Tutoriel "Objets Distribués, Interopérabilité, CORBA"

Nacer Boudjlida

Objets distribués

Tutoriel BDA'98, Hammamet, Tunisie, 26 cotobre 1998

Nacer Boudjlida

LORIA,UHP Nancy 1
Campus scientifique, BP 239
54506 Vandoeuvre Lès Nancy CEDEX

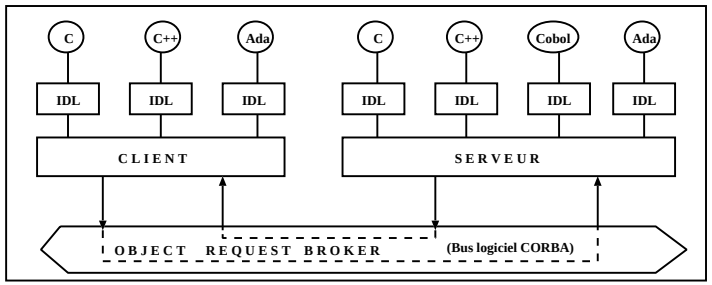
Nacer.Boudjlida@loria.fr ou nacer@loria.fr

http://www.loria.fr/~ nacer/

Common Object Request Broker Architecture

- *http://www.omg.org*
- Projet "middleware" Client/Serveur
- Object Management Group (OMG) [$\simeq$ 700 Compagnies *sauf* Microsoft (DCOM)]
- Proposition de *spécifications*, "pas de code" :
  - Standards pour l'intégration d'applications distribuées
  - Technologie objets
  - Architecture (Object Management Architecture) : logiciel $\simeq$ objet
  - Applications non orientées objets (*Legacy Systems*) : encapsulation $\longrightarrow$ comportement conforme à CORBA

- *Objet distribué CORBA* : {Composants binaires accessibles à distance par des clients}
- *Interface Definition Language* (IDL)
  - "Langage neutre"
  - "Bindings" vers C, C++, Smalltalk, Java, Cobol, ...



- *Pour le Client* : *TRANSPARENCE* vis-a-vis :
  - Langage d'implémentation des objets du serveur
  - *Localisation*
  - *Systèmes d'exploitation*
  - *Seule connaissance* : Interface(s) publiée(s)
- *Mots-clés* :
  - Client-Middleware-Serveur
  - Hétérogénéité (langages, systèmes, plates-formes)
  - Objets
  - Distribution

Structure de la présentation

I. Les architectures client/serveur

- Typologie
- Composants et technologies

II. Objets Distribués et CORBA

- Anatomie
- Notions de programmation en CORBA
- Client/Serveur, Web et CORBA
- CORBA vs [D]COM

III Conclusion

Partie I : Les architectures client/serveur

I- Notions de Client/Serveur

- *Application "Client"* : demande(s) de services
- *Application "Serveur"* : fournisseur de services

- *Machine Client* : résidence de l'application client
- *Machine(s) Serveur* : résidence du (des) fournisseur(s) de services
- *Machine Client et machine serveur* : unique ou pas

- Un client *peut aussi* être serveur

Client/Serveur : Définition

- Environnement *ouvert* réunissant des systèmes éventuellement *hétérogènes* avec :

1. *Partage de ressources* :
   - Relation 1 (Serveur)-N (Clients)
   - Contrôle des accès aux ressources

2. *Asymétrie des protocoles*
   - *Serveur* : attente passive des demandes
   - *Client* : initiateur du dialogue

3. *Transparence*
   - Localisation des services
   - Unicité/Multiplicité des machines

4. *Indépendance matériel-logiciel* : Logiciels client et serveur indépendants
   - de la plateforme
   - du système d'exploitation

5. *Faible couplage* : Communication par messages
   - C $\longrightarrow$ S : Demande de service
   - S $\longrightarrow$ C : Réponses aux demandes

6. *Encapsulation des services*
   - Services connus par leur interface

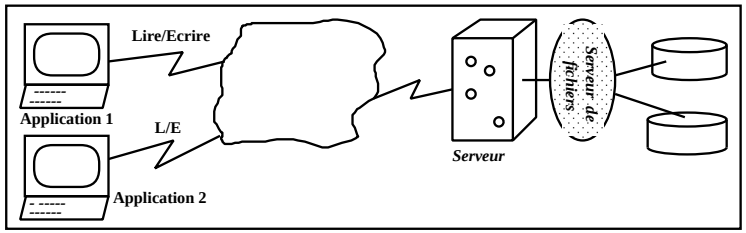
7. *Evolutivité*
   - *Horizontale* : Ajout/Retrait de machines clients
   - *Verticale* : Migration du serveur sur des machines plus puissantes ou vers une architecture multi-serveurs

8. *Intégrité et Maintenabilité*
   - Code et données gérés de façon centralisée par le serveur
   - Clients indépendants (isolation)

Typologie des architectures Client/Serveur

1. Serveurs de fichiers
2. Serveurs Bases de données
3. Serveurs de "Groupware"
4. Serveurs de transactions
5. Serveurs d'objets
6. Serveurs Web
7. Serveurs Objets + Web

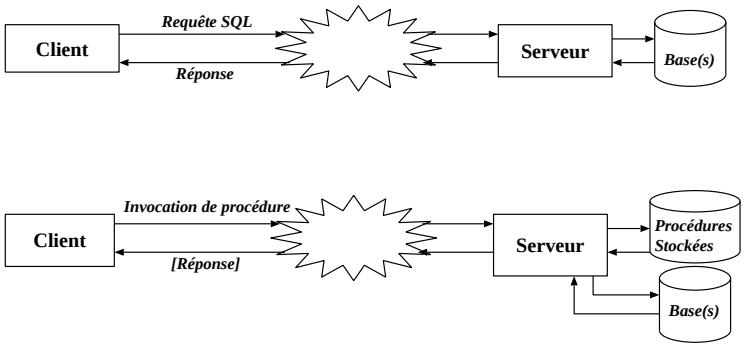
Typologie (1/7) : Serveurs de fichiers



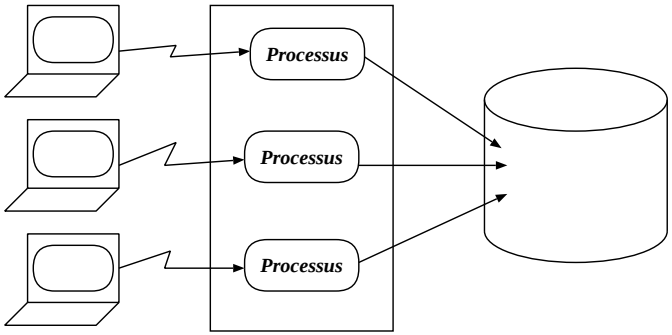
Typologie (2/7) : Serveurs bases de données

- Similaires aux serveurs de fichiers
- Client $\longrightarrow$ Serveur : Requêtes SQL ou invocation de procédure stockée
- Serveur $\longrightarrow$ Client : Réponses

Serveurs bases de données : SQL distant vs procédures stockées

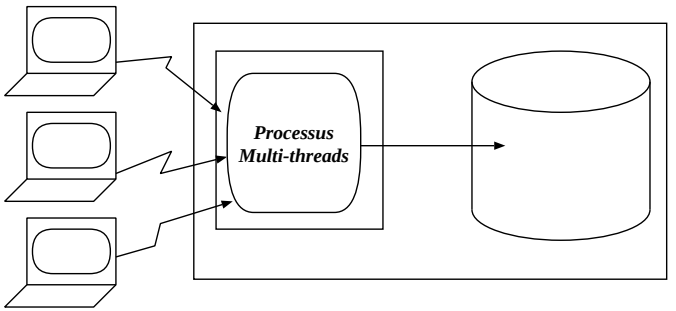


Serveurs bases de données : Un client-Un processus



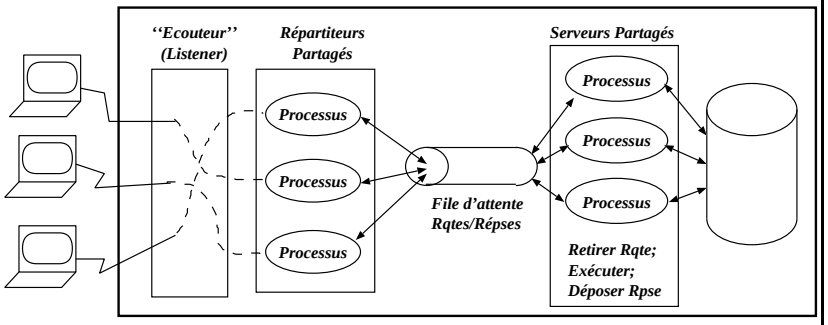
- *Exemple* : Oracle V6

Serveurs bases de données : N clients, M processus (N > M)



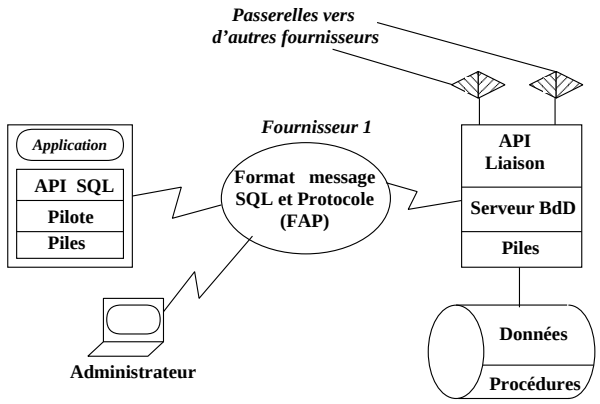
- *Exemple* : Sybase, SQL Server

Serveurs bases de données : Architecture hybride

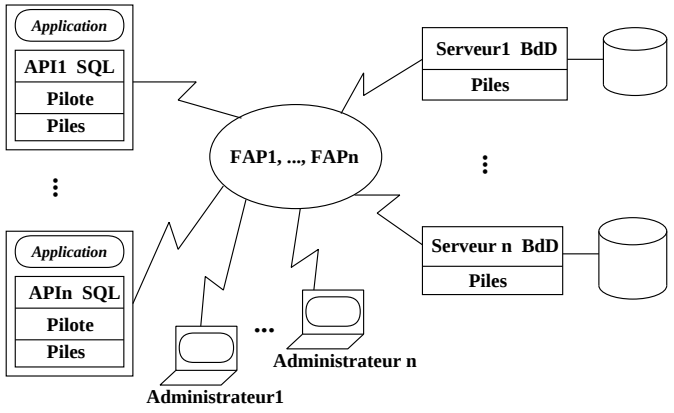


- *Exemple* : Oracle 7

Bases de données Fédérées Homogènes



Bases de données Fédérées Hétérogènes



Bases de données Fédérées Hétérogènes

- Hétérogénéité sans véritable interopérabilité
- *Solutions* :
  1. API SQL commune + pilotes spécifiques $\Longrightarrow$ Différents FAPs
  2. API SQL commune + pilote passerelle + Passerelle SQL ouverte
     $\Longrightarrow$ Un FAP (pour la passerelle)
  3. + FAP SQL standard

