

Le modèle Objet de OMG-CORBA

UV B : Système et Applications Réparties

Année 95/96

Laurence Duchien

CNAM-CEDRIC, 292, rue st Martin

75141 Paris Cedex 03

tel : 40 27 25 83

e_mail : duchien@cnam.fr

<http://cedric.cnam.fr/personne/duchien/poly.html>

Plan

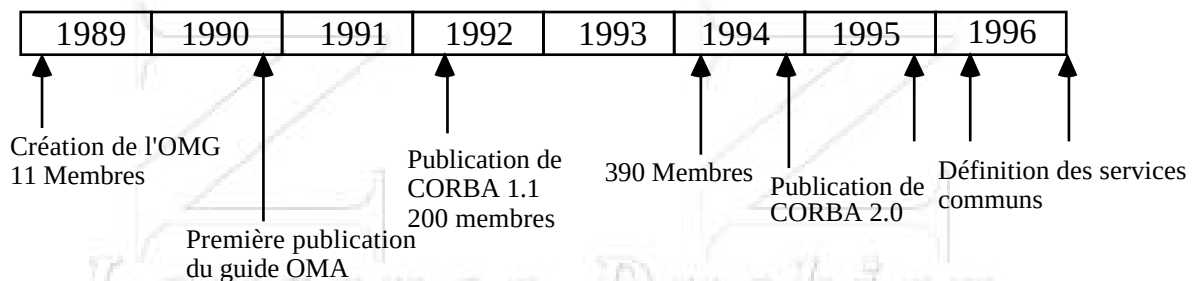
L'OMG- Object Management Group.....	4
Les objets distribués de CORBA.....	5
Tout est dans l'IDL.....	6
L'objectif de CORBA	7
L'OMA : Object Manangement Architecture.....	8
L'ORB (Object Request Broker).....	9
ORB Versus RPC.....	11
Les services objets.....	12
Les facilités communes.....	14
Le Modèle Objet pour CORBA	15
La sémantique de l'objet	16
L'implantation de l'objet.....	18
L'architecture de CORBA	19
Structure de l'interface de l'ORB	20
Le client.....	21
L'implantation objet côté client.....	22
L'implantation objet	23
La référence d'objet.....	23
Le talon IDL côté client.....	24
L'interface d'invocation dynamique (DII)	25
L'API du référentiel d'interface.....	25
L'interface ORB.....	26
Le bus ORB (Object Request Brocker core)(1)	27
Le bus ORB (Object Request Brocker core)(2)	28
Les talons IDL du serveur (skeleton).....	29
L'interface dynamique (squelette dynamique) (DSI)	30
L'adaptateur d'objet	31
Différents adaptateurs d'objets.....	33
Référentiels des interfaces et des implantations	37
L'interface d'ORB coté serveur.....	38
Le serveur.....	39
L'implantation côté serveur.....	40
L'invocation de méthodes	41
Quelques exemples d'ORBs.....	44
L'interopérabilité.....	45
Les ponts entre ORBs.....	47
L'IDL (langage de définition d'interface).....	48
La structure de l'IDL CORBA.....	49
Un exemple d'IDL.....	50
Le langage de définition de l'interface(IDL) de CORBA.....	51
L'IDL et l'héritage.....	52
Exemples IDL et DII (Dynamic Invocation Interface).....	53
Le référentiel d'interface Corba 2.0.....	54
Les classes du référentiel d'interface	55
Les services CORBA : le nommage.....	56
L'interface du service de nommage.....	58
Le service de courtier (trader).....	59
Les services CORBA : le cycle de vie.....	60
Les interfaces du cycle de vie.....	61
Les services CORBA : les événements.....	62
Les services CORBA : les transactions	63
Les éléments du service transactionnel.....	64
Les services CORBA : la concurrence	65
Les services CORBA : la persistance.....	67
Sauvegarde à un niveau contre la sauvegarde à deux niveaux.....	68

POS : La vue du client.....	69
POS : La vue de l'objet persistant.....	69
Les éléments du POS.....	70
Des implantations CORBA.....	72
Bibliographie.....	73

MINAMIN
A
Z
Laurence Duchien
O

L'OMG- Object Management Group

- Un consortium composé de vendeurs de systèmes, de développeurs de logiciels, d'utilisateurs développant et promouvant la théorie et la pratique des objets distribués.
- 400 membres (Sunsoft, HP, DEC, IBM, Iona, Hyperdesk, Object Design,...)
- Produit des standards / documents qui définissent les mécanismes de distribution et de communication dans un réseau.
 - OMA (Object Management Architecture)
 - CORBA (Common Object Request Broker Architecture)



- ftp et http : omg.org

Les objets distribués de CORBA

- Les objets CORBA sont des éléments que l'on trouve n'importe où dans le réseau.
- Ils correspondent à des composants qui peuvent être accédés par n'importe quel client distant via des invocations de méthodes.
- Le client n'a pas besoin de connaître :
 - la localisation des objets distribués,
 - le système qui les exécute,
 - la façon dont le serveur implémente les objets

Exemple :

un serveur peut être implémenté sous forme de classes C++ ou avec des millions de lignes COBOL, le client ne voit pas la différence.

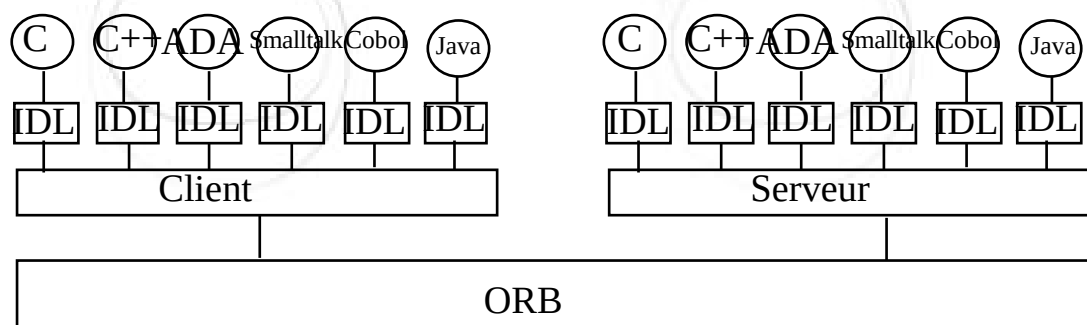
Il doit seulement connaître l'interface publié par le serveur

L'interface : le contrat qui va lier client et serveur

Tout est dans l'IDL

IDL : Interface Definition Language

- L'IDL est un langage de notation standard. Il fournit une notation universel pour spécifier des APIs.
 - Il est purement déclaratif
 - **Son objectif** : définir **les limites des composants** et **les interfaces contractuelles** avec les clients potentiels
 - L'IDL fournit des interfaces **indépendants des systèmes et des langages de programmation** pour tous les services et les composants résidants sur le Bus CORBA.
- => Permet à des clients et des serveurs écrits dans des langages différents de coopérer à travers le réseau et les systèmes d'exploitation.
- La grammaire IDL est un sous-ensemble du C++ avec des mots-clés supplémentaires pour supporter la distribution



L'objectif de CORBA

- Toute objet est équipé d'un clou... Tout individu est équipé d'un marteau

=> le clou est l'IDL CORBA.

Il permet aux fournisseurs de composants de spécifier dans le langage standard de définition l'interface et la structure des objets qu'ils fournissent.

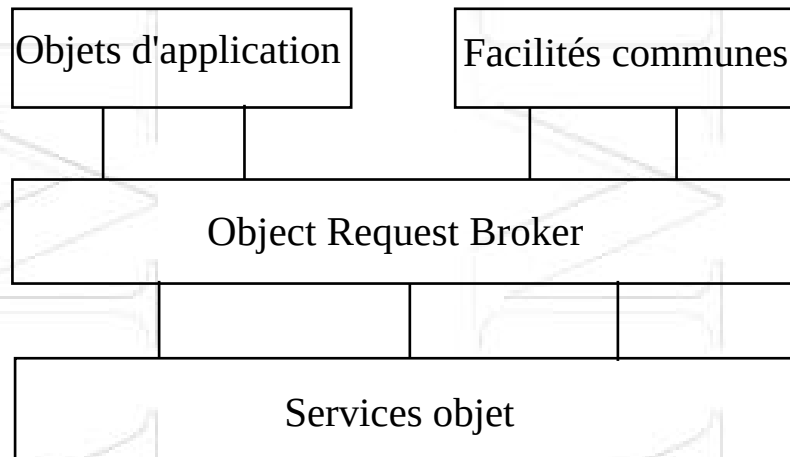
=> Le marteau correspond à l'ensemble des services distribués de CORBA.

Ils déterminent quels sont les objets dans le réseau, quelles méthodes ils fournissent et quels adaptateurs d'interface ils supportent.

=> **Le référentiel des interfaces** (Interface Repository)
CORBA contient toutes les définitions de ces interfaces et des **données méta** qui permettent aux composants de se découvrir mutuellement au moment de l'appel.

=> **Ce qui fait de CORBA un système se décrivant lui-même.**

L'OMA : Object Management Architecture



- **Services Objet**
Les fonctions de base pour mettre en oeuvre les objets:

- cycle de vie
- nommage
- Gestion des événements
- Persistance
- Contrôle de concurrence

Normes appelées COSS(Common Object Services Specifications)

- **Object request Broker (ORB)**
Bus logiciel qui assure la communication entre les objets du réseau

- **Les facilités communes**

Collection de services orientés vers l'utilisateur:

- Messagerie électronique,
- Mise en oeuvre de documents
- Accès à des bases de données

- **Objets d'application**

Les objets de l'application de l'utilisateur

L'ORB (Object Request Broker)

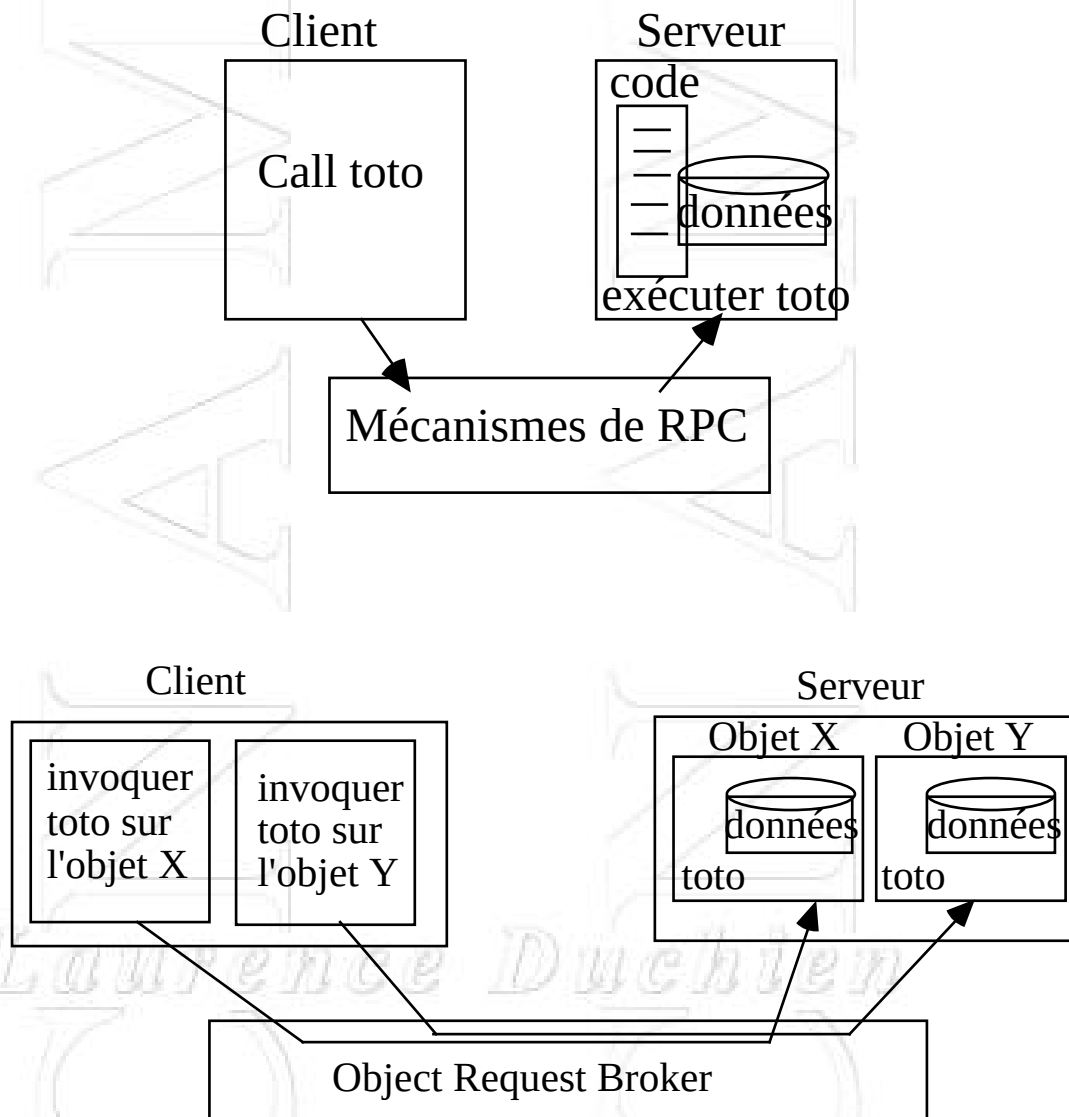
- Bus objet
- Permet aux objets de faire des requêtes vers des objets locaux ou distants et de recevoir les réponses.

