DISTRIBUTED SYSTEMS SERVICE ORIENTED COMPUTING

Goals

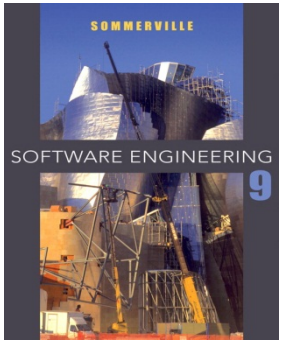
- Understand the Foundations
  - Characterization
  - Programming models
  - Remote Invokation
- Get knowledge on Middleware
  - Distributed Objects
  - Web Services
- Advanced concepts
  - Transations
  - Security

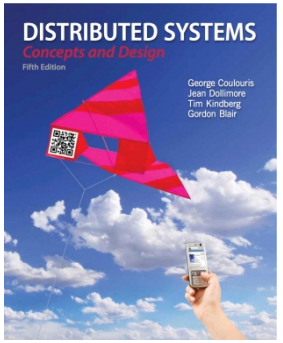
Content

- Foundations
- Platforms
- Distributed Object & Components
- Service Computing
- *Cloud*

Bibliography

- Slides of Martin Quinson (ESIAL 2008)
- Courses on the Internet
  - Systèmes et Applications Répartis (S. Krakowiak)
  - http://sardes.inrialpes.fr/~krakowia/Enseignement/M2P-GI/
  - Ecole d'été sur les intergiciels
  - http://sardes.inrialpes.fr/ecole/2003/support.html
- Distributed Systems 5th Edition
  - George Coulouris, Jean Dollimore, Tim Kindberg and Gordon Blair
  - http://www.cdk5.net/wp/
- Software Engineering 9
  - Ian Sommerville





Distributed Systems

- Components communicate and coordinate their actions only by message passing.
- Computer connected by a Network

  - Concurrency
  - No Global Clock
  - Independant Failure

Examples

- E-Commerce et trading
- Banking
- Call Centers
- Entertainment
- Science
- …

- Lot of heterogeneous informations sources
- Heterogeneous Systems
- High throughtput
- Rapide evolution

Evolution of scales

- Centralised computing
- Distributed applications on a local network (LAN)
- Distributed applications on a wide area network - Internet is the computer (Service, P2P)
- Elastic / Utility Computing (Cloud, Grid)
- Mobile/Ubiquitous computing

Distributed System

- Definition
  - A distributed system consists of multiple autonomous computers that communicate through a computer network. The computers interact with each other in order to achieve a common goal. (Wikipedia http://en.wikipedia.org/wiki/Distributed_system)

Trends in distributed systems

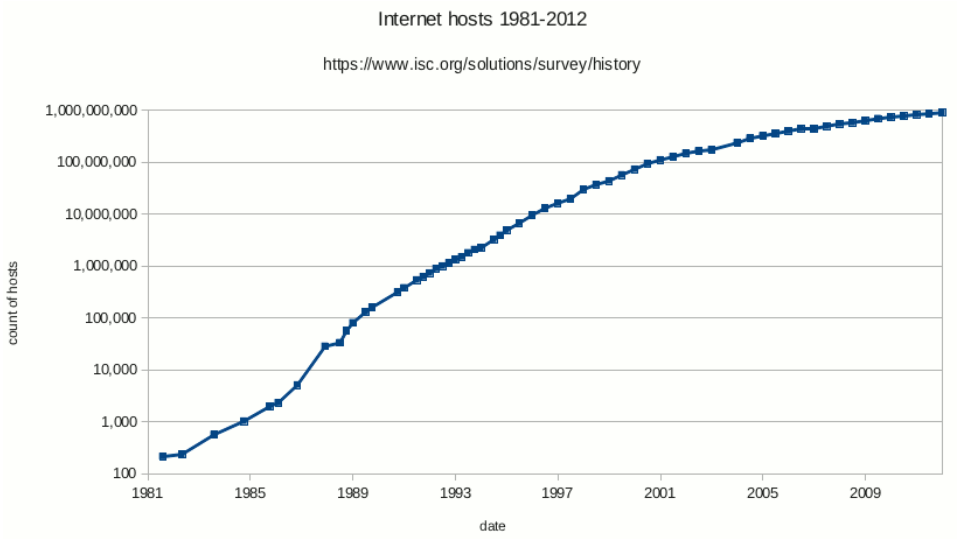
- Pervasive Networking
- Mobile and ubiquitous computing
  - Small portable devices
  - Wearable
  - Embedded
  - Computer everywhere
- Distributed Multimedia Systems
  - Webcasting, streaming
  - Quality of service
- Distributed computing as a utility (commoditisation)
  - Services
  - Cloud computing

Challenges

- Heterogeneity
- Openness
  - Key interfaces are published
- Security
- Scalability
- Failure Handling
- Concurrency
- Transparency
- Quality of service

Scalability

- Controling the cost of physical resources (O(n))
- Controling the performance lost (O(log n))
- Preventing software resources running out
- Avoiding performance bottleneck



Failure Handling

- Detecting failures
- Masking failures
- Tolerating failures
- Recovery from failure
- Redundancy

- Availability : from 99 to 99.99999

Transparency

- Access transparency : local vs remote
- Location transparency
- Concurrency transparency
- Replication transparency
- Failure transparency
- Mobility transparency
- Perdormance transparency
- Scaling transparency

System Models

- Physical Models
- Architectural Models
- Fundamentals Models

Physical Models

Distributed systems:	*Early*	*Internet-scale*	*Contemporary*
Scale	Small	Large	Ultra-large
Heterogeneity	Limited (typically relatively homogenous configurations)	Significant in terms of platforms, languages and middleware	Added dimensions introduced including radically different styles of architecture
Openness	Not a priority	Significant priority with range of standards introduced	Major research challenge with existing standards not yet able to embrace complex systems
Quality of service	In its infancy	Significant priority with range of services introduced	Major research challenge with existing services not yet able to embrace complex systems

