Introduction to SpringSource dm Server

Rod Johnson

Agenda

- Why a new server?
- Clearing up common misconceptions
- Background: OSGi 101
- dm Server tour
- Migrating to dm Server
- The Future

Motivation: The Problem

- Java EE servers are big and heavy
- Traditional stovepipe server makes less and less sense
  - Push toward SOA means different deployment models
  - Java EE APIs less relevant (web applications) or not relevant at all (SOA applications)
  - Want a la carte middleware
- Limitations of Java EE deployment units

Java EE Deployment units: The Good

- Widely understood
- Supported by tooling and build scripts

- …not many other positives

- *Mainly talking about WARs, although most comments apply also to EARs*

Java EE Deployment units: The Irritating

- Duplication of configuration
  - web.xml deployment descriptor just bootstraps a framework in a modern application
  - Duplication, possible errors
- Excessively large WARs and EARs
  - The application contains half the server in a modern application
  - More code in libraries than in Tomcat
  - Duplication between applications
- *Deployment units don't actually make much sense – we're just used to them*

Java EE Deployment Units: The Bad

- Usually need to restart application to upgrade any business logic
  - What if you need content cached (LinkedIn)?
  - Why can't you upgrade dynamically with service references replaced transparently?
- Classpath hell
  - What if you have a version conflict between libraries?
  - What if a library your application uses conflicts with on in the server?
  - **Java EE classloading is not sophisticated enough to resolve these problems, which are becoming more and more painful**

Java EE Deployment Units: The Bad

- LinkedIn experience
  - *1 WAR with N services does not scale for developers (conflicts, monolithic).*
  - *N wars with 1 service does not scale for containers (no shared JARs). You can add containers, but there's only 12GB of RAM available.*

Motivation: What should the solution look like?

- Server must be modular, to minimize footprint
  - Must not be tied to a particularly application type
- Must allow us to modularize applications *dynamically*, allowing for change at runtime
- Must be able to avoid conflict between libraries (isolation)
- Must allow for library dependencies to be moved to server, to shrink deployment units

Enter *SpringSource dm Server™*

- Squarely addresses these problems
- Modular server
- Can run standard WAR files, but introduces modular deployment units built on OSGi concepts
- Not tied to particular deployment units

dm Server

- Modular
  - Choose exactly the modules you need
  - Extend with your own modules
- Serviceability as a central design tenet
  - Per application tracing, logging, data capture on failure
  - Deadlock detection
- Integrated Tomcat

Designed for Spring Applications

- Native support for Spring Framework and Spring Dynamic Modules
- Understands common Spring usage patterns
  - Avoid repetitive configuration
  - Concentrate on application-specific tasks

Clearing up misconceptions

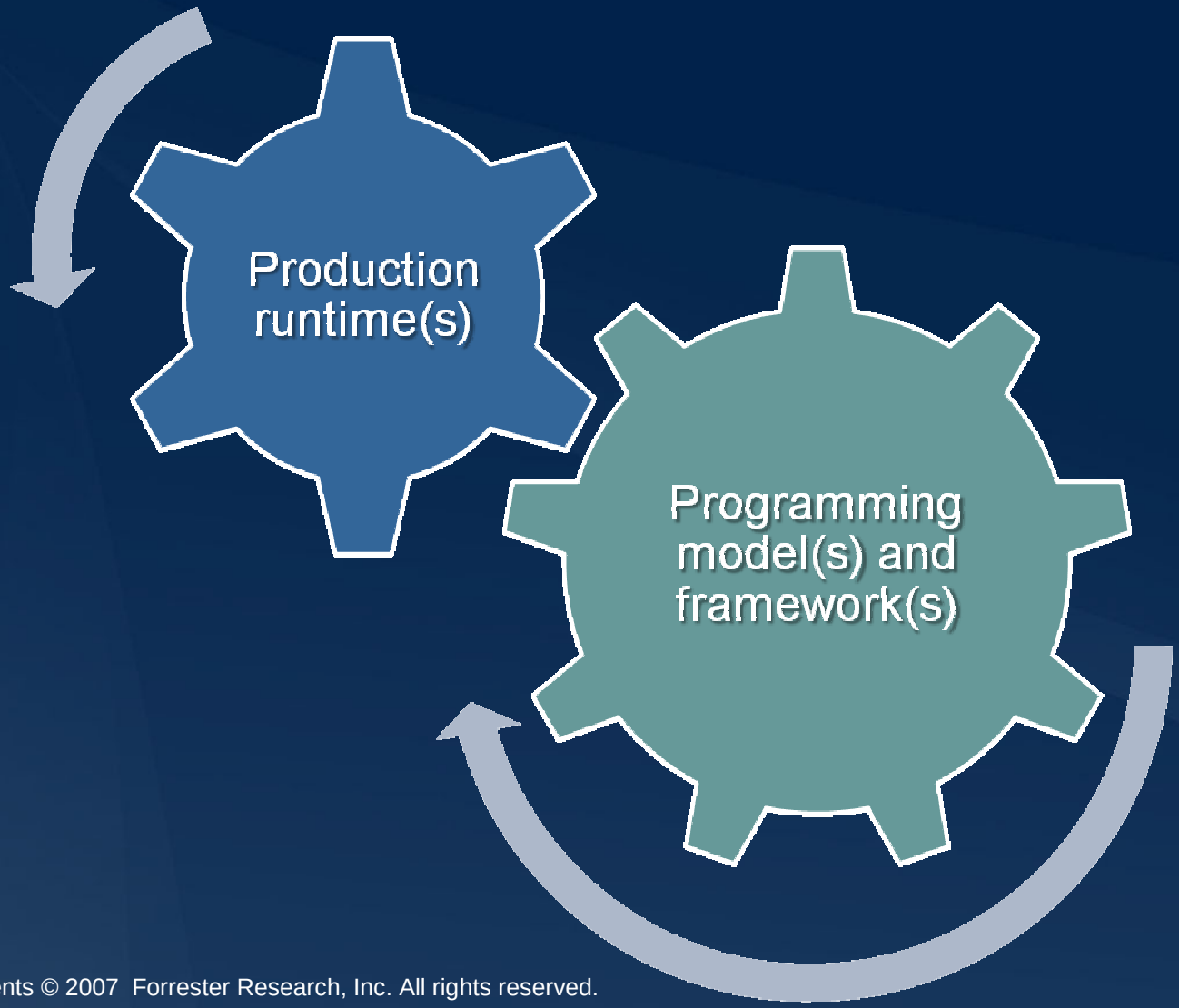
- Spring and dm Server
- Spring Dynamic Modules and dm Server
- Open source
- Standards

Spring and dm Server

- *Is Spring becoming a server?*
- No. Spring is a portable component model.
- *Will Spring be tied to dm Server?*
- Spring remains portable across all environments, as the de fact standard component model for enterprise Java.

