



OSGi in the home gateways

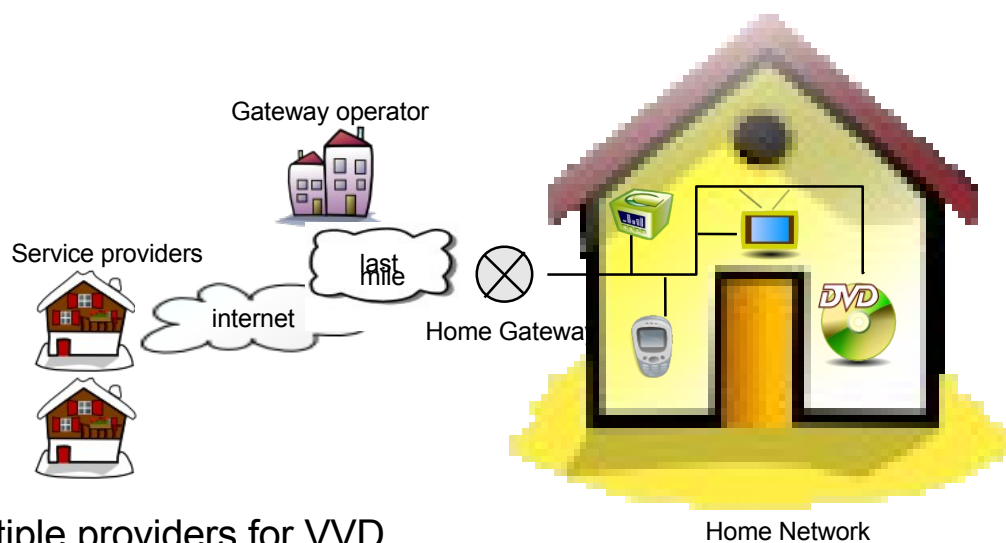


MUSE Winter School
BB Europe Antwerp
December 6th, 2007
Stéphane Frénot,
Thanks to Didier Donsez



Multi Service Access Everywhere

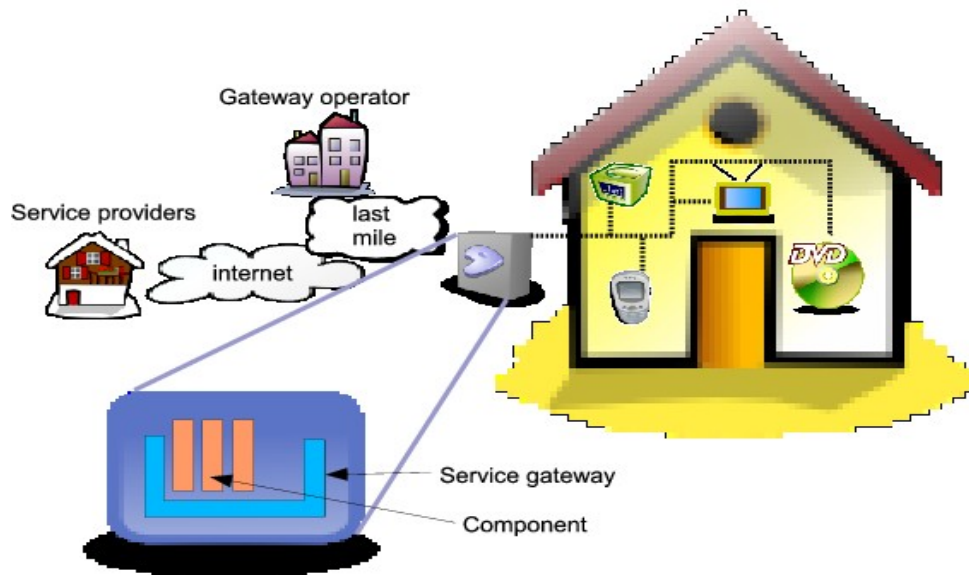
Multi-play



- > Multiple providers for VVD
- > The home user chooses on-demand

- > Drawbacks of multi-play
 - More and more dedicated set-top boxes
 - Complexity
 - Manufacturing and maintenance costs
 - Problems with standardisation
 - Many closed / proprietary solutions
 - Multimedia VV services only

- > Management/supervision services at home
 - Domotics
 - Heating, lights
 - Health care devices supervision
 - Pacemaker, elder people, quality of life
 - Whitegoods supervision
 - fridge, dishwasher...
- > Applicative services
 - Games
 - Graphical user interfaces for management
- > ==> Multi-service/multi-provider/multi-device



- > Hosts software services (any service)

- > Need for a single platform
 - Open and extensible (dynamic market)
- > Execution Environment
 - Dynamic and collaborative aspects of services
- > Management infrastructure
 - Local and remote management
 - Deployment
- > Move user-related intelligence to the border of the network

Why the Java/OSGi stack are promising ?



> Why Java

- It is object Oriented
- It has a huge standard API
- It is dynamic:
 - New objects can be used at run-time
 - New classes can be defined at run-time
- It is secured
 - Types are known at compile time and at run-time
 - A security Manager can control access to sensible methods

> Why OSGi

- It is component Based
- It has service-oriented programming features
 - => It is simplifying Java dynamicity integration
- It defines standard services

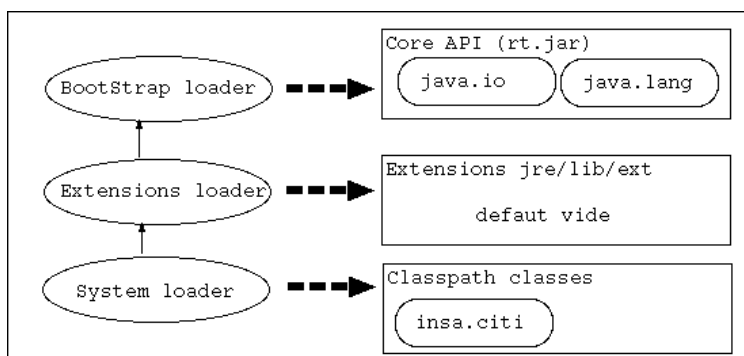


Why java is dynamic ?



> In basic java : **test.MyClass exemple=new test.MyClass();**

- Is controlled at compile time
- Is automatically controlled and loaded in the Java virtual machine at run-time
- Once loaded it is never contested



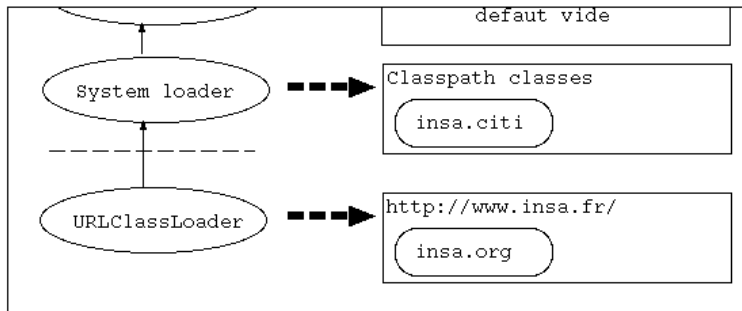
> THIS IS NOT DYNAMIC!! Everything is closed

Why java is dynamic ?



- > Because classloader architecture can be extended

```
URLClassLoader myCL = new URLClassLoader("http://www.somewhere.biz");
Class myClass = myCl.load("test.MyClass");
test.MyClass exemple = (test.MyClass)myClass.newInstance();
```



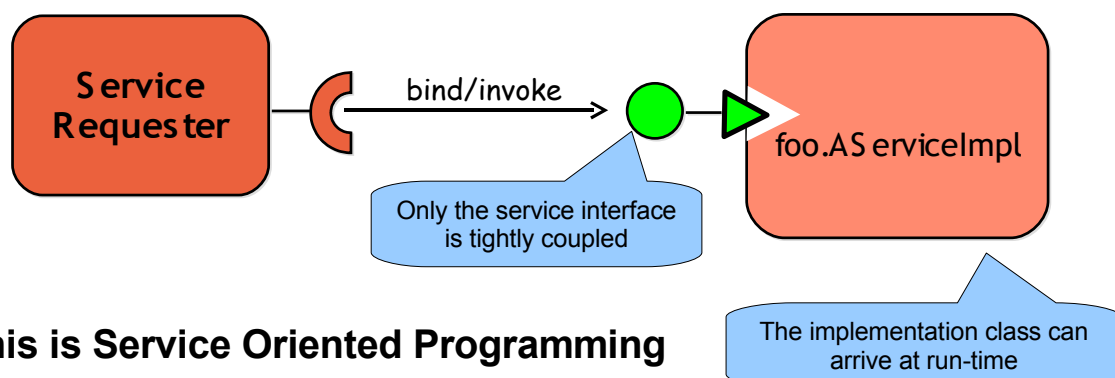
- > Classes byte-codes arrive at run-time (Applet, Java RMI)
- > A class is uniquely identified by its name **AND** the ClassLoader that defines it
- > THIS IS DEFINITELY STILL NOT DYNAMIC !!!

What is missing to be really dynamic ?

