Introduction to Distributed Programming

Grado en Ingeniería Informática

Pablo García Sánchez

Departamento de Ingeniería Informática
Universidad de Cádiz
Pablo García Sánchez
Based on the work by Guadalupe Ortiz, Mei Ling-Liu, Marteen Van Steem and A. Tanenbaum

Curso 2017 – 2018

Indice

Section 1 | Parallel Vs. Distributed Computing

Concepts

- Parallel Computer
- Parallel Programming
- Distributed Programming
- Different definitions depending on the context

Parallel Computing

- Generally concerned with accomplishing a particular computation as fast as possible, exploiting multiple processors.
- Related to tightly-coupled applications.
- Goals:
  - Solve compute-intensive problems faster;
  - Solve larger problems in the same amount of time;
  - Solve same size problems with higher accuracy in the same amount of time.
- May use shared-memory, message-passing or both (OpenMP with MPI); or even GPUs accelerators.
- Do not take into account issues such as failures, network partition, etc.

Distributed Computing

- Related to loosely-coupled applications.
- Goals (for distributed supercomputing):
  - To solve problems otherwise too large or whose execution may be divided on different components ...
  - ...that could benefit from execution on different architectures.
- Models: client-server, peer-to-peer, etc.
- Security, failures, network partition etc. must be taken into account at design time.

Why distributed computing?

- Economics: distributed systems allow the pooling of resources, including CPU cycles, data storage, input/output devices, and services.
- Reliability: a distributed system allow replication of resources and/or services, thus reducing service outage due to failures.
- The Internet has become a universal platform for distributed computing.

Strengths of Distributed Computing

- The affordability of computers and availability of network access
- Resource sharing
- Scalability
- Fault Tolerance

Weaknesses of Distributed Computing

- Multiple Points of Failures: the failure of one or more participating computers, or one or more network links, can spell trouble.
- Security Concerns: In a distributed system, there are more opportunities for unauthorized attack.

Evolution of paradigms

- Client-server: Socket API, remote method invocation
- Distributed objects
- Object broker: CORBA
- Message oriented middleware (MOM): Java Message Service
- Service-oriented: SOAP Web Services/Restful Services, request/response
- Message Broker: Queues and topics, publish/subscribe
- Event-driven: publish/subscribe

Section 2 | Architectural styles

Architectural styles

Basic idea

A style is formulated in terms of

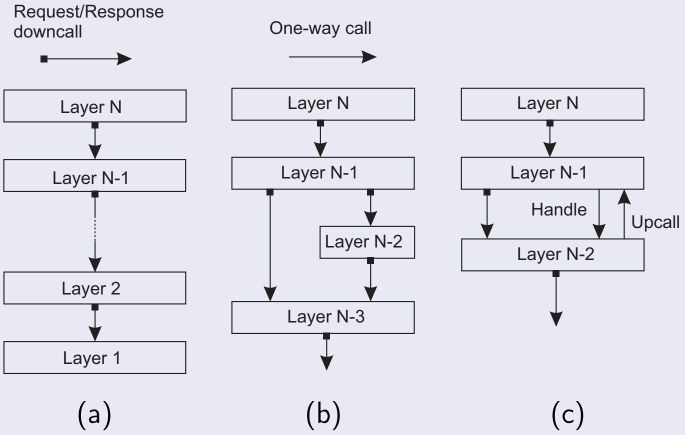
- (replaceable) components with well-defined interfaces
- the way that components are connected to each other
- the data exchanged between components
- how these components and connectors are jointly configured into a system.

Connector

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

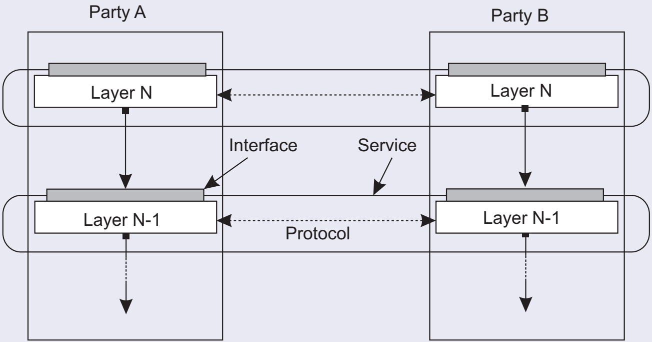
Layered architecture

Different layered organizations



Example: communication protocols

Protocol, service, interface



Two-party communication

Server

```
from socket  import *
s = socket(AF_INET, SOCK_STREAM)
(conn, addr) = s.accept()  # returns new socket and
while True:                # forever
  data = conn.recv(1024)   # receive data from clie
  if not data: break       # stop if client stopped
  conn.send(str(data)+"*") # return sent data plus
conn.close()               # close the connection
```

