Remote Invocation

- request-reply communication: most primitive; minor improvement over underlying IPC primitives
  - 2-way exchange of messages as in client-server computing
- **Remote Procedure Call (RPC):** extension of conventional procedural programming model
  - allow client programs to transparently call procedures in server programs running in separate processes, and in separate machines from the client
- **Remote Method Invocation (RMI):** extension of conventional object oriented programming model
  - allows objects in different processes to communicate
  - extension of local method invocation: allows object in one process to invoke methods of an object living in another process

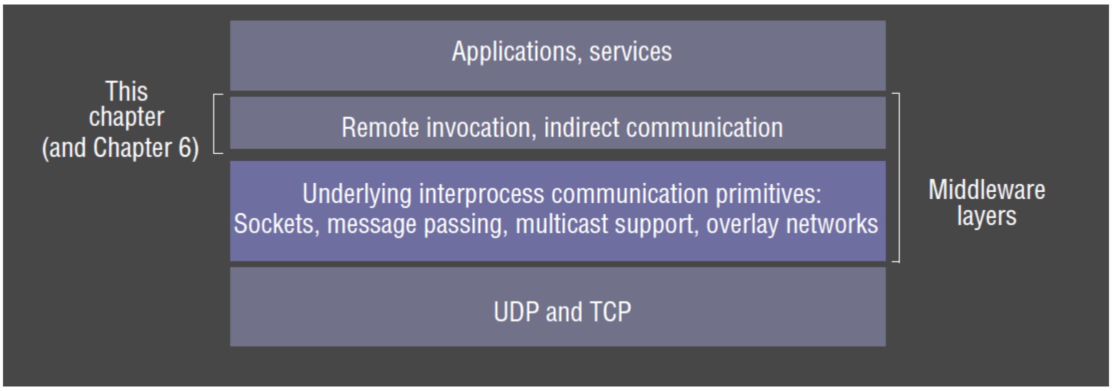

Applications, services
This chapter (and Chapter 6)
Remote invocation, indirect communication
Underlying interprocess communication primitives: Sockets, message passing, multicast support, overlay networks
Middleware layers
UDP and TCP

Figure 1: Middleware Layers

Request-Reply Protocol

- most common exchange protocol for remote invocation

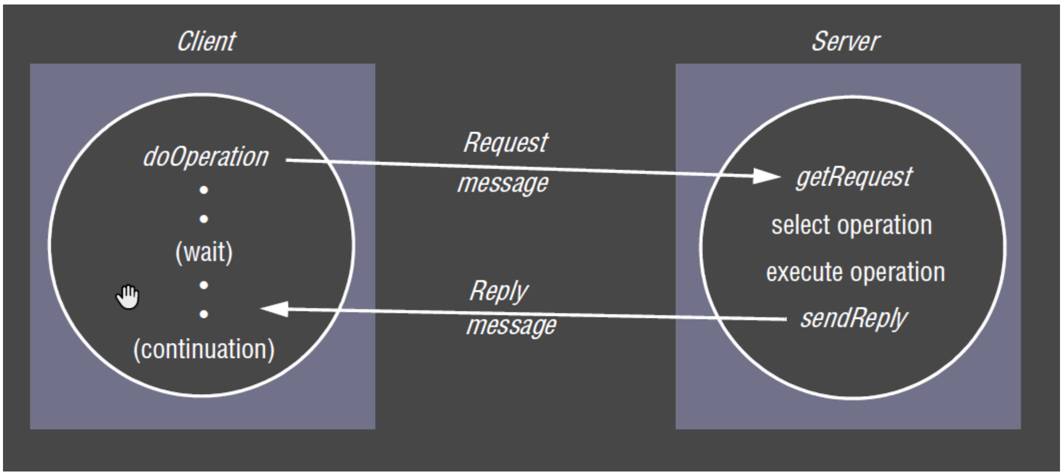


Figure 2: Request-Reply communication

Operations

- `doOperation()`: send request to remote object, and returns the reply received
- `getRequest()`: acquire client request at server port
- `sendReply()`: sends reply message from server to client

Design issues

- timeouts: what to do when a request times out? how many retries?
- duplicate messages: how to discard?
    - e.g. recognise successive messages with the same request ID and filter them
- lost replies: dependent on idempotency of server operations
- history: do servers need to send replies without re-execution? then history needs to be maintained

Design decisions

- retry policy
    - how many times to retry?
- duplicate filter mechanism
- retransmission policy

Exchange protocols

Different flavours of exchange protocols:

- **request (R):** no value to be returned from remote operation
  - client needs no confirmation operation has been executed
  - e.g. sensor producing large amounts of data: may be acceptable for some loss
- **request-reply (RR):** useful for most client-server exchanges. Reply regarded as acknowledgement of request
  - subsequent request can be considered acknowledgement of the previous reply
- **request-reply-acknowledge (RRA):** acknowledgement of reply contains request id, allowing server to discard entry from history

