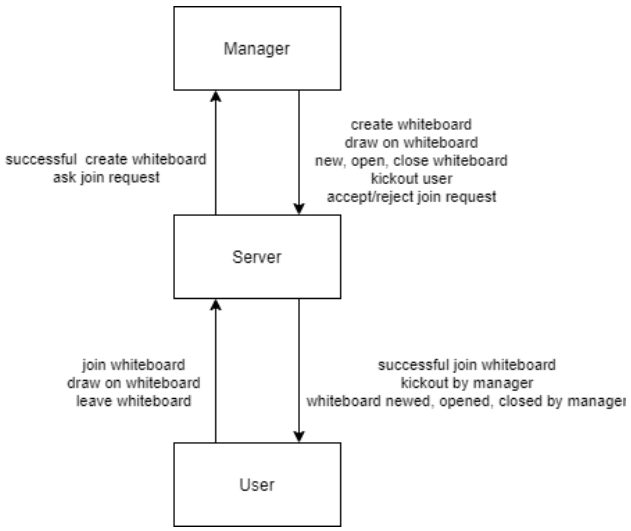
COMP90015 Projet2 2020S1 Report

Xulin Yang 904904

System architecture



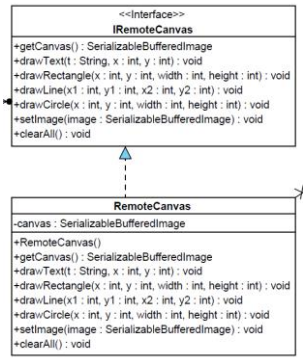
This the system architecture diagram. The system has a central server. The whiteboard and users' information are stored on the server. Different users like manager and user can use the whiteboard with different privileges. The server has RMI objects to be used by clients and also having TCP requests from clients' operations.

The reason that I choose the central server architecture is that I can benefit from the following advantages:

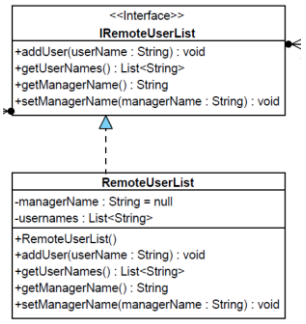
1. Easy to physically secure. It is easy to secure and service the server and client nodes by virtue of their location
2. Smooth and elegant personal experience – A client has a dedicated system which he uses (for example, a personal computer) and the company has a similar system which can be modified to suit custom needs
3. Dedicated resources (memory, CPU cores, etc)
4. More cost efficient for small systems upto a certain limit – As the central systems take less funds to set up, they have an edge when small systems must be built
5. Quick updates are possible – Only one machine to update.
6. Easy detachment of a node from the system. Just remove the connection of the client node from the server and voila! Node detached.

Communication protocols & message formats

The communication protocol is that I have used RMI to handle remote object. In the system I have one remote object for drawing shapes on the canvas, and one remote object



This the class for drawing on the whiteboard uses RMI for the system. The client can get the remote object by using the lookup remote object by the key from the rmiregistry. The client can draw on whiteboard by calling the remote methods defined in the IRemoteCanvas.



This the class for holding the client's names information uses RMI for the system. The client can get the remote object by using the lookup remote object by the key from the rmiregistry. The client can get client's name who is using the whiteboard by calling the remote methods defined in the IRemoteCanvas.

The reason that I use RMI for these component objects is that RMI has the following advantages.

1. Handles threads and Sockets for underlying communication.
2. Server-side implementation can be changed without the knowledge of the client side.
3. Easy to extend an RMI solution, you can extend or add new classes, just like in a non-distributed application.

As we might extend the functionality of the whiteboard canvas such as add more shapes to draw, using RMI for the canvas object and userList object for the user information to be stored can be beneficial. If I use TCP as the communication method, the design of the system will be complicated and I need to ensure the consensus of the data which will make it very difficult.

Although we have the advantages for using the RMI as listed following:

1. Overhead of marshaling and unmarshaling
2. Overhead of object serialization

But this will not affect us too much. Because the object is simple, and the serialization won't be time consuming.

To sum it up, I choose RMI as the communication protocol for the features of these two objects.

The system also uses TCP as the communication between clients and server.

The server can process request:

1. Create Whiteboard
2. Join Whiteboard
3. Ask Join Whiteboard result
4. User leave Whiteboard
5. Manager close Whiteboard
6. Kick out user
7. Manager new a whiteboard
8. Manager open a whiteboard

The manager can process request:

1. Ask Join Whiteboard

The client can process request:

1. Kick out user
2. Manager closed the whiteboard
3. Manager new a whiteboard
4. Manager open a whiteboard

The message is JSON formatted. With {"request": task above, } and other key, value pair as data if needed.

The reason that I use TCP as the communication method for these operations is because I can benefit from the following advantages:

1. The connections between server and client can be established at any time and disconnected at any time.
2. Server and client applications can evolve independently (i.e., Client APIs can evolve independently of the server APIs).

As these requests are not associated with a specific object, using TCP as communication protocol is more suitable in this case compared to using RMI. Furthermore, these requests do not need to be extended for more features which makes RMI not useful.

Although using TCP can has the following disadvantage:

1. Delay in speed of communication.

But these messages will not be transmitted for too many times in the system. So, it is ok to transmit these messages using the TCP.

To sum it up, I choose TCP as the communication protocol for the features listed above.

