Institute of Computer Science
Department of Mathematics and Computer Science

Freie Universität Berlin

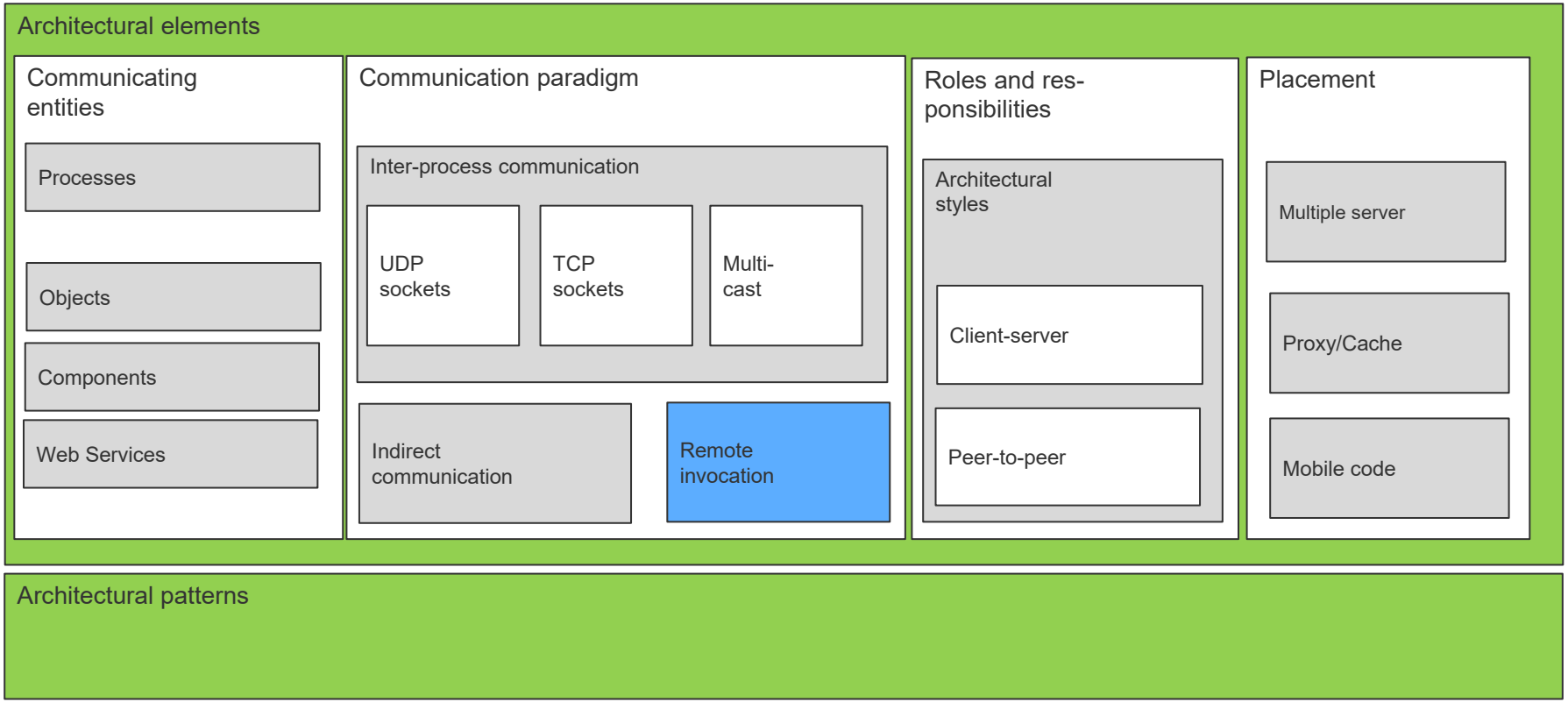
Algorithms and Programming IV

Remote Invocation

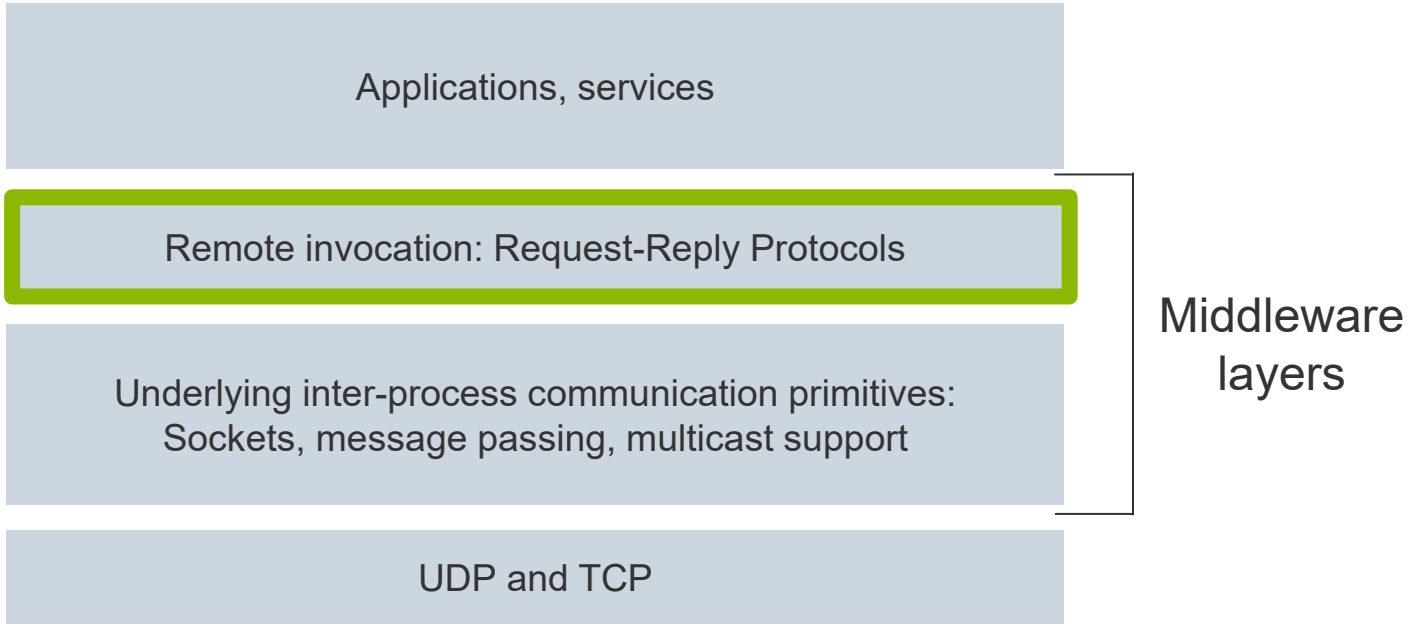
Summer Term 2021 | 16.06.2021

Barry Linnert

Recap: Architectural Model



Middleware Layers



Motivation

Observation 1:
Message-based interaction between processes over sockets in distributed software is cumbersome, untyped, error-prone.

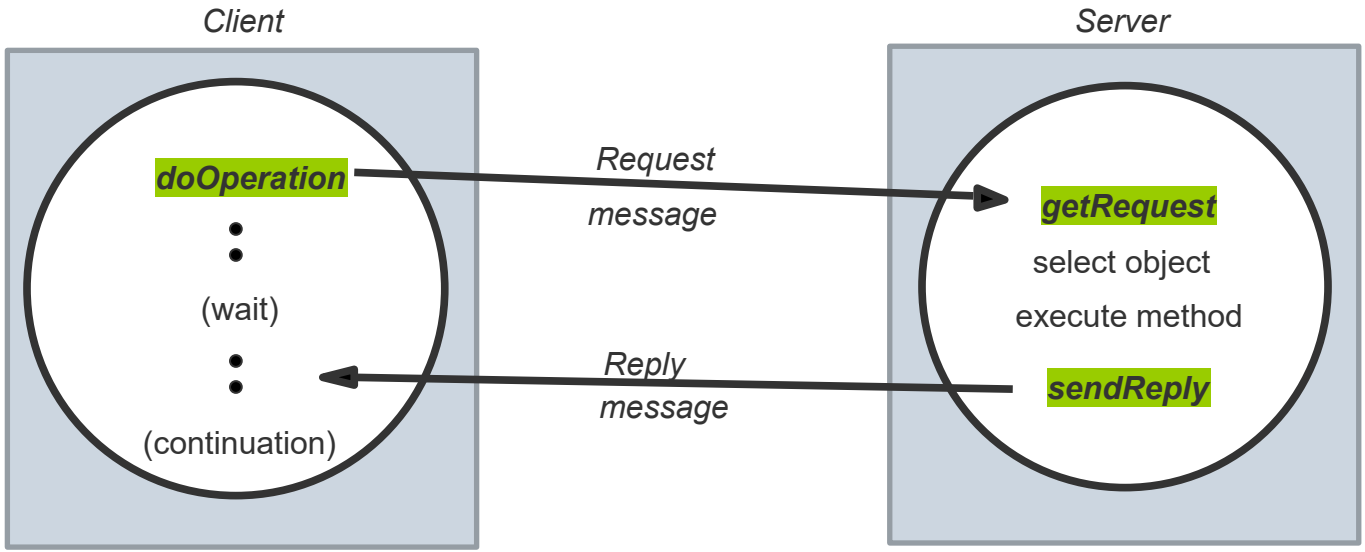
Observation 2:
The service-oriented question/answer pattern is similar to the call-based interaction pattern between procedures, methods, ... for non-distributed software.

Conclusion:
Design a question/answer message pair as a programming-language call – and thus, develop distributed software similar to a non-distributed software!

Remote invocation

REQUEST-REPLY PROTOCOLS

UDP Style Request-Reply Protocol



Operations of the Request-Reply Protocol (UDP)

public byte[] doOperation (RemoteRef s, int operationId, byte[] arguments)

sends a request message to the remote server and returns the reply.
The arguments specify the remote server, the operation to be invoked and the arguments of that operation.

