Software Architectures

Distributed Systems

Software Architectures to Handle Complexity

Distributed systems are complex

- In order to manage their intrinsic complexity, distributed systems should be properly organised
- Organisation of a distributed system is mostly expressed in terms of its software components

Software architectures expresses component organisation

- Many ways to organise components of a distributed system, classified as *software architectures*
- Many instantiations where components have their actual placed in a distributed system—often called *system architectures*

Architectural Style

An architectural style is formulated in terms of...

- components
- the way in which components are connected to each other
- the data flowing through the components
- the way in which all the above things are configured altogether to build the system

The notion of architectural style...

- encompasses a way to cluster and classify groups of similar systems, that is, having the same sort of organisation
- allow distributed systems to be compared
- but also provide general patterns for their overall design

Components & Connectors I

Components

- A component is a modular unit with well-defined *interfaces*
- which is *replaceable* within its environment
- interfaces are both *required* and *provided*—both ways, then

Connectors

- A connector is an abstraction *mediating* communication, coordination, cooperation among components
- that is, anything providing a *mechanism for interaction* among components

Components & Connectors II

Putting together components and connectors

- . . . produces a huge range of possible organisations and configurations
- that are then classified in terms of architectural styles

Architectural Styles for Distributed Systems

Identification of architectural styles

- Architectural styles – like patterns in software engineering – are to be *devised out* rather than invented
- Today, four different architectural styles have been identified as the main ones for distributed systems

The main architectural styles for distributed systems

- *Layered* architectures
- *Object-based* architectures
- *Data-centered* architectures
- *Event-based* architectures

Layered Architectures

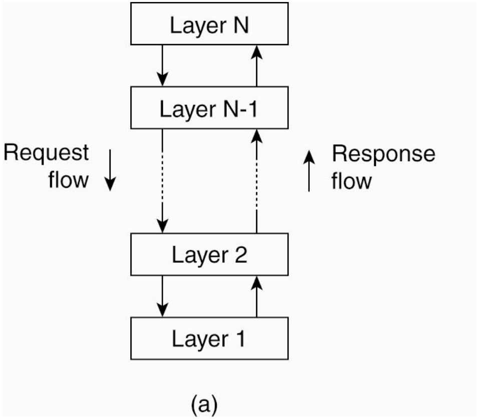
Basic idea

- Components are organised in a *layered fashion*
- where components of a layer *only* call components of the layer below, and are *only* called by the components of the layer above

Data flow

- The request-response flow is always top-down / bottom-up
- Control flow follow the same pattern along with data

Layered Architecture Style



Object-based Architectures

Basic idea

- Components are objects
- Components are connected through a RPC mechanism

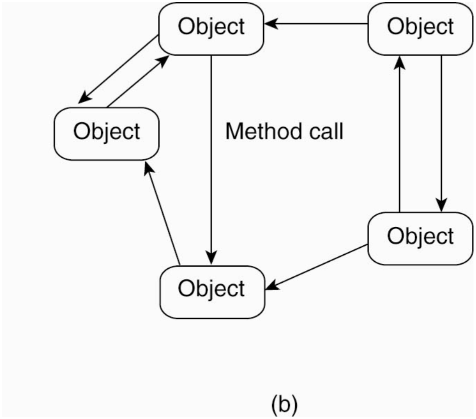
Client-server architectures

- . . . are built out of this style

Layered and object-based architectures

- are the most important styles for distributed systems today
- However, a lot of things are going to happen in the future, which may change such an overall picture

Object-based Architecture Style



Data-centred Architectures I

Basic idea

- Communication among processes occurs through a shared repository
- The repository might be either passive (reactive) or (pro)active

Main features

- . . . depends on the choice made for the shared repository
- how information is represented
- how events are handled
- how the shared repository behave in response to interaction
- how processes interact with / through the shared repository

Data-centred Architectures II

Examples are everywhere

- Web-based systems, for instance, are largely data-centric
- Also, many distributed applications still work by sharing files around the network

Event-based Architectures

Basic idea

- Processes communicate through an event bus
- through which events are propagated
- possibly carrying data along

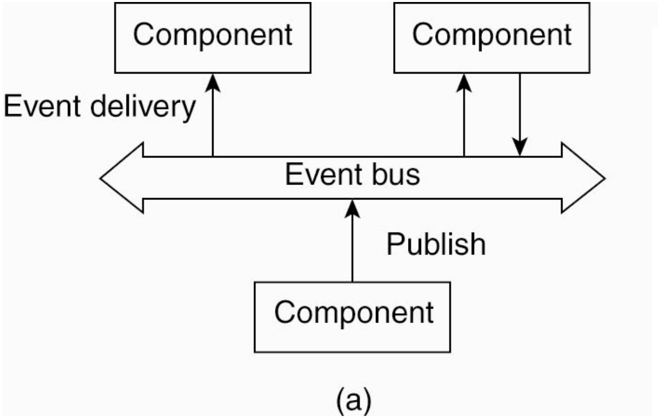
Main example: Publish / subscribe systems

