Bases de données et environnements distribués
Chapitre III : CORBA

Éric Leclercq

Département IEM / Laboratoire LE2i

Octobre 2016

émail : Eric.Leclercq@u-bourgogne.fr
http://ludique.u-bourgogne.fr/~leclercq
http://ufrsciencestech.u-bourgogne.fr

Plan

- La vision globale de l'OMG
- L'architecture objets distribués : le modèle d'abstraction
- La construction d'application réparties avec CORBA (services)
- Le langage IDL
- Illustration des composants, constituants du bus CORBA
- Les mécanismes dynamiques de CORBA seront abordés à la fin de ce cours

Bibliographie CORBA

- Au coeur de CORBA avec Java, Jérôme Daniel, Vuibert, 2000, 464p., ISBN 2-7117-8659-5
- Client/Server Programming with Java and CORBA, Dan Harkey, Robert Orfali, 1072p., Wiley, 1998, ISBN 047124578X
- CORBA des concepts à la pratique, 2ème édition, Jean-Marc Geib, Christophe Gransart, Philippe Merle, InterEditions, 1999, 360p., ISBN 2-10-004806-6
- Objets répartis Guide de survie, Jeri Edwards, D. Harkey, Robert Orfali, Vuibert, 1996, 656p., ISBN 2-84180-043-1
- CORBA fundamentals and programming, Wiley, 1996, 720p., ISBN 0-471-12148-7

Vision globale de la construction d'applications réparties

- L'Object Management Group (OMG) : consortium international créé en 1989
- Buts :
  - Développer des spécifications d'architectures distribuées et ouvertes
  - Regroupe plus de 850 acteurs du monde informatique :
    - des constructeurs (HP, IBM, Sun, etc.)
    - des producteurs de logiciel (Netscape, Inprise ex-Borland/Visigenic, IONA, etc.)
    - des utilisateurs (Boeing, Alcatel, etc.)
    - des institutions et des universités
  - Faire émerger des standards pour l'intégration d'applications distribuées hétérogènes à partir des technologies orientées objet

Vision globale de la construction d'applications réparties

Les propriétés fondamentales visées par les travaux de l'OMG sont :

- La réutilisabilité
- L'interopérabilité
- La portabilité de objets logiciels

L'élément clé de la solution proposée est CORBA (*Common Object Request Broker Architecture*) :

- Middleware orienté objet (*ORB Object Request Broker*)
- Plus précisément c'est un bus d'objets répartis :
  - il offre un support d'exécution masquant les couches bases d'un système réparti (système d'exploitation, processeur et réseau)
  - il prend en charge les communications entre les objets logiciels formant les applications réparties hétérogènes
- Un ensemble de services

Le modèle d'interaction d'objets CORBA

CORBA propose un modèle de coopération :

- Client serveur
- Abstrait
- Orienté objet

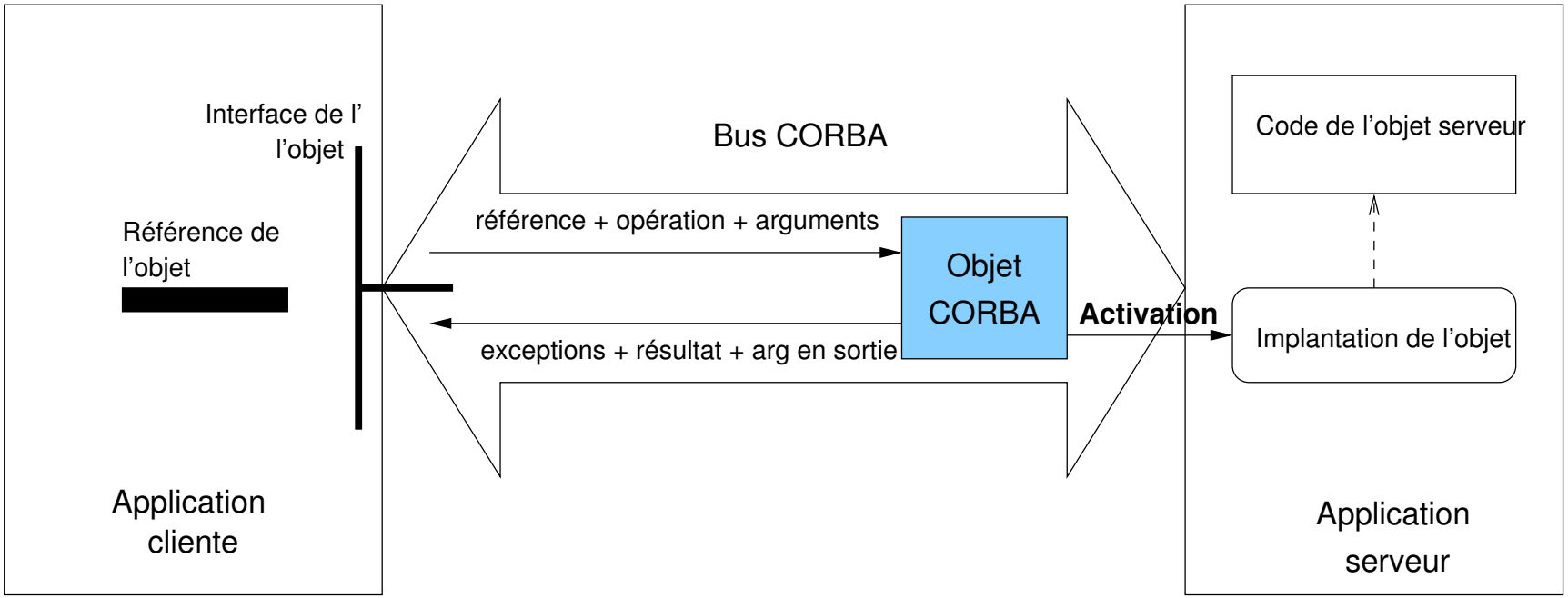
Chaque application peut exporter :

- certaines de ses fonctionnalités (services)
- sous la forme d'objets CORBA.
- l'object CORBA est la composante d'abstraction ou structuration du modèle
- l'objet CORBA est virtuel

Les interactions entre applications se font via :

- invocations à distance des méthodes des objets CORBA
- Ainsi, la notion client/serveur intervient uniquement lors de l'utilisation d'un objet (l'application implantant l'objet est le serveur, l'application utilisant l'objet est le client).

Le modèle d'interaction d'objets CORBA



Le modèle d'interaction d'objets CORBA

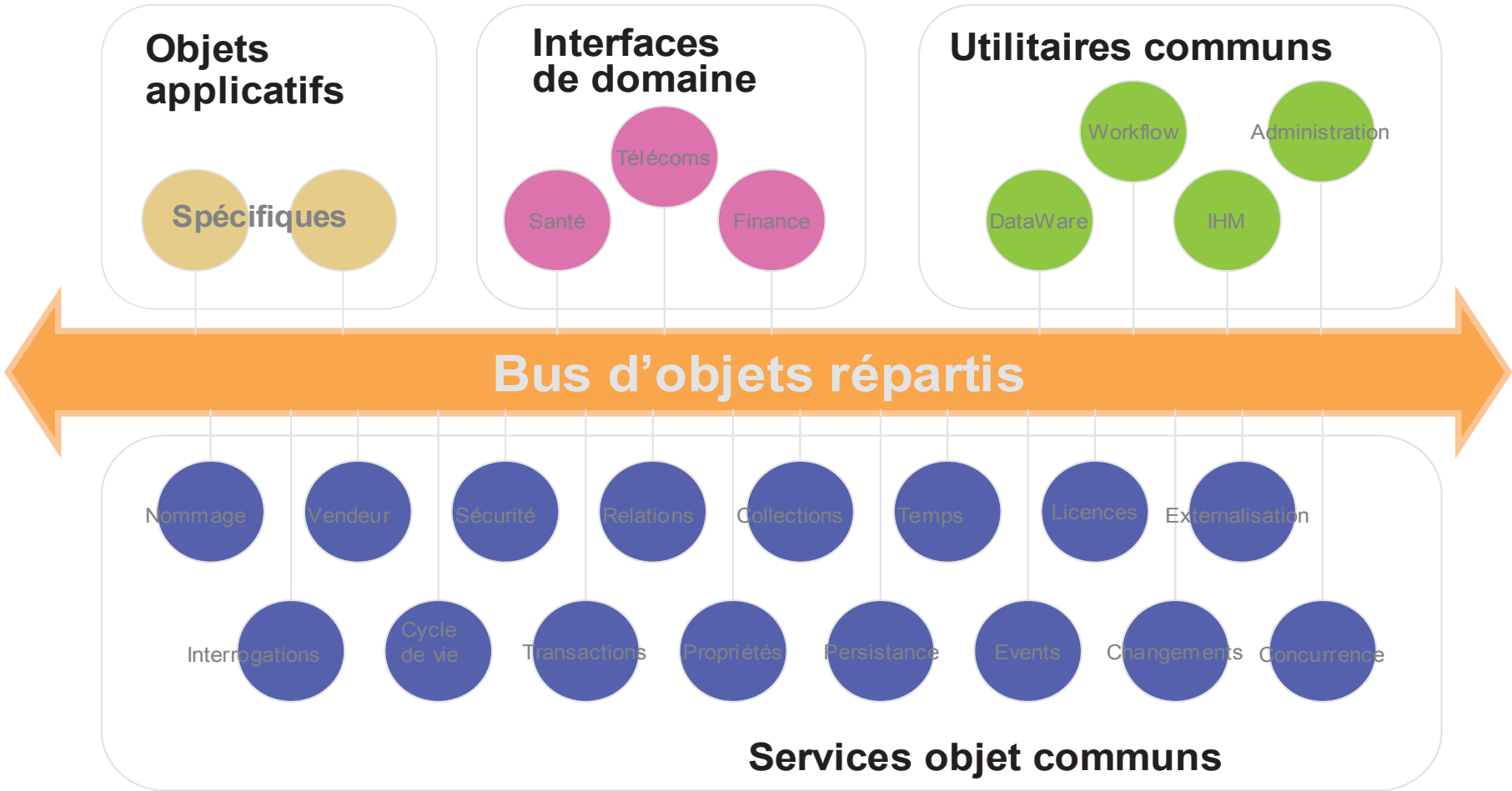
Quelques définitions :

