Projets Middleware

Voyage au pays des Middlewares

A la découverte d'intergiciels « exotiques »

Thème 1

- Middleware pour le Grid Computing
  - Grid computing ?
  - Etudier le projet Globus
    - Middleware pour le Grid
    - Fonctionnalités,
    - Expérimentation
  - Autres projets similaires similaires/intéressants/relatifs
  - 1 à 3 étudiants

Thème 2

- Real-time Middleware
  - Temps-réel?
  - Etudier les différents projets
    - Fonctionnalités,
    - En choisir 1 par élève : expérimentation
  - Autres projets similaires/intéressants/relatifs
  - 1 à 3 étudiants

Thème 3

- P2P Middleware
  - P2P ?
  - Etudier les projets JXTA, Bit Torrent, Pastry
    - Fonctionnalités,
    - Expérimentation (comparaison des performances entre étudiants)
    - Avalanche
  - Autres projets similaires similaires/intéressants/relatifs
  - 1 à 6 étudiants

Objectif final

- Présentation des concepts/technologies
- Domaines d'application
- Maturité des technologies
- Comparaison (fonctionnalités, performances)
- Démonstration des plateformes
- Petite application commune :
  - Chat ? Pierre-papier-ciseaux?

Corba

Common Object Request Broker Architecture

(D'après les cours de C. Bouthier, K. Baina, P.Merle)

Laurent Ciarletta
@mines.inpl-nancy.fr

Organisation

- Rappels et Contexte
- OMG

Corba

- Common Object Request Broker Architecture
- Une architecture d'interopérabilité distribuée et orientée objet selon un modèle client-serveur ouvert

Plan

- Architecture CORBA
- Le langage IDL
- CORBA en Java : JavaIDL
- Le service de nommage
- CORBA en C++ : Mico

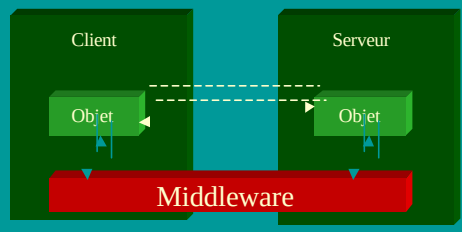
CORBA

Architecture

Contexte

- Applications distribuées
- Objets répartis
- Autres technos :
  - RPC
  - COM+
  - RMI
  - .NET

Principe simplifié



Object Management Group (OMG)

- Créé en 1989
- But non lucratif
- Plus de 850 membres (Sun, IBM, Microsoft, ...)
- Crée et maintient les spécifications
  - CORBA
  - UML
- http://www.omg.org

L'Object Management Group

Consortium créé en 1989
actuellement plus de 850 membres

Objectif: faire émerger des standards pour l'intégration d'applications distribuées hétérogènes et la réutilisation de composants logiciels

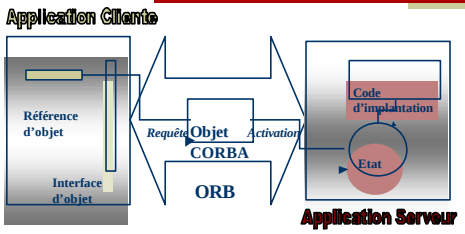
OMG : concepts-clés

- Permettre l'interopérabilité des composants / applications par l'intermédiaire d'un mode de coopération unifié: l'appel à des objets distants
- Gérer l'hétérogénéité des machines, systèmes et langages par l'utilisation d' un langage pivot commun: OMG-IDL
- Offrir une architecture globale

OMG : vision globale

- Common Object Request Broker Architecture CORBA
  - un bus à objets répartis ORB
    - transport des requêtes, activation des objets
  - des services de base (CORBAservices)
  - des utilitaires communs (CORBAfacilities)
  - des interfaces de domaines = objets métiers
    - « interopérabilité sémantique »

OMG : le modèle client/serveur objet



OMG : les objectifs techniques

- Liaison avec tous les langages
- Transparence des invocations
- Invocation statique et dynamique
- Système auto-descriptif
- Activation automatique
- Interopérabilité entre bus

OMG-IDL : le langage d'interfaces

Ce langage a été défini:

- ⌘ pour décrire les interfaces des objets
- ⌘ comme langage pivot entre applications
- ⌘ pour générer des squelettes de programme dans les langages de programmation des applications