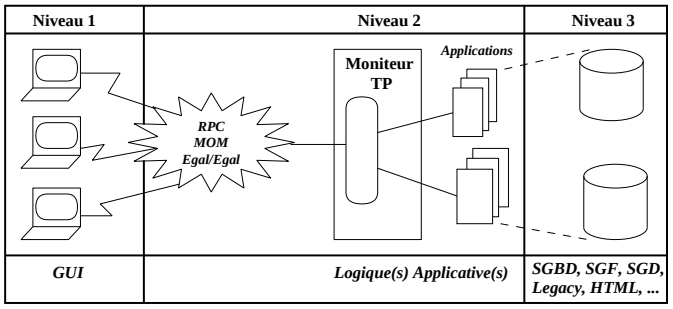
Typologie (3/7) : Serveurs de groupware

- Communication entre individus
- Information semi-structurée : images, textes, messagerie, etc
- Exemple : Lotus Notes

Typologie (4/7) : Serveurs de transactions

- *Transaction* : suite d'actions considérée comme *atomique*
- "Tout ou rien"
- Client $\longrightarrow$ Serveur : $\simeq$ invocation de procédures distantes
- *Client* dispose, généralement, d'une GUI
- *Serveur* :
  - {Transactions}
  - éventuellement, un moniteur transactionnel

Moniteurs transactionnels (1/2)



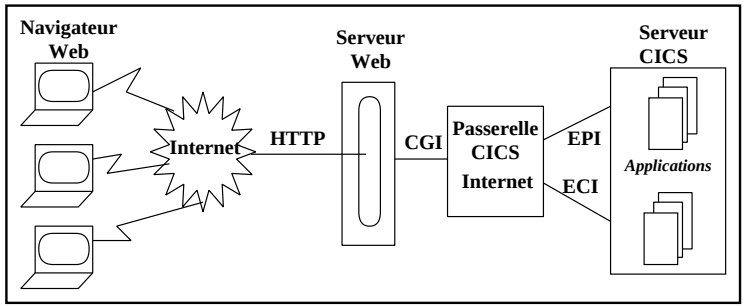
Moniteurs transactionnels (2/2)

- $\simeq$ système d'exploitation pour la gestion de transactions :
  - Lancement processus serveurs
  - Routage vers les serveurs
  - Contrôle de l'exécution
  - Equilibrage de charge entre processus
  - Gestion de transactions : ACID (au minimum)
- Moniteurs "ouverts" : *Encina* (Transarc/IBM), *TopEnd* (AT&T), *Tuxedo* (BEA/Novell)
- Moniteurs "fermés" : *CICS/MVS*, *CICS/TP* (IBM), *Pathway* (Tandem)

Moniteurs transactionnels (3/3)

- Avantages et tendances :
  - Agents de communication "universels" (Workflow, Internet, etc)
  - *"Entente cordiale"* avec des outils de développement C/S (PowerSoft, Delphi, etc) : transparence moniteur-développeur
  - *Gain économique* : par rapport aux serveurs BdD
    - $\simeq$ 30% sur le coût total du système
    - $\simeq$ 40 à 50% sur le coût de développement
  - $\longrightarrow$ Environnements de développement d'applications portables
  - $\longrightarrow$ Orientation objets : services accessibles via des bibliothèques C++ (mutation en ORB ?)

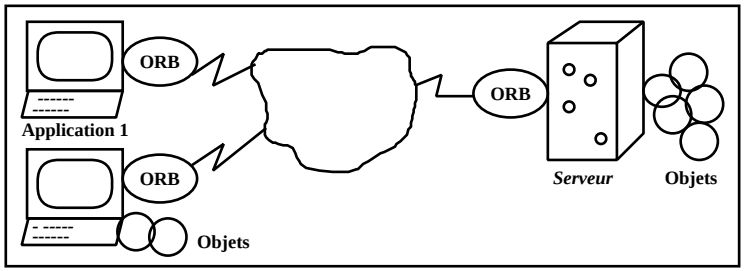
Moniteurs transactionnels + Web



- EPI : External Presentation Interface (émulation terminaux clients)
- ECI : External Call Interface (Transactional RPC)

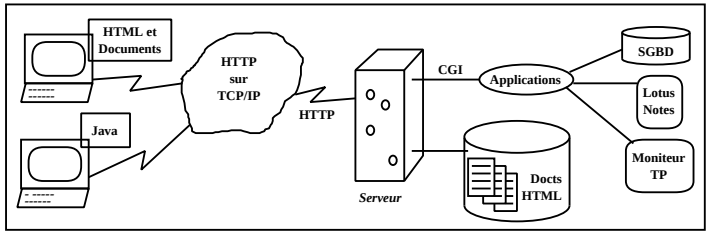
Typologie (5/7) : Serveurs d'objets

- *Application* = {Objets communicants}
- Communication par le biais d'un *négociateur de requêtes objets (Object Request Broker)*



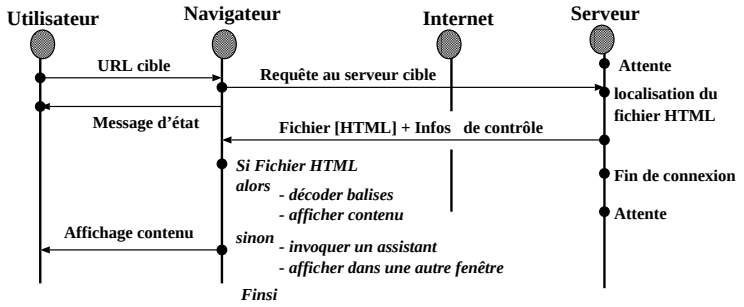
Typologie (6/7) : Serveurs Web

- *Clients* : "légers", portables communicant avec de gros serveurs
- *Communication* : HTTP
- *Convergence* Objets-Web

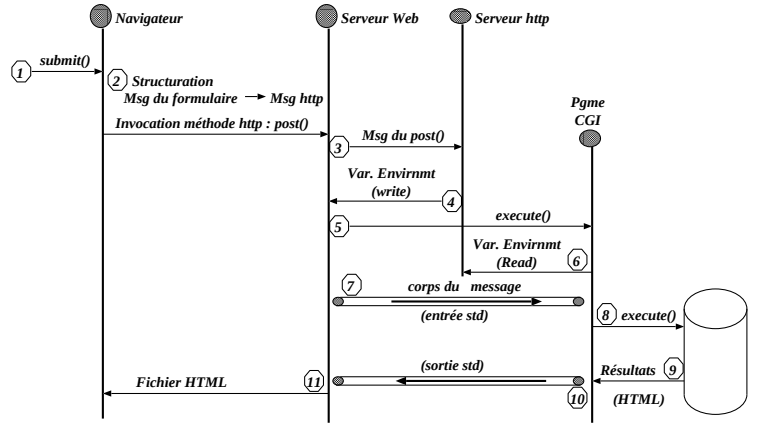


- *Services de formatage* Relationnel/Objet $\longleftrightarrow$ HTML (tables)

Serveurs Web : interaction C/S

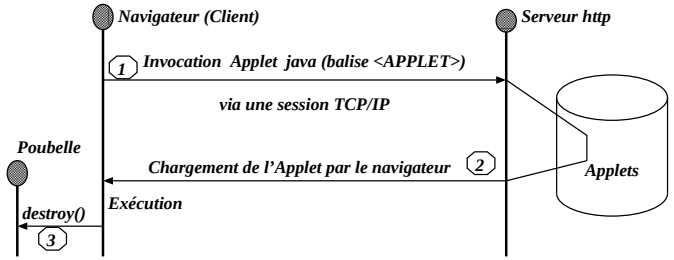


Serveurs Web : CGI



- *Variables environnement* : server_name, script_name, etc

Serveurs Web : Applet



- *CGI* : Protocole serveur Web ⟷ programme CGI
  - Serveur ⟶ Programme : méthode + paramètres
  - Serveur ⟵ Programme : résultat (HTML)
- *Applet* : Composant (code) s'exécutant sur le client

Typologie (7/7) : Serveurs Web + Objets

Patience !

Client/Serveur : Composants et Technologies

1. Middleware
2. Anatomie d'un serveur
3. Anatomie d'un client
4. Types de communication entre systèmes
5. Systèmes d'exploitation : clients, serveurs
6. Systèmes d'exploitation réseaux
7. Conclusion

1/6) Composants des architectures Client/Serveur : le "/"

- Client-*Middleware*-Serveur

- *Middleware* :
  - Depuis les API, côté client
  - Jusqu'au retour des réponses du serveur

  - *Deux Classes* :
    1. Middleware Général
    2. Middleware spécifique à un service ou une architecture

Middleware Général

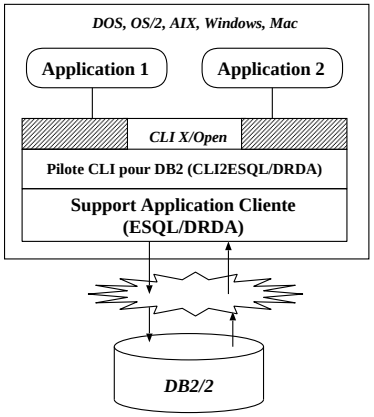
- *Comprend* :
  - Piles de communication
  - Annuaire réparti
  - Services d'authentification
  - Synchronisation d'horloges
  - Appels de procédures distantes
  - Gestion files d'attente