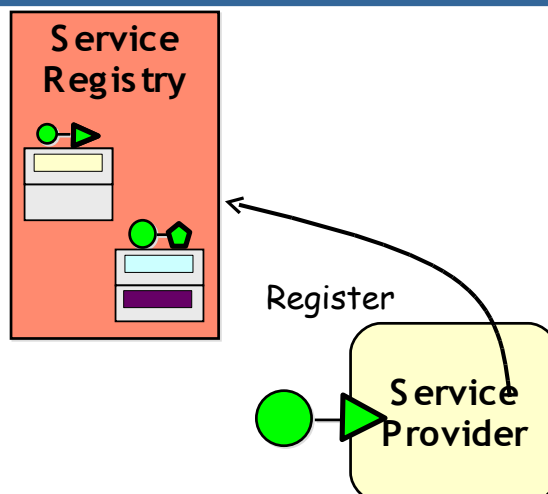
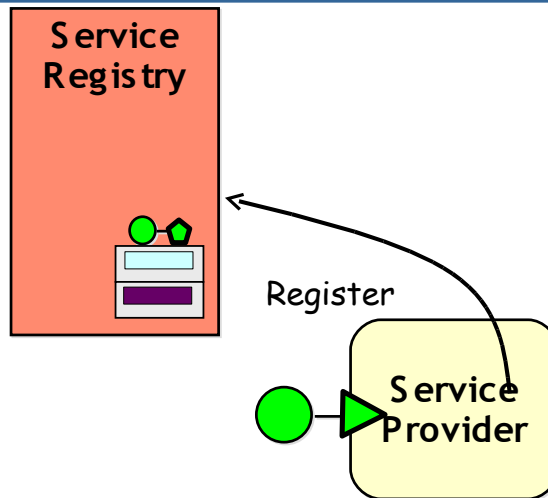


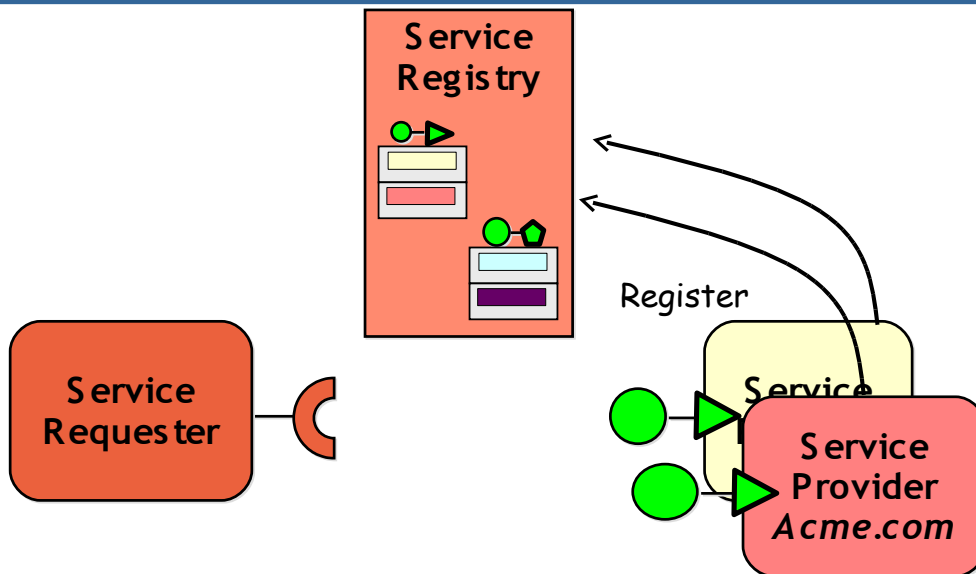
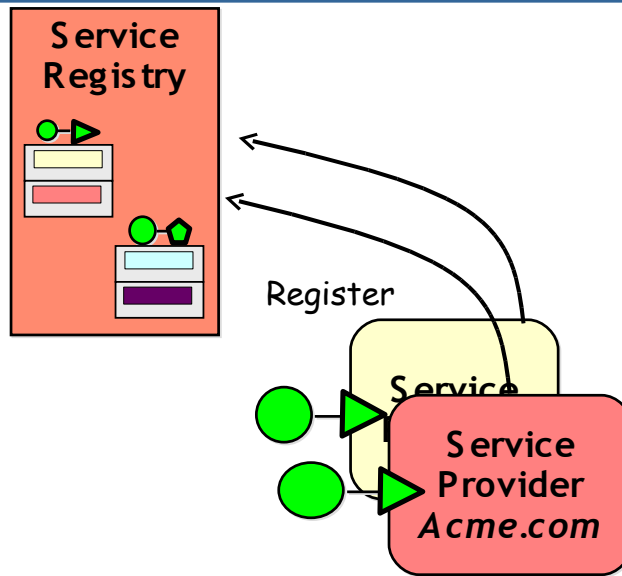
- > We need a loose coupling between the requester and the provider class

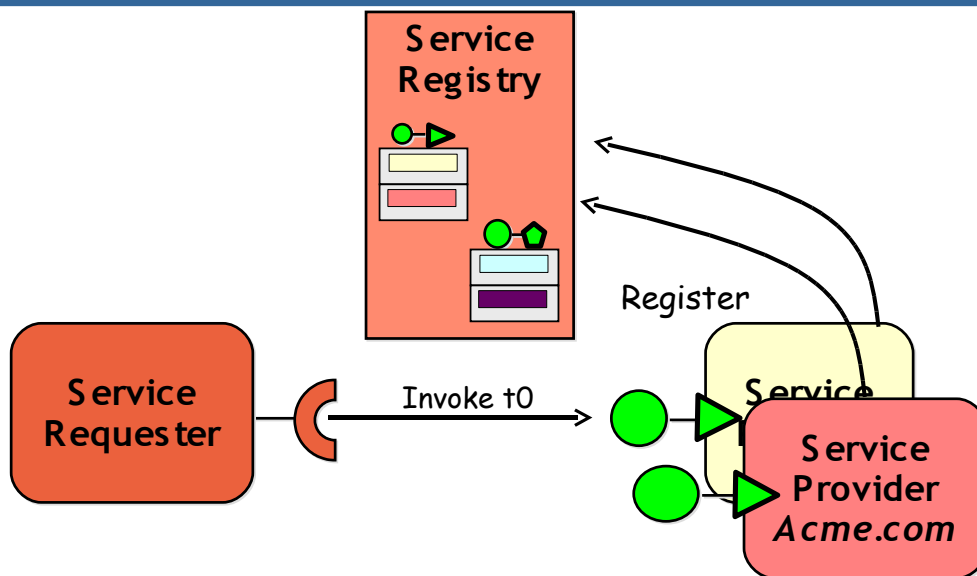
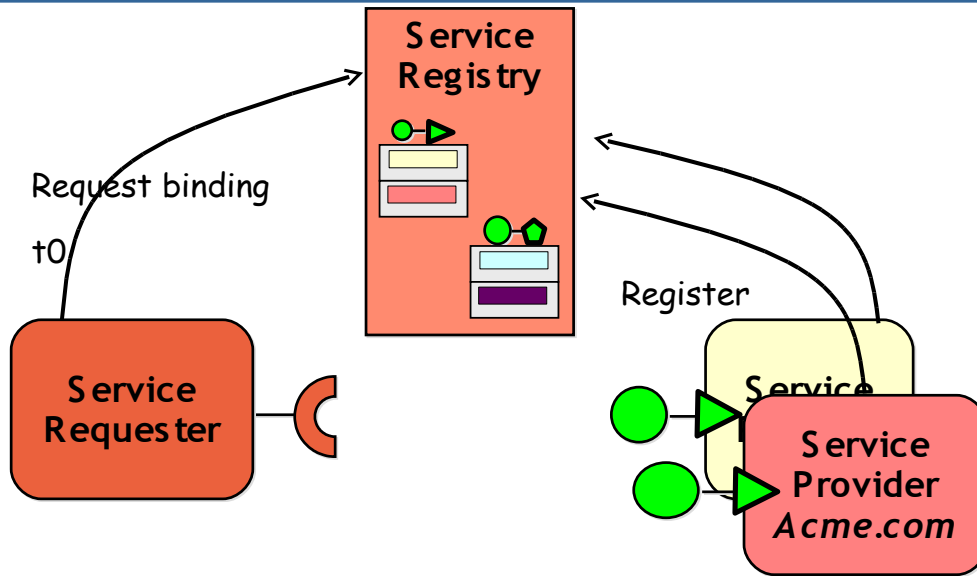
```
URLClassLoader myCL = new URLClassLoader("http://www.somewhere.biz");
Class myClass = myCl.load("foo.AServiceImpl");
test.Service exemple = (test.Service)myClass.newInstance();
```

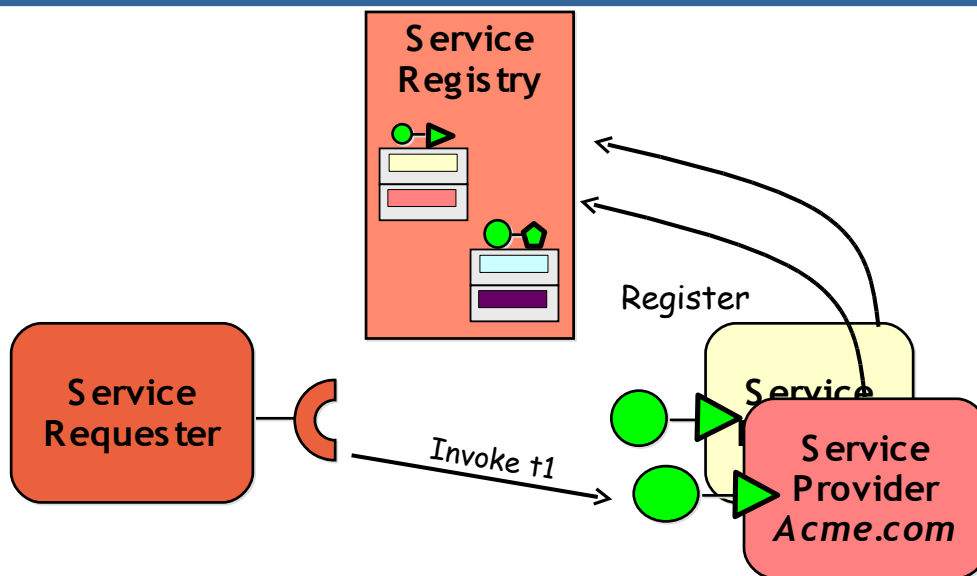
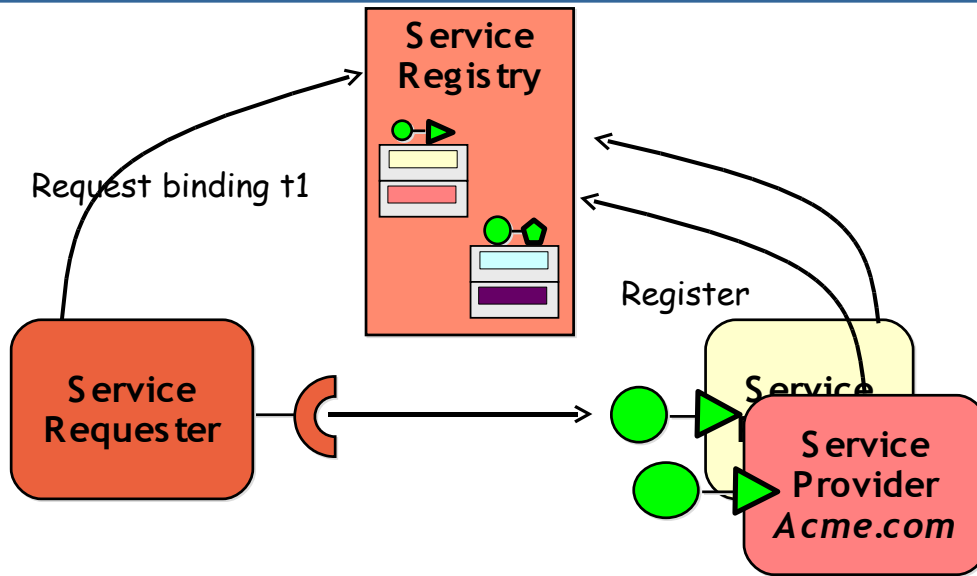