Design diagrams

Class diagram

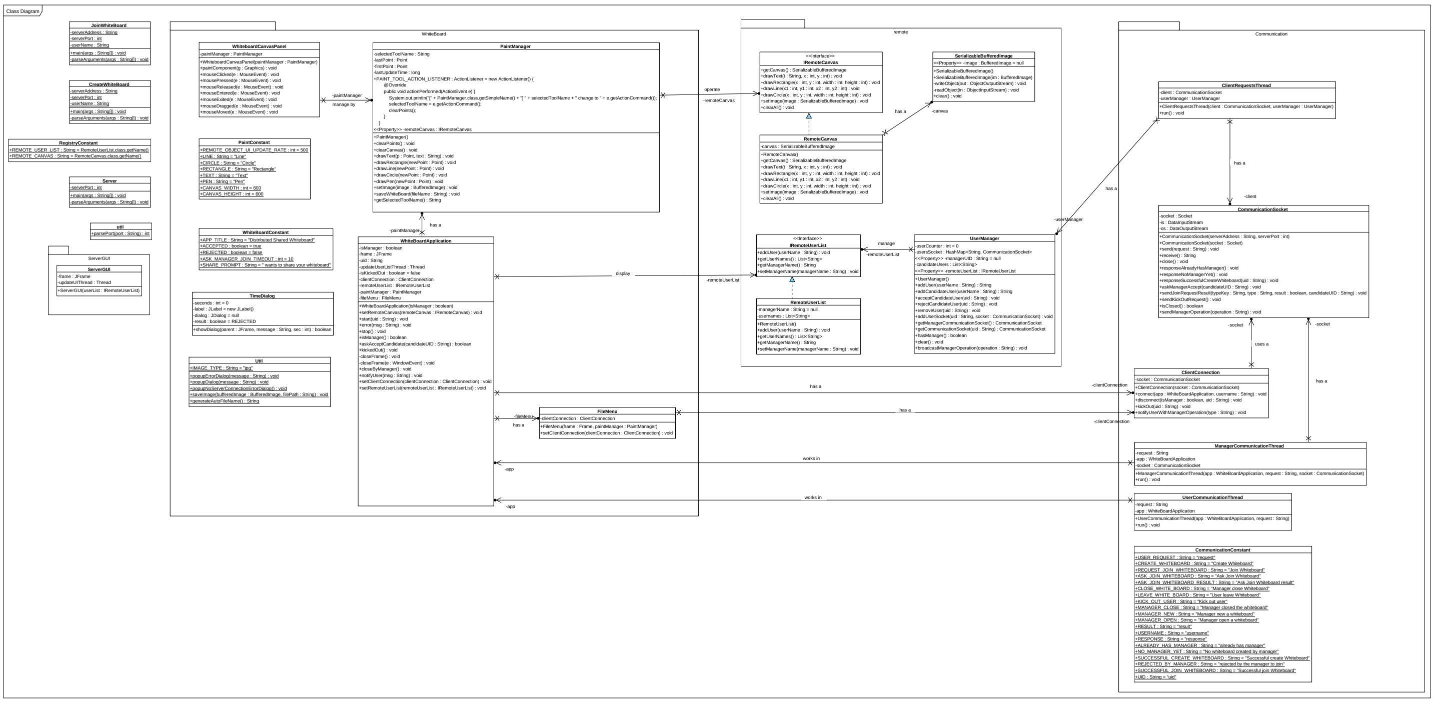
As you can see from the class diagram, in the root package, the JoinWhiteBoard, CreateWhiteBoard and Server are three classes for start the system for user, manager and the central server. The RegistryConstant contains the constant for the JoinWhiteBoard and CreateWhiteBoard to lookup the RMI remote canvas and userlists from the server. The util class contains some helper functions.

In the ServerGUI package, there is a gui class for the server which is ServerGUI to be invoked when the server starts.

In the remote package, I have two major class uses RMI to be used during the client server communication. RemoteCanvas implement the IRemoteCanvas class with a SerializableBufferedImage class because the BufferedImage class in JAVA is not serializable thus it cannot be passed in the RMI communication. So, I serialize it to make the BufferedImage can be used as whiteboard canvas. The RemoteCanvas is a shared object for all clients of shared whiteboard to draw shapes on. The RemoteUserList implement IRemoteUserList class to be used to store all client's usernames including manager, users and users waiting for manager's acceptance/rejection. And there is a UserManager class for the server to perform the management actions on IRemoteUserList.

In the Communication package, I have ClientRequestsThread works at server to process client's requests and send responses by using the methods provided by CommunicationSocket class. ClientConnection works at the manager and the user side which can be used to establish the connection to the server by sending requests. ManagerCommunicationThread works at manager side to receive request from server and process and send responses to the server. UserCommunicationThread works at user side to receive request from server and process and send responses to the server. CommunicationConstant contains the constant use during the client and server communication.

In the WhiteBoard package, it contains the class used for the GUI and logic of the frontend application. WhiteBoardApplication is the GUI class used by the manager and the user. It initializes the UI components. If the client is manager, then the application has the functionality provided by the FileMenu class. The WhiteBoardApplication uses WhiteboardCanvasPanel to let client to draw shapes on it. The logic of drawing is handled by the PaintManager. TimeDialog is a class with Dialog but auto select cancel if the client makes no choices within the timeout period. The PaintConstant class contains the constant for painting such as tool names. The WhiteBoardConstant class contains constants used by the application. The Util class has some helper functions for the application.


