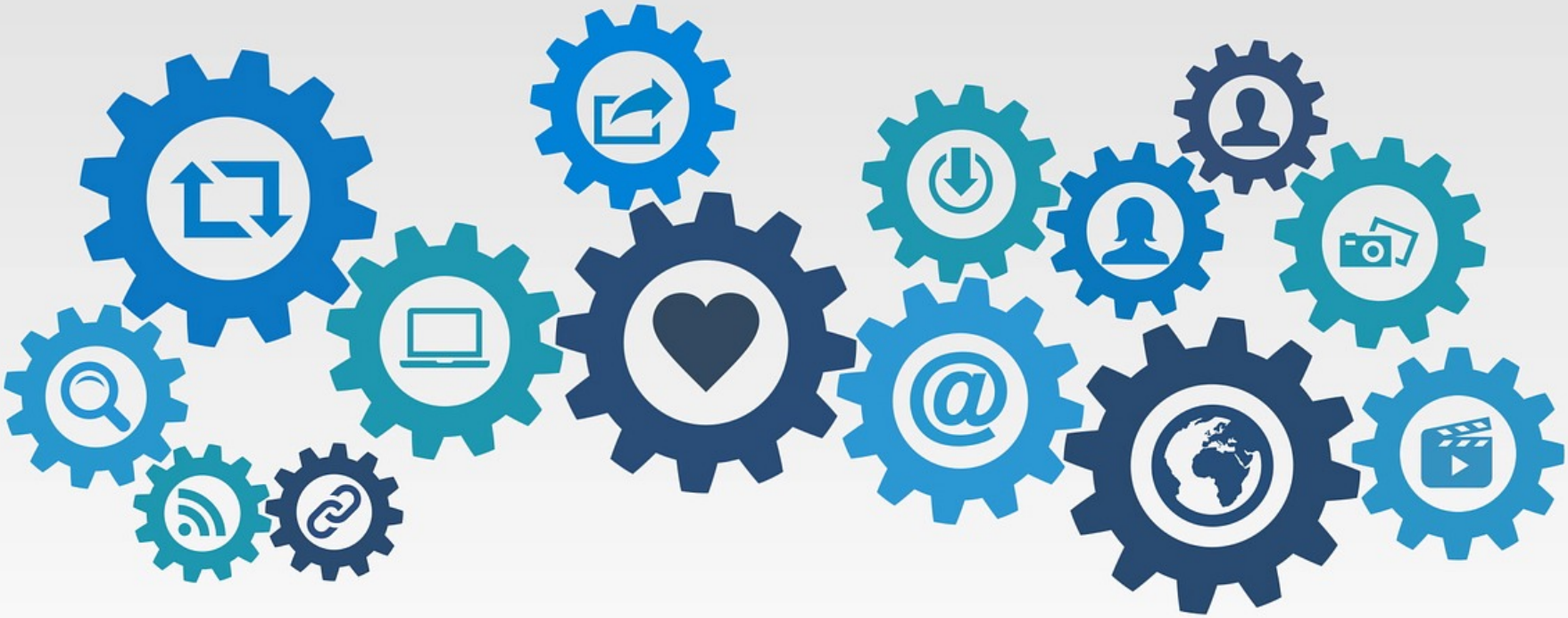
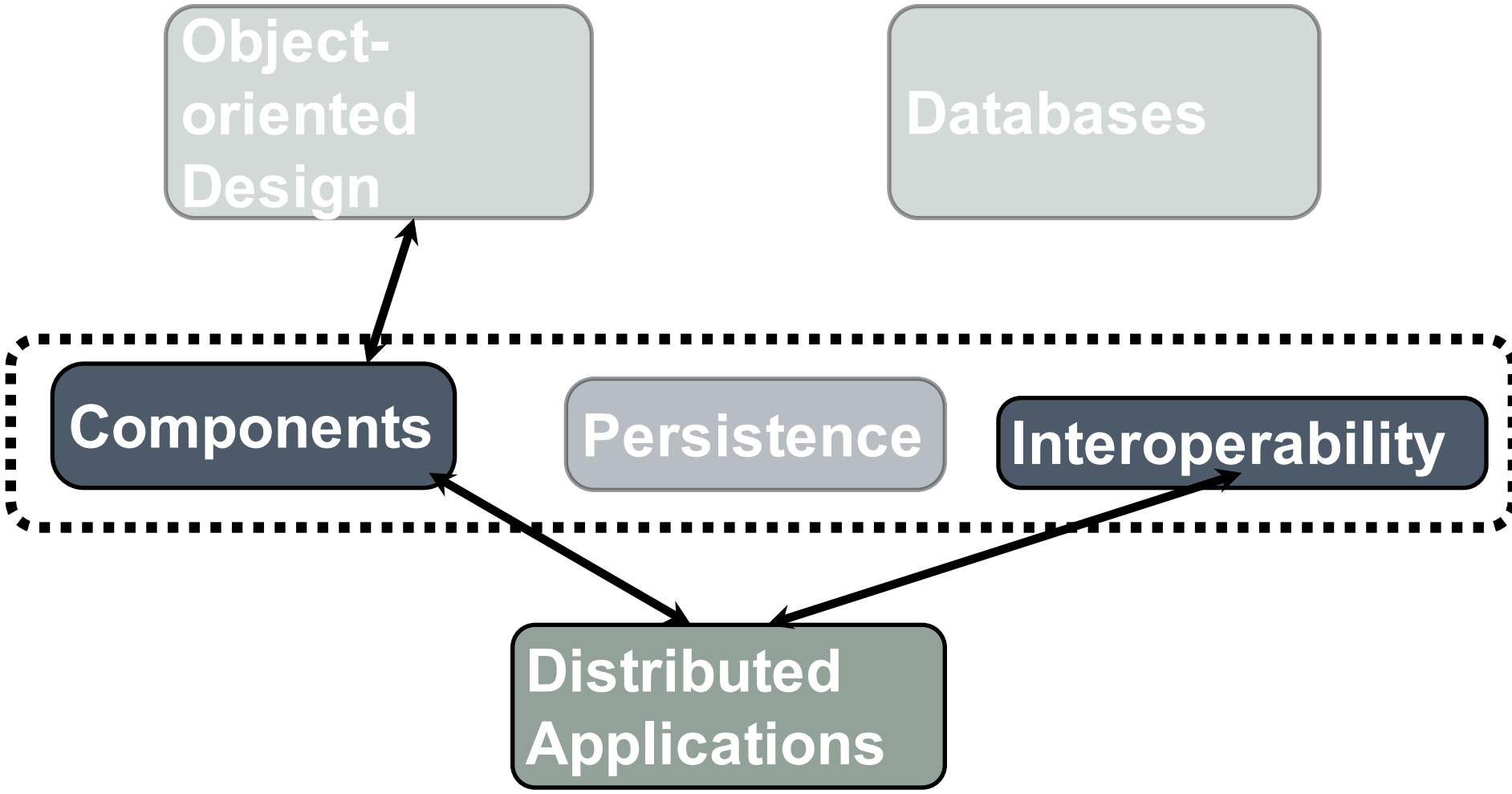


Client - Serveur
Interoperabilité
=> Composants

pictures: sxc.hu


Object-oriented Design
Databases
Components
Persistence
Interoperability
Distributed Applications

Voir ou revoir des concepts de base

- Protocoles de transports
- Protocoles d'application
- Distinguer:

1. transport des données
2. sérialisation des données
3. définition et gestion des services

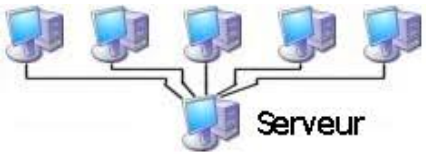
- Comprendre comment on définit des services
- Distinguer couplage fort et couplage faible

