CENTRE D'INNOVATION
EN TELECOMMUNICATION
& INTEGRATION DE SERVICES

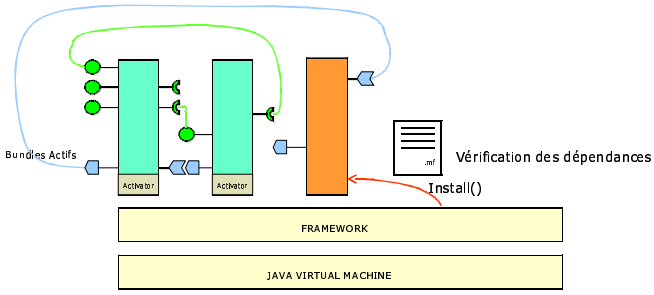
Programmation orientée services sous OSGi

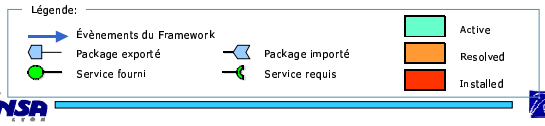
ARÈS

Le plan

- Définition de service
  - Utilisation
  - Déclaration
- BundleContext
  - Resources
- Evénements

Scénario: Arrivée d'un Bundle





La classe Activator du bundle

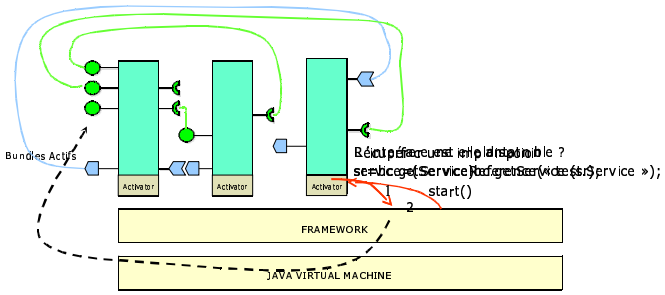
- Classe publique
  - Implémente les 2 méthodes start() et stop() de BundleActivator
  - qui reçoivent une référence sur un contexte.
- *start(BundleContext ctxt)*
  - recherche et obtient des services requis auprès du contexte et/ou positionne des listeners sur des événements
  - enregistre les services fournis auprès du contexte
- *stop(BundleContext ctxt)*
  - désenregistre les services fournis
  - relâche les services requis
    - Cependant le FW fait ces opérations si stop() en oublie !
- Classe définie dans le manifest pour « démarrer » le **bundle**

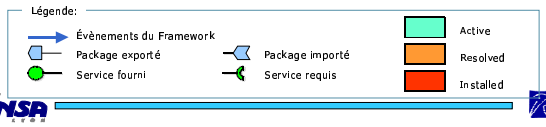
Services

- Le service est la brique de la programmation orientée Service (SOP)
- Il se décompose en deux parties :
  - Une interface de description.
    - Dans le cas d'OSGi c'est une interface java
  - Une ou plusieurs classes d'implantations

  * La description d'un service n'est pas liée à un bundle particulier
- Pour **utiliser** un service
  - Il faut demander au framework s'il existe un service qui met en œuvre une interface particulière (interface==service)
  - Dans une seconde étape demander à obtenir une implantation de ce service
- Pour **mettre** à disposition un service
  - Il faut fabriquer une instance d'implantation du service
  - Mettre à disposition du framework cette implantation sous l'interface voulue
- L'association entre deux services n'est pas gérée par le framework
  - Soit contrôle systématique, soit événement

