CoLab

System Requirements
and Project Plan

CS 3300

Group Seven

Liem, Johannes
Luongo, Matt
Martin, Chris
Overman, Pamela

February 5, 2008

Contents

1System Overview

1.1Overview (Executive Summary)

CoLab is a network-enabled utility for communication, brainstorming, and collaborative document authoring. It is centered around a basic chat-room style environment, and is enhanced with inclusion of several tools to facilitate idea sharing within small groups. These tools include a collaborative document editor and whiteboard drawing tool.

Users create and join communities which are hosted on a single server. All of a community's related data is stored on a server for this application, including revisions of documents, saved drawings, and chat logs.

Within a community interface, users can open multiple editors in new channels and co-author documents simultaneously. For example, several people may be working on a diagram in the drawing tool while discussing it in a text chat.

1.2Deliverables

The final product consists of two independent applications: a server and a client. Deliverables will consist of the full source code and scripts to build both the server and client applications.

1.3Definitions

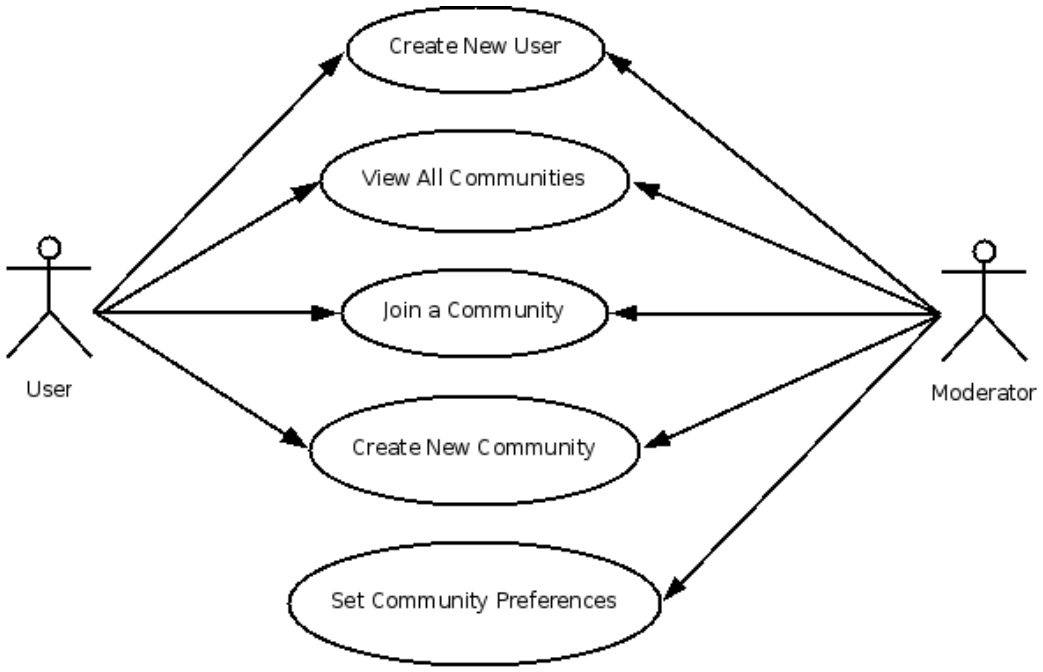
This application introduces several terms to cover collaborative groups and functions. A *community* represents a group of people (*users*) and a project on which they are collaborating, such as a university class or a company project group.

Users with the permission to control membership in the community and access to channels are called *moderators*. In a classroom setting, moderator status would be limited to instructors and teaching assistants (TAs). In a flat group with no distinct authority, all users may be moderators.

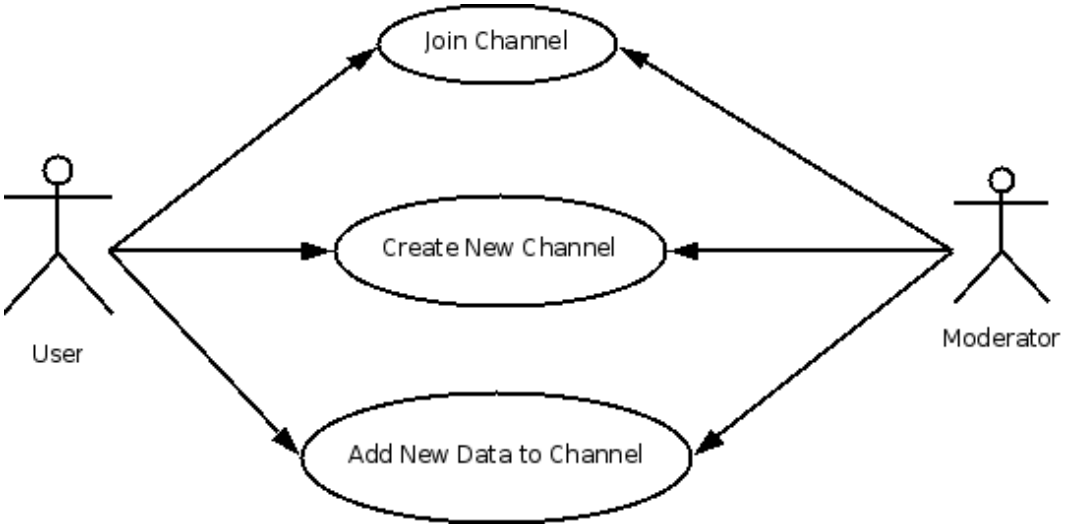
Each document or collaborative entity is referred to as a *channel*. A channel represents content being edited, and the workspace in which the group is dealing with it. A *protected channel* can only be modified by community moderators. Each channel is of a particular typed called a *channel protocol* which specifies the type of data represented by the channel and the way in which that data is displayed and manipulated by users. Examples of channel protocols are chat, document, and whiteboard.

2Use Cases

2.1Entire system usage, including setup



2.2Normal channel usage



3Scenarios

3.1Tutor and Student

Theodore is a tutor for CS 1331. Samuel, a student who lives off-campus, needs tutoring help but lives off-campus, so physically meeting with Theodore is inconvenient. Theodore decides to try some new software on the market called CoLab.

