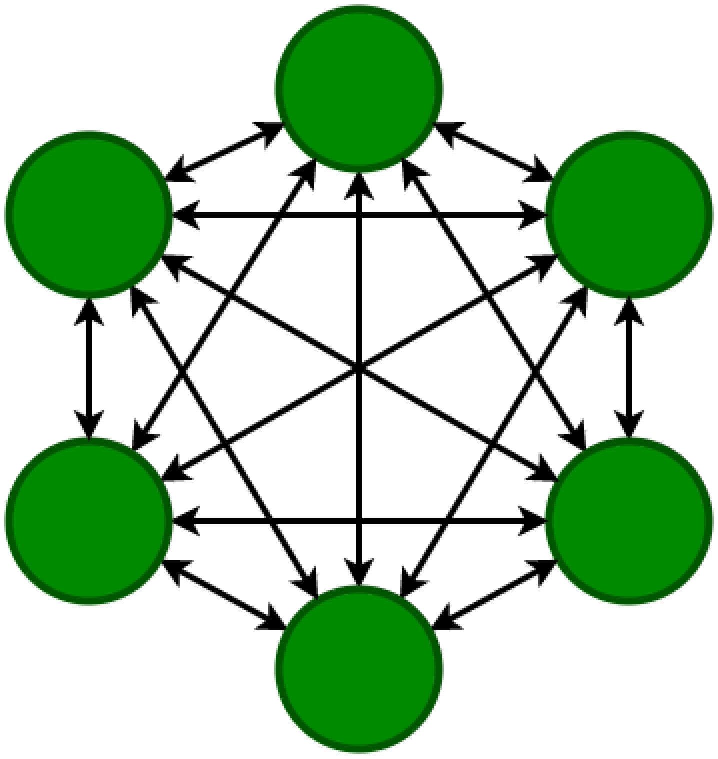
Chat Application - Design

Kevin Imlay
Matthew Flanders
Randy Duerinck
Yasmin Vega

Topology Description

Logical View

The mesh topology is when every node in the topology is connected to every other node within the topology, or in other words, a full mesh. Each connection is bi-directional, meaning messages can be sent from one node to every other node with only one hop.



