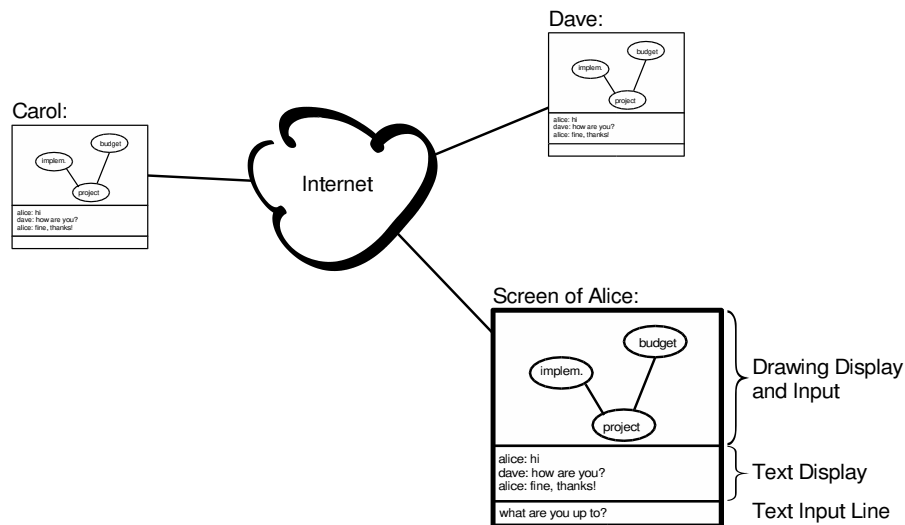


Figure 1: The basic idea behind this project is to allow users to interact in a textual and graphical way.

1.3 Example Applications

The possibilities of using graphical elements to share ideas over the Internet are almost endless. TeamDraw focuses on the following applications:

Sketching

- Explaining in simple sketches what is hard to phrase in words (e.g. the way to a location).

Technical Applications

- Creating mind maps in a team.
- Drawing technical diagrams and charts (e.g. UML, flow charts, software architectures, ...)

1.4 Task Description

This task outline is based on a project work description that was originally worded in German ¹.

The goal of this task is the design and reference implementation of a chat client with graphic capabilities. A protocol is designed to embed graphical entities in the existing IRC protocol.

A typical use case for such a system is a brainstorm session over the Internet. Users create graphical objects and texts with a graphic tablet or a mouse.

Any conventional IRC server can be used to connect the chat participants. The handling of the graphical entities can be achieved by implementing a special client ("IRC Bot").

A user client consists of a graphical and a textual input and output module. Graphical

¹<http://www-t.zhwin.ch/pada/Preview.jsp?ArbeitID=1438>

elements will most likely be constructed and transmitted as vector graphics, since the main applications are of a technical nature.

The communication between clients and servers uses the IRC protocol (RFC 1459). The graphical extensions are embedded in the payload of the IRC communication for example as an SVG snippets.

2 Instant Communication Over The Internet

This chapter explains the basics of instant communication over the Internet on the example of IRC (Internet Relay Chat). The full protocol is specified in RFC 1459 ².

2.1 How IRC Works

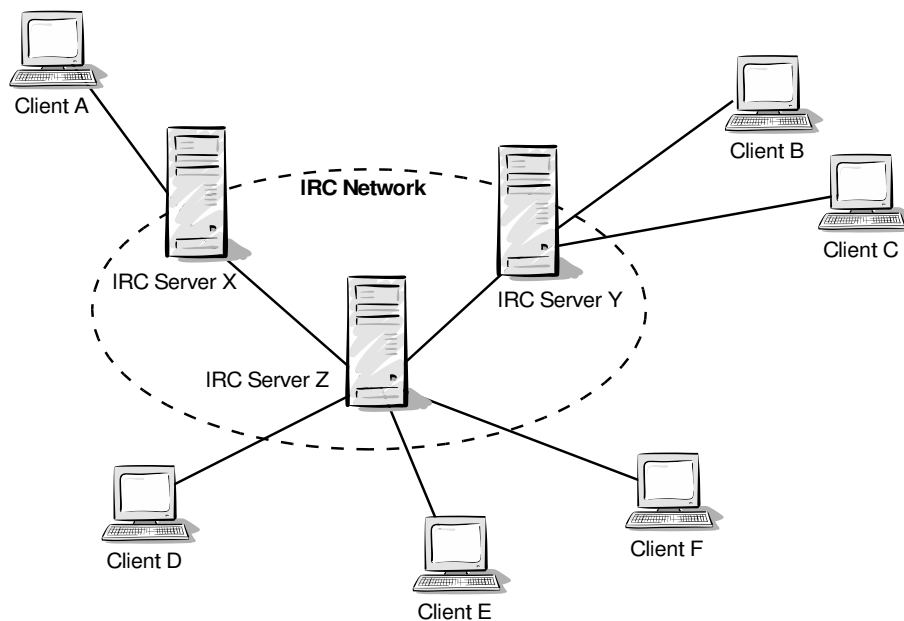


Figure 2: An IRC network with clients.

Figure 2 shows an overview over a typical IRC network with a couple of clients. The IRC network consists of one or more IRC servers which are configured in a way that they know about each other and form a spanning tree ³. Client C and Client D are both connected to the same IRC network although they are not connected to the same server. If, for example, Client C wants to talk to Client D, the message is sent to Server Y. Server Y forwards the message to Server Z which knows Client D and delivers the message.

Large scale IRC networks like "Undernet" ⁴ consist of between 20 and 50 servers and are serving over 100,000 clients at any time.

2.2 Who Uses IRC?

Although the vast majority of chatters use IRC for fun, there are also many serious applications for IRC. Many Open Source project teams use IRC as their main communication medium to organize their efforts. But managing international teams is also an important issue for any global organization or company. IRC is a service that can certainly help.

²<http://www.ietf.org/rfc/rfc1459.txt>

³Spanning trees prevent loops in a network.

⁴<http://www.undernet.org>

3 From IRC To TeamDraw

This chapter gives an overview of the main issues that have to be solved when introducing graphical content in an IRC client.

3.1 Messages vs. Documents

Conventional chat services allow users to send messages to each other. Once a message was sent and received, the system as such does not remember it. Although the system is not entirely stateless, the messages are not part of its internal state.

This model is not sufficient to share and modify documents like drawings. A client that disconnects from the network must have the possibility to receive the last state of the drawing again.

3.2 Stateful IRC And Persistence

There are two possibilities to add a state to an IRC system and make documents persistent:

- A) The clients exchange the data among each other to keep synchronized (see Figure 3).
- B) A special client acts as the master client which has the copy of the document that is relevant for all other clients (see Figure 4).

3.2.1 Clients With Equal Status

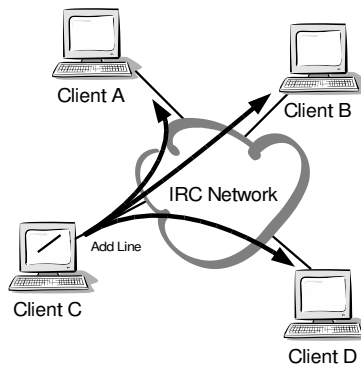


Figure 3: A system in which each client holds a copy of the drawing document. This copy can be modified by the other clients for example by adding lines.

Figure 3 shows a system in which all clients have an equal status. This architecture raises a number of issues:

- The drawings of the clients can easily run out of synchronization, for example if a client has to reconnect and misses some messages by doing that.
- It is not clearly defined how a client synchronizes with the others when joining a new team.

- If every client is responsible for the graphical elements he has created, that responsibility has to be handed over when a client leaves.
- When all clients are disconnected from the network, the drawing is no longer available to other users who might join at a later time.
- A client does not know which drawings the other clients are working on. Therefore all graphical messages would have to be sent to all clients, regardless of whether they work on the same drawing or not.

For these reasons, a centralized system with a clearly defined master appears to be more appropriate.

3.2.2 Centralized System With A Bot

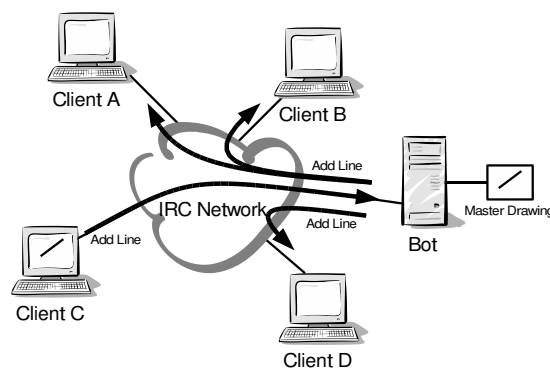


Figure 4: In TeamDraw, a system was implemented with only one relevant master copy of the document. This document is managed by a master client (a bot). All clients synchronize their local copy of the drawing with the master drawing.

For the reasons outlined in section 3.2.1, adding state to IRC is commonly done with so-called bots. These special automated clients can provide various services to the users. In TeamDraw, a bot processes messages which modify a drawing. If, for example, a user draws a line into the graphical area of his screen, a message is sent to the bot that contains the geometry of that line. The bot receives this message, adds the line to the drawing it belongs to and forwards that event to the other clients. The bot acts as the master of the system. It is clear at any time that a drawing has to match the master drawing on the bot to be synchronized.

3.3 Performance

Usually performance is not a big issue when transferring simple vector graphics over the Internet. However, IRC networks have to protect themselves from flooding⁵. They do this by limiting the maximum number of messages that can be sent in a given time frame (typically about one message per second). Users who do not obey to this rule usually get kicked or banned from a channel or even a whole network.

⁵Flooding occurs when a user sends out a large number of messages in a short time

Being able to send one message per second is not enough for the application of TeamDraw. Imagine a drawing with 120 graphical elements has to be sent to a user who selects that drawing to work on. Transferring the whole drawing to that user would take about two minutes which is hardly acceptable. Now imagine ten users joining a channel and selecting such a drawing at the same time. It would take the bot about 20 minutes to deliver the drawing to all users without flooding the IRC network.

For this reason, a standard IRC network is not suitable for working with TeamDraw. Luckily, a standalone IRC network that is set up for the use with TeamDraw and can only be accessed by trusted users can be easily tuned to allow a much better performance. Even drawings with well over 100 entities can be transferred to multiple users within seconds.

4 Application Requirements

This chapter lists all requirements that should be fulfilled by TeamDraw.

