UNIVERSITÉ DE STRASBOURG

cnrs

Systèmes Distribués

Introduction, RPC/RMI/SOAP

Pascal Mérindol (CM)
Benoit Sonntag (TD + TP)

merindol@unistra.fr
http://www-r2.u-strasbg.fr/~merindol/

Ce cours est construit sur la base de plusieurs supports pédagogiques, en particulier les cours de Pierre Gançarski et Guillaume Latu. L'usage de ce support ne peut être qu'académique.

Plan & Organisation

- **Introduction aux Systèmes Distribués**
- **La pratique : RPC, RMI, SOAP, etc (TP & Projet)**
- **La «théorie» (en CM/TD) :**
  - Horloges & Diffusion/Partage
  - Exclusion Mutuelle & Blocages
  - Ordonnancement
- **Divers :** tolérance aux pannes, consensus, intro. sécurité, etc.

- **Benoit Sonntag TD/TP; Pascal Mérindol CM**
  - 50% CC2, 25% CC1, 25% Projet.
  - 9*2h TP, 6*2h TD, 10*2h CM.

Liens utiles (& refs du cours)

- **http://robinet.u-strasbg.fr/enseignement/sd/sd + moodle**
- **http://iut-info.unistra.fr/~gancars/enseignement/index_enseignement.htm**
- **http://www.pps.jussieu.fr/~rifflet/enseignements/AlgoProgSysRep/**
- **http://www.informatics.sussex.ac.uk/courses/dist-sys/node1.html**
- **http://code.google.com/intl/fr-FR/edu/parallel/**

- G. F. Coulouris, J. Dollimore, T. Kindberg. ``Distributed Systems -- Concepts and Design". Ed. Addison-Wesley, 4th Edition. 2005.

Définition(s)

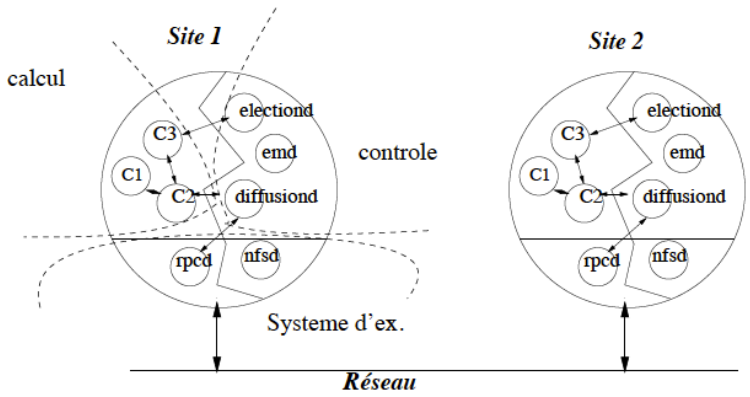
- Assurer la coopération d'un ensemble de processus dans un environnement distribué
- Services rendus aux applications réparties (multi-sites)
  - communications inter-processus
  - partage des ressources physiques sous jacentes
- Mécanismes nécessaires :
  - transfert, partage et reconnaissance d'informations,
  - contrôle de cohérence, diffusion, synchronisation, ordonnancement,
  - contrôle global du système, élection, exclusion mutuelle, reprise sur panne,
  - transactions sécurisées, administration, etc ...
- Objet du cours : les systèmes faiblement couplés

Principes de base

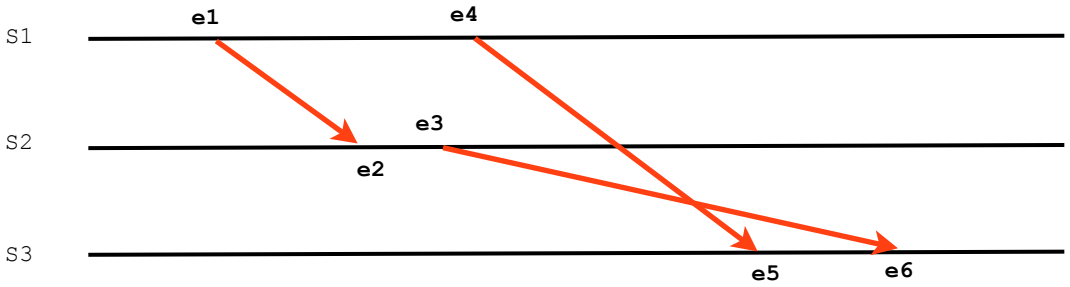
- Pas d'état global
  - les processus sont «égo-centrés» : pas de connaissance ni d'ordre strict global !
- Les données sont distribuées
  - duplication : cohérence ?
  - partitionnement multi-site : où retrouver et comment désigner l'info ?
- Pas de contrôle global
  - pas de hiérarchie type processus maitre...(robustesse du système)
  - ...ou élection pour des taches spécifiques (reprise sur panne)

- Comment «s'en sortir» dans un tel environnement ?

Briques de base



- Protocole
  - Comportements et règles inter-processus
- Processus / sites
  - dit de calcul (liés à l'application)
  - dit de contrôle (inter-agissent avec le système d'exploitation)
- Liaisons logiques : *le graphe de communication*
  - quelle structure ? anneau, étoile, arbre, ...
  - quel quantité de messages ? combien de pertes tolérées ?
  - quel comportement ? FIFO..?

