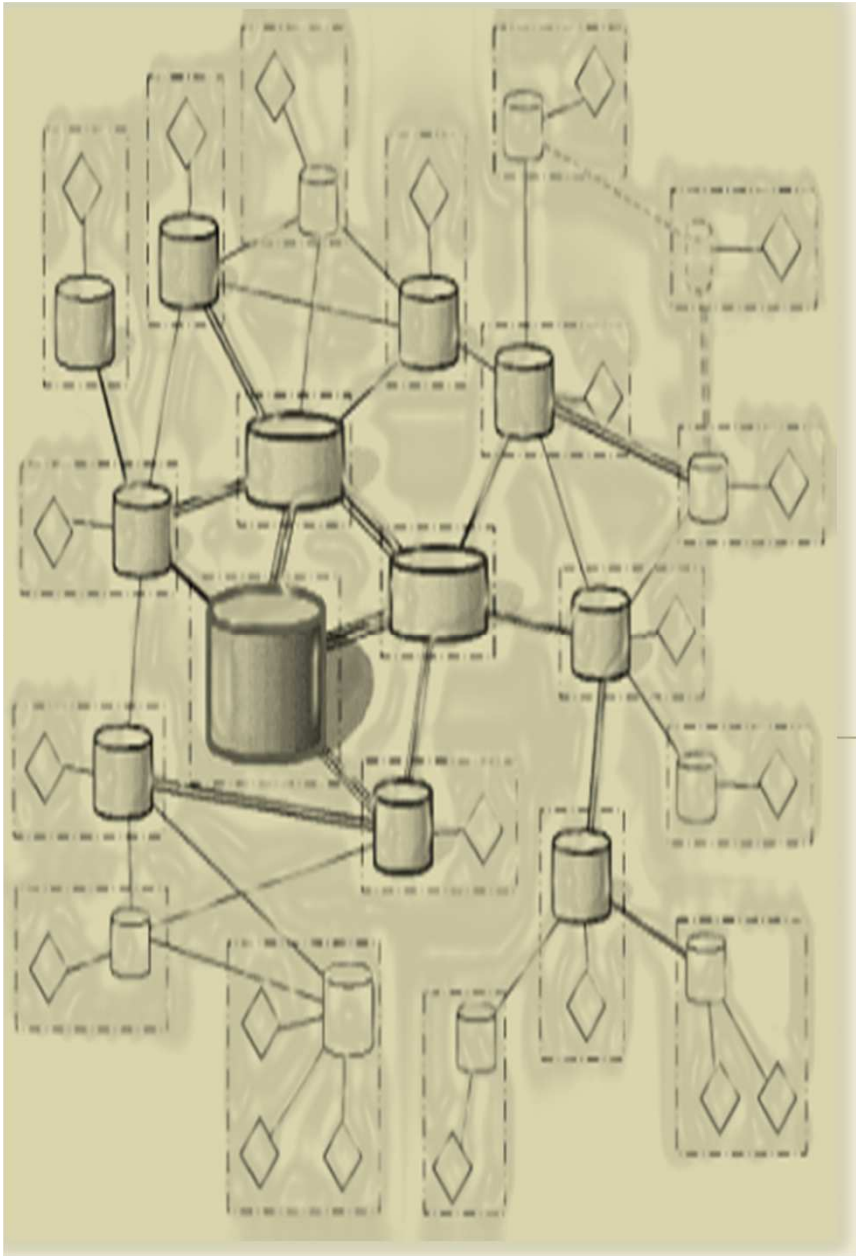


DÉVELOPPEMENT D'APPLICATIONS RÉPARTIES

Chapitre 5

CORBA

(Common Object Request Broker Architecture)

©Amen Ben Hadj Ali

amenbha@hotmail.com

ISI-L3SIL

2011-2012

Plan

- *Architecture CORBA*
- *Le langage IDL*
- *CORBA en Java : JavaIDL*
- *Le service de nommage*
- *CORBA en C++ : Mico*

Architecture

Introduction

- *Le problème : Intégration des applications*
- *Pas de consensus sur les langages de programmation*
- *Pas de consensus sur les plate-formes de développement*
- *Pas de consensus sur les systèmes d'exploitation*
- *Pas de consensus sur les protocoles réseau*
- *Pas de consensus sur les formats des données manipulées par les applications*

→ *Consensus pour l'interopérabilité*

CORBA

Common Object Request Broker Architecture

- standard ouvert pour les applications réparties
- bus logiciel orienté objet "orb"
- communication par appel de méthode à distance
- géré par l'"Object Management Group" (OMG).
- indépendant
  - du matériel, du système, des langages
  - Mais aussi des vendeurs

→ interopérabilité

Object Management Group (OMG)

- Crée en 1989
- But non lucratif
- Plus de 850 membres (Sun, IBM, Microsoft, ...)
- Crée et maintient les spécifications
  - CORBA
  - UML
- http://www.omg.org

Objectif de l'OMG

- *Promouvoir la technologie orientée objet dans les systèmes informatiques distribués*
- *Fournir une architecture de base pour l'intégration d'applications distribuées tout en garantissant la réutilisabilité, l'intéropérabilité et la portabilité.*

