

TP1

Un article de EOSGi.

Sommaire

- 1 1 Installation et prise en main d'une plate-forme OSGi
 - 1.1 1.1 Installation de la plate-forme Felix
 - 1.2 1.2 Utilisation de la plate-forme Felix
 - 1.3 1.3 Utilisation de la console graphique de Felix
 - 1.4 1.4 Utilisation de la console telnet de Felix
 - 1.5 1.5 Utilisation d'une console Web avec Felix
 - 1.6 1.6 Autres consoles
- 2 2 Installation du dépôt de bundles local et déploiement de bundles avec l'OBR
 - 2.1 2.1 Génération de l'index OBR
 - 2.2 2.2 Installation
- 3 3. Déploiement depuis un dépôt Maven
- 4 4. Déploiement depuis un répertoire
- 5 5. Déploiement d'archives Java (jar) qui ne sont pas des bundles OSGi
- 6 6. Niveaux de démarrage

1 Installation et prise en main d'une plate-forme OSGi

Dans le cadre de ces travaux pratiques, nous utiliserons la plate-forme OSGi Felix (<http://cwiki.apache.org/FELIX/index.html>) du projet Apache.org (<http://www.apache.org/>) . Il faut savoir qu'il existe d'autres supports d'exécution OSGi, tels que Knopflerfish (<http://www.knopflerfish.org/>) , Equinox (<http://www.eclipse.org/equinox/>) , Concierge (<http://concierge.sourceforge.net/>) ou Prosyst (<http://www.prosyst.com/>) .

Felix, est une implantation *open-source* du cadre de conception OSGi et d'une partie des services standards répondant à la spécification OSGi Release 4 (<http://www2.osgi.org/javadoc/r4/>) .

1.1 Installation de la plate-forme Felix

Les distributions officielles de la plate-forme Felix peuvent être obtenues depuis le site Web : <http://cwiki.apache.org/FELIX>.

Il est également possible de construire la plate-forme Felix à partir des sources issues du dépôt SVN <http://svn.apache.org/repos/asf/felix/trunk> en utilisant Ant et Maven 2.

Cependant pour simplifier l'installation, vous trouverez l'ensemble des binaires dans l'archive ICAR08.zip (<http://www-valoria.univ-ubs.fr/Nicolas.Le-Sommer/ICAR08.zip>) .

Les répertoires de l'archive contiennent :

- `env_felix_tps/felix` : l'environnement d'exécution de Felix,
- `env_felix_tps/felix/bundle` : les bundles nécessaires au fonctionnement de Felix (e.g. console),
- `env_felix_tps/mvn_repository` : le dépôt maven, dépôt qui inclue l'OBR (OBR pour *OSGi Bundle Repository*) contenant des bundles qui serviront de support aux travaux pratiques,
- `env_felix_tps/tools` : des utilitaires pour le développement de bundles OSGi (maven, ant),
- `env_felix_tps/trunk` : le trunk SVN de felix,
- `tps`: les sujets des travaux pratiques,
- `dev-tps`: les codes sources nécessaires à la réalisation des travaux pratiques,
- `env_tps.sh`: scripts permettant de définir l'environnement de travail.

Installation et utilisation

```
unzip ICAR08.zip
cd ICAR08
source env_tps.sh
run_felix.sh
```

1.2 Utilisation de la plate-forme Felix

Lorsque vous aurez installé la plate-forme Felix, vous exécuterez la commande `felix.sh` (ou `felix.bat` pour Windows ou `java -jar bin/felix.jar`) pour démarrer la plate-forme. Vous accéderez dès lors à la console locale de la passerelle, console qui vous permettra de pouvoir tester quelques commandes. Les commandes fournissant des informations sur les bundles installés sont :

- `help` : affiche la liste des commandes disponibles.
- `ps` : affiche la liste des bundles ainsi que leurs états (remarque: cette commande est l'équivalent de la commande `bundles` d'Equinox).
- `ps -l` : idem mais avec l'URL utilisée pour l'installation initiale.
- `headers bid` : retourne les informations contenues dans le fichier Manifest du bundle *bid*.
- `packages` : liste les paquetages (versionnés) exportés par les bundles.
- `packages bid` : liste les paquetages (versionnés) exportés par le bundle *bid*.
- `services` : liste les services fournis par les bundles.
- `services -a` : liste les services fournis (y compris les services implémentant `org.apache.felix.shell.Command`) par les bundles.
- `services bid` : donne les informations détaillées sur les services fournis par le bundle *bid*.
- `services -u bid` : donne les informations détaillées sur les services utilisés par le bundle *bid*.

