



Introduction CORBA

Alexandre Denis – Alexandre.Denis@inria.fr

**Inria Bordeaux – Sud-Ouest
France**

Contexte

- Besoins
 - Interconnexion de systèmes d'information
 - Réutilisation de codes existants
- Hétérogénéité -> interopérabilité
 - Matériel, langage, OS, etc.
- Un **standard** d'architecture d'applications réparties
 - Intéropérabilité entre plate-formes, langages, etc.

OMG – Object Management Group

- Consortium à but non-lucratif fondé en 1989
 - Plus de 850 membres (constructeurs, SSI, utilisateurs, recherche)
- **Standards** pour les applications réparties
- Fonctionnement
 - Propositions, discussions, vote
 - -> **spécifications**
- Ne fournit pas d'implémentation
- ex.: OMA, CORBA, UML, MOF



Vue d'ensemble de CORBA

- CORBA - Common Object Request Broker Architecture
 - CORBA 1.0 – 1991, modèle objet
 - CORBA 2.0 – 1995, interopérabilité, IIOP
 - CORBA 3.0 – 2002, modèle composant
(dernière révision complète : CORBA 3.3 – 2012)
 - Révisions de chapitres
 - IDL 4.2 – mars 2018
 - Binding C++11 v1.3 – mars 2018
- Paradigme unifié : appel à des objets distants
- Langage d'interface unifié : IDL
- Large collection de services communs
 - Présentés sous forme d'objets CORBA



Propriétés de CORBA

- Intergiciel suivant un standard ouvert
 - Plusieurs implémentations
 - ex.: omniORB, MICO, TAO, ORBacus, JacORB, ORBexpress, ORBit, VBOrb, R2CORBA, IIOP.NET, JDK, ...
- **Interopérabilité**
 - Entre langages
 - Entre machines, entre OS
 - Entre constructeurs

ORB

- ORB – bus logiciel entre applications

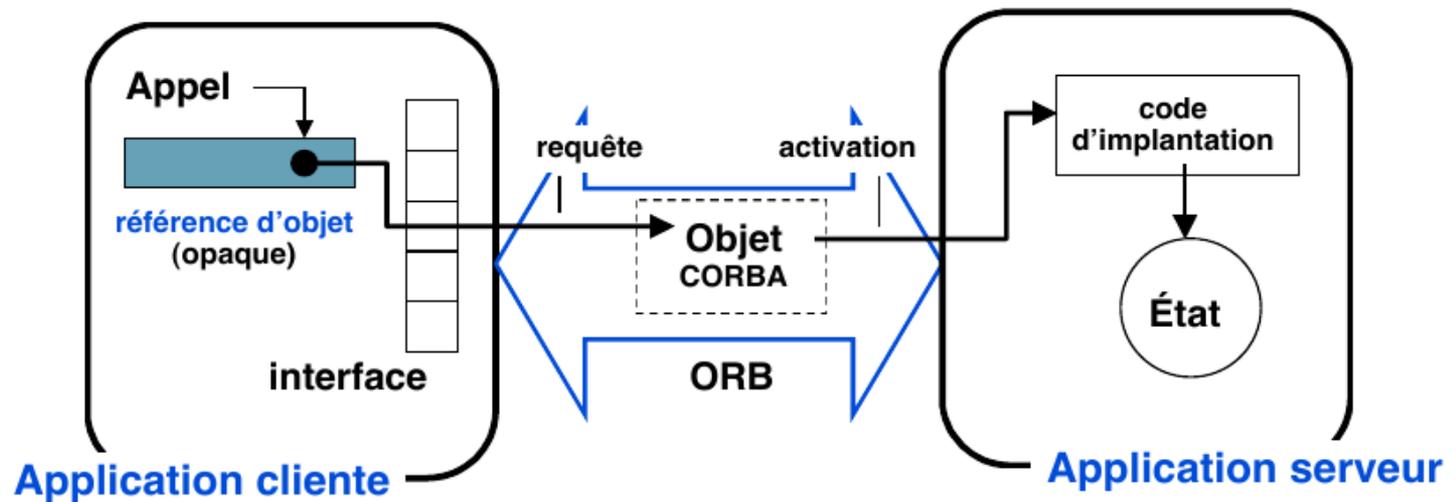


Figure: Krakowiak

Objet CORBA

- Définition
 - Entité logicielle désignée par une référence, recevant les requêtes émises par les applications clientes
- Objet CORBA
 - **Interface IDL**
 - **Implémentation** – *servant*
 - Classe et instance
 - **Référence** localisant l'objet sur le réseau
 - IOR – Interoperable Reference

Protocoles et interopérabilité

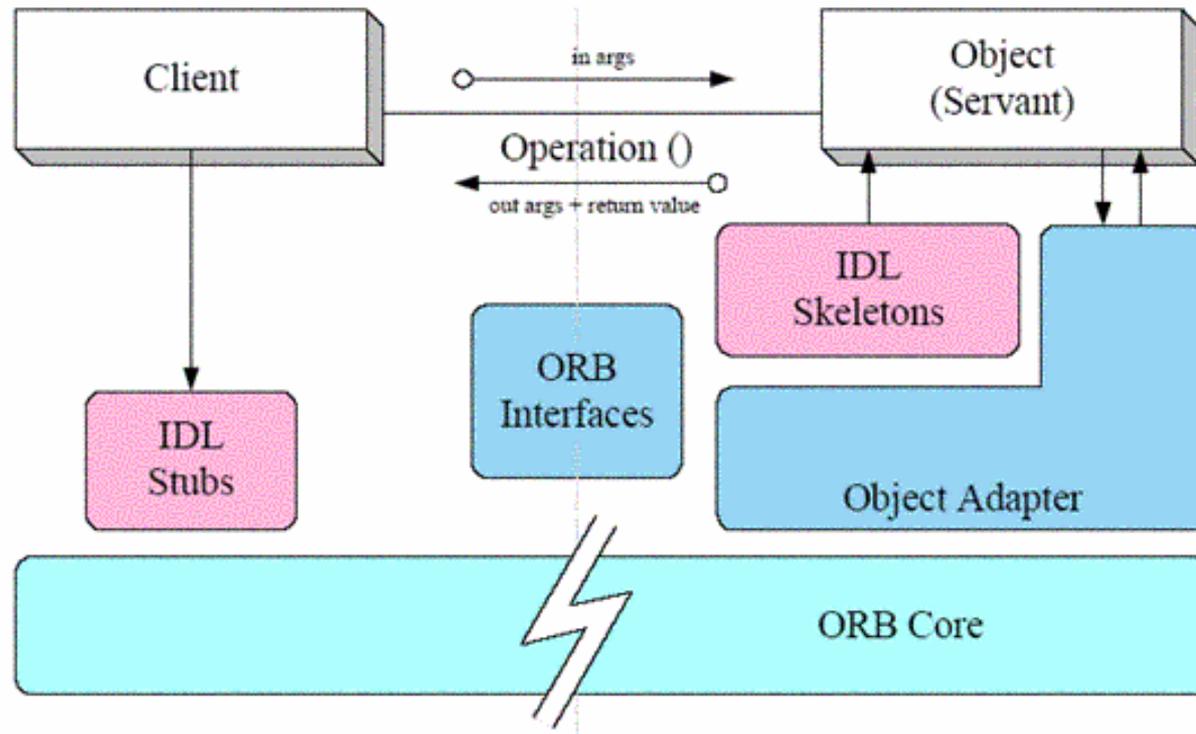
- GIOP – General Inter-ORB Protocol
 - Protocole générique de transport pour CORBA
 - Représentation des données commune : CDR
 - Référence des objets : IOR – Inter-Operable Reference
 - Contient : adresse de la machine, version de la couche transport, clef identifiant l'objet (*servant*) dans le serveur
 - Format des messages
- IIOP – Internet Inter-ORB Protocol

Protocoles et interopérabilité

- GIOP – General Inter-ORB Protocol
- IIOP – Internet Inter-ORB Protocol
 - Instanciation de GIOP sur TCP/IP
 - **Interopérable** entre implémentations
 - Plus que GIOP/socket
 - Multiplexage, keep-alive, etc.
- ESIOP - Environment Specific Inter-ORB Protocol
 - Possibilité de protocole spécifique, non-intéropérable

Vue d'ensemble

- ORB, stub, skeleton, servant



OMG IDL – Interface Definition Language

- Langage de **description des interfaces**
 - Indépendant du langage d'implémentation
 - Purement déclaratif
 - Sépare l'interface et l'implémentation
 - « **Projections** » vers les langages d'implémentation
- Syntaxe proche de C++, Java

OMG IDL – Exemple

- Exemple d'interface en IDL

```
module Finance
{
  typedef sequence<string> StringSeq;
  struct AccountDetails
  {
    string      name;
    StringSeq   address;
    long        account_number;
    double      current_balance;
  };
  exception insufficientFunds { };
  interface Account
  {
    void deposit(in double amount);
    void withdraw(in double amount) raises(insufficientFunds);
    readonly attribute AccountDetails details;
  };
};
```

OMG IDL – Elements

- Éléments de base
 - `module` – ensemble de définitions
 - `interface` – interface d'objet, avec héritage
 - `const` - constante
 - `enum` - type énuméré
 - `typedef` – déclaration de type
 - `struct` – type structuré
 - `sequence` – type indexé (tableau)
 - `attribute` – attribut d'interface, éventuellement `readonly`
 -

OMG IDL – Elements, suite

- Éléments de base
 - Opérations de l'interface
 - Exemple :
`type fonction(in int a, out int b);`
 - **Sens de passage des paramètres explicite** : in, out, inout
 - exception – définition d'exceptions
 - Types primitifs
 - void, short, long, long long, float, double, long double, boolean, octet, string
 - Projetés vers les types primitifs du langage

Stubs, squelettes

- Génération automatique de stub à partir de l'IDL
 - *Stub* = souche, talon ; côté client
 - *Skeleton* = squelette ; côté serveur