OBJECTIF

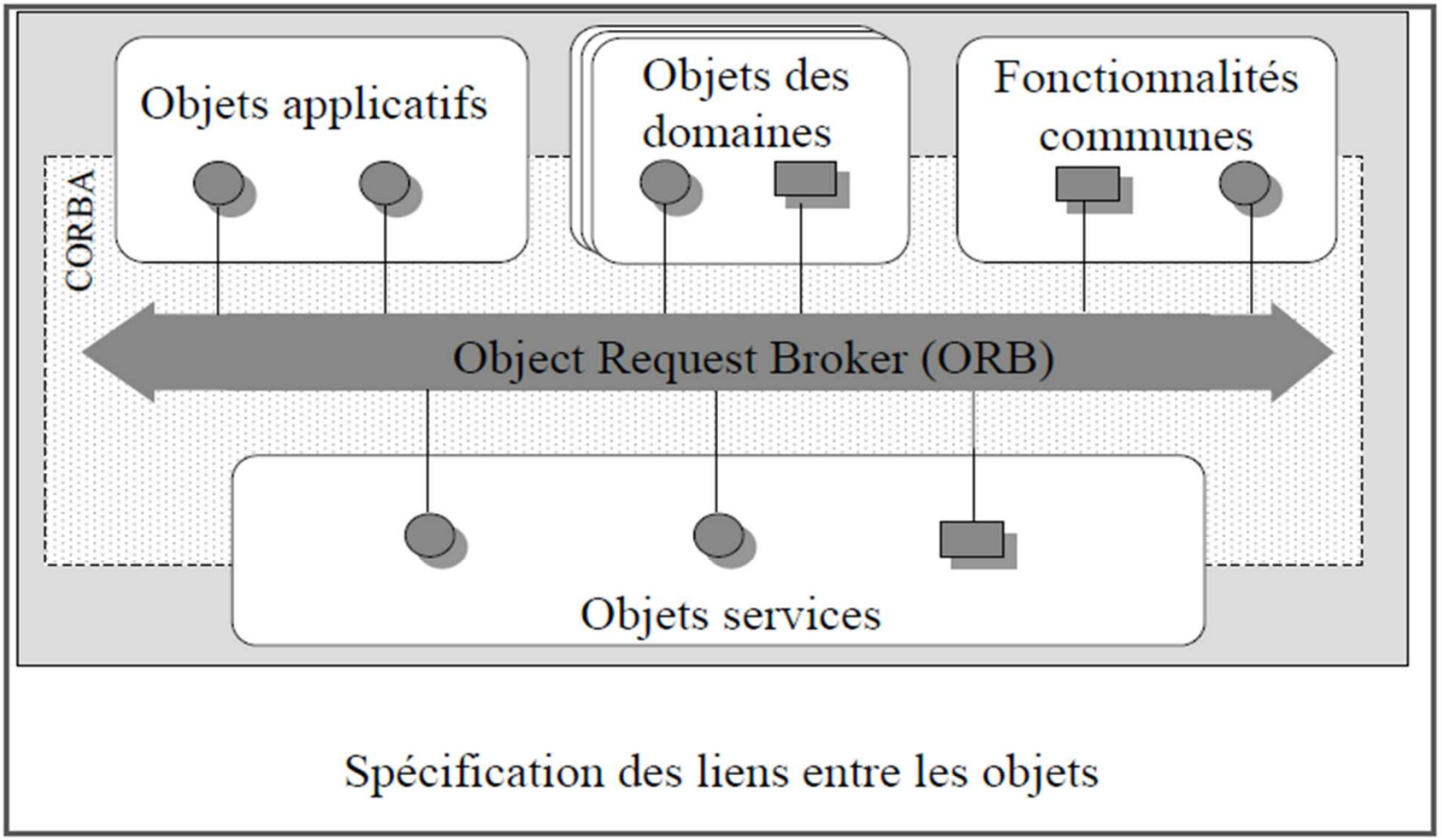
- *Favoriser l'intéropérabilité et la portabilité d'applications réparties à travers :*
- *une terminologie unique dans le domaine de l'objet;*
- *un model de référence commun;*
- *des interfaces et des protocoles communes.*

Les spécifications de l'OMG

- L'OMG spécifie tous les constituants d'un modèle objet global appelé O.M.A. (Object Model Architecture)

  - CORBA est une partie de ce modèle,
  - Utilitaires communs (services),
  - Eléments spécifiques à des corps de métier (objets de domaines).

Architecture du modèle de référence (OMA)



1.CorbaServices : Les services objets

- *Un ensemble d'outils destinés au développeur d'application distribuée.*
- *But : Centraliser des sous-ensembles répétitifs de modules retrouvés dans la plupart des développements.*
  - *Ex : un développeur peut utiliser le service générique de sécurité « security » afin de fiabilisé sans développement sans redévelopper des modules spécifiques.*

1. CorbaServices : Les services objets

Service	Description
Nommage	Recherche d'un objet par son nom (pages blanches)
Evénement	Gestion d'événements en mode serveur /Client- push ou pull
Cycle de vie	Créé, copie, déplace, détruit des objets
Persistance	Stocke et restaure l'état des objets (support persistant)
Relations	Gestion d'association entre objets
Externalisation	Placement des objets dans un flux – internalisation
Transaction	Permet des échanges transactionnels entre objets
Concurrence	Gestion de verrous pour les accès concurrents
Licence	Contrôle l'utilisation des objets
Interrogation	Envoi de requêtes vers les objets
Collection	Gestion de collection d'objets (itérateur)
Propriétés	Ajout dynamique de propriété à des objets
Sécurité	Authentification, contrôle d'accès, chiffrement, ...
Time	Synchronisations d'horloge lancement d'action simultanées
Vendeur	Recherche d'un objet par ses propriétés

2. CORBAFacilities : utilitaires

- *À la différence des Services qui s'adressent aux objets, les facilités s'adressent aux applications.*
- *Exemple : gestionnaires d'impression, des modules de messageries, des outils de gestion documentaires, …*

3. Interface de domaine : Domain interface

- *Ensemble d'objets et de composants metiers spécialisés dans un domaine particulier:*

  exemple : un objet de gestion des cartes bancaire pour les banques, un objet de gestion des dossiers de patient pour le domaine hospitaliers,…

4. Objets applicatifs :

- Ce sont les objets des applications qui vont être crées par les développeurs.
- Ils vont bénéficier des différents services, utilitaires et interfaces offerts par le OMA

5.L'ORB (Object Request Broker)

- ORB (bus logiciel) est au cœur de l'architecture OMA.
- L'ORB est responsable de la mise en relation et de la communication entre tous les éléments présents dans cette architecture distribuée (OMA)
- L'orb rend la communication entre le client et le serveur transparente.

