Distributed Systems: Assignment 3
Middleware: Peer-to-Peer Communications

Petru Eles, Sergiu Rafiliu, and Ivan Ukhov
{firstname.surname}@liu.se

January 14, 2013

1 Introduction

The goal of the current assignment is to make the code, written for the previous assignment, properly handle situations when peers join and leave your distributed system. This functionality will be delegated to a separate class, `PeerList`, which will take advantage of the object request broker from Assignment 2. Also, apart from sole communications with the name service, which we realized in Assignment 2, you will introduce interactions between peers, *i.e.*, peers will be remotely invoking methods [1] of each other. In order to put all these together and ensure that they work properly, you will implement an application for exchanging text messages between peers. Such a system of chatterers is depicted in Figure 1.

2 Your Task

2.1 Preparation

Download [2] and extract the source code for the assignment. In the archive, you will find the following files, most of which are already familiar to you:

- `lab3/chatPeer.py` — the chat application (no changes are needed);
- `lab3/test.sh` — a shell script that you can use for testing;
- `modules/Server/peerList.py` — the module that maintains a list of peers of a given type (should be modified);
- `modules/Common/nameServiceLocation.py` — the same as for Assignment 2;
- `modules/Common/objectType.py` — the same as for Assignment 2 (overwrite with your implementation);
- `modules/Common/orb.py` — the same as for Assignment 2 (overwrite with your implementation);
- `modules/Common/wrap.sh` — the same as for Assignment 1.

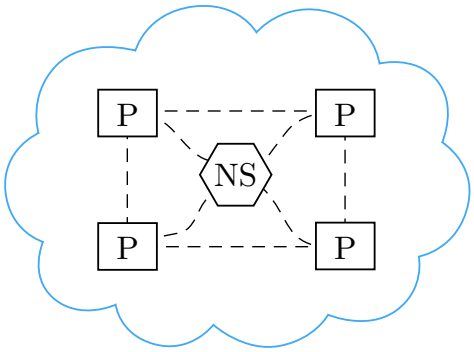

Figure 1: A name service (NS) and a number of chat-peers (P). The peers discover each other through the name service; then, they exchange messages directly.

The major functionality is concentrated in `peerList.py`; carefully read this code and understand what it does. In particular, ensure that you understand the purpose of the locks utilized throughout the module.

Try to run `chatPeer.py` in several terminals. If you have overwritten `orb.py` with your (presumably correct) implementation from Assignment 2, the application should start without any errors and display its menu as follows (note that you can specify your unique identifier with the `-t` command argument instead of `objectType.py`):

```
$ python chatPeer.py -t ivan
Choose one of the following commands:
    l                       ::  display the peer list,
    <PEER_ID> : <MESSAGE>   ::  send <MESSAGE> to <PEER_ID>,
    h                       ::  print this menu,
    q                       ::  exit.
ivan(1) > l
List of peers of type 'ivan':
ivan(1) > 2 : Can anybody hear me?
Cannot send messages to 2. Make sure it is in the list of peers.
ivan(1) >
```

Unfortunately, this instance of the application cannot see the other instances, which are running in parallel, and, therefore, cannot send messages to them.

2.2 Implementation

As you have noticed, the class `PeerList` contains two missing parts, which you are asked to fulfill. These two parts are in the following functions:

- `initialize` — the function is called whenever a new peer joins the system; it should (a) request the peer list of your name space from the name service and (b) register the current peer within each peer from the obtained list.

- `destroy` — the function is called whenever a peer leaves the system; it should unregister the current peer from each peer in the peer list of your name space.

After the implementation is completed, run the system once again and make sure that now you can exchange messages without any problems.

3 Conclusion

In the previous and this assignments, you have implemented a solid ground for your future distributed system with an arbitrary number of peers sharing the fortune database. Now, you can perform remote method invocations in a straightforward manner, thanks to your object request broker and smart peer list. However, we shall postpone the integration of this code with the code from Assignment 1 till the last assignment, Assignment 5. The reason is that there is one crucial component which is still missing. This component is a distributed lock [3], which we shall closely look at in Assignment 4.

References

[1] http://www.ida.liu.se/~TDDD25/lecture-notes/lect2-3.frm.pdf.

[2] http://www.ida.liu.se/~TDDD25/labs/assignment3.zip.

[3] http://www.ida.liu.se/~TDDD25/lecture-notes/lect6-7.frm.pdf.