Project 2: Distributed Shared White Board

Dr. Tawfiq Islam

The University of Melbourne, Australia

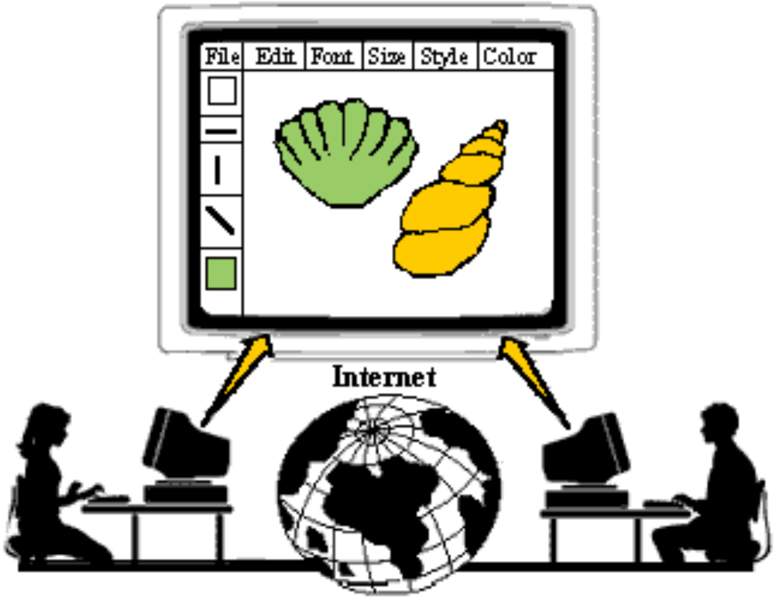
Other contributors: All Tutors

Distributed Shared White Board

- In these slides, we are offering mainly the **guidelines** for satisfactory work, but **be innovative and creative**, which will be valued a lot.
- Team/Members Size: 2 – Group-based (like Assignment 1).
- General help: Ask **your** tutor during/after tutorial session. Also use "Discussion Board" in LMS.
- Marks Allocated: 20

- Note: We expect all the groups to just finish and submit only the features noted in this specification.
- To help you in planning, we proposed:
  - Basic Features (first complete a system with these features as they are easier)
  - Advanced Features

Shared White Board – Distributed Users



- Shared whiteboards allow multiple users to draw simultaneously on a canvas. There are multiple examples found on the Internet that support a range of features such as freehand drawing with the mouse, drawing lines and shapes such as circles and squares that can be moved and resized, and inserting text.

Main Challenges

- Dealing with concurrency
  - Regardless of the technology you use, you will have to ensure that access to shared resources is properly handled and that simultaneous actions lead to a reasonable state.
- Structuring your application and handling the system state
  - For example, you can have multiple servers that communicate with each other or a single central one that manages all the system state.
- Dealing with networked communication
  - You have to decide when/what messages are sent across the network.
  - You may have to design an exchange protocol that establishes which messages are sent in which situation and the replies that they should generate.
  - If you use RMI, then you have to design your remote interface(s) and servants
- Implementing the GUI.
  - The functionality can resemble tools like MS Paint.
  - You can use any tool/API/library you want.
  - e.g.: Java2D drawing package (http://docs.oracle.com/javase/tutorial/2d/index.html)

Distributed White Board

- Develop a white board that can be shared between multiple users over the network.
- The system must be implemented in Java but you can choose the technology (e.g., it can be even Sockets) you want to use to build your distributed application:
  - Sockets
    - TCP or UDP?
    - Message format and Exchange protocol?
      - can be XML-based or your own format
      - Client can broadcast a message with updates to all other clients, other clients reply acknowledging the message.
  - Java RMI?
    - Remote Objects/Remote Interface?
  - File or Database for Storage
  - Please Choose any technology of your choice
    - Make sure that your project can achieve the goal when you are choosing a “new” technology (otherwise, stick to what you already know).

Requirements – Basic Features

- Whiteboard – Multiuser system
  - Multiple users can draw on a shared interactive canvas, created by a manager.
  - Your system will support a single whiteboard that is shared between all of the clients.
  - Consistent user pool view for all clients.
  - Management of all corresponding exceptions.
  - Key Elements with GUI
    - Shapes: at least your white board should support for line, circle, triangle, and rectangle.
    - Free-hand drawing.
    - Text inputting– allow user to type text anywhere inside the white board.
    - User should be able to choose their favourite colour to draw the above features. At least 16 colours should be available.

