Server-Side OSGi

Adrian Colyer
CTO, Interface21

Agenda

- What is OSGi?
- What problems does it help us to solve?
- OSGi on the server-side
- Spring Dynamic Modules project

OSGi™

- The Dynamic Module System fo

It's a Module System...

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

...and it's 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 fin services
  - and to bind to them
- Services come and go at runtime, all taken of for you

Where does it come from?

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

Implementations

- Eclipse Equinox
- Apache Felix
- Makewave Knopflerfish
- Prosyst mBedded Server Professional Editior

OSGi in action

- Automotive
  - BMW, Siemens, Volvo, ...
- Mobile
  - Nokia, Motorola, ...
- SmartHome
  - Philips, Bosch, Siemens, ...
- Enterprise
  - IBM, BEA, Oracle, Interface21, Paremus, ...

How does OSGi help *me*?

Benefits

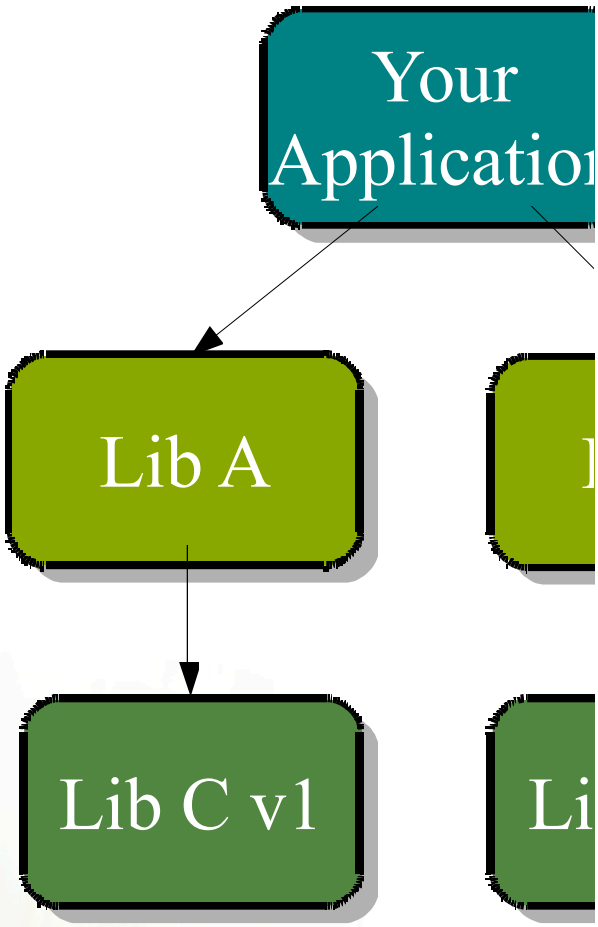
- Strong modularity
- Versioning support
- Operational control – life cycle

