Modèles et Systèmes de programmation distribuée

Nhan LE THANH

Cours LpSIL – Spécialité IDEE
Octobre 2004

Plan du cours

I. États de l'art
1- Objectifs, caractéristiques
2- Communication en réseaux
3- Structures logiques
4- Problématique
5- Développement, déploiement

II. Modèles d'exécution
1- Modèles client-serveur
2- Modèles de communication par messages
3- Modèles de communication par évènements
4- Autres modèles

III. Systèmes pair à pair
1- Principes et composantes
2- Calcul distribué Pair à Pair
3- Échange de données Pair à Pair
4- Étude de cas
5- Problématique et évolution

IV. Systèmes transactionnels
1- Transactions et transactions réparties
2- Systèmes transactionnels
3- Programmation transactionnelle
4- SGBD répartis

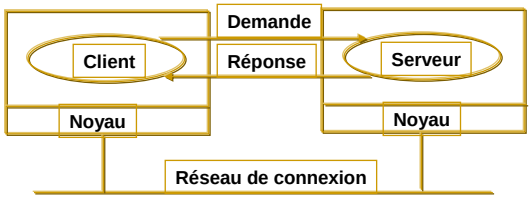
PART 2 : Modèles d'exécution

Plan

1- Modèles client-serveur
2- Modèles de communication par messages
3- Modèles de communication par évènements
4- Autres modèles

II. Modèles d'exécution
1. Modèle Client-Serveur

- Principes



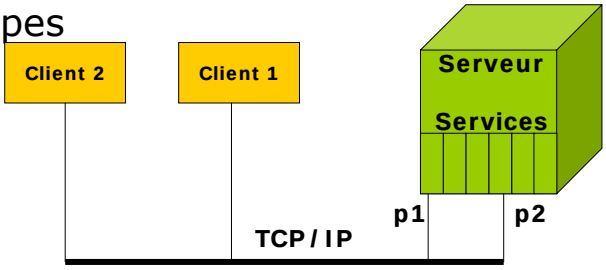
- Protocole simple sans connexion de type Question / Réponse

- Avantage : simplicité - efficacité

- Noyau : assurer l'envoi et réception des messages avec 2 primitives :
- send (dest, &ptrm)
- receive (adr, &prtm) (adr : l'adresse d'écoute du récepteur)

II. Modèles d'exécution
1. Modèle Client-Serveur

- Principes



Client :

- émet des requêtes de demandes de service
- mode d'exécution synchrone
-le client est l'initiateur du
-dialogue client - serveur

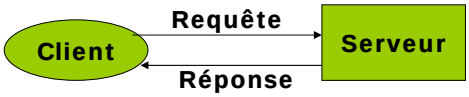
Serveur :

-ensemble de services exécutables
-exécute des requêtes émises par des clients
-exécution (séquentielle ou parallèle) des services

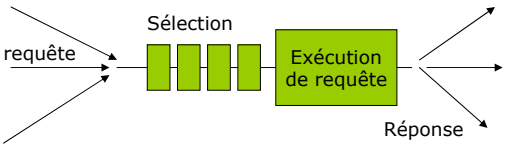
II. Modèles d'exécution
1. Modèle Client-Serveur

- Principes

Client :
-Interface d'utilisateur
-Saisie de requête
-Envoie de requête
- attente du résultat
- affiche du résultat



Serveur :
- Gestion des requêtes (priorité)
- Exécution du service (séquentiel, concurrent)
- Mémorisation ou non de l'état du client

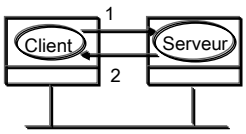


Deux modèles d'exécution :
- Client-Serveur à primitives : SEND et RECEIVE
- Client-Serveur à RPC (Remote Procedure Call)

II. Modèles d'exécution
1. Modèle Client-Serveur

- Modèle à primitifves : adressage

Localisation connue

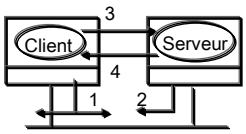


1 : demande vers S.p1
2 : réponse vers C.p2

- tout client doit connaître la localisation et le nom interne de tous les services. Il demande donc explicitement sous format :

machine.processus

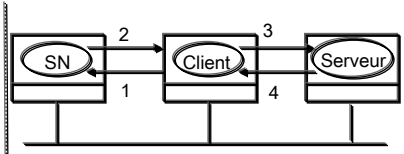
Localisation inconnue



1 : recherche par diffusion
2 : je suis là
3 : demande
4 : réponse

- chaque processus choisit un nom aléatoire telle sorte qu'il soit unique
- client diffuse une demande de localisation d'un processus
- le serveur contenant ce processus donne son adresse machine

Localisation centralisée

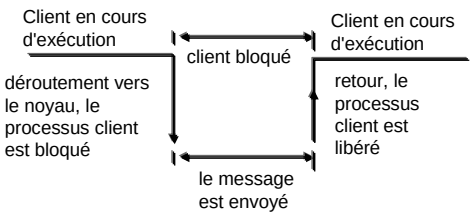


1 : demande à SN
2 : réponse de SN
3 : demande
4 : réponse

- les serveurs ont leur nom unique en code ascii
- un serveur gère la localisation et le nom interne des machines
- le client connaît seulement l'adresse du serveur de nom. Il lui demande de fournir l'adresse des serveurs à l'exécution

II. Modèles d'exécution
1. Modèle Client-Serveur

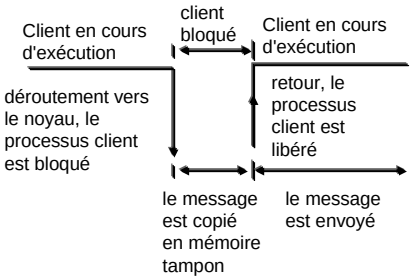
- Modèle à primitives : bloquantes ou non



- le processus appelant send ou receive sera bloqué pendant l'émission (ou réception) du message

- l'instruction qui suit send (ou receive) sera exécutée lorsque le message a été complètement émis (ou reçu)

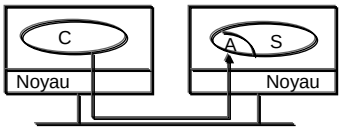
- avantage : simple et claire, facile à la mise en oeuvre
- inconvénient : moins performant



- le processus appelant send sera libéré toute de suite, le message sera envoyé en // à l'exécution
- avantage : performant
- inconvénient : perte de mémoire tampon, difficile à réaliser, risque de modif. du tampon

II. Modèles d'exécution
1. Modèle Client-Serveur

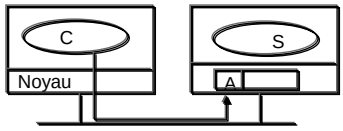
- Modèle à primitives : avec ou sans tampon



réseau

- sans tampon : le message envoyé par le client sera copié par le noyau du serveur dans la zone ptrm et pour le processus A à l'adresse adr dans l'appel receive(adr,ptrm) déclenché par le processus A

- problème : si le processus client a appelé send avant que le processus serveur appelle receive alors, le noyau du serveur n'a ni adresse du processus serveur ni l'adresse de l'endroit où il doit copier le message reçu !!!



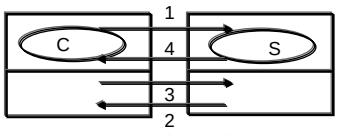
réseau

- avec tampon : les messages envoyés par les clients sont déposés dans une boite aux lettres. Le processus serveur A qui appelle un receive va déclencher une recherche par le noyau dans la boite aux lettres. Si le message existe, le noyau va le copier dans la zone de mémoire ptrm et libérer le processus A. Sinon l'appel receive sera bloqué pour l'attente de l'arrivée du message

- problème : quand la boite aux lettres est saturée, on doit refuser tous les messages qui arrivent !!!

