REMOTE METHOD INVOCATION (RMI)

Dr. Ali Haider Shamsan

Lecture Outlines

RMI

Review

- RPC

Keywords

Distributed Systems, DS, Remove Invocation, RMI

Remote Method Invocation (RMI)

Intro

- Remote method invocation (RMI) is **closely** related to **RPC** but **extended** into the world of **distributed objects.**
- In RMI, a **calling object** can **invoke** a method in a potentially **remote object**. As with **RPC**, the underlying details are generally hidden from the user.

Remote Method Invocation (RMI)

The commonalities between RMI and RPC

- They **both support** programming **with interfaces**, with the resultant benefits that stem from this approach.
- They are **both** typically **constructed** on **top** of request-reply protocols and can offer a range of call semantics such as **at-least-once and at-most-once**.
- They **both offer** a **similar** level of **transparency** – that is, **local** and **remote** calls employ the **same syntax** but remote interfaces typically expose the distributed nature of the underlying call, for example by supporting remote exceptions.
- The

Design issues for RMI

- As mentioned above, **RMI** shares the **same design** issues as **RPC** in terms of
  - programming with interfaces,
  - call semantics and
  - level of transparency.

Design issues for RMI

The object model •

- An **object-oriented program**, for example in Java or C++, consists of a **collection of interacting** objects, each of which consists of a **set of data** and a set of **methods**.
- An object **communicates** with other objects by **invoking** their **methods**, generally passing **arguments** and receiving **results**.
- Objects can **encapsulate** their data and the code of their methods.
- Some languages, for example Java and C++, **allow** programmers to **define objects** whose instance variables can be **accessed directly**.
- But for use in a **distributed object system**, an object's data should be accessible only via its **methods**.

Design issues for RMI

Object references:

- Objects can be **accessed** via **object references**.
  - For example, in Java, a variable that appears to hold an object actually holds a reference to that object.
- To **invoke a method** in an object, the object reference and method name are given, together with any necessary arguments.
- The object whose **method is invoked** is sometimes called the **target and sometimes the receiver**.
- Object references are **first-class values**, meaning that they may, for example, be **assigned to variables**, passed as **arguments** and returned as **results of methods**.

Design issues for RMI

Interfaces:

- An interface provides a **definition** of the **signatures** of a set of **methods** (that is, the types of their arguments, return values and exceptions) **without** specifying their **implementation**.
- An object will **provide a particular interface** if its class contains code that **implements** the **methods** of that interface.
- In Java, a **class may implement several interfaces**, and the methods of an interface may be **implemented by any class**.
- An interface also **defines types** that can be used to declare the **type of variables** or of the **parameters** and return values of methods. Note that interfaces do not have constructors.

Design issues for RMI

Actions :

- **Action** in an object-oriented program is **initiated** by an object **invoking a method** in another object.
- An invocation can **include additional** information (arguments) needed to carry out the method.
- The **receiver executes** the appropriate method and then **returns control** to the invoking object, sometimes supplying a result.
- An invocation of a method can have three effects:
  - 1. The **state** of the **receiver** may be **changed**.
  - 2. A **new object** may be **instantiated**, for example, by using a constructor in Java or C++.
  - 3. **Further invocations** on methods in other objects may **take place.**

Design issues for RMI

Exceptions :

- Programs can **encounter** many sorts of **errors** and **unexpected conditions** of varying seriousness.
- **During** the **execution** of a method, many **different problems** may be discovered: for example, inconsistent values in the object's variables, or failures in attempts to read or write to files or network sockets.

Design issues for RMI

Garbage collection:

- It is necessary to **provide a means** of **freeing** the space occupied by objects when they are **no longer needed**.
- A **language** such as **Java**, that can detect automatically when an object is **no longer accessible** recovers the space and makes it available for allocation to other objects.
- This process is called **garbage collection**. When a language (for example, C++) does not support garbage collection, the programmer has to cope with the freeing of space allocated to objects. **This can be a major source of errors.**

Design issues for RMI

Distributed objects •

- The **state of an object** consists of the values of its instance variables.
- In the object-based paradigm the state of a program is **partitioned** into **separate parts**, each of which is associated with an object.
- Since **object-based** programs are logically partitioned, the physical distribution of objects into **different processes** or **computers** in a distributed system is a natural extension.

