Distributed Systems
Principles and Paradigms

Maarten van Steen

VU Amsterdam, Dept. Computer Science
Room R4.20, steen@cs.vu.nl

Chapter 02: Architectures

Version: February 21, 2011

vrije Universiteit amsterdam

Contents

2 / 25

Architectures

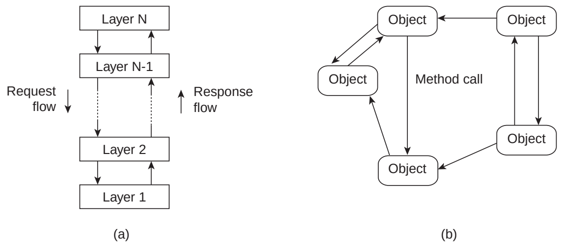
- Architectural styles
- Software architectures
- Architectures versus middleware
- Self-management in distributed systems

3 / 25

Architectural styles

Basic idea

Organize into logically different components, and distribute those components over the various machines.

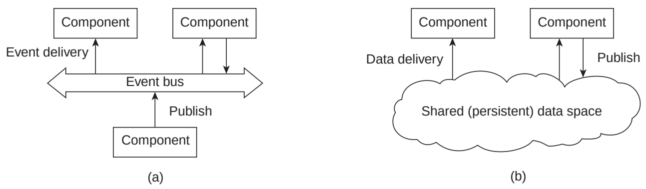


(a) Layered style is used for client-server system
(b) Object-based style for distributed object systems.

Architectural Styles

Observation

Decoupling processes in space (“anonymous”) and also time (“asynchronous”) has led to alternative styles.



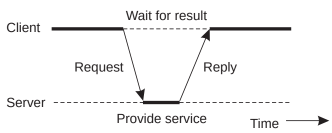
(a) Publish/subscribe [decoupled in space]
(b) Shared dataspace [decoupled in space and time]

Centralized Architectures

Basic Client–Server Model

Characteristics:

- There are processes offering services (servers)
- There are processes that use services (clients)
- Clients and servers can be on different machines
- Clients follow request/reply model wrt to using services



Application Layering

Traditional three-layered view

- User-interface layer contains units for an application's user interface
- Processing layer contains the functions of an application, i.e. without specific data
- Data layer contains the data that a client wants to manipulate through the application components

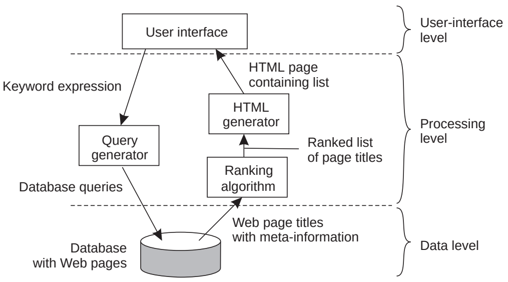
Observation

This layering is found in many distributed information systems, using traditional database technology and accompanying applications.

Architectures 2.2 System Architectures

7 / 25

Application Layering



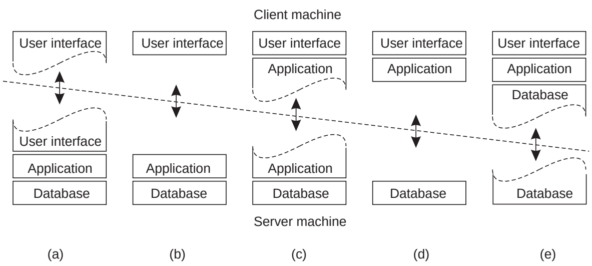
Architectures 2.2 System Architectures

8 / 25

Multi-Tiered Architectures

Single-tiered: dumb terminal/mainframe configuration
Two-tiered: client/single server configuration
Three-tiered: each layer on separate machine

Traditional two-tiered configurations:



Architectures 2.2 System Architectures

9 / 25

Decentralized Architectures

Observation

In the last couple of years we have been seeing a tremendous growth in peer-to-peer systems.

