Applications distribuées - 1
Généralités et introduction aux sockets
Module RCPI01

Sebastien.Kramm@univ-rouen.fr

IUT R&T Rouen

2018-2019

Sommaire

Définition

Application distribuée : Définition

- Une **application distribuée** est une application informatique dans laquelle les traitements sont effectués de concert par plusieurs ordinateurs d'un réseau informatique.
- Un protocole de communication établit les règles selon lesquelles les ordinateurs communiquent dans le cadre de l'application en question.

UNIVERSITÉ DE ROUEN

Sommaire

Architecture logicielle & technique

Architecture logicielle

Vue tournée sur l'organisation interne et le découpage d'un logiciel en modules.

- nature des différents modules d'un logiciel,
- responsabilités et les fonctionnalités de chaque module,
- quelle machine va les exécuter, quand,
- nature des relations entre les modules.

Architecture technique

Vue tournée sur l'**organisation logique** de la plateforme informatique : moyens techniques clés qui seront utilisés par tous les logiciels applicatifs.

- matériel informatique,
- logiciels systèmes,
- middlewares,
- réseaux de télécommunication,
- relations entre ces différents éléments.

Modèles d'architecture technique

Classiquement, on distingue ces trois approches :

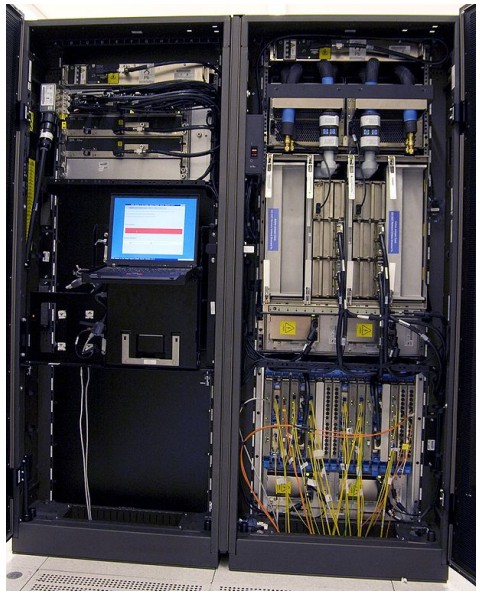
1. Mainframe
2. Client serveur
3. Peer to peer (P2P) → peu utilisé dans le contexte d'applications métier

Aujourd'hui, le "client-serveur" est dominant.

Architecture de type *Mainframe*

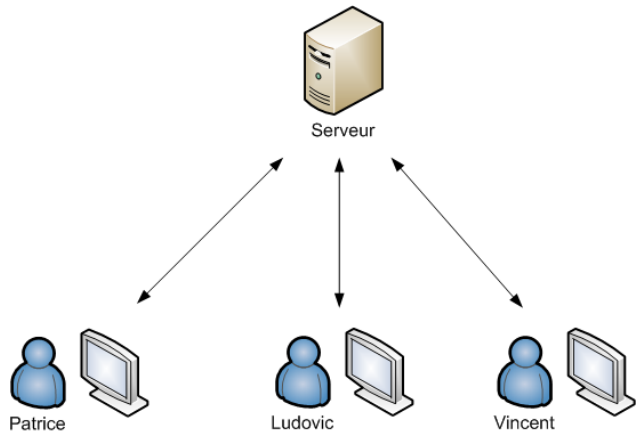
- Ordinateur central de grande puissance de traitement
- Fonctionne selon un modèle centralisé : serveur unique connecté à un ensemble de terminaux ("clients légers")
- Surtout utilisé dans : banque, assurance, organisation gouvernementale :
  - gros volume de transactions
  - impératif de fiabilité

  (≠ superordinateur, qui est lui destiné au calcul haute performance)
- Marché dominé par IBM à 90%
  Voir IBM Z systems



Architecture client serveur

- Mode de communication à travers un réseau entre plusieurs programmes ou logiciels

