Objets distribués - Corba - Code Mobile
Partie 1 : Concepts - IDL/Corba

Cyrille Bertelle et Damien Olivier

Cyrille.Bertelle@univ-lehavre.fr Damien.Olivier@univ-lehavre.fr

Laboratoire d'informatique du Havre

Introduction

On évoque les technologies actuelles permettant la distribution d'objets dans des environnements de réseaux informatiques : les objets distribués avec Java/RMI, les brokers d'objets à la norme CORBA, les agents mobiles, les applications aux au e-commerce.

Organisation du cours

1 Concepts sur les systèmes distribués à objets et l'IDL/CORBA *(Jeudi 4/12/2003 - C. Bertelle)*

2 Les bases de CORBA : ORB, projections, invocation statique *(Lundi 5/01/2004 - C. Bertelle)*

3 Service de nommage, adaptateur d'objet (POA) et référentiel d'interfaces en CORBA
(Jeudi 8/01/2004 - D. Olivier)

4 Les mécanismes dynamiques en CORBA
(Jeudi 15/01/2004 - D. Olivier)

5 Codes mobiles et Aglets
(Jeudi 22/01/2004 - D. Olivier)

6-8 Applications professionnelles : le e-commerce
(11,12 et 13 Fevrier 2004 - B. Adouobo)

Bibliographie de base

- Livres :

“Client server programming with Java & Corba”

Orfali, Edwards, Harley - ITP France

“CORBA : des concepts à la pratique”

Geib, Gransart, Merle - Masson 97

“The CORBA reference guide”

A. Pope - Addison Wesley 97

“Au cœur de Corba avec Java”

J. Daniel - Vuibert 2000

Webographie de base

- Web :
  OMG :
  `http://www.omg.org`
  ORBacus :
  `http://www.ooc.com`
  CorbaScript :
  `http://corbaweb.lifl.fr`
  D. Schmidt Page perso :
  `http://www.cs.wustl.edu/~schmidt/`
  U. Vienne :
  `http://www.infosys.tuwien.ac.at/Research/Corba/`
  Portail :
  `http://www.cetus-links.org/oo_corba.html`

Plan : Concepts sur les systèmes distribués à objets et IDL/Corba

1 Objets et informatique distribuée ;
2 Les concepts de Corba ;
3 IDL/Corba

1. Objets et informatique distribuée

Plan

1.1 Middleware et distribution : mutation des Systèmes Informatique
1.2 Appréhender les systèmes décentralisés
1.3 L'hétérogénéité dans les systèmes informatiques distribués
1.4 L'évolution des modèles d'architectures distribuées
1.5 L'évolution des communications dans les systèmes informatiques distribués
1.6 Les mécanismes de l'invocation distribué
1.7 De l'approche orientée objet aux Objets distribués

1.1 Middleware et distribution : mutation des Systèmes Informatiques

- Mutation des SI : du sédentarisme au multi-coopératif
  - Développement des réseaux hétérogènes tant sur les architectures que les systèmes ;
  - De la révolution Web/documents partagés vers la révolution ORB/Programmation répartie :
    - appel dynamique de services distants ;
    - agents mobiles ;
    - composants logiciels coopératifs.
  - Middleware : canaux et protocoles d'échange, plate-forme d'intégration.

1.1 Middleware et distribution :

mutation des Systèmes Informatiques

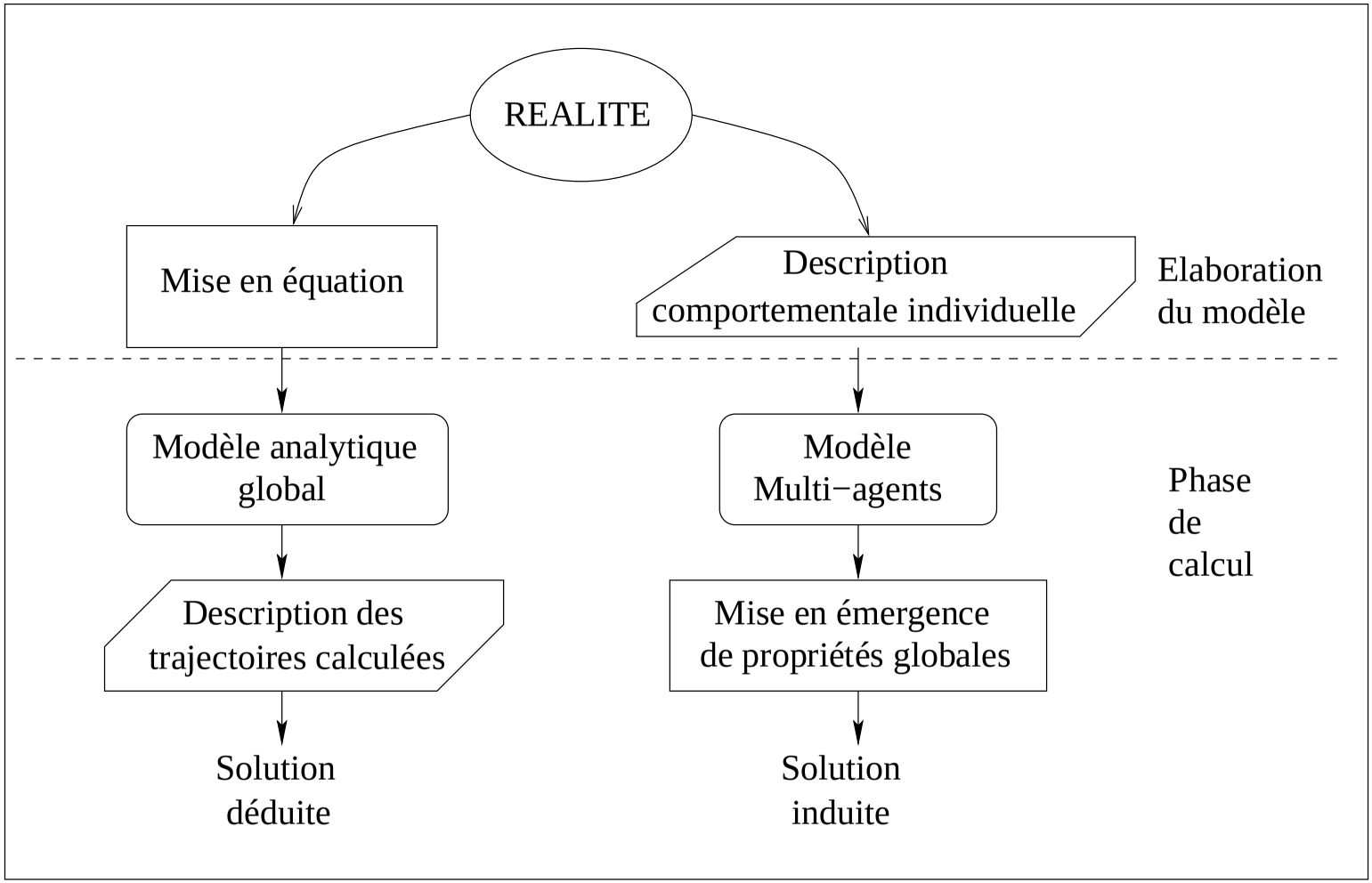
- Des nouveaux besoins
  - Intégration de logiciels d'origine diverses ;
  - Accès aux logiciels à l'intérieur et à l'extérieur des sociétés, via Internet :
    - gestion agences/succursales d'entreprises ;
    - échange clients/fournisseurs (e-commerce).
  - On cherche à répartir la charge statiquement ou dynamiquement ;
  - Fiabilisation et tolérance aux pannes par replication.

→ Le développement logiciel décentralisé vu comme un nouveau paradigme

1.2 Appréhender les systèmes décentralisés

- Nous vivons et participons à un monde naturellement décentralisé, générant des organisations (écologiques, sociales, ...) émergentes $\rightarrow$ auto-organisation ;
- Modèles centralisés vs. modèles décentralisés
  - Systèmes explicatifs majoritairement centralisés (lois globales) ;
  - Approches récentes orientées vers le décentralisme : cybernétique, systémique.