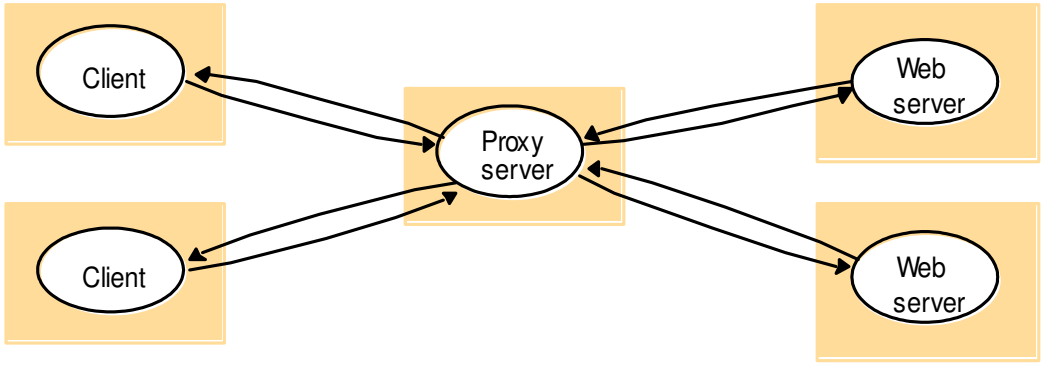
Architectural Models

- Communicating Elements and Paradigms

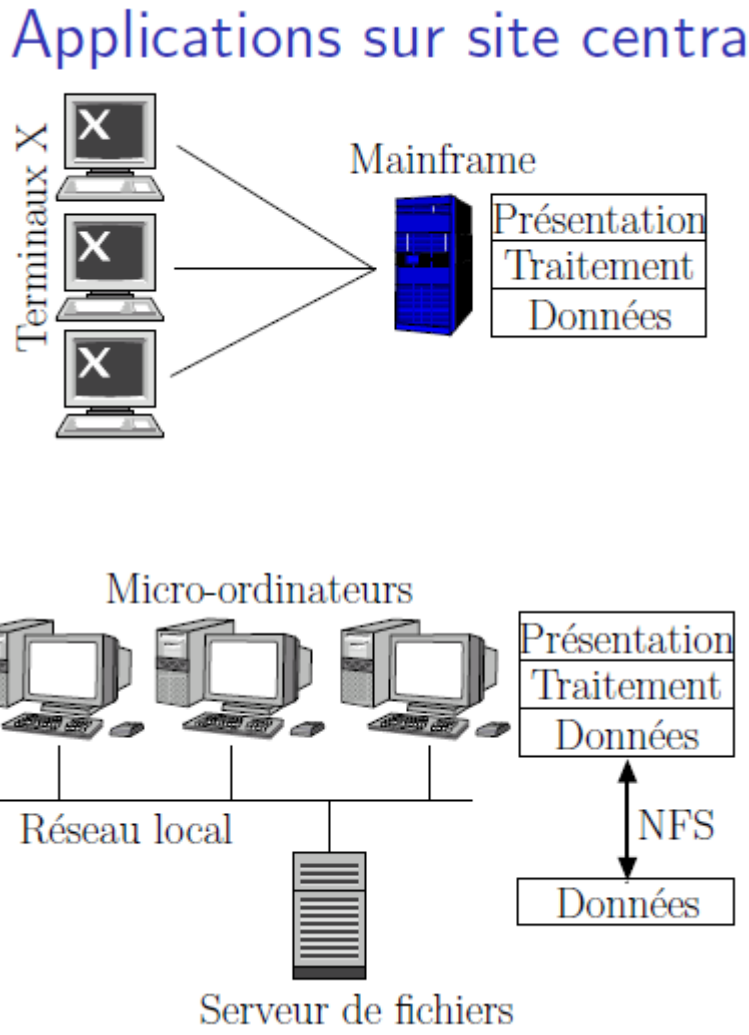
Communicating entities (what is communicating)		*Communication paradigms (how they communicate)*		
System-oriented entities	*Problem-oriented entities*	*Interprocess communication*	*Remote invocation*	*Indirect communication*
Nodes	Objects	Message passing	Request-reply	Group communication
Processes	Components	Sockets	RPC	Publish-subscribe
	Web services	Multicast	RMI	Message queues
				Tuple spaces
				DSM

