Distributed Systems

(3rd Edition)

Chapter 02: Architectures

Architectural styles

Architecture: How the components are logically organized and communicate with each other

Basic idea

A style is formulated in terms of

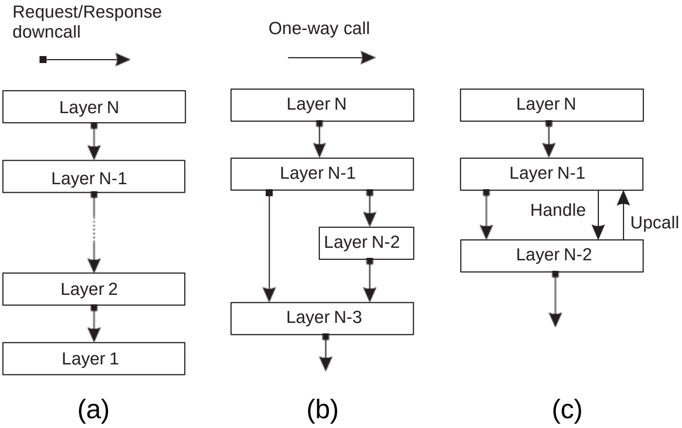
- (replaceable) components with well-defined interfaces
- the way that components are connected to each other
- the data exchanged between components
- how these components and connectors are jointly configured into a system.

Connector

A mechanism that mediates communication, coordination, or cooperation among components. Example: facilities for (remote) procedure call, messaging, or streaming.

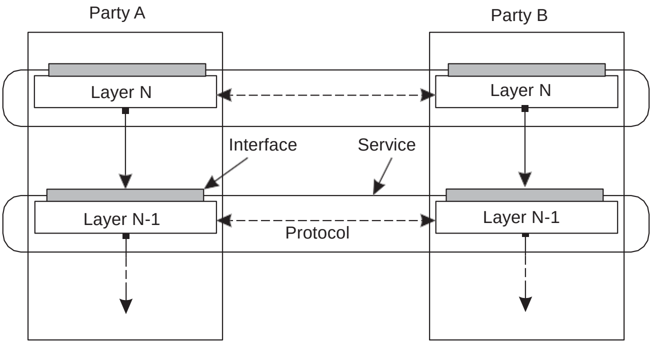
Layered architecture

Different layered organizations



Example: communication protocols

Protocol, service, interface



Application Layering

Traditional three-layered view

- Application-interface layer contains units for interfacing to users or external applications
- 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

Application Layering

Traditional three-layered view

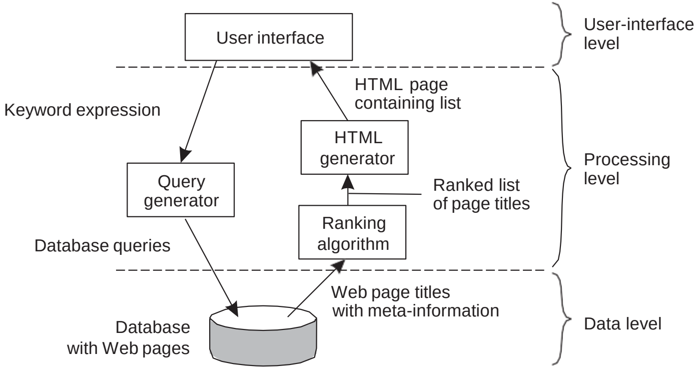
- Application-interface layer contains units for interfacing to users or external applications
- Processing layer contains the functions of an application, i.e., without specific data
- Data layer contains the data that a client wants to manipulate through the application components

Observation

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

Application Layering

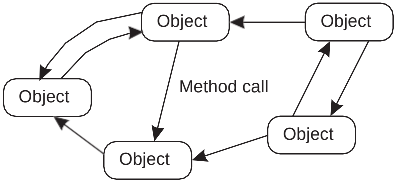
Example: a simple search engine

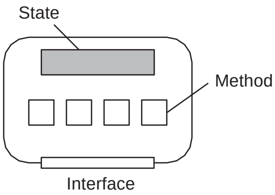


Object-based style

Essence

Components are objects, connected to each other through procedure calls. Objects may be placed on different machines; calls can thus execute across a network.





Encapsulation

Objects are said to encapsulate data and offer methods on that data without revealing the internal implementation.

RESTful architectures- Resource based architecture

Representational State Transfer - REST

Essence

View a distributed system as a collection of resources, individually managed by components. Resources may be added, removed, retrieved, and modified by (remote) applications.

