

Programmation des Applications Réparties

Aurélien Esnard

ENSEIRB PG306

aurelien.esnard@labri.fr
<http://www.labri.fr/~esnard>

Plan

- PG306 (7 cours-intégrés de 4h, RSR + PRCD)
 - Technologies étudiées : RPC, Java RMI, CORBA, Web Services, EJB, Fractal, CCM, ...
 - Evaluation : *à définir*

Introduction aux Systèmes Répartis

Systemes Répartis

- Définition générale

- C'est un ensemble de processus communicants, répartis sur un réseau de machines le plus souvent hétérogènes, et coopérant à la résolution d'un problème commun.

« A distributed system is a collection of independent computers that appear to the users of the system as a single computer. » [A. Tanenbaum, Prentice-Hall, 1994]

- Intergiciel

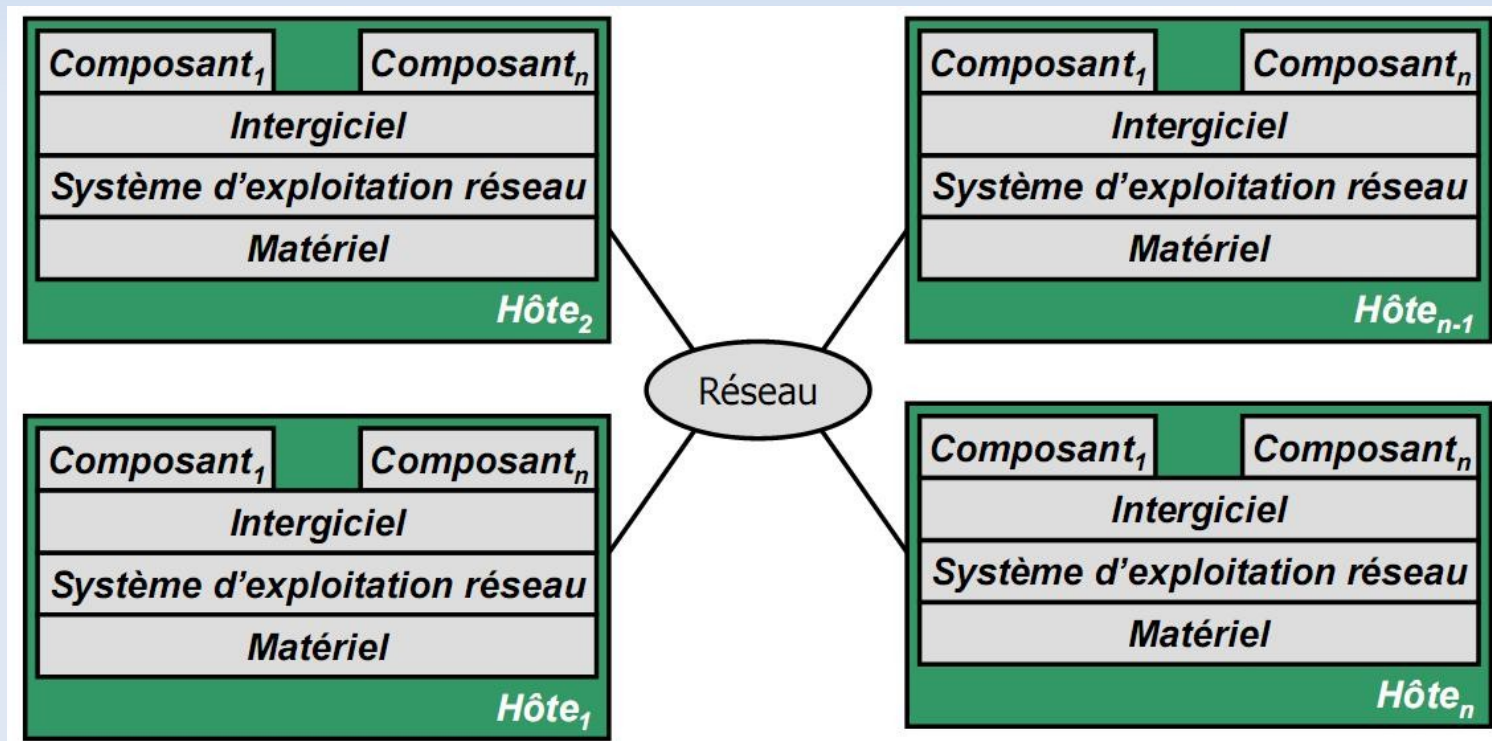
- Couche logicielle intermédiaire située entre l'application et le système d'exploitation de la machine...
- Servant à concerver / développer / déployer une application répartie

- En résumé...

- Les systèmes répartis, c'est régler à plusieurs des problèmes qu'on aurait pas eus tout seul !

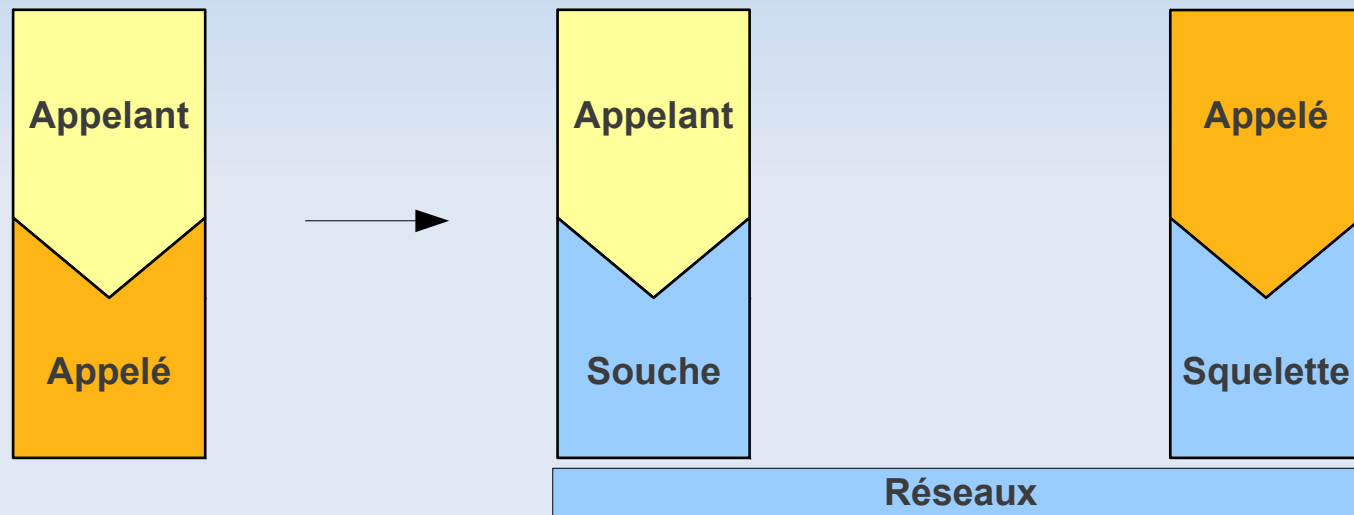
Systemes Répartis

- Plusieurs composantes réparties sur plusieurs machines et interconnectés grâce à un intergiciel



Systemes Répartis

- Du centralisé vers le réparti...



- Nouveaux problèmes !!!
 - localisation/nommage, transparence, interopérabilité, ...

Systemes Répartis

- Les défis des systèmes répartis !
 - Réutiliser les codes patrimoniaux (“legacy code”)
 - Hétérogénéité : différents vendeurs, différents langages, différents protocoles, différentes machines, ...
 - Portabilité, Interopérabilité, Extensibilité
 - Ouverture : standards ouverts, interfaces uniformes et normalisées
 - Transparence : masquer aux utilisateurs la répartition des données et des traitements (localisation / nommage)
 - Services communs (annuaire, sécurité, transaction, ...)
 - Performance, passage à l'échelle (“scalability”)
 - Tolérance aux fautes, reconfiguration dynamique, ...

Historique

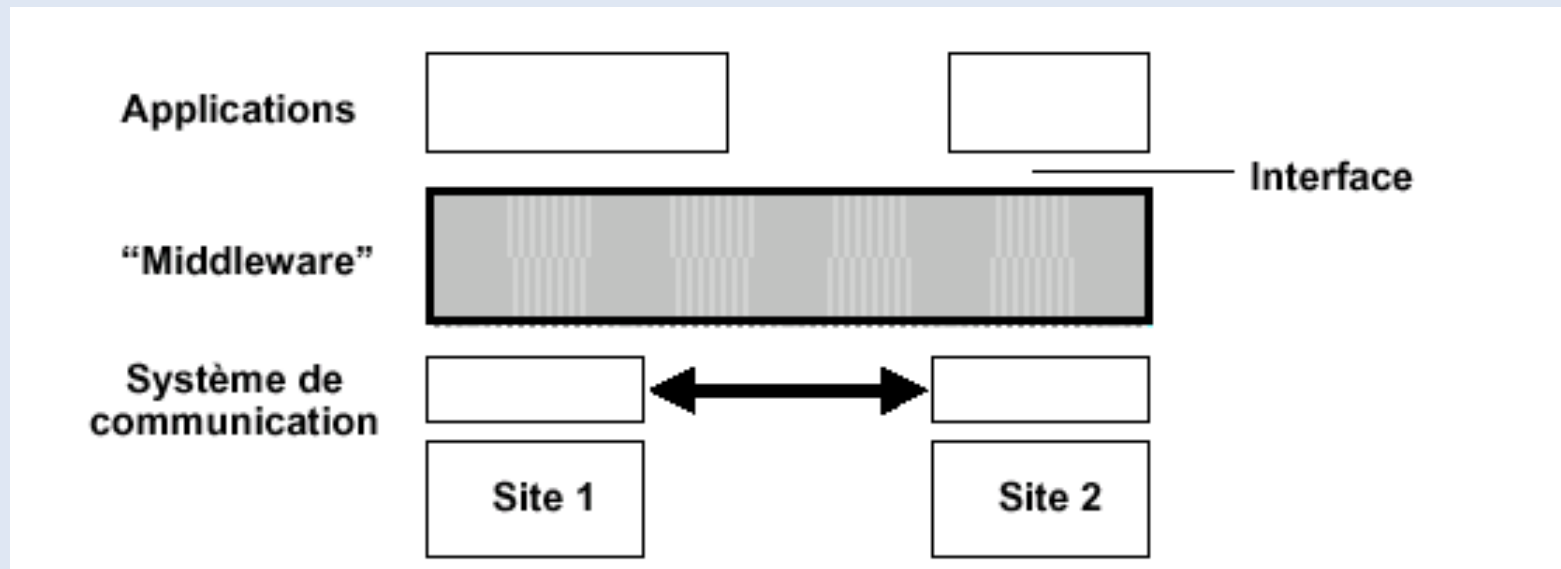
- Internet
 - 1969 : Arpanet, l'ancêtre d'Internet
 - 1973 : Ethernet 3 Mbits/s
 - 1983 : Naissance d'Internet et TCP/IP (mail, ftp, telnet)
 - 1990 : World Wide Web
 - 1980 : Ethernet 10 Mbits/s
 - 1998 : Gigabit Ethernet
 - ~2000 : ADSL

Historique

- **Systemes répartis**
 - RPC : Sun RPC créé pour NFS (1989), version 2 (1995), ...
 - OMG : CORBA 1 (1991), CORBA 2 avec IIOP (1995), CORBA 3 avec CCM (2002)
 - Sun Java : JDK 1.0 (1996), Java RMI (1997), J2EE / EJB (1998)
 - SGBD : SQL (IBM, norme ISO en 1986), ODBC inspiré par Microsoft (1992), JDBC de Sun
 - Microsoft : COM (1993), DCOM (1996), .NET (2002)
 - Web Services (~2000) : XML-RPC, SOAP

Intergiciel (Middleware)

- 3 objectifs principaux
 - Masquer la complexité de l'infrastructure sous-jacente
 - Faciliter la conception, le développement et le déploiement d'applications réparties
 - Fournir des services communs

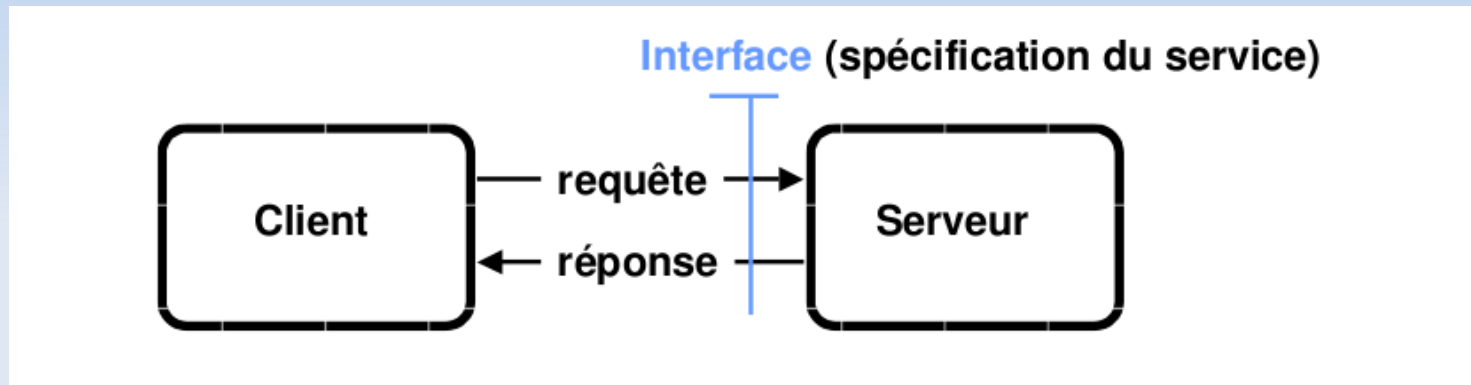


Les Classes d'Intergiciel

- Orientés objets distribués
 - Java RMI, CORBA, ...
- Orientés composants distribués
 - EJB, CCM, ...
- Orientés transactionnels
 - JTS de Sun, MTS de Microsoft
- Orientés messages
 - JMS de Sun, MSMQ de Microsoft, MQSeries de IBM, ...
- Orientés Web
 - Web Services (XML-RPC, SOAP)
- Orientés SGBD / SQL
 - ODBC (Open DataBase Connectivity), JDBC de Sun

Architecture Client-Serveur

- Interaction client/serveur

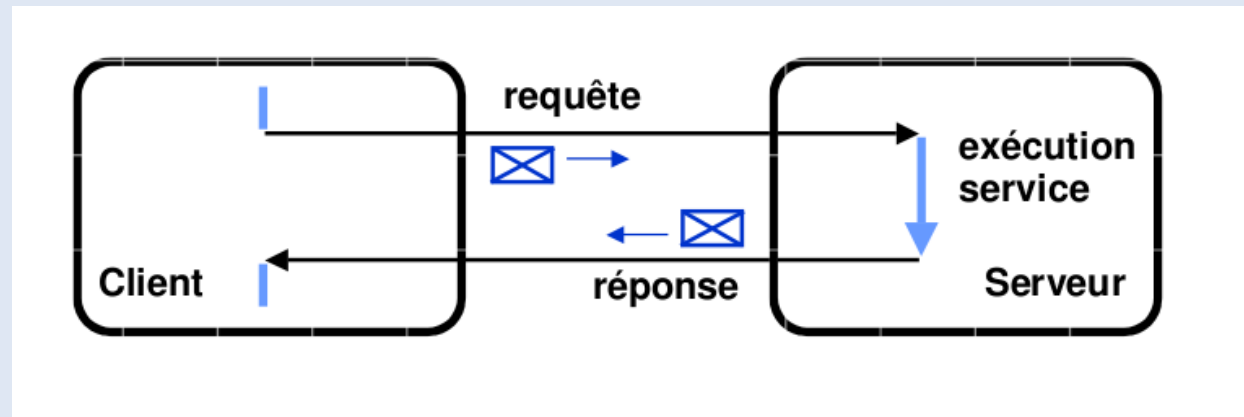


Source : Krakowiak

- Exemples
 - consultation des pages Web (HTTP)
 - envoi & réception des e-mails (SMTP, POP, IMAP)
 - X Window : serveur muni écran (protocole X basé sur IP)
 - NFS (protocole basé sur Sun RPC)

Architecture Client-Serveur

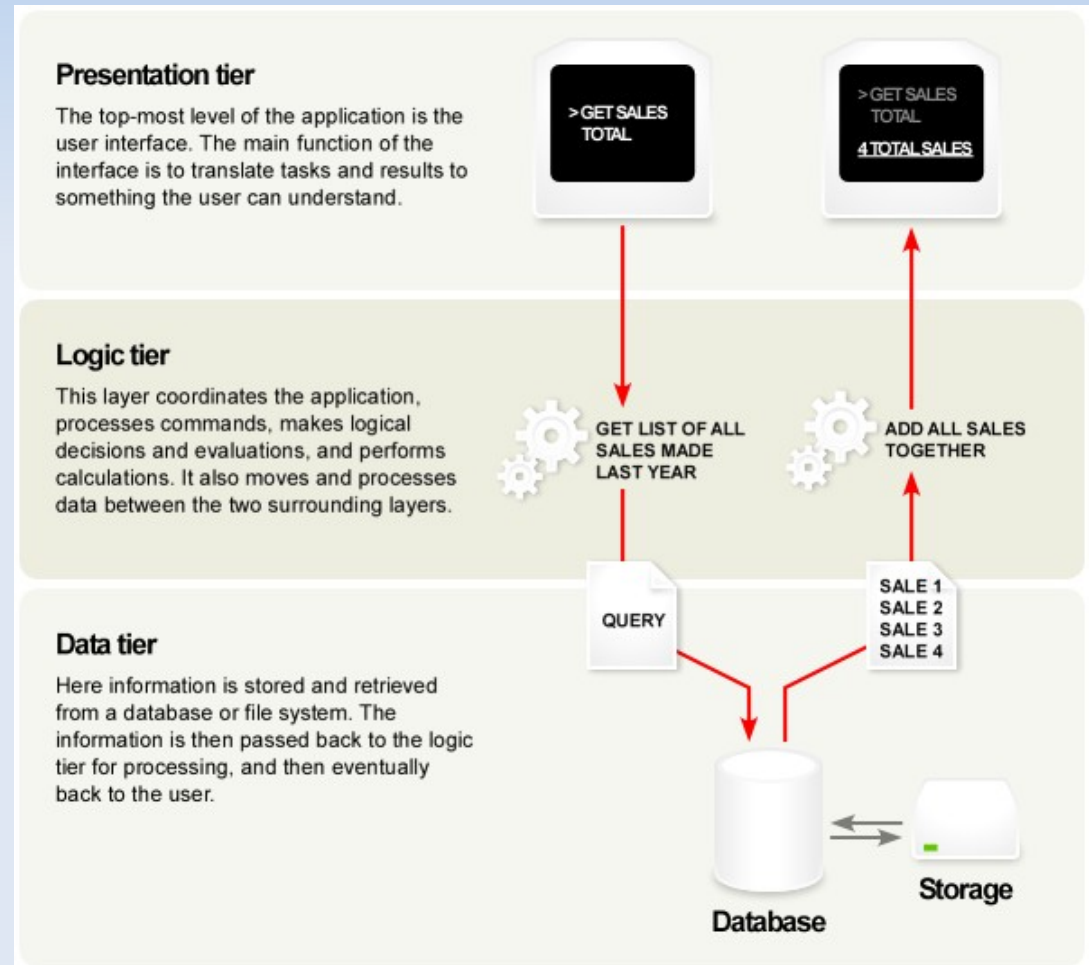
- Communication par échange de messages
 - le client envoie une requête et reste bloqué en attente de la réponse (communication synchrone)
 - le serveur passif attend une requête cliente, exécute le service demandé et renvoie une réponse au client



Source : Krakowiak

Architecture 3-tiers

- Architecture logique à trois étages (“tiers” en anglais)
 - Étage présentation
 - Étage logique ou “métier”
 - Étage données persistantes (SGBD)



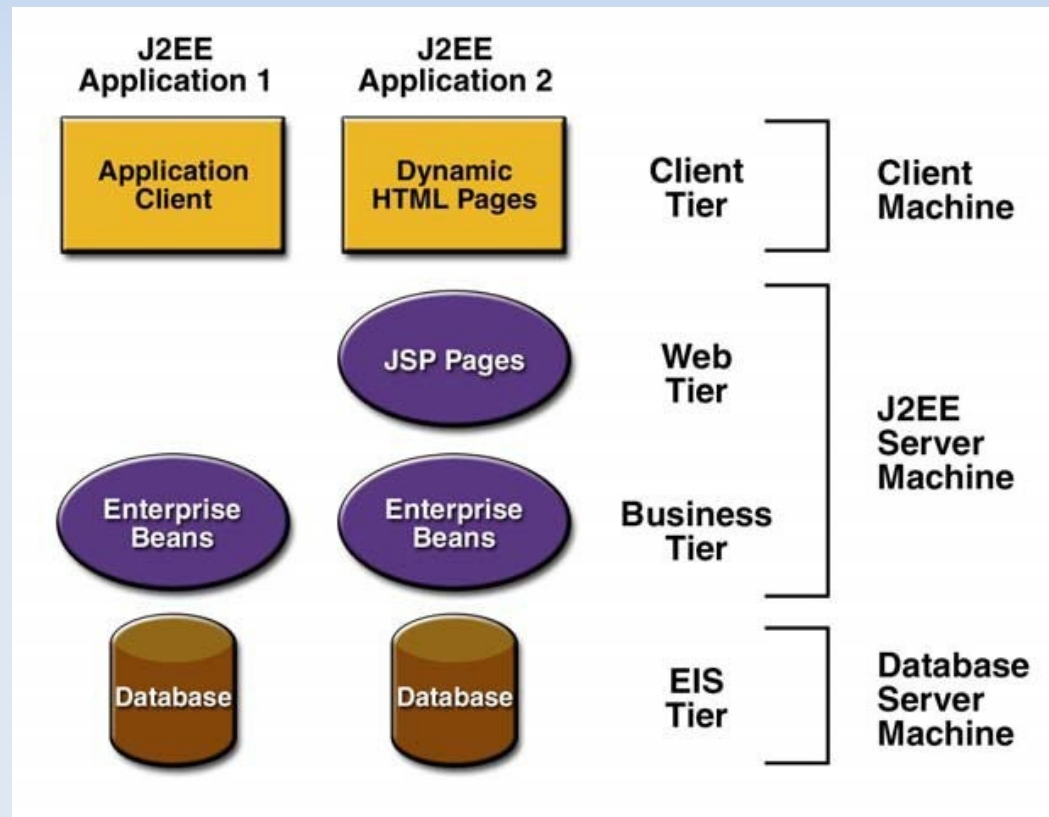
Source : wikipedia

Architecture 3-tiers

- Interface de communication
 - chaque niveau ne communique qu'avec ses voisins immédiats au travers une interface de communication bien définie
 - pas de communications directes entre la couche présentation et données !
- Avantages
 - séparation de la couche présentation et métier, trop souvent imbriquées dans les architectures client/serveur classiques
 - meilleure maintenance : isolement des fonctionnalités (e.g. changer facilement l'IHM)
 - allègement du poste de travail client (par rapport aux clients/serveur de données)

Architecture Multi-Tiers

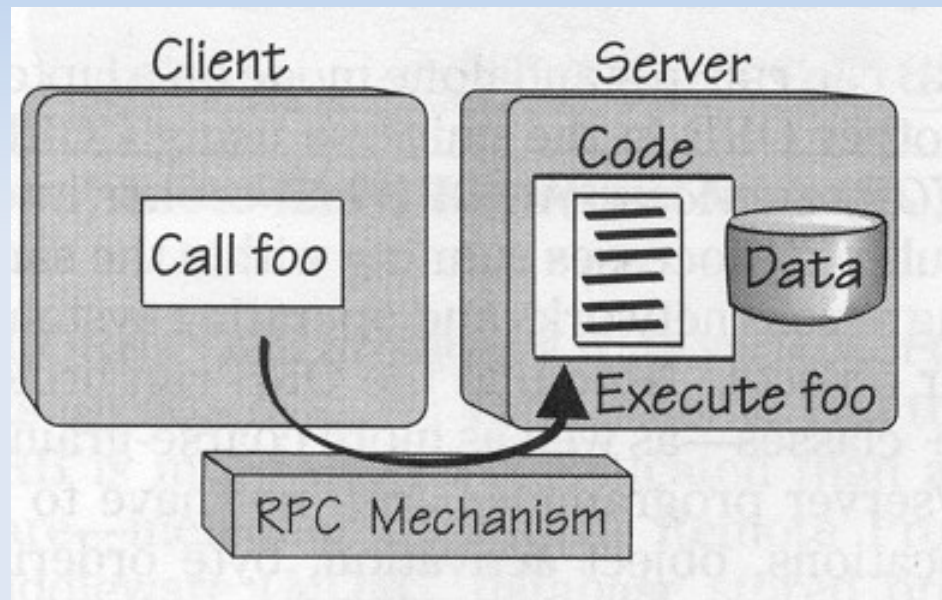
- Extension des architectures 3-tiers, ex. J2EE



Source : Sun

Paradigme RPC

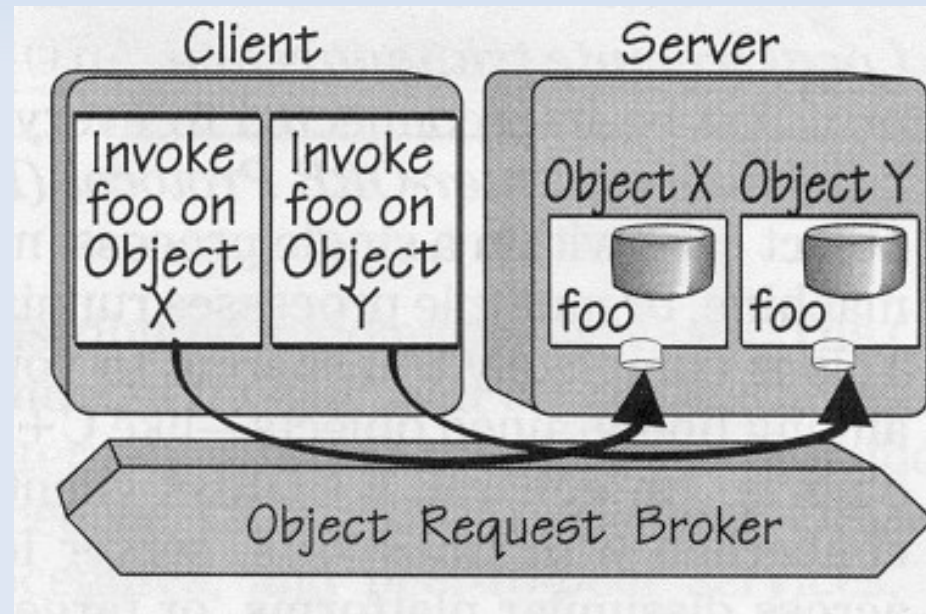
- RPC (Remote Procedure Call)



- Exemples
 - Sun RPC, DCE RPC, XML-RPC, ...

Paradigme RMI

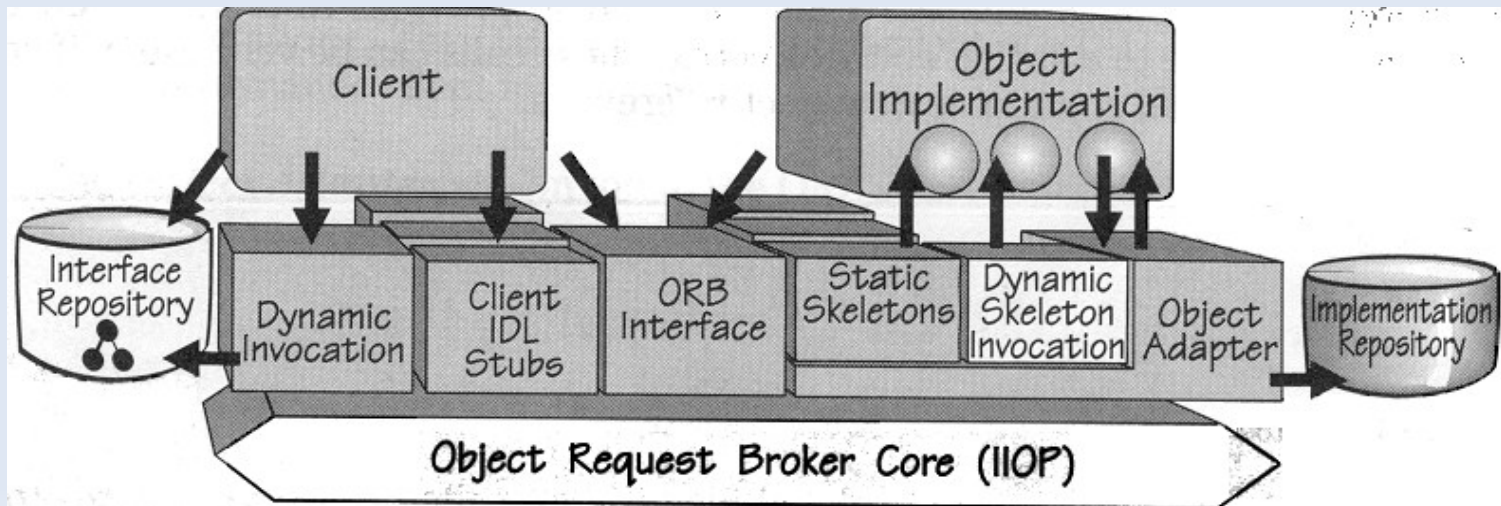
- RMI (Remote Method Invocation)
 - RPC “orientés objet” (encapsulation, héritage, ...)
 - objet : entité de désignation et de distribution



- Exemples
 - SmallTalk, Java RMI, CORBA, ...

