OSGi with Spring DM

Dmitry Sklyut
Chariot Solutions

Agenda

- OSGi quickly
  - What is OSGi?
  - Why use OSGi?
- Enter Spring DM
  - Objectives of Spring DM
  - Spring DM fundamentals
  - Test support
  - Web Support
- Q&A

A little history of OSGi

- Sponsored by OSGi Alliance
  - http://www.osgi.org
- Started in 1999 with focus on embedded Java and networked devices
  - JSR 8
- 2003 – support for mobile devices
- 2004 – major open source adoption
- now – moving into server-side...

What is OSGi

- OSGi™
  - Open Service Gateway Initiative
- Currently known as
  - The Dynamic Module System for Java™

OSGi is Universal Middleware.

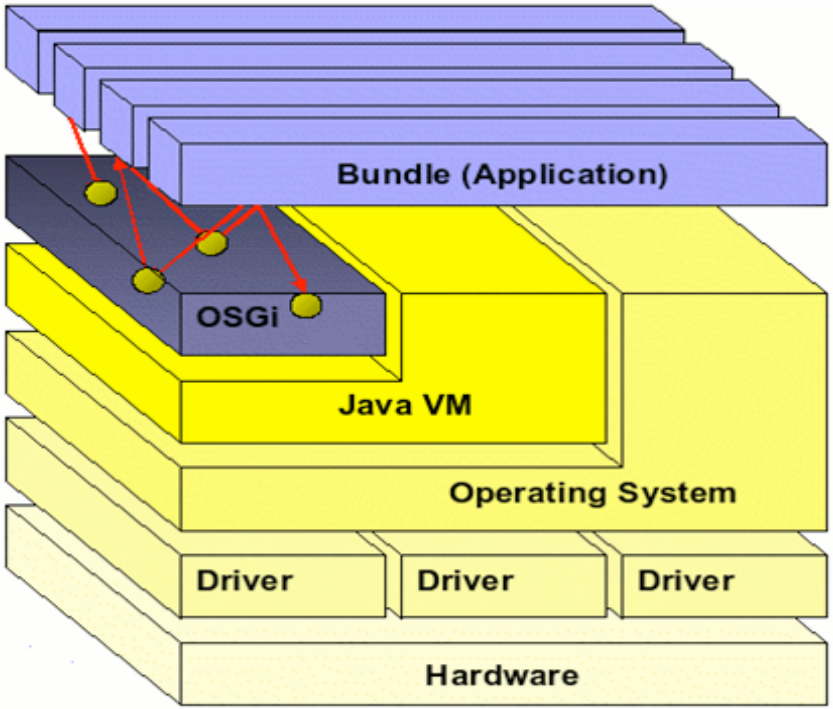
Software that you write once and can use in binary form universally: in many different platforms, many different industries, and for many different purposes.

Peter Kriens (OSGi evangelist)

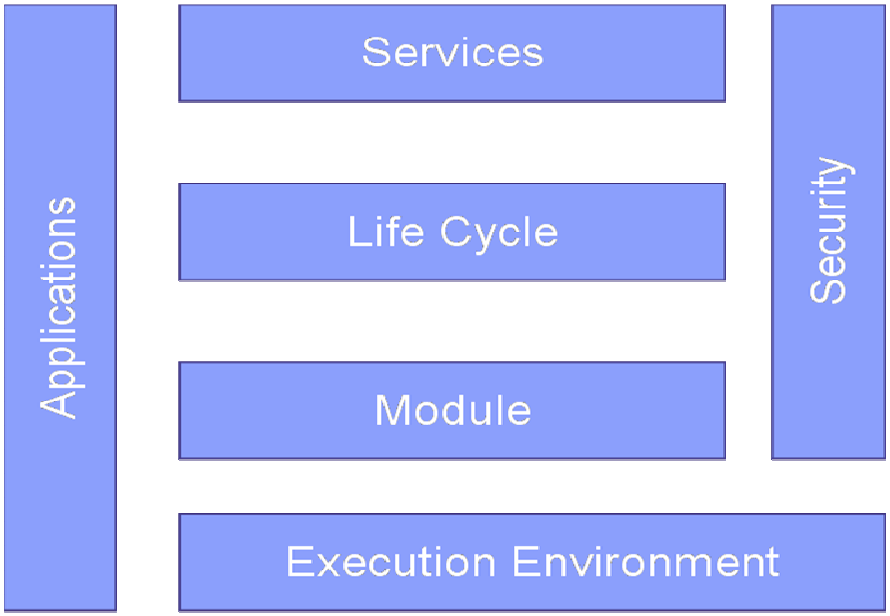
OSGi is a ***platform*** for building modular component based applications

OSGi Framework

- Thin layer on top of JVM
- Allows applications to run in shared environment
- Provides Classloading
- Life-Cycle Management
- Communication
- Collaboration



OSGi layers


Applications
Services
Life Cycle
Module
Security
Execution Environment

Bundle?

