14 octobre 2008

Paris JUG

www.parisjug.org

PARIS JUG

14 octobre 2008

OSGI

Cyrille Le Clerc

Nicolas Griso

Xebia

www.parisjug.org

La modularité

Caractéristiques d'un module ?

- Expose un contrat versionné
- Masque ses détails d'implémentation
- Décrit ses dépendances

Pourquoi la modularité ?

- **Limites du monolithique**

  Windows Vista (50 M SLOC) est le dernier OS monolithique de Microsoft

- **Réutilisation et émergence de stacks**

  Middleware Java
  - Serveur JavaEE, ESB/BPM, portail, télécoms,
  - Consolidation de marché
  - Maturité des briques et API de base

- ***« One size does NOT fit all ! »***

  Profiles Java EE 6

Xebia

Business Integration Architects

La modularité en Java

L'existant

- *Les jars*
- *Les classloaders hiérarchiques*
- *Maven 2*

Le futur

- *Java Module System*
- *OSGi*

Module de base : le Jar

- Concept de build sans réalité au runtime
- Pas de gestion de version
  Une seule version d'un jar peut être chargée
- Pas d'intermédiaire de visibilité entre `protected` et `public`
- Pas de description des dépendances

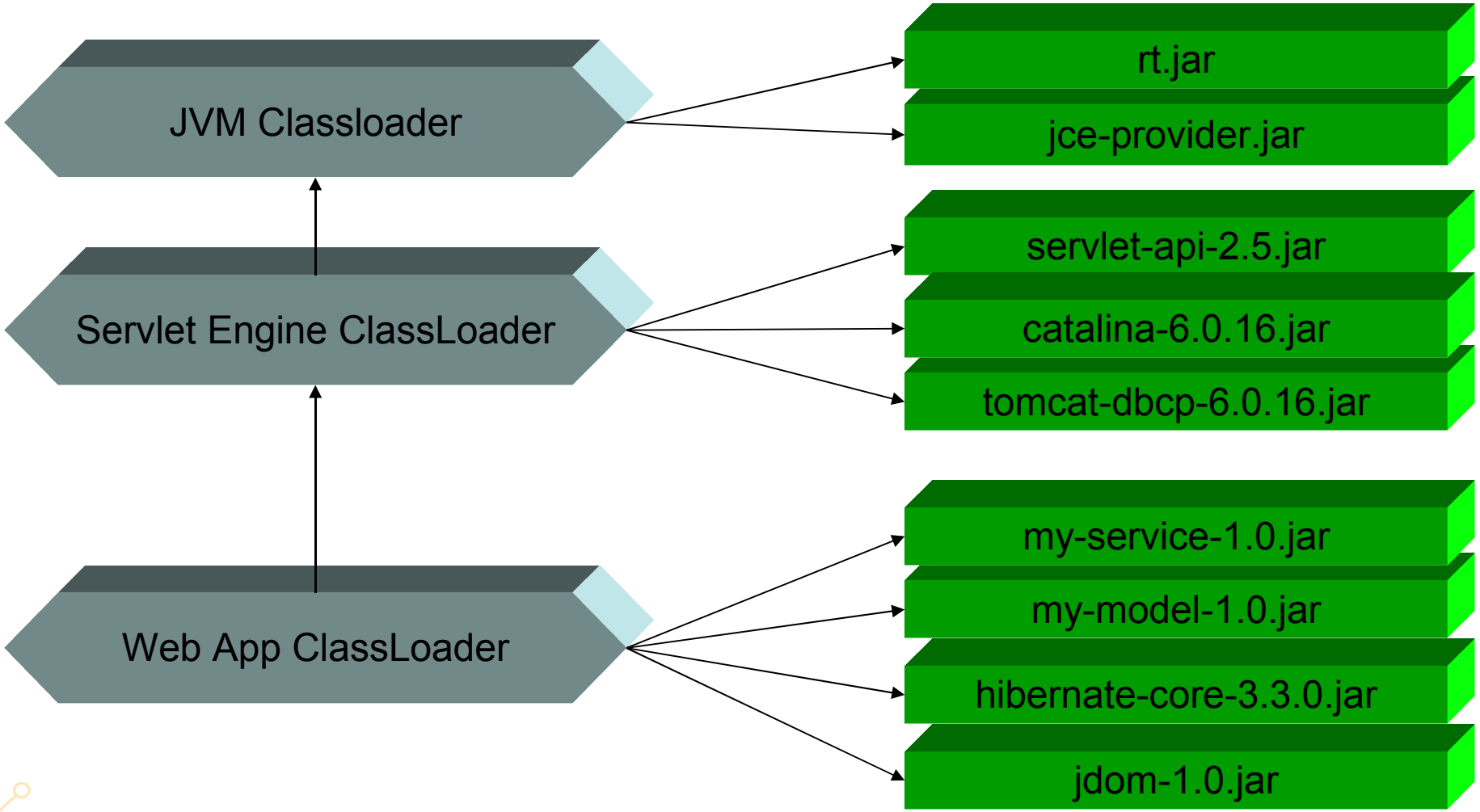
Java s'est jusqu'à présent peu soucié du concept de module

La modularité en Java
L'existant : les classloaders hiérarchiques

- **Les classloaders sont hiérarchiques et héritent de la visibilité du parent**
- **Visibilité globale dans un classloader**
- **Impossibilité de charger plusieurs version d'une classe dans un classloader**
- **Les packages ne sont pas contraints au runtime**

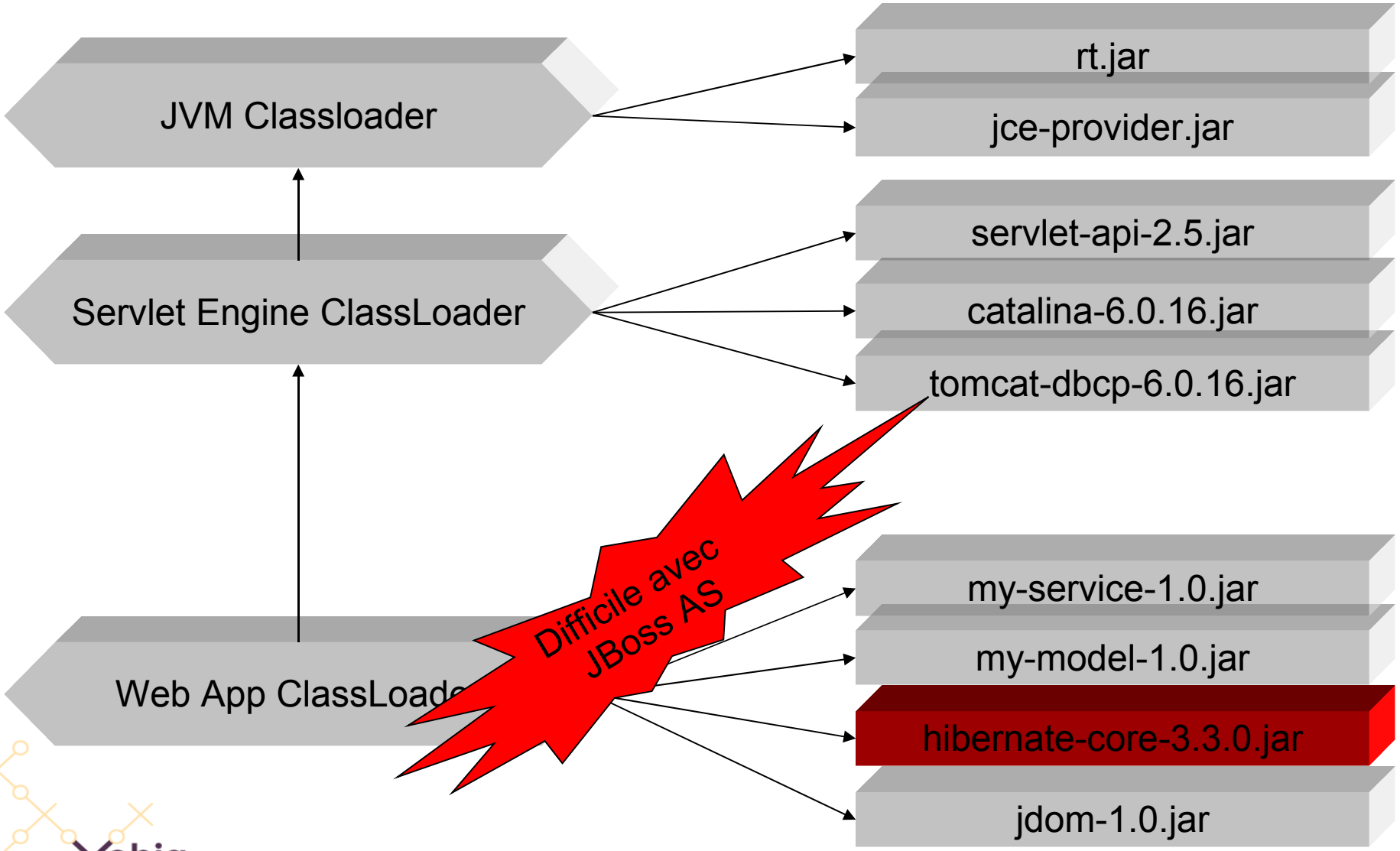
  Deux classes d'un package peuvent provenir de jars différents

