First Assignment – Tutorial lecture

INF5040 (Open Distributed Systems)

Dat Tien Le (dattl@ifi.uio.no)

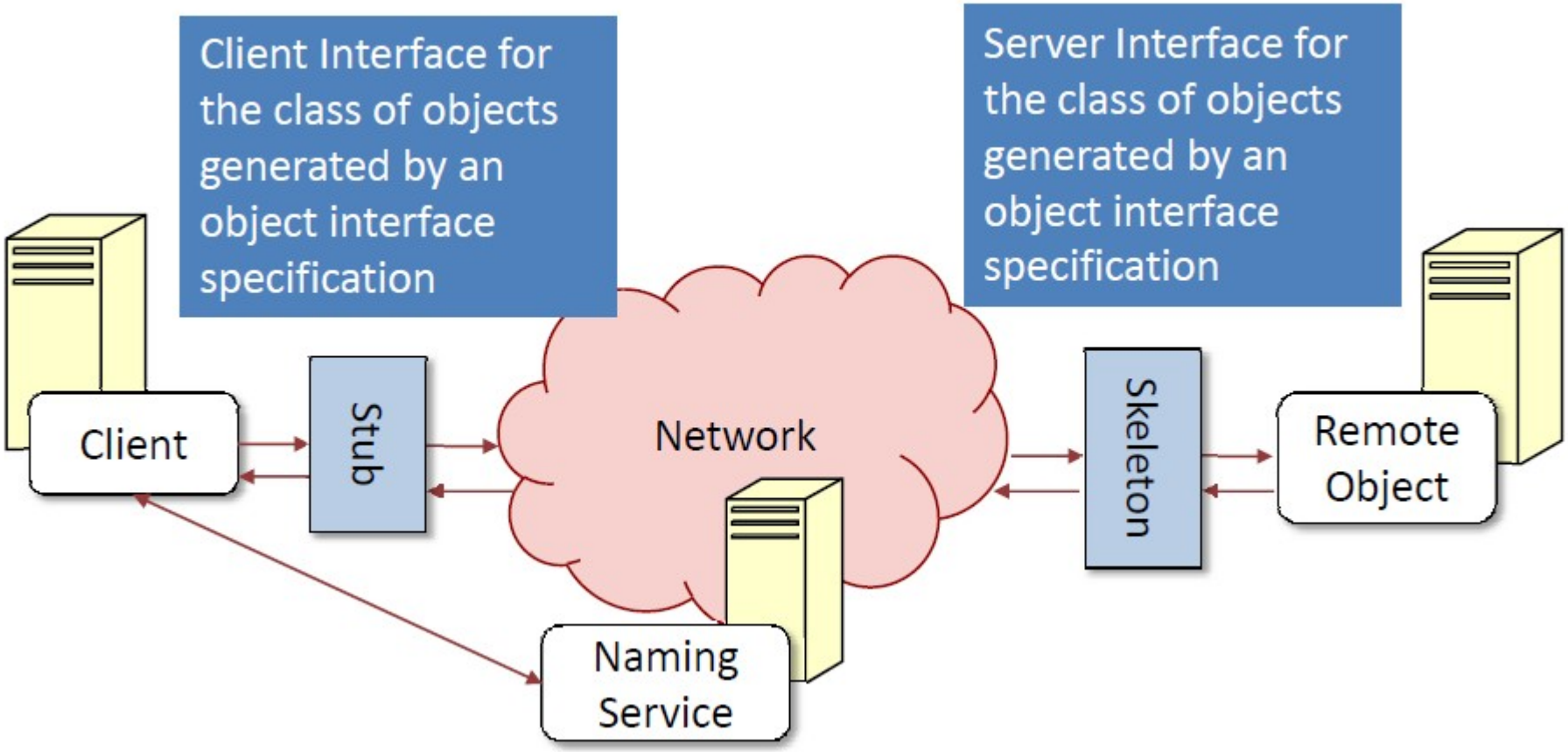
Department of Informatics
University of Oslo

September 18, 2017

What we will be using today

- Your knowledge about distributed objects and RMI
- Eclipse IDE (or your favorite development environment)
- Java SDK latest version
  - 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 Object Request Broker (ORB)
- Specification developed by members of the Object Management Group (www.omg.org)

CORBA Java Binding

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

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

CORBA Java Binding

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

```
idlj -fall -td <toFolder> Hello.idl
```

  - **fall**: Create client and server code (stub and skeleton)

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):

```
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();

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

        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 servant

Get the root naming context

CORBA Java Binding

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

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

Bind the object reference in naming service

```
orb.run();
```

Wait for remote invocations

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

Handle any resulting exception

CORBA Java Binding

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

```
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");
        NamingContext ncRef = NamingContextHelper.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):

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

Invoke the remote method using the reference (stub)

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

System.out.println("Message from Server: " + message);

