CprE 450/550x
Distributed Systems and Middleware

Inter-process Communication

Yong Guan
3216 Coover
Tel: (515) 294-8378
Email: guan@ee.iastate.edu
February 24, 2004

Some info

- Microsoft Tech Talk – Visual Studio.NET Walkthrough
  - **Where:** **Coover 2222**
  - **When:** **Thursday, February 26, 2004**
  - **Time:** **5pm to 6pm**

Readings for Today's Lecture

- References
  - **Chapter 2 of "Distributed Systems: Principles and Paradigms"**
  - **Chapter 11 of "Java Network Programming and Distributed Systems"**
  - **Java.RMI The Remote Method Invocation Guide**

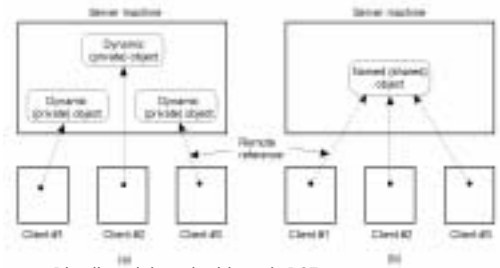
The DCE Distributed-Object Model

- DCE is an example of a distributed system that may have been in the right place but at the wrong time:
  - DCE is RPC-based
  - DCE did not have objects
- DCE people argued with advocates of object technology that they did support objects:
  - All implementation and distribution aspects are hidden behind interfaces
- DCE support distributed objects explicitly.
  - DCE objects form a direct refinement of the RPC-based client-server model.
  - IDL Extension and C++ language bindings
  - Distributed objects take the form of remote objects whose implementation is at a server.
  - A server is responsible for creating C++ objects locally and making methods accessible to remote clients.

The DCE Distributed-Object Model

- Two type of distributed objects supported:
- Distributed dynamic object: An object that a server creates locally on behalf of a client, and is accessible to that client.
  - To create an object, a client issues a request at the server. Each class of dynamic objects has an associated create procedure that can be called using a standard RPC.
  - After creating a dynamic object, the DCE runtime system administrates the new object.
- Distributed named objects: Not intended to be associated with only a single client but are created by a server to have it shared by several clients.
  - Named objects are registered with a directory service so that a client can look up the object and subsequently bind to it.
  - Registration yields that a unique identifier for that object is stored and the contact information.

The DCE Distributed-Object Model



a) Distributed dynamic objects in DCE.
b) Distributed named objects

7

The DCE Distributed-Object Model

- DCE Remote Object Invocation:
  - Each remote object invocation in DCE is done via RPC.
  - Client passes object identifier, the identifier of the interfaces that contains the method, and parameters to the server
  - Server maintains an object table from which it can derive which object is to be invoked if given the object and interface identifiers, and then dispatch the requested method with its parameters.
  - Objects can be put in main memory and secondary storage
- DCE does not support transparent object references. But a client can use binding handle associated with a named object.
  - Binding handle contains an identification of an interface of the object, transport protocol, server's host address and endpoint.
- Lacking a systemwide object reference makes parameter passing harder.
  - Passing objects in RPC means that objects need be explicitly marshaled to be passed by value.

8

Jave RMI

- RMI: A Java technology that allows one JVM to communicate with another JVM and have it execute an object method.
- RPC and RMI
  - RPC supports multiple languages, whereas RMI only support Java
  - RMI deals with objects, but RPC does not support the notion of objects
  - RPC offers procedures (not associated with a particular object)
- Java support remote objects as the only form of distributed objects: The state always resides on a single machine, but the interfaces can be accessed from remote processes.

9

Jave RMI

- A few differences between remote and local objects:
  - **Cloning local and remote objects**
    - Cloning local objects results in a new object of the same type and with exactly the same state. Exact copy of the object being cloned
    - What about cloning a remote object?
      - Not only clone the actual object at its server but also the proxy at each client currently bound to it?
      - In reality, exact copy at the server's address space. Proxies are not cloned. If a client wants access to the cloned object at the server, it need bind to that object again.
  - **Blocking (synchronization): If two processes simultaneously call a synchronized method, only one of the processes will proceed while the other will be blocked.**
    - Where?
      - Client-sub? Synchronize different clients at different machines?
      - Blocking only at the server? What about client failure?
      - Explicit distributed locking schemes needed.

