La plateforme dynamique à services OSGi™

Gaël Thomas
gael.thomas@lip6.fr

Université Pierre et Marie Curie
Master Informatique
M2 – Spécialité SAR

Notion de Service

Service : entité informatique définie par un contrats

Contrat : ensemble de propriétés définissant l'accès à l'entité

Exemple :
- ✓ Une interface Java ou IDL (interface offerte ou interface requise)
- ✓ Le nombre de dpi pour une imprimante

Relation contractuelle : constitué de
- ✓ Contrats des services (des fournisseurs et des clients)
- ✓ Propriétés contractuelle

Exemple :
- ✓ Deux interfaces égales (interface serveur = interface du client)
- ✓ Pas de perte de message et 14 requêtes par seconde

Notion de Service

Résolution de contrat : construction de la relation contractuelle
- ✓ Mise en adéquation des contrats
- ✓ Définition des propriétés contractuelles

Annuaire : associe des contrats avec des services

⇒ Reconnaissance d'un service en fonction de ce qu'il sait faire
i.e : pas de dépendance vis à vis d'un nom indépendant

⇒ Mécanisme de recherche intuitif
"Cherche une imprimante couleur à l'étage"

```
Interface d'imprimante {
  void print(PostScript file)
}
```

```
Propriétés :
  • couleur = vrai
  • étage = mon-étage
```


Notion de Service

Trois intervenants

Fournisseur de service (serveur)
Imprimante, amazone.com

Utilisateur de service (client)
Utilisateur final, application à base de services, moteur d'orchestration

Un annuaire de services (Service Registry)
UDDI, pages jaunes, registry OSGi

Remarque : un service peut, bien sûr, être à la fois fournisseur et client

Notion de Service

Plateforme à service statique :

Résolution des contrats à la conception

- Impossible de sélectionner le fournisseur à l'exécution
+ Facile à programmer, client reste indépendant de l'implémentation

Plateforme à service dynamique

Résolution des contrats au chargement

- Impossible de changer de fournisseur en cours d'exécution
+ Facile à programmer

Plateforme à service dynamique et adaptable

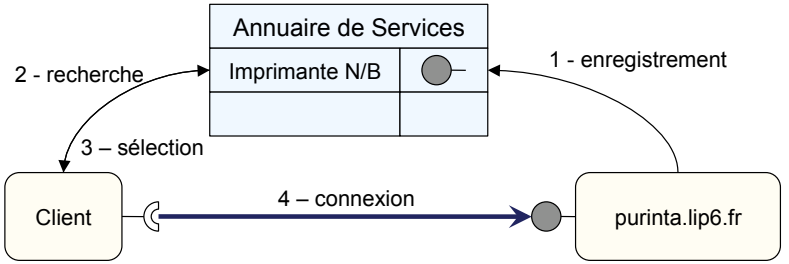
Résolution des contrats au chargement

+ Modification des contrats en fonction des départs et arrivées dynamique des services
- Difficile à programmer