Le modèle abstrait d'objets

- Modèle définissant la façon de décrire des objets distribués dans des environnements hétérogènes.
- Utilisé dans toutes les technologies conformes à l'OMG (ex : CORBA)
- Spécifie une sémantique commune définissant le comportement externe des objets d'une manière standard (indépendante des langages et des implémentations).

Définitions

CLIENT

- *Entité capable d'émettre des requêtes vers des objets qui fournissent des services.*
- *Le client manipule des références vers des objets distants.*

Référence objet

- *Objet manipulé par le client pour invoquer des services sur un objet distant : objet implémentation.*
- *Terme utilisé: proxy*
- *Un proxy est un représentant local au client d'un objet distant.*

Objet implémentation

- *Objet situé sur le serveur qui implémente le code des méthodes*
- *des opérations définies en IDL.*

Définitions

Requête

- *Emise par un client pour demander l'exécution d'une opération*
- *sur un objet cible.*
- *La requête contient l'opération à exécuter, l'objet cible et les paramètres éventuels.*

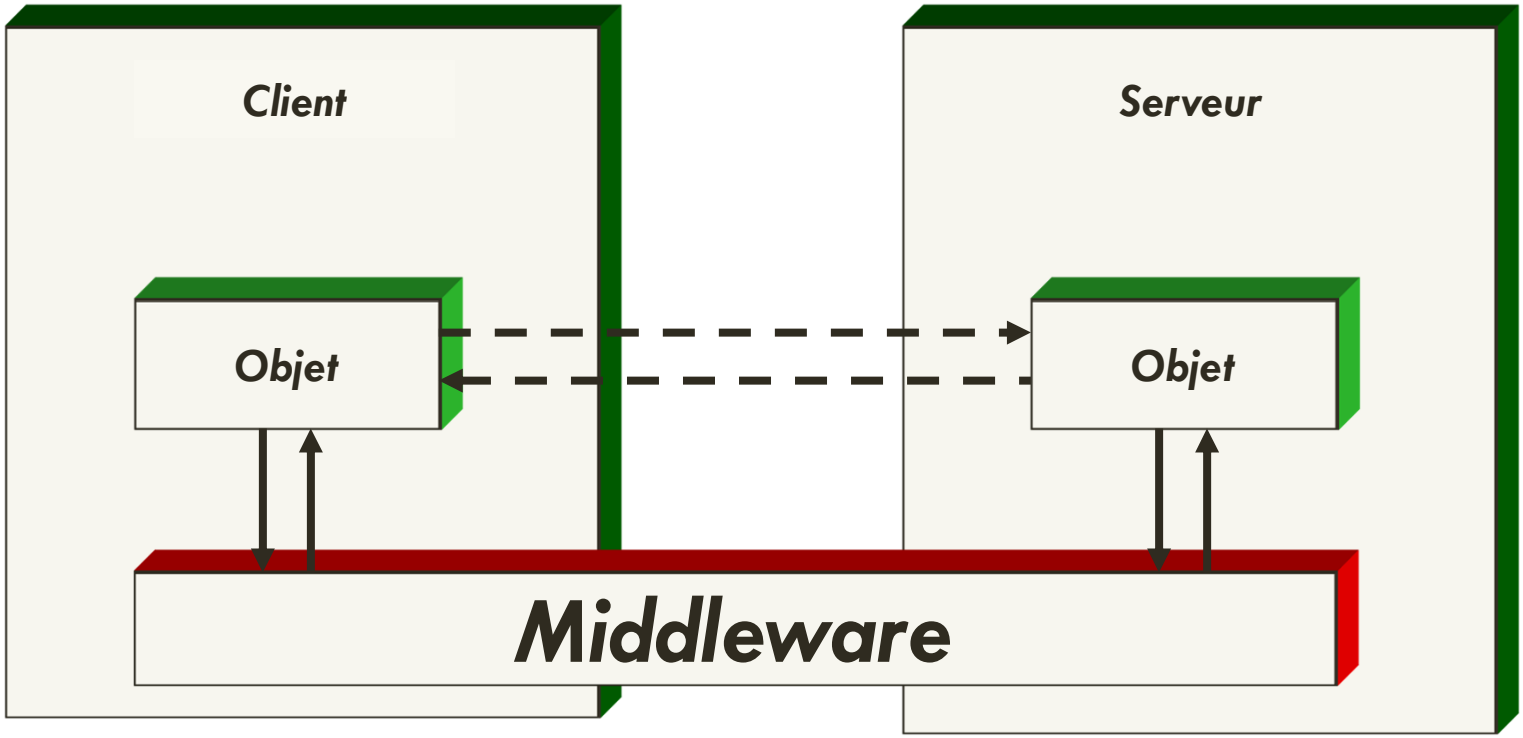
Interface

- *Description d'un ensemble d'opérations disponibles sur un objet.*
- *Spécification des interfaces en IDL.*

Opération

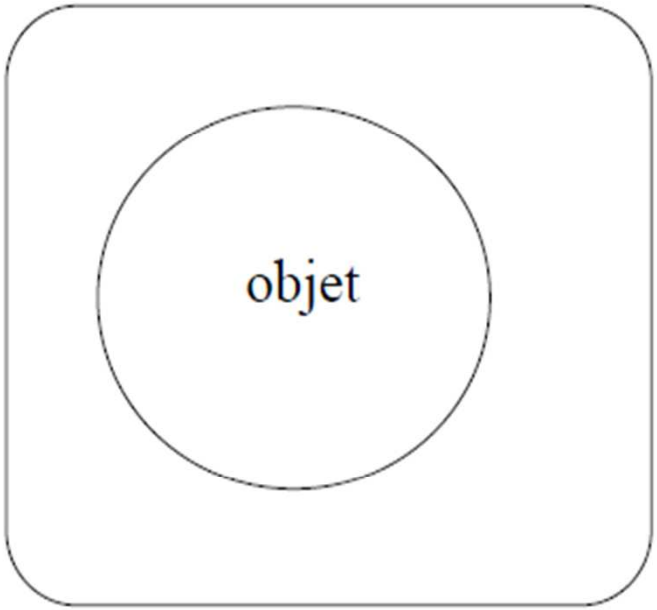
- *Entité identifiable caractérisée par une signature décrivant les paramètres de la requête et les valeurs de retour.*