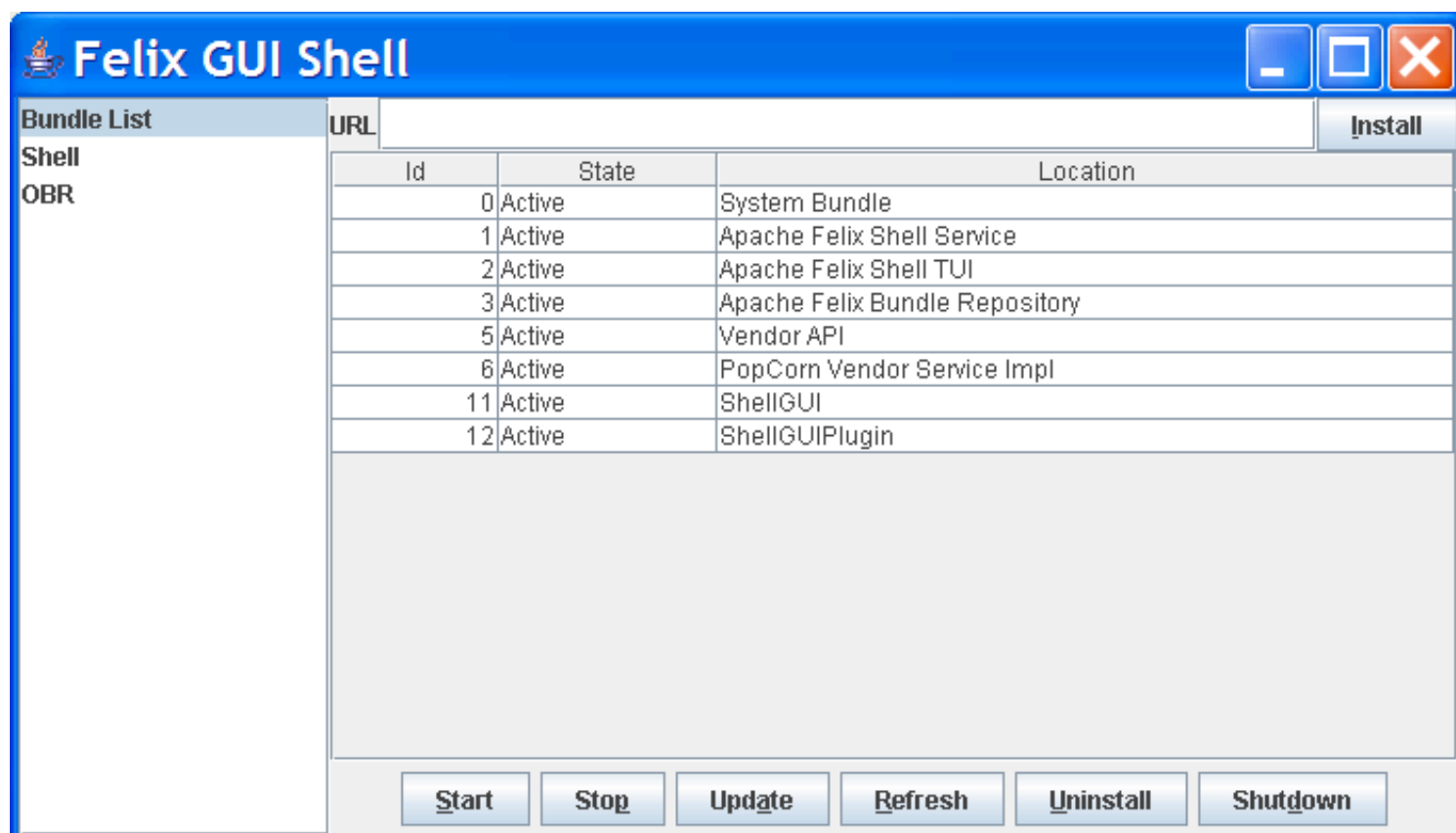
Les commandes liées à l'installation/démarrage sont :

- `cd baseurl` : facilite l'écriture des urls
- `install url` : installe un bundle à partir d'une URL
 - `file:...` pour un bundle stocké dans un système de fichiers local
 - `http:...` pour un bundle stocké sur un serveur distant
- `uninstall id` : désinstalle un bundle
- `start bid` : démarre un bundle en invoquant la méthode `start()` de l'activateur (s'il est spécifié dans le manifeste)
- `start url` : installe à partir d'une URL *url* et démarre celui-ci
- `stop bid` : arrête un bundle (et donc ses services)
- `update bid` : prépare une mise à jour du bundle (le id est donné dans la liste lors de la commande `ps`)

- `update bid url` : prépare une mise à jour du bundle id avec le jarfile donné par l'`url`
- `refresh` : exécute les mises à jour ou desinstallations
- `obr` : permet d'installer et de démarrer un bundle et ses dépendances (voir obr help)
- `shutdown` : arrête la passerelle (remarque: équivalent à `stop 0`)
- `gc` : déclenche le ramasse-miette (garbage collector)

1.3 Utilisation de la console graphique de Felix

Felix fournit une console graphique munis de plusieurs plugins pour simplifier la visualation/manipulation des bundles déployés et à déployer.



ID	State	Level	Name
[0]	[Active]	[0]	System Bundle (1.1.0.SNAPSHOT)
[1]	[Active]	[1]	Apache Felix Shell Service (1.1.0.SNAPSHOT)
[2]	[Active]	[1]	Apache Felix Shell TUI (1.1.0.SNAPSHOT)
[3]	[Active]	[1]	Apache Felix Bundle Repository (1.1.0.SNAPSHOT)
[5]	[Active]	[1]	Vendor API (1.0.0)
[6]	[Active]	[1]	PopCorn Vendor Service Impl (1.0.0)
[11]	[Active]	[1]	ShellGUI (0.9.0.SNAPSHOT)
[12]	[Active]	[1]	ShellGUIPlugin (0.9.0.SNAPSHOT)

Resources

- My repository
 - Apache Felix Declarative Services (0.9.0.SNAPSHOT)
 - Apache Felix EventAdmin (0.9.0.SNAPSHOT)
 - Apache Felix Log Service (0.9.0.SNAPSHOT)
 - HTTP Service (0.9.0.SNAPSHOT)
 - OSGi R4 Compendium Bundle (4.0.0)
 - OSGi R4 Core Bundle (4.0.0)

Apache Felix Declarative Services

```

description = Implementation of the Declarative Services specification 1.0
documentation = http://www.apache.org/
id = 6
  
```

Pour la démarrer, il vous faut déployer les bundles avec les commandes suivantes:

```

obr add-url file://repository_local_maven_utilisé_en_tp/repository.xml
obr start "ShellGUI"
obr start "ShellGUIPlugin"
  
```

1.4 Utilisation de la console telnet de Felix

```
obr start telnetd
```

Ouvrez un client Telnet (telnet ou PuTTY par exemple) avec l'adresse IP ou DNS de la passerelle (localhost si la passerelle est locale) sur le port 6623 et choisissez le shell de oscar (Oscar est l'ancien nom de Felix).

1.5 Utilisation d'une console Web avec Felix

Lancez le serveur HTTP et le bundle "Apache Felix Web Management Console".

```
obr start "Apache Felix Web Management Console"
```

Ouvrez un navigateur Web à l'adresse `http://localhost:8080/system/console/` ! login est admin et le mot de passe est admin

Remarque: "Apache Felix Web Management Console" est développé au moyen des éléments suivants:

- HTTP Service,
- OSGi R4 Compendium Bundle,
- Servlet 2.1 API,
- Apache Felix Declarative Services.

L'arrêt de l'un de ces éléments provoque naturellement l'arrêt du composant fournissant la servlet.

1.6 Autres consoles

- XMPP (fournit avec DysoWeb (<http://forge.objectweb.org/projects/dysoweb/>))
- MOSGI Console (<http://felix.apache.org/site/mosgi-managed-osgi-framework.html>)
- Jasmine Console (<http://wiki.jasmine.objectweb.org/xwiki/bin/view/Documentation/OSGiConsole>)
- UPnP OGD
- UPnP ExePf

2 Installation du dépôt de bundles local et déploiement de bundles avec l'OBR

L'OBR (*OSGi Bundle Repository*) RFC0112 (http://www2.osgi.org/download/rfc-0112_BundleRepository.pdf) est un service qui déploie un bundle et ses dépendances (transitives). Il se base sur un index XML qui peut être généré par l'outil BIndex. Apache Felix en fournit une des premières implémentations.

