Le standard CORBA

Département informatique et réseaux

Frank Singhoff

C-214

singhoff@enst.fr

Introduction

CORBA[a] est un standard normalisant l'accès à des objets répartis avec des objectifs :

- D'interopérabilité, de portabilité et de réutilisabilité.
- De transparence à la localisation.

et définissant principalement :

- Une architecture standard (réutilisabilité).
- Un langage de description des objets (l'IDL[b]) ainsi que sa correspondance avec le langage utilisé pour l'implantation des objets.
- Un protocole assurant l'interopérabilité.

[a] **CORBA** pour **C**ommon **O**bject **R**equest **B**roker **A**rchitecture.

[b] **IDL** pour **I**nterface **D**efinition **L**anguage.

Historique

Standard défini par l'OMG [a].

Consortium composé de 800 membres, créé en 1989.

Standards importants produits depuis sa création :

- Corba 1.2 en 1993[OMG 93].
- Corba 2.0 à 2.3 depuis 1995[OMG 99].

[a] **OMG** pour **O**bject **M**anagement **G**roup.

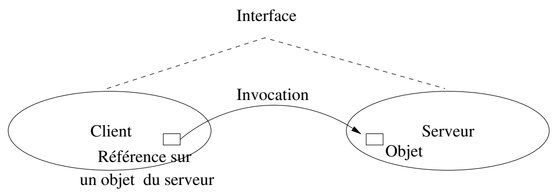
Sommaire

Partie 1

Concepts de base

et
architecture

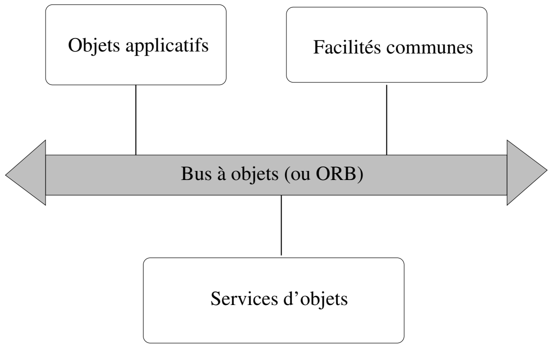
Le modèle objet de CORBA



- Notions de :
  - Objets, interfaces, opérations, implémentations, références d'objet.
  - Séparation de l'interface et de son implémentation. Le client ne connait que l'interface.
  - Requêtes (référence + opération + paramètres + exceptions + contexte). Mécanisme synchrone.
  - Sémantique d'invocation (exactement une fois ou au plus une fois).

L'architecture OMA (1)

OMA pour **O**bject **M**anagement **A**rchitecture.



L'architecture OMA (2)

– ORB (Object Request Broker) ; ou bus à objets. Fournit les fonctionnalités nécessaires à l'acheminement des invocations.

– Les services d'objets fournissent des fonctionnalités de bas niveaux.

– Les objets applicatifs sont les objets développés par les utilisateurs de l'architecture.

– Les facilités sont des objets intermédiaires entre les services et les objets applicatifs.

Et concrétement (1)

☞ Un ORB peut être :

– Une simple bibliothèque.

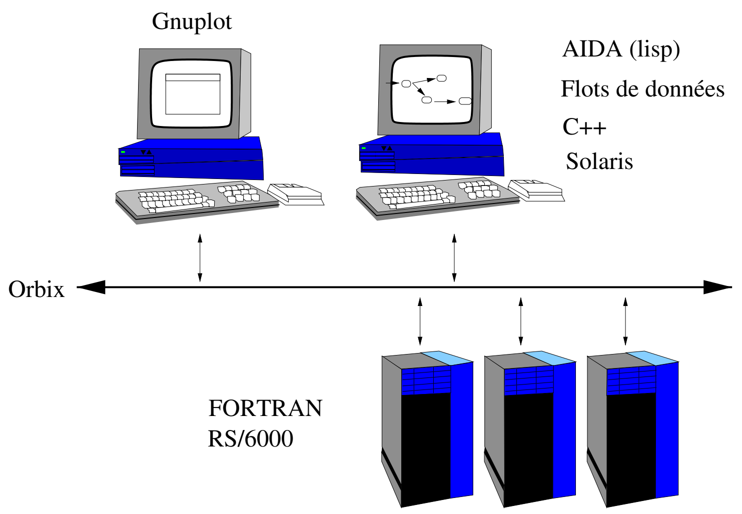
– Un ou des processus.

– Un système d'exploitation.

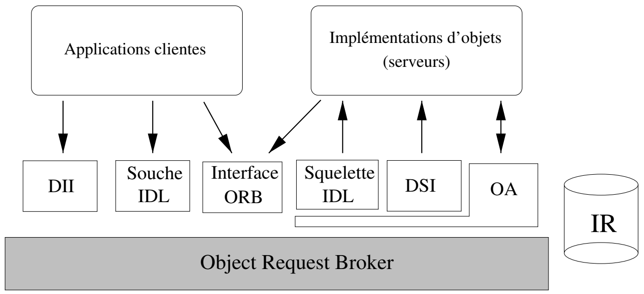
– Etc.

☞ Un client, un objet, un serveur peuvent être ...

