Algorithms and Programming IV
Implementation of RMI (19-2)

Summer Term 2021 | 18.06.2021
Barry Linnert
The Process of Remote Method Invocation
What does the Communication Module do?

Two cooperating communication modules carry out the request-reply protocol.

The content of request and reply messages:

<table>
<thead>
<tr>
<th>messageType</th>
<th>requestId</th>
<th>remoteReference</th>
</tr>
</thead>
</table>

The communication modules provide together a specified invocation semantics.

The communication module in the server selects the dispatcher for the class of the object to be invoked, passing on the remote object’s local reference.
Responsibilities of Server’s Communication Module

In the context of client-server communication, the responsibilities of the server’s communication module involve handling requests and replies through the following components:

- **Object A**: Represents the client side.
- **Proxy for B**: Handles requests on behalf of Object A.
- **Remote Reference Module**: Stores remote references for objects.
- **Communication Module**: Facilitates the exchange of requests and replies.
- **Remote Object B**: The server-side object.
- **Skeleton & Dispatcher for B’s Class**: Manages the dispatching of requests to the correct method in Remote Object B.
- **Servant**: The actual object performing the request.

The diagram illustrates the flow of data from Object A to Remote Object B via the Proxy and Communication Module, ensuring the correct execution of requests and the return of their replies.

ALP IV: Concepts of Non-Sequential and Distributed Programming | Summer Term 2021
What does the Remote Reference Module do?

It is responsible for translating between local and remote object references and for creating remote object references.

The remote reference module holds a (remote object) table that records the correspondence between local object references in that process and remote object references (which are system-wide).

Table includes

- An entry for all remote objects held by the process
- An entry for each local proxy
Remote Reference Module/Servant

- **client**
  - **object A**
  - **proxy for B**
  - **remote reference module**
  - **communication module**

- **server**
  - **Skeleton & dispatcher for B’s class**
  - **remote object B**
  - **servant**

**Request and Reply**

- **Request** from client to server
- **Reply** from server to client
RMI Software

Client
- object A
- proxy for B
- remote reference module

Server
- remote object B
- Skeleton & dispatcher for B’s class
- servant
- remote reference module
- communication module

Request and Reply
Generation of Classes for Proxies, Dispatcher and Skeleton

Classes for proxies, dispatcher and skeleton are generated automatically by an interface compiler.

Java RMI contains a set of methods offered by a remote object defined as a Java interface that is implemented within the class of the remote object.

Java RMI compiler generates the proxy, dispatcher and skeleton classes from the class remote object.
Dynamic Invocation: An Alternative to Proxies

Dynamic invocation gives the client access to a generic representation of a remote invocation.

To make a dynamic invocation, not only the information about the interface of the remote object are included in the remote object reference (e.g., name), but also the names of the methods and the types of the argument are required.

When is it useful?

- In applications, where some of the interfaces of the remote objects cannot be predicted at design time.
Factory Methods

Servants are created either in the initialization section or in methods in a remote interface designed for that purpose

*Factory method*: used to refer to a method that creates servants

*Factory object*: object with factory methods
Binder

A binder in a distributed system is a separate service that maintains a table containing mappings from textual names to remote object references.

It is used by servers to register their remote objects by name and by clients to look them up.

The Java binder is called RMIregistry.
Activation of Remote Objects

A remote object is described as active when it is available for invocation from a running process, whereas it is called passive if it is not currently active but can be made active.

Activation consists of creating an active object from the corresponding passive object by creating a new instance of its class and initialize its instance variables from the store state.

Activator is responsible for

- Registering passive objects that are available for activation
- Starting named server processes and activating remote objects in them
- Keeping track of the locations of the servers for remote objects that it has already activated
Java Distributed Garbage Collection Algorithm

client

object A

proxy for B

request

reply

communication module

remote reference module

server

Skeleton & dispatcher for B’s class

remote object B

servant

remote reference module
Java Distributed Garbage Collection Algorithm (cont.)

Each server process contains a set of names of the processes that hold remote object references for each of its remote objects.

client

server

addRef(B)
removeRef(B)

b.holder