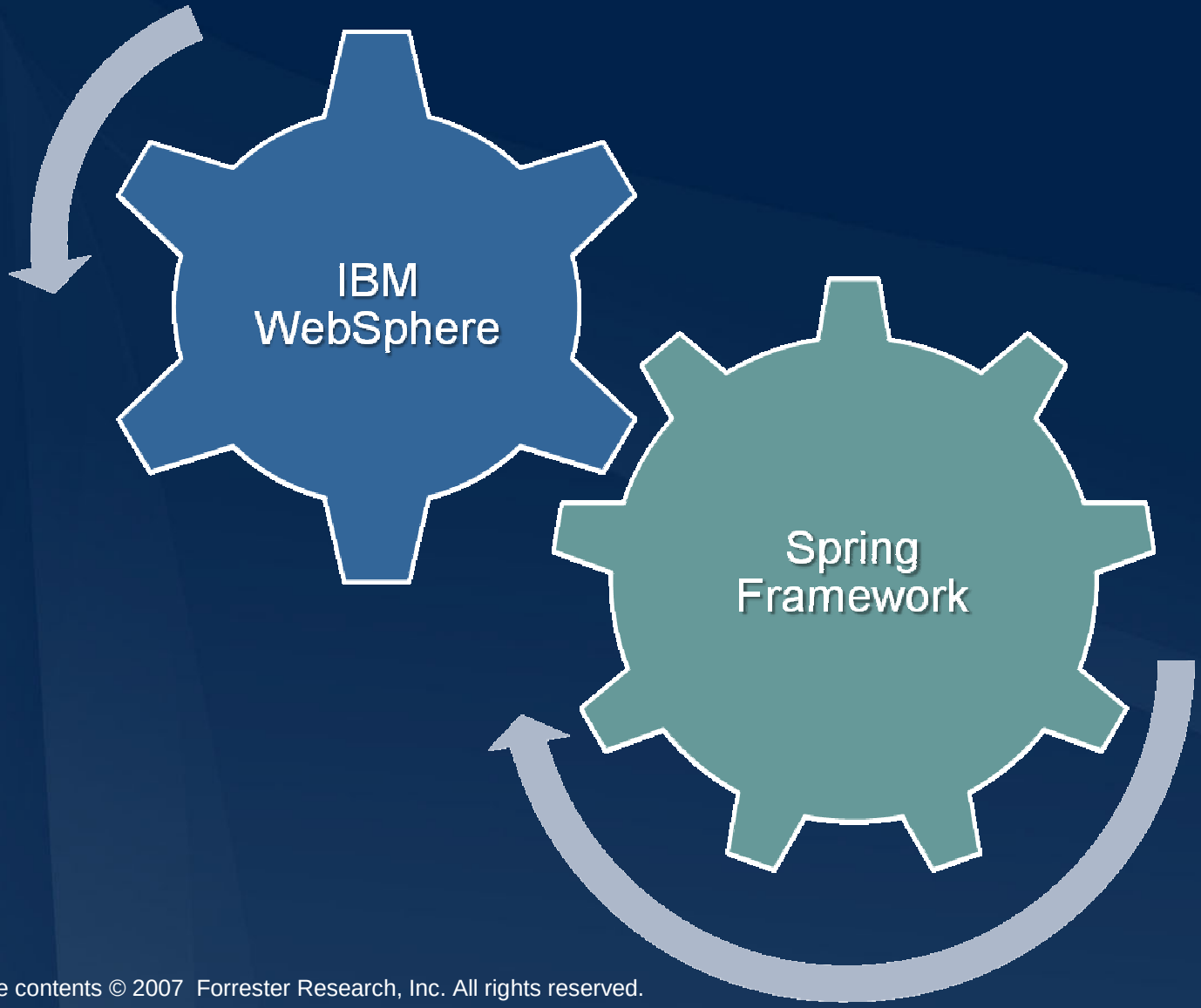
dm Server is a new product that brings the familiar Spring values to the server arena

Independently evolving spheres allow fluid innovation in Java platforms now



FORRESTER®

… making possible mixing and matching



Spring DM and dm Server

- *How does SpringSource dm Server relate to Spring Dynamic Modules?*
- Spring Dynamic Modules is an implementation of OSGi RFC124 and a building block for dm Server
- Not intended for independent use

Spring DM provides the programming model for Spring dm Server
Spring POJOs + OSGi dynamicity

dm Server and standards

- dm Server is not defined by any single standard but by solving real problems
- Implements important standards from the JCP and other standards bodies
- Likely to implement Java EE 6 web profile
  - Will never implement the whole Java EE legacy model

dm Server and standards (2)

- Fully honors OSGi bundle semantics
  - Bundle activation etc.
- Servlet Spec
  - 2.5
  - JSTL
  - JSP
  - Can run standard WAR files
- Spring DM standardization
  - Blueprint Service
  - OSGi RFC 124

Background: OSGi

- The Dynamic Module System for Java
- *Part* of the solution
- But not enterprise-ready or very useable on its own

It's a Module System…

- Partition a system into a number of modules
  - "bundles"
- Strict *visibility* rules
- *Resolution* process
  - satisfies dependencies of a module
- Understands *versioning*

...and it's (potentially) Dynamic

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

It's even Service Oriented

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

Where does OSGi come from?

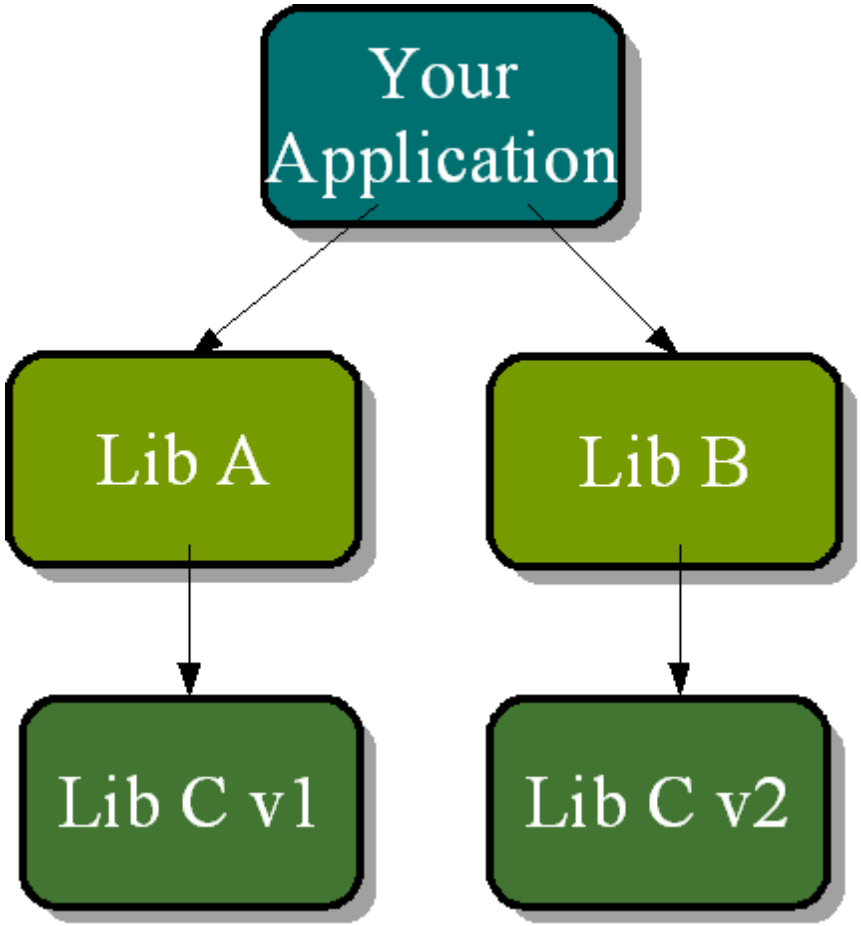
- Backed by the OSGi Alliance
- Understood the need to be lightweight and dynamic from day one
  - started in 1999, focus on embedded Java and networked devices
  - 2003 extended support to mobile devices
  - 2004 significant open source community adoption
  - 2006 server-side Java applications

