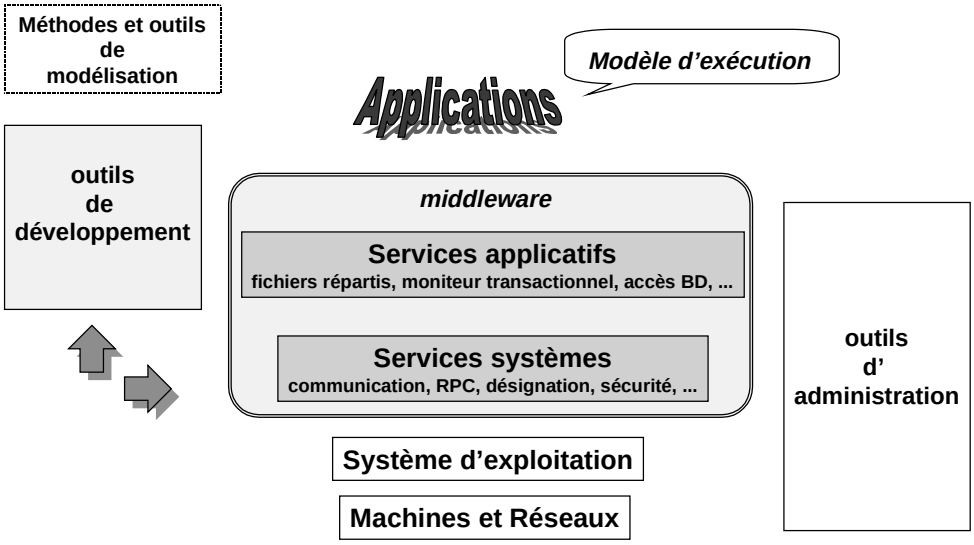
Modes de structuration d'applications réparties

Roland Balter

Problématique (1)

- **application répartie = traitements coopérants sur des données réparties**
- **coopération = communication + synchronisation**
  - modèle d 'exécution
  - interface de programmation (et/ou langage)
  - outils de développement
  - mise en œuvre : services systèmes (pour différentes infrastructures)

Problématique (2)



Modèles d'exécution

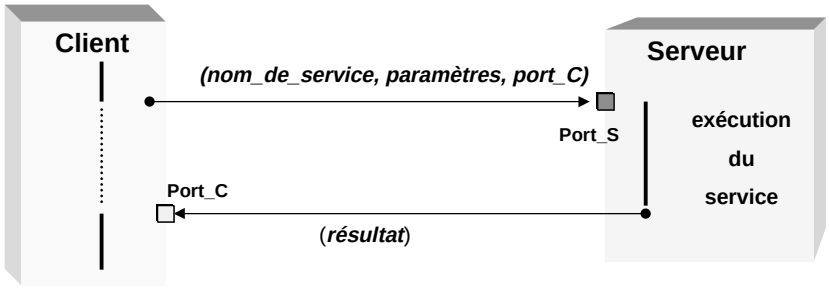
- ✔ **communication par messages**
- ❑ **événements/réactions**
- ❑ **demande de service : *mode client-serveur***
  - ◆ client-serveur "traditionnel"
  - ◆ client-serveur "de données"
  - ◆ client-serveur "à objets"
  - ◆ client-serveur "à composants"
- ❑ **code mobile**
- ❑ **mémoire virtuelle partagée**
- ❑ **autres**
  - ◆ "peer to peer", transactionnel, . .

Communication par messages
principes directeurs

- **Principes directeurs**
  - ***communication asynchrone***
  - communication directe ou indirecte (via des “portes”)
    ---> problème de désignation des entités coopérantes
  - messages éventuellement typés
- **Interface de programmation et mise en oeuvre**
  - interface “socket” sur le niveau transport
  - primitives de communication élémentaires ("envoyer", recevoir") :
    - » dans une architecture de type micro-noyau : Chorus, Mach
    - » dans un environnement de programmation parallèle : PVM, MPI
- **Outils de développement**
  - peu évolués et de bas niveau

Communication par messages
exemple

- **Réalisation d’une interaction de type « client-serveur »**



"Middleware" à messages

- **BQM (Business Quality Messaging)**
  - messages (attributs)
    - » **identification unique**
    - » **typage**
    - » **persistance**
    - » **modes de délivrance (avec ou sans accusé de réception)**
  - queues de messages
    - » **identification unique**
    - » **partagées par les applications**
    - » **modes de réception variable**
  - transactions
    - » **messages vus comme des ressources "transactionnelles"**
  - sécurité
    - » **encodage**
    - » **listes d'accès**

Communication par messages
extensions au modèle de base

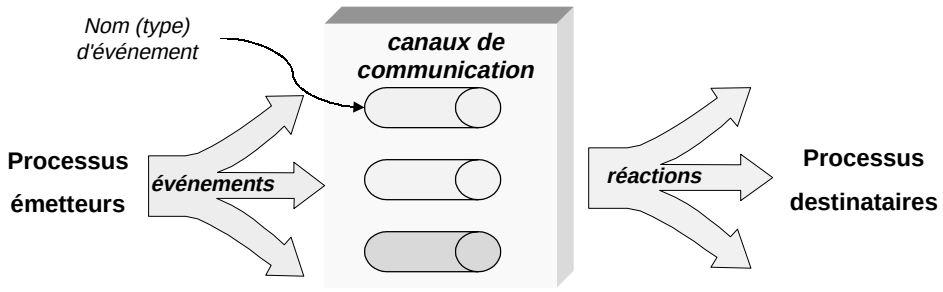
- **Communication de groupe**
  - ***groupe*** : ensemble de récipiendaires identifiés par un nom unique
  - gestion dynamique du groupe : arrivée/départ de membres
  - différentes politiques de service dans le groupe : 1/N, N/N
  - mise en œuvre : utilisation possible de IP multicast
  - exemple : Isis, 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

Modèles d'exécution

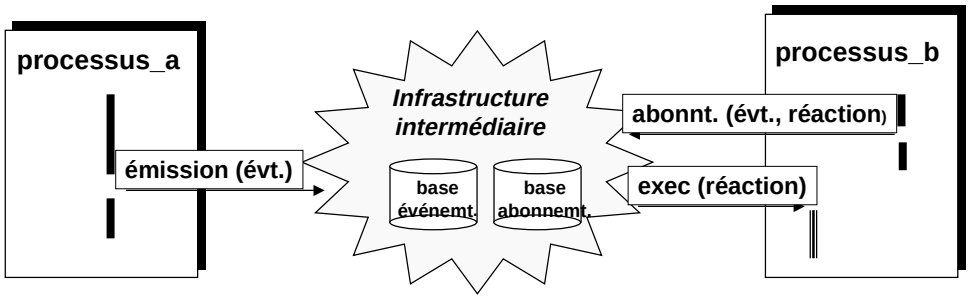
- ❑ **communication par messages**
- ✔ **événements/réactions**
- ❑ **demande de service : *mode client-serveur***
  - ◆ client-serveur "traditionnel"
  - ◆ client-serveur "de données"
  - ◆ client-serveur "à objets"
  - ◆ client-serveur "à composants"
- ❑ **code mobile**
- ❑ **mémoire virtuelle partagée**
- ❑ **autres**
  - ◆ "peer to peer", transactionnel, . .

