USTL – Master mention Informatique – M1

Construction d'applications réparties

VI. Le bus logiciel CORBA

Lionel.Seinturier@lifl.fr

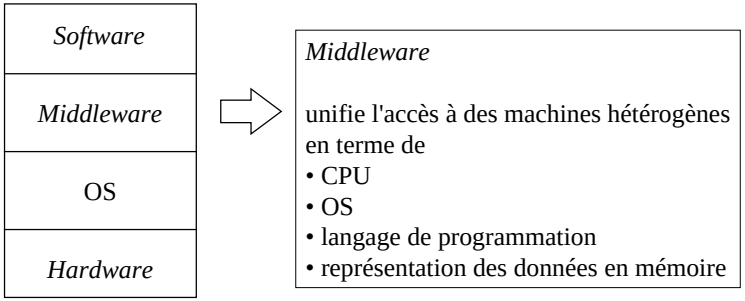
Plan

Introduction

CORBA

Common Object Request Broker Architecture

• *middleware* de communication multi-OS et multi-langages
• orienté objet (mais pas uniquement)



Introduction

OMG

www.omg.org

- 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 3.0
• début : 1989 / v2 : 1995 / support Java : 1998

Introduction

OMG

• définit des interfaces et des spécifications
• implantations du ressort des membres

⇨ 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

Introduction

CORBA

Nombreuses implémentations de CORBA disponibles

Commerciales

• ORBIX	IONA	`www.iona.com`
• VisiBroker	Borland	`www.borland.com/visibroker/`
• ORBacus	OOC	`www.orbacus.com`

Libres (et/ou gratuites)

• JDK	Sun	`java.sun.com`
• MICO	Univ. Francfort	`www.mico.org`
• JacORB	Univ. Berlin	`www.jacorb.org`
• TAO	Univ. Washington	`www.cs.wustl.edu/~schmidt/TAO.html`

`www.cetus-links.org` en recense 40+

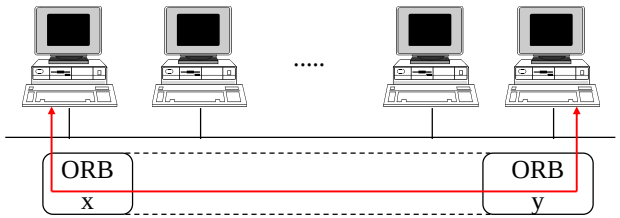
Introduction

CORBA

Un environnement de programmation répartie avec

• un RPC objet multi-OS, multi-langage
• des services (15) utilisables par les applications
• des fonctionnalités additionnelles par rapport à un (simple) RPC

Métaphore du bus logiciel (ORB)



Introduction

CORBA

Quelques grands principes

• Transparence à la localisation
• Transparence d'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
- clients dépendent des interfaces, pas des implantations
- références d'objets typées par les interfaces
- héritage permet d'étendre, de faire évoluer et de spécialiser les services

Introduction

Bibliographie

• J.-M. Geib, C. Gransart, P. Merle. *CORBA: des concepts à la pratique*. Dunod, 2ème édition, 1999.

• G. Brose, A. Vogel, K. Duddy. *Java Programming with CORBA*. 3rd ed. Wiley, 2001.

• 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`

IDL CORBA

Lionel Seinturier

Université des Sciences et Technologies de Lille

`Lionel.Seinturier@lifl.fr`

IDL

CORBA IDL (*Interface Definition Language*)

Langage de définition des services proposés par un objet serveur CORBA

⇨ seules les méthodes de l'interface pourront être invoquées à distance
⇨ interface IDL = contrat entre le client et le serveur

1. Ecriture d'une interface IDL
2. Ecriture d'une classe implantant l'interface
3. Ecriture du programme serveur
4. Ecriture du programme client

≡

1. déclaration des services accessibles à distance
2. définition du code des services
3. instanciation et enregistrement de l'objet serveur
4. interactions avec le serveur

IDL

Structure d'un fichier IDL

3 éléments principaux

- module : un espace de définitions
- interface : un regroupement de services
- méthode : un service

```
<modules>
    <interfaces>
        <methodes>
```

```
module universite {
    interface Etudiant {
        string coordonnees();
        boolean bacplus( in long annee );
    };
};
```

• plusieurs modules possibles dans un même fichier
• modules peuvent être emboîtés les uns dans les autres ou non

• éléments "secondaires" pouvant être définis dans un fichier IDL : types, constantes, exceptions, attributs

IDL

Syntaxe IDL

- proche de la syntaxe C/C++
- commentaires `/* */` ou `//`
- identificateurs : séquence de caractères alphabétiques, chiffres ou _
  (doivent commencer par un caractère alphabétique)
- casse des identificateurs non déterminante ⇨ `age` et `Age` même identificateur
- mais utilisation identificateur doit respecter sa casse de définition
  définition `CoordX` devra être utilisé `CoordX`

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
double          module          string          wstring
```


IDL

Syntaxe IDL

Différences / à la syntaxe C/C++

- identificateur pour chaque paramètre d'une méthode
- liste de paramètres vide `()` pas `(void)`
- les caractères ne peuvent pas être associés aux modificateurs `signed` ou `unsigned`

Préprocesseur

- identique à celui de C/C++
- substitution de macros (`#define`)
- compilation conditionnelle (`#ifdef` et `#ifndef`)
- inclusion de sources (`#include`)
- directives de compilation (`#pragma`)

IDL

Syntaxe IDL

Eléments syntaxiques

- types de base & construits
- modules
- interfaces
- méthodes

- constantes
- exceptions

- attributs
- héritage d'interfaces

IDL

∀ OS, archi., langage représentation normalisée
conversion prise en charge par l'ORB si machines avec représentation ≠

Types de base : entiers

Représentation signée

- `short` : 2 octets [-32768,+32767]
- `long` : 4 octets [-2^{31}, 2^{31}-1]
- `long long` : 8 octets [-2^{63}, 2^{63}-1]

Représentation non signée

- `unsigned short` : 2 octets [0,65535]
- `unsigned long` : 4 octets [0, 2^{32}-1]
- `unsigned long long` : 8 octets [0, 2^{64}-1]

Transmis tel quel sans conversion : type `octet`

IDL

Types de base : réels

- `float` : 4 octets
- `double` : 8 octets
- `long double` : 16 octets

- notation scientifique 17.5e-3 -8.0 0.57e3

Types de base : booléens, chaînes et caractères

- `boolean` : 2 valeurs TRUE et FALSE
- `char` : caractère (ISO Latin-1) 'e' '\" '\\'
- `string` : chaîne de caractères taille quelconque "hello"
- `string<n>` : chaîne de caractères taille max n

- `wchar` : caractère étendu (Unicode) 16 bits
- `wstring` : chaîne de caractères étendus

IDL

Types construits : structures, énumérations, définitions

Structures : regroupement d'éléments (≡ `struct` C, `tuple` SQL)

```
struct identificateur { liste d'éléments typés ; };
struct personne { string nom; long age; };
```

Enumération : liste de valeur pour un type (≡ `enum` C)

```
enum identificateur { liste de valeurs , };
enum couleur {rouge, vert, bleu};
```

⇨ attention : unicité des noms de valeurs entre **plusieurs** `enum`

```
enum piste { rouge, noire };  interdit
```

Définition de type : utilisation d'un raccourci pour un type (≡ `typedef` C)

```
typedef type identificateur;
typedef string<16> tprenom;
```


IDL

Types construits : unions

Union : type variable en fonction d'un type discriminant (de base ou énuméré)

```
enum categorie { racine, noeud, feuille };
typedef ... tracine;
typedef ... tnoeud;
typedef ... tfeuille;

union arbre switch(categorie) {
   case racine:  tracine dracine;
   case noeud:   tnoeud dnoeud;
   case feuille: tfeuille dfeuille;
};
```

Type par défaut possible lorsque tous les cas ne sont pas couverts

```
union foo switch(long) {
   case 0: float for0;
   default: string other;
};
```


IDL

Types construits : tableaux

Tableaux multi-dimensionnels de taille fixe (pas de mot clé spécial → `[]`)

```
type identificateur [taille] +
long vecteur[10]
float matrice[10][5]
```

les tailles sont obligatoires et constantes

Tableaux unidimentionnels de taille quelconque

```
sequence<type> identificateur ;
sequence<long> vecteur;
```

possibilité de spécifier une taille maximale