Principe simplifié

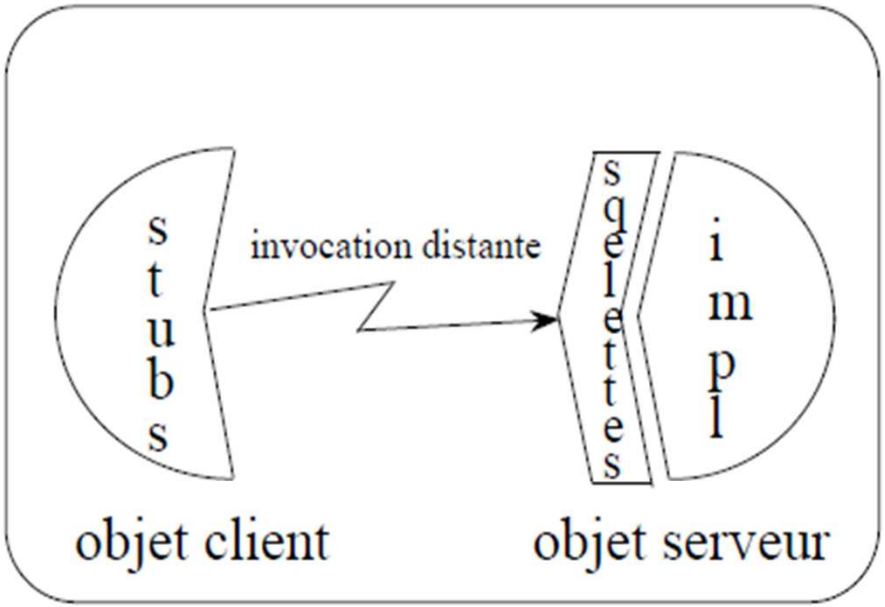


Concepts

Vue d'un objet dans une application monolithique



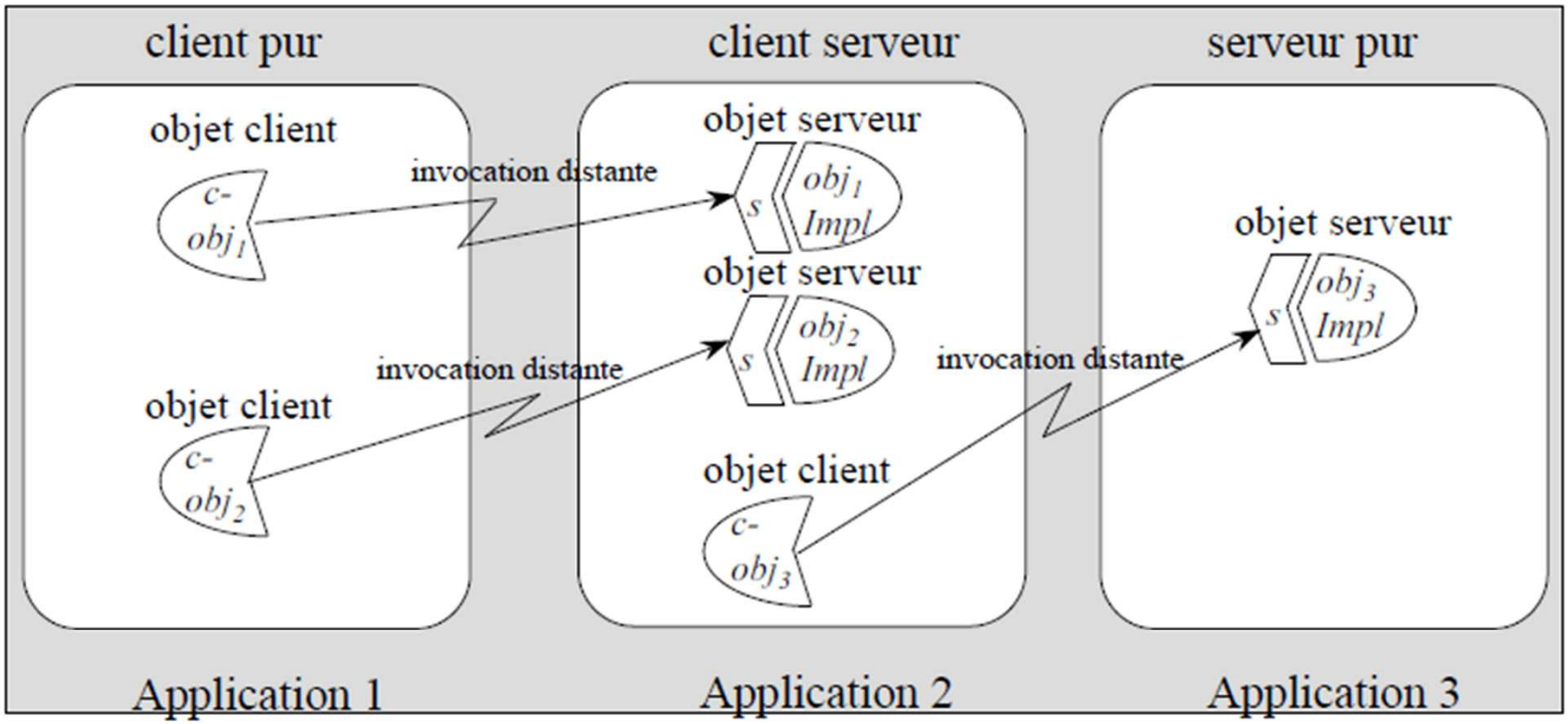
Vue d'un objet dans une application d'objets distribués



Legende

impl = objet implementation

Concepts



Legende :

