Corba, une introduction par l'exemple

par Sylvain BARTHELEMY

Date de publication : 18/11/2000

Dernière mise à jour :

L'informatique répartie est un des enjeux principaux de l'informatique actuelle et à venir. De l'architecture de terminaux centralisée qui servait encore de modèle il y a peu de temps, nous somme passé à l'architecture client/serveur. L'information est maintenant répartie entre différents serveurs du système et non plus centralisée sur un seul et même serveur : elle est répartie et distribuée aux clients de façon transparente. L'information distribuée, voilà donc le maître mot. Le world wide web, qui nous est si familier, est un exemple réussi d'application distribuée. Les clients, par l'intermédiaire d'un navigateur, se connectent à des serveurs sur lesquels des pages web nous sont proposées (le plus souvent encore statiques). Cependant, rien ne vous empêche de proposer vous aussi de l'information ou même d'installer votre serveur (sauf peut être l'InterNIC...), vous êtes alors client... mais aussi serveur : le web se sont des clients mutliples mais aussi des serveurs multiples.

I - La norme CORBA

CORBA ou *Common Object Request Broker Architecture* est né dans les années 1990 du besoin de faire communiquer ensemble des applications en environnement hétérogène (plusieurs systèmes et plusieurs langages). CORBA n'est qu'un sous ensemble du projet ODP-RM ou *Open Distributed Processing Référence Model* vise à mettre en place une norme d'architectures distribuées ouvertes. L' *Object Management Group* , créé en 1989 et regroupant plus de 750 membres (vendeurs de logiciels, développeurs et utilisateurs), collabore avec l'ODP pour la mise en place d'une norme sur le middleware, et a créé à cet effet la norme CORBA.

CORBA est donc une norme et rien ne contraint l'implémentation les développeurs de produits CORBA. Ils ont le libre choix de l'implémentation. La popularité de l'Internet à ensuite amené les concepteurs de la norme CORBA à définir un protocole de communication entre les objets adaptés à Internet. Ce protocole s'appelle IIOP, pour *Internet Inter-ORB Protocol*. Sommairement, l'*Object Management Architecture Guide* défini :

1. **CORBA** : cadre visible que doivent respecter les développeurs de bus CORBA;
2. **CORBA Services** : les services objet communs (*Common Object Services*) proposent des outils pour faciliter l'utilisation courante des objets CORBA. Les services sont constitués des interfaces suivantes (CORBA 2.2 / 98-10-53) : Naming, Event, Persistent Object, Life Cycle, Concurrency Control, Externalization, Relationship, Transaction, Query, Licensing, Property, Time, Security, Object Trader, Object Collections
3. **CORBA Facilities** : les utilitaires communs définissent un cadre de haut niveau pour la création de logiciels à l'aide de composants réutilisables. Ils sont à ce jour bien moins avancés que les services. Les domaines concernés par les utilitaires sont : la gestion des tâches, l'interface utilisateur, la gestion de l'information, l'administration système
4. **Domain Interfaces** : les interfaces de domaines concernent des marchés verticaux et définissent donc des interfaces spécialisée répondant aux besoins spécifiques de tel ou tel marché comme les télécommunications ou la finance par exemple.

A l'heure où j'écris ces lignes, la norme CORBA en est à sa version 2.2 mais on parle déjà de la version 3.

Si CORBA constitue la première initiative majeure dans le domaine des objets distribués, quelques alternatives sont aujourd'hui possibles. Ainsi, il m'a semblé important ici de noter quelques différences entre CORBA et d'autres solutions proposées pour l'informatique répartie et les objets distribués : RPC et RMI et DCOM. Le cas de RMI ou *Remote Method Invocation* ne concerne que les dialogues Java/Java et donc se différencie de Java par le fait que cette méthode est attachée à un langage particulier. RPC ou *Remote Procedure Call* permet comme son nom l'indique d'invoquer des procédures distantes mais se trouve mal adapté aux dialogues d'objets distants. Enfin, le cas de DCOM est plus complexe car DCOM est l'alternative CORBA proposée par Microsoft. Si CORBA défini une norme, DCOM défini aussi son implémentation et se limitait jusque récemment à l'interface Windows. De très bonnes études sont disponibles sur le Web au sujet de la comparaison des deux systèmes. Ayant personnellement pratiqué les deux, CORBA me parait beaucoup plus simple et pratique d'utilisation.

II - Le langage de définition d'interfaces : IDL

A la base de CORBA, est défini l'IDL ou *Interface Definition Language*. Il permet de définir dans un langage commun à l'ensemble des compilateurs, la partie visible d'un objet : méthodes et propriétés. Le principe utilisé ici est la séparation de l'interface d'un objet et de son implémentation.