Placement

- Mapping of service to multiple servers
- Caching
- Mobile code
- Mobile Agent

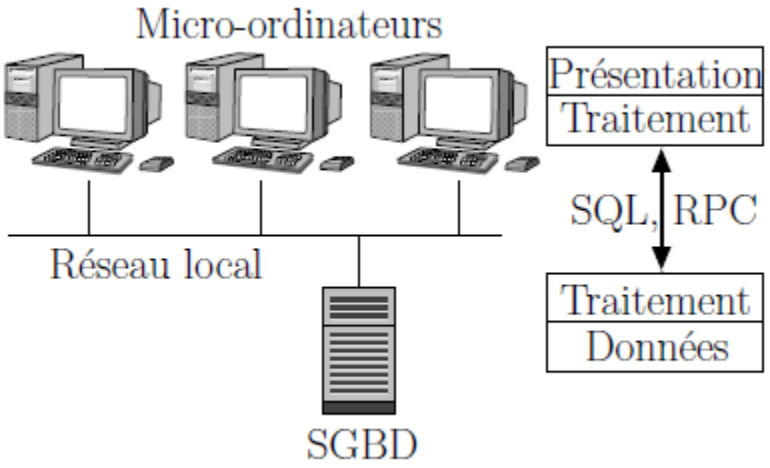


Tiered Architectures – One Tier

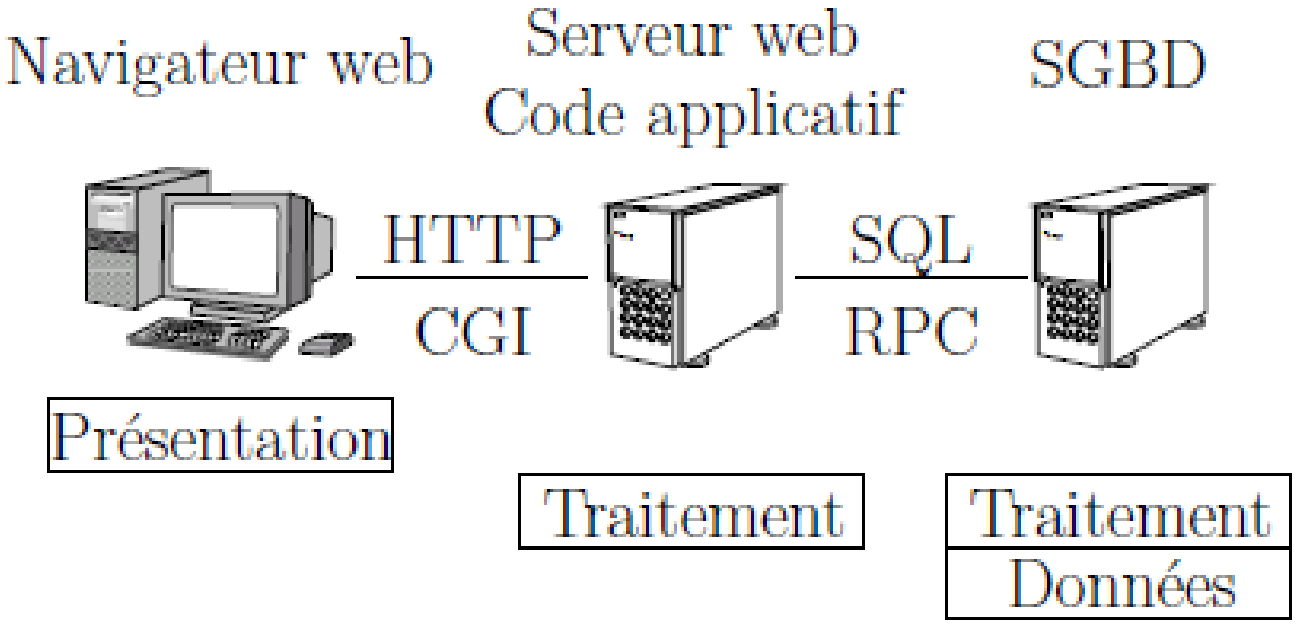


Two Tiers

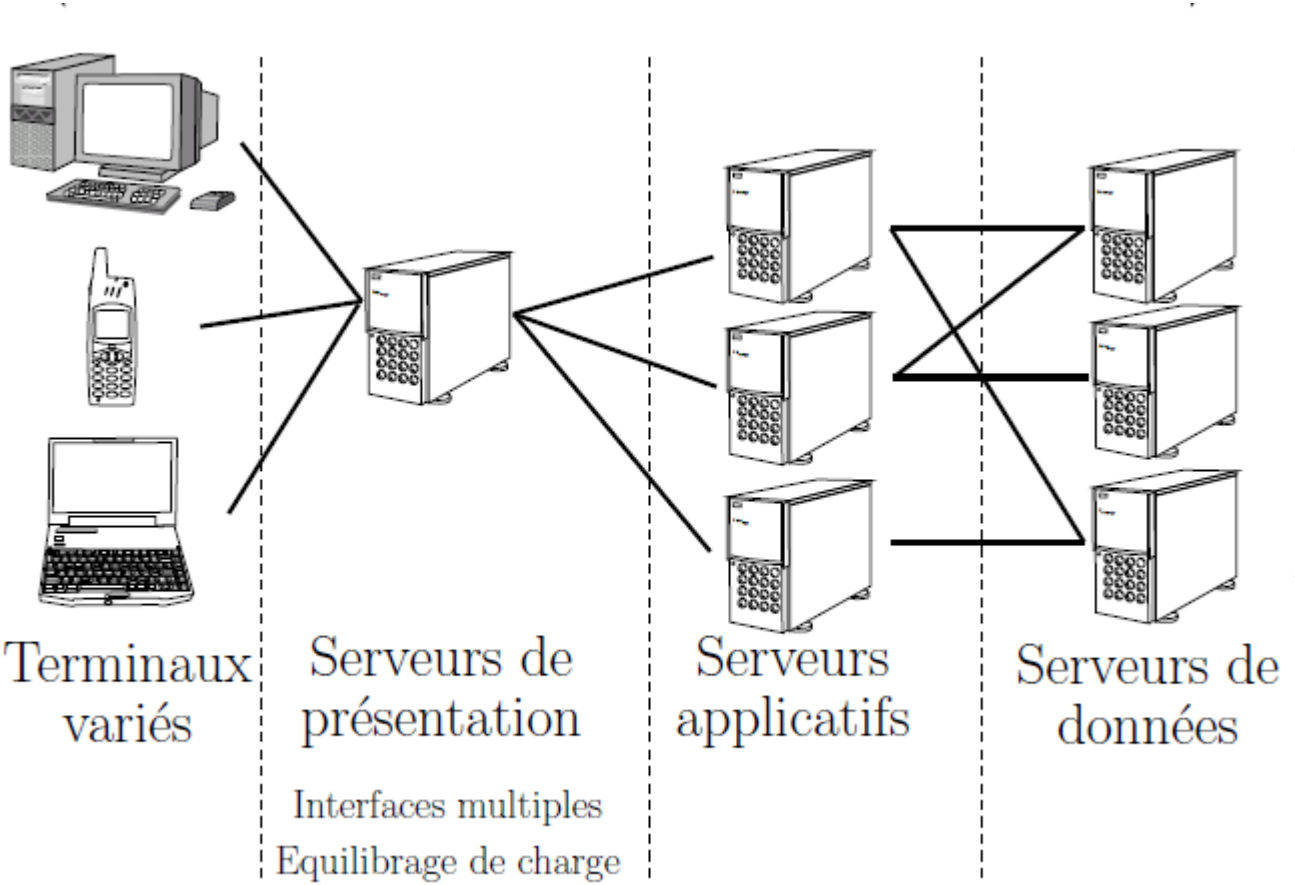
- Problems of maintenance and deploiement
- Consistency



Three tiers


Navigateur web
Serveur web
Code applicatif
SGBD
HTTP
CGI
SQL
RPC
Présentation
Traitement
Traitement
Données

Multi-tiers


Terminaux variés
Serveurs de présentation
Interfaces multiples
Equilibrage de charge
Serveurs applicatifs
Serveurs de données

Other patterns

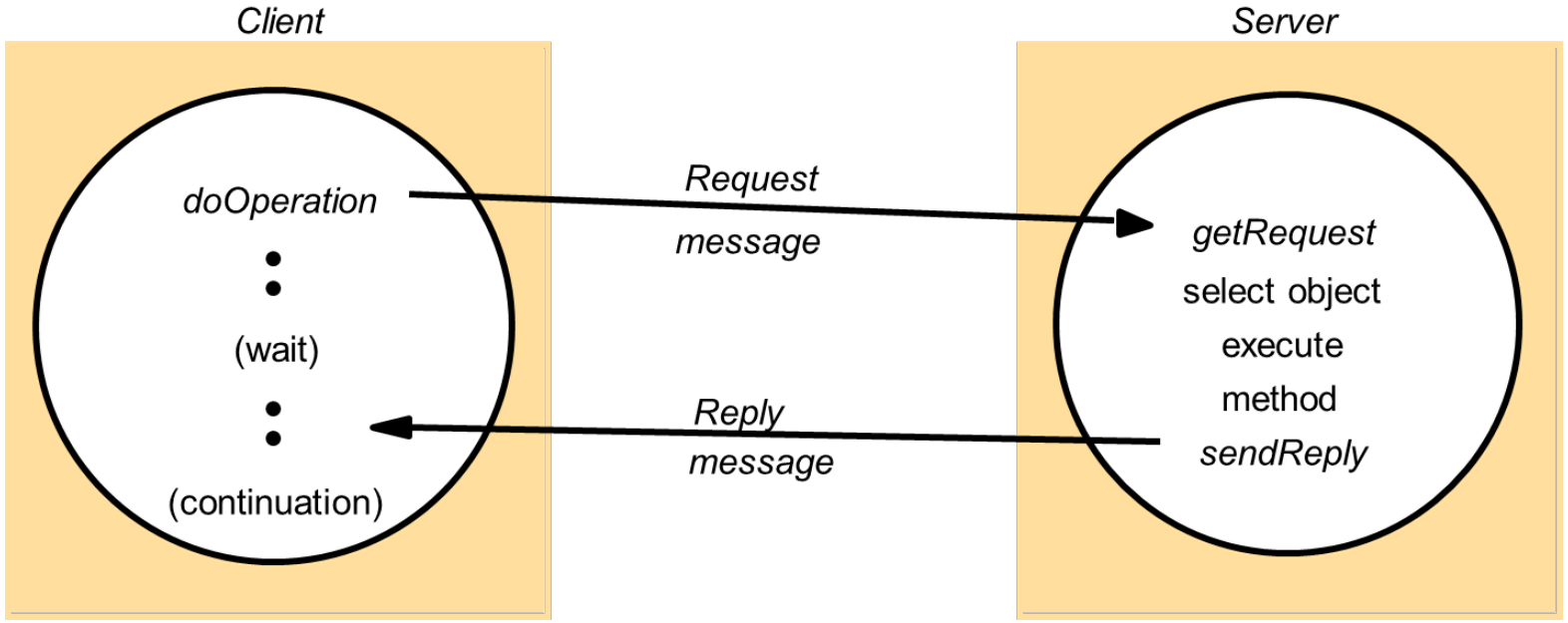
- Thin Client
- Layered
- Proxy
- Broker
- Reflection

