Introduction à CORBA

Julien Sopena
Julien.Sopena@lip6.fr
(basé sur un cours de **Gaël Thomas** et de **Lionel Seinturier**)

Université Pierre et Marie Curie
Master Informatique
M1 – Spécialité SAR

Introduction à CORBA

1. Caractéristiques

CORBA = Common Object Request Broker Architecture

- Bus à objets multi-OS, multi-langages
- Orienté objets
- Défini par l'Object Mangagement Group (OMG)
  - ✓ Organisme de standardisation internationale depuis 1989
    Réuni de vendeurs de matériels, de systèmes, d'applications, de conseil, des organismes de recherche…
  - ✓ Environ 1000 membres (institutions)
  - ✓ Produit des spécifications

OMA (Object Management Architecture)

- Définit une structure générale pour la gestion d'objets répartis

CORBA

- Un des composants de l'OMA
- Permet aux objets distribués de communiquer
- Actuellement CORBA 3.0

1. Caractéristiques

Corba : nombreuses implantations

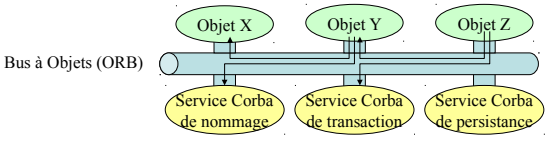
- Commerciales
  - ORBIX — IONA — www.iona.com
  - VisiBroker — Borland — www.borland.com/bevisibroker
  - ORBacus — OOC — www.orbacus.com
- Logiciels libres (et/ou gratuits)
  - JDK — Sun — www.java.sun.com
  - MICO — Univ. Francfort — www.mico.org
  - JacORB — Univ. Berlin — www.jacorb.org
  - TAO — Univ. Washington — www.theaceorb.com
- Plus de 40 implantations recensées

1. Caractéristiques

CORBA = environnement de programmation et d'exécution réparti

- Un appel de procédure multi-OS, multi-langages
- Des services supplémentaires (15) utilisables par l'application
  Service de résolution de noms, de transaction, de persistance…
- Des fonctionnalités additionnelles par rapport à un simple RPC

Correspond à un bus à objets



1. Caractéristiques

Principes de la spécification CORBA

- Transparence de la localisation
- Transparence de l'accès
- Séparation des interfaces et des implantations
- Interfaces typées
- Support de l'héritage multiple d'interfaces

⇒

- Utilisation d'un objet indépendamment de sa localisation
- Services accédés en invoquant des méthodes sur des objets locaux
- Client dépendant de l'interface, pas de l'implantation
- Références d'objets typées par des interfaces
- Évolution de service facilitées

1. Caractéristiques

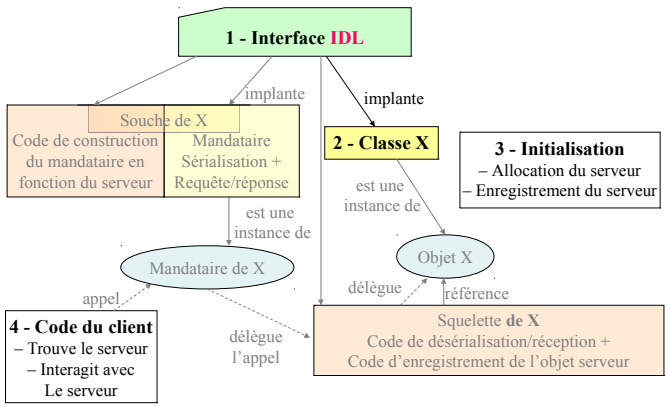
Construction d'une application CORBA

1. Écriture d'une interface IDL (Interface definition language)
2. Écriture d'une classe implantant l'interface IDL
3. Écriture d'un programme serveur
4. Écriture du programme client

⇔

1. Déclaration des services accessibles à distance
2. Définition du code du service
3. Instanciation et enregistrement du serveur
4. Recherche et interaction du serveur

1. Caractéristiques



2. Le langage IDL

CORBA IDL (Interface Definition Language) = Contrat entre le client et le serveur

Langage de définition des services proposés par un objet serveur CORBA
- Définit les méthodes qu'un client peut invoquer sur le serveur
- Définit les champs accessibles à distance du serveur (get/set)

Génération des souches et squelettes à partir de l'interface IDL
- **Génération pour plusieurs langages** (Java, C++…, C…)
  Si le langage est non objet, la projection respecte une sémantique objet
- Le client et le serveur ne sont pas forcement écrits dans le même langage!

L'interface IDL suffit à caractériser totalement les fonctionnalités d'un serveur

2. Le langage IDL

Trois éléments hiérarchiques principaux
- Module : espace de définition possédant un nom
  Hiérarchie de modules (peuvent être imbriqués)
- Interface : regroupement de services
  Définition de l'objet serveur accessible à distance
- Méthode : définition des fonctionnalités du serveur

Éléments Secondaires d'un fichier IDL
Types complexes (structures et unions), constantes, exceptions, attributs d'interface

Exemple :
```
module banque {
 interface Compte {
  public float solde(); } }
```


2. Le langage IDL

Syntaxe IDL : proche de la syntaxe C/C++
- Commentaires : /* */, //
- Identificateurs : [alpha][alpha,0-9,_]*, casse non déterminante (mais doit être la même dans un fichier)
- Identificateur pour chaque paramètre de méthode, pas de (void)
- Signed et unsigned pour les primitifs, sauf caractères
- Préprocesseur (#define, #ifdef/#ifndef, #include, #pragma)

Mots clés IDL :

any	enum	Object	struct
attribute	exception	octet	switch
boolean	FALSE	oneway	TRUE
case	float	out	typedef
char	in	raises	union
const	inout	readonly	unsigned
context	interface	sequence	void
default	long	short	wchar

2. Le langage IDL

Types simples normalisés et projetés pour chaque langage/OS

Booléan :
boolean (2 valeurs : TRUE et FALSE)
Octet :
Octet (1 octet, signé ou non signé, dépend du langage)
Nombres signés :
short (2 octets), long (4 octets), long long (8 octets)
Nombres non signés :
unsigned short (2 cotets), unsigned long (4 octets), unsigned long long (8 octets)
(projeté vers l'équivalent signé en Java!)
Nombres à virgule flottante :
float (4 octets), double (8 octets), long double (16 octets, pas de projection vers Java!))
Caractère et chaîne de caractères :
char (1 octet, ISO Latin1), string (chaîne de char de taille), string<n> (à taille fixe)
wchar (2 octets, unicode), wstring (chaîne de wchar),

2. Le langage IDL

Types construits : structures, énumération, définition, tableaux
- Structure : regroupement d'attributs (⇔ struct C ou tuple SQL)
  struct *identificateur* { liste d'attributs typés; };
  struct Personne { string nom; long age; };
- Énumération : liste de valeur pour un type (⇔ enum C)
  enum *identificateur* { liste de valeurs, };
  enum couleur { rouge, vert, bleu };
- Définition de type : nom symbolique (⇔ typedef C)
  typedef type *identificateur*
  typedef string<16> tprenom;
- Tableau mutli-dimensionnel de taille fixe :
  type *identificateur* [taille]+;
  long matrice[32][16];
- Tableau uni-dimensionnel de taille quelconque :
  sequence<type> *identificateur*; ou **sequence**<type, max> *identificateur*;
  sequence<long> vecteur; ou **sequence**<long, 16> vecteur;