Exemple du middleware CORBA

- CORBA (Common Object Request Broker Architecture)
 - standard ouvert pour les applications réparties (OMG, 1995)
 - architecture permettant de faire collaborer des applications réparties sur des environnements différents
 - Points clés
 - client/serveur, orienté objet, bus logiciel (ORB), RMI
 - interopérabilité (machines, systèmes, langages, vendeurs)

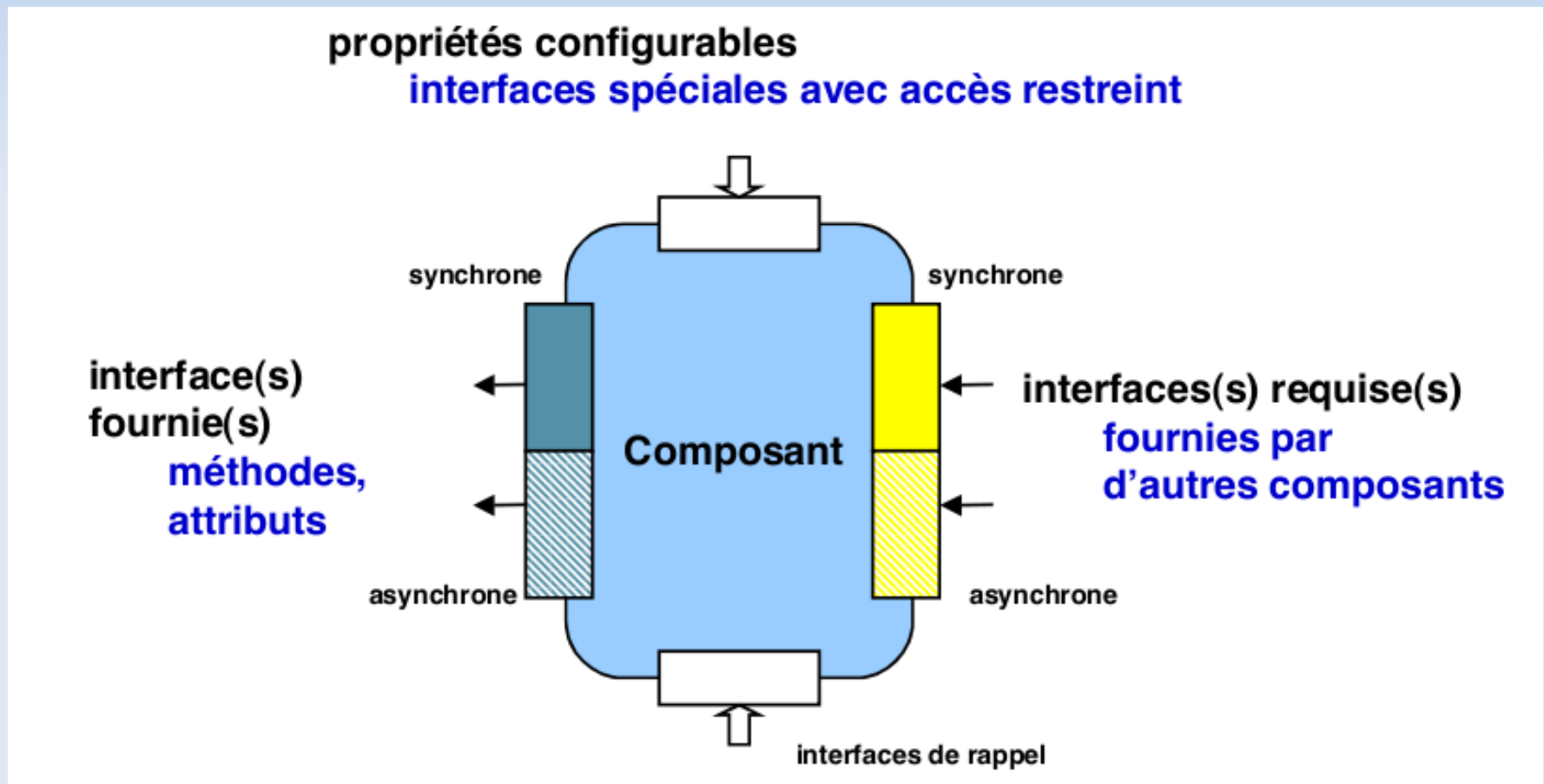


Composants logiciels

- Objectifs
 - diminuer la complexité des applications
 - améliorer la qualité logicielle
 - augmenter la productivité
- Définition
 - *“Un composant logiciel est une unité de composition dotée d'interfaces spécifiées. Un composant logiciel peut être déployé indépendamment et sujet à une composition par une tierce entité.” (Szyperski et Pfister, 1997)*
- Exemples
 - EJB, CCM, DCOM, .NET, ...

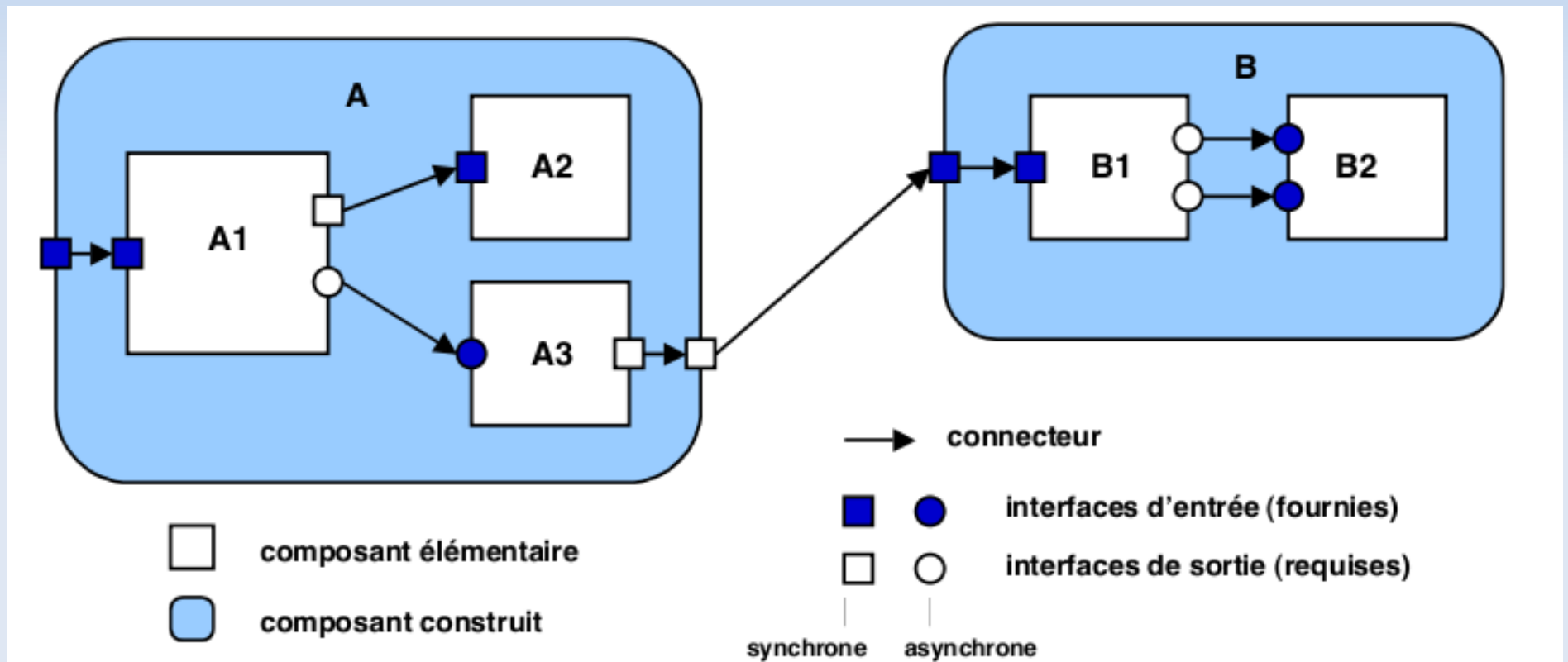
Composants logiciels

- Modèle composant



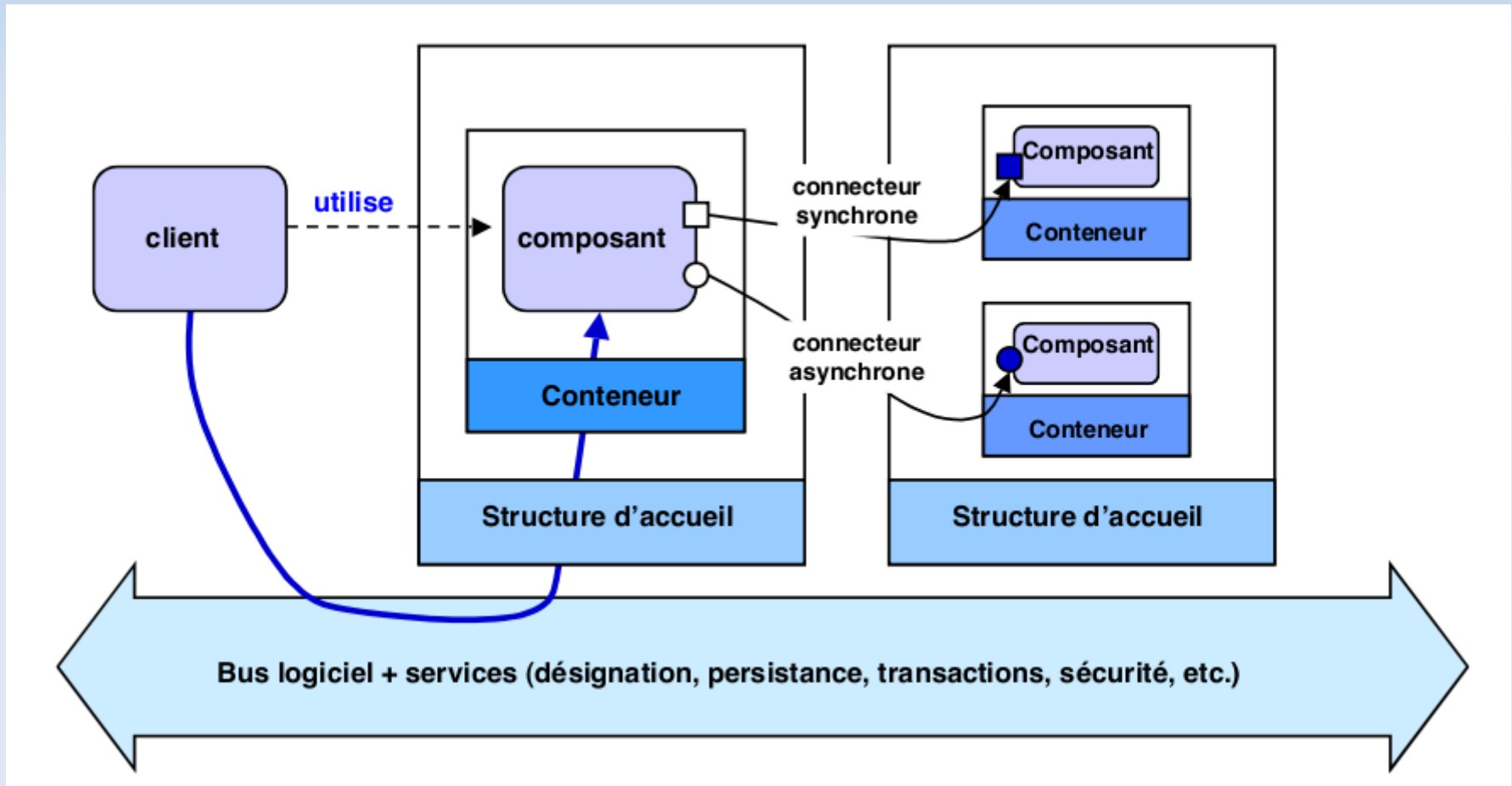
Composants logiciels

- Assemblage de composants (composition)



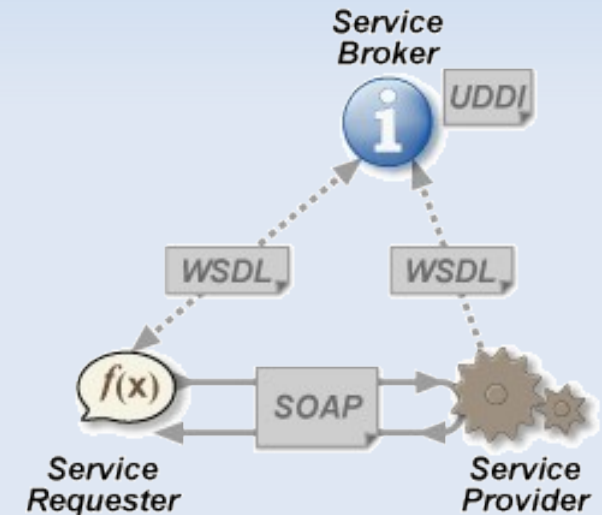
Composants logiciels

- Mise en oeuvre des composants (conteneurs)



Web Services

- Définition W3C
 - “A software system designed to support interoperable Machine to Machine interaction over a network. (...)”
- Technologies
 - SOAP (Simple Object Access Protocol)
 - protocole d'échange de messages XML
 - basé essentiellement sur HTTP/HTTPS
 - paradigme RPC mais pas seulement
 - WSDL (Web Service Description Language)
 - format de description XML des interfaces de service et des protocoles...
 - UDDI (Universal Description, Discovery and Integration)
 - protocole de publication/découverte des méta-données



Source : wikipedia

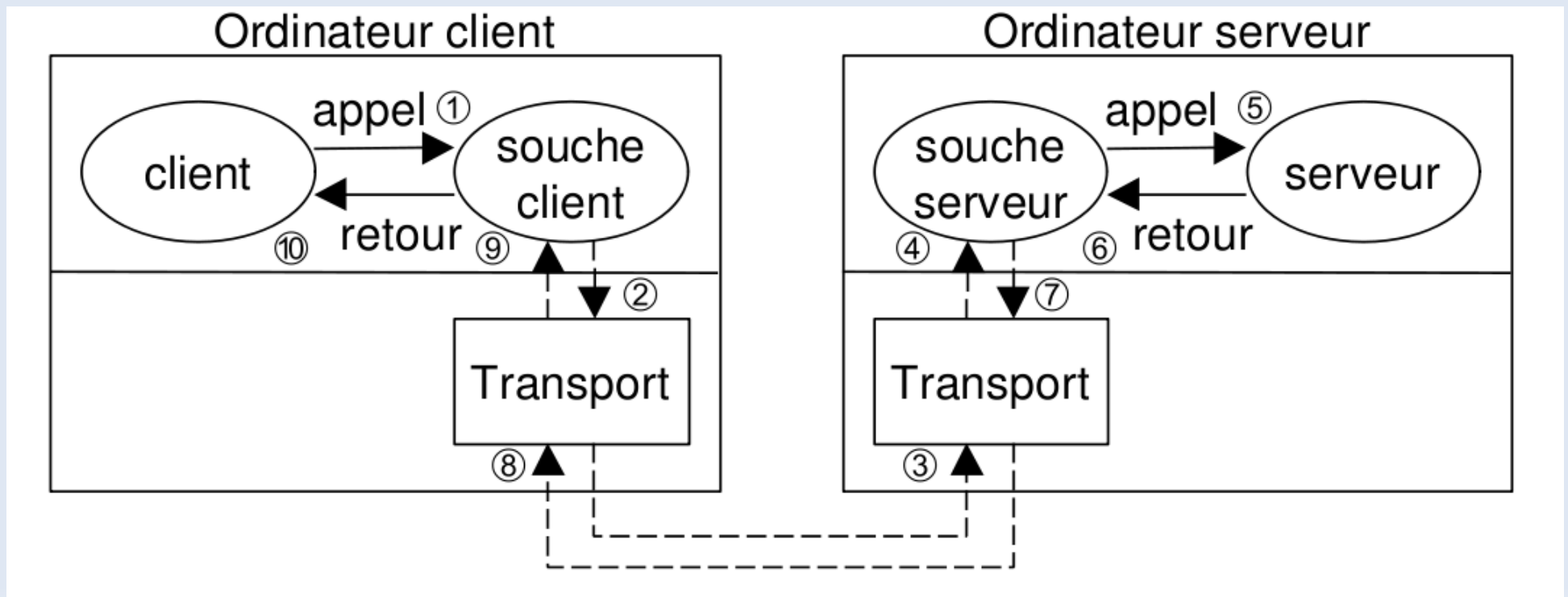
RPC

Introduction à RPC

- Remote Procedure Call (RPC)
 - paradigme de communication permettant d'appeler une procédure à distance (i.e. machine distante)
 - transparence réseau : l'appel de procédure distante se programme de la même manière qu'en local !
- Exemples
 - ONC RPC (Sun), DCE RPC (Open Group), MS RPC (Microsoft)
 - XML-RPC, SOAP
- Applications réparties utilisant les RPCs
 - NFS (Network File System)
 - rstatd / rup (statistique sur une machine distante)
 - rwalld (envoi de message à distance)

Principe du RPC

- Transparence réseau assuré par les souches (stubs)
 - interception local de l'appel de procédure par la souche
 - empaquetage des arguments et transport vers le serveur
 - ...



ONC RPC

- ONC RPC (Open Network Computing RPC)
 - v1 en 1989 (RFC 1057), v2 en 1995 (RFC 1831)
 - sérialisation des données via XDR
 - XDR = eXternal Data Representation (RFC 1832)
 - transport via TCP ou UDP
 - déploiement : accès aux services RPC via le *port mapper* qui écoute sur le port 111
 - outils : rpcgen, rpcinfo, portmap

Service RPC

- Définition d'un service RPC
 - Description d'un programme dans le langage RPCL (fichier .x)
 - Un “programme” fourni plusieurs procédures
 - Un client effectue à distance un appel synchrone/bloquant d'une procédure d'un programme
- Une procédure est identifiée par 3 entiers positifs
 - numéro du programme
 - serveurs officiels : 0x00000000 - 0x1FFFFFFF
 - adresse libres : 0x20000000 - 0x3FFFFFFF
 - numéro de version du programme
 - numéro de procédure au sein du programme

RPCL

- Description d'un programme dans le langage RPCL
 - une procédure ne possède qu'un seul argument en entrée (et en retour)
 - utilisation de structure pour passer ou retourner de multiples arguments
- RPCL : syntaxe proche du C
 - void, int, float, double, ...
 - enum, struct, string, ...
 - tableaux fixes : type[n]
 - tableaux variables : type<n>, type<>
- Exemple

```
struct exemple {  
    int val1;  
    float val2 [10];  
    string val3<10>;  
    float val4[5];  
};
```

Exemple

- Interface “calculation.x”

```
struct two_int {  
    int a;  
    int b;  
};
```

```
program CALCULATION_PROG {
```

```
    version CALCULATION_VERS {
```

```
        int sum(two_int) = 1; ← numéro de procédure
```

```
    } = 1; ← numéro de version
```

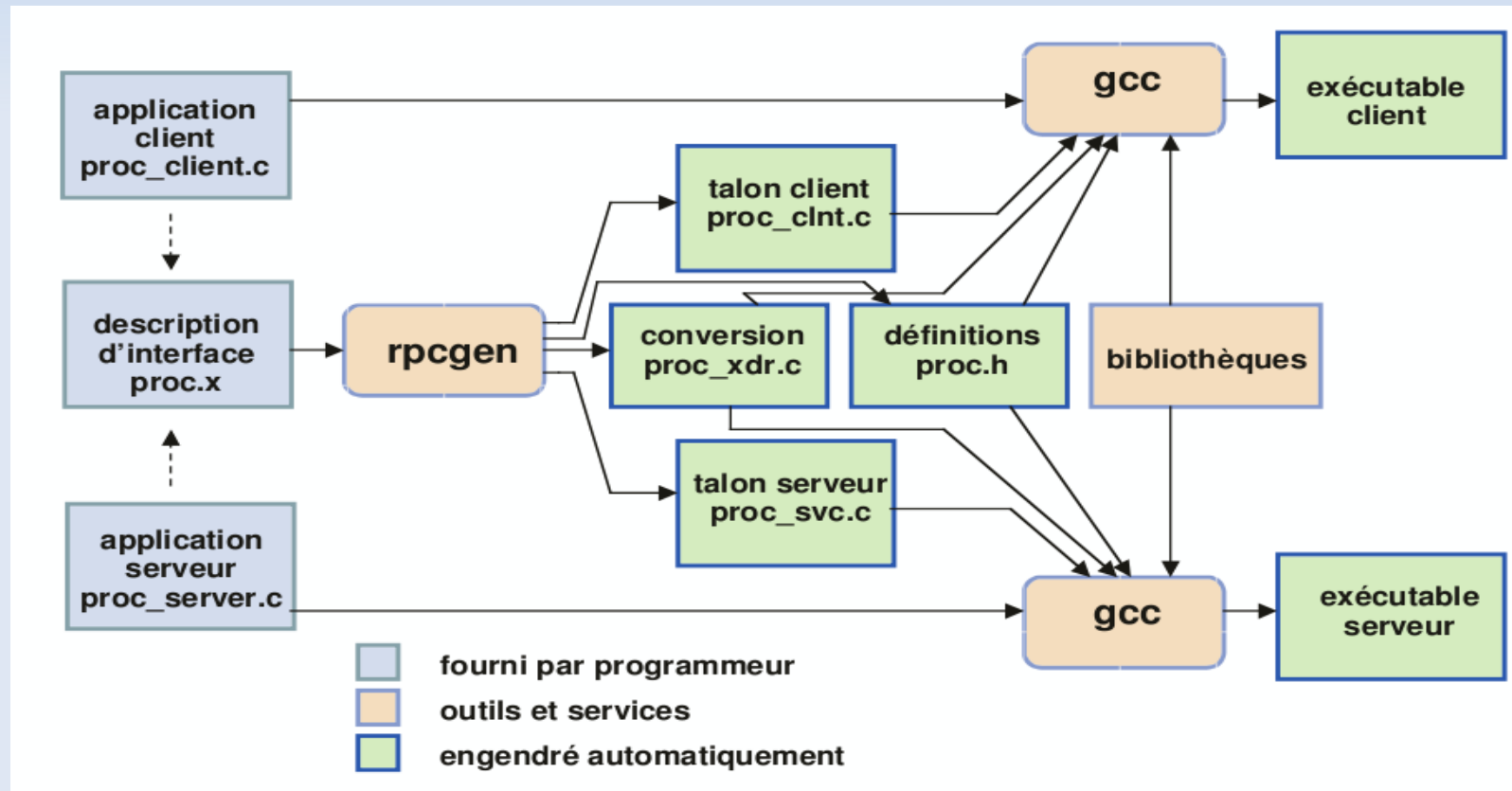
```
} = 0x20000000; ← numéro de programme
```

RPC + XDR

- XDR (eXternal Data Representation)
 - représentation portable des données à travers le réseau
 - encodage des arguments des RPC avec XDR
 - opérations d'empaquetage / dépaquetage des messages
 - basé sur un flux de type XDR (sérialisation)
 - possibilité encoder des types complexes
- Routines de base de la forme `xdr_xxx(XDR*, xxx *, ...)`
 - `xdr_int()`, `xdr_long()`, `xdr_short()`, `xdr_float`, `xdr_double()`
 - `xdr_char()`, `xdr_string()`, `xdr_wrapstring()`
 - `xdr_void()`

Compilation (rpcgen)

- Génération des souches et des fonctions XDR avec rpcgen
- Génération d'un exemple de code client/serveur (rpcgen -a)



Exemple

- En résumé
 - Description du service dans “calculation.x”
 - Fonction de traduction XDR dans “calculation_xdr.c” (**génééré**)
 - Projection du contrat dans “calculation.h” (**génééré**)
 - Côté client
 - calculation_client.c : fonction main(), utilisation du service...
 - calculation_clnt.c (**génééré**): souche cliente
 - Côté serveur
 - calculation_serveur.c : implantation du service
 - calculation_svc.c (**génééré**) : fonction main() + souche serveur

Exemple

- Extrait du code généré “calculation.h”

```
struct two_int {  
    int a;  
    int b;  
};
```

```
typedef struct two_int two_int;
```

```
#define CALCULATION_PROG 0x20000000
```

```
#define CALCULATION_VERS 1
```

```
extern int * sum_1(two_int *, CLIENT *); ← à utiliser côté client...
```

```
extern int * sum_1_svc(two_int *, struct svc_req *); ← à implanter côté serveur...
```

```
/* ... */
```

Exemple

- Code serveur “calculation_server.c”

```
#include "calculation.h"
```

```
int * sum_1_svc(two_int * arg, struct svc_req * req)
{
    static int result;
    result = arg->a + arg->b;
    return &result;
}
```

Exemple

- Code client “calculation_client.c”

```
#include "calculation.h"
```

```
CLIENT *clnt;  
char * host;  
int * result;  
two_int arg;
```

```
clnt = clnt_create (host, CALCULATION_PROG, CALCULATION_VERS, "udp");
```

```
if (clnt == NULL) { clnt_pcreateerror(host); exit(1); }
```

```
result = sum_1(&sarg, clnt); ← appel de la procédure distante...
```

```
if (result == NULL) { clnt_perror (clnt, "call failed"); clnt_destroy (clnt); exit(1); }
```

```
printf("%d+%d=%d\n", arg.a, arg.b, *result);
```

```
clnt_destroy (clnt);
```

Exemple

- Compilation

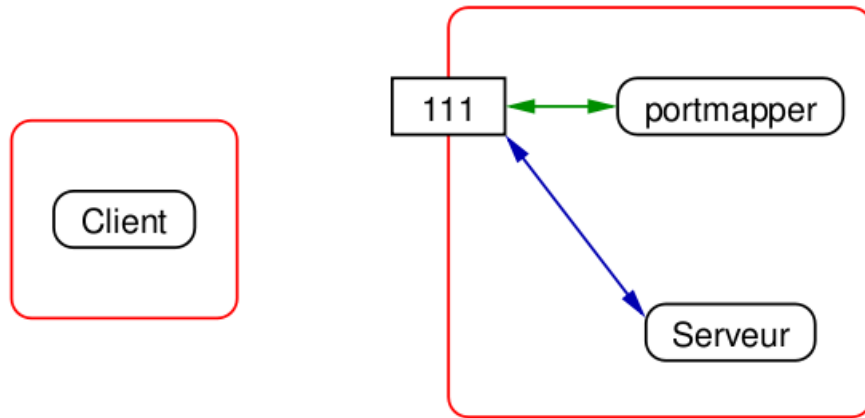
```
rpcgen calculation.x
```

```
cc -g -c *.c
```

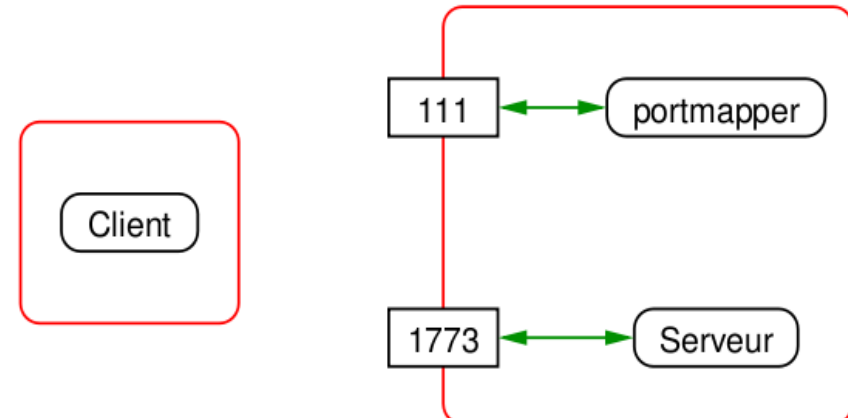
```
cc -g -o calculation_client calculation_clnt.o calculation_xdr.o calculation_client.o -lnsl
```

```
cc -g -o calculation_server calculation_svc.o calculation_xdr.o calculation_server.o  
-lnsl
```

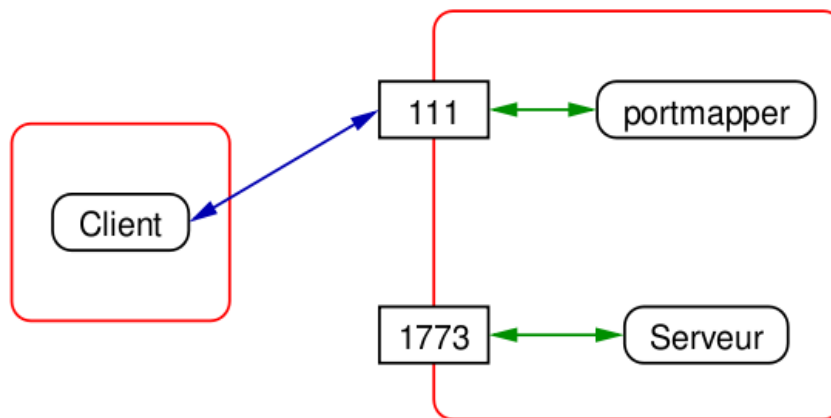
Déploiement : port mapper



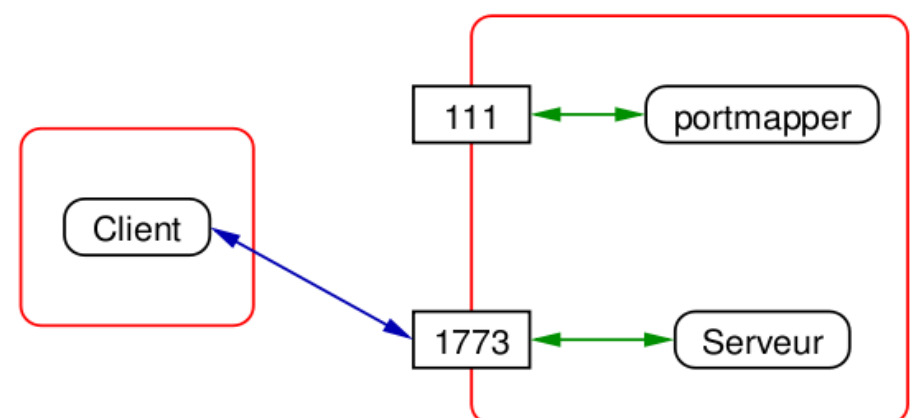
(1) Publication du programme (RPC)



(2) Le programme écoute sur le port 1773



(3) Demande de résolution (RPC)



(4) Appel de la fonction recherchée

rpcinfo

- Interrogation du port mapper
 - Liste des services disponibles sur une machine
 - `rpcinfo -p host`
 - Liste des sites fournissant un service (broadcast)
 - `rpcinfo -b numprog numvers`
- Exemple

```
[orel@kamille:~]$ rpcinfo -p localhost
```

```
program vers proto  port
100000  2  tcp    111  portmapper
100000  2  udp    111  portmapper
100001  1  udp   56558  rstatd
100001  2  udp   56558  rstatd
100001  3  udp   56558  rstatd
100001  4  udp   56558  rstatd
100001  5  udp   56558  rstatd
100008  1  udp   55622  walld
536870912  1  udp   60214
536870912  1  tcp   51749
```

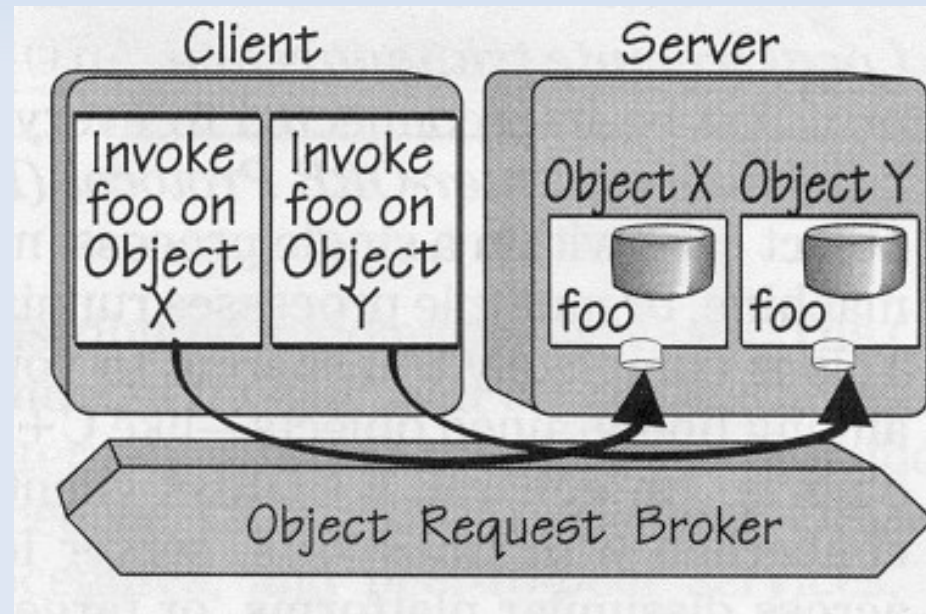


le service calculation

RMI

Paradigme RMI

- RMI (Remote Method Invocation)
 - RPC “orientés objet” (encapsulation, héritage, ...)
 - objet : entité de désignation et de distribution



- Exemples
 - SmallTalk, Java RMI, CORBA, ...

