Chapter 5: Remote invocation

From **Coulouris, Dollimore, Kindberg and Blair**
Distributed Systems: Concepts and Design

Edition 5, © Addison-Wesley 2012

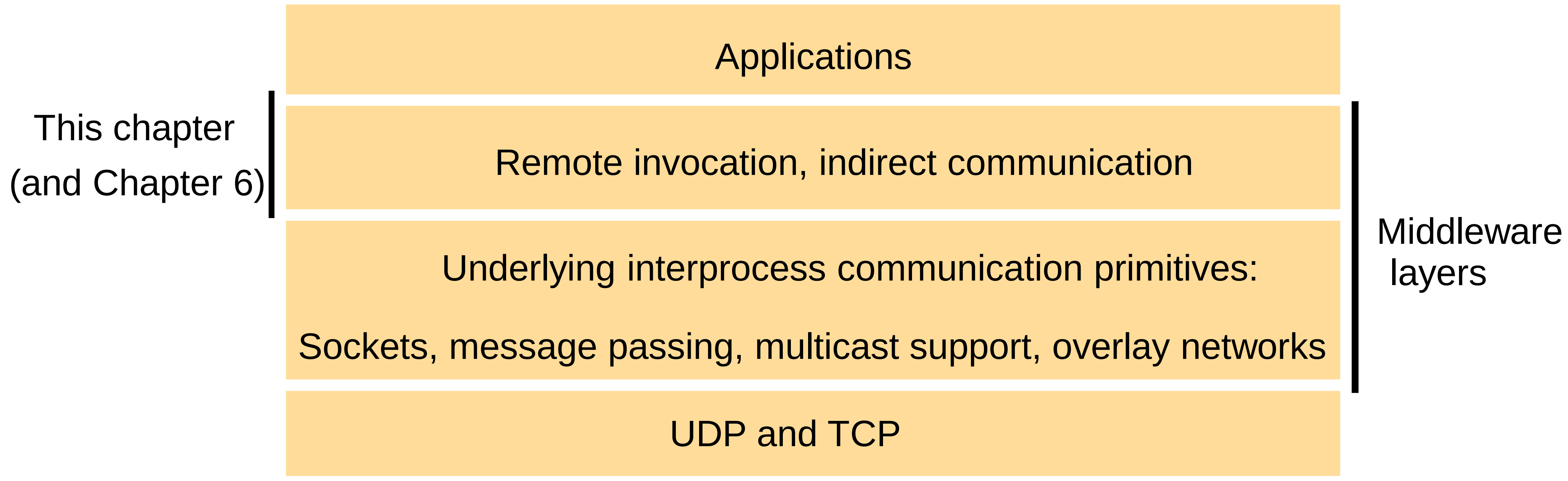
Outlines

- Introduction
- Request-reply protocols
- Remote procedure call
- Remote method invocation

Introduction

- **Request reply protocol** is a two-way exchange of message (like client server message exchange) that is built on the top of message passing. It supports RPC and RMI.
- **Remote procedure call (RPC)** extends the common programming abstraction of the procedure call to a distributed environment, allowing a calling process to call a procedure in a remote node as if it is in a local one.
- **Remote method invocation (RMI)** is like RPC but for distributed objects, with added benefits in term of using object-oriented-programming concept in DS and extending the concept of an object reference to the global distributed environment and allowing the use of object references as parameters in remote invocations.

Figure 5.1 Middleware layers


Applications
This chapter
(and Chapter 6)
Remote invocation, indirect communication
Underlying interprocess communication primitives:
Sockets, message passing, multicast support, overlay networks
Middleware layers
UDP and TCP

Request-reply protocol

- This form of communication is designed to support the roles and message exchanges in typical **client-server** interactions.
- In the normal case, request-reply communication is **synchronous** because the client process blocks until the reply arrives from the server.
- It can also be reliable because the reply from the server is effectively an **acknowledgement** to the client.
- **Asynchronous** request-reply communication is an alternative that may be useful in situations where clients can afford to retrieve replies later.

