

Introduction aux Systèmes Répartis Client/Serveur

Julien Sopena

Julien.Sopena@lip6.fr

(basé sur un cours de **Gaël Thomas** et de **Lionel Seinturier**)

Université Pierre et Marie Curie

Master Informatique

M1 – Spécialité SAR

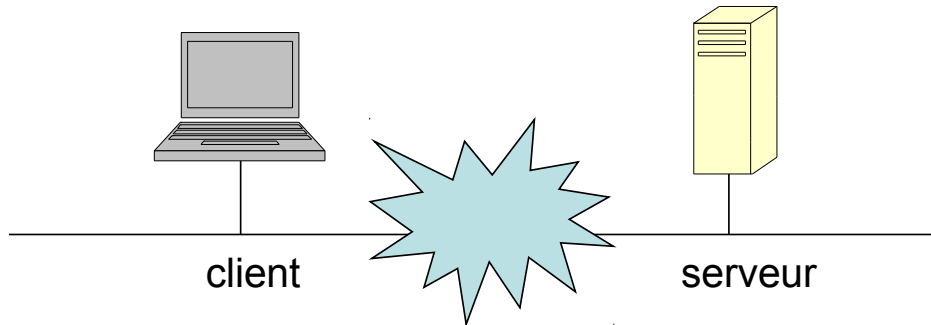
Introduction : Objectifs

- Concepts et notions de base pour le client/serveur :
 - Modèle de programmation
 - Services fournis
 - Architecture et mécanismes internes

- Étude de trois environnements :
 - API Java : programmation réseau et programmation concurrente
 - RMI (Remote Method Invocation) : appel de méthodes distantes
 - CORBA : objets répartis

- Notions étudiée : socket, thread, souche/squelette, IDL

Introduction : architecture client/serveur



Sépare l'utilisateur d'une fonctionnalité du fournisseur

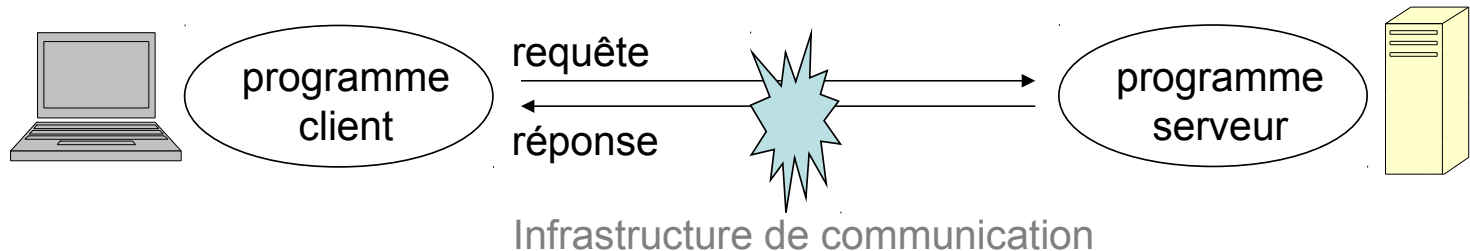
- Serveur : fournit une fonctionnalité
- Client : utilise cette fonctionnalité

Intérêt : sépare le client et le serveur et **fait apparaître le lien entre eux**

- Permet de répartir clients et serveurs sur plusieurs machines
- Permet différentes paradigmes de communication entre clients et serveurs

Introduction : paradigmes de communication (i)

Communication par requête/réponse (RPC, RMI, Corba, EJB...)



Interaction : 1 requête + 1 réponse

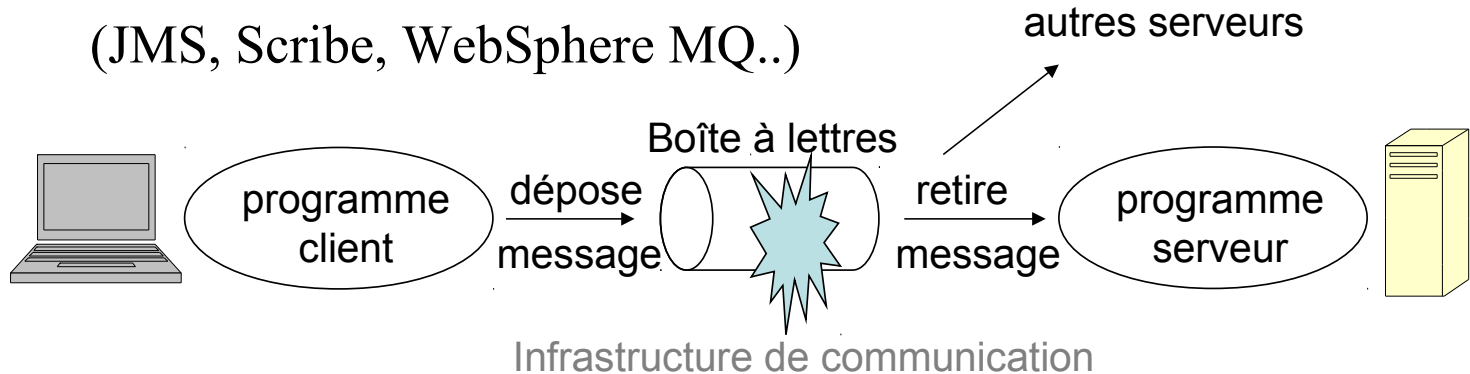
- Client : envoie une requête et attend la réponse
- Serveur : attend une requête, exécute un traitement et renvoie la réponse