4.1 Functional Requirements

4.1.1 Basic Chat Functionality

TeamDraw must provide all basic functionality of a text-only chat client. The most important parts are:

Administrative Functionality

- Handling a list of servers (add, edit, remove servers)
- Connecting to multiple servers
- Joining multiple channels per server
- Displaying a user list for each channel

Core Functionality

- Handling channels (join, leave, create)
- Displaying channel messages
- Typing text messages for other users

4.1.2 Graphical Extensions

In addition to the basic text based communication, TeamDraw should offer the following graphical extensions:

- Share one graphical desktop per channel (vital)
- Set / change attributes of entities (optional):
 - Color
 - Owner
 - Width
 - Line style
- Drawing entities:
 - Lines (important)
 - Ellipses (optional)
 - Texts (important)
 - Arrows (optional)
 - Rectangles (optional)
- Modifying entities
 - Selecting entities (important)
 - Moving entities (important)
 - Scaling entities (optional)
 - Deleting entities (important)
- Viewing:
 - Auto zoom (important)

- Zoom in / out (important)
- Window Zoom (optional)
- Panning (important)
- Scrolling (vital)
- Highlight entities drawn by a specific user (optional)
- Highlight entities that are currently locked (selected by another user)
- Highlight entities that are currently locked (selected) by this user

4.1.3 Managing Drawings

- Drawings are persistent (important)
- Previous drawing sessions can be restored (important)
- All available drawing sessions can be listed (important)
- New drawings can be added (important)
- Existing drawings can be deleted (optional)
- The current drawing can be changed during a session (important)

4.1.4 Platforms

Description

The system must be portable to the following platforms:

- Linux (vital)
- Windows (vital)
- Mac OS X (optional)

4.2 Interface Requirements

4.2.1 Graphical User Interface Of The Client

The main components for the Graphical User Interface (GUI) of TeamDraw are:

Server Connection Views: A connection to a server is shown as a child window of the main application window in a similar way like documents are shown in other applications (see Figure 5).

Channel Views: Channels are visualized on tabs within the view of the server connection they belong to (see Figure 5). One tab is reserved for channel independent information that is received from the IRC server. Each other tab visualizes one channel.

Each channel tab must contain:

- Topic
- Document list (shows a list of available drawings and allows the user to change the current drawing)
- Drawing area (changing and viewing the current drawing)
- Drawing tools
- Text display area (shows all text messages from users in the same channel)
- Text input field to type messages for the other users
- Users list (shows a list of all users in the same channel)

Menus:

- File: Open local drawing, Export, Quit, New Connection
- Windows: Window handling
- Help: About

Dialogs:

- Connecting to servers
- Handling a server list
- Joining channels

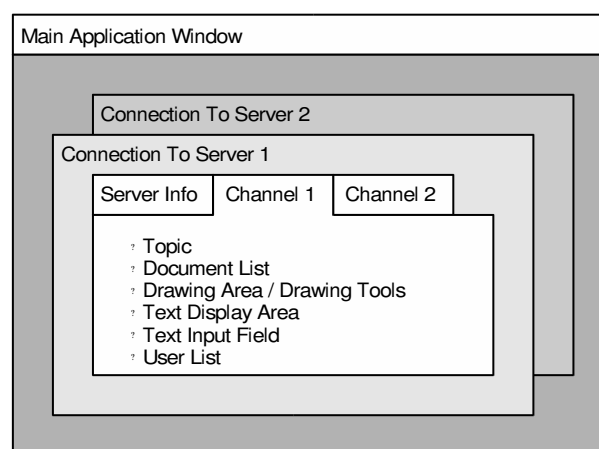


Figure 5: The fundamental components of the user interface.

5 Design

This chapter introduces the architecture of TeamDraw and the environment it requires. Further, the design of the TeamDraw modules is documented.

5.1 Architecture Overview

Figure 6 shows a typical system architecture in which TeamDraw can be used.

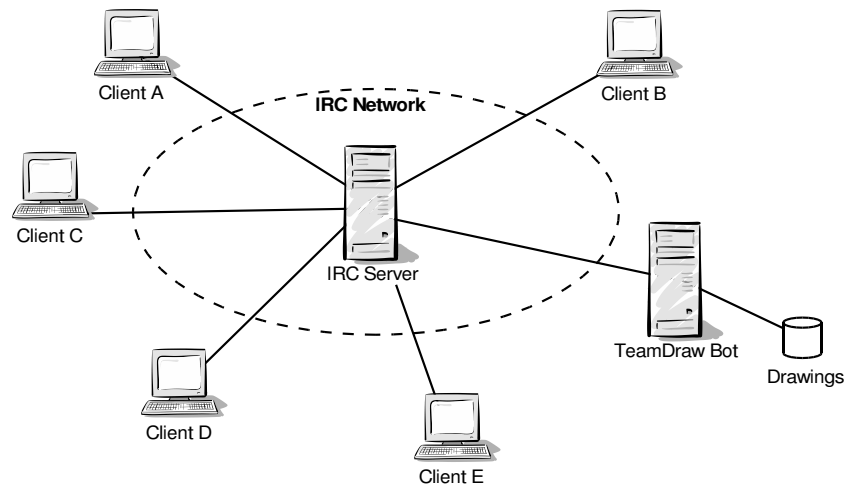


Figure 6: Architecture overview.

For the development and testing of TeamDraw only one IRC server was used. It can be safely assumed that TeamDraw would also work with multiple servers as shown in Figure 2, page 4. However, it may not be realistic for other reasons (such as performance, and administration) to use TeamDraw in large scale networks.

In Figure 6, the bot is running on a separate server in the network. Depending on the network infrastructure it might be more suitable to run the bot on the IRC server or on one of the client machines. All three scenarios are possible. Ideally, the bot should run on a machine that is permanently online.

5.2 Overall Design

TeamDraw is split up into six modules:

Libraries

- **libgraphics:** The graphics library of TeamDraw covers all functionality that exclusively deals with the construction of drawings. This library could be used in any application that requires a drawing engine. There are no network or IRC specific classes in this library.
- **libirc:** The IRC library was designed to implement the IRC protocol and the extensions that were required to implement TeamDraw.

- **libcore:** This library contains classes that represent the basic objects of TeamDraw (e.g. channels, users, server connections, ...).
- **libtools:** Classes that are commonly used in any kind of application (e.g. container classes, classes for debugging, classes to store and load application settings).

Applications

- **bot:** The implementation of the bot as a special client, who handles the master drawings. This is a console application.
- **teamdraw:** The implementation of the GUI for the graphical TeamDraw client.

5.3 Design Of The TeamDraw Modules

5.3.1 Graphics Library

The graphics library features a set of classes that can be used to implement an application with basic drawing functionality. It is designed after the Model-View-Controller (MVC) architecture.

Model

The model consists of:

- An entity container which holds all entities (graphical elements)
- Classes that define entities (Lines, Ellipses, Texts).

The entity classes define the behavior, geometry and look of an entity. The idea behind this design is that new entity types can be easily supported by adding a new entity class which implements all functions that are declared by the *TDR_Entity*⁶ interface.

The design of the model classes of TeamDraw is shown at the right in Figure 7.

View

The view is a class that can display an entity container with scroll bars (*TDR_GraphicView*). It offers methods for defining the current scaling factor and position of the viewport that is currently displayed.

Each view is associated to an action handler (*TDR_ActionHandler*, Figure 7). The action handler knows what kind of activity the user is currently doing (e.g. drawing lines, panning, ...). It also links the view to the controller classes.

⁶All classes of TeamDraw use the prefix *TDR_*.

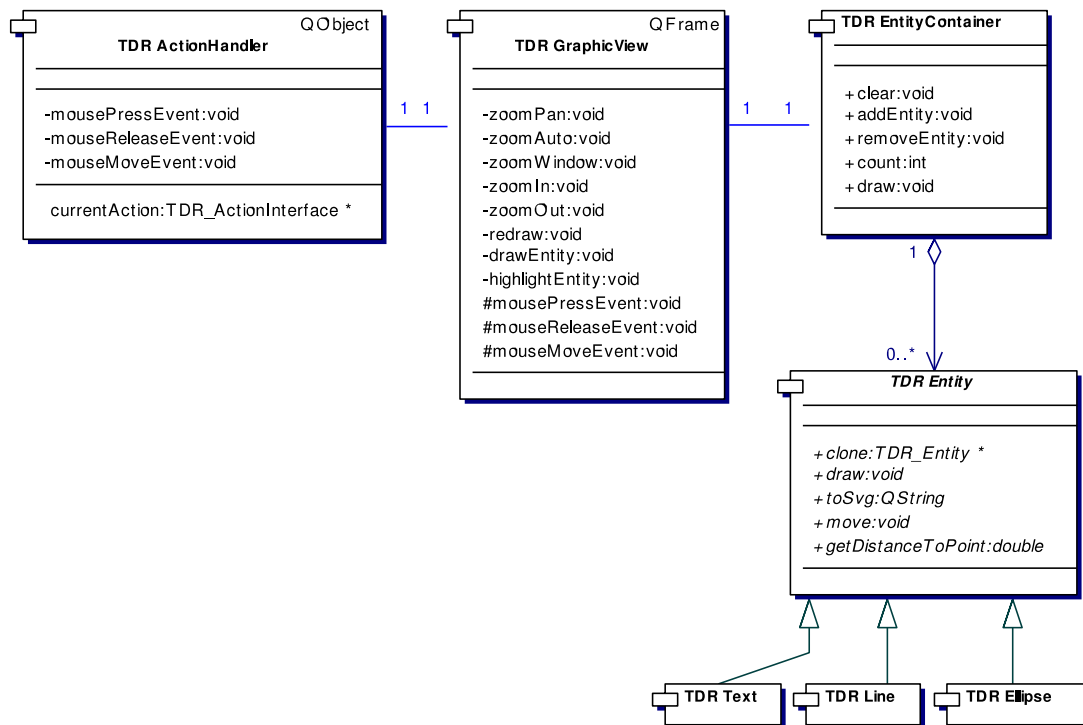


Figure 7: Graphics library design: model and view classes.

Controller

The controller classes shown in Figure 8 handle all user interactions to create and modify entities and to change the view. These controller classes are also called "actions" in TeamDraw. Actions are handled by the action handler class as previously mentioned.

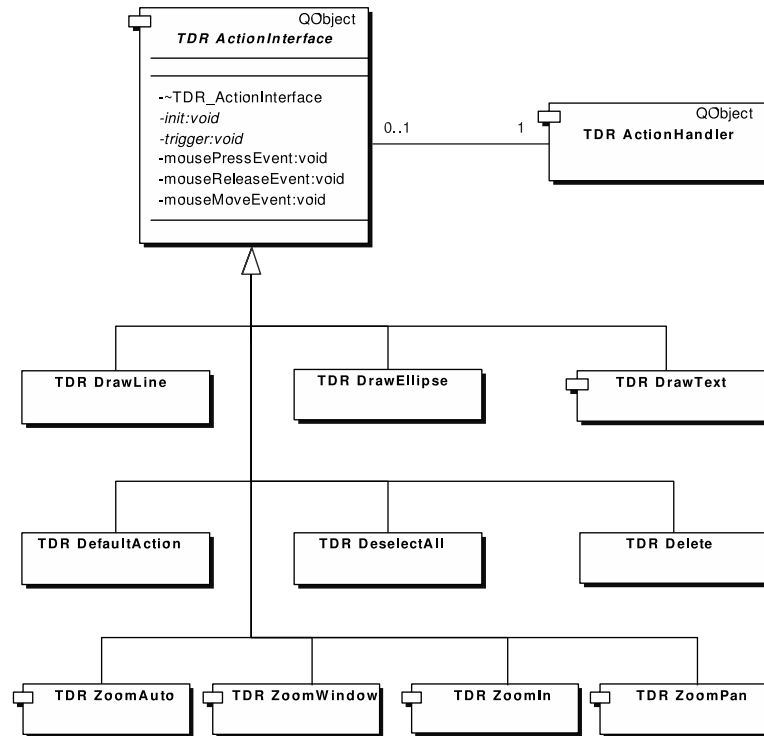


Figure 8: Graphics library design: controller classes.