- Application cliente : programme qui invoque les méthodes des objets à travers le bus CORBA
- Référence d'objet : structure désignant l'objet CORBA et contenant l'information nécessaire pour le localiser sur le bus (et bien entendu sur le réseau)
- Interface de l'objet : type abstrait spécifiant l'objet par ses opérations et attributs (se définit au moyen langage IDL)
- Requête : le mécanisme d'invocation d'une opération ou d'un accès à un attribut de l'objet
- Bus CORBA : composant ou couche logicielle qui achemine les requêtes de l'application cliente vers l'objet serveur en masquant tous les problèmes d'hétérogénéité (langages, systèmes d'exploitation, matériels, réseaux)

Le modèle d'interaction d'objets CORBA

- Objet CORBA : objet logiciel cible de l'appel de méthode (entité virtuelle gérée par le bus CORBA)
- Activation : processus permettant l'association d'un objet d'implantation (réel) à un objet CORBA
- Implantation de l'objet : code de l'objet CORBA à un instant donné, gère un état de l'objet temporaire. Au cours du temps, un même objet CORBA peut être associé des implantations différentes
- Code d'implantation : regroupe les traitements associés à l'implantation des opérations de l'objet CORBA. Par exemple une classe Java ou un ensemble de fonctions C

L'architecture globale



L'architecture globale

- L'OMG a proposé des spécifications afin de préciser sa vision globale de la construction d'applications réparties (Object Management Architecture Guide 1995).
- Ce document vise à classifier les différents objets qui interviennent dans une application en fonction de leurs rôles :
  - Le bus objets répartis : assure le transport des requêtes entre tous les objets CORBA offre un environnement d'exécution masquant l'hétérogénéité liée aux langages de programmation, aux systèmes d'exploitation, aux processeurs et aux protocoles réseaux
  - Les services objet communs (CORBAservices) : fournissent sous forme d'objets CORBA, les fonctions systèmes nécessaires à la plupart des applications réparties
    - les annuaires,
    - le cycle de vie des objets,
    - les relations entre objets,
    - les événements, les transactions, la sécurité, la persistance, etc.

L'architecture globale

- Les utilitaires communs (CORBAfacilities) : ce sont des canevas d'objets (*Frameworks*) qui répondent plus particulièrement aux besoins des utilisateurs. Ils standardisent l'interface utilisateur, l'administration, le Workflow, etc.
- Les interfaces de domaine (Domain Interfaces) : définissent des objets de métiers spécifiques à des secteurs d'activités (finance, santé (dossier médical) ou télécoms). Leur objectif est d'assurer l'interopérabilité sémantique entre les systèmes d'informations d'entreprises d'un même métier (Business Object Frameworks BOF).
- Les objets applicatifs (Application Objects) : spécifiques à une application répartie

Les services communs (SC)

Objectif : standardiser les interfaces des fonctions système indispensables à la construction et l'exécution de la majorité des applications réparties

- La recherche d'objets : offrir des mécanismes pour rechercher dynamiquement les objets sur le bus (équivalents des annuaires téléphoniques)
  - Nommage (Naming Service) est l'équivalent des "pages blanches" : les objets sont désignés par des noms symboliques. Cet annuaire est matérialisé par un graphe de répertoires de désignation
  - Vendeur (Trader Service) est l'équivalent des "pages jaunes" : les objets peuvent être recherchés en fonction de leurs caractéristiques/fonctionnalités

Les services communs (SC)

Cycle de Vie (Life Cycle Service) : décrit des interfaces pour la création, la copie, le déplacement et la destruction des objets sur le bus Il définit pour cela la notion de fabrique d'objets *Object Factory*

- Propriétés (Property Service) : permet aux utilisateurs d'associer dynamiquement des valeurs nommées à des objets Ces propriétés ne modifient pas l'interface IDL, mais représentent des besoins spécifiques au client (annotations ou méta-données par exemple)
- Relations (Relationship Service) : sert à gérer des associations dynamiques (appartenance, inclusion, référence, auteur, etc.) reliant des objets sur le bus

Les services communs (SC)

- Externalisation (Externalization Service) : mécanisme standard pour fixer ou extraire des objets du bus
- Persistance (Persistent Object Service) : stocker des objets sur un support persistant Quel que soit le support utilisé, ce service s'utilise de la même manière via un "Persistent Object Manager" Un objet persistant doit hériter de l'interface *Persistent Object* et d'un mécanisme d'externalisation
- Interrogations (Query Service) : permet d'interroger les attributs des objets Repose sur les langages standards d'interrogation (SQL3 ou OQL). L'interface Query permet de manipuler les requêtes comme des objets CORBA. Les objets résultats sont enregistrés dans une collection
- Collections (Collection Service) : permet de manipuler d'une manière uniforme des objets sous la forme de collections et d'itérateurs

Les services communs (SC)

- Changements (Versionning Service) : Permet de gérer et de suivre l'évolution des différentes versions des objets Maintient des informations sur les évolutions des interfaces et des implantations Pas encore complètement spécifié officiellement ?
- Sécurité (Security Service) : permet d'authentifier les clients, de chiffrer et de certifier les communications Minimum = IIOP sur Secure Socket Layer
- Transactions (Object Transaction Service) : assure l'exécution de traitements transactionnels impliquant des objets distribués et des bases de données en fournissant les propriétés ACID (atomicité, consistance, isolation, durabilité)
- Concurrence (Concurrency Service) : fournit les mécanismes pour contrôler et ordonnancer les invocations concurrentes sur les objets

(SC) Communications synchrones et asynchrones

- La coopération des objets CORBA à un mode de communication client/serveur synchrone
- Il existe néanmoins des services spécifiques assurant des communications asynchrones
- Événements (Event Service) : permet aux objets de produire des événements asynchrones à destination d'objets consommateurs au travers de canaux d'événements. Les canaux fournissent deux modes de fonctionnement.
  - mode push : le producteur a l'initiative de la production, le consommateur est notifié des événements
  - mode pull : le consommateur demande explicitement les événements, le producteur est alors sollicité

(SC) Communications synchrones et asynchrones

- Notification (Notification Service) : extension du service précédent ; les consommateurs sont uniquement notifiés des événements les intéressant
- Messagerie (CORBA Messaging) : modèle de communication asynchrone qui permet de gérer des requêtes persistantes. Utiles lorsque l'objet appelant et l'objet appelé ne sont pas présents simultanément sur le bus (mobilité ?) Cela permet d'avoir des fonctionnalités proches des MOMs (Message Oriented Middleware)

(SC) Autres services communs

- Temps (Time Service) : fournit des interfaces permettant d'obtenir une horloge globale sur le bus (Universal Time Object), Mesurer le temps et de synchroniser les objets
- Licences (Licensing Service) : permet de contrôler l'utilisation des objets

