

# La plate-forme OSGi

Frénot Stéphane

Université de Lyon, INRIA

INSA-Lyon, CITI, F-69621, France

[stephane.frenot@insa-lyon.fr](mailto:stephane.frenot@insa-lyon.fr)

<http://perso.citi.insa-lyon.fr/sfrenot/>

Avec de nombreux transparents fournis par Didier Donsez

# Potentiel de Java

## ■ Industrialisation

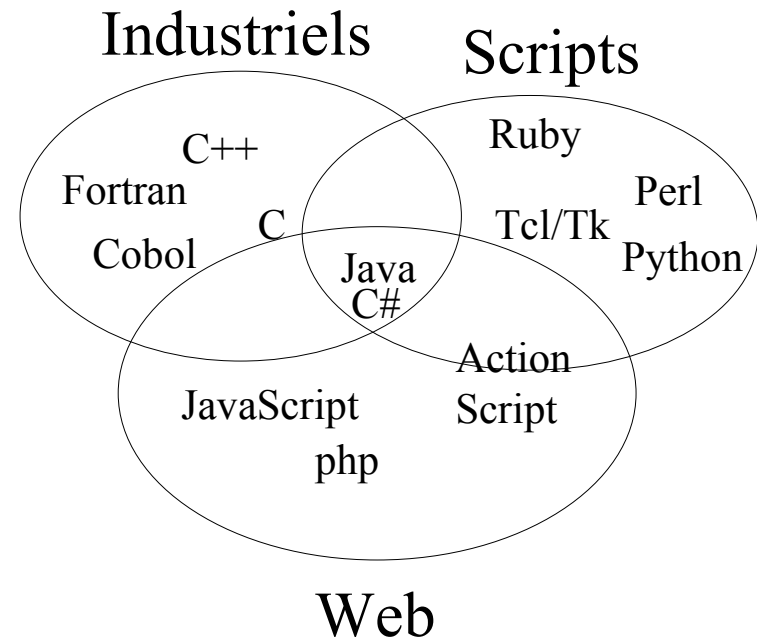
- ◆ Ciblage : serveurs d'applications e-business, téléphonie mobile...
- ◆ Write once Run Everywhere
- ◆ Temps d'apprentissage réduit
- ◆ Bibliothèques standardisées

## ■ Langage adapté au Web

- ◆ Applet, Servlet et EJB
- ◆ Prise en main rapide
- ◆ Surcouches JSP, Struts...

## ■ Langage de script

- ◆ Dynamisme
- ◆ Prototypage
- ◆ Résultats d'exécution immédiats



# Potentiels de Java

---

## ■ Programmation Objet

- ◆ Encapsulation
- ◆ Communication par messages et par notification
- ◆ Polymorphisme, héritage et interfaces

## ■ Machine virtuelle

- ◆ Standardisation des comportements
- ◆ Virtualisation des exécutions
- ◆ Séparation des rôles d'optimisation

# Mais...

---

## ■ Programmation Objet

- ◆ Dynamisme ?
- ◆ Encapsulation parfaite ?
- ◆ Modèles de programmation C/S et Évènementiel est-il simple ?
- ◆ Les bibliothèques standards ne sont spécifiées que par Sun !

## ■ Machine virtuelle

- ◆ Multi-applications ?
- ◆ Pourquoi faut-il la redémarrer ?

# La plate-forme OSGi

---

# Dynamisme Java

---

---

## ■ Au chargement d'une classe

- ◆ Dans le monde compilé, toutes les fonctions sont résolues au démarrage
- ◆ Dans le monde interprété, les fonctions peuvent arriver en cours d'exécution
- ➔ Les Langages Compilés et Interprétés ne sont pas dans le même univers, coexistence permanente des deux classes

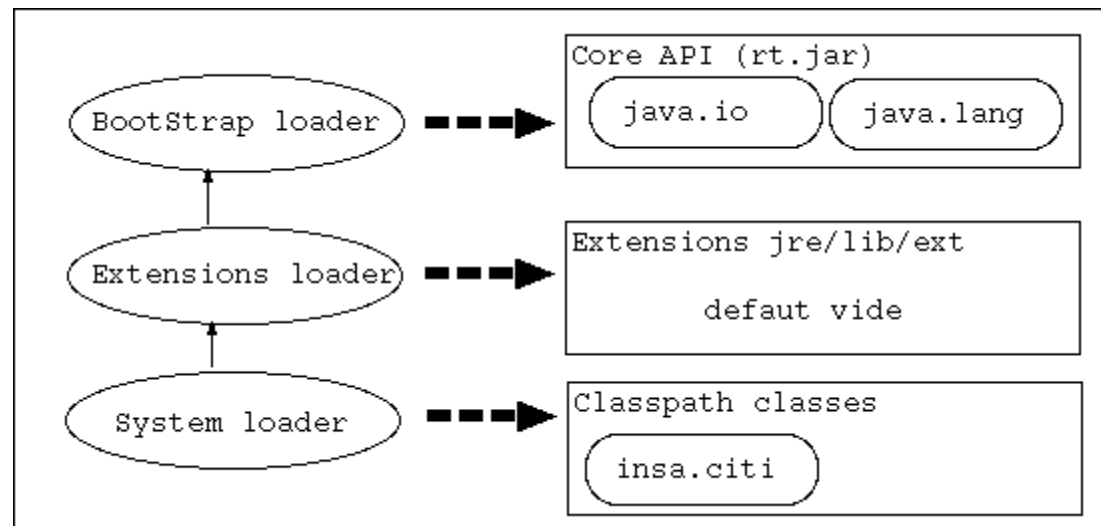
## ■ N'importe quand en cours d'exécution du système

- ◆ Une fois une fonction chargée est-il possible de la remettre en question à l'exécution sans redémarrage du système ?

# Chargement implicite d'une classe

## ■ En java de base : `test.MaClasse exemple=new test.MaClasse()`

- ◆ Le type `test.MaClasse` est contrôlé à la compilation
- ◆ Il est contrôlé et chargé dans la machine virtuelle à l'exécution
- ◆ Une fois chargé il n'est plus remis en question, mais il peut être garbage-collecté

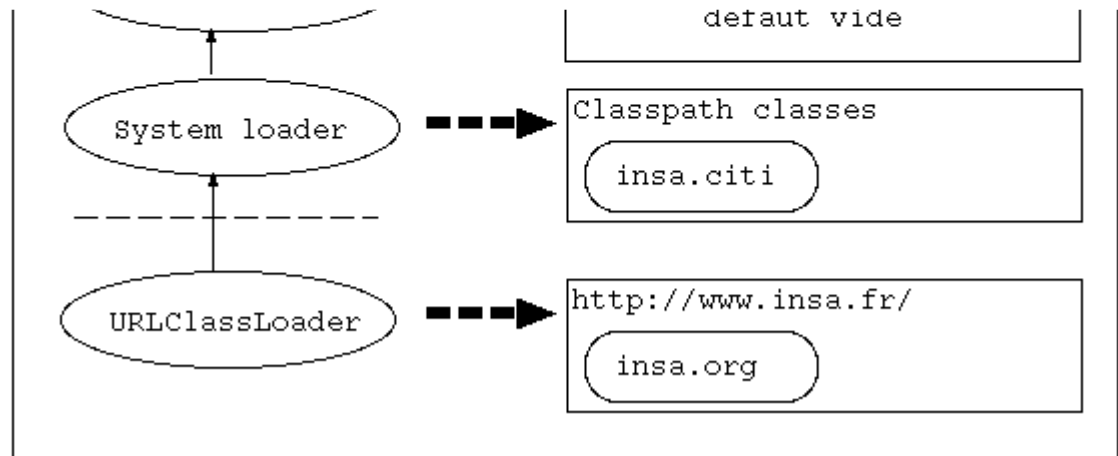


- ◆ **CE N'EST PAS DYNAMIQUE CA !!**

# Chargement explicite d'une classe

## ■ L'architecture des chargeurs de classes est extensible

```
URLClassLoader myCL = new URLClassLoader("http://www.somewhere.biz");  
Class myClass = myCl.load("test.MyClass");  
test.MyClass exemple = (test.MyClass)myClass.newInstance();
```



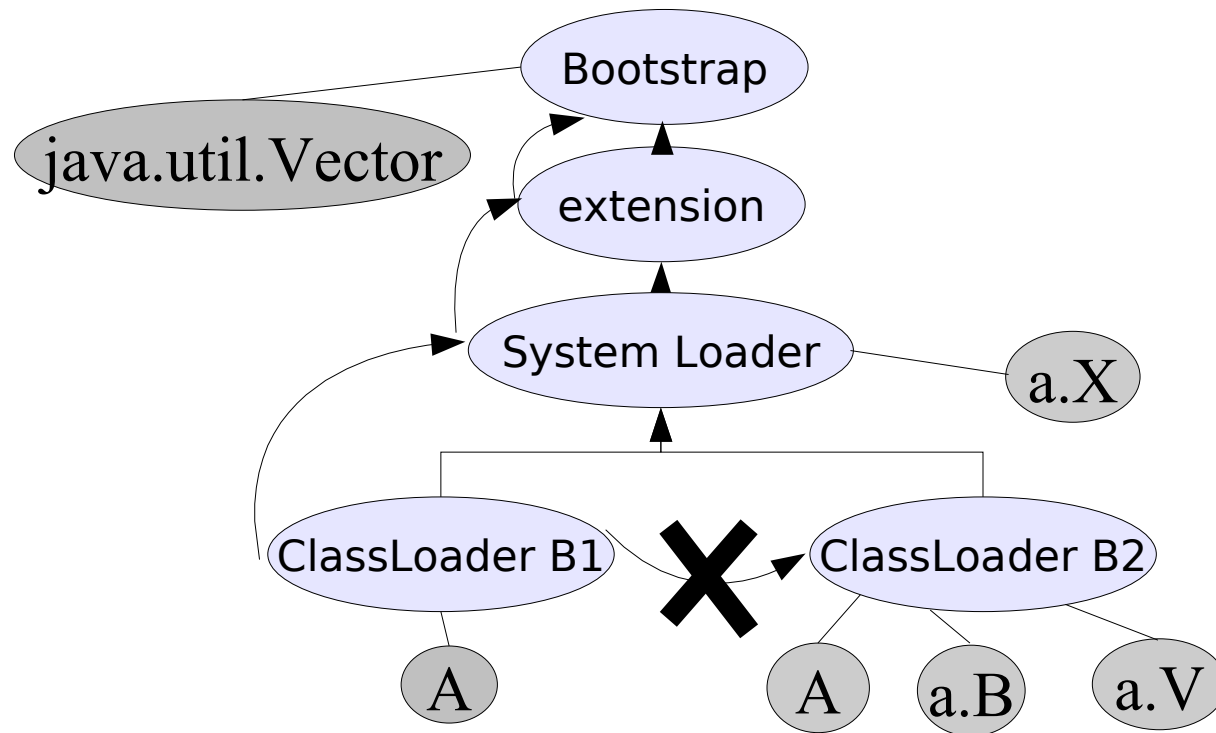
■ Les bytes-codes des classes arrivent à l'exécution (Applet, Java RMI)

■ Une classe est identifiée de manière unique par son nom et son classloader -> Isolation

==> On est dynamique à l'initialisation, mais pas en cours d'exécution

# Chargeur de classe explicite : restrictions

- Une classe doit être chargée par le classloader le plus haut dans une hiérarchie de délégation
- Toutes les classes d'un même package doivent être chargées par un même classloader



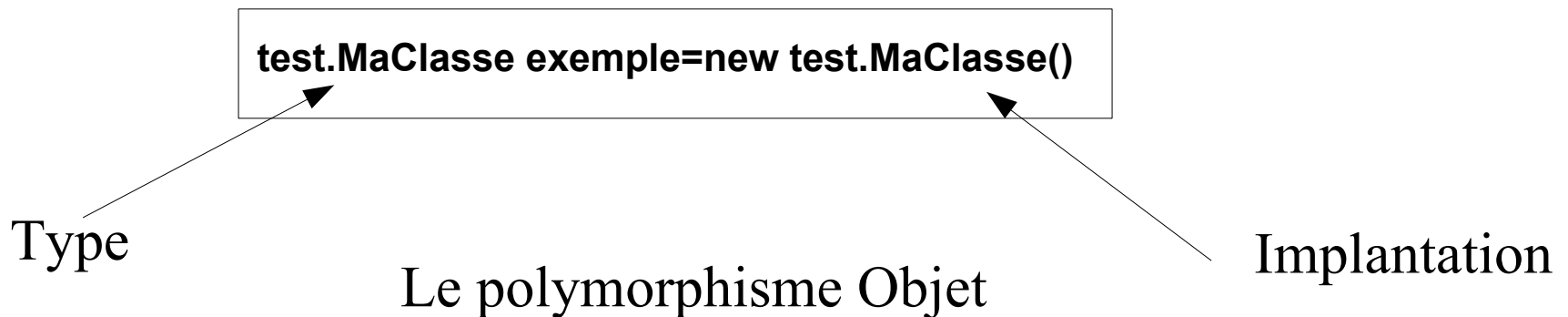
# Programmation orientée services

---

---

## ■ Peut-on être dynamique en dehors de l'initialisation de la classe ?

- ◆ Peut-on obtenir une nouvelle implantation à chaque invocation de méthode ?
- ◆ Peut-on recevoir une nouvelle implantation à chaque mise à jour sur un site distant ?



# Interfaces Java et polymorphisme

---

## ■ On peut écrire :

```
test.MaClasse exemple=new autrechose.Truc();
```

## ■ A conditions que test.MaClasse et autrechose.Truc soient liées

### ◆ Soit par une relation d'héritage

❖ test.MaClasse est une super-classe de autrechose.Truc

### ◆ Soit par une relation d'interface

❖ autrechose.Truc implante l'interface test.MaClasse

## ■ L'approche par l'interface

◆ Permet de fournir une implantation conforme à un type, mais qui peut avoir un comportement tout à fait spécifique, l'implantation réelle est masquée

◆ On peut insérer toute une chaîne d'interposition entre le « client » et l'implantation

◆ Le client n'est plus lié par une relation de typage fort avec l'implantation

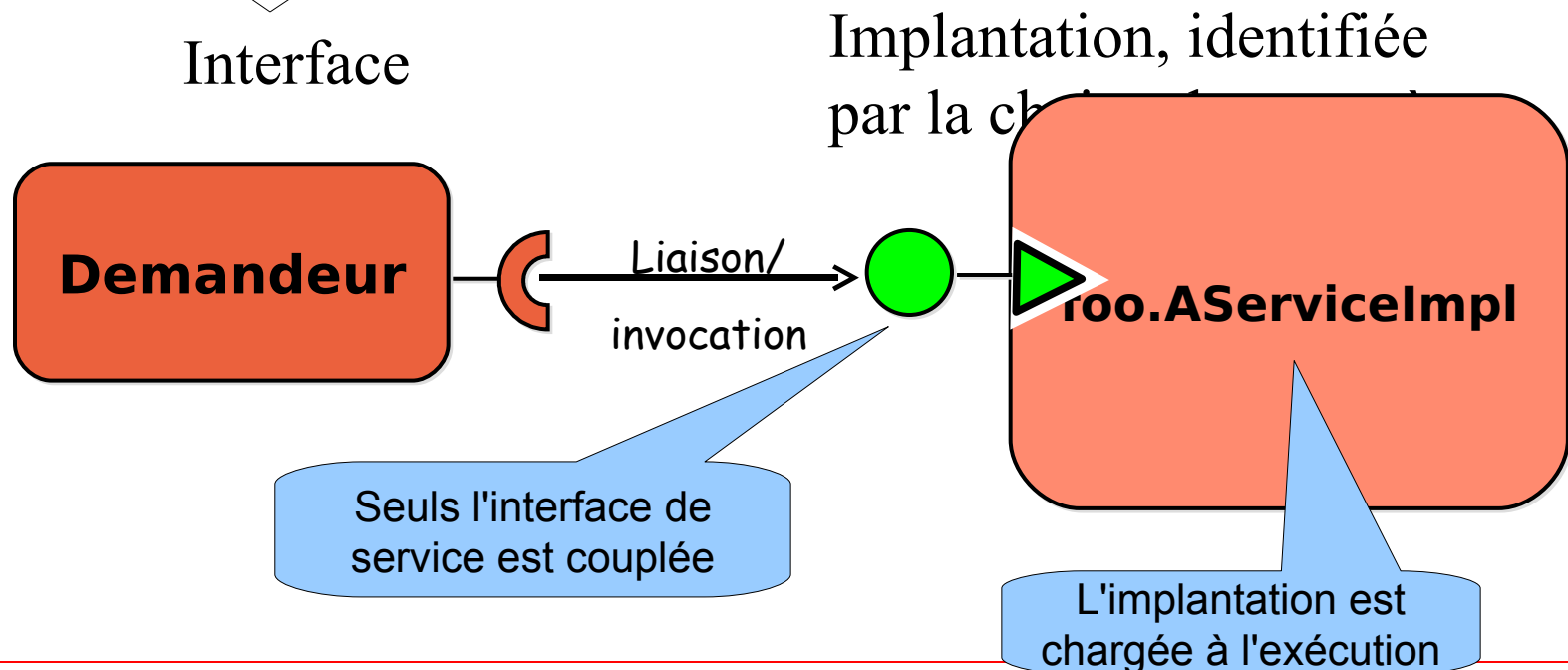
❖ La liaison existe par l'interface, qui est moins contraignante que l'implantation

