Interopérabilité : Corba
Constructions d'application réparties

Hazaël JONES

Master 1 - 2007/2008

Plan de l'exposé

Les adaptateurs d'objets

Quelques questions

- Pourquoi un adaptateur d'objet ?
- Comment obtenir toujours les mêmes références d'objet ?
- Comment avoir des objets persistants ?
- Que faire quand plusieurs clients accèdent au même objet ?

Les politiques du POA

Politiques

- A chaque POA est associé un ensemble de politiques qui définissent entre autres :
  - le mode d'attribution des identificateurs des objets au sein du POA ;
  - la persistance des références ;
  - le mode d'activation des servants ;
  - la mode de localisation des servants ;
  - le modèle de concurrence.

Les politiques du POA

Politiques

- Les politiques d'un POA sont définies une fois pour toute au moment de sa création.
  - Les politiques du POA racine ne peuvent pas être modifiées
- Besoin de créer des nouveaux POA pour choisir ses politiques
  - POA organisés de façon hiérarchique à partir du RootPOA
  - POA créés par : poa.create_POA(nom, manager, politiques) ;
- POA associé au moment de sa création à un manager qui gère son état :
  - récupéré depuis un autre POA : poa.the_POAManager()
  - créé automatiquement en passant null

Création d'un nouveau POA

Code de création d'un nouveau POA

```
1  Object o =
       orb.resolve_initial_references("RootPOA");
2  POA rootPOA = POAHelper.narrow(o);
3  POAManager manager = rootPOA.the_POAManager();
4  Policy[] policies = new Policy[] { ... };
5  POA myPOA = rootPOA.create_POA("child",
       manager , policies);
```

Les différentes politiques

Politique de gestion de la concurrence : ThreadPolicy

- Par défaut et RootPOA : ORB_CTRL_MODEL
- Autres modes : SINGLE_THREAD_MODE

- Appels concurrents sur un même objet peuvent être confiés a plusieurs processus légers...
- Possibilité de forcer le traitement itératif

```
1 ThreadPolicyValue v =
2   ThreadPolicyValue.SINGLE_THREAD_MODEL;
3 Policy p = rootPOA.create_thread_policy(v);
```

Les différentes politiques

Politique d'activation implicite : ImplicitActivationPolicy

- Par défaut : NO_IMPLICIT_ACTIVATION
- RootPOA : IMPLICIT_ACTIVATION

- Possibilité d'avoir une activation implicite des servants (pas d'appel activate_object() ou activate_object_with_id())
- Création de la politique à affecter au nouveau POA (politique par défaut du RootPOA) :

```
1 ImplicitActivationPolicyValue v =
2   ImplicitActivationPolicyValue.IMPLICIT_ACTIVATION;
3 Policy iap = rootPOA.create_implicit_activation_policy(v);
```

- Activation automatique par les méthodes poa.servant_to_id() et poa.servant_to_reference()

Les différentes politiques

Politique de gestion des identificateurs : LifespanPolicy

- Par défaut et RootPOA : TRANSIENT
- Autres modes : PERSISTENT

- Faire en sorte que l'IOR d'un objet soit le même d'une exécution a l'autre de l'ORB
- Fixer l'identificateur (entier) et le port du serveur avec des propriétés spécifiques a l'ORB de Sun (com.sun.CORBA.POA.ORBServerId et com.sun.CORBA.POA.ORBPersistentServerPort)
- Forcer un choix reproductible des identificateurs d'objet :

```
1    LifespanPolicyValue v =
2      LifespanPolicyValue.PERSISTENT;
3    Policy p = rootPOA.create_lifespan_policy(v);
```

Les différentes politiques

Politique de gestion des identificateurs

- Fixer soi-même les identificateurs des objets
- Permet de stocker des informations dans la référence de l'objet
- Les étapes :
  - Création des identificateurs :
    byte[] id = "Mon identificateur".getBytes();
  - Affectation de l'identificateur au moment de l'activation :
    childPOA.activate_object_with_id(id, servant);

