CORBA : Exemple d'application

repris de : J.-M. Geib, C. Gransart et Ph. Merle
http://corbaweb.lifl.fr/CORBA_des_concepts_a_la_pratique/slides_Annuaire.pdf

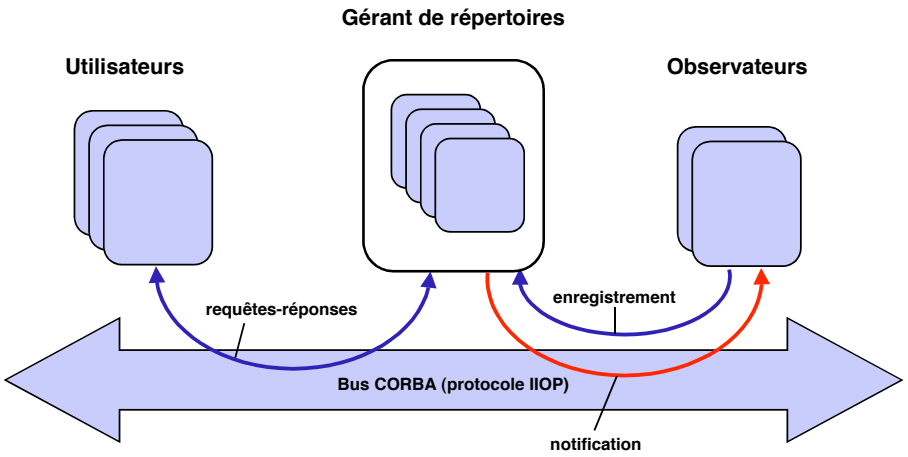
Sacha Krakowiak
Université Joseph Fourier
Projet Sardes (INRIA et IMAG-LSR)

http://sardes.inrialpes.fr/~krakowia

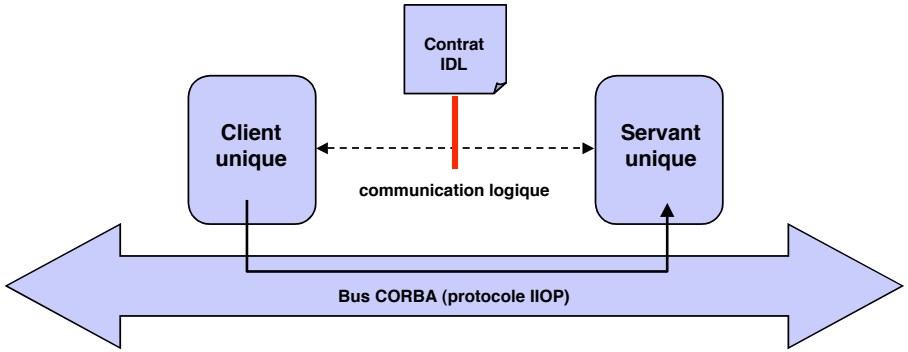
Une application simple : Annuaire

- **Fonctions**
  - Annuaire = répertoire gérant une liste de personnes avec des propriétés
  - Associe un nom de personne à un ensemble de propriétés (exemple : adresse, numéro de téléphone, e-mail, etc.)
- **Utilisation**
  - On peut créer plusieurs répertoires, chacun désigné par un nom (libellé)
  - On peut utiliser un répertoire via son interface : ajouter, supprimer, modifier, obtenir, lister des personnes
  - Utilisation avancée : observation
    - permet à un observateur extérieur d'être avisé de certaines modifications portant sur un répertoire donné

L'application Annuaire



Première étape : application client-serveur



Implémentation du client et du servant en langage Java

Le contrat OMG IDL (1/2)

Fichier *annuaire.idl*

```
// contrat OMG IDL de l'application Annuaire
//
# include <date.idl>                        // Réutilisation du service de dates
#pragma prefix = « lifl.fr »                // Organisation auteur
module annuaire {
        typedef string Nom;                 // Nom d'une personne
        typedef sequence<Nom> DesNoms;      // Ensemble de noms
        struct Personne {                   // Description d'une personne
                Nom nom;                    //   - son nom
                string informations;        //   - données diverses
                string telephone;           //   - numéro téléphone
                string email;               //   - adresse éléctronique
                string url;                 //   - adresse web
                ::date::Date date_naissance;//   - date de naissance
        };
```


Le contrat OMG IDL (2/2)