◆ On peut ajouter des « facettes » aux objets, chaque facette est représentée par une interface

# Mixer interfaces et chargement dynamique

- On peut obtenir un couplage léger entre demandeur et fournisseur d'une classe

```
URLClassLoader myCL = new URLClassLoader("http://www.somewhere.biz");  
Class myClass = myCl.load("foo.AServiceImpl");  
test.Service exemple = (test.Service)myClass.newInstance();
```



# Mixer interfaces et chargement dynamique

---

## ■ Contraintes

```
URLClassLoader myCL = new URLClassLoader("http://www.somewhere.biz");  
Class myClass = myCl.load("foo.AServiceImpl");  
test.Service exemple = (test.Service)myClass.newInstance();
```

## ■ On a perdu le constructeur

- ◆ Il suffit d'avoir une facette/fonction d'initialisation

## ■ Il faut une infrastructure intermédiaire qui automatise la récupération des implantations

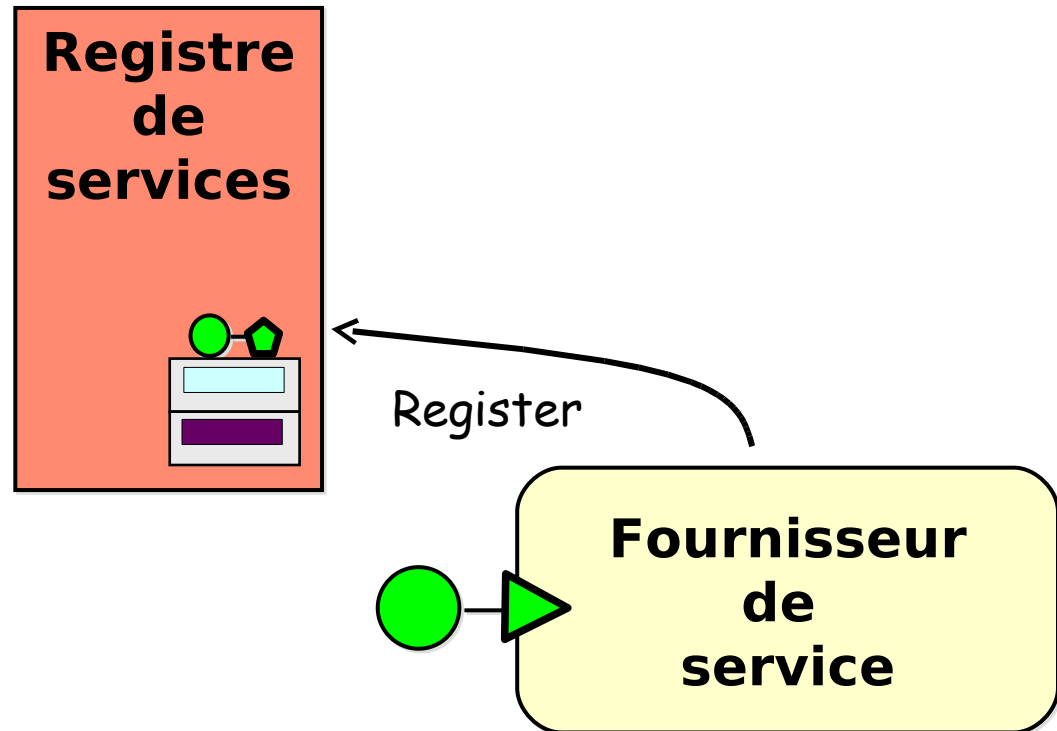
## ■ Le client est quand même lié à l'interface

## ■ Le client doit être adapté pour demander le rechargement d'une nouvelle implantation

- ◆ Un scénario d'usage....

# Programmation orientée services (1/8)

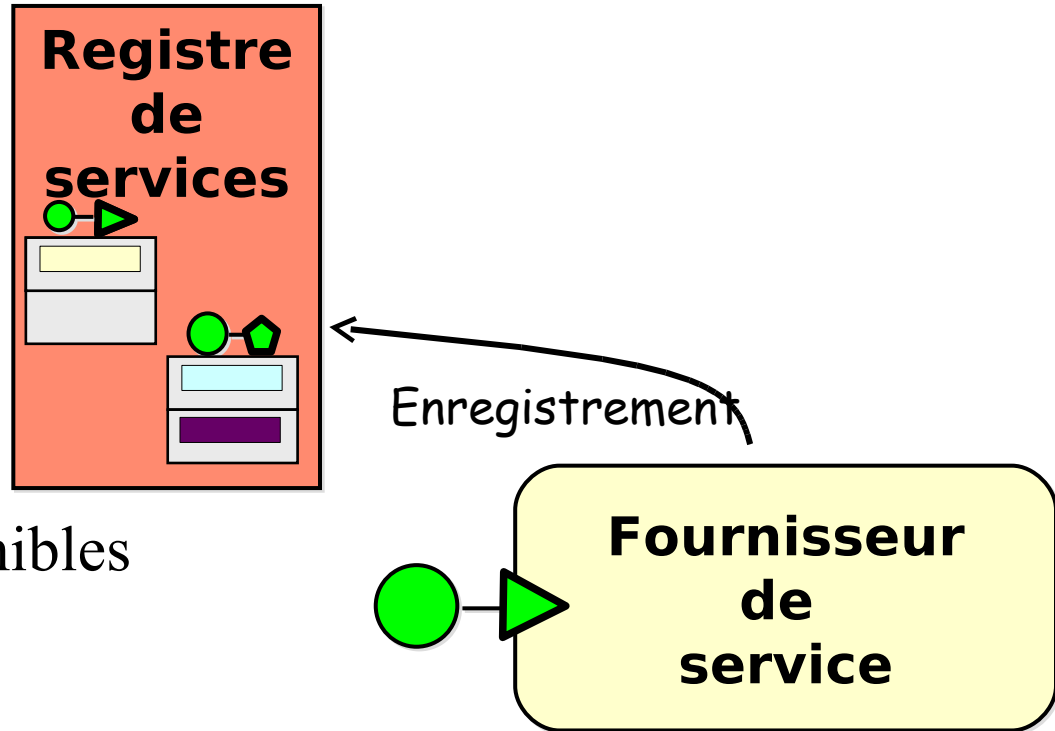
---



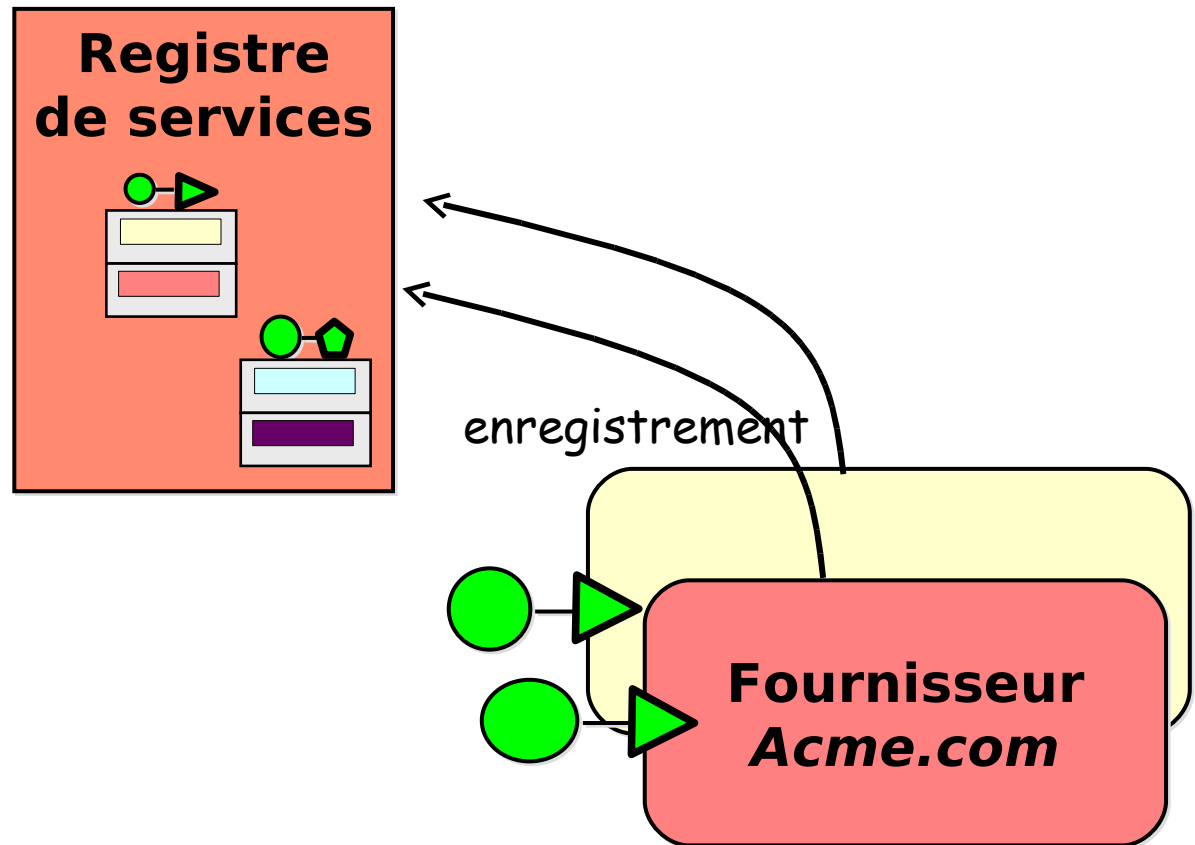
Un service est une implantation conforme à une interface d'enregistrement

