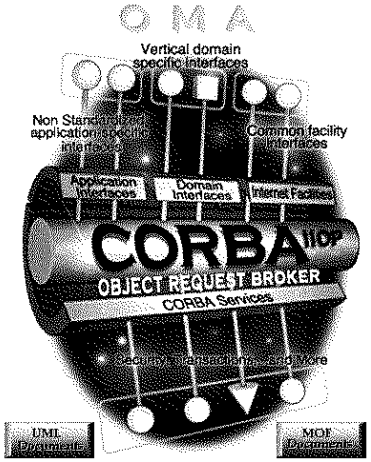
Objets Distribués



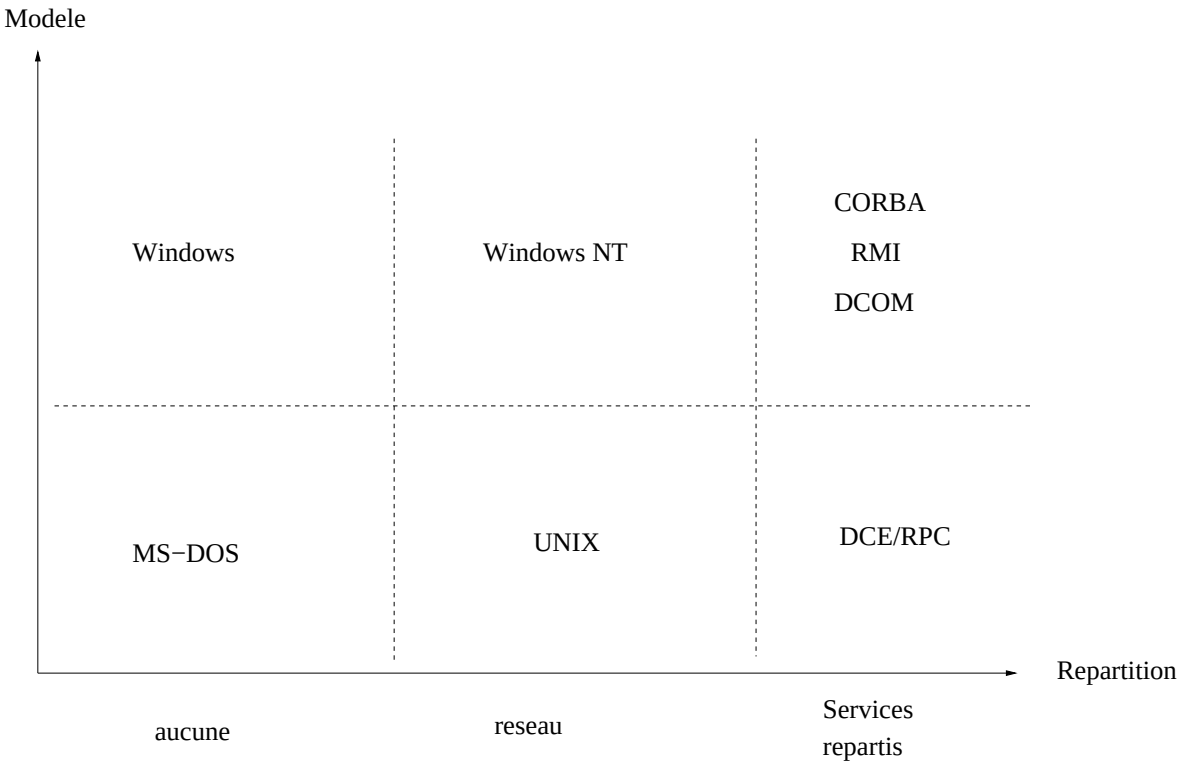
Alexis Nédélec

Ecole Nationale d'Ingénieurs de Brest
Technopôle Brest-Iroise, Site de la Pointe du Diable
CP 15 29608 BREST Cedex (FRANCE)
e-mail : nedelec@enib.fr

Table des Matières

Table des Matières

Introduction

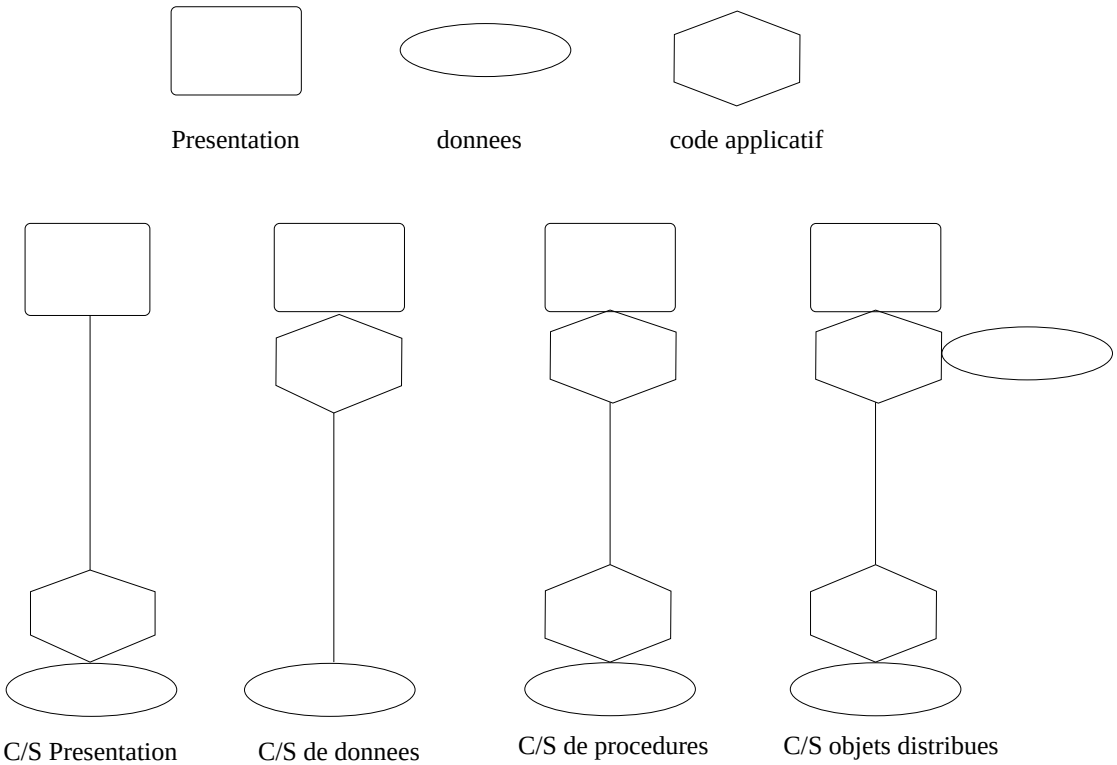


Introduction

Architecture client/serveur distribuée, répartie

- ▷ architecture à deux niveaux, strates (2-tiered C/S Architecture)
  - ◇ une application cliente accède directement à un serveur de données
  - ◇ le serveur de données gère les requêtes clientes
  - ◇ problème du nombre de clients: moniteurs transactionnels
- ▷ architecture à trois niveaux, strates (3-tiered C/S Architecture)
  - ◇ serveur applicatif (middle-tier) entre clients et serveurs de données
- ▷ architecture middleware
  - ◇ répartition du serveur applicatif

Introduction



Introduction

Deux couches distinctes dans les systèmes distribués

- ▷ couche services (objets d'applications)
- ▷ couche transport (administration d'objets)

Répartition d'applications

- ▷ Sockets: développement fastidieux
- ▷ RPC (**R**emote **P**rocedure **C**all): approche procédurale
- ▷ DCE (**D**istributed **C**omputing **E**nvironment): IDL procédural
- ▷ CORBA: IDL objet (**I**nterface **D**escription **L**anguage)
- ▷ DCOM: IDL Microsoft, dérivé de DCE et CORBA
- ▷ RMI (**R**emote **M**ethod **I**nvocation): Java, JNI, RMP (IIOP), EJB

Introduction

Principaux concurrents

- ▷ **CORBA**: **C**ommon **O**bject **R**equest **B**roker **A**rchitecture
- ▷ **DCOM**: **D**istributed **C**omponent **O**bject **M**odel
- ▷ **EJB**: **E**nterprise **J**ava**B**eans

Pour pouvoir faire

- ▷ communiquer des objets
- ▷ répartis sur plusieurs machines
- ▷ entre applications développées séparément
- ▷ sur des plateformes hétérogènes

L'objectif est de définir des "composants" objets pour applications distribuées

Object Management Group

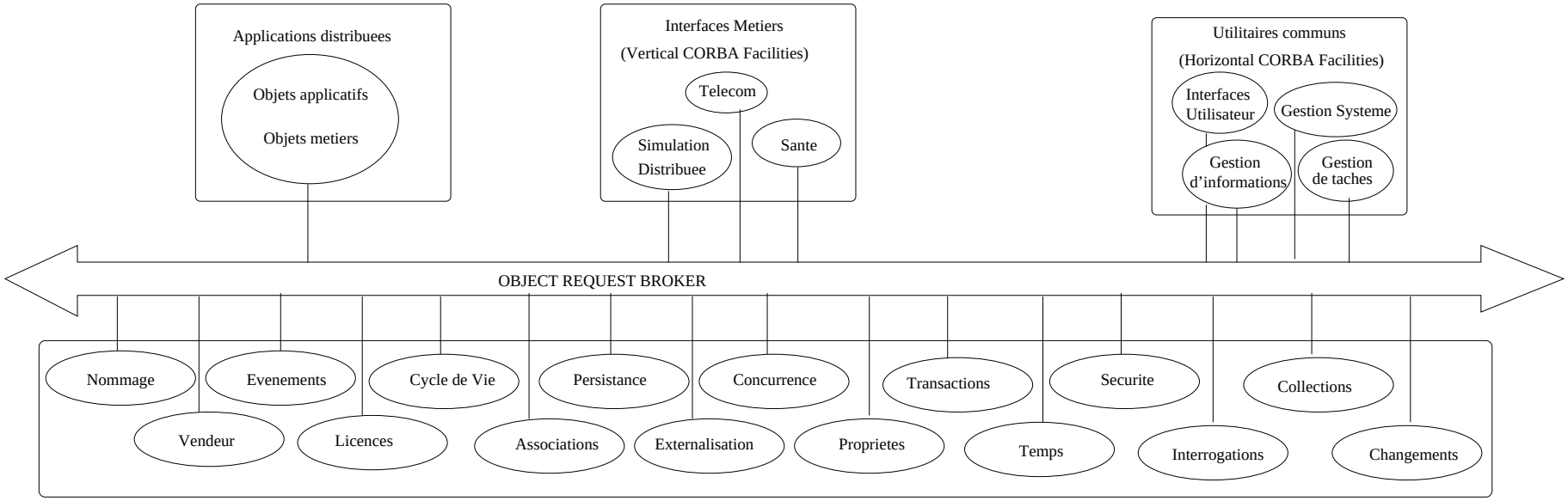
Site de l'OMG: `http://www.omg.org`

- ▷ Consortium fondé en 1989 pour
  - ◇ promouvoir la technologie Objet
  - ◇ dans des systèmes distribués
  - ◇ en environnements hétérogènes
- ▷ actuellement plus de 800 compagnies membres
  - ◇ constructeurs (IBM, Compaq, NCR,...)
  - ◇ éditeurs (Inprise, Netscape, Oracle, Rational,...)
  - ◇ universitaires (CERN, CNET, ...)
  - ◇ utilisateurs industriels (Air France, Alcatel, France Telecom,...)
  - ◇ ...

Groupe de travail pour définir une architecture universelle: **OMA**

Object Management Architecture

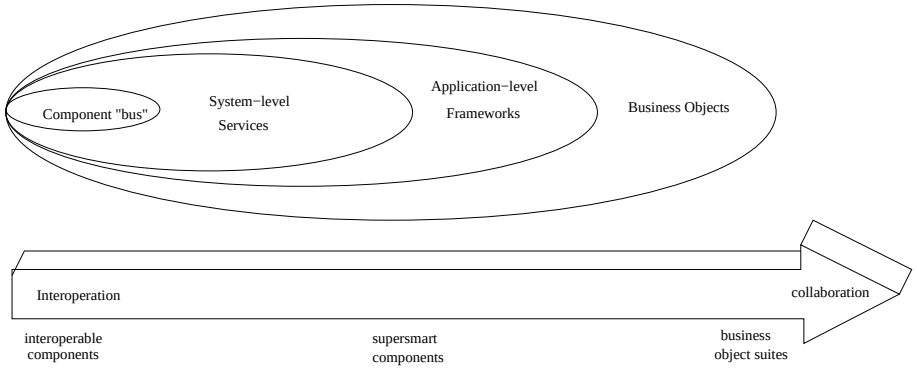
Architecture pour objets distribués



Object Management Architecture

De l'interopérabilité à la collaboration

- ▷ bus à objets (ORB) : au-delà des langages, systèmes, réseaux
- ▷ services objets (COSS) : nommage, événements, transaction, sécurité, ...
- ▷ canevas applicatifs : Corba Facilities
- ▷ Objets applicatifs: collaboration entre objets



Common Object Request Broker Architecture

Objectifs de la norme CORBA pour objets distribués

- ▷ bus Objet (ORB) reliant les clients aux implémentations d'objets
- ▷ langage de description (IDL) des services d'objets
- ▷ projection (mapping) de l'IDL vers les langages d'implémentation
- ▷ invocation statique d'opérations sur des objets (SII, SSI)
- ▷ invocation dynamique d'opérations (DII, DSI)
- ▷ entrepôt d'interfaces et d'implémentation (IR)
- ▷ références d'objets (IOR : Interoperable Object Reference)
- ▷ support d'adaptation pour l'objet serveur (BOA, POA,..)
- ▷ protocoles d'interopérabilité (GIOP, IIOP, ESIOP, ...)