Modèles centralisés vs. modèles décentralisés



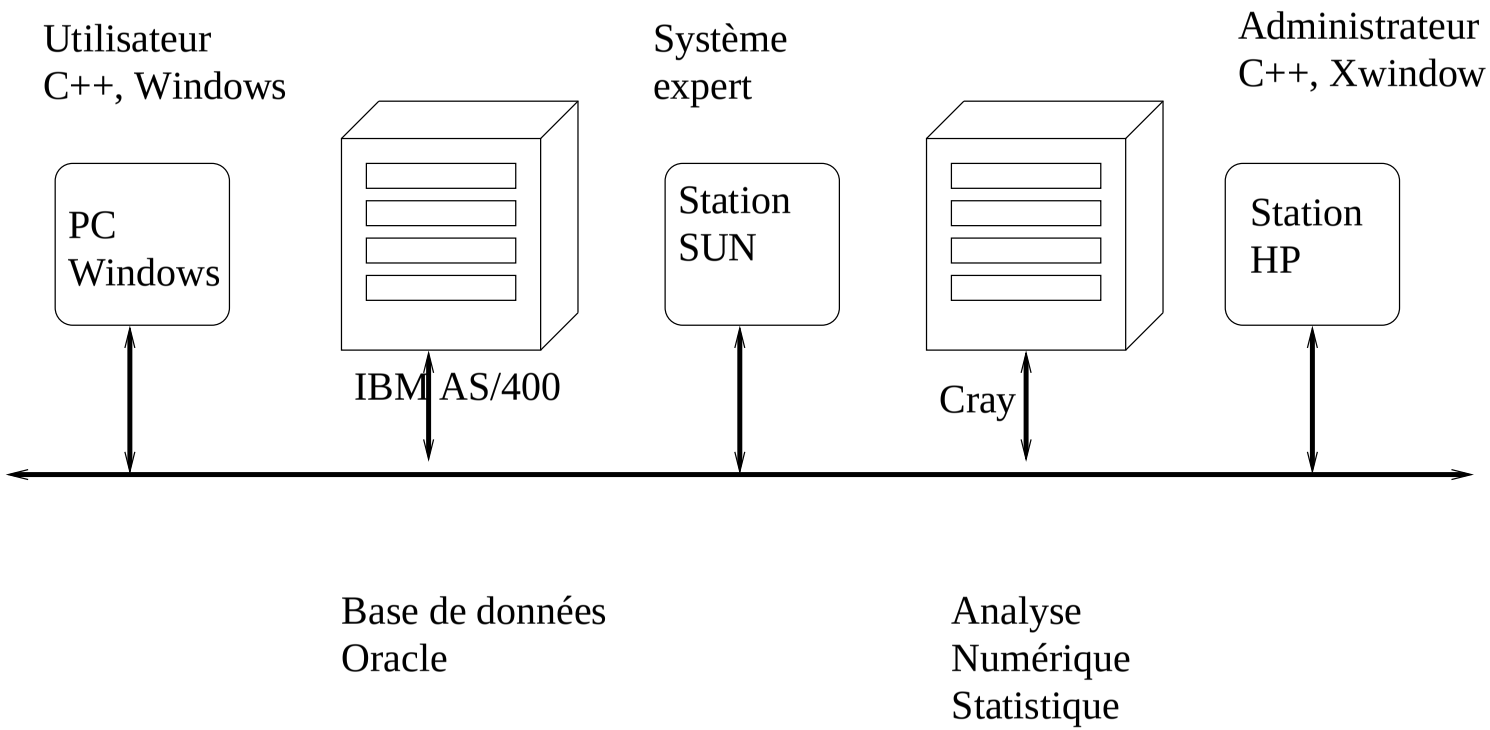
1.2 Appréhender les systèmes décentralisés

Des systèmes décentralisés ...
... aux implémentations par interactions

Changer de paradigme de programmation

- Le développement logiciel vu comme un univers parallèle, interactif et dynamique.
- Construire des composants logiciels qui vont coexister, collaborer et coopérer et qui font partie d'un tout en interaction.

1.3 L'hétérogénéité dans les systèmes informatiques distribués



1.3 L'hétérogénéité dans les systèmes informatiques distribués

Les traits d'hétérogénéité à prendre en compte

Machine/Os hétérogènes

- Caractéristiques des processeurs
  - représentation mémoire des nombres en 16, 32 ou 64 bits suivant les machines (Intel, Sparc, Alpha, ...)
- Les différents systèmes d'exploitation
  - Mono-utilisateur (Windows 3.x, 9x, 2000, XP, ...) ou multi-utilisateurs (Unix, ...) ;
  - Mono-tâche ou multi-tâches (+ fréquents aujourd'hui)

1.3 L'hétérogénéité dans les systèmes informatiques distribués

Les traits d'hétérogénéité à prendre en compte

Langages hétérogènes

- impératifs : C, Pascal, Fortran, ...
- objets : C++, Smalltalk, Ada, Java, ...
- fonctionnels : Lisp, Caml, ...
- logiques ou orientés IA : Prolog, systèmes experts, ...

$\rightarrow$ Plusieurs modèles de programmation avec des puissances d'expressions mais aussi souvent des faiblesses d'efficacité.

$\rightarrow$ Solution idéale : pas de langage universel mais développement de chaque partie d'une application dans le langage le mieux adapté.

1.3 L'hétérogénéité dans les systèmes informatiques distribués

Les traits d'hétérogénéité à prendre en compte

Nouveaux concepts du Génie Logiciel

- Développement coopératif et multi-participants du logiciel ;
- Réutilisabilité ;
- Composants logiciels modulaires.

1.3 L'hétérogénéité dans les systèmes informatiques distribués

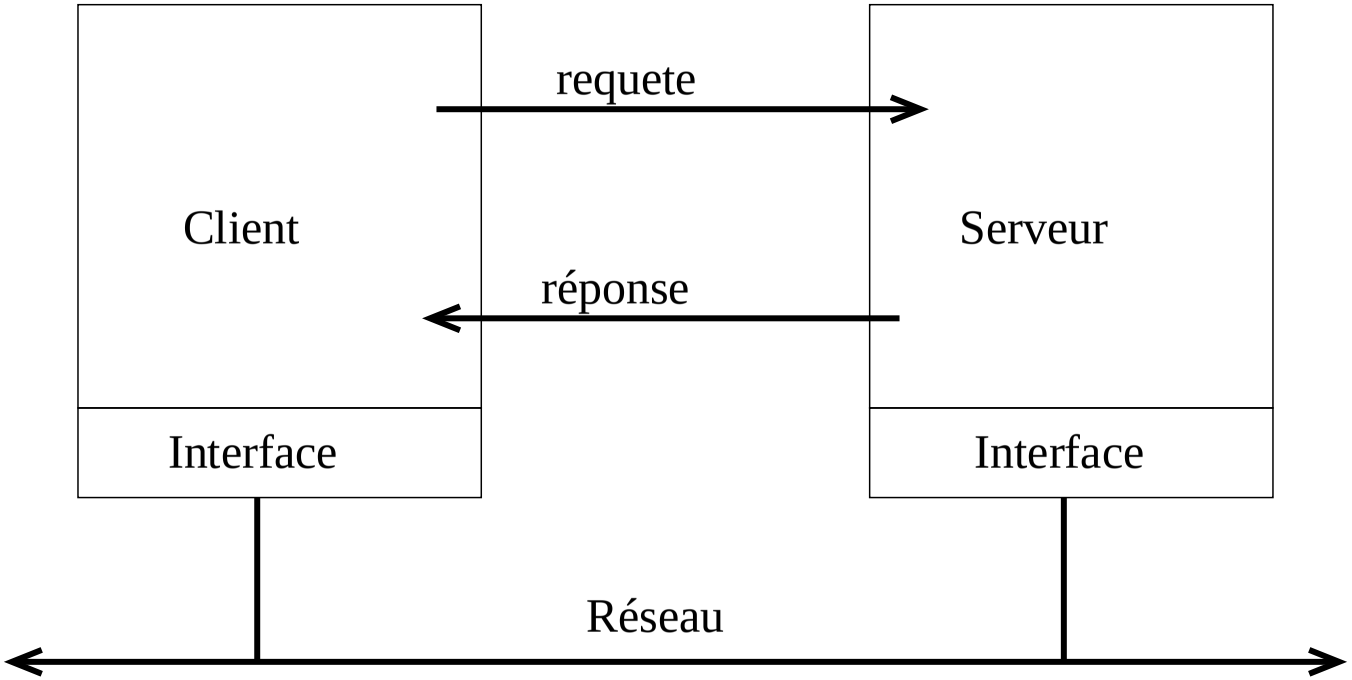
Les traits d'hétérogénéité à prendre en compte

Réseaux et systèmes de communication

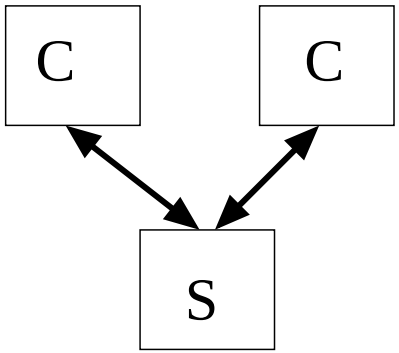
- entre machines et OS différents ;
- entre langages différents ;
- interfaçage/intégration de composants.