Middleware Solutions

Major categories:	*Subcategory*	*Example systems*
Distributed objects (Chapters 5, 8)	Standard	RM-ODP
	Platform	CORBA
	Platform	Java RMI
Distributed components (Chapter 8)	Lightweight components	Fractal
	Lightweight components	OpenCOM
	Application servers	SUN EJB
	Application servers	CORBA Component Model
	Application servers	JBoss
Publish-subscribe systems (Chapter 6)	-	CORBA Event Service
	-	Scribe
	-	JMS
Message queues (Chapter 6)	-	Websphere MQ
	-	JMS
Web services (Chapter 9)	Web services	Apache Axis
	Grid services	The Globus Toolkit
Peer-to-peer (Chapter 10)	Routing overlays	Pastry
	Routing overlays	Tapestry
	Application-specific	Squirrel
	Application-specific	OceanStore
	Application-specific	Ivy
	Application-specific	Gnutella

Remote Invocation

- How process communicate in a distributed system
- Request-reply protocols
- What are the problems and the solutions ?



Request Reply Protocols

public byte[] doOperation (RemoteRef s, int operationId, byte[] arguments)

sends a request message to the remote server and returns the reply.
The arguments specify the remote server, the operation to be invoked and the arguments of that operation.

public byte[] getRequest ();

acquires a client request via the server port.

public void sendReply (byte[] reply, InetAddress clientHost, int clientPort);

sends the reply message reply to the client at its Internet address and port.

Call Semantics

Fault tolerance measures			*Call semantics*
Retransmit request message	*Duplicate filtering*	*Re-execute procedure or retransmit reply*	
No	Not applicable	Not applicable	*Maybe*
Yes	No	Re-execute procedure	*At-least-once*
Yes	Yes	Retransmit reply	*At-most-once*

Transparency

- Goal : remote calls as similar as local calls as possible
- Location and access transparency
- New kind of failures (network)
- Latency
- Parameter passing

- Difference between local and remote call should be expressed in the interface

RPC

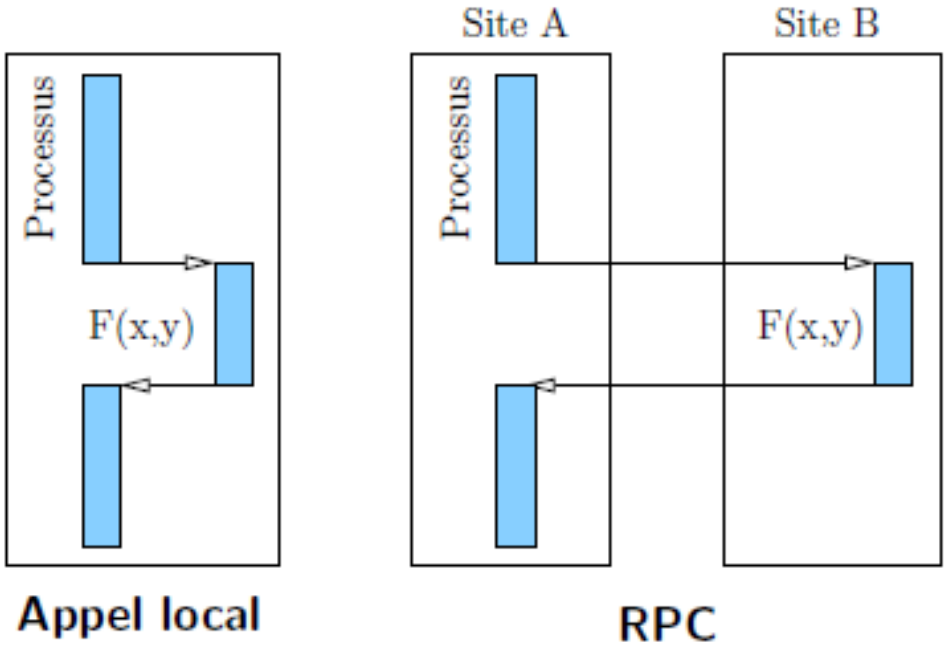
- History
  - Invented in1976 (RFC 707)
  - Implemented by SUN in 1988
  - Used in NFS
- Design issues
  - Programming with interfaces
  - Call semantics
  - Transparency

Three key issues

- Programming with interfaces
  - Hide the implementation details
  - Hide programming language
  - Easier evolution
- Call Semantics
  - Sun RPC provides « at-least-once » semantics
- Transparency
  - Location and access transparency
  - No call by reference

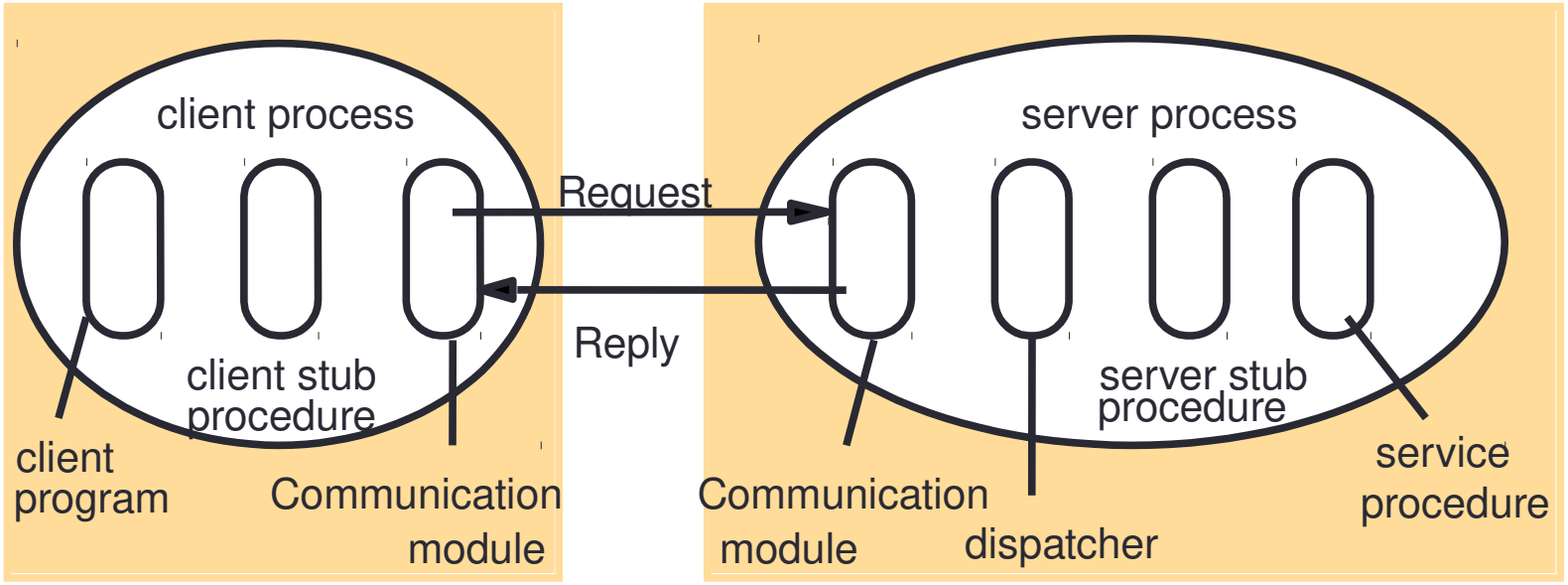
Splitting the application

- Procedure as the unit of abstraction
- RPC – remote procedure call



SUN RPC (1988)

