



OSGi™ and the Enterprise

Ben Hale
SpringSource

ben.hale@springsource.com

- **What is OSGi?**
- Where does it come from and who is using it?
- Concepts: bundles, services, versions
- Benefits
- OSGi for the Enterprise
- SpringSource Application Platform

OSGi™

The Dynamic Module System for Java

- Partition a system in a number of modules
 - “*bundles*”
- Strict visibility rules
- Resolution process
 - satisfies dependencies of a module
- Understands versioning

- Modules can be
 - installed
 - started
 - stopped
 - uninstalled
 - updated
- ... at runtime

- Bundles can *publish* services dynamically
- Service Registry allows other bundles to *find* servers
 - and to *bind* to them
- Services come and go at runtime, all taken care of for you

- What is OSGi?
- **Where does it come from and who is using it?**
- Concepts: bundles, services, versions
- Benefits
- OSGi for the Enterprise
- SpringSource Application Platform

- Backed by the OSGi Alliance
 - <http://www.osgi.org>
- Understood the need to be lightweight and dynamic from day one
 - 1999, embedded Java and networked devices
 - 2003, extended support to mobile devices
 - 2004, significant open source adoption
 - 2006, server-side Java applications

- Eclipse Equinox
- Apache Felix
- Makewave Knopflerfish
- Prosyst mBedded Server Profession Edition

- Automotive
 - BMW, Siemens, Volvo, ...
- Mobile
 - Nokia, Motorola, ...
- SmartHome
 - Philips, Bosch, Siemens, ...
- Enterprise
 - SpringSource, IBM, BEA, Oracle, ...

- What is OSGi?
- Where does it come from and who is using it?
- **Concepts: bundles, services, versions**
- Benefits
- OSGi for the Enterprise
- SpringSource Application Platform

- OSGi Service Platform
 - Lightweight container
 - Standalone or embedded
 - OSGi Service Platform Core Specification R4.1
 - “Compendium Services” also available

- The fundamental unit of deployment and modularity in OSGi
- Just a jar file
 - Some required entries in **META-INF/MANIFEST.MF**

- Common manifest headers
 - Bundle-SymbolicName**
 - Bundle-Version**
 - Bundle-Name**
 - Bundle-ManifestVersion**
 - Bundle-Vendor**

- Used to declare package-level dependencies of your bundle
 - `Import-Package: com.xyz.foo`
 - `Import-Package: com.xyz.foo;version="1.0.3"`
 - `Import-Package: com.xyz.foo;version="[1.0.3, 1.1.0)"`
 - `Import-Package: com.xyz.foo;version="[1.0.3, 1.1.0)", com.xyz.bar;version="[1.0.3, 2.0.0)"`

OSGi does not define version semantics, only a syntax for a version

- Version syntax
 - **major.minor.micro.qualifier**
 - major ::= number
 - minor ::= number
 - micro ::= number
 - qualifier ::= (alphanum | '_' | '-') +

Common when dealing with bundles with well defined versioning semantics

- Version Ranges

- "at least" → "1.0.3"

- interval

- inclusive → "[version, version]"

- exclusive → "(version, version)"

- common

- "[version, version)"

Uses is used to ensure that complex wirings use the same bundles

- Declare types that other bundles can import
 - `Export-Package: com.xyz.foo`
 - `Export-Package: com.xyz.foo;version="1.0.5"`
 - `Export-Package: com.xyz.foo;version="1.0.5", com.xyz.bar;version="2.0.0"`
 - `Export-Package: com.xyz.foo;version="1.0.5"; uses:="org.abc"`

Find: Also a version that takes a String filter expression

- Import and Export objects as opposed to types

- Publish

```
- ServiceRegistration reg =  
  bundleContext.registerService (Bar.class.getName () , null) ;  
  ...  
  reg.unregister () ;
```

- Find

```
- ServiceReference ref =  
  bundleContext.getServiceReference (Bar.class.getName () ) ;
```

Registry maintains a reference count

- Bind

```
- Bar bar = (Bar) bundleContext.getService(ref);  
...  
bundleContext.ungetService(ref);  
// bar should no longer be used here
```



Demo

Bundles in OSGi

- What is OSGi?
- Where does it come from and who is using it?
- Concepts: bundles, services, versions
- **Benefits**
- OSGi for the Enterprise
- SpringSource Application Platform

- Strong Modularity
- Operational Control - life cycle
- Versioning Support

- <http://blog.springsource.com/main/2008/05/15/why-should-i-care-about-osgi-anyway/>

- By default a bundle is a black box
–isolated from other bundles
- A bundle can *export* one or more packages

Strictly enforces development best-practices.

