

CORBA

Common Object Request Broker Architecture
Mise en pratique avec Orbacus

EPU Nice – Sophia Antipolis – Dep SI - 2010

Mireille Blay-Fornarino
✉ blay@polytech.unice.fr

Consensus pour l'interopérabilité

Le problème : Intégration des applications

Pas de consensus sur les langages de programmation

Pas de consensus sur les plate-formes de développement

Pas de consensus sur les systèmes d'exploitation

Pas de consensus sur les protocoles réseau

Pas de consensus sur les formats des données manipulées
par les applications

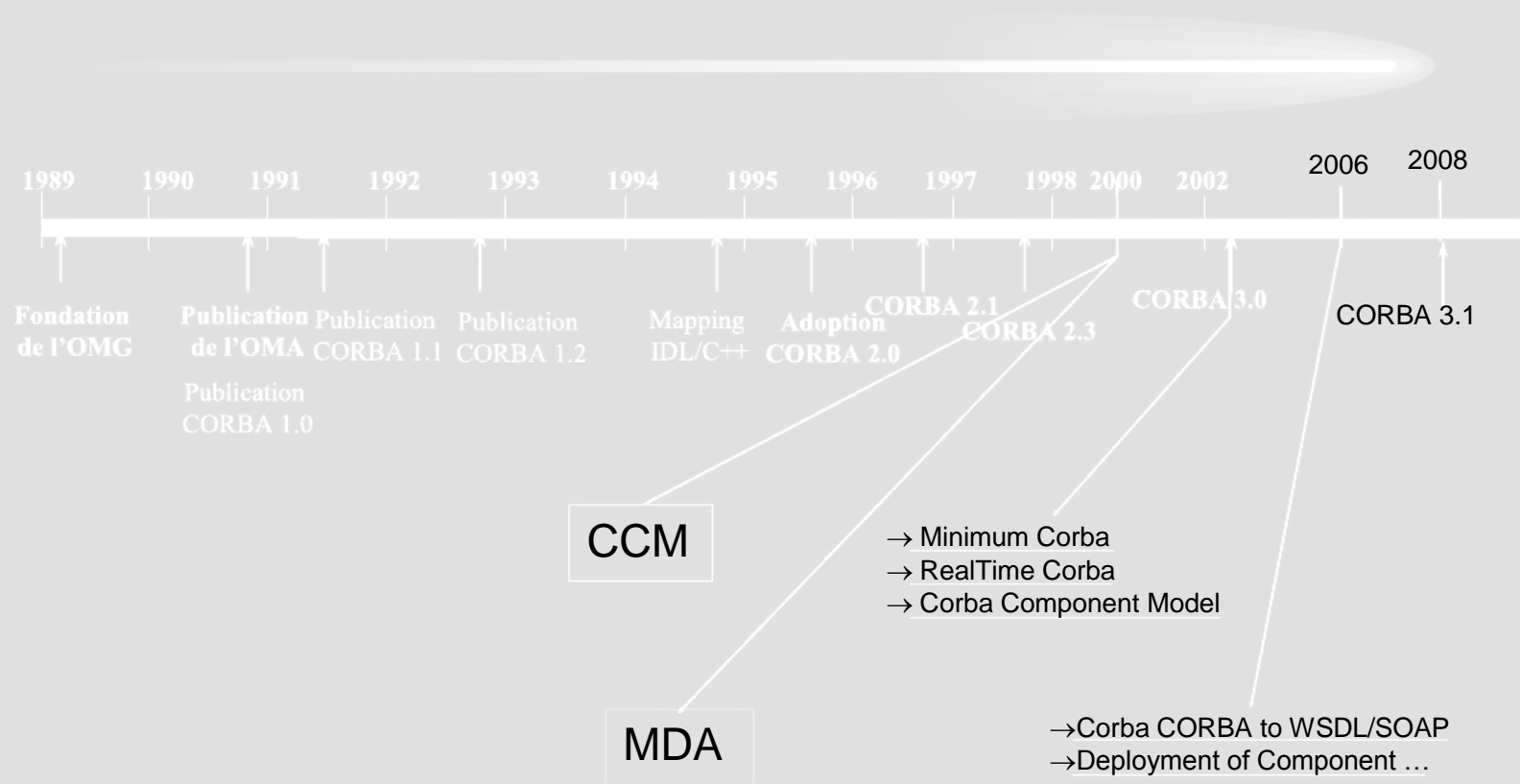
➔ Recherche d'un Consensus pour l'interopérabilité

Common Object Request Broker Architecture : CORBA Plate-forme client/serveur orientée objets

Un standard pour l'interopérabilité entre objets

- Support pour différents langages
- Support pour différentes plate-formes (interopérabilité)
- Communications au travers du réseau
- Des services (Distributed transactions, events, ...)
- Guides et modèles de programmation

Historique



Object Management Group (OMG)

<http://www.omg.org>

- consortium international créé en 1989
- but non lucratif
- regroupement de plus de 460 organismes
 - constructeurs (SUN, HP, DEC, IBM, ...)
 - environnements systèmes (Microsoft, OSF, Novell, ...)
 - outils et langages (Iona, Object Design, Borland, ...)
 - produits et BD (Lotus, Oracle, Informix, O2, ...)
 - industriels (Boeing, Alcatel, Thomson, ...)
 - institutions et universités (INRIA, NASA, LIFL, W3C)

OMG : quelques spécifications

OMA : pour l'architecture logicielle

MDA (Model Driven Architecture) ➡

CORBA : pour le « middleware » technique

UML pour la modélisation

UML = Unified Modeling language

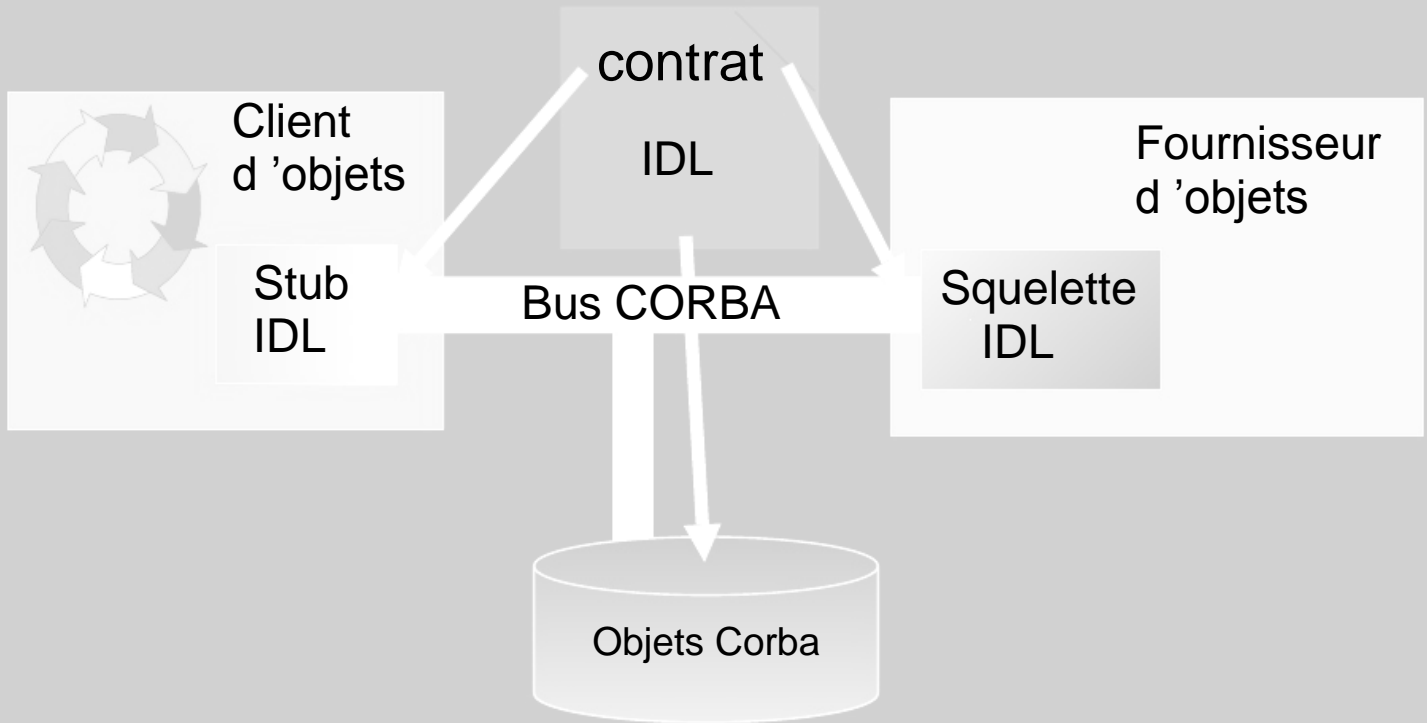
MOF pour la méta-modélisation

MOF = Meta-Object Facility

Business Process Modeling Notation (BPMN) Information

Data-Distribution Service for Real-Time Systems (DDS)