- *Exemples* : LAN/WAN Servers, NetWare, TCP/IP, DCE, MOM, etc

Middleware Spécifique

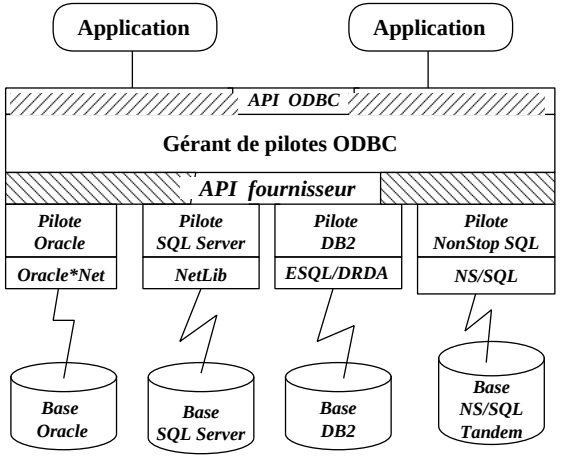
- *Bases de données* : ODBC, JDBC, DRDA, EDA/SQL
- *Transactionnel* : TxRPC et XATMI (X/Open), Transactional RPC (Encina)
- *Groupware* : SMTP, Appels à Lotus Notes
- *Objets répartis* : CORBA, Network OLE/DCOM
- *Internet* : HTTP, S(ecure)-HTTP
- *Administration système* : SNMP (Simple Network Management Protocol), ORB

Middleware bases de données : Callable Level Interface X/Open

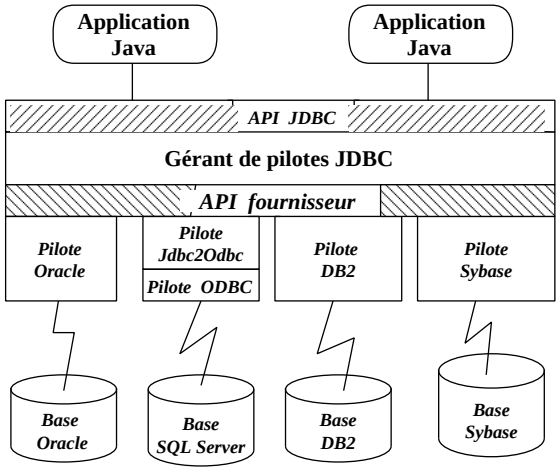


- *EDA/DRDA* : Entreprise Data Access/Distributed Relational Data Access

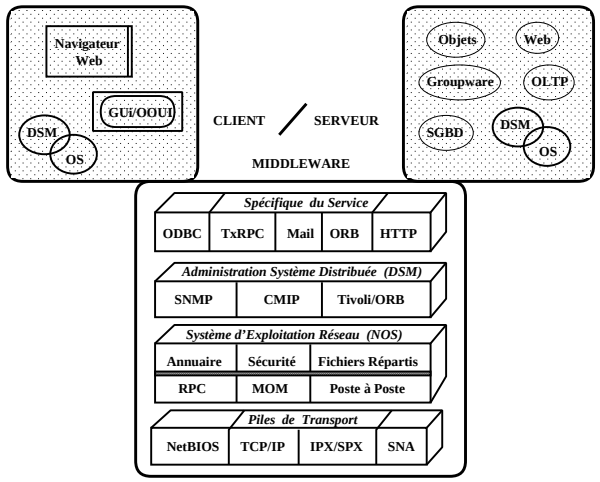
Middleware Serveurs bases de données : ODBC



Middleware bases de données : JDBC

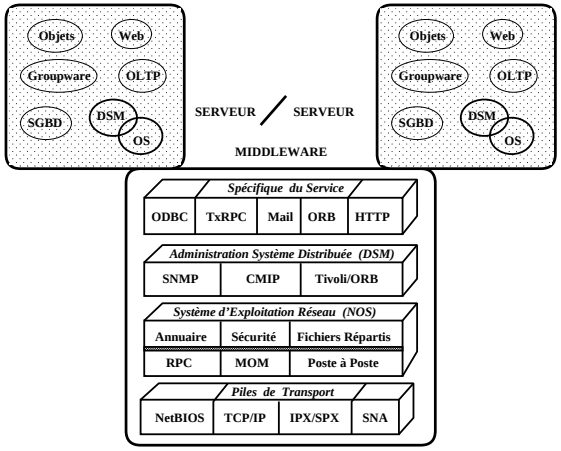


Composants des architectures Client/Serveur (1/2)



Composants des architectures Client/Serveur (2/2)

- Interaction Serveur-Serveur de type Client-Serveur



2/6) Fonctionalités d'un programme serveur

- Réception demandes clients
- Exécution simultanée des demandes
- Gestion des ressources partagées
- Priorité inter-demandes
- Sécurité de fonctionnement

3/6) Fonctionalités d'un programme client

- Offrir les services
- *Typologie* :
  1. Sans interface graphique : lecteur de codes barres
  2. Avec interface graphique (GUI) : Windows 3.1, Motif
  3. Interface Orientée Objets : Windows 9x, MacOS, OS/2 NextStep

4/6) Communications Client-Serveur

- A la charge du Network Operating System (NOS) :
  - Communication Client-Serveur
  - Synchronisation Requêtes-Réponses
  - Traiter les différences de représentation des données
- *Quatre modes de communication* :
  1. Poste à poste
  2. rpc
  3. Files d'attente (Message Oriented Middleware)
  4. Object Request Broker

Communications (1/3) : Poste à poste

- Protocole de bas niveau : sockets, Transport Layer Interface (TLI, AT&T, $\simeq$ sockets)
- Interface identique Client et Serveur
- Pas de transparence du niveau physique
- Codage et maintenance difficiles
- *Délaissé* au profit de rpc, MOM et ORB

Communications (2/3) : rpc

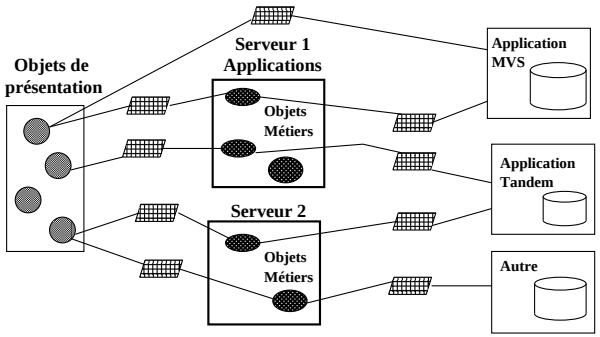
- Communication synchrone
- *Problèmes d'implantation* :
  1. Localisation et activation serveur (connue, fichier de configuration, annuaires, etc)
  2. Passage de paramètres (IDL $\longrightarrow$ Souches)
  3. Sécurité et traitement des pannes
  4. Hétérogénéité des représentations des données
     - *Sun* : Conversion en eXternal Data Representation par les clients
     - *DCE* : Plusieurs formats (Neutral Data Representation)
       - Client : Annotation des données avec nom de format
       - Serveur : Conversion dans un de ses formats

Communications (3/3) : MOM

- Client et Serveur déposent/retirent des messages
- Modèles (1-N), (N-1), (N-M)
- Files persistantes ou non, locales ou distantes
- 1993 : Consortium MOM de normalisation
  - IBM : MQSeries
  - PeerLogic : PIPES
  - Horizon Strategies : Message Express
  - etc

Communications MOM : C/S avec MQSeries3T

- MQ3T : Routage
- Windows, NT, OS/2, AIX, Grands systèmes, etc



5/6) Composants des architectures C/S : Systèmes d'exploitation (OS)

- *OS Clients* :
  - OS/2 Warp Connect (IBM)
  - Windows'xx, Windows NT Workstation
  - MacOS

- *OS Serveurs* :
  - Netware 4.1 (Novell)
  - OS/2 Warp Server (IBM)
  - NT Server
  - Unix

(Quelques) Services d'un OS pour un client

Mécanisme	Sans GUI		GUI	UI OO
	Monotâche	Multi-tâches		
Routage Requêtes, Réponses	Oui	Oui	Oui	Oui
Transparence local/distant	S	S	S	S
Multi-tâche pré-emptif	Non	Oui	S	Oui
Priorité inter-tâches	Non	Oui	S	Oui
Protection inter-tâches	Non	Oui	S	Oui

- **S** : Souhaité

Services d'un OS pour un serveur

1. *Services de base* :
   (a) Multi-tasking en pré-emption
   (b) Priorités, Protection inter-tâches, Exclusion mutuelle
   (c) Communication inter-processus (locaux et distants)
   (d) SGF muti-utilisateurs
   (e) Gestion mémoire
2. *Services étendus* pour :
   - accès aux données partagées
   - gestion du système réparti
   - "ouverture" (extensibilité)

Services étendus d'un OS pour un serveur

1. Communications : piles de protocoles

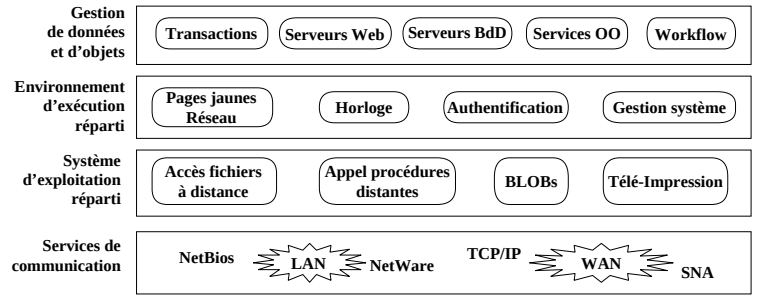
2. Réseau :
   - Accès fichiers distants
   - Appels de procédures distantes (rpc)
   - Gestion de blobs (formats, transfert efficace, etc)

3. Environnement d'exécution répartie :
   - Pages jaunes (localisation serveurs et services)
   - Horloge
   - Authentification et autorisation
   - Gestion système (configuration, performances, protection, etc)

4. Gestion de données et d'objets :
   - Transactions ([moniteurs])
   - Serveurs de bases de données, Web, d'objets, workflow, etc

Aucun système ne fournit (aujourd'hui) tous ces services

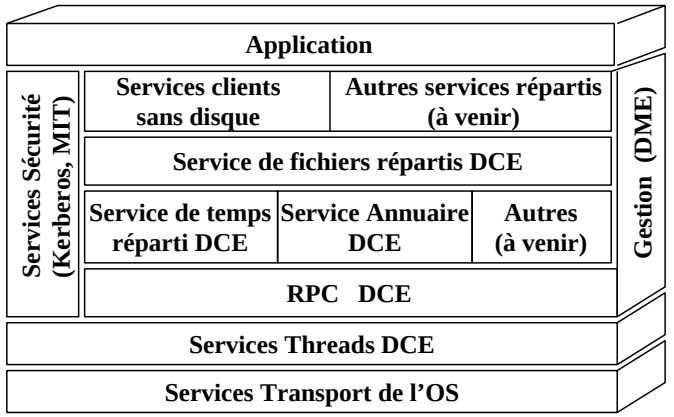
Services Étendus d'un OS



6/6) Composants des architectures C/S : Systèmes d'exploitation réseaux (NOS)

- Permettre à des applications écrites pour un OS local de fonctionner sur un réseau
- *Assurer la transparence* :
  - Localisation d'une ressource,
  - Nommage,
  - Réplication,
  - Horloge,
  - Administration système.

