CORBA : méthode d'invocation dynamique

Frank Singhoff

Bureau C-202

Université de Brest, France

Lab-STICC/UMR 3192

singhoff@univ-brest.fr

Sommaire

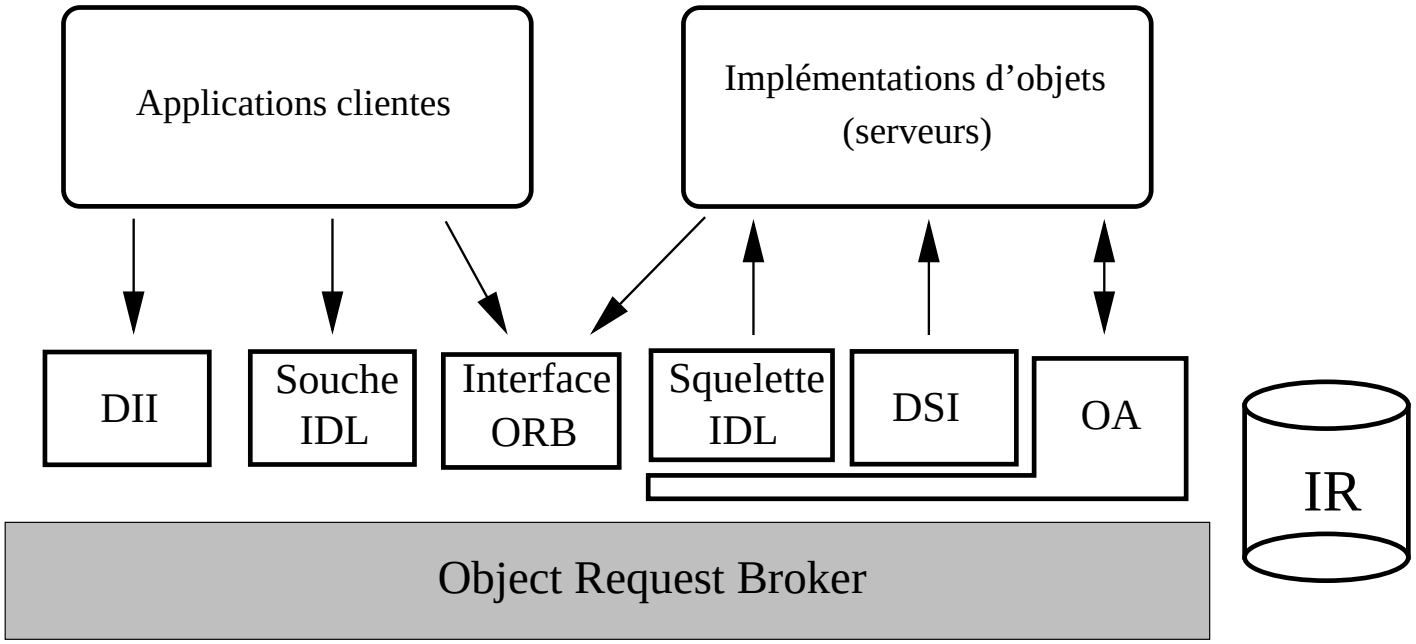
1. Rappels, invocation statique.
2. Principe de l’invocation dynamique.
3. DII : structures de données et primitives.
4. Exemples d’invocations avec la DII.
5. L’interface repository.
6. Exemple avec l’interface repository.
7. Ce qu’il faut retenir.

Rappels, invocation statique (1)

- **Fonctionnement:**
  - Le client invoque le serveur grâce à une souche ($stub$).
  - La souche est générée depuis l'IDL grâce à un "compilateur" IDL.
  - La souche offre les services d'encodage/décodage et de communication.

- **Propriétés:**
  - Le client est écrit **après** la rédaction de l'IDL.
  - Contrôle à la compilation (ex : typage).
  - Facilité d'emploi : transparence d'accès.
  - Efficace (a priori).

Rappels, invocation statique (2)



Rappels, invocation statique (3)

- 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 IDL.

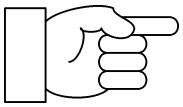
Sommaire

1. Rappels, invocation statique.
2. Principe de l’invocation dynamique.
3. DII : structures de données et primitives.
4. Exemples d’invocations avec la DII.
5. L’interface repository.
6. Exemple avec l’interface repository.
7. Ce qu’il faut retenir.