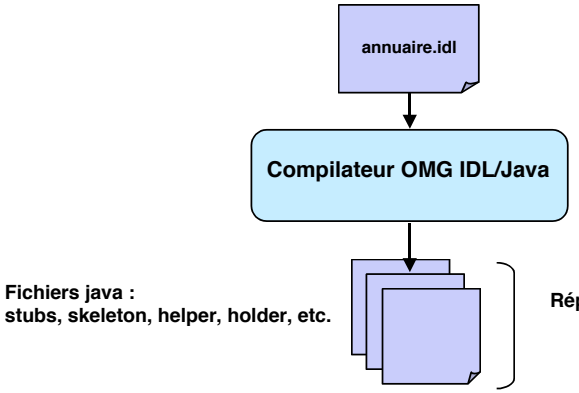
```
    interface Repertoire {
        readonly attribute string libelle;          // le libellé (nom) du répertoire

        exception ExisteDeja (Nom nom; );
        exception Inconnu (Nom nom; );

        void ajouterPersonne (in Personne personne)
                                        raises ExisteDeja ;
        void retirerPersonne(in Nom nom)
                                        raises Inconnu ;
        void modifierPersonne(in Nom nom, in Personne personne)
                                        raises Inconnu ;
        Personne obtenirPersonne (in Nom nom)
                                        raises Inconnu ;
        DesNoms listerNoms ();
    };                                          // fin interface
};                                              // fin module
```


Commentaires sur le contrat OMG IDL initial

- **Réutilisation de spécifications OMG IDL : *date***
- **Regroupement de définitions communes : *annuaire***
- **Définition des concepts manipulés (typedef) : *Nom, Personne***
- **Définition des données échangées : *DesNoms***
- **Définition de l'interface des objets : *Répertoire***
  - **Opérations : ajouterPersonne**
  - **Exceptions : *ExisteD eja***
  - **Attributs (propriétés) : *libelle***

La projection IDL -> Java du contrat *annuaire.idl*

unix> idl2java annuaire.idl



unix> javac fr/lifl/annuaire/*.java

La projection Java du contrat *annuaire.idl*

- Le module *annuaire*
  - => le package *fr.lifl.annuaire*
  - L'alias `Nom`
    - La classe utilitaire `NomHelper`
  - L'alias `DesNoms`
    - La classe `DesNomsHolder` pour le passage out et inout
    - La classe utilitaire `DesNomHelper`
  - La structure `Personne`
    - La classe `Personne`
    - La classe `PersonneHolder` pour le passage out et inout
    - La classe utilitaire `PersonneHelper`

L'interface Java `Repertoire`

```
package fr.lifl.annuaire;
    public interface Repertoire extends org.omg.CORBA.Object {
    public String libelle ();              // le libellé (nom) du répertoire

     public void ajouterPersonne (Personne personne)
                        throws fr.lifl.annuaire.RepertoirePackage.ExisteDeja ;
     public void retirerPersonne(String nom)
                         throws fr.lifl.annuaire.RepertoirePackage.Inconnu ;
     public void modifierPersonne (String Nom nom, Personne personne)
                        throws fr.lifl.annuaire.RepertoirePackage.Inconnu ;
     public Personne obtenirPersonne (String nom)
                        throws fr.lifl.annuaire.RepertoirePackage.Inconnu ;
     public String() listerNoms ();
};
```


Autres classes pour la projection de l'interface `Repertoire`

- La classe `RepertoireHolder`
  - Pour le passage out et inout
- La classe utilitaire `RepertoireHelper`
  - Avec l'opérateur `narrow`
- La classe `_RepertoireStub`
  - Le talon client
- La classe `_RepertoireImplBase`
  - Le squelette pour l'implantation
- Le module `RepertoirePackage`
  - Projection des exceptions `ExisteDeja` et `Inconnu`

Ce qui sera manipulé en Java

- La classe `fr.lifl.annuaire.Personne`
- L'interface `fr.lifl.annuaire.Repertoire`
- L'opérateur `narrow` de `fr.lifl.annuaire.RepertoireHelper`
- Les classes pour les exceptions
  - `fr.lifl.annuaire.RepertoirePackage.ExisteDeja`
  - `fr.lifl.annuaire.RepertoirePackage.Inconnu`

L'implantation Java de l'interface `annuaire::Repertoire`

- **Importe les modules de la projection vers Java**
  - `fr.lifl.annuaire`
  - `fr.lifl.annuaire.RepertoirePackage`
- **Hérite de la classe squelette Java**
  - `fr.lifl.annuaire._RepertoireImplBase`
- **Définit la structure interne des objets de la classe**
  - **Le libellé et la liste des personnes**
- **Implante un constructeur Java pour la classe**
- **Implante les attributs et opérations**
  - **Respecte les sighnatures définies par le squelette**