Et concrétement (2)



Structure d'un bus à objets (1)



Structure d'un bus à objets (2)

- Interface de l'ORB. Offre des services de base communs aux clients et aux serveurs.
- Souche et squelette IDL. Invocation statique des objets.
- OA (Object Adapter). Gestion des implémentations d'objets et des serveurs.
- DII (Dynamic Invocation Interface). Interface d'invocation dynamique.
- DSI (Dynamic Skeleton Interface). Squelette dynamique d'invocation.
- Interface Repository. Référentiel des interfaces d'objets.
- Implementation Repository. Référentiel des implémentations d'objets.

Partie 2

Le langage de description d'interface

L'IDL : présentation (1)

Objectif : permettre une description des objets de façon indépendante du langage d'implantation et du système sous-jacent.

- *Dans CORBA, tout est IDL !* Notion d'objets et de pseudo-objets.
- Très proche du C++.
- CORBA définit les règles de correspondance de l'IDL vers les langages cibles (*mapping*). Mapping normalisé pour C, C++, ADA, Smalltalk, COBOL, JAVA.
- Utilisation de préprocesseurs possible. Invocation statique $\Longrightarrow$ compilation pour générer les souches et squelettes.

L'IDL : présentation (2)

Ce qui est défini en IDL :

- Des modules. Unités de structuration (opérateur de portée ::).
- Des constantes.
- Des interfaces.
- Des opérations.
- Des types construits à partir de types simples prédéfinis.
- Des exceptions utilisateurs. Les objets peuvent lever des exceptions et retourner des informations.

L'IDL : présentation (3)

- Un mapping ? pourquoi faire ? $\Longrightarrow$ API portable de l'utilisateur.
- Difficultés :
  - Abstractions offertes par le language ciblé (ex : COBOL, C) et particularités fonctionnelles.
  - Portabilité du language (ex : C++).
- Ce que normalise le mapping :
  - Spécifications IDL des objets (types, exceptions, constantes, etc).
  - Pseudo objets CORBA (ORB, BOA, DII, etc).
  - Les souches et squelettes.

Les interfaces

☞ Unité représentant un point d'accès à un objet. Héritages multiples sous certaines contraintes (ex : pas de surcharge de méthodes).

☞ Est constituée de :

- Types, exceptions.
- Attributs (option *readonly*).
- Opérations.

Les opérations

☞ Opération = identificateur + paramètres + valeur de retour + sémantique d'invocation + exceptions + contexte :

- Paramètres *in*/*out*/*inout*.
- Sémantique d'invocation. Méthodes *oneway* = au plus une fois. Exactement une fois sinon (sauf si exception levée : au plus une fois dans ce dernier cas).

! *oneway* ≠ invocation asynchrone.

Types simples et construits (1)

Types simples :

- *long, short, char, double, unsigned, boolean.*
- *string.* Chaîne de caractères bornée ou non.
- *octet.* Information de 8 bits sans conversion.
- *Object.* Désigne une référence d'objet CORBA.
- *any.* Désigne n'importe quels types IDL.

$\Longrightarrow$ Extensions mineures à partir de CORBA 2.1

Types simples et construits (2)

Constructeurs de type :

- opérateur *typedef.*
- *struct.* Structure, identique au langage C/C++.
- *enum.* Énumération ordonnée.
- *union.*
- *sequence* et tableaux. Séquence = tableau à une dimension d'un type quelconque. Séquence bornée ou non. Notion de taille maximale et de taille courante.
- *interface.* Objet fils de *Object.*

Exemple

```
module polonaise {

  exception erreur {long codeErreur;};
  enum ope {add, sous, div, mult};

  struct operande {
    string base;
    string element;
  };
  typedef sequence<operande>operandes;

  interface calculatrice
    {
    const long maxPile = 10;
    readonly attribute long taillePile;

    long empileD(in operandes donnee) raises (erreur);
    long empileO(in ope operation) raises (erreur);
    long depileR(out operande res) raises (erreur);

    oneway void videPile();
    };
};
```


Partie 3

Zoom sur le bus à objets

Zoom sur le bus à objets

1. Les services du noyau de l'ORB.
2. Les adaptateurs d'objets et leurs interactions avec les serveurs.
3. Invocation statique et dynamique.

Services du noyau de l'ORB (1)

☞ Fonctions communes aux clients et aux serveurs, et qui concernent :

- L'initialisation de l'environnement CORBA.
- La gestion des références d'objets.
- L'accès aux objets qui fournissent des services de base (service de nom, référentiel d'interface).
- La gestion des pseudo-objets (ex : *NVList, Request, Context, Environnement, NamedValue, ORB, OA, TypeCode, Principal*).

Services du noyau de l'ORB (2)

```
module CORBA {

  ORB ORB_init(in string argv, in ORBid orb);

  pseudo interface ORB {
    string object_to_string(in Object obj);
    Object string_to_object(in string str);
    BOA BOA_init(in string argv, in BOAid boa);
    Object resolve_initial_references(in
        Objectid ident);
    create_{NVList, NamedValue, Context};
    };

  pseudo interface Object {
    boolean is_nil();
    boolean is_a();
    Object duplicate();
    void release();
    interfaceDef get_interface();
    implementationDef get_implementation();
    create_Request(...);
    };
};
```


Les adaptateurs d'objets (1)

Fonctions d'un adaptateur = protocole ORB/Serveurs :

- Gestion des implémentations d'objets et des serveurs. Serveurs $\neq$ objets.
- Activation, désactivation et invocation des objets.
- Sécurité.

