Construction d'Applications Réparties avec CORBA

Serge Midonnet
(*serge.midonnet@univ-paris-est.fr*)

Institut Gaspard-Monge, Université Paris Est Marne-La-Vallée

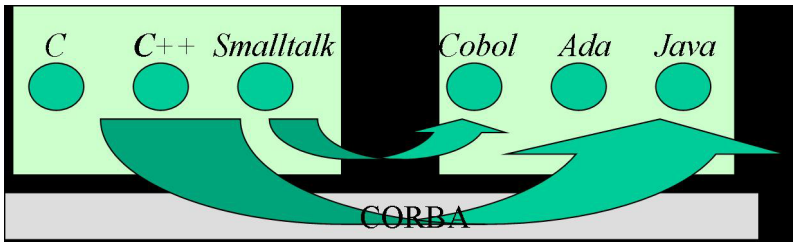
Février 2013

Outline

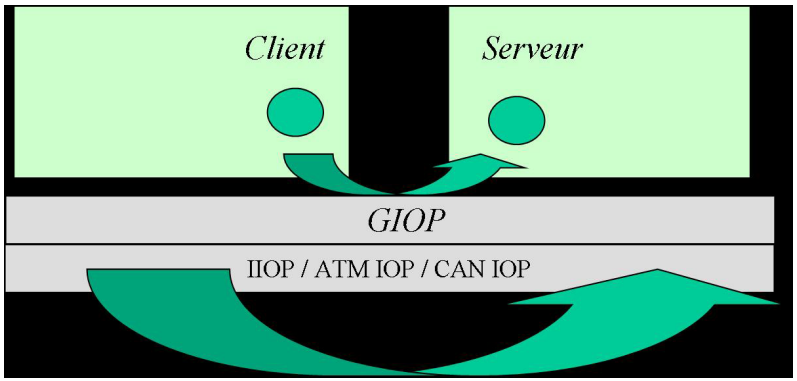
1. Introduction au modèle client serveur CORBA.
2. Description des composants de l'application: Client, Servant, Serveur
3. Langage IDL en détail
4. Programmation d'une application
5. Localisation des objets (Service de nommage).

Common Object Request Brocker Architecture (CORBA) spécification de l'OMG (Object Management Group).
Fournir un environnement de développement et d'exécution d'applications réparties portable:

- Indépendance vis à vis des langages de programmation (IDL).



- Indépendance vis à vis des protocoles de communication (GIOP).



Les domaines d'applications de CORBA:

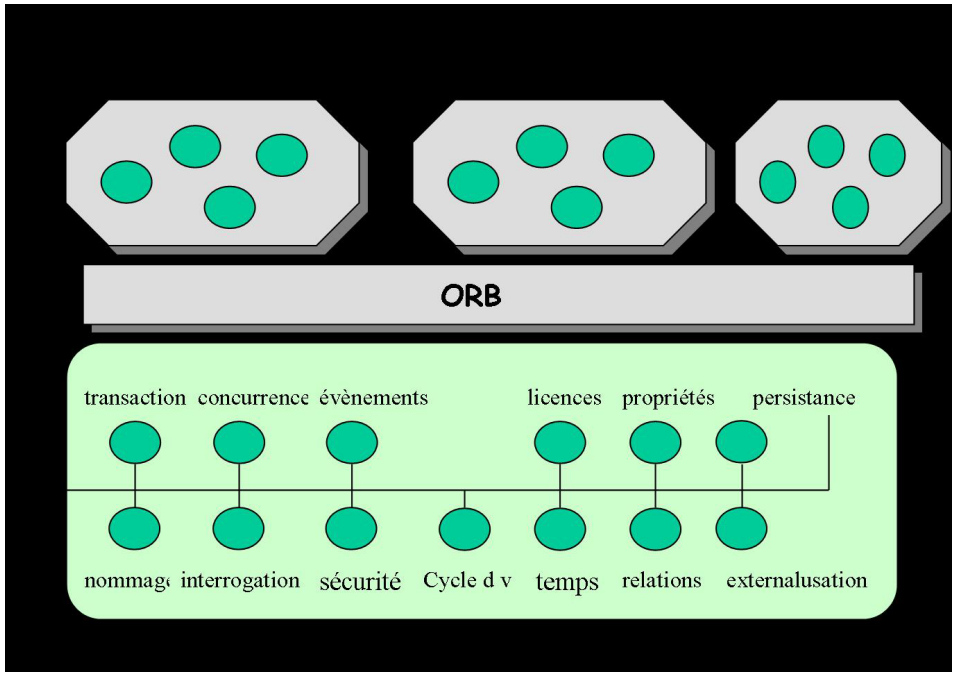
- Applications Militaires (Awacs) & Aéronautique (Eurocontrol) 1
- Télécommunications et transmissions de données
- Equipements Mobiles [Prismtech::eORB]
- Transports
- Controle de Processus industriel
- Robotique

1: http://www.ois.com/markets/vert-7-5.asp Les implantations principales:

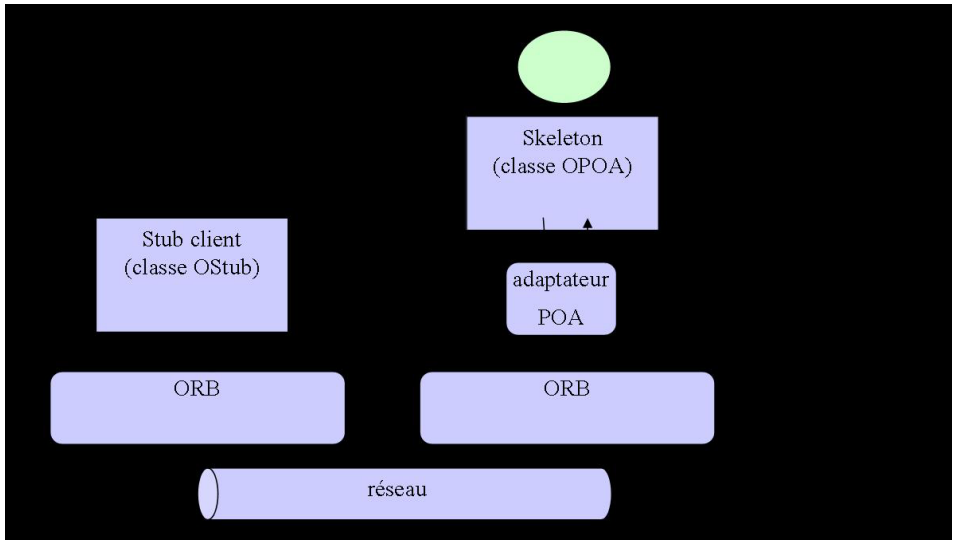
- Libres: Jacorb; OpenORB; Orbacus, TAO, MICO.
- OpenFusion [Prismtech], eORB [Prismtech], Orbix[Iona], Visibroker[Borland], ORBexpress [OIS],

Des interfaces spécifiées

- interfaces d'objets distants: services communs; utilitaires communs; interfaces métiers; applications
- interfaces de pseudo objets (internes): [POA] et extensions [RT-CORBA, MinimumCORBA]



- composants corba du serveur [orb][poa][skeleton][servant]
- composants corba du client [orb][stub]



Certains composants sont développés (Servant), certain sont générés par le compilateur IDL (Skeleton, Stub) d'autres proviennent de l'api corba (Orb, Poa ...)

Le développement d'une application Corba nécessite d'effectuer les étapes suivantes:

- Etape 1: Spécifier l'interface IDL du servant
- Compiler le fichier d'interface `X.idl` pour la génération des classes utils au développement de l'application
- Etape 2: Développer le Servant `XImpl.java` en s'aidant de le traduction IDLversJava `XOperations.java`
- Etape 3: Développer le Serveur `Xserver.java`
- Etape 4: Développer le Client `Xclient.java`

jdk:
```
idlj -fall X.idl
```

jacorb:
```
java -classpath /usr/local/apps/Jacorb/lib/idl.jar:
/usr/local/apps/Jacorb/lib/logkit-1.2.jar
org.jacorb.idl.parser X.idl
```

generated:

`XOperations.java, XPOA.java, XStub.java, XHelper.java, XHolder.java.`

- Exécution du Client et du Serveur

```
java -Djava.endorsed.dirs=/usr/local/apps/Jacorb/lib
-Djacorb.home=/usr/local/apps/Jacorb
-Dorg.omg.CORBA.ORBClass=org.jacorb.orb.ORB
-Dorg.omg.CORBA.ORBSingletonClass=org.jacorb.orb.ORBSingleton
-classpath . %1
```

Etape 1: Spécification de l'interface du servant. Le langage IDL est un langage neutre. Corba propose un mapping vers Ada, Java, C++, Smalltalk, C, Python. Premier pas avec IDL: module (->package), struct (classe), interface (->classe), attribut (->méthode), opération (->méthode).

```
module td2_basic {
  struct Message {
    string title;
    string author;
    string date;
    string body;
  };

  interface Forum {
    readonly attribute string theme;
    readonly attribute string moderator;
    boolean postMessage(in Message m);
    Message getMessage(in string title);
    boolean removeMessage(in string title);
  };
};
```

Etape 2: La classe servant `XImpl` va hériter de la classe `XPOA` (skeleton de X) générée par le compilateur.

```
package td2_basic;
public class ForumImpl extends ForumPOA {
  ForumImpl(String theme, String moderator) {
  code du constructeur
  }
  public String theme() {
  code de l'accesseur theme
  }
  public String moderator() {
  code de l'accesseur moderateur
  }
  public boolean postMessage(Message m) {
code de l'opération postMessage
  }
  public Message getMessage(String title) {
code de l'opération getMessage
  }
  public boolean removeMessage(String title) {
code de l'opération removeMessage
  }
```

Etape 3: Développement du serveur: (1) intilalisation de l'orb, (2)Obtenir référence du RootPOA, (3) instancier le servant, (4) activer le servant (5) obtenir la référence du servant (6) publier la référence du servant, (7) activer le POA, (8) attendre des requÍtes.

