Lecture 3

Middleware support for remote execution:

- Remote procedure call
- Remote method invocation

Socket programming

- Proprietary distributed applications were initially programmed with sockets (IP + port number).
- While the host name of the server can be resolved by DNS, the port number should have been wired in the client code - this approach lacks flexibility.

- By raising the level of abstraction, the problem can be solved.
- The concept: distribution is hidden and the call to the server can be dealt with as a call to a local procedure.
- Practically, the communication between the client and the server takes place on the network.

Features of remote execution

- An application can be distributed and part(s) of the code will be executed on remote computer(s).
- Remote code (procedure) is called the same way as if it would be running on the same host.
- *The code distribution is transparent*.

- Benefits:
  - better use of resources;
  - load balancing;
  - depending on the network characteristics, the execution time can be shorter.

- Problems:
  - space of addressing;
  - heterogeneity of computing platforms;
  - networks can introduce errors, delays.

Implementation

- The client-server model is used.
- The client (caller) first sends a call message to the server and then waits for a reply message. Messages will include parameters and results, respectively. The client will extract the results and resume execution.
- The server is awaiting the arrival of a call message. When one arrives, the server process extracts the procedure's parameters, computes the results, sends a reply message, and then awaits the next call message.
- Several concurrent implementations are possible, e.g., call is synchronous vs. asynchronous, multi-thread server, error handling.

I. Remote Procedure Call (RPC)

- 1976: idea of remote execution. (RFC 707)
- Most popular was Sun's RPC (ONC RPC) used as the basis of NFS (Network File System) (RFC 1831).
- Provides the means for calling procedures that are hosted on remote computers.
- *It is a network abstraction that gives the impression that one calls local procedures.*
- The RPC system bundles the parameters, ship them off to another computer, passes them to a server running there, takes the results, packages them and ships them back to the caller.
- RPC relieves the programmer of the task of implementing protocols for packaging/unpacking, sending/receiving parameters.
- RPC does not provide any means for reliability. This should be implemented by the application (or not if TCP is used).

RPC Registration

- A port mapper running on a well-known port number registers procedures/services running on the remote computer by mapping them to ports.

- When an application requests a service, it will address the *port mapper*; the port mapper will return the port number of the service being sought.

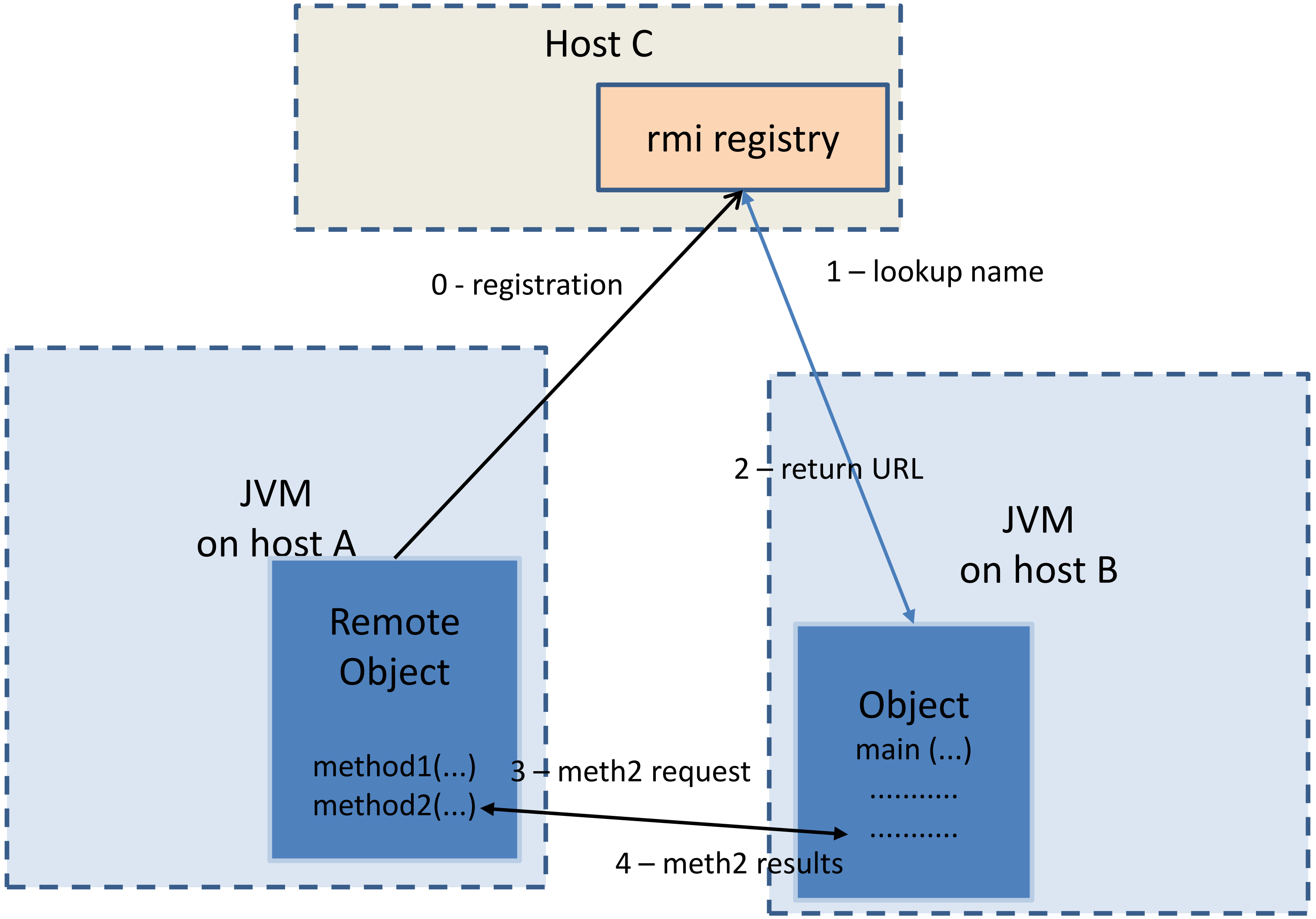
- Tools: interface description language (IDL), external data representation (XDR), compiler.