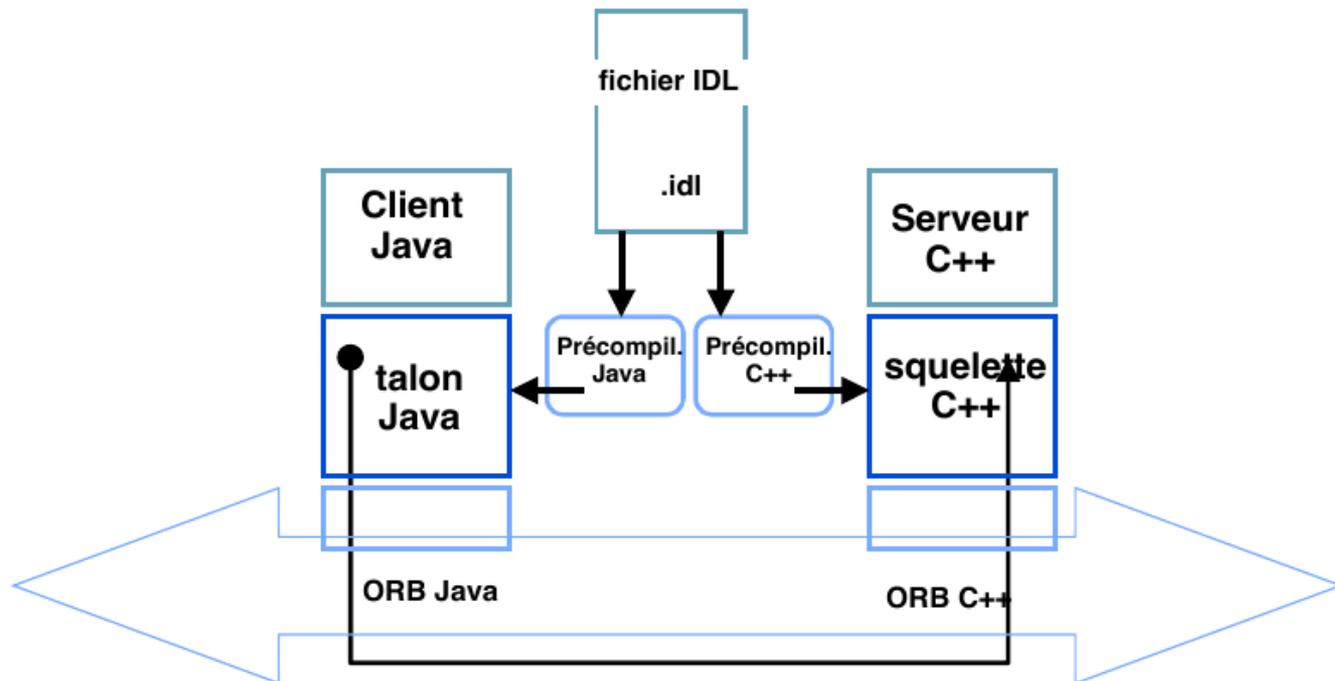
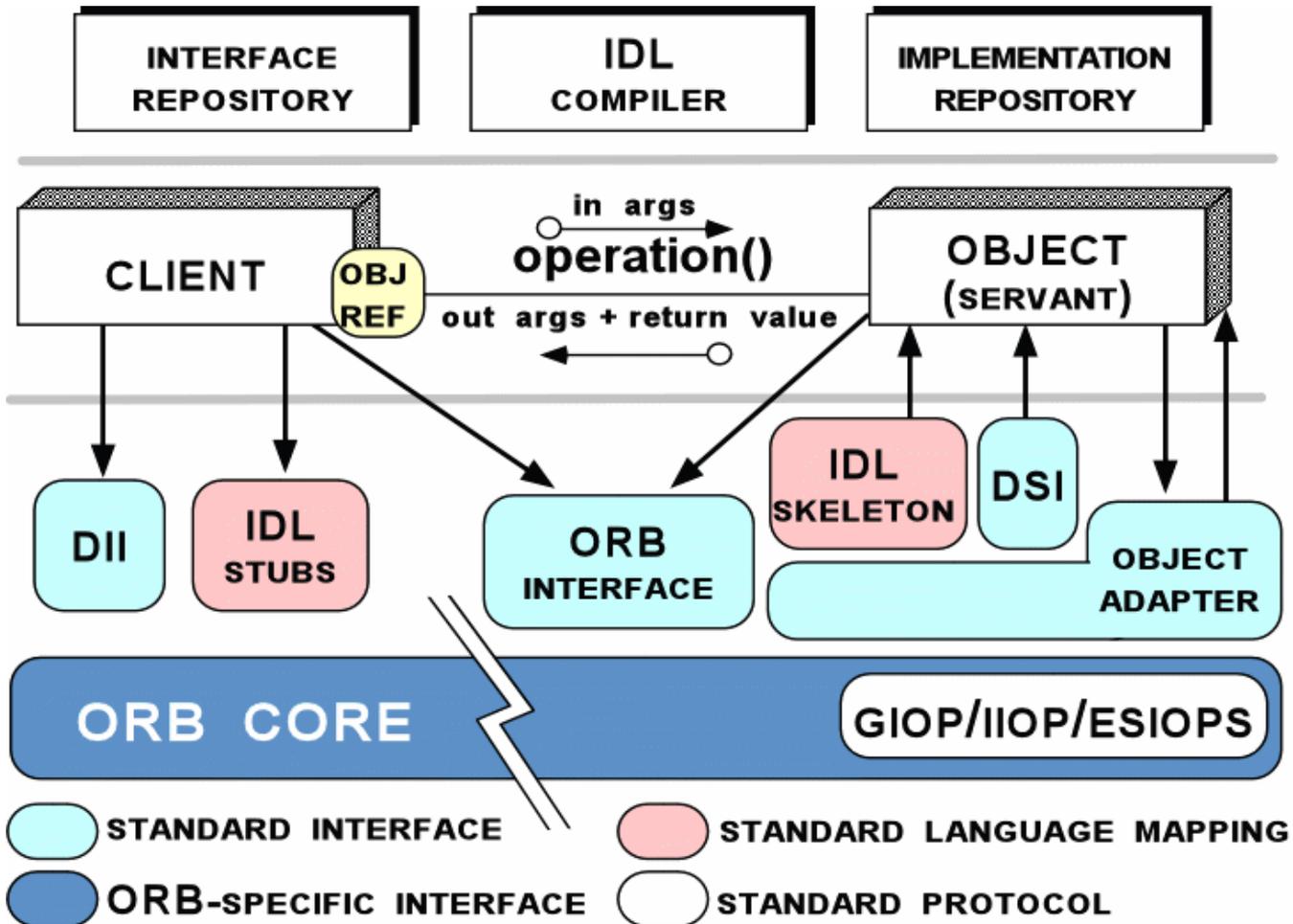


Figure: Krakowiak

Anatomie de CORBA



Interface ORB

- Les fonctionnalités CORBA sont décrites en IDL
 - Projection vers le langage cible
- L'ORB lui-même est accessible via une **référence d'objet**
 - Obtenue à l'initialisation
 - `orb = CORBA::ORB_Init(argc, argv);`
 - Paramétré via la ligne de commande
- Fournit des services de base
 - Références initiales vers des objets connus
 - Gestion de l'**adaptateur d'objets**

Adaptateur d'objet

- Gère la liaison entre *servant* et ORB ~ **démultiplexeur**
 - Gère le mapping des objets -> servant
 - Génération des références
 - Adapte les spécificités du langage à l'ORB
 - Durée de vie des objets
 - Activation de processus
 - Modèle de threads
- Implémentations
 - BOA, CORBA 1.0 – spécifique à chaque implémentation
 - POA, CORBA 2.0 – interface standard
 - Plusieurs POA peuvent être instanciés

Référence

- Désignation unique d'un objet
 - Opaque pour l'utilisateur, pas directement une URL
 - URL possible pour les références initiales
 - `corbaloc`, `corbaname`
- Obtenue
 - À l'activation d'un objet (par le POA)
 - Par l'intermédiaire du serveur de nom (*Naming Service*)
 - Conversion string <-> référence
 - À l'initialisation
 - Référence vers l'ORB, vers le POA, vers le service de nom

Références IOR, exemples

- IOR : Interoperable Reference
 - Forme binaire
 - Forme « *stringifiée* » (~sérialisée)

```
IOR:00000000000000001749444c3a48656c6c6f4170702f48656c6c6f3a312e300000000000  
10000000000000006c000102000000000d3134372e3231302e31382e310000df7400000021af  
abcb00000000209db9e72400000001000000000000000000000000004000000000a0000000000  
001000000010000002000000000000100010000000205010001000100200001010900000001  
00010100
```

- En référence initiale : `printf` de l'IOR (ou sauvegarde dans un fichier)
 - Simple et rustique

Références initiales

- **corbaloc – URL absolue**
 - `corbaloc:iiop:1.2@host1:3075/NameService`
 - Protocole : iiop; version 1.2 ; serveur : host1 ; port : 3075; étiquette (telle que passé à `resolve_initial_reference`) : NameService
 - `corbaloc::host1:2809/NameService`
 - Sert principalement à donner la référence du NameService
- **corbaname – URL dans le NameService**
 - `corbaname::foo.bar.com:2809/NameService#x/y`
 - NameService tourne sur `foo.bar.com:2809`, objet `y` dans le contexte `x`
- Ligne de commande standardisée
 - Forme générale `-ORBInitRef etiquette=URL`
 - `-ORBInitRef NameService=corbaloc::host1:3075/NameService`

COS – services communs

- CORBA Services – Cos
 - Implémentés sous forme d'**objets CORBA**, interfaces en IDL
- Service de nommage :
 - Module CosNaming, objet NameService
- Trade service – résolution par type ~ pages jaunes
 - Module CosTrading
- Service d'évènements - EventService
 - Modèle push/pull, couplage faible
- Object Transaction Service – OTS
 - begin, commit, rollback
 - Objet TransactionCurrent

Service de nommage - NameService

- Annuaire contenant des références d'objets CORBA
 - Association : nom -> IOR
 - Organisé en arborescence, comme un système de fichiers
 - Répertoire -> NamingContext
 - Chemin -> Name
 - Système de nommage nom.extension (id.kind)
- Référence initiale (bootstrap)
 - Opération
`CORBA::ORB::resolve_initial_references("NameService")`
 - Conversion de **CORBA::Object** vers `CosNaming::NamingContext`
 - Opération CORBA `_narrow`

Service de nommage – CosNaming.idl

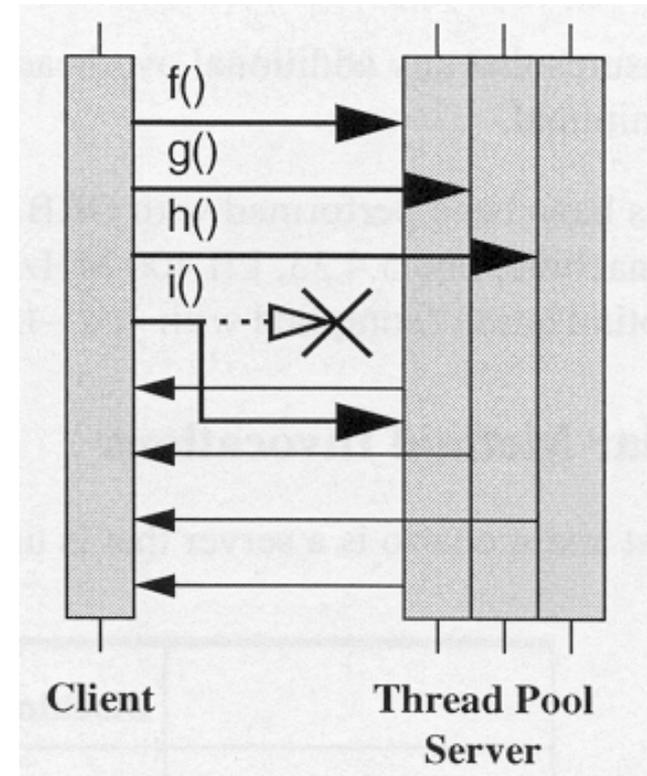
```
module CosNaming {
    typedef string Istring;
    struct NameComponent {
        Istring id;
        Istring kind;
    };
    typedef sequence<NameComponent> Name;
    enum BindingType {nobject, ncontext};
    struct Binding {
        Name binding_name;
        BindingType binding_type;
    };
    typedef sequence <Binding> BindingList;

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