```
sequence<type,max> identificateur ;
sequence<long,16> vecteur;
```

IDL

Types construits : divers

Type objet CORBA

- permet de passer en paramètre **par référence** un objet CORBA
- mot clé : `Object`

Type indifférencié

- permet de passer en paramètre n'importe quel type (de base ou construit)
- mot clé : `any`

Type vide

- permet de spécifier un type vide (utilisation uniquement pour retour méthode)
- mot clé : `void`

IDL

Modules

conteneurs de définitions de : types, constantes, exceptions, **interfaces**, modules
hiérarchies de modules permettent de structurer les applications

```
module ident { ... };
```

Exemple

```
module mesanimaux {
    ...
    module alaville { ... };
    module alacampagne { ... };
    ...
};
```

- opérateur de résolution de portée `::` `mesanimaux::alaville::...`
- visibilité des définitions respecte l' $\subset$ des modules

IDL

Interfaces

• points d'accès aux objets CORBA
• identiques aux interfaces Java ou aux classes abstraites C++

• peuvent contenir : types, constantes, exceptions, attributs, **méthodes**
• peuvent hériter leur structure d'une ou +sieurs autres interfaces
• peuvent être prédéclarées en vue d'une utilisation ultérieure

```
interface ident [ : heritage ] { ... };
interface ident ;
```

Exemple de prédéclaration

```
interface A;          // prédéclarée (le compilateur «sait» que
                      // l'ident. A est associé à une interface
interface B { ... };  // utilise A
interface A { ... };  // utilise B
```


IDL

Méthodes

• sont nommées
• prennent des paramètres et retournent un résultat
• peuvent lever une ou +sieurs exceptions

```
typeRetour ident ( [ paramètre+ ] ) [raises (exception+)] ;
paramètre = mode type ident
```

Exemple

```
module banque {
   interface CompteItf {
      void deposer( in float montant );
      boolean retirer( in float montant );
      float solde();
      string getTitulaire();
   };
};
```

IDL

Méthodes

• surcharge **interdite** (pas 2 méthodes de même nom dans 1 interf.)
• paramètres
  - n'importe quel type de base
  - si type construit, définition d'un *typedef*
  - interface IDL

Exemple

```
module foo {
   interface FooItf {
      void setPrenoms( in sequence<string> prenoms ); // interdit
      typedef sequence<string> tprenoms;
      void setPrenoms( in tprenoms prenoms ); // ok
}; };

module bar {
   interface BarItf {
      void go( in foo::FooItf callback ); }; };
```


IDL

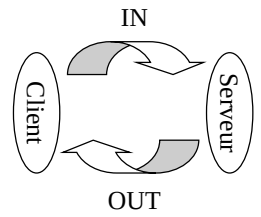
Méthodes

Passage de paramètres par **référence** pour les **objets CORBA**
par **valeur** pour tous les **autres**

• `in` (entrée)
  • le client fournit la valeur
  • si le serveur la modifie,
    le client **ne "voit" pas** la modification

• `inout` (entrée/sortie)
  • le client fournit la valeur
  • si le serveur la modifie,
    le client "voit" la modification

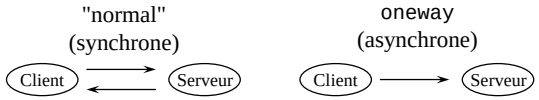
• `out` (sortie)
  • le **serveur** fournit la valeur
  • le client "voit" la modification



IDL

Méthodes

• peuvent être déclarée `oneway` (asynchrone)



• normal : le client attend la réponse avant de continuer
• `oneway` : **pas de réponse** + le client continue tout de suite après l'envoi
  ⇨ client et serveur s'exécutent en //

Particularité CORBA ≠ de sémantique entre normal et `oneway`

• normal : livraison messages garantie (dans la limite du protocole de transport)
• `oneway` : pas de garantie sur la livraison des messages
  ⇨ à n'utiliser que pour des invocations non vitales

IDL

Méthodes *oneway*

Pas de message de retour donc

- pas de paramètre de retour
- pas de paramètre `inout` ou `out`
- pas d'exception

Exemple

```
interface Itf {
   oneway float foo( in long bar );  // interdit
   oneway void foo( in long bar );   // ok
   long bar();
};
```

IDL

Constantes

Des variables typées dont la valeur est fixe

```
const type ident = expr ;
```

• types autorisés pour les constantes : uniquement **types de base**
• opérateurs autorisés dans l'expression
- unaires `+ - ~`
- binaires `+ - * / % << >> & | ^`

Exemple

```
interface Vrac {
    const long taille = 8;
    const long segments = 1024/taille;
    const string usage = «Hello»;
};
```


IDL

Exceptions

Des structures de données qui signalent une situation exceptionnelle

• utilisateur
• système

• utilisateur : déclarées et doivent être obligatoirement traitées
• système : peuvent être traitées (ou non)

Exceptions utilisateur

```
exception ident { donnée* };
```

```
exception Overflow { double limit; };
interface Math {
    double exp( in double x ) raises(Overflow);
};
```


IDL

Exceptions

Exemple d'exceptions système

CORBA::NO_MEMORY	pb allocation mémoire
CORBA::COMM_FAILURE	pb communication
CORBA::INV_OBJREF	référence d'objet invalide
CORBA::INTERNAL	erreur interne au bus CORBA
CORBA::MARSHAL	erreur encodage/désencodage données
CORBA::NO_IMPLEMENT	pas d'implémentation
CORBA::BAD_OPERATION	mauvaise opération
CORBA::FREE_MEM	mauvaise libération mémoire
CORBA::DATA_CONVERSION	mauvaise conversion de données
CORBA::OBJECT_NOT_EXIST	objet n'existe pas

IDL

Visibilité constantes, exceptions & *typedef*

Peuvent être définis à ≠ niveaux dans un fichier IDL

• globalement (en dehors de tout module)
• localement à un module
• localement à une interface

⇨ influence la visibilité de la définition

```
const long c0 = 0;
module m {
   const long c1 = 1;
   interface Itf {
      const long c2 = 2;
   };
};
```

Accès : `c0 , m::c1 , m::Itf::c2`

Attributs

Possibilité d'attacher des propriétés typées aux interfaces

≈ **accessibles avec des méthodes** setter/getter

```
[readonly] attribute type nom ;
```

```
interface Capteur {
   attribute float seuil;
   readonly attribute float temp; };
```

≡

```
interface Capteur {
   void setSeuil( in float val );
   float getSeuil();
   float getTemp(); };
```

`seuil` et `temp` ne sont jamais accédés comme des variables

mais

systématiquement via leurs méthodes setter/getter

`readonly` : en lecture seule

Héritage d'interfaces

Simple

```
interface Ouvrier : Employe { ... };
```

Multiple

```
interface Ouvrier : Employe, Personne { ... };
```

Une sous-interface

- hérite de tous les éléments de sa ou ses super-interfaces
- peut ajouter de nouveaux éléments
- peut redéfinir les types, constantes, exceptions d'une super-interface
- **ne peut pas** redéfinir les **attributs**, les **méthodes**

Héritage d'interfaces

- ordre de déclaration des interfaces non significatif
- possibilité d'héritage «en losange» (A←B A←C B,C←D)
- graphe d'héritage doit être sans cycle (A←B B←A **interdit**)

Utilisation éléments des super-interfaces doit être non ambigue

```
interface A { typedef string tnom; };
interface B { typedef struct {string nom; string prenom;} tnom; };
interface C: A,B {
   attribute tnom nom1;     // interdit: tnom ambigu
   attribute A::tnom nom2;  // A::tnom ok
   attribute B::tnom nom3;  // B::tnom ok
};
```


Héritage d'interfaces

Constantes, exceptions et *typedef* remplacés dès leur utilisation

```
interface A {
   const long taille = 3;
   typedef float coord[taille];
   void f( in coord tableau );
};
interface B { const long taille = 4; };
interface C: A,B {}
```

Signature `C::f void f(in coord tableau);` → `coord ≡ float[taille]`
Mais `C::taille` ambigu → ? 3 ? 4

Solution : la constante est remplacée dès son utilisation
⇨ dans `A` signature `A::f` `void f(in float tableau[3]);`

IDL

Conclusion IDL

Langage de déclaration des services offerts par un objet CORBA

Pas de code
Syntaxe proche C/C++/Java

Utilisé ∀ le langage de programmation utilisé pour l'implantation des services
Une interface IDL peut être implantée +sieurs fois
- de façon différente dans un même langage
- de façon différente/identique dans des langages ≠

Processus de développement CORBA

1. Ecriture d'une interface IDL
2. Ecriture d'une classe implantant l'interface
3. Ecriture du programme serveur
4. Ecriture du programme client

Mapping Java

Lionel Seinturier

Université des Sciences et Technologies de Lille

Lionel.Seinturier@lifl.fr

08/09/06

Mapping Java

Implantation des services

Choix d'un langage de programmation

- 6 langages supportés par l'OMG de façon standard
  C++, Java, Ada, Cobol, Smalltalk, C
- programmation CORBA possible avec d'autres langages
  Tcl, Perl, CLOS, Eiffel, Python, Modula, VB, …

Mapping : règles de traduction IDL → langage de programmation

Mapping Java

Mapping IDL → Java

- pour une interf. IDL `CompteItf` classe Java héritant de `CompteItf`**`POA`**
- classe *`foo`*`POA` générée à partir interf. IDL *`foo`*
- autant de méthodes dans la classe que dans l'interf. (sinon erreur compil.)
- possibilité ajout méthodes → non accessibles à distance