3.1.1Setup

Theodore downloads the executable CoLab program and installs it on his machine. Theodore runs the program and is presented with a login screen asking for a username, password and server IP, as well as a button to create a new user. He first enters the server IP he will use to run CoLab then clicks the "Create New User" button and enters a username and password of his choosing in text fields. The program checks that his username has not been taken, and if the username is valid, Theodore is logged in and is presented with a list of communities on this server. Theodore wishes to create his own community to tutor students, so he clicks a button to create a new community and enters the community's name ("Theodore's Tutoring"). Theodore is logged into the new community automatically, and opens the preferences dialog through the menu found on the main lobby screen. Theodore enters a new password for the community so users will be required to know the password before joining.

Samuel, the student, also downloads CoLab and installs it. He creates a new username and password like Theodore and after his new username is validated, Samuel is presented with a list of communities. Samuel picks "Theodore's Tutoring" from the list. Because Samuel is not a member of the community, he must enter a password, which Theodore tells him over the phone. Samuel enters the password in the text box and joins the community by clicking the "Join" button.

3.1.2Usage

Once Theodore and Samuel are in the same community, Theodore creates two new channels by clicking the "New Channel" button on the main screen, entering the channel name in a text box and selecting a type (whiteboard, document editor, chatroom) from a dropdown. Theodore creates a whiteboard and a chat room, and Samuel's screen shows him the two new channels. By double-clicking a channel name, a new window opens up on Samuel's screen with the appropriate GUI (a painting surface for the whiteboard, and a text box for the chat room).

Theodore explains the concepts of object-oriented programming by using the chat room to describe OOP textually while using the whiteboard to draw illustrations. Samuel receives the chat text as Theodore sends them, and receives the whiteboard updates in near real-time. Theodore and Samuel are pleased with the convenience of the all-in-one solution CoLab provides and are happy they bought the software.

3.2Student Group Meeting

Adam, Barbara, Cody, and Dieter are all in a CS 3300 class and are doing a software engineering project together. Instead of braving late nights in the CoC to get their documentation and design completed, they decide to use CoLab to meet.

3.2.1Community Setup

The four of them already have CoLab installed and set up on their home computers and have their own usernames, but there is no community established yet. Dieter is the first one to sign on around their meeting time. He opens the CoLab program and enters his log in information; when prompted for what community he wishes to enter, he opts to create a new community. He clicks the "Create New Community" button and calls the new community "ABCD." Once the ABCD community main screen opens, he navigates to the moderator preferences menu and sets up a joining password for ABCD of "3300xyz" and emails it to his group members. Once each of them logs in and selects the community name from the drop down menu of available communities, they each enter the password and are members. Before the meeting, Dieter opens the moderator preferences once more and selects Adam, Barbara, and Cody to each have moderator privileges.

3.2.2Usage

After logging in, all users see a "Main Lobby" screen with a chat box and a list of available channels. Barbara suggests in the chat box that they first design the logo for their program, Loom. Cody agrees and clicks the button that says "New Channel." He names it "LoomLogo" and chooses "Whiteboard" from a drop-down of channel types; the whiteboard opens on his screen. Barbara soon sees "LoomLogo" on the current channels list and double-clicks on it to open the whiteboard. Dieter wants to write use cases and opens a document editor by clicking on the "New Channel" button, choosing "Document" from the drop-down to create a new document channel. Dieter names it "Use Cases" and begins writing a use case. Cody also agrees, and opens "Use Cases" by double-clicking on its name from the list of ABCD's current channels shown on the community main screen. Adam had walked away from his computer for a few minutes after logging in and when he returns sees that two new

channels have been opened and added to the current channels list, a whiteboard channel called "LoomLogo" and a document editor channel called "Use Cases." Adam would rather design a logo than write use cases, so he clicks on "LoomLogo." The doodles that Barbara and Cody have been working on load and Adam types in the chat box that he likes one of the sketches more than the other, then proceeds to add a few of his own flourishes to it.

Since Dieter is a fast but sloppy typist, he makes lot of typos, so while Dieter types out a use case, Cody reads along and fixes typos. Barbara soon gets tired of drawing logos and realizes that they have not started on a UML diagram. She opens another whiteboard channel and names it "UML." She types in the chat box that she had forgotten some of the class names, and Adam replies that he had a list of their classes along with their CRC information in a text file. Adam opens the text file on his computer and copies the text then pastes it into a new document editor channel he names "CRC." Barbara opens "CRC" and begins working on the "UML" whiteboard with the "CRC" information. Adam at this point feels that the logo he designed is perfect and doesn't want anyone else to change anything on it, so he clicks the "Export" button to save it as a PNG on his desktop. He then goes to the moderator preferences screen, selects his channel and changes its status to "Protected," which means only moderators can modify it. Finally, Adam closes the "LoomLogo" channel on his screen.

Once Dieter and Cody finish a couple of use cases, Dieter opens the "LoomLogo" and compliments Adam on the logo. Dieter then opens "CRC" and "UML" and starts giving Barbara some guidance since he noticed that she had confused the meaning of a solid arrow with the meaning of an empty arrow and had also forgotten to denote inheritance properly. While Barbara fixes the inheritance issues, he fills in a few empty arrows and whites out a few of the solid arrow heads. Adam and Cody continue the discussion in the chat box of what approach they should take with their program in regards to a backing end. Soon, their meeting time ends, and all the channels are saved and closed. They each exit the main screen by logging out. They close the executable program, and leave their computers.

3.3Two TAs Teach a Recitation

Any School USA University (ASUSAU) is trying to find a way for more students to attend math class recitations and has decided to try an online recitation environment using CoLab. However, instead of replacing every recitation with CoLab, they select a pre-calculus class of 34 students which is alloted two teacher assistants (TAs). Every Tuesday afternoon at 3:00, the students are supposed to log in and ask questions from their homework if they have any or watch the TAs work and explain some example problems.

