Plates-formes de déploiement et d'exécution d'applications

Stéphane Frénot

CITI / ARES - INSA Lyon

Contexte

- Explosion d'internet (24/24, 7/7)
- Révolution Java (MV, Langage, Bibliothèque)
- Emergeance de l'informatique ubique (embarqué, domotique, adHoc...)

- ==> Définition de plates-formes permettant la gestion et l'exécution garantie d'applications pour les clients

Plan

- Historique
- Motivations
- Les plates-formes existantes
- La programmation orientée composants
- Conclusion

Origine du problème : La programmation socket

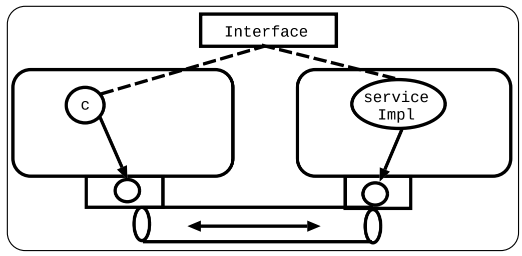
- Deux machines distantes peuvent communiquer
- Les deux machines communiquent selon un protocole d'échange
- Deux principes le mode flux le mode packet

=>Programmation non-fonctionnelle

==> Masquer la couche réseau au programmeur

Historique : Appel distant

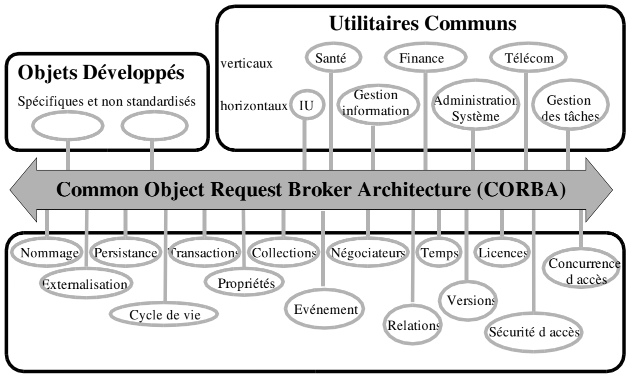
- Masquer à un client la localisation du service
- Notion d'interface de service
- Infrastructure neutre de gestion : RPC, ORB, RMI




```
Resultat res=service.calcul(param);
```


Historique : CORBA 2

- CORBA : le bus d'interconnexion
  - Démons de gestion du bus
  - Notion de services de base
  - Cycle de développement
  - Les services sont accessibles de manière homogène sur le bus
  - Corba est tout à la fois (middleware, framework, indépendant...)
  - Mais cela reste une spécification...
  - Naissance du middleware
- Composant : Unité d'exécution décrite par son interface dans un langage neutre
- Services : Corba Services, Corba Facilities




```
Service serv=CosNaming.lookup("Service");
Res res=t.calcul();
```


Historique : les EJB

- Un objet de service doit être identifié (marké)
- Il n'y a pas que le code réseau qui ne soit pas du code fonctionnel (sécurité, transaction, traces)
- Un composant peu intéragir avec des services standards
- Notion de déploiement du composant sur le middleware

Historique : les EJB

Les EJB : Enterprise Java Beans

- Composants logiciels (interface de description)

Historique : les EJB

Les EJB : Enterprise Java Beans

- Composants logiciels (interface de description)
- Famille de composants (Entity, Session SL/SF, Message Driven)
- Une usine permet au client d'obtenir la référence au service
- Conteneur réalise une interposition Client/Service
- Intégration de la phase de déploiement
- Java / RMI

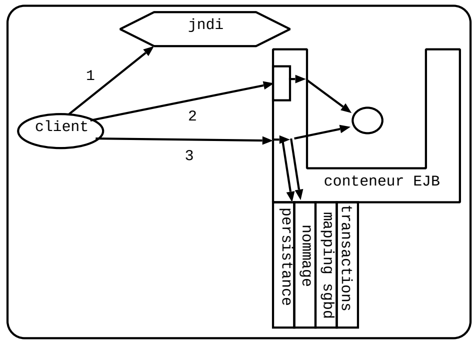
Les EJB

- Un composant EJB : Unité de packaging (déploiement + exécution) décrite par
  - Une interface de service (RemoteInterface)
  - Une interface de cycle de vie pilotée par le conteneur (HomeInterface)
  - Un descripteur de déploiement générique (indépendant du conteneur)
  - Un descripteur de déploiement spécifique