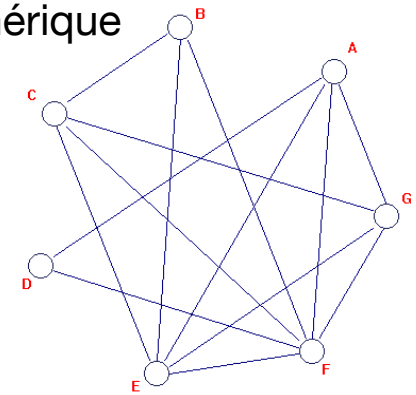


- FIFO ne veut pas dire que e4 est antérieur à e3 vu de S3

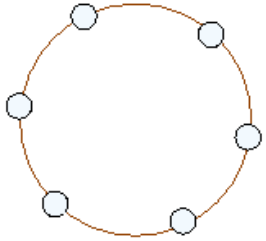
Aspects réseau

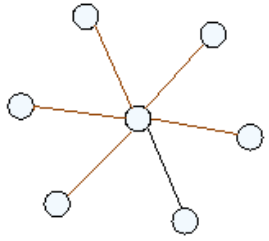
Graphe & topologies

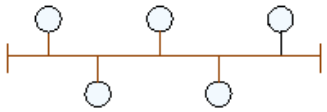
modèle générique

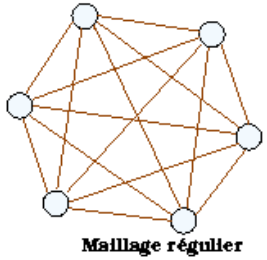


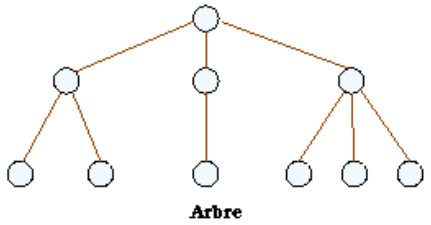
modèles spécifiques











Robustesse aux pertes

Soit *p* la probabilité de perte d'un msg et *n* le nb de msg nécessaire à un protocole.

La probabilité *P(X=k)* de perte de *k* msg est alors de $C_n^k p^k (1-p)^{n-k}$

La probabilité que le protocole aboutisse est de $P(X=0) = (1-p)^n$

ex : n=1000 messages et $p=10^{-3}$ alors ~2/3 d'aboutir du premier coup seulement...

Combien de tentatives *t* pour assurer au moins une probabilité *x* de réussite (*1-x* : échec) ?

La probabilité d'échec après *t* tentatives est de $(1-P(X=0))^t$ donc $(1-P(X=0))^t < 1-x$

ex : pour x=0.99 (et situation identique à l'ex. précédent) on a t=10

Les protocoles

- Calcul diffusant

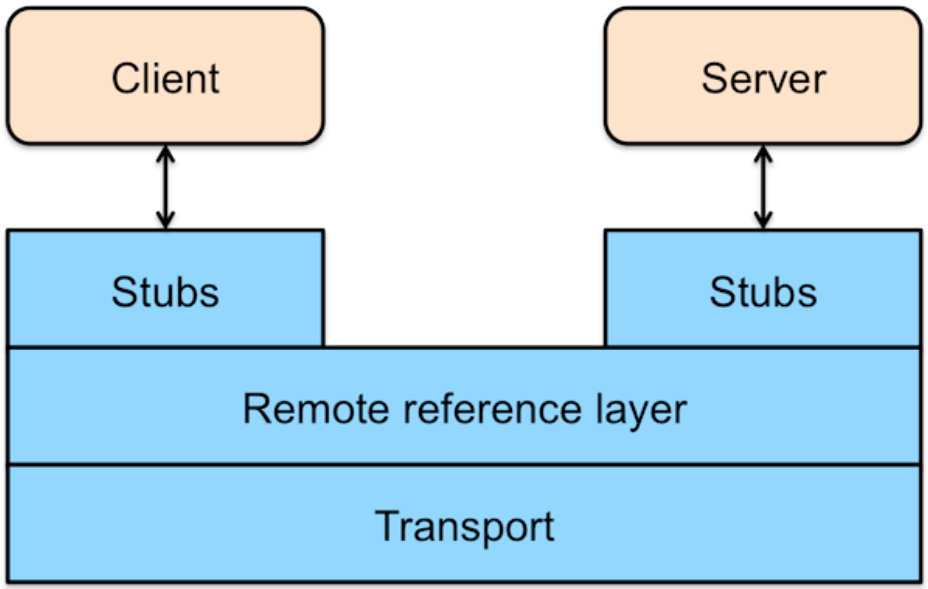
le droit d'émettre initialement détenu par une racine est diffusé, svnt un arbre diffusant :

messages diffusés de père en fils et calcul de «bas en haut»

- Jeton circulant
  - privilège circulant autour d'un anneau logique ou physique
- Estampillage
  - horloge logique
  - horloge vectorielle & causalité

Mais avant tout...la pratique !