La classe d'implantation Java `RepertoireImpl.java` (1/4)

```
// importation de la projection IDL/Java

import fr.lifl.annuaire.*;                          // le module annuaire
import fr.lifl.RepertoirePackage ;          // les exceptions

public class RepertoireImpl
                extends fr.lifl.annuaire._RepertoireImplBase {

        protected String le_libelle;                // le libellé (nom) du répertoire
        protected java.util.Hashtable personnes ;

        // le constructeur
        public RepertoireImpl (String le_libelle) {
                this.le_libelle =  le_libelle ;
                this.personnes = new java.util.Hashtable();
                }
// à suivre …
```


La classe d'implantation Java `RepertoireImpl.java` (2/4)

```
//OMG IDL : readonly attribute string libelle

public String libelle() {
        return this.le_libelle;
        }

//OMG IDL : void ajouterPersonne (in Personne personne)
//                                                raises ExisteDeja ;

public void ajouterPersonne (Personne personne)
                                                throws ExisteDeja {
        if (this.personnes.containsKey(personne.nom))
                throw new ExisteDeja(personne.nom) ;
        this.personnes.put(personne.nom, personne);
        }

// à suivre …
```


La classe d'implantation Java `RepertoireImpl.java` (3/4)

```
// OMG IDL : void retirerPersonne (in Nom nom) raises inconnu ;

public void retirerPersonne (String nom) throws Inconnu {
        if (this.personnes.remove(nom)  == null)
                throw new Inconnu (nom);
        }

// OMG IDL : void modifierPersonne (in Nom nom,  in Personne personne)
//                raises inconnu ;

public void modifierPersonne (String nom, Personne personne)
                throws Inconnu {
        if (this.personnes.remove(nom)  == null)
                throw new Inconnu (nom);
        this.personnes.put(personne.nom, personne);
        }
// à suivre …
```

La classe d'implantation Java `RepertoireImpl.java` (4/4)

```
// OMG IDL : Personne obtenirPersonne (in Nom nom) raises inconnu ;
public Personne obtenirPersonne (String nom)
                        throws fr.lifl.annuaire.RepertoirePackage.Inconnu {
        Personne resultat = (Personne) this.personnes.get(nom);
        if (resultat == null)
                throw new Inconnu(nom);
        return resultat;
        }
// OMG IDL : DesNoms  listerNoms () ;
public String[ ]  listerNoms () {
        int i = 0;
        String[ ] resultat = new String[this.personnes.size()];
        for java.util.Enumeration e = this.personnes.keys();
                                        e.hasMoreElements(), i++) {
                resultat[i] = (String) e.nextElement(); }
        return resultat;
        }
}                                                       // fin classe
```


Commentaires sur l'implantation Java

- **L'implantation Java d'une interface OMG IDL doit respecter les signatures (paramètres et exceptions) définies par la projection**
  - **Héritage de la classe squelette et, transitivement, de l'interface Java**
- **L'implantation est écrite en Java "classique"**
  - **Les structures de données (`java.util.*`)**
  - **Les exceptions sont levées classiquement**
  - **Utilisation possible avec packages plus évolués**
    - **Ex : JDBC pour stocker les infos sur personnes dans une BD**

Le scénario d'un serveur CORBA

- **Initialiser le bus CORBA**
  - **Obtenir l'objet ORB**
- **Initialiser l'adaptateur d'objets**
  - **Obtenir le POA**
- **Créer les implantations d'objets**
- **Enregistrer les implantations par l'adaptateur**
  - **Explicite en Java, C++, etc., implicite en CorbaScript**
- **Diffuser leurs références**
  - **Afficher une chaîne codifiant l'IOR**
- **Attendre des requêtes venant du bus**

Le serveur Java d'un répertoire

```
public class ServeurSimple {
        public static void main(String[ ] args) throws Exception {
        // initialisation du bus CORBA pour un processus serveur
        // création des objets ORB et BOA
        org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init (args.null);
        org.omg.CORBA.BOA boa = orb.BOA_init (args.null);
        // (en pratique on utilise un POA)
        // création de l'objet Répertoire
        RepertoireImpl repertoire = new RepertoireImpl("LIFL");
         // obtenir sous forme textuelle l'IOR de l'objet
        String chaineIOR = orb.object_to_string (repertoire);
        // diffuser la chaîne …
        // mettre le serveur en attente des requêtes venant du bus CORBA
        boa.impl_is_ready(null);
        }
}
```

Le scénario d'un client CORBA

- **Initialiser le bus (objet *orb*)**
- **Créer les talons (*stubs*) des objets à utiliser**
  - **Obtenir les références d'objets (IOR)**
  - **Convertir vers les types nécesaires**
  - **L'opération *narrow* (réalisée dans *Helper*) contrôle le typage à travers le réseau**
- **Réaliser les traitements**
  - **Exécuter le programme du *main* client**