Le conteneur

- Les services du conteneur d'EJB
  - Passivation
  - Mapping SGBDR
  - Gestion des Transactions
  - Sécurisation des appels
  - Annuaire pour retrouver un objet
  - Invocation distante
- Les services définissent l'ensemble des appels non fonctionnels d'une application.
- La phase de déploiement met en place les liaisons entre un composant et les services

Creation d'EJB

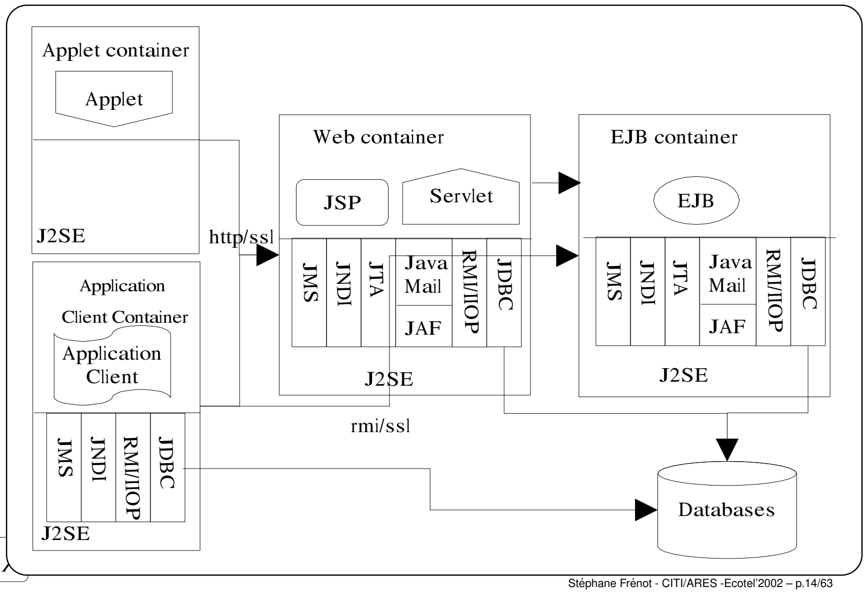



```
ServiceHome servHome=JNDI.lookup("ServiceHome");
Service serv=servHome.create();
Resultat res=serv.calcul();
```


Historique 4/

La J2EE

- Plate-forme de services définie par Sun
- Conteneurs (EJB, WEB, Applets)
- Repose sur l'API Java
- Intègre un ensemble de services standards



- La plate-forme ne spécifie pas comment sont exécutés/organisés les services
- La plate-forme ne permet pas d'extensions
- Dépend des spécifications de Sun (JSR)

Motivations pour une plate-forme de services

Plate-forme d'exécution de services

- Intégration de nouveaux services (mobilité, adaptabilité...)
- Optimisation des services en fonction des besoins, et de l'environnement d'exécution
- Efficacité d'exécution
- Gestion dynamique des services exécutés
- Fédération de plate-formes

Intégration d’un moteur de reconnaissance de la parole

- Intégration d’un moteur de reconnaissance dans un serveur J2EE
- Interfacer le moteur avec le conteneur d’EJB

Plates-formes d’exécution et de déploiement de services

Plates-formes d'exécution de services

- Définir un service spécifique (interface/composant)
- Déployer le service (nommage)
- Piloter l'exécution du service (cycle de vie)
- Piloter les échanges entre services (communication)
- Plates-formes
  - JMX/JBoss : L'instrumentation réseau au service de la J2EE
  - OSGi : passerelle de services
  - Avalon : un framework pour le développement de services