DCE : exemple de NOS serveur



Composants des architectures C/S : Systèmes d'exploitation réseaux (NOS)

- Distributed Computing Environment (OSF, X/Open)

+ *OS Serveurs* :

- NetWare (Novell)
- OS/2 Warp Server (IBM)
- NT Server
- *Monde Unix* : ONC (Sun), DCE

Conclusion : Tendances des NOS

- Fonctions NOS incorporées aux OS serveurs
- Au-delà des réseaux locaux (NFS, Sun RPC)
- ORB "absorberont" les NOS : interfaces objets des services des NOS

Partie II : Objets distribués et CORBA

Objets distribués et CORBA

1. Anatomie de CORBA : Architecture et composants
   (a) Modèle de données et IDL
   (b) Notions de programmation en CORBA
   (c) ORB et Object Adapter(s)
   (d) Quelques services communs
   (e) Les utilitaires communs et Corba/OpenDoc
   (f) CORBA/OpenDoc vs OLE/DCOM
   (g) Interopérabilité et fédération d'ORBs
2. Client/Serveur Corba, Objets répartis, Web

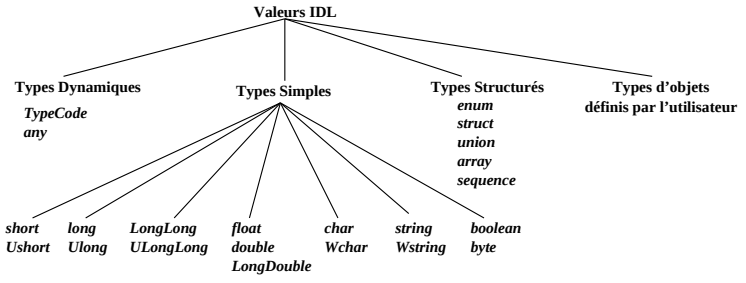
CORBA : l'IDL

- Interfaces décrites dans un "langage pivot" (IDL)
- Traduction dans "n'importe quel" langage de programmation
- Interface d'un service indépendante de la localisation de l'objet ou de son langage d'implémentation
- Types des données exprimés dans un modèle de représentation

CORBA : Modèle de données

- *Types de base*,
- *Types énuméré* et *any*,
- *Types construits* (produit cartésien, union, séquence, tableau et interface)
- *Type interface* : spécifie les opérations (ou *services*) offertes par une instance d'un type d'objet
- Interfaces héritables
- Interface principale : fermeture transitive du graphe d'héritage.
- "Compatible" ODMG

CORBA : les types de données de l'IDL



Structure d'un fichier IDL

```
module < NomModule >
{
    < Déclararions de types >;
    < Déclarations d'exceptions >;
    interface < NomInterface > [: < Héritage >]
    {
        < Déclarations de types >;
        < Déclarations de constantes >;
        < Déclarations d'attributs > ;
        < Déclarations d'exceptions > ;
        [< Type du résultat >] < NomOpération > (< Paramètres >)
        [raises NomException] [context];
        .....
        .....
        [< Type du résultat >] < NomOpération > (< Paramètres >)
        [raises NomException] [context];
        .....
    }
    interface < NomInterface > [ : < Héritage >]
    {
        .....
    }
```

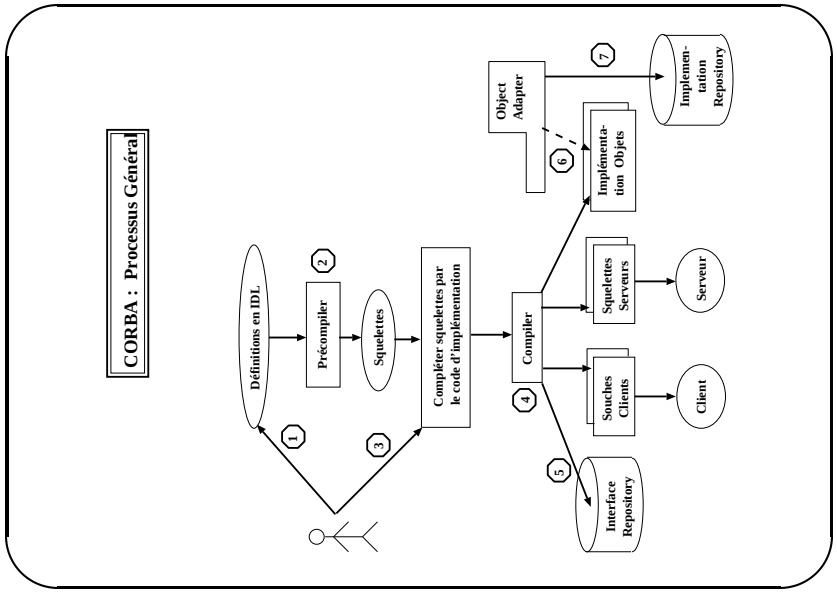
CORBA : l'IDL (exemples)

Exemple 1 : Spécification en IDL :

```
interface UnObjet{
   long UneOp(IN long param1);
                }
```

"Compilation" Idl2C :

```
typedef Object UnObjet;
extern long UnObjet_UneOp(UnObjet o,
                          Environnement *env,
                          long param1);
```

CORBA : Un exemple

Client *(AppliCl.java) sous W'95*	Serveur *(ServerMess) sous NT*
Entier → / ← Chaîne	
① *Construire le fichier interface* *Message.idl*	*Message.idl*
② *(pré-) Compiler* *idl2java Message.idl*	*idl2java Message.idl*
③ *Compléter Souche et squelette* *Applicl.java*	*ServerMess.java* (Serveur) *MessageImpl.java* (Implémentation Objet)
④ *Compiler : visibroker java compiler* *vbjc Applicl.java*	*vbjc ServerMess.java* *vbjc MessageImpl.java*
⑤ *Lancer un OSagent sur le client ou sur le serveur (catalogue d'objets distribués)*	*osagent()*
⑦ *Lancer le client* *vbj AppliCl*	⑥ *Lancer le serveur* *start vbj ServerMess* *(vbj = ''sur-couche'' de la JVM)*

Exemple : Fichier interface

Message.idl

```
module modulmess {
    interface message {
        string  get (int short val);
          . . .
              };
          };
```

Exemple : Application cliente (applicl.java)

```
public class applicl {

  public static void main (string argv[ ]) {

    try { org.omg.CORBA.ORB unOrb = org.omg.CORBA.ORB.init();
          modulmess.message gerantMsg = modulmess.messageHelper.bind (unOrb  "Msg");

          string m = gerantMsg.get((short) 4);
          system.out.println ("Resultat :" + m);

                - - - - -
        }
    catch (org.omg.CORBA.NO_IMPLEMENT e) {
          system.out.println ("Serveur inaccessible ou non implante");
        }
   }
}
```


Exemple : Le Serveur (ServerMess.java)

```
public class ServerMess {

    public static void main (string[ ] args) {

          org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init (args, null);
          org.omg.CORBA.BOA boa = orb.BOA_init ( );

    //  création d'un objet message

          modulmess.unmessage  messager = new MessageImpl("Msg");

    //  Informer l'environnement de sa disponibilité

          boa.obj_is_ready (messager);

          boa_impl_is_ready ();
    }
}
```


Exemple : L'implémentation de l'objet (MessageImpl.java)

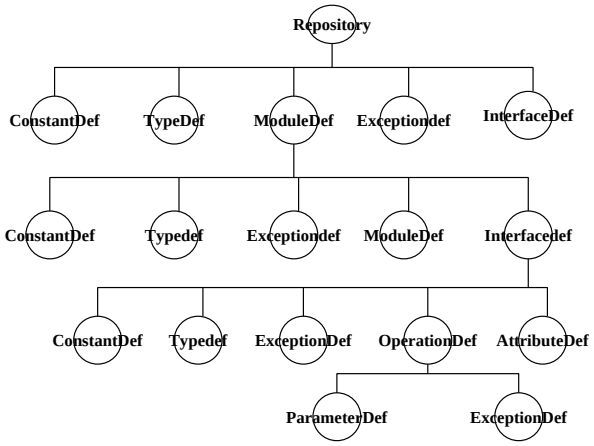
```
public class MessageImpl
        extends modulmess.messageImplBase {

    MessageImpl (string  st) {
            super(st)
    }

    public string get (short value) {
            switch (value) {
               case 1: return ("Bonjour");
               case 2: return ("Sabah el Kheir");
               case 3: return ("Good morning");
               case 4: return ("Yobosseyo");
            }
            return ("Code inconnu");
           }
} // fin de la classe Message
```

Classe abstraite engendrée par idl2java

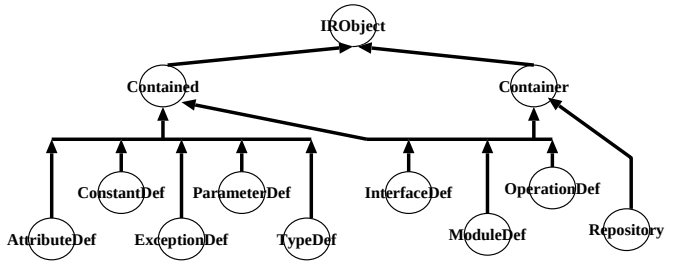
Interface Repository

- $\simeq$ Méta-base ou Dictionnaire
- Objets *auto-décrits* : *get_interface()* invocable sur tout objet CORBA, rend la description de son interface
- Clients et outils : création dynamique d'invocations de services
- Test conformité invocations-signatures
- Connexion inter-ORBs :
  - Fédération d'IRs
  - Traduction d'objets inter-ORBs hétérogènes

Interface Repository : Hiérarchie de contenance



Interface Repository : Hiérarchie d'héritage

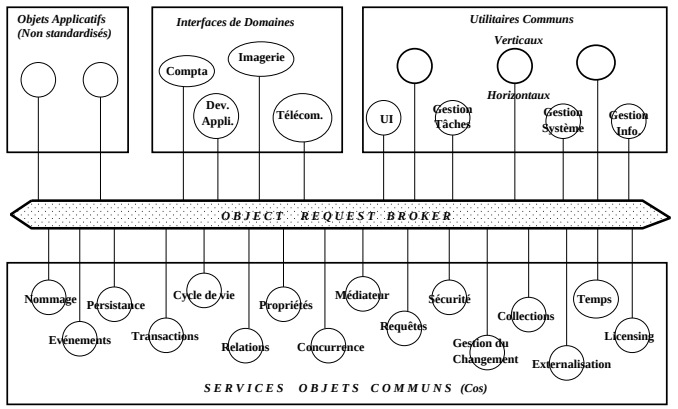


- IRObject : *destroy()*
- Contained : *describe(), move()*
- Container : *lookup(), lookup_name(), describe_contents(), create_[module(), interface(), . . . , alias()]*