OMG-IDL : les éléments

- **pragma**
- **module**
- **types de base au format binaire normalisé**
- **nouveaux types**
  (typedef, enum, struct, union, array, sequence)
- **exception**
- **interface (avec héritage multiple sans surcharge)**
- **opération (synchrone ou asynchrone sans résultat)**
- **attribut (possibilité de lecture seule)**
- **types de méta-données (TypeCode, any)**

OMG-IDL : évaluation

- Bon langage de spécification d'interfaces d'objets
- Facile à apprendre

⇳ types de base figés
⇳ pas de passage d'objets par valeur
⇳ pas de sémantique ni qualité de service
⇳ pas de notion d'architecture logicielle

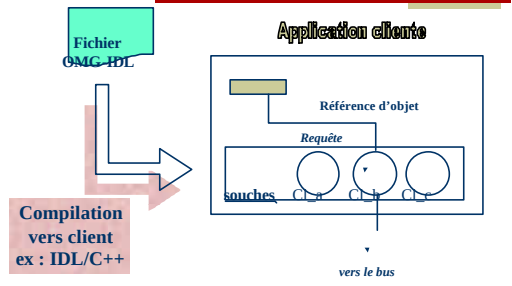
OMG-IDL : compilation

- Projection des descriptions OMG-IDL vers les langages d'implantation des clients et serveurs.
  - mode « statique »
- Instanciation sous forme d'objets CORBA des descriptions OMG-IDL dans un référentiel des interfaces commun.
  - mode « dynamique »

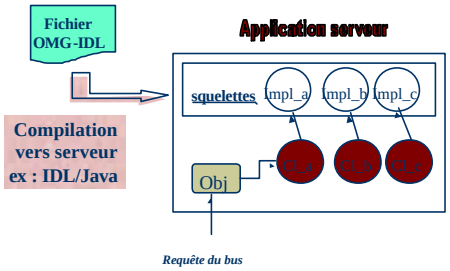
CORBA : le mode statique

- Les clients et serveurs implantent des descriptions OMG-IDL communes, statiquement précisées dans la phase de compilation/projection du source IDL.
- Les souches générées encapsulent l'utilisation du bus, l'activation et la distribution des composants et l'hétérogénéité de l'architecture.

La projection client



La projection serveur



CORBA : le mode dynamique

- Un « référentiel des interfaces » stocke sous forme d'objets les descriptions d'interfaces OMG-IDL.
- Une API (DII: Dynamic Invocation Interface) permet de construire des requêtes à l'exécution.

Le référentiel des interfaces

- Graphe d'objets « concepts » d 'IDL
  - ModuleDef, InterfaceDef, OperationDef, AttributeDef, TypedefDef, ...
- Par navigation dans ce graphe ou à partir d'une référence d'objet, on peut retrouver le contenu d'une interface, la signature d'une opération, ...

L'invocation dynamique

- API (DII) de construction de requêtes
  - sans passer par des souches pré-générées
- Un objet *Request* = un nom d'opération, une liste de couples valeur - type (au sens de l'IR) et une structure pour le résultat
  - *invoke*
  - *send_deferred + get_response, poll_response*
  - *send_oneway*

Le squelette dynamique

- API (DSI) de décodage de requêtes
  - sans utiliser de squelettes pré-générés
- Création de passerelles
- Utilisée pour implanter dynamiquement des objets

CORBA : les références d'objet

- GIOP protocole d'interopérabilité entre bus de la norme CORBA 2.
  - Internet Inter ORB Protocol: IIOP
- Interoperable Object Reference: IOR
  - interface OMG-IDL
  - adresse + port IP
  - clé de format libre

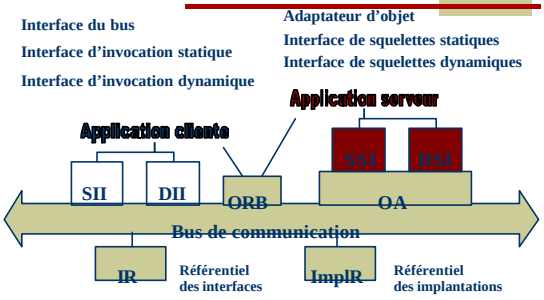
CORBA : les interfaces de base

- L'interface Object
  - implicitement héritée par toute interface
  - encapsule une IOR
- L'interface ORB
  - initialisations
  - fournit les « objets notoires »
  - *object_to_string, string_to_object*

