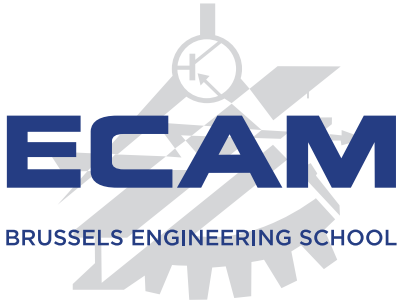


I402A Software Architecture
and Quality Assessment

Session 8

Distributed Architecture

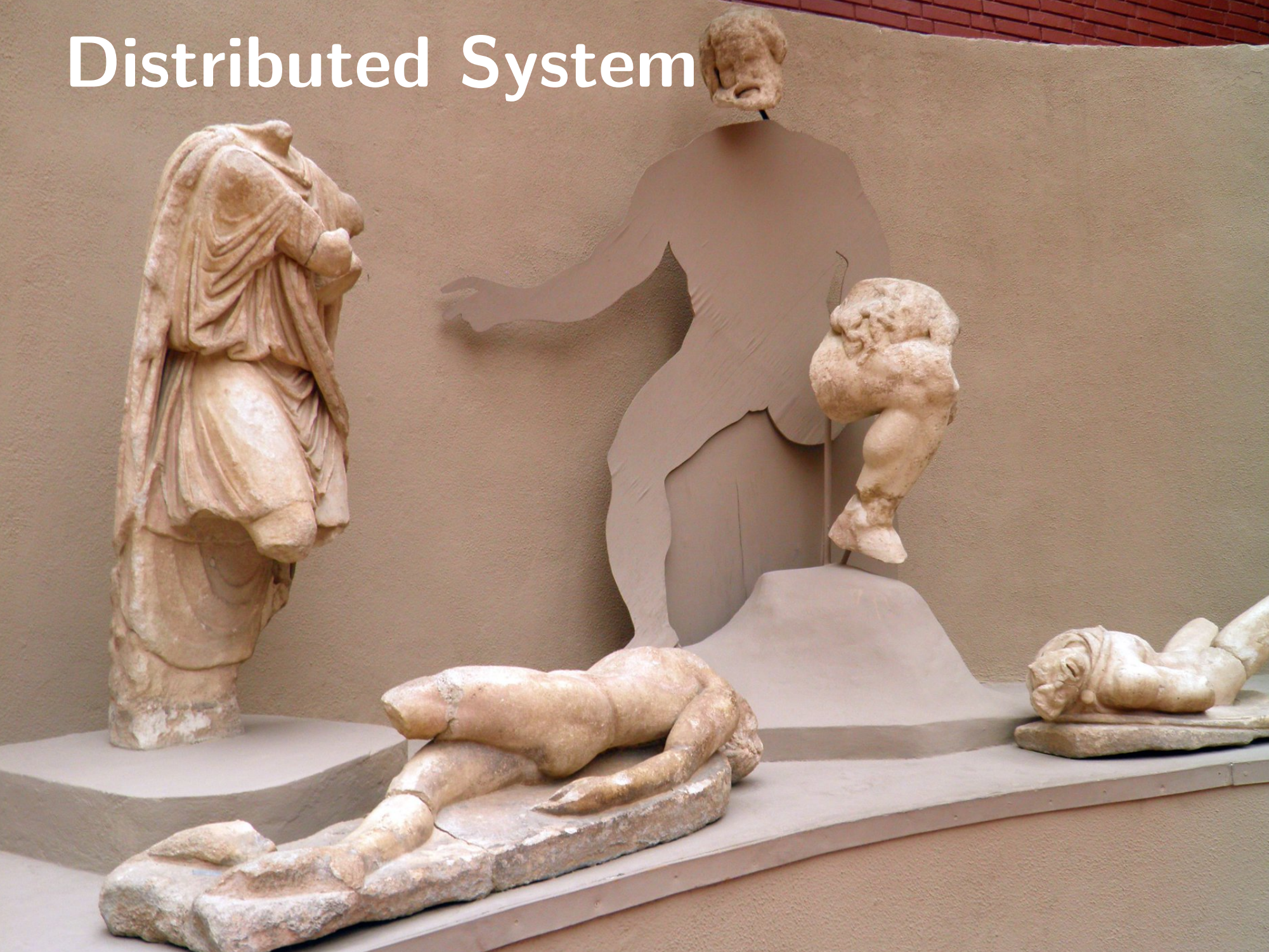
Sébastien Combéfis

Fall 2019

Objectives

- Understand how distributed architecture works
  - Structure, purpose and components
  - Advantages and disadvantages
  - Middleware
- Examples of distributed architecture
  - Client-server architecture
  - Multi-Tier (n-Tier architecture)
  - Broker architecture

Distributed System



Distributed System (1)

- Components distributed on several platforms

  Whether this is hardware or software platforms

- Cooperation between components to achieve a goal

  Using a communication network

- In opposition to a centralised network

  Everything on a single machine with direct communication

Distributed System (2)

- General definition of a distributed system

  "Collection of independent computers that appear as a single coherent system for the users of the system"

- Emergence following technological evolutions

  Powerful and cheap CPUs: PCs, embedded systems, PDAs, etc.

- Several kinds of distributed architecture are possible

  Client-server, Multi-Tier, Broker, Service-Oriented

Distributed System (3)

- **Advantages**
  - Resource saving with inherent distribution
  - Increased speed and reliability
  - Incremental growth of the system
- **Disadvantages**
  - Software complexity and interoperability
  - Need to have a communication network
  - Security and more risk of components failure

Goal

- A distributed system should appear as a single system

  Interactions de plusieurs composants au sein du système

- Five main goals of a distributed system
  - Transparency
  - Openness
  - Reliability
  - Performance
  - Scalability

Transparency

- Make several machines seem to be a single one

  Using uniform access interfaces to functions

- Hide the distribution of the user and applications