La modularité en Java
L'existant : les classloaders hiérarchiques

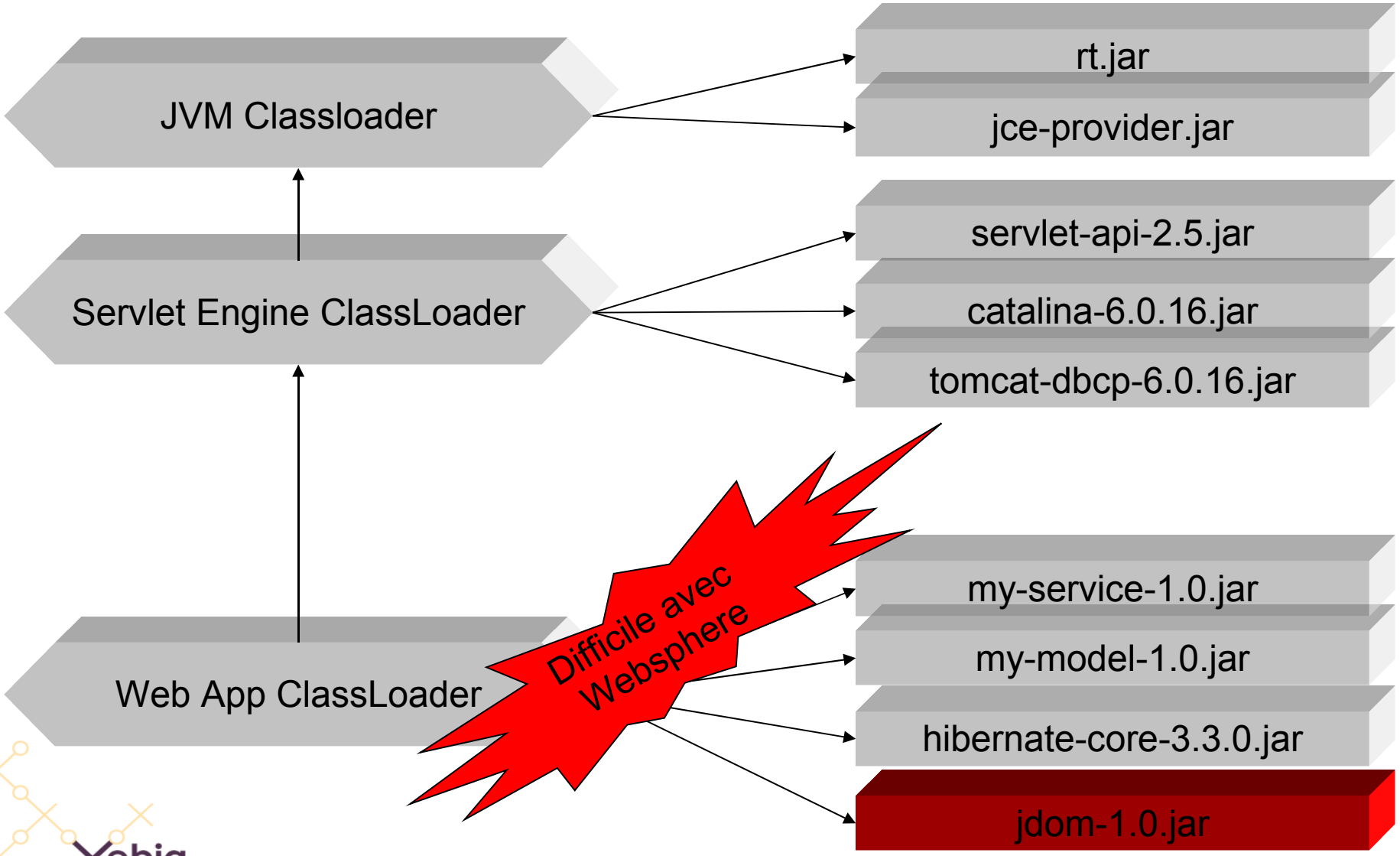


Web Application Classique

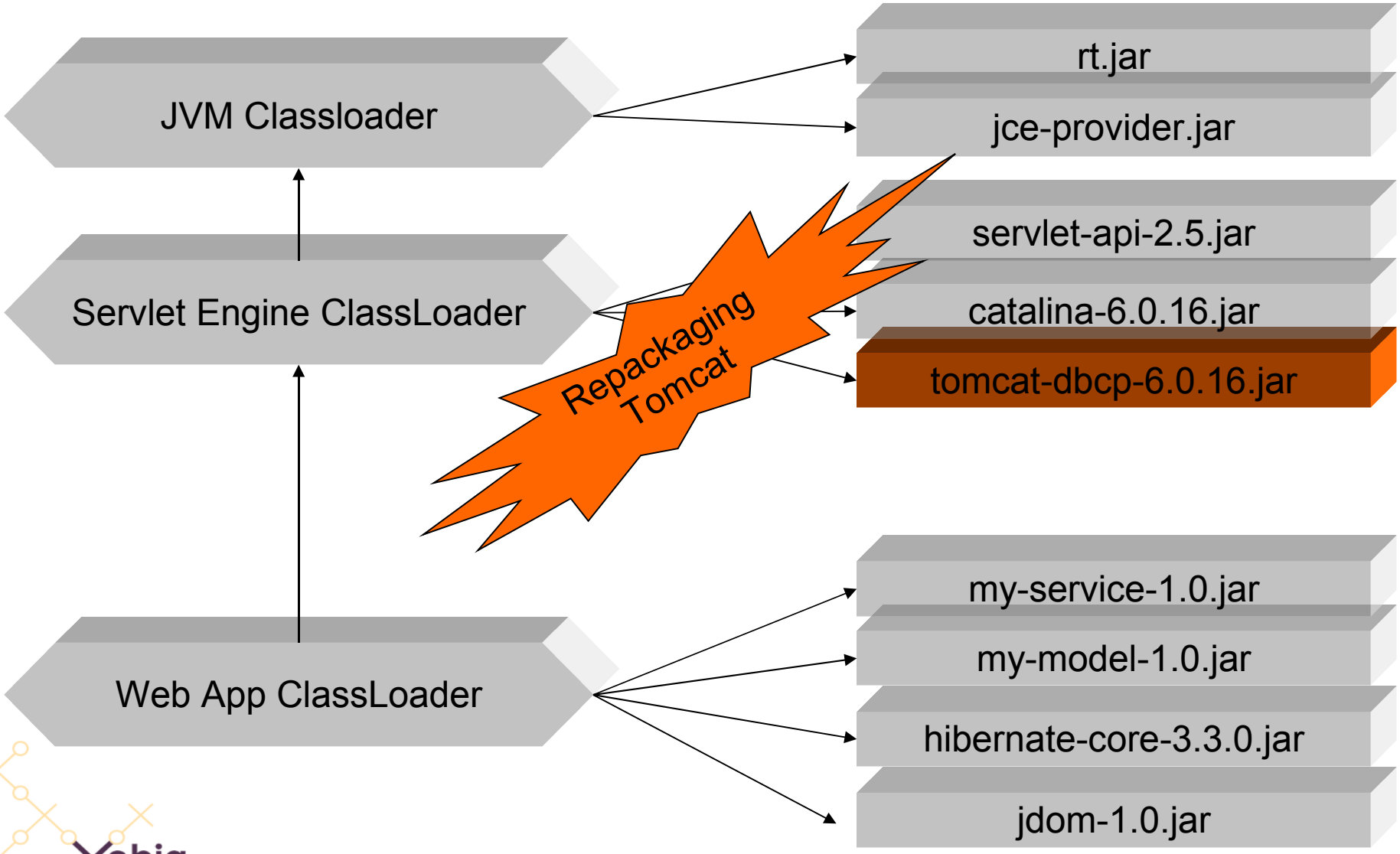
La modularité en Java
L'existant : les classloaders hiérarchiques