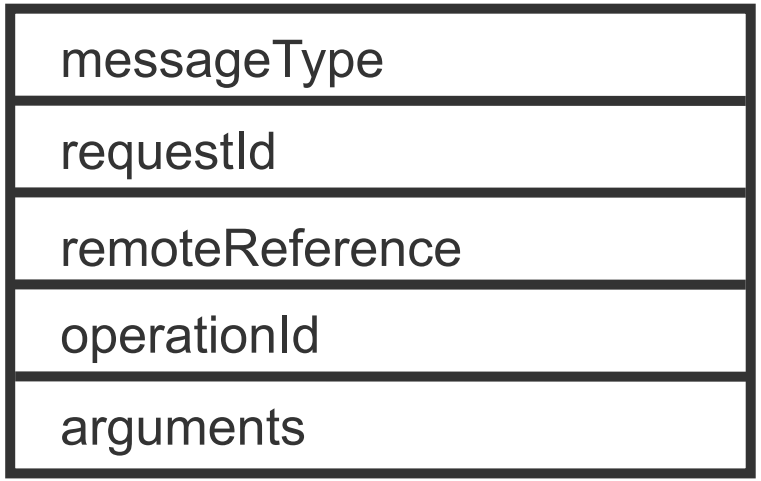
public byte[] getRequest ();

acquires a client request via the server port.

public void sendReply (byte[] reply, InetAddress clientHost, int clientPort);

sends the reply message reply to the client at its Internet address and port.

Request-Reply Message Structure



int (0=Request, 1= Reply)

int

RemoteRef

int or Operation

array of bytes

Message Identifiers

Unique message identifies is needed for any scheme that involves management of messages to provide additional properties such as

- reliable delivery,
- request-reply communication.

Parts of a message identifier

- requestID, which is taken from an increasing sequence of integers by the sending process
- an identifier for the sender process, for example, its port and Internet address.

UDP Style request-reply protocols

FAILURE MODEL OF UDP REQUEST-REPLY PROTOCOL

Approaches to Handle Failures

Repeatedly request message

- doOperation sends the request message repeatedly until either it gets a reply or it is reasonable sure the the delay is due to lack of response from the server, rather than lost messages

Discarding duplicate request messages

- Server may receive more than one request message, e.g. server needs longer than the client's timeout to execute the command and return reply
- Problem: Operation is more than once executed to the same request
- Protocol is designed to recognize successive messages (from the same client) with the same request identifiers

Approaches to Handle Failures (*cont.*)

Lost reply messages

- Problem: Server has already sent the reply when it receives a duplicate request it will need to execute the operation again to obtain the result
- *Idempotent* operation is an operation that can be performed repeatedly with the same effect as if it had been performed exactly once

History

- Refer to a structure that contains a record of (reply) messages that have been transmitted
- Entry contains: request identifier, message, identifier of a client

Possible Exchange Protocols

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

R = no response is needed and the client requires no confirmation

RR = a server's reply message is regarded as an acknowledgement

RRA = Server may discard entries from its history

(Identified by Spector[1982])

Request-reply protocols

USE OF TCP STREAMS TO IMPLEMENT REQUEST-REPLY PROTOCOL

HTTP: an Example of a Request-Reply Protocol

HTTP specifies the messages involved in a request-reply exchange, the methods, arguments and results, and the rules for representing (marshalling) them in the messages

Fixed set of resources are applicable to all of server's resources, e.g., GET, PUT, POST

Additional functions

- *Content negotiation*: clients' requests can include information as to what data presentation they can accept (e.g. language)
- *Authentication*: Credentials are used to support password-style authentication

Client/server interaction
HTTP over TCP (original version)

1. The client requests and the server accepts a connection at the default server port or at the 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.

RFC 2616, (Fielding et al. 1999)

Client/server interaction
HTTP 1.1 over TCP

- Usage of persistent connections
- Connections remain open over a series of request-reply exchanges between client and server
- Connection may be closed by client or server any time by sending an indication to the other participant

RFC 2616, (Fielding et al. 1999)

HTTP Methods

GET

- Requests the resource whose URL is given as its argument

HEAD

- Request is identical to GET but does not return any data
- Returns all the information about the data such as time of last modification

PUT

- Requests that the data supplied in the request is stored with the given URL as its identifier either as a modification of an existing resource or as a new resource

HTTP Methods (*cont.*)

POST

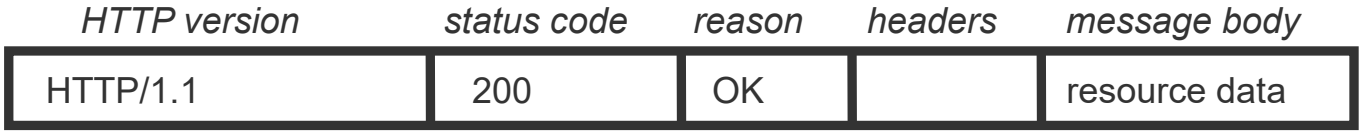
- Is used to send data to the server to be processed in some way
- Designed to deal with
  - Providing a block of data to a data-handling process such as a servlet
  - Posting a message to a mailing list or updating member details
  - Extending a database with an append operation

Additional methods: DELETE, OPTIONS, TRACE

Message Contents

- HTTP request message

- HTTP reply message



Status code definitions and more:
http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

Middleware Layers