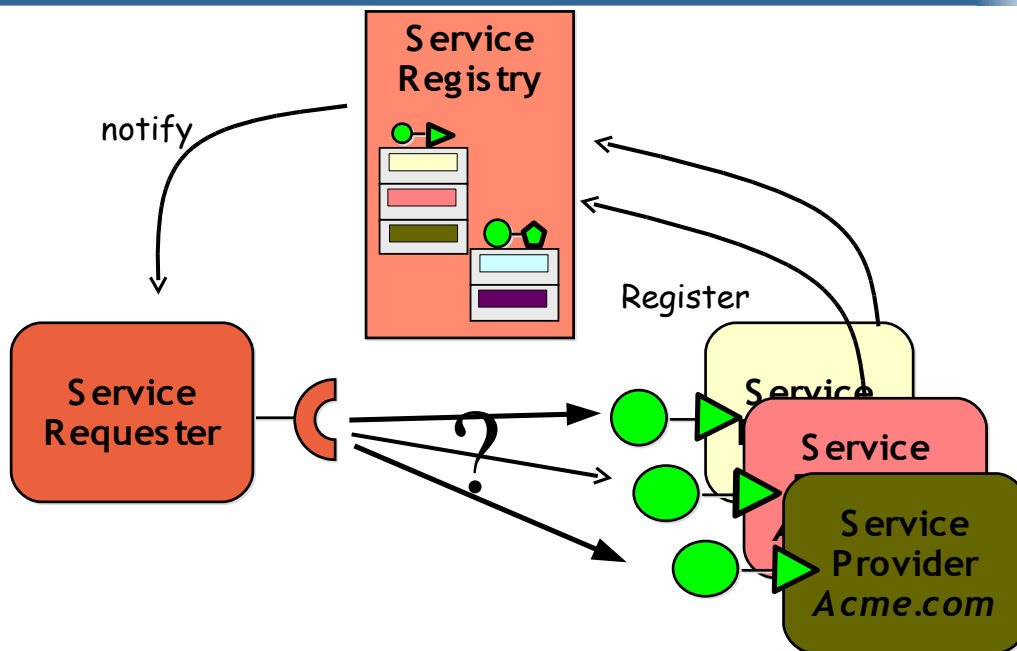
- > **This is Service Oriented Programming**











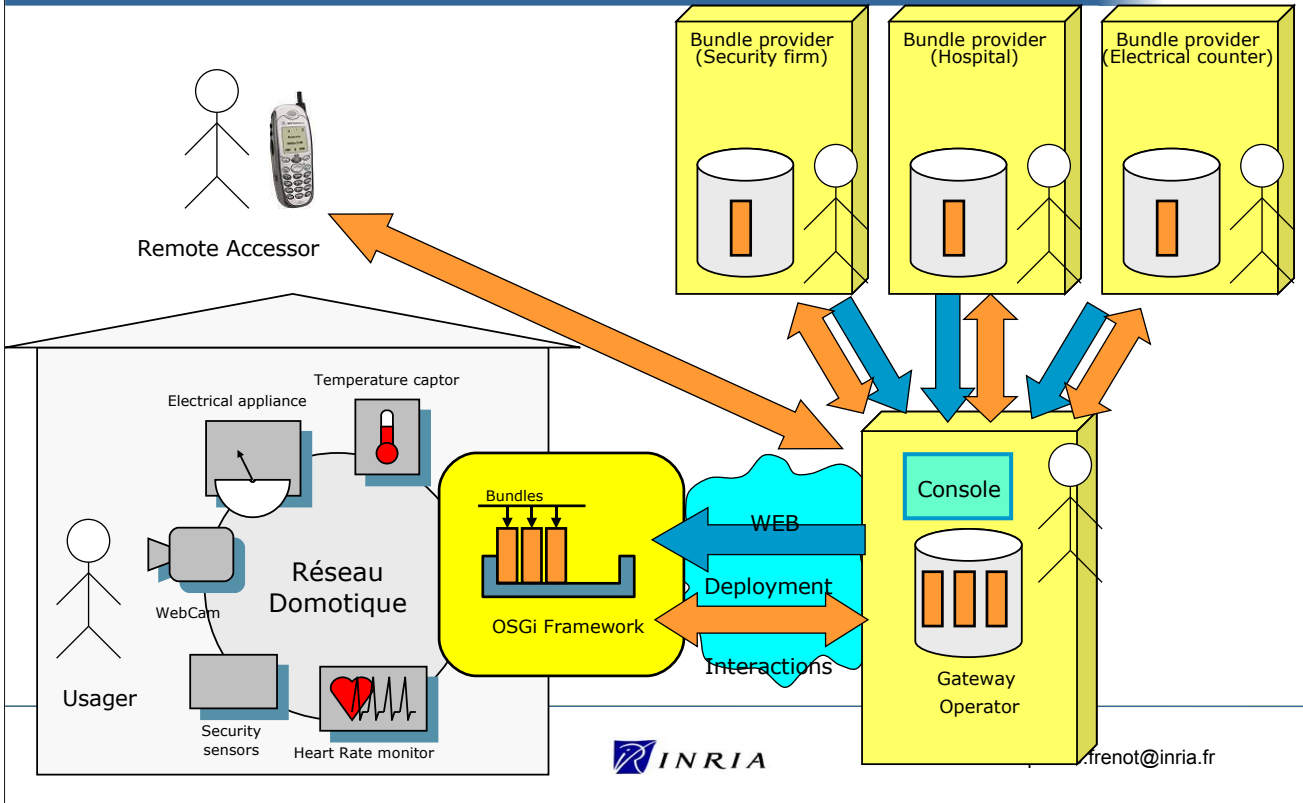
> Why Java

- It is object Oriented
- It has a huge standard API
- It is dynamic:
 - New objects can be used at run-time
 - New classes can be defined at run-time
- It is secured
 - Types are known at compile time and at run-time
 - A security Manager can control access to sensible methods