Les différentes politiques

Politique de gestion des identificateurs : IdAssignmentPolicy

- Par défaut et RootPOA : SYSTEM_ID
- Autres modes : USER_ID

- Forcer la définition des identificateur par l'utilisateur
- Création de la politique à affecter au nouveau POA :

```
1  Policy p = rootPOA.
2    create_id_assignment_policy(IdAssignmentPolicyValue.USER_ID);
```

- Interdit l'utilisation de activate_object()
- Incompatible avec la politique IMPLICIT_ACTIVATION

Les différentes politiques

Politique de gestion des identificateurs : IdUniquenessPolicy

- Par défaut et RootPOA : UNIQUE_ID
- Autres modes : MULTIPLE_ID

- Fixer plusieurs identificateurs à un même objet
- Pour différencier les clients accédant à un même objet
- Les étapes :
  - Création de la politique à affecter au nouveau POA :
    IdUniquenessPolicyValue v = IdUniquenessPolicyValue.MULTIPLE_ID ;
    Policy p = rootPOA.create_id_uniqueness_policy(v) ;
  - Activation multiple d'un objet avec activate_objet() ou activate_object_with_id()

Les différentes politiques

Politique de gestion des identificateurs

- Comment identifier le client coté serveur ?
- Au cours de l'appel de méthode coté serveur, le contexte Current permet de récupérer le POA (get_POA()) et l'identificateur (get_object_id()) stockés dans la référence du client

```
1  private Current getCurrent() throws InvalidName {
2     Object current = _orb().
3       resolve_initial_references("POACurrent");
4     return CurrentHelper.narrow(current); }
```

- Par défaut, dans un POA, les associations identifiant/servant sont stockées dans une table des objets actifs (Active Object Map)
- Par défaut, cette table est utilisée pour retrouver, au moment d'un appel, un servant à partir de son identifiant

Les différentes politiques

Créer des références sans activation

- Il est possible de créer des références sans activation, c'est-à-dire, sans les associer a un servant
- Pour cela il faut créer des références a partir du type IDL comme suit :

```
1    org.omg.CORBA.Object ref =
2      poa.create_reference_with_id(id,"IDL:type:1.0");
```

ou

```
1    org.omg.CORBA.Object ref =
2      poa.create_reference("IDL:type:1.0");
```

Les différentes politiques

Politique de sauvegarde des associations identifiant/servant : ServantRetentionPolicy

- Par défaut et RootPOA : RETAIN
- Autres modes : NON_RETAIN

- Il est possible de demander que les identifiants des objets activés ne soient pas sauvegardés dans l'OAM
- Nécessite un autre mécanisme pour trouver le servant
- Permet une gestion dynamique des objets servants
- Création de la politique à affecter au nouveau POA :

```
1   ServantRetentionPolicyValue v =
2     ServantRetentionPolicyValue.NON_RETAIN;
3  Policy srp =
       rootPOA.create_servant_retention_policy(v);
```

- Interdit l'utilisation de toutes les fonctions du type id_to_reference() qui reposent sur l'OAM

Les différentes politiques

Comment trouver le servant ? (RequestProcessingPolicy)

- Par défaut et RootPOA : USE_ACTIVE_OBJECT_MAP_ONLY
- Autres modes : USE_DEFAULT_SERVANT et USE_SERVANT_MANAGER

- Si l'identificateur n'est pas trouvé dans l'OAM, il est possible d'utiliser :
  - Un serveur par défaut :
    ```
    RequestProcessingPolicyValue v =
    RequestProcessingPolicyValue.USE_DEFAULT_SERVANT ;
    Policy p = rootPOA.create_request_processing_policy(v) ;
    ```
  - Un gestionnaire de servants :
    ```
    RequestProcessingPolicyValue v =
    RequestProcessingPolicyValue.USE_SERVANT_MANAGER ;
    ```

