Prof. Dr. Claudia Müller-Birn
Institute for Computer Science, Networked Information Systems

Our course at a glance

February 7, 2012

Netzprogrammierung
(Algorithmen und Programmierung V)

Scope of this course

„In this class you will learn about principles, methods, languages and middleware for developing distributed systems, especially web-based applications.“

We will not talk about

- Theory of computer networks
- Telematics
- Theory of distributed systems
- Design of distributed algorithms
- Design of distributed databases

Goal of this course

At the end of this course, you should be able to

- Differentiate relevant interaction paradigms such as client/server or peer-to-peer
- Knowing the different levels of support for distributed computing
- Develop distributed software based on local inter-process communication (remote procedure calls) as well as socket-based network communication
- Implement distributed software based on Java RMI
- Knowing middleware technologies and understanding their differences
- Describe the main design principles of cloud computing and its application areas
- Development of web-based, distributed software based on relevant standards

Grading

Your final grade is only based on the result of your written exam.

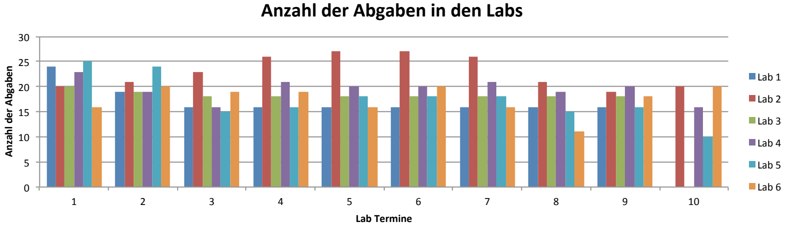
But

In order to actively participate in this course, you need to fulfill **ALL** of the following requirements

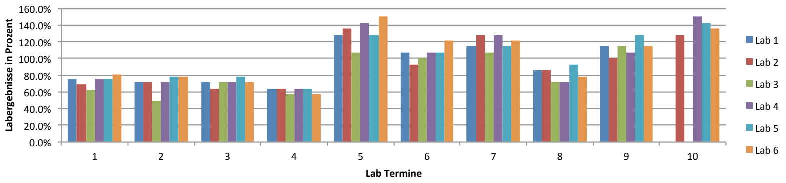
- you have to submit (n-2) of all assignments that are distributed in the labs,
- you need to get at least 50 % of all points in each assignment,
- you must present at least one assignment,
- the mean (= average) of all your assignments need to be above 60 %.

Lab results

Anzahl der Abgaben in den Labs



Durchschnittlich erreichte Punktzahl in Prozent

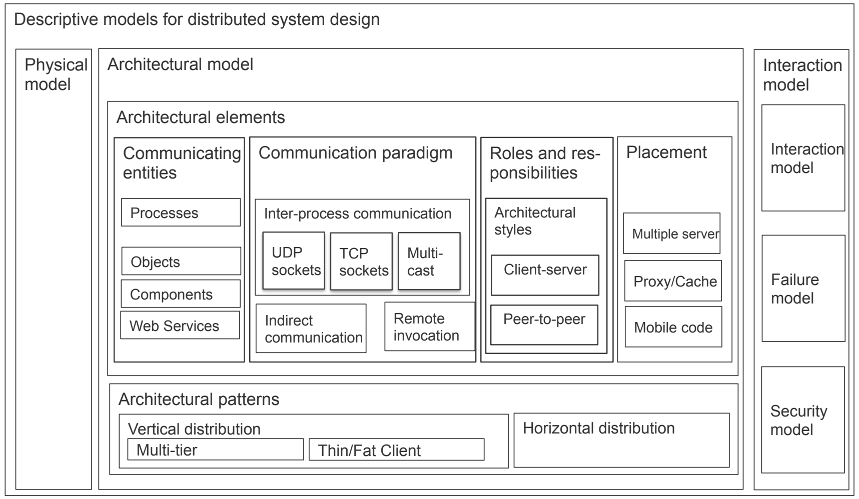


Defining a distributed system

"A distributed system consists of a collection of autonomous computer linked by a computer network and equipped with distributed system software. Distributed system software enables computers to coordinate their activities and to share the resources of the system – hardware, software, and data – " (Coulouris et al., 1994)

"[…] so that users perceive the system as a single, integrated computing facility."

"Most computer software today runs in distributed systems, where the interactive presentation, application business processing, and data resources reside in loosely-coupled computing nodes and service tiers connected together by networks." (Buschmann et al., 2007)



Descriptive models for distributed system design

Physical model

Architectural model

Architectural elements

Communicating entities

Processes

Objects

Components

Web Services

Communication paradigm

Inter-process communication

UDP sockets

TCP sockets

Multi-cast

Indirect communication

Remote invocation

Roles and res-ponsibilities

Architectural styles

Client-server

Peer-to-peer

Placement

Multiple server

Proxy/Cache

Mobile code

Architectural patterns

Vertical distribution

Multi-tier

Thin/Fat Client

Horizontal distribution

Interaction model

Interaction model

Failure model

Security model

Architecture of distributed systems

Three generations of distributed systems

Early distributed systems

- Emerged in the late 1970s and early 1980s because of the usage of local area networking technologies
- System typically consisted of 10 to 100 nodes connected by a LAN, with limited Internet connectivity and supported services (e.g., shared local printer, file servers)

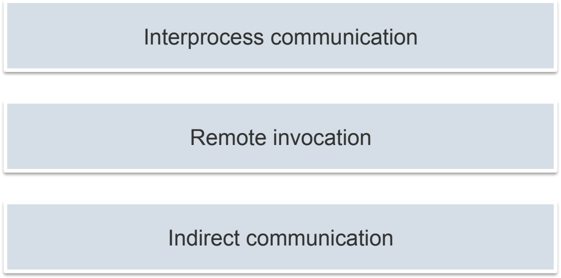
Internet-scale distributed systems

- Emerged in the 1990s because of the growth of the Internet
- Infrastructure became global