OSGi and Modularity

- By default a bundle is a black box
  - isolated from other bundles
- A bundle can *export one or more packages*
  - optionally with version information
- Only exported packages are visible outside of the exporting bundle
  - stops unintended coupling between modules
  - enables independent development
  - faster development cycles

Versioning

- Packages are imported
  - optionally with version information
- Can have multiple versions of same package concurrently



Operational control

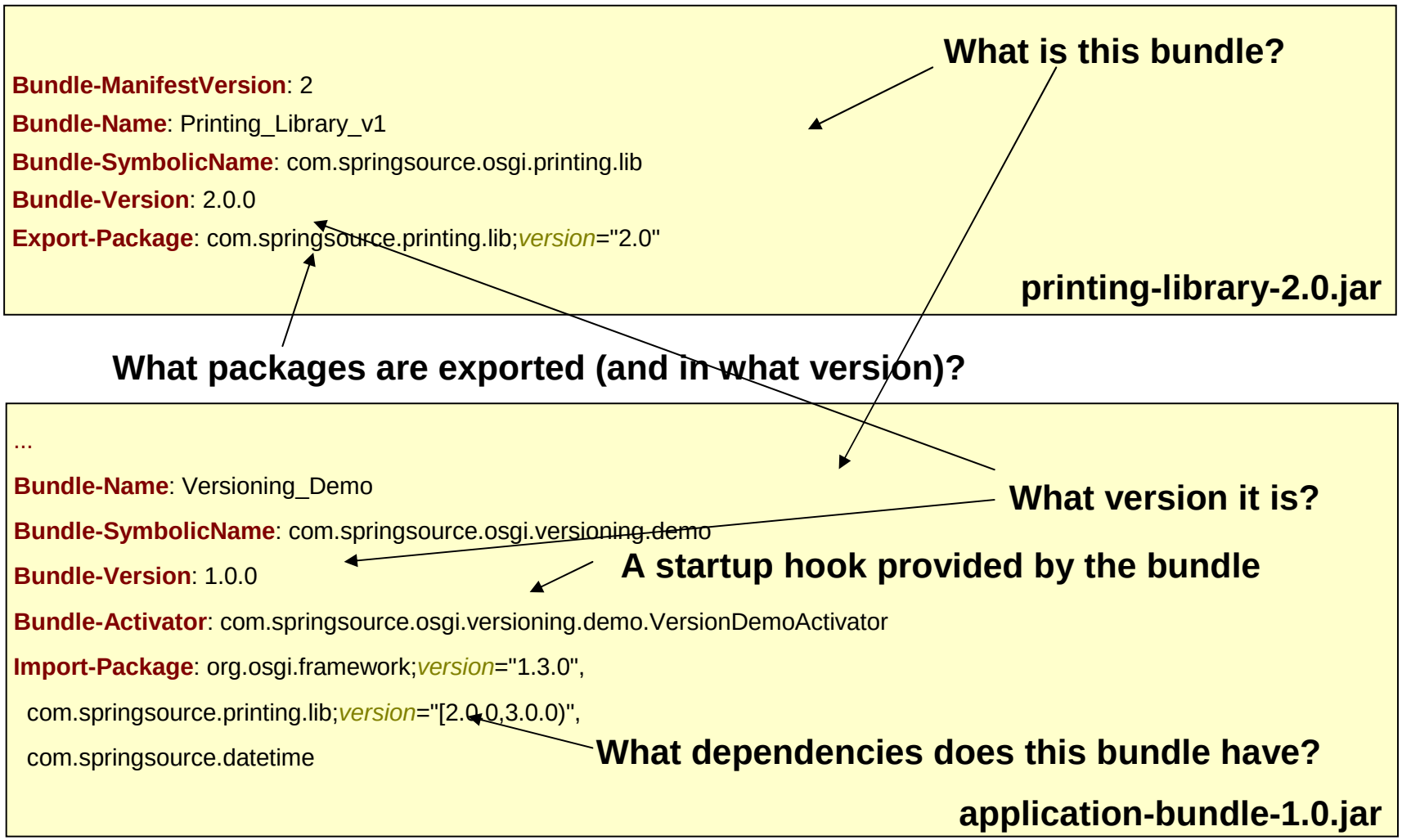
- See all modules and their status
- Get information on wiring
- Install new bundles
- Activate bundles (and publish services)
- Deactivate bundles (and unregister services)
- Update bundles
- Stop bundles
- Uninstall bundles

All without stopping or restarting the application

Essential Concept: Bundle

- The fundamental unit of deployment and modularity in OSGi
- Just a JAR file
  - OSGi metadata in META-INF/MANIFEST.MF
- Manifest headers (from MANIFEST.MF) define:
  - bundle identification (symbolic name and version)
  - what does the bundle expose (exported packages)
  - what dependencies need to be satisfied in order for the bundle to be functional (imported packages)

MANIFEST-MF (excerpts)



Essential Concept: Service Registry

- 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());`
  - also a version that takes a String filter expression

Essential Concept: Service Registry

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

Application Design

- Application becomes a set of co-operating bundles
  - vertical first
  - then horizontal
- Communication via service registry

Concept is great
Not so easy to bring it to reality

OSGi challenges

- Design considerations
  - Asynchronous activation
    - service dependency management
  - Platform dynamics
    - services may come and go at any time
  - Testing
- Enterprise Libraries under OSGi
  - Code designed without OSGi in mind is likely to run into class and resource-loading problems

What about Enterprise applications on OSGi?

- 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 ability to use familiar enterprise libraries and approaches
- Split application into a number of OSGi bundles

Within a bundle

- Bundle components need
  - instantiating
  - configuring
  - assembling
  - decorating
- When a bundle is started
  - "bundle blueprint"
- Should we code this ourselves?

Between bundles

- Need 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 service removed
  - broadcast operations
  - etc.