Request-reply protocol

A **protocol** built over datagrams avoids overheads associated with the TCP stream protocol. In particular:

- **Acknowledgements** are redundant, since requests are followed by replies.
- **Establishing a connection** involves two extra pairs of messages in addition to the pair required for a request and a reply.
- **Flow control** is redundant for the majority of invocations, which pass only small arguments and results.

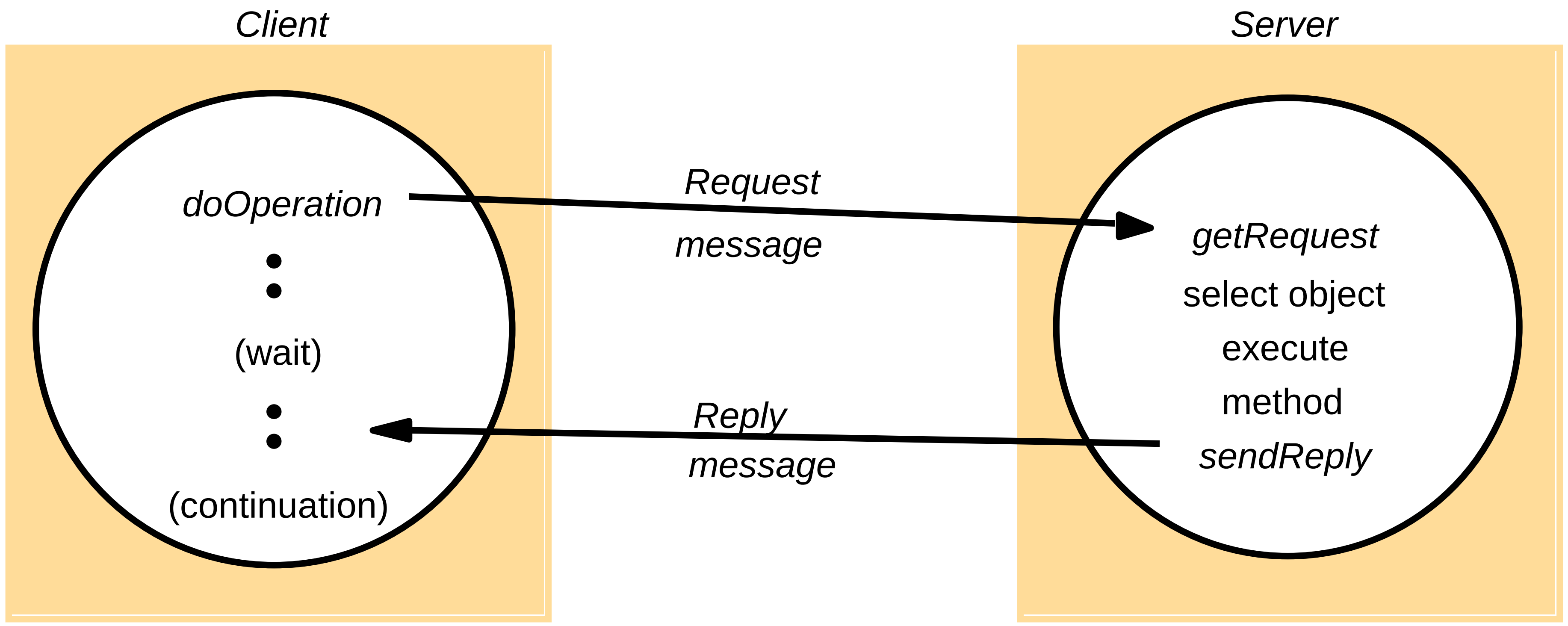
Request-reply protocol

- The **request-reply protocol** is based on three communication primitives, ***doOperation***, ***getRequest*** and ***sendReply***, as shown in **Figure 5.2**.
- This request-reply protocol matches requests to replies.
- It may be designed to provide certain delivery guarantees.
- If **UDP datagrams** are used, the delivery guarantees must be provided by the request-reply protocol, which may use the server reply message as an acknowledgement of the client request message.