Notion de Service

Architecture orientée service dynamique

Exemples : Web Services, OSGi™, Jini…

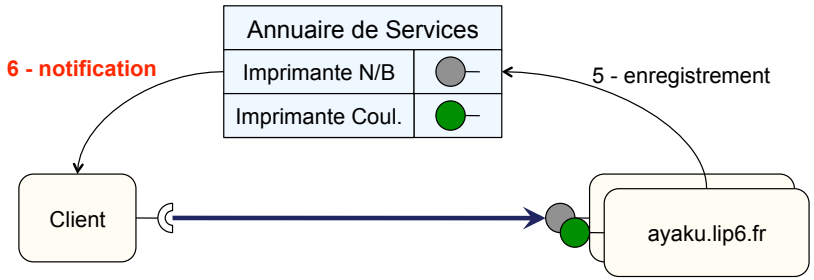


Notion de Service

Architecture orientée service dynamiquement adaptable

Adaptation du client au contexte

Exemples : OSGi™

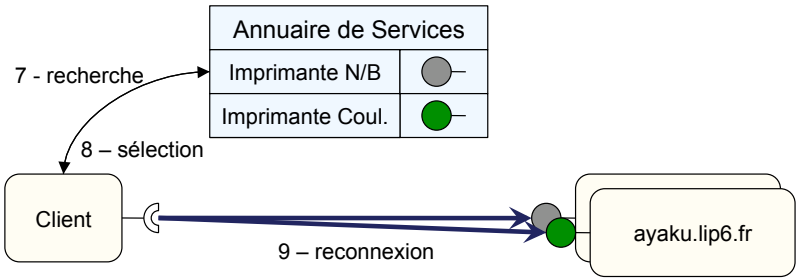


Notion de Service

Architecture orientée service dynamiquement adaptable

Adaptation du client au contexte

Exemples : OSGi™

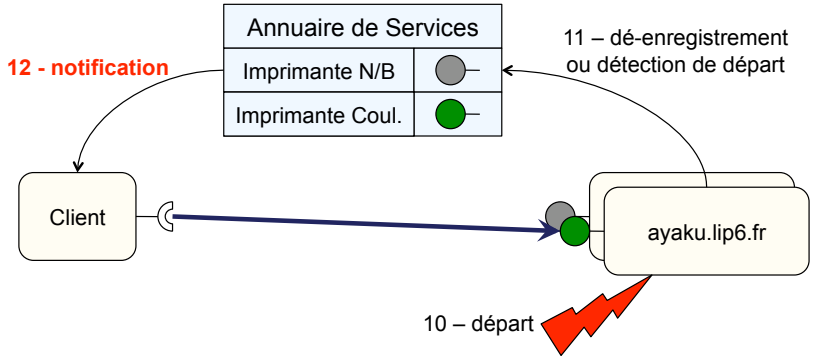


Notion de Service

Architecture orientée service dynamiquement adaptable

Adaptation du client au contexte

Exemples : OSGi™

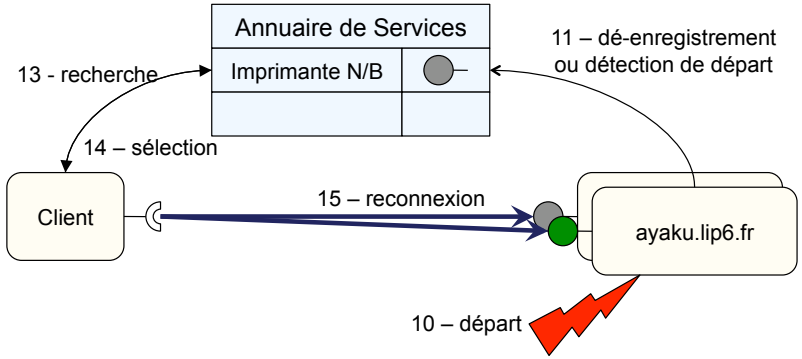


Notion de Service

Architecture orientée service dynamiquement adaptable

Adaptation du client au contexte

Exemples : OSGi™



Notion de Service

Séparation supplémentaire entre modules logiciels

Un utilisateur conçoit son application en fonction d'un service offert et non d'une implémentation précise du service

Par exemple : j'utilise un service météo au lieu de Météo France

La liaison avec une implémentation de service est effectuée

- ✓ Lors de la conception (programmation classique)
  Mais l'application reste indépendante de l'implémentation
- ✓ Lors de la compilation
- ✓ Pendant l'exécution

Les services augmentent

- ✓ L'indépendance entre modules logiciels
- ✓ L'adaptation dynamique

Notion de Service

Hypothèse forte

Un contrat ne change jamais (statiquement ou dynamiquement)

Modification d'un des contrats

⇒ renégociation de la relation contractuelle

La plateforme à service OSGi™

Plateforme à service

- ✓ Centralisé (une seule VM)
- ✓ Orienté au départ vers les Gateway
- ✓ Dynamiquement adaptable
- ✓ Liaison (de base) uniquement par appel de méthode
- ✓ Pas de couche ou de squelette

Corporation indépendante

- ✓ Fondée en mars 1999
- ✓ Soutenue par de nombreux acteurs (IT, téléphonie, Eclipse, Apache)

Remarque : Open Services Gateway Initiative est obsolète

La plateforme à service OSGi™

Trois principales implantations de la spécification

- ✓ Felix (Apache)
- ✓ Eclipse-equinox (IBM)
  (Moteur à pluggin d'eclipse)
- ✓ Knopflerfish