Advanced Features

1. Chat Window (text based): To allow users to communicate with each other by typing a text.

2. A “File” menu with *new*, *open*, *save*, *saveAs* and *close* should be provided (only the manager can control this).

3. Allow the manager to kick out a certain peer/user.

4. All corresponding exceptions should be managed.

Guidelines on Usage/Operation

- Users must provide a username when joining the whiteboard. There should be a way of uniquely identifying users, either by enforcing unique usernames or automatically generating a unique identifier and associating it with each username.
- All the users should see the same image of the whiteboard and should have the privilege of doing all the drawing operations.
- When displaying a whiteboard, the client user interface should show the usernames of other users who are currently editing the same whiteboard.
- Clients may connect and disconnect at any time. When a new client joins the system, the client should obtain the current state of the whiteboard so that the same objects are always displayed to every active client.
- Only the manager of the whiteboard should be allowed to create a new whiteboard, open a previously saved one, save the current one, and close the application.
- Users should be able to work on a drawing together in real time, without appreciable delays between making and observing edits.

Proposed Startup/Operational Model

- The first user creates a whiteboard and becomes the whiteboard's manager
  - java CreateWhiteBoard <serverIPAddress> <serverPort> username
- Other users can ask to join the whiteboard application any time by inputting server's IP address and port number
  - java JoinWhiteBoard <serverIPAddress> <serverPort> username
- A notification will be delivered to the manager if any peer wants to join. The peer can join in only after the manager approves
  - A dialog showing "someone wants to share your whiteboard".
- An online peer list should be maintained and displayed
- All the peers will see the identical image of the whiteboard, as well as have the privilege of doing all the operations.
- Online peers can choose to leave whenever they want. The manager can kick someone out at any time.
- When the manager quits, the application will be terminated. All the peers will get a message notifying them.

Guidelines/Suggestions for Implementation

- These phases are suggestions for timely progression, you are most welcome to follow your own approach.
- Phase 1 (whiteboard) – (aim to finish within 2 weeks of announcement)
  - **As a starting point:** Single-user standalone whiteboard (OR) You are most welcome to implement a single user and single server.
  - Task A: Implement a client that allows a user to draw all the expected elements.
  - Task B: Implement a server so that client and server are able to communicate entities created in Task A

Guidelines/Suggestions for Implementation

- Phase 2 (user management skeleton)
  - Allow the manager to create a whiteboard
  - Allow other peers to connect and join in by getting approval from the manager
  - Allow the manager to choose whether a peer can join in
    - join in means the peer name will appear in the user list
  - Allow the joined peer to choose quit
  - Allow the manager to close the application, and all peers get notified
  - Allow the manager to kick out a certain peer/user

Guidelines – Suggestions (You are most welcome follow your own approach)

- Phase 3 (Final)
  - Integrate the whiteboard with the user management skeleton (phases 1 and 2)
  - Design issues:
    - What communication mechanism will be used?
      - Socket, RMI, or any other frameworks of your choice.
    - How to propagate the modification from one peer to other peers?
      - You may need an event-based mechanism
    - How many threads do we need per peer?
      - At least one for drawing, one for messaging

Deliverables, Marks and Deadlines

- Report (4 marks)
- Code and Demo of Network-based Distributed Users, Shared Whiteboard:
  - Basic System (12 marks)
  - Advanced Features (4 marks)
- **Deadline:**
  - **October 21, 2022 (Friday) at 5:00pm**
- Note: You are NOT allowed to use ANY code taken from any existing shared whiteboard implementation. Full design, code, report has to be your OWN work. Copying code/content from other sources carries penalty and disciplinary action as per the University rules.

Final Submission

- Report
  - You should write a report that includes the system architecture, communication protocols and message formats, design diagrams (class and interaction), implementation details.
  - Don’t document anything you haven’t implemented in the report. This is misconduct and will result in severe penalties.
- You need to submit the following via LMS:
  - Your report in PDF format *only*.
  - The *executable jar* files used to run your system’s clients/server(s)
  - Your source files in a .ZIP or .TAR archive *only*.

Milestone: Demonstration

- Demonstrations
  - You will showcase your system and discuss your design choices during the demos.
  - Date and venue will be announced closer to the submission date.

Penalties for assignments

- Assignments submitted late will be penalized in the following way:
  - 1 day late: -1 mark
  - 2 days late: -2 marks
  - 3 days late: -3 marks
  - 4 days late: -4 marks
  - etc. (that is, -1 mark for each day delay).
- No Demo results in significant mark deduction
- Re-scheduling of demo is not permitted