Helps get to ideal development situation.

- Only exported packages are visible outside of the exporting bundle
 - stops unintended coupling between modules
 - enables independent development
 - faster development cycles

- See all modules and their status
- Get information on wiring
- Install new bundles
- Active bundles (and publish services)
- Deactivate bundles (and unregister services)
- Update bundles
- Stop bundles
- Uninstall bundles

All without stopping or restarting the application



Demo

Life Cycle

A service bundle...

Service interface types
exported (with version
information)

Export-Package: a.b.c

private implementation
packages

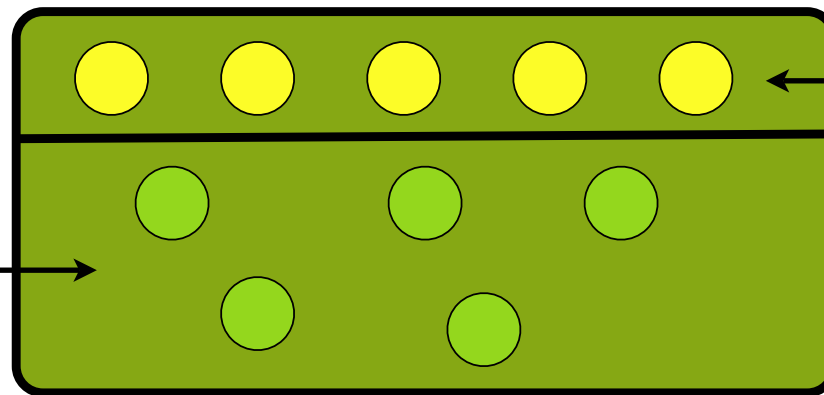
Service implementation
locked away

“Passive” contribution

- types added to type space
- bundles see new version on resolution after install/refresh

A service bundle...