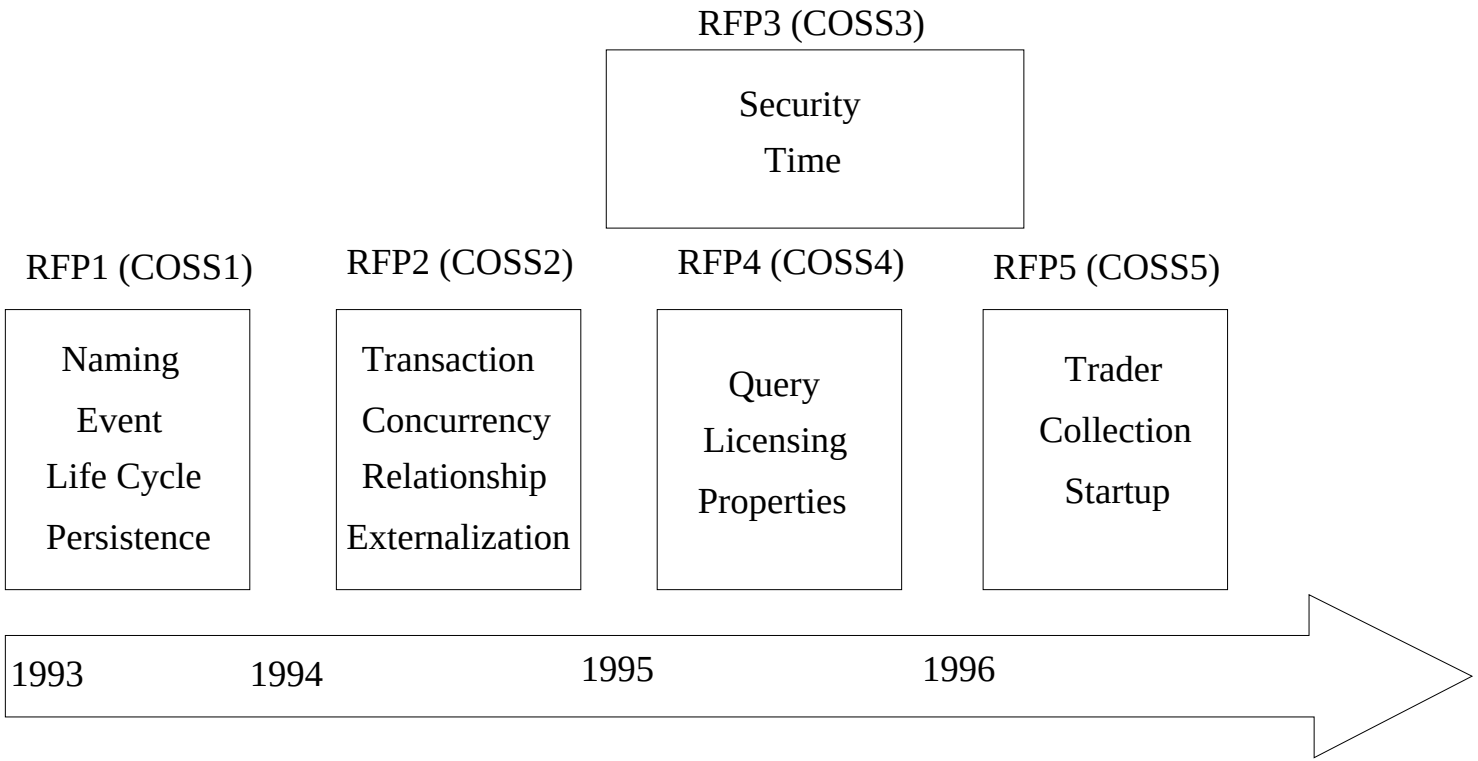
Historique

Différentes évolutions de la norme

- ▷ CORBA 1.0 (1991): manque d'interopérabilité entre les ORBs
- ▷ CORBA 2.0 (1994): standardisation des protocoles GIOP, IIOP
- ▷ CORBA 2.3 (1999):
  - ◇ intégration des RMI dans le protocle IIOP
  - ◇ amélioration du POA (**P**ortable **O**bject **A**daptator)
  - ◇ transmission d'objest par valeurs (OBV: Object By Value)
- ▷ CORBA 3.0 (en cours!):
  - ◇ modèles de composants (intégration des JavaBeans)
  - ◇ spécification des Firewalls
  - ◇ services de nommage interopérable (intégration des IOR)
  - ◇ services de messages asynchrones: Message Oriented Middleware (MOM)

Common Object Services Specifications

COSS: ensemble des services pour objets CORBA



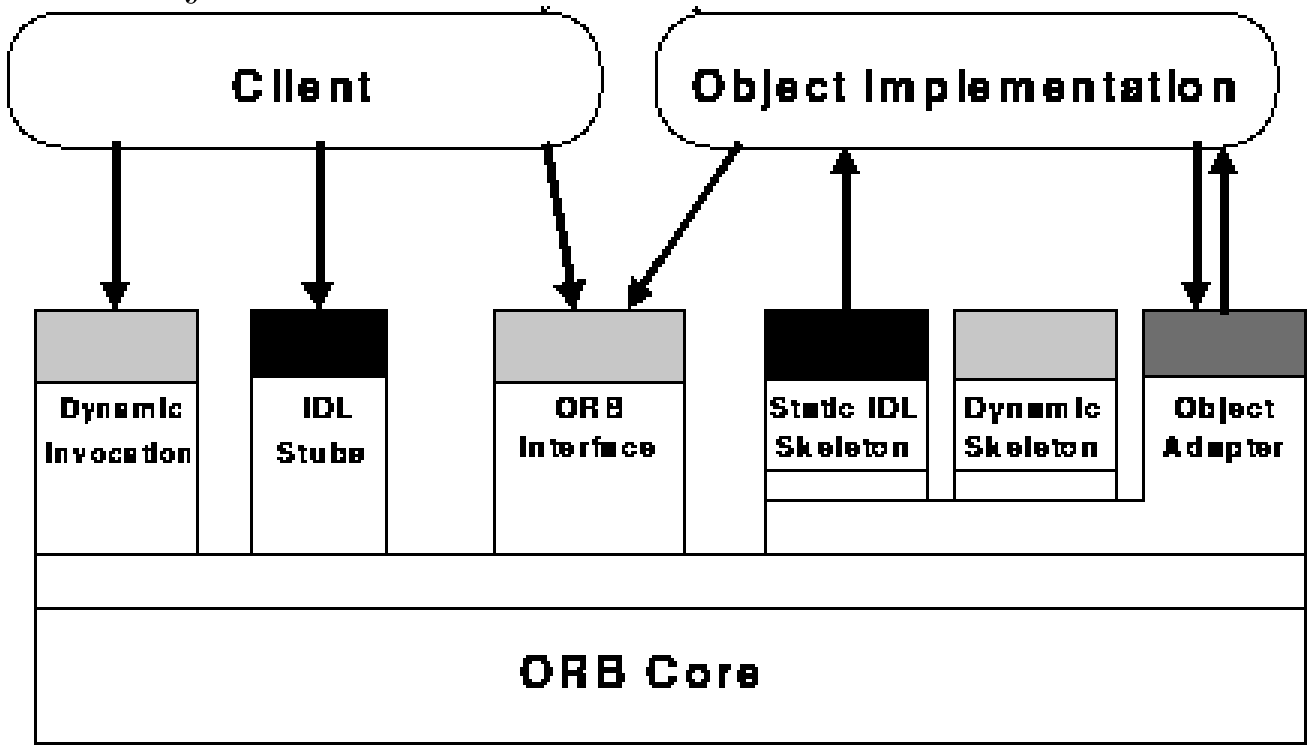
Common Object Services Specifications

Autre classement des services issus des RFP (**R**equests **F**or **P**roposal)

- ▷ Annuaire : Naming, Trader
- ▷ Communication : Events, Transaction, Concurrency
- ▷ Problèmes existentiels : LifeCycle, Persistence, Relationship
- ▷ Manipulation : Collections, Query, Properties, Externalisation
- ▷ Protections : Security, Licensing
- ▷ Décalage Horaire : Time
- ▷ Réveil : Startup

Object Request Broker

ORB: le bus à objets de CORBA



Object Request Broker

Caractéristiques d'un ORB coté client

- ▷ Invocation Statique sur un objet distant via un IDL stub (SII)
- ▷ Invocation Dynamique d'Interfaces (DII):
  - ◇ lorsqu'on découvre l'objet à l'éxécution
  - ◇ utilisation d' un Référentiel d'Interfaces (IR)

Caractéristiques d'un ORB coté serveur

- ▷ Invocation Statique sur un objet via un IDL skeleton (SSI)
- ▷ Invocation Dynamique d'Interfaces (DSI) :
  - ◇ interception des requêtes clients (DII)
  - ◇ utilisation d'un Référentiel d'Implémentation (IR)
- ▷ gestion des objets de l'application serveur l'adaptateur d'objet (POA)

Object Request Broker

Principe de la communication téléphonique

- ▷ un téléphone
- ▷ un numéro de correspondant
- ▷ on ignore tout de la connexion

Invocation statique d'objet distant

- ▷ un objet intermédiaire (talon, squelette)
- ▷ un identifiant de l'objet à invoquer (référence d'objet)
- ▷ un bus de communication, l'ORB

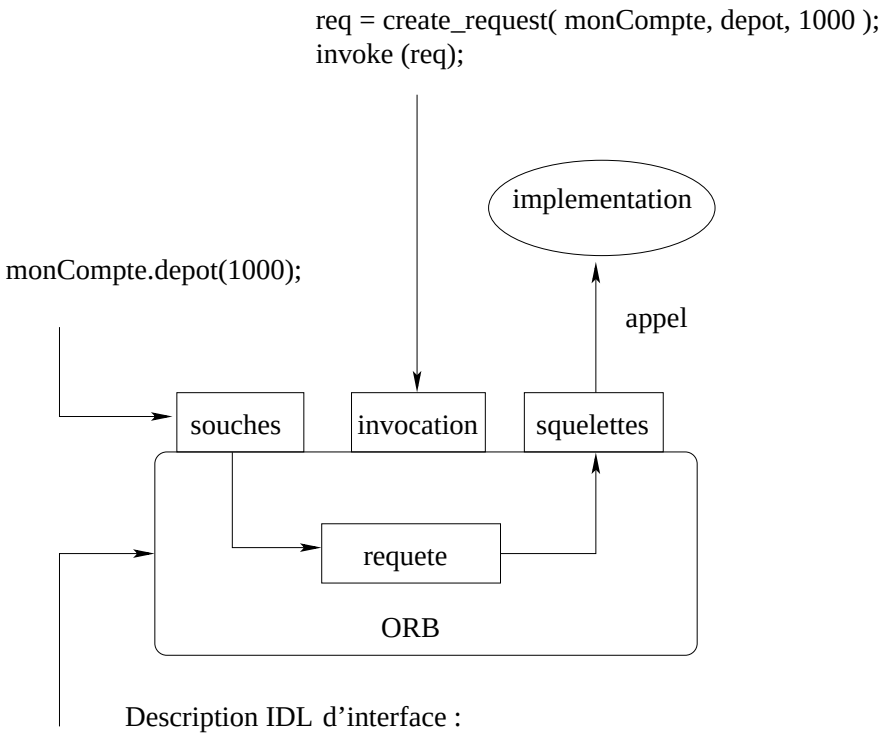
Object Request Broker

```
Talon::methode(args) {    |  Squelette::methode(requete r) {
  codage(r,args);         |    decodage(r,args);
  envoi(r);               |    Serveur::methode(args);
  attente_reponse(r);     |    codage(r,args);
  decodage(r,args);       |    envoi_resultat(r);
}                         |  }
```

Talons et squelettes sont:

- ▷ chargés de l'empaquetage, dépaquetage des informations
- ▷ générés à partir d'une interface IDL
- ▷ stockés dans un Référentiel d'Interfaces (IR)

