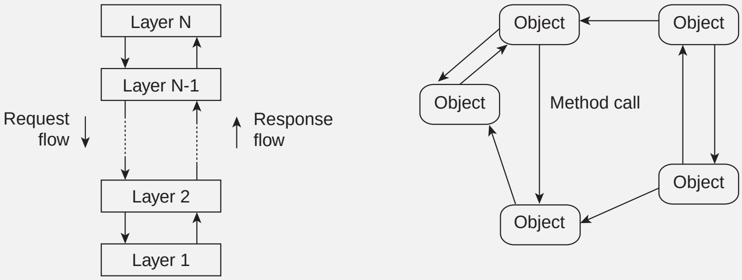
Architectural styles

Basic idea

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



(a)

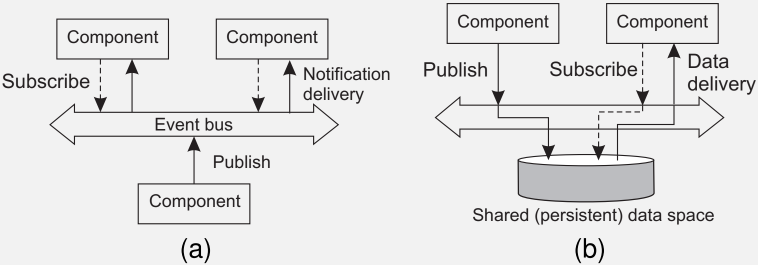
(b)

(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.



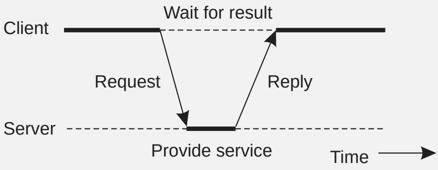
(a) Publish/subscribe [decoupled in space]
(b) Shared dataspace [decoupled in space and time]

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.

Application Layering

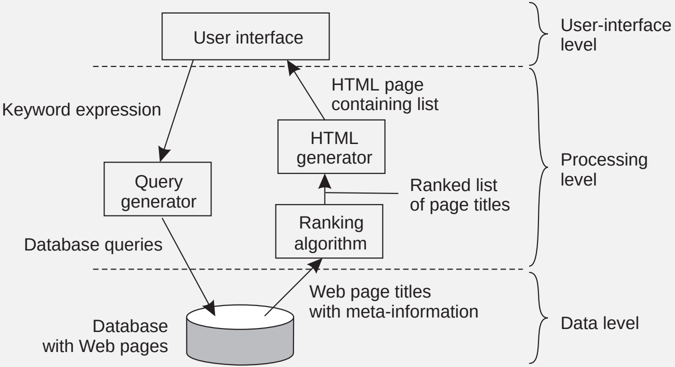
Traditional three-layered view

- 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.

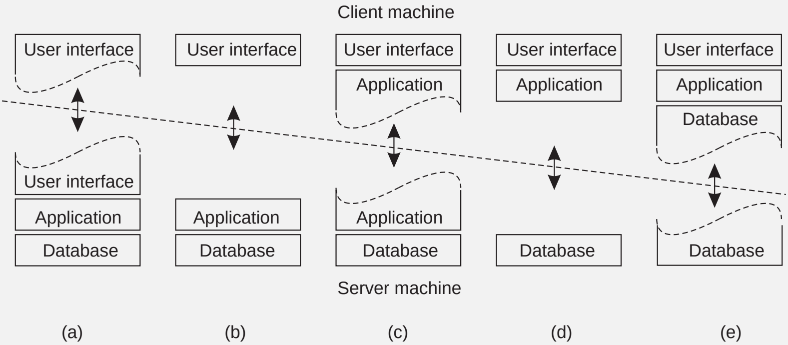
Application Layering



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:



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)

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

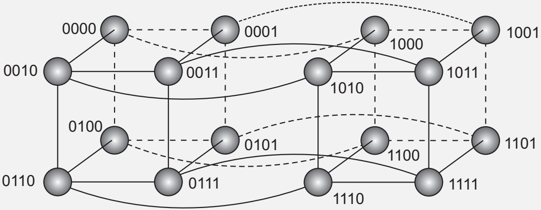
Note

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

Structured P2P Systems

Basic idea

Organize the nodes in a structured overlay network such as a logical ring, or a hypercube, 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.