Scénario: Utilisation d'un service





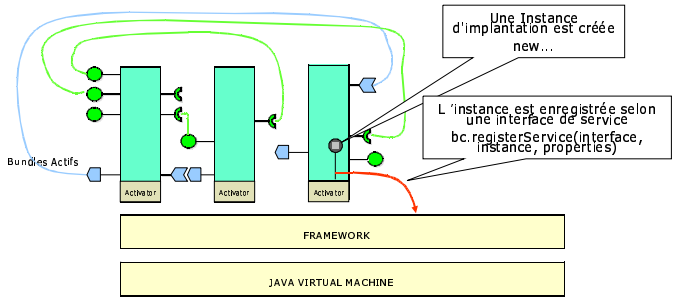
Code d'exploitation d'un service

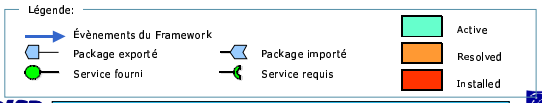
```
package useurlservice;

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

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.URL;

import org.ungoverned.oscar.bundle.url.URLService;

public class UseURLService implements BundleActivator {
  public void start(BundleContext bc) throws Exception {
    ServiceReference sr=bc.getServiceReference("org.oscar.bundle.url.URLService");
    if (sr!=null){
      URLService urls=(URLService)bc.getService(sr);
      if (urls !=null){
...
```


Code d'exploitation d'un service

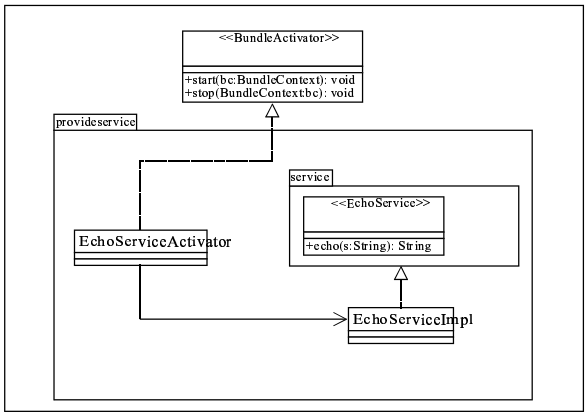
```
      BufferedReader br=new BufferedReader(
                              new InputStreamReader(urls.getURLInputStream(
                                                    new URL("http://telecom.insa-lyon.fr"))));
      String t=br.readLine();
      while (t!=null){
        System.out.println(">> "+t.toString());
        t=br.readLine();
      }
     }else{
      System.out.println("Erreur, Impossible d'obtenir l'implantation");
     }
    }else{
     System.out.println("Erreur, impossible d'obtenir le service");
    }
  }
  public void stop(BundleContext bc){
    System.out.println("Au revoir");
  }
}
```

Scénario: Mise à disposition d'un service





Exemple de service : echo



Enregistrement du service

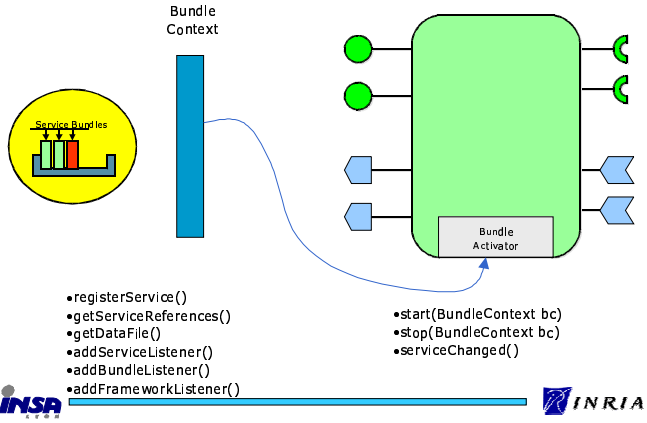
```
package provideservice;
import org.osgi.framework.BundleActivator; import org.osgi.framework.BundleContext;
import org.osgi.framework.ServiceRegistration;

public class EchoServiceActivator implements BundleActivator {
  ServiceRegistration sr;
  EchoServiceImpl es;

  public void start(BundleContext bc){
    es=new EchoServiceImpl();
    sr=bc.registerService("provideservice.service.EchoService", es, null);
  }
  public void stop(BundleContext bc){
    sr.unregister();
    es=null;
    System.out.println("Le service est désenregistré");
  }
}
```

BundleContext

- Interface vers le framework
  - Passé lors des invocations de start() et stop() de l'Activator
  - C'est la référence java vers le framework
- Permet
  - L'enregistrement de services
  - L'accès aux ressources du bundle
  - Le courtage de services
  - L'obtention et la libération des services
  - La souscription aux évènements du Framework.