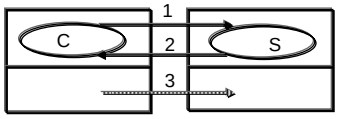
II. Modèles d'exécution
1. Modèle Client-Serveur

- Modèle à primitives : fiables ou non fiables



Message avec accusés de réception individuels

1- Demande (client au serveur)
2- ACK (noyau à noyau)
3- Réponse (serveur à client)
4- ACK (noyau à noyau)



Message avec la réponse = 1 accusé de réception

1- Demande (client au serveur)
2- Réponse (serveur à client)
3- ACK dans certain cas (noyau à noyau)
(si le calcul est cher, on peut bloquer le résultat au serveur jusqu'à qu'il reçoit un ACK du client (3)

On peut avoir un compromis entre 2 solutions précédantes :

1- Demande (client au serveur)
2- ACK éventuellement si la réponse n'est pas revenue dans certain temps (noyau à noyau)
3- Réponse servant ACK si dans le lape de temps prévu (serveur à client)
4- ACK éventuellement (noyau à noyau)

II. Modèles d'exécution
1. Modèle Client-Serveur

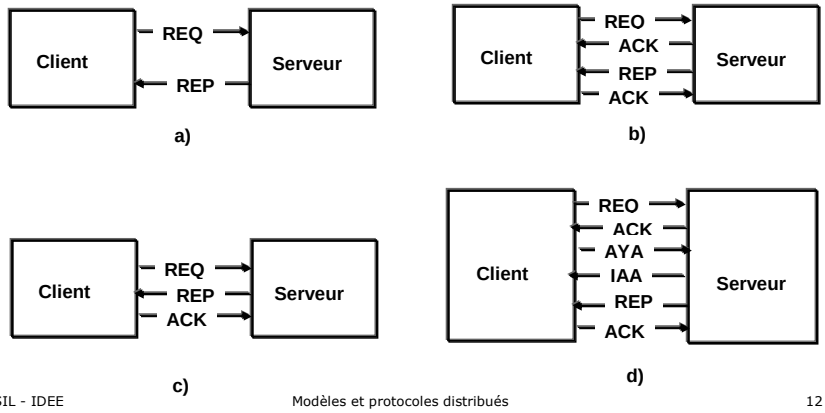
- Modèle à primitives : Résumé

Objet	Option 1	Option 2	Option 3
Adressage	numéro de la machine	adresse choisie par le proc.	nom ASCII avec SN
Blocage	primitives bloquantes	primitives non-bloquantes avec copie vers le noyau	primitives non-bloquantes avec interruption
Mémorisation	pas de tampon	tampon temporaire	boite aux lettre
Fiabilité	non fiable	demande-ACK-réponse-ACK	demande--réponse-ACK

Code	Type de paquet	source-dest.	Description
REQ	demande	client-serveur	le client désire un serveur
REP	réponse	serveur-client	la réponse du serveur au client
ACK	accusé de réception	noyau-noyau	le paquet précédent est arrivé
AYA	es-tu vivant ?	client-serveur	le serveur fonctionne-t-il ?
IAA	Je suis vivant	serveur-client	le serveur n'est pas en panne
TA	essaye à nouveau	serveur-client	le serveur est saturé
AU	adresse inconnue	serveur-client	aucun processus sur serveur utilise cette adresse

II. Modèles d'exécution
1. Modèle Client-Serveur

- Modèle à primitives : Résumé
  - Quelques exemples



II. Modèles d'exécution
1. Modèle Client-Serveur

- Modèle RPC (Remote Procedure Call): Introduction

Problème du modèle Client/Serveur

Concept basé sur l'échange explicite de données donc orienté Entrées/Sorties qui ne peut pas être le principe clé d'un système distribué

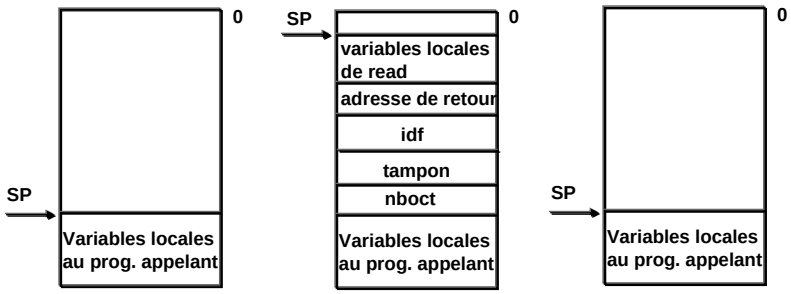
Appels de procédures à distance

Concept introduit par Birrell et Nelson en 1984 : lorque un processus sur la machine A appelle une procédure sur une machine B, le processus sur A est suspendu pendant l'exécution de la procédure appelée se déroule sur B. Des informations peuvent être transmises de l'appelant vers l'appelé ou l'inverse par des paramètres

II. Modèles d'exécution
1. Modèle Client-Serveur

- Modèle RPC : Introduction

compte = read(idf, tampon, nboct);



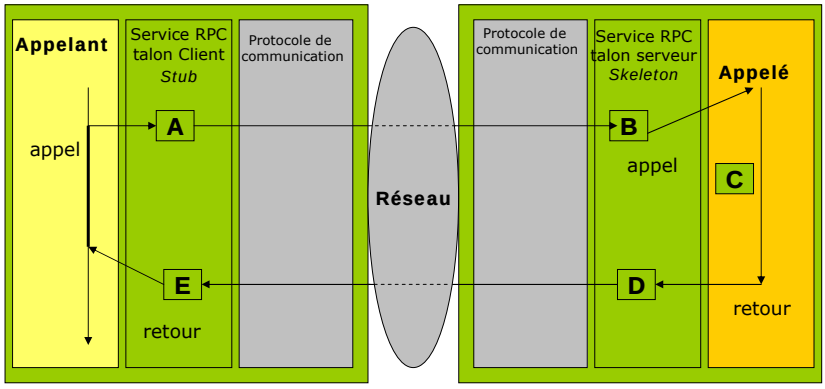
- modes de passage : par valeur ou par pointeur
-nature des paramètres : entré - sorti - entré / sorti
- SP : Pointeur de la pile système (Stack Pointer)

II. Modèles d'exécution
1. Modèle Client-Serveur

- Modèle RPC : Opérations de base
  - L'opération à réaliser est présentée sous la forme d'une procédure
  - (le service), située sur un site distant, dont le client demande l'exécution
  - Objectif : forme et effet "identiques" à ceux d'un appel local
    - interface de programmation
    - sémantique
  - Opérations de base
    - Client
      - doOp **(IN Port serverId, Name opName, Msg *arg, OUT Msg *result)**
    - Serveur
      - getRequest **(OUT Port clientId, Message *callMessage)**
      - sendReply **(IN Port clientId, Message *replyMessage)**

II. Modèles d'exécution
1. Modèle Client-Serveur

- Modèle RPC : **Birrel et Nelson (1984)**



II. Modèles d'exécution
1. Modèle Client-Serveur

- Modèle RPC : **Rôle des talons**

Talon client *STUD*

procédure d'interface sur le site client
- reçoit l'appel en mode local,
- "emballe" les paramètres et le transforme en appel distant en envoyant un message,
- attend les résultats en provenance du serveur,
- "déballe" les paramètres résultats, et les retourne au programme client comme dans un retour de procédure locale.
- Liaison initiale

Talon serveur *SKELETON*

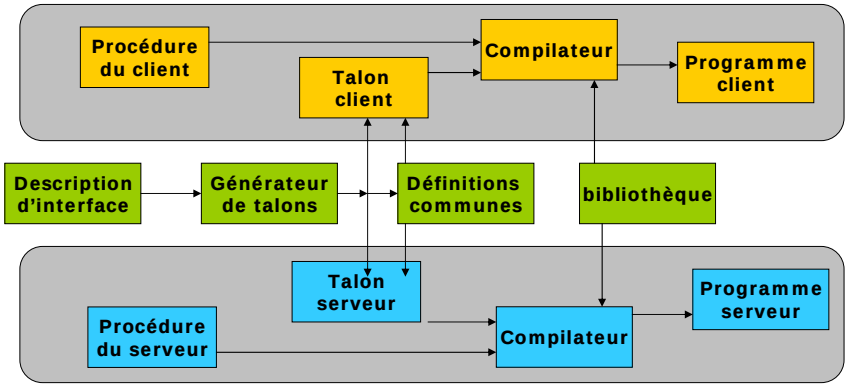
o procédure d'interface sur le site serveur
- reçoit l'appel sous forme de message,
- "déballe" les paramètres d'appel,
- fait réaliser l'exécution sur le site serveur par la procédure de service invoquée,
- "emballe" les paramètres résultats et les retransmet par message.
- Enregistrement initial

II. Modèles d'exécution
1. Modèle Client-Serveur

- Modèle RPC : **Langages de description d'interfaces (IDL)**
  - **Motivations**
    - Spécification commune au client et au serveur
      contrat entre le client et le serveur
    - Définition du type et de la nature des paramètres (IN, OUT, IN-OUT)
    - Indépendance des langages de programmation
  - **Principes**
    - Langage déclaratif
    - Outils de génération des talons (stub/proxy et skeleton/squelette)