## Programmation orientée services (2/8)

L'annuaire enregistre  
pour chaque interface  
les implantations disponibles

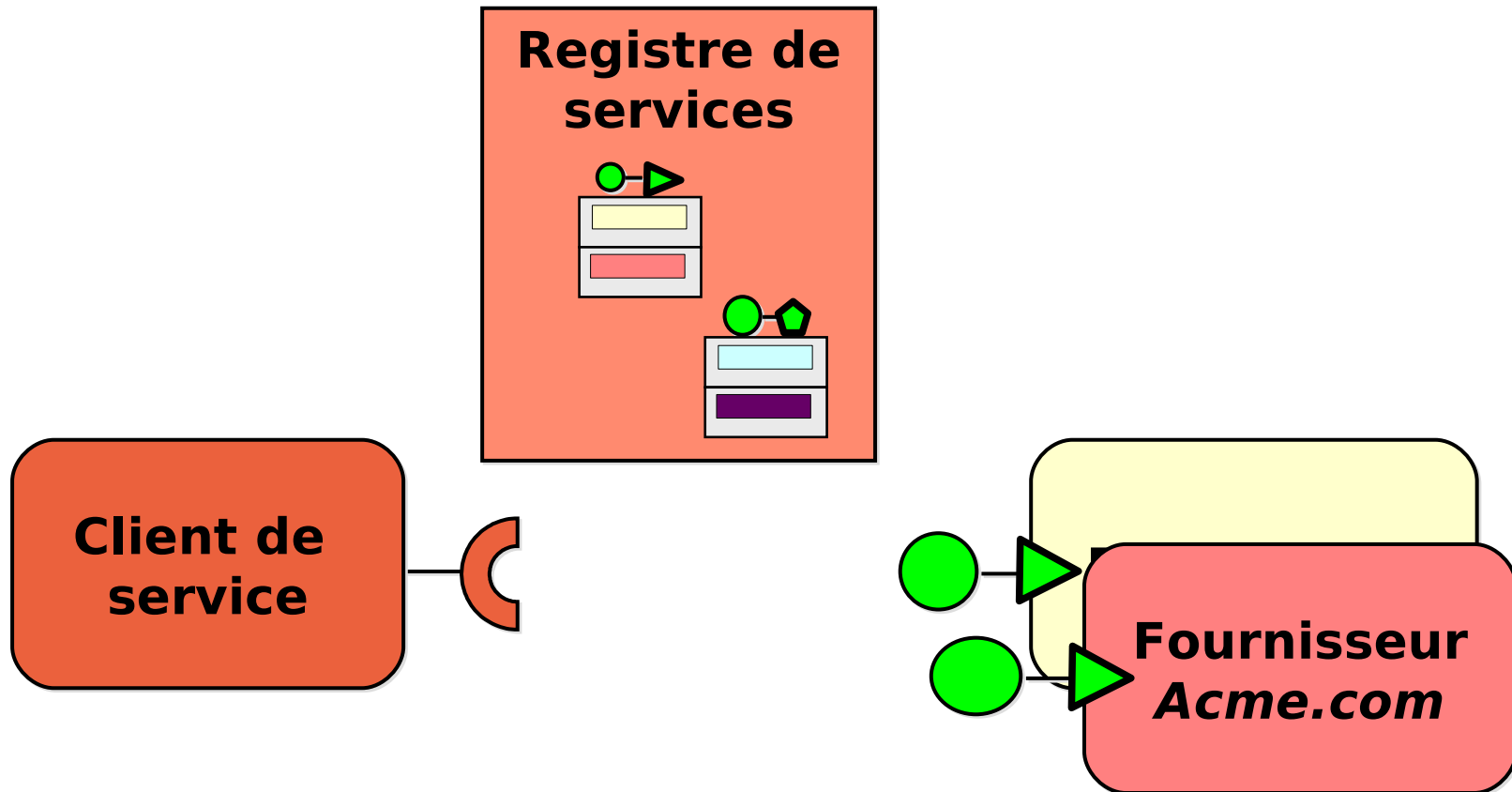


# Programmation orientée services (3/8)



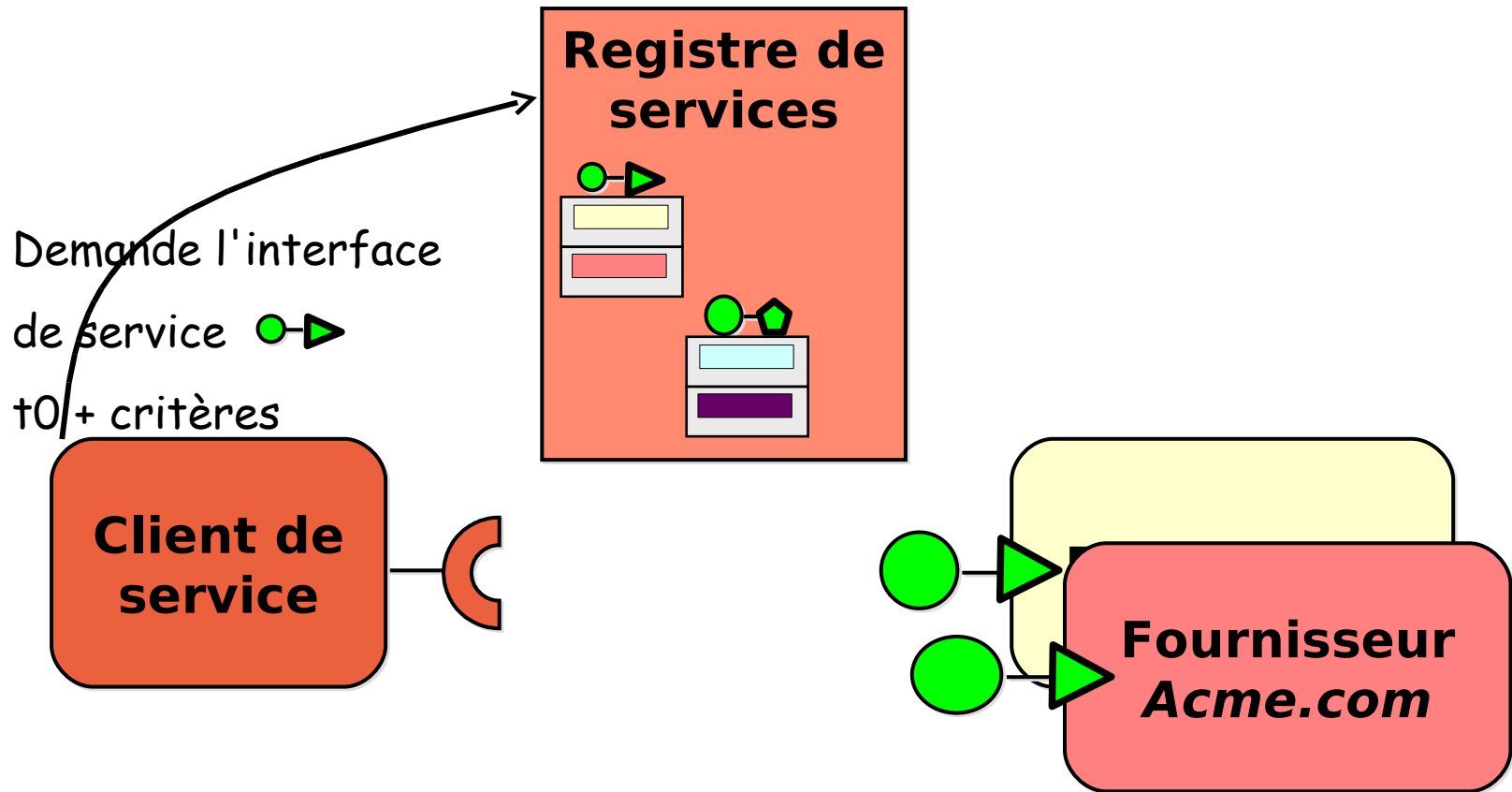
Une même interface peut être fournie par plusieurs fournisseurs d'implantation

# Programmation orientée services (4/8)

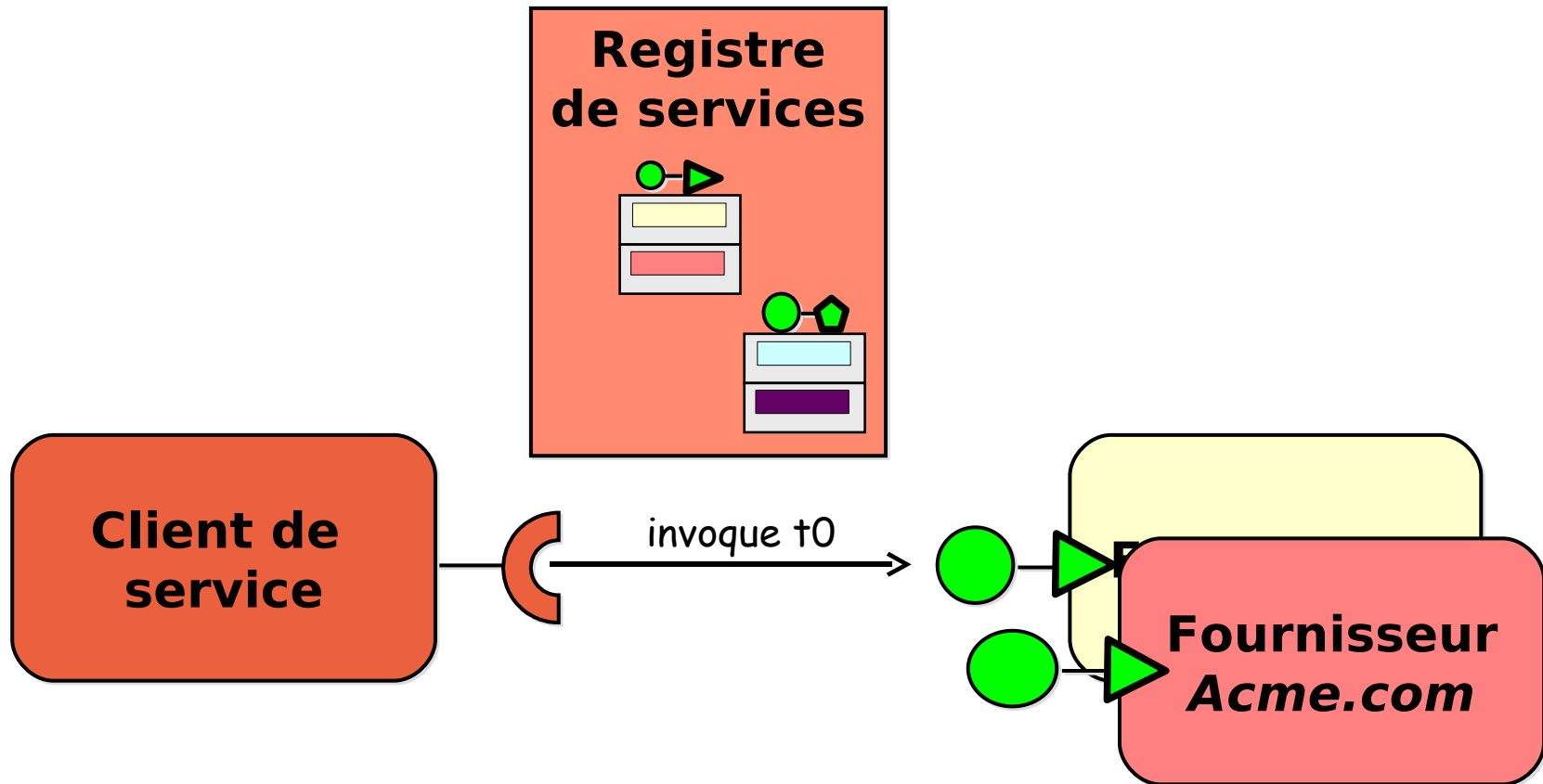


Lorsqu'un client arrive

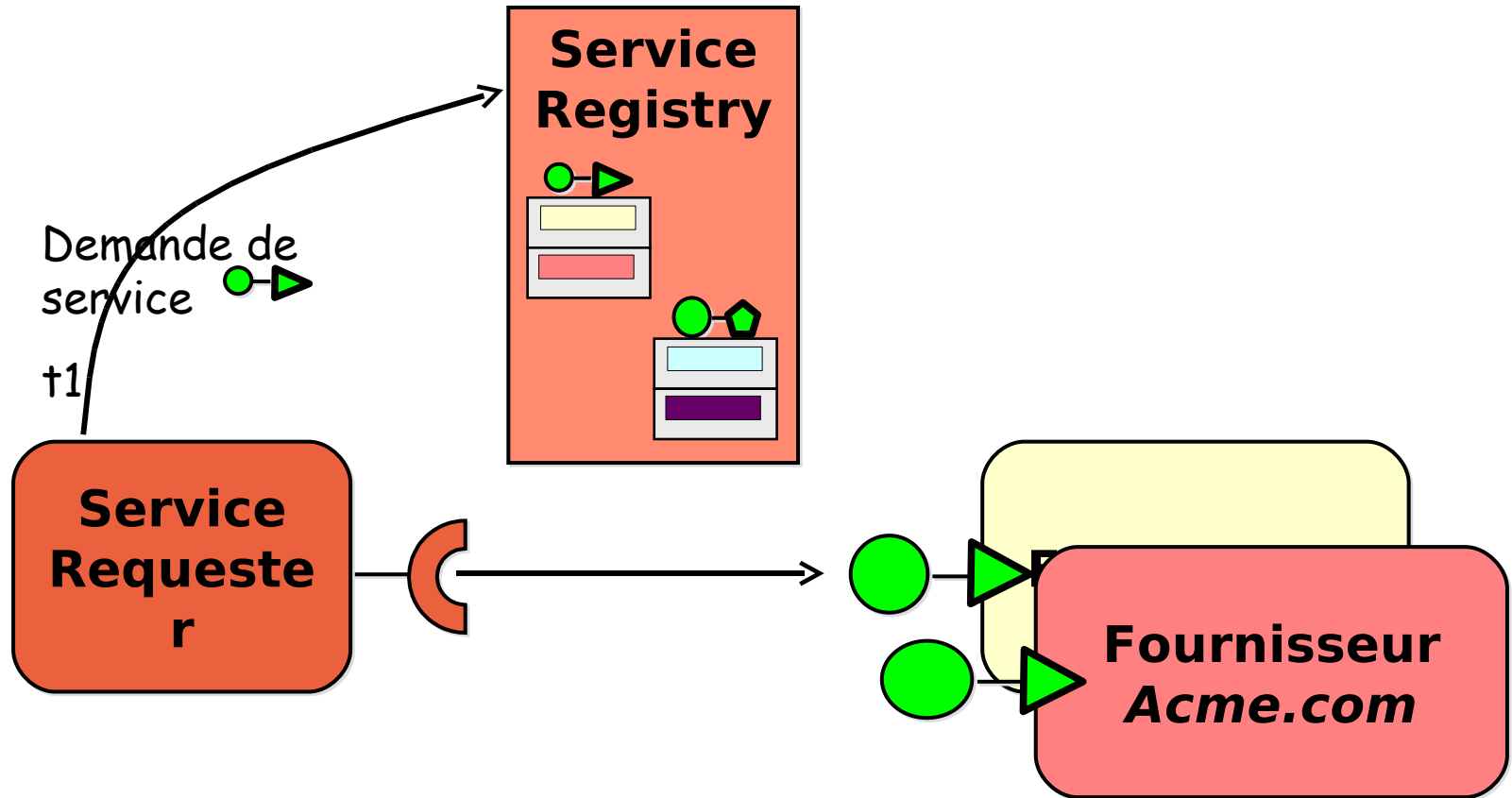
# Programmation orientée services (5/8)



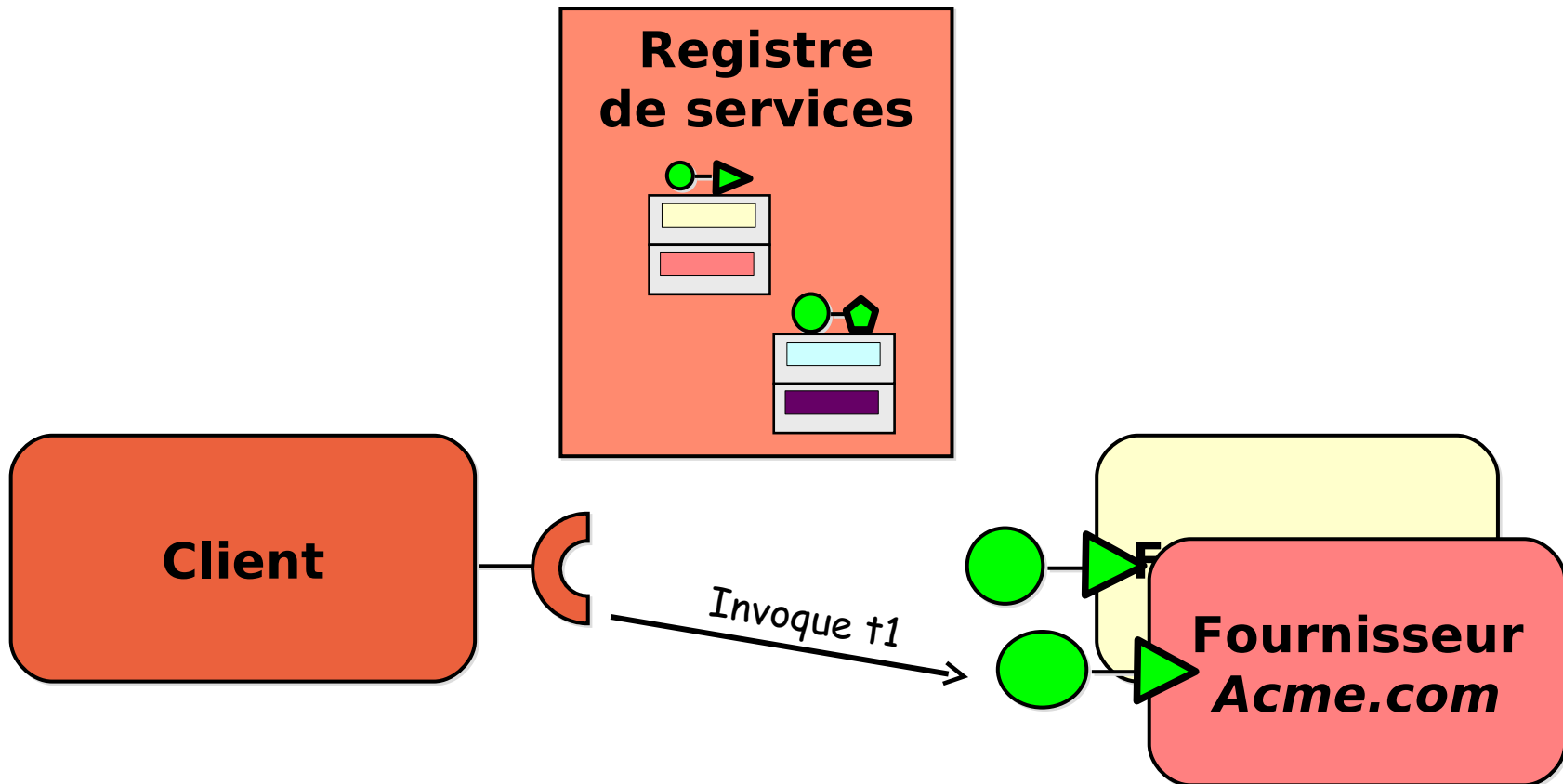
# Programmation orientée services (6/8)