Les différentes politiques

Utiliser un servant par défaut

- Affecter la politique USE_DEFAULT_SERVANT au POA
- Implanter une classe servant qui gère le comportement par défaut
- Enregistrer le servant par défaut dans le POA : poa.set_servant(defaultServant) ;
  - poa.get_servant() ; : opération inverse

Les différentes politiques

Utiliser un gestionnaire de servants

- Affecter la politique USE_SERVANT_MANAGER au POA
- Implanter une classe implantant l'interface ServantActivator ou ServantLocator (toutes deux héritant de ServantManager) et héritant de LocalObject
- Installer une instance de cette classe dans le POA : poa.set_servant_manager(activator) ;
  - poa.get_servant_manager() : opération inverse

Objets temporaires

Le localisateur de servants

- Appelé lors de chaque accès à une référence
- Nécessite la politique NON_RETAIN
- Méthode preinvoke() retourne le servant associé à la référence recherchée
  - peut lever une exception ForwardRequest pour indiquer une nouvelle localisation d'objet
  - peut lever une exception OBJECT_NOT_EXIST en cas de création impossible
- Méthode postinvoke() appelée une fois l'appel de méthode traitée

Objets temporaires

Le localisateur de servants

```
 1  package fr.umlv.ir3.corba.cours4;
 2  import org.omg.CORBA.LocalObject;
 3  import org.omg.PortableServer.*;
 4  import org.omg.PortableServer.ServantLocatorPackage.CookieHolder;
 5  public class HelloServantLocator extends LocalObject
 6                                   implements ServantLocator {
 7    public Servant preinvoke(byte[] oid, POA adapter, String operation,
 8                             CookieHolder the_cookie) throws ForwardRequest {
 9      HelloImpl servantObj = new HelloImpl();
10      return servantObj;
11    }
12    public void postinvoke(byte[] oid, POA adapter, String operation,
13                           Object the_cookie, Servant the_servant) {
14    }
15  }
```

Activation d'Objets

Activateur de servants

- Gère l'activation et la désactivation dynamique des servants en coopération avec l'OAM
- Nécessite la politique RETAIN
- Méthode incarnate() appelée si l'identificateur n'est pas trouvé dans l'AOM
  - peut lever une exception ForwardRequest pour indiquer une nouvelle localisation d'objet
  - peut lever une exception OBJECT_NOT_EXIST en cas de création impossible
- Méthode etheralize() appelée lorsque le POA est désactivé via poa.destroy(true,waitForCompletion)

Activation d'Objets

Activateur de servants

```
 1  package fr.umlv.ir3.corba.cours4;
 2  import org.omg.CORBA.*;
 3  import org.omg.CORBA.Object;
 4  import org.omg.PortableServer.*;
 5  public class HelloServantActivator extends LocalObject
 6                implements ServantActivator {
 7    private Object otherRef;
 8    HelloServantActivator(Object otherRef) { this.otherRef = otherRef; }
 9    public Servant incarnate(byte[] oid, POA adapter)
10                                    throws ForwardRequest {
11      String str = new String(oid);
12      if (str.equals("MyOID")) {
13        return new HelloImpl();
14      } else {
15        if (str.equals("OtherID")) {
16          throw new ForwardRequest(otherRef);
17        } else {
18          throw new OBJECT_NOT_EXIST();
19        }
20      }
21    }
22  }
23
24  public void etherealize(byte[] oid, POA adapter, Servant serv,
25      boolean cleanup_in_progress, boolean remaining_activations) {
26    if (remaining_activations == false) {
27      serv.save();
28    }
29  }
```