JMX : Java Management eXtension

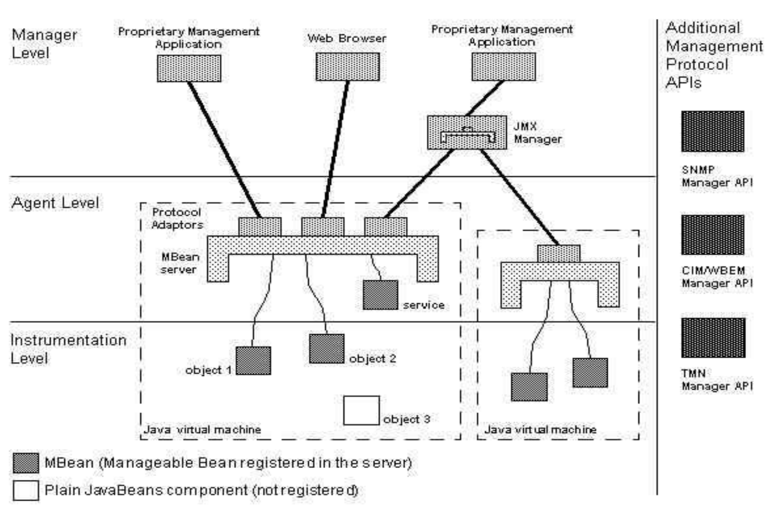
Standard pour instrumenter des ressources pilotables

- une API et une architecture pour accéder et contrôler des classes Java
- Instrumentation, Agent et Manager
- Les composants JMX
  - MBeanServer
  - MBeans (Standard, Dynamic, Model)
  - Notification
  - Adaptors
  - Connectors

JMX : MBean Server

- MBeanServer
  - Gère le cycle de vie des MBeans (creation, enregistrement, suppression)
  - Fournit des services aux MBeans (nommage, notification, relation, surveillance, timer)
- MBean
  - Classe concrête (service)
  - Elle implante une extension spécifique de l'interface MBean
  - Elle presente un constructeur publique

JMX : Vue d'ensemble



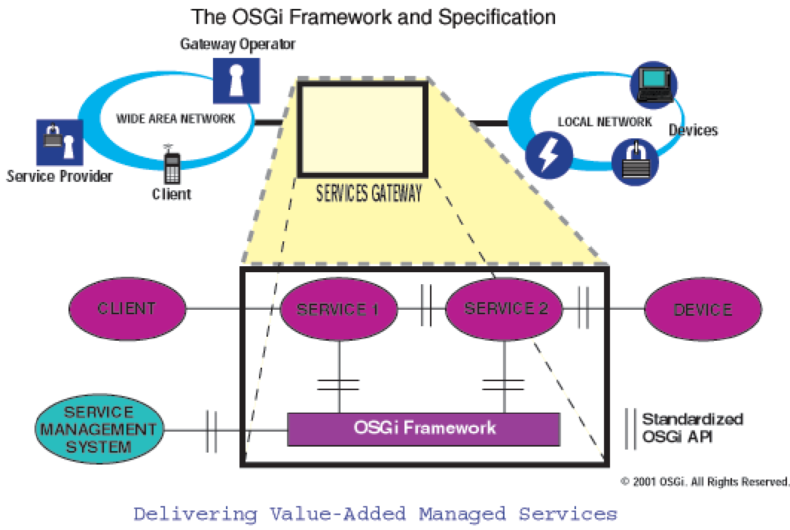
JBoss : J2EE / JMX

- JBoss utilise un noyau JMX pour exécuter les services de la plate-forme J2EE
- Chaque service de la plate-forme est un MBean, répondant au moins à une interface imposée par JBoss
- Le coeur de JBoss est un MBeanServer
- Les services offerts par JMX permettent aux services de mieux fonctionner

OSGi

- Passerelle de service (Service Gateway)
  - Périphérique qui joue le rôle de point d'entrée dans un environnement de type réseau local, où les "services" peuvent être déployés par des fournisseurs de services
- Framework
  - Le conteneur qui met à disposition des services un environnement d'exécution et l'environnement qui permet de les piloter
- Service
  - Interface java avec une sémantique d'implantation

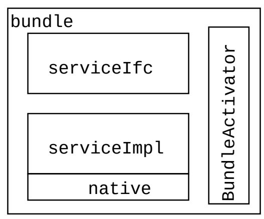
OSGi



Bundle

- Unité d'implantation et de déploiement des services OSGi
- Une archive jar OSGi peut contenir
  - Classes java
  - Bibliothèques java emballées
  - Librairies de code natif
  - Un manifest de description du service
- Les bundles enregistrent les services dans le framework
- Les bundles ne peuvent interagir avec d'autres bundles que par la notion de service (POS)