Published services

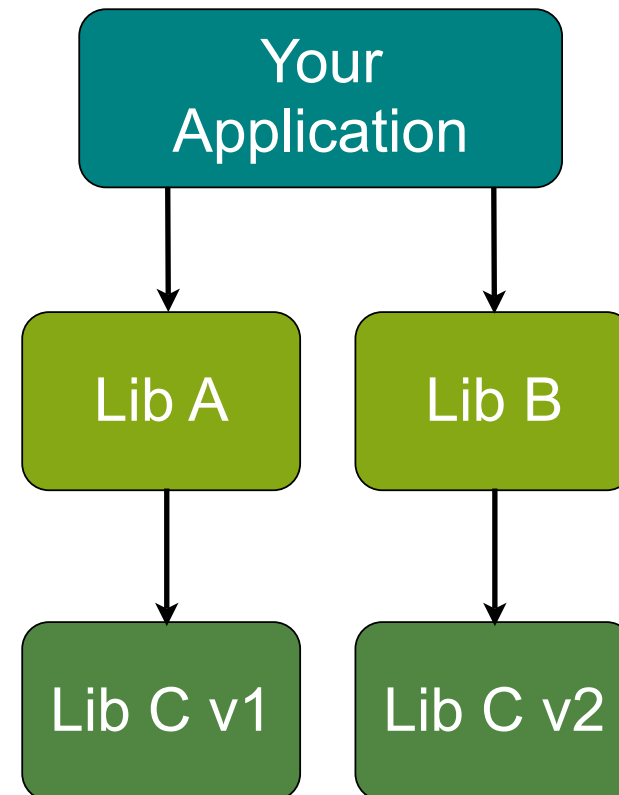


Private implementation
objects

“Active” contribution

- services published in registry
- bundles see service changes immediately

- Packages are imported
- Can have multiple versions of a package concurrently





Demo

Versioning

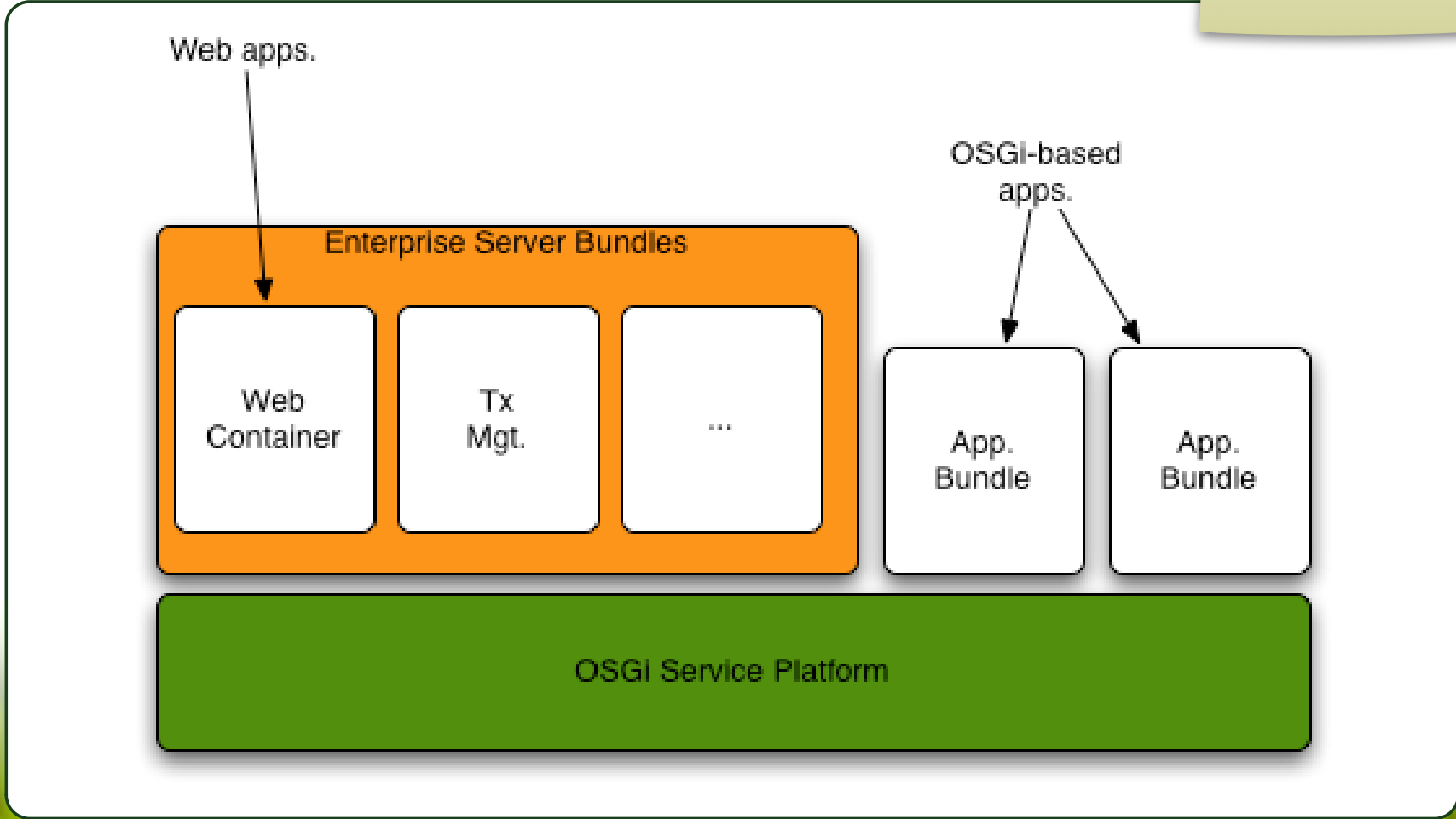
- What is OSGi?
- Where does it come from and who is using it?
- Concepts: bundles, services, versions
- Benefits
- **OSGi for the Enterprise**
- SpringSource Application Platform

