Programmation Répartie

Généralités - implémentation

Yvan Peter

IUT A - Université Lille 1

2017-2018

Objectifs / évaluation

Objectifs

- Comprendre les spécificités de la programmation répartie
- Comprendre la notion de protocole
- Savoir réaliser une communication client-serveur avec des sockets (TCP)
- Appréhender les architectures client-serveur récentes
- Savoir développer selon les principes REST en Java

Programmation répartie : motivation

- Une nécessité depuis le développement de l'ordinateur personnel et des réseaux
- Les ressources sont naturellement réparties
  - Bases de données, fichiers...
- Certain services sont mis en place sur des serveurs spécifiques
  - DNS, serveur mail, serveur web...
- La communication entre utilisateurs s'est développée
  - mail, chats, téléphonie, réseaux sociaux...

Programmation vs. programmation répartie

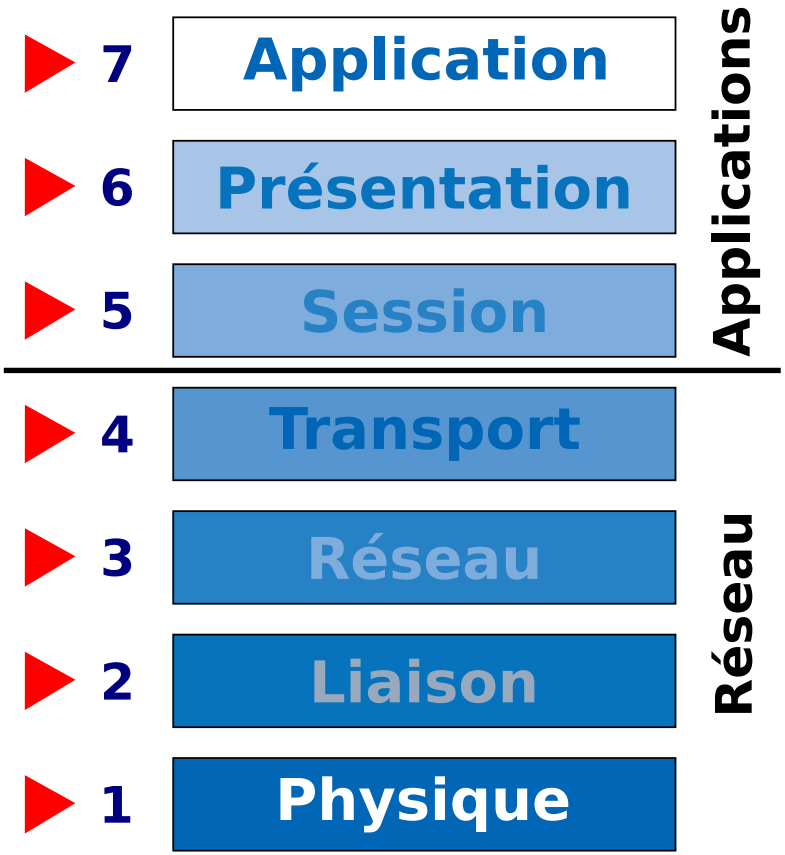
Programmation classique

- Ajout de fonctionnalité par utilisation d'une librairie, d'un objet, etc
- Tout le code nécessaire s'exécute dans le même espace d'adressage (processus)
- Erreurs simples (plantage du processus)