. RMI, Corba, Web Services, Composants J2E…

Client -,Serveur

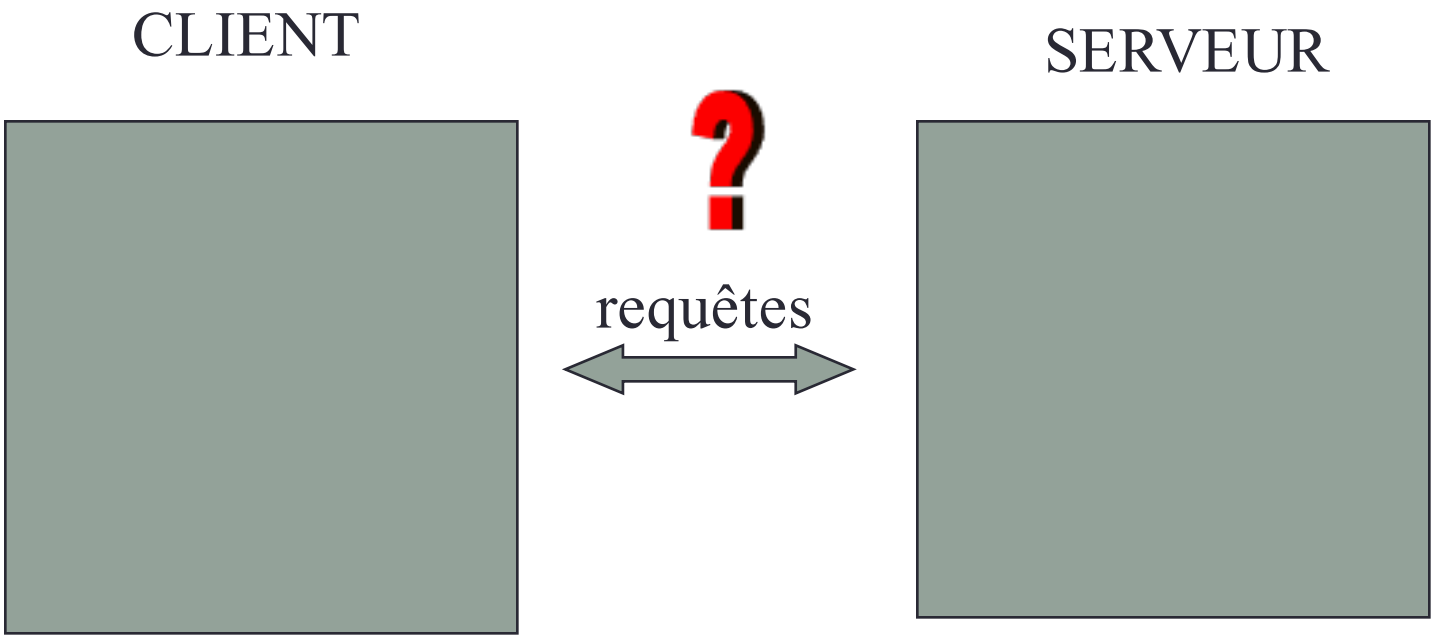
1/ Protocole d’application

2/ Marshalling Unmarshalling



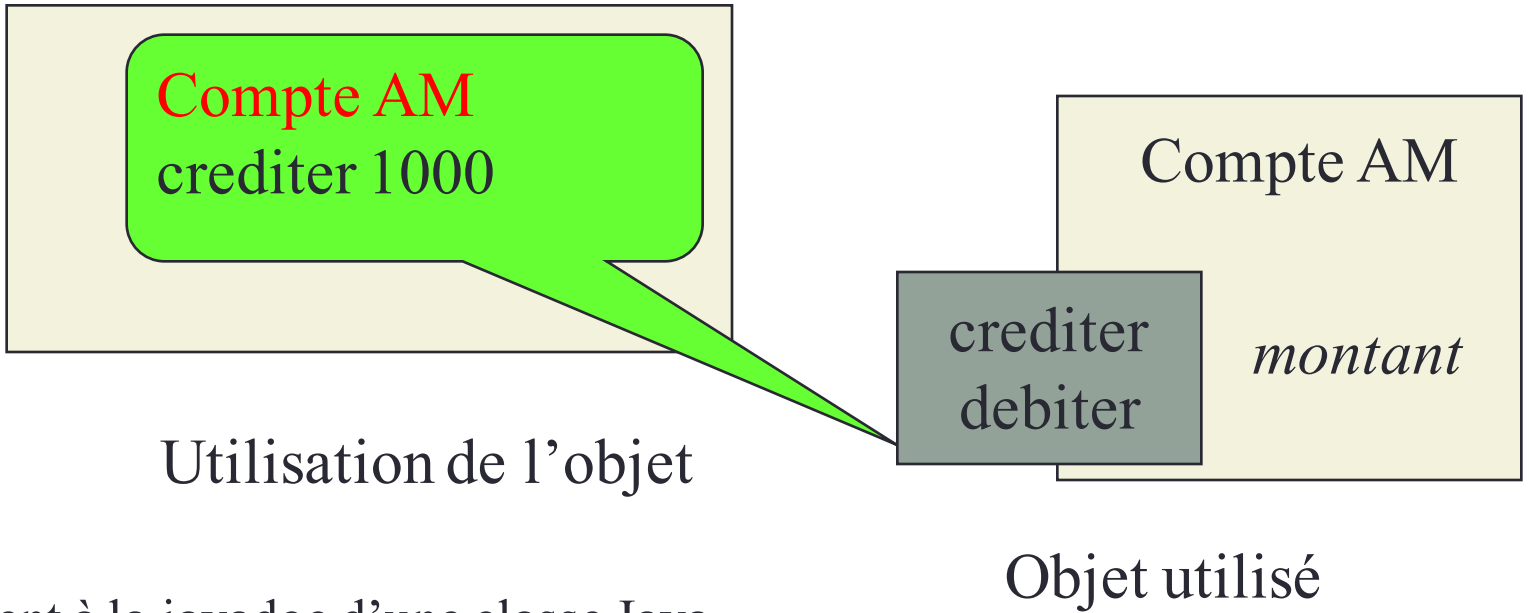
3/ Communication réseau

Requêtes Client Serveur : appeler les services proposés par le serveur



Ensemble des requêtes = protocole d'application du service

Exposer le protocole d'application : liste des services offerts par un serveur



Equivalent à la javadoc d'une classe Java
Interface = partie visible de l'objet
Implémentation = partie privée inaccessible depuis d'autres objets
Modifier l'implémentation des classes sans altérer l'utilisation
Interface = contrat entre l'objet et le monde extérieur

Protocole SMTP, RFC1822/3

HELO
MAIL From: pinna@essi.fr
RCPT To: pinna@essi.fr
DATA
From: pinna@essi.fr
Subject: Qui est là ?\n");
"Vous suivez toujours ?
QUIT

Approche réseau : Questions préliminaires

- Protocoles de transport TCP et UDP ?
- Utilisation des adresses Internet ?
- Utilisation des ports ?
- Programmation sockets : gestion d'entrées/sorties

Client : ?
Serveur : ?
Serveur de noms ? (DNS, LDAP) ?

Un peu de vocabulaire

Client : entité qui fait l'appel
Sockets : moyen de communication entre ordinateurs
Adresses IP : adresse d'un ordinateur
Serveur : entité qui prend en charge la requête
Serveur de noms (DNS) : correspondances entre noms logiques et adresses IP (Annuaire)
Port : canal dédié à un service
Protocole : langage utilisé par 2 ordinateurs pour communiquer entre eux

protocole de transport : comment véhiculer les données – construction de la trame réseau

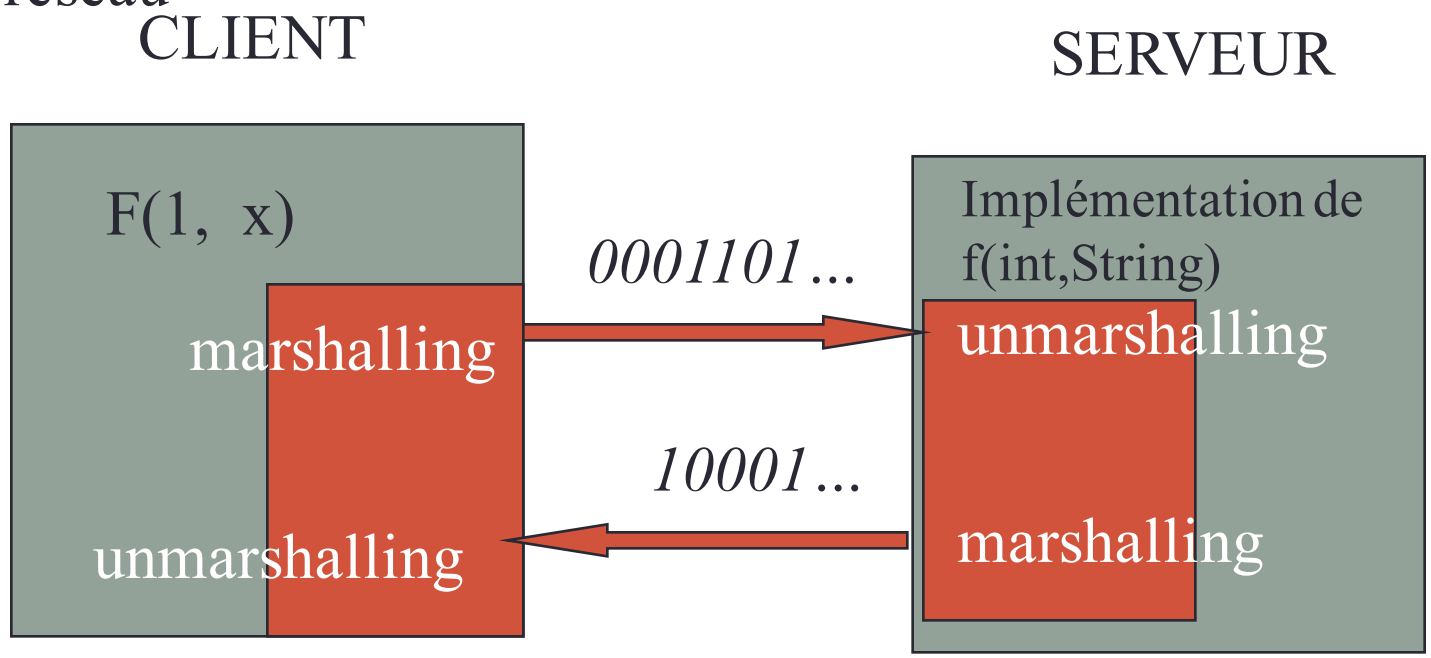
protocole d'application : comment le client et le serveur structurent les données échangées

Architecture client serveur

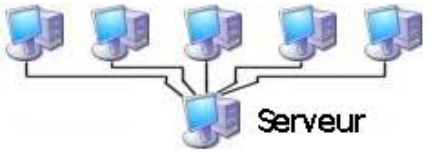
Un hôte établit une communication avec un autre hôte qui fournit un **service**

Appel à Distance

Marshalling : Préparer le format et le contenu de la trame réseau
Unmarshalling : Reconstruire l'information échangée à partir de la trame réseau

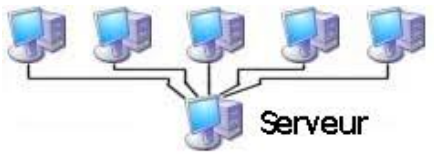


Comment cela fonctionne au niveau du réseau ?



- Identification de la machine qui abrite le serveur par le client
- Identification du serveur sur la machine
- Canal de communication entre le serveur et le client
- **Construction de la trame réseau**
- Echange **du protocole d'application au dessus d'un protocole de transport**

Programmation Socket : échanges sur le réseau



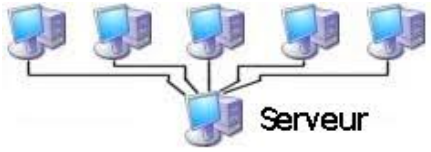
Les applications client/server communiquent via des sockets

- Deux types de transports via les socket API:
  - Datagramme (non reliable)
  - Orienté flux d'octets (reliable)

socket

Une porte à travers laquelle l'application peut à la fois envoyer et recevoir des messages d'une autre application

Exemple socket Java



Communication par flot de données TCP

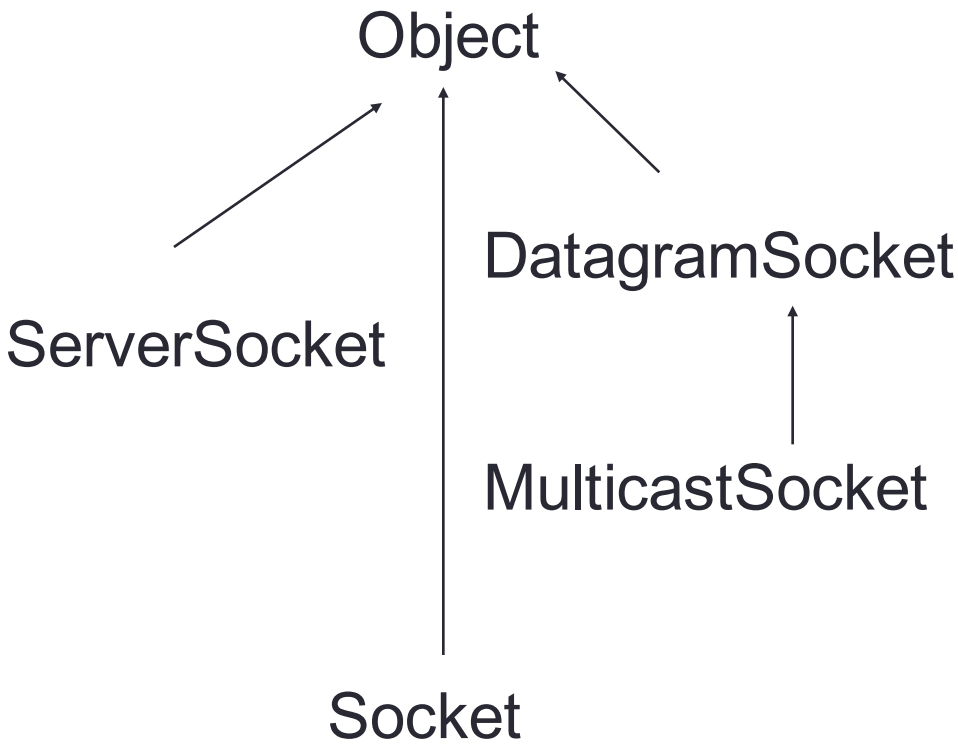
- fortement connectée
- synchrone
- type client-serveur

Communication par messages UDP

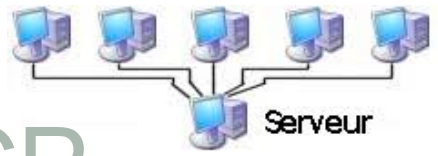
- en mode datagramme
- en mode déconnecté