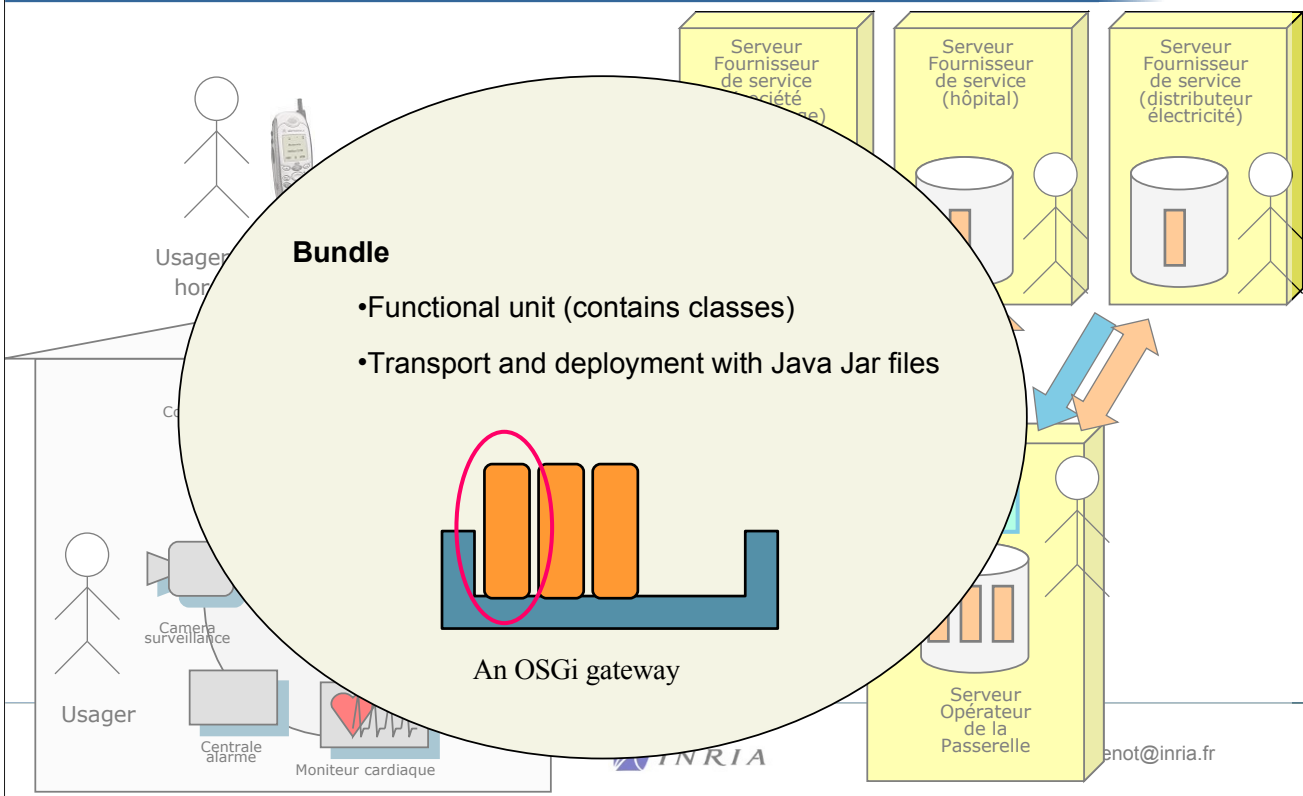
> Why OSGi

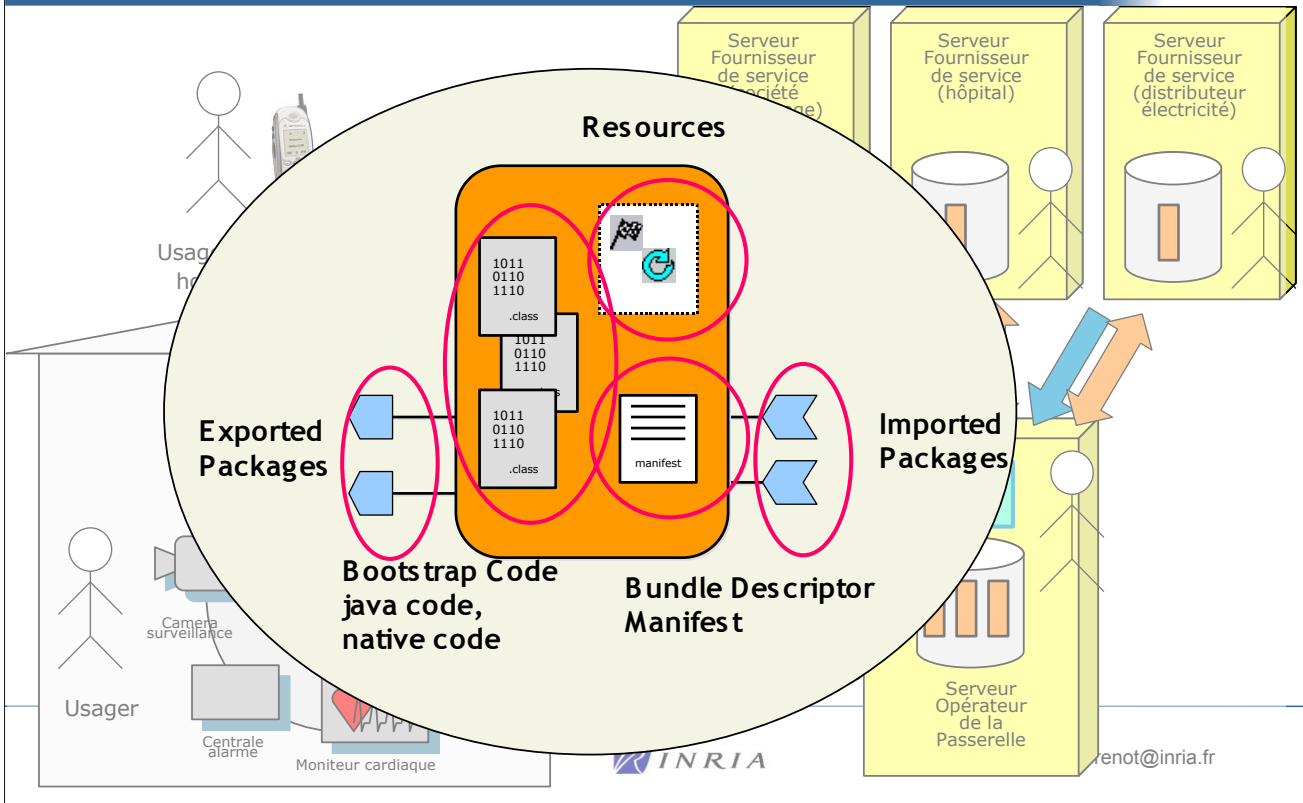
- It is component Based
- It has service-oriented programming features
 - => It is simplifying java dynamicity integration
- It defines standard services

Why is OSGi component based ?



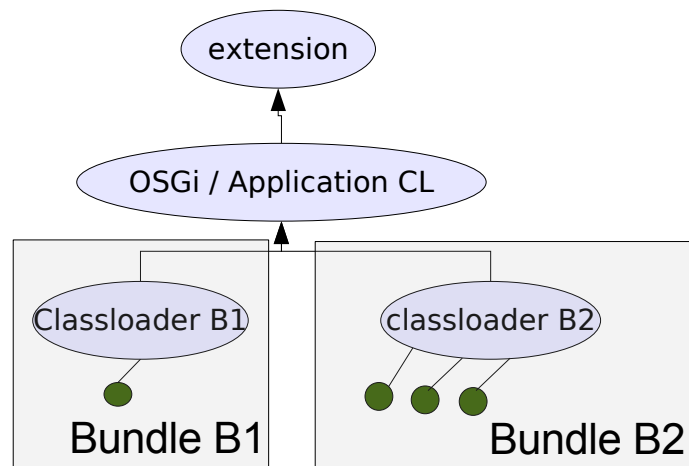
Components as Bundles





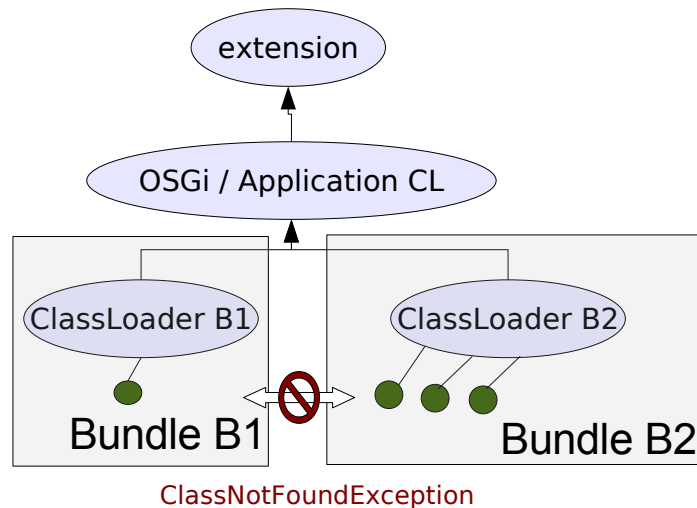
Bundle characteristics 1: A ClassLoading perspective

- > A bundle has a dedicated ClassLoader which means that:
 - Every class and resources it defines can be used
 - When removing the bundle all classes can be removed



> A bundle has a dedicated ClassLoader which means that:

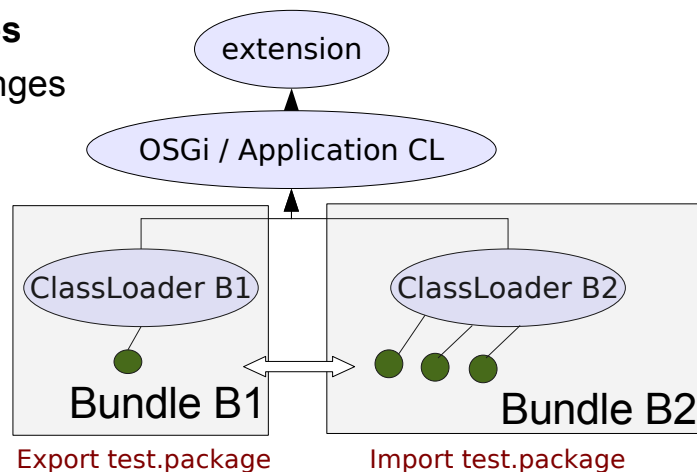
- Every class and resources it defines can be used
- When removing the bundle all classes can be removed



> A bundle has a dedicated ClassLoader which means that:

- Every class and resources it defines can be used
- When removing the bundle all classes can be removed
- **Import/Export of packages**

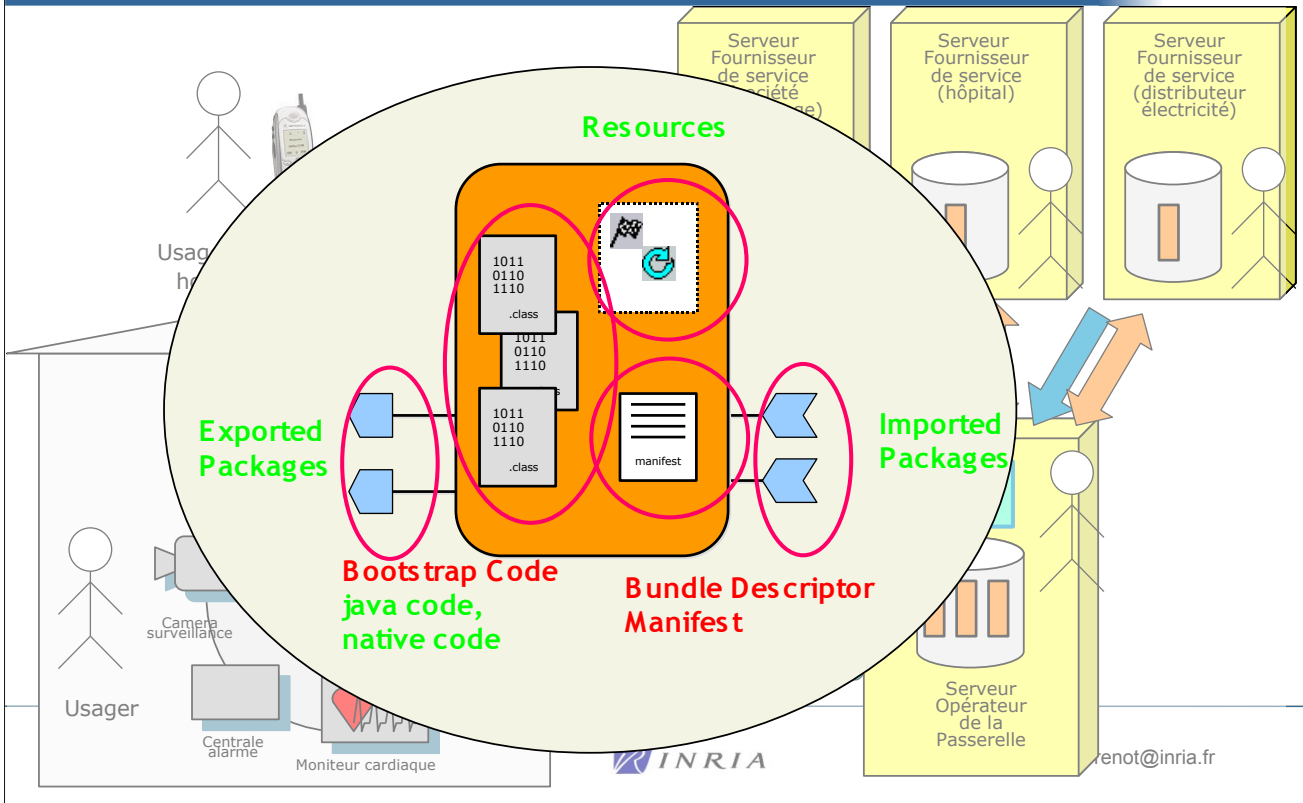
enables classes exchanges



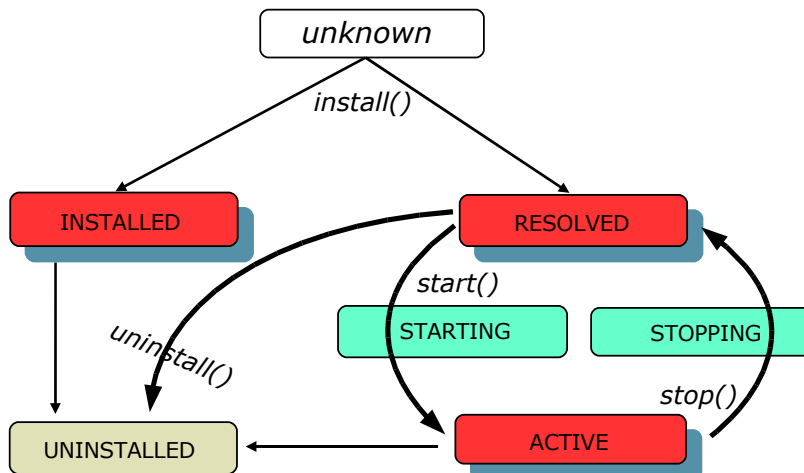
> This features guarantees,

that every needed class is resolved.