- Le bundle : Unité de packaging (composant)
  - jar
  - manifest
  - package : {Import|Export}-Package
  - BundleActivator : Services / Package
  - service : {Import|Export}-Service
  - Status : installed, resolved, active, uninstalled, starting, stopping

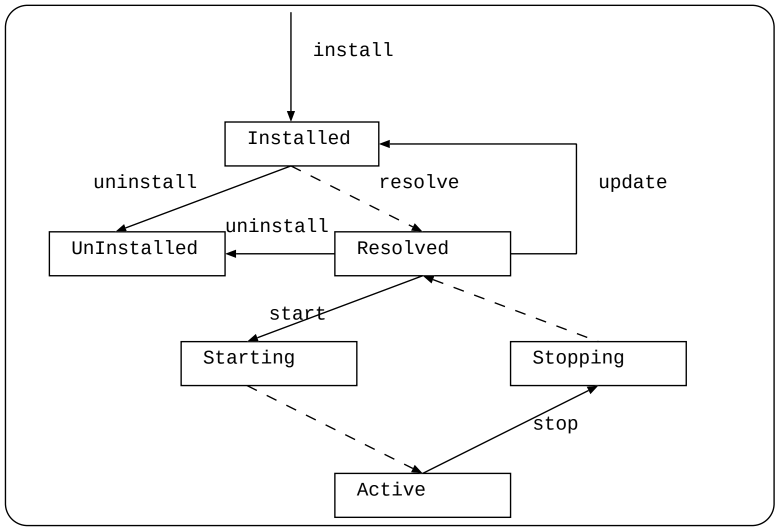


Exemple de manifest :

```
Manifest-Version: 1.0
Created-By: stephane.frenot@insa-lyon.fr
Bundle-NativeCode: \\
  libinsertcard.so;osname=Linux;processor=x86
Import-Package: insa.device.pcmcia.device,\\
  org.osgi.service.device;specification-version="2.1"
Bundle-ClassPath: .
Export-Package: insa.device.wlan.device;\\
  specification-version="2.1"
Bundle-Activator: insa.device.pcmcia.driver.PcmciaDriverImpl
```

- Les services
  - Evénements, Securité
  - Admin, ServiceTracker
  - Log, http
  - Device Access

Cycle de vie d’un Bundle



Avalon

Avalon

- Projet Jakarta (http://jakarta.apache.org)
- **Framework** qui permet à des composants de différentes tailles d'être créés et gérés par un cycle vie spécifique.
- Tout est organisé autour de conteneurs et de composants standards
- Approche fortement guidée par les DP

Concepts de développement

- Inversion du controle (IOC) : Principe qu'un composant est toujours piloté de l'extérieur. Tout le cycle de vie d'un composant est géré par le code qui l'a créé. (notion de conteneur)
- Séparation des objectifs : Chaque objectif d'un service est capturé dans une interface spécifique. Principe similaire à la programmation par aspects.
- Programmation orientée composants : Chaque composant présente une interface de fonctionnement et des contrats spécifiques autour de l'interface
- Programmation orientée services

Avalon définitions

- Composant : Combinaison entre une interface de travail et une implantation de l'interface. Son utilisation permet un couplage plus souple qu'entre objets, dans la mesure ou l'implantation peut changer indépendamment des clients
- Service : Groupe d'un ou plusieurs composants fournissant une solution complète. Exemple : gestionnaire de protocole, ordonnanceur, authentification...

L'organisation des appels

1. LogEnabled
2. Contextualizable
3. Composable
4. Configurable
5. Parameterizable
6. Initializable
7. Startable
8. Suspendable
9. Recontextualiazble
10. Recomposable
11. Reparameterizable
12. Stoppable
13. Disposable

Avalon conclusion

- Approche structurante pour le développement d'applications à base de composants
- Pas de framework de base
- Communauté openSource
- Produits complet monté sur le framework Cocoon, Phoenix
- Approche déclarative des interactions entre composants ADL représenté en XML

Projets Approches connexes

- JavaBeans
- Java Network Launcher Protocol (JNLP)
- NetBeans / Eclipse
- OpenWings
- Jini, CoolTown, .NET
- Fractal, http://www.objectweb.org
- ...