Java RMI

- Des “RPC orientés objets” en Java
 - passage des paramètres par copie lors des appels de méthode à distance (sérialisation)
 - transport TCP (essentiellement, ...)
 - déploiement via le registre de nom “rmiregistry”
- Autres caractéristiques
 - ramasse-miette réparti (Distributed Garbage Collector)
 - intégration des exceptions
 - chargement dynamique de “byte code” (codebase)
 - aspect sécurité, applets, persistance, proxy, serveur http

Java RMI

- Appel synchrone de la méthode distante
 - client bloqué tout le temps de l'appel distant
 - 1 thread par client côté serveur
- Appel asynchrone
 - utilisation explicite d'un thread côté client !
- Appels concurrents côté serveur
 - exclusion mutuelle avec le mot clef “synchronized”

Interface de l'Objet Distribu  (OD)

- Quelques r gles
 - l'interface de l'OD doit  tendre “java.rmi.Remote”
 - les m thodes de l'OD doivent d clarer qu'ils sont susceptibles de lancer des exceptions “java.rmi.RemoteException”
 - les m thodes de l'OD doivent manipuler des types primitifs, “Serializable” (e.g. “String”, ..., d fini par l'utilisateur) ou “Remote”
- Exemple

```
// Hello.java
import java.rmi.Remote;
import java.rmi.RemoteException;

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

HelloServer.java (1/2)

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

```
// implantation de l'objet distribué Hello
```

```
class HelloImpl implements Hello {
```

```
    public String sayHello() throws RemoteException {
```

```
        System.out.println("Remote Invokation of method sayHello()");
```

```
        return "Hello World!";
```

```
    }
```

```
}
```

HelloServer.java

```
public class HelloServer {  
  
    public static void main(String args[]) {  
  
        try {  
            // activation de l'objet distribué  
            Hello hello = new HelloImpl();  
            Hello stub = (Hello) UnicastRemoteObject.exportObject(hello, 0);  
  
            // binding de l'objet distribué dans le registre de nom  
            Registry registry = LocateRegistry.getRegistry();  
            registry.rebind("hello", stub);  
  
        } catch (Exception e) {  
            System.out.println("HelloServer Exception: " + e.getMessage());  
            e.printStackTrace();  
        }  
    }  
}
```

HelloClient.java

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

public class HelloClient {

    public static void main(String args[]) {

        try {
            // recherche de l'objet distribué dans le registre de nom
            Registry registry = LocateRegistry.getRegistry(args[0]); // ← machine serveur
            Hello hello = (Hello) registry.lookup("hello");

            // appel de méthode à distance
            String s = hello.sayHello();
            System.out.println(s);

        } catch (Exception e) {
            System.out.println("HelloClient exception: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```


Registre de Nom

- `rmiregistry [port]`
 - démarre le registre de nom des objets distants sur la machine en cours avec un certain numéro de port (par défaut, port 1099)
- Exemples
 - `rmiregistry &`
 - `rmiregistry 2011 &`

Déploiement

1) Compilation du client et du serveur RMI

```
HostA> javac *.java
```

2) Démarrage du registre de nom RMI (par défaut port 1099)

```
HostA> rmiregistry 5555 &
```

3) Démarrage du serveur (sur la même machine que le registre de nom)

```
HostA> java HelloServer 5555
```

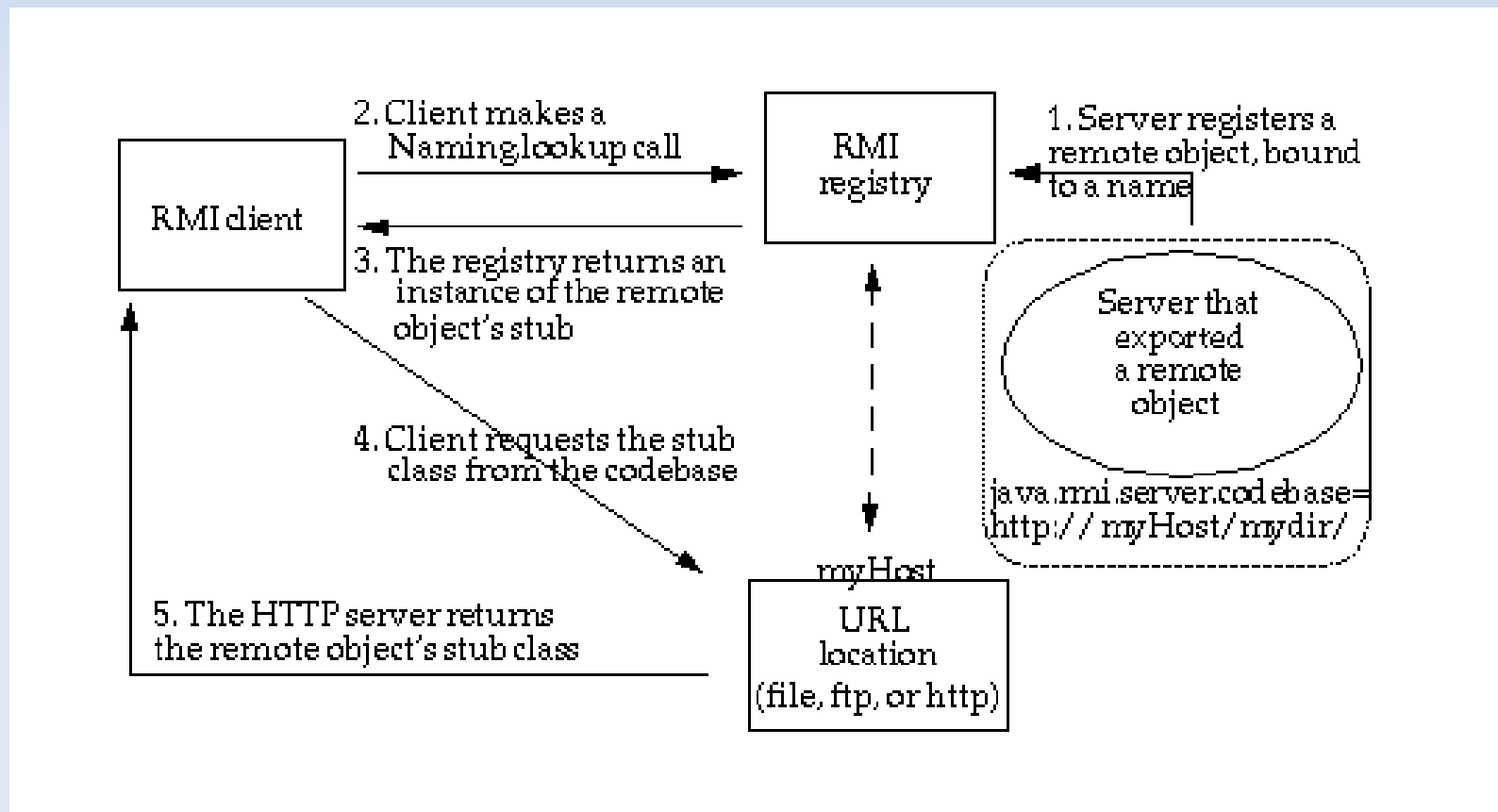
4) Démarrage du client RMI (en local et à distance)

```
HostA> java HelloClient localhost:5555
```

```
HostB> java HelloClient HostA:5555
```

Téléchargement des stubs

- Le client ne connaît que l'interface Remote et ne dispose pas par avance des stubs...

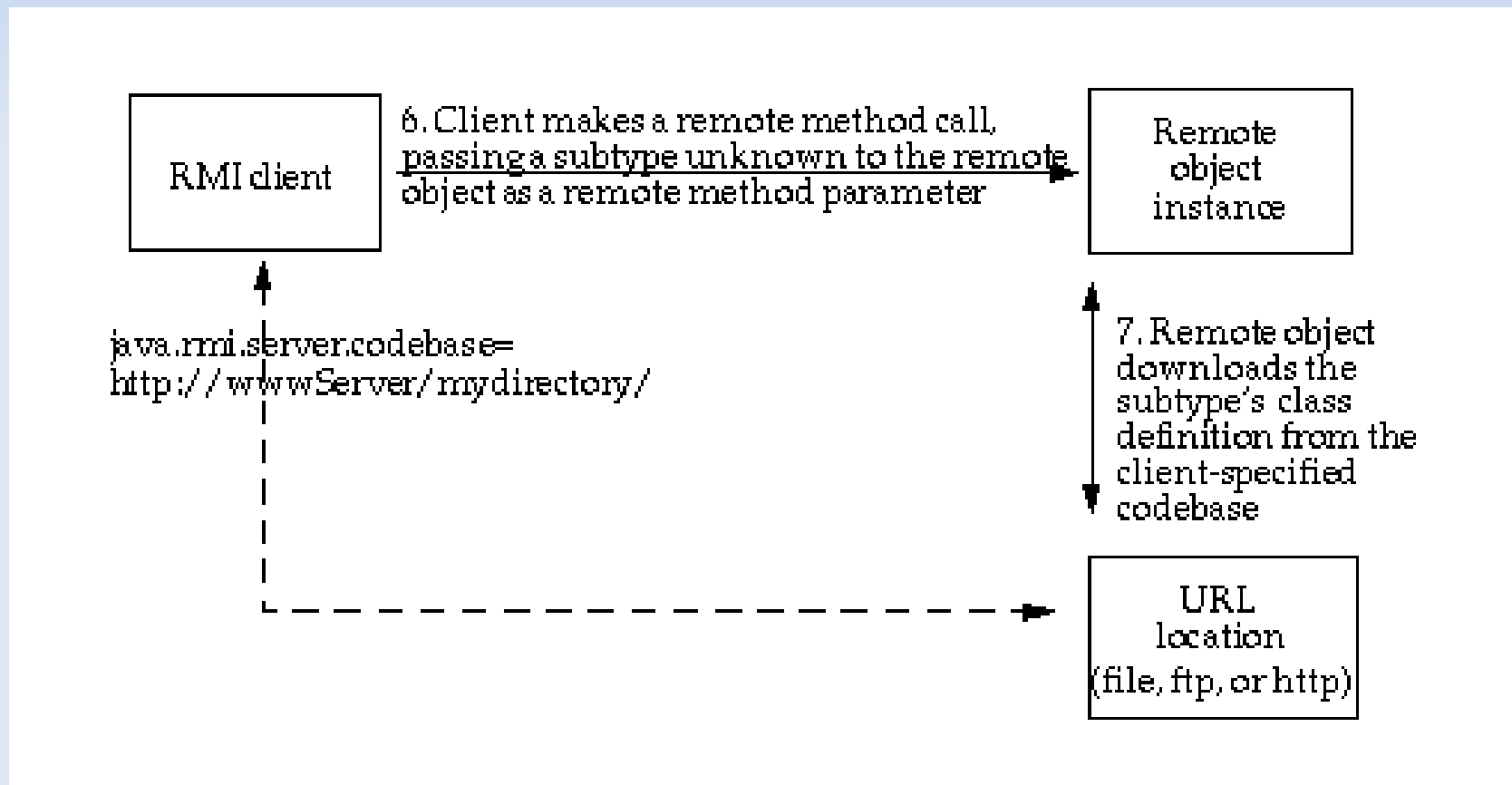


Source : Sun

Le RMI Registry doit pouvoir accéder aux stubs directement ou via le codebase...

Téléchargement d'un sous-type

- Le serveur manipule un sous-type défini uniquement par le serveur...



Source : Sun

Le client doit indiquer via un codebase les classes utiles au serveur...

Security Manager et Codebase

- Utilisation d'un Security Manager

```
if (System.getSecurityManager() == null) {  
    System.setSecurityManager(new SecurityManager());  
}
```

- Politique de sécurité (fichier all.policy)

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

- Prise en compte de la politique de sécurité

```
java -Djava.security.policy=all.policy ...
```

- Utilisation du codebase avec HTTP (extension du classpath)

```
java ... -Djava.rmi.server.codebase=http://webserver/classes/
```

CORBA

OMG et CORBA

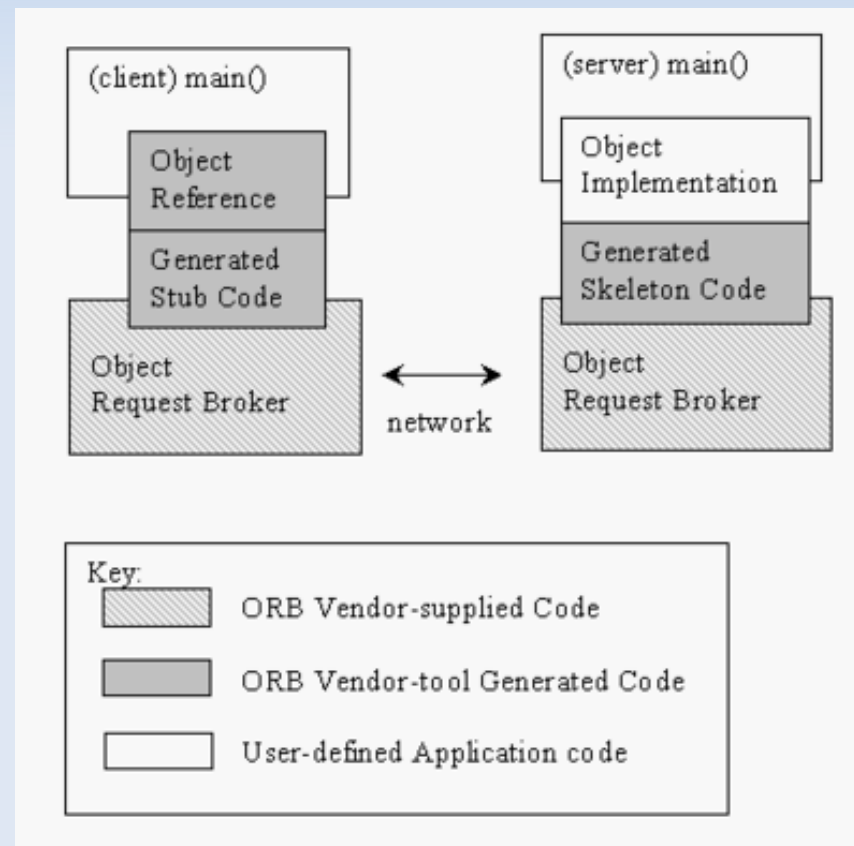
- **OMG (Object Management Group)**
 - consortium responsable de la spécification de CORBA, regroupant plus de 800 entreprises du monde entier
- **CORBA (Common Object Request Broker Architecture)**
 - spécification qui décrit une architecture permettant de faire collaborer des applications réparties sur des machines, des environnements et des langages différents
- **Historique**
 - CORBA 1.0 : 1991 (modèle objet)
 - CORBA 2.0 : 1995 (interopérabilité, IIOP)
 - CORBA 3.0 : 2002 (modèle composant)

Caractéristiques

- Standard ouvert pour les applications réparties
 - bus logiciel à requêtes (ORB)
 - architecture client/serveur
 - orienté objet
 - communication par appel de méthode à distance (RMI)
 - interopérabilité :
 - indépendant du matériel
 - indépendant du système
 - indépendant des langages
 - indépendant des “vendeurs”

ORB

- Élément “central” de l’architecture CORBA en charge d’assurer la collaboration entre les applications



Source : Wikipedia

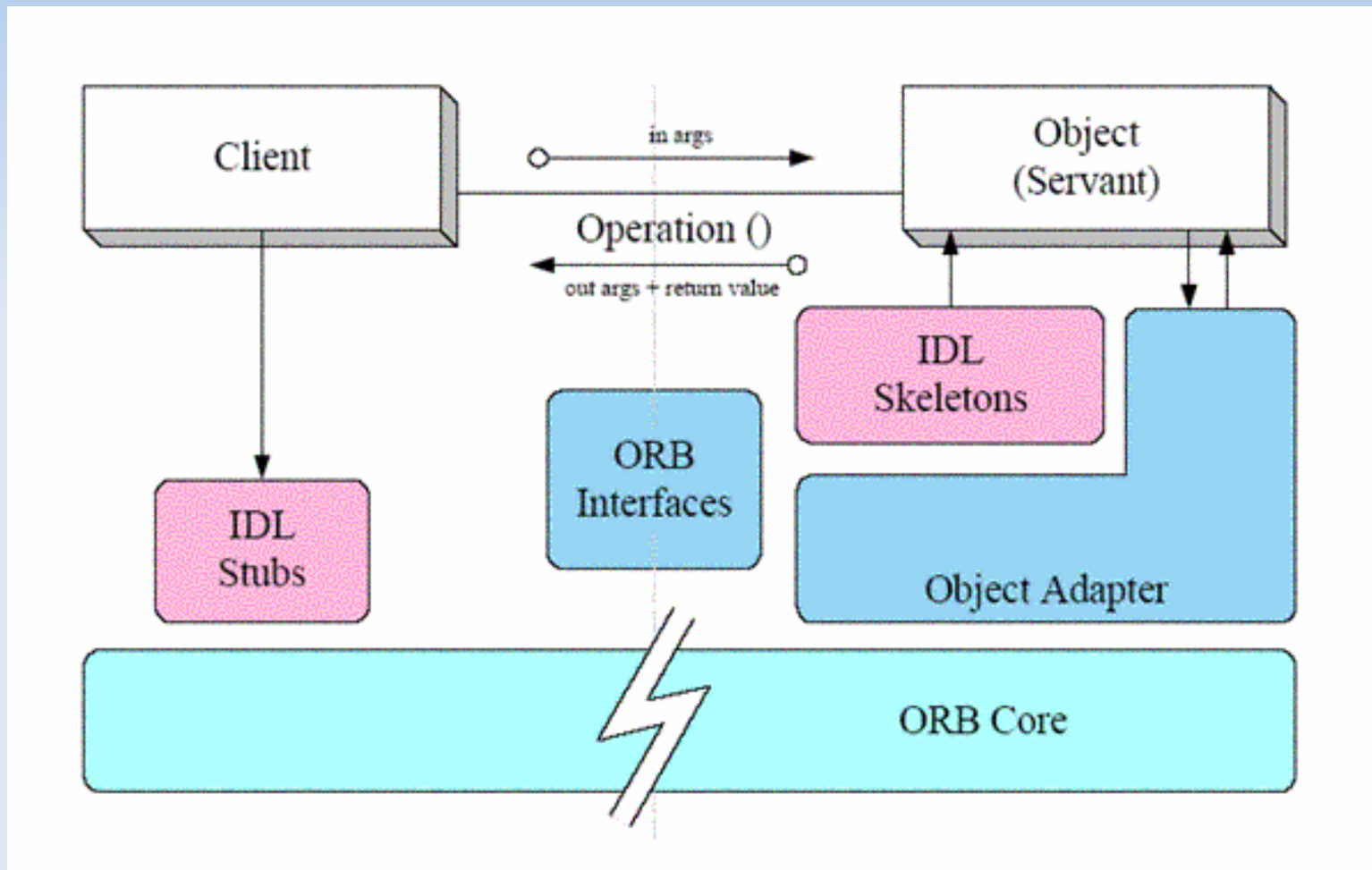
Objet CORBA

- Définition
 - entité logicielle désignée par une référence et recevant les requêtes émises par les applications clientes
- Un objet CORBA comporte
 - une référence ou IOR, localisant l'objet sur le réseau
 - une interface IDL
 - une implantation (le servant)

Protocole IIOP

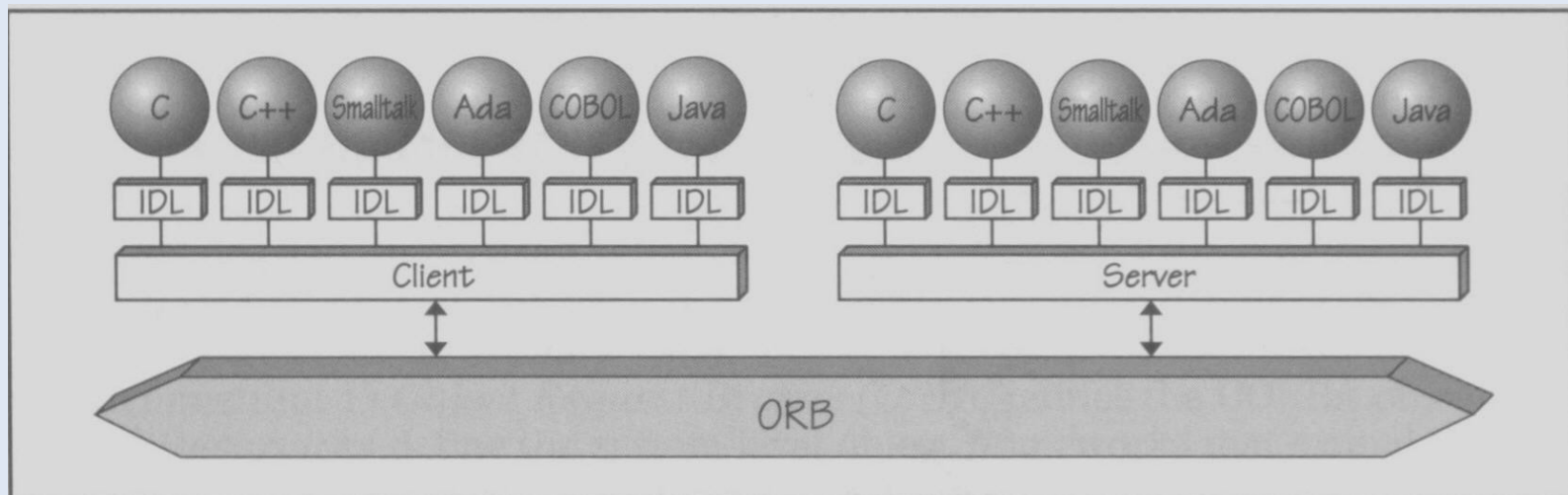
- GIOP (General Inter-ORB Protocol)
 - spécification du protocole générique de transport de CORBA
 - représentation commune des données (CDR)
 - format des références d'objet (IOR)
 - transport des messages (transparence réseaux)
- IIOP (Internet Inter-ORB Protocol)
 - implantation de GIOP au dessus de TCP/IP
 - GIOP/IIOP permet l'interopérabilité entre des ORBs provenant de “vendeurs” différents

Vue d'ensemble



IDL

- IDL (Interface Description Language)
 - définir un contrat entre les fournisseurs et les utilisateurs de services, en séparant l'interface de l'implantation des objets
 - langage neutre, avec une syntaxe proche du C++
 - masquer les divers problèmes liés à l'hétérogénéité
 - projection possible vers de nombreux langages



IDL

- modules
 - `module NAME { ... } ;`
- interfaces
 - `interface NAME { ... } ;`
- constantes
 - `const TYPE NAME = VALEUR ;`
- types énumérés
 - `enum NAME { v1, ... , vn } ;`
- nouveaux types
 - `typedef DECL NAME ;`

IDL

- structures
 - `struct NAME { DECL } ;`
- attributs de l'interface
 - `[readonly] attribute TYPE NAME ;`
- opérations de l'interface
 - `TYPE NAME(ARG1, ARG2, ...) ;`
 - `TYPE NAME(ARG1, ARG2, ...) raises (EXCP1, EXCP2, ...) ;`
 - `ARGi = [in | inout | out] TYPE NAME`
- exceptions
 - `exception NAME { ... } ;`

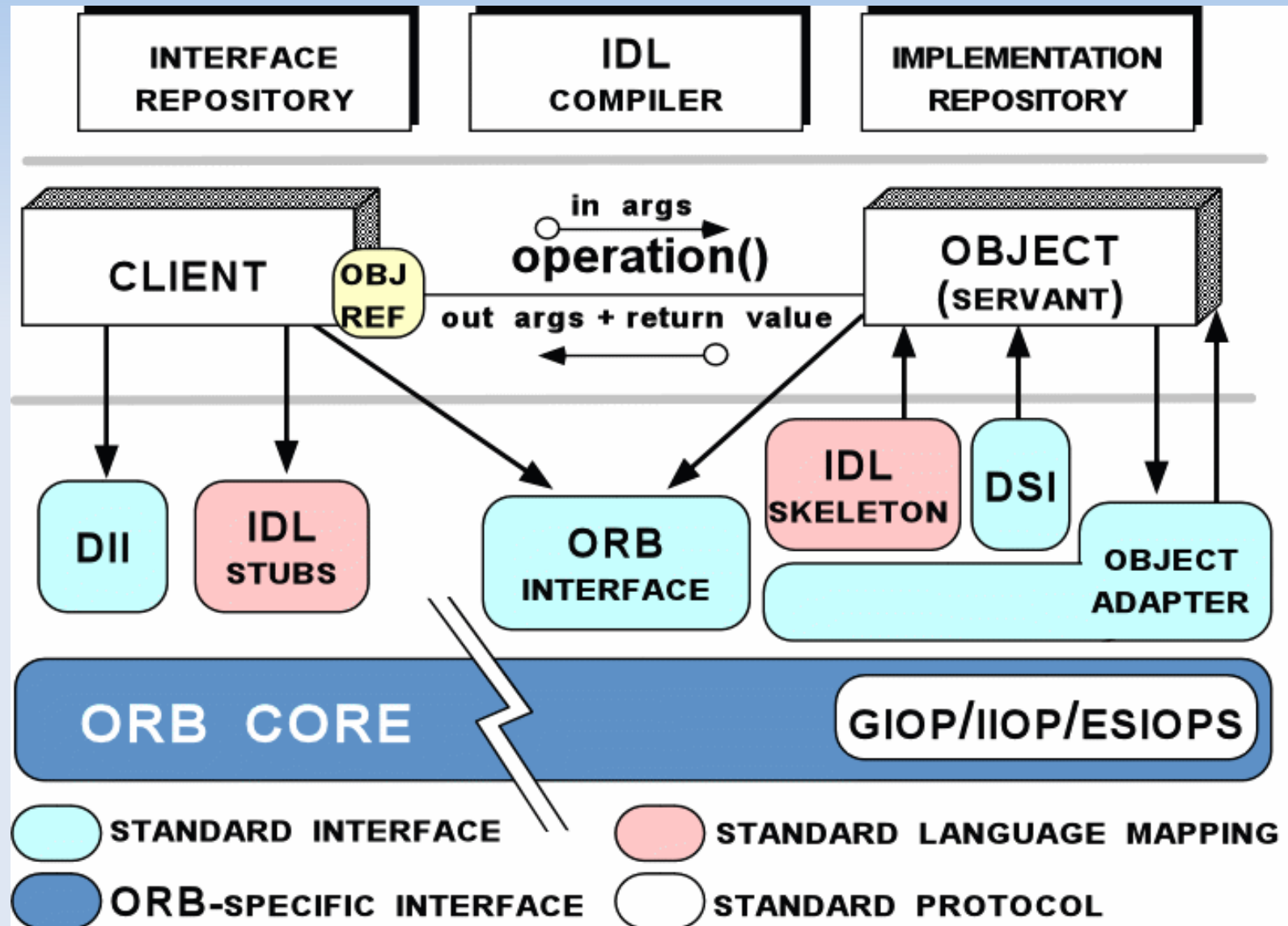
- Types primitifs
 - void
 - short, unsigned short : entiers sur 16 bits
 - long, unsigned long : entiers sur 32 bits
 - long long, unsigned long long : entiers sur 64 bits
 - float : nombres flottants 32 bits au format standard IEEE
 - double : nombres flottants 64 bits au format standard IEEE
 - long double : nombres flottants 128 bits
 - boolean : valeurs booléennes FALSE ou TRUE
 - octet : 8 bits
 - string : chaînes de caractères

Exemple IDL