- Generation of code common to all calls



SUN RPC

- The Client Stub
  - Receive the local call
  - Wrap the parameters (marshalling)
  - Generate an identifier
  - Send the client call
  - Waits for the result
  - Receive and unwrap the result (unmarshalling)
  - Send the result to the caller (return)
- The Server Skeleton
  - Receive the call
  - Unwrap the parameters
  - Execute the local call
  - Wrap and return the result

Naming and binding

- Naming : name given by an entity to another
- Binding : setting in relation entities

- Naming: what procedure is called on which computer
  - Naming should not depend on localisation

- Late or early binding
  - Early : known at compile time
  - Late : selected at execution time

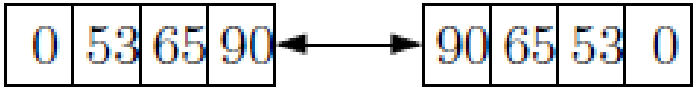
Registry : from name to address

- 1: Binding
- 2: Lookup

- Criteria
  - Name (service, computer, node)
  - Properties

Data representation

- Computers heterogeneity
- Difference of internal data representation
  - Little-endian (x86) vs Big-endian (ppc, sparc)
  - Floating points(IEEE 754)
  - Data alignment
  - Size



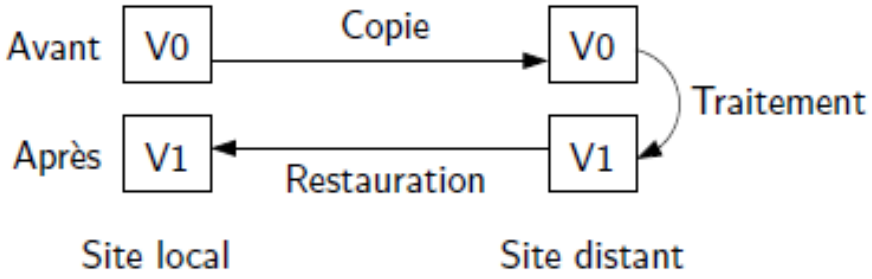
type	char	short	int	long	long long	*	float	double
sizeof	1	2	4	4/8	8	4/8	4	8

Data representation on the network

- eXternal Data Representation (XDR, SUN RPC)
- Other solutions
  - htonl, ntohl, etc : manual conversions
  - ASN.1
  - XML : SOAP, JXTA.
  - IIOP : CORBA
  - NDR : Network Data Representation

Parameter passing

- By value: foo(43)
  - No problem
- By reference : foo(&x)
  - Impossible – not the same address space
- By copy/restore
  - Potential problem



Aliasing Problems

Définition

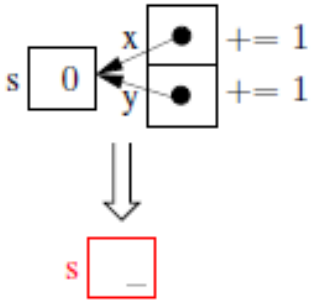
```
procedure inc2(x,y)
  x += 1
  y += 1
```

Invocation

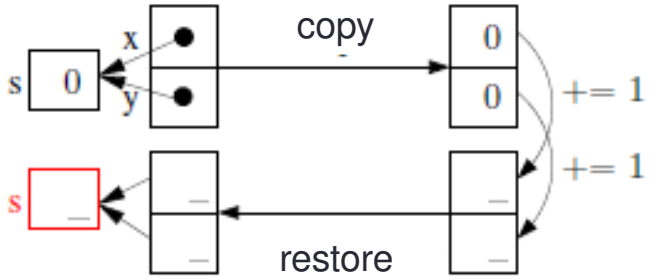
```
s = 0
inc2(s,s)
```

What is happening ?

By reference



By copy/restore



Failures

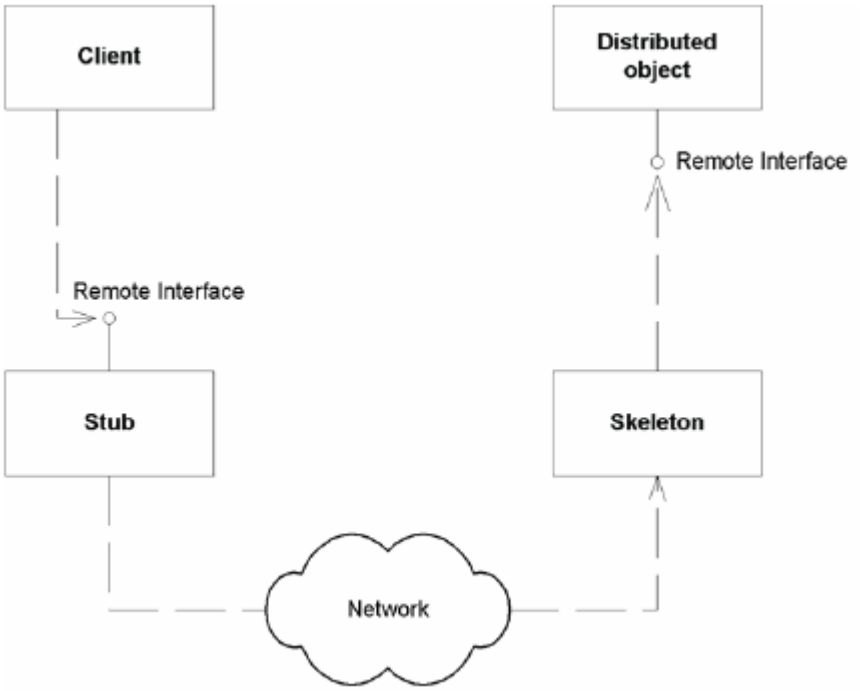
Class of failure	*Affects*	*Description*
Fail-stop	Process	Process halts and remains halted. Other processes may detect this state.
Crash	Process	Process halts and remains halted. Other processes may not be able to detect this state.
Omission	Channel	A message inserted in an outgoing message buffer never arrives at the other end's incoming message buffer.
Send-omission	Process	A process completes a *send,* but the message is not put in its outgoing message buffer.
Receive-omission	Process	A message is put in a process's incoming message buffer, but that process does not receive it.
Arbitrary (Byzantine)	Process or channel	Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times, commit omissions; a process may stop or take an incorrect step.