Invocation dynamique (1)

- **Principe :** construire les requêtes manuellement.
- **Avantages :**
  - Permet d'exploiter des interfaces inconnues lors de la compilation du client (ex : pont, débogueur, découverte dynamique de services dans systèmes de grande taille).
  - Sémantique d'invocation identique, mais mécanismes d'invocation plus riches (invocations asynchrones, parallèles).
  - Déployement d'applications facilité sur les postes clients (systèmes répartis de grande taille, gestion de version des souches).
- **Inconvénients :**
  - A priori plus coûteuse (gestion mémoire, contrôle en ligne du type des arguments) et plus dangereuse (nouvelles exceptions).
  - Perte de la transparence d'accès.
- Composants de l'ORB utilisés : l'IR et la DII.

Invocation dynamique (2)



Principe d'une invocation dynamique :

1. Obtenir une référence d'objet.
2. Construire une requête :
   - Allouer un pseudo-objet *Request*.
   - Insérer le nom de la méthode.
   - Décrire et insérer les paramètres in/inout, les exceptions, la valeur de retour.
3. Invoquer la méthode à l'aide de la DII.
4. Extraire les paramètres en out/inout, les exceptions et la valeur de retour.

Sommaire

1. Rappels, invocation statique.
2. Principe de l’invocation dynamique.
3. DII : structures de données et primitives.
4. Exemples d’invocations avec la DII.
5. L’interface repository.
6. Exemple avec l’interface repository.
7. Ce qu’il faut retenir.

Structure d'une requête (1)

- Structure d'une requête:

```
pseudo interface Request {
   attribute Object target;  // Référence d'objet
   attribute Identifier operation; // nom de méthode
   attribute NVList arguments;  // Liste d'argument
   attribute NamedValue result;  // Valeur de retour
   };
```

- Un argument est encodé par un $NamedValue$.
- Une liste d'arguments est encodée par un $NVList$.

```
typdef string Identifier;
struct NamedValue {
   Identifier name; // nom de l'argument
   any argument; // Valeur et type de l'argument
   long len; // taille
};
typedef sequence<NamedValue> NVList;
```

Structure d'une requête (2)

- **Le type IDL any encode un argument d'une méthode et est constitué de:**

  1. Un $TypeCode$ qui décrit le type de la donnée (ex: $long$).
  2. Une donnée encodée/sérialisée (ex : 10).

- Un type IDL any est manipulé par la classe Java Any. Pour un type IDL X:

  - Insertion/encodage dans un any, d'une donnée de type Y en Java correspondant au type IDL X:

    ```
    void insert_X(Y y);
    ```

  - Extraction/décodage depuis un any, d'une donnée de type Y en Java correspondant au type IDL X:

    ```
    Y extract_X();
    ```

Structure d'une requête (3)

- **La classe** Any **:**

```
package org.omg.CORBA;
public abstract class Any implements org.omg.CORBA.portable.IDLEntity {

    abstract public org.omg.CORBA.TypeCode type();
    abstract public void type(org.omg.CORBA.TypeCode type);

    abstract public int extract_long();
    abstract public void insert_long(int i);

    abstract public double extract_double();
    abstract public void insert_double(double d);

    abstract public boolean extract_boolean();
    abstract public void insert_boolean(boolean b);

    abstract public String extract_string();
    abstract public void insert_string(String s);
    ...
}
```

Structure d'une requête (4)

- **Exemple de manipulation d'un type** any **:**

```
import org.omg.CORBA.*;

public class AnyTest {
   public static void main(String args[]) throws IOException {
      ORB orb = ORB.init(args, null);

      // "any" instancie via la méthode "create_any" de l'ORB
      //
      Any any=orb.create_any();
      any.insert_string(new String("coucou"));
      System.out.println("String = " + any.extract_string());

      any.insert_long(100);
      System.out.println("long = " + any.extract_long());

      any.insert_double(100.50);
      System.out.println("double = " + any.extract_double());
      }
}
```

Structure d'une requête (5)

