Common Object Request Broker Architecture

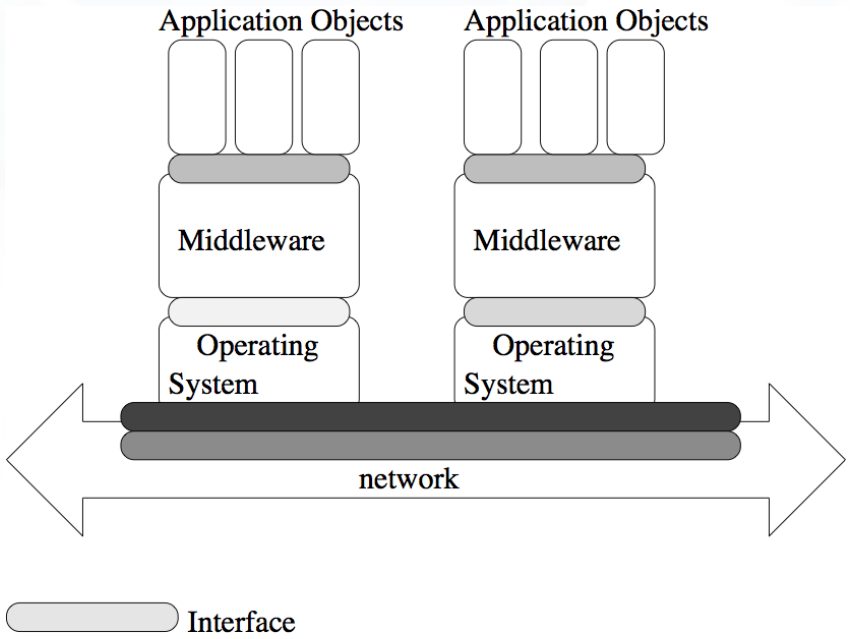
Ce support est très largement inspiré des cours de S. Chabridon et C. Taconet

Middleware

The middleware is a software layer providing :

- programming interfaces,
- data exchange format,
- a protocol for interoperability

to support distribution.



Distribution models

- Point to point message (Message Oriented Middleware - MOM)
- Point to multipoint message
- event/action
- client/server
- mobile code
- virtual shared memory

Client-server models

- Traditional (Remote Procedure Call - RPC)
- Object-oriented (Remote Method Invocation - RMI)
- Data-oriented (using SQL requests)
- web-based (using HTTP requests)

Middleware for distributed objects

Comes from two technologies :

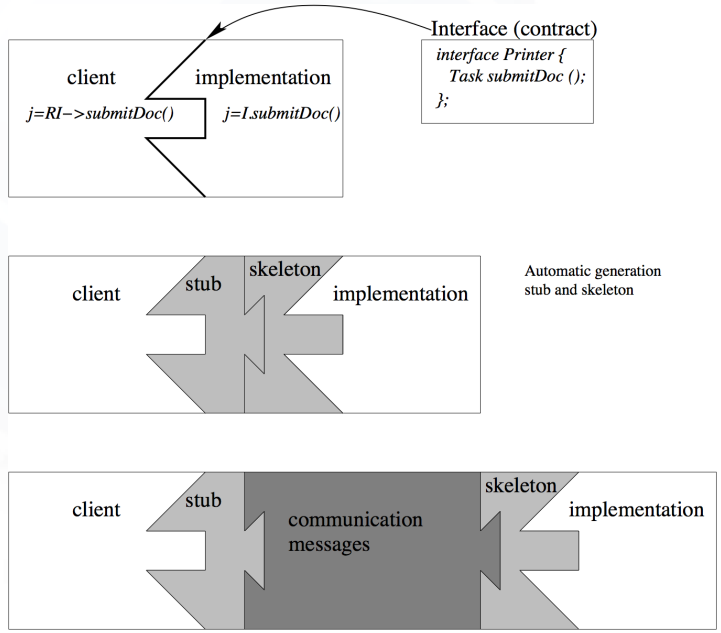
- objects (inheritance, encapsulation and polymorphism)
- RPC or Remote Procedure Call (distribution, heterogeneity, data marshalling and unmarshalling)

Goal : interoperability

- existing *"legacy code"*,
- numerous languages,
- several operating systems,
- various hardware,
- several network protocols

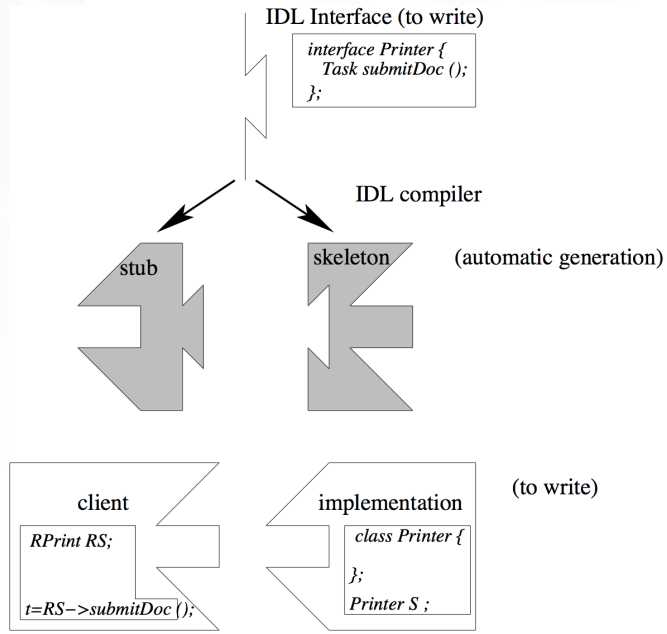
$\Rightarrow$ need for interoperability !