  - At the high level, hide the distribution for the user
  - At the low level, make system transparent for programs

Transparency Form (1)

- Resources access

  Hide data and how resources are accessed

- Hidden resources location

  Do not depend on where resources are

- Presence of different technologies

  Hide different languages, OS, frameworks, etc.

- Resources migration

  Hide that the location of a resource can be changed

Transparency Form (2)

- Resources replication

  Hide that a resource can be replicated on components

- Concurrency between several users

  Hide that a resource can be shared between users

- Breakdown and resources failure

  Hide the failure and restart of a resource

- Resources persistence

  Hide the fact that a resource is in memory or on disk

Openness

- Services offer in accordance with published standards

  For example, Internet with protocols published in RFCs

- Standardised interface described with IDL

  Separation of the mechanisms policy

- Make easy the building and change

  Interoperability, portability and scalability

- Use of equipment and software from different vendors

Reliability

- Distributed system must be more reliable than a single system
  - Fraction of time during which the system is available
  - Redundancy improves reliability, but need for consistency
  - Security more complex, but important
  - Fault tolerance to manage transparently

Performance

- Distributed systems must improve performance
  - Increasing the computing power by combinations
  - Loss of performance due to communication
  - Fault tolerance management makes the system heavier
- Two types of usable parallelism
  - Fine-grained with very high degree of interaction
  - Coarse-grained with a better decoupling

Scalability (1)

- The system must be able to evolve easily

  Dynamic management of computing power/storage, etc.