```
interface CompteItf {
   string getTitulaire();
   float solde();
   void deposer( in float montant );
   boolean retirer( in float montant );
};

public class CompteImpl extends CompteItfPOA {
   public String getTitulaire() { return ... }
   public float solde() { return ... }
   public void deposer( float montant ) { ... }
   public boolean retirer( float montant ) { ... }
}
```

Mapping Java

Compilation

Avec l'implantation CORBA du JDK 1.4

1. Génération des souches clients et serveur avec `idlj` ($JAVAHOME/bin)
2. Compilation avec `javac`

```
idlj -fall CompteItf.idl
javac *.java
```

Quelques options utiles de `idlj`

-td path	répertoire dans lequel les fichiers doivent être générés
-fall	génère les souches cliente et serveur (-fclient / -fserver)
-fallTIE	génère les souches selon le modèle "tie" (voir + loin)

Rq : principe identique avec d'autres implantations de CORBA

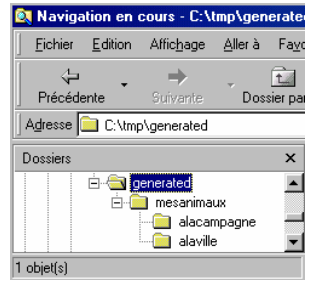
Mapping Java

Compilation

Les fichiers générés par `idlj` suivent la hiérarchie des modules IDL

```
module mesanimaux {
   module alaville { ... };
   module alacampagne { ... };
};
```

```
idlj -fall -td generated Animaux.idl
```



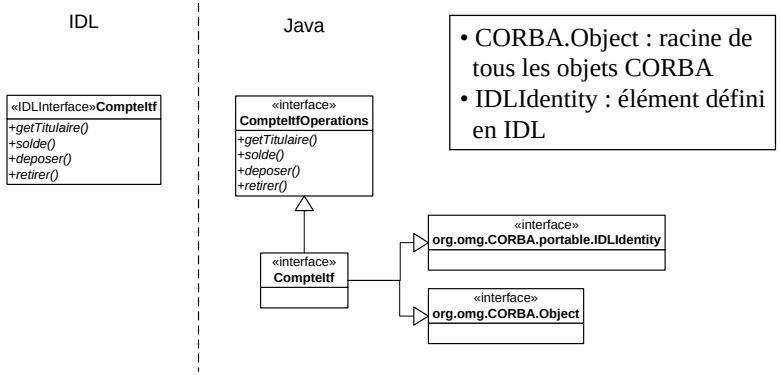
Fichiers générés pour chaque interf. IDL *foo*

- *foo*POA.java	squelette serveur
- *_foo*Stub.java	souche cliente
- *foo*Operations.java	interface Java ≡ interface IDL
- *foo*.java	interface Java (≡ IDL) + méthodes CORBA
- *foo*Helper.java	classe de gestion des objets CORBA implantant *foo*
- *foo*Holder.java	classe pour le mode `inout` (voir + loin)

Mapping Java

Code généré

À partir d'une interface IDL CompteItf

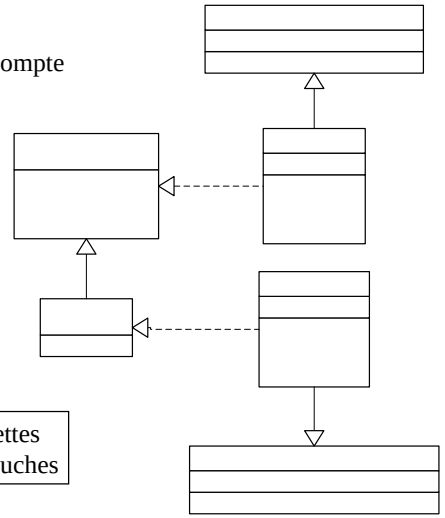


- CORBA.Object : racine de tous les objets CORBA
- IDLIdentity : élément défini en IDL

Mapping Java

Code généré (suite)

À partir d'une interface IDL Compte



- Servant : racine des squelettes
- ObjectImpl : racine des souches

Mapping Java

Correspondance types IDL → Java

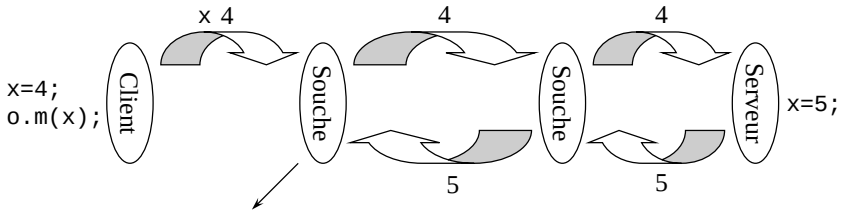
IDL	Java
octet	byte
short	short
long	int !!
long long	long !!
float	float
double	double
long double	*pas de correspondance* !!
char,wchar	char
string,wstring	java.lang.String
unsigned short	short
unsigned long	int
unsigned long long	long

Mapping Java

Passage de paramètres *inout* et *out*

Java : types simples (`int`, `float`, ...) transmis par valeur

⇨ comment transmettre en `inout` (ou `out`) un `int` ?



- la souche connaît simplement la valeur 4
- elle n'a pas la référence de x
⇨ elle ne peut pas la mettre à jour

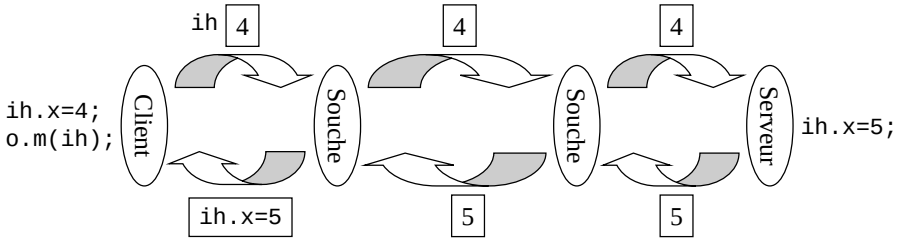
Mapping Java

Passage de paramètres *inout* et *out*

Solution

- encapsuler l'entier dans un "conteneur" (*holder*)
- `class IntHolder { int x; }`

⇨ la souche cliente peut mettre à jour la valeur contenue



Mapping Java

Passage de paramètres *inout* et *out*

Les implantations de CORBA fournissent des classes
`org.omg.CORBA.IntHolder StringHolder DoubleHolder ...`
pour tous les types de base

Chaque classe *Holder* contient

- un champ publique `value` du type contenu
- un constructeur avec 1 paramètre (du type contenu)

```
interface FooItf {
   void incr( inout double x );
};

import org.omg.CORBA.IntHolder;
public class FooImpl extends FooItfPOA {
   public void incr( DoubleHolder x ) { x.value = 3.14; }
}
```

Mapping Java

Correspondance types IDL → Java

IDL	Java
module	package
typedef *type*	substitution dans le code Java
`typedef string tnom;`	substitution de `tnom` par `string`
const	champ `final public static`
- dans une interface	champ dans l'interface Java associée interf. IDL
- hors d'une interface	
⇨ génération **interface Java** même nom que la constante	
⇨ contenant 1 champ `final public static` de nom `value`	
struct, enum, union	pas de construction ≡ en Java → classe

Mapping Java

Correspondance types IDL → Java

1 exception IDL → 1 classe Java

- de même nom que l'exception
- étend `org.omg.CORBA.UserException` (elle-même étend `java.lang.Exception`)
- autant de champs publiques que de champs associés à l'exception
- un constructeur avec autant de paramètres que de champs

```
exception Overflow { double limit; };
```

```
final public class Overflow extends org.omg.CORBA.UserException {
   public double limit = 0.0;
   public Overflow( double limit ) { ... }
   ...
}
```

- exception def. dans 1 module : générée dans le répertoire associé au module
- exception def. dans 1 **interface** *foo*: générée dans un rép. *foo*Package

Mapping Java

Correspondance types IDL → Java

1 type *struct* IDL → 1 classe Java

- de même nom que le type
- autant de champs dans la classe que dans la *struct*
- un constructeur vide
- un constructeur avec autant de champs que dans la *struct*