Principle of distributed objects

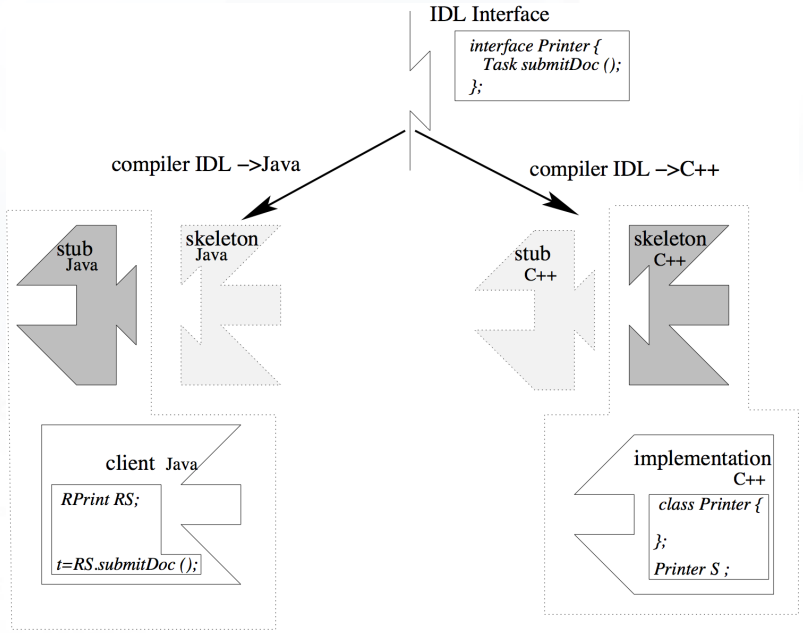


Development principles

1. description of the interface in **IDL**
2. IDL compiler creates the stub and the skeleton
3. the developer has to write both **client and server implementations**



Multi-languages (or multi-ORBs, or multi-OSs)



ORB - Object Request Broker

- assigns to each object an **object reference**,
- is in charge of object localization,
- automatic activation of server object,
- transports requests

Inherent complexity of distribution

- no global state,
- poor debugging tools,
- partial failures, network partition,
- requests in parallel (concurrency management)
- trusting the caller (authentication)

Main distributed object middleware

CORBA (OMG) 1991

Java RMI (Sun) 1997

DCOM - .Net (Microsoft) 1998

CORBA architecture

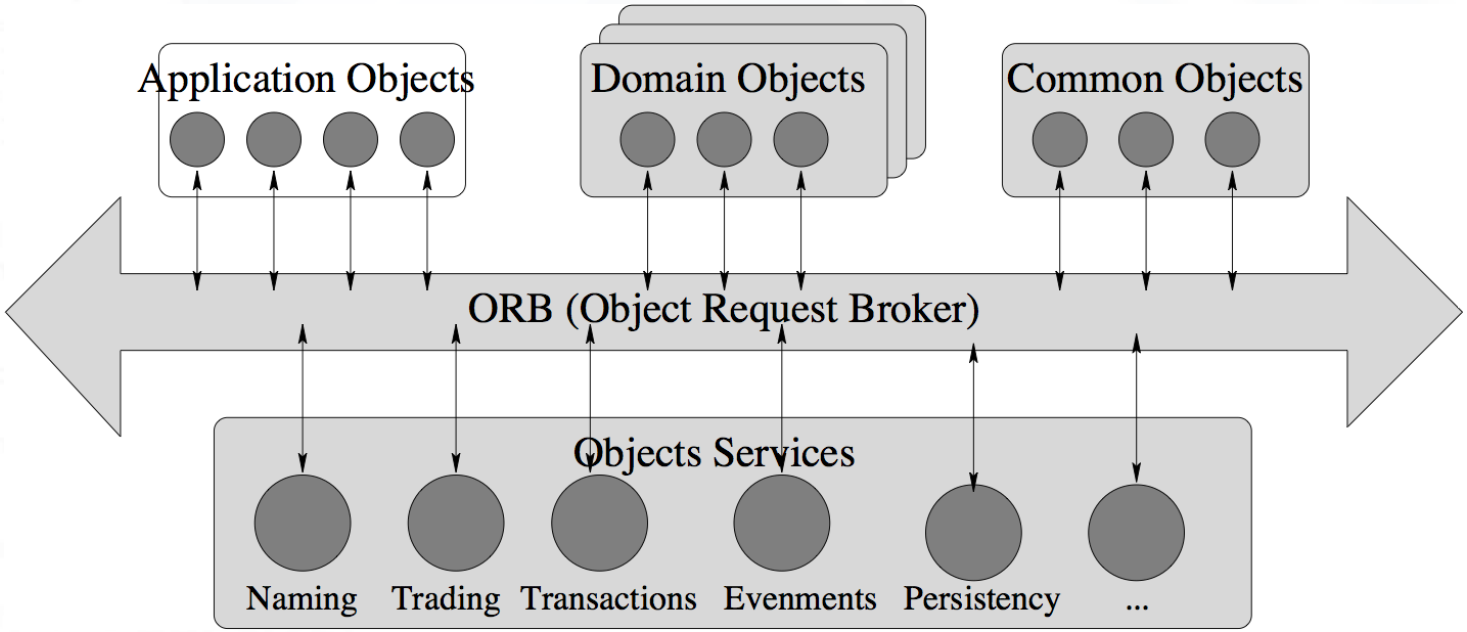
- OMG presentation - OMA architecture
- OMG specifications
- CORBA components
- Static and dynamic invocation
- Object adapter
- Persistent and transient objects
- Some ORBs

