Distributed Chat, Part 3 - Implementation

Joe Domabyl V
Daniel Rydberg
Nick Nannen

Implementation Problems:

The key problem our team had implementing our design was with our messaging protocol. The JOIN and NOTE message types worked as intended, but the LEAVE and SHUTDOWN message types did not. As each of our peers possesses a list of active chat participants, each list of each peer must be modified everytime a peer sends a LEAVE or SHUTDOWN message. As this is a multithreaded process, there was a race condition whenever two peers attempted to write to the same participants list at the same time, leading to the ConcurrentModificationException error, and an ungraceful termination of the program.

Design Deviations:

We decided to add an additional class called SenderWorker - which extends the Thread class. Our Sender class uses it to send a message from its peer to all of the other peers currently in its participants list. A new ServerWorker object is created whenever a peer sends a message. In our original design, this process was meant to be handled by the Sender class itself. We decided that a separate class was warranted because in a true peer-to-peer network, each peer must be capable of sending a message to each other connected peer. This is in principle similar to our Receiver and ReceiverWorker classes from our original design. Finally, we decided not to implement a MessageQueue, as discussed in Part 2. The addition of the SenderWorker rendered the queue completely unnecessary.