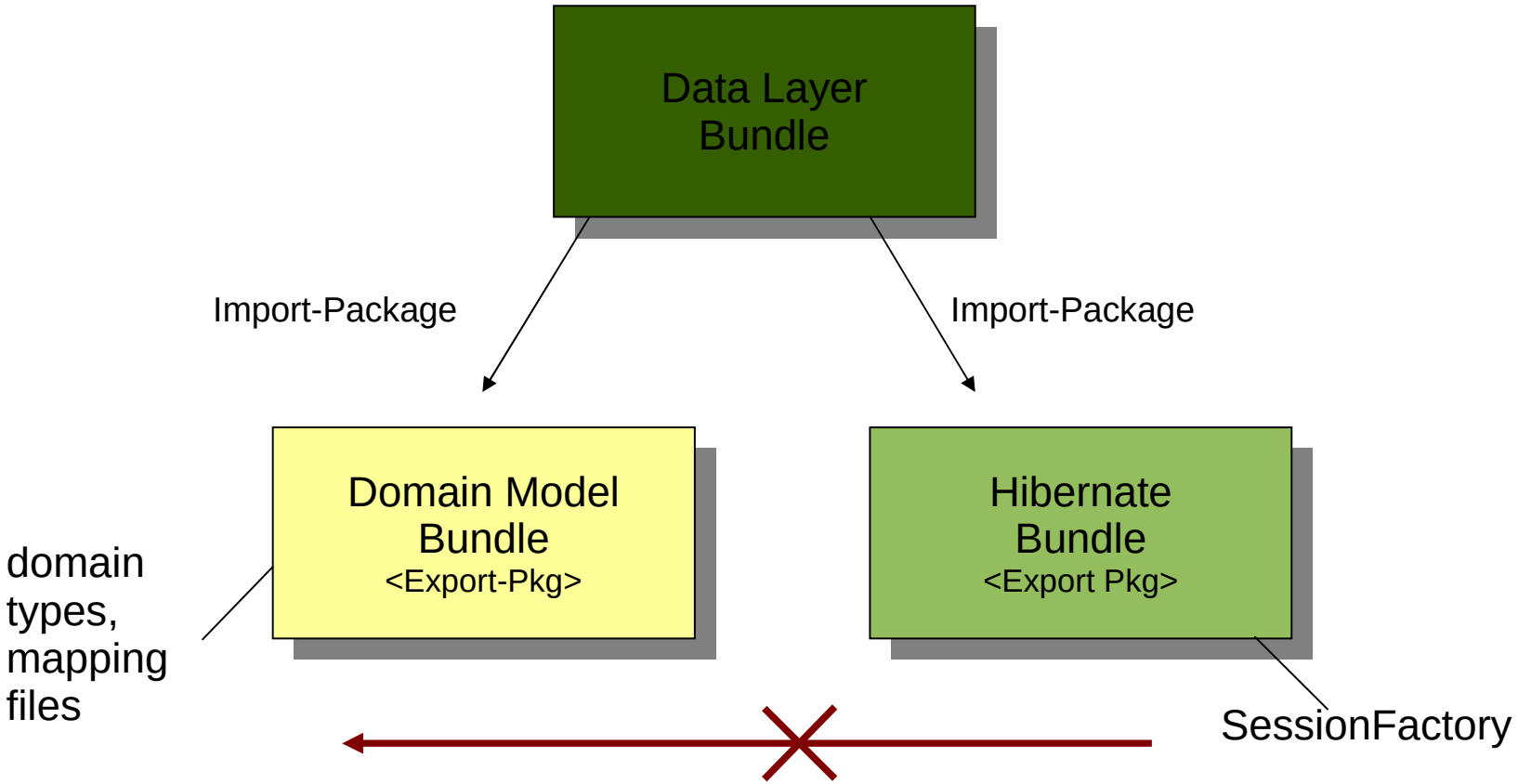
Preserve ability to test

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

Enterprise Libraries and Frameworks under OSGi

- Class and resource-loading problems
  - class visibility
  - context class loader
  - classpath scanning
  - load time weaving
  - resources in META-INF
  - …

Example: Class visibility



Other questions

- How do you stop a service / type from not leaking out of an application? There are no applications in OSGi…
- Basic infrastructure for OSGi + Web has to be done by yourself
- What do we do about WARs?

dm Server solves all these problems, and more

dm Server concepts

- Module
  - A bundle with a component model (Spring application context)
- Library
  - Collection of bundles
  - Built on OSGi standard
- PAR file
  - Collection of modules forming an application
  - Provides new scoping concept

Library concept

- Allows developers to express their intent in their own language, not that of OSGi
- Developers want to say "I want to use Hibernate 3.3.2," not import bundles individually
- Allows developers to use third party services to define and manage libraries
- Allows organizations to provide governance for what libraries are available

Library example

- A library is defined in a .libd file
- Bundles are OSGi standard
- SpringSource dm Server expand library info as macros at runtime

```
Library-SymbolicName: org.hibernate.ejb
Library-Version: 3.3.2.GA
Library-Name: Hibernate JPA
Import-Bundle:
   com.springsource.org.hibernate;version="[3.2.6.ga, 3.2.6.ga]",
 com.springsource.org.hibernate.annotations;version="[3.3.1.ga, 3.3.1.ga]",
 com.springsource.org.hibernate.annotations.common;version="[3.3.0.ga,
   3.3.0.ga]",
 com.springsource.org.hibernate.ejb;version="[3.3.2.GA, 3.3.2.GA]",
 com.springsource.javax.persistence;version="[1.0.0, 1.0.0]"
```

