redhat.

Java. Cloud. Leadership.

Benefits of OSGi in Practice

David Bosschaert
Principal Engineer
JBoss/Red Hat
OSGi EEG co-chair

October 2012

About me

- Background in AI / Computer Science,

  University of Amsterdam (1996)
- Developing software since mid-1980s,

  working with Java since 1997
- Joined IONA Technologies, Ireland, 1999
- Involvement with OSGi specs from 2007
- EEG co-chair 2009
- At JBoss/Red Hat since 2010
- Involved in Apache, JBoss, Eclipse

  and some github-based opensource projects

Agenda

- Introduction to OSGi
- OSGi Modularity
- OSGi Services
- Specifications
- Demo

OSGi
a brief introduction

OSGi – a brief intro

- Dynamic Module and Services platform for Java
- Started 1999
- Specifications created in the OSGi Alliance
  - a non-profit Standards Development Organization
  - members include: IBM, Oracle, Red Hat, Adobe, Siemens, TIBCO, France Telecom, Deutsche Telekom, Technicolor, NTT, Hitachi & many more
- www.osgi.org

OSGi – areas

Core: the OSGi Framework

- Modularity
- Services
- Lifecycle and Dynamicity
- Security

Enterprise: services & component on top of Core Framework

- Addressing Enterprise use-cases, such as:
  - Service Distribution
  - Component models and IoC
  - Configuration and Management
  - JavaEE integration
  - Cloud Computing

Residential and embedded services/components

OSGi Framework implementations

- The most popular OSGi implementations are open source:
  - Eclipse Equinox
  - Apache Felix
  - JBoss AS 7
  - Knopflerfish

- Commercial implementations also exist

OSGi Enterprise implementations

Also available in open source, in projects such as:

Apache Aries	Eclipse Gemini
Apache CXF	Ops4J Pax
Apache Felix	Oracle GlassFish
Eclipse ECF	JBoss
Eclipse Equinox	KnowHowLab.org

- Commercial implementations exist
- For the full list go to:
  http://en.wikipedia.org/wiki/OSGi_Specification_Implementations

OSGi – Modularity

Modularity

“I struggle to maintain this application because I don't fully understand what it does”

Non-modular components often keep growing

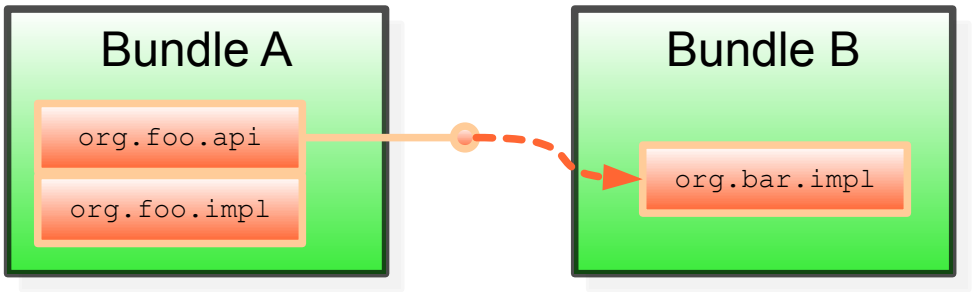
- blurring their responsibility
- increasing complexity
- making maintenance harder over time
- scope creep
- in the end: a Big Ball of Mud[1]

OSGi:

- instead of an ever growing component:
  - many separate modules
  - each with a clear function (and clear API)

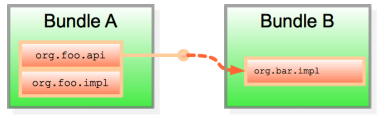
[1] http://en.wikipedia.org/wiki/Big_ball_of_mud