RPC Call semantics

- Maybe semantics
  - Not certain that the procedure has been executed
- At least once semantics
  - Retransmission of the request message
  - The procedure can be executed more than once
- At most once semantics

Distributed Objects

- RPC but with objects
- The remote object can be called as a local object



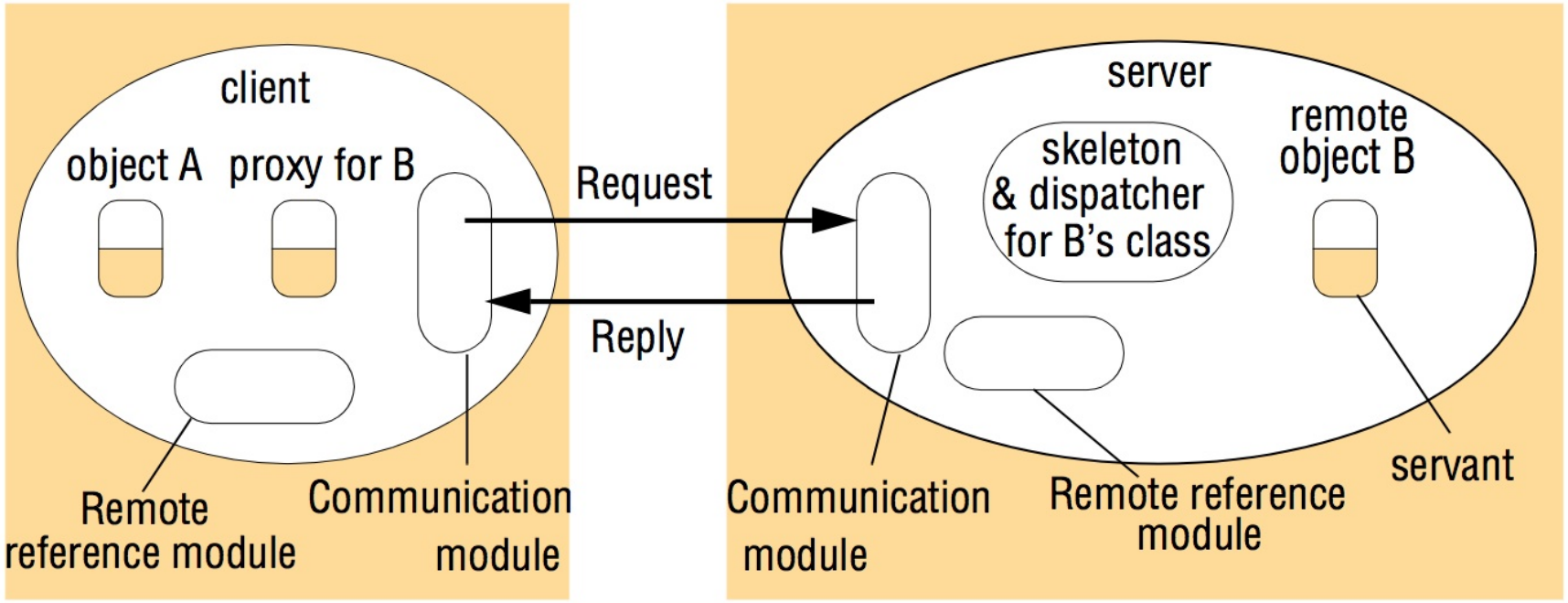
RPC vs RMI (Remote Method Invocation)

- Commonalities
  - Programming with interfaces
  - Request reply protocol
  - Similar level of transparency
- Differences
  - RMI : object oriented programming
  - Object have an identity and can be passed as parameters

Java RMI

- Object model
  - Object references
  - Interfaces
  - Actions
  - Exceptions
  - Garbage Collection
- Distributed object model
  - Remote object
  - Remote object reference
  - Remote interfaces
  - Actions
  - Exceptions
  - Distributed Garbage Collection

Implementation of RMI



Implementing a distributed object

- Big picture
  - Define the interface
  - Implement it (servant)
  - Implement the server (instantiate the servant and bind it)
  - Implement the client
- The interface contract
  - Must be public
  - Must extends java.rmi.Remote
  - Each method must throw java.rmi.RemoteException

Programming principles

- Servant
  - Must implement the remote interface (Remote)
  - May implement local methods
- Server
  - Instantiate and install a security manager
  - Instantiate the servant class
  - Bind the instant in the name registry
- Registry
  - Maintain a reference between the name and the reference
  - Implement java.rmi.registry.Registry
  - Provide the bind and lookup service

Example

- HelloInterface.java

```
package cours.serveur;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface HelloInterface extends Remote {
    public String sayHello() throws RemoteException;
}
```

Example

- HelloImpl.java

```
package cours.serveur;

import java.rmi.RemoteException;

public class HelloImpl implements HelloInterface {
    private String message;

    public HelloImpl(String s) {
        message=s;
    }

    public String sayHello(String s) throws RemoteException {
        return message+" "+s;
    }

}
```

Example

- Se

```
package cours.serveur;

import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;

public class Serveur {

    public static void main(String[] args) {
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new SecurityManager());
        }
        try {
            String name = "Hello";
            HelloInterface hello = new HelloImpl("hello");
            HelloInterface stub =
                (HelloInterface) UnicastRemoteObject.exportObject(hello, 0);
            Registry registry = LocateRegistry.getRegistry();
            registry.rebind(name, stub);
            System.out.println("Hello bound");
        } catch (Exception e) {
            System.err.println("Hello exception:");
            e.printStackTrace();
        }
    }
}
```

Example

- Client.java

```java
package cours.client;

import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

import cours.serveur.HelloInterface;

public class Client {
    public static void main(String[] args) {
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new SecurityManager());
        }
        try {
            String name="Hello";
            Registry registry=LocateRegistry.getRegistry();
            HelloInterface hello=(HelloInterface) registry.lookup(name);
            System.out.println(hello.sayHello("les amis"));
        } catch(Exception e) {
            System.err.println("Hello exception:");
            e.printStackTrace();
        }

    }

}
```

Compilation and deploiement

- Compile all the classes
  - javac *
- Server
  - HelloInterface, HelloImpl, Serveur
- Client
  - Client, HelloInterface
- Create jar for deploiement

Execution

- rmiregistry &
  - Default port : 1099
  - -J-Dsun.rmi.loader.logLevel=BRIEF to see what is going on
- Start the server
  - java -Djava.security.policy=java.policy HelloServer &
- Start the client
  - java HelloClient &

Security

- Allowing distant connection is dangerous
- Executing downloaded code is dangerous
- Security policies describe authorisations
  - No policy implies that external connection are refused
  - java.policy files