2. Le langage IDL

- Union : type variable en fonction d'un discriminant (de base ou énuméré)
  union *identificateur* **switch**(*enumIdentificateur* | *typeBase*) {
  case valeur : type *identificateur*; +
  default: type *identificateur*; [0,1]
  };
  union Foo **switch**(long) { **case** 0 : float toto; **default**: double tata; }
  union Bar **switch**(couleur) {
  case rouge : long r; **case** vert : boolean v; **case** bleu : long long b; };

Autres types primitifs :
- Type Object Corba : tout objet Corba (passage d'un objet par référence)
  void f(in **Object** o); ou struct bip { **Object** x; };
- Type indifférencié : n'importe quel type (de base ou construit)
  void f(in **any** o); ou struct bip { **any** x; };
- Type vide : spécifie qu'une méthode ne renvoie rien
  void f(in Foo f);

2. Le langage IDL

Modules : espaces de définition de symboles
- Types, constantes, exceptions, modules, interface
- Hiérarchie de modules pour structurer les applications
- Opérateur de résolution de portée : "::"
- Visibilité respecte l'inclusion

```
module pim {
 typedef wstring<32> tadresse;
 module pam {
  typedef wtring<16> tnom;
 };
 module poum {
  struct pom {
   pam::tnom nom;          /* ou pim::pam:tnom */
   tadresse    adresse; }; }; };
```


2. Le langage IDL

Interfaces : points d'accès aux objets CORBA
- Contiennent types, constantes, exceptions, attributs, méthodes
- Peuvent hériter (multiple) d'autres interfaces (interface X : Y, Z)
- Contenues dans un module
- Autorise les pré-définition (interface X; interface Y {uses X}; interface X {uses Y})

```
module pim {
 struct Person {  // équivalent à la déclaration d'interface Serializable en RMI
  wstring name;
  wstring adress;
 };

 interface Pom { // equivalent à la déclaration d'interface Remote en RMI
   Person find(in wtring name);
 };
};
```

2. Le langage IDL

Méthode : un des services proposé par un objet CORBA

- Sont nommées par un identificateur
- Peuvent recevoir des arguments et en renvoyer
- Peuvent lever une (ou plusieurs) exceptions

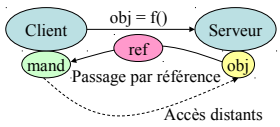
typeRetour identificateur([paramètres]*) [raises [exceptions]+];
Paramètre = mode type identificateur
Mode : in en entrée, out en sortie, inout en entrée et sortie
Type : tout type de base ou construit avec un typedef
void f(in sequence<string> arg); // interdit
typdef sequence<string> tsa; void f(in tsa arg); // autorisé
Exemple : void f(**in** Person arg);

- Surcharge interdite (pas deux méthodes ayant le même nom)

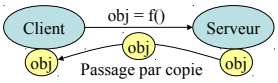
2. Le langage IDL

Passage par référence ou par valeur

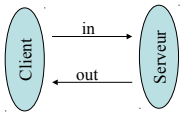
- Par référence si
  - ✓ Object CORBA (interface et Object)



- Passage par copie si
  - ✓ Types simples (float, long, double…)
  - ✓ Types construit (struct, sequence…)



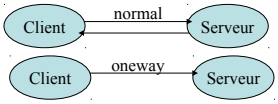
- Paramètre in : le client fournit la valeur
  - ✓ Si le serveur la modifie, le client n'est pas mis à jour
- Paramètre inout : le client fournit la valeur
  - ✓ Si le serveur la modifie, le client est mis à jour
- Paramètre out : le serveur fournit la valeur
  - ✓ Le client est mis à jour



2. Le langage IDL

Méthode oneway : pas d'attente du retour

- Appel normal : attend le retour
  Exécution de l'appel garanti
- Appel oneway : pas d'attente
  Exécution de l'appel non garanti
  Utilisable uniquement pour méthodes
  - ✓ Sans paramètre de retour (void)
  - ✓ Sans paramètre d'entrée inout ou out




```
module banque {
 interface Compte {
  oneway void setTitulaire(in Person p) raises PlusDeSous;
  Person getTitulaire(); // oneway impossible car renvoie une structure  Person
 }; };   // dans cet exemple, Person est un struct => passé par copie
```


2. Le langage IDL

Constantes : variable typées dont la valeur est fixe
const type *identificateur* = valeur;
Exemple **: const** long *taille* = 16*2; typedef wtring<taille> tnom;

- Uniquement type de base
- Opérateurs autorisés : +, *, /, -, %, &, |, ^, <<, >>

Exception : structure qui signale une situation exceptionnelle
exception *Identificateur* { [attributs typés]* };
Exemple : **exception** *Overflow* { double limite; };
interface Math { double exp(in double x) raises(Overflow); }

Attributs : propriétés typées attachées aux interfaces
Propriétés variables, accessible avec des méthodes **getter/setter**
[readonly] **attribute** type identificateur;
Exemple : interface X { attribute float lim; readonly attribute float maxLim; };

2. Le langage IDL

Héritage d'interface
Simple : interface X : Y { … };
Multiple : interface X : Y, Z { … };

Une sous-interface hérite de tous les éléments de ses super-interfaces

- Peut ajouter de nouveaux éléments
- Peut redéfinir les types, constantes, exceptions
- Ne peut pas redéfinir les attributs et les opérations!

Graphe d'héritage

- Possibilité d'héritage en losange (B ← A, C ← A, D ← B,C)
- Cycle dans le graphe d'héritage interdit (A ← B, B ← A interdit)

2. Le langage IDL

Utilisation des éléments des super-interfaces ne doit pas être ambiguë

```
interface A { typedef wstring<16> tnom; };
interface B { typedef wstring<32> tnom; };
interface C { const A::tnom n1 = "Bob";
              const B::tnom n2 = "Bill"; /* pas de const tnom n3 = "Jules"; */ };
```

Remplacement des constantes : dès l'utilisation

```
interface A {
 const long taille = 16;
 typedef float coord[taille];
 void f(in coord c);
};
interface B { const long taille = 712; } ;
interface C : B, A {}; // coord a une taille de 16
```


3. Projection vers Java

Implantation d'un service

- Projection de la description IDL vers un langage cible
  6 langages supportés par l'OMG (C++, Java, Ada, Cobol, Smalltalk, C)
  Nombreuses autres projections existent (Tcl, Perl, Clos, Eiffel, Python, VB, Modula…)
- Règles de traduction définies pour chaque langage
  Génération des souches et des squelettes
- Un outil de traduction par langage
  Par exemple, idlj –fall traduit des interfaces IDL en squelettes/souches Java

3. Projection vers Java

Traduction IDL → Java : les types primitifs

IDL	*Java*	*IDL*	*Java*
octet	byte		
short	short	unsigned short	short
long	int	unsigned long	int
long long	long	unsigned long long	long
float	float	char, wchar	char
double	double	string, wstring	String
long double	pas de correspondance		

3. Projection vers Java

Traduction IDL → Java : les structures (paramètres passé par copie)

- Foo.java : une classe Java contenant les champs de la structure
  - ✓ Hérite de IDLEntity
  - ✓ Deux constructeurs (vide et avec un paramètre par champs de la structure)
- FooHelper.java : classe de gestion des objets Foo (voir + loin)
- FooHolder.java : classe utilitaire pour la gestion des out/inout (voir + loin)

```
module pim {
  struct Person {
    string nom;
    float age;
  };
};
```