Contemporary distributed systems

- Emergence of mobile computing leads to nodes that are location-independent
- Need to added capabilities such as service discovery and support for spontaneous interoperation
- Emergence of cloud computing and ubiquitous computing

Types of communication paradigms


Interprocess communication

Remote invocation

Indirect communication

Interprocess communication

- Low-level support for communication between processes in distributed systems including message parsing-primitives

- Direct access to the API offered by Internet protocols (socket programming) and support for multicast communication

Remote invocation

Covering a range of techniques based on a two-way exchange between communicating entities

Resulting in the calling of a remote operation, procedure or method

- Request-reply protocols: more a pattern imposed on an underlying message-parsing service to support client-server computing

- Remote procedure calls: procedures in processes on remote computers can be called as if they are procedures in the local address space

- Remote method invocation: a calling object can invoke a method in a remote object

Remote invocation

Covering a range of techniques based on a two-way exchange between communicati

Resulting in t

- Reque ing message-parsin
- Remot mputers can be called
- Remot d in a remote object

- Communication represent a two-way relationship between sender and receiver
- Sender explicitly directing messages/invocations to the associated receivers
- Receivers are aware of senders
- Must exist at the same time

Remote invocation

Covering a range of techniques based on a two-way exchange between communicati

Resulting in t

- Reque ing message-parsin
- Remot mputers can be called
- Remot d in a remote object

- Sender do not need to know who they are sending to (space uncoupling)
- Senders and receivers do not need to exist in the same time (time uncoupling)

Indirect communication

Group communication

- Delivery of messages to a set of recipients
- Abstraction of a group which is represented in the system by a group identifier
- Recipients elect to receive message send to a group a joining a group

Publish-subscribe-systems

- A large number of producers (publisher) distribute information items of interest (events) to a similarly large number of consumers (subscribers)

Message queues

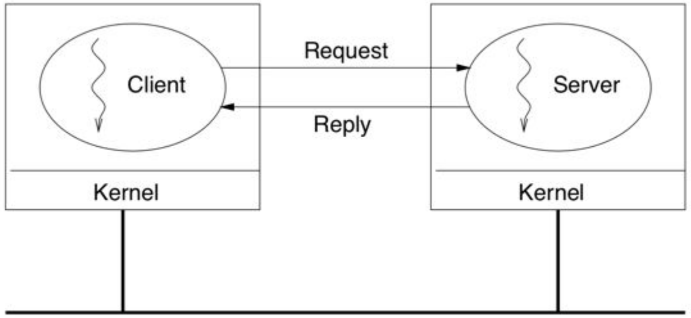
- Message queues offer a point-to-point service whereby producer processes can send messages to a specified queue and consumer processes can receive messages from the queue or being notified

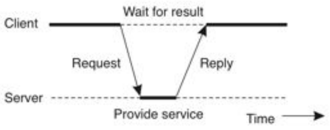
Architectural styles

client-server

peer-to-peer

Client-server





Fundamental issue with client-server

Client-server offers a direct, relatively simple approach to the sharing of data and other resources

➥ But it scales poorly

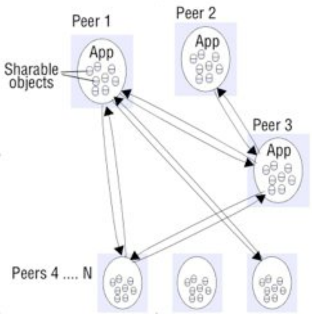
The centralization of service provision and management implied by placing a service at a single address does not scale well beyond the capacity of the computer that hosts the service and the bandwidth of its connections.

Even though, there a several variations of the client-server architecture to respond to this problem but none of them really solve it.

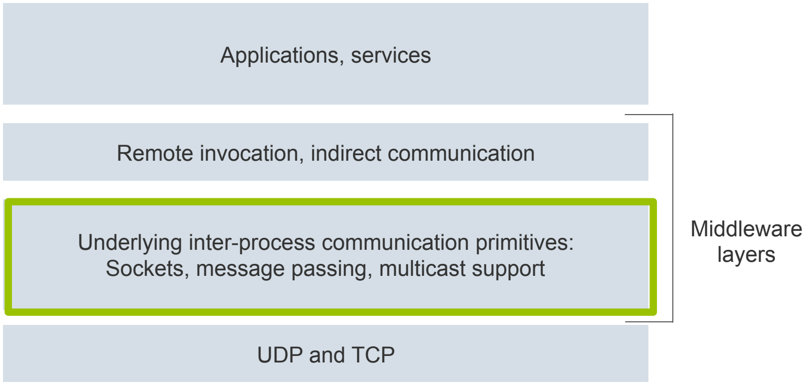
There is a need to distribute shared resources much more widely in order to share the computing and communication loads amongst a much larger number of computers and network links.

Peer-to-peer application

- Is composed of a large number of peer processes running on separate computers
- All processes have client and server roles: servent
- Patterns of communication between them depends entirely on application requirements
- Storage, processing and communication loads for accessing objects are distributed across computers and network links
- Each object is replicated in several computers to further distribute the load and to provide resilience in the event of disconnection of individual computers
- Need to place and retrieve individual computers is more complex then in client-server architecture



Middleware layers



Applications, services

Remote invocation, indirect communication

Underlying inter-process communication primitives:
Sockets, message passing, multicast support

UDP and TCP

Middleware layers

Socket address = IP address and port number

Sockets

- Sockets provide an interface for programming networks at the transport layer.
- Network communication using Sockets is very much similar to performing file I/O
- Socket-based communication is programming language independent.