II. Modèles d'exécution
1. Modèle Client-Serveur

- Modèle RPC : **Schéma opérationnel**



II. Modèles d'exécution
1. Modèle Client-Serveur

- Modèle RPC : **Compilation des interfaces**
- Produire les talons client et serveur
  - différents langages cibles
  - Talons client et serveur générés avec la même version du compilateur et de la spécification
    - vérification par estampille
- Procédure générée
  - empaquetage des paramètres
  - identification de la procédure à appeler
  - procédure de reprise après expiration des délais de garde
- Coté client
  - Remplacer les appels de procédure distants par des appels au talon client
- Coté serveur
  - Au démarrage le serveur se fait connaître (enregistrement dans un service de désignation)
  - recevoir l 'appel et effectuer l'appel de la procédure

II. Modèles d'exécution
1. Modèle Client-Serveur

- Modèle RPC : **Problème de traitement des défaillances**
  - **Gestion des erreurs**
    - panne et/ou congestion du réseau,
    - panne du client pendant le traitement de la requête,
    - panne du serveur avant ou pendant le traitement de la requête,
  - **Gestion de l'Hétérogénéité**
  - **Désignation et liaison**
  - **Problèmes de sécurité**
    - authentification du client et du serveur
    - confidentialité des échanges
  - **Aspects pratiques**
    - adaptation à des environnements d'utilisation variables (protocoles, langages, matériels)
    - performance

II. Modèles d'exécution
1. Modèle Client-Serveur

- Modèle RPC : **Problème de traitement d'erreur**
  - **sémantique** variable selon la nature du mécanisme de reprise mis en œuvre
    - Indéfini
    - Au plus une fois
      - si expiration du délai A alors erreur
      - pas de mécanisme de reprise
    - Au moins une fois
      - si expiration du délai A alors ré-émission de la requête,
      - plusieurs exécutions possibles du service,
      - acceptable si opération idempotente
    - Exactement une fois (idéal...)
      - ré-émission de la requête et/ou de la réponse si nécessaire,
      - élimination des "doublons"

II. Modèles d'exécution
1. Modèle Client-Serveur

- Modèle RPC : **Problème de traitement d'erreur**

  Cinq classes d 'erreurs principales

  - **1. Le client est incapable de localiser le serveur**
  - **2. La demande du client au serveur est perdue**
  - **3. La réponse du serveur au client est perdue**
  - **4. Le serveur tombe en panne après avoir reçu une demande**
  - **5. Le client tombe en panne après avoir envoyé une demande**

II. Modèles d'exécution
1. Modèle Client-Serveur

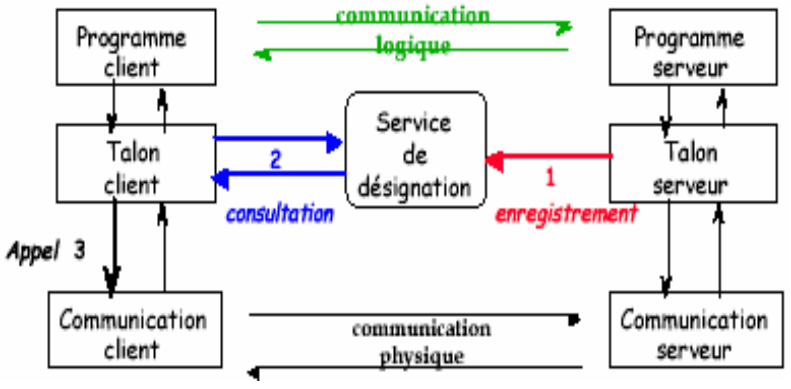
- Modèle RPC : **Problème de formats hétérogènes**
  - **Problème : représentation des données**
    - Conversion nécessaire si le site client et le site serveur n 'utilisent pas le même système de codage (big endian versus little endian)
    - utilisent des formats internes différents (pour les types de base: caractère, entier, flottant, …)
  - **Solutions**
    - Syntaxe abstraite de transfert ASN 1
    - Représentation externe commune : XDR Sun (non optimal si même représentation)
    - Représentation locale pour le client et conversion par le serveur
    - Choix d'une représentation parmi n (standard), conversion par le serveur
    - Négociation client serveur

II. Modèles d'exécution
1. Modèle Client-Serveur

- Modèle RPC : **Problème de désignation**
  - **Problèmes**
    - objets à désigner : le site d 'exécution, le serveur, la procédure
    - désignation globale indépendante de la localisation
      - possibilité de reconfiguration des services (pannes, régulation de, …)
- **Mise en œuvre : statique ou dynamique**
  - statique : localisation du serveur connue à la compilation
  - dynamique : localisation déterminée à l'exécution (au premier appel)
    - séparer connaissance du nom du service de la sélection de la procédure qui va l 'exécuter
    - permettre l 'implémentation retardée

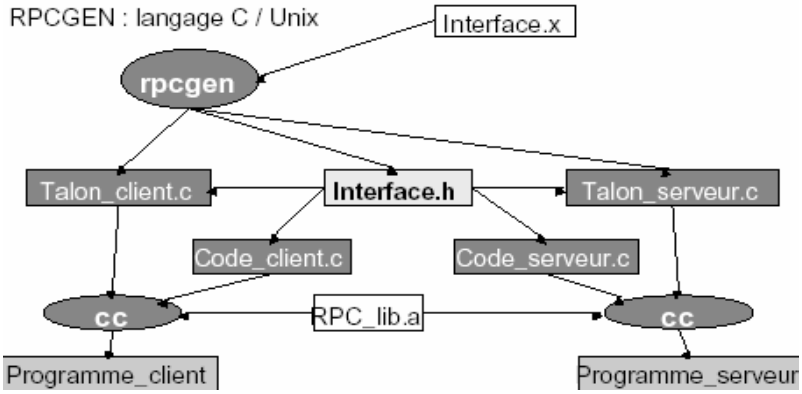
II. Modèles d'exécution
1. Modèle Client-Serveur

- Modèle RPC : **Problème de désignation**



II. Modèles d'exécution
1. Modèle Client-Serveur

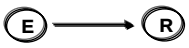
- Modèle RPC: **Exemple IDE**



II. Modèles d'exécution
1. Modèle Client-Serveur

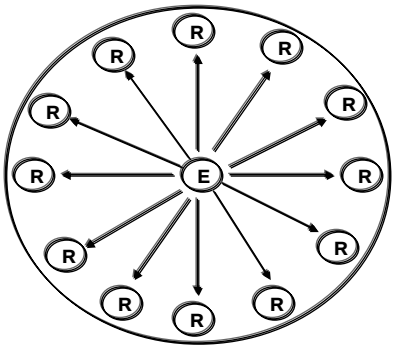
- Modèle Multipoint :

Point-à-Point



Généralisation du RPC

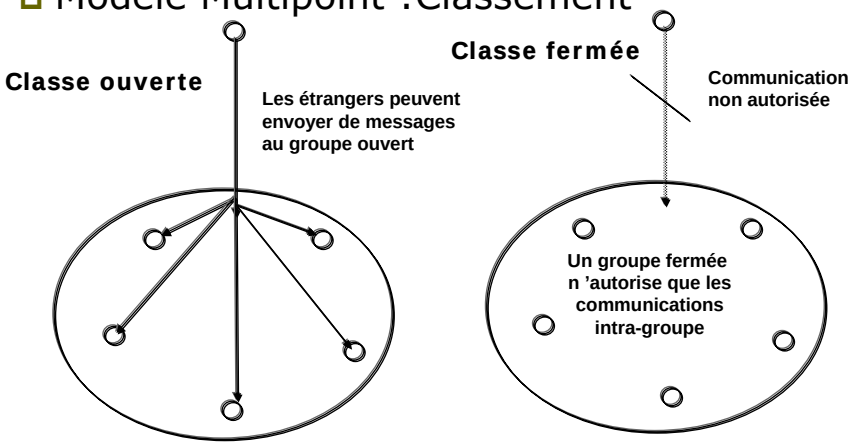
Un émetteur doit communiquer en parallèle à n récepteurs regroupés dans un groupe sémantique