Traduction →

```
package pim;
public class Person implements IDLEntity{
  public String nom;
  public float age;

  Person() {}
  Person(String nom, float age) {
    this.nom = nom;  this.age = age;
  }
}
```


3. Projection vers Java

Traduction IDL → Java : les exceptions

- Foo.java : une classe Java contenant les champs de l'exception
  - ✓ Hérite de UserException (qui hérite de Exception et IDLEntity)
  - ✓ Deux constructeurs (vide et avec un paramètre par champs de l'exception)
- FooHelper.java : classe de gestion des objets Foo (voir + loin)
- FooHolder.java : classe utilitaire pour la gestion des out/inout (voir + loin)

```
module pim {
  exception Overflow {
    long limit;
  };
};
```

Traduction →

```
package pim;
public class Overflow extends UserException {
  public int limit;

  Overflow() { super(OverflowHelper.id()); }
  Overflow(int limit) {
    super(OverflowHelper.id());
    this.limit = limit;
  }
}
```


3. Projection vers Java

Traduction IDL → Java : les énumérations

- Foo.java : une classe Java définissant l'énumération
  - ✓ Hérite de IDLEntity
  - ✓ Un champs int associé à chaque valeur de l'énumération
- FooHelper.java : classe de gestion des objets Foo (voir + loin)
- FooHolder.java : classe utilitaire pour la gestion des out/inout (voir + loin)

```
module pim {
  enum Couleur {
    rouge, vert, bleu
  };
};
```

Traduction →

```
package pim;
public class Couleur implements IDLEntity {
  private int __value;
  private Couleur[] __array = new Couleur[3];

  public static final int _rouge = 0;
  public static final Couleur rouge = new Couleur(_rouge);
  …
  protected Couleur(int v) { __value = v; __array[v] = this; }
  public int value() { return __value; }
  public Couleur from_int(int v) { return __array[v]; } }
```


3. Projection vers Java

Traduction IDL → Java : les unions

- Foo.java : une classe Java définissant l'union
  - ✓ Hérite de IDLEntity
  - ✓ Possède un discriminant associé au type en cours + un champs par type
- FooHelper.java : classe de gestion des objets Foo (voir + loin)
- FooHolder.java : classe utilitaire pour la gestion des out/inout (voir + loin)

```
module pim {
  union UC switch(Couleur) {
    case rouge : float f;
    case vert : double v;
    case bleu : long b;
  };
};
```

Traduction →

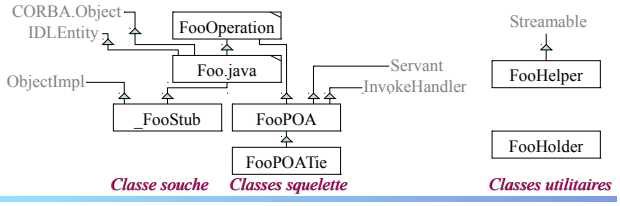
```
package pim;
public class UC implements IDLEntity {
  private float __f;   private double __d;   private long __l;
  private Couleur discriminator;

  public float r() {
    if(descriminator != Couleur.rouge) throw …
    return __r;
  }
  public void r(float value) {
    discriminator = Couleur.rouge;    __r = value; } }
```


3. Projection vers Java

Traduction IDL → Java : les interfaces

- FooOperation.java : interface Java traduisant l'interface IDL
- Foo.java : interface de l'objet Corba traduisant l'interface IDL
- _FooStub.java : souche cliente
- FooPOA.java : squelette du serveur (implantation par héritage)
- FooPOATie.java : squelette du serveur (implantation par délégation)
- FooHelper.java : classe de gestion des objets implantant Foo (voir + loin)
- FooHolder.java : classe utilitaire pour la gestion des out/inout (voir + loin)



3. Projection vers Java

Traduction IDL → Java : les paramètres des méthodes

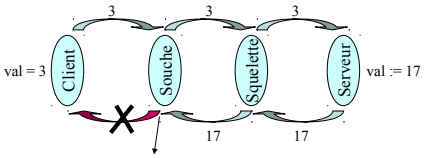
- Problème : les paramètres out et inout doivent être modifié chez le client

  interface X { void f(inout long val); }

  val doit être mis à jour chez l'appelant après l'appel de f
- Impossible à faire de manière transparente

  Java : void caller(X x) { int val = 3; x.f(val); /* ici, val = 3 */ }

  val est copié lors de l'appel : sa valeur n'est donc pas modifiée



La souche ne connaît que la valeur 3, elle n'a pas de référence vers val
Et ne peut donc pas mettre la variable à jour

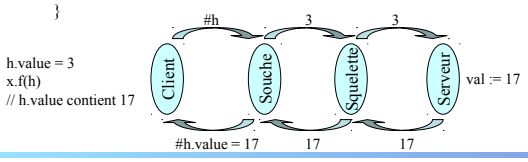
3. Projection vers Java

Traduction IDL → Java : les paramètres des méthodes

- Utilisation de Holder (conteneurs)

  Classe encapsulant une autre classe
  - Un champs publique du type contenu
  - Un constructeur avec un paramètre du type contenu
  - Une par type primitif (IntHolder, StringHolder…)
  - Une par type complexe généré lors de la traduction (struct, enum, interface, …)

```
void caller(X x) {
  IntHolder h = new IntHolder(3);
  x.f(h);      System.out.println("valeur: " + h.value); // nouvelle valeur
}
```



3. Projection vers Java

Traduction IDL → Java : les interface

- Attribut IDL ⇒ méthode getter et setter (si !readonly)

```
interface X {
  attribute string nom;
  readonly attribute float solde;
};
```

Traduction →

```
public class XOperation {
  public void nom(String n);
  public String nom();
  public float solde();
}
```

- Méthode IDL ⇒ méthode Java
  - ✓ Paramètre in : utilisation directe du type
  - ✓ Paramètres out et inout : utilisation de Holder

```
interface X {
  float f(out long l, in Person p);
  void g(in long l, inout Person p);
};
```

Traduction →

```
public class XOperation {
  public float f(IntHolder l, Person p);
  public void g(int l, PersonHolder p);
}
```

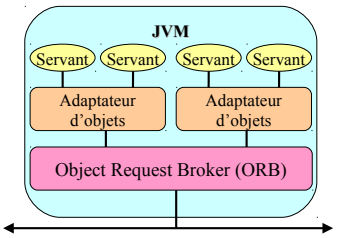
4. Développement Java

Trois entités fondamentales :

- Servant : objet d'un langage de programmation implantant un service Corba
- Adaptateur d'Objets (POA) : entité chargée de gérer des servants (activation, transmission de requêtes, …)
- Le bus à objet : entité chargée d'acheminer les requêtes entre serveurs

Écriture du programme serveur :

1. Création d'une ou plusieurs instances de servant
2. Enregistrement des servants dans l'adaptateur d'objets



4. Développement Java

Le bus à objets CORBA : connexion avec l'extérieur

- Trois méthodes fondamentales

```
class org.omg.CORBA.ORB {
  static ORB init(String[], Properties);        // initialisation l'ORB (un par JVM)
  void run();                                   // lancement de l'ORB
                  // aucun objet ne reçoit de requête avant l'appel à run()
  org.omg.CORBA.Object resolve_initial_references(String name);
                  // trouve un objet initial à partir de son nom
}
```

