Distributed Systems 3d Homework

Artjom.Lind@ut.ee

December 16, 2015

The deadline for submitting is the 15th of January 2016. You can work in pairs. You can rely on provided code of FCF v1.0 or FCF v2. You can use you own code of FCF v1.0 or FCF v2.0. As we agreed in the last seminar, HW003 will be split into three different difficulty levels:

1. Level is obligatory to submit till 15th of January 2016
   (a) This one however depends on HW001[1] or HW002[2], so at least one of them must be 100% functional.
2. Level is optional and may be delivered after 15th of January 2016
   (a) This one also depends on HW001[1] or HW002[2]
   (b) Does not depend on Level 1 task, but optionally may be integrated with it.
3. Level is optional and may be delivered after 15th of January 2016
   (a) This one requires Level 1 task to be 100% functional
   (b) Does not depend on Level 2 task, but optionally may be integrated with it.

Now the rules are defined, let's discuss the details.

1 Difficulty Level I

1.1 Intro

Here we take the one of implemented HW001[1] or HW002[2] as base.

1. HW was about multi-player real-time game (FCF version 1.0), where we designed the primitive 2D game with simple logic. The core task was to allow the user to control his/hers character on server; and of course provide the user with visual feedback. We agreed we design the corresponding protocol from scratch on bare TCP connections. The advanced part was game-client with graphical UI, which was not obligatory as the command-line based rendering was also acceptable.

2. HW was about improving the HW001 solution:

   (a) Introducing object-oriented design

   (b) Removing bare TCP connections as well as the custom made protocol

   (c) Introducing Remote Objects

      i. Controlling player's character explicitly on server using Proxy objects

      ii. Querying the player's visibility area explicitly from the server using Proxy objects

   (d) Minimal modification to the Game logics (refactoring for object oriented design should have demanded the drastic changes in the Game logic).

So here we are with two solutions we can reuse in this homework. Before we actually start discussing the specs., let's discuss the disadvantages that both HW001 and HW002 still have.

1.2 Disadvantages

Let's agree we here focus on the distributes system related disadvantages and not the game logics, rendering or UI . So far we had two applications: server code and the client code. In order to start the game we started server code first, the we waited for players to connect. We needed however manually inform players about server's IP address in order to play together (as the IP address will eventually change if we join with different networks: each new game session we had to re-inform client's about server's new IP address). Client had to manually provide the IP address by typing it into GUI window or providing command line arguments. Additional disadvantage: if the user, responsible for running server was willing to join the game he had to start the client code separately and connect to his own server running on the same host.

As the real-time multi-player games appeared back in the past, the described issue was solved, and the well established pattern was put in use:

- User runs only one executable for game application (no separate files for client and server).
  - User has an option of multi-player game
    - User has an option to start his/hers own game server
      - User joins the created game session automatically
    - User has an option to join existing game session
      - By default the game servers in the local network (LAN) are automatically discoverable. User may select one of them to join the game session.

· Optionally user can provide manually an IP address of the server (in our implementation we will not use this option).

Further we will discuss the specs. of the changes we need to introduce into our code having in mind the previously described pattern.

1.3 Specification

1. Join the client code and sever code into one executable
   - One exe, jar etc., in case of python leave only one *.py file having main method (there can be many *.py files still but only one with main method).
   - In case of command line based application
     - Additional command line argument specifies if server or client needs to be started
   - In case of GUI
     - Additional radio-box specifying "server" or "client" and "start" button
2. Modify the server code
   (a) Server shall now be started separately from the main method or GUI if user specifies so.
      i. So if you specify the game field size from user provided value, make sure user has this option here too, you may use
         A. separate command line argument
         B. additional input field in GUI
   (b) (Optional) After server is started the user is automatically suggested with the player initialization dialogue:
      i. Provide player name
      ii. Provide player character (frog or fly)
      iii. Join the game session running on the server the user have just created
         A. user does not need to provide any
            - Server IP in case of HW001
            - Remote Object URI in case of HW002
         B. client-server communication is not needed, as the player object and the game itself are in the same runtime, so the user can control the player object directly.
      iv. User may not join the game if he just wants to hold the server and watch the others playing (spectator mode).

