Compléments sur CORBA

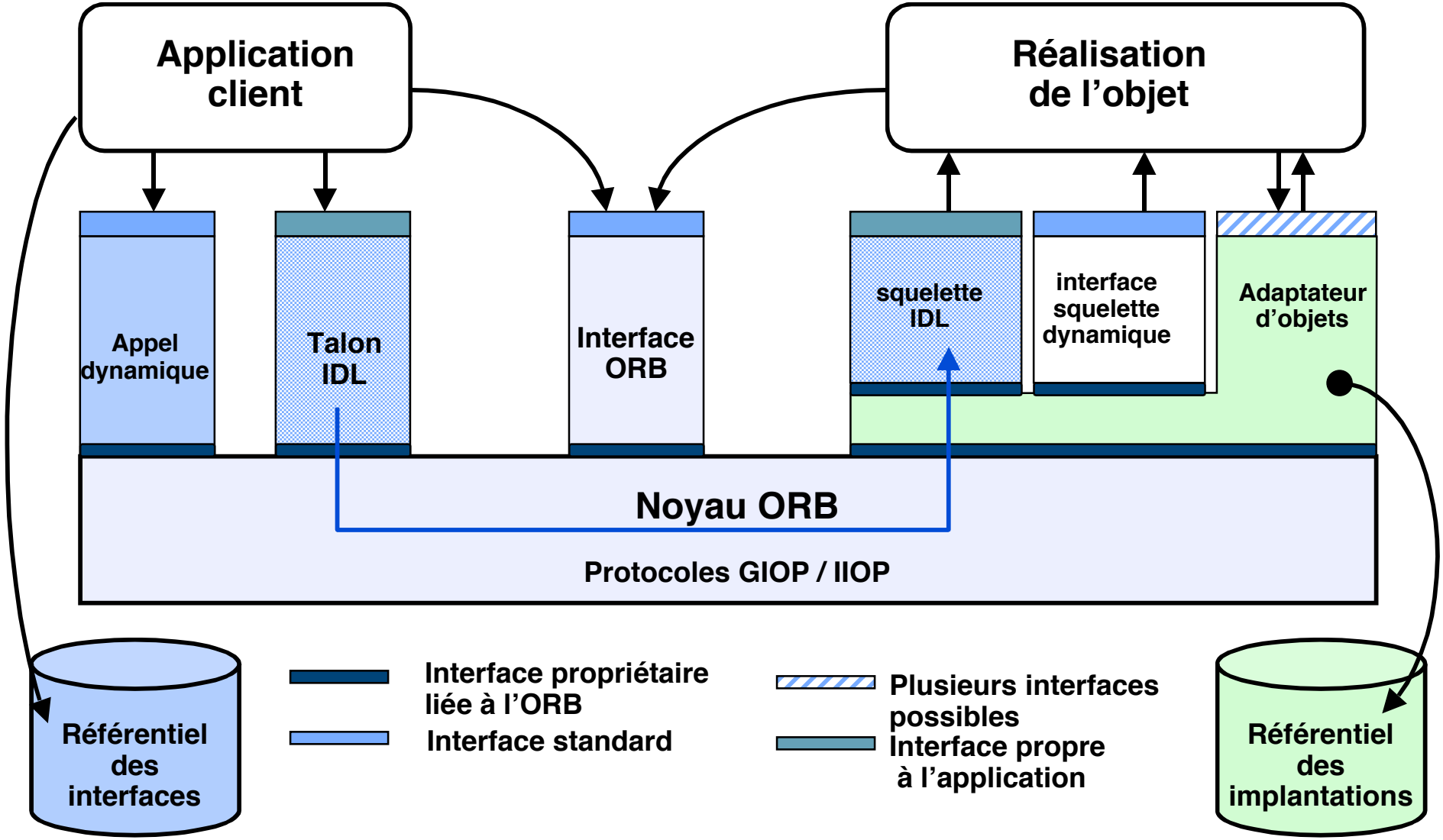
S. Krakowiak

Université Joseph Fourier

Projet Sardes (INRIA et IMAG-LSR)

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

Schéma d'ensemble de CORBA (rappel)



© 2006-2007, S. Krakowiak

Plan de la présentation

- La construction d'une application CORBA
- Répertoire d'interfaces et appel dynamique
- Adaptateur d'objets
  - POA (Portable Object Adaptor)
- Détails sur les références d'objets
- Interopérabilité entre bus CORBA

compléments sur utilisation

fonctionnement interne

Sources :

J.M. Geib, C. Gransart, Ph. Merle. *CORBA, des concepts à la pratique*, 2ème édition, InterEditions 1999

M. Henning. Binding, Migration and Scalability in CORBA

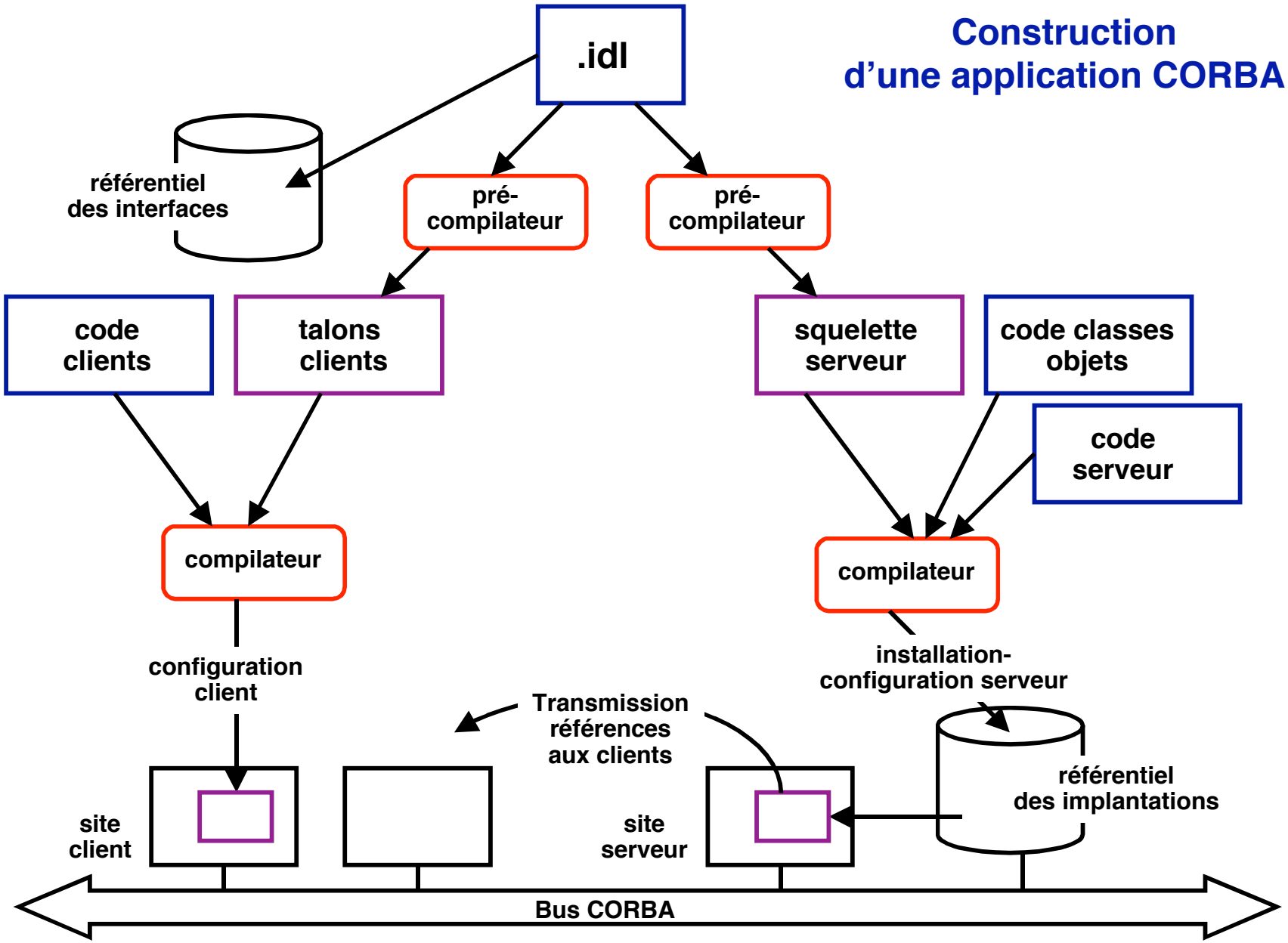
D. Schmidt, S. Vinoski. Using the Portable Object Adapter for the Transient and Persistent CORBA Objects, *C++ Report* 1, 4 (April 1998)

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

J. Daniel, *Au cœur de CORBA avec Java*, 2ème edition, Vuibert Informatique, 2002

© 2006-2007, S. Krakowiak

Construction d'une application CORBA



Référentiel des interfaces

- **Motivation**
  - Rendre l'environnement CORBA auto-descriptif (découverte dynamique des ressources disponibles)
- **Fonction**
  - Conserver les définitions d'interface (IDL) de tous les objets disponibles (méta-données)
  - Fournir des fonctions de recherche et de navigation dans l'espace des méta-données
- **Mise en œuvre**
  - Les interfaces décrivant le référentiel et permettant de l'exploiter sont elles-mêmes stockées dans le référentiel
  - Le référentiel est lui-même un "objet notoire" de l'ORB

© 2006-2007, S. Krakowiak

Référentiel des interfaces : usage

- **Installation et distribution des interfaces IDL**
  - Stockage en vue d'une utilisation ultérieure
- **Utilisation par les outils de développement**
  - Navigateurs de classes, générateurs de code
- **Objets auto-descriptifs**
  - Découverte de l'interface d'un objet (*Object*::*_get_interface_def*)
  - Construction de requêtes dynamiques
- **Contrôles et vérifications sur les interfaces**
  - Vérification de l'absence de circuit dans le graphe d'héritage
  - Vérification de la signature des opérations
- **Interconnexion de bus CORBA**
  - Conversion de requêtes

© 2006-2007, S. Krakowiak

Référentiel des interfaces : organisation

Méta-définition des objets

- Méta-données décrivant les interfaces IDL de tous les objets CORBA (*Repository, ModuleDef, InterfaceDef, AttributeDef, OperationDef, ParameterDef, ExceptionDef, TypedefDef, ConstantDef*)
- Décrit les opérations génériques possibles sur les objets correspondants

Hiérarchie des interfaces

- Graphe d'héritage (multiple) permettant de définir les interfaces ci-dessus à partir d'interfaces abstraites génériques

© 2006-2007, S. Krakowiak

Référentiel des interfaces : exemple d'utilisation en Java (1)

```
// utiliser le référentiel des interfaces pour découvrir l'interface d'un objet
//          (dont a récupéré une référence par un moyen quelconque (IOR "stringifié", résultat, etc.)
// on peut utiliser cette information de diverses manières
//          - à des fins de documentation : imprimer l'interface (ce qu'on va faire dans cet exemple)
//          - pour construire un appel dynamique (ce qu'on fera plus loin)

import java.io;
Import org.omg.CORBA;
import org.omg.CORBA.InterfaceDefPackage.*;

public class InterfaceExplorer {

   public explore(Object obj);        // le type de la référence n'est pas connu : on utilise Object

// 1. Obtenir la définition d'interface (méthode de la classe Object)

          InterfaceDef itf_def = InterfaceDefHelper.narrow(obj._get_interface_def()) ;
// _get_interface_def() est une méthode de la classe Object

// 2. Obtenir une description complète de cette interface

          FullInterFaceDescription full_itf_desc = itf_def.describe_interface();

// 3. Utiliser cette description pour trouver tous les éléments de l'interface : nom, type du résultat,
          liste des paramètres avec leur sens, type et nom
```