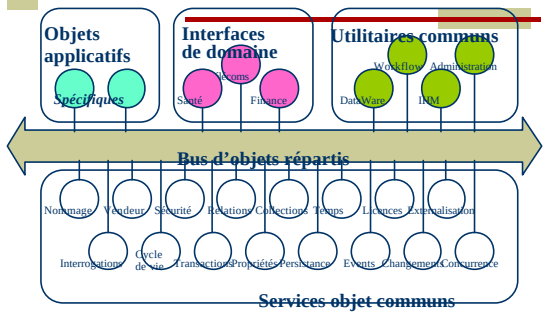
CORBA : les adaptateurs d'objets

- Intermédiaire entre le bus et les implantations possibles des objets
  - BOA pour processus
  - OODA pour SGBDOO
  - LOA pour bibliothèques
  - COA pour carte à microprocesseur
  - POA de CORBA 2.2
  - Un OA peut utiliser un Référentiel des Implantations

CORBA : les composantes du bus



OMA : l'architecture globale



OMA : les services communs

- Services de recherche d'objets
  - Service Nommage (Naming)
    - pour retrouver un objet par un nom
    - service de « pages blanches »
  - Service Vendeur (Trader)
    - pour retrouver un objet par des propriétés
    - service de « pages jaunes »

OMA : les services communs

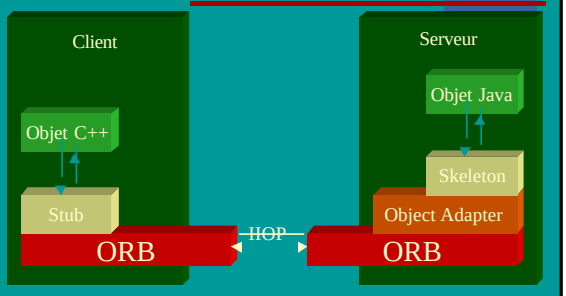
- Services de communications asynchrones
  - Events, Notification, Messaging
- Services de sûreté de fonctionnement
  - Security, Transactions, Concurrence
- Services concernant la vie des objets
  - Life Cycle, Property, Relationship, Externalization, Persistent Object, Query, Collection, Versionning, Time, Licencing

OMA : les canevas d'objets

- Canevas horizontaux (CORBAfacilities)
  - workflow, realtime, test, UML
- Canevas verticaux (Domain Interfaces ou objets métiers)
  - Business Object Frameworks: BOF
  - CORBAmed, Electronic Commerce, Financial Services, Telecom, Simulation, Life Sciences

En pratique

Architecture de CORBA



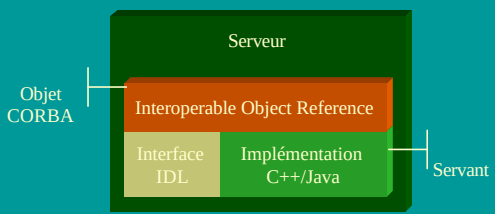
Composants

- Stub : partie cliente
- Skeleton : partie serveur

} générés automatiquement

- Object Adapter : enregistrement des objets
- ORB : passage de message
- IIOP : Internet Inter-ORB Protocol

Objet CORBA



Exemple d'IOR

IOR:000000000000001049444c3a5472697669616
c3a312e3000000000010000000000000007c0001
02000000000d3135322e38312e342e313130000
0048000000025abacab31313030333836323133
36005f526f6f74504f410000cafebabe3bd5b8780
0000000000000000000010000001000000002c00
0000000010001000000040001002000010109000
1010005010001000101090000000200010100
05010001

Interface Definition Langage (IDL)

- Description des interfaces
- Indépendant du langage d'implémentation
- Indépendant de la plate-forme client
- Indépendant de la plate-forme serveur

- Ressemble beaucoup au C++

Exemple

```
module example {
   interface monExample {
        void methode1();
        long methode2();
        void methode3(in long param, out long result);
   };
};
```


Stub (souche)

- Code client
- Interface entre objet et ORB
- Traduit les invocations sur l'objet serveur
  - > marshalling
- Traduit les messages en valeurs de retour
  - unmarshalling

Skeleton (squelette)

- Code serveur
- Interface entre implémentation et ORB
- Traduit les invocations client vers l'implémentation
  - > unmarshalling
- Traduit la valeur de retour en message vers client
  - > marshalling

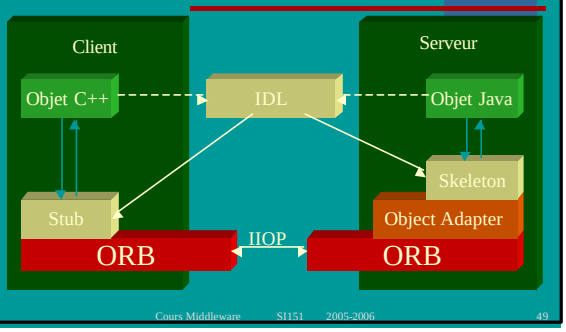
Object Request Broker (ORB)