```
interface NamingContext {
    void bind(in Name n, in Object obj) raises(...);
    void rebind(in Name n, in Object obj) raises(...);
    void bind_context(in Name n, in NamingContext nc)
        raises(...);
    void rebind_context(in Name n,
                      in NamingContext nc) raises(...);
    Object resolve(in Name n) raises(...);
    void unbind(in Name n) raises(...);
    NamingContext new_context();
    NamingContext bind_new_context(in Name n)
        raises(...);
    void destroy() raises(...);
    void list(in unsigned long how_many,
             out BindingList bl,
             out BindingIterator bi);
};
... // interface NamingContextExt omitted for brevity
};
```

Modèle d'exécution

- Modèle de thread
 - **Dépend de l'implémentation**
 - Single thread
 - ex.: MICO
 - Thread par connexion
 - Thread par requête
 - Thread pool
 - ex.: Java
 - Configurable *via* le POA
 - ex.: omniORB, TAO



Exemple : Hello world !

- Interface Echo.idl

```
interface Echo
{
    string echoString(in string mesg);
};
```

Exemple : compilation (java)

- Compilation de l'IDL en Java
 - `idlj -fall Echo.idl`
 - Génère les parties client (`-fclient`), serveur (`-fserver`), ou les deux (`-fall`)
 - `EchoOperations.java` – projection de l'interface en Java
 - `EchoStub.java` – souche client
 - `EchoPOA.java` – squelette serveur
 - + divers helpers
- Compilation
 - `javac *.java`

Exemple : servent (java)

- Implémentation servent en Java : EchoImpl.java
 - Implémentation concrète de l'interface
 - Antisèche (pour les projections de type) : allez voir EchoOperations.java !

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;
import java.util.Properties;

class EchoImpl extends EchoPOA
{
    public String echoString(String msg)
    {
        System.out.println("msg: " + msg);
        return msg;
    }
}
```

Exemple : serveur (java)

```
public class EchoServer
{
    public static void main(String args[])
    {
        ORB orb = ORB.init(args, null); // create and initialize the ORB
        // get reference to rootpoa & activate the POAManager
        org.omg.CORBA.Object objRef = orb.resolve_initial_references("RootPOA");
        POA rootpoa = POAHelper.narrow(objRef);
        rootpoa.the_POAManager().activate();

        // get the naming service
        objRef = orb.resolve_initial_references("NameService");
        NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

        // instanciate the servant
        EchoImpl echoImpl = new EchoImpl();
        // get object reference from servant
        objRef = rootpoa.servant_to_reference(echoImpl);
        // convert the generic CORBA object reference into typed Echo reference
        Echo echoRef = EchoHelper.narrow(objRef);

        // bind the object reference in the naming service
        NameComponent path[ ] = ncRef.to_name("echo.echo"); // id.kind
        ncRef.rebind(path, echoRef);

        orb.run(); // start server...
    }
}
```

Exemple : client (java)

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

public class EchoClient
{
    public static void main(String args[])
    {
        org.omg.CORBA.Object objRef;

        ORB orb = ORB.init(args, null); // create and initialize the ORB

        // get the naming service
        objRef = orb.resolve_initial_references("NameService");
        NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
        // resolve the object reference from the naming service
        objRef = ncRef.resolve_str("echo.echo");

        // convert the CORBA object reference into Echo reference
        Echo echoRef = EchoHelper.narrow(objRef);

        // remote method invocation
        String response = echoRef.echoString("coucou");
        System.out.println(response);
    }
}
```

Exemple : déploiement (java)

- Déploiement
 - Démarrer le serveur de nom :
 - `tnameserv -ORBInitialPort 2810`
 - Lancer le serveur
 - `java EchoServer -ORBInitRef NameService=corbaloc::host:2810/NameService`
 - Lancer le client
 - `java EchoClient -ORBInitRef NameService=corbaloc::host:2810/NameService`

Exemple : servant C++

- Servant

```
#include <CORBA.h>
#include <Naming.hh>
#include "Echo.hh"
using namespace std;

class EchoImpl : public POA_Echo,
                 public PortableServer::RefCountServantBase
{
public:

    virtual char* echoString(const char* msg)
    {
        cout << msg << endl;
        return CORBA::string_dup(msg);
    }
};
```

Exemple : serveur C++

- Serveur

```
int main(int argc, char** argv)
{
    CORBA::Object_var objRef;
    // create and initialize the ORB
    CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
    // get reference to rootpoa & activate the POAManager
    objRef = orb->resolve_initial_references("RootPOA");
    PortableServer::POA_var poaRef = PortableServer::POA::_narrow(objRef);
    poaRef->the_POAManager()->activate();
    // get the naming service
    objRef = orb->resolve_initial_references("NameService");
    CosNaming::NamingContext_var ncRef =
        CosNaming::NamingContext::_narrow(objRef);
    // instantiate the Echo CORBA object
    EchoImpl * echoImpl = new EchoImpl( );
    Echo_var echoRef = echoImpl->_this();
    // bind the object reference in the naming service
    CosNaming::Name name;
    name.length(1);
    name[0].id = (const char*)"echo";
    name[0].kind = (const char*)"echo";
    ncRef->rebind(name, echoRef);
    // start server...
    orb->run();
    return EXIT_SUCCESS;
}
```

Exemple : client C++

- Client

```
#include <CORBA.h>
#include <Naming.hh>
#include "Echo.hh"

int main (int argc, char **argv)
{
    CORBA::Object_var objRef;
    // initialize the ORB
    CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
    // get the naming service
    objRef = orb->resolve_initial_references("NameService");
    CosNaming::NamingContext_var nsRef =
        CosNaming::NamingContext::_narrow(objRef);

    // resolve the "echo" CORBA object from the naming service
    CosNaming::Name name; name.length(1);
    name[0].id = (const char*) "echo";
    name[0].kind = (const char*) "echo";
    objRef = nsRef->resolve(name);
    Echo_var echoRef = Echo::_narrow(objRef);

    // remote method invokation
    cout << echoRef->echoString("coucou") << endl;