OMG (Object Management Group)

- international consortium founded in 1989 by 8 companies *(3Com Corporation, American Airlines, Canon Inc., Data General, Hewlett-Packard, Philips Telecommunications N.V., Sun Microsystems and Unisys Corporation)*
- in 1998 more than 800 members (computer manufacturers, software vendors, academics...),

Its role : promoting distributed objects technologies, and standardizing **specifications** (accepted by vote).
See http ://www.omg.org

OMA (Object Management Architecture)



The OMA components

ORB	requests transportation between objects
objects services	system services
application objects	specific to each application
domain objects	common for one application domain (telecommunication, medical, finance . . .)
common objects	common objects but not system (printer, mail, documents composition, network management)

OMG specifications (1)

Architecture

- OMA (Object Management Architecture) (1990),

Middleware

- CORBA (Common Object Request Broker Architecture) (1991 1.0 ; 1998 2.2),
- IIOP (Internet Inter-ORB Protocol) (1996),
- IDL (Interface Definition Language) and Language mappings
- CCM (CORBA Component Model)

OMG specifications (2)

Modelling

- UML (Unified Modeling Language)(1998)
- CWM (Common Warehouse Metamodel), MOF (Meta-Object Facility), XMI (XML Metadata Interchage) (1999)

Services

- Transaction, Event, Notification, Time, Property, Concurrency, Collection etc.

CORBA services

CORBA specifies IDL description for some fundamental services :

Collection Provides a uniform way to create and manipulate the most common collections generically

Concurrency Concurrent accesses management (locking mechanism...)

Enhanced view of time Generalized Clock interface to represent clocks with differing characteristics.

Event Event channels between producer and consumer objects

Externalization Protocols and conventions for externalizing (recording the object's state in a stream of data) and internalizing objects.

Licensing Interfaces that support management of software licenses.

LifeCycle Services and conventions for creating, deleting, copying and moving objects.

Naming Look-up an object using its name (white pages)

Notification Event service extension (event subscription, filtering, discovery, QoS,...)

CORBA services

Persistent State	Interfaces which present persistent information as storage objects stored in storage homes.
Property	Associate (attribute,value) pairs dynamically to CORBA objects
Query	Provides query operations on collections of objects.
Relation	Define relation objects ro represent dependency graphs among CORBA objects
Security	Authentication, access control and data encryption
Time	Global clock and object synchronisation
Trading Object	Look-up an object using its properties (yellow pages)
Transaction	Transactions involving distributed objects(ACID properties)

CORBA Object

- identified by an **object reference** (**IOR** Interoperable Object Reference),
- implements an IDL CORBA interface

Its implementation, location, incarnation . . . are transparent.

CORBA components

ORB core request transport protocol,

ORB-applications interface initialisation APIs,

object adapters on server side, ORB-implementation interface

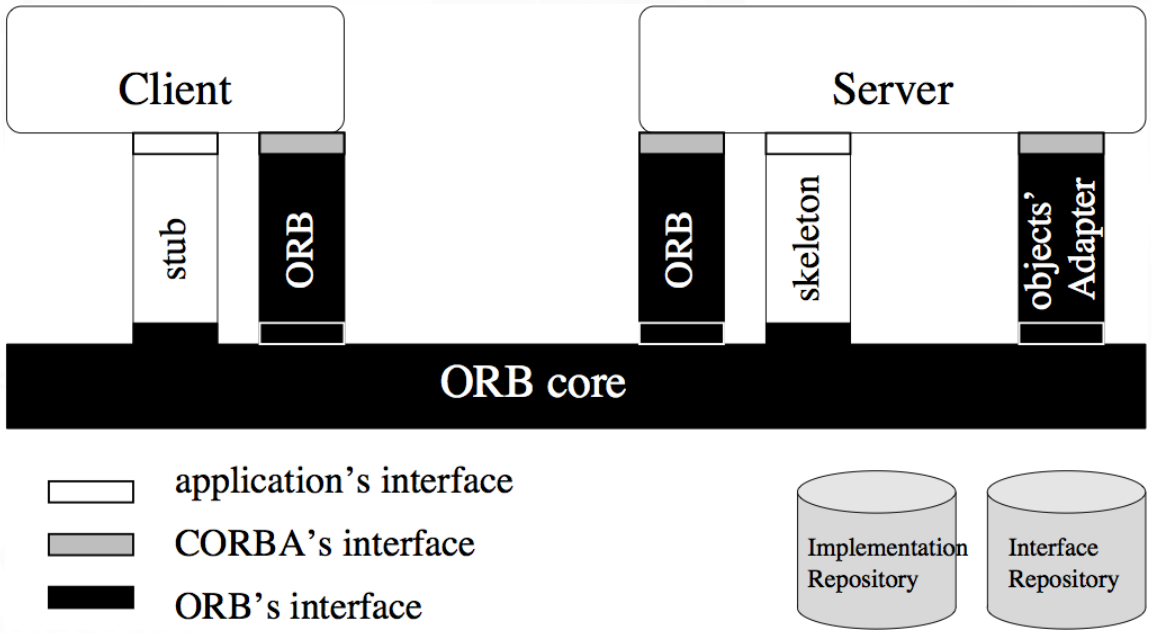
CORBA components

Each service may be registered in :