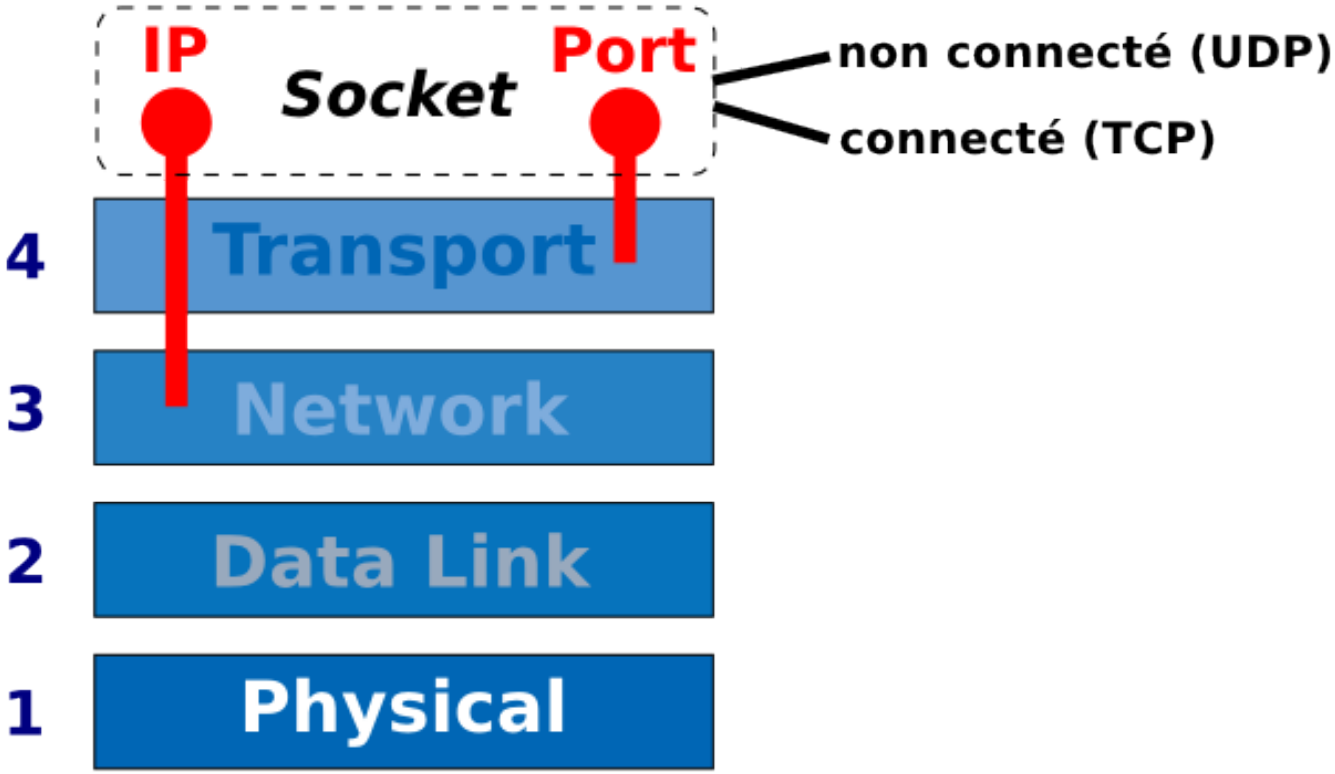
Programmation répartie

- Les fonctionnalités/ressources nécessaires sont dans des espaces d'adressages différents
- L'accès au service peut prendre du temps (communication réseau) voire échouer
- Les erreurs peuvent être plus complexes quand seulement une partie du service n'est pas accessible
- Les problèmes d'hétérogénéité doivent être pris en compte
- Il est nécessaire de pouvoir identifier et localiser les services