Communication événementielle : principes de fonctionnement

- ❑ concepts de base : *événements*, *réactions* (traitements associés 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



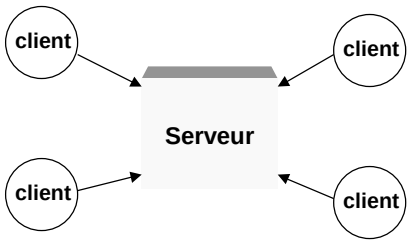
Communication événementielle : principes de réalisation



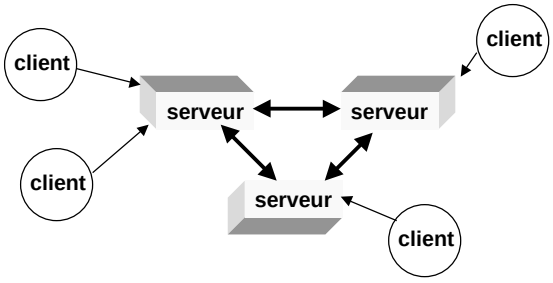
- serveur centralisé ("Hub and Spoke")
- serveur réparti ("Snowflake")
- service réparti (bus logiciel)

Communication événementielle : mise en œuvre (1)

- Serveur centralisé de gestion d'événements

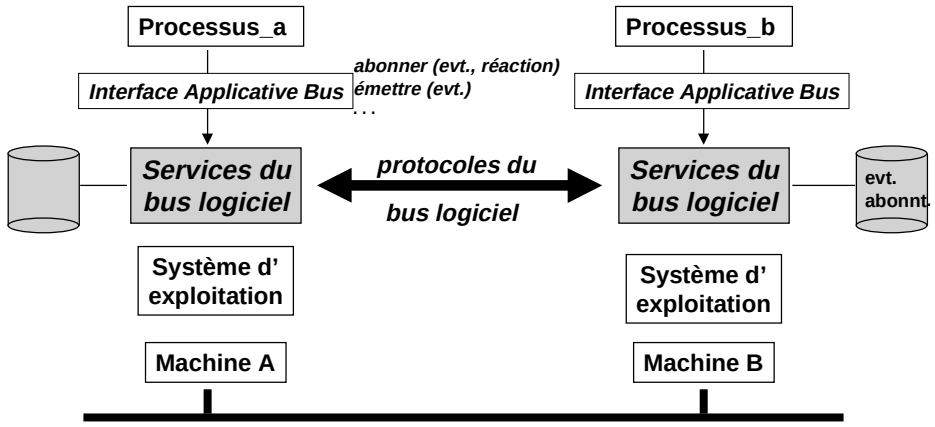


- Serveur réparti de gestion d'événements



Communication événementielle : mise en œuvre (2)

- **Service réparti (bus logiciel)**



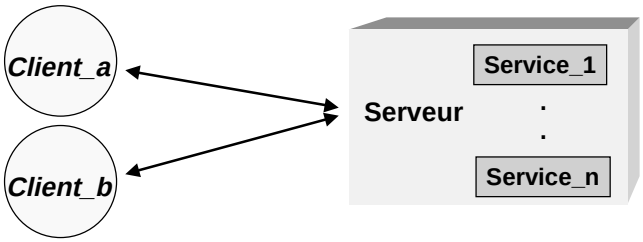
Communication événementielle : conclusion

- **Domaines d'application**
  - génie logiciel (coopération entre outils de développement) : SoftBench, ToolTalk , DecFuse, . .
  - Workflow : KoalaBus, . .
  - diffusion de logiciels et d'information sur le Web : iBus, Castanet, Ambrosia, Smartsockets, TIB/Rendezvous, Active Web,
- **Infrastructures propriétaires**
  - interface applicative et protocoles propres à chaque système ---> problèmes de portabilité et d'interopérabilité
  - effort de normalisation (ECMA)
- **Outils de développement**
  - sommaires

Modèles d'exécution

- ❑ **communication par messages**
- ❑ **événements/réactions**
- ✔ **demande de service : *mode client-serveur***
  - ◆ client-serveur "traditionnel"
  - ◆ client-serveur "de données"
  - ◆ client-serveur "à objets"
  - ◆ client-serveur "à composants"
- ❑ **code mobile**
- ❑ **mémoire virtuelle partagée**
- ❑ **autres**
  - ◆ "peer to peer", transactionnel, . .

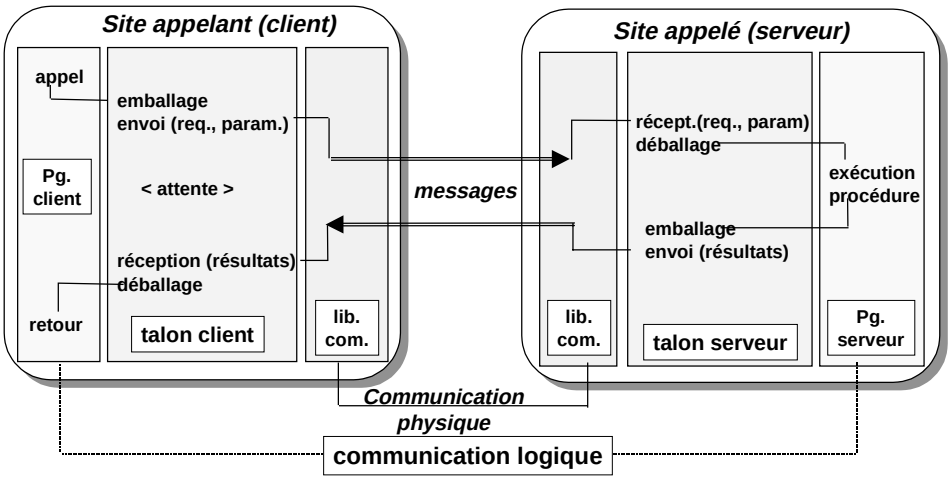
Modèle client-serveur : principes directeurs



Client	Serveur
émet des requêtes (demandes de service) mode d'exécution ***synchrone*** le client est l'initiateur du dialogue client-serveur	ensemble de services exécutables exécute des requêtes émises par des clients --> exécution des services (exécution séquentielle ou parallèle)

Exemples : **serveur de fichiers, serveur de base de données, serveurd'impression, serveur de noms (annuaire des services)**

modèle client-serveur : mise en œuvre
appel de procédure à distance (1)

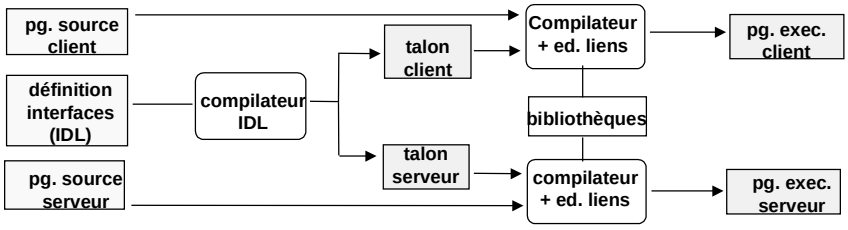


appel de procédure à distance (2)

- **fonction des talons**
  - emballage-déballage des paramètres et résultats
  - conversion de données
  - gestion des processus exécutants (serveur)
  - gestion des erreurs (mise en œuvre "sémantique" RPC)
- **fonction des modules de communication**
  - transmission des paramètres et résultats "emballés"
  - gestion des erreurs de communication
- **liaison** **(détermination de l'adresse de l'appelé)**
  - adresse fixe connue lors de la génération du talon
  - désignation logique + appel d'un serveur de noms

Modèle client-serveur - outils de développement

- **Langages de description d'interface (*IDL*)**
  - spécification commune au client et au serveur
  - définit le type des paramètres et résultats
  - indépendant des langages de programmation
- **Compilateur d'interface**
  - à partir d'une définition d'interface (en IDL) produit les talons client et serveur pour un langage donné