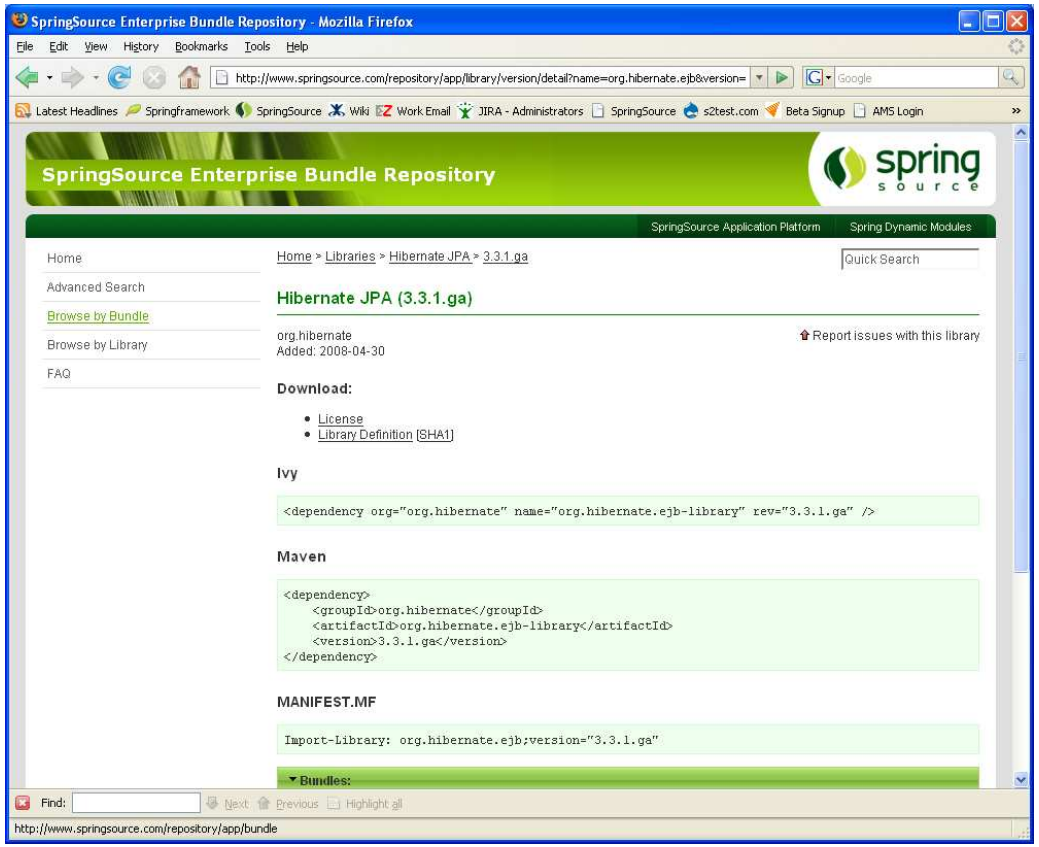
Sourcing Libraries and bundles

- Obstacles to reaping the benefit of OSGi in the enterprise

Obstacle	Solved by
Incompability between OSGi and Java EE	Java EE becoming less important OSGi moving into server space
No POJO programming model for OSGi	Spring Dynamic Modules makes Spring OSGi-ready Spring becoming the default component model for OSGi
Bundle concepts too low level	SpringSource dm Server *library* concept
Where do we source OSGi bundles for enterprise Java?	SpringSource Enterprise Bundle Repository

SpringSource Bundle Repository

- Over 400 popular enterprise libraries and bundles
- One stop shop for enterprise Java code ready for OSGi use



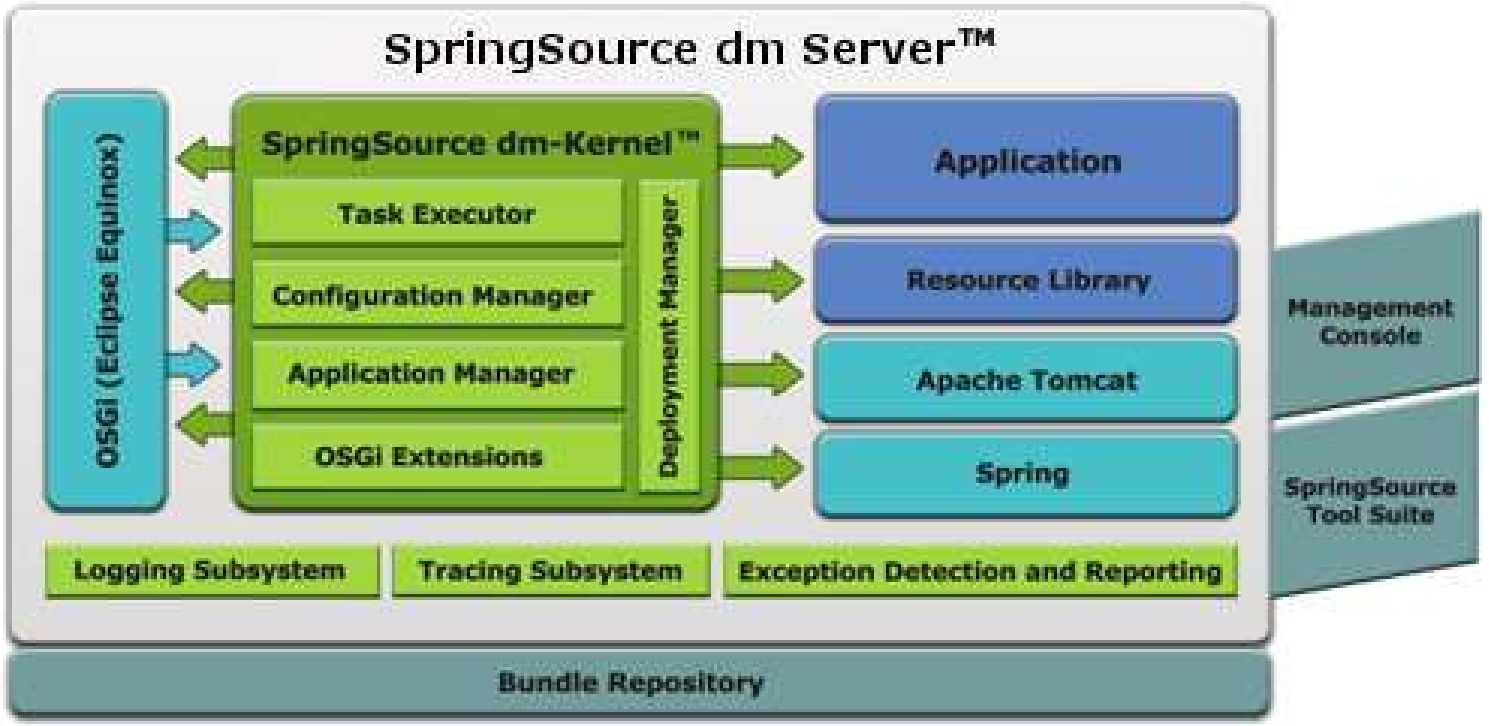
- Ivy/Ant
- Maven
- Dependencies
- Version

dm Server: What's in the box

- Tomcat 6.0.18
- Equinox 3.4
- Spring Framework 2.5.5
- Spring DM 1.1.1
- AspectJ 1.6.1