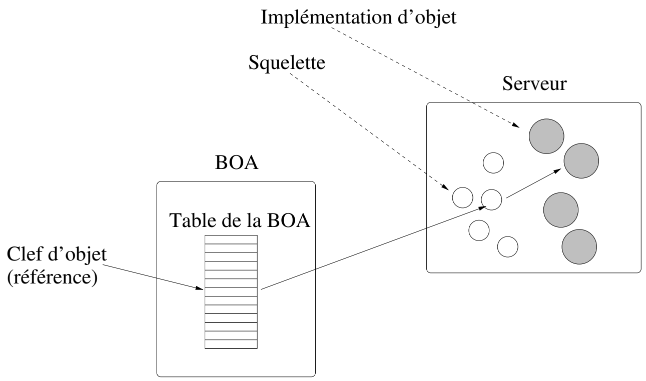
$\Longrightarrow$ Historique : BOA[a] puis POA[b].

[a] **BOA** pour **B**asic **O**bject **A**daptor.
[b] **POA** pour **P**ortable **O**bject **A**daptor.

Les adaptateurs d'objets (2)

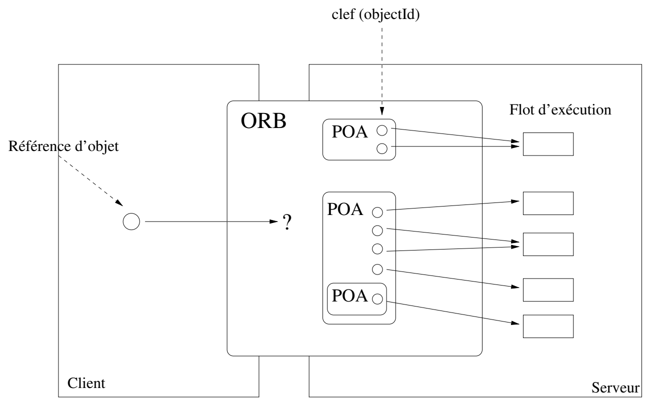
```
module CORBA {

  pseudo interface BOA {
    void dispose(in Object obj);
    void change_implementation(in Object obj,
        in ImplementationDef impl);
    void impl_is_ready(in ImplementationDef impl);
    void deactive_impl(in ImplementationDef impl);
    void obj_is_ready(in Object obj,
        in ImplementationDef impl);
    void deactive_obj(in Object obj);
    }

}
```


Les adaptateurs d'objets (3)



- Applications peu portables car spécifications peu précises : pas de notion précise de *thread*, de serveurs, de mode d'activation.
- Choix de la politique d'activation laissé aux implanteurs d'ORB.

Les adaptateurs d'objets (4)



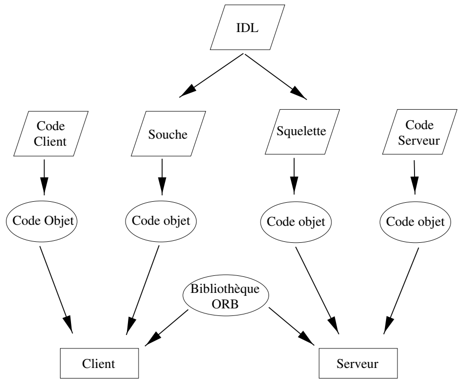
- Notion de politique : un POA = une politique donnée.
- Doit permettre une meilleure portabilité des serveurs : un serveur nécessite alors une POA dont la spécification est connue.
- POA encore peu utilisé aujourd'hui.

Les adaptateurs d'objets (5)

☞ Exemples de politique :

- Persistence, mode d'activation des objets et serveurs (automatique ou non, objet de courte durée de vie), choix de l'exécutant (pool, gestion de ressource).
- Modèle de thread.
- Comportement lors d'une invocation (recherche de l'objet, filtre).
- Gestion des POA (structuration de l'application).
- ...

Invocation statique (1)



- Interfaces des objets connues à la compilation. Contrôle de type.
- Certainement la solution la plus utilisée, car simple et efficace. Invocation d'un objet CORBA identique à un objet quelconque.

Le standard CORBA

Invocation statique (2)

Fonctions des souches :

- Emballage des invocations.
- Transmission vers le serveur.

Fonctions des squelettes :

- Déballage des invocations.
- Permettre aux OA d'invoquer les méthodes concernées (un squelette fournit les *upcalls* vers l'implémentation).

Invocation dynamique (1)

Caractéristiques :

- Plus coûteuse (gestion mémoire, contrôle en ligne du type des arguments).
- Permet d'exploiter des interfaces non enregistrées lors de la compilation du client (ex : pont, débogueur).
- Sémantique d'invocation identique, mais mécanismes plus riches (invocations non bloquantes, en parallèles).

Composants de l'ORB utilisés : le référentiel d'interfaces, DII, DSI.

Invocation dynamique (2)

Le référentiel d'interfaces fournit des informations sur les interfaces.

- Référentiel = base de données (arborescente).
- Utilisateur du référentiel : les clients et serveurs, les outils de compilation IDL, les outils d'administration, l'ORB lui-même (par des ponts CORBA), etc.
- L'OMG définit les interfaces IDL permettant l'accès et le stockage des informations du référentiel. L'accès peut se faire par l'ORB ou directement par le référentiel (par identifiant unique ou par parcours de l'arborescence).

Invocation dynamique (3)

☞ Invocation d'une méthode :

- A partir d'une référence d'objet, obtenir de l'ORB un pseudo-objet *request*.
- Construire la requête (nom de la méthode, liste des paramètres, sémantique, etc.) par le biais d'autres pseudo-objets (*NamedValue*, *NVList*, *Context*). Les informations sur la méthode peuvent être extraites du référentiel d'interfaces.
- Invoquer la méthode à l'aide de la DII.