2.1 Génération de l'index OBR

Le plugin maven-bundle-plugin développé dans le cadre du projet Apache Felix, permet de construire et de

mettre à jour un OBR à partir de projets Maven lors de l'exécution du but `install`. L'OBR est situé dans le dépôt local Maven. L'emplacement par défaut est `C:\Documents and Settings\your_username\.m2\repository\repository.xml` sur Windows et `~/.m2/repository/repository.xml` sur Linux.

Dans le cadre de ces travaux pratiques, le dépôt Maven est le répertoire `env_felix_tps/mvn_repository` contenu dans l'archive qui vous a été donnée. Vous pouvez déposer vos projets dans l'OBR via la commande suivante:

```
mvn -Dmaven.repo.local=repertoire_archive/mvn_repository install
```

ou

```
mvn install
```

si vous avez exécuté préalablement la commande `source env_tps.sh`

Pour plus d'information concernant ce plugin, vous pouvez vous référer au fichier HTML contenu dans le répertoire `tools/maven-bundle-plugin/doc` de l'archive qui vous a été fournie. Vous pouvez également utiliser le modèle de document `pom.xml` contenu dans le répertoire `scripts` de cette archive.

L'index de l'OBR peut également être généré grâce à Bindex. Une fichier Ant réalisant cette opération est disponible dans le répertoire `tools` de votre archive. Pour construire l'index de l'OBR via cette méthode, vous devez utiliser la commande suivante:

```
ant -f tools/obr_bindex/build.my.repository.xml
```

Cette commande génère les 2 fichiers :

- `my.repository.xml` que vous installerez dans l'OBR
- `my.repository.html` que vous pourrez visualiser dans votre navigateur Web

2.2 Installation

Pour installer un nouveau dépôt de bundles, vous devez utiliser la commande ci-dessous

```
obr add-url url_vers_l_index_du_depot
```

Pour le TP, la commande sera :

```
obr add-url file:C:\eosgi\repo\my.repository.xml
```

Pour installer le dépôt de bundles local, vous spécifierez une URL du type `file:/path/repository/repository.xml` ou `path` est le répertoire dans lequel vous avez installé l'intergiciel Felix.

Pour utiliser les bundles installés dans le dépôt Maven, la commande sera :

```
obr add-url file://repository_local_maven_utilisé_en_tp/repository.xml
===2.3 Déploiement de bundles===
Les bundles indexés par le dépôt sont listés par la commande suivante :
<pre>obr list
```

Comme la liste peut être longue, la commande suivante limite l'affichage

```
obr list "SnackBar"
```

Pour installer des bundles depuis le (ou les) dépôt(s) de bundles, vous devez utiliser la commande suivante :

```
obr start nom_du_bundle
```

Par exemple:

```
obr start "HTTP Service"
```

```
obr start "Apache Felix WebAdmin"
```

Dès lors, votre bundle est déployé sur votre passerelle Felix, et peut être géré comme les autres bundles avec les commandes start, stop, uninstall.

3. Déploiement depuis un dépôt Maven

Il est également possible de déployer des bundles à partir d'un dépôt Maven2 ou depuis un répertoire quelconque (ie non indexé pour un OBR). Le projet *Pax Runner* (<http://wiki.ops4j.org/confluence/display/ops4j/Pax+Runner>) fournit des *URL Handlers* qui permettent cela.

L'exemple suivant illustre l'usage du handler Maven avec lequel la syntaxe des URLs doit être `mvn:[repository_url!]groupId/artifactId[/[version]/[type]]`.

```
...
start handler-mvn-0.5.0-SNAPSHOT.jar
install mvn:http://repo1.maven.org/maven2!org.apache.felix/org.osgi.core/1.0.0
install mvn:org.apache.felix/javax.servlet
install mvn:org.apache.felix/org.osgi.compendium/1.0.0
start mvn:org.apache.felix/org.apache.felix.scr/LATEST
start mvn:org.apache.felix/org.apache.felix.eventadmin
start mvn:org.apache.felix/org.apache.felix.wireadmin/SNAPSHOT
ps -l
update 5
```

Remarques:

1. *Pax Runner* permet bien d'autres choses encore ...
2. les URL de type `mvn:${groupId}/${artifactId}/LATEST` peuvent être utilisé pour renseigner l'entrée `Bundle-UpdateLocation` du manifeste du bundle.
3. contrairement à l'OBR, ces handlers ne déploient pas les dépendances entre bundles.

4. Déploiement depuis un répertoire

Le bundle `FileInstall` (<http://www.aqute.biz/Code/FileInstall>) déploie automatiquement les bundles placés dans un répertoire (designé par la propriété de configuration `aQute.fileinstall.dir`). Les bundles peuvent être copiés, mis à jour ou supprimés dynamiquement dans ce répertoire : `FileInstall` répercute les `install/start/update/stop/uninstall` des bundles immédiatement (modulo la propriété de configuration `aQute.fileinstall.poll` qui définit l'intervall de polling en seconde).

Ajoutez ces propriétés au fichier de configuration de Felix (`%EOSGI_HOME%\felix\conf\config.properties`) pour surcharger les propriétés par défaut.

```
-----
aQute.fileinstall.poll=2000
aQute.fileinstall.dir=../loaddir.for.fileinstall
aQute.fileinstall.debug=1
-----
```

Lancez Felix et déployez `FileInstall`

```
-----
start http://www.aqute.biz/repo/biz/aQute/fileinstall/1.3.4/fileinstall-1.3.4.jar
rem install wrap:mvn:biz.aQute/fileinstall/LATEST pour la dernière version
-----
```

Dans un shell:

```
-----
cd %EOSGI_HOME%
mkdir loaddir.for.fileinstall
copy repo\vendor-1.0.0.jar loaddir.for.fileinstall
copy repo\vendor.weiner-0.1.0.jar loaddir.for.fileinstall
-----
```

Dans la console de Felix:

```
-----
ps
services
-----
```

Dans un shell:

```
-----
del loaddir.for.fileinstall\vendor.weiner-0.1.0.jar
-----
```

Dans la console de Felix:

```
-----
ps
-----
```

```
services
```

Remarque: FileInstall agit également sur la (re)configuration des bundles: pour cela, un fichier dont le nom est le PID du ManagedService suffixé par l'extension `.cfg` doit contenir un fichier contenant les propriétés de (re)configuration. Par exemple, placez ce fichier `eosgi.snackbar.vendor.weiner.cm.cfg` dans le répertoire `loadDir.for.fileinstall`

```
stocklevel=100
vcard.address=Grenoble, France
```

Remarque: FileInstall devrait rejoindre le projet Apache Felix.