La Programmation Orientée Services

La programmation Orientée Services

- Eléments
  - Contrats (Interfaces)
  - Composants (Services)
  - Connecteurs (Interactions)
  - Conteneurs (Cycle de vie)
  - Contextes (Environnement)
- Aspects
  - Conjonctivité
  - Déploiement
  - Mobilité
  - Sécurité
  - Disponibilité
  - Intéropérabilité

Et Java ?

- Les plates-formes sont proposées initialement sur java
- La machine virtuelle met à disposition des services spécifiques pour la programmation orientée services
  1. L'architecture des classLoader
  2. L'accès au byte-code
  3. Les proxys dynamiques

Classloader java

- Permet de piloter finement le chargement d'une classe java
- Permet de piloter finement le chargement des ressources
- Permet de gérer des versions d'une même classe
- Permer de rendre le code client totalement indépendant du code de service (à l'interface près)
- On utilise les constructions de type Class.forName()

Exemple CL

```
public interface Point {
  public void move (int dx, int dy);
}

public class PointImpl {
  public void move (int dx, int dy){
    this.x+=dx; //erreur
  }
}
```

==> Approche Composant

Exemple CL

```
public class Client {
  public static void main(String [] arg){
    Point p1=new Point();
    p1.move(10,10);
  }
}
```

- La correction du serveur impose l'arrêt du client
- Client et serveur sont dépendants
- Pour les rendre indépendants il faut un conteneur/intercepteur/fabrique...
- Le modèle est Client/Conteneur/Serveur

Exemple CL

```
public class ServeurDePoint{
  static ClassLoader c1;
  static Class ptClass;
  public static Point createPoint{Point tmp){
    Point newPt=(Point)ptClass.newInstance();
    if (tmp!=null){
      newPt.move(tmp.getX(), tmp.getY();
    }
    return newPt;
  }

  public synchronized void reloadImpl() throws Exception{
    URL[] serverURLS=new URL[]{new URL("file:subdir/")};
    cl=new URLClassLoader(serverURLS);
    ptClass=cl.classLoader("PointImpl");
  }
}
```

Le loader permet de charger dynamiquement une nouvelle version de classe

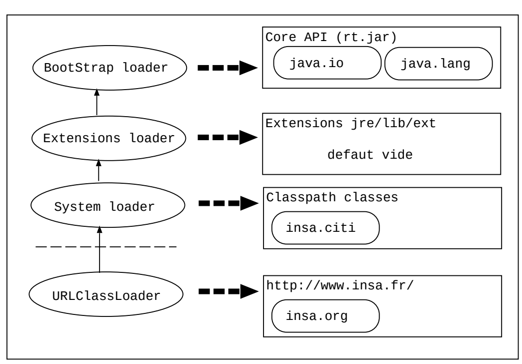
Exemple CL : client

```
public class Client{
  static Point pt;
  public static void main(String [] arg){
   pt=PointServer.createPoint(null);
   while (true){
     ...
     pt.move(1,1); // pt=1,0
     ...
     PointServer.reloadImpl();
     pt=PointServer.createPoint(pt);
     pt.move(1,1); // pt=1,1
   }
 }
}
```

Le client

1. Ne peut pas référencer l'implantation
2. Les versions d'implantation et d'interface sont les mêmes
3. Il faut pouvoir transmettre l'ancien état
4. Le client doit fournir la nouvelle et détruire l'ancienne référence

Les classloaders standards



==> Securité !

