

Distribution Infrastructures

Chapter 5

Object Model

- ◆ Object References (OR)
- ◆ Interfaces
- ◆ Actions
- ◆ Exceptions
- ◆ Garbage Collection

Object References

- ◆ Objects are accessed via object references.
- ◆ Methods are invoked on a target object by providing object reference, method name and arguments.
- ◆ Methods are invoked on the receiver or target object.
- ◆ OR are first class values meaning that they can be assigned, passed as arguments, and returned as results.

Interfaces

- ◆ An interface provides a definition of the signatures of a set of methods without specifying their implementation.
- ◆ Interface can also define new types for parameters and return values.
- ◆ Interfaces do NOT have constructors.
- ◆ An object may implement many interfaces.
- ◆ An interface may be implemented many ways.

Actions

- ◆ Actions in an object-oriented program is initiated by an object invoking a method in another object.
- ◆ An invocation can have two effects:
 - The state of the receiver may change
 - Further invocations may take place
 - If no exceptions occur control will eventually return to the caller.

Exceptions

- ◆ Exceptions provide a clean way to deal with error conditions.
- ◆ Each method heading explicitly lists as exceptions the error conditions it may encounter.
- ◆ A block of code is defined to throw an exception on error and catch blocks to capture and process the errors.
- ◆ Control may not return to the point the exception was thrown.

Garbage Collection

- ◆ Is means of freeing up space occupied by objects which are no longer needed.
- ◆ Java has a built in mechanism to do that.
- ◆ But for development in languages such as C++ it is the responsibility of the programmer to do this.
- ◆ If garbage is collected properly it may lead memory leak and eventual break down of the software system.

9/24/2004

B.Ramamurthy

7

Distributed Object Model

- ◆ Remote Object Reference (ROR)
- ◆ Remote Interface
- ◆ Distributed Actions
- ◆ Distributed Garbage Collection
- ◆ Exceptions (Remote in Java)
- ◆ Look at Java RMI **implementation** as an example.

9/24/2004

B.Ramamurthy

8

RMI Method Invocation Semantics

- ◆ Any request-reply model can be implemented in different ways to provide different delivery guarantees.
 - Retry request message: whether to transmit the request until server receives it.
 - Duplicate filtering: filter duplicate requests to the server.
 - Retransmission of results: Keep a history of requests to enable retransmission of results without re-executing requests.

9/24/2004

B.Ramamurthy

9

Transparency

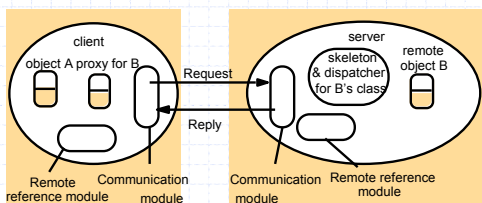
- ◆ Originally RPC (Birell and Nelson) aimed to make RPC no different from a local call.
- ◆ CORBA and RMI do hide marshalling and messaging from the programmer.
- ◆ But latency and partial failure of server cannot be ignored.
- ◆ Waldo (1994) says it cannot be transparent. We should assign a different syntax to remote calls.

9/24/2004

B.Ramamurthy

10

The role of proxy and skeleton in remote method invocation



9/24/2004

B.Ramamurthy

11

Communication Module

- ◆ Two co-operating modules carry out the request-reply protocol. Message structure given in next slide.
- ◆ Communication model is responsible for providing invocation semantics, we discussed earlier.
- ◆ Object references are responsibility of remote object reference module.
- ◆ Method id and marshalling aspects are responsibility the RMI software.

9/24/2004

B.Ramamurthy

12

Request-reply message structure

messageType	<i>int (0=Request, 1= Reply)</i>
requestId	<i>int</i>
objectReference	<i>RemoteObjectRef</i>
methodId	<i>int or Method</i>
arguments	<i>array of bytes</i>

9/24/2004

B.Ramamurthy

13

Remote Reference Module

- ◆ A remote reference module is responsible for translating between local and remote object reference and for creating remote object references.
- ◆ Remote reference module in each process has remote reference table that has
 - An entry all the objects held by the process.
 - An entry for each local proxy

9/24/2004

B.Ramamurthy

14

Actions of the Remote Reference Module

- ◆ "Send": When a remote object is to be passed as argument or result for the first time, ROR is created and added to the table.
- ◆ "Receive": When a ROR arrives in a request or reply, Remote Reference module has to search its table for a local object reference which may be a proxy or a remote object. If it is not there a proxy is created and added to the table.
- ◆ Basically maintain a table of ROR and proxies.

9/24/2004

B.Ramamurthy

15

RMI Software

- ◆ Between application-level objects and remote reference module and communication module.
- ◆ It contains the middleware objects: proxy, dispatcher, and skeleton.
- ◆ We will study the roles of these classes next:

9/24/2004

B.Ramamurthy

16

RMI Software: Proxy

- ◆ Role of proxy is to make remote invocation transparent to the clients.
- ◆ There is one proxy for each remote object for which a process holds remote object reference.
- ◆ Class of a proxy implements the methods in the remote interface quite differently.
- ◆ For each method proxy marshals a reference to the target object, its own methodId and its arguments into a request message, and sends it to the target, awaits the reply message, unmarshals it and return the result to the invoker.

9/24/2004

B.Ramamurthy

17

RMI Software: Dispatcher, Skeleton

- ◆ A server has one dispatcher and skeleton for each class representing a remote object.
- ◆ It then maps the methodID to the appropriate method in the skeleton.
- ◆ Skeleton unmarshals the arguments and invokes the actual remote object method and marshals the result and or exceptions, if any.

9/24/2004

B.Ramamurthy

18

RMI design and Implementation

- ◆ Remote objects are created by factory methods.
- ◆ A binder is needed by the server to **register** and object and by the server to **lookup** an object.
- ◆ A binder maintains a table containing mappings of textual references to ROR.
- ◆ RMI registry is a binder.

9/24/2004

B.Ramamurthy

19

RMI Naming class

- ◆ RMIRegistry is accessed by the Naming class.
- ```
void rebind(String name, Remote obj)
void bind(String Name, Remote obj)
void unbind(String name, Remote obj)
void lookup(String name)
String[] list() list all the bindings.
```

9/24/2004

B.Ramamurthy

20

## Use of Reflection in RMI

- ◆ Reflection is a process by which an object can be queried to reveal its methods.
- ◆ Reflection is used to pass information in request messages about the method to be invoked.
- ◆ It uses a class called Method defined in java.lang.reflect package.
- ◆ *Method* can be *invoked* on an object of suitable class.

9/24/2004

B.Ramamurthy

21

## Using Method class In RMI

- ◆ Proxy marshals by putting method in Method object, and arguments in an array of Objects.
- ◆ Dispatcher unmarshals (a) the Method object and (b) array of argument objects and directly invokes by using invoke method and the two parameters above.
- ◆ So the dispatcher can be generic and may not need the support of a object-specific skeleton.

9/24/2004

B.Ramamurthy

22

## Assignment

- ◆ Read Section 5.4 Event and Notifications which is another example dealing with distributed infrastructures.

9/24/2004

B.Ramamurthy

23