Cours 2.1

Premiers pas avec OSGi (suite – Les Trackers)
La plate-forme dynamique orientée composants

Bachir Djafri

1

Déjà vu sur le modèle OSGi

- OSGi est une modèle de composants dynamiques orienté services
  - Basé sur Java (implémentation avec Java)
  - Non distribué, mais gestion distante possible (installation, désinstallation, etc.)
- Les points positifs
  - C'est une technologie en pleine expansion soutenue par une communauté active
  - Ce modèle fournit un comportement dynamique simple
  - Il fournit une solution aux problèmes de classpath et à la gestion de versions des composants
  - De nombreux bundles fournissent des fonctionnalités avancées
- Les points négatifs (à ce stade)
  - Comme pour toutes ces technologies nouvelles : le manque d'outils d'ingénierie logicielle
  - Le déchargement de classes pas toujours facile
  - L'écoute des évènements non plus n'est pas facile (*si l'on s'en tient à ce qu'on a vu jusque là !)*

2

… et aussi

- Notion de ServiceListener
  - *introduite avec la release 1 (une) d'OSGi*
  - Enregistrement de services et d'écouteurs d'événements auprès du contexte
  - Le développeur doit, dans la classe d'activation (méthode start) :
    - Créer et enregistrer les services qu'il fournit
    - Créer et enregistrer les écouteurs d'événements des services qu'il requiert pour fonctionner (services requis via interfaces requises)
    - Implémenter au sein des écouteurs d'événements la manière dont le composant doit réagir aux événements perçus (reçus)
- Deux autres notions introduites depuis
  - Release 2 – ServiceTracker
  - Release 4 – Declarative Services (Cours OSGI avancé)

3

Service Tracker

- Un Service Tracker (traqueur de service) est un mécanisme pour traquer l'activation, la désactivation et la modification de services vérifiant un contrat donné (type de service)
  - Contrairement au ServiceListener, un ServiceTracker est associé à un seul type de service
- Un Service Tracker peut être customisé (personnalisé) :
  - pour ne traquer que des services spécifiques d'un type donnée
  - Effectuer des actions dédiées lors de l'activation, la désactivation ou la modification de l'un de ces services
- Il peut mettre à jour l'état des bundles en fonction des évènements qu'il observe
- Techniquement, cela induit à :
  - Modifier les méthodes **start()** et **stop()** de la classe d'activation du bundle
  - Définir des objets de type **ServiceTrackerCustomizer** qui ajoutent des fonctionnalités de gestion de cycle de vie des bundles au tracker standard.
- Voir https://osgi.org/javadoc/r6/core/index.html

4

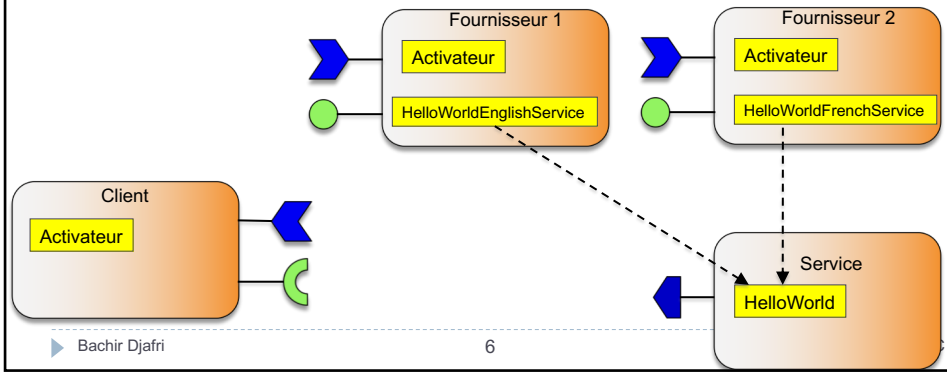
Service Tracker

- ServiceTracker (traqueur de service) = objet d'une classe qui implémente l'interface ServiceTrackerCustomizer<S,T>
  - S : type du service traqué
  - T : type de l'objet de service traqué (peut être différent de S)
- Classe ServiceTracker<S,T> :
  - une implémentation de l'interface ServiceTrackerCustomizer
  - implémente les méthodes de l'interfaces avec des corps vides qu'on peut redéfinir;
  - simplifie l'utilisation des traqueurs de services en offrant des opérations pour ouvrir (lancer) et fermer (arrêter) les traqueurs + d'autres opérations
  - peut être customisée avec un traqueur de service (passé en paramètre à la construction) ou en redéfinissant les 3 méthodes de l'interface.
- Besoin du package org.osgi.util.tracker