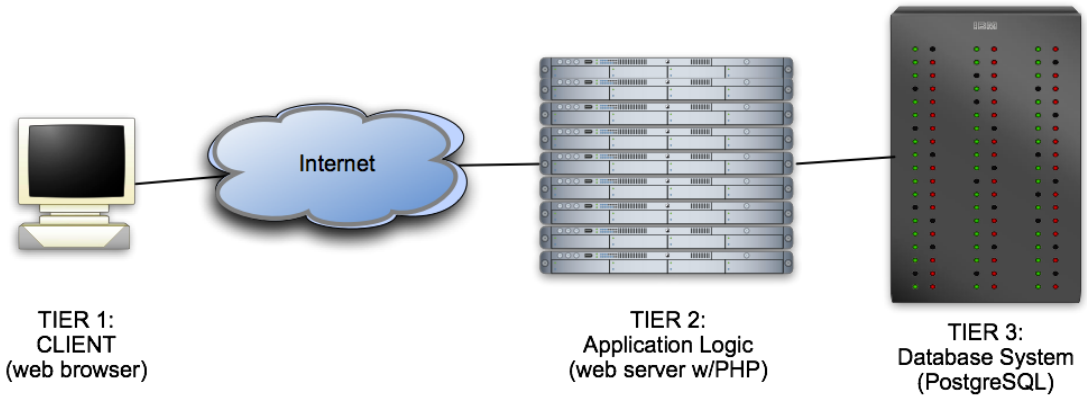


src : OpenClassrooms

  - Le client demande l'exécution d'un service (génération d'une "requête")
  - Le serveur réalise le service et renvoie les données demandées au client
  - Client et serveur sont en général localisés sur deux machines distinctes
  - Indépendance interface-réalisation

Architecture client serveur

- En pratique on parle souvent d'architecture "3-tiers"
  1. Le client : navigateur visualisant une présentation des données, ou objet connecté, ou autre machine/serveur...
  2. Le serveur et son moteur applicatif (PHP, ASP.NET, JEE,...)
     ⇒ Gère la génération de données envoyées (HTML ou XML) et la génération des requêtes au SGBD
  3. Le SGBD (Système de Gestion de Base de Données)



src :University of California, Berkeley

- Architecture "n-tiers : ce découpage peut-être encore affiné, la couche applicative pouvant être décomposée (middleware, serveur web, serveur d'application, etc.)

Client serveur : mise en oeuvre

Plusieurs approches :

- Opérations de bas niveau, fournies par l'OS : sockets
  ⇒ Transfert binaire brut
- Opérations de haut niveau via l'utilisation d'un *middleware* spécialisé
- Approche intermédiaire reposant sur une API ou une architecture
  - programmation fonctionnelle : Appel de procédure à distance (RPC : *Remote Procedure Call*)
  - programmation orientée objet (POO) : Appel de méthodes sur des objets distants
  - WebService : architecture REST

  ⇒ Transfert de données ayant une sémantique

UNIVERSITÉ DE ROUEN

Types de clients

- Client léger
  L'application fonctionne entièrement sur le serveur, le poste client reçoit des réponses pré-formatées à ses requêtes.
  ⇒ navigateur web qui reçoit des pages web complètement statiques
- Client lourd
  Application locale qui s'appuie sur l'OS pour exécuter une partie des traitements.
  - Inconvénient : nécéssite la maintenance du poste client de façon périodique (mises à jour), et nécessite le développement de ce client pour éventuellement plusieurs plateformes :
    ⇒ coûts d'administration élevés
  - Avantage : confort d'utilisation pour l'utilisateur (moins de latence)
- Client riche : approche intermédiaire
  Le client met en oeuvre un outil avec des fonctionnalités comparables à celles d'un client lourd. Les traitements sont effectués majoritairement sur le serveur, et le client gère la partie présentation.
  ⇒ Solution de type « navigateur web + technos AJAX ou Flash »

API ? ? ?