```
module ServiceDate {  
  
    typedef unsigned short Jour;  
  
    enum Mois {  
        Janvier, Fevrier, Mars, Avril, Mai, Juin,  
        Juillet, Aout, Septembre, Octobre,  
        Novembre, Decembre  
    };  
  
    typedef unsigned short Annee;  
  
    struct Date {  
        Jour j;  
        Mois m;  
        Annee a;  
    };  
  
    typedef sequence<Date> DesDates;  
  
    exception MauvaiseDate{  
        string raison;  
    };  
};
```

```
interface Calendrier {  
    attribute Annee annee_courante;  
    boolean verifier_date(in Date d);  
    void jour_suivant(inout Date d);  
    Date convertir_chaine(in string chaine)  
        raises (MauvaiseDate);  
    string convertir_date(in Date d)  
        raises (MauvaiseDate);  
};  
  
interface CalendrierFerie : Calendrier {  
    void jours_feries(in Annee a,  
        out DesDates dates);  
};  
};
```

Anatomie de CORBA

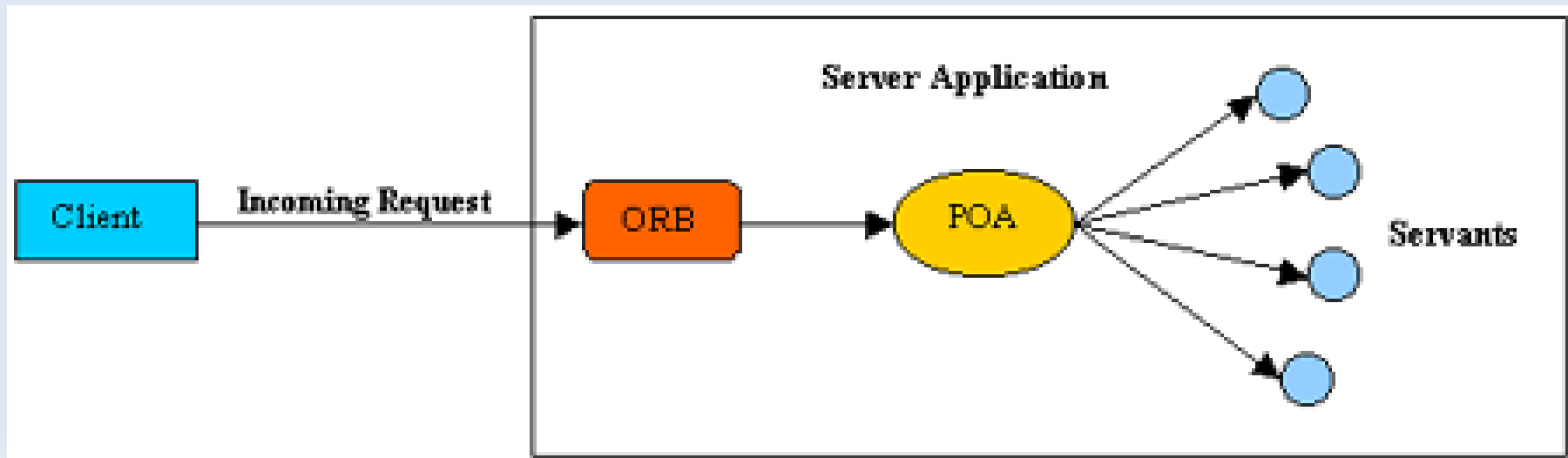


Anatomie de CORBA

- ORB Interface
 - l'interface des fonctionnalités communes à tout le système
- Souche cliente (IDL Stub)
 - chaque stub contient le code (généré) pour accéder à un type d'objets distants d'une manière dépendant du langage utilisé
- Squelette d'implantation (IDL Skeleton)
 - chaque squelette fournit une interface au travers duquel l'ORB invoque une opération sur l'objet
- Servant
 - implantation de l'objet côté serveur

Anatomie de CORBA

- Adaptateur d'objets
 - Pour adapter un modèle d'objet spécifique à un langage au modèle de l'ORB.
 - Gestion de l'activation, la désactivation, la création des objets, la gestion des références, ...
 - Durée de vie des objets : *transient*, *persistent*
 - BOA (Basic Object Adapter), POA (Portable Object Adapter)

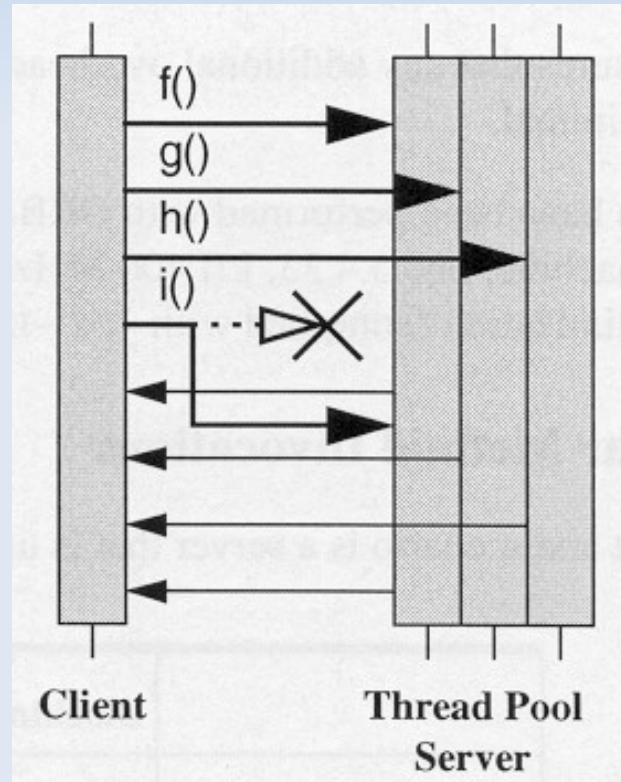


Anatomie de CORBA

- L'interface d'invocation dynamique (DII)
 - à la place d'utiliser les stubs, le client peut dynamiquement construire et invoquer des requêtes
- L'interface de squelette dynamique (DSI)
 - même chose côté serveur
- Divers
 - Interface Repository, Implementation Repository
- COS (Common Object Services)
 - Naming Service, Trading Service, Event Service, ...
 - Externalization Service, Lifecycle Service, Persistent State Service, ...

Modèle d'exécution

- Modèle "thread pool"
 - Mais d'autres modèles disponibles selon les ORBs...



Exemple : Hello World !

- Interface IDL

```
interface Echo
{
    string echoString(in string mesg);
};
```

- Implantation du client et du serveur

- en Java (IDL Sun) : EchoClient.java / EchoServer.java
- en C++ (Mico ou OmniORB) : EchoClient.cc / EchoServer.cc

EchoServer.java (1/3)

```
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;
import java.util.Properties;
```

*doit hériter de EchoPOA, une classe
générée par le compilateur IDL*

```
class EchoImpl extends EchoPOA {

    public String echoString(String msg)
    {
        System.out.println("msg: " + msg);
        return msg;
    }

}
```

*objet d'implantation (ou servant)
traduction en Java de l'IDL*

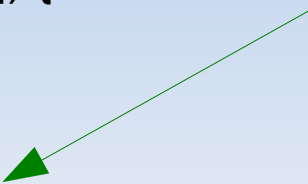
../..

EchoServer.java (2/3)

```
public class EchoServer {
```

```
    public static void main(String args[]) {  
        org.omg.CORBA.Object objRef;
```

*args : contient la localisation
du service de nommage*



```
        // create and initialize the ORB  
        ORB orb = ORB.init(args, null);
```

```
        // get reference to rootpoa & activate the POAManager  
        objRef = orb.resolve_initial_references("RootPOA");  
        POA rootpoa = POAHelper.narrow(objRef);  
        rootpoa.the_POAManager().activate();
```

```
        // get the naming service  
        objRef = orb.resolve_initial_references("NameService");  
        NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
```

../..

EchoServer.java (3/3)

```
// get object reference from the servant
EchoImpl echolmpl = new EchoImpl();
objRef = rootpoa.servant_to_reference(echolmpl);

// convert the CORBA object reference into Echo reference
Echo echoRef = EchoHelper.narrow(objRef);

// bind the object reference in the naming service
NameComponent path[ ] = ncRef.to_name("echo.echo"); // id.kind
ncRef.rebind(path, echoRef);

// start server...
orb.run();
}
}
```

activation de l'objet distant

inscription dans le service de nommage

démarrage du serveur (appel bloquant)

EchoClient.java (1/2)

```
import org.omg.CosNaming.*;  
import org.omg.CosNaming.NamingContextPackage.*;  
import org.omg.CORBA.*;
```

```
public class EchoClient  
{
```

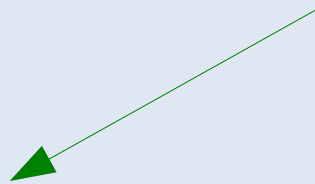
```
    public static void main(String args[])  
    {  
        org.omg.CORBA.Object objRef;
```

```
        // create and initialize the ORB  
        ORB orb = ORB.init(args, null);
```

```
        // get the naming service
```

```
        objRef = orb.resolve_initial_references("NameService");  
        NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
```

*args : contient la localisation
du service de nommage*



../..

EchoClient.java (2/2)

```
// resolve the object reference from the naming service
```

```
objRef = ncRef.resolve_str("echo.echo");
```

```
// convert the CORBA object reference into Echo reference
```

```
Echo echoRef = EchoHelper.narrow(objRef);
```

```
// remote method invocation
```

```
String response = echoRef.echoString("coucou");
```

```
System.out.println(response);
```

```
}  
}
```

← *invocation à distance*

Compilation

- Génération des souches et squelettes Java

```
idlj -fall Echo.idl
```

- Fichiers générés

- projection de l'interface IDL en Java : EchoOperations.java

- souche cliente : EchoStub.java

- squelette serveur : EchoPOA.java

- divers : EchoHolder.java, EchoHelper.java

- Compilation

```
javac *.java
```

Déploiement

- Démarrage du service de nommage...

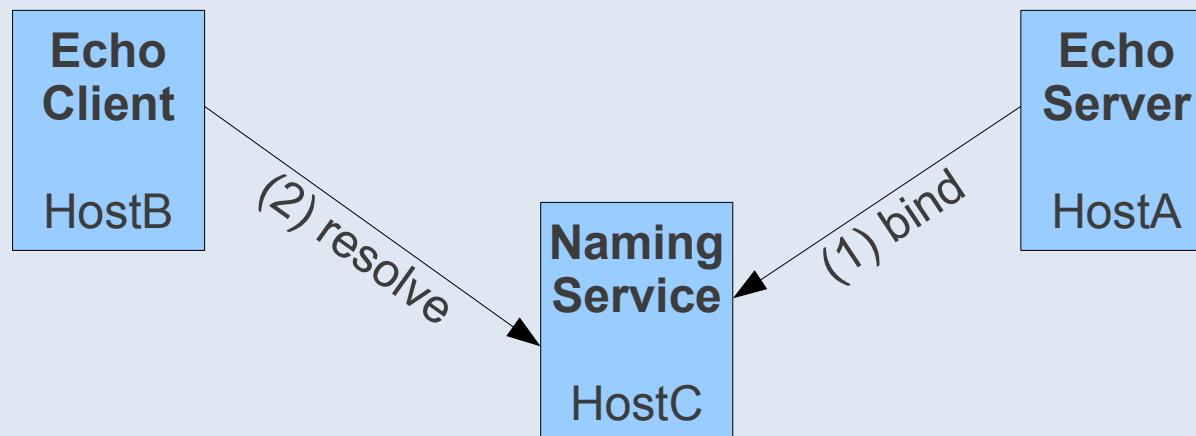
HostC\$ tnameserv -ORBInitialPort 12456

- Démarrage du serveur...

HostA\$ java EchoServer -ORBInitRef NameService=corbaloc::HostC:12456/NameService

- Démarrage du client...

HostB\$ java EchoClient -ORBInitRef NameService=corbaloc::HostC:12456/NameService



EchoServer.cc (1/3)

```
#include <CORBA.h>
#include <coss/CosNaming.h>
#include "echo.h"
using namespace std;
```

*doit hériter de POA_Echo, une classe
générée par le compilateur IDL*

```
class EchoImpl : public POA_Echo,
public PortableServer::RefCountServantBase
{
public:

virtual char* echoString(const char* msg) {
    cout << msg << endl;
    return CORBA::string_dup(msg);
}

};
```

*objet d'implantation (ou servant)
traduction en C++ de l'IDL*

.../...

EchoServer.cc (2/3)

```
int main(int argc, char** argv) {
    CORBA::Object_var objRef;

    // create and initialize the ORB
    CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);

    // get reference to rootpoa & activate the POAManager
    objRef = orb->resolve_initial_references("RootPOA");
    PortableServer::POA_var poaRef = PortableServer::POA::_narrow(objRef);
    poaRef->the_POAManager()->activate();

    // get the naming service
    objRef = orb->resolve_initial_references("NameService");
    CosNaming::NamingContext_var ncRef =
        CosNaming::NamingContext::_narrow(objRef);

    // instantiate the Echo CORBA object
    EchoImpl * echoImpl = new EchoImpl( );
    Echo_var echoRef = echoImpl->_this();
```

activation de l'objet distant

../..

EchoServer.cc (3/3)

// bind the object reference in the naming service

```
CosNaming::Name name;
```

```
name.length(1);
```

```
name[0].id = (const char*)"echo";
```

```
name[0].kind = (const char*)"echo";
```

```
ncRef->rebind(name, echoRef);
```

← *inscription dans le service de nommage*

// start server...

```
orb->run();
```

← *démarrage du serveur (appel bloquant)*

```
return EXIT_SUCCESS;
```

```
}
```

EchoClient.cc (1/2)

```
#include <CORBA.h>
#include <coss/CosNaming.h>
#include "echo.h"

int main (int argc, char **argv)
{
    CORBA::Object_var objRef;

    // initialize the ORB
    CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);

    // get the naming service
    objRef = orb->resolve_initial_references("NameService");
    CosNaming::NamingContext_var nsRef =
        CosNaming::NamingContext::_narrow(objRef);
```

../..

EchoClient.cc (2/2)

```
// resolve the "echo" CORBA object from the naming service
```

```
CosNaming::Name name; name.length(1);  
name[0].id = (const char*) "echo";  
name[0].kind = (const char*) "echo";  
objRef = nsRef->resolve(name);  
Echo_var echoRef = Echo::_narrow(objRef);
```

```
// remote method invocation
```

```
cout << echoRef->echoString("coucou") << endl; ← invocation à distance
```

```
return 0;  
}
```

Compléments : les séquences

- Séquence

- tableau extensible en CORBA (type IDL `sequence<>`)

- Exemple IDL

```
typedef sequence<long> SeqLong;  
interface Test { void test(in SeqLong seq); };
```

- Exemple en C++

```
SeqLong seq;  
seq.length(5);  
for(int i = 0 ; i < seq.length() ; i++)  
    seq[i] = i;
```

- Exemple en Java

```
int [ ] seq = new int[5];  
for(int i = 0 ; i < seq.length ; i++)  
    seq[i] = i;
```

Compléments : le type any

- Le type any
 - Pas de surcharge possible des méthodes dans l'interface IDL, mais possibilité d'utiliser un type générique, le type IDL any.

- Exemple IDL

```
struct Personne {string nom; string prenom; long age; };  
interface Test { void test(in any arg); };
```

- Insertion en Java

```
org.omg.CORBA.Any arg = orb.create_any();  
arg.insert_long(6);  
arg.insert_string("par exemple");  
Personne p = new Personne;  
p.nom = "esnard"; p.prenom= "aurelien";  
PersonneHelper.insert(arg, p);
```

- Extraction en Java

```
org.omg.CORBA.TypeCode tc = arg.type();  
if(tc.kind().value() == TCKind._tk_long)  
    int x = arg.extract_long();  
if(tc.kind().value() == TCKind._tk_string)  
    String s = arg.extract_string();  
if(tc.equal(PersonneHelper.type()))  
    Personne p = PersonneHelper.extract(arg);
```

Compléments : le type XXX_var

- Pointeur “intelligent” CORBA/C++
 - facilite la gestion de la mémoire (compteur de référence)
- Soit XXX un type IDL
 - pointeur sur XXX : XXX * ou XXX_ptr

```
XXX * pxxx = new XXX;
pxxx->foo();
```
 - pointeur intelligent CORBA sur XXX : XXX_var

```
XXX_var xxx = new XXX;
xxx->foo(); // se manipule comme un pointeur !
```
 - passage des arguments via le type XXX_var

```
xxx.in(), xxx.inout(), xxx.out(), xxx._retn()
```

Common Object Services (COS)

- A compléter...

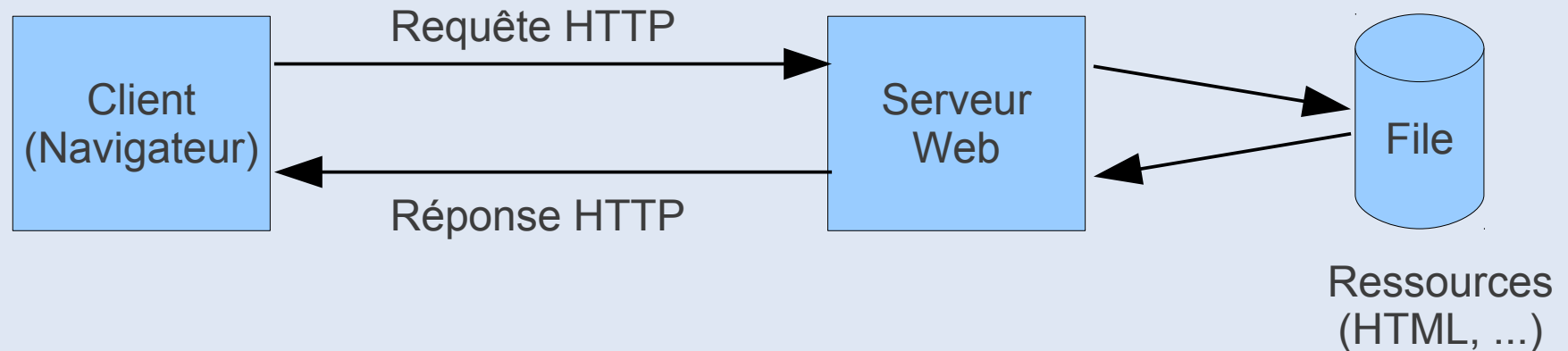
Persitance

- A compléter...
 - ordbd, POA, ServantActivator

Rappel sur le Web

HTTP

- HTTP (HyperText Transfer Protocol)
 - Protocole *stateless* basé sur TCP/IP (port 80)
 - Le navigateur effectue une requête HTTP pour obtenir la ressource URI (Uniform Resource Identifier)
 - Le serveur traite la requête puis retourne une réponse HTTP, typiquement une page HTML



HTTP

- Requêtes
 - GET : demander une ressource (la plus courante)
 - POST : ajouter une nouvelle ressource (ex. message de forum)
 - HEAD : demander uniquement l'en-tête HTTP
 - TRACE : echo de la requête
 - CONNECT, PUT, DELETE, ...
- Historique
 - Version 0.9 : requête GET, réponse HTML
 - Version 1.0 : gestion de cache, description du type MIME des ressources (content-type), ...
 - Version 1.1 : connexion persistante (keep-alive), négociation de contenu (accept-*), ...

Un peu de HTML

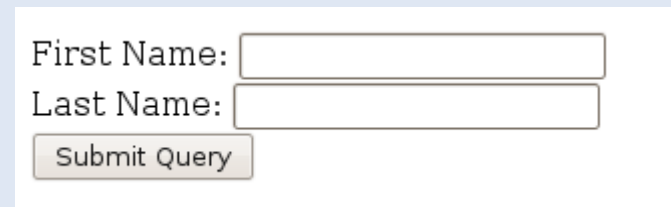
- Structure classique

```
<html>
  <head>
    <title>Hello World!</title>
    ...
  </head>
  <body>
    <center>
      <h1>Hello World!</h1>
    </center>
    <h2> Subtitle </h2>
    <p> paragraph </p>
    ...
  </body>
</html>
```

- Formulaire HTML

- Passage de paramètres (POST)

```
...
<form action="/target" method=POST>
  First Name:
  <input type=text size=20 name=firstname>
  <br>
  Last Name:
  <input type=text size=20 name=lastname>
  <br>
  <input type=submit>
</form>
...
```



First Name:

Last Name:

Les formulaires HTML

- A compléter....
 - <http://www.commentcamarche.net/contents/html/htmlform.php>
3

Exemple HTTP

- Requête

GET /HelloWorld.html HTTP/1.1

← commande GET

Host: localhost:8080

User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.0.15)
Gecko/2009102815 Ubuntu/9.04 (jaunty) Firefox/3.0.15

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

Accept-Language: en-us,en;q=0.5

Accept-Encoding: gzip,deflate

Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7

Keep-Alive: 300

Connection: keep-alive

If-Modified-Since: Thu, 19 Nov 2009 14:06:01 GMT

If-None-Match: W/"153-1258639561000"

Cache-Control: max-age=0

header

Exemple HTTP

- Réponse

HTTP/1.1 200 OK

Server: Apache-Coyote/1.1

Accept-Ranges: bytes

ETag: W/"153-1258639561000"

Last-Modified: Thu, 19 Nov 2009 14:06:01 GMT

Content-Type: text/html

← *type MIME de la ressource*

Content-Length: 153

Date: Tue, 24 Nov 2009 15:48:32 GMT

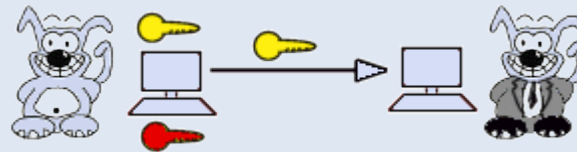
Connection: close

```
<html>
  <head>
    <title>Hello World!</title>
  </head>
  <body>
    <center>
      <h1>Hello World!</h1>
    </center>
  </body>
</html>
```

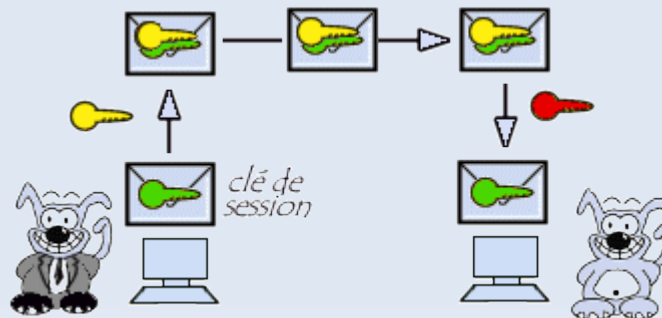
corps de la réponse

Rappel Cryptographie

- Cryptographie à clé publique (asymétrique)
 - Confidentialité : chiffrement par Bob du message avec la clé publique d'Alice et déchiffrement par Alice avec sa clé privée
 - Signature : chiffrement par Alice d'un condensat du message avec sa clé privée et déchiffrement par Bob avec la clé publique d'Alice, qui peut ainsi vérifier le condensat



- Etablissement d'une clé de session symétrique...

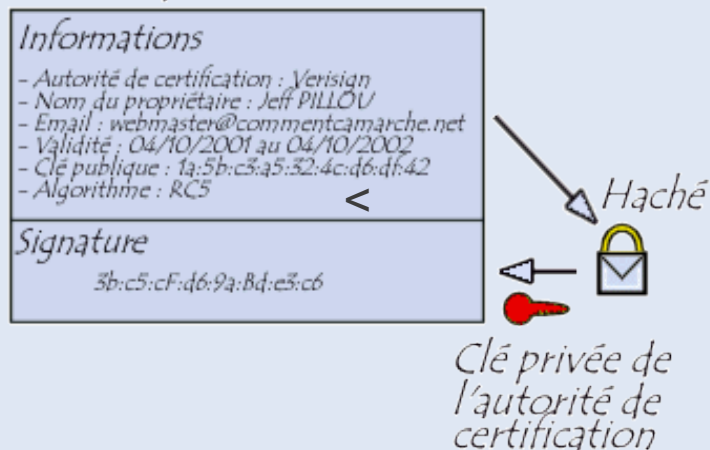


HTTPS

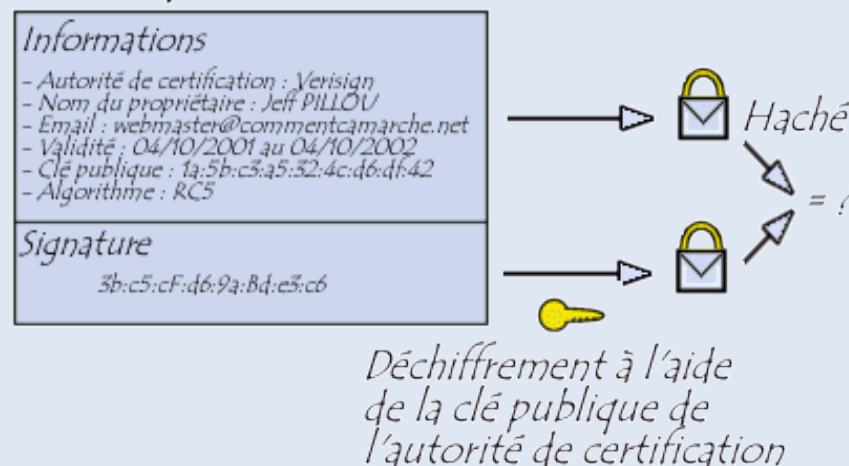
- HTTPS (HTTP Secure)

- Utilisation transparente du protocole HTTP au-dessus de TLS/SSL (port 443 au lieu de 80)
- Authentification du serveur web via son certificat (signé du CA)
- Confidentialité et intégrité des données envoyées au serveur
- Authentification du client facultative

Certificat

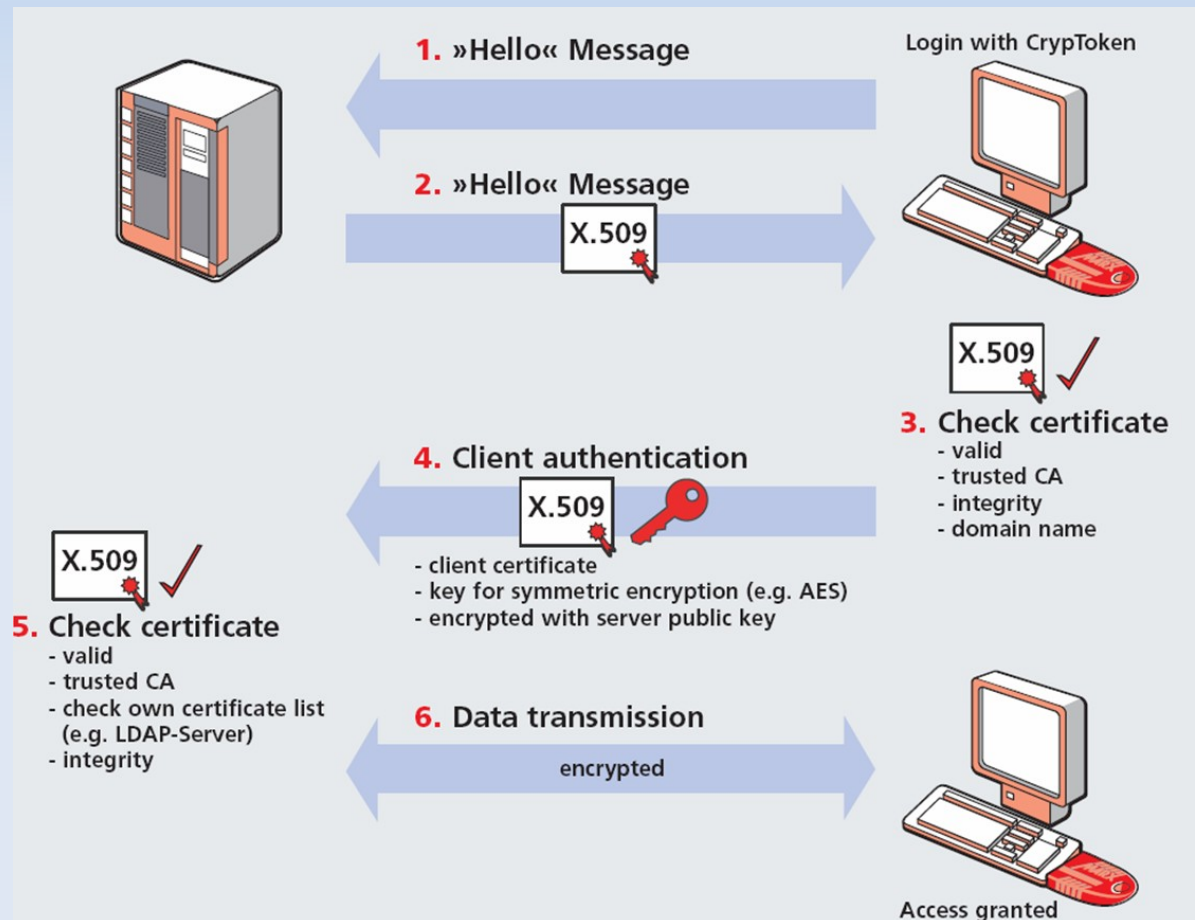


Certificat



HTTPS

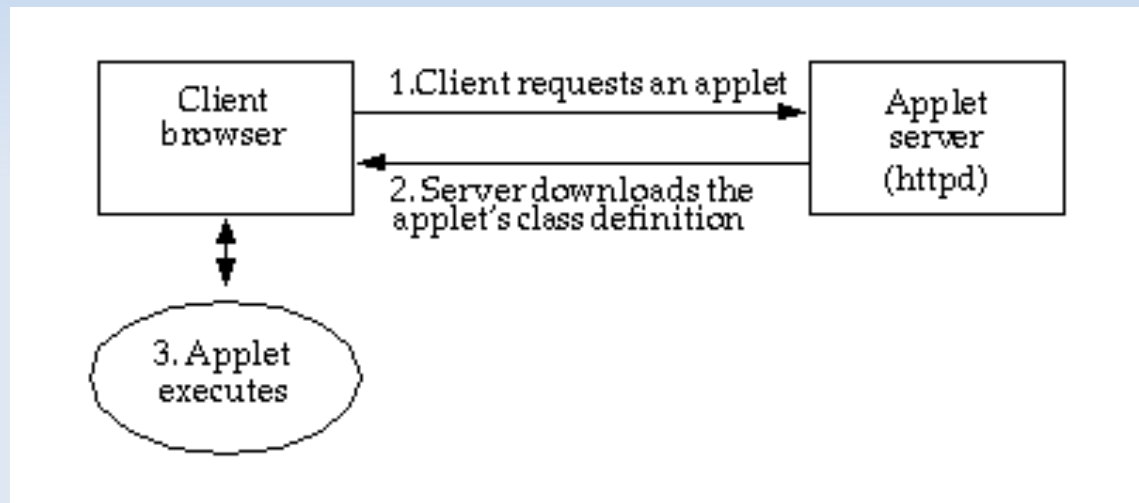
- Authentication du serveur et du client avec SSL/TLS



Source : wikipedia

Applet

- Applet : une application Java téléchargée et exécutée côté client web



Source : Sun

- Exemple de code HTML

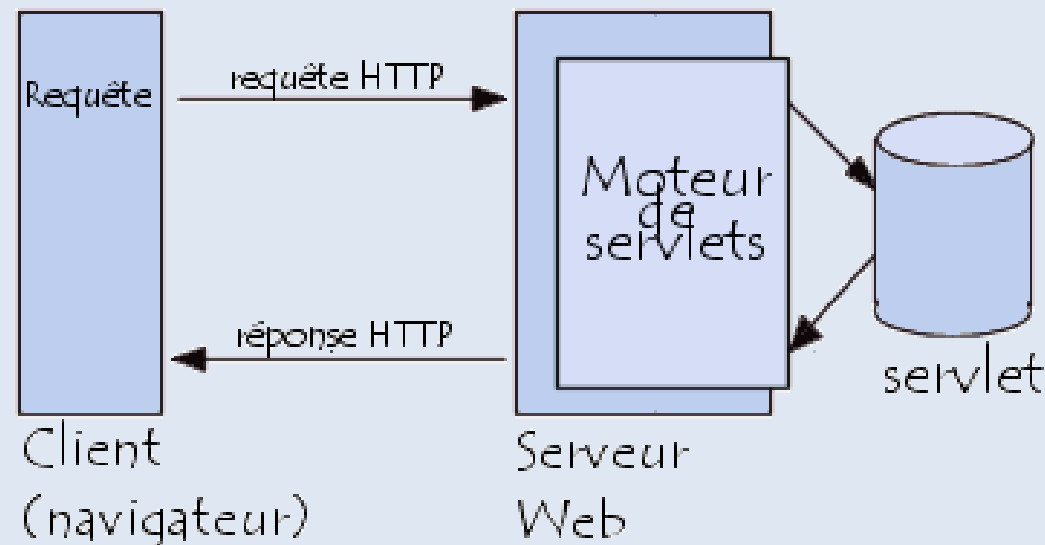
```
<applet height=100 width=100 codebase="myclasses/" code="My.class">
  <param name="ticker">
</applet>
```

Exemple Applet

- (...)

Servlet

- Technologie Java permettant de générer des pages web dynamiquement côté serveur web
 - Traitement des requêtes HTTP (GET et/ou POST)
 - Exécution côté serveur web dans un moteur de servlet (un thread par requête)
 - Comparable à CGI, PHP, ASP, ...



Servlet

- Servlet : classe Java dérivant de HttpServlet traitant une requête et générant une réponse
 - Implémentant essentiellement les méthodes suivantes :
`void init(ServletConfig config)`
`void destroy()`
`void doGet(HttpServletRequest request, HttpServletResponse response)`
`void doPost(HttpServletRequest request, HttpServletResponse response)`
 - Génération de la réponse HTML
`PrintWriter out = response.getWriter();`
`out.println("CODE HTML");`
 - Récupération des paramètres transmis à la servlet sous la forme d'une liste de paire "key / value"
`String value = request.getParameter(String key)`

Servlet

- Un petit exemple en Java...