5

Service Tracker : Reprise du SmartCustomer

- Contexte
  - Un seul service, un client de ce service et deux fournisseurs de ce service
  - Mais le client
    - préfère utiliser le service français
    - Peut démarrer sans fournisseur de ce service (service optionnel)
    - Choisit le service anglais si celui-ci est le seul disponible



6

Enregistrement des ServiceTracker

- Pour chaque service requis par un composant (bundle) B
  - On enregistre dans la classe d'activation de B, le ServiceTracker correspondant
  - On ouvre le tracker pour qu'il commence à travailler (à traquer les services)

```
14
15  public class Activateur implements BundleActivator, ServiceTrackerCustomizer<HelloWorld, HelloWorld> {
16
17      private BundleContext context;
18      private HelloWorld service;
19      private ServiceReference<HelloWorld> ref;
20      private ServiceTracker<HelloWorld, HelloWorld> tracker;
21
22      public void start(BundleContext bundleContext) throws Exception {
23          context = bundleContext;
24          ref = null;
25          service = null;
26          tracker = new ServiceTracker<>(context, HelloWorld.class, this);
27          tracker.open();
28          chercherService();
29      }
30
31      public void stop(BundleContext bundleContext) throws Exception {
32          if (service != null) {
33              service.goodbye();
34              service = null;
35              context.ungetService(ref);
36              ref = null;
37              tracker.close();
38              tracker = null;
39          }
40          context = null;
41      }
```

objet gérant la réception des services traqués

Contexte dans lequel traquer les services

Type des services à traquer ou filtre sur les services à traquer

Gestionnaire en cas de service trouvé

7

Enregistrement des ServiceTracker

- Pour chaque service requis par un composant (bundle) B
  - on ferme le tracker avant de quitter le composant (stop dans le bundle)
- Lorsqu'un service S est traqué, trois actions sont possibles
  - S vient d'être activé, enregistré (addingService)
  - S vient d'être désactivé, désenregistré (removedService)
  - S vient d'être modifié (modifiedService)
- Le ServiceTrackerCustomizer prend en compte ces 3 évènements
- Sans ServiceTrackerCustomizer, c'est le ServiceTracker qui les prendrait directement en charge en redéfinissant les 3 opérations (héritage)
  - Un ServiceTracker est un cas particulier de ServiceTrackerCustomizer dont les 3 opérations sont vides (corps vide).

8

Définition d'un ServiceTrackerCustomizer

- Un ServiceTrackerCustomizer est appelé lorsque le ServiceTracker associé est tenu au courant d'un événement sur un service qu'il traque

```
6  import org.osgi.framework.BundleContext;
7  import org.osgi.framework.InvalidSyntaxException;
8  import org.osgi.framework.ServiceReference;
9  import org.osgi.util.tracker.ServiceTrackerCustomizer;
10
11 import exemple.contrat.HelloWorld;
12
13 public class HelloWorldTraqueurService implements ServiceTrackerCustomizer<HelloWorld, HelloWorld> {
14     private ServiceReference<HelloWorld> ref;
15     private BundleContext context;
16     private HelloWorld service;
17
18     public HelloWorldTraqueurService(BundleContext bc, ServiceReference<HelloWorld> sr, HelloWorld s) {
19         context = bc; ref = sr; service = s;
20     }
21
22     public HelloWorld addingService(ServiceReference<HelloWorld> sr) {
38
39     public void modifiedService(ServiceReference<HelloWorld> sr, HelloWorld service) {
45
46     public void removedService(ServiceReference<HelloWorld> sr, HelloWorld service) {
55
56     private void chercherService() {
74 }
75
```


