Objets distribués

Architectures, état de l'art, pratiques

Partie C-Corba

Plan

- Précédente séance:
  - A-Vue d'ensemble sur les SàOD (45mn)
  - B-Java RMI (35mn)

- Cette séance:
  - C-Corba
    - Architecture et modele objet de CORBA
    - Les composants fournis par l'ORB
    - Le langage IDL (Interface Definition Language)
    - Mise en œuvre en C++ (java sera vu en TP)
    - Le service de nom (Naming Service)

C- CORBA

Partie 1/5 - Architecture et modèle objet de CORBA

Le standard Corba

- CORBA signifie Common Object Request Broker Architecture.
- C'est une architecture et infrastructure ouverte, indépendante du constructeur, utilisable pour des applications fonctionnant en réseau quelque soit la configuration de :
  - ordinateur
  - systèmes d'exploitation
  - langage de programmation
  - Types de réseaux
- L'OMG (Object management group) maintient le standard CORBA.
- CORBA **n'est pas** un Système Distribué mais sa simple specification
  - spécifications principale de plus 700 pages.
  - Plus 1200 pages pour spécifier les services naviguant autour
- Versions:
  - 1991: 1ere spécification corba : standardisation de IDL et OMA
  - 1996: Corba 2: spécifications plus robustes- standardisation de IIOP, POA, DynAny
  - 2002: Corba 3: ajout du component model, Qos (asynchronisme, tolerance de panne, RT Corba- Corba temps réel)

Object Management Group

- créé en 1989 en tant que société non commerciale
- But non (directement) lucratif
- plus de 800 membres (Sun, DEC, IBM, Apple, Hewlett-Packard, ...)- plus gros consortium du monde!
- Petite équipe dédiée (30 personnes en 2000) - pas de dév interne (uniquement des spécifications)
- http://www.omg.org
- Dedié à la création de standards OO pour l'integration d' applications

- a créé et maintient les spécifications
  - CORBA (1er version au début des années 1990)
  - UML
  - Autres: MOF(Meta Object Facility) & XMI (Xml Metadata Interchange)

Objectif de corba

- CORBA permet l'interconnexion d'Objets (donc d'applications) quelque soit
  - Le langage des applications qui fournissent ou utilisent les objets
  - La position de l'ordinateur (connexion à travers Internet possible)
  - L'architecture machine des ordinateurs reliés

- Une petite analogie:
  - *CORBA est pour l'informatique OO ce que le World Wide Web est pour les documents.*

Moyen: corba IDL

- CORBA IDL
  - Pour permettre l'independence de langage, de site et de plateforme, une **interface** entre les objets est ecrite dans un langage de specification appelé **IDL**.
  - IDL est ensuite utilisé pour générer du code concret dans un L3G spécifique à une plateforme (cl ou srv)

- Pour poursuivre l'analogie:
  - *CORBA est pour l'informatique OO ce que le World Wide Web est pour les documents.*
  - *IDL est à CORBA ce que HTML est au World Wide Web.*
- *Exemple de fichier IDL:*

```
module demo {
        interface hello {
            string sayHelloWorld();
        };
};
```

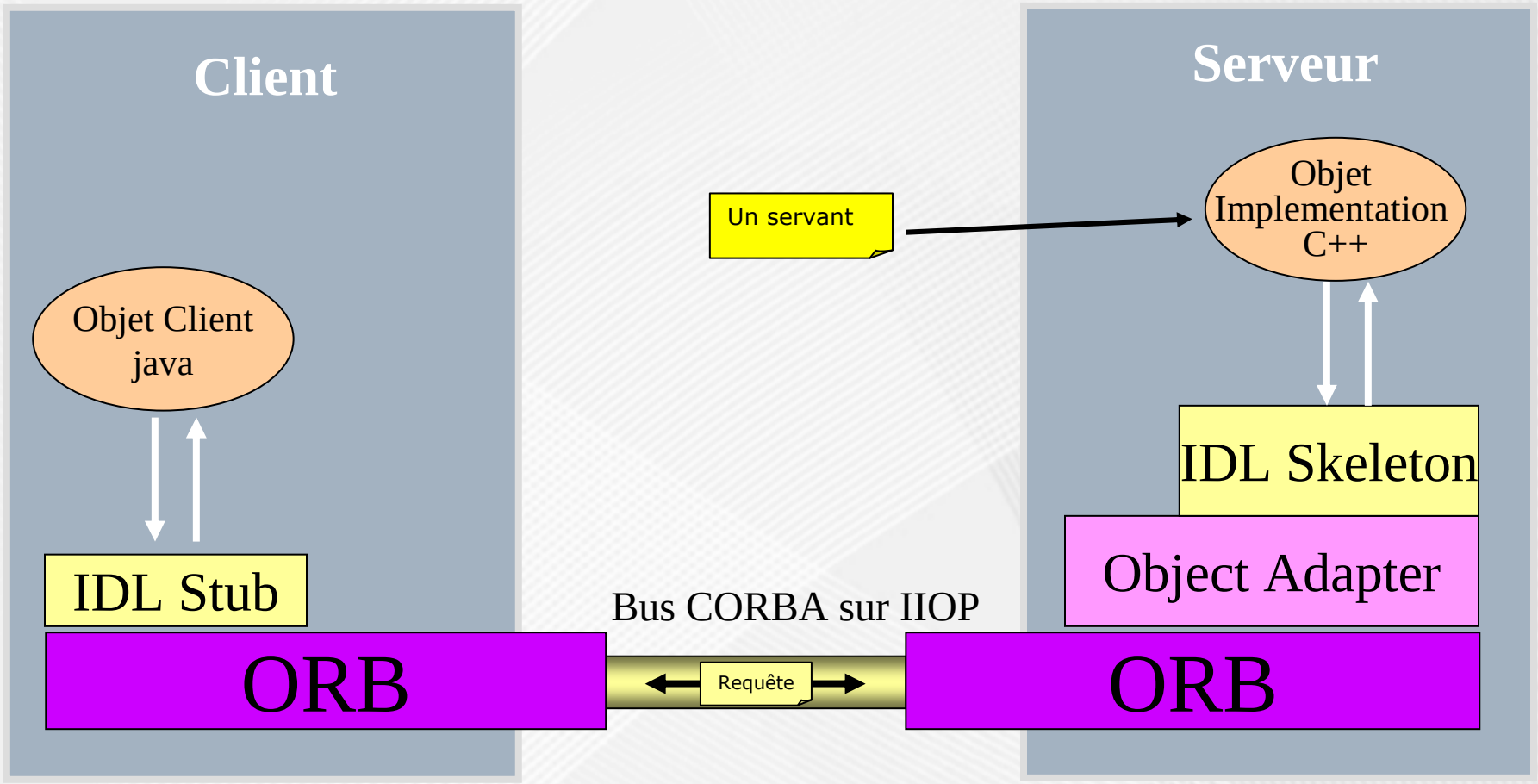
Le modèle d'application Corba

- Les applications CORBA sont composées d'objets Corba
- Pour chaque objet, les développeurs définissent une interface en IDL (OMG Interface Definition Language).
- L'interface représente le contrat que l'objet serveur fournis aux clients potentiels.
- Tout client voulant appeler une méthode de l'objet Corba doit utiliser cette interface IDL pour désigner la méthode en question, et "marshalliser" les paramètres.
- L'appel passe par l'objet Request broker (ORB) pour atteindre l'objet cible. Pour cela il a fait usage de la même definition IDL pour "unmarshalliser" les paramètres
- L'ORB utilise la même definition IDL pour "marshaliser" le resultat pour le retour
- L'ORB de l'objet appelant doit à son tour "unmarshalliser" le retour

Compilation IDL

- Le fichier IDL est compilé en utilisant un compilateur IDL (fournis par l'ORB) vers le langage de programmation cible
  - On dit aussi « Projection IDL » vers un langage ou encore « Mapping IDL »
  - Il existe des compilateur IDL pour tous les langages de programmation connus, dont JAVA, C, and C++.
- Cette interface "concrète" est ensuite implémentée dans le langage de programmation cible (par le développeur)
- Stubs et Skeletons (Souche et squelette)
  - Le compilateur IDL a en fait généré le code d'un stub pour le coté client et un skeleton pour le coté serveur.
  - Ceux ci sont chargé de gérer la communication bas niveau (réseaux) liés aux appels de méthodes

le modèle objet de Corba



Le Bus Objet

- Les ORBs (Object Request Brokers) sont lancés sur chacune des machine du système distribué
- Lancé sur le même port de chaque machine, ils forment ainsi un « bus »
- Les ORBs se transmette les requêtes entre Objet CORBA distribué via le protocole IIOP (Internet Inter-Orb Protocol)
- Les ORBs sont plateforme dépendants, mais disponible pour toutes les plateformes connues
  - Pour un même ORB, il existe donc une distribution pour chaque OS

Les Objets CORBA

- **Objet CORBA** : un "objet virtuel" pouvant être localisé par un ORB et recevoir des requêtes clients.
  - Peut être instancié par un ou plusieurs Servants.
- **Servant** : objet concret dans un langage de programmation existant dans la mémoire d'un serveur et contenant le code implémentant un objet CORBA.
- Tout Objet CORBA possède une référence d'objet.
- **Référence d'Objet** : correspond à l'identité d'un objet Corba, pouvant être utilisé par les clients pour appeler des méthodes sur ses instances (les servants)
  - Utilisé comme « handle » d'objets
  - Les Informations dans la référence d'objet contiennent des informations sur la localisation et le type de l'instance en question.