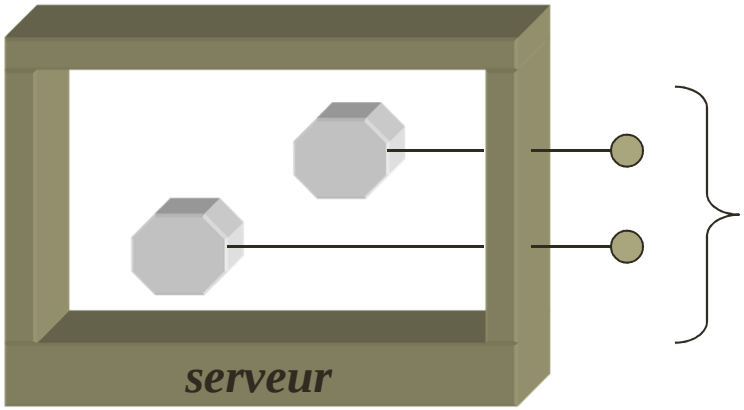
- c-obj$_i$: objet client obj$_i$
- obj$_i$ Impl : objet implementation obj$_i$
- s : squelette

Le modèle objet de CORBA

- Un serveur CORBA peut héberger plusieurs objets CORBA.
- Chaque objet est accessible indépendamment des autres objets du serveur.
- Chaque objet exprime son offre de services sous forme d'une interface IDL
  - Pour cela, on utilise un langage de description de services appelé IDL CORBA.
  - Il s'agit de décrire au sein d'une interface (vue cliente de l'objet) la liste des services offerts (ensemble de méthodes).

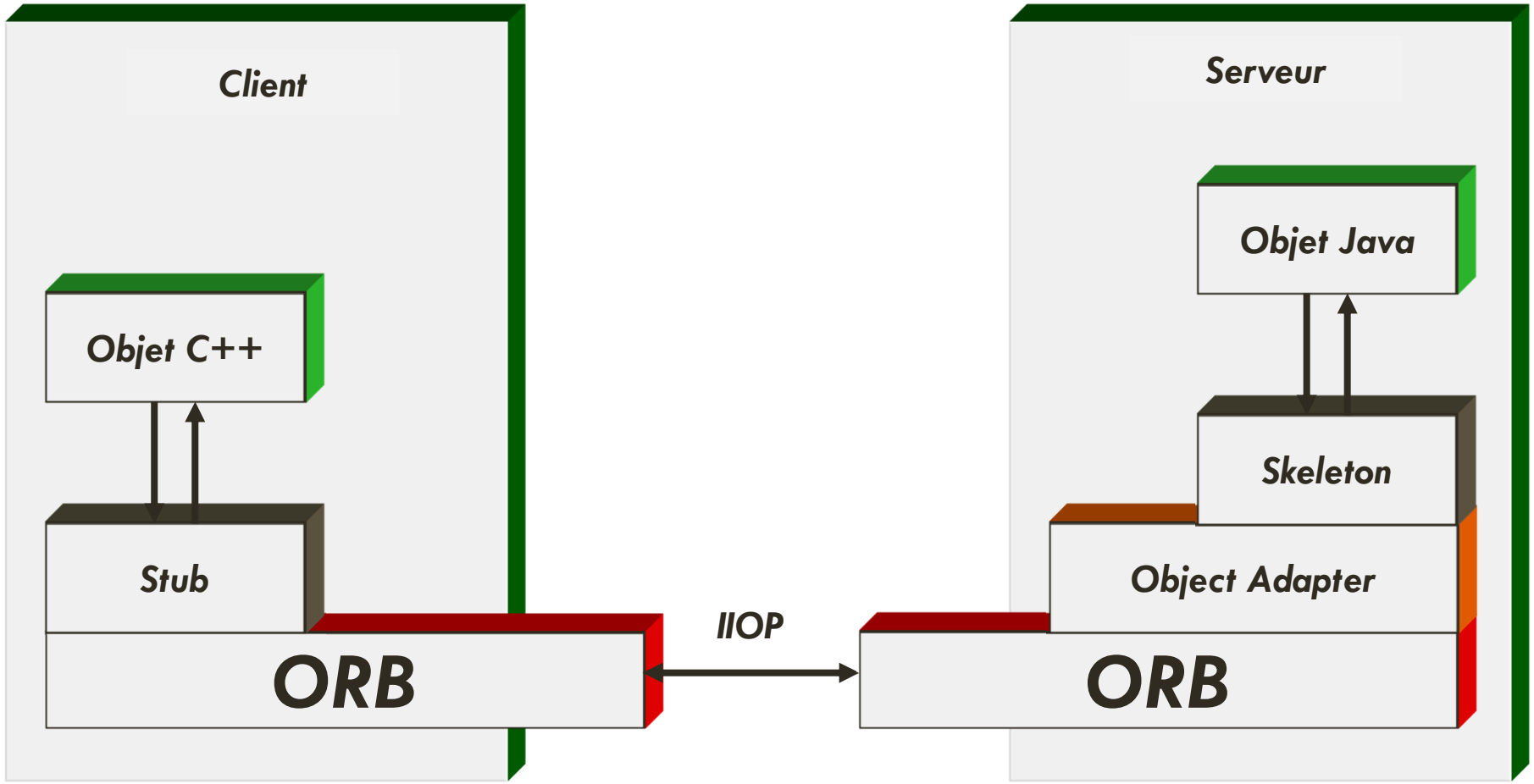
L'identité d'un objet CORBA

- Chaque objet CORBA. est associé à **une référence d'objet** qui forme son identité.
- Deux objets CORBA. du même type (exemple deux objets Horloge) ont **deux identités différentes.**



Les références d'objets sont le moyen d'accès à un objet.

Architecture de CORBA



ORB

ORB: Object Request Broker

- *Middleware qui gère les relations client/serveur entre les objets*

Rappel du concept middelware *(Courtier d'objets en français).*

- *Ensemble des logiciels nécessaires pour permettre*
- *et organiser la communication et l'échange de messages entre client et serveur.*

ORB

Composant central du standard CORBA qui gère :

- *la location d'objet*
- *la désignation des objets*
- *l'empaquetage des paramètres (marshalling)*
- *le dépaquetage des paramètres (unmarshalling)*
- *l'invocation des méthodes*
- *la gestion des exceptions*

Les communications avec CORBA

- Les participants à un échange CORBA communiquent à l'aide d'un protocole spécifique à CORBA : **IIOP** (Internet Inter-ORB Protocol).