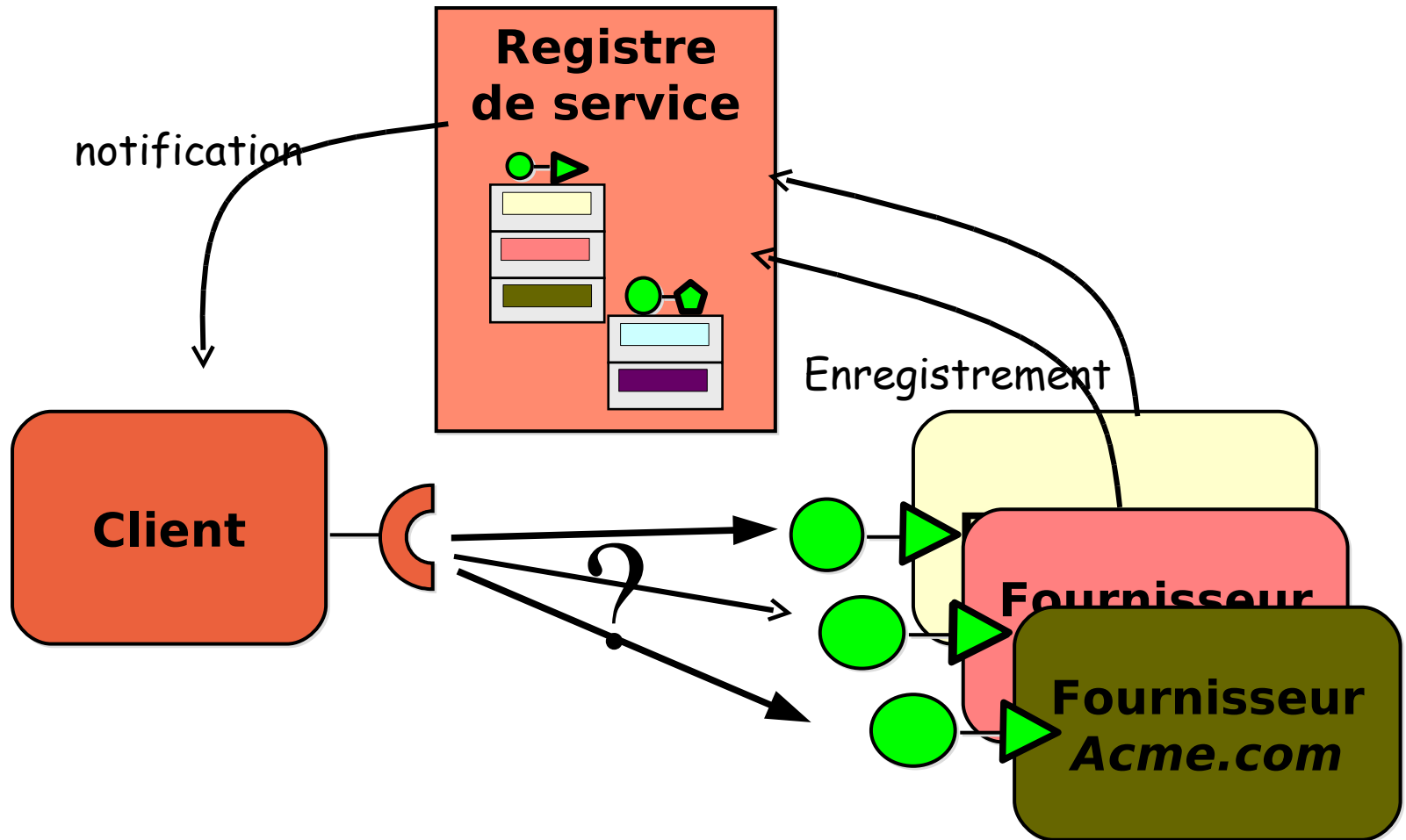
# Programmation orientée services (7/8)



# Programmation orientée services (8/8)



# Programmation orientée services : notifications



# Mais... en exploitant java efficacement

---

## ■ Programmation Objet

- ◆ Dynamisme ?
- ◆ Encapsulation parfaite ?
- ◆ Modèles de programmation C/S et Évènementiel est-il simple ?
- ◆ Les bibliothèques standards ne sont spécifiées que par Sun !

## ■ Machine virtuelle

- ◆ Multi-applications ?
- ◆ Pourquoi faut-il la redémarrer ?

- Chargement dynamique de code
- Programmation orientée services

# Plan

---

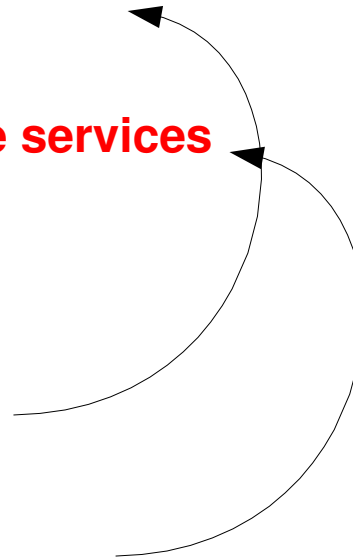
---

## ■ Fondamentaux java

- ◆ Dynamisme
- ◆ Programmation orientée services

## ■ OSGi

- ◆ Introduction
- ◆ Approche composants
- ◆ Approche services



# Qu'est ce que OSGi™ ?

---

## ■ **Spécification OSGi**

- ◆ **définit un canevas de déploiement et d'exécution de services Java**
- ◆ **multi-application, télé-administré**
- ◆ **Cible initiale : set top box, modem cable, ou une passerelle résidentielle dédiée.**

## ■ **OSGi Alliance**

- ◆ **Corporation indépendante**
- ◆ **Soutenus par les acteurs majeurs des IT, home/building automation, telematics (car automation), ...**
- ◆ **de la téléphonie mobiles (Nokia et Motorola)**
  
- ◆ **et Eclipse pour les plugins de son IDE !**
- ◆ **et maintenant Apache pour ses serveurs**

# Qu'est ce que OSGi™ ?

---

## ■ Histoire

- ◆ Mars 1999 : Fondation de l'OSGi Alliance
- ◆ Novembre 1999: SUN transfère le JSR008 du JCP à OSGi
- ◆ 1.0 : Mai 2000 (189 pages)
- ◆ 2.0 : Octobre 2001 (288 pages)
- ◆ 3.0 : Mars 2003 (602 pages)
- ◆ 4.0: Octobre 2005 (1000 pages)
- ◆ 4.1: Juin 2007 (optimisation du core R4)

## ■ Remarque

- ◆ *Open Services Gateway Initiative est un terme obsolète*

# L'ancêtre :

## JSR-8 : Open Services Gateway (OSG)

---

- **Java Embedded Server**

- ◆ **JavaOne e-Fridge**

- **Domain : SOHO / ROBO Gateway**

- **EG**

- ◆ **Spec leader : Robert Mines (Sun)**
- ◆ **Sun Microsystems, IBM, Nortel, Alcatel, Cable and Wireless, EDF, Enron,**
- ◆ **Ericsson, Lucent, Motorola, NCI, Phillips, Sybase, Toshiba**

- **Package names**

- ◆ **javax.osg.servicespace**
- ◆ **javax.osg.remote**
- ◆ **javax.osg.service**

- **Transferred to the OSGi Alliance**

# Principales propriétés du canevas OSGi

## ■ Modularisation des applications

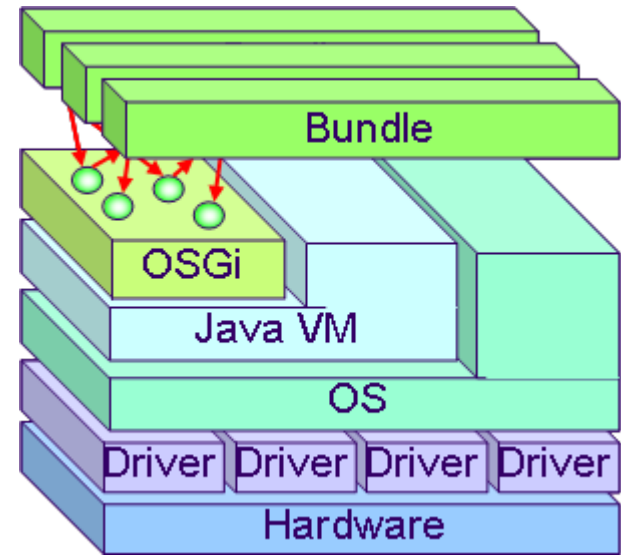
- ◆ Chargement/Déchargement de code dynamique
  - ❖ Langage Java
- ◆ Déploiement dynamique d'applications sans interruption de la plateforme
  - ❖ Installation, Lancement, Mise à jour, Arrêt, Retrait
  - ❖ « *No reboot* »
- ◆ Résolution des dépendances versionnées de code

## ■ Architecture orientée service

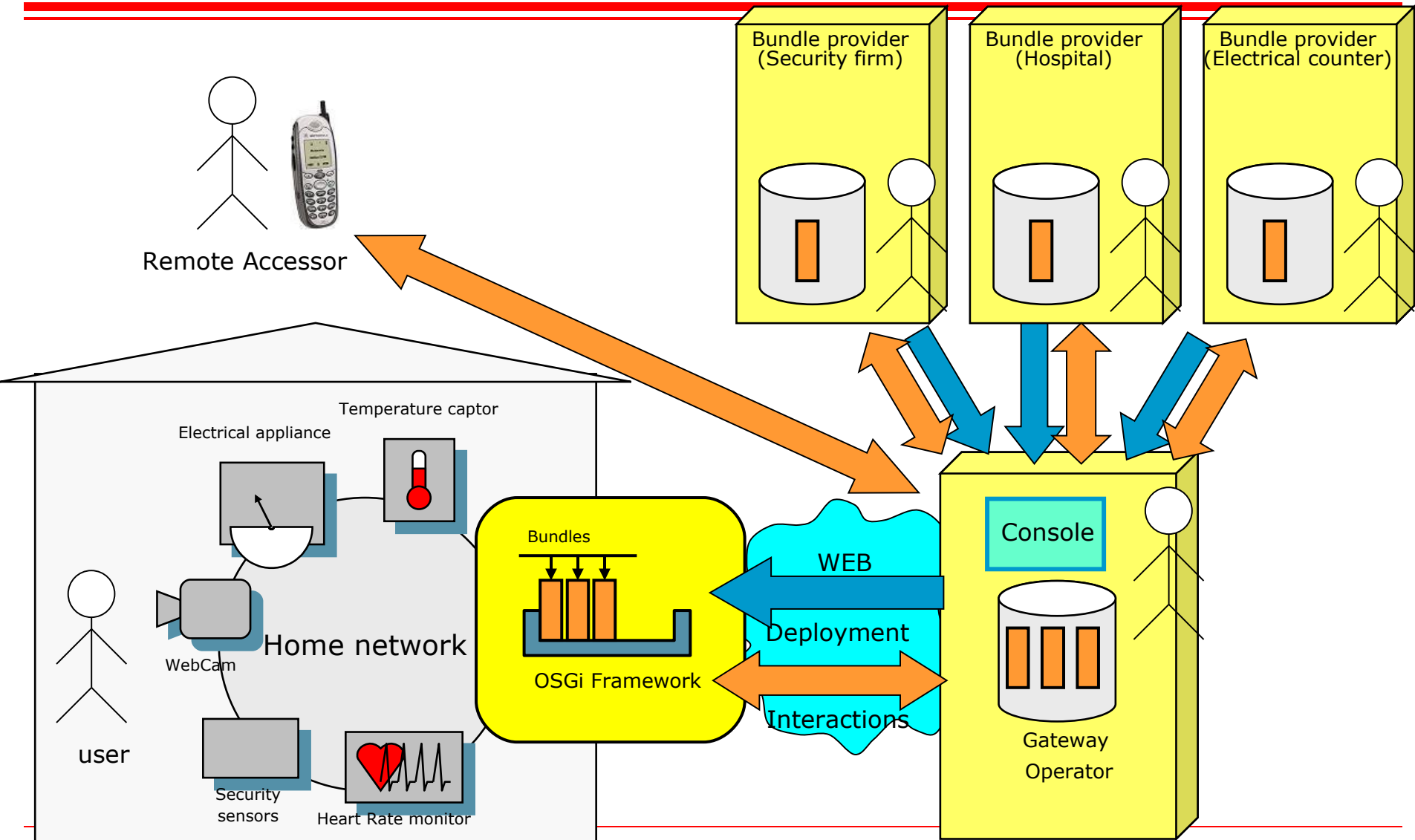
- ◆ Couplage faible, late-binding
- ◆ Reconfiguration dynamique des applications (plugins, services techniques)

## ■ Vise des systèmes à mémoire restreinte

- ◆ s'accroche à J2ME/CDC
- ◆ même si de plus en plus Java Platform 1.5, 6, 7, ...