OSGi Modularity

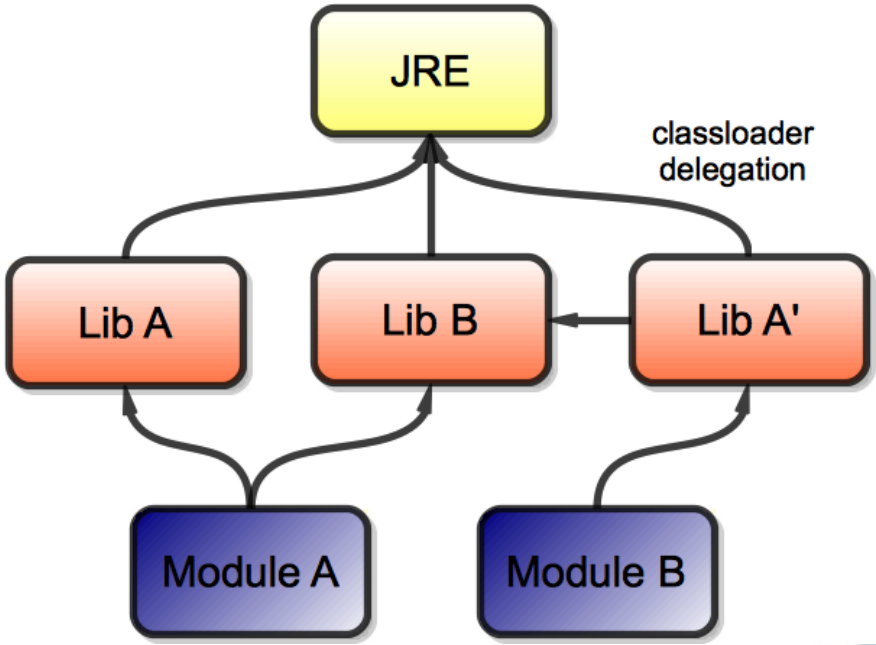


- In OSGi Modules are called **Bundles**
- With OSGi Modularity you *must* declare what a module provides and what it needs.
  - Typically Java packages
  - Can be other capabilities
- Everything not explicitly declared as provided is internal → not accessible to other modules

OSGi Modularity (2)



- Improves maintainability
  - as module boundaries and function must be made clear
  - tends to improve cohesion
    - and reduce coupling
- Non-monolithic
  - allow for fine-grained updates of the system
- Concurrent versions
  - multiple versions of the same module can co-exist
  - allows for gradual upgrades



OSGi Modularity in Practice

- From a Java point of view. Just use classes as normal:

```
package org.acme.package1;
import org.acme.package2.MyClass;

...

MyClass ax = new MyClass();
ax.foo()
```

- OSGi Module Metadata in MANIFEST.MF

```
Export-Package: org.acme.package1; version=1.2

Import-Package: org.acme.package2; version=[1.1, 2)
```

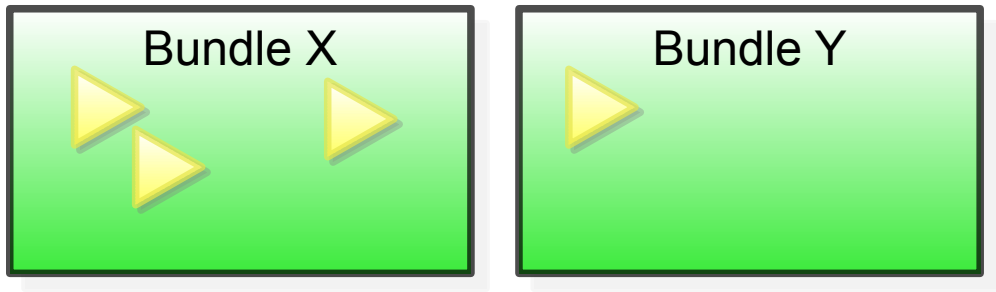
Compiler support by some, e.g Eclipse javac

Runtime enforcement by OSGi Framework

OSGi – Services

Brief intro to **OSGi Services**

- Services are Java Objects (POJOs)
  - registered by Bundles
  - consumed by Bundles
- “SOA inside the JVM”