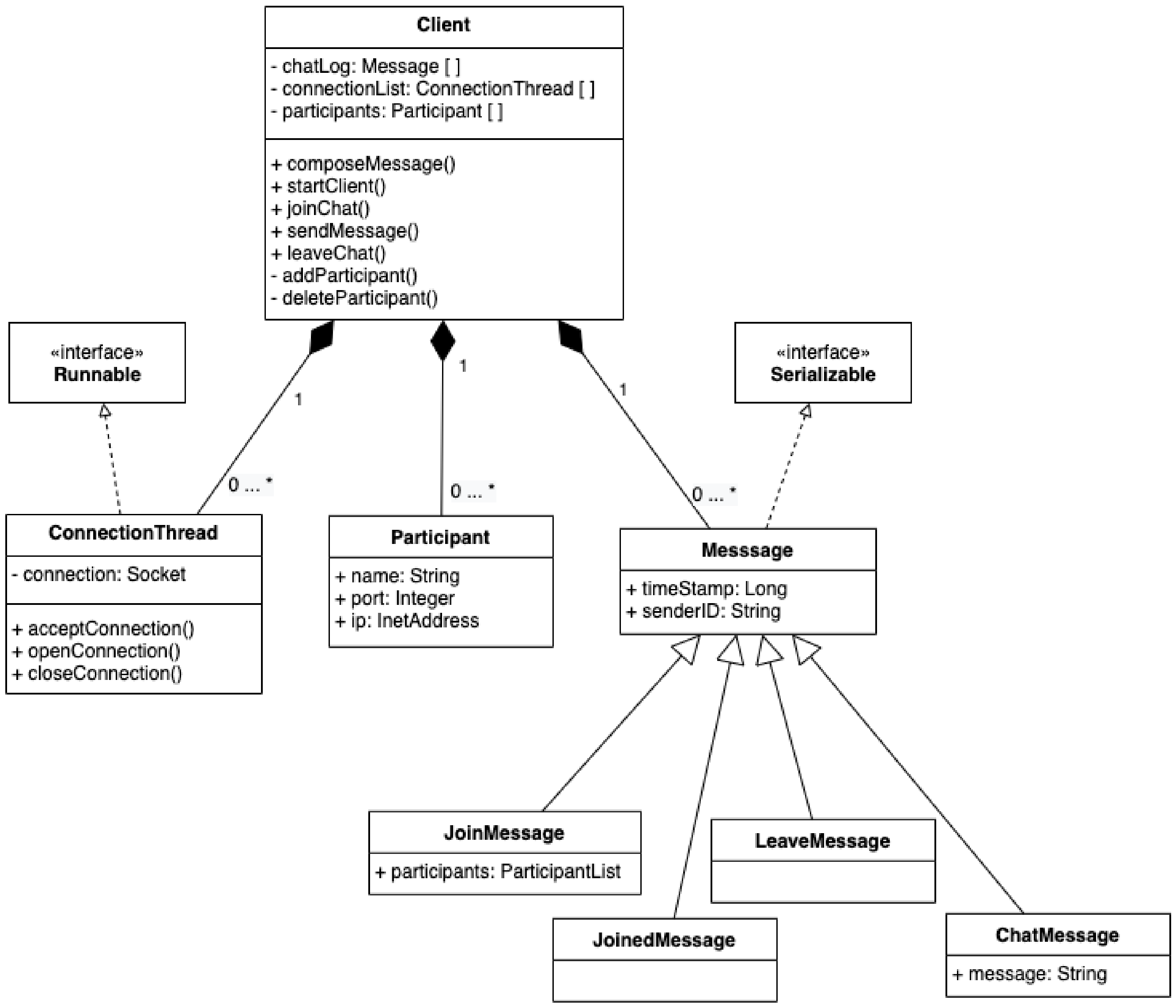
Transactional View

Joining a Chat:
When a new node joins the topology, it initially connects to one known node using that known node's IP address and port number. The known node will pass a copy of its participant list to the new node and add the new node into its own participant list. The new node will use the received participant list to then connect to every other node in the topology, and each node that was connected to will add the new node to their participant lists.

Leaving a Chat:
When a node leaves the topology, the participant sends a message to every node in the chat indicating its departure. This message contains the leaving participant node’s connectivity information and logical name. Once the nodes in the topology receive the leaving node’s message, they remove the leaving node from their participant list. The connections the leaving node holds with the rest of the topology will be severed and the nodes in the topology will sever their connection with the leaving node.

Sending/Receiving a Message:
When a node sends a message, that message will be sent through every connection that the node holds. This will distribute the message to every other node in the topology in just one hop. There is no need for the receiving nodes to forward the message because by the definition of a full mesh, every node will have received a copy.