Object Request Broker

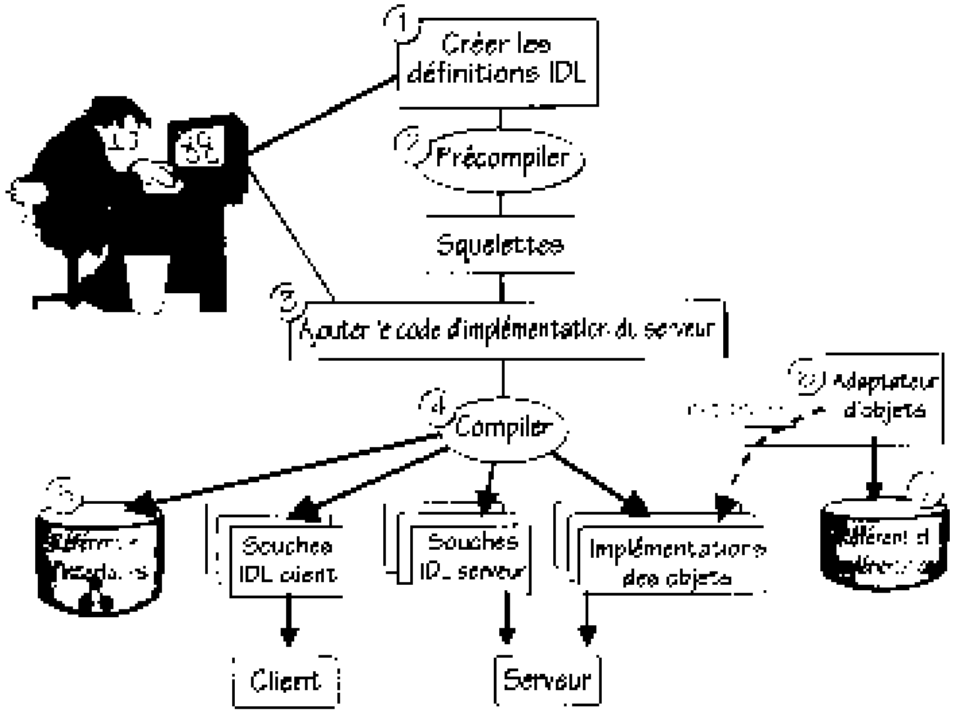


Description IDL d'interface :

```
interface Compte {
  attribute long numero;
  long depot (in long montant );
  long retrait ( in long montant );
}
```

Interface Definition Language

Langage universel pour décrire les services d'objets du serveur aux clients



Interface Definition Language

Langage pour la définition de “services” dans des “contextes”

- ▷ description des échanges entre clients et serveurs
- ▷ dans une approche orienté objet
- ▷ avec une syntaxe proche du `C++` et de Java

L’IDL repose sur les notions

- ▷ Module: espace de nommage regroupant un ensemble d’interfaces
- ▷ Interface: équivalent d’une classe définissant les services
- ▷ Opérations: description des services (signature de méthodes)
- ▷ Types de données: décrivant les arguments d’opérations.
  - ◇ de base: short, long, float, char, boolean,...
  - ◇ structurés: enum, union, string, struct, array, sequence, any ,...

Interface Definition Language

```
module Bancaire  {
  // declarations
...
  interface Compte {
    // attributs  et operations
...
  };
  interface Client {
    // attributs  et operations
  };
...
};
```

Interface Definition Language

```
module <ident> {
  <types      decl>;
  <constants  decl>;
  <exceptions decl>;

  interface <ident> [:<inheritance>] {
         <types       decl>;
         <constants   decl>;
         <attributes  decl>:
         <exceptions  decl>;
...
        [<op_type>] <ident>(<params>)
         [raises(<exceptions>)][context];

  };
  interface <ident> [:<inheritance>] {
...
  };
};
```

```
module Bancaire  {
  const string banque    = ''CAIO''
  const float  maxCompte = 100.000;


  interface Compte {
    readonly attribute long numero;
    attribute string nom;
    exception valeurInvalide {float exces};

    float consulter();
    float depot  (in float montant);
    float retrait(in float montant) raises (valeurInvalide);
  };


};
```

Attributs

Les attributs peuvent-être en
- ▷ lecture, écriture
- ▷ lecture seule (`readonly`)

Description IDL d'un attribut :

```
attribute float aFloat
readonly attribute float aFloat
```

La génération de code (mapping) en C++:

```
CORBA::Float aFloat() const;        // en lecture
void  aFloat(CORBA::Float aFloat);  // en ecriture
```

Opérations

Pour chaque opération il faut définir:

- ▷ type des paramètres et nommage des paramètres
- ▷ mode de transmission du paramètre (`in, out, inout`)
- ▷ si aucune valeur de retour, mode asynchrone possible (`oneway`)

```
module Arguments {
  interface PassageDeParametres{
    void  passageInOut ( in    long    inLong,
                         out   long   outLong,
                         inout long inoutLong);
  };
};
```

Opérations

Mapping Java, incomplet, de l'opération précédente

```
package Arguments;
  public PassageDeParametres {
    public void  passageInOut ( int                       inLong,
                                org.omg.CORBA.IntHolder   outLong,
                                org.omg.CORBA.IntHolder   inoutLong );
  };
};
```

Opérations

Implémentation du coté client

```
...
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(...);
PassageDeParametres objClient = PassageDeParametresHelper.bind(orb);
IntHolder outLong = new IntHolder();
IntHolder inoutLong = new IntHolder(0);
objClient.passageInOut ( 0, outLong, inoutLong);
System.out.println("out param: " +  outLong.value);
System.out.println("inout param: " +  inoutLong.value);
...
```

Opérations

Mapping C++, incomplet, de l'opération précédente

```
class  PassageDeParametres:
  public  Arguments::PassageDeParametres_ops, ... {
    public:
    ...
    virtual void  passageInOut ( CORBA::Long     _inLong,
                                 CORBA::Long_out _outLong,
                                 CORBA::Long&    _inoutLong );
};
```

Héritage

Restriction sur l'héritage

- ▷ interdiction de redéfinition des noms et type d'attributs hérités
- ▷ pas de redéfinition ou surcharge des opérations de base
- ▷ pas d'héritage multiple d'interfaces ayant des opérations de même nom

```
module Heritage {
  interface InterfaceInherit: Interface1, Interface2, ..  {
  };
};
```

Si l'héritage provient d'une classe d'un autre module:

```
module Heritage {
  interface InterfaceInherit: Heritage1::Interface1, Interface2, ..  {
  };
};
```

Références Circulaires

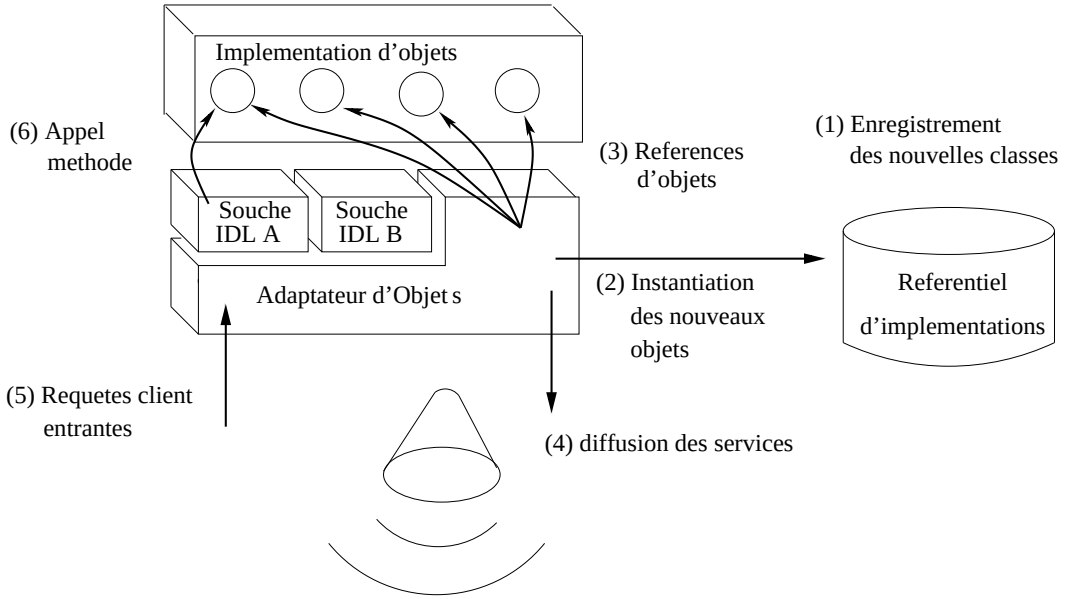
Déclaration Avancée (forward declaration)

```
module ForwardDeclaration {
  interface B;
  interface A {
    void useB( in B aB );
  };
  interface B {
    void useA( in A anA );
  };
};
```

Portable Object Adapter

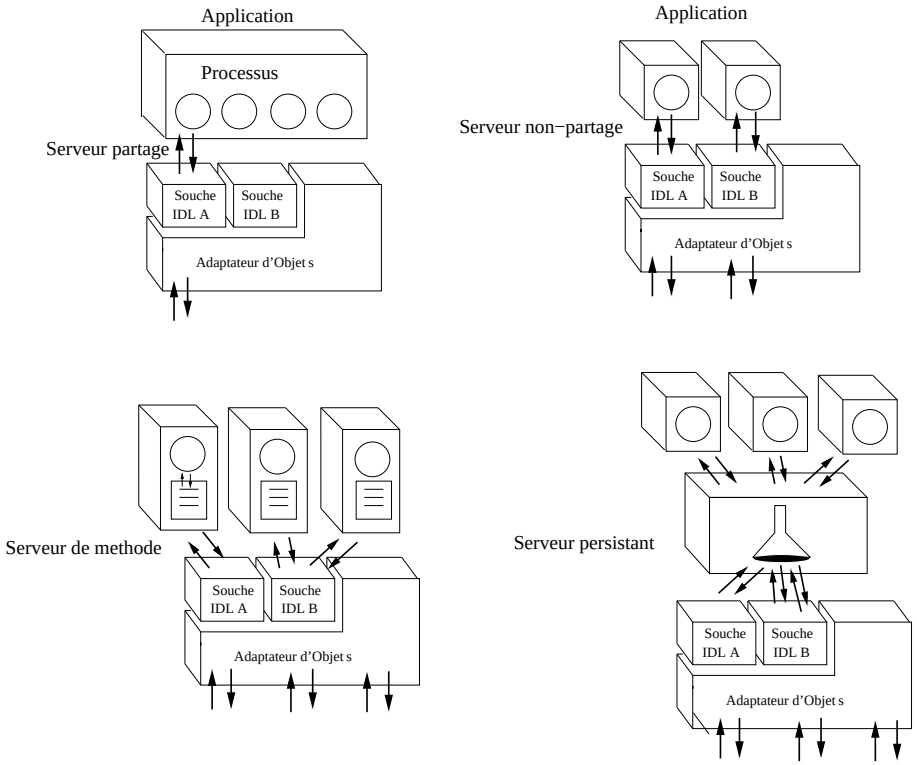
Adapteur d'objets permettant aux objets serveurs de recevoir les appels:

- ▷ BOA: Adapteur minimal imposé par la norme CORBA 1.0
- ▷ POA: Evolution de l'adapteur minimal pour l'interopérabilité (CORBA 2.0)



Portable Object Adapter

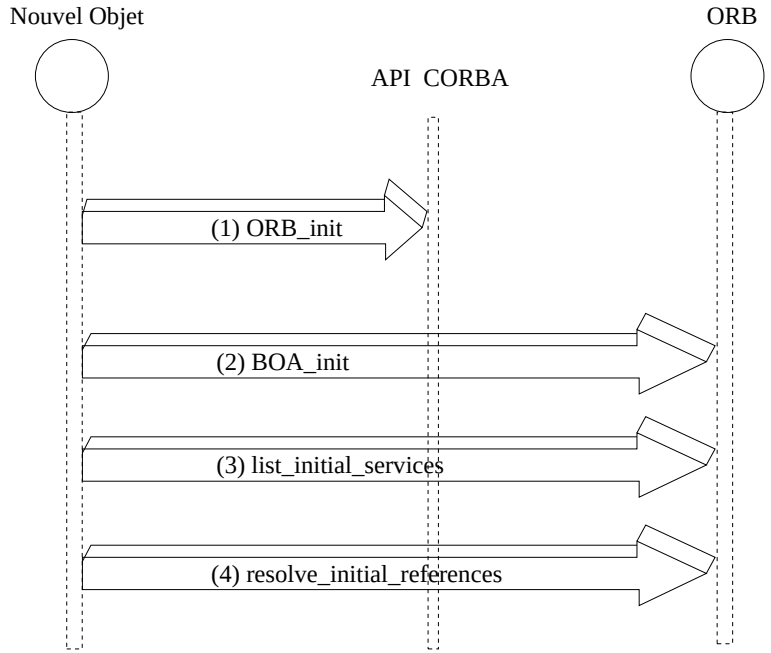
Quatre types d'activation d'objets pour le BOA