1.4 L'évolution des modèles d'architectures distribuées

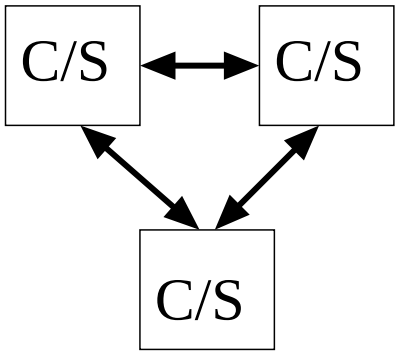
Modèle client/serveur



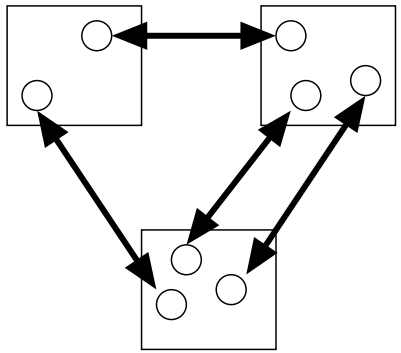
1.4 L'évolution des modèles d'architectures distribuées



1. un serveur et des clients



2. des services et des clients distribués

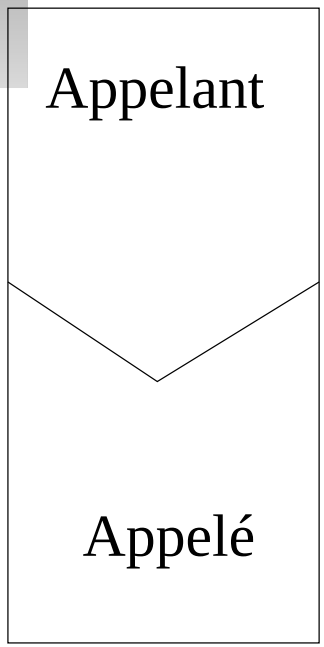


3. des objets distribués

1.5 Evolution des communications

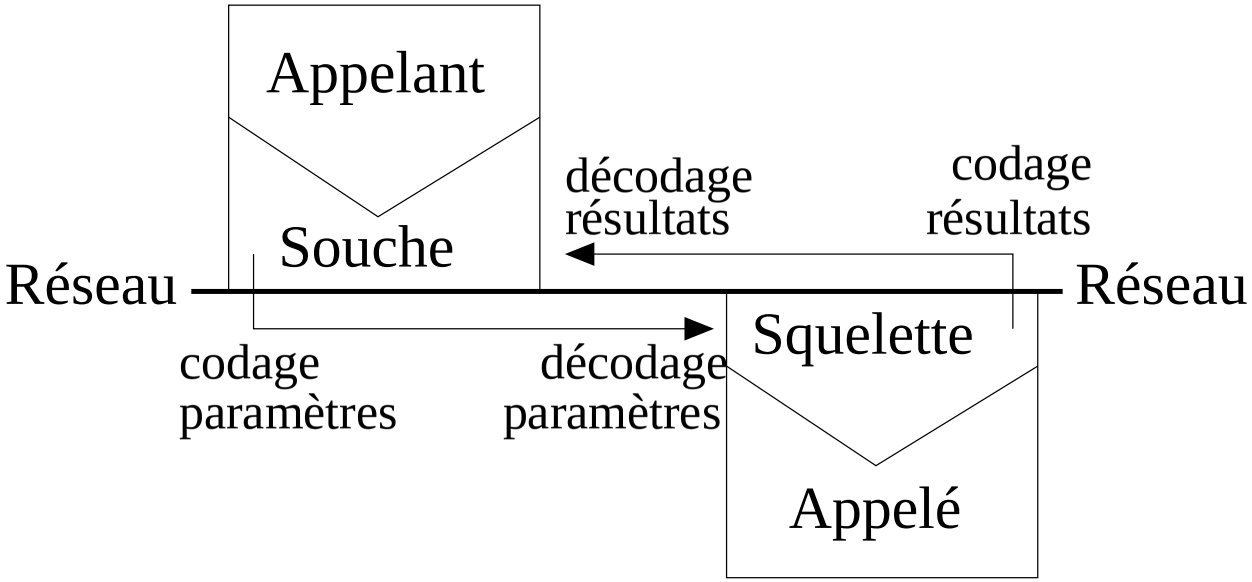
- IP : Communication orientée Paquets
  communication par échanges de paquets
- TCP/IP : Communication orientée Flux
  communication par flux (sockets, messages, ...)
- PVM/MPI : Communication orientée Messages
  flux de messages gérés par bibliothèques logicielles
- RPC : Communication orientée Procédures
  invocation de procédures dans un autre processus
- HTML/XML/CGI : Communication orientée Documents
  documentation distribuée + appel d'applications de service interfacées
- RMI, CORBA : Communication orientée Objets
  communication entre objets (invocation de méthodes)

1.6 Les mécanismes de l'invocation distribuée



Appel local

...



Invocation distribuée

Appel distant : code général

Appelant

```
stub_F(args) {
  requete r =
    new_requete(F);
  codage(r, args);
  envoi(r);
  attend_reponse(r);
  decodage(r, results);
  return results;
}
```

Appelé

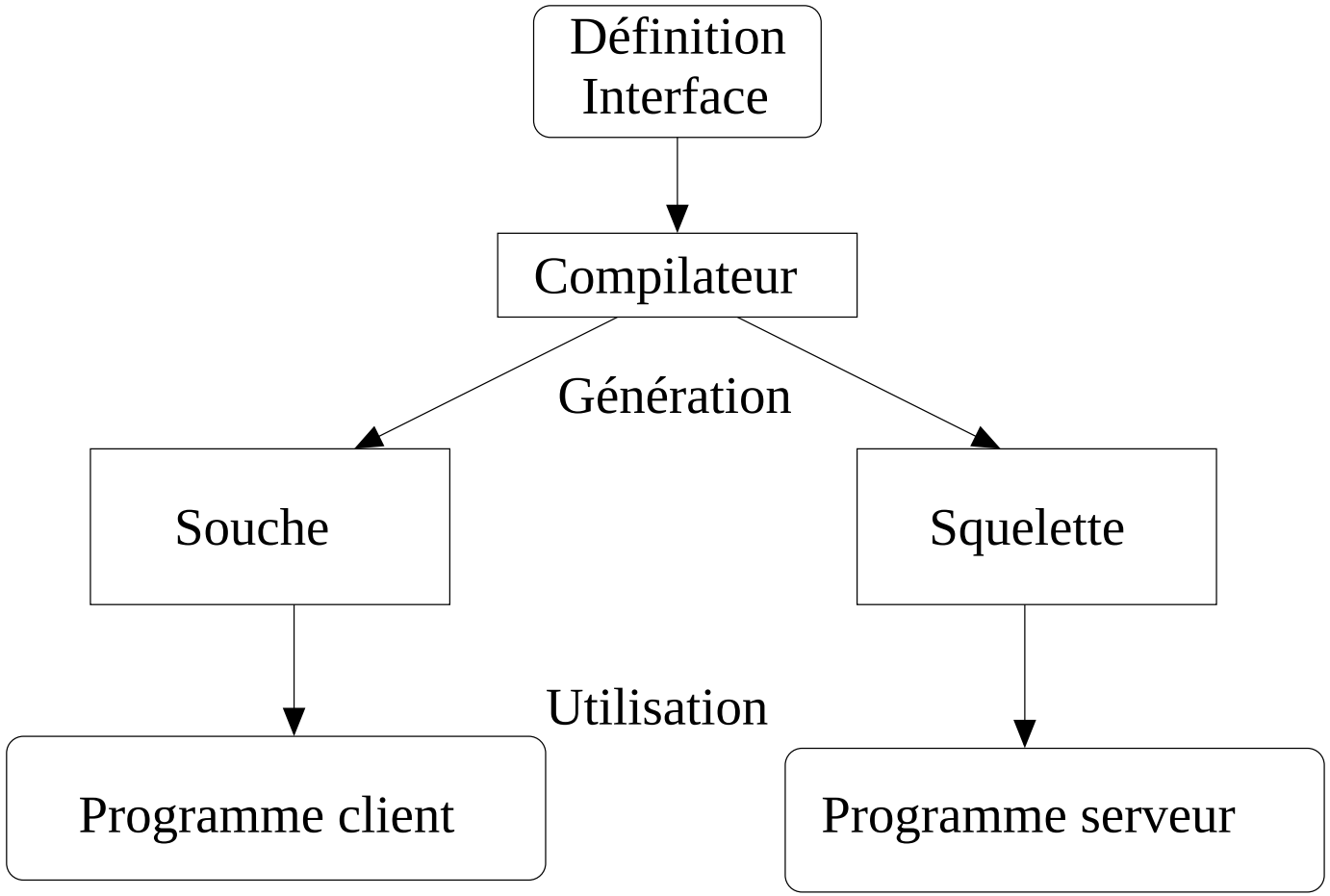
```
skel_F(requete r) {
  // declaration args

  decodage(r, args);

  F(args);
  codage(r, results);
  envoi_results(r);
}
```