Le client Java d'un répertoire

```
public class ClientSimple {
    public static void main(String[ ] args) throws Exception {
        // initialisation du bus CORBA pour un processus client
        // création de l'objet ORB
        org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init (args.null);
        // instanciation d'un talon java Répertoire
         org.omg.CORBA.Object objet =
                        orb.string_to_object(args[0]);
        // on suppose qu'on a obtenu sous forme textuelle l'IOR de l'objet
        // et qu'on l'a passé en premier paramètre (args[0])
        fr.lifl.annuaire.Repertoire repertoire =
                fr.lifl.annuaire.RepertoireHelper.narrow(objet) ;
         // narrow permet de passer du type Object au type réel de l'objet

        traitement(repertoire);              // traitement spécifique, cf plus loin
        }
}
```


Le traitement Java sur un répertoire (1/2)

```
static void traitement (fr.lifl.annuaire.Repertoire repertoire) throws Exception {
        String libelle = repertoire.libelle();
        System.out.println("Libellé du répertoire : " + libelle);
         (fr.lifl.annuaire.Personne personne =
                                new fr.lifl.annuaire.Personne ();
        personne.nom = "Merle Philippe" ;
        personne.informations = "chercheur" ;
        …
        try {
                repertoire.ajouterPersonne(personne);
            } catch (fr.lifl.annuaire.RepertoirePackage.ExisteDeja ed)  {
                System.out.println("attention, Merle Philippe déjà ajouté");
        }

        // à suivre
```


Le traitement Java sur un répertoire (2/2)

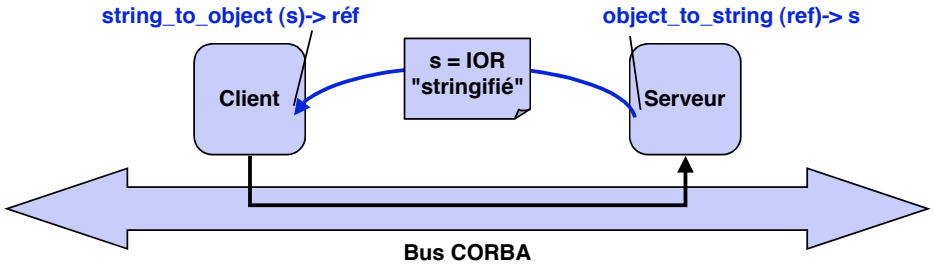
```
        personne.url = ... ;
        repertoire.modifierPersonne ("Merle Philippe", personne);
        String noms  = repertoire.listerNoms();
        for (int i =0; i<noms.length; i++) {
                personne = repertoire.obtenirPersonne(noms[i]);
                System.out.println("Nom : " + personne.nom);
                … afficher les autres champs …
                }
        try {
                repertoire.retirerPersonne(("Merle Philippe");
         } catch(fr.lifl.annuaire.RepertoirePackage.Inconnu e) {
                System.err.println("personne non présente dans le répertoire"),
        }
    }                                                   // fin traitement
}                                                       // fin classe
```

Commentaires sur le client Java du répertoire

- Appel du serveur à travers les talons OMG IDL
  - Package
- L'appel se déroule comme s'il était local
  - Interface `fr.lifl.annuaire.Repertoire`
  - Notation classique d'appel de méthodes
  - Passage par valeur pour le mode `in`
  - Toutes les entités sont des objets (`Personne`, etc)
- Gestion des exceptions "à la Java"

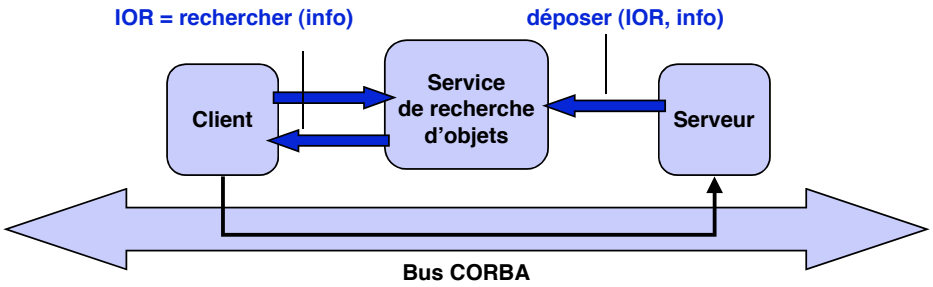
Commentaire sur la transmission des références

- Le mode de transmission de la référence de l'objet `repertoire` entre client et serveur est très primitif
  - Transmission d'une référence IOR sous forme de chaîne
  - Le moyen de transmission n'est pas spécifié (non traité par le bus CORBA) : mail, fichier partagé, courrier, téléphone, etc.



