CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE REGIONAL MIDI-PYRENEES

Introduction aux systèmes répartis

Objectif de ce chapitre

- Définir les systèmes répartis
- Les situer dans le monde de l'informatique
- ~~Analyser leurs historiques et~~ Commenter leurs évolutions
- Préciser les domaines d'application des systèmes répartis
- Introduire les moyens de mise en œuvre des systèmes répartis
- Identifier les standards, les normes, et les logiciels mettant en œuvre les applications distribués

Définitions (1/2)

- Une première définition : "*un système réparti est un système informatique dans lequel les* ***ressources*** *ne sont pas* ***centralisées***". Ces ressources sont notamment :
  - les moyens de stockage (données, fichiers)
  - la charge CPU
  - les utilisateurs
  - les traitements …
- Systèmes répartis : le fruit du mariage entre l'informatique et les moyens de télécommunication (réseau)
- système réparti ⇔ système distribué
- Des systèmes distribués aux systèmes à agents mobiles (à large échelle dans un contexte réseau dynamique)
- Les calculs répartis : une autre dimension des systèmes répartis
- La mondialisation de l'informatique + Internet = répartir l'architecture d'un système d'information

Définitions (2/2)

- La problématique de la répartition est née avec l'idée de faire communiquer des ordinateurs via un réseau de communication
- Modélisation : les ressources (ordinateurs) sont les "nœuds" du modèle et la communication se fait par échange de message
- "nœud" ➔ serveur ➔ ressources ➔ protocole de répartition ➔ **Middleware**
- Les middlewares = moyens logiciels de mise en œuvre des applications distribués (RMI, CORBA, …)

Middleware (1/3)

- Middle-Software : Logiciel intermédiaire
- ou bus logiciel
- Communication inter-processus (IPC)
- Se situe au-dessus de la couche de transport (couches 5, 6 et 7)
- Missions du middle-ware:
  - la gestion des appels de fonctions de l'application ou la gestion du renvoi des résultats entre les clients et les serveurs
  - la mise en forme des données en vue de leur prise en charge par la couche transport
- Deux composants :
  - Le protocole d'accès formaté (*Format And Protocol*, FAP) met en forme les différentes données au niveau du réseau et définit le protocole d'échange des données
  - L'interface de programmation (*Application Programming Interface*, API) se charge :
    - des connexions et déconnexions avec le serveur;
    - de la définition de l'environnement de la connexion (variables de contexte, zones tampon); et
    - du transfert des requêtes et de la réception des résultats (n-uplet par n-uplet ou de façon globale)

Middleware (2/3)

- Exemples :
  - IAE (Intégration d'Application d'Entreprise): Gestion de Bussiness Object
  - CORBA : Norme d'architecture logicielle dont les composants sont distribués sur un bus logiciel (ou ORB)
  - ODBC : La technologie ODBC permet d'interfacer de façon standard une application à n'importe quel serveur de bases de données, pour peu que celui-ci possède un driver ODBC (la quasi-totalité des SGBD possèdent un tel pilote).
  - SOA : Architecture Orientée Service
  - WSOA : SOA basé sur des WebServices
- Les middlewares les plus en vogue dans les architectures dites *trois tiers* sont :
  - les middleware "orientés objets ou composants distribués" : ce sont les **ORB** ou Object Request Broker
  - les middleware "**transactionnels**" : ce sont les *moniteurs transactionnels* (comme CICS d'IBM, Tuxedo de BEA, MTS de Microsoft, JTS de Sun, TopEnd de NCR ou encore Jaguar de Sybase, ...)
  - les middleware "orientés messages" : ce sont les **MOM** (comme MQ Series d'IBM, JMS de Sun, MSMQ de Microsoft).
  - les middleware "SOA/WSOA" : J2EE, .NET, Apache Struts, ….

Ne pas confondre SOA et SOAP

- ne pas confondre SOA et SOAP = Simple Object Access Protocol
- SOAP est un protocole de RPC orienté objet bâti sur XML
- Il permet la transmission de messages entre objets distants
  - il autorise un objet à invoquer des méthodes d'objets physiquement situés sur un autre serveur
  - le transfert se fait le plus souvent à l'aide du protocole HTTP, mais peut également se faire par un autre protocole, comme SMTP.
- Il permet ainsi de définir des services WEB. Les paquets de données circulent sous forme de texte structuré au format XML (Extensible Markup Language).
- Microsoft a basé sur **SOAP** sa nouvelle architecture services Web **.NET**.
- Il existe des implémentations **Java**, et Borland vient déjà d'implémenter **SOAP** sous Windows dans **Delphi 6** et sous **Linux** avec **Kylix**.
- Les avantages :
  - l'interopérabilité, parce qu'il est indépendant des plate-formes et des languages de programmation.
  - le déploiement des applications, principalement dans un contexte multi-sîtes, pour communiquer entre 2 sociétes via internet, c'est souvent mission-impossible d'utiliser autre chose que du HTTP ou du POP/SMTP à cause des Firewalls, car pour les autres protocoles il faut les reconfigurer, avec tous les trous de sécurité que cela peut engendrer, et cela implique souvent de longue négociation **SOAP** (**S**imple **O**bject **A**ccess **P**rotocol) définit un protocole permettant des appels de procédures à distances (RPC) s'appuyant principalement sur le protocole HTTP et sur XML, mais aussi SMTP et POP. Il permet ainsi de définir des services WEB. Les paquets de données circulent sous forme de texte structuré au format XML (Extensible Markup Language).

Les domaines d'application des systèmes répartis

- Mathématique, algorithme, "puissance" de calcul
- Innovations (agents, collaboration, IA, …)
- Systèmes d'information d'entreprise
- Les particuliers : partager ses données, jeux, accès et collaboration de services communs, …
- Sites Internet : un site est un lieu de partage de l'information et de distribution de service
- et bien d'autres à venir ….
- Le monde des systèmes informatiques répartis et distribués sont en constantes évolutions : les réseaux évolues, les techniques logicielles sont en constantes révolutions, la prolifération du logiciel libre accentue cette évolution, les performances sont en constante croissance.

Les systèmes répartis et les réseaux

- Pas de réseau → pas de systèmes répartis
- Evolution des réseaux → évolutions des systèmes répartis
- L'évolution des réseaux ne suive pas nécessairement les besoins des systèmes répartis : exemple le réseau Internet qui n'est pas prévu pour faire de la répartition de l'information mais on s'adapte
- Les technologies comme les DNS, les LDAP appartiennent au monde des systèmes répartis
- ~~Les protocole réseau utilisés entraînent des contraintes sur la mise en œuvre des systèmes répartis~~
- L'Intelligence Artificielle utilise les principes algorithmiques des systèmes répartis dans les réseaux de neurones : répartir la connaissance et ses mécanismes et non les centraliser

Les concepts des systèmes répartis (1/2)

- L'évolution et les performances actuelles des réseaux informatiques permettent de mettre en pratique les concepts de la répartition dans des applications informatiques de tous les jours
- En quoi la programmation d'application répartis se distingue d'une application centralisée ➔ 2 hypothèses perdues et 2 propriétés affaiblies qu'il faut corriger :
  - PAS D'ETAT GLOBAL
  - PAS D'HORLOGE GLOBALE
  - FIABILITE RELATIVE
  - SECURITE RELATIVE
- Pas d'état global :
  - un nœud ne possède pas une connaissance immédiate exacte de l'état d'un autre nœud car cette connaissance passe par l'échange d'un message qui introduit obligatoirement un délai => recherche d'algorithme qui permettent de construire des clichés globaux qui représentent un état passé possible de l'application
- Pas d'horloge global :
  - horloge propre à chaque nœud (ordinateur)
  - pas de synchronisation des horloges => l'ordre des évènements répartis dans l'application n'est pas déductible à partir des datations locales => définir des datations logiques

Les concepts des systèmes répartis (2/2)

- Fiabilité relative :
  - deux éléments contradictoires :
    - tolérance à la panne d'un nœud par des moyens de réplication et de délocalisation
    - maîtriser et rendre robuste de telles architectures demandent de gros efforts informatiques qui ne sont pas souvent à la hauteur
  - le risque de défaillance d'un nœud augmente avec leur nombre mais rend non exceptionnel un tel évènement
- Sécurité relative :
  - difficulté de protéger un architecture répartie contre les intrusions
  - les points d'accès aux ressources sont souvent "hors des murs"
  - demande donc une authentification
  - les nœuds peuvent être dynamique (de-re-localisation) => architecture réseau variable donc difficile à protéger
  - apparition massif des ordinateurs portables accroît cette fragilité
- Non déterminisme
  - une application réparti est une application asynchrone : les nœuds s'exécutent en parallèle et chaque nœud peut lui-même comporter plusieurs activités parallèles (processus, threads)
  - il existe ainsi un non déterminisme, une non reproductibilité et une explosion combinatoire des états (difficultés classiques de la programmation parallèle asynchrone: jetons, "synchronized", …)

Les avantages des systèmes répartis (1/2)

- Partage et Mise à disposition
  - partager des ressources et des services disponibles (ex: les systèmes d'exploitation répartis qui permettent de mettre en place un service de gestion de fichiers partagés et répartis)
- Répartition géographique
  - mettre à disposition des usagers les moyens informatiques locaux en même temps que ceux distants de leurs collègues (ex: un système de réservation d'hôtel répartis en différents pays, compagnies ou agences; autre ex: un système bancaire avec ses agences régionales et son siège social)
- Puissance de calcul :
  - paralléliser les algorithmes de calcul avec des environnements d'exécution spécifique comme PVM (Parallel Virtual Machine) ou MPI (Message Passing Interface)
- Disponibilité d'un service
  - continuer un service globalement même dégradé
  - exemple courant: la réplication d'un même service par l'installation de plusieurs serveurs équivalents (attention si les serveurs sont à états rémanents => réplication)