Class Diagram
JoinWhiteBoard
-serverAddress : String
-serverPort : int
-userName : String
+main(args : String[]) : void
-parseArguments(args : String[]) : void
CreateWhiteBoard
-serverAddress : String
-serverPort : int
-userName : String
+main(args : String[]) : void
-parseArguments(args : String[]) : void
RegistryConstant
+REMOTE_USER_LIST : String = RemoteUserList.class.getName()
+REMOTE_CANVAS : String = RemoteCanvas.class.getName()
Server
-serverPort : int
+main(args : String[]) : void
-parseArguments(args : String[]) : void
util
+parsePort(port : String) : int
ServerGUI
ServerGUI
-frame : JFrame
-updateUIThread : Thread
+ServerGUI(userList : IRemoteUserList)
WhiteBoard
WhiteboardCanvasPanel
-paintManager : PaintManager
+WhiteboardCanvasPanel(paintManager : PaintManager)
+paintComponent(g : Graphics) : void
+mouseClicked(e : MouseEvent) : void
+mousePressed(e : MouseEvent) : void
+mouseReleased(e : MouseEvent) : void
+mouseEntered(e : MouseEvent) : void
+mouseExited(e : MouseEvent) : void
+mouseDragged(e : MouseEvent) : void
+mouseMoved(e : MouseEvent) : void
-paintManager
manage by
PaintConstant
+REMOTE_OBJECT_UI_UPDATE_RATE : int = 500
+LINE : String = "Line"
+CIRCLE : String = "Circle"
+RECTANGLE : String = "Rectangle"
+TEXT : String = "Text"
+PEN : String = "Pen"
+CANVAS_WIDTH : int = 600
+CANVAS_HEIGHT : int = 600
WhiteBoardConstant
+APP_TITLE : String = "Distributed Shared Whiteboard"
+ACCEPTED : boolean = true
+REJECTED : boolean = false
+ASK_MANAGER_JOIN_TIMEOUT : int = 10
+SHARE_PROMPT : String = " wants to share your whiteboard"
TimeDialog
-seconds : int = 0
-label : JLabel = new JLabel()
-dialog : JDialog = null
-result : boolean = REJECTED
+showDialog(parent : JFrame, message : String, sec : int) : boolean
Util
+IMAGE_TYPE : String = "jpg"
+popupErrorDialog(message : String) : void
+popupDialog(message : String) : void
+popupNoServerConnectionErrorDialog() : void
+saveImage(bufferedImage : BufferedImage, filePath : String) : void
+generateAutoFileName() : String
PaintManager
-selectedToolName : String
-lastPoint : Point
-firstPoint : Point
-lastUpdateTime : long
+PAINT_TOOL_ACTION_LISTENER : ActionListener = new ActionListener() {
@Override
public void actionPerformed(ActionEvent e) {
System.out.println("[" + PaintManager.class.getSimpleName() + "] " + selectedToolName + " change to " + e.getActionCommand());
selectedToolName = e.getActionCommand();
clearPoints();
}
}
<<Property>> -remoteCanvas : IRemoteCanvas
+PaintManager()
+clearPoints() : void
+clearCanvas() : void
+drawText(p : Point, text : String) : void
+drawRectangle(newPoint : Point) : void
+drawLine(newPoint : Point) : void
+drawCircle(newPoint : Point) : void
+drawPen(newPoint : Point) : void
+setImage(image : BufferedImage) : void
+saveWhiteBoard(fileName : String) : void
+getSelectedToolName() : String
has a
-paintManager
WhiteBoardApplication
-isManager : boolean
-frame : JFrame
-uid : String
-updateUserListThread : Thread
-isKickedOut : boolean = false
-clientConnection : ClientConnection
-remoteUserList : IRemoteUserList
-paintManager : PaintManager
-fileMenu : FileMenu
+WhiteBoardApplication(isManager : boolean)
+setRemoteCanvas(remoteCanvas : IRemoteCanvas) : void
+start(uid : String) : void
+error(msg : String) : void
+stop() : void
+isManager() : boolean
+askAcceptCandidate(candidateUID : String) : boolean
+kickedOut() : void
-closeFrame() : void
-closeFrame(e : WindowEvent) : void
+closeByManager() : void
+notifyUser(msg : String) : void
+setClientConnection(clientConnection : ClientConnection) : void
+setRemoteUserList(remoteUserList : IRemoteUserList) : void
display
-fileMenu
has a
FileMenu
-clientConnection : ClientConnection
+FileMenu(frame : Frame, paintManager : PaintManager)
+setClientConnection(clientConnection : ClientConnection) : void
-app
-app
operate
-remoteCanvas
-remoteUserList
remote
<<Interface>>
IRemoteCanvas
+getCanvas() : SerializableBufferedImage
+drawText(t : String, x : int, y : int) : void
+drawRectangle(x : int, y : int, width : int, height : int) : void
+drawLine(x1 : int, y1 : int, x2 : int, y2 : int) : void
+drawCircle(x : int, y : int, width : int, height : int) : void
+setImage(image : SerializableBufferedImage) : void
+clearAll() : void
SerializableBufferedImage
<<Property>> -image : BufferedImage = null
+SerializableBufferedImage()
+SerializableBufferedImage(im : BufferedImage)
-writeObject(out : ObjectOutputStream) : void
-readObject(in : ObjectInputStream) : void
+clear() : void
has a
-canvas
RemoteCanvas
-canvas : SerializableBufferedImage
+RemoteCanvas()
+getCanvas() : SerializableBufferedImage
+drawText(t : String, x : int, y : int) : void
+drawRectangle(x : int, y : int, width : int, height : int) : void
+drawLine(x1 : int, y1 : int, x2 : int, y2 : int) : void
+drawCircle(x : int, y : int, width : int, height : int) : void
+setImage(image : SerializableBufferedImage) : void
+clearAll() : void
<<Interface>>
IRemoteUserList
+addUser(userName : String) : void
+getUserNames() : List<String>
+getManagerName() : String
+setManagerName(managerName : String) : void
manage
-remoteUserList
-userManager
UserManager
-userCounter : int = 0
-usersSocket : HashMap<String, CommunicationSocket>
<<Property>> -managerUID : String = null
-candidateUsers : List<String>
<<Property>> -remoteUserList : IRemoteUserList
+UserManager()
+addUser(userName : String) : String
+addCandidateUser(userName : String) : String
+acceptCandidateUser(uid : String) : void
+rejectCandidateUser(uid : String) : void
+removeUser(uid : String) : void
+addUserSocket(uid : String, socket : CommunicationSocket) : void
+getManagerCommunicationSocket() : CommunicationSocket
+getCommunicationSocket(uid : String) : CommunicationSocket
+hasManager() : boolean
+clear() : void
+broadcastManagerOperation(operation : String) : void
RemoteUserList
-managerName : String = null
-usernames : List<String>
+RemoteUserList()
+addUser(userName : String) : void
+getUserNames() : List<String>
+getManagerName() : String
+setManagerName(managerName : String) : void
has a
has a
works in
works in
Communication
ClientRequestsThread
-client : CommunicationSocket
-userManager : UserManager
+ClientRequestsThread(client : CommunicationSocket, userManager : UserManager)
+run() : void
has a
has a
-client
CommunicationSocket
-socket : Socket
-is : DataInputStream
-os : DataOutputStream
+CommunicationSocket(serverAddress : String, serverPort : int)
+CommunicationSocket(socket : Socket)
+send(request : String) : void
+receive() : String
+close() : void
+responseAlreadyHasManager() : void
+responseNoManagerYet() : void
+responseSuccessfulCreateWhiteboard(uid : String) : void
+askManagerAccept(candidateUID : String) : void
+sendJoinRequestResult(typeKey : String, type : String, result : boolean, candidateUID : String) : void
+sendKickOutRequest() : void
+isClosed() : boolean
+sendManagerOperation(operation : String) : void
-socket
-socket
uses a
has a
-clientConnection
-clientConnection
ClientConnection
-socket : CommunicationSocket
+ClientConnection(socket : CommunicationSocket)
+connect(app : WhiteBoardApplication, username : String) : void
+disconnect(isManager : boolean, uid : String) : void
+kickOut(uid : String) : void
+notifyUserWithManagerOperation(type : String) : void
ManagerCommunicationThread
-request : String
-app : WhiteBoardApplication
-socket : CommunicationSocket
+ManagerCommunicationThread(app : WhiteBoardApplication, request : String, socket : CommunicationSocket)
+run() : void
UserCommunicationThread
-request : String
-app : WhiteBoardApplication
+UserCommunicationThread(app : WhiteBoardApplication, request : String)
+run() : void
CommunicationConstant
+USER_REQUEST : String = "request"
+CREATE_WHITEBOARD : String = "Create Whiteboard"
+REQUEST_JOIN_WHITEBOARD : String = "Join Whiteboard"
+ASK_JOIN_WHITEBOARD : String = "Ask Join Whiteboard"
+ASK_JOIN_WHITEBOARD_RESULT : String = "Ask Join Whiteboard result"
+CLOSE_WHITE_BOARD : String = "Manager close Whiteboard"
+LEAVE_WHITE_BOARD : String = "User leave Whiteboard"
+KICK_OUT_USER : String = "Kick out user"
+MANAGER_CLOSE : String = "Manager closed the whiteboard"
+MANAGER_NEW : String = "Manager new a whiteboard"
+MANAGER_OPEN : String = "Manager open a whiteboard"
+RESULT : String = "result"
+USERNAME : String = "username"
+RESPONSE : String = "response"
+ALREADY_HAS_MANAGER : String = "already has manager"
+NO_MANAGER_YET : String = "No whiteboard created by manager"
+SUCCESSFUL_CREATE_WHITEBOARD : String = "Successful create Whiteboard"
+REJECTED_BY_MANAGER : String = "rejected by the manager to join"
+SUCCESSFUL_JOIN_WHITEBOARD : String = "Successful join Whiteboard"
+UID : String = "uid"