```
struct Personne { string nom; long age; };
```

```
public final class Personne {
   public String nom;
   public int age;
   public Personne() {}
   public Personne( String _nom, int _age ) { nom=_nom; age=_age; }
}
```


Mapping Java

Correspondance types IDL → Java

1 type *enum* IDL → 1 classe Java

- de même nom que le type
- avec un champ `int` pour chaque valeur de l'énumération
- avec un champ du type de la classe pour chaque valeur de l'énumération
- une méthode `value()` retournant un `int`

```
enum Couleur {rouge};
```

```
public class Couleur {
   private int __value;
   public static final int _rouge = 0;
   public static final Couleur rouge = new Couleur(_rouge);
   public int value() { return __value; }
   public static Couleur from_int(int value) { ... }
   ...
}
```

Mapping Java

Correspondance types IDL → Java

1 type *union* IDL → 1 classe Java

- de même nom que le type
- autant de champs que de types possibles
- un couple de méthodes setter/getter pour chaque champ
- un champ `__discriminator` du type du discriminant

```
union Foo switch(long) { case 0:float for0; default:string other; };
```

```
public class Foo {
   private int __discriminator;
   private float __for0;
   private String __other;

   public int discriminator() { return __discriminator; }
   public float for0() { return __for0; }
   public void for0( float value ) { __for0 = value; }
   ...
}
```


Mapping Java

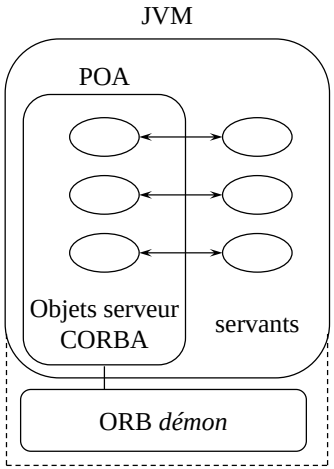
Ecriture du programme serveur

1. création d'1 ou +sieurs instances
2. appels système CORBA

POA : gestionnaire d'objets CORBA
entité chargée de gérer les objets CORBA
(activation, transmission de requêtes, ...)

- servant
- gestionnaire d'objets CORBA (POA)

90% cas : 1 servant ↔ 1 objet CORBA



Mapping Java

Ecriture du programme serveur

Principaux appels système CORBA

Classe `org.omg.CORBA.ORB`

- `static ORB init( String[], Properties )` : initialisation de l'ORB
- `void run()` : lancement de l'ORB
- `org.omg.CORBA.Object resolve_initial_references( String )`
  chaque ORB gère un ensemble de services (nommage, ...)
  le POA par défaut (*RootPOA*) ∈ à ces services

Tout objet CORBA

- implante une interface IDL
- hérite de `org.omg.CORBA.Object` (racine des objets CORBA)

Par extension les services (aussi *RootPOA*) sont aussi des objets CORBA

Mapping Java

Ecriture du programme serveur

Tout objet CORBA est associé à une classe *Helper*

- objet utilisateur : `CompteItf` → `CompteItf`**`Helper`**
- objet système : `org.omg.PortableServer.POA` → `POA`**`Helper`**

Méthode `narrow` assure la conversion de type

```
class CompteItfHelper {
  static CompteItf narrow(org.omg.CORBA.Object);
  // ...
}
```

```
class POAHelper {
  static POA narrow(org.omg.CORBA.Object);
  // ...
}
```

Conversion de type entre obj. CORBA avec narrow jamais avec cast Java

Mapping Java

Ecriture du programme serveur

Fichier CompteItf.idl

```
module banque {
   interface CompteItf {
      string getTitulaire();
      float solde();
      void deposer( in float montant );
      boolean retirer( in float montant );
}; };
```

Fichier CompteImpl.java

```
package banque;
public class CompteImpl extends CompteItfPOA {
   public String getTitulaire() { return ... }
   public float solde() { return ... }
   public void deposer( float montant ) { ... }
   public boolean retirer( float montant ) { ... }
}
```


Mapping Java

Ecriture du programme serveur

1er éléments du programme serveur

```
import org.omg.CORBA.ORB;
import org.omg.PortableServer.POA;
import org.omg.PortableServer.POAHelper;

public static void main( String[] args ) {

   /** Initialisation de l'ORB. */
   ORB orb = ORB.init(args,null);

   /** Récupération RootPOA et conversion vers POA. */
   org.omg.CORBA.Object rootobj =
      orb.resolve_initial_references("RootPOA");
   POA poa = POAHelper.narrow(rootobj);

   .............

   /** Attente de requête client. Tant que l'on n'a pas appelé
       run(), les objets CORBA ne répondent pas aux requêtes. */
   orb.run();
}
```


Mapping Java

Ecriture du programme serveur

Principaux appels système sur le POA (`org.omg.PortableServer.POA`)

- `the_POAManager().activate()`
  démarre le POA (voir + loin architecture)

- `org.omg.CORBA.Object servant_to_reference( Servant )`
  enregistre un servant et retourne sa référence CORBA

Programme serveur (suite)

```
.............
poa.the_POAManager().activate();

/** Création d'un objet CORBA et enregistrement. */
CompteImpl servant = new CompteImpl();
org.omg.CORBA.Object obj = poa.servant_to_reference(servant);

.............
```


Mapping Java

Ecriture du programme serveur

Publications des @ des objets CORBA

Plusieurs solutions

- enregistrement dans un fichier lu par le client
- enregistrement dans le serveur de noms de CORBA

Enregistrement de @ d'un objet CORBA dans un fichier

```
String ior = orb.object_to_string(obj);

FileWriter fw = new FileWriter("obj.ior");
fw.write(ior);
fw.close();
```

Mapping Java

Résumé

Fichier Serveur.java

```
public class Serveur {
   public static void main( String[] args ) {
      ORB orb = ORB.init(args,null);
      org.omg.CORBA.Object rootobj =
         orb.resolve_initial_references("RootPOA");
      POA poa = POAHelper.narrow(rootobj);
      poa.the_POAManager().activate();
      banque.CompteImpl servant = new banque.CompteImpl();
      org.omg.CORBA.Object obj = poa.servant_to_reference(servant);
      String ior = orb.object_to_string(obj);
      FileWriter fw = new FileWriter("obj.ior");
      fw.write(ior);
      fw.close();
      orb.run();
}  }
```


Mapping Java

Ecriture du programme client

1. initialisation de l'ORB côté client
2. récupération de l'@ de l'objet serveur
3. invocation des méthodes de l'objet serveur

```
ORB orb = ORB.init(args,null);

BufferedReader br = new BufferedReader( new FileReader("obj.ior") );
String ior = br.readLine();
br.close();
org.omg.CORBA.Object obj = orb.string_to_object(ior);
CompteItf compte = CompteItfHelper.narrow(obj);

String titulaire = compte.getTitulaire();
```


Mapping Java

Exécution

Avec l'implantation CORBA du JDK 1.4

1. Lancement du démon `orbd` ($JAVAHOME/bin)
2. Lancement du serveur
3. Lancement du client

Sur tous les sites
`orbd -ORBInitialPort 1704` → `port du démon orbd`
rq : pas nécessairement le même port sur tous les sites

Site serveur
```
java Serveur -ORBInitialPort 1704
```

Site client
```
java Client -ORBInitialPort 1704
```


Mapping Java

Utilisation du serveur de noms

Le démon `ordb` est aussi un serveur de noms

Le serveur de nom implante l'interface `org.omg.CosNaming.NamingContext`
Méthode principale `rebind` (ré-enregistre un objet)

Chaque objet serveur peut avoir +sieurs noms
Chaque nom est un couple de 2 *strings* : nom de base + type

Chaque nom est une instance de la classe `org.omg.CosNaming.NameComponent`
+sieurs noms = tableau de `NameComponent`

```
import org.omg.CosNaming.NameComponent;

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

Mapping Java

Utilisation du serveur de noms

Récupération du serveur de noms

1. le serveur de noms est local ⇨ liste des services initiaux

```
org.omg.CORBA.Object ncobj =
   orb.resolve_initial_references("NameService");
```

2. le serveur de noms est distant ⇨ URL corbaloc

```
corbaloc::serveur de noms:port/NameService
corbaloc::milo.upmc.fr:1704/NameService

org.omg.CORBA.Object ncobj =
   orb.string_to_object("corbaloc::milo.upmc.fr:1704/NameService");
```