- Services looked up by type and/or custom filter
  - “I want a service that implements org.acme.Payment where location=US”
  - One or many
- Dynamic! Services can be updated without taking down the consumers
  - OSGi Service Consumers react to dynamism

OSGi Services

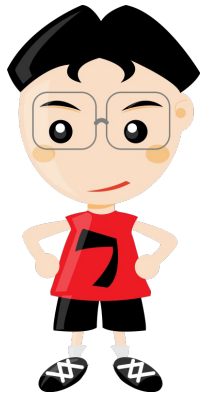
“We have many different components that perform a similar task”

How can we increase software reuse?

- With OSGi Services there is no direct link between the service consumer and provider
  - not even a text or XML file somewhere
  - they communicate through a predefined API
  - improves re-usability
- In OSGi software reuse is visibly much higher than elsewhere
- Given clear APIs implementations can be swapped
  - even at runtime
- OSGi standardizes some Service APIs
  - organizations sometimes do the same

OSGi Services

"We have many customizations that are labour intensive to support"

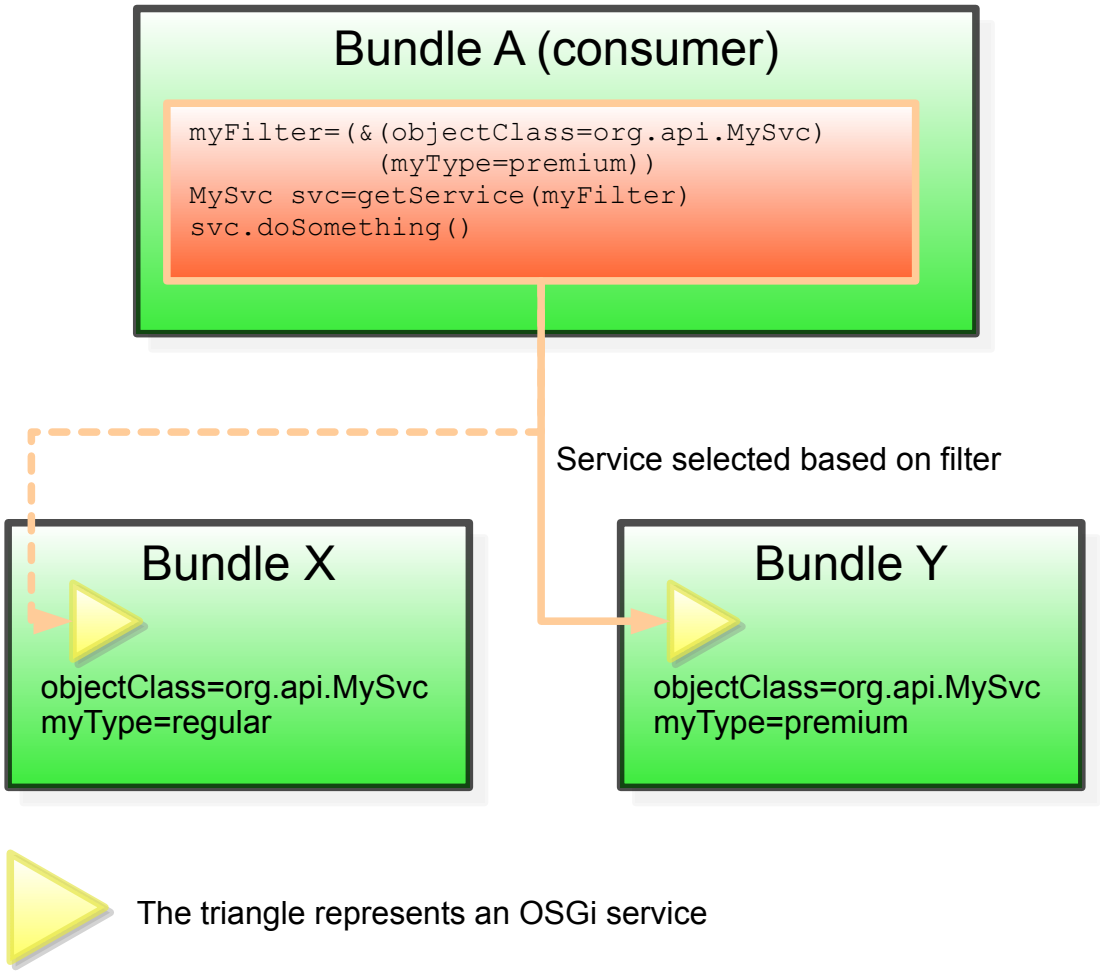


How can I tailor my services more effectively?

- Customizations are needed for
  - Premium customers vs community users
  - Customers who bought specific functionality
  - Government vs commercial customers

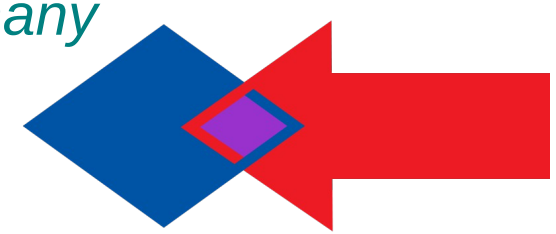
  ➔ Services can be used to swap in/out customizations
- To tailor a service, just provide an alternative for a given API