```
public class HelloWorld extends HttpServlet {
```

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
```

```
throws IOException, ServletException
```

```
{
```

```
    response.setContentType("text/html");
```

```
    PrintWriter out = response.getWriter();
```

```
    out.println("<html>");
```

```
    out.println("<head>");
```

```
    out.println("<title>Hello World!</title>");
```

```
    out.println("</head>");
```

```
    out.println("<body>");
```

```
    out.println("<h1>Hello World!</h1>");
```

```
    out.println("</body>");
```

```
    out.println("</html>");
```

```
    out.close();
```

```
}
```

```
}
```

type MIME de la ressource

contenu de la réponse HTML

JSP

- JSP (Java Servlet Page)
 - Page HTML incluant des scriptlets Java
 - La JSP est dynamiquement compilé en Servlet Java
- Syntaxe
 - Directives (page, include, taglib) : `<%@ ... %>`
 - Déclarations (attributs et méthodes) : `<%! ... %>`
 - Scriptlet (bloc d'instructions) : `<% ... %>`
 - Expressions : `<%= ... %>`
 - Commentaires : `<%-- ... --%>`
 - Expression Language (EL) : `${ expr }`
- Variables globales accessibles depuis la JSP
 - request, response, out, ...

Example JSP

```
<%-- My first Hello World JSP --%>
<%@ page language="java" %>
<%@ page import="java.util.*" %>

<HTML>
<HEAD>
<TITLE>Hello World from JSP</TITLE>
<%!
String message = "Hello World";
%>
</HEAD>

<BODY>
<H1>Hello World from JSP</H1>

<H3> JSP Scripting </H3>
Expression: <%= message %> <br>
Date: <%= new Date() %> <br>
```

```
Scriptlet:<br>
<%
for(int i = 0; i < 10 ; i++) {
    out.print("<font size=" + i + "> hello </font>");
}
%>

<H3> JSP Expression Language (EL) </H3>
12 + 4 = ${12 + 4} <br>
firstname = ${param.firstname} <br>
lastname = ${param.lastname} <br>

</BODY>
</HTML>
```

Example JSP

<http://localhost:8080/test/HelloWorld.jsp?firstname=toto&lastname=tutu>

Hello World from JSP

JSP Scripting

Expression: Hello World

Date: Wed Nov 25 11:56:58 CET 2009

Scriptlet:

`hello` `hello` `hello` `hello` `hello` `hello` `hello` `hello` `hello` `hello`

JSP Expression Language (EL)

`12 + 4 = 16`

`firstname = toto`

`lastname = tutu`

JSP : Compléments

- Cycle de vie : JspInit() et JspDestroy()

```
<HTML>
  <HEAD>
    <TITLE>Using jspInit and jspDestroy</TITLE>
  </HEAD>

  <BODY>
    <H1>Using jspInit and jspDestroy</H1>
    <%!
      int number;

      public void jspInit()
      {
        number = 5;
      }

      public void jspDestroy()
      {
        number = 0;
      }
    %>

    <%
      out.println("The number is " + number + "<BR>");
    %>
  </BODY>
</HTML>
```

Web Services

Deux écoles de Web Service

- Les web services de type WS-*
 - cf. ce cours...
- Les web services de type REST
 - Requête HTTP GET d'une URI → Réponse XML ou JSON
 - à compléter
 - Exemple : API Graph Explorer de Facebook
 - <https://developers.facebook.com/tools/explorer>

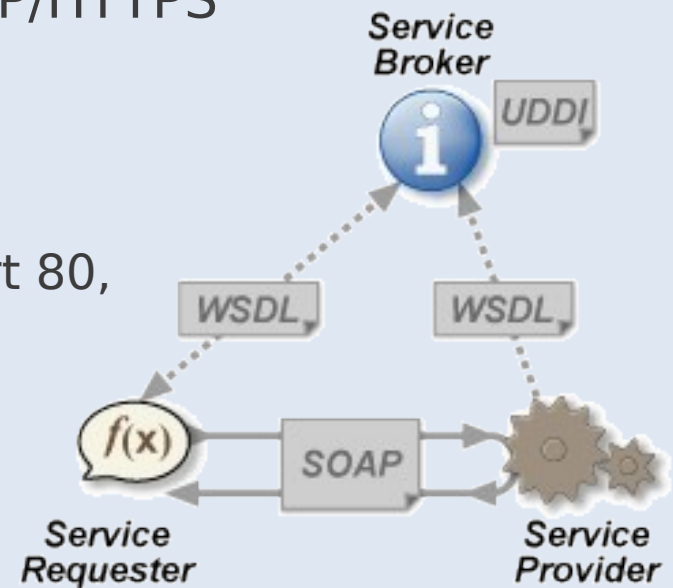
Introduction au Web Services

- Définition
 - Une infrastructure *indépendante des langages et des systèmes* permettant des interactions *faiblement couplées* et *interopérables* entre des applications distribuées sur Internet.
 - séparation de la spécification et de l'implémentation
 - faiblement couplé, car basé sur l'échange de messages
 - interopérable, car basé sur des standards
- Les Web Services en 3 acronymes
 - SOAP (Simple Object Access Protocol)
 - WSDL (Web Services Description Language)
 - UDDI (Universal Description, Discovery and Integration)

“A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.” [W3C]

SOAP, WSDL et UDDI

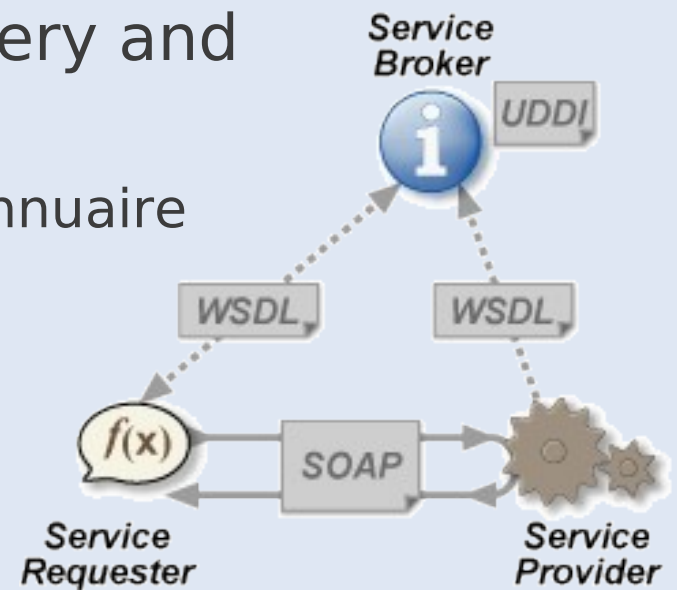
- SOAP (Simple Object Access Protocol)
 - protocole d'échange de messages inter-applications, indépendant de toute plate-forme
 - message ASCII en langage XML
 - transport basé essentiellement sur HTTP/HTTPS
 - paradigme RPC mais pas seulement
 - support des firewalls
 - SOAP utilise le protocole HTTP sur le port 80, généralement ouvert...



Source : wikipedia

SOAP, WSDL et UDDI

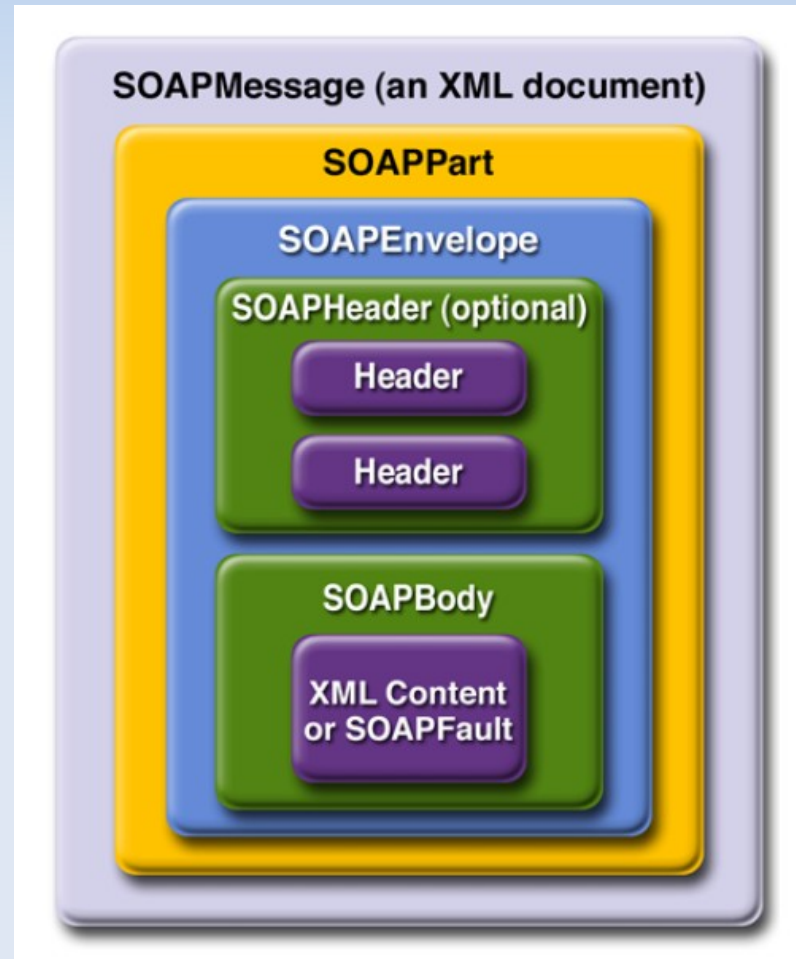
- WSDL (Web Service Description Language)
 - format de description des Web Services en XML
 - description des méthodes et des points d'accès (URL, port, protocole)
- UDDI (Universal Description, Discovery and Integration)
 - un Web Services normalisé servant d'annuaire distribué des Web Services
 - publication / recherche (exploration)



Source : wikipedia

Structure du Message SOAP

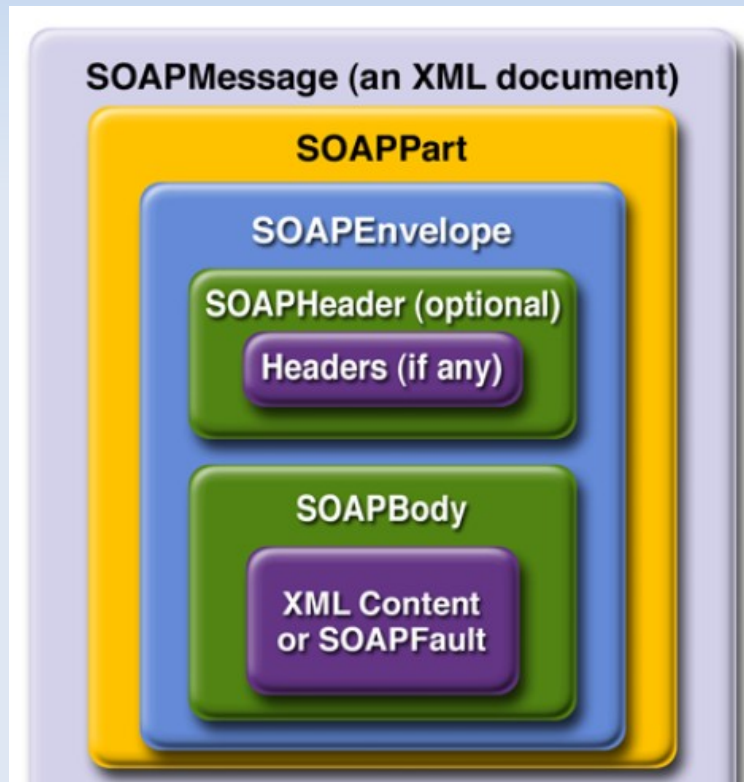
- Message SOAP sans attachement



Source : Sun

Structure du Message SOAP

- Message SOAP avec deux objets attachés



../..



Source : Sun

Web Service Toolkit

- Pas de modèle de programmation spécifique, à la différence de Java RMI ou CORBA !
 - On spécifie la structure des messages, mais pas la manière dont ils sont produits ou consommés...
- En conséquence, beaucoup d'outils divers et variés
 - Minimalistes : NuSOAP (PHP), **Axis** (Java), ...
 - Sophistiqués : BEA WebLogic, IBM WebSphere, .NET, ...

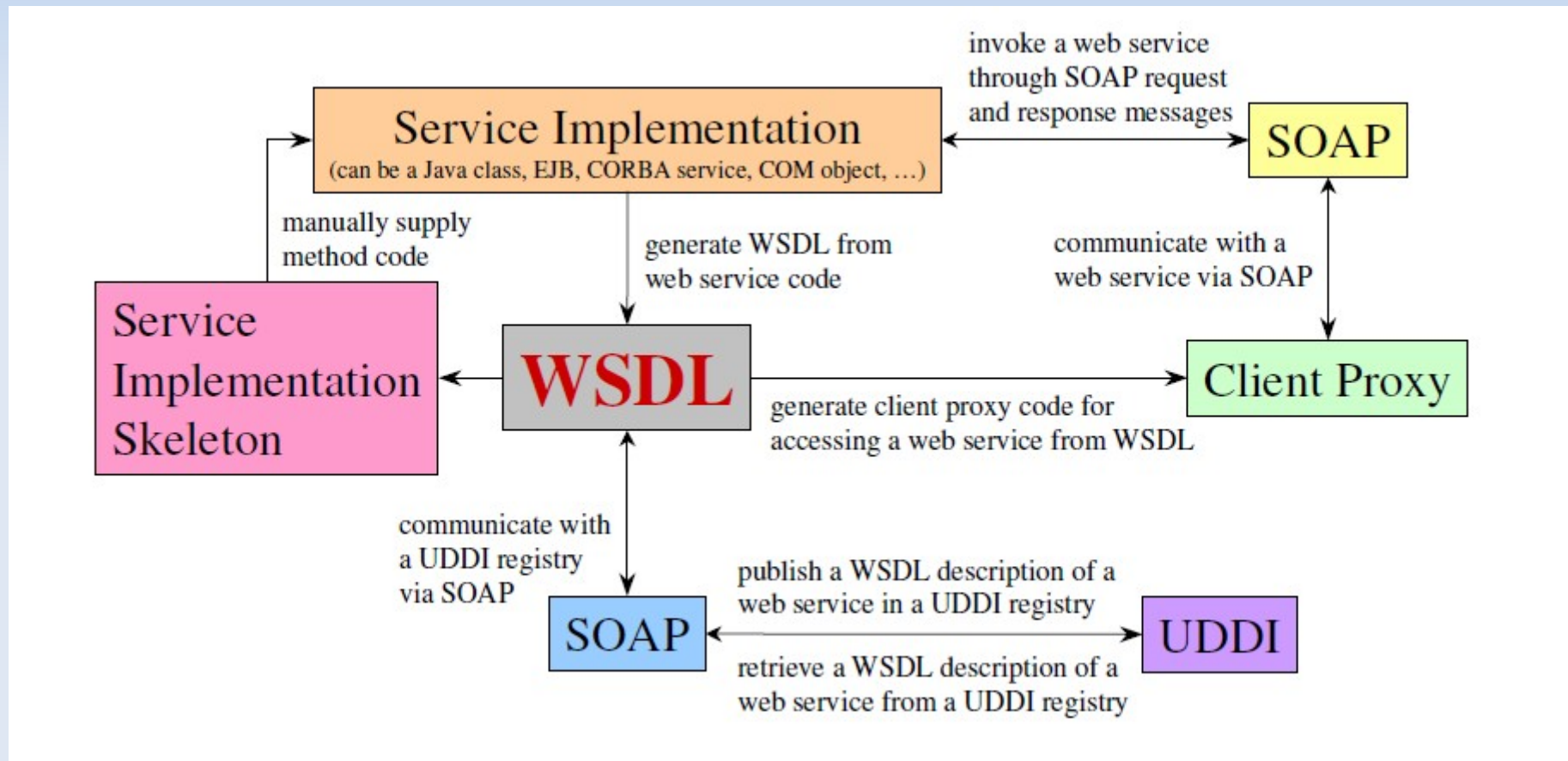
Axis

- Axis (Apache eXtensible Interaction Service)
 - Un toolkit open source de Web Service basé sur Apache Tomcat
 - Deux styles de déploiement des Web Services
 - Instant Deployment (JWS)
 - Custom Deployment (WSDD)
 - Programmation d'application Java avec JAX-RPC
 - Client statique ou dynamique
 - Outils pour WSDL : WSDL2Java et Java2WSDL
 - Support de EJB : Session Bean accessible comme WS



Axis Toolkit

- Vue d'ensemble du toolkit



Instant Deployment (JWS)

- Principe
 - Une simple classe Java renommée avec l'extension jws
 - Placer le fichier **jws** dans **<tomcat>/webapps/axis/...**
 - Traduction automatique en Service Web
 - Déploiement instantané dans **http://<server>:<port>/axis/...**
 - Scope Request : un nouvel objet instancié à chaque requête
- Exemple HelloWorld.jws

```
public class HelloWorld
{
    public String test(String data)
    {
        return "Hello World! You sent the string '" + data + "'.";
    }
}
```

Messages SOAP

- Requête à partir d'un navigateur web

<http://localhost:8080/axis/HelloWorld.jws?method=test&arg=toto>

- Message SOAP de la requête

```
-<soapenv:Envelope>  
  -<soapenv:Body>  
    -<ns1:test soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">  
      <ns1:arg0 xsi:type="soapenc:string">toto</ns1:arg0>  
    </ns1:test>  
  </soapenv:Body>  
</soapenv:Envelope>
```

- Message SOAP de la réponse

```
-<soapenv:Envelope>  
  -<soapenv:Body>  
    -<testResponse soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">  
      <testReturn xsi:type="xsd:string">Hello World! You sent the string 'toto'.</testReturn>  
    </testResponse>  
  </soapenv:Body>  
</soapenv:Envelope>
```


Type Mapping

- Mapping standard des types WSDL/XSD/SOAP en Java
 - Sérialisation / désérialisation des objets Java en XML dans les messages SOAP

xsd:base64Binary	byte[]
xsd:boolean	boolean
xsd:byte	byte
xsd:dateTime	java.util.Calendar
xsd:decimal	java.math.BigDecimal
xsd:double	double
xsd:float	float
xsd:hexBinary	byte[]
xsd:int	int
xsd:integer	java.math.BigInteger
xsd:long	long
xsd:QName	javax.xml.namespace.QName
xsd:short	short
xsd:string	java.lang.String

Source : <http://ws.apache.org/axis/java/user-guide.html>

WSDL

- Structure générale d'un document WSDL
 - **<wsdl:definitions>** : la racine du document XML décrivant un service web
 - **<wsdl:type>** : définition des types de données utilisées
 - **<wsdl:message>** : description du type des messages (type des paramètres **<wsdl:part>**)
 - **<wsdl:portType>** : un type de port, décrivant l'ensemble des opérations du service (**<wsdl:operation>**)
 - **<wsdl:binding>** : une liaison décrivant le protocole de transport, et les types des messages associés aux opérations du service...
 - **<wsdl:service>** : une collection de ports, chaque port (**<wsdl:port>**) associant une liaison et une adresse réseau explicite (endpoint)

WSDL

<http://localhost:8080/axis/HelloWorld.jws?wsdl>

```
-<wsdl:definitions targetNamespace="http://localhost:8080/axis/HelloWorld.jws">
  -<wsdl:message name="testRequest">
    <wsdl:part name="data" type="xsd:string"/>
  </wsdl:message>
  -<wsdl:message name="testResponse">
    <wsdl:part name="testReturn" type="xsd:string"/>
  </wsdl:message>
  -<wsdl:portType name="HelloWorld">
    -<wsdl:operation name="test" parameterOrder="data">
      <wsdl:input message="impl:testRequest" name="testRequest"/>
      <wsdl:output message="impl:testResponse" name="testResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  -<wsdl:binding name="HelloWorldSoapBinding" type="impl:HelloWorld">
    <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    -<wsdl:operation name="test">
      <wsdlsoap:operation soapAction=""/>
      -<wsdl:input name="testRequest">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="http://DefaultNamespace" use="encoded"/>
      </wsdl:input>
      -<wsdl:output name="testResponse">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="http://localhost:8080/axis/HelloWorld.jws" use="encoded"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  -<wsdl:service name="HelloWorldService">
    -<wsdl:port binding="impl:HelloWorldSoapBinding" name="HelloWorld">
      <wsdlsoap:address location="http://localhost:8080/axis/HelloWorld.jws"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

endpoint

Messages
(request, response)

Port Type
(operations)

Binding

Service

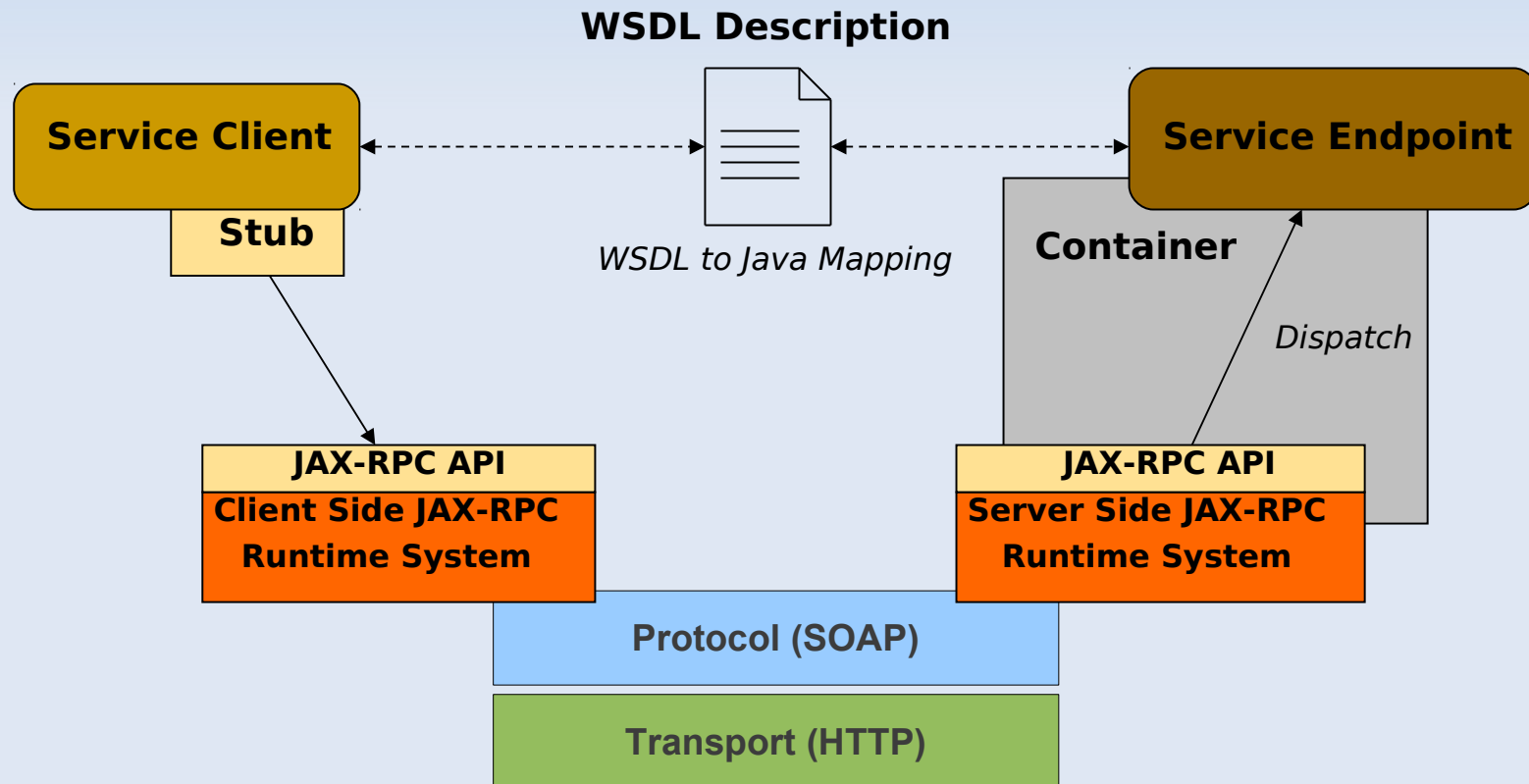
JAX-RPC

- Java API for XML-based RPC (JAX-RPC)
 - Modèle de programmation Sun Java des Web Services (en mode RPC), comparable à RMI...
 - Mapping des types Java et WSDL
 - Client statique ou dynamique
 - Génération des stubs avec WSDL2Java...
 - Utilisation des interfaces *Service* et *Call*
 - Version 2.0 de JAX-RPC...
 - Java API for XML Web Services (JAX-WS)

Client Statique (JAX-RPC)

- Génération des stubs à partir du WSDL...

`java org.apache.axis.wSDL.WSDL2Java HelloWorld.wsd`



Source : Pankaj Kumar

Client Statique (JAX-RPC)

- Utilisation du stub généré...

```
import java.rmi.RemoteException;  
import javax.xml.rpc.ServiceException;  
import localhost.axis.HelloWorld_jws.*;
```

import des classes stub

```
public class HelloWorldStaticClient  
{  
    public static void main(String[ ] args)  
        throws ServiceException, RemoteException  
    {  
        HelloWorldService locator = new HelloWorldServiceLocator();  
        HelloWorld stub = locator.getHelloWorld();  
        String returnValue = stub.test("toto");  
        System.out.println(returnValue);  
    }  
}
```

localisation du endpoint

récupération du stub

interface du service HelloWorld

Client Dynamique (JAX-RPC)

- Généralités
 - Pas d'utilisation de WSDL
 - Utilisation des interfaces *Service* et *Call*
 - *org.apache.axis.client.Service* -> *javax.xml.rpc.Service*
 - *org.apache.axis.client.Call* -> *javax.xml.rpc.Call*
 - Pas de vérification du type des paramètres à la compilation
 - Passage d'un tableau d'Object...

Client Dynamique (JAX-RPC)

```
import org.apache.axis.client.Call;  
import org.apache.axis.client.Service;
```

```
public class HelloWorldClient
```

```
{
```

```
    private static final String ENDPOINT = "http://localhost:8080/axis/HelloWorld.jws";  
    private static final String NAMESPACE = "http://soapinterop.org/";  
    private static final String OPERATION = "test";
```

```
    public static void main(String[ ] args)
```

```
        throws ServiceException, MalformedURLException, RemoteException
```

```
{
```

```
    Service service = new Service();
```

```
    Call call = (Call)service.createCall();
```

```
    call.setTargetEndpointAddress(new URL(ENDPOINT));
```

```
    call.setOperationName(new QName(NAMESPACE, OPERATION));
```

```
    String returnValue = (String)call.invoke(new Object[ ]{"toto"});
```

```
    System.out.println(returnValue);
```

```
}
```

```
}
```

*Les paramètres sont passés comme un tableau d'Object,
et le résultat retourné comme un Object...*

Custom Deployment (WSDD)

- Principe

- Déploiement personnalisé d'un Service Web
 - Scope : Request, Session ou Application
 - Accessible par défaut dans <http://<server>:<port>/axis/services/...>
- Pas besoin des sources
 - Placer les classes dans [<tomcat>/webapps/axis/WEB-INF/classes/...](tomcat/webapps/axis/WEB-INF/classes/...)
 - Placer les jar dans [<tomcat>/webapps/axis/WEB-INF/lib/...](tomcat/webapps/axis/WEB-INF/lib/...)
- Basé sur un fichier WSDD (Web Service Deployment Descriptor)
 - Format XML spécifique à Axis
- Outil **AdminClient** pour gérer le déploiement...
 - List : `java org.apache.axis.client.AdminClient list`
 - Deploy : `java org.apache.axis.client.AdminClient deploy.wsdd`
 - Undeploy : `java org.apache.axis.client.AdminClient undeploy.wsdd`

Custom Deployment (WSDD)

- deploy.wsdd

```
-<deployment>  
  -<service name="HelloWorld" provider="java:RPC" style="rpc" use="encoded">  
    <parameter name="wsdlTargetNamespace" value="urn:HelloWorld"/>  
    <parameter name="wsdlServiceElement" value="HelloWorldService"/>  
    <parameter name="wsdlServicePort" value="HelloWorld"/>  
    <parameter name="className" value="HelloWorld_pkg.HelloWorldSoapBindingSkeleton"/>  
    <parameter name="wsdlPortType" value="HelloWorld"/>  
    <parameter name="typeMappingVersion" value="1.2"/>  
    <parameter name="allowedMethods" value="*/>  
    <parameter name="scope" value="Session"/>  
  </service>  
</deployment>
```

- undeploy.wsdd