Remarque: Le projet Sling propose un outil similaire `org.apache.sling.jcr.jcrinstall` qui s'appuie sur un Content Repository comme JackRabbit (JSR 170 et JSR 283).

5. Déploiement d'archives Java (jar) qui ne sont pas des bundles OSGi

Pax Runner fournit également un autre handler `wrap`: qui emballe un jar standard en bundle (<http://wiki.ops4j.org/confluence/display/ops4j/Handler+-+Wrap>). Ce handler utilise l'outil BND (<http://www.aqute.biz/Code/Bnd>) de Peter Kriens. Ce wrapper permet ainsi de déployer directement des archives Java (jar) qui ne sont pas des bundles OSGi et qui sont déposés dans un répertoire local ou distant, ou dans un dépôt Maven.

Quelques exemples avec le handler `wrap`:

```
...
start pax-runner-handler-wrap-0.5.0-SNAPSHOT.jar
install wrap:file:c:/repo/commons-logging-1.1.jar
headers bid
install wrap:http://repo1.maven.org/maven2/commons-logging/commons-logging/1.1/commons-logging-1.1.jar
install wrap:mvn:commons-logging/commons-logging/1.1
install wrap:mvn:commons-logging/commons-logging/LATEST
install wrap:mvn:commons-logging/commons-logging/1.1,file:commons-logging-1.1.bnd
install wrap:mvn:commons-logging/commons-logging/1.1,file:commons-logging-1.1.bnd$Bundle-SymbolicName
```

6. Niveaux de démarrage

La plate-forme OSGi définit plusieurs niveaux de démarrage pour les *bundles* installés. Le niveau 0 est celui du *bundle System* (toujours identifié 0). Le passage d'un niveau vers un niveau supérieur provoque le démarrage des *bundles* du niveau supérieur. Inversement, le passage vers un niveau inférieur provoque l'arrêt des *bundles* du niveau courant. Le service *Start Level* administre ces transitions de niveau.

Les deux commandes du shell de Felix relatives à ce service sont `startlevel` et `bundlelevel`.

- `bundlelevel level bid1 bid2 ...` : positionne le niveau de démarrage d'un ou de plusieurs bundles
- `bundlelevel bid` : donne le niveau de démarrage d'un bundle
- `startlevel` : donne le niveau de démarrage de la plateforme
- `startlevel level` : positionne le niveau de démarrage de la plateforme

Exercice de changement de niveaux à rédiger

```
startlevel  
obr start telnetd  
bundlelevel 4  
ps  
bundlelevel 2 4  
ps
```

Que s'est il passé pour le bundle telnetd (bid=4) ?

```
startlevel 2  
startlevel  
ps
```

Que s'est il passé pour le bundle telnetd (bid=4) ?

```
startlevel 1  
startlevel  
ps
```

Que s'est il passé pour le bundle telnetd (bid=4) ?

Récupérée de « <http://localhost/eOSGi/index.php?title=TP1> »

ICAR-08-TP2

Un article de EOSGi.

Application Snackbar



Dans le cadre de ce TP, nous considérons la mise en œuvre d'une application baptisée Snackbar de manière modulaire en plusieurs bundles utilisant et offrant des services dynamiquement au travers du registre de services d'OSGi. Cette application sera construite selon les principes de l'architecture orientée service dynamique (DSOA). Cette application sera développée en utilisant des solutions basées respectivement sur:

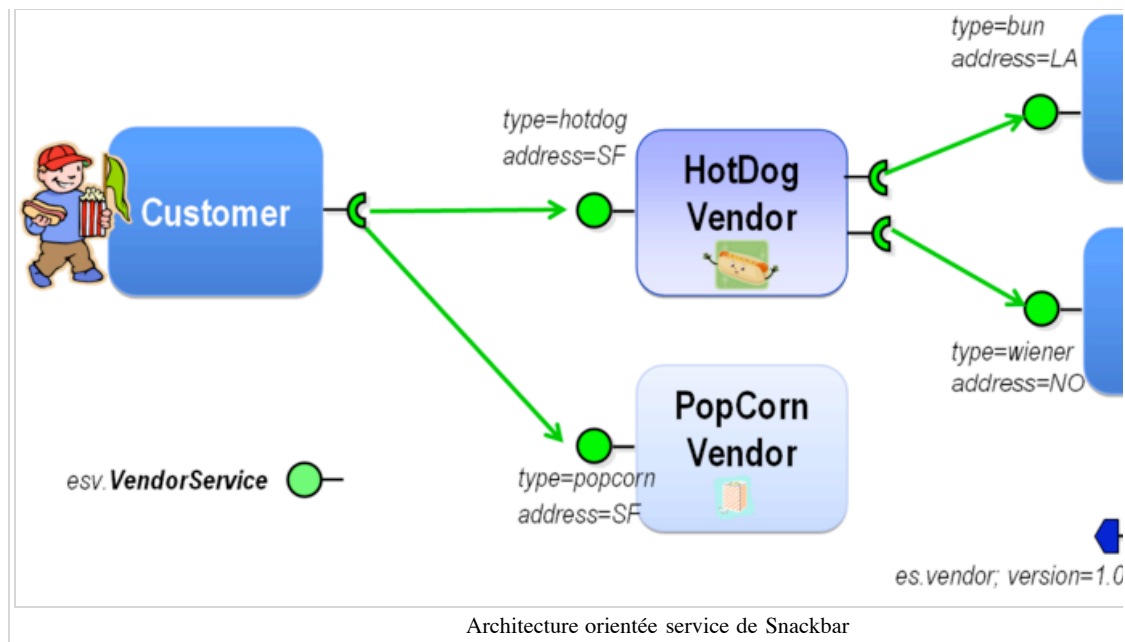
- `org.osgi.framework.ServiceListener`
- `org.osgi.util.tracker.ServiceTracker`
- `org.osgi.framework.ServiceFactory`

Sommaire

- 1 Architecture de l'application
- 2 Les vendeurs
 - 2.1 L'interface de service `VendorService` version 1.0.0
 - 2.2 Enregistrement des services des vendeurs dans l'OBR
 - 2.3 Développement et installation du bundle `vendor.popcorn`
 - 2.4 Déploiement des bundles de type `VendorService` sur la passerelle Felix
- 3 Recherche des services pour les clients
 - 3.1 Recherche de service pour les clients (avec `ServiceListener`)
 - 3.2 Recherche de service pour les clients (avec `ServiceTracker`)
 - 3.3 Fabrique d'instance de services

Architecture de l'application

L'architecture de l'application est donnée par la figure suivante:



Les clients requièrent le service d'un vendeur pour acheter des hot-dogs ou du pop-corn selon la disponibilité de la marchandise chez les vendeurs.