- $TypeCode$ **est un container qui décrit un type CORBA et est constitué de:**

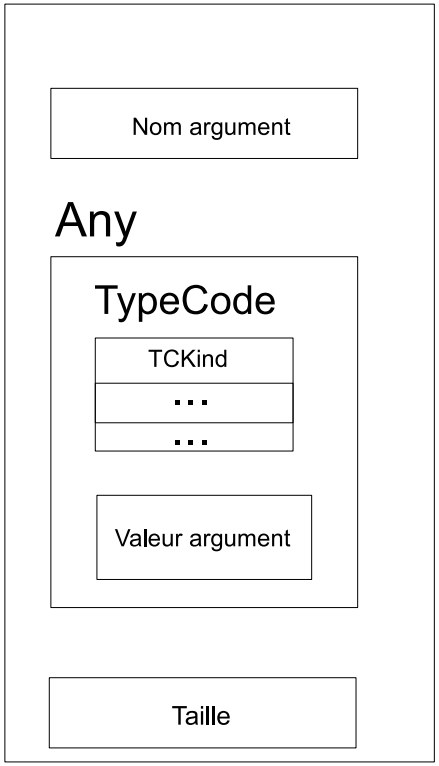
1. Un $TCKind$ décrivant la famille de type CORBA encodé.
2. 0 ou n champs décrivant le type encodé.

- Exemple IDL 1 : $long$ est encodé par $TCKind.tk\_long$ et 0 champs additionnel.
- Exemple IDL 2 : $typedef\ sequence < boolean, 10 > boolSeq$ est encodé par $TCKind.tk\_sequence$ et deux champs additionnels pour $boolean$ et 10.

Structure d'une requête (6)

- **Petit résumé :**

NamedValue (argument)



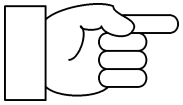

```
typdef string Identifier;

struct NamedValue {
   Identifier name;
   any argument;
   long len;
};

typedef sequence<NamedValue> NVList;

pseudo interface Request {
   attribute Object target;
   attribute Identifier operation;
   attribute NVList arguments;
   attribute NamedValue result;
   };
```

Construire la requête (1)



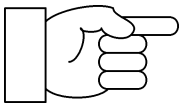
Principe d'une invocation dynamique :

1. Obtenir une référence d'objet.
2. Construire une requête :
   - Allouer un pseudo-objet *Request*.
   - Insérer le nom de la méthode.
   - Décrire et insérer les paramètres in/inout, les exceptions, la valeur de retour.
3. Invoquer la méthode à l'aide de la DII.
4. Extraire les paramètres en out/inout, les exceptions et la valeur de retour.

Construire la requête (2)

- **Les objets Request :**
  - Définissent la structure d'une requête.
  - Instanciés à partir d'une référence d'objet.
  - Offrent des primitives pour la manipulation des arguments et de la valeur de retour. Le nom de la méthode est initialisé lors de l'allocation d'un objet $Request$.
  - Offrent des primitives de communication : émission de la requête et réception de la réponse. Primitives de communication supplémentaires offertes par l'interface de l'ORB.

Construire la requête (3)



Insérer/extraire les arguments, valeur de retour :

- Insérer un argument en in :

```
any add_in_arg();
```

- Insérer un argument en $inout$:

```
any add_inout_arg();
```

- Insérer un argument en out :

```
any add_out_arg();
```

- Spécifier le type de la valeur de retour :

```
void set_return_type(in TypeCode tc);
```

- Extraction de la valeur de retour:

```
any return_value();
```

Construire la requête (4)

```
pseudo interface Object{
   InterfaceDef get_interface();
   boolean _is_nil();
   boolean _is_a(in string logical_type_id);
   boolean _non_existent();
   boolean _is_equivalent(in Object other_object);
   ...
   // Creation d'un Request
   Request _request(in Identifier operation);
   ...
};
```

Construire la requête (5)

```
pseudo interface Request {

   readonly attribute Object target;
   readonly attribute Identifier operation;
   readonly attribute NVList arguments;
   readonly attribute NamedValue result;

   any add_in_arg();
   any add_inout_arg();
   any add_out_arg();
   void set_return_type(in TypeCode tc);
   any return_value();

   void invoke();
   void send_oneway();
   void send_deferred();
   void get_response();
   boolean poll_response();
};
```

Construire la requête (6)