Ports

- Port is represented by a positive (16-bit) integer value
- Some ports have been reserved to support common/well known services such as ftp (20 for data and 21 control)
- User level process/services generally use port number value >= 1024

Realizing process-to-process communication

UDP features

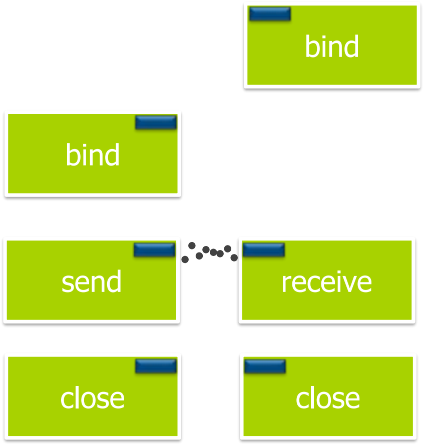
- UDP datagram encapsulated inside an IP package
- Header includes source and destination port numbers
- No guarantee of delivery
- Message size is limited
- Restricted to applications and services that do not require reliable delivery of single or multiple messages

TCP features

- Provides reliable delivery of arbitrarily long sequences of bytes via stream-based programming abstraction
- Connection-oriented service
- Before data is transferred, a bidirectional communication channel is established

UDP Sockets

1. Client creates socket bound to a local port
2. Server binds its socket to a server port
3. Client/Server send and receive datagrams
4. Ports and sockets are closed



bind

bind

send

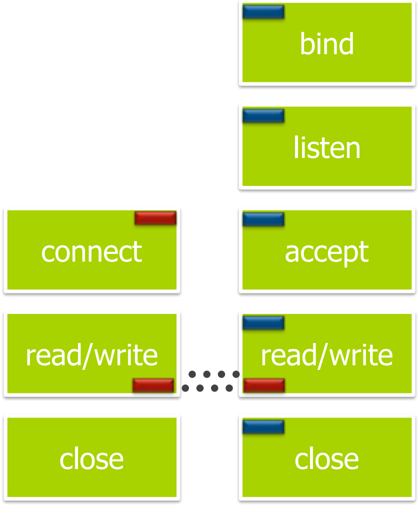
receive

close

close

TCP Sockets

1. Server bind port
2. Server is ready and listening
3. Server is waiting for request, client sends request, server accepts
4. Client and server are conncected - bidirectional!
5. Connection is closed



bind

listen

connect

accept

read/write

read/write

close

close

Approaches for external data representation

CORBA's common data representation

- Concerned with an external representation for the structured and primitive types that can be passed as the arguments and results of remote invocation in CORBA.

Java's object serialization

- Refers to the activity of flattening an object or even a connected set of objects that need to be transmitted or stored on a disk

XML

- Defines a textual format for representing structured data

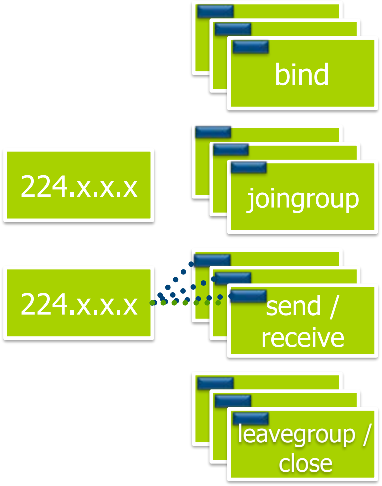
Protocol buffer

JSON

Multicast communication

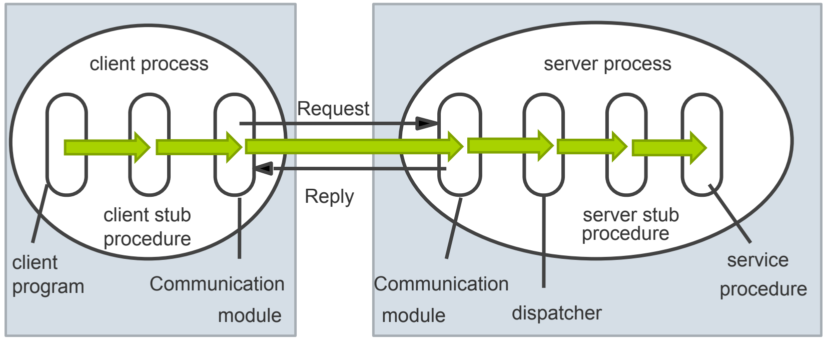
Multicast Sockets

1. Participants bind socket
2. Participants join group
3. Particpants receive messages from sender
4. Partcipants leave group and release socket



Remote invocation (RPC and RMI)

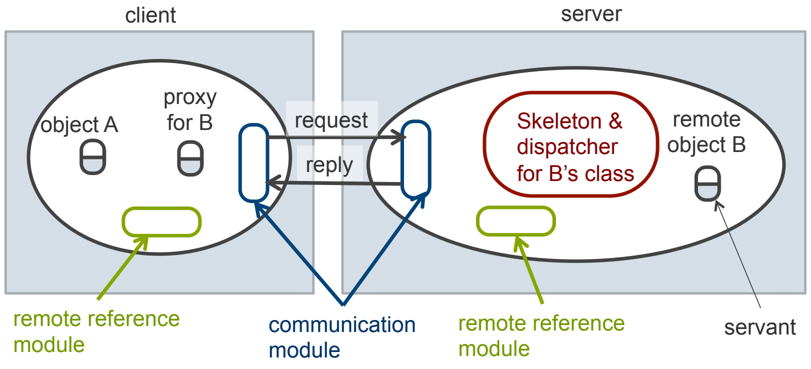
Implementation of RPC



Commonalities of RMI and RPC

- Support of programming languages with interfaces
- Both are typically constructed on top of the request-reply protocol
- Offer call semantics such *as at-least-once* and *at-most-once*
- Offer a similar level of transparency, means local and remote calls employ the same syntax but remote interfaces expose the distributed nature for example by supporting remote exceptions