Portable Object Adapter

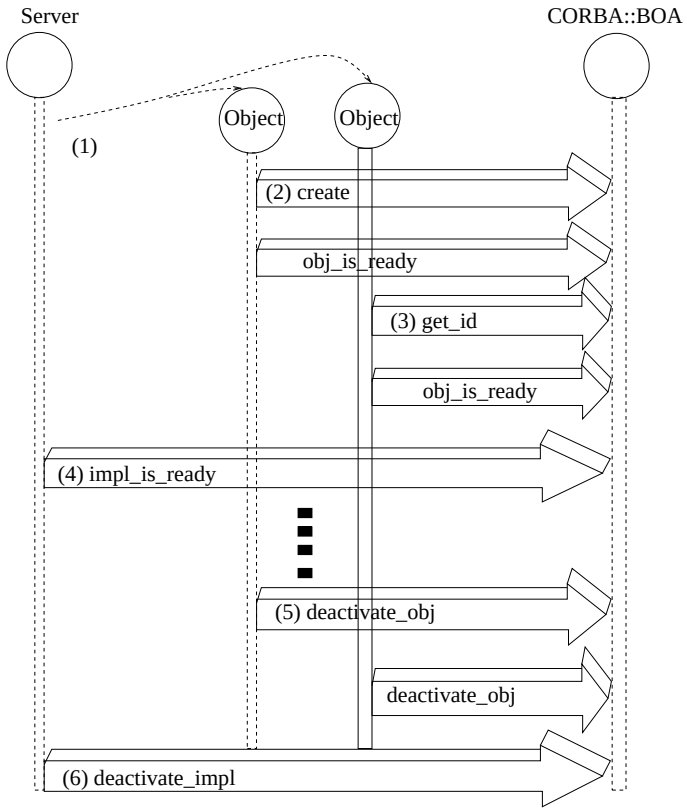
Coté serveur:

1. connexion sur l'ORB, Initialisation du BOA (POA)
2. présentation des services offerts, accès aux services



Portable Object Adapter

Mise à disposition des objets instanciés sur le serveur



Portable Object Adapter

Evolution de la norme pour

- ▷ assurer la portabilité du code serveur entre ORBs
- ▷ activation implicite des objets serveur
- ▷ gérer plusieurs POA sur un même serveur

Apparition de nouvelle terminologie

- ▷ servant: autre appelation de l'implémentation d'objet
- ▷ serveur: application proposant des servants aux clients
- ▷ client: application utilisant les services des objets servants

Broker JacORB

Définition IDL d'un objet serveur (`server.idl`)

```
module jacorb {
  module demo {
    module example1 {
      typedef sequence < string > stringSeq;
      typedef string stringArray[5];
      interface server {
        string writeMessage(in string a1);
        void arryfy1(in string a1, out stringSeq s);
        void arryfy2(in string a1, out stringArray s);
      };
    };
  };
};
```

Appel de méthode distante, utilisation des paramêtres et valeurs de retour

- `string writeMessage(in string a1);`
- `void arryfy2(in string a1, out stringArray s);`

Téléchargement: `http://www.jacorb.org`

Broker JacORB

Mapping de l'IDL en Java (`server.java`) avec l'outil `idl2j`

```
package jacorb.demo.example1;

public interface server extends org.omg.CORBA.CORBject {
        java.lang.String writeMessage(java.lang.String a1);
        java.lang.String writeMessages(java.lang.String[] a1);
        void arryfy1(java.lang.String a1, int a2, jacorb.Orb.SequenceOutHolder/*out*/ s);
        void arryfy2(java.lang.String a1, jacorb.demo.example1.StringArray5OutHolder/*out*/ s);
}
```

Génération automatique des talons client-serveur (`jgen server.class`)

- ▷ `serverStub.java, serverSkeleton.java`

N.B: dernières version JacORB 1.3:
le compilateur IDL (`idl`) génère les classes Java

- ▷ `server, serverOperations`
- ▷ `_serverStub, serverPOA, serverPOATie`

Broker JacORB

Implémentation des services de l'objet en Java

```
package jacorb.demo.example1;

public class serverImpl implements server {

  public String writeMessage( String s ){
    System.out.println("Message: " + s );
    return s + " written";
  }
  public void arryfy1(java.lang.String a1, int a2, jacorb.Orb.SequenceOutHolder/*out*/ s) {
    String result [] = new String[a2];
    for( int j = 0; j < a2; j++ ) result[j] = a1;
    s.set_value(result);
  }
  public void arryfy2(java.lang.String a1, jacorb.demo.example1.StringArray5OutHolder/*out*/ s) {
    int sz = s.size();
    String result [] = new String[sz];
    for( int j = 0; j < sz; j++ ) result[j] = a1;
    s.set_value(new StringArray5(result));
  }
}
```

Broker JacORB

Implémentation du serveur d'objet (`remoteServer.java`)

```
package jacorb.demo.example1;
import org.omg.CosNaming.*;

public class remoteServer {
  public static void main( String[] args )
  {
      org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);
      try
      {
        org.omg.PortableServer.POA poa =
           org.omg.PortableServer.POAHelper.narrow(orb.resolve_initial_references("RootPOA") );
        poa.the_POAManager().activate();
        org.omg.CORBA.Object obj = poa.servant_to_reference(new serverImpl());
        ...
        NamingContextExt nc =
           NamingContextExtHelper.narrow(orb.resolve_initial_references("NameService") );
        nc.bind( nc.to_name("serverObject"), obj);

      } catch (Exception e ) { e.printStackTrace(); }
      orb.run();
  }
}
```

Broker JacORB

Implémentation du client (`client.java`)

```
package jacorb.demo.example1;
import org.omg.CosNaming.*;

public class Client {
  public static void main( String[] args )
  {
    ...
    try
    {
      server s;
      org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);
      ...
      NamingContextExt nc = NamingContextExtHelper.narrow(orb.resolve_initial_references("NameService") );
      s =  serverHelper.narrow ( nc.resolve(nc.to_name("serverObject")) );
      ...
      System.out.println( s.writeMessage( args[0] ));
      StringArray5OutHolder sh5 = new StringArray5OutHolder();
      s.arryfy2( args[0], sh5 );
      String a[] = sh5.value();
      for( int i = 0; i < a.length; i++) System.out.println("From example1: " + a[i] + " " + i );
    } catch (Exception e){ e.printStackTrace(); }
  }
}
```

Broker JacORB

Lancement du serveur:

▷ `$ java jacorb.demo.example1.remoteServer`

Lancement du client:

▷ `$ java jacorb.demo.example1.client hello`

Affichage coté serveur

```
Message: hello          --->  s.writeMessage( args[0] );
```

Affichage coté client

```
hello written           --->  System.out.println( s.writeMessage(args[0]) );
From Example1: hello 0  --->  for( int i = 0; i < a.length; i++)
From Example1: hello 1  --->        System.out.println("From example1: "+a[i]+" "+i);
From Example1: hello 2
From Example1: hello 3
From Example1: hello 4
```

Broker ORBit

Le broker de GNOME (GNU Network Object Model Environment)

Exemple de définition d'IDL CORBA dans un fichier: `messageBox.idl`

```
module HELLOWORLD {
  interface MessageBox
  {
    void addMessage ( in string text);
  };
};
```

Broker ORBit

Compilation et génération de code C à partir du fichier IDL

```
{logname@hostname} orbit-idl --skeleton-impl messageBox.idl
```

Fichiers générés

- ▷ `messageBox.h` : interface C correspondant à l'IDL
- ▷ `messageBox-skelimpl.c`: pour l'implémentation de l'objet serveur
- ▷ `messageBox-common.c`: code commun au client et au serveur
- ▷ `messageBox-skels.c`: représentant local du serveur
- ▷ `messageBox-stubs.c`: représentant local du client

Broker ORBit

Déclarations des types et structures pour un objet CORBA:

```
typedef CORBA_Object HELLOWORLD_MessageBox;
```

Ce qui représente la définition

- ▷ d'un Object CORBA (`MessageBox`),
- ▷ dans un contexte de nommage spécifique (`HELLOWORLD`).

Structure nécessaire au POA (Portable Object Adapter):

```
typedef struct {
  void *_private;
  void (*addMessage)( PortableServer_Servant _servant,
                      CORBA_char * text,
                      CORBA_Environment *ev );
} POA_HELLOWORLD_MessageBox__epv;
```

Broker ORBit

Implémentation de l'objet serveur CORBA

- ▷ `impl_POA_HELLOWORLD_MessageBox` : structure représentant les servants
- ▷ `impl_HELLOWORLD_MessageBox__create()` : initialisation de l'objet serveur
- ▷ `impl_HELLOWORLD_MessageBox__destroy()`: destruction de l'objet serveur
- ▷ `impl_HELLOWORLD_MessageBox_addMessage()`: implémentation de l'objet serveur

```
typedef struct {
  POA_HELLOWORLD_MessageBox servant;
  PortableServer_POA        poa;
            /****************** inserted code       ***************/
  GtkMsgBox *gtk_msg_box;
            /****************** end of inserted code ***************/
} impl_POA_HELLOWORLD_MessageBox;
```

messageBox-skelimpl.c

Initialisation de la structure de l'objet serveur (newservant)

```
static
HELLOWORLD_MessageBox impl_HELLOWORLD_MessageBox__create( PortableServer_POA poa,
                                                          CORBA_Environment *ev) {
  HELLOWORLD_MessageBox retval;
  impl_POA_HELLOWORLD_MessageBox *newservant;
  PortableServer_ObjectId *objid;

  newservant = g_new0(impl_POA_HELLOWORLD_MessageBox, 1);
  newservant->servant.vepv = &impl_HELLOWORLD_MessageBox_vepv;
  newservant->poa = poa;
            /****************** inserted code       ****************/
  newservant->gtk_msg_box = gtk_msgbox_new ();
            /****************** end of inserted code ***************/
  POA_HELLOWORLD_MessageBox__init((PortableServer_Servant)newservant, ev);
  objid = PortableServer_POA_activate_object(poa, newservant, ev);
  CORBA_free(objid);
  retval = PortableServer_POA_servant_to_reference(poa, newservant, ev);
  return retval;
}
```

messageBox-skelimpl.c

Destruction de l'objet serveur

```
static void
impl_HELLOWORLD_MessageBox__destroy( impl_POA_HELLOWORLD_MessageBox *servant,
                                     CORBA_Environment *ev)
{
  PortableServer_ObjectId *objid;

  objid = PortableServer_POA_servant_to_id(servant->poa, servant, ev);
  PortableServer_POA_deactivate_object(servant->poa, objid, ev);
  CORBA_free(objid);

  POA_HELLOWORLD_MessageBox__fini((PortableServer_Servant)servant, ev);
            /****************** inserted code       ****************/
  gtk_msgbox_free(servant->gtk_msg_box );
            /****************** end of inserted code ***************/
  g_free(servant);
}
```

messageBox-skelimpl.c

Implémentation de(s) méthode(s) d'objet serveur

```
static void
impl_HELLOWORLD_MessageBox_addMessage( impl_POA_HELLOWORLD_MessageBox *servant,
                                       CORBA_char * text,
                                       CORBA_Environment *ev )
{
             /****************** inserted code       ****************/
  GtkMsgBox* gtk_msg_box;

  gtk_msg_box = servant->gtk_msg_box;
  gtk_msgbox_add(gtk_msg_box, text);
           /****************** end of inserted code ***************/
}
```

msgServeur.c

Implémentation du serveur d'objets: connexion sur l'ORB, activation du POA

```
#include "gtkMsgBox.h"
#include "messageBox.h"
#include "messageBox-skelimpl.c"

int
main(int argc, char *argv[]) {
  PortableServer_POA poa;
  CORBA_Environment ev;
  CORBA_ORB orb;
  HELLOWORLD_MessageBox ref_hello;
  FILE *ior_file;
  gchar *set_ior;
  gchar  get_ior[1024];

  CORBA_exception_init(&ev);
  orb = (CORBA_ORB) gnome_CORBA_init("Message Box", "1.0",&argc, argv, 0, &ev);
  poa = (PortableServer_POA) CORBA_ORB_resolve_initial_references(orb," RootPOA",&ev);
  PortableServer_POAManager_activate( PortableServer_POA__get_the_POAManager(poa,&ev),
                                      &ev );
```

msgServeur.c

Implémentation du serveur d'objets: création d'objet, de référence d'objet et attente de clients

```
  ref_hello = impl_HELLOWORLD_MessageBox__create ( poa, &ev );
  if (!ref_hello) {
    return 1;
  }
  set_ior = CORBA_ORB_object_to_string(orb, ref_hello, &ev);
  ior_file = fopen("hello-ior.ref","w+");
  if (!ior_file) {
    exit(-1);
  }
  else fprintf(ior_file,  "%s", set_ior);
  fflush(ior_file);
  fclose(ior_file);
  CORBA_free(set_ior);
  /*   CORBA_ORB_run(orb, &ev); */
  gtk_main();
  return 0;
}
```