Communication réseau par diffusion UDP

- En Java : sockets dans le package **java.net**

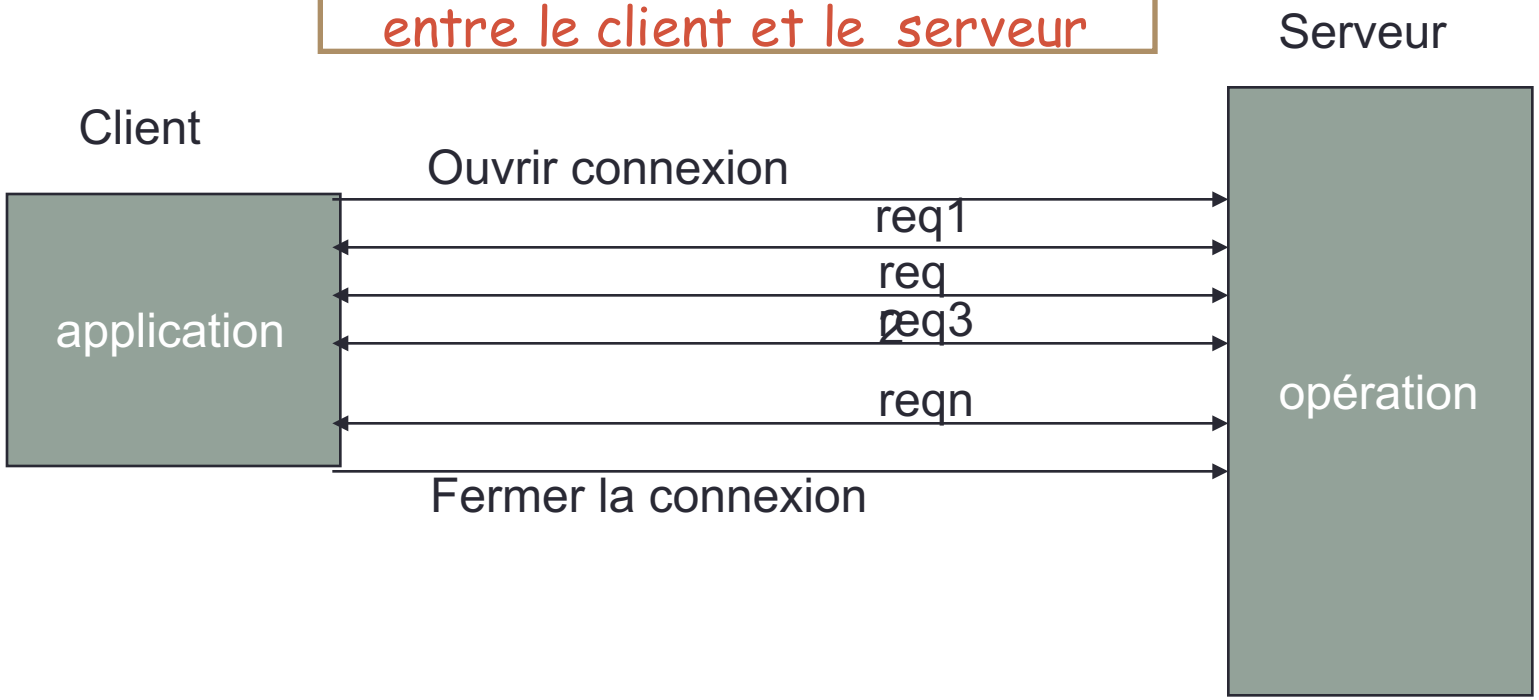


Flot de requêtes du client vers le serveur avec des Sockets TCP

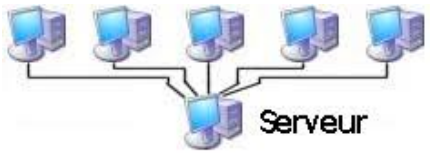


Point de vue application

TCP fournit un transfert fiable, conservant l'ordre de transfert des octets ("pipe") entre le client et le serveur



Scénario d'un serveur



Créer le socket de communication avec le client

Attente de données correspondant à la requête sur le flux d'entrée

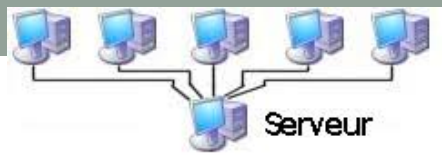
Réception et Analyse des données en entrée (unmarshalling)

Calcul

Construction de la réponse (marshalling

Fermer le socket de communication

Scénario d'un client



Créer le socket de connexion avec le serveur
Attendre que la connexion soit établie
Récupérer la socket de communication

Préparer la requête **(marshalling)**

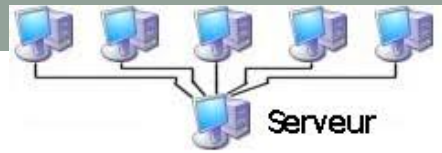
l'envoyer sur le flux de sortie

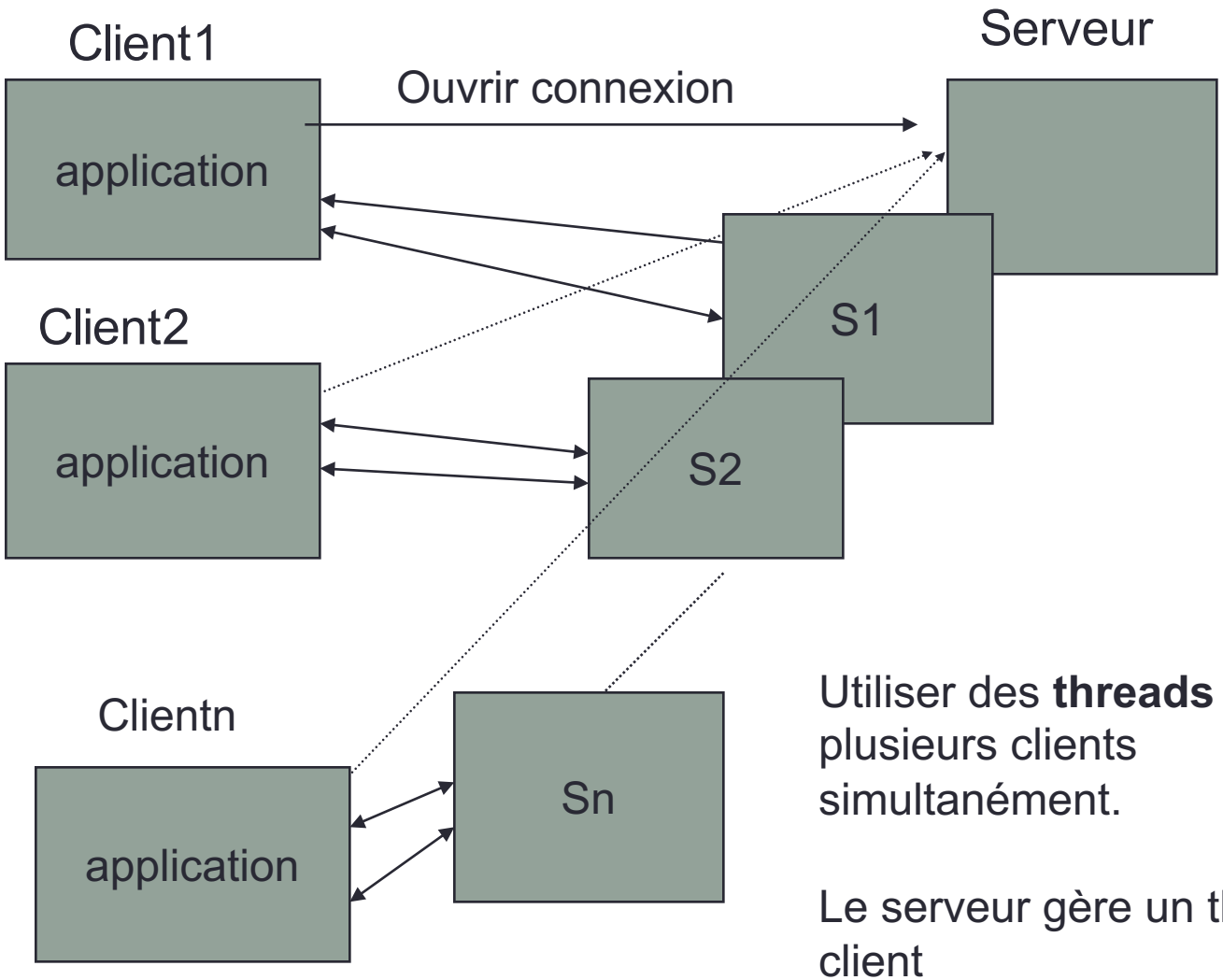
Attendre les données correspondant à la réponse sur le flux d' entrée

les lire, les **analyser (unmarshalling)**et les traiter

Fermer le socket

Plusieurs clients

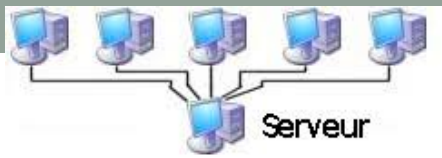


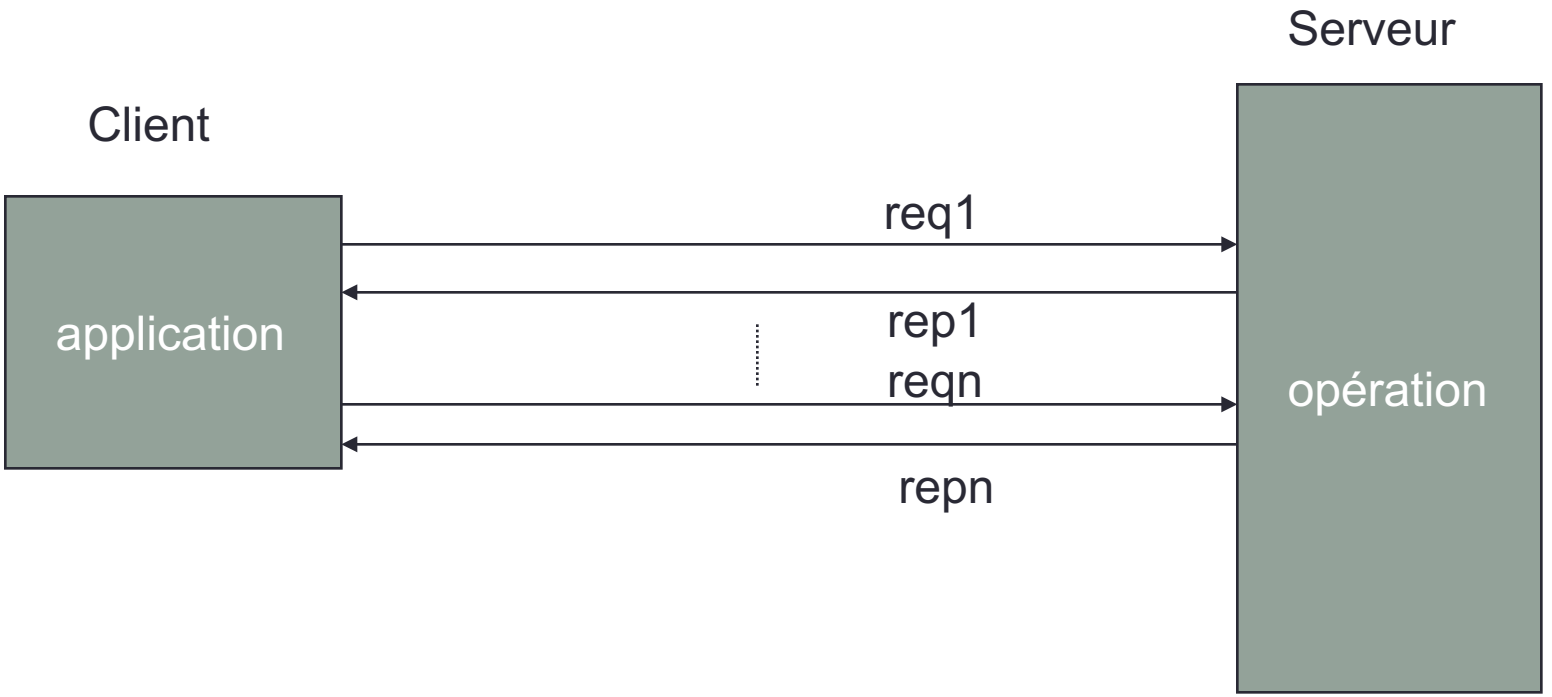


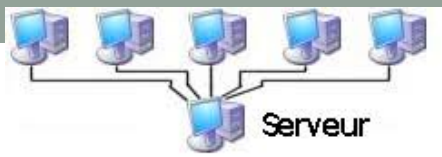
Utiliser des **threads** pour accepter plusieurs clients simultanément.