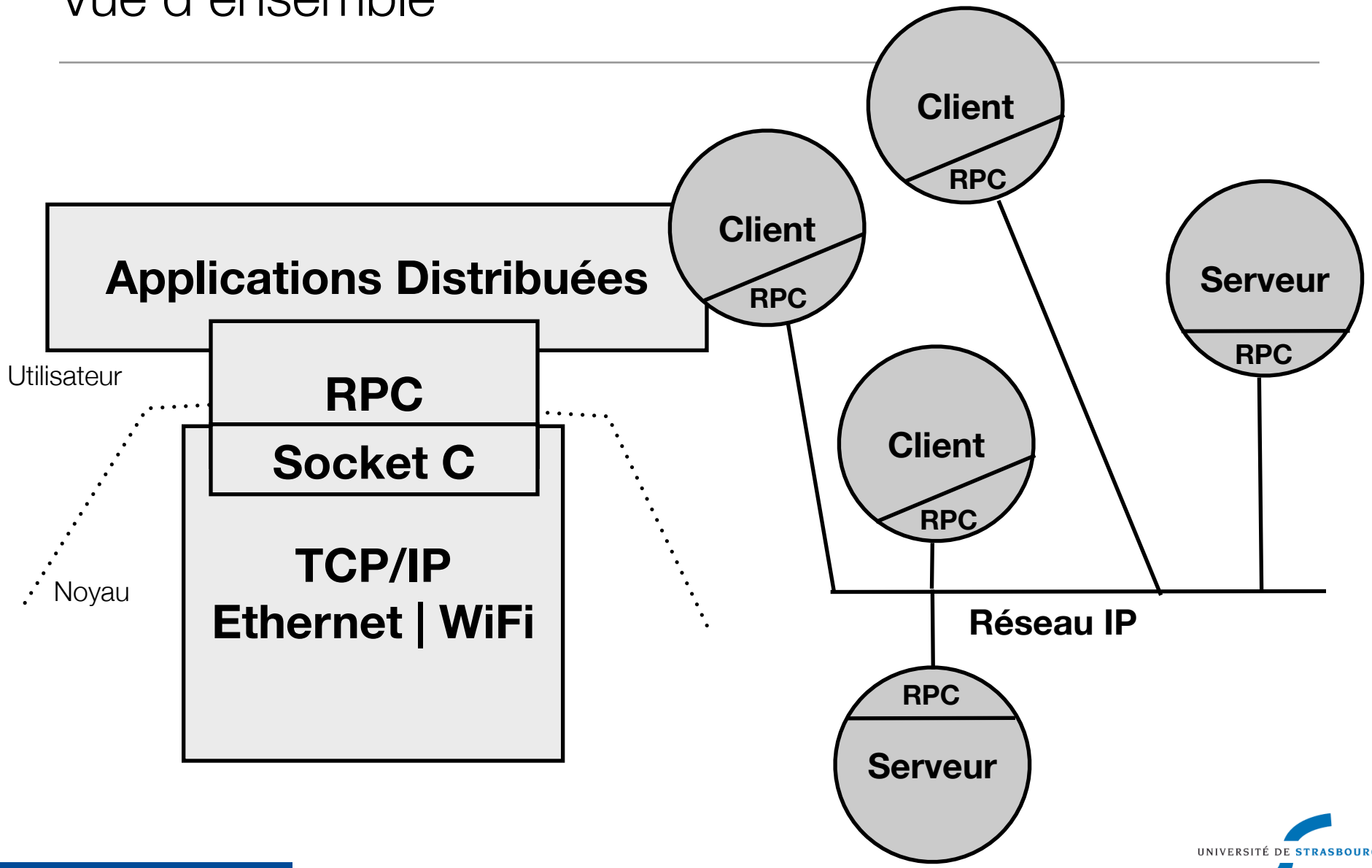
RPC, RMI, SOAP, REST, CORBA, etc



RPC : Remote Procedure Call

- **Middleware pour facilement définir des applis distribuées (SUN)**
- **Modèle transactionelle de type client / serveur (/ lib socket en C)**
  - offrir des services distants via du code C à plus haut niveau que les socks
  - notion de protocole ou automate à états
- **Intergiciel ?**
  - Applis / NIS | NFS | etc [/ portmap] / **RPC** / Socket / TCP/IP / Ethernet | WiFi | etc
- **Encodage XRP : formatage réseau (SUN)**
- **Passage par pointeurs généralisé**
  - => un seul paramètre
    - -> utilisation de structures

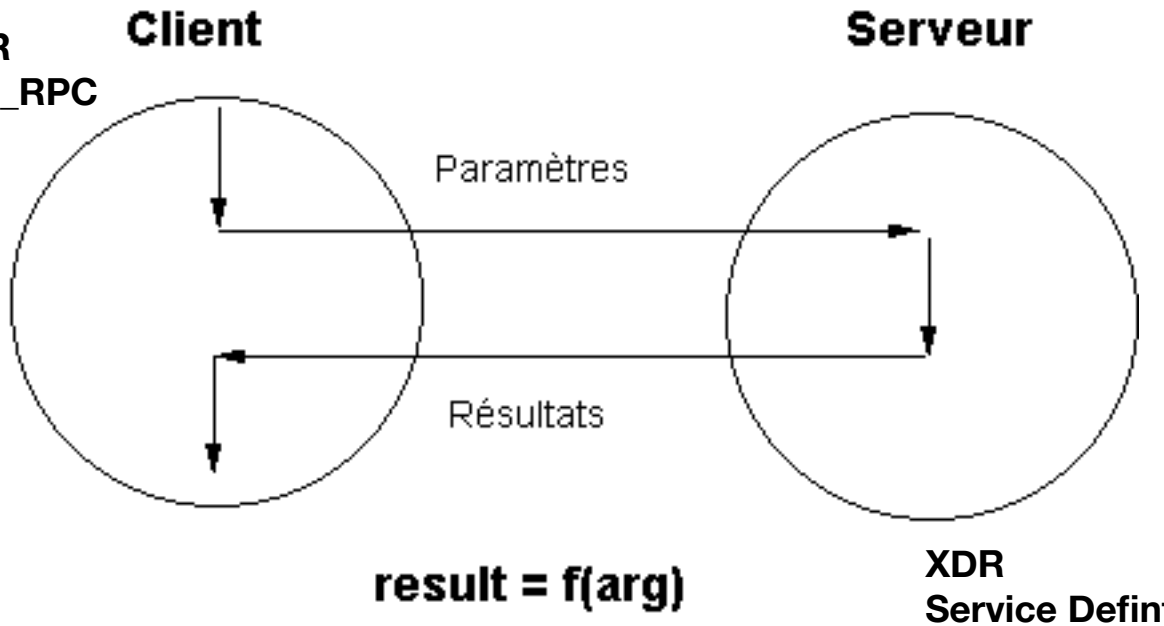
Vue d'ensemble



Le fonctionnement globale

```
int *f(int *a) {
  static int q;
  q=*a * *a;
  return &q;
}

main () {
  int b = 1;
  printf("res : %d\n",*f(&b));
}
```



Serveur : enregistrer et lancer son service

- `int registerrpc (u_long no_pg, u_long no_vers, u_long no_proc, void * (* fonction) ( ), xdrproc_t xdr_param, xdrproc_t xdr_result);`

no_pg: numéro du programme où enregistrer la fonction

no_vers: numéro de version

no_proc: numéro de procédure à donner à la fonction

fonction: pointeur sur la fonction à enregistrer

xdr_param: fonction d'encodage/décodage des paramètres

xdr_result: fonction d'encodage/décodage du résultat

+ `void svc_run();`

Client : appel bloquant

```
int callrpc (char * machine, u_long no_prog, u_long no_vers,
u_long no_proc, xdrproc_t xdr_param, void *param, xdrproc_t
xdr_result, void *result);
```

machine: nom de la machine avec la fonction à exécuter

<no_prog, no_vers, no_proc> : triplet identifiant la fonction à appeler

xdr_param: filtre XDR pour les paramètres

param: pointeur sur le paramètre à passer à la procédure

xdr_result: filtre XDR pour le résultat

result: pointeur sur la zone avec le résultat du call_RPC

Sérialisation

Cette opération consiste à ajouter (ou extraire) des informations dans un flot XDR réalisant ainsi le codage (ou le décodage).