- Les objets initiaux sont
  - ✓ Le RootPOA : le père de tout adaptateur d'objets (voir transparent suivant)
  - ✓ Éventuellement le service de résolution de noms si il est local

4. Développement Java

Le POA : gère un ensemble de servants ayant des caractéristiques communes

- Ces caractéristiques sont contrôlées par les stratégies du POA
  Durée de vie des objets, allocation des requêtes à différents threads, allocation des identifiants d'objets…
- Tout servant est associé à un POA
- Existence d'un POA Racine (RootPOA) dans l'ORB
  Utilisation de stratégies par défaut suffisantes la plupart du temps

- A chaque POA est associé un POAManager qui contrôle l'état du POA
  Activé, mode tampon, désactivé
  Activation d'un POA : poa.the_POAManager().activate();

4. Développement Java

Objet CORBA, servant et mandataire

Un servant est référencé par objet CORBA

- Objet CORBA = référence distante (CORBA) vers un servant

Exp : org.omg.CORBA.Object rootref = orb.resolve_initial_references("RootPOA");

- **Un Objet CORBA n'est pas un mandataire!**
- ⇒ Rootref est une référence vers le RootPOA

Construction des mandataires à partir d'un objet CORBA

- A toute interface CORBA est associée à une classe *Helper*
  CompteHelper, POAHelper, CosNamingHelper…
- Méthode narrow dans les classes *Helper* pour la construction du mandataire

Exp : static org.omg.PortableServer.POA narrow(org.omg.CORBA.Object)
convertit org.omg.CORBA.Object en un mandataire du POA

Attention : ne jamais essayer d'obtenir un mandataire à partir d'un objet CORBA en utilisant le cast Java
Compte cpt = (Compte)obj; ⇒ plantage

4. Développement Java

Les IOR : Interoperable Object References

- Identifiant unique représentant un objet Corba
  - ✓ Référence distante vers un objet Corba
  - ✓ Contenu dans un objet du type CORBA.Object
- Peut être représenté sous la forme d'un chaîne de caractères
  org.omg.CORBA.Object obj = …;
  String s = orb.object_to_string(obj);
  ⇒ s est une chaîne qui contient l'IOR
- Une référence CORBA peut être construite à partir d'une chaîne
  String ior = …;
  org.omg.CORBA.Object obj = orb.object_to_string(ior);

- Utile pour échanger une référence distante entre un serveur et un client

4. Développement Java

Échange direct d'IOR

Côté serveur :

```
void saveIOR(String fileName, ORB orb, CORBA.Object obj) {
  String ior = orb.object_to_string(obj); // représentation sous forme de string de l'ior
  FileWriter fw = new FileWriter(fileName); // un flux de sortie vers un fichier
  fw.write(ior); // rempli avec l'ior
  fw.close(); // et fermé
}
```

Côté client :

```
CORBA.Object restoreIOR(String fileName, ORB orb) {
  // ouverture du flux d'entrée
  BufferedReader br = new BufferedReader(new FileReader(fileName));
  String ior = br.readLine(); // récupère l'ior enregistrée dans le fichier
  br.close(); // ferme le flux
  return orb.string_to_object(ior); // convertit la chaîne en object CORBA
}
```


4. Développement Java

Écriture du servant : deux méthodes

- Par héritage

```
public class Impl1 extends FooPOA {
  // implantation des méthodes de FooOperation
}
```

- Une instance de Impl1 est directement un servant Corba (car hérite de Servant)
- Impl1 ne peut pas hériter d'une autre classe Java (car héritage simple en Java)

- Par délégation

```
public class Impl2 implements FooOperation {
  // implantation des méthodes de FooOperation
}
```

- Une instance de Impl2 n'est pas un objet Servant, le Servant est construit par new FooPOATie(new Impl2());
- Utilisation possible de l'héritage simple Java

4. Développement Java

Écriture du servant par héritage

Fichier Compte.idl

```
module banque {
  interface Compte {
    string getTitulaire();
    float solde();
  }; };
```

Fichier CompteImpl.java

```
package banque;

public class CompteImplInherit extends ComptePOA {
  public String getTitulaire() { return ... }
  public float solde() { return ... }
}
```

Écriture du servant par délégation

Fichier Compte.idl

```
module banque {
  interface Compte {
    string getTitulaire();
    float solde();
 }; };
```

Fichier CompteImpl.java

```
package banque;

public class CompteImplTie implements CompteOperations {
  public String getTitulaire() { return ... }
  public float solde() { return ... }
}
```


Écriture du programme serveur

```
public static void main(String[] args) {
 ORB orb = ORB.init(args, null); /** Initialisation de l'ORB. */

 /** Récupération RootPOA, conversions et activation**/
 org.omg.CORBA.Object rootobj = orb.resolve_initial_references("RootPOA");
 POA poa = POAHelper.narrow(rootobj);
 poa.the_POAManager().activate();

 ComptePOA servant = new CompteImplInherit();   /** création de l'objet serveur */
  /** ou servant = new ComptePOATie(new CompteImplTie()); **/
 org.omg.CORBA.Object obj = poa.servant_to_reference(servant); /** enregistrement **/

 saveIOR("compte.ior", orb, obj); /** sauvegarde l'ior de obj dans compte.ior **/

 orb.run();  /** traitement des requêtes de clients **/
}
```


Écriture du programme client

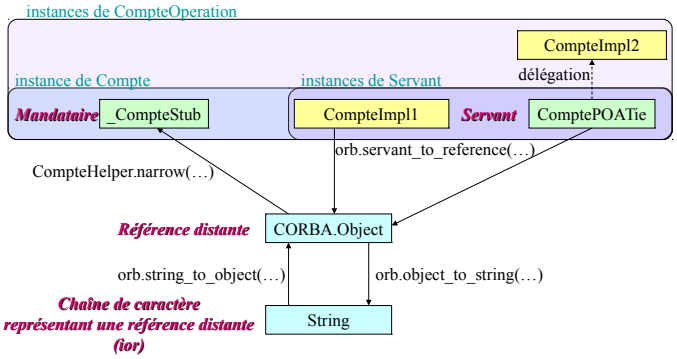
```
public static void main(String[] args) {
 ORB orb = ORB.init(args, null); /** Initialisation de l'ORB. */

 /** Récupération IOR et construction de la référence Corba **/
 org.omg.CORBA.Object obj = restore("compte.ior", orb);

 /** construction du mandataire **/
 Compte cpt = CompteHelper.narrow(obj);

 /** utilisation de compte **/
 System.out.println("Solde: "  + cpt.solde());
}
```


Les différentes représentations d'un objet



Compilation

- Compilation de Compte.idl : idlj Compte.idl
- Compilation des sources : java *.java

Exécution

- Lancement du serveur sur une machine m1 : java Server
- Copie de compte.ior sur la machine m2 du client
- Lancement du client sur m2

Utilisation du serveur de nom

- Le démon `orbd` est le serveur de nom CORBA
- Implémente l'interface `org.omg.CosNaming.NamingContext`
- Méthode principale : rebind(NameComponent[] nc, object o)
  - ✓ Associe l'objet o avec le nom désigné par nc
    - Arborescence de contexte nc[n+1] dans le contexte nc[n]
    - Nc[0] dans le contexte principal
  - ✓ Efface éventuellement l'ancienne valeur de nc
- Chaque NameComponent est un couple de 2 *strings* : nom de base + type
  - ✓ Chaque nom est une instance de la classe `org.omg.CosNaming.NameComponent`

```
import org.omg.CosNaming.NameComponent;
{ …
  NameComponent[] noms =
     new NameComponent[]{ new NameComponent("Bob","") };
  nc.rebind(noms, obj );
  … }
```