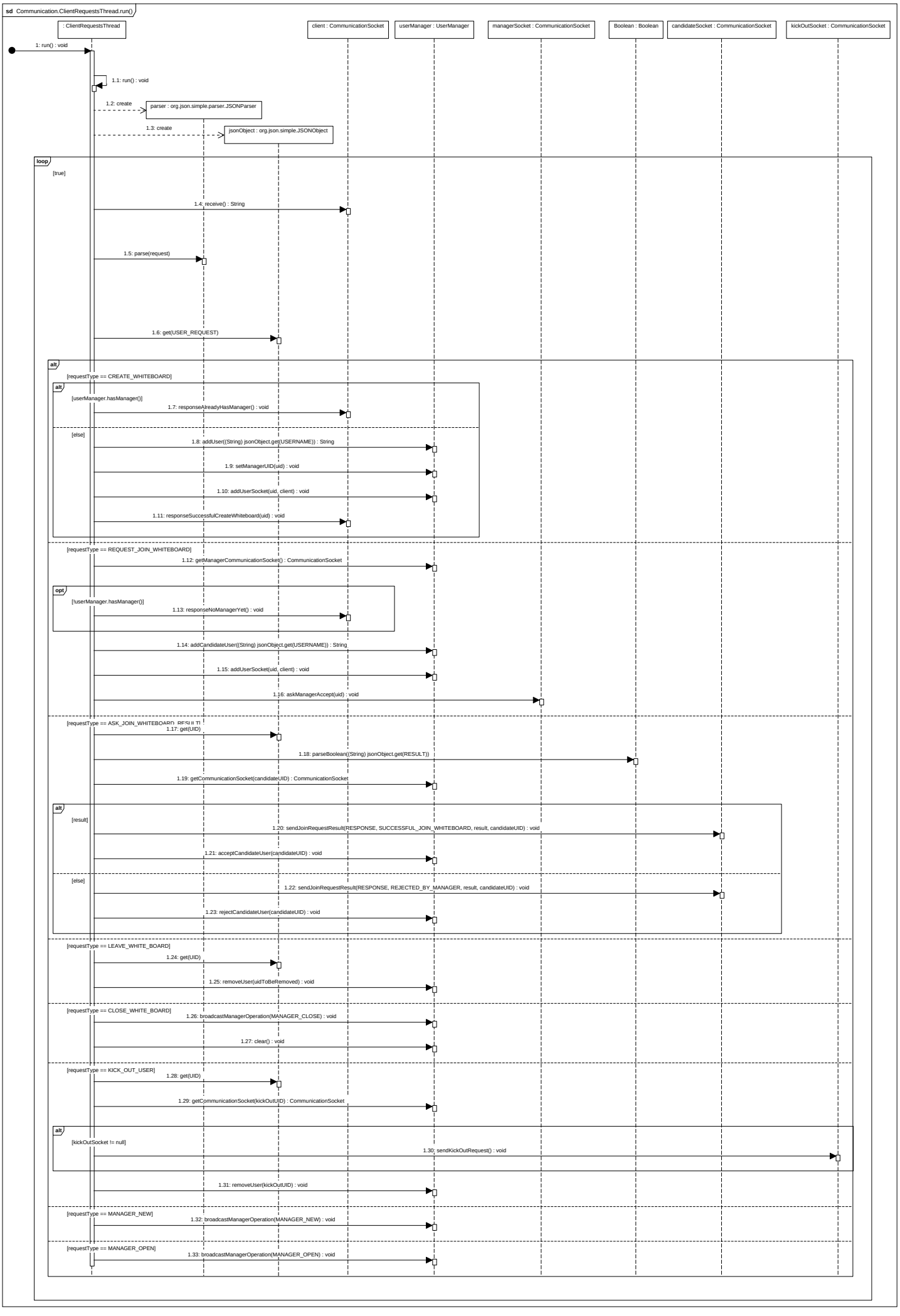
Sequence diagram

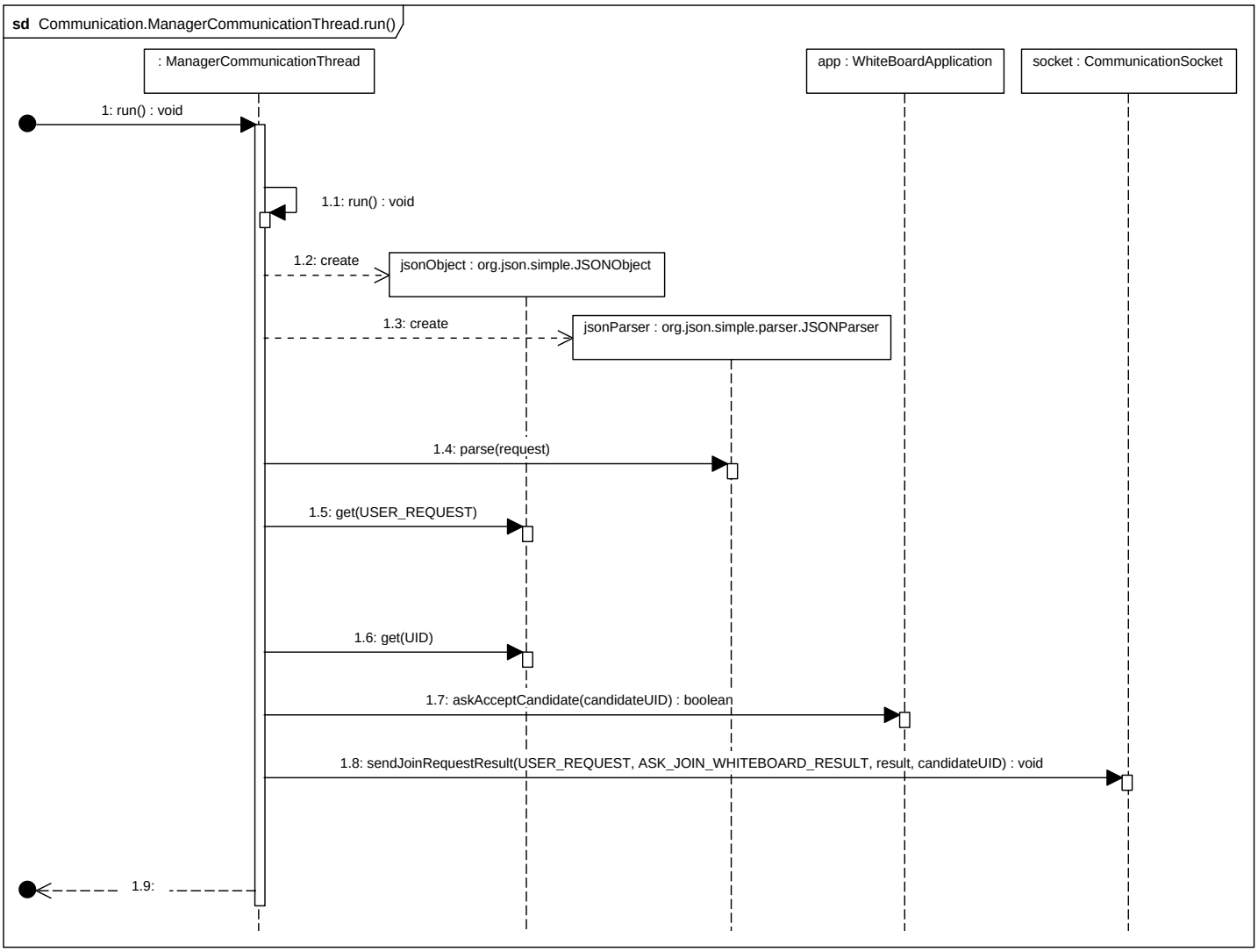
The first sequence diagram is the process for the server to parse the client's TCP requests. Firstly, it receives the request string. Secondly, it parses it as JSON object. Thirdly, it performs action according to the request message type.

