

Composants adaptables au dessus d'OSGi

ABDELMADJID KETFI
DIDIER DONSEZ
HUMBERTO CERVANTES
RICHARD S. HALL



Sommaire

- ◆ Présentation OSGi
 - ◆ Avantages, limitations
- ◆ OSGi comme une couche de base pour un modèle à composants
 - ◆ Beanome
- ◆ Adaptation dynamique des composants
- ◆ Travaux actuels et conclusions

OSGi

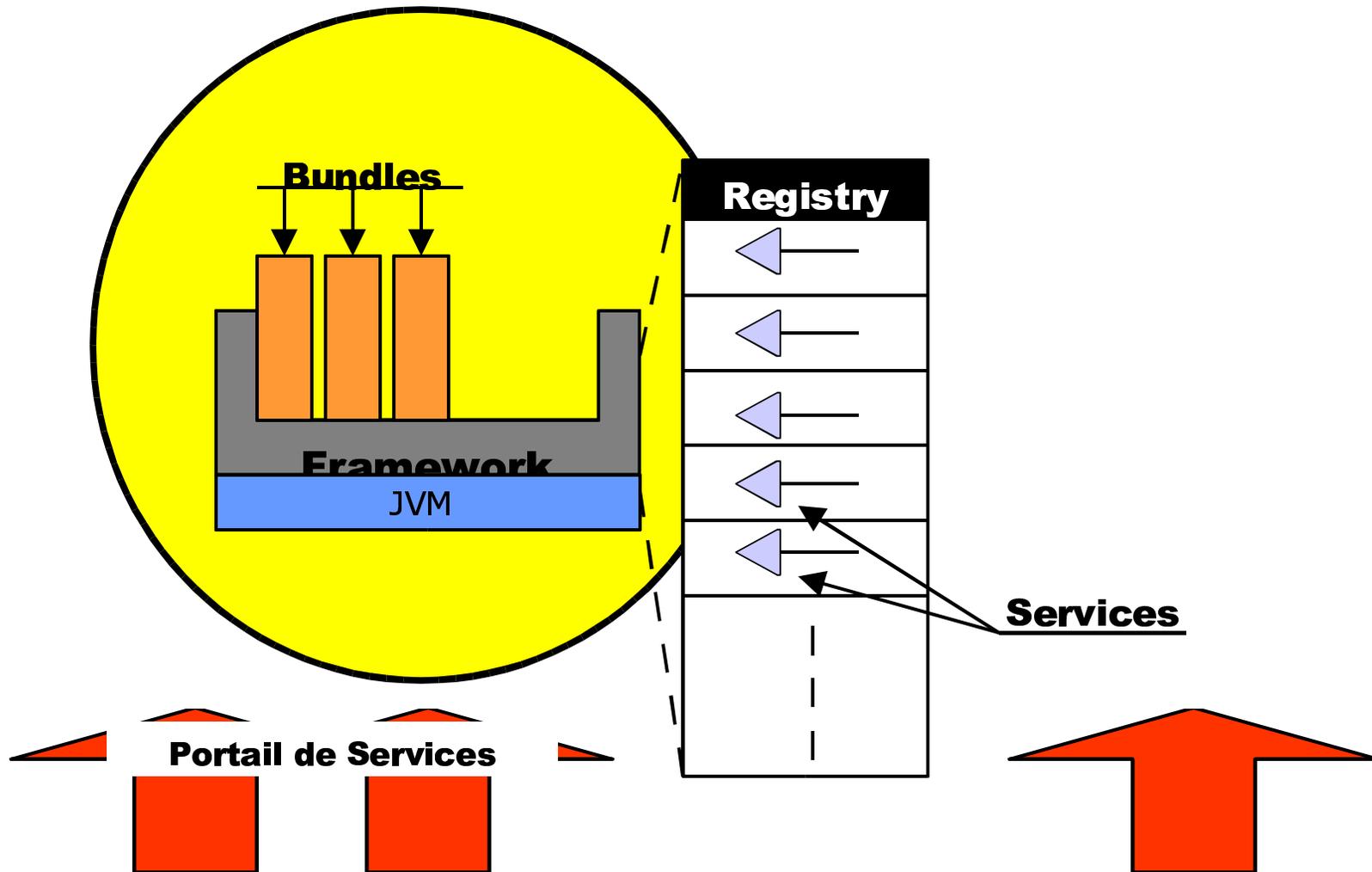


OSGi

- ◆ Open Services Gateway Initiative
 - ◆ Fondée en 1999
 - ◆ Soutien de plus de 70 compagnies
 - ◆ Objectif: définir un ensemble d'APIs JAVA qui définissent l'architecture d'un portail de services.
- ◆ Spécification OSGi:
 - ◆ 1ère livraison: Mai 2000
 - ◆ 2ème livraison: Octobre 2001
- ◆ OSCAR
 - ◆ Implémentation open source par Richard S. Hall



Vision d'ensemble



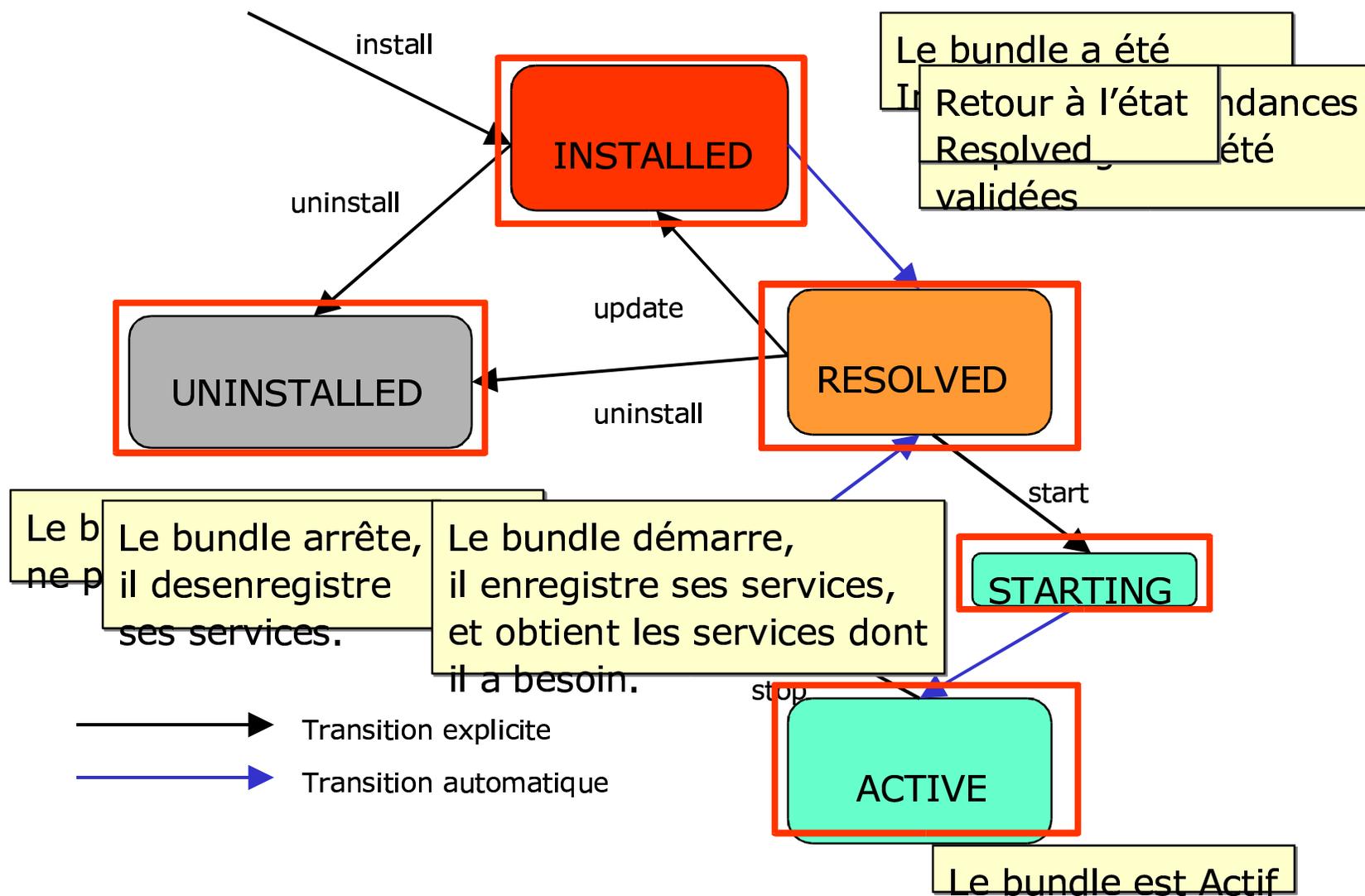


Services, Bundles, Framework

- ◆ Service: Une interface Java
 - ◆ ensemble de propriétés attribut valeur
- ◆ Bundle: Unité de livraison et déploiement des services
 - ◆ Dépendances de package et service
- ◆ Framework : Accueille les bundles, fournit un registre