1.6 Les mécanismes de l'invocation distribuée

Compilation de l'interface de l'invocation



1.7.1 Objet

objet = identifiant + comportement + état

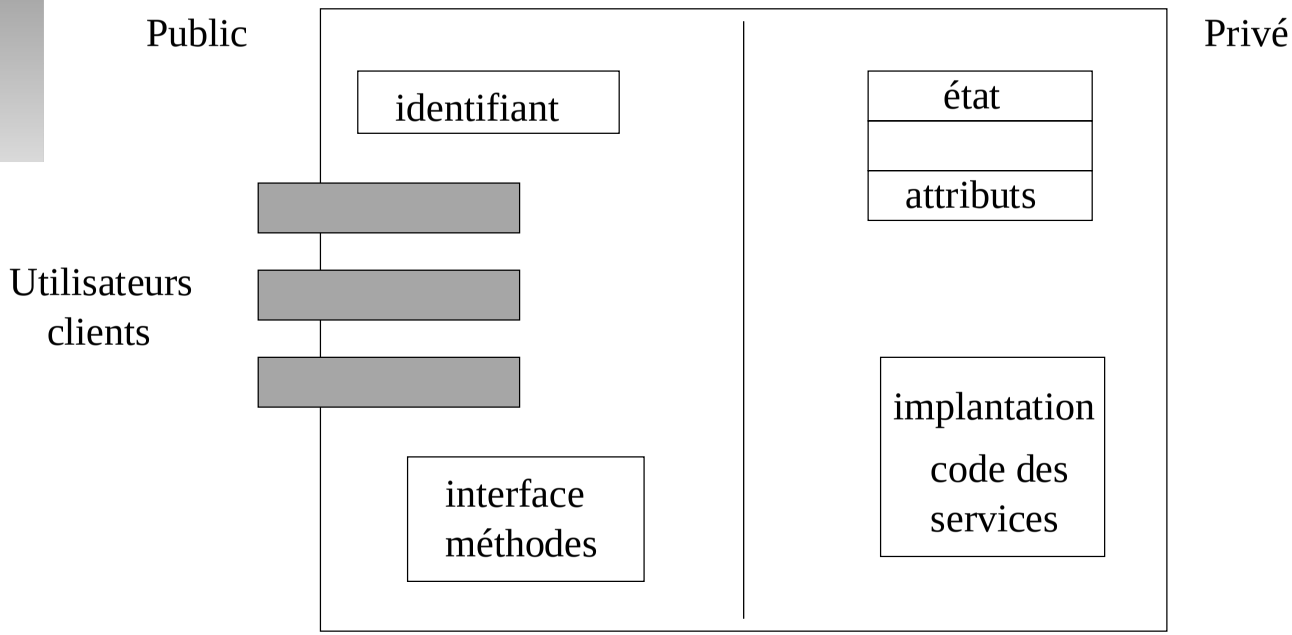
- identifiant : identificateur unique et invariant qui permet de référencer l'objet indépendemment des autres objets.
- Comportement : défini par les méthodes applicables à sa classe.
- Etat : valeur simple ou structurée de l'objet.

1.7.2 Classe

classe = attributs + méthodes + instanciation

- Définit le type et le comportement commun des instances ;
- Objet = instance d'une classe ;
- Attributs : variables d'instances, définissent l'état de l'objet ;
- Méthodes : opérations applicables à un objet de la classe ;
- Instanciation : mécanisme de création d'un objet.

1.7.3 Interface et services

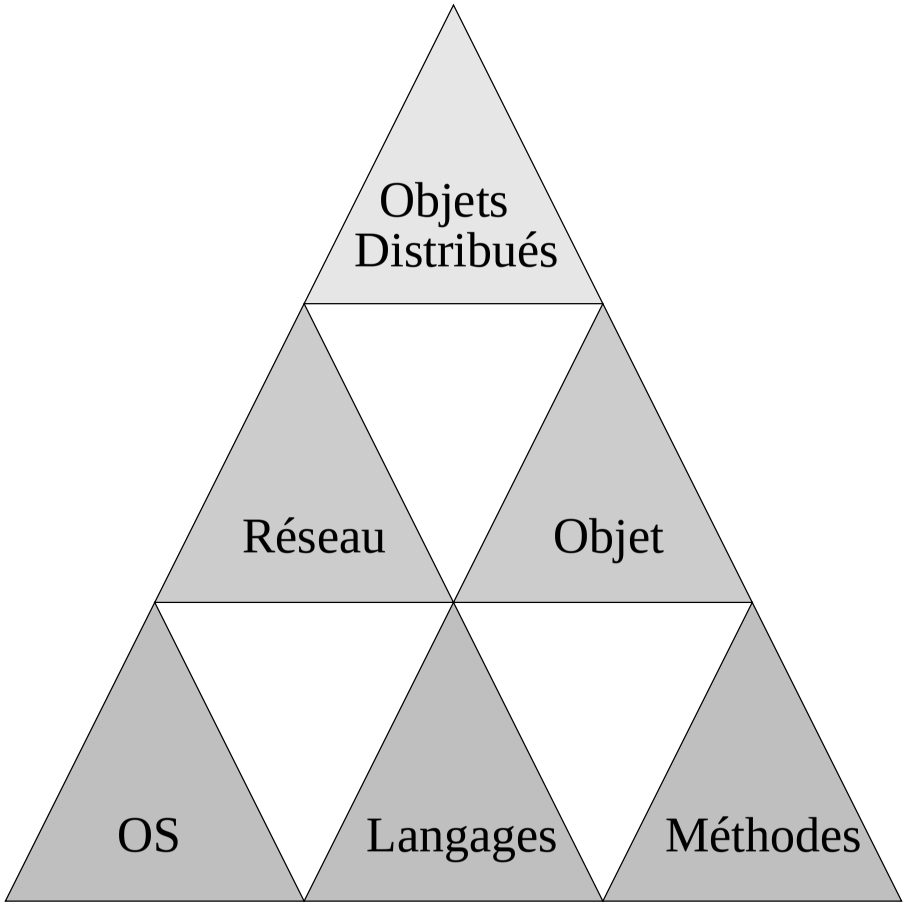


- Séparation entre interface et implantation ;
- Principe d'encapsulation ;
- Client = entité capable d'évoquer un service de l'objet ;
- Fournit un ou plusieurs services aux clients.

1.7.4 Conception et réalisation de logiciels

- Méthodologie de développement (OMT, UML) ;
- Bases de données (O^2) ;
- Bibliothèques de composants logiciels (GUI) ;
- Langages de programmation C++, Java . . . ;
- O.S. (Guide, Taligent, COM-OLE)
- Permet au niveau du G.L. :
  - Modularité, encapsulation $\mapsto$ extensibilité, extension
  - réutilisation, héritage aggrégation $\mapsto$ reduction des coûts

1.7.5 Les fondements des objets distribués



1.7.6 Les solutions Middleware objet proposées

Différentes solutions Middleware objet issues de différentes sociétés/organismes

- Microsoft : applicatifs bureautiques communiquant (OLE puis COM après accord avec Digital) étendus à des environnements distribués (DCOM) puis composants logiciels réactifs ou agissant sur l'applicatif les appelant (ActiveX).
- Sun : version légère d'objets distribués et d'invocations distantes, en Java spécifiquement (RMI). Développement de composants logiciels (Java Beans).
- Corba (Common Object Request Broker Architecture) : un modèle fédérateur autour d'un interfaçage client-serveur ouvert d'objets distribués interopérables.