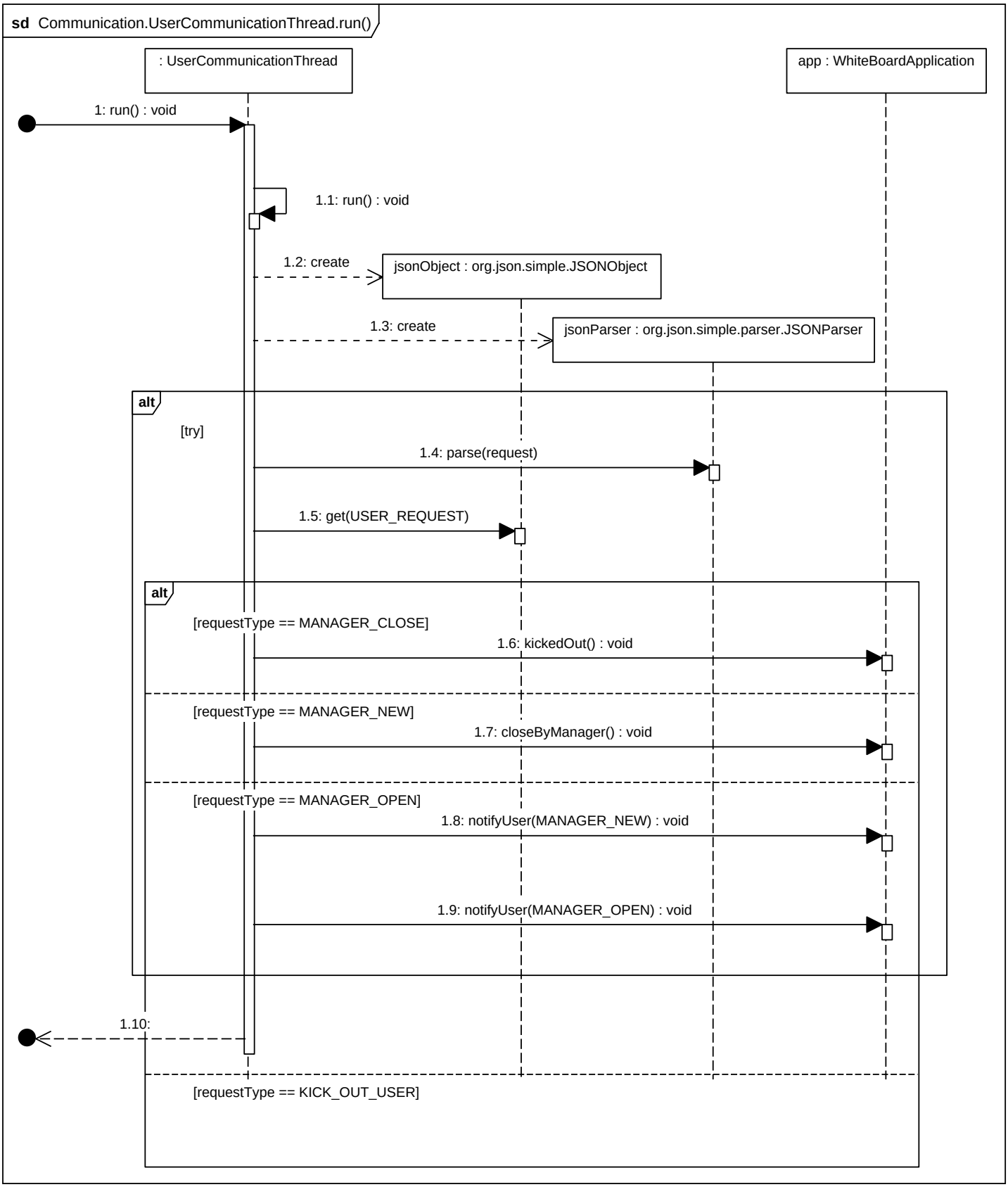
The second sequence diagram is the process for the manager to parse the server's TCP requests. Firstly, it receives the request string. Secondly, it parses it as JSON object. Thirdly, it performs action according to the request message type. The manager returns acceptance/rejection to the user's request to ask to join the whiteboard.