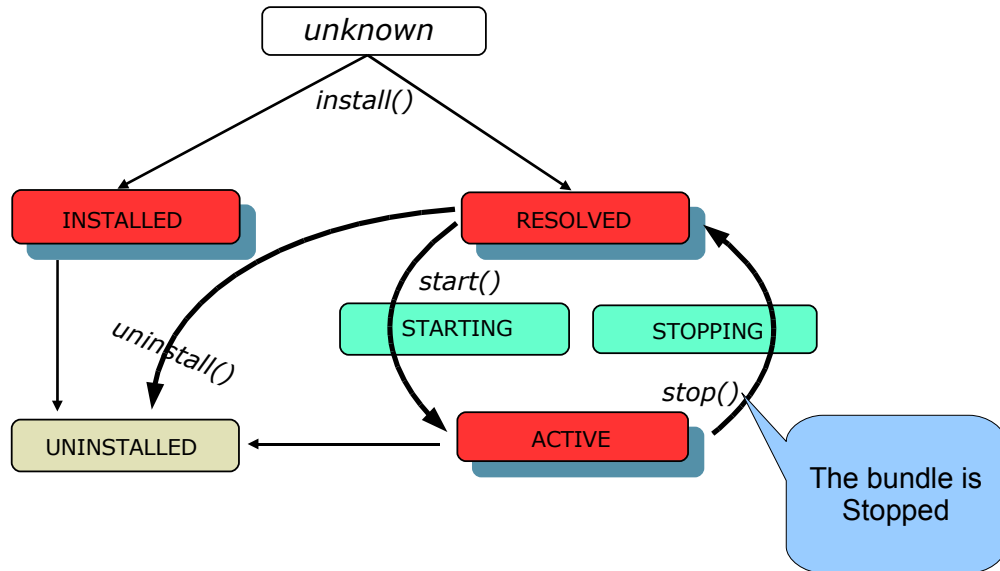
Bundle structure



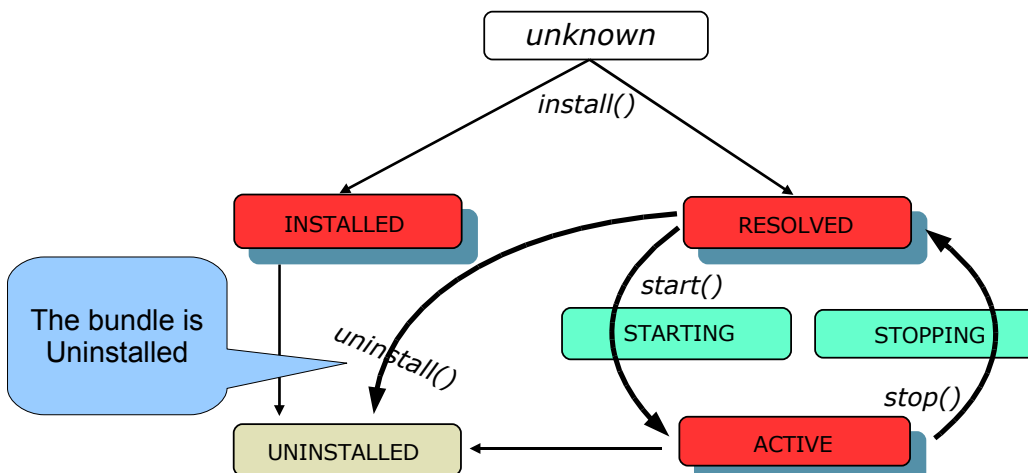
Bundle characteristics 2: Life-cycle perspective



Bundle characteristics 2: Life-cycle perspective



Bundle characteristics 2: Life-cycle perspective



Bundle characteristics 2: Life-cycle perspective



```

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

public class HelloWorld implements BundleActivator {
    public void start(BundleContext bc){
        System.out.println("Bonjour");
    }

    public void stop(BundleContext bc){
        System.out.println("Au revoir");
    }
}
    
```



Bundle-Name: Hello World
 Bundle-Description: The simple bundle
 Bundle-Activator: hello.HelloWorld

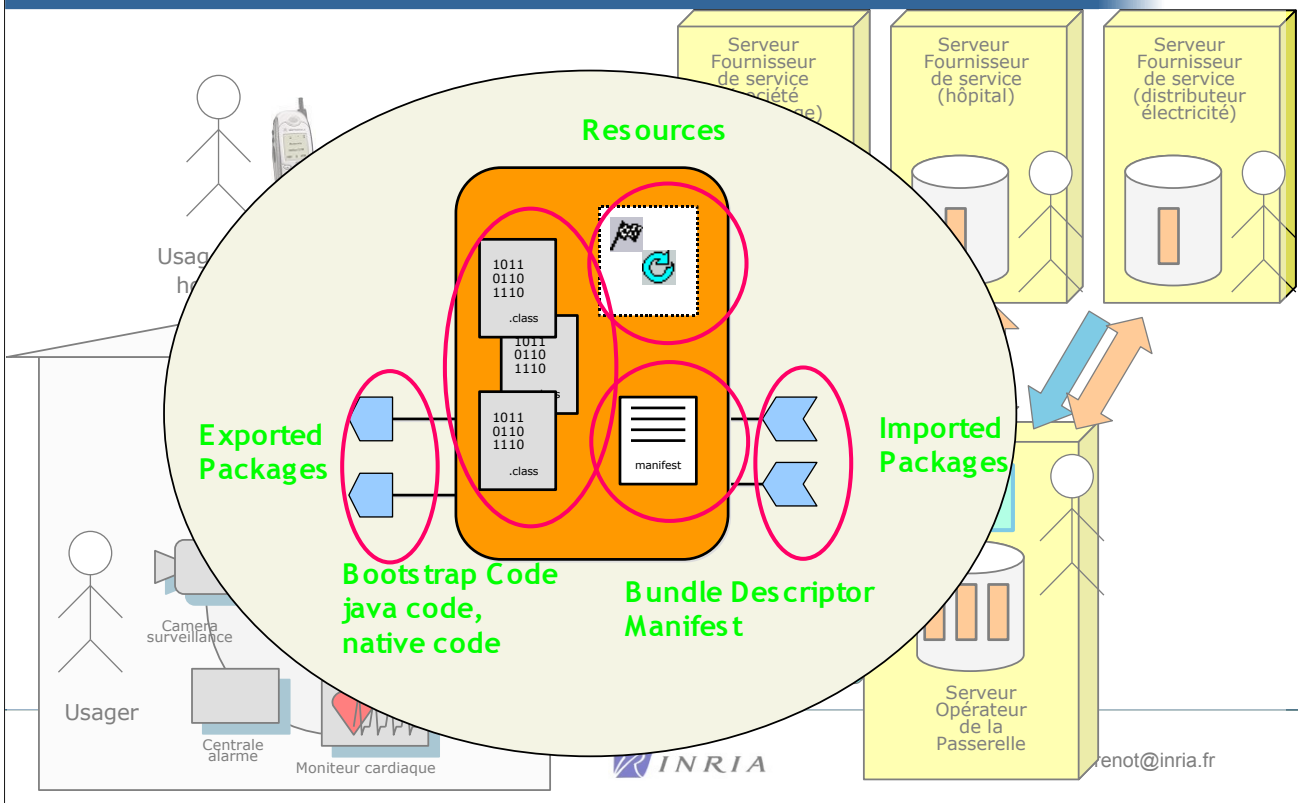
The bundle is "Launched"

The bundle is Stopped



200 Tue Jun 15 16:20:00 CEST 2004 META-INF/MANIFEST.MF
 723 Tue Jun 15 16:20:00 CEST 2004 hello/HelloWorld.class

Bundle structure



Why the Java/OSGi stack are promising ?

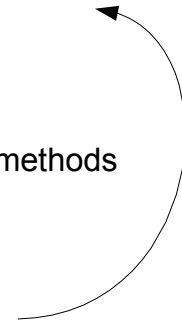


> Why Java ?

- It is object Oriented
- It has a huge standard API
- It is dynamic:
 - New objects can be used at run-time
 - New classes can be defined at run-time
- It is secured
 - Types are known at compile time and at run-time
 - A security Manager can control access to sensible methods

> Why OSGi ?

- It is component Based
- It has service-oriented programming features
 - => It is simplifying Java dynamicity integration
- It defines standard services



Java / OSGi Security



> Two aspects for java:

- A java archive can contain signature files. Unvalid signature makes the archive to be rejected
- A class should not execute specific methods
 - SBAC : Stack Based Access Control

```
m1.test();  
System.halt(); --> A security Manager checks a policy file to know  
whether the call is authorized or not
```

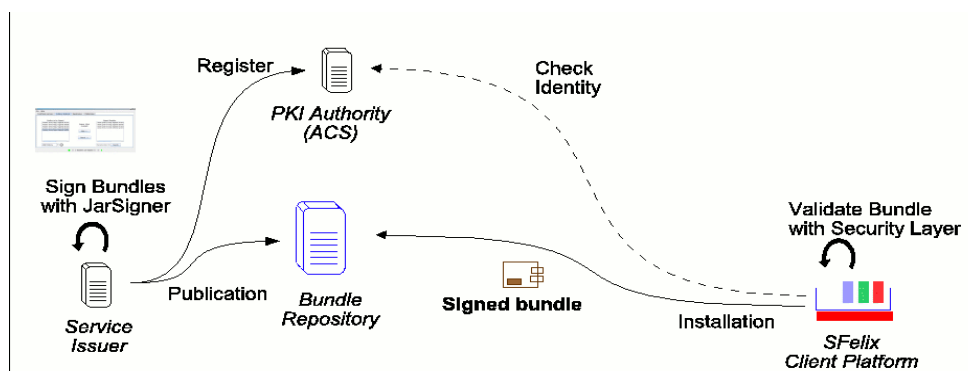
> OSGi extends both security aspects

- Some management methods can be enforced by the security Manager
- Bundles can be constrained by signatures files.