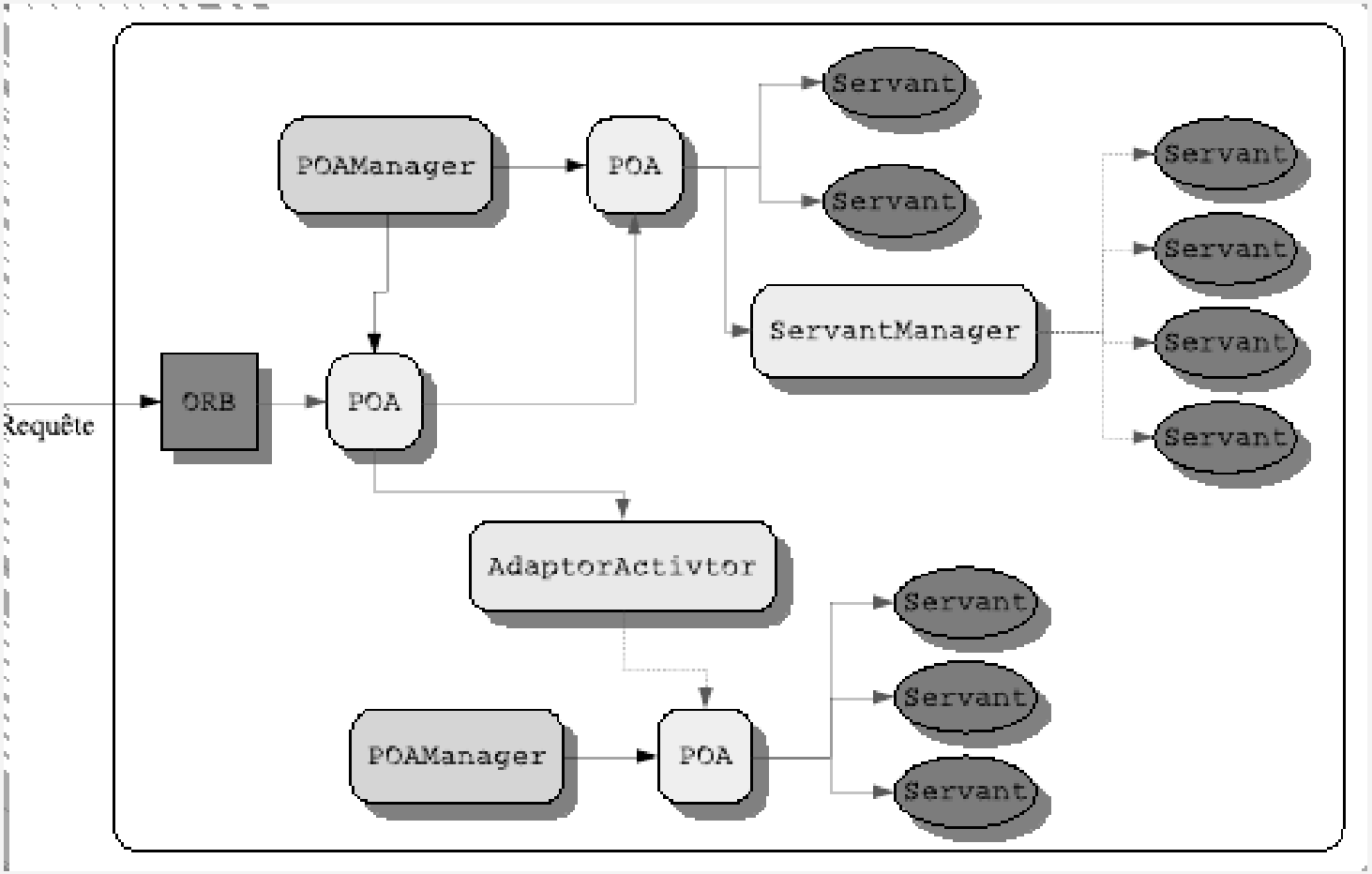
Activation d'Objets

Gérer la persistance

- Il faut être capable de reconstruire le serveur a partir de "rien"
- Permettre la réactivation et la sauvegarde des servants avec les activateurs de servants
- Permettre la réactivation des POA en implantant AdapterActivator et installant une instance dans le POA parent avec la méthode parentPOA.the_activator(activator) Permettre la réactivation du serveur avec la commande servertool qui interagit avec orbd