Mapping Java

Utilisation du serveur de noms

Côté serveur

```
import org.omg.CosNaming.NameComponent;
import org.omg.CosNaming.NamingContext;
import org.omg.CosNaming.NamingContextHelper;


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

banque.CompteItf servant = new banque.CompteImpl();
org.omg.CORBA.Object obj = poa.servant_to_reference(servant);

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


Mapping Java

Utilisation du serveur de noms

Côté client

URL corbaname

```
corbaname::serveur de noms:port#nom
corbaname::serveur de noms:port#répertoire/.../nom
corbaname::milo.upmc.fr:1704#Bob
```

Recherche d'un objet

```
String url = "corbaname::milo.upmc.fr:1704#Bob";
org.omg.CORBA.Object obj = orb.string_to_object(url);
CompteIft compte = CompteItfHelper.narrow(obj);
```


Architecture CORBA

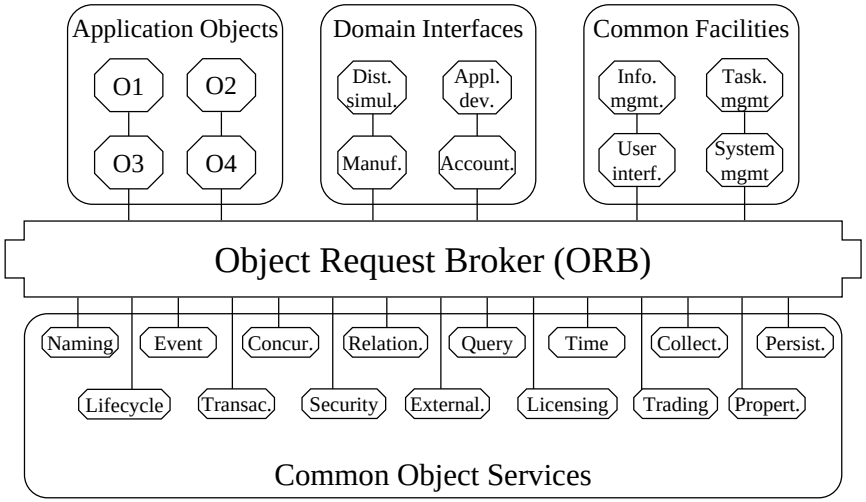
Lionel Seinturier

Université des Sciences et Technologies de Lille

Lionel.Seinturier@lifl.fr

08/09/06

Architecture



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 (de 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

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

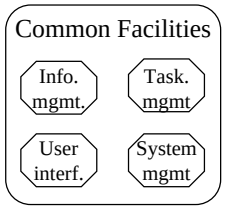
- requête
- gestion de licence
- propriétés
- sécurité
- temps
- **courtage**
- collection

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



Architecture

Les facilités communes

Interface utilisateur

- gestion des documents composites
- scripts ....

Administration du système

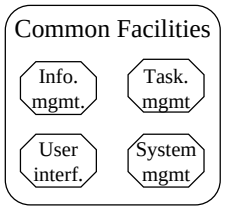
- instrumentation
- collecte de données
- sécurité
- suivi d'instance
- ordonnancement
- qualité de service
- gestion des événements

Gestion des tâches

- flux d'activités ...

Gestion de l'information

- modélisation
- stockage structuré
- échange
- codage et représentation

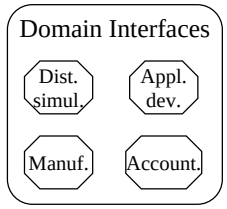


Architecture

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



Architecture

Les interfaces de domaine

Telecom

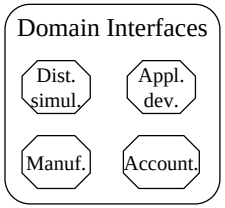
- AVS : Audio/Video Streams
- Telecom Log Service

Finance

- Currency Specification

Médical

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



Architecture

OMG

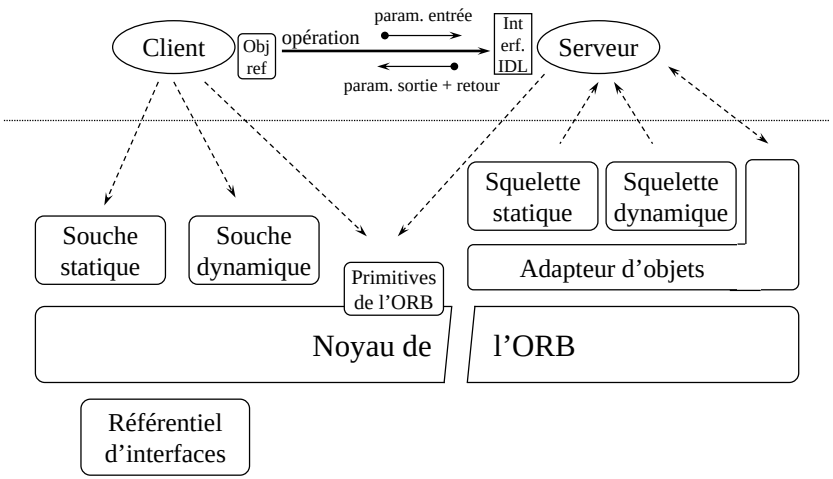
S'intéresse aussi à l'analyse/conception & à la gestion de l'information

• 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

Architecture



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

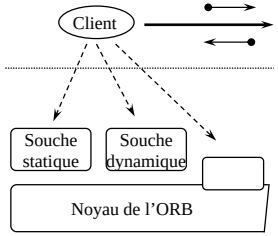
Le référentiel d'interfaces (*Interface Repository*)

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

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

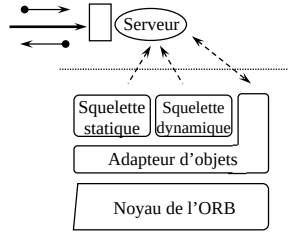
Souche dynamique

- 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)

Architecture

Squelette

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



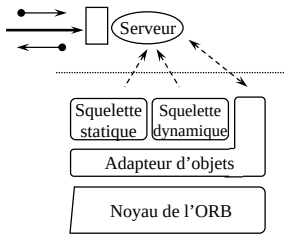
Adapteur d'objets

- 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 ≠

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)

Architecture

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

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

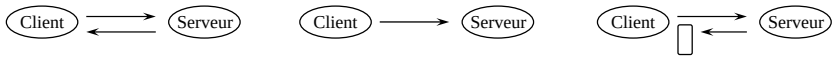
Type de l'objet	Adresse réseau	Clé de l'objet

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

Architecture

Modes d'invocation de méthodes



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

Rq: 3 n'est disponible qu'avec des souches dynamiques

Sémantique d'invocation

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

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 erronné	(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)

Architecture

Adapteurs d'objets CORBA

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 persistence des objets

Architecture

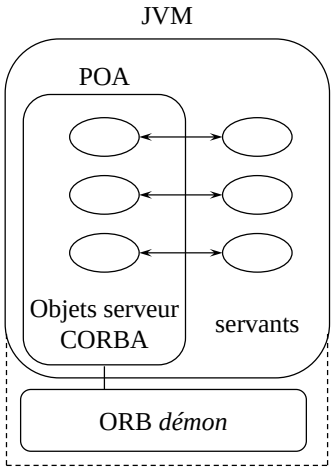
POA : *Portable Object Adapter*

Depuis CORBA 2.2

- séparation notions
  - objets serveur CORBA
  - implémentations des objets dans un langage de programmation (***servants***)

- IOR persistants
- activation implicite objets
- +sieurs références par objet
- hiérarchie de POAs

- notion de gestionnaire de POA (*POAManager*) pour gérer les POAs



Architecture

Paramètres de configuration du POA

Lifespan persistance de l'IOR

- TRANSIENT (par défaut)
- PERSISTANT

Id Assignment création *object id* (clé) des IOR

- SYSTEM_ID (par défaut)
- USER_ID (i.e. programme)

Id Uniqueness 1 objet CORBA par servant vs plusieurs

- UNIQUE_ID (par défaut)
- MULTIPLE_ID

Architecture

Paramètres de configuration du POA

Implicit Activation servant activé 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

Architecture

Paramètres de configuration du POA

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

- RETAIN (par défaut)
- NON_RETAIN

Thread — politique création threads ORB-*dependant* vs 1 seul thread

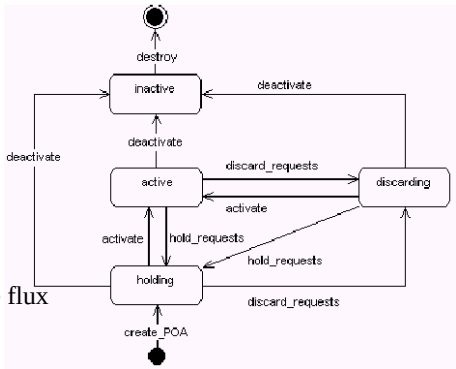
- ORB_CTRL_MODEL (par défaut)
- SINGLE_THREAD_MODEL

Architecture

POA Manager

Etats du *POA manager*

- *holding*
  les requêtes sont mises en attente
- *active*
  les requêtes sont traitées
- *discarding*
  les requêtes sont rejetées
  ⇨ permet de mettre en oeuvre un mécanisme de contrôle de flux
- *inactive*
  sur le point d'être détruit



Services

Lionel Seinturier

Université des Sciences et Technologies de Lille

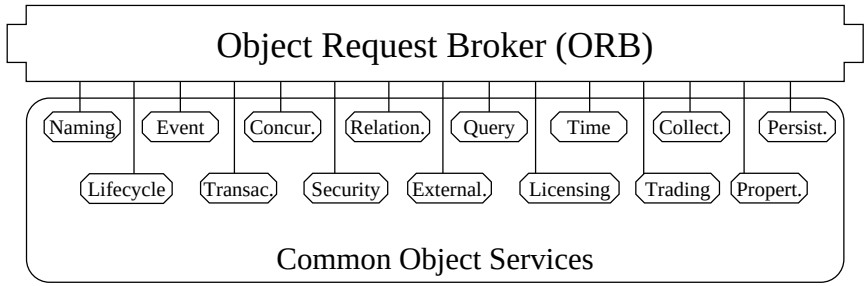
Lionel.Seinturier@lifl.fr

02/03/04

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



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

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

Services

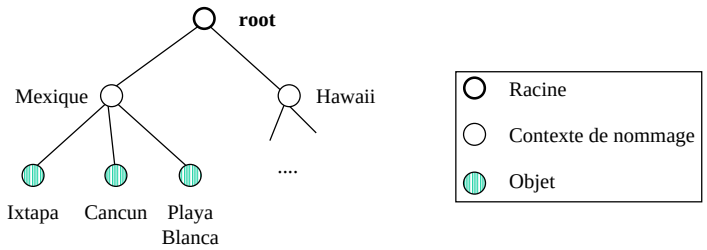
1. Nommage
2. Courtage
3. Evénement
4. Transaction

1. Nommage

Permet de localiser un objet CORBA

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

- «contexte de nommage» = nœud de la hiérarchie
- l'opération d'enregistrement d'un objet : une «liaison»
- l'opération de recherche d'un objet : «résolution» de nom



1. Nommage

Interface du service de nommage

Rq: exceptions omises

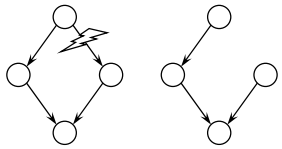
```
module CosNaming {
 struct NameComponent { string id; string kind; }
 typedef sequence<NameComponent> Name;