If, for example, the user wants to draw a line, the current action is set to *TDR_DrawLine*. All events received by the graphic view are forwarded to the action handler which dispatches them to the current action class *TDR_DrawLine*. This class handles the events by setting the end points or showing a preview. Figure 9 shows the sequence diagram of the *TDR_DrawLine* action with the classes involved.

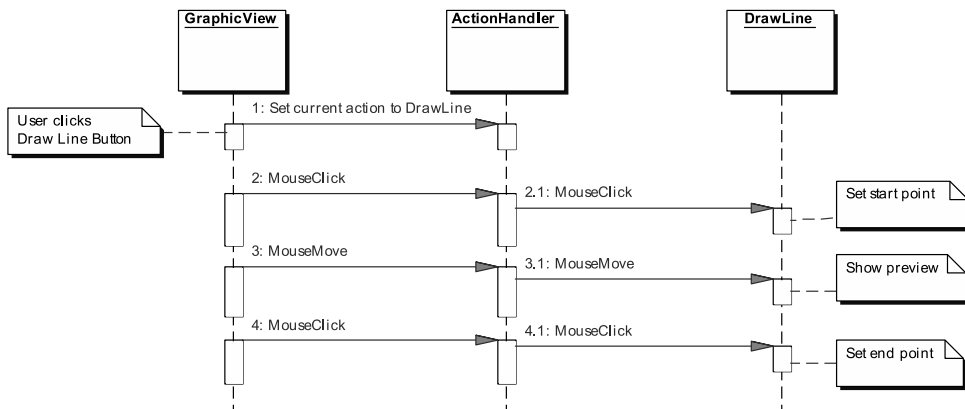


Figure 9: Sequence diagram for drawing a line.

Separate from the actions, there is a controller class *TDR_FileIO*, that can store an entity container to a file or load it from a file. The format that is used is Scalable Vector Graphics (SVG) [7].

5.3.2 IRC Library

The IRC library implements the IRC protocol and offers various TeamDraw specific extensions for sending graphical entities over IRC. The main components of the library are:

- *TDR_IrcConnection*: The IRC connection class manages the socket to the server and offers an extensive interface for sending and receiving messages.
- *TDR_IrcMessage*: A thin wrapper around an IRC message (either a TeamDraw specific command, a normal IRC message or a message that contains graphical information).

Further there are helper classes that implement message queues. Message queues are used to hold back messages to prevent the client from flooding the IRC server (see also 3.3).

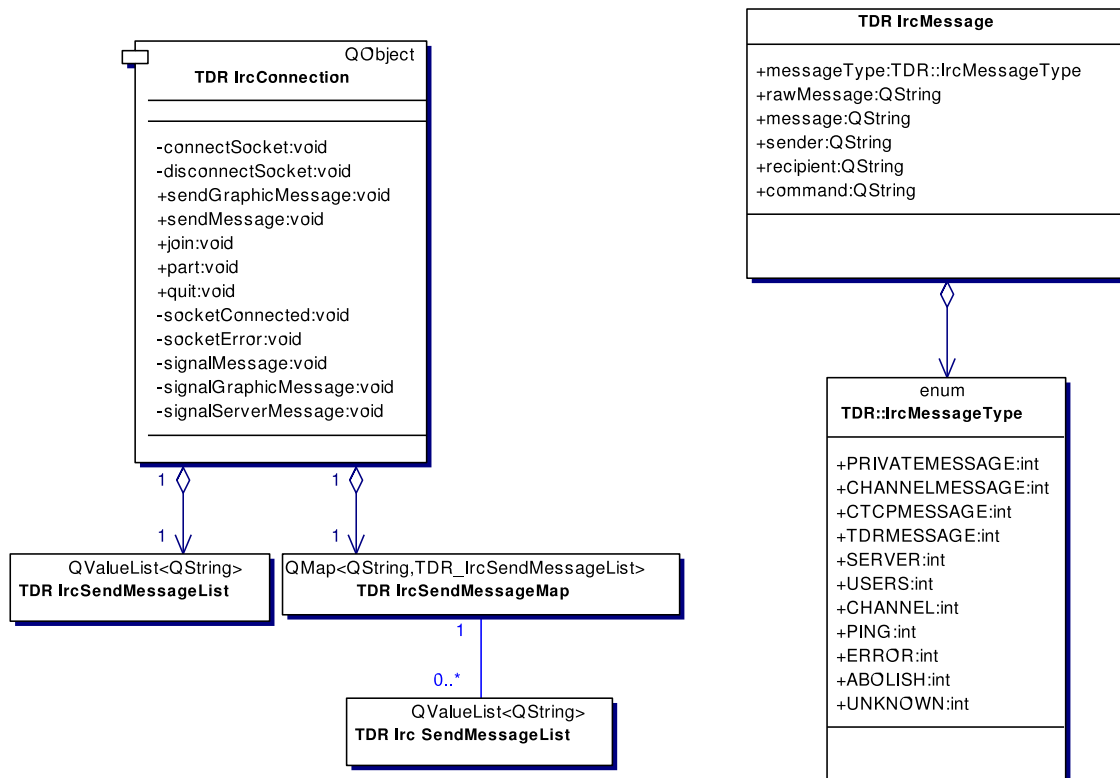


Figure 10: IRC library design.

5.3.3 Core Library

The core library offers a few generic data structures for the implementation of a chat application with server connections, channels, users and drawings. The TeamDraw client as well as the bot both internally store a structure as shown in Figure 11 to manage these objects.

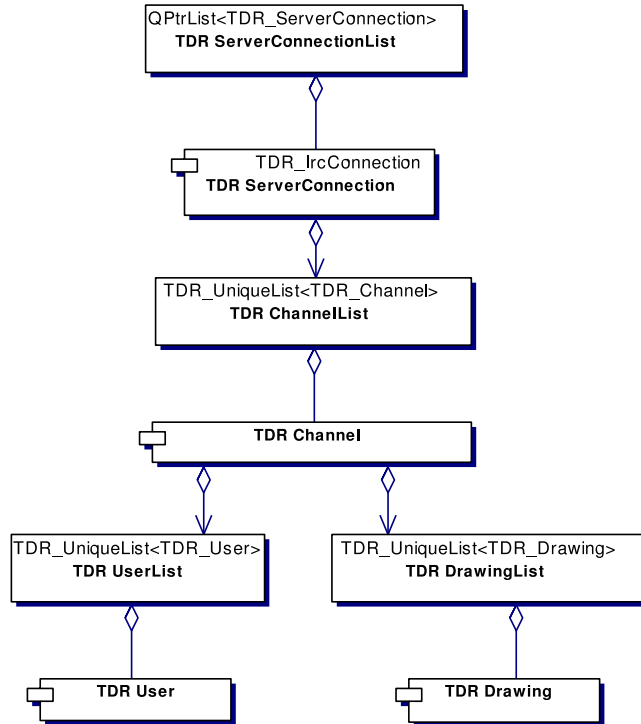


Figure 11: Core library design.

5.3.4 Application GUI

The GUI of the TeamDraw client is implemented as specified in section 4.2 **Interface Requirements**, page 10.

The main window class *TDR_ApplicationWindow* creates and manages one or more *TDR_MDIWindows* for every server connection.

If the client is connected to a server, the *TDR_MDIWindow* contains the widgets for working on a remote drawing as shown in Figure 12.

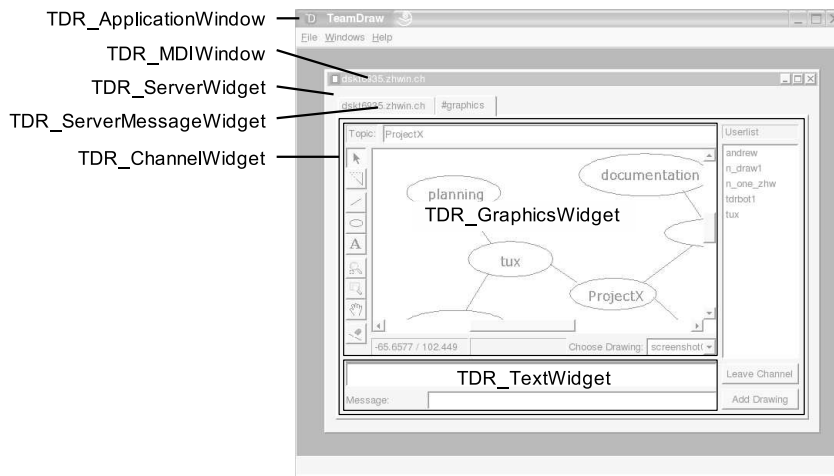


Figure 12: The classes of the TeamDraw GUI in client mode.

A *TDR_ServerWidget* consists of one *TDR_ServerMessageWidget* and one or more *TDR_ChannelWidgets*. The *TDR_ServerMessageWidget* displays information about and from the IRC server.

A *TDR_ChannelWidget* represents one channel (chat room). It contains a *TDR_TextWidget*, a *TDR_GraphicsWidget* and further a user list and buttons for leaving the channel and adding drawings. The text area (*TDR_TextWidget*) works like in common IRC clients and is just used for text based communication. The graphics widget serves as a drawing canvas for drawing and displaying graphical documents.

In addition to the scenario above, TeamDraw offers an alternative mode to create and edit local drawings without connecting to a server. In this mode, the *TDR_MDIWindow* only contains a graphics widget as shown in Figure 13.

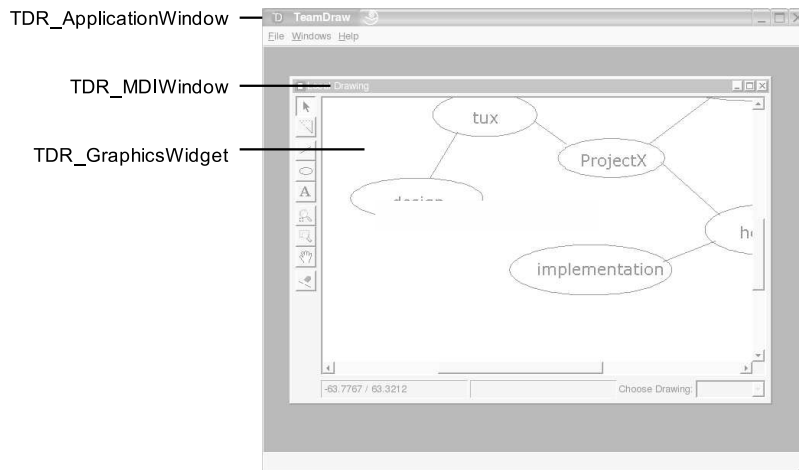


Figure 13: The classes of the TeamDraw GUI in the standalone mode.