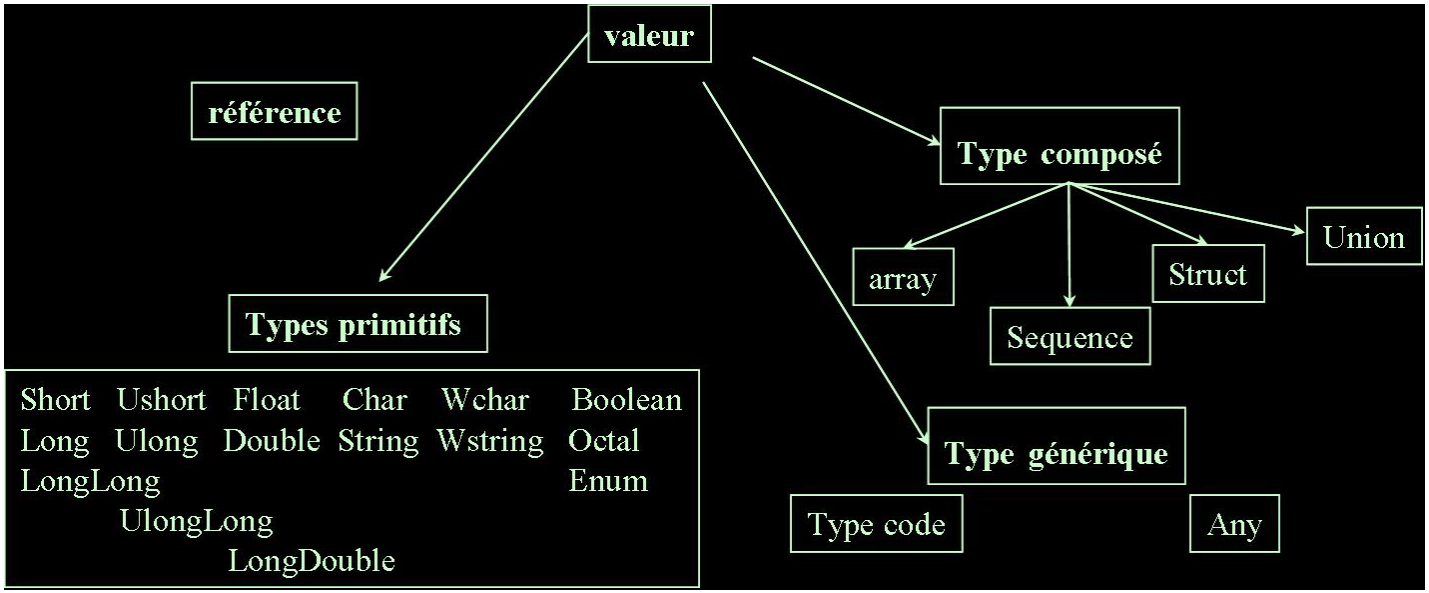
```
(1)    ORB orb = ORB.init(args, null);

(2)    POA rootPOA =
      POAHelper.narrow(orb.resolve_initial_references("RootPOA"));

(3)    ForumImpl forum = new ForumImpl("rtsj","damien");

(4)    byte[] servantId = rootPOA.activate_object(forum);

(5)   String reference =
      orb.object_to_string(rootPOA.id_to_reference(servantId));

(6)   PrintWriter file = new PrintWriter("ObjectRef");
     file.println(reference); file.close();

(7)   rootPOA.the_POAManager().activate();

(8)   orb.run();
```

Etape 4: Développement du client: (1) intilalisation de l'orb, (2) Obtenir la référence du servant corba distant (org.omg.Corba.Object) (3) Instancier un stub (proxy) de l'objet distant (XHelper.narrow), (4) utiliser le proxy pour effectuer des invocations distantes.

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

(2)   BufferedReader fileReader =
       new BufferedReader(new FileReader("ObjectRef"));
      String stringIOR = fileReader.readLine(); fileReader.close();

(3)   org.omg.Corba.Object reference = orb.string_to_object(stringIOR);

(4)   Forum forumProxy = ForumHelper.narrow(reference);

(5)   Message m = new Message("CORBA","moi","10/01/2008","Corba c est bien");

      forumProxy.postMessage(m);

      Message tmp = forumProxy.getMessage("CORBA");
      System.out.println("Titre : " + tmp.title);
      System.out.println("Auteur : " + tmp.author);
      System.out.println("Message : " + tmp.body);
  }
```

module td1

- interface INTERFACE NAME
  - définitions:
    - de types utilisateur (typedef, struct),
    - d'exceptions,
    - d'attributs.
  - opérations: oneway nom de méthode (attribut directionnel (in out inout) paramêtre) exceptionA, exceptionB



Dans l'IDL:

```
typedef sequence <Message> MessageSet;
MessageSet messageList();
```

Dans le servant:

```
private ConcurrentHashMap<String,Message> messages =
  new ConcurrentHashMap<String,Message>();
.....................................................
public Message[] messageList() {
return messages.values().toArray(new Message[messages.size()]);
}
.....................................................
```

Dans l'IDL:

```
exception Reject { string message; };
```

Dans le Servant:

```
public void postMessage(Message m) throws Reject {
  Message mInMap = messages.putIfAbsent(m.title,m);
  if (mInMap != null) {
    throw new Reject("PostMessage : message with title "+
     m.title + " already exists!");
  }}
```

Dans le Client:

```
    try {
    Message m = new Message("CORBA", "moi",
(new Date()).toString(),
"Corba c'est bien !");
      forumProxy.postMessage(m);
    } catch (Reject r) {
      System.err.println(r.message);
    }
```

Dans l'IDL:

```
void getInfo(out string theme, out string moderator, out long size);
```

Dans le servant:

```
public void getInfo(StringHolder theme, StringHolder moderator, IntHolder siz
  theme.value = this.theme;
  moderator.value = this.moderator;
  size.value = messages.size();
}
```

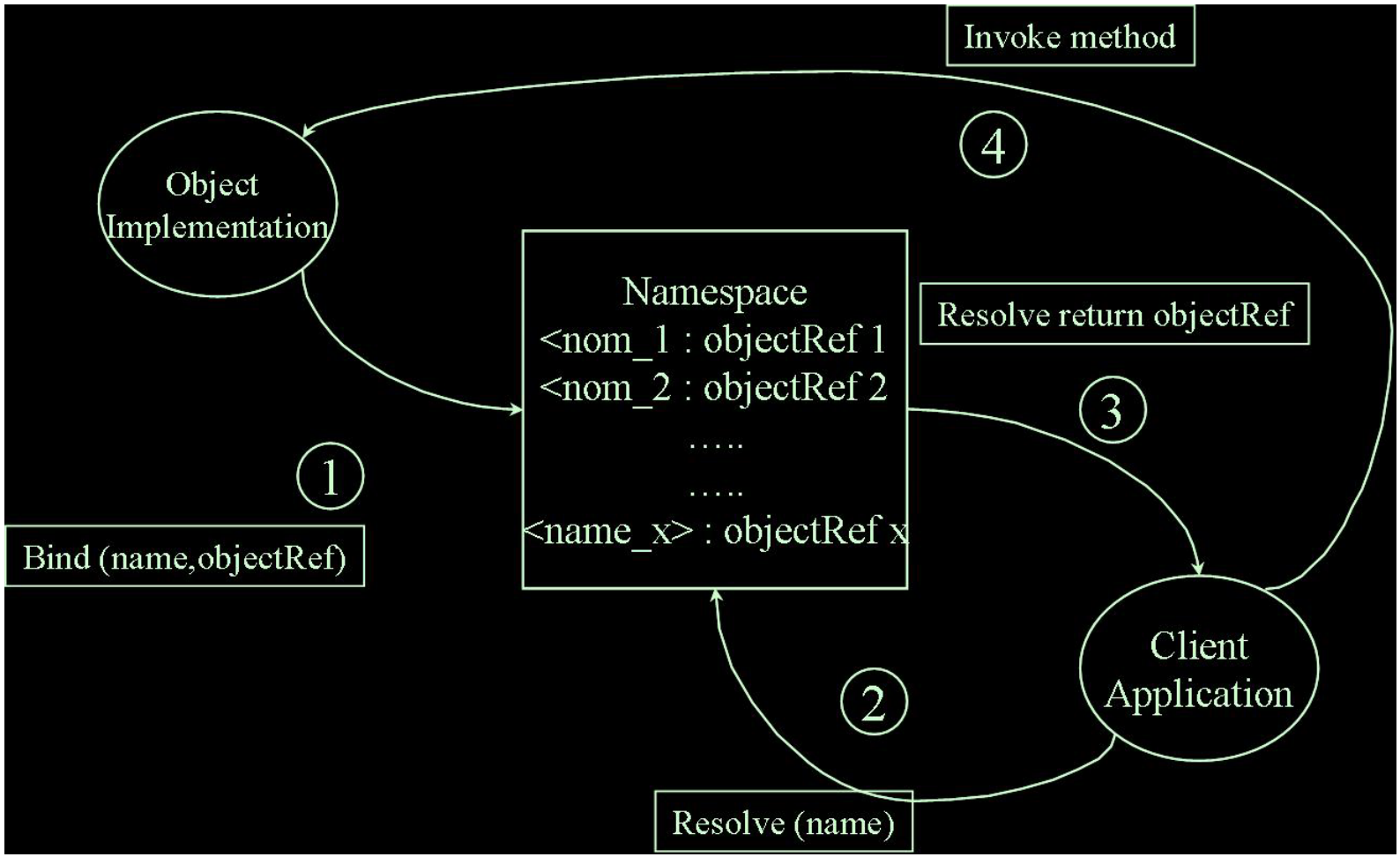
Dans le client:

```
StringHolder themeHolder = new StringHolder();
StringHolder moderatorHolder = new StringHolder();
IntHolder sizeHolder = new IntHolder();
forumProxy.getInfo(themeHolder,moderatorHolder,sizeHolder);
System.out.println("Theme : " + themeHolder.value);
System.out.println("Modérateur : " + moderatorHolder.value);
System.out.println("Nombre de messages : " + sizeHolder.value);
```

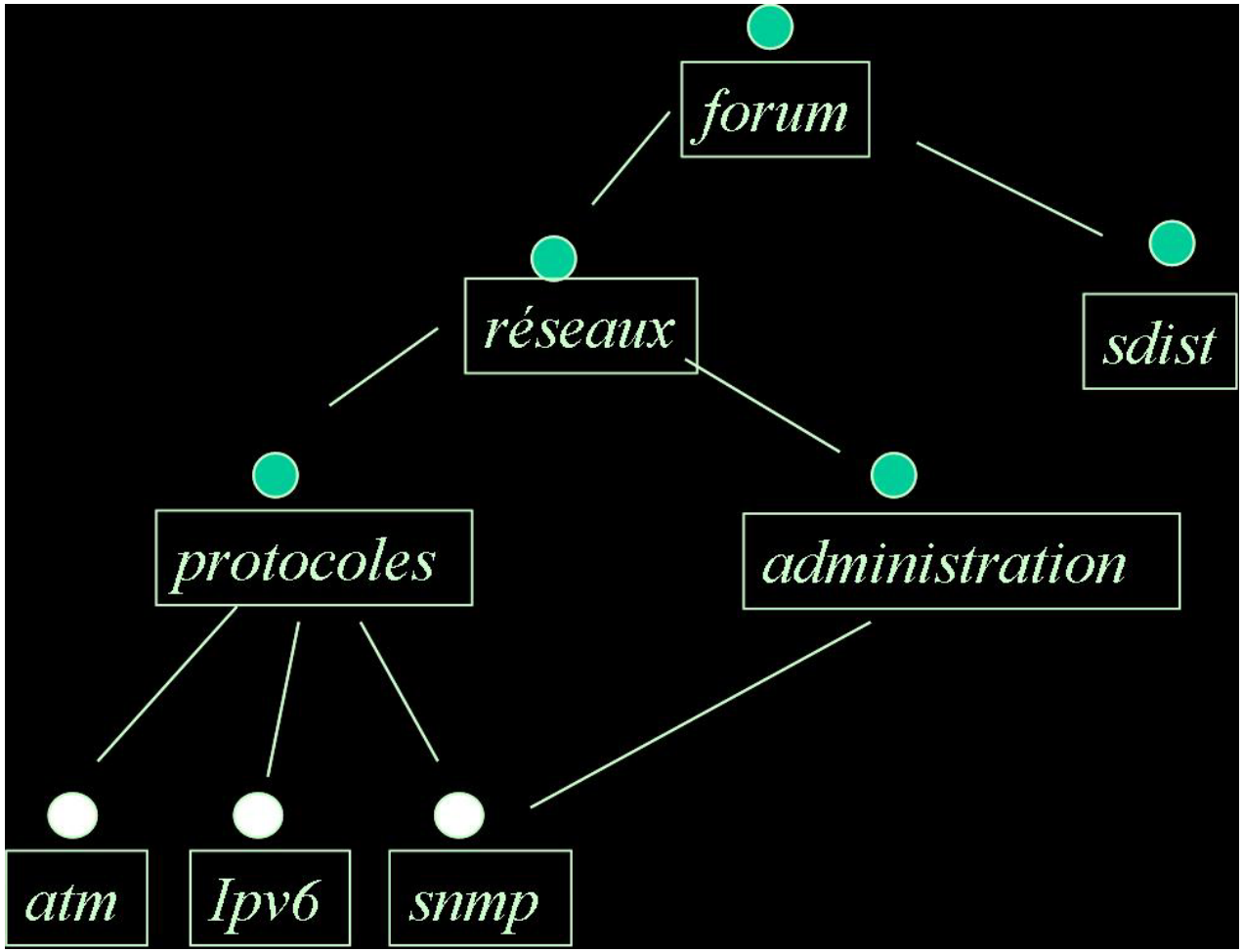
```
interface Forum {
readonly attribute string theme;
readonly attribute string moderator;
boolean postMessage(in Message m);
Message getMessage(in string title);
boolean removeMessage(in string title);
};
interface ForumAdmin {
attribute string theme;
attribute string moderator;
}

public class ForumImpl implements ForumOperations, ForumAdminOperations {

String theme ();
String moderator ();
boolean postMessage (Message m);
Message getMessage (String title);
boolean removeMessage (String title);

void theme (String newTheme);
void moderator (String newModerator);
}
```

```
ORB orb = ORB.init(args, null);
POA rootPOA = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));

ForumImpl  f1 = new ForumImpl ("rtsj","damien");
Forum  ftie = new ForumPOATie (f1);

ForumAdmin  fAtie = new ForumAdminPOATie (f1);

byte[] servantId1 = rootPOA.activate_object(ftie);
byte[] servantId2 = rootPOA.activate_object(fAtie);

NamingContextExt context = NamingContextExtHelper. narrow ( orb. resolve_initia
NameComponent [] name1 = context . to_name (" Forum");
NameComponent [] name2 = context . to_name (" ForumAdmin");
context.rebind (name1 , rootPOA . id_to_reference( servantID1));
context.rebind (name2 , rootPOA . id_to_reference( servantID2));
```

Le Service de nommage est composé de conteneurs (NamingContext) et d'objets associant noms et références des Servants. Les objets NamingContext sont des objets Corba.



Les NamingContext peuvent etre organisés an arbre de nommage. Un nom sera composé: tableau d'éléments "nom.type" et d'un séparateur "/". Exemple: forum.nc/reseaux.nc/protocoles.nc/atm.service



Module CosNaming /Interfaces NamingContext et NamingContextExt

```
module CosNaming {
typedef string Istring;
struct NameComponent  {
Istring id;
Istring kind; }
typedef sequence<NameComponent> Name;
.....................................................
Interface NamingContextExt : NamingContext
{
typedef string StringName;
StringName to_string (in Name n) raises (InvalidName);
Name to_name(in StringName sn) raises (InvalidName);
Object resolve_str (in StringName n);
```

Interface NamingContext(1)

- ajouter une liaison de type liaison d'objet
  `Void bind ( in Name n, in Object obj) raises (NotFound, CannotProceed, InvalidName, AlreadyBound);`
- pour modifier une liaison deja existante :
  `rebind (in Name n, in Object obj):` pour modifier une liaison d'objet
- supprimer une liaison de l'arbre :
  `unbind (in Name n)`
- Retouver un objet dans l'arbre : résolution de désignation
  `Object resolve (in Name n)`

- Créer un nouveau context de désignation
  bind_new_context()
- Modifier une liaison de contexte de désignation rebind_context() : pour modifier une liaison de contexte
- supprimer un contexte de désignation
  destroy()