Le type **`xdrproc_t`** désigne de manière générique le pointeur sur les fonctions XDR pour décoder ou encoder les données.

```
typedef bool_t (*xdrproc_t) (XDR*, void*)
```

Exemples de fonctions standard :

```
bool_t xdr_char(XDR *, char *)
bool_t xdr_int(XDR *, int *)
bool_t xdr_float(XDR *, float *)
```

Sinon à vous de jouer (composition de type élémentaire) :

```
bool_t mafonction (XDR *, (mastructure *) pointeur)
```

XDR : eXternal Data Representation

Types de base (décodage : **allocation** taille fixe) :

```
xdr_bool, xdr_char, xdr_int, xdr_long, xdr_short
  bool_t xdr_type (xdr_handle, pobj)
  XDR *xdr_handle;
  type *pobj;
```

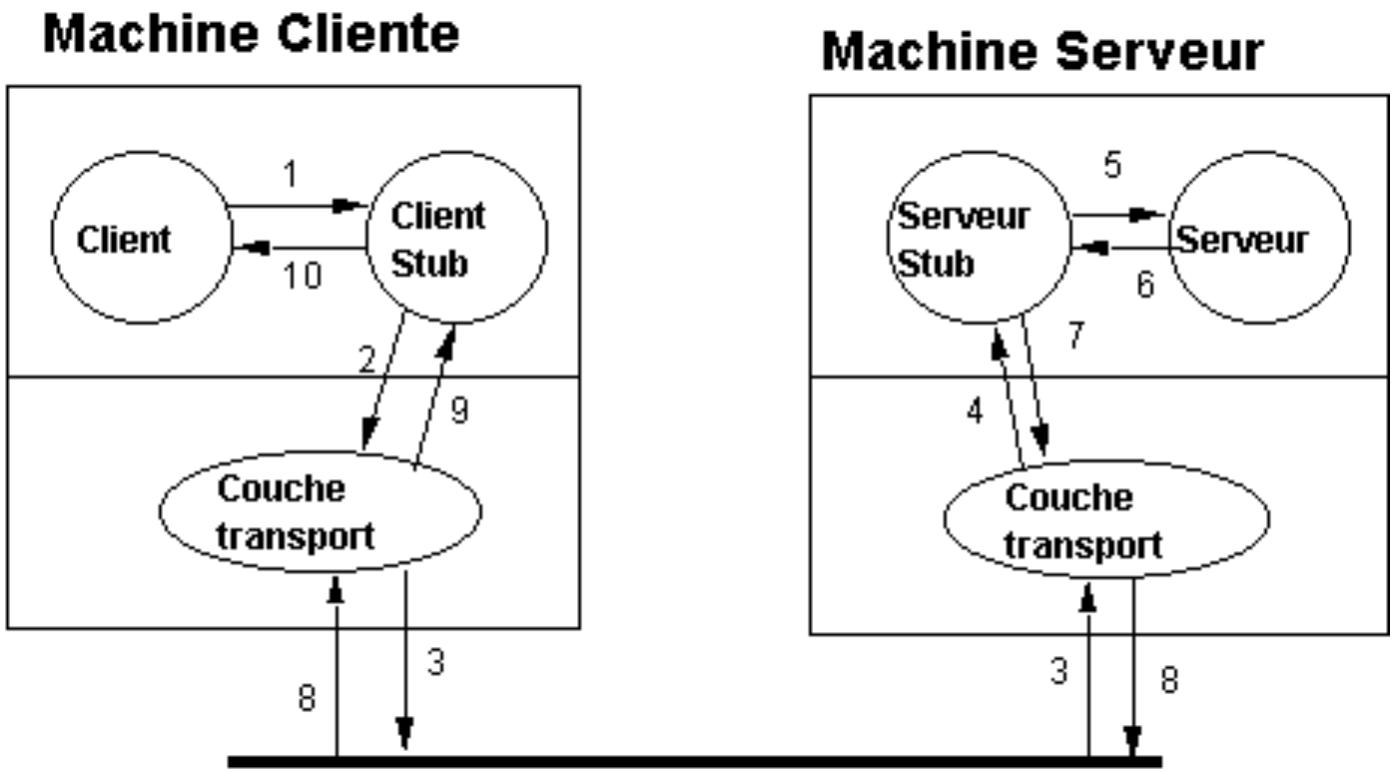
Types composites (décodage : **allocation** taille variable) :

```
xdr_string, xdr_array, xdr_bytes
 bool_t xdr_string (xdr_handle, ptr, lgmax)
 XDR *xdr_handle;
 char **ptr;
 const unisigned int lgmax;
```

Création des flux XDR :

```
xdrmem_create(&xdr_encode, tab, TAILLE, XDR_ENCODE);
xdrmem_create(&xdr_decode, tab, TAILLE, XDR_DECODE);
```

Etapes de connexion



RPC : niveaux d'utilisation pratique

- **Niveau haut** : certaines fonctions standard appelables à distances sont définies dans la bibliothèque **`libpcsrv`** (utilisation de RPC en «interne»)
  - Exemple : **`rnusers`** (<machine>) : renvoie le nombre d'utilisateurs connectés sur la machine; **`nfs`**; **`mount`**; **`nis`**.

- **Niveau intermédiaire** : trois étapes côté serveur
  1) écriture des fonctions d'encodage/décodage et fonction appelable à distance
  2) enregistrement des services
  3) mise en attente

- **Bas niveau** : en plus du niveau intermédiaire (et au prix d'une programmation plus «système») possibilité de gestion fine des échanges de message :
  - le type des paquets (TCP ou UDP),
  - les délais maximaux admis,
  - les retransmissions, les échecs de communications, etc.

RPC identification par triplet

- **< n° de programme, n° de version, n° de procédure >**
- Sun possède le contrôle des numéros 0x00000000 à 0x1fffffff
- Pour les programmes utilisateurs : 0x20000000 jusque 0x40000000 exclu.