    return 0;
}
```

Compilation C++ - omniORB

- Utilisation de l'implémentation CORBA omniORB
- Compilation IDL
 - `omniidl -bcxx Echo.idl`
 - Génère `EchoSK.cc` et `Echo.hh`
- Serveur de nom
 - `omniNames -start`
- Support Python

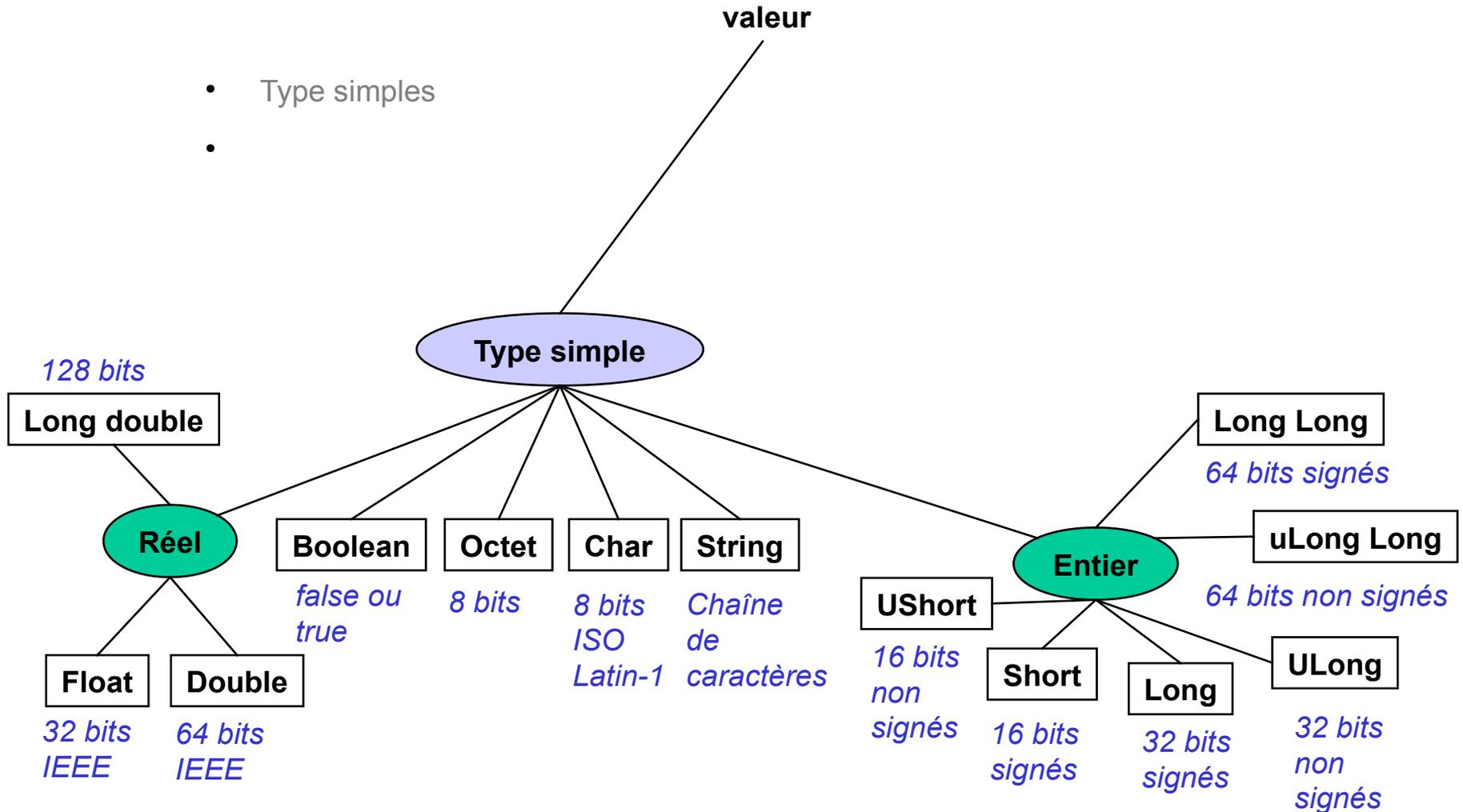
Fonctions avancées CORBA 2.x

CORBA avancé

- Interface dynamique
- Type Any, DynAny
- Valuetype
- Pointeur intelligent *_var
- POA avancé : ServantLocator, ServantActivator
- CosEvent
- Persistance
-

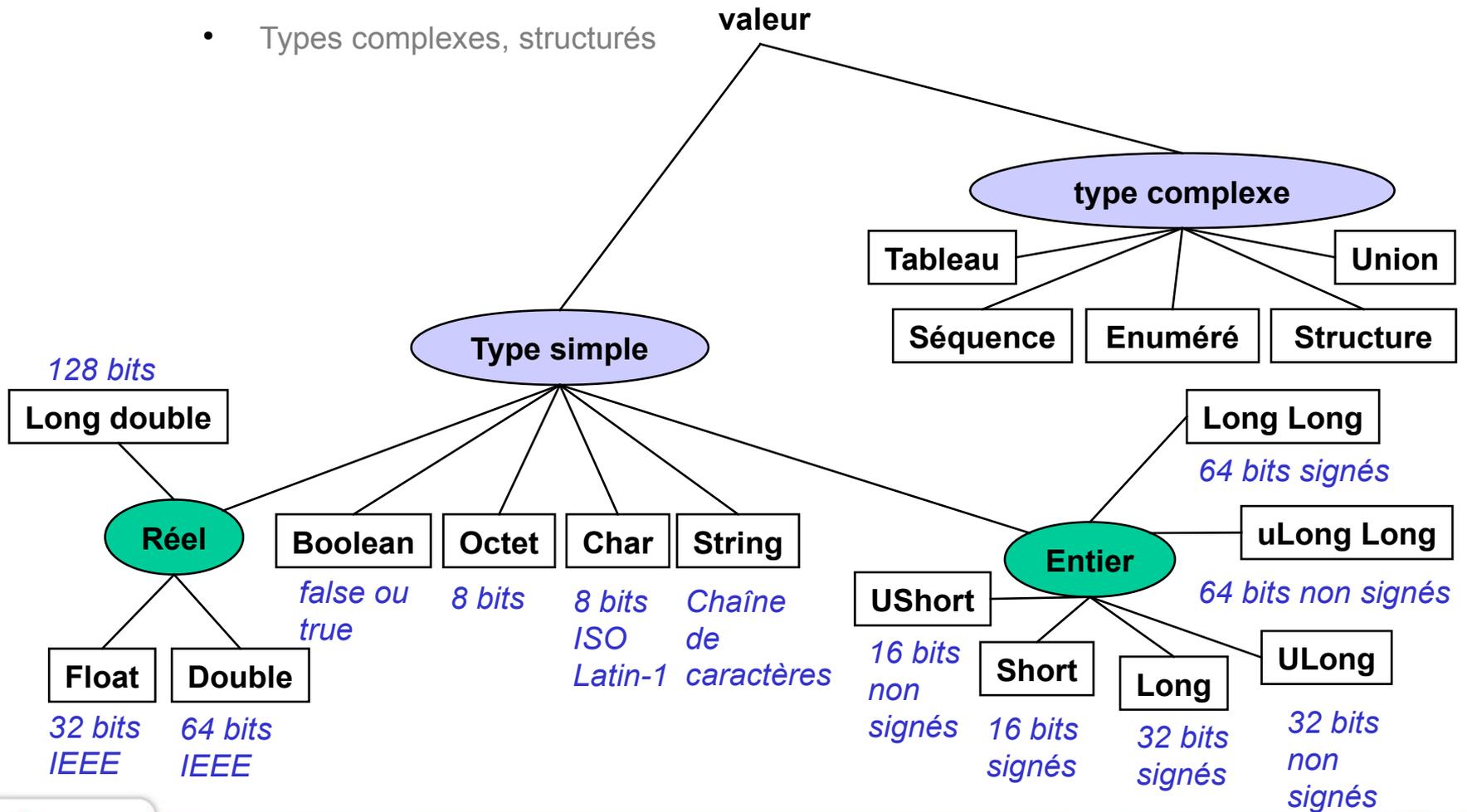
Types IDL

- Type simples
-



Types IDL

- Types complexes, structurés



Séquences

- Tableau
 - Taille fixe uniquement (donc très peu utilisé!)
 - Définition obligatoire d'un type !

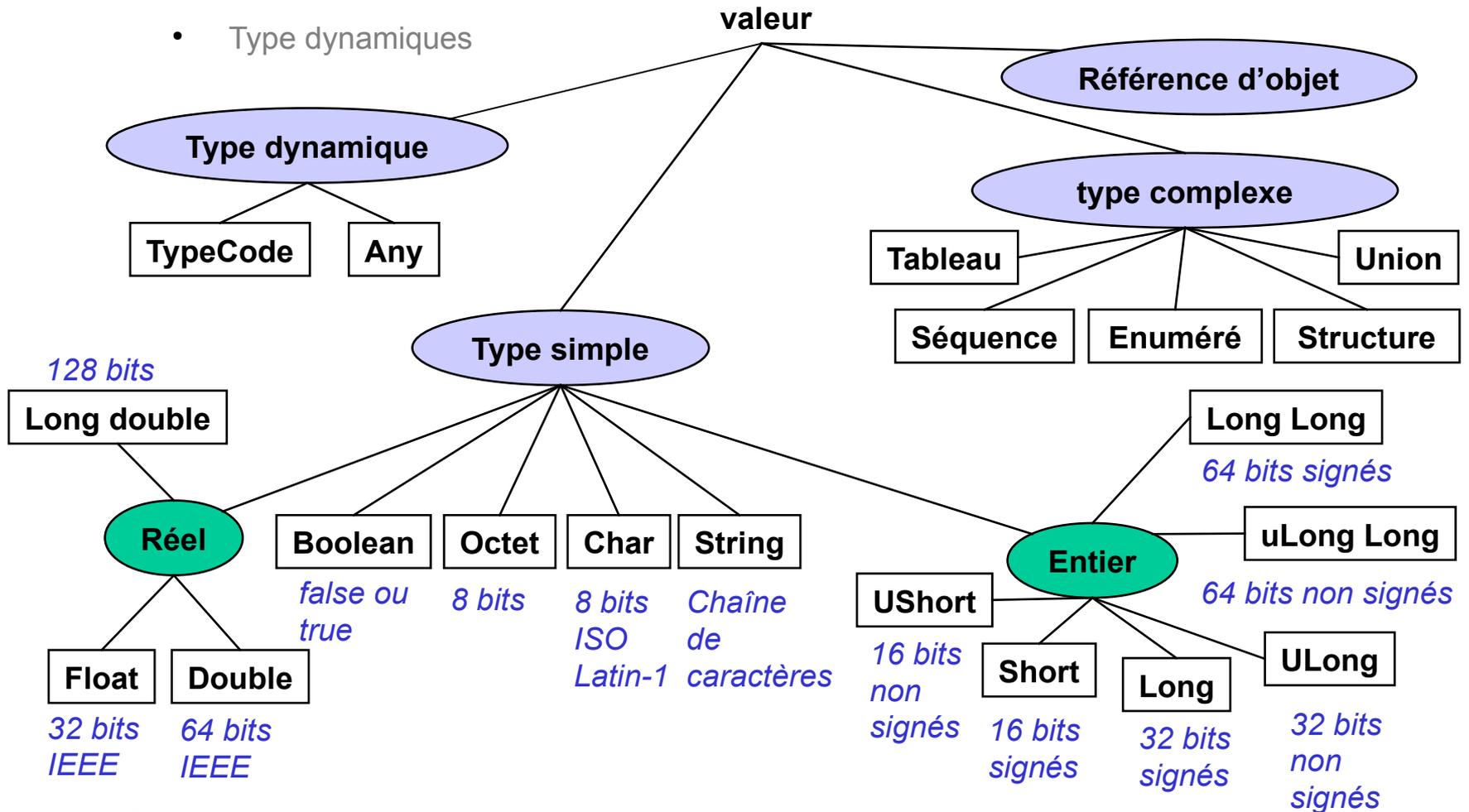
```
const short TAILLE = 10;  
typedef float[20][TAILLE] Matrice;
```

- Sequence
 - Vecteur de taille quelconque
 - non borné
 - borné

```
typedef sequence<string> StrSeq // séquence de chaîne  
typedef sequence<short> entiers; // séquence non bornée  
typedef sequence<long, 100> Nombres; // au plus 100 nombres  
typedef sequence<Nombres> ListDeNombres; // séquence de séquence
```

Types IDL

- Type dynamiques



Types dynamiques

- TypeCode
 - Structure décrivant un type de données (nom + champs)
- **any**
 - Contient
 - le type de la donnée (CORBA TypeCode)
 - la valeur de la donnée
 - Très utile pour les opérations polymorphes
 - Permet un nombre variable de paramètres
- dynany
 - Type *dynamique* pour manipuler localement un any avec itérateurs

```
interface ValueStore
{
    void put(in string name, in any value);
    any get(in string name);
};
```

Types valuetype

- Types qui sont des objets :
 - CORBA::Object – objet accessible à distance
 - champs privés, attachés à leur serveur (passés par référence)
 - IDL struct – données sérialisables
 - Passés par valeur, données uniquement (pas de méthodes)
 - Valuetype – objet **local sérialisable**
 - Passés par valeur
 - Méthodes avec invocation locale uniquement
 - Pas de garantie que tous les noeuds utilisent la même implémentation des méthodes...

```
valuetype Date {  
    short    year;  
    short    month;  
    short    date;  
    void     next_day();  
    void     previous_day();  
};
```

Types C++ *_var – Gestion de la mémoire

- Type **T_var**
 - Généré à partir de la description IDL
 - Gère l'allocation et la désallocation (~ ramasse miette)
 - Compteur de références automatique
 - Y compris les références de l'ORB vers les objets
- Type **T_ptr**
 - Type pointeur vers une donnée de type T
 - Généré à partir de la description IDL
 - Allocation/désallocation manuelles par le programmeur

```
IDL: interface A { ...};
```

```
C++: typedef ... A_var;  
      typedef ....A_ptr;
```

Interface de l'ORB

- L'ORB vu comme un objet CORBA
 - Uniquement local !

```
#pragma prefix "omg.org"

module CORBA {
  ...
  interface ORB {
    ...
    string object_to_string ( in Object obj );
    Object string_to_object ( in string str );

    typedef string ObjectId;
    Object resolve_initial_references ( in ObjectId id)
        raises (InvalidName);

    void run();
    boolean work_pending();
    void perform_work();
  };
};
```

Interface dynamique

- Utiliser CORBA sans stub/squelette
 - Sérialiser/désérialiser des requêtes **explicitement**
- Utile pour :
 - Langages avec **typage dynamique**, typage faible
 - ex.: python, perl, Tcl, LISP, Visual Basic, etc.
 - Debug
 - Prototyper rapidement un client « script » pour tests unitaires
 - Interfaces IDL évolutives
 - S'adapter à la volée au type
 - Passerelles, proxy, firewall
 - Contrôler manuellement l'envoi des requêtes
 - Asynchronisme, envoi groupé, etc.

DII - Interface client dynamique

- DII – Dynamic Invocation Interface
 - Usage assez courant
- Manipulation **explicite** de `CORBA::Request`
 - Création : `CORBA::Object::_create_request()`
 - Empaquetage explicite des arguments dans des `Any`
 - Contiennent des `NamedValue` – paire nom, valeur
 - Invocation : `CORBA::Request::invoke()`
 - Récupération explicite du résultat :
`CORBA::Request::get_response()`
- Code équivalent au *stub* généré par le compilateur IDL

DII - Exemple

- IDL de l'exemple : `short anOpn(in string a);`
- Exemple DII détaillé

```
CORBA::NVList_var args;  
orb->create_list(1, args);  
*(args->add(CORBA::ARG_IN)->value()) <<= (const char*) "Hello World!";  
  
CORBA::NamedValue_var result;  
orb->create_named_value(result);  
result->value()->replace(CORBA::_tc_short, 0);  
  
CORBA::Request_var req = obj->_create_request(CORBA::Context::_nil(),  
                                              "anOpn", args, result, 0);
```

- Même exemple, syntaxe idiomatique :

```
CORBA::Request_var req = obj->_request("anOpn");  
req->add_in_arg() <<= (const char*) "Hello World!";  
req->set_return_type(CORBA::_tc_short);
```

DSI - Interface serveur dynamique

- DSI – Dynamic Skeleton Interface
 - Usage plus anecdotique que DII
- Servant générique déclaré au POA : `PortableServer::DynamicImplementation`
 - Hériter de la classe générique
 - Surcharger la fonction `invoke(CORBA::ServerRequest)` qui reçoit tous les appels
 - Dépaqueter le contenu de `ServerRequest`, traiter la requête, empaqueter le résultat, de manière symétrique à la gestion des DII
 - Servant enregistré dans le POA comme n'importe quel servant
 - Le plus souvent via un `ServantManager` pour gérer plusieurs objets
- Code équivalent au *squelette* généré par le compilateur IDL

DSI - Exemple