Components of the RMI architecture



Abstraction layers in the RMI implementation

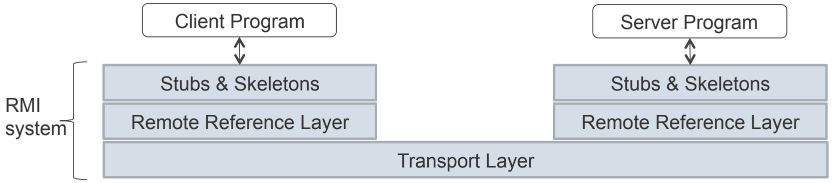
1. **Stub and Skeleton layer**

   Intercepts method calls made by the client to the interface reference variable and redirects these calls to a remote RMI service
2. **Remote Reference Layer**

   Interpret and manage references made from clients to the remote service objects
3. **Transport layer**

   Is based on TCP/IP connections between machines in a network

   Provides basic connectivity, as well as some firewall penetration strategies



Proxy design pattern: Applications

Virtual Proxies: delaying the creation and initialization of expensive objects until needed, where the objects are created on demand.

Remote Proxies: providing a local representation for an object that is in a different address space. A common example is Java RMI stub objects. The stub object acts as a proxy where invoking methods on the stub would cause the stub to communicate and invoke methods on a remote object (called skeleton) found on a different machine.

Protection Proxies: where a proxy controls access to RealSubject methods, by giving access to some objects while denying access to others.

Smart References: providing a sophisticated access to certain objects such as tracking the number of references to an object and denying access if a certain number is reached, as well as loading an object from database into memory on demand.

Reflections

Reflection enables Java code

- to discover information about the fields, methods and constructors of loaded classes, and
- to use reflected fields, methods, and constructors to operate on their underlying counterparts on objects, within security restrictions.

More information: http://download.oracle.com/javase/tutorial/reflect/

Using Reflection in RMI

- Proxy has to marshal information about a method and its arguments into a request.
- For a method it marshals an object of class *Method* into the request. It then adds an array of *objects* for the method's arguments.
- The dispatcher unmarshals the *Method* object and its arguments from request message.
- The remote object reference is obtained from remote reference module.
- The dispatcher then calls the *Method* object's "invoke" method, supplying the target object reference and the array of argument values.
- After the method execution, the dispatcher marshals the result or any exceptions into the reply message.

Remote Reference Layer

Defines and supports the invocation semantics of the RMI connection

Provides a RemoteRef object that represents the link to the remote service implementation object

JDK 1.1 implementation of RMI

- Provides a unicast, point-to-point connection
- Before a client can use a remote service, the remote service must be instantiated on the server and exported to the RMI system

Java 2 SDK implementation of RMI

- When a method call is made to the proxy for an activatable object, RMI determines if the remote service implementation object is dormant
- If yes, RMI will instantiate the object and restore its state from a disk file

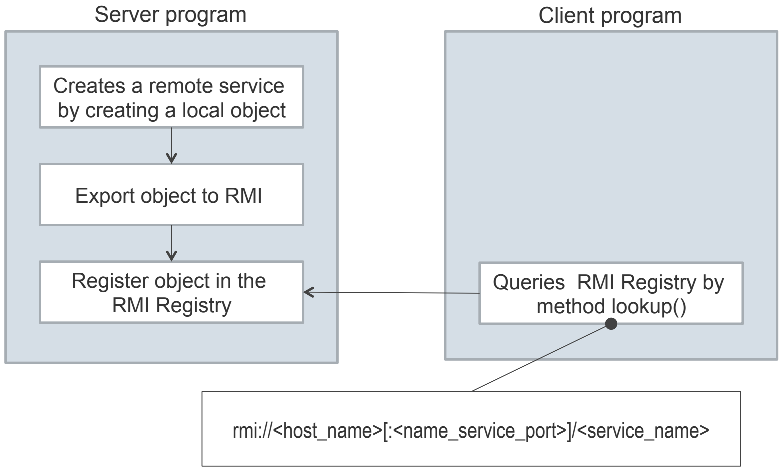
Naming Remote Objects

How does a client find a RMI remote service?

RMI includes a simple service called the RMI Registry, *rmiregistry*.

The RMI Registry runs on each machine that hosts remote service objects and accepts queries for services, by default on port 1099.

Naming Remote Objects (*cont.*)



Server program

Client program

Creates a remote service by creating a local object

Export object to RMI

Register object in the RMI Registry

Queries RMI Registry by method lookup()

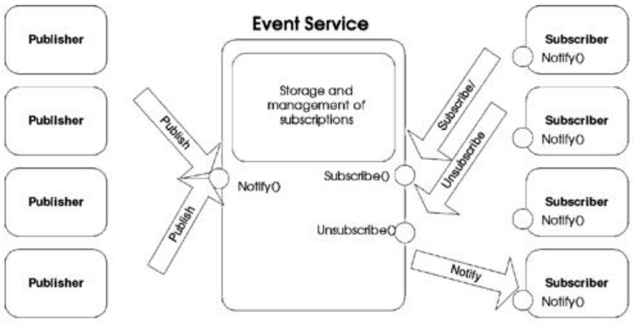
rmi://<host_name>[:<name_service_port>]/<service_name>

Indirect communication

Publish-subscribe systems

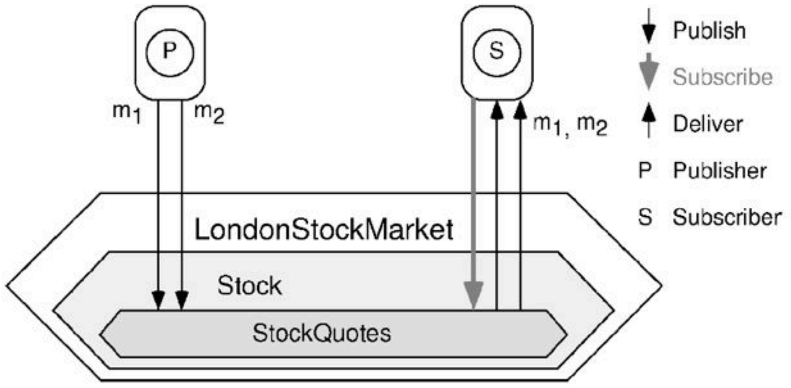
A publish-subscribe system is a system…

…where publishers publish structured events to an event service and subscribers express interest in particular events through subscriptions.