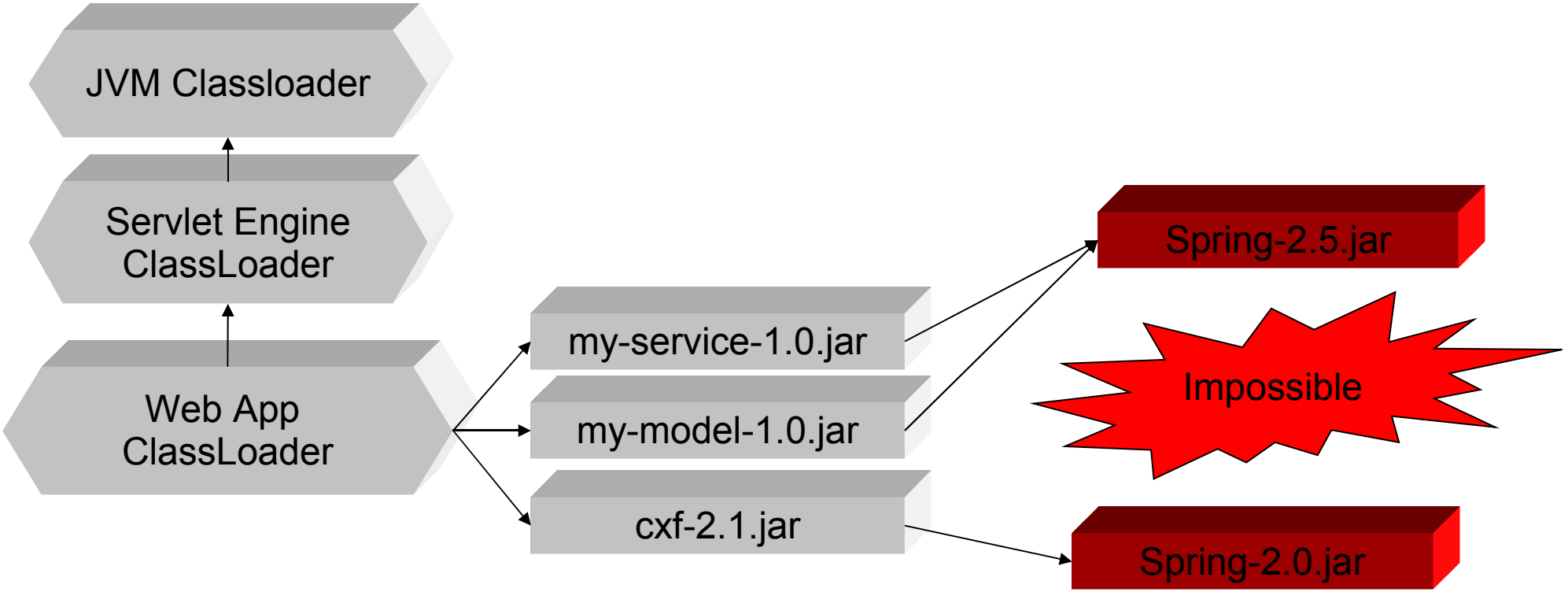
La modularité en Java
L'existant : les classloaders hiérarchiques



La modularité en Java
L'existant : les classloaders hiérarchiques



La modularité en Java
L'existant : les classloaders hiérarchiques

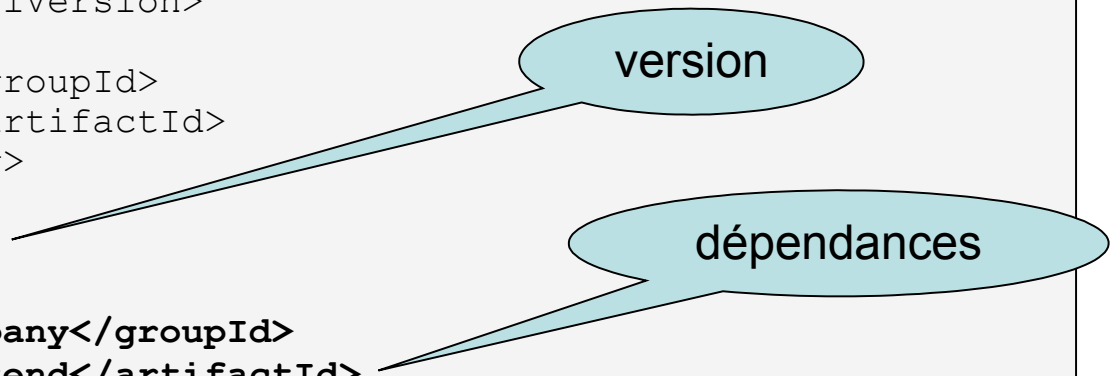


Il est aujourd'hui impossible de charger deux versions d'un même jar dans une web app !

Maven 2

- Versionnage des jar
- Description des dépendances

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <name>My Service</name>
   <groupId>com.mycompany</groupId>
   <artifactId>my-service</artifactId>
   <packaging>jar</packaging>
   <version>1.0</version>
   <dependencies>
      <dependency>
         <groupId>com.mycompany</groupId>
         <artifactId>my-backend</artifactId>
         <version>1.0</version>
      </dependency>
    </dependencies>
</project>
```



Les limitations

- Dépendances explicites seulement au build (pas runtime)
- Pas de visibilité *bundle*

Les défis

- Qualité des méta-données
- Dépendances transitives
- Conflits de versions
- Dépendances par nom de *bundle* vs. par package

- **OSGi rejoint le JCP**

- **OSGi**
  - Unité de travail : le bundle
  - Gère la visibilité des bundles et leur dépendances, leur versionning
  - Dynamique (cycle de vie des bundles géré au runtime)
  - Orienté service
    - » Les bundles publient des services dynamiquement
    - » Recherche de services
    - » Bind
    - » Cycle de vie des services lié aux bundles qui les supportent

La modularité en Java
Le futur : JSR 277 – Java Module System

- **Associé à JSR 294 - super packages**

- **Utilisation des annotations Java 5 pour décrire les dépendances et les version**