3.3.1Setup

The head TA, Suzie, at the beginning of the semester, downloaded and installed CoLab; the first time she opened the program, she was presented with a login screen asking for a username, password and server IP, as well as a button to create a new user. Since she had never used the program before, she first entered the server IP the class would use then clicked the "Create New User" button and entered a username and password of her choosing in text fields. The program checked that her username has not been taken and if the username was valid. Once Suzie was logged in, she was presented with a list of communities on this server. Since Suzie needed a community for her pre-calculus section, she clicked a button to create a new community and enters the community's name ("Math1111Q"). Suzie was logged into the new community automatically and opened the moderator preferences dialog through the main lobby screen. She entered a new password for the community for her students to be required to know the password before joining. Everything was then set up for the online recitations.

3.3.2User Management

A week before the first day of class, she emails the other TA, Bartholomew, as well as the rest of the class with the password. She instructs the students to use their ASUSAU ID for their username. Each student, as well as Bartholomew, sign up with new screen names the same way as Suzie and each choose "Math1111Q" from the drop down menu when prompted for a community. They enter the password they received in the text field that opens, then are able to see the "Math1111Q" lobby screen. Suzie, before the first recitation, changes Bartholomew's status to moderator. She then notices that there are 53 students in the community when there are only 34 students in the class, so she navigates to the moderator preferences menu and looks at the class roster to remove the usernames that do not match from the community user list and changes the password for joining the community.

3.3.3Usage

Once 3pm on Tuesday arrives, Suzie and Bartholomew sign in early and discuss their lesson plan briefly in the chat box on the main lobby window. The students begin to sign on and at five minutes after the hour, Suzie opens a whiteboard channel and names it "Recitation 1/16/2007." The students all double click on "Recitation 1/16/2007" and are able to see the same white board. Suzie announces in the chat box that anyone with a question should type a "?" in the chat box, and anyone who has an answer for a question she or Bartholomew asked should type a "!."

She first wants to quiz the students on transformations of graphs. She clicks on a drawing tool icon on the white board and, using her mouse, draws a simple parabola on a simple

cartesian coordinate system graph and asks for the basic equation that yields that graph. Several students type "!", and Bartholomew calls on jdoe13 to answer in the chat box. He types "y = x^2." Suzie marks his name down for participation then clicks on the erasure tool on the white board to erase the current parabola; she then clicks back on the drawing tool and draws a parabola shifted down three units on the graph and asks her students for the new equation. No one responds, so Bartholomew gives an explanation of graph transformations while Suzie draws a few more examples to illustrate his points. Another student, sdogg42 types a "?" in the chat box, and Bartholomew calls on him. He asks about one of the graphs on "Recitation 1/16/2007" and Suzie does her best to answer.

At the end of the recitation period, Suzie navigates to the moderator preferences menu and selects "Recitation 1/16/2007" to modify its status to "Protected" which allows only she and Bartholomew to edit the channel; several of the students export the whiteboard and chat box and print them for their notes.

4Feature List

4.1Administrative Functions

4.1.1User Accounts

A user needs the ability to create a user account. The account has a unique username and a password, which together are used for authentication.

A user may optionally provide an alias (such as the user's real name), the primary name which is viewable to other users.

4.1.2Communities

Every channel exists within a community. A community member can see every channel in the community, and the data is hidden from non-members.

Once a user authenticates with the server, he may then sign in to any community in which he has membership. An instance of the client application can participate in at most one community at a time.

A user can join a new community by choosing it from a list of all communities on the server and providing the community password.

A new community can be created by any authenticated user.

4.1.3Moderation

Access to a community is controlled by users who are designated as moderators. A moderator can set a password that is required to initially join the community, and may kick/ban joined users.

Initially, the only moderator is the user who created the community. Any moderator of the community can grant the moderation privilege to another member.

4.2Chat Channel Protocol

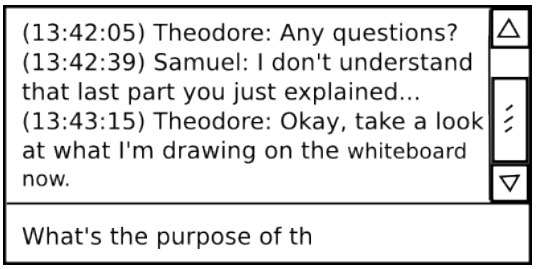
4.2.1Functionality

Each user must be users to post messages to the channel. When a message is posted, it becomes immediately visible to all other users participating in the channel.

Each message includes a timestamp, and all messages are seen chronologically.

4.2.2View

The view looks like a typical chat system. Most of the screen space is used to display the list of messages which have been posted; older messages can be viewed by scrolling upwards in this panel. A single-line text entry mechanism below that display is used to post new messages.



Basic layout of a chat channel, with scrolling content panel and input field

4.2.3Export

The entire history of a chat channel can be exported as a plain text file.

4.3Document Channel Protocol

A document contains textual data. It supports plain text only.

4.3.1Editing

Documents are divided into newline-delimited entities called *paragraphs*. At most one user can modify a paragraph at one time.

When a user begins editing a paragraph, his client acquires a lock on that paragraph to prevent editing conflicts. All other clients in the channel are alerted to this lock and cannot edit the paragraph while that user is still typing. Once the user stops typing and several seconds pass, his client sends the updated paragraph text to the server, and the paragraph lock is freed. All clients see the changes, and anyone can now make their own changes to the text.

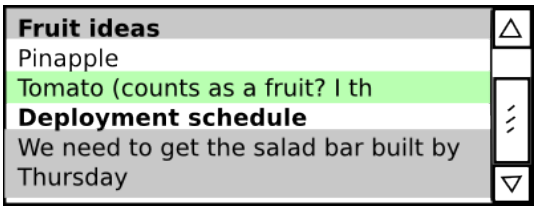
A paragraph can be designated as a *header*. Headers come in several levels to enable hierarchically organized documents. Since designating a paragraph as a header or changing

its level does not create edit conflicts, it can occur regardless of any locks held on the paragraph.

4.3.2View

The entire view for a document channel consists of a scrolling panel which displays the document text.

A lock held by another user is designated by a discolored background on the paragraph. Similarly, another color signifies a lock held by one's own client.



Basic layout of a document channel, with the user editing the line highlighed in green (paragraph 3) while two other people are editing the lines highlighted in gray

4.3.3Revision History

A user can switch his view into *revision history* mode, which provides a navigation mechanism for selecting prior revisions of the document and viewing them in an uneditable panel.