Le serveur gère un thread par client

Communication par message : Envoi de datagrammes







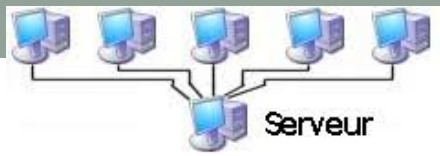
Scénario d'un client

Créer le socket d'entrée

Créer un paquet de sortie (marshalling)
Préparer et Envoyer une donnée

….

Fermer le socket d'entrée



Scénario d'un serveur

Créer le socket d'entrée

Création d'un paquet d'entrée
Attente de données en entrée (unmarshalling)
Réception et traitement des données en entrée

….

Fermer le socket d 'entrée

Marshalling et unmarshalling

l'information qui est lue doit être du même type et du même format que celle qui est écrite

Quel moyen connaissez vous en java ?

Interface Serializable

- Ne contient pas de méthode
- -> enregistrement et récupération de toutes les variables d'instances (pas de static)
  + informations sur sa classe (nom, version), type et nom des variables
    - 2 classes compatibles peuvent être utilisées

Objet récupéré = une copie de l'objet enregistré

Sérialisation-Desérialisation
Persistance et transfert réseau

- Enregistrer des objets dans un flux
  - Via la méthode writeObject()
    - Classe implémentant l'interface OutputObject
    - Exemple : la classe OutputObjectStream
    - Sérialisation d'un objet -> sérialisation de tous les objets contenus par cet objet
      - Un objet est sauvé qu'une fois : cache pour les listes circulaires
- Récupérer des objets dans un flux
  - Exemple : la classe InputObjectStream
  - Via la méthode readObject()
    - Classe implémentant l'interface InputObject

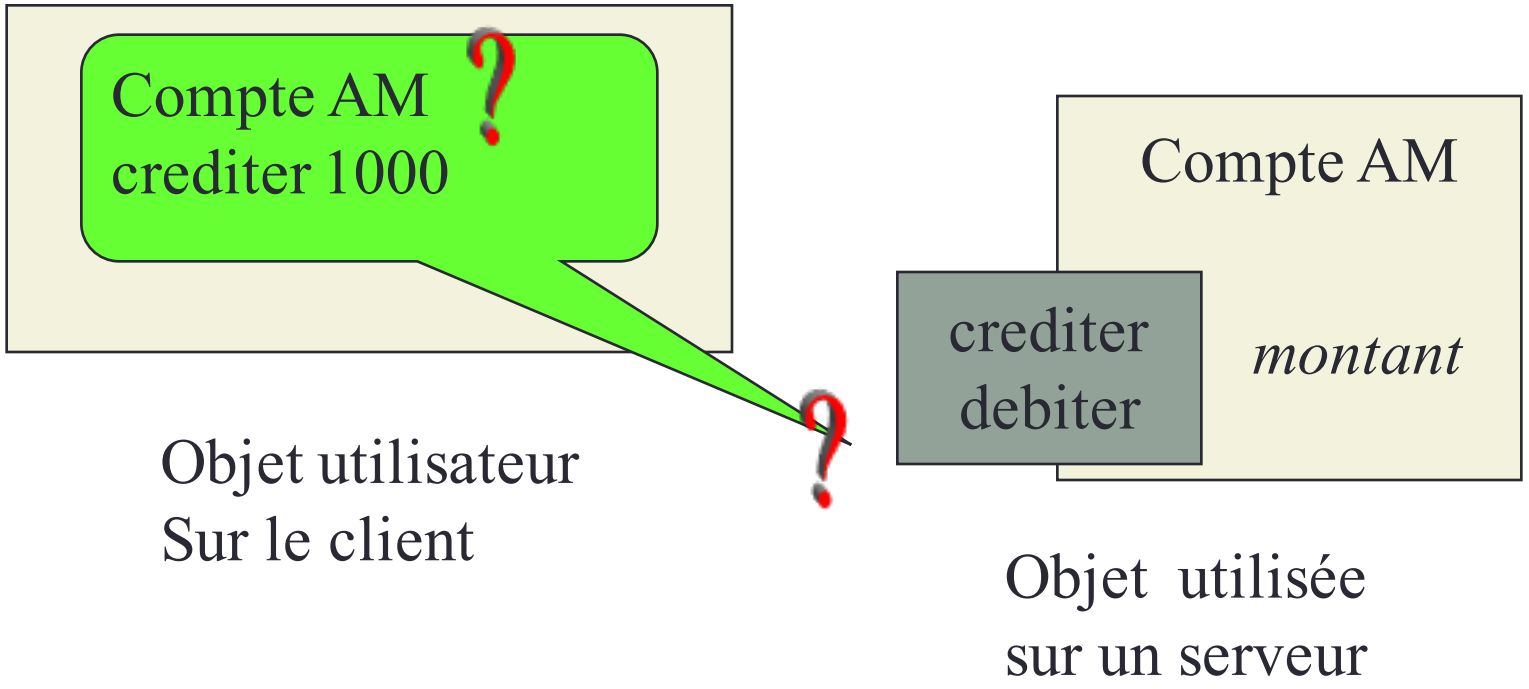
Objets distants
RPC : Remote Procedure Call

1/ Définition d'un contrat

2/ AutomatIsation
du Marshalling

3/ Couplage FORT

Pour les objets distants



crediter et débiter <-> services proposés par le serveur

Un serveur peut abriter plusieurs objets distants et plusieurs types d'objets distant

Que peut on générer à partir d'une description des services ?

Spécifications des données

Interface . Java

Générateurs

Fichiers générés

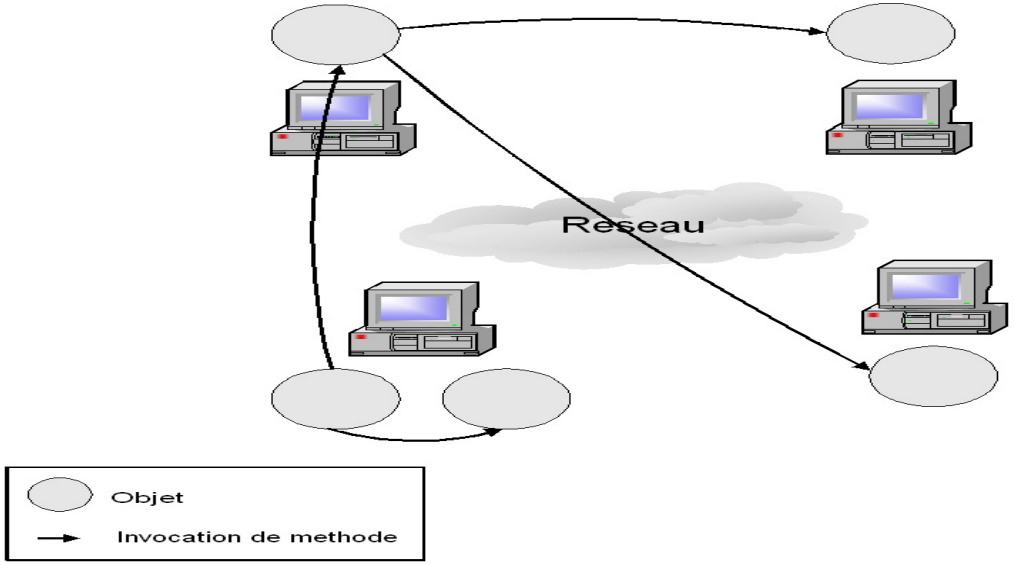
Sérialisation et désérialisation
Code du serveur
Appels des clients

Stubs
Skeletons
Proxy

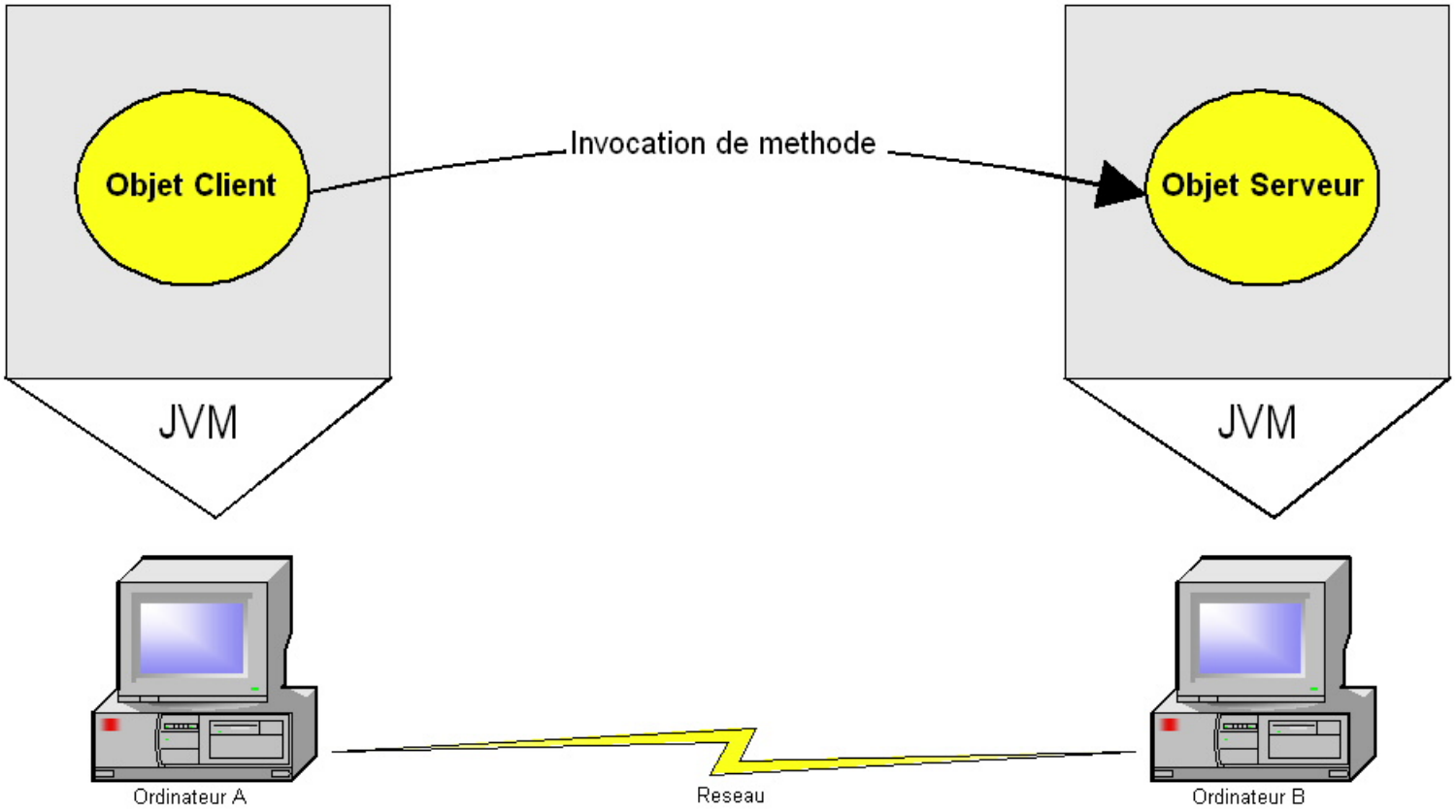
INVOCATION DE MÉTHODE À DISTANCE
EXEMPLE : JAVA
REMOTE METHOD INVOCATION

Invocation de méthodes distantes

- Mécanisme qui permet à des objets localisés sur des machines distantes de s'échanger des messages (invoquer des méthodes)