- java -Djava.ext.dirs=myext MaClasse
- grant codeBase "file:${java.home}/lib/ext/*"{permission java.security.AllPermission;};

Le Byte Code Java

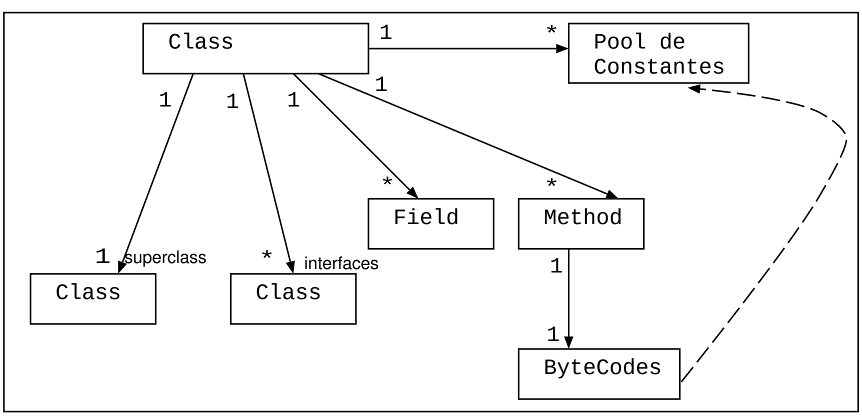
```
public class Test{
  public void test1 () { System.out.println("Coucou 1");}
  public void test2 () { System.out.println("Coucou 2");}
}

public class ClientTest {
  public static void main(String [] arg){
    Test t=new Test();
    t.test1();
}
```

Si Test est modifiée, que se passe-t'il pour ClientTest ?

- en C++ l'invocation d'une méthode est résolue par un offset mémoire
- en Java le byte code maintient des meta data, qui indiquent que la classe serveur n'est plus compatible

Schéma de classes

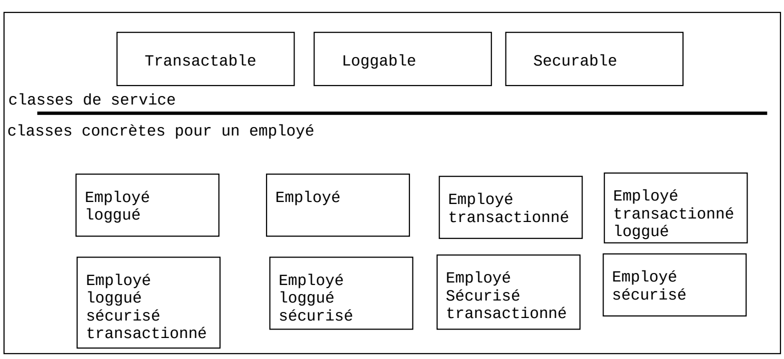


- javap Test ==> Renvoie les structures de la classe
- javap -c ==> Présente le bytecode de la classe
- nombreux autre outils (bcel, jad...)

Les proxys dynamiques

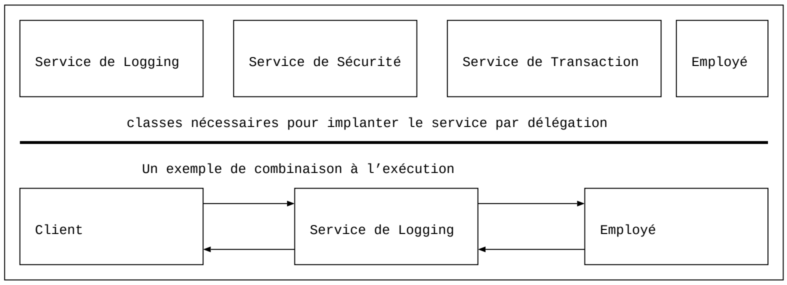
- **Reflexion** : Un client utilise une API générique pour appeler des méthodes sur une classe serveur inconnue à la compilation.
- **Proxys Dynamiques** : Un serveur utilise une API générique pour implanter une méthode qui est fournie à l'exécution pour satisfaire le besoin du client.
  - Mécanisme de base pour les intercepteurs génériques (conteneurs J2EE)
  - Modèle de délégation lorsque l'héritage explose