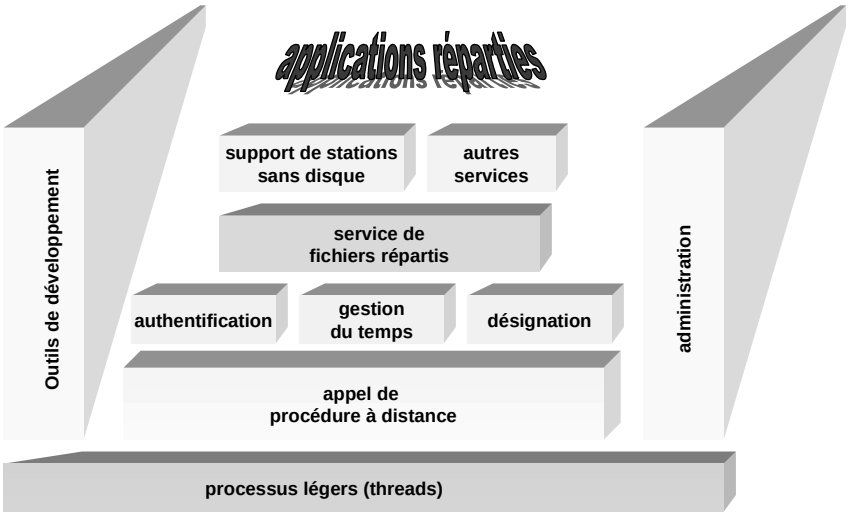
Les limites du modèle client-serveur

- **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

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"**
  - CORBA : un environnement fondé sur un "courtier" d'objets répartis et un ensemble de services applicatifs
  - SPRING : un prototype de système de gestion d'objets répartis
- ❑ **environnement client-serveur "de données"**
  - exécution de requêtes SQL
- ❑ **environnement client-serveur sur le Web**
- ❑ **environnement client-serveur "à composant"**
  - COM: un environnement pour la gestion de documents composites

DCE : architecture générale



DCE : description des services (1)

- **Processus légers (threads)**
  - définis pour assurer la portabilité des applications DCE (hétérogénéité des noyaux de processus légers existants)
- **Appel de procédure à distance**
  - langage de description d'interface et son compilateur
  - mis en œuvre sur TCP/IP
- **Service de noms (Directory Service)**
  - gestion des informations (association nom/attributs) sur les ressources disponibles du système distribué.
  - service structuré en deux niveaux :
    - » **Un espace local à une cellule ou CDS qui maintient les informations locales à une cellule (groupe de machines)**
    - » **Un espace global ou GDS qui permet d'interconnecter les espaces de noms des cellules entre eux dans une même hiérarchie**

DCE : description des services (2)

- **Service distribué de gestion du temps**
  - synchronisation des horloges des différentes machines du système avec le *temps universel coordonné.*
- **Service de sécurité**
  - authentification
  - sécurisation des communications
  - contrôle d'accès (autorisation)
- **Service de fichiers distribués (DFS)**
  - permet le partage des fichiers sans connaître la localisation physique.
    - » **espace de noms global**
  - DCE DFS comprend un système physique de fichiers
    - » **qui permet la duplication, la journalisation et la gestion de listes d'accès.**

Atouts et limites de 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

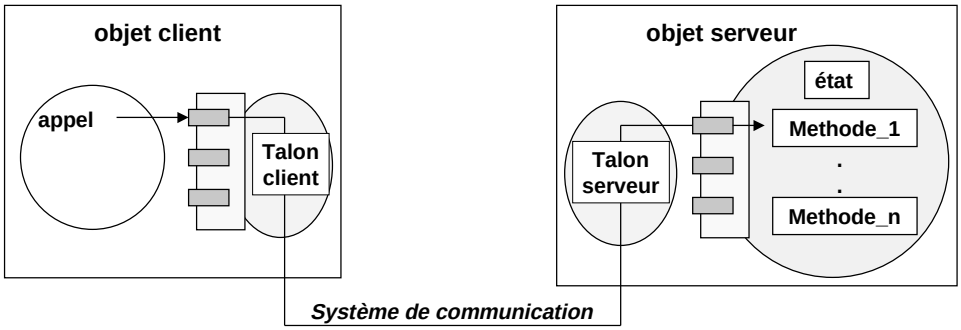
– **développement d’applications “à grain fin” *laborieux***
  - ◆ ex. : utilisation des services de thread et du RPC
    (limitations équivalentes à celles du client-serveur)
  - ◆ peu d’outils de développement disponibles aujourd’hui

Environnements client-serveur

- ❑ **environnement client-serveur "traditionnel"**
- ✔ **environnement client-serveur "à objet"**
- ❑ **environnement client-serveur "de données"**
- ❑ **environnement client-serveur sur le Web**
- ❑ **environnement client-serveur "à composant"**

Client-serveur "à objet"

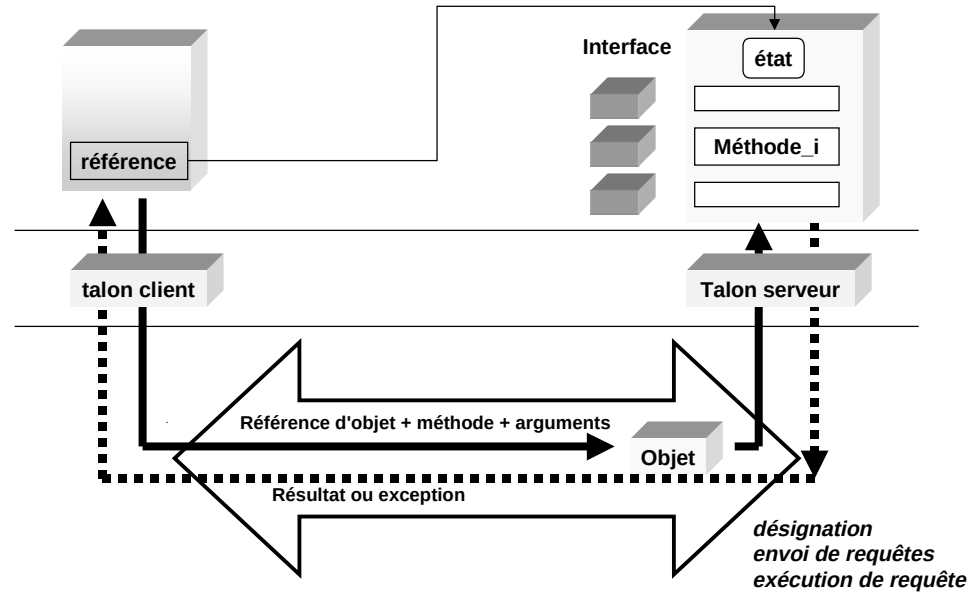
- **Motivations**
  - propriétés de l'*objet* (encapsulation, modularité, réutilisation, polymorphisme, composition)
  - objet : unité de désignation et de distribution
- **Principe de fonctionnement**



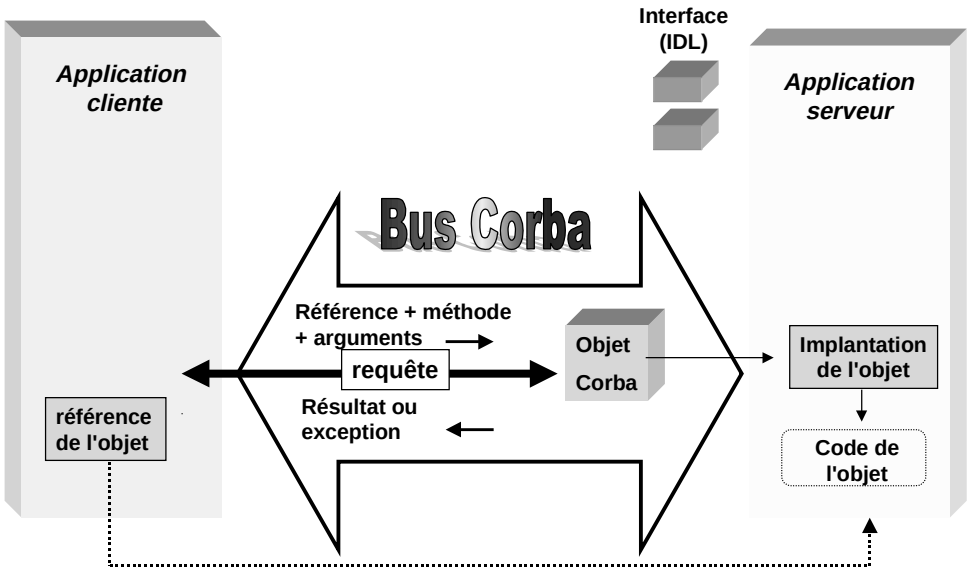
Appel de méthode à distance
Remote Method Invocation (RMI)