Activation d'Objets

Gérer la persistance



Activation d'Objets

Activateur de servants

```
1  package fr.umlv.ir3.corba.cours4;
2  import org.omg.CORBA.*;
3  import org.omg.PortableServer.*;
4  import org.omg.PortableServer.POAPackage.*;
5  public class HelloAdapterActivator extends LocalObject
6                              implements AdapterActivator {
7    public boolean unknown_adapter(POA parent, String name) {
8      try {
9        Policy[] p = new Policy[2];
10       RequestProcessingPolicyValue rppv =
11         RequestProcessingPolicyValue.USE_SERVANT_MANAGER;
12       LifespanPolicyValue lp =
13         LifespanPolicyValue.PERSISTENT;
14       p[0] = parent.create_request_processing_policy(rppv);
15       p[1] = parent.create_lifespan_policy(lp);
16       POA poa = parent.create_POA(name,parent.the_POAManager(),p);
17       poa.the_activator(parent.the_activator());
18       poa.set_servant_manager(new HelloServantActivator());
19       return true;
20     } catch (AdapterAlreadyExists e) {
21     } catch (InvalidPolicy e) {
22     } catch (WrongPolicy e) {
23     }
24     return false;
25   }
26 }
```

La commande servertool

Gérer la persistance

- Gère les serveurs activés par orbd

```
1       # servertool -ORBInitialPort 1234
2       servertool > register -classpath . -server Server
3               serveur inscrit (ID serveur = 257).
4       servertool > listactive
5               ID serveur      Nom de classe serveur
6               ----------      --------------------
7                  257          Server
8       servertool > shutdown -serverid 257
9               Serveur arrete
10      servertool > quit
```

- Méthode main() du serveur ne doit pas lever d'exception
- L'option classpath est relative au répertoire de démarrage de orbd

Plan de l'exposé

Le type Any

- Transmettre des arguments ou des valeurs de retour génériques
- Type IDL any correspondant au type Java Any
- Instance vide construite via orb.create_any()
- Instance remplie avec une valeur particulière par :
  - orb.insert_type() pour les types primitifs
  - Type Helper.insert() pour les types utilisateurs
  - Classes particulières pour insérer des tableaux ou séquences (i.e. LongSeqHelper)

Récupération du type du contenu

- Il faut connaître le type de l'objet pour l'extraire
- Type représenté par un objet TypeCode retourné par la méthode type()
- Contient des informations caractérisant le type de l'objet :
  - la sorte (type primitif, structure, séquence, alias, . . .) retournée par la méthode kind(), représentée par l'énumération IDL TCKind (entier retourné par value())
  - pour les types utilisateurs, l'identificateur de type(que l'on trouve aussi dans les IOR) retourné par la méthode id()
  - pour les types conteneurs simples (alias, séquence ou tableau), le type du ou des éléments contenus retourné par content_type()
  - pour les types conteneurs complexes (structures, unions, . . .), le nombre de membres (member count()) et, pour chacun d'eux, leur nom (member_count(i)) et leur type (member_type(i))

Récupération du type du contenu

- Comparer la sorte (kind().value()) en utilisant les constantes entières TCKind._tk_kind
- Comparer directement l'objet TypeCode avec les méthodes equal() ou equivalent() (pour les alias) avec les objets :
  - Type Helper.type() pour les types utilisateurs
  - orb.get_primitie_tc(TCKind.tk_type) pour les types primitifs

Récupérer le contenu d'un Any

- any.extract_type() pour les types primitifs
- Type Helper.extract(any) pour les autres types