A titre d'exemple, le listing 1. Présente l'interface IDL d'un service minimal de gestion du compte d'un client (cet exemple est l'équivalent CORBA du fameux hello.c,hello.java, hello.pl...) :

Listing 1.

```
// CompteClient.idl
interface CompteClient {
     void credit ( in unsigned long montantFRF );
     void debit  ( in unsigned long montantFRF );
     long solde  ( );
};
```

Au premier coup d'oeil, les développeurs familiers de C++ auront sans doute remarqué certaines ressemblances entre ce dernier et IDL. Peut importe l'implémentation de l'interface CompteClient du moment qu'elle répond à l'interface spécifiée. IDL génère une souche (stub) pour le client et un squelette (skeleton) pour le serveur.

Cette opération est effectuée simplement par la commande :

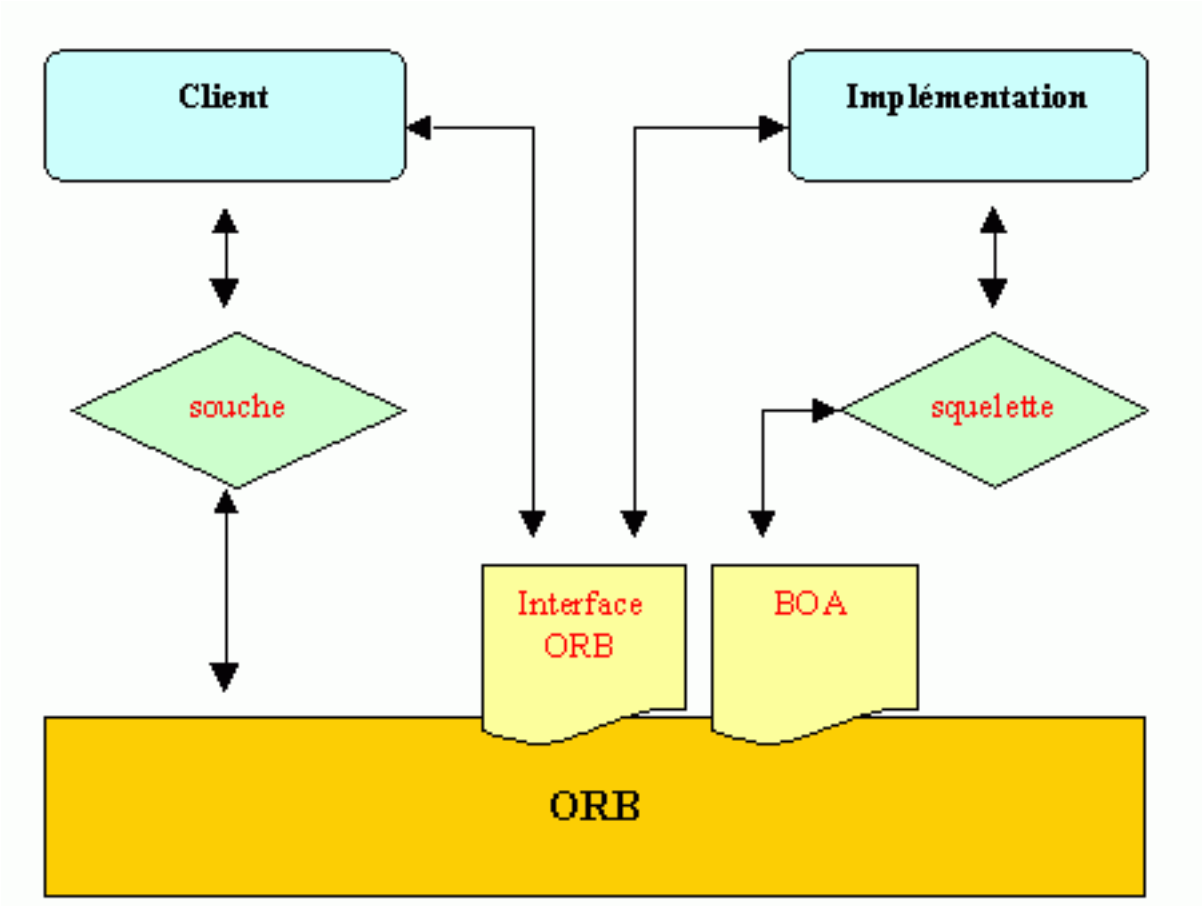
```
idl CompteClient.idl
```

qui génère les classes C++ requises dans le fichier CompteClient.cc.