Les avantages des systèmes répartis (2/2)

- Flexibilité
  - par nature modulaire
  - continuité de service pendant la maintenance (remplacement d'un nœud)
  - l'informatique nomade : portable et points d'accès mobiles sur un réseau réparti aux frontières floues (Internet)
  - les problèmes posés sont :
    - la localisation
    - l'identification
    - l'authentification
    - l'optimisation des temps de connexion
- Adaptabilité à une forte croissance des besoins informatiques d'une entreprise
- => une meilleure définition :
  - **La répartition est la mise à disposition d'un ensemble de ressources et services connectés via un réseau pour tous les usagers qui possèdent un droit d'accès en un point quelconque.**

L'approche des systèmes répartis (1/2)

- par la Théorie
  - modélisation formel, sémantique
  - modèle de calcul traduisant de façon abstraite et formelle les propriétés d'un calcul réparti
    - algèbre de processus communicant
    - protocole de communication entre acteurs communicant par envoi de message
- par l'algorithmie : pour pallier aux 2 hypothèses "perdues"
  - définition de protocole point à point ou à diffusion (automates, Pétri, langage LOTOS, …) dont le standard : l'appel procédural à distance
  - appliqués au temps réel
  - influence du monde parallèle (exclusion mutuelle, interblocage, atomicité, réplication, …)
  - problèmes nouveaux (ex: le pb de terminaison d'une application répartie, le calcul d'états globaux, la réalisation d'un consensus

L'approche des systèmes répartis (2/2)

- par les langages de programmation
  - développement d'une API permettant d'échanger des messages (sockets par ex)
  - apparition de langages spécifiques (ex: OCCAM) ou extension de langage existant (ex: ADA)
  - la possibilité de faire des appels procéduraux à distance :
    - pas de changement dans le modèle de programmation
    - modèle aujourd'hui le plus utilisé
    - est un modèle client-serveur
    - définition d'un IDL, stub, skeleton
- par les systèmes d'exploitation

  2 approches :
  - création de nouveaux noyaux dédiés à la répartition (il faut repartir à 0) (ex: le système CHORUS) : les micro-noyaux répartis
  - étendre les systèmes d'exploitation centralisés :
    - ajout de système de gestion de fichiers répartis (NFS, …)
    - ajout de moyen de communication (sockets, RPC, …)
- par l'apparition de environnements d'exécution répartie (middleware)
  - maîtriser l'hétérogénéité matérielle et logicielle
  - basé sur le schéma de communication client/serveur
  - basé sur la notion de "bus logiciel" : accès à des services spécifiés par leur interface → utilisation d'annuaire permettant de trouver les nœuds serveurs (ex CORBA)

Principes de conception (1/9) : introduction

- **Un bon système réparti est un système qui "à l'air centralisé" (paradoxe)**
- Afin de simplifier l'effort de programmation dans une application répartie, il vaut mieux masquer le plus possible les propriétés délicates de la répartition
- Mais, il est impossible d'être transparent sur tous les aspects
- Les propriétés de transparence :
  - transparence d'accès
  - transparence de localisation
  - transparence du partage
  - transparence de la réplication
  - transparence des fautes
  - transparence de la migration
  - transparence de charge
  - transparence d'échelle

Principes de conception (2/9) : transparence d'accès

- l'interface d'accès à une ressource ne doit pas dépendre que la ressource soit locale ou distante
- la syntaxe utilisée ne doit pas être différente
- très souvent difficile à mettre en place en fonction de la nature de la ressource
- exemple: l'accès un fichier que le fichier soit local ou distant grâce à la transparence obtenue par l'utilisation de NFS
- exemple : l'accès à l'interface d'un objet distribué (méthode distante)

Principes de conception (3/9) : transparence de localisation

- la désignation de la ressource est indépendante de la localisation de cette ressource
- nécessite un service de nommage global (annuaire)
- exemple : utilisation des "Naming services" de la norme CORBA pour les objets distribués
- nécessite une connaissance de l'architecture du réseau

Principes de conception (4/9) : transparence du partage

- les accès concurrents à la ressource sont contrôlés de telle façon que l'intégrité de la ressource est assurée
- synchronisation des accès en lecture et écriture de la ressource
- nécessite de décrire les traitements informatiques sous la forme de "transaction" dont l'atomicité garantit l'intégrité de la données
  - exemple des transactions dans des bases de données réparties
  - exemple de l'enchaînement des méthodes synchronisées d'un objet distribué

Principes de conception (5/9) :transparence de la réplication

- consiste à s'assurer que l'accès à une ressource soit identique quel que soit la forme d'implantation d'une ressource répliquée
- utilisé pour augmenter la tolérance aux fautes
- l'accès à une ressource répliquée est indépendante de l'indisponibilité d'une de ses occurrences
- la mise en œuvre de la réplication est un exercice difficile. Mais facilité par l'usage de mécanismes de réplication intégrés comme ceux des SGBD
- la réplication permet un routage des transactions (lecture / écriture)

Principes de conception (6/9) : transparence des fautes

- détection de la défaillance des nœuds du système réparti
- pallier à cette défaillance d'une manière transparente
- tolérance aux fautes de l'ensemble des services d'un système réparti
- ne pas bloquer le fonctionnement global de l'application
- la réplication permet une transparence de l'indisponibilité d'une ressource et/ou d'un service

Principes de conception (7/9) : transparence de la migration

- la propriété de migration consiste à assurer qu'une ressource peut migrer d'un noeud à un autre sans que les usagers s'en aperçoive.
- déplacer un service dynamiquement vers un serveur moins chargé
- mettre en œuvre des stratégies de régulation de charge sur une architecture répartie

Principes de conception (8/9) : transparence de charge

- la régulation de la charge de chaque noeud peut permettre une meilleure exploitation du système global et une meilleure satisfaction des utilisateurs.
- la mise en oeuvre est délicate car une régulation de la charge globale implique une bonne connaissance de l'état global du système (difficile à obtenir)
- nécessite la mise en place d'algorithmes répartis spécifiques
- nécessite une bonne connaissance des contraintes du réseau utilisé

Principes de conception (9/9) : transparence d'échelle

- un changement d'échelle d'une application réparti consiste à augmenter le nombre de nœud
- ceci est facilité par la propriété modulaire et dynamique d'une architecture répartie
- l'ajout de noeuds ne doit pas nécessiter l’arrêt du système
- ce changement d'échelle n’est pas forcement transparent pour les usagers, il faut donc le contrôler et le rendre le plus transparent possible.

Exemple : la programmation d'agents informatiques (1/3)

Le CONTEXTE :

- Un agent est un objet possédant un cycle de vie (naît, vit, meurt)
  - exemples : un agent de supervision, un jeu de la vie, une simulation comportemental, …
- Chaque agent est un nœud du réseau (logique)
  - c'est à dire que chaque agent est un objet distribué et donc utilise, si besoin, tous les objets distribués, du réseau.
- Le comportement de chaque Agent peut être donc dépendant de celui des autres agents :
  - exemple pour des agents comportemental : il se meut dans un espace sur lequel les autres agents rentrent en conflit et/ou en collaboration avec lui.

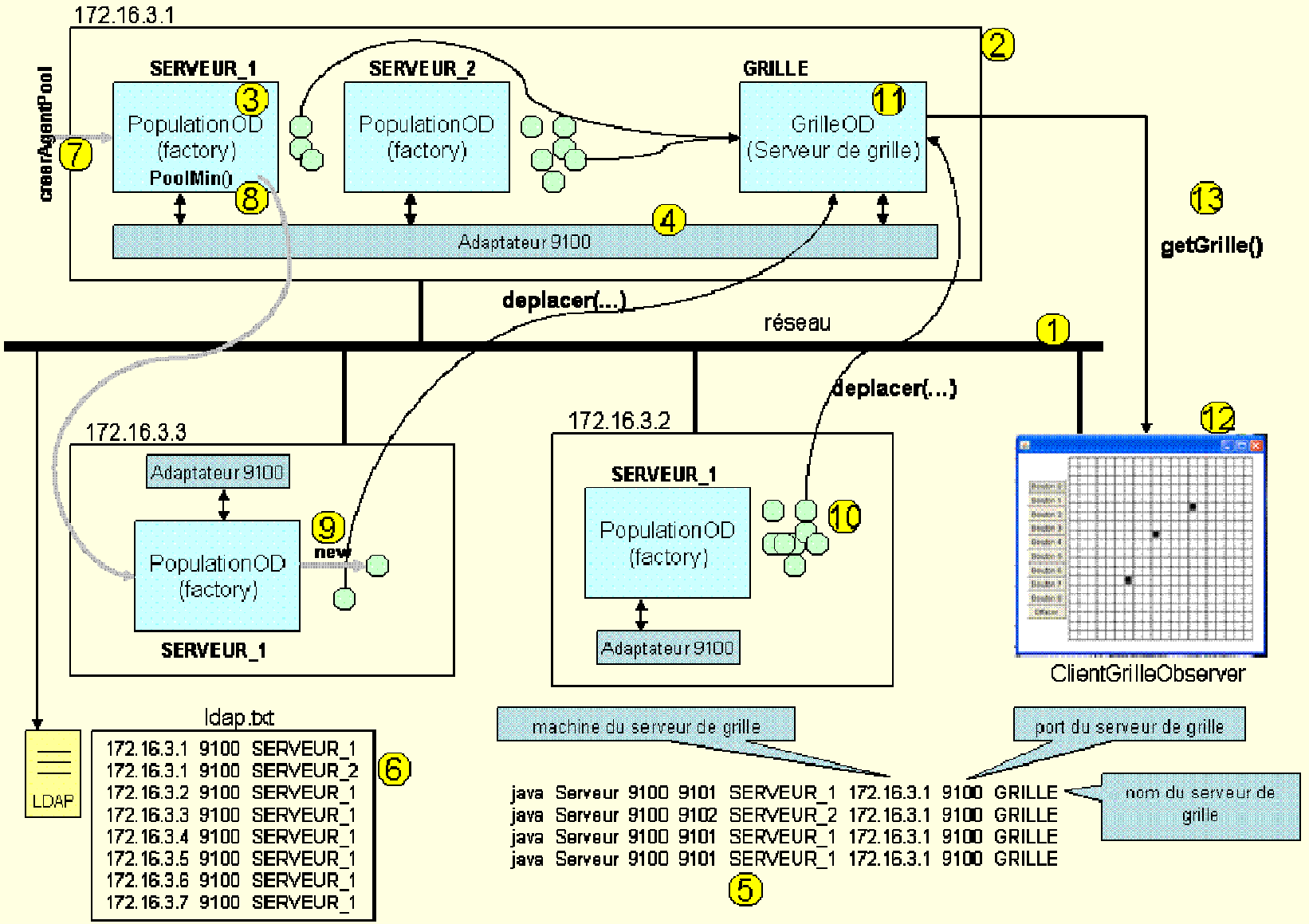
Exemple : la programmation d'agents informatiques (2/3)