A button allows the user to revert to a given revision. This updates the current contents of the channel to an old state, and puts the view back into editing mode. All locks are removed, and all participants in the channel see the change immediately.

4.3.4Exporting

The entire contents of the document can be exported as an html file. From the revision history view, the contents of any revision can also be exported in the same manner.

4.4Whiteboard Channel Protocol

A whiteboard represents a two-dimensional raster image.

4.4.1Drawing

Users can select a color, and modify an image using several basic drawing tools such. A pen tool allows freeform drawing. A shape tool allows easy addition of shapes such as rectangles, ovals, and straight lines.

Drawing takes place on an infinite canvas; strokes are positioned relatively, and the canvas view scrolls as far as it needs to accommodate the content which has been drawn.

A *clear* button is available to wipe the entire canvas, removing all changes. All previous changes will still be available in the revision history.

4.4.2Revision History

A user can switch into *revision history* mode, similar to that of the document channel protocol. Revision history can be used to view and revert to older versions of the drawing.

4.4.3Exporting

A flattened version of the image can be exported as a png file. From the revision history view, any revision can be exported in the same manner.

5Non-functional Requirements

5.1Usability

The client application has an interface which permits using multiple channels simultaneously. These can be displayed in parallel using multiple windows or panels, or serially via a tabbed interface.

The basic view framework should operate intuitively enough to be usable by novice users without training. This includes connecting to a server, creating a user account, and joining (but not moderating) a community. Individual channel protocols may provide help dialogs if their functionality is sufficiently complex or unintuitive.

5.2Performance

The goal of the application is to provide as close to realtime collaboration as possible. Therefore, it is important to minimize latency to facilitate rapid communication and effective simultaneous use of resources.

5.3Security

The CoLab server must be able to provide service to clients in a limited trust environment.

5.3.1User Authentication

To protect the privacy of groups and user data, it is important to ensure the security of the user login process.

User passwords are required to have a sufficient degree of complexity to ensure their effectiveness for security.

The server discourages password guessing by imposing a rate limit on login attempts from each connection.

Passwords are never sent over the network in clear text, and a one-way hash is applied to any password before it is stored on the server.

5.3.2Access Limitations

It must be impossible for a user to view any data from a community of which he is not a member. A user can access a channel unless he is a member of the group to which it belongs.

5.4Scalability

The aim of CoLab at this time is to solidly support small communities (under 30 users). A single server running on an average modern computer should be able to maintain several (5 at minimum) active communities.

The system must be designed with consideration to expanding its scale in the future. Significant data or network bottlenecks should be documented, and the architecture should be sufficiently modular to allow changes to components which may need restructuring for to address efficiency problems.

5.5Modularity and Extensibility

5.5.1Channel Protocols

These requirements specify three channel protocols: chat, document, and whiteboard. These are to be implemented in a modular manner, such that additional functionality can be added to the system by appending new channel protocols at a later date.

5.5.2Persistent Storage

The server would scale better using a database instead of a file system structure to back its persistent storage, as it would require less data to be held in program memory. However, this is not within the scope of the project at this time. The architecture should have sufficient program-data independence to allow the backing store to be replaced by a database application.

5.6Language

All of the client and server code is written in C#.

5.7Platform Compatibility

As C# is a Microsoft-developed language, and Microsoft's .NET implementation runs only in Windows, Windows XP is the primary supported platform for running CoLab. Cross-compatibility with Unix-based operating systems is a secondary priority, and developers will evaluate at the implementation phase whether other technologies such as Mono reasonably permit platform compatibility beyond the base requirement.

6Class Responsibilities and Collaborations

6.1Channel

Subclasses	WhiteboardChannel, DocumentChannel, ChatChannel
Purpose	Represents a channel in the system, including data, usernames and metadata.
Stereotype	Information Holder, Coordinator
Responsibilities, Collaborators	Knows list of users Send data to receiver - ServerConnection, ClientConnection, ClientData Receive data from sender - ServerConnection, ClientConnection, ClientData Store data about channel - ChannelData

6.2ChannelManager

Purpose	Manage channels
Responsibilities, Collaborators	Creates and knows about channels - Channel Manage file IO from channels

6.3WhiteboardChannel

Purpose	Represents a whiteboard channel
Superclass	Channel
Responsibilities, Collaborators	Knows whiteboard data - WhiteboardChannelData Send data to receiver - ServerConnection, ClientConnection, WhiteboardChannelData Receive data from sender - ServerConnection, ClientConnection, WhiteboardChannelData Store data about channel - WhiteboardChannelData

6.4DocumentChannel

Purpose	Represents a document editor channel
Superclass	Channel
Responsibilities, Collaborators	Knows document data - DocumentChannelData Send data to receiver - ServerConnection, ClientConnection, DocumentChannelData Receive data from sender - ServerConnection, ClientConnection, DocumentChannelData Store data about channel - DocumentChannelData

6.5ChatChannel

Purpose	Represents a chat channel
Superclass	Channel
Responsibilities, Collaborators	Knows chat data - ChatChannelData Send data to server - ServerConnection, ClientConnection, ChatChannelData Receive data from server - ServerConnection, ClientConnection, ChatChannelData Store data about channel - ChatChannelData

6.6ChannelData

Purpose	Represents the data from a single revision in a channel.
Subclasses	WhiteboardChannelData, DocumentChannelData, ChatChannelData
Stereotype	Information Holder
Responsibilities, Collaborators	Knows the data in a channel Knows the user who created this data Knows the time the data was created

6.7WhiteboardChannelData

Purpose	Represents whiteboard channel data
Superclass	ChannelData
Stereotype	Information Holder
Responsibilities, Collaborators	Knows the pen stroke on the whiteboard

6.8Stroke

Purpose	Contains information about a pen stroke on the whiteboard
Stereotype	Information Holder
Responsibilities, Collaborators	Knows the array of points drawn on the screen

6.9DocumentChannelData

Purpose	Represents document channel data
Superclass	ChannelData
Stereotype	Information Holder
Responsibilities, Collaborators	Knows the paragraph of text being sent

6.10DocumentRevision

Purpose	Contains information about one revision to a document
Stereotype	Information Holder
Responsibilities, Collaborators	Holds the contents of a paragraph Knows the User who made the change, the time at which it was made