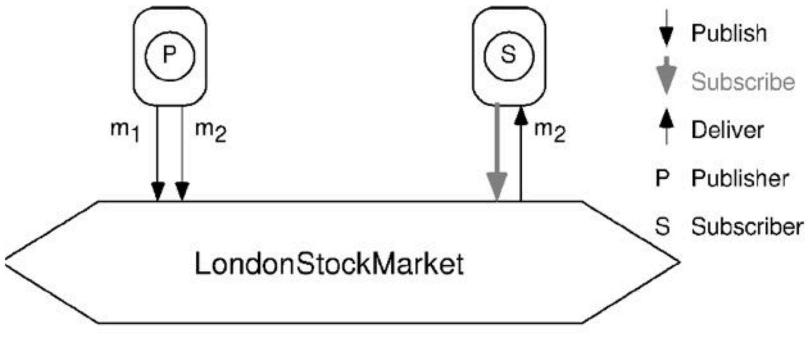


A simple object-based p-s system

Topic-based publish-subscribe interactions



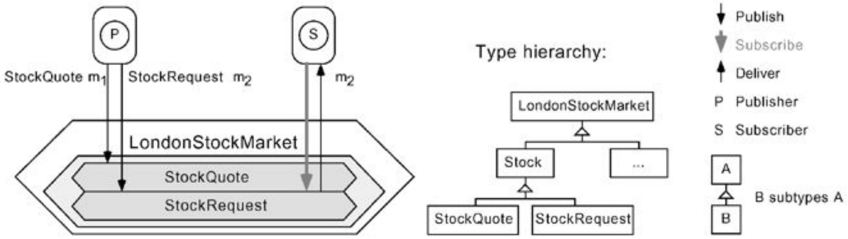
Content-based publish-subscribe interactions



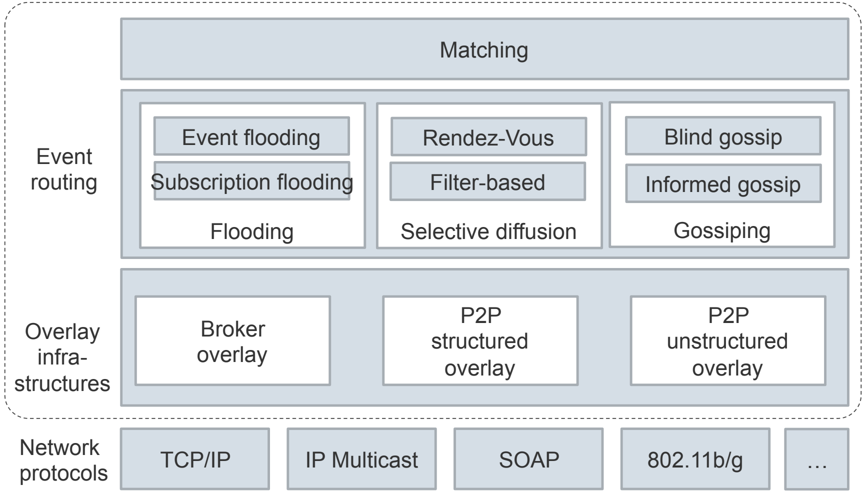
m_1: { ..., company: "Telco", price: 120, ..., ... }

m_2: { ..., company: "Telco", price: 90 , ..., ... }

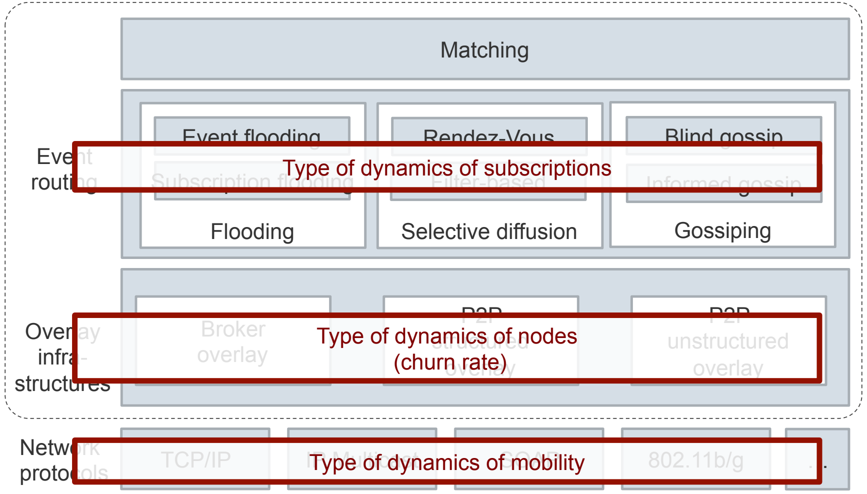
Type-based publish-subscribe interactions



The architecture of publish-subscribe systems



The architecture of publish-subscribe systems



Type of dynamics of subscriptions

Type of dynamics of nodes (churn rate)

Type of dynamics of mobility

Web Services

Characteristics of a web service

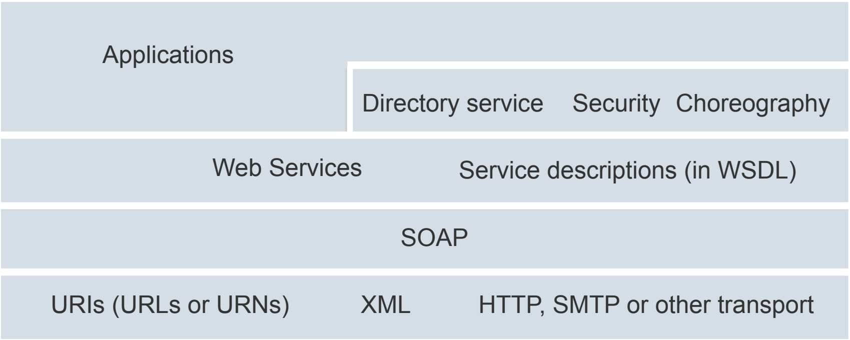
A web service interface generally consists of a collection of operations that can be used by a client over the Internet. The operations in a web service may be provided by a variety of different resources, for example, programs, objects, or databases.