msgClient.c

Implémentation du client : connexion sur l'ORB

```
#include "gtkMsgBox.h"
#include "messageBox.h"
int
main(int argc, char *argv[]) {
  PortableServer_POA poa;
  CORBA_Environment ev;
  CORBA_ORB orb;
  HELLOWORLD_MessageBox ref_hello;
  FILE *ior_file;
  gchar *set_ior;
  gchar  get_ior[1024];

  CORBA_exception_init(&ev);
  orb = (CORBA_ORB) gnome_CORBA_init("Message Box", "1.0",&argc, argv, 0, &ev);
  /*
  poa = (PortableServer_POA) CORBA_ORB_resolve_initial_references(orb,"RootPOA",&ev);
  PortableServer_POAManager_activate( PortableServer_POA__get_the_POAManager(poa,&ev),
                                      &ev);
  */
```

msgClient.c

Implémentation du client: obtenir une référence d' objet

```
ior_file = fopen("hello-ior.ref","r");
if (!ior_file) {
  printf("Impossible d'ouvrir le fichier de l'IOR\n");
  exit(-1);
} else fscanf(ior_file,"%s\n", get_ior );
fclose(ior_file);
if (!get_ior) {
  printf("Impossible de trouver l'IOR\n");
  exit(-1);
}
ref_hello = CORBA_ORB_string_to_object(orb, get_ior, &ev);
if (!ref_hello) {
  printf("Impossible de joindre %s\n",  get_ior );
  return 1;
}
```

msgClient.c

Implémentation du client: appel de la méthode distante

```
  HELLOWORLD_MessageBox_addMessage(ref_hello, argv[1], &ev );
  if (ev._major != CORBA_NO_EXCEPTION) {
    printf("exception %d recue de HELLOWORLD_MessageBox_addMessage\n", ev._major);
    return 1;
  }
  CORBA_Object_release(ref_hello, &ev);
  if (ev._major != CORBA_NO_EXCEPTION) {
    printf("exception %d recue de CORBA_Object_release \n", ev._major);
    return 1;
  }
  CORBA_Object_release( (CORBA_Object) orb, &ev);
  /*  gtk_main(); */
  return 0;
}
```

`makefile`

pour obtenir l'application **`msgCient`** sur l'application **`msgServeur`**

```
C_FLAGS= -c ‘gnome-config --cflags gnomeui gnorba‘
LD_FLAGS=‘gnome-config --libs gnomeui gnorba‘
CCC=gcc
LD=gcc
all : msgServeur msgClient
idl :
        orbit-idl --skeleton-impl messageBox.idl
msgServeur : messageBox-skelimpl.o messageBox-common.o messageBox-skels.o \
                gtkMsgBox.o msgServeur.o
        $(LD) -o msgServeur messageBox-skelimpl.o messageBox-common.o messageBox-skels.o \
                gtkMsgBox.o msgServeur.o  $(LD_FLAGS)
msgClient : messageBox-common.o messageBox-stubs.o gtkMsgBox.o msgClient.o
        $(LD) -o msgClient messageBox-common.o messageBox-stubs.o \
                gtkMsgBox.o msgClient.o $(LD_FLAGS)
.c.o :
        $(CCC) $(C_FLAGS) $<
clean :
        rm -f  *~ *.o  core a.out msgServeur msgClient
```

"Hello World" Distribué

Lancement du serveur

```
{logname@hostname} msgServeur
```

Lancements du client

```
{logname@hostname}  msgClient hello &
{logname@hostname}  msgClient world &
{logname@hostname}  msgClient distribue
```

Résultat sur la partie serveur

Interoperable Object Reference

IOR: identification d'objects par une référence unique sur:

- ▷ nom de l'objet, nom de classe
- ▷ adresse ou se trouve l'objet, port de communication
- ▷ ...

IOR est une représentation binaire

- ▷ de la référence d'un objet
- ▷ interprétable par tous les ORB

L'OMG spécifie le contenu des IOR et leur méthodes de conversion

```
// C++
char*             object_to_string( CORBA::Object_ptr obj);
CORBA::Object_ptr string_to_object( char*              str);
// Java
public String object_to_string( Object obj);
public Object string_to_object( String str);
```

Interoperable Object Reference

Scénario classique de connexion client/serveur par IOR

- ▷ création de référence d'objet coté serveur
- ▷ stockage de l'IOR dans un fichier
- ▷ mis à disposition sur un serveur de fichiers
- ▷ accès client et ouverture du fichier
- ▷ conversion IOR en référence d'objet générique
- ▷ conversion vers le type d'objet voulu

Contenu d'une IOR

```
IOR:010000001b00000049444c3a48454c4c4f574f524c442f4d6573736167653a312e300000020000
0000caaedfba44000000010100002b0000002f746d702f6f726269742d6e6564656c65632f6f72622d
373731333038353838313736323239303039300000000000000800000000000000010000000000000
240000000101000000d0000006c696e312e656e69622e667200644c060800000000000000001000000
```

Broker ORBit

Utilisation du broker de GNOME pour la création d'IOR coté serveur

```
#include <orb/orbit.h>
int main (int argc, char **argv) {
  PortableServer_POA poa;
  CORBA_Environment ev;
  CORBA_ORB orb;
  HELLOWORLD_MessageBox ref_hello;
  FILE *ior_file;
  gchar *set_ior;
  gchar  get_ior[1024];

  CORBA_exception_init(&ev);
  orb = (CORBA_ORB) gnome_CORBA_init("Message Box", "1.0",&argc, argv, 0, &ev);
  poa = (PortableServer_POA) CORBA_ORB_resolve_initial_references(orb," RootPOA",&ev);
  PortableServer_POAManager_activate(
    PortableServer_POA__get_the_POAManager(poa,&ev), &ev );
  ...
```

Broker ORBit

```
  ...
  ref_hello = impl_HELLOWORLD_MessageBox__create ( poa, &ev );
  ...
  set_ior = CORBA_ORB_object_to_string(orb, ref_hello, &ev);
  ior_file = fopen("hello-ior.ref","w+");
  ...
  fprintf(ior_file,  "%s", set_ior);
  ...
  CORBA_free(set_ior);
  ...
  CORBA_ORB_run(orb, &ev);
  return 0;
}
```

▷ création d'objet, conversion en IOR, sauvegarde dans un fichier

Broker ORBit

Exemple de conversion d'IOR coté client

```
int main(int argc, char *argv[]) {
  PortableServer_POA poa;
  CORBA_Environment ev;
  CORBA_ORB orb;
  HELLOWORLD_MessageBox ref_hello;
  FILE *ior_file;
  gchar *set_ior;
  gchar  get_ior[1024];

  ior_file = fopen("hello-ior.ref","r");
  fscanf(ior_file,"%s\n", get_ior );
  fclose(ior_file);
  ...
  ref_hello = CORBA_ORB_string_to_object(orb, get_ior, &ev);
  HELLOWORLD_MessageBox_addMessage(ref_hello, argv[1], &ev );
  if (ev._major != CORBA_NO_EXCEPTION) { ... }
  ...
  return 0;
}
```

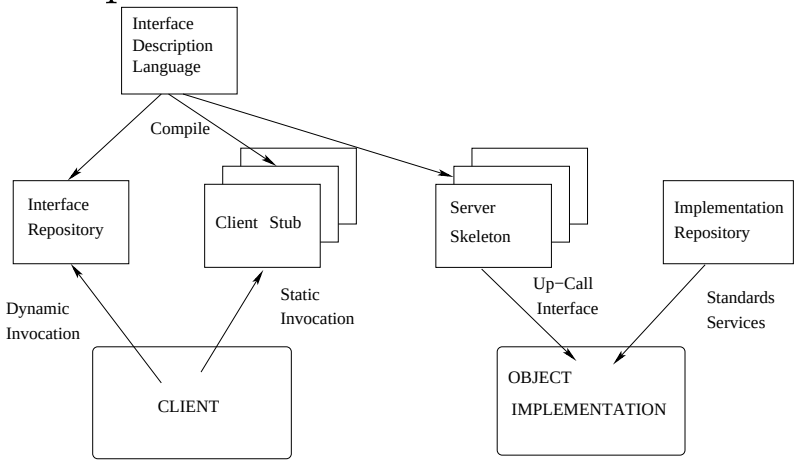
Référentiel d'Interfaces

Interface **R**epository (IR)

Entrepôt, Référentiel, Dictionnaire ou encore Dépôt d'Interfaces

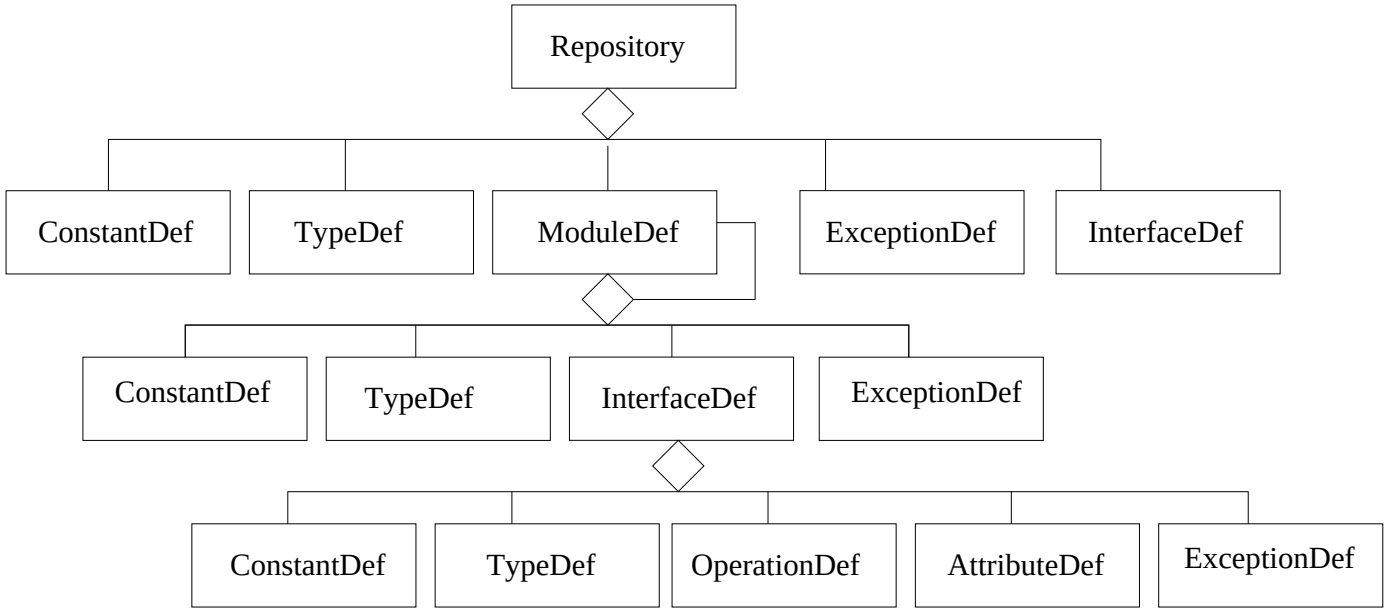
Base de Données de tous les IDL du client

- ▷ contenu généré par la compilation des IDL
- ▷ permet de retrouver tout type d'objet accessible
- ▷ et d'invoquer dynamiquement des méthodes