Design issues for RMI

Distributed objects •

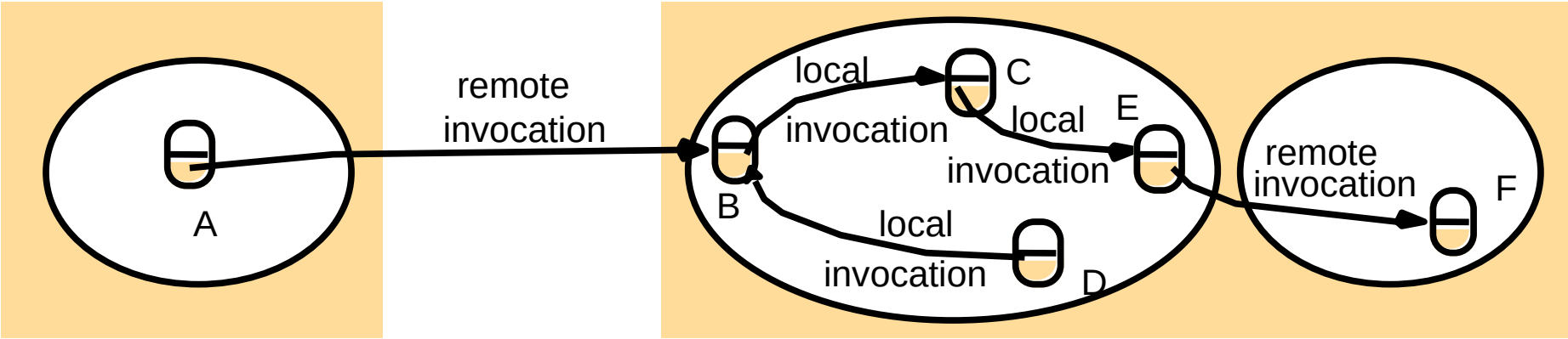
- **Distributed object** systems may adopt the client-server architecture.
- In this case, **objects** are managed by **servers** and their **clients invoke their methods** using remote method invocation.
- In RMI, the client's **request** to invoke a method of an object is **sent** in a **message to the server** managing the object.
- The invocation is **carried out by executing** a method of the object at the server and the result is **returned to the client** in another message.

Design issues for RMI

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, as shown in Figure 5.12
- **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**.

Figure 5.12
Remote and local method invocations



- We refer to objects that can receive remote invocations as remote objects.
- In Figure 5.12, the objects B and F are remote objects.
- All objects can receive local invocations, although they can receive them only from other objects that hold references to them.

Design issues for RMI

The distributed object model •

- The following **two fundamental** concepts are at the heart 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.
  - For example, a remote object reference for B in Figure 5.12 must be available to A.
- **Remote interfaces:** Every remote object has a remote interface that specifies which of its methods can be invoked remotely.
  - For example, the objects B and F in Figure 5.12 must have remote interfaces.

Design issues for RMI

Actions in a distributed object system •

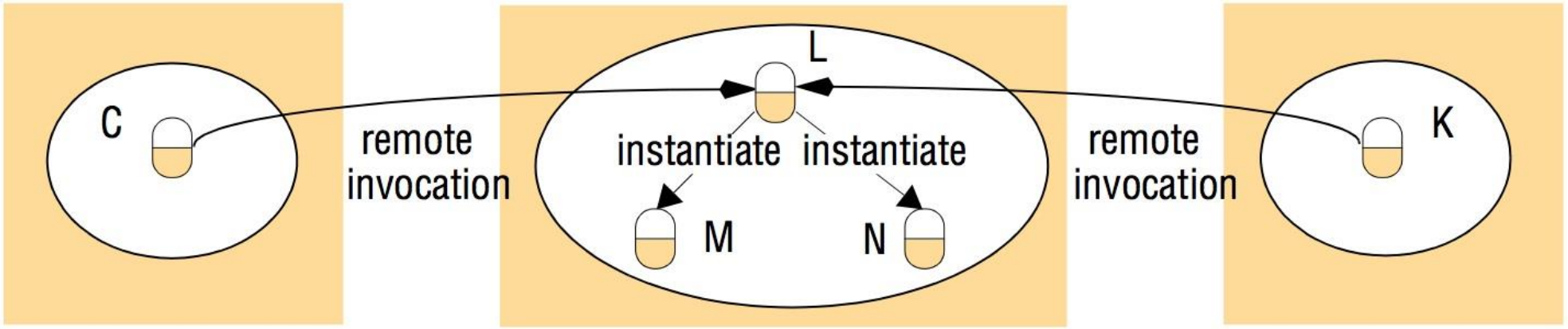
- As in the **non-distributed case**, an action is **initiated** by a **method invocation**, which may result in further invocations on methods in **other objects**.
- But in the distributed case, the **objects involved** in a **chain** of **related invocations** may be located in different processes 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.
  - In Figure 5.12, object A needs to hold a remote object reference to object B. Remote object references may be obtained as the results of remote method invocations. For example, object A in Figure 5.12 might obtain a remote reference to object F from object B.

Design issues for RMI

Actions in a distributed object system •

- When an **action leads** to the **instantiation** of a new object, that **object** will normally **live within the process** where **instantiation is requested** – for example, where the constructor was used.
- If the **newly instantiated** object has a **remote interface**, it will be a **remote object with a remote** object reference.
- **Distributed applications** may provide remote objects with methods for instantiating objects that can be accessed by RMI, thus effectively providing the effect of remote instantiation of objects.

Figure 5.14
Instantiation of remote objects



- For example, if the **object L** in Figure 5.14 contains a method for creating **remote objects**, then the remote invocations from C and K could lead to the instantiation of the objects M and N, respectively.

Design issues for RMI

Actions in a distributed object system •

- For example, if the **object L** in Figure 5.14 contains a method for creating **remote objects**, then the remote invocations from C and K could lead to the instantiation of the objects M and N, respectively.

Next lecture

- Group communication
- Publish-subscribe systems

Assignment

,.differentiate between RPC and RMI

Deadline

Next lecture

mr.ali.h.sh@gmail.com