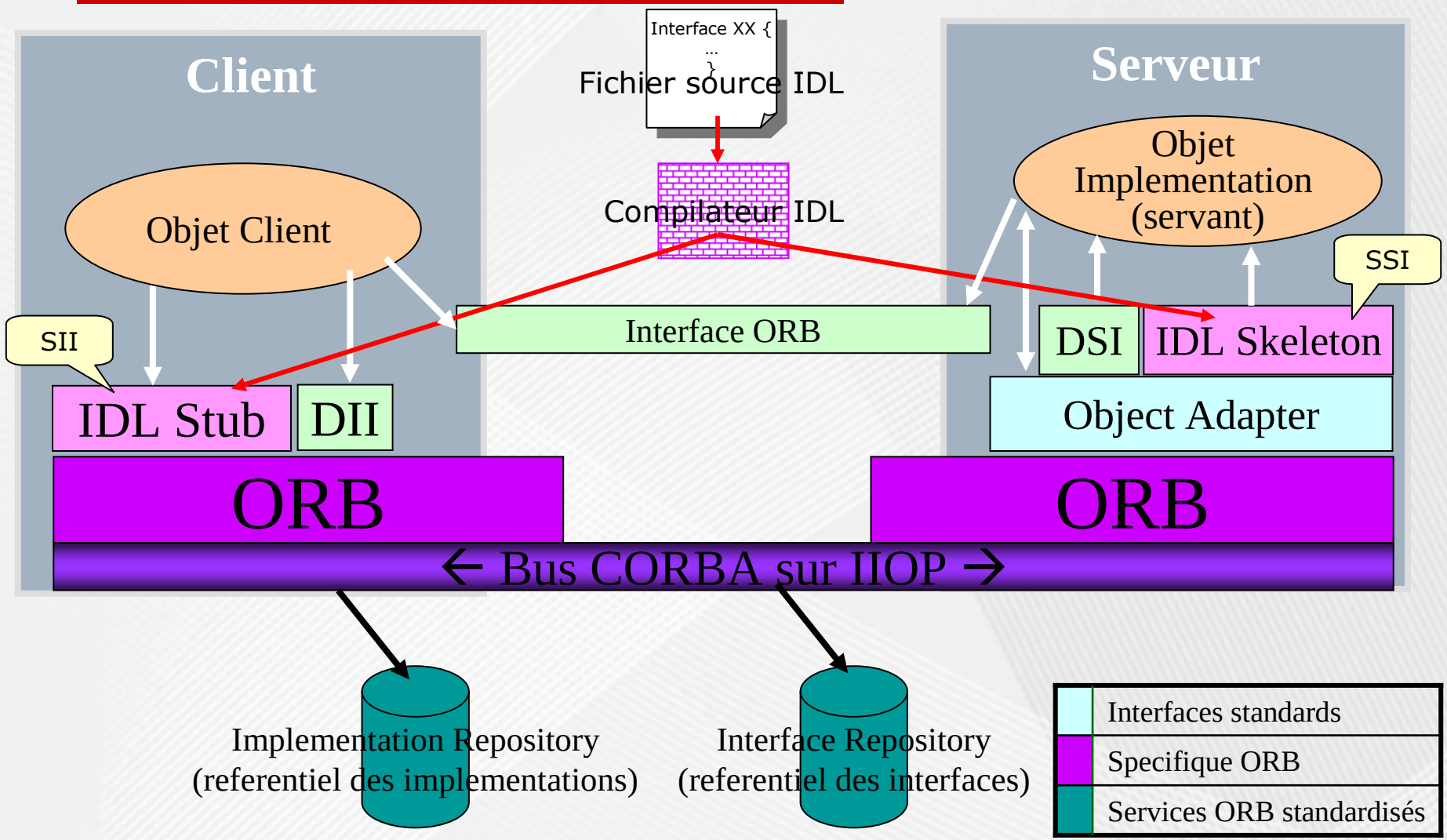
Reference d'Objet (IOR)

- Nommé Ior=Interoperable Object Reference
- Un IOR est aussi un objet en interne
- Contient principalement :
  - Une série de paramètres regroupés par "profile"
  - Un "profile" : Un protocole, hôte, port et une "Clé" objet
  - Ce sont ces profiles qui rendent la référence interoperable
- Les IORs peuvent être convertis en chaine et vice versa
  - Fonction ORB::Object_to_string
    IOR Objet → IOR chaine
  - Fonction ORB::string_to_Object
    IOR chaine → IOR objet
  - Plusieurs formats possibles :
    - "IOR:" "FILE:" "HTTP:" "CORBALOC:" et "FTP:"

EXEMPLE d'IOR:

IOR:010000002a00000049444c3a6f6f632e636f6d2f436f734e616d696e672f4f424e616d696e67436f6e746578743a312e
3000610001000000000000000700000000010102000e00000038322e36362e3139322e3137350034072800000abacab305
f526f6f74504f4100526f6f74436f6e74657874504f4100004e616d65536572766963650100000001000000200000001fe
120001000100020000002000010001000105000101000100000009010100

le modèle (plus) complet de Corba-1



le modèle (plus) complet de Corba-2

Coté Client

- **Static Invocation Interface (SII): stubs IDL**
  - La colle entre le client et l'ORB.
  - Les stubs agissent comme des proxy d'appels pour le client car ils passent l'appel à l'ORB
  - Ils gèrent le marshalling de la requête (conversion des paramètres dans un format de transport réseau).
- **Object Request Broker (ORB)**
  - Fournis les mécanismes de transparence du transport.
- **Interface ORB**
  - découple une application de l'implémentation de l'ORB
- **Dynamic Invocation Interface (DII)**
  - Permet de générer des requêtes dynamique.
  - Alternative aux stubs de SII pour permettre aux clients de construire un appel à l'exécution.

Coté Serveur

- **Static Skeleton Interface (SSI) : skeletons IDL**
  - La colle entre le serveur et l'ORB.
  - Reçoivent les appels (requêtes) de l'objet Adapter et les transmettent à la méthode de l'objet implémentation.
  - Ils gèrent l'unmarshalling de la requête (conversion des paramètres d'un format de transport réseau vers un format mémoire).
  - Fournis aussi à l'ORB la liste des méthodes qu'ils fournis (détails sur les paramètres et autres)
- **Object Adapter**
  - assiste l'ORB pour délivrer les requêtes.
  - Relie et associe un servant à un ORB.
  - Tout ORB compatible Corba doit être compatible avec la dft du BOA ou du POA
- **Dynamic Skeleton Interface (DSI)**
  - Le pendant serveur de DII.
- **Servant**
  - Un objet d'implémentation (concret) dans un langage de programmation donné.