Request-reply protocol

- The ***doOperation*** method is used by clients to invoke remote operations.
- The caller of ***doOperation*** is blocked until the server performs the requested operation and transmits a reply message to the client process.
- The ***getRequest*** is used by a server process to acquire service requests.
- When the server has invoked the specified operation, it then uses ***sendReply*** to send the reply message to the client.
- When the reply message is received by the client the original ***doOperation*** is unblocked and execution of the client program continues.

Figure 5.2 Request-reply communication



Request-reply protocol

Failure model of the request-reply protocol:

- If the three primitives ***doOperation***, ***getRequest*** and ***sendReply*** are implemented over **UDP datagrams**, then they suffer from the same communication failures. That is:
    1) They suffer from omission failures.
    2) Messages are not guaranteed to be delivered in sender order.
- To allow for occasions when a server has failed or a request or reply message is dropped, ***doOperation*** uses a **timeout** when it is waiting to get the server’s reply message.
- The action taken when a timeout occurs depends upon the delivery guarantees being offered.
- **Messages** can be re-transmitted.

Request-reply protocol

Timeouts:

- The ***doOperation*** may encounter timeouts when a reply message is lost (or the operation is not performed by the server).
- The request is re-sent again.

Discarding duplicate request messages:

- In cases when the request message is retransmitted, the server may receive it more than once.
- To avoid this, the protocol is designed to recognize successive messages (from the same client) with the same request identifier and to filter out duplicates.

Request-reply protocol

Lost reply messages:

- If the server has already sent the reply when it receives a duplicate request it will need to execute the operation again to obtain the result, unless it has stored the result of the original execution.

History:

- It is used to retransmit replies without re-executing the operations.

Request-reply protocol

Styles of exchange protocols:

- Three protocols, that produce differing behaviors in the presence of communication failures are used for implementing various types of request behavior:
    1) The request (R) protocol
    2) The request-reply (RR) protocol
    3) The request-reply-acknowledge reply (RRA) protocol
- The messages passed in these protocols are summarized in **Figure 5.5.**

Figure 5.5 Remote Procedure Call (RPC) exchange protocols

Name	*Messages sent by*		
	Client	*Server*	*Client*
R	*Request*		
RR	*Request*	*Reply*	
RRA	*Request*	*Reply*	*Acknowledge reply*

Request-reply protocol

Request (R) protocol:

- In the **R protocol**, a single ***Request*** message is sent by the client to the server.
- The **R protocol** may be used when there is no value to be returned from the remote operation and the client requires no confirmation that the operation has been executed.
- The **client** may proceed immediately after the request message is sent as there is no need to wait for a reply message.
- This **R protocol** is implemented over **UDP datagrams** and therefore suffers from the same communication failures.

Request-reply protocol

Request-Reply (RR) protocol:

- The RR protocol is useful for most client-server exchanges because it is based on the request-reply protocol.
- Special acknowledgement messages are not required, because a server's reply message is regarded as an acknowledgement of the client's request message.

Request-Reply-Acknowledge reply (RRA) protocol:

- The RRA protocol is based on the exchange of three messages: request-reply-acknowledge reply.
- RRA clears the history at the server side.

Request-reply protocol

Use of TCP streams to implement request-reply protocol:

- If the **TCP protocol** is used, it ensures that request and reply messages are delivered reliably, so there is no need for the request-reply protocol to deal with retransmission of messages and filtering of duplicates or with histories.

Request-reply protocol

HTTP: An example of a request-reply protocol

- The **HyperText Transfer Protocol (HTTP)** used by web browser clients to make requests to web servers and to receive replies from them.
- Web servers manage resources implemented in different ways:
  1) As **data** – for example, the text of an HTML page and an image or the class of an applet.
  2) As a **program** – for example, PHP or Python programs that run on the web server.