- Transporte les messages entre les objets
- Relie les stubs aux skeletons correspondants et vice-versa
- Bus à objets
- Communications inter-ORBs :
  - GIOP (General Inter-ORB Protocol)
  - IIOP (Internet Inter-ORB Protocol) (GIOP on TCP/IP)

Object Adapter (OA)

- Enregistre et gère les implémentations
- Activation et désactivation des objets
- Invocation des méthodes
- Authentification du client / contrôle d'accès
- Différents types :
  - BOA – Basic Object Adapter
  - POA – Portable Object Adapter

Architecture de CORBA



Création d'une application CORBA

- 1 – Définir l'interface IDL
- 2 – Compiler l'interface IDL
- 3 – Créer l'implémentation de l'interface IDL
- 4 – Créer le serveur :
  - > publication de l'objet CORBA
- 5 – Créer le client :
  - > appel de l'objet CORBA

CORBA

IDL

Exemple

```
module example {
   interface monExample {
        void methode1();
        long methode2();
        void methode3(in long param, out long result);
   };
};
```


Module

```
module monModule {
 …
};
```

- espace de nom
  - C++ : namespace
  - Java : package

Interface

```
interface monInterface {
   …
};
```

- Classe
  - C++ : class
  - Java : interface

Méthode

```
void methode(in long param, out long result);
```

- Méthodes : comme en C++ et Java, sauf
  - in : paramètre utilisé en entrée (lu, non modifié)
  - out : paramètre utilisé en sortie (non lu, modifié)
  - inout : paramètre utilisé en entrée et en sortie (lu et modifié)

Syntaxe

- Commentaire : comme C++ et Java
  - // : jusqu'à la fin de la ligne
  - /* … */ : bloc de commentaire
- Préprocesseurs : #define, #include, #ifdef, #endif
- Alias : typedef
- Possibilité de définir des attributs
- Constantes : const

Types primitifs

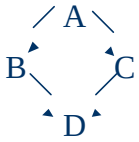
- void
- short
- long
- long long
- float
- double
- boolean

Types complexes

- string
- struct
- enum
- union
- any

Héritage

- Héritage multiple
- Surcharge et redéfinition interdites



CORBA

JavaIDL

JavaIDL

- ORB CORBA en Java de Sun
- Inclus au JDK
- idlj : compilateur idl vers java
  - Syntaxe : idlj -fall <fichier.idl>
- orbd : ORB et serveur de nom indépendant
  - Syntaxe :

  orbd -ORBInitialPort <port> -ORBInitialHost localhost

Implémentation

- Fichier MonObject.IDL :

  interface MonObjet { … };
- Classe qui implémente le contrat IDL
  - Doit dériver de MonObjetPOA
  - MonObjetPOA est généré par le compilateur IDL
- Lancement client ou serveur :
  - java <serveur/client> -ORBInitialPort <port> -ORBInitialHost localhost

Serveur / 1

- Initialiser l'ORB :
  - Méthode statique init(args, null) de la classe ORB
  - retourne un objet ORB
- Objet CORBA :
  - org.omg.CORBA.Object
  - Ne pas confondre avec java.lang.Object
  - Retourné par différentes méthodes
  - **Cast** avec méthode statique narrow de la classe MonTypeHelper

Serveur / 2

- Utilisation d'un POA :
  - Méthode resolve_initial_references("RootPOA") de l'objet ORB
  - Retourne un objet CORBA à caster en objet POA
  - Puis méthode the_POAManager().activate() de l'objet POA
- Enregistrer le servant :
  - Méthode servant_to_reference(servant) de l'objet POA
  - retourne un objet CORBA
- Lancement de l'ORB (à la fin) :
  - Méthode run de l'objet ORB

Client

- Initialiser l'ORB :
  - Méthode statique init(args, null) de la classe ORB
  - retourne un objet ORB
- Récuperer un objet CORBA depuis son IOR :
  - Méthode string_to_object(IOR) de l'objet ORB
  - Retourne un objet CORBA
  - Cast dans son vrai type avec la méthode statique narrow de la classe MonObjetHelper
  - MonObjetHelper est généré par le compilateur IDL