  - *L'accès aux propriétés du framework*
  - *L'installation de nouveaux bundles*
  - *L'accès à la liste des bundles*

BundleContext et Activator



Accès aux ressources et aux fichiers

- Ressources
  - this.getClass().getRessourceAsStream(String path)
    - path="/" correspond à la racine du JAR (BUNDLE-CLASSPATH)
    - Liée à l'architecture des classloaders
- Support de persistance
  - BundleContext.getDataFile(String path)
    - path="" correspond à la racine du cache du bundle
  - FileService
    - Contrôle les permissions du Bundle au passage
  - Aussi
    - Bundle - public URL getResource(String name) throws IllegalStateException;

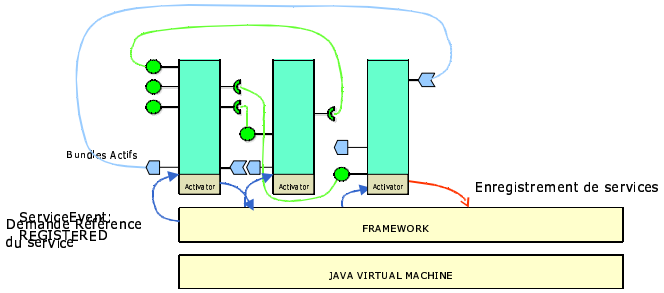
Interroger le bundlecontext

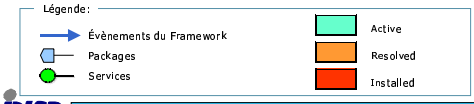
```
package org.osgi.framework;
public interface BundleContext {...
  public Bundle getBundle() throws IllegalStateException;
  public Bundle getBundle(long id);
  public Bundle[] getBundles();
  public File getDataFile(String filename) throws IllegalStateException;
  public String getProperty(String key);
  public Bundle installBundle(String location) throws BundleException,
     SecurityException;
  public Bundle installBundle(String location, InputStream in) throws BundleException;
...}
```


Evénements

- Les services dépendent les uns des autres
- L'environnement est dynamique
  - Un service n'est pas lié de manière persistante à un autre
  - Un mécanisme de gestion d'événements permet de suivre le cycle de vie des services

Evénements





Événements

- ServiceEvent
  - Notifie l'enregistrement ou le retrait de services
  - interface `ServiceListener` méthode `serviceChanged`
    - Traitement séquentiel et synchrone des listeners
- FrameworkEvent
  - Notifie le démarrage et les erreurs du Framework
  - 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 changement d'état)

Interdépendance de services

- Si un service dépend d'un autre...
  - La plate-forme ne gère pas cette information
  - Un service, ne peut jamais supposer de l'existence d'un autre service, donc :
    - Soit il y a recherche du service à chaque invocation de la méthode (coût d'exécution, possible par ex. debug)
    - Soit le service est un « listener » des événements concernant l'autre service
  - Remarques :
    - on ne présuppose pas non plus de l'ordre de lancement des services
    - C'est au client de gérer les références mortes

Enregistrement de services avec des propriétés

```
package com.lexmark.printer.laser.impl;
public class Activator implements BundleActivator {
  private ServiceRegistration reg=null;
  private PrintService theService=null;
  public void start(BundleContext ctxt) throws BundleException {
    theService=new PrintServiceImpl();
    Properties props=new Properties();
    props.put("type", "laser");
    props.put("dpi", "72,150,300,600,1200");
    props.put("location", "1st floor");
    reg=ctxt.registerService("org.device.print.PrintService", theService, props);
  }
  public void stop(BundleContext ctxt) throws BundleException {
    if(reg != null) reg.unregister();
  }
}
```

Recherche (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`
  - Attribut insensible aux majuscules/minuscules
  - L'attribut `objectClass` représente le nom du service
  - Opérateurs `>=, <=, =, ~=` (approximativement égal), `=*` (présent)
  - Connecteurs logiques `&, |, !`

Recherche de services

- Tous les services d'impression

```
refs=bundleContext.getServiceReferences(
    "org.device.print.PrintService", null);
refs=bundleContext.getServiceReferences(
    null, "(objectClass=org.device.print.PrintService)");
```