The key characteristic of (most) web services is that they can process XML-formatted SOAP messages. An alternative is the REST approach.

Each web service uses its own service description to deal with the service-specific characteristics of the messages it receives.

Commercial examples include Amazon, Yahoo, Google and eBay.

Web service infrastructure and components



Applications

Directory service Security Choreography

Web Services Service descriptions (in WSDL)

SOAP

URIs (URLs or URNs) XML HTTP, SMTP or other transport

The Hypertext Transfer Protocol (HTTP)

HTTP Requests

After opening a connection, the client sends a request

- The method indicates the action to be performed on the resource
- HTTP's most interesting methods are: GET, HEAD, POST
- Other interesting methods are: PUT, DELETE

The URI identifies the resource to which the request should be applied

- Absolute URIs are required when contacting Proxies
- Absolute paths are required when contacting a server directly
- The URI may contain query information
- Fragment identifiers are not sent (they are interpreted on the client side)

The host header field must be included in every request.

(Wilde, 2008)

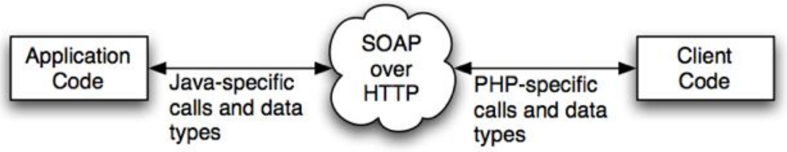
Realizing web services with SOAP

Simple Object Access Protocol (SOAP)

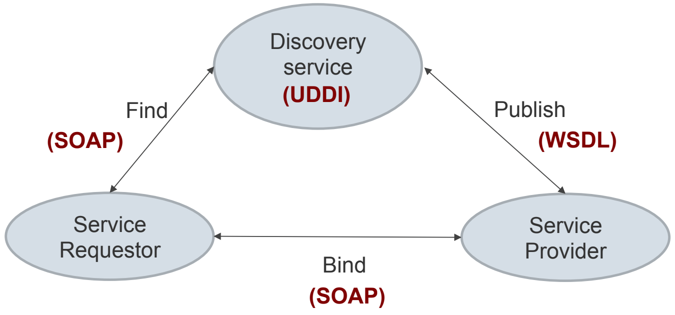
SOAP is designed to enable both client-server and asynchronous interaction over the Internet. It defines a scheme for using XML to represent the contents of request and reply messages as well as a scheme for the communication of documents.

It is used for information exchange and RPC, usually (but not necessarily) over HTTP.

(Very) basic SOAP architecture:


Application Code
Java-specific calls and data types
SOAP over HTTP
PHP-specific calls and data types
Client Code

Web service architecture (simplified)



http://www.w3.org/TR/ws-arch/

Representational State Transfer (REST)

REST design principles

Stateless Client/Server Protocol: Each message contains all the information needed by a receiver to understand and/or process it. This constraint attempts to “keep things simple” and avoid needless complexity.

A set of uniquely addressable resources enabled by a universal syntax for resource identification; “Everything is a Resource” in a RESTful system.

A set of well-defined operations that can be applied to all resources; In the context of HTTP, the primary methods are POST, GET, PUT, and DELETE, similar (but not exactly) to the database world's notion of CRUD (Create, Read, Update, Delete).

Resources are typically stored in a structured data format that supports hypermedia links, such as HTML or XML.

Establishing a common model

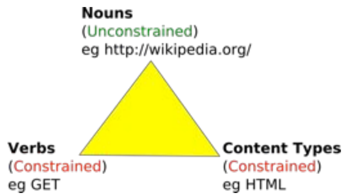
Distributed systems must be based on a shared model

- Traditional systems must agree on a common API
- REST systems structure agreement into three areas

REST is built around the idea of simplifying agreement

- *nouns* are required to name the resources that can be talked about
- *verbs* are the operations that can be applied to named resources
- *content types* define which information representations are available

REST triangle



REST vs. SOAP-based web services

REST is a description of the Web's design principles

- It is not something new, it is simply a systematic view of the Web
- REST's claim is to be able to learn from the Web's success

Web Services (in their narrow sense) do not build on REST

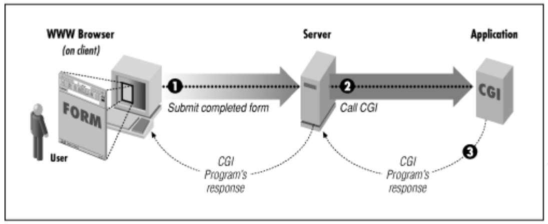
- They use HTTP as a transport protocol
- They re-create Web functionality through additional specifications (WS-*)
- They have been built by programmers using a top-down approach

REST and Web Services have different design approaches

- REST starts at the resources and takes everything from there
- Web Services focus on messages, which in most cases are operations

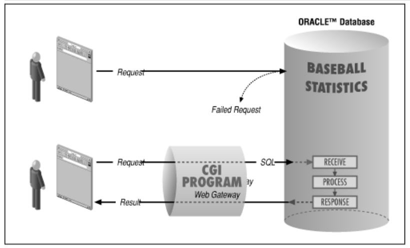
Common Gateway Interface (CGI)

Application areas



The Web server can call up a program, while passing user-specific data to the program.