```
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
.............................
public class ForumServer {
  public static void main(String args[]) throws org.omg.CosNaming.NamingContext
      org.omg.CosNaming.NamingContextPackage.NotFound,
      org.omg.CosNaming.NamingContextPackage.CannotProceed {
.........................
    NamingContextExt namingContext =
      NamingContextExtHelper.narrow(orb
        .resolve_initial_references("NameService"));

    org.omg.CORBA.Object forumRef =
      rootPOA.id_to_reference(forumServantId);

    NameComponent[] name = namingContext.to_name("Forum");

    try {
      namingContext.bind(name,forumRef);
    } catch (AlreadyBound e) {
      System.err.println("Nom : " + name + " déjà utilisé" );
      namingContext.rebind(name,forumRef);
    }
```

```
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
.............................
public class ForumClient {
  public static void main(String[] args) throws org.omg.CosNaming.NamingContext
  org.omg.CosNaming.NamingContextPackage.NotFound,
  org.omg.CosNaming.NamingContextPackage.CannotProceed {
    try {
      ORB orb = ORB.init(args, null);

      NamingContextExt namingContext =
        NamingContextExtHelper.narrow(orb
          .resolve_initial_references("NameService"));

      NameComponent[] name = namingContext.to_name("Forum");

      Forum forumProxy =
        ForumHelper.narrow(namingContext.resolve(name));
```

```
void list (in unsigned long how_many, out BindingList bl, out BindingIterator b
```

on récupËre un itérateur de liaisons via l'opération list de NaminContext

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

une liste de binding de type BindingList est décrite par :

```
struct Binding {
Name binding_name;
BindingType binding_type};
typedef sequence<Binding> BindingList;
```

- Lancement du serveur de nommage [JacORB] java -Djava.endorsed.dirs=/usr/local/apps/JacORB/lib -Djacorb.home=/usr/local/apps/JacORB -Dorg.omg.CORBA.ORBClass=org.jacorb.orb.ORB -Dorg.omg.CORBA.ORBSingletonClass=org.jacorb.orb.ORBSingleton -Djacorb.naming.ior_filename=NSREF -classpath . org.jacorb.naming.NameServer
- Lancement du serveur de nommage [JDK] orbd - ORBInitialPort 1234
- Lancement Client/Serveur [JacORB] java -Djava.endorsed.dirs=/usr/local/apps/JacORB/lib -Djacorb.home=/usr/local/apps/JacORB -Dorg.omg.CORBA.ORBClass=org.jacorb.orb.ORB -Dorg.omg.CORBA.ORBSingletonClass=org.jacorb.orb.ORBSingleton -classpath . %1 -ORBInitRef NameService=file:///d:/serge/dev/corba/NSREF
- Lancement Client/Serveur [JDK] java Client -ORBInitialPort 1234 -ORBInitialHost host java Server -ORBInitialPort 1234 -ORBInitialHost host

Dans le cas de l application Forum: plusieurs types de structures Message

```
struct MessageAudio{
string titre;
string auteur;
}
struct MessageVideo{
string titre;
string auteur;
}
```

Vous modifierez le prototype des opérations posterMessage et lireMessage:

```
Interface Forum{
boolean postMessage(in any m, in string titre) ;
any getMessage(in string titre)
}
```

Création d'un any : create_any de la classe ORB

```
org.omg.CORBA.ORB.Any a = orb.create_any();
```

- pour insérer un type primitif : la classe org.omg.CORBA.Any possède un ensemble de méthodes insert_xxx (xxx = un type primitif (insert_long , insert_string)
- pour insérer un type utilisateur (struct, interface) : la classe Helper associée possède une méthode insert
- extraire un type primitif : méthode extract_xxx de la classe org.omg.CORBA.Any
- extraire un type utilisateur : méthode extract de la classe Helper

Comment connaitre le contenu d'un Any Si un client envoie un paramètre de type Any comment savoir quelle méthode d'extraction utiliser (elle dépend du type contenu). Tout type IDL est associé à un TypeCode (c'est aussi un type IDL). Un élément de type TypeCode contient un élément TcKind. Tous les types IDL possèdent un TcKind. TcKind est une énumération :

```
enum TcKind {tk_null, tk_void, tk_short, tk_long}
```

Comment générer un TypeCode associé à un type IDL

- pour un type primitif : méthode get_primitive_tc() de la classe ORB.
- pour un type utilisateur : méthode type() de la classe Helper

exemple : obtenir le typecode du type float:

```
org.omg.CORBA.TypeCode tc = orb.get_primitive_tc ( org.omg.CORBA.TCKind.tk_floa
```

ex : typecode de la structure C :

```
tc = Chelper.type () ;
```

- pour un type primitif : méthode get_primitive_tc() de la classe ORB.
- pour un type utilisateur : méthode type() de la classe Helper

exemple : obtenir le typecode du type float:

```
org.omg.CORBA.TypeCode tc = orb.get_primitive_tc ( org.omg.CORBA.TCKind.tk_floa
```

ex : typecode de la structure C :

```
tc = Chelper.type () ;
```

```
MessageAudio m1 = new MessageAudio("Nouveau MP3","John","10/10/03","song.mp3");
MessageVideo m2 = new MessageVideo("Nouvelle vidéo","John","15:37","video.avi")
```

On crée deux objets de type Any :

```
org.omg.CORBA.Any any_envoye1 = orb.create_any();
org.omg.CORBA.Any any_envoye2 = orb.create_any();
```

On affecte les Messages dans les any:

```
MessageHelper.insert(any_envoye1,m1);
MessageHelper.insert(any_envoye2,m2);
```

On poste les deux messages :

```
f1.posterMessage(any_envoye1,"Nouvel album 1");
f1.posterMessage(any_envoye2,"Nouvel album 2");
```

Récupération des deux messages

```
org.omg.CORBA.Any any_recu1 = f1.lireMessage("Nouvel album 1");
org.omg.CORBA.Any any_recu2 = f1.lireMessage("Nouvel album 2");
```

On regarde le type du premier message et on l'extrait en fonction de ce dernier :

```
org.omg.CORBA.TypeCode tc1 = any_recu1.type();
switch(tc1.kind().value())
{
case org.omg.CORBA.TCKind._tk_struct :
if(tc1.equal(MessageAudioHelper.type()))
{
System.out.println("Le message est de type MessageAudio");
MessageAudio mess = MessageAudioHelper.extract(any_recu1);
}
else if(tc1.equal(MessageVideoHelper.type()))
{
System.out.println("Le message est de type Message2");
MessageVideo mess = MessageVideoHelper.extract(any_recu1);
}
```

Fonctionalités:

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

Trois types d'intercepteurs: Client, Serveur, IOR.

5 Points d'interception pour un intercepteur Client:

- send_request (sending request)
- send_poll (sending request)
- receive_reply (receiving reply)
- receive_exception (receiving reply)
- receive_other (receiving reply)

5 points d'interception pour un intercepteur Serveur:

- receive_request_service_contexts (receiving request)
- receive_request (receiving request)
- send_reply (sending reply)
- send_exception (sending reply)
- send_other (sending reply)

Un point d'interception pour un intercepteur d'IOR:

`establish_component` appelé lors de la création d'une nouvelle référence d'objet.

L'enregistrement des intercepteurs est fait via l'interface `ORBInitializer`.

Appelé lors de l'initialisation de l'ORB (orb.init()). Permet de créer les intercepteurs

```
public class MyInterceptorORBInitializer extends LocalObject
implements ORBInitializer {
  public static Interceptor interceptor;
  public String name() { return ""; }

  public void pre_init(ORBInitInfo info) {
try {
ClientInter inter = new ClientInter();
info.add_client_request_interceptor(inter);
} catch (Exception ex) {}

public void post_init(ORBInitInfo info) {}
}
```

On ajoute la propriété suivante dans le serveur:

```
Properties props = new Properties();
props.put("org.omg.PortableInterceptor.ORBInitializerClass.monORBInitializer","
ORB orb = ORB.init(args, props);
```

Intercepteur Client(exemple)

```
public class ClientInter extends org.omg.CORBA.LocalObject
implements ClientRequestInterceptor {
public ClientInter(){
}
public String name(){
return "monInterceptor";}
public void destroy(){
}
public void send_request(ClientRequestInfo ri) throws org.omg.PortableIntercept
{
System.out.println( "operation invoquée: " + ri.operation() );
}
public void send_poll(ClientRequestInfo ri){
System.out.println("clientssend_poll");
}
public void receive_reply(ClientRequestInfo ri){
System.out.println("receive_replyC");
}
public void receive_exception(ClientRequestInfo ri){
System.out.println("receive_exceptionC");
}
public void receive_other(ClientRequestInfo ri){
System.out.println("receive_otherC");
```

Intercepteur Client(exemple)

Créer Le Type Service Contexte

```
byte[] context_data Données associées au service context
int context_id  service contextID
```

Pour ajouter le Service Contexte à l'objet ClientRequestInfo

```
public void add_request_service_context
(ServiceContext service_context, boolean†replace)
```

```
public class MonClientRequestInterceptor
extends org.omg.CORBA.LocalObject
implements ClientRequestInterceptor {

static int reqCount = 0;

public void send_request(ClientRequestInfo ri) throws ForwardRequest {
// On ajoute l'information dans le ServiceContext n.199 qui
// nous indique le numero de l'appel fait par la methode send_request
System.out.println("********INTERCEPTOR*********");
System.out.println("la requete est : "+
 ri.operation());
System.out.println("****************************");
reqCount++;
byte[] data = new byte[1];
data[0] = new Integer(reqCount).byteValue();
ServiceContext sc = new ServiceContext(199, data);
ri.add_request_service_context(sc, false);
}
```

Intercepteur Serveur(exemple)