```
@Version("1.0")
@ImportModules({
   @ImportModule(name="com.mycompany.mybackend", version="1.0+")
 })
super package com.mycompany.myservice {
   export com.mycompany.myservice.*;
}
```

Vive opposition des partisans d'OSGI

Xebia

Business Integration Architects

OSGi Alliance

L'histoire
Le fonctionnement
OSGi Alliance et le JCP

OSGi Alliance
L'histoire

- **Consortium créé en 1999**
- **Objectifs initiaux : Java dans le monde embarqué (domotique, automobile, telco, etc)**
- **Membres : EDF, Siemens, BMW, Ericsson, Nokia, Motorola, Sprint, IBM, SpringSource, Sun, etc**
- **Date clefs**
  - » 1999 : l'embarqué et networked devices, la domotique, l'automobile
  - » 2003 : les télécommunications
  - » 2004 : virage open source (Eclipse)
  - » 2006 : server side Java
- **Besoin d'être léger et dynamique dès l'origine**

  Besoin d'isolation des composants fournis par différents éditeurs

OSGi Alliance
Le fonctionnement

- **Les experts group**
  - Core platform EG
  - Vehicle EG
  - Mobile EG
  - Entreprise EG
  - Residential EG
- **Entreprise Expert Group**
  - Gestion de la scalabilité (multi-conteneur, multi-processeurs)
  - Ouverture à d'autres languages
  - Distribué (SCA)
  - Intégration à JavaEE
  - Modèle de composants (Spring DM)

OSGi Alliance
OSGi et le JCP

- **1999 : JSR 8 – OSGi specification => retirée en 1999**
- **2006 : JSR 291- Dynamic Component Support for JavaTM SE**
  - Spec Lead : IBM
  - Craintes que le JCP n'ai qu'un rôle de validation du travail de l'OSGi Alliance
  - Craintes du chevauchement avec JSR 277 – Java Module System
- **2008 : Sun embauche Richard Hall (Apache Felix) => intégration d'OSGi dans Glassfish**

Une communauté plus fermée que le JCP ?

	OSGi Alliance	JCP
Direction	Partagée	Exclusive Sun
Membres	Principalement des entreprises	Variés : entreprises, universitaires, individuels
Spécifications release	Public	Public
Spécifications drafts	Membres payants	Public
TCK	Membres payants	Membres payants + OSS
Reference Impl	Membres payants	Public
Débats	Confidentiels	De plus en plus publics

OSGi Alliance Full Membership : 20 k$/an

Xebia

Business Integration Architects

La plateforme OSGI

Les bundles

Le réseau de classloader

Les contraintes du Manifest

- **Unité de déploiement sur la plateforme**
- **Un simple jar**
  - Plus quelques entrées dans `META-INF/MANIFEST.MF`
- **Peut être utilisé hors contexte OSGi**

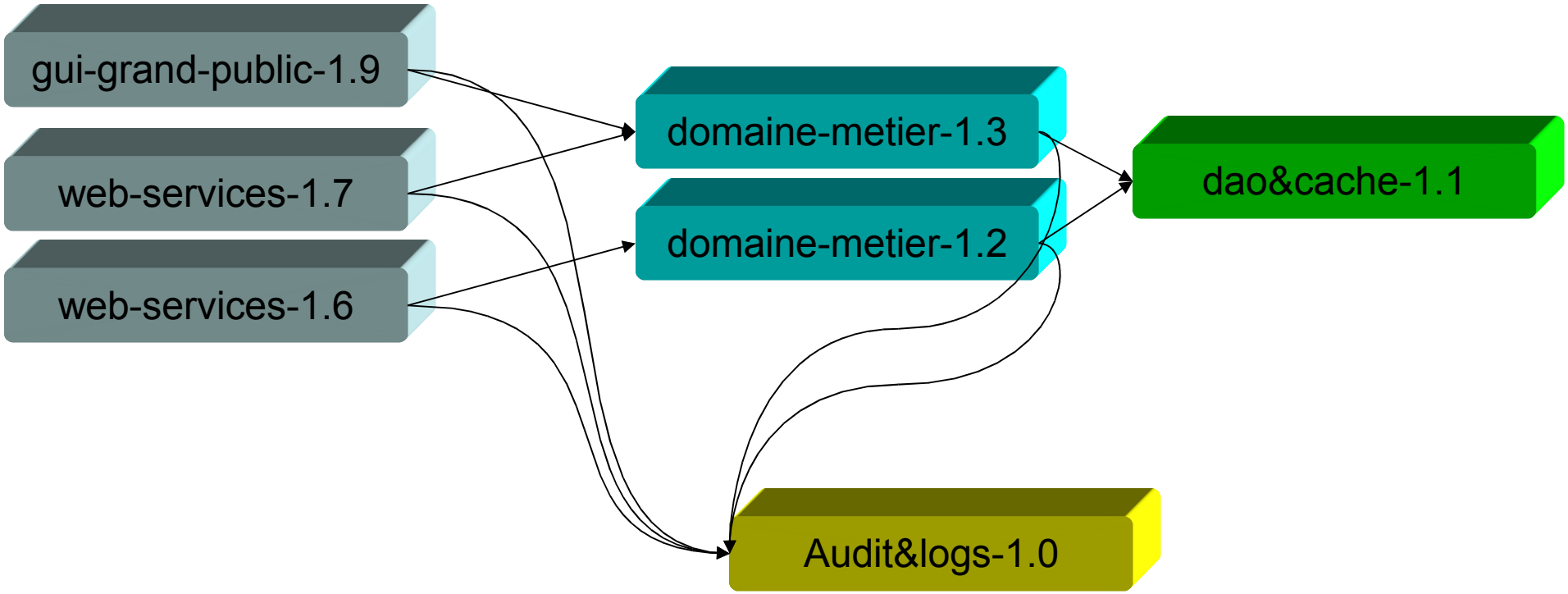
Déclaration des dépendances niveau package

Déclaration des packages exportés

Description du bundle

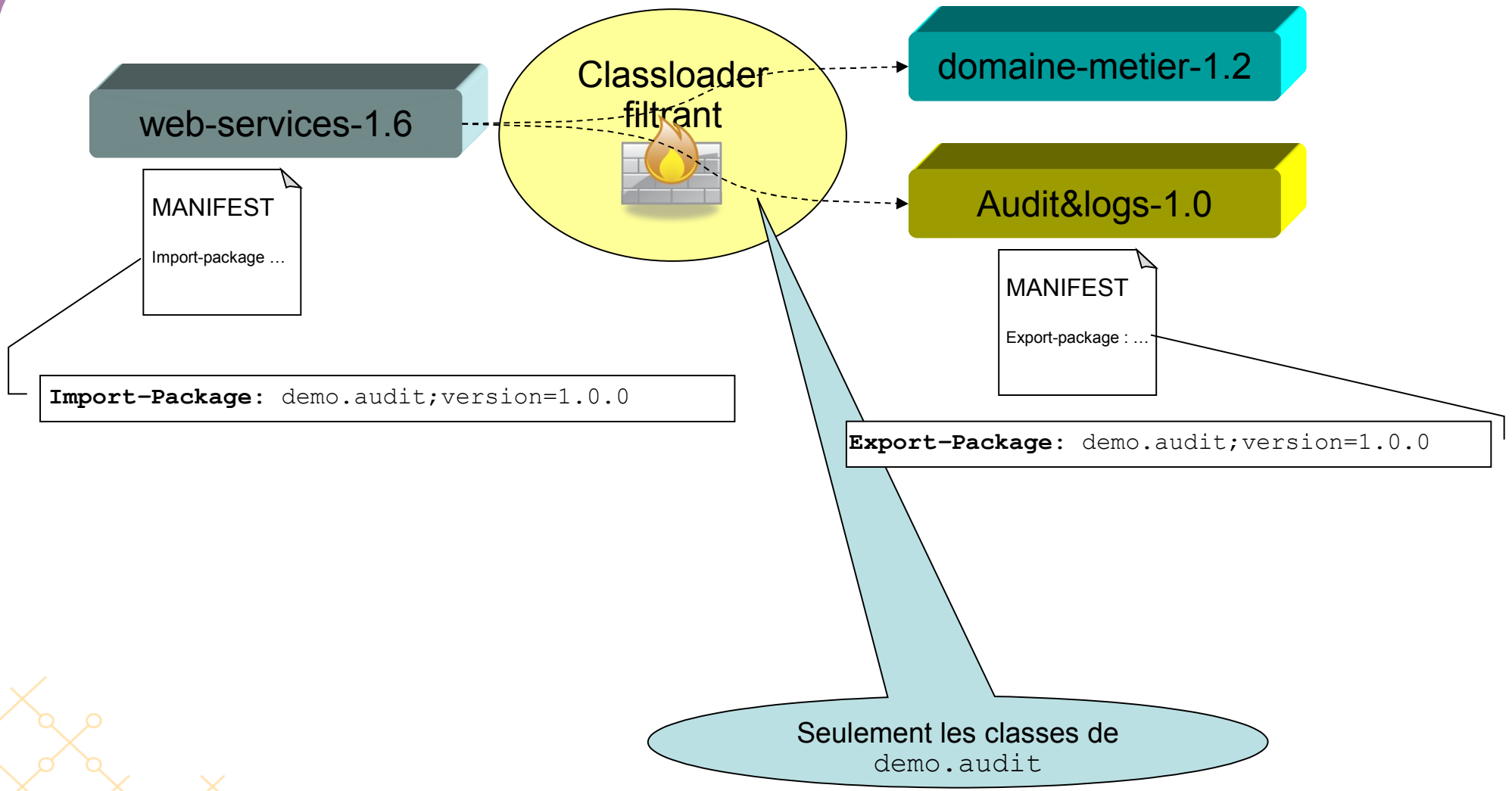
```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: My Service
Bundle-SymbolicName: com.mycompany.myservice
Bundle-Version: 1.0.0
Bundle-Activator: com.mycompany.myservice.Activator
Import-Package: javax.servlet;version="2.4.0",
 javax.servlet.http;version="2.4.0",
 org.osgi.framework;version="1.3.0",
 org.osgi.service.http;version="1.2.0",
 org.osgi.util.tracker;version="1.3.1"
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
Export-Package: com.mycompany.myservice
Require-Bundle: org.apache.log4j;bundle-version="1.2.13",
 com.mycompany.backend;bundle-version="1.0.0"
```

- **Le versionning**
  - major.minor.micro.qualifier
    - `3.4.0.v20080603-2000`
  - `[version, version]`
  - `(version, version)`
  - `[version, version)`



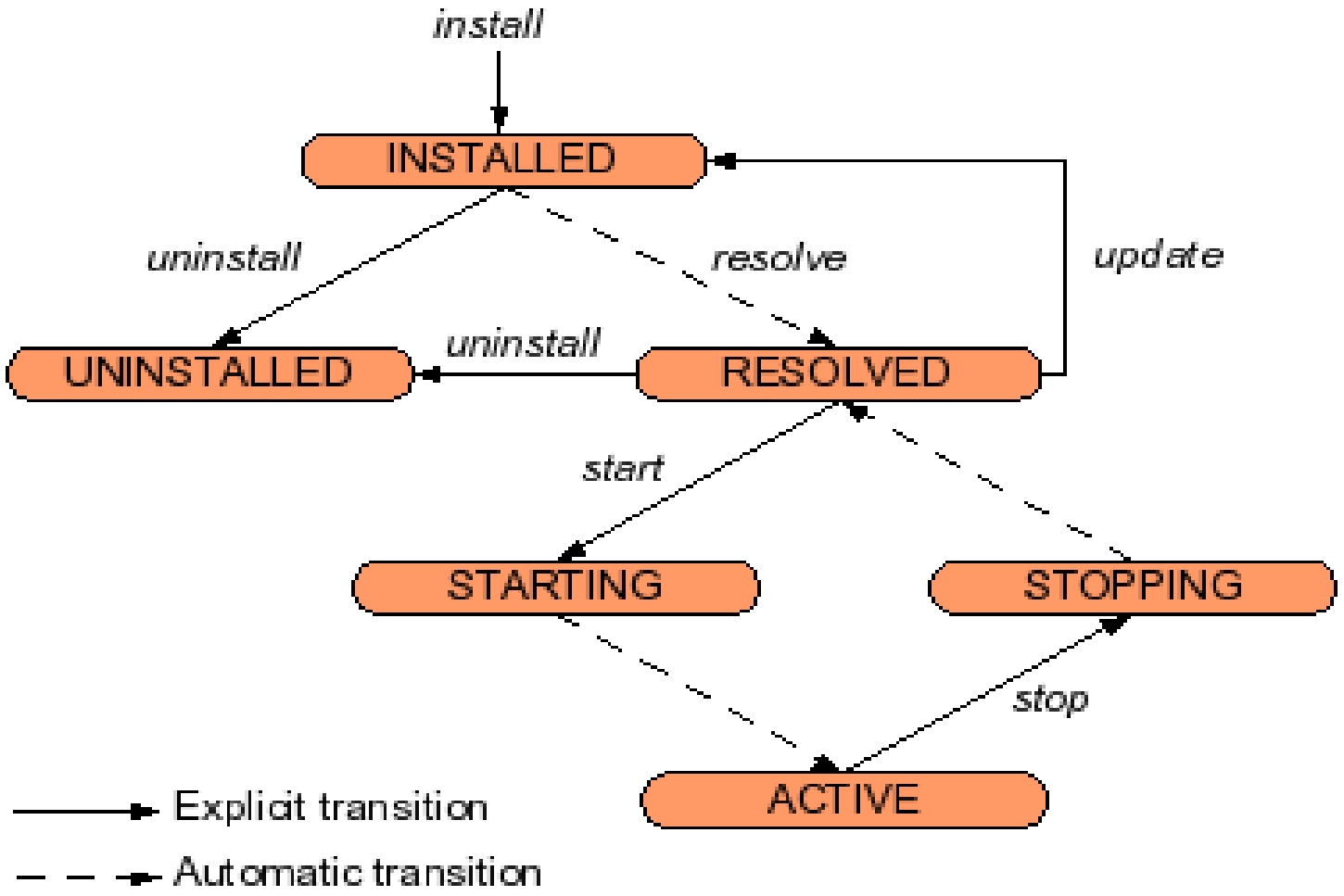
Exemple de graphe de modules

La Plateforme
Le réseau de Class Loaders



- Environnement d'exécution des bundles
- Standalone ou embarqué
- Gère le cycle de vie des bundles

La plateforme
Le cycle de vie des bundles



La plateforme
Le Service Registry

- **Cœur du framework**
- **Gère les services. Permet de**
  - Publier des services
  - Rechercher des services
  - Binder des services
- **Découplage entre services**

Assemblage des services OSGI
BundleActivator et ServiceTracker