Localisation du serveur de résolution de noms

- Si serveur local : service initiaux

  orb.resolve_initial_reference("NameService")
- Si serveur distant : URL corbaloc

  corbaloc://serveur:port/NameService

```
{…
  org.omg.CORBA.Object ncobject =
        orb.string_to_object("corbaloc://localhost:1704/NameService");
…}
```

- URL corbaname :
  - ✓ Trouve un objet sur un serveur de noms
  - ✓ Construit avec corbaname::machine:port#nom

  Exp : corbaname::localhost:1704#Bob

Utilisation du serveur de résolution de noms

- Côté serveur

```
org.omg.CORBA.Object ncobj = orb.resolve_initial_reference("NameService");
NamingContext nc = NamingContextHelper.narrow(ncobj);

Compte servant = new ComptePOATie(new CompteImpl2());
org.omg.CORBA.Object obj = poa.servant_to_reference(servant);

NameComponent[] names =
    new NameComponent[] { new NameComponent("Bob", "") };
nc.rebind(names, obj);
```

- Côté client

```
String url = "corbaname::aphrodite.lip6.fr:1704#Bob";
org.omg.CORBA.Object obj = orb.string_to_object(url);
Compte compte = CompteHelper.narrow(obj);
```

Annexe

Quelques notions avancées sur Corba

5. Architecture

CORBA

www.omg.org

Défini par l'Object Management Group (OMG)

- organisme de standardisation international depuis 1989
- groupe de vendeurs de matériel, de système, d'applications, de conseils, d'organismes de recherche, ...
- ≈ 1000 membres (Sunsoft, HP, Compaq, IBM, Iona, Alcatel, ...)
- produit des spécifications

• OMA (Object Management Architecture)
architecture générale pour la gestion d'objets distribués

• CORBA
un des composants de l'OMA
permet aux objets distribués de communiquer

• Actuellement CORBA 2.6 (3.0 en préparation)

5. Architecture

OMG

Définit des interfaces et des spécifications

• implantations du ressort des membres
• interfaces et spec. disponibles librement (www.omg.org)

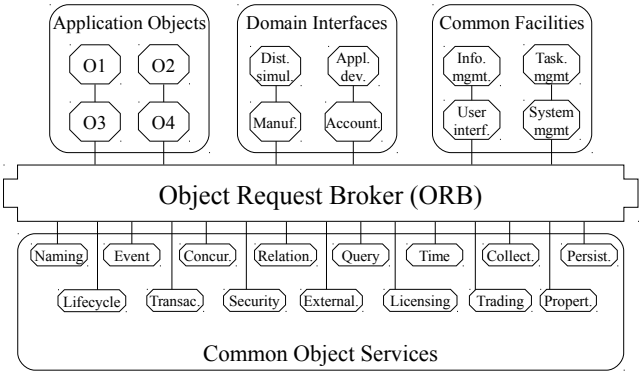
⇨ But : assurer l'**intéropérabilité** entre implantations

Fonctionnement

• apparition d'un besoin
• définition d'une RFP (Request For Proposal) avec objectifs et calendrier

• tous les membres intéressés soumettent une réponse avec un prototype
• les propositions sont révisées jusqu'à atteindre un consensus
• prototype final à fournir dans l'année

5. Architecture



5. Architecture

Définitions

• Bus logiciel (ORB : *Object Request Broker*)
infrastructure de communication de l'OMA
CORBA: spécifications de l'ORB

• Services (COS : *Common Object Services* ou *CORBAServices*)
«librairies» (classes) de services systèmes de base
COSS : spécifications des COS

• Facilités (*Common Facilities* ou *CORBAFacilities*)
«frameworks» logiciels pour des traitements courants

• Interfaces de domaines (*Domain Interfaces*)
«frameworks» logiciels spécialisés pour des domaines d'activités

• Objets d'application (*Application Objects*)
applications mises en place par les développeurs

5. Architecture

Les services communs

• abstractions de fonctionalités système courantes (nommage, transaction, ...)
• indépendants du domaine d'applications
• 16 services spécifiés actuellement
• rassemblés selon leur importance

COS sets 1 & 2

- **nommage**
- **événement** & notification
- cycle de vie
- persistance
- **transaction**
- concurrence
- relation
- externalisation

COS sets 3, 4 & 5

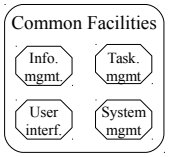
- requête
- gestion de licence
- propriétés
- sécurité
- temps
- **courtage**
- collection

5. Architecture

Les «facilités» communes

• «frameworks» logiciels de plus hauts niveaux que les services
• indépendantes du domaine d'applications
• également appelées facilités horizontales

- interface utilisateur
- gestion de l'information
- gestion du système
- gestion des tâches
- temps et internationalisation
- agent mobile
- impression



5. Architecture

Les «facilités» communes

Interface utilisateur

- gestion des documents composites
- scripts ....

Administration du système

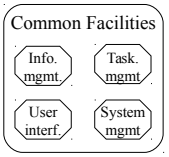
- instrumentation
- collecte de données
- sécurité
- suivi d'instance
- ordonnancement
- qualité de service
- gestion des événements

Gestion des tâches

- flux d'activités ...

Gestion de l'information

- modélisation
- stockage structuré
- échange
- codage et représentation

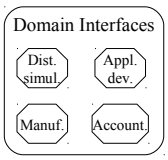


5. Architecture

Les interfaces de domaine

• «frameworks» logiciels de plus hauts niveaux que les services
• spécialisées pour un domaine d'applications particulier
• également appelées facilités verticales

- imagerie
- autoroute de l'information
- simulation distribuée
- comptabilité
- industrie pétrolière
- construction
- médical
- télécom
- finances
- commerce électronique
- transport



5. Architecture

Les interfaces de domaine

Telecom

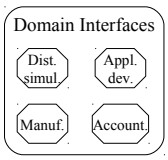
- AVS : Audio/Video Streams
- Telecom Log Service

Finance

- Currency Specification

Médical

- Person ID Specification
- Lexicon Query
- Resource Access Decision Facility
- Clinical Observation Access Service



5. Architecture

OMG

S'intéresse aussi à l'analyse/conception & à la gestion de l'information

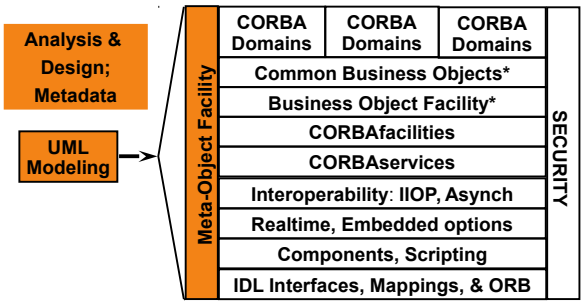
• UML : Unified Modeling Language
notation + processus (RUP) d'analyse/conception