Retour sur le mode dynamique

- Petits noms du mode dynamique:
  - Dynamic Invocation Interface (DII)
  - Dynamic Skeleton Interface (DSI)

- **Pourquoi**? Les codes de stub et de skeleton sont statique, cela peut atteindre ses limites à l'usage

- Grace à DII, on peut définir un appel pour les clients qui connaissent au moins:
  - La reference objet
  - Le type d'interface
- Grace à DII, ils peuvent construire des appels sans avoir à s'appuyer sur un stub généré par un IDL.
  - Création d'un objet *Request* qui désigne une méthode et ses paramètres sous forme de liste de couples (valeur,type)

- DSI gère les requetes envoyées par les clients de manière generique.
  - En regardant les méthodes appelées ainsi que leur paramètres et en interprétant la semantique dynamiquement.

mode dynamique: DII et DSI

- DII permet d'envoyer des requetes de 3 manières:
  - **Synchrone**: invoke() qui est bloquant
  - **Asynchrone**: send_deferred() qui est non bloquant (on appelle ensuite get_response() ou poll_response() pour obtenir le retour de l'execution- get etant bloquant alors que poll non)
  - **Aller-Simple**: send_oneway() signifie qu'on fait un appel sans demander de réponse au serveur *(imaginons un appel déclenchant un exit() coté serveur)*
- DSI permet d'appeler une implementation (un servant) sans Skeleton statique

Retour sur les termes

- CORBA
  - ***Une spécification (une doc!)***
  - Common Object Request Broker Architecture
- IDL
  - ***Un langage de description***
  - Interface Definition Language
- ORB
  - ***logiciel de communication réseaux***
  - Object Request broker
- IIOP
  - ***Protocole réseau***
  - Internet Inter-Operability Protocol
- OMG
  - ***Un Organisme***
  - créateur de la norme Corba et UML
  - Object Management Group

Retour sur IIOP

- IIOP Signifie Internet Inter-ORB Protocol. C'est GIOP sur TCP
- GIOP Signifie "General Inter-ORB Protocol"
  - C'est un Framework de protocole de transport
  - Le protocole définis 8 *types de messages* différents
  - Fait pour être fiable, orienté connexion
  - GIOP peut être appliqué sur de nombreux protocoles de transport "concrets"
  - Pour chaque protocole en gros il faut implémenter 5 *types de messages*.

- Type de Message de GIOP:

Principalement: Request et reply

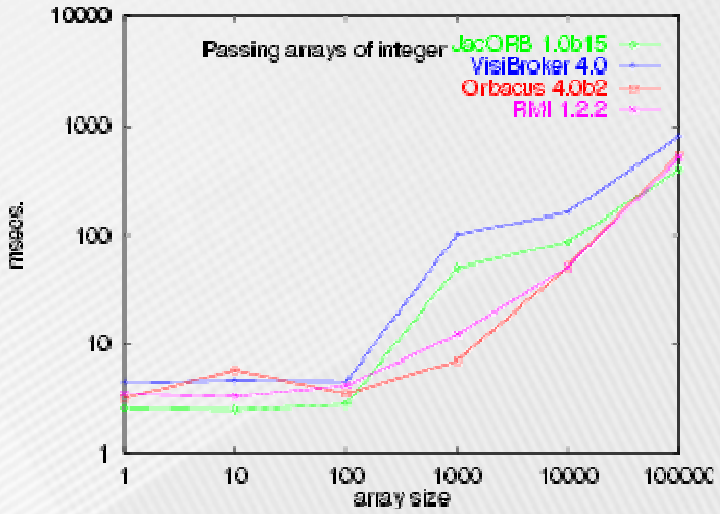
Type de Message	Emetteur	Description
Request	Client	une requête invocation
Reply	Serveur	la réponse à une invocation
LocateRequest	Client	une requête demandant la position d'un objet
LocateReply	Serveur	information sur la position d'un objet
CancelRequest	Client	indiquer que le client n'attend plus de réponse
CloseConnection	Les 2	Indication que la connexion va être fermée
MessageError	Les 2	Contient les informations sur une erreur
Fragment	Les 2	Partie d'un gros message

CORBA

Partie 2/5 - Les composants fournis par l'ORB

Principaux ORB

- Commerciaux:
  - Visibroker (Borland/inprise) (c++ et java)
  - Orbix (Iona) (c++ et java)
  - e*ORB (PrismTech) (c++ et java)
  - DAIS(PeerLogic) (c++ et java)
  - Component Broker (IBM) (java)
- Gratuits/Open source :
  - Orbacus (Iona) (c++ et java)
  - OmniOrb (c++)
  - JacOrb (java)
  - Orbit (C et Ada)
  - Mico (C++)



Comparatif de performance entre quelques ORB

(Source : [JacORB Performance compared])

Composants fournis par l'ORB

- un Dynamic Invocation Interface (DII)
- un Dynamic Skeleton Interface (DSI)
- L'Implementation Repository (IMR)
- L'Interface Repository (IR)
- L'interface ORB
- L'Object Adapter (adaptateur d'objet)
  - Basic Object Adapter
  - Portable Object Adapter
  - …
- Une implémentation plus ou moins complète des "Corba Services" (page blanche, etc) – qu'on voit plus loin

Rappel : Intermédiaire entre le bus et les objets implémentations

Ce qui est fourni varie d'un ORB à l'autre, surtout s'il est open source..

Sélection: ORB & fonctionnalités

	Langages							Fonctionnalités				Corba services					
Nom	**C++**	**C**	**SmalTk**	**Ada**	**Java**	**Python**	**COBOL**	**DII**	**DSI**	**BOA**	**POA**	**Nm**	**Ev**	**Tr**	**Tx**	**Tm**	**Sc**
PrismTech e*ORB	Y	Y	Y	-	Y		-	Y	Y	Y	-	Y	Y				
IONA Orbix	Y	-	-	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y		Y
Inprise Visibroker	Y	-	Y	-	Y		Y	Y	Y	Y	Y	Y	Y		Y		Y
PeerLogic DAIS	Y	Y	-	-	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y		Y
IBM Comp. Broker	Y	Y	Y	Y	Y		Y	Y	Y	Y	-	Y					
Iona ORBacus	Y	-	-	-	Y			Y	Y	-	Y	Y	Y	Y		Y	Y
JacORB	-	-	-	-	Y		-	Y	Y	-	Y	Y	Y	Y	Y		Y
omniORB	Y	-	-	Y	-	Y	-	Y	Y	Y	Y	Y	Y				
Mico	Y	-	-	-	-		-	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
ORBit	-	Y	-	Y	-		-	Y	Y	-	Y						
TAO	Y	-	-	-	-		-	Y	Y	-	Y	Y	Y	Y		Y	Y

NM	Naming Service
EV	Event Service
TR	Trading Service
TX	Transaction Service
TM	Time Service
SC	Security Service

Fonctions de l'ORB

- **Fonction principale** :
  - L'ORB (Object Request Broker) joue le role de médiateur entre objets à travers le réseau
  - Délivrer des requêtes aux objets et renvoyer les reponses aux clients appelants.

- Cache les détails bas-niveau aux objets de manière transparente vis à vis de:
  - **L'Emplacement des objets**: les clients ne savent pas où se situent les objets cible (Locaux ou distants)
  - **L'Implémentation des objets** : Langage et plateforme (OS). les clients ne savent pas comment les objets sont implémentés.
  - **L'état d'exécution des objets :** En cours d'exécution ou nécessitant une initialisation.
    Les clients ne savent pas si les objets sont activés ou désactivés quand ils font des requêtes. C'est l'ORB qui s'occupe de ca.
  - **Le Mécanisme de Communication entre objet :** TCP/IP, shared memory, appel local de méthode.
    Les clients ne savent pas quel méthode l'ORB utilise pour transmettre les requêtes.

Les interfaces fournies par l'ORB

- L'interface Objet
  - encapsule une IOR
  - toute interface déclarée en IDL hérite de Object de manière implicite

- L'interface ORB
  - Fournit une méthode à appeler pour initialiser l'ORB:
    - CORBA::ORB_init()
  - Fournit Object_to_string(), string_to_Object()
    - Représentation interne <-> chaine IOR
  - Méthode pour récupérer les « objets ORB » :
    - ORB::resolve_initial_reference()
    - Utilisé pour récupérer les objets ORB comme le **RootPOA** et le Naming

On voit ca plus en détail plus loin dans le cours..

L'adaptateur d'objet

- Joue le rôle de "colle" entre les implémentations d'Objets CORBA (les servants) et l'ORB.
- Forcément dépendant du langage de programmation

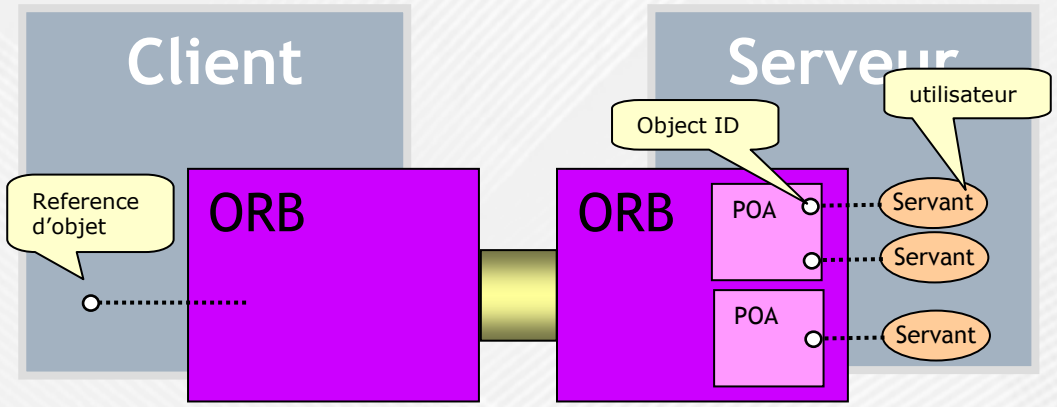
- Fonctions principales :
  - Retrouver un servant**:** pour une requête arrivant pour un objet Corba, retrouve le bon servant.
  - Distribution d'appel**:** une fois que le servant est retrouvé, passe la requête au servant.
  - Activation et désactivation: revient à incarner/etherealiser les servants
  - Générer les Références d'objets