- Services can be selected based on API and Properties
- Properties can be used to find the right set of services for a given customer

Dynamic Service Selection



OSGi Services and Consolidation

"Business consolidation brings many integration challenges"



I need to replace a Service with an alternative technology

- If the API of the new technology can be mapped to the existing service
  - Create an OSGi Service that wraps the alternative
  - Replace the bundle providing the existing service with another bundle providing the wrapped technology
- If no mapping can take place
  - OSGi can still help through multiple concurrent versions
  - Gradual migration

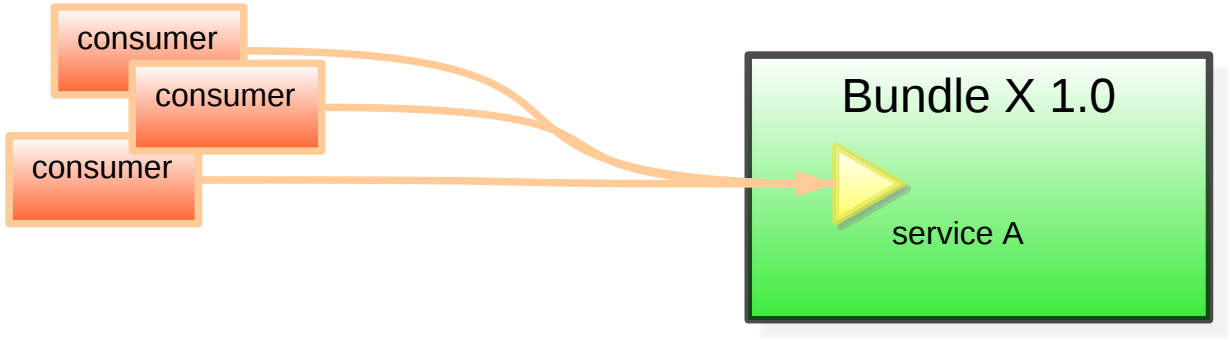
OSGi Services and Bugfixes

"We need to minimize downtime needed to apply bugfixes"

- OSGi Service-based development can help a lot here
- Let's say Service A1 contains a bug
- Service A1 has many active clients
- We don't want to kill the clients
- But new clients should use the fixed service
- Service A2 is a fixed version of the service
  - Implementation has changed, API hasn't
  - Install bundle with Service A2
  - Service A1 and A2 exist concurrently
  - Uninstall bundle with Service A1
    - Old clients can still finish using the service
    - Will not be handed out to new clients