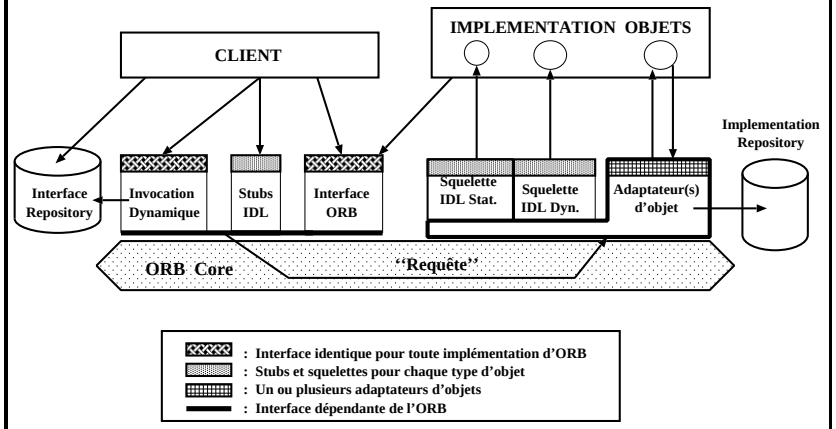
CORBA : Architecture et Composants

1. Architecture et services pour la gestion d'objets
2. Structure du bus logiciel
   - Object Request Broker et ses composants
   - Adaptateur(s) d'objets
3. Quelques services
4. Interopérabilité et Fédération d'ORBs

1) Architecture pour la gestion d'objets



2) Structure de l'ORB CORBA 2.0



CORBA : Object Request Broker

- ORB : Bus transportant requêtes et réponses
- L'implémentation d'un objet reçoit une demande de service ou "requête" à travers un squelette IDL
- Requête émise par un *client*
  1. via un *stub* i.e. procédure locale permettant d'invoquer une opération ($\simeq$ *remote procedure call* statique)
  2. ou via *l'interface d'invocation dynamique*
- Implémentation d'un objet peut faire appel aux services d'un *adaptateur d'objet (OA)* (Min. 1 Basic OA/ORB)

CORBA : Object Request Broker (ORB)

- Achemine les invocations et leurs réponses
- Assure la *transparence de la localisation* des objets :
  - Les clients ont leurs propres références d'objets
  - ORB établit la correspondance références des clients - références effectives des objets
- *Connectivité inter-ORB* $\longrightarrow$ mécanismes de gestion des références sur chaque ORB

ORB CORBA : Composants

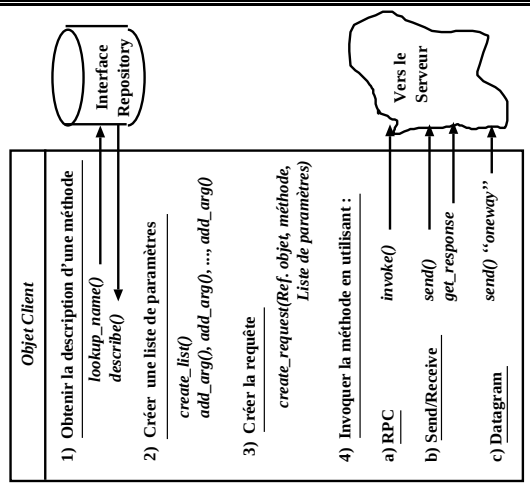
I. *Côté Clients*
1. Souches (stubs) IDL Clients
2. Interface d'Invocation Dynamique (DII)
3. APIs de l'Interface Repository
4. Interface ORB

II. *Côté Serveurs*
1. Interface ORB
2. Implementation Repository
3. Interface Squelettes Dynamique (DSI $\simeq$ DII)
4. Souches/Squelettes IDL Serveurs
5. Adaptateur d'objets

Côté client : Interface d'Invocation Dynamique

- APIs pour localiser et invoquer dynamiquement des méthodes
  - Consultation (*look_up*) de l'IR
  - Génération des paramètres
  - Emission de la requête
  - Récupération des résultats
- Serveur : pas de différence entre invocation statique ou dynamique
- Dans chacun des cas, l'ORB :
  - localise un adaptateur d'objets
  - lui transmet les paramètres
  - transfère le contrôle à l'implémentation de l'objet

Côté client : Interface d'Invocation Dynamique (suite et fin)



Côtés client et serveur (1/1) : Interface ORB

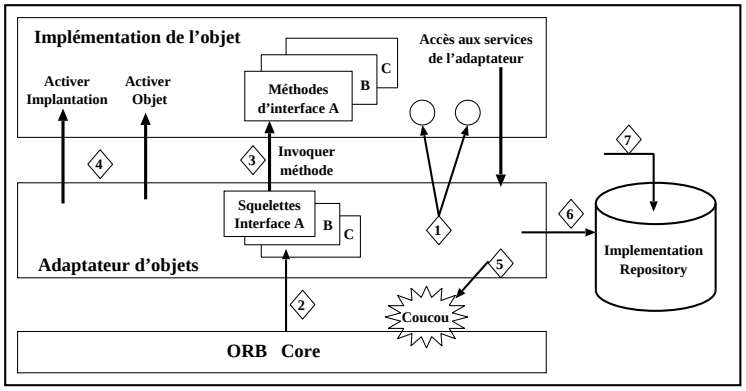
- Offre des APIs de services locaux, d'intérêt pour une application
- *Exemple* : Convertir référence d'objet $\longleftrightarrow$ Chaîne (pour stocker/communiquer des références d'objets)

Côté serveur : Adaptateur d'Objets et Implementation Repository

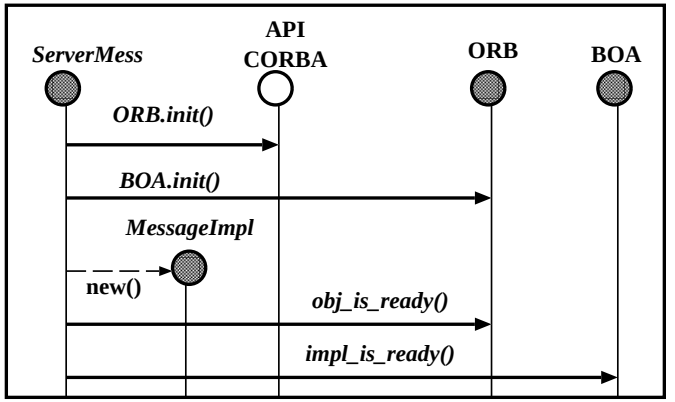
1. Engendre et interprète les références d'objets,
2. Authentifie les clients et les requêtes, lie celles-ci aux squelettes
3. Invoque, à travers les squelettes, les méthodes implémentant les services,
4. Active/désactive un objet ou une implémentation d'objet,
5. Diffuse la présence d'objets serveurs,
6. Crée des instances d'objets [Nombre = f("charge" clients)]
7. Enregistre les classes dans *l'Implementation Repository*

- *Implementation Repository* peut aussi contenir des informations de trace, d'audit, de sécurité, etc.

Adaptateur d'objets



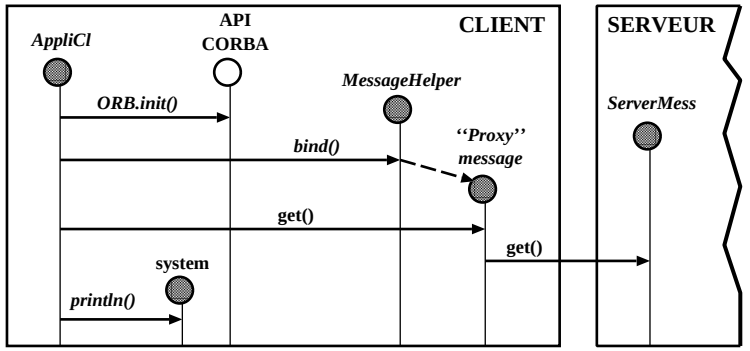
Objets Serveurs et Adaptateur d'objets



Objets Serveurs et Adaptateur d'objets : Retour sur l'exemple (1/2)

```
public class ServerMess {

    public static void main (string[ ] args) {

        org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init (args, null);
        org.omg.CORBA.BOA boa = orb.BOA_init ( );

    //  création d'un objet message

        modulmess.unmessage  messager = new MessageImpl("Msg");

    //  Informer l'environnement de sa disponibilité

        boa.obj_is_ready (messager);

        boa_impl_is_ready ();
    }

}
```


Objets Clients et Adaptateur d'objets



Objets Clients et Adaptateur d'objets : Retour sur l'exemple (2/2)

```
public class applicl {

  public static void main (string argv[ ]) {

    try { org.omg.CORBA.ORB unOrb = org.omg.CORBA.ORB.init();
          modulmess.message gerantMsg = modulmess.messageHelper.bind (unOrb  "Msg");

          string m = gerantMsg.get((short) 4);
          system.out.println ("Resultat :" + m);

                    - - - - -
        }
      catch (org.omg.CORBA.NO_IMPLEMENT e) {
            system.out.println ("Serveur inaccessible ou non implante");
        }
    }
}
```


3) CORBA : Présentation de quelques services

CORBA : Les services

1. *Services de bases* :
   (a) Nommage
   (b) Traders
   (c) Cycle de vie
   (d) Evénements
   (e) [Méta-Données]
2. *Transactions, Accès concurrents, Persistance*
3. *Interrogation, Collections et Relations* :

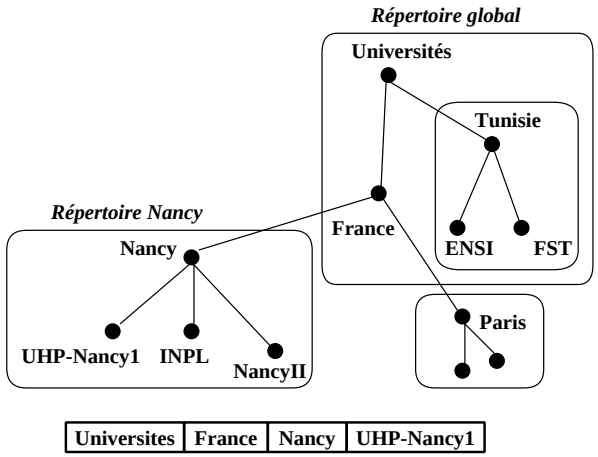
CORBA : Les services (suite et fin)