Application Programming Interface

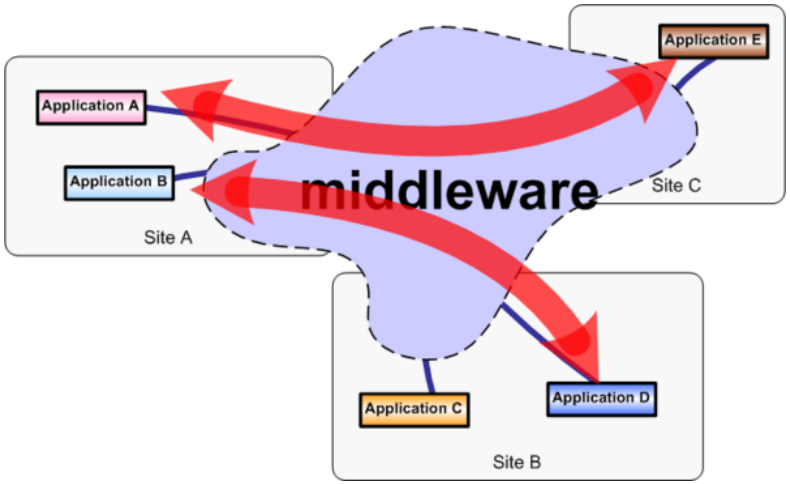
Déf : Ensemble normalisé de classes, de méthodes ou de fonctions qui sert de façade par laquelle un logiciel offre des services à d'autres logiciels.

- Désigne deux réalités :
  - API "locale" : concept de **bibliothèque logicielle** , utilisé par un programmeur pour écrire une application.
    Exemple : un système d'exploitation fournit une API permettant d'accéder aux ressources de la machine.
  - API "distante" : concept de **Service web**
    Protocole permettant la communication et l'échange de données entre applications et systèmes hétérogènes dans des environnements distribués.
    Exemple : API de Google Maps permettant d'obtenir des cartes personnalisées.

UNIVERSITÉ DE ROUEN

Middleware

- Un middleware est un logiciel tiers qui offre des fonctionnalités de support pour les applications réparties
- Objectifs principaux :
  - Masquer la complexité de l'infrastructure sous-jacente
  - Permettre l'interopérabilité entre différents systèmes et technologies
  - Faciliter la conception, le développement et le déploiement d'applications réparties
  - Fournir des services communs
- Exemple de standard sur lequel s'appuie un middleware : CORBA



src :http://middleware.smile.fr

CORBA

- Définition : standard définissant une architecture logicielle pour le développement de composants logiciel, destinés à être assemblés et exécutés dans des processus séparés et/ou sur des machines distinctes.
- Maintenu par l'Object Management Group : organisation de standardisation de plus de 500 entreprises.
  Historique : 1991 : 1.0, 1996 : 2.0, 2002 : 3.0
- Indépendant d'un OS ou d'un langage de programmation
- Approche orientée objet, chaque composant est décrit sous la forme d'une interface écrite dans un langage spécifique IDL: Interface Definition Language, qui décrit l'interface externe de ce composant.
- Standard (trop ?) complexe, difficile à implémenter correctement et complètement
  ⇒ Actuellement en perte de vitesse devant d'autres approches (Web services)