Un-à-Plusieurs

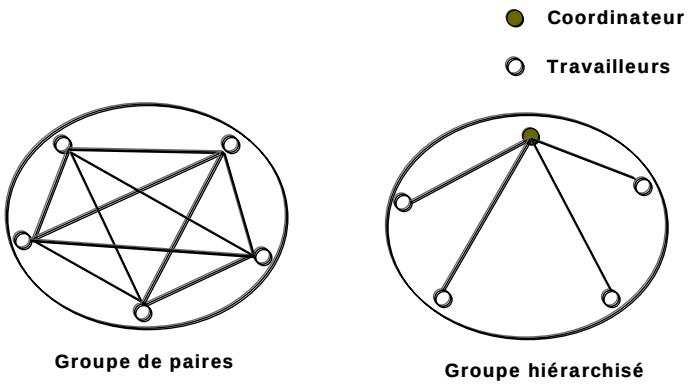
II. Modèles d'exécution
1. Modèle Client-Serveur

- Modèle Multipoint :Classement



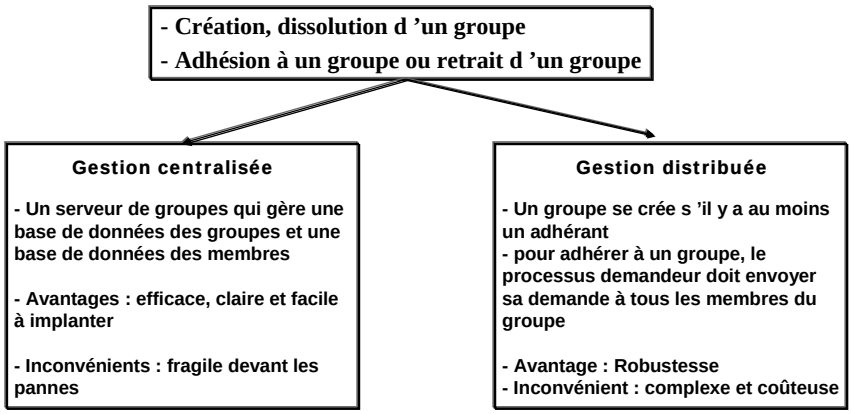
II. Modèles d'exécution
1. Modèle Client-Serveur

- Modèle Multipoint :Classement



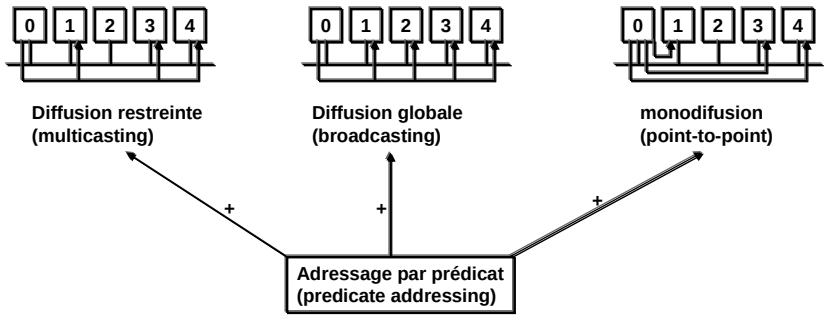
II. Modèles d'exécution
1. Modèle Client-Serveur

- Modèle Multipoint :Gestion de groupes



II. Modèles d'exécution
1. Modèle Client-Serveur

- Modèle Multipoint :Modes d'adressage



II. Modèles d'exécution
1. Modèle Client-Serveur

- Modèle Multipoint : Diffusion atomique

Propriété de diffusion atomique (atomic broadcast)

Un message envoyé à un groupe doit être reçu par tous ses membres ou par aucun

Algorithme de « relais intra-groupe » (Joseph et Birman 1989)

1- L 'expéditeur envoie un message à tous les membres d 'un groupe (des temporisateurs sont armés et des retransmissions faites si nécessaire)

2- Tout membre de ce groupe, qui a reçu ce message pour la première fois, doit l 'envoyer à tous les membres du groupe (des temporisateurs sont armés et des retransmissions faites si nécessaire)

3- Tout membre de ce groupe, qui a reçu ce message plus qu 'une fois, doit simplement le détruire

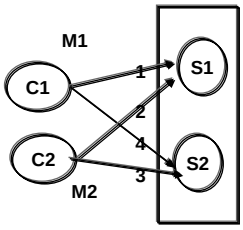
II. Modèles d'exécution
1. Modèle Client-Serveur

- Modèle Multipoint : Séquencement

Propriété de séquencement

Soient A et B 2 messages à envoyer à un groupe de processus G. Si A est envoyé avant B sur le réseau, ils doivent être reçus par tous les membres du groupe G dans le même ordre : A puis B

MAJ Incohérente



Solutions

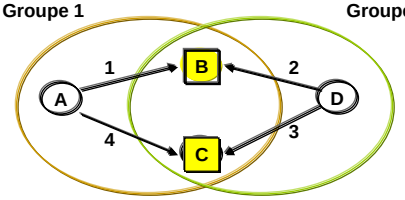
- Séquencement global (global time ordering) :
conserver, en réception, l 'ordre des messages à émission.
Si C1 envoie M1 avant que C2 envoie M2, alors le système doit assurer que tous les membres de G reçoivent M1 avant M2

- Séquencement cohérent (Consistent time ordering) :
Le système établir l 'ordre d 'envoie des messages et assurer que cet ordre soit respecté à la réception des messages.

II. Modèles d'exécution
1. Modèle Client-Serveur

- Modèle Multipoint : Groupes non disjoints

Groupe 1 Groupe 2



L 'ordre de recouvrement des groupes non respecté :
- B reçoit le message de A puis D
- C reçoit le message de D puis A

Si B et C est une base de données dupliquée on peut avoir une incohérence de MAJ

Solution

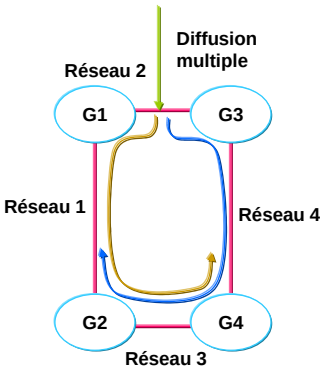
Introduction des mécanisme de séquencement qui tient compte du recouvrement des groupes

Problème

La mise en œuvre est compliquée et coûteuse

II. Modèles d'exécution
1. Modèle Client-Serveur

- Modèle Multipoint : élasticité



Propriété

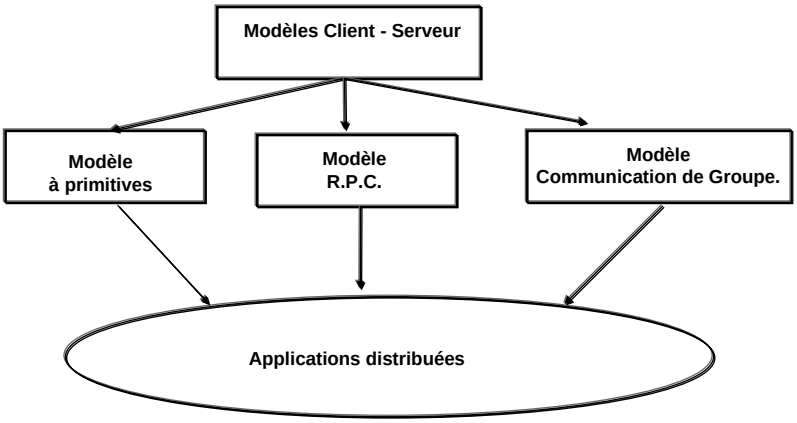
La communication de groupe doit être invariante par rapport à la taille du groupe : L'algorithme doit être efficace dans un petit groupe comme dans un grand groupe (nombre de membre et taille de l'espace)

Problèmes

- beaucoup d 'algorithmes fonctionnent très bien dans des petits groupes mais deviennent non fiable dans des grands groupe
- la taille de réseau avec la présence des passerelles rend plus compliqué le problème de communication de groupe