Modularity

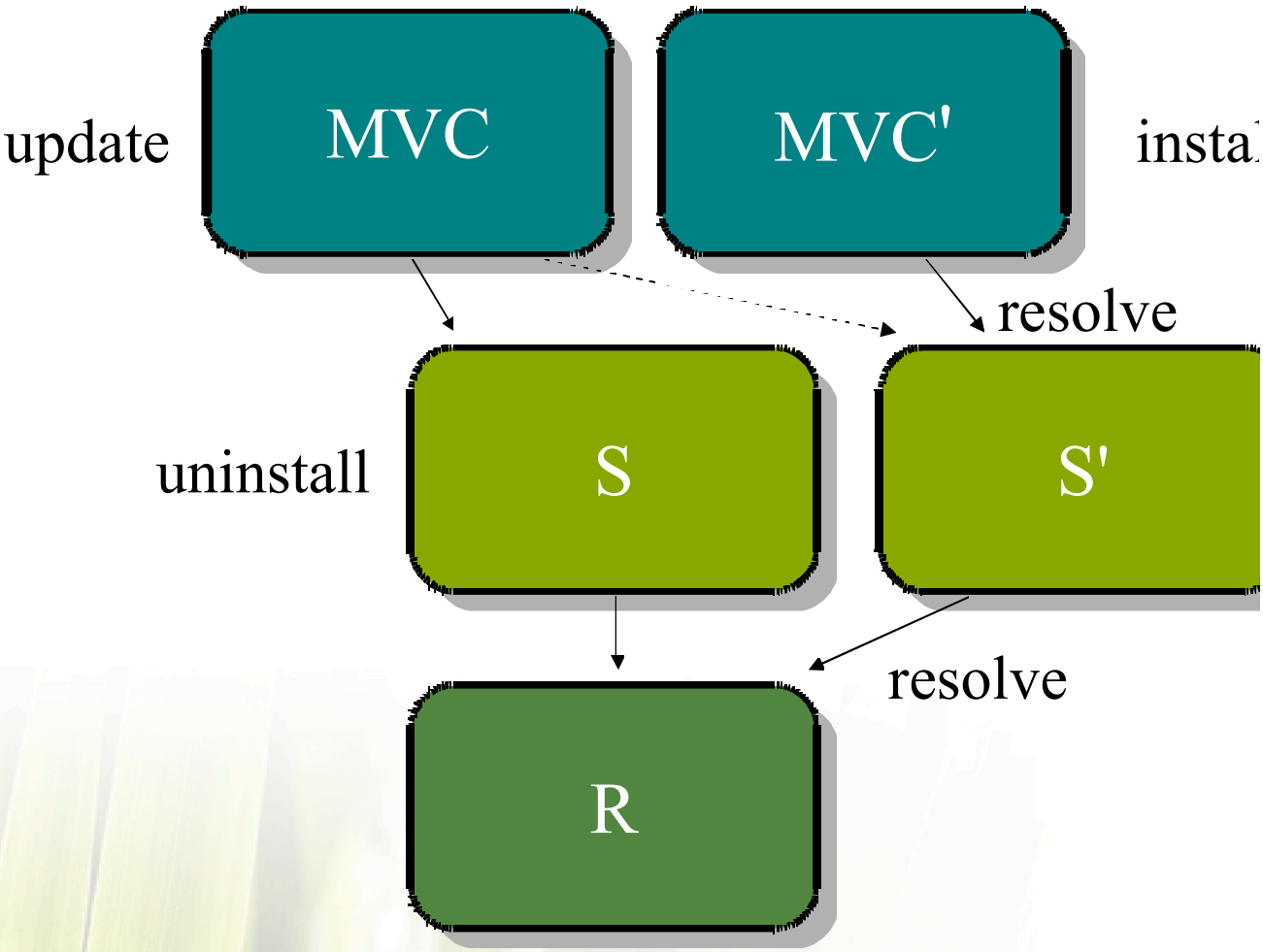
- By default a bundle is a black box
  - isolated from other bundles
- A bundle can export one or more packages
  - optionally with version information
- Only exported packages are visible outside the exporting bundle
  - stops unintended coupling between modules
  - enables independent development
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

Versioning

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



Update Scenarios



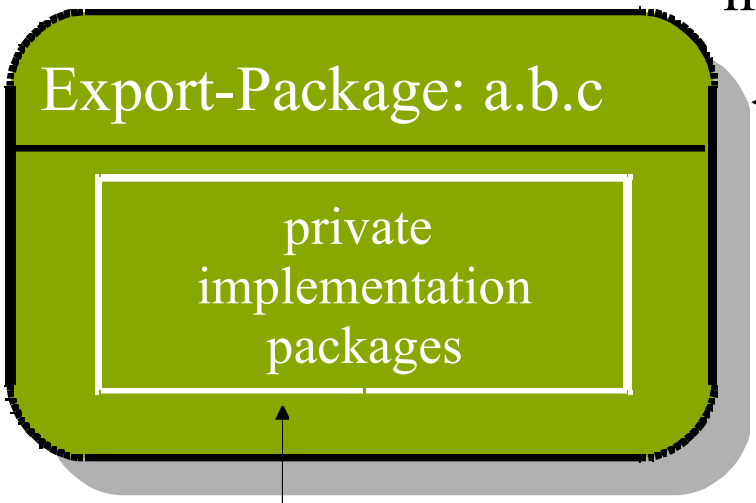
Operational Control

- See all modules and their status
  - OSGi console
  - JMX
- 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

Dealing with dynamics

A service bundle…



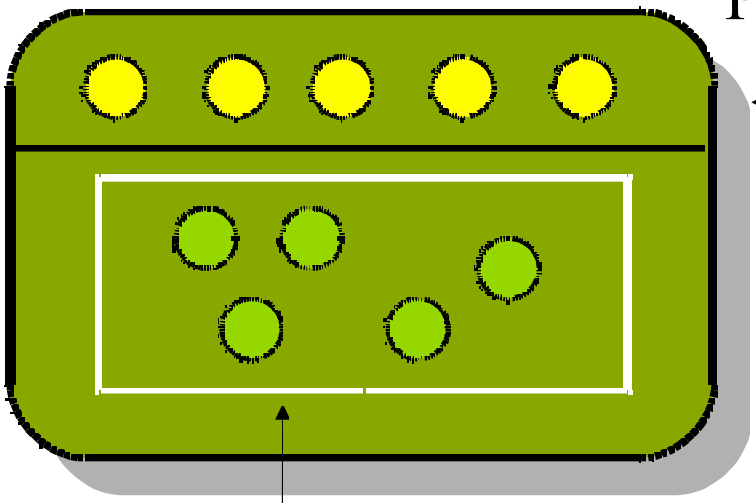
Service inter
exported [wi
information]