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

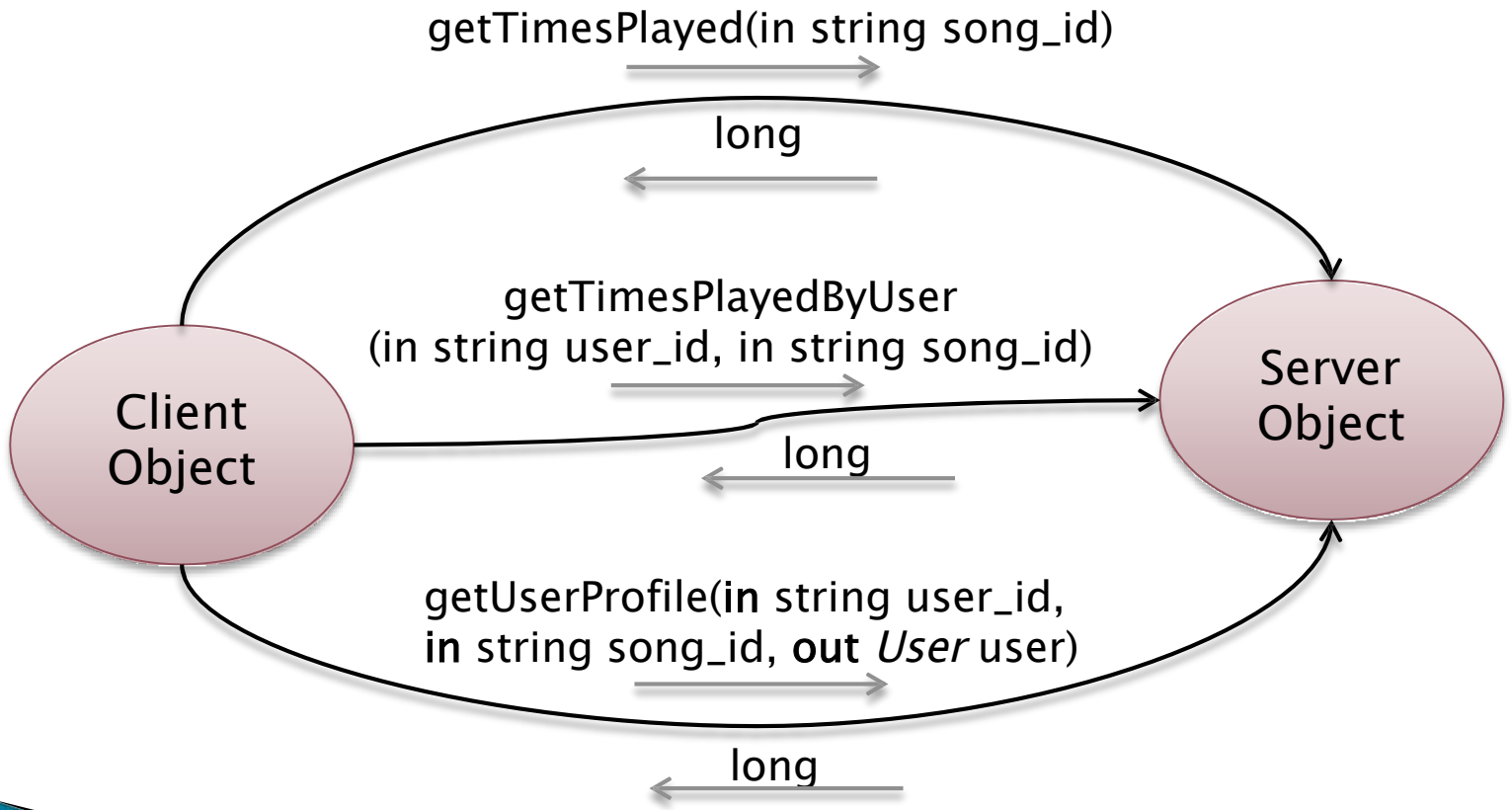
Handle any resulting exception

CORBA Java Binding

- To run the application:
  - Start the ORB daemon with the command:
    - **`orbd -ORBInitialPort <port>`**
  - Run the Server java application:
    - **`java -jar Hello.jar -ORBInitialPort <port> -ORBInitialHost <host>`**
  - Run the Client Java application:
    - Same as server

First Programming Assignment

- CORBA Musical Taste Profile Service



First Programming Assignment

- The server has access to the musical taste profile data set:
  - ~48.000.000 entries in the format:
    - `<User ID>` `<Song ID>` `<Play count>`
  - The server **cannot** keep all the entries in memory!

- Devise caching strategies to keep popular information memory:
  - Popular users are the most active ones (highest number of songs played).
    - **You can keep at most 1000 user profiles in memory!** (around 30MB)
  - Keeping a separate cache for the getTimesPlayed() method is acceptable.
    - 400.000 entries of <String, Integer> (around 10MB)

First Programming Assignment

- 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. (61 ms)`
    - `Song SONKFWL12A6D4F93FE played 2 times by user b64cdd1a0bd907e5e00b39e345194768e330d652. (62 ms)`
- 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

- The IDL file will be made available at the website by the end of this lecture
  - You will have to include the new method and compile it.
- 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 files

First Programming Assignment

- The complete assignment specification and required material can be found at:
  - http://www.uio.no/studier/emner/matnat/ifi/INF5040/h17/timeplan/index.html
- The next meeting is reserved for assistance with the first assignment.
- Questions outside the meeting should be sent to:
  - dattl@ifi.uio.no
- Deadline: **October 9, 23:00**