CORBA

Service de nommage

Services CORBA

- Service de cycle de vie
- Service d'événements
- Service de concurrence
- Service de transaction
- Service de persistance
- Service d'interrogation
- Service de collection

Services de recherche d'objets

- Service de nommage (Naming)
  - Recherche d'objet par nom
  - pages blanches
- Service vendeur (Trader)
  - Recherche d'objet par propriété
  - pages jaunes

Besoin du Naming

- Référence d'un objet : IOR
  - Fichier partagé, …
- Service à la DNS :
  - Service accessible par le bus ORB
  - Service standard entre ORBs
  - Un nom spécifique <-> un objet corba
  - Gère les contextes de nom

Utilisation du Naming

- Module **CosNaming**
- Interface **NamingContext**
- Nouvelle interface **NamingContextExt**
  - Création d'une association : **bind, rebind**
  - Résoudre une association : **resolve**
  - Détruire une association : **unbind**
- Programme indépendant à lancer avant : **orbd**

Obtenir le Naming

- Naming = objet CORBA
  - Défini en IDL
  - Associé au nom « NameService »
- Racine de l'arbre de référence :
  - Méthode resolve_initial_references de l'ORB
  - Conversion en NamingContext ou NamingContextExt
- Possibilité de spécifier où chercher les initial_references

Naming dans JavaIDL : coté serveur

- Lancer orbd avant
- Récupérer le Naming :
  - orb.resolve_initial_references("NameService") retourne un objet CORBA
  - Puis cast avec NamingContextExtHelper.narrow(obj)
- Créer une association (objet NamingContextExt) :
  - to_name("NOM") retourne un chemin sous forme de NameComponent[]
  - Puis enregistrement avec rebind(chemin, obj)

Naming dans JavaIDL : coté client

- Même initialisation/récupération :
  - orb.resolve_initial_references("NameService") retourne un objet CORBA
  - Puis cast avec NamingContextExtHelper.narrow(obj)
- Récupération de l'objet CORBA à partir du nom :
  - ns.resolve(ns.to_name("NOM")) retourne un objet CORBA
  - Puis cast avec MonObjetHelper.narrow(obj)

Vers CORBA 3

- CORBA 2.2
  - client/serveur orienté objet, les interfaces de base et le POA, les mécanismes dynamiques, GIOP, OMG-IDL et ses projections vers C, C++, SmallTalk, COBOL, ADA et Java.
- CORBA 3.0
  - interfaces multiples, passage par valeur,
  - modèle de composant, langage de script,
  - minimumCORBA, realtimeCORBA, CORBA/COM/DCE
  - messaging, printing, fault tolerance, firewall

Conclusion

- Solution non propriétaire
- Standard
- Libre choix de l'implémentation et du langage
- Solution fonctionnelle d'interopérabilité
- Plusieurs implémentations libres et open-source

- Mais : un peu « usine à gaz » …

CORBA

Mico

Mico Is COrba

- ORB CORBA en C++
- Open-source et libre (GPL)
- Multi-plateforme :
  - Linux/Unix
  - Windows
    - Visual C++
    - Cygwin
  - MacOS X
  - Pocket PC

Client

- idl : compilateur idl vers C++
  - Syntaxe : idl <fichier.idl>
  - Génère monObjet.cc
- Compilation :
  - Compiler monObjet.cc en monObjet.o
  - Lier avec monObjet.o, mico2.3.5, micocoss2.3.5, dl, m
- Indiquer où trouver le Naming :

```
<client>
-ORBDefaultInitRef corbaloc::localhost:<port>
```


Code client / 1

- Initialisation ORB :
  - Namespace CORBA
  - Fonction ORB_init(argc, argv, "mico-local-orb") retourne un objet ORB_var
- Name Service :
  - Namespace CosNaming
  - resolve_initial_references("NameService") de l'objet ORB
  - retourne un objet CORBA (CORBA::Object_var)
  - Cast avec méthode de classe _narrow(obj) de la classe NamingContextExt

Code client / 2

- Naming (suite)
  - Cast retourne un objet NamingContextExt _var
- Récuperation d'un objet par le Naming :
  - Méthode resolve_str("Nom") de l'objet NamingContextExt _var
  - Retourne un objet CORBA
  - Cast avec la méthode de classe _narrow(obj) de la classe MonObjet
  - Retourne un objet MonObjet_var

Pragma

- **pragma** (pragmatic information)
  A standardised form of comment which has meaning to a compiler. It may use a special syntax or a specific form within the normal comment syntax. A pragma usually conveys non-essential information, often intended to help the compiler to optimise the program.