# OSGi vue globale

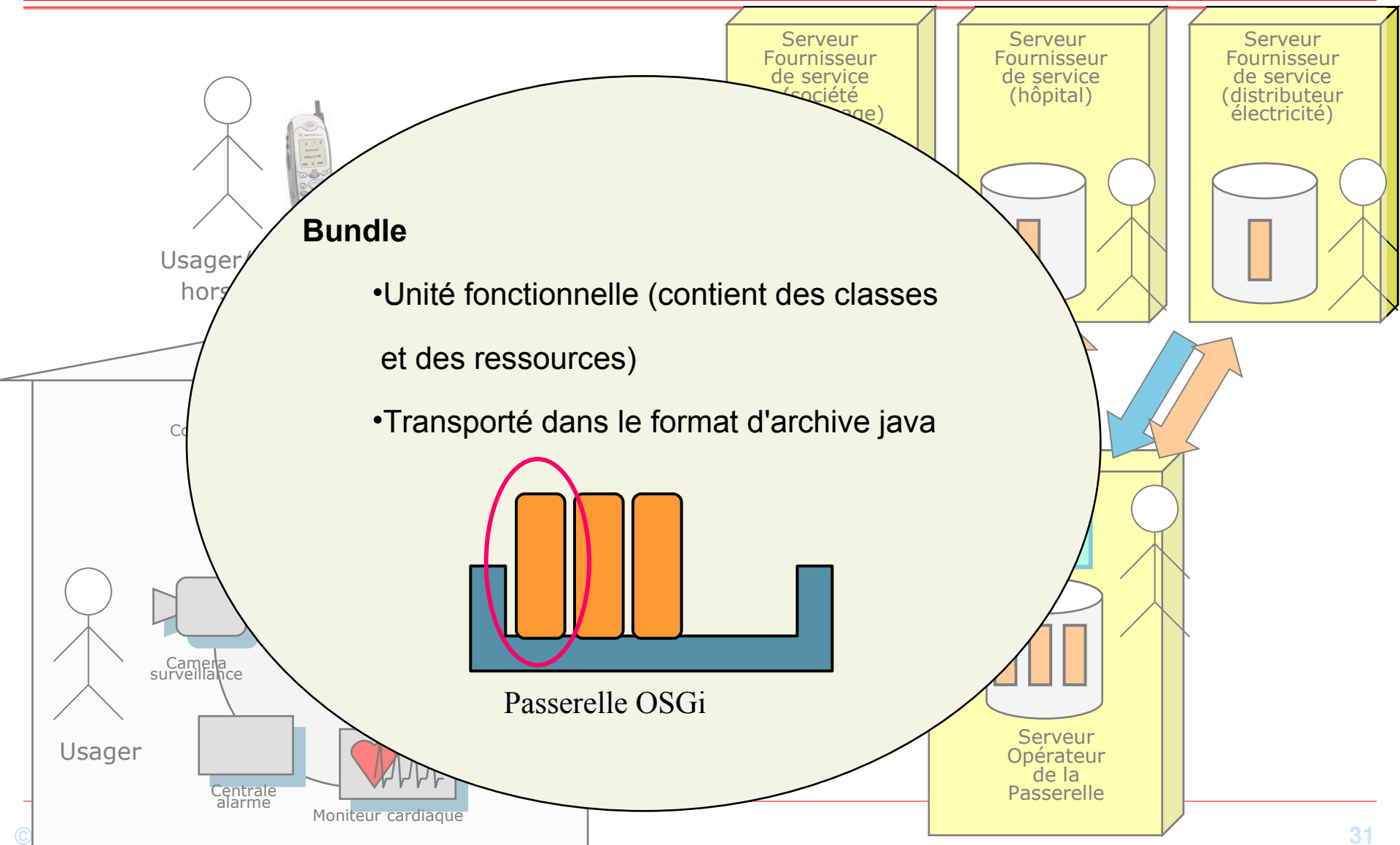


# OSGi une approche à composants

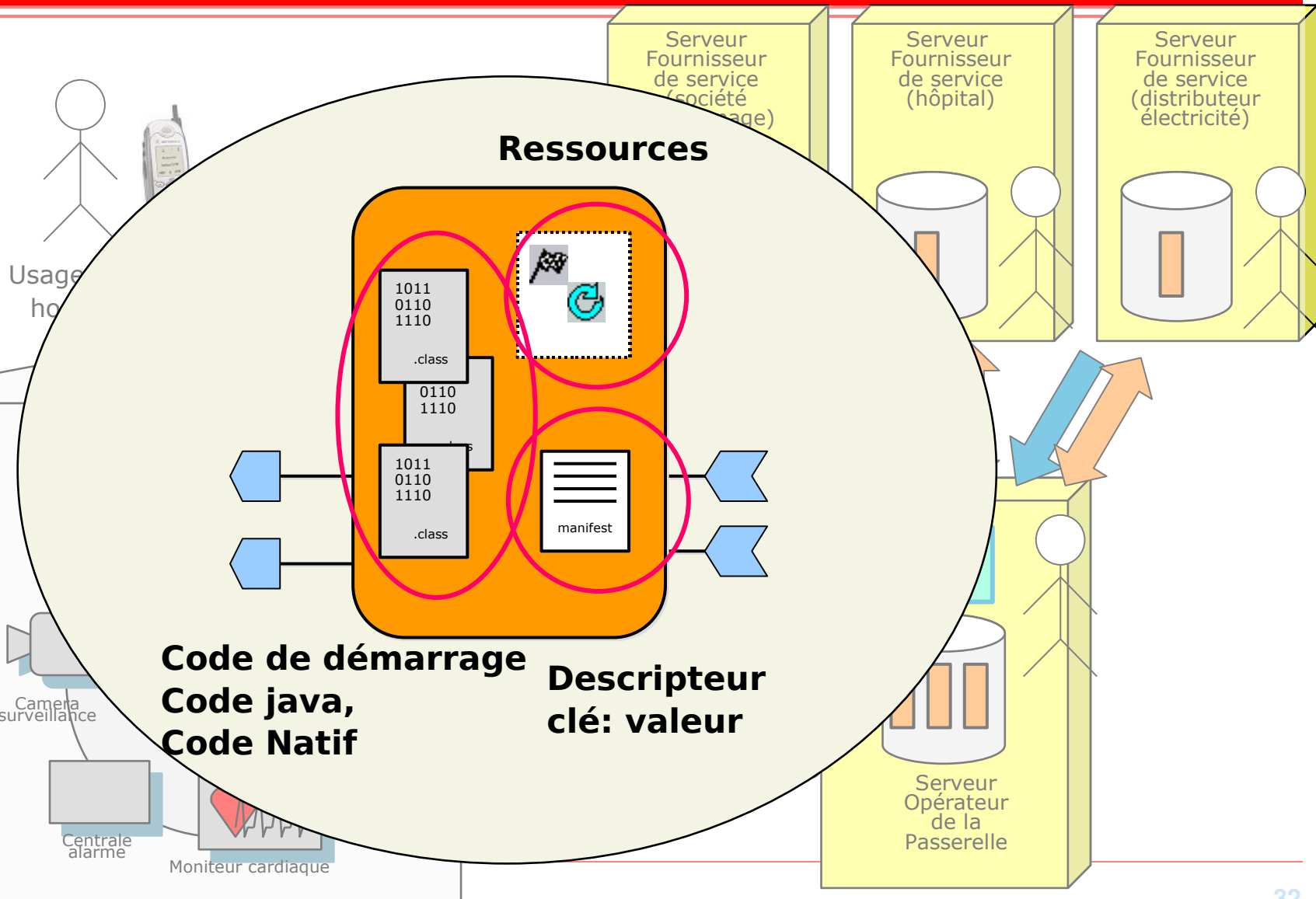
---

- Shell de commande Java
- Composants == Bundle == Fagot == Application
- Un bundle est téléchargé à partir d'une URL
- Un bundle possède un point de lancement, l'activator
  - ◆ ~~public static void main(String [] arg);~~
  - ◆ **public void start(BundleContext bc) throws BundleException;**
- Et un point d'arrêt
  - ◆ **public void stop(BundleContext bc) throws BundleException;**
- Un bundle véhicule des informations pré-lancement
  - ◆ **Dépendances sur des packages**
  - ◆ **Librairies natives**

# Le composant est un bundle



# Structure interne d'un Bundle



# Hello World

```
package hello;
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;

public class HelloWorld implements BundleActivator {
    public void start(BundleContext bc){
        System.out.println("Bonjour");
    }

    public void stop(BundleContext bc){
        System.out.println("Au revoir");
    }
}
```



```
Bundle-Name: Hello World
Bundle-Description: The simple bundle
Bundle-Activator: hello.HelloWorld
```

Le Bundle est  
"démarré"

Le Bundle est  
« arrêté »



```
200 Tue Jun 15 16:20:00 CEST 2004 META-INF/MANIFEST.MF
723 Tue Jun 15 16:20:00 CEST 2004 hello/HelloWorld.class
```

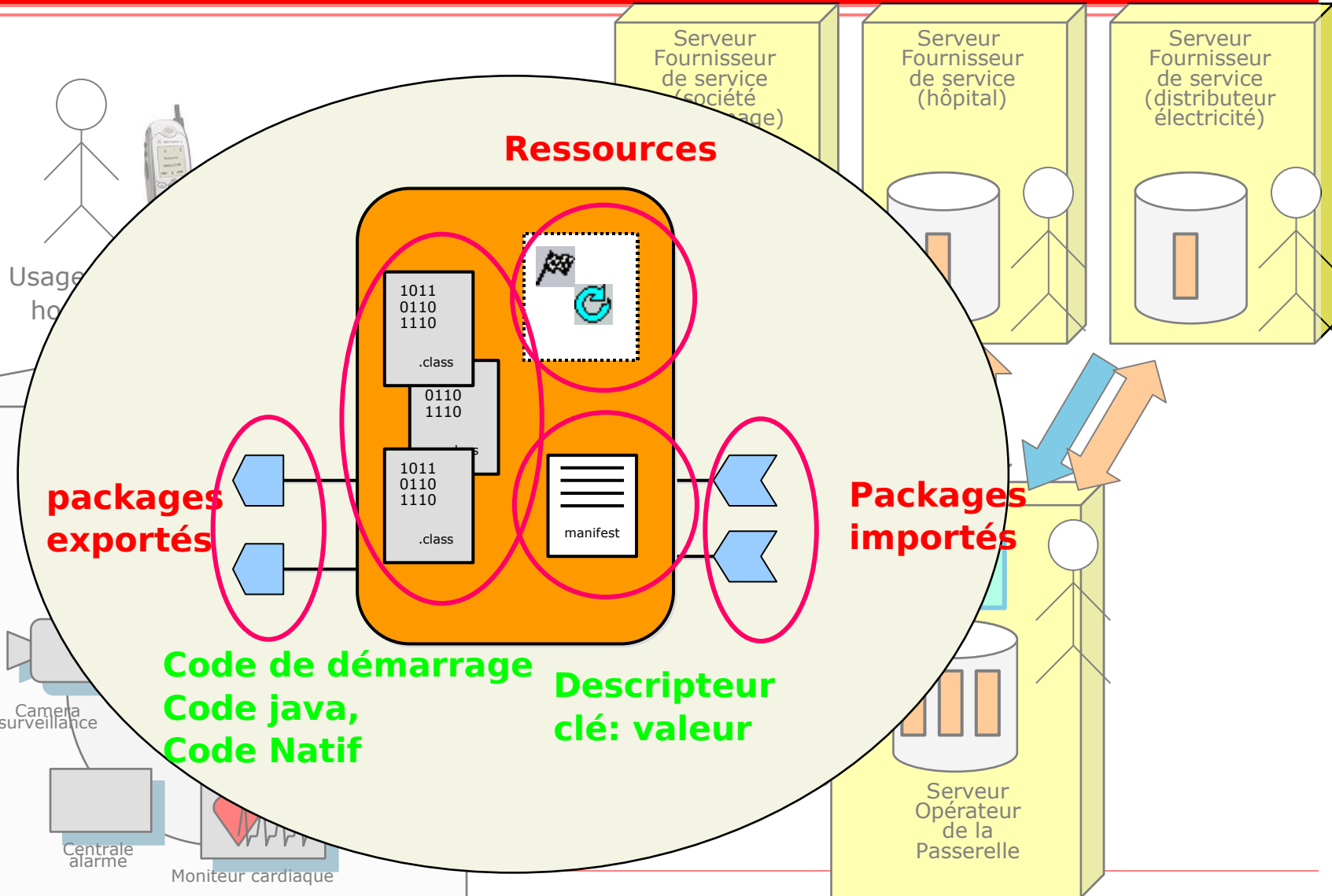
# Démarrage d'un bundle

---

- 1) Décompactage de l'archive dans un cache
- 2) Ouverture du manifest
- 3) Identification de la clé : Bundle-Activator
- 4) Création d'un chargeur de classe dédié
- 5) Instanciation explicite de la classe d'activation
- 6) Invocation de la méthode start dessus

```
BundleClassLoader myCL = new BundleClassLoaderr("file:/tmp/hello.jar");
String clazzName=myCL.getManifest().getBundleActivatorName();
Class myClass = myCl.load(clazzName");
BundleActivator act = (BundleActivator)myClass.newInstance();
act.start(myCl.getBundleContext());
...
```

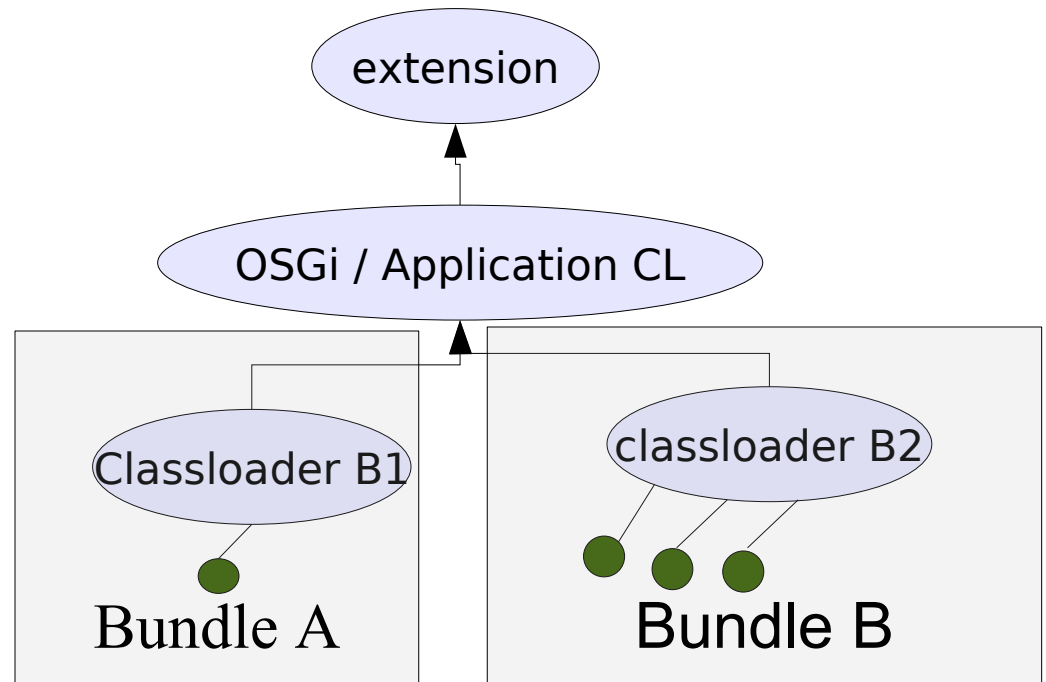
# Structure interne d'un Bundle



# Bundle : point de vue du chargeur de classes

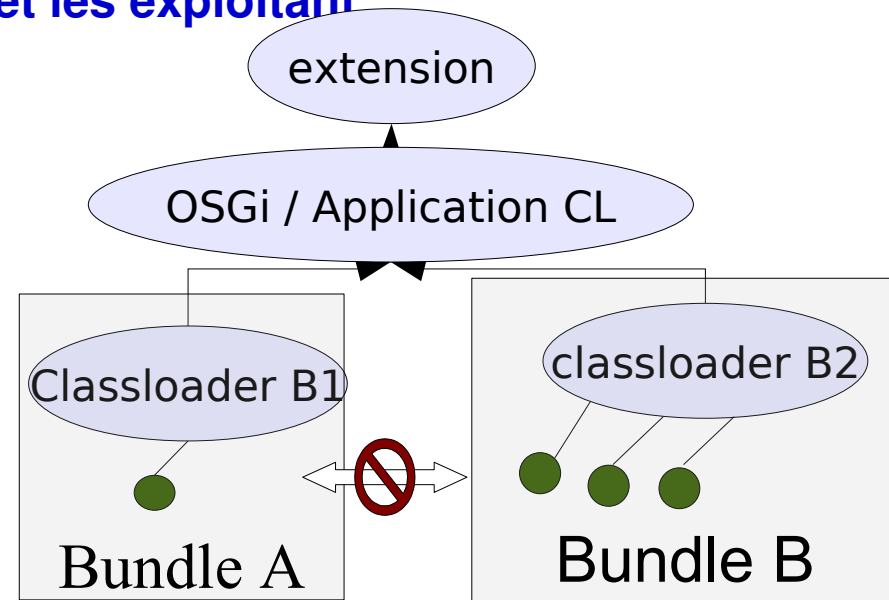
## ■ Un bundle possède un chargeur de classes dédié :

- ◆ Chaque classe et chaque ressource est isolé
- ◆ Quand le bundle est supprimé les classes peuvent être garbage collectées dès qu'il n'y a plus d'objet les exploitant



# Bundle : point de vue du chargeur de classes

- Un bundle possède un chargeur de classes dédié :
  - ◆ Chaque classe et chaque ressource est isolé
  - ◆ Quand le bundle est supprimé les classes peuvent être garbage collectées dès qu'il n'y a plus d'objet les exploitant