• MOF : Meta-Object Facility
standard de méta-modélisation

• XMI : XML Model Interchange
format d'échange de données

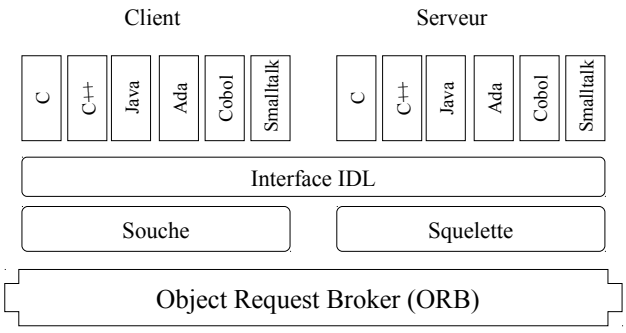
5. Architecture

OMG

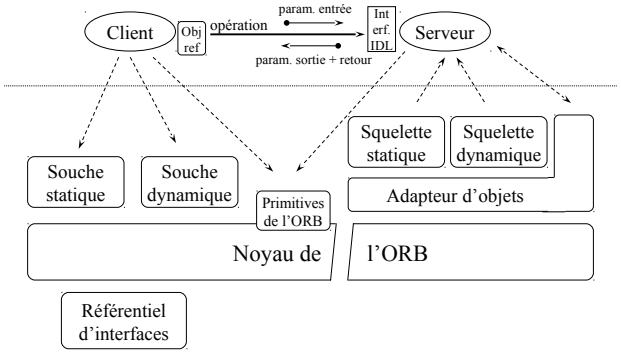


5. Architecture

CORBA



5. Architecture



5. Architecture

Le noyau de l'ORB (*ORB core*)

• gère la localisation des objets dans l'environnement
• implante les protocoles de communication entre objets
• accessible au travers d'un ensemble de primitives

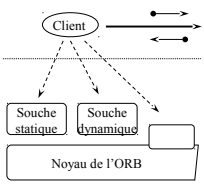
Le référentiel d'interfaces (*Interface Repository*)

• base de données des interfaces des objets serveurs
• une par environnement (groupement logique de machines)
• possibilité de fédérer les référentiels de différents environnements

5. Architecture

Souche

• prépare les paramètres d'entrée de l'invocation
• décode les paramètres de sortie et le résultat



Souche statique

• une par type d'objet serveur à invoquer
• identique aux souches clientes RPC
• générée à la compilation à partir de l'interface IDL

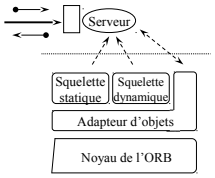
Souche dynamique

• souche générique construisant dynamiquement tout type de requêtes
• permet d'invoquer des objets serveurs que l'on découvre à l'exécution (i.e. dont on ne connaît pas l'interface à la compilation)

5. Architecture

Squelette

- symétrique de la souche
- décode les paramètres d'entrée des invocations
- prépare les paramètres de sortie et le résultat



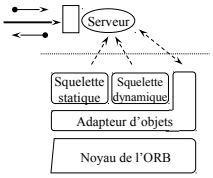
Adapteur d'objets

- réceptacle pour les objets serveurs
- interface entre les objets serveurs et l'ORB
- gère l'instantiation des objets serveurs
- crée les références d'objets
- aiguille les invocations de méthodes vers les objets serveurs
- plusieurs adapteurs (≠ ou identiques) peuvent cohabiter sur une même machine dans des espaces d'adressage ≠

5. Architecture

Squelette statique

- un par type d'objet serveur invoquable
- identique aux souches serveurs RPC
- généré à la compilation à partir de l'interface IDL



Squelette dynamique

- squelette générique prenant en compte dynamiquement tout type de requêtes
- permet de créer à l'exécution des classes d'objets serveurs (i.e. que l'on ne connaissait pas à la compilation)

5. Architecture

Référence d'objets (IOR: *Interoperable Object Reference*)

information permettant d'identifier de manière **unique** et non ambiguë tout objet dans un ORB

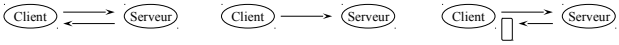
Type de l'objet	Adresse réseau	Clé de l'objet

Type de l'objet : permet de différencier des types d'objets ≠
Adresse réseau : adresse IP et numéro de port acceptant des invocations de méthodes pour cet objet
Clé de l'objet : identité de l'adaptateur sur ce site et de l'objet sur cet adaptateur

IDL:monInterf:1.0	milo.upmc.fr:1805	OA7, obj_979

5. Architecture

Modes d'invocation de méthodes



1. Synchrone (*synchronous*)
2. Asynchrone (*asynchronous*)
3. Semi-synchrone (*deferred synchronous*)

Rq: 3 n'est pas disponible avec un talon statique

Sémantique d'invocation

- 1 et 3 : «au plus une fois»
- 2 : «au mieux» (la réception du message n'est pas garantie)

- 1 : bloquant
- 2 et 3 : non bloquant

5. Architecture

GIOP (*General Inter-Orb Protocol*)

Protocole comprenant 8 messages pour assurer les communications entre objets

Request	invocation d'une méthode	(C)
Reply	réponse à une invocation	(S)
CancelRequest	annulation d'une invocation	(C)
LocateRequest	localisation d'un objet	(C)
LocateReply	réponse de localisation	(S)
CloseConnection	fermeture de connexion	(S)
MessageError	signalisation de message erroné	(S/C)
Fragment	fragmentation de messages	(S/C)

- GIOP nécessite un protocole de transport fiable, orienté connexion
- IIOP (*Internet IOP*) : implantation de GIOP au-dessus de TCP
- Autres implantations de GIOP au-dessus de HTTP, RPC DCE, RPC Sun
- Associé à un format d'encodage des données (CDR)

5. Architecture

Gestionnaires des objets CORBA

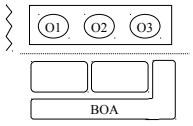
- prennent en charge la "vie" de l'objet
- se différencient par la façon dont ils
  - instancient les objets
  - activent les objets
  - gèrent la concurrence des traitements
  - gèrent les références d'objets
  - gèrent la persistance des objets

2 adaptateurs

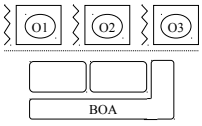
- BOA : le plus simple, historiquement le 1er, en voie d'extinction
- POA : plus complet, plus détaillé

5. Architecture

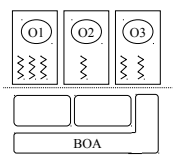
BOA séquentiel



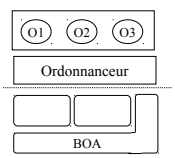
BOA avec concurrence inter-objets



BOA concurrent



BOA avec gestionnaire d'activités



5. Architecture

POA : *Portable Object Adapter*

Depuis CORBA 2.2

- séparation entre les notions de références d'objets CORBA (IORs) et les objets des lang. de prog. (appelés ***servants***) qui implantent le code des requêtes

- références d'objets persistantes
  - ⇨ en cas de plantage d'un *servant* un nouveau *servant* peut être réactivé avec le même IOR
- activation implicite des objets serveurs à partir du référentiel d'implantation
- +sieurs références possibles pour un même objet
- possibilité d'avoir une hiérarchie de POAs
  - ⇨ chaque POA dispose de sa (propre) politique de gestion des objets en terme d'activation et de concurrence

