Cours 2.1

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

Bachir Djafri

1

Recherche ou Courtage de services

- Filtrage par des expressions de condition LDAP (RFC1960) sur les propriétés enregistrées par les services
- Expressions de filtrage
  - Expressions simples (attribut opérateur valeur)
  - Valeurs de type `String, Numerique, Character, Boolean,Vector, Array`
  - Attributs insensibles aux majuscules/minuscules
  - L'attribut ***`objectClass`*** représente le nom du service
  - Opérateurs `>=, <=, =, ~=` (approximativement égal), `=*` (présent)
  - Connecteurs logiques `&, |, ! : (&(K1)(K2)(K3))`
- Exemples
  - Tous les services de type HelloWorld ayant Langue=Fr

```
getServiceReferences(HelloWorld.class.getName(), "(Langue=FR)")
getServiceReferences(HelloWorld.class,
            "(&(objectClass=exemple.service.HelloWorld)(Langue=FR))")
```

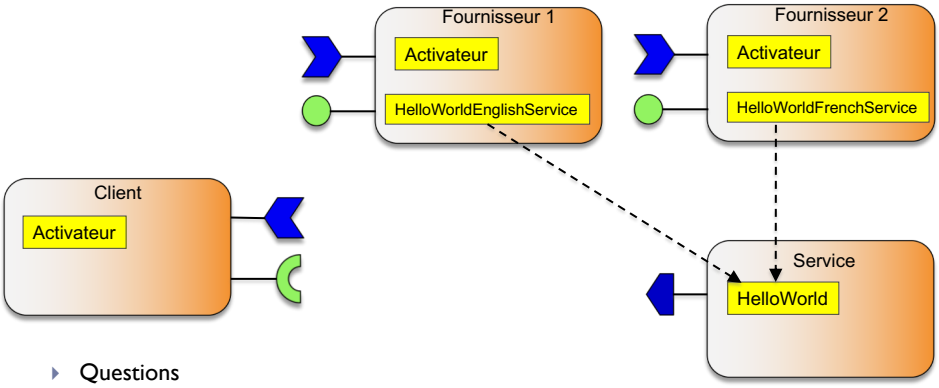
  - Tous les services ayant une langue quelconque

```
getServiceReferences(HelloWorld.class, "(Langue=*)");
```


2

Exemple 3 : extension de l'exemple 2

- Un 2nd fournisseur fournit un service satisfaisant le contrat « HelloWorld »



- Questions
  - Quel est le comportement dynamique du client quand les 2 fournisseurs sont actifs avant lui ?
  - Si le client utilise le 1er fournisseur et que celui-ci est désactivé, que se passe t-il ?
  - Peut-il utiliser le second fournisseur ? Si oui, comment ?

3

Exemple 3

- Pour avoir un bundle dynamique (réactif) aux modifications du contexte, il faut que le bundle soit à l'écoute de ces modifications du contexte
- La première solution consiste à utiliser des Ecouteurs
  - Ecouteurs de services;
  - Ecouteurs de composants (bundles);
  - Ecouteurs d'environnement (Framework);
  - Autres écouteurs.
- Application du patron Observateur/Observable
  - Le composant client = observateur;
  - L'environnement (context) = observable / observé;
  - Le client reçoit les événements envoyés par l'environnement selon le type de l'écouteur enregistré auprès de l'environnement.

4

Le concept de liaison dynamique (1/2)

- OSGi permet à ses composants (bundles) de sélectionner à l'exécution le composant qui fournira le service requis (s'il y a plusieurs fournisseurs)
- Ceci induit un comportement dynamique fort
  - Un bundle attend que les services dont il a besoin soient actifs pour être lui-même actif
  - Il peut dynamiquement changer de service s'il trouve un service plus « performant »
  - La disparition d'un bundle est aussi gérée
- Ce dynamisme est géré au travers d'évènements qui sont
  - Le départ ou l'arrivée d'un bundle — interface BundleListener
  - Le départ ou l'arrivée d'un service — interface ServiceListener
  - Relatif à la plateforme (environnement) — interface FrameworkListener
  - Autres — interfaces LogListener, …

5

Le concept de liaison dynamique (2/2)

- Cycle de vie « traditionnel » d'un bundle B
  - Recherche initiale des services nécessaires au fonctionnement
    - Si présent, on utilise le service le plus adéquat (au travers du questionnement sur ses propriétés)
    - Sinon, on attend un service (ou on sort, throw new Exception)
  - Lors de l'arrivée/départ d'un service S

⇨ génération d'un événement capturé par un « ServiceListener »

- Si S part alors qu'il était utilisé, B recommence une recherche initiale
- Si S arrive et que S est plus intéressant que le service actuel pour B, B change pour S
- Si le service S est modifié (changement de propriétés), B doit s'assurer que S répond toujours bien au service demandé