- Bundle is nothing more than a jar with custom manifest headers

```
Bundle-ManifestVersion: 2
Bundle-Name: Service Client Bundle
Bundle-SymbolicName: service.client
Bundle-Version: 1.0.0
Bundle-Activator: demo.client.internal.Activator
Export-Package: demo.client
Import-Package: demo.service;version="[1.0.0,3.0.0)",
 org.osgi.framework;version="1.0.0",
 org.osgi.util.tracker
```

- ***Export-Package*** header
  - is used to export packages for other bundles to use
  - If package is not exported it is not visible outside of the bundle
- ***Import-Package*** header
  - is used to set-up bundles classpath from external contributions
- Can carry imbedded resources
  - Other jars, html help, source, etc

Why use OSGi?

- Modularization
- Management
  - including remote
- Versioning

It’s Modular

- Breaks up system into a number of modules
  - a.k.a. “bundle” - as a unit of deployment and modularization
- Strict visibility rules
- Provisioning process
  - dependencies resolved before bundle is started
- Versioning support!

And Dynamic

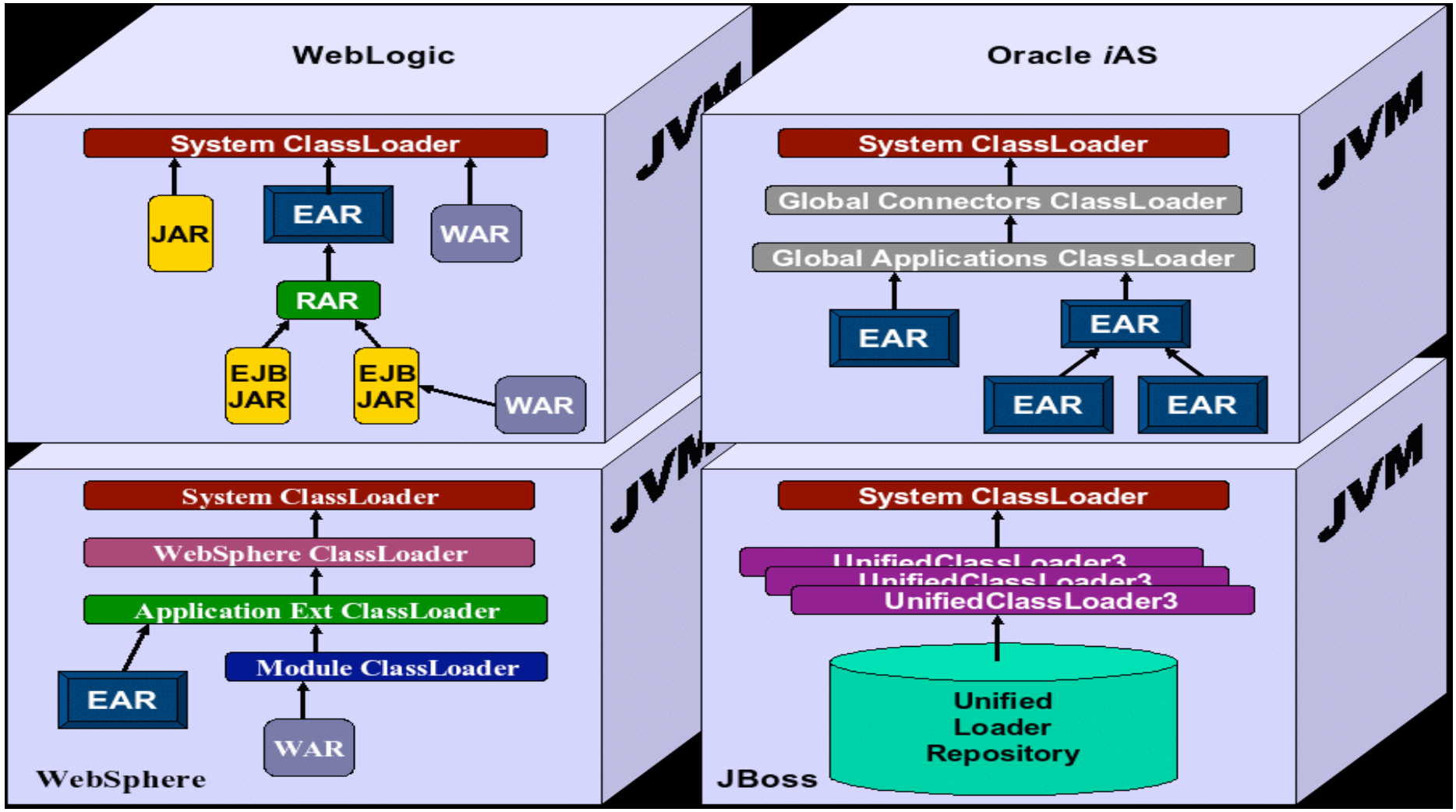
- Modules have a defined lifecycle
- Modules can be:
  - installed
  - updated
  - started
  - stopped
  - uninstalled
- All of that without a runtime restart!