Plan

2.1 Les protagonistes ... OMG
2.2 Modèle Client/Serveur
2.3 Architecture OMA
2.4 IDL
2.5 Requêtes à invocation statique (SII)
2.6 Référentiel d'interfaces (IFR)
2.7 Requêtes à invocation dynamique (DII)
2.8 Interface de squelette dynamique (DSI)
2.9 Adaptateur d'objets (OA)
2.10 Interopérabilité
2.11 Architecture

2.1 Les protagonistes ... OMG

OMG : Object Management Group

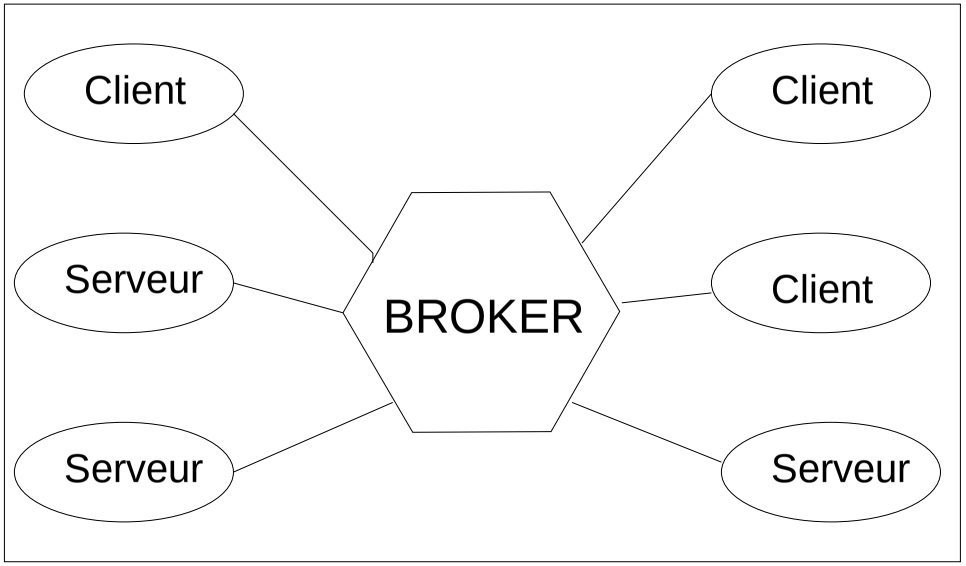
- Organisation non commerciale fondée en 1989 (USA)
- Aujourd'hui + de 850 membres :
  - constructeurs (IBM, SUN, HP, INTEL, ...),
  - éditeurs (Netscape, Inprise, Microsoft, ...),
  - utilisateurs (Boeing, Alcatel, NASA, ...),
  - laboratoires (INRIA, CERN, LIFL, ...)
  - ...

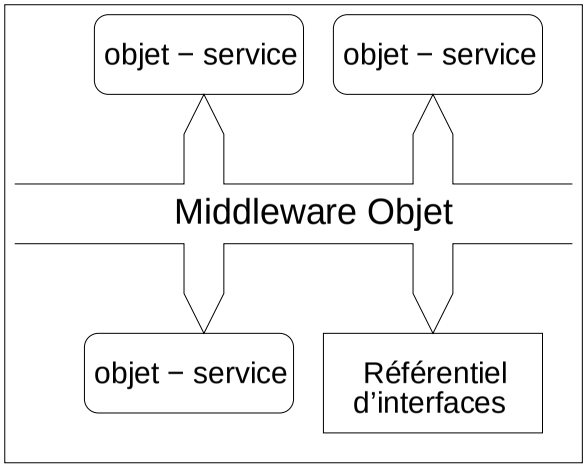
2.1 Les protagonistes ... OMG

OMG ... quels objectifs ?

- Constats : des approches objets incompatibles (BDs, langages, ...)
- Objectifs :
  - centralisation des spécifications émises par ses membres ;
  - pour développer une architecture unifiée ;
  - promotion des technologies objets ... jusqu'aux applicatifs de métiers ;
  - intégration des applications distribuées ;
  - spécifier des standards d'environnement logiciel supportant
    - réutilisabilité et portabilité des composants ;
    - hétérogénéité et interopérabilité entre différents architectures, OS et langages.

2.2 Modèle Client/Serveur





2.3 Architecture OMA

Object Management Architecture

Description globale des différents composants et technologies de Corba et leurs articulations :

- ORB ou Bus d'Objets ... transport des requêtes
- Objets d'applications définis spécifiquement
- Corba Services : services réutilisables de manipulation d'objets (cycle de vie, communications, recherche ...)
- Corba Facilities et Domains Services : canevas d'objets dédiés aux secteurs d'activités / métiers ou applicatifs transversaux.

2.3.1 Corba Services : les services objets

- Outils destinés aux développeurs d'applications distribuées
- Définis et spécifiés par l'OMG pour simplifier le développement en normalisant des services génériques

2.3.2 Corba Services : des exemples

- Service de noms : gestion d'une référence nominative unique pour chaque objet distribué.
- Service d'évènements : gestion d'évènements asynchrones au niveau des serveurs.
- Service cycle de vie : gestion des objets pendant leur fonctionnement (copier, déplacer ou supprimer).
- Service sécurité : gestion de l'authentification des clients, cryptage des données sur le réseau, certificats, ...

2.3.3 Corba Facilities : des utilitaires communs

- Se situent à un niveau d'abstraction plus élevé
- Outils utilisés par des applications (et plus des objets)
- Exemples : Gestionnaires d'impression, module de messagerie, outils de gestion documentaire . . .

2.3.4 Domain Services : des interfaces de domaines

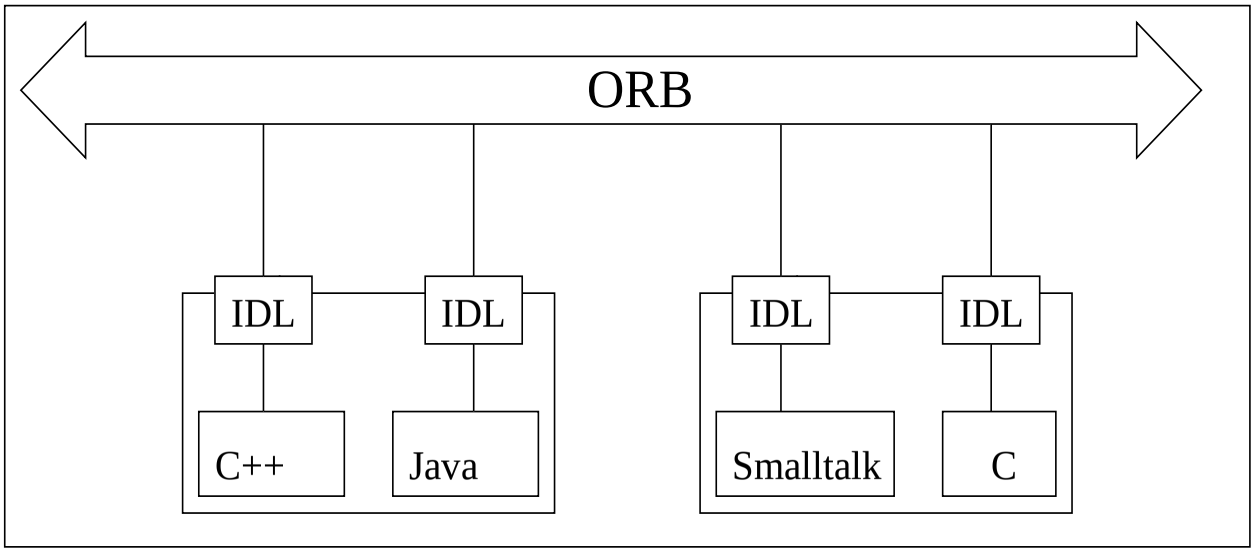
- Spécialisations métiers
- Objets utilitaires destinés à un secteur d'activité donné
- Exemples : gestion de cartes bancaires, ...