Source : Brose, Vogel, Duddy, *Java Programming in CORBA,* 3rd ed. Wiley, 2001

© 2006-2007, S. Krakowiak

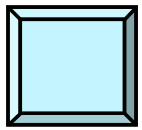
Référentiel des interfaces : exemple d'utilisation en Java (2)

```
// 3. Utiliser cette description pour trouver tous les éléments de l'interface : nom, type du résultat,
      liste des paramètres avec leur sens, type et nom

        int i, j, nparam;
        for (i=0; i < full_itf_desc.operations.length; i++) {          // itération sur les opérations

                nparam = full_itf_desc.operations[i].parameters.length; // nb de paramètres

                // imprimer le type du résultat et le nom de l'opération
                System.out.printn(
                        full_itf_desc.operations[i].result + " " +
                        full_itf_desc.operations[i].name + " (" );
                String mode, in, out, out;
                in = new String("in"); inout = new String("inout"); out = new String("out");
                char last = ",";

                // imprimer les paramètres de l'opération
```

© 2006-2007, S. Krakowiak

Référentiel des interfaces : exemple d'utilisation en Java (3)

```
for (i=0; i < full_itf_desc.operations.length; i++) {          // itération sur les opérations
        // ...
        // imprimer les paramètres de l'opération
        for (j=0; j<nparam; j++) {
            switch (full_itf_desc.operations[i].parameters[j].mode.value() ) {
                case ParameterMode._PARAM_IN:
                        mode = in; break;
                case ParameterMode._PARAM_INOUT:
                        mode = inout; break;
                case ParameterMode._PARAM_OUT:
                        mode = out; break;
                default:
                        mode = new String("inconnu");
            }
            if (j == nparam-1) {
                last = " ";        // pas de virgule après dernier paramètre
            System.out.println(
                        mode + " " +
                        full_itf_desc.operations[i].parameters[j].type + " " +
                        full_itf_desc.operations[i].name " " + last );
        }           // fin d'impression des paramètres avec sens, type, nom

        // imprimer la parenthèse fermant la liste des paramètres
        System.out.println(" ); \n");
    } // fin itération sur opérations
}} // fin méthode, fin classe
```



Référentiel des interfaces : exemple d'utilisation en C++(1)

```
//Consulter le référentiel

// 1. obtenir le référentiel (objet notoire)
CORBA::object_var objet =
        orb->resolve_initial_references ("InterfaceRepository");
// 2. convertir vers le type réel
CORBA::Repository_var referentiel = CORBA::Repository::_narrow(object):

// 3. rechercher une définition par son nom étendu (hiérarchie de classes)
CORBA::Contained_var definition = referentiel->lookup ("MonModule::MonInterface");

// 4. rechercher une définition par son identifiant dans le référentiel
definition = referentiel->lookup_id("IDL:MonModule/MonInterface:1.0");

// 5. conversion vers le type voulu
CORBA::Container_var conteneur = CORBA::Container::_narrow (definition);

// 6. obtenir la liste de tous les objets contenus
CORBA::ContainedSeq_var contenus = conteneur->contents (CORBA::dk_all, CORBA::TRUE);

cout << "Liste des noms des objets du conteneur" << endl;
// 7. parcourir la séquence des objets contenus
for (CORBA::ULong i=0; i<contenus->length(); i++ {
        // 8. consultation du nom de chaque objet contenu
        CORBA::String_var nom=contenus[i] ->name();
        cout <<  " " << nom << endl;
        }
```

© 2006-2007, S. Krakowiak

Référentiel des interfaces : exemple d'utilisation en C++(2)

```
//Consulter l'interface d'un objet (l'interface est passee en parametre)

void afficher_interface (CORBA::InterfaceDef_ptr interface)
{
        CORBA::String_var nom_absolu = interface->absolute_name();

        CORBA::InterfaceDef::FullInterfaceDescription_var
                description = interface->describe_interface ();

        // on peut maintenant obtenir tous les elements
        cout << "Nom       = " << description->name << endl; // (idem pour version etc.)

        // la liste des noms des operations

        CORBA::OpDescriptionSeq_ptr operations = description->operations;
        cout <<"Liste des noms des operations" << endl;
        for CORBA::Ulong i=0; i<operations.length(); i++){
                cout << " " << operations[i].name << endl;
        }
        // idem pour les attributs avec description->attributes
}

//Utilisation depuis le main

// obtenir une reference d'objet
CORBA::Object_var objet = ...
CORBA:InterfaceDef_var interface = objet->get_interface ()
afficher_interface (interface)
```

© 2006-2007, S. Krakowiak

Appel dynamique

Motivations

- Une application cliente souhaite pouvoir appeler un objet dont l'interface n'est pas connue au moment de la compilation de l'application
- Exemples :
  - un navigateur qui doit interroger un ensemble d'objets créés dynamiquement
  - un langage de script (exécution interprétative d'objets créés dynamiquement

Réalisation

- Un référentiel des interfaces (description d'interfaces disponibles)
- Un outil de construction dynamique de requêtes
- Une interface d'appel dynamique (côté client)
- Une interface de squelette dynamique (côté serveur)

© 2006-2007, S. Krakowiak

Étapes d'un appel dynamique

- Trouver la référence de l'objet à appeler
  - Via serveur de noms ou *string_to_object*
- Obtenir l'interface de l'objet
  - Opération *ref._get_interface_def*
- Obtenir la description de l'opération à appeler
  - Nom, type et mode de transmission des paramètres
  - Via le référentiel des interfaces
- Construire ("à la main") une liste d'arguments
  - Arguments de type *NamedValue*
  - Liste de type *NVList*
- Créer une requête à partir de la liste
  - Via les opérations ad hoc sur le type *request*
- Envoyer la requête
  - Différents modes d'envoi : *invoke, send_oneway, send_deferred,…*
- Recevoir et traiter les résultats

© 2006-2007, S. Krakowiak

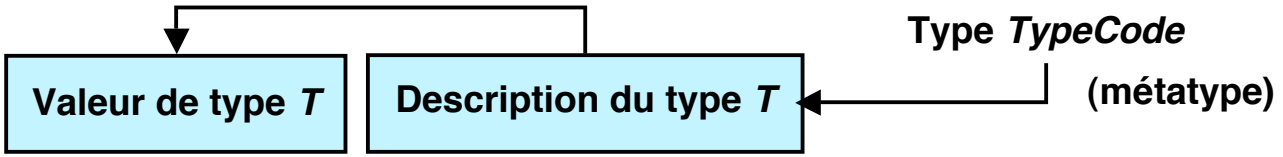
Un outil utile : le type générique *Any*

Motivations

- On veut réaliser des opérations polymorphes (dont les paramètres peuvent être de différents types possibles)
- Dans les appels dynamiques, on veut manipuler des objets dont le type n'est pas connu a priori

Solution : le type générique *org.omg.CORBA.Any*

- Un objet de type *Any* se compose d'une valeur et d'un descripteur de type



- Des opérations permettent de construire un tel objet, de connaître le type, d'extraire la valeur

```
Any  x = orb.create_any ;                          // objet initialement vide (pas de valeur)
Any y =  orb.create_any ;                          // un autre
x.insert_string ("bonjour") ;                      // donner une valeur de type primitif String
p = new personne( ...) ;                           // créer un objet de type construit
personneHelper.insert(y, p) ;                      // donner une valeur de type construit personne
String s = x.extract_string() ;                    // récupérer la valeur (supposant connu son type)
personne z = personneHelper.extract(y) ;           // idem pour type construit
TypeCode t = y.type();                             // permet de connaître le type d'un Any
Boolean b = t.equal(personneHelper.type());        // renvoie TRUE
```

© 2006-2007, S. Krakowiak

Appel dynamique (côté client) : format d'une requête

Une requête est du type org.omg.CORBA.Request

Elle contient les éléments suivants :
- **le nom de la méthode appelée**
- **les paramètres (une liste de NamedValues)**
- **les résultats (une NamedValue)**

Interface IDL de NamedValue

```
typedef string identifier;
pseudo interface NamedValue{
        readonly attribute identifier name
        readonly attribute any value
        readonly attribute Flags flags}
```

Interface IDL de NVList // liste de NamedValues

```
pseudo interface NVList{
        readonly attribute unsigned long count;
        NamedValue add(in Flags flags);
        add_value(in identifier item_name,
                in any val, in Flags, flags);
        NamedValue item(in usigned long index)
                raises CORBA::Bounds;
        void remove(in unsigned long index)
                raises CORBA::Bounds;
        }
```

Interface Java de NamedValue

```
public abstract class NamedValue {
        public abstract String name();
        public abstract Any value();
        public abstract int flags();}

Création :

NamedValue create_named_value(String name,
        Any value, int flags);
```

Interface Java de NVList

```
public abstract class NVList {
        public abstract void add(int flags);
        public abstract void add_item(String item_name,
                inf flags);
        public abstract void add_value(String item_name,
                Any val, int flags);
        public abstract int flags();
        public abstract int count();
        public abstract NamedValue item(int index)
                throws org.omg.CORBA.Bounds;
        public abstract void remove(int index)
                throws org.omg.CORBA.Bounds; }
```

© 2006-2007, S. Krakowiak

Appel dynamique (côté client) : exemple

On reprend l'exemple donné pour le référentiel des interfaces : on a obtenu *full_itf_desc* pour un objet *obj* (inconnu lors de la compilation). On va faire un appel dynamique sur la première méthode de cet objet (dans l'ordre des opérations de la description).

Appel dynamique

```
// 1. Créer l'objet requête
Request req =
    obj._request(full_itf_desc.operations[0].name);

//2. Créer une liste d'arguments (NVList)
NVList arg_list = req.arguments();

// 3. Initialiser la liste (après avoir trouvé sa longueur)
nparam =
    full_itf_desc.operations[0].parameters.length;
for (i=0 ;i< nparam; i++) {
    Any argAny = orb.create_any();
    argAny.type(full_itf_desc.operations[0].parameters[i].type);
    // ici code (en général interactif) pour remplir la valeur
    // de l'argument si IN ou INOUT
    arg_list.add_value(
        full_itf_desc.operations[0].parameters[i].name,
        argAny,
        full_itf_desc.operations[0].parameters[i].mode.value());}

//4. Fixer le type du résultat
req.set_return_type(full_itf_desc.operations[0].result);
```

Appel dynamique (suite)

```
// 5. Exécuter l'appel
req.invoke();

// 6. Obtenir résultat

Any res_any = req.result().value();

// utiliser le résultat...

//7. Obtenir les arguments OUT
NVList nv_list = req.arguments();
Int i;
for (i=0; i<nparam; i++) {
    // utiliser valeurs des arguments OUT
    // (tels que la valeur de
    // parameters[i].mode soit OUT)
```

// fixe le *TypeCode* d'une valeur de type *Any*

Appel dynamique (côté serveur)

L'équivalent côté serveur de l'interface DII est l'interface DSI (*Dynamic Skeleton Interface*). Elle permet de construire "à la main" (sans compilation IDL) la partie serveur (plus précisément servant) d'un appel d'objet.