- A lot of new code in the dmKernel
  - OSGi management

dm Server architecture



Dm Server tour

- Directory structure
- Serviceability
- Startup

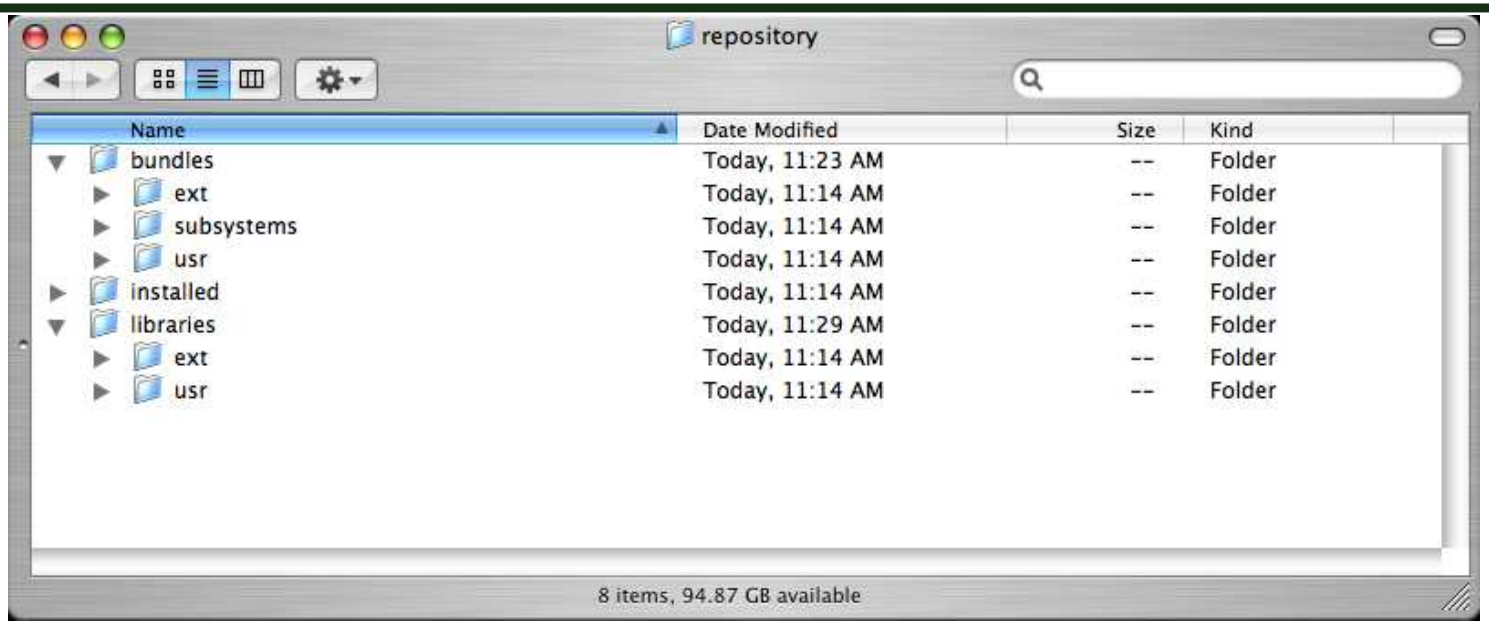
dm Server directory structure

/pickup	applications go here
/repository	provisioning repository (shared bundles)
/serviceability	
/dump	dumps in event of errors, subfolder for each
/trace	application-specific logging
/logs	main server log and web access logs
/config	all server configuration

The Repository

- Conceptually, everything in dm Server is a bundle
- Applications, server infrastructure itself (above dm Kernel bootstrap subsystem)
- Repository contains shared bundles

Inside the repository



- /bundles contains all the bundles available in the repository
- /libraries contains all library definitions
- /installed is for server use only
- Contents of /ext directories are supplied with the server
- **You should install artifacts in the /usr subdirectories**

Serviceability

- dm Server is designed with serviceability in mind
  - Simple example: Log conventions
- Dumps allow for enhanced diagnostics and support for the server itself
- Partitioned trace logging simplifies application management

Logging conventions

Log codes are of the form: '**SPXY1234L**', where SP stands for *Spring Platform*, **XY** stands for the subsystem code, **1234** represents the error number, and **L** conveys the level of severity of the event being logged

Subsystem codes

- CC - Concurrent
- CN - Control
- CO - Config
- DE - Deployer
- FF - FFDC
- KB - Bootstrap
- KE - Kernel
- OP - OSGi
- PM - Profile
- SC - Servlet
- WE - Web

Event severity

- E - Error
- W - Warn
- I - Info

```
[2008-11-20 11:46:03.312] main          <SPKB0001I> Server starting.
[2008-11-20 11:46:10.718] main          <SPOF0001I> OSGi telnet console availab
[2008-11-20 11:46:14.578] main          <SPKE0000I> Boot subsystems installed.
[2008-11-20 11:46:15.640] main          <SPKE0001I> Base subsystems installed.
[2008-11-20 11:46:16.359] server-dm-5    <SPPM0000I> Installing profile 'web'.
[2008-11-20 11:46:17.593] server-dm-1    <SPSC0001I> Creating HTTP/1.1 connec
      scheme http on port 8080.
```

Dump analysis

- Any exception in dm Server code triggers a dump
- Application code never triggers a dump

- Each dump produces a subfolder under /serviceability/dump
- Contains
  - Summary, including exception trace
  - Heap dump (if configured)
  - Information about server config

Tracing

- Global trace.log in serviceability/trace
- Application traces are written to specific trace log in /serviceability/trace/<application-name>

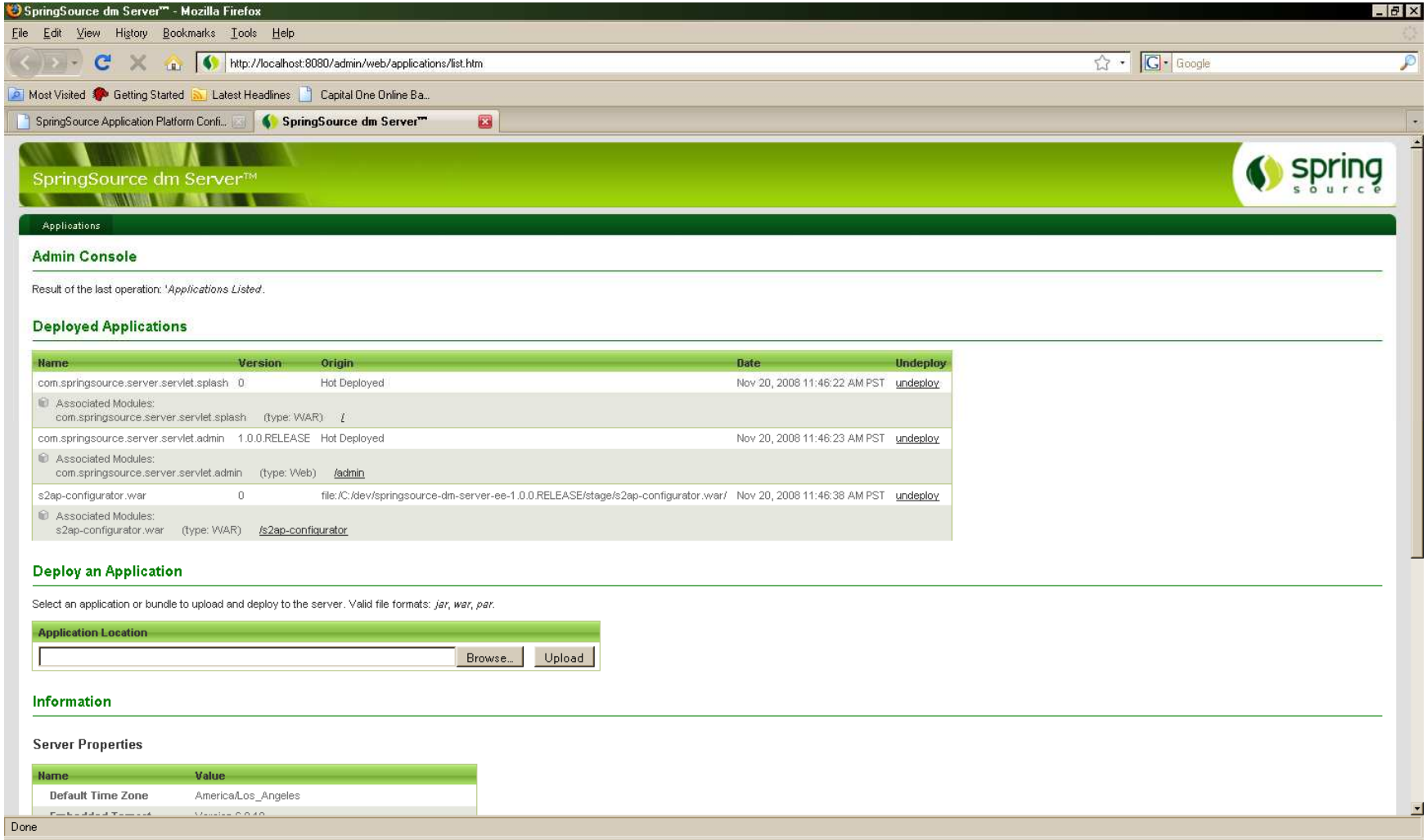
- dm Server knows how to capture output from Commons Logging and Log4j for this routing
- Also captures and routes System.out and System.err