• Quelques avantages de l'ORB :

- **Les invocations statiques et dynamiques** de méthodes
- **Les liaisons avec des langages de haut niveau.** Le client utilise le langage de son choix sans se préoccuper du langage utilisé par le serveur.
- **Un système se décrivant lui-même.** CORBA fournit des données de niveau méta permettant de décrire chaque interface de serveur connu du système. Ces données sont générées automatiquement à partir des pré-compilateur d'IDL. (exemple metaware C++)
- **La transparence locale ou distante :** un ORB peut fonctionner seul ou en interconnexion avec d'autres ORB (2.0).
- **Les transactions et la sécurité intégrées :** L'ORB inclut des informations de contexte dans ses messages pour gérer la sécurité et les transactions entre machines.
- **Les messages polymorphes :** un ORB invoque une opération distante sur un objet cible. Ce qui veut dire que la même opération peut avoir des comportements différents selon l'objet récepteur.

- **Quelques nuances sur ces avantages :**
 - **Les premiers ORBs commerciaux sont lents et inefficaces...** ils n'implémentent pas toutes les fonctionnalités (ramasse-miettes, répartition de charge, gestion du contrôle, tolérance aux fautes..)
 - **Le code des serveurs n'est pas très portable :** La spécification de Corba ne décrit pas suffisamment toutes les interfaces pour écrire un code de serveur portable. Par contre pour les clients, elle est complète.
 - **l'IDL a besoin d'extensions de niveau sémantique :** une forme de description flexible des messages est nécessaire

ORB Versus RPC



Les services objets

- Ensemble de services de niveau système formaté comme des objets avec une interface spécifiée en IDL.
- augmente et complète les fonctionnalités de l'ORB
- 11 objets de services :
 - **service de cycle de vie** : Opérations pour créer, copier, déplacer et effacer des objets sur le bus
 - **service de persistance** : Fournit une interface simple pour stocker des composants de manière persistante sur un ensemble de serveurs de stockage (bases de données objet, bases de données relationnelles, simples fichiers).
 - **service de nommage** : Permet aux composants sur le bus de s'enregistrer et de se retirer dynamiquement dans des événements spécifiques.
 - **service événement**: Le service définit un objet bien connu "**event channel**" qui collecte et distribue les événements à travers le réseau.
 - **service de contrôle de concurrence** : Fournit un gestionnaire de verrous qui peut obtenir des verrous pour des transactions ou des processus classiques.
 - **service de transaction** : Fournit le protocole de validation à deux phases pour des transactions plates ou imbriquées

-
- **service de relation** : permet de créer des associations dynamiques (ou liens) entre des objets qui ne connaissent rien les uns des autres.
 - **service d'externalisation** : fournit un moyen standard pour prendre ou mettre des données d'un objet en utilisant un mécanisme de flot.
 - **service de requête** : fournit des opérations de requêtes pour objets. C'est un sur-ensemble de SQL avec une partie SQL3 et une partie OQL (Object Query Language).
 - **service de licence** : fournit des opérations pour mesurer l'utilisation des composants afin d'assurer une utilisation équitable.
 - **service de propriétés** : fournit des opérations pour associer des valeurs ou des propriétés à un objet (titre, date,...).
 - **service de sécurité**... en cours de définition
 - **service de temps** en cours de définition
 - **service d'administration**...
 - **service de courtier** ...

Ces services sont décrits (ou en cours de descriptions dans des RFP (Request For Proposal) ou encore COSS (Common Object Service Specification)

L'utilisation des services permet de construire des objets en mixant ses propres objets et les services offerts par CORBA en fonction des besoins.

Exemple :

Création d'un objet "voiture" avec une version persistante, transactionnelle et concurrente en multihéritant des services correspondants.

Les facilités communes

- Collection d'objets définis avec une interface IDL pouvant être utilisés directement par les applications objets
- Deux catégories :
 - **Horizontale :**
 - **Interface utilisateur pour des services d'éditions** similaire à ceux trouvés dans OpenDoc ou OLE
 - **services de gestion d'information :** stockage de document, échange de données
 - **services de gestion système :** pour gérer, instrumenter, configurer, installer des objets
 - **services de gestion de tâches :** incluant des transactions longues, des agents, des scripts et la messagerie.
 - **Verticale :**

Fournira des services pour des segments de marché comme la santé, la finance, les télécommunications...

Le Modèle Objet pour CORBA

- Le **modèle objet de l'OMG** est un modèle abstrait.
- Le modèle utilisé pour CORBA est un modèle objet concret qui diffère du modèle abstrait sur plusieurs points :
 - **Élaboration** :
exemple : définition de la forme de paramètres des requêtes, du langage utilisé pour spécifier les types
 - **Précision** :
exemple : définition d'instances spécifiques d'entités (objets, opérations, types)
 - **Restriction** :
exemple : élimination d'entités, ajout de restrictions sur l'utilisation.
- **Le principe du modèle à objets de CORBA** :
 - => Isolation des demandeurs de service (client) et des fournisseurs (serveur)
 - => Côté client
 - création et identité d'un objet
 - requête et opérations
 - type et signature
 - => Côté serveur
 - implémentation (méthodes, activation)
 - => + des éléments dépendants de la machine (modèle d'exécution et de contrôle)

La sémantique de l'objet

- Un client d'un service est une entité quelconque capable de demander un service
- Un objet est une entité
 - **identifiable**
 - **encapsulée**
 - **qui fournit 1 ou + services à des clients**
- **Les requêtes**

Il s'agit d'un événement auquel on associe une opération, un objet, 1 ou + paramètres, et un éventuel contexte de la demande.

La **forme de la requête** correspond à une description qui peut être évaluée plusieurs fois. On les définit soit dans un langage de "liaison" particulier (langage de liaison C pour IDL) ou avec des appels sur des interfaces d'invocation dynamique.

Une **valeur** est quelque chose qui peut être un paramètre dans une requête. Une valeur peut identifier un objet. Plus particulièrement, une valeur est une instance d'un type de donnée de l'IDL.

Une **référence d'objet** est un nom d'objet qui désigne un objet.

Une requête correspond à la **demande de service** d'un client. Un **résultat d'exécution** est retourné au client. Il peut s'agir d'une valeur de paramètre ou d'une exception.

Les paramètres sont des paramètres d'entrée, de sortie, ou d'entrée-sortie.

- **La création et la destruction des objets**

Du point de vue du client, il n'a pas de mécanisme spécial pour créer ou détruire un objet.

Le résultat de la requête de création fournit la référence du nouvel objet.

- **Les types**

Un type est une entité identifiable avec un prédicat associé défini sur des valeurs

Les types sont utilisés dans les signatures pour restreindre un paramètre ou un résultat.

Le type d'un objet est un type dont les membres sont des objets.

On a les types de base (entier, booléen, caractères,...) et les types construits (structures, séquence, union, ...)

- **Les interfaces**

Une interface est la description des opérations possibles que le client peut demander sur un objet.

Les interfaces sont spécifiées en IDL.

L'héritage d'interface fournit les mécanismes de composition permettant à un objet de supporter plusieurs interfaces.

- **Les opérations**

Une opération est une entité identifiable qui définit un service qui peut être demandé.

On identifie une opération par un identificateur d'opération.

Une opération a une signature qui décrit les paramètres, les résultats, les exceptions, le contexte, et une indication sur la sémantique d'exécution.

L'implantation de l'objet

=> correspond au comportement de l'objet sur un système

=> les activités nécessaires à la réalisation du comportement de l'objet incluent le calcul du résultat de la requête et la mise jour de l'état du serveur.

- **Le modèle d'exécution**

- Les données représentent **un composant de l'état** du système

- le code des services demandés qui permet de **modifier l'état du système**

- **un moteur d'exécution** est une **machine abstraite qui peut interpréter les méthodes**. Il définit un contexte dynamique pour l'exécution d'une méthode.

- **le modèle de construction**

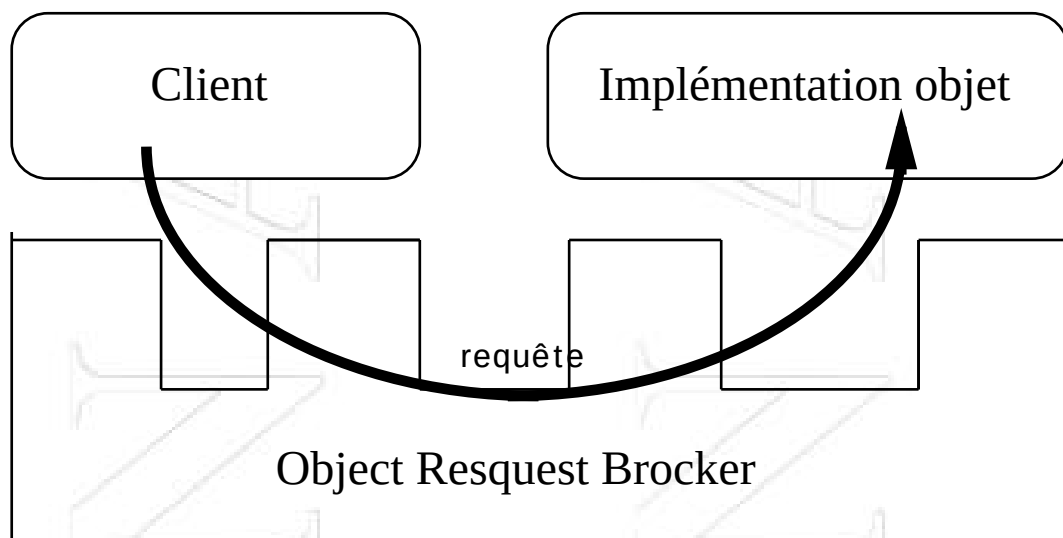
Le système à objet doit fournir **des mécanismes pour réaliser les comportements des requêtes**.

Ces mécanismes incluent **les définitions des objets, des méthodes et aussi l'infrastructure des objets** sélectionne **les méthodes à exécuter et la visibilité des données** pour les méthodes.