Seules utilisations autorisées

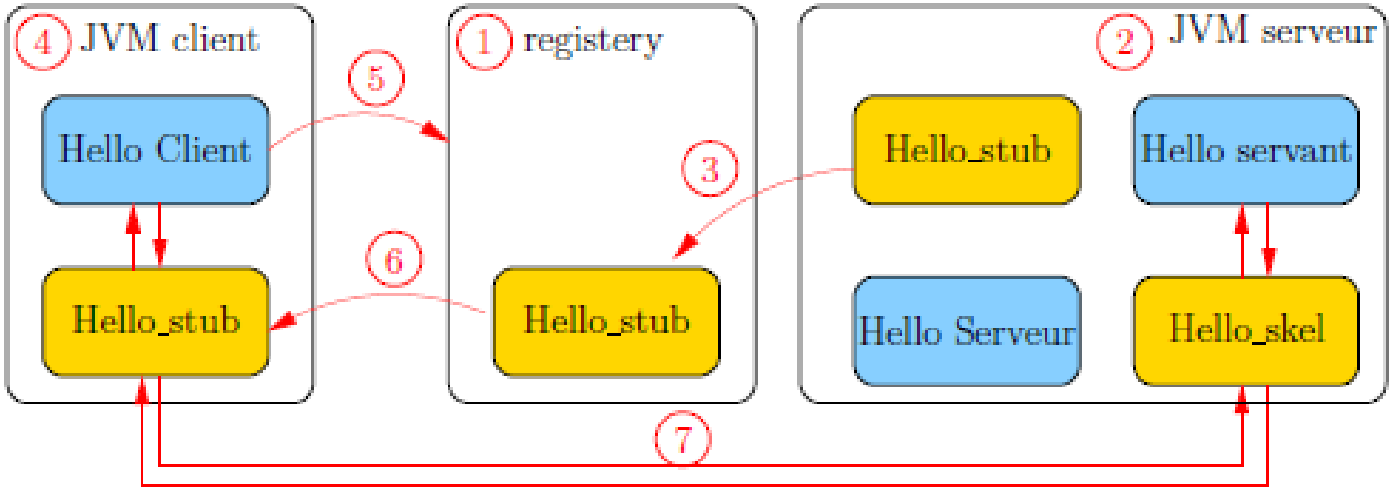
```
grant {
 permission java.net.SocketPermission
   "*:1024-65535","connect,accept,resolve";

 permission java.net.SocketPermission
  "*:80", "connect";
};
```

Dangereux ! (dans la vraie vie)

```
grant {
  permission java.security.AllPermission;
};
```

```
-Djava.security.policy=<nom du fichier>
```

RMI execution in Java

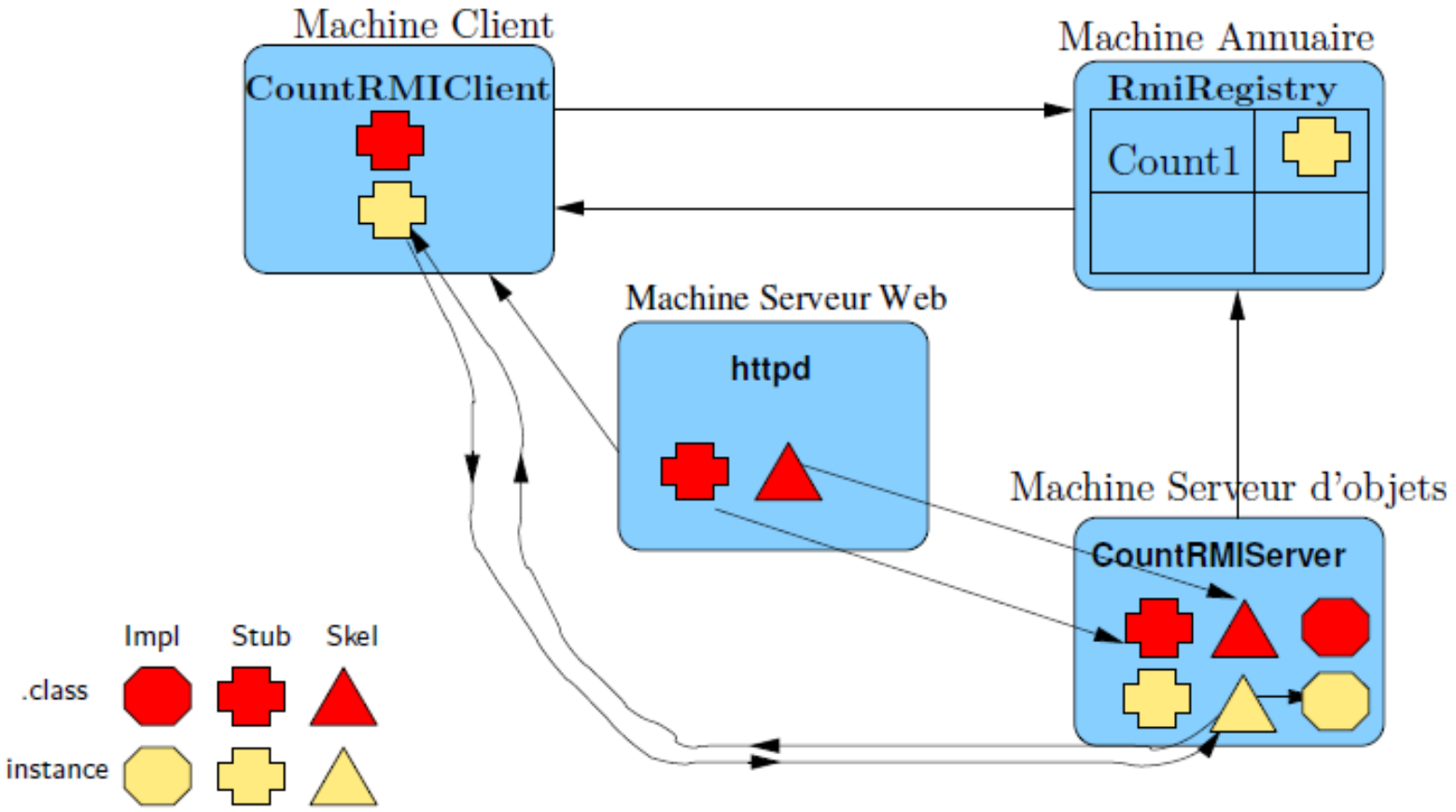


Downloading classes

- Stubs and class can be stored on a web site
- Automatic download on clients
- Download occurs when
  - The client receive a stub not in his CLASSPATH
  - The server receive an object reference for an unknown class
- Class localisation using a codebase specification

```
-Djava.rmi.server.codebase="http://toto.loria.fr/truc.jar http://sun.com/JavaDir/"
```

RMI and Code downloading



Polymorphism and RMI

- Polymorphism
  - As usual by subtyping
  - The subtype code is downloaded
  - Allows code evolution
- Parameter passing
  - The server provides m(C p)
  - The client request server.m(x) . Classe(x)<C
- Result
  - The client calls C r=server.m(…)
  - The server return a result of class C2<C

Object Factory and reference passing

- A client want a new reference from a server object
- It requires it from a factory object
  - Example : a session object for each client

Asynchronous calls and concurrency

- Several clients ask for services
  - What is the behaviour of the server ?
- Use of thread in the server
  - Concurrent calls
  - Concurrent access to ressources
  - Explicit management of concurrency

Callbacks

- The server informs the clients of events
- Solution
  - The client implements a remote object that can be called by the server
  - The server provides an interface to register the client remote reference (the callback object)
  - When a relevant event occurs, the server calls a method on the client.
- The server maintains a list of client objects.
- The server needs to make series of synchronous calls