Cycle de vie d'un bundle

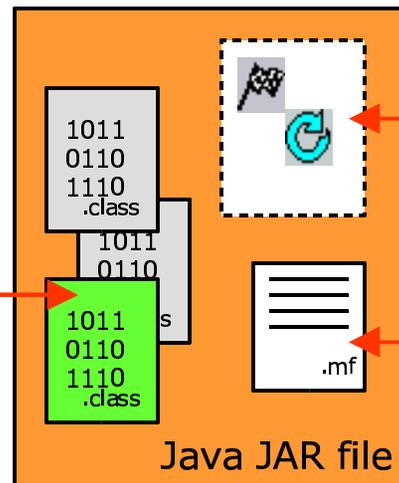




Bundle

◆ Fichier JAR

- Interfaces des Services
- Implementations
- un Activator



- Ressources

- Fichier Manifeste
(Méta-données)

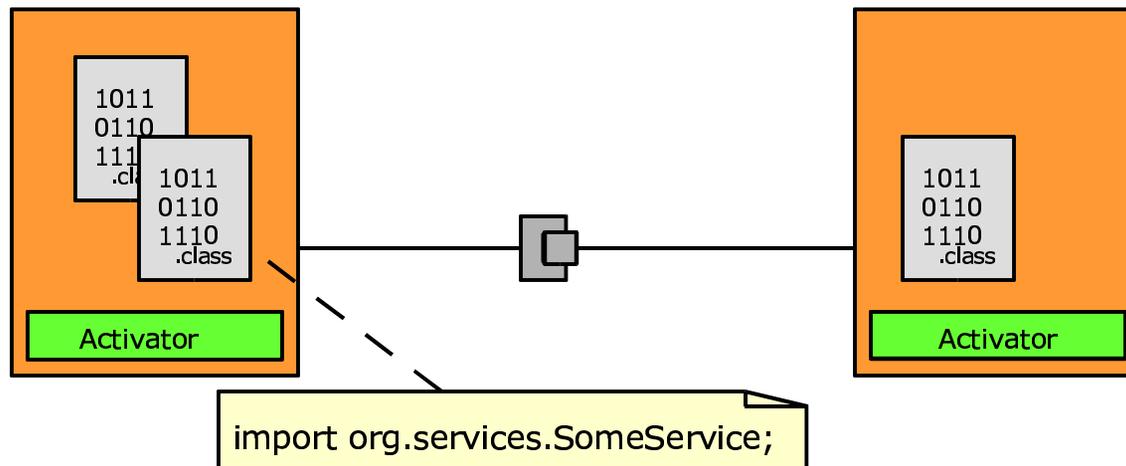


Dépendances de Package

- ◆ Le code à l'intérieur d'un bundle importe du code d'un package exporté par un autre bundle
- ◆ Administrées par le framework
 - Packages ont un numéro de version
- Doivent être validées pour que le bundle soit 'resolved'

Import-Package: org.services
specification-version="1.0.0"

Export-Package: org.services
specification-version="1.0.0"



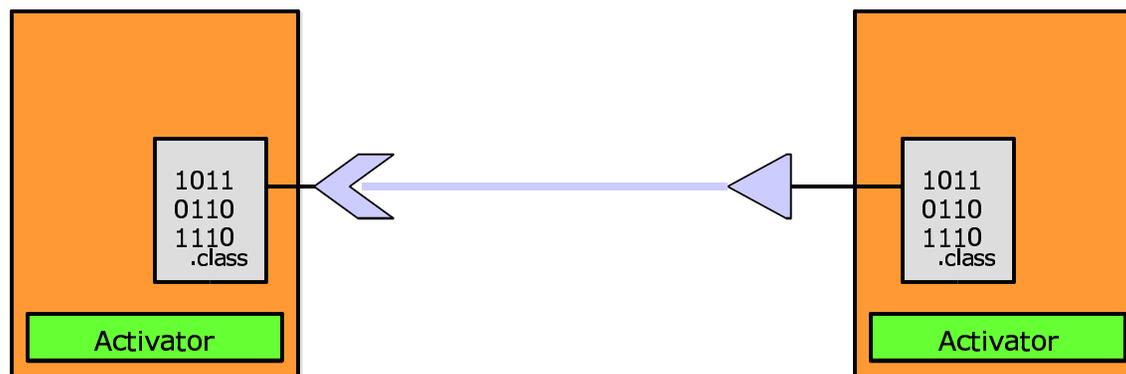


Dépendances de Service

- ◆ Le code à l'intérieur d'un bundle utilise des services implémentés par d'autres bundles
- ◆ Non-administrées par le framework
- ◆ Peuvent être service vers service
- ◆ Statiques, dynamiques, cardinalité... → activator

Import-Service: org.services.LogService

Export-Service: org.services.LogService





Activator du bundle

- ◆ Classe spéciale qui peut être contenue dans un bundle qui reçoit un *contexte* pour accéder aux fonctionnalités du framework:
 - ◆ Administration des bundles
 - ◆ Enregistrement ou récupération des services
 - ◆ Abonnement aux évènements du framework
- ◆ `start(BundleContext ctxt)` et `stop(BundleContext ctxt)`



Enregistrement et courtage d'un service

- ◆ Enregistrement du service:

- ◆ Une interface Java

- `org.services.PrintService`

- ◆ Un objet qui l'implémente

- `org.services.impl.LaserPrintServiceImpl`

- ◆ Des propriétés qui le caractérisent

- `type=laser`

- `location=firstfloor`

- `brand=HP`

- ◆ Courtage à partir d'une requête (0..n réponses):

- Le nom de l'interface du service

- `org.services.PrintService`

- Un filtre avec une syntaxe LDAP

- `(&(type=laser)(location=firstfloor))`



Benefices d'OSGi

- ◆ Puissante plate-forme d'administration des bundles
 - ◆ Installation via URL
 - ◆ Administration a distance
- ◆ Registre de services
 - ◆ Requêtes LDAP



Limitations d'OSGi

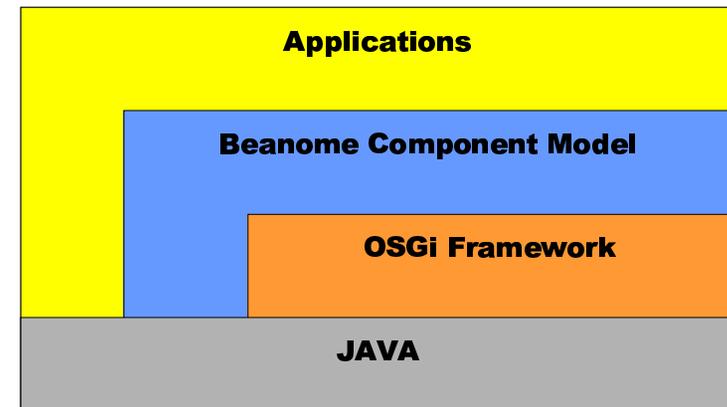
- ◆ Seules les dépendances de package sont gérées
- ◆ OSGi différent des modèles à composant 'classiques'
 - ◆ Les services sont des singletons partagés entre tous les clients
 - ◆ Un bundle est un singleton aussi
- ◆ Pas d'architecture explicite d'une application.
 - ◆ Parce qu'assemblage dynamique
- ◆ Coder l'activator est une tâche complexe.
 - ◆ Enregistrement et récupération de services
 - ◆ Être à l'écoute des divers évènements