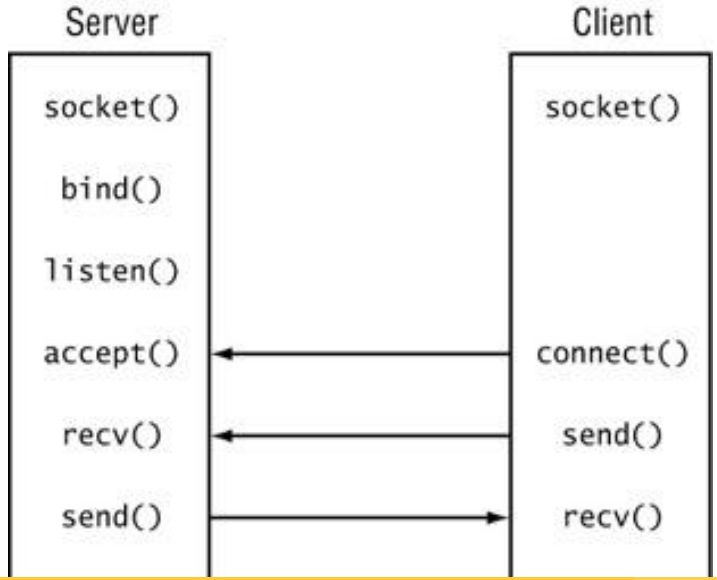
UNIVERSITÉ DE ROUEN

Sommaire

Sommaire

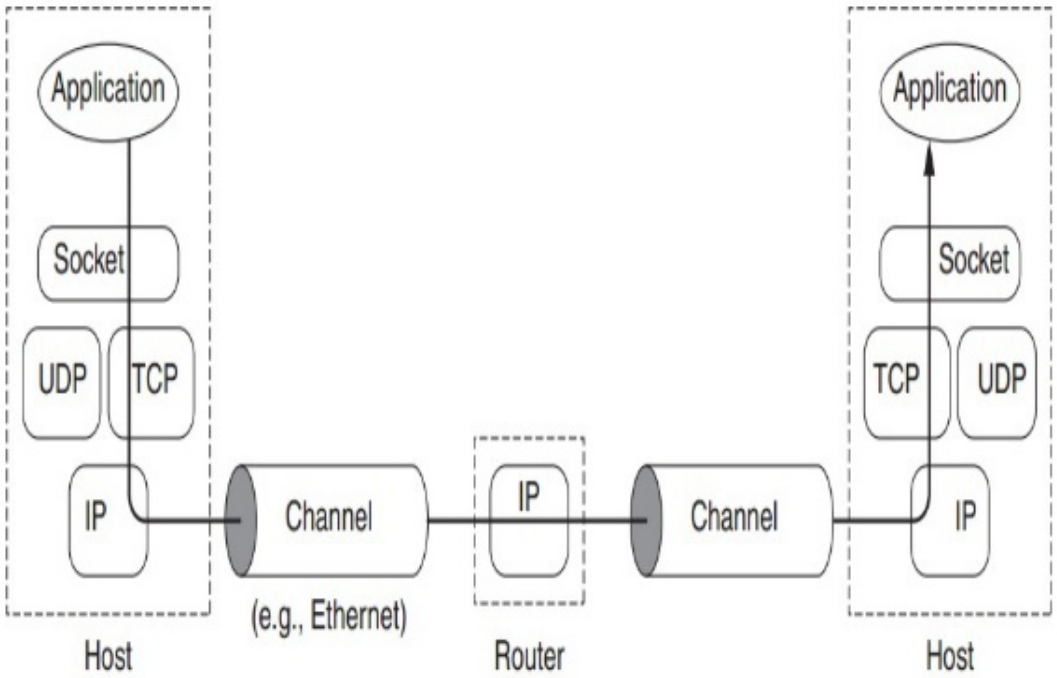
Sockets

- Fonction offerte par les OS, utilisables quel que soit le langage
- Destiné à de la **communication inter processus** : IPC (*Inter-Process Communication*)
- Représente une "prise" bidirectionnelle par laquelle une application peut envoyer et recevoir des données vers un autre processus, local ou distant.
- Histoire : université de Berkeley debut 1980, distribution UNIX BSD (*Berkeley Software Distribution*)
- API de base, implémentée depuis dans tous les OS.



Sockets

- Un socket est identifié par :
  - une adresse IP,
  - un numéro de port sur 16 bits.
- L'aspect transport est transparent (pris en charge par les couches inférieures)



Modes : connecté ou pas

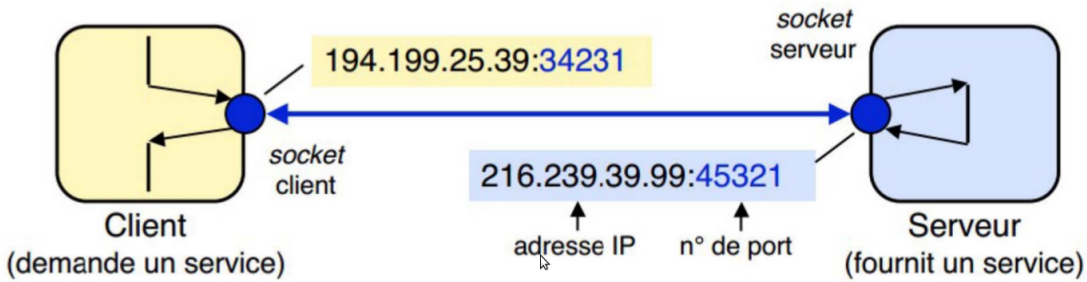
- Mode **non connecté** (UDP) : aucune liaison initiale n'est établie, chaque message est envoyé individuellement
  - Avantage : simplicité de mise en oeuvre
  - Inconvénient : aucune garantie sur la bonne réception !
- Mode **connecté** (TCP) : une liaison préalable aux données est établie
  - Avantage : fiabilité de la transmission
  - Inconvénient : mise en œuvre plus complexe que l'UDP

UNIVERSITÉ DE ROUEN

Sockets : client ou serveur ?