For the sake of completeness, Figure 14 shows the class diagram of the TeamDraw GUI.

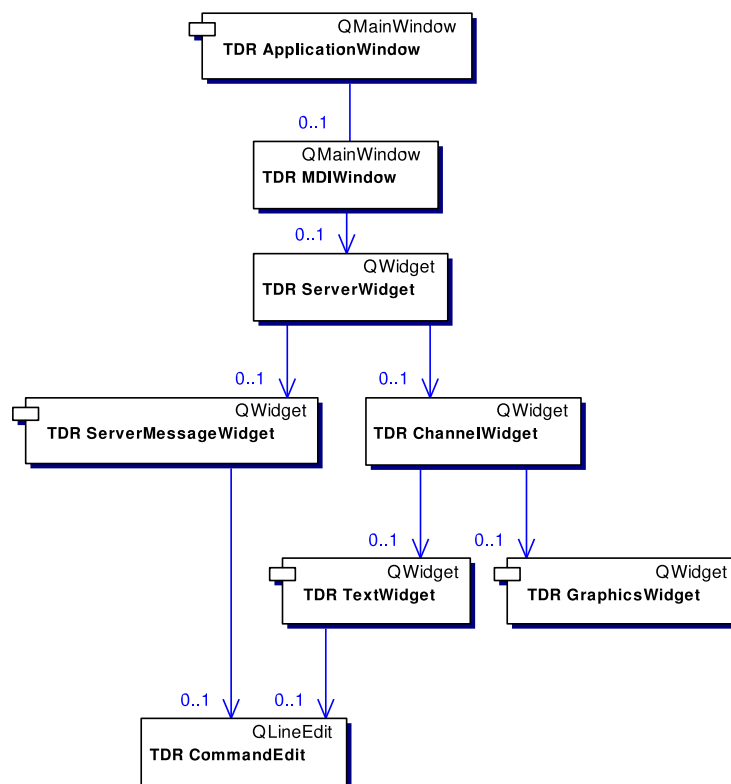


Figure 14: Application GUI design.

6 The TeamDraw Client

This chapter is a brief introduction to get started with the TeamDraw client.

6.1 The Main Application Window

The TeamDraw user interface presents itself as an empty window after startup. Usually the first thing a user does after starting TeamDraw is to connect to an IRC server by selecting the menu: 'File' -> 'New Connection'. This menu shows the server connection dialog (Figure 15).

6.2 Connecting To An IRC Server

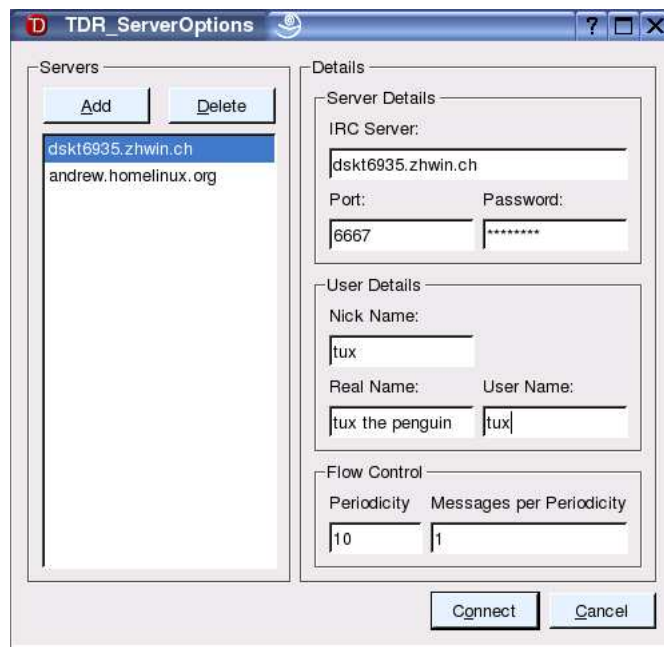


Figure 15: The Server connections dialog assists the user to configure server connections and to connect to a previously configured server.

To add a new server connection and connect to it, please proceed as follows:

1. Click the 'Add' button at the top to add a new server to the list of your favorite IRC servers.
2. Type in the information at the right as described in Table 1
3. Click the 'Connect' button

The new connection will later be available in the server list at the left, named after the "IRC Server".

Parameter:	Description:	Example value:
IRC Server	'IRC Server' specifies the URL of the server	teamdraw.dnsalias.org
IRC Port	To connect an IRC server, the port to reach the service needs to be specified. Public IRC servers are usually reachable over the default IRC port 6667.	6667
IRC Password	The access to an IRC Server can be restricted with a server password.	
Nickname	Nickname is the name which identifies you on the IRC server. Other users on IRC will know you by that name. The nickname is limited to 9 characters.	tux
Realname	The entered real name can also be seen by other people on IRC. However, unlike the nickname it does not have to be unique in the IRC network.	Tux the Penguin
Username	The username is used by the IRC Servers.	tux
Periodicity	The periodicity specifies the time span between sending messages in milliseconds. The configuration of the periodicity is needed to prevent an "Excess Flood" on the IRC server. The value is dependent on the configuration of the IRC server.	1100
Messages per period	The messages per period defines the number of messages sent every period. The configuration depends on the IRC server and on the configured periodicity.	1

Table 1: Server connection parameters.

6.3 Channel Dialog

After a connection was successfully established to an IRC server, the user has to join a chat channel before he can communicate with other users. A user can join more than one channel per server connection. Directly after connecting to the server, the channel dialog is shown to join the first channel.

To join an additional channel, the channel dialog can be launched by clicking the 'Join Channel' button in the server messages tab.

6.4 Joining And Leaving A Channel

Immediately after joining a channel, the drawing area of the channel widget is disabled. To enable it, the user first has to select the drawing he wants to work on. This is done by selecting the name of one of the available drawings in the combo box at the right side below



Figure 16: The channel dialog is shown to configure and join channels.

the drawing area.

Alternatively, the user can create a new drawing by clicking the button 'Add Drawing'.

The button 'Leave Channel' closes the current channel widget and leaves the channel when pressed.

The TeamDraw client is split up into two sections: text based communication and graphical communication.

6.5 Text Based Communication

The text area below the drawing area provides the basic functionality of a conventional IRC client (see Figure 17).

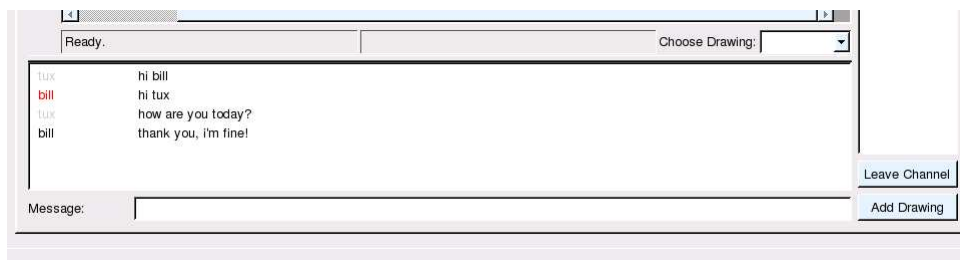


Figure 17: The text area of TeamDraw.

6.6 Graphical Communication

The drawing area of TeamDraw provides a canvas that is shared among all users working on the same drawing.

The combo box at the right below the drawing area (see Figure 18) allows the user to change the drawing to work on at any time. If there are no drawings available, there might be no drawings in this particular channel or no TeamDraw bot is available in that channel.

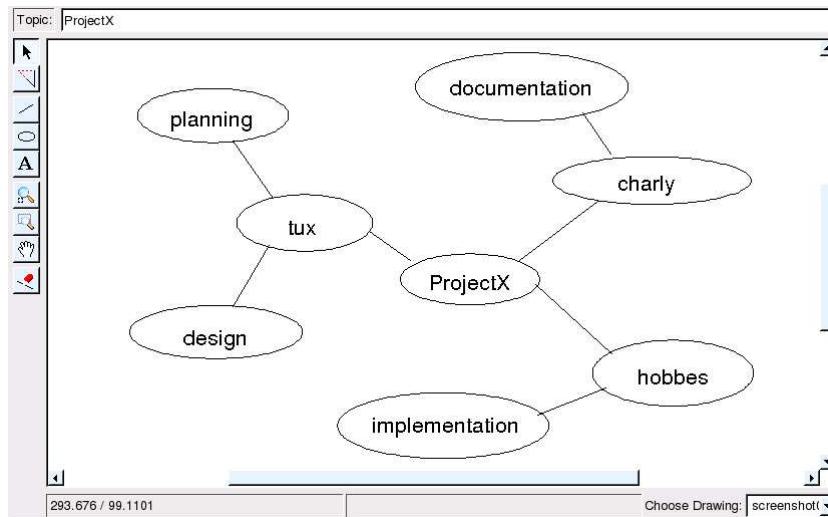


Figure 18: The graphics area for graphical communication.

6.6.1 Tools










Icon:	Tool:	Description:
	Select	Select one or more elements and drag them around on the drawing area.
	Deselect All	Deselect all elements
	Line	Draw line: First click sets the starting point, the second click sets the ending point.
	Ellipse	Draw ellipse: First click sets the left point at the head, the second click sets the lower right point.
	Text	Sets a text on the Drawing Area. The size of the text can be entered in the dialog box.
	Auto Zoom	Automatically zooms to have a full view of the drawing.
	Window Zoom	Select a section to be zoomed.
	Panning	Free panning on the drawing area.
	Delete	Delete the selected items (see section 6.6.3).

Table 2: TeamDraw tools.

6.6.2 Highlighting

TeamDraw has a feature to show what each user has contributed to the current drawing. A click on the nick name in the user list highlights the user's elements as shown in Figure 19.

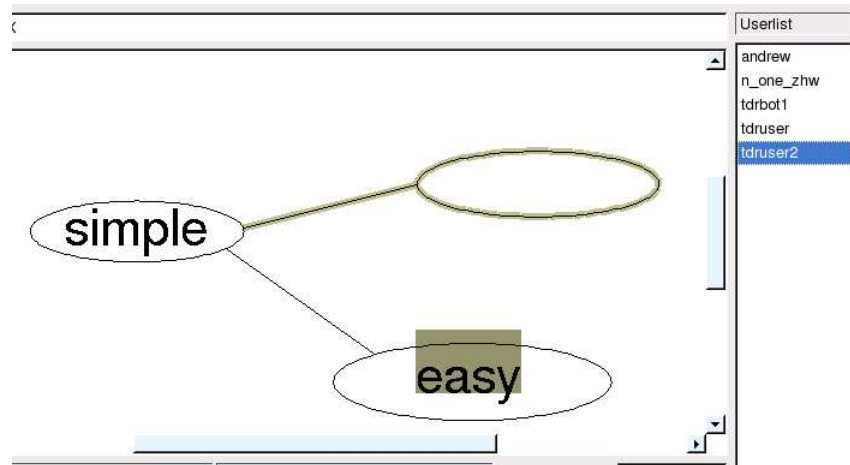


Figure 19: Highlighting in TeamDraw.