Beanome



Objectifs de Beanome

- ◆ Ajouter les concepts 'basiques' d'un modèle à composants (local) au dessus d'OSGi
 - ◆ Types, instances , fabriques
 - ◆ Assemblages
 - ◆ Graphe d'instances interconnectées
 - ◆ Hiérarchies
- ◆ Un registre de composants
 - ◆ Localisation de fabriques
 - ◆ Récupération d'instances (i.e. *finder*)

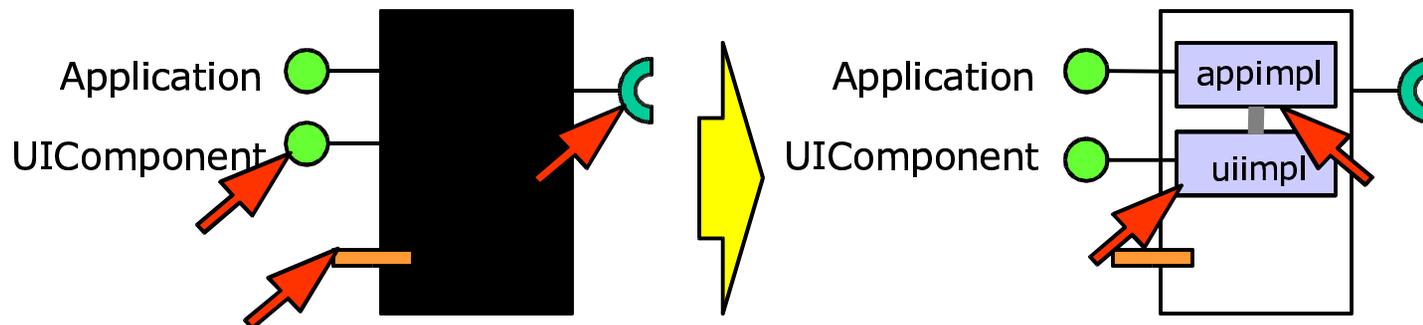




Types de composant

Descripteur de composant (fichier XML)

```
<component name="mycomponents.SimpleComponent" version="1.0.0" runtime="1.3.0" icon="res/icon.gif">  
  <provides name="org.beanome.interfaces.Application" implementation="appimpl"/>  
  <provides name="org.beanome.interfaces.UIComponent" implementation="uiimpl"/>  
  <requires name="org.beanome.viewers.GraphViewer"  
    filter="(&(component=myComponents.GraphViewerComponent)(version=1.0.0))" />  
  <property name="testmode" type="boolean" value="true"/>  
  <implementation name="appimpl" definition="fr.imag.examples.ApplicationImpl" type="class"/>  
  <implementation name="uiimpl" definition="fr.imag.examples.UIComponentImpl" type="class"/>  
  <bind source="appimpl" target="uiimpl" definition="setAppReference" type="method" />  
</component>
```





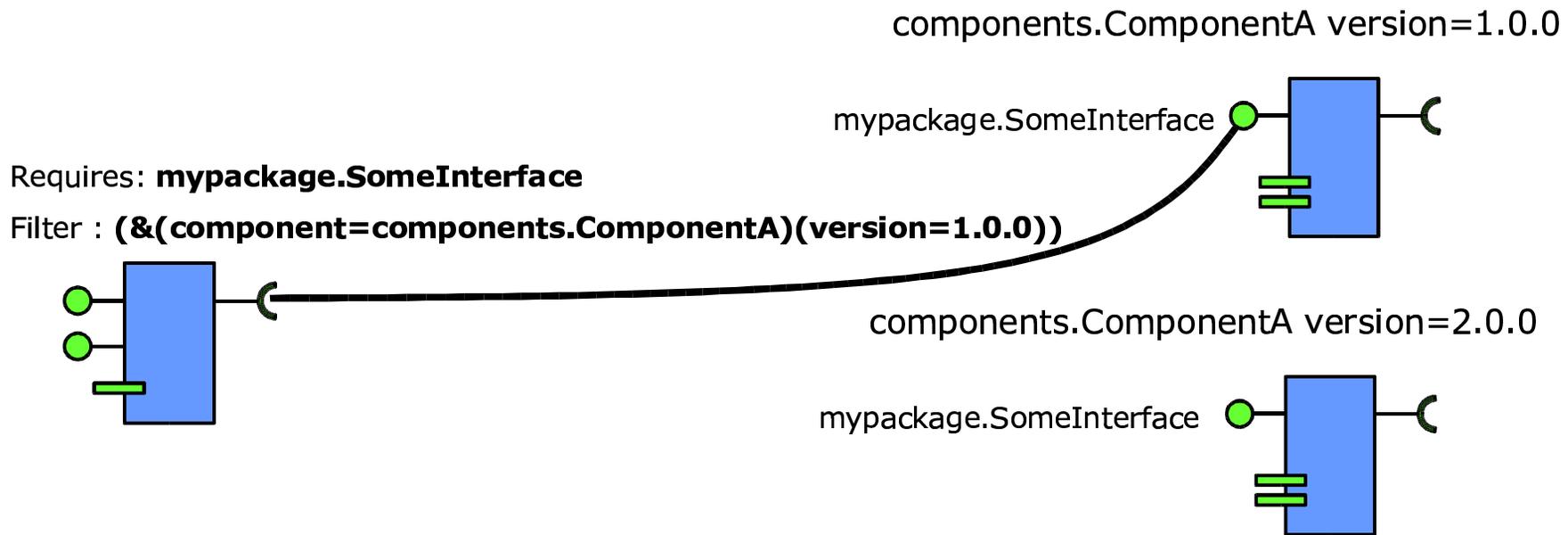
Mapping vers OSGi

- ◆ Le noyau Beanome est contenu dans un bundle.
 - ◆ GenericActivator
 - ◆ Framework (accès au registre, introspection...)
- ◆ Le descripteur de composants étend le manifest.
 - ◆ 1 bundle → plusieurs composants
- ◆ Fabriques créées et enregistrées comme services OSGi.
 - ◆ Les propriétés du service fabrique incluent:
 - ◆ Nom du component
 - ◆ Version
 - ◆ Interfaces fournies
- ◆ Requêtes LDAP pour récupérer les fabriques.
 - Vs. CLSID



Instantiation des Components

- ◆ Dépendances résolues à travers des filtres



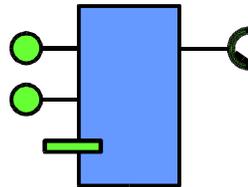


Instantiation des Components

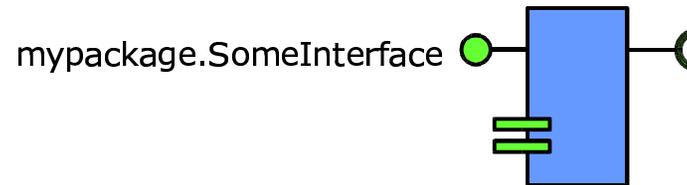
- ◆ A partir des interfaces fournies

Requires: **mypackage.SomeInterface**

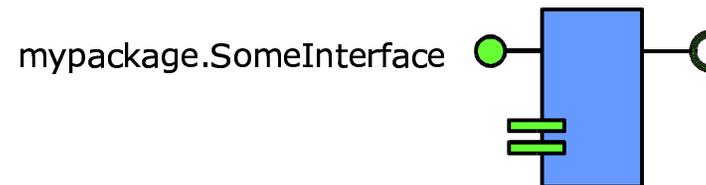
Filter : **(&(provides=*mypackage.SomeInterface*)
(provides=*mypackage.OtherInterface*))**