- 2 adaptateurs d'objet specifiés dans CORBA:
  - **Le BOA** (Basic Object Adapter) est obsolète. Les spécifications étaient incomplètes. Manque de portabilité entre ORB. Pas de gestion de servant
  - **Le POA** (Portable Object Adapter) a été bâti sur les erreurs du BOA. spécification d'interfaces pour gérer les références d'objets et les servants.
  - *Le POA est supposé portable entre ORB !*

Le POA (Portable Object Adapter)

- un ORB contient un ou plusieurs POA
- Tout ORB contient un POA spécial appelé ”rootPOA”
- Les POA supplémentaires sont créés comme fils du RootPOA
- La raison? Différente ”politique” de distribution peuvent etre affectée à chaque POA :
  - politique de distribution d’appel
    - partagé: un process serveur pour toutes les requêtes
    - Par client: un process serveur par client
    - Par requête: un process par requête
    - …Entre autres
  - politique de durée de vie des références Objets
    - références volatiles
    - références persistantes

Le POA (Portable Object Adapter)



- ObjectID: un identificateur, unique dans un POA donné, utilisé par le POA pour retrouver un objet CORBA
  - c'est une partie de la "clé" objet contenu dans l'IOR (1 objet CORBA pour 1 ObjectID)
  - Il est possible d'avoir plusieurs servants pour un même objectId
  - Il est possible d'avoir plusieurs objectId incarnés par un seul servant (servant par defaut)
- Actions du POA sur les servants:
  - Incarner : l'action de fournir un servant pour les requêtes associés à un objectID particulier
  - Etherealiser: l'action de detruire une association objectID/servant
  - Activer: Quand un objet CORBA a été associé à un servant incarnant l'objet
  - Desactiver: quand un objet CORBA n'a plus de servant incarné

L'Interface Repository (IR)

- Un démon (prog serveur) fournis par l'ORB
- Une base persistante de définitions d'interfaces
  - Stocke les signatures de méthodes
  - La clé est générée par le compilateur IDL (pour chaque fichier compilé)
  - C'est souvent un process serveur à part entière
- Utilisé pour :
  - interopérabilité entre ORBs différent
  - Contrôle de type de signatures de requêtes (appels dynamiques)
  - Contrôle de l'héritage
- tout ORB à au moins un IR

- Exemple avec ORBacus:
  - irserv, irfeed, irdel

Objectifs de l'IMR

- L'IMR est un démon
  - Exemple: orbixd, osagent, imr, imrd
- Objectifs:
  - Gestion des IORs persistants
  - Permettre à un serveur de s'arrêter et d'être réveillé à la demande
  - Pour cela autorise le Mécanisme **de la liaison dynamique**
  - **Ce mécanisme permet de donner** plusieurs serveurs/emplacements pour un objet
  - Permet de gérer efficacement le load-balancing et la tolerance de panne

L'Implementation Repository (IMR)

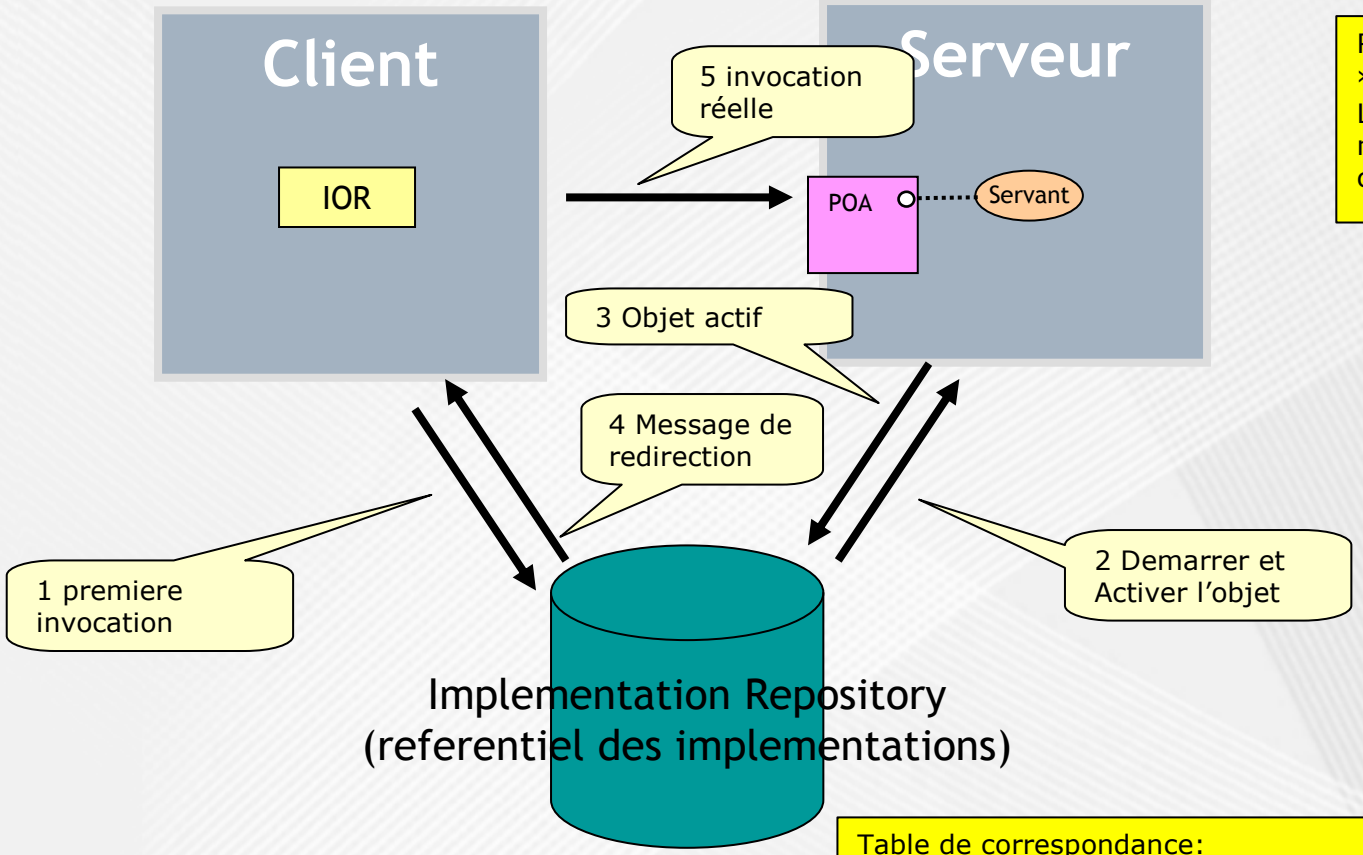
- Un démon (prog serveur) fournis par l'ORB
- Une base d'information sur les servants (objets d'implementation) que l'ORB utilise pour retrouver le serveur hébergeant un servant
- L'IMR stocke une correspondance entre des noms d'adaptateurs d'objets et des addresses (host:port) serveur.
  - L'IMR doit savoir démarrer le serveur également!