Les interfaces de domaines (ID)

Parallèlement aux services l'OMG a cherché à définir des canevas d'objets dédiés à des secteurs d'activité (ou métiers) ou des besoins applicatifs transverses à des métiers. Groupes de travail nommés selon les cas : Working Groups, Domain Task Forces ou Special Interest Groups

- Business Objects DTF standardise les méthodologies et les technologies pour la conception d'applications métier
- Workflow Workgroup travaille sur les outils CORBA pour les applications de gestion de flux de travail et les activités coopératives au sein des entreprises
- C4I DSIG fait la promotion de CORBA au sein des organismes et administrations liés à la défense en jouant un rôle pédagogique
- End User SIG travaille en relation avec les utilisateurs finaux à fin de déterminer leurs besoins, de connaître leurs expériences

Les interfaces de domaines (ID)

- CORBAmed pour le domaine de la médecine pour permettre l'interopérabilité entre les acteurs de la santé
- Life Sciences Research DTF groupe dédié à l'utilisation de CORBA par les instituts de recherche dans les sciences de la vie
- Electronic Commerce pour les applications de commerce électronique
- Financial Services DTF domaine de la finance/banque
- Telecom DTF fait la liaison entre le monde CORBA et le monde des télécommunication
- Internet Platform SIG travaille sur les rapprochements des technologies CORBA avec les technologies d'Internet
- Manufacturing DFT tente de combler le fossé entre les technologies OMG et les besoins des industrie
- Realtime SIG réfléchit sur la prise en compte des aspects temps réel au sein du bus CORBA
- Distributed Simulation SIG adresse les problèmes liés à la simulation distribuée et son exécution sur un bus CORBA

Les interfaces de domaines (ID)

- Test SIG doit définir les méthodologies, les outils, les protocoles pour automatiser les tests d'applications réparties CORBA
- Object Model Working Group travaille sur l'unification des modèles orientés objet en vue de créer un standard plus généraliste que celui proposé actuellement par CORBA
- UML Revision Task Force travaille sur le langage UML standardisé par l'OMG (`http://www.omg.org/uml/`)

Vers une indsutrie du composant logiciel et au delà

- Le langage IDL permet d'isoler l'aspect fonctionnel de certains objets de l'implémentation : cette abstraction contribue à l'interopérabilité.
- IDL ouvre un nouveau paradigme celui des services, les objets métier, les objets du monde réel sont dissocié des services.
- Les objets distribués dialogue au moyen du bus CORBA (ORB) et protocole de communicaiton IIOP.
- L'architecture OMA classifie les objets distribués en 4 catégories : services communs, utilitaires, interfaces de domaines, objets applicatifs.
- La gestion des services implémenté sous forme d'OD est réalisée par les utilitaires communs (nommage, cycle de vie, transaction).

Le bus à objets répartis

Le bus CORBA (ORB) est un intermédiaire/négociateur à travers lequel les objets vont pouvoir dialoguer. Il présente les caractéristiques suivantes :

- Liaison avec tous les langages de programmation :
  - en réalité seulement quelques quelques mapping
  - dépendant du vendeur de l'OR B
  - C, C++, SmallTalk, Ada, COBOL, Java, Python, Ruby
- Transparence des invocations : les requêtes aux objets semblent toujours être locales au programme, le bus CORBA se charge de les acheminer
- Invocation statique et dynamique : 2 mécanismes pour soumettre les requêtes aux objets
  - en statique, les invocations sont contrôlées au moyen d'une référence connue
  - en dynamique, les invocations doivent être contrôlées à l'exécution

Le bus à objets répartis

L'interopérabilité entre bus :

- à partir des spécifications CORBA 2.0, définition d'un protocole générique de transport des requêtes (GIOP General Inter-ORB Protocol)
- permet l'interconnexion de bus CORBA provenant de fournisseurs différents
- une de ses instanciations de GIOP est l'Internet Inter-ORB Protocol (IIOP) fonctionnant au dessus de TCP/IP

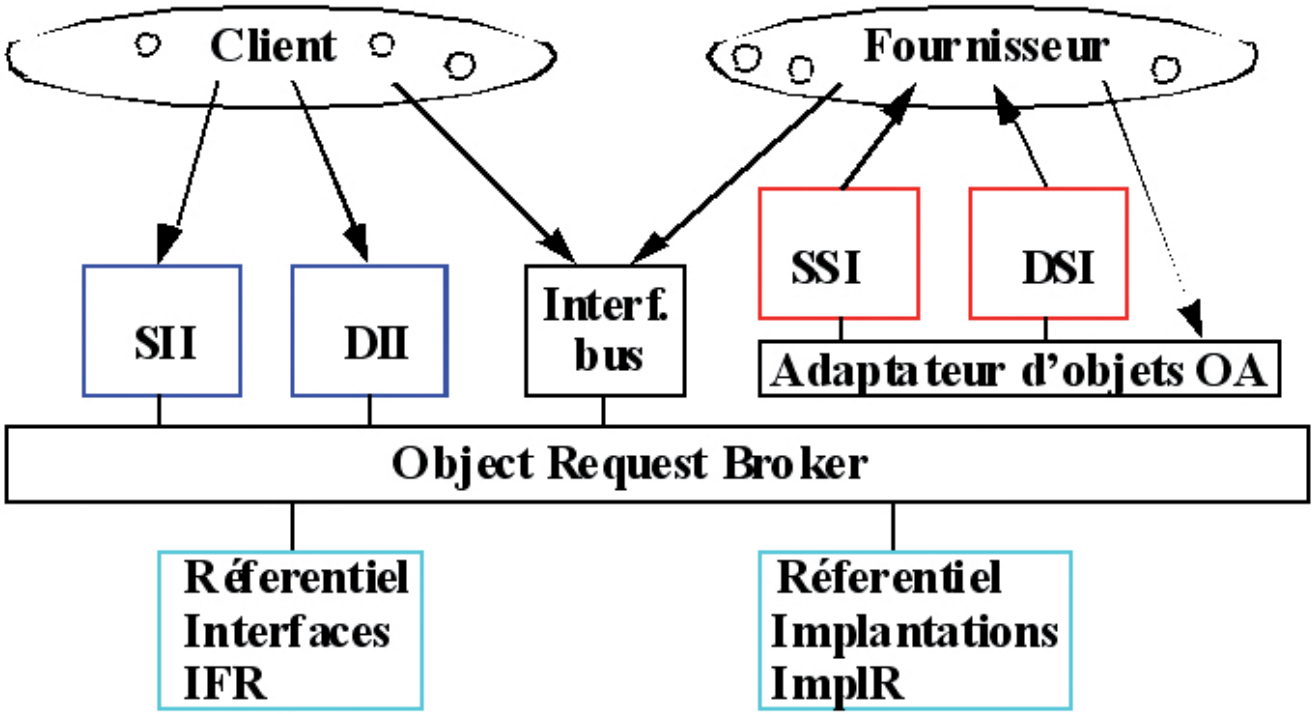
Le protocole GIOP définit :

- une représentation commune des données (CDR ou Common Data Representation)
- un format de références d'objet interopérable (IOR ou Interoperable Object Reference)
- un ensemble de messages de transport des requêtes aux objets (Request, Reply, ...)

Le bus à objets répartis

- Système auto-descriptif : les interfaces des objets sont connues du bus et sont aussi accessibles par les programmes par l'intermédiaire d'un référentiel interfaces
- Activation automatique et transparente des objets : les objets sont en mémoire uniquement s'ils sont utilisés par des applications clientes
- Exemples d'implémentations (libres) :
  - opalORB (perl), Orbit2 (C,C++, Python,...), ILU
  - Orbix, Visibroker, ORBacus

Les composants de l'ORB



Les composants de l'ORB

- ORB (Object Request Broker) : noyau de transport des requêtes Intègre au minimum les protocoles GIOP et IIOP L'interface du bus fournit les primitives de base comme l'initialisation de l'ORB
- SII (Static Invocation Interface) : interface d'invocations statiques Permet de soumettre des requêtes contrôlées à la compilation des programmes
- DII (Dynamic Invocation Interface) : interface d'invocations dynamiques Permet de construire dynamiquement des requêtes vers n'importe quel objet CORBA sans générer/utiliser une interface SII