```
void MyDynImpl::invoke(CORBA::ServerRequest_ptr request)
{
    try {
        if( strcmp(request->operation(), "echoString") )
            throw CORBA::BAD_OPERATION(0, CORBA::COMPLETED_NO);
        CORBA::NVList_ptr args;
        orb->create_list(0, args);
        CORBA::Any a;
        a.replace(CORBA::_tc_string, 0);
        args->add_value("", a, CORBA::ARG_IN);
        request->arguments(args);
        const char* mesg;
        *(args->item(0)->value()) >= mesg;
        CORBA::Any* result = new CORBA::Any();
        *result <= CORBA::Any::from_string(mesg, 0);
        request->set_result(*result);
    }
    catch(CORBA::SystemException& ex){
        CORBA::Any a;
        a <= ex;
        request->set_exception(a);
    }
    catch(...){
        cout << "echo_dsiimpl: MyDynImpl::invoke - caught an unknown exception."
        << endl;
        CORBA::Any a;
        a <= CORBA::UNKNOWN(0, CORBA::COMPLETED_NO);
        request->set_exception(a);
    }
}
```

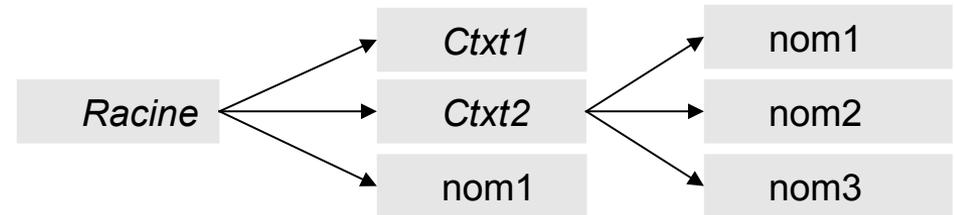
Asynchronisme (ou presque)

- Abus de langage – en fait, requête **non-bloquante**
- Construction du langage IDL : **oneway**
 - Invocation non-bloquante, pas de valeur de retour, pas d'exception
 - **Aucune garantie** de fiabilité !
- Utilisation de DII
- Construction selon un pattern *callback*
 - Le servant rend la main immédiatement, crée un thread pour le travail, appelle une fonction en retour sur le client pour donner le résultat
- AMI – Asynchronous Message Interface
 - Un pattern *callback* généré automatiquement par le compilateur IDL
 - Partie *optionnelle* de la norme CORBA, implémentation peu répandue

Asynchronisme, pour de vrai

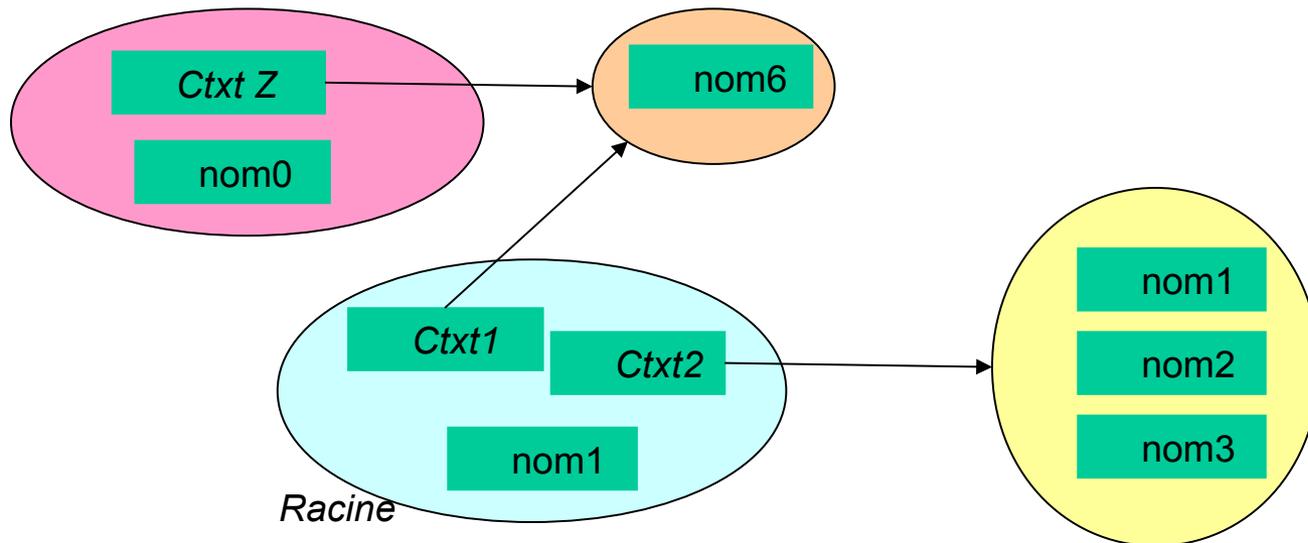
- **Vrai asynchronisme** : la requête n'est pas un point de synchronisation (**barrière**) entre client et serveur
 - Couplage faible
 - Messages asynchrones, boîte aux lettres
- Avec une **tierce-partie** persistante
 - Pattern *Mediator*
 - Les messages transitent par la tierce-partie