Configuration

- JSON format

```
/*
 * SpringSource dm Server profile manager default configuration file.
 */
{
    "profile": {
            "version" : 1.0,
            "name" : "web",
            "subsystemsDir" : "repository/bundles/subsystems",
            "subsystems" : ["com.springsource.server.deployer",
    "com.springsource.server.jndi", "com.springsource.server.servlet",
    "com.springsource.server.web"],
            "optionalSubsystems" : []
    }
}
```

Admin console: /admin



OSGi console

- telnet <host> 2401
- Demo
  - status command
- Can disable for security

JMX

- Start server with –jmxremote to enable JMX connectivity
- Allows to take service dump etc.

MIGRATING TO DM SERVER

WAR-based deployment options: Incremental adoption path

- Standard WAR
  - Get started immediately deploying your existing web apps
- *Shared Library* WAR
  - Share libraries by explicitly importing libraries
- *Shared Services* WAR
  - Share libraries and services between applications

“Shared Library” WAR

META-INF/MANIFEST.MF (from WAR)

```
Manifest-Version: 1.0
Import-Library:
   org.springframework.spring;version="2.5.4“,
   org.hibernate.ejb;version="[3.3.2.GA,3.3.2.GA]"
Import-Package: javax.annotation
```

- No more library bloat in WEB-INF/lib
- Just import libraries/versions
- Just deploy your business logic and resources, not half your server
  - Much smaller WAR files
  - Faster deployment

What distinguishes dm Server from the competition?

- Other server vendors are moving to OSGi to modularize their servers
  - IBM
  - Oracle
  - GlassFish
  - Even Red Hat
- But we modularize applications, not just servers

If OSGi is so good you want to use it to build servers, why not to build user applications also?

Shared Services WAR

- Share services as well as libraries
- Motivation
  - To share services between web applications as well as libraries
  - Avoid inefficient, complex remoting
  - Allow dynamic upgrade of services across multiple applications
- Realization
  - Requires a programming model for dynamic modularization

Spring Dynamic Modules

- Provides the best programming model for code running in dm Server
- The simplicity and power of Spring…

  …with the dynamic module system of OSGi

Spring DM Objectives

- Use Spring container to configure modules
- Make it easy to publish and consume services
  - across a number of bundles
- Enable applications to be coded without dependency on OSGi APIs
  - Easy integration testing
- Provide the needed bundles and infrastructure to deploy OSGi based applications to application servers

Spring DM Objectives

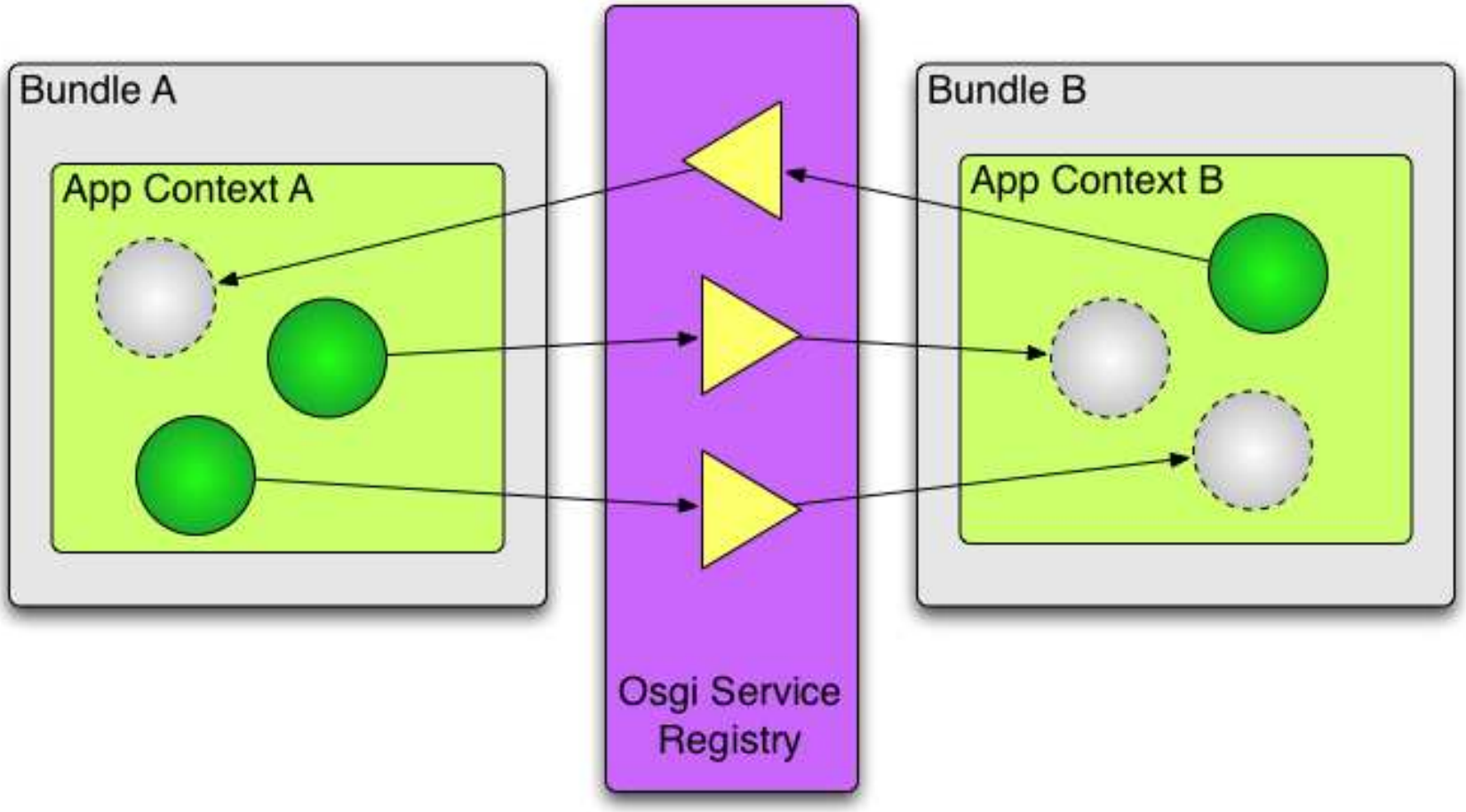
Basically

POJO development on OSGi

Bundles and Module Contexts

- OSGi bundle <==> Spring Application Context
  - we call it a *module context*
- *Module context created when bundle is started*
  ...destroyed when bundle is stopped
- *Module components <==> Spring beans*
  - *instantiated, configured, decorated, assembled by Spring*
- *Components can be imported / exported from OSGi service registry*