4. *Gestion système et sécurité*
   (a) Security
   (b) Licensing
   (c) Time
   (d) Change Management
   (e) Properties
   (f) Externalization

Services de base (COS) : Nommage

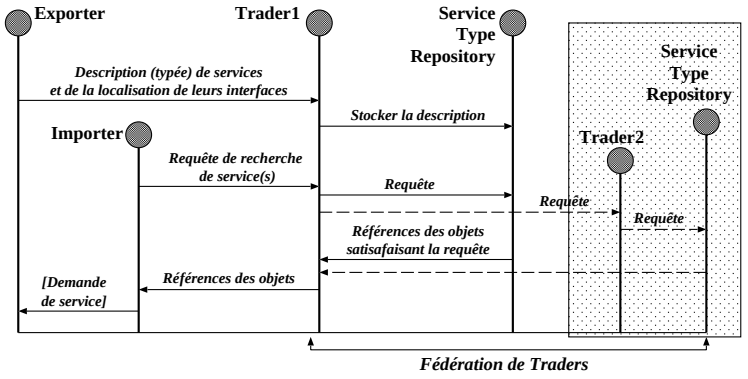
- $\simeq$ pages *blanches* (localiser un objet par son nom "externe")
- Lier un *nom* à un objet dans un *contexte de nommage* (*bind()*)
  - Gestion d'un graphe de contextes de nommage
  - *Contexte de nommage* : objet contenant un ensemble de liaisons (Nom-référence d'objet)
  - Chemin dans le graphe de nommage : *CosNaming::Nom*
- *Composant du nom* : $<$*identifier* : string, *kind* : ... $>$

Services de base (COS) : Nommage (suite et fin)



Services de base (COS) : Négociateurs (Traders)

- $\simeq$ pages *jaunes* : publication/recherche de services

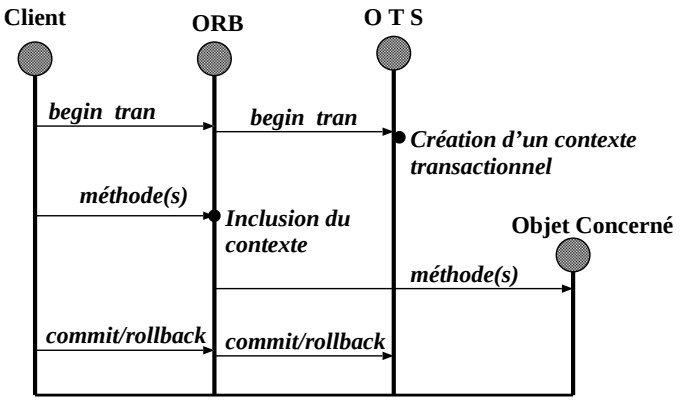


2) Transactions, Concurrence et Persistance

2.1) Services Communs : Object Transaction Services

- "Plates" (obligatoire) et imbriquées
- Sur ORB hétérogènes
- Applications ORB et non ORB peuvent contribuer à la même transaction
- Rendre un objet *"transactionnel"* : utiliser une interface qui hérite d'une classe abstraite OTS
- *Transactions imbriquées* : relaxation du I (*ACID*) : visibilité des résultats intermédiaires d'une transaction parente
- *Composants* :
  1. Client transactionnel
  2. Serveur transactionnel (coordinateur)
  3. Serveur sujet à reprise (*Recoverable Server*)

Composants OTS (1/3) : Client Transactionnel



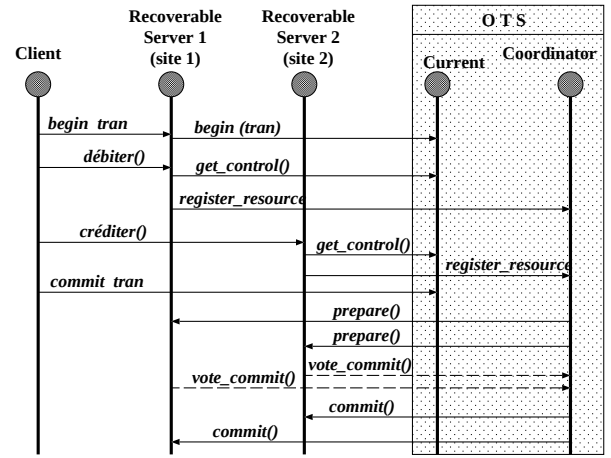
Composants OTS (2/3) : Serveur Transactionnel

- {Objets} sans ressources propres, non sujets à reprise
- Propage les méthodes transactionnelles
- Peut forcer une annulation (rollback)

Composants OTS (3/3) : Recoverable Server

- {Objets} sujets à reprise i.e à commit/rollback (ex : fichiers, BdD)
- *Recoverable Objects* :
  - notifient l'OTS qu'une ressource est concernée par une transaction : *register_resource()*
  - fournissent les méthodes pour rpc via l'ORB

OTS (fin) : Scenario d'un 2 phase commit



2.2) Services Communs : Concurrence

- Contrôle d'accès à des ressources partagées
- Par verrouillage
- Modes : *read, write, intention, upgrade*

2.3) Services Communs : Persistence Object Services

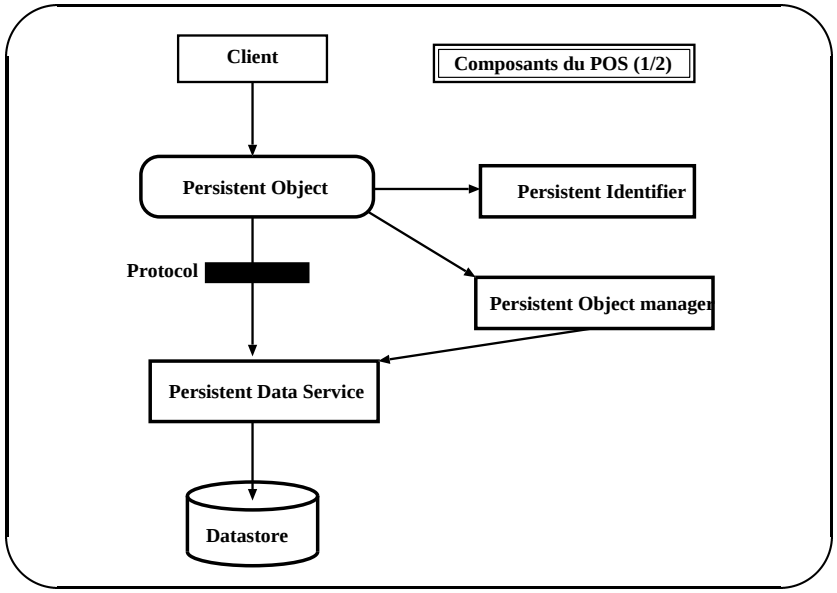
- Interfaces communes vers les gérants de données/objets persistants
- Pour systèmes à un ou deux niveaux (SGBDOO vs SGBDR ou SGF)
- *Composants du POS* :
  1. Persistent Objects (PO) :
     - Par héritage de la classe PO et d'un mécanisme d'externalisation
     - Identifiant unique (*Persistent IDentifier*)
  2. Datastores (DSs) : SGFs, SGBDs
  3. Persistent Object Manager (POM) :
     - Interface indépendante de toute implémentation
     - $\simeq$ routeur d'objets pour les DSs

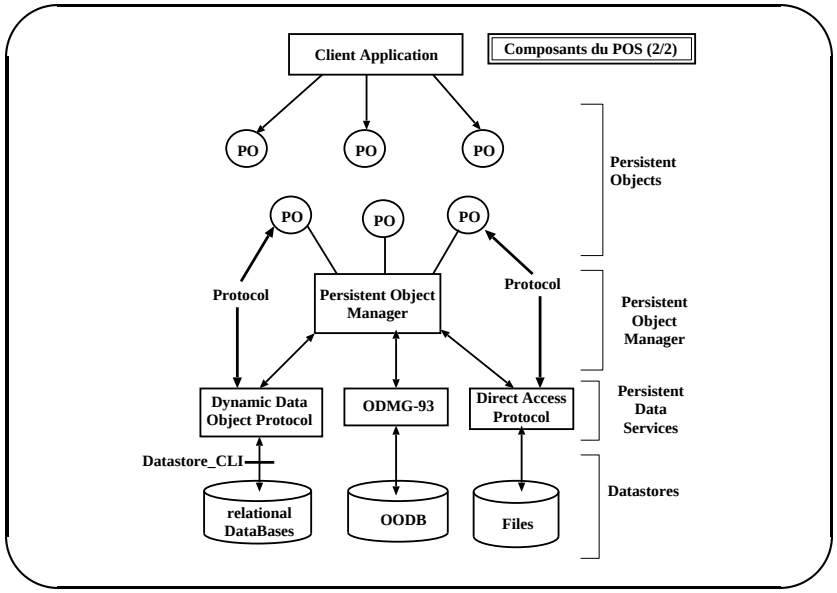
4. Persistent Data Services (PDSs) :
   - Interface pour un DS particulier
   - Pour le transfert de données entre un objet et un DS
   - Peuvent supporter des prootocoles particuliers pour accès/stockage de données
   - *Trois protocoles définis dans le POS* :

   (a) *Direct Access protocol* : accès grâce à un DDL "à la" IDL (sous-ensemble ODMG-93)

   (b) *ODMG-93 protocol* : Accès à partir de C++, Smalltalk, Java

   (c) *Dynamic Data Object protocol* :
   - Représentation *neutre* des données
   - IDL non nécessaire

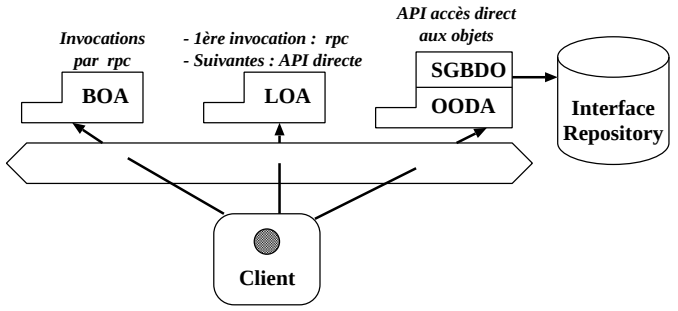




Persistance : CORBA et ODMG

- Environnement OMG : SGBDO pour concurrence d'accès à des bases de grande taille
- ODMG-93 : extension du POS (protocole d'accès pour un Persistent Store avec objets à grain fin)
- *Library Object Adapter* : API d'accès direct via l'ORB
- *Object Oriented Database Adapter* : Normalisation de LOA (à la demande des vendeurs de SGBDO)