Définition d'un ServiceTrackerCustomizer

- Un ServiceTrackerCustomizer est appelé quand le ServiceTracker associé est tenu au courant d'un événement sur un service qu'il traque
- Activation du service (enregistrement d'un nouveau service)
  - Connaissance de la référence sur le service en question (sr)

```
22     public HelloWorld addingService(ServiceReference<HelloWorld> sr) {
23         System.out.println("Nouveau service : (Langue = " + sr.getProperty("Langue") + ")");
24         if (this.ref == null) { // pas encore de service
25             ref = sr;
26             service = context.getService(ref);
27             service.hello(); // utilisation du service.
28         } else if (!ref.getProperty("Langue").equals("Fr") && sr.getProperty("Langue").equals("Fr")) {
29             // meilleur service
30             service.goodbye();
31             context.ungetService(ref);
32             ref = sr;
33             service = context.getService(ref);
34             service.hello(); // utilisation du service.
35         } // sinon, ne rien faire.
36         return service;
37     }
38
```

Définition d'un ServiceTrackerCustomizer

- Un ServiceTrackerCustomizer est appelé quand le ServiceTracker associé est tenu au courant d'un événement sur un service qu'il traque
- Modification du service
  - Connaissance de la référence sur le service en question

```
39    public void modifiedService(ServiceReference<HelloWorld> sr, HelloWorld service) {
40        if (ref.equals(sr) && !sr.getProperty("Langue").equals("Fr")) {
41            // chercher un meilleur service s'il y en a
42            chercherService();
43        } // sinon, un autre service a été modifié. On peut voir s'il est mieux
44    }
```

Référence sur le service qui a été modifié

11

Définition d'un ServiceTrackerCustomizer

- Un ServiceTrackerCustomizer est appelé quand le ServiceTracker associé est tenu au courant d'un événement sur un service qu'il traque
- Suppression du service (désenregistrement)
  - Connaissance de la référence sur le service en question

```
45
46    public void removedService(ServiceReference<HelloWorld> sr, HelloWorld service) {
47        if (ref != null && ref.equals(sr)) {
48            service.goodbye();
49            service = null;
50            context.ungetService(ref);
51            ref = null;
52            chercherService();
53        }
54    }
```

Référence sur le service qui a été supprimé

12

Définition d'un ServiceTrackerCustomizer

```
56    private void chercherService() {
57        try {
58            Collection<ServiceReference<HelloWorld>> refs = context.getServiceReferences(HelloWorld.class, "(Langue=*)");
59            if (!refs.isEmpty()) { // il existe refs.size() service.s disponible.s
60                Iterator<ServiceReference<HelloWorld>> it = refs.iterator();
61                ref = it.next(); // on prend le premier service et on va chercher un en français.
62                while (!ref.getProperty("Langue").equals("Fr") && it.hasNext()) {
63                    ServiceReference<HelloWorld> r = it.next();
64                    if (r.getProperty("Langue").equals("Fr")) {
65                        ref = r;
66                        break;
67                    }
68                }
69                service = context.getService(ref); // on prend le meilleur service trouvé.
70                service.hello();
71            }
72        } catch (InvalidSyntaxException e) { e.printStackTrace(); }
73    }
74 }
75
```


13

Conclusion sur les ServiceTrackers

- Fournit des traqueurs dédiés à chaque service requis (par type de service)
- Est-ce réellement mieux que les écouteurs de services ?
  - Code très proche (implémentation des 3 méthodes)
  - Toujours au sein du code uniquement (plusieurs choses à la charge du développeur)

14

Solution ?

- Les Services Déclaratifs (Declarative Services)
  - Ajouté à partir de la release 4
- Niveau d'abstraction plus élévé
- Nouvelle architecture
- Nouveaux types de composants
- Nouveaux cycles de vie
- Nouveaux outils : nouvel ADL (descriptions xml)