Principe de DSI :`

Deux opérations sont prévues :

invoke(ServerRequest request) qui réalise l'appel
_ids() qui renvoie l'identification (dans le référentiel des interfaces) de l'interface réalisée

Le réalisateur du squelette dynamique doit implémenter ces opérations.
Les étapes pour *invoke* sont les suivantes :

récupérer le nom de l'opération appelée (et vérifier que celui ou l'un de ceux attendu(s))
récupérer les paramètres, qui sont fournis sous forme d'*Any* (via *extract*)
définir le type du résultat éventuel qui sera renvoyé (via *insert()*)
définir les exceptions éventuelles qui seront renvoyées
réaliser l'opération demandée (directement ou en appelant un servant)

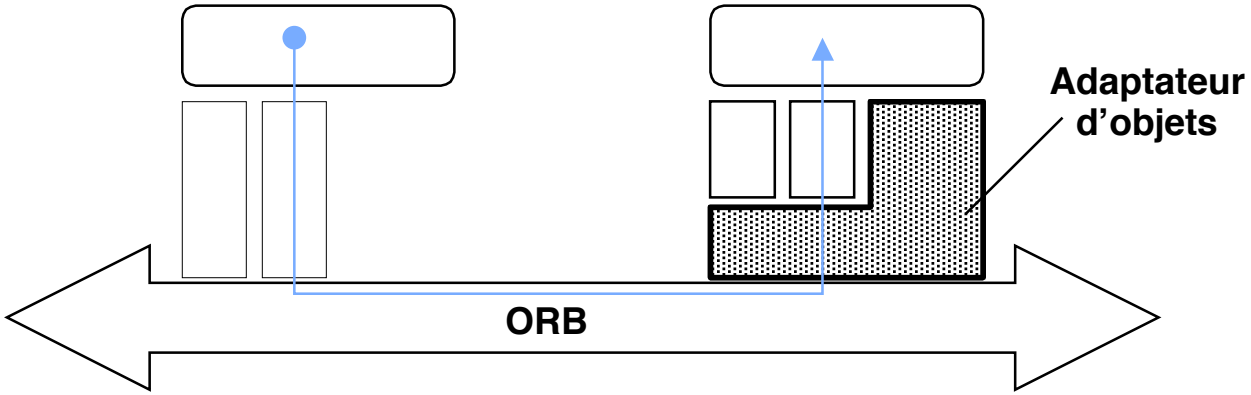
enfin enregistrer le squelette dans l'adaptateur (cf plus loin) et faire connaître sa référence au client

© 2006-2007, S. Krakowiak

Adaptateur d'objets

Motivations

- Isoler les fonctions de gestion des implémentations des objets ("servants").
- Ces fonctions auraient pu être intégrées dans l'ORB, mais avec des inconvénients :
  - Alourdissement de l'ORB, perte de performance pour l'ensemble des applications
  - Manque de souplesse d'adaptation (modifier l'ensemble de l'ORB pour introduire une nouvelle fonction)



© 2006-2007, S. Krakowiak

Adaptateur d'objets

Fonctions

- Aiguillage des requêtes des clients
  - Vers l'objet appelé
  - Vers l'opération appelée (à travers le squelette)
- Activation et désactivation des objets
  - Association entre object CORBA et servant
  - Création et lancement de processus ou *threads*
  - Arrêt ou destruction des processus
- Génération des références d'objets (IORs)
  - IOR = Interoperable Object Reference
  - Utilisée par le client pour l'appel d'un objet
  - Utilisée par l'adaptateur pour retrouver l'objet
- Gestion du référentiel des implémentations

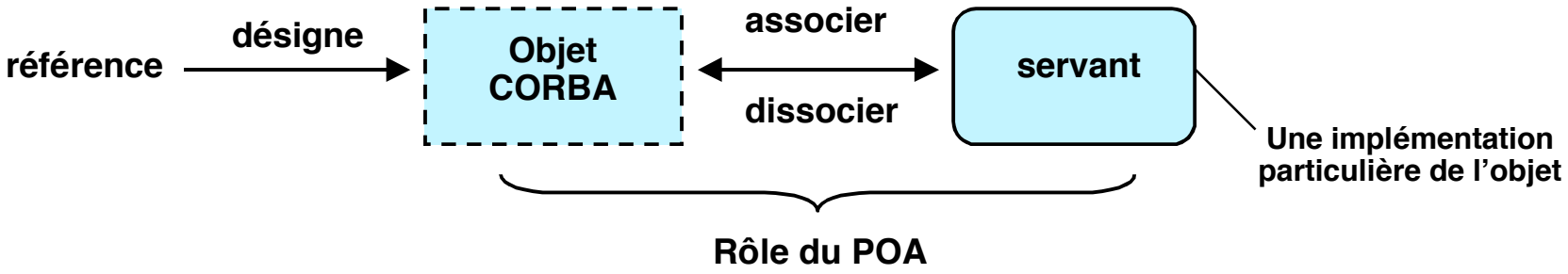
© 2006-2007, S. Krakowiak

Adaptateur d'objets : état courant

- **Initialement : BOA (Basic Object Adapter)**
  - Fonctions minimales
  - Spécifications imparfaites et incomplètes
- **Actuellement : POA (Portable Object Adapter)**
  - En vigueur à l'OMG depuis 1998 (le BOA devient obsolète)
  - Permet de personnaliser les politiques de gestion des servants
    - Persistant ou transitoire
    - Format des références (IOR)
    - Activation (implicite, sur demande, serveur par défaut, etc.)
    - Génération des serveurs d'objets (héritage, etc.)
  - Plusieurs POAs peuvent coexister

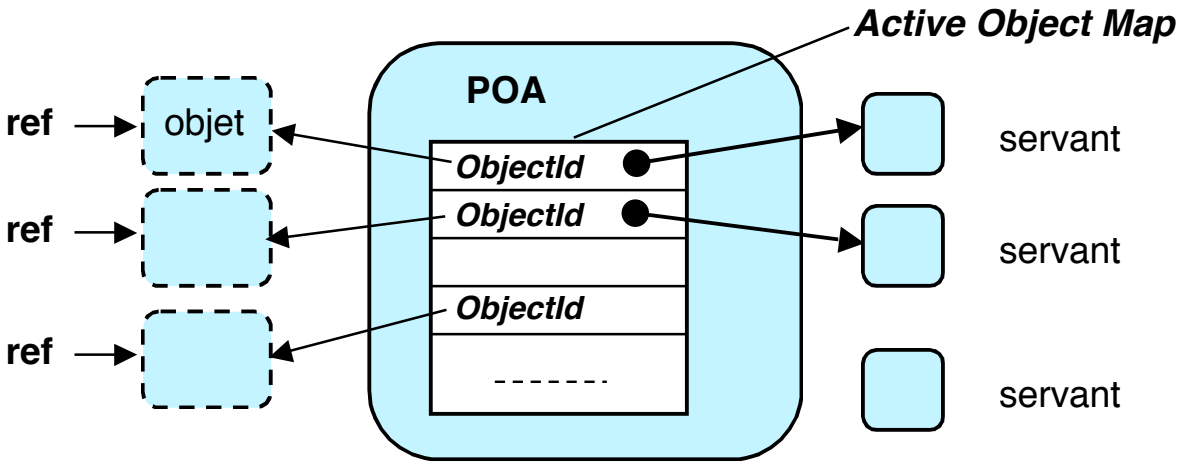
© 2006-2007, S. Krakowiak

Relations entre objets CORBA et servants dans un POA



Dans le POA, un objet CORBA est désigné par un identificateur d'objet (*ObjectId*).
Cet *ObjectId* n'a de signification qu'à l'intérieur du POA, alors qu'une référence d'objet est utilisable partout

Le POA gère une table d'association (*Active Object Map*) entre *ObjectId* et servant

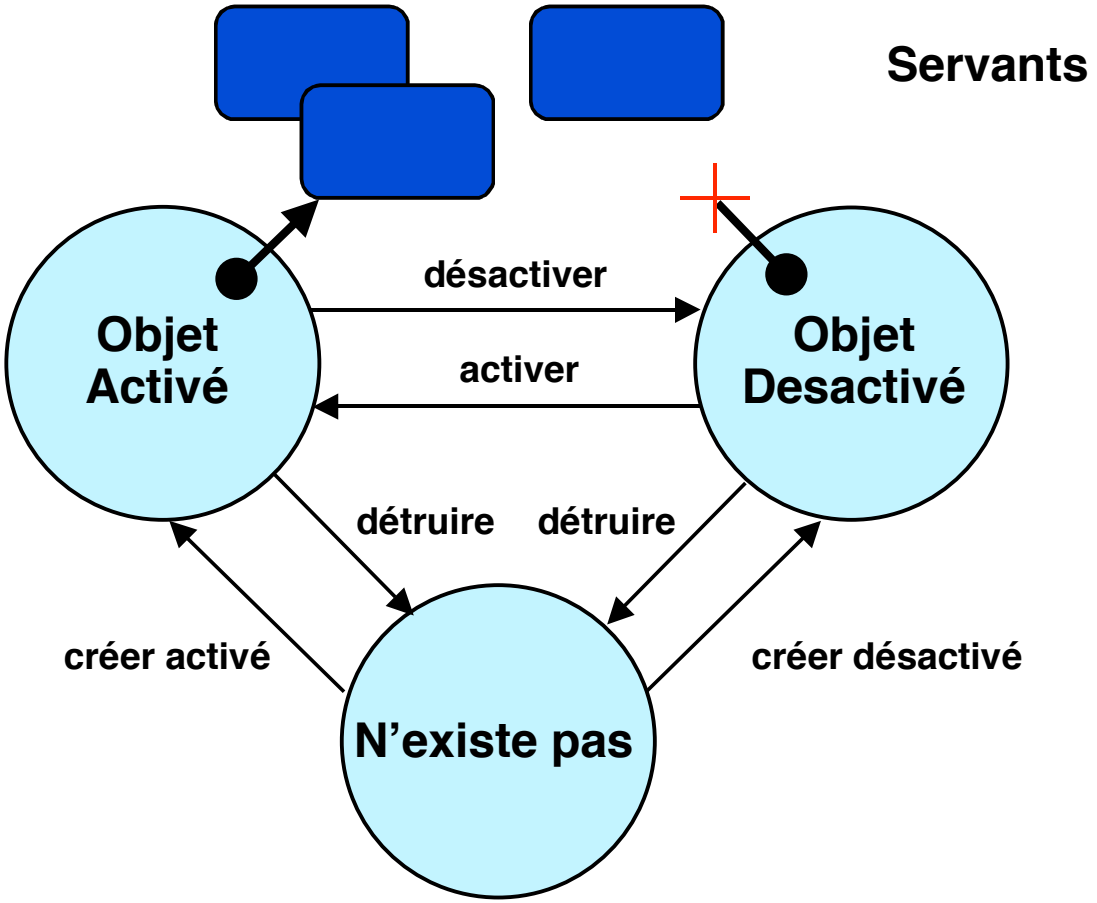


Remarque importante

Les durées de vie des objets et des servants sont indépendantes
Un servant peut exister sans être associé à un objet, et vice-versa

© 2006-2007, S. Krakowiak

Cycle de vie des objets dans un POA (simplifié)



Un objet peut être associé à un servant (et un seul). Il est alors activé et peut être appelé par un client.
Sinon il est désactivé et ne peut pas être appelé.