TCP vs UDP

- limited length of datagrams may affect transparency of RMI/RPC systems which should be able to accept data of any size
- TCP can be chosen to avoid multipacket protocols, avoiding this issue
- TCP additional overheads: acknowledgements, connection establishmen
- TCP also ensures reliable delivery
  - no need to filter duplicates or use histories
- TCP therefore simplifies implementation of request-reply protocol
- if application doesn't require all of TCP facilities, more efficient, tailored protocol can be implemented over UDP

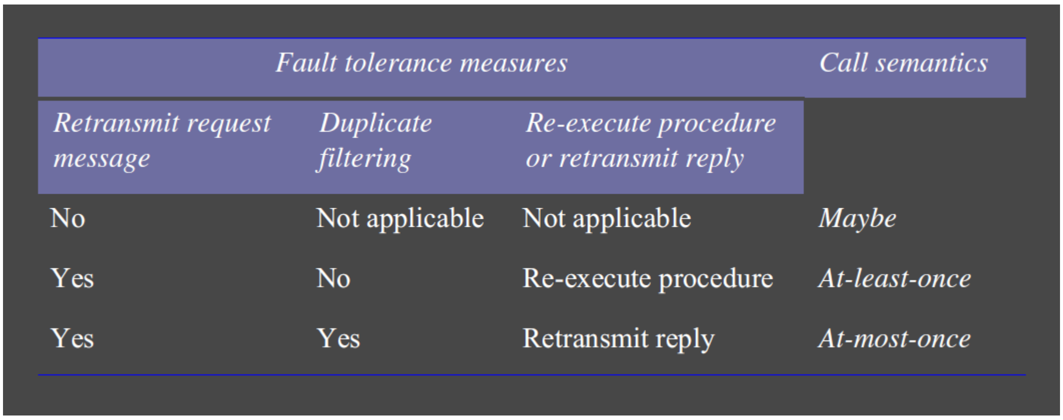
Invocation semantics

- **maybe:** RPC may be executed once or not at all
  - unless call receives result, it is unknown whether RPC was called
- **at-least-once:** either
  - remote procedure was executed at least once and caller received a response, or
  - caller received exception to indicate remote procedure was not executed at all
- **at-most-once:** RPC was either

  - executed exactly once, in which case caller received response, or
  - not executed at all, and caller receives an exception

- level of transparency provided depends on design choices and objectives
- Java RMI supports at-most-once invocation semantics
- Sun RPC supports at-least-once

Fault tolerance



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*

Figure 3: Call Semantics

Transparency

- location and access transparency are usually goals for remote invocation
- sometimes complete transparency undesirable:
  - remote invocations are more prone to failure due to network/remote machines
  - latency of remote invocations significantly higher than local ones
- many implementations provide access transparency, but not complete location transparency, allowing programmer to optimise based on location

HTTP: RR protocol

- see comp sys notes

RPC

- RPCs enable clients to execute procedures in server processes based on a defined service interface
- generally implemented over request-reply protocol

RPC Roles

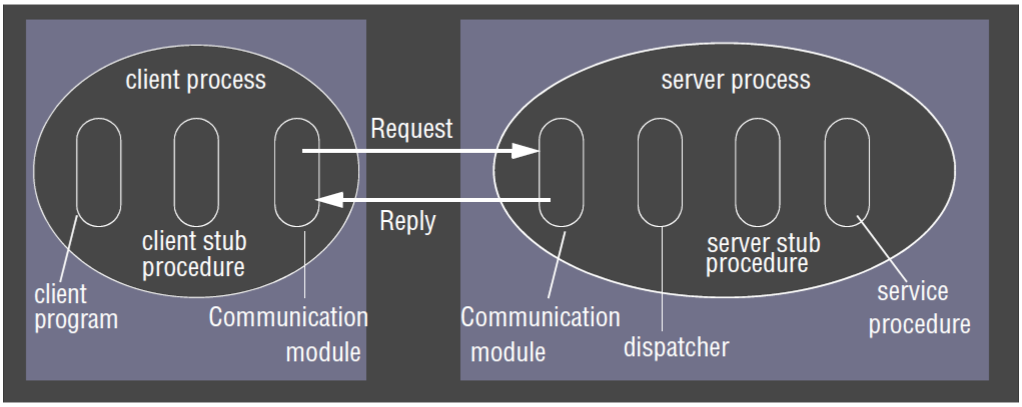


Figure 4: RPC roles

- **communication module:** implements design w.r.t. retransmission of requests, duplicate handling, result retransmission
- **client stub procedure:** behaves like a local procedure to client
  - marshals procedure identifiers and arguments, and passes it to communication module
  - unmarshals the results in the reply
- **dispatcher:** selects server stub based on procedure identifier, forwarding request to the server stub
- **server stub procedure:** unmarshalls arguments in request message, and forwards to service procedure
  - marshals arguments in result message and returns to client
- **service procedure:** actual procedure to call, implements procedures in the service interface
- client/server stub procedures, as well as dispatcher, can be generated automatically by an interface compiler