0x00100000 portmapper (portmap sunrpc. Manages use of transport ports)
0x00100002 rusersd (Remote users)
0x00100003 nfs (Network file system)
0x00100004 ypserv (ypprog. Yellow pages directory service)
0x00100005 mountd (Mount protocol)
0x00100009 yppasswd (Yellow pages password server)

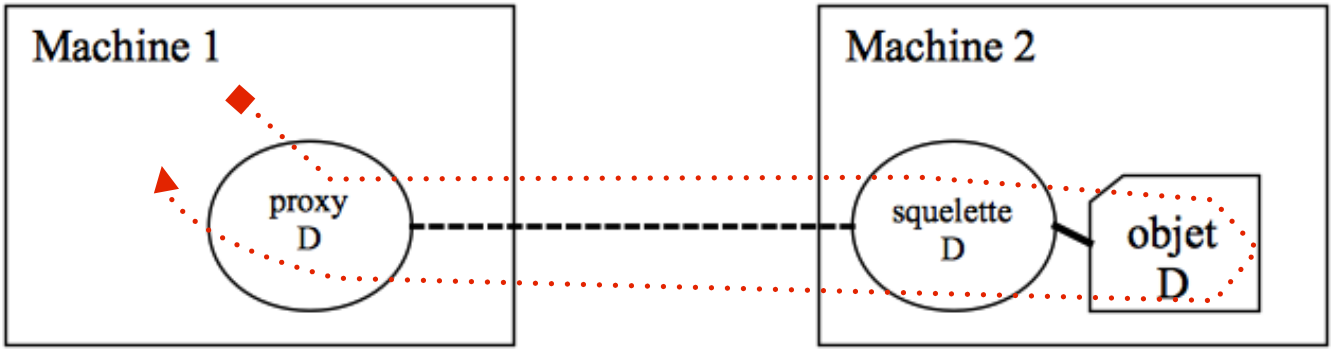
Serveur de liaison (ou de nommage) : lien entre le triplet id -> port_server

- **Démarrage du serveur RPC via son serveur de liaison : il indique le port sur lequel il écoute au *portmapper, portmap (Linux), rpcbind (UNIX)*.**
- **Connection client - serveur : contacte du serveur de liaison de la machine hébergeant le serveur (programme n° 100000 sur le port 111).**
  - **`rpcinfo`**

RMI : Remote Method Invocation

- **Middleware pour définir des applis distribuées en... JAVA (SUN encore)**
  - Mêmes objectifs/motivations que RPC mais en Prog. Objet

- **Notion de proxy/talon/souche (stub)**
  - interface de l'objet distant pour la sérialisation
  - vs. skeleton/squelette coté serveur pour la désérialisation

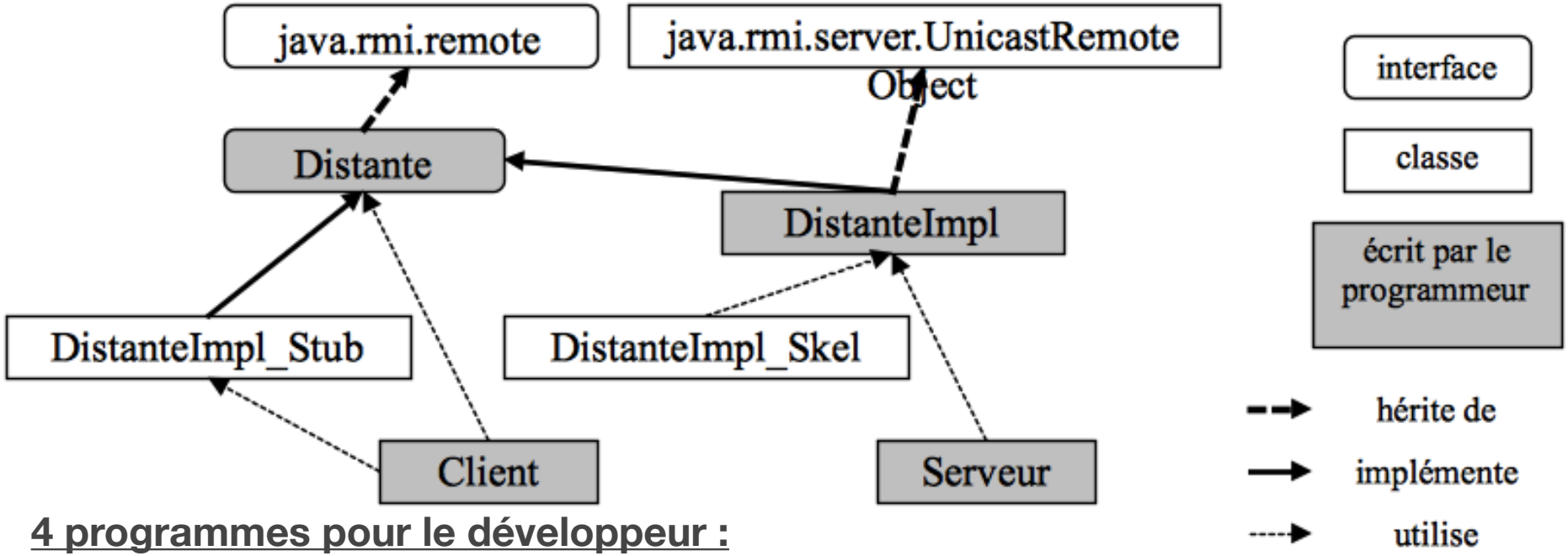


RMI : bien ou pas bien ?

- **Pros**
  - RMI est simple à mettre en oeuvre
  - RMI étend le garbage collector de java (DGC)
  - RMI permet une gestion de la sécurité (RMISecurityManager)

- **Cons**
  - RMI est la propriété de SUN
  - RMI n'offre la possibilité d'interagir avec des objets d'autres langages (interopérabilité)
  - RMI est plus lent que les implémentations de CORBA

Graphe de classe pour une app RMI



- **4 programmes pour le développeur :**
  - **Le «contrat abstrait» : interface d'appel à distance**
  - **Son implem : les fonctions du service**
  - **Le serveur : il crée le service concret**
  - **Le client : il appelle le service via l'interface**

En pratique (en quatre «étapes» - I)

- **Message.java // le contrat abstrait (le proxy pour le client)**

```
import java.rmi.Remote;

import java.rmi.RemoteException;

public interface Message extends Remote {
    public String messageDistant()
    throws RemoteException ;
}
```

En pratique (II)

- **MessageImpl.java // le «vrai service» coté serveur**

```
  import java.rmi.server.UnicastRemoteObject ;
  import java.rmi.RemoteException ;

public class MessageImpl
    extends UnicastRemoteObject
    implements Message  {

    public MessageImpl () throws RemoteException {super();};
    public String messageDistant() throws RemoteException {
   return("Message : Salut !!!") ;
    }
}
```

En pratique (III)