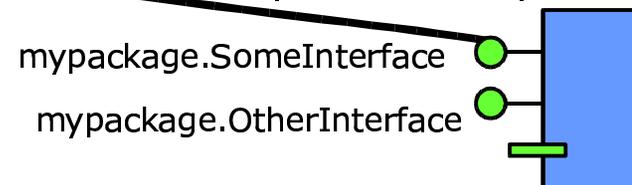
components.ComponentA version=1.0.0



components.ComponentA version=2.0.0

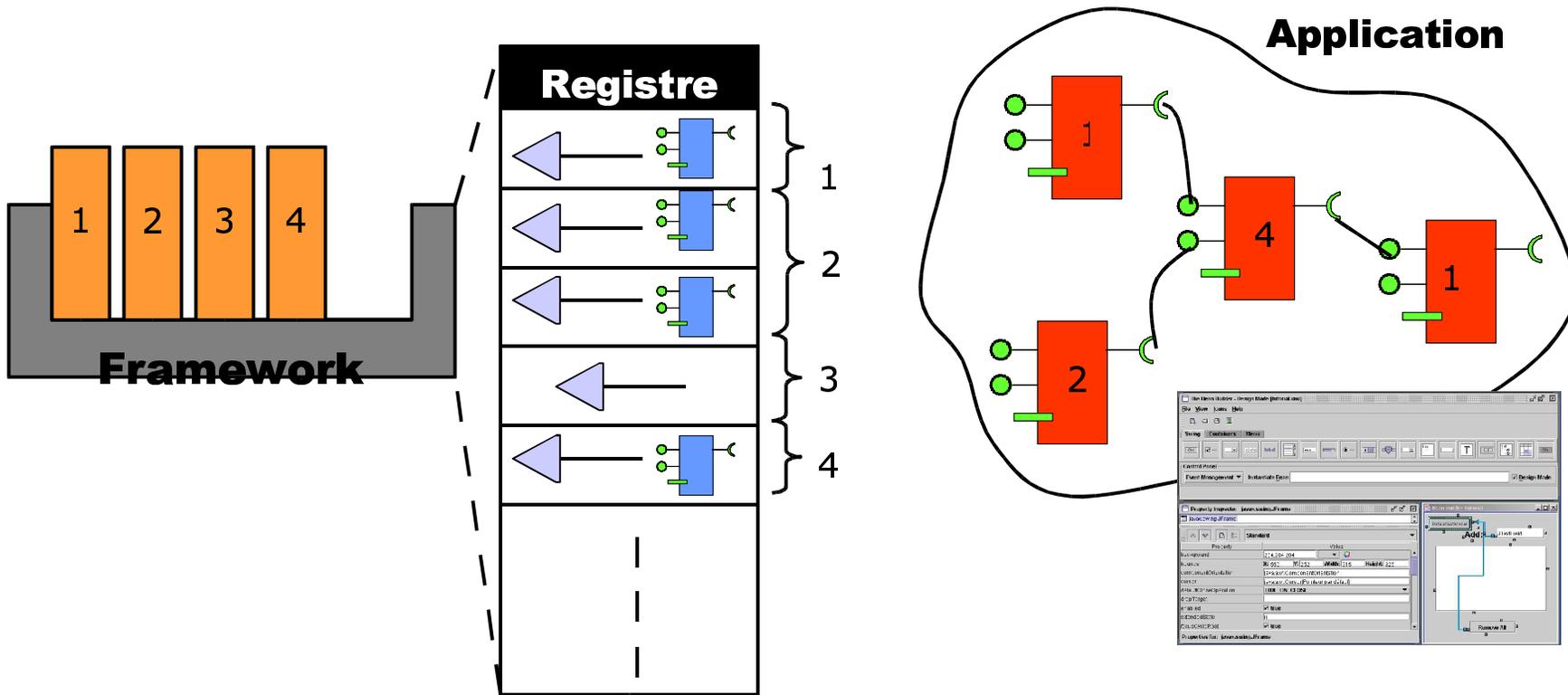


components.ComponentB version=1.0.0





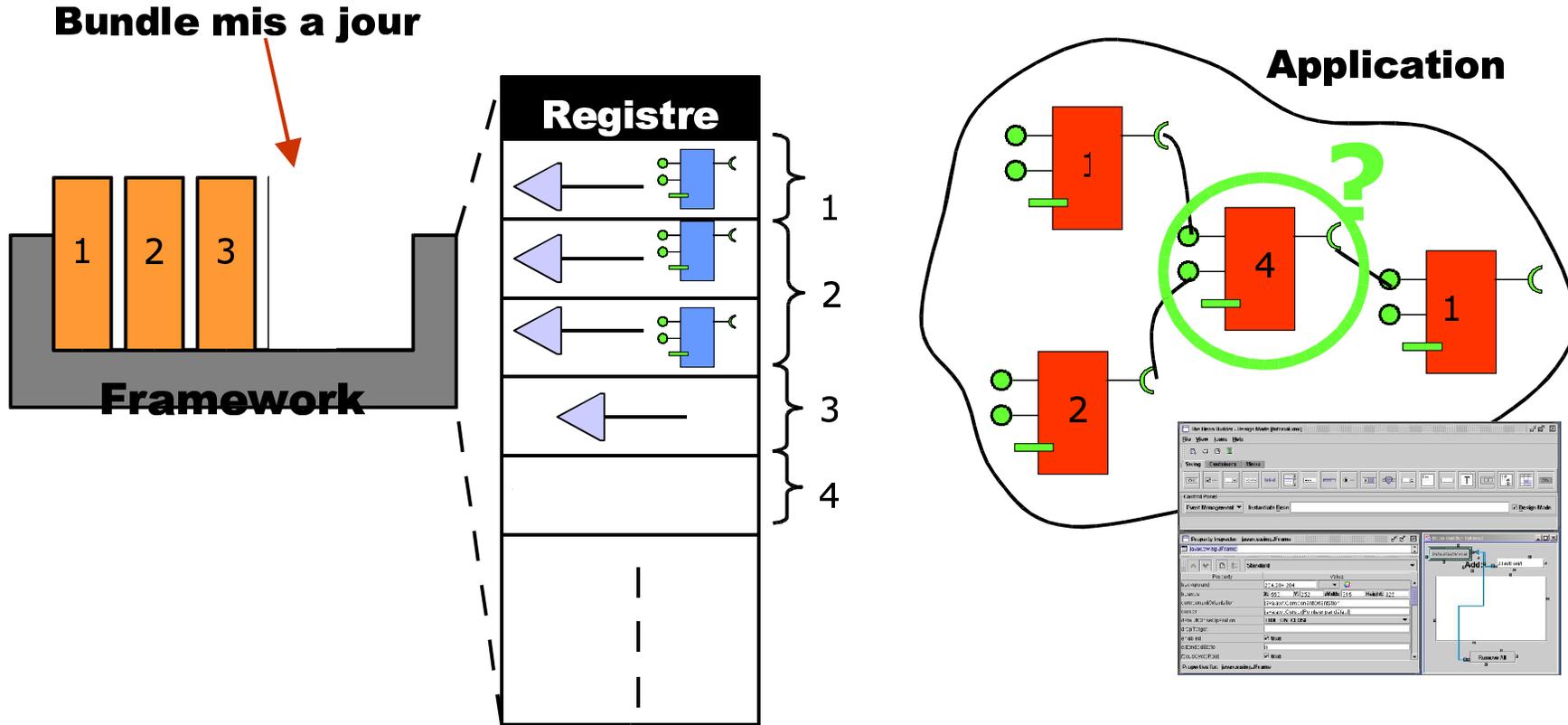
Application Beanome



- ◆ Couvre tout le spectre
- ◆ Runtime et composants ont accès aux services



Application Beanome



- ◆ Comment gérer, au niveau des instances, un événement de mise à jour d'un bundle dans l'application qui s'exécute?