Invocation dynamique (4)

☞ Mécanismes d'invocation : invocation synchrone ou synchrone différée, invocations multiples.

☞ Dans le cas différé :

- Réception bloquante ou non.
- Scrutation active (*polling*). Pour les invocations multiples, l'ordre de réception n'est pas déterminé.

Invocation dynamique (5)

```
module CORBA {

  pseudo interface Request {
    Status add_args(...);
    Status invoke(in Flags f);
    Status send(in Flags f);
    Status get_response(in Flags f);
  };

  pseudo interface Object {
    create_Request(...);
    interfaceDef get_interface();
    ...
  };

  pseudo interface ORB {
    Status send_multiple_request(in seqRequest req,
      in Flags f);
    Status poll_next_response(void);
    Status get_next_response(out seqRequest req);
    ...
  };
};
```


Invocation dynamique (6)

☞ La DSI est le pendant de la DII.

Objectif : permettre la prise en compte dynamique de squelette $\Longrightarrow$ ex : pont CORBA, gestion de documents, etc.

☞ Deux phases :

- Création du squelette et enregistrement.
- A la réception d'un pseudo-objet *Request*, invocation d'un DIR[a] (un par objet).

• Points importants : toute liberté est donnée sur les techniques de productions, d'enregistrement et sur le contenu des squelettes.

[a] **DIR** pour **D**ynamic **I**mplementation **R**outine.

Partie 4

CORBA et l'interopérabilité

CORBA et l'interopérabilité (1)

☞ Objectif = étendre l'interopérabilité entre des ORB différents, voir des systèmes objets "non CORBA".

- Que faut-il pour interopérer ?

1. Manipuler un modèle objet identique.
2. Être capable d'interpréter convenablement une référence d'objet (ex : localiser son implémentation).
3. Être capable de véhiculer les invocations et les exceptions.
4. Manipuler une représentation communes des types, indépendamment des systèmes sous jacent (condition nécessaire dans un environnement constitué d'un seul ORB).

CORBA et l'interopérabilité (2)

☞ Solution apportée dans CORBA 2

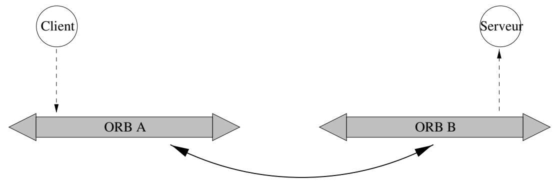
(CORBA 1 normalise uniquement l'interface de programmation). La version 2 définit :

- Des mécanismes de ponts.
- Un protocole standard : GIOP [a], ainsi que sa correspondance sur TCP/IP : IIOP [b].
- Des environnements spécifiques (ex : interopérabilité avec DCOM ou par DCE).

☞ Compatible CORBA 2 = API CORBA + IIOP.

[a] **G**IOP pour **G**lobal **I**nter-**O**RB **P**rotocol.
[b] **IIOP** pour **I**nternet **I**nter-**O**RB **P**rotocol.

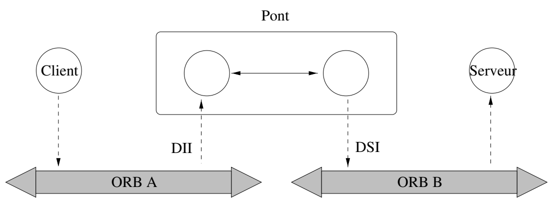
CORBA et l'interopérabilité (3)



☞ Utilisation native de IIOP.

- Interopérabilité d'office.
- Le plus courant dans les ORB commercialisés aujourd'hui.

CORBA et l'interopérabilité (4)



– Permet la mise en œuvre de protocoles optimisés, adaptés à certains types d'environnements ou d'applications.

– Conservation de l'existant.

– Le pont mémorise des objets *proxies*.

– Conversion de protocole, de référence d'objet, voir de modèle objet.

• Combinaison de ponts + IIOP en fond de panier.

Les références d'objet (1)

Référence d'objet = type opaque pour l'utilisateur. Dans un même domaine de référence, les différents ORB doivent pouvoir interpréter les références d'objets $\Longrightarrow$ utilisation d'IOR[a] :

Une référence d'objet c'est avant tout une information sur une extrémité, vis-à-vis d'un protocole (ex : adresse IP + port + clef d'objet).

[a] **IOR** pour **I**nteroperable **O**bject **R**eference.

Les références d'objet (2)

```
module IOP {
  typedef unsigned long ProfiledId;
  struct TaggedProfile {
    ProfileId tag;
    sequence<octet>profile_data;

  struct IOR {
    string type_id;
    sequence<TaggedProfile>profiles;
  };
};
```

- L'OMG n'impose pas l'utilisation des IOR pour les communications internes d'un ORB.
- Un IOR peut il traverser plusieurs protocoles de transport ? En théorie oui, en pratique, ca reste peu utile !

Présentation de GIOP (1)

Protocole permettant la communication entre deux ORB CORBA potentionellement différents, afin de :

- Transmettre les invocations d'objets et les exceptions.
- Offrir un service de localisation pour implanter des outils de migration d'objets.
- Assurer le support d'opérations sur des références d'objet (*Object::get_interface*(), *Object::get_implementation*(), *Object::is_a*(), et *Object::non_existent*()).

