Sistemas Distribuídos

Interprocess Communication and Synchronization

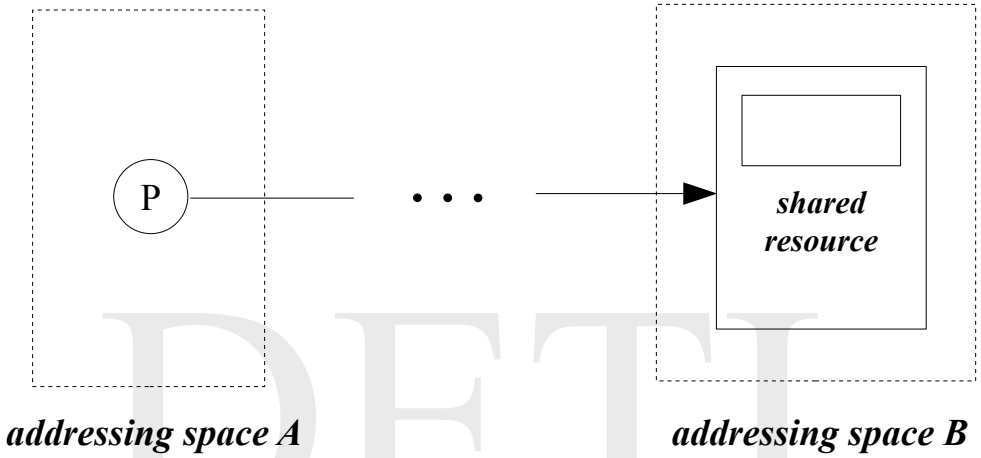
Remote Objects - 1

António Rui Borges

Summary

- *What is a remote procedure call*
- *Architecture of a remote procedure call environment*
- *Code migration*
- *Suggested reading*

What is a remote procedure call - 1



In a *remote procedure call*, the process which invokes a method on a shared resource, and the shared resource itself do not share the same addressing space. This means that

- in the addressing *B* the shared resource belongs to, a reference to it must be generated and made available to others
- in the addressing *A* where the calling process resides, a reference to the shared resource must be obtained before a method can be invoked on it.

What is a remote procedure call - 2

A *remote* procedure call does not behave exactly as a *local* procedure call. There are three distinctive features one should be aware of

- *the call may fail, even if the code has no errors*: this is due to the fact that either the remote resource, belonging to a different addressing space, is not presently instantiated, or the communication infrastructure connecting the two addressing spaces is not operating correctly
- *all procedure parameters and the return value, if it exists, must be passed by value*: since the invoking process and the shared resource reside in different addressing spaces, the sole available means of communication is through passing the relevant data itself, together with its format and structure; thus, information marshaling must take place at the source and information unmarshaling at the destination
- *remote procedure execution takes longer than local procedure execution*, a communication mechanism between the two addressing spaces is implicit, which requires that some kind message exchange between them exists.

What is a remote procedure call - 3

addressing space A

```
/* get a remote reference to the
   shared resource */
remRef = getRef (namingService);
            .  .  .
/* invoke the procedure xyz on the
   shared resource */
try
{ remRef.xyz ();
}
catch (error)
{ /* process error */
}
```

addressing space B

```
/* instantiate the shared resource */
locRef = new SharedResource ();

/* generate a remote reference to the
   shared resource */
remRef = generateRef (pubListenAdd);

/* register it in a naming service */
regRef (remRef);
```