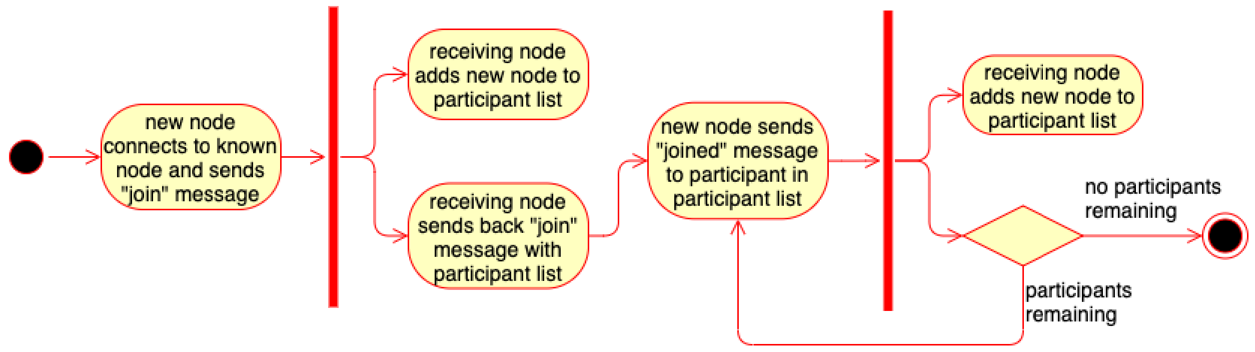
Class Diagrams



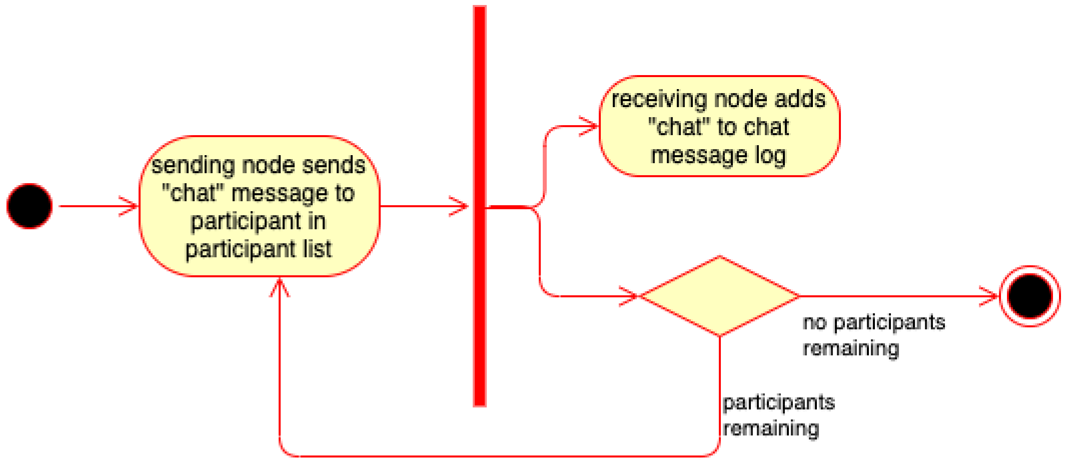
The Client class is responsible for managing the chat application. It has a chat log for storing incoming chats, a list of other clients it's connected to, and a set of threads that manage the connections to the connected clients. The ConnectionThread class manages a connection, and runs as a separate thread so all connections can send and receive concurrently. The Participant class keeps track of a participant that is connected to, storing a chat name to display in the chat and the IP address and port number for the connection. The Message class is an abstract class and a parent to the four different types of messages that are shared in the chat system. All messages are serializable so that their objects can be sent over the network connection and turned back into objects on the other side. JoinMessage includes a field for returning a participant list when a new client is added. ChatMessage includes a message field for transferring the chat message text. JoinedMessage and LeaveMessage do not need to transfer any additional information.

Activity Diagrams

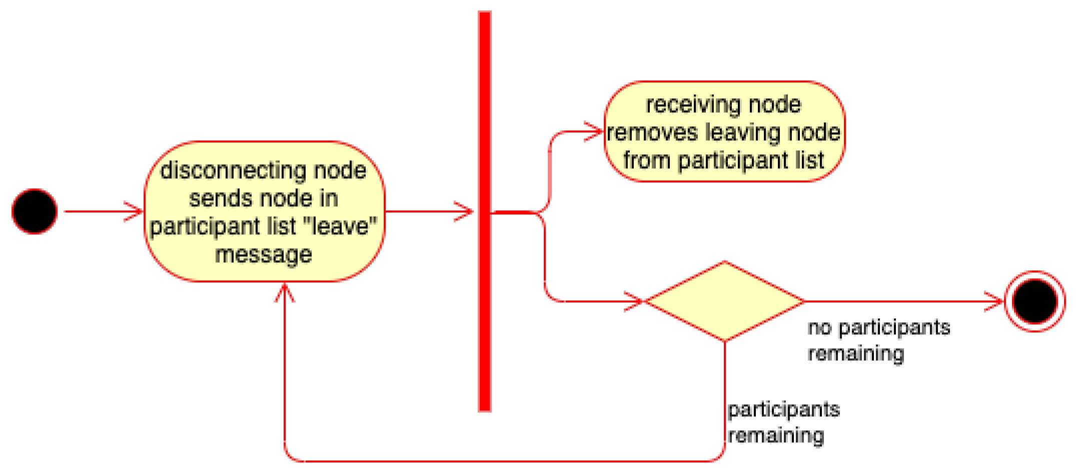
This is the “dynamic” view of our application. To begin with, a participant joins the chat by sending a “join” message to a known participant already in the chat. The existing participant will receive the message, and add the joining participant into its participant list. Then, the existing participant will send a “join” message with the participant list. The joining participant receives the participant list and sends a “joined” message to everyone in the participant list. Anyone that receives this “join” message will add the new participant into their own participant list. The joining participant is now fully connected to everyone in the chat. If a participant wishes to send a “chat” message, the participant will be sending the “chat” message to everyone in the participant list. Everyone receiving the “chat” message will be displaying the message to the chat message log. If a participant wishes to leave the chat, they will send a “leave” message to everyone in the participant list. Those that receive a “leave” message will remove the leaving participant from their participation list. Further information about the messages can be found in the Message Protocol section below.