- Le protocole IIOP est **indépendant** du langage de programmation, du système d'exploitation et de la machine utilisée.

→ ***Un client Java pourra utiliser un serveur C++***

Interface Definition Langage (IDL)

- *langage de spécification d'interfaces, supportant l'héritage multiple;*
- *Indépendant du langage d'implémentation (de tout langage de programmation ou compilateur)*
- *Indépendant de la plate-forme client*
- *Indépendant de la plate-forme serveur*
- *langage utilisé pour générer les stubs, les squelettes et pour définir les interfaces du Référentiel d'interface;*
- *la correspondance IDL langage de programmation est fournie pour les langages C, C++, Java, Smalltalk, Ada, Cobol.*
- *Ressemble beaucoup au C++*

Stub (souche)

- Code client
- Interface entre objet et ORB
- Traduit les invocations sur l'objet serveur
  - > marshalling
- Traduit les messages en valeurs de retour
  - unmarshalling

Skeleton (squelette)

- Code serveur
- Interface entre implémentation et ORB
- Traduit les invocations client vers l'implémentation
  - > unmarshalling
- Traduit la valeur de retour en message vers client
  - > marshalling

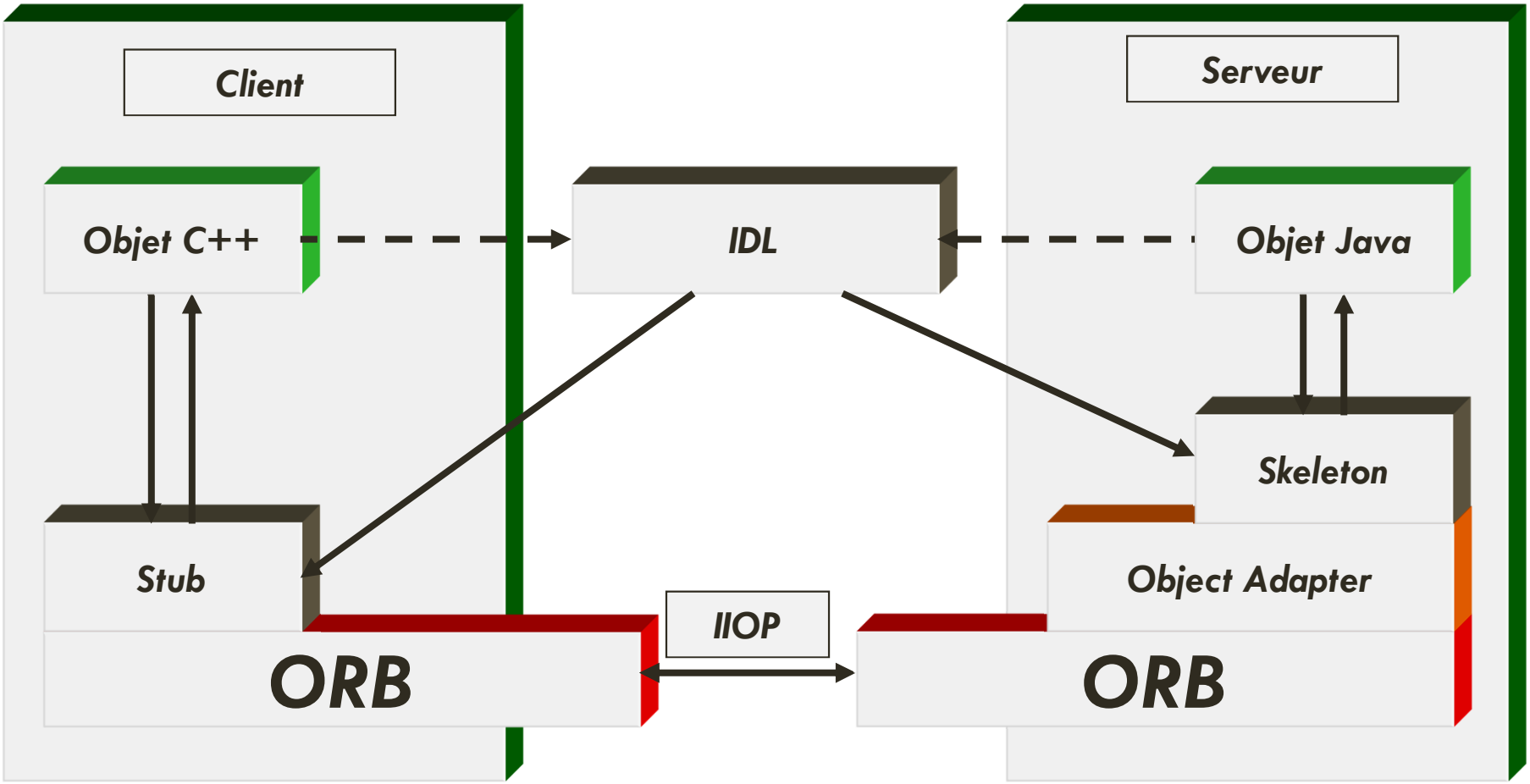
Object Request Broker (ORB)

- *Transporte les messages entre les objets*
- *Relie les stubs aux skeletons correspondants et vice-versa*
- *Bus à objets*
- *Communications inter-ORBs :*
  - *GIOP (General Inter-ORB Protocol)*
  - *IIOP (Internet Inter-ORB Protocol) (GIOP on TCP/IP)*

Object Adapter (OA)

- *Enregistre et gère les implémentations*
- *Activation et désactivation des objets*
- *Invocation des méthodes*
- *Authentification du client / contrôle d'accès*
- *Différents types :*
  - *BOA – Basic Object Adapter*
  - *POA – Portable Object Adapter*