2.4.1 Langage d'interfaçage contractuel

- Nécessité d'un langage de description des interfaces contractuelles pour les services et requêtes :
- IDL (Interface Description Langage)
- Projection des interfaces IDL dans le langage des applicatifs
- Toutes les technologies Corba sont décrites en IDL

2.4.2 Langage pivot

Langage pivot entre les applications CORBA

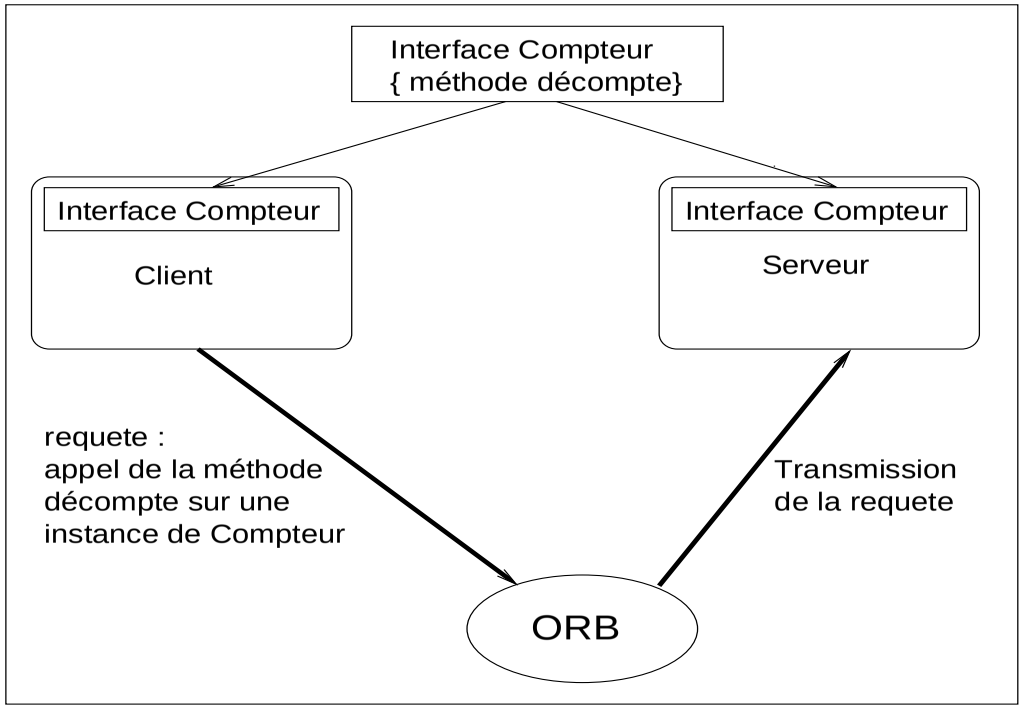


Problèmes de projection dans les langages : template C++, passage paramètres modifiables C/Java, exceptions C, . . .

Masque l'hétérogéneité

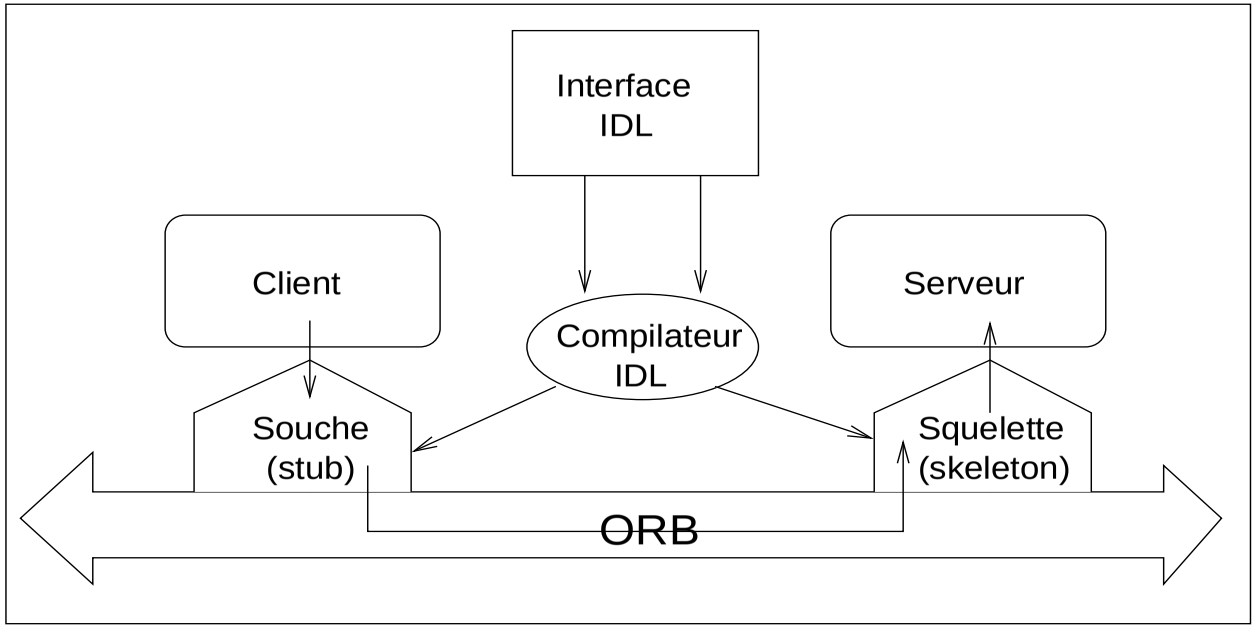
2.5 Requêtes à invocation statique (SII)

Une requête statique client/serveur interfacée :



2.5 Requêtes à invocation statique (SII)

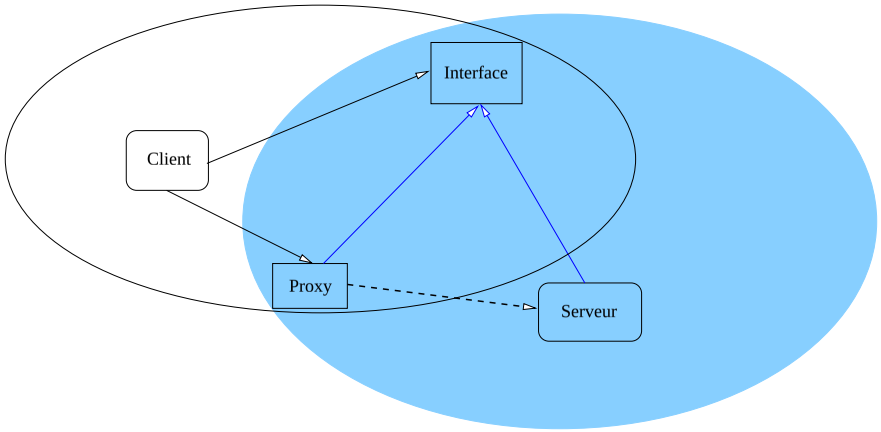
Une requête statique client/serveur interfacée :



2.5 Requêtes à invocation statique

(SII) Notion de proxy

Un objet dans un contexte est représenté dans un autre contexte par un proxy.



- Le proxy transmet les appels de méthodes ;
- stub=proxy.

