UiO : Universitetet i Oslo

IN5020 (Distributed Systems)

First Assignment - Tutorial

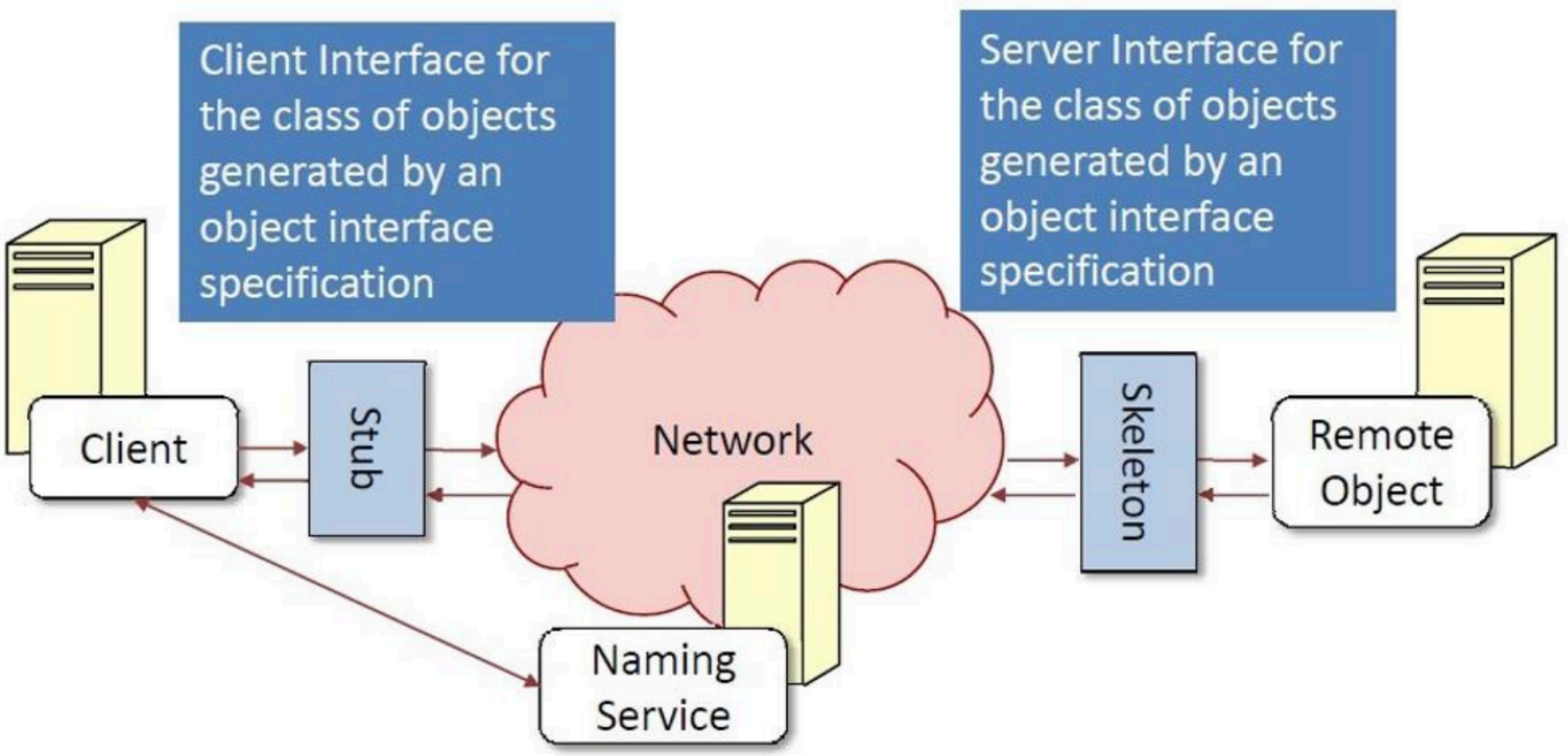
Mohammad H. Tabatabaei
(mohammht@ifi.uio.no)

13.09.2019

Requirements for the first programming assignment

- Your knowledge about distributed objects and RMI
- Eclipse IDE (or your favorite development environment)
- Java SDK (JDK 8)
  - You need the **idlj** and **orbd** tools located at the bin folder

Architecture for Distributed Object Systems



CORBA

- **C**ommon **O**bject **R**equest **B**roker **A**rchitecture
- Offers mechanisms that allow objects to invoke remote methods and receive responses in a transparent way.
  - **Location** transparency
  - **Access** transparency
- The core of the architecture is the **O**bject **R**equest **B**roker
- Specification developed by members of the **O**bject **M**anagement **G**roup (www.omg.org)

CORBA Java Binding

- 1st step: Define the IDL for the remote methods:

```
module HelloApp {
        interface Hello {
                string sayHello(in string message);
        };
};
```