Adaptation dynamique

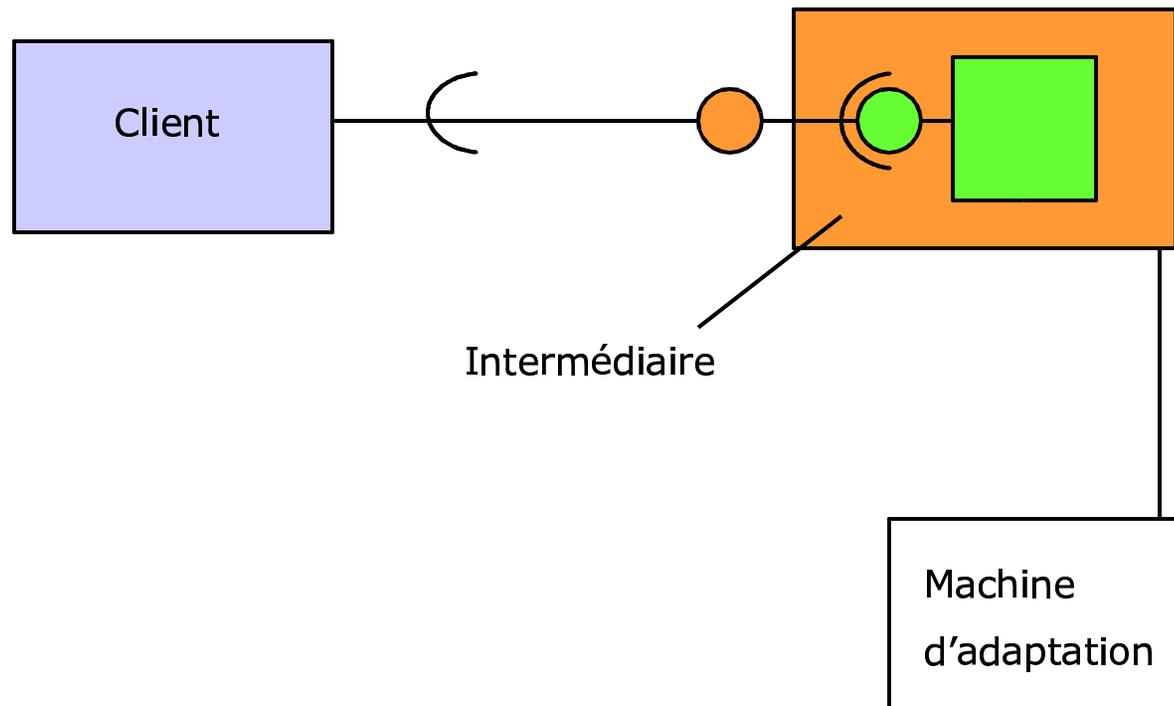


Adaptation dynamique

- ◆ Changements au niveau d'une instance
 - ◆ Application en exécution.
 - ◆ Nécessaire dans des applications a haute disponibilité
- ◆ 2 cas d'adaptation:
 - ◆ Une autre instance issue du même type de composant
 - ◆ Même interface
 - Une instance issue d'un type de composant différent.
 - Interface différente
 - ◆ Emploi de règles de correspondance
 - ◆ Pour transfert d'état, et reconnexion