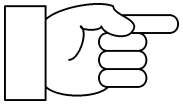
```
abstract public class Request {

    abstract public String operation();
    abstract public org.omg.CORBA.NVList arguments();
    abstract public org.omg.CORBA.NamedValue result();
    abstract public org.omg.CORBA.ExceptionList exceptions();

    abstract public org.omg.CORBA.Any add_in_arg();
    abstract public org.omg.CORBA.Any add_inout_arg();
    abstract public org.omg.CORBA.Any add_out_arg();
    abstract public void set_return_type(org.omg.CORBA.TypeCode tc);
    abstract public org.omg.CORBA.Any return_value();

    abstract public void invoke();
    abstract public void send_oneway();
    abstract public void send_deferred();
    abstract public void get_response() throws
                org.omg.CORBA.WrongTransaction;
    abstract public boolean poll_response();
    ...
}
```

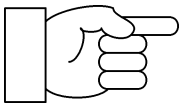
Mécanismes d'invocations (1)



Principe d'une invocation dynamique :

1. Obtenir une référence d'objet.
2. Découvrir le service offert par l'objet, via l'IR par exemple.
3. Construire une requête :
   - Allouer un pseudo-objet *Request*.
   - Insérer le nom de la méthode.
   - Décrire et insérer les paramètres in/inout, les exceptions, la valeur de retour.
4. Invoquer la méthode à l'aide de la DII.
5. Extraire les paramètres en out/inout, les exceptions et la valeur de retour.

Mécanismes d'invocations (2)



Mécanismes d'invocation : invocation synchrone, synchrone différée (c-à-d asynchrone).

1. **Emission de la requête :**
   - Emission synchrone :

     ```
     void invoke();
     ```

   - Emission asynchrone avec sémantique au plus une fois (sans réponse) :

     ```
     void send_oneway();
     ```

   - Emission asynchrone avec sémantique exactement une fois :

     ```
     void send_deferred();
     ```

2. **Réception de la réponse :**
   - Réception bloquante :

     ```
     void get_response();
     ```

   - Scrutation active (*polling*) :

     ```
     boolean poll_response();
     ```

Mécanismes d'invocations (3)

```
pseudo interface Request {

   readonly attribute Object target;
   readonly attribute Identifier operation;
   readonly attribute NVList arguments;
   readonly attribute NamedValue result;

   any add_in_arg();
   any add_inout_arg();
   any add_out_arg();
   void set_return_type(in TypeCode tc);
   any return_value();

   void invoke();
   void send_oneway();
   void send_deferred();
   void get_response();
   boolean poll_response();
};
```

Mécanismes d'invocations (4)

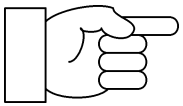
```
abstract public class Request {

    abstract public String operation();
    abstract public org.omg.CORBA.NVList arguments();
    abstract public org.omg.CORBA.NamedValue result();
    abstract public org.omg.CORBA.ExceptionList exceptions();

    abstract public org.omg.CORBA.Any add_in_arg();
    abstract public org.omg.CORBA.Any add_inout_arg();
    abstract public org.omg.CORBA.Any add_out_arg();
    abstract public void set_return_type(org.omg.CORBA.TypeCode tc);
    abstract public org.omg.CORBA.Any return_value();

    abstract public void invoke();
    abstract public void send_oneway();
    abstract public void send_deferred();
    abstract public void get_response() throws
                org.omg.CORBA.WrongTransaction;
    abstract public boolean poll_response();
    ...
}
```

Mécanismes d'invocations (5)



Mécanismes d'invocation multiple : émission de plusieurs requêtes simultanément puis réception des réponses au fur et à mesure de leur arrivée.

1. **Emission des requêtes :**
   - Sémantique au plus une fois (pas de réponse) :

     ```
     void send_multiple_requests_oneway(in RequestSeq req);
     ```

   - Sémantique exactement une fois :

     ```
     void send_multiple_requests_deferred(in RequestSeq req);
     ```

2. **Réception des réponses :**
   - Réception bloquante :

     ```
     void get_next_response(out Request req);
     ```

   - Polling :

     ```
     boolean poll_next_response();
     ```

Mécanismes d'invocations (6)

```
pseudo interface ORB {
  string object_to_string(in Object obj);
  Object string_to_object(in string str);
  Object resolve_initial_references(in ObjectId identifier)
      raises (InvalidName);
  void run();
  ...

  // Requetes multiples
  typedef sequence<Request> RequestSeq;

  void send_multiple_requests_oneway(in RequestSeq req);
  void send_multiple_requests_deferred(in RequestSeq req);
  boolean poll_next_response();
  void get_next_response(out Request req);
  ...
};
```

Sommaire

1. Rappels, invocation statique.
2. Principe de l’invocation dynamique.
3. DII : structures de données et primitives.
4. Exemples d’invocations avec la DII.
5. L’interface repository.
6. Exemple avec l’interface repository.
7. Ce qu’il faut retenir.