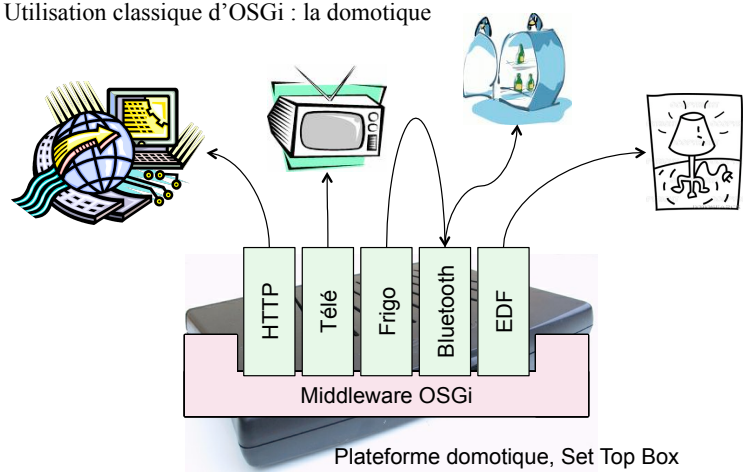
La plateforme à service OSGi™

Motivation

- ✓ Programmation orientée service en Java
  Surcouche logicielle (middleware) au dessus des JVM
- ✓ **Cible des systèmes à mémoire restreinte**
  Compatible Java Micro Edition
- ✓ **Mise à jour dynamique d'application sans interruption**
  Notification d'arrivée et de départ de service
- ✓ Installation, mise à jour, lancement, arrêt, retrait
  Cycle de vie des applications pris en charge par le middleware
- ✓ Résolution des dépendances de version de code
  Plusieurs versions de la même classe peuvent cohabiter simultanément
- ✓ Chargement/déchargement de code en Java
  Surtout déchargement!
- ✓ **Communication rapide entre services**
  Pas de souche ou de squelette entre le fournisseur et le client (⇒ centralisé)

La plateforme à service OSGi™

Utilisation classique d'OSGi : la domotique



Plateforme domotique, Set Top Box
Modem…

La plateforme à service OSGi™

OSGi : deux parties complémentaires

Partie déploiement et cycle de vie
- ✓ Chargement de nouvelles applications dans la JVM
- ✓ Résolution des dépendances de classes
- ✓ Déchargement et mise à jour des applications

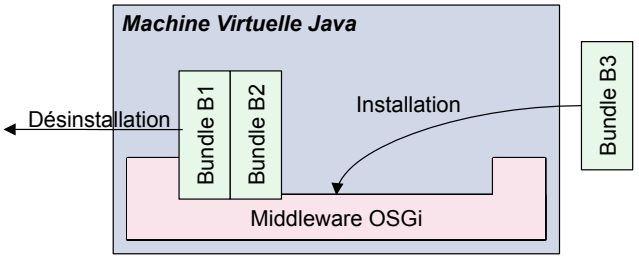
Partie plateforme à service
- ✓ Création, destruction de services
- ✓ Annuaire à service
- ✓ Notification de changement d'état des services

Déploiement et cycle de vie

Bundle

Unité fonctionnelle fondamentale de OSGi™
- ✓ Offre des services
- ✓ Se lie (utilise) des services

Un bundle est l'unité de déploiement d'OSGi™



Déploiement et cycle de vie

Présentation d'un bundle

Un bundle est un fichier .jar contenant
- ✓ Un **manifest** : la description du bundle
- ✓ Des classes Java : le code du bundle
- ✓ D'autres fichiers .jar : des bibliothèques internes
- ✓ Des ressources : des images ou autre