OSGi Services and Bugfixes

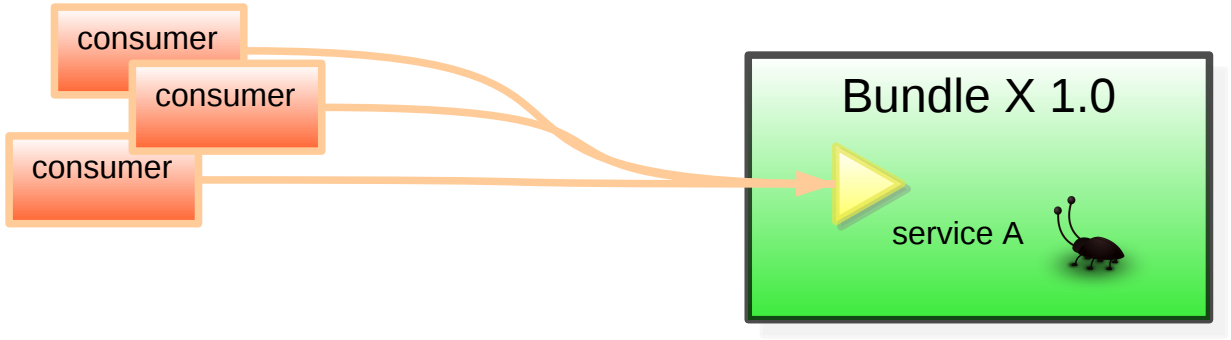
“We need to minimize downtime needed to apply bugfixes”



Apply service bugfixes without downtime

OSGi Services and Bugfixes

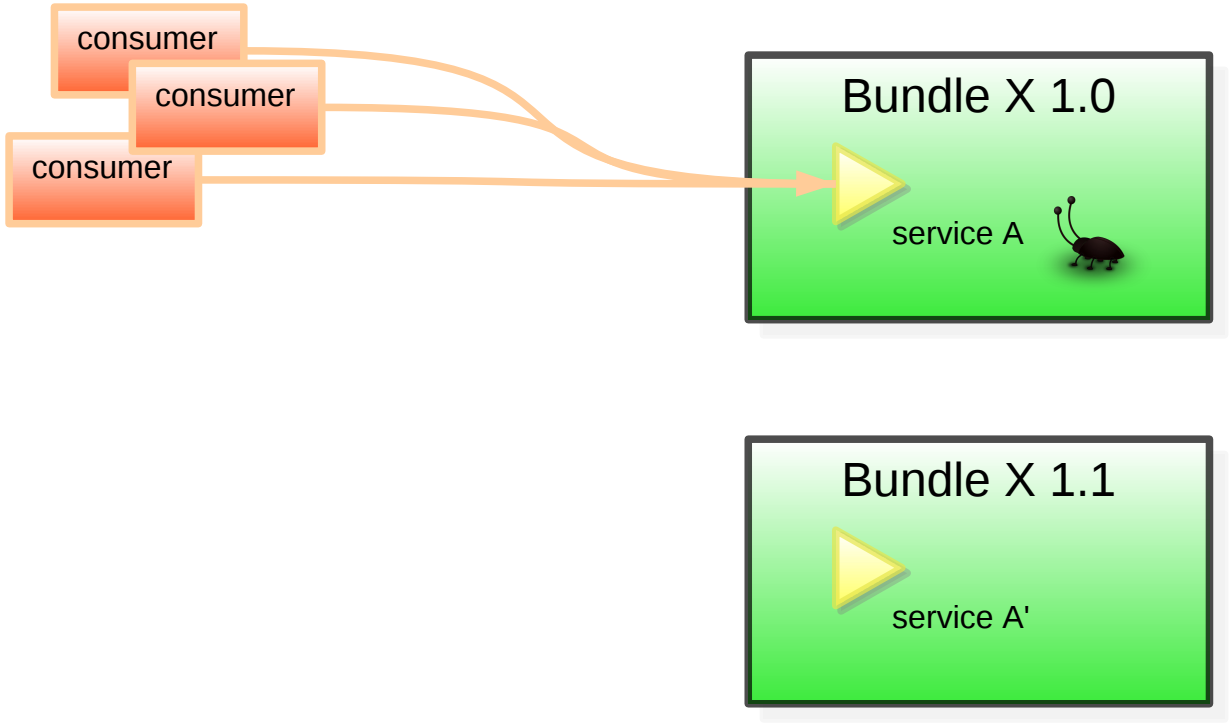
“We need to minimize downtime needed to apply bugfixes”