Un servant peut être associé à plusieurs objets. Il doit alors gérer la synchronisation nécessaire à ce partage.

Un objet peut être associé à différents servants au cours du temps (mais à un seul à un moment donné).

Vice-versa un servant ne peut être appelé que s'il est associé à un objet (on dit qu'il incarne l'objet)

© 2006-2007, S. Krakowiak

Programmation d'un POA

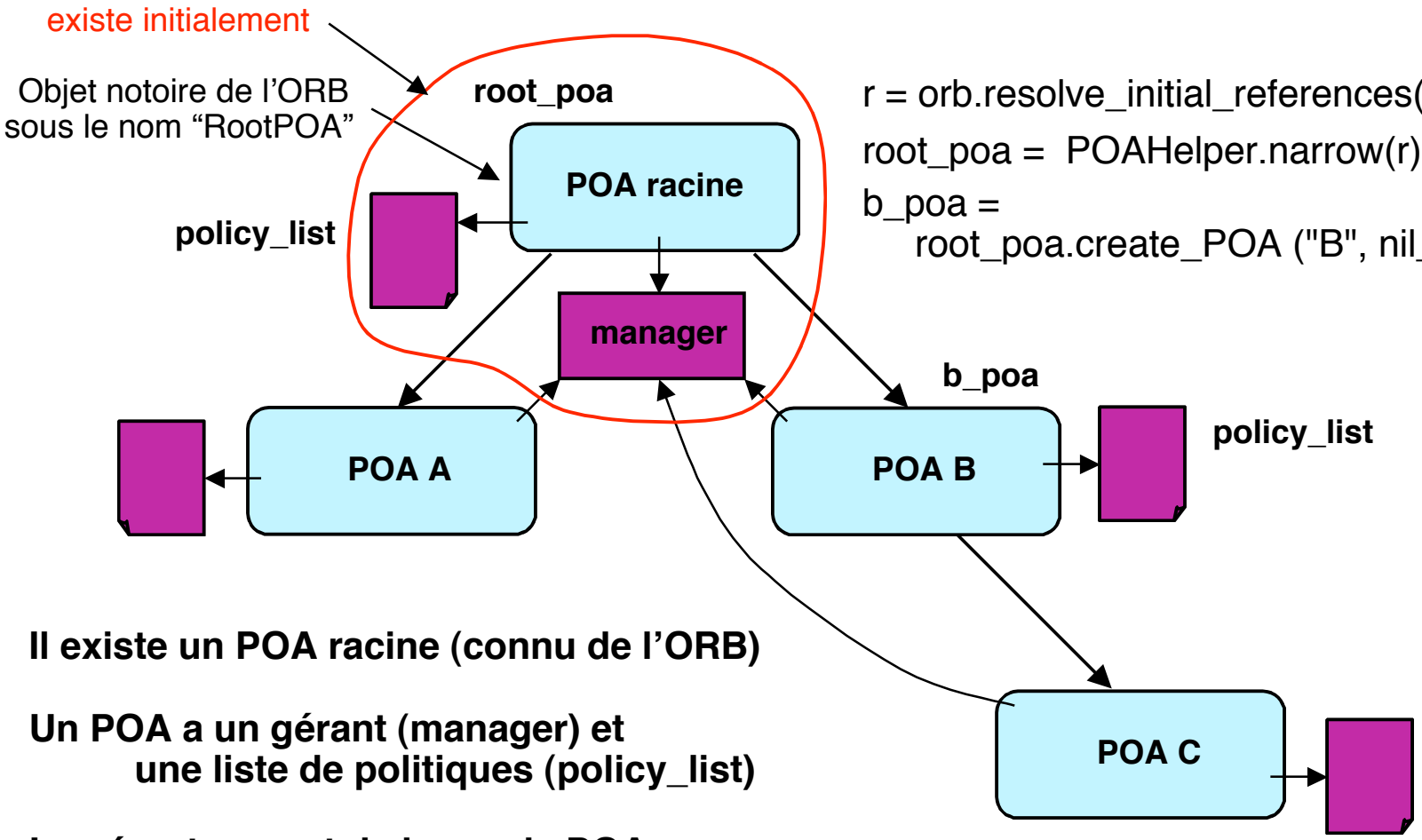
Le POA fournit diverses opérations pour gérer objets et servants, par ex. :

```
void activate_object_with_id(in ObjectId, in Servant p_servant)
        raises (ServantAlreadyActive, ObjectAlreadyActive, WrongPolicy);
// crée un objet activé, avec un Oid fourni par le programmeur (si la politique spécifiée le permet)

ObjectId activate_object (in Servant p_servant)
        raises (ServantAlreadyActive, WrongPolicy);
// crée un objet activé, et crée un Oid qu'il renvoie

Object create_reference_withId(in ObjectId oid, in CORBA::RepositoryId intf);
// crée un objet non activé, avec un Oid fourni et une interface spécifiée et renvoie la référence

Object create_reference in CORBA::RepositoryId intf);
// crée un objet non activé avec une interface spécifiée et renvoie la référence

Object servant_to_reference(in Servant p_servant)
        raises (ServantNotActive, WrongPolicy)
// crée un objet activé, associé au servant, et renvoie sa référence
```

NB les descriptions des effets sont simplifiées ; en fait elles dépendent des politiques

© 2006-2007, S. Krakowiak

Organisation des POA




```
r = orb.resolve_initial_references("RootPOA") ;
root_poa =  POAHelper.narrow(r) ;
b_poa =
    root_poa.create_POA ("B", nil_ mgr, policy_list) ;
```

Il existe un POA racine (connu de l'ORB)

Un POA a un gérant (manager) et une liste de politiques (policy_list)

Le gérant permet de lancer le POA

La liste de politiques définit la manière dont le POA gère les servants (persistance, activation, désignation, synchro., etc.)

© 2006-2007, S. Krakowiak

Utilisation d'un adaptateur d'objets : exemple (POA) en Java

```
import org.omg.CORBA.*

public static void main (String [] args)
    {   // 1. Initialiser l'ORB
        ORB orb = ORB.init (args, null);

        //2. Obtenir ref d'objet pour le POA racine
         POA root_poa = POAHelper.narrow (orb.resolve_initial_references ("RootPOA"));

        //3. Créer un servant d'objet pour traiter les requêtes des clients
        ServantInterface servant = new ServantImpl;
                // classe et interfaces définies par ailleurs (implémentation d'un objet)

        //4. Créer un objet CORBA et enregistrer implicitement le servant auprès du POA racine
        Object impl = root_poa.servant_to_reference(servant);

        //5. Exporter la nouvelle référence (impl) auprès des clients (par ex. via serveur de noms)

        //6. Activer le POA, c'est-à-dire le mettre en attente de requêtes
        root_poa.the_POAManager .activate ();

        //7. Activer l'ORB (boucle d'attente des requêtes venant des clients distants)
        orb.run
        …
    }
```

Utilisation d'un adaptateur d'objets : exemple (POA) en C++

```
int main (int argc, char **argv){
        // 1. Initialiser l'ORB
        CORBA::ORB_var orb =
                CORBA::ORB_init (argc, argv);

        //2. Obtenir ref d'objet pour le POA racine
         CORBA::Object_var obj =
                orb->resolve_initial_references ("RootPOA");
        Portable_Server::POA_var poa =
                 Portable_Server::POA_narrow (obj);

        //3. Créer un servant d'objet pour traiter les requêtes des clients
        Null_Servant_impl servant; // classe définie par ailleurs (implémentation d'un objet)

        //4. Créer un objet CORBA et enregistrer implicitement le servant auprès du POA racine
        Null_var null_impl = servant._this  ();

        //5. Exporter la nouvelle référence auprès des clients
                <à faire par exemple via serveur de noms>

        //6. Activer le POA, c'est-à-dire le mettre en attente de requêtes
        Portable_Server::POAManager_var poa_mgr =
                poa->the_POAManager ());
         poa_mgr.activate ();

        //7. Activer l'ORB (boucle d'attente des requêtes venant des clients distants)
        orb->run
        …
}
```

source : D.C. Schmidt, S. Vinoski, Object Interconnections SIGS C++ Report Magazine, 10, 4, April 1998

© 2006-2007, S. Krakowiak

Références d'objets

- **Définition (rappel)**
  - Référence = identification unique d'une instance d'objet (locale ou distante)
  - Un client ne peut appeler une opération sur un objet que s'il possède une référence désignant cet objet
- **Comment obtenir une référence**
  - Lors de la création de l'implémentation d'un objet (fonctions spécialisées fournies par l'adaptateur d'objets)
  - Par un service de noms
    - Service primitif de l'ORB : *string_to_object ()*
    - Services élaborés (CORBA Object Services)
      - Naming Service (pages blanches)
      - Trading Service (pages jaunes)

© 2006-2007, S. Krakowiak

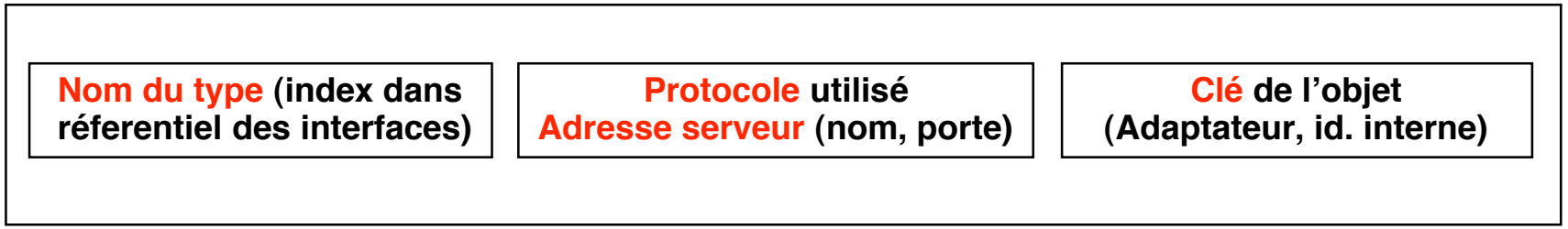
Références d'objets

Propriétés

- Une référence est opaque (son contenu n'est pas accessible aux clients)
- Une référence est utilisable via un protocole déterminé (éventuellement plusieurs protocoles peuvent être spécifiés) - exemple : IIOP
- Une référence peut être transitoire ou persistante
  - Transitoire : fonctionne tant que le serveur qui gère l'objet est en service
  - Persistante : indépendante de la durée de vie du serveur (le serveur peut être arrêté, déplacé, etc.).