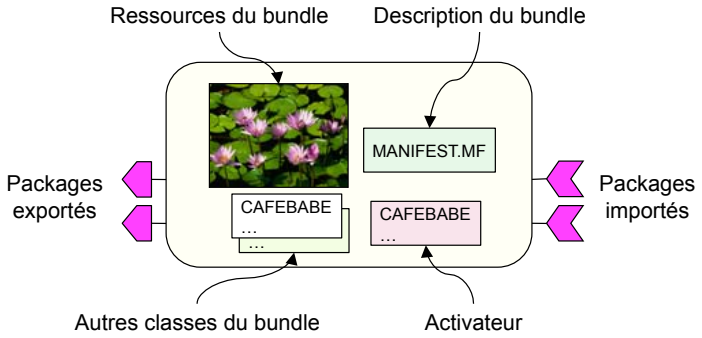
Description du bundle (le manifest) :
- ✓ La classe **Activateur** : invoquée lors du démarrage ou de l'arrêt du bundle
- ✓ Les packages requis et les packages offerts
- ✓ Des informations supplémentaires

Mais où sont passés les services?!

Instanciés, trouvés ou enregistrés par l'activateur

Déploiement et cycle de vie

Chargement dans la machine virtuelle Java :
- ✓ Instanciation de l'activateur
- ✓ Invocation de la méthode start() dessus



Déploiement et cycle de vie

Sept actions sur les bundles

Installe le bundle
Charge le bundle dans le système de fichier

Résout le bundle
Charge le bundle dans la machine virtuelle

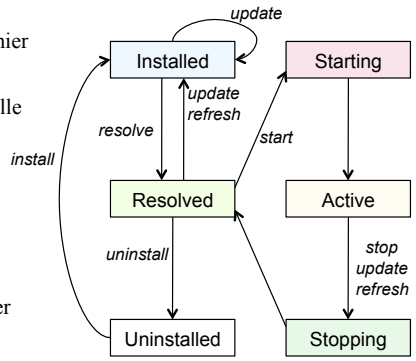
Démarre le bundle
Appel la méthode start() de l'activateur

Stop le bundle
Appel la méthode stop() de l'activteur

Désinstalle le bundle
Supprime le bundle du système de fichier

Met à jour le bundle
Ré-installe le bundle

Rafraîchie le bundle
Reconstruit toutes les liaisons du bundle



Déploiement et cycle de vie

Le descripteur de déploiement : MANIFEST.MF

- ✓ Bundle-Name : nom du bundle
- ✓ Bundle-Description : chaîne de caractère pour décrire le bundle

- ✓ Import-Package : packages importés (au chargement)
- ✓ Dynamic-Import-Package : package importés (au fur et à mesure des besoins)
- ✓ Export-Package : packages exportés
- ✓ Bundle-Activator : classe de l'activateur

- ✓ Bundle-ClassPath : à l'intérieur de Bundle
- ✓ Bundle-UpdateLocation : url pour mise à jour du bundle

- ✓ Bundle-Version : la version du Bunlde
- ✓ Bundle-Vendor : le vendeur du bundle
- ✓ Bundle-ContactAddress : adresse mail de contact
- ✓ Bundle-Copyright : chaîne de caractère

Déploiement et cycle de vie

Exemple de bundle : l'activateur

```
package lip6.simple;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;

public class Test implements BundleActivator {
  public void start(BundleContext context) {
    System.out.println("Hello!");
  }

  public void stop(BundleContext context) {
    System.out.println("Goodbye!");
  } }
```


Déploiement et cycle de vie

Exemple de bundle : le fichier manifest.mf

```
Bundle-Name: A simple bundle
Bundle-Description: A simple bundle to test deployment
Bundle-Vendor: UPMC
Bundle-Version: 1.0.0
Bundle-Activator: lip6.simple.Test
Import-Package: org.osgi.framework
```

Si on exporte le package lip6.simple :

```
Export-Package: lip6.simple
```

Déploiement et cycle de vie

Le packaging (sources disponibles sur le site du cours : simple.zip)

Bundle : fichier jar avec le manifest OSGi
Fichier jar = fichier zip!