6.11ChatChannelData

Purpose	Represents chat channel data
Superclass	ChannelData
Stereotype	Information Holder
Responsibilities, Collaborators	Holds an ordered collection of ChatMessages

6.12ChatMessage

Purpose	Represents a message posted by a user to a chat channel
Stereotype	Information Holder
Responsibilities, Collaborators	Knows the message content

6.13Server

Purpose	Coordinates all server-side functions
Stereotype	Coordinator, Information Holder
Responsibilities, Collaborators	Knows all clients that are connected Knows list of all possible users Sends data to client - ClientConnection, ServerConnection Receives data from client - ClientConnection, ServerConnection

6.14ServerConnection

Purpose	Client-side class to allow the client to talk to the server
Stereotype	Interface
Responsibilities, Collaborators	Knows all channels of a client - Channel Collects data from all channels on this client - Channel, ChannelData Sends data to server

6.15IncomingConnectionManager

Purpose	Server-side class to manage incoming connections
Stereotype	Coordinator
Responsibilities, Collaborators	Creates socket connections to clients as they connect to the server - ClientConnection

6.16ClientConnection

Purpose	Server-side class to represent a socket to a client
Stereotype	Interface
Responsibilities, Collaborators	Knows the server-client socket Knows the user Knows the user's current community Knows the channels the user is on Change all user attributes - User Change all community attributes - Community Logs users into communities - UserManager Can join and create channels - ChannelManager

6.17Client

Purpose	Aggregate all client-side data and functions
Stereotype	Coordinator, Interface
Responsibilities, Collaborators	Knows server connection Knows channels the user is on

6.18UserManager

Purpose	Manages users and communities
Stereotype	Controller
Responsibilities, Collaborators	Knows all users and all communities Creates a new user - User Creates a new community - Community

6.19User

Purpose	Represents a user
Stereotype	Information Holder
Responsibilities, Collaborators	Knows the username Knows the actual person's name (display name) Knows the hashed password

6.20Community

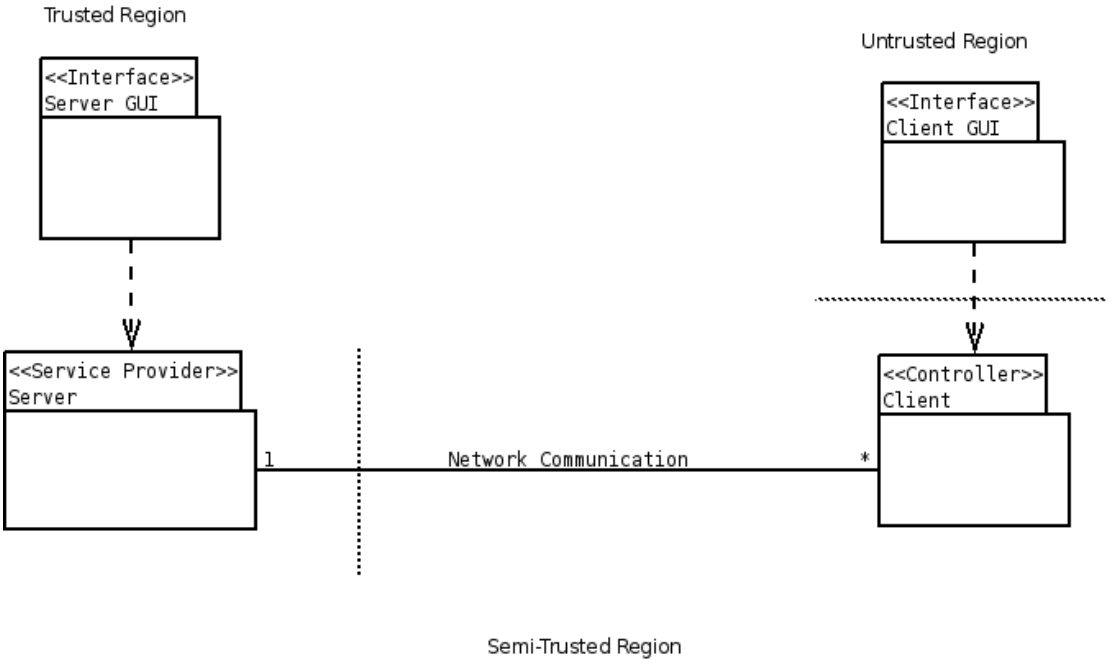
Purpose	Represents a community, which is a collection of users and channels
Stereotype	Structurer
Responsibilities, Collaborators	Knows all valid users Knows whether any particular user is a moderator Knows the write status (protected or normal) Knows the password, if any

7 Architectural Model

The application is based on an event-driven control scheme; the purpose of the application is so closely coupled to user interaction that any other scheme would be cumbersome. Each user edit of a channel will raise an event, and trigger a mechanism to keep all users' documents concurrent.

The issues of channel concurrency and the large set of shared data required suggest a repository-based architecture. The application is aimed at small user groups. A typically sized community, in the proposed academic setting, might be thirty users, and there should rarely be need of more than twenty active communities, disregarding occasional small communities used for personal meetings and one-on-one tutoring. Because the application is based on a repository, and because of the project's scale, the application is built around a two-tier client/server architecture.