 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

  ...
```


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();
 };

};
```

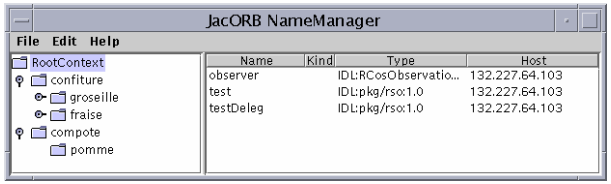
Rq : attention aux destruction de contexte



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

1. Nommage

Désignation

URL corbaname

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

corbaname::milo.upmc.fr:1704#observer
corbaname::milo.upmc.fr:1704#confiture/groseille/Bob
```

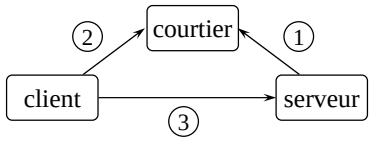
Recherche d'un objet

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

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

2. Courtage

Exemple

Service d'impression
≠ imprimantes dans +sieurs bâtiments avec des propriétés ≠

Propriétés caractéristiques de chaque imprimante :
étage, bâtiment, couleur/NB, résolution, bac recto-verso, agrafage, vitesse, coût

Exemple de recherches possibles

- l'imprimante géographiquement la + proche
- l'imprimante avec la meilleure résolution
- l'imprimante qui agrafe et fait du recto-verso

⇨ une ou +sieurs imprimantes peuvent correspondre

2. Courtage

Caractéristiques

- le client peut fournir des contraintes sur les objets recherchés
- le client peut demander à ce que les offres soient ordonnées
- le client peut limiter la recherche (ex. par + de *x* résultats)

- le courtier peut limiter la recherche
- on peut créer des fédérations de courtier
  → graphe orienté
  → liens unidirectionnels
  → la recherche peut donner lieu à un parcours du graphe

Obtention d'une référence sur le service de courtage

```
org.omg.CORBA.Object resolve_initial_references( "TradingService" )
```

retourne un objet implantant l'interf. IDL `org::omg::CosTrading::Lookup`

2. Courtage

Concepts introduits

Exportateur	un fournisseur de service (objet CORBA) (soit l'obj. s'exporte lui-même, soit on le fait pour lui)
Importateur	un utilisateur recherchant un service

Type de service

information nécessaire à la description d'un service
- une interface IDL (implantée par le service)
- 0 ou n propriétés : triples <attribut, nom (`string`), valeur (`any`)>

attributs
- *mandatory* : une instance de service doit fournir une valeur pour la prop.
- *readonly* : après qu'une valeur ait été fournie pour cette prop. on ne peut en accepter d'autres
- *mandatory* & *readonly*
- normal : tous les autres cas

2. Courtage

Concepts introduits

Offre de service

information nécessaire à l'**enregistrement** d'un service
- un nom de type de service
- IOR d'un objet qui fournit ce service
- 0 ou n propriétés pour ce service

```
struct Property {
    string name;
    any value;
};
typedef sequence<Property>
            PropertySeq;
struct OfferInfo {
    Object reference;
    string type;
    PropertySeq properties;
};
```

Propriétés dynamiques

valeur évaluée à l'exécution

Identificateur d'offre

retourné à l'exportateur par le courtier

```
typedef string OfferId;
```


2. Courtage

Sélection d'offre(s)

Liste potentiellement importante de services pour chaque sélection

Importateur spécifie

- type de service
- contrainte de sélection de service
- préférence dans la présentation des résultats
- politique de suivi de lien (si fédération)
- paramètres de limitations de la recherche

Contrainte de sélection de service

expression logique sur la valeur des propriétés

ex : `batiment=='a' && (etage==5 || etage==6)`

2. Courtage

Sélection d'offre(s)

Préférence dans la présentation des résultats

ordonnancement de la liste de résultat selon 5 possibilités

```
max expression numérique
min expression numérique
with expression booléenne
random
first
```

Exemple

`max batiment`
ordre descendant à partir de l'offre ayant la valeur de la prop. `batiment` la + élevée

2 groupes pour les offres

- préférence satisfaite
- pas possible de satisfaire la préférence

⇨ les offres sont retournées à l'importateur selon cet ordre

2. Courtage

Sélection d'offre(s)

Politique de suivi des liens

```
enum FollowOption {
  local_only,          // pas de propagation
  if_no_local,         // propagation si pas de réponse locale
  always               // systématique
};
```

chaque lien est associé
- à une valeur "max"
- à une valeur par défaut

- si la requête spécifie une valeur pour cette prop.
  elle est "bridée" par la valeur max.
- sinon
  la valeur par défaut est prise

2. Courtage

Sélection d'offre(s)

Politique de limitation de la recherche

courtier fournit des valeurs max et par défaut
l'importateur fournit (ou non) des valeurs

Essentiellement

- # d'offres à examiner
- # d'offres à retourner
- # de courtiers traversés
- courtier de départ

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)

2. Courtage

IDL

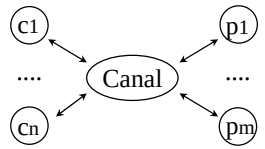
```
interface Lookup {
  void query (
    in string type, in string constr, in string pref,
    in PolicySeq policies, in SpecifiedProps desired_props,
    in unsigned long how_many,
    out OfferSeq offers, out OfferIterator offer_itr,
    out PolicyNameSeq limits_applied
  ) raises ( ... );
};