Arborescence classique d'un bundle

`/`	racine du jar
`/un-autre-jar.jar`	des sous-jars (voir Bundle-ClassPath)
`/une-ressource.avi`	des ressources annexes
`/lip6/simple/Test.class`	notre activateur
`/META-INF/MANIFEST.MF`	le manifest du bundle

Par défaut : Bundle-ClassPath = /

Déploiement et cycle de vie

Le déploiement

Installer felix : `http://felix.apache.org/`
Lancer felix : java –jar $FELIX_PATH/bin/felix.jar

Profile felix : un ensemble de bundles installés, plusieurs profiles possibles
Voir répertoire ~/.felix/nom-du-profile
Pour voir les bundles installés dans le profile : ps (Apache Felix Shell Service : le terminal felix)

Installer le bundle : install file:///chemin_complet_du_bundle/simple.jar
⇒ copie le .jar dans le profile et lui attribue un BundleID

Démarrer/arrêter le bundle : start BundleID/stop BundleID

Mettre à jour le bundle : update BundleID

Déploiement et cycle de vie

Bundle = application Java

OSGi = serveur d'application

Problème de liaison :

- ✓ bundle compliqué : ImportPackage lip6.simple
  class Test2 extends lip6.simple.Test { … }
  update du bundle simple!

- ✓ Solution : refresh
  Update de tous les bundles dont dépendent les bundles mis à jour

Services OSGi™

Service OSGi : le contrat

Fournisseur : offre un service
- ✓ Défini par une interface Java
- ✓ Possède des propriétés : couple clé/valeur

Client : cherche un service
- ✓ Défini par une interface Java
- ✓ Filtre sur les propriétés
- ✓ Sélectionne un service qui correspond
- ✓ **Appel direct sur l'instance du service**

Annuaire OSGi
- ✓ Associe des contrats et des instances de service
- ✓ Renvoie des références OSGi vers les services (ServiceReference)

Interface du contrat = interface Java

⇒ fichier de classe, déployé dans un bundle

Règle : éviter de dupliquer le code de l'interface

i.e. ne pas la charger dans deux bundles différents

⇒ L'interface d'un contrat doit être packagée dans un bundle indépendant (ni le fournisseur, ni le client)

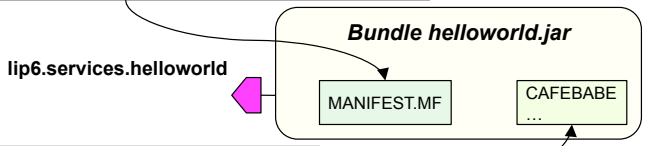
Bonne programmation OSGi ⇒ trois classes de bundle

- ✓ **Les bundles de contrats** : contiennent des interfaces + exceptions associées
- ✓ **Les bundles bibliothèques** : contiennent des classes fournies par des tiers
- ✓ **Les bundles de services** : contiennent des services (fournisseurs et/ou clients)

Pas requis par la spécification, juste une bonne manière de programmer ☺

Le bundle contrat typique

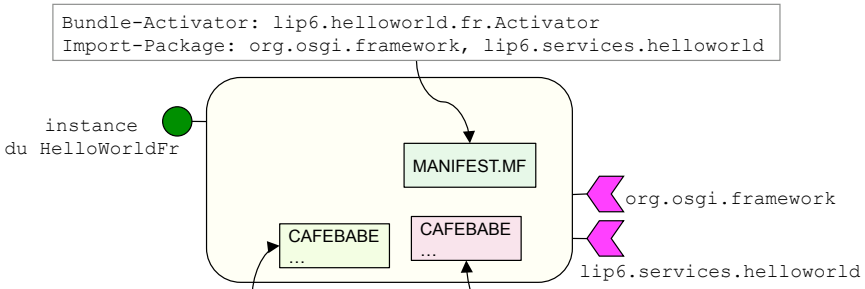
```
Bundle-Name: HelloWorld interface
Bundle-Description: The HelloWorld interface
Bundle-Vendor: UPMC
Bundle-Version: 1.0.0
Export-Package: lip6.services.helloworld
```




```
package lip6.services.helloworld;
public interface HelloWorld {
       public void sayHello();
}
```


Le bundle fournisseur typique

```
Bundle-Activator: lip6.helloworld.fr.Activator
Import-Package: org.osgi.framework, lip6.services.helloworld
```




```
class HelloWorldFr
        implements HelloWorld
```

```
props.put("Lang", "Fr");
context.registerService(
  HelloWorld.class.getName(), new HelloWorldFr(), props);
```