$\Longrightarrow$ Support de GIOP en "natif" ou sous forme de pont.

Présentation de GIOP (2)

Caractéristiques :

- Protocole asymétrique ; notion de client et de serveur (ex : un ORB ou un pont).
- Protocole orienté connexion. Le client est l'initiateur des connexions. Le serveur ne peut pas en ouvrir. Multiplexage autorisé.
- Emballage CDR [a] des messages.
- Nécessite l'utilisation d'une couche transport fiable. Gestion des déséquencements de messages non nécessaire.
- Sept types de messages seulement car protocole minimal = surcoût minimal.

[a] **CDR** pour **C**ommon **D**ata **R**epresentation.

Les messages GIOP (1)

Messages constitués de trois parties :

- Entête GIOP.
- Entête spécifique au type de message.
- Corps de message optionnel.

```
module GIOP {
  struct MessageHeader {
    char magic[4];
    Version GIOP_version;
    boolean byte_order;
    octet message_type;
    unsigned long message_size;
  };
```

Les messages GIOP (2)

Émission d'une invocation.

```
module GIOP {
  struct RequestHeader {
    IOP::ServiceContextList service_context;
    unsigned long request_id;
    boolean response_expected;
    sequence<octet>object_key;
    string operation;
    Principal requesting_principal;
  };
};
```

- Le corps optionnel est constitué d'une structure comportant les paramètres *in* et *inout*.

Les messages GIOP (3)

Transfert des résultats chez le client.

```
module GIOP {
  enum ReplyStatusType {
    NO_EXCEPTION, LOCATION_FORWARD,
    USER_EXCEPTION, SYSTEM_EXCEPTION
  };

  struct ReplyHeader {
    IOP::ServiceContextList service_context;
    unsigned long request_id;
    ReplyStatusType reply_status;
  };
};
```

- Le corps optionnel de la réponse (exécution réussie, exception levée ou objet absent).
- LOCATION_FORWARD transparent pour l'application. Réémission par le client GIOP.

Les messages GIOP (4)

Gestion de la localisation d'objet :

```
module GIOP
  enum LocateStatusType {
    UNKNOWN_OBJECT, OBJECT_HERE,
    OBJECT_FORWARD
  };
  struct LocateRequestHeader {
    unsigned long request_id;
    sequence<octet>object_key;
  };
  struct LocateReplyHeader {
    unsigned long request_id;
    LocateStatusType locate_status;
  };
};
```

- Fonctionnalité transparente à l'application.
- Corps optionnel composé d'un IOR.

Les messages GIOP (5)

Autres messages :

- Annulation par le client d'une requête (CancelRequest).
- Fermeture d'une connexion (CloseConnection).
- Erreur de protocole (MessageError).

Le protocole IIOP

☞ Implantation de GIOP sur TCP/IP (TCP).

- Description de profil GIOP pour IIOP (valeur du champs tag = TAG_INTERNET_IOP) :

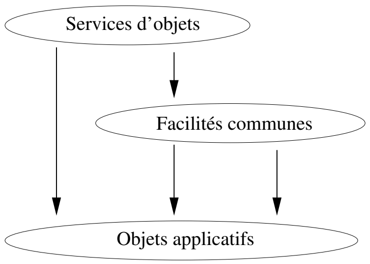
```
module IIOP{
  struct Version {
    char major;
    char minor;
  };

  struct ProfileBody {
    Version iiop_version;
    string host;
    unsigned short port;
    sequence<octet>object_key;
  };
};
```


Partie 5

Les services et facilités CORBA

Les services et facilités CORBA (1)

- Les facilités communes[OMG 97a] et les services d'objets [OMG 97b] sont des composants logiciels. Objectif : alléger la réalisation d'applications CORBA.



  - Limite mouvante entre les trois groupes de composants.
  - Service/facilité = interfaces + implémentations.
  - Raffinement possible (par héritage, par modification d'implémentations).

Les services et facilités CORBA (2)

☞ Les facilités sont classées en deux groupes :

- Les facilités horizontales, qui sont des composants pouvant servir dans plusieurs domaines d'applications (ex : facilité d'impression).
- Les facilités verticales, qui sont spécifiques à un domaine = objets métiers (ex : paye, gestion commercial, gestion hospitalière).

Les services et facilités CORBA (3)

Pas de classement pour les services d'objets, qui sont des objets nécessaires à la construction des applications. Exemples de services normalisés à ce jour :

- Service de nom, d'événements, de cycle de vie.
- Service de persistence.
- Service de sécurité.
- Service transactionnel.
- Service de contrôle de la concurrence.
- Service de temps.
- etc.

Exemples de services

1. Le service de noms.
2. Le service de cycle de vie.
3. Le service d'événements.

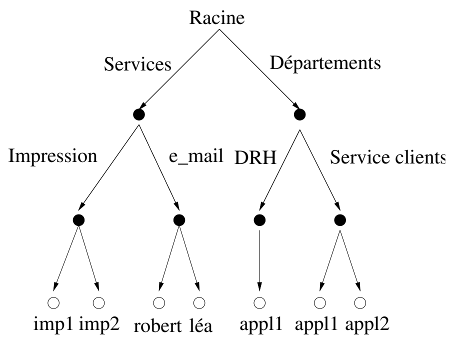
Le service de noms (1)

Permet aux serveurs d'associer un nom symbolique à un objet. Les clients utilisent alors ces noms symboliques pour obtenir des références sur les objets en question (mécanismes de liaison/résolution).

