Remote Invocation

Nicola Dragoni
Embedded Systems Engineering
DTU Informatics

1. Introduction
2. Remote Method Invocation (RMI)
3. RMI Invocation Semantics
From the First Lecture (Architectural Models)...

- The architecture of a system is its structure in terms of separately specified components and their interrelationships.

- 4 fundamental building blocks (and 4 key questions):
  - Communicating entities: what are the entities that are communicating in the distributed system?
  - Communication paradigms: how do these entities communicate, or, more specifically, what communication paradigm is used?
  - Roles and responsibilities: what (potentially changing) roles and responsibilities do these entities have in the overall architecture?
  - Placement: how are these entities mapped on to the physical distributed infrastructure (i.e., what is their placement)?
Communication Paradigms

- 3 types:
  - **direct communication**
  - **interprocess communication**
    - low level support for communication between processes in the distributed system, including message-passing primitives, socket programming, multicast communication
  - **remote invocation**
    - most common communication paradigm, based on a two-way exchange between communicating entities and resulting in the calling of a remote operation (procedure or method)
  - **indirect communication**
    - communication is indirect, through a third entity, allowing a strong degree of decoupling between senders and receivers.
    - Examples: publish subscribe systems, distributed shared memory (DSM).
Remote Invocation

- **RPC (Remote Procedure Call)**
  - the earliest programming model for distributed programming:
    - allows client programs to call procedures in server programs running in separate processes (and generally in different computers from the client).

- **RMI (Remote Method Invocation)**
  - extension of local method invocation of object-oriented programming
    - allows an object living in one process to invoke the methods of an object living in another process. Most famous example: Java RMI (--> tutorial!)
Remote Method Invocation (RMI)
Let Us Start from Scratch: the Object Model (...in 2 slides...)

• An object-oriented program (Java, C++, ...) consists of a collection of interacting objects, each of which consists of a set of data and a set of methods.

• An object can communicate with other objects by invoking their methods, generally passing arguments and receiving results (request/reply protocol).

• Objects can encapsulate their data and the code of their methods.

• Some languages (JAVA, C++) allow programmers to define objects whose instance variables can be accessed directly.

• BUT in a distributed object system, an object’s data should be accessible only via its methods (or interface).
Actions in the Object Model

- An **action** in an object-oriented program is initiated by an object invoking a method in another object.

- The **receiving object** executes the appropriate method and then returns control to the **invoking object**, sometimes supplying a result.

- An **invocation of a method** can have **3 possible effects**:
  - the state of the receiver may be changed
  - a new object may be instantiated (i.e., by using a constructor in Java)
  - further invocations on methods in other objects may take place
How to extend the “traditional” object model to make it applicable to distributed systems?
The Distributed Object Model

- Each process contains a collection of objects
  - some of which can receive both local and remote invocations
  - whereas the other objects can receive only local invocations.

- Method invocations between objects in different processes, whether in the same computer or not, are known as remote method invocations.

- Method invocations between objects in the same process are local method invocations.
Remote Objects

- **Remote objects**: objects that can receive remote invocations.

- **Fundamental concepts** of the distributed object model:
  - [Remote Object References] other objects can invoke the methods of a remote object if they have access to its *remote object reference*.
  - [Remote Interfaces] every remote object has a *remote interface* that specifies which of its methods can be invoked remotely.
Remote Object Reference

• A remote object reference is an identifier that can be used throughout a distributed system to refer to a particular unique remote object.

• A remote object reference is passed in the invocation message to specify which object is to be invoked.

• Remote object references are analogous to local ones in that:

  ▸ the remote object to receive a remote method invocation is specified by the invoker as a remote object reference

  ▸ remote object references may be passed as arguments and results of remote method invocations
Remote Interface

The remote interface specifies which methods of an object can be invoked remotely.

Objects in other processes can invoke only the methods that belong to the remote interface of a remote object.

The class of a remote object implements the methods of its remote interface.

Local objects can invoke the methods in the remote interface as well as other methods implemented by a remote object.
Actions... in a *Distributed* Object System

- **As in the non-distributed case:** an action is initiated by a method invocation, which may result in further invocations on methods in other objects.

- **BUT in the distributed case:** the objects involved in a chain of related invocations may be located in different processes or different computers.

- When an invocation crosses the boundary of a process or computer, RMI is used and the remote reference of the object must be available to the invoker.

- Remote object references may be obtained as the *results of remote method invocations* (example: A might obtain a remote reference to F from B)
Creation of Remote Objects

• When an action leads to the instantiation of a new object, that new object will normally live within the process where the instantiation is requested.