- Certains services d'impression

```
refs=bundleContext.getServiceReferences(
    "org.device.print.PrintService",
    "(&(!(type=laser))(capability=double-sided)(!(dpi<=300))(location=*)" );
```

- Tous les services de org.device

```
refs=bundleContext.getServiceReferences(null,
    "(objectClass=org.device.*)");
```

- Le service d'impression et de fax au 3ième étage

```
refs=bundleContext.getServiceReferences(null,
    "(&(objectClass=org.device.print.PrintService)" +
    "(objectClass= org.device.fax.FaxService)" +
    "(location=4th floor))" );
```

Fonctions de filtrage

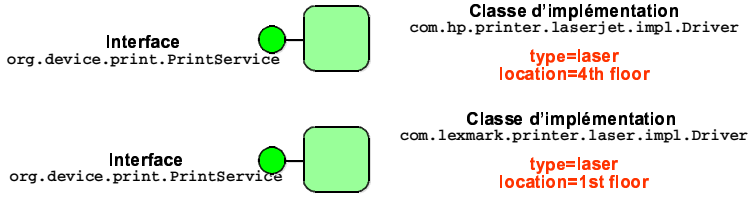
```
public interface BundleContext {…
  public Filter createFilter(String filter) throws InvalidSyntaxException;
  public ServiceRegistration registerService (String[] clazzes, Object service, Dictionnary
     properties) throws IllegalArgumentException, SecurityException, IllegalStateException;
…
}
```

```
public interface Filter {
  public boolean equals(Object obj);
  public int hashCode();
  public boolean match(Dictionnary dictionnary) throws IllegalArgumentException;
  public boolean match(ServiceReference reference);
  public String toString();
}
```


Un exemple plus complet ...

Service

- Une interface et des implémentations
  - se trouvent dans des packages différents
  - implémentation normalement non publique.
  - multiples implémentations possibles
    « emballées » dans les bundles.
- Qualifié par des propriétés.

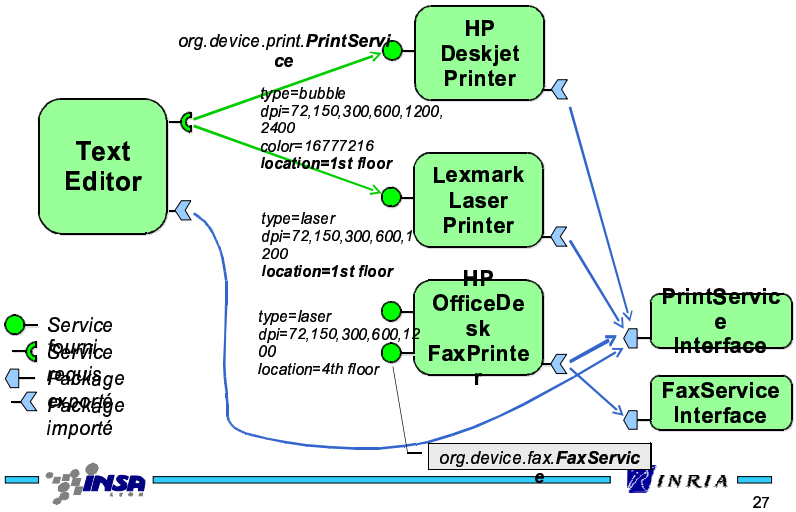


Exemple de service

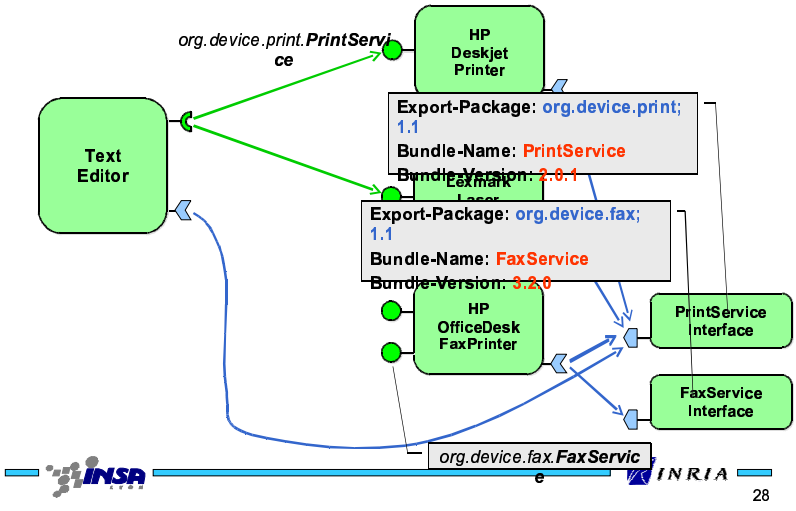
Interface
org.device.print.PrintService

```
package org.device.print;

public interface PrintService {

     public int print(OutputStream out,
                      String[] printparams)
            throws PrintException;

     public void kill(int jobnumber)
            throws PrintException;

     public Job[] list()
            throws PrintException;

     public Job status(int jobnumber)
            throws PrintException;

}
```