Persistance : CORBA et ODMG (suite et fin)

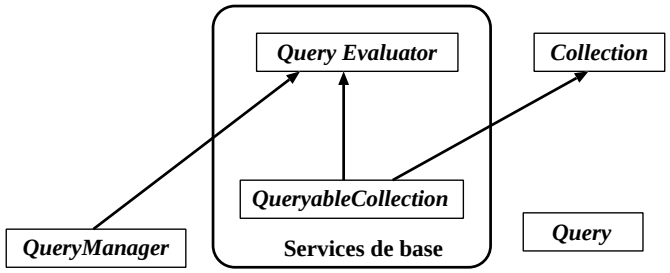


3) Services Communs : Requêtes, Collection, Relations

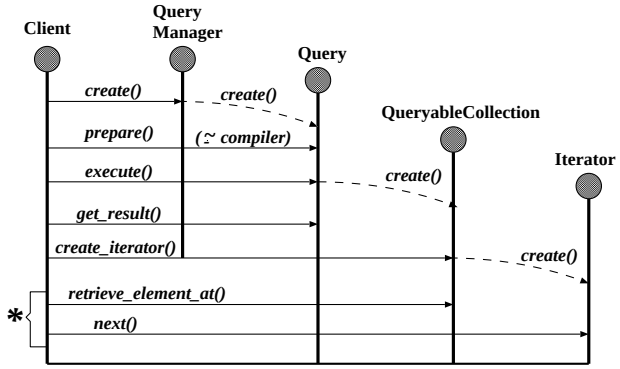
Services Communs : Requêtes (Object Query Services) et Collections

- en OQL (ODMG), SQL + extensions objets ou un sous-ensemble des deux
- *Requête* i.e. pas d'accès à l'état des objets
- *Réponse* : *Collection* d'objets

Services Communs : Interfaces OQS



Services Communs : Scenario OQS



Services Communs : Relations

- *CosRelationship* : Définition de types de relations ($\simeq$ Associations : rôles, attributs, cardinalités)
- Instance : Graphes d'objets (*CosGraph*)
- *Navigation* dans le graphe (exemple : PCIS2 Traceability)

CORBA : Utilitaires Communs et Documents Composites

1. *Utilitaires Communs*
   (a) Gestion système
   (b) Gestion des tâches
   (c) Interface utilisateur
   (d) Gestion d'informations
2. *CORBA et Documents Composites* : OpenDoc

Utilitaires Communs (1/4) : Gestion système

- *Interfaces et services pour* :
  - gérer
  - configurer
  - installer
  - réparer

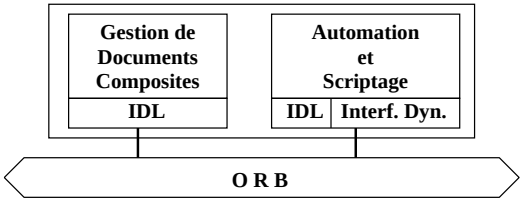
  des composants objets distribués

Utilitaires communs (2/4) : Gestion de tâches

- *Cadre pour la gestion de* :
  - workflow
  - agents
  - scriptage
  - automation
  - règles (ECA)
  - longues transactions

Utilitaires communs (3/4) : Interface Utilisateur

- *Document composite* : métaphore pour organiser des composants
  - visuellement
  - à travers des relations "contenant-contenu"
- *Utilitaires UI* : Technologie pour documents composites (partage d'écran d'affichage, acheminement d'événements aux composants t.q. barres de menus, d'outils)

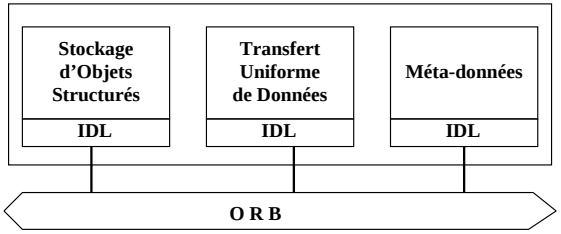


Utilitaires communs (3/4) : Interface Utilisateur

- *"Scriptage"* : $\simeq$ programme
  - Script appelé par un document sur événement
  - *Exemples* : journalisation des lectures/écritures, notification par mel, authentification des accès, etc
- *"Automation"* :
  - Ecriture et attachement dynamique de scripts
  - Coordination/Collaboration de suites de composants

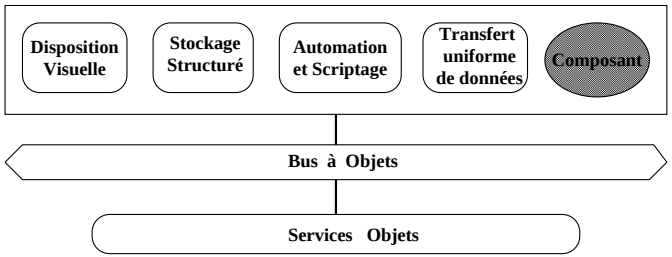
Utilitaires communs (4/4) : Gestion d'informations

- Stockage de documents structurés
- Echange de données
- Définition de standards de définition, représentation, échange de données et méta-données
- Copie/Déplacement des extensions profondes d'objets

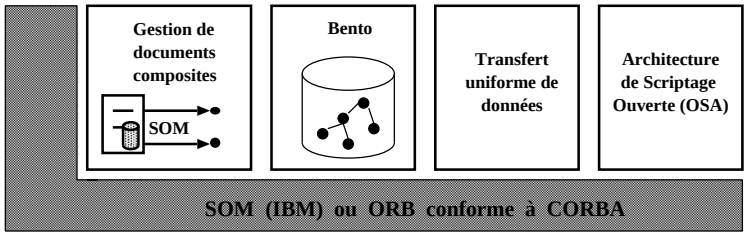


CORBA/OpenDoc pour Documents Composites

Canevas pour Documents composites



Canevas pour documents composites CORBA/OpenDoc



- *OpenDoc* : Component Integration Labs (IBM, Novell, Apple, Oracle, SunSoft, Taligent, Xerox, . . .)

CORBA/OpenDoc

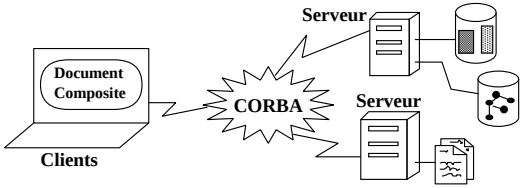
- OpenDoc : $\simeq$ CORBA sur le bureau : parties du bureau
  - Collaborent avec d'autres parties,
  - Accèdent à des serveurs d'objets via un ORB CORBA (SOM, IBM)
- *Règles pour les parties pour* :
  - partage de la zone d'affichage
  - stockage de leurs données dans un *conteneur*
  - coordination par des scripts, des événements sémantiques, des appels de méthodes CORBA
  - inter-opération avec d'autres modèles de composants (OLE, applets Java) : cf *ComponentGlue*

OpenDoc : ComponentGlue (Novell)

- Passerelle OpenDOc $\longleftrightarrow$ OLE
  - Partie OpenDoc dans/comme objet OLE
  - Objet OLE avec OCXs dans/comme partie OpenDoc
- Correspondance directe API OpenDoc-API OLE

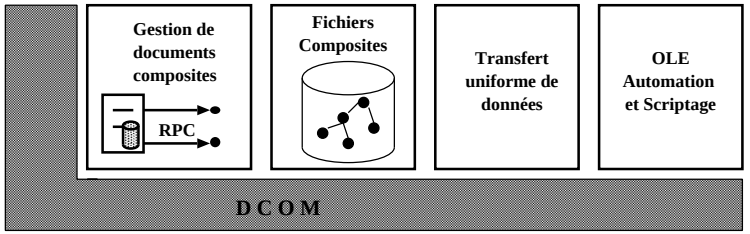
CORBA/OpenDoc : Conclusion

- *OpenDoc* :
  - bus CORBA "sur le bureau" pour la communication inter-applications
  - Applications monolithiques "encapsulables" pour coopération locale et distante
- *Document OpenDoc* : base de l'intégration Client/Serveur



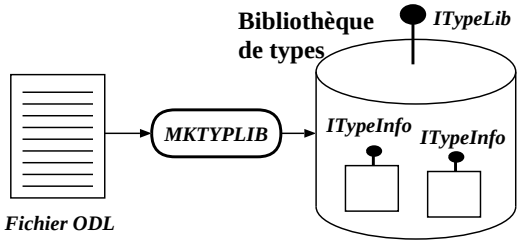
OLE/DCOM : OLE/DCOM

- Concurrent CORBA/OpenDoc
- = {services communs} pour composants
- *Canevas pour documents composites OLE/DCOM*



OLE/DCOM : Bibliothèque de types et localisation des objets

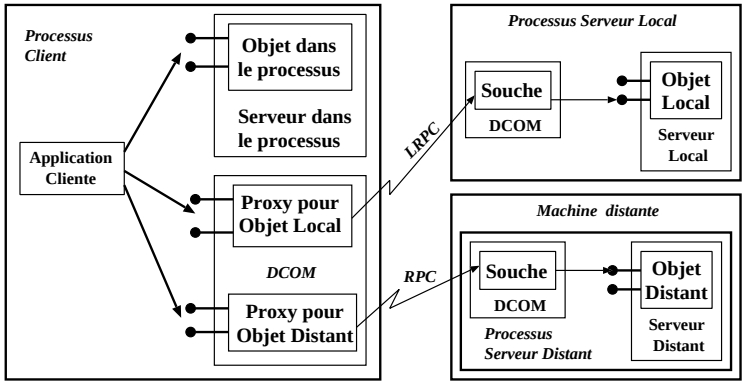
- *IDL-DCOM* : Description d'interfaces par des développeurs
- *ODL-DCOM* : Description des interfaces pour une *bibliothèque de types* ($\simeq$ IR CORBA)



OLE/DCOM : Bibliothèque de types et localisation des objets

- *Service Control Manager (SCM)*
  - DCOM $\longrightarrow$ SCM : ClassID (demande de services d'un composant)
  - SCM : Localisation d'un serveur S
  - SCM $\longrightarrow$ serveur S : Création d'une instance du composant
- *DCOM + SCM* $\simeq$ ORB + BOA
  - Localisation d'objets distants ou situés dans un autre processus + lancement
  - Mécanismes de souches et de proxy

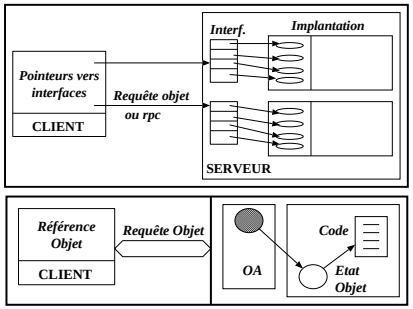
OLE/DCOM : Architecture Client/Serveur



- *LRPC* : Lightweight RPC

CORBA vs DCOM

- Objet-Interfaces : (1-1) vs (1-n)
- Référence objet vs Pointeur



- IDL standard vs IDL propriétaire

OLE/DCOM : Conclusion

- *Proposition propriétaire !*
- David Linthium, Open Computing, Jan. 1995 :

  "OpenDoc is the "rest of the world's" response to Microsft's OLE. It even encapsulate its archenemy OLE."