interface Query
  OfferId export (
    in Object reference, in string type,
    in PropertySeq properties
  ) raises ( ... );
};
```


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

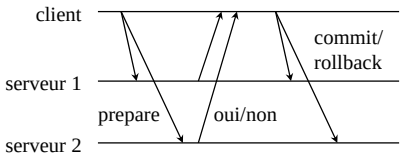
4. Transaction

Permet d'effectuer des transactions sur des objets CORBA

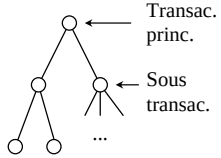
Propriétés «habituelles» des transactions (ACID)

- 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

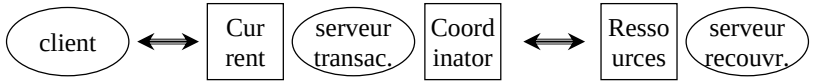
4. Transaction

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

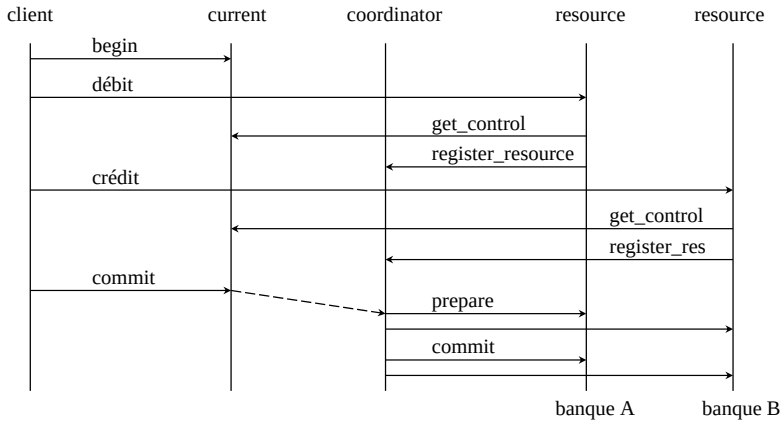
- serveur transactionnel
  - `Current` : interf. entre le client et le serveur transac.
  - `Coordinator` : interf. entre le serveur transac. et les serveurs de recouvr.
- serveur de recouvrement
  - `Resource` : interf. de gestion des ressources transactionnelles



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



Fonctionnalités avancées

Lionel Seinturier

Université des Sciences et Technologies de Lille

`Lionel.Seinturier@lifl.fr`

02/03/04

Plan

1. Délégation
2. OBV
3. Type Any
4. Invocation dynamique (DII)
5. Intercepteurs

1. Délégation

Rappel : Ecriture du programme serveur

Fichier CompteItf.idl

```
module banque {
   interface CompteItf {
      void string getTitulaire();
      float solde();
      void deposer( in float montant );
      boolean retirer( in float montant );
}; };
```

Fichier CompteImpl.java

```
package banque;
public class CompteImpl extends CompteItfPOA {
   public String getTitulaire() { return ... }
   public float solde() { return ... }
   public void deposer( float montant ) { ... }
   public boolean retirer( float montant ) { ... }
}
```


1. Délégation

Implantation d'un objet serveur par délégation

Pb : `CompteImpl` ∈ à une hiérarchie d'héritage
(pas d'héritage multiple en Java)

Génération d'une classe de délégation (dite *tie*) **`CompteItfPOATie`**

- qui hérite de la classe `CompteItfPOA`
- qui délègue le traitement des méthodes à `CompteImpl`
  condition : `CompteImpl` doit implanter l'interface `CompteItfOperations`
  interface Java contenant la déclaration des méthodes de l'interface IDL

```
package banque;
public class CompteImpl implements CompteItfOperations {
   public String getTitulaire() { return ... }
   public float solde() { return ... }
   public void deposer( float montant ) { ... }
   public boolean retirer( float montant ) { ... }
}
```


1. Délégation

Programme serveur

```
/** Création d'un objet CORBA et enregistrement aupres du POA. */
CompteItfOperations op = new CompteImpl();
CompteItf servant = new CompteItfPOATie(op);
org.omg.CORBA.Object obj = poa.servant_to_reference(servant);
```

2. OBV

Object by value (OBV)

Passage d'objet par valeur ⇨ l'objet est "envoyé" sur un site distant
Rappel : mot-clé IDL `Object` ⇨ passage par référence

Principe : définition d'un *valuetype* ≈ définition d'une classe contenant

- des champs : nommés, typés, publics ou privés
- des méthodes dite *factory* de construction du *valuetype*
- des méthodes

Syntaxe IDL

```
valuetype ident {
  (public|private) type attribut ; *
  factory profileDeMéthode ; *
  profileDeMéthode ; *
};
```


2. OBV

Exemple

```
module graphe {

 /** Un noeud d'un arbre binaire. */
 valuetype Noeud {

  private string nom;
  private long poids;
  public Noeud voisinDroit;
  public Noeud voisinGauche;

  factory construct( string nom, long poids );

  long sommeDesPoids();  // Σ des poids des noeuds du sous-arbre
} };
```

Mapping IDL → Java

valuetype
→ classe abstraite avec les champs et profils de méthode du *valuetype*
→ classe *factory* avec les profils des méthodes *factory*

2. OBV

Implantation d'un *valuetype*

• par héritage
• la classe doit avoir le même noeud que le *valuetype* + suffixe Impl

```
public class NoeudImpl extends Noeud

  public int sommeDesPoids() {

    int ret = poids;
    if (voisinDroit!=null) ret+=voisinDroit.sommeDesPoids();
    if (voisinGauche!=null) ret+=voisinGauche.sommeDesPoids();
    return ret;
  }
}
```


2. OBV

Utilisation d'un *valuetype*

valuetype définit un type IDL (idem type de base, ou interface "normale")
⇨ utilisable en paramètre d'une méthode

```
interface FooItf {
  void bar( in Noeud racine );
};
```

En Java

```
/** Création de la factory, puis d'un noeud. */
NoeudDefaultFactory ndf = new NoeudDefaultFactory();
Noeud noeud = ndf.construct("racine",12);

/** Transmission du noeud par valeur. */
aFoo.bar( noeud );
```

2. OBV

Divers

valuetype & héritage (simple uniquement)

```
valuetype ident : vt1 { ... };
```

→ *ident* hérite sa structure de *vt1*
→ redéfinitions interdites

valuetype & interfaces

1 *valuetype* peut implanter 1 ou +sieurs interfaces

```
interface Itf1 { ... };
interface Itf2 { ... };
valuetype ident supports Itf1, Itf2 { ... };
```

→ code des méthodes à fournir dans la classe d'implantation du *valuetype*

2. OBV

Divers

Interfaces abstraites

- passage par référence ? par valeur ?
- interface abstraite
  - soit étendue par une interface "normale"
  - soit implantée par un *valuetype*

```
abstract interface abs { ... };
interface Itf : abs { ... };
valuetype vt supports abs { ... };
```


3. Type Any

Type Any

Paramètres de n'importe quel type (simple, construit, objet CORBA)

- mot clé IDL `any`
- est mis en correspondance avec la classe `org.omg.CORBA.Any`

Instances de `org.omg.CORBA.Any` contiennent

- la donnée
- son type (instance de `org.omg.CORBA.TypeCode`)

- méthodes de lecture de la données
- méthodes d'écriture de la données

```
typeJava extract_typeIDL() throws org.omg.CORBA.BAD_OPERATION;
void insert_typeIDL( typeJava valeur );

int extract_long();
void insert_long( int valeur );
```


3. Type Any

Classe `org.omg.CORBA.Any`

- autant de `extract/insert` que de types simples
- instances créées à partir de `create_any()` classe `org.omg.CORBA.ORB`
- méthode `type()` retourne son type (`org.omg.CORBA.TypeCode`)

Classe `org.omg.CORBA.TypeCode`

- méthode `kind()` retourne une instance de `org.omg.CORBA.TCKind`
  - ⇨ valeur correspond au type contenu
  - ⇨ valeurs possibles

```
tk_boolean, tk_char, tk_long, tk_float, tk_double, ...
tk_enum, tk_struct, tk_union
tk_except
```

3. Type Any

Classe `org.omg.CORBA.TypeCode`

Méthodes appelables sur `TypeCode` en fonction du type

- struct, union, enum, exception
```
int member_count();
String member_name( int index );
```
- struct, union, exception
```
TypeCode member_type( int index );
```
- union
```
Any member_type( int index );
TypeCode discriminator_type();
int default_index();
```
- string, sequence, array
```
int length();
```
- typedef, sequence, array
```
TypeCode content_type();
```

∀ type
```
String name();
```


3. Type Any

Exemple d'utilisation du type Any

```
interface FooItf {
  void bar( in any paral );
};
```

```
import org.omg.CORBA.Any;
import org.omg.CORBA.TCKind;
import org.omg.CORBA.TypeCode;
```