II. Modèles d'exécution
1. Modèle Client-Serveur

- Résumé : les approches



II. Modèles d'exécution
1. Modèle Client-Serveur

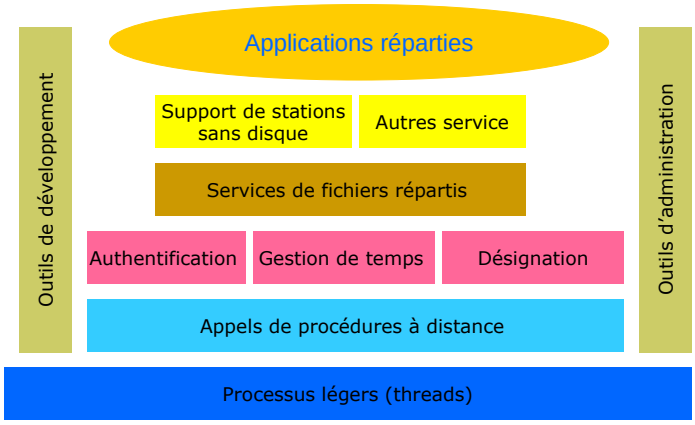
- Limites des modèles Client – Serveur
  - **modèle de structuration**
    - permet de décrire l'interaction entre **deux** composants logiciels
      - absence de vision globale de l'application
    - schéma d'exécution répartie élémentaire (appel synchrone)
      - absence de propriétés portant sur la synchronisation, la protection, la tolérance aux pannes, . .
  - **services pour la construction d'applications réparties**
    - le RPC est un mécanisme de "bas niveau"
    - des services additionnels sont nécessaires pour la construction d'applications réparties (désignation, fichiers répartis, sécurité, etc.)
  - **outils de développement**
    - limités à la génération automatique des talons
    - peu (ou pas) d'outils pour le déploiement et la mise au point d'applications réparties

II. Modèles d'exécution
1. Modèle Client-Serveur

- Quelques environnements Client – Serveur
  - **environnement client-serveur "traditionnel"**
    - DCE (Distributed Computing Environment) :
      - un exemple d'environnement "intégré" d'applications réparties,fondé sur le modèle client-serveur
  - **environnement client-serveur "à objet"**
  - **environnement client-serveur "de données"**
  - **environnement client-serveur "à composant"**

II. Modèles d'exécution
1. Modèle Client-Serveur

- Environnement DCE

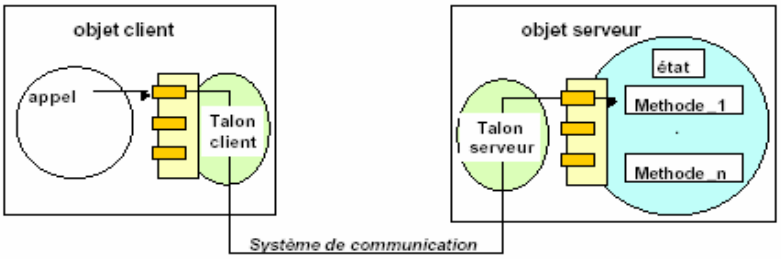


II. Modèles d'exécution
1. Modèle Client-Serveur

- Environnement DCE
  - **architecture de type "boîte à outils"**
    - chaque fonction est utilisable via une interface "normalisée"
    - accès direct à des fonctions évoluées
      - intégration "à gros grain"
  - **"standard" de fait** (diffusion dans la communauté industrielle)
    - ex. : RPC, Service de noms, service d'authentification
      - portabilité et interopérabilité
  - **développement d'applications "à grain fin" laborieux**
    - ex. : utilisation des services de thread et du RPC (limitations équivalentes à celles du client-serveur)
    - peu d'outils de développement disponibles aujourd'hui

II. Modèles d'exécution
1. Modèle Client-Serveur

- Environnement Client-serveur à objets
  - **Motivations**
    - propriétés de l'objet (encapsulation, modularité, réutilisation, polymorphisme, composition)
    - objet : unité de désignation et de distribution
  - **Principe de fonctionnement**

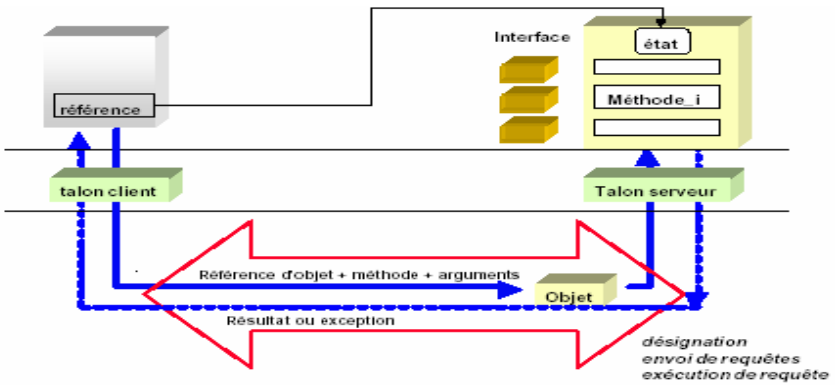


II. Modèles d'exécution
1. Modèle Client-Serveur

- Environnement Client-serveur à objets RMI (Remote Method Invocation)
  - **Éléments d'une "invocation"**
    - référence d'objet ("pointeur" universel)
    - identification d'une méthode
    - paramètres d'appel et de retour (y compris signal d'exception)
      - passage par valeur : types élémentaires et types construits
      - passage par référence
  - **Objets "langage"**
    - représentation propre au langage : instance d'une classe
    - exemple : Java RMI
  - **Objets "système"**
    - représentation "arbitraire" définie par l'environnement d'exécution
    - exemple : CORBA

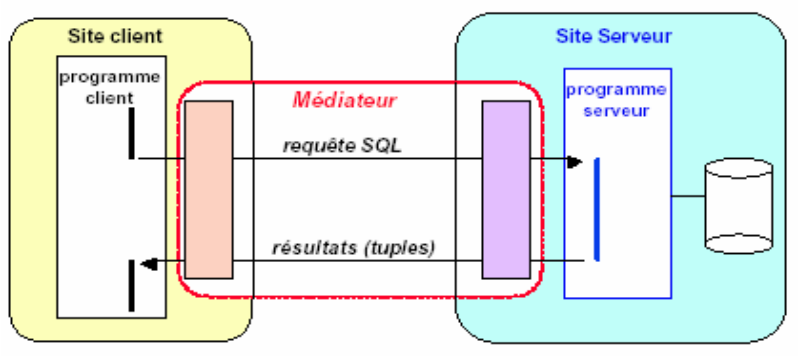
II. Modèles d'exécution
1. Modèle Client-Serveur

- Environnement Client-serveur à objets RMI (Remote Method Invocation)



II. Modèles d'exécution
1. Modèle Client-Serveur

- Environnement Client-Serveur de données



II. Modèles d'exécution
1. Modèle Client-Serveur

- Environnement Client-Serveur de données
  - **fonction du client**
    - code de l'application non lié aux données
    - dialogue avec l'utilisateur
  - **fonction du serveur**
    - stockage des données, gestion de la disponibilité et de la sécurité
    - interprétation/optimisation des requêtes
  - **fonctions du "médiateur"**
    - procédures de connexion/déconnexion
    - préparation/communication de requêtes
    - gestion de caches (requêtes et résultats)

II. Modèles d'exécution
1. Modèle Client-Serveur

- Environnement Client-Serveur de données
  - **Environnement Applicatif Commun (CAE) de l'X/OPEN**
    - objectif : portabilité des applications
    - interface applicative CLI (Call Level Interface)
      - standardisation des appels SQL depuis un programme (C, Cobol, . .)
      - connexion/déconnexion, préparation/exécution des requêtes, . . .
    - protocole d'échange standardisé RDA (Remote Data Access)
    - partie générique (dialogue et contrôle transactionnel)
    - partie spécifique SQL (codage commandes et résultats)
  - **Outils de développement**
    - L4G : intègre accès aux tables, contrôle, affichage, saisie, ... indépendants des langages de programmation langage cible : C (ou Cobol)
    - AGL : fonctions des L4G + gestion d'un référentiel des objets (schémas de données, code des opérations, enchaînement, . . .)