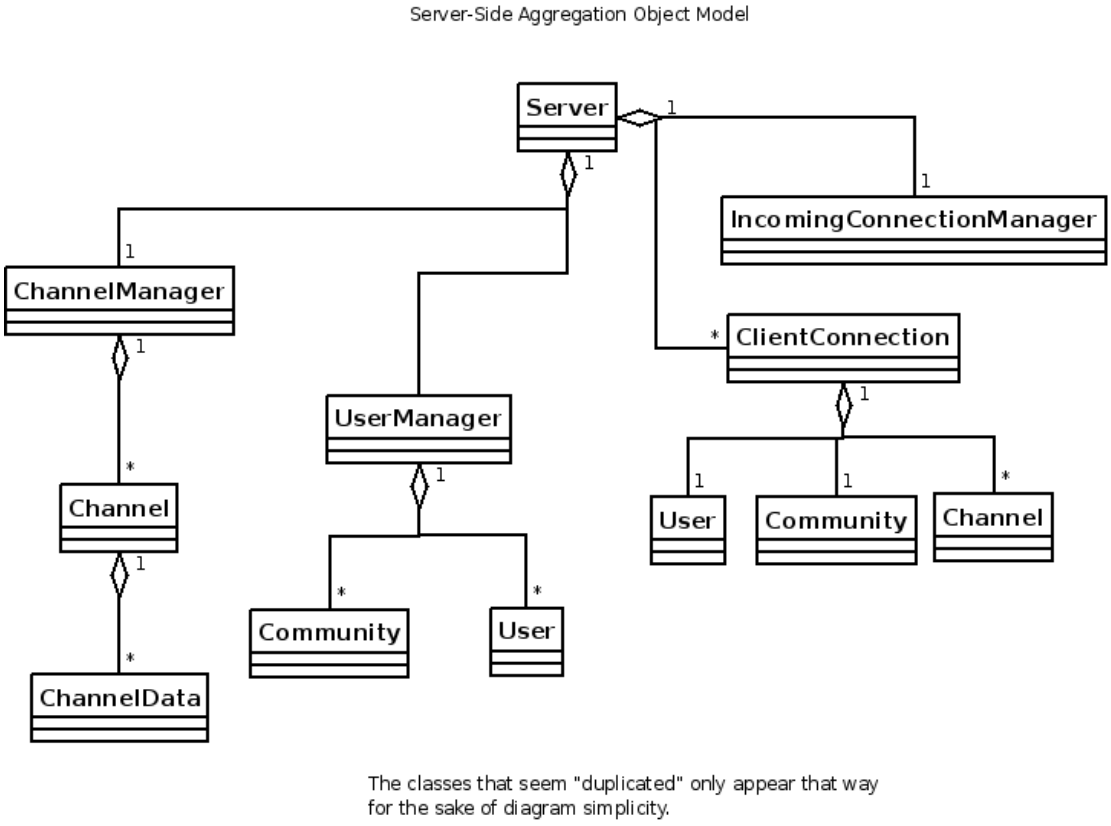
7.1Trust Boundaries



The trust boundaries in a two-tier architecture as this are rarely difficult to distinguish. Usually, the innards of a server are considered trusted, and anything outside of these bounds is considered untrusted; however, the security requirements of this application are not nearly so stringent. For this reason, we define a semi-trusted region where the untrusted region

would usually lie, excluding the client-side user interface. Since most users will be using the application over a local network, this is not an unreasonable distinction.

7.2Server



CoLab uses a dedicated server application that handles all of the networking necessary for document concurrency, without any direct peer-to-peer communication. The server is also responsible for all persistent data storage, including metadata on communities and users, as well as any logs and files produced by the application. This central role in the application lends itself toward two general server functions: user/community and content management. The one all-pervasive facilitator for these functions is the network.

The function of content management is taken by the ChannelManager class, and user and community management is handled through the UserManager class. Network abstraction is provided by IncomingConnectionManager and ClientConnection. IncomingConnectionManager listens for connection requests, and wraps newly-opened connections in a ClientConnection object. The ClientConnection object acts as the

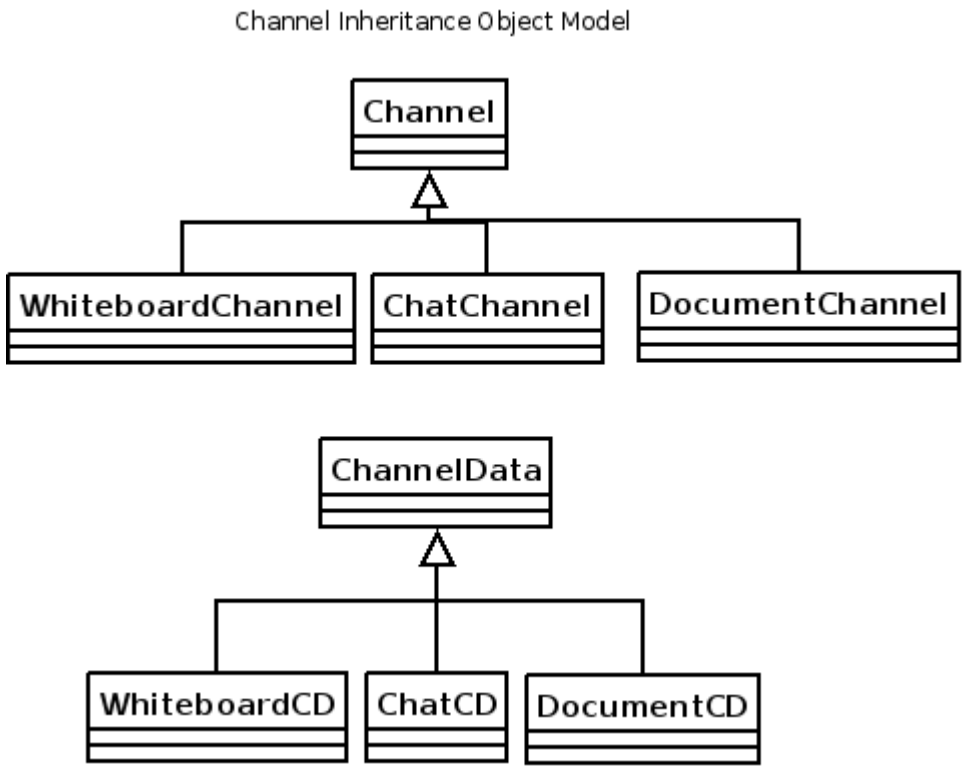
client/server interface. It interprets and carries out client commands through the use of UserManager, ChannelManager, User, Community, and Channel.

7.2.1User Management

The User class represents a user, complete with user name and hashed password. The Community class is a structural element, consisting of User members, at least one designated User moderator, and intra-community preferences, including a community password.

The UserManager class holds all Community and User objects registered with the server. It has two functions, both mutually obvious and mutually important. It is responsible for the persistent storage of the User/Community hierarchy. It also managers creation and authentication of users and communities.