Java Security: Process



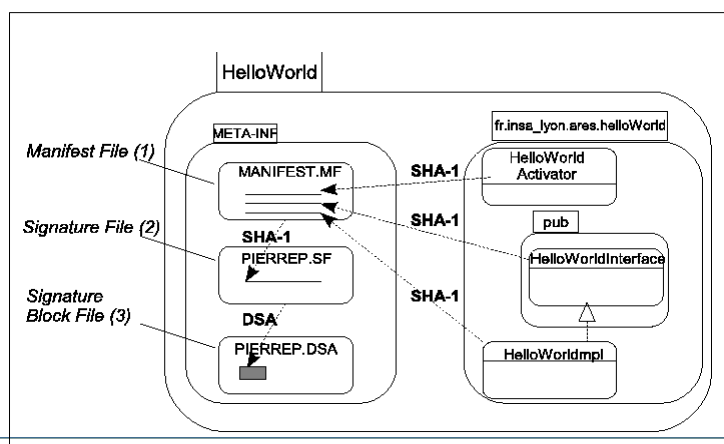
- > Signer Pierre generates a public/private key pair
- > Pierre registers its public key on a Certification Authority
 - And sends its certificate to the client Platform
- > Bundle resources are signed with the private key
- > Verification is performed with the public key at the client platform



Java Security: Archive Signature



- > Embedded in the archive
- > Digest of each file is stored in the Manifest
- > Intermediate Signature File to support multiple signer
- > Digital Signature in Signature Block File (CMS format)



> Java Archive

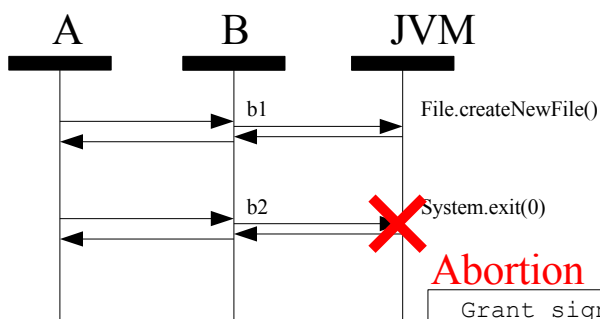
- Integrity of each class is checked when executed
- Abortion if error is detected
- Signed Items are not modified
- Resources can be added
- Resources can be removed
- No signer means a valid archive

> OSGi Bundle

- Same definition that Java Archive
- Stronger validity criteria
 - Resources can not be removed
 - Resources can not be added
 - Resources must be in a valid order
- Not specified
 - Bundles SHOULD be rejected when unsigned
 - Zero-Permissions for untrusted bundles

> Stack-based Access Control (SBAC)

- Class A is in the Protection Domain for signer Alice
 - Bundle for A is signed by Alice
- Class B is in the Protection Domain for signer Bob



```
Grant signer Alice{
    permission java.io.FilePermission "read",write";
}
Grant signer Bob{
    permission java.io.FilePermission "read",write";
    permission java.lang.RuntimePermission "exitVM";
}
```

> OSGi Permissions

- Dependency resolution (Package, Services)
- Fragments
- Bundle Management (install, start/stop)

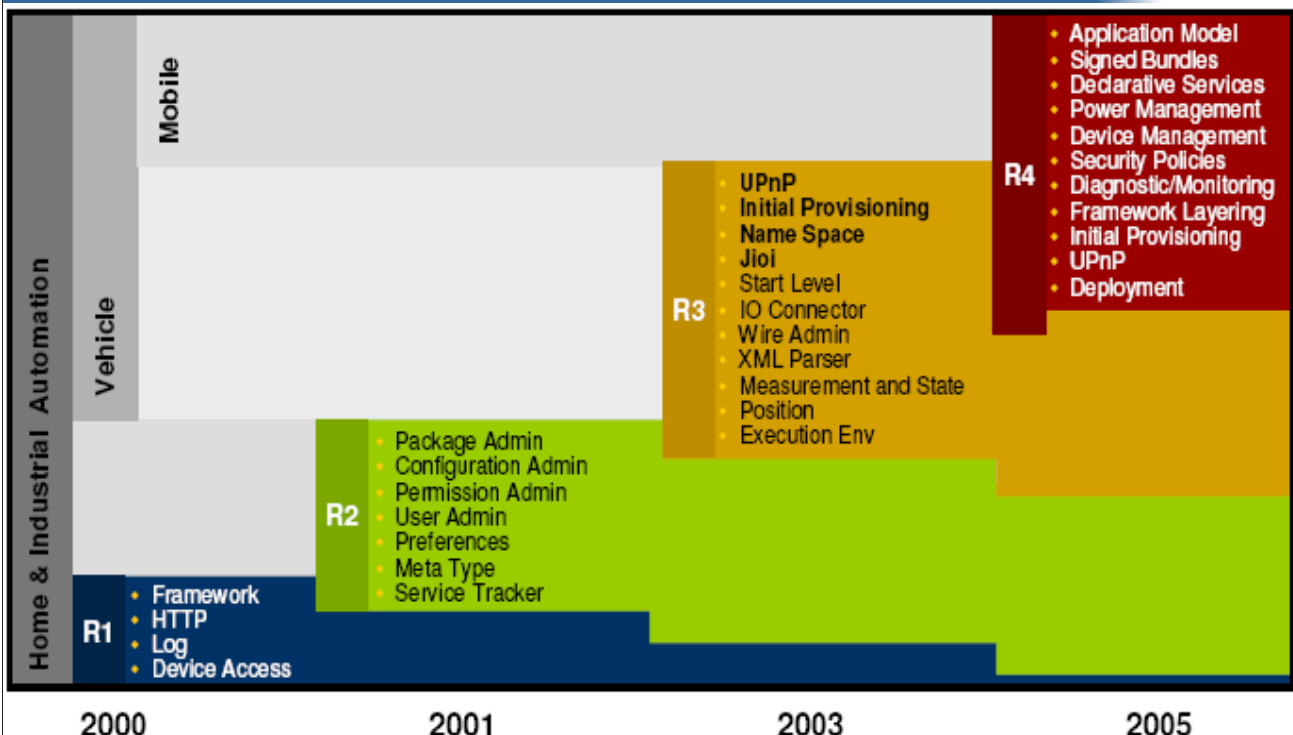
> Conditional Permissions

- OSGi R.4
- More fine-grained control

```

{
  [ .. BundleSignerCondition "*" ; o=ACME ]
  ( .. AdminPermission "(signer=\* ; o=ACME)" "*" )
  ( .. ServicePermission "..ManagedService" "register" )
  ( .. ServicePermission "..ManagedServiceFactory" "register" )
  ( .. PackagePermission "..cm" "import" )
}
    
```

Standard services RoadMap



Why the Java/OSGi stack are promising ?

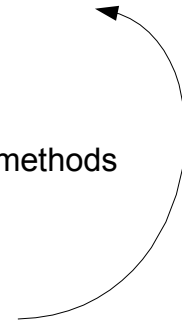


> Why Java ?

- It is object Oriented
- It has a huge standard API
- It is dynamic:
 - New objects can be used at run-time
 - New classes can be defined at run-time
- It is secured
 - Types are known at compile time and at run-time
 - A security Manager can control access to sensible methods

> Why OSGi ?

- It is component Based
- It has service-oriented programming features
 - => It is simplifying Java dynamicity integration
- It defines standard services



OSGi Main Concepts



■ Framework:

- Bundles execution environment
 - Oscar (Objectweb) / Felix (Apache), SMF, Knopperfish, Equinox
- Event notification



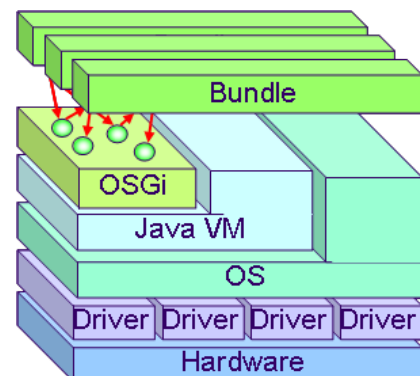
■ Bundles:

- Services diffusion and deployment unit

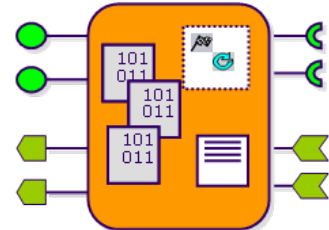


■ Services:

- Java Object implementing a well define contract



- > Modularize the middleware/application
 - Distribute the different middleware services
 - Better component visibility
 - Need of a deployment container
 - Partial update without restart all



- > Implementation
 - Based on Jarfile and Manifest entries
 - Explicit Package dependencies and Versioning (range)
- > Ready for probably next generation standard
 - JSR 277, Java Module System
 - Overtake JNLP(JSR-56), J2EE EAR, OSGi R3 bundle
 - J2SE 1.7 (2007)



Vertues

- Lightweight
- Clean Packaging
 - Avoid CLASSPATH Hell
- Dynamicity at run-time
- Non-stop VM



Vices

- Programatic approach
 - Event programming is painfull
- No non-fonctionnal services
- Centralized approach

- | | |
|--------------------------------|----------------------------|
| > Java language | > C language |
| > Automatic memory management | > Manual memory management |
| > Component convergence | > Packaging heterogeneity |
| > Service oriented programming | > ipc/socket programming |
| > Mono-user | > Multi-user |
| > Moderately efficient | > Very efficient |
| > Heavy weight | > Light weight |