- **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

Client-serveur "à objet" : mise en œuvre



Client-serveur à "objet" : bus Corba et objets "Corba"



Corba : Interface Definition Language (IDL)

- L'IDL est un langage de spécification d'interface, supportant l'héritage, non utilisable directement dans les programmes
- L'IDL est utilisé pour générer les talons et les squelettes (stubs and skeletons) et pour définir les interfaces dans le Référentiel d'Interface (Interface Repository)
- L'IDL est indépendant de tout langage de programmation
- La correspondance IDL ---> Langage doit être fournie pour de nombreux languages (initialement C, puis C++ et SmallTalk)
- Les interfaces des objets coopérants sont nécessairement définies en IDL

Corba : invocation "dynamique"

- **Motivations**
  - effectuer une requête sur un service dont l'identité et/ou la structure n'est pas connue à la compilation du programme client (*statiquement*)
- **Principe de réalisation**
  - construction dynamique d'un appel de méthode
    - » recherche d'un objet fournissant le service désiré
    - » recherche des caractéristiques du service (nom de méthode, nombre et type des paramètres, etc.)
    - » construction par programme d'une requête (en utilisant des primitives ad hoc)

Environnements intégrés : systèmes répartis à objets (OOOS)

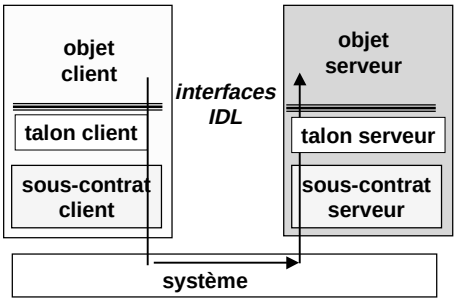
- **Intégration au niveau système**
  - applications et services de base sont définis comme des objets
  - langage(s) de programmation + IDL
  - le système gère directement des objets (Object-Oriented OS)
  - exemple: Spring
- **Intégration au niveau système + langage**
  - langage unique pour la description des objets et de la distribution
  - le système est l'environnement d'exécution du langage
  - exemple : Guide-Oode

Spring : principes directeurs

- **application = ensemble d'objets**
  - "*tout est objet*" : composants d'une application et services applicatifs
  - interface décrite en IDL
  - implémentation dans un langage quelconque
    (en fait C/C++ : mapping entre l'IDL et le langage de programmation)
  - héritage des interfaces, mais pas des implémentations
  - sous-contrat : spécialisation de l'appel de procédure à distance
- **système : gestionnaire d'objets distribués**
  - ORB
  - services de base "intégrés" avec l'ORB
    (pour la gestion des activités et des objets)

Spring : mécanisme d'appel d'objet

- **Sous-contrats**
  - permet de préciser la sémantique de l'appel de méthode
  - différentes implémentations d'un même type peuvent avoir des sous-contrats différents
  - **objet = { variables d'état**
    table de méthodes
    opérations du sous-contrat }
  - compatibilité des sous-contrats



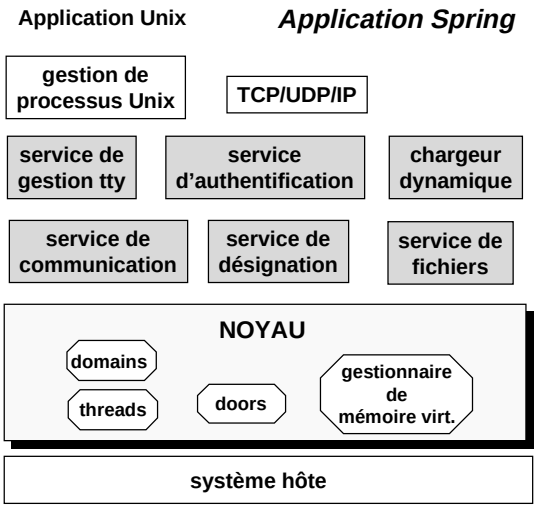
Exemples :

réplication transparente : n copies d'un objet serveur avec des protocoles variés de mise en cohérence

appel fiable : protocoles de reprise après panne

cache "client" : gestion d'un cache local

Spring : architecture



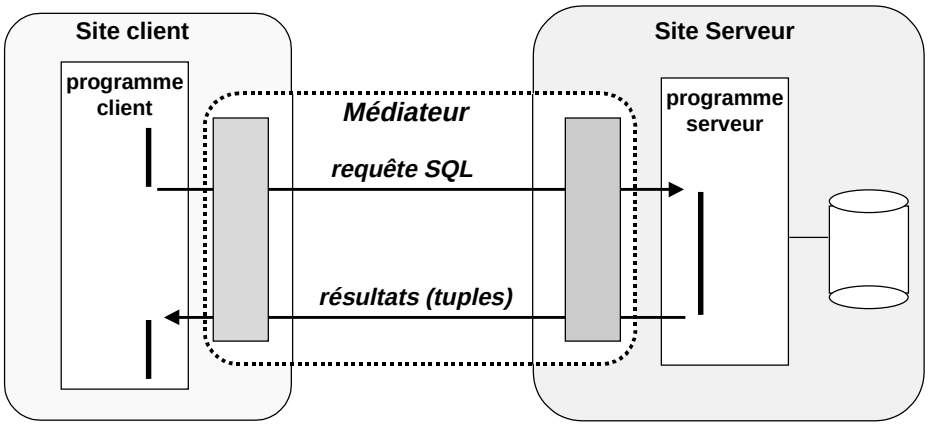
Spring : bilan

- **prototype expérimental (“véhicule de recherche”)**
  - disponible sur machine Sun
  - interopérabilité avec Solaris/ONC+
  - environnement de développement C++
  - système de fichiers local et client NFS
- ☛ **une “source” de DOE**
  (Distributed Objects Everywhere)
- ☛ **une “source” de Java**
  - langage Java
  - système JavaOS
  - système d'objets Java répartis Jini

Environnements client-serveur

- **environnement client-serveur "traditionnel"**
- **environnement client-serveur "à objet"**
- ✔**environnement client-serveur "de données"**
- **environnement client-serveur sur le Web**
- **environnement client-serveur "à composant"**

le client-serveur de données
principes directeurs (1)



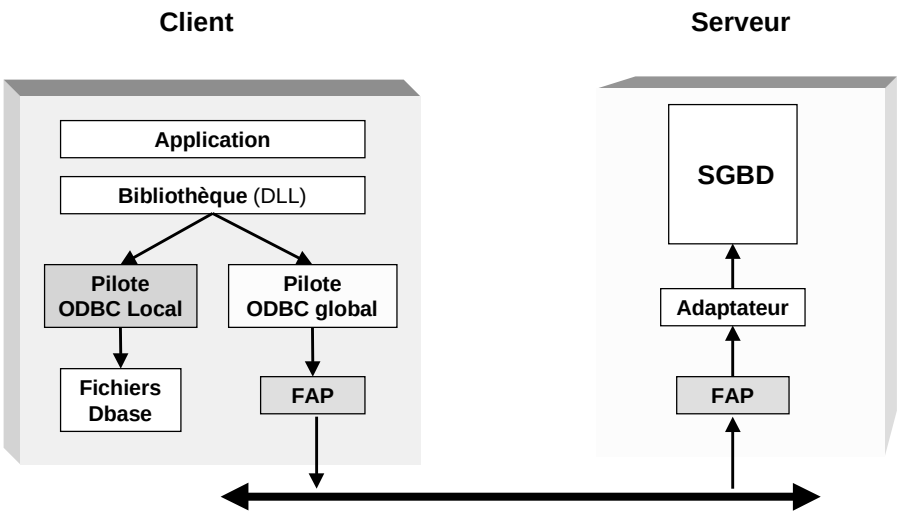
Le client-serveur de données
principes directeurs (2)

- **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)

Le client-serveur de données
environnement applicatif

- **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 de l'application
    (schémas de données, code des opérations, enchaînement, . . .)