Ce service introduit les notions de :

- Nom symbolique. Un nom est unique dans un espace de nommage.
- Contexte, qui définit un espace de nommage. Les contextes peuvent être concaténés.

Le service de noms (2)



- Contextes + noms = arborescences.
- Feuilles = objets ; nœuds = contextes.
- Fonctionnement proche d'un système de fichiers hiérarchique.

Le service de noms (3)

```
module cosNaming {

  interface NamingContext {
    NamingContext bind_new_context(in Name n);
    void bind(in Name n, in Object obj);
    void unbind(in Name n);
    void bind_context(In Name n,in NamingContext nc);
    Object resolve(in Name n);
    void list(out BindingIterator i, ...);
    ...
  };

  interface BindingIterator {
    boolean next_one(out Binding b);
    ...
  };
};
```


Le service de cycle de vie (1)

Offrir des outils de gestion des objets CORBA.

- Opération de création :
  - Utilisation d'usines à objets spécifiques.

    ```
    interface doc;
    interface usine_docs {
      doc create_doc(in param ...);
    };
    ```

  - Usines à objets génériques $\Longrightarrow$ politiques de gestion des ressources.
- Opérations de suppression, copie ou migration d'objets (ou de graphes d'objets).

Le service de cycle de vie (2)

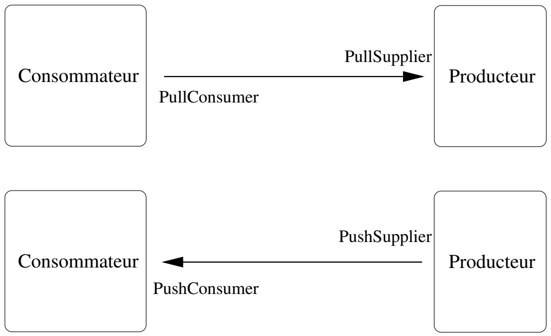
```
module CosLifeCycle {
  interface FactoryFinder {
    Factories find_factories(in Key factory_key);
  };

  interface LifeCycleObject {
    LifeCycleObject copy(in FactoryFinder there,
      in Criteria c);
    void move(in FactoryFinder there, in Criteria c);
    void remove();
  };

  interface GenericFactory {
    Object create_object(in Key k, in Criteria c);
    ...
  };
};
```


Le service d'événements (1)

CORBA offre un modèle d'invocation principalement synchrone $\Longrightarrow$ ce service doit permettre l'échange asynchrone d'événements de producteurs vers des consommateurs.

- Service très populaire par sa souplesse.
- La spécification laisse une large liberté quant aux différentes implémentations possibles.
- Combinaisons possibles :
  - Initiateur des communications.
  - Événements "génériques" ou typés.
  - Utilisation de canaux.

Le service d'événements (2)

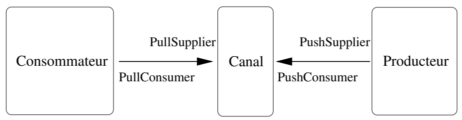


- Communications à l'initiateur du producteur (*push*) ou du consommateur (*pull*).
- Échange des références d'objets, identités connues.
- Ne permet pas le découplage producteur/consommateur.

Le service d'événements (3)

```
module CosEventComm {
  interface PushConsumer {
    void push(in any event);
    void disconnect_push_consumer();
  };

  interface PushSupplier {
    void disconnect_push_supplier();
  };

  interface PullSupplier{
    any pull();
    any try_pull(out boolean has_event);
    void disconnect_pull_supplier();
  };

  interface PullConsumer {
    void disconnect_pull_consumer();
  };
};
```

Le service d'événements (4)



- Canal = objet CORBA permettant de découpler les communications entre producteurs/consommateurs.
- Utilisation de filtres. Combinaisons de canaux. QoS non spécifiée.
- Identités inconnues.
- Interfaces permettant de créer, détruire des canaux, connecter des producteurs/consommateurs.
- Événements typés possibles.
- Communication n vers n possible.

Partie 6

Conclusion

Synthèse, conclusion, bilan (1)

- Synthèse :

1. Modèle d'objets répartis :
   - Transparence à la localisation, intégration, portabilité, interopérabilité.
   - Séparation interface et implémentation. Réference. IDL et Mapping.
   - Mécanisme synchrone. Sémantique exactement une fois ou au plus une fois.
2. Interopérabilité :
   - Les protocoles GIOP et IIOP.
   - Notion de référence interopérable d'objet.
   - Format de données (CDR).
   - Considérations architecturales.
3. Exemples de services.

Synthèse, conclusion, bilan (2)

- Bilan sur les objectifs :

1. Transparence à la localisation (oui).
2. Interopérabilité (oui).
3. Intégration de systèmes, langages et réseaux. (oui)
4. Portabilité (oui du coté client, non du coté serveur).

- Performance $\Longrightarrow$ coût de la généralité.
- Modèle structurant (modèle objet).
- Facilite le développement.
- Applications réparties? $\Longrightarrow$ état global, sémantiques/mécanismes.
- Avenir : OLE/DCOM, Java RMI , services CORBA, CORBA 3.0.

Partie 7

Annexes

Sous partie 7.1

Le mapping C++

Généralités (1)

Objectif d'un mapping : définition de l'API manipulée par le développeur lors de la construction d'applications CORBA *portables*.