```
public class Activator implements BundleActivator {

    public void start(BundleContext context) throws Exception {
        // INSTANTIATE MY_BACKEND_SERVICE IMPLEMENTATION
        MyBackendService myBackendServiceImpl = new MyBackendServiceImpl();
        // REGISTER MY_BACKEND_SERVICE
        context.registerService(MyBackendService.class.getName(), myBackendServiceImpl, null);
    }

    public void stop(BundleContext context) throws Exception {
        // MY_BACKEND_SERVICE WILL AUTOMATICALLY BE UNREGISTERED
    }
}
```

Gestion du cycle de vie du Bundle

Enregistrement du service

Dé-enregistrement implicite du service

Le manifest sert à déclarer le `Bundle-Activator`

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: My Service
Bundle-SymbolicName: com.mycompany.myservice
Bundle-Version: 1.0.0
Bundle-Activator: com.mycompany.myservice.Activator
Import-Package: javax.servlet;version="2.4.0",
 javax.servlet.http;version="2.4.0",
 org.osgi.framework;version="1.3.0",
 org.osgi.service.http;version="1.2.0",
 org.osgi.util.tracker;version="1.3.1"
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
Export-Package: com.mycompany.myservice
Require-Bundle: org.apache.log4j;bundle-version="1.2.13",
 com.mycompany.backend;bundle-version="1.0.0"
```

Déclaration de l'`Activator`

Assemblage des services OSGI
BundleActivator et ServiceTracker

Gestion du cycle de vie du Bundle

```
public class Activator implements BundleActivator {

    private ServiceTracker myBackendServiceTracker;

    public void start(final BundleContext context) throws Exception {

        // GET A TRACKER ON MY_BACKEND_SERVICE
        myBackendServiceTracker = new ServiceTracker(context, MyBackendService.class.getName(), null);
        myBackendServiceTracker.open();

        // REGISTER MY_SERVICE
        MyService myService = new MyServiceImpl(myBackendServiceTracker);
        context.registerService(MyService.class.getName(), myService, null);
     }

    public void stop(BundleContext context) throws Exception {
        myBackendServiceTracker.close();
        myBackendServiceTracker = null;
    }
}
```

Création du ServiceTracker

Injection du ServiceTracker

Fermeture du ServiceTracker

Assemblage des services OSGI
BundleActivator et ServiceTracker

```
public class MyServiceImpl implements MyService {

    private ServiceTracker myBackendServiceTracker;

    public MyServiceImpl(ServiceTracker myBackendServiceTracker) {
        super();
        this.myBackendServiceTracker = myBackendServiceTracker;
    }

    @Override
    public String sayHello(String message) {
        MyBackendService myBackendService = (MyBackendService)myBackendServiceTracker.getService();

        String result;
        if (myBackendService == null) {
            result = "Sorry '" + message + "', backend NOT available";
        } else {
            result = "Hello " + myBackendService.doJob(message) + " !";
        }
        return result;
    }
}
```

Injection du ServiceTracker

Résolution du service à chaque utilisation

Gestion de l'indisponibilité

Le ServiceTracker est intrusif sur le code !

Enregistrement d'une servlet sur le `HttpService`

Surcharge des méthodes `addingService(…)` et `removedService(…)`

```
public void start(final BundleContext context) throws Exception {

    // REGISTER MY_SERVICE_SERVLET TO THE HTTP_SERVICE
    httpServiceTracker = new ServiceTracker(context, HttpService.class.getName(), null) {
        public Object addingService(ServiceReference reference) {
            HttpService httpService = (HttpService)context.getService(reference);
            try {
                httpService.registerServlet("/my-service", new MyServiceServlet(null), null, null);
            } catch (Exception e) {
                e.printStackTrace();
            }
            return httpService;
        }

        public void removedService(ServiceReference reference, Object service) {
            HttpService httpService = (HttpService)service;
            httpService.unregister("/my-service");
            super.removedService(reference, service);
        }
    };
    httpServiceTracker.open();
```

ServiceTracker devient vite lourd à gérer !