5. Architecture

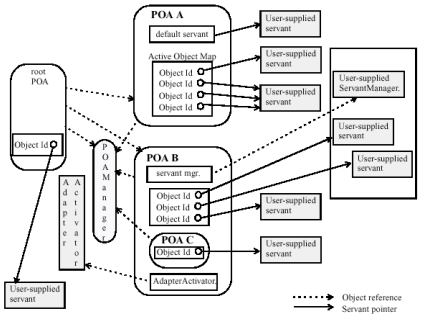
POA : *Portable Object Adapter*

Notions associées au POA

- *servant* — *a programming language object or entity that implements requests on one (or more) objects*
- *servant manager* — *activates/deactivates servants on demand*
- *object id* — une clé pour identifier chaque objet géré par un POA
- *object reference* — une référence d'objet CORBA (ie un IOR)
- *root POA* — le POA par défaut, racine de tous les POAs
- *POA manager* — *an object that encapsulates the processing state of one or more POAs*
- *adapter activator* — *creates POA on demand*

5. Architecture

POA : *Portable Object Adapter*



5. Architecture

POA : *Portable Object Adapter*

Les propriétés (*policies*) associées à un POA définissent son comportement

- Lifespan
- Id Assignment
- Id Uniqueness
- Implicit Activation
- Request Processing
- Servant Retention
- Thread

Lifespan définit si la référence d'objet survit ou non à son créateur

- TRANSIENT (par défaut)
- PERSISTANT

5. Architecture

POA : *Portable Object Adapter - Policies*

Id Assignment partie *object id* des références d'objet créée par le POA ou l'application

- SYSTEM_ID (par défaut)
- USER_ID

Id Uniqueness un servant dédié à un objet CORBA ou gérant +sieurs objets (l'identité du servant est alors distincte de l'*object id*)

- UNIQUE_ID (par défaut)
- MULTIPLE_ID

5. Architecture

POA : *Portable Object Adapter - Policies*

Implicit Activation servant activé explicitement par l'application ou à la demande

- NO_IMPLICIT_ACTIVATION (par défaut)
- IMPLICIT_ACTIVATION

Request Processing requêtes dirigées vers les servants par le POA (basé sur une table des objets actifs) ou par l'application (via un servant par défaut ou via un gestionnaire de servants)

- USE_ACTIVE_OBJECT_MAP_ONLY (par défaut)
- USE_DEFAULT_SERVANT
- USE_SERVANT_MANAGER

5. Architecture

POA : *Portable Object Adapter - Policies*

Servant Retention servant conservé en mémoire à tout moment ou détruit après utilisation

- RETAIN (par défaut)
- NON_RETAIN

Thread allocation des requêtes aux *threads* laissée à la charge de l'ORB ou séquentialisée par le POA
(un seul *thread* aiguille les requêtes mais réentrant)

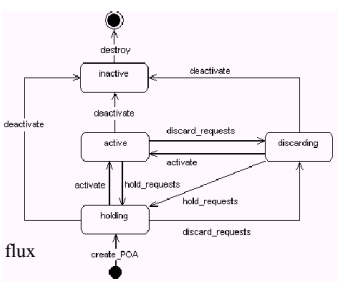
- ORB_CTRL_MODEL (par défaut)
- SINGLE_THREAD_MODEL

5. Architecture

POA : *Portable Object Adapter*

Etats du *POA manager*

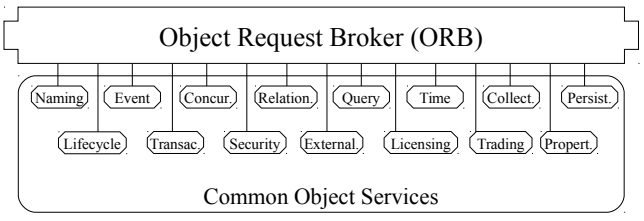
- *holding*
  les requêtes sont mises en attente tant que le POA n'est pas activé
- *active*
  le POA traite les requêtes
- *discarding*
  le POA rejette les requêtes
  ⇨ permet de mettre en oeuvre un mécanisme de contrôle de flux
- *inactive*
  le POA est sur le point d'être détruit



6. Services

Services système pour les applications CORBA

- ensemble d'objets serveurs remplissant des fonctions courantes
- chaque service est défini par une ou +sieurs interfaces IDL
- 16 services actuellement



6. Services

Services disponibles

- nommage, courtage, événements & notification, transactions

- cycle de vie	gestion de l'évolution des objets (déplacement, copie, ...)
- persistance	sauvegarde de l'état des objets
- concurrence	gestion de verrous
- relation	gestion d'associations (E/A) entre objets
- externalisation	mécanisme de «mise en flux» pour des objets
- requête	envoi de requête «à la SQL» vers des objets
- licence	contrôle de l'utilisation des objets
- propriétés	gestion d'attributs dynamiques pour des objets
- sécurité	gestion sécurisée de l'accès aux objets
- temps	serveur de temps et synchronisation d'horloges
- collection	gestion de groupes d'objets

Rq : peu d'ORBs offrent tous ces services

6. Services

Services

Tous les services sont des objets CORBA

- accessibles à distance
- associé à une **interface IDL**

Accès aux services

localement

```
org.omg.CORBA.Object orb.resolve_initial_references( "nom" )
```

à distance : URL corbaloc

```
corbaloc::serveur:port/nom
org.omg.CORBA.Object orb.string_to_object( "corbaloc:: ..." )
```

Les valeurs de *nom* utilisables sont définies dans les spec. :

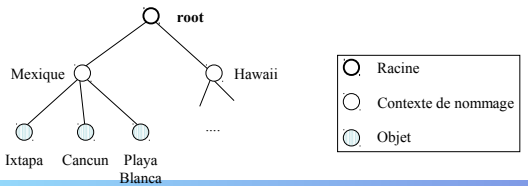
NameService, TradingService, NotificationService, TransactionCurrent, RootPOA, InterfaceRepository

6.1 Nommage

Permet de localiser un objet CORBA

- ≡ à un annuaire (pages blanches), DNS, ...
- organisé de façon hiérarchique (≡ hiérarchie de fichiers)

- chaque répertoire est appelé un «contexte de nommage»
- l'opération d'enregistrement d'un objet : une «liaison»
- l'opération de recherche d'un objet : «résolution» de nom



6.1 Nommage

Interface du service de nommage

Rq: exceptions omises

```
module CosNaming {
 struct NameComponent { string id; string kind; }
 typedef sequence<NameComponent> Name;

 interface NamingContext {

  void bind( in Name n, in Object o );        // enregistre un nom
  void rebind( in Name n, in Object o );      // ré-enregistre un nom
  void unbind( in Name n );                   // supprime un nom
  Object resolve( in Name n );                // résoud un nom

  void bind_context( in Name n, in NamingContext nc );
  void rebind_context( in Name n, in NamingContext nc );
  NamingContext new_context();
  NamingContext bind_new_context( in Name n );

  void destroy();      // détruit le contexte courant

  ...
```


6.1 Nommage

Interface du service de nommage (suite)

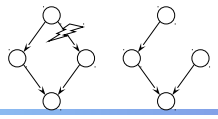
```
  void list( in unsigned long max,            // taille max pour bl
             out BindingList bl,              // tableau
             out BindingIterator bi );        // iterateur si > max
 }

 enum BindingType { nobject, ncontext };
 struct Binding { Name binding_name; BindingType binding_type; };
 typedef sequence<Binding> BindingList;

 interface BindingIterator {
   boolean next_one( out Binding b );
   boolean next_n( in unsigned long how_many, out BindingList bl );
   void destroy();
 };

};
```

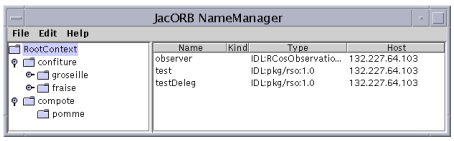
Rq : attention aux destruction de contexte