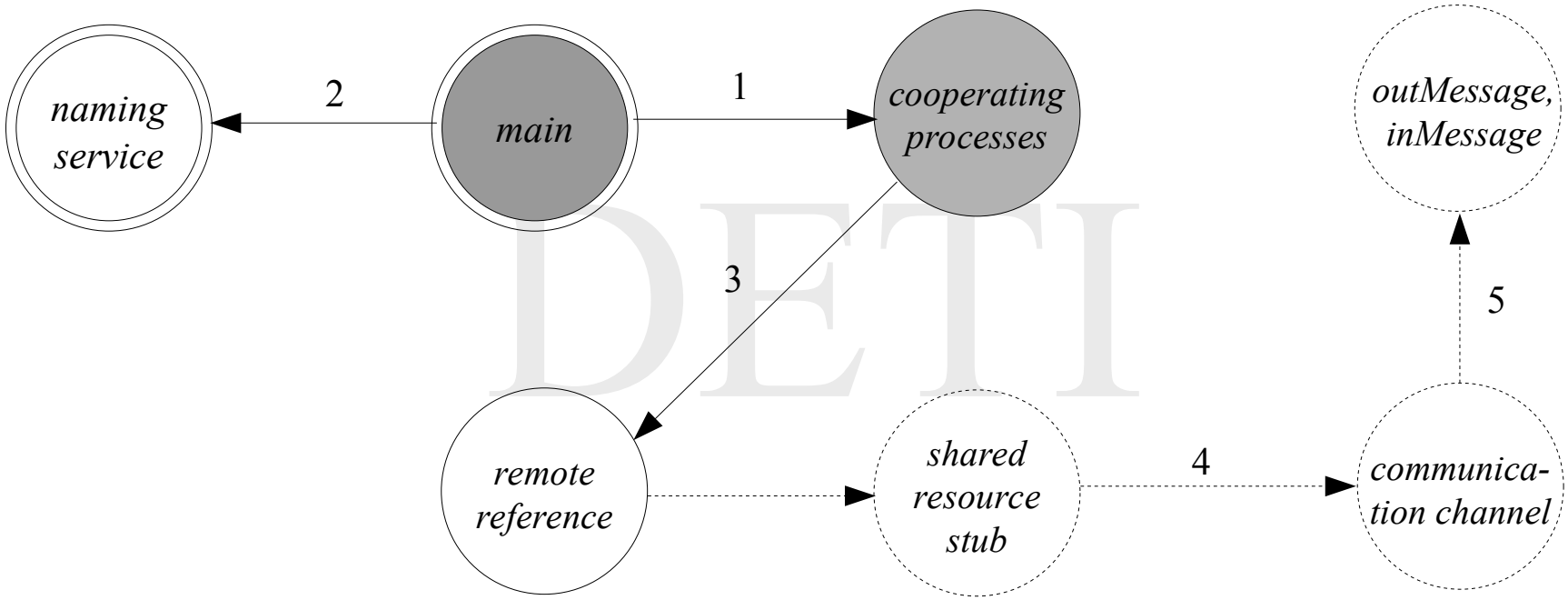
What is a remote procedure call - 4

An environment which implements remote procedure calls, must provide a *naming service* to register shared resources which are to be accessed remotely. On way of looking at it is by assuming it works like a dynamically organized telephone directory, mapping in this case the publicly known name of a shared resource to its location and access features. Thus, the application programmer in order to have network transparency, is only required to know the location of the naming service and the name of the entry to the shared resource in it.

On the other hand, the *remote* reference to the shared resource can be thought of containing the internet address of the platform where the shared resource is instantiated, the port number used to establish a communication with it, the signature of all the procedures which can be invoked on it and the name of the files which provide a description of the data types used as procedure parameters or return values.

Architecture of a remote procedure call environment – 1

Addressing space A



1 – instantiate, start, join
2 – get remote reference
3 – invocation of remote methods

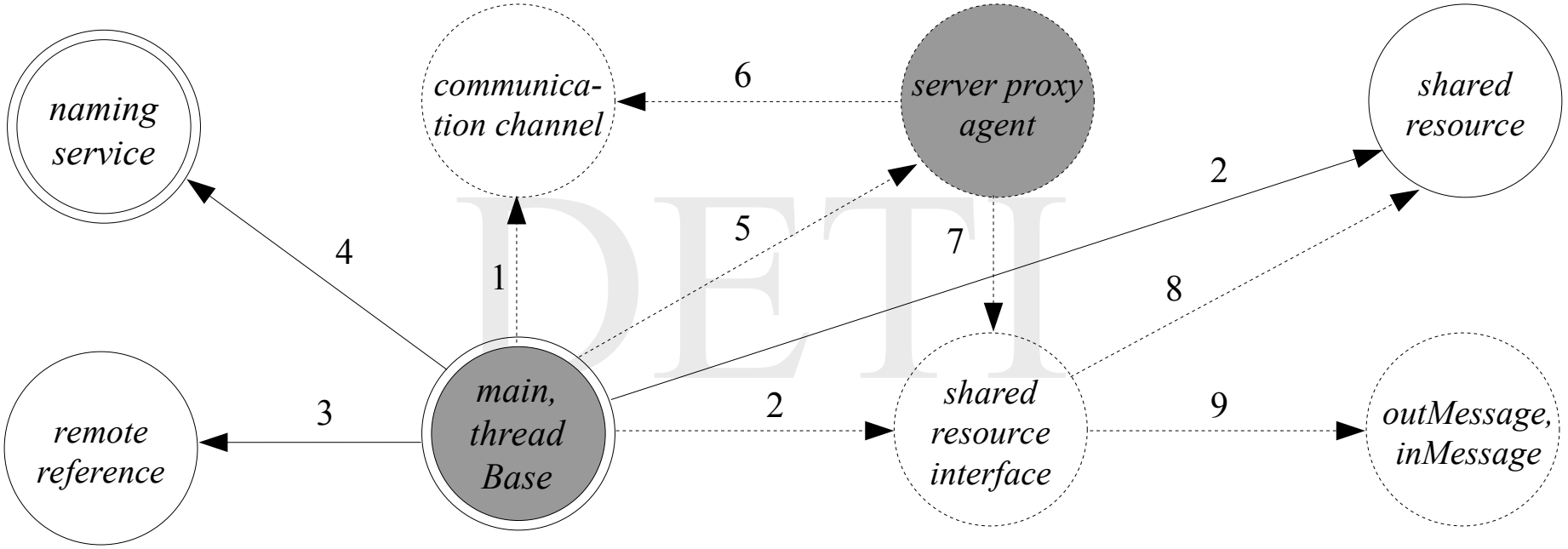
4 – instantiate, open, close, writeObject, readObject
5 – instantiate, get field values