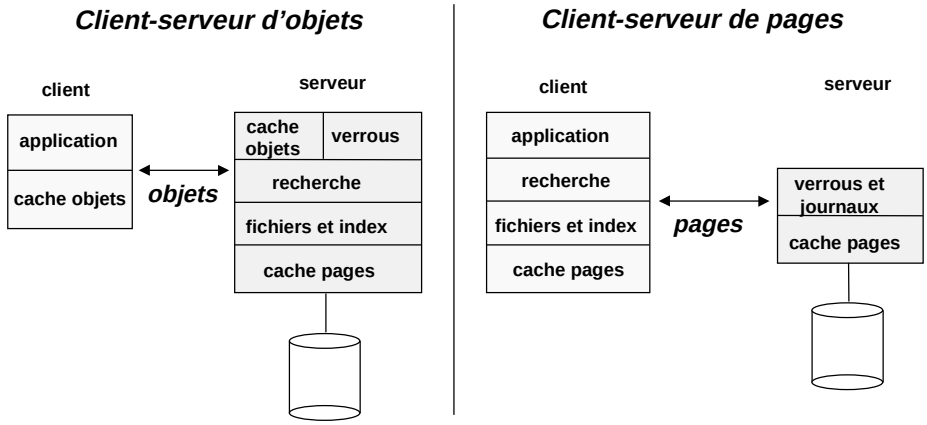
Client-serveur de données :
la solution ODBC



Client-serveur de données "à objet" : la solution 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

SGBDO : architecture client-serveur



Environnements client-serveur

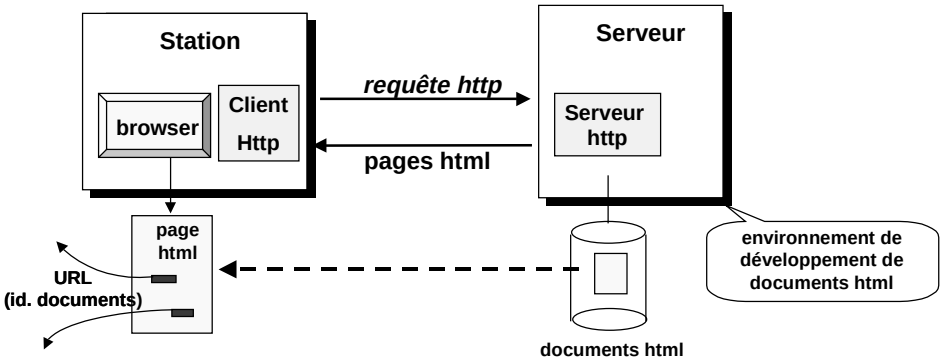
- ❑ **environnement client-serveur "traditionnel"**
- ❑ **environnement client-serveur "à objet"**
- ❑ **environnement client-serveur "de données"**
- ✔ **environnement client-serveur sur le Web**
- ❑ **environnement client-serveur "à composant"**

Environnements client-serveur sur le Web

- ❑ **le Web "traditionnel"**
  accès à l'information répartie
- ❑ **le Web comme support d'applications client-serveur**
  protocole d'échange de "haut niveau"
- ❑ **les documents "actifs"**
  code mobile (ex. "applets" Java)
- ❑ **le Web et les objets distribués**
  convergence de 2 technologies

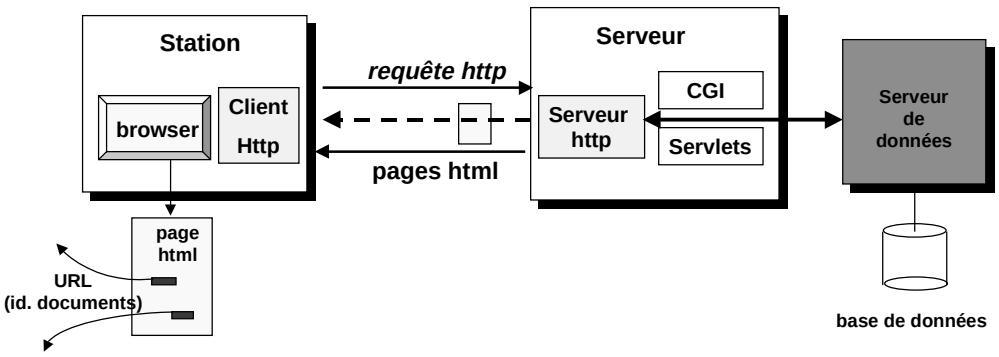
le Web : support pour l'accès à l'information (1)

- un protocole client-serveur (mis en œuvre sur TCP/IP) : http
- un système de désignation universel : URL
- une représentation de documents hypertextes : html
- *des navigateurs ("browser") évolués*



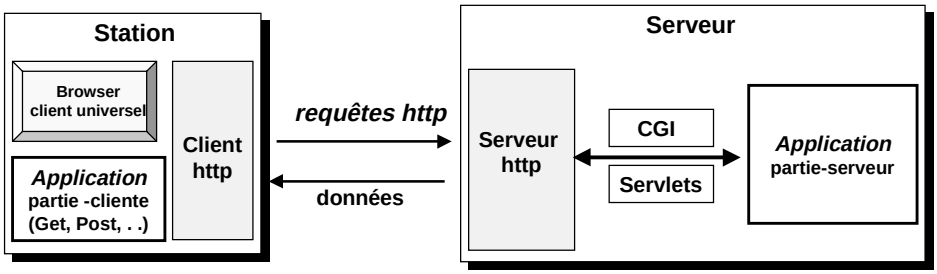
le Web support pour l'accès à l'information (2)

- **Accès à des serveurs "externes"**
  - utilisation de l'interface CGI (Common Gateway Interface) et/ou de "servlets"
    - » **requêtes à un serveur de données**
    - » **transformation des résultats en pages html**



le Web comme support d'applications structurées selon le modèle client-serveur

- utilisation du protocole http pour le dialogue client-serveur

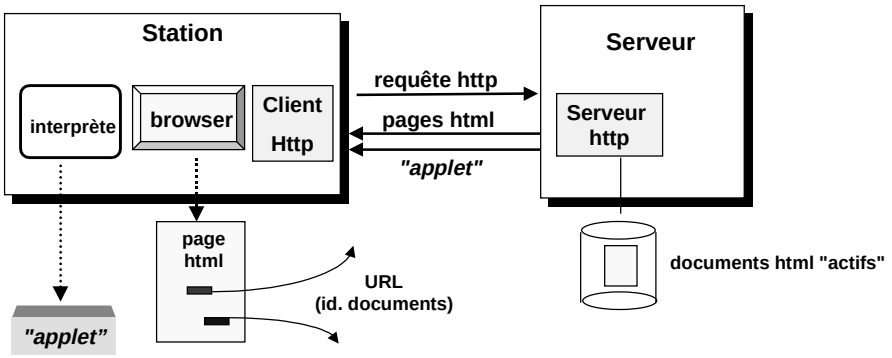


+ ***avantages*** :
  support universel de communication
  simplicité de la mise en œuvre

- ***inconvénients*** :
  primitives élémentaires (Get, Post, . .)
  serveur http non adapté