II. Remote Method Invocation

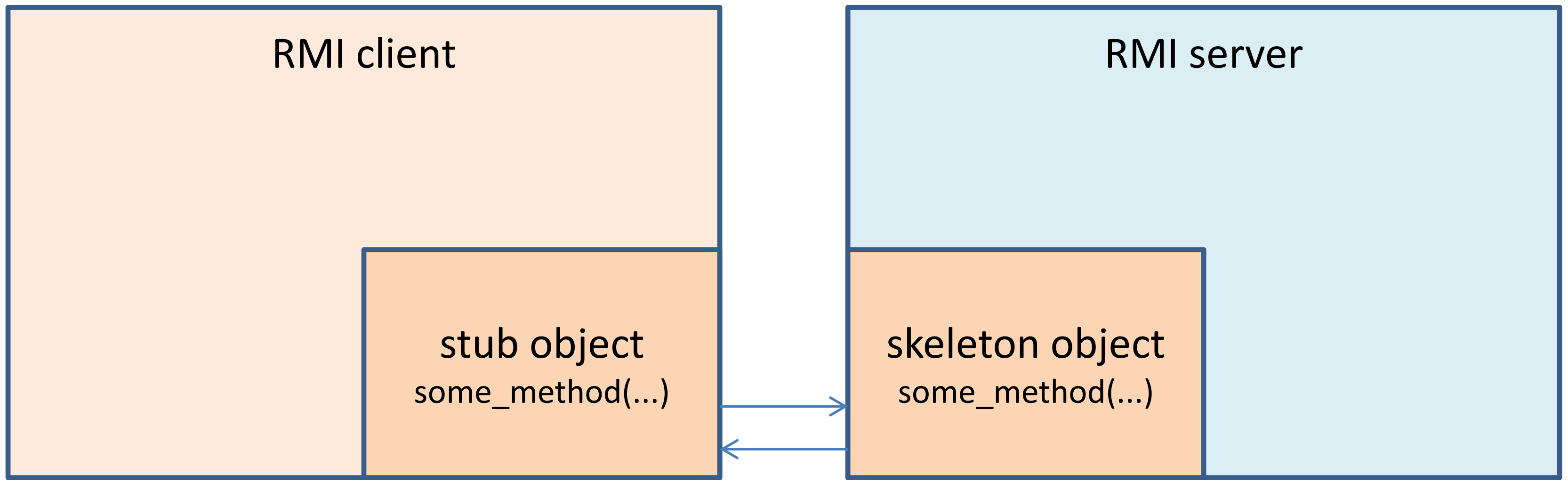
- Environment:
  - the platform is JVM;
  - client/server architecture;
  - servers register their RMI services with a lookup service;
  - clients lookup the registry for a remote object reference.

- Implementation:
  - as easy as local execution;
  - the service is located by URL returned by the lookup process;
  - rmi registry returns URLs like: *//hostname:port/service_name*.

RMI computing model



The RMI system at work



The stub packages the parameters (data marshalling)…
The skeleton does the opposite (data un-marshalling).