```
void bar( Any param ) {
  TypeCode type = param.type();
  String name = type.name();
  TCKind kind = type.kind();
```

Pb : méthode à appeller pour extraire la valeur dépend du type
(`extract_long(), extract_string(), ...`)

→ tester toutes les valeurs possibles de `kind` !!
→ lourd !!

3. Type Any

Exemple d'utilisation du type Any

→ en pratique

• soit vraiment besoin extraction
faire en sorte limiter à quelques types (ex long, string, double)
rq : mais dans ce cas on peut aussi utiliser union

• soit juste besoin transmettre la valeur sans la consulter

Client

```
Any param = orb.create_any();
param.insert_string("Hello");
aFoo.bar(param);
```


3. Type Any

Types construits et any

Classe Helper associée au type contient des méthodes `static` pour lire/écrire

```
void insert( Any any, type valeur );
type extract( Any any );
```

Exemple

```
struct Personne { string nom; string prenom; };
```

```
Any param = orb.create_any();
PersonneHelper.insert( param, new Personne("Marley","Bob") );
aFoo.bar(param);
```

Classe Helper

méthode `static TypeCode type()`
⇨ utilisation côté serveur

4. Invocation dynamique

DII : *Dynamic Interface Invocation*

Permet d'invoquer des objets dont on ne connaît pas l'interface à la compilation

⇨ interface «découverte» à l'exécution

Les interfaces stockées dans le **référentiel d'interfaces**

- base de données des interfaces des objets serveurs
- une par environnement (groupement logique de machines)
- possibilité de fédérer les référentiels de ≠ environnements
- alimenté avec des fichiers IDL
- organisation hiérarchique
- structure reflète celles des éléments IDL (modules, interfaces, ...)

4. Invocation dynamique

Référentiel d'interfaces

Repository
|-> ConstantDef
|-> TypeDef
|-> ExceptionDef
|-> InterfaceDef
|-> ModuleDef
|-> ...
|-> InterfaceDef
|-> ...
|-> AttributeDef
|-> OperationDef
|-> ParameterDef
|-> ExceptionDef

Exemple

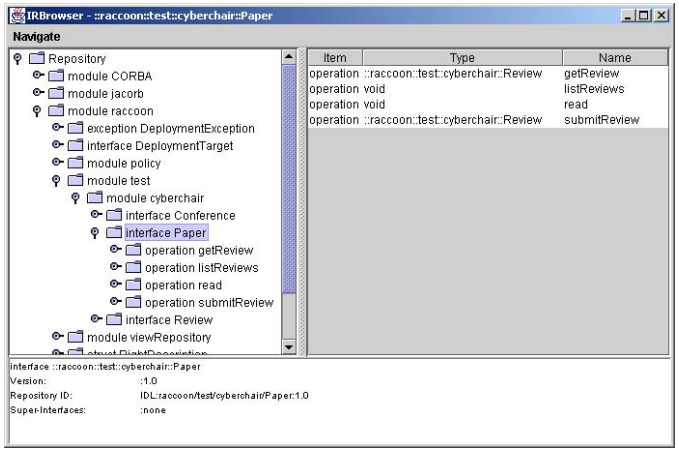
```
module m {
  interface I {
    long meth( in long arg );
} }
```

est stockée

Repository
|-> (*ModuleDef*) m
|-> *(InterfaceDef)* I
|-> *(OperationDef)* (meth,long)
|-> *(ParameterDef)* (in,long,arg)

4. Invocation dynamique

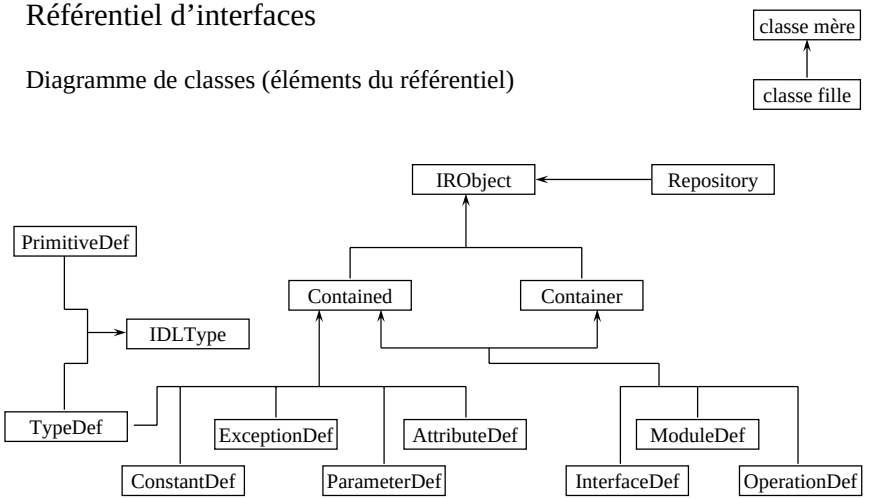
Référentiel d'interfaces



4. Invocation dynamique

Référentiel d'interfaces

Diagramme de classes (éléments du référentiel)



4. Invocation dynamique

Référentiel d'interfaces

```
org.omg.CORBA.Object o = ...;

/** Récupération de l'interface. */
InterfaceDef itf = o._get_interface();
```

Invocation dynamique

```
/** Requête: méthode foo, param in = 12, retour de type long. */
Request request = o._request("foo");
request.add_in_arg().insert_long(12);
request.set_return_type(
   orb.get_primitive_tc(org.omg.CORBA.TCKind.tk_long) );

/** Invocation synchrone (bloquante). */
request.invoke();

/** Résultat. */
Any resultat = request.return_value();
System.out.println( resultat.extract_long() );
```


4. Invocation dynamique

Invocation dynamique

Sémantiques d'invocation possibles

Méthodes de la classe `org.omg.CORBA.Request`

- `invoke` appel bloquant
- `send_oneway` appel asynchrone (*oneway*)
- `send_deferred` appel non bloquant
- `poll_response` teste la présence de la réponse
- `get_response` récupère la réponse

Méthodes de la classe `org.omg.CORBA.ORB`

- `send_multiple_request_oneway(Request[])` envoi +sieurs requêtes *oneway*
- `send_multiple_request_deferred` envoi non bloquant
- `poll_next_response` teste la présence d'une réponse
- `get_next_response` récupère une réponse

4. Invocation dynamique

Avantages

- il n'est pas nécessaire de connaître les interfaces à la compilation
- plus flexible que l'invocation statique
- sémantiques d'invocations plus riches
- permet de manipuler les contextes d'exécutions des clients

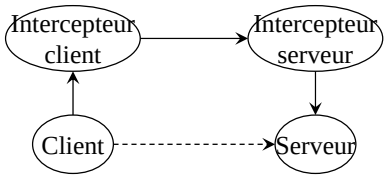
Inconvénients

- plus difficile à manipuler
- pas de vérification de type
- plus lent

5. Intercepteurs

Portable Interceptors (PI)

Mécanisme permettant d'intercepter les communications
(≈ API `java.lang.reflect`)
But : audit, sécurité, ajout d'appels à d'autres services/objets, …



3 types d'intercepteurs

- requête côté client
- requête côté serveur
- création d'objet (mécanisme de personalisation des IOR)

Portable Interceptors (PI)

Package `org.omg.PortableInterceptor`

Intercepteur côté client	`ClientRequestInterceptor`
• envoi d'une requête	`send_request`
• envoi d'un "poll"	`send_poll`
• reception d'une réponse	`receive_reply`
• reception d'une exception	`receive_exception`
Intercepteur côté serveur	`ServerRequestInterceptor`
• reception d'une requête	`receive_request`
• envoi d'une exception	`send_exception`
• envoi d'une réponse	`send_reply`
Intercepteur de création d'obj.	`IORInterceptor`
• création obj. CORBA	`establish_components`

Portable Interceptors (PI)

Fonctionalités

- lire les paramètres des requêtes (mais pas modifier)
- détourner l'appel vers un autre destinataire (exception `ForwardRequest`)
- ajouter/récupérer des informations aux requêtes (`service_context`)
- ajouter/récupérer des informations aux IOR (`tag_component`)

Enregistrement des intercepteurs auprès de l'ORB lors de son initialisation (appel `init`)
⇨ fournir une classe implantant `org.omg.PortableInterceptor.ORBInitializer`

```
void pre_init( ORBInitInfo info )
void post_init( ORBInitInfo info )

org.omg.PortableInterceptor.ORBInitInfo
    add_client_request_interceptor, add_server, add_ior
    register_initial_reference
```