- Le PROBLEME : gérer l'espace commun à tous les agents
- Solution 1 : répartition par accès distant à un chef d'orchestre
  - création d'un nœud particulier qui contient l'espace
  - synchronisation de déplacement
  - si indisponibilité de ce nœud alors indisponibilité de l'ensemble ou partie de l'application
  - chaque opération nécessite un appel distant au chef d'orchestre
- Solution 2 : répartition par image
  - tous les agents s'observent les uns les autres
  - chaque agent peut obtenir l'image de tous les autres
  - chaque image est mis à jour sur changement de la source (réplication)
  - disponibilité car même si un des agents est indisponible les autres peuvent fonctionner (l'image est bien sûr rémanente). Cela nécessite une persistance de chaque agent en cas de reprise

Exemple : la programmation d'agents informatiques (3/3)

- Solution 3 : répartition par circulation
  - faire circuler un message (jeton) entre les agents contenant l'espace
  - avoir le jeton => avoir le droit de le modifier
  - augmentation de la bande passante du réseau à cause de la circulation du jeton sur la boucle locale des noeuds du réseau
  - défaillance en cas de perte du jeton
  - défaillance d'un nœud => le jeton ne circule plus. A moins d'avoir une connaissance globale des nœuds

Architecture de la solution 1 (1/5)



Architecture de la solution 1 (2/5)

- 1 : Le réseau utilisé est un réseau classique, en TCP/IP, sur lequel sont répartis plusieurs machines référencées par leurs adresses IP.
- 2 : Sur chaque machine on exécute un adaptateur RMI qui sert de service de nommage pour l'ensemble des objets distribués qui s'exécutent sur la machine. On exécute 1 ou plusieurs objets distribués de Population (ou appelé « serveur de population »).
- 3 : Un serveur de population est donc un objet distribué RMI qui s'enregistre dans l'adaptateur RMI (9100) de la machine et qui attend que l'on utilise ses méthodes distantes. Il utilise le fichier LDAP pour connaître tous les serveurs de population actifs du réseau (dont lui-même).
- 4 : Le service de nommage utilisé ici est un adaptateur RMI qui s'exécute sur chaque machine, sur le port 9100. En RMI-IIOP ou CORBA, il n'y aurait qu'un seul service de nommage exécutée sur une machine en particulier.

Architecture de la solution 1 (3/5)

- 5 : On exécute les serveurs via une commande d'exécution Java qui prend en entrée :
  - *le port de l'adaptateur RMI dans lequel le serveur doit s'enregistrer*
  - *le port de traitement de requête du serveur de population et de tous ces agents, qui est imposé afin de correspondre aux contraintes du Firewall du réseau*
  - le nom du serveur de population. *Si plusieurs serveurs de population sont exécutés sur la même machine, ils doivent avoir des noms différents puisqu'ils s'enregistrent dans le même adaptateur.*
  - *l'adresse IP du serveur de grille, le port de l'adaptateur dans lequel s'est enregistré le serveur de grille*, le nom du serveur de grille

    (en italique ➔ RMI, souligné ➔ Corba)
- 6 : Afin que tous les serveurs de population connaissent tous les autres serveurs de population, le fichier LDAP contient les coordonnées RMI de tous les serveurs potentiels du réseau.
  - On a les informations suivantes dans le fichier : l'adresse IP de la machine sur lequel le serveur s'exécute, le port de l'adaptateur dans lequel il s'enregistre, le nom du serveur. Ce fichier LDAP est unique sur le réseau et accessible sur un montage commun (NFS).
  - Ce fichier LDAP est remplacé par le Naming Services en CORBA si réseau local. Indispensable si plusieurs réseau inter-connecté

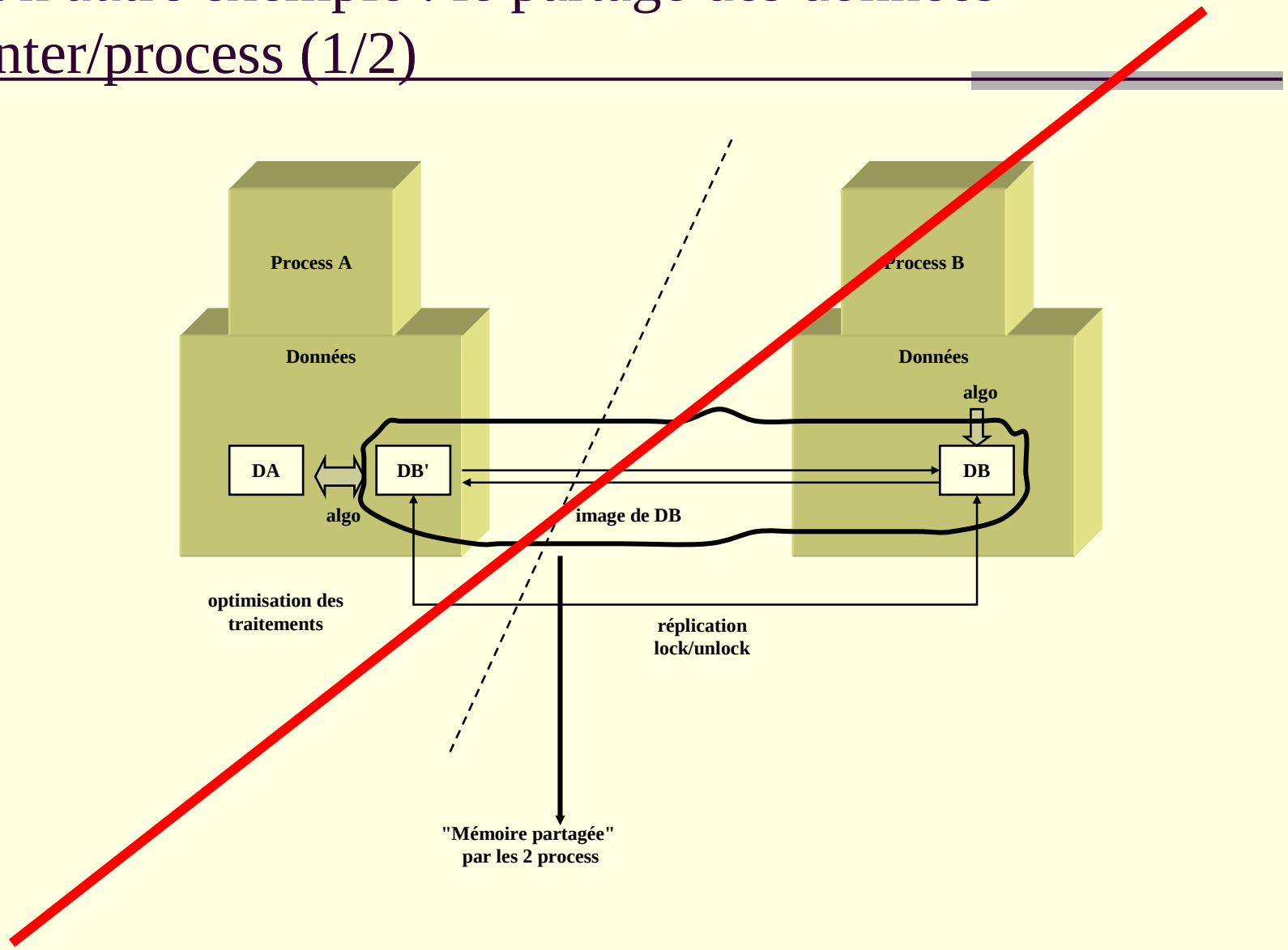
Architecture de la solution 1 (4/5)

- 7 : Un client qui est un programme quelconque s'exécutant sur n'importe quelle machine du réseau, appelle la méthode distante d'un serveur actif afin de lui demander de créer des agents.
- 8 : Alors, le serveur de population, pour chaque agent devant être créé, interroge tous les serveurs de population pour déterminer celui qui possède le moins d'agent en mémoire. Il lui délègue alors la création de l'agent.
- 9 : Le serveur de population le moins chargé crée alors le nouvel agent sous la forme d'un objet distribué.
- 10 : Tous les agents utilisent la méthode distante du serveur de grille pour essayer de se déplacer. C'est le serveur de grille qui décide si le déplacement est possible ou non en fonction de la position des autres agents dans la grille.