Service implementation
locked away

"Passive" contribution
• types added to type s
• bundles see new vers
resolution after install

Dealing with dynamics

A service bundle…



Private implementation objects

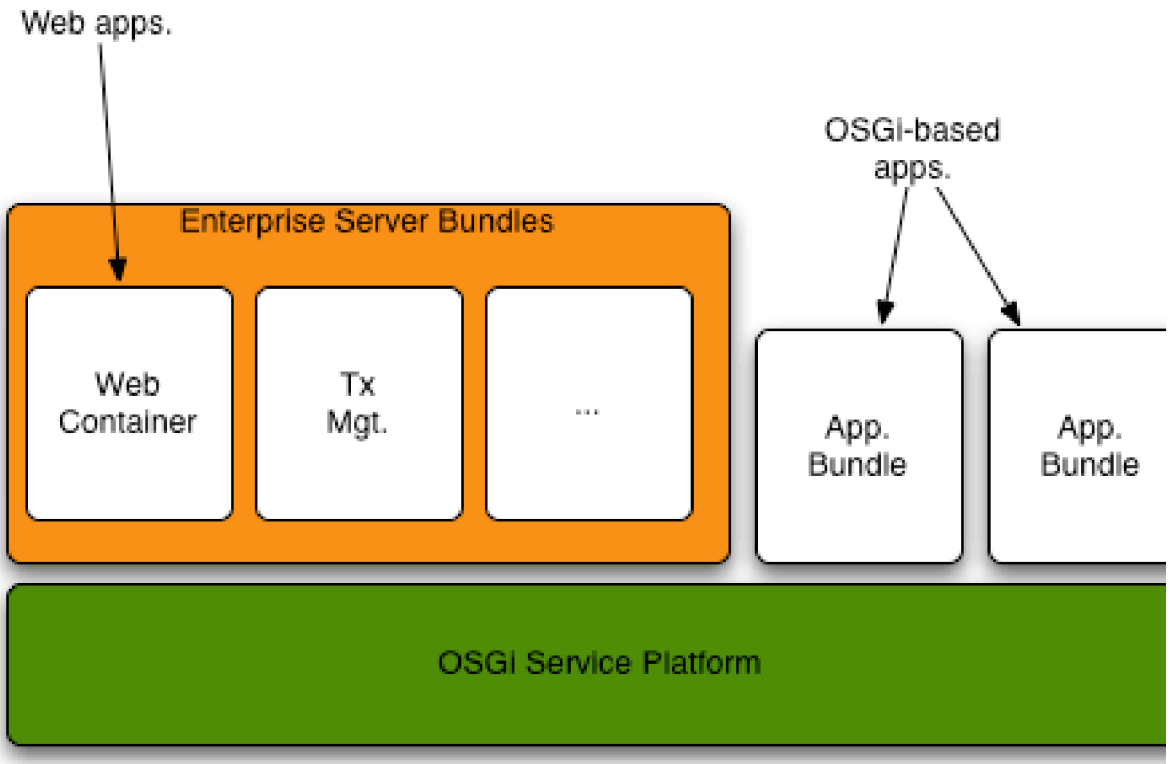
"Active" contribution
- services published in
- bundles see service cl immediately