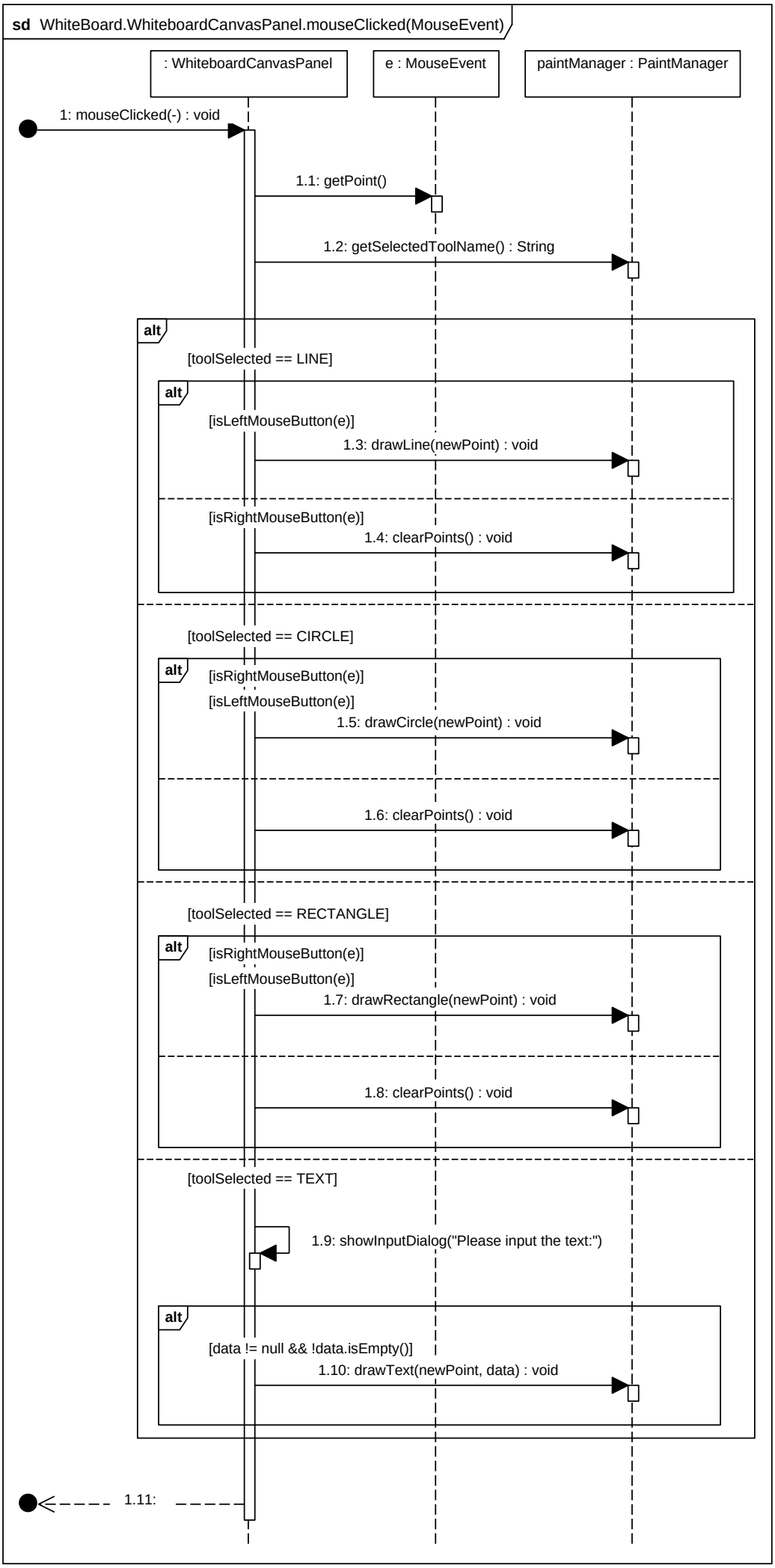
The third sequence diagram is the process for the user to parse the server's TCP requests. Firstly, it receives the request string. Secondly, it parses it as JSON object. Thirdly, it performs action according to the request message type. The application will notify user with the notification from the server to the user.

The fourth sequence diagram is the process for the application to response when the user draws the shape on the whiteboard by clicking on the canvas. It receives the point on the canvas that the user clicked and invokes various methods in the PaintManager according to the tools that the client has selected.

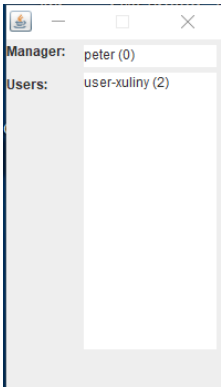

sd Communication.ClientRequestsThread.run()
: ClientRequestsThread
client : CommunicationSocket
userManager : UserManager
managerSocket : CommunicationSocket
Boolean : Boolean
candidateSocket : CommunicationSocket
kickOutSocket : CommunicationSocket
1: run() : void
1.1: run() : void
1.2: create
parser : org.json.simple.parser.JSONParser
1.3: create
jsonObject : org.json.simple.JSONObject
loop
[true]
1.4: receive() : String
1.5: parse(request)
1.6: get(USER_REQUEST)
alt
[requestType == CREATE_WHITEBOARD]
alt
[userManager.hasManager()]
1.7: responseAlreadyHasManager() : void
[else]
1.8: addUser((String) jsonObject.get(USERNAME)) : String
1.9: setManagerUID(uid) : void
1.10: addUserSocket(uid, client) : void
1.11: responseSuccessfulCreateWhiteboard(uid) : void
[requestType == REQUEST_JOIN_WHITEBOARD]
1.12: getManagerCommunicationSocket() : CommunicationSocket
opt
[!userManager.hasManager()]
1.13: responseNoManagerYet() : void
1.14: addCandidateUser((String) jsonObject.get(USERNAME)) : String
1.15: addUserSocket(uid, client) : void
1.16: askManagerAccept(uid) : void
[requestType == ASK_JOIN_WHITEBOARD_RESULT]
1.17: get(UID)
1.18: parseBoolean((String) jsonObject.get(RESULT))
1.19: getCommunicationSocket(candidateUID) : CommunicationSocket
alt
[result]
1.20: sendJoinRequestResult(RESPONSE, SUCCESSFUL_JOIN_WHITEBOARD, result, candidateUID) : void
1.21: acceptCandidateUser(candidateUID) : void
[else]
1.22: sendJoinRequestResult(RESPONSE, REJECTED_BY_MANAGER, result, candidateUID) : void
1.23: rejectCandidateUser(candidateUID) : void
[requestType == LEAVE_WHITE_BOARD]
1.24: get(UID)
1.25: removeUser(uidToBeRemoved) : void
[requestType == CLOSE_WHITE_BOARD]
1.26: broadcastManagerOperation(MANAGER_CLOSE) : void
1.27: clear() : void
[requestType == KICK_OUT_USER]
1.28: get(UID)
1.29: getCommunicationSocket(kickOutUID) : CommunicationSocket
alt
[kickOutSocket != null]
1.30: sendKickOutRequest() : void
1.31: removeUser(kickOutUID) : void
[requestType == MANAGER_NEW]
1.32: broadcastManagerOperation(MANAGER_NEW) : void
[requestType == MANAGER_OPEN]
1.33: broadcastManagerOperation(MANAGER_OPEN) : void