Différences

- Le **client** est à l'initiative de la communication, il doit spécifier l'adresse (IP) et le port du serveur.
- Le **serveur** aura (par défaut) l'adresse de la machine sur laquelle il s'exécute, le socket doit préciser le port d'écoute.



source : C. Zanni-Merck

Remarques

- Client comme serveur peuvent ouvrir des connexions sur plusieurs sockets, on peut avoir plusieurs sockets dans un programme.
- Le choix du port du socket client est géré par l'OS

Sockets : numéro de port

Numéros de port

- Les numéros de port (sur 16 bits) sont **normalisés**, certains sont réservés :
  - 0-1023 : réservés au système
  - 1024-49151 : ports utilisateur, peuvent être enregistrés à l'IANA
  - 49152-65535 : ports non enregistrables à l'IANA

  Si on les prend au hasard on risque de perturber le fonctionnement de l'OS ou un autre programme en cours d'exécution.

Remarque

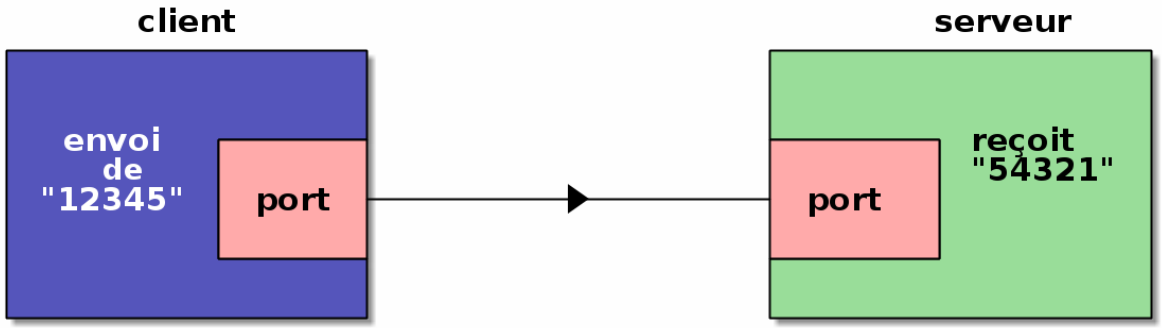
49152 = `$c000` = `1100.0000.0000.0000`

Sockets : difficultés

- Le transfert de données se fait par lecture/écriture de flots d'octets, de façon similaire à un fichier sur disque : données brutes ou formattées en lignes de texte.
- Pas de sémantique sur le transfert de données, c'est au concepteur des logiciels client **et** serveur de définir un protocole :

- Client :
  - quelles données envoyer ?
  - dans quel ordre ?
  - à quel moment ?
- Attention au problème de l'Endianness :

- Serveur :
  - quoi faire avec les données reçues ?
  - quoi répondre au client ?
  - ...