- Increasing throughput by adding resources

  Could be done dynamically during the execution

- Characteristics of the decentralisation
  - No machine has the complete state of the whole system
  - Machine decisions made with local information
  - Failure of a machine does not ruin the whole system

Centralised vs. Distributed

Criterion	**Centralised system**	**Distributed system**
Economy	$--$	$++$
Availability	$--$	$++$
Complexity	$--$	$++$
Consistency	*Simple*	*High*
Scalability	$--$	$++$
Technology	*Homogeneous*	*Heterogeneous*
Security	$++$	$--$

Scalability (2)

- A scalable system must remain efficient
  - When the number of users increases
  - When the number of resources increases
- Two problems that are different
  - Size scalability prevents the system from overloading
  - Geographical scalability avoids communication delays
- Three techniques to scale the system

  Decentralisation, reduced communications and replication

Scalability (3)

- Eliminate performance bottlenecks for size scalability
  - Centralised services (e.g. unique server)
  - Centralised data (e.g. unique DNS table)
  - Centralised algorithm (e.g. computation based on global info.)
- Reducing communications between machines

  Move some server computations to the client
- Data and services replication
  - Decrease of the server load and transfer latency
  - Risk of consistency problems

Middleware

Middleware

- Setting up a middleware infrastructure

  Supports distributed application development and execution

- Acts as a buffer between applications and the network

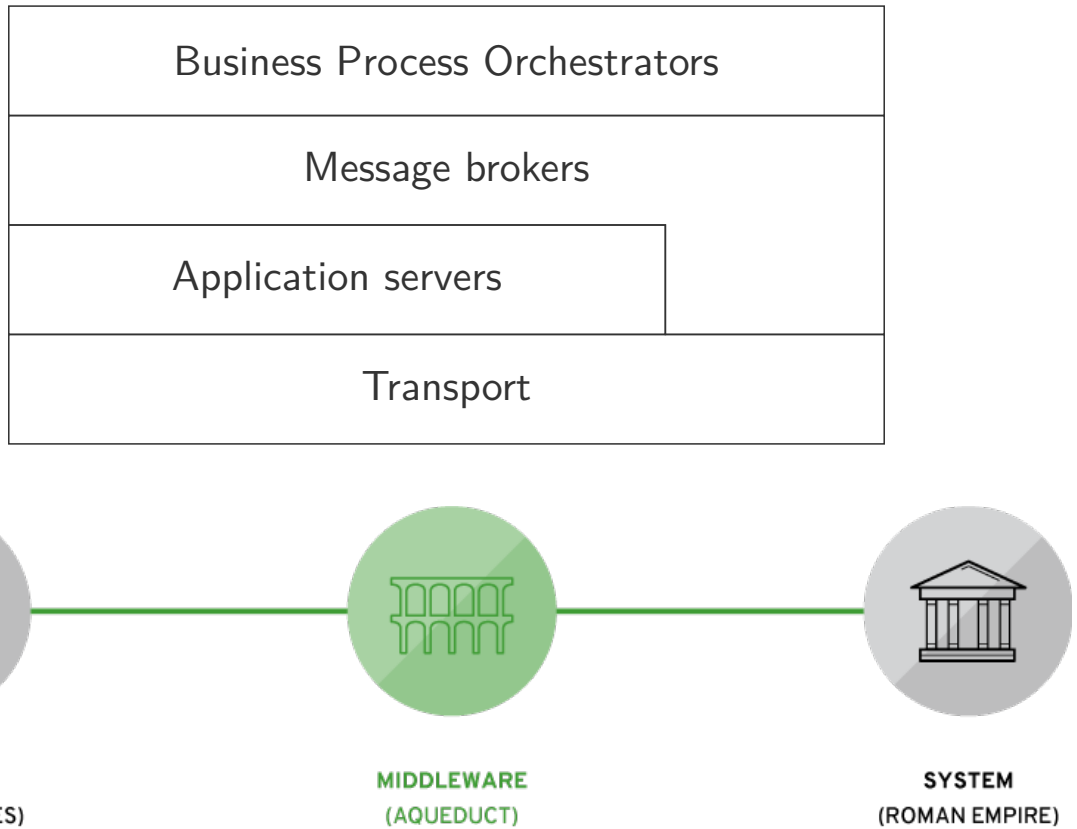
  Manages and supports the components of the application

- Similar to plumbing and piping
  - Component connection and communication path
  - Specific topology: one-to-one, one-to-many, etc.
  - Middleware infra. invisible to the user (except if failure)