Architecture of a remote procedure call environment – 2

- only the code associated with the entities described by continuous lines have to be written by the application programmer
- the code associated with the entities described by dashed lines is generated by the environment in a manner completely transparent to the programmer
- the programmer has only to provide the signature of all the procedures which can be invoked on the remote object and a description of the data types used as procedure parameters or return values

Architecture of a remote procedure call environment – 3

Addressing space B



1 – instantiate, start, end, accept
2 – instantiate
3 – generate
4 – register
5 – instantiate, start
6 – readObject, writeObject, close
7 – processAndReply
8 – method invocation
9 – instantiate, get field values

Architecture of a remote procedure call environment – 4

- only the code associated with the entities described by continuous lines have to be written by the application programmer
- the code associated with the entities described by dashed lines is generated by the environment in a manner completely transparent to the programmer
- the programmer has only to provide the signature of all the procedures which can be invoked on the now local object and a description of the data types used as procedure parameters or return values
- the aggregate of the entities described by dashed lines is usually called *skeleton* and has an execution which is independent of the *main thread* of the application
- thus, the *thread base* may be thought of as a thread which is instantiated and started when the remote reference is generated

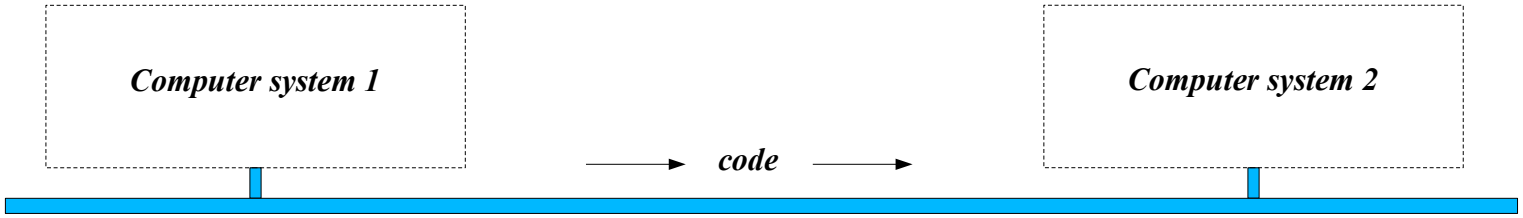
Code migration – 1

Code migration is a highly desired feature in a distributed system which enables some components of an application to be transferred from one processing node of the underlying parallel computer system to another during program execution.

Several reasons may lead to this

- *taking advantage of the computation power of specific processing nodes*: not all the nodes of a parallel computer system may be similar, therefore, parts of the computations to be carried out may be moved to specific nodes, on demand and in a dynamic way, to have the program run more efficiently
- *fault tolerance*: during execution, some of the processing nodes may fail, so it would be important, if one wants not to crash the application being executed, to be able to detect the malfunction and perform a dynamic reconfiguration of the software components previously assigned to the faulty nodes to new ones.

Code migration – 2



An issue one has to deal with is what form the code to be moved will take.

Several options are available

- *executable code*: this is the simpler form, but it requires that the original and the destination nodes are relatively similar so that the code will run without further processing
- *source code*: this is the most general form, no assumptions need to be made about the similarities of the original and the destination nodes
- *intermediate code*: this is typical of situations where the code to be moved is run through an interpreter.

Code migration – 3

Code migration, although being quite a desirable feature to have, has *security* problems in a general context. That is, one has to ensure that the code received at the destination node, being any, will not put at risk the integrity of the resources of the computer system where the node is located.

The way one usually deals with the issue, is to introduce a component in the code that manages the migration which will continuously monitor the access to the resources provided by the operating system, deciding on a case based manner whether the accesses should be allowed or denied.

This component is sometimes called the *security manager*.

Suggested reading

- *Distributed Systems: Concepts and Design, 4th Edition, Coulouris*, Dollimore, Kindberg, Addison-Wesley
  - Chapter 5: *Distributed objects and Remote invocation*
    Sections 5.1 to 5.3 and 5.5
- *Distributed Systems: Principles and Paradigms, 2nd Edition,* Tanenbaum, van Steen, Pearson Education Inc.
  - Chapter 10: *Distributed object-based systems*
    Sections 10.1 to 10.3.4