- Running OSGi on the server-side
- Application design considerations
- Using existing enterprise libraries

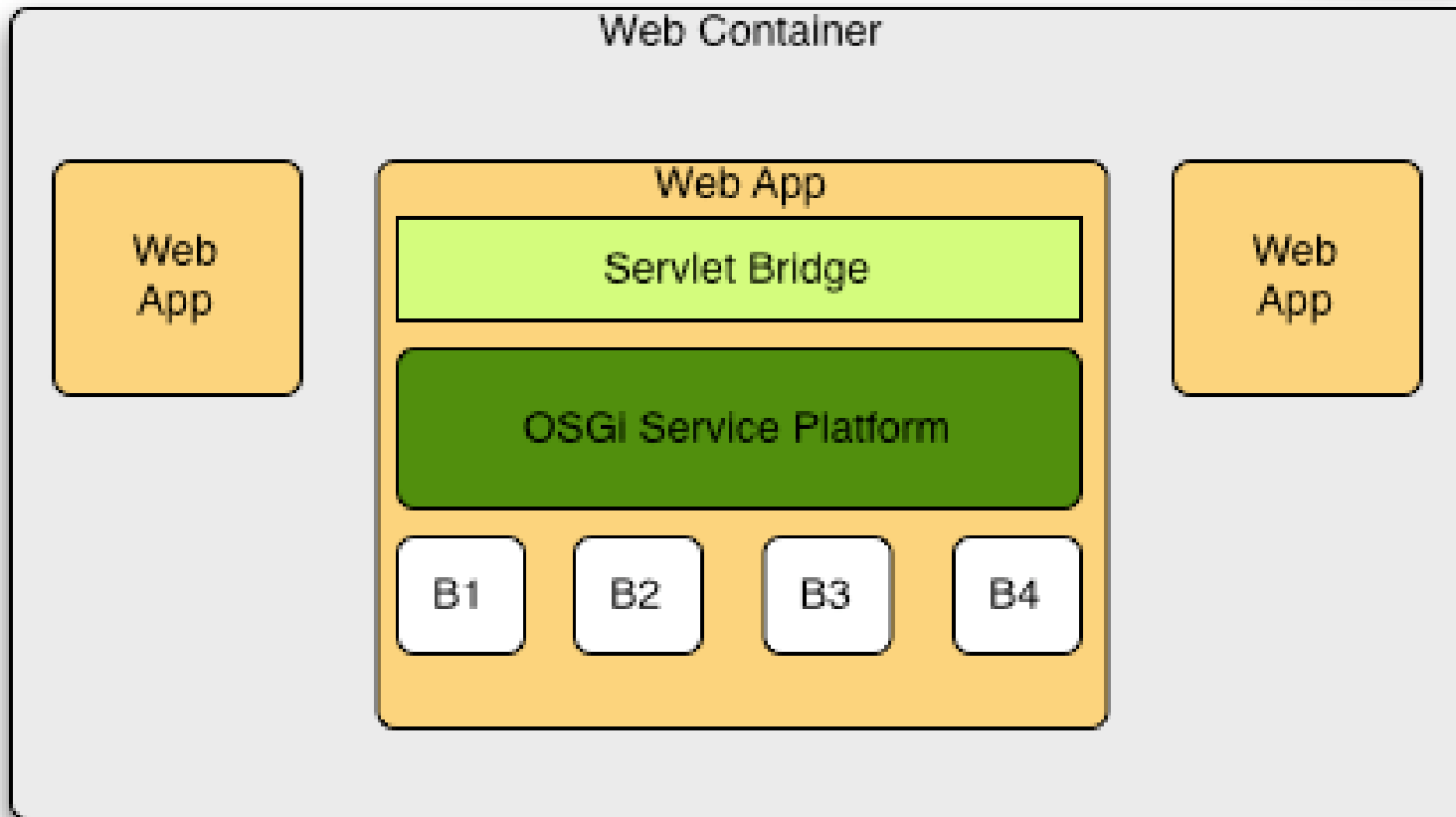