10

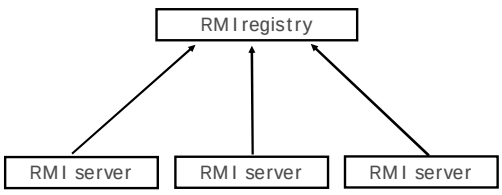
More on Java RMI

- At the language level, Java hides most of the differences during a remote method invocation.
- Any primitive or object type can be passed as a parameter to an RMI, provided that the type can be marshalled (Serializable). But some platform-dependent objects such as file descriptors and sockets, cannot be serialized.
- Local objects are passed by value whereas remote objects are passed by reference (a reference to the object is passed as parameter instead of a copy of the object).
- Reference to a remote object: network address and local identifier of the actual object in the server's address space.
- A remote object is built from two classes:
  - Server class: implementation of server-side code containing object's state and method implementation on that state
  - Client class: implementation of client-side code containing a proxy

11

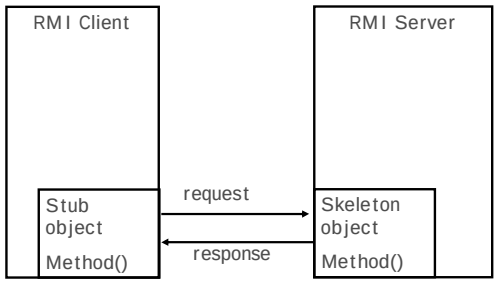
How RMI works

- The format used by RMI for representing a remote object reference: rmi://hostname:port/servicename



12

How RMI works



13

Define a RMI Service Interface

```
Public interface RMILightBulb extends java.rmi.Remote
{
   Public void on() throws java.rmi.remoteexecution;
   Public void off() throws java.rmi.remoteexecution;
   Public boolean ison() throws java.rmi.remoteexecution;
}
```

14

Implement a RMI Service Interface

```
Public class RMILightBulbImpl
      extends java.rmi.server.UnicastRemoteObject
      implements RMILightBulb
{
      Public RMILightBulbImpl () throws java.rmi.remoteexecution
      {setBulb(false);}

      Private boolean lighton;

      Public void on() throws java.rmi.remoteexecution
      {       setBulb(true); }

      Public void off() throws java.rmi.remoteexecution
      {       setBulb(false); }

      Public boolean ison() throws java.rmi.remoteexecution
      {return getBulb();}

      Public void setBulb(boolean value)
      {lighton = value;}

      Public void getBulb()
      {return lighton;}

}
```

15

Create Stub and Skeleton Classes

Rmic RMILightBulbImpl

Two files would be produced:

- **RMILightBulbImpl_Stub.class**
- **RMILightBulbImpl_Skeleton.class**

16

Create a RMI Server

```
import java.rmi.*;
Import java.rmi.server.*;

Public class LightBulbServer
{
      Public static void main(String args[])
      {
            Try{
                  RMILightBulbImpl bulbService=new RMILightBulbImpl();
                  RemoteRef location = bulbService.getRef();

                  String registry = args[0];

                  String registration = "rmi://"+registry+"/RMILightBulb";

                  Naming.rebind(registration, bulbService);
            }
      }
}
```

17

Create a RMI Client

```
import java.rmi.*;

Public class LightBulbClient
{
      Public static void main(String args[])
      {
            Try{
                  String registry = args[0];

                  String registration = "rmi://"+registry+"/RMILightBulb";

                  Remote remoteService = Naming.lookup(registration);

                  bulbService.on();
                  system.out.println(bulbService.isOn());

                  bulbService.off();
                  system.out.println(bulbService.isOn());


            }
      }
}
```

18

Running the RMI system

- You need to add /usr/local/java/bin/ to your path.
- Copy all necessary files to a directory on the local file system of all clients and the server.
- Change to the directory where the files are located, and run rmiregistry.
- Creating Stub and Skeleton classes: *rmic RMILightBulbIml*
- In a separate console window, run the server with a hostname where rmiregistry is running.

  Java LightBulbServer hostname
- In a separate console window (another machine), run the client with a hostname where rmiregistry is running.

  Java LightBulbClient hostname

19

Any Questions?

See you next time.