Exemple : Récupération du type réel

```
1  TypeCode getRealType(TypeCode type) {
2     try {
3        if (type.kind().value() == TCKind._tk_alias) {
4           return getRealType(type.content_type());
5        }
6        return type;
7     } catch (BadKind e) {
8        assert false;
9        throw new RuntimeException(e);
10    }
11  }
```

Affichage des informations sur un type

```
1    TypeCode type = ...
2    int n = type.member_count();
3    System.err.println("Nombre de membres : " + n);
4    for (int i = 0; i < n; i++) {
5      System.err.print("Membre n " + i + " : ");
6      System.err.print(type.member_name(i));
7      System.err.print(" de type ");
8      System.err.println(type.member_type(i)
9    );
10   }
```

Construction de l'objet Any


```
1  Any arg = orb.create_any();
2  arg.insert_string(``Marne-la-Vallee'');
3  System.out.println(h.hello(arg));
4  LocalityHelper.insert(arg,new Locality(``Paris''));
5  System.out.println(h.hello(arg));
6  String [] t = new String[] ``Paris'', ``Londres'',
       ``Tokyo''
7  StringSeqHelper.insert(arg,t);
8  System.out.println(h.hello(arg));
9  arg.insert_long(2);
10 System.out.println(h.hello(arg));
```

Construction de l'objet Any


```
1   public String hello(Any arg) {
2     try {
3       String h = "Hello from ";
4       TypeCode type = getRealType(arg.type());
5       switch (type.kind().value()) {
6       case TCKind._tk_string:
7         return h + arg.extract_string();
8       case TCKind._tk_struct:
9         if (type.equal(LocalityHelper.type())) {
10          return h + LocalityHelper.extract(arg).name;
11        }
12        break;
13      case TCKind._tk_sequence:
14        TypeCode stringType = ORB.init().get_primitive_tc(TCKind.tk_string);
15        if (type.content_type().equivalent(stringType)) {
16          for (String from : StringSeqHelper.extract(arg)) {
17            h += from + " ";
18          }
19          return h;
20        }
21        break;
22      }
23      return h + "nowhere";
24    } catch (BadKind e) {
25      assert false;
26      throw new RuntimeException(e);
27    } catch (Bounds e) {
28      assert false;
29      throw new RuntimeException(e);
30    }
```

Plan de l'exposé

Invocation dynamique

- Possibilité d'invoquer une méthode sur un objet CORBA sans disposer de son interface ni de sa classe Helper
- Utilisé dans les souches (stub)
- Nécessite de connaître :
  - les profils des méthodes
  - les classes des types utilisateurs utilisées pour les paramètres, la valeur de retour et les exceptions

Création de l'objet requête

- Obtenir une référence de l'objet CORBA sur lequel invoquer une méthode
- Créer un objet requête (Request) associé à une réference avec la méthode _request() de celle-ci ref._request("hello")
- Préciser les arguments :
  - Créer, dans l'ordre des paramètres, des objets Any pour les contenir soit avec add_in_arg(), add_out_arg() ou add_inout_arg(), soit avec add_named_in_arg(), add_named_out_arg() ou add_named_inout_arg()
  - Remplir les objets retournés par ces méthodes avec les arguments
- Préciser le type de la valeur de retour avec la méthode set_return_type() qui prend en argument un objet TypeCode

Création dynamique de requête

```
1  Object h = context.resolve_str("Hello");
2  Request request = h._request("hello");
3  Any arg = request.add_in_arg();
4  Any value = orb.create_any();
5  value.insert_string("Marne-la-Vallee");
6  arg.insert_any(value);
7  TypeCode returnType =
8          orb.get_primitive_tc(TCKind.tk_string);
9  request.set_return_type(returnType);
```