Il décrit également **les opérations de création et de destruction des objets**.

L'architecture de CORBA

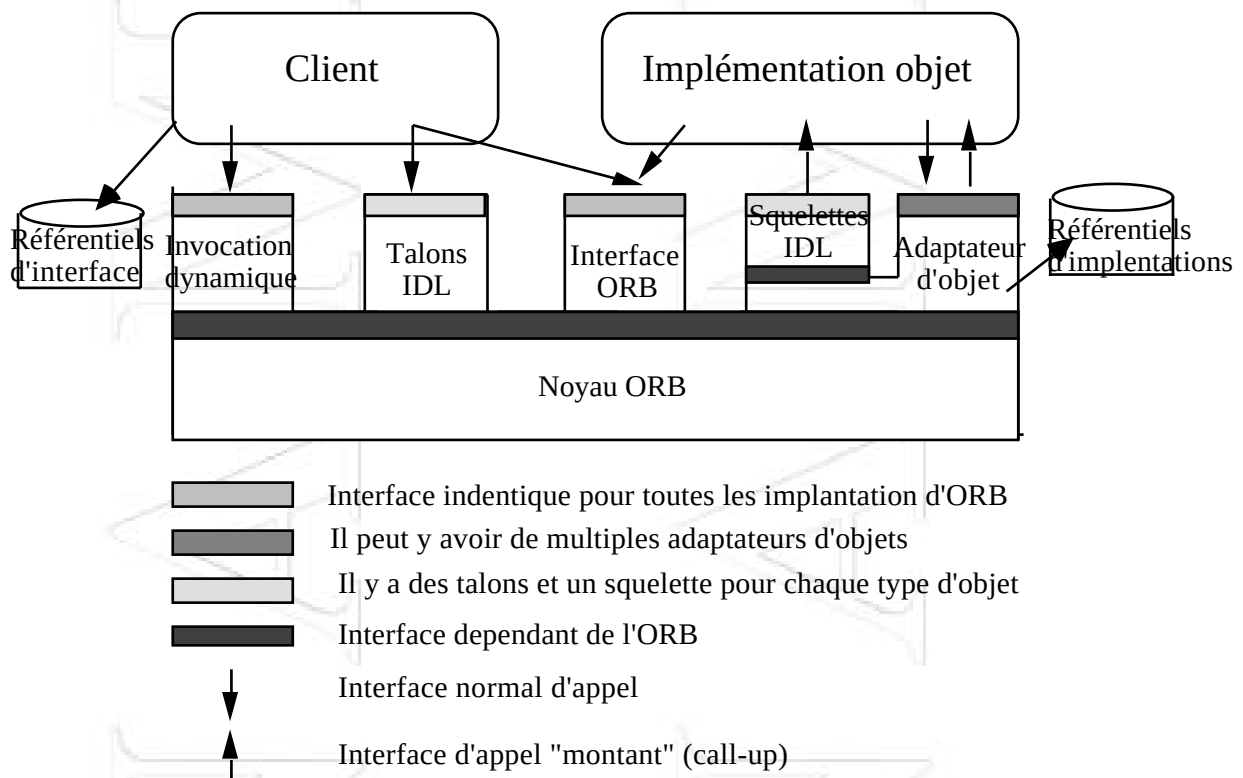
- a été structurée pour permettre **l'intégration et l'interopérabilité** d'un grand nombre de systèmes à objets.
- permet l'échange de messages via l'ORB et l'activation de méthodes d'objets



- l'ORB : **localise l'objet serveur, gère le formatage des messages portant les requêtes, leurs paramètres et les résultats, gère le retour du résultat au client.**

Contrôle aussi l'intégrité du système

Structure de l'interface de l'ORB



Pour réaliser une requête, le client utilise l'interface d'invocation dynamique (la même que pour l'objet cible) ou un talon IDL.

Le client peut aussi agir directement avec l'ORB pour quelques fonctions (retrouver la référence d'un objet par exemple).

L'implémentation de l'objet reçoit une requête comme un appel à travers l'IDL généré par le squelette.

L'implémentation peut aussi utiliser l'adaptateur d'objet et l'ORB.

Le client

- Le client est client par rapport à un objet spécifique.
- L'implantation d'un objet peut être client d'autres objets
- Le client demande l'exécution d'une requête en ayant la référence de l'objet et l'opération souhaitée
- Le client initialise la requête en appelant les primitives des talons spécifique à un objet (Interface IDL) ou en construisant la requête de manière dynamique.
- Le client voit la même interface sans savoir où est l'objet et sans connaître le langage de programmation utilisé pour l'implantation de l'objet.
- Les clients sont portables autant que possible.
- Le client n'a aucune connaissance sur l'implantation de l'objet, sur l'ORB utilisé, sur l'adaptateur d'objet utilisé.

Laurence Duchien

L'implantation objet côté client

L'interface d'un objet peut être défini de deux manières :

- **statiquement**

Par l'utilisation d'un Langage de Définition d'Interface (IDL) qui définit :

- les types d'objets,
- les paramètres des opérations

Proche du langage C++.

Après l'étape de compilation, les programmes sont figés.

- **dynamiquement**

Les requêtes sont spécifiées à l'exécution sans faire appel aux interfaces pré-définies.

Le client précise dans sa requête :

- l'objet auquel il doit accéder
- l'opération à réaliser
- les paramètres de l'opération

Le client est responsable de la spécification des paramètres et des résultats attendus. Ces informations sont communiquées par le référentiel d'interface (Interface Repository service) qui contient :

- Les descriptions des classes enregistrées
- Les méthodes et paramètres des classes

Les deux manières sont sémantiquement équivalentes.

L'implantation objet

- L'implantation objet **fournit la sémantique de l'objet** définie par l'instance de l'objet et le code des méthodes de l'objet.
 - **Plusieurs implantations** d'objets peuvent être supportées, en incluant des serveurs séparés, des librairies, un programme par méthode, une application encapsulée, une base de données orientée-objet,...
 - Généralement les implantations d'objets ne dépendent pas de l'ORB ou de la manière dont le client invoque l'objet.
 - L'implantation peut sélectionner les interfaces de services dépendant des ORB par le choix de l'adaptateur de l'objet.
 - Les implantations d'objet sont portables à travers tout ORB qui supporte le langage souhaité et qui implante l'adaptateur d'objet.
-

La référence d'objet

- La référence d'un objet est l'information nécessaire pour spécifier et désigner un objet avec un ORB. Client et implantations d'objet ont une vision de cette référence à travers le langage de liaison.
- Deux implantations d'ORB peuvent différer par leur choix de la représentation des références des objets.
- La représentation d'une référence d'un objet utilisé par un client n'est valide que pendant la "vie" de ce client.
- Tous les ORBs doivent fournir le même langage de liaison pour une référence d'objet pour un langage particulier de programmation.

Le talon IDL côté client

- Fournit les **interfaces statiques** des services d'objets.
- Ces talons pré-compilés définissent comment les clients invoquent les services correspondants sur les serveurs.
- Du point de vue du client, **le talon se comporte comme s'il s'agissait d'un appel local** = > C'est un mandataire pour un serveur d'objet distant.
- Les services sont définis en utilisant l'IDL, et les talons du client et du serveur sont **générés par le compilateur IDL**
- Le client doit avoir un talon IDL pour chaque interface qu'il utilise sur le serveur.
- Le talon contient **le code pour exécuter l'opération "d'empaquetage"** (codage et décodage du nom de l'opération, des paramètres et des valeurs)
- Les talons présentent un accès à des opérations dans un IDL défini pour un objet de manière unifiée de façon à ce que ce soit facile pour un programmeur familiarisé avec l'IDL et le langage de prédire la sémantique
- Si il a plusieurs ORBs disponibles, il peut y avoir différents talons correspondant à ces différents ORBs.

L'interface d'invocation dynamique (DII)

- Permet de faire **la construction d'invocations dynamiques** au moment de l'exécution
- Corba définit **des APIs standards pour regarder les données méta** qui définissent l'interface du serveur, générant les paramètres, faisant l'appel de méthodes et traitant le résultat
- Le client **spécifie l'objet qui est invoqué, l'opération à exécuter, l'ensemble des paramètres pour l'opération par un appel ou une séquence d'appel;**
- L'interface de l'invocation dynamique peut beaucoup varier d'un langage à l'autre

L'API du référentiel d'interface

- Permet **l'obtention et la modification des descriptions de toutes les interfaces** des objets enregistrés, les méthodes supportées et les paramètres.
- Corba appelle **ces descriptions : signature de méthode**
- Le référentiel d'interface est **une base de données** qui contient des versions d'interface en IDL définies des machines accessibles,
- Les APIs permettent aux objets d'accéder, de stocker, et de mettre à jour ces informations de manière dynamique.

L'interface ORB

- Consiste en **quelques APIs de services locaux** qui peuvent intéressés une application.

Par exemple, CORBA fournit le moyen de convertir une référence d'objet en une chaîne de caractères.

- C'est la même pour tous les ORB et ne dépend pas de l'interface de l'objet ou de l'adaptateur de l'objet
- Parce que la plupart des fonctionnalités de l'ORB sont fournies par l'intermédiaire de l'adaptateur de l'objet, des talons, des squelettes, de l'invocation dynamique, il y a très peu d'opérations
- Ces opérations sont utiles à la fois aux client et aux implantations d'objets.

Laurence Duchien

Le bus ORB (Object Request Broker core)(1)

L'ORB est responsable de :

- **des mécanismes pour trouver l'implantation** de l'objet pour la requête demandée
- **de préparer l'implantation de l'objet** pour recevoir la requête
- **de communiquer les données** de la requête

Il s'agit d'un mécanisme par lequel un applicatif client accède à des classes serveurs.

Les actions de l'ORB :

- **Localisation de l'objet** invoqué par le client,
- **Transmission des paramètres** de la requête
- **Transfert du contrôle** à l'implémentation de l'interface à travers le squelette d'IDL.
- Une fois la requête terminée, **retour** des résultats au client.

Le bus ORB (Object Request Broker core)(2)

L'ORB n'est pas implémenté par un seul composant, mais est **défini par rapport à ces interfaces.**

Toute implantation ORB qui fournit une interface appropriée est acceptable.

Les interfaces sont organisées en trois catégories :

- **les opérations qui sont les mêmes pour toutes les implémentations d'ORB.**
- **les opérations qui sont spécifiques pour des types particuliers d'objets,**
- **les opérations qui sont spécifiques pour des styles particuliers d'implémentations d'objets**

Différents ORBs peuvent avoir des choix d'implantations différents et fournir un ensemble de services aux clients et d'implantations d'objets qui ont des propriétés et des qualités différentes.

=> Corba est conçu pour supporter **une représentation de base des objets et des requêtes de communication** qui peut être différente selon les implantations d'ORBs.