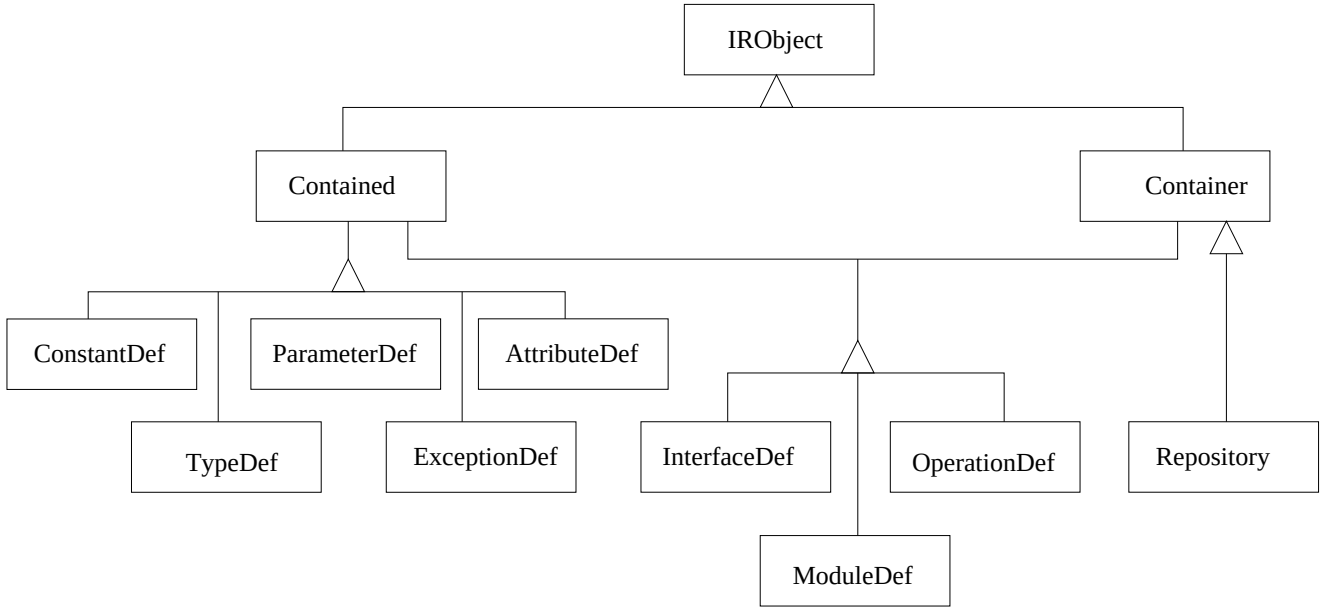
Référentiel d'Interfaces

Le Référentiel d'Interfaces est un ensemble d'objets CORBA



Référentiel d'Interfaces

La norme CORBA2.0 intoduit une hiérarchie d'inclusion des définitions



Référentiel d'Interfaces

Neuf opérations seront suffisantes pour la navigation au sein du référentiel

- ▷ Cinq d'entre sont dérivées des classes de base
  - ◇ `Container`
  - ◇ `Contained`
- ▷ les quatre autres sont spécifiques des IDL
  - ◇ `Repository`
  - ◇ `InterfaceDef`

Référentiel d'Interfaces

IDL `Contained`

- ▷ `describe()` : retourne une `Description` de l'IDL de l'objet

IDL `Container`

- ▷ `lookup()` : retourne une suite de pointeurs sur les objets qu'il contient
- ▷ `lookup_name()`: localisation d'un objet de container par son nom
- ▷ `contents()` : retourne la liste d'objets (directement contenus ou hérités)
- ▷ `describe_contents()` : description des objets qu'il contient (combinaison `describe() et contents()`)

Référentiel d'Interfaces

IDL `InterfaceDef`

- ▷ `describe_interface()` : retourne une description complète de l'IDL
  - ◇ nom, ID de référentiel, numéro de version
  - ◇ opérations, attributs
  - ◇ interfaces parents
- ▷ `is_a()` : retour vrai si du même type ou héritage du type

IDL `Repository`

- ▷ `lookup_id()` : récupération d'un ID d'objet du référentiel
- ▷ `get_primitive()` : référence sur un objet de type non-modifiable

Référentiel d'Interfaces

Définition IDL des interfaces de l'IR avec JacORB (`ir.idl`)

```
module org {
  module omg {
    module CORBA {
      ......
      enum DefinitionKind {
        dk_none, dk_all,
        dk_Attribute, dk_Constant, dk_Exception, dk_Interface,
        dk_Module, dk_Operation, dk_Typedef,
        dk_Alias, dk_Struct, dk_Union, dk_Enum,
        dk_Primitive, dk_String, dk_Sequence, dk_Array,
        dk_Repository
      };
      interface IRObject {
        readonly attribute DefinitionKind def_kind;
        void destroy ();
      };
      ......
```

Référentiel d'Interfaces

```
        ......
        interface Contained : IRObject {
          struct Description {
            DefinitionKind kind;
            any value;
          };
        Description describe ();
        void move ();
        };
        interface Container : IRObject {
        ......
        Contained lookup();
        ContainedSeq contents();
        ContainedSeq lookup_name ();
        struct Description {
          Contained contained_object;
          DefinitionKind kind;
          any value;
        };
        typedef sequence<Description> DescriptionSeq;
        DescriptionSeq describe_contents ();
        ModuleDef create_module ();
        ......
};
```

Service de Nommage

Service de base d'un broker CORBA (`CosNaming`)

- ▷ associer des noms logiques aux objets CORBA
- ▷ constituer un annuaire, catalogue d'objets distribués

Le Service de Noms est un graphe de noms

- ▷ alimenté par les serveurs
- ▷ parcourus par les clients

Les instances d'objets sont répertoriées dans le service

- ▷ par le stockage de références d'objets

Service de Nommage

Concepts du Service de Nommage

- ▷ espaces de noms (namespaces): racine du graphe
- ▷ contexte de nommage (`NamingContext`): noeud du graphe
- ▷ les noms proprement dit (`Name`): constituants d'un noeud
- ▷ les composants de noms (`NameComponent`): constituants d'un nom

Module CORBA du service de Nommage:

```
module CosNaming {
   typedef string Istring;
   struct NameComponent {
     Istring id;
     Istring kind;
   };
   typedef sequence<NameComponent> Name;
...
```

Service de Nommage

Utilité du Service de Nommage:

- ▷ association nom-objet (`bind()`)
- ▷ valable dans un contexte (naming context)
- ▷ où tous les noms (names) sont différents
- ▷ et récupérer l'objet (référence) par son nom (`resolve()`)

Conversion de noms en chaîne de caractères et réciproquement

- ▷ "/": séparation entre composants-noms: (CORBA/ORB/)
- ▷ ".": séparation dans un composant nom, `id.kind`, (`orb.idl`)

Service de Nommage

Problème du contexte de nommage

▷ associer un nom à un contexte de nommage ou à un objet (`bind()`)

▷ retrouver un contexte de nommage ou un objet par un nom (`resolve()`)

```
interface NamingContext {
  void bind(in Name n, in Object obj)
           raises(NotFound, CannotProceed, InvalidName, AlreadyBound);
  void rebind(in Name n, in Object obj)
           raises(NotFound, CannotProceed, InvalidName);
  void bind_context(in Name n, in NamingContext nc)
          raises(NotFound, CannotProceed, InvalidName, AlreadyBound);
  void rebind_context(in Name n, in NamingContext nc)
          raises(NotFound, CannotProceed, InvalidName);
  Object resolve(in Name n) raises(NotFound, CannotProceed, InvalidName);
  void unbind(in Name n) raises(NotFound, CannotProceed, InvalidName);
  NamingContext new_context();
  NamingContext bind_new_context(in Name n)
          raises(NotFound, AlreadyBound, CannotProceed, InvalidName);
  void destroy() raises(NotEmpty);
};
```

Broker ORBit

Exemple de nommage d'objet coté serveur

```
main () {
  CORBA_Object orb;
  CORBA_Environment ev;
  CosNaming_NamingContext root;
  CosNaming_NameComponent name_component[3] = { {"GNOME", "subcontext"},
                                                {"Servers", "subcontext"}
                                                {"gnome_panel", "server"} };
  CosNaming_Name name = {3, 3, name_component, CORBA_FALSE};
  CORBA_Object panel;
  CORBA_exception_init (&ev);
  CORBA_ORB_init (&ev);
  root = CORBA_ORB_resolve_initial_service (orb, "NameService", &ev);
  if (ev->_major != CORBA_NO_EXCEPTION) {
    fprintf ( stderr, "Error: could not get name service: %s\n",
              CORBA_exception_id(&ev) );
    exit (1);
  }
```

Broker ORBit

```
/*
   POA: insertion code pour enregistre rle serveur sur l'ORB
   initialiser une reference d'objet (panel)
   ...
*/

   CosNaming_NamingContext_bind (root, &name, panel, &ev);
   if (ev->_major != CORBA_NO_EXCEPTION) {
       fprintf ( stderr, "Error: could register object: %s\n",
                 CORBA_exception_id(&ev) );
       exit (1);
   }
   CORBA_exception_free (&ev);
   return 0;
 }
```

Broker ORBit

Récupération de la référence d'objet coté client

```
#include <corba.h>

  main (int argc, char *argv) {
   CORBA_Object orb;
   CORBA_Environment ev;
   CosNaming_NamingContext *root;
   CosNaming_NameComponent name_component[3] = { {"GNOME", "subcontext"},
                                                 {"Servers", "subcontext"}
                                                 {"gnome_panel", "server"} };
   CosNaming_Name name = {3, 3, name_component, CORBA_FALSE};
   CORBA_Object panel;
   CORBA_exception_init (&ev);
   CORBA_ORB_init (&ev);
   root = CORBA_ORB_resolve_initial_service (orb, "NameService", &ev);
   if (ev->_major != CORBA_NO_EXCEPTION) {
     fprintf( stderr, "Error: could not get name service: %s\n",
              CORBA_exception_id(&ev) );
     exit (1);
   }
```

Broker ORBit

```
   panel = CosNaming_NamingContext_resolve (root, &name, &ev);
   if (ev->_major != CORBA_NO_EXCEPTION) {
       fprintf (stderr,
                "Error: could resolve object: %s\n",
                CORBA_exception_id(&ev));
       exit (1);
   }

/*
   invocation de methodes distantes du CORBA_Object panel
   ...
*/
   CORBA_free (root);
   CORBA_exception_free (&ev);
   return 0;
 }
```

Service de Nommage

`CosNaming`: module IDL du Service de Nommage de CORBA

- ▷ serveur DNS ≡ serveur de noms CORBA
- ▷ adresse IP ≡ référence d'objet
- ▷ le serveur enregistre le nom
- ▷ les clients demandent au serveur de noms
  - ◇ si l'objet est enregistré
  - ◇ récupèrent une référence sur l'objet
- ▷ les clients envoient des requêtes sur l'objet serveur

Service de Nommage

Définitions du service de Nommage

- ▷ les noms sont associés (binding) à une référence d'objet
- ▷ dans un contexte de nommage (NamingContext)
- ▷ résoudre (resolving) un nom revient à
- ▷ trouver sa référence associée dans le contexte de nommage
- ▷ contextes (objets CORBA) utilisés pour créer des graphes de noms
- ▷ chaque nœud est un contexte, chaque feuille est un objet
- ▷ chaque nom d'objet est composé des noms de nœuds et de feuille
- ▷ chaque nom représente un parcours complet dans l'arbre

Module CosNaming

```
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;
   };
```

IDL NamingContext

```
interface NamingContext {
  void bind(in Name n, in Object obj)
           raises(NotFound, CannotProceed, InvalidName, AlreadyBound);
  void rebind(in Name n, in Object obj)
           raises(NotFound, CannotProceed, InvalidName);
  void bind_context(in Name n, in NamingContext nc)
          raises(NotFound, CannotProceed, InvalidName, AlreadyBound);
  void rebind_context(in Name n, in NamingContext nc)
          raises(NotFound, CannotProceed, InvalidName);
  Object resolve(in Name n)
          raises(NotFound, CannotProceed, InvalidName);
  void unbind(in Name n)
          raises(NotFound, CannotProceed, InvalidName);
  NamingContext new_context();
  NamingContext bind_new_context(in Name n)
          raises(NotFound, AlreadyBound, CannotProceed, InvalidName);
  void destroy()
          raises(NotEmpty);
  };
};
```

IDL NamingContext

Fonctionnalités du service de Nommage ORBit

- ▷ gestion des feuilles de l'arbre
  - ◇ `bind(), rebind(),unbind()`
- ▷ gestion des nœuds de l'arbre
  - ◇ `bind_context(), rebind_context(),`
  - ◇ `new_context(), bind_new_context()`

Coté Serveur

Exemple de nommage d'objet sur le serveur

```
main () {
  CORBA_Object orb;
  CORBA_Environment ev;
  CosNaming_NamingContext root;
  CosNaming_NameComponent name_component[3] = { {"GNOME", "subcontext"},
                                                {"Servers", "subcontext"}
                                                {"gnome_panel", "server"} };
  CosNaming_Name name = {3, 3, name_component, CORBA_FALSE};
  CORBA_Object panel;
  CORBA_exception_init (&ev);
  CORBA_ORB_init (&ev);
  root = CORBA_ORB_resolve_initial_service (orb, "NameService", &ev);
  if (ev->_major != CORBA_NO_EXCEPTION) {
    fprintf ( stderr, "Error: could not get name service: %s\n",
              CORBA_exception_id(&ev) );
    exit (1);
  }
```

Coté Serveur

```
  /* code to register the server against the ORB
   * create valid object reference stored in panel.
   * this is for the POA
   */
  CosNaming_NamingContext_bind (root, &name, panel, &ev);
  if (ev->_major != CORBA_NO_EXCEPTION) {
      fprintf ( stderr, "Error: could register object: %s\n",
                CORBA_exception_id(&ev) );
      exit (1);
  }
  CORBA_exception_free (&ev);
  return 0;
}
```