Architecture de CORBA



Création d'une application CORBA

- 1 – Définir l'interface IDL
- 2 – Compiler l'interface IDL
- 3 – Créer l'implémentation de l'interface IDL
- 4 – Créer le serveur :
  - > publication de l'objet CORBA
- 5 – Créer le client :
  - > appel de l'objet CORBA

IDL

Interface IDL

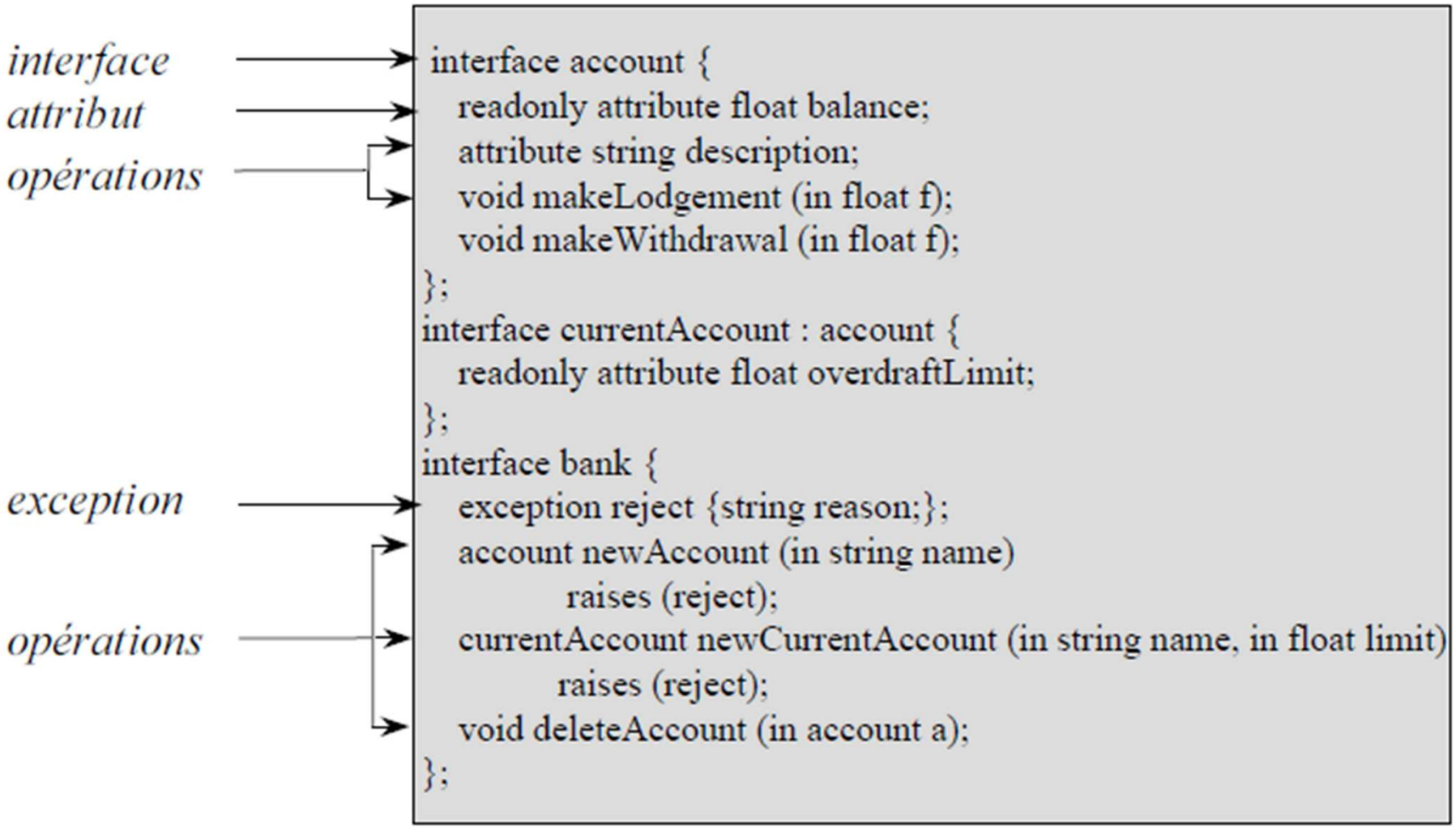
- *Une spécification IDL définit un ou plusieurs types, constantes,*
- *exceptions, interfaces, modules*
- *Un module permet de limiter la validité des identificateurs*
- *Interface : ensemble d'opérations et de types => classe C++*
- *Syntaxe*

Interface | [<héritage>] { <interface Body>};

Exemple


```
module example {
    interface monExample {
            void methode1();
            long methode2();
            void methode3(in long param, out long result);
    };
};
```

Exemple


interface
attribut
opérations
exception
opérations
interface account {
readonly attribute float balance;
attribute string description;
void makeLodgement (in float f);
void makeWithdrawal (in float f);
};
interface currentAccount : account {
readonly attribute float overdraftLimit;
};
interface bank {
exception reject {string reason;};
account newAccount (in string name)
raises (reject);
currentAccount newCurrentAccount (in string name, in float limit)
raises (reject);
void deleteAccount (in account a);
};

IDL vs C++

- Même règles lexicales que C++
- Grammaire IDL : sous ensemble de la grammaire ANSI C++ avec constructions supplémentaires
- Nouveaux mots clés :
  - ATTRIBUTE
  - INTERFACE
  - MODULE
  - ONEWAY
  - READONLY
  - SEQUENCE
  - ANY

Types primitifs

- void
- short
- long
- long long
- float
- double
- boolean

Types complexes

- *string*
- *struct*
- *enum*
- *union*
- *any*

Syntaxe

- Commentaire : comme C++ et Java
  - // : jusqu'à la fin de la ligne
  - /* … */ : bloc de commentaire
- Préprocesseurs : #define, #include, #ifdef, #endif
- Alias : typedef
- Possibilité de définir des attributs
- Constantes : const

Attribut IDL

Définition d'attribut

```
interface account {
        readonly attribute float balance;
        attribute string description;
        ...
};
```