Le fournisseur de service HelloWorldFr

```
package lip6.helloworld.fr;
public class HelloWorldFR implements HelloWorld {
   public void sayHello() {
     System.out.println("Bonjour, monde!"); } }
```

Manifest du fournisseur

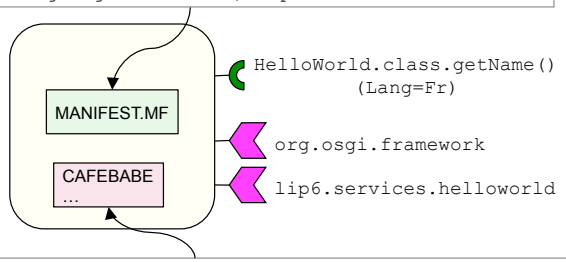
```
Bundle-Name: The french HelloWorld
Bundle-Description: The french implementation
Bundle-Vendor: UPMC
Bundle-Version: 1.0.0
Bundle-Activator: lip6.helloworld.fr.Activator
Import-Package: org.osgi.framework, lip6.services.helloworld
```

L'activateur du bundle

```
package lip6.helloworld.fr;

public class Activator implements BundleActivator {
  public void start(BundleContext context) {
    System.out.println("Démarre le helloworld français");
    HelloWorld hw = new HelloWorldFR();
    Properties props = new Properties();
    props.put("Lang", "Fr");
    context.registerService(
    HelloWorld.class.getName(), hw, props);
  }
  public void stop(BundleContext context) {
    System.out.println("Stop le helloworld français");
  } }
```


Le bundle client typique

```
Bundle-Activator: lip6.helloworld.client.v0.Activator
Import-Package: org.osgi.framework, lip6.services.helloworld
```




```
ServiceReference[] refs =
      context.getServiceReferences(
                        HelloWorld.class.getName(), "(Lang=Fr)");
HelloWorld hw = (HelloWorld)context.getService(refs[0]);
```


Activateur du client

```
package lip6.helloworld.client.v0;

public class Activator implements BundleActivator {
  public void start(BundleContext context) throws  … {
    ServiceReference[] refs =
      context.getServiceReferences(
        HelloWorld.class.getName(), "(Lang=Fr)");
    if(refs == null) return;
    HelloWorld hw = (HelloWorld)context.getService(refs[0]);
    hw.sayHello();
    context.ungetService(refs[0]); } }

  public void stop(BundleContext context) { } }
```


Manifest du client

```
Bundle-Name: The simple HelloWorld client
Bundle-Description: A simple client for the helloworld service
Bundle-Vendor: UPMC
Bundle-Version: 1.0.0
Bundle-Activator: lip6.helloworld.client.v0.Activator
Import-Package: org.osgi.framework, lip6.services.helloworld
```

Services OSGi™

Propriétés et filtres de recherche

expressions LDAP classiques

- ✓ Opérateurs de compositions : &, |, !…
- ✓ Opérateurs de regroupement : ()
- ✓ Opérateurs d'égalité, inégalité : =, <, >…
- ✓ **Clé spéciale objectClass : l'interface du service (String)**

Exemple

- ✓ getServiceReference(HelloWorld.class.getName(), "|(Lang=Fr)(cpt<18)");
  Tous les services de type HelloWorld ayant Lang=fr ou cpt<18
- ✓ getServiceReferences(null, "&(objectClass=*)(Lang=Fr));
  Tous les services (tous type) ayant Lang=Fr

Services OSGi™

Construction d'une application OSGi

Bundle contrat :

- ✓ Manifest : pas d'activateur + export l'interface du service
- ✓ Code : une interface de service + classes d'exception

Bundle fournisseur

- ✓ Manifest : activateur + importe la plateforme + importe l'interface du service
- ✓ Activateur : alloue et enregistre une instance du service
- ✓ Code : le code de l'implémentation du service

Bundle client

- ✓ Manifest : activateur + importe la plateforme + importe l'interface du service
- ✓ Activateur : cherche, sélectionne et utilise un founrisseur

Liaison dynamique

Liaison dynamique = sélection à l'exécution du service

- ✓ Attend d'avoir un service disponible avant d'enregistrer le sien
- ✓ Choix d'un meilleurs service lorsqu'il apparaît
- ✓ Prise en compte de la disparition d'un service

Change fondamentalement la façon de programmer

Une référence Java n'est pas toujours disponible!