Coté Client

Récupération de la référence d'objet sur le client

```
#include <corba.h>

  main (int argc, char *argv) {
   CORBA_Object orb;
   CORBA_Environment ev;
   CosNaming_NamingContext *root;
   CosNaming_NameComponent name_component[3] = { {"GNOME", "subcontext"},
                                                 {"Servers", "subcontext"}
                                                 {"gnome_panel", "server"} };
   CosNaming_Name name = {3, 3, name_component, CORBA_FALSE};
   CORBA_Object panel;
   CORBA_exception_init (&ev);
   CORBA_ORB_init (&ev);
   root = CORBA_ORB_resolve_initial_service (orb, "NameService", &ev);
   if (ev->_major != CORBA_NO_EXCEPTION) {
     fprintf( stderr, "Error: could not get name service: %s\n",
              CORBA_exception_id(&ev) );
     exit (1);
   }
```

Coté Client

```
  panel = CosNaming_NamingContext_resolve (root, &name, &ev);
  if (ev->_major != CORBA_NO_EXCEPTION) {
      fprintf (stderr,
               "Error: could resolve object: %s\n",
               CORBA_exception_id(&ev));
      exit (1);
  }

  /* there, the panel variable holds a CORBA_Object to the gnome panel
   * it is now possible to call some of the panel methods defined in
   * gnome-core/idl/gnome-panel.idl
   */

  CORBA_free (root);
  CORBA_exception_free (&ev);

  return 0;
}
```

Coté Client

Le client n'a pas à connaître nécessairement l'objet serveur, il peut

▷ récupérer la liste des services sur l'ORB: `list_initial_services()`

▷ puis une référence d'objet: `resolve_initial_reference ()`

Opérations définis dans le module IDL de l'ORB

```
module CORBA {
   interface ORB {
     typedef string ObjectId;
     typedef sequence <ObjectId> ObjectIdList;
     ObjectIdList list_initial_services ();

     exception InvalidName {};
     Object resolve_initial_reference (in ObjectId identifier)
            raises (InvalidName);
   };
};
```

Coté Client

```
#include <corba.h>

 int main (int argc, char **argv) {
   CORBA_ORB orb;
   CORBA_Environment ev;
   CORBA_sequence_ObjectId *seq;
   CORBA_unsigned_long i;

   CORBA_exception_init (&ev);
   orb = CORBA_ORB_init (&ev);
   seq = CORBA_ORB_list_initial_services (orb, &ev);
   for (i=0; i<seq->_length ;i++) {
     fprintf (stdout, "%s\n" ,(seq->_buffer)[i]);
   }

   return 0;
 }
```

P.S.: utiliser plutôt l'API GNOME: `gnome_get_name_service()`)

Service d'Evénements

Invocation classique de service par appel de méthodes distantes:

▷ du client vers le serveur

Nécessité de notification du serveur aux clients (communication asynchrone)

▷ messageries, alertes, cotations en direct,...

Deux sortes de possibilités de scénarios

▷ interrogation périodique du client vers le serveur (polling)

▷ insertion d'un objet serveur sur le client, appel de méthode sur le client

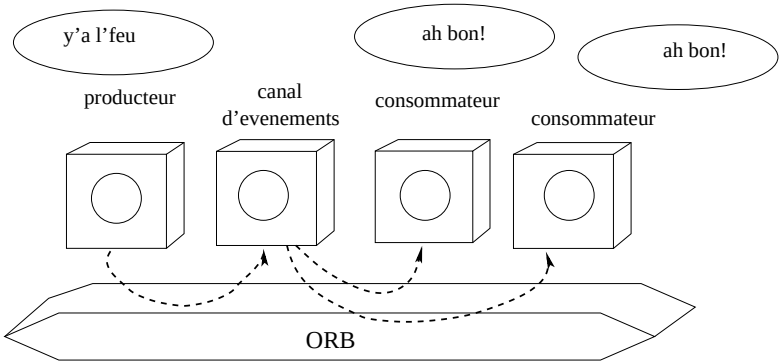
Nécessitent un fort couplage client/serveur, solution CORBA:

▷ service d'événements

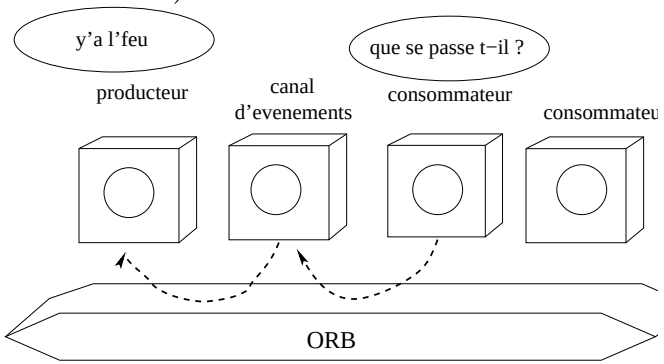
Service d'Evénements

Définition d'un module CORBA (`CosEvent`)

- ▷ consommateurs d'événements (consumer)
- ▷ fournisseurs d'événements (supplier)
- ▷ reliés par un canal d'événements (event channel)



MODELE DEPOT



MODELE RETRAIT

Service d'Evénements

Consommateurs et fournisseurs fonctionnent sur les modes
- ▷ dépot (push), retrait (pull) d'informations

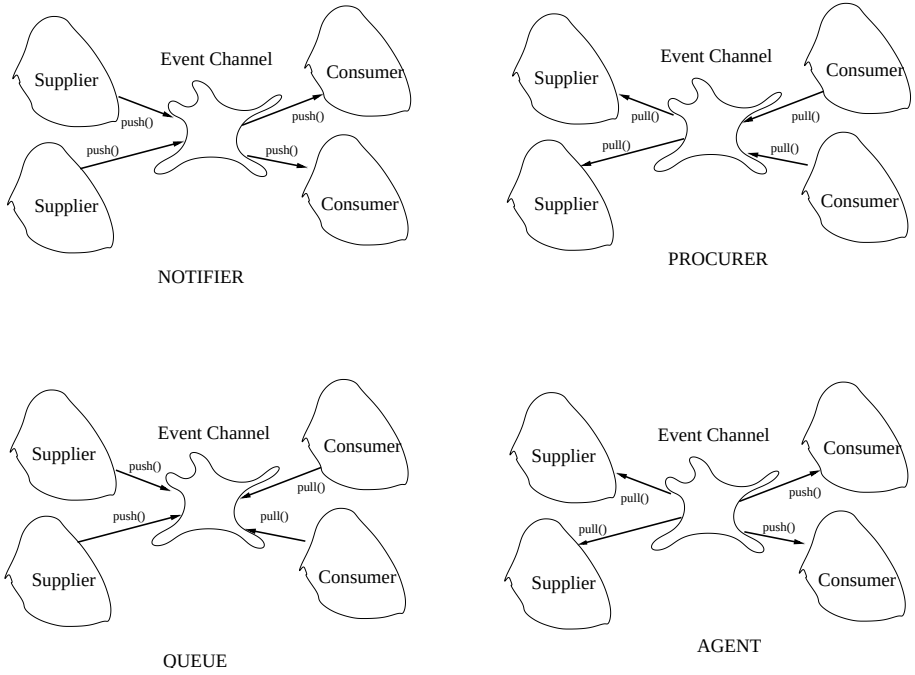
Définis par des interfaces IDL CORBA
- ▷ `PushSupplier`: contrôle l'émission de données sur le canal
- ▷ `PushConsumer`: les consommateurs sont avertis par le canal
- ▷ `PullSupplier`: le canal d'événement interroge les fournisseurs
- ▷ `PullConsumer`: les fournisseurs interrogent le canal d'événements

Le Canal d'événements est un médiateur entre fournisseurs et consommateurs
- ▷ `ProxyPullConsumer, ProxyPushConsumer`: auprès des fournisseurs
- ▷ `ProxyPushSupplier, ProxyPullSupplier`: auprès des consommateurs

Service d'Evénements

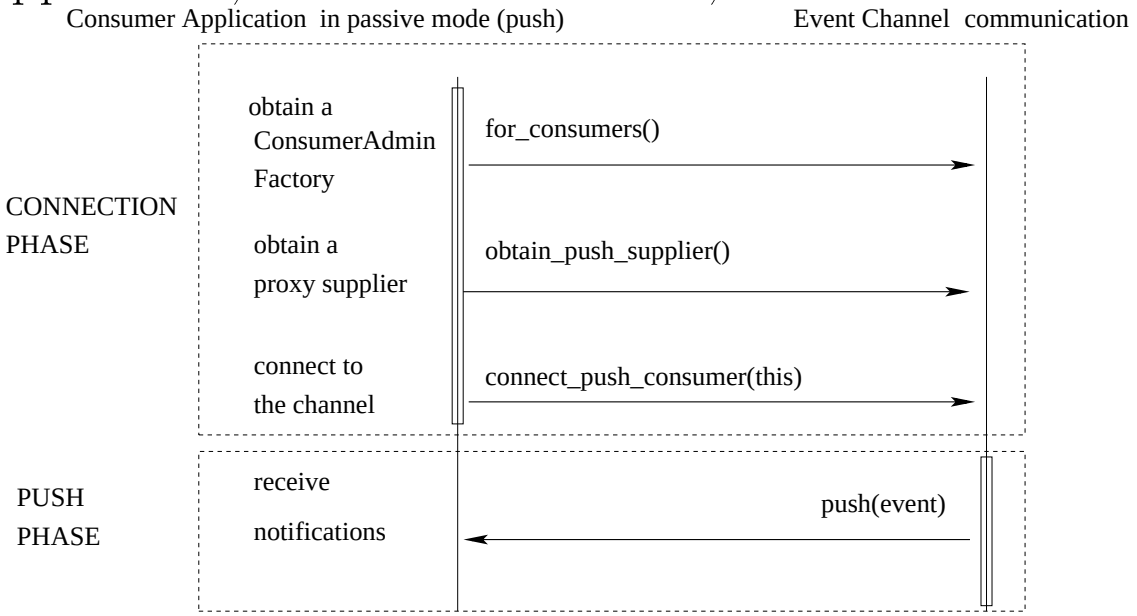
Communication asynchrone entre objets "producteur" et "consommateur"

- ▷ quatre modes de fonctionnement (fournisseur, demandeur, file, agent)
- ▷ du canal d'événement (non-typé, typé)



Deuxième Programme

Scénarion d'application, coté consommateurs, en mode Push



▷ obtenir une référence sur un objet `ConsumerAdmin`

▷ choisir un fournisseur proxy en mode actif ou passif `Proxy...Supplier`

▷ s'enregistrer, via ce proxy, sur le canal `connect_..._consumer(this)`

Modules IDL

Interfaces pour les modules du service d'événements

- ▷ `CosEventComm` : communications consommateurs et producteurs
  - ◇ `PushSupplier` : producteur actif
  - ◇ `PullSupplier` : producteur passif
  - ◇ `PushConsumer` : consommateur passif
  - ◇ `PullConsumer` : consommateur actif
- ▷ `CosEventChannelAdmin` : connexions sur le canal d'événements
  - ◇ `ConsumerAdmin` : gestion des consommateurs
  - ◇ `SupplierAdmin` : gestion des producteurs
  - ◇ `EventChannel` : gestion du canal d'événements
  - ◇ `ProxyPushSupplier, ...` : représentants sur le canal d'vénements

Module de Communication

Interfaces IDL du module de communication

```
module CosEventComm {
  exception Disconnected{};
  interface PushConsumer {
    void push (in any data) raises(Disconnected);
    void disconnect_push_consumer();
  };
  interface PushSupplier {
    void disconnect_push_supplier();
  };
  interface PullSupplier {
    any pull () raises(Disconnected);
    any try_pull (out boolean has_event) raises(Disconnected);
    void disconnect_pull_supplier();
  };
  interface PullConsumer {
    void disconnect_pull_consumer();
  };
};
```

Module de Connexion

Interfaces IDL du module de connexion sur le canal d'événements

```
module CosEventChannelAdmin {
  exception AlreadyConnected {};
  exception TypeError {};
...
  interface ProxyPushSupplier: CosEventComm::PushSupplier {
    void connect_push_consumer( in CosEventComm::PushConsumer push_consumer)
                                                              raises(AlreadyConnected, TypeError);
  };
...
  interface ConsumerAdmin {
    ProxyPushSupplier obtain_push_supplier();
    ProxyPullSupplier obtain_pull_supplier();
  };
  interface SupplierAdmin {
     ProxyPushConsumer obtain_push_consumer();
     ProxyPullConsumer obtain_pull_consumer();
  };
  interface EventChannel {
     ConsumerAdmin for_consumers();
     SupplierAdmin for_suppliers();
     void destroy();
  };
};
```