Equivaut à :

```
float _get_balance();
string _get_description();
void _set_description(in string s);
```

Méthode (opérations)

void methode(in long param, out long result);

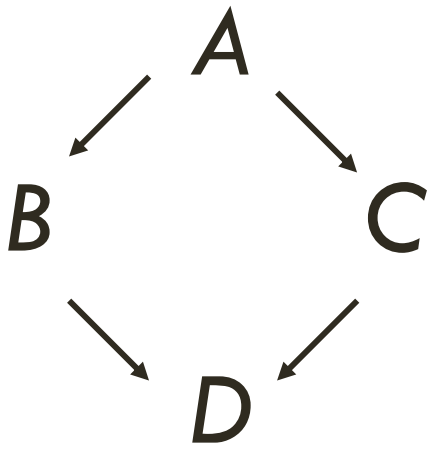
- *Paramètres nommés et associés à un mode*

- *Méthodes : comme en C++ et Java, sauf*
  - *in : paramètre utilisé en entrée (lu, non modifié)*
  - *out : paramètre utilisé en sortie (non lu, modifié)*
  - *inout : paramètre utilisé en entrée et en sortie (lu et modifié)*

Héritage

- *Héritage multiple*
- *Surcharge et redéfinition interdites*

- *interface A { …};*
- *interface B : A { … };*
- *interface C : A { … };*
- *interface D : B, C { … };*



JavaIDL

JavaIDL

- ORB CORBA en Java de Sun
- Inclus au JDK
- idlj : compilateur idl vers java
  - Syntaxe :
- orbd : ORB et serveur de nom indépendant
  - Syntaxe :

Implémentation

- Fichier MonObject.IDL :

  interface MonObjet { … };

- Classe qui implémente le contrat IDL
  - Doit dériver de MonObjetPOA
  - MonObjetPOA est généré par le compilateur IDL
- Lancement client ou serveur :
  - java <serveur/client> -ORBInitialPort <port>
    -ORBInitialHost localhost

Serveur / 1

- *Initialiser l'ORB :*
  - *Méthode statique init(args, null) de la classe ORB*
  - *retourne un objet ORB*
- *Objet CORBA :*
  - *org.omg.CORBA.Object*
  - *Ne pas confondre avec java.lang.Object*
  - *Retourné par différentes méthodes*
  - ***Cast** avec méthode statique narrow de la classe MonTypeHelper*

Serveur / 2

- *Utilisation d'un POA :*
  - *Méthode resolve_initial_references("RootPOA") de l'objet ORB*
  - *Retourne un objet CORBA à caster en objet POA*
  - *Puis méthode the_POAManager().activate() de l'objet POA*
- *Enregistrer le servant :*
  - *Méthode servant_to_reference(servant) de l'objet POA*
  - *retourne un objet CORBA*
- *Lancement de l'ORB (à la fin) :*
  - *Méthode run de l'objet ORB*

Client

- *Initialiser l'ORB :*
  - *Méthode statique init(args, null) de la classe ORB*
  - *retourne un objet ORB*
- *Récuperer un objet CORBA depuis son IOR :*
  - *Méthode string_to_object(IOR) de l'objet ORB*
  - *Retourne un objet CORBA*
  - *Cast dans son vrai type avec la méthode statique narrow de la classe MonObjetHelper*
  - *MonObjetHelper est généré par le compilateur IDL*

Service de nommage

Services CORBA

- Service de cycle de vie
- Service d'événements
- Service de concurrence
- Service de transaction
- Service de persistance
- Service d'interrogation
- Service de collection

Services de recherche d'objets

- Service de nommage (Naming)
  - Recherche d'objet par nom
  - pages blanches
- Service vendeur (Trader)
  - Recherche d'objet par propriété
  - pages jaunes

Besoin du Naming

- Référence d'un objet : IOR
  - Fichier partagé, …
- Service à la DNS :
  - Service accessible par le bus ORB
  - Service standard entre ORBs
  - Un nom spécifique <-> un objet corba
  - Gère les contextes de nom

Utilisation du Naming

- Module **CosNaming**
- Interface **NamingContext**
- Nouvelle interface **NamingContextExt**
  - Création d'une association : **bind, rebind**
  - Résoudre une association : **resolve**
  - Détruire une association : **unbind**
- Programme indépendant à lancer avant : **orbd**

Obtenir le Naming

- Naming = objet CORBA
  - Défini en IDL
  - Associé au nom « NameService »
- Racine de l'arbre de référence :
  - Méthode resolve_initial_references de l'ORB
  - Conversion en NamingContext ou NamingContextExt
- Possibilité de spécifier où chercher les initial_references

Naming dans JavaIDL : coté serveur

- Lancer orbd avant
- Récupérer le Naming :
  - orb.resolve_initial_references("NameService") retourne un objet CORBA
  - Puis cast avec NamingContextExtHelper.narrow(obj)
- Créer une association (objet NamingContextExt) :
  - to_name("NOM") retourne un cheming sous forme de NameComponent[]
  - Puis enregistrement avec rebind(chemin, obj)

Naming dans JavaIDL : coté client

- *Même initialisation/récupération :*
  - *orb.resolve_initial_references("NameService") retourne un objet CORBA*
  - *Puis cast avec NamingContextExtHelper.narrow(obj)*
- *Récupération de l'objet CORBA à partir du nom :*
  - *ns.resolve(ns.to_name("NOM")) retourne un objet CORBA*
  - *Puis cast avec MonObjetHelper.narrow(obj)*

Conclusion: Avantages de CORBA

- *CORBA présente des avantages importants pour les systèmes distribués comme :*
  - *La transparence;*
  - *La portabilité;*
  - *L'interopérabilité;*
  - *L'adaptabilité;*
  - *La disponibilité;*
  - *La stabilité.*

Conclusion: Inconvénients de CORBA

- *Il présente aussi des désavantages comme par exemple:*
  - *La complexité;*
  - *Le prix élevé;*
  - *Une formation spécialisée pour les développeurs.*