=> Ce sont **les interfaces qui sont censées masquer les différences**

Les talons IDL du serveur (skeleton)

- Fournit **les interfaces statiques** de chaque service exporté par le serveur.
- Ces talons, comme ceux du client, sont **créés en utilisant un compilateur d'IDL**.
- L'interface est en général **une interface "up-call"**
- L'ORB les appelle à travers le squelette
- L'existence d'un squelette n'implique pas l'existence du talon client correspondant (utilisation de l'interface dynamique pour le client)

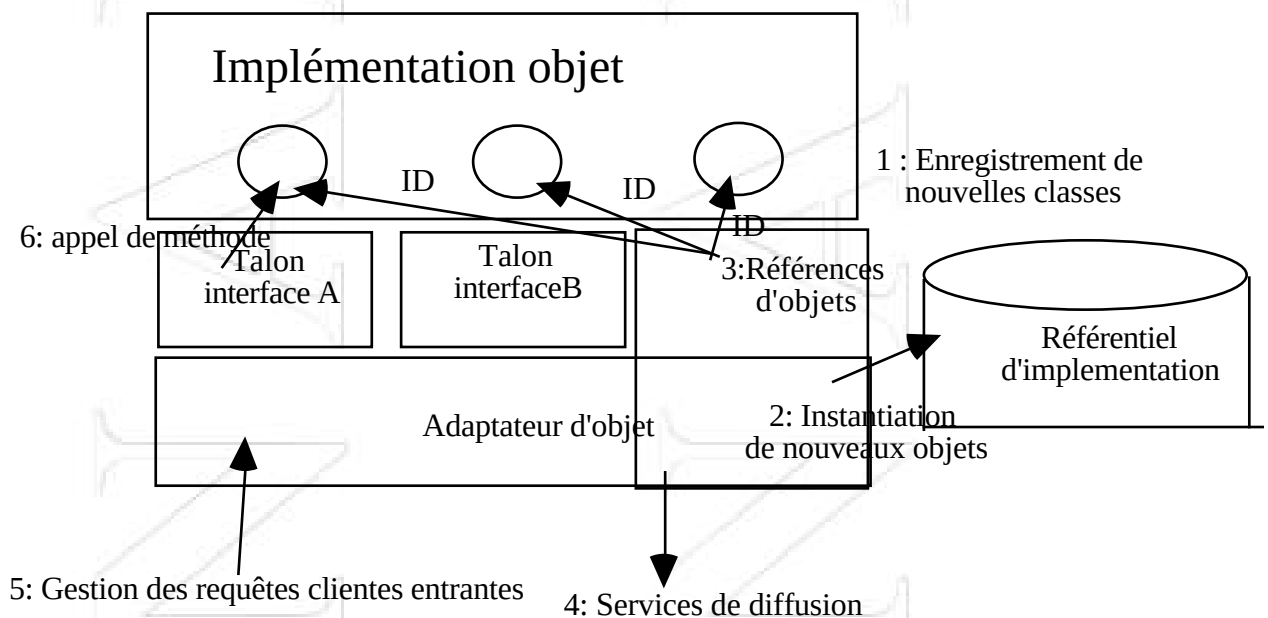
L'interface dynamique (squelette dynamique) (DSI)

- Introduit dans CORBA 2.0
- Fournit un **mécanisme de liaison dynamique** pour les serveurs ayant besoin de gérer des appels de méthodes entrant **pour des objets n'ayant pas de squelettes compilés(ou talons)**
- Le squelette dynamique regarde la valeurs des paramètres dans le message entrant pour voir à qui est destiné le message : **détermination de l'objet et de la méthode cible**
- Ces squelettes dynamiques sont très utiles dans le cadre d'implantation **de ponts génériques entre des ORBs.**
- Peuvent aussi être utilisés **par des interpréteurs et des langages de script pour générer dynamiquement des implémentations d'objets**
- le DSI est l'équivalent au DII

Laurence Duchien

L'adaptateur d'objet

- Un adaptateur d'objet est un moyen **permettant à l'implantation d'objet d'accéder aux services fournis par l'ORB.**
- On espère qu'il y aura peu d'adaptateurs d'objet, largement diffusés, avec des interfaces qui sont appropriés pour des types spécifiques d'objets.



Ses fonctions :

- génère et interprète les références d'objets
- gère les invocation de méthodes
- gère la sécurité des interactions
- active et désactive l'implantation des objets
- gère la correspondance entre les références d'objet et leur implantations
- enregistre les implantations

-
- Souvent l'ORB fournira l'ensemble de ces services.

Si ce n'est pas le cas, l'adaptateur d'objet fournit une interface vers lui.

Dans le cas contraire, l'OA doit l'implanter.

- La spécification de CORBA définit **l'adaptateur de base**. D'autres adaptateurs devraient apparaître bientôt.
- La grande variété de granularité des objets, de la politique de gestion des objets, du style d'implantation **rend difficile la fourniture d'une simple interface** qui soit pratique et efficace pour tous les objets

Différents adaptateurs d'objets

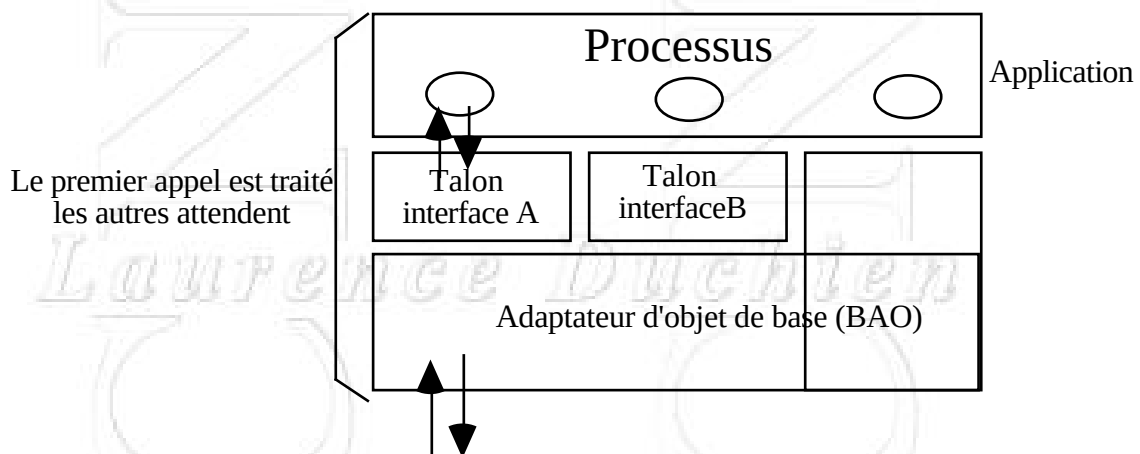
- Un BOA pour serveur partagé :

Des **objets multiples peuvent résider sur le même processus**.

Le **BOA active le processus serveur** la première fois qu'une requête est exécutée sur un objet.

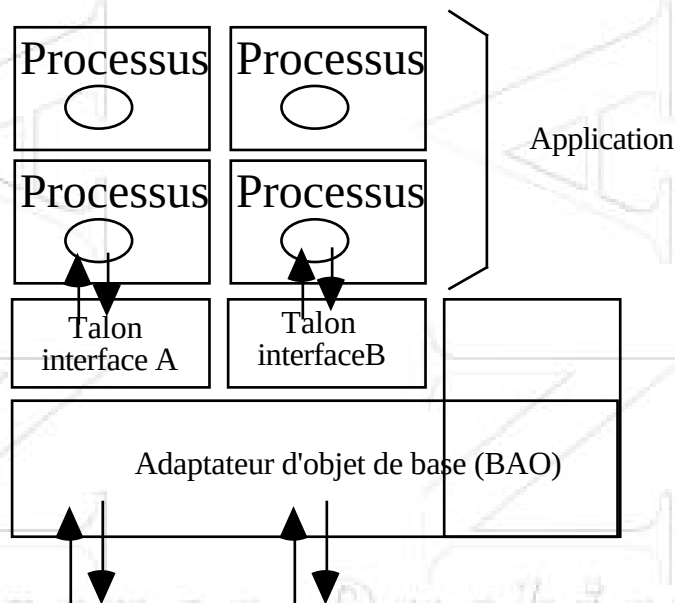
Toutes les requêtes suivantes de l'objet sont alors délivrées à ce processus serveur.

Un serveur gère alors une requête à la fois



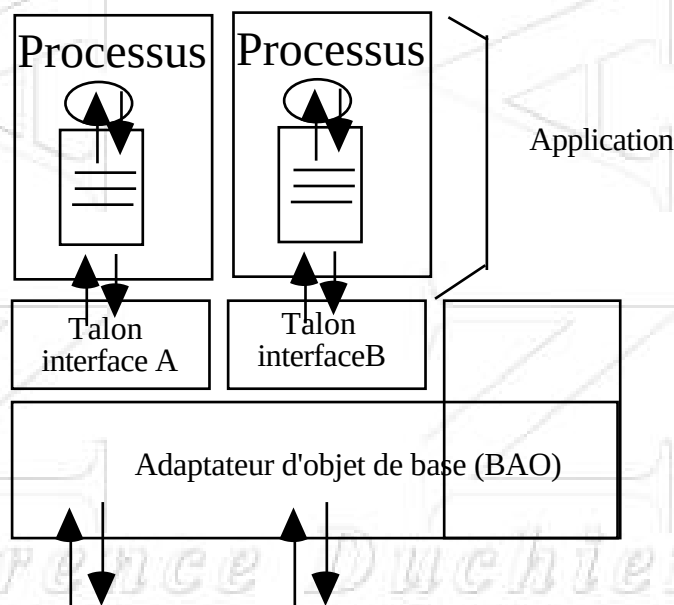
- Un BOA pour serveur non partagé :

- Chaque **objet réside dans un processus serveur différent.**
- Un nouveau serveur est activé à la première requête sur l'objet
- Un nouveau serveur est démarré quand une requête est effectuée pour un objet qui n'est pas encore actif, même si un serveur d'un autre objet avec la même implémentation est actif.



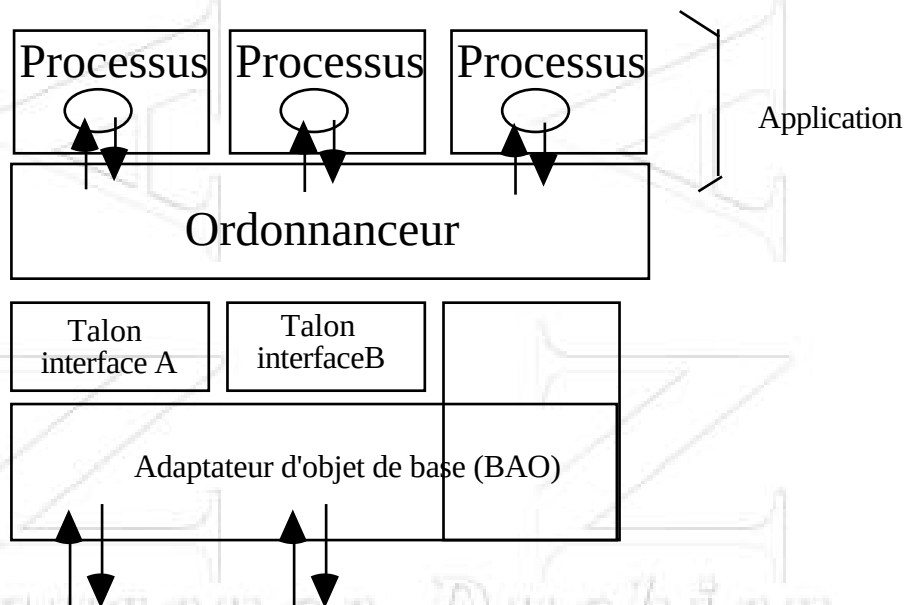
- Un BOA avec un serveur par méthode :

- Un nouveau serveur est démarré à chaque fois qu'une requête est appelée
- Le serveur est actif pendant seulement la durée de la méthode
- Plusieurs serveurs par objet et même plusieurs serveurs par méthode



- Un BOA avec serveur persistant :

- Les serveurs sont activés en dehors de l'adaptateur,
- Un BOA peut démarrer l'application serveur qui notifie le BOA qu'elle est prête à accepter le travail
- Le BOA envoie les activations à un seul processus : l'ordonnanceur



Référentiels des interfaces et des implantations

- Fournit **un référentiel d'information à propos des classes** qu'un serveur supporte, les objets qui sont instanciés et leurs IDs.
- Sert aussi de **place commune pour sauvegarder des informations additionnelles** avec l'implémentation des ORBs (information de trace, sécurité, informations administratives)
- Il permet pour un programme **l'utilisation d'un objet, dont l'interface n'était pas connue au moment de la compilation**, d'être capable de déterminer quelles sont les opérations valides sur l'objet et de réaliser une invocation sur elles.