```
import org.omg.PortableInterceptor.*;
public class monORBInitializer extends org.omg.CORBA.LocalObject
implements org.omg.PortableInterceptor.ORBInitializer
{
public void pre_init(ORBInitInfo info){
System.out.println("preinit");
}
public void post_init(ORBInitInfo info){
System.out.println("postinit");
try {
//création des intercepteurs
ServerInter interS = new ServerInter();
IORInter interI = new IORInter();
//enregistrement
info.add_server_request_interceptor(interS);
info.add_ior_interceptor(interI);
}
...
```

```
public void receive_request_service_contexts(ServerRequestInfo ri)
{}
public void receive_request(ServerRequestInfo ri)
{}
public void send_reply(ServerRequestInfo ri)
{}
public void send_exception(ServerRequestInfo ri)
{}
public void send_other(ServerRequestInfo ri)
{}
```

Le service d'événements CORBA permet une communication de type producteur / consommateur avec une transparence entre producteurs et consommateurs (découplage). Le producteur ne connait ni la référence ni le nombre de consommateurs. Le canal est un object dont le comportement vis-a-vis des entités producteur et consommateur est configurable.

- Le canal peut se comporter comme un client vis à vis du producteur il va alors extraire les données du producteur en appelant une méthode *pull* de l'interface du Supplier. On parle dans ce cas de modèle Pull/Supplier.
- Le canal peut se comporter comme un serveur vis à vis du producteur il recoit alors les données transmises par le Supplier. On parle dans ce cas de modèle Push/Supplier.

- La spécification du service Event Chanel porte sur la définition des interfaces *pull* et *push* pour le producteur et pour le consommateur.
- Le canal est typé ou non typé. Si le canal est non typé les données transmises doivent être obligatoirement de type ANY. Les données transmises le sont par l'intermédiaire des paramètres des opérations pull() et push().
- Un mécanisme est prévu pour les transmissions typées, c'est toujours le même type de données qui transitera dans le canal.

```
import org.jacorb.events.*;
import org.omg.CosEventChannelAdmin.*;
import org.omg.CosNaming.*;

    org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(argv, null);
    POA poa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));

    NamingContextExt nc = NamingContextExtHelper.narrow(
              orb.resolve_initial_references("NameService"));

    EventChannelImpl channel = new EventChannelImpl(orb,poa);
    poa.the_POAManager().activate();

    byte[] channel_id = poa.activate_object(channel);
    org.omg.CORBA.Object o = poa.id_to_reference(channel_id);
    nc.bind(nc.to_name("eventchannel"), o);

    orb.run();
```

- Obtention de la référence d'un objet d'administration et de la référence d'un objet proxy.

```
SupplierAdmin supplierAdmin = cnl.for_suppliers();
PoxyPushConsumer ppcons = supplierAdmin.obtain_push_consumer();
```

- Initialisation et activation (enregistrement auprès du POA) de l'objet producteur (servant).

```
PushSupplier supplier = new PushSupplier(orb , consumer);
byte[] supplierId = rootPOA.activate_object(supplier);
```

- Connection du producteur au canal (transmission de la référence du producteur au canal pour les appels callback.

```
ppcons.connect_push_supplier (rootPOA.id_to_reference(supplierId)));
```

- production

```
public void run () {
  org.omg.CORBA.Any any = orb.create_any();
any.insert_string("abc");
consumer.push(any);
}
```

Push Supplier/2

- Construction de la classe producteur (PushSupplier)

```
public void disconnect_push_supplier ()
{
  System.out.println ("Supplier disconnected");
}
```

Pull Supplier/1

- demande d'un proxy vers l'event channel

- le supplier demande un proxy du consumer en mode pull à l'admin du channel

```
SupplierAdmin supplierAdmin = e.for_suppliers();
ProxyPullConsumer proxyPullConsumer =
       supplierAdmin.obtain_pull_consumer();
```

- création de l'objet (servant) supplier

```
PullSupplierPOATie pt = new PullSupplierPOATie( new PullSupplier());
org.omg.CORBA.Object o = poa.servant_to_reference( pt );
```

- enregistrement du supplier au canal

```
proxyPullConsumer.connect_pull_supplier( PullSupplierHelper.narrow(o) );
```

Pull Supplier/2

Création de la classe PullSupplier:

```
class PullSupplier implements PullSupplierOperations
public Any pull() throws Disconnected
  {
    System.out.println("I m being pulled.");
    event = org.omg.CORBA.ORB.init().create_any();
    event.insert_string("Pull.");
    return event;
  }
public void disconnect_pull_supplier()
  {
  System.out.println("Bye.");
  }
```

Pull Consumer/1

```
NamingContextExt nc =
      NamingContextExtHelper.narrow(orb.resolve_initial_references("NameService
ecs = EventChannelHelper.narrow(nc.resolve(nc.to_name("eventchannel.example")))

pullConsumer = new ForumPullConsumer();
ca  = ecs.for_consumers();
pps = ca.obtain_pull_supplier();
pps.connect_pull_consumer(pullConsumer);
```

```
int i=0;
while( i<10 )
{
System.out.println("pulling event " + i);
org.omg.CORBA.BooleanHolder bh = new org.omg.CORBA.BooleanHolder();
try{
   received = pps.try_pull(bh);
// received = pps.pull();
   if( bh.value ){
    System.out.println("received " + (i++) + " : " +
        received.extract_string() );
     }
     else{
          Thread.currentThread().sleep(2000);
        }
    }
    pps.disconnect_pull_supplier();
  }
}
```