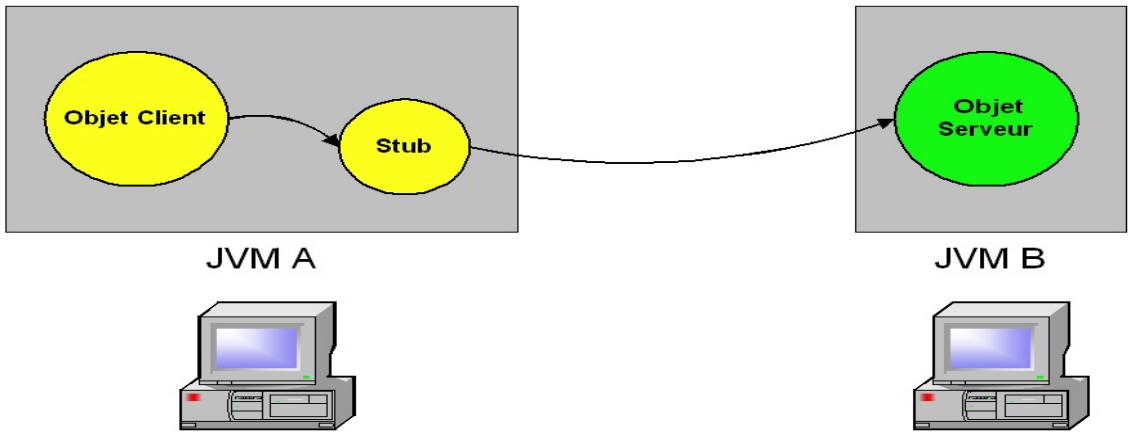
Invocation de méthodes distantes



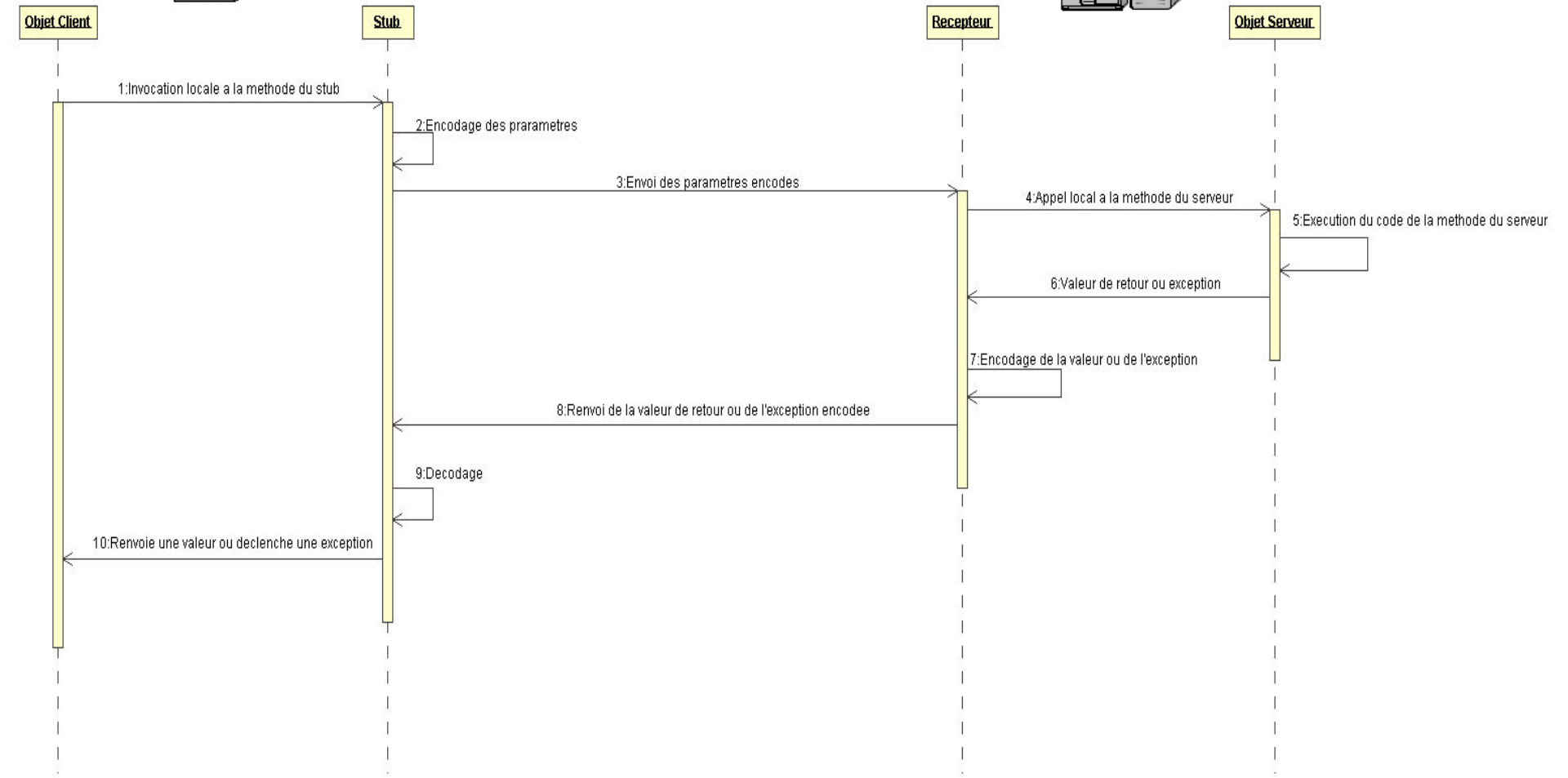
Java *Remote Method Invocation*

Stubs et encodage des paramètres

- le code d'un objet client invoque une méthode sur un objet distant
  - Utilisation d'un objet substitut dans la JVM du *client* : le **stub** (souche) de l 'objet serveur



Stubs et encodage des paramètres



Du côté du client

- Appel d'une méthode du stub (de façon entièrement transparente)

- le stub construit un bloc de données avec
  - identificateur de l'objet distant à utiliser
  - description de la méthode à appeler
  - paramètres encodés qui doivent être passés

- puis il envoie ce bloc de données au serveur...

Du côté du serveur

- un objet de réception (Squeleton) effectue les actions suivantes :
  - décode les paramètres encodés
  - situe l ’objet à appeler
  - invoque la méthode spécifiée
  - capture et encode la valeur de retour ou l ’exception renvoyée par l ’appel

Encodage des paramètres

- Encodage dans un bloc d ’octets afin d ’avoir une représentation indépendante de la machine
- Types primitifs et «basiques» (int/Integer...)
  - Encodés en respectant des règles établies
  - Big Endian pour les entiers...
- Objets ...

Générateur de stubs

- Les stubs gèrent la communication ainsi que l'encodage des paramètres
- Processus évidemment complexe...
- Entièrement automatique
  - Un outil permet de générer les `stubs` pour les OD

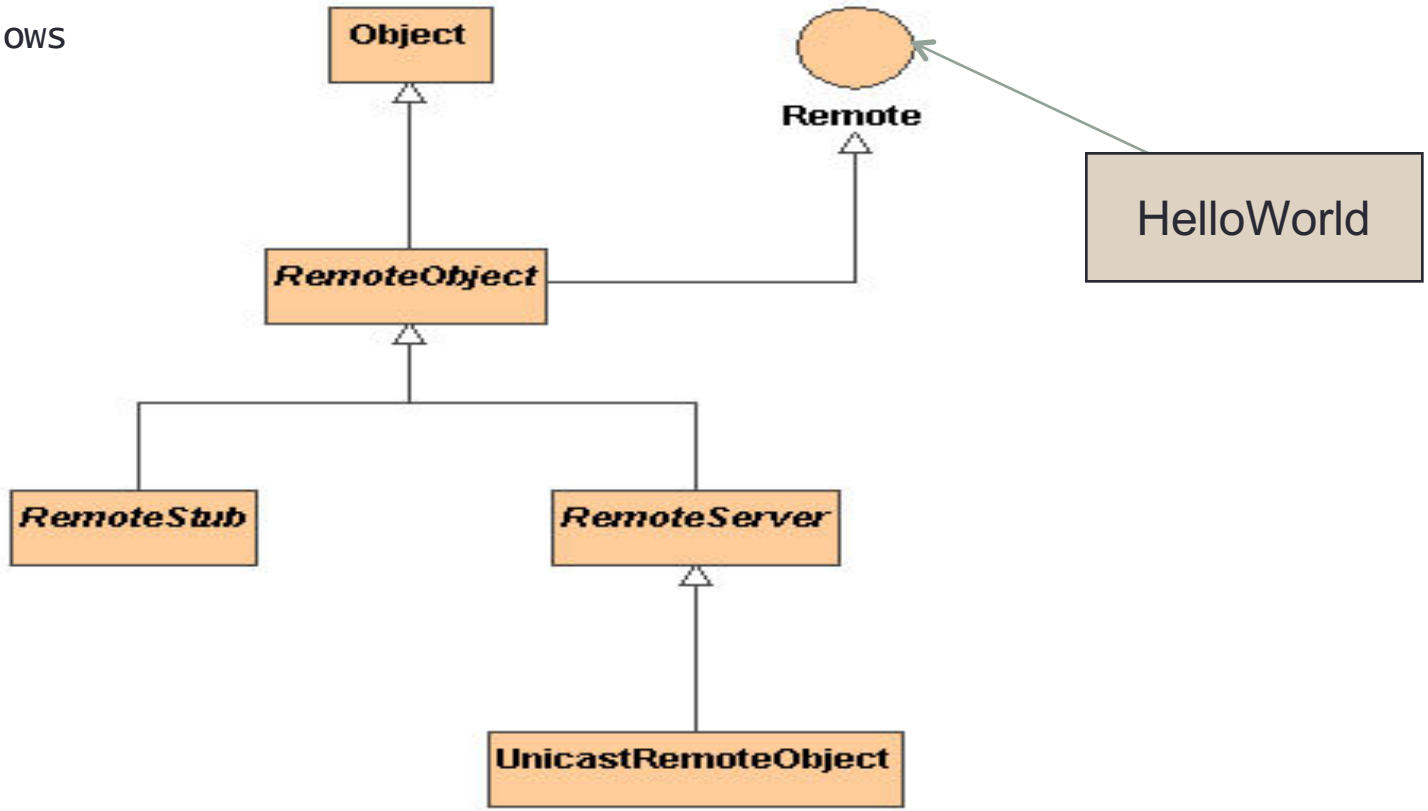
- En RMI La commande `rmic` du jdk rend transparent la gestion du réseau pour le programmeur
  - une référence sur un ODréférence son `stub` local