Invocations avec la DII (1)

- Exemple d'une invocation synchrone :

```
interface compte {
 ...
 long debiter(in long montant);
};
```

```
org.omg.CORBA.Object obj = nom.resolve("nom du service");

Request a_request=obj._request("debiter");
a_request.add_in_arg().insert_long(1000);
a_request.set_return_type(orb.get_primitive_tc(
                org.omg.CORBA.TCKind.tk_long));

// Invocation synchrone (bloquante)
a_request.invoke();
int solde=a_request.return_value().extract_long();
```

Invocations avec la DII (2)

- Exemple d'une invocation asynchrone bloquante:

```
org.omg.CORBA.Object obj = nom.resolve("nom du service");

Request a_request=obj._request("debiter");
a_request.add_in_arg().insert_long(1000);
a_request.set_return_type(orb.get_primitive_tc(
                org.omg.CORBA.TCKind.tk_long));

// Emission requete asynchrone/différée
a_request.send_deferred();

// Faire autre chose
...

// Reception synchrone (bloquante)
a_request.get_response();
int solde=a_request.return_value().extract_long();
```

Invocations avec la DII (3)

- Exemple d'une invocation asynchrone non bloquante :

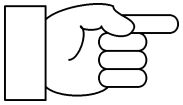
```
org.omg.CORBA.Object obj = nom.resolve("nom du service");

Request a_request=obj._request("debiter");
a_request.add_in_arg().insert_long(1000);
a_request.set_return_type(orb.get_primitive_tc(
                org.omg.CORBA.TCKind.tk_long));

// Emission requete asynchrone/différée
a_request.send_deferred();

// Reception asynchrone
while (!a_request.poll_response())
    {
    // Faire autre chose
    ...
    }
int solde=a_request.return_value().extract_long();
```

Invocations avec la DII (4)



Exceptions lors des invocations dynamiques :

- $BAD\_OPERATION$: méthode inconnue.
- BAD_PARAM : erreur sur les arguments de la méthode.
- $BAD\_TYPECODE$: $TypeCode$ incorrect (description du type d'un argument).
- $MARSHAL$ et $DATA\_CONVERSION$: erreur d'encodage/décodage d'une donnée.
- BAD_INV_ORDER : ordre d'invocation des méthodes faux.
- $NO\_RESPONSE$: message de réponse non disponible (non expédié par le serveur).

Sommaire

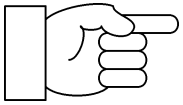
1. Rappels, invocation statique.
2. Principe de l'invocation dynamique.
3. DII : structures de données et primitives.
4. Exemples d'invocations avec la DII.
5. L'interface repository.
6. Exemple avec l'interface repository.
7. Ce qu'il faut retenir.

Interface repository (1)

☞ Pour utiliser la DII, il faut connaître l'interface de l'objet à invoquer.

☞ Comment découvrir l'interface d'un objet CORBA ?

Interface repository (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).

Interface repository (3)

- **Types de données:**

```
// Attention : IR simplifiée vis-à-vis du standard CORBA
//
module Ir {

    typedef string Identifier;

    enum TCKind {
        tk_null,     tk_void,
        tk_short,    tk_long,         tk_ushort,  tk_ulong,
        tk_float,    tk_double,       tk_boolean, tk_char,
        tk_octet,    tk_any,          tk_TypeCode,tk_Principal,
        tk_struct,   tk_union,        tk_enum,    tk_string,
        tk_sequence,tk_array,         tk_alias,   tk_except,
        tk_longlong,tk_ulonglong,     tk_longdouble,
        tk_wchar,    tk_wstring,      tk_fixed,
        tk_value,    tk_value_box,
        tk_native,
        tk_abstract_interface
    };
```

Interface repository (4)

- **Chaque entité est représentée (ex: attribut):**

```
// readonly ou read-write ?
enum AttributeMode {ATTR_NORMAL, ATTR_READONLY};

interface AttributeDescription {
    // Nom de la variable
    readonly attribute Identifier    name;
    // Type de donnee
    attribute TCKind                 type;
    // Mode d'acces
    attribute AttributeMode          mode;
};
```