1. Resources are identified through a single naming scheme
2. All services offer the same interface
3. Messages sent to or from a service are fully self-described
4. After executing an operation at a service, that component forgets everything about the caller

Basic operations

Operation	Description
PUT	Create a new resource
GET	Retrieve the state of a resource in some representation
DELETE	Delete a resource
POST	Modify a resource by transferring a new state

Example: Amazon's Simple Storage Service

Essence

Objects (i.e., files) are placed into buckets (i.e., directories). Buckets cannot be placed into buckets. Operations on ObjectName in bucket BucketName require the following identifier:

http://BucketName.s3.amazonaws.com/ObjectName

Typical operations

All operations are carried out by sending HTTP requests:

- Create a bucket/object: PUT, along with the URI
- Listing objects: GET on a bucket name
- Reading an object: GET on a full URI

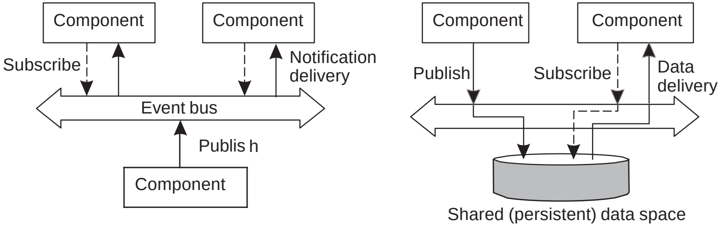
Publish-subscribe architectures

Coordination

- Temporal and referential coupling
- Referential – explicit reference to communication
- Temporal – Both process has to be up and running

	Temporally coupled	Temporally decoupled
Referentially coupled	Direct (cellphone)	Mailbox
Referentially decoupled	Event- based	Shared data space

Event-based and Shared data space

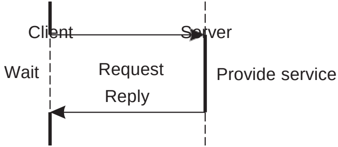


Distributed System Architectures
Centralized system architectures

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 with respect to using services

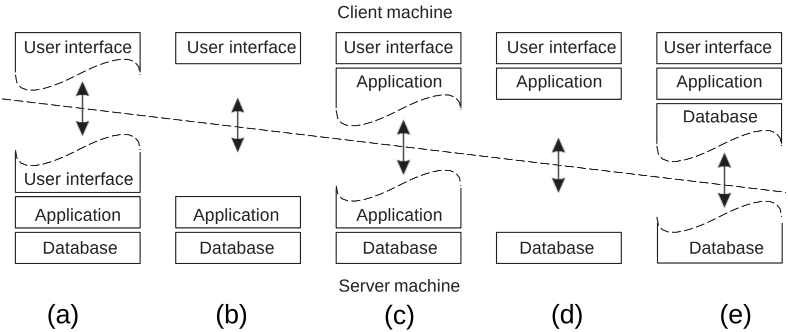


Multi-tiered centralized system architectures

Some traditional organizations

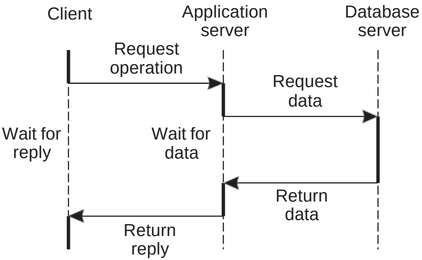
- Single-tiered: dumb terminal/mainframe configuration
- Two-tiered: client/single server configuration
- Three-tiered: each layer on separate machine

Traditional two-tiered configurations



Being client and server at the same time

Three-tiered architecture



Decentralized - Peer-to-Peer architectures

Processes are all equal: -the functions that need to be carried out are represented by every process.

Each process will act as a client and a server at the same time (i.e., acting as a servant).

Structured P2P

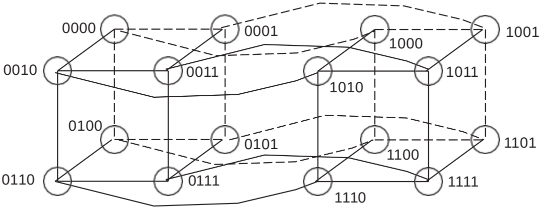
Essence

Make use of a semantic-free index: each data item is uniquely associated with a key, in turn used as an index. Common practice: use a hash function

$$key(data\ item) = hash(data\ item\text{'s } value).$$

P2P system now responsible for storing (*key*,*value*) pairs.

Simple example: hypercube



Looking up d with key $k \in \{0,1,2,\ldots,2^4 - 1\}$ means routing request to node with identifier k.

Unstructured P2P

Essence

Each node maintains an ad hoc list of neighbors. The resulting overlay resembles a random graph:

Searching

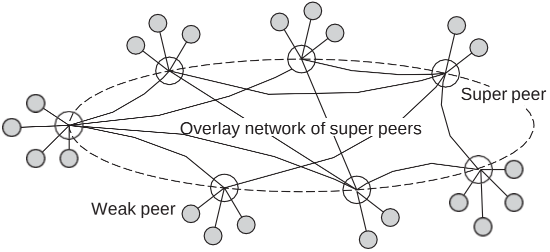
- Flooding: issuing node u passes request for d to all neighbors. Request is ignored when receiving node had seen it before. Otherwise, v searches locally for d (recursively). May be limited by a Time-To-Live: a maximum number of hops.
- Random walk: issuing node u passes request for d to randomly chosen neighbor, v. If v does not have d, it forwards request to one of *its* randomly chosen neighbors, and so on.

Super-peer networks

Essence

It is sometimes sensible to break the symmetry in pure peer-to-peer networks:

- When searching in unstructured P2P systems, having index servers improves performance
- Deciding where to store data can often be done more efficiently through brokers.



Edge-server architecture – Hybrid architecture

Essence

Systems deployed on the Internet where servers are placed at the edge of the network: the boundary between enterprise networks and the actual Internet.