- **Objectifs:**
  - Gestion des IORs persistants
  - Permettre à un serveur de s'arreter et d'etre réveillé à la demande
  - Pour cela autorise le Mecanisme **de la liaison dynamique**
  - **Ce mécanisme permet de donner** plusieurs serveurs/emplacements pour un objet
  - Permet de gérer efficacement le load-balancing et la tolerance de panne

- Exemple: Imr avec Orbacus (ORBD avec Java JDK 1.4)

L'Implementation Repository (IMR)

- Interêt principal: la liaison indirecte CORBA



Redirection: « LocationForward »
Lorsqu'un ORB recoit ce message, il refait un essai avec cette nouvelle adresse

Table de correspondance:
Nom d'objet corba <-> machine/port + commande de démarrage
Exemple de commande (unix): rsh -n machine /usr/corba/AppliServeur

L'Implementation Repository (IMR)

- Liaison Directe
  - L'ORB du client contacte l'ORB du serveur grace à l'addresse contenue dans l'IOR
  - Utilisé avec les IOR provisoires
  - Avantage : rapide (un seul aller-retour)
  - Desavantage: si le serveur ne tourne pas !?
- Liaison indirecte
  - Utilisé pour les IOR **persistants**
  - L'ORB client contacte l'ORB du serveur via un 'daemon' (un process serveur distinct de l'ORB) lors du premier appel.
  - Avantage : le serveur peut etre lancé lorsque necessaire. Possibilité de déplacer le port et l'IP des machines sans changer l'IOR
  - Desavantage: plus lent (deux allers-retours)
- Mais le serveur IMR doit toujours etre lancé !
- Les serveurs doivent connaitre les coordonnées de l'IMR (host:port)
  - De manière à placer le hostname:port de l'IMR dans les IORs qu'il délivre.
  - C'est l'ORB du serveur qui se charge de cela

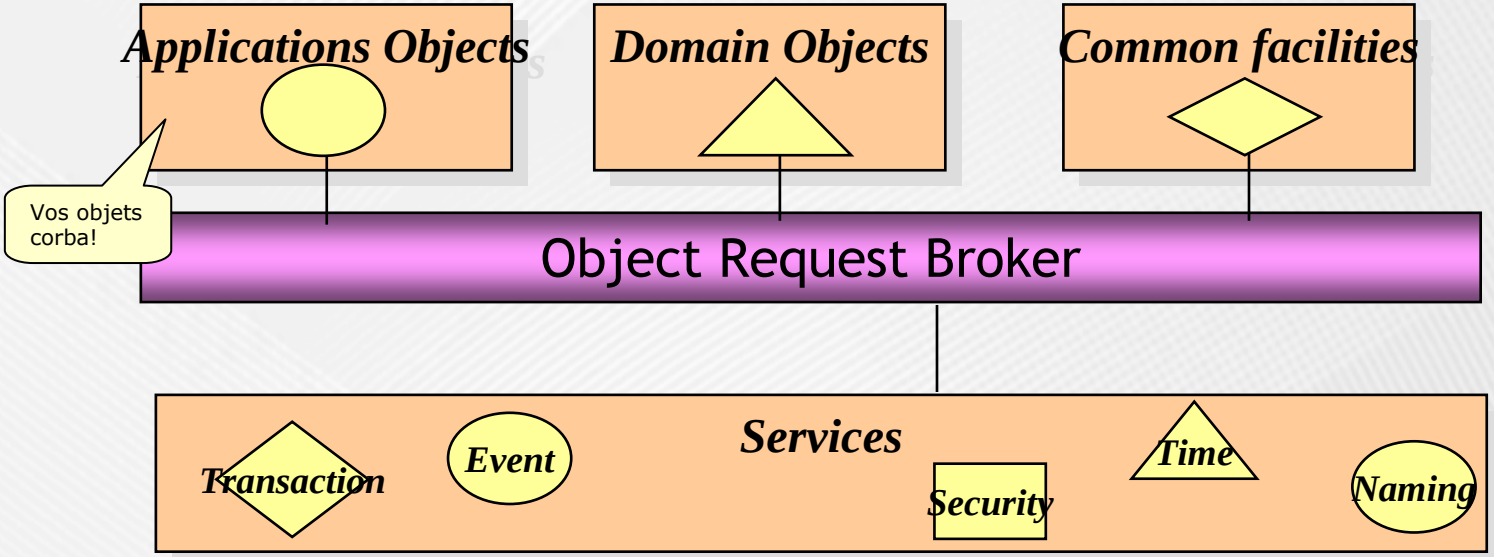
Object Management Architecture (OMA)

- OMG maintient aussi la specification de l'OMA
- OMA définis principalement le "Modèle de Reference"
- Le Modèle de Reference introduit **4 grandes categories** pour les objets Corba
  - CORBAServices : préfixés par COS=Common objet Services
  - CORBAFacilities : Common Facilities ou encore Common Services ou Horizontal Facilities
  - CORBADomains : Domain objects ou encore Domain Interfaces ou Vertical Facilities
  - Objets Applications : applications utilisateur

Voir prochain slide

- OMA definis les interfaces pour ces objets en laissant l'implementation à sociétés commerciales

Object Management Architecture (OMA) – Modele de référence



- **CORBA Services**
  - Services independents du Domaine
  - *Ex: Naming Service and Trading Service*
  - *COS= Common Object Services*
- **CORBA Facilities**
  - Ou encore "Common Services" ou Horizontal Facilities
  - Facilités génériques, potentiellement utiles quelquesoit le domaine d'application
  - *Seulement 4 actuellement connues: Printing, Secure Time, Internationalization, Mobile Agent.*
- **Domain Objects**
  - Ou encore Vertical Facilities ou CORBADomains
  - Services dependents du Domaine, pour des domaines specifiques d'application.
  - *Finance, Commerce Electronique, Medical, Aeronautique..*
- **Applications**
  - Ou encore Application Interfaces ou Application Services
  - services developpsé spécifiquement pour une application donnée.

Les principaux CORBA Services

Service	Description
Concurrency	Facilité pour autoriser/bloquer l'accès concurrent à des objets partagés
Transaction	transactions sur les appels de méthodes sur plsuieurs objets
Event	Facilité pour des communication asynchrone à travers des événements
Externalization	Facilité pour marshaliser et démarshaliser des objets
Life cycle	Facilité pour création, suppression, copie, et déplacement d'objets
Licensing	Facilité pour attacher une licence à un objet
Naming	Facilité pour nommage global d'objets (annuaire)
Property	Facilité pour associer des paires « attribut, valeur » avec objets
Trading	Facilité pour publier et trouver les services qu'un objet peut offrir (pages jaune)
Persistence	Facilité pour sauvegarder sur disque des objets
Security	Mécanismes pour tunnels sécurisés, autorisations
Time	Fournit l'heure courante avec report de fuseaux horaires

On voit le naming plus en détail plus loin dans le cours..

CORBA

Partie 3/5 - Le langage IDL

IDL: définition d'interface d'objet

- Ce langage est utilisé:
  - comme langage universel entre client et serveur
  - pour décrire les interfaces des objets
- Un langage pour exprimer les functionalités et l'utilisation d'un objet
  - Pour le client, l'interface represente une "promesse"
  - pour le createur d'implementation, elle represente une "obligation"
- Un fichier IDL est un "contrat"
- IDL est independant des langages L3G

IDL: définitions

- Une langage de spécification
  - modulaire
  - fortement typé
  - orienté objet (héritage multiple, polymorphisme)
- Pas un langage de programmation même si
  - lexicalement proche du C / C++ (avec constructions supplémentaires)
  - Vérification lexicale, syntaxique et sémantique réalisées par un compilateur
- avec macros et preprocesseur ANSI C++
  - inclusion de fichiers : #include
  - définition de macros : #define, #ifdef, #endif ...

IDL : éléments

- Même règles lexicales que C++ pour : identificateurs, mots clés, literaux (constantes), operateurs, separateurs
- commentaires: // and /* … */
- Nouveaux mots clés :
  - attribute
  - interface
  - module
  - oneway
  - readonly
  - sequence
  - any
- Les types de base on une taille normalisé
- Types utilisateurs (typedef, enum, struct, union, array, sequence)
- exceptions
- interface (avec héritage multiple sans surcharge)
- types de méta-données (TypeCode, any)
- Mot clé const pour définir des constantes

IDL: Noms et portée

- Un fichier IDL forme un bloc de nom
- Nom complet: identifier::identifier...
- Les mots clés induisants des portées imbriquées:
  - module, interface, structure, union, operation, exception
- *Module CORBA (dans orb.idl)*
  - Les noms definis par les spec CORBA sont dans le module CORBA
  - en particulier les exceptions std (ci-après)
  - Contient en particulier l'interface ORB

IDL: types

- Types de base (Format binaire normalisé) :
  - void — vide
  - short — entier 16 bits
    unsigned short — entier 16 bits non signé
    long — entier 32 bits
    unsigned long — entier 32 bits non signé
    long long — entier 64 bits
    unsigned long long — entier 64 bits non signé
  - float — réel 32 bits (IEEE)
    double — réel 64 bits (IEEE)
    long double — réel 128 bits (IEEE)
  - boolean — booléen
  - octet — opaque 8 bits
  - char — caractère 8 bits
    wchar — caractère unicode (w=wide)
- Les chaînes de caractères
  - non bornées : string ou wstring (w=wide)
  - bornées : string<80> ou wstring<80>
- *On ne parlera pas du type Any*

```
module Exemple{

  const boolean  vrai = TRUE;
  const char     unChar = 'A';
  const short    unShort = -1;
  const unsigned short etc = 15907;
  const long     unLong = -12345;
  const unsigned long etc2 = 981008;
  const float    unFloat = 3.14159;
  double unDouble = 2.718281828459045;
  const string   unString = "Java";
}
```

valeur jamais convertie

IDL: Litéraux constants

- Entiers
  - “douze” peut etre ecrit: 12, 014, 0XC
- Charactères
  - Simple quotes (‘x’), escape (‘\b’, ‘\uhhh’), caractère etendu(=wide) (L’X’)
- Chaine
  - double quotes (“hello”), wide string (L”me”)
- Flottants
  - On peut écrire: 3.14, 3F
- booleens: valeurs TRUE, FALSE (majuscules)