```
-<undeployment>  
  <service name="HelloWorld"/>  
</undeployment>
```

WSDL2Java

- Génération des stubs & skeletons à partir du WSDL

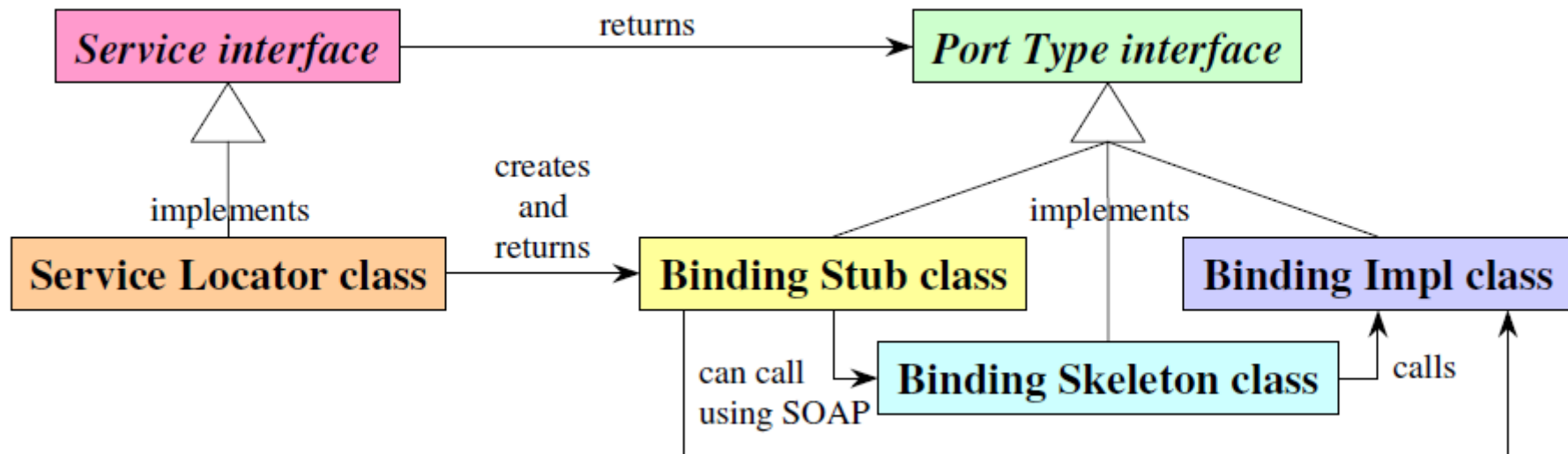
```
java org.apache.axis.wsdl.WSDL2Java -s -S true -d <scope> -o <dir> file.wsdl
```

-s -S true : options pour générer la partie serveur (skeleton)

<scope> = { Request | Session | Application }

-o <dir> : output directory

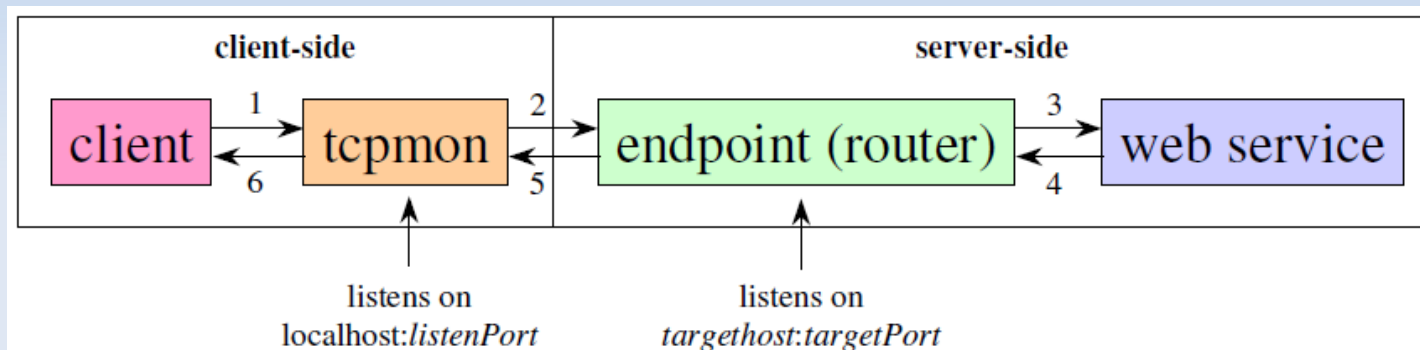
- Relation entre les différents fichiers générés



SOAP Message Monitoring

- Utilisation de l'outil tcpmon

```
java org.apache.axis.utils.tcpmon [listenPort targetHost targetPort]
```



Source : Mark Volkmann

- Exemple

```
java org.apache.axis.utils.tcpmon 1234 localhost 8080
```

```
http://localhost:1234/axis/services/HelloWorld?method=test&arg=toto
```

SOAP Message Monitoring

The screenshot shows the TCPMonitor application window. At the top, there are controls for 'Admin' (Port 8081), 'Stop', 'Listen Port' (8081), 'Host' (services.xmeth), 'Port' (80), and a 'Proxy' checkbox. Below this is a table with columns: State, Time, Request Host, Target Host, and Request.. The table contains two rows of data, both with a 'Done' state and a 'Content-Length: 512' request.

State	Time	Request Host	Target Host	Request..
---	Most Recent	---	---	---
Done	03/29/02 04:02:54 PM	localhost	services.xmethods.net	Content-Length: 512
Done	03/29/02 04:02:56 PM	localhost	services.xmethods.net	Content-Length: 512

Below the table are 'Remove Selected' and 'Remove All' buttons. The main area is split into 'Request' and 'Response' panes. The Request pane shows a POST request to /soap/servlet/rpcrouter with SOAPAction: "" and XML body. The Response pane shows an HTTP 200 OK response with headers and XML body.

Request:

```
POST /soap/servlet/rpcrouter HTTP/1.0
Content-Length: 512
Host: localhost
Content-Type: text/xml; charset=utf-8
SOAPAction: ""

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <nsl:getTemp xmlns:nsl="urn:xmethods-Temperature">
      <arg0 xsi:type="xsd:string">90210</arg0>
    </nsl:getTemp>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Response:

```
HTTP/1.0 200 OK
Date: Fri, 29 Mar 2002 22:02:55 GMT
Status: 200
Content-type: text/xml; charset=utf-8
Servlet-engine: Lutris Enhydra Application
Content-length: 465
Set-cookie: JSESSIONID=nPk90p9b2rh5BwMRDKnS
Server: Enhydra-MultiServer/3.5.2

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <nsl:getTempResponse xmlns:nsl="urn:xmethods-Temperature">
      <return xsi:type="xsd:float">58.0</return>
    </nsl:getTempResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

At the bottom, there are buttons for 'XML Format', 'Save', 'Resend', 'Switch Layout', and 'Close'.

Java EE & EJB 3

Introduction

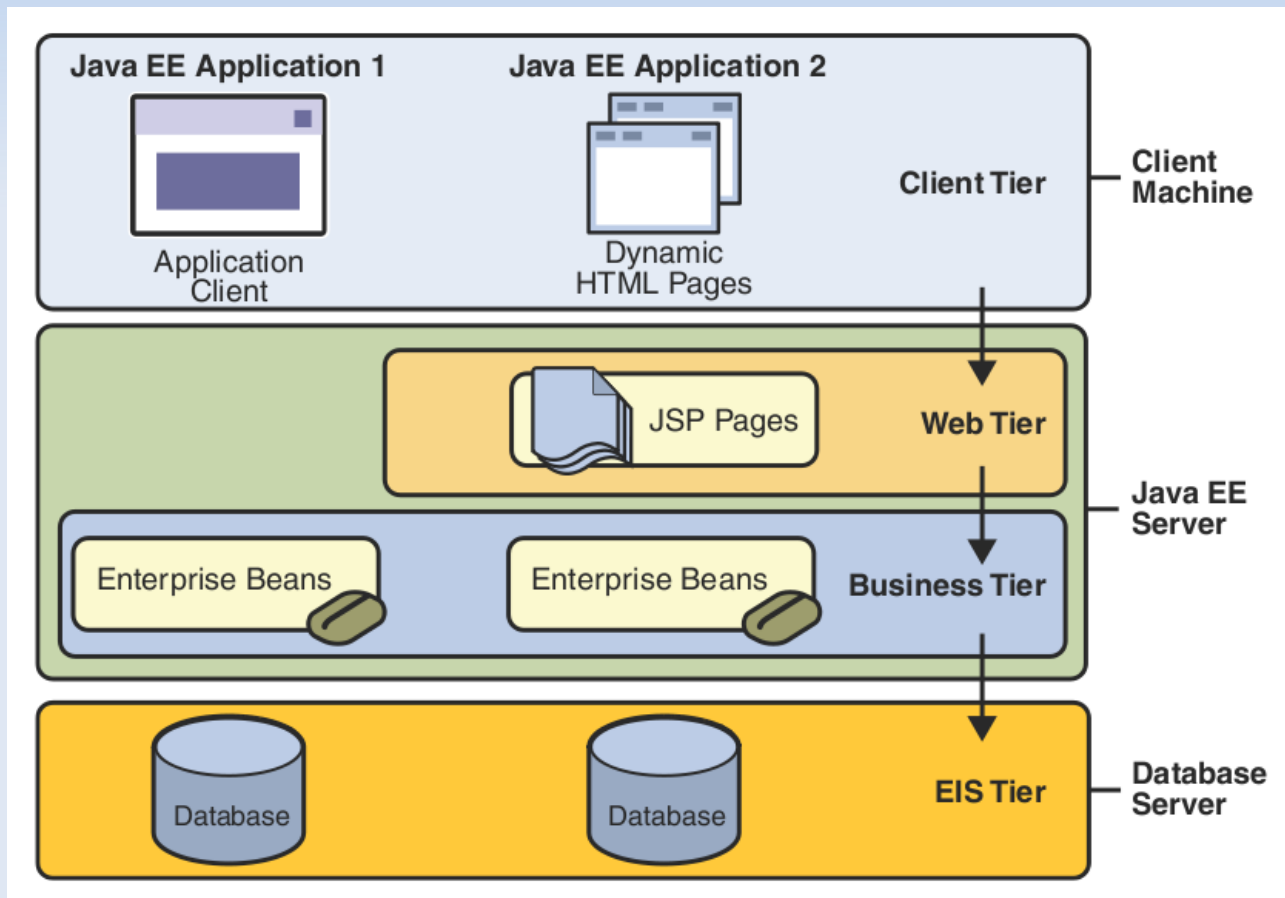
- Java EE (Java Platform Enterprise Edition)
 - Spécifications Sun d'un framework Java destiné au développement d'application d'entreprise répartie
 - Orienté composant
 - Concept clé : EJB (Enterprise Java Bean)
 - Spécification de multiples APIs *javax.**, qui étendent Java SE
 - Enterprise JavaBean (EJB)
 - Java transaction API (JTA)
 - Java Message Service (JMS)
 - Java Persistence API (JPA), implanté par Hibernate
 - Anciennement J2EE

Introduction

- Serveur d'application
 - Une application Java EE s'exécute dans un serveur d'application
 - Exemple : *GlassFish, JOnAS, JBoss, ...*
- Les différentes couches (tiers)
 - Client : standalone client, web client, applet
 - Web : Servlet et JSP
 - Business : Enterprise Java Beans (EJB)
 - EIS : Enterprise Information System (database, ...)

Introduction

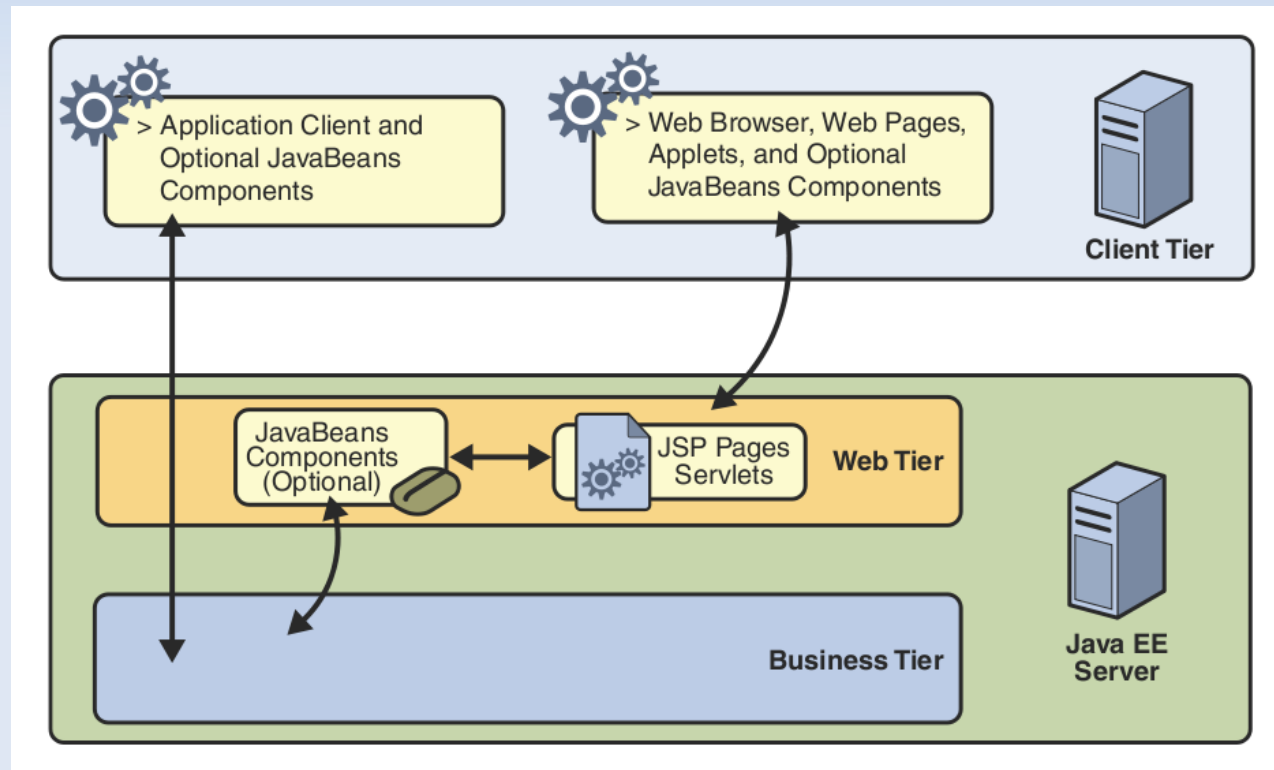
- Architecture multi-tiers de Java EE



Source : Sun

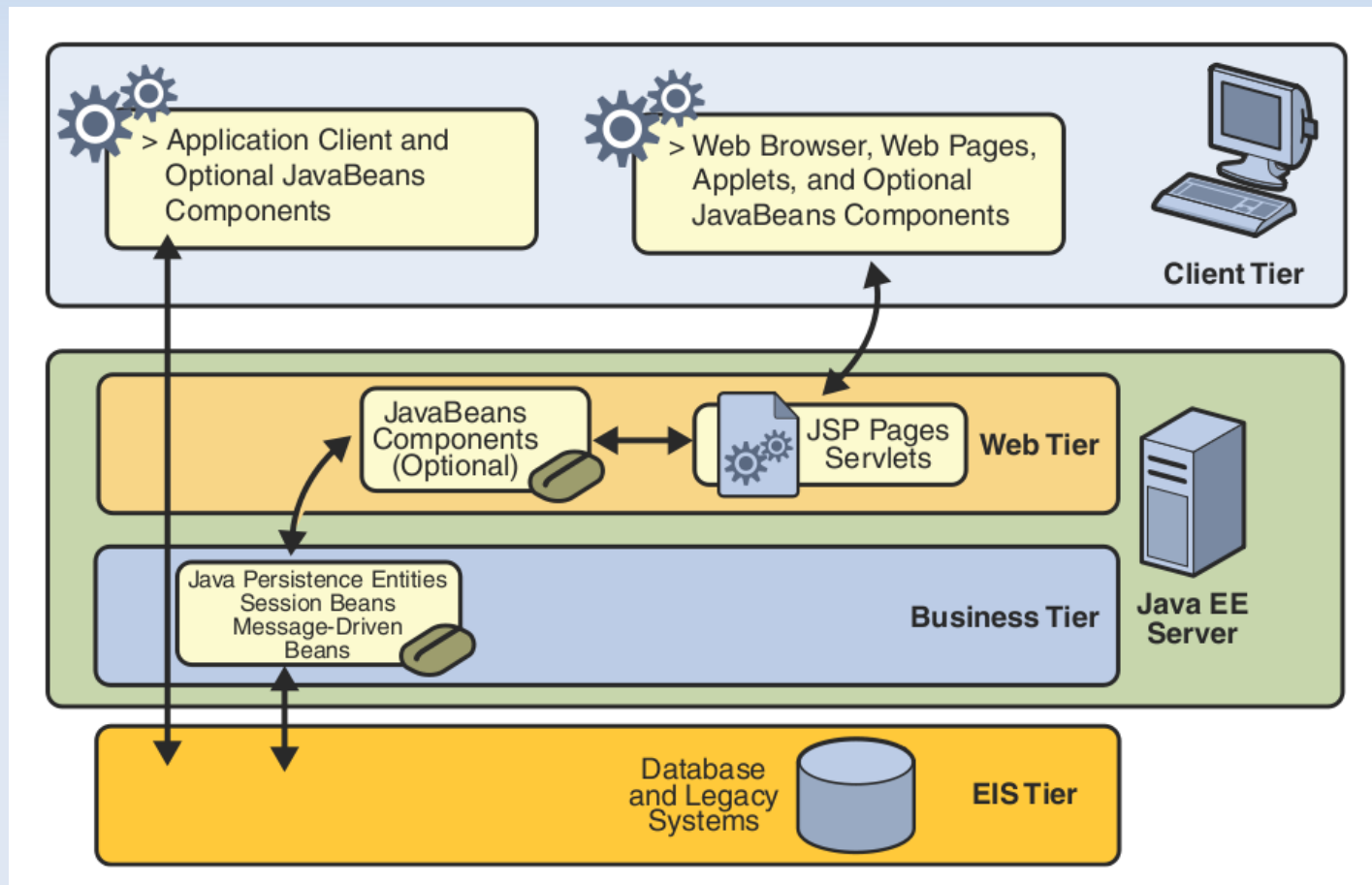
Introduction

- Une application simple de type “Web”
 - Pas de persistance (JSP, Servlet, Session Bean)



Introduction

- Une application plus complexe de type “Enterprise”
 - Avec de la persistance (... , Entity Bean, SGBD)



Enterprise Java Beans

- Les différents EJBs
 - Session Bean : offre un service au client
 - Stateless ou Stateful
 - Entity Bean : représente un objet persistant lié à un mécanisme de stockage (SGBD)
 - Message-Driven Bean : se comporte comme un *listener* du JMS (Java Message Service) et traite les messages de manière asynchrone

Enterprise Java Beans

- Caractéristiques et nouveautés de EJB 3
 - Concept POJO (Plain Old Java Object)
 - Les EJBs sont des objets “simples”
 - Configuration par des annotations (inspiré de XDoclet)
 - Plus de fichiers XML (sauf persistence.xml)
 - Configuration par exception
 - Paramétrage par défaut pour accélérer le développement
 - Injection de dépendance
 - Le conteneur est responsable de la création des objets et résout les dépendances entre les objets

Enterprise Java Beans

- Interfaces métier

- **@Local** : interface accessible localement par les autres composant du serveur d'application Java EE, typiquement d'autres EJBs ou une servlet.
- **@Remote** : interface accessible par un client distant (standalone).
- **No interface-view**. Pas d'interface spécifiée, équivalent à un @Local, mais en interdisant @Remote.

```
@Local  
public class HelloBeanLocal {  
    public String sayHello();  
}
```

```
@Remote  
public class HelloBeanRemote {  
    public String sayHello();  
}
```

Enterprise Java Beans

- Stateless Session Bean

- L'état du bean n'est pas maintenu entre les différentes invocations des clients...

`@Stateless`

```
public class HelloBean  
    implements HelloBeanLocal, HelloBeanRemote {  
    public String sayHello() { return ("Hello World !"); }  
}
```

← Accessible localement et à distance...

Enterprise Java Beans

- Stateful Session Bean

- Conversation entre un client particulier et un bean
- L'état du bean est conservé tout le long de la conversation et disparaît lorsque le bean est détruit

@Stateful

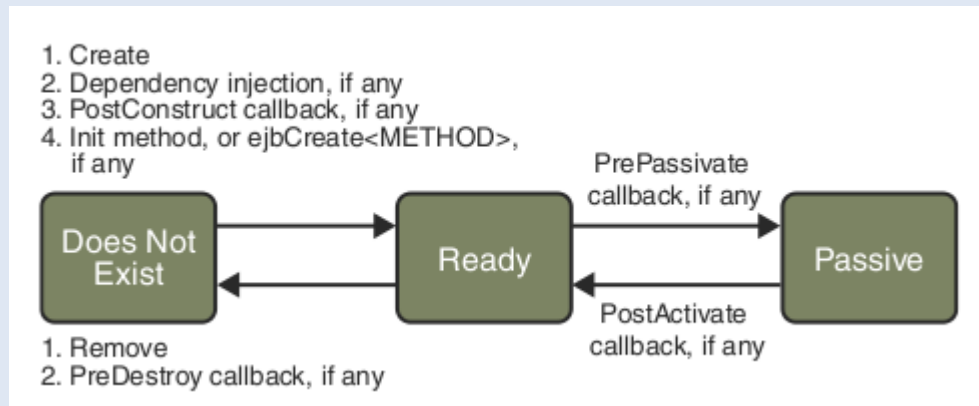
```
public class HelloLoginBean {  
  
    private String user;  
  
    public void login(String user) {  
        this.user = user;  
    }  
  
    public String sayHello() {  
        if (user == null)  
            return ("Hello world !");  
        return ("Hello " + user + " !");  
    }  
}
```

← No interface-view (accessible uniquement en local)

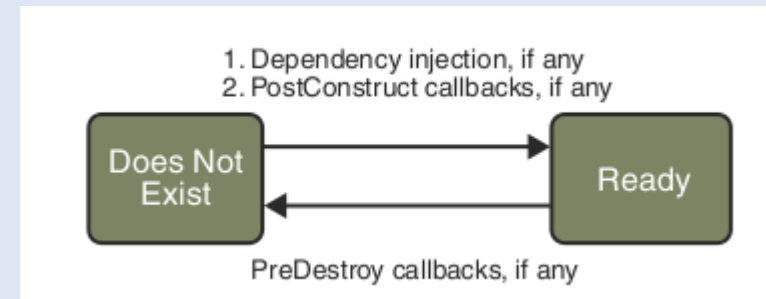
Enterprise Java Beans

- Cycle de vie des Session Beans

- Instantiation implicite via le conteneur (nécessite un constructeur public sans args)
- Appel de la méthode annotée par @Remove pour la destruction



Stateful



Stateless

- Intercepteurs

- @PostConstruct, @PreDestroy
- @PrePassivate, @PostActivate (stateful seulement)

Enterprise Java Beans

- Entity Bean

- Objet dont l'état est persistant entre deux sessions
- Liée à une base de données via un mapping objet-relationnel
- Configuration du mapping directement par des annotations JPA

@Entity

@Table(name="NewsEntity")

← Nom de la table (par défaut, le nom de la classe)

```
public class NewsEntity implements Serializable {  
    private static final long serialVersionUID = 1L;
```

@Id

```
private Long id;  
private String title;  
private String body;
```

} Attributs persistants (par défaut)

```
public String getBody() { return body; }  
public void setBody(String body) { this.body = body; }  
public String getTitle() { return title; }  
public void setTitle(String title) { this.title = title; }  
public Long getId() { return id; }  
public void setId(Long id) { this.id = id; }
```

} Getter & setter

```
}
```

Enterprise Java Beans

- Java Persistence API (JPA)

- La classe `@Entity` doit posséder un constructeur sans argument
- `@Id` permet de définir le champs utilisé comme clé primaire
- `@Basic` permet de définir un champs persistant (optionnel)
- `@Transient` pour préciser qu'un champs n'est pas persistant
- `@Column` pour configurer les propriétés persistantes d'une colonne dans la table (longueur, unicité, ...)

```
@Column(length=30, unique=true)
```

```
String login;
```

- Les champs relationnels
 - `@OneToOne`, `@OneToMany`, `@ManyToOne`, `@ManyToMany`
- (...)

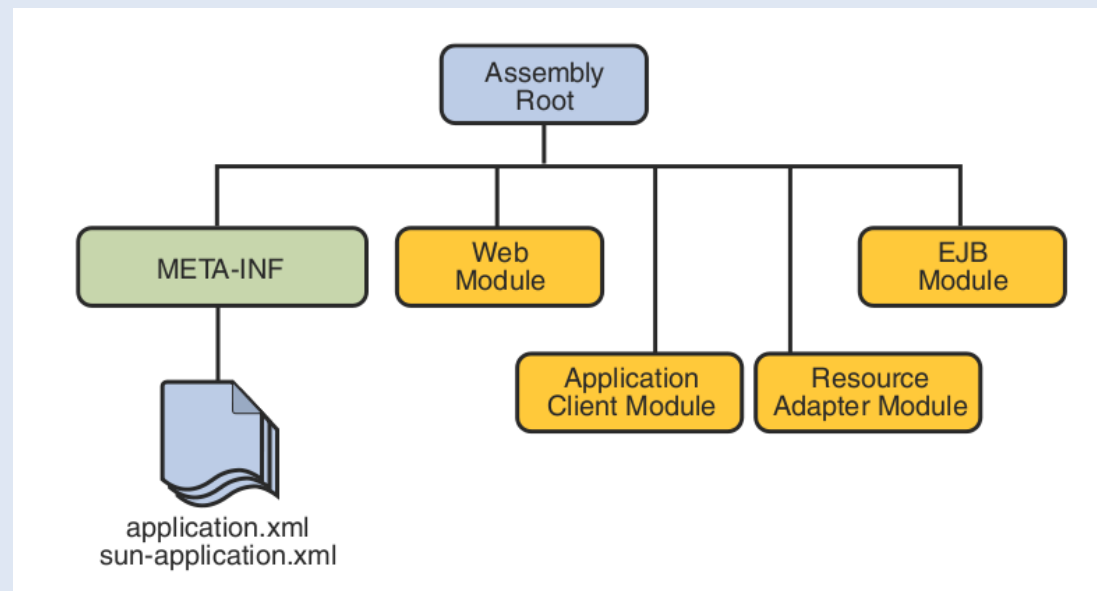
Enterprise Java Beans

- Cycle de vie des Entity Bean en 4 états
 - New : simple instance de l'entity (pas encore de contexte persistant)
 - Managed : entity associé à un contexte persistant
 - [EntityManager.persist\(\)](#), [EntityManager.merge\(\)](#), ...
 - Detached : instance de l'entity qui n'est plus associé au contexte persistant d'où elle provient
 - Cas où l'entity est envoyé à un autre tiers (servlet)
 - Removed : instance de l'entity destinée à être supprimée de la base de données
 - [EntityManager.remove\(\)](#)
- Callbacks
 - @PrePersist, @PostPersist, @PreRemove, @PostRemove,...

Introduction

- Packaging

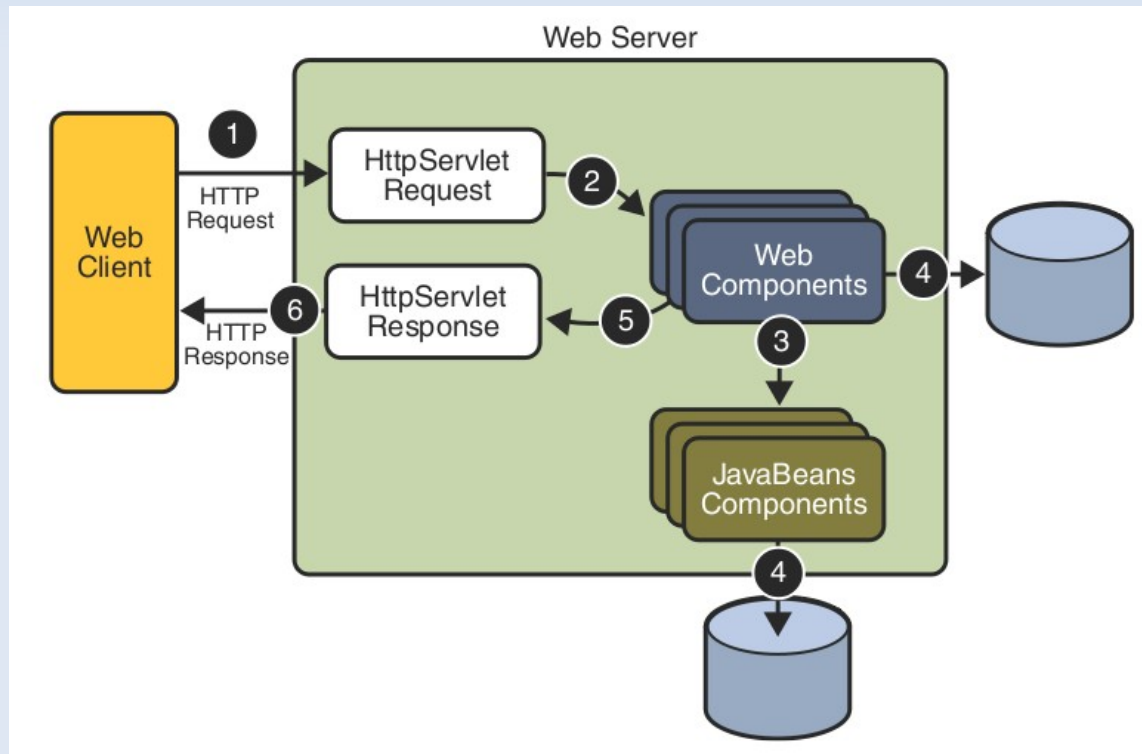
- EAR : archive globale d'une application Java EE, déployable dans un serveur d'application...
- WAR : archive du module Web
- JAR : archive classique, utilisée pour le module EJB, l'application cliente



Organisation d'un package Java EE

Exemple Converter

- Web Application
 - Stateless Session Bean : ConverterBean.java
 - Servlet : ConverterServlet.java



Exemple Converter

- ConverterBean.java

@Stateless



Déclaration d'un composant EJB Session Stateless

```
public class ConverterBean {
```

```
    private BigDecimal euroRate = new BigDecimal("0.0078");  
    private BigDecimal yenRate = new BigDecimal("96.0650");
```

```
    public BigDecimal dollarToYen(BigDecimal dollars) {  
        BigDecimal result = dollars.multiply(yenRate);  
        return result.setScale(2, BigDecimal.ROUND_UP);  
    }
```

```
    public BigDecimal yenToEuro(BigDecimal yen) {  
        BigDecimal result = yen.multiply(euroRate);  
        return result.setScale(2, BigDecimal.ROUND_UP);  
    }
```

```
}
```

La classe ne contient que les business methods et rien de plus !

Exemple Converter

- ConverterServlet.java

```
@WebServlet  
public class ConverterServlet extends HttpServlet {
```

← Déclaration d'un composant Web Servlet

```
@EJB  
ConverterBean converter;
```

← Injection de dépendance, instantiation implicite !