 - Le récepteur lit explicitement sa boîte aux lettres (modèle *pull*) ou reçoit une notification quand il est connecté (modèle *push*)
- Avec des événements asynchrones
 - Pattern *Publish/subscribe*
 - Implémenté dans les services CosEvent et CosNotification

NameService avancé



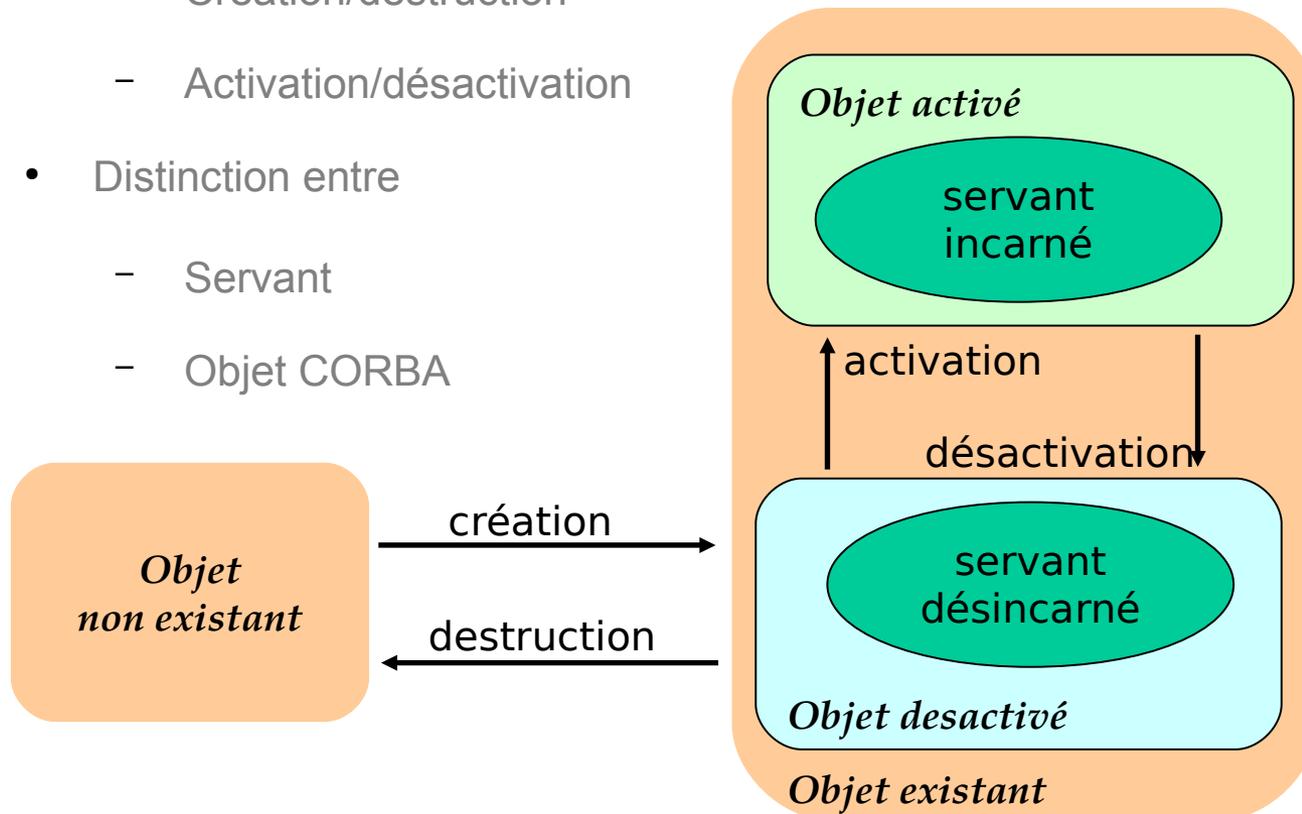
- Fédération de name service
 - Un NamingContext est un objet CORBA comme un autre
 - Les références peuvent être externes

Autre Racine



Cycle de vie d'un objet CORBA

- Notions orthogonales
 - Création/destruction
 - Activation/désactivation
- Distinction entre
 - Servant
 - Objet CORBA

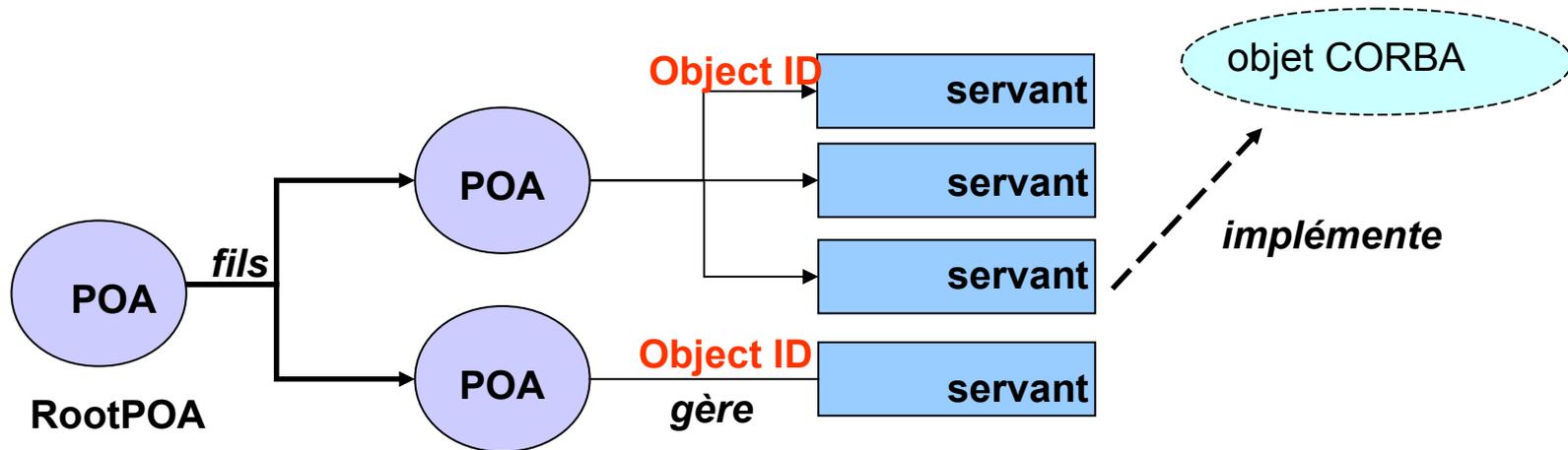


POA – Portable Object Adapter

- Intermédiaire entre l'ORB et l'implémentation d'un objet CORBA (le servant)
- Fournit des services pour
 - La création d'objet CORBA
 - La création de référence
 - L'aiguillage des requêtes aux servants appropriés
 - L'activation/déactivation de servants
 - Décrire des propriétés non fonctionnelles des servants
 - Politiques de : multithreading, persistance, ...

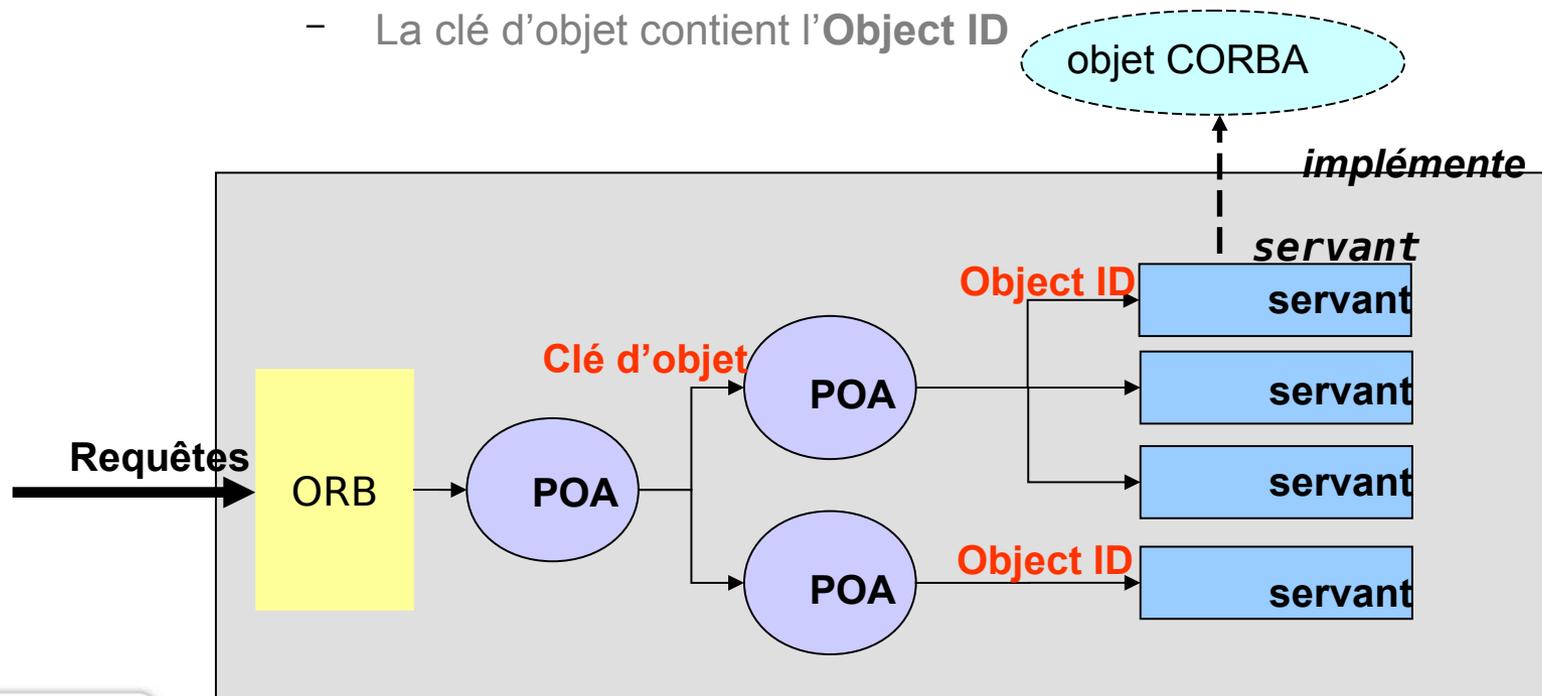
Hierarchie de POA

- Les POA sont organisés en arbre
 - La racine est le RootPOA
 - Chaque POA peut avoir des politiques différentes
- Un POA peut gérer plusieurs objets CORBA
- Un servant est identifié via son ObjectID dans son POA



Cheminement dans le POA

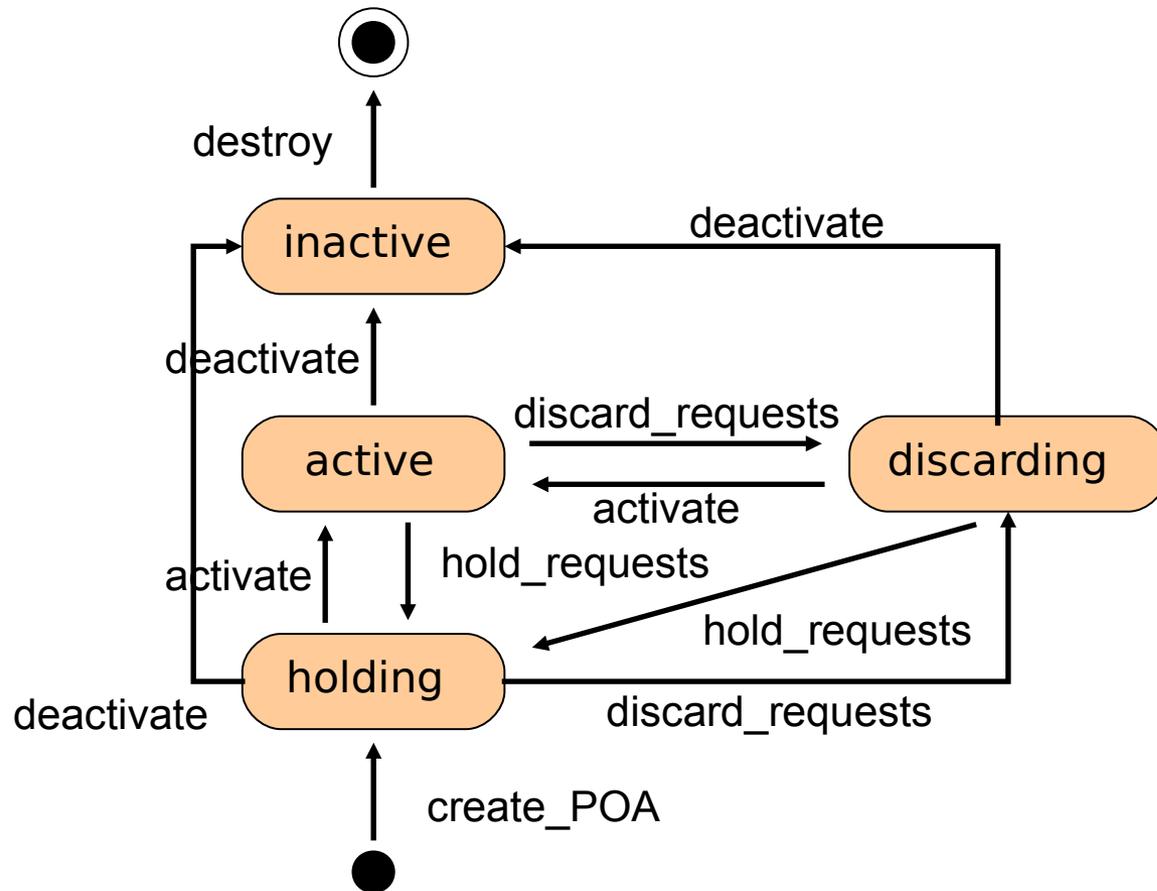
- Référence d'objet: type opaque
 - Doit permettre à l'ORB de trouver le POA associé via le champ "clé d'objet" (*object key*)
 - La clé d'objet contient l'**Object ID**



Gestion des POA

- Les POA sont gérés par des POAManager
 - Tout POA à son POAManager
 - Un POAManager peut avoir plusieurs POA
 - Permet d'avoir des opérations groupées atomiques
- Les gestionnaires de POA contrôlent l'état des POA
 - **Holding** : stocke les requêtes (en nombre limité puis discard)
 - **Active** : traite les requêtes (par ex. délivre aux servants)
 - **Discard** : détruit les requêtes (retourne TRANSIENT)
 - **Inactive** : détruit les requêtes (non-spécifié)
-

Diagramme d'état du POAManager



IDL du POAManager