6

Les trois catégories d'évènements

- FrameworkEvent
  - Notifie le démarrage et les erreurs du Framework (Felix, Equinox, etc.)
  - Interface `FrameworkListener`, Méthode `frameworkEvent`
    - Traitement séquentiel et asynchrone des listeners (par event dispatcher)
- BundleEvent
  - Notifie les changements dans le cycle de vie des bundles
  - Interface `BundleListener`, méthode `bundleChanged`
    - Traitement séquentiel et asynchrone des listeners (par event dispatcher)
  - Interface `SynchronousBundleListener`, méthode `bundleChanged`
    - Traitement séquentiel et synchrone des listeners (avant le traitement du R2 changement d'état)
- ServiceEvent (traité dans ce cours)
  - Notifie l'enregistrement ou le retrait de services
  - Interface `ServiceListener`, méthode `serviceChanged`
    - Traitement séquentiel et synchrone des listeners

Exemple 4

- Toujours un service, un client de ce service et deux fournisseurs de ce service (Français et Anglais)
- 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
- En somme, le client doit être à l'écoute du contexte et plus précisément des enregistrements (ou départs) des services qui l'intéressent !
- Seul le composant (bundle) client est modifié (pour être dynamique)

Exemple 4

L'activateur devient aussi écouteur de services

Capture de l'événement « un bundle implémentant HelloWorld » est modifié (props modifiées)

Capture de l'événement « un bundle implémentant HelloWorld » est activé (enregistrement de services)

Capture de l'événement « un bundle implémentant HelloWorld » est désactivé (retrait de services)

```
13
14  public class Activateur implements BundleActivator, ServiceListener {
15
16      private BundleContext context;
17      private HelloWorld service;
18      private ServiceReference<HelloWorld> ref;
19
20+     public void start(BundleContext bundleContext) throws Exception {
27
28+     public void stop(BundleContext bundleContext) throws Exception {
38
39-     public void serviceChanged(ServiceEvent event) {
40          ServiceReference<?> r = event.getServiceReference();
41          String[] objectClasses = (String[]) r.getProperty("objectClass");
42          if (objectClasses[0].equals(HelloWorld.class.getName())) {
43              // L'évenement concerne un service de type HelloWorld
44              ServiceReference<HelloWorld> sr = (ServiceReference<HelloWorld>) r;
45              switch (event.getType()) {
46              case ServiceEvent.MODIFIED:
47                  traitementModificationService(sr);
48                  break;
49              case ServiceEvent.REGISTERED:
50                  traitementNouveauService(sr);
51                  break;
52              case ServiceEvent.UNREGISTERING:
53                  traitementDepartService(sr);
54              }
55          }
56      }
57
58+     private void traitementNouveauService(ServiceReference<HelloWorld> sr) {
73
74+     private void traitementModificationService(ServiceReference<HelloWorld> sr) {
80
81+     private void chercherService() {
99
100+     private void traitementDepartService(ServiceReference<HelloWorld> sr) {
109 }
```