Interface repository (5)

- **Chaque entité est représentée (ex: opération):**

```
// Semantique exactement ou au plus une fois
enum OperationMode {OP_NORMAL, OP_ONEWAY};
// Parametre in/out/inout
enum ParameterMode {PARAM_IN, PARAM_OUT, PARAM_INOUT};

struct ParameterDescription {
    Identifier                      name; // nom argument
    TCKind                          type; // type argument
    ParameterMode                   mode; // mode passage
};
typedef sequence <ParameterDescription>  ParDescriptionSeq;

interface OperationDescription {
    readonly attribute Identifier  name;  // nom du sous-programme
    attribute TCKind               result; // valeur de retour
    attribute ParDescriptionSeq    params; // liste arguments
    attribute OperationMode        mode;  // semantique
    ...
};
```

Interface repository (6)

- **Services de mise à jour et de consultation:**

```
typedef sequence <OperationDescription> OpDescriptionSeq;
typedef sequence <AttributeDescription> AttrDescriptionSeq;

interface InterfaceDescription  {

  readonly attribute Identifier   name; // nom interface
  attribute OpDescriptionSeq      operations; // liste methodes
  attribute AttrDescriptionSeq    attributes; // liste attributs

  AttributeDescription create_attribute (
                              in Identifier              name,
                              in TCKind                  type,
                              in AttributeMode           mode);
  OperationDescription create_operation (
                              in Identifier              name,
                              in TCKind                  result,
                              in OperationMode           mode,
                              in ParDescriptionSeq       params);
};
```

Sommaire

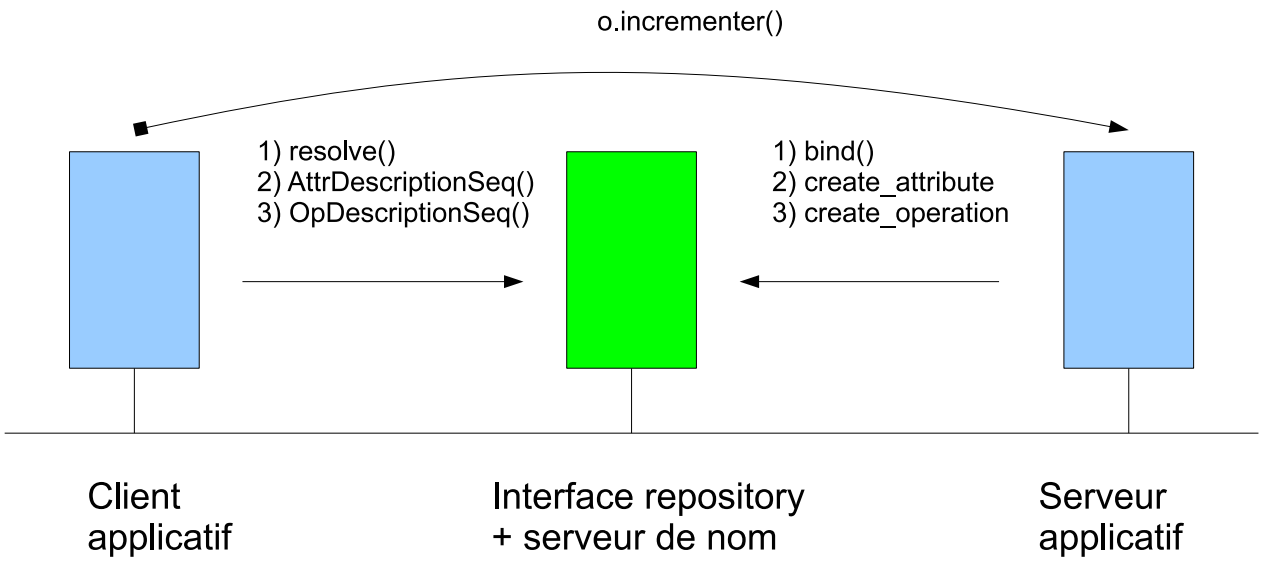
1. Rappels, invocation statique.
2. Principe de l’invocation dynamique.
3. DII : structures de données et primitives.
4. Exemples d’invocations avec la DII.
5. L’interface repository.
6. Exemple avec l’interface repository.
7. Ce qu’il faut retenir.