Client

```
from socket  import *
s = socket(AF_INET, SOCK_STREAM)
s.connect((HOST, PORT)) # connect to server (block
s.send('Hello, world')  # send some data
```

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

Traditional three-layered view

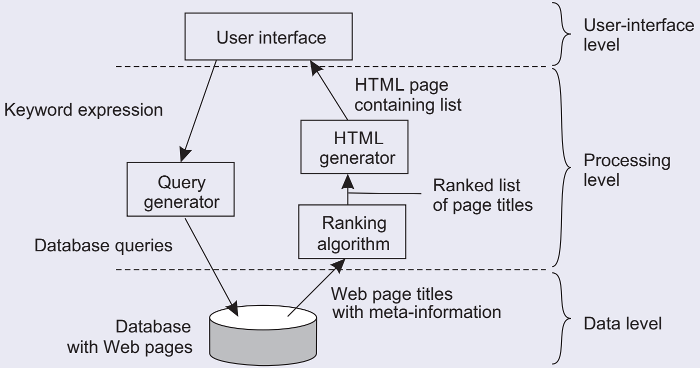
- Application-interface layer contains units for interfacing to users or external applications
- Processing layer contains the functions of an application, i.e., without specific data
- Data layer contains the data that a client wants to manipulate through the application components

Observation

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

Application Layering

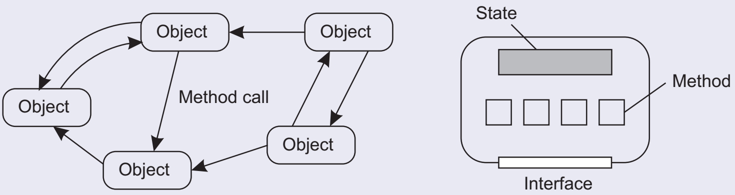
Example: a simple search engine



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

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

On interfaces

Issue

Many people like RESTful approaches because the interface to a service is so simple. The catch is that much needs to be done in the parameter space.

Amazon S3 SOAP interface

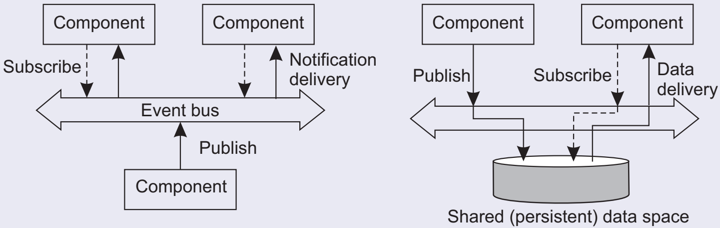
Bucket operations	Object operations
ListAllMyBuckets	PutObjectInline
CreateBucket	PutObject
DeleteBucket	CopyObject
ListBucket	GetObject
GetBucketAccessControlPolicy	GetObjectExtended
SetBucketAccessControlPolicy	DeleteObject
GetBucketLoggingStatus	GetObjectAccessControlPolicy
SetBucketLoggingStatus	SetObjectAccessControlPolicy

Coordination

Temporal and referential coupling

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

Event-based and Shared data space



Section 3 | Middleware organization

Using legacy to build middleware

Problem

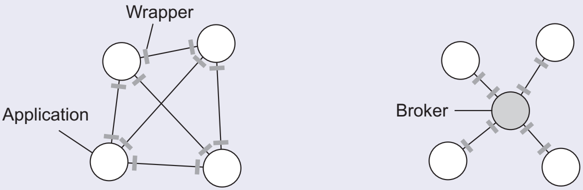
The interfaces offered by a legacy component are most likely not suitable for all applications.