Deuxième Programme

Implémentation du Consommateur Passif

```
public class PushConsumerImpl extends Thread implements PushConsumer {
...
  static public void main  (String argv[]) {
    EventChannel         ecs = null;
    ConsumerAdmin        ca;
    PushConsumerImpl pushConsumer = null;
    ProxyPushSupplier pps;
      try {
       ecs = EventChannelHelper.narrow( NameServer.locate(channelServer.CHANNELNAME));
      } catch (Exception e) {e.printStackTrace();}
      ca  = ecs.for_consumers();
      pps = ca.obtain_push_supplier();
      try {
       pushConsumer = new PushConsumerImpl( pps );
       pps.connect_push_consumer( (PushConsumer) pushConsumer );
     } catch (Exception e) { e.printStackTrace();}
     System.out.println("PushConsumerImpl registered.");
  }
}
```

Deuxième Programme

Implémentation du Producteur Actif

```
class PushSupplierImpl implements PushSupplier {
  public void disconnect_push_supplier() {};
  static public void main (String argv[]) {
    org.omg.CosEventChannelAdmin.EventChannel e = null;
    try {
     e = org.omg.CosEventChannelAdmin.EventChannelHelper.narrow(
                                              NameServer.locate(channelServer.CHANNELNAME)
                                                                );
    } catch (Exception ex) {ex.printStackTrace();}
    SupplierAdmin supplierAdmin = e.for_suppliers();
    ProxyPushConsumer proxyPushConsumer = supplierAdmin.obtain_push_consumer();
    Any event = new Any();
    int i=0;
    while (i<10){
      try {
        event.insert("Test the channel!" + i);
        System.out.println("Pushing event # " + (i++) );
        proxyPushConsumer.push( event );
     } catch (Disconnected d) {d.printStackTrace();}
   }
   proxyPushConsumer.disconnect_push_consumer();
  }
}
```

Deuxième Programme

Méthode `push()` du consommateur

```
...
  public void push(jacorb.Orb.Any data)
     throws org.omg.CosEventComm.Disconnected
  {
    System.out.println("received event " + (eventCounter++) + " : " + data.extract_string());
   if( eventCounter == 5 ) quit();
  }
....
```

Connexions sur le canal d'événements

```
public class channelServer {
  public static String CHANNELNAME = "theChannel";
    static public  void main( String[] argv ) {
      try {
        org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init();
        org.omg.CORBA.BOA boa = orb.BOA_init();
        org.omg.CORBA.Object o = boa.create( new EventChannelImpl(),"IDL:jacorb/Events/EventChannel:1.0");
        boa.named_obj_is_ready(o,CHANNELNAME);
        boa.impl_is_ready();
    } catch ( Exception e) {e.printStackTrace();}

  }
}
```

Conclusion

Faire communiquer des objets

- ▷ répartis sur plusieurs machines
- ▷ entre applications développées séparément
- ▷ sur des plateformes hétérogènes

Deux concurrents

- ▷ **CORBA** : **C**ommon **O**bject **R**equest **B**roker **A**rchitecture
- ▷ **DCOM** : **D**istributed **C**omponent **O**bject **M**odel

Conclusion

Les services communs d'objets CORBA (COSS)

- ▷ Nommage : obtenir des références d'objets
- ▷ Courtage : recherche de services
- ▷ Evénements : communication asynchrone (producteur/canal/consommateur)
- ▷ Cycle de vie : manipulation d'objets (création,destruction,copie,migration,...)
- ▷ relations : associations dynamiques de propriétés d'objets
- ▷ propriétés : associations dynamiques entre objets, graphe d'objets
- ▷ collections : gestion d'ensemble d'objets

Conclusion

Les services communs d'objets CORBA (COSS)

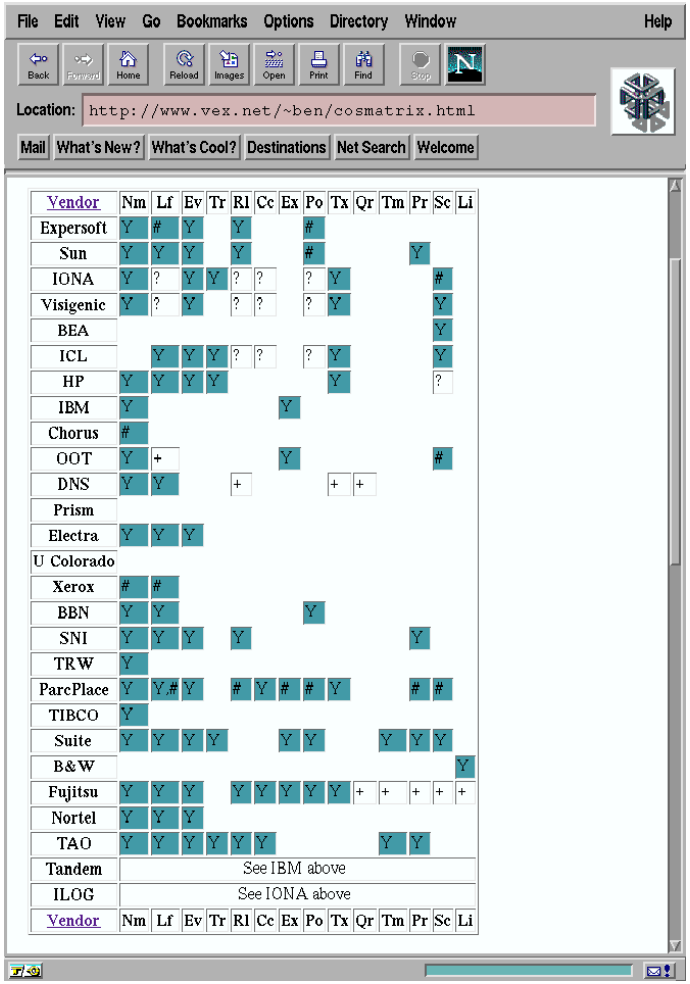
- ▷ persistance : sauvegarde de l'état d'objet (SGBD)
- ▷ interrogation : requêtes sur une base de données (OSQL)
- ▷ transaction : encapsulation des mécanismes existants
- ▷ concurrence : synchronisation d'accès
- ▷ sécurité : droits d'accès, encryptage,authentification, ...
- ▷ licence : contrôle d'utilisation d'objets, facturation, ...
- ▷ temps : synchronisation des horloges dans un système distribué

Conclusion

Les composantes du standard CORBA

- ▷ le bus Objet (ORB)
- ▷ langage de description (IDL)
- ▷ projection de l'IDL vers les langages d'implémentation (client,serveur)
- ▷ invocation statique
- ▷ invocation dynamique (DII, DSI)
- ▷ support d'adaptation pour l'objet serveur (BOA, POA,..)
- ▷ entrepôt d'interfaces et d'implementation (IR, DII)
- ▷ protocoles d'interopérabilité (GIOP, IIOP, ESIOP, ...)

Conclusion



Vendor	Nm	Lf	Ev	Tr	Rl	Cc	Ex	Po	Tx	Qr	Tm	Pr	Sc	Li
Expersoft	Y	#	Y		Y			#						
Sun	Y	Y	Y		Y			#				Y		
IONA	Y	?	Y	Y	?	?		?	Y				#	
Visigenic	Y	?	Y		?	?		?	Y				Y	
BEA													Y	
ICL		Y	Y	Y	?	?		?	Y				Y	
HP	Y	Y	Y	Y					Y				?	
IBM	Y						Y							
Chorus	#													
OOT	Y	+					Y						#	
DNS	Y	Y			+				+	+				
Prism														
Electra	Y	Y	Y											
U Colorado														
Xerox	#	#												
BBN	Y	Y						Y						
SNI	Y	Y	Y		Y							Y		
TRW	Y													
ParcPlace	Y	Y,#	Y		#	Y	#	#	Y			#	#	
TIBCO	Y													
Suite	Y	Y	Y	Y			Y	Y			Y	Y	Y	
B&W														Y
Fujitsu	Y	Y	Y		Y	Y	Y	Y	Y	+	+	+	+	+
Nortel	Y	Y	Y											
TAO	Y	Y	Y	Y	Y	Y					Y	Y		
Tandem	See IBM above													
ILOG	See IONA above													
Vendor	Nm	Lf	Ev	Tr	Rl	Cc	Ex	Po	Tx	Qr	Tm	Pr	Sc	Li

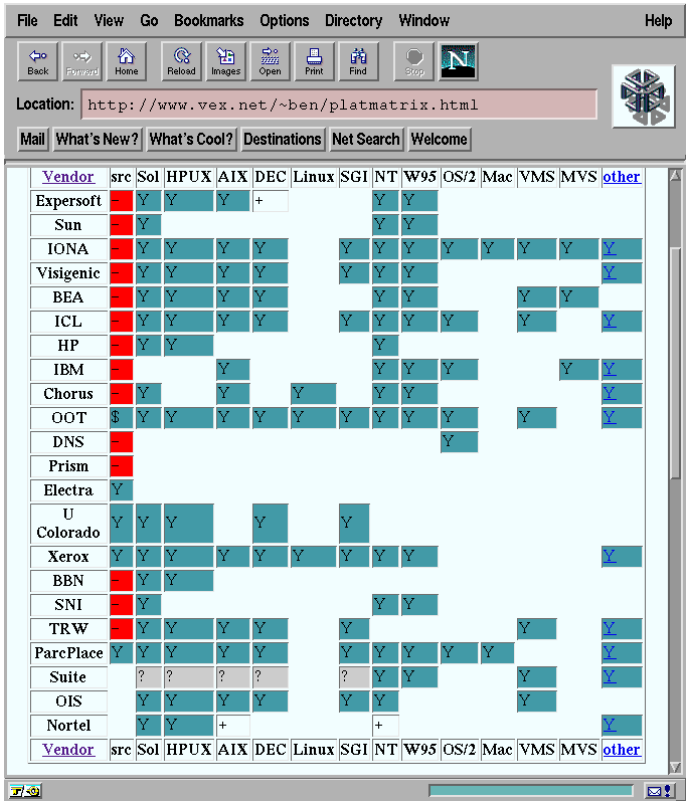
Conclusion

Les services COSS

- ▷ Nm : Naming, Tr : Trading, Lf : Lifecycle, Ev : Event,
- ▷ Rl : Relationships, Cc : Concurrency, Ex : Externalization,
- ▷ Po : Persistent Objects, Tx : Transactions, Qr : Query
- ▷ Cl : Collections, Tm : Time,Pr : Properties
- ▷ Sc : Security, Li : Licensing
- ▷ Cm : Configuration Management (not yet standardized)

Conclusion

Disponibles sur les plateformes



Vendor	src	Sol	HPUX	AIX	DEC	Linux	SGI	NT	W95	OS/2	Mac	VMS	MVS	other
Expersoft	-	Y	Y	Y	+			Y	Y					
Sun	-	Y						Y	Y					
IONA	-	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y
Visigenic	-	Y	Y	Y	Y		Y	Y	Y					Y
BEA	-	Y	Y	Y	Y			Y	Y			Y	Y	
ICL	-	Y	Y	Y	Y		Y	Y	Y	Y		Y		Y
HP	-	Y	Y					Y						
IBM	-			Y				Y	Y	Y			Y	Y
Chorus	-	Y		Y		Y		Y	Y					Y
OOT	$	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y		Y
DNS	-									Y				
Prism	-													
Electra	Y													
U Colorado	Y	Y	Y		Y		Y							
Xerox	Y	Y	Y	Y	Y	Y	Y	Y	Y					Y
BBN	-	Y	Y											
SNI	-	Y						Y	Y					
TRW	-	Y	Y	Y	Y		Y					Y		Y
ParcPlace	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y			Y
Suite		?	?	?	?		?	Y	Y			Y		Y
OIS		Y	Y	Y	Y		Y	Y				Y		
Nortel		Y	Y	+				+						Y
Vendor	src	Sol	HPUX	AIX	DEC	Linux	SGI	NT	W95	OS/2	Mac	VMS	MVS	other

CORBA

- David Acremann et. al. : "Développer avec CORBA en Java et C++" CampusPress `www.campuspress.net`
- Orfali, Harkey : "Client/Server Programming with Java & CORBA" Wiley Computer Publishing 1997
- Orfali, Harkey, Edwards : "Objets répartis : guide de survie" International Thomson Publishing 1996
- Chauvet: "Corba, ActiveX et Java Beans" Eyrolles 1997
- Mathieu Lacage & Dirk-Jan C. Binnema: "GNOME & CORBA" `http://developer.gnome.org/doc/guides/corba/html/book1.html`
- CHORUS : "COOL-ORB Tutorial " Chorus System 1997 racheté par SUN en 1998
- Linux Magazine (depuis sa création)

CORBA

Et les bonnes adresses

- ▷ Object Management Group
  `http://www.omg.org`
- ▷ Objets Distribués
  `http://www.DistributedObjects.com`
- ▷ sur les brokers du marché
  `http://www.infosys.tuwien.ac.at/Research/Corba/software.html`
  `http://patriot.net/ tvalesky/freecorba.html`
- ▷ le broker ORBit
  `http://www.labs.redhat.com/orbit`
- ▷ le broker JacORB
  `http://jacorb.inf.fu-berlin.de`