Unstructured P2P Systems

Essence

Many unstructured P2P systems are organized as a random overlay: two nodes are linked with probability p.

Observation

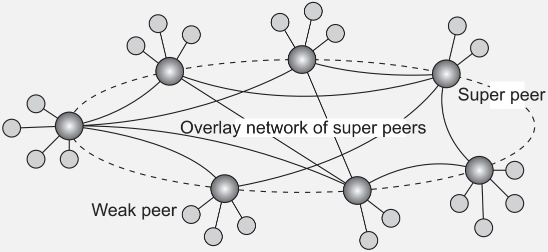
We can no longer look up information deterministically, but will have to resort to searching:

- Flooding: node u sends a lookup query to all of its neighbors. A neighbor responds, or forwards (floods) the request. There are many variations:
  - Limited flooding (maximal number of forwarding)
  - Probabilistic flooding (flood only with a certain probability).
- Random walk: Randomly select a neighbor v. If v has the answer, it replies, otherwise v randomly selects one of *its* neighbors. Variation: parallel random walk. Works well with replicated data.

Superpeers

Observation

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



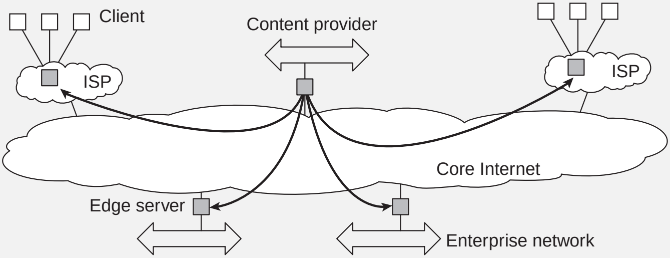
Examples

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

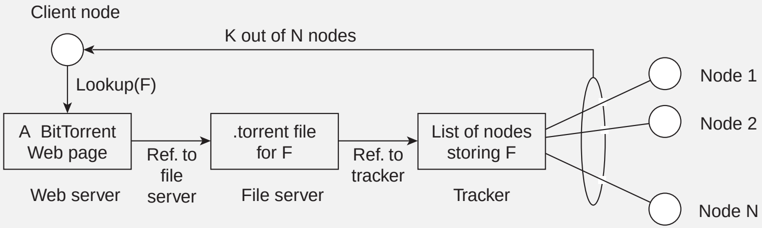
Hybrid Architectures: Client-server combined with P2P

Example

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



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 versus Middleware

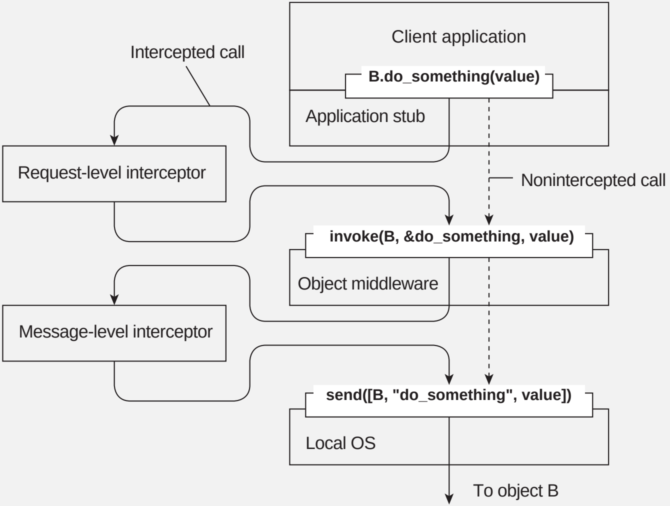
Problem

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

Interceptors

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

Interceptors



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?

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?

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?

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?

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?

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.

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.

Feedback Control Model

Observation

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