Exemple d'interface repository (1)

- **Exemple d'une interface IDL:**

```
interface variable {
        attribute double valeur;
        attribute string nom;
        void incrementer(in double donnee);
        void decrementer(in double donnee);
 };
```

- **Un serveur couplant service de nom + IR:**



Exemple d'interface repository (2)

- **Un serveur couplant service de nom + IR:**

```
interface IRepository  {
      void resolve(in Identifier object_name,
                   inout InterfaceDescription interface_desc,
                   inout Object ref_obj);

      InterfaceDescription bind(in Object ref_obj,
                                in Identifier object_name,
                                in Identifier interface_name);
 };
```

- $bind$: associe un objet CORBA ref_obj a un nom symbolique $object\_name$ et une interface IDL $interface\_name$. Nécessite la description de l'interface.
- $resolve$: retourne pour le nom symbolique $object\_name$ sa référence d'objet ref_obj et son interface IDL $interface\_desc$.

Exemple d’interface repository (3)

- **Exemple d’une publication/liaison:**

```
// Création de l’objet  applicatif de type "variable"
variableImpl varImpl = new variableImpl("une variable") ;
org.omg.CORBA.Object varRef = poa.servant_to_reference(varImpl);
```

```
// Récupération de l’IOR sur l’IR
String ior_IR = "IOR:01890A9902F232DDDC...";
org.omg.CORBA.Object obj = orb.string_to_object(ior_IR);
IRepository IR = IRepositoryHelper.narrow(obj);
```

```
// Déclaration de l’objet "varImpl"  auprès de l’IR
// et récupération de la référence d’objet sur son interface
InterfaceDescription interfaceRef = IR.bind(
     varRef , "une variable" , "variable" ) ;
```

Exemple d'interface repository (4)

- **Exemple d'une publication/liaison (suite):**

```
// Déclarer les attributs de l'interface
AttributeDescription valeur_desc = interfaceRef.create_attribute(
     "valeur", TCKind.tk_double, AttributeMode.ATTR_NORMAL);
AttributeDescription nom_desc = interfaceRef.create_attribute(
     "nom", TCKind.tk_string, AttributeMode.ATTR_NORMAL);

// Déclarer les méthodes : un seul argument par méthode
ParameterDescription[] liste_params = new ParameterDescription[1];
liste_params[0] = new ParameterDescription(
     "donnee", TCKind.tk_double, ParameterMode.PARAM_IN);

OperationDescription variable_plus_double_desc =
   interfaceRef.create_operation("incrementer", TCKind.tk_void,
        OperationMode.OP_NORMAL, liste_params);
OperationDescription variable_plus_double_desc =
    interfaceRef.create_operation("decrementer", TCKind.tk_void,
        OperationMode.OP_NORMAL, liste_params);
```

Exemple d’interface repository (5)

- **Exemple d’une résolution/invocation:**

```
// Récupération de l’IOR sur l’IR
String ior_IR = "IOR:01890A9902F232DDDC...";
org.omg.CORBA.Object obj = orb.string_to_object(ior_IR);
IRepository IR = IRepositoryHelper.narrow(obj);

// Récupération de la référence d’objet ainsi que sa description
InterfaceDescriptionHolder interfaceH = new InterfaceDescriptionHolder();
org.omg.CORBA.ObjectHolder refH = new org.omg.CORBA.ObjectHolder();
IR.resolve( "ma variable", interfaceH, refH);

// Découverte des méthodes et attributs
// de l’interface grâce à l’IR
AttributeDescription[] liste_attr = interfaceH.value.attributes();
OperationDescription[] liste_op = interfaceH.value.operations();

// Invocation de l’objet sur le serveur applicatif
org.omg.CORBA.Request requete = refH.value._request(...);
```

Sommaire

1. Rappels, invocation statique.
2. Principe de l’invocation dynamique.
3. DII : structures de données et primitives.
4. Exemples d’invocations avec la DII.
5. L’interface repository.
6. Exemple avec l’interface repository.
7. Ce qu’il faut retenir.

Ce qu'il faut retenir

- Pourquoi utiliser la DII.
- Souches ou DII = sémantique identique mais mécanismes de communication différents.
- Mécanismes d'invocation asynchrones et concurrents.
- Structure d'une requête : $Request$, any, $TypeCode$.
- La notion de référentiel d'interface (interface repository).