- **Serveur.java // instantiation du service concret**

```
  import java.net.* ;
  import java.rmi.* ;

public class Serveur {
   public static void main(String [] args) {
       try {
      MessageImpl objLocal = new MessageImpl ();
      Naming.rebind("rmi://localhost:1099/Message",objLocal) ;
      System.out.println("Serveur pret");
       }
    catch (RemoteException re) { System.out.println(re) ; }
    catch (MalformedURLException e) { System.out.println(e) ; }
  }
}
```

En pratique (IV)

- **Client.java // l'appel au service via le stub coté client**

```
import java.rmi.* ;
import java.net.MalformedURLException ;

public class Client {
    public static void main(String [] args) {
       try {
    Message b =(Message) Naming.lookup("//"+args[0]+"/Message");
    System.out.println("Le client recoit : "+b.messageDistant());
        }
        catch (NotBoundException re) { System.out.println(re) ; }
        catch (RemoteException re) { System.out.println(re) ; }
        catch (MalformedURLException e) { System.out.println(e) ; }
    }
}
```

Utilisation

- **Compilation + Déploiement**
  - `javac *.java`
  - `rmic MessageImpl (pour le stub et le skeleton)`
  - `rmiregistry (sur la machine du serveur)`
  - `java Serveur (sur la machine du serveur)`
  - `java Client <machine Serveur> (sur la machine du client)`

Fonctionnement interne

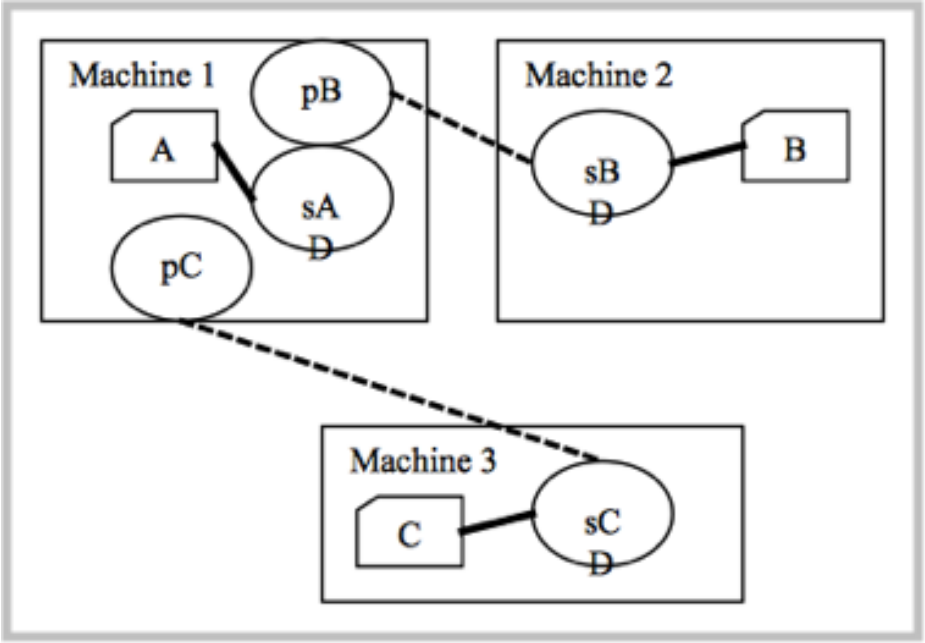
- **Passage d'arguments**
  - tjs par référence ?
    - les primitifs sont passés par valeur (copie)
    - les objts sérialisables (**`java.io.Serializable`**) sont passés par copie
    - les objts accessibles à distance sont référencés
    - sinon exception !
  - **recopie** => «inconsistance» entre instances indépendantes ?
  - la sérialisation linéarise les objs multidimensionnels pour leur exportation

Proxy

- Pour qu'un client puisse appeler une méthode sur un objet distant, il a besoin de disposer d'un objet local, **un proxy**.
  - Pour l'obtenir, deux solutions sont possibles :
    - Utiliser un service de nommage (i.e associations entre objet accessible à distance et nom externe = une chaîne de caractères ou une URL).
    - Avoir appelé une méthode (à distance) qui transmet/renvoie un (autre) objet accessible a distance.

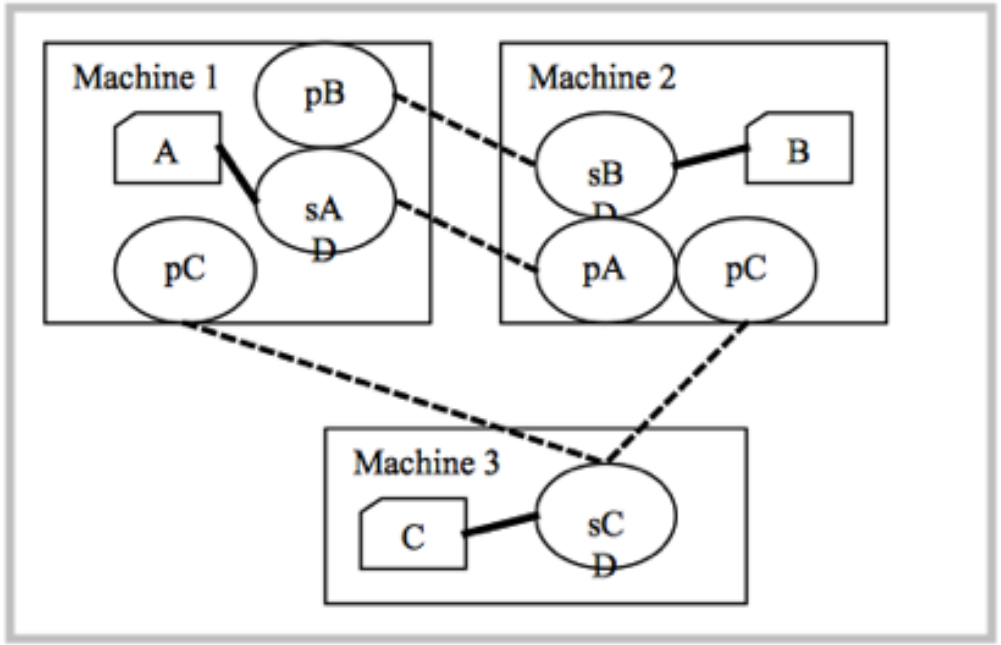
- **Fonctionnement du proxy :**
  - Le client récupère une référence sur le proxy local de l'objet distant.
  - Ce proxy implémente l'interface de l'objet distant. A chaque appel de méthode sur le proxy, il contacte le squelette de l'objet sur la machine du serveur et déclenche à distance la même méthode.
  - Le squelette sait lui-même déclencher une méthode de l'objet du serveur.