Une meilleure solution pour la transmission des références

- Services de recherche d'objets
  - "pages blanches" : service de nommage (*Naming*)
  - "pages jaunes" : service de courtage (*Trading*)



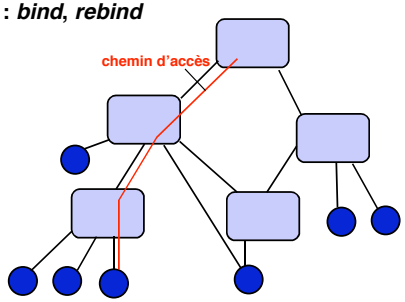
Le service de nommage ("pages blanches")

- Permet le partage de références entre applications CORBA
- Fournit un espace de désignation sysmbolique des références d'objets CORBA
  - Association de noms symboliques à des IOR
  - Graphes de répertoires et chemins d'accès (comme dans un SGF)
- Ce service est décrit en OMG IDL
  - Service `CosNaming` ("`Cos`" = préfixe commun aux *Common Object Services* de CORBA)

L'interface OMG IDL du service de nommage

- **Le module `CosNaming` définit**
  - Des associations noms-références d'objets
  - Des graphes de répertoires (`NamingContext`)
  - Un chemin d'accès (`Name`)
- **Opérations principales du `NamingContext`**
  - Ajouter une association nom-référence : *bind*, *rebind*
  - Résoudre une association : *resolve*
  - Détruire une association : *unbind*
  - Lister le contenu : *list*
  - Détruire le contexte : *destroy*

Nœud = contexte
Feuille = objet



Unicité des noms dans un contexte
Un objet peut avoir plusieurs noms
Analogie avec SGF

Le chemin d'accès

```
#pragma prefix "omg.org"
module CosNaming {
        typedef string Istring;
        // Définition d'un composant
        struct NameComponent {
                Istring id;
                Istring kind; // infos complémentaires, souvent vide en pratique
        };
        // Définition d'une séquence de composants (chemin d'accès)
        typedef sequence<NameComponent> Name;
        // …
};
```


Le contexte de nommage (1/2)

Définition des exceptions

```
interface NamingContext {
        enum NotFoundReason {
                missing_node, not_context, not_object };
        exception NotFound {            // pas de référence associée au nom
                NotFoundReason why; Name rest_of_name; };
        exception CannotProceed {       // impossible d'effectuer l'opération
                NamingContext cxt; Name rest_of_name; };
        exception InvalidName { };      // nom contenant des caractères invalides
        exception AlreadyBound { };     // nom déjà utilisé
        exception NotEmpty { };};       // destruction d'un contexte non vide
```


Le contexte de nommage (2/2)

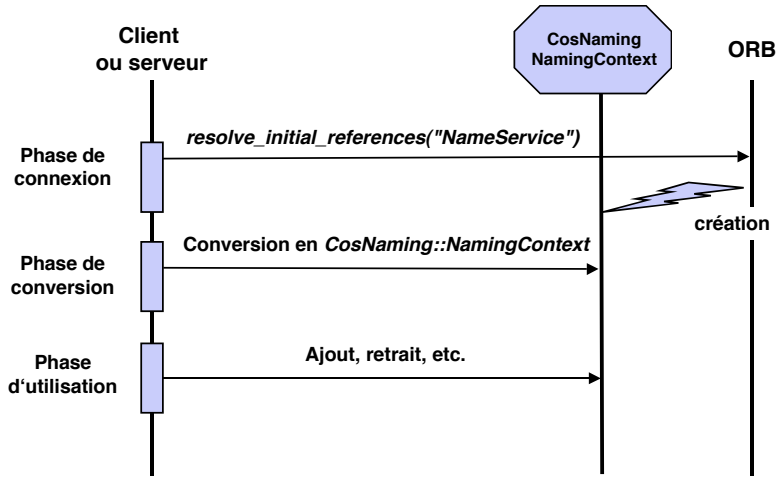
Définition des opérations

```
interface NamingContext {
// déclaration des exceptions

  void bind(in Name n, in Object obj) raises(NotFound, ...);          // lier un objet
  void rebind(in Name n, in Object obj) raises(NotFound, ...);        // lier un objet,
                                                                      // supprimer ancien lien
  void bind_context(in Name n, in NamingContext nc) raises(...);      // lier un contexte
  void rebind_context(in Name n, in NamingContext nc)raises(...);     // lier un contexte,
                                                                      // supprimer ancien lien
  Object resolve (in Name n) raises(NotFound, ...);                   // résoudre un lien
  void unbind(in Name n) raises(NotFound, ...);                       // supprimer un lien
  NamingContext new_context();                                        // créer contexte
  NamingContext bind_new_context(in Name n) raises(...);              // créer et lier contexte
  void destroy() raises(NotEmpty);                                    // détruire contexte

// autres déclarations
}
```