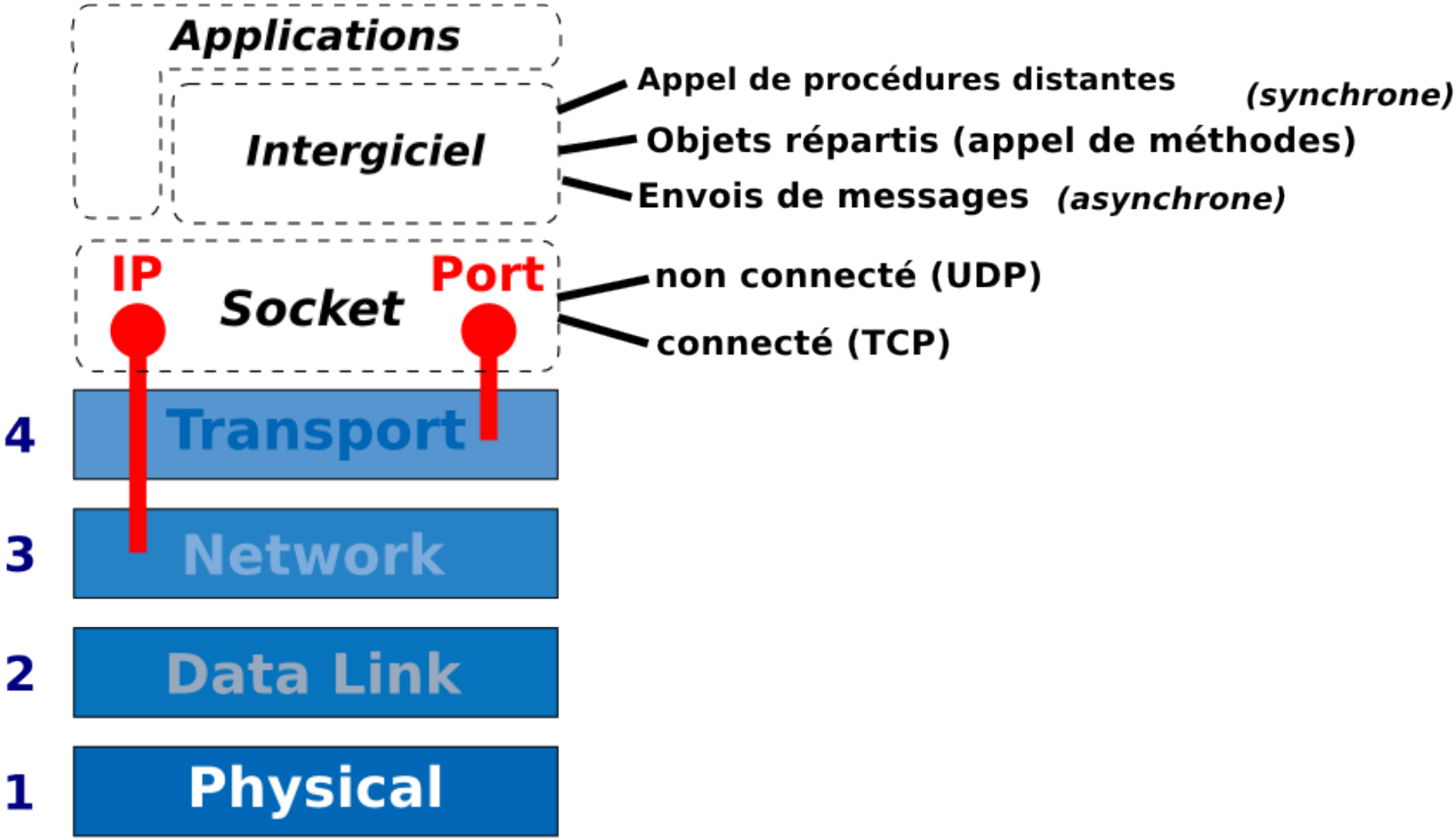
Programmation répartie



Programmation répartie



Programmation répartie



Généralités

- Les sockets permettent une connexion
  - point à point
    - sans connexion (UDP)
    - avec connexion (TCP)
  - multipoint (adresses de classe D)
- Une communication est identifiée par :
  - les adresses source et destination (= les machines)
  - les numéros de ports source et destination (= les applications)
- Le couple {adresse IP, port} constitue une socket

Généralités

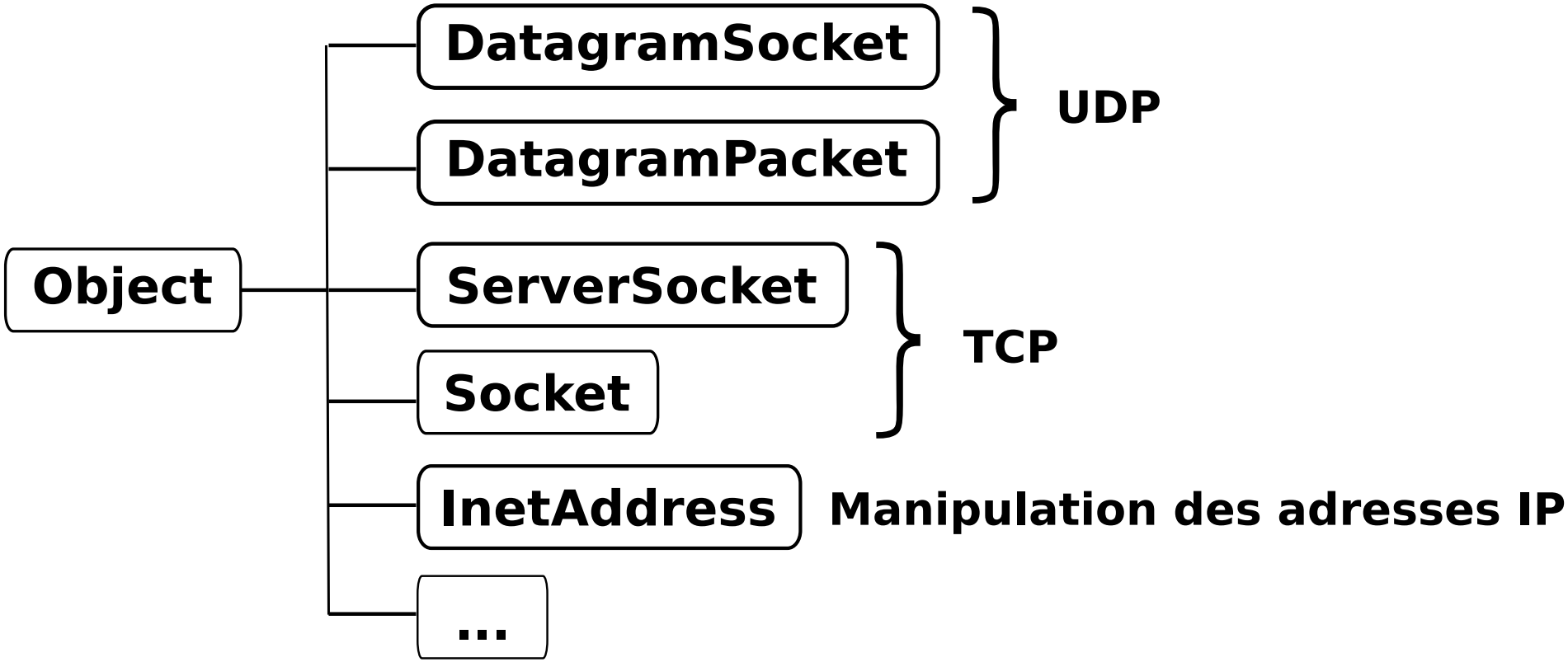
Le client

- est à l'origine de la communication. Il envoie des requêtes à un service particulier.
- utilise un numéro de port éphémère fournit par le système.