Le mapping C++ est défini dans CORBA 2. Problèmes :

- Gestion mémoire de C++.
- Incompatibilités entre les différents compilateurs (RTTI[a], gestion des exceptions, *namespace*).

$\Longrightarrow$ le mapping offre sur certains points plusieurs alternatives $\Longrightarrow$ pb de portabilité.

[a] **RTTI** pour **R**un-**T**ime **T**ype **I**nformation.

Généralités (2)

Ce que définit le mapping C++ :

- Signatures des classes qui vont implanter les squelettes et les souches.
- Correspondance type IDL vers type C++.
- Correspondance pseudo-objets en C++ $\Longrightarrow$ une classe C++ par pseudo-objet décrivant les prototypes des opérations spécifiées en "pseudo" IDL.

La classe CORBA regroupe la définition des pseudo-objets et des types standards.

Mapping des types IDL (1)

- Types standards *double, long, octet*, etc. = *CORBA::Double*, *CORBA::Long*, etc.
- Types construits *sequence, string, union* = classe C++. Ces classes exportent des méthodes permettant de manipuler certaines caractéristiques des types IDL.
- Mapping pour les interfaces du coté client : héritage de *CORBA::Object*. Ajout de certaines fonctionnalités telles que *narrow*.

Mapping des types IDL (2)

Mapping pour les opérations :

- La gestion mémoire entre l'appelant et l'appelé.
- Règles de passage des paramètres ainsi que la signature des méthodes.

• Ces règles dépendent :

- Du type des paramètres manipulés (de taille fixe ou variable).
- Si les paramètres sont passés en *in*,*out* ou en *inout*.

Mapping pour les attributs : par deux méthodes, une seule si l'attribut est *readonly*.

La gestion mémoire

☞ Pb : le mapping introduit de nombreuses allocations dynamiques (pour l'utilisation de certains types IDL).

- Le mapping offre des outils permettant une gestion automatique de l'allocation et de la libération de la mémoire :
  - Par les types *T_ptr* et *T_var*.
  - Par l'ajout de mécanismes dans certaines classes implantant des types IDL (*sequence*, *union*, *string*, etc.). Lors d'affectation, de sortie de blocs.
- La gestion automatique peut parfois être désactivée (ex : *sequence*).

Les classes d'implémentation

☞ Du coté serveur, une interface IDL est implantée par une classe C++.

☞ CORBA **propose** deux méthodes pour connecter une classe d'implémentation à son squelette, tout en laissant à l'implanteur une totale liberté :

- Par héritage. Le squelette est une classe abstraite qui fournit une méthode virtuelle pure pour chaque opération IDL. La classe d'implémentation est fille du squelette.
- Par la délégation (ou *tie*). La classe *tie* joue le rôle de squelette et invoque directement la classe d'implémentation.

Invocation statique en C++

```
CORBA::ORB_ptr orb = CORBA::ORB_init();

CORBA::Object_var obj;
CORBA::Object_var sn;

sn = orb->resolve_initial_references("NameService");

NamingContext_var nom = NamingContext::_narrow(sn);
if (CORBA::is_nil(nom))
  fprintf(stderr,"Echec lors du narrow\n");

obj = nom->resolve("nom du service");

calculatrice_var calc = calculatrice::_narrow(obj);
if (CORBA::is_nil(calc))
  fprintf(stderr,"Echec lors du narrow\n");

calc->multiplication(2,3);
```


Invocation dynamique en C++

```
CORBA::ORB_ptr orb = CORBA::ORB_init();

CORBA::Object_var obj;
CORBA::Object_var sn;

sn = orb->resolve_initial_references("NameService");

NamingContext_var nom = NamingContext::_narrow(sn);
if (CORBA::is_nil(nom))
  fprintf(stderr,"Echec lors du narrow\n");

obj = nom->resolve("nom du service");

CORBA::Request_var request;
request = obj ->_request("multiplication");
request ->add_in_arg() <<= (CORBA::Long) 2 ;
request ->add_in_arg() <<= (CORBA::Long) 3 ;
request ->invoke();
```

Exemple d'un serveur CORBA

```
CORBA::ORB_ptr orb = CORBA::ORB_init();
CORBA::BOA_ptr boa = orb->BOA_init();

calculatrice_impl *calc = new calculatrice_impl;

boa->obj_is_ready(calc);

CORBA::Object_var sn;
sn = orb->resolve_initial_references("NameService");

NamingContext_var nom = NamingContext::_narrow(sn);
if (CORBA::is_nil(nom))
  fprintf(stderr,"Echec lors du narrow\n");

nom->bind(calc,"nom du service");

boa->impl_is_ready();
```


Sous partie 7.2

Le mapping Java

Java : Les types IDL

1. Types simples : correspondance directe en Java pour *double*, *char*, *boolean*, *float*, *short*. Autres : *octet* en *byte*, *string* en *String*, *long* en *int*.

2. Types complexes = classes Java (*typeHelper*).

Java : les exceptions

```
  exception a{};
  exception b{};

  interface c {
    void m() raises (a,b);
}
```

devient :

```
void m()
    throws a,b
  {
  ...
    throw new a();
  }
```

$\Longrightarrow$ exception utilisateur = classe héritant de java.Exception (exception contrôlée).

Objets, souches et squelettes

- Pseudo-objets : mappés en classes Java.
- Interface : une interface Java + une classe *interfaceHelper* et *interfaceHolder*.
- Souche : une classe Java avec héritage de org.omg.CORBA.Object.
- Squelette : une classe Java (liens d'héritage implementation ⟹ squelette ⟹ org.omg.CORBA.Object).