Il fournit également **des informations stockées pour réaliser le contrôle de type des signatures des requêtes, des librairies de talons ou de squelettes, des primitives** qui peuvent traiter un type particulier d'objets, **des infos pour du debug,....**

- Le **référentiel d'implantation** contient des **informations qui permettent à l'ORB de localiser et d'activer l'implantation** des objets.

Bien que ce soit très souvent spécifique à un ORB ou aux environnements d'exploitation accueillant les objets, il s'agit du lieu où sont stockées les informations d'implantation, de politique d'utilisation, d'activation,....

On y stocke aussi les infos pour du debug, du contrôle d'administration, d'allocation de ressource, de sécurité,...

L'interface d'ORB coté serveur

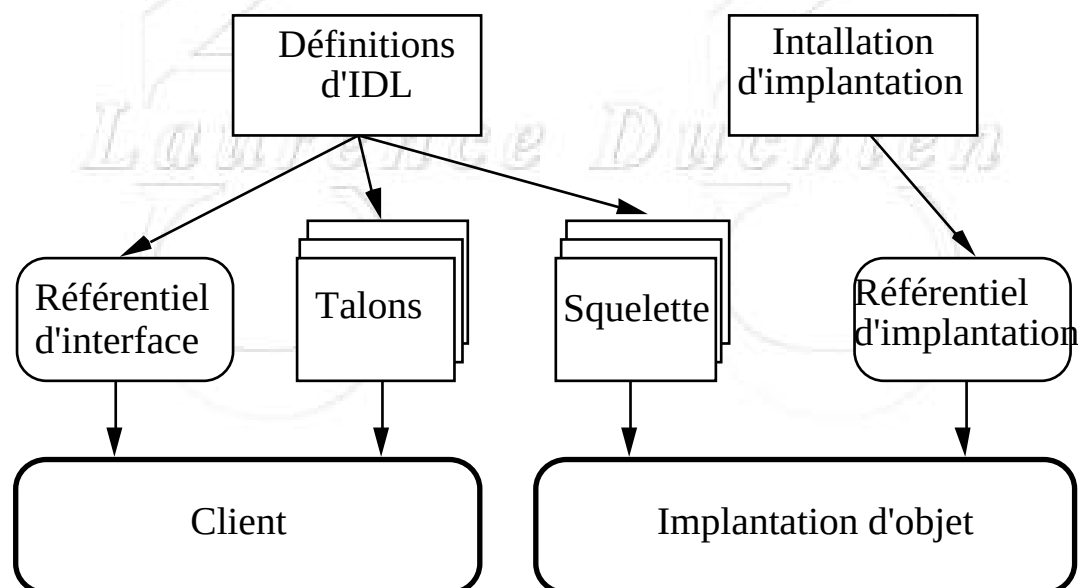
- Correspond à un ensemble d'APIs pour des services locaux qui sont identique au côté client.

Le serveur

- Doit fournir la sémantique de l'objet
- Définir les données pour l'implantation
- Définir le code pour les méthodes de l'objet
- Peut utiliser d'autres objets (et éventuellement retourner des références vers d'autres objets)
- Ou peut manipuler d'autres logiciels qui ne sont pas eux-mêmes des objets.

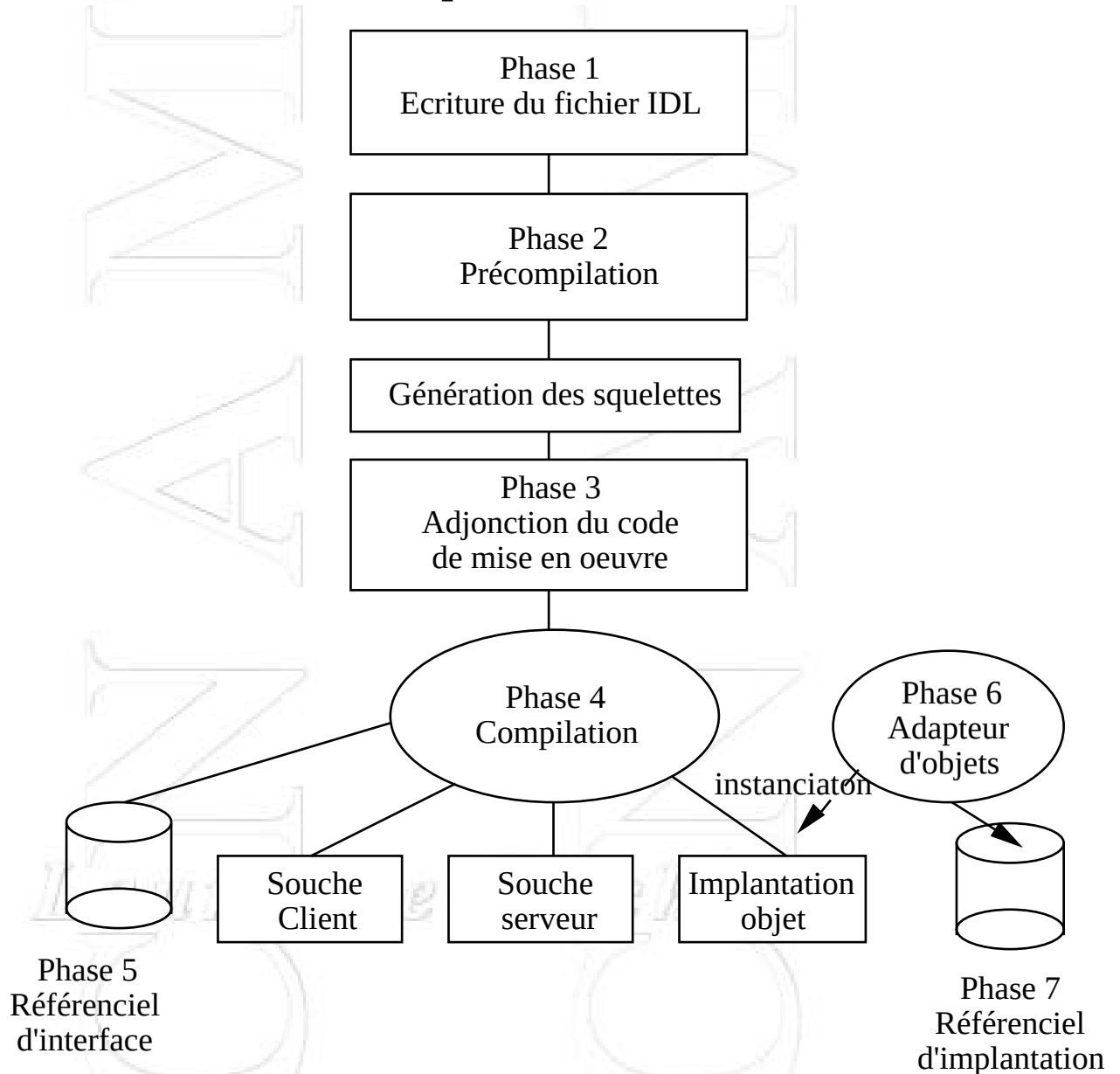
L'implantation côté serveur

- Squelette IDL (IDL Skeleton) ou Souche serveur
Définition des méthodes de l'implantation
- L'adaptateur d'objet
Mécanisme de mise en oeuvre de l'application serveur
 - génération et interprétation des références d'objets
 - prise en compte des appels clients et routage vers les méthodes appropriées
 - diffusion des services disponibles sur l'ORB
- Utilisation d'une base de données de mise en oeuvre (référentiel d'implémentation) qui contient des informations sur les classes serveurs et les objets instanciés.



L'invocation de méthodes

Solution 1 : l'accès statique



Phase 1 : Description IDL de l'interface du type à programmer

- . types d'objet
- . Attributs
- . Méthodes
- . Paramètres à fournir

Phase 2 : Précompilation du fichier IDL

Création des squelettes C d'implantation des classes serveurs

Phase 3 : Ajout par le développeur du code de mise en oeuvre des méthodes dans les squelettes. Création des classes serveurs.

Phase 4 : Compilation du code source résultant et génération de 4 fichiers:

- . fichier d'importation qui décrit les objets pour une interface de référentiel
- . talon Client pour les méthodes définies en IDL. Ces talons sont invoqués par un programme client
- . talon Serveur qui appelle les méthodes sur le serveur
- . Code d'implantation des classes serveurs

Phase 5 : liaison des définitions de classes avec le référentiel d'interfaces : utilisation d'un utilitaire pour lier les informations de l'IDL dans un référentiel d'interface.

Phase 6 : Instantiation des objets sur le serveurs (par l'intermédiaire de l'adaptateur d'objets)

Phase 7 : Enregistrement des objets exécutables dans le référentiel d'implantation

- facile à programmer
- fournit un contrôle de type

Solution 2 : l'accès dynamique

Phase 1 : Obtention d'une méthode dans le référentiel d'interface

->lookup_name() -- appel de méthodes d'objets de localisation et description

-> describe() -- permet l'obtention d'une description complète de l'IDL

Phase 2 : Création d'une liste d'arguments avec les paramètres à passer

Création d'une structure de données pour le passage de paramètres appelée NameValuelist et avec autant d'appels add_args que nécessaire.

-> create_list()

-> add_arg()...add_arg()...add_arg()...

Phase 3 : Création de la requête avec :

- la référence de l'objet
- le nom de la méthode à invoquer
- la liste d'arguments

create_request(ObjectReference, Method,ArgumentList)

Phase 4 : Appel de la requête :

3 moyens pour effectuer l'invocation à distance

- en utilisant un RPC : invoke()
- en utilisant le mode asynchrone : send() et get_response()
- en appel asynchrone simple : send()
- plus flexible
- permet d'ajouter des nouvelles classes au système sans changement dans le code du client
- écriture de code générique avec des API dynamiques

Quelques exemples d'ORBs

- Il existe un grand nombre d'implantations d'ORB possible avec la spécification de CORBA:

- ORB avec un client et une implantation résidents

S'il existe un mécanisme de communication qui convient, un ORB peut être implanter avec les primitives résidentes dans le client et l'implantation

On utilise les IPC locaux ou les accès directs à un service pour établie les communications avec les implantations.

- ORB basé sur un ou + serveurs

Pour centraliser la gestion de l'ORB, tous les clients et les implantations peuvent communiquer avec un ou plusieurs serveurs dont le travail est de router les demandes venant des clients vers les implantations.

- ORB basé sur le système

Pour assurer la sécurité, la performance ou une certaine robustesse, l'ORB peut être fourni comme un simple service de base du système d'exploitation.

- ORB basé sur les librairies

Pour des objets dont les implantations peuvent être partagées, l'implantation peuvent être dans une librairie. Dans ce cas les talons pourront être les méthodes réelles.

Il faut assumer le fait qu'il est possible pour un programme client d'accéder aux données des objets et que l'implantation demande aux client de ne pas endommager les données...