Le serveur

- fournit un service. Il attend les connexions des clients.
- utilise en général un numéro de port fixe (voir `/etc/services`)

Les classes de java.net



Manipulation des adresses IP

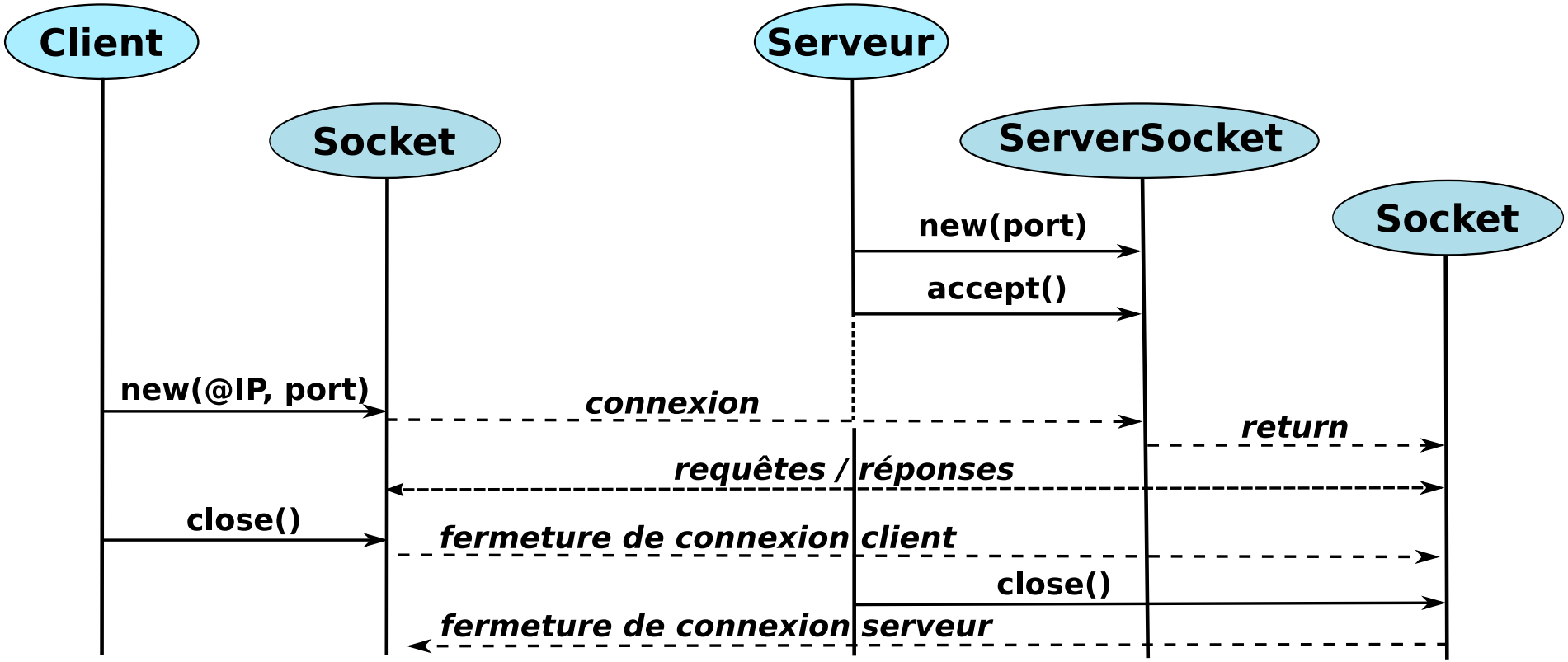
La classe `InetAddress`

- représente une adresse IP (v4 ou v6) et éventuellement le nom associé
- permet la résolution nom → adresse

Exemple

```
InetAddress ip = InetAddress.getByName("localhost");
```

TCP : fonctionnement



TCP : la classe `ServerSocket`

Permet de créer et manipuler un point de connexion (associant adresse IP et port) pour le serveur

Exemple

Le serveur crée un point de connexion puis attend celles-ci via la méthode `accept()` bloquante.

```
ServerSocket serveur = new ServerSocket(1234);
Socket s = serveur.accept();
...
serveur.close();
```

TCP : la classe `Socket`

Représente une connexion entre un client et un serveur. Outre les informations sur la connexion, cette classe donne accès aux flux d'entrée/sortie de la socket.