6.6.3 Selection

With the arrow tool the user can select one or more elements in the drawing. The selected elements are highlighted in red color for the user who has selected them. For all other users these elements appear in blue color. Blue elements are not editable anymore for the others until the user deselects them again.

An element can only be edited (moved, deleted) if the user was able to select it successfully.

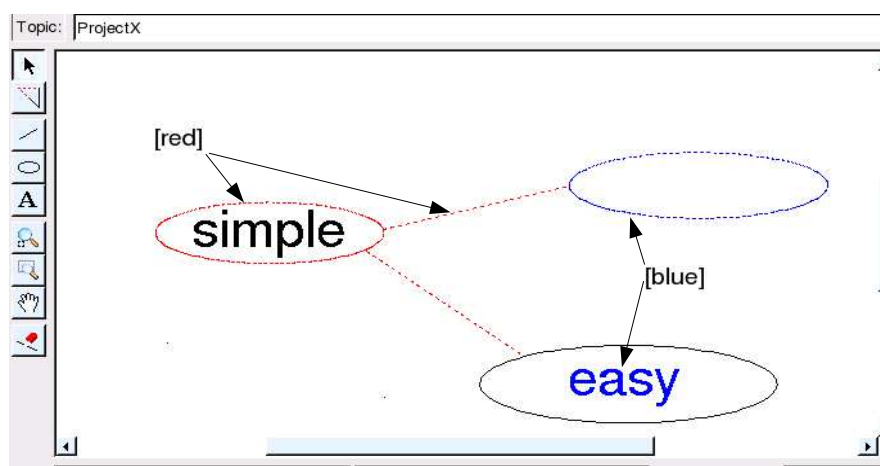


Figure 20: Selection in TeamDraw.

7 The TeamDraw Bot

In this chapter the implementation of the bot is discussed. The bot deals with graphical messages and the belonging user handling. The last chapter introduced you to the possibilities of the client, the bot serves all the needs of the client.

The topics covered in this chapter are:

- Locking Graphical Entities For Modification
- Persistence Of Drawings
- Receiving And Forwarding Graphical Messages
- Optimized Distribution Of Messages
- Distinguishing Users
- Configuration And Usage Of The Bot
- IRC Commands For The Bot
- Settings File

7.1 Locking Graphical Entities For Modification

A client sends a message to the bot to acquire or release the lock on a graphical entity. The client has to wait until the bot returned a message about the locking state. The owner of an entity can always release the lock on that entity. After a change of the locking state on the bot, it will forward the state to the other users of the drawing. Locking is needed for modification or deleting of entities.

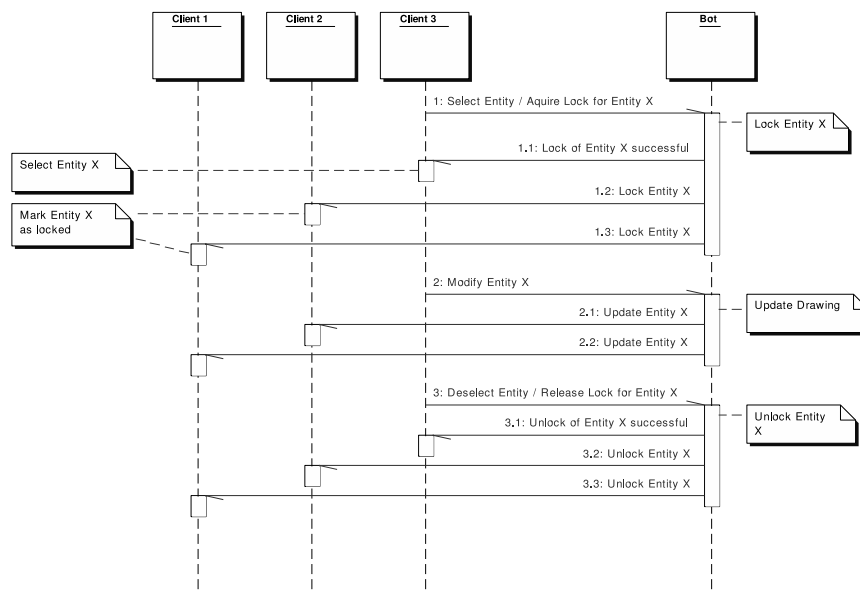


Figure 21: Locking and modification sequence.

7.2 Persistence Of Drawings

The persistence of drawings is solved with periodic saving. The bot keeps the drawings in the memory and saves them in intervals to the hard drive. On exit, the bot saves the drawings. On restart, the bot searches for drawings in the save path and joins the required channels. After a successful join, it will load the drawings for the channel.

7.3 Receiving And Forwarding Graphical Messages

Clients send graphical entities to the bot. The bot saves the entities to the drawing and forwards the message to the other users.

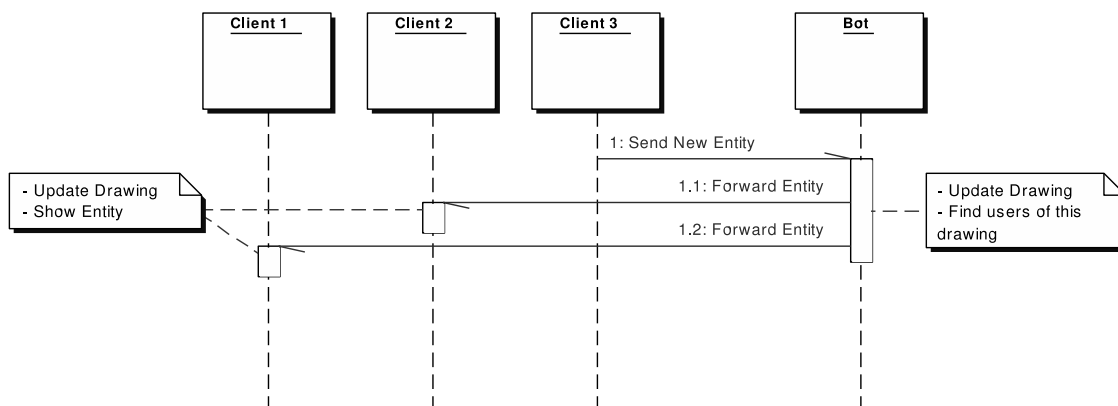


Figure 22: Sequence for adding graphical entities.

7.4 Optimized Distribution Of Messages

The bot can become a bottleneck, depending on the amount of users to serve. For instance, if a user activates a big drawing, the bot transmits the drawing to this user. This causes a big delay for all other users, they have to wait until the bot has sent all entities. To prevent this situation and serve users on a fair basis, a cyclic allocation or better known as Round Robin is used to forward graphical messages. Common IRC messages are in a higher priority queue to prevent delays in communication with the bot.

To give an overview of the round robin process, Figure 23 shows the structure of the queues. A queue with higher priority messages is tracked in *priorityMessageQueue*. If that queue is empty, Round Robin is used on the *userMessageMap*. This prevents starvation of users and makes a fair service.

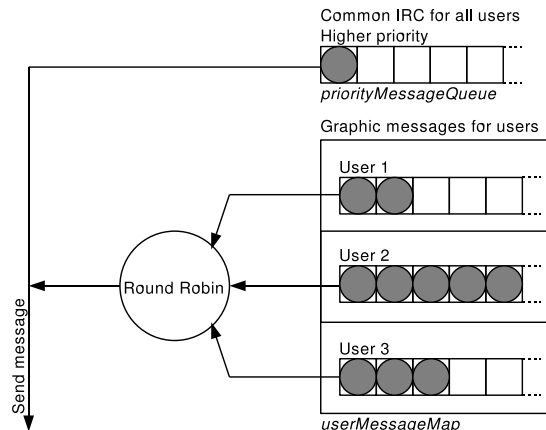


Figure 23: Round robin.

Implementation of round robin in **pseudo code**:

Initialize:

```

userMessageMap      (map of message queues, contains a message queue
                    for every user e.g. userMessageMap["tux"])
priorityMessageQueue (queue of messages with priority)
numberOfMessages    (number of messages to send)

```

Add Messages (invoked when a message arrives):

```

if(the received message is a graphical message) {
    add message to userMessageMap into the queue of the appropriate user
} else {
    add message to priorityMessageQueue
}

```

Send Messages (invoked periodically):

```

for(int i=0; i<numberOfMessages; i++) {
    if(Message in priorityMessageQueue) {
        transmit first Message of priorityMessageQueue
        remove first Message from priorityMessageQueue
    } else {
        do {
            find next non empty message queue
            transmit first message of userMessageMap[user]
            remove first message from userMessageMap[user]
        } while (round not completed)
    }
}

```

7.5 Distinguishing Users

The TeamDraw bot needs to be able to distinguish TeamDraw users, conventional IRC users and other TeamDraw bots. A bot registers users who join a channel and figures out if they are TeamDraw enabled. The bot sends a specific message to ask the new client about its TeamDraw capability. A client, which implements the graphical functionality, will reply with a specific message. A common IRC client will not reply. This prevents common IRC clients from receiving unneeded messages.