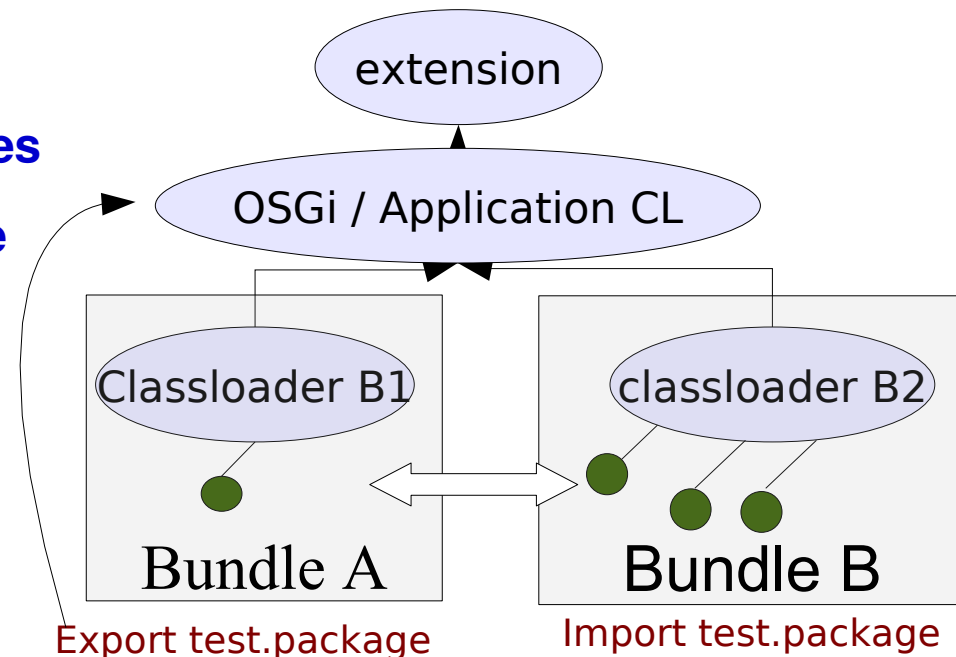


- ◆ Comment B1 utilise une classe dans B2 ? `ClassNotFoundException`

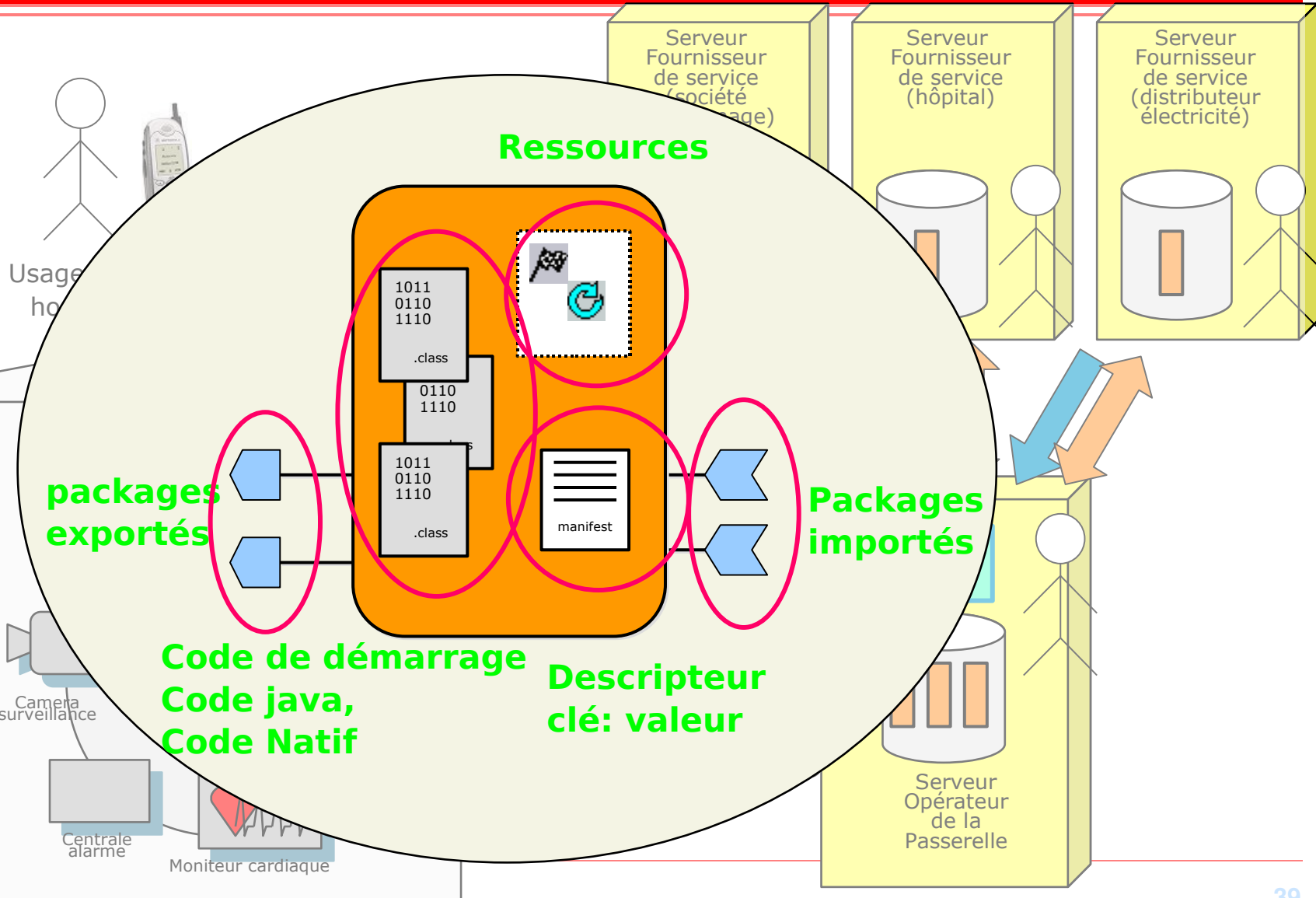
# Bundle : point de vue du chargeur de classes

## ■ Un bundle possède un chargeur de classes dédié :

- ◆ Chaque classe et chaque ressource est isolé
- ◆ Quand le bundle est supprimé les classes peuvent être garbage collectées dès qu'il n'y a plus d'objet les exploitant
- ◆ Import/Export de package
  - ❖ Permet l'échange de classes
- ◆ Chaque classe est alors résolue

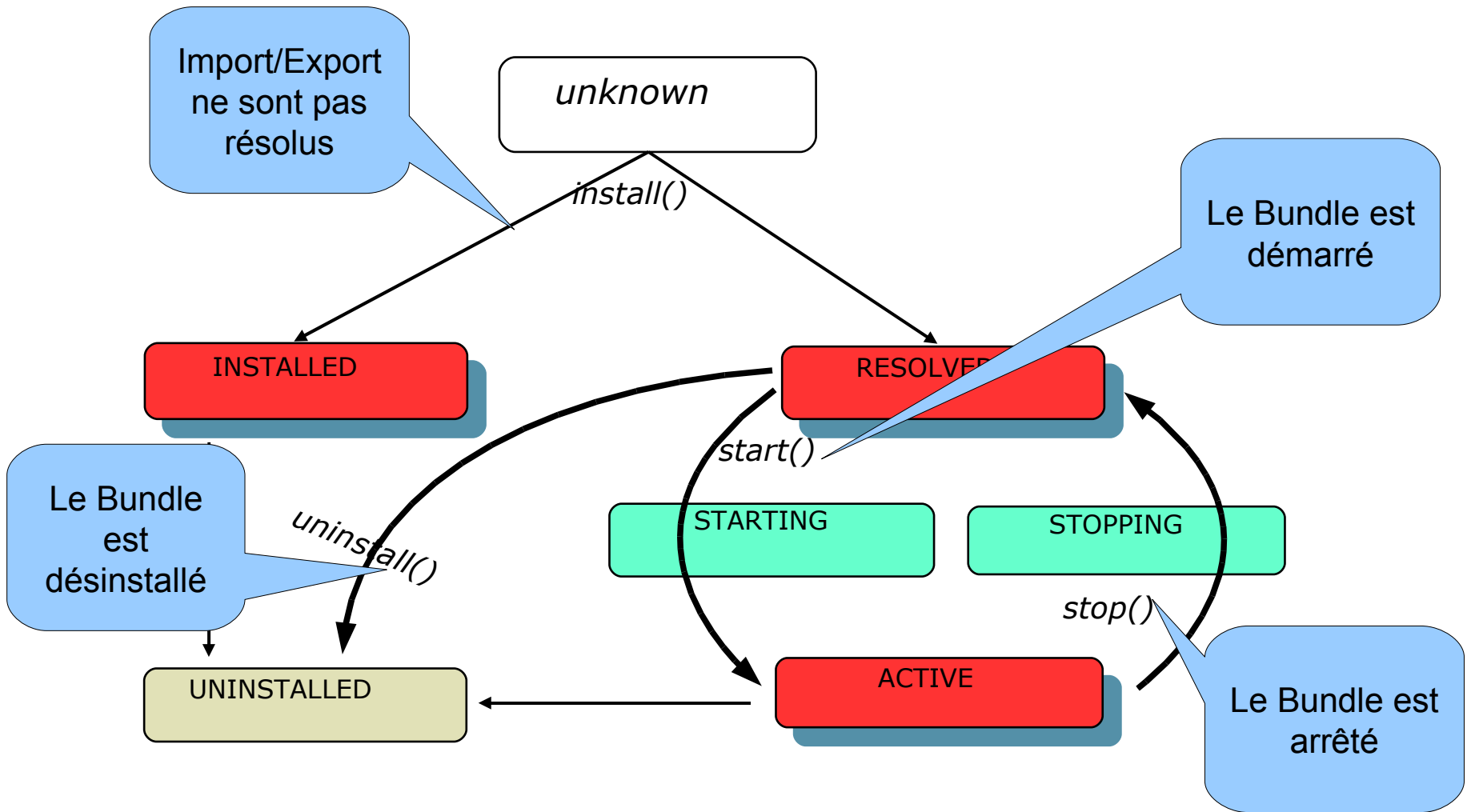


# Structure interne d'un Bundle





# Cycle de vie : Transitions



# Approche Composant par le bundle

---

- Un bundle est une unité de déploiement (pas de composition)
- Il emballe des classes, mais aussi...
  - ◆ Des ressources par exemple pour des bundles de personnalisation
  - ◆ Du code natif,

```
Bundle-NativeCode:  
    com/mycomp/impl/nativesample/libnat.so;  
    osname=Solaris; processor=sparc; osversion=5.5
```
  - ◆ Des jar tels quels comme des parsers xml ou des outils

```
Bundle-Classpath: ., ./nano.xml
```
- C'est une application autonome, qui peut utiliser les classes provenant d'autres bundles ==> Environnement multi-application pour java
- Un bundle peut ne pas avoir d'activator s'il ne fait qu'exporter des packages
- Le BundleContext est l'interface avec le framework OSGi. Tout bundle en recoit un au démarrage

# Où ca peut coincer

---

## ■ Si l'activateur bloque

- ◆ L'activateur, comme son nom l'indique doit activer l'application et rendre la main au plus vite

## ■ Si un bundle ou une bibliothèque intégrée font du chargement explicite

- ◆ Certains code font du chargement dynamique et ne sont pas forcément intégrable directement, car il cassent la politique de chargeurs

## ■ MAIS LA PROGRAMMATION ORIENTEE SERVICES DANS TOUT CA ?

# Plan

---

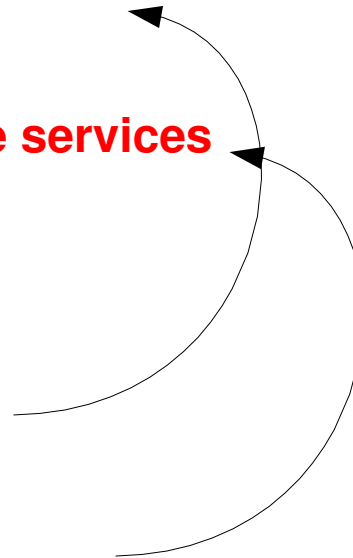
---

## ■ Fondamentaux java

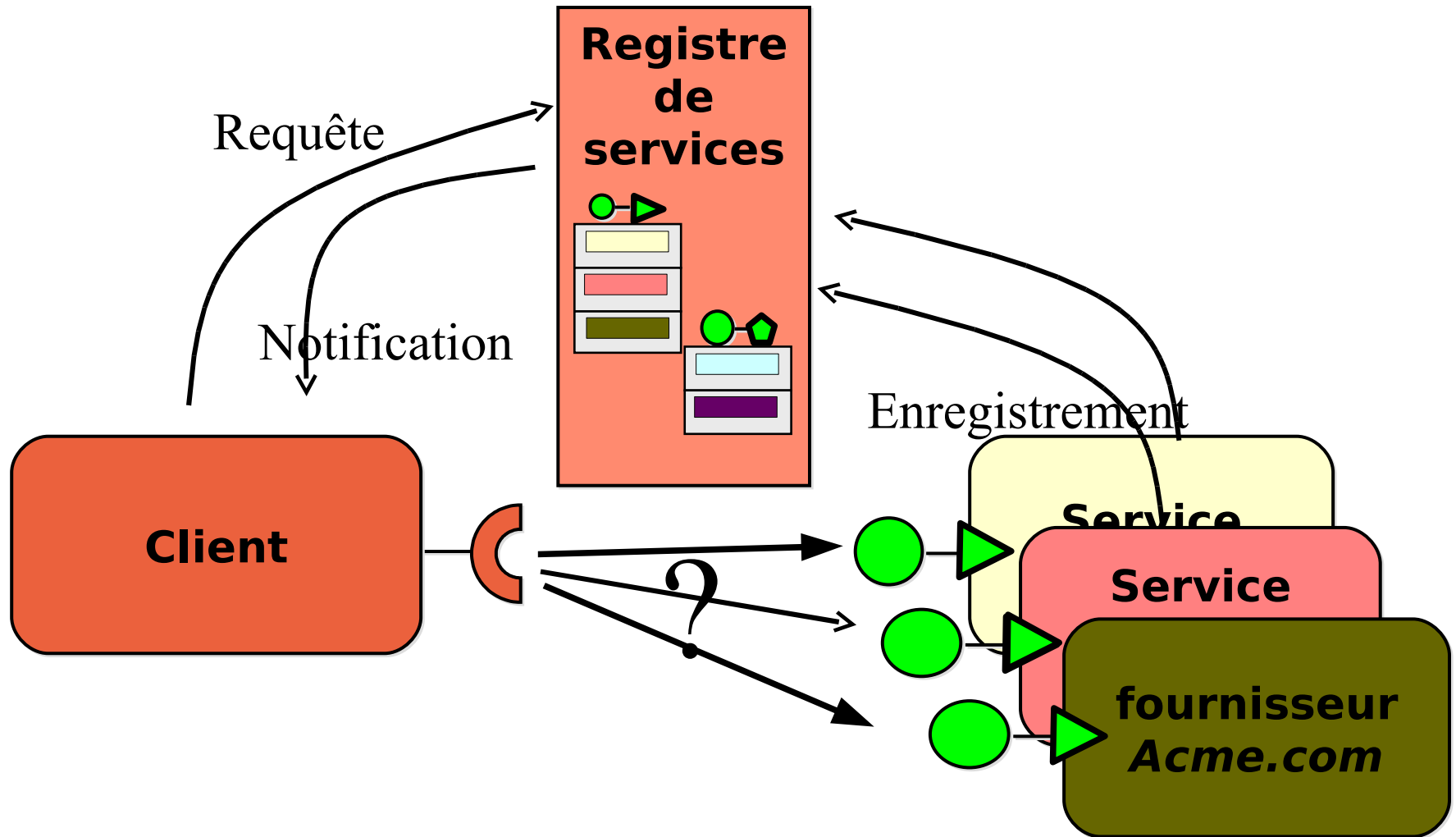
- ◆ Dynamisme
- ◆ Programmation orientée services

## ■ OSGi

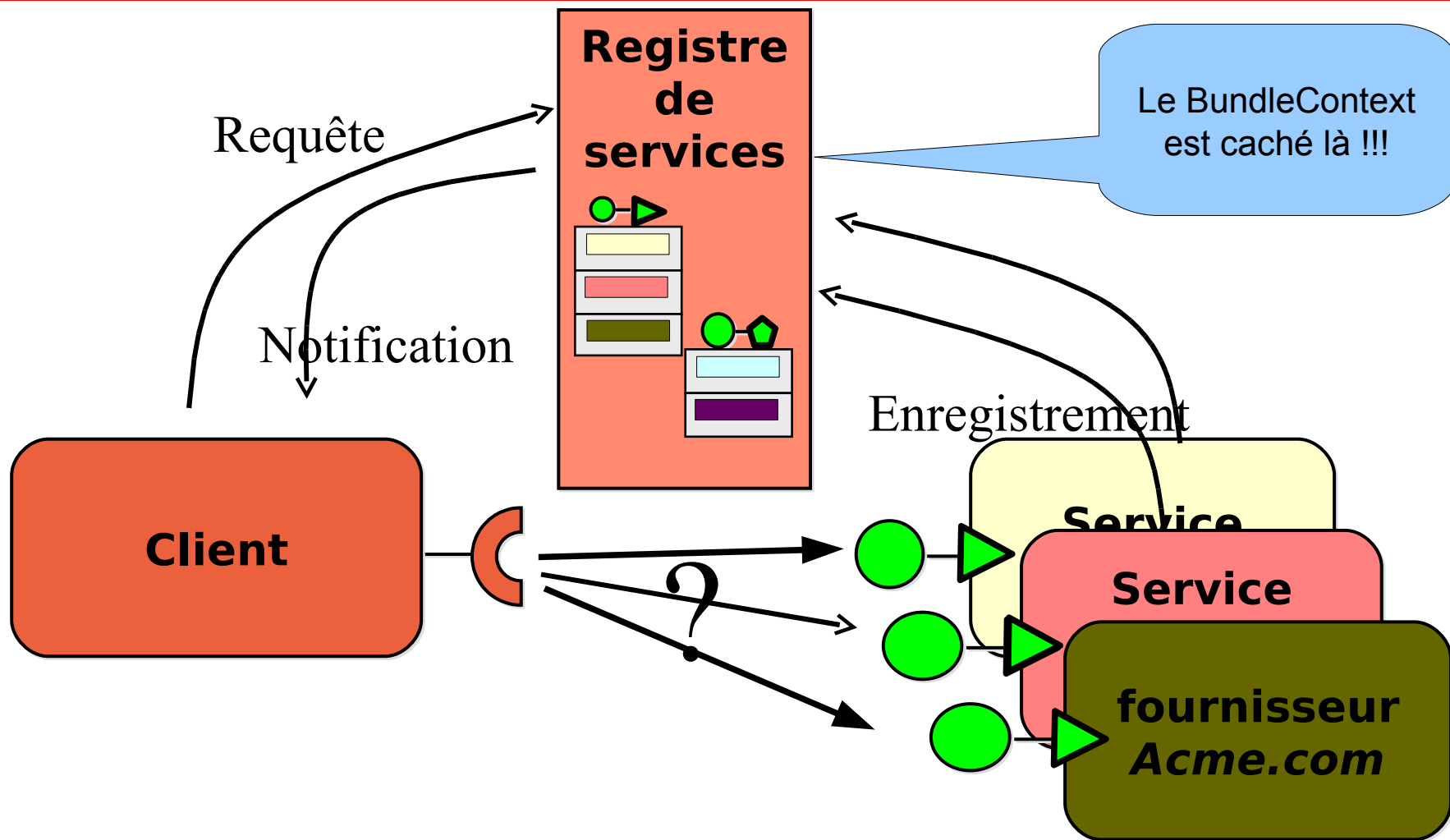
- ◆ Introduction
- ◆ Approche composants
- ◆ Approche services