• If a newly instantiated object has a remote interface, it will be a remote object with a remote object reference.
Exceptions

• Any remote invocation may fail for reasons related to the invoked object being in a different process or computer from the invoker.

Example: the process containing the remote object may have crashed or may be too busy to reply, or the invocation or result message may be lost.

• Remote method invocation should be able to raise exceptions!

› Timeouts that are due to distribution

› Exceptions raised during the execution of the method invoked:
  - attempt to read beyond the end of a file
  - attempt to access a file without the correct permissions
  - ...


RMI Invocation Semantics
Local Method Invocation Semantics

- Local method invocations are executed **exactly once**

  *exactly once* invocation semantics = every method is executed exactly once

- *This cannot always be the case for remote method invocation!*

- Request-reply protocols, such as RMI, can be implemented in different ways to provide different **delivery guarantees**.

- These choices lead to a variety of possible **semantics** for the reliability of remote invocations as seen by the invoker.
Main Design Choices for Implementing RMI

- **Retry request message**: whether to retransmit the request message until either a reply is received or the server is assumed to have failed.

**Diagram:**
- Sender
- Receiver
- Request
- Deadlock!

**Error control mechanisms:**
- Timeout + retransmission of request msg
Main Design Choices for Implementing RMI

- **Duplicate filtering:** when retransmissions are used, whether to filter out duplicate requests at the server.

**Error control mechanisms:** numbering scheme
Main Design Choices for Implementing RMI

• Retransmission of results: whether to keep a history of result messages to enable lost results to be retransmitted without re-executing the operations at the server.

Error control mechanisms: numbering scheme + history of result msgs
Main Design Choices for Implementing RMI

• Combination of these choices lead to a variety of possible semantics for the reliability of remote invocations: Maybe, At-least-once, At-most-once.

Retry request message

Duplicate filtering

+/?-?

Retransmission of results
RMI Invocation Semantics: Maybe

• The remote method may be executed once or not at all.

• Maybe semantics arises when no fault tolerance measures are applied.

• Useful only for applications in which occasional failed invocations are acceptable.

• This model can suffer from the following types of failure:
  
  ‣ omission failures if the invocation or result message is lost

  ‣ crash failures when the server containing the remote object fails.
Exercise

• Develop (in your preferred programming language) a client-server system implementing a request-reply protocol compliant with the MAYBE semantics.
RMI Invocation Semantics: At-Least-Once

• The invoker receives either
  ‣ a result, in which case the invoker knows that the method was executed at least once, or
  ‣ an exception informing it that no result was received.

• Can be achieved by the retrasmision of request messages, masking the omission failures of the invocation or result message.

• This model can suffer from the following types of failure:
  ‣ crash failures when the server containing the remote object fails
  ‣ arbitrary failures, in cases when the invocation message is retransmitted, the remote object may receive it and execute the method more than once, possibly causing wrong values to be stored or returned.
Exercise

• Develop (in your preferred programming language) a client-server system implementing a request-reply protocol compliant with the AT-LEAST-ONCE semantics.
RMI Invocation Semantics: At-Most-Once

• The invoker receives either
  ‣ a result, in which case the invoker knows that the method was executed exactly once, or
  ‣ an exception informing it that no result was received, in which case the method will have been executed either once or not at all.

• Can be achieved by using a combination of fault tolerance measures (retransmission + duplicate filtering).
  ‣ The use of retries masks any omission failures of the invocation or result messages.
  ‣ Arbitrary failures are prevented by ensuring that for each RMI a method is never executed more than once.
Exercise

• Develop (in your preferred programming language) a client-server system implementing a request-reply protocol compliant with the AT-MOST-ONCE semantics.
RMI Invocation Semantics Summary

<table>
<thead>
<tr>
<th>Fault tolerance measures</th>
<th>Invocation semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retransmit request message</td>
<td>Duplicate filtering</td>
</tr>
<tr>
<td>No</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

- In Java RMI the invocation semantics is *at-most-once*.

- In CORBA is *at-most-once* but *maybe* semantics can be requested for methods that do not return results.
Remote Procedure Call (RPC)
• **RPC (Remote Procedure Call)**: allows client programs to call procedures in server programs running in separate processes and generally in different computers from the client.
RPC vs RMI?

• A remote procedure call is very similar to a RMI in that a client program calls a procedure in another program running in a server process.

• Server may be clients of other servers to allow chains of RPCs.

• A server process must define in its service interface the procedures that are available for calling remotely.

• RPC, like RMI, may be implemented to have one of the choices of invocation semantics previously discussed (maybe, at-least-one, at-most-one).