Middleware Technology (1)

- Several abstraction levels for a middleware infrastructure

  Layers view, each with its own responsibility



Middleware Technology (2)

- Transport layer for sending queries and moving data

  Direct exchange of data between application components

- Building a server application layer on the transport

  Transaction, security and services directory

- Adding a message processing engine by brokers

  High level exchange, manipulation and message routing

- Business Process Orchestrators (BPO) for workflows

  Business process that can take several days

Distributed System Types

- Three main types of distributed systems

  Depending on the type of operation and the environment

- Distributed computing system

  Cluster and computing grid

- Distributed information system

  Transaction processing, enterprise integration

- Ubiquitous/embedded distributed system

  Maison embarquée, e-Health Care, réseau de senseurs

Pitfalls

- Several beginners traps in distributed systems
  - The network is reliable, secure and homogeneous
  - The network topology does not change
  - Zero latency and infinite bandwidth
  - Zero cost of transport
  - There is only one administrator
- Need for coordination between multiple actors

  Deploying a distributed application cannot be improvised


Client-Server

Client-Server Architecture (1)

- Decomposition in two subsystems or logical processes
  - **Client**

    Process that issues requests to the server
  - **Server**

    Receives requests, handles them and sends answer to customer
- Services offered by the server and used by the client

  Server does not know clients, but not conversely

Client-Server Architecture (2)

Light client

Presentation

Data management
Application logic

Heavy client

Presentation
Application logic

Data management

Light Client

- Main role played by the server
  - Server responsible for the application logic and data
  - Client responsible only for the presentation
- Migrating a legacy system to client-server architecture

  The system acts as a server and the client implements an UI
- Big load on the server and the network

  Many operations on the server and full of communications

Heavy Client

- Moving the application logic to the client
  - Server responsible only for data
  - Client responsible for application logic and user interaction
- Development of a new system following client-server

  Based on known capabilities of client systems
- Much more complex management than with a light client

  New versions of the application to deploy on the clients

Advantage

- Separation of concerns between presentation and logic

  May be used to implement a MV architecture*

- Simplifies design and development of distributed application

  Reusing server components, possibility of concurrency

- Easy to migrate or integrate existing applications

  High flexibility and modification of a distributed application

- Very good use of resources on the server side

  When full of clients on a high performance server

Disadvantage

- Potential heavy load put on the server

  Or moved to the client with heavy clients

- Security of the server is very important

  Availability and reliability of the server is critical

- Limited test and scalability of client-server application

  The server or the client can quickly become very big and complex


Multi-Tier

Multi-Tier Architecture

- Client-server with physical separation of features

  Presentation, application logic, data management, etc.

- Separation of an application in stages (tiers)

  Allows a greater modularity of the application

- Creating flexible and reusable applications

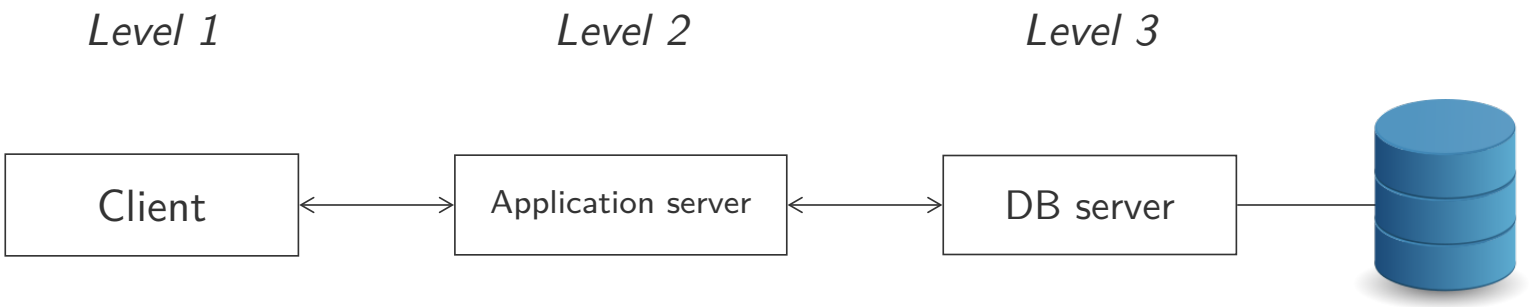
  Direct addition, deletion and modification of the application