Distributed Garbage Collection

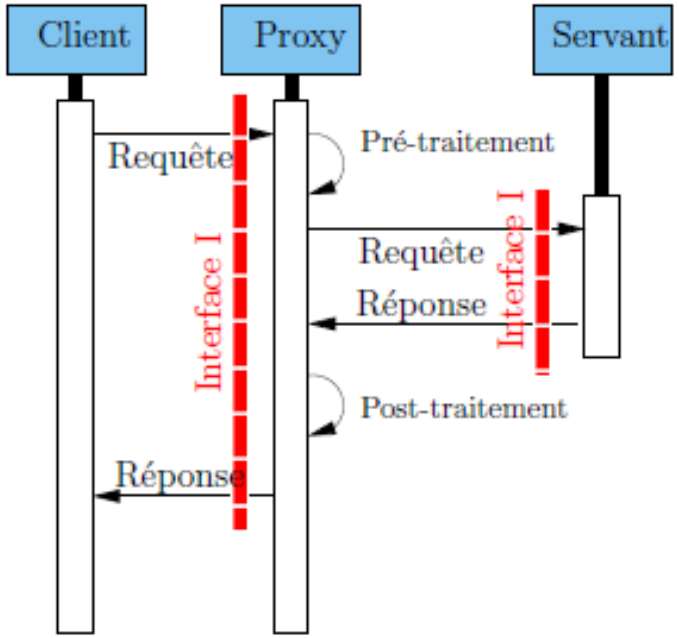
- If there is a reference to an object (remote or local) the object continue to exist
  - Uses reference counting
  - When a client C receive a remote ref, it notify the server process and creates a proxy.
  - When the ref is garbaged on C the ref is removed on the server
  - Lease expiration is used to cope with client failure

Design Patterns and Distributed Systems

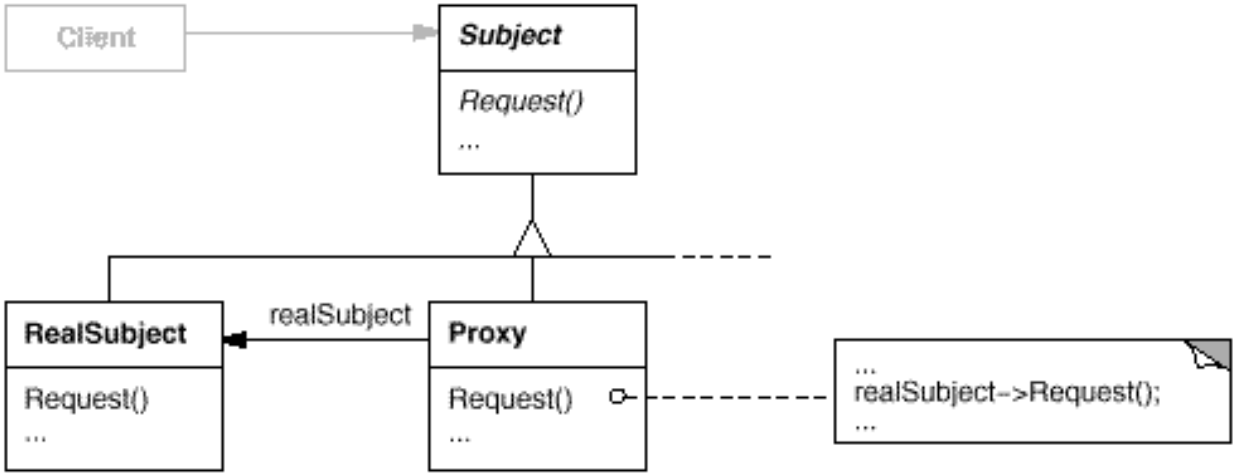
- General principle : Separation of concern
  - Manage separately orthogonal dimensions
  - Reduce interferences
  - Allow independant evolutions
- Obtained with
  - Encapsulation
  - Abstraction
  - Components
  - Aspect Oriented Programming

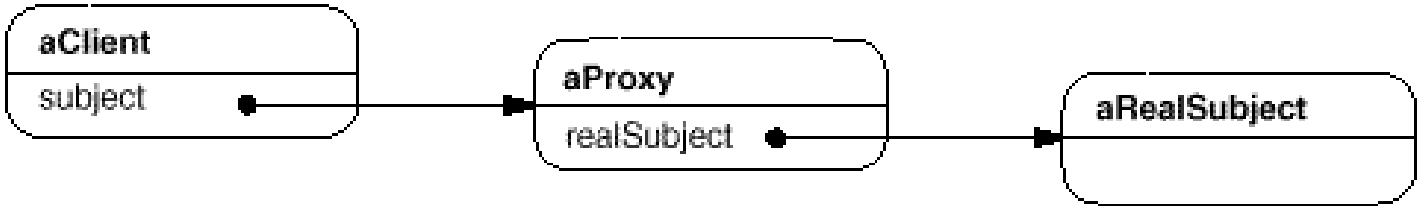
Proxy

- Access to a distant object
- Isolate the client from the server
- The proxy object manage the communication



Proxy (from the GOF Book)



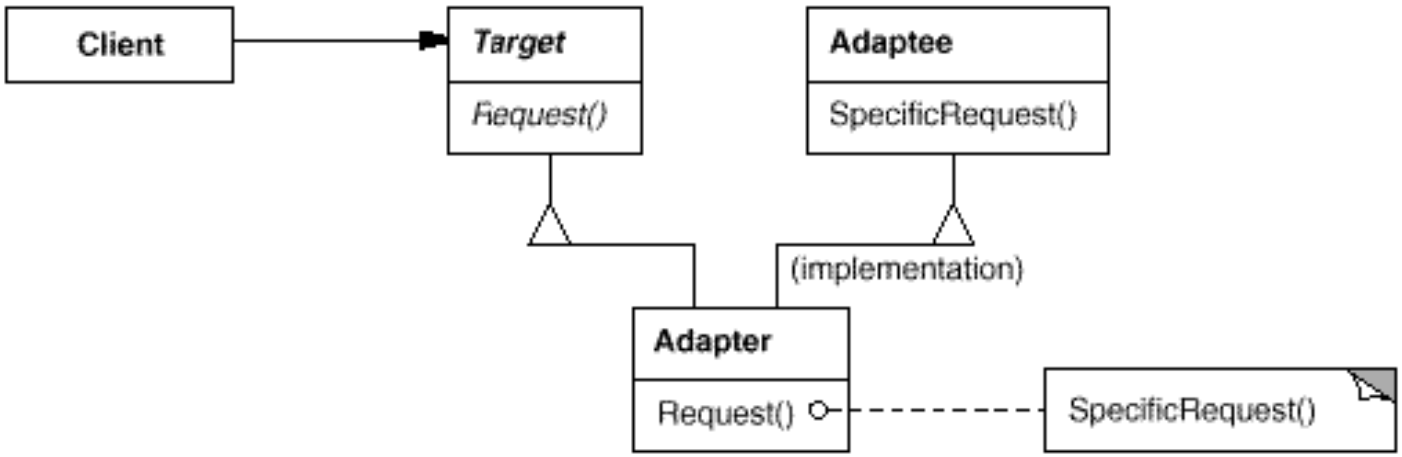


The Factory

- Separate the choice of an implementation from its use
  - Provide methods to create concrete objects
  - The client only knows interfaces*
- Replace by dependency injection

Wrapper ou adapter

- A client requires a different interface than the servant



C. A. R. Hoare

I conclude that there are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies and the other way is to make it so complicated that there are no obvious deficiencies.