II. Modèles d'exécution
1. Modèle Client-Serveur

- Environnement Client-Serveur de données Solution « à objets » ODMG
  - **Objectifs**
    - Portabilité des sources d'une application sur divers "moteurs" de bases de données à objet
  - **Principes directeurs**
    - modèle objet = sur-ensemble du modèle OMG
    - un langage de Définition de Données : ODL (basé sur l'IDL OMG)
    - un langage de Requêtes : OQL, évolution de SQL
    - intégration complète des mécanismes dans le langage hôte : C++ (avec propositions d'évolution de la norme) et Smalltalk

II. Modèles d'exécution
1. Modèle Client-Serveur

- Environnement CL/SV à composants
  - **Motivations**
    - à l'origine, environnements pour la gestion de documents composites
    - automatisation du processus de création d'un document à partir d'informations hétérogènes fournies par des utilitaires différents
    - exemples : OLE/COM, OpenDoc/SOM
  - **Approche**
    - interopérabilité entre composants logiciels binaires
    - environnements homogènes
    - interaction de type client-serveur
    - format de données "pivot" et procédures de conversion

II. Modèles d'exécution
2. Communication par messages

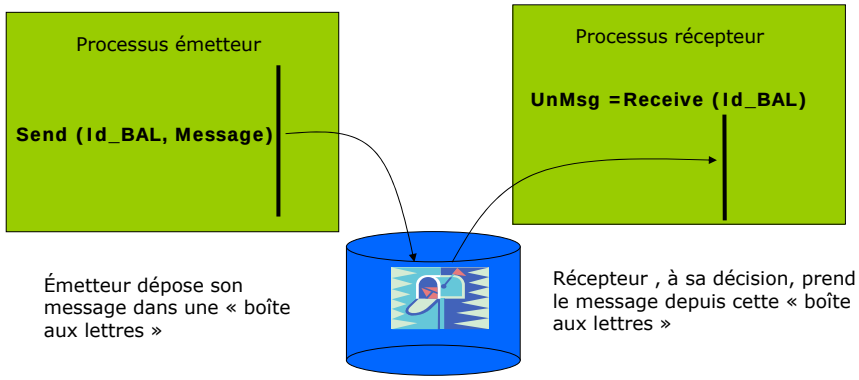
- Introduction : historique
  - **Un mode de communication classique**
    - Le courrier électronique
      - communication de type asynchrone
      - listes de diffusion (communication de groupe)
    - Les forums (News)
      - communication filtrée par "sujet"
      - abonnement avant réception
  - **Émergence de paradigmes de programmation**
    - Années 70: Message Queuing (IBM MQ, Oracle, Sun, etc.)
    - modèles événements/réaction, publish/subscribe
    - modèles d'acteurs
    - programmation par agents

II. Modèles d'exécution
2. Communication par messages

- Modes de programmation
  - **Mode de synchronisation**
    - Communication asynchrone
    - émission non bloquante
    - réception bloquante (attente jusqu'à réception d'un message)
  - **Mode de communication**
    - communication directe entre processus (acteurs, agents, etc.)
    - communication indirecte entre processus via des "portes" (boîtes aux lettres)
  - **Mode de transmission**
    - messages possiblement typés

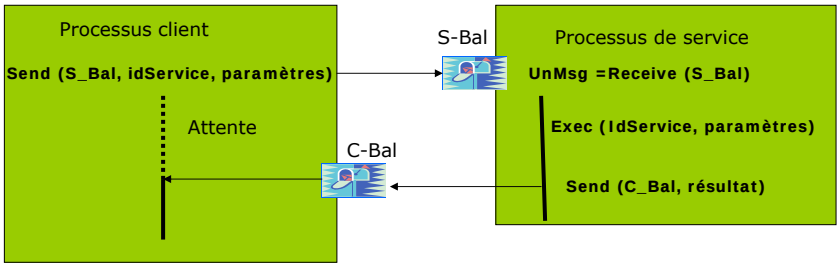
II. Modèles d'exécution
2. Communication par messages

- Exemple de communication asynchrone :
  - L'émetteur et récepteur n'ont pas de contact direct



II. Modèles d'exécution
2. Communication par messages

- Exemple de communication synchrone :
  - L'émetteur (Client) et récepteur (Serveur) n'ont pas de contact direct et chacun dispose sa boite aux lettres



La communication est synchrone, l'émetteur se met en attente du résultat. Cependant, il n'y a pas de contact direct entre eux.

II. Modèles d'exécution
2. Communication par messages

- Environnement de développement :
  - **Environnement de type "micro-noyau"**
    - mécanisme et primitives de base
    - exemples : Chorus, Mach/OSF-1
  - **Environnement "à la unix"**
    - "sockets"
  - **Environnement de programmation parallèle**
    - PVM et/ou MPI
  - **Environnement "industriel" d'intégration d'applications**
    - middleware à messages: MOM, Application serveurs, etc.
    - interface de programmation ad hoc
      - tentative de normalisation via Java JMS

II. Modèles d'exécution
2. Communication par messages

- MOM (Message Oriented Middleware) :
  - Modèle de communication entre logiciels
    - Intégration de modules hétérogènes distribués
    - Indépendance (asynchronisme)
    - Fiabilité
  - Plusieurs implantation
    - IBM (MOM websphere)
    - Application Servers (Sun, Oracle, etc.)
    - MOM open source : Joram, etc.
  - Une interface normalisée (en Java)
    - JMS (Java Message Services) de SUN

II. Modèles d'exécution
2. Communication par messages

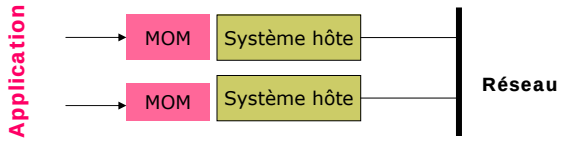
- MOM: Principes de base
  - *Message Queueing*
    - Queues de messages persistantes
    - Transmission des messages asynchrone (stockage des messages si nécessaire)
    - Reprise après panne
  - Un émetteur remet son message au système et peut continuer son exécution sans se soucier de l'état du destinataire (émetteur non bloquant)

II. Modèles d'exécution
2. Communication par messages

- MOM: Caractéristiques
  - Modes de communication
    - Point-à-point (PTP): émetteur, récepteur et queue
    - Publication/Souscription (Pub/Sub): émetteur, abonné et nœud
  - Modèle de programmation
    - Réception explicite / implicite
    - Messages, queues de messages
  - Messages
    - Messages dotés d'attributs et de propriétés
    - Priorités, garantie de délivrance

II. Modèles d'exécution
2. Communication par messages

- MOM: Caractéristiques
  - transactions
    - messages vus comme des ressources "transactionnelles"
  - sécurité
    - encodage des messages
    - contrôle d'accès aux messages
  - couche de logiciel sur le système hôte

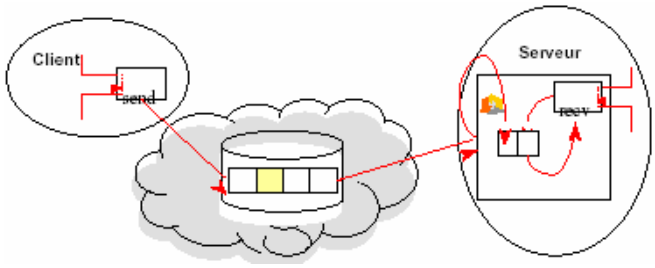


II. Modèles d'exécution
2. Communication par messages

- MOM: Messages
  - **Identification unique**
  - **Structure**
    - Entête :
      - Information permettant l'identification et l'acheminement du message
    - Attributs :
      - Couples (nom, valeur) utilisables par le système ou l'application pour sélectionner les messages
    - Données :
      - Définies par l'application
  - **Paramètres**
    - durée de vie
    - priorité
    - sécurité

II. Modèles d'exécution
2. Communication par messages

- MOM : Queues de messages
  - **identification unique**
  - **persistantes** (résistance aux défaillances)
  - **partagées par les applications**
  - **modes de réception variable**



II. Modèles d'exécution
2. Communication par messages

- MOM : API
  - **MsgQ.attach(name, type) –> msgQ**
    - Attacher à une queue de messages. Le retour est un pointeur sur cette queue
  - **SendQ.sendMsg(msg)**
    - Envoyer un message dans une queue de message (sendQ)
  - **RecvQ.recvMsg(wait) –> msg**
    - Récupérer un message depuis d'une queue (RecvQ). Le résultat est un messgage
  - RecvQ.confirmMsg(msg)
    - Confirmer la réception d'un message
  - **MsgQ.detach()**
    - Détacher d'une queues de message