It’s even Service Oriented!

- Bundles can publish services
- Service Registry allows other bundles to
  - consume published services
  - look up service with query language
- Service lifecycle is handled by the runtime

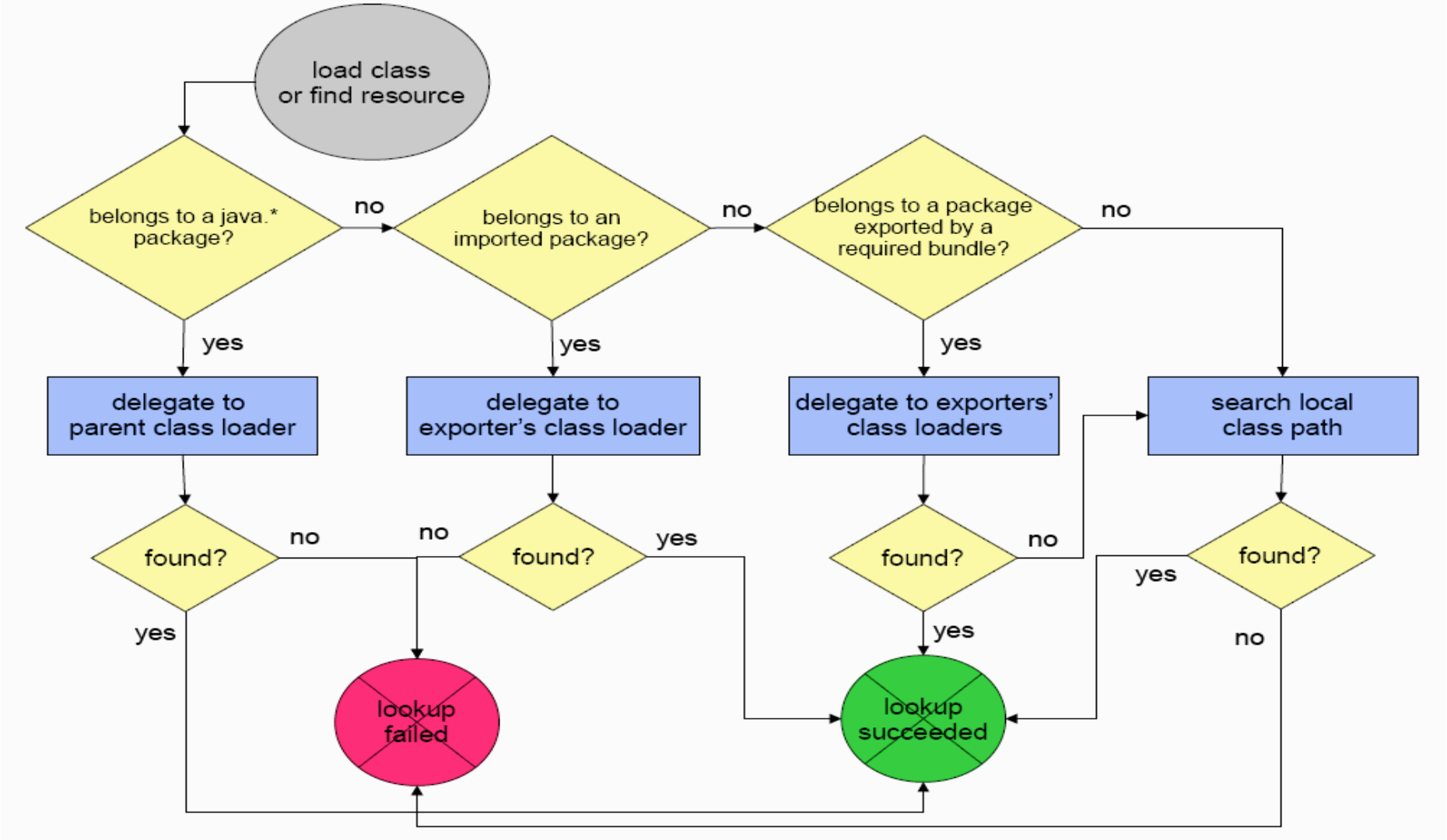
SOA in a Box

Classloading: old and busted



Classloading: new and shiny

Classloading: new and shiny



Classloading

- OSGi module layer guarantees that
  - there is just one FQN loaded in a classloader for a bundle.
  - no possibility for Class Cast Exception
- This maxim allows for versioning support

Problems with raw OSGi

- Context Class Loader is undefined
  - used by many enterprise libraries
- Highly asynchronous
  - must manage service dependencies manually