new node connects to known node and sends "join" message
receiving node adds new node to participant list
receiving node sends back "join" message with participant list
new node sends "joined" message to participant in participant list
receiving node adds new node to participant list
no participants remaining
participants remaining

Joining a node into the topology. A joining node connects to a node known to be in the topology. That known node sends back a participant list, and the new node iterated over this list, forming connections with each participant. All nodes being connected to add the new node to their participant lists.


sending node sends "chat" message to participant in participant list
receiving node adds "chat" to chat message log
no participants remaining
participants remaining

Sending a message. The sending node iterates through the participant list, sending the chat message to each node.


disconnecting node sends node in participant list "leave" message
receiving node removes leaving node from participant list
no participants remaining
participants remaining

Removing a node from the topology. The disconnecting node iterates over its participant list, sending a disconnect message to each node. Each receiving node removes the disconnecting node from their participant list. Then the disconnecting node leaves the topology.

Messaging Protocol

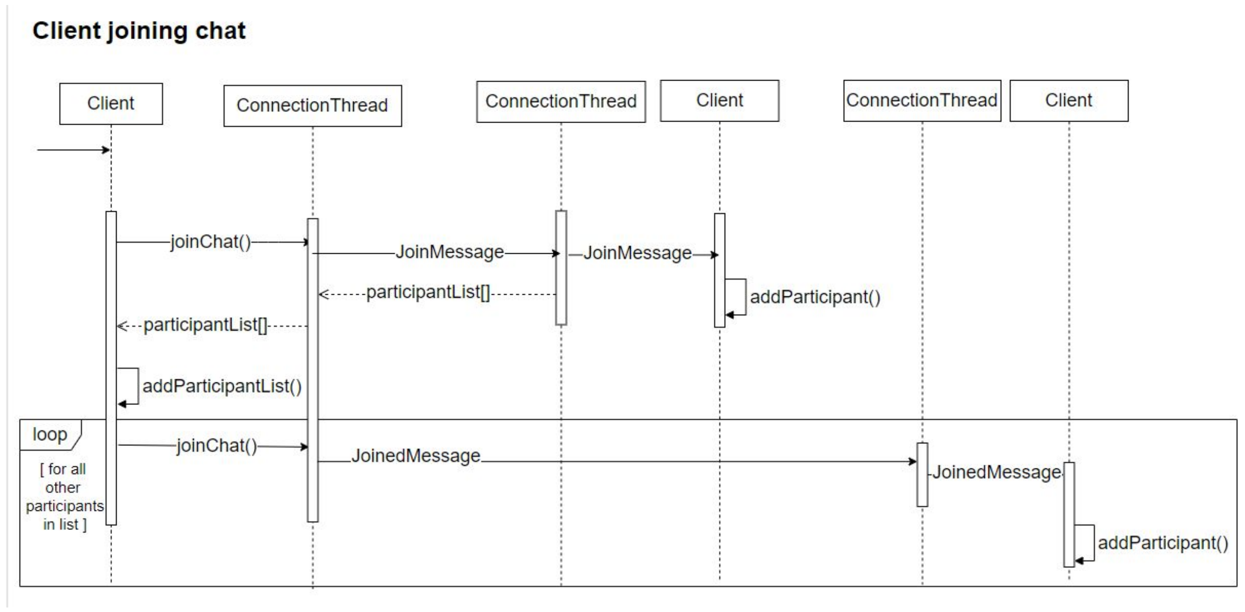
Message Types

Join:
The “join” message is used to create a request to join the chat. It includes the IP address and port number of a participant in the chat. A network connection will be opened up using the credentials in the message. The node that received the “join” message sends back the participant list. At this point there is only a connection between the joining node and the node that sent the participant list back.

Joined:
The “joined” message includes the node’s IP address, port number, and logical name who just joined with a participant. The message is sent to the rest of the nodes in the chat. Those who receive the message will input the information into the participant list.