Invocation de méthode et récupération du résultat

- Plusieurs modes d'invocation :
  - invocation classique avec invoke(), la méthode bloque tant que la réponse n'est pas reçue
  - invocation oneway avec send_oneway(), la méthode n'attend pas de réponse, elle est déclarée oneway dans l'IDL
  - invocation différée avec send_deferred(), la méthode n'attend pas l'arrivée de la réponse
  - invocations multiples :
    - différées avec orb.send_multiple_requests_deferred()
    - oneway avec orb.send_multiple_requests_oneway()
- Récupération du résultat par get_response() sous la forme d'un objet Any
  - Méthode bloquante en cas d'appel différé
  - poll_response() permet de savoir si la réponse est arrivée

Invocation de méthode et récupération du résultat

```
1  request.invoke();
2  Any result = request.return_value();
3  System.out.println(result.extract_string());
```

La gestion des paramètres out et inout

- Arguments out et inout ne sont pas manipulés via les Holder
- Objet Any associé à l'argument joue le rôle d'enveloppe
- Pour inout, le type de l'argument est précisé par la valeur enveloppée
- Pour out, le type de l'argument doit être précisé par la méthode type de l'objet Any

La gestion des paramètres out et inout

- Pour la méthode :

```
1  void m(in string a, out string b, inout string c);
```

- L'appel est le suivant :

```
1   Object o = ...
2   Request request = o._request("m");
3   Any arg1 = request.add_in_arg();
4   Any arg2 = request.add_out_arg();
5   Any arg3 = request.add_inout_arg();
6   arg1.insert_string("A");
7   arg2.type(orb.get_primitive_tc(TCKind.tk_string));
8   arg3.insert_string("C");
9   request.invoke();
10  System.out.println(arg2.extract_string());
11  System.out.println(arg3.extract_string());
```

La gestion des exceptions

- Comme pour les paramètres out, besoin de préciser les types des exceptions susceptibles d'être levées par la méthode avant de l'appeler
  - Récupérer la liste des exceptions via la méthode exceptions()
  - Ajouter les types des exceptions susceptibles d'être levées avec la méthode add()
- Après l'appel l'exception levée est récupérée via env().exception()
- Elle est stockée dans un objet Any qui est lui-même stocké dans le champs except de l'exception de type UnknowUserException

La gestion des exceptions

- Pour la méthode :

```
1  void m() raises (Reject);
```

- L'appel est le suivant :

```
1  Object o = ...
2  Request request = o._request("m");
3  request.exceptions().add(RejectHelper.type());
4  request.invoke();
5  UnknownUserException ex =
6      (UnknownUserException)request.env().exception();
7  if (ex != null)
8    Any exception = ex.except;
9    throw RejectHelper.extract(exception);
```

Plan de l'exposé

Possibilité de placer des intercepteurs coté serveur

- Préciser grâce à une propriété org.omg.PortableInterceptor.ORBInitializerClass.nom.de.classe.OrbInitializer, le nom de la classe d'un objet CORBA implantant l'interface ORBInitializer
- Implanter les méthodes de l'interface ORBInitializer
- Dans l'implantation de la méthode post_init() installer, grâce à la méthode add_server_request_interceptor() de l'argument de type ORBInitInfo un intercepteur de requête
- Intercepteur de requête est un objet CORBA qui implante ServerRequestInterceptor
  - La méthode name() permet de lui donner un nom unique
  - Les autres méthodes (receive_request(), send_reply, . . .) permettent d'intercepter la requête à différents moments

Intercepteurs de requêtes

- Préciser la classe de l'objet a installer :

```
1    System.setProperty("org.omg.PortableInterceptor.ORBInitializerClass."+
2                       "fr.umlv.ir3.corba.cours5.MyServerORBInitializer","");
3    ORB.init(args, null);
```

- Implanter l'intercepteur d'initialisation :

```
1    public class MyServerORBInitializer extends LocalObject
2                                        implements ORBInitializer {
3      public void pre init(ORBInitInfo info) {}
4      public void post init(ORBInitInfo info) {
5         MyServerRequestInterceptor inter = new MyServerRequestInterceptor();
6         try {
7           info.add_server_request_interceptor(inter);
8         } catch (DuplicateName e) {
9           throw new RuntimeException(e);
10        }
11        System.out.println("Intercepteur Serveur enregistre");
12     }
13   ...
14   }
```