Architecture de la solution 1 (5/5)

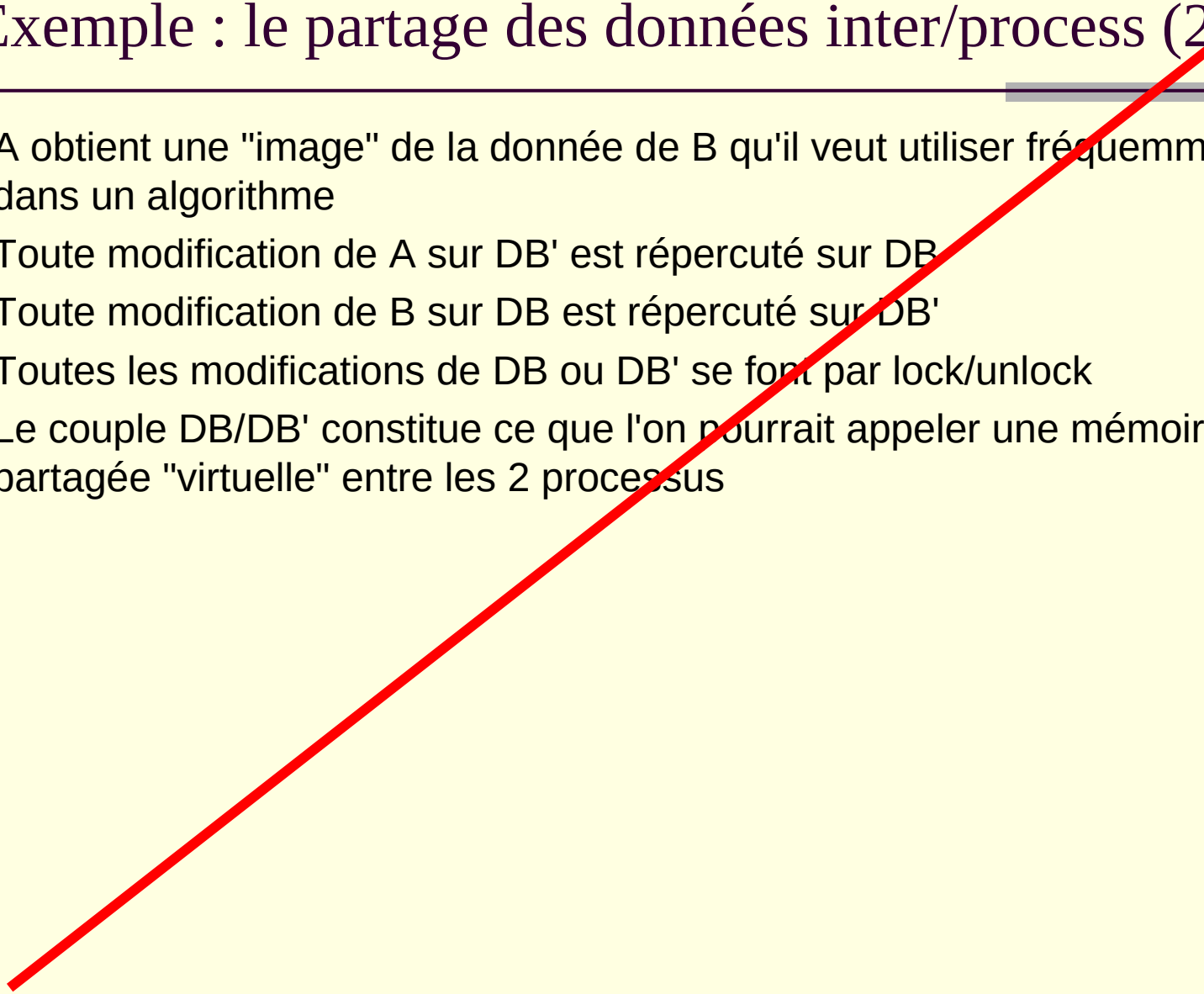
- 11 : Le serveur de grille est un objet distribué (ou « serveur de grille ») qui s'exécute sur une machine du réseau. Il contient une grille qui est partagé par tous les agents du réseau. La méthode distante deplacer est synchrone afin que chaque déplacement de chaque agent se fasse l'un après l'autre, car le déplacement doit tenir compte de la position de tous les autres afin d'autoriser ou non le déplacement.
- 12 : Pour pouvoir visualiser la grille du serveur de grille qui n'est qu'un tableau dans la mémoire du serveur de grille, on crée un client IHM dont le rôle est de demander, cycliquement, au serveur de grille, le contenu de sa grille afin d'afficher la position de tous les agents.
- 13 : Pour cela, le client IHM utilise une méthode distante du serveur de grille qui lui permet d'obtenir un état globale du serveur de grille.

Un autre exemple : le partage des données inter/process (1/2)



Exemple : le partage des données inter/process (2/3)

- A obtient une "image" de la donnée de B qu'il veut utiliser fréquemment dans un algorithme
- Toute modification de A sur DB' est répercuté sur DB
- Toute modification de B sur DB est répercuté sur DB'
- Toutes les modifications de DB ou DB' se font par lock/unlock
- Le couple DB/DB' constitue ce que l'on pourrait appeler une mémoire partagée "virtuelle" entre les 2 processus

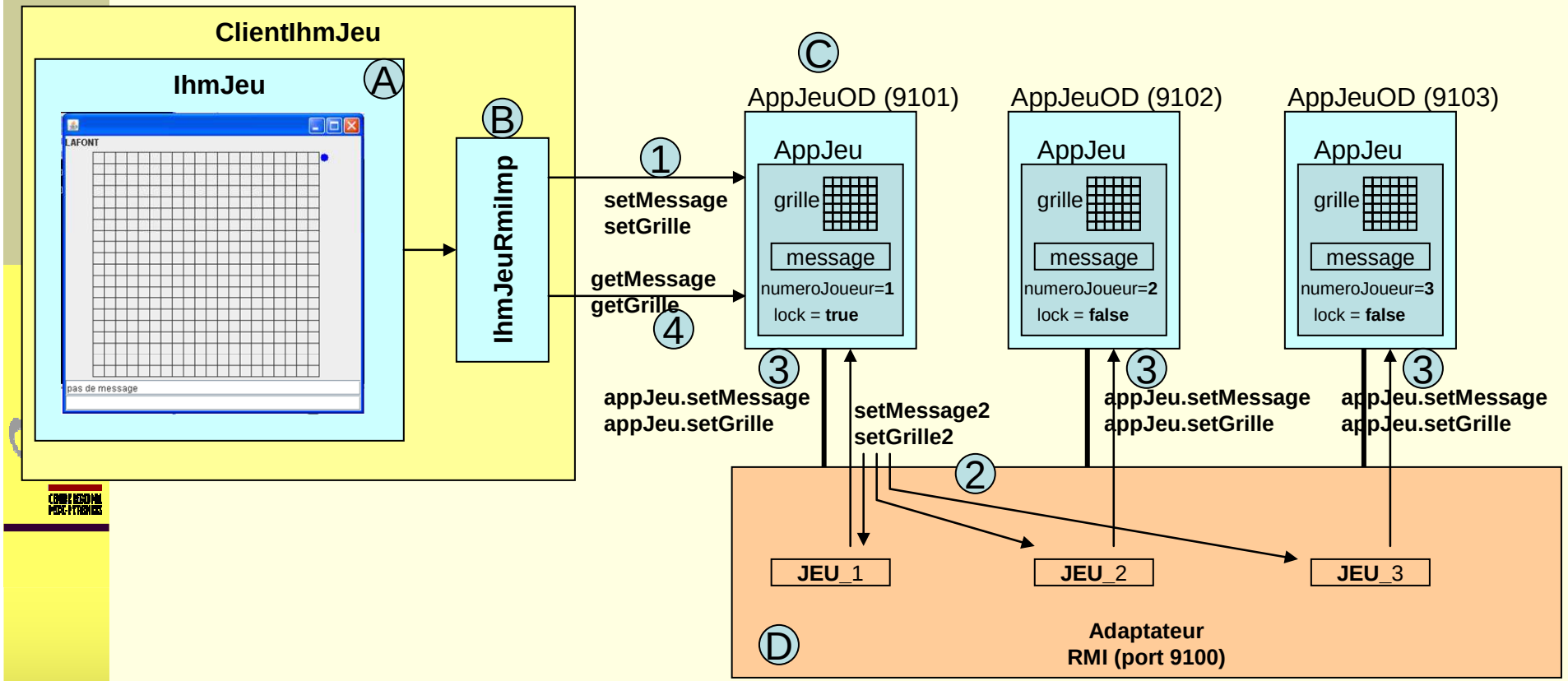


Exemple 2 : la propagation des états (1/3)

Le CONTEXTE :