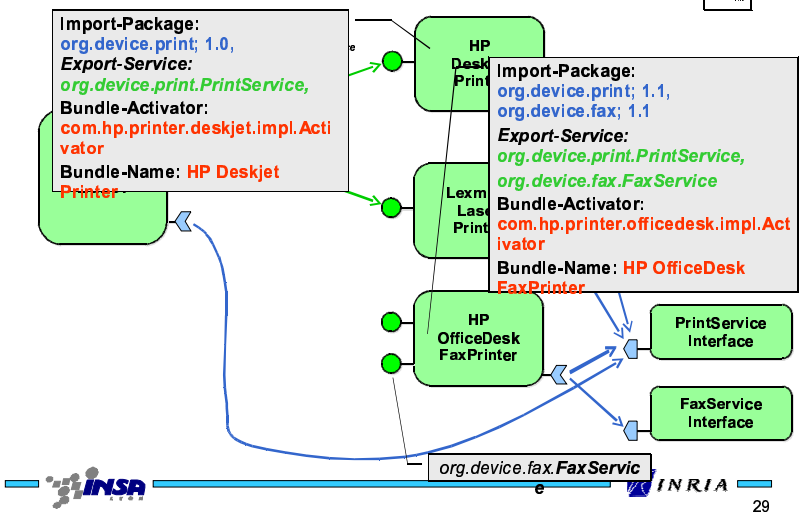
Exemple d'application



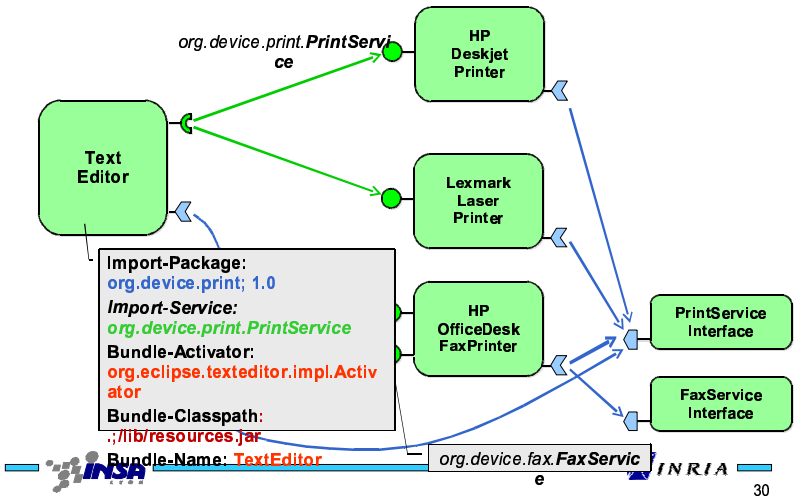
Exemple de manifest (i)



Exemple de manifest (ii)



Exemple de manifest (iii)



Enregistrement de services (Lexmark Laser Printer)

```
package com.lexmark.printer.laser.impl;
public class Activator implements BundleActivator {
  private ServiceRegistration reg=null;
  private PrintService theService=null;
  public void start(BundleContext ctxt) throws BundleException {
    theService=new PrintServiceImpl();
    Properties props=new Properties();
    props.put("type", "laser");
    props.put("dpi", "72,150,300,600,1200");
    props.put("location", "1st floor");
    reg=ctxt.registerService("org.device.print.PrintService", theService, props);
  }
  public void stop(BundleContext ctxt) throws BundleException {
    if(reg != null) reg.unregister();
  }
}
```