© 2006-2007, S. Krakowiak

Contenu d'une référence d'objet (IOR)



Nom du type

Le type le plus détaillé (spécialisation) à la création de la référence

Protocole et adresse du serveur

La forme de l'adresse dépend du protocole

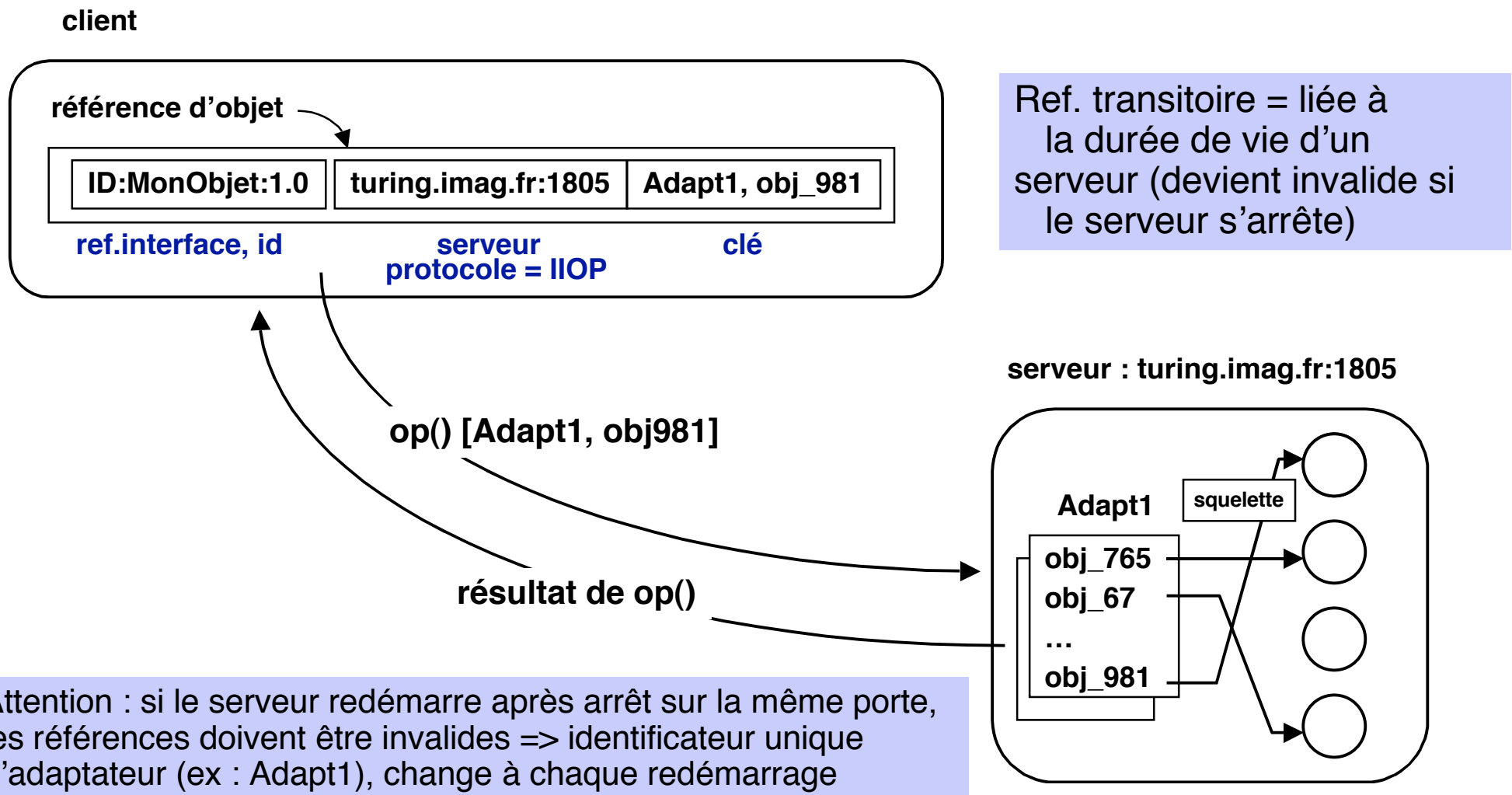
Clé de l'objet

La désignation de l'adaptateur est propre au serveur
Le format de l'id. interne est propre à l'adaptateur

On peut spécifier plusieurs couples (protocole et adresse, clé) pour la tolérance aux fautes (serveurs redondants)

© 2006-2007, S. Krakowiak

Interprétation des références “transitoires”



Ref. transitoire = liée à la durée de vie d’un serveur (devient invalide si le serveur s’arrête)

Attention : si le serveur redémarre après arrêt sur la même porte, les références doivent être invalides => identificateur unique d’adaptateur (ex : Adapt1), change à chaque redémarrage

Interprétation des références “persistantes” (1)

Ref. persistante = indépendante de la durée de vie d’un serveur (reste valide si le serveur s’arrête et redémarre, même sur une autre machine)

Référentiel des implémentations

- **maintient, sur un site serveur de référence (adresse et porte fixées), un annuaire des serveurs connus**
  - **identifie la localisation de chaque serveur (site, porte)**
  - **s’occupe de l’activation d’un serveur si nécessaire**

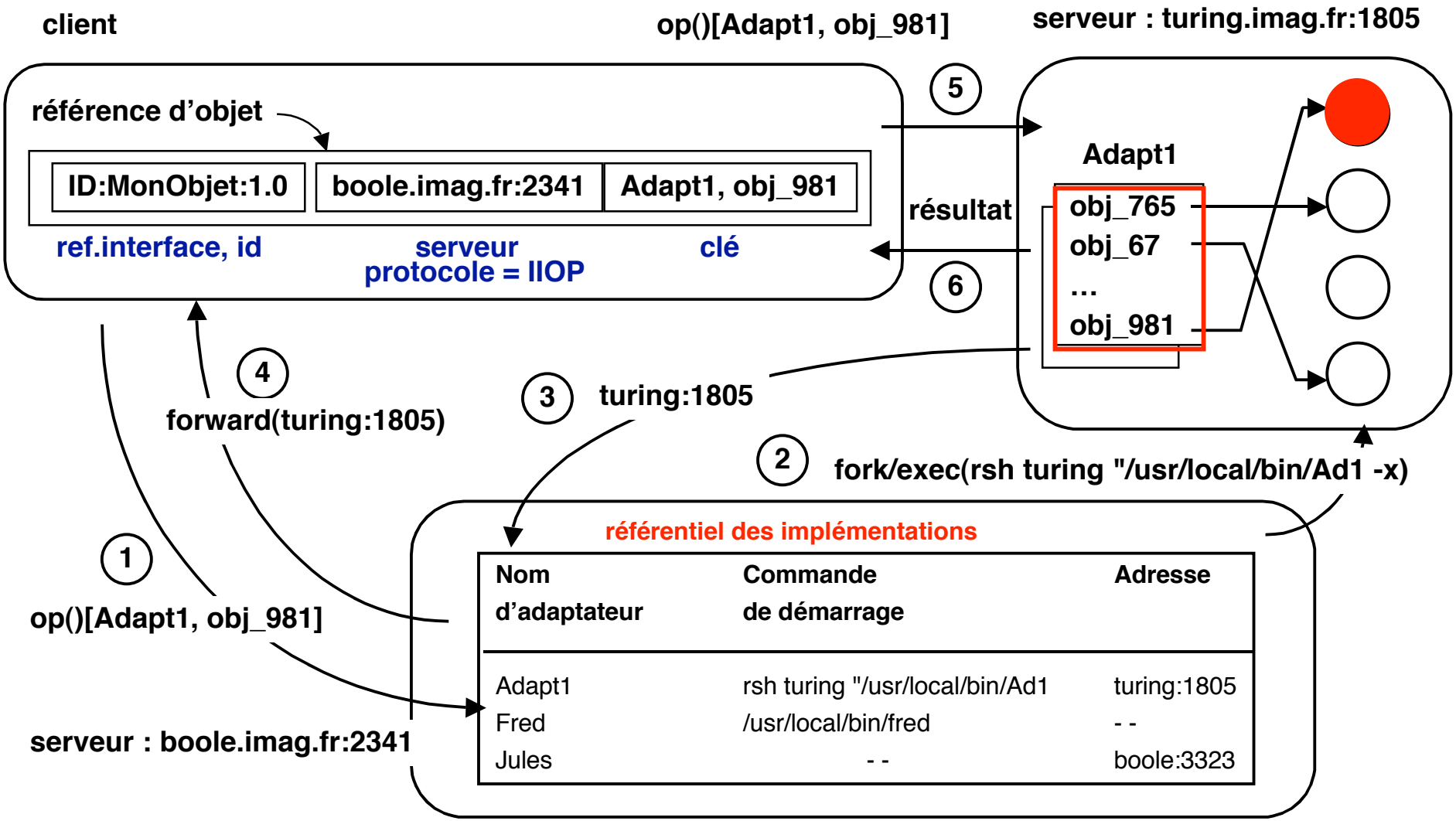
Nom d’adaptateur	Commande de démarrage	Adresse
Adapt1	rsh turing "/usr/local/bin/Ad1	turing:1789
Fred	/usr/local/bin/fred	- -
Jules	- -	boole:3323

local

lancé à la main

© 2006-2007, S. Krakowiak

Interprétation des références “persistantes” (2)



© 2006-2007, S. Krakowiak

Formes asynchrones de l'appel d'objets (1)

Appel standard (synchrone)

Utilise la méthode *invoke()* de l'interface *Request : req.invoke()*

Variante : appel à sens unique, sans résultat attendu (*oneway*)