Obtenir la référence du service Nommage

- **Comment retrouver la référence du service Nommage ?**
  - Problème classique d'amorçage (*bootstrap*)
- **Solutions : les "objets notoires" du bus CORBA**
  - Petit annuaire contenant les services indispensables
  - Noms prédéfinis par convention (ici : "`NameService`")
- **Scénario unique, quel que soit le langage**
  a) opération *CORBA::ORB::resolve_initial_references*
  b) Conversion en *CosNaming..NamingContext*
  via l'opérateur *narrow*

Obtenir la référence du service Nommage (scénario)



Obtenir le service Nommage en Java

```
import org.omg.CosNaming.*;
org.omg.CORBA.Object objRef = null;
// obtenir la référence du service de nommage
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 vers un objet de type CosNaming
NamingContext nsRef = NamingContextHelper.narrow(objRef);
if ( nsRef == null ) {
        outils.ARRET ("Le service initial 'NameService' n'est pas
                un objet CosNaming::NamingContext");
}
```


Créer un nom (chemin) en Java

```
import org.omg.CosNaming.*;

// créer un chemin simple
NameComponent[ ] nsNom = new NameComponent [1];
nsNom[0] = new NameComponent( "repertoire", ""); // le paramètre kind est vide

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

Enregistrer un objet

- **Opération pour publier un objet**
  - **En général réalisée par le serveur**
    - **Éventuellement par le client, cas du *callback***
- **Scénario type**
  - **Créer un objet**
  - **Construire un chemin d'accès (*Name*)**
  - **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);

void **rebind**(in Name n, in Object obj)
raises(NotFound, CannotProceed, InvalidName);

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 dans le type correct**

```
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
Repertoire uneRefObjet = RepertoireHelper.narrow (objRef);
```

Lister le contenu

- **Permet d'obtenir le contenu d'un contexte**
  - **Équivalent de *ls* dans un SGF**
- **Utilisation**
  - **Opérations de l'interface *NamingContext***
  - **Nouvelle interface *BindingIterator***

Retour sur le contexte de nommage

```
module CosNaming {
        // définitions pour parcourir contexte de nommage
        enum BindingType {nobject, ncontext};
        struct Binding {
                Name binding_name;
                BindingType binding_type;
        };
        typedef sequence<Binding> BindingList;
        interface BindingIterator;
        interface NamingContext {
        // précédentes déclarations + opérations pour lister contenu
                void list(in unsigned long how_many,
                        out BindingList bl, out BindingIterator bi);
        };
};
```


L'interface *BindingIterator*

```
module CosNaming {
        interface BindingIterator {
        // obtenir l'entrée suivante dans le contexte courant
        boolean next_one(out Binding b);
        // obtenir les n entrées suivantes dans le contexte courant
        boolean next_n(in unsigned long how_many,
                out BindingList bl);
        // détruire l'itérateur
        void destroy();
        };
};
```


Lister le contenu en Java

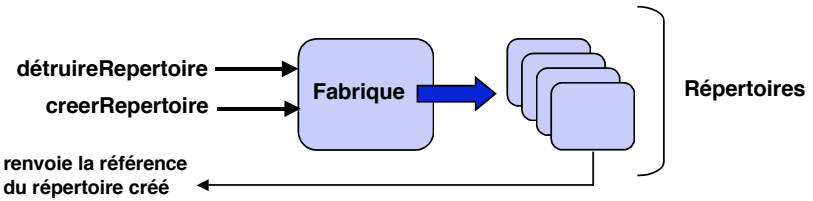
```
import org.omg.CosNaming.*;
// Notez que la projection IDL/Java d'une séquence est un tableau
Binding[] bl = null;
BindingListHolder blh = new BindingListHolder();
BindingIteratorHolder bih = new BindingIteratorHolder();
contexte.list (10, blh, bih);
bl = blh.value;
for (int i=0; i < bl.length; i++) {
        System.out.println ( Name2String (bl[i].binding_name) ); }
BindingIterator bi = bih.value;
if (bi != null) {
        boolean continuer = true;
        while (continuer) {
                continuer = bi.next_n(10,blh);
                bl = blh.value;
                for (int i=0; i < bl.length; i++) {
                        System.out.println ( Name2String (bl[i].binding_name) );
                }
        }
        bi.destroy();
}
```

Cycle de vie des objets

- **Problème**
  - Actuellement, 1 répertoire = 1 serveur
  - Pas de création ou destruction d'objets à distance
  - Seules opérations = appels d'opérations
- **Solution**
  - Notion de fabrique d'objets (cf Java RMI)
  - Exprimée en OMG IDL
  - Canevas de conception général (cf Java RMI)

L'interface OMG IDL de la fabrique de répertoires