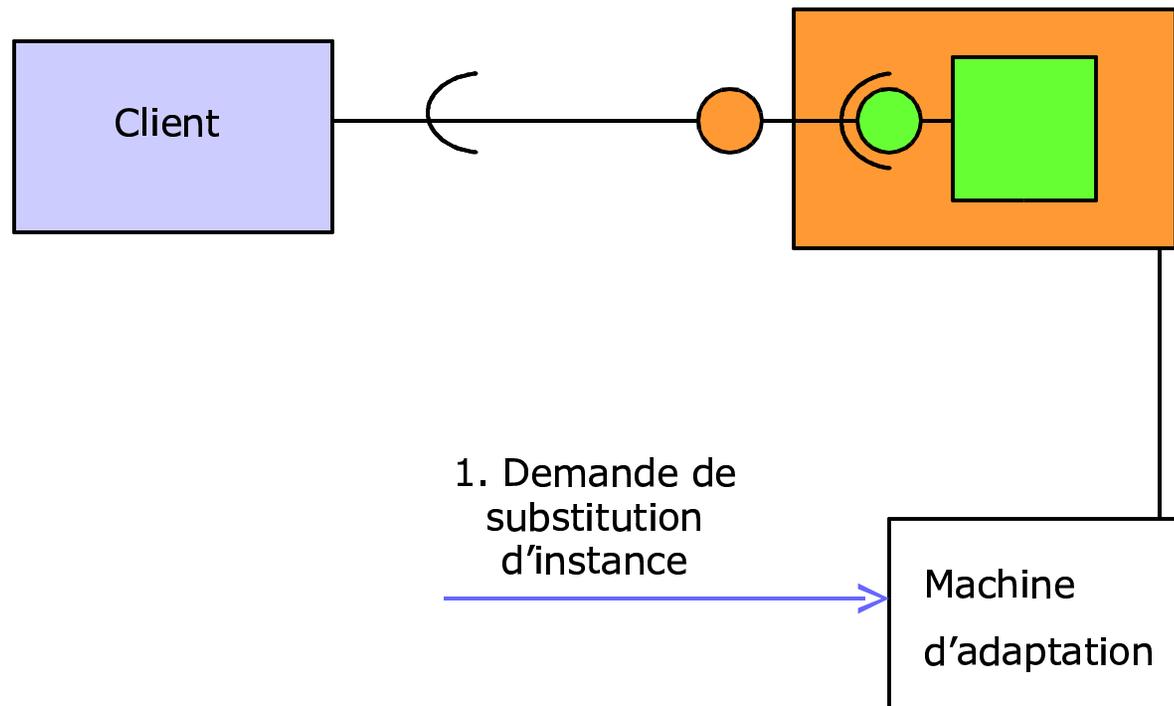


Principe d'adaptation



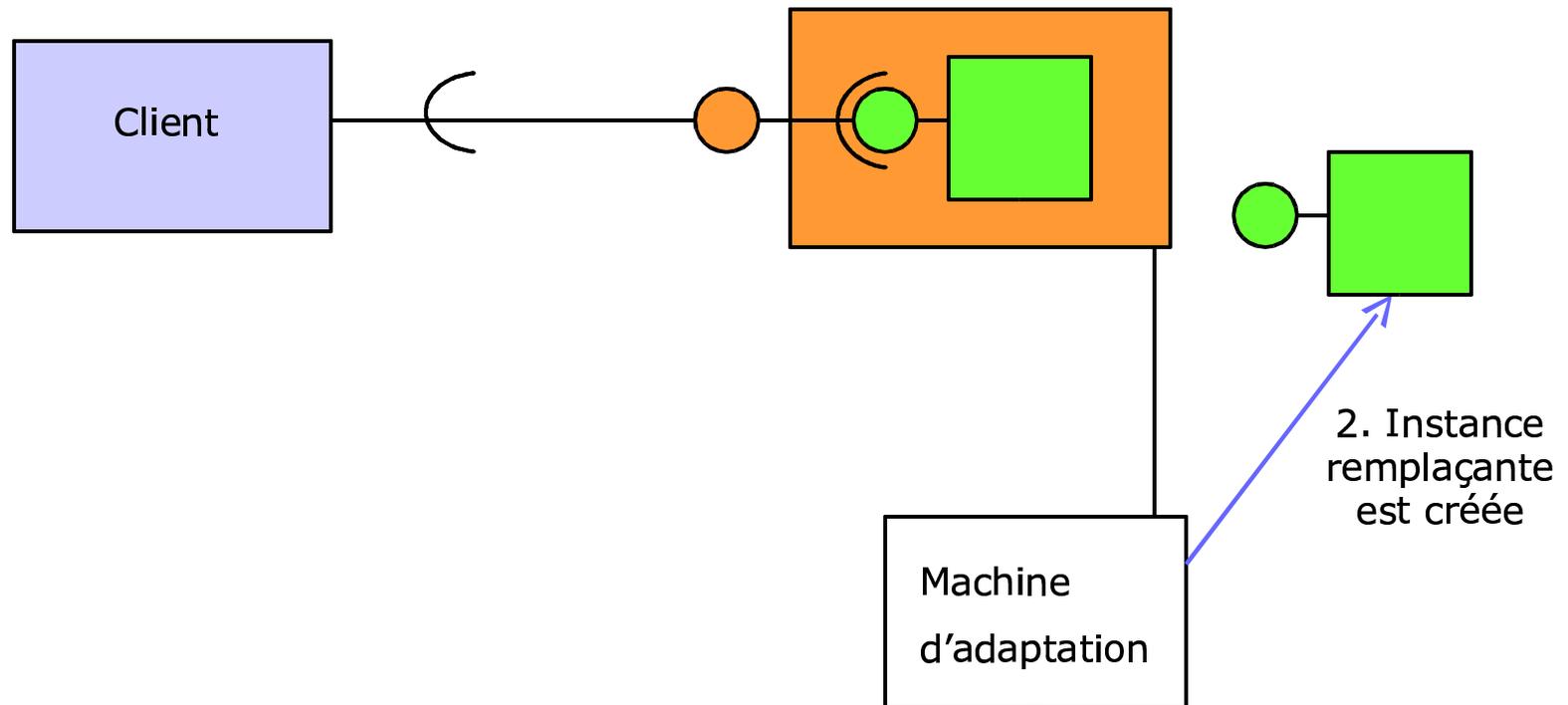


Principe d'adaptation



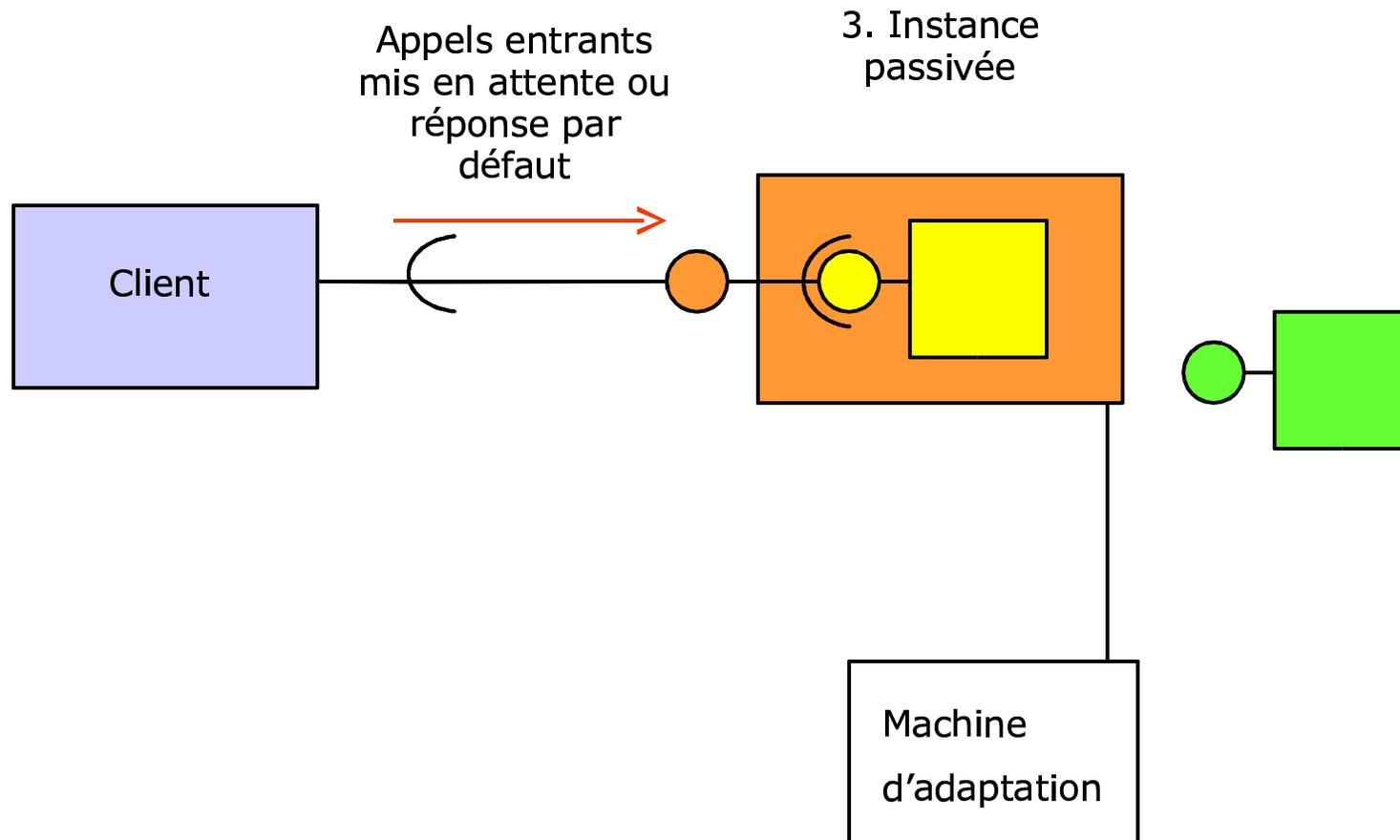


Principe d'adaptation



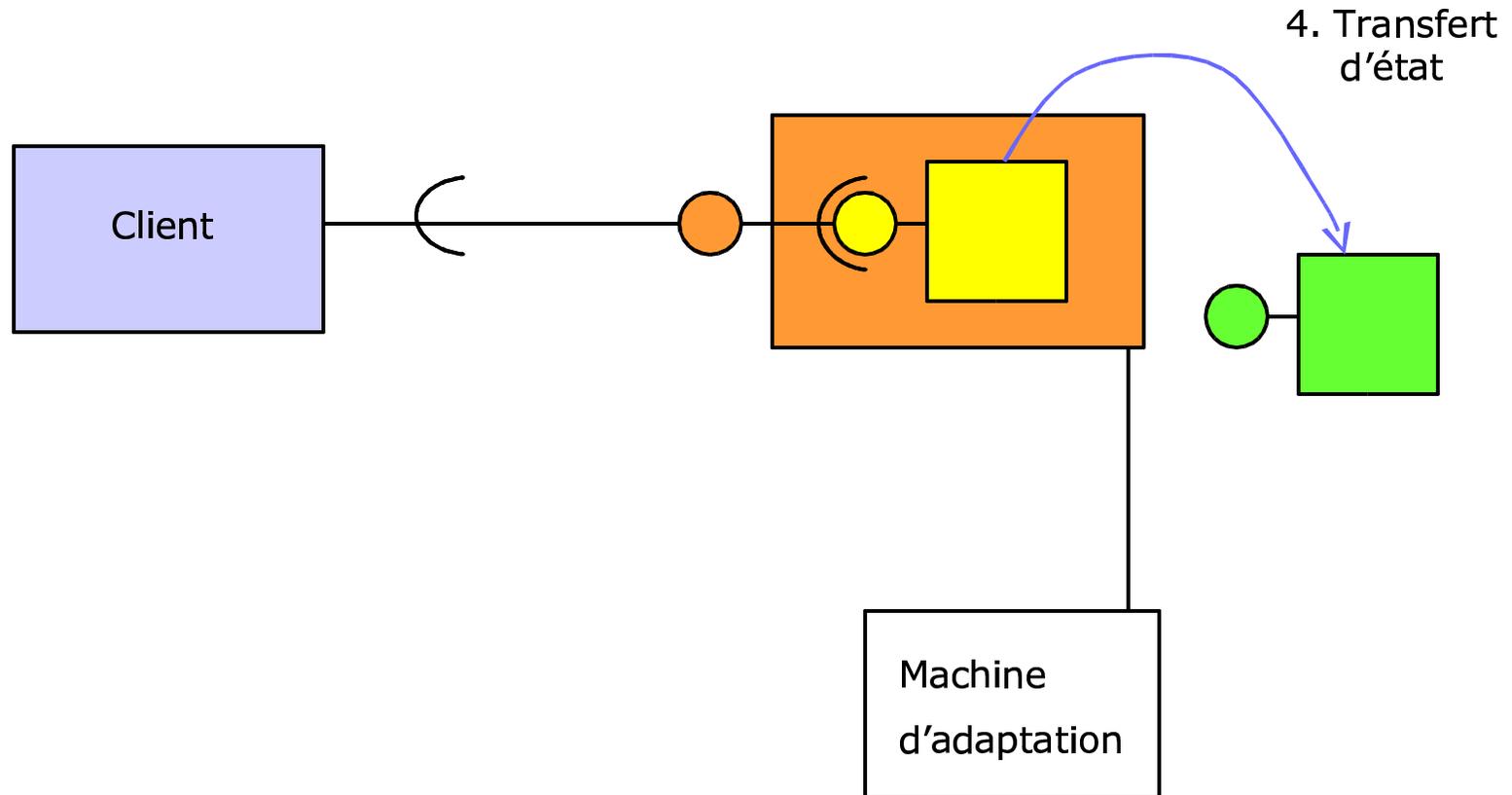