- syntaxe = un appel local
  - `objetDistant.methode()`

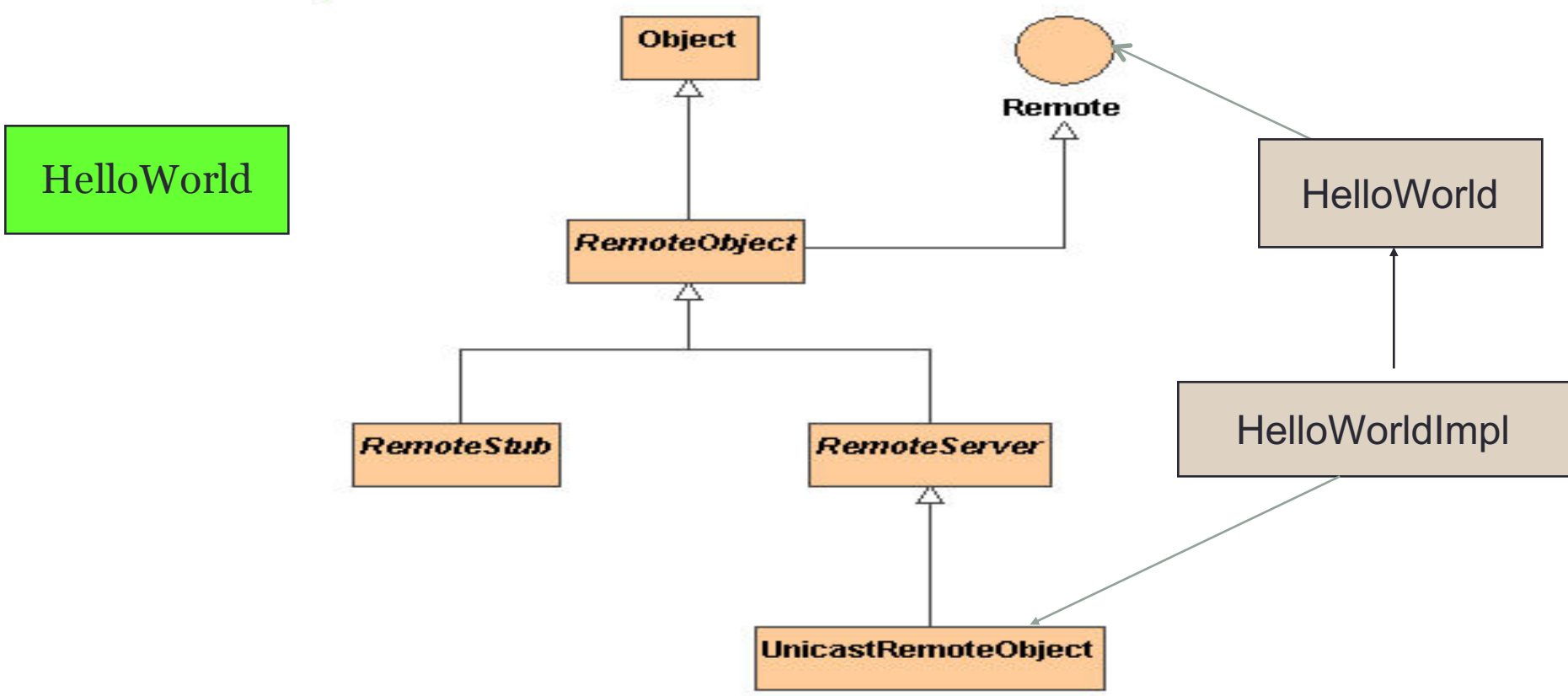
Rôle de l ’interface : contrat

```
import java.rmi.*;
interface HelloWorld extends
Remote {
        public String
sayHello()
                throws
RemoteException;
}
```

HelloWorld



Implémentation côté serveur



Interface = protocole d ’application contrat entre un serveur et ses clients entre l’objet appelé et l’objet appelant

- L ’interface `HelloWorld`

```
import java.rmi.*;
interface HelloWorld extends Remote {
 public String sayHello()
     throws RemoteException;
}
```

Quels paramètres ?

- On sera souvent amenés a passer des paramètres aux méthodes distantes...
- Les méthodes distantes peuvent retourner une valeur ou lever une exception...
- On a deux types de paramètres
  - Les objets locaux
  - Les objets distants

Passage de paramètres: ATTENTION

- Certains objets sont copiés (les objets locaux), d'autres non (les objets distants)
- Les objets distants, non copiés, résident sur le serveur
- Les objets locaux passés en paramètre doivent être sérialisables afin d'être copiés, et ils doivent être indépendants de la plate-forme
  - Les objets qui ne sont pas sérialisables lèveront des exceptions
  - Attention aux objets sérialisables qui utilisent des ressources locales !!!

Référence sur un objet

- On sait implanter un serveur d ’un côté, et appeler ses méthodes de l ’autre

MAIS

- Comment obtient-on une référence vers un stub de notre objet serveur ???

Localisation des objets de serveur

- On pourrait appeler une méthode sur un autre objet serveur qui renvoie une référence sur le stub...
- Mais quoi qu ’il en soit, le premier objet doit lui aussi être localisé (La poule et l ’œuf) !!!
- On a alors recours a un Service de Nommage

Les Services de Nommage

- Obtention d'une première référence sur un objet distant : « bootstrap » à l'aide d'un Service de Nommage ou Annuaire
- Enregistrement des références sur des objets du serveur afin que des clients les récupèrer
- Association de la référence de l'objet à une clé unique
- Recherche par la clé, d'un objet : le service de nommage renvoie la référence distante (le stub) de l'objet enregistré pour cette clé

Enregistrement d ’une référence

- L ’objet serveur HelloWorld (coté serveur bien entendu…)
  - On a créé l'objet serveur et on a une variable qui le référence

```
HelloWorld hello = new HelloWorldImpl();
```

  - On va enregistrer l'objet dans le RMIRegistry

```
Naming.rebind("HelloWorld",hello);
```

  - L'objet est désormais accessible par les clients

Obtention d'une référence coté client

- sur l'objet serveur HelloWorld
  - On déclare une variable de type HelloWorld et on effectue une recherche dans l'annuaire

```
HelloWorld hello =
(HelloWorld)Naming.lookup("rmi://www.helloworldserver.
com/HelloWorld");
```

  - On indique quelle est l'adresse de la machine sur laquelle s'exécute le RMIRegistry ainsi que la clé
  - La valeur retournée doit être transtypée (castée) vers son type réel

Processus de développement

1) Définir le contrat : la liste des services
   définir une interface Java pour un OD
2) Implémenter les services dans le serveur
   créer et compiler une classe implémentant cette interface
3) Créer le serveur (enregistrer le ou es objets dans le serveur de noms, etc)
   créer et compiler une application serveur RMI
4) Générer le marshalling – unmarshalling
   créer les classes Stub (**`rmic`**)
5) Lancer le serveur (et le serveur de noms)
   démarrer **`rmiregistry`** et lancer l'application serveur RMI
6) Les clients peuvent maintennat être créés
   créer, compiler et lancer un programme client accédant à des OD du serveur

DEPLOIEMENT

1/ Définition d'un contrat

2/ AutomatIsation
du Marshalling

3/ Couplage FORT

Que doit connaître le client ?

- Le contrat
- Le nom et le service de nommage
- Lorsqu'un objet serveur est passé à un programme, soit comme paramètre soit comme valeur de retour, ce programme doit être capable de travailler avec le stub associé

- Le programme client doit connaître la **classe** du stub

Que doit connaître le client ?

- les classes des paramètres, des valeurs de retour et des exceptions doivent aussi être connues...
  - Une méthode distante est déclarée avec un type de valeur de retour...
  - ...mais il se peut que l ’objet réellement renvoyé soit une sous-classe du type déclaré

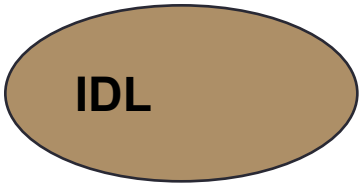
Déploiement dynamique vs déploiement statique

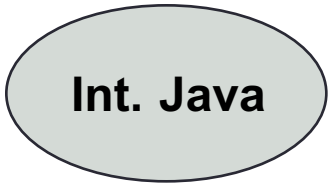
- Pour ne plus déployer les classes du serveur chez le client
  - Utilisation des chargeurs de classes qui téléchargent des classes depuis une URL
  - Utilisation d ’un serveur Web qui fournit les classes
- Ce que ça change
  - Bien entendu, les classes et interfaces de l ’ objet distant ne changent pas
  - Le code du serveur ne change pas
  - **le client et la façon de le démarrer sont modifiés**
  - **Et lancer un serveur Web pour nos classes**

DES OBJETS DISTRIBUÉS AUX COMPOSANTS

Générateurs

Spécifications des données



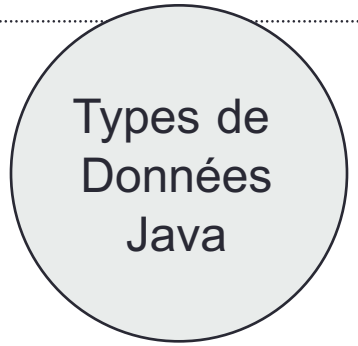


Générateurs

Fichiers générés

Stubs Skeletons Proxy

(mise en œuvre de la sérialisation et désérialisation…)



Protocoles d'application et Langages de spécifications

- Spécifications des types de données qui transitent sur le réseau

```
Protocole := CHOICE {
                requete [0] REQUETE,
                reponse [1] REPONSE }
```

ASN.1 et norme ISO

```
Programme reqrep {
        version {
            REPONSE rerep(REQUETE) = 1
        }= 1
} = 10000
```

XDR et RPC de SUN

Générateurs de Stubs

Spécifications des données

ASN1

XDR

Générateurs

RPCGEN MAVROS

Fichiers générés

Types de données C Lisp Java

Librairie marshalling et unmarshalling squelettes du client et du serveur

Types de données C

Attention au vocabulaire

- Coté client :
  - stub en CORBA
  - proxy en OLE
  - stub/proxy en Java
- Côté Serveur :
  - stub en OLE
  - skeleton en CORBA
  - implémentation d'une interface en RMI

Services et Objets Distribués

Middleware CORBA

- Services normalisés
- Seulement certains sont implémentés
- Naming, Trading, Event

Middleware RMI

Des services en programmant avec Java
Sécurité, Threads, Événements

Url et Web

Non intégrés à RMI

Un composant, c'est quoi ?