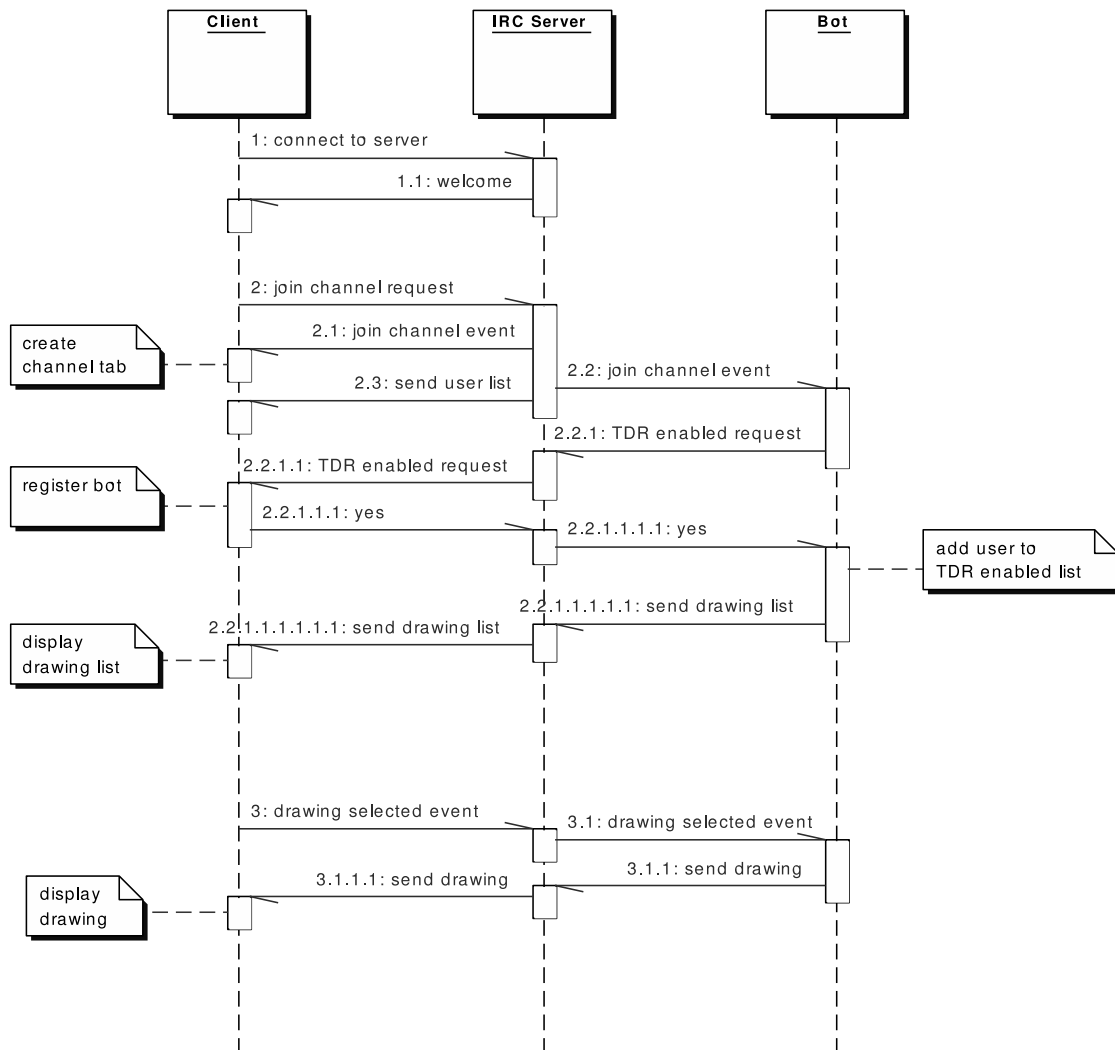


Figure 24: Login sequence of a graphics enabled user.

There needs to be one single bot in a channel to control the TeamDraw functionality. The first bot, who joined a channel will become the owner of that channel. Another bot who tries to join that channel will part automatically.

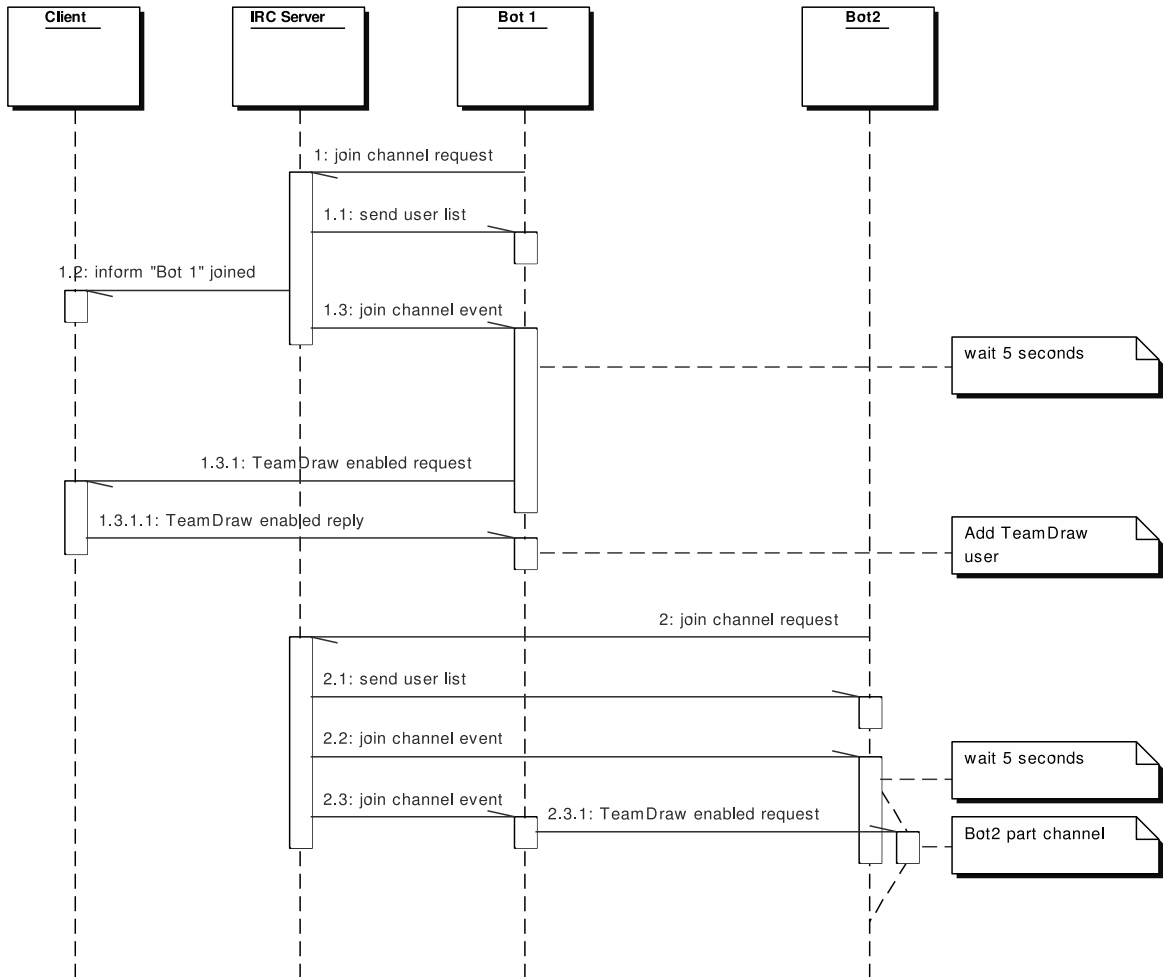


Figure 25: Sequence when a bot joins a channel where already another bot exists.

7.6 Configuration And Usage Of The Bot

Drawings are managed and held persistent by the bot. The preferred working environment of the bot is on the server, this is advantageous due to speed and simplification for the user. However, the bot can be started on the client side. This does not force the server administrator to install and support the bot. Running the bot on the client side is more error-prone. The bot is a console application and is currently available for Linux only. The configuration of the bot is saved in a settings file.

Running the bot in the console with `./tdrbot -help` shows the available options:

Usage: `./tdrbot [OPTION]`

Run bot for TeamDraw to handle drawings.

Available options for tdrbot.

```
-c, --create-settings  Create a default settings file and exit
-h, --help             Display this help and exit
-u id, --use-setting id Start tdrbot with configuration 'id'
-v, --version          Output version information and exit
```

7.7 IRC Commands For The Bot

Information about the bot is retrieved as private messages. To query the bot, a common IRC client or TeamDraw can be used. The following commands are available on the bot to get status information or for runtime configuration.

Command	Description	Example
active	List of users and their active drawing on a channel.	<code>/msg tdrbot active #graphics</code>
add	Add a new drawing to a channel (bot has to be joined on that channel).	<code>/msg tdrbot add #graphics name</code>
help	Get available commands.	<code>/msg tdrbot help</code>
join	Join a new channel.	<code>/msg tdrbot join #graphics</code>
list	List of drawing.	<code>/msg tdrbot list</code>
part	Part a channel.	<code>/msg tdrbot part #graphics</code>
save	Save drawings.	<code>/msg tdrbot save</code>
users	Get list of users on bot.	<code>/msg tdrbot users</code>
version	Get version of the bot.	<code>/msg tdrbot version</code>

Table 3: Overview of bot commands.

7.8 Settings File

For the first time running the bot, it is recommended to create a default settings file with `./tdrbot -c`. The settings file will be created in the users home directory (e.g. `/home/username/.tdr/tdr_botrc`). To change the settings, a text editor is needed.

Content of the default settings file:

```
[connection1]
channels/channel1=#graphics
drawings/path=/home/username/drawings
drawings/periodicity=5
flowcontrol/messages=1
flowcontrol/periodicity=20
serverdata/name=TeamDraw default setting
serverdata/password=somepass
serverdata/port=6667
serverdata/url=teamdraw.dnsalias.org
userdata/nickname=tdrbot
userdata/realname=TeamDraw Bot
userdata/username=tdrbot
```

The settings file is the source to configure the bot. The configurations are in the same style as already explained in the client section in table 1 on page 20.

Additional settings are explained in the following table:

Parameter:	Description:
[connectionX]	Specifies the configuration section (X starts with 1, a whole section can be copied, X needs to be incremented)
channels/channelY= #teamdraw	Specifies a channel Y to join on start (Y starts with 1 and needs to be incremented for every additional channel to join).
drawings/path= /home/username/drawings	Specifies the save path for the drawings.
drawings/periodicity=5	Specifies the periodicity to save drawings (in seconds).

Table 4: Bot options, settings file.

After changing the settings file, a restart of the bot is needed to apply the new configuration. The bot exits, when "Ctrl-C" is pressed.

The default connection settings used is [connection1]. To start the bot with another connection setting, the console switch `-c` is needed. For example, section [connection7] is used for connecting with the command `./tdrbot -c connection7`.

8 Project Management

8.1 Project Initiation

8.1.1 Choosing A License

From the very beginning of the project it was clear to us that TeamDraw will be developed under an open source license. The GPL⁷ was chosen primarily because it is well known and the most widely used open source license.

8.2 Documentation Tools

8.2.1 Source Code Documentation

The source code is documented in the sources themselves. Only an automated way of extracting this documentation and generating a set of HTML pages from it can be considered a solution. doxygen[4] was chosen for its compatibility with the well known JavaDoc utility and for the fact that it supports C++ and runs under Linux.

8.2.2 Technical Documentation

For the technical documentation of the project \LaTeX and OpenOffice were considered. Since we've used \LaTeX on various occasions before and have made very good experiences with it, the choice was rather clear. \LaTeX also has the advantage of being a plain text document format which can be managed by CVS. Further, it is very practical to automatically generate and include parts of the documentation. For diagrams and graphics, OpenOffice was chosen due to its good graphical capabilities and its built-in EPS support (a requirement needed for integration in \LaTeX documents).

Description	Priority	\LaTeX	OpenOffice
Mandatory			
Availability on Linux		✓	✓
Desired			
Version Control	10	10	1
Automated generation and inclusion of documentation parts	8	10	3
Efficiency	8	8	6
Know-how in the team	7	6	5
Points (\sum Priority · Points)		286	117

Table 5: Value benefit analysis for the technical documentation.

⁷<http://www.opensource.org/licenses/gpl-license.html> - the GPL.

8.3 Development Platform

Possible choices for a development platform were Windows and Linux. Both were available to all project members. The value benefit analysis in the following table is based on personal experiences, requirements and know-how. It is not a generic, representative analysis to compare Windows and Linux. The analysis is correct for this very project and our team.