• Etape supplémentaire (class *portable*) pour les applets.

Passage de paramètres

- Paramètres passés en *in* :

```
void methode(in double t);
```

devient :

```
void methode(double t);
```

- Paramètres passés en *inout* ou *out* : Java = passage par valeur, sauf pour les objets ⟹ utilisation de classes *Holder*.

```
void methode(out double t);
```

devient :

```
void methode(doubleHolder t);
```

Invocation statique en Java

```
ORB orb = ORB.init();

org.omg.CORBA.Object obj;
org.omg.CORBA.Object sn;

sn = orb.resolve_initial_references("NameService");

NamingContext nom = NamingContextHelper.narrow(sn);
if (nom==null)
  System.err.println("Echec lors du narrow");

obj = nom.resolve("nom du service");

calculatrice calc = calculatriceHelper.narrow(obj);
if (calc==null)
  System.err.println("Echec lors du narrow");

res=calc.multiplication(2,3);
```


Invocation dynamique en Java

```
ORB orb = ORB.init();

org.omg.CORBA.Object obj;
org.omg.CORBA.Object sn;

sn = orb.resolve_initial_references("NameService");

NamingContext nom = NamingContextHelper.narrow(sn);
if (nom==null)
  System.err.println("Echec lors du narrow");

obj = nom.resolve("nom du service");

Request request=obj._request("multiplication");
request.add_in_arg().insert_long(2) ;
request.add_in_arg().insert_long(3) ;
request.invoke();
res=request.return_value().extract_long();
```

Exemple d'un serveur CORBA

```
ORB orb = ORB.init();
BOA boa = orb.BOA_init();

calculatrice_impl calc = new calculatrice_impl();

boa.obj_is_ready(calc);

org.omg.CORBA.Object sn;
sn = orb.resolve_initial_references("NameService");

NamingContext nom = NamingContextHelper.narrow(sn);
if (nom==null)
  System.err.println("Echec lors du narrow");

nom.bind(calc,"nom du service");

boa.impl_is_ready();
```


Sous partie 7.3

Travaux de standardisation actuels

Standardisation en cours (1)

Voir :

`http://www.omg.org/techprocess/meetings/schedule/index.html`

Dont les sujets suivants :

- Le passage d'objets par valeur[OMG 98].
- Interfaces multiples et composition d'interfaces[DST 97].
- Expression et respect de contraintes temps réel[OMG 97c] ainsi que la spécification d'un modèle CORBA minimal et modulaire[HP 97] pour les systèmes embarqués.
- CORBA et la sécurité[SYS 97].

Standardisation en cours (2)

- La standardisation de nouveaux services ou facilités communes (tel que des facilités pour l'impression[OMG 96]), ou l'évolution de certains (service de persistence[EXP 97]).
- Standardisation d'objets métiers dans de nombreuses branches d'activités + méthodologies : Télécommunications, santé/médical, transport, commerce électronique, finance, etc.
- ...

Sous partie 7.4

Quelques pointeurs sur CORBA

Quelques pointeurs sur CORBA

1. L'OMG :

   `http://www.omg.org`

2. Des mailings lists sur CORBA :

   `http://www.omg.org/news/index.html#mail`

3. Une liste d'ORB (dont certains sont freeware) :

   `http://cedric.cnam.fr/personne/duchien/CORBA.html`
   `http://www.infosys.tuwien.ac.at/`
   `Research/Corba/software.html`
   `http://adams.patriot.net/~tvalesky/freecorba.html`

4. Les spécifications CORBA :

   `http://www.enst.fr/~singhoff/DOC/CORBA/CORBA2`

5. Quelques travaux sur CORBA temps réel :

   `http://www.enst.fr/~singhoff/corba.html`

Partie 8

Références

[DST 97] DSTC, HP Company, IBM, ICL, IONA, et Prism Technologies. « Multiple Interfaces and Composition ». OMG TC Document orbos/97-02-06, February 1997.

[EXP 97] Expersoft, IONA, Oracle, Visigenic, et Al.. « Persistent State Service 2.0 ». OMG TC Document orbos/97-11-05, November 1997.

[HP 97] HP, Sun Microsystems, Alcatel, et Lucent. « Minimum CORBA ». OMG Document orbos/97-11-08, November 1997.

[OMG 93] OMG. « The Common Object Request Broker Architecture and Specification, Revision 1.2 ». TC Document 93-12-43, December 1993.

[OMG 96] OMG. « CORBA facilities RFP-6 : Printing Facility ». Document cf/96-04-03, April 1996.

[OMG 97a] OMG. « Common Facilities Architecture ». TC Document 97-06-15, November 1997.

[OMG 97b] OMG. « CORBAservices : Common Object Services Specification ». TC Document 97-12-02, November 1997.

[OMG 97c] OMG. « Real time CORBA 1.0 - Request For proposal ». Document orbos/97-09-31, 1997.

[OMG 98] OMG. « Object By Value ». OMG TC Document orbos/98-01-18, January 1998.

[OMG 99] OMG. « The Common Object Request Broker : Architecture and Specification. Revision 2.3 ». TC Document 99-10-07, October 1999.

[SYS 97] BEA Systems, Expersoft Corporations, IONA, Sun Microsystems, Visigenic Software IBM, et Fujitsu Limited. « CORBA/Firewall Security ». OMG Document orbos/97-10-07, 1997.