Le bundle vendeur de pop-corn et le bundle vendeur de hot-dog mettent en application (et fournissent) le service `VendorService`. Le vendeur de hot-dog dépend de deux autres services `VendorService` pour acheter les ingrédients: des brioches (bun) et des saucisses (weiner).

Chaque client et chaque vendeur sera conditionné dans un bundle indépendant. L'application sera ainsi constituée des bundles suivants:

- customer,
- vendor.popcorn,
- vendor.hotdog,
- vendor.bun,
- vendor.weiner.

Ils dépendent tous du bundle `snackbar.vendor` dont ils importent le paquetage `icar.snackbar.service.vendor; version=1.0.0`

Ces bundles pourront être déployés et/ou être mis-à-jour dans un ordre quelconque. L'établissement des liaisons (de service) entre les clients et les vendeurs s'effectueront via le registre de service. Pour ce faire, chaque vendeur publiera son service `VendorService` (en le qualifiant de la propriété `type` indiquant le produit vendu), et chaque client recherchera les services `VendorService` désirés (en fonction de la valeur de la propriété `type`) et réagira aux départs et aux arrivées des services `VendorService`.

Remarques:

Selon les principes de l'Architecture orientée service (SOA), le choix d'un service parmi plusieurs est "décidé" lors de l'exécution (late binding). En outre, selon les principes du SOA dynamique, tout service est substituable en cours d'exécution par un autre si ce dernier fournit le service demandé (interface Java + propriétés d'enregistrement). Nous verrons qu'au cours de l'exercice, nous déploierons de nouveaux bundles qui se substitueront aux bundles initiaux (`vendor.hotdog.scr`, `vendor.weiner.scr`, ...) ou compléteront l'application (`vendor.pizza`).

Les vendeurs

L'interface de service `VendorService` version 1.0.0

L'interface de service utilisée entre les éléments de l'application `SnackBar` est la suivante. Elle constitue une spécification partagée entre tous les développeurs de clients et de vendeurs.

```

package eosgi.snackbar.service.vendor;
/**
 * A simple service interface for the SnackBar application
 * @version 1.0.0
 */
public interface VendorService
{
    /**
     * Service registration property (String) defining the type of thing sold by the vendor.
     * This property is mandatory.
     */
    public final static String TYPE = "type";

    /**
     * Get the name of the vendor
     * @return the name of the vendor
     */
    public String getName();

    /**
     * Buy a thing
     * @return a string representing the bought thing, null if the sale was discarded
     */
    public String buy();
}

```

Cette interface est fournie par le bundle `snackbar.vendor-1.0.0.jar`. Le code source de ce bundle est placé dans le répertoire `dev/snackbar/vendor`.

Pour les besoins de ce TP, vous allez devoir compiler et déposer le bundle contenant cette interface dans l'OBR. Pour ce faire, vous devez effectuer les opérations suivantes:

```

source env_tps.sh
cd tp-2/dev/vendor
mvn install

```

Remarque: Les informations nécessaires à la création de ce bundle sont définies dans le fichier `pom.xml`. Ces données sont extraites par le plugin `maven-bundle-plugin` pour créer le bundle, et le déposer dans l'OBR lorsque vous exécutez la commande `mvn install`.

Enregistrement des services des vendeurs dans l'OBR

L'enregistrement des services fournis par un bundle se fait généralement lors de l'exécution de la méthode `start(BundleContext)` au moyen de la méthode `BundleContext.registerService(nom_interface_service, servent, propriétés_d_enregistrement)[1]` (<http://www2.osgi.org/javadoc/r4/org/osgi/framework/BundleContext.html>). L'objet `ServiceRegistration[2]` (<http://www2.osgi.org/javadoc/r4/org/osgi/framework/ServiceRegistration.html>) est conservé pour modifier les propriétés d'enregistrement en cours d'exécution et pour désenregistrer le service au moment de l'arrêt du bundle (méthode `stop(BundleContext)`) ou bien si les conditions pour rendre le service ne sont plus remplies; (par exemple, le vendeur de hotdog est en rupture de stock de saucisses : il ferme la boutique !)

```

package eosgi.snackbar.bundle.vendor.bun.impl;
import java.util.Dictionary;
import java.util.Hashtable;
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.framework.ServiceRegistration;
import eosgi.snackbar.service.vendor.VendorService;
/**
 * A simple service implementation for the SnackBar application
 * @author eosgi
 * @version 1.0.0
 */
public class Vendor implements VendorService, BundleActivator {
    private BundleContext bundleContext;
    private ServiceRegistration serviceRegistration;

    public String buy() {
        System.out.println(
            "Bundle "+bundleContext.getBundle().getBundleId()+": "
            +"Somebody buys "+ "Bun(1)");
        return "Bun(1)";
    }
}

```

```

    }

    public String getName() {
        return "Bun vendor";
    }

    public void start(BundleContext bundleContext) throws Exception {
        this.bundleContext=bundleContext;
        Dictionary serviceRegistrationProperties=new Hashtable();
        serviceRegistrationProperties.put(TYPE,"bun");
        serviceRegistration=bundleContext.registerService(VendorService.class.getName(), this, servi
    }

    public void stop(BundleContext bundleContext) throws Exception {
        serviceRegistration.unregister();
    }
}

```

Remarque:

1. cette classe fait à la fois office d'activateur et de servent.

Compilation et installation du bundle `vendor.bun` dans l'OBR

```
cd vendor.bun
mvn install
```

Compilation et installation du bundle `vendor.weiner` dans l'OBR

```
cd vendor.weiner
mvn install
```

Compilation et installation du bundle `vendor.hotdog` dans l'OBR

```
cd vendor.bun
mvn install
```

Développement et installation du bundle `vendor.popcorn`

En vous inspirant de l'exemple précédent et des exemples de bundles `vendor.bun`, `vendor.weiner` développez, compilez et installez le bundle modélisant le vendeur de popcorn (i.e. le bundle `vendor.popcorn`).