Description	Priority	Windows	Linux
Mandatory			
Availability at school		✓	✓
Availability at home		✓	✓
Desired			
Tools (CVS, ssh, shell, vim, distcc, LaTeX, doxygen, compiler)	8	3	10
Stability, reliability	5	6	9
Effort to set up the working environment	7	3	10
Personal efficiency when working with this platform	8	3	10
Points (\sum Priority · Points)		75	195

Table 6: Value benefit analysis for the development platform.

8.4 Programming Language

Choices for the programming language were rather limited since only Java and C++ are known to all project members and learning a new language would have been an additional risk that could have compromised the schedule.

Description	Priority	Java	C++
Mandatory			
Availability of a compiler and API on Unices, Linux, Windows, Mac OS X		✓	✓
Desired			
Performance	5	4	8
Know-how in the team	7	6	7
Overall user experience for applications written in that language (installation, snappy GUI, ..)	8	6	10
Points (\sum Priority · Points)		110	169

Table 7: Value benefit analysis for the programming language.

The decisive factors for choosing C++ were the overall user experience that we find to be generally better with a C++ application.

8.5 GUI Toolkit

Portability was the main issue when choosing a GUI toolkit for TeamDraw. The application GUI must run under Linux but should also be available for Windows users. Coding the GUI twice was not considered to be an acceptable solution.

Qt[6] was chosen mainly for being a well designed and highly portable C++ GUI toolkit. There is also a good level of experience with Qt in our team. Qt is licensed under the GPL for Linux, Unix systems and Mac OS X. However, the Qt version for Windows is proprietary software.

8.6 Version Control System

CVS was chosen over the relatively new Subversion version control system. CVS was already known and used before by all team members. Further, CVS installations are available on public development platforms such as SourceForge.

8.7 Generation Of Executables

The use of makefiles is the preferred way over using IDEs to build TeamDraw. Makefiles can be generated with qmake (part of Qt) for a variety of platforms and compilers. qmake was chosen due to the simple configuration and because it was available with the Qt installation.

8.8 Graphics Format

TeamDraw internally uses a graphics format for storing drawings to the disk and to pack graphical information about entities into the textual payload of the IRC protocol (serialization).

SVG was chosen for the various advantages the XML based format offers over other vector based image formats:

- Internationalization (Unicode support)
- Easy parsing and manipulation through standard APIs
- SVG is a human readable text based format

8.9 Project Organization

8.9.1 Project Responsibilities and Deliverables

Laurent Cohn

- Design and Implementation of:
 - TeamDraw Application GUI (menus, dialogs, controls)
- Documentation
 - The TeamDraw Client (50%)
 - Design (Application GUI)
 - Abstract
- Ports
 - Windows

Thomas Jund

- Design and Implementation of:
 - IRC Library
 - Bot
 - Core Library (50%)
- Documentation
 - The TeamDraw Bot
 - Protocol Specification
 - Supported SVG Elements

Andrew Mustun

- Design and Implementation of:
 - Graphics Library
 - Tools Classes
 - Core Library (50%)
- Documentation
 - Introduction
 - Instant Communication Over The Internet
 - From IRC to TeamDraw
 - Design (Architecture Overview, Overall Design)
 - The TeamDraw Client (50%)
- Ports
 - Mac OS X

8.9.2 Project Schedule

Figure 26 shows the schedule as it was planned in the first phase of the project.

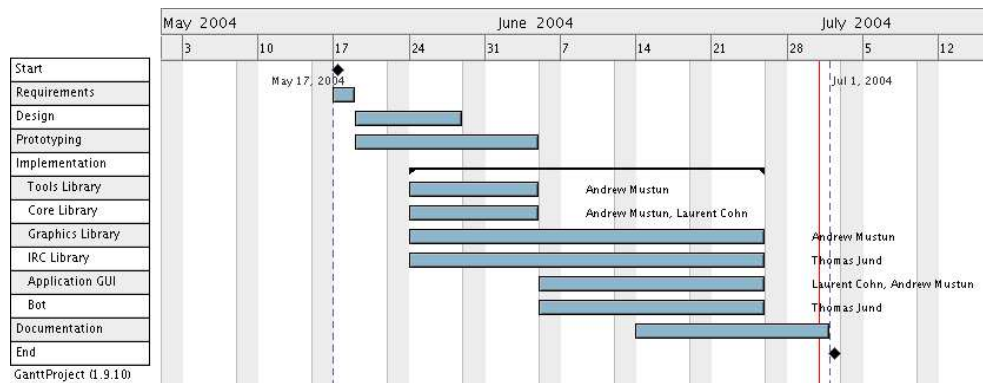


Figure 26: Planned schedule.

In Figure 27, the actual development efforts are visualized for the individual modules as well as for the documentation efforts. The statistics is based on the CVS logs and was generated with StatCvs [11].

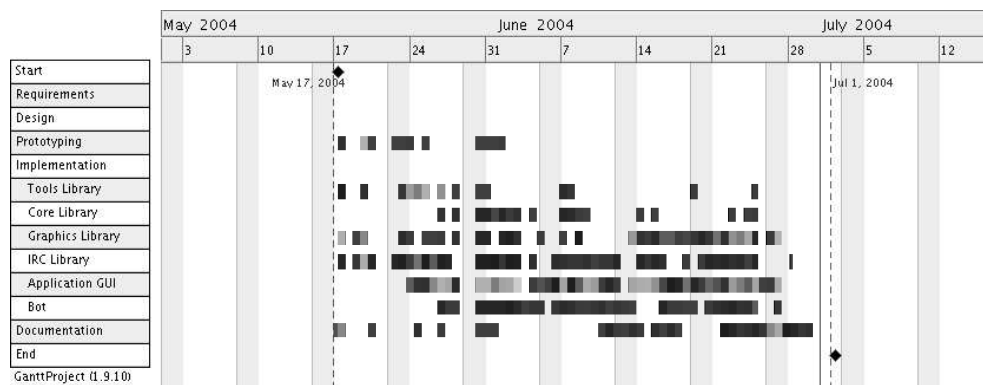


Figure 27: Measured progress by module.

9 Conclusion

In this section we look back on the first implementation of TeamDraw and discuss some thoughts about the technology we have used.

9.1 Room For Improvements And Additional Features

Due to the limited time frame for this project (7 weeks) we have restricted the project on implementing only the basic functionalities of a graphical IRC Client.

The following features have not been implemented but would be considered to be interesting for future versions:

- Loading drawings from clients to the bot.
- Opening a separate tab for private messages.
- Implementing security measures to protect drawings from being deleted by non-authorized users.
- Adding more drawing tools (poly line, rectangles, freehand lines, arrows, . . .)

There is still room for improvement in the following areas:

- Automatic reconnection of clients and bot (e.g. needed when IP changes)
- Improving the performance of the bot (for example with multi-threading to serve multiple users concurrently)

9.2 Significance Of The Project

The implementation proved that collaborating on creating drawings over the Internet is indeed realistic. In its current version, TeamDraw can be used for brainstorming or for fun projects. However, we believe that there are other possible areas of application.

9.3 Learning Process

Thomas primarily worked on the implementation of the IRC protocol and the network layer. He found working with IRC and CTCP to be the most useful parts of his work. Further, he has gained more experience with using UML and especially sequence diagrams.

Laurent deepened his knowledge about Qt and C++ programming by implementing parts of the GUI client. He also gained experience in using Together [?] to create class diagrams. While implementing the server options dialog, the complexity required a complete refactoring at a certain point which proved to be a good learning experience for Laurent.

Andrew found the project management to be the biggest challenge of the project. Since he already has a considerable amount of experience with software development, he was the one who was concerned with the organization and management of the team.

A Protocol Specification

This chapter is a description of the messages used by TeamDraw enabled clients and bots. To support the graphical functionality of TeamDraw on the network level, a protocol definition is needed. For instance, graphical messages must not disturb conventional IRC clients.

A.1 Message Details

TeamDraw works with the IRC protocol for text and graphical messages. Messages of TeamDraw are embedded in the Client-To-Client Protocol (CTCP). A message is extracted as string of octets from the CTCP payload. Bots and clients send messages to each other, which may or may not generate a reply. Some of the commands are just needed for replies. The IRC protocol is documented in [RFC 1459](http://ietf.org/rfc/rfc1459.txt?number=1459)⁸ and CTCP messages are defined in the [CTCP specification](http://www.irchelp.org/irchelp/rfc/ctcpspec.html)⁹.

A.2 Message Overview

An example message is used to give a general overview of the protocol encapsulation. In this example a user in channel "#teamdraw", sends a line to the bot with the nickname "team_bot" concerning the drawing "roadmap". This generates an IRC messages "PRIVMSG" to the bot (team_bot).

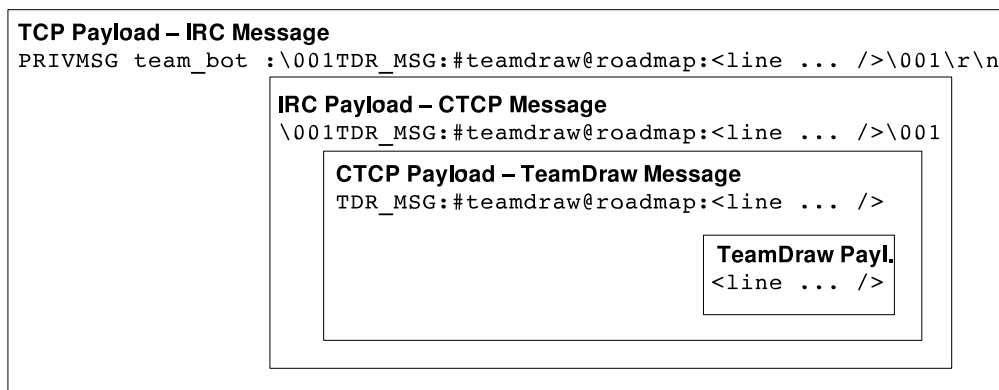


Figure 28: Message overview.

In the payload of the private message, the CTCP protocol is embedded. The escape sequence \001 in Figure 28 represents the ASCII character SOH (0x01). This character is used to identify CTCP messages. Details of TeamDraw messages are explained later in this chapter.

⁸<http://ietf.org/rfc/rfc1459.txt?number=1459>

⁹<http://www.irchelp.org/irchelp/rfc/ctcpspec.html>

A.2.1 Syntax Format

This section describes the format by an example. A detailed description of the syntax is provided in ABNF notation.

Example of a TeamDraw message to send a graphical entity (when the client adds a new line, this message is sent from the client to the bot):

TDR_MSG: #*teamdraw@roadmap* :< *linestyle* = " *stroke* : *rgb*(0,0,0);.../ >

1.
2.
3.

1. TeamDraw messages have the prefix "TDR_" and are followed by three characters to identify the command. In this case is the full command "TDR_MSG" which identifies a graphical message.
2. Parameters are identified by a colon (":") after the prefix. For example, the graphical message needs its drawing identification. In some cases more than one parameter is needed, they are concatenated with a colon in advance. This restricts parameters to strings without colons.
3. This part of the message is the payload. Valid characters are restricted due to the IRC protocol. For instance, Carriage Return and New Line characters are not allowed in the payload. The leading colon identifies the payload. In most cases the payload contains an SVG snippet.