Les composants de l'ORB

- IFR (Interface Repository) : référentiel des interfaces contient une représentation des interfaces OMG-IDL accessible par les applications durant l'exécution
- SSI (Skeleton Static Interface) : interface de squelettes statiques permet à l'implantation des objets de recevoir les requêtes leur étant destinées générée comme l'interface SII
- DSI (Dynamic Skeleton Interface) : interface de squelettes dynamiques permet d'intercepter dynamiquement toute requête sans générer une interface SSI pendant de DII coté serveur
- OA (Object Adapter) : adaptateur d'objets Crée les objets CORBA, Maintient les associations entre objets CORBA et les implantations Réalise l'activation automatique si nécessaire
- ImplR (Implementation Repository) : référentiel des implantations Contient l'information nécessaire à l'activation Ce référentiel est spécifique à chaque produit CORBA

Les composants de l'ORB

Différentes approches peuvent être envisagées pour construire un bus :

- 1 bus = 1 processus : les objets serveur sont dans le même espace mémoire que les clients. C'est le cas typique des applications embarquées
- 1 bus = 1 OS : les clients et les serveurs sont des processus différents sur la même machine (le bus est une fonctionnalité du noyau - extension des IPC ?)
- 1 bus = 1 ensemble de processus communicants : les processus sont sur des sites différents et les requêtes sont véhiculées à travers le réseau (Internet avec IIOP)
- 1 bus = une bibliothèque d'abstraction des couches de communication

Les composants de l'ORB

Fédération de bus CORBA :

- Plusieurs bus distincts peuvent être reliés pour former une fédération
- Les communications entre ces bus peuvent se faire :
  - soit par le protocole commun IIOP
  - soit à travers des passerelles spécifiques ou génériques comme par exemple : DII / DSI

Les composants de l'ORB

Retour sur les protocoles et le problème des références,...

Les IORs utilisés avec le protocole IIOP doivent contenir :

- le nom complet de l'interface IDL de l'objet
- l'adresse IP de la machine où est localisé l'objet
- le port IP pour se connecter au serveur de l'objet
- une clé pour désigner l'objet dans le serveur : son format est libre et il est différent pour chaque implantation du bus CORBA

Un bus CORBA peut utiliser d'autres protocoles de transport des requêtes aux objets. Il existe des bus multi-protocoles permettant d'avoir simultanément des communications en IIOP, UDP-IOP et Multicast-IOP

Le langage IDL

Le langage OMG-IDL (Interface Definition Language) :

- Permet d'exprimer sous la forme de contrats IDL la coopération entre les fournisseurs et les utilisateurs de services
- Sépare l'interface de l'implantation des objets
- Masque les divers problèmes liés à l'interopérabilité dont l'hétérogénéité des langages
- Un contrat IDL spécifie les types manipulés pour ensemble d'applications réparties : les types d'objets (ou interfaces IDL) et les types de données échangés entre les objets
- Le contrat IDL : isole les clients et fournisseurs de l'infrastructure logicielle et matérielle les met en relation à travers le bus CORBA

Principe d'utilisation d'IDL avec CORBA

Modèle général des objets distribué :

1. Déclaration des services accessibles à distance
2. Écriture du code des services
3. Instanciation et enregistrement de l'objet serveur
4. Interactions du client avec le serveur

Modèle pour CORBA :

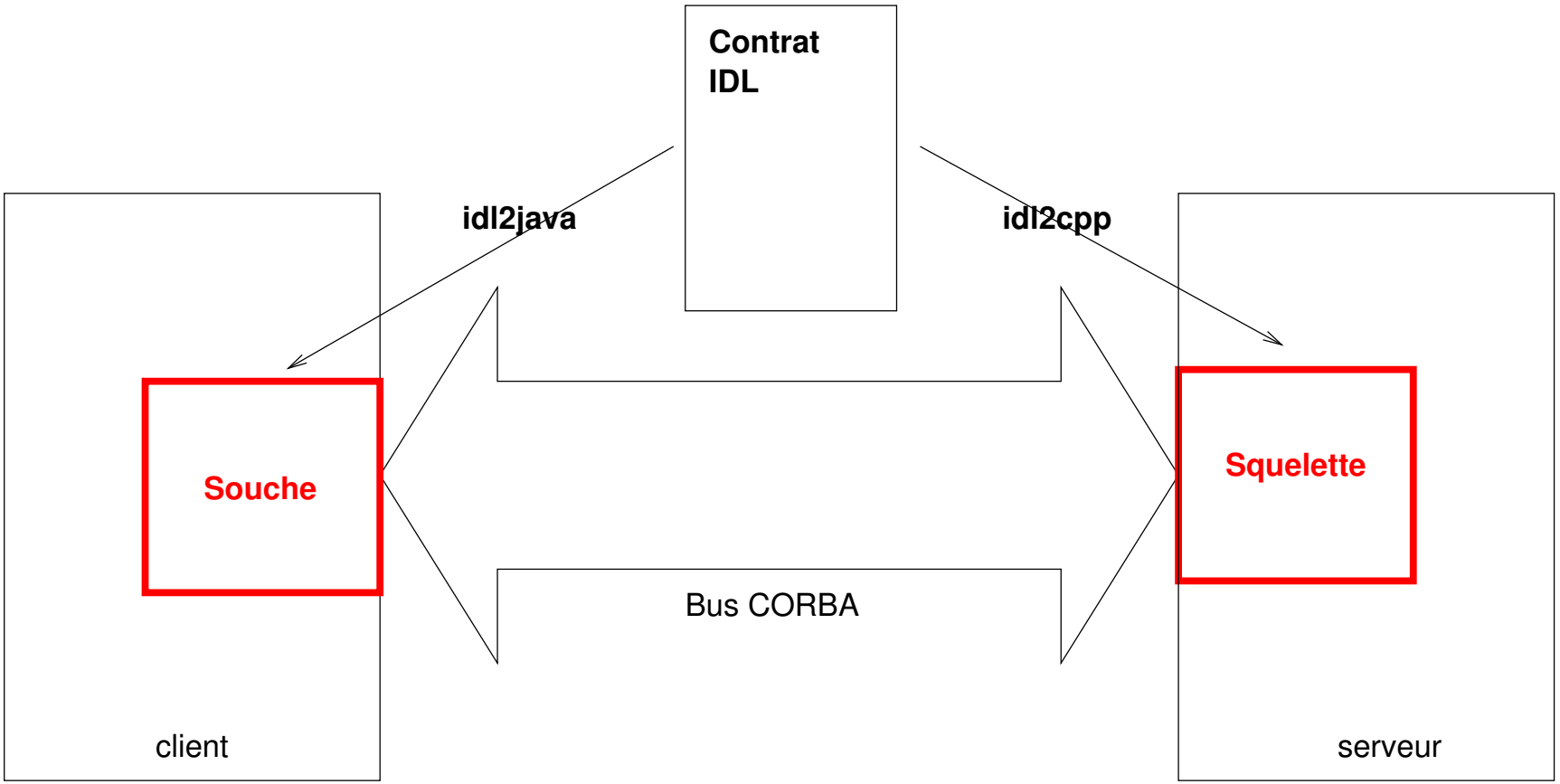
1. Écriture d'une interface IDL
2. Écriture d'une classe implantant l'interface
3. Écriture du programme serveur (accès au bus)
4. Écriture du programme client (accès au bus)

Le langage IDL : le contrat

Les contrats IDL sont projetés (pour le langage retenu) :

- en souches IDL (ou interface d'invocations statiques SII) dans l'environnement de programmation du client
- en squelettes IDL (ou interface de squelettes statiques SSI) dans l'environnement de programmation du fournisseur
- le client invoque localement les souches pour accéder aux objets
- les souches construisent des requêtes qui vont être transportées par le bus puis délivrées par celui-ci aux squelettes qui les délégueront aux objets

Le contrat IDL et la projection



Le langage IDL : le contrat

Exemple :

```
#pragma prefix "u-bourgogne.fr"
module date {
 typedef short Annee;
 typedef sequence<Annee> DesAnnees;
 enum Mois {Janvier, Fevrier, Mars, Avril, Mai, Juin,
     Juillet, Aout, Septembre, Octobre, Novembre,
    Decembre};
 typedef sequence<Mois> DesMois;
 enum JourDansLaSemaine {Lundi, Mardi, Mercredi, Jeudi,
      Vendredi, Samedi, Dimanche};
 typedef sequence<JourDansLaSemaine> DesJoursDansLaSemaine;
  typedef unsigned short Jour;
 typedef sequence<Jour> DesJours;
```