Une brique permettant la programmation par assemblage

Une solution facilitant le déploiement, la gestion du cycle de vie des applications logicielles

Une meilleure intégration des services

EJB – CORBA 3: Apports

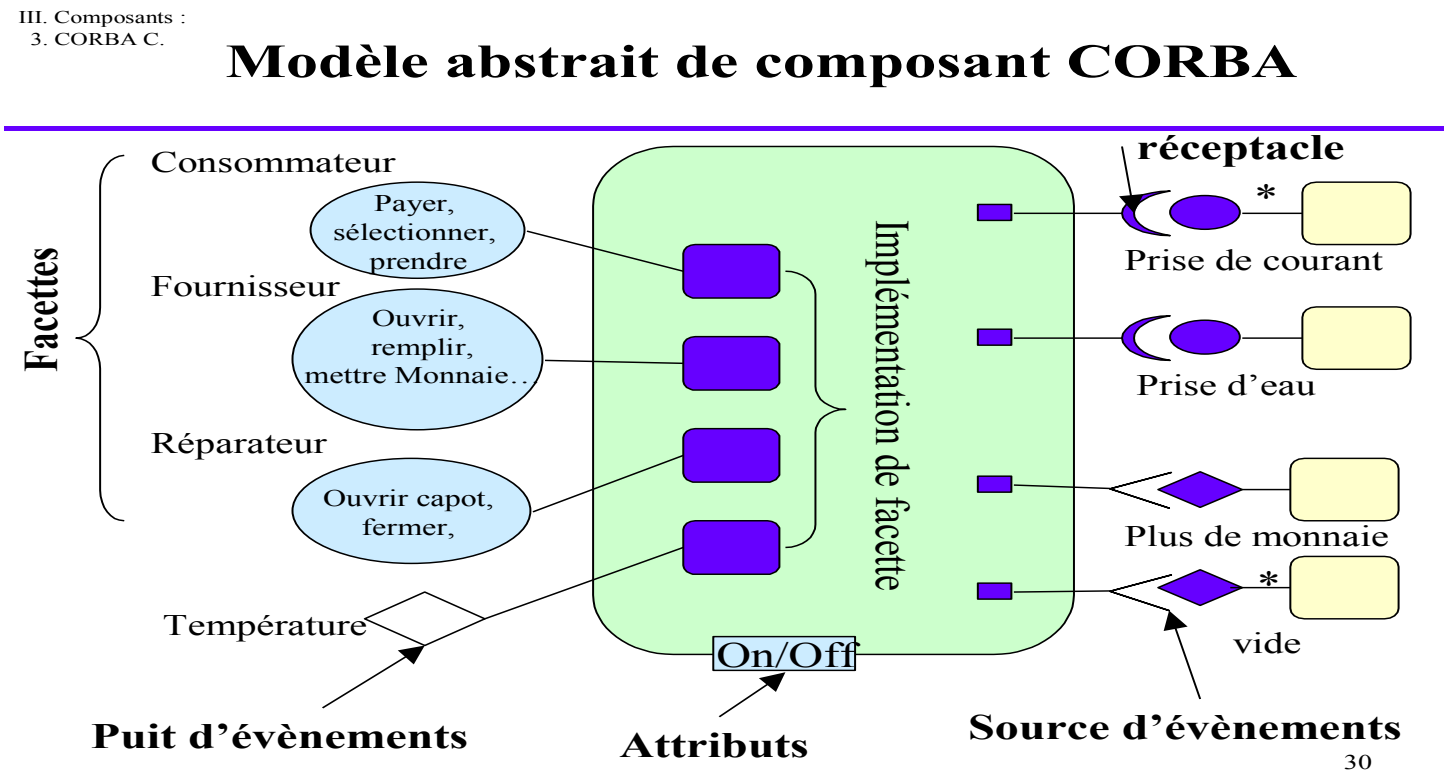
Interfaces entrées et sorties : ports requis et offerts

Conteneur : intégration des propriétés non fonctionnelles (sécurité, persistance, transaction)

Home : fabrique et navigation

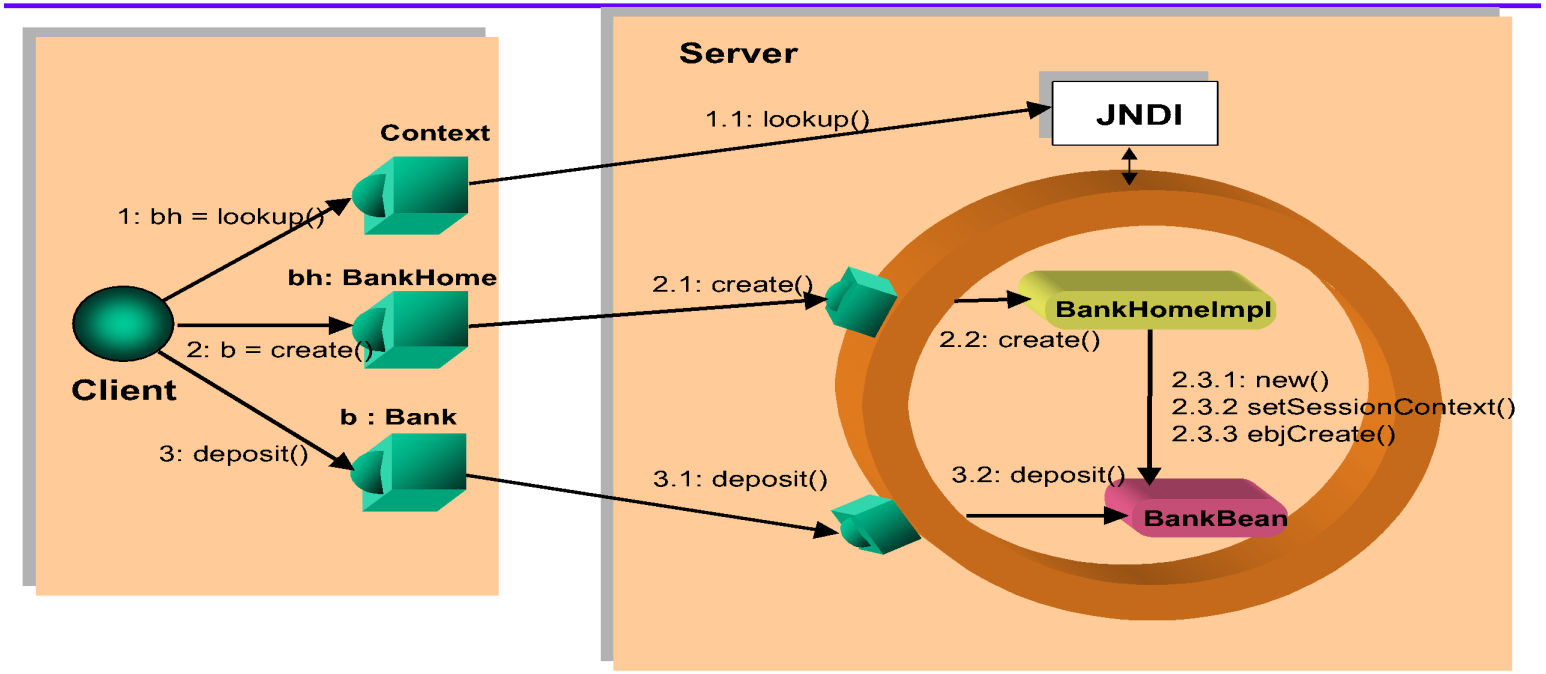
Communication par envoi de message et notification (événement)

Exemple


Modèle abstrait de composant CORBA
Facettes
Consommateur
Payer, sélectionner, prendre
Fournisseur
Ouvrir, remplir, mettre Monnaie..
Réparateur
Ouvrir capot, fermer,
Température
Implémentation de facette
On/Off
réceptacle
*
Prise de courant
Prise d'eau
Plus de monnaie
*
vide
Puit d'évènements
Attributs
Source d'évènements

Exemple

Création et utilisation de Bank



Points communs avec les middlewares objets

Langages de description : CIDL ou Interfaces Java

Infrastructure : ORB / RMI

Marshalling : repose sur Corba / RMI

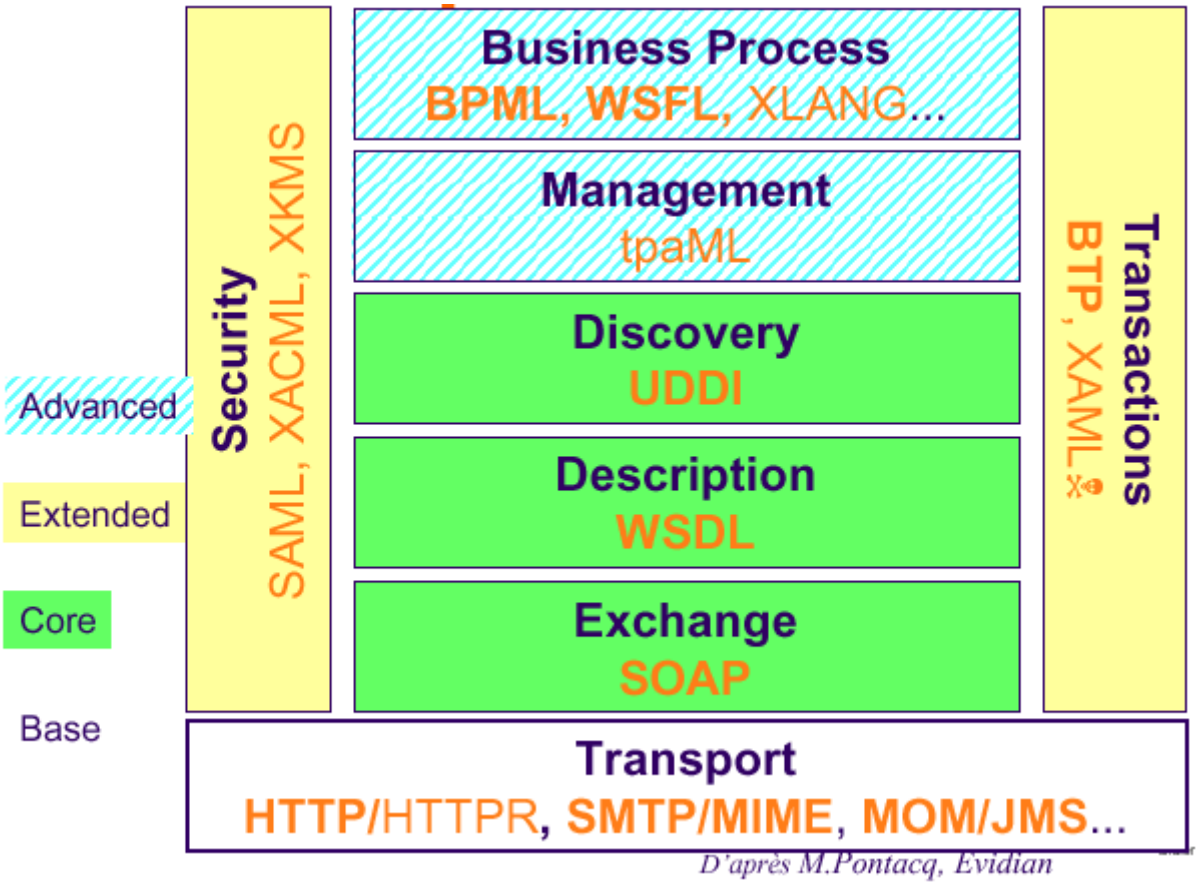
Nommage : Home ++

Interface : Héritage + Composition

Un Service Web, c'est quoi ?