- Save the IDL as a Hello.idl file

CORBA Java Binding

- 2nd step: Compile the interface using the IDL compiler for Java (IDLJ)

  idlj -fall -td <dir> Hello.idl

- -fall: Create client and server code (stub and skeleton)
- -td <dir>: use <dir> for the output directory instead of the current directory

CORBA Java Binding

- **HelloPOA.java**
  - Abstract class of the stream-based server skeleton
- **Hello.java**
  - Interface containing the Java version of the IDL interface
- **HelloOperations.java**
  - Interface containing the method sayHello()
- **_HelloStub**
  - Class of the client stub
- **HelloHelper**
  - Provides auxiliary functionality, such as the narrow() method
- **HelloHolder**
  - Delegates to the methods in the Helper class for reading and writing

CORBA Java Binding

- 3rd step: Implement the Servant class that must extend the POA generated class.

The Servant extends the basic class that handles remote invocations.

```
public class HelloServant  extends HelloPOA {
    public String sayHello(String message) {
            Calendar cal = Calendar.getInstance();
            SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");
            String now = sdf.format(cal.getTime());
            System.out.println("Message from client: " + message);
            return "Hello from Server at " + now;
    }
}
```

CORBA Java Binding

- 4th step: Implement the Server(1/2)

```
public class HelloServer {

public static void main(String[] args) {

        try{

ORB orb = ORB.init(args, null);

POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
rootpoa.the_POAManager().activate();

HelloServant helloImpl = new HelloServant();
org.omg.CORBA.Object ref = rootpoa.servant_to_reference(helloImpl);
Hello href = HelloHelper.narrow(ref);

org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
```

Create and initialize the CORBA ORB

Get reference to the root POA and activate the POA manager

Get object reference from the servant

Get the root naming context

CORBA Java Binding

- 4th step: Implement the Server(2/2)

```
            String name = "Hello";
            NameComponent path[] = ncRef.to_name( name );
            ncRef.rebind(path, href);

            orb.run();

        } catch(Exception e) {
                System.err.println("ERROR: " + e.getMessage());
                e.printStackTrace(System.out);
        }

    }
}
```

Binding the object reference in naming service

Wait for remote invocations

Handle any resulting exception

CORBA Java Binding

- 5th step: Implement the Client(1/2)

```
public class HelloClient {

public static void main(String[] args) {

        try{
            ORB orb = ORB.init(args, null);

            org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
            NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

            String name = "Hello";
            Hello helloRef = HelloHelper.narrow(ncRef.resolve_str(name));
```

Create and initialize the CORBA ORB

Get the root naming context

Resolve the object reference in naming service

CORBA Java Binding

- 5th step: Implement the Client(2/2)

```
            Calendar cal = Calendar.getInstance();
            SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");
            String now = sdf.format(cal.getTime());

            String message = helloRef.sayHello("Hello from Client at " + now);

            System.out.println(message);

        } catch(Exception e) {
                System.out.println("HelloClient Error: " + e.getMessage());
                        e.printStackTrace(System.out);
        }
    }
}
```

Invoke the remote method using the reference (stub)

Handle any result exception

CORBA Java Binding

- To run the application
  - Start the ORB daemon (from the command line)
    - **orbd -ORBInitialPort <port>&**
    - Windows: **start orbd -ORBInitialPort <port>**