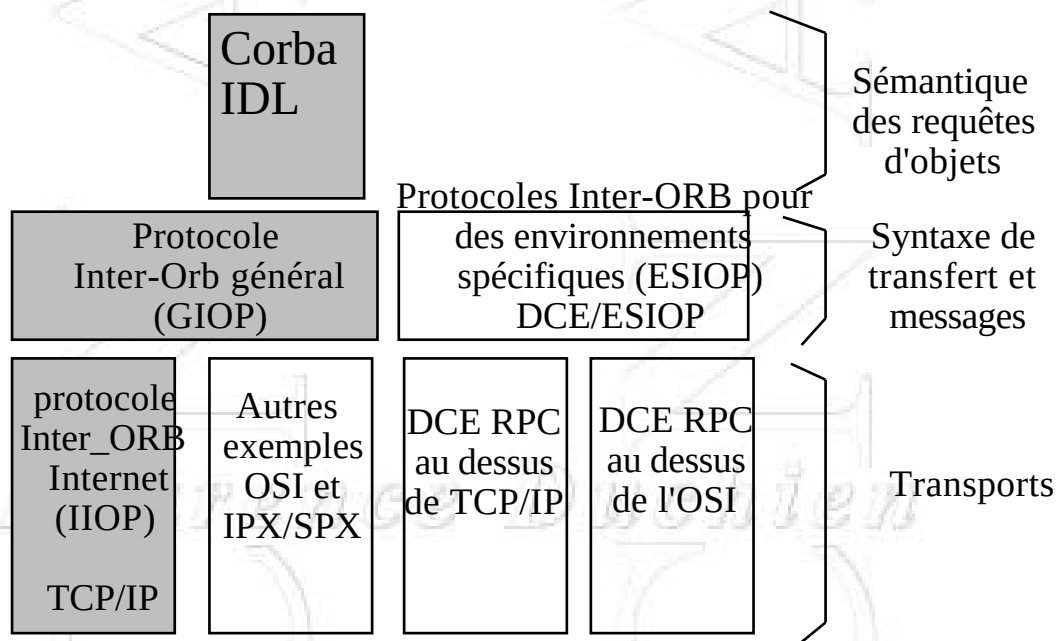
L'interopérabilité

- L'apport de Corba 2.0 est la spécification d'un protocole obligatoire :

Internet Inter-ORB Protocol (IIOP)

L'IIOP est basé sur TCP/IP et correspond à un protocole d'échange qui sert de "backbone".

Tout ORB qui dit respecté les normes CORBA doit soit implémenté de manière native l'IIOP ou une passerelle vers un IIOP



- **GIOP** : correspond à un ensemble de formats de messages et de représentation de données pour la communication entre ORBs.

- a été construit pour les interactions entre ORBs
- il doit travailler directement au-dessus de n'importe quel protocole de transport en mode connecté
- 7 formats de message pour couvrir les différentes sémantiques des appels et des réponses
- La mode de représentation est CDR (Common Data Représentation. Il permet de coder les types définis dans les IDL en une représentation de message plat transportable à travers le réseau

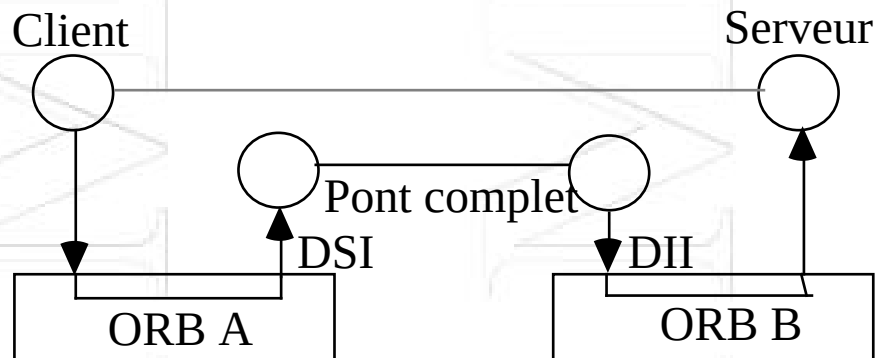
- **IIOP** : spécifie comment les messages GIOP sont échangés au-dessus de TCP/IP

- **ESIOPs** (Protocoles Inter_ORB pour des environnements spécifiques) : sont utilisés pour des interopérabilité avec des réseaux spécifiques.

- DCE est le premier des ESIOP à être opérationnel

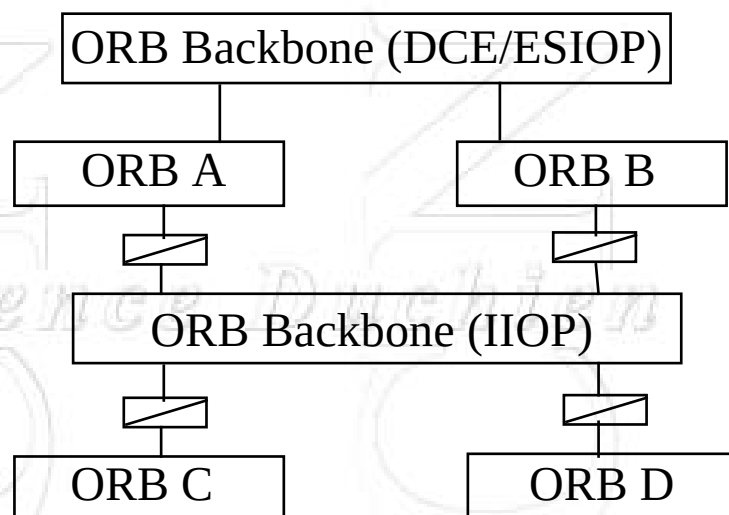
Les ponts entre ORBs

- Ponts complets



- ORB fédérés :

- on utilise des ponts inter-ORB et des IIOPs pour créer des topologies très flexibles



L'IDL (langage de définition d'interface)

- définit les types d'objet par spécification des interfaces;
- Une interface consiste en **un ensemble d'opérations nommées et des paramètres sur ces opérations**.
- l'IDL est le moyen par lequel **un objet particulier indique ses clients potentiels les opérations disponibles et comment les invoquer**.
- C'est un langage **neutre et totalement déclaratif**
- A partir des définitions d'IDL, il est possible de **mettre en correspondance les objets CORBA** dans un langage de programmation particulier ou avec des objets systèmes.

Laurence Duchien

La structure de l'IDL CORBA

```
module <identifïer>
    /*définition d'un contexte nommé*/
{
    <type declarations>;
    <constant declarations>;
    <exception declarations>;

    interface <identifïer> [:<inheritance>]
        /* définition d'une classe CORBA*/
        {
            <type declarations>;
            <constant declarations>
            <attribute declarations>
            <exception declarations>

            [<op_type>] <identifïer>(<parameters>)
            [raises exception][context];
            /*définition d'une méthode*/
            ....

            [<op_type>] <identifïer>(<parameters>)
            [raises exception][context];
        }
        interface <identifïer>[:<inheritance>]
        ...
    }
}
```

Un exemple d'IDL

```
module Mes_Animaux;
{
    /* définition de la classe Chien */
    interface Chien : animal_de_compagnie, animal;
    {
        attribute integer age;
        exception Non_Interesse {string explication};

        void aboie (in short combien_de_temps)
            raises (Non_Interesse );
        void assis (in string ou)
            raises (Non_Interesse );
        void grogne (in string apres_qui)
            raises (Non_Interesse );
    }

    /* Définition de la classe chat */
    interface Cat : animal;
    {
        void Eat();
        void Viens();
    }
} /* fin de Mes_animaux */
```

Laurence Duchien

Le langage de définition de l'interface (IDL) de CORBA

- C++
- Types des IDL
 - . Entier : long (32 bits), short (16 bits), unsigned long, unsigned short
 - . Réel : float (32 bits), double (64 bits)
 - . Caractères : char (8 bits)
 - . Booléen : boolean
 - . Octet : octet (8 bits, sans conversion)
 - . n'importe quoi : any (permet n'importe quel type d'IDL)
 - . Structures : struct
 - . Unions: union
 - . Enumeration : enum
 - . Sequence : sequence
 - . Strings : string (chaîne de caractères)
 - . Typedefs
- L'interface objet défini en accord avec les types de base
 - . Attributes
 - . Methodes
- Le préprocesseur C++ des IDL
 - . La substitution de macro
 - . La compilation conditionnelle
 - . Inclusion de fichiers sources
- Les commentaires, les déclarations différées, les constantes de C++
- Les exceptions
 - . Les exceptions systèmes
 - . Les exceptions définies dans l'implantation objet
- Les types de méthodes
 - . normal
 - . en cas de panne : sémantique au plus une fois
 - . sans panne : sémantique exactement une fois
 - . un seul sens
 - . le mieux possible

L'IDL et l'héritage

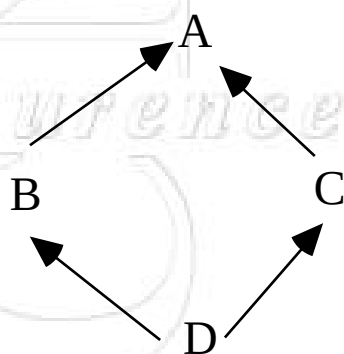
- L'héritage d'une interface par une autre interface
- Possibilité d'**héritage multiple**
- Il est illégal d'hériter de deux interfaces avec le même nom d'opération ou d'attribut.
- Il est aussi illégal de redéfinir un nom d'opération ou d'attribut dans l'interface dérivé
- Il est légal **de redéfinir un type, une constante, une exception** dans une interface dérivée.
- Légal

Interface A {...}

Interface B : A {...}

Interface C : A {...}

Interface D : B , C {...}



Exemples IDL et DII (Dynamic Invocation Interface)

```
//IDL
interface foo {
    string op1(in long x, inout float y, out string x);
};

// Client C++/IDL (Orbix/INS bindings)
char *result;
long x;
float y;
char *z;
foo *myFoo;

myFoo=foo::_bind();// Bind to any "foo" on the network
result = myFoo->op1(x, y, z);

// Client C++/DII (Orbix bindings)
CORBA::ObjectRef target = CORBA::Object::_bind("foo");
CORBA::Request r(target, "op1");
r << x << CORBA::inOUTMode << y << CORBA::outMode << z;
r.invoke();
r >> result;

// C++ Object Implementation (Orbix)
class foo_impl : public fooBOAImpl
{
public:
    virtual char *op1(long x, float &y, char &*z,
        CORBA::Environment &env);
}

char *
foo_impl::op1(long x, float &y, char &*z, CORBA::Environment &)
{
    y = (x+y) * .5;
    z = new char[30];
    strcpy(z, "hello");
    return(z);
}

main()
{
    foo_impl obj;
    CORBA::Orbix.impl_is_ready("foo");
}
```

Le référentiel d'interface Corba 2.0

- fédération possible de référentiels d'interface qui coopèrent à travers les ORBs en maintenant l'autonomie des administrateurs locaux
- Deux caractéristiques supplémentaires :
 - un ensemble de nouvelles méthodes pour mettre à jour de manière incrémentale les contenus des référentiels à l'exécution
 - des identificateurs de référentiels uniques et globaux à utiliser dans l'espace de noms réparti

Définition :

C'est une base de données en ligne de définitions d'objets.

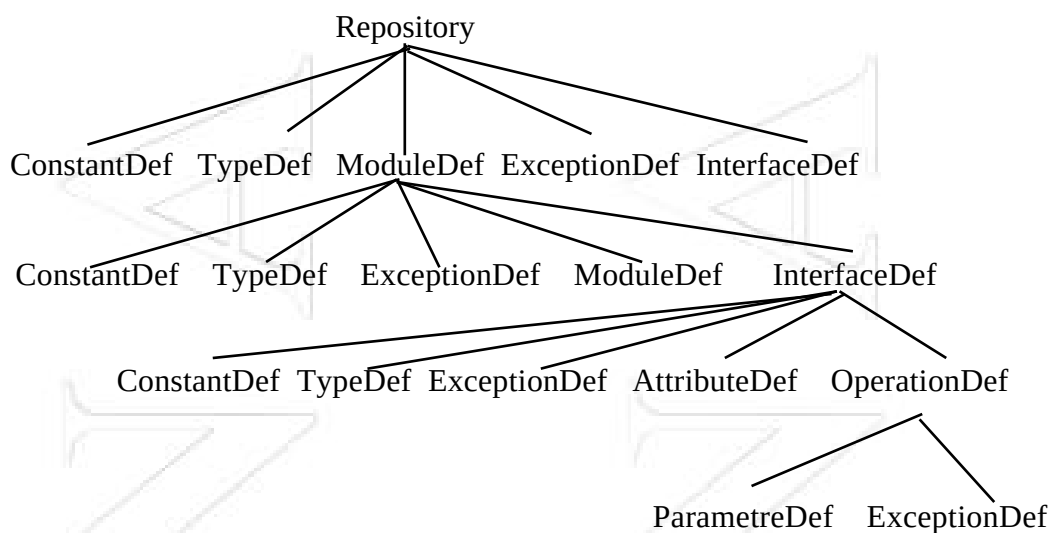
- Les définitions d'objets sont fournis par les compilateurs d'IDL ou à travers les fonctions d'écriture des référentiels d'interface.
- La spécification CORBA ne décrit pas l'organisation et la recherche d'information dans le référentiel, mais définit un ensemble de classes dont les instances représentent l'information qui sont dans le référentiel.