# Le schéma de la POS



# Le schéma de la POS



# La programmation orientée services

---

- **Permet de ré-exploiter des interfaces standards**
- **De remettre en cause les implantations en cours d'exécution**
- **D'avoir plusieurs implantations concurrentes**
- **Couvre 2 activités**
  - ◆ **La déclaration d'un service**
  - ◆ **La récupération d'une implantation**
    - ❖ **Récupération directe**
    - ❖ **Maintient d'un pointeur par réaction événementielle**

# Exemple : le service de log

---

- Remplacement du

```
public void hello{
    System.out.println("Entrée dans la fonction");
    i++;
    System.out.println("Sortie de la fonction");
}
```

- Par un joli code

```
public void hello{
    logger.log(Log.INFO, "Entrée dans la fonction");
    i++;
    logger.log(Log.INFO, "Sortie de la fonction");
}
```

- Pas de commentaires de code avant livraison, comportement idempotent

- Plusieurs implantations possibles du service

  - ◆ Production et Consommation d'entrées de log non liées

# Exemple : le service de log

---

## ■ Une Interface de service

```
package logsystem;  
public interface Log{  
    public void log(String level, String message);  
}
```

## ■ Pleins d'implantation possibles...

```
package frenot;  
public LeLog implements logsystem.Log{  
    public void log(String level, String message){  
        System.out.println(message);  
    }  
}
```

# Exemple : un Bundle de service de Log

```
package logsystem;  
public interface Log{  
    public void log(String level,  
                    String message);  
}
```



```
Bundle-Name: logger  
Bundle-Description: un logger  
Bundle-Activator: frenot.LeLog  
Import-Package: org.osgi.framework  
Export-Package: logsystem
```

```
import org.osgi.framework.BundleActivator;  
import org.osgi.framework.BundleContext;  
import logsystem.Log;  
  
package frenot;  
public LeLog implements Log, BundleActivator{  
    public void log(String level, String message){  
        System.out.println(message);  
    }  
    public void start(BundleContext bc){  
        bc.registerService("logsystem.Log", this, null);  
    }  
    public void stop(BundleContext bc){  
    }  
}
```



```
200 Tue Jun 15 16:20:00 CEST 2099 META-INF/MANIFEST.MF  
723 Tue Jun 15 16:20:00 CEST 2099 logsystem/Log.class  
825 Tue Jun 15 16:20:00 CEST 2099 frenot/LeLog.class
```

# Exemple : un autre Bundle de service de Log

Bundle-Name: logger  
Bundle-Description: un logger  
Bundle-Activator: frenot.LeLog  
Import-Package: **logsystem**,  
org.osgi.framework



```
import org.osgi.framework.BundleActivator;  
import org.osgi.framework.BundleContext;  
import logsystem.Log;  
  
package superlog;  
public SuperLog implements Log, BundleActivator{  
    public void log(String level, String message){  
        System.out.println(level+ " : "+message");  
    }  
    public void start(BundleContext bc){  
        bc.registerService(logsystem.Log.class.getName(),  
            this, props);  
    }  
    public void stop(BundleContext bc){  
    }  
}
```

Le package du log est  
fourni par un autre bundle

Des propriétés  
distinguent les deux  
implantations

Une implantation radicalement  
différente



200 Tue Jun 15 16:20:00 CEST 2099 META-INF/MANIFEST.MF  
825 Tue Jun 15 16:20:00 CEST 2099 superlog/SuperLog.class

# Un service est fourni par

---

- 1 bundle,
  - 2 bundles en séparant l'interface et l'implantation,
  - N bundles 1 pour chaque implantation
- 
- Il y a indépendance complète entre interface de service, implantations de services et bundles
- 
- Il ne reste plus qu'à utiliser le service....
    - ◆ Et discuter

# Un service est fourni par

---

- 1 bundle,
  - 2 bundles en séparant l'interface et l'implantation,
  - N bundles 1 pour chaque implantation
- 
- Il y a indépendance complète entre interface de service, implantations de services et bundles
- 
- Il ne reste plus qu'à utiliser le service....
    - ◆ Et discuter

# Un service est fourni par

## ■ Appel direct :

```
ServiceReference sr=bc.getServiceReference("logsystem.Log");  
Log lelog=(Log)bc.getService(sr);  
lelog.log("DEBUG", "it works !");
```

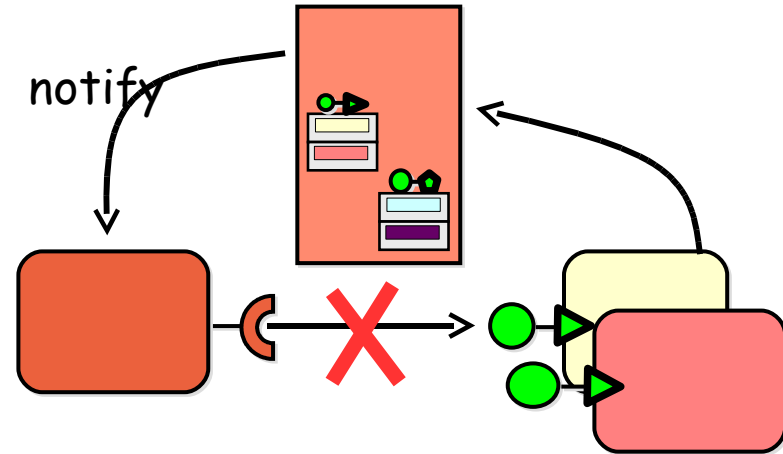
## ■ Programmation réactive

```
bc.addServiceListener(this, "(ObjectClass=javax.log.LogInterface)");  
public void serviceChanged(ServiceEvent serviceevent) {  
    ServiceReference servicereference= serviceevent.getServiceReference();  
    switch (serviceevent.getType()) {  
        case ServiceEvent.REGISTERED :  
            this.log=(Log) serviceevent.getService();  
            break;  
        case ServiceEvent.UNREGISTERING :...
```

# Simplification du modèle de programmation événementiel

## ■ Extensions à OSGi

- ◆ Service Tracker
- ◆ Gravity
- ◆ Declarative services
- ◆ IPOJO



## ■ Alternatives à la programmation orientée services

- ◆ Avalon
- ◆ Pico|Nano containers
- ◆ Metro
- ◆ Spring
- ◆ Google Juice

# Un dernier point sur la sécurité

---

## ■ Deux aspects :

- ◆ Une archive peu contenir des fichiers de signature. Une signature incorrecte nécessite de rejeter l'archive
- ◆ Une classe ne doit pas exécuté une méthode spécifique
  - ❖ **SBAC : Stack Based Access Control**

```
m1.test();
```

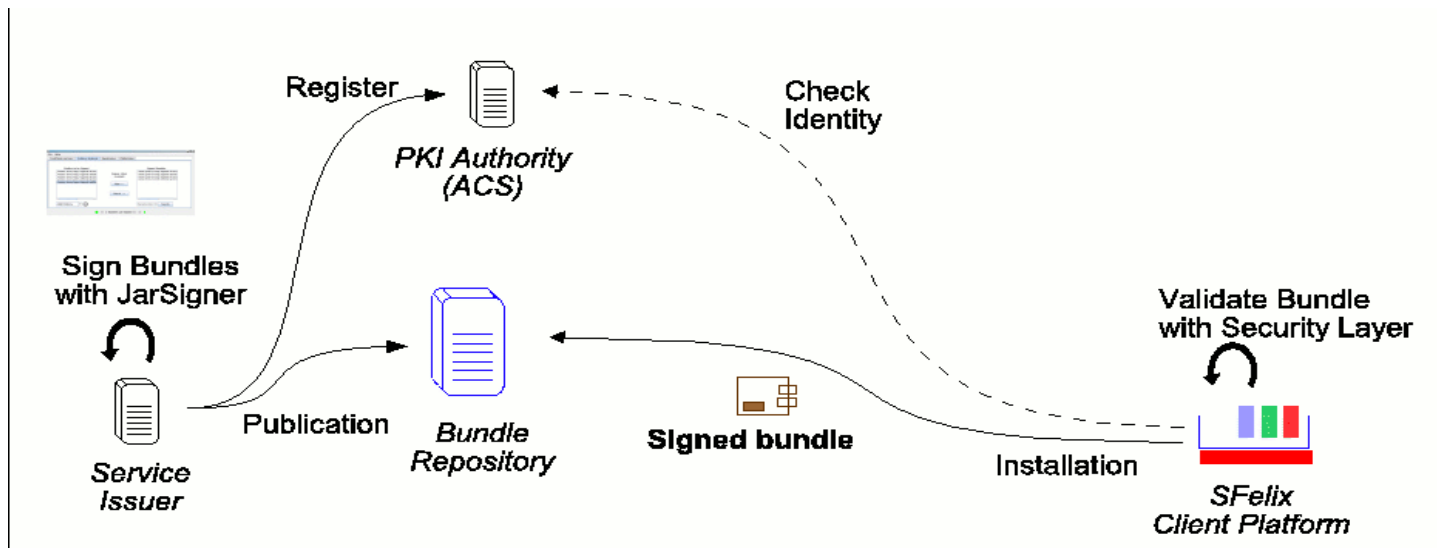
```
System.halt(); --> Le gestionnaire de sécurité contrôle un fichier  
de politique pour savoir si l'appel est autorisé ou non
```

## ■ OSGi améliore les deux aspects

- ◆ Les bundles sont contraints par des fichiers de signature
- ◆ Certaines méthodes de gestion sont assujeties au contrôle du gestionnaire de sécurité

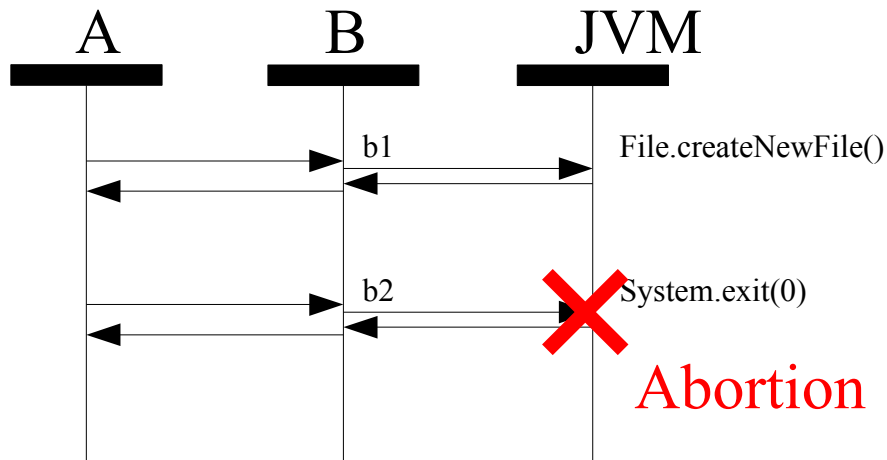
# Fichier de signature

- Signer Pierre generates a public/private key pair
- Pierre registers its public key on a Certification Authority
  - ◆ And sends its certificate to the client Platform
- Bundle resources are signed with the private key
- Verification is performed with the public key at the client platform



# Appel de méthodes

- Stack-based Access Control (SBAC)
  - ◆ Class A is in the Protection Domain for signer Alice
    - ❖ Bundle for A is signed by Alice
  - ◆ Class B is in the Protection Domain for signer Bob



MAIS....

```
Grant signer Alice{
    permission java.io.FilePermission "read",write";
}
Grant signer Bob{
    permission java.io.FilePermission "read",write";
    permission java.lang.RuntimePermission "exitVM";
}
```

# Sécurité inutilisée pour l'instant

---

## ■ Fondamentalement : Mono-fournisseur

## ■ Pour les fichiers de signature

- ◆ Processus lourd et diffusion complexe
- ◆ Protocoles de crypto souvent incompatible avec une approche open-source
- ◆ Un bundle non sécurisé et non installé sont perçus de la même manière

## ■ Pour le contrôle d'exécution de fonction

- ◆ 30% de surcoût
- ◆ Manque d'adaptabilité et de configurabilité

# Pourquoi la pile java/OSGi est importante

---

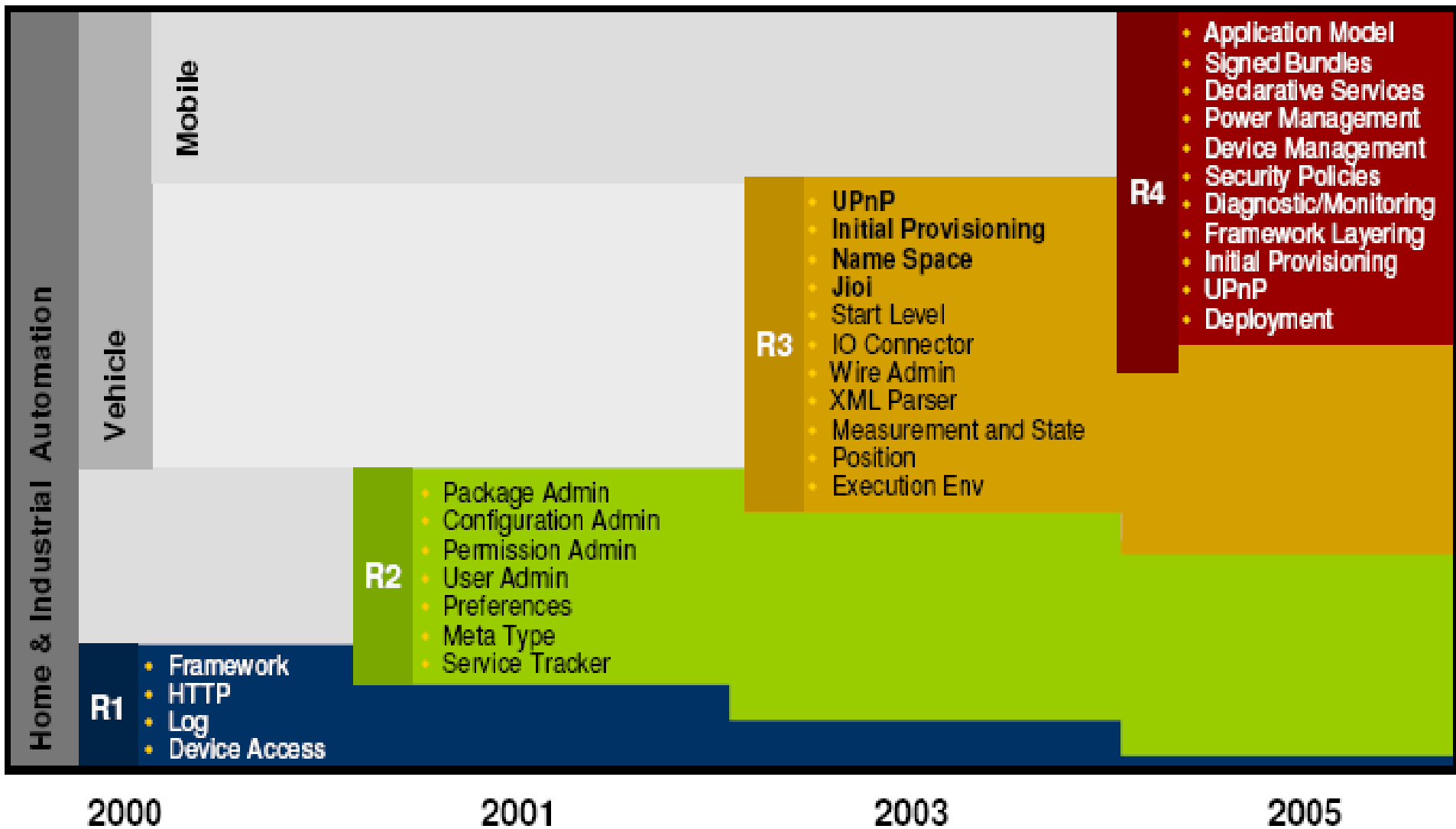
## ■ Pourquoi Java ?