Apply service bugfixes without downtime

OSGi Services and Bugfixes

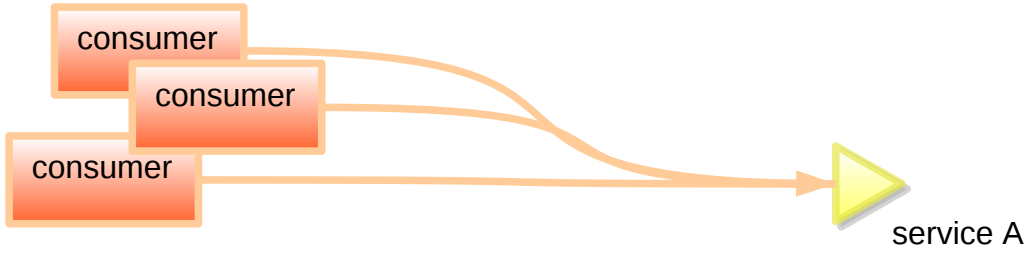
"We need to minimize downtime needed to apply bugfixes"

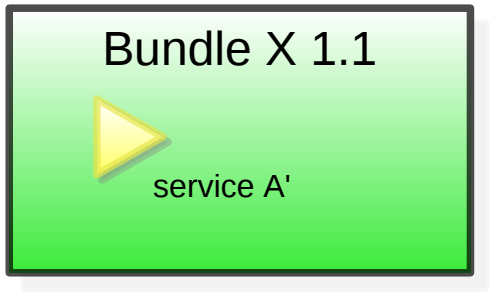


Apply service bugfixes without downtime

OSGi Services and Bugfixes

“We need to minimize downtime needed to apply bugfixes”