OSGi as a Server Pla

Examples:
Glassfish,
WAS

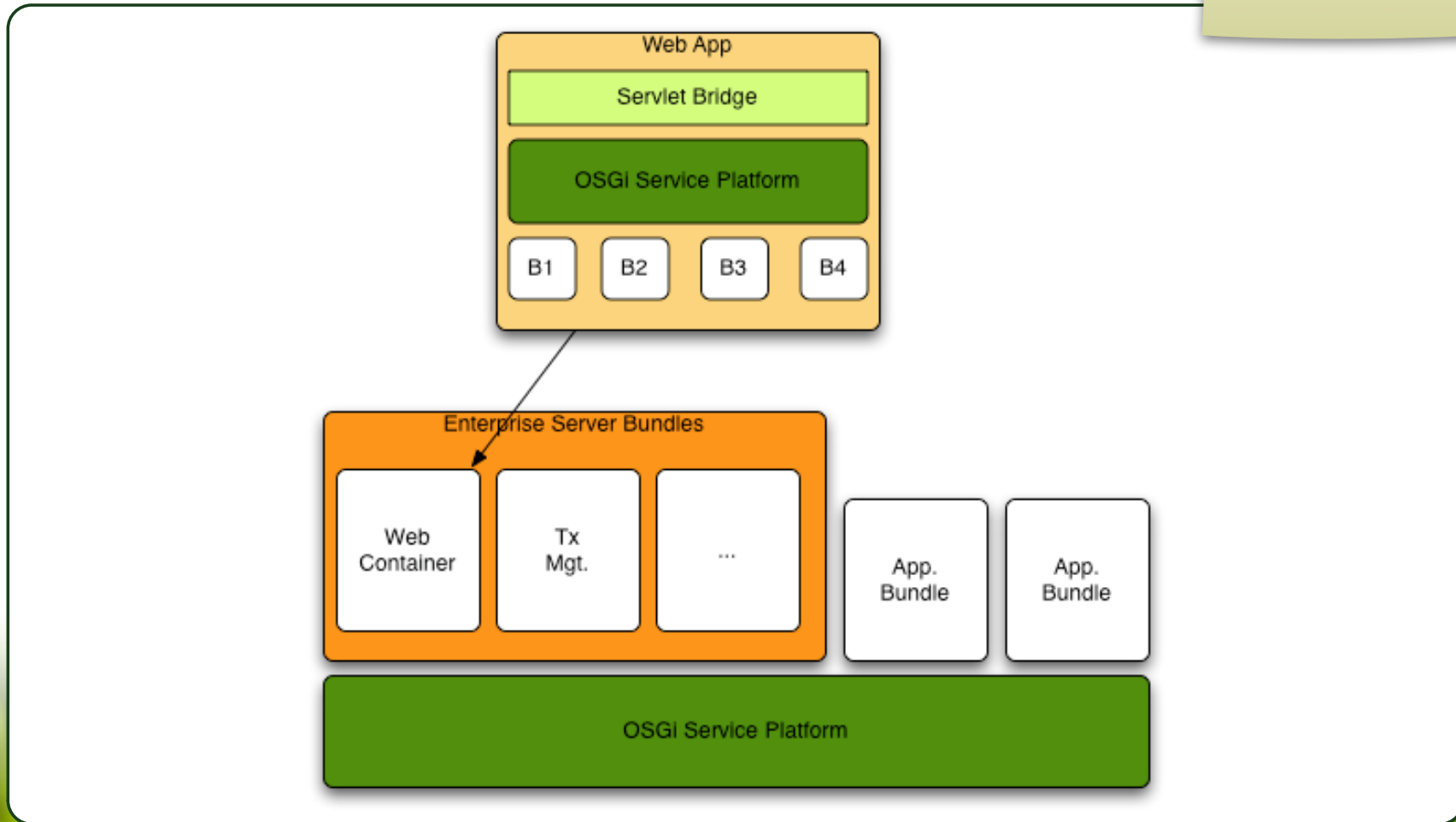


Examples:
Roll your own



Examples:

SpringSource
Application
Platform



- Application becomes a set of co-operating bundles
 - Vertical decomposition first
 - Then horizontal
- Communication via the service registry



Demo

OSGi Petclinic

- Platform dynamics
 - Services may come and go at any time
 - **ServiceTracker**
- Asynchronous activation
- Service dependency management
- Testing
- Concurrency and thread management

- Code designed without OSGi in mind may run into class and resource-loading problems
 - class visibility
 - `Class.forName()`
 - context class loader
 - resources in **META-INF/**



Enterprise Libraries under OSGi

- Good news everyone! Spring 2.5 and most other Spring projects are OSGi-ready
 - modules shipped as bundles
 - all class loading behaves correctly under OSGi
- Also, hundreds of other enterprise libraries are packaged for use under OSGi at the SpringSource Enterprise Bundle Repository

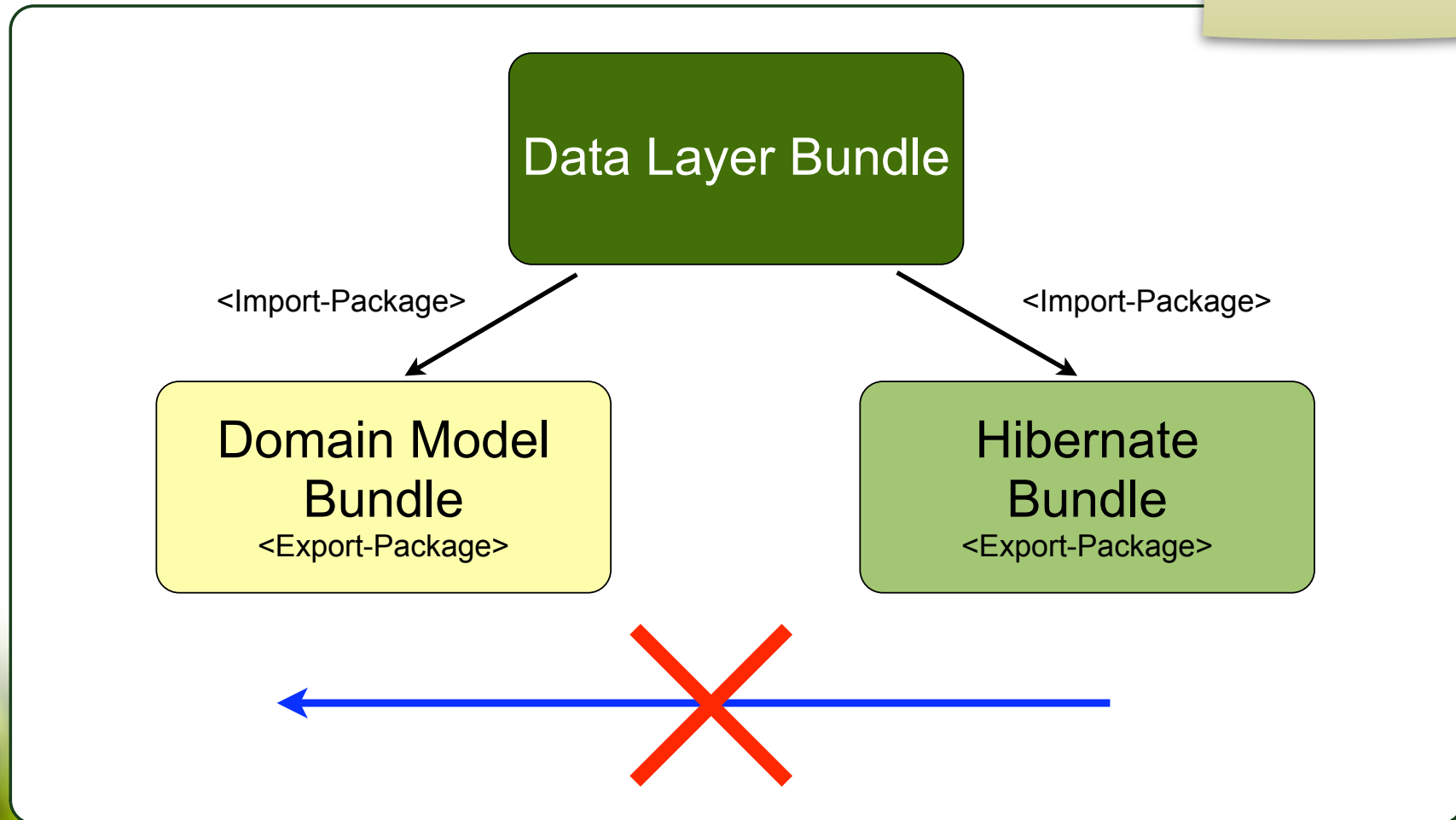


Demo

SpringSource Enterprise Bundle Repository

Domain Bundle:
Domain types,
mapping files

Hibernate Bundle:
SessionFactory



- **Dynamic-ImportPackage**
 - a last resort, too broad a scope
 - does not affect module resolution
- **Fragments**

- Heavily used in enterprise Java
- Expected to have visibility of application types + classpath
- ContextClassLoader is undefined in OSGi!
 - No notion of context
 - No notion of “application”

- Solutions:
 - Eclipse Equinox: Context Finder
 - Spring Dynamic Modules: CCL management
 - SpringSource Application Platform!

- OSGi HttpService
 - register Servlets and resources under aliases
 - programmatically configuration
- Equinox Http Registry bundle
 - register servlets and resources using Eclipse extension registry
- BUT... only Servlet 2.1 spec, limited use

- OSGi offers an excellent foundation
- What if we want to build enterprise applications on top of it?
- Need to exploit the power and sophistication of OSGi
 - without adding complexity
 - retaining the ability to use familiar enterprise libraries and approaches

- What is OSGi?
- Where does it come from and who is using it?
- Concepts: bundles, services, versions
- Benefits
- OSGi for the Enterprise