2.6 Référentiel d'interfaces (IFR)

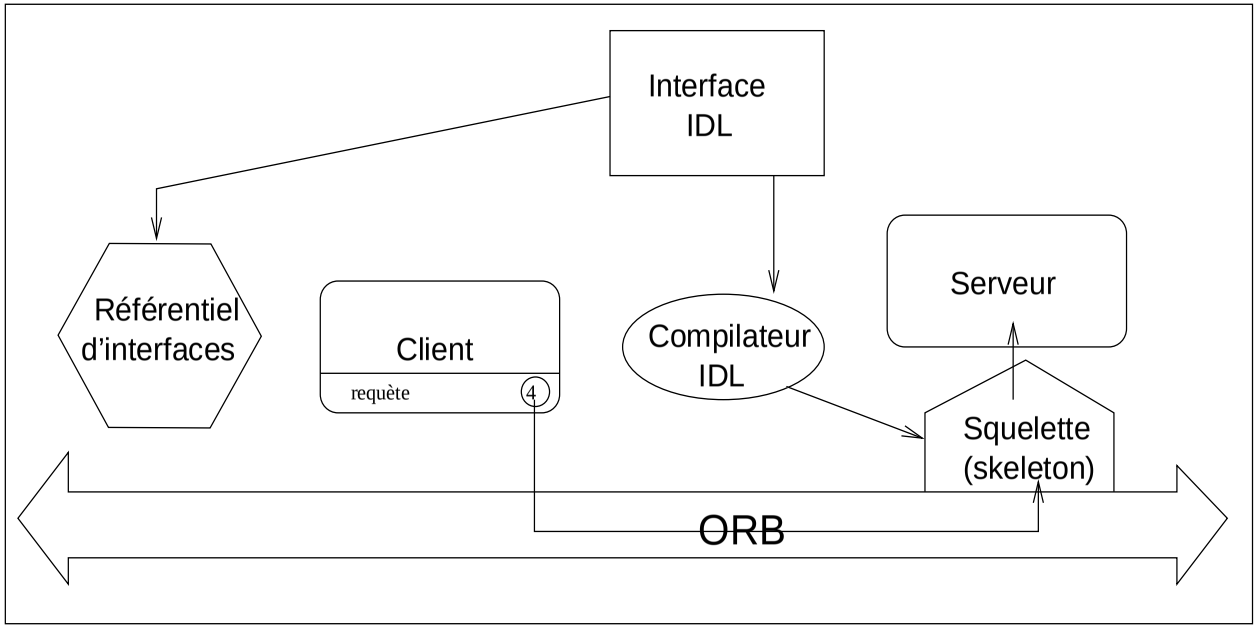
- Permet de stocker des archives d'interfaces IDL. L'IFR est accessible depuis l'ORB grâce à sa propre interface IDL (masquage de sa réelle implantation).
- Outils de manipulation : consultation, modification et navigation.
- Possibilité d'IFR multiples coopératifs et communiquants.

2.7 Requêtes à invocation dynamique (DII)

Ici, l'interface IDL du service invoqué n'est pas connu du client

- qui consulte et découvre l'interface en interrogeant l'IFR ;
- qui construit dynamiquement sa requête en cours d'exécution, puis en fait l'invocation.

2.7 Requêtes à invocation dynamique (DII)



2.8 Interface de squelette dynamique (DSI)

- Fait référence à des implémentations de services non interfacés ... plus vraiment d'objet Corba ! Le serveur fait croire à un tel objet en construisant dynamiquement une interface.
- Fonctionnement transparent pour le client.
- Permet l'intégration de serveurs existants dans un ORB.
- Permet les passerelles entre des ORB distincts ou l'interfaçage avec d'autres protocoles d'objets distribués.

2.9 Adaptateur d'objets (OA)

2.9.1 Son rôle

- Module de "connexion" du serveur sur l'ORB ;
- Créé ou active un object suite à une invocation ;
- Désactive un objet ;
- Assure la réception des requêtes auprès des objets et informe l'ORB du bon acheminement + sécurisation des échanges ;

2.9.2 Les différents types

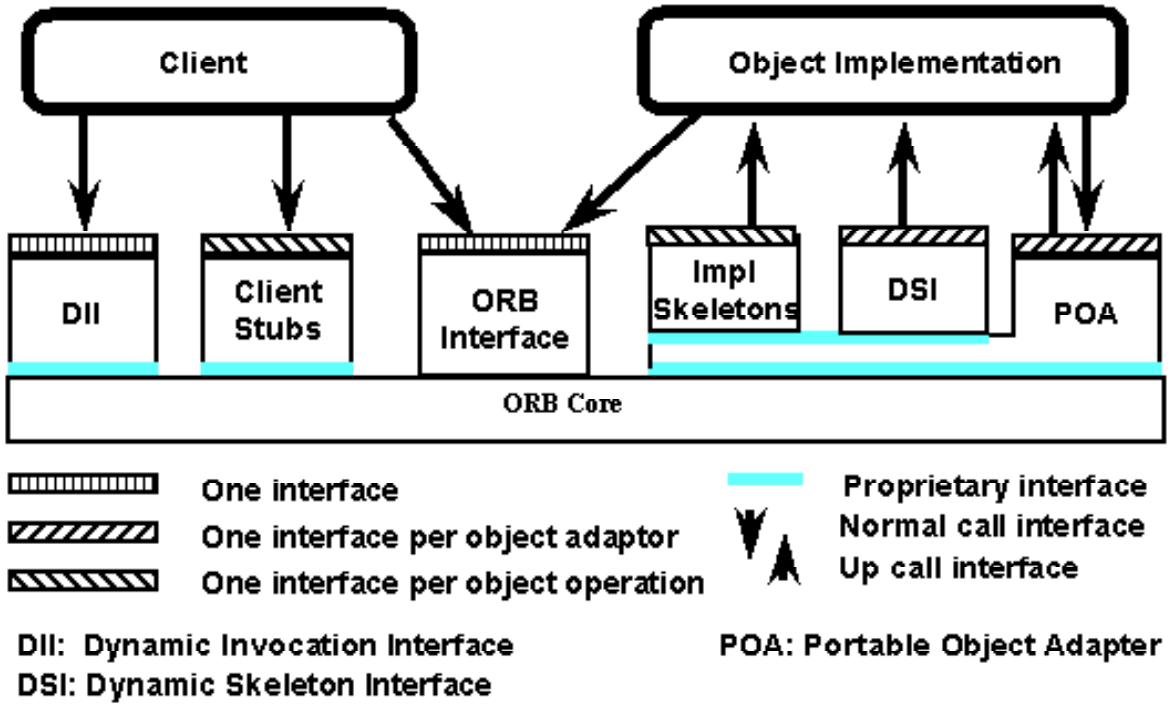
- BOA (Basic OA) OA de base initialement spécifié de manière incomplète
- POA (Portable OA) nouvel OA de base avec compléments de spécification
- OODA (Object Oriented Database Adapter) accès à des Bases de Données OO
- LOA (Library OA) accès à des objets stockés dans des bibliothèques logicielles.

2.10 Interopérabilité

Spécifie la gestion des communications entre ORB.
Précise les protocoles d'échanges utilisés :

- GIOP (General Inter-ORB Protocol) protocole générique avec définition commune des données, des références d'objets et des messages de communication.
- IIOP (Internet IOP) implémentation de GIOP au-dessus de TCP-IP
- ESIOP (Environment Specific IOP) pour répondre aux préoccupations de certains membres de l'OMG ... implémenté au-dessus de DCE/RPC devient DCE-ESIOP

2.11 Architecture

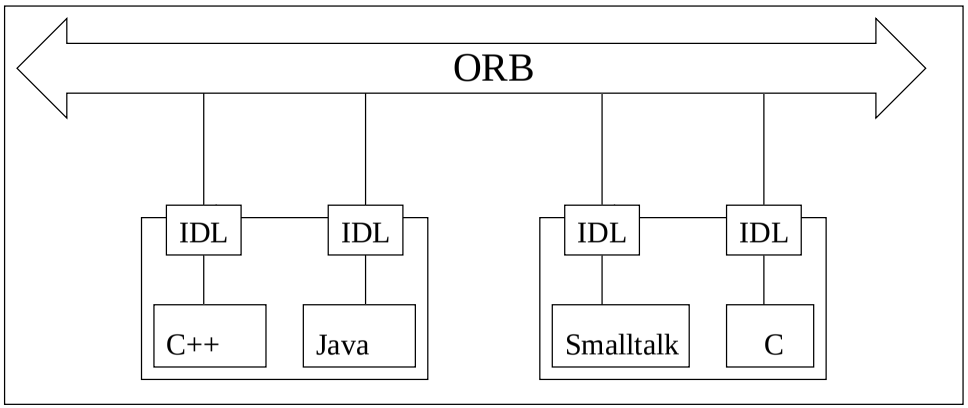


Plan

3.1 Les types
3.2 Arborescence des types IDL simples et structurés
3.3 Les types simples IDL
3.4 Les types structurés IDL
3.5 Constantes et alias
3.6 Interface d'objets
3.7 L'héritage
3.8 Compléments

3. IDL/CORBA

C’est le langage de description de toutes les technologies CORBA. Il permet notamment la description des interfaces des objets distribués. Ces interfaces seront ensuite projetées dans les langages des applicatifs.