- Structured P2P: nodes are organized following a specific distributed data structure
- Unstructured P2P: nodes have randomly selected neighbors
- Hybrid P2P: some nodes are appointed special functions in a well-organized fashion

Note

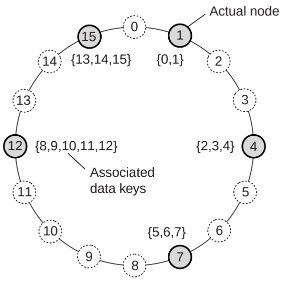
In virtually all cases, we are dealing with overlay networks: data is routed over connections setup between the nodes (cf. application-level multicasting)

Architectures 2.2 System Architectures

Structured P2P Systems

Basic idea

Organize the nodes in a structured overlay network such as a logical ring, and make specific nodes responsible for services based only on their ID.



Note

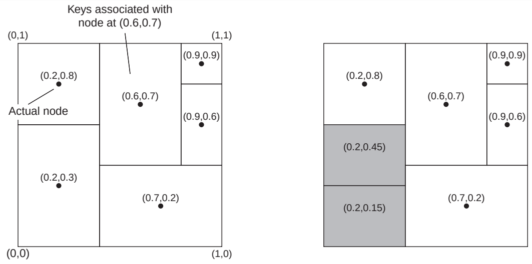
The system provides an operation *LOOKUP(key)* that will efficiently route the lookup request to the associated node.

Architectures 2.2 System Architectures

Structured P2P Systems

Other example

Organize nodes in a *d*-dimensional space and let every node take the responsibility for data in a specific region. When a node joins ⇒ split a region.



(a) (b)

Architectures 2.2 System Architectures

Unstructured P2P Systems

Observation

Many unstructured P2P systems attempt to maintain a random graph.

Basic principle

Each node is required to contact a randomly selected other node:

- Let each peer maintain a partial view of the network, consisting of c other nodes
- Each node P periodically selects a node Q from its partial view
- P and Q exchange information **and** exchange members from their respective partial views

Note

It turns out that, depending on the exchange, randomness, but also robustness of the network can be maintained.

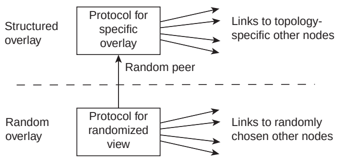
Architectures 2.2 System Architectures

13 / 25

Topology Management of Overlay Networks

Basic idea

Distinguish two layers: (1) maintain random partial views in lowest layer; (2) be selective on who you keep in higher-layer partial view.



Note

Lower layer feeds upper layer with random nodes; upper layer is selective when it comes to keeping references.

Architectures 2.2 System Architectures

14 / 25

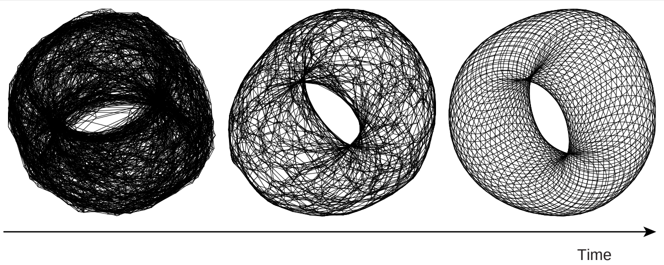
Topology Management of Overlay Networks

Constructing a torus

Consider a $N \times N$ grid. Keep only references to nearest neighbors:

$$\| (a_1, a_2) - (b_1, b_2) \| = d_1 + d_2$$

$$d_i = \min\{N - |a_i - b_i|, |a_i - b_i|\}$$



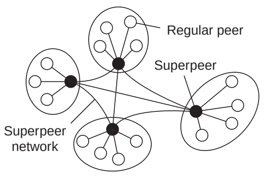
Architectures 2.2 System Architectures

15 / 25

Superpeers

Observation

Sometimes it helps to select a few nodes to do specific work: superpeer.