Exemple sur trois machines



Appel de méthode depuis la machine 1 :
B.methodeDist(A,C)

Sur la machine 2
- création auto. de pA et pC
- exécution de la méthode



Service de Nommage

- La publication des objets est réalisée via un processus d'enregistrement/ nommage, i.e. au sein du programme médiateur **`rmiregistry`**.
  - attribution d'un nom (une simple chaîne de caractères).
- **Ce nom désigne l'objet pour un programme client désirant y accéder ; celui-ci correspond à une URL.**
  - Par exemple, "rmi://blabla.u-strasbg.fr:1099/monObjet" désigne un objet référencé avec le nom "*monObjet*" sur la machine "*blabla*".
  - Il est possible d'omettre le nom de la machine et/ou le port à utiliser (les valeurs par défaut : localhost:1099.
- **Le serveur de nom peut être démarré de façon logicielle par un appel à `java.rmi.registry.LocateRegistry.createRegistry(8000);`**
  - Fonctions disponibles :
    - **`bind(), rebind(), unbind(), list(), lookup()`**

Couches de Transport

- Pour l'utilisateur, les connexions semblent s'effectuer directement entre le client et le serveur...mais, dans la pratique, les données opèrent un parcours à travers un certain nombre de couches réseaux.
- **Deux des couches traversées sont le Stub et le Skeleton qui sont utilisés respectivement sur le client avec un rôle de proxy et sur le serveur avec pour rôle la gestion des communications vers le Stub des clients.**
  - Ces deux couches prennent la forme de classes Java compilées (générées avec `rmic`).
  - Le serveur devra avoir accès aux Stubs et aux Skeletons des classes implantées alors que seules les Stubs doivent être accessibles aux clients.
  - Les connexions et transferts d'informations sont entièrement pris en charge par Java : la couche TCP/IP est utilisée via un protocole dédié (Java Remote Method Protocol, JRMP).
    - TCP/IP fournit une connexion persistante entre deux machines définie par des adresses IP et des ports à chaque bout.

Packages Java.RMI

- **Les RPC sont basés sur la notion d'appel de procédure, tandis que RMI travaille sur des objets.**
- **Quatre packages Java pour l'utilisation des RMI :**
  - **`java.rmi`** : Définition des classes, interfaces et exceptions qui concernent le client (accès automatisé à des méthodes distantes).
  - **`java.rmi.server`** : Définition des classes, interfaces et exceptions qui concernent le serveur (création d'objets distants à l'intention de clients).
  - **`java.rmi.registry`** : Localisation et dénomination des objets distants (publication des objets distants accessibles).
  - **`java.rmi.dgc`** : Ramasse miettes distribué (récupération automatique de la mémoire qui n'est plus utilisée).

Plus en détail

- **Chargement dynamique de classes**
  - **`java.rmi.server.codebase=<url>`**

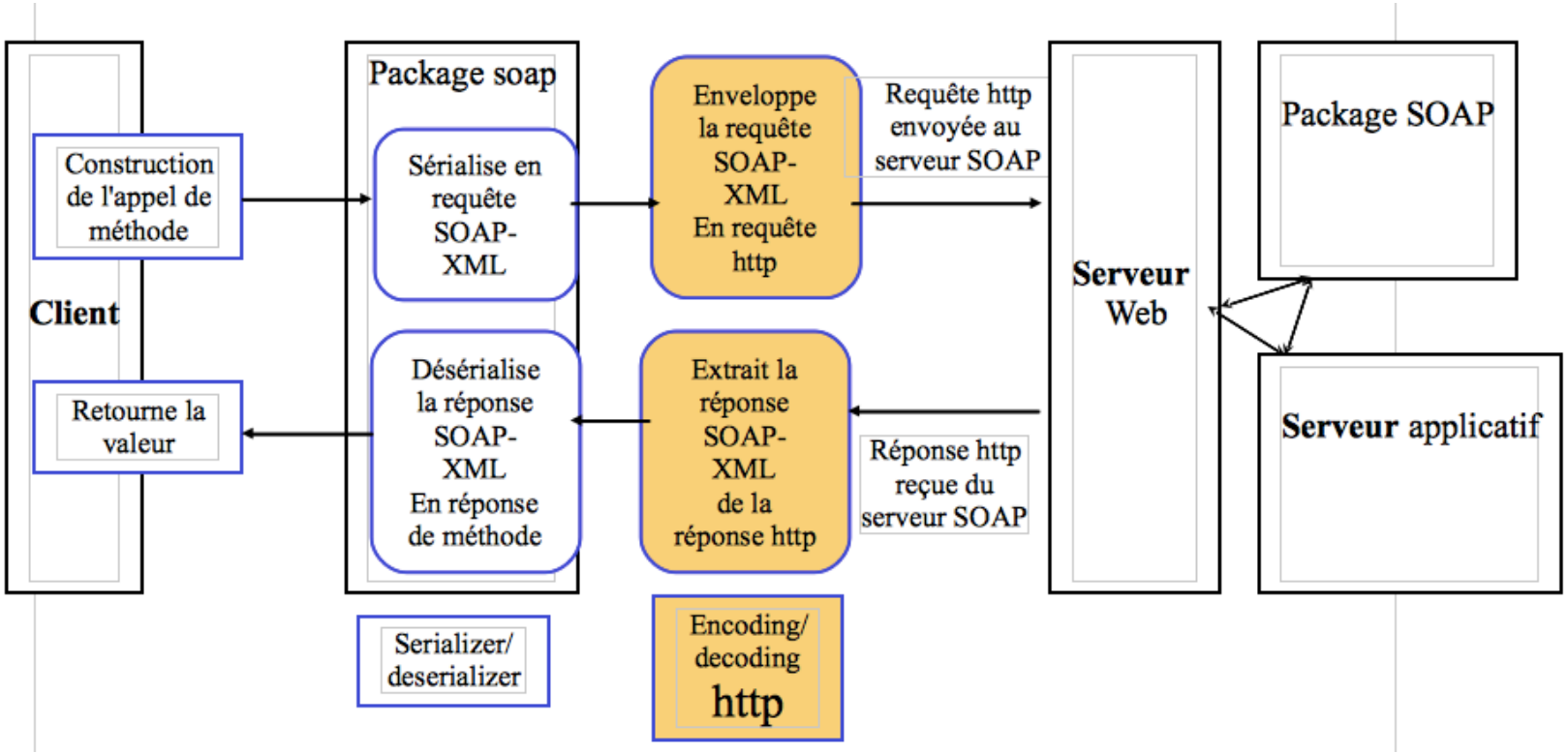
- **Sécurité**
  - Quel politique de protection ?