Déploiement des bundles de type `VendorService` sur la passerelle Felix

- Démarrer la passerelle Felix à l'aide de la commande `run_felix.sh`.
- Ajouter votre OBR local dans la liste des OBRs utilisés par Felix via la commande suivante.

```
obr add-url URL_OBR_LOCAL
```

- Listez les bundles disponibles via l'OBR grâce à la commande suivante:

```
obr list
```

- Installez les différents bundles implantant l'interface de service `VendorService` via la commande suivante:

```
obr start BUNDLE_A_DEMARRER
```

Recherche des services pour les clients

Recherche de service pour les clients (avec ServiceListener)

La recherche des services couramment enregistrés se fait au moyen de la méthode

`BundleContext.getServiceReferences (nom_interface_service, filtre_LDAP)` [3]

(<http://www2.osgi.org/javadoc/r4/org/osgi/framework/BundleContext.html#getServiceReferences%28java.lang.String,%20java.lang.String%29>)

. L'expression de filtre LDAP est appliquée sur les propriétés d'enregistrement des services. Dans

l'exemple ci-dessous, on ne recherche que les vendeurs de popcorn ou les vendeurs de hotdog.

```

List<VendorService> vendorServices=...;

ServiceReference[] serviceReferences=bundleContext.getServiceReferences(VendorService.class.getName(),
if(serviceReferences!=null) {
    for (int i = 0; i < serviceReferences.length; i++) {
        ServiceReference serviceReference=serviceReferences[i];
        if(serviceReference!=null && serviceReference.getBundle().getState()!=Bundle.STOPPING)
            VendorService vendorService=(VendorService) bundleContext.getService(serviceReference);
        vendorServices.add(vendorService);
    }
}
...
foreach(v : vendorServices) {
    // fait quelque chose avec chaque service
}
...

```

Une fois la liste `vendorServices` remplie, le bundle pourrait utiliser les services sans se soucier de le "devenir" (*Remarque: c'est ce que font beaucoup de développeurs de plugin Eclipse*).

Cependant, dans un canevas OSGi, de nouveaux services peuvent être enregistrés après cette phase et des services utilisés peuvent être désenregistrés (arrêt ou mise à jour du bundle qui les fournit ou bien conditions non satisfaisantes pour remplir le service).

Le canevas dispose d'un système événementiel permettant de notifier les changements

(`org.osgi.framework.ServiceEvent` [4])

(<http://www2.osgi.org/javadoc/r4/org/osgi/framework/ServiceEvent.html>) dans les services. Les bundles

doivent suivre ces événements en positionnant des `org.osgi.framework.ServiceListener` [5]

(<http://www2.osgi.org/javadoc/r4/org/osgi/framework/ServiceListener.html>) .

L'utilisation de `ServiceListener` est donné par le code suivant extrait du bundle `customer.simple`.

Remarque : cet extrait est surtout là pour vous effrayer car des solutions simplifient la prise en charge de la dynamique des services.

```

...
private static final String FILTERSTR = "(|(type=popcorn)(type=hotdog))";

private Map<ServiceReference, VendorService> services;

private void fillServices() throws InvalidSyntaxException {
    ServiceReference[] serviceReferences=bundleContext.getServiceReferences(VendorService.class.getName(),
if(serviceReferences!=null) {
    for (int i = 0; i < serviceReferences.length; i++) {
        ServiceReference serviceReference=serviceReferences[i];
        if(serviceReference!=null && serviceReference.getBundle().getState()!=Bundle.STOPPING)
            VendorService vendorService=(VendorService) bundleContext.getService(serviceReference);
        services.put(serviceReference, vendorService);
    }
}
}

private void releaseServices() throws InvalidSyntaxException {
    Iterator<Map.Entry<ServiceReference, VendorService>> iterator=services.entrySet().iterator();
    while(iterator.hasNext()){
        Map.Entry<ServiceReference, VendorService> entry=iterator.next();
        bundleContext.ungetService(entry.getKey());
        entry=null;
        iterator.remove();
    }
}

public void start(BundleContext bundleContext) throws Exception {
    this.bundleContext=bundleContext;
    services=new HashMap<ServiceReference, VendorService>();
    fillServices(); // synchronized (services) is not necessary
    bundleContext.addServiceListener(this, FILTERSTR);
}

```

```

    }

    public void stop(BundleContext bundleContext) throws Exception {
        bundleContext.removeServiceListener(this);
        releaseServices(); // synchronized (services) is not necessary
    }

    public void serviceChanged(ServiceEvent serviceEvent) {
        ServiceReference serviceReference=serviceEvent.getServiceReference();
        switch (serviceEvent.getType()) {
            case ServiceEvent.REGISTERED:
                synchronized (services) {
                    services.put(serviceReference, (VendorService) bundleContext.getServ
                }
                break;

            case ServiceEvent.UNREGISTERING:
                synchronized (services) {
                    services.remove(serviceReference);
                }
                bundleContext.ungetService(serviceReference);
                break;

            case ServiceEvent.MODIFIED:
                ... et c'est pas fini ...
                break;
        }
    }
}

```

Remarque: l'accès au champs *services* doit être synchronisé !!!!!

Compilez, conditionnez, déposez le bundle `customer.simple` avec :

```
cd customer.simple
ant install
```

Déployez et démarrez ce bundle sur la plateforme Felix avec:

```
start customer.simple-0.1.0.jar
```

Recherche de service pour les clients (avec ServiceTracker)

`org.osgi.util.tracker.ServiceTracker`[6]

(<http://www2.osgi.org/javadoc/r4/org/osgi/util/tracker/ServiceTracker.html>) est une classe utilitaire qui masque grandement la complexité d'un `ServiceListener`.

Son usage est illustré dans le bundle `customer.simple.st` et également dans le bundle `vendor.hotdog` qui en utilise un pour traquer les vendeurs de brioches et un pour traquer les vendeurs de saucisses.