Intérêt :

- ✓ Mise à jour dynamique de morceaux d'application
  (Plugin Eclipse par exemple)
- ✓ Adaptation dynamique à l'environnement
  (Maison ubiquitaire par exemple)

Liaison dynamique

Événements OSGi

- ✓ Départ ou arrivée d'un bundle (interface BundleListener)
- ✓ **Départ ou arrivée d'un service (interface ServiceListener)**
- ✓ Événement sur la plateforme (interface FrameworkListener)
- ✓ Autres événements (LogListener, CinfigurationListener, MonitorListener…)

Schéma de fonctionnement général

- ✓ **Recherche initiale de service**
  Si présent, utilise le service le plus adéquat
  Sinon, attend un service
- ✓ **Arrivée/départ de service ⇒ génération d'un événement ServiceListener**
  Si départ de S et si S était utilisé, recommence une recherche initiale
  Si arrivée de S et si S plus intéressent que service courant, change pour S
  Si événement modification du service, s'assure que le service est toujours adéquat

Liaison dynamique

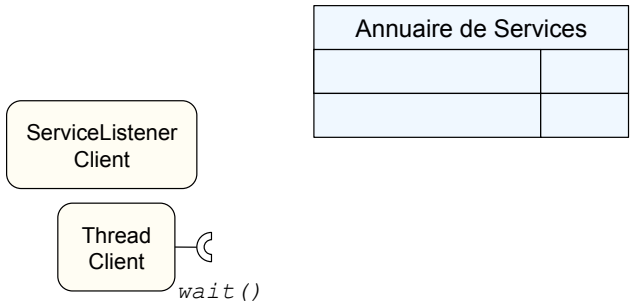
Le ServiceListener : être informée de l'arrivée/départ d'un service HelloWorld

```
public class Ecoute implements ServiceListener {
  public void serviceChanged(ServiceEvent ev) {
    ServiceReference ref = event.getServiceReference();
    String objectClasses[] =
      (String[])ref.getProperty("objectClass");

    if(objectClasses[0].equals(HelloWorld.class.getName()) {
      switch(ev.getType()) {
        case MODIFIED:      /* props ont changées */
        case REGISTERED:    /* arrivée d'un fournisseur */
        case UNREGISTERING: /* départ d'un fournisseur  */
      } } } }

Activator : context.addServiceListener(new Ecoute(), filtre));
```


Liaison dynamique

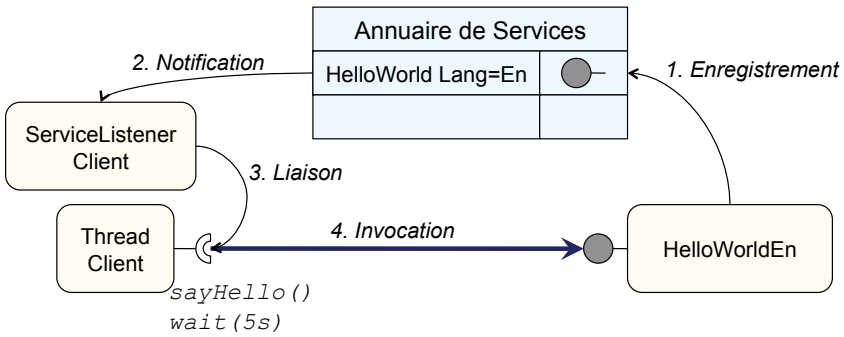
Exemple : un client évolué pour HelloWorld

- ✓ Thread invoque le service toutes les 5s
- ✓ Service préféré en Lang=Fr, si absent, prend le premier
- ✓ Dort si aucun service disponible



Liaison dynamique

Exemple : arrivée d'un HelloWorld avec Lang=En



Liaison dynamique

Exemple (1/8) : variables du client

```
package lip6.helloworld.client;
public class Activator implements
        BundleActivator, Runnable, ServiceListener {
  // gestion du thread : stop indique la terminaison
  private boolean    stop;
  private Thread     self;

  // contexte associé au Bundle
  private BundleContext context;

  // référence OSGi vers le service courant (null ⇒ absent)
  private ServiceReference ref;
  // référence Java vers le service courant
  private HelloWorld hw;
```