Exceptions

- Exceptions utilisateur
  - Format: exception ex {};
  - Les méthodes peuvent lancer ces exceptions
- Exemple:

```
interface Inutile {
    exception CompteVide {
          string raison;
    };
};
```

- Il existe des exceptions standard
  - Définies dans le module CORBA
  - intègrent un “minor code” et un “completion status”

- principales exceptions standard:
  - CORBA::COMM_FAILURE — pb de communication
  - CORBA::INV_OBJREF — référence d’objet invalide
  - CORBA::INTERNAL — erreur interne au bus CORBA
  - CORBA::MARSHAL — erreur d’emballage/déballage
  - CORBA::OBJECT_NOT_EXIST — objet n’existe pas ou plus
  - *Il en existe beaucoup plus !*

IDL: Déclaration de méthode

- valeur de retour (void possible)
- modes de passages des parametres:
  - in (passé du client vers le serveur)
  - out (passé du serveur vers le client)
  - inout (passé dans les 2 directions)
- Indicateur de réponse inutile *(rien n'est retourné par le serveur)*:
  - Oneway
- Indicateur d'exception : raises

```
interface Inutile {
     exception CompteVide {
           string raison;
     };
     void retrait(in float amount) raises (CompteVide);
 };
```

- Attribut *"oneway" (rien n'est retourné du tout)*
- Exemples:

```
interface Inutile {
    void Rien(in string valeur);
    oneway void Shutdown(in short exitCode);
 };
```

IDL: Types utilisateurs

- Ecriture comme en C:
  - Typedef
  - Enum
  - Struct
  - Tableaux
    - Il faut passer par un typedef pour pouvoir utiliser un tableau comme parametre
    - Plusieurs dimensions possible!
- Ecriture différente:
  - Union
    - Le type specifié apres le mot « switch »peut etre integer, char, boolean ou enum
- Nouveauté:
  - Sequence
    - Sequence de n'importe quel type IDL. Equivaut à un tableau à une dimension
    - Il faut passer par un typedef pour pouvoir utiliser une séquence comme parametre

```
typedef float Montant;
typedef sequence<string> SeqDeNom;


enum FormatDeDate { chaine, numerique, structure };

 struct DateStructure {
 short Jour;
 short Mois;
 short An;
 };

 union UnionDate switch (FormatDeDate) {
      case chaine :   string stringFormat;
      case numerique: long digitalFormat;
      default:        DateStructure
           structFormat;
 };

sequence<DateStructure> dates;

//maximum 50 elements
sequence< Montant, 50> comptes;
comptes.length(10); // changer lataille à la volée

//tableau
Montant montants[30];
Montant cube[3][3][3];


//utiliser promo comme parametre de méthode
typedef String[100] Promo;
```

Noter que les sequence sont toujours plus souples que les tableaux qui ont toujours une taille fixe

CORBA

Partie 4/5 - Mise en œuvre en C++
(java sera vu en TP)

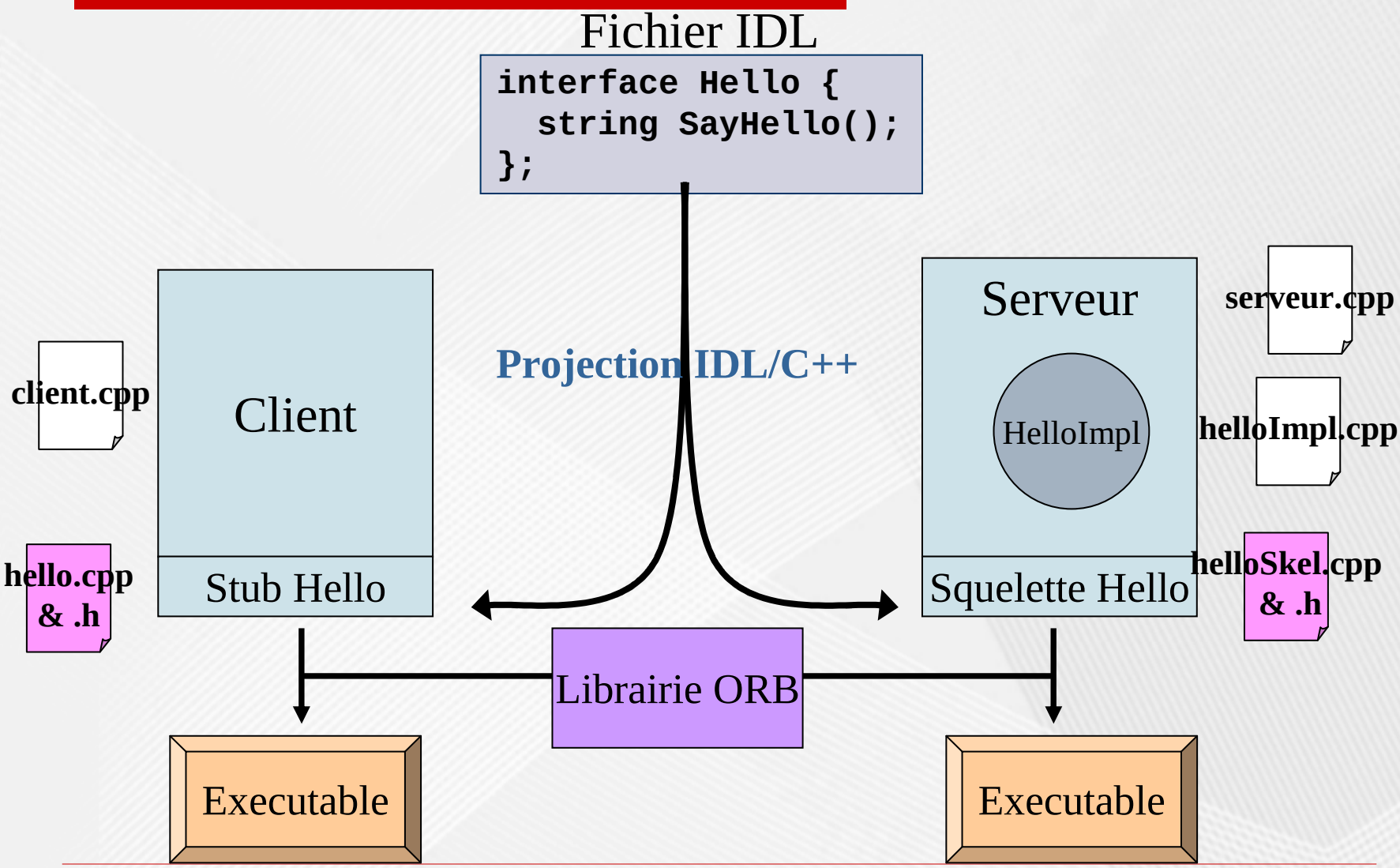
L'ORB Orbacus

- Produit et supporté par Iona
- Orbacus 4.2 disponible depuis le 9 juin 2004
- Compatible CORBA 2.5 et 3.0.x
- URL: http://www.orbacus.com
- gratuit pour utilisation non commerciale
- les sources sont entièrement disponibles
- Mais il faut obligatoirement le recompiler!
- Version « pro » payante: Orbix
- Compatible avec Windows et Linux, ainsi que d'autres unix
- Facile à utiliser: pas de démon
- Fournis JThreads/C++, pour remettre à niveau C++ par rapport à java
- Fournis:
  - Des CORBA services (name,event,property etc..)
  - Un POA
  - Implementation Repository
  - Interface Repository

CORBA: étape de mise en oeuvre

- 1) Définition du contrat IDL: définir une interface pour un Objet CORBA dans un fichier IDL.
- 2) Projection du fichier IDL: cela générera les fichiers spécifiques au langage de programmation ciblé (stub et skeleton)
- 3) Créer une classe implémentant cette interface généré par le compilateur IDL (un servant)
- 4) créer un programme principal serveur utilisant la classe d'implementation. Il initialise l'ORB et l'informe de la création d'objets Corba
- 5) Compiler tout ce code (généré et écrit) avec le compilateur du langage de programmation ciblé (C++, java..)
- 6) Créer un programme client accédant à ces objets corba du serveur via le «naming» (annuaire) ou leur identifiant (IOR)

Mise en œuvre: exemple



Liaison entre le client et le serveur ?