II. Modèles d'exécution
2. Communication par messages

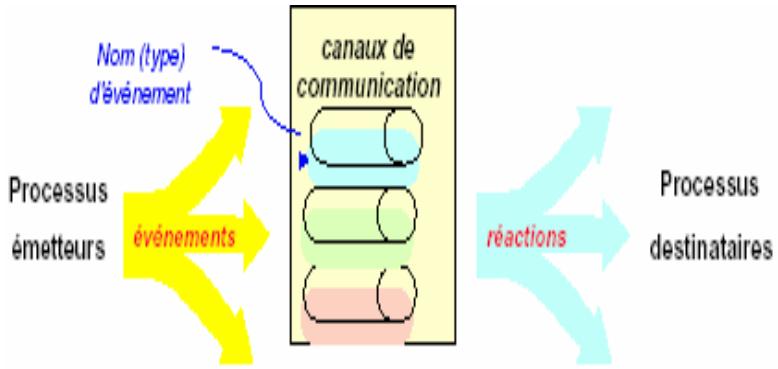
- MOM : Extensions
  - **Communication de groupe**
    - groupe : ensemble de récepteurs identifiés par un nom unique
    - gestion dynamique du groupe : arrivée/départ de membres
    - différentes politiques de service dans le groupe : 1/N, N/N
    - mise en œuvre : utilisation possible de IP multicast
    - exemple : Isis, Horus, Ensemble (Cornell university)
    - applications : tolérance aux fautes (gestion de la réplication), travail coopératif
  - **Communication anonyme**
    - désignation associative : les récipiendaires d'un message sont identifiés par leurs propriétés et pas par leur nom
    - propriété : attribut du message ou identificateur externe
      - indépendance entre émetteur et récepteurs

II. Modèles d'exécution
3. Communication par évènements

- Principes
  - **concepts de base :** événements, réactions
    - réaction : traitement associé à l'occurrence d'un événement
  - **principe d'attachement**
    - association dynamique entre un nom d'événement et une réaction
  - **communication anonyme**
    - indépendance entre l'émetteur et les "consommateurs" d'un événement

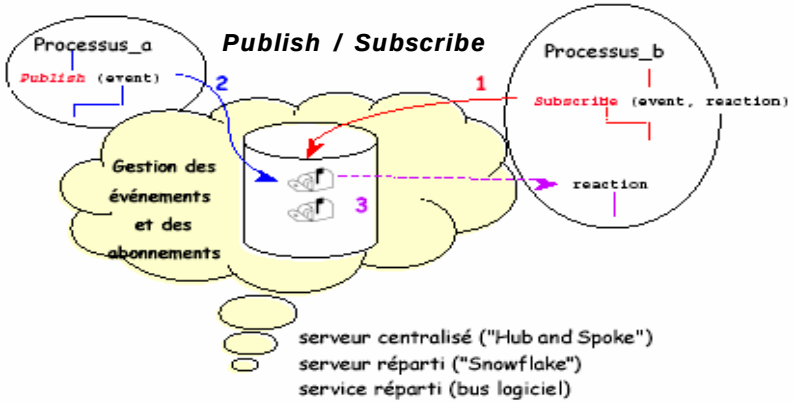
II. Modèles d'exécution
3. Communication par évènements

- Principes



II. Modèles d'exécution
3. Communication par évènements

- Mises en œuvre : principes

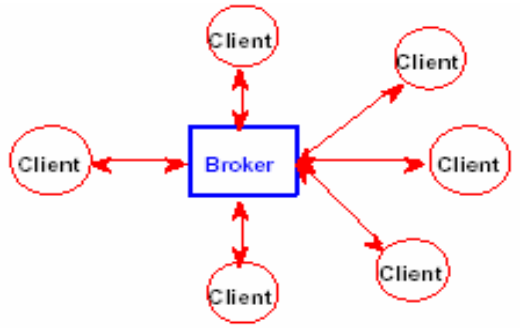


II. Modèles d'exécution
3. Communication par évènements

- Gestion des évènements
  - **Mode** "Pull"
    - Les clients viennent "prendre" périodiquement leurs messages sur le serveur.
  - **Mode** "Push"
    - Une méthode prédéfinie (réaction) est attachée à chaque type de message (événement) ;
    - l'occurrence d'un événement entraîne l'exécution de la réaction associée.

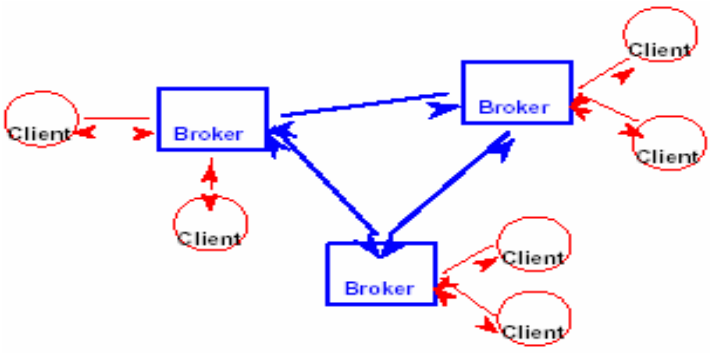
II. Modèles d'exécution
3. Communication par évènements

- Architecture **"Hub and Spoke"**
  - Serveur centralisé de gestion d'événements
    - protocole de communication point-à-point



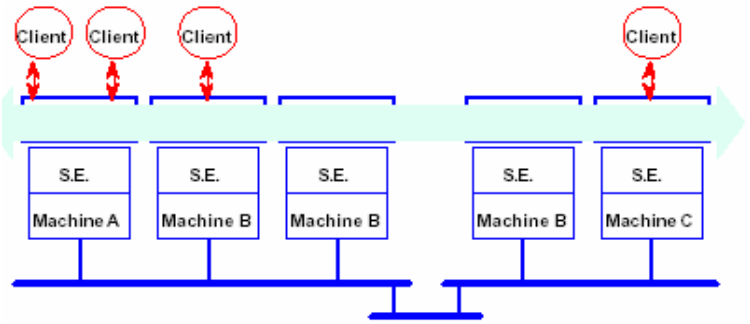
II. Modèles d'exécution
3. Communication par évènements

- Architecture **"Snowflake"**
  - protocole de communication point-à-point
    - Serveur réparti de gestion d'événements



II. Modèles d'exécution
3. Communication par évènements

- Architecture **bus de messages**
  - protocole de communication multicast
    - Service réparti de gestion d'événements

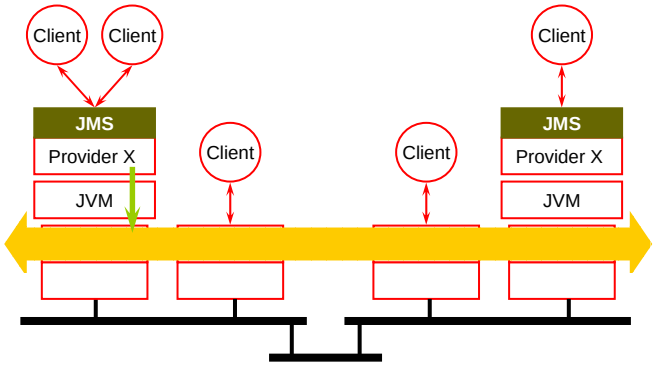


II. Modèles d'exécution
3. Communication par évènements

- MOM : interface Java Message Service (JMS)
  - **Objectif : API Java d'accès uniforme aux systèmes de messagerie**
    - IBM MQSeries
    - Novell, Oracle, Sybase
    - Tibco
    - Sun, etc.
  - **Propriétés**
    - Point-à-Point
    - Publish/Subscribe

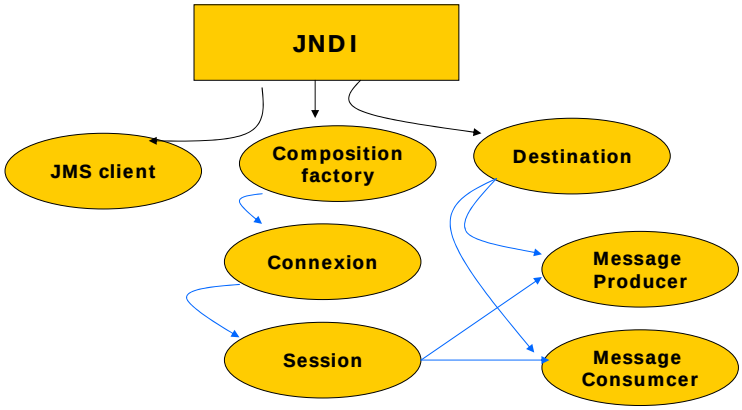
II. Modèles d'exécution
3. Communication par évènements

- MOM : interface Java Message Service (JMS)
  - API Java d'accès uniforme aux systèmes de messagerie



II. Modèles d'exécution
3. Communication par évènements

- MOM : interface Java Message Service (JMS)