Comme vous pouvez le constater dans l'extrait de code suivant tiré de `customer.simple.st`, le code s'est allégé par rapport à l'étape précédente

```

private ServiceTracker serviceTracker;

public void start(BundleContext bundleContext) throws Exception {
    String filterStrWithObjectclass = "("+Constants.OBJECTCLASS+"="+VendorService.class.getName()+")";
    serviceTracker=new ServiceTracker(bundleContext,bundleContext.createFilter(filterStrWithObjectclass));
    serviceTracker.open();
}

public void stop(BundleContext bundleContext) throws Exception {
    serviceTracker.close();
}

public void ...() {
    Object[] services=(Object[]) serviceTracker.getServices();

    if(services!=null && services.length!=0) {
        for (int i = 0; i < services.length; i++) {
            VendorService vendorService=(VendorService)services[i];
            vendorService.buy();
        }
    }
}
}

```

Compilez, conditionnez, déposez le bundle `acustomer.simple.st` avec :

```
cd customer.simple.st
ant install
```

Déployez et démarrez ce bundle sur la plateforme Felix avec:

```
obr start "OSGi R4 Compendium Bundle"
start customer.simple.st-0.1.0.jar
```

L'interface `org.osgi.util.tracker.ServiceTrackerCustomizer`[7] (<http://www2.osgi.org/javadoc/r4/org/osgi/util/tracker/ServiceTrackerCustomizer.html>) permet a un objet `ServiceTracker` de personnaliser la traque des services traqués.

Exercice: ajoutez un `org.osgi.util.tracker.ServiceTrackerCustomizer` pour tracer les ajouts et les retraits de liaisons.

Fabrique d'instance de services

L'interface `org.osgi.framework.ServiceFactory`[8] (<http://www2.osgi.org/javadoc/r4/org/osgi/framework/ServiceFactory.html>) permet d'implémenter le patron de conception *Fabrique* (*Factory* de E. Gamma). Cette interface comporte 2 méthodes :

- Object `getService(Bundle bundle, ServiceRegistration registration)`
- void `ungetService(Bundle bundle, ServiceRegistration registration, Object service)`

La méthode `getService()` fabrique une instance quand le bundle demandeur invoque la méthode object `getService(ServiceReference reference)` sur le bundle context. Ainsi plusieurs différentes instances remplissent le même service pour des clients (requester) différents. Les utilisations possibles du `ServiceFactory` sont:

- Gestion de sessions multiples (comme dans des GUI avec Multi-fenêtrages, des shells à sessions multiples, ...)
- Contrôle d'accès en fonction du bundle Requester (le bundle réclamant le service est connu)
- Personnalisation en fonction du bundle Requester (le bundle réclamant le service est connu)
- Si le service manipule son `ServiceRegistration`

Un exemple d'usage de `ServiceFactory` est le suivant: `vendor.weiner`.

```
package eosgi.snackbar.bundle.vendor.weiner.impl;

import java.util.Dictionary;
import java.util.Hashtable;

import org.osgi.framework.Bundle;
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.framework.ServiceFactory;
import org.osgi.framework.ServiceRegistration;

import eosgi.snackbar.service.vendor.VendorService;

/**
 * A simple service implementation for the SnackBar application (using ServiceFactory)
 * @version 0.1.0
 */
public class Vendor implements VendorService, BundleActivator, ServiceFactory {

    private BundleContext bundleContext;
    private ServiceRegistration serviceRegistration;

    public String buy() {
        return null;
    }

    public String getName() {
        return null;
    }

    public void start(BundleContext bundleContext) throws Exception {
        this.bundleContext=bundleContext;
        Dictionary serviceRegistrationProperties=new Hashtable();
        serviceRegistrationProperties.put(TYPE, "weiner");
        serviceRegistration=bundleContext.registerService(VendorService.class.getName(), this, servi
```

```
}  
public void stop(BundleContext bundleContext) throws Exception {  
    serviceRegistration.unregister();  
}  
public Object getService(Bundle bundle, ServiceRegistration serviceRegistration) {  
    return new VendorDelegate(bundleContext, bundle.getId());  
}  
public void ungetService(Bundle bundle, ServiceRegistration serviceRegistration, Object object)  
{  
}  
class VendorDelegate implements VendorService {  
    private BundleContext bundleContext;  
    private long bundleId;  
    public VendorDelegate(BundleContext bundleContext, long bundleId) {  
        this.bundleContext=bundleContext;  
        this.bundleId=bundleId;  
    }  
    public String buy() {  
        System.out.println(  
            "Bundle "+bundleContext.getBundle().getId()+" : "  
            +"Bundle "+bundleId+" buys "+ "Weiner(1)");  
        return "Weiner(1)";  
    }  
    public String getName() {  
        return "Weiner vendor";  
    }  
}  
}
```

Exercice : Modifiez ce service pour refuser tous les achats venant de bundles dont le Bundle-SymbolicName contient *customer* ! (ce service ne sert que des revendeurs !!)

Que se passe avec les instances au moment des getService() et ungetService() ?

Testez le avec la commande `snackbar` ?

Récupérée de « <http://localhost/eOSGi/index.php?title=ICAR-08-TP2> »

ICAR-08-TP3

Un article de EOSGi.