Principe d'adaptation





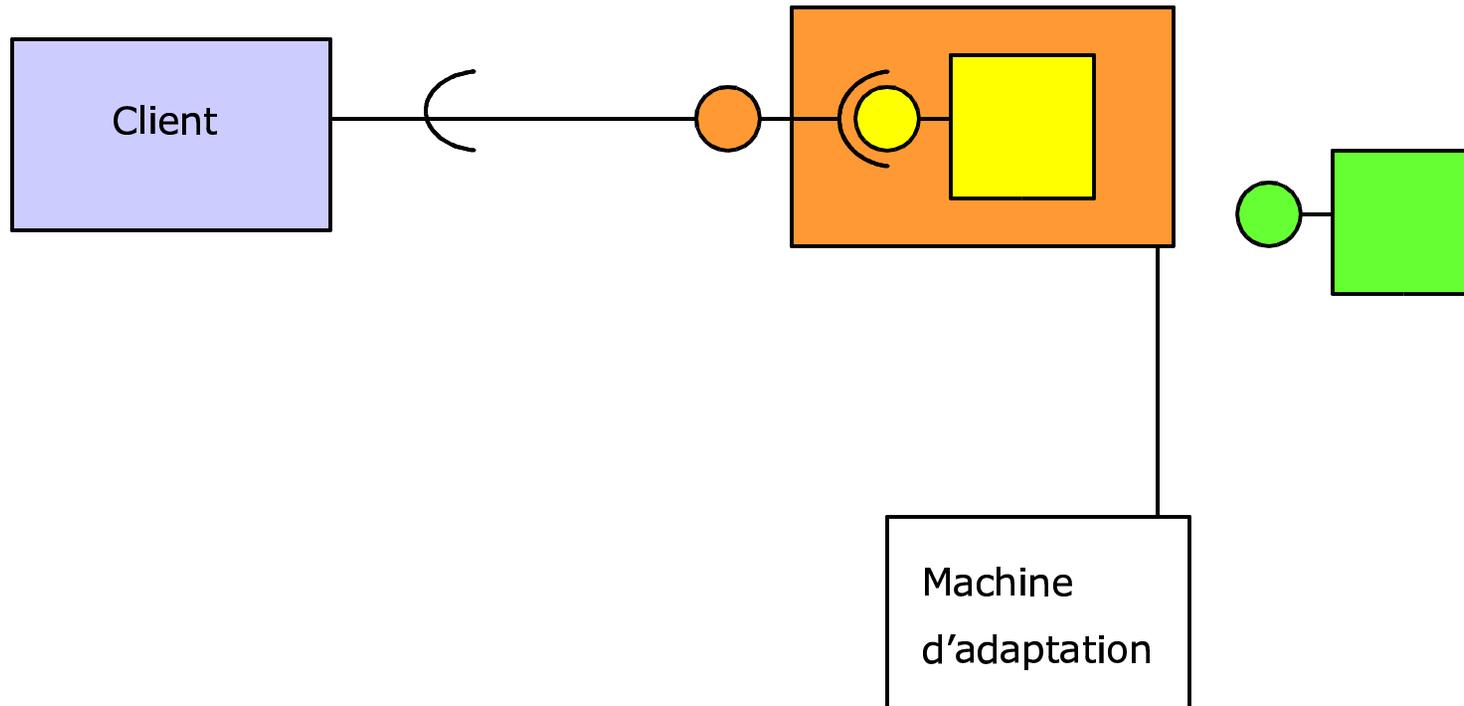
Principe d'adaptation





Principe d'adaptation

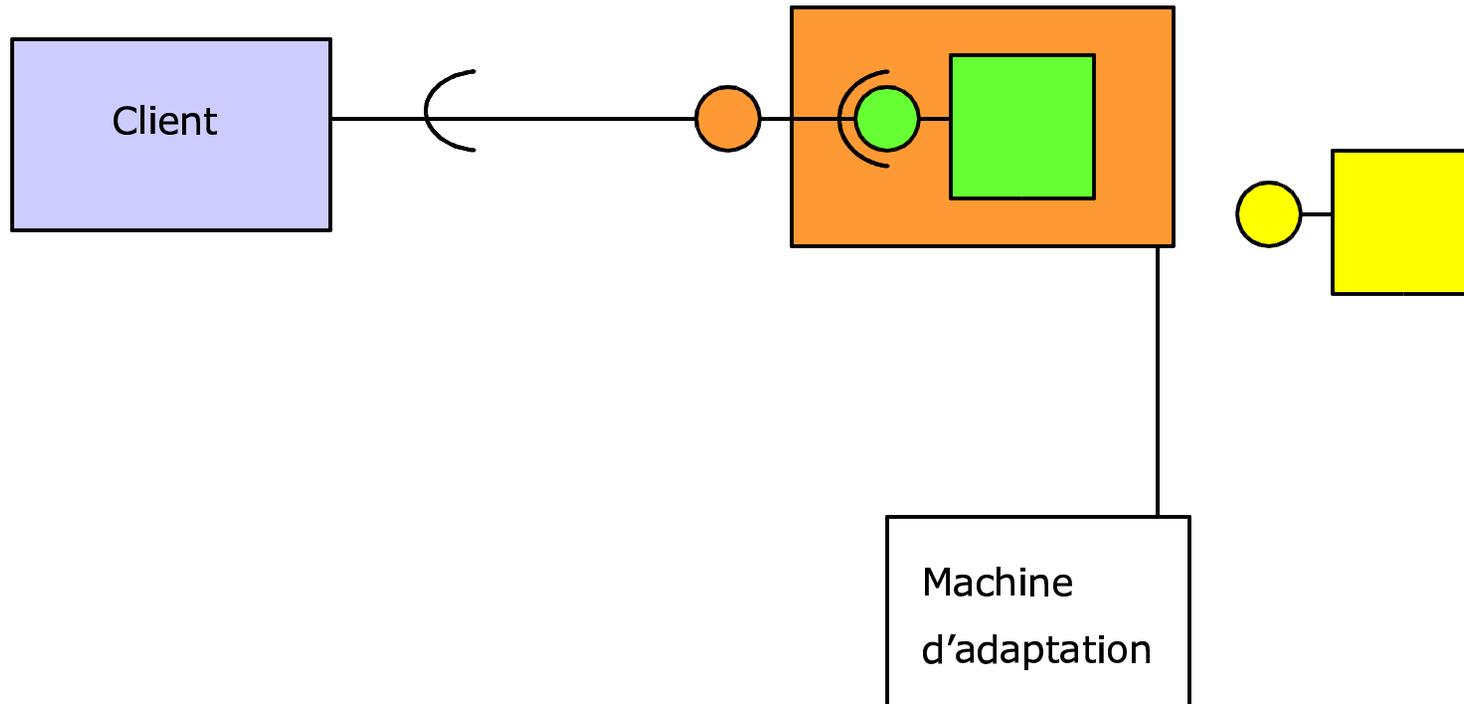
5. Déconnexion de l'instance à remplacer