Examples

- Peers maintaining an index (for search)
- Peers monitoring the state of the network
- Peers being able to setup connections

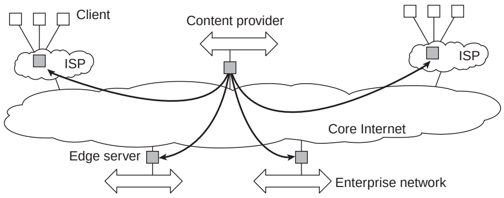
Architectures 2.2 System Architectures

16 / 25

Hybrid Architectures: Client-server combined with P2P

Example

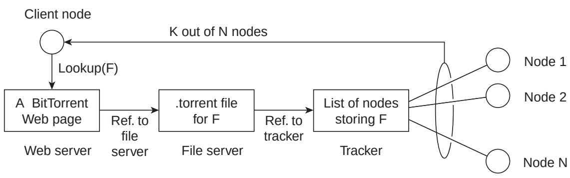
Edge-server architectures, which are often used for Content Delivery Networks



Architectures 2.2 System Architectures

17 / 25

Hybrid Architectures: C/S with P2P – BitTorrent



Basic idea

Once a node has identified where to download a file from, it joins a swarm of downloaders who in parallel get file chunks from the source, but also distribute these chunks amongst each other.

Architectures 2.2 System Architectures

18 / 25

Architectures 2.3 Architectures versus Middleware

Architectures versus Middleware

Problem

In many cases, distributed systems/applications are developed according to a specific architectural style. The chosen style may not be optimal in all cases ⇒ need to (dynamically) adapt the behavior of the middleware.

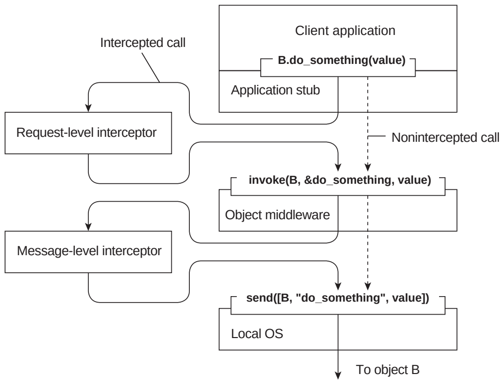
Interceptors

Intercept the usual flow of control when invoking a remote object.

19 / 25

Architectures 2.3 Architectures versus Middleware

Interceptors



20 / 25

Architectures 2.3 Architectures versus Middleware

Adaptive Middleware

Separation of concerns: Try to separate extra functionalities and later weave them together into a single implementation ⇒ only toy examples so far.

Computational reflection: Let a program inspect itself at runtime and adapt/change its settings dynamically if necessary ⇒ mostly at language level and applicability unclear.

Component-based design: Organize a distributed application through components that can be dynamically replaced when needed ⇒ highly complex, also many intercomponent dependencies.

Fundamental question

Do we need adaptive software at all, or is the issue adaptive systems?

21 / 25

Self-managing Distributed Systems

Observation

Distinction between system and software architectures blurs when automatic adaptivity needs to be taken into account:

- Self-configuration
- Self-managing
- Self-healing
- Self-optimizing
- Self-*

Warning

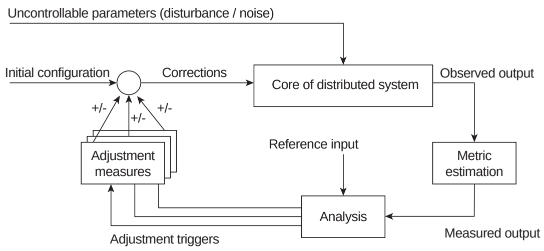
There is a lot of hype going on in this field of autonomic computing.

Architectures 2.4 Self-management in Distributed Systems

Feedback Control Model

Observation

In many cases, self-* systems are organized as a feedback control system.



Architectures 2.4 Self-management in Distributed Systems

Example: Globule

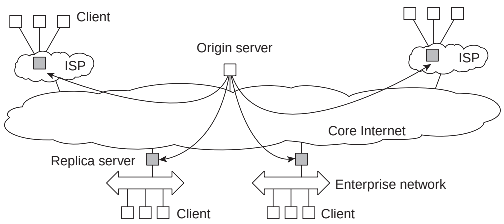
Globule

Collaborative CDN that analyzes traces to decide where replicas of Web content should be placed. Decisions are driven by a general cost model:

$$cost = (w_1 \times m_1) + (w_2 \times m_2) + \cdots + (w_n \times m_n)$$

Architectures 2.4 Self-management in Distributed Systems

Example: Globule



- Globule origin server collects traces and does what-if analysis by checking what would have happened if page *P* would have been placed at edge server *S*.
- Many strategies are evaluated, and the best one is chosen.

Architectures 2.4 Self-management in Distributed Systems

25 / 25