Le langage IDL: Un « esperanto » pour les objets

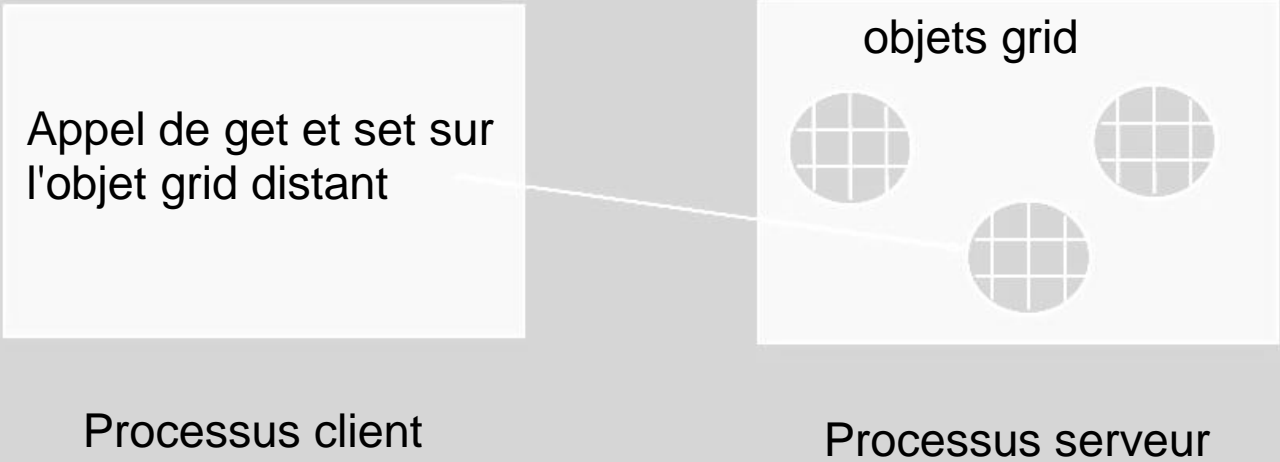


Spécification interface IDL (1/2)

Un exemple :

Un objet grid est un tableau contenant des valeurs.

Les valeurs sont accessibles grâce aux opérateurs get et set qui peuvent être invoqués à distance.



Appel de get et set sur
l'objet grid distant

The diagram consists of two white rectangular boxes. The left box contains the text 'Appel de get et set sur l'objet grid distant'. The right box is titled 'objets grid' and contains three circular icons, each with a 3x3 grid pattern. A thin white line connects the text in the left box to the middle grid icon in the right box. Below the left box is the label 'Processus client' and below the right box is the label 'Processus serveur'.

Processus client

objets grid

Processus serveur

Spécification interface IDL (2/2)

```
interface Grid {  
    readonly attribute short height;  
    readonly attribute short width;  
    void set (in short n, in short m, in long value);  
    long get(in short n, in short m);  
    void copyIn(inout Grid g);  
};
```

Le langage IDL: Interface Definition Language

- ✓ Langage de spécification d'interfaces, supportant
 - l'héritage multiple
 - indépendant de tout langage de programmation ou compilateur (langage pivot entre applications)

- Langage utilisé pour générer les stubs, les squelettes et pour définir les interfaces du référentiel d'interface
 - la correspondance IDL-langage de programmation est fournie pour les langages C, C++, Java, Smalltalk, Ada, Cobol.

Éléments IDL

Une spécification IDL définit un ou plusieurs types, constantes, exceptions, interfaces, modules,...

- pragma (prefix,version,... : #pragma prefix « omg.org »)
- constantes (const Ident = expression)
- types de base au format binaire normalisé
- nouveaux types
(typedef, enum, struct, union, array, sequence)
- exceptions
- types de méta-données (TypeCode, any)

Exemples

définitions
de type

```
module unService {  
  
    typedef unsigned long EntierPositif;  
    typedef sequence<Positif> desEntiersPositifs;  
  
    struct Date { // Structure pour la notion de date.  
interface Premier {;  
    Mois le_mois;  
    Annee l_annee;  
    } boolean est_premier ( in EntierPositif nombre);  
    desEntiersPositifs nombres_premiers (in EntierPositif nombre);  
};  
};  
  
module monApplication {  
    interface MonService {  
        unService::EntierPositif prochainPremier(..);  
    };  
};
```

Éléments IDL

- Un module permet de limiter la validité des identificateurs (namespace/package)
- Interface : ensemble d'opérations et de types =>(classe C++/Java)

Syntaxe :

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

- opération (synchrone ou asynchrone sans résultat)
- attribut (possibilité de lecture seule)

Surcharge Interdite

- ✓ Réutilisation de spécifications existantes (#include/import)

Exemple

```
module unService {  
    typedef unsigned long EntierPositif;  
    typedef sequence<Positif> desEntiersPositifs;  
  
    interface Premier {  
  
        boolean est_premier ( in EntierPositif nombre);  
        desEntiersPositifs nombres_premiers (in EntierPositif nombre);  
    };  
};  
  
module monApplication {  
    interface MonService {  
        unService::EntierPositif prochainPremier(..);  
    };  
};
```

module

définitions de type

interface

opérations

Exemple

interface
attribut

opérations

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

exception

opérations

Attribut IDL

Définition d'attribut

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

Equivaut en java à :

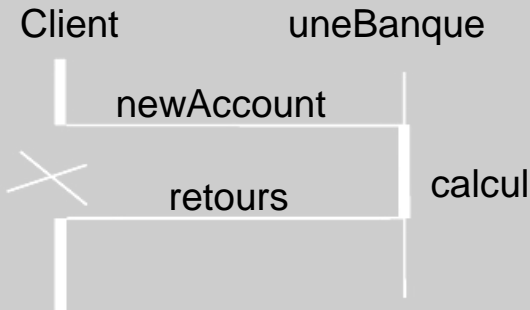
```
float balance();  
string description();  
void description(in string s);
```


Operations (1/2)

- ✓ Paramètres nommés et associés à un mode
- ✓ Opérations bloquantes par défaut

```
void op1 (in long input,  
         out long output,  
         inout long both);
```

```
interface account;  
  
interface bank {  
    account newAccount (in string name);  
  
void deleteAccount (in account old);  
};
```

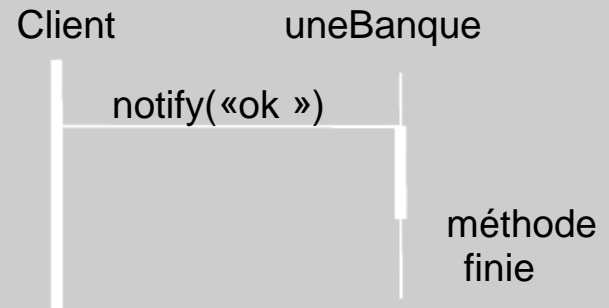


<uneOpération> ::= <modeInvocation> <typeRetour> <identificateur>
 ' (' <paramètres> ') ' <clausesExceptions><clausesContextes>

Opérations (2/2)

oneway void notify (in string message);

- ❑ Opération non bloquante
- ❑ Pas de paramètre de type out, inout ou d'exceptions
- ❑ Valeur de retour : void
- ❑ Pas d'exceptions déclenchées.