Exemple

- Côté client : on crée une socket pour un serveur particulier :

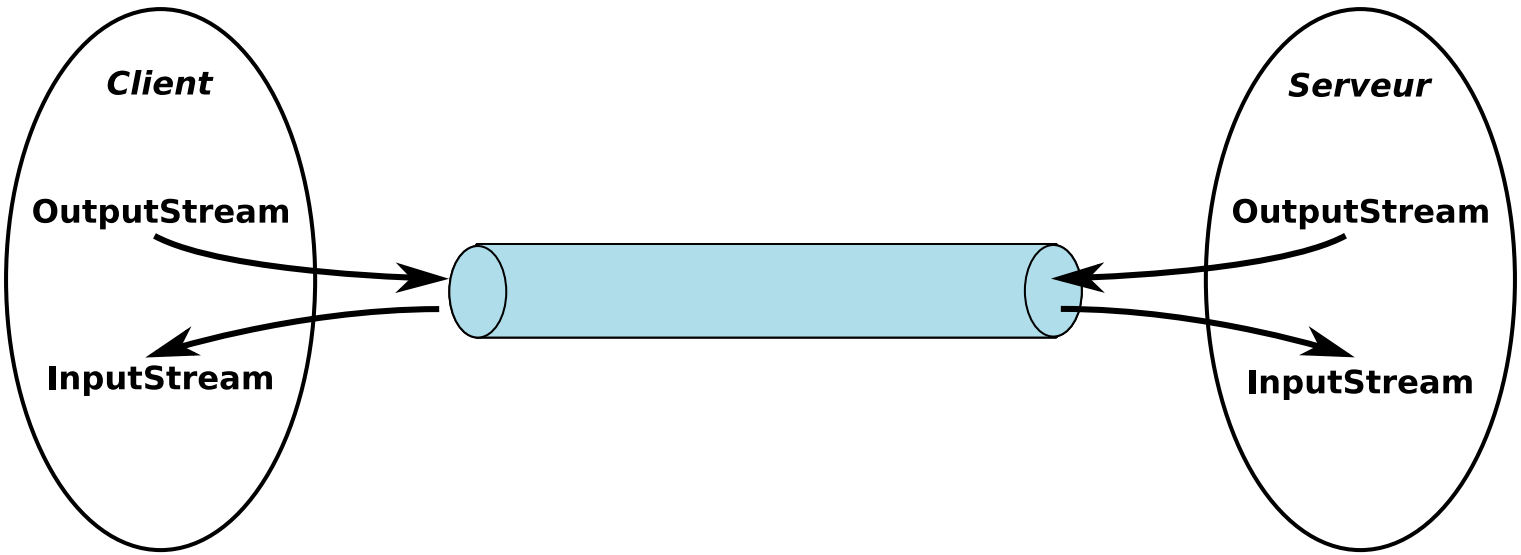
```
Socket client = new Socket("localhost", 1234);
System.out.println( client.getLocalPort() );
...
client.close();
```

- Côté serveur, la connexion d'un client débloque la méthode `accept()` qui renvoie une instance de `Socket` correspondant à la nouvelle connexion.

TCP : les entrées-sorties

La classe `Socket` permet de récupérer un flux d'octets en entrée et en sortie pour la communication via les méthodes :

```
public InputStream getInputStream()
public OutputStream getOutputStream()
```



TCP : les entrées-sorties

`InputStream` et `OutputStream` sont des classes abstraites. A vous de choisir l'implémentation qui vous convient.

Exemple

```
BufferedInputStream file =
  new BufferedInputStream(new FileInputStream("photo.jpg"));
BufferedOutputStream out = client.getOutputStream();
int b;
while ( (b = file.read()) != -1 ) {
  out.write(b);
}
file.close();
out.close();
```

TCP : les entrées-sorties

Quand le client et le serveur échangent des chaînes de caractères, il est plus pratique d'utiliser des flux de caractères (par ex. `PrintWriter` et `BufferedReader`).

Exemple

- flux de caractères en écriture :
  `envoi = new PrintWriter(client.getOutputStream(), true);`
  paramètre **true** : `println`, `printf` et `format` provoquent le vidage du buffer
- flux de caractères en lecture :
  `reception = new BufferedReader( new InputStreamReader(client.getInputStream()));`

Servir plusieurs clients

Pendant que le serveur sert un client, les demandes de connexion sont mises en file d'attente jusqu'au prochain appel à la méthode `accept()`.

Pour servir plusieurs clients, on créé un thread par connexion.

```
while ( true ) {
  unClient = serveur.accept();
  new ThreadClient(unClient).start();
}
```

La classe `ThreadClient` peut implémenter l'interface `Runnable` ou hériter directement de la classe `Thread`.