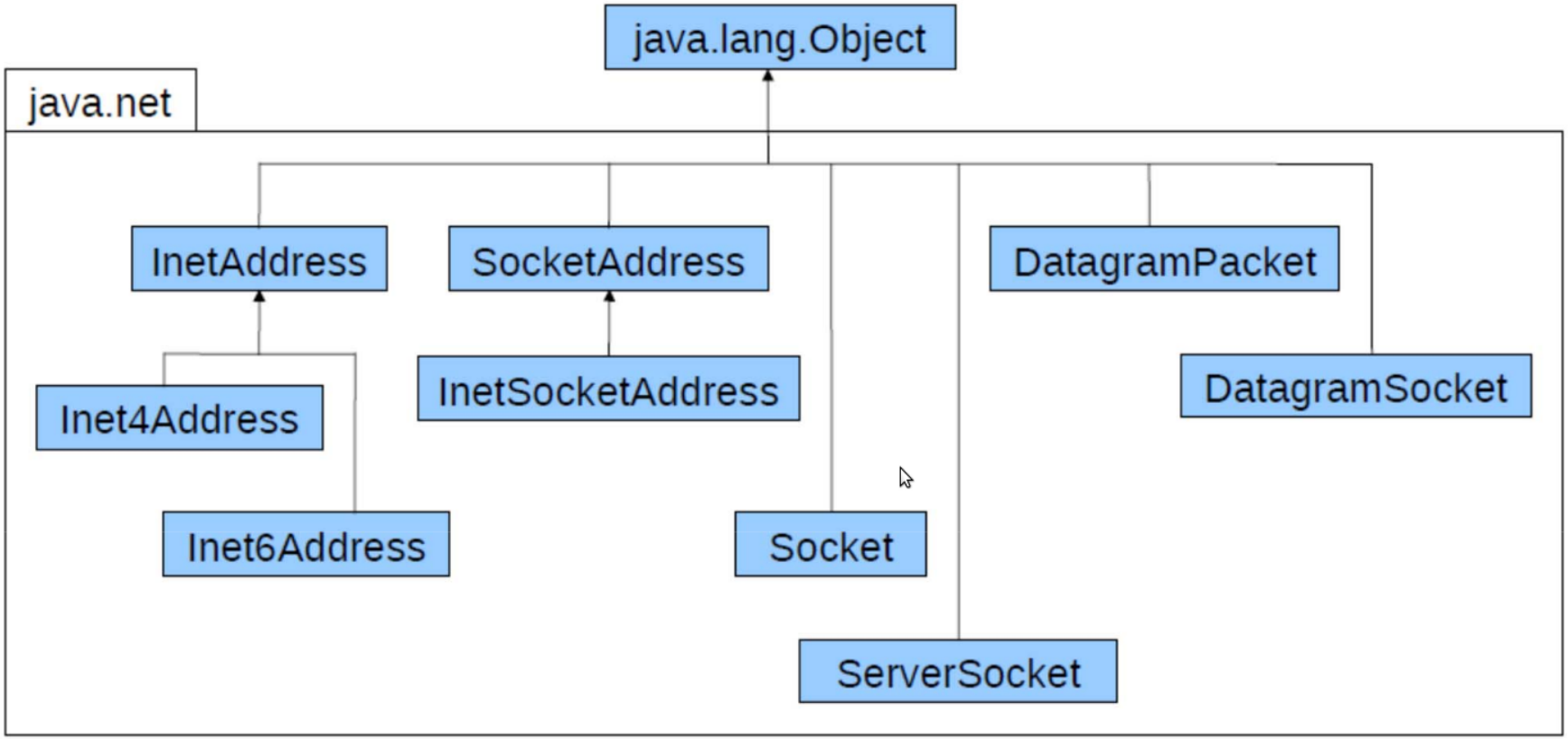
Sommaire

Packages Java à utiliser

- Java intègre nativement les fonctionnalités de communication réseau au dessus de la couche transport (TCP ou UDP) via le package java.net.
- Ce package est composé de 2 parties :
  - Une API de bas niveau, qui gère les adresses, les sockets (UDP et TCP) et les interfaces réseau.
  - Une API de haut niveau, qui gère les URL, URI et les connexions aux ressources pointées par les URLs.
- Le plus souvent, vu qu'il s'agit de fonctionnalités de type "fichiers", il faudra aussi utiliser le package java.io :

```
import java.net.*;
import java.io.*;
```

Diagramme de classes du package



Gestion des adresses

- Quel que soit le mode (UDP ou TCP), on utilisera la classe `InetAddress` qui contient l'adresse IP (IP4 ou IP6).
- Adresse locale :

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

Ou :

```
InetAddress addr = InetAddress.getLocalHost();
```

- On peut aussi utiliser les services de nommage de façon transparente (DNS ou résolution locale) :

```
InetAddress addr = InetAddress.getByName("univ-rouen.fr");
```

- Ou spécifier une adresse fixe :

```
InetAddress addr = InetAddress.getByName("45.46.47.48");
```