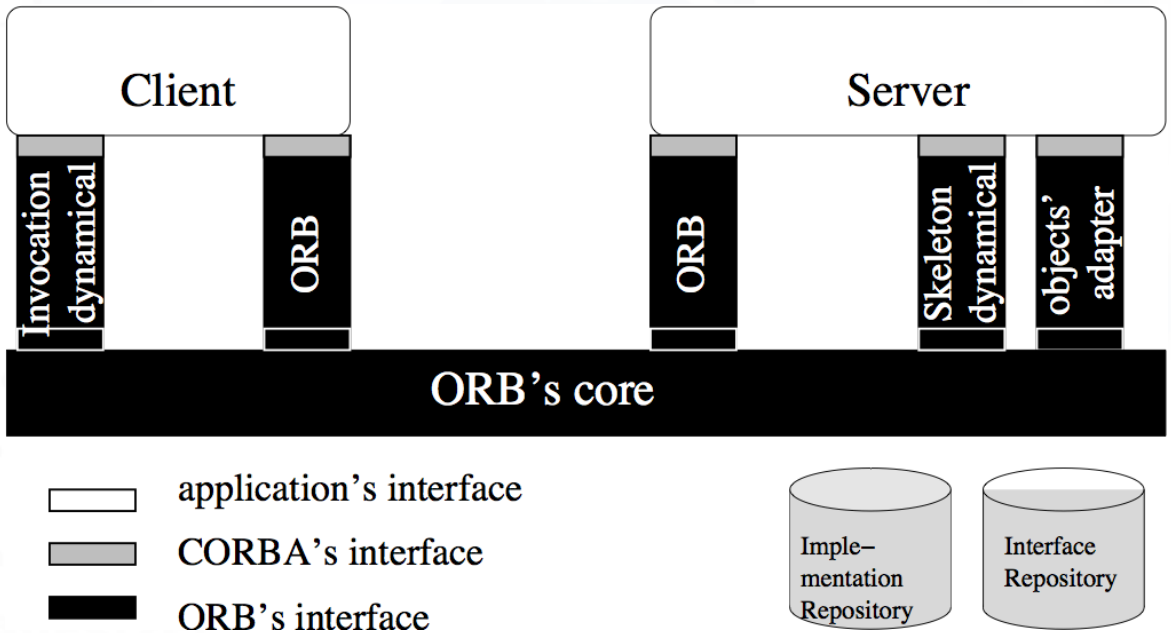
the implementation repository name of the server, path of the binary, activation mode ... (necessary for automatic activation)

the interface repository description of the object's interface (necessary for dynamic invocation)

CORBA components : static invocation



CORBA components : dynamic invocation



ORB interface

Every object may invoke methods on the ORB (pseudo CORBA object).
The ORB services are described in an IDL interface.
ORB services :

- initialisation,
- get the list of CORBA services supported by the ORB,
- store object references in strings,
- support dynamic invocations

Object Adapter

Other services are offered by object adapters.
An ORB (according to its implementation) may be bound to one of these object adapters :

- BOA (Basic Object Adapter) : deprecated,
- POA (Portable Object Adapter) specified in CORBA 2.2,
- OODA (Object Oriented Database Adapter).
- ...

Functions of an Object Adapter

- link between the ORB and servants
- activation and deactivation of servants (thanks to the implementation repository),
- creation and interpretation of object references,
- dispatch of invoked methods,
- client authentification.

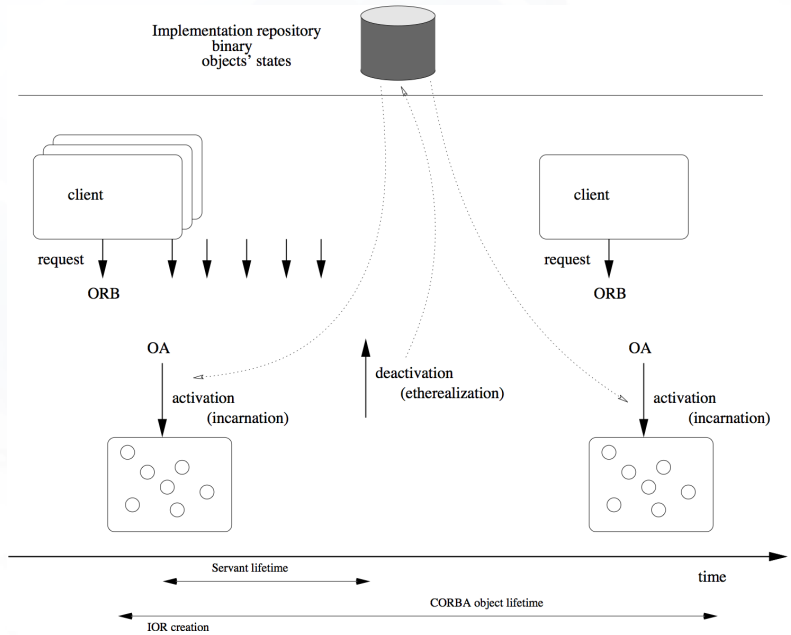
OAs depend on implementation mode and programming language.
An OA is a pseudo CORBA object.