Intercepteurs de requêtes

- Implanter l"intercepteur de requêtes :

```
1  public class MyServerRequestInterceptor extends LocalObject
       implements
2      ServerRequestInterceptor {
3    public String name() {
4      return "MyServerRequestInterceptor";
5    }
6    public void receive_request(ServerRequestInfo ri)
7          throws ForwardRequest {
8      System.out.print("Requete concernant un objet de type :
         ");
9      System.out.println(ri.target_most_derived_interface());
10     System.out.print("En appelant la methode: ");
11     System.out.println(ri.operation());
12   }
13   public void send reply(ServerRequestInfo ri) {}
14   public void send exception(ServerRequestInfo ri)
15         throws ForwardRequest {}
16   public void send other(ServerRequestInfo ri) throws
       ForwardRequest {}
17   public void destroy() {}
18   public void
       receive_request_service_contexts(ServerRequestInfo
       ri)
19         throws ForwardRequest {}
20 }
```

Possibilité de placer des intercepteurs coté client

- Préciser grâce a une propriété org.omg.PortableInterceptor.ORBInitializerClass.nom.de.classe.OrbInitializer, le nom de la classe d'un objet CORBA implantant l'interface ORBInitializer
- Implanter les méthodes de l'interface ORBInitializer
- Dans l'implantation de la méthode post_init() installer, grâce a la méthode add_client_request_interceptor() de l'argument de type ORBInitInfo un intercepteur de requête
- Intercepteur de requête est un objet CORBA qui implante ClientRequestInterceptor
  - La méthode name() permet de lui donner un nom unique
  - Les autres méthodes (send_request(), receive_reply, . . .) permettent d'intercepter la requête à différents moments

Intercepteurs de requêtes

- Préciser la classe de l'objet a installer :

```
1    System.setProperty("org.omg.PortableInterceptor.ORBInitializerClass."+
2                       "fr.umlv.ir3.corba.cours5.MyClientORBInitializer","");
3   ORB.init(args, null);
```

- Implanter l'intercepteur d'initialisation :

```
1   public class MyClientORBInitializer extends LocalObject
2                                       implements ORBInitializer {
3     public void pre init(ORBInitInfo info) { }
4     public void post init(ORBInitInfo info) {
5        MyClientRequestInterceptor inter = new MyClientRequestInterceptor();
6        try {
7          info.add client request interceptor(inter);
8        } catch (DuplicateName e) {
9          throw new RuntimeException(e);
10       }
11       System.out.println("Intercepteur client enregistre");
12    }
```

Intercepteurs de requêtes

- Implanter l'intercepteur de requêtes :

```
1   public class MyClientRequestInterceptor extends LocalObject
2                                       implements ClientRequestInterceptor {
3     public String name() {
4       return "MyClientRequestInterceptor";
5     }
6     public void send_request(ClientRequestInfo ri)
7           throws ForwardRequest {
8       System.out.print("Requete concernant un objet : ");
9       System.out.println(ri.target());
10      System.out.print("En appelant la methode: ");
11      System.out.println(ri.operation());
12    }
13    public void send_poll(ClientRequestInfo ri) {}
14    public void receive_reply(ClientRequestInfo ri) {}
15    public void receive_exception(ClientRequestInfo ri)
16          throws ForwardRequest {}
17    public void receive_other(ClientRequestInfo ri)
18          throws ForwardRequest {}
19    public void destroy() {
20    }
21  }
```

Conclusion

CORBA

- Solution non propriétaire
- Libre choix de l'implémentation et du langage
- Solution fonctionnelle d'interopérabilité
- Plusieurs implémentations libres et open-source
- MAIS : un peu usine à gaz... (bon courage pour le projet :-))