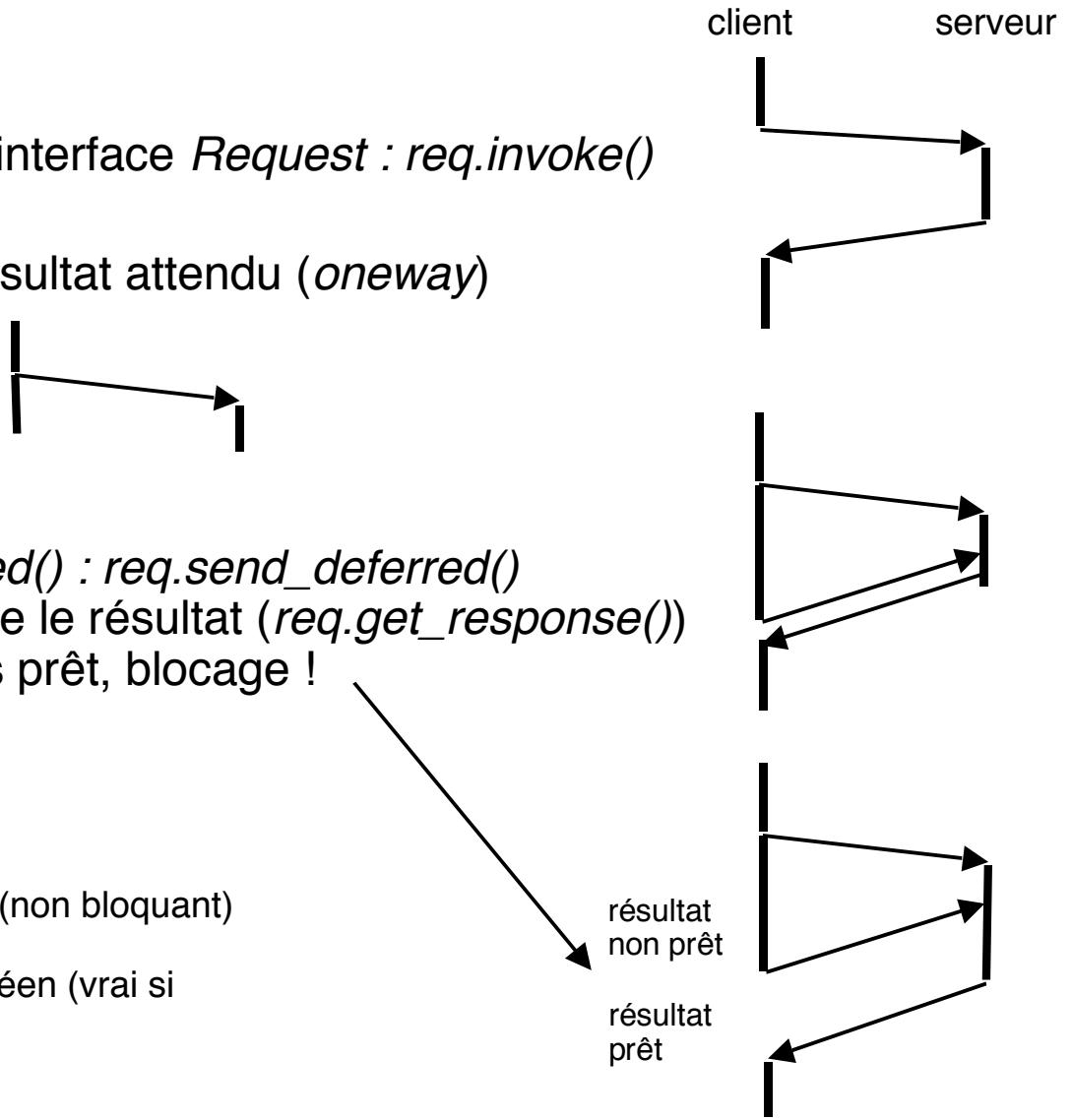
req.send_oneway()

Appel différé

Utilise la méthode *send_deferred() : req.send_deferred()*
Le client doit récupérer lui-même le résultat (*req.get_response()*)
mais si celui-ci n'est pas prêt, blocage !

Variante : **appel différé avec test**

Pour éviter le blocage, on peut faire un test (non bloquant) avant d'appeler *req.get_response()*
req.poll_response() renvoie un résultat booléen (vrai si résultats prêts, faux sinon)



Formes asynchrones de l'appel d'objets (2)

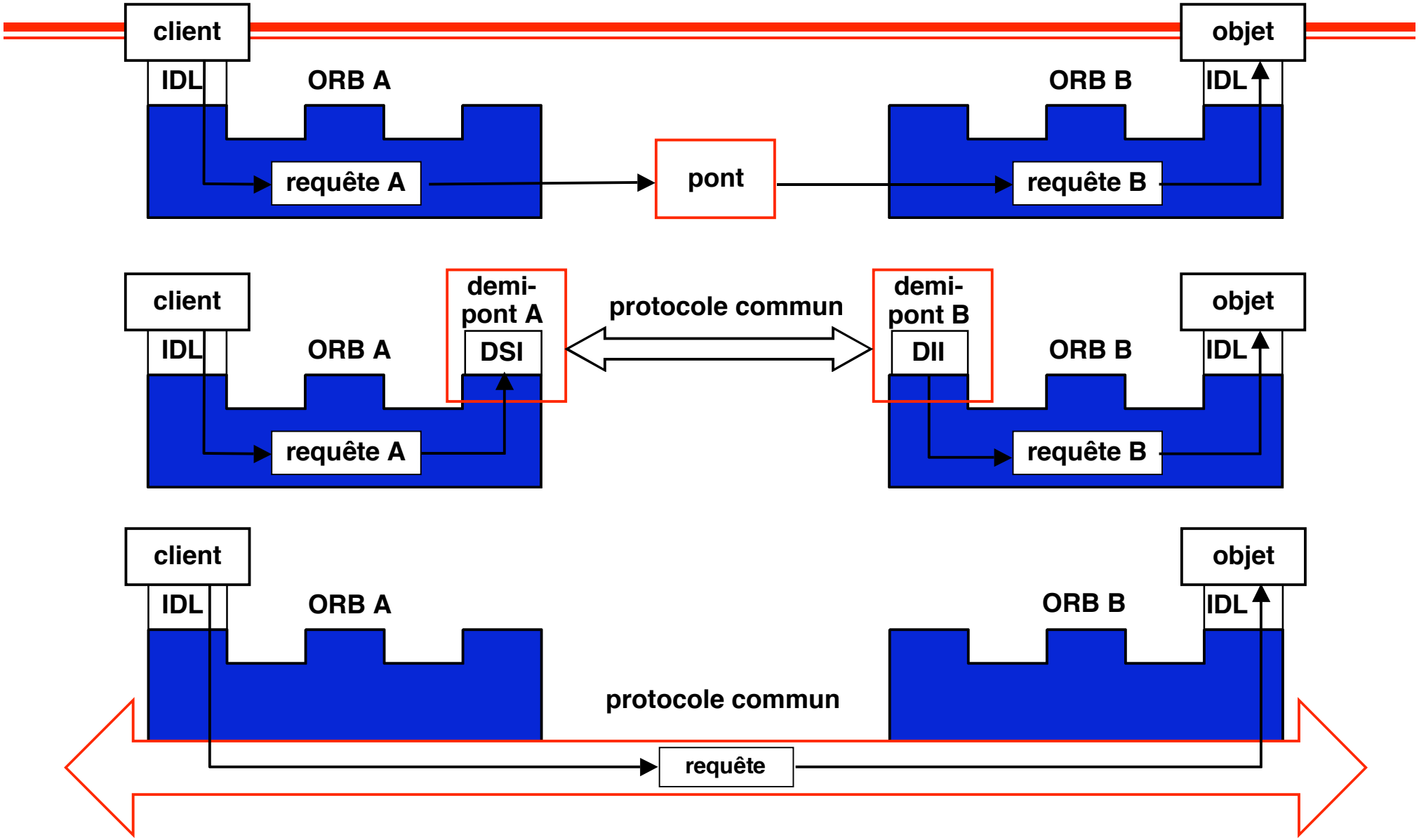
L'appel asynchrone nécessite la construction dynamique de requêtes

Interface IDL

```
Interface Annuaire {
        Int rechercher (in String nom) ;
        …
}
```

```
Soit annuaire un objet qui implémente l'interface Annuaire
…
req = annuaire._request("rechercher") ;          // requête pour appel de l'opération rechercher

Any param = req.add_in_arg() ;                   // définir paramètre
param.insert("Machin") ;

TypeCode type_retour =
        get_primitive_tc (TCKind.tk_int) ;       // définir type valeur de retour
req.set_return_type(type_retour) ;

req.send_deferred();                             // envoi de la requête
…                                                // faire autre chose
if (req.poll_response()) {                       // le résultat est prêt
        req.get_response() ;
        Any resultat = req.return_value() ;
        int numero = resultat.extract_int() ;
        …
} else {                                         // faire autre chose, revenir plus tard
}
```

Interconnexion de bus CORBA

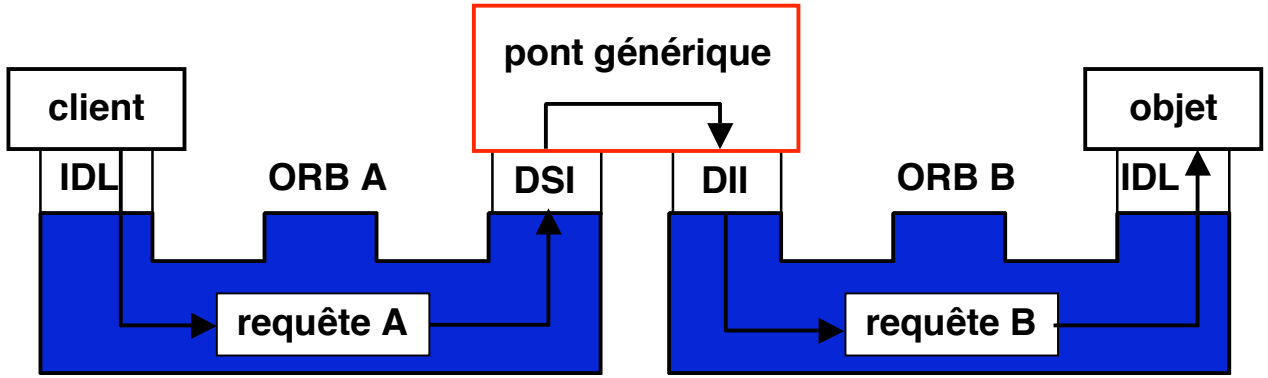


Protocole commun

- **GIOP (*General Inter-ORB Protocol*)**
  - Définit un petit nombre d'opérations génériques nécessaires au fonctionnement d'un ORB et indépendantes d'un protocole de communication ; définit des formats de messages et de représentation des données (références)
  - Doit faire l'objet d'une implémentation spécifique pour être utilisable
- **IIOP (*Internet Inter-ORB Protocol*)**
  - La forme spécifique du GIOP utilisant les protocoles de l'Internet (TCP, IP)
  - Utilisé non seulement pour communiquer entre les ORB, mais aussi à l'intérieur même d'un ORB (évite ainsi les conversions)
  - Utilisé aussi en dehors de CORBA (par exemple RMI sur IIOP, facilite l'interopérabilité)

© 2006-2007, S. Krakowiak

Exemple d'interconnexion : pont générique



N.B. Cet exemple est artificiel et sert à illustrer l'usage des interfaces DII et DSI

Fonction

intercepte les requêtes
les redirige vers l'objet destinataire

Intérêt de l'appel dynamique

pas de recompilation du pont quand on modifie les interfaces existantes
ou qu'on rajoute de nouvelles interfaces

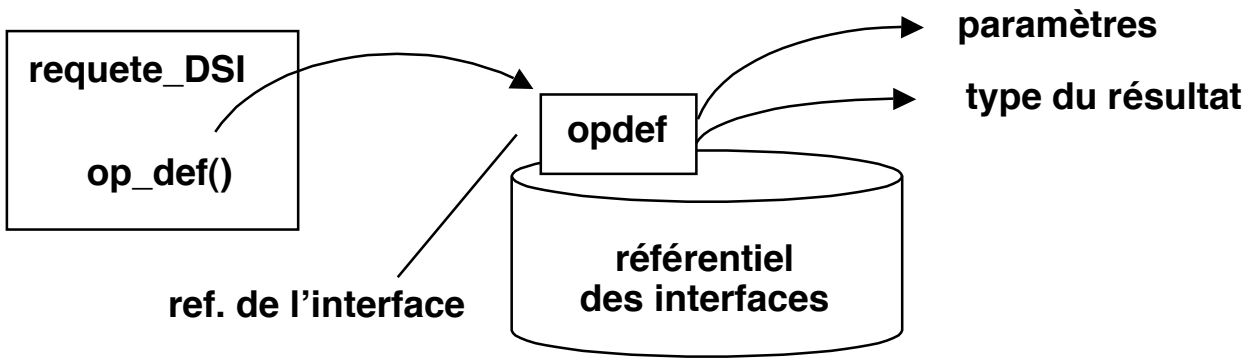
L' appel dynamique utilise le référentiel des interfaces pour trouver les signatures (types des paramètres) des requêtes à transmettre