UNIVERSITÉ DE ROUEN

Classes Java pour sockets

- Classes utilisées pour socket UDP (client ou serveur)
  - `DatagramSocket` : socket mode non connecté (client ou serveur)
  - `DatagramPacket` : paquet de données envoyé ou reçu via un socket UDP
- Classes utilisées pour socket TCP
  - `Socket` : socket mode connecté (client ou serveur)
  - `ServerSocket` : socket d'attente de connexion du coté server
  - Transmission des données : deux approches
    - Mode "texte"
    - Mode "bloc d'octets"

UNIVERSITÉ DE ROUEN

Transfert de données sur socket TCP

Mode "texte" (lignes)

- Ex : protocole HTTP, FTP, SMTP
- Mise au point facilitée (un humain peut "voir" les données)
- Utilisation des classes `BufferedReader` et `BufferedWriter`

Mode "Bloc d'octets"

- Requêtes et réponses : blocs d'octets d'une taille et d'un format connus de l'autre (autre solution : taille est donnée dans une entête)
- Exemples : RPC, RMI, CORBA,...
- Utilisation des classes `DataInputStream` et `DataOutputStream`

Ces classes sont déclarées dans le package `java.io`

UNIVERSITÉ DE ROUEN

Gestion des exceptions

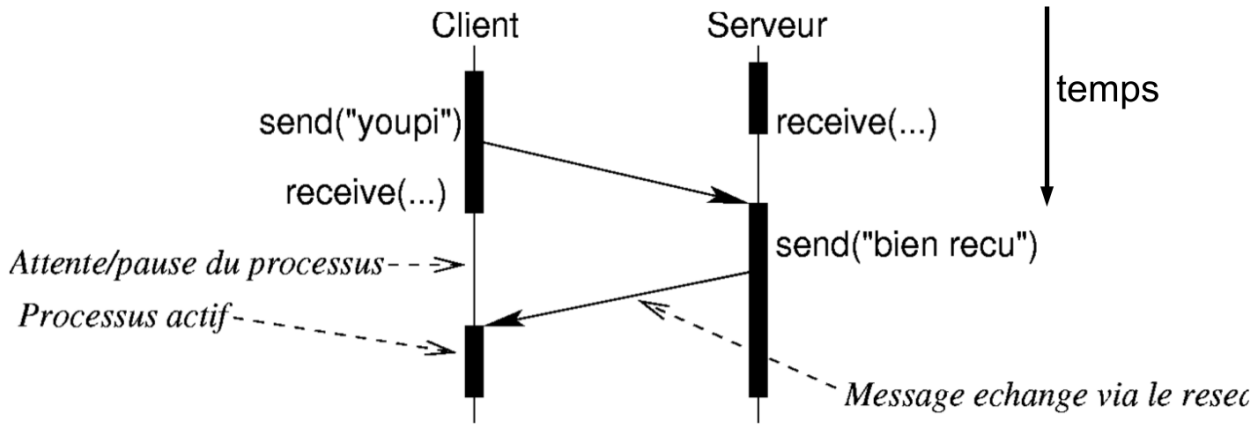
- Attention : par définition, la connexion à un autre processus via un socket peut échouer (problème sur la couche réseau).
- Donc, toute tentative d'ouverture d'un socket pourra échouer, Java lèvera alors une exception.
- Celle-ci devra être prévue par le programme, via un bloc `try... catch`
  Il faudra alors afficher (par exemple) un message dans la console, et afficher la pile des appels :

```
try {
    ... mon code
}
catch( IOException ex ) {
    System.out.println( "erreur I/O" );
    ex.printStackTrace();
}
```

- La granularité de gestion des exception est modulable, on peut se contenter de rattraper seulement l'exception "racine" `Exception`

Et si plusieurs clients ?

- Principe : les processus distants communiquants sont actifs en parallèle ("concurrents")
- En pause lors d'attente de messages
- Exemple de flux d'exécution :



- Problème : Tant que la connexion n'est pas fermée par le client, le serveur ne peut pas pas répondre à un autre client...

Plusieurs clients : solution

- Solution : *threads* (processus)
- Nouvelle connexion : un nouveau processus est créé pour gérer la communication avec le nouveau client.
- Exemple de flux d'exécution :