Request-reply protocol

- **HTTP** is implemented over TCP. In the original version of the protocol, each client-server interaction consisted of the following steps:

  1) The client requests and the server accepts a connection at the default server port or at a port specified in the URL.
  2) The client sends a request message to the server.
  3) The server sends a reply message to the client.
  4) The connection is closed.

Remote Procedure Call (RPC)

In RPC:

- Procedures on remote machines can be called as if they are in local address space.
- It is used for conventional programming; that is, achieving a high level of distribution transparency.
- It involves using request-reply protocol.
- Programming with interfaces (abstraction).
- Programs are organized into modules; each one is implemented in a procedure and has an interface.

Remote Procedure Call (RPC)

Design issues for RPC:

There are **three issues** that are important in understanding this concept:

1. The style of programming promoted by RPC – **programming with interfaces.**
2. The **call semantics** associated with RPC.
3. The key issue of **transparency** and how it relates to remote procedure calls

Remote Procedure Call (RPC)

Programming with interfaces:

- Most modern programming languages provide a means of organizing a program as a **set of modules** that can communicate with one another.
- **Communication** between modules can be by means of procedure calls between modules or by direct access to the variables in another module.
- In order to control the possible interactions between modules, an **explicit interface** is defined for each module.

Remote Procedure Call (RPC)

RPC call semantics:

The ***doOperation*** can be implemented in different ways to provide different delivery guarantees. The main choices are:

- **Retry request message:** Controls whether to retransmit the request message until either a reply is received, or the server is assumed to have failed.
- **Duplicate filtering:** Controls when retransmissions are used and whether to filter out duplicate requests at the server.
- **Retransmission of results:** Controls whether to keep a history of result messages to enable lost results to be retransmitted without re-executing the operations at the server.
- **Figure 5.9** shows the choices of interest, with corresponding names for the semantics that they produce.

Figure 5.9 Call semantics

Fault tolerance measures			*Call semantics*
Retransmit request message	*Duplicate filtering*	*Re-execute procedure or retransmit reply*	
No	Not applicable	Not applicable	*Maybe*
Yes	No	Re-execute procedure	*At-least-once*
Yes	Yes	Retransmit reply	*At-most-once*

Semantics:

- Maybe: The RPC is executed once or not at all
- At-least-once: The RPC is executed at least once
- At-most-one: The RPC is executed at most once

Remote Procedure Call Semantics

Maybe:

- The RPC is executed once or not at all.

At-Least-Once:

- The invoker receives either a result, in which case the invoker knows that the procedure was executed at least once, or an exception informing it that no result was received. Then, the invoker will retransmit request again, and that causes the RPC to be executed again. (So, RPC executed once or more) (at least once).

At-Most-Once:

- The caller receives either a result, in which case the caller knows that the procedure was executed exactly once, or an exception informing it that no result was received. Then, the invoker will retransmit request, however, the server is using history of previous requests. So, if the request is a duplicate, then it will be filtered and there is no need to execute RPC again, only saved result in the history will be resent to invoker. So, RPC executed at most one.

Remote Procedure Call (RPC)

Transparency:

- **RPC** offers at least location and access transparency, hiding the physical location of the (potentially remote) procedure and accessing local and remote procedures in the same way.

Remote Method Invocation (RMI)

RMI (Remote Method Invocation):

- **RMI** is closely related to **RPC** but extended into **distributed objects**.

The common between RMI and RPC are as follows:

1. They both support programming with interfaces.
2. 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.
3. 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.

Remote Method Invocation (RMI)

The **differences** between **RMI** and **RPC** are as follows:

1) The programmer is able to use the full expressive power of object-oriented programming in the development of distributed systems software, including the use of objects, classes and inheritance, and can also employ related object-oriented design methodologies and associated tools.

2) Building on the concept of object identity in object-oriented systems, all objects in an RMI-based system have unique object references (whether they are local or remote), such object references can also be passed as parameters, thus offering significantly richer parameter-passing semantics than in RPC.