© 2006-2007, S. Krakowiak

Recherche des signatures

Le pont (ProxyImpl) a une seule opération : invoke(CORBA_ServerRequest_ptr requete_DSI)

```
void ProxyImpl:: invoke(CORBA_ServerRequest_ptr requete_DSI )
{
        try {
                // 1. obtenir l'opération dans le référentiel des interfaces
                CORBA_OperationDef_var opdef = requete_DSI->op_def();

                // 2. tester si ce n'est pas une référence à nil
                if (CORBA_is_nil(opdef) return;

                // 3. obtenir la description des paramètres de  l'opération
                CORBA_ParDescriptionSeq_var parametres = opdef->params();

                // 4. obtenir le type du resultat retourné par l'opération
                CORBA_TypeCode_var resultat_type = opdef->result
```



© 2006-2007, S. Krakowiak

Extraction des arguments d'une requête DSI

```
// 5. créer une liste pour récupérer les arguments
CORBA_NVList_ptr nvlist;
_orb->create_list(parametres->length(), nvlist);

// 6. créer les arguments de la liste
for CORBA_ULong i = 0; i < parametres->length(); i++) {
        CORBA_Any *any;

        // 7. créer chaque argument selon son mode de passage
        switch ((*parametres[i].mode) {
        case CORBA_PARAM_IN:
                any = nvlist->add(CORBA_ARG_IN)->value();
                break;
        case CORBA_PARAM_INOUT:
                any = nvlist->add(CORBA_ARG_INOUT)->value();
                break;
        case CORBA_PARAM_OUT:
                any = nvlist->add(CORBA_ARG_OUT)->value();
                break;
        }
// 8. fixer le type de chaque argument
any->replace((*parametres[i].type, 0);`

// 9. récupérer seulement les paramètres IN et INOUT
requete_DSI->params(nvlist);
```

© 2006-2007, S. Krakowiak

Création d'une requête DII

```
// 10. créer une NamedValue pour contenir le résultat
CORBA_NamedValue_ptr resultat;
_orb->create_named_value(resultat);
resultat->value()->replace(resultat_type, 0);

// 11. créer une requête DII
CORBA_RequestVar requete_DII;
_objet->create_request (0, requete_DSI->op_name (),
        nvlist, resultat, 0, 0, requete_DII, 0);

// 12. invocation de la requête DII
requete_DII->invoke();
```

© 2006-2007, S. Krakowiak

Traitement du retour d'une requête DII

```
// 13. Y a-t-il eu une exception ?
CORBA_Exception* exception = requete_DII->env()-> exception();
if (exception) {
            CORBA_Any* any = new CORBA_Any;
           *any <<= *exception;

// 14. remplir l'exception de la requête DSI
requete_DSI-> exception(any);
return;

// 15. remplir les arguments INOUT et OUT de la requête DSI avec les valeurs retournées par la
requête DII
CORBA_NVList_ptr args = requete-DII->arguments();
for CORBA_ULong i = 0; i < nvlist->count(); i++; {
            if (nvlist->item(i)->flags() != CORBA_ARG_IN)
                        *(nvlist->item(i)->value()) = *(args->item(i)->value());
}

// 16. remplir la valeur du résultat de la requête DSI avec la valeur retournée par la requête DII
if (resultat_type != CORBA__tc_void)
            requete_DSI->result(requete_DII->result()->value());
```

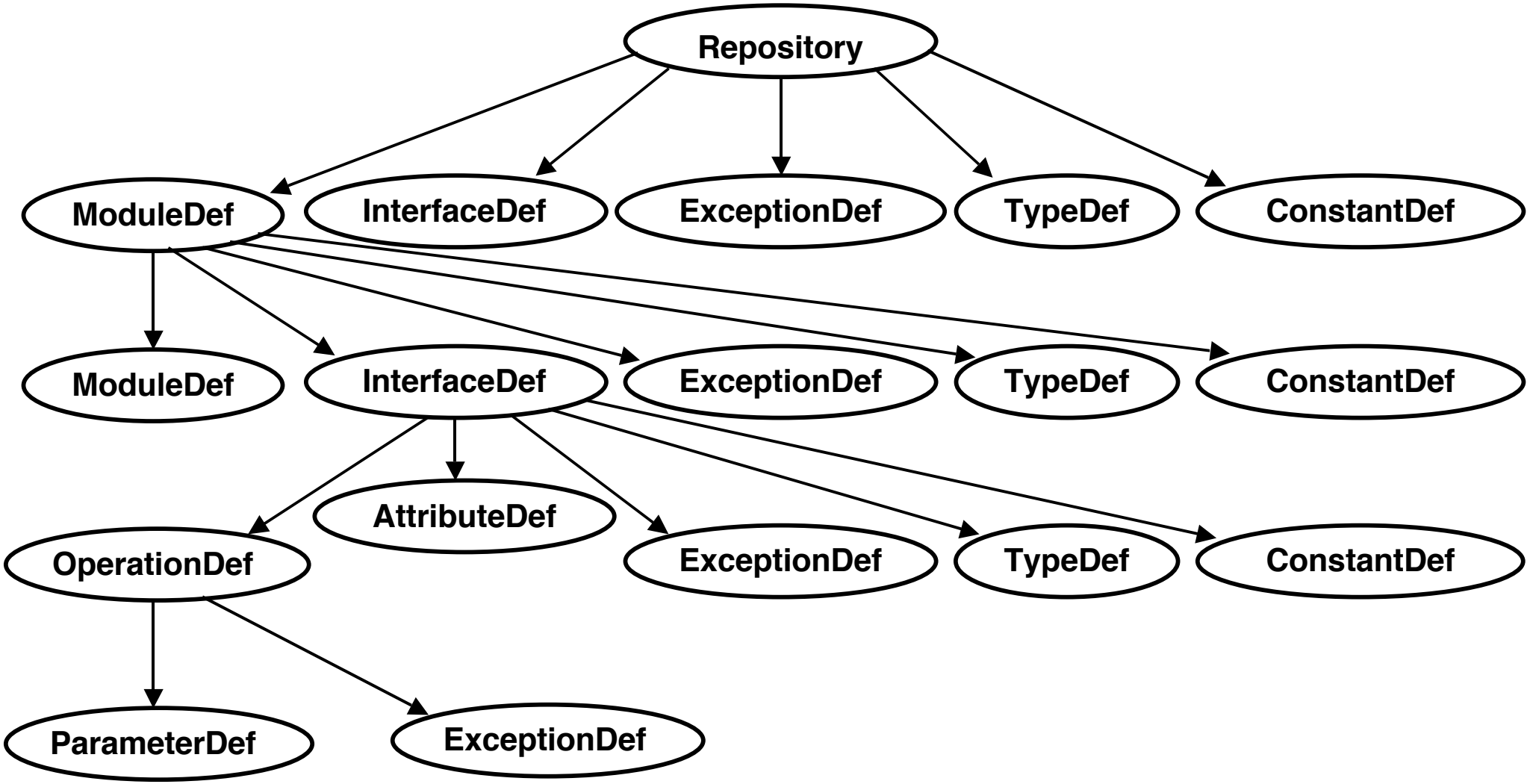
© 2006-2007, S. Krakowiak

Initialisation du pont générique

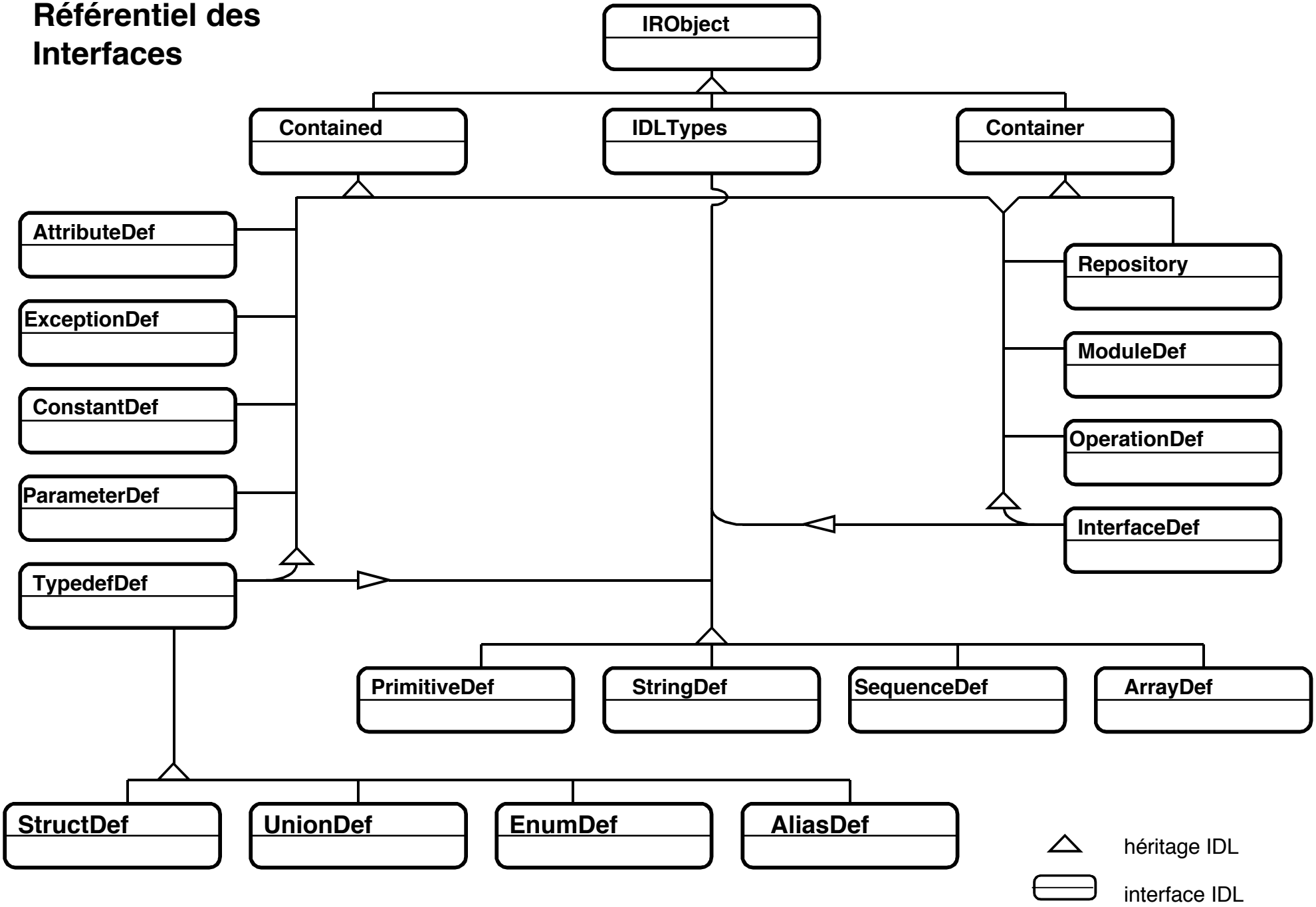
```
#include iostream.h
int main (int arg, char **argv); // l'IOR (sous forme chaîne) de l'objet destinataire est le paramètre argv[1]
{
        try {
                // 18. initialiser le bus CORBA
                CORBA_ORB_var orb = CORBA_ORB_init (argc, argv);

                // 19. initialiser l'adaptateur BOA
                CORBA_BOA_var boa = orb->BOA_init (argc, argv);

                // 20. convertir la chaîne IOR (de l'objet destinataire) en référence d'objet
                CORBA_Object_var objet = orb->string_to_object ( argv[1]);

                // 21. créer l'objet mandataire (proxy), c'est-à-dire le pont lui-même
                ProxyImpl* proxy = new ProxyImpl (objet, orb);

                // 22. convertir la référence du pont en chaîne iOR et l'imprimer sur sortie standard
                CORBA_STRING_var s = orb->object_to_string(proxy->_this());
                cout << s << endl;

                // 23. Le pont générique est prêt
                boa->impl_is_ready(CORBA_ImplementationDef::_nil());

        // récupérer l'exception éventuelle
        } catch (CORBA_Exception &) {}
return 0;
}
```