Persistent and transient objects

transient object its lifetime is limited to the life of its server,

persistent object may be incarnated by several servers successively.
Its reference is made persistent by the ORB and remains the same.
If required, its state can be saved and restored by the application.
persistent reference $\neq$ persistent object state

CORBA object lifecycle with persistent reference



Some ORBs

commercial

Orbix	IONA
ORBacus	Object Oriented Concepts - IONA
VisiBroker	Inprise
DAIS	PeerLogic
ORB Plus	Hewlett Packard
M3	BEA

free for non commercial use

MICO	Frankfort University
TAO	Washington University
JDK	SUN
OmniORB	AT&T
Jonathan	ObjectWeb - French Consortium
ORBit	RHAD Labs

Préparation de l'environnement

- Récupérer le Live-CD debian-live-7.0.0-i386-rescue.iso et l'archive contenant le disque virtuel systdist.zip
- Créer une machine virtuelle Linux 2.6 équipée d'une mémoire de 512 Mo avec comme disque dur le contenu du systdist.zip et comme lecteur CD lecteur CD le fichier ISO de Debian Wheezy
- Configurer l'interface réseau en "Bridged Adapter" ou "Host-only Adapter"
- Démarrer la machine virtuelle
  - Choisir l'option "Live 686 (pae)"
- Définir un mot de passe pour l'utilisateur "user"

```
1 sudo passwd user
```

- Récupérer l'adresse IP de la machine virtuelle

```
1 sudo ifconfig eth0
```

- Revenir sur la machine et se connecter en SSH sur la machine virtuelle

```
1 ssh user@192.168.1.22
```

Configuration de l'environnement

- L'environnement de développement GNU, omniORB et Sun JDK sont utilisés
- Monter le disque virtuel sur le point de montage /opt

```
1 sudo mount -t ext4 UUID=7c7224bf-41d0-44ba-94aa-8bdfcf7a5047 /opt
```

- Lancer le script de configuration de l'environnement

```
1 source /opt/configure.sh
```

Application Echo

- L'application consiste en un objet hébergé sur le serveur qui renvoie la chaîne de caractères donnée en argument par le client
- Tous les mentionnés par la suite seront copiés sur la machine virtuelle avec un client SSH (scp, putty, ...)
- Fichier IDL echo.idl

```
1 // echo.idl
2 interface Echo {
3         string echoString(in string mesg);
4 };
```

- Génération des fichiers C++ (Stub, Skeleton, Entête, ...)

```
1 omniidl -bcxx echo.idl
```

1ère version : espace mémoire commun

- Le client et le serveur sont confondus, se trouvent sur le même espace mémoire et sont codés en C++
- Tous les mentionnés par la suite seront copiés sur la machine virtuelle avec un client SSH (scp, putty, ...)
- Compiler les fichiers C++

```
1 gcc -c echoSK.cc echov1.cc
```

- Créer l'exécutable

```
1 g++ -o echov1 echoSK.o echov1.o -lomnithread -lomniORB4
```

- Lancer l'exécutable

```
1 ./echov1
```

1ère version : echov1.cc

```
1  // echov1.cc
2  #include "echo.hh"
3  #ifdef HAVE_STD
4          #include <iostream>
5          using namespace std;
6  #else
7  #  include <iostream.h>
8  #endif
9
10 class Echo_i : public POA_Echo {
11         public:
12                 inline Echo_i() {}
13                 virtual ~Echo_i() {}
14                 virtual char* echoString(const char* mesg);
15 };
16
17 char* Echo_i::echoString(const char* mesg) {
18         return CORBA::string_dup(mesg);
19 }
20
21
22 static void hello(Echo_ptr e) {
23         if(CORBA::is_nil(e)) {
24                 cerr << "hello: The object reference is nil!\n" << endl;
25                 return;
26         }
27         CORBA::String_var src = (const char*) "Hello!";
28         CORBA::String_var dest = e->echoString(src);
29         cout << "Got \"" << (char*)dest << "\" from server" << endl;
30 }
```

1ère version : echov1.cc

```
int main(int argc, char** argv) {
        try {
                // Initialise the ORB.
                CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
                // Obtain a reference to the root POA.
                CORBA::Object_var obj = orb->resolve_initial_references("RootPOA");
                PortableServer::POA_var poa = PortableServer::POA::_narrow(obj);
                // We allocate the object on the heap.  Since this is a reference
                // counted object, it will be deleted by the POA when it is no
                // longer needed.
                Echo_i* myecho = new Echo_i();
                // Activate the object. This tells the POA that this object is
                // ready to accept requests.
                PortableServer::ObjectId_var myechoid = poa->activate_object(myecho);
                // Obtain a reference to the object.
                Echo_var myechoref = myecho->_this();
                // Decrement the reference count of the object implementation, so // that it will be properly
        cleaned up when the POA has determined // that it is no longer needed.
                myecho->_remove_ref();
                // Obtain a POAManager, and tell the POA to start accepting // requests on its objects.
                PortableServer::POAManager_var pman = poa->the_POAManager();
                pman->activate();
                // Do the client-side call.
                hello(myechoref);
                // Clean up all the resources.
                orb->destroy();
        }
```

1ère version : echov1.cc

```
        catch(CORBA::SystemException& ex) {
                cerr << "Caught CORBA::" << ex._name() << endl;
        }
        catch(CORBA::Exception& ex) {
                cerr << "Caught CORBA::Exception: " << ex._name() << endl;
        }
        catch(omniORB::fatalException& fe) {
                cerr << "Caught omniORB::fatalException:" << endl;
                cerr << "  file: " << fe.file() << endl;
                cerr << "  line: " << fe.line() << endl;
                cerr << "  mesg: " << fe.errmsg() << endl;
        }
        return 0;
}
```

2ème version : espace mémoire distinct

- Le client et le serveur se trouvent sur la même machine mais dans des espaces mémoire distincts (un programme client et un programme serveur) et sont codés en C++
- Le client utilise la référence IOR de l'objet distant pour s'y connecter
- Tous les mentionnés par la suite seront copiés sur la machine virtuelle avec un client SSH (scp, putty, ...)
- Compiler les fichiers C++

```
1 gcc -c echoSK.cc echov2-server.cc echov2-client.cc
```