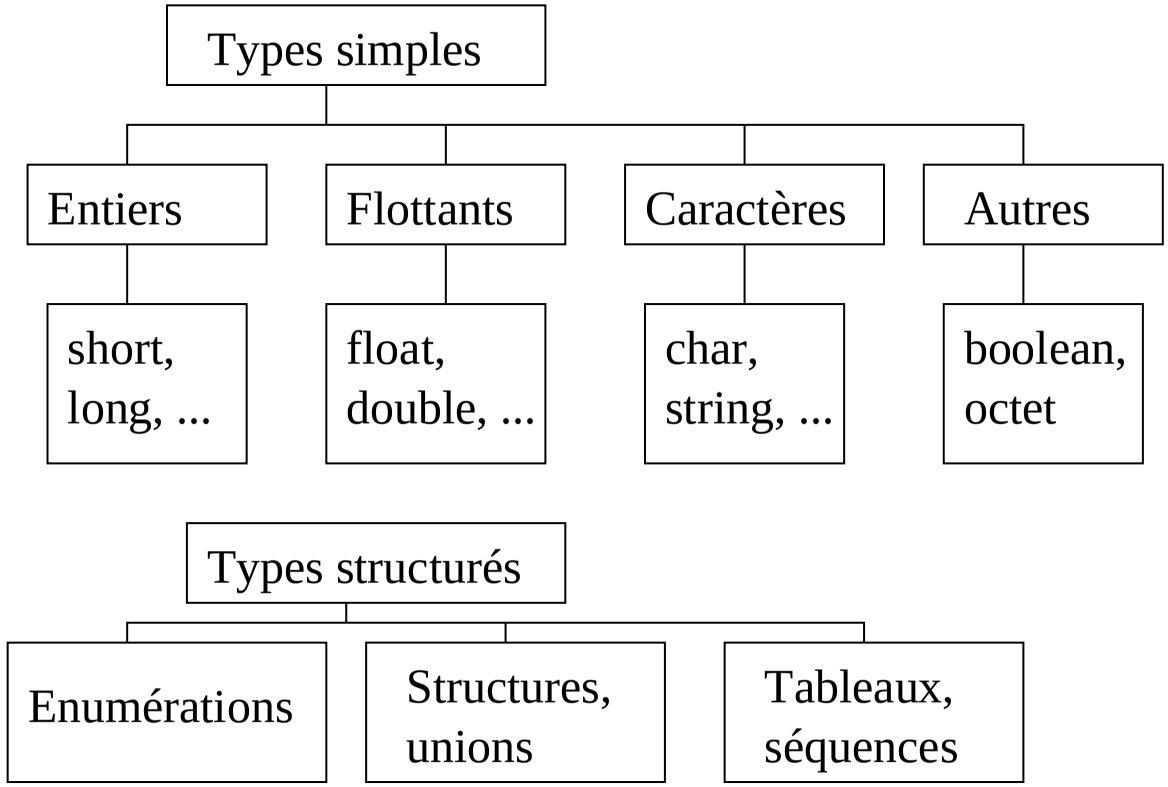


3.1 Les types

- types simples de base : voisins de ceux de Java
- types structurés : `enum, struct, union, array, sequence`
- types objets : `interface`
- types dynamiques et auto-descriptifs : `TypeCode, Any`

→ Redéfinition possible de types avec `typedef`

3.2 Arborescence des types IDL simples et structurés



3.3 Les types simples IDL

- short : entier 16 bits ;
- long : entier 32 bits ;
- float : décimal 32 bits IEEE ;
- double : décimal 64 bits IEEE ;
- char : caractère 8 bits ASCII standard ;
- string : chaînes de char ;
- boolean : variable booléenne ;
- octet : 8 bits sans représentation ;

Et les types plus exotiques

- wchar et wstring (à base de code ASCII étendu sur plusieurs octets par caractère) ;
- long long et long double (pas toujours de projection réelle...).

3.4 Les types structurés IDL

- Les énumérations : structure classique en C et C++ et inexistente en Java. La projection dans ce langage est ésotérique. Exemple :

```
enum Jours
  { Lundi, Mardi, Mercredi, Jeudi
    Vendredi, Samedi, Dimanche };
```

- Les structures : Elles ont un rôle classique de conteneur permettant de regrouper un ensemble de données quelconques IDL.

```
struct date
  { string jourSemaine;
    short jour; short mois; short annee; }
```

3.4 Les types structurés IDL

- Les unions : ce sont des structures de données qui contiennent un type de donnée déterminé par valeur d'un discriminant.
  Exemple :

```
union ExUnion switch (short v) {
   case 1  : long a;
   case 2  : string b;
   default : float c;
}
```

les unions - suite

- Un objet de ce type, sera du type long si v vaut 1, string si v vaut 2 et du type float sinon ou si v n'est pas donné.
- En C++, cette notion existe sans discriminant. En Java, elle n'existe pas et se projette en un objet avec des fonctions d'accès au discriminent et aux différentes données éventuelles.
- Nous n'en dirons pas plus car son intéret est limité : on utilisera le type dynamique any à la place de manière plus efficace et plus conventionnelle en CORBA.

3.4 Les types structurés IDL

- Les tableaux : On peut définir en IDL des tableaux sous la forme d'un alias (cf. suite).
  Exemple :

```
typedef float vecteur[10];
typedef vecteur matrice[10];
// ou encore typedef float matrice[10][10];
```

3.4 Les types structurés IDL

- Les séquences : ce sont des tableaux sans taille limite. Toutefois il est possible de borner la taille en l'indiquant dans sa description.
  Exemple :

```
sequence<float> vecteur;
sequence<long, 10> vecteur_borne;
```

3.5 Constantes et alias (1)

Les constantes : On peut définir des constantes en IDL, comme en C/C++.
Exemple :

```
const float PI=3.14;
```

La traduction en Java correspond simplement à un champ particulier de la classe dans laquelle la constante est définie. Si la constante IDL est définie hors classe, alors on créé une classe spécifique Java portant le nom de la constante IDL et ayant un champ `value` du type de la constante.

3.5 Constantes et alias (2)

les alias : il s'agit du `typedef` du C/C++.
Exemple :

```
typedef sequence<float> vecteur;
```

En Java, cette construction n'existe pas et la projection correspondante revient à la recopie du type aliasé !

3.6 Interface d'objets

Permet la description des classes d'objets à partir :

- de la (ou des) classe(s) dont elles héritent
- des noms et types des attributs
- des en-têtes des méthodes

$\rightarrow$ Des interfaces peuvent être réunies dans un même module.

3.6 Interface d'objets

Exemple :

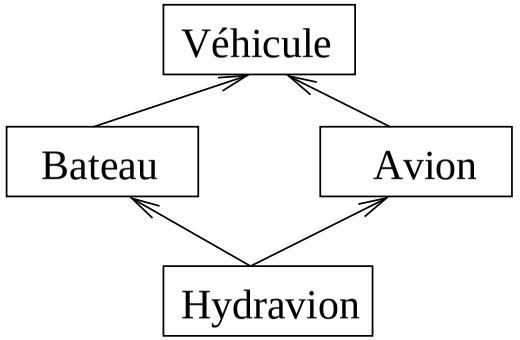
```
module Vehicules {
   interface QuatreRoues { ... }
   interface Voiture: QuatreRoues {
      attribute float poids;
      readonly attribute short puissance;
      void accelerer (in int arg);
   }
}
```

3.6 Interface d'objets

- attribute
  - permet l'accès à des propriétés de l'objet
  - sa projection génère une paire de méthodes : accès à l'attribut/ modification de l'attribut (sauf si `readonly`)
- 3 modes de passages de paramètres
  - **in** : accès en lecture dans la méthode appelée
  - **out** : en retour uniquement vers la méthode appelante
  - **inout** : les deux

3.7 L'héritage

Héritage multiple autorisé mais surcharge ou redéfinition de noms interdits (éviter conflit).




```
interface Vehicule {...}
interface Avion :
  Vehicule { ...}
interface Bateau :
  Vehicule {...}
interface Hydravion :
  Avion, Bateau {...}
```

3.8 Compléments

- Visibilité
  un identifiant est reconnu comme l'identifiant du bloc de description le plus proche, dans l'ordre : le bloc courant, les blocs hérités puis les blocs supérieurs.
- Exceptions
  CORBA étend le mécanisme d'exception : une exception peut être générée dans un premier programme (p.e. le serveur) et être interceptée dans un second programme (p.e. le client).

3.8 Compléments

- Oneway

Un appel d'une méthode est bloquant par défaut. En qualifiant la méthode de Oneway, on rend cette méthode non bloquante et on autorise la suite du déroulement du programme sans attendre la fin de son exécution. Attention ! à utiliser avec précautions.