Assemblage des services OSGI
BundleActivator et ServiceTracker

- **Mécanisme de gestion des dépendances le plus stable et mature d'OSGi**
- **Très en retard par rapport aux mécanismes d'injection de dépendances Java actuels (EJB 3, Spring, Google Juice, etc)**
- **Contraint par les limites des Execution Environments OSGI ?**

Assemblage des services OSGI
Declarative Service

- Créé en 2005 (OSGI R4)
- Mis à jour via la RFC 134

```
<?xml version="1.0" encoding="UTF-8"?>
<component name="myService">
    <implementation class="com.mycompany.myservice.impl.MyServiceImpl"/>
    <service>
        <provide interface="com.mycompany.myservice.impl.MyService"/>
    </service>
    <reference name="backendService"
        interface="com.mycompany.backend.MyBackendService"
        bind="setBackendService"
        unbind="unsetBackendService"/>
</component>
```

Assemblage des services OSGI
Declarative Service

- Semble peu utilisé
- Techniquement limité
  - Ne gère pas les dépendances intra-bundles
- Remis en cause par RFC 124 (Spring DM)

Assemblage des services OSGI
Spring DM

- En cours de standardisation pour OSGi R4.2
  - RFC 124

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:osgi="http://www.springframework.org/schema/osgi"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/osgi
http://www.springframework.org/schema/osgi/spring-osgi.xsd">

    <osgi:reference id="backendService"
        interface="com.mycompany.backend.MyBackendService" />

    <bean id="myService"
        class="com.mycompany.myservice.impl.MyServiceImpl">
        <property name="backendService" ref="backendService" />
    </bean>