The program then processes that data and the server passes the program's response back to the Web browser.



Forms, e.g. shopping, booking

Gateways, e.g. search engine, database

Virtual documents, e.g. guestbook, chat, bulletin board, dictionary

Hypertext Preprocessor (PHP)

Basic application areas

Server-side scripting

- Most traditional and main target field
- Ingredients: PHP parser (e.g., CGI), a web server and a web browser

Command line scripting

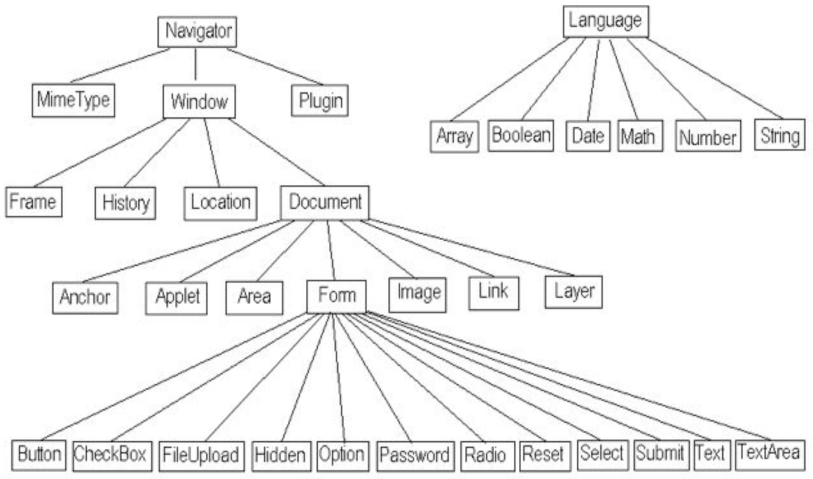
- Use a PHP script to run it without any server or browser
- Ingredients: PHP parser

Writing desktop applications

- Well, is not the very best language to create a desktop application with a graphical user interface
- Ingredients: you know PHP very well, and if you need some advanced PHP features in your client-side applications use PHP-GTK

JavaScript

JavaScript object hierarchy



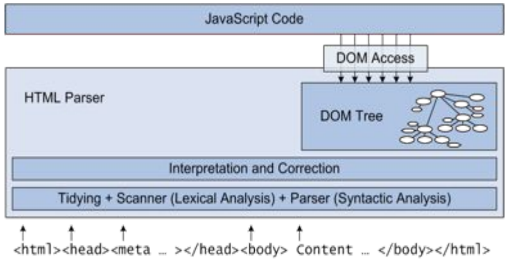
http://www.comptechdoc.org/independent/web/cgi/javamanual/javaobjheir.html

JavaScript principle

Contents	Behavior	Presentation
HTML	*JavaScript* (event handling)	*CSS*
<head> <script src="myJS.js"> <link rel="stylesheet" type="text/css" href="myCSS.css"> ...	myJS.js	myCSS.css

W3C Document Object Model (DOM)

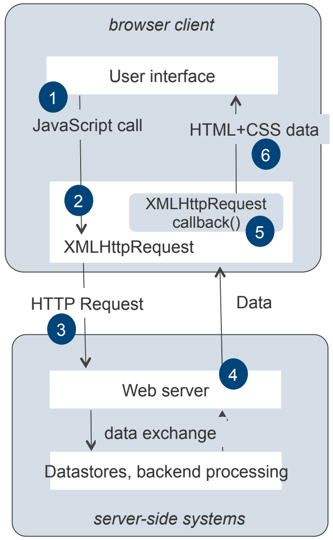
It describes a tree structure of all HTML elements, including attributes and the text they contain. (http://www.w3.org/DOM/DOMTR)


JavaScript Code
DOM Access
HTML Parser
DOM Tree
Interpretation and Correction
Tidying + Scanner (Lexical Analysis) + Parser (Syntactic Analysis)
<html><head><meta … ></head><body> Content … </body></html>

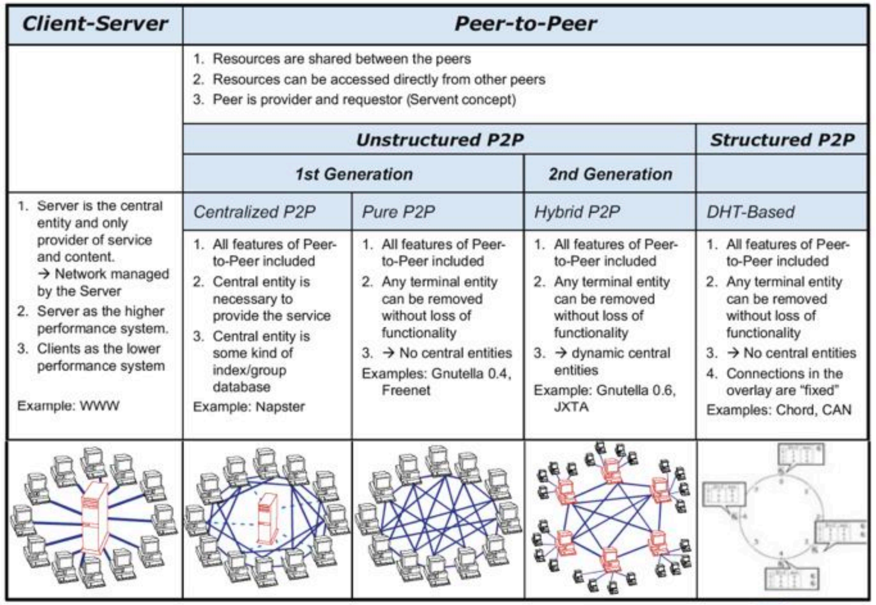
Asynchronous Java and XML (AJAX)