- comment le client retrouve l'objet servant du serveur ?
  - Via une référence d'objet CORBA (IOR)

- Rappel sur la référence d'objet CORBA
  - C'est une clé unique qui se compose de:
    - nom de l'interface OMG IDL
    - protocole de communication (IIOP)
    - Adresse du serveur (machine et port IP)
    - clé unique pour retrouver l'objet dans le serveur

- comment le client obtient-t-il la référence de l'objet CORBA ?
  - Via Stdout !
  - Via un fichier
  - via le service d'annuaire

EXEMPLE d'IOR:

IOR:010000002a00000049444c3a6f6f632e636f6d2f436f734e616d696e672f4f424e616d696e67436f6e746578743a312e
3000610001000000000000000700000000010102000e00000038322e36362e3139322e3137350034072800000abacab305
f526f6f74504f4100526f6f74436f6e74657874504f4100004e616d655365727669636501000000010000002000000001fe
120001000100020000002000010001000105000101000100000009010100

L'interface IDL CORBA::Object

```
module CORBA {
    interface Object {
        // Teste si une référence ne pointe sur aucun objet.
        boolean _is_nil ();
        // Teste si un objet n'existe plus.
        boolean _non_existent();
        // Teste si 2 références désignent le même objet.
        boolean _is_equivalent(in Object obj);
        // Calcule une clé de hachage.
        long _hash (in long maximum);
        // et bien d'autres opérations . . .
    };
};
```

→ **Implicitement héritée par tout objet CORBA**

Accéder à l'ORB

- L'ORB représente la passerelle vers le bus CORBA pour un programme
- Le code implémentant la communication avec le bus est linké via une librairie
  - Avec omniORB :omniORB304_rt.lib
  - Avec Orbacus: ob.lib
- A l'exécution, l'ORB est défini comme
  - un objet local (pas d'IOR)
  - l'interface CORBA::ORB
- Il permet de retrouver le rootPOA

L'interface CORBA::ORB

```
module CORBA {
    //Objet local représentant le bus
    interface ORB {
        //représentation interne vers chaîne IOR
        string Object_to_string (in Object obj);

        //chaîne IOR vers représentation interne
        Object string_to_Object (in string str);

        // et bien d'autres opérations . . .
    };
    // Initialiser et obtenir localement l'ORB.
    // On lui donne : argc, argv, "omniORB3"
    ORB ORB_init ( . . . );
};
```

L'interface CORBA::POA

```
module Portableserver {
        typedef sequence<octet> ObjectId;

        interface POA {
                //obtenir l'unique instance du POA manager pour pouvoir l'activer
                POAManager the_POAManager();

                // activer une instance d'une implémentation
                ObjectId activate_Object(Object o);
        };

        interface POAManager {
                void activate();
        };
};

module CORBA {
        interface ORB {
                ...
                //recupérer une reference sur le POA root – il faut donner "RootPOA"
                Object resolve_initial_references(string name);
                ...
        };
};
```

Corba: un exemple

- Un « Hello World » distribué :
  - invocation distante de la méthode sayHello() d'un objet Corba.
  - sayHello() renvoie une chaine

- 4 sources sont nécessaires :
  - Hello.idl : l'interface décrivant l'objet corba
  - HelloImpl.h & HelloImpl.cpp : l'implémentation de l'objet Corba
  - HelloServer.cpp : une application principale (main) pour le serveur
  - HelloClient.cpp : l'application cliente utilisant l'objet Corba à distance

Hello.idl

```
interface Hello{
    string sayHello();
};
```

Le stub Hello (orbacus)

```
class Hello;
typedef Hello* Hello_ptr;
typedef OB::ObjVar< Hello > Hello_var;

class Hello {
     public:
     static Hello_ptr _narrow(::CORBA::Object_ptr);
     static _ptr_type _nil();
     static inline Hello_ptr duplicate(Hello_ptr p);

    //
    // IDL:Hello/sayHello:1.0
    //
    virtual char* sayHello() = 0;

     ... partie spécifique à l'ORB ...
};
```

Le reste est spécifique à l'ORB, mais quelquesoit l'ORB on va trouver une classe fille héritant de CORBA::Object:

```
class OBProxy_Hello : virtual public ::Hello,
                      virtual public OBCORBA::Object
 {
     public:
     char* sayHello();
};
```

le Squeleton Hello (orbacus)

```
typedef _objref_Hello* Hello_ptr;


//POA_Hello utilisé coté serveur
class POA_Hello: : virtual public PortableServer::ServantBase {
    public:
    inline Hello_ptr _this();

    //
    // IDL:Hello/sayHello:1.0
    //
    virtual char* sayHello()
        throw(::CORBA::SystemException) = 0;
};
```

L’implantation C++ de Hello

```
// Implémentation par héritage du squelette IDL.
class hello_Impl : public POA_Hello,
        public PortableServer::RefCountServantBase {

protected:
    int count;
public:
    inline hello_Impl() {count=0;}
    virtual ~hello_Impl() {}
    virtual char* sayHello();
};

// Implémetation de l'opération IDL sayHello()
char* hello_Impl::sayHello() throw(CORBA::SystemException) {
    cout << "Appel de sayHello() #" << (count+1) << endl;

    ostringstream s;
    s<<"Hello World #"<< ++count << " !";

    return CORBA::string_dup((const char*)s.str().c_str());
}
```

Vous pouvez ecrire du code “C” (en utilisant sprintf par exemple)

Programme principal du Serveur C++

```
//1- Initialise le bus CORBA
CORBA::ORB_var orb = CORBA::ORB_init (argc, argv);

//2- Retrouve le Root POA
CORBA::Object_var obj = orb->resolve_initial_references("RootPOA");
PortableServer::POA_var poa = PortableServer::POA::_narrow (obj);

//3- recupere une reference sur le POA manager
PortableServer::POAManager_var manager= poa->the_POAManager();

hello_Impl* myhello = new hello_Impl(orb, poa);

//4- enregistrer l'instance servant auprès du POA manager
PortableServer::ObjectId_var myhelloid = poa->activate_Object(myhello);

//5- publier la référence d'objet
// recupere une reference de l'Objet et l'affiche en IOR stringifiée
obj = myhello->_this();
CORBA::String_var sior(orb->Object_to_string(obj));
cout << "'" << (char*)sior << "'" << endl;

// Executer l'implementation
manager->activate();

//6- Se placer en attente de requetes du bus (appel bloquant)
orb->run();
```

Classes _var permettent la libération automatique de la mémoire

Programme principal du Client C++

```
//1- Initialise le bus CORBA
CORBA::ORB_var orb = CORBA::ORB_init (argc, argv);

//2- Recupere l'objet "hello" sous forme de stub référençant l'objet CORBA.
har* IOR_en_dur = "IOR:01fa12000e00000049444c3a… ";
CORBA::Object_var obj = orb->string_to_Object(IOR_en_dur /*argv[1]*/ );

// Conversion vers une référence Hello.
// "helloref" est la souche pour referencer l'objet distant
Hello_var helloref = Hello::_narrow (obj);

if ( CORBA::is_nil(helloref) ) {
     cout << "Erreur a la conversion de reference vers le type Hello" << endl;
     return 1;
}

//3- appel
CORBA::String_var dest = helloref->sayHello();

cout << "Message de l'objet corba Hello: \"" << (char*)dest <<"\"." << endl;
```

Compilation et exécution en C++

- Génération des souches/squelettes C++
  - <ORB home>/bin/idl Hello.idl
- Compilation C++ des sources applicatifs et générés
  - unix: gcc -o HelloSKel.o -c HelloSKel.cc
  - windows: cl -c HelloSKel.cpp
  - Idem pour les autres sources
- Link avec les librairies de l'ORB
  - Unix: gcc -o HelloServer HelloServer.o HelloSKel.o <librairies ORB>
  - Windows: link /out: HelloServer HelloServer.obj HelloSKel.obj <librairies ORB>
- Idem pour le client
  - gcc -o HelloClient HelloClient.o <bibliothèques ORB>
- Exécution des programmes :
- > ./HelloServeur
- > ./HelloClient IOR: . . .