- Les nœuds du réseau contiennent de l'information communes mais répartis
- exemple :
  - Un joueur gère une grille de jeu et un message de communication…
  - Chaque joueur modifie sa grille de jeu et/ou son message
  - Tous les joueurs connectés sont mises à jour par propagation des setteurs de l'objet (nœud du réseau) : la grille et le message
  - Similaire à un lien de réplication

Exemple : la propagation des états (2/3)



Exemple 2 : la propagation des états (3/3)

- A : le client d'un joueur
- B : le middleware
- C : le nœud du réseau : un joueur = états de jeu
- D : le service de nommage

- 1/ via le client l'état de jeu d'un joueur évolue : valeurs de la grille et valeur du message
- 2/ via le service de nommage : propagation des setteurs
- 3/ changement des états de jeu de tous les joueurs : appel aux setteurs
- 4/ le client récupère les états de jeu : appel aux getteurs

Les deux modèles des systèmes répartis(1/2)

- modèle des processus communicant par envoi de message
  - modèle fondateur (modèle standard)
  - basé sur le parallélisme des communications
  - communication synchrone ou asynchrone
  - communication point à point ou par diffusion
  - modèle CSP (Communicating Sequentiel Processes) et CCS (Calculus of Communicating Systems)
    (http://wwwsi.supelec.fr/fb/enseignement/3A/modelisation/Modelisation.pdf)
- modèles fondés sur la notion d'objet
  - très répandus par l'usage des technologies objets dont les LOO
  - masquer le phénomène de communication (et donc de répartition)
  - les traitements : procédure accessible à distance (RPC)
  - les données : mémoires partagées (ex: les objets)
  - difficulté : éviter une trop forte synchronisation des accès à ces mémoires partagées
  - compromis entre une sémantique "centralisée" et des performances acceptables

Les modèles de la répartition (2/2)

- Ces modèles permettent tous deux la généralisation et l'amélioration du schéma client/serveur
  - serveurs multiples (équilibrage de charge, redondance)
  - systèmes multi-couches (tiers)
  - peer to peer
- Les problèmes à faire face
  - l'interblocage
  - la terminaison stable (oscillation et interblocage)
  - calcul d'un état global (prise de cliché)
  - le problème du consensus : les entités doivent s'accorder sur une valeur unique après avoir fait chacun une proposition

Exemples de modèle d'architecture répartie

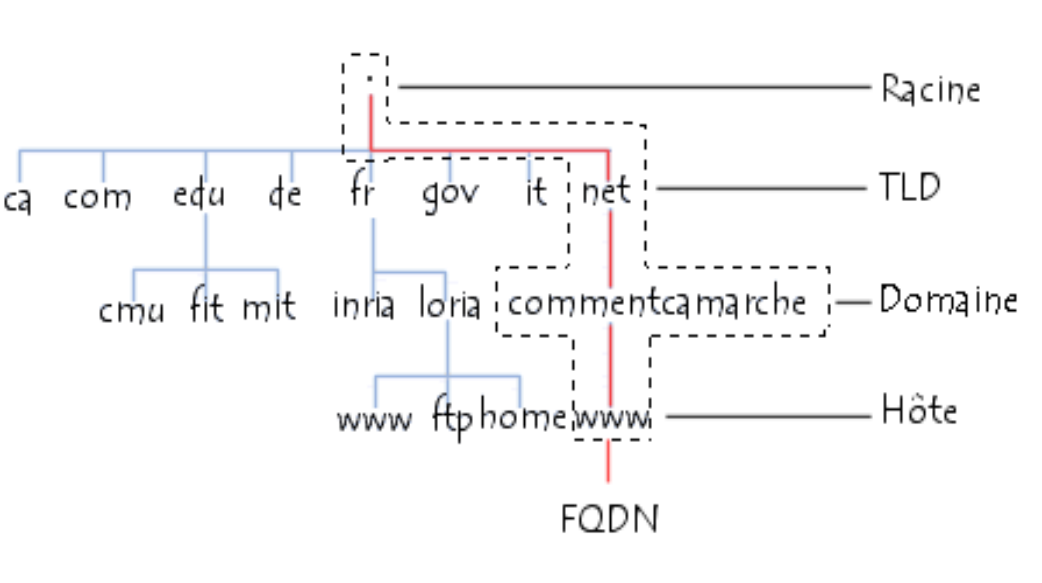
DNS	Base de données répartie
Annuaire X500	Base de données répartie
Web	Ordinateur réparti : données, traitements, …
Systèmes multi-couches	architecture classique pour les systèmes mélangeant serveur web, base de données, traitement évolué, etc
Peer to peer	système complètement décentralisé, en général pour le stockage Réparti
Clusters	calcul réparti
Grid	super-ordinateur pour le calcul et le stockage réparti

DNS : architecture logique (1/2)

- DNS = Domain Name System
- http://www.commentcamarche.net/internet/dns.php3
- permet de trouver l'@IP associée à un nom de machine
- système hiérarchique : arbre de nœud (la racine est vide)
- association nœud ⇔ enregistrements
- un domaine est un sous-arbre
- le nom d'un domaine est le nom d'un nœud
- le nom complet d'accès d'un domaine = chemin dans l'arbre = FQDN (Full Qualified Domain Name)

DNS : architecture logique (2/2)

- l'espace de nom est structuré : TLD (Top Level Domains)
- un DNS est une base de données répartie contenant des enregistrements, appelés RR (Resource Records)



DNS : architecture physique

- un système de serveurs distribués (les serveurs de noms)
- basé sur un système de délégation et de réplication :
  - un serveur DNS gère un domaine
  - chaque domaine est géré par au moins 2 serveurs
  - le protocole DNS propose un mécanisme de réplication (un serveur maître et des esclaves)
  - le gestionnaire d'un domaine peut déléguer la gestion d'un de ses sous domaines à un autre gestionnaire
- les requêtes utilisent un système de cache local (TTL) avec délégation de requête (principe de base de la répartition) : durée de vie limité
- Le système de nom est une architecture distribuée, où chaque entité est responsable de la gestion de son nom de domaine. Il n'existe donc pas d'organisme ayant à charge la gestion de l'ensemble des noms de domaines.

DNS : Résolution du nom de domaine

- la requête est envoyé au serveur DNS primaire configuré (auto dans le fournisseur d'accès à Internet)
- si dans cache alors retourne l'addr IP sinon redirige la requête au serveur de nom racine
- Le serveur de nom racine renvoie une liste de serveurs de noms faisant autorité sur le domaine
- Effet de cascade
- Attribution des noms de domaine par des organismes centraux (InterNIC, AFNIC, …)
- La modification s'accompagne d'un mise à jour ou obsolescence des caches

Annuaires X500

- annuaires basés sur les mêmes principes que celui du DNS
- plus récent et plus complets (enregistrements, attributs privés, héritage, ..)
- Exemple d'application : LDAPv3 (Lightweight Directory Access Protocol)

Serveurs WEB

- protocole utilise : HTTP
- intrinsèquement réparti à cause des liens (par exemple, toutes les requêtes cgi peuvent être exécutée sur un serveur différent du serveur principal)
- solution de base de type client/serveur
- traitement sur le client (Javascript, Java, Flash, SVG, etc.) → répartition des calculs entre client et serveur
- équilibrage de charge (*load balancing*)
- les requêtes arrivent sur un serveur principal. Elles sont relayées (mécanisme de *proxy*) sur plusieurs serveurs effectifs
- programmes sur le serveur : cgi, servlet, langages de script serveur (JSP, PHP, ASP, etc.) → base des architectures multi niveaux

Systèmes multi-couches

- 1ère couche : Présentation : tout ce qui permet l'interaction avec l'utilisateur
  - réalisée par des extensions du serveur : cgi (C,C++,Perl), servlet, JSP, ASP, PHP, …
  - réalisée par des ihms déportées (applets, portlet)
- 2ème couche : Application (*Business* ou Métier) : toute la logique applicative (vérification des entrées de l'utilisateur, etc.)
  - accès direct au SGBD : gestion de pools de connexion, ODBC, JDBC, interface Perl ou PHP d'accès
  - accès masqués au SGBD : persistance (Hibernate)
- A 2 couches :
  - classique, séparation de la BD, la couche métier est appelée par la couche de présentation
- A 3 couches et plus :
  - approche distribuée des services, serveurs dédiées
  - passerelle HTTP de communication

Le peer-to-peer (1/2)

- Une autre façon de répartir des données et des services
- architecture point à point, adapté à l'envoi de message
- robuste par définition : pas de serveur pas de client pour communiquer entre les participants (tout le monde peut devenir serveur)
- il existe quand même des serveurs (≠ "peer-to-peer pur")
  - gère les index documents/@IP_cliente
  - optimise les recherches dans la répartition en centralisant
  - améliore la bande passante utilisée
- Exemples :
  - Napster (1 serveur central)
  - Edonkey, Emule (plusieurs serveurs)
  - Freenet et overnet (pas de serveur)

Le peer-to-peer (2/2)

- Dans le cas serveur : protocole en 2 étapes
  - demande au serveur les clients possédant un document
  - établi une communication client/client pour télécharger le document
  - morceau par morceau, indisponibilité d'un client ➔ routage vers un autre client (indispo / réplication)
- Dans le cas sans serveur : protocole de propagation
  - demande à un "ami" (client privilégié) un document si il ne peut pas ou pour équilibré alors routage vers un autre client
  - la réponse suit le chemin inverse