A.2.2 ABNF Notation

The extracted message is parsed into the components <prefix>, <command>, <parameters> and <payload>.

```

message    = prefix command [params] [payload]
prefix     = "TDR_"
command    = 3*3(ALPHA | "?") [ "_" 1*1ALPHA ]
params     = ":" *noctl
payload    = ":" *TEXT

```

```

noctl     = %x0B-0C / %x0E-1F / %x21-39 / %x3B-FF
           ; excluding special chars like NUL, CR, LF, " ", ":", "@"

```

A.3 Message Details of TeamDraw Messages

A.3.1 Overview

Command	Type	Parameters/Content	Short Description
TDR_AC?	Request	Entity, Lock, Drawing ID	Acquire lock on graphical element
TDR_ACQ	Reply	Entity, Selector, Drawing ID	Acquisition status of lock
TDR_CHO	Request	Drawing ID	Choose drawing
TDR_CMP	Transmit	Drawing ID	Transmission of drawing completed
TDR_EN?	Request	Channel	Identify TeamDraw ability
TDR_ENA	Reply	Channel	Identification successful
TDR_JOI	Request	Channel	Join a channel (for bot)
TDR_LS?	Request	Channel	Request list of drawings
TDR_LST	Reply	Drawing list	List of drawing IDs
TDR_MSG	Transmit	SVG snippet	Graphical message
TDR_MSG_S	Transmit	SVG snippets	Start of long graphical message
TDR_MSG_C	Transmit	SVG snippets	Continuing of graphical message
TDR_MSG_E	Transmit	SVG snippets	End of long graphical message

Table 8: Overview of message details.

A.3.2 Acquire Request Message

Command:	TDR_AC?
Parameter:	Entity ID, Lock, Drawing ID
Payload:	None

Description:	The TDR_AC? command is used to lock and unlock graphical entities. Typically a client acquires a lock from a bot.
--------------	---

Example:	TDR_AC?:nickname_9:true:#teamdraw@roadmap
Reply:	TDR_ACQ from the bot.

A.3.3 Acquire Reply Message

Command:	TDR_ACQ
Parameter:	Entity ID, Selector, Drawing ID
Payload:	None

Description:	The TDR_ACQ is used to reply the state of a graphical entity. Typically the bot replies the state of an entity to a client.
--------------	---

Example:	TDR_ACQ:nickname_9:ownernick:#teamdraw@roadmap
Reply:	None

A.3.4 Choose Drawing Message

Command:	TDR_CHO
Parameter:	Drawing ID
Payload:	None
Description:	The TDR_CHO is used from the client to activate a drawing. Typically the bot sends the whole drawing as reply.
Example:	TDR_CHO:#teamdraw@roadmap
Reply:	Graphical entities with TDR_MSG from the bot.

A.3.5 Drawing Completed Message

Command:	TDR_CMP
Parameter:	Drawing ID
Payload:	None
Description:	The TDR_CMP is used to inform the user about the end of transmission of a selected drawing. Typically the bot sends it to the client which will make the drawing active.
Example:	TDR_CMP:#teamdraw@roadmap
Reply:	None

A.3.6 Registration Message

Command:	TDR_EN?
Parameter:	channel
Payload:	None
Description:	The TDR_EN? command is used to identify an TeamDraw enabled user on a channel. Typically the bot asks a user who joined a channel about his TeamDraw ability.
Example:	TDR_EN?:#teamdraw
Reply:	TDR_ENA from a graphics enabled client.

A.3.7 Registration Reply

Command:	TDR_ENA
Parameter:	Channel
Payload:	None
Description:	The TDR_ENA command is used to reply to a registration message. Typically the client sends the reply to the bot.
Example:	TDR_ENA:#teamdraw
Reply:	None

A.3.8 Join Message

Command:	TDR_JOI
Parameter:	Channel
Payload:	None
Description:	The TDR_JOI message is used to tell the bot to join a new channel.
Example:	TDR_JOI:#teamdraw
Reply:	None

A.3.9 Drawing list Request Message

Command:	TDR_LS?
Parameter:	channel
Payload:	None
Description:	The TDR_LS? message is used by clients to request a list of available drawings on a channel.
Example:	TDR_LS?:#teamdraw
Reply:	TDR_LST from the bot on that channel.

A.3.10 Drawing list Reply Message

Command:	TDR_LST
Parameter:	Whitespace separated list of Drawing IDs
Payload:	None
Description:	The TDR_LST message is used by a bot to send the list of available drawings on a channel to the clients.
Example:	TDR_LST:#teamdraw@roadmap1 #teamdraw@roadmap2
Reply:	None

A.3.11 Graphical Message

Command:	TDR_MSG
Parameter:	Drawing ID
Payload:	SVG snippet
Description:	The TDR_MSG is used to transmit a graphical entity.
Example:	TDR_MSG:#teamdraw@roadmap:<line style="stroke:rgb(0,0,0);... />
Reply:	None

A.3.12 Graphical Message Start

Command:	TDR_MSG_S
Parameter:	None
Payload:	Part of SVG snippet or SVG snippets
Description:	The TDR_MSG_S introduces the transmission of graphical entities.
Example:	TDR_MSG_S:<line style="stroke...
Reply:	None

A.3.13 Graphical Message Continue

Command:	TDR_MSG_C
Parameter:	None
Payload:	Part of SVG snippet or SVG snippets
Description:	The TDR_MSG_C continues the initiated transmission of graphical entities.
Example:	TDR_MSG_C:#teamdraw@roadmap:...roke:rgb(255,255,0);...
Reply:	None

A.3.14 Graphical Message End

Command:	TDR_MSG_E
Parameter:	Drawing ID
Payload:	Part of SVG snippet or SVG snippets
Description:	The TDR_MSG_E ends the transmission of graphical entities.
Example:	TDR_MSG_E:#teamdraw@roadmap:.../>
Reply:	None

B Supported SVG Elements

In its current version, TeamDraw supports three types of entities. In SVG they are represented by the following basic shapes:

Ellipse

The `<ellipse>` tag is used to serialize ellipses. Example SVG snippet for an ellipse entity:

```
<ellipse cx="0.61157" cy="0.0330579" rx="6.69421" ry="7.00826" stroke="#000000"
stroke-width="0" id="nickname_0" />
```

Line

The `<line>` tag is used to serialize lines. Example SVG snippet for a line entity:

```
<line x1="7.53719" y1="7.80165" x2="6.18182" y2="-7.07438" stroke="#000000"
stroke-width="0" id="nickname_1" />
```

Text

The `<text>` tag is used to serialize text elements. Example SVG snippet for a text entity:

```
<text x="5.5625" y="30.8000" font-family="Helvetica" font-size="12" fill="#000000"
id="nickname_1">Written Text</text>
```

The `id` attribute of each entity is used to store the owner (creator) of the entity. To keep IDs unique they also contain a counter which numbers the entities for every user.

C Glossary

ABNF: Internet technical specifications often need to define a format syntax and are free to employ whatever notation their authors deem useful. Over the years, a modified version of BNF, called ABNF, has been popular among many Internet specifications. It balances compactness and simplicity with reasonable representational power. The syntax format is defined in **RFC 2234**¹⁰. [3]

Channel: There can be many simultaneous discussions going on at once on one IRC server; each conversation is assigned a unique channel. Messages that are addressed to that channel can be read by all users who have joined that channel. Synonym: Chat Room.

Entity: This document refers to entities as graphical elements such as lines, circles, . . .

IRC Bot: An IRC bot is special IRC client which does not represent a real person but a computer program. Such programs are often used for monitoring or to add functionality to a channel that is not provided by standard IRC (e.g. user identification). In TeamDraw, a bot is used to manage and store drawings created by the users.

IRC Client: A user who wants to participate to an IRC conversation needs an interface to connect the IRC networks. Such interfaces are available as locally running software as well as web interfaces. There are a lot of freely available IRC clients for nearly any platform. Popular clients are mIRC (Windows), Ircl (Mac OS X) and XChat (Linux).

IRC: Internet Relay Chat – A protocol and a program type that allows participants to “chat” online in a live forum that usually centers around a common interest. IRC is the earliest form of online chat [10].

stateful: A system is stateful if it has the capability to maintain state, which means that it remembers what was previously done.

stateless: Stateless systems have no information about what occurred previously.

¹⁰<http://www.ietf.org/rfc/rfc2234.txt>

D References

- [1] *TeamDraw User Manual, 2004*,
T. Jund, L. Cohn and A. Mustun, ZHW, Winterthur
- [2] *Wikipedia*,
<http://www.wikipedia.org>
- [3] *Wilde's WWW Online Glossary*,
<http://dret.net/glossary/>
- [4] *doxygen*,
<http://www.doxygen.org>, Dimitri van Heesch
- [5] *DocBook*,
<http://www.oasis-open.org/docbook>, OASIS
- [6] *Qt - Multi-platform, C++ Application Framework*,
<http://www.trolltech.com>, Trolltech AS
- [7] *SVG - Scalable Vector Graphics*,
<http://www.w3.org/Graphics/SVG/>
- [8] *The C++ Programming Language, Third Edition*,
Bjarne Stroustrup, Addison Wesley
- [9] *Valgrind*,
<http://valgrind.kde.org>
- [10] *GetNetWise*,
<http://www.getnetwise.org>
- [11] *StatCvs*,
<http://statcvs-xml.berlios.de>

E CD ROM Contents

The CD ROM contains the following directory structure:

- **bin**
 - **bot**
 - * **linux:**
Executable for Linux (tested on SuSE 9.0)
 - **teamdraw**
 - * **linux:**
Executable for Linux (tested on SuSE 9.0)
 - * **windows:**
Executable for Windows (tested on XP)
- **doc**
 - **classref:**
Class reference documentation
 - **technical:**
Project report
- **res:**
RFC, Specifications
- **src:**
Source code
 - **bot**
 - **libcore**
 - **libgraphics**
 - **libirc**
 - **libtools**
 - **teamdraw**

F About the Authors

Thomas Jund, Andrew Mustun and Laurent Cohn are undergraduate students at the Zurich University of Applied Sciences Winterthur (ZHAW).

Thomas Jund received his certificate of ability as electrical draughtsman from the Swiss Federation in 1998. He worked in the area of computer aided design (CAD), documentation publishing and technical documentation services for a company specialized in building automation. In addition to his interests in networking and CAD, he has also been active in web design and web programming.

Andrew Mustun has been heavily involved in various large and middle sized open source projects since 1995. He is the founder of well known projects like Qcad, ManStyle and dxflib. Andrew's main interests are in the CAD / CAM area as well as in documentation and information management. He worked for 14 months on a collaboration solution for a UK based company which is specialized in document management for the building industry.

Laurent Cohn has his main interests in web programming and web design. He has designed several web pages for companies and well-known non-profit organizations. Laurent has a lot of programming experience with PHP and MySQL. His experience with working with graphics and his intuition for good design, combined with his C++ skills make him a great GUI programmer.