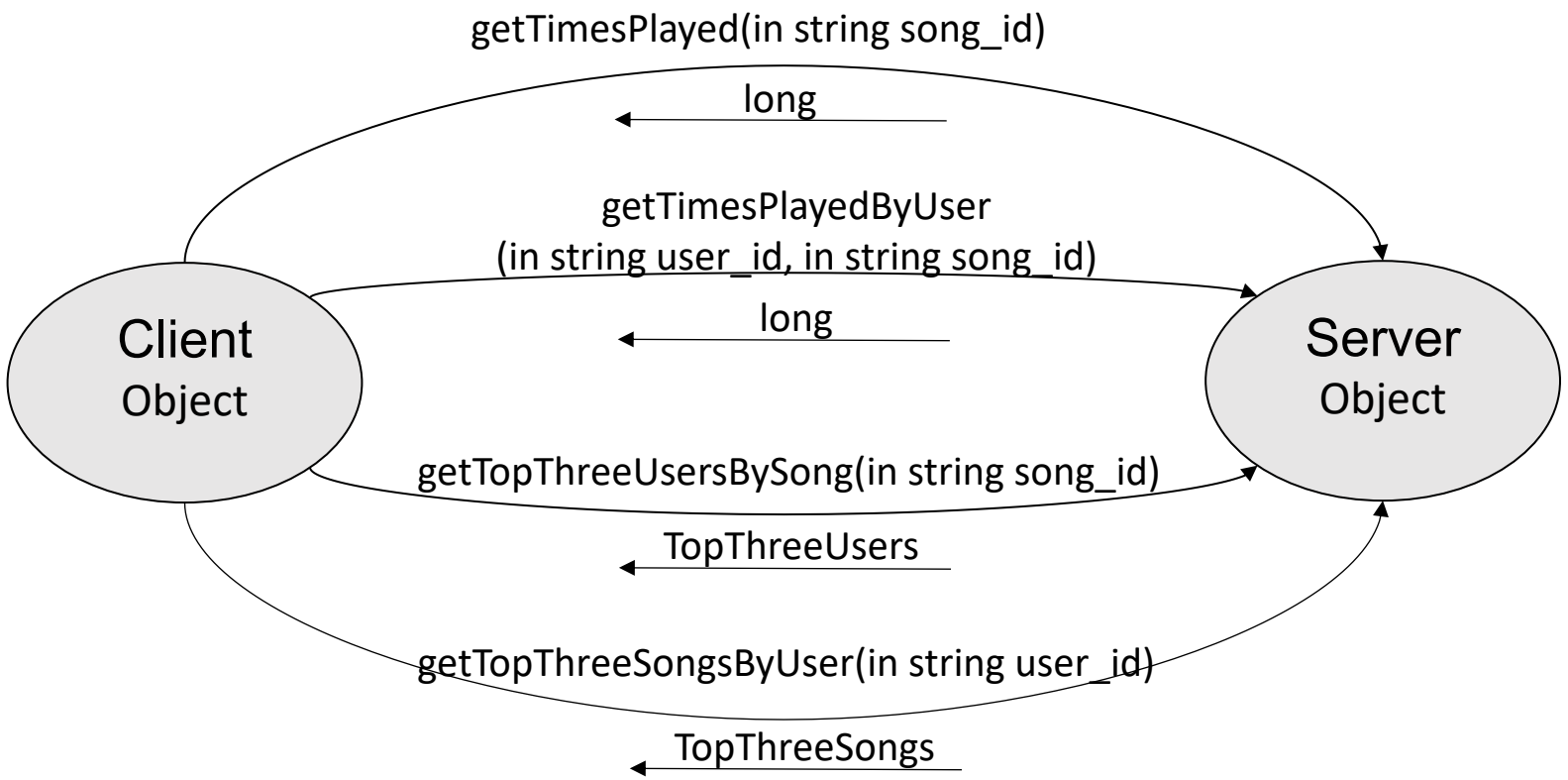
- https://docs.oracle.com/javase/7/docs/technotes/tools/share/orbd.html

CORBA Java Binding

- From Eclipse
  - Right click HelloServer.java -> Run as -> Run configurations . .
  - Open **Java Application** in the left pane. If **HelloServer** is not there, then double click **Java Application**.
  - In the box to the right, open the **(x)= Arguments-tab**, and write **-ORBInitialPort <PORT>** in the **Program Arguments**: text-box.
  - Click **Apply** and then **Run** to start the server.
- Client:
  - Same as for the server
- <PORT>:
  - The port you're running your ORB-daemon on

First Programming Assignment

- CORBA Musical Taste Profile Service



First Programming Assignment

- The server has access to two song profiles data sets:
  - Each file has about 24.000.000 entries in the format:

    <Song ID> <User ID> <Play count>
  - The server cannot keep all the entries in memory!
- The client will invoke the remote methods on the server and print the results.
  - To the standard output and to a file. Example:
    - Song SOJCPIH12A8C141954 played 11205 times. (81 ms)
    - Song SONKFWL12A6D4F93FE played 2 times by user b64cdd1a0bd907e5e00b39e345194768e330d652. (82 ms)

First Programming Assignment

- Devise caching strategies to keep popular information in memory:
  - Popular users are the most active ones (played highest number of songs).
    - **You can keep at most 1000 user profiles in memory!** (around 30 MB)
  - Keeping a separate cache for the getTimesPlayed() and getTopThreeUsersBySong() method is acceptable.
    - 400.000 entries of <String, **SongProfile**> (around 40MB)

- Clients are expected to follow a particular behavior:
  - Most probably query popular users and songs.
  - Most probably perform queries about the same user with consecutive method invocations.

First Programming Assignment

- You have to implement the server and client code according to the specification:
  - Generate the stub and skeleton code
  - Implement the servant
  - Implement the client
  - Run the application and produce output file

First Programming Assignment

- Remain time after theoretical exercises of next meeting is reserved for assistance with the first assignment.
- Questions outside the meeting should be sent to:
  - mohammht@ifi.uio.no

- **Deadline: September 27, 23:59.**

Thanks!

Any questions?

You can find me at:

mohammht@ifi.uio.no