- Créer l'exécutable du serveur

```
1 g++ -o echov2-server echoSK.o echov2-server.o -lomnithread -
      lomniORB4
```

- Créer l'exécutable du client

```
1 g++ -o echov2-client echoSK.o echov2-client.o -lomnithread -
      lomniORB4
```

2ème version : espace mémoire distinct

- Lancer le serveur dans un premier terminal et noter la référence IOR

```
1 ./echov2-server
```

- Lancer le client dans un second terminal en indiquant la référence IOR

```
1 ./echov2-client IOR:010000000
    d00000049444c3a4563686f3a312e3000000000010000000000000064000000010102000
```

2ème version : echov2-server.cc

```
1  // echov2-server.cc
2  #include "echo.hh"
3  #ifdef HAVE_STD
4          #include <iostream>
5          using namespace std;
6  #else
7          #include <iostream.h>
8  #endif
9
10 class Echo_i : public POA_Echo {
11         public:
12                 inline Echo_i() {}
13                 virtual ~Echo_i() {}
14                 virtual char* echoString(const char* mesg);
15 };
16
17 char* Echo_i::echoString(const char* mesg) {
18         cout << "Upcall " << mesg << endl;
19         return CORBA::string_dup(mesg);
20 }
21
22 int main(int argc, char** argv) {
23         try {
24                 CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
25                 CORBA::Object_var obj = orb->resolve_initial_references("RootPOA");
26                 PortableServer::POA_var poa = PortableServer::POA::_narrow(obj);
27                 Echo_i* myecho = new Echo_i();
28                 PortableServer::ObjectId_var myechoid = poa->activate_object(myecho);
29                 // Obtain a reference to the object, and print it out as a // stringified IOR.
30                 obj = myecho->_this();
31                 CORBA::String_var sior(orb->object_to_string(obj));
32                 cout << (char*)sior << endl;
33                 myecho->_remove_ref();
34                 PortableServer::POAManager_var pman = poa->the_POAManager();
35                 pman->activate();
36                 orb->run();
37         }
```

2ème version : echov2-server.cc

```
        catch(CORBA::SystemException& ex) {
                cerr << "Caught CORBA::" << ex._name() << endl;
        }
        catch(CORBA::Exception& ex) {
                cerr << "Caught CORBA::Exception: " << ex._name() << endl;
        }
        catch(omniORB::fatalException& fe) {
                cerr << "Caught omniORB::fatalException:" << endl;
                cerr << "  file: " << fe.file() << endl;
                cerr << "  line: " << fe.line() << endl;
                cerr << "  mesg: " << fe.errmsg() << endl;
        }
        return 0;
}
```

2ème version : echov2-client.cc

```
1  // echov2-client.cc
2  #include "echo.hh"
3  #ifdef HAVE_STD
4          #include <iostream>
5          #include <fstream>
6          using namespace std;
7  #else
8          #include <iostream.h>
9  #endif
10
11 static void hello(Echo_ptr e) {
12         CORBA::String_var src = (const char*) "Hello!";
13         CORBA::String_var dest = e->echoString(src);
14         cout << "Got \"" << (char*) dest << "\" from server" << endl;
15 }
16
17 int main(int argc, char** argv) {
18         try {
19                 CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
20                 if( argc != 2 ) {
21                         cerr << "usage:  eg2_clt <object reference>" << endl;
22                         return 1;
23                 }
24                 CORBA::Object_var obj = orb->string_to_object(argv[1]);
25                 Echo_var echoref = Echo::_narrow(obj);
26                 if( CORBA::is_nil(echoref) ) {
27                         cerr << "Can't narrow reference to type Echo (or it was nil)." << endl;
28                         return 1;
29                 }
30                 hello(echoref);
31                 orb->destroy();
32         }
```

2ème version : echov2-client.cc

```
        catch(CORBA::TRANSIENT&) {
                cerr << "Caught system exception TRANSIENT -- unable to contact the "
                << "server." << endl;
        }
        catch(CORBA::SystemException& ex) {
                cerr << "Caught a CORBA::" << ex._name() << endl;
        }
        catch(CORBA::Exception& ex) {
                cerr << "Caught CORBA::Exception: " << ex._name() << endl;
        }
        catch(omniORB::fatalException& fe) {
                cerr << "Caught omniORB::fatalException:" << endl;
                cerr << "  file: " << fe.file() << endl;
                cerr << "  line: " << fe.line() << endl;
                cerr << "  mesg: " << fe.errmsg() << endl;
        }
        return 0;
}
```

3ème version : utilisation d'un serveur de nom

- Le client et le serveur se trouvent sur la même machine ou sur deux machines différents et sont codés en C++
- Le serveur s'enregistre auprès d'un serveur de nom
- Le client s'adresse au serveur de nom pour trouver le serveur
- Tous les mentionnés par la suite seront copiés sur la machine virtuelle avec un client SSH (scp, putty, ...)
- Compiler les fichiers C++

```
1 gcc -c echoSK.cc echov3-server.cc echov3-client.cc
```

- Créer l'exécutable du serveur

```
1 g++ -o echov3-server echoSK.o echov3-server.o -lomnithread -
      lomniORB4
```

- Créer l'exécutable du client

```
1 g++ -o echov3-client echoSK.o echov3-client.o -lomnithread -
      lomniORB4
```

3ème version : utilisation d'un serveur de nom

- Lancer le serveur de nom dans un premier terminal

```
1 omniNames -logdir /tmp -start
```

- Lancer le serveur dans un deuxième terminal

```
1 ./echov3-server -ORBInitRef NameService=corbaname::localhost
```