</beans>
```

- **Exemples**
  - Log Service
  - Http Service
  - Configuration Admin Service
  - Preferences Service
  - Event Admin Service

- **Plus orientés embarqué / J2ME que informatique de gestion / Java 5+**
  - Fonctionnalités limitées
  - Ex: HttpService ne gère ni les Filter ni les ServletContextListener

La plateforme
Les implémentations Open Source

- Eclipse Equinox
- Apache Felix
- Knopflerfish (maintenu par Makewave)
- ProSyst Open Source mBedded Server Equinox Edition
- Newton Project

- La liste complète
  - http://www.osgi.org/Markets/HomePage

La plateforme
Les chantiers en cours

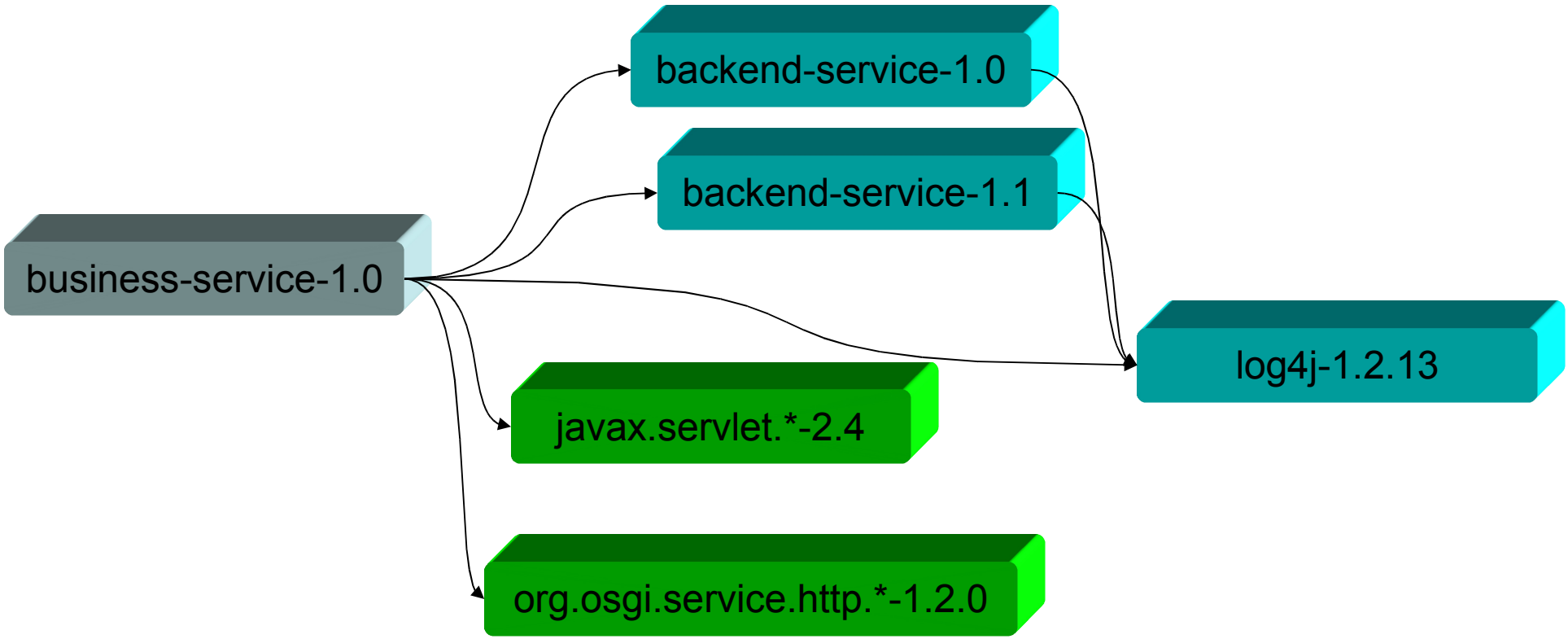
- **Spécifiés dans le draft sur OSGi R4.2**

- **Ajout de services « Java EE »**
  - RFC 98: gestion des transaction via JTA et/ou XA
  - RFC 119: OSGi distribué
    - » Appel de services distants
- **Nouveau modèle de composant**
  - En concurrence avec Declarative Service (OSGI R4 !)
  - RFC 124 – « A Component Model for OSGi »
    - » Standardisation de Spring DM

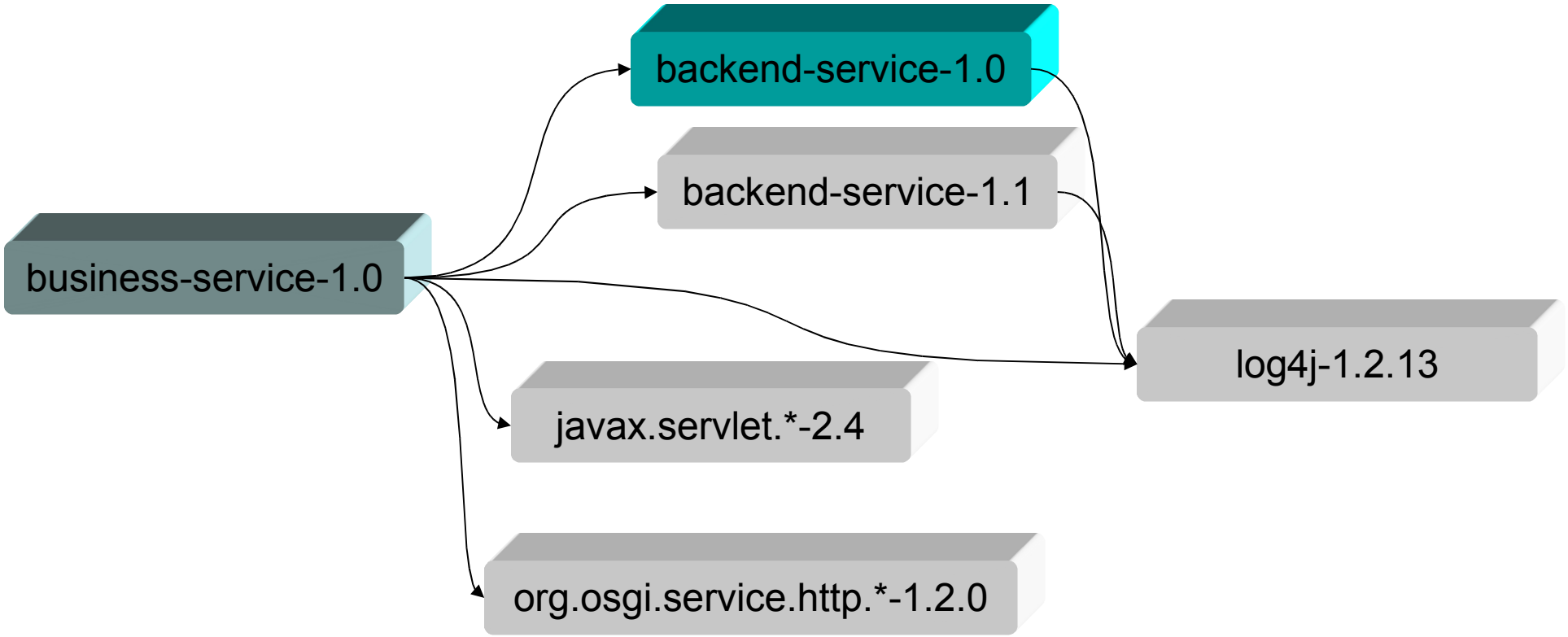
Demo OSGi
Description

- Un service dépend d'un backend
- Un service expose une servlet
- Les composants utilisent Log4j
- Upgrade à chaud du service

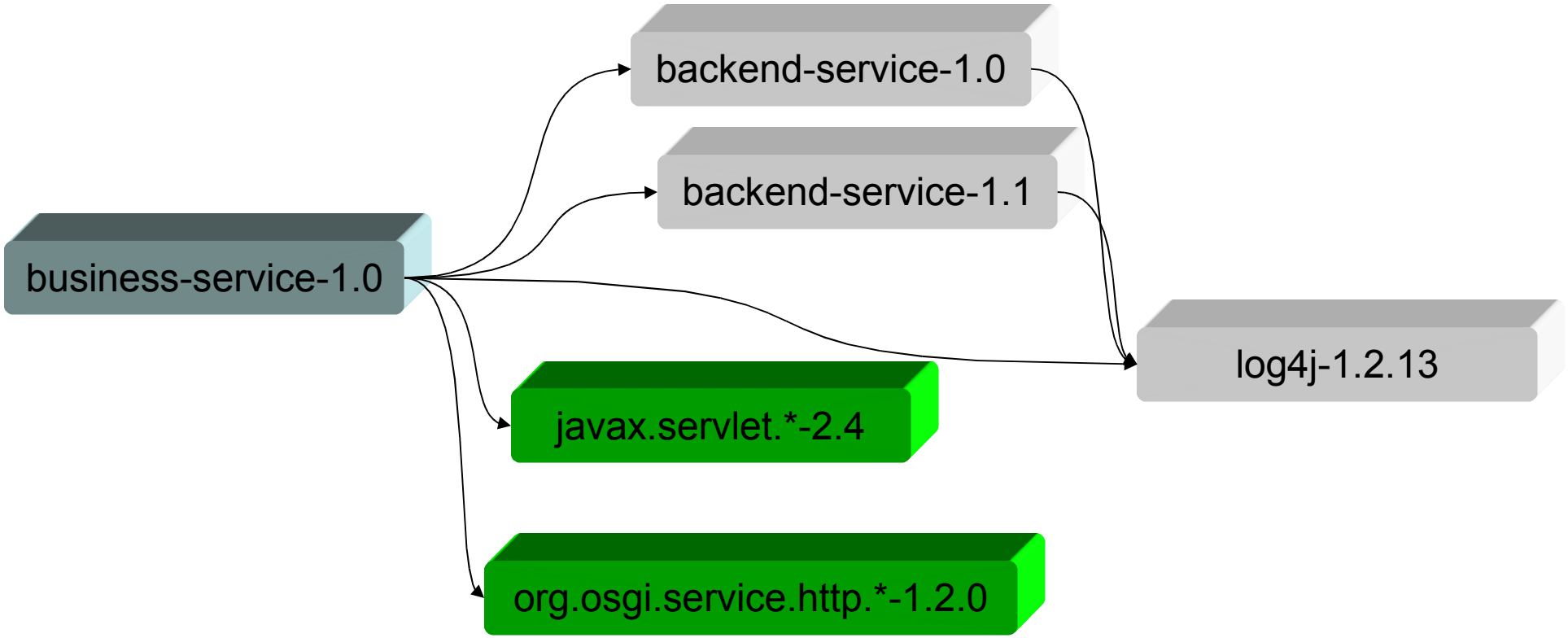
Demo OSGi
Bundles & packages



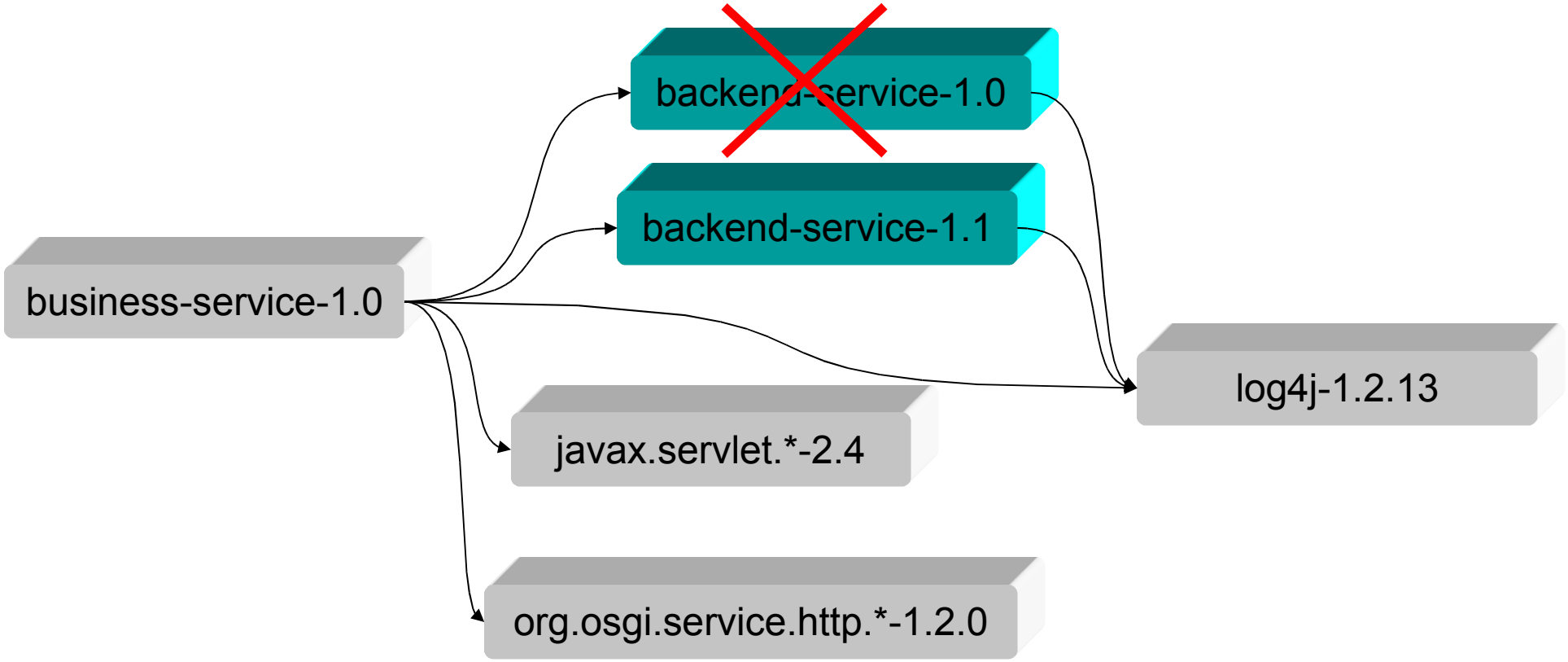
Demo OSGi
Import d'un service



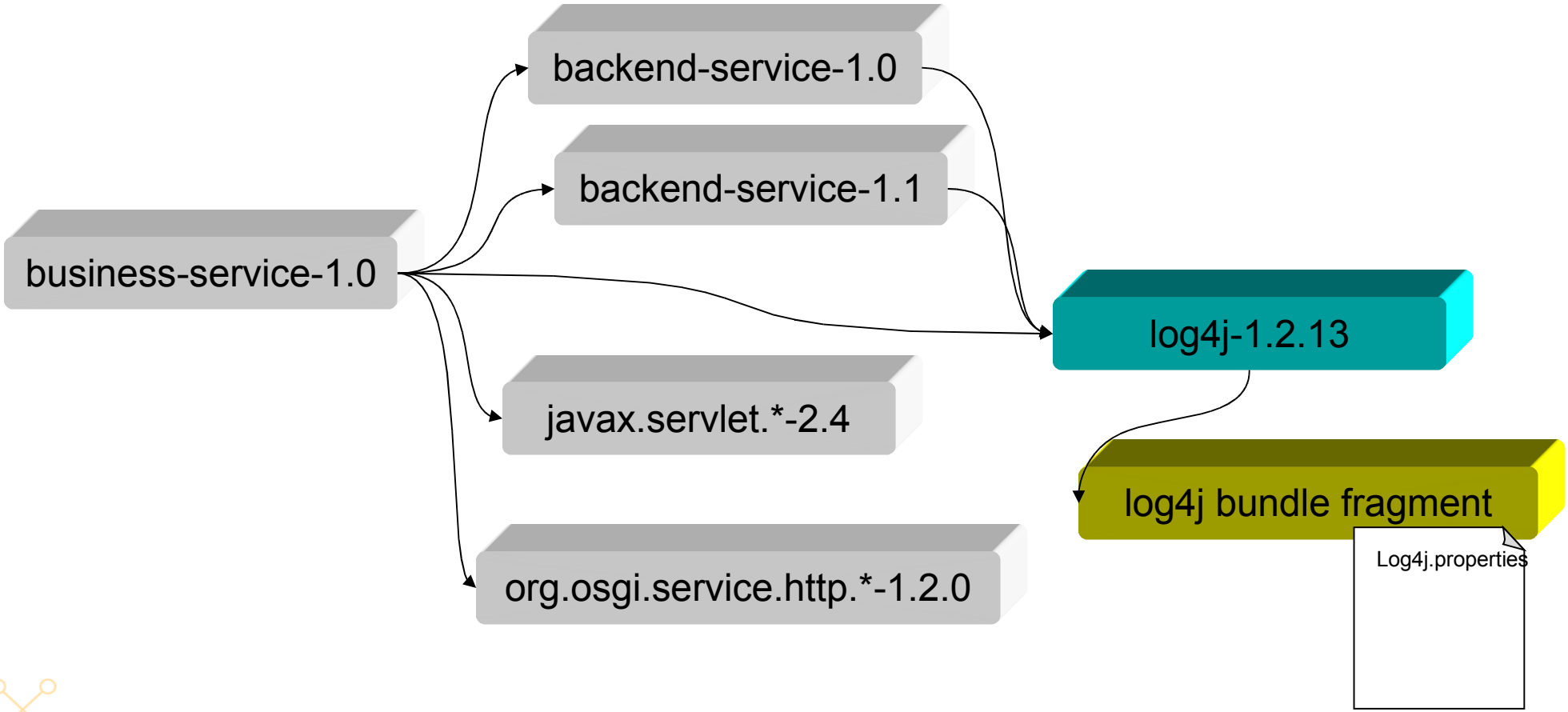
Demo OSGi
Exposition d'une servlet



Demo OSGi
Upgrade à chaud d'un bundle



Demo OSGi
Log4j !



OSGi dans le monde Java EE

Client side

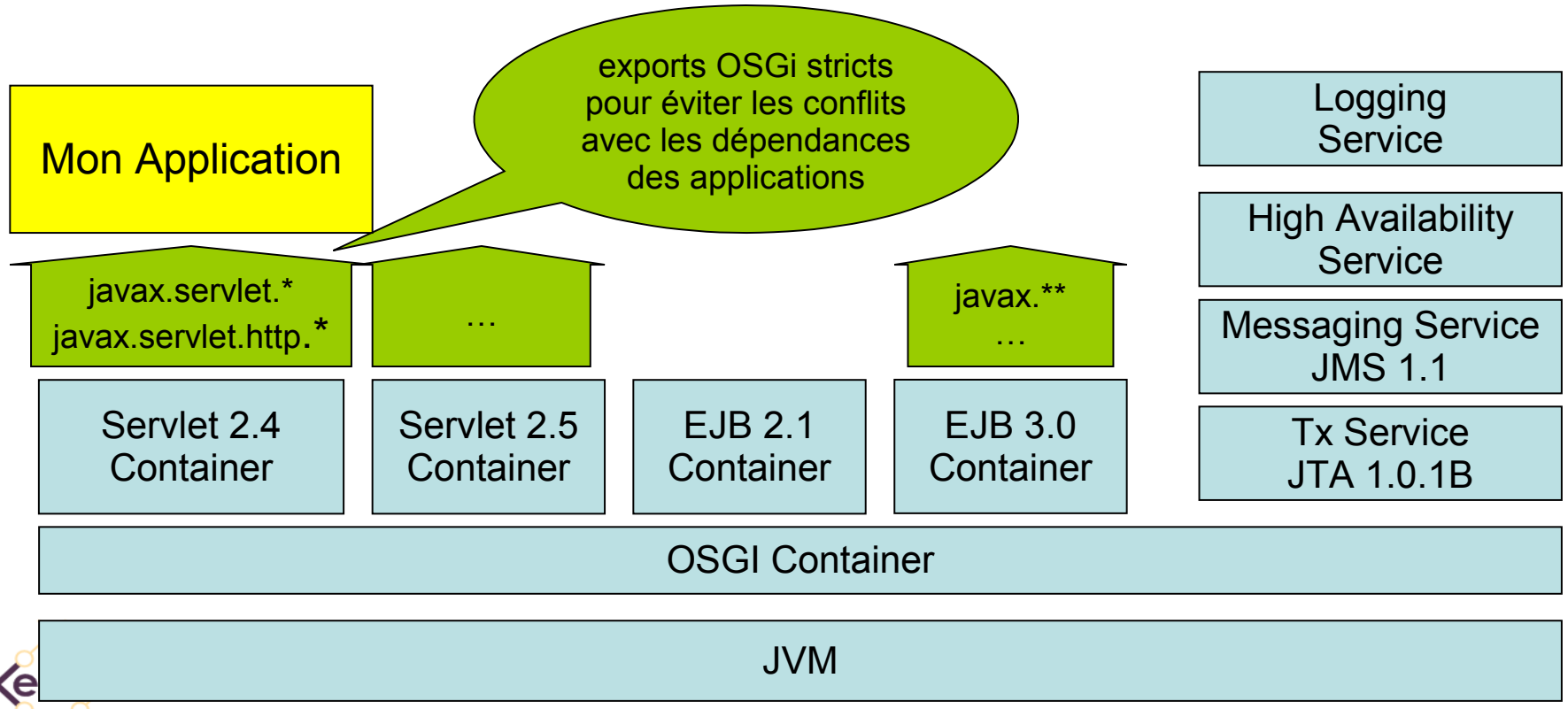
Server Side

- **Eclipse RCP**
  - 1ère utilisation d'OSGi dans le monde Open Source
  - Ajout de fonctionnalités propriétaires
    - `Eclipse-BuddyPolicy` et `Eclipse-RegisterBuddy`
    - Les Features
    - La gestion des updates
    - Les extensions
  - Limitations

    Cycle de vie des plugins : *On redémarre toujours Eclipse après avoir activé un plugin*

OSGi dans le monde Java EE
Server side – Middlewares

- **Isolation middleware / application**
- **Isolation des composants du middleware (multi versions)**
- **Lifecycle (start, stop, install, uninstall)**



OSGi dans le monde Java EE
Server side - Middlewares

- **Websphere 6.1 (2006) est full-osgi**