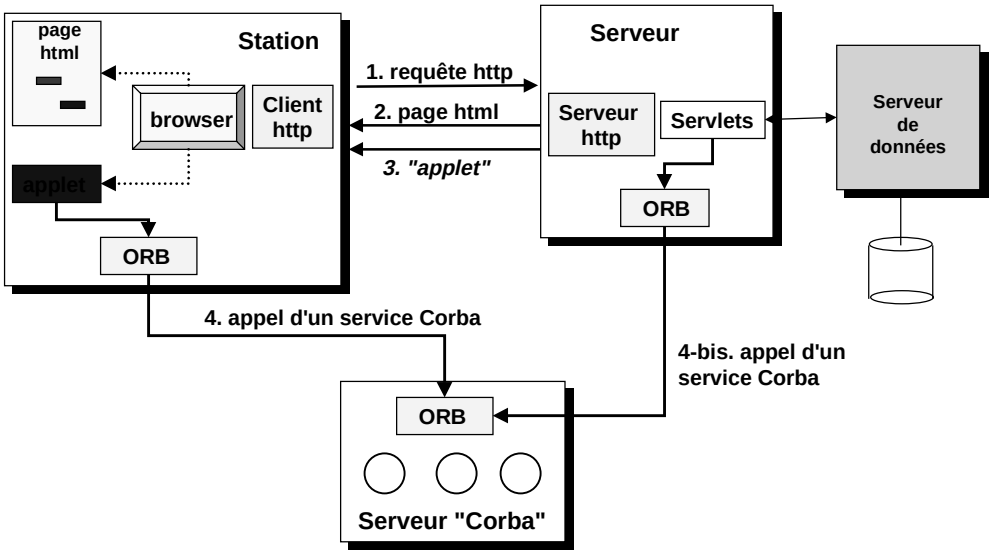
Web : documents "actifs"

- séquences exécutables ("applets", "contrôles") dans les pages html



☛ **sécurité : contrôle des actions exécutées par le code importé**

Le Web et les objets distribués



Environnements client-serveur

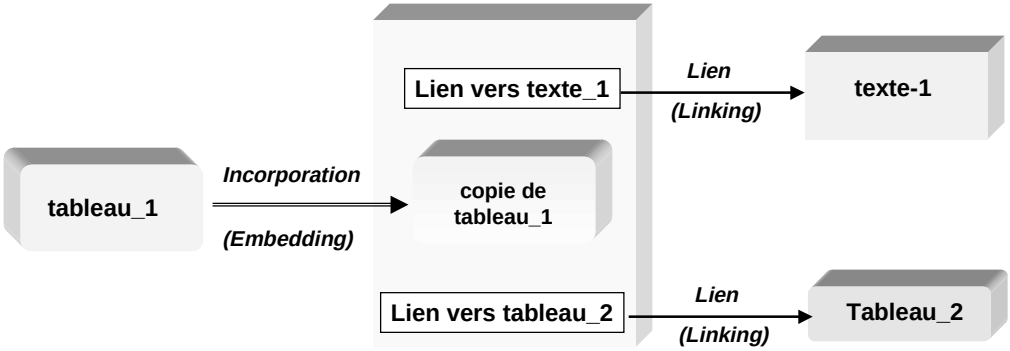
- ❑ environnement client-serveur "traditionnel"
- ❑ environnement client-serveur "à objet"
- ❑ environnement client-serveur "de données"
- ❑ environnement client-serveur sur le Web
- ✔ environnement client-serveur "à composant"

Client-serveur 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
    - » **ex. document composé de texte (word), tableaux (excel), données extraites d'une base de données, données trouvées sur le Web**
- **Approche**
  - modèle et mécanismes pour réaliser l'interopérabilité entre "composants" logiciels
    - » ***réutilisation de composants binaires***
    - » coopération entre composants
      - interaction de type client-serveur
      - format de données "pivot" et procédures de conversion
      - règles d'échange entre composants

Exemple : gestion de documents composites

- **OLE (Object Linking and Embedding)**



Client-serveur à "composants" : principes directeurs

- **principes directeurs**
  - définition d'un environnement à base "d'objets" pour la représentation et la manipulation des composants logiciels
    - » modèle à objets
    - » environnement d'exécution
  - la définition des services de gestion des documents composites utilise les fonctions de l'environnement sous-jacent
  - les fonctions de l'environnement peuvent être utilisées "directement" pour réaliser la coopération entre des composants logiciels

- **exemples :**

  OpenDoc - SOM/DSOM (IBM & Apple)

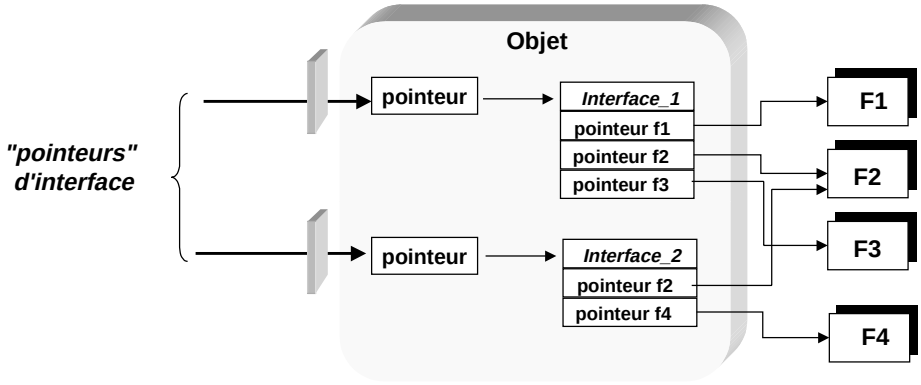
  ✔ OLE - COM/DCOM/ActiveX (Microsoft)

COM : principes directeurs (1)

- **Concepts**
  - **objet/composant** : encapsulation de données, manipulées par des fonctions
  - **interface(s)** d'un objet/composant : ensemble de fonctions d'accès à un objet/composant
    - » un composant peut avoir plusieurs interfaces
    - » les interfaces sont non modifiables ("immutable")
    - » les interfaces sont désignées par un identificateur global GUI : Global Unique Identifier (128 bits)
  - **classe** : ensemble de services réalisés par un serveur (EXE) ou une bibliothèque (DLL)
    - » un objet/composant peut faire référence à plusieurs classes

COM : principes directeurs (2)

❑ **Représentation d'un objet composant**



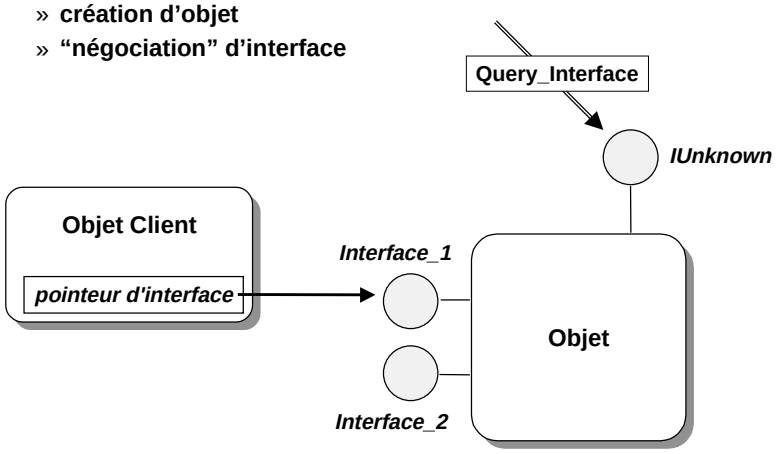
COM : principes directeurs (3)

❑ **relation entre objets**

◆ un pointeur d'interface peut être obtenu à partir :

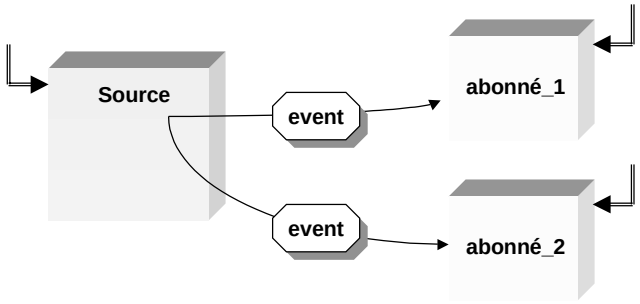
» **création d'objet**

» **"négociation" d'interface**



Programmation par composants JavaBeans

- **Bean : objet Java présentant certaines caractéristiques syntaxiques**
  - attributs publics pouvant être positionnés de l'extérieur :
    - » par d'autres Beans
    - » par des outils
  - modèle de communication de type événement/réaction