- ◆ **Orienté-Objet**
- ◆ **Bibliothèque standardisée volumineuse**
- ◆ **Dynamique**
  - ❖ **Les classes sont ajoutées et retirées dynamiquement**
- ◆ **Sécurisé**
  - ❖ **Langage typé**
  - ❖ **Gestionnaire de sécurité**

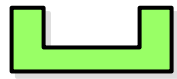
## ■ Pourquoi OSGi

- ◆ **Orienté composant**
- ◆ **Comportement de programmation orientée services**
  - ❖ **Simplifie l'intégration des aspects dynamiques de Java**
- ◆ **Définit des interfaces de services standard**

# Carte de spécification des services standards



# Concepts principaux d'OSGi



## ■ Framework:

- Environnement d'exécution de Bundles
  - Oscar (Objectweb) / Felix (Apache), SMF, Knopperfish, Equinox
- Notification événementielle



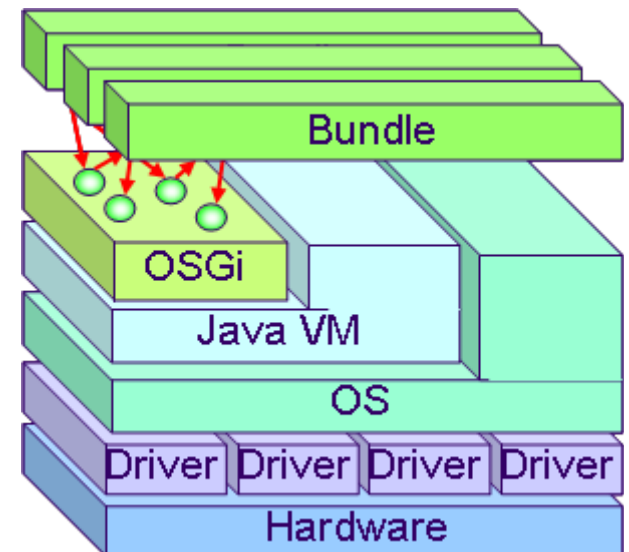
## ■ Bundles:

- Unité de diffusion de ressources



## ■ Services:

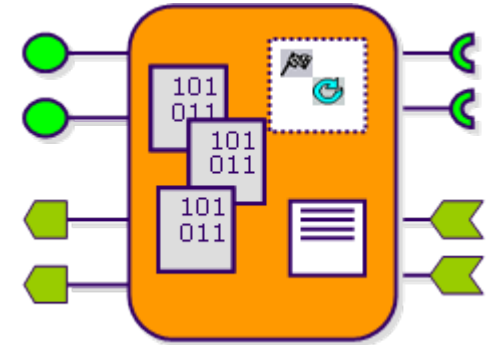
- Objet Java qui implante un contrat spécifique



# Middleware et Packaging d'applications

## ■ Modularisation du middleware et des applications

- ◆ Distribue les services du middleware
- ◆ Meilleure visibilité des composants
- ◆ Conteneur de déploiement
- ◆ Mise à jour partielle sans redémarrage



## ■ Implantation

- ◆ Basé sur des jar et des entrées manifest
- ◆ Dépendance explicites de packages et de versions

## ■ Fondamentaux pour la prochaine génération de standards

- ◆ JSR 277, Java Module System
- ◆ JNLP(JSR-56), J2EE EAR, OSGi R3 bundle
- ◆ J2SE 1.7 (2007)

# Vice et vertues



## Vertues

- ◆ Léger
- ◆ Packaging propre
  - ❖ Masque les problèmes du classpath
- ◆ Dynamique à l'exécution
- ◆ Exécution non interrompue



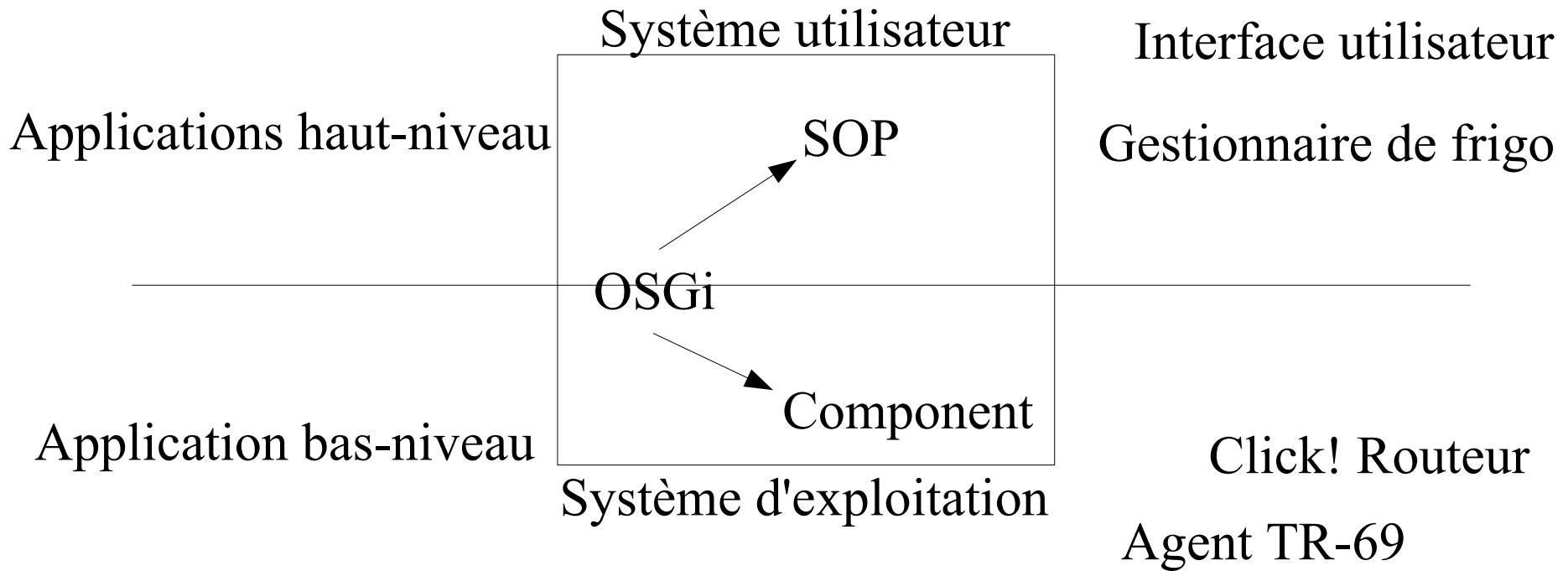
## Vices

- Approche programmatique
  - Programmation événementielle est pénible
- Pas de service non-fonctionnel
- Approche d'exécution locale

# OSGi dans un environnement domestique

---

---



# OSGi Java et Unix

---

- **Java language**
- **Automatic memory management**
- **Component convergence**
- **Service oriented programming**
- **Mono-user**
- **Moderately efficient**
- **Heavy weight**
- **Simply dynamic**
- **C language**
- **Manual memory management**
- **Packaging heterogeneity**
- **ipc/socket programming**
- **Multi-user**
- **Very efficient**
- **Light weight**
- **Eventually dynamic**

# YOUR WORST FEARS DEP'T

• NUMBER 17 •

Look, Mommy... My Sally Shopper doll's  
Java Embedded Server is automatically  
ordering her new Spring wardrobe.

BZZZZZZ  
WHEEEEE  
CLICK

Phil Frank

# Un mariage de raison : maven

---

---

- **Ce que le bundle OSGi est au run-time, l'artéfact Maven l'est pour le déploiement logiciel**
- **Intégration dans une vision composant et cycle de vie du développement logiciel**
- **Partant d'archétype de projets, pour aboutir à la génération d'artéfacts qui sont des composants logiciels déployés**
- **Remplacement majeur d'ant**

# Maven des constats

---

## ■ Les développement autour de Java sont tous similaires

- ◆ Structure de développement (src, classes, dist...)
- ◆ Gestion complexe du classpath avec des jar
- ◆ Packaging final sous la forme d'archives
- ◆ Difficultés liées aux versions spécifique et au suivi large échelle

## ■ Structure de projet

- ◆ Archétype : ejb, osgi, renault, spring...

## ■ Composants logiciels

- ◆ Artifacts : jfreechart.jar, shell.jar, osgi-bundle-packager.jar

# Similarités/Différences par rapport à OSGi

---

- Conteneur de bundle

- Bundle

- Cache

- Manifest de description

- Dépôt de bundles (obr)

- Services standards

- Apparaît à l'installation

- Focalisé sur l'exécution

- Conteneur d'artéfacts

- Artéfact

- .m2

- pom.xml

- Dépôt d'artéfacts

- Fonctions additionnelle

- Avant l'installation

- Focalisé sur le déploiement et la compilation

# Maven la chaîne

---

---

mvn install

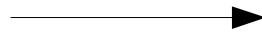


validate  
generate-source  
process-source  
generate-resources  
process-resources  
compile  
process-classes  
generate-test-sources  
process-test-sources  
generate-test-resources  
process-test-resources  
test-compile  
test  
package  
pre-integration-test  
integration-test  
post-integration-test  
verify  
install  
deploy

# Maven OSGi archetype

---

```
mvn archetype:create \  
-DgroupId=icar.2008 \  
-DartifactId=osgiprojet
```



**src/main/java**

**src/main/resources**

**src/test/java**

**src/test/resources**

**src/site**

*target*

**pom.xml**

# Project Object Model : pom.xml

---

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <packaging>bundle</packaging>
  <name>Apache Felix MOSGi JMX agent</name>
  <groupId>org.apache.felix</groupId>
  <artifactId>org.apache.felix.mosgi.jmx.agent</artifactId>
  <version>0.9.0-SNAPSHOT</version>
  <dependencies>
    <dependency>
      <groupId>${pom.groupId}</groupId>
      <artifactId>org.osgi.core</artifactId>
      <version>1.1.0-SNAPSHOT</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>
  <build>
    ...
  </build>
</project>
```

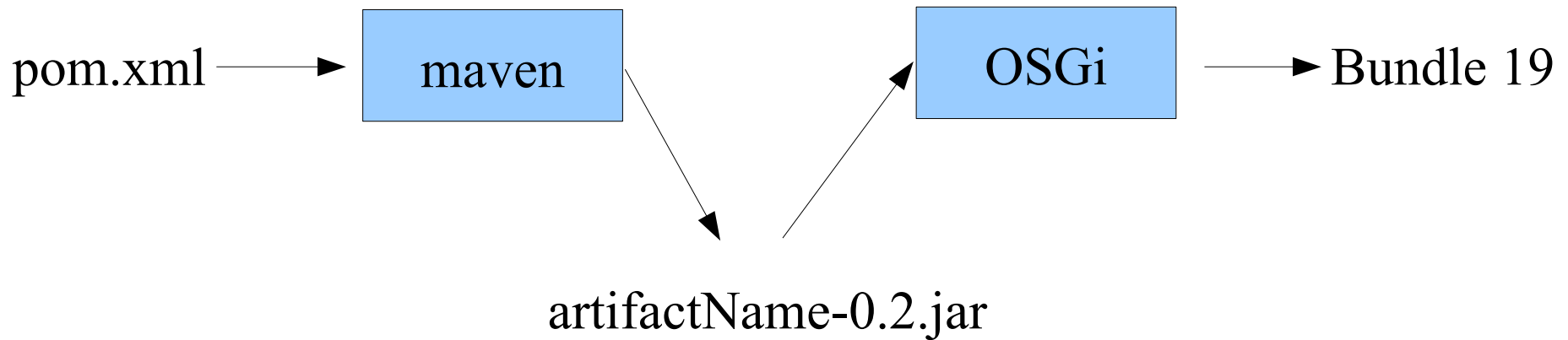
# Project Object Model : pom.xml

```
...
<plugins>
  <plugin>
    <groupId>org.apache.felix</groupId>
    <artifactId>maven-bundle-plugin</artifactId>
    <version>1.4.0</version>
    <extensions>>true</extensions>
    <configuration>
      <instructions>
        <Bundle-Name>un nom</Bundle-Name>
        <Bundle-Activator>jmx.agent.AgentActivator</Bundle-Activator>
        <Export-Package>jmx.agent.mx4j.util</Export-Package>
        <Private-Package>${pom.artifactId}.*</Private-Package>
      </instructions>
    </configuration>
  </plugin>
</plugins>
</build>
```

# Le mariage maven / OSGi

---

---



# Bibliographie & Webographie

---

## ■ Spécification

- ◆ OSGi Alliance, « OSGi service gateway specification », <http://www.osgi.org>

## ■ Ouvrage (1 seul pour le moment, un peu ancien)

- ◆ Kirk Chen, Li Gong, « Programming Open Service Gateways with Java Embedded Server Technology », Pub. Addison Wesley, August 2001  
ISBN#: 0201711028. 480 pages

## ■ Articles

- ◆ Li Gong, « A Software Architecture for Open Service Gateways », IEEE Internet Computing, January/February 2001 (Vol. 5, No. 1), pp. 64-70
- ◆ Dave Marples, Peter Kriens, The Open Services Gateway Initiative, an Introductory Overview, IEEE Communications Magazine, December 2001
- ◆ puis plein d'autres

## ■ Blogs

- ◆ Peter Kriens, *l'évangéliste OSGi*, <http://www.osgi.org/blog/>
- ◆ Java modularity, JSR 277, JSR 291, JSR 294, OSGi, open source, and software design, <http://underlap.blogspot.com/>

# Bibliographie & Webographie

---

## ■ Framework open source

- ◆ Oscar, Felix, Equinox, Knopperfish

## ■ Index de bundles

- ◆ <http://bundles.osgi.org/browse.php>

## ■ Exhibitions

- ◆ <http://www.osgiworldcongress.com/>

## ■ Complément de cours

- ◆ Donsez, Hall, Cervantes <http://www-adele.imag.fr/users/Didier.Donsez/cours/osgi.pdf>
- ◆ Frénot <http://citi.insa-lyon.fr/~sfrenot/cours/OSGi/>
- ◆ INTech <http://rev.inrialpes.fr/intech/Registration?op=511&meeting=27>

## ■ Plus

- ◆ <http://france.osgiusers.org/Main/Documentation>