Servir plusieurs clients

Exemple avec Thread

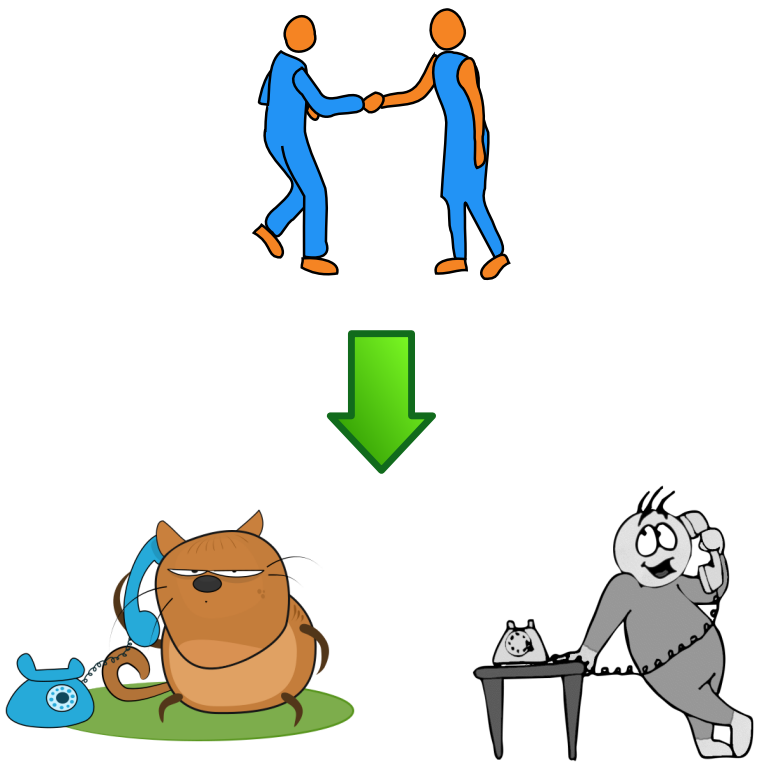
```
public class ClientThread extends Thread {
  private Socket client;
  public ClientThread(Socket c) {
    client = c;
  }
  public void run() {
    ...
     envoi.println("Salut client");
  }
}
  ...
   while ( true ) {
     unClient = serveur.accept();
     new ClientThread(unClient).start();
  }
  ...
```

Servir plusieurs clients

Exemple avec Runnable

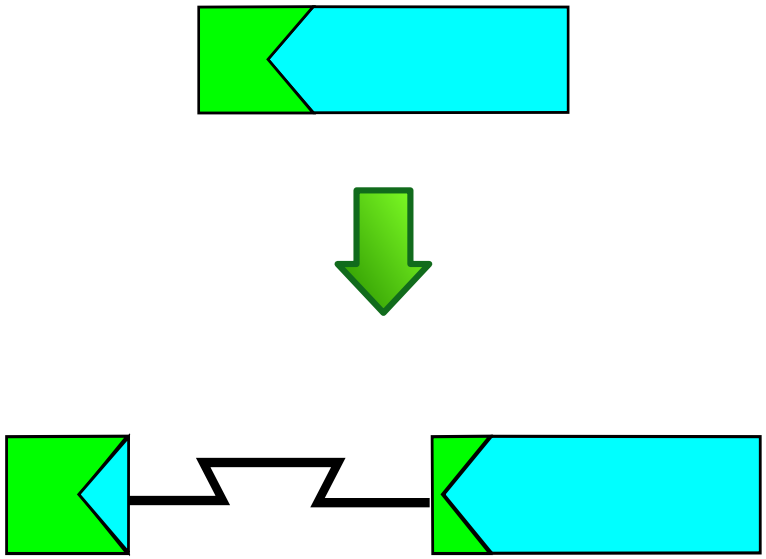
```
public class ClientRunnable implements Runnable {
  private Socket client;

  public ClientRunnable(Socket c) {
    client = c;
  }
  public void run() {
    ...
    envoi.println("Salut client");
  }
}
...
  while (true) {
    unClient = serveur.accept()
    new Thread(new ClientRunnable(unClient)).start();
  }
...
```

Motivation



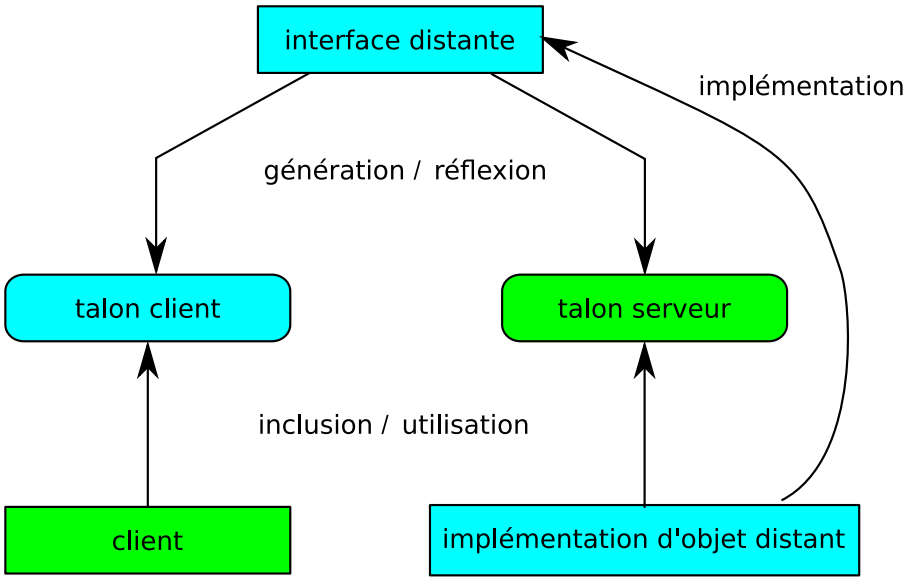
- Collaborer à distance nécessite
  - un téléphone
  - le numéro du correspondant
- On ne se préoccupe pas de la connexion ni du mode de communication