II. Modèles d'exécution
3. Communication par évènements

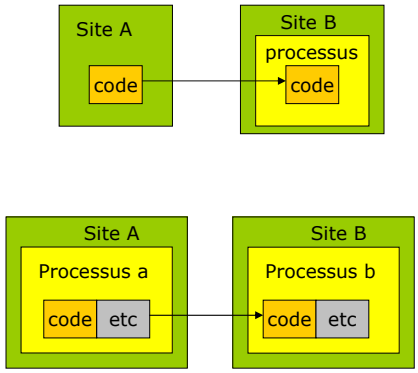
- Résumé:
  - **Domaines d'application**
    - génie logiciel (coopération entre outils de développement) :
    - SoftBench, ToolTalk , DecFuse, . .
    - workflow : KoalaBus, . .
    - Intégration/extension d'applications existantes : AAA, . .
    - diffusion de logiciels et d'information sur le Web : Bus, Castanet, Ambrosia, Smartsockets, TIB/Rendezvous, Active Web,
  - **Infrastructures propriétaires**
    - interface applicative et protocoles propres à chaque système
      - problèmes de portabilité et d'interopérabilité
  - **Outils de développement**
    - sommaires

II. Modèles d'exécution
4. Autres modèles

- Modèle à codes mobiles
  - **Définition**
    - programmes pouvant se déplacer d'un site à un autre
    - exemples : requête SQL, "applet" Java
  - **Motivations**
    - rapprocher le traitement des données
      - réduire le volume de données échangées sur le réseau
      - partage de charge
    - function shipping versus data shipping
  - **Caractéristiques**
    - code interprétable
    - sécurité
    - schémas d'exécution à base de code mobile

II. Modèles d'exécution
4. Autres modèles

- Modèle à codes mobiles
  - **code "à la demande"**
    - mobilité "faible«
    (code exécutable, sans contexte)
    - exemple : "applet" Java
  - **agents mobiles**
    - mobilité "faible"
    - code exécutable + données modifiées
    - exemple : Aglets
    - mobilité "forte"
    - code exécutable, + données + contexte d'exécution
    - exemples : AgentTcl

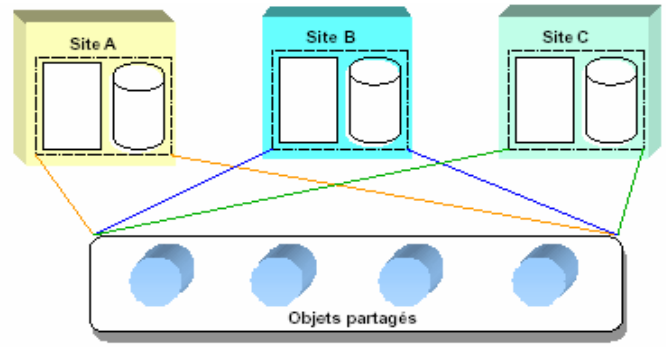


II. Modèles d'exécution
4. Autres modèles

- Modèle à mémoire distribuée
  - **Motivations**
    - (re)placer le programmeur d'application dans les conditions d'un système centralisé
      - utiliser une mémoire commune comme espace de communication
      - synchronisation par variables partagées
    - avantages attendus (pour le programmeur)
      - simplicité : distribution "transparente"
      - efficacité : utilisation des paradigmes usuels de la programmation concurrente
  - **problématique**
    - utilisation des outils de développement existants : langages, compilateurs, metteurs au point, . . .
    - mise en œuvre "efficace" d'une mémoire partagée distribuée

II. Modèles d'exécution
4. Autres modèles

- Modèle à mémoire distribuée
  - **Principe de réalisation**
    - "simulation" d'une mémoire globale (d'objets) partagée

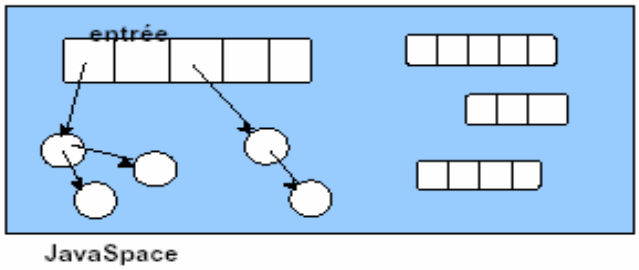


II. Modèles d'exécution
4. Autres modèles

- Modèle à mémoire distribuée
  - **Approches**
    - Modèles à espace de tuples
      - base de données (de tuples) partagée
      - modèle de programmation "à la Linda"
        - **dépôt, retrait, et consultation d'objets**
      - exemple : JavaSpaces
    - Modèles à objets répartis partagés
      - espace d'objets répartis partagés
      - interface de programmation : langage à objet "étendu"
      - plusieurs modes de réalisation
        - objets répliqués (exemple : Javanaise)
        - objets à image unique (exemple : Guide)

II. Modèles d'exécution
4. Autres modèles

- Modèle à mémoire distribuée
  - Environnement de Javaspace
    - Un JavaSpace : un espace de tuples (entrées)
    - Une entrée : un ensemble de champs
    - Un champ : une référence à une instance Java



II. Modèles d'exécution
4. Autres modèles

- Modèle à mémoire distribuée
  - Environnement de Javaspace
    - **Opérations de base**
      - écriture dans un JavaSpace (dépôt)
      - lecture dans un JavaSpace (avec conformité)
      - retrait dans un JavaSpace (avec conformité)
      - notification de l'écriture d'une entrée conforme
      - transaction groupant plusieurs opérations (sur potentiellement plusieurs JavaSpaces)
      - persistance
        - par sérialisation des objets d'une entrée

II. Modèles d'exécution
4. Autres modèles

- Modèle à mémoire distribuée
  - Environnement de Javaspace
    - **Différence avec bases de données**
      - pas de langage de requête
      - sélection de données par "conformité" (vis-à-vis d'un modèle)
      - pas de modification sur les données de la base (ajouts et retraits seulement)
      - utilisation : espace de travail partagé pour applications coopératives
    - **Différence avec Linda**
      - les entrées et les objets sont typés
      - gestion possible de plusieurs espaces transactions
    - **JavaSpaces : interface de programmation**
      - **Les primitives**
        - write : depôt d'une nouvelle entrée
        - read : lit une entrée conforme à un template
        - take : lit et retire une entrée conforme à un template
        - notify : enregistre une notification à envoyer lorsqu'une nouvelle entrée conforme à un template est écrite
        - la méthode notify du gestionnaire d'événement (listener) est appelée
      - template : modèle pour la recherche d'entrées dans un JavaSpace

II. Modèles d'exécution
4. Autres modèles

- Modèle à mémoire distribuée
  - Environnement de Javaspace
    - **Les transactions**
      - avec write
        - écriture visible ssi la transaction valide (commit)
        - si write suivi de take dans une transaction : pas visible de l'extérieur
      - avec read
        - recherche dans les JavaSpaces et dans les dépôts de la transaction en cours (sans ordre)
        - une entrée lue (read) ne peut pas être retirée (taken) par une autre transaction

II. Modèles d'exécution
4. Autres modèles

- Modèle à mémoire distribuée
  - Environnement de Javaspace
    - **Les transactions (suite)**
      - avec take
        - une entrée retirée (taken) ne peut être lue (read) ou retirée (taken) par une autre transaction
      - avec notify
        - hors transaction, les notifications de voient que les écritures validées
        - dans une transaction, les notifications voient en plus les écritures internes à la transaction
        - la fin d'une transaction annule les définitions de notification dans la transaction
      - Les propriétés ACID sont respectées

II. Modèles d'exécution
4. Autres modèles

- Modèle à mémoire distribuée
  - Environnement de Javaspace : résumé
    - **Utilisation**
      - coordination entre applications réparties
      - découplage entre le client et le serveur
      - remarque : nécessite toujours au moins le partage d'une interface (comme RMI) ...
    - **Implantations possibles**
      - client/serveur : requêtes distantes depuis tous les clients
      - objets dupliqués : localité des accès à la base
    - **État des lieux**
      - Une version fournie dans l'environnement Jini
        - liaison dynamique à des ressources dans un réseau

II. Modèles d'exécution
4. Autres modèles

- Modèle à objets répartis
  - **Modèle de programmation**
    - modèle à objets
    - modèle d'exécution
      - objets "actifs" versus objets "passifs"
    - langage de programmation d'applications réparties intégrant distribution, parallélisme, synchronisation, persistance, etc.
      - extension d'un langage existant (pre-processeur) ou langage ad-hoc
  - **Principes de mise en œuvre**
    - désignation : références universelles d'objets
    - gestion de la persistance
    - gestion du partage : synchronisation, cohérence
    - image unique d'un objet versus copies (cohérentes) d'un objet