sd Communication.ManagerCommunicationThread.run()
: ManagerCommunicationThread
app : WhiteBoardApplication
socket : CommunicationSocket
1: run() : void
1.1: run() : void
1.2: create
jsonObject : org.json.simple.JSONObject
1.3: create
jsonParser : org.json.simple.parser.JSONParser
1.4: parse(request)
1.5: get(USER_REQUEST)
1.6: get(UID)
1.7: askAcceptCandidate(candidateUID) : boolean
1.8: sendJoinRequestResult(USER_REQUEST, ASK_JOIN_WHITEBOARD_RESULT, result, candidateUID) : void
1.9:

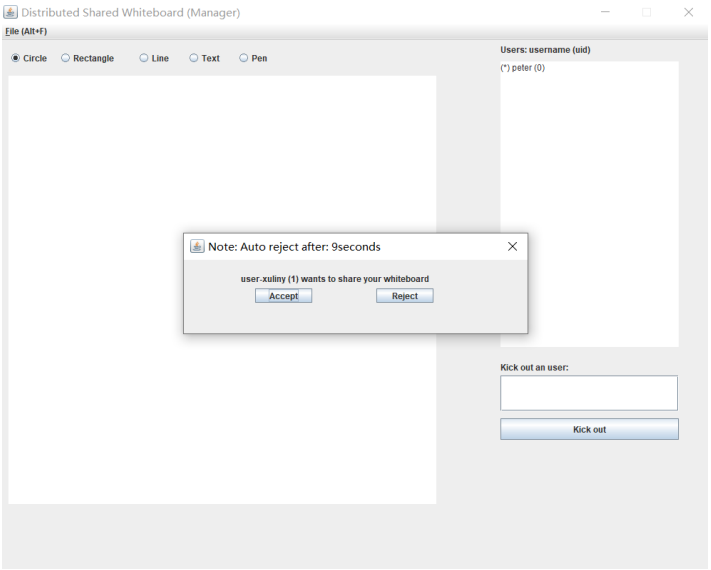

sd Communication.UserCommunicationThread.run()
: UserCommunicationThread
app : WhiteBoardApplication
1: run() : void
1.1: run() : void
1.2: create
jsonObject : org.json.simple.JSONObject
1.3: create
jsonParser : org.json.simple.parser.JSONParser
alt
[try]
1.4: parse(request)
1.5: get(USER_REQUEST)
alt
[requestType == MANAGER_CLOSE]
1.6: kickedOut() : void
[requestType == MANAGER_NEW]
1.7: closeByManager() : void
[requestType == MANAGER_OPEN]
1.8: notifyUser(MANAGER_NEW) : void
1.9: notifyUser(MANAGER_OPEN) : void
1.10:
[requestType == KICK_OUT_USER]


sd WhiteBoard.WhiteboardCanvasPanel.mouseClicked(MouseEvent)
: WhiteboardCanvasPanel
e : MouseEvent
paintManager : PaintManager
1: mouseClicked(-) : void
1.1: getPoint()
1.2: getSelectedToolName() : String
alt
[toolSelected == LINE]
alt
[isLeftMouseButton(e)]
1.3: drawLine(newPoint) : void
[isRightMouseButton(e)]
1.4: clearPoints() : void
[toolSelected == CIRCLE]
alt
[isRightMouseButton(e)]
[isLeftMouseButton(e)]
1.5: drawCircle(newPoint) : void
1.6: clearPoints() : void
[toolSelected == RECTANGLE]
alt
[isRightMouseButton(e)]
[isLeftMouseButton(e)]
1.7: drawRectangle(newPoint) : void
1.8: clearPoints() : void
[toolSelected == TEXT]
1.9: showInputDialog("Please input the text:")
alt
[data != null && !data.isEmpty()]
1.10: drawText(newPoint, data) : void
1.11:

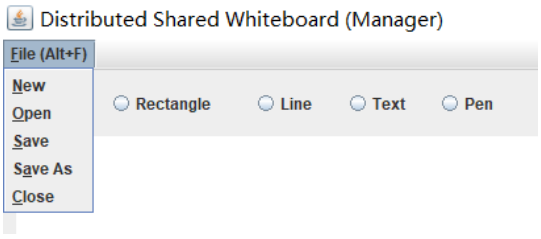
Implementation details



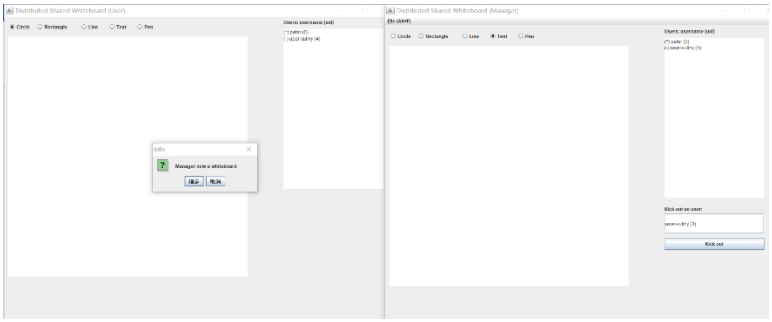
This is the server GUI. To see all the users who are using the whiteboard.