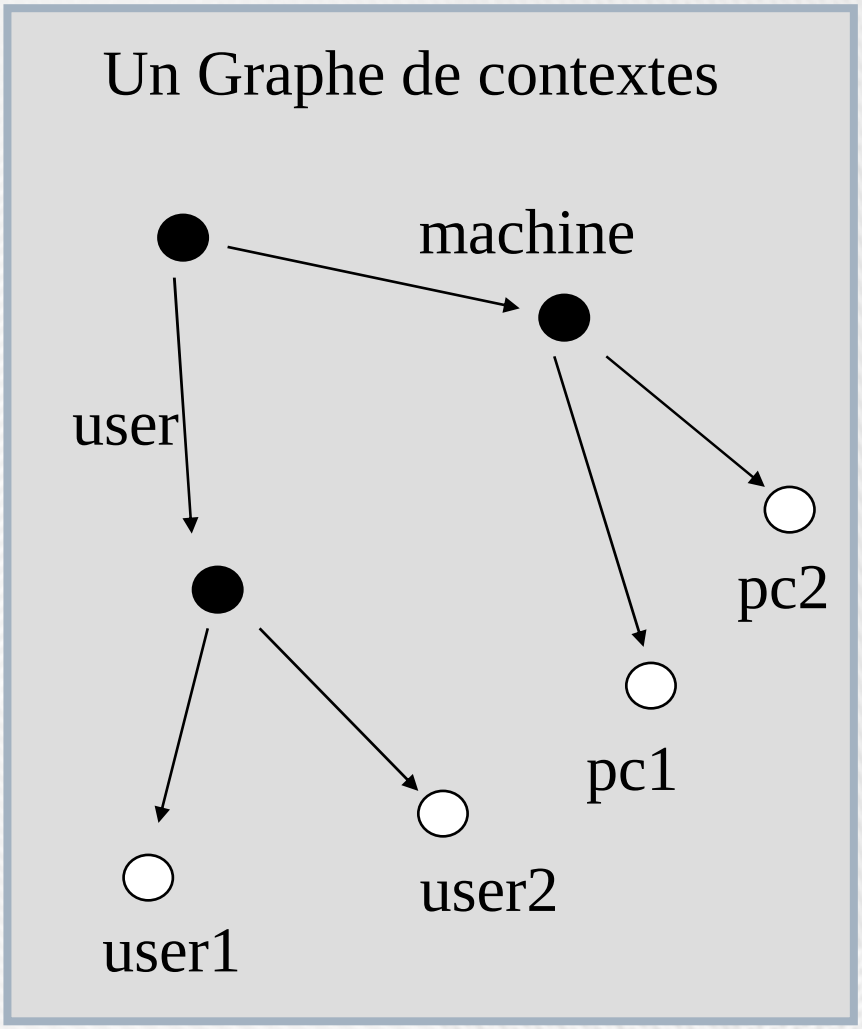
CORBA

Partie 5/5 - Le service de nom (Naming Service)

Le service de nom

- Service *Pages blanches*
- Nommé aussi "COSnaming" (COS= Common Object Services)
- Resolution: nom => objet
- **"Naming context":** groupe de noms (uniques dans le groupe)
- Les contextes peuvent être reliés à d'autres contexee
- Graphe de Nom: graphe complet dans lequel les noeuds sont des contextes

- (En clair un "context" est un noeud, une feuille sera un "object"/une feuille-voir le type `BindingType` plus loin)



Interface du Naming Service

- Publication pour les noms (les objets “feuilles”):
  - bind(Name, Object)
  - rebind(Name, Object)
- Publication pour les contextes de noms (les noeuds):
  - bind_context(Name, NamingContext)
  - rebind_context(Name,NamingContext)
  - NamingContext new_context()
  - NamingContext bind_new_context(Name)
- Recherche:
  - Object resolve(Name)
- Libération:
  - destroy() → à appeler sur un contexte de nom avant le unbind
  - unbind(Name)
- Lister les noms disponibles:
  - list(long, out BindingList, out BindingIterator) →
  - *Exemple un peu plus loin*

BindingIterator
- *boolean next_one(out Binding)*
- *boolean next_n(long, out BindingList)*
- *destroy()*

Noms dans le service

- Definition IDL:

```
struct NameComponent {
        string id;
        string kind;
   };
typedef sequence <NameComponent> Name;
```

Peut être comparé au nom et extension pour un fichier

```
enum BindingType {nObject, ncontext};
struct Binding {
        Name        binding_name;
        BindingType binding_type;
   };
```

Ceci sert surtout pour lister les noms disponibles (Voir plus loin)

Exemple de nom:

```
NameComponent[] Nom= new NameComponent[1];
Nom[0] = new NameComponent();
Nom[0].id = “mickey”;
Nom[0].kind = “”;
```

Un programme client typique

```
// Initialisation de l'ORB
CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);

// récupération du naming service.
CORBA::Object_var  obj = orb -> resolve_initial_references("NameService");
CosNaming::NamingContext_var nc = CosNaming::NamingContext::_narrow(obj.in());

if ( CORBA::is_nil(obj.in()) ) {
    cout << "'NameService' introuvable" << endl;
    return 1;
}

// Résolution d'un nom
CosNaming::Name aName;
aName.length(1);
aName[0].id = CORBA::string_dup("Hello");
aName[0].kind = CORBA::string_dup("");

CORBA::Object_var aObj = nc -> resolve(aName);
Hello_var helloref = Hello::_narrow(aObj.in());

// appel d'une methode de l'objet
CORBA::String_var dest = helloref->sayHello();
cout << "Message de l'objet corba Hello: \"" << (char*)dest <<"\"." << endl;
```

Un programme serveur typique

```
// Initialisation de l'ORB
CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);

// Retrouve le Root POA
CORBA::Object_var poaObj = orb -> resolve_initial_references("RootPOA");
PortableServer::POA_var rootPOA = PortableServer::POA::_narrow(poaObj.in());

// recupere une reference sur le POA manager
PortableServer::POAManager_var manager = rootPOA -> the_POAManager();

// creation d'implementations
Hello_impl* a1p = new Hello_impl;
Hello_var a1 = a1p -> _this();

// récupération du naming service.
CORBA::Object_var  obj = orb -> resolve_initial_references("NameService");
CosNaming::NamingContext_var nc = CosNaming::NamingContext::_narrow(obj);

// creation et publication du nom
CosNaming::Name a1Name;
a1Name.length(1);
a1Name[0].id = CORBA::string_dup("Hello");
a1Name[0].kind = CORBA::string_dup("");
nc -> bind(a1Name, a1);

//lancer l'implementation.
manager -> activate(); //  POA manager
orb -> run();
```

Lister les noms publiés

```
void AfficherUnNom(const CosNaming::Binding& b){
    CORBA::ULong i;
    // afficher le label et le type de nom
    for(i = 0 ; i < b.binding_name.length() ; i++)
        cout << b.binding_name[i].id;
   switch(b.binding_type)    {
        case CosNaming::nobject: cout << " (objet)" << endl; break;
        case CosNaming::ncontext: cout << " (contexte)" << endl;
    break;
    }
}

//dans le programme principal...

cout << "Affichage de tous les noms publiés:" << endl;
CosNaming::BindingList_var bl;
CosNaming::BindingIterator_var bi;
nc -> list(999999, bl.out(), bi.out());
for(i = 0 ; i < bl -> length() ; i++)
   AfficherUnNom(bl[i]);
```

nc est un CosNaming::NamingContext_var

Démarrer le name service avec ORBacus

- Demarrer le service:

```
nameserv.exe  --ior -IIOPport 1973
```

- Lancer un programme lié au nameservice:

```
server.exe  -ORBInitRef NameService=corbaloc:iiop:localhost:1973/NameService

client.exe  -ORBInitRef
  NameService=corbaloc:iiop:localhost:1973/NameService
```

- On utilise pour cela un des paramètres spécifiques ORB de la ligne de commande: ORBInitRef
  - format: `-ORBInitRef <ObjectID>=<ObjectURL>`
  - Autre exemple:

```
-ORBInitRef NameService=IOR:00230021...
```

CORBA

References

Références

- [CORBA FAQ]
  http://www.omg.org/gettingstarted/corbafaq.htm
- [CORBA-Overview]
  http://www.cs.wustl.edu/~schmidt/corba-overview.html
- [CORBA/IIOP 2.2 Specification]
  http://www.omg.org/library/specindx.html
  - Chapitre 3: OMG IDL Syntax and Semantics
  - Chapitre 24: Mapping OMG IDL to Java
- [Introduction to CORBA IDL]
  http://www.iona.com/support/docs/manuals/orbix/33/html/orbix33cxx_pguide/IDL.html
- [JacORB Performance compared]
  - http://www.jacorb.org/performance/
- [ORB Core Feature Matrix]
  - http://www.jetpen.com/~ben/corba/orbmatrix.html
- [CORBA : des concepts à la pratique]
  - http://corbaweb.lifl.fr