7.2.2Content Management



Data is represented congruently on both the client- and server-side. The Channel abstraction - objects built of individual revisions, acting as communication media - is a flexible solution

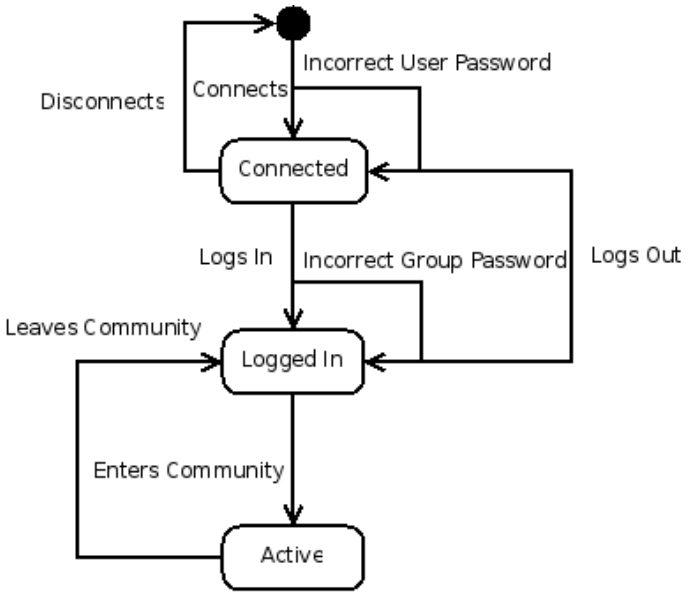
to similarity of storage between the client and server, easing the difficulty of intercommunication. Channel, and its constituent, ChannelData, are each subclassed to allow different channel protocols, corresponding to the typed of tools available to an end users.

A benefit gained from using the Channel abstraction is that Channel objects - potentially text documents, images, et cetera - will have simple, uniform methods to serialize and save the data they represent. On the server side, whenever a ChannelData object is added to a Channel, an event is fired to save the change to a storage backend. On the client side, the same event will trigger the revision being applied to the visual client model.

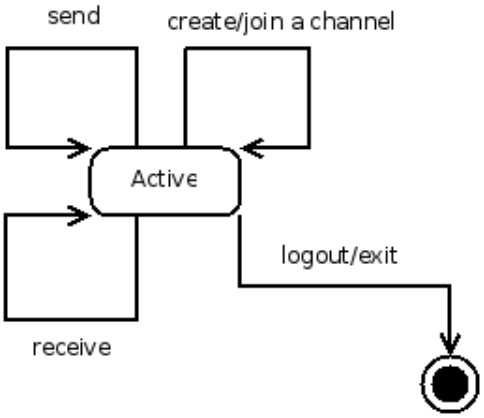
ChannelManager manages both Channel objects as they are used in a session, and Channel objects as they are viewed as files on the server side. Although the ChannelManager will listen for Channel events, it will not involve itself in file I/O. Its responsibilities consist of merely providing create/join functionality for channels in a community, and managing the file structure to which those channels will commit their own changes.

7.2.3Networking

The server accepts and maintains a socket connection for each client, wrapped in a ClientConnection object. These objects have three important states, and an initial disconnected state.



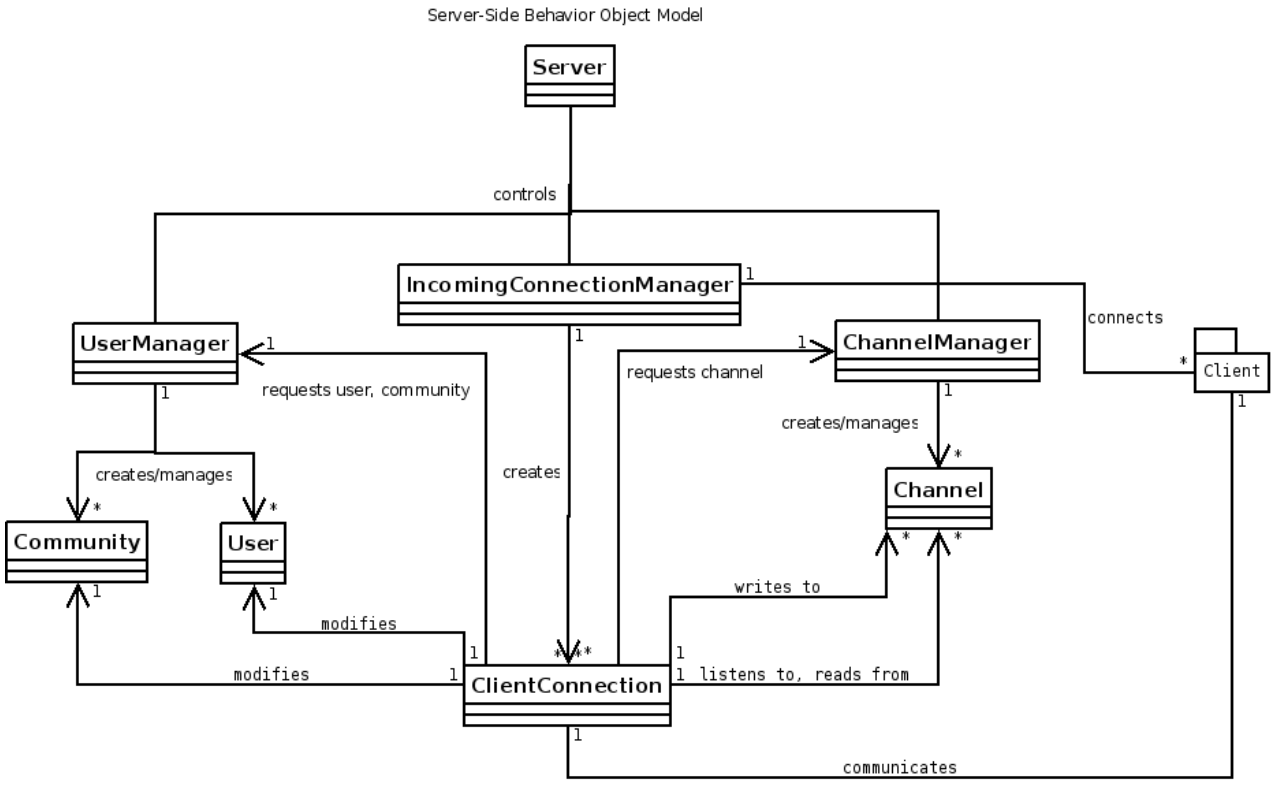
The three significant states are represented by the creation of a ClientConnection object, a reference to a valid User object, and a reference to a valid Community. The Active state can be further subdivided into additional transitions, as shown on the diagram below, where the final state is either the Logged In, Connected, or Disconnected state shown in the diagram above.