Le langage IDL : le contrat

Exemple :

```
struct Date {
    Jour le_jour;
    Mois le_mois;
    Annee l_annee;
  };
  typedef sequence<Date> DesDates;
  union DateMultiFormat
   switch(unsigned short) {
    case 0: string chaine;
    case 1: Jour nombreDeJours;
    default: Date date;
  };
  typedef sequence<DateMultiFormat> DesDateMultiFormats;
  exception ErreurInterne {};
  exception MauvaiseDate { DateMultiFormat date; };
```

Le langage IDL : le contrat

Exemple :

```
interface Traitement {
    boolean verifierDate(in Date d);

    JourDansLaSemaine calculerJourDansLaSemaine(in Date d)
    raises(ErreurInterne, MauvaiseDate);

    long nbJoursEntreDeuxDates(in Date d1, in Date d2)
     raises(MauvaiseDate);

    void dateSuivante (inout Date d, in Jour nombreJours)
      raises(MauvaiseDate);
  };
```

Le langage IDL : le contrat

Exemple :

```
interface Convertisseur {

    Jour convertirDateVersJourDansAnnee(in Date d)
     raises(MauvaiseDate);

    Date convertirChaineVersDate(in string chaine)
     raises(MauvaiseDate);

    string convertirDateVersChaine(in Date d)
 raises(MauvaiseDate);
```

Le langage IDL : le contrat

Exemple :

```
attribute Annee annee_courante;
// Fixer l'année courante pour l'opération suivante.
   Date convertirJourDansAnneeVersDate(in Jour jour);
   readonly attribute Annee base_annee ;
// L'année de référence pour les opérations suivantes.

   Date convertirJourVersDate(in Jour jour);
   Jour convertirDateVersJour(in Date d)
   raises(MauvaiseDate);

   void obtenirDate (in DateMultiFormat dmf, out Date d)
    raises(MauvaiseDate);
  };
   interface ServiceDate : Traitement, Convertisseur { };
```

Constructions IDL

- Un pragma permet de fixer les identifiants désignant les définitions IDL à l'intérieur du référentiel des interfaces
- Les identifiants assurent une désignation unique des définitions IDL quel que soit le référentiel des interfaces consulté
- Ils sont utilisés pour fédérer des IFR
- La forme la plus utilisée est le pragma prefix qui fixe l'organisation définissant un contrat IDL en utilisant son nom de domaine Internet. Les conflits de nom de module sont réglés par le pragma prefix
- Le pragma version permet de définir le numéro de version d'une spécification IDL
- Un module sert à regrouper des définitions de types qui ont un intérêt commun
- Permet aussi de limiter les conflits de noms pouvant intervenir entre plusieurs spécifications

Constructions IDL

- Les types de données de base sont :

  `void, short, unsigned short, long, unsigned long, long long (64 bits), unsigned long long, float, double, long double (128 bits), boolean, octet, char, string, wchar et wstring (format de caractères international) et fixed pour les nombres à précision fixe`

- Le format binaire de ces types est défini par la norme afin de régler les problèmes d'échanges de données entre environnements hétérogènes

Constructions IDL

- Les types de méta-données (TypeCode et any) sont une composante spécifique à IDL
- Le type TypeCode permet de stocker la description de n'importe quel type IDL
- Le type any permet de stocker une valeur IDL de n'importe quel type en conservant son TypeCode
- Ces méta-types permettent de spécifier des contrats IDL génériques indépendants des types de données manipulés,
- Par exemple une pile d'any stocke n'importe quelle valeur
- Une constante se définit par un type simple, un nom et une valeur évaluable à la compilation `const double PI = 3.1415 ;`

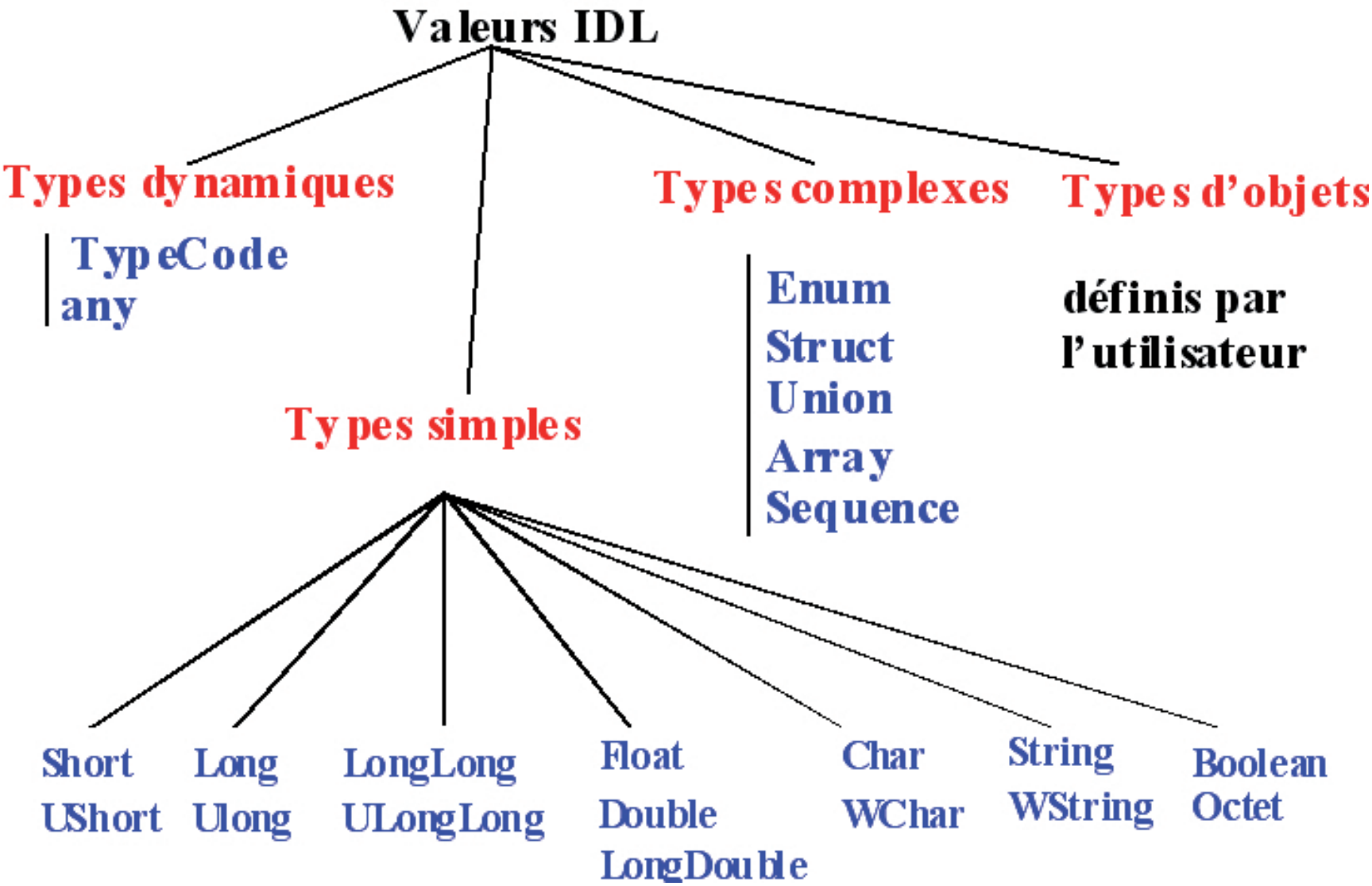
Constructions IDL

- Un alias (typedef) permet de créer de nouveaux types :
- Par exemple, il est plus clair de spécifier qu'une opération retourne un Jour plutôt qu'un entier X bits signé
- Une énumération (enum) définit un type discret via un ensemble d'identificateurs
- Par exemple : énumérer les jours de la semaine que d'utiliser un entier
- Une structure (struct) définit une organisation regroupant des champs cette construction est fortement employée car elle permet de transférer des structures de données composées entre objets CORBA
- Une union juxtapose un ensemble de champs, le choix étant arbitré par un discriminant de type simple (entiers, caractère, booléen ou énumération)

Constructions IDL

- Un tableau (array) sert à transmettre un ensemble de taille fixe de données homogènes
- Une séquence permet de stocker/transférer un ensemble de données homogènes dont la taille sera fixée à l'exécution
- Une exception spécifie une structure de données permettant à une opération : de signaler les cas d'erreurs ou de problèmes exceptionnels pouvant survenir lors de son invocation.
- Une interface décrit les opérations fournies par un type d'objets CORBA (Traitement et Convertisseur)
- Une interface IDL peut hériter de plusieurs autres interfaces (ServiceDate)

Le contrat IDL



Constructions IDL

L'héritage multiple est autorisé :