Beans and Services



Bundles and contexts

- Spring application context (module context) per bundle (module)
  - created automatically for you by Spring extender bundle
  - no need to depend on any OSGi APIs

From JAR file to Spring bundle…

- 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: org.xyz.myapp.mymodule
  - Bundle-Version: 1.0
  - Bundle-ManifestVersion : 2
- Place configuration files in META-INF/spring

Exporting a service

```
<bean id="printService"
   class="com.springsource.osgi.print.internal.PrintServiceImpl"
   init-method="init"
   destroy-method="destroy"/>

<osgi:service ref="printService"
        interface="com.springsource.osgi.print.PrintService"/>
```

- *Any* Spring bean can be exported as OSGi service
- *Any* Spring application is OSGi-ready, for dm Server or any OSGi environment

Importing a service

```
<bean id="printClient"
   class="com.springsource.osgi.print.client.Client"
   init-method="init">
   <property name="printService" ref="printService"/>
</bean>

<osgi:reference id="printService"
        interface="com.springsource.osgi.print.PrintService"/>
```

- Locates the best OSGi service that matches the description
- Handles the service dynamics internally

Service Dynamics

- Context creation
  - blocks until all mandatory service references are satisfied
  - simply start your bundles and let Spring Dynamic Modules figure it out

Service Dynamics

- What happens if..
  - there isn't a matching service?
  - there are several matching services?
  - a matched service goes away at runtime?
  - new matching services become available at runtime?
- Spring DM provides a resolution mechanism for each case

Shared Services WAR

- Still a WAR with OSGi metadata
- Allows publishing and consuming services (normally using Spring OSGi configuration)
- Improved modularity
- Reduces footprint if running multiple web applications

Web Modules and PARs

- "Native" OSGi formats
- Web Module is an OSGi bundle
  - Gets rid of WAR, web.xml entirely
  - "Native" Spring MVC and Spring Framework recognition
- PAR
  - EAR for the OSGi generation

Regular bundle (Module) example: Exports Service

- *OSGi Bundle*
- */META-INF/MANIFEST.MF*

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: GreenPages Service
Bundle-SymbolicName: greenpages
Bundle-Vendor: SpringSource Inc.
Bundle-Version: 1.0
Import-Library:
org.springframework.spring;version="[2.5.5.A,3.0.0)"
Export-Package: greenpages;version="1.0"
```

Consuming the module

- In a *web module*
- */META-INF/MANIFEST.MF*

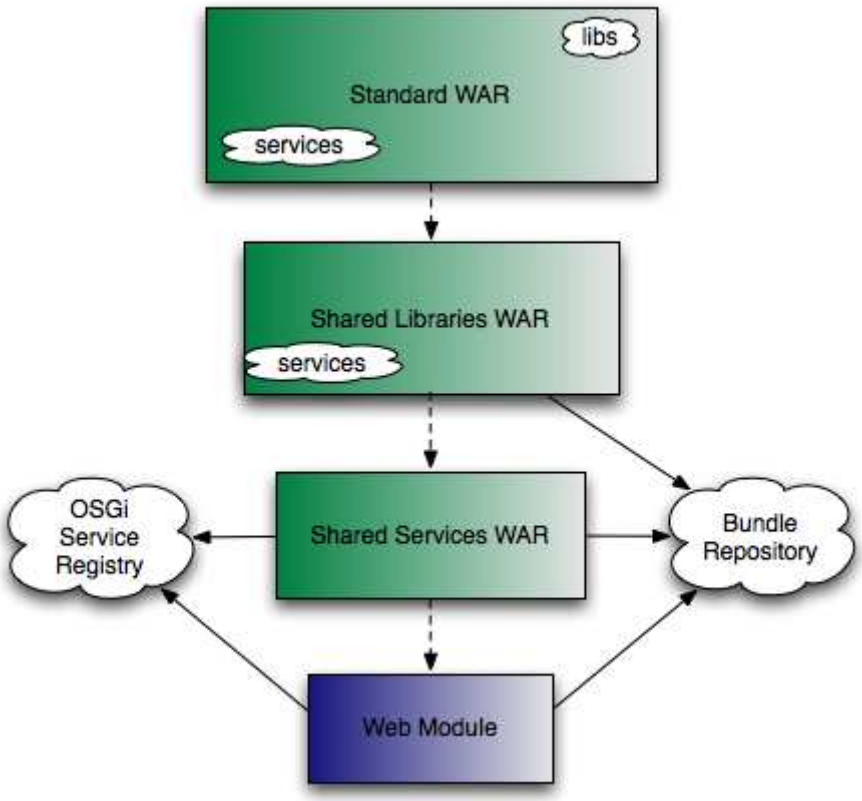
```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: GreenPages Web
Bundle-SymbolicName: greenpages.web
Bundle-Vendor: SpringSource Inc.
Bundle-Version: 1.0
Import-Library:
org.springframework.spring;version="[2.5.5.A,2.6.0)"
Import-Package: greenpages;version="[1.0.0,1.0.0]"

Module-Type: web
Web-ContextPath: greenpages
```

What is a Web Module?

- An OSGi bundle
- Module-Type: Web
- Optional Spring-specific headers
- Automatic configuration of Spring MVC Dispatcher server defined by Spring DM ApplicationContext
  - Loaded from /META-INF/spring/*.xml

Deployment unit summary



Commercial vs Open Source

SpringSource Application Platform Ver. 1

- Open Source Edition
  - GPL + EPL
- Enterprise Edition
  - Commercial license
  - Certified, indemnified, warranted
  - SpringSource Application Management Suite
  - SpringSource Tool Suite
  - SpringSource Advanced Pack for Oracle
  - **Available to Spring subscription support customers**

SpringSource Application Platform Ver.2

- High Availability features

Eclipse Tooling: dm Server tools

- dm Server tools works with Spring IDE
  - Open source
- Comprehensive SpringSource Tool Suite support built into SpringSource Tool Suite
  - SpringSource subscription customers

The Future

- "Personality" concept
  - Different deployment units
  - Batch server
- Distributed deployment, provisioning and management
  - Come to Spring One to hear the roadmap

Summary

- SpringSource dm Server is a next generation application platform
- Modular from the ground up
- Includes the ability to modularize your applications
- Builds on OSGi foundations to provide an enterprise-ready solution
- **Hopefully this talk has helped to encourage you to try it and know what to expect**