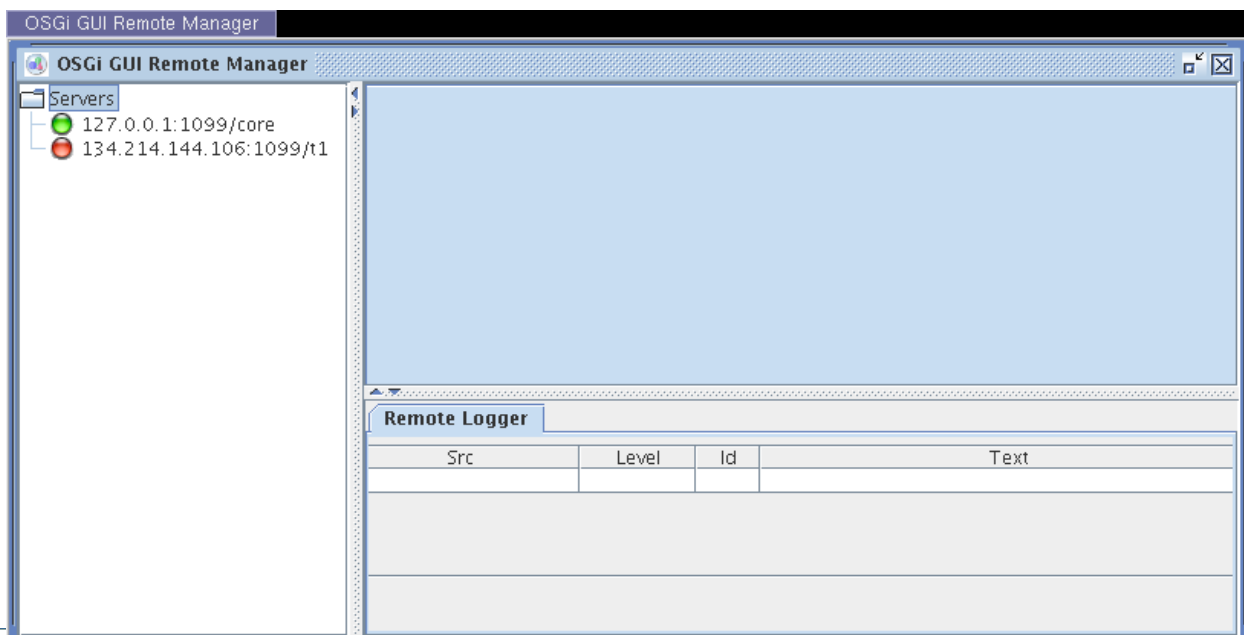


- > Based on Java programming
 - New classes at runtime, thanks to dynamic class loading mechanism
- > Container oriented
 - Life-cycle management of components, access control, Inversion Of Control pattern (IoC)
- > Service Oriented Programming
 - Complex services are made through composition
 - Standard services
 - != SOA

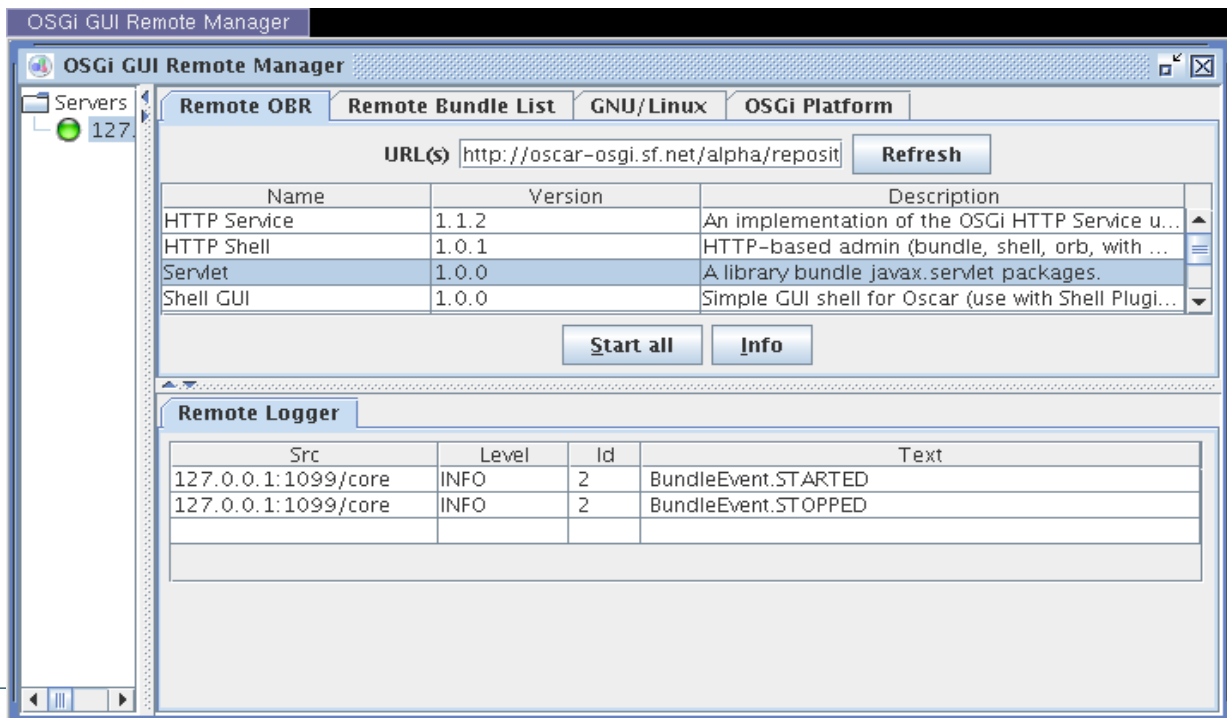
- > So what is missing ?

- > What is missing ?
 - A management layer for OSGi gateways : **MOSGi**
 - A support for Multi providers environment : **VOSGi**
- > Is it embeddable ?
 - Java jdk1.3 -> 9Mo != NLSU2, Linksys, 8Mo
 - Speed, Bandwidth
 - **ROCS**
- > Is it secured ?
 - Bundle-Signatures
 - RBAC --> SBAC --> **CBAC**
 - **SFELIX**

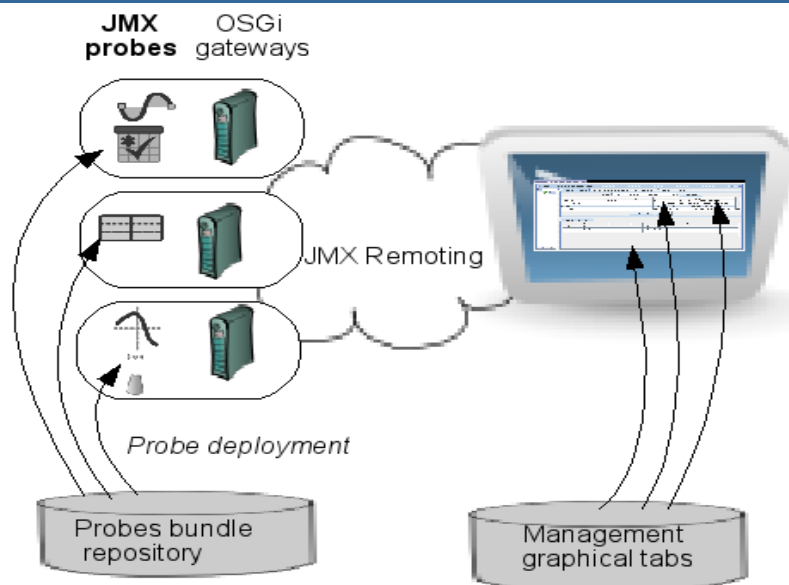
- > An end-to-end management framework for OSGi Gateways
- > Based on JMX management architecture
 - Java Standard
 - Easy to implement
 - Open-source initial distributions
- > End-to-End
 - Probes: low level data from the system, linux, OSGi, installation
 - Agent and services: notifications, rmi and xml connector
 - Dedicated Management Console: On-Demand Management Panels
- > Embeddable
 - SOP approach to the implementation
 - Lightweight embedded JMX agent