```
@Override  
protected void doGet(HttpServletRequest request, HttpServletResponse response)  
    throws ServletException, IOException {
```

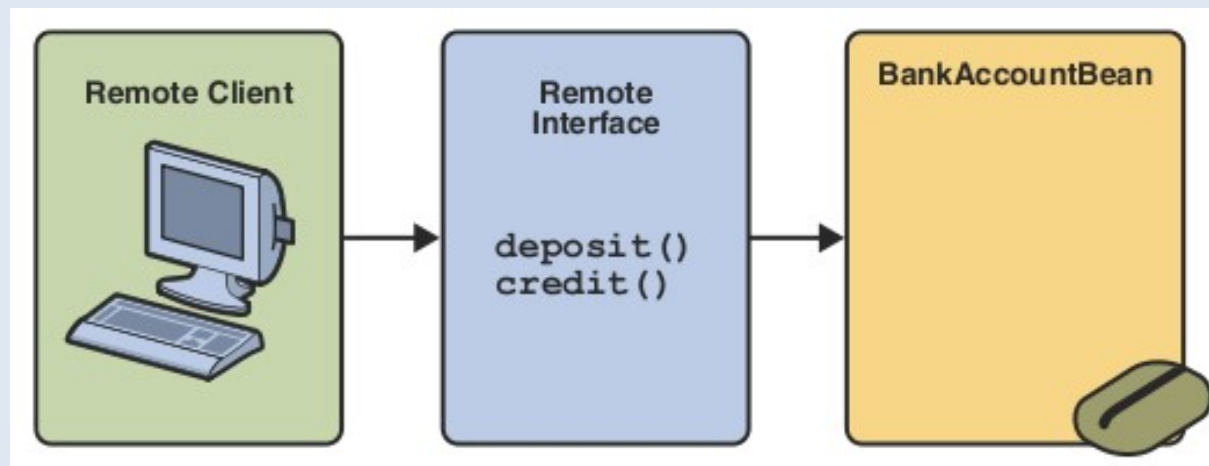
← Redéfinition d'une méthode, vérification à la compilation !

```
    response.setContentType("text/html;charset=UTF-8");  
    PrintWriter out = response.getWriter();  
    out.println("<html>");  
    /* ... */  
    String amount = request.getParameter("amount");  
    BigDecimal d = new BigDecimal(amount);  
    BigDecimal yenAmount = converter.dollarToYen(d);  
    /* ... */  
    out.println("</html>");  
    out.close();  
}  
}
```

← Utilisation du bean...

Exemple Cart

- Mise en oeuvre d'un client distant
 - Interface Remote : Cart.java
 - Stateful Session Bean : CartBean.java
 - Standalone Client : CartClient.java



Exemple Cart

- CartRemote.java

@Remote

```
public interface CartRemote {
```

```
    public void initialize(String person);  
    public void addBook(String title);  
    public void removeBook(String title);  
    public List<String> getContents();  
    public void remove();
```

```
}
```

← Déclaration d'une interface de composant EJB accessible par un client distant

} business methods...

Exemple Cart

- CartBean.java

`@Stateful` ← Déclaration d'un composant EJB Session Stateful
`public class CartBean implements CartRemote {`

```
List<String> contents;  
String customerName;
```

```
public void initialize(String person) {  
    customerName = person;  
    contents = new ArrayList<String>();  
}
```

```
public void addBook(String title) { contents.add(title); }  
public void removeBook(String title) { contents.remove(title); }  
public List<String> getContents() { return contents; }
```

`@Remove` ← Méthode permettant de détruire le bean...

```
public void remove() {  
    contents = null;    /* libération de la mémoire... */  
}
```

```
}
```

Exemple Cart

- CartClient.java

```
public class CartClient {  
  
    @EJB  
    private static CartRemote cart;  
  
    public static void main(String[] args) {  
        test();  
    }  
  
    public static void test() {  
        cart.initialize("Moi");  
        cart.addBook("Oui Oui en Chine");  
        cart.addBook("Discours de la Méthode");  
        List<String> bookList = cart.getContents();  
        Iterator<String> iterator = bookList.iterator();  
        while (iterator.hasNext()) {  
            String title = iterator.next();  
            System.out.println("Retrieving book title from cart: " + title);  
        }  
        cart.remove();  
    }  
}
```

Injection en static, car un session bean par client !

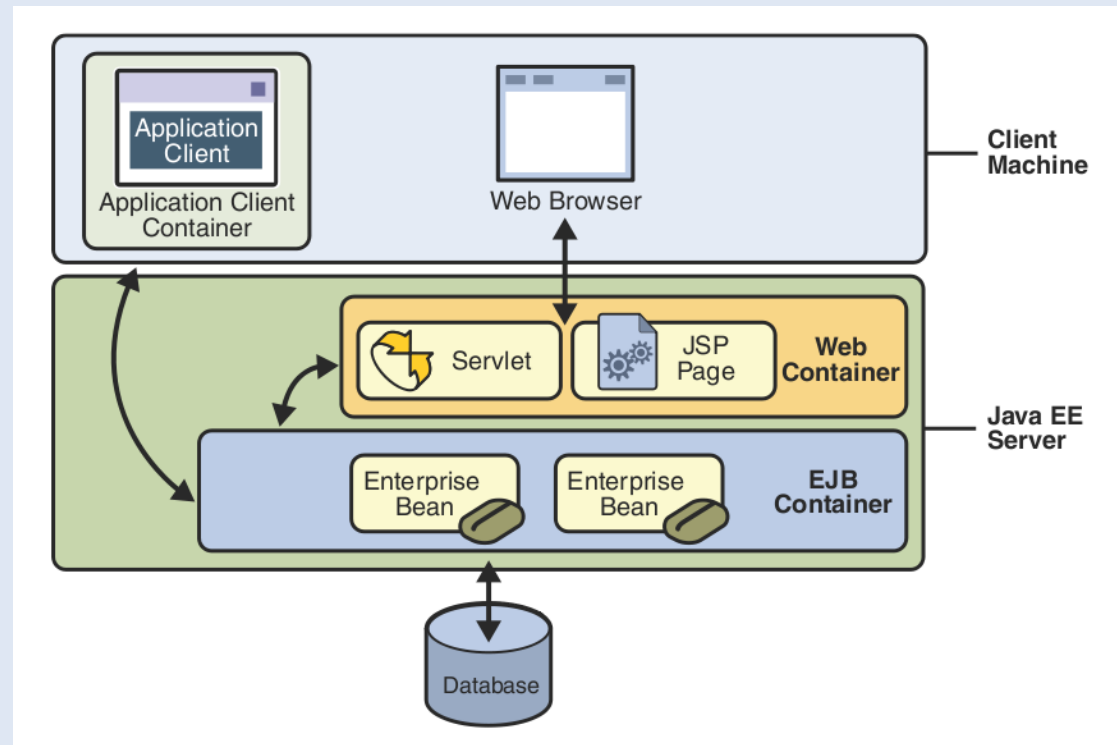
Injection de dépendance, instantiation implicite !
(possibilité d'utiliser la méthode lookup() de JNDI)

Exemple de session...

Destruction explicite par appel de la méthode @Remove

Exemple Persistant

- Enterprise Application
 - Classe Entity : User.java
 - Façade pour l'Entity : UserFacade.java (Session Bean Stateless)
 - Servlet : UserServlet.java



Exemple Persistent

- User.java (Entity Class)

`@Entity`

```
public class User implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    private String firstname;
    private String lastname;

    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public String getFirstname() { return firstname; }
    public void setFirstname(String firstname) { this.firstname = firstname; }
    public String getLastname() { return lastname; }
    public void setLastname(String lastname) { this.lastname = lastname; }

    /* ... */
}
```

Exemple Persistent

- UserFacade.java (Session Bean for Entity...)

`@Stateless`

```
public class UserFacade {
```

```
    @PersistenceContext(unitName = "MyFirstEntityApp-ejbPU")  
    private EntityManager em;
```

```
    public void create(User user) {  
        em.persist(user); // stockage dans la database...  
    }
```

```
    public void edit(User user) {  
        em.merge(user);  
    }
```

```
    public void remove(User user) {  
        em.remove(em.merge(user));  
    }
```

```
    public User find(Object id) {  
        return em.find(User.class, id);  
    }
```

```
    public List<User> findAll() {  
        CriteriaQuery cq = em.getCriteriaBuilder().createQuery();  
        cq.select(cq.from(UserEntity.class));  
        return em.createQuery(cq).getResultList();  
    }
```

Exemple Persistent

- UserServlet.java

```
@WebServlet(name = "UserServlet", urlPatterns = {"/UserServlet"})  
public class UserServlet extends HttpServlet {
```

```
    @EJB
```

```
    private UserFacade userFacade;
```

```
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {
```

```
        String firstname = request.getParameter("firstname");  
        String lastname = request.getParameter("lastname");
```

```
        if ((firstname != null) && (lastname != null)) {
```

```
            User user = new User();  
            user.setFirstname(firstname);  
            user.setLastname(lastname);  
            userFacade.create(user);  
            firstname = lastname = null;
```

```
        }
```

```
        response.setContentType("text/html;charset=UTF-8");
```

```
        PrintWriter out = response.getWriter();
```

```
        out.println("<html>");
```

```
        /* ... */
```

```
        out.println("<h2>Add User</h2>");
```

```
        out.println("<form>");
```

```
        out.println("First Name: <input type='text' name='firstname'><br/>");
```

```
        out.println("Last Name: <input type='text' name='lastname'><br/>");
```

```
        out.println("<input type='submit'><br/>");
```

```
        out.println("</form>");
```

```
        /* ... */
```


Le mot de la fin...

- De l'impossibilité de développer des applications d'entreprise “*from scratch*”
- Java EE, une solution intéressante
 - Le développeur de l'EJB se concentre uniquement sur la logique métier
 - Accès aux services du système (transaction, sécurité) via le container de l'EJB
 - Le développeur du client se concentre uniquement sur la présentation
 - Client léger sans “code-métier”, ni accès à la base de données et pouvant être déployé sur des petits “devices”
 - Les EJBs sont portables et peuvent être réutilisés et assemblés pour construire de nouvelles applications...
 - Communauté très active...

Annexes

EJB 2

Converter.java (EJB 2)

- L'interface distante du composant

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;
import java.math.*;

public interface Converter extends EJBObject {

    public BigDecimal dollarToYen(BigDecimal dollars)
        throws RemoteException;

    public BigDecimal yenToEuro(BigDecimal yen)
        throws RemoteException;
}
```

ConverterHome.java (EJB 2)

- L'interface distante de la maison du composant

```
import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;

public interface ConverterHome extends EJBHome {

    Converter create() throws RemoteException,
    CreateException;

}
```

ConverterBean.java (EJB 2)

- Le code du composant (Session Bean)

```
import java.rmi.RemoteException;  
import javax.ejb.SessionBean;  
import javax.ejb.SessionContext;  
import java.math.*;
```

```
public class ConverterBean implements SessionBean {
```

```
    BigDecimal yenRate = new BigDecimal("121.6000");  
    BigDecimal euroRate = new BigDecimal("0.0077");
```

```
    public BigDecimal dollarToYen(BigDecimal dollars)  
        throws RemoteException { ... }
```

```
    public BigDecimal yenToEuro(BigDecimal yen)  
        throws RemoteException { ... }
```

Business methods

../..

ConverterBean.java (EJB 2)

- La suite du code...

```
public ConverterBean() {}  
public void ejbCreate() {}  
public void ejbRemove() {}  
public void ejbActivate() {}  
public void ejbPassivate() {}  
public void setSessionContext(SessionContext sc) {}  
  
}
```

Callback methods
(required by EJB spec.)

ConverterClient.java (EJB 2)

- Le code de l'application cliente

```
import javax.naming.Context;  
import javax.naming.InitialContext;  
import javax.rmi.PortableRemoteObject;  
import java.math.BigDecimal;
```

```
public class ConverterClient {  
    public static void main(String[ ] args) {  
        try {  
            Context initial = new InitialContext();  
            Context myEnv = (Context) initial.lookup("java:comp/env");  
            Object objref = myEnv.lookup("ejb/SimpleConverter");  
  
            ConverterHome home =  
                (ConverterHome) PortableRemoteObject.narrow(objref,  
                    ConverterHome.class);
```

} Localisation de
la maison via
l'API JNDI

../..

ConverterClient.java (EJB 2)

- La suite...

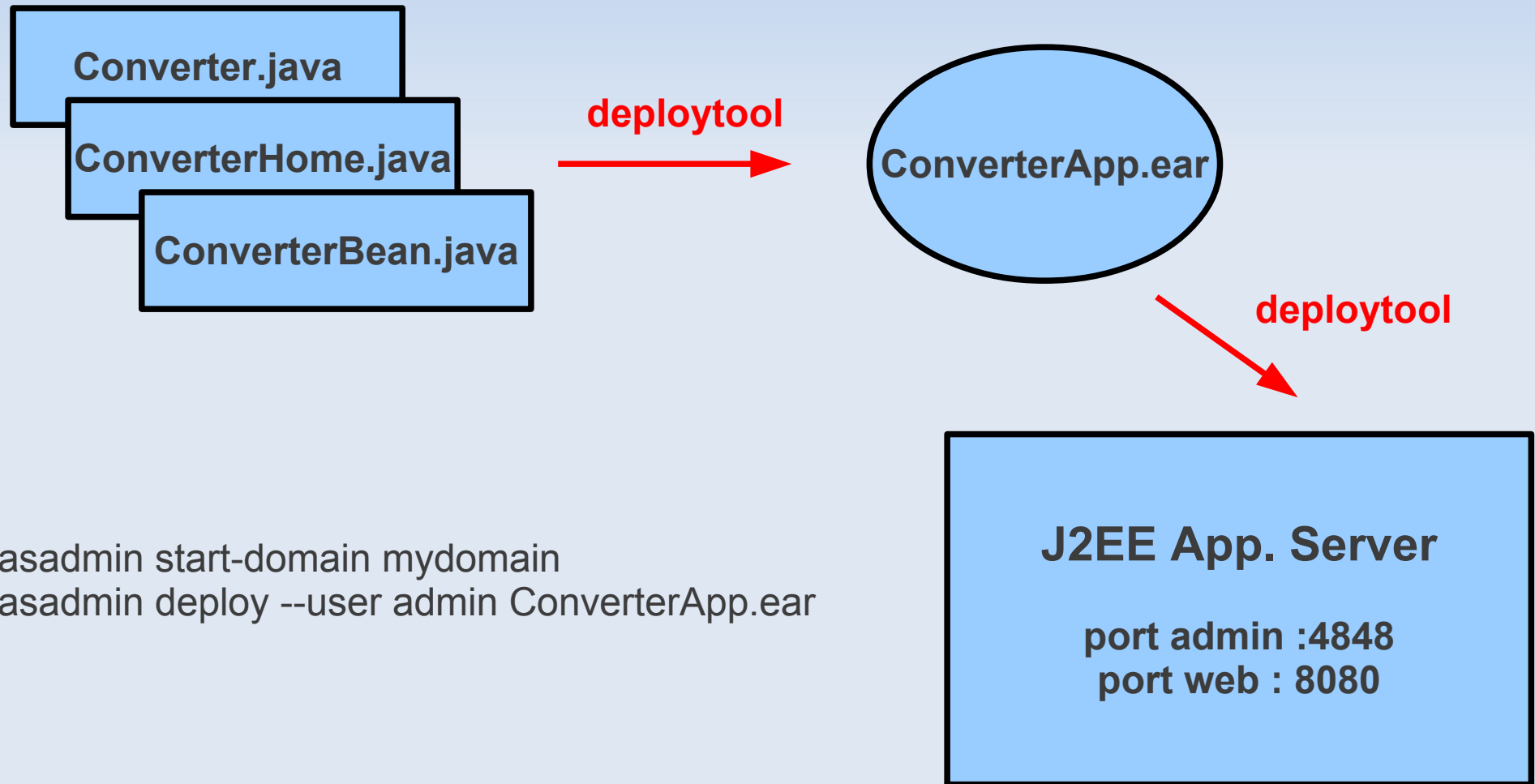
```
    Converter currencyConverter = home.create();

    BigDecimal param = new BigDecimal("100.00");
    BigDecimal amount =
currencyConverter.dollarToYen(param);

    System.out.println(amount);
    amount = currencyConverter.yenToEuro(param);
    System.out.println(amount);

    System.exit(0);
} catch (Exception ex) {
    System.err.println("Caught an unexpected exception!");
    ex.printStackTrace();
}
}
```


Empaquetage & Déploiement



JNDI

- JNDI (Java Native Directory Interface)
 - API Java pour les annuaires (notamment LDAP)
 - Permet au client de localiser la maison du composant
 - JNDI name : “java:comp/env/ejb/greeter”

```
javax.naming.Context ctxt = new javax.naming.InitialContext();  
java.lang.Object obj = ctxt.lookup("java:comp/env/ejb/greeter");
```

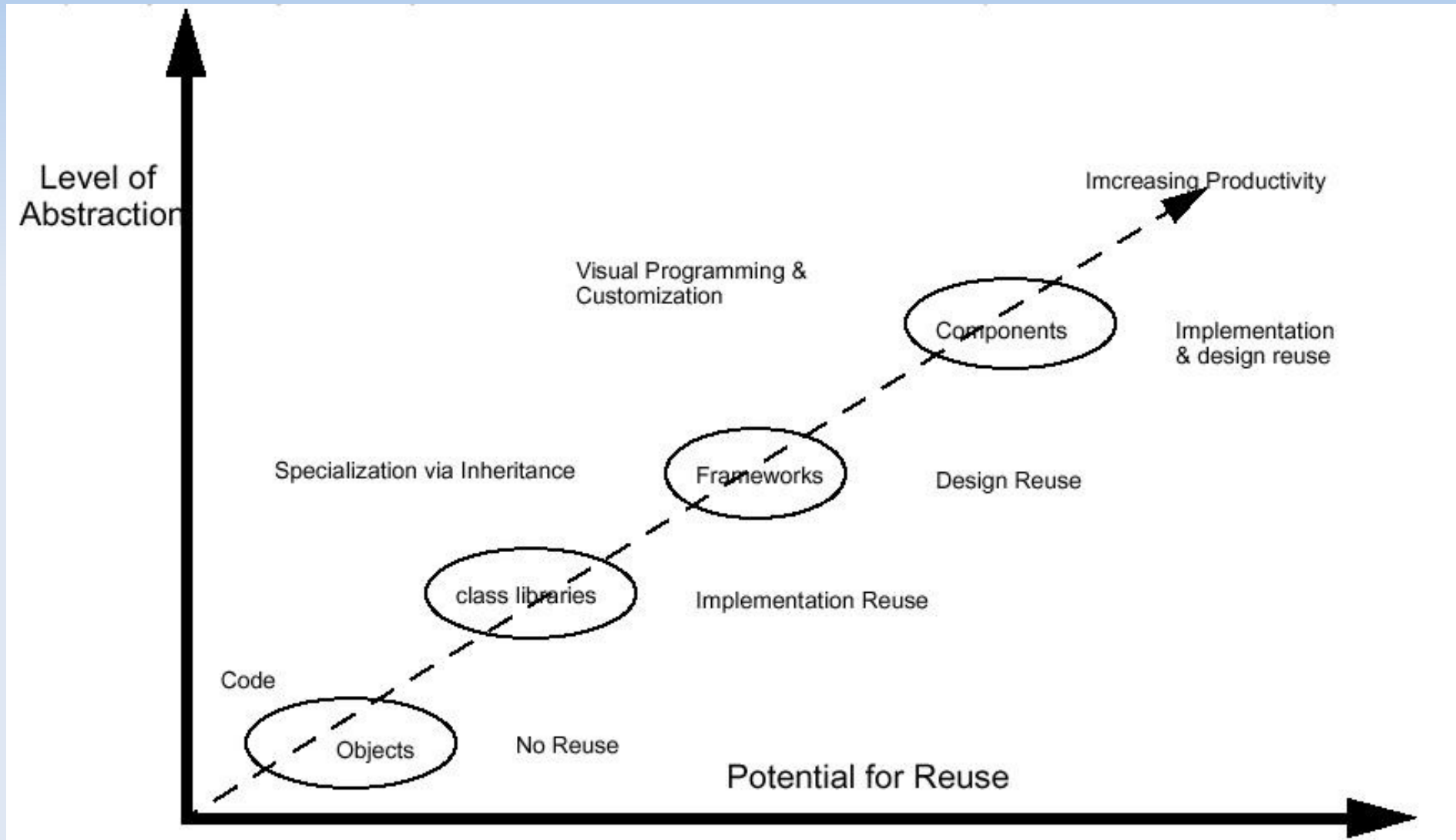
Fractal

D'après Lionel Seinturier, ICAR 2008.

Les composants logiciels

- Objectifs
 - diminuer la complexité des applications logicielles
 - améliorer la qualité logicielle, augmenter la productivité
 - faciliter et unifier la conception, le développement, la maintenance, le déploiement et l'administration
- Composant vs Objet
 - plus haut-niveau d'abstraction (connexions plus explicites)
 - meilleur réutilisabilité, encapsulation, autonomie, ...
- Différents environnements
 - EJB, CCM, COM, .NET, Fractal, CCA, ...

Réutilisabilité

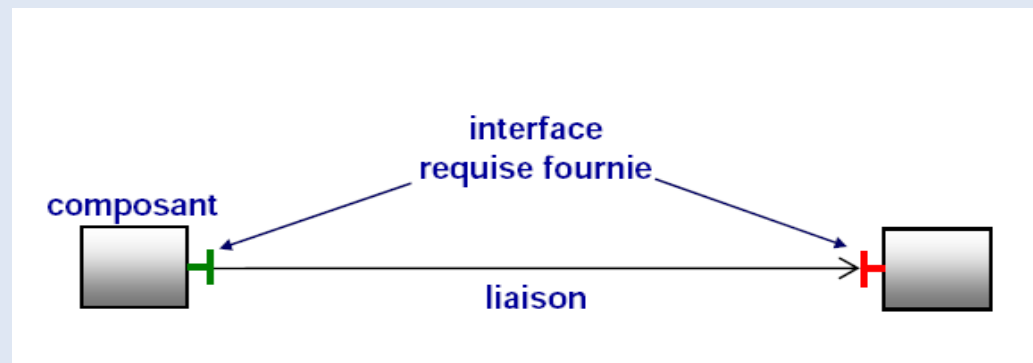


Définitions

- Composant
 - A component is a unit of composition with **contractually specified interfaces** and **context dependencies** only. A software component can be **deployed** independently and is subject to **composition** by third parties. [Szyperski 97]
- Architecture Logicielle
 - A software architecture of a program or computing system is the structure or structures of the system, which comprise **software components**, the externally visible **properties**, and the **relationships** among them. [Bass 98]

Introduction

- Modèle de composant
 - Un composant fournit et/ou requiert une ou plusieurs interfaces
 - Une liaison est un chemin de communication entre une interface requise et une interface fournie

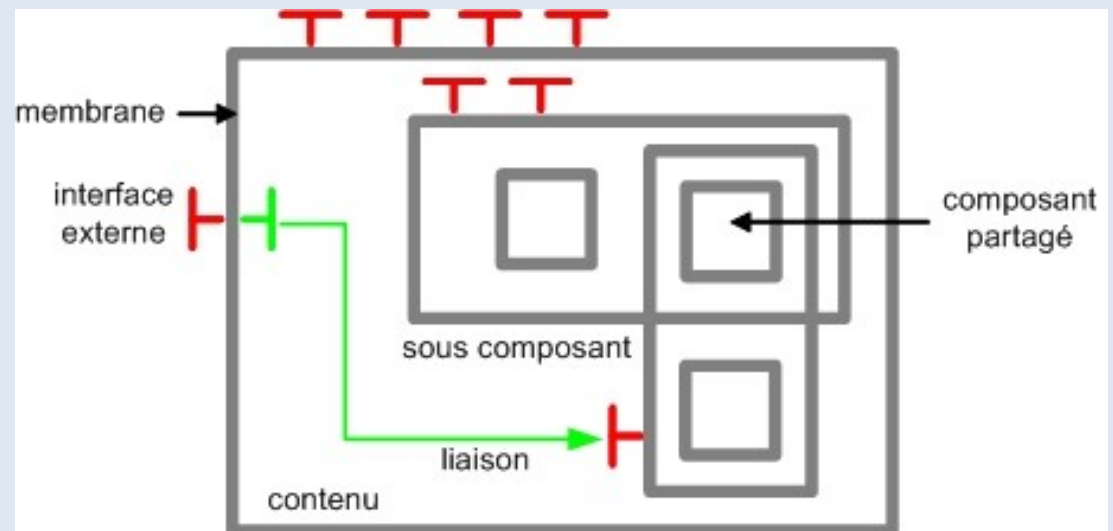


- Architecture
 - Composants assemblés pour construire une architecture

Fractal

- Fractal
 - Modèle de composant open source et extensible
 - Développé à partir de 2000, 1^{ère} version en 2002
 - Implantation en Java, C, C++, SmallTalk, .NET
 - <http://fractal.objectweb.org> (ObjectWeb consortium)

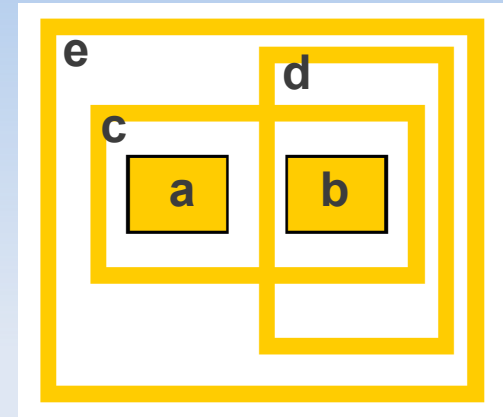
- Modèle
 - Composant
 - Interface
 - Liaison
 - Instanciation



Composant

- Caractéristiques

- Modèle hiérarchique (**récurtivité**)
 - Composant composite ou primitif
- Notion de **partage** de composant
- Introspection (**reflexivité**)

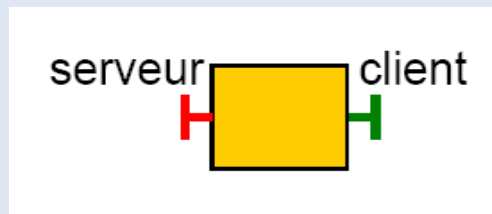


- 2 dimensions

- Métier (fonctionnel)
 - Ce que doit faire l'application...
- Contrôle (extra-fonctionnel)
 - Mis en oeuvre par une **membrane** composée d'un ensemble de **contrôleurs** accessible via une interface

Interface

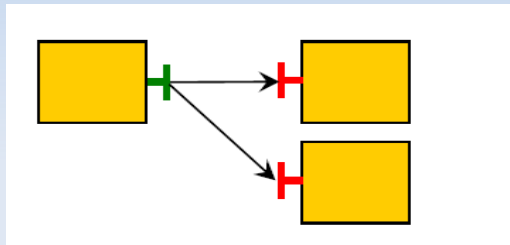
- Point d'accès à un composant
 - Interface nommée et typée (signature)
 - Plusieurs interfaces possibles par composant
 - Interface serveur ou client
 - Serveur : service fournis
 - Client : service requis (nécessaires au fonctionnement du composant)



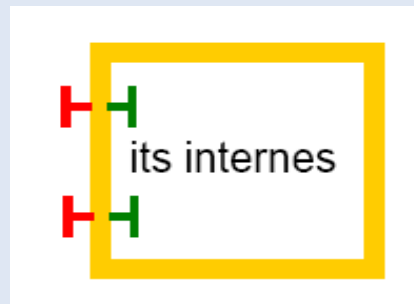
- Interface obligatoire ou optionnelle

Interface

- Interface simple ou multiple
 - Cardinalité de la liaison (référence ou collection de références)

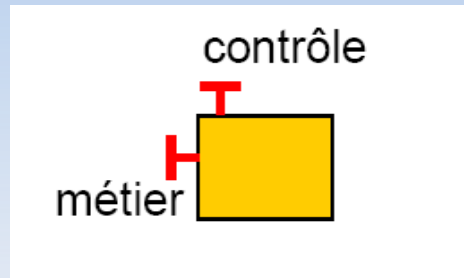


- Interface interne ou externe
 - Externe : en façade d'un composant ou d'un composite
 - Interne : pendant d'une interface externe pour un composite



Interface

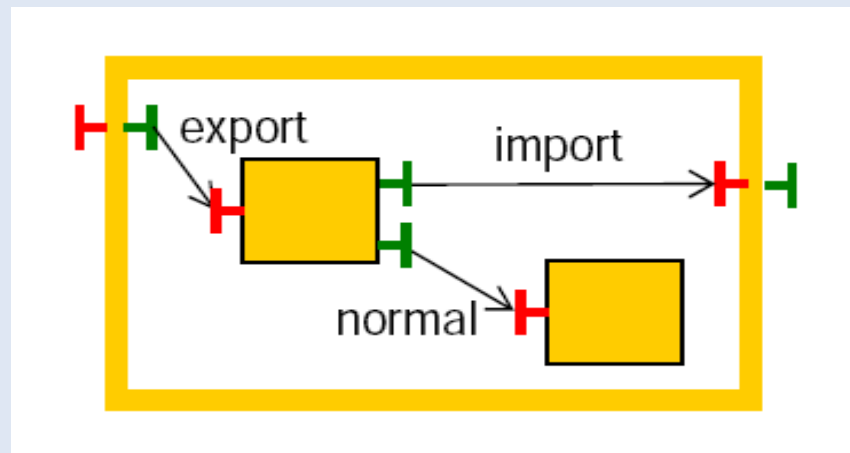
- Interface métier ou contrôle



- Interfaces de contrôle prédéfinies dans Fractal
 - BC (binding-controller) : gestion des liaisons
 - LC (lifecycle-controller) : cycle de vie d'un composant
 - NC (name-controller) : nommer le composant
 - SC (super-controller) : le composite contenant le composant
 - CC (content-controller) : gérer le contenu d'un composite

Liaison (binding)

- Chemin de communication entre composants
 - Plus précisément, entre une interface client et une interface serveur
 - Explicite et matérialise les dépendances entre composants
 - Liaison primitive vs composite
 - Primitive : invocation méthode locale (même espace adressage)
 - Composite : invocation méthode distante (?)