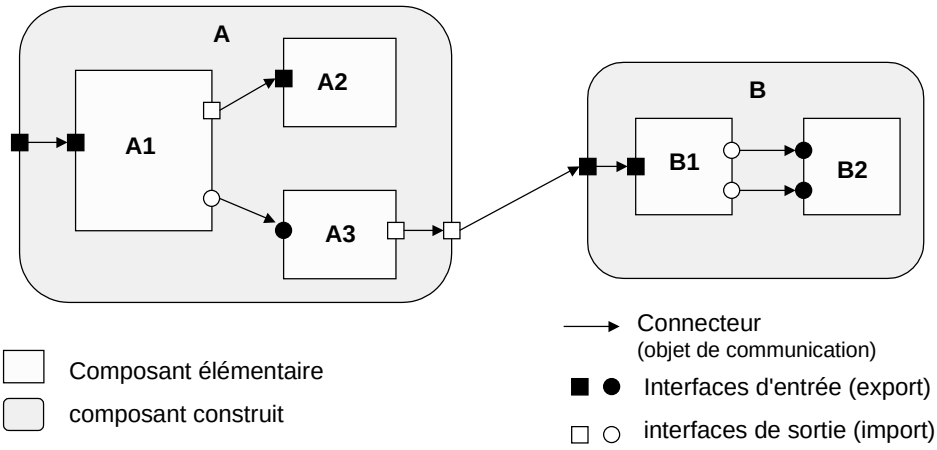


Programmation constructive

- **Motivation : réutilisation de logiciel**
  - intégration de modules logiciels existants
  - ➥ construction d'applications réparties par assemblage de modules logiciels existants
  - ☛ *programmation à gros grain ("programming in the large")*
- **Approche : description de l'architecture de l'application à l'aide d'un langage déclaratif**
  - composant : interfaces, attributs, implémentation
  - modèle de construction des composants
  - description des interactions entre composants (connecteurs)
  - description de variables d'environnement (placement, regroupement, sécurité, etc.)
  - ☛ *langages de configuration*

Programmation constructive : MIL, ADL

MIL (Module Interconnection language)
ADL (Architecture Description language)



Modèles d'exécution

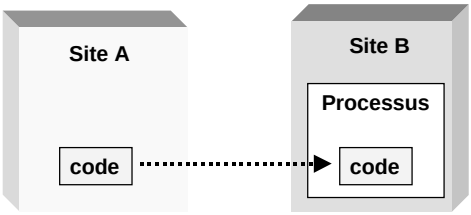
- ❑ **communication par messages**
- ❑ **événements/réactions**
- ❑ **demande de service : *mode client-serveur***
  - ◆ client-serveur "traditionnel"
  - ◆ client-serveur "de données"
  - ◆ client-serveur "à objets"
  - ◆ client-serveur "à composants"
- ✔ **code mobile**
- ❑ **mémoire virtuelle partagée**
- ❑ **autres**
  - ◆ "peer to peer", transactionnel, . .

Code mobile

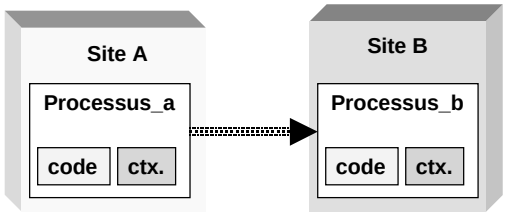
- **Définition**
  - programmes pouvant se déplacer d'un site à un autre
  - exemples :
    - » requête SQL
    - » "applet" Java
- **Motivations**
  - rapprocher le traitement des données
    - » réduire le volume de données échangées sur le réseau
    - » partage de charge
  - ✔function shipping versus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

Modèles d'exécution pour la mobilité

- **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



Systèmes d'agents mobiles

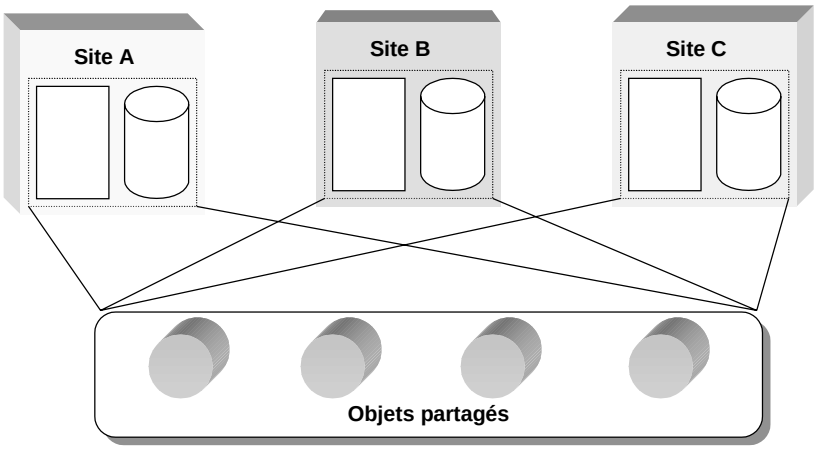
- **Modèle de programmation**
  - **agents** : unité de structuration d'une application (stationnaires ou mobiles)
  - **places** : endroit visité par un agent
  - **déplacement** : changement de place
  - **interaction** entre agents :
    - » **co-localisés (notion de "meeting")**
    - » **distants (par message)**
- **Mise en œuvre : problèmes**
  - gestion de l'état d'un agent mobile
  - gestion des canaux de communication entre agents mobiles
- **exemples**
  - Java : Aglets (IBM), Odyssey (General Magic), MOA (OpenGroup)
  - autres langages : Telescript (General magic), AgentTCL (U. Dartmouth) Tacoma (U. Cornell : environnement multi-langages)

Modèles d'exécution

- **communication par messages**
- **événements/réactions**
- **demande de service : *mode client-serveur***
  - client-serveur "traditionnel"
  - client-serveur "de données"
  - client-serveur "à objets"
  - client-serveur "à composants"
- **code mobile**
- ✔**mémoire virtuelle partagée**
- **autres**
  - "peer to peer", transactionnel, . .

Systèmes à "mémoire virtuelle partagée" (1)

- **Principes de fonctionnement**
  - "simulation" d'une mémoire globale (d'objets) partagée



Systèmes à mémoire virtuelle partagée (2)

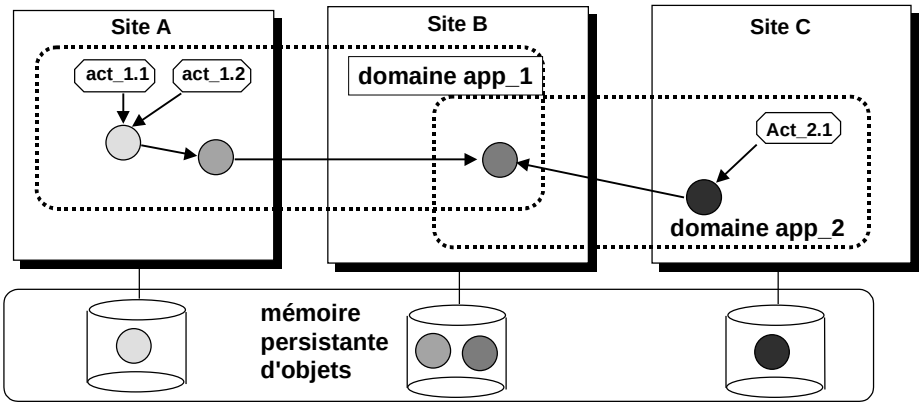
- **Principes de mise en œuvre**
  - schéma d'exécution
    - » objets "actifs" versus objets "passifs"
    - » migration des objets versus migration du contrôle
  - désignation
  - gestion du partage : synchronisation, cohérence
    - » image unique d'un objet versus copies (cohérentes) d'un objet
  - gestion de la persistance
- **Interface de programmation**
  - langage de programmation d'applications réparties intégrant distribution, parallélisme, synchronisation, persistance, etc.
  - extension d'un langage existant (pre-processeur) ou langage ad-hoc
  - ☛ les services du système constituent l'exécutif du langage