Les Clusters et les Grids

- Définition : un regroupement de deux serveurs ou plus, en vue de créer un "super serveur virtuel"
- Ensemble de plusieurs machines vues comme une seule permettant d'obtenir de grandes puissances de traitement
- SMP : Système Multi-Processeur Symétriques
- but: augmente la puissance de calcul
- Les technos :
  - processeurs / cache : répartis, plusieurs µP par machine, et plusieurs machines
  - mémoire partagée (mais pas répartie)
  - threads
  - connexion à hauts débits
- Notion de cluster logiciel : Lotus (Cluster Notes)
- Programmation parallèle: pas de vrai standard
- Les Grids : beaucoup d'ordinateur (ex: Seti), trop nouveau, pas de norme, "à la frontière du réel"

La communication dans les systèmes répartis

- la communication doit être transparente dans les systèmes répartis
- 4 niveaux hiérarchique de couche de communication :
  - envoi de message
  - appel de fonction
  - appel d'une méthode
  - appel d'un service
- L'envoi de message se fait par un protocole basé sur la communication socket
- L'appel d'une fonction se fait par l'utilisation d'une API de type RPC
- L'appel d'une méthode se fait par l'utilisation d'un "bus logiciel" (ORB)
- L'appel d'un service se fait par l'utilisation d'une architecture 3-Tiers de type "Web Services"

La communication socket

- communication bi-directionnelle entre 2 programmes, multi-machines
- pas de transparence : host + port
- nécessaire de créer son propre protocole de communication
- pas d'interopérabilité simple
- basés sur TCP et UDP : les service SMTP, FTP, HTTP
- les systèmes répartis basés sur TCP et UDP
  - DNS
  - LDAP
  - Edonkey, freenet, ….
- il faut tout faire (serveur, API, transparence, interopérable, …)

La représentation des données

- la représentation interne des données ne doit pas dépendre de la machine
- niveau structuré et hiérarchique (historique: XDR) langage de description des données
- évolution vers l'objet à travers le langage de description XML
- l'aboutissement avec SOAP (Simple Object Access Protocol : une évolution de XML-RPC définie par le W3

L'appel de fonctions à distance (RPC)

- XML RPC
- SOAP (Simple Object Access Protocol)
- transformation de l'appel à une méthode sous la forme d'un message
- transformation du message reçu en l'appel à une fonction
- la réponse se fait également sous la forme de l'envoi d'un message
  - les mécanismes mises en oeuvre :
    - la souche (ou stub)
    - le squelette (ou skeleton)
    - l'empaquetage (marshalling) et le dépaquetage (unmarshhalling) permettant l'encodage et décodage des paramètres (sérialisation)

L'appel d'une méthode (RMI)

- par définition: orientée objet
- les méthodes sont des méthodes distantes associées à un objet distribué (ou servant)
- on parle d'objets distants décrits par une interface (IDL)
- plusieurs grands standards :
  - CORBA (Common Object Request Broker)
  - Java RMI
  - Microsoft COM+/DCOM (Distributed Component Object Model) : à utiliser que dans des environnements 100% Microsoft

Java RMI (1/2) : Introduction

- RMI = Remote Methode Invocation
- packages Java appartenant au JDK de base (java.rmi, java.rmi.server, …)
- solution pure Java : pas d'interopérabilité avec d'autres langages
- la mise en œuvre est facile et efficace
- les composants :
  - côté serveur : la classe UnicastRemoteObject
  - côté client : l'interface qui hérite de Remote
  - le compilateur rmic : crée les amorces et squelettes
  - le service de nom : le registre RMI (LocateRegistry)
  - le passage des paramètres : la sérialisation (Serializable)
- tout objet Java peut devenir un objet distribué sur le réseau :
  - encapsulation
  - interface des méthodes distantes
- créer un objet distribué à distance : créer un factory

Java RMI (2/2) : La conception

- la conception d'une application distribuée est une conception objet :
  - un objet distribué est un serveur qui implémente l'interface de l'objet distant et qui hérite de UnicastRemoteObject
  - un objet client est une interfaces de l'objet distant
  - les méthodes distantes sont des méthodes de l'objet distant
  - les paramètres des méthodes distantes sont des types primitifs ou des objets (prédéfinis ou non) → grande souplesse

Java RMI : Exemple (1/3)

```
import java.rmi.*;
public interface HelloInterface
extends Remote {
/* méthode qui imprime un messageprédéfini
dans l'objet appelé */
public String sayHello ()
                throws java.rmi.RemoteException;
}
```

Objet distant

```
import java.rmi.*;
import java.rmi.server.*;
public class Hello extends java.rmi.server.UnicastRemoteObject implements
HelloInterface {
private String message;
/* le constructeur */
public Hello (String s) throws RemoteException
{
  message = s ;
} ;
/* l'implémentation de la méthode */
public String sayHello () throws RemoteException
{
  return message ;
} ;
}
```

Java RMI : Exemple (2/3)

côté client

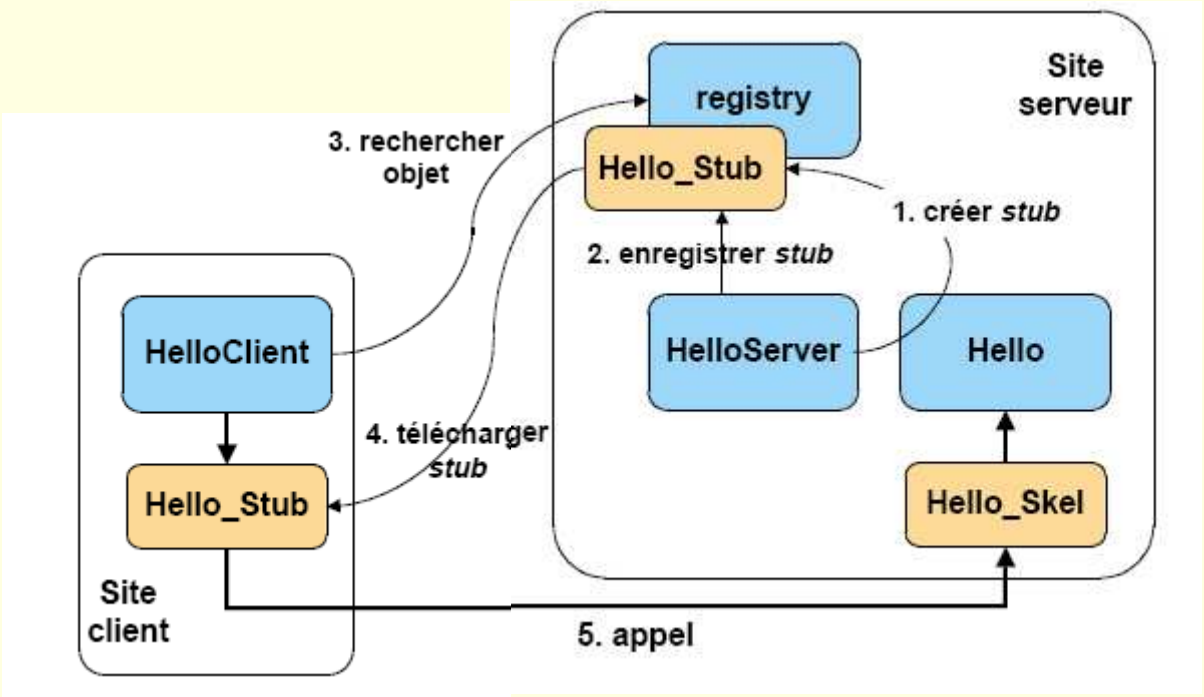
```
import java.rmi.*;
public class HelloClient {
public static void main (String [ ] argv) {

  /* lancer SecurityManager */
  System.setSecurityManager (new
RMISecurityManager ()) ;
  try {

/* trouver une référence vers l'objet distant */
 HelloInterface hello = (HelloInterface)
Naming.lookup("rmi://goedel.imag.fr/Hello1") ;

 /* appel de méthode à distance */
 System.out.println (hello.sayHello()) ;

} catch (Exception e) {
System.out.println
("Erreur client : " + e) ;
}}}
```

côté serveur

```
import java.rmi.*;
public class HelloServer {
public static void main (String [ ] argv) {

/* lancer SecurityManager */
System.setSecurityManager (new
RMISecurityManager ()) ;

try {

/* créer une instance de la classe Hello et
l'enregistrer dans le serveur de noms */
Naming.rebind ("Hello1",new Hello ("Hello
world !")) ;

System.out.println ("Serveur prêt.") ;

} catch (Exception e) {
System.out.println
("Erreur serveur : " + e) ;
}}}
```

Java RMI : Exemple (3/3) / L'exécution

- **Lancer le serveur de noms (sur la machine serveur) ou l'exécuter dans le main du serveur (rmiregistry *port* &, LocateRegistry.createRegistry(*port*))**
  - **par défaut, le registry écoute sur le port 1099**
  - **si port alors URL doit être rmi://<serveur>:<port>/<name>**
- **Lancer le serveur**
  - **java -Djava.rmi.server.codebase=http://goedel.imag.fr/<répertoire des classes> -Djava.security.policy=java.policy HelloServer &**
  - **Le contenu du fichier java.policy spécifie la politique de sécurité**
  - **L'URL donnée par codebase sert au chargement de classes par le client**
- **Lancer le client**
  - **java -Djava.security.policy=java.policy HelloClient**
  - **Le talon client sera chargé par le client depuis le site du serveur, spécifié dans l'option codebase lors du lancement du serveur**