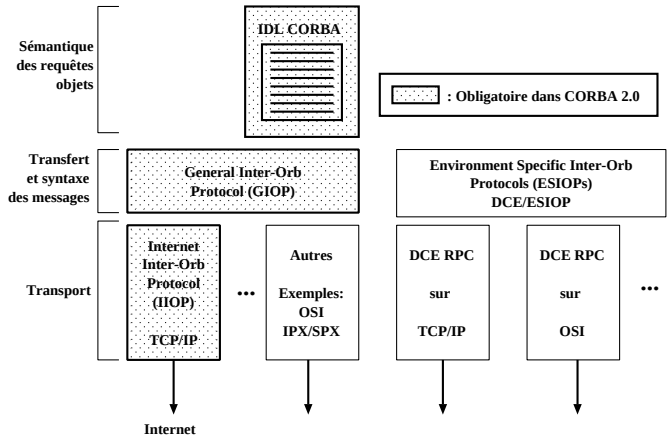
- Cliff Reeves, Director of Objects IBM, Jan. 1995 :

  "Comparing OpenDoc with OLE/COM is like comparing a modern human with Neanderthal. If you were to dress Neanderthal in a suit and stand back, the two might look the same. But you would have to look much closer if one was going to help run your business."

5) Interopérabilité et Fédération d'ORBs

1. Architecture
2. *GIOP* : General Inter-Orb Protocol
3. *IIOP* : Internet Inter-Orb Protocol

Interopérabilité entre ORBs : Architecture



Interopérabilité entre ORBs : GIOP

- *General Inter-Orb Protocol* spécifie :
  - un ensemble de formats de messages
  - des représentations communes de données
- *Sept formats de messages* couvrant la sémantique des requêtes/réponses de l'ORB
- *Common Data Representation* définit une correspondance types de l'IDL $\longrightarrow$ représentation plate

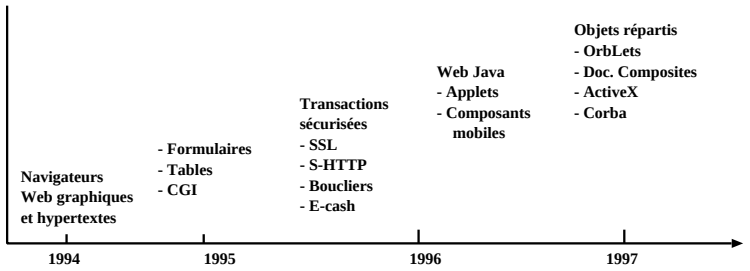
  $\Longrightarrow$ Pas de négociation de formats

Interopérabilité entre ORBs: IIOP

- *IIOP (Internet Inter-Orb Protocol)* : TCP/IP + échanges de messages définis par CORBA
- ORB conforme à CORBA doit :
  - implanter IIOP de façon native
  - ou offrir un 1/2 passerelle : ORB propriétaire peut communiquer avec l'univers CORBA en traduisant des requêtes de et vers le "dorsal" IIOP.

Client/Serveur, Web, CORBA

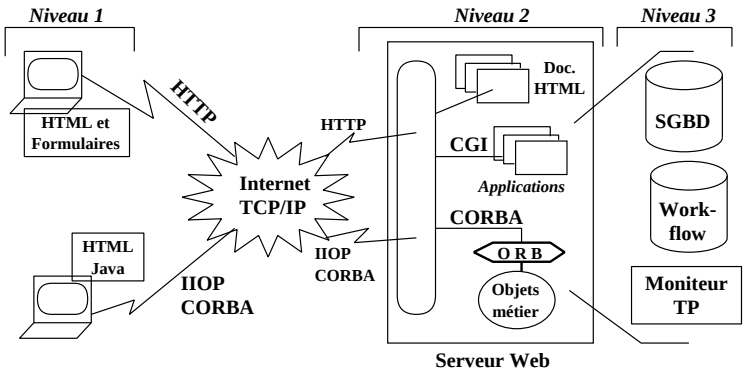
WebObject : Evolution



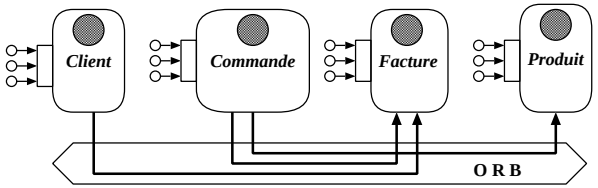
WebObject : un modèle C/S Corba, Java, Web

- *Niveau 3* : Serveurs traditionnels
- *Niveau 2* : Objets de métier CORBA
  - Logique de l'application
  - Encapsulation des serveurs traditionnels
  - Remplacement des applications CGI
- *Niveau 1* : Clients Java
  - Communication avec un objet CORBA via ORB Java
  - CORBA : middleware de communication objet à objet (à la place de HTTP/CGI)

WebObject : un modèle C/S Corba, Java, Web

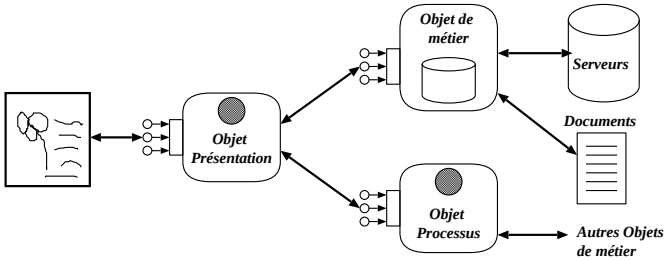


CORBA : Les objets de métiers



- *OMG Business Object Management Special Interest Group*
- *BO* : Objet de niveau application autonome offrant
  - Interfaces
  - Protocoles de coopération avec d'autres BO (cf diagrammes d'interactions)
- Application $\simeq$ suite de BOs, fournit l'environnement d'exécution

CORBA : Anatomie des objets de métiers

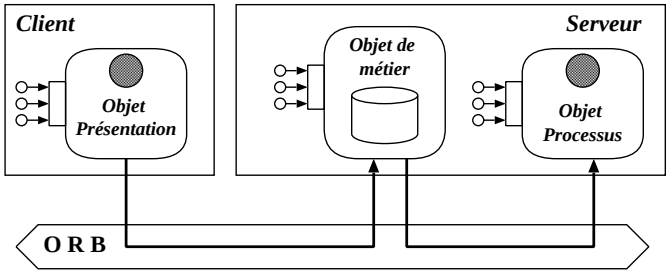


- *Présentation* : Apparence(s) externe(s) de l'objet

CORBA : Anatomie des objets de métiers

- *Métier* encapsule, pour une entité :
  - données,
  - méta-données,
  - réactions événements,
  - concurrence,
  - business rules
- *Processus* : encapsule
  - logique d'affaires (business logic)
  - processus de courte durée gérés par le BO
  - processus de longue durée, comme workflow, longues transactions (par spécialisation du BO)

CORBA : Les objets de métiers en Client/Serveur

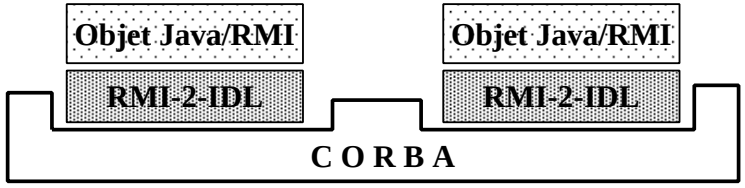


Corba et Java/Remote Method Invocation

- "Réponse" OMG-Sun à OLE/DCOM

1/3) CORBA 3 : intégré dans une version future du JDK (1.4 ?)

- JDK 1.2 : cohabitation RMI et ORB CORBA 2 minimal



Corbabeans et Javabeans

2/3) Composants ré-utilisables :

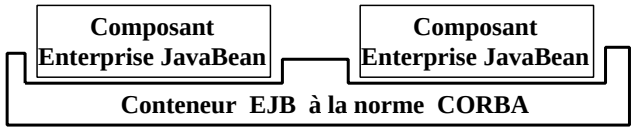
- *Corbabeans* selon le modèle *Javabeans*
- avec indépendance vis-à-vis de leur langage d'implémentation

Enterprise Javabeans

3/3) *Enterprise Javabeans (EJB)* :

- Standardisation des échanges composants Java-Conteneurs
- Conteneur : N'importe quel midleware, dont ORB CORBA

- Possibilité :
  1. *Assemblage* d'applications "trois tiers" avec des EJB
  2. *Déploiement* sur un ORB exploitant les EJB



CORBA : des implémentations

- Orbix : IONA
- SOM : IBM
- ObjectBroker : Digital
- ORB+ : HP
- VisiBroker : Visigenic
- etc

CorbaScript et CorbaWeb (LIFL)

- *CorbaScript* : langage de script interprété dédié à CORBA
  - Simplicité apprentissage et utilisation
  - Typage dynamique via IR
  - Accès direct aux objets (DII)
- *CorbaWeb* : passerelle HTTP-CORBA
  - scripts passerelle en CorbaScript
  - documents dynamiques HTML peuvent contenir du CorbaScript
- *http://corbaweb.lifl.fr*

Corba/Web : PCIS2 (CELAR (F), US Navy/SPAWAR)

- CELAR, DCN (Toulon), LORIA-Nancy, Sema Group
- SPAWAR/US Navy, Arizona State U.
- Environnement distribué de développement, sensible aux procédés
- *http://pcis2.nosc.mil/*

CORBA : Conclusion

- Propositions de l'OMG : une architecture et une spécification des fonctionalités de ses composants
- Faiblesses :
  - IDL : pas de contraintes sur les objets, les paramètres (*Tradex, MCSEAI'98*)
  - Si *invocation dynamique d'un service* : description "syntaxique" de l'interface insuffisante pour identifier le ou les objets capables de satisfaire la demande
- Compétences
- Environnements et outils