Liaison dynamique

Exemple (2/8) : gestion du thread

```
public void run() {
  try {
    synchronized(this) {
      while(!stop) {
        if(ref == null)      // pas de service dispo ⇒ dort
          wait();
        else {
          hw.sayHello();     // invoque le service
          wait(5000);        // dort 5s
        } } }
  } catch(Exception e) { }
}
```


Liaison dynamique

Exemple (3/8) : liaison/dé-liaison avec le service (synchronisation à l'extérieur)

```
  private void useService(ServiceReference ref,
                          ServiceReference old) {
    this.ref = ref;
    hw = (HelloWorld)context.getService(ref);
    if(old == null)
      notify(); } // réveille le thread oisif

  private ServiceReference unuseService() {
    if(ref == null) return null;
    ServiceReference old = ref;
    context.ungetService(ref);
    ref = null;
    return old; }
```


Liaison dynamique

Exemple (4/8) : recherche initiale

```
public void lookup() throws … {
  ServiceReference old = unuseService(); // lâche ref actuelle
  ServiceReference[] refs = // recherche tous les HelloWorld
      context.getServiceReferences(
          HelloWorld.class.getName(), null);
  if(refs == null) return; // aucun service trouvé…

  // cherche en premier un service Lang=Fr
  for(int i=0; i<refs.length && ref==null; i++)
    if(refs[i].getProperty("Lang").equals("Fr"))
      useService(refs[i], old);
  // sinon, le premier fait l'affaire
  if(ref == null) useService(refs[0], old); }
```


Liaison dynamique

Exemple (5/8) : démarrage du bundle

```
public void start(BundleContext context) throws  … {
  this.context = context;

  // gestion thread
  self = new Thread(this, "The simple HelloWorld client");
  self.start();

  synchronized(this) { // évite événement pendant start
    context.addServiceListener(  // enregistre
            this,         // le gestionnaire d'événement this
            "(&(objectClass=" + HelloWorld.class.getName()
               + ")" + "(Lang=*))");
    lookup(); } }  // recherche initiale
```

Liaison dynamique

Exemple (6/8) : fin du bundle

```
public void stop(BundleContext context) throws … {
  synchronized(this) {
    unuseService(); // lâche le service
    stop = true;    // indique au thread « fini »
    notify();       // réveille immédiat du thread
  }
  self.join();      // attend terminaison
}
```


Liaison dynamique

Exemple (7/8) : la gestion des événements (1/2)

```
public void serviceChanged(ServiceEvent ev) {
  ServiceReference newRef = ev.getServiceReference();

  synchronized(this) { // à cause du thread
    switch(ev.getType()) {
      case ServiceEvent.REGISTERED: // si arrivée de service
        if(ref == null)
          useService(newRef, ref);
        else if(newRef.getProperty("Lang").equals("Fr") &&
                !ref.getProperty("Lang").equals("Fr")) {
          // un Fr arrive et on utilise un !Fr
          useService(newRef, unuseService());
        }
        break;
```


Liaison dynamique

Exemple (8/8) : la gestion des événements (2/2)

```
      case ServiceEvent.UNREGISTERING: // si désenregistrement
        if(newRef == ref) // du service utilisé
          lookup();  // relance une recherche initiale
        break;
      case ServiceEvent.MODIFIED: // si modification props
        if((newRef == ref) &&     // du service utilisé
           !ref.getProperty("Lang").equals("Fr")) // !Fr
          lookup();  // relance une recherche initiale
        break;
    }
  }
}
```


Conclusion

OSGi : modèle de plateforme à service locale en Java

Points positifs :

- ✓ Très performant : pas de mandataire entre serveur et client
- ✓ Uniquement local : de nombreux travaux existent autour de la répartition
- ✓ Uniquement en Java pour Java : contribue à obtenir de bonnes performances!
- ✓ Technologie en pleine expansion

Points négatifs :

- ✓ Programmation dynamique difficile à prendre en main…
- ✓ Déchargement de classe difficile (voir cours chargeur de classes)
- ✓ Encore peu d'outils d'ingénierie logiciel