Exceptions

```
exception reject {string reason;};  
  account newAccount (in string name)  
    raises (reject);
```

```
exception DateErronnee {  
  String raison; };
```

CORBA::Exception

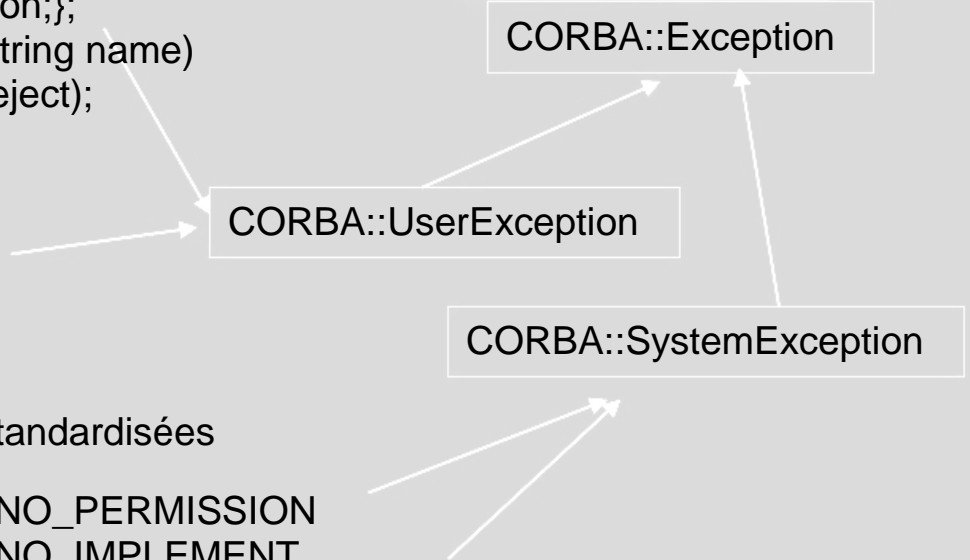
CORBA::UserException

CORBA::SystemException

Des exceptions CORBA standardisées

UNKNOWN
BAD_PARAM
COM_FAILURE
INV_OBJREF

NO_PERMISSION
NO_IMPLEMENT
OBJECT_NOT_EXIST
.....



Exceptions

```
exception reject {string reason;};  
account newAccount (in string name)  
  raises (reject);
```

```
exception DateErronnee {  
  String raison; };
```

CORBA::Exception

CORBA::UserException

CORBA::SystemException

Des exceptions CORBA standardisées

UNKNOWN
BAD_PARAM
COM_FAILURE
INV_OBJREF

NO_PERMISSION
NO_IMPLEMENT
OBJECT_NOT_EXIST
.....



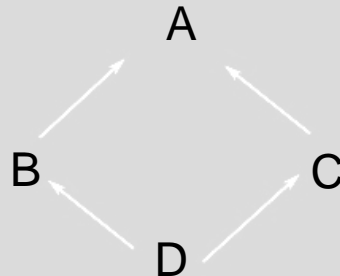
Gestion explicite de la part du client

Définitions circulaires

```
module Circulaire {  
    interface B;  
    interface A {  
        void utiliséB(in B unB);  
    }  
    interface B {  
        void utiliséA(in A unA);  
    }  
}
```

Héritage multiple

```
interface A { ... }  
interface B : A { ... }  
interface C : A { ... }  
interface D : B, C { ... }
```



Types de base et autres types

Types de base

- short
- long
- unsigned short
- unsigned long
- float
- double
- char
- boolean
- octet
-

Types construits

- struct
- union
- enum

```
struct Compte {  
    short ref;  
    string nom;  
    short somme;  
};
```

Types génériques

- array
- sequence
- string

MétaTypes

- any
- TypeCode

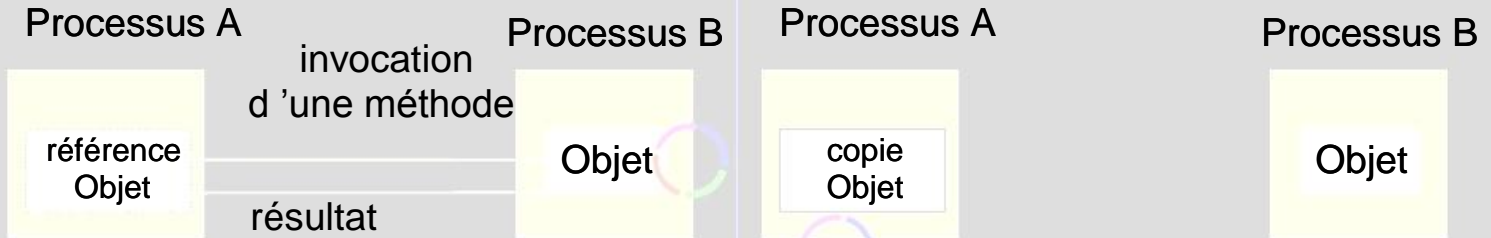
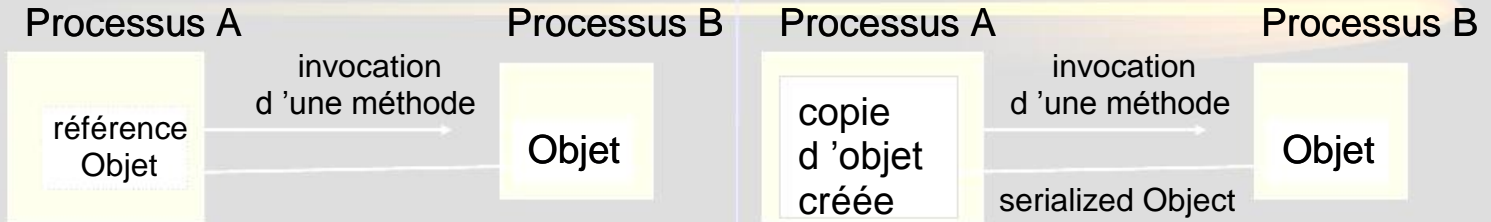
Types de données dynamiques et auto-descriptifs

Type Any

```
interface PileDeChaines {  
    readonly attribut string sommet;  
    void poser(in string valeur);  
    string retirer();  
};
```

```
interface PileGénérique {  
    readonly attribut Any sommet;  
    readonly attribut TypeCode typeDesValeurs;  
    void poser(in Any valeur);  
    Any retirer();  
};
```


Passage d'un objet par référence ou par valeur



Partager des données,
Contrôler les données

Encapsuler des données,
Travailler sur des copies

Passage par valeur : Value Type

```

valuetype EmployeeRecord { // note this is not a CORBA::Object
struct Address {
    string street;
    string state; ...}
// state definition
public string name;

public Address addr;

private string email;
private string SSN;
void display();
// initializer
factory init(in string name, in string SSN);
};

```

Interface GestionDuPersonnel

{ void memorize(in EmployeeRecord e) }

Implémentée localement

Mapping CORBA vers Java ou C++

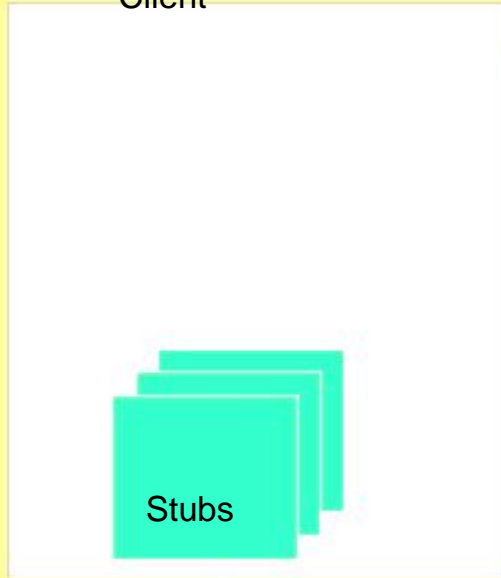
Projection vers un langage de programmation