A typical Ajax request

1. User clicks, invoking event handler
2. Handler's JS code creates an XMLHttpRequest object
3. XMLHttpRequest object requests a document from a web server
4. Server retrieves appropriate data, sends it back
5. XMLHttpRequest fires event to say that the data has arrived (this is often called a callback; you can attach a handler to be notified when the data has arrived)
6. Your callback event handler processes the data and displays it



Peer-to-peer-systems



Client-Server	***Peer-to-Peer***			
	1. Resources are shared between the peers 2. Resources can be accessed directly from other peers 3. Peer is provider and requestor (Servent concept)			
	Unstructured P2P			***Structured P2P***
	1st Generation		***2nd Generation***	
1. Server is the central entity and only provider of service and content. → Network managed by the Server 2. Server as the higher performance system. 3. Clients as the lower performance system Example: WWW	*Centralized P2P*	*Pure P2P*	*Hybrid P2P*	*DHT-Based*
	1. All features of Peer-to-Peer included 2. Central entity is necessary to provide the service 3. Central entity is some kind of index/group database Example: Napster	1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → No central entities Examples: Gnutella 0.4, Freenet	1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → dynamic central entities Example: Gnutella 0.6, JXTA	1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → No central entities 4. Connections in the overlay are "fixed" Examples: Chord, CAN

(Eberspächer, & Schollmeier 2005)

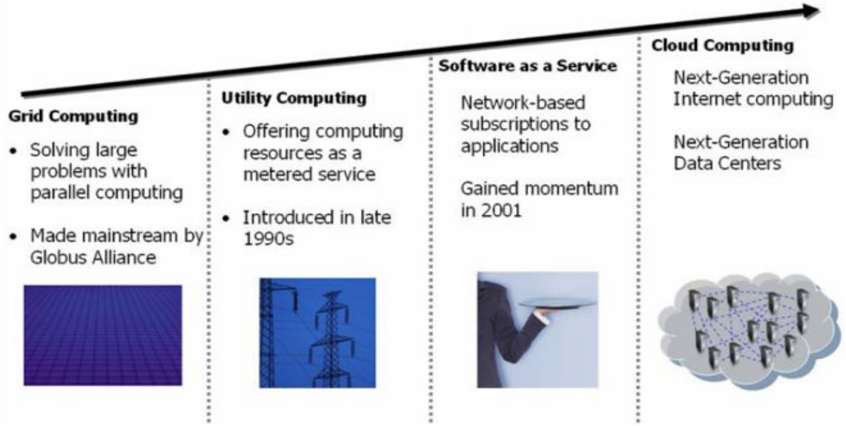
Comparison of discussed algorithms

PsP system	Model	Parameters	Hops to locate data	Routing state	Peers joins and leaves	Reliability
Napster	Centralized metadata index; Location inquiry from central server; Download directly from peer	None	Constant	Constant	Constant	Central server returns multiple download locations; client can retry
Gnutella	Broadcast request to as many peers as possible, download directly	None	no guarantee	Constant (approx 3-7)	Constant	Receive multiple replies from peers with available data; requester can retry
Pastry	Plaxton-style global mesh	N – number of peers in network b – base of the chosen identifier	$\log_b N$	$\log_b N$	logN	Replicate data across multiple peers; Keep track of multiple paths to each peer

Cloud Computing

Computing Models

“Why do it yourself if you can pay someone to do it for you?”



Grid Computing

- Solving large problems with parallel computing
- Made mainstream by Globus Alliance

Utility Computing

- Offering computing resources as a metered service
- Introduced in late 1990s

Software as a Service

Network-based subscriptions to applications

Gained momentum in 2001

Cloud Computing

Next-Generation Internet computing

Next-Generation Data Centers

What is the difference between cloud and grid computing?

Grid computing is where more than one computer coordinates to solve a problem together. Often used for problems involving a lot of number crunching, which can be easily parallelizable.

Cloud computing is where an application doesn't access resources it requires directly, rather it accesses them through something like a service.

The service maps any requests for resources to its physical resources, in order to provide for the application. Usually the service has access to a large amount of physical resources, and can dynamically allocate them as they are needed.

A cloud would usually use a grid. A grid is not necessarily a cloud or part of a cloud.

This excellent answer is taken from:
http://stackoverflow.com/questions/1067987/what-is-the-difference-between-cloud-computing-and-grid-computing

Cloud computing stack

Software as a Service (SaaS) **consume**

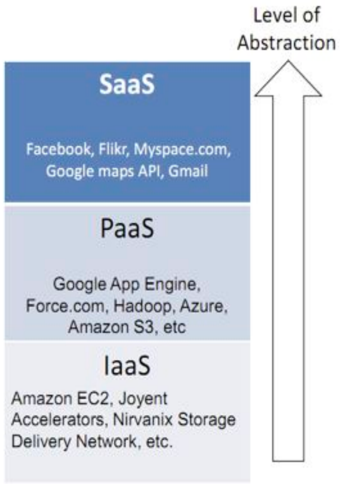
- A way to access applications hosted on the web through your web browser

Platform as a Service (PaaS) **build**

- A pay-as-you-go model for IT resources accessed over the Internet

Infrastructure as a Service (IaaS) **host**

- Use of commodity computers, distributed across Internet, to perform parallel processing, distributed storage, indexing and mining of data
- Virtualization


Level of Abstraction
SaaS
Facebook, Flikr, Myspace.com, Google maps API, Gmail
PaaS
Google App Engine, Force.com, Hadoop, Azure, Amazon S3, etc
IaaS
Amazon EC2, Joyent Accelerators, Nirvanix Storage Delivery Network, etc.

Virtualization techniques

- Emulation
- Full virtualization
- Paravirtualization
- Hardware-assisted virtualization
- Application-level virtualization

That’s all.