- Lancer le client dans un troisième terminal

```
1 ./echov3-client -ORBInitRef NameService=corbaname::localhost
```

3ème version : echov3-server.cc

```
1  // echov3-server.cc
2  #include "echo.hh"
3
4  #ifdef HAVE_STD
5          #include <iostream>
6          using namespace std;
7  #else
8          #include <iostream.h>
9  #endif
10
11 static CORBA::Boolean bindObjectToName(CORBA::ORB_ptr, CORBA::Object_ptr);
12
13 class Echo_i : public POA_Echo {
14         public:
15                 inline Echo_i() {}
16                 virtual ~Echo_i() {}
17                 virtual char* echoString(const char* mesg);
18 };
19
20 char* Echo_i::echoString(const char* mesg)    {
21         return CORBA::string_dup(mesg);
22 }
23
24 int main(int argc, char **argv) {
25         try {
26                 CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
27                 CORBA::Object_var obj = orb->resolve_initial_references("RootPOA");
28                 PortableServer::POA_var poa = PortableServer::POA::_narrow(obj);
29                 Echo_i* myecho = new Echo_i();
30                 PortableServer::ObjectId_var myechoid = poa->activate_object(myecho);
31                 // Obtain a reference to the object, and register it in // the naming service.
32                 obj = myecho->_this();
33                 CORBA::String_var x;
34                 x = orb->object_to_string(obj);
35                 cout << x << endl;
```

3ème version : echov3-server.cc

```
                if(!bindObjectToName(orb, obj))
                        return 1;
                myecho->_remove_ref();
                PortableServer::POAManager_var pman = poa->the_POAManager();
                pman->activate();
                orb->run();
        }
        catch(CORBA::SystemException& ex) {
                cerr << "Caught CORBA::" << ex._name() << endl;
        }
        catch(CORBA::Exception& ex) {
                cerr << "Caught CORBA::Exception: " << ex._name() << endl;
        }
        catch(omniORB::fatalException& fe) {
                cerr << "Caught omniORB::fatalException:" << endl;
                cerr << "  file: " << fe.file() << endl;
                cerr << "  line: " << fe.line() << endl;
                cerr << "  mesg: " << fe.errmsg() << endl;
        }
        return 0;
}

static CORBA::Boolean bindObjectToName(CORBA::ORB_ptr orb, CORBA::Object_ptr objref) {
        CosNaming::NamingContext_var rootContext;
        try {
                // Obtain a reference to the root context of the Name service:
                CORBA::Object_var obj;
                obj = orb->resolve_initial_references("NameService");
                // Narrow the reference returned.
                rootContext = CosNaming::NamingContext::_narrow(obj);
                if( CORBA::is_nil(rootContext) ) {
                cerr << "Failed to narrow the root naming context." << endl;
                return 0; }
        }
```

3ème version : echov3-server.cc

```
        catch (CORBA::NO_RESOURCES&) {
                cerr << "Caught NO_RESOURCES exception" << endl;
                return 0;
        }
        catch (CORBA::ORB::InvalidName&) {
                // This should not happen!
                cerr << "Service required is invalid [does not exist]." << endl;
                return 0;
        }
        try {
                // Bind a context called "test" to the root context:
                CosNaming::Name contextName;
                contextName.length(1);
                contextName[0].id = (const char*) "test";
                contextName[0].kind = (const char*) "my_context";
                CosNaming::NamingContext_var testContext;
                try {
                        // Bind the context to root.
                        testContext = rootContext->bind_new_context(contextName);
                }
                catch(CosNaming::NamingContext::AlreadyBound& ex) {
                        // If the context already exists, this exception will be raised.
                        // In this case, just resolve the name and assign testContext
                        // to the object returned:
                        CORBA::Object_var obj;
                        obj = rootContext->resolve(contextName);
                        testContext = CosNaming::NamingContext::_narrow(obj);
                        if( CORBA::is_nil(testContext) ) {
                                cerr << "Failed to narrow naming context." << endl;
                                return 0;
                        }
                }
```

3ème version : echov3-server.cc

```
        // Bind objref with name Echo to the testContext:
        CosNaming::Name objectName;
        objectName.length(1);
        objectName[0].id = (const char*) "Echo";
        objectName[0].kind = (const char*) "Object";
        try {
                testContext->bind(objectName, objref);
        }
        catch(CosNaming::NamingContext::AlreadyBound& ex) {
                testContext->rebind(objectName, objref);
        }
    }
    catch(CORBA::TRANSIENT& ex) {
        cerr << "Caught system exception TRANSIENT -- unable to contact the "
        << "naming service." << endl
        << "Make sure the naming server is running and that omniORB is "
        << "configured correctly." << endl;
        return 0;
    }
    catch(CORBA::SystemException& ex) {
        cerr << "Caught a CORBA::" << ex._name()
        << " while using the naming service." << endl;
        return 0;
    }
    return 1;
}
```

3ème version : echov3-client.cc

```
1  // echov3-client.cc
2  #include "echo.hh"
3  #ifdef HAVE_STD
4          #include <iostream>
5          using namespace std;
6  #else
7          #include <iostream.h>
8  #endif
9
10 static CORBA::Object_ptr getObjectReference(CORBA::ORB_ptr orb);
11
12 static void hello(Echo_ptr e) {
13         if( CORBA::is_nil(e) ) {
14                 cerr << "hello: The object reference is nil!\n" << endl;
15                 return;
16         }
17         CORBA::String_var src = (const char*) "Hello!";
18         CORBA::String_var dest = e->echoString(src);
19         cerr << "Got \"" << (char*)dest << "\" from server" << endl;
20 }
21
22 int main (int argc, char **argv) {
23         try {
24                 CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
25                 CORBA::Object_var obj = getObjectReference(orb);
26                 Echo_var echoref = Echo::_narrow(obj);
27                 hello(echoref);
28                 orb->destroy();
29         }
30         catch(CORBA::TRANSIENT&) {
31                 cerr << "Caught system exception TRANSIENT -- unable to contact the "
32                 << "server." << endl;
33         }
34         catch(CORBA::SystemException& ex) {
35                 cerr << "Caught a CORBA::" << ex._name() << endl;
36         }
```