Recherche de services (TextEditor)

```
package org.eclipse.texteditor.impl
import org.device.print.PrintService;
class Activator implements BundleActivator {
    public void start(BundleContext ctxt) throws BundleException {
            private PrintService ser;
            // On va voir si quelqu'un offre un PrintService ...
            ServiceReference[] tempRefs
                        =ctxt.getServiceReferences
                        ("org.device.print.PrintService","(location=1st floor)");
            if(tempRefs!=null) {
               System.out.println("Found a PrintService! I will use it!!!");
               // On prend le premier offert!
               ser=(PrintService) ctxt.getService(tempRefs[0]);
            }
            ...
    }
    ...
}
```

Prendre en compte l'enregistrement et le retrait de service (i)

- Les bundles « requesters » doivent impérativement prendre en compte l'enregistrement et le retrait de services « importés »
- Exemple

```
public class PrintListenerActivator implements BundleActivator {

  public void start(BundleContext context) {
    // this bundle is interested in all events in the framework
    PrintServiceListener listener = new PrintServiceListener(context);
    context.addServiceListener(listener);
  }
  public void stop(BundleContext context) {
    // we don't have to do any thing because the framework is
    // going to remove all our listeners
  }
}
```


Prendre en compte l'enregistrement et le retrait de service (ii)

```
class PrintServiceListener implements ServiceListener {
  public void serviceChanged(ServiceEvent e) {
    ServiceReference ref = e.getServiceReference();
  if(((String)ref.getProperty("objectClass").equals("org.device.print.PrintService"))){
      switch (e.getType()) {
        case ServiceEvent.REGISTERED:
          println(ref + " has been registered by "+ ref.getBundle().getLocation()); break;
        case ServiceEvent.UNREGISTERING:
          println(ref + " is being unregistered"); break;
      case ServiceEvent.MODIFIED:
          println("properties of "+ref+" have been modified:");
          String[] keys = ref.getPropertyKeys();
          for (int i=0; i<keys.length; i++) println(keys[i] + "=" + ref.getProperty(keys[i])); break;
}}}
  void println(String msg) {System.out.println("events: "+msg); } }
```

Prendre en compte l'enregistrement et le retrait de service (iii)

```
public class Activator implements BundleActivator {
  final static String filterStr
    ="(&(objectClass=org.device.print.PrintService)(location=4th floor))";
  Map printservices; /*<ServiceReference,PrintService>*/
  BundleContext context;
  public void start(BundleContext context) throws BundleException {
    this.context=context;
    printservices=new HashMap();
    BindingController ctlr=new BindingController(context,filterStr,printservices);
    ctlr.fillServices();
    context.addServiceListener(ctlr);
  }
...
```


Prendre en compte l'enregistrement et le retrait de service (iv)

```
public class BindingController implements ServiceListener {
   Map services; /*<ServiceReference,Object>*/
   String filterStr;
   Filter filter;
   BundleContext context;

   public BindingController(BundleContext context, String filterStr, Map services){
      this.context=context;
      this.filterStr=filterStr;
      this.services=services;
      filter=context.createFilter(filterStr);
   }
```

Prendre en compte l'enregistrement et le retrait de service (v)

```
...
   // fill the services map
   public void fillServices() {
      // get references on service
      ServiceReference[] refs=context.getServiceReferences(null,filterStr);
      for(int i=0;i<refs.length;i++){
         Object svc = context.getService(refs[i]);
         if(svc!=null) services.put(refs[i],svc);
      }
   }
...
```


Prendre en compte l'enregistrement et le retrait de service (vi)

```
...
public void serviceChanged(ServiceEvent e) {
   ServiceReference servref = e.getServiceReference();
   Object ref;
   switch (e.getType()) {
   case ServiceEvent.REGISTERED:
      if(filter.match(servref)){
         println(servref + " (from "+ servref.getBundle().getLocation() + ") is added");
         services.put(servref,context.getService(servref));
      };
      break;
...
```

Prendre en compte l'enregistrement et le retrait de service (vii)

```
        ....
      case ServiceEvent.UNREGISTERING:
        ref=services.remove(servref);
        if(ref!=null) {
          println(servref + " is removed");
          context.ungetService(servref);
        } break;
      case ServiceEvent.MODIFIED:
        ref=services.get(servref);
        if(ref!=null && !filter.match(servref)){
          println(servref + " is removed since properties has changed");
          services.remove(servref);
          context.ungetService(servref);
        } break;
    }
}
```


Compléments...

