Holochat Call States and Message Flow

Arnaud Fickinger

2018

1 Simulation of Call flow 1 with Java threads

To familiarize myself with the first call flow, I simulated it on Java. I created a class "Client" which extends Thread and to go faster (because simulation of call flow 1 is not really the project's goal), I created two classes "Alice" and "Bob" extending "Client". Both "Client" share an object "Communication", a class containing methods "userActionCall..." that I made synchronizable using a ReentrantLock with Conditions "messageCall...". If you run the programm (callflow1.jar) you will see 2 windows containing each 1 Client's log with state change, received and sent information. CallFlow1Threads.png show a screenshot of the programm and you have the code in the folder "Call Flow 1 Threads Simulation".

2 Datastructure of the state machine

I choose to use enum instead of simple integer for my states and events in order to improve the lisibility of my code and to make it easier to add events and states in the future. Firstly I wanted to create a matrix of size |states|*|events| and put null where the transition is not valid, but it's a lost of space. I rather see my state machine as a graph where states are nodes and events are edges. I create a HashMap<Event,State> for each States, so I can find the next State in constant time and I use a minimum of space. When we look at the call flow 1 we can see that when A is in some state, he automatically sends events, so I would like to add a method "action()" to each State that will run at state change. I googled "functors in java" and I found out that functors don't exist in Java... It's not a problem, I will create a Runnable field named "action" in my State class. The problem is that some actions need information about the Client that contains the State field, which would be possible if State were a class and not a enum. Because all action are simple event sender, I just have to add a field EventToSend in my enum State which corresponds to the event I want to send when I'm changing to a certain state. I had to add two states in Call Flow 1 to fit my datastructure (see NewCallFlow1.png). All of Clients methods return boolean so I can quickly check if an error occured.

3 App Developpement

For the backend I chose to use Cloud Firestore as realtime DB. It's the new DB service (brought out end 2017) from Firebase which uses a new datastructure (no more JSON, easier to organize complex data). It's the first time I use this new service. For event listening my first idea was to create a document (in Cloud Firestrore data is organized in "document" and "collection") containing 3 fields : sender, receiver and event. But it meant that every users had to listen to every documents of the collection events, even if they are not the receiver. My second idea was to use a Listener on the document ReceiverId which contains two fields SenderId and Event.