1-Tier and 2-Tier

- 1-Tier architecture in one monolithic block
  - A single installed program managing everything
  - Data saved and accessed locally (file)
  - A single user on a heavy client
- Client-server is typically a 2-Tier architecture
  - Logic and presentation to clients
  - Data persistence to the server
  - Continuous communication between the client and the server
  - Security by authentication and data protection by role

3-Tier

- Most common use is the 3-Tier architecture
  - **Presentation** as a web page or GUI
  - **Application server** with business logic coordinating everything
  - **DB server** stores and retrieves information from DB



n-Tier

- n-Tier architecture adds abstraction and interaction

  Break down complex requests, delegation, etc.

- Several types of levels that can be added
  - Database and data access
  - High level business layer for clients
  - Validation
  - Presentation and templating

Advantage

- Intermediate solution between the light and heavy client

  Better performance than light, easier to handle than heavy

- Improve the reusability and scalability of the system

  And better maintainability and flexibility

- Support for the multi-threading of the application

  Adding servers on-the-fly as demand grows

Disadvantage

- Reliability and availability of important servers

  Not many test tools for Multi-Tier applications

- Heavy communication between layers/components

  Communication protocol between several technologies

Broker

Broker Architecture (1)

- Middleware architecture for distributed application

  Coordinates communication between registered client and server

- Maximum decoupling of the components of the system

  Interaction by invoking remote services

- Communication through object request broker
  - Software bus middleware system
  - No direct interaction between client and server
  - A server proposes services by registering them

Broker Architecture (2)

- Three great strenghts of the broker architecture
  - Component must access services offered by others

    Remotely and independently of the location
  - Exchanging, adding and removing components at runtime

    Somewhat following the hot plug and play paradigm
  - Hide system and implementation details

    From the users of components and services
- Broker component decouples clients from the server

  Calls for services by sending messages across the network

Broker

- Coordination of the communication made by the broker

  Forward and dispatch results and exceptions

- Several responsibilities
  - Service requests brokering
  - Retains server service record
  - Provides APIs to clients to allow queries

Stub and Skeleton

- Static generation of stubs deployed on clients

  Used as proxies for the client

- Makes it possible to use a distant object as it was local

  Hidden IPC and parameters (de)serialisation

- Skeletons generation for server services

  Requests (de)serialisation, then execution

Bridge

- Connection of different networks

  With different communication protocols

- Used for interoperation between two brokers

  Translation from one format to another

CORBA (1)

- Common Object Request Broker Architecture (CORBA)

  International standard for middleware Object Request Broker

- Defined by the Object Management Group (OMG)

  First version released in Octobre 1991

- Eases the communication between multi-platform systems

  Operating systems, programming languages, hardware, etc.