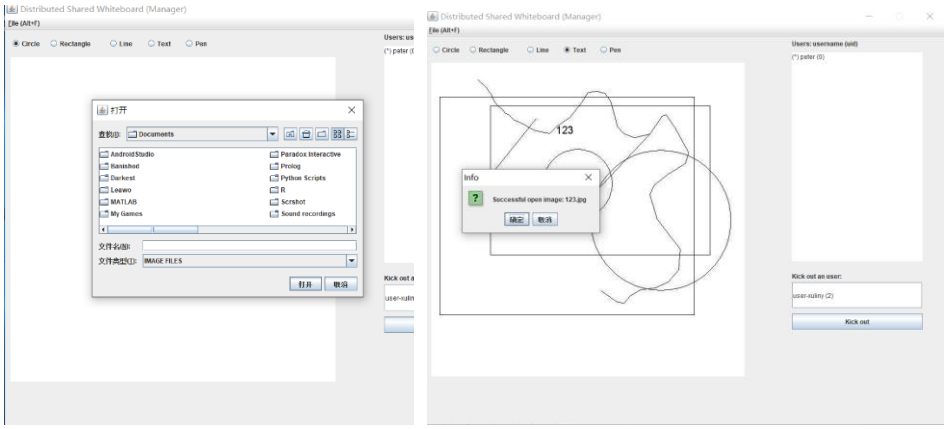
The manager can accept or auto reject timeout in 10 seconds user's request to join the whiteboard.



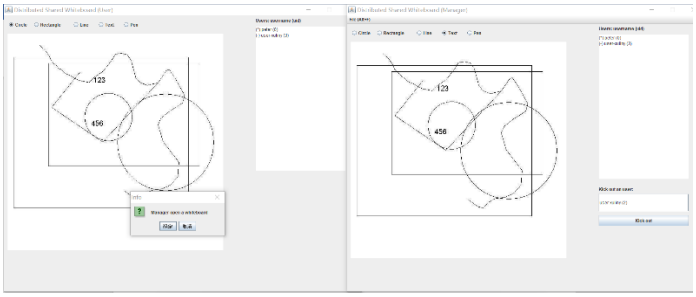
The manager have the privileges to new/open/save/saveAs/close the whiteboard.



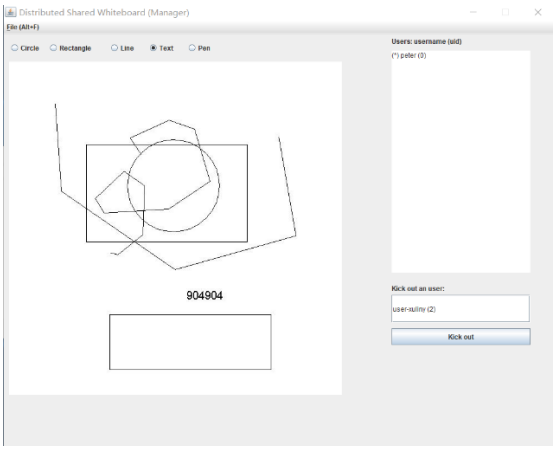
When the manager new a whiteboard, all the users will be notified.

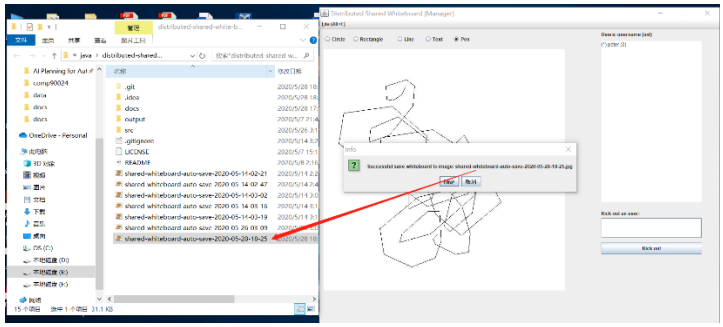


Manager can open a saved whiteboard.

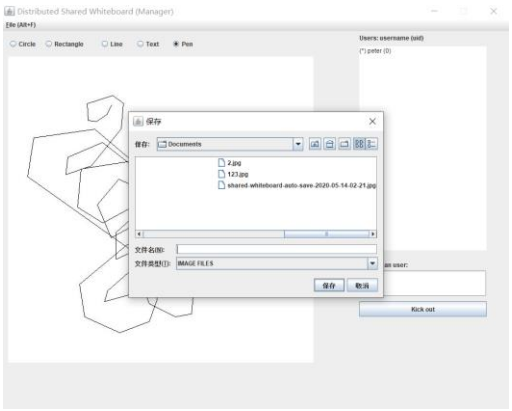


And the user will be notified

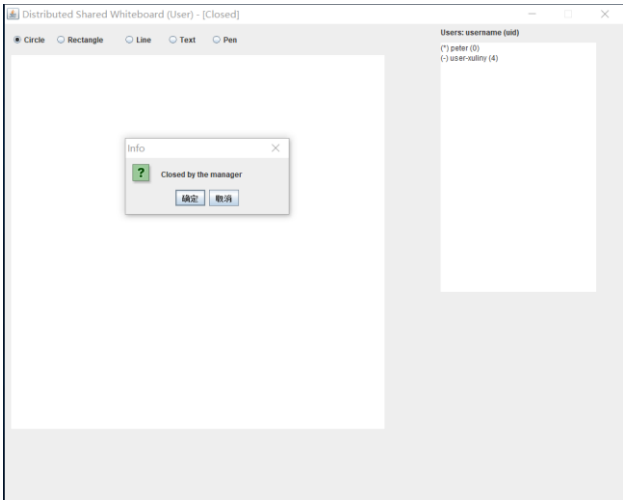




The manager can save the whiteboard to a the default directory with an auto generated file name.

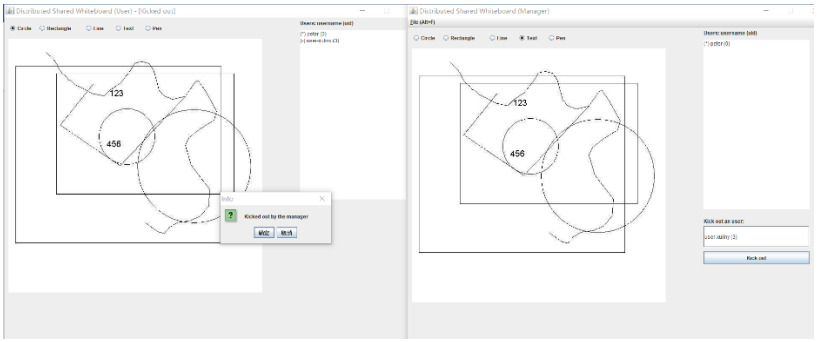


The manager can save the whiteboard by specifying the path and the file name.



Users will be notified if the manager has closed the whiteboard.

All the clients can draw circle, rectangle, lines, text on the canvas and updated simultaneously on all the clients.



User will be notified if kickout by the manager.