- Une « unité logique applicative »
- Une «librairie» fournissant des données et des services à d'autres applications.
- Un objet métier déployé sur le web (vision objet)
- Un « module » ou « composant » ?
- Une sorte d'objet… plutôt qu'un composant

Architecture globale



D'après M.Pontacq, Evidian

Points communs avec les middlewares objets

Un langage de description : WSDL

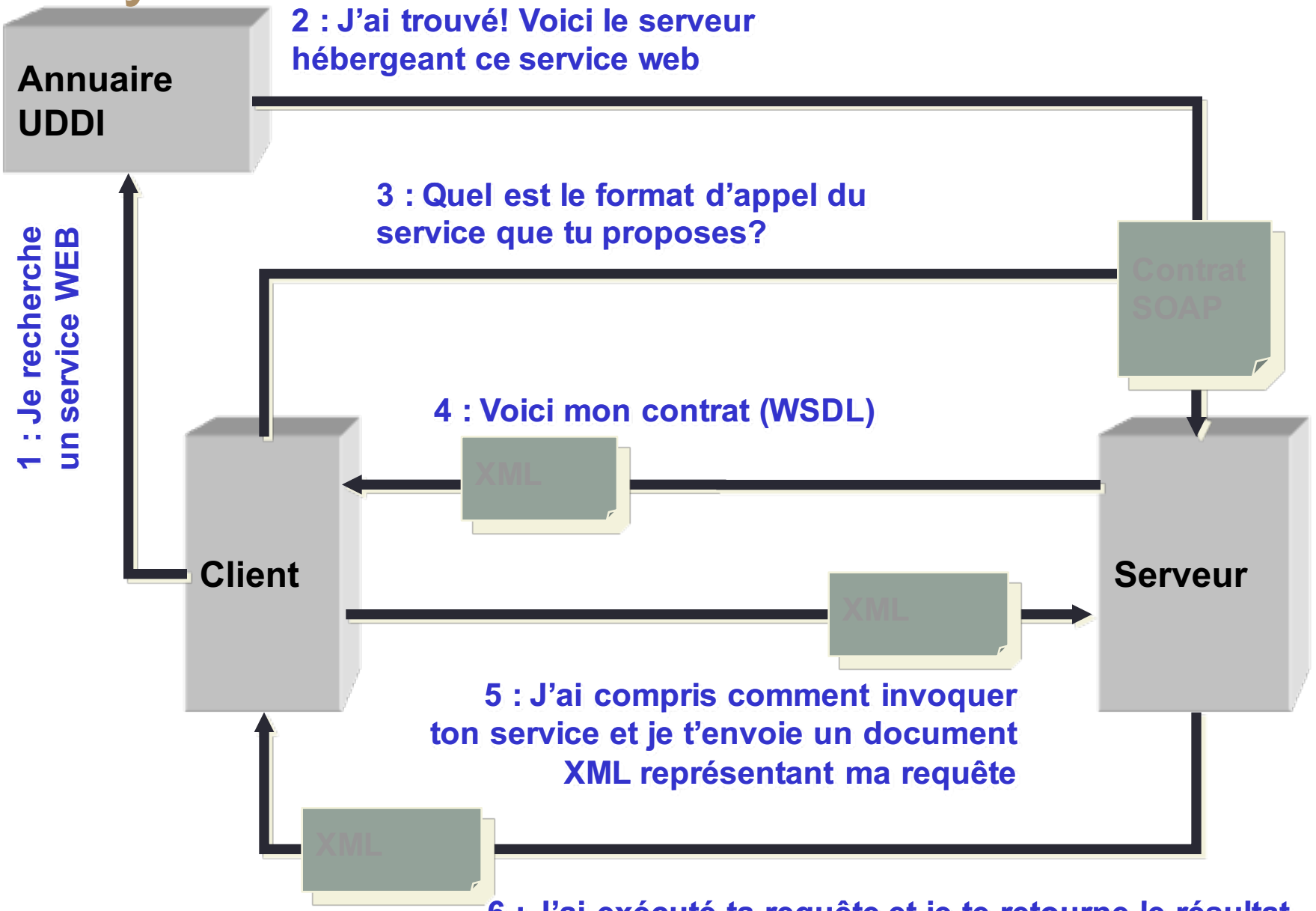
Une infrastructure : Le Web et http

Une communication par envoi de messages : SOAP

Du marshalling : XML

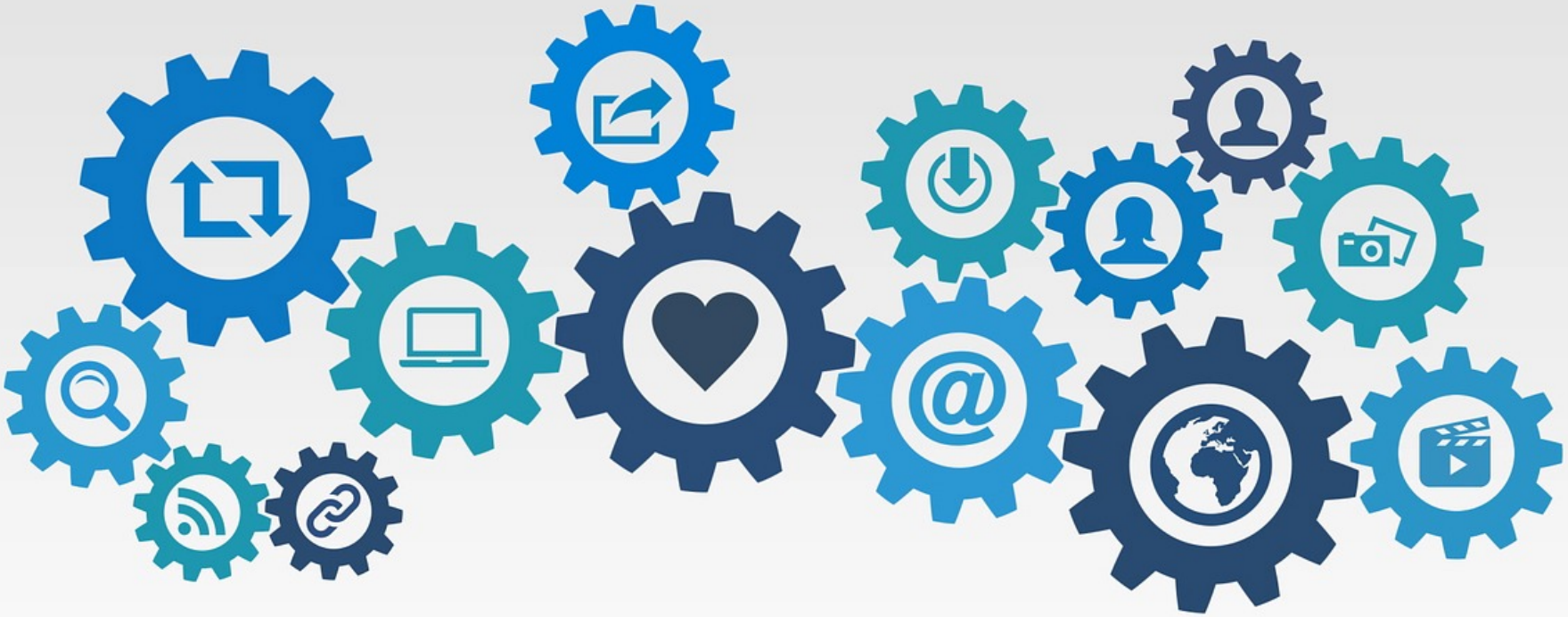
Un service de nommage « dynamique » : UDDI

Cycle de vie d'utilisation



Environnements intégrés .net

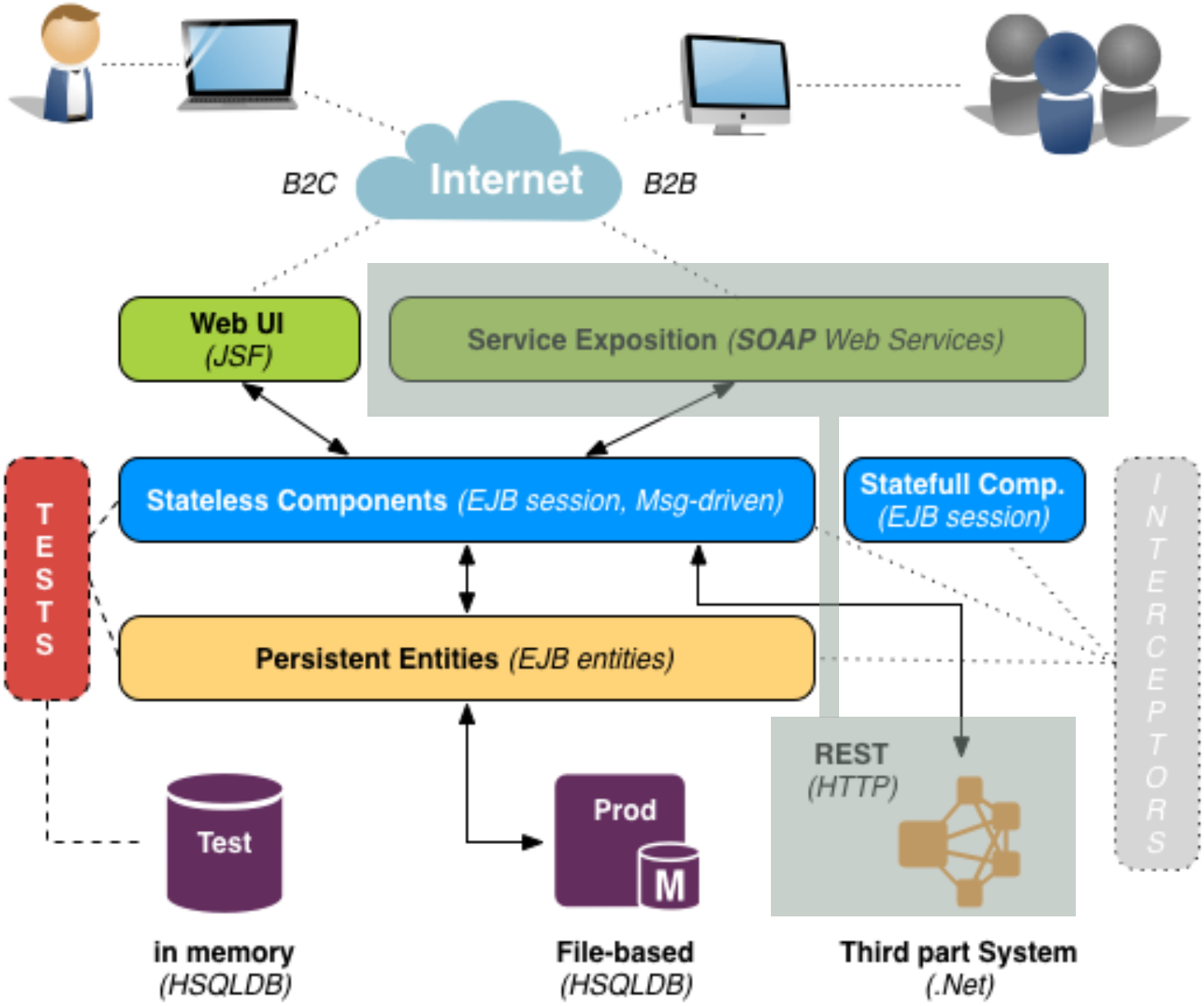
- Toute la mécanique est cachée
- On peut se concentrer sur la conception
- Aide à l'assemblage ?



Interoperability with Web Services

AM Dery
Fortement inspirée des cours de
S Mosser

pictures: sxc.hu


B2C
Internet
B2B
Web UI
(JSF)
Service Exposition (SOAP Web Services)
TESTS
Stateless Components (EJB session, Msg-driven)
Statefull Comp.
(EJB session)
INTERCEPTORS
Persistent Entities (EJB entities)
REST
(HTTP)
Test
Prod
M
in memory
(HSQLDB)
File-based
(HSQLDB)
Third part System
(.Net)

Interoperability ?

Heterogeneous System

REST vs SOAP

SOAP ➔
exposes **procedures** (*aka Remote Procedure Call*, RPC)

REST ➔
exposes **resources** (*i.e.*, nouns instead of verbs).