- On parle d'héritage de spécifications (ou de fonctionnalités)
- La seule contrainte imposée est qu'une interface ne peut pas hériter de deux interfaces qui définissent des opérations de mêmes noms
- Ce problème doit être résolu manuellement en évitant/éliminant les conflits de noms

La surcharge est interdite en IDL

Un objet CORBA ne peut implanter qu'une seule interface IDL. CORBA 3.0 intégre la notion d'interfaces multiples (Java ,COM)

Constructions IDL

Une opération se définit par :

- une signature qui comprend le type du résultat, le nom de l'opération, la liste des paramètres et la liste des exceptions
- un paramètre se caractérise par un mode de passage, un type et un nom formel :
- les modes de passages autorisés sont in, out et inout
- le résultat et les paramètres peuvent être de n'importe quel type exprimable en IDL
- Par défaut, l'invocation d'une opération est synchrone, cependant, il est possible de spécifier qu'une opération est asynchrone (oneway), le résultat est de type void, tous les paramètres sont en mode in qu'aucune exception ne peut être déclenchée

CORBA ne spécifie pas complètement l'opération oneway : l'invocation peut échouer, être exécutée plusieurs fois sans que l'appelant ou l'appelé soient informés

Constructions IDL

- Un attribut exprime une paire d'opérations pour consulter et modifier une propriété d'un objet
- Il se caractérise par un type et un nom
- On peut spécifier si l'attribut est en lecture seule (readonly) ou consultation/modification (mode par défaut)
- Le terme IDL attribut est trompeur, l'implantation d'un attribut IDL peut être faite par un traitement quelconque

Les limites d'IDL

- Le langage OMG-IDL n'offre pas de constructions pour exprimer la sémantique d'utilisation des objets :
  - des contraintes ;
  - des pré et post conditions sur les opérations ;
  - des invariants.
- L'expression de la sémantique permettrait :
  - de tester et valider automatiquement les objets ;
  - de décrire la qualité de service, c'est-à-dire les caractéristiques d'une implantation.

Les limites d'IDL

- L'impossibilité de sous-typer (d'étendre) la définition d'une structure ou d'une union
- L'interdiction de conflits de noms à l'intérieur d'un module ou d'une interface implique l'interdiction de surcharger des opérations
- Le passage par valeur de structures de données et non d'objets : si l'on veut transférer un graphe d'objets, il faut l'aplatir dans une séquence de structures (évolutions de CORBA 3.0)
- Impossibilité de spécifier des types intervalles (pratique pour exprimer des intervalles de temps)

Projection IDL vers un langage de programmation

Une projection au sens de CORBA (*mapping*, correspondance) est la traduction d'une spécification IDL dans un langage d'implantation.

Pour permettre la portabilité des applications d'un bus vers un autre :

- les règles de projection sont précisement définies (standard) ;
- elles fixent précisément la traduction de chaque construction IDL en une ou plusieurs constructions du langage cible ; '
- des règles existent pour les langages C, C++, SmallTalk, Ada, Java et Cobol orienté objet

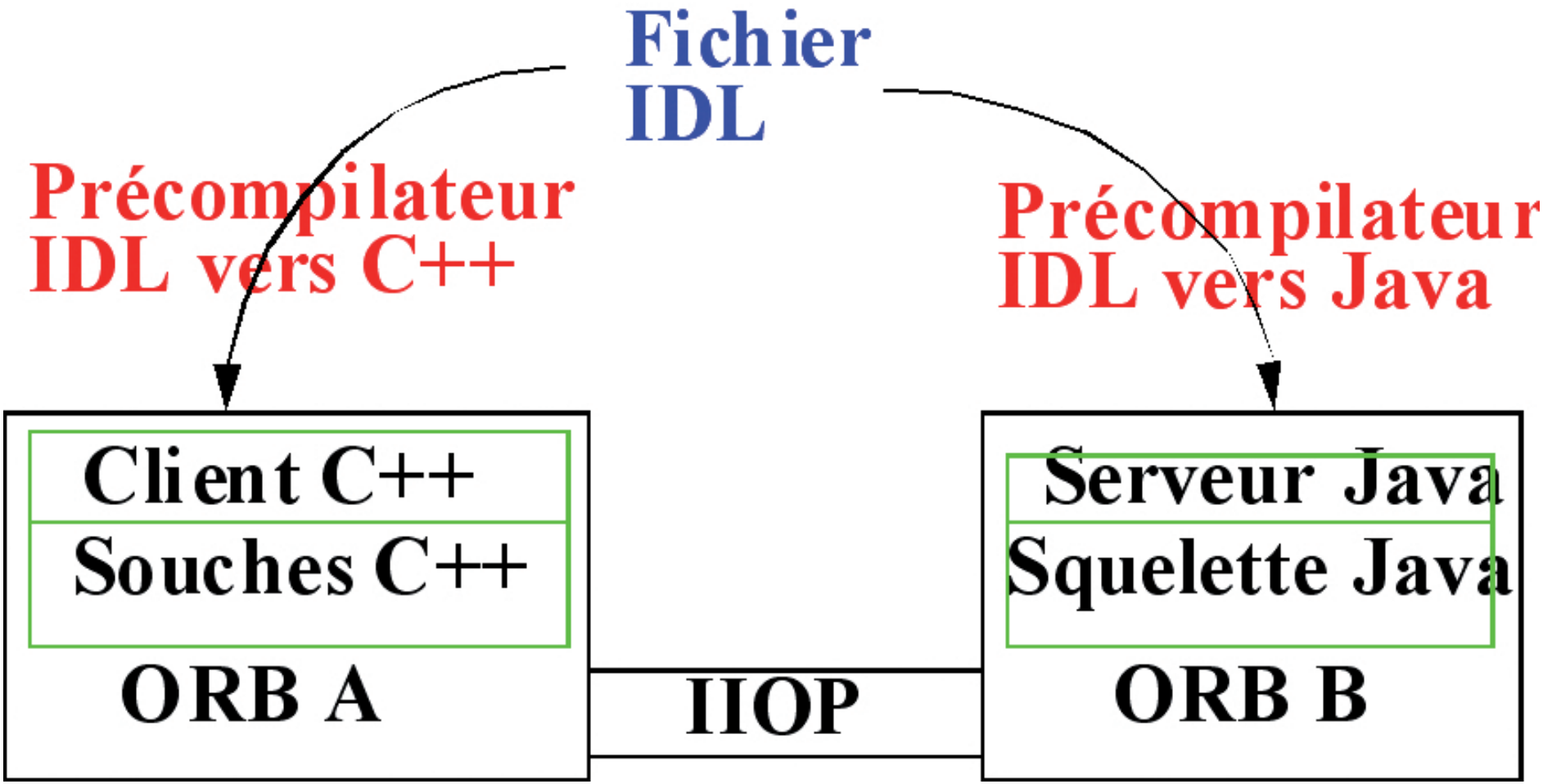
Projection

Construction OMG-IDL	Projection en C++	Projection en Java
`module M { … };`	Au choix du produit CORBA : `namespace M { … }` ou `class M { … }` ou préfixe `M_`	`package M`
`interface J:I {…};`	`class J : public virtual I {…};`	`interface J extends I {…}`
`String`	`char *`	`java.lang.String`

Réalisation de la projection

- La projection est réalisée par un pré-compilateur IDL dépendant du langage cible et de l'implantation du bus CORBA chaque produit CORBA fournit un pré-compilateur IDL pour chacun des langages supportés
- Le code des applications est alors portable d'un bus à un autre les souches/squelettes générés s'utilisent toujours de la même manière quel que soit le produit CORBA
- Par contre, le code des souches et des squelettes IDL n'est pas forcément portable il dépend de l'implantation du bus pour lequel ils ont été généré

Projection : les outils



Étapes de la réalisation d'une application CORBA

1. Définition du contrat IDL :
   - définir les objets composants l'application à l'aide d'une méthodologie orientée objet cette modélisation est ensuite traduite sous la forme de contrats IDL
2. Pré-compilation du contrat IDL :
   - les interfaces des objets sont décrites dans des fichiers texte (.idl)
   - le pré-compilateur prend en entrée un fichier .idl et opère un contrôle les définitions IDL
3. Projection vers les langages de programmation :
   - le pré-compilateur IDL génère le code des souches qui sera utilisé par les applications clientes des interfaces décrites dans le fichier IDL
   - le code des squelettes pour les programmes serveurs implantant ces types
4. Implantation des interfaces IDL : en complétant et/ou en réutilisant le code généré pour les squelettes, le développeur implante les objets