  - must manage threads/concurrency manually
- Testing
  - dependency on OSGi API

Enter Spring DM

Objectives

… make development of OSGi application simpler and more productive by building on the ease-of-use and power of Spring Framework

(Spring DM reference doc)

Objectives

- Bring the benefit of OSGi to enterprise applications
  - strong module foundation
  - versioning support
  - service oriented architecture
  - continuous operation

(From Spring DM web-cast by Costin Leau)

Fundamentals

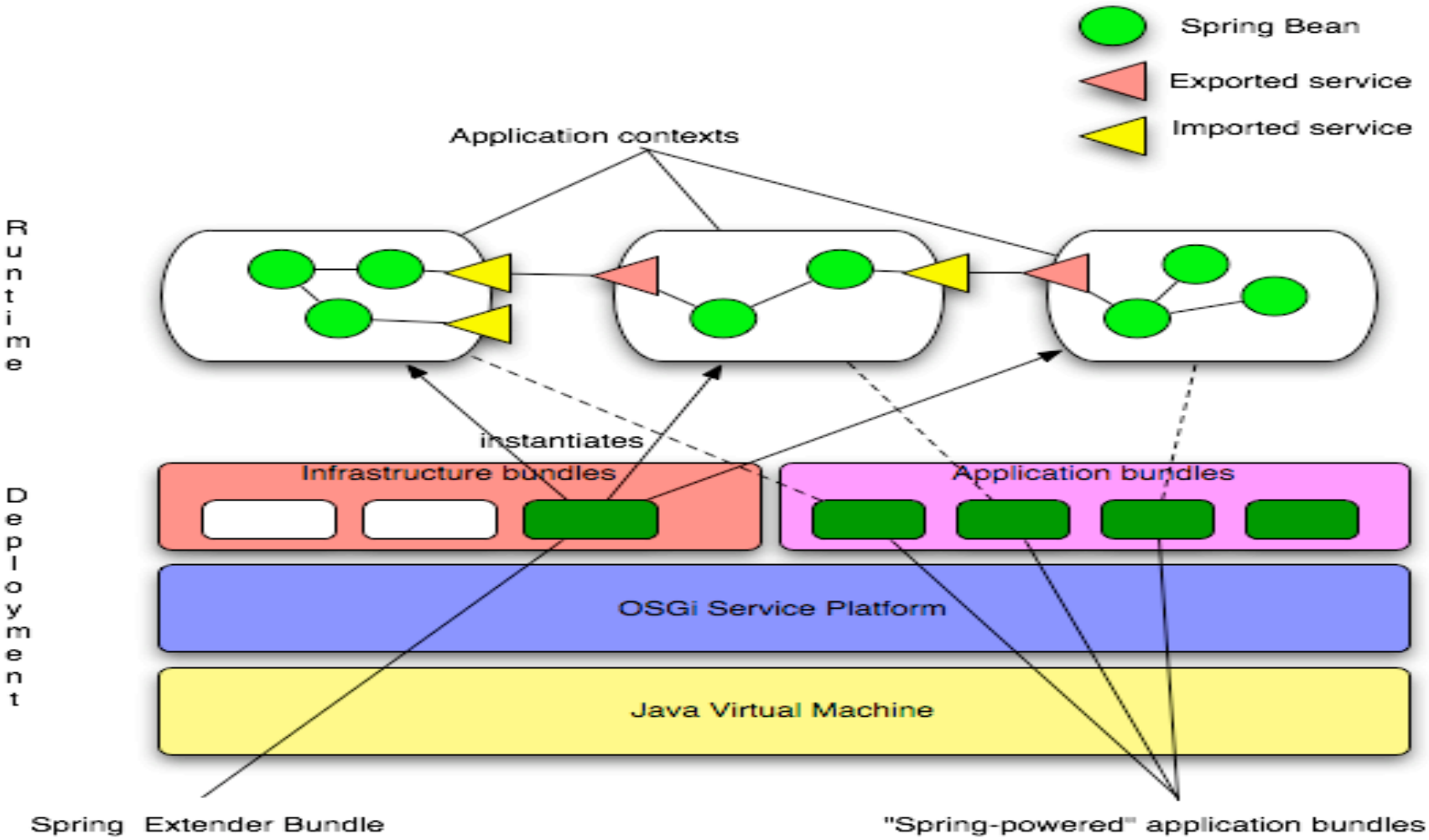
- Bundle == Application Context
  - each bundle has it's own Application Context
- Spring Extender Bundle
  - responsible for automatic context creation
- Bundle Scope
  - <bean id="…" scope="***bundle***" />