Once connected new user interface



General overview of MOSGi



Integrated in Felix Apache project, 2005

What OSGi Lacks

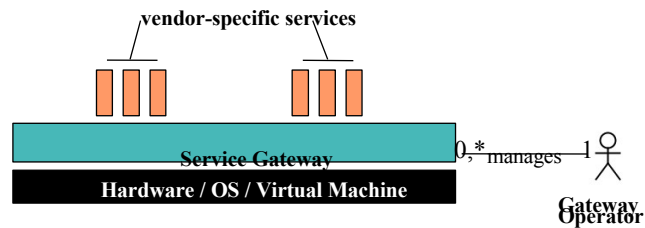


> Management unspecified

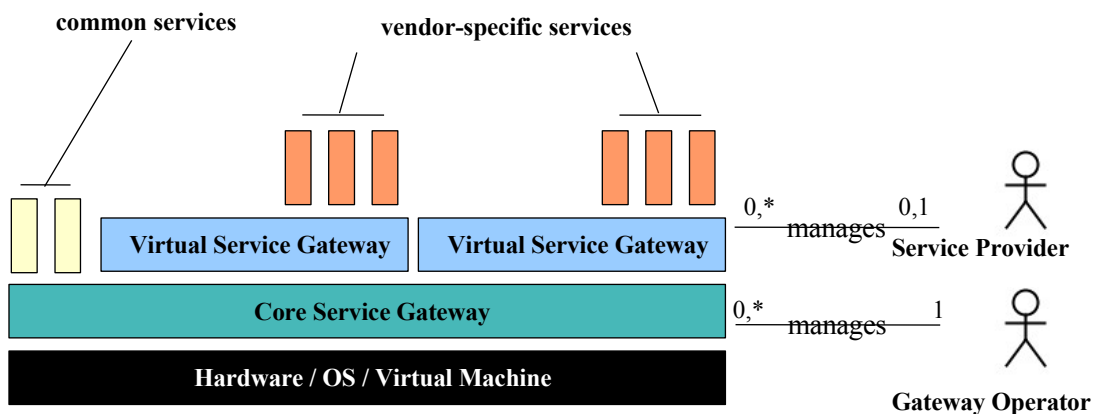
- MOSGi ✓

> No multi-user

- VOSGi

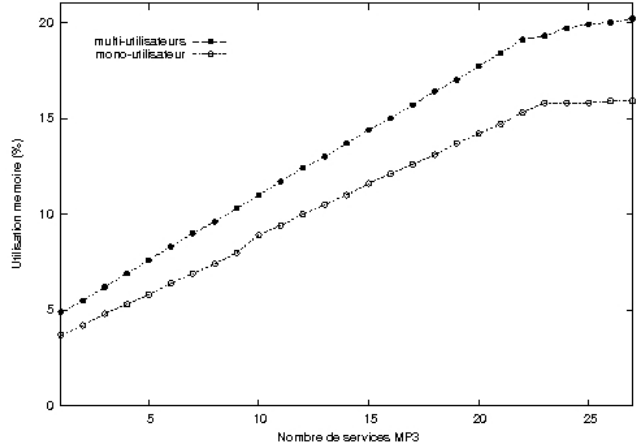


VOSGi: Virtual OSGi

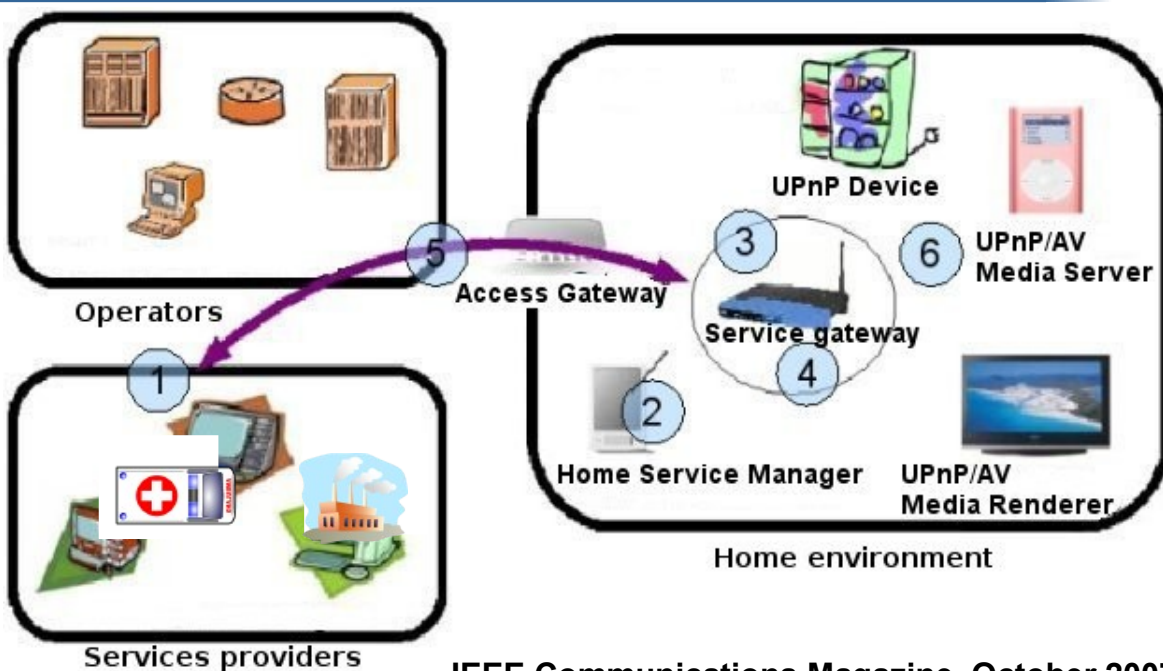


- ✗ Per-provider isolation of services
 - Explicit service sharing on demand
- ✗ Isolation of management agents

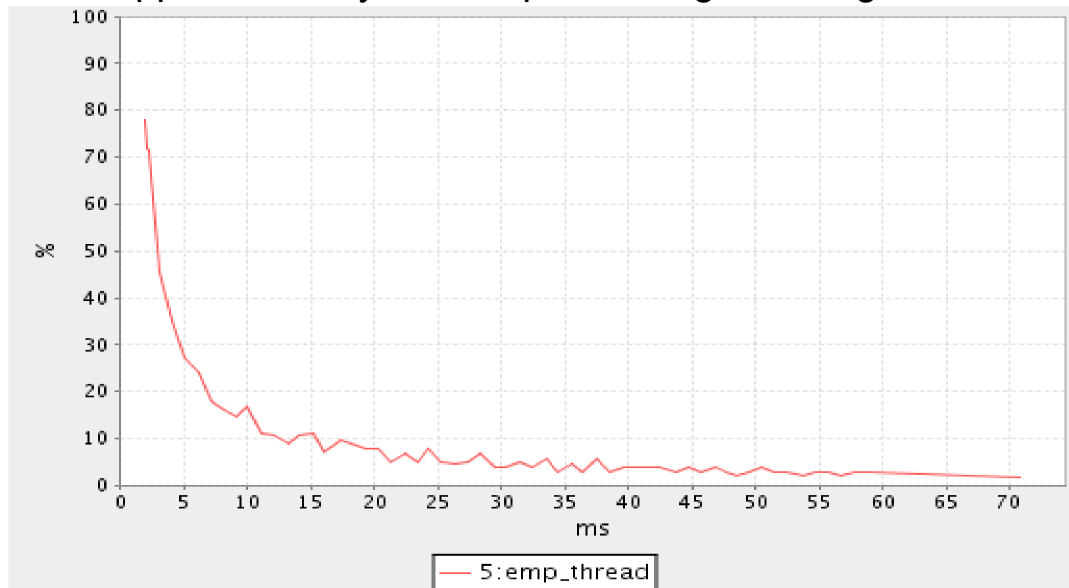
- > Runs OSGi within OSGi
- > Patch to Apache Felix and Concierge
- > Namespace isolation
- > Service sharing
- > Works right now



• **Published in CBSE'06, Stockholm**

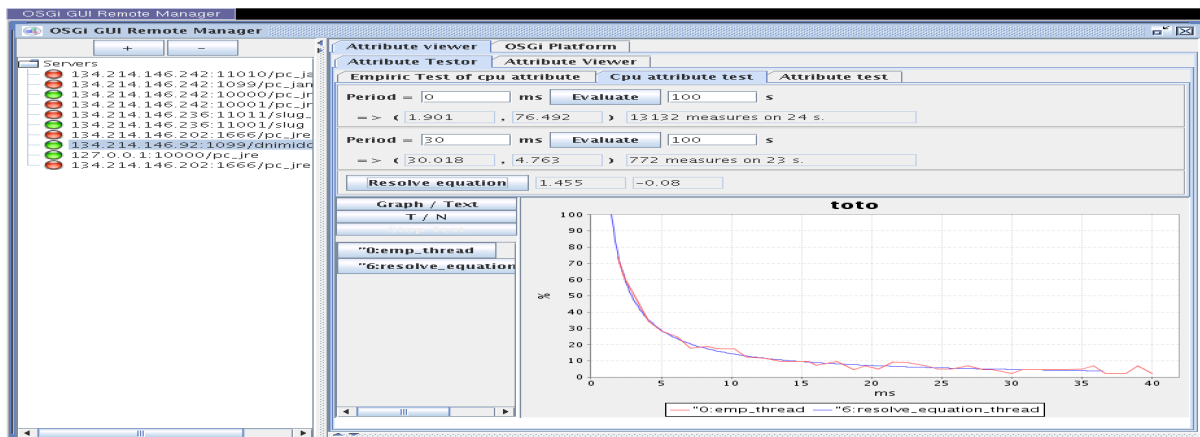


> What happens if many service providers get management data ?



> A framework to schedule management queries

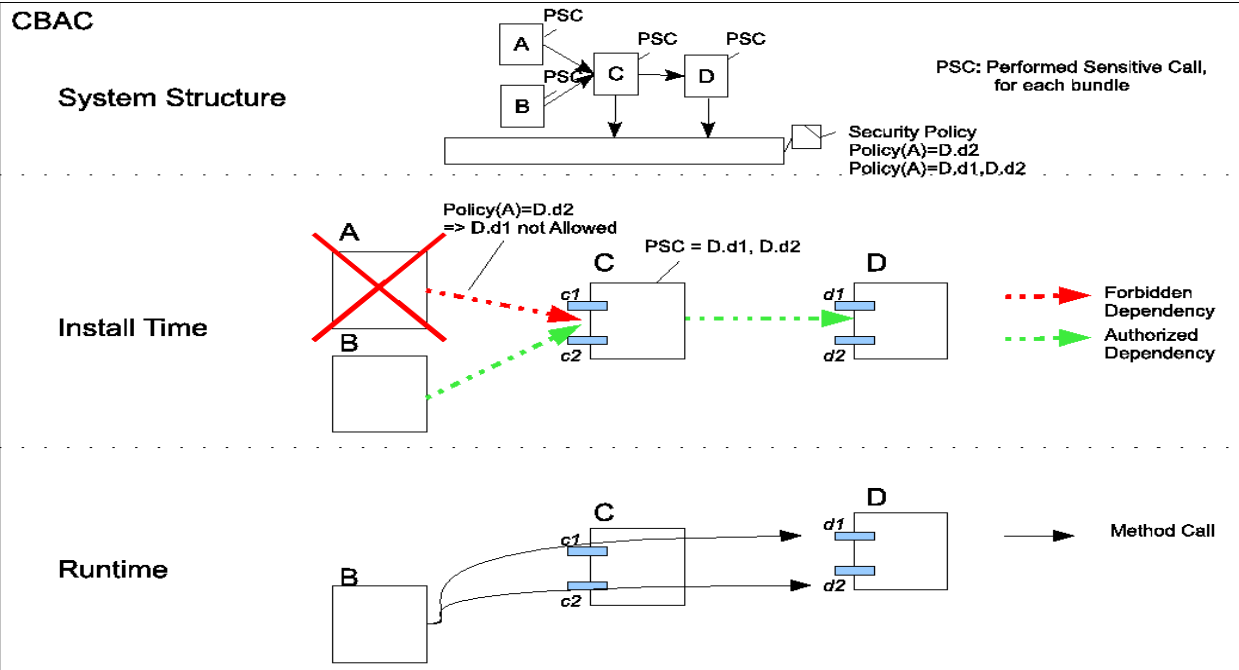
- Each probe is evaluated through mosgi
- The manager has to find a balance between rate of queries and implied load. The tool automates this.



Monitoring Scheduling for Home Gateways,
Stéphane Frénot Yvan Royon Pierre Parrend Denis Beras, NOMS 2008

- > Recommendations for implementing Secure OSGi Platform
 - Never install a bundle when signer is not trusted
 - Maximum size of installed archive to be set
 - Maximum number of published services
 - Launch the activator of bundles in a separate thread
 - Always remove data associated with uninstalled bundles

- > Component-Based Access Control
- > Static Analysis at install time
- > Benefits
 - No performance loss at runtime
 - No untrusted applications are installed
 - No runtime abortion



Work In-Progress

- > Embedded virtual machines **ROCS**
- > **HGL** A native Linux implementation

- > TBC.... Questions ?