Étapes de la réalisation d'une application CORBA

5. Implantation des serveurs d'objets :
   - le développeur écrit les programmes serveurs qui incluent l'implantation des objets et les squelettes pré-générés
   - Ces programmes contiennent le code pour :
     - se connecter au bus
     - instancier les objets racines du serveur
     - rendre publiques les références sur ces objets à l'aide par exemple du service Nommage
     - se mettre en attente de requêtes pour ces objets.
6. Implantation des applications clientes des objets : le développeur écrit un ensemble de programmes clients qui agissent sur les objets ces programmes incluent le code des souches, le code pour l'IHM et le code spécifique à l'application
7. Les clients obtiennent les références des objets serveurs en consultant le service Nommage

Étapes de la réalisation d'une application CORBA

8. Installation et la configuration des serveurs : cette phase consiste à installer dans le référentiel des implantations les serveurs pour automatiser leur activation lorsque des requêtes arrivent pour leurs objets
9. Diffusion et la configuration des clients
10. Exécution répartie de l'application

Les composants fondamentaux du bus

- La structure d'accueil des objets, c'est-à-dire le serveur fournit :
  - un espace mémoire pour l'état des objets
  - un contexte d'exécution des opérations
- L'adaptateur d'objets permet à une structure d'accueil de :
  - savoir générer et interpréter des références d'objets
  - connaître l'implantation courante associée à un objet
  - savoir déléguer les requêtes aux objets à leur implantation

L'adaptateur d'objet

- L'adaptateur d'objets isole le bus des différentes technologies pouvant être employées
- Différents types d'adaptateurs sont envisageables :
  - BOA (Basic Object Adapter) : les structures d'accueil sont matérialisées par des processus systèmes
  - POA (Portable Object Adapter) : l'implantation des objets est réalisée par des objets fournis par un langage de programmation
  - OODA (Object-Oriented Database Adapter) : la structure d'accueil est une base de données orientée objet
  - LOA (Library Object Adapter) : le code d'implantation des objets est stocké dans des bibliothèques chargées dans l'espace mémoire des applications clientes
  - COA (Card Object Adapter) : l'implantation des objets est stockée dans une carte à microprocesseur

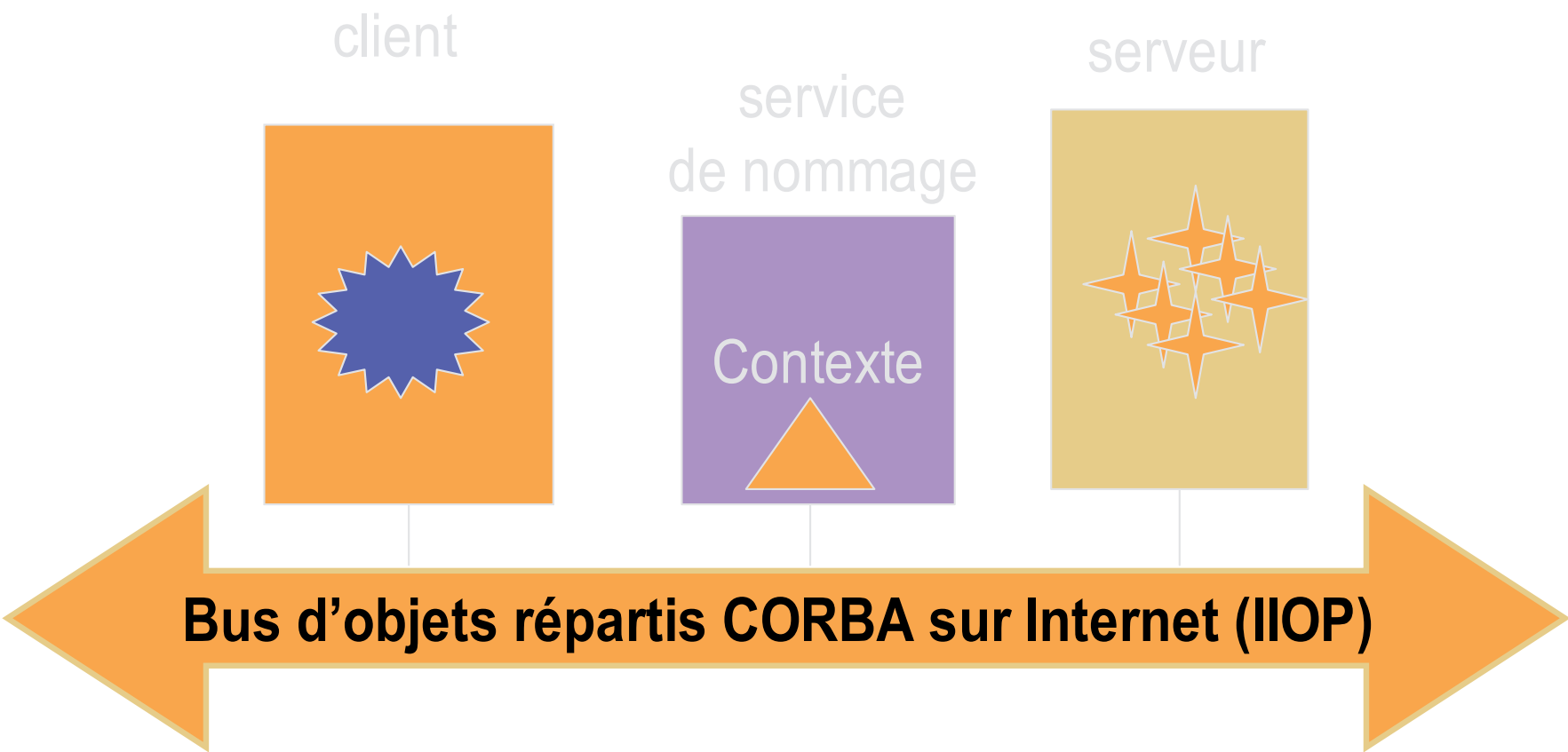
L'adaptateur d'objet

- Les premières versions de CORBA spécifiaient uniquement le BOA
- le BOA fut sous-spécifié entraînant ainsi des incompatibilités entre différents produits CORBA
- L'OMG a défini un nouvel adaptateur d'objets POA dans CORBA 2.2
- ORBacus v3 : fournit l'interface BOA offre une opération pour se mettre en attente des requêtes aux objets (`impl_is_ready`) et une opération dans l'ORB pour initialiser l'adaptateur d'objets (`BOA_init`)

Services de nommage et recherche d'objets

- Il est difficile de manipuler manuellement les IOR
- Pour masquer cette manipulation directe, le bus CORBA fournit des services standards de recherche d'objets :
  - Le service Nommage (Naming Service) permet la recherche des objets en fonction de noms symboliques leur ayant été associés
  - Le service Vendeur (Trader Service) permet la recherche des objets en fonction de leurs caractéristiques

Services de base de l'ORB



Services de nommage et recherche d'objets

- Le principe de mise en oeuvre de ces services est le suivant
- Enregistrement : les applications fournissant des objets enregistrent les références auprès du service de recherche en leur associant soit un nom symbolique soit des caractéristiques
- Interrogation : les applications clientes interrogent le service pour obtenir les références des objets qu'elles veulent invoquer
- Recherche : le service recherche dans l'ensemble des références d'objet enregistrées celles qui correspondent le mieux aux critères demandés par les clients

Services de nommage et recherche d'objets

- Le service Nommage (CosNaming) définit un espace de désignation symbolique des objets
- Structuré par un graphe de contextes de nommage (interface NamingContext)
- Chaque contexte maintient une liste d'associations entre des noms symboliques et des références d'objet.
- À l'intérieur d'un contexte, un nom (struct NameComponent) doit être unique et désigne soit une référence d'objet soit un autre contexte
- La concaténation de plusieurs noms forme alors un chemin d'accès (typedef Name) à travers l'espace de désignation symbolique

Services de nommage et recherche d'objets

Un contexte fournit des opérations :

- pour ajouter une association entre un nom et une référence (`bind`)
- mettre à jour une telle association (`rebind`),
- se connecter ou reconnecter à un contexte (`bind_context` et `rebind_context`),
- rechercher la référence désignée par un chemin (`resolve`)
- détruire une association (`unbind`)
- créer un nouveau contexte indépendant (`new_context`)
- créer un contexte et le connecter (`bind_new_context`)
- détruire définitivement le contexte (`destroy`)
- lister le contenu (`list`)
- Ces opérations peuvent déclencher des exceptions : `NotFound`, `CannotProceed`, `InvalidName`, `AlreadyBound`, `NotEmpty`