A quoi sert le référentiel d'interface :

- permet à l'ORB de comprendre la définition des objets sur lequel il travail par accès dynamique aux définitions :
 - fournit un contrôle de type des signatures de méthodes
 - aide les ORB à se connecter entre eux
- - fournit des informations meta aux client et aux outils
 - fournit la propre description des objets.

Les classes du référentiel d'interface

- Le référentiel d'interface est implémenté sous forme d'un ensemble d'objets persistants
- CORBA groupe ses méta-données dans des modules qui représentent des espaces de noms
- CORBA définit une interface pour chacune de ses 8 structures IDL :



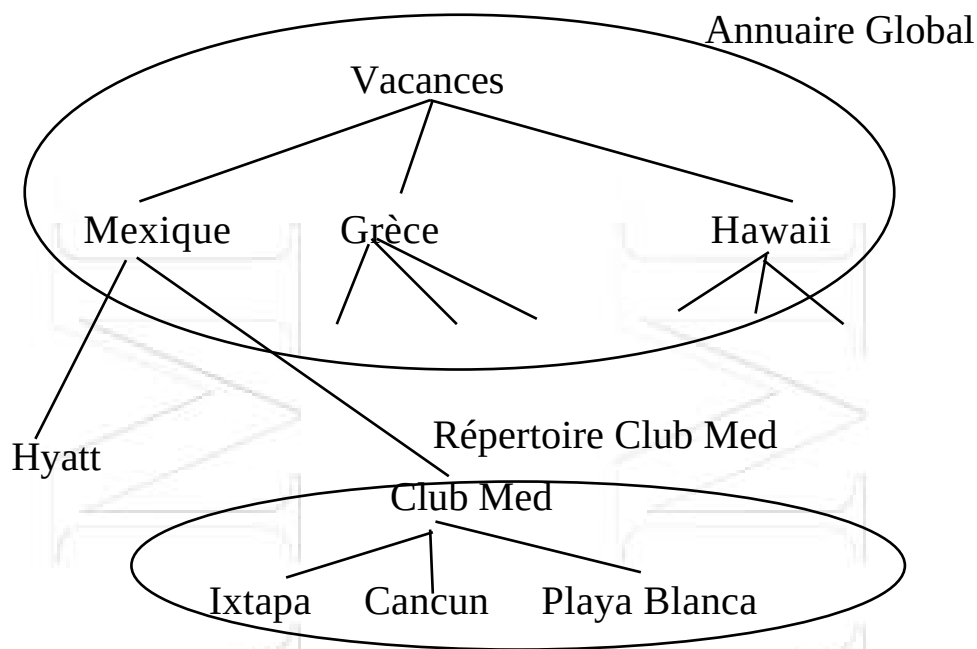
Les services CORBA : le nommage

le service de noms : les pages blanches de l'annuaire

=> permet de localiser un objet à partir de son nom

Nom : forme facilement reconnaissable par un humain.

- Le service de nommage fait **l'association entre le nom et la référence de l'objet.**
- Cette opération est appelée la liaisons de noms (**name-binding**)
- Un contexte de nommage est un espace de noms dans lequel les noms des objets sont uniques.
- Chaque objet a une référence unique (un ID). Mais on peut associer optionnellement un ou plusieurs noms à une référence d'objet.
- Le service encapsule de manière transparente les services d'annuaires existants comme X500, DNS, NIS+, CDS.
- Utilisation d'une hiérarchie où le client navigue à travers différents arbres de contexte de nommage



- **Un nom simple** : Playa Blanca
- **Un nom composé** : un nom simple + des contextes
Playa Blanca + Club Med, Mexico, Vacances
- Chaque objet a une structure avec deux attributs :
 - **l'identifieur** : la chaîne de caractères
 - **type** : une chaîne dans laquelle on a un attribut de description pour le nom (exemple type fichier)

L'interface du service de nommage

- Interface **NamingContext**
/* **liaison de nom à objet avec nom unique** */
 resolve() /* trouve un objet lié à un nom
 dans un contexte donné */
 list() /* permet d'itérer la demande sur
 un ensemble de noms */
 destroy() /* efface le contexte nommé */
 new_context() /* retourne le nom du contexte */
 bind() /* associe un nom d'objet à un
 contexte de liaison */
 rebind()
 bind_context()
 rebind_context()
 bind_new_context()

• Interface **BindingIterator**
 next_one() /* permet d'itérer sur un objet
 BindingIterator */
 next_n() /* même chose */
 destroy() /* permet de libérer l'itération */

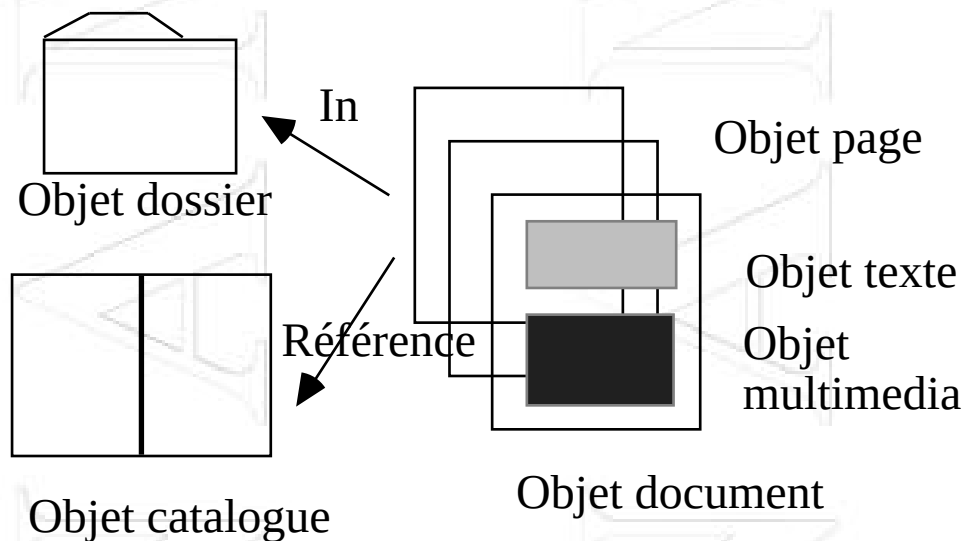
Le service de courtier (trader)

=> **correspond aux pages jaunes de l'annuaire**

- Le standard n'est pas encore stabilisé (fin de l'année 96 ?)
- Permet à **un nouveau service de s'enregistrer** par l'intermédiaire **du courtier**.
- Le fournisseur de service donnera entre autres les informations suivantes :
 - **Une référence d'objet** : C'est la référence que les clients utilisent pour se connecter au service voulu et pour invoquer les opérations
 - **Une description du type de service offert**
- Le courtier stocke ces informations de manière permanente.
- Le client **utilise le courtier pour retrouver la liste des services ou pour demander un service par type**
- Le courtier doit retrouver **la meilleure réponse pour le client en fonction du contexte et des offres des fournisseurs**
- Les courtiers de différents domaines **peuvent créer des fédérations** avec des politiques de gestion et de sécurité.

Les services CORBA : le cycle de vie

- Fournit les opérations **de création, de destruction, de déplacement, de copie des objets**
- Ces services ont été étendus à la notion de groupe d'objets
- **Un exemple de cycle de vie**



Le cycle de vie maintient **un graphe de l'ensemble de ces associations**

Un déplacement du document vers un autre dossier fera en sorte **que tous les objets dépendants** (les pages et leur contenu, les références) **soient également déplacés** vers le dossier cible

Les interfaces du cycle de vie

Les services du cycle de vie travaillent à la fois sur les objets eux-mêmes et leurs relations entre-eux.

- Pour créer un nouvel objet, un client doit trouver un objet "factory" (c'est un objet qui sait comment instancier un objet de cette classe). Par cet objet factory, la demande création est effectuée et il y retour d'une référence d'objet.
- On peut aussi créer des clones d'objet par copie
- L'objet "factory" **doit allouer des ressources, obtenir les références et enregistrer les nouveaux objets auprès de l'adaptateur d'objet et des référentiels d'implantation.**

Les interfaces :

```
FactoryFinder {  
    find_factories();  
}
```

```
GenericFactory {  
    create_objetc()  
}
```

```
LiveCycleObjext {  
    copy();  
    move();  
    remove()  
}
```

Il existe des addendum récents pour supporter les gestion de structures avec des accès sur des noeuds, des rôle ou des relations

Les services CORBA : les événements

- Permet à des objets **d'enregistrer ou de retirer dynamiquement leur intérêt pour des événements**
- **Un événement est une occurrence** spécifiée pour avoir un intérêt quelconque pour un ou plusieurs objets
- **Une notification** est un message qu'un objet envoie vers les parties intéressées pour les informer qu'un événement spécifique est arrivé.
- **L'objet qui génère l'événement ne connaît pas les objets intéressés** par cet événement => C'est le rôle de ce service de trouver les objets à qui faire part de l'événement.
- Le service événement crée **un canal de communication** faiblement couplé entre les objets.
- Deux rôles
 - **les fournisseurs** : produisent des événements
 - **les consommateurs** : reçoivent et traitent les événements
- **Deux modèles de communication**
 - **Un modèle "push"** : le fournisseur prend l'initiative et initialise le transfert des données de l'événement vers le consommateur
 - **Un modèle "pull"** : le consommateur prend l'initiative et demande le transfert des données de l'événement au fournisseur.
- **Un canal événement est un objet** intervenant entre du consommateur et du fournisseur d'événement

Les services CORBA : les transactions

- L'un des services les plus importants de CORBA et le plus difficile à mettre en œuvre...
 - L'ORB deviendra la prochaine génération de moniteur TP...
 - Les **caractéristiques** des services de transactions objet :
 - **supportent les transactions plates et emboîtées**
 - permettent **aux applications ORB et non ORB de participer à une même transaction** (à condition d'adopter le standard X/OPEN DTP)
 - supportent **des transactions qui recouvrent des ORBs hétérogènes**. Des objets sur de multiples ORBs peuvent participer à une seule transaction.
 - **supportent des interfaces IDL existants**: une simple interface supporte des implémentations transactionnelles et non transactionnelles.
- => Pour faire un objet transactionnel, **on utilise une interface ordinaire qui hérite de la classe abstraite OTS**

Les éléments du service transactionnel

- **le client transactionnel**

- correspond à un ensemble d'invocations de méthodes qui sont délimitées par des marques de début et fin de transactions,
- l'ORB intercepte le début de transaction et le dirige vers le service transactionnel.
- l'ORB fait également suivre vers le service transactionnel lorsque le client valide ou abandonne la transaction.
- le client ne gère que le strict nécessaire : il démarre la transaction, fait des invocations de méthodes, valide ou abandonne la transaction