MultiThreading

- Motivations
  - Un service peut utiliser des threads (en poll ou non) pour ses besoins
- Obligation
  - Ces threads doivent être terminées dans le stop() de l'activateur

```
public class PrimeSearchActivator implements BundleActivator, Runnable {
   PrimeSearch t;boolean cont;
   public void start(BundleContext context) {
     t=new PrimeSearchImpl();
     t.start();
   }
   public void stop(BundleContext context) { cont=false;t=null;  }

   public void run(){ while(cont) { /* compute primes */  } }
}
```


Sécurité

- Basé sur les permissions du JDK1.2
  - Le SecurityManager vérifie les permissions de chaque bundle (FilePermission, DialPermission, …)

```
java -Djava.security.manager -Djava.security.policy=java.policy ...
grant codeBase "file:/home/rickhall/projects/oscar/bundle/servicelookup.jar" {
        permission org.osgi.framework.ServicePermission "org.*", "get";};
```

  - Accès aux fichiers
    - Ceux du cache et aux ressources du bundle
- 3 permissions propres à OSGi
  - AdminPermission
    - Autorise l'accès aux fonctions d'administration du framework.
  - ServicePermission
    - Contrôle l'enregistrement et la récupération de services (GET)
  - PackagePermission
    - Contrôle l'import et l'export de packages (EXPORT/IMPORT)
- org.osgi.service.PermissionAdmin
  - Service de gestion des permissions des bundlesci

Interface AdminPermission

```
public final class AdminPermission extends java.security.BasicPermission {
  // Cycle de vie des bundle
  public AdminPermission();
  public AdminPermission(String name, String actions);
  public boolean equals (Object obj);
  public boolean implies(java.security.Permision p);
  public java.security.PermissionCollection newPermissionCollection();
}
```

```
public final class PackagePermision extends java.security.BasicPermission {
  public static final String EXPORT, IMPORT;
  ...
}
```

```
public final class ServicePermission extends java.security.BasicPermission {
  public static final String REGISTER, GET;
  ...
}
```


Interface Configurable

- Permet la configuration des services
  - (si AdminPermission)
- au travers d'un objet de configuration
  - Le service doit être une instance de l'interface Configurable
  - java.lang.Object getConfigurationObject()
- Remarque
  - L'objet de configuration peut être un JavaBean (setXXX/getXXX) pour faciliter l'introspection de l'objet de configuration par des « outils » (services) de configuration.

Exemple de Configuration

```
if(sv instanceof Configurable) {
  Object o=sv.getConfigurationObject();
  if(o instanceof SerialPortConfigurationObject) {
    SerialPortConfigurationObject cfg=(SerialPortConfigurationObject) o;
    System.out.println("Previous configuration : " + cfg.getPort()+ " "
                                    + cfg.getBaudRate() + " bauds");
    cfg.setPort("COM1");
    cfg.setBaudRate(4800);
  } else { ... }
} else { ... }
```


ServiceFactory

- Motivation
  - Patron de conception de la fabrique (Gamma)
  - Retourne une instance par client
    - Différentes instances pour un même service
    - Transparent pour le bundle client, uniquement utilisé par le framework
- Permet de donner la main au développeur pour gérer le cycle de vie du service indépendamment du cycle de vie du bundle

package org.osgi.framework

Interfaces
Bundle
BundleActivator
BundleContext
BundleListener
Configurable
Constants
Filter
FrameworkListener
ServiceFactory
ServiceListener
ServiceReference
ServiceRegistration
SynchronousBundleListener

Classes
AdminPermission
BundleEvent
FrameworkEvent
PackagePermission
ServiceEvent
ServicePermission
Exceptions
BundleException
InvalidSyntaxException