6.1 Nommage

Interface graphique

Exemple



RootContext : la racine du serveur de nom

Name	: le nom de l'objet
Kind	: une chaîne précisant le «type» de service fournit par l'objet
Type	: interface IDL implantée par l'objet
Host	: @ IP

6.1 Nommage

Désignation

URL corbaname

```
corbaname::serveur de noms:port#nom
corbaname::serveur de noms:port#répertoire/.../nom

corbaname::localhost:1704#observer
corbaname::localhost:1704#confiture/groseille/Bob
```

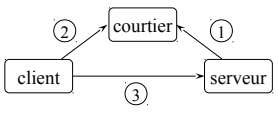
Recherche d'un objet

```
String url = "corbaname::localhost:1704#observer";
org.omg.CORBA.Object obj = orb.string_to_object(url);
```


6.2 Courtage

Permet de rechercher un type d'objet CORBA

- ≡ à un annuaire de pages **jaunes**
- on recherche un objet CORBA à partir de ses fonctions
- utilise un courtier (*trader*)
  1. le serveur s'enregistre auprès du courtier
  2. le client interroge le courtier
  3. le client invoque le serveur



Rq : le service de courtage est souvent utilisé conjointement au DII

6.2 Courtage

IDL

9 interfaces dans le module `org::omg::CosTrading`

Interfaces d'utilisation
- `Lookup` : pour rechercher un service
- `Register` : pour enregistrer un service
- `DynamicPropEvel` : interf. d'évaluation d'une propriété dynamique

Interfaces d'administration
- `Admin` : interf. d'admin. des attributs du courtier
- `ServiceTypeRepository` : référenciel des types de service

Interfaces d'itération
- `OfferIterator` : pour itérer sur un ensemble d'offres
- `OfferIdIterator` : pour itérer sur un ensemble d'ident. d'offres

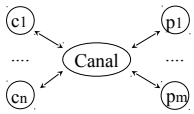
Interfaces de fédération
- `Link` : lien entre 2 courtiers
- `Proxy` : squelette d'offre (lien vers le courtier effectif)

6.3 Evénement

Permet de s'abonner auprès d'objets CORBA diffuseurs

2 rôles
- producteur d'événements (≡ d'information)
- consommateur d'événements (s'abonne auprès d'1 ou +sieurs producteurs)

Notion de canal d'événements
- liaison (n-m) entre producteurs et consommateurs par laquelle transitent les évts



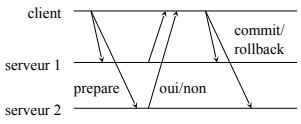
2 modes de diffusion
- «push» : initié par le producteur
- «pull» : initié par le consommateur

6.4 Transaction

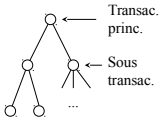
Permet d'effectuer des transactions sur des objets CORBA

Propriétés «habituelles» des transactions (ACID)
- atomicité : transac. effectuée complètement ou pas du tout
- cohérence : transac. préserve la cohérence des données
- isolation : exéc. // équivalentes à exéc. séquentielles
- durabilité : résultats de la transac. persistants

Algorithme de validation à 2 phases



Modèle de transactions imbriquées

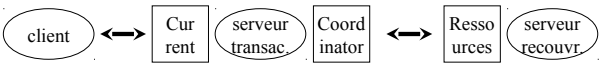


Optimisation : valid. 1 phase lorsque 1 seul serveur

6.4 Transaction

3 types d'acteurs
- client transactionnel
- serveur transactionnel : implante l'algorithme de validation à 2 phases
- serveur de recouvrement
  gère les ressources (données) sur lesquelles s'effectue la transaction

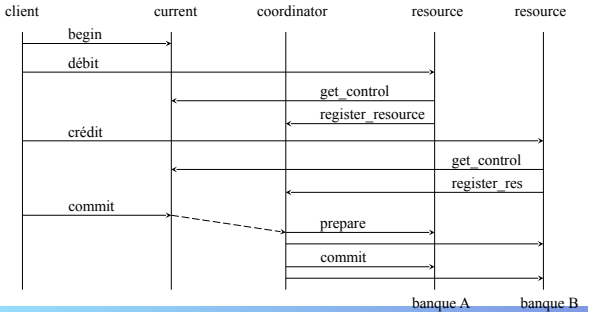
Interfaces
- serveur transactionnel
  - `Current` : interf. entre le client et le serveur transac.
  - `Coordinator` : interf. entre le serveur transac. et les serveurs de recouvr.
- serveur de recouvrement
  - `Resource` : interf. de gestion des ressources transactionnelles



6.4 Transaction

Scenario de fonctionnement

Débit de x frs sur le compte d'une banque A et
Crédit de x frs sur le compte d'une banque B



7. Fonctionnalités additionnelles

Délégation
- implantation des interfaces IDL par héritage (`extends ...POA`)
- pb lorsque classe ∈ hiérarchie héritage

⇨ implantation par délégation : 2 instances (délégateur + délégué)
- délégateur (`...POATie`) généré automatiquement par compilateur IDL
- délégué : classe code des services (`implements ...Operations`)

OBV (*Object By Value*)
- transmission d'objet CORBA par valeur
- pb hétérogénéité language (objet Java ↔ C++)
- nouveau mot-clé IDL `valuetype`

7. Fonctionnalités additionnelles

Type Any
- argument de méthode de "n'importe quel type"
- mot-clé IDL `any`
- classe `org.omg.CORBA.Any`

Invocation dynamique (DII *Dynamic Interface Invocation*)
- ≈ API `java.lang.reflect`
- appel d'objets CORBA dont on ne connait pas a priori l'interface
- découverte interface à l'exécution
- référentiel d'interfaces (IR *Interface Repository*)
- construction dynamique des requêtes (API `org.omg.CORBA.Request`)

7. Fonctionnalités additionnelles

Intercepteurs
- intercepter les communications entre objets CORBA
- ajouter des fonctionnalités (audit, sécurité, réplication, tolérance aux pannes, …) sans modifier les applications

- interception côté client et/ou côté serveur
- interception lors de la création d'un objet CORBA

8. Conclusion

Extensions CORBA 3.0

Communication en mode message (MOM : *Message-Oriented Middleware*)
- communication par boîtes à lettres + souple que le *deferred synchronous*
- 2 modèles : *callback* et *pooling*

Qualité de service (QoS : *Quality of Service*)
- permet d'indiquer la QoS voulue en terme de priorité des messages

Modèle de composants CCM
- but : enrichir la notion d'objet pour améliorer le déploiement et la config.

- Tolérance aux fautes
- Temps réel
- Extensible Transport Framework

8. Conclusion

Bibliographie

- Jérôme Daniel. *Au cœur de CORBA*. Vuibert, 2ème édition, 2001.
- J.M. Geib, C. Gransart, P. Merle. *CORBA: des concepts à la pratique*. Dunod, 2ème édition, 1999.
- R. Orfali, D. Harkey, J. Edwards. *Instant CORBA*. Wiley, 1996.
- J. Siegel. *CORBA 3 Fundamentals and Programming*. Wiley, 2000.
- T. Mowbray, R. Zahavi. *The Essential CORBA*. Wiley, 1995.

- OMG. *The Common Object Request Broker: Architecture and Specification*. `http://www.omg.org`
- Documentation de *IONA* : http://www.iona.html