- Publishers publish events through the middleware
- Subscribers receive events to which they have subscribed

Main feature

- Processes can communicate with no need of reference each other / to know each other, they are *referentially decoupled*
- Processes can communicate with no need to share the same space, they are *decoupled in space*

Event-based Architecture Style



Shared Data-space Architectures I

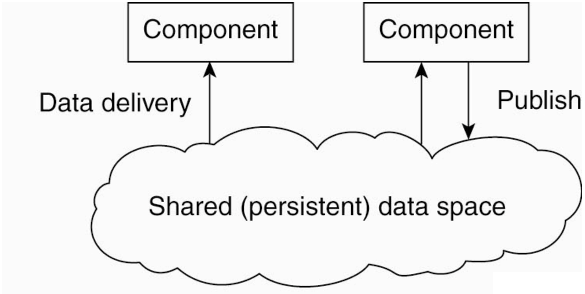
Basic idea

- Putting together Data-centric and Event-based architectures
- The shared repository is a shared persistent data-space, and also an event bus
- where data is stored and accessed
- along with related events

Main example: Blackboard systems

- Processes put data in the blackboard
- The blackboard aggregates knowledge, implements policies and drive the coordination of processes

Shared Data-space Architecture Style



(b)

Architectural Styles for Distributed Systems

Identification of architectural styles

- Architectural styles – like patterns in software engineering – are to be *devised out* rather than invented
- Today, four different architectural styles have been identified as the main ones for distributed systems

The main architectural styles for distributed systems

- *Layered* architectures
- *Object-based* architectures
- *Data-centered* architectures
- *Event-based* architectures

Clients & Servers

Main feature

- In a centralised architecture, *clients* request *services* from *servers*—and that is all, more or less
- In the basic client-server model, processes are classified in two groups—obviously, clients and servers
- Possibly, the two groups may overlap

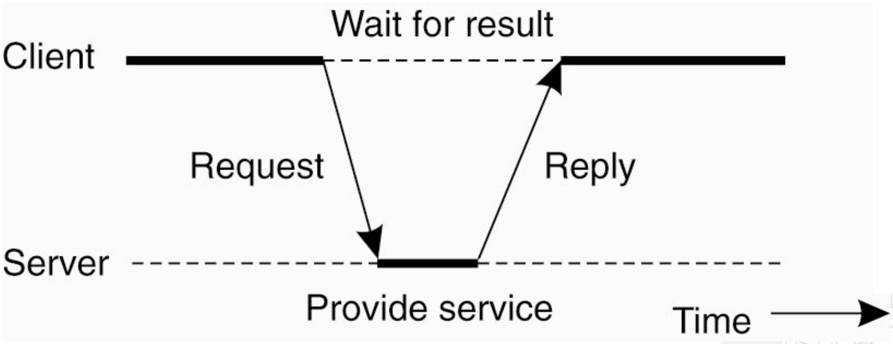
Servers

A server is a process implementing a specific service—like, say, a database service

Clients

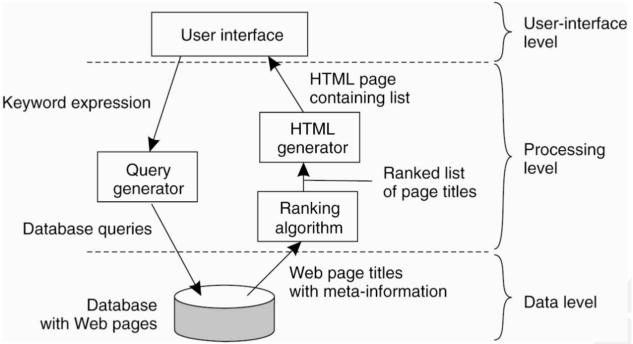
A client is a process requiring a specific service from a server

Client-server Interaction



Scheme of client-server interaction: *request-reply behaviour*

Example: Internet Search Engine



The simplified organisation of an Internet search engine into three different layers

References I

Ozalp Babaoglu, Márk Jelasity, Alberto Montresor, Christof Fetzer, Stefano Leonardi, Aad van Moorsel, and Maarten van Steen, editors.
Self-star Properties in Complex Information Systems: Conceptual and Practical Foundations, volume 3460 of *Lecture Notes in Computer Science*.
Springer, 2005.

Jeffrey O. Kephart and David M. Chess.
The vision of autonomic computing.
IEEE Computer, 36(1):41–50, January 2003.

Philip K. McKinley, Seyed Masoud Sadjadi, Eric P. Kasten, and Betty H.C. Cheng.
Composing adaptive software.
IEEE Computer, 37(7):56–64, 2004.

References II

Andrew S. Tanenbaum and Marteen van Steen.
Distributed Systems. Principles and Paradigms.
Pearson Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2007.