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

- **Sérialisation**
  - ***`ObjectOutputStream`***
  - ***`writeObject / readObject`***

SOAP : Simple Object Access Protocol (!?!)

- **Objectif : offrir un service web (HTTP en pratique)**
  - par ex. sur une architecture SOA pour le système de réservation SNCF



Pourquoi SOAP ?

- **Problème du passage à l'échelle des solutions précédentes**
  - Firewall :(
  - Contraintes coté client avec les souches/proxy
  - Compléxité de mise en oeuvre de plateforme dédiée

- **SOAP est simple et relativement léger**
- **SOAP est basé sur du XML pour son extensibilité...plus si léger que ça :(**
- **SOAP est portable**
- **SOAP est normalisé (W3C)**

Langage XML : eXtensible Markup Language

- **Basé sur un ensemble non fini de balises**
- **Meta-Langage pour définir des grammaires**
- **Permet de distinguer la forme et le fond**

- **Notion de Schéma (W3C)**
  - un schéma définit les éléments syntaxiques et sémantiques ~ une grammaire
  - expressivité puissante : type, structure, héritage
  - un doc XML est validé/vérifié s'il est conforme à une grammaire

- **Dans le contexte SOAP**
  - définit/structure/formate/contrôle les données et leurs échanges

XML : Namespace

- **Espace de nommage : gestion des conflits de noms**
  - plusieurs Schemas dans un seul doc.
    - par ex : dissocier les elts HTML des balises XML

- **Référence unique**
  - URI = Universal Resource Identifier
    - ex : xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      - affecte le préfixe "rdf" à l'espace nominatif donné dans l'URL
    - xmlns="http://purl.org/rss/1.0/"
      - déclare que l'espace nominatif par défaut est "http://purl.org/rss/1.0/"

Les messages SOAP

- **Namespaces SOAP**
  - Balises : http://schemas.xmlsoap.org/soap/envelope
  - Encodage : http://schemas.xmlsoap.org/soap/encoding

- **Structuration**
  - Une délaration XML (en option)
  - Une envelope SOAP, une racine avec un ou deux fils
    - Header (en option) : pour interpréter le msg
    - Body (obligatoire) : données du msg
      - Des données de type doc XML
      - ou des méthodes RPC et leurs paramètres

Exemple simple

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/
envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/
soap/encoding/"/>

  <SOAP-ENV:Body>
    <m:GetLastTradePriceResponse xmlns:m="Some-URI">
      <Price>34.5</Price>
    </m:GetLastTradePriceResponse>
  </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">

  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="Some-URI">
      <symbol>DIS</symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

Type de données

- **Vocabulaire SOAP pour l'encodage des données typées**
- Value : Simple value (string, integers), Compound value (array, struct)
- Type : Simple Type, Compound Type

- **Valeurs encadrées par des balises XML**
  - ré-utiliser des Schéma existant

```
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/1999/XMLSchema"
        <SOAP-ENC:int>45</SOAP-ENC:int>
        <cost xsi:type="xsd:float">29.5</cost>
```

Traitement d'un message SOAP

- **Les éléments nécessaire pour un appel à distance**
  - l'adresse du noeud SOAP final
  - le nom du service (ou classe ou méthode) à invoquer
  - l'identification et le type des paramètres E/S
- **Un header HTTP est défini : SOAPAction**
- **Le programmeur ne gère que très rarement la partie XML**
- **Notion de relais éventuels**
  - scanne tous les blocs du Header le concernant
  - traite et enlève les blocs le concernant
  - si besoin ajoute ses propres blocs
  - détermine le noeud suivant et lui transmet le msg modifié

WSDL : Web Services Description Language

- **Interface publique d'accès à un service Web = liste de ports et d'opérations**
- **Deux parties dans une telle description**
  - interface : description du services (méthode, types de params)
  - implantation : description des aspects techniques, e.g., protocole utilisé, adresse du service, pour la connection effective.
- **En pratique, un doc XML avec les balises suivantes :**
  - <definitions> : racine du WSDL = nom du service + namespaces
  - <portType> : un ensemble d'<operation> (nom + param E/S)
    - <message> : E/S d'une <operation>
  - <binding> : association <portType> <-> protocole (ici SOAP)
  - <service> : URI + collection de <port> (i.e. associations <binding> <-> URI)

Exemple WSDL

```
<definitions name="HelloService"

   targetNamespace="http://www.examples.com/wsdl/HelloService.wsdl"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://www.examples.com/wsdl/HelloService.wsdl"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   <message name="SayHelloRequest">
      <part name="firstName" type="xsd:string"/>
   </message>
   <message name="SayHelloResponse">
      <part name="greeting" type="xsd:string"/>
   </message>

   <portType name="Hello_PortType">
      <operation name="sayHello">
         <input message="tns:SayHelloRequest"/>
         <output message="tns:SayHelloResponse"/>
      </operation>
   </portType>

...
```

```
   <binding name="Hello_Binding" type="tns:Hello_PortType">
   <soap:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http"/>
   <operation name="sayHello">
      <soap:operation soapAction="sayHello"/>
      <input>
         <soap:body
            encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"
            namespace="urn:examples:helloservice"
            use="encoded"/>
      </input>
      <output>
         <soap:body
            encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"
            namespace="urn:examples:helloservice"
            use="encoded"/>
      </output>
   </operation>
   </binding>

   <service name="Hello_Service">
      <documentation>WSDL File for HelloService</
documentation>
      <port binding="tns:Hello_Binding" name="Hello_Port">
         <soap:address
            location="http://www.examples.com/SayHello/">
      </port>
   </service>
</definitions>
```