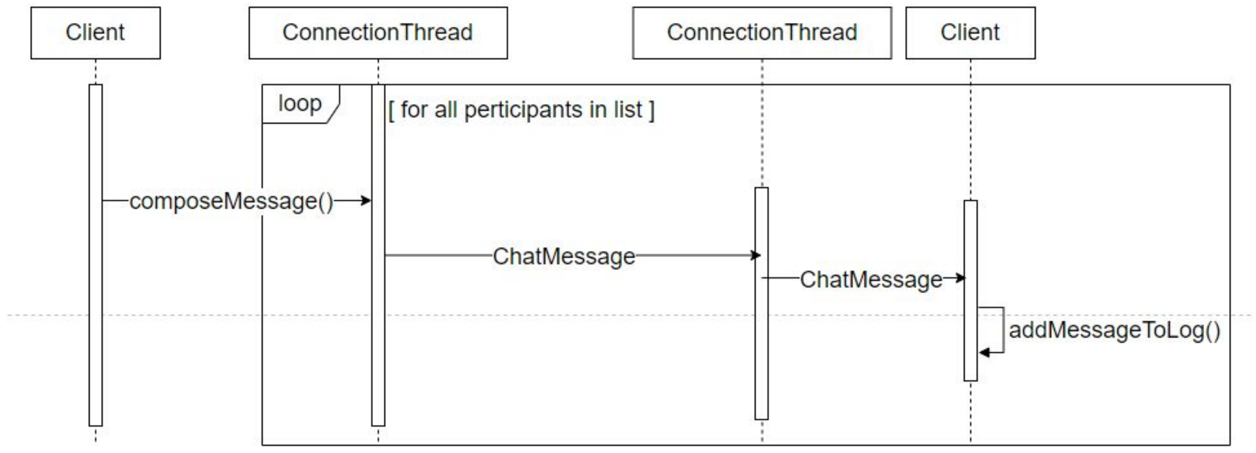
Chat:
The “chat” message includes the sending node’s logical name and message data. The sender sends this message out to everybody in the chat. When a node receives a “chat” message, all they do is display that message to the message log.

Leave:
A “leave” message contains the leaving node’s connectivity information and logical name. It is sent to everybody in the chat. Upon reception of a “leave” message, the receiving node removes the leaving node from the participant list.



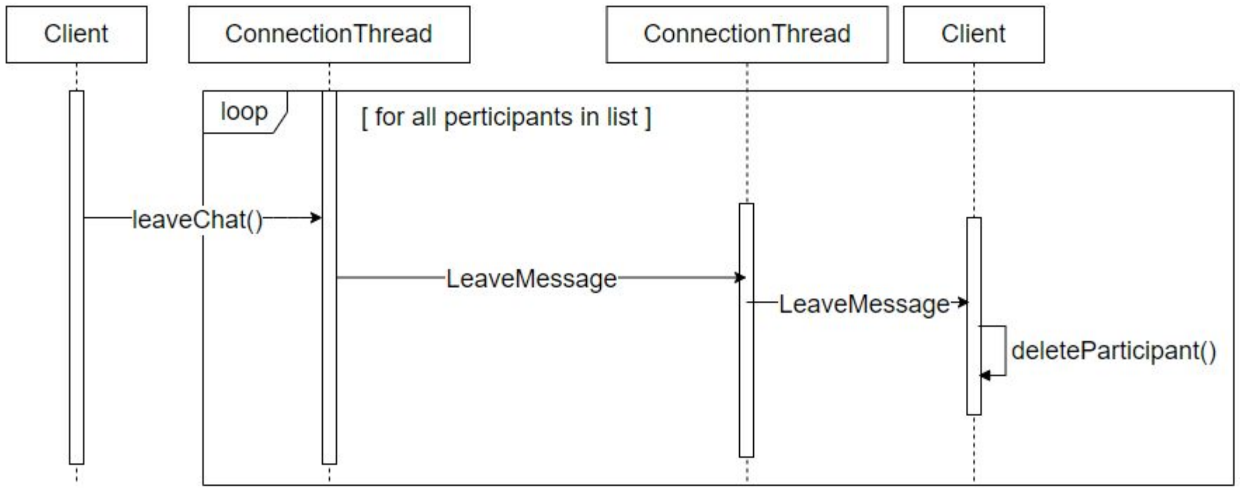
Class actions taken to join an existing chat. Client calls joinChat which the ConnectionThread sends a JoinMessage to the known client’s ConnectionThread. The connectionThread returns its participant list and the client adds the new participant. The client joining chat adds all participants in the returned list to its participantList. It then loops across all participants in the list and creates a new connection to them while those participants add the new client to their participantList.

Client sending message



Class actions taken to send a message. Client calls composeMessge() the ConnectionThread takes the message and loops through all participants in the list sending them the ChatMessage. When the ChatMessage is received by the client addMessageToLog() is called to show the message.

Client leaving chat



Class actions taken to leave chat. Client calls leaveChat() where the ConnectionManager sends a LeaveMessage to all participants in the list. The connection is then closed and the client wanting to leave chat is removed from each participant's participantList.