The ClientConnection objects completely encapsulate the sending and receiving of network data from data storage by only communicating with User, Community, and most importantly, Channel, and by interacting only with UserManager and ChannelManager to request those objects.

7.2.4Behavior

Together, these functions comprise the server solution.



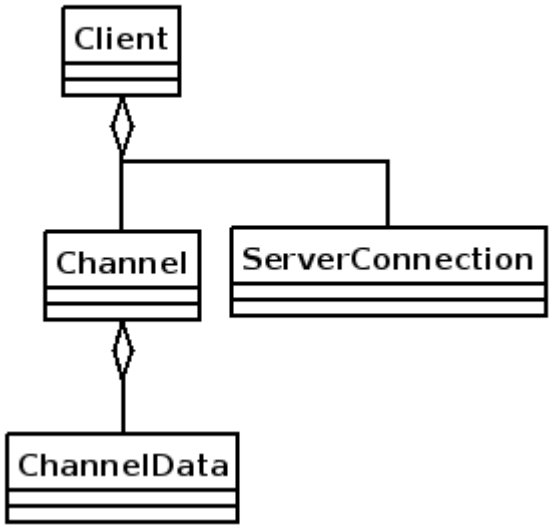
IncomingConnectionManager accepts new connections and wraps them in ClientConnection objects. ClientConnection objects request authentication from UserManager, request channel references from ChannelManager, modify their own User and Community objects, and write to and receive data from Channel objects. ClientConnection objects respond to data they receive from client side and send the client changes in a Channel. UserManager supplies User objects when given the proper password, creates and supplies new Users when given a unique user name, and handles persistent storage of User and Community objects. UserManager also supplies Community objects when given a valid User, or password. ChannelManager supplies a Channel when given a valid channel name and Community, or creates a channel when given a valid and unique channel name, User, and Community. ChannelManager also listens to changes in Channel objects, and responds by logging them in their proper files.

7.3Client

CoLab's simpler client-side solution has two basic responsibilities: keeping channel concurrency over the network, and informing the user when a channel changes.

The user will be informed of channel alterations in a similar fashion with which the server keeps up with them, except instead of listeners that update the server back-end, the listeners will fire events in the Client class for display by the GUI.

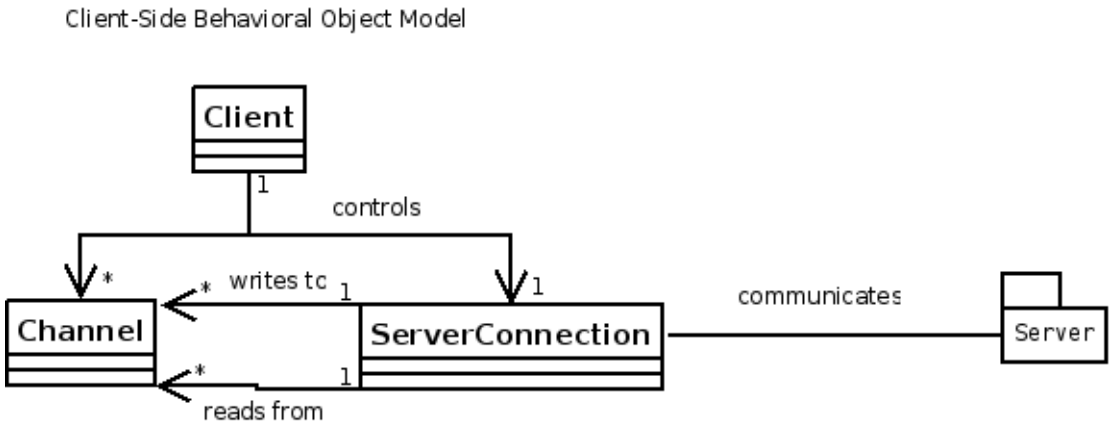
Client-Side Aggregation Object Model



7.3.1 Networking

All client-side network activity is accomplished through the ServerConnection object. A client has exactly one ServerConnection which is responsible for sending updates to the server whenever a client-side Channel object is revised, and for updating a client-side Channel whenever the server notifies that its Channel counterpart has changed. The networking function of the client is minimal, which makes sense, given the two-tier architecture.

7.3.2 Behavior



8Validation

CoLab will be validated through two usage scenarios of the application using the think-aloud method.

8.1TA Review Session

Participants
10 students, 1-2 TAs (moderators)

Goal
Use CoLab to hold a review session online that is comparable to one held in a physical classroom.

Criteria for Validity
Students must be able to ask a question at any time.

Students should be able to read answers to previously asked questions from TAs.

TAs should be able to draw diagrams as needed to illustrate concepts.

Software should be fast and responsive, providing updates in a real-time manner that is similar to the experience of a classroom environment.

8.2Mutual Peer Group Meeting

Participants
4 students (all moderators)

Goal
Collaboratively write a review of the software user experience

Criteria for validity
Students must be able to communicate easily with one another.

Students should be able to simultaneously discuss what they are doing while doing it.

Editing conflicts should be minimal.

Software should be fast and responsive, and updates should be as close to real-time as possible.

9Delivery Plan

9.1Delivery 1: February 21

- Client can enter user name and password and login
- Server can authenticate incoming clients
- Chat channel protocol is functional
- Client automatically opens a single chat channel upon login
- GUI available for client login, lobby chat

9.2Delivery 2: March 21

- Document channel protocol is functional
- User can create and use a document channel
- User can create user accounts and communities
- User can create channels, join multiple channels concurrently
- Moderators can modify community preferences (adding passwords, changing blacklist, etc.)

9.3Delivery 3: April 21

- Whiteboard channel protocol is functional
- User can create and use a whiteboard channel
- All functionality implemented