© 2006-2007, S. Krakowiak

Le graphe des objets du référentiel des interfaces

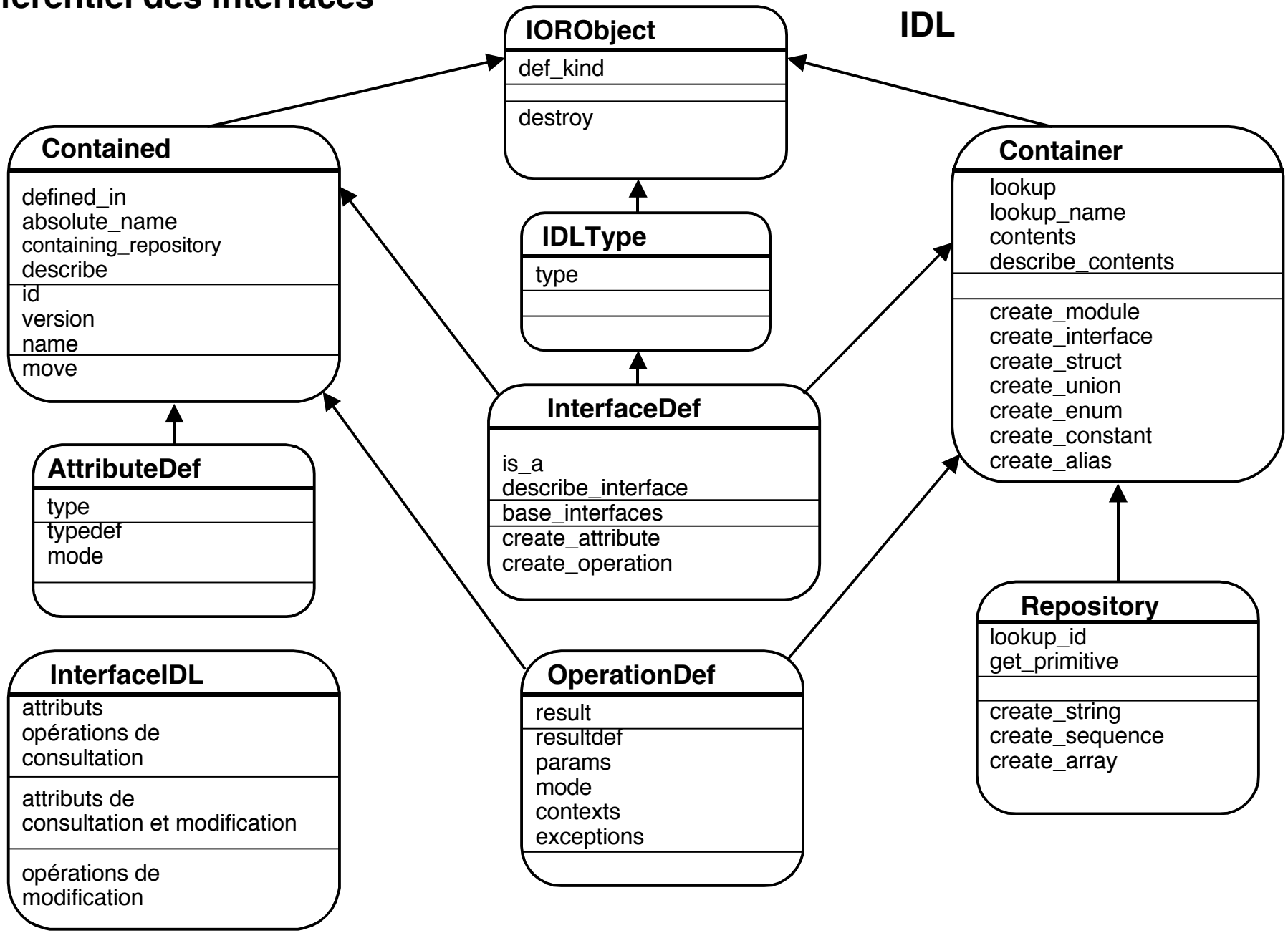


Référentiel des Interfaces

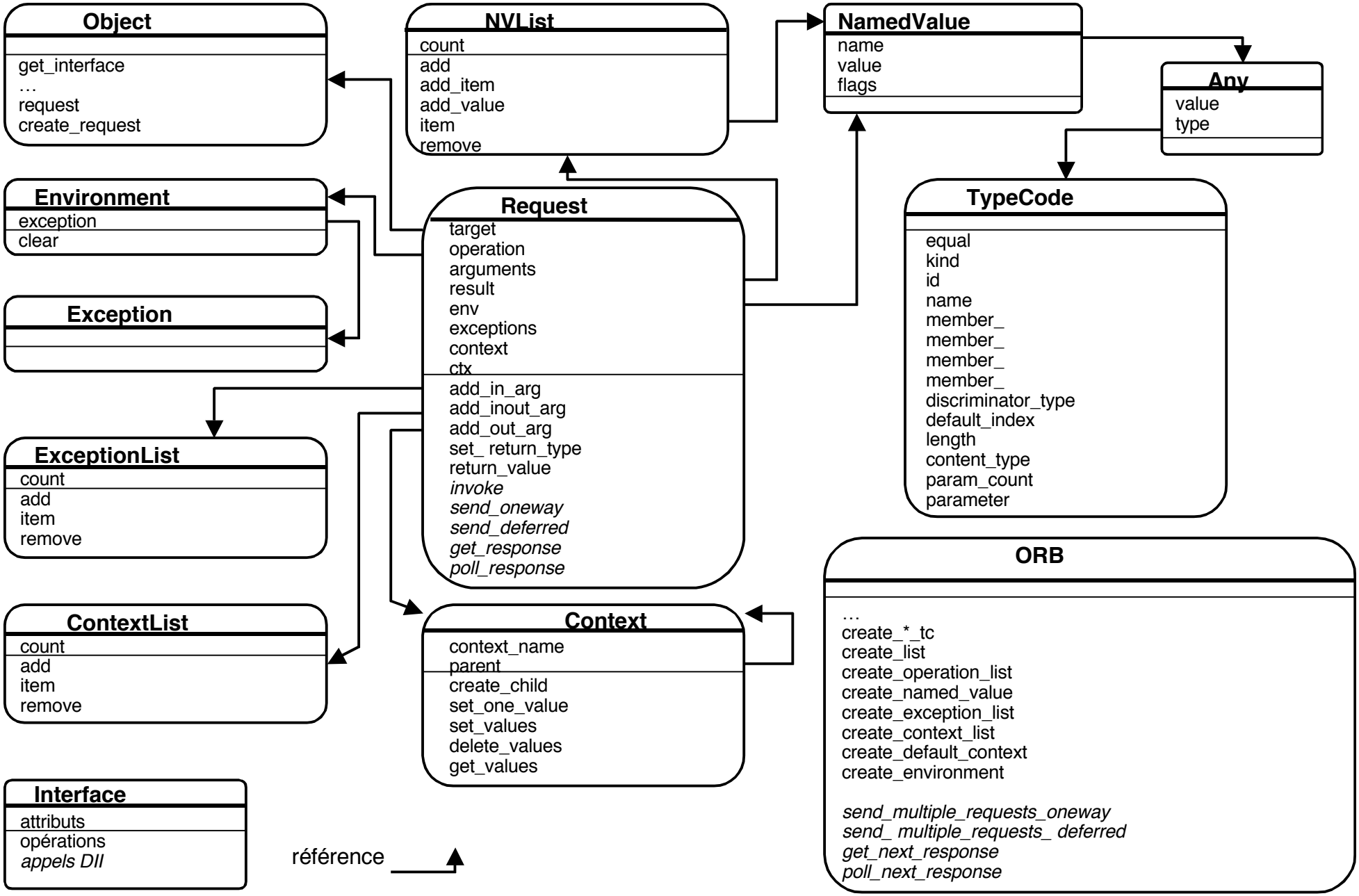


Référentiel des interfaces

Hiérarchie des interfaces IDL



Les interfaces du mécanisme DII (appel dynamique)



Le module CORBA et l'interface Object

```
// Définit par l'OMG.
#pragma prefix "omg.org"

// L'ensemble des composantes du bus CORBA.
module CORBA {

  exception COMM_FAILURE { … };
  // Autres exceptions systèmes.

  interface Object {

    // Duplique une référence d'objet CORBA.
    Object _duplicate();

    // Libère une référence d'objet.
    void _release();

    // Teste si une référence ne dénote aucun objet.
    boolean _is_nil();

    // Teste si un objet référencé n'existe plus.
    boolean _non_existent();

    // Teste si 2 références désignent la même IOR.
    boolean _is_equivalent(in Object that);

    // Calcule une clé de hachage.
    long _hash(in long maximum);

    // Teste si un objet est d'un type donné.
    boolean _is_a(in string type_identifier);

    // Autres opérations des mécanismes dynamiques.
    InterfaceDef _get_interface();
    Request _request(in string s);
    Request _create_( . . . );
    Request _create_request2( . . .);
  };
};
```

L'interface ORB et BOA

```
module CORBA {
  // non standardisé par l'OMG.
  interface ImplementationDef { … };

  interface BOA {
    // Se mettre en attente des requêtes aux objets.
    void impl_is_ready (in ImplementationDef impl);
  };

  interface ORB {
    // Une référence d'objet vers une chaîne IOR.
    string object_to_string (in Object obj);

    // Une chaîne IOR vers une référence d'objet.
    Object string_to_object (in string str);

    // La liste des objets notoires.
    typedef string ObjectId;
    typedef sequence<ObjectId> ObjectIdList;
    ObjectIdList list_initial_services ();

    // Obtenir un objet notoire.
    Exception InvalidName {};
    Object resolve_initial_references (
         in ObjectId identifier) raises(InvalidName);

    // Obtenir l'adaptateur d'objets.
    typedef sequence<string> arg_list;
    typedef string OAid;
    BOA BOA_init (inout arg_list argv,
                              in OAid oa_identifier);

    // Autres opérations.
  };

  // Obtenir l'ORB.
  typedef string ORBid;
  typedef sequence<string> arg_list;
  ORB ORB_init (inout arg_list argv,
                            in ORBid orb_identifier);
```