9

Exemple 4

utilisation de l'écouteur d'évènement

On informe le contexte de l'arrivée de cet écouteur

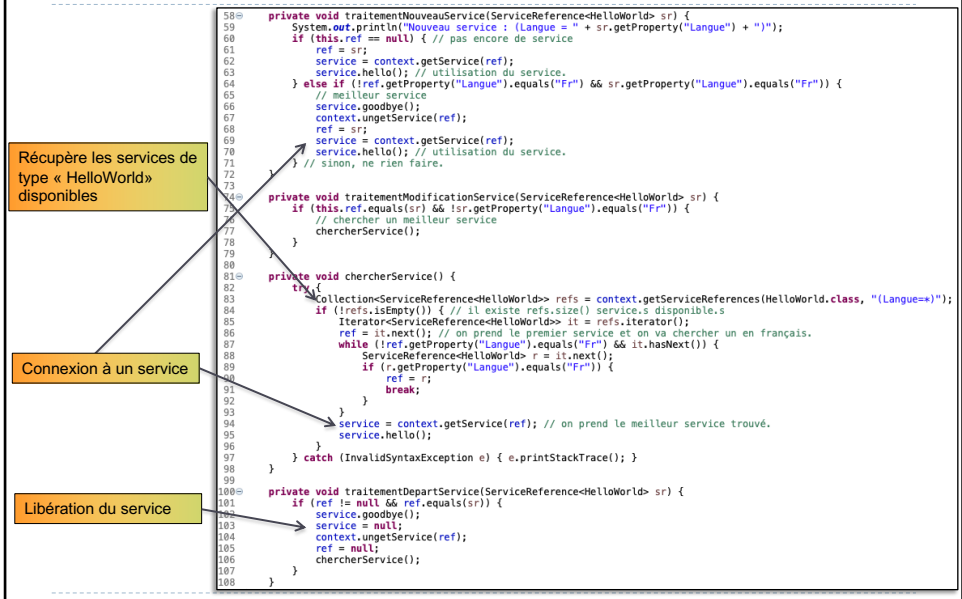
On recherche les service disponibles

On informe le contexte du départ d'un écouteur

```
14  public class Activateur implements BundleActivator, ServiceListener {
15
16      private BundleContext context;
17      private HelloWorld service;
18      private ServiceReference<HelloWorld> ref;
19
20      public void start(BundleContext bundleContext) throws Exception {
21          context = bundleContext;
22          ref=null;
23          service=null;
24          context.addServiceListener(this);
25          chercherService();
26      }
27
28      public void stop(BundleContext bundleContext) throws Exception {
29          if (service != null) {
30              service.goodbye();
31              service = null;
32              context.ungetService(ref);
33              ref=null;
34              context.removeServiceListener(this);
35          }
36          context = null;
37      }
38
39+     public void serviceChanged(ServiceEvent event) {
57
58+     private void traitementNouveauService(ServiceReference<HelloWorld> sr) {
73
74+     private void traitementModificationService(ServiceReference<HelloWorld> sr) {
80
81+     private void chercherService() {
99
100+     private void traitementDepartService(ServiceReference<HelloWorld> sr) {
109 }
110
```


10

Exemple 4




```
58    private void traitementNouveauService(ServiceReference<HelloWorld> sr) {
59        System.out.println("Nouveau service : (Langue = " + sr.getProperty("Langue") + ")");
60        if (this.ref == null) { // pas encore de service
61            ref = sr;
62            service = context.getService(ref);
63            service.hello(); // utilisation du service.
64        } else if (!ref.getProperty("Langue").equals("Fr") && sr.getProperty("Langue").equals("Fr")) {
65            // meilleur service
66            service.goodbye();
67            context.ungetService(ref);
68            ref = sr;
69            service = context.getService(ref);
70            service.hello(); // utilisation du service.
71        } // sinon, ne rien faire.
72    }
73
74    private void traitementModificationService(ServiceReference<HelloWorld> sr) {
75        if (this.ref.equals(sr) && !sr.getProperty("Langue").equals("Fr")) {
76            // chercher un meilleur service
77            chercherService();
78        }
79    }
80
81    private void chercherService() {
82        try {
83            Collection<ServiceReference<HelloWorld>> refs = context.getServiceReferences(HelloWorld.class, "(Langue=*)");
84            if (!refs.isEmpty()) { // il existe refs.size() service.s disponible.s
85                Iterator<ServiceReference<HelloWorld>> it = refs.iterator();
86                ref = it.next(); // on prend le premier service et on va chercher un en français.
87                while (!ref.getProperty("Langue").equals("Fr") && it.hasNext()) {
88                    ServiceReference<HelloWorld> r = it.next();
89                    if (r.getProperty("Langue").equals("Fr")) {
90                        ref = r;
91                        break;
92                    }
93                }
94                service = context.getService(ref); // on prend le meilleur service trouvé.
95                service.hello();
96            }
97        } catch (InvalidSyntaxException e) { e.printStackTrace(); }
98    }
99
100   private void traitementDepartService(ServiceReference<HelloWorld> sr) {
101       if (ref != null && ref.equals(sr)) {
102           service.goodbye();
103           service = null;
104           context.ungetService(ref);
105           ref = null;
106           chercherService();
107       }
108   }
```


11

Exemple 4

- Une classe Ecouteur de services

```
 9  public class Ecouteur implements ServiceListener {
10      private ServiceReference<HelloWorld> ref;
11
12      public void serviceChanged(ServiceEvent event) {
13          ServiceReference<?> r = event.getServiceReference();
14          String[] objectClasses = (String[]) r.getProperty("objectClass");
15          if (objectClasses[0].equals(HelloWorld.class.getName())) {
16              // L'évenement concerne un service de type HelloWorld
17              ref = (ServiceReference<HelloWorld>) r;
18              switch(event.getType()) {
19                  case ServiceEvent.REGISTERED : traitementNouveauService(ref); break;
20                  case ServiceEvent.MODIFIED : traitementModificationService(ref); break;
21                  case ServiceEvent.UNREGISTERING : traitementDepartService(ref);
22              }
23          }
24      }
25
26      private void traitementNouveauService(ServiceReference<HelloWorld> ref) {
29
30      private void traitementModificationService(ServiceReference<HelloWorld> ref) {
33
34      private void traitementDepartService(ServiceReference<HelloWorld> ref) {
37  }
38
```


12

Première conclusion 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 !)*

… et à compléter

- 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)