Invocation distante



- La distribution des objets repose sur le même principe
  - un objet intermédiaire local sert de représentant de l'objet distant (*talon*)
  - un identifiant permet de référencer l'objet invoqué (*référence d'objet*)
- le bus de communication se charge de la communication (en utilisant les sockets)

Mise en œuvre



- En général on défini les méthodes accessibles à distance au moyen d'un langage de définition d'interface (IDL ou *Interface Definition Language*)
- Cet interface permet de générer les objets intermédiaires utilisés côté client et serveur

Mise en œuvre

Talon client

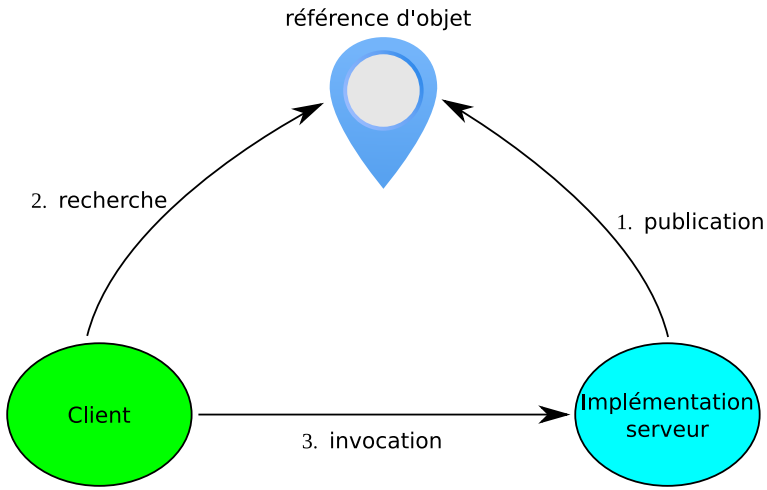
```
public class Talon {
 public <T> methode(args)
 {
  msg = codage(args);
  envoi(msg);
  attente_reponse(r);
  rslt = decodage(r);
  return rslt;
 }
}
```

Talon serveur

```
public class Talon {
 public void methode(requête r)
 {
  args = decodage(r);
  rslt =
   implementation.methode(args);
  msg = codage(rslt);
  envoi(msg);
 }
}
```

Gestion des références

- Une référence d'objet désigne une instance d'une implémentation d'une interface distante
- une implémentation doit pouvoir publier sa référence
- un client doit pouvoir retrouver des références d'objet pour invoquer des méthodes dessus
- les intergiciels à objets disposent en général d'un service d'annuaire ou de courtage pour cela



Mises en œuvre

Approches historiques

- Common Object Request Broker Architecture (CORBA), une implémentation multi-langages et multi-systèmes
- Java Remote Method Invocation, l'implémentation dans java
- Distributed Common Object Model (DCOM), la mise en œuvre de Microsoft