Solution

A wrapper or adapter offers an interface acceptable to a client application. Its functions are transformed into those available at the component.

Organizing wrappers

Two solutions: 1-on-1 or through a broker



Complexity with N applications

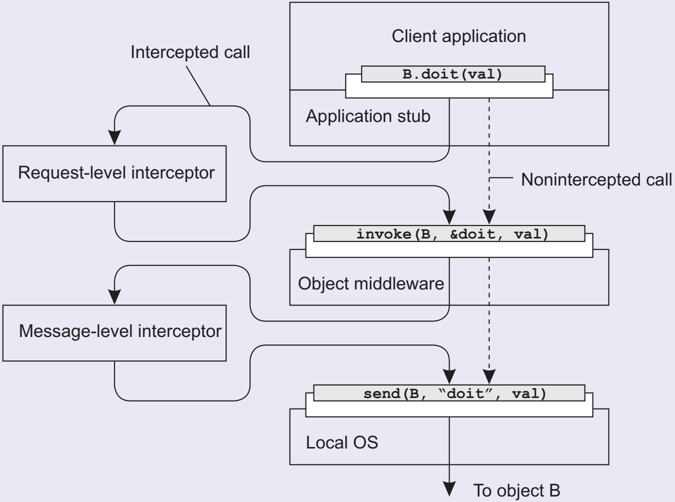
- 1-on-1: requires $N \times (N-1) = \mathcal{O}(N^2)$ wrappers
- broker: requires $2N = \mathcal{O}(N)$ wrappers

Developing adaptable middleware

Problem

Middleware contains solutions that are good for most applications ⇒ you may want to adapt its behavior for specific applications.

Intercept the usual flow of control



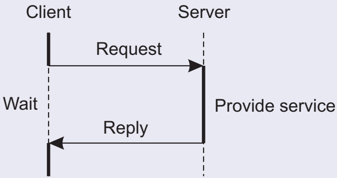
Section 4 | System architecture

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

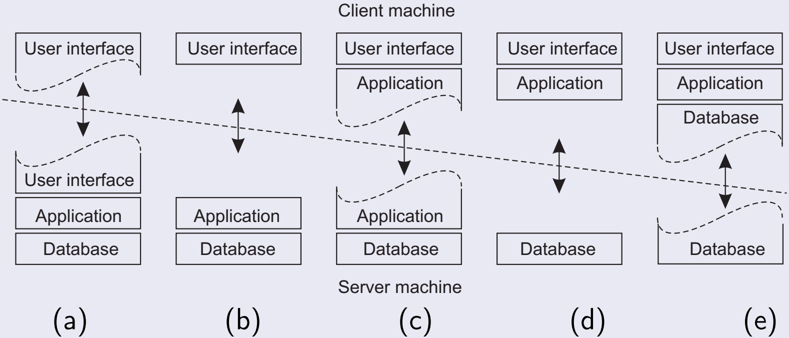


Multi-tiered centralized system architectures

Some traditional organizations

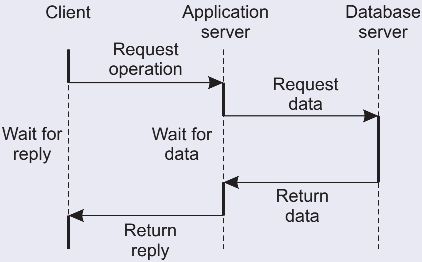
- Single-tiered: dumb terminal/mainframe configuration
- Two-tiered: client/single server configuration
- Three-tiered: each layer on separate machine

Traditional two-tiered configurations



Being client and server at the same time

Three-tiered architecture



Alternative organizations

Vertical distribution

Comes from dividing distributed applications into three logical layers, and running the components from each layer on a different server (machine).

Horizontal distribution

A client or server may be physically split up into logically equivalent parts, but each part is operating on its own share of the complete data set.

Peer-to-peer architectures

Processes are all equal: the functions that need to be carried out are represented by every process $\Rightarrow$ each process will act as a client and a server at the same time (i.e., acting as a servant).

References

- Distributed Systems. Marteen Van Steen and Andrew Tanenbaum (2017).
- Mei Ling-Liu. Distributed Computing Algorithms.
- Classroom notes by Guadalupe Ortiz