- **Le serveur transactionnel :**

- c'est un ensemble d'objets concernés par la transaction mais n'ayant pas d'états récupérables
- l'ORB propage les contextes des transactions de manière implicite quand les objets appellent des ressources récupérables.
- le serveur transactionnel ne participe pas à la terminaison de la transaction, mais il peut forcer une transaction à abandonner

- **Un serveur de recouvrement**

- c'est une collection d'objets dont les données sont affectées par la validation ou l'abandon d'une transaction
- on utilise des objets qui permettent d'avoir un coordinateur pour orchestrer la transaction et un protocole de validation à deux phases

Les services CORBA : la concurrence

- Fournit des interfaces **pour acquérir et relâcher des verrous** afin de permettre à des clients multiples **d'accéder en concurrence à des données partagées**,
- Supporte à la fois **le mode transactionnel et un mode non transactionnel**
- a été conçu pour **être utilisé avec OTS** pour coordonner les transactions
- Le service de contrôle de concurrence et les transactions :
un client pour acquérir des verrous doit choisir l'un des moyens suivants :
 - **au nom d'une transaction** : dans ce cas c'est le service de transaction qui gère le relâchement des verrous lors de la validation ou de l'abandon de la transaction
 - **au nom d'un client non transactionnel** : la responsabilité du relâchement des verrous est à la charge du client.
- **Les verrous** :
 - **un verrou est un jeton** qu'un client détient dans le but d'accéder à une ressource.
 - **le rôle est le partage de ressource et la gestion de conflits éventuels entre plusieurs clients.**
 - il existe plusieurs modes de gestion selon différentes catégories d'accès: **lire, écrire, intention de lire, intention d'écrire, lecture avec conflit sur soi-même** (pour prévenir les interblocages)

- **ensemble de verrous**

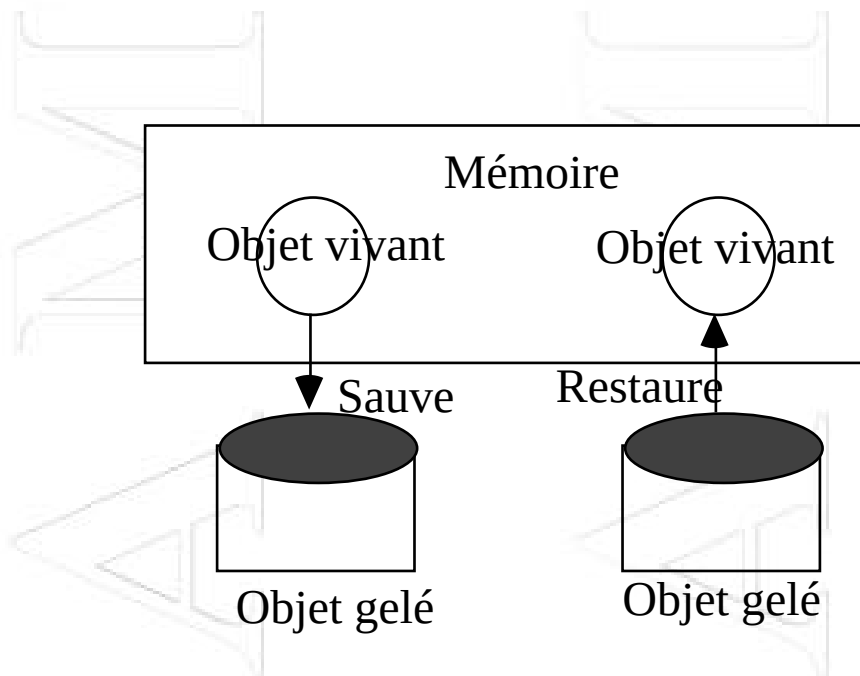
- Le service de contrôle ne définit pas la granularité des ressources qui sont verrouillés.

- il définit un "lockset" qui est une collection de verrous associés à une seule ressource

exemple : si un objet est une ressource, il doit créer de manière interne un lockset et le maintenir.

- le service de contrôle de concurrence définit un coordinateur de verrous qui gère le relâchement de verrous.

Les services CORBA : la persistance



- Le temps de vie d'un objet peut être infini
- Le service de persistance des objets (POS) permet de sauvegarder l'état d'un objet sur une mémoire permanente.
- Le POS peut gérer des objets spécialisés tels que des fichiers, des bases de données relationnelles, les bases de données objet, des documents composés,....
- Le POS fournit une interface aux clients pour sauvegarder des objets sans qu'ils aient à connaître les mécanismes de stockage.

Sauvegarde à un niveau contre la sauvegarde à deux niveaux

- à un seul niveau

POS gère pour **les bases de données à objets un seul niveau de stockage.**

Dans ce cas, le client n'est pas conscient du fait que les données soient en mémoire ou sur disque.

On a une un gros système de stockage virtuel : mémoire et stockage persistant sont la même chose.

- à deux niveaux

POS gère pour les fichiers et les bases de données relationnelles un système à deux niveaux.

La mémoire est alors séparée du système de stockage : l'objet doit passer explicitement de la mémoire au système de sauvegarde.

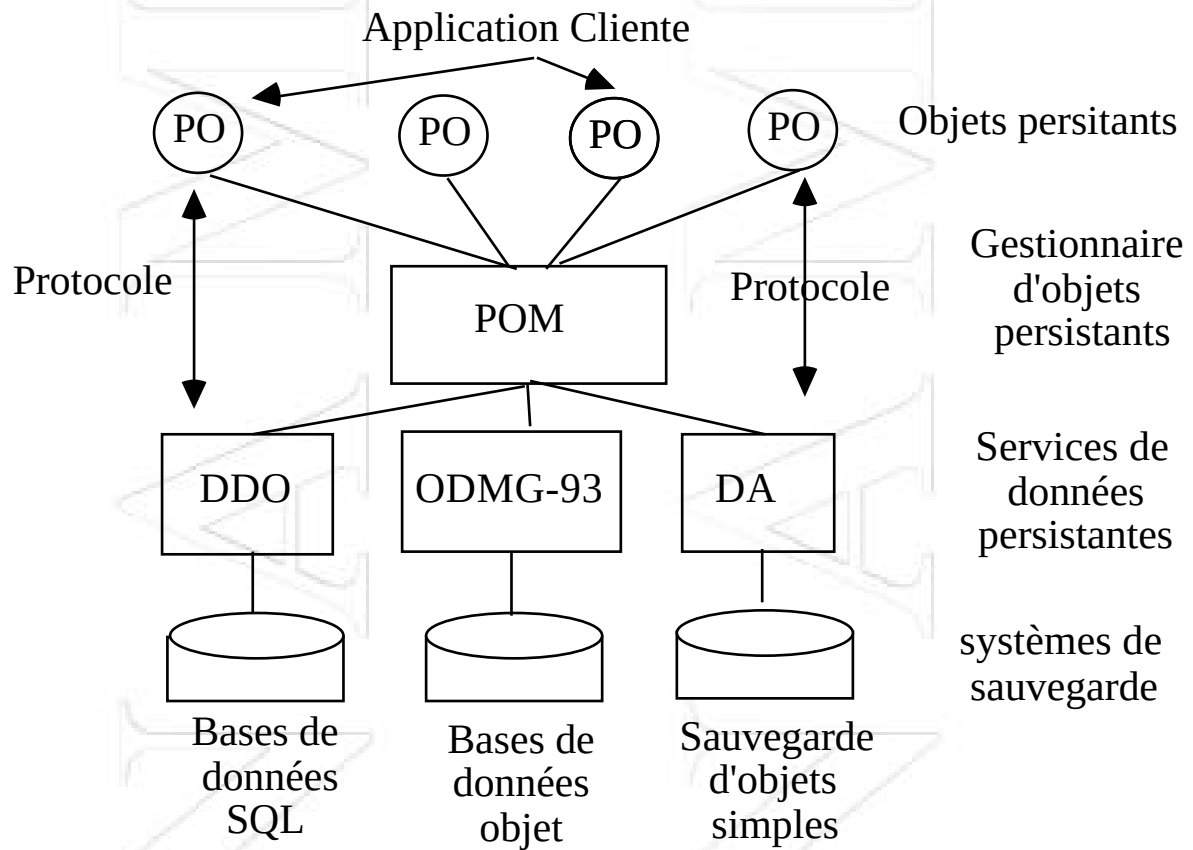
POS : La vue du client

- **POS offre aux clients le moyen de contrôler la persistance des objets** (s'ils le souhaitent)
- Il existe plusieurs niveaux de services :
 - **système de persistance complètement invisible** pour l'utilisateur
 - **Utilisation de protocoles spécifiques de sauvegarde**
- neuf opérations sur trois interfaces pour que le client puisse contrôler la persistance

POS : La vue de l'objet persistant

- **L'objet persistant a en charge sa persistance.**
- Il décide **quel protocole utiliser et quelle visibilité accorder aux clients.**
- L'objet persistant peut aussi **déléguer la gestion de sa persistance aux services sous-jacents de persistance.**
- Le grain de contrôle de persistance peut être très fin.
- L'objet persistant **hérite, si besoin, des fonctions nécessaires à la gestion de la persistance**
- Au minimum il collabore avec les systèmes de sauvegarde pour traduire son état en éléments que les services sous-jacents peuvent gérer.

Les éléments du POS



-
- **Les objets persistants (POs) :**
 - un objet peut-être **persistant par héritage de la classe PO.**
 - il peut aussi être persistant par héritage d'un mécanisme pour rendre externe son état (via un protocole)

Tout objet persistant a un identificateur persistant (PID) qui décrit la localisation de l'objet (sur le système de sauvegarde)
 - **Le gestionnaire d'objets persistants (POM)**
 - C'est une interface **indépendante de l'implémentation pour la persistance d'objets**
 - le POM peut diriger les appels des PO vers le service approprié de persistance en examinant le PID de l'objet.
 - Il fournit **une vue uniforme de la persistance** au système à travers de multiples services de données
 - **Les services de persistance de données (PDSs)**
 - Ce sont **les interfaces des implémentations particulières** des systèmes de sauvegarde.
 - Ils effectuent **le travail de déplacement des objets entre l'objet et le système de sauvegarde**
 - Tous ont une interface spécifiée en IDL.
 - certains utilisent un protocole dépendant de l'implémentation
 - **Système de sauvegarde**

ce sont les implémentations qui sauvent les objets persistants indépendamment de l'espace d'adressage contenu dans l'objet.

Da : Attributs directs

ODMG : Groupe de gestion des bases de données objet

DDO : Objet de données dynamiques (bases de données relationnelles)

Des implantations CORBA

- DEC ACA (AIX, VMS, Ultrix, MS Windows, Mac, Unix, HP-UX)
- HP ORB Plus & HP Distributed Smalltalk
- HyperDesk HD-DOMS (Sun, HP-UX, AIX,...)
- IBM SOM
- IONA Orbix (SunOS, Solaris, DOS, Windows, SGI)
- SuiteSoftware SuiteDome (VMS, Unix, PC)
- ExperSoft Xshell
- Sun DOE
- Tivoli
- Others (pas complètement Corba) : Electra, Isis, Cool, Oode,....

Il en sort 2 à 3 par mois....

Une source :

http://cui_www.unige.ch/Chloe.FAQ/OO-FAQ/oo-faq-toc.html#TOC

Laurence Duchien

Bibliographie

Corba 1.2 Specification

<http://www.acl.lanl.gov/sunrise/Objects/OMG/93-12-43.ps>

INS bindings :

<http://www.acl.lanl.gov/sunrise/Objects/OMG/93-11-6.ps>

CORBA 2.0

<http://www.acl.lanl.gov/sunrise/Objects/OMG/93-11-6.ps>

Livre :

T. Mowbray, R.Zahavi, " The Essential Corba" OMG , Wiley, 1995

R. Orfali, D. Harkey, J. Edwards, "The essential Distributed Objects, Survival Guide", Wiley, 1996.

Laurence Duchien