```
#pragma prefix "omg.org"
module PortableServer
{
  local interface POAManager
  {
    exception AdapterInactive{};
    enum State {HOLDING, ACTIVE, DISCARDING, INACTIVE};

    void activate() raises(AdapterInactive);
    void hold_requests( in boolean wait_for_completion)
      raises(AdapterInactive);
    void discard_requests( in boolean wait_for_completion)
      raises(AdapterInactive);
    void deactivate( in boolean etherealize_objects, in boolean wait_for_completion)
      raises(AdapterInactive);
    State get_state();
  };
  local interface POA
  {
    readonly attribute POAManager the_POAManager;
  };
};
```

Références et activation

- Opérations distinctes :
 - Création de référence

```
local interface POA {  
    // reference creation operations  
    Object create_reference ( in CORBA::RepositoryId intf) raises (WrongPolicy);  
    Object create_reference_with_id ( in ObjectId oid, in CORBA::RepositoryId intf );  
};
```

- Activation (incarnation) d'un objet

```
local interface POA {  
    ObjectId activate_object( in Servant p_servant)  
        raises (ServantAlreadyActive, WrongPolicy);  
    void activate_object_with_id( in ObjectId id, in Servant p_servant)  
        raises (ServantAlreadyActive, ObjectAlreadyActive, WrongPolicy);  
    void deactivate_object( in ObjectId oid)  
        raises (ObjectNotActive, WrongPolicy);  
};
```

- Activation **implicite** par défaut (dans le RootPOA)

Fonctionnement du POA

- Idée intuitive : une table ObjectID -> servant

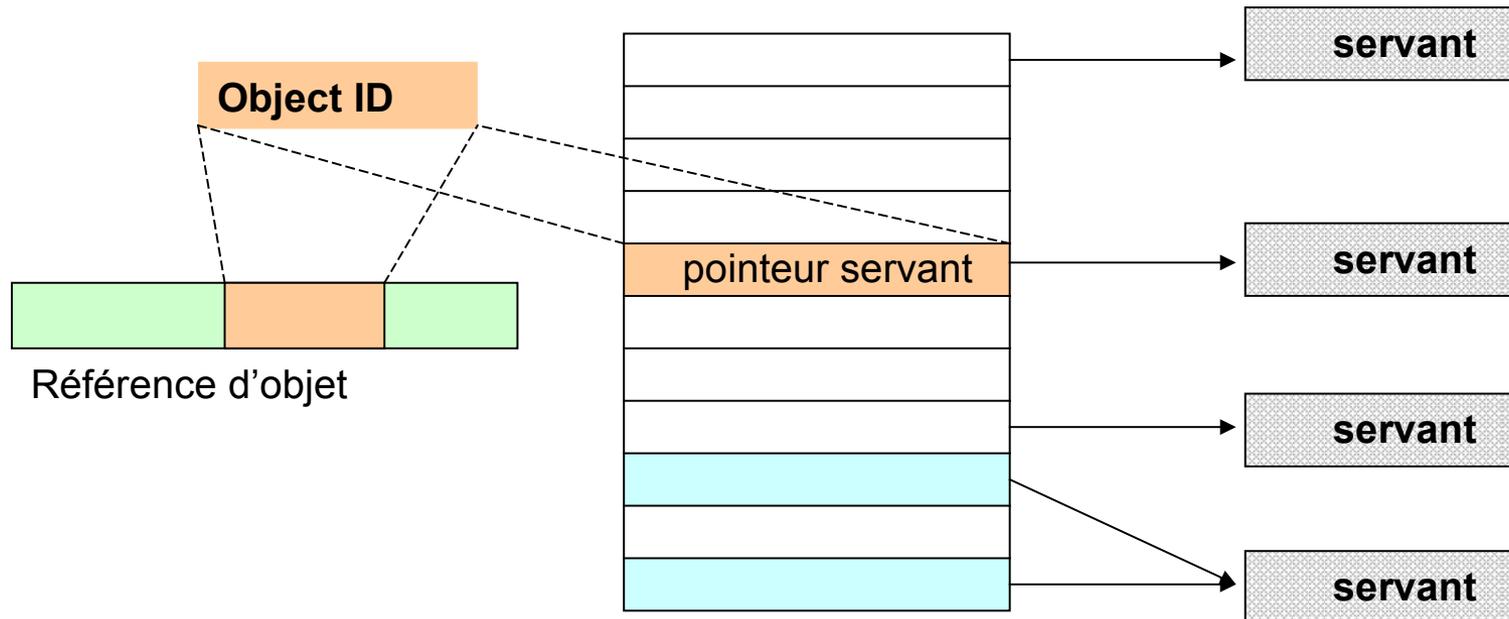
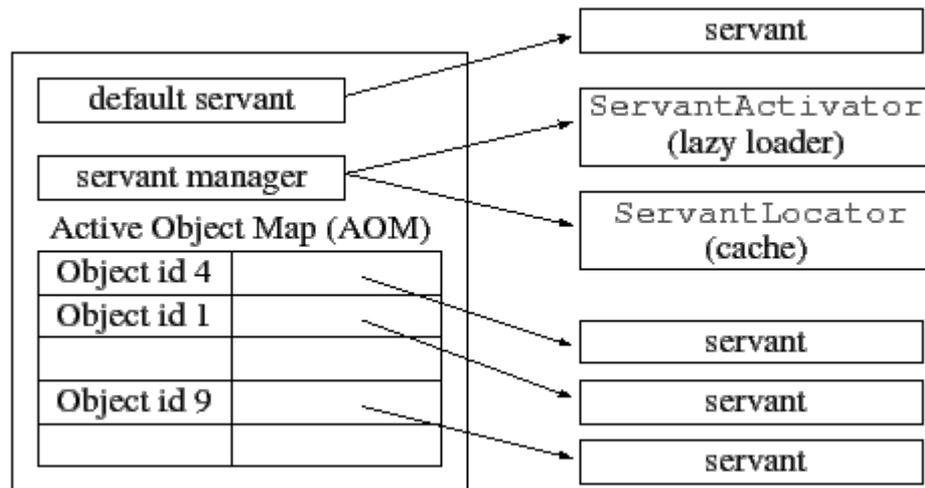


Tableau des objets actifs
POA Active Object Map

Fonctionnement du POA

- Un peu plus qu'une simple table...
 - Servant par défaut – incarne tous les objets du POA
 - Servant activator – instantiation à la demande
 - Servant locator – démultiplexage à la charge du programmeur
 - Cache, équilibrage de charge, forward, etc.



Politiques du POA

- Thread - ThreadPolicy
 - Séquentiel, main thread, multithread
- Durée de vie – LifeSpanPolicy
 - Ephémère, persistant
- Unicité des ObjectID – IdUniquenessPolicy
 - Un servant par objet, un servant pour plusieurs objets (sans état)
- Identification des objets – IdAssignmentPolicy
 - Par l'utilisateur, par le système
- Rétention des associations – ServantRetentionPolicy
 - Retenir l'Id d'un servant dans l'active object map
- Association des requêtes aux servants – RequestProcessingPolicy
 - Active Object Map, default servant, servant manager
- Activation implicite – ImplicitActivationPolicy
 - Activation automatique à la création de la référence

Politiques du POA - Référence

```
// IDL
#pragma prefix "omg.org"
module PortableServer {

    enum ThreadPolicyValue { ORB_CTRL_MODEL, SINGLE_THREAD_MODEL, MAIN_THREAD_MODEL };
    local interface ThreadPolicy : CORBA::Policy { readonly attribute ThreadPolicyValue value; };
    enum LifespanPolicyValue { TRANSIENT, PERSISTENT };
    local interface LifespanPolicy : CORBA::Policy { readonly attribute LifespanPolicyValue value; };
    enum IdUniquenessPolicyValue { UNIQUE_ID, MULTIPLE_ID };
    local interface IdUniquenessPolicy : CORBA::Policy { readonly attribute IdUniquenessPolicyValue value; };
    enum IdAssignmentPolicyValue { USER_ID, SYSTEM_ID };
    local interface IdAssignmentPolicy : CORBA::Policy { readonly attribute IdAssignmentPolicyValue value; };
    enum ImplicitActivationPolicyValue { IMPLICIT_ACTIVATION, NO_IMPLICIT_ACTIVATION };
    local interface ImplicitActivationPolicy : CORBA::Policy { readonly attribute ImplicitActivationPolicyValue v; };
    enum ServantRetentionPolicyValue { RETAIN, NON_RETAIN };
    local interface ServantRetentionPolicy : CORBA::Policy { readonly attribute ServantRetentionPolicyValue val; };
    enum RequestProcessingPolicyValue { USE_ACTIVE_OBJECT_MAP_ONLY, USE_DEFAULT_SERVANT, USE_SERVANT_MANAGER };
    local interface RequestProcessingPolicy : CORBA::Policy { readonly attribute RequestProcessingPolicyValue v; };

    local interface POA {
        // Factories for Policy objects
        ThreadPolicy          create_thread_policy          ( in ThreadPolicyValue value);
        LifespanPolicy        create_lifespan_policy        ( in LifespanPolicyValue value);
        IdUniquenessPolicy    create_id_uniqueness_policy   ( in IdUniquenessPolicyValue value);
        IdAssignmentPolicy    create_id_assignment_policy   ( in IdAssignmentPolicyValue value);
        ImplicitActivationPolicy create_implicit_activation_policy ( in ImplicitActivationPolicyValue value);
        ServantRetentionPolicy create_servant_retention_policy ( in ServantRetentionPolicyValue value);
        RequestProcessingPolicy create_request_processing_policy ( in RequestProcessingPolicyValue value);
    };
};
```

Contraintes sur les politiques du POA

- Toutes les combinaisons ne sont pas possibles
 - NON_RETAIN incompatible avec l'utilisation du tableau des objets actifs
 - UNIQUE_ID pas de sens avec NON_RETAIN
 - IMPLICIT_ACTIVATION réclame l'utilisation de SYSTEM_ID et RETAIN
-

Combinaison des politiques du POA

- Création automatique de serviant
 - RETAIN et USE_ACTIVE_OBJECT_MAP_ONLY
- Identique + contrôle de l'activation de serviant
 - RETAIN et USE_SERVANT_MANAGER
- Un serviant gère tous les objets inconnus
 - RETAIN et USE_DEFAULT_SERVANT
- Un serviant par requête
 - NON-RETAIN et USE_SERVANT_MANAGER
- Un serviant pour tous les objets
 - NON-RETAIN et USE_DEFAULT_SERVANT
-

Servant Manager

- Interface **ServantActivator**: politique RETAIN
 - Opération appelée par le POA lorsqu'il a besoin d'incarner/désincarner un objet

```
Servant incarnate ( in ObjectId oid, in POA adapter)  
    raises (ForwardRequest);  
void etherealize ( in ObjectId oid, in POA adapter, in Servant serv,  
    in boolean cleanup_in_progress,  
    in boolean remaining_activations);
```

- Interface **ServantLocator** : politique NON_RETAIN
 - Opération appelée pour chaque requête

```
Servant preinvoke( in ObjectId oid, in POA adapter,  
    in CORBA::Identifier operation, out Cookie the_cookie)  
    raises (ForwardRequest );  
void postinvoke( in ObjectId oid, in POA adapter,  
    in CORBA::Identifier operation, in Cookie the_cookie,  
    in Servant the_servant );
```

Durée de vie, persistance

- Durée de vie de l'objet CORBA par rapport au servant
 - Éphémère (TRANSIENT) : l'objet est détruit à la destruction du POA
 - Persistant (PERSISTENT) : l'objet survit au POA
 - Les références restent valides au redémarrage du serveur
- CORBA ne s'occupe pas de sauvegarder l'état interne de l'objet
 - Seule la **référence** est persistante
- Combinable avec les autres politiques
 - Par exemple ServantManager pour réinstancier automatiquement un servant au démarrage
 - Utilisation habituelle avec un *daemon* tel que orbd

Création d'un POA avec persistance

```
// résolution du RootPOA
POA rootPOA = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));

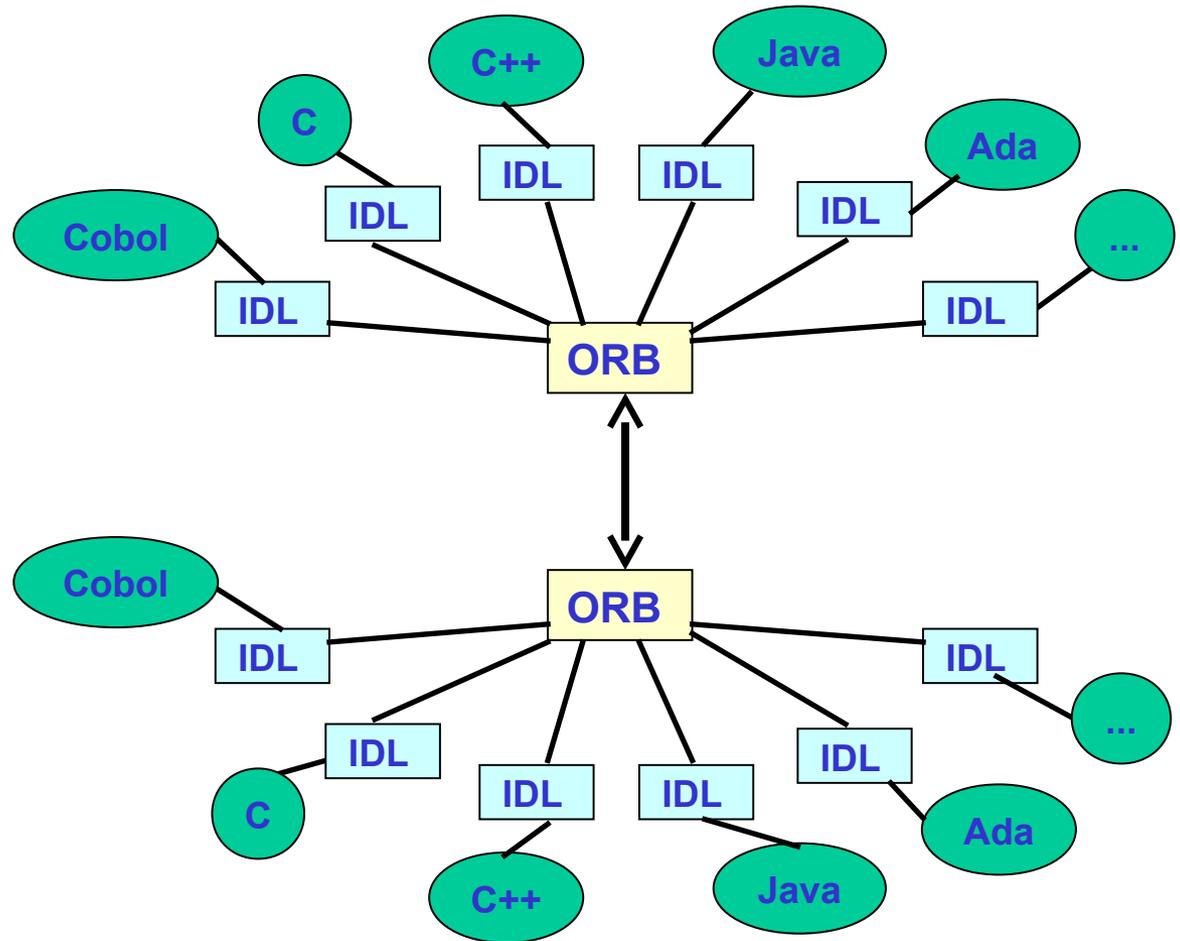
// création de la politique PERSISTENT
Policy[] persistentPolicy = new Policy[1];
PersistentPolicy[0] =
    rootPOA.create_lifespan_policy(LifespanPolicyValue.PERSISTENT);