Utilisation des quelques services standards

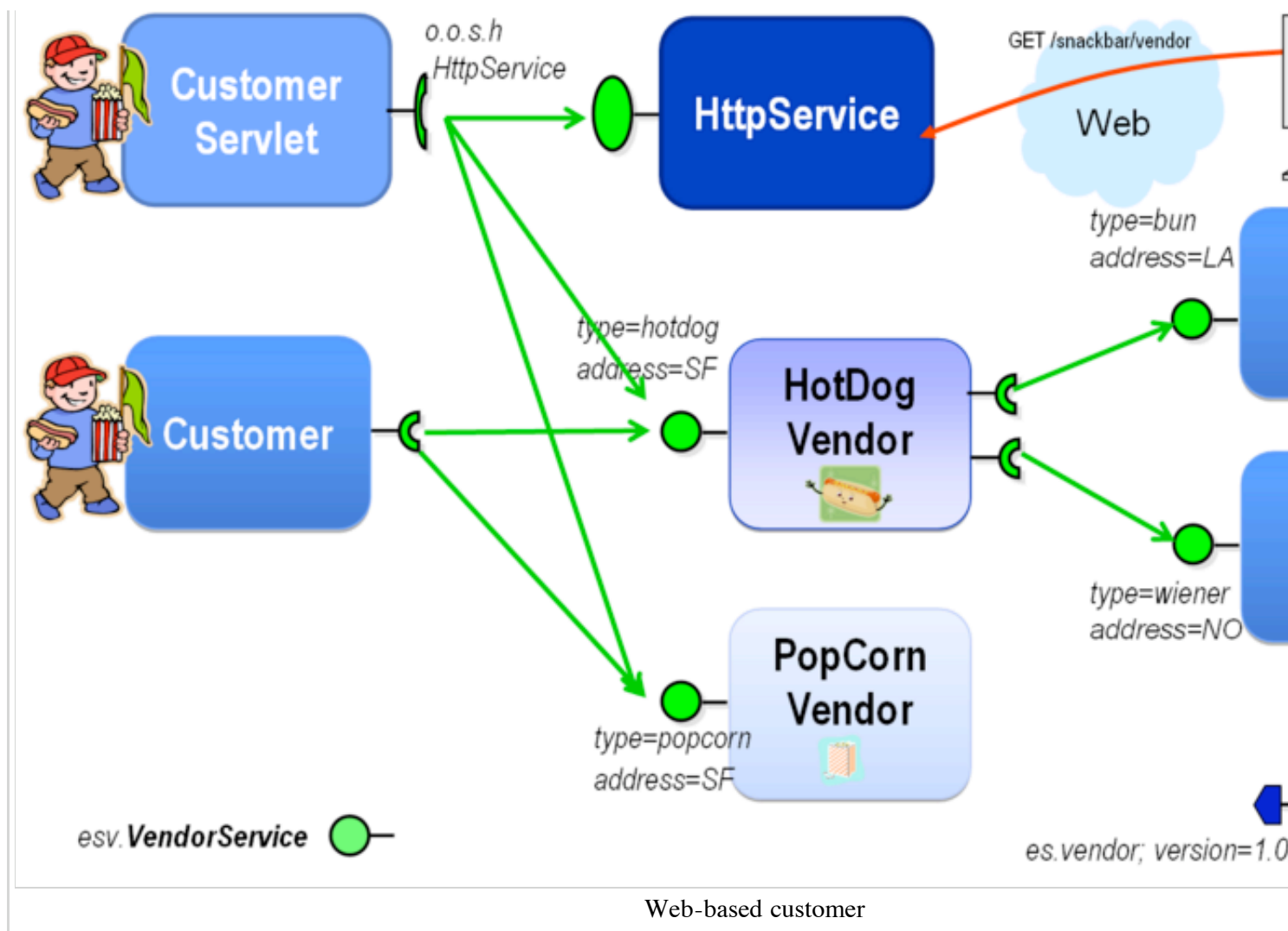
Sommaire

- 1 Le service HttpService
- 2 Le service de journalisation
 - 2.1 LogService
 - 2.2 LogListener

Le service HttpService

Dans le cadre de cet exercice, on souhaite compléter l'application Snackbar par un bundle permettant à un client d'acheter des produits aux vendeurs via un navigateur Web.

La nouvelle application est illustrée par le schéma suivant:



Pour concevoir ce bundle, vous devrez compléter le code source de la servlet `HttpCustomer` contenue dans le bundle `customer.http`, puis déployer le service implémentation `HttpService` (<http://www2.osgi.org/javadoc/r4/org/osgi/service/http/package-summary.html>) sur votre passerelle, et utiliser l'API Servlet pour mettre en oeuvre votre bundle. Celui-ci devra être capable de répondre à des requêtes HTTP de type GET sur l'URL [1] (<http://localhost:8080/snackbar/customer>).

Le service de journalisation

Le service de journalisation `LogService` (<http://www2.osgi.org/javadoc/r4/org/osgi/service/log/package-summary.html>) permet d'enregistrer des événements, des alarmes, des traces se produisant au cours de la vie d'un bundle ou d'un service. Dans la spécification OSGi release 4, la journalisation repose sur 4 types d'éléments, à savoir :

- `LogService`, qui est l'interface de service qui permet à un bundle de journaliser une information composée d'un message, d'un niveau de journalisation, d'une exception, d'un objet de type `ServiceReference`,
- `LogEntry`, qui est l'interface définissant une entrée d'un journal,
- `LogListener`, qui est l'interface à implanter afin de pouvoir être mis à l'écoute de la création de nouveaux objets `LogEntry`,
- `LogReaderService`, qui est l'interface de service permettant d'accéder à la liste des objets de type

LogEntry et d'enregistrer des objets de type LogListener, objets qui seront invoqués lors de création de nouveaux objets LogEntry,

LogService

L'exercice consiste premièrement à remplacer les *horribles* `System.out.println(...)` et `System.err.println(...)` qui se trouve un peu partout dans les bundles. Pour cela, il vous faudra vous lier au service `org.osgi.service.log.LogService` au moyen d'un `ServiceTracker`.

Par exemple avec `ServiceTracker` :

```
private ServiceTracker logServiceTracker;

public void start(BundleContext bundleContext) throws Exception {
    logServiceTracker=new ServiceTracker(bundleContext,LogService.class.getName(),null);
    logServiceTracker.open();
}

public void stop(BundleContext bundleContext) throws Exception {
    logServiceTracker.close();
}

public void ...(){
    if(error){
        LogService logService = (LogService) logServiceTracker.getService();
        if(logService==null) return;
        logService.log(LogService.LOG_ERROR, message);
    }
}
```

Attention, n'oubliez pas d'ajouter `org.osgi.service.log` dans l'`Import-Package` du manifeste du bundle.

LogListener

Pour "écouter" les entrées d'un journal, il est nécessaire d'enregistrer un objet `org.osgi.service.log.LogListener` auprès du service `org.osgi.service.log.LogReaderService` du journal.

Un exemple de `LogListener` est illustré par le bundle `loglistener.console`. Les entrées écoutées sont affichées sur la console `System.out`.

```
import java.io.PrintStream;
import org.osgi.framework.BundleContext;
import org.osgi.service.component.ComponentContext;
import org.osgi.service.log.LogEntry;
import org.osgi.service.log.LogListener;
import org.osgi.service.log.LogReaderService;

/**
 * This class implements a LogListener which displays log entries on the standart output
 */
public class ConsoleLogListener implements LogListener {

    private LogFormatter logFormatter=new TextLogFormatter(); // XmlLogFormatter, HtmlLogFormatter, JSONL
    private PrintStream out;

    public void logged(LogEntry entry){
```

```
        out.print(logFormater.getLogEntry(entry));
    }
// lifecycle callback methods
public void activate(ComponentContext ctxt) {
    out.print(logFormater.getLogHead());
}

public void deactivate(ComponentContext ctxt) {
    out.print(logFormater.getLogTail());
    out=null;
}

// binding callback methods
public void bindLogReaderService(LogReaderService logReaderService) {
    logReaderService.addLogListener(this);
}

public void unbindLogReaderService(LogReaderService logReaderService) {
    logReaderService.removeLogListener(this);
}
}
```

Récupérée de « <http://localhost/eOSGi/index.php?title=ICAR-08-TP3> »