# Distributed Systems: Assignment 2
# Middleware: Object Request Brokers

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

January 14, 2013

## 1 Introduction

In the previous assignment, you implemented a distributed system with a single server and an arbitrary number of clients. The clients were required to explicitly specify the host name and port of the server in order to interact with the database. It was not convenient as this information could easily change. The situation would have become significantly more complicated if there had been many servers joining and leaving the network in an arbitrary way. Such a freedom is at heart of a distributed system, and it is exactly what we aim to achieve by the end of the programming assignments. Therefore, in this assignment, you will implement the code that will allow the participants of your distributed system to smoothly discover each other. At this point, we shift our focus from the client-server to peer-to-peer model [1] and refer to participants as peers.

The problem outlined above is solved by introducing into our distributed system a so-called middleware [2]; carefully read the corresponding sections in the provided reference to get a better understanding of the assignment. Loosely speaking, a middleware of a distributed system is an auxiliary layer that hides all the complexity of the communication process between peers. Such a layer is typically divided into two parts: object services (*e.g.*, name services) and object request brokers.

A name service is a global registrar of all the objects, *i.e.*, peers, that participate in the network. Upon request, a name service can, for instance, provide information about the current location of a particular peer; this scenario is depicted in Figure 1. For our purposes, we assume that the name service is unique and reliable, and its address is fixed and known to everyone. Such a name service is already available for you in the LiU network and can be accessed as we shall describe later on.

An object request broker [2, 3] is a piece of code on a peer's side that interacts with a name service and creates the illusion for the peer that other peers—which might be running on different machines or different processes on the same machine—are running in separate threads of this peer. In other words, it creates an impression that everything is local to the peer, and there is no need of any network communications, which is very convenient.

Your task is to learn how to interact with the provided name service. This will help you in the future to seamlessly incorporate this auxiliary component

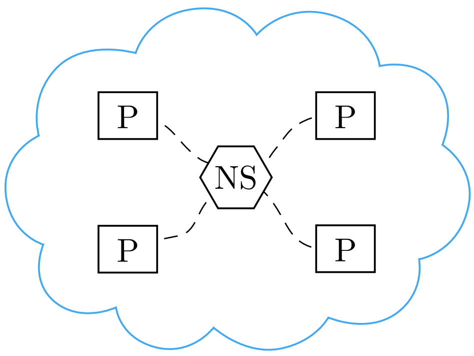

Figure 1: A number of peers (P), equipped with object request brokers, discovering each other through a name service (NS).

into your distributed database of fortunes. To this end, for this assignment, you will put aside your previous code and implement a simple application with the only purpose of registering in the name service and listing other peers.

## 2 Your Task

### 2.1 Preparation

Download [4] and extract the source code for the assignment. In the archive, you will find the following files:

- `lab2/peer.py` — the `Peer` application (no changes are needed);
- `lab2/test.sh` — a shell script that you can use for testing;
- `modules/Common/nameServiceLocation.py` — the file that contains the address of the name service available in the network (should be kept unchanged);
- `modules/Common/objectType.py` — the file that defines your unique identifier of your peers registered within the name service as explained below (should be modified);
- `modules/Common/orb.py` — the code of your object request broker (should be modified);
- `modules/Common/wrap.sh` — the same as for Assignment 1.

Read and understand the code; in particular, you should be able to identify three methods that are involved in the interaction with the name server.

Since there is only one name service for all the students, several students can interfere with each other. To prevent this, the name service was designed in such a way that each student is able to work within a separate name space. `objectType.py` defines the name of your space; you should change it to something unique, *e.g.*, your student identifier. You can also experiment with other groups of students (do not forget to tell them) joining several networks together by choosing the same name in `objectType.py`; otherwise, keep it secret.

Try to run `peer.py` in parallel using several terminal windows. Recall that you are also provided with a test script, `lab2/test.sh`. The expected output, after a successful completion of the assignment, is the list of currently registered peers of your type (`objectType.py`) in the name service. However, as you see, the application cannot even start without your help.

### 2.2 Implementation

Now, you are ready to complete the implementation of the object request broker in the module `orb.py`. The module consists of three major classes:

- `Stub` — the class that represents the image of a remote object on the local machine (therefore, it is also called a proxy); the main purpose is the remote method invocation [1].
- `Skeleton` — the class used to listen to incoming connections and forward them to the main object, `Peer`.
- `Peer` — the class that implements basic bidirectional communications using `Stub` and `Skeleton`; any object wishing to transparently interact with remote objects should extend this class, which is done in `peer.py`.

As before, you are supposed to implement the functions marked with "Your code here". Pay attention to the guidelines given in the code. After you have done this, run several instances of `peer.py` again and observe that each new instance outputs more and more peers currently registered in the name service, *i.e.*, the first one will see only itself, the second one will see itself and the previous one, and so on.

## 3 Conclusion

Having completed the assignment, you have made an important step towards your distributed database of fortunes: now you have an object request broker that can properly interact with the name service. However, the simple listing of available peers achieved so far is not something extremely useful by itself. Moreover, as you saw, the earlier joined peers could not see newcomers (recall that the first peer could see only itself, *etc.*). On the other hand, if some peer left the system, the machines that initially registered it (*i.e.*, the ones that were started after the leaving peer) would not notice this change and, thus, would be mislead. Therefore, in the next assignment, you will (a) extend the application constructed here to call some useful functions of other peers and (b) implement an auxiliary class to properly maintain the peer list on a peer's side.

## References

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

[2] `http://www.ida.liu.se/~TDDD25/lecture-notes/lect4.frm.pdf`.

[3] `http://en.wikipedia.org/wiki/Object_request_broker`.

[4] http://www.ida.liu.se/~TDDD25/labs/assignment2.zip.