  - Classloader OSGI pour isoler les applications du middleware
  - Equinox est le conteneur de Websphere
  - Tous les jars sont osgi-fiés (repository IBM)
  - Compositions Application Server, ESB, Process Server, Portal, Telecom Server

- **Weblogic 10 (2007) utilise OSGI**
  - Plusieurs jars sont osgi-fiés

- **Glassfish**
  - Glassfish démarre sur Equinox et Knopflerfish

OSGi dans le monde Java EE
Server side - Middlewares

- **Service Mix 4**
  - Réécriture full OSGI / Spring DM de ServiceMix
  - Service Mix Kernel est un enrichissement d'un conteneur OSGI
  - Possibilité de déployer des médiations en packaging JBI ou OSGI

- **Spring DM**
  - Propose OSGI en alternative à Java EE

- **Autres acteurs Java EE : Jonas, JBoss**

Ces projets utilisent-ils OSGI basic ?
ServiceTracker ?

Bonnes pratiques OSGi
Les enjeux d'OSGi pour Java EE

Bonnes pratiques

- **Préférer `Import-Package` à `Require-Bundle`**
- **Utilisation des ranges de versions**
- **Bien designer ses bundles**
  - Prévoir le couplage de ses bundles
  - Ne pas tout mettre dans un bundle
- **Séparer les APIs exportées des implémentations**
- **Penser Orienté Service**
- **Gérer le cycle de vie des dépendances**
  - ServiceTracker, Spring DM, Declarative Service

Les enjeux d'OSGi pour Java EE

- **L'utilisation d'anti-pattern OSGi**
  - Class.forName / Factory Pattern
  - System.exit()

- **Réutilisation des libraries existantes non-OSGi**
  - Repackaging et prolifération de repositories
    OSGi.org, Spring, Servicemix, IBM (privé), etc
  - Modification du source code (http://www.dynamicjava.org/)
  - Bytecode patching au runtime (knoplerfish)

- **Gestion des meta données**
  - OSGi: Entrées dans le manifest
  - Java Module System: annotations et nouvelle syntaxe java

Les enjeux d'OSGi pour Java EE

- **Problématique cross-bundle**
  - AOP
  - Sécurité
  - Transaction
- **Testabilité des bundles**
- **Packaging des applications**
  - Archive .par chez Spring
  - Features chez eclipse

Conclusion

- Socle robuste et éprouvé
- Répond à des problématiques techniques complexes
- Manque de maturité dans le monde Java EE
- En pleine évolution pour les aspects Java EE

Questions

Sponsors

Merci de votre attention!

www.parisjug.org

Licence

http://creativecommons.org/licenses/by-nc-sa/2.0/fr/