- **SpringSource Application Platform**

- Bundle components need
 - instantiating
 - configuring
 - assembling
 - decorating
- “Bundle blueprint” when a bundle is started
 - Should we be coding this ourselves?*

- Need an easy way to expose bundle objects as services
 - ... and wire service references between bundles
- Don't want to work with error-prone resource acquisition/release APIs

- Need an easy way to manage dynamics
 - what happens when services go away and come back?
 - new services are published, old services removed
 - broadcast operations

- Don't want hard dependencies on running inside of OSGi
 - keep environmental assumptions out of code - "The Spring Way"
- Avoid lookups and unnecessary dependencies on OSGi APIs
- Enable testing outside of container
 - unit testing
 - simple integration testing



SpringSource Application Platform

- The newest member of the SpringSource family
- The simplicity and power of Spring...
- ... with the dynamic module system of OSGi
- All in a single pre-package runtime

- A Spring `ApplicationContext` based on an OSGi bundle
- Uses bundle context and classloader to load resources
- Implements Spring's resource abstraction for OSGi
 - relative resources paths resolved to bundle entries
 - “bundle:” prefix for explicit specification



Creating a Bundle Application Context

- Possible to create a bundle application context programmatically...
- ... but like Spring's `ContextListenerLoader` you don't typically
- SpringSource Application Platform detects context files and creates one for you
 - configuration files in `META-INF/spring`

- Starting with an ordinary jar file containing classes and resources for a module
 - mymodule.jar
- Add needed headers to **META-INF/MANIFEST.MF**
 - Bundle-SymbolicName
 - Bundle-Version
 - Bundle-ManifestVersion: 2



Demo

SpringSource Application Platform

- OSGi is a dynamic module system for Java
- Key benefits: modularity, operational control, versioning
- Key disadvantages: error-prone APIs, integration testing, working with existing libraries not designed for OSGi
- See the SpringSource Application Platform for solutions



Q & A

Ben Hale
SpringSource

ben.hale@springsource.com