CORBA (1/5) : Introduction

- CORBA = Common Object Request Broker (Architecture d'objets communs pour échanger des requêtes) Commun = Appartient à tout le monde
- CORBA est une norme (spécification, cahier de charge)
- Des producteurs de logiciel applique cette norme pour créer des environnements de programmation permettant de bâtir des applications distribuées (appelé ORB)
- La spécification est large : tous les ORB n'implémentent pas toutes les spécifications de la norme
- Il y a interopérabilité entre les différents ORB qui implémentent la norme CORBA mais il peut apparaître certaines incompatibilités
- Permet aux applications d'interagir quelque soit leur localisation, leur langage et leur machine
- Maintenant, on rencontre CORBA dans tous les environnements et pour tous les langages de programmation
- Il existe même des environnements comme Weblogic Websphere qui cachent l'utilisation de CORBA

CORBA (2/5) : Définition

- le "bus logiciel" d'un ORB représente sa couche de communication : IIOP (Internet Inter Orb Protocol)
- l'ORB se charge de localiser le serveur sur lequel un servant s'exécute : Naming Services
- le principe de la communication est l'appel de méthode distante
- les paramètres échangés ne sont pas définis sous la forme d'objet comme RMI mais sous la forme de structures typées définies dans une IDL
- l'IDL (Interface Definition Language) est un langage commun à tous les ORB permettant d'exprimer les interfaces des servants
- l'ORB s'occupe de générer les éléments nécessaires à assurer la communication entre le serveur et le client

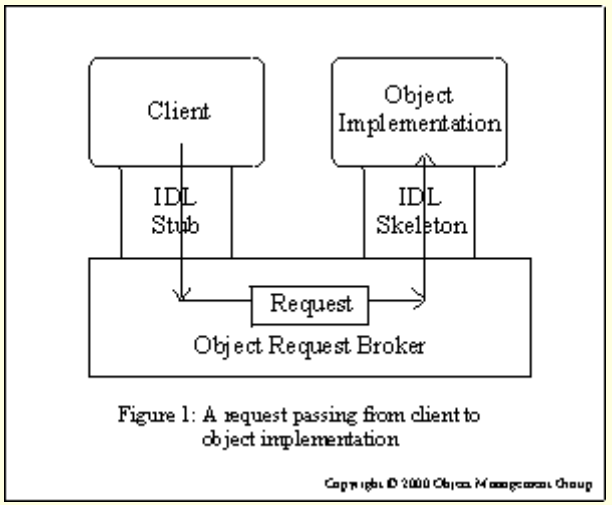


Figure 1: A request passing from client to object implementation

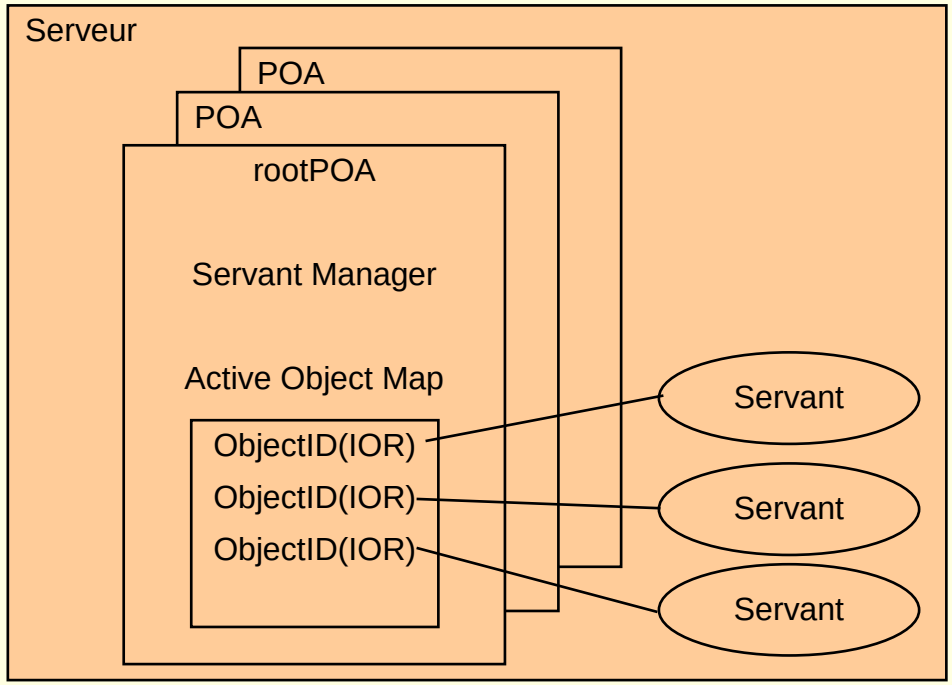
CORBA (3/5) : La hiérarchie

- la hiérarchie des éléments et concepts de CORBA

  Serveur → rootPOA → POA_i → $servant_j$
- Un serveur peut mettre à disposition plusieurs servants. Il utilise pour cela une objet CORBA appelé Portable Object Adaptater

Le fonctionnement:

1) Initialisation de l'ORB
2) Création d'un POA
3) Activation d'un manager de POA
4) Activation des servants
5) Mise en attente de requêtes clientes



CORBA (4/5) : Les services (1/1)

- **Naming Services** : permet d'associer un ou plusieurs noms logiques avec une référence objet (IOR) et de stocker ces noms dans un espace de stockage
  - Tout servant à un IOR (Interoperable Object Reference)
- **Event Services :** découple la communication entre les objets
  - est basé sur un modèle de communication fournisseur-client qui utilise un objet intermédiaire appelé Event Channel (canal d'événements
  - permet à plusieurs fournisseurs de transférer des données de manière asynchrone à plusieurs clients
  - modèles de communication push et pull
- **Persistence** : stockage et restauration des objets
- **Transaction** : garantir l'intégrité des différents objets du système quels que soient les problèmes survenant en cours de traitement
- **Concurrency :** gère les problèmes de concurrence d'accès à des ressources partagées

CORBA (5/5) : Les services (2/2)

- **Query :** requêtes d'accès aux caractéristiques des objets par un autre objet
- **Collection** : outils pour parcourir une collection d'objets grâce à des itérateurs.
- **Properties** : ajout dynamique et temporaire d'une propriété supplémentaire à un objet.
- **Security** : authentification des clients, cryptage des données, ...
- **Time** : synchroniser les clients et le serveur.
- **Trader** *(Vendeur)* : localiser des objets en fonction de critères d'interrogation précis
- …etc… en constance évolution en fonction de la norme

Les ORB du marché

- **Visibroker** de Borland CORBA 3.0 Java, C++, …
- **Orbix** de IONA CORBA 2.0 C++
- **TAO** (open source) CORBA 2.0 C++
- **OmniBroker** (open source) CORBA 2.0
- **Jacorb** (open source)
- …..

- Matrices de correspondances des ORB en fonction
  - langage et concepts
    - http://www.jetpen.com/~ben/corba/orbmatrix.html
  - services
    - http://www.jetpen.com/~ben/corba/cosmatrix.html
  - platforms
    - http://www.jetpen.com/~ben/corba/platmatrix.html

Les Web Services (1/5)

- Un **service Web** est un ensemble de protocoles et de normes informatiques utilisés pour échanger des données entre les applications.
- Les logiciels écrits dans divers langages de programmation et sur diverses plateformes peuvent employer des services Web pour échanger des données à travers des réseaux informatiques comme **Internet**.
- Cette interopérabilité est due à l'utilisation de normes ouvertes regroupées au sein du terme générique de **SOA** (*Service Oriented Architecture* ou Architecture orientée services).

Les WEB SERVICES (2/5)

- Les standards employés
  - *Web Services Protocol Stack* : Les services Web se composent d'une collection de standards que l'on regroupe sous ce terme.
  - XML : Toutes les données à échanger sont formatées en XML.
  - Protocoles communs : Des données en XML peuvent être transportées entre les applications en utilisant des protocoles communs tels que HTTP, FTP, SMTP ....
  - WSDL : L'interface publique au service Web est décrite par ce protocole en cours de normalisation. C'est une description XML qui décrit la façon de communiquer en utilisant le service Web.
  - UDDI (Universal Description Discovery and Integration) : Le service Web est connu sur le réseau au moyen de ce protocole. Il permet à des applications de rechercher le service web dont elles ont besoin. L'annuaire UDDI est consultable de différentes manières :
    - ***Les pages blanches*** comprennent la liste des entreprises ainsi que des informations associées à ces dernières. Nous y retrouvons donc des informations comme le nom de l'entreprise, ses coordonnées, la description de l'entreprise mais également l'ensemble des ses identifiants.
    - ***Les pages jaunes*** recensent les services web de chacune des entreprises sous le standard WSDL.
    - ***Les pages vertes*** fournissent des informations techniques précises sur les services fournis. Ces informations concernent les descriptions de services et d'information de liaison ou encore les processus métiers associés.

Les WEB SERVICES (3/5)

- Avantages des services webs
  - Les services Web fournissent l'interopérabilité entre divers logiciels fonctionnant sur diverses plateformes.
  - Les services Web utilisent des standards et protocoles ouverts.
  - Les protocoles et les formats de données sont au format texte dans la mesure du possible, facilitant ainsi la compréhension du fonctionnement global des échanges.
  - Basés sur le protocole HTTP, les services Web peuvent fonctionner au travers de nombreux *firewalls* sans nécessiter des changements sur les règles de filtrage.
- Inconvénients des services webs
  - Les normes de services Web dans les domaines de la sécurité et des transactions sont actuellement inexistantes ou toujours dans leur petite enfance comparée à des normes ouvertes plus mûres de l'informatique répartie telles que CORBA.
  - Les services Web souffrent de performances faibles comparée à d'autres approches de l'informatique répartie telles que le RMI, CORBA, ou *DCOM*.
  - Par l'utilisation du protocole HTTP, les services Web peuvent contourner les mesures de sécurité mises en place au travers des *firewalls*.