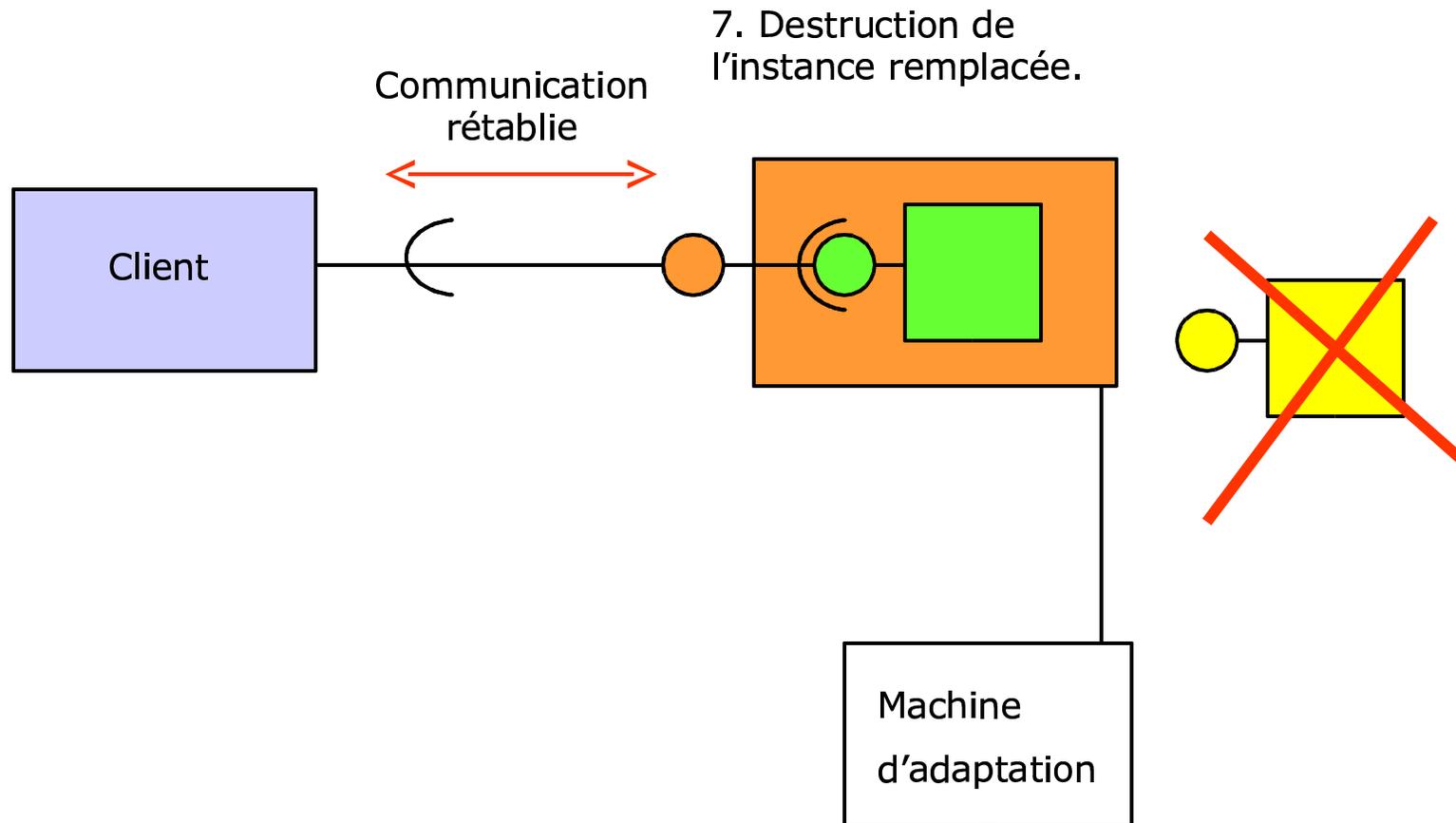
Principe d'adaptation

6. Connexion de la nouvelle instance au moyen des règles





Principe d'adaptation



Conclusions & Travaux Actuels



Conclusion

- ◆ OSGi est un candidat idéal pour être une couche de base pour un modèle à composant
 - ◆ Très bonne gestion des unités de livraison + registre
 - ◆ Très dynamique
 - ◆ Non distribué
- ◆ Il faut cependant rajouter certains concepts
 - ◆ Type, instances, fabriques
- ◆ Adaptation dynamique
 - ◆ Regles de correspondance
 - ◆ Solution pour les mises à jour
 - ◆ Besoin d'intermédiaires : containers



Travaux en cours

- ◆ Formalisation de l'adaptation dynamique
- ◆ Gravity
 - ◆ Reprend principes introduits dans Beanome
 - ◆ Regles pour automatiser dépendances
 - ◆ Bundle vers service
 - ◆ Service vers service
 - ◆ Instance vers service
- ◆ Conteneurs implémentés par services
- ◆ CCM au dessus de OSGi

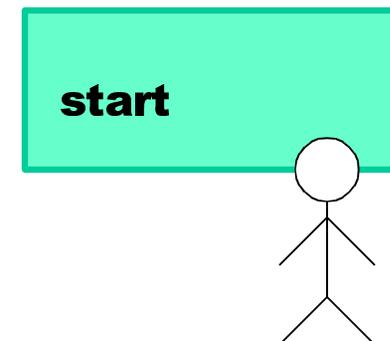
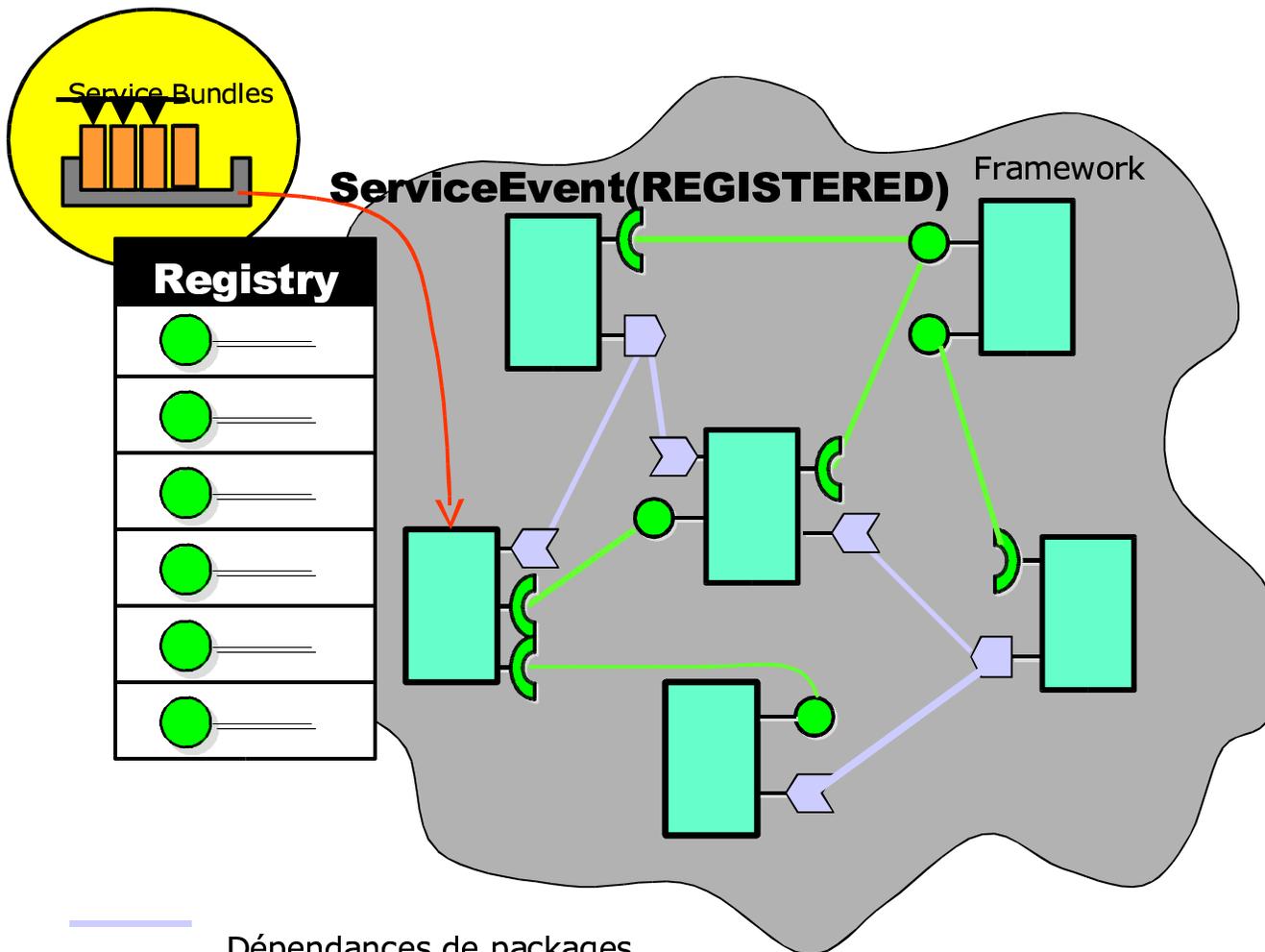


Liens

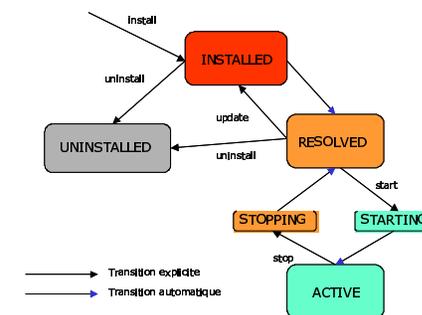
- ◆ Beanome
www-adele.imag.fr/BEANOME
- ◆ OSCAR implementation open source de OSGi
<http://oscar-osgi.sourceforge.net>



Une application OSGi



Bundle :



Dépendances de packages

Dépendances de services