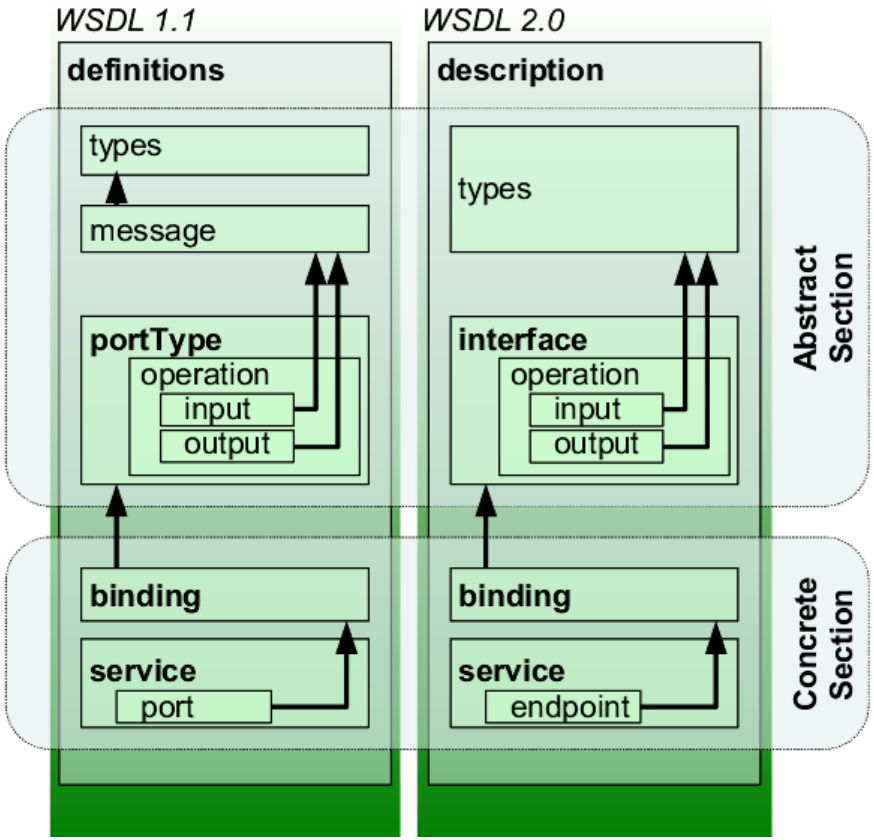
Approches Web Services

- utilisation de protocoles de communication standards pour le transport des requêtes et réponses (par ex. HTTP)
- description des interfaces en XML (Web Service Definition Language, WSDL)
- encodage des méthodes, arguments et résultats en XML (SOAP)

Web Services

Web Service Description Language

WSDL permet de décrire en XML l'interface du service distant



By Cristcost - Own work, Public Domain,

https ://commons.wikimedia.org/w/index.php ?curid=7642526

Web Services

Exemple WSDL

```
<wsdl:types>
  <s:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/">
   <s:element name="Add">
    <s:complexType>
     <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="intA" type="s:int"/>
      <s:element minOccurs="1" maxOccurs="1" name="intB" type="s:int"/>
     </s:sequence>
    </s:complexType>
   </s:element>
   <s:element name="AddResponse">
    <s:complexType>
     <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="AddResult" type="s:int"/>
     </s:sequence>
    </s:complexType>
   </s:element>
  </s:schema>
 </wsdl:types>
```

```
<wsdl:message name="AddSoapIn">
   <wsdl:part name="parameters" element="tns:Add"/>
 </wsdl:message>
 <wsdl:message name="AddSoapOut">
   <wsdl:part name="parameters" element="tns:AddResponse"/>
</wsdl:message>
```

```
<wsdl:portType name="CalculatorSoap">
   <wsdl:operation name="Add">
     <wsdl:documentation>Adds two integers.</wsdl:documentation>
     <wsdl:input message="tns:AddSoapIn"/>
     <wsdl:output message="tns:AddSoapOut"/>
   </wsdl:operation>
 </wsdl:portType>
```

Web Services

Exemple WSDL

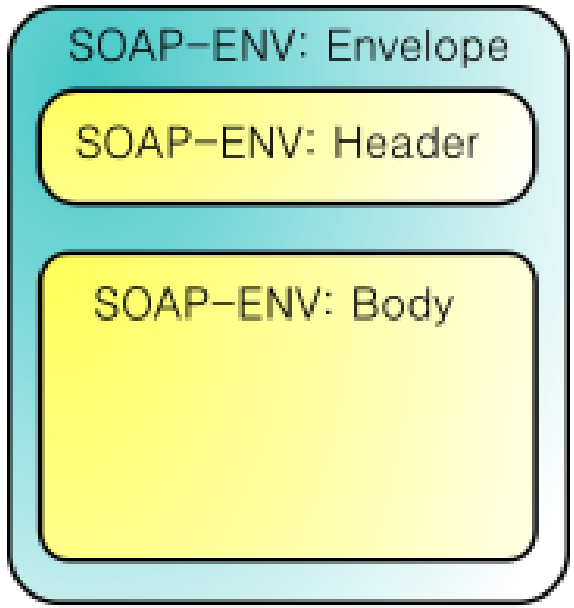
```
<wsdl:binding name="CalculatorSoap" type="tns:CalculatorSoap">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="Add">
    <soap:operation soapAction="http://tempuri.org/Add" style="document"/>
    <wsdl:input>
      <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
```

```
<wsdl:service name="Calculator">
  <wsdl:port name="CalculatorSoap" binding="tns:CalculatorSoap">
  <soap:address location="http://www.dneonline.com/calculator.asmx"/>
  </wsdl:port>
</wsdl:service>
```

Web Services

Simple Object Access Protocol

C'est le protocole de transport des requêtes et réponses décrites en WSDL. Ces messages sont décrits en XML



By Silver Spoon Sokpop - Own work, CC BY-SA 3.0,

https ://commons.wikimedia.org/w/index.php?curid=8019414

Web Services

Exemple SOAP

```
POST /calculator.asmx HTTP/1.1
Host: www.dneonline.com
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/Add"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope>
  <soap:Body>
    <Add xmlns="http://tempuri.org/">
      <intA>32</intA>
      <intB>10</intB>
    </Add>
  </soap:Body>
</soap:Envelope>
```

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

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope>
  <soap:Body>
    <AddResponse xmlns="http://tempuri.org/">
      <AddResult>42</AddResult>
    </AddResponse>
  </soap:Body>
</soap:Envelope>
```