Extension : appel de procédure en réparti

Introduction : paradigmes de communication (ii)

Communication par messages

(JMS, Scribe, WebSphere MQ..)



Interaction : par envoie de message

- Client : envoie un message
 - ✓ Asynchrone : n'attend pas la réception du message
 - ✓ Synchrone : attend la réception et le traitement du message
- Serveur : attend des messages et exécute un traitement

□ ≠ envoie de message par socket

Protocole applicatif, diffusion, synchrone/asynchrone + propriétés
(transaction)

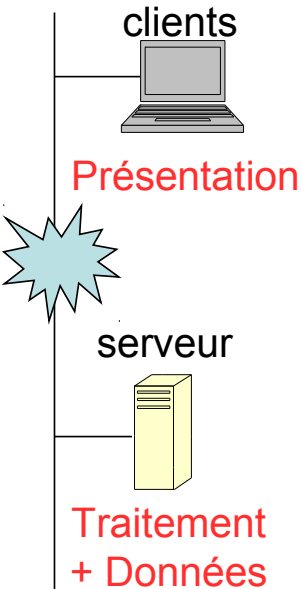
Introduction : besoins

- Architecture client/serveur invisible pour l'utilisateur final
 - L'utilisateur se sert d'un client comme de toute autre application...
- Gestion de l'hétérogénéité (Matériel, OS, langage)
 - Le client et le serveur doivent pouvoir être hétérogènes
- Gestion transparente des communications distantes pour les clients et les serveurs
 - Représentation des données, protocoles de communication
- Gestion de la concurrence transparente pour le client
 - Création, terminaison d'activités, synchronisation

Introduction : vers des architecture 3-tiers (i)

Problème : la séparation entre les préoccupations

- Données : manière de stocker des données
- Présentation : manière de représenter une donnée
- Traitement : code de traitement de données



■ Avantages

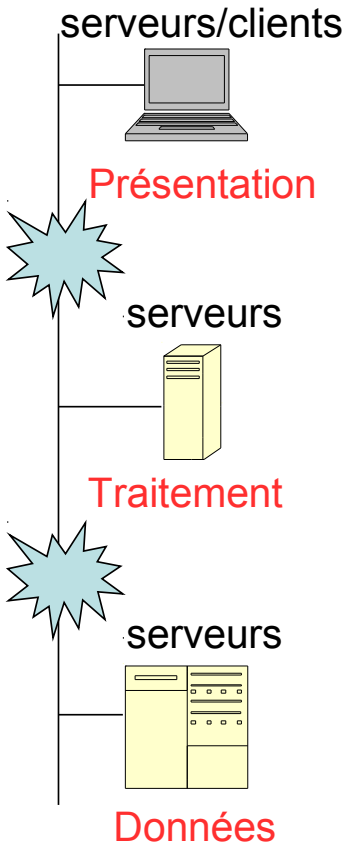
- Pas de duplication de données (centralisée sur le serveur)
- Gestion simple de la cohérence et de l'intégrité des données

■ Inconvénients

- Pas d'évolutivité (données et traitement sont liées)
- Souvent solution propriétaires non standardisées et chères
- Pas de répartition de charge

Introduction : vers des architecture 3-tiers (ii)

Architecture 3-tiers : sépare les préoccupations (voir module MDoc)



■ Avantages

- Meilleure répartition de la charge
- + évolutif (standard types J2EE existants)
- Économiquement moins cher

■ Inconvénients

- Administration plus difficile
- Mise en œuvre plus difficile
- Duplication des données (cohérence entre BD et traitement)