(c) After server is started the separate thread responsible for server announces is also started

i. We implement here simple LAN broadcast or multi-cast messages, in the same way we did it for the Lab012[3]

A. We obviously need separate port (of kind UDP) in order to send out

B. Announce-thread does send broadcast/multi-casts each T interval (5 seconds is OK)

- In case of broadcasts the choice of target port is free, just make sure the client uses the same when listening to broadcasts.
- In case of multi-casts the choice of multi-cast group if free, just make sure the client subscribes to the same group.

C. Warning! if you decided to rely on the code of HW001, then the content of broadcast/multi-cast message if free to choose, just make sure the client expects the same

- Hello message
- Server's version
- Servers unique identifier
- May also left empty, as the server's IP address will be anyway visible from the source field of the UDP packet's header.

D. Warning! if you decided to rely on the code of HW002 the server announce messages shall contain the corresponding URI of the Game object on server (RMI or Pyro specific URI, or any other if you use for Remote Objects ...)

(d) User has to be able to close the server, but still not kill the application and appear in main dialog (to start the server again or join the existing game session).

(e) If the user suddenly closes the the server, all the player's are of course disconnected, if the want to continue playing they have to find the other server, or agree who will start the one.

3. Modify the clients code

(a) After client is started the separate thread is started for listening for server announces

i. In case of broadcasts, separate UDP port is required for listening (make sure server actually sends broadcasts to this port).

ii. In case of multi-casts, the we need to subscribe to the same broadcast group the server is sending to.

iii. In case you rely on HW001:

A. The received messages may be checked for content, if server uses any defined content and client expects it and not anything else. This might protect if there are different servers in LAN broadcasting to the same port or multi-casting to the same group.

- If content of the message is not as expected message should be skipped.

B. If the content of the received message is not checked or is as expected, client takes the server's IP address from the UDP packet's header (the same packet we received the server's announce with).

C. The server's IP is then used to populate the the list of active severs in LAN. This list should be also reflected to GUI or command line for user to select the server to join with.

iv. In case you rely on HW002:

A. The received messages contain the RMI or Pyro (or other framework) specific URI of remote object on the server.

B. The received URI is then used to populate the the list of active severs in LAN. This list should be also reflected to GUI or command line for user to select the server to join with.

v. We may rely on the timeouts in order to forget inactive servers (in case server goes down and does not send announces anymore). Obviously the timeout shall bigger then the server's announce interval. If outdated server is discovered it should be removed from the list of active servers.

A. You may implement as separate watchdog thread to periodically check for outdated servers (work for all).

B. You may use player's input event to trigger the check (works for GUI).

C. You may use GUI redraw call to trigger the check (works for GUI).

(b) After client is started the user is provided with dialog to join the game:

i. Select the server to join the game from the list of active ones

A. Should obviously wait until at least one server is in the list

ii. Provide player's name

iii. Connect to selected server

A. In case you rely on HW001 the selected server's IP is used to establish connection and register player. Player then becomes a spectator.

B. In case you rely on HW002 the selected URI is used to connect to remote game object and register player. Player then becomes a spectator.

iv. After connected and registered to server

A. Provide player's character (frog or fly)

B. Join the game, start playing.

(c) If server is suddenly shutdown while client playing on it, client obviously loses the game session. User can however select the other server from the list of active servers and join. User can also start his own server and wait for somebody to join.

(d) User has to be able to leave the game, but not kill the application and appear in main dialog (to start the server again or join the existing game session).

2 Difficulty Level II

//TODO

3 Difficulty Level III

//TODO

References

[1] Artjom Lind. 1st Homework . https://courses.cs.ut.ee/2015/ds/fall/uploads/Main/Homeowrk1.pdf, 2015. [Online; accessed 9-Nov-2015].

[2] Artjom Lind. 2nd Homework . https://courses.cs.ut.ee/2015/ds/fall/uploads/Main/Homeowrk2.pdf, 2015. [Online; accessed 9-Nov-2015].

[3] Artjom Lind. Peer-to-Peer I. https://courses.cs.ut.ee/2015/ds/fall/uploads/Main/Lab0012.pdf, 2015. [Online; accessed 11-Nov-2015].