Les WEB SERVICES (4/5)

- Les raisons de créer des services Web
  - Les services Web implémentent de la logique **métier** rendue consommable (on consomme un service web => utiliser) par l'utilisation de standards et de pouvoir l'exploiter, facilitant ainsi l'interopérabilité des applications.
  - La création de services Web se justifie par l'architecture **orientée service**, c'est à dire la volonté de rendre accessible un service qui implémente une logique métier cachée à des utilisateurs.
  - Dans le cadre de contrats d'échange de données en Business to Business (B2B), comme en Business to Consumer (B2C), un autre intérêt pour lequel des services Web sont employés est le fait qu'ils se fondent sur le protocole HTTP (qui utilise le **port 80** par défaut). Pour comprendre ceci, gardez à l'esprit que beaucoup d'entreprises se sont protégées en employant des ***firewalls*** qui filtrent et bloquent beaucoup de trafic d'Internet pour des raisons de sécurité. Dans ce milieu, beaucoup de (presque tous les) ports sont fermés au trafic entrant et sortant et les administrateurs de ces firewalls ne sont pas désireux de les ouvrir. Le port 80, cependant, est toujours ouvert parce qu'il est employé par le protocole HTTP utilisé par les navigateurs Web.

Les WEB SERVICES (5/5)

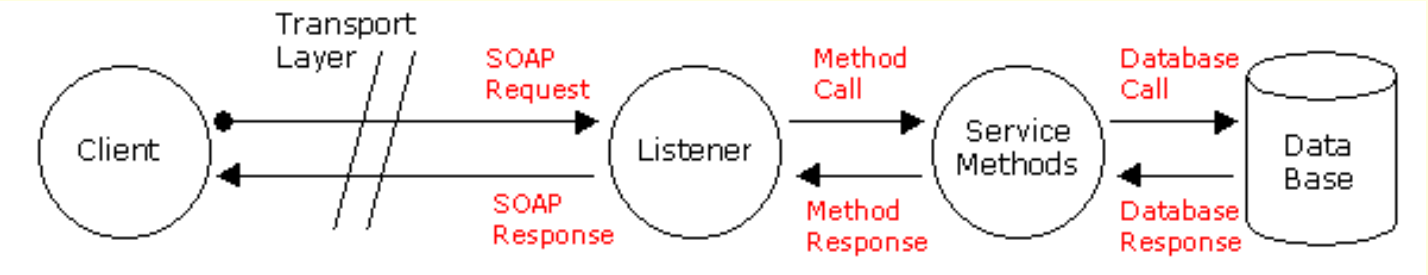
- Des services Web peuvent être déployés en employant un logiciel de serveur d'application :
  - Axis et le serveur de Jakarta Tomcat (open source d'Apache Software Foundation)
  - *XFire* de CodeHaus offre Java avec une approche différente de Axis
  - ColdFusion MX de Macromedia
  - Bibliothèque pour les développeurs de services Web en Java (JWSDP) de Sun Microsystems (basé sur Jakarta Tomcat)
  - Serveurs HTTP IIS (Internet Information Services) est un serveur HTTP et/ou FTP créé par Microsoft pour ses systèmes d'exploitation Windows (avec .NETWebLogic de BEA
  - WebSphere Application Server d'IBM (basé sur le serveur d'Apache et la plateforme de J2EE)
  - Oracle Application Serveur d'Oracle Corporation
  - ZenWorks de Novell
  - Bibliothèque pour les développeurs de services Web en PHP *NuSOAP*
  - JBoss Application Server de la société JBoss. Composant du JEMS (JBoss Enterprise Middleware System) dont fait également partie le framework de persistance relationnelle Hibernate.

SOAP (1/3)

- **Simple Object Access Protocol** (SOAP) est un protocole de RPC orienté objet bâti sur XML
- Il permet la transmission de messages entre objets distants, ce qui veut dire qu'il autorise un objet à invoquer des méthodes d'objets physiquement situés sur une autre machine. Le transfert se fait le plus souvent à l'aide du protocole HTTP, mais peut également se faire par un autre protocole, comme SMTP.
- Le protocole SOAP est composé de deux parties :
  - une enveloppe, contenant des informations sur le message lui-même afin de permettre son acheminement et son traitement,
  - un modèle de données, définissant le format du message, c'est-à-dire les informations à transmettre.
- SOAP a été initialement défini par Microsoft et IBM, mais est devenu depuis une recommandation du W3C, utilisée notamment dans le cadre d'architectures de type **SOA** (*Service Oriented Architecture*) pour les Web Services.

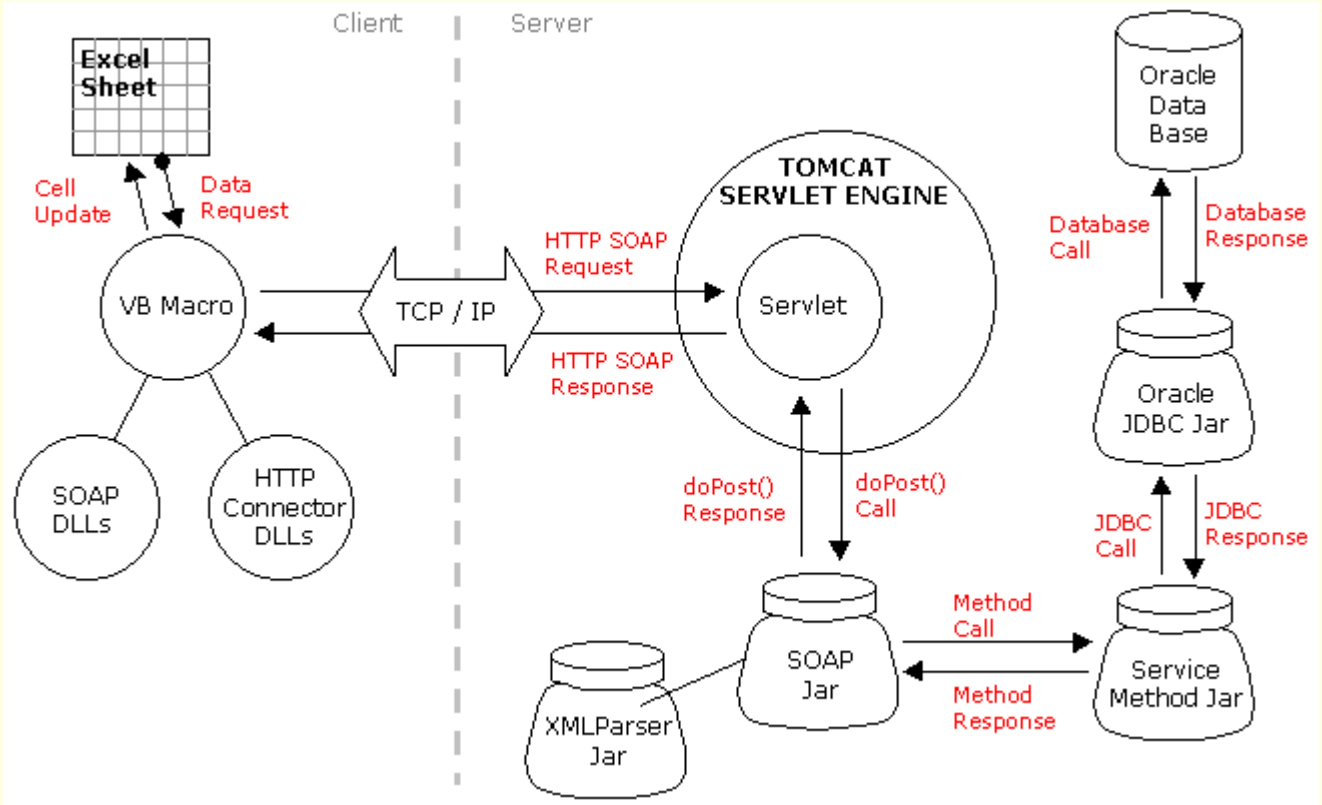
SOAP (2/3)

- protocole de transmission de message
- permet l'exécution des dialogues requête-réponse RPC
- s'appuie notamment sur le protocole HTTP
- indépendant de l'OS et du langage de programmation
- composant de base pour développer des applications distribués qui exploitent des fonctionnalités publiées comme services par des intranets ou internet.
- le protocole est basé sur XML (meilleur visibilité de l'encodage des requêtes)



SOAP (3/3)

- utilisé dans des architectures 3-Tiers



AJAX : Asynchronous JavaScript and Xml

- Asynchronous JavaScript and XML est une méthode informatique de développement d'applications Web.
- AJAX n'est pas une technologie en elle-même, mais un terme qui évoque l'utilisation conjointe d'un ensemble de technologies couramment utilisées sur le Web :
  - HTML (ou XHTML) pour la structure sémantique des informations ;
  - CSS pour la présentation des informations ;
  - DOM et JavaScript pour afficher et interagir dynamiquement avec l'information présentée ;
  - l'objet XMLHttpRequest pour échanger et manipuler les données de manière asynchrone avec le serveur web.
    - **XMLHttpRequest** est un objet Javascript qui permet d'obtenir des données au format XML, mais aussi HTML, ou encore texte simple à l'aide de requêtes HTTP.
  - XML et XSLT (langage de transformation XML de type fonctionnel)