III - Le bus d'objets de CORBA

Le coeur de CORBA est constitué par l'ORB qui est le conduit/bus par lequel les requêtes sur les objets transitent. L'*Object Request Broker* ou ORB n'est pas directement accessible au programmeur d'applications CORBA qui n'utilise qu'une référence. L'ORB gère les transferts entre les clients et les serveurs. L'adaptateur d'objets, est quand à lui chargé de la gestion des références aux objets et de l'activation des implémentations : l'adaptateur d'objets est donc, comme son nom l'indique, le " canal " par lequel l'ORB est connectée à l'implémentation de l'objet. Différents types d'adaptateur d'objets peuvent être implémenté mais la norme CORBA 2.0 ne requière que le BOA ou *Basic Object Adapter*.

Ainsi, à l'aide d'un compilateur idl et de vos fichiers idl (présenté dans la section précédente), vous générez des souches pour les clients et des squelettes pour les serveurs. Vous n'avez qu'à programmer l'implémentation de votre classe serveur (ici une classe est assimilée à une classe C++) qui doit hériter du squelette généré par idl. Une fois vos programmes clients et serveurs compilés, vous pourrez lancer votre serveur et l'ORB fera tout le travail de pliage/dépliage des requêtes des clients sur les objets locaux et distants à vôtre place. Pour plus de clarté un schéma explicatif du bus CORBA est donné ci-dessous.



Le bus CORBA

IV - Exemple

Afin d'illustrer les concepts d'ORB et de BOA voiçi le code d'un serveur CORBA (avec nommage) associé à l'interface CompteClient :

```
// serveur.cc
#include "mico/naming.h"
#include "CompteClient.h"

// --- implémentation
class CompteClient_impl : virtual public CompteClient_skel {
    CORBA::Long _solde;
public:
    CompteClient_impl () { _solde = 0; }
    void credit( CORBA::ULong montantFRF ) { _solde += montantFRF; }
    void debit ( CORBA::ULong montantFRF ) { _solde -= montantFRF; }
    CORBA::Long solde()                    { return _solde; }
};

// --- lancement du serveur
int main(int argc, char* argv[])
{
// --- initialisation du bus et de l'adaptateur d'objets
CORBA::ORB_var orb = CORBA::ORB_init ( argc, argv, "mico-local-orb" );
CORBA::BOA_bar boa = CORBA::BOA_init ( argc, argv, "mico-local-boa" );

// --- création de l'objet
CompteClient* cc = new CompteClient_impl;

// --- connexion au service de nommage
CORBA::Object_var            ns = orb-resolve_initial_references ("NameService");
CosNaming::NamingContext_var nc = CosNaming::NamingContext::_narrow (ns);

// --- définition du chemin d'accès
CosNaming::Name name;
name.length (1);
name[0].id   = CORBA::string_dup ("MonCompte");
name[0].kind = CORBA::string_dup ("");

// --- enregistrement de l'objet
nc-bind (name, cc);

// --- serveur ok
boa-impl_is_ready (CORBA::ImplementationDef::_nil());
orb-run ();

// --- fin...
CORBA::release(cc);
return 0;
}
```

Le client n'implante pas d'objets et n'a donc pas besoin du BOA

```
// client.cc
#include "iostream.h"
#include "mico/naming.h"
#include "CompteClient.h"

int main(int argc, char* argv[])
{
// --- initialisation du bus et de l'adaptateur d'objets
CORBA::ORB_var orb = CORBA::ORB_init ( argc, argv, "mico-local-orb" );

// --- connexion au service de nommage
CORBA::Object_var            ns = orb-resolve_initial_references ("NameService");
CosNaming::NamingContext_var nc = CosNaming::NamingContext::_narrow (ns);

// --- définition du chemin d'accès
CosNaming::Name name;
name.length (1);
name[0].id   = CORBA::string_dup ("MonCompte");
name[0].kind = CORBA::string_dup ("");
```

```
	// --- résolution du nom pour obtenir une référence sur MonCompte
	CORBA::Object_var obj    = nc-resolve (name);
	CompteClient_var  compte = CompteClient::_narrow( obj );

	// --- quelques opérations sur mon compte...
	compte-credit(1000000);
	cout  "solde: "  compte-solde()  endl;
	// --- ... super !

	return 0;
	}
```

V - Bibliographie

Je remercie les auteurs cités dans la bibliographie pour leurs ouvrages dans lesquels j'ai largement puisé pour écrire cette note.

[GGM97] J.M. Geib, C. Gransart, P. Merle, *CORBA : des concepts à la pratique*, Masson, 1997.

[POP97] A. Pope, *The CORBA référence guide : understanding the common object request broker architecture*, Addison Wesley, 1997.

[PUD98] A. Puder, *The MICO CORBA compliant system*, DDJ, p.44-51, november 1998.