// création du nouveau POA
POA persistentPOA = rootPOA.create_POA("childPOA", null,
    persistentPolicy);

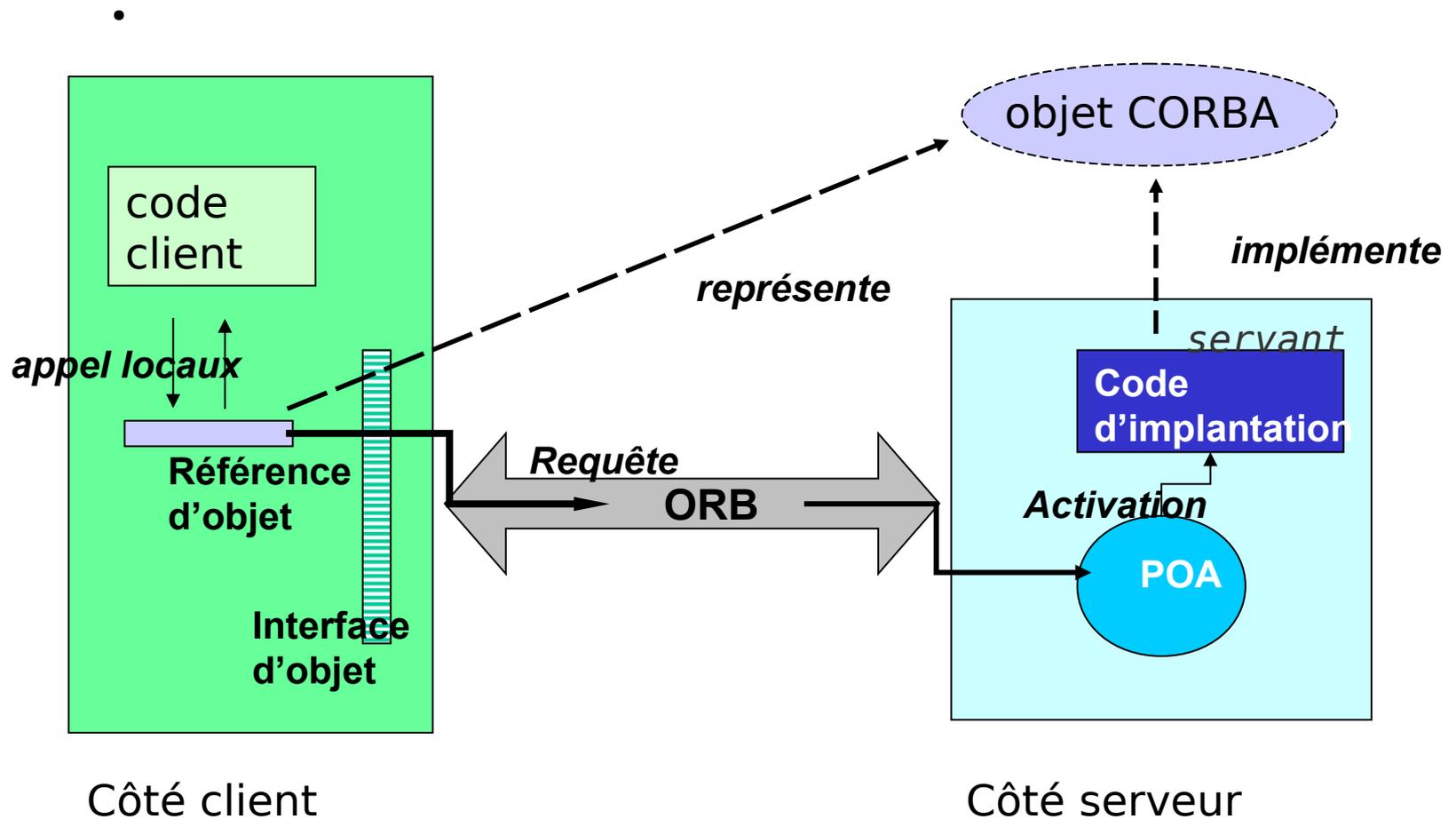
// Activation du nouveau POA, i.e. passage de l'état HOLD à ACTIVE
persistentPOA.the_POAManager().activate( );

// enregistrement d'un servent dans le POA persistent
persistentPOA.activate_object(servant);
```

Résumé CORBA



Résumé CORBA



Travail à faire

Echauffement

- Debuguer les exemples Hello World
 - Il manque quelques import, la gestion des exceptions
- Compiler et faire tourner les exemples Hello World
 - En local
 - A distance
 - Avec le serveur d'un voisin
- Modifier la méthode pour manipuler différents types : string, long, structures
 - Observez les changements dans EchoOperations.java

Application bancaire

- Nous allons simuler le fonctionnement du système bancaire
 - Pas complètement réaliste...
- Différentes entités :
 - Banques : **Bank**
 - Entité interbancaire (chambre de compensation) : **InterBank**
 - Assure la sincérité des échanges
 - Compte bancaire : **Account**
 - Clients, titulaire de comptes bancaires dans une banque
- **Attention : tout n'est pas un objet CORBA**
- Opérations : ouverture de compte, dépôt, retrait, obtention du solde, virement intra-banque, virement interbancaire

Application bancaire, v0

- Nous allons faire une première version :
 - Synchronone
 - Sans intermédiaire
 - Sans persistance
- Mettre en place l'interface exposée par la banque aux autre banques pour les virements interbancaires
 - Écrire le code local de gestion des comptes bancaires
 - pas de CORBA ici !
 - Écrire l'IDL, le servant, le serveur pour une banque
 - Écrire une classe de test qui effectue quelques opérations
 - Faire des tests avec plusieurs banques

Avec intermédiaire bancaire

- Architecture générale
 - Entité interbancaire : Interbank
 - Route les transactions
 - Maintient un historique des transactions
 - Les banques
 - Se connectent à Interbank
 - Enregistrent un callback pour recevoir des transactions
 - Les clients
 - Se connectent à leur banque par une interface web, en REST
 - À voir dans 15 jours

Intermédiaire, asynchronisme

- Mettre en place l'architecture générale
 - Types de données échangées (transaction)
 - Interfaces IDL pour :
 - Interbank – le service interbancaire
 - BankTransaction – l'interface de callback pour recevoir des transactions depuis l'Interbank
 - Désignation : numéro de banque, numéro de compte

Virement interbancaire

- Virement asynchrone
 - Pas perdu, même si le serveur destinataire ne tourne pas
- Déroulement de la transaction
 - Banque A -> B, service interbancaire I
 - message : A->I ; puis I -> B
 - une fois exécuté : confirmation B ->I (inscription dans l'historique) ; confirmation I->A (débit du compte)

Observateur

- Écrire un mini-client CORBA pour :
 - Afficher les banques connectées à l'Interbank
 - Afficher l'historique des transactions mémorisées dans l'Interbank

Application bancaire - objectifs

- Propriétés souhaitables
 - Encapsulation – les objets représentant les comptes bancaires ne sont pas accessibles de l'extérieur
 - De l'extérieur : numéro de compte
 - **Asynchronisme** des requêtes
 - On peut faire un virement vers une banque même si son serveur n'est pas connecté
 - Persistance
 - Une redémarrage du serveur ne réinitialise pas les comptes à zéro !
 - Transactions
 - Un virement réussit (crédit destinataire & débit émetteur) ou échoue

Persistence

- Le service interbancaire doit être persistant
 - Référence constante
 - Instanciation automatique au démarrage
 - Restauration de l'état au démarrage
- Deux options
 - Policy PERSISTENT du POA
 - « Vraie » persistence
 - Default servant
 - Équivalent si l'objet est un singleton

Persistence par Default servant

- Alternative à la persistance de la référence de l'objet
 - Référence non-persistante pour l'InterBank mais un **default servant**
 - On gère alors l'objet comme un **singleton**
 - L'objet est seul dans son POA
 - C'est à vous de restaurer manuellement l'état interne de l'objet
 - Pensez à utiliser la *sérialisation* automatique Java

Persistence avec PERSISTENT policy (optionnel)

- « Vraie » persistance de référence
- Implémentez la persistance à l'aide d'un POA doté de la politique PERSISTENT et du serveur d'application orbd et de l'outil `servertool` de Java IDL
- Suivez le tutoriel Oracle :
<http://docs.oracle.com/javase/7/docs/technotes/guides/idl/jidlExample2.html>

Travail attendu

- Code :
 - Interfaces IDL, implémentations d'objets, code serveur, scripts nécessaires pour le déploiement et la persistance
 - Code observateur en CORBA
 - Code client permettant de créer des comptes, lancer des transactions, obtenir le solde en REST (voir TP 4)
- Rapport :
 - Explication de la démarche, de l'articulation des différents objets, du fonctionnement global
 - Notice de déploiement (comment compiler/lancer tous les processus)
 - Difficultés rencontrées, problèmes non-résolus
- Aspects évalués :
 - Asynchronisme, transactions, encapsulation, persistance, tests, robustesse, respect du schéma demandé
 - Clarté, explications globale, concision, **orthographe**

Formalisme

- Rendre l'ensemble de l'application bancaire
 - par e-mail à `Alexandre.Denis@inria.fr`
 - Au plus tard le **23 novembre 2018**
- Archive `.tar.gz` contenant :
 - Rapport en pdf
 - Code (compilable !)
 - Makefile (!)
- Nom de fichier contenant votre nom
- « PG306 » dans le sujet du mail



À vous de jouer !

inria
informatics mathematics

<http://dept-info.labri.fr/~denis/>