Implantation par héritage



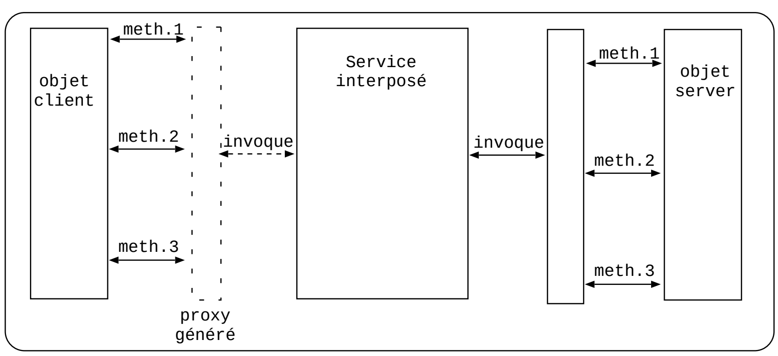
- Explosion de classes
- Evolutivité, nouvelle interface –> X nouvelles classes
- Souplesse, quelles sont les classes concrètes significatives

Implantation par Délégation



- Génération statique de code client (stub) (ex : jonas)
- Proxy dynamique (ex : jboss)

La génération dynamique



Exemple d’interposition : le client

```
PersonneItf p=UsineDePersonne.create("Dupond");
p.getNom();
```

L’usine fabrique normalement un client, mais elle peut interposer un service.

Exemple d’interposition : l’usine

```
public static create(String name){
  PersonneItf personne=new PersonneImpl(name);
  PersonneItf proxy=(PersonneItf)Proxy.newProxyInstance(
    UsineDePersonne.class.getClassLoader(),
    new Class[]{Personne.class},
    new Interposition(personne));
  return (proxy);
}
```

L’usine fabrique un proxy dynamique, conforme à l’interface du client

Exemple d'interposition : le service

```
import java.lang.reflect.*;
public class Interposition implements InvocationHandler{
  private Object delegate;
  public Interposition (Object delegate){ this.delegate=delegate; }
  public Object invoke(Object source, Method method, Object[] args)
      throws Throwable {
    /* Interposition avant */
    Object result=null;
    try{
      result=method.invoke(delegate,args); /* getNom est invoqué */
    }catch(InvocationTargetException e){ throw e.getTargetException(
    /* Interposition après */
    if (result.equals("Dupond")){ result="Frenot"; }
    return result;
  }
}
```

L'interposition agit !

Les avantages des Proxys dynamiques

N'apportent rien par rapport à coder la délégation à la main

1. Ils sont dynamiques et génériques
2. Validation de paramètres
3. Sécurité
4. Propagation de contexte
5. Audit, trace, débuggage
6. Dérouter un appel java vers un autre environnement

==> Coût supérieur à un codage à la main

Performances puissance de 10ns

1. Incrément d'un champ entier : 10e0 (1ns)
2. Invocation d'une méthode virtuelle : 10e1
3. Invocation par délégation manuelle : 10e1-10e2
4. Invocation reflexive : 10e4-10e5
5. Invocation par un proxy dynamique : 10e4-10e5
6. Incrémentation reflexive d'un entier : 10e4-10e5
7. Appel RMI sur une machine locale : 10e5
8. Ouverture de fichiers : 10e6
9. Vitesse de la lumière parcours 3000km : 10e7

Stuart Dabbs Halloway, Component development for the Java Platform

Les autres apports de la MV java

- La sérialisation
- La signature de classe
- La signature d'archives (packaging/deploiement)
- Les interface natives
- Les appels RMI

Conclusion

- De nombreuses plates-formes orientées composants
- Reposent sur java pour la mise en oeuvre
- Les plates-formes sont souvent centrées serveur
- Aucune plate-forme ne répond entièrement aux problèmes soulevés par les réseaux hétérogènes
- Marché du mobile et de l'embarqué
- Séparation des approches distribuées et des approches par composants
- Programmation orientée Aspects

Sites de référence

- http://www.objectweb.org/
  - Framework fractal
  - Jonas
  - OpenCCM
  - Jonathan
- http://jakarta.apache.org
  - Struts
  - Avalon

ressources JMX/JBoss

1. JMX – http://java.sun.com/products/JavaManagement/
2. JDMK – http://java.sun.com/products/jdmk
3. MX4J – http://sourceforge.net/projects/mx4j/
4. JSR 77 – http://java.sun.com/jsr
5. JBoss – http://www.jboss.org

OSGi Références

1. http://www.osgi.org/
2. http://oscar-osgi.sourceforge.net