Manual

1. How to use the code?

Importing the code into Eclipse:

a. File → New → Java project
    i. Choose name "DS" ➔ finish

!!Note: requires java version 1.7 or newer!!

b. Drag the files of the "src" folder to "src" in the Eclipse IDE
c. Right click on "DS" → Build Path → Configure Build Path
    i. Go to libraries → Add external JARS
        1. Select all .jar files from the "lib" folder (except "avro-tools-1.7.7.jar")
        2. Apple ➔ OK

Single computer functionality

!! Note that you always run the controller before any client. Otherwise you will get the error message "Cannot establish connection with the controller!"!!

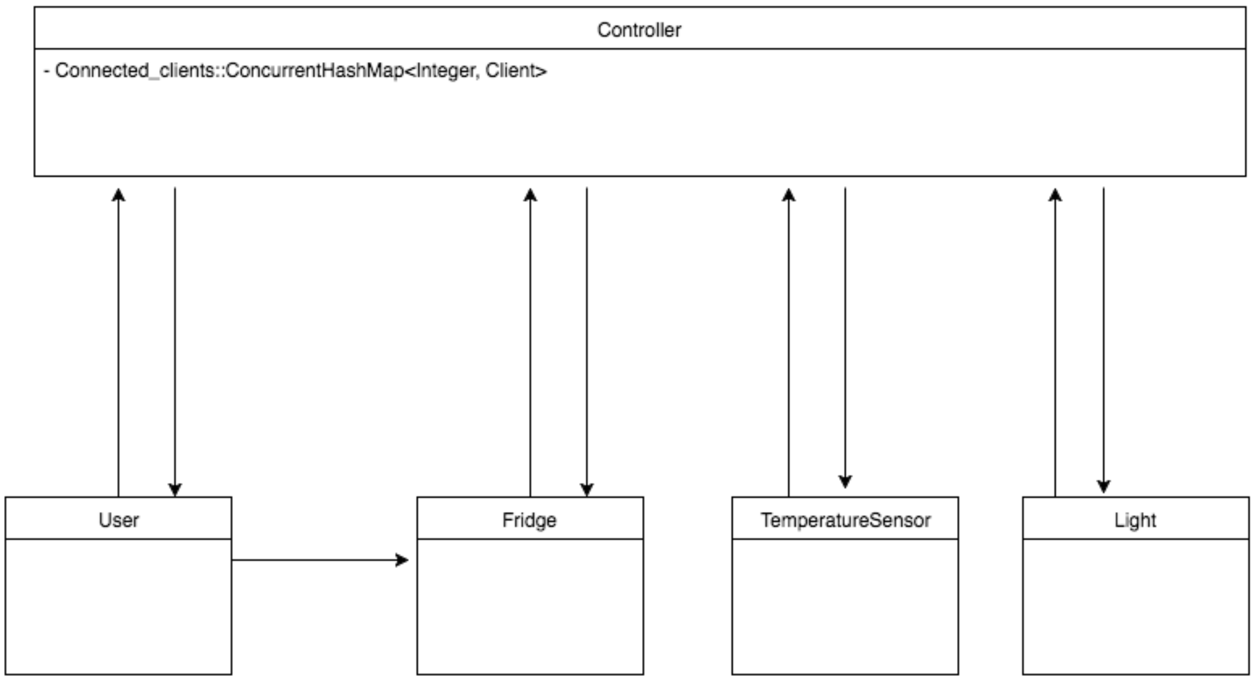
a. Go to controller/Controller.java in the package explorer
    a. Click the green arrow
        i. The controller-server is now running

    Note: If new controller is elected and you want to restore old controller ➔ give IP address and port of new controller as arguments and run controller/Controller.java.
b. Go to clients
    a. Select the desired client and click the green arrow
    b. Choose your IP address and inet-address
        i. The client is now connected with the controller and running
    c. Type "list" to see which commands you can execute with this client

Distributed functionality

a. Type "ifconfig" in the Terminal (@server side) to determine its IP address
b. Give the IP address (see previous step) and port (found @server side pc) from the controller as arguments for the client you want to run.
c. Now follow the steps as described in the single computer functionality section.

2. Architecture

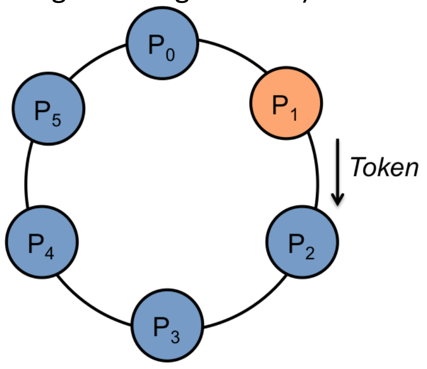


Pinging:

Controller pings to every client to see if it is still responding. The client also pings but only to the controller. This pinging method is a frequently asked method in distributed systems. Pinging allows us to keep the hash map of connected clients up to date.

Fault tolerance:

When the controller fails, the clients arrange themselves into a ring and start the election process (according to Chang-Roberts).



Threads:

The user interface runs on its own thread (in client and controller) to allow for true fault tolerance.

Time synchronization:
I chose the Cristian Algorithm for this part. This algorithm will adapt the internal clock of the temperature sensors as follows:

a. T_r = Time request is sent to time server
b. T_a = Time answer (from time server) is received
c. New time of the sensor = $Time_{server} + (T_a - T_r)/2$

I chose this algorithm because it was the best fit for this project. The controller is the timeserver and has his own clock (which is initially 0, so if there is an election in which a new controller is selected the clock will be 0 again). The temperature sensors each have their own internal clock which will sync after 5 seconds. You can change the synchronization interval using the "changeUT" command in the UI of the sensor.

Consistency model:
The project is based on client-centric consistency. The data items have an owner (fridge etc.). Moreover, it is a "Read your writes" consistency because an item added by a process in a fridge will be always available to a successive inventory/read operation performed by the same process on the fridge. For replication purposes the clients will periodically (every 2 seconds) request backup from the server.

3. Used technologies and libraries

- Eclipse (Open source IDE)
- Apache Avro (Communication between server and clients and the mutual communication between the clients)
- Imports:
  - Java.util (Map, concurrent hashmap, list, random etc.)
  - Java.io (IOException, InputStreamReader, BufferedReader etc.)
  - Java.net (InetAddress, InetSocketAddress etc.)
  - Java.lang.reflect.UndeclaredThrowableException
  - Java.Date (formatting of the time for the time synchronization part)