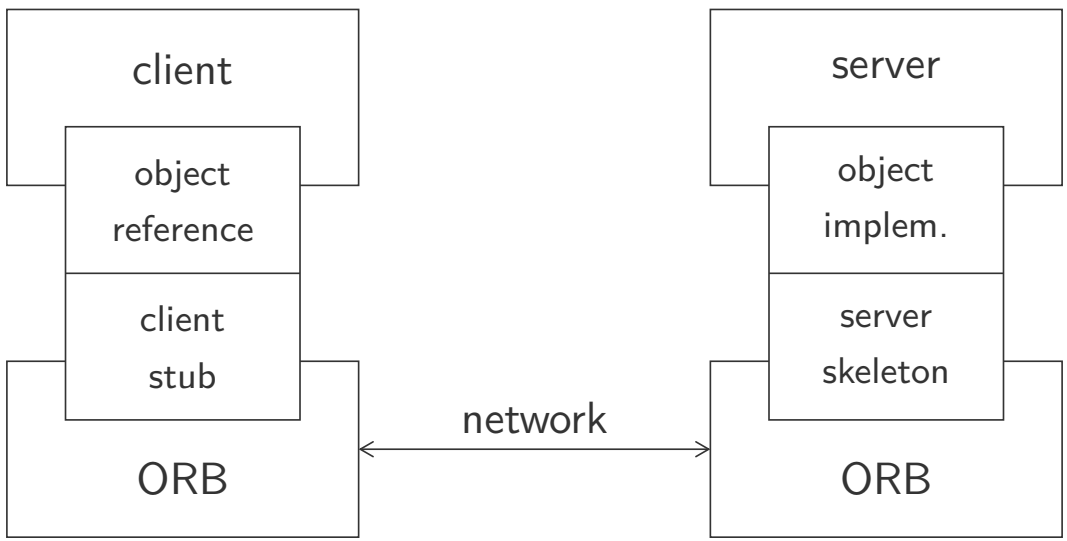
CORBA (2)

- Using an Interface Definition Language (IDL)

  Specification of the interface exposed by an object

- Mapping from an IDL to the used programming language

  Adapter pattern managed by an Object Request Broker (ORB)



Java RMI

- Java Remote Method Invocation (RMI)

  Proprietary solution for Java objects

- Calling a method on a remote object

  On a remote machine or another JVM

- Three components involved in a program with RMI

  Client, server and objects register

Advantage

- Location transparency for the server, service and client
- Modifiability and extensibility of components

  Modification possible while preserving the same interfaces

- Great portability of the broker system and interoperablity

  OS & network hidden by indirection layers, API, proxy and bridge

- Reusability of already defined services

Disadvantage

- Decreased performance due to the use of IPC

  And also because of the indirection layers for portability

- Less good fault tolerance

  Services offered by a server fall with it

References

- Stanislav Kozlovski, *A Thorough Introduction to Distributed Systems: What is a Distributed System and why is it so complicated?*, April 27, 2018.
  https://medium.com/free-code-camp/a-thorough-introduction-to-distributed-systems-3b91562c9b3c
- James Furbush, *Distributed systems: A quick and simple definition: Get a basic understanding of distributed systems and then go deeper with recommended resources*, December 6, 2018.
  https://www.oreilly.com/ideas/distributed-systems-a-quick-and-simple-definition
- Red Hat, *What is middleware?*, retrieved on October 14, 2019.
  https://www.redhat.com/en/topics/middleware/what-is-middleware
- Rameez m Sydeek, *What Is Middleware & How Does It Work?*, December 21, 2017.
  https://www.feedough.com/what-is-middleware-how-does-it-work
- Anirudh Rajmohan, *Client-Server Architecture*, September 11, 2018.
  https://medium.com/@anirudh.rajmohan/client-server-architecture-1bbaf457876a
- Stackify, *What is N-Tier Architecture? How It Works, Examples, Tutorials, and More*, May 19, 2017.
  https://stackify.com/n-tier-architecture
- Vijini Mallawaarachchi, *10 Common Software Architectural Patterns in a nutshell*, September 4, 2017.
  https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013

Credits

- Carole Raddato, April 20, 2011, https://www.flickr.com/photos/carolemage/5741945301.
- Rennett Stowe, December 26, 2018, https://www.flickr.com/photos/tomsaint/46597544302.
- Red Hat, https://www.redhat.com/cms/managed-files/diagram-middleware-aqueduct-850x219.png.
- Gideon, May 29, 2006, https://www.flickr.com/photos/malias/155901595.
- michael, September 27, 2014, https://www.flickr.com/photos/fallsroad/16009679178.
- https://openclipart.org/detail/68413/database.
- Voluntary Amputation, April 5, 2010, https://www.flickr.com/photos/photopunk13/4515404300.