```
module annuaire (
        …
        interface Fabrique (
           Repertoire creerRepertoire (in string libelle) ;

           void detruireRepertoire(in Repertoire r);
        );
);
// attention au passage par référence
```



L'implantation Java de la fabrique de répertoires

```
import fr.lifl.annuaire.*;
public class FabriqueImpl extends _FabriqueImplBase {
        public FabriqueImpl () { … }

        public Repertoire creerRepertoire (String libelle) {
                return new RepertoireImpl(libelle);
        }

        public void detruireRepertoire(Repertoire repertoire) {
                org.omg.CORBA.ORB.init().disconnect(repertoire);
        }
}
```


Le serveur de la fabrique

- **Initialiser le bus CORBA**
  - Obtenir l'objet ORB
- **Initialiser l'adaptateur d'objets**
  - Obtenir le BOA (maintenant remplacé par POA, cf cours suivant)
- **Créer la fabrique**
- **Obtenir le service de nommage (NS)**
- **Enregistrer la fabrique dans le NS**
- **Attendre des requêtes venant du bus**

Le 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 {
        ORB orb = ORB.init (args.null);
        BOA boa = BOA_init (args.null);
        // (en pratique on utilise un POA)
        // création de l'objet Fabrique
        FabriqueImpl fabrique = new FabriqueImpl();
        NamingContext nc = NamingContextHelper.narrow(
                orb.resolve_initial_references ("NameService")) ;
        NameComponent[] nom = new NameComponent[1];
        nom[0] = new NameComponent( "Fabrique", "");
        nc.rebind(nom, fabrique);          // enregistrer la fabrique
        boa.impl_is_ready(null);
        }
}
```

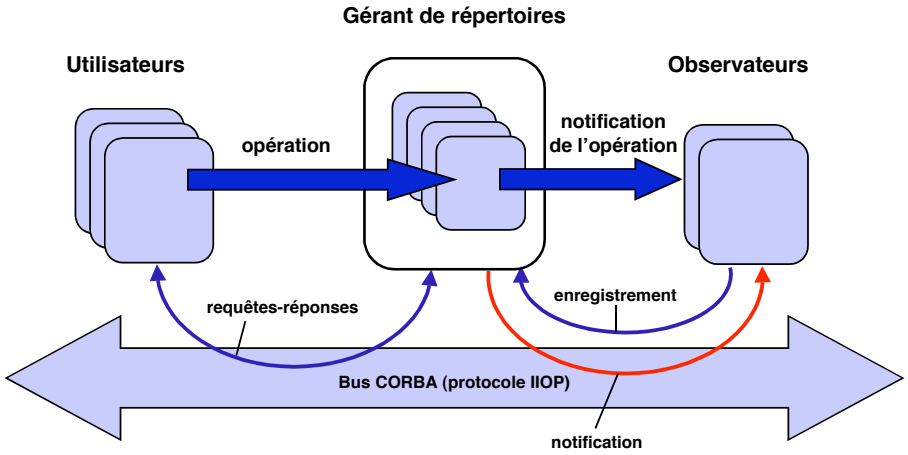

Application d'administration de la fabrique

- Création d'un nouveau répertoire et mise à disposition par le service Nommage
- Initialiser le bus CORBA
- Obtenir le service Nommage (NS)
- Obtenir la fabrique depuis le NS
- Créer un répertoire
- Enregistrer le répertoire dans le NS

L'application d'administration en Java

```
import org.omg.CORBA.*;                    // le bus CORBA
import org.omg.CosNaming.*;                // le service Nommage
import fr.lifl.annuaire.*;
public class Administration {
        public static void main(String[ ] args) throws Exception {
        ORB orb = ORB.init (args.null);
        NamingContext nc = NamingContextHelper.narrow(
                orb.resolve_initial_references ("NameService")) ;
        NameComponent[] nom = new NameComponent[1];
        nsNom[0] = new NameComponent( "Fabrique", "");
        org.omg.CORBA.Object objet = nc.resolve(nsNom);
        Fabrique fabrique = FabriqueHelper.narrow(objet);
        Reprtoire r = fabrique.creerRepertoire("LIFL");
        nsNom[0] = new NameComponent( "MonRepertoire", "");
        nc.rebind(nsNom, r);
        }
}
```