Étude de cas : Guide/Oode

- **Objectif**
  - faciliter la programmation et la mise au point d'applications réparties
- **principes directeurs**
  - modèle de structuration d'application
    - » **application = {objets coopérants]**
    - » **objets partagés, distribués et persistants**
    - » **modèle d'exécution avancé (distribution, parallélisme, communication, synchronisation, exceptions, etc.)**
  - langage de programmation
    - » **Guide : langage orienté objet "distribué" et "persistant" ("à la Java")**
    - » **OC++ : C++ + {persistance et distribution}**
  - intégration "forte" langage et système
    - » **le système (OOOS) est l'environnement d'exécution du langage**
    - » **le système est extensible : les nouveaux services sont définis en termes d'objets**

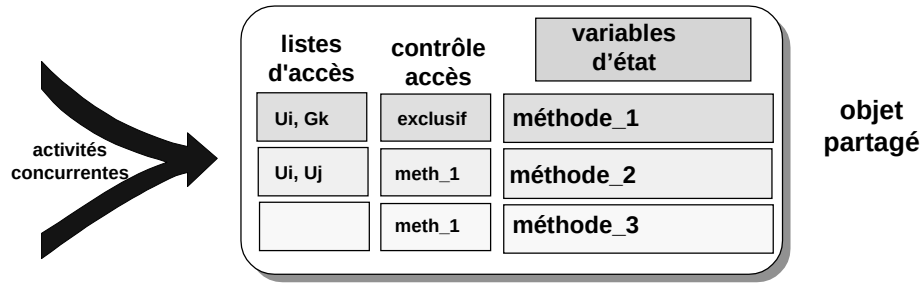
Guide/Oode : modèle d'exécution (1)

domaine : espace d'adressage distribué ("processus Unix distribué")
activité : flot de contrôle distribué ("thread distribué")
une activité exécute des appels de méthode sur des objets
liaison dynamique des objets (et choix du site d'exécution)
partage des objets entre activités (d'un domaine ou de plusieurs domaines)



Guide/Oode : modèle d'exécution (2)

- **Partage d'objets**
  - le modèle de programmation fournit l'abstraction d'une mémoire commune d'objets
    - » **communication par objets partagés**
    - » **synchronisation des accès concurrents**
    - » **protection**

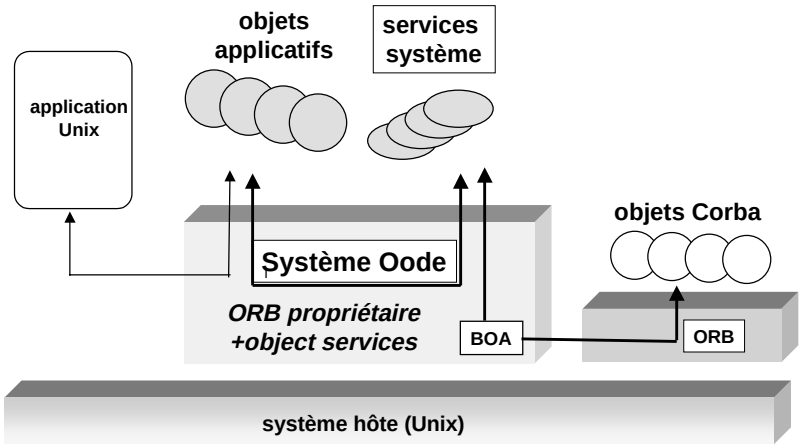


Oode : environnement de développement

- **OC++ : un langage à objets persistant et réparti**
  - extension de C++ : **mot clé supplémentaire *"distributed"***
  - définition des types conforme à l'IDL de CORBA
  - pré-processeur : génération de code C++
  - accès aux bibliothèques usuelles
  - ☛ un seul langage pour l'écriture de l'application, et pour la gestion de la distribution et de la persistance

- **Outils de développement**
  - metteur au point distribué
  - "fouineur" de classes
  - outils de configuration et d'administration

Oode : architecture

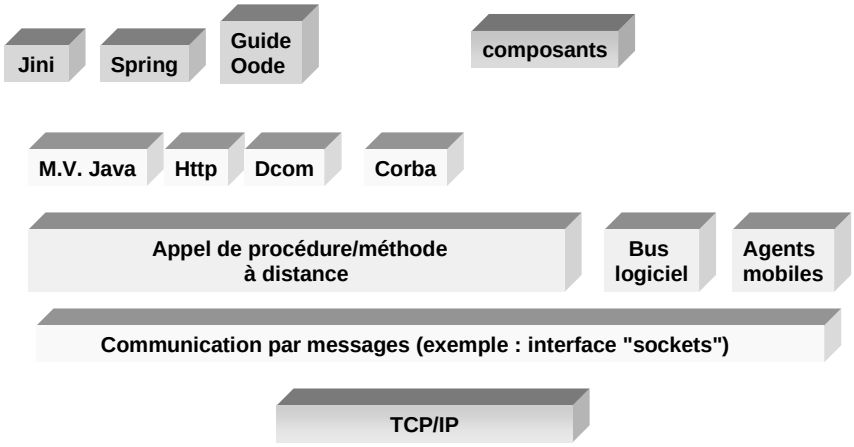


Oode : bilan

- **prototype expérimental**
  - disponible sur AIX/DCE et Solaris
  - interopérabilité "partielle" avec un environnement Corba
- **avantages**
  - gain de productivité pour le développement d'applications réparties
    - » **modèle d'exécution**
    - » **distribution et persistance non gérées par le programmeur**
    - » **langage de programmation "intégré"**
  - système extensible
    - » **nouveaux services construits comme des objets**
- **limitations**
  - langage non conventionnel (C++ étendu)
  - intégration des applications existantes
  - déploiement des applications

Synthèse

☛ **Hiérarchie de modèles**



Quelques références bibliographiques (1)

S. Mullender, "Distributed Systems", Addison-Wesley, 1989

G. Coulouris, J. Dollimore, T. Kindberg, "Distributed Systems : Concepts and Design", 2nd Edition, Addison-Wesley, 1994

A. Tanenbaum, "Systèmes d'exploitation : systèmes centralisés, systèmes distribués", InterEditions, 1994

R. Balter, J.-P. Banâtre, S. Krakowiak, "Construction des systèmes d'exploitation répartis", Collection Didactique de l'INRIA, 1991

A. Birrell, B. Nelson, "Implementing Remote Procedure Calls", ACM Trans. on Computer Systems, Vol. 2, 1984

S. Reiss, "Connecting Tools using message passing in the FIELD environment", IEEE Software, juillet 1990

J. Arnold, G. Memmi, "Intégration par le contrôle et son rôle dans l'intégration de logiciels", Proc. of ICSE'92, Toulouse, Décembre 1992

Quelques références bibliographiques (2)

W. Rosenberg, D. Kenney, G. Fisher, “Comprendre DCE”,
Addison-Wesley, 1993

http://www.osf.org/dce

http://www.omg.org
http://www.omg.org/corba.htm

J.-M. Geib, C. Gransart, P. Merle, "Corba : des concepts à la pratique", InterEditions, 1997

http://www.odmg.org

R. Cattell, “Object Databases : The ODMG-93 Standard”,
Morgan & Kaufman Ed., 1993

G. Gardarin, O. gardarin, "Le client-serveur", Eyrolles, 1996

http://www.microsoft.com

http://www.javasoft.com

Quelques références bibliographiques (3)

http://www.sun.com/tec/projects/spring/index.html

J. Mitchell & al. “An overview of the Spring System, Proc. of Compcon Spring 1994, Feb 94

G. Hamilton, J. Mitchell, M. Powell, “Subcontract; a Flexible Base for Distributed programming”,
Proc. of 14th Symposium on Operating Systems Principles, Dec. 93

http://sirac.imag.fr

R. Balter, S. Krakowiak, "Retrospective sur le projet Guide : un environnement à base d'objets pour applications réparties, L'Objet, Vol. 3 (2), juin 1997