OSGi on the Server-Side

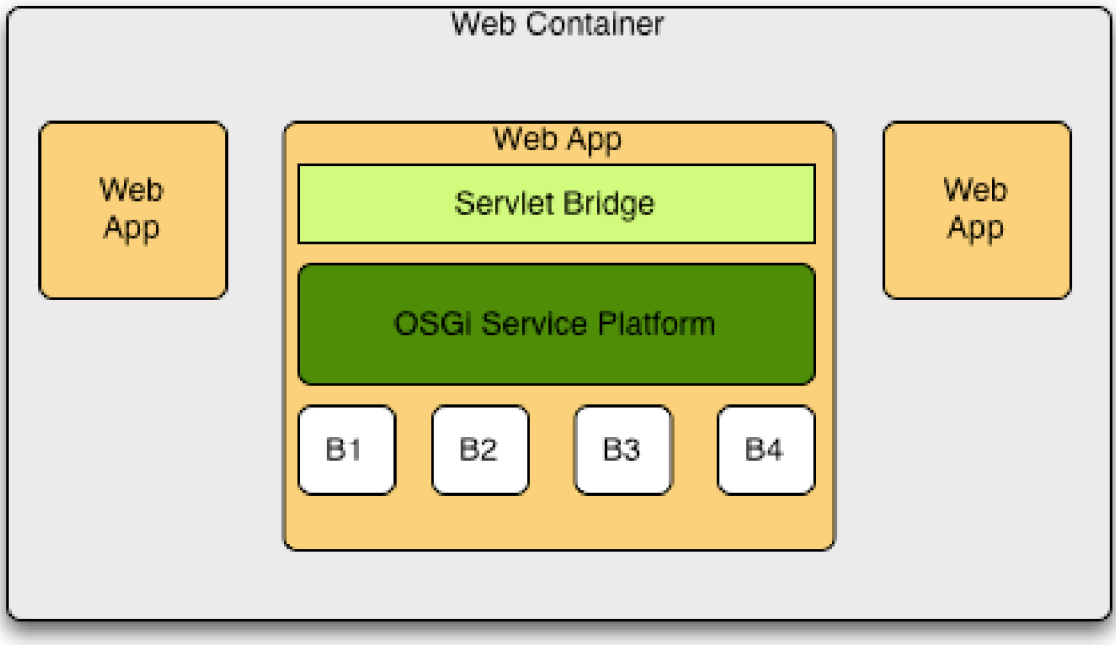
Server-side issues

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

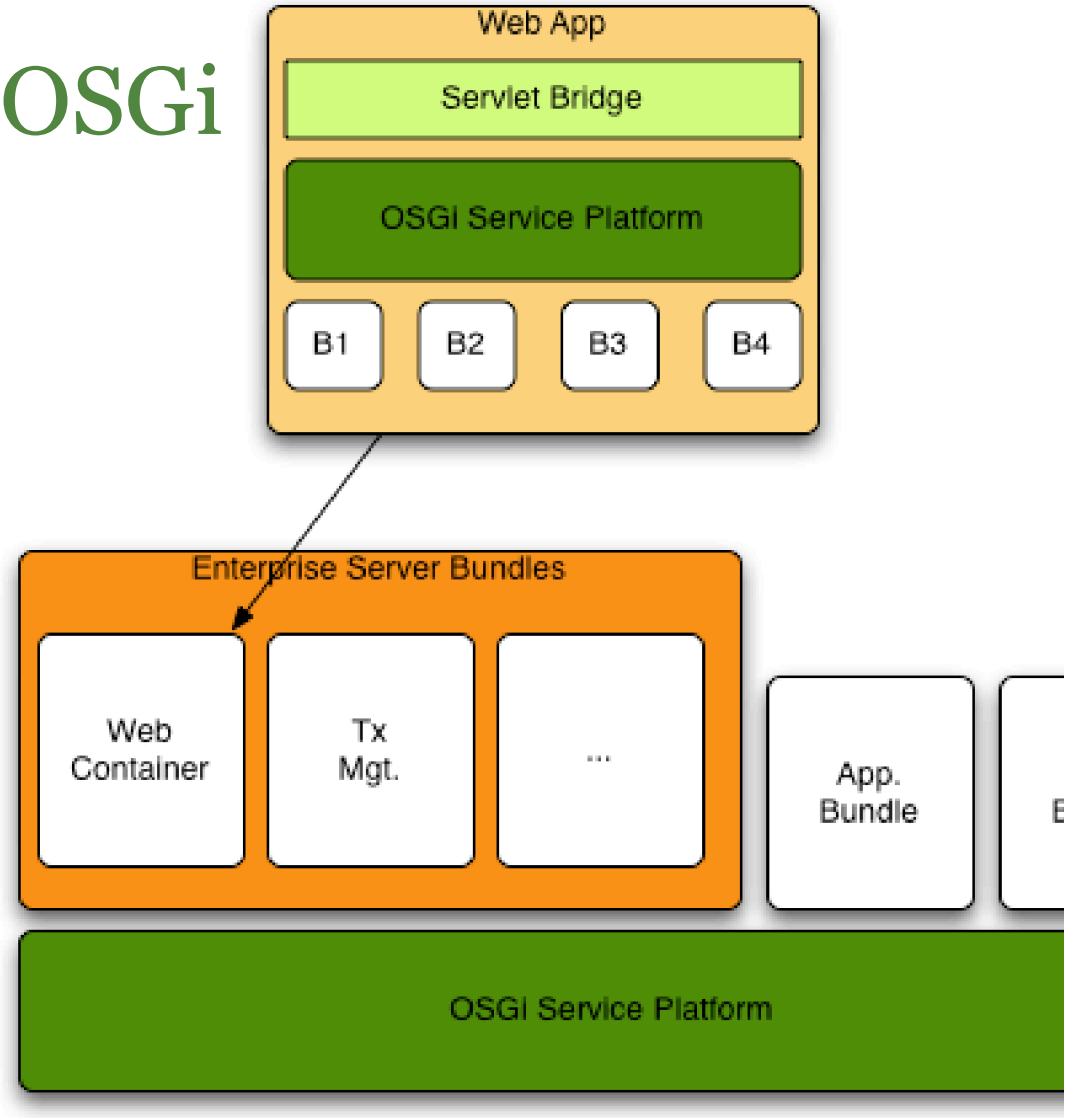
OSGi as a server platform



Embedded OSGi