Spring Extender Bundle

- Automatically creates ApplicationContext for all ***ACTIVE*** bundles
  - no OSGi API dependency
  - no need to have an Activator
- Extender looks for:
  - META-INF/spring/*.xml
  - Spring-Context header in manifest

Demo

Packaging and Deploying



Few Words on the Context Lifecycle

- By default context creation blocks until all mandatory services are resolved
- Differs from bundle life cycle
  - bundle can be ACTIVE without having spring context created
  - context can be destroyed without stopping the bundle
- Context is published once created
  - org.springframework.context.service.name = bundleSymbolicName

Exporting Services

- ***Any*** Spring bean can be exported as an OSGi service

```
<!-- create bean -->
<bean id="personService"
          class="com...internal.PersonServiceImpl">
     <property name="personDao" ref="personDao"/>
</bean>

<!-- export services to service registry -->
<osgi:service ref="personService"
                    interface="com...services.PersonService">
</osgi:service>
```

Importing Services

- ***Any*** OSGi service can be imported

```
<osgi:reference id="personDao"
                interface="com...persistance.PersonDao">
</osgi:reference>

<bean id="personService"
      class="com...internal.PersonServiceImpl">
        <property name="personDao"  ref="personDao“/>
</bean>

<osgi:reference id=“httpService” filter=“…”
        context-class-loader="client"
        cardinality="1..1"
        interface=“org.osgi.service.http.HttpService”/>
```

Service cardinality

- ***Optional***
  - ***osgi:reference cardinality="0..1"***
- Creates a proxy and watches for service (re)appearance
- ServiceUnavaliableException after a timeout

Service Cardinality, con't.

- ***Mandatory***
  - ***<osgi:reference cardinality="1..1" />***
- Default setting
- Un-register dependent services when goes away

Service lifecycle events

- Optionally listen for
  - register/unregister event on service side
    - <osgi:registration-listener />
  - bind/unbind on reference side
    - <osgi:listener />

Handling ContextClassLoader

- <osgi:service context-class-loader="…"/>
  - unmanaged
    - default
    - anything goes
  - service-provider
    - creates a proxy
    - ensures visibility of service provider resources

Handling ContextClassLoader

- <osgi:reference context-class-loader="…"/>
  - client
    - default
    - importer's bundle classpath
  - service-provider
    - service publisher classpath
  - unmanaged
    - could be service publisher
    - no one knows like Domino's …

Testing support

- Run JUnit tests
  - from IDE
  - ant
  - maven
- Takes care of
  - OSGi platform configuration
  - OSGi manifest creation
  - test bundle lifecycle
  - wraps unit test in virtual bundle

Testing Demo

- Screencast also available on
  - http://www.springframework.org/osgi/demos

OSGi Web Applications

- Number of options
  - OSGi HttpService
    - Servlet 2.1 spec API
    - Look at PAX Web for current spec support
  - Eclipse Extension Registry
    - Equinox specific
  - Spring DM 1.1 release
    - Run war in native containers

Spring Web Support

- Don’t re-invent the wheel
  - let containers parse web.xml
- Allow deployment of war files
  - with full support for modularity
- Solve issues with
  - JSP
  - taglibs
  - etc

Q&A