Definitions
IDL



Précompilateur
IDL

Client



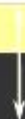
Serveur

Objet
implementation

Methodes

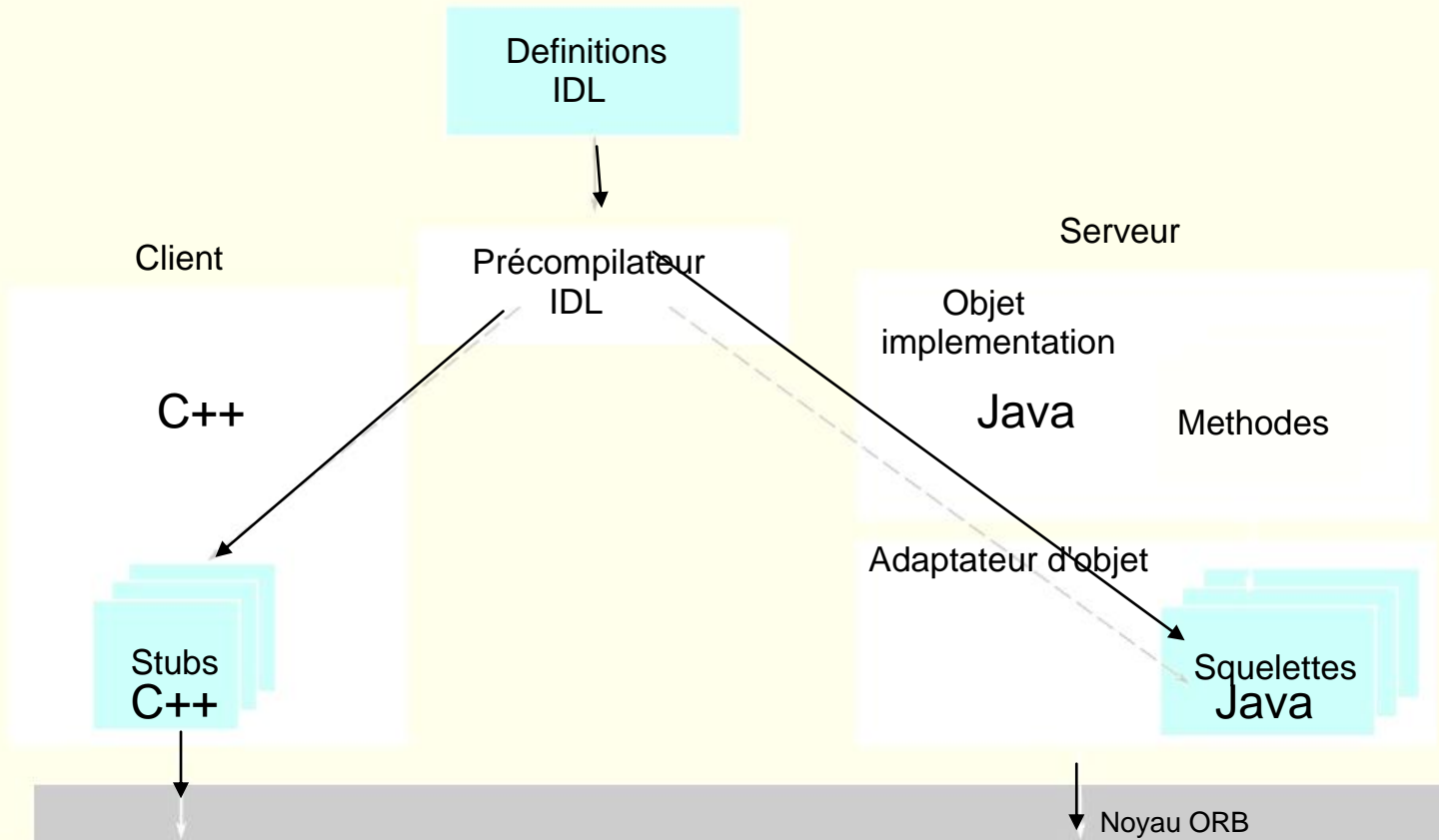
Adaptateur d'objet

Squelettes



Noyau ORB

Projection vers un langage de programmation



Mapping C++ des modules

- traduit en namespace C++
- les modules CORBA contiennent des types prédéfinis des interfaces et des opérations

```
// IDL
module M
{
// définitions
};
```



```
// C++
namespace M
{
// définitions
};
```

Mapping Java des modules

- traduit en package Java

```
// IDL
module M
{
// définitions
};
```



```
// Java
package M
{
// définitions
};
```

Projections de types élémentaires

OMG IDL	C++
short	CORBA::Short
long	CORBA::Long
long long	CORBA::LongLong
unsigned short	CORBA::UShort
unsigned long	CORBA::ULong
unsigned long long	CORBA::ULongLong
float	CORBA::Float
double	CORBA::Double
long double	CORBA::LongDouble
char	CORBA::Char
wchar	CORBA::WChar
boolean	CORBA::Boolean
octet	CORBA::Octet

IDL Type	Java type
boolean	boolean
char	char
wchar	char
octet	byte
string	java.lang.String
wstring	java.lang.String
short	short
unsigned short	short
long	int
unsigned long	int
long long	long
unsigned long long	long
float	float
double	double
fixed	java.math.BigDecimal

Projections en C++ et Java

- énumération : enum en C++ ; constantes encapsulées dans une classe dont le nom est celui du type énuméré en Java
- séquences: encapsulées dans une classe qui contrôle les débordements en C++; des tableaux dynamiques en Java
- tableaux : tels que.
- Interface : classe C++ interface Java

interface J:I {...}

class J : public virtual I

interface J extends I

- Opérations : méthodes
- Attributs : une ou deux méthodes d'accès

Projection en Java d'une énumération

IDL : enum Mois {Janvier, Fevrier,}

Java généré :

```
public final class Mois {  
    public static final int _Janvier = 0;  
    public static final Mois Janvier = new Mois(_Janvier);  
    public static final int _Fevrier = 1;  
    public int value() {...}  
    public static Mois from_int(int value) {...}
```

Utilisation :

```
Mois m = Mois.Janvier;
```

```
switch (m.value()) {  
    case Mois._Janvier : ...;}
```

```
m = Mois.from_int(7);  
if (m == Mois.Aout) { ...}
```

Passage de paramètres en java

- paramètres en out ou inout :
 - utilisation de Holder en Java

```
final public class XHolder {  
    { public X value;  
      public XHolder(){ }  
      public XHolder(X initial)  
        { value = initial; }  
}
```

```
final public class BooleanHolder {  
    { public Boolean value;  
      public BooleanHolder(){ }  
      public BooleanHolder(Boolean initial)  
        { value = initial; }  
}
```

Classes Holder pour les
types élémentaires

Passage de paramètres en java

```
// OMG IDL
interface Modes {
    long operation ( in long inArg, out long outArg, inout long inoutArg);
}
//Java
```

```
interface Modes {
    int operation(int inArg, org.omg.CORBA.IntHolder outArg,
                 org.omg.CORBA.IntHolder inoutArg);
}
```

//Utilisation

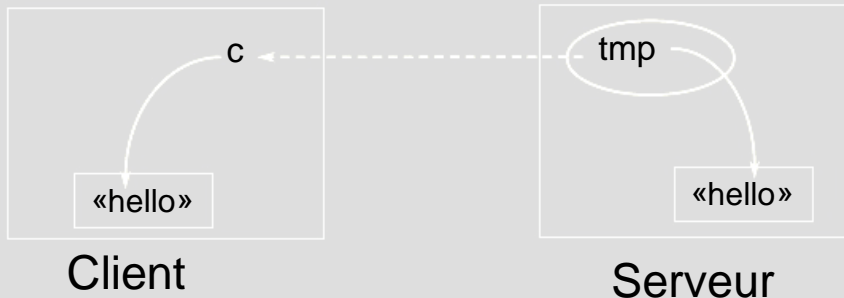
```
Modes m = ...;
IntHolder argSortie = new IntHolder(); IntHolder argES = new IntHolder(131);
int result = m.operation(57,argSortie,argES);
```

Gestion mémoire : pointeurs en C++

Les clients et les serveurs sont dans des processus différents.
Problèmes : Qui fait l'allocation de la mémoire ? (malloc, new)
Qui fait la désallocation de la mémoire ? (delete)

```
interface T {  
    string op1()  
};
```

```
char* T_i::op1() {  
    char* tmp = CORBA::string_alloc(6);  
    strcpy(tmp, «hello»);  
    return tmp;  
};
```



Client

Serveur

Mapping C++ des références objet

Interface

- traduit en classes C++
- traduit en deux types C++

- A_var

Référence sur un objet (instance de classe A)
avec gestion automatique de la mémoire.

- A_ptr

Similaire aux pointeurs C++.

```
interface A
{
    void op1 (in long param);
};
```



```
// C++-used
A_ptr aPtr;
A_var aVar;
aVar->op1(2);
aPtr->op1(5);
....
aVar._retn();
```

Utilisation à partir de C++(1)

```
interface Calendrier {
    attribute Annee anneeCourante;
    exception DateErronnee {String raison; };

    boolean verifierUneDate(in Date d);
    void leJourSuivant(inout Date d);
    Date convertirChaine (in String uneChaine) raises (DateErronnee);
    String convertirDate(in Date d);
    Calendrier_var monCalendrier = ... référence;
    Annee anneeATraiter = monCalendrier -> anneeCourante();
    monCalendrier -> anneeCourante(anneeATraiter +1);

    if (monCalendrier -> verifierDate (uneDate)) {
        monCalendrier -> leJourSuivant(date);
        CORBA::String chaine = monCalendrier -> convertirDate (uneDate);

    try { date = convertirChaine ("31 fevrier 2009 "); }
        catch (DateErronnee& exception) { cout << exception.raison};
```

IDL

Utilisation à partir de C++(2)

```
interface Calendrier { .... }
```

IDL

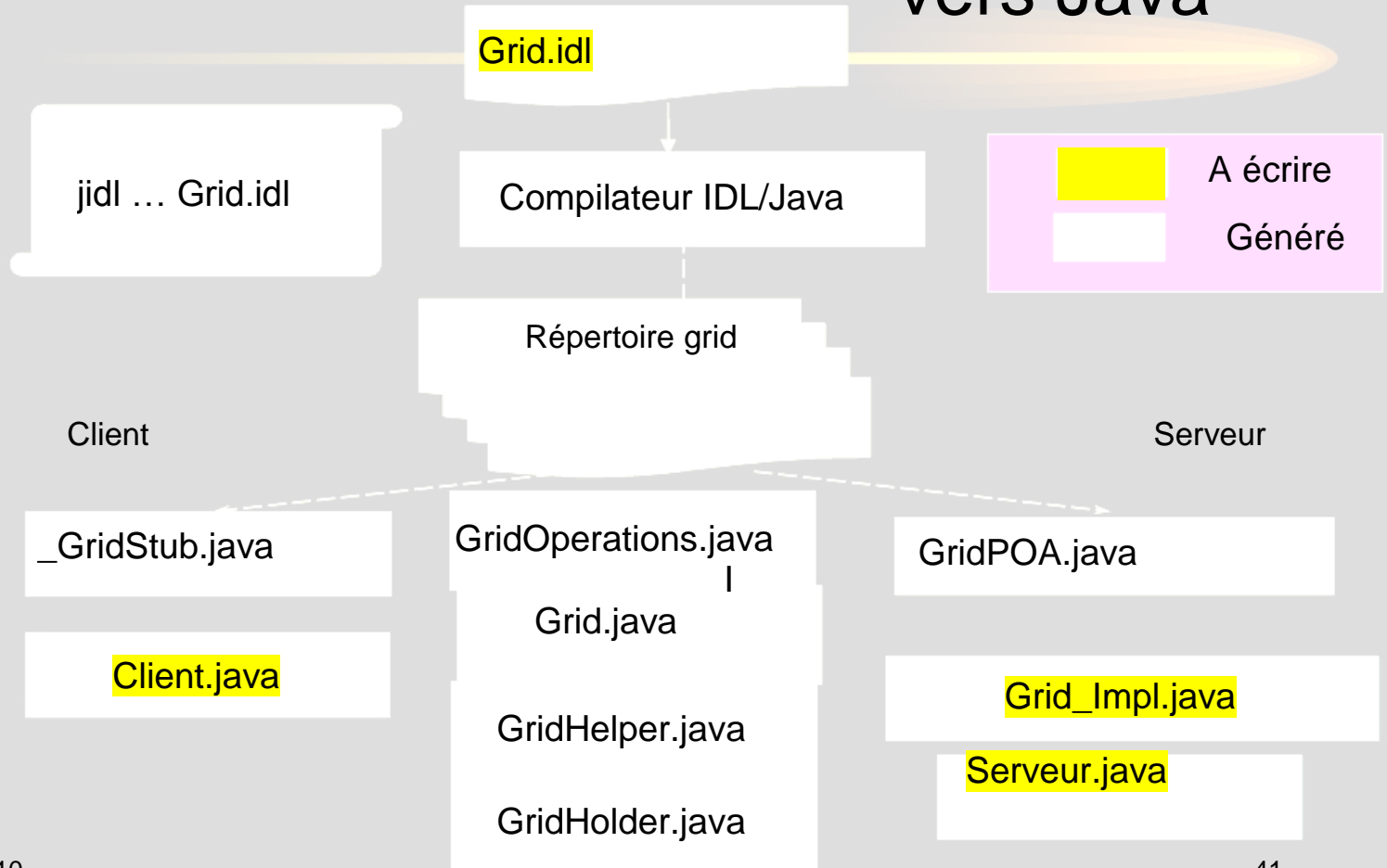
```
typedef sequence <Date> desDates;  
interface CalendrierFerie : Calendrier {  
    void lesJoursFeries (in Annee UneAnnee, out desDates dates);}
```

```
CalendrierFerie_var monCalendrierFerie =  
    CalendrierFerie::_narrow (tonCalendrier);
```

```
desDates datesRepos;  
monCalendrierFerie -> lesJoursFeries (2009, datesRepos);  
for (CORBA::Ulong i=0; i< datesRepos.length(); i++){  
    CORBA::String_var chaine;  
    chaine = monCalendrierFerie -> convertirChaine(datesRepos[i]); }
```


Processus de développement CORBA

Exemple Compilation interface IDL vers Java



Orbacus IDL-to-Java translator

`jidl [options] idl-files`

Options for `jidl`

`-h, --help`
`-v, --version`
`-e, --cpp NAME`
`-d, --debug`
`-DNAME`
`-DNAME=DEF`
`-UNAME`
`-IDIR`
`-E`
`--no-skeletons`
`--locality-constrained`
`--all`
`--tie`
`--file-list FILE`
`--no-local-copy`
`--case-sensitive`

These options are the same as for the `idl` command.

`--no-comments`

The default behavior of `jidl` is to add any comments from the IDL file starting with `/**` to the generated Java files. Specify this option if you don't want these comments added to your Java files.

`--package PKG`

Specifies a package name for the generated Java classes. Each class will be generated relative to this package.

`--prefix-package PRE PKG`

Specifies a package name for a particular prefix 1. Each class with this prefix will be generated relative to the specified package.

`--auto-package`

Derives the package names for generated Java classes from the IDL prefixes. The prefix `ooc.com`, for example, results in the package `com.ooc`.

`--output-dir DIR`

Specifies a directory where `jidl` will place the generated Java files. Without this option the current directory is used.

`--clone`

Generates a `clone` method for struct, union, enum, exception, valuetype and abstract interface types. For valuetypes, only an abstract method is generated. The valuetype implementer must supply an implementation for `clone`.

`--impl`

Generates example servant implementation classes. For IDL interface types, a class is generated in the same package as the interface classes, having the same name as the interface with the suffix `_impl`. The generated class extends the POA class of the interface. For IDL valuetypes, a class is generated in the same package as the valuetype with the suffix `ValueFactory_impl`. You must not use `--no-skeletons` in combination with this option.

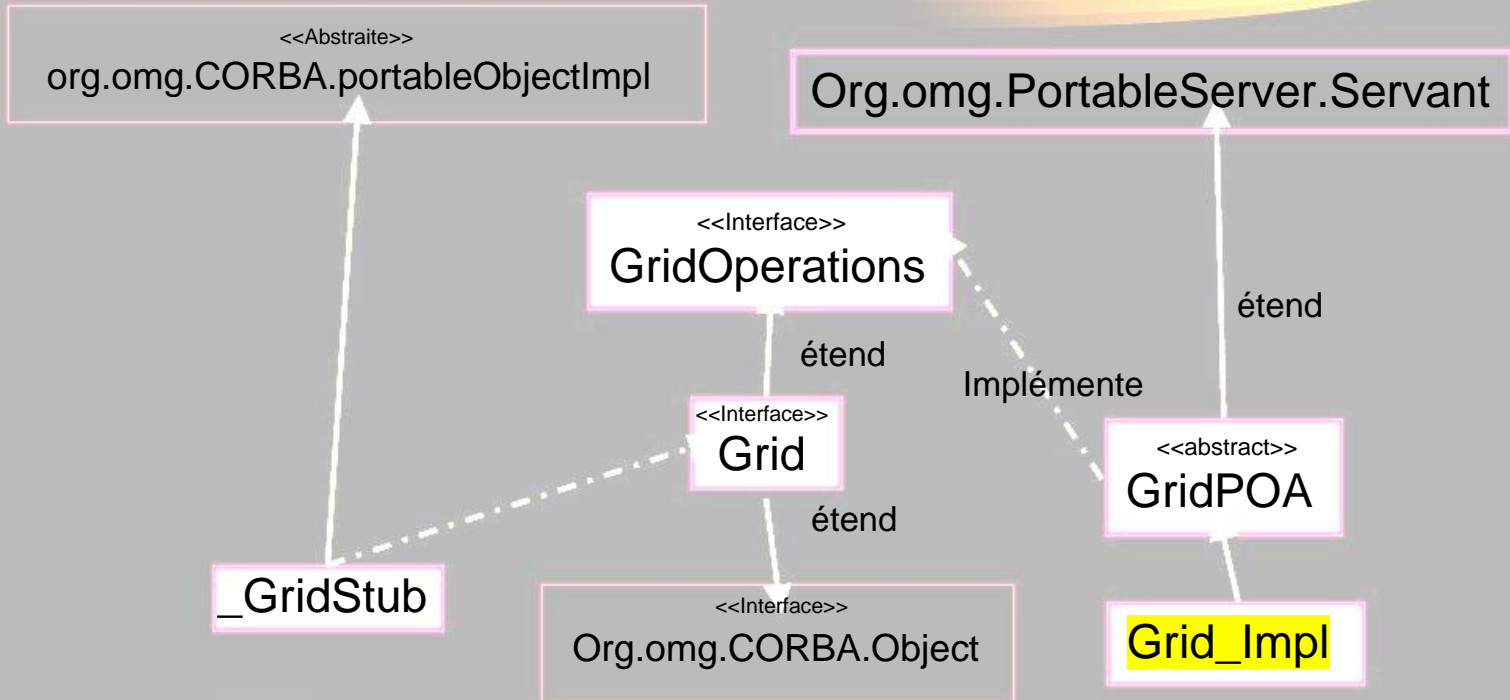
`--impl-tie`

Similar to `--impl`, but implementation classes for interfaces implement the `Operations` interface to facilitate the use of TIE classes. You must not use `--no-skeletons` in combination with this option.

`--with-interceptors-args`

Generate code with support for arguments, result and exception list values for interceptors. Note that use of this option will generate proprietary stubs and skeletons which are not compatible with ORBs from other vendors.

Hiérarchie Java des classes



Implémentation de l'interface (Classes servants)

Ecrire une classe Java qui hérite de la classe GridPOA

```
public class Grid_impl extends GridPOA {
    public int [][] valeur;
    public Grid_impl(short width, short height) {
        valeur = new int [width] [height]; .....}
    public int get(short n, short m) {
        return valeur[n][m]; }
    public short height(){
        return (short) valeur[1].length;}

    public void copyIn(GridHolder g)
    .....
        g.value.set(i,j,this.valeur[i][j]) ;

    ...}
```

Implémentation du serveur (1)

1. Initialiser le bus CORBA
 - obtenir l'objet ORB
2. Initialiser l'adaptateur d'objets
 - obtenir le POA
3. Créer les implantations d'objets
4. Enregistrer les implantations par l'adaptateur
5. Diffuser leurs références
 - afficher une chaîne codifiant l'IOR
6. Attendre des requêtes venant du bus
7. Destruction du Bus

Implémentation du serveur (2)

```
public class ServeurDeGrilles{
public static void main(String args[]) throws Exception
{ // Propriétés nécessaires à l'utilisation d'ORBACUS ORB avec JDK 1.2 ou +.
  java.util.Properties props = System.getProperties(); props.put("org.omg.CORBA.ORBClass", "com.ooc.CORBA.ORB"); props.put("org.omg.CORBA.ORBSingletonClass", "com.ooc.CORBA.ORBSingleton");
  // Initialisation du bus CORBA pour un processus serveur.
int status = 0;
org.omg.CORBA.ORB orb = null;
try
{ orb = org.omg.CORBA.ORB.init(args, props); (1)
  status = run(orb); (2,3,4,5,6)
} catch(Exception ex)
  { ex.printStackTrace(); status = 1; }
if(orb != null)
{ try {
  orb.destroy();} (7)
  catch(Exception ex)
  { ..... ex.printStackTrace();
```

Implémentation du serveur (3)

```
public class ServeurDeGrilles{ ...
static int run(org.omg.CORBA.ORB orb) throws org.omg.CORBA.UserException
{ org.omg.PortableServer.POA rootPOA =
    org.omg.PortableServer.POAHelper.narrow(
        orb.resolve_initial_references("RootPOA")); (2)
org.omg.PortableServer.POAManager manager = rootPOA.the_POAManager();
Grid_Impl grilleImpl = new Grid_Impl(100,100); // (3) « création de servants »
Grid Grille = Grid_Impl._this(orb); (4) « activation de servants »
try
{ // Obtenir sous forme textuelle l'IOR de l'objet.
  String chainelOR = orb.object_to_string (grille); // (5)
  // . . . diffuser la chaîne . .
  System.out.println("IOR : " + chainelOR);
  // Mettre le serveur en attente des requêtes venant du bus CORBA.
  manager.activate();
  orb.run(); // (6)
}
```


Implémentation du client

1. Initialiser le bus (objet ORB)
2. Créer les souches des objets à utiliser
 - 2.a. obtenir les références d'objet (IOR)
 - 2.b. convertir vers les types nécessaires
 - narrow contrôle le typage à travers le réseau
3. Réaliser les traitements

Implémentation du client en Java

```
public class ClientDeGrille {
    public static void main(java.lang.String[] args) {
        //..... Comme le serveur avec
        status = run(orb, args[0]); //IOR est ici passé en argument
    }
    ...

    static int run(org.omg.CORBA.ORB orb, String ref)
    { try
        { org.omg.CORBA.Object obj = orb.string_to_object(ref); // (2.a)
          Grid grille = GridHelper.narrow(obj); // (2.b)
          System.out.println(grille.height());
          System.out.println(grille.width());
          grille.set((short) 2, (short) 4, 123);
          System.out.println("grid[2,4] is " + grille.get((short)2, (short)4));
        }
    }
}
```

Références et strings

Vous pouvez convertir toute référence en une string et inversement pour par exemple:

- le serveur crée un objet et écrit sa référence dans un fichier

```
String ref = orb.object_to_string(grille);
String refFile = « Grid.ref » ;
FileOutputStream file = new FileOutputStream(refFile);
PrintWriter out = new PrintWriter(file);
out.println(ref); out.flush(); file.close();
```

- Le client lit le fichier et utilise la référence pour accéder à l'objet.

```
FileInputStream file = new FileInputStream(refFile);
string ref = new BufferedReader ( new
                                InputStreamReader(file)).readLine();
file.close();
org.omg.CORBA.Object obj = orb.string_to_object(ref);
```

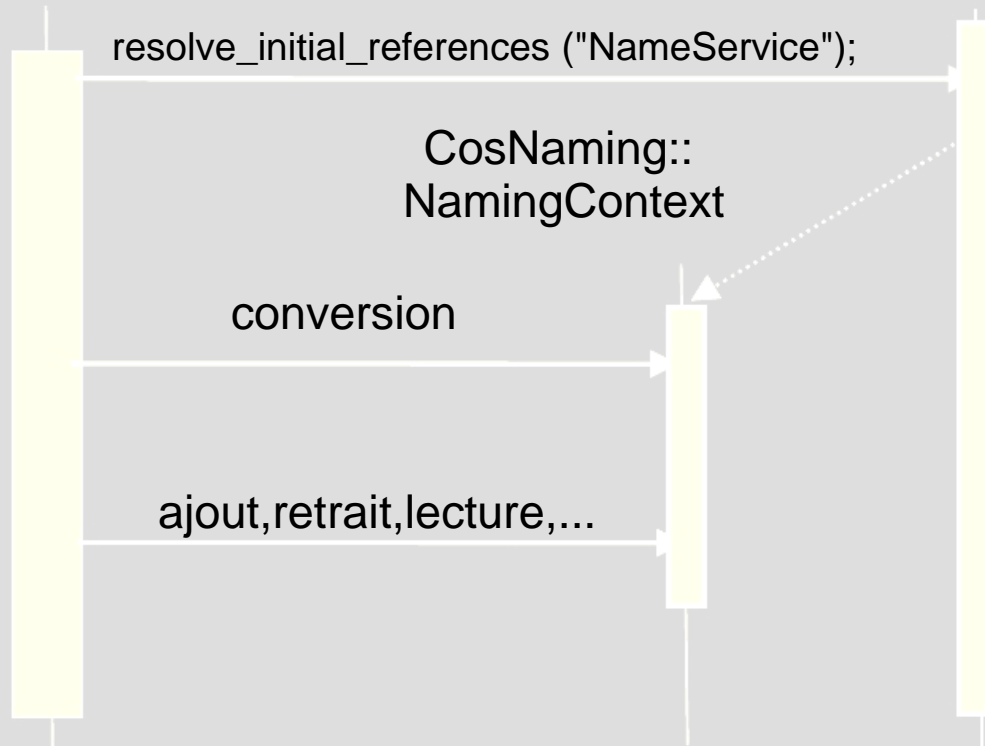
- Stockage dans une BD

=>Service de nommage

Scénario d'obtention de la référence du service de nommage

Client ou Serveur

ORB

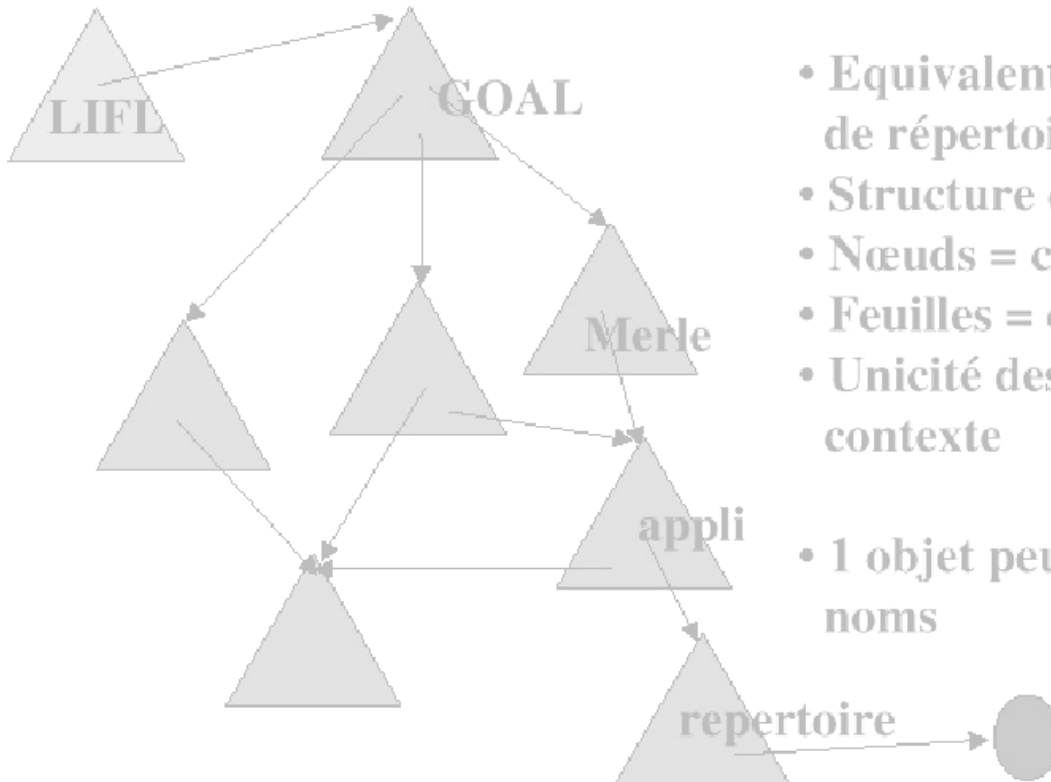


Obtenir le service de Nommage en Java

```
import org.omg.CosNaming.*;
...
//retrouver la référence de l'objet notoire « NameService »
org.omg.CORBA.Object objRef = null;
try {
    objRef = orb.resolve_initial_references ("NameService");
} catch( org.omg.CORBA.ORBPackage.InvalidName e ) {
    outils.ARRET ("Le service initial NameService est inconnu");
}

//la convertir en une référence à un objet
//de type CosNaming::NamingContext
NamingContext nsRef = NamingContextHelper.narrow(objRef);
if ( nsRef == null ) {
    outils.ARRET ("Le service initial 'NameService' n'est pas
                un objet CosNaming::NamingContext");
}
```

Notion de chemin d'accès



- Equivalent à la notion de répertoire/sous répertoire
- Structure de graphe
- Nœuds = contextes
- Feuilles = objets
- Unicité des noms dans un contexte
- 1 objet peut avoir plusieurs noms



Créer un nom/chemin en Java

```
import org.omg.CosNaming.*;  
//créer un chemin simple  
NameComponent[] nsNom = new NameComponent [1];  
nsNom[0] = new NameComponent( "grilleA ", "");
```

```
//créer un chemin composé  
NameComponent[] nsNom = new NameComponent [2];  
nsNom[0] = new NameComponent( "appli", "");  
nsNom[1] = new NameComponent("grille ", "");
```

Enregistrer un objet

- Opération pour publier un Objet
 - en général, opération réalisée par le serveur
- Scénario Type
 1. Créer un objet
 2. Construire un chemin d'accès (Name)
 3. Appeler l'opération « bind » ou « rebind » avec le chemin et la référence de l'objet

```
void bind (in Name n, in Object obj)  
  raises (NotFound, CannotProceed, InvalidName, AlreadyBound);
```


Enregistrer un objet en Java

```
//créer un chemin
NameComponent[] nsNom = new NameComponent [1];
nsNom[0] = new NameComponent("MONOBJET", "");
//enregistrer l 'objet
try {
    nsRef.bind (nsNom, uneRefObjet);
} catch (org.omg.CosNaming.NamingContextPackage.NotFound enf) {
    ...
} catch(org.omg.CosNaming.NamingContextPackage.AlreadyBound eab){
    ...
} catch(org.omg.CosNaming.NamingContextPackage.CannotProceed
ecp){
    ...
} catch(org.omg.CosNaming.NamingContextPackage.InvalidName ein) {
    ...
}
```

Retrouver un objet

- Opération réalisée par un client ou un serveur
- Scénario type :
 - construire un chemin d'accès (Name)
 - appeler l'opération « resolve » avec le chemin
 - convertir la référence obtenue dans le bon type

Object resolve (in Name n)
raises (NotFound, CannotProceed, InvalidName)

Retrouver un objet en Java

```
//créer un chemin
NameComponent[] nsNom = new NameComponent [1];
nsNom[0] = new NameComponent("MONOBJET","");
    //retrouver l 'objet
org.omg.CORBA.Object objRef = null;
try {
objRef = nsRef.resolve (nsNom);
} catch (org.omg.CosNaming.NamingContextPackage.NotFound enf) {
...
} catch(org.omg.CosNaming.NamingContextPackage.CannotProceed ecp)
...
} catch (org.omg.CosNaming.NamingContextPackage.InvalidName ein)
...
}
    //convertir la référence
Grille uneRefObjet = GrilleHelper.narrow (objRef);
```

Éléments complémentaires

- Tie : réutilisation de code et héritage
- Valeur
- Embarqué



OMG's CORBA/e Middleware Specification for Distributed Real-time & Embedded Systems
First two adopted profiles, CORBA/e Compact Profile and CORBA/e Micro Profile, target resource-constrained embedded systems

Le cycle de vie des objets

- Problème
 - actuellement, 1 grille = 1 serveur
 - pas de création/destruction d'objets à distance
 - seulement invocation d'opérations
- Solution
 - notion de fabrique d'objets
 - exprimée en OMG-IDL
- C'est un canevas de conception : Design pattern
 - voir aussi le service LifeCycle

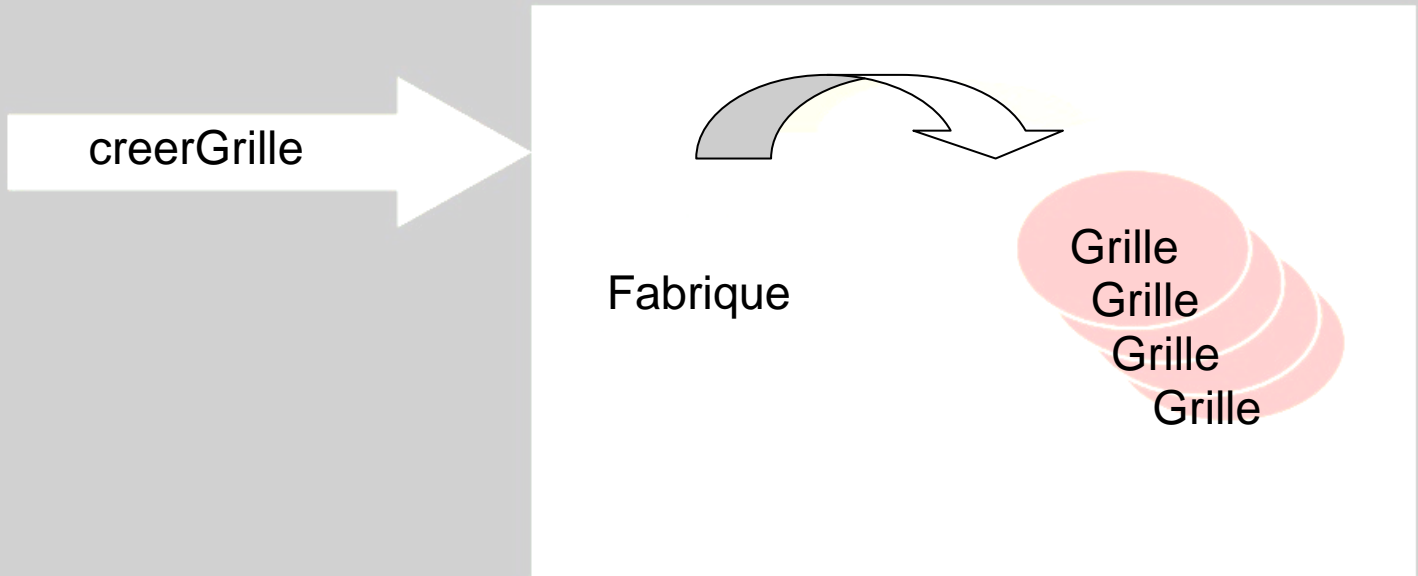
Le cycle de vie des objets

- Problème
 - actuellement, 1 grille = 1 serveur
 - pas de création/destruction d'objets à distance
 - seulement invocation d'opérations
- Solution
 - notion de fabrique d'objets
 - exprimée en OMG-IDL
- C'est un canevas de conception : Design pattern
 - voir aussi le service LifeCycle

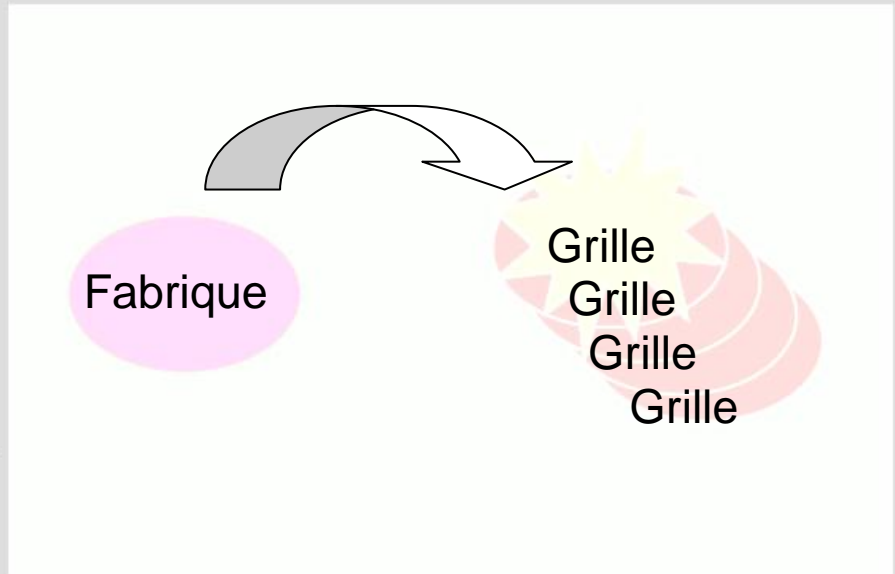
Autres usages de la fabrique :

- gestion de droits, load-balancing, polymorphisme, ...

L'implantation de la fabrique



L'implantation de la fabrique



destruireGrille

Interface IDL d 'une fabrique de Grilles

```
module grilles {  
  ...  
  interface Fabrique {  
    Grid newGrid(in short width,in short height);  
  }; };
```

Une application d'administration de la fabrique

- Création d'une nouvelle grille et mise à disposition par le service Nommage
 - 1. Initialiser le bus CORBA
 - 2. Obtenir le service Nommage (NS)
 - 3. Obtenir la fabrique depuis le NS
 - 4. Créer un répertoire
 - 5. Enregistrer le répertoire dans le NS

Serveur Java de la fabrique

```
import org.omg.CORBA.*; // le bus CORBA.
import org.omg.CosNaming.*; // le service Nommage.
public class ServeurFabrique {
public static void main(String args[]) throws Exception
{ ... (1) (2) status = run(orb);
}
```

```
static int run(org.omg.CORBA.ORB orb)...
{FabriqueImpl fabrique = new FabriqueImpl(); // (3)
  NamingContext nc = NamingContextHelper.narrow( // (4)
orb.resolve_initial_references ("NameService"));
  NameComponent[] nom = new NameComponent[1];
  nom[0] = new NameComponent("FABRIQUE", "");
  nc.rebind(nom,fabrique); // (5)
  ...; // (6)
}
```

L 'application d 'administration en Java

```
public class Administration {  
public static void main(String args[]) throws Exception {  
    ORB orb = ORB.init (args,null); // (1)  
    NamingContext nc = NamingContextHelper.narrow( // (2)  
        orb.resolve_initial_references ("NameService"));  
    NameComponent[] nsNom = new NameComponent [1];  
    nsNom[0] = new NameComponent("FABRIQUE", "");  
    org.omg.CORBA.Object objet=nc.resolve(nsNom); //(3.a)  
    Fabrique fabrique=FabriqueHelper.narrow(objet);//(3.b)  
    Grid g = fabrique.newGrid( 4,4); // (4)  
    nsNom[0] = new NameComponent( "Grille4x4", "");  
        nc.bind (nsNom, g); // (5)  
}
```

CORBA vs Web Services

	Corba	WebServices
data Model	Object Model	Soap Messages
Client-Server coupling	fort	faible
Location	Object reference	url
Type system	IDL/static & dyn	XML Schema/dyn
Passage de paramètres	Par référence /valeur	Par valeur
Error Handling	IDL Exception	Soap fault message
Protocoles	GIOP/IIOP	HTTP

Que reste-t-il à savoir ?

- Plus sur la génération de stubs, les possibilités Corba (DII, Activation d'objets...)
- Le service d'événements Corba

- En TP & autres cours
 - Plus sur le service de nommage Corba
 - Interopérabilité et JNDI
 - Du service d'événements aux MOM
 - Un exemple de MOM JMS