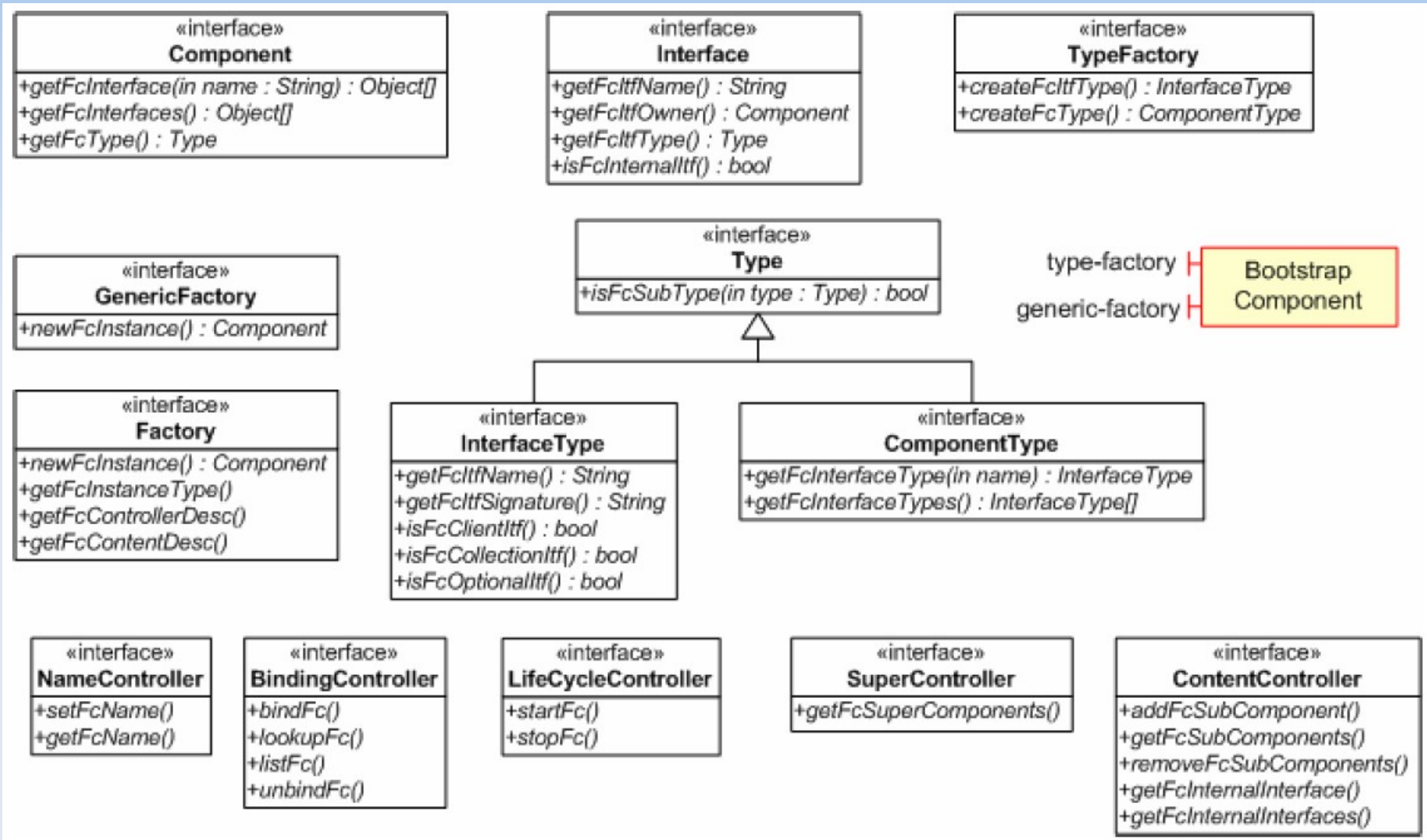
Instanciación

- Créer une instance à partir d'une définition de composant
 - Utilisation d'une fabrique, le Bootstrap Component
 - Prédéfini dans Fractal qui permet d'instancier n'importe quel composant...

Développer avec Fractal

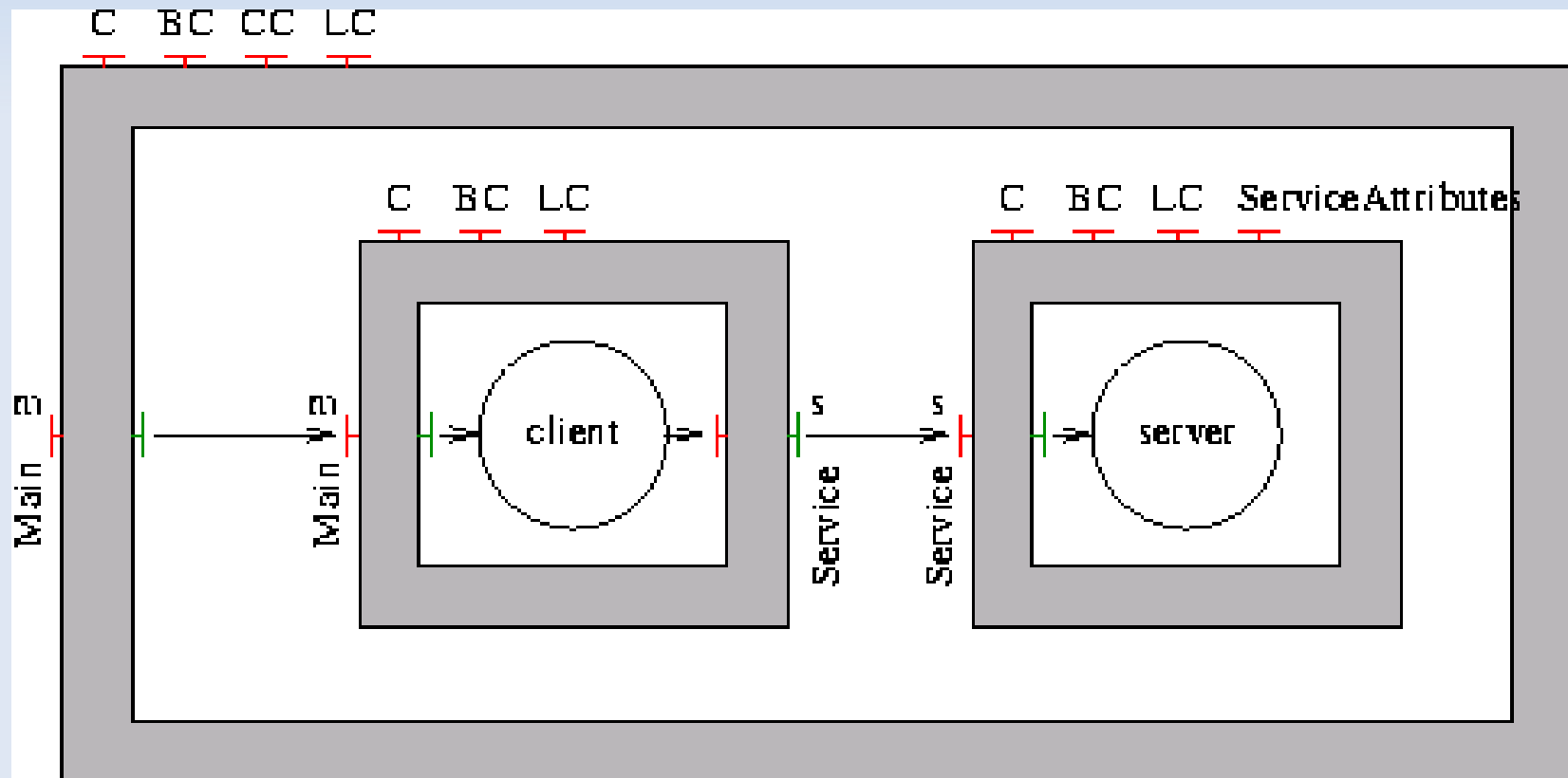
- Développer des applications Fractal en Java
 - Fractal API : ensemble d'interface Java pour l'introspection, la création et la reconfiguration dynamique des composants
- 2 outils complémentaires
 - Fractal ADL : langage de description d'architecture (XML)
 - Fraclet : modèle de programmation à base d'annotations (@)
- Implantations
 - Julia : implantation de référence en Java
 - AOKell : une autre implantation Java “orientée aspect”

Fractal API



Exemple "Hello World"

- Démo en TP avec Julia...
 - Version "pure-java" vs ADL



Exemple “Hello World”

- Les interfaces

```
public interface Main {  
    void main ();  
}
```

```
public interface Service {  
    void print (String msg);  
}
```

interfaces “métier”

```
public interface ServiceAttributes extends AttributeController {  
    String getHeader ();  
    void setHeader (String header);  
}
```

interface de contrôle
pour les attributs
(méthodes get/set)

Exemple “Hello World”

- Implantation du composant Server

```
public class ServerImpl implements Service, ServiceAttributes {  
    private String header;
```

```
    public ServerImpl () {  
        System.err.println("Server created");  
    }
```

```
    public void print (String msg) {  
        System.out.println(header + msg);  
    }
```

```
    public String getHeader () {  
        return header;  
    }
```

```
    public void setHeader (final String header) {  
        this.header = header;  
    }
```

```
}
```



implantation “métier”



implantation des “attributs”

Exemple “Hello World”

- Implantation du composant Client

```
public class ClientImpl implements Main, BindingController {
```

```
    private Service service;
```

```
    public ClientImpl () {  
        System.err.println("Client created");  
    }
```

```
    public void main () {  
        service.print("Hello world !");  
    }
```

```
}
```

implantation “métier”

../..

Exemple "Hello World"

../..

```
public String[] listFc () {  
    return new String[] { "s" };  
}  
  
public Object lookupFc (final String cltf) {  
    if (cltf.equals("s")) return service;  
    return null;  
}  
  
public void bindFc (final String cltf, final Object sltf) {  
    if (cltf.equals("s")) service = (Service)sltf;  
}  
  
public void unbindFc (final String cltf) {  
    if (cltf.equals("s")) service = null;  
}  
}
```

implantation du
binding-controller
pour la gestion d'une
connexion cliente

Exemple "Hello World"

- Programme principal (version "pure-java")

```
public class HelloWorld {
```

```
    public static void main (final String[] args) throws Exception {
```

```
        // 0.a) get bootstrap component provided by Fractal
```

```
        Component boot = Fractal.getBootstrapComponent();
```

```
        // 1) get useful factory to create types
```

```
        TypeFactory tf = Fractal.getTypeFactory(boot);
```

```
        // 1.a) create type of root component
```

```
        ComponentType rType = tf.createFcType(new InterfaceType[] {  
            tf.createFcltfType("m", "Main", false, false, false)  
        });
```

```
        // 1.b) create type of client component
```

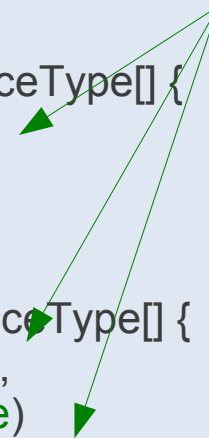
```
        ComponentType cType = tf.createFcType(new InterfaceType[] {  
            tf.createFcltfType("m", "Main", false, false, false),  
            tf.createFcltfType("s", "Service", true, false, false)  
        });
```

TypeFactory.createFcltfType(...)

- name

- signature (class name)

- isClient, isOptional, isCollection



Exemple “Hello World”

../..

// 1.c) create type of server component

```
ComponentType sType = tf.createFcType(new InterfaceType[] {  
    tf.createFcltfType("s", "Service", false, false, false),  
    tf.createFcltfType("attribute-controller", "ServiceAttributes", false, false, false)  
});
```

GenericFactory.newFcInstance(...)

- interface type
- “composite” or “primitive”
- class name (or null if no content)

// 2) get useful factory to instantiate components

```
GenericFactory cf = Fractal.getGenericFactory(boot);
```

// 2.a) create root component

```
Component rComp = cf.newFcInstance(rType, "composite", null);
```

// 2.b) create client component

```
Component cComp = cf.newFcInstance(cType, "primitive", "ClientImpl");
```

// 2.c) create server component

```
Component sComp = cf.newFcInstance(sType, "primitive", "ServerImpl");
```

// 3.a) component assembly

```
Fractal.getContentController(rComp).addFcSubComponent(cComp);
```

```
Fractal.getContentController(rComp).addFcSubComponent(sComp);
```

// 3.b) component bindings

```
Fractal.getBindingController(rComp).bindFc("m", cComp.getFcInterface("m"));
```

```
Fractal.getBindingController(cComp).bindFc("s", sComp.getFcInterface("s"));
```

../..

Exemple "Hello World"

```
../..
```

```
// 4) configuration of attributes
```

```
ServiceAttributes sa = (ServiceAttributes)sComp.getFcInterface("attribute-controller");  
sa.setHeader("->");
```

```
// 5) start the root component
```

```
Fractal.getLifeCycleController(rComp).startFc();
```

```
// 6) call the entry point of the application
```

```
((Main)rComp.getFcInterface("m")).main();
```

```
}
```

```
}
```

- Compilation et lancement de l'exemple avec **ant**...

```
$ ant compile run
```

```
compile:
```

```
[javac] Compiling 6 source files...
```

```
run:
```

```
[java] Client created
```

```
[java] Server created
```

```
[java] ->Hello world !
```


Exemple “Hello World” (ADL)

- Programme principal (version ADL)

```
public class HelloWorld {
```

```
    public static void main(String[] args) throws Exception {
```

```
        // 1) load the ADL definition
```

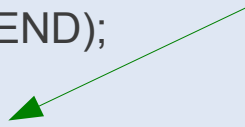
```
        Factory factory =
```

```
            FactoryFactory.getFactory(FactoryFactory.FRAGMENTAL_BACKEND);
```

```
        Component root = (Component)
```

```
            factory.newComponent("ClientServerImpl", new HashMap());
```

optional string
parameters used in
ADL as \${key}



```
        // 2) start the root component
```

```
        Fractal.getLifeCycleController(root).startFc();
```

load “ClientServerImpl.fractal”



```
        // 3) call the entry point of the application
```

```
        ((Main)root).getFcInterface("m").main();
```

```
    }
```

```
}
```

Exemple "Hello World" (ADL)

- ClientServerImpl.fractal

```
<definition name="ClientServerImpl">
```

```
<interface name="m" role="server" signature="Main"/>
```

```
<component name="client" definition="ClientImpl" />
```

```
<component name="server" definition="ServerImpl" />
```

```
<binding client="this.m" server="client.m" />
```

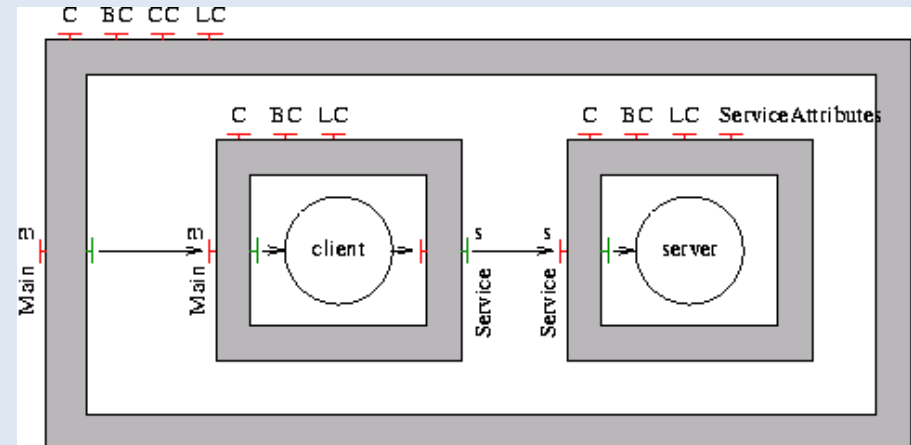
```
<binding client="client.s" server="server.s" />
```

```
</definition>
```

components defined in
"ServerImpl.fractal" and
"ClientImpl.fractal"

sub-components

bindings



Exemple "Hello World" (ADL)

- ServerImpl.fractal

```
<definition name="ServerImpl">  
  <interface name="s" role="server" signature="Service"/>  
  <content class="ServerImpl"/>  
  <attributes signature="ServiceAttributes">  
    <attribute name="header" value="->"/>  
  </attributes>  
  <controller desc="primitive"/>  
</definition>
```

contents defined in classes
"ServerImpl.java" and
"ClientImpl.java"

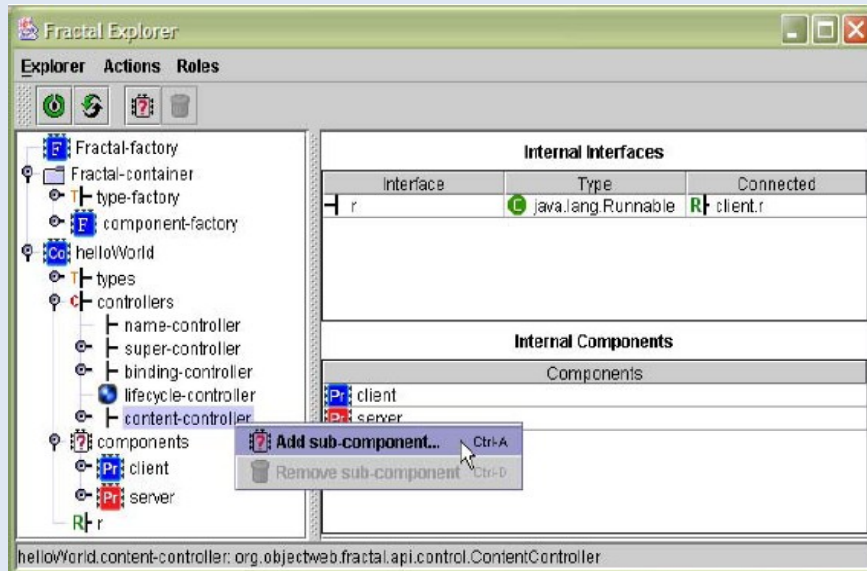
- ClientImpl.fractal

```
<definition name="ClientImpl">  
  <interface name="m" role="server" signature="Main"/>  
  <interface name="s" role="client" signature="Service" />  
  <content class="ClientImpl"/>  
</definition>
```

Quelques outils...

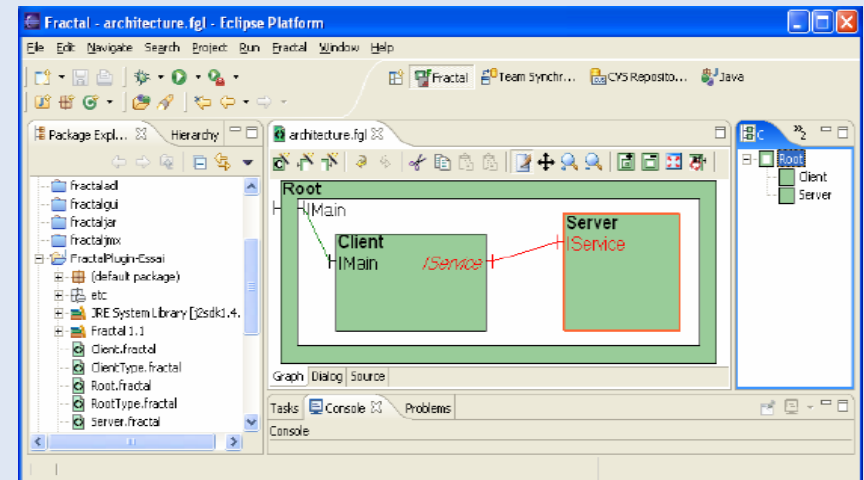
- FractalExplorer

- Console d'administration
- Pilotage *run-time* d'une application fractal



- FractalGUI

- Outil de conception d'architecture Fractal
- Génération de squelette de code



Binding RMI

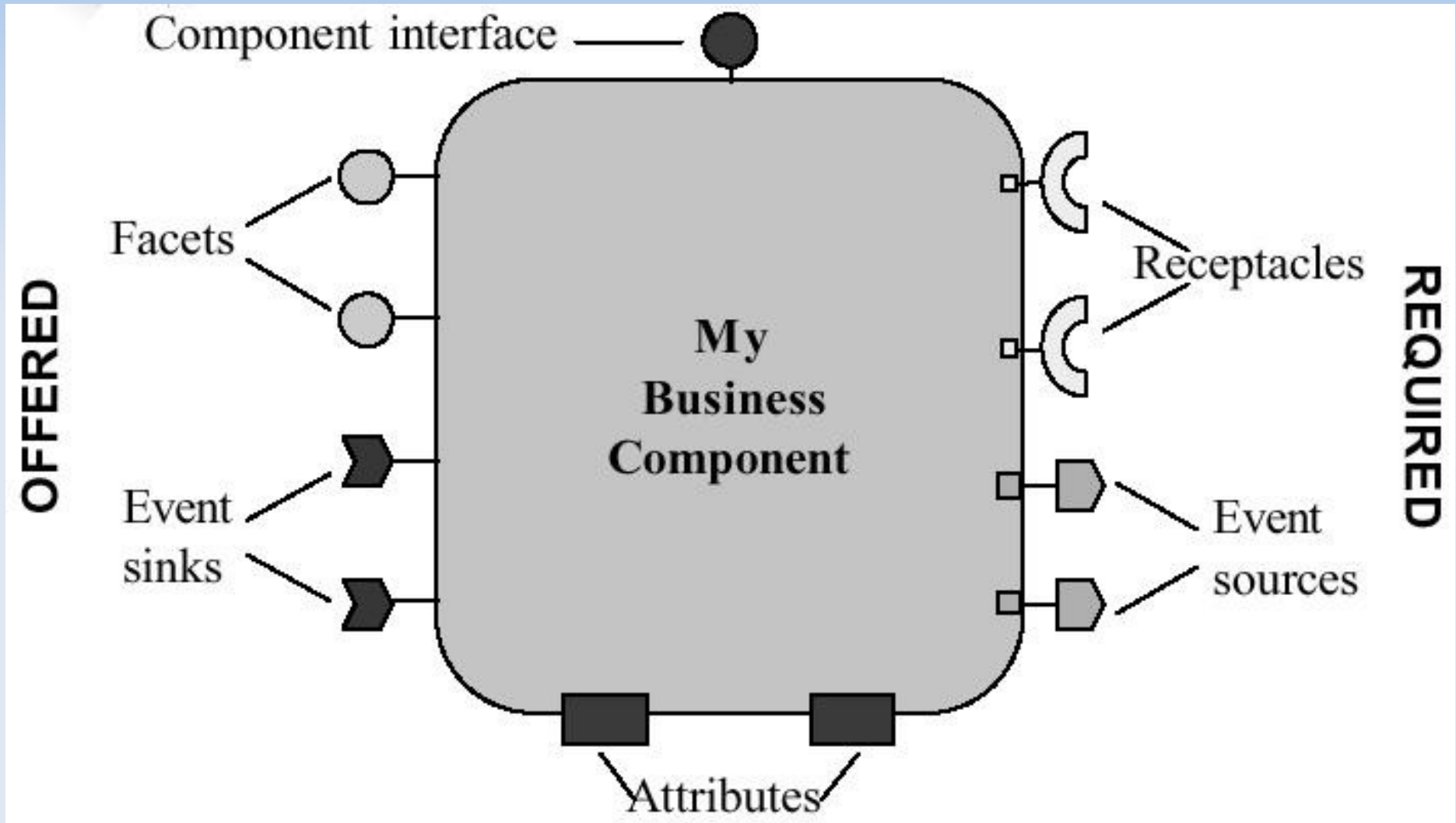
- A compéter...

CCM (CORBA Component Model)

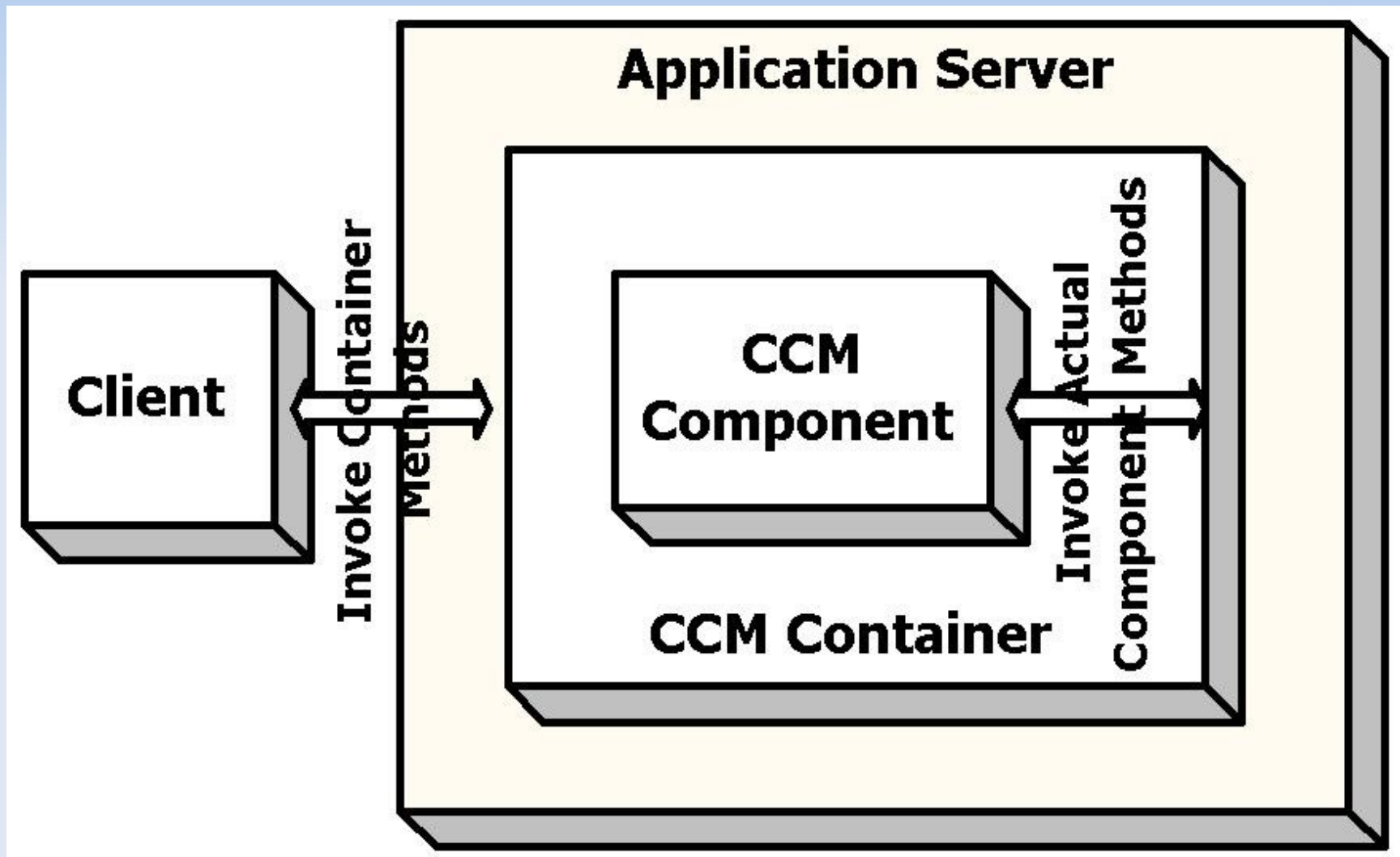
CCM (CORBA Component Model)

- Le modèle abstrait
 - IDL3
- Le modèle d'implantation
 - CIDL/CIF
- Le modèle des conteneurs
- Le modèle de déploiement
 - distribution, assemblage, déploiement

Modèle abstrait



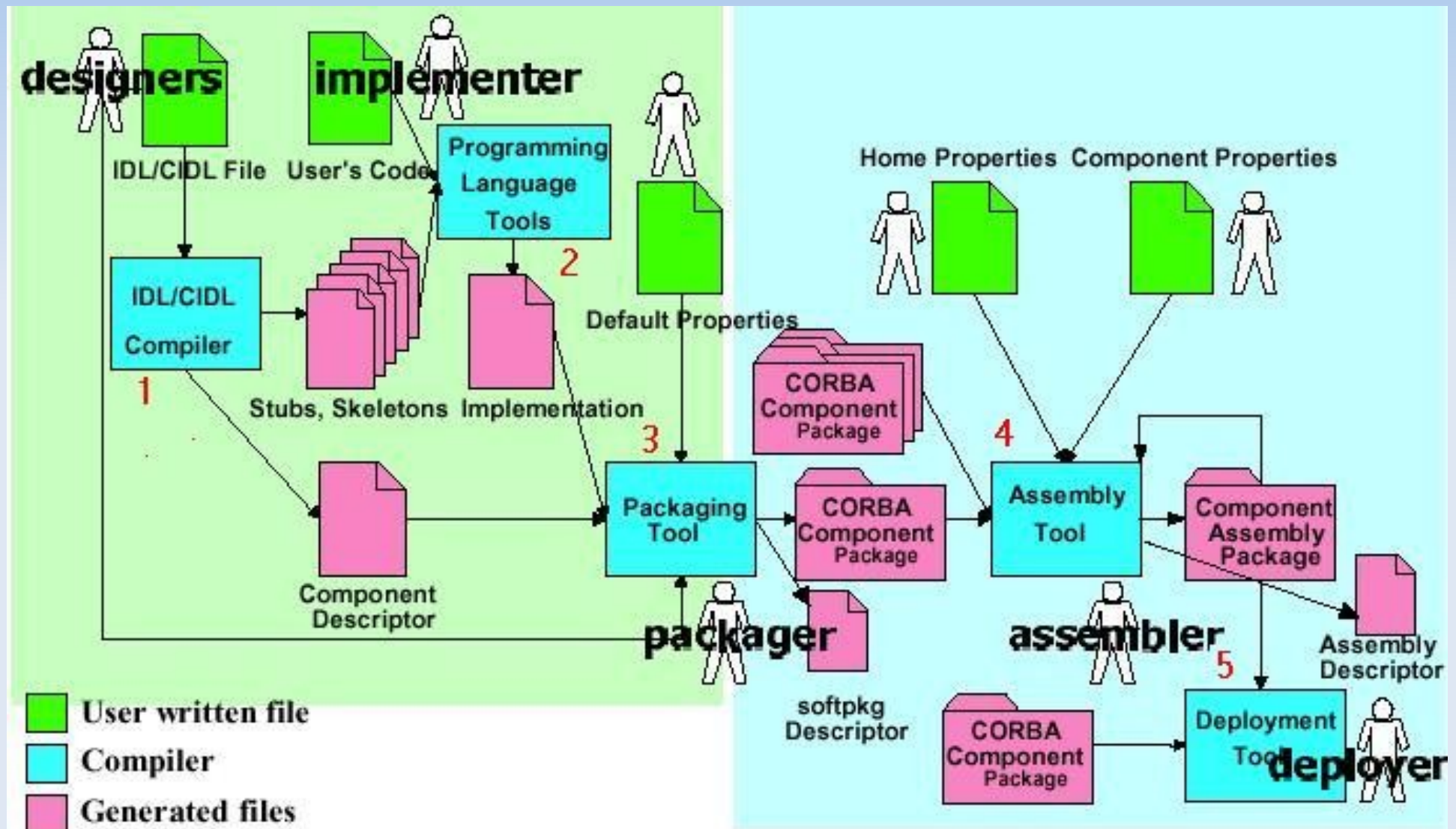
Modèle des conteneurs



Les cinq étapes...

- Modèle abstrait et modèle d'implantation du composant
- Implantation du composant
- Packaging
- Assemblage
- Déploiement

Les cinq étapes...



- A compléter...

P2P

JXTA

Annexes

Compléments : Threads en Java

- Héritage de la classe Thread

```
class MyThread extends Thread {  
    public void run() {  
        /* point d'entrée du thread */  
    }  
}
```

```
public class TThreadsDemo {  
    public static void main (String[ ] args) {  
        /* démarrage du thread */  
        new MyThread().start();  
    }  
}
```

Compléments : Threads en Java

- Utilisation de l'interface "Runnable"

```
class MyThread implements Runnable {  
    public void run() {  
        /* point d'entrée du thread */  
    }  
}
```

```
public class MyThreadDemo {  
    public static void main (String[ ] args) {  
        /* démarrage du thread */  
        new Thread(new MyThread()).start();  
    }  
}
```