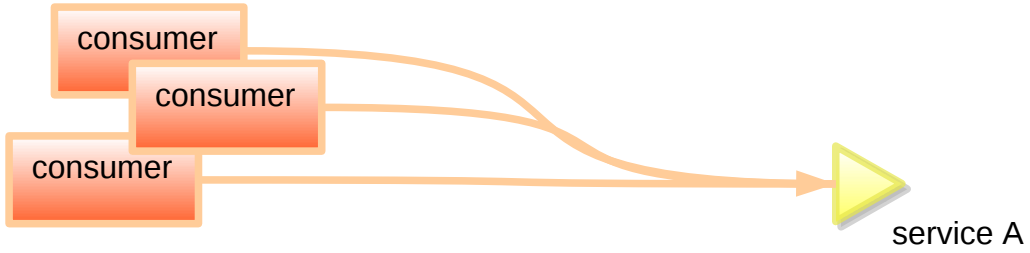


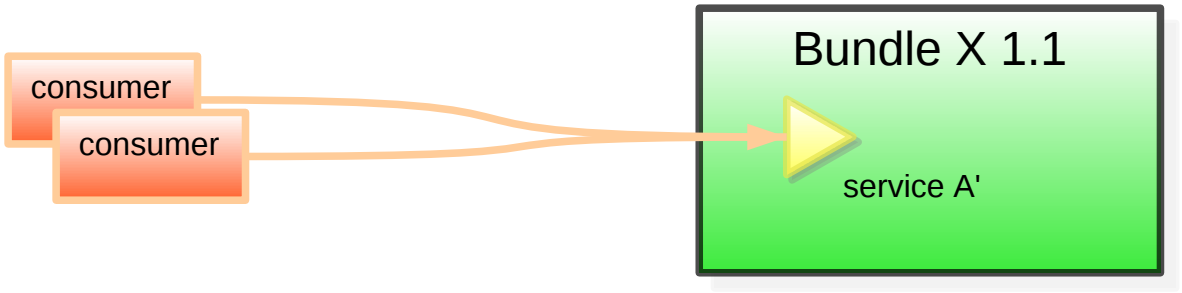


Apply service bugfixes without downtime

OSGi Services and Bugfixes

"We need to minimize downtime needed to apply bugfixes"

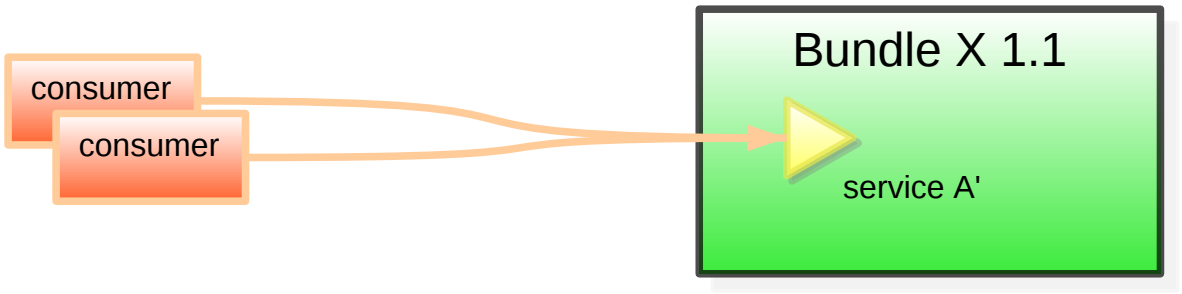




Apply service bugfixes without downtime

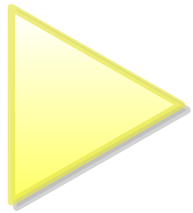
OSGi Services and Bugfixes

“We need to minimize downtime needed to apply bugfixes”



Apply service bugfixes without downtime

OSGi Services Summary



- SOA within the Java VM
- Multiple services can provide the same API
- Dynamic (can come and go at runtime)
  - and OSGi Service Consumers know how to deal with this
- Lookup based on LDAP-style filters
  - look up one or many
- OSGi Services are normally within the Java VM
- also: Distributed OSGi Services
  - OSGi Remote Services specifications

OSGi migration challenges

Challenges – modularity

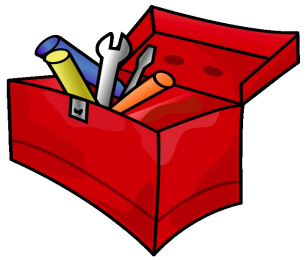
Migrating existing applications to modular environment is challenging

- Applies to any modular system, not just OSGi
- Requires some planning
- I would suggest a gradual approach
  - start with a large bundle containing much of the existing functionality
  - splitting off smaller modules over time

Prevalent modularity anti-patterns

  - Class.forName()
  - java.util.ServiceLoader
  - Thread context classloader
- Solutions exist for each, but requires some attention