Services de nommage et recherche d'objets

Enregistrer une référence :

```
CosNaming_Name_var nsNom = new CosNaming_Name();
nsNom->length(1);
nsNom[0].id = (const char*) "nom logique objet";
nsRef->rebind (nsNom, objet);
```

Services de nommage et recherche d'objets

Une application cliente doit obtenir les références des objets qu'elle désire invoquer :

- Soit elle utilise la primitive `string_to_object` sur une chaîne codifiant une IOR
- Soit elle utilise le service Nommage :
  - obtenir la référence de ce service
  - créer un chemin valide
  - rechercher la référence associée à ce chemin via l'opération resolve.

```
CosNaming_Name_var nsNom = new CosNaming_Name();
  nsNom->length(1);
  nsNom[0].id = (const char*) "description objet";
  CORBA_Object_var objRef = nsRef->resolve (nsNom);
```

Exemple avec ORBacus 3.1.2

Accessible sur les machines (/usr/local/OB-X.Y.Z)

- Le contrat en IDL : (Hello.idl)

```
interface Hello
{
    void hello();
};
```

- Projection :

```
$bash2.5 idl Hello.idl
=> Hello.h, Hello.cpp, Hello_skel.h, Hello_skel.cpp
```

Implémentation du serveur

Pour créer le serveur :

- Définir une classe `Hello_impl` descendant de `POA_Hello` définie dans `Hello_skel.h`

  Fichier Hello_impl.h

```
#include <Hello_skel.h>
class Hello_impl : public Hello_skel{
 public: virtual void hello();
};
```

  Fichier Hello_impl.cpp

```
#include <OB/CORBA.h>
#include <Hello_impl.h>
void Hello_impl::hello(){
 cout << "Hello World!" << endl;}
```

Implémentation du serveur

```
#include <OB/CORBA.h>
#include <Hello_impl.h>
#include <fstream.h>
int main(int argc, char* argv[],char*[])
{
CORBA_ORB_var orb =CORBA_ORB_init(argc, argv);
CORBA_BOA_var boa = orb -> BOA_init(argc, argv);
Hello_var p = new Hello_impl;
CORBA_String_var s = orb -> object_to_string(p);
const char* refFile = "Hello.ref";
ofstream out(refFile);
out << s << endl;
out.close();
boa -> impl_is_ready(CORBA_ImplementationDef::_nil());
}
```

Implémentation du client

```
#include <OB/CORBA.h>
#include <Hello.h>
#include <fstream.h>
int main(int argc, char* argv[], char*[])
{
CORBA_ORB_var orb = CORBA_ORB_init(argc, argv);
const char* refFile = "Hello.ref";
ifstream in(refFile);
char s[1000];
in >> s;
CORBA_Object_var obj = orb -> string_to_object(s);
Hello_var hello = Hello::_narrow(obj);
hello -> say_hello();
}
```

Implémentation du serveur en Java

- Interface en IDL :

```
1  interface Hello
2  {
3      void say_hello();
4      void shutdown();
5  };
```

- Implémentation de l'interface

```
1  package hello;
2  public class Hello_impl extends HelloPOA{
3   // The servants default POA
4      private org.omg.PortableServer.POA poa_;
5      Hello_impl(org.omg.PortableServer.POA poa){
6          poa_ = poa;
7      }
8      public void say_hello(){
9          System.out.println("Hello World!");
10     }
11     public void shutdown(){
12         _orb().shutdown(false);
13     }
14     public org.omg.PortableServer.POA_default_POA(){
15         return poa_;
16     }
17 }
```

Implémentation du serveur en Java

- Version serveur Java avec POA :

```
1  package hello;
2  public class Server{
3     static int run(org.omg.CORBA.ORB orb, String[] args)
4      throws org.omg.CORBA.UserException{
5      // Resolve Root POA
6      org.omg.PortableServer.POA rootPOA =
7              org.omg.PortableServer.POAHelper.narrow(
8                  orb.resolve_initial_references("RootPOA"));
9      // Get a reference to the POA manager
10     org.omg.PortableServer.POAManager manager = rootPOA.the_POAManager();
11     // Create implementation object
12     Hello_impl helloImpl = new Hello_impl(rootPOA);
13     Hello hello = helloImpl._this(orb);
14     // Save reference
15     try {
16             String ref = orb.object_to_string(hello);
17             String refFile = "Hello.ref";
18             java.io.FileOutputStream file = new java.io.FileOutputStream(refFile);
19             java.io.PrintWriter out = new java.io.PrintWriter(file);
20             out.println(ref); out.flush(); file.close();
21         } catch(java.io.IOException ex) {
22             System.err.println("hello.Server:␣can't␣write␣to␣'" +  ex.getMessage()
                   + "'");
23             return 1;
24         }
```

Implémentation du serveur en Java

- Version serveur Java avec POA :

```
1   public static void
2     main(String args[])
3     {
4         java.util.Properties props = System.getProperties();
5         props.put("org.omg.CORBA.ORBClass", "com.ooc.CORBA.ORB");
6         props.put("org.omg.CORBA.ORBSingletonClass", "com.ooc.CORBA.ORBSingleton");
7         props.put("ooc.orb.id", "Hello-Server");
8         int status = 0;
9         org.omg.CORBA.ORB orb = null;
10        try{
11            orb = org.omg.CORBA.ORB.init(args, props);
12            status = run(orb, args);
13        } catch(Exception ex){
14            ex.printStackTrace();
15            status = 1;
16        }
17        if(orb != null){
18            try{orb.destroy();}
19            catch(Exception ex) {
20                ex.printStackTrace();
21                status = 1;
22            }
23        }
24        System.exit(status);
25    }
26  }
```

Implémentation du client en Java

- Version Client Java avec POA :

```
1  package hello;
2  import java.awt.*;
3  import java.awt.event.*;
4  public class Client extends java.applet.Applet implements ActionListener{
5      static int run(org.omg.CORBA.ORB orb, String[] args)
6         throws org.omg.CORBA.UserException{
7         // Get "hello" object
8        org.omg.CORBA.Object obj = orb.string_to_object("relfile:/Hello.ref");
9        if(obj == null){
10             System.err.println("hello.Client: cannot read IOR from Hello.ref");
11             return 1;
12         }
13         Hello hello = HelloHelper.narrow(obj);
14         // Main loop
15         System.out.println("Enter 'h' for hello, 's' for shutdown or 'x' " +  "for
               exit:");
16         int c;
17         try{
18             String input;
19             java.io.BufferedReader in = new java.io.BufferedReader(
20                 new java.io.InputStreamReader(System.in));
```

Implémentation du client en Java

- Version Client Java avec POA :

```
1  do {
2    System.out.print(">␣");
3    input = in.readLine();
4    if(input.equals("h"))
5              hello.say_hello();
6    else if(input.equals("s"))
7              hello.shutdown();
8   } while(!input.equals("x"));
9  }catch(java.io.IOException ex) {
10       System.err.println("Can't␣read␣from␣`" + ex.getMessage() + "'");
11       return 1;
12 }
13 return 0;
14 }
15 public static void main(String args[]){
16          int status = 0;
17          org.omg.CORBA.ORB orb = null;
18          java.util.Properties props = System.getProperties();
19          props.put("org.omg.CORBA.ORBClass", "com.ooc.CORBA.ORB");
20          props.put("org.omg.CORBA.ORBSingletonClass","com.ooc.CORBA.ORBSingleton");
21          props.put("ooc.orb.id", "Hello-Client");
22          try{
23              orb = org.omg.CORBA.ORB.init(args, props);
24              status = run(orb, args);
25          }catch(Exception ex){ ex.printStackTrace(); status = 1;}
```

Implémentation du client en Java

- Version Client Java avec POA :

```
1  if(orb != null) {
2              try{
3                  orb.destroy();
4              }catch(Exception ex){ ex.printStackTrace(); status = 1;}}
5          System.exit(status);}
6      // Members only needed for applet
7      private Hello hello_;
8      private Button button_;
9      // Applet initialization
10     public void init() {
11         String ior = getParameter("ior");
12         // Create ORB
13         org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(this, null);
14         // Create client object
15         org.omg.CORBA.Object obj = orb.string_to_object(ior);
16         if(obj == null) throw new RuntimeException();
17         hello_ = HelloHelper.narrow(obj);
18         // Add hello button
19         button_ = new Button("Hello");
20         button_.addActionListener(this);
21         this.add(button_);}
22     // Handle events
23     public void actionPerformed(ActionEvent event) {
24        hello_.say_hello(); }
25 }
```