3ème version : echov3-client.cc

```
        catch(CORBA::Exception& ex) {
                cerr << "Caught CORBA::Exception: " << ex._name() << endl;
        }
        catch(omniORB::fatalException& fe) {
                cerr << "Caught omniORB::fatalException:" << endl;
                cerr << "  file: " << fe.file() << endl;
                cerr << "  line: " << fe.line() << endl;
                cerr << "  mesg: " << fe.errmsg() << endl;
        }
        return 0;
}

static CORBA::Object_ptr getObjectReference(CORBA::ORB_ptr orb) {
        CosNaming::NamingContext_var rootContext;
        try {
                // Obtain a reference to the root context of the Name service:
                CORBA::Object_var obj;
                obj = orb->resolve_initial_references("NameService");
                // Narrow the reference returned.
                rootContext = CosNaming::NamingContext::_narrow(obj);
                if( CORBA::is_nil(rootContext) ) {
                        cerr << "Failed to narrow the root naming context." << endl;
                        return CORBA::Object::_nil();
                }
        }
        catch (CORBA::NO_RESOURCES&) {
                cerr << "Caught NO_RESOURCES exception. You must configure omniORB "
                << "with the location" << endl
                << "of the naming service." << endl;
                return 0;
        }
```

3ème version : echov3-client.cc

```
        catch(CORBA::ORB::InvalidName& ex) {
                // This should not happen!
                cerr << "Service required is invalid [does not exist]." << endl;
                return CORBA::Object::_nil();
        }
        // Create a name object, containing the name test/context:
        CosNaming::Name name;
        name.length(2);
        name[0].id = (const char*) "test";
        name[0].kind = (const char*) "my_context";
        name[1].id = (const char*) "Echo";
        name[1].kind = (const char*) "Object";
        try {
                // Resolve the name to an object reference.
                return rootContext->resolve(name);
        }
        catch(CosNaming::NamingContext::NotFound& ex) {
                // This exception is thrown if any of the components of the // path [contexts or the object] aren't
        found:
                cerr << "Context not found." << endl;
        }
        catch(CORBA::TRANSIENT& ex) {
                cerr << "Caught system exception TRANSIENT -- unable to contact the "
                << "naming service." << endl
                << "Make sure the naming server is running and that omniORB is "
                << "configured correctly." << endl;
        }
        catch(CORBA::SystemException& ex) {
                cerr << "Caught a CORBA::" << ex._name()
                << " while using the naming service." << endl;
                return 0;
        }
        return CORBA::Object::_nil();
}
```

4ème version : multi-langages

- Le client et le serveur se trouvent sur la même machine ou sur deux machines différents
- Le serveur est codé en C++ et le client en Java
- Le serveur s'enregistre auprès d'un serveur de nom
- Le client s'adresse au serveur de nom pour trouver le serveur
- Tous les mentionnés par la suite seront copiés sur la machine virtuelle avec un client SSH (scp, putty, ...)
- Génération des fichiers Java (Stub, Skeleton, ...)

```
1 idlj echo.idl
```

- Compilation des fichiers Java

```
1 javac Echo.java EchoHelper.java EchoHolder.java EchoOperations.java
      _EchoStub.java Echov3Client.java
```

4ème version : multi-langages

- Lancer le serveur de nom dans un premier terminal

```
1 omniNames -logdir /tmp -start
```

- Lancer le serveur dans un deuxième terminal

```
1 ./echov3-server -ORBInitRef NameService=corbaname::localhost
```

- Lancer le client Java dans un troisième terminal

```
1 java Echov3Client -ORBInitRef NameService=corbaname::localhost:2809
```

4ème version : Echov3Client.java

```java
// Echov3Client.java
import org.omg.CosNaming.*;
import org.omg.CORBA.*;
import java.util.Properties;

public class Echov3Client {
        public static void main(String args[]) {
                try{
                        // create and initialize the ORB
                        ORB orb = ORB.init( args, null );
                        // get the root naming context
                        org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
                        NamingContext ncRef = NamingContextHelper.narrow( objRef );
                        System.out.println("Naming Service was found successfully..." );
                        // resolve the Object Reference in Naming:
                        NameComponent nc1 = new NameComponent("test", "my_context");
                        NameComponent nc = new NameComponent("Echo", "Object" );
                        NameComponent path[] = {nc1, nc};

                        System.out.println("About to find Component...." );
                        Echo echoRef = EchoHelper.narrow(ncRef.resolve(path));

                        // call the "ECHO" server object and print results
                        String msg = "Hello!";
                        String hello;

                        hello = echoRef.echoString(msg);
                        System.out.println("Got \"" + hello + " from server \"");

                }
                catch (Exception e) {
                        System.out.println("ERROR: " + e) ;
                        e.printStackTrace( System.out );
                }
        }
}
```

References

- Cours de S. Chabridon et C. Taconet
- Site d'omniORB : http ://omniorb.sourceforge.net/
- "Distributed Systems : Principles and Paradigms", A. S. Tanenbaum
- "Distributed Systems", P. Krzyzanowski