RMI components

- Every RMI service is defined by an interface that describes the methods that can be invoked remotely.
- The stub object implements a particular RMI interface, on the client side.
- The skeleton object, on the server side, listens for incoming RMI requests and passes them to the RMI service.
- The implementation object will be called by the skeleton which invokes the appropriate method and, later, will pass the results back to the stub.
- The stub and the skeleton use TCP sockets for communication.

The RMI service interface

- Defines the methods that can be invoked remotely, specific parameters, return types and exceptions.
- The stub, the skeleton and the RMI service implement this interface.

- RMI service interfaces extend the

  java.rmi.Remote interface.

- Only methods defined by the interface can be executed remotely, other methods of the object are hidden from RMI clients.

Example: lightbulb control

Part of the IoT, electric bulbs can be addressed and controlled individually.

```
public interface RMILightBulb extends java.rmi.Remote
{
        public void on () throws java.rmi.RemoteException;
        public void off () throws java.rmi.RemoteException;
        public boolean isOn () throws java.rmi.RemoteException;
}
```

```
public class RMILightBulbImpl
          extends java.rmi.server.UnicastRemoteObject
          implements RMILightBulb
{
          public RMILightBulbImpl() throws java.rmi.RemoteException
          {
                    setBulb(false);
          }
          private boolean lightOn;
          public void on() throws java.rmi.RemoteException
          {
                    setBulb (true);
          }
          public void off() throws java.rmi.RemoteException
          {
                    setBulb (false);
          }
          public boolean isOn() throws java.rmi.RemoteException
          {
                    return getBulb();
          }
```

Contd.

```
// Locally accessible "setBulb" method, changes state of bulb
        public void setBulb (boolean value)
        {
                lightOn = value;
        }

        // Locally accessible "getBulb" method, returns state of bulb
        public boolean getBulb ()
        {
                return lightOn;
        }
}
```

Stub and Skeleton classes

- The rmic tool (JDK) creates the stub and skeleton classes, based on the interface and implementation.
- After compilation of the interface and implementation, the rmic will be invoked:

```
rmic implementation
```

- Two files will be produced:
  - Implementation_Stub.class
  - Implementation_Skeleton.class

The RMI server

```
import java.rmi.*;
import java.rmi.server.*;
public class LightBulbServer
{
	public static void main(String args[])
	{
		System.out.println ("Loading RMI service");
		try
		{
			// Load the service
			RMILightBulbImpl bulbService = new RMILightBulbImpl();
			// Examine the service, to see where it is stored
			RemoteRef location = bulbService.getRef();
			System.out.println (location.remoteToString());
			// Check to see if a registry was specified
			String registry = "localhost";
			if (args.length >=1)
			{
				registry = args[0];
			}
```

Cntd.

```
                    // Registration format //registry_hostname (optional):port /service
                    String registration = "rmi://" + registry + "/RMILightBulb";
                    // Register with service so that clients can find us
                    Naming.rebind( registration, bulbService );


}

                    catch (RemoteException re)
                    {
                              System.err.println ("Remote Error - " + re);
                    }
                    catch (Exception e)
                    {
                              System.err.println ("Error - " + e);
                    }
          }
}
```

The RMI client

```
import java.rmi.*;
public class LightBulbClient
{
        public static void main(String args[])
        {
                System.out.println ("Looking for light bulb service");
                try
                {
                        String registry = "localhost";
                        if (args.length >=1)
                        {
                                registry = args[0];
                        }
                        String registration = "rmi://" + registry + "/RMILightBulb";
                        Remote remoteService = Naming.lookup ( registration );
                        RMILightBulb bulbService = (RMILightBulb) remoteService;
                        System.out.println ("Invoking bulbservice.on()");
                        bulbService.on();
                        System.out.println ("Bulb state : " + bulbService.isOn()  );
```

```
                                                    System.out.println ("Invoking bulbservice.off()");
                              bulbService.off();
                              System.out.println ("Bulb state : " + bulbService.isOn() );
                    }
catch (NotBoundException nbe)
                    {
                              System.out.println ("No light bulb service available  in registry!");
                    }
                    catch (RemoteException re)
                    {
                              System.out.println ("RMI Error - " + re);
                    }
                    catch (Exception e)
                    {
                              System.out.println ("Error - " + e);
                    }
          }
}
```

References

- ONC RPC: http://tools.ietf.org/html/rfc1831
- http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp
- The lightbulb example in this lecture is from “Java Network Programming and Distributed Computing” by David Reilly and M. Reilly, Addison-Wesley, 2002.