Notification des clients



Canevas de conception pour la notification

- **Patron de conception "Observé - Observateur"**
- **Côté application prévenue (Observateur)**
  - **Interface à implanter ; ObservateurRepertoire**
  - **1 opération par type d'événement**
  - **`oneway` pour opération asynchrone**
- **Côté serveur notifiant (Observé)**
  - **1 interface pour enregistrer les observateurs (souhaitant être prévenus d'une opération particulière)**
  - **Héritage d'interfaces OMG IDL (un répertoire observé est aussi un `Repertoire`)**

Interfaces OMG IDL pour la notification

```
module annuaire {
        // Interface pour l'application cliente.
        interface ObservateurRepertoire {
                oneway void annoncerCreationPersonne (in Nom nom);
                oneway void annoncerDestructionPersonne (in Nom nom);
                oneway void annoncerModificationPersonne (in Nom nom);
        };
        // Interface pour l'application serveur.
        interface RepertoireObserve : Repertoire {
                void ajouterObservateur (in ObservateurRepertoire obs);
                void retirerObservateur (in ObservateurRepertoire obs);
        };
};
```


L'implantation du répertoire observé

- **Utilisation du squelette OMG IDL**
- **Réutilisation de l'implantation existante**
  - **Par héritage ou par délégation**
- **Gestion de la liste des observateurs**
  - **Attention : pour compareer des interfaces d'objets CORBA, utiliser `CORBA::Object::is_equivalent`**
- **Redéfinition des opérations impliquant une notification**
  - **Réutilisation du code existant**
  - **Notification des observateurs**
  - **Prise en charge des pannes : `CORBA::COMM_FAILURE`**

L'implantation Java du répertoire observé (1/3)

```
import fr.lifl.annuaire.*;
import fr.lifl.annuaire.RepertoirePackage.*;
import java.util.*;
public class RepertoireObserveImpl
                        extends _RepertoireObserveImplBase {
// Utilisation de la délégation car pas d'héritage multiple.

        protected RepertoireImpl le_repertoire;
        protected Vector observateurs;
        public RepertoireObserveImpl (String libelle) {
                this.le_repertoire = new RepertoireImpl (libelle);
                this.observateurs = new Vector ();
        }
```

L'implantation Java du répertoire observé (2/3)

```
public String libelle() {
        return le_repertoire.libelle(); }
public Personne obtenirPersonne(String nom) throws Inconnu {
        return le_repertoire.obtenirPersonne (nom); }
public String[] listerNoms() {
        return le_repertoire.listerNoms (); }
public void ajouterObservateur(ObservateurRepertoire obs) {
        this.observateurs.addElement (observateur); }
public void retirerObservateur(ObservateurRepertoire obs) {
        for(int i=0; i<this.observateurs.size(); i++) {
                ObservateurRepertoire o = (ObservateurRepertoire)this.
                observateurs.elementAt(i);
                if (o._is_equivalent(obs)) {
                        this.observateurs.removeAt(i);
                return;}
                }
        }
```


L'implantation Java du répertoire observé (3/3)

```
public void ajouterPersonne(Personne personne)throws ExisteDeja {
        le_repertoire.ajouterPersonne (personne);
        for (Enumeration e=observateurs.elements(); e.hasMoreElements(); ) {
                ObservateurRepertoire o = (ObservateurRepertoire)
                e.nextElement();
                try { o.annoncerCreationPersonne(personne.nom);
                } catch (org.omg.CORBA.COMM_FAILURE cf) {
                        retirerObservateur(o); }
                }
        }
```

Les autres méthodes sont modifiées de manière analogue

Dans `retirerPersonne()`, appel de l'opération `annoncerDestructionPersonne` de chaque observateur

Dans `modifierPersonne()`, appel de l'opération `annoncerModificationPersonne` de chaque observateur

Bilan sur l'application

- **Pour construire une application CORBA, il faut**
  - **Respecter la structure des applications (décomposition, interfaces)**
  - **Se plier aux règles de projection / programmation**
    - **Dépendent du langage utilisé**
- **On obtient ainsi**
  - **La portabilité du code**
  - **L'interopérabilité entre applications**
  - **La facilité d'adaptation de l'application à d'autres contextes**
- **Pour aller plus loin**
  - **Remplacer l'utilisation du BOA (obsolète, mais toujours utilisable) par celle du POA, plus complet (mais plus complexe)**
  - **L'usage du POA peut être explicite, ou bien caché sous une interface de programmation (exemple : environnement J2SE 1.4 pour Java, cf TP)**