Challenges – tooling



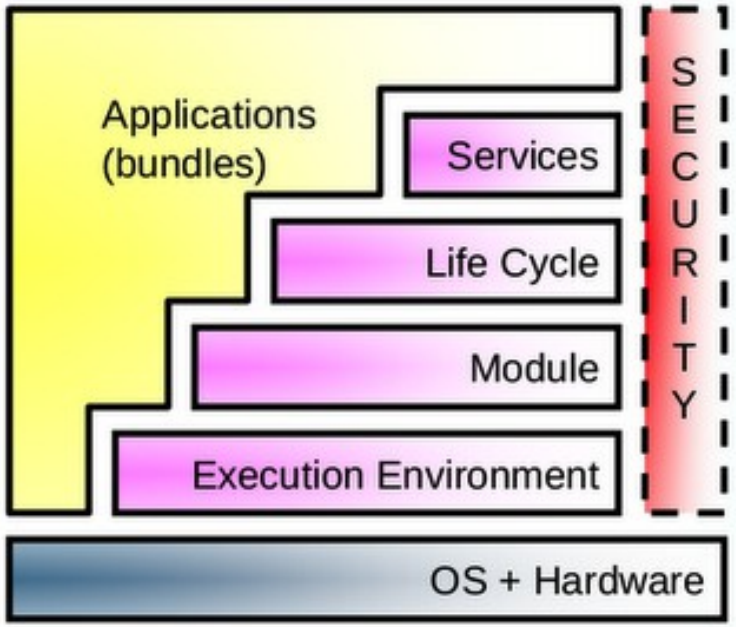
- Using OSGi works best when using the right tools
- Generally a combination of tools should be selected, for
  - Development (IDE)
  - Command-line build
  - Testing
  - Deployment
  - etc...

- New documentation on tool*chains:*
  - http://wiki.osgi.org/wiki/ToolChains
  - Great new book: Enterprise OSGi in Action, Cummins et al

OSGi – Specifications

OSGi Core Specification (R5)

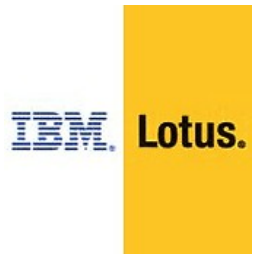
- Modularity layer and Bundle specification
- Services layer
  - Service Registry
- Life Cycle layer
  - Everything dynamic!
    add, remove, start, stop,
    update bundles
- Security layer
- Generic capabilities and requirements model
- … and some low-level
  building blocks …

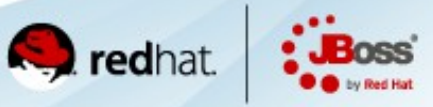


OSGi Enterprise Specification (R5)

- Addresses Enterprise use-cases
- Standard Service APIs
  - Configuration
    - Config Admin / MetaType specs
  - Component models (IoC)
    - Blueprint and Declarative Services
  - Eventing
    - Event Admin spec
  - Service Distribution
    - Remote Services and Remote Service Admin
  - Repository specification
  - Subsystems (multi-bundle deployment entities)
  - JavaEE integration (JTA, JNDI, JPA, JDBC, ...)
  - … and more …

OSGi Applied





Java. Cloud. Leadership.

Demo and Questions

- Online version:

  http://www.youtube.com/watch?v=S8AI3TqiVk4

References

Specs: http://www.osgi.org/Specifications

Semantic Versioning White Paper: http://www.osgi.org/wiki/uploads/Links/SemanticVersioning.pdf

Books:

Enterprise OSGi in Action (Holly Cummins / Tim Ward)
ISBN: 9781617290138 (Nov 2012)
early access download:
http://www.manning.com/cummins/

OSGi in Action (Richard Hall et al)
ISBN: 9781933988917

OSGi in Depth (Alex Alves)
ISBN: 9781935182177

Java Application Architecture: Modularity Patterns with Examples using OSGi (Kirk Knoernschild)
ISBN: 9780321247131

Image credits

- Images from openclipart.org

  Cockroach – TrnsltLife
  Merge Arrow – spongman
  Buildings – jean_victor_balin
  Mr Happy – shokunin
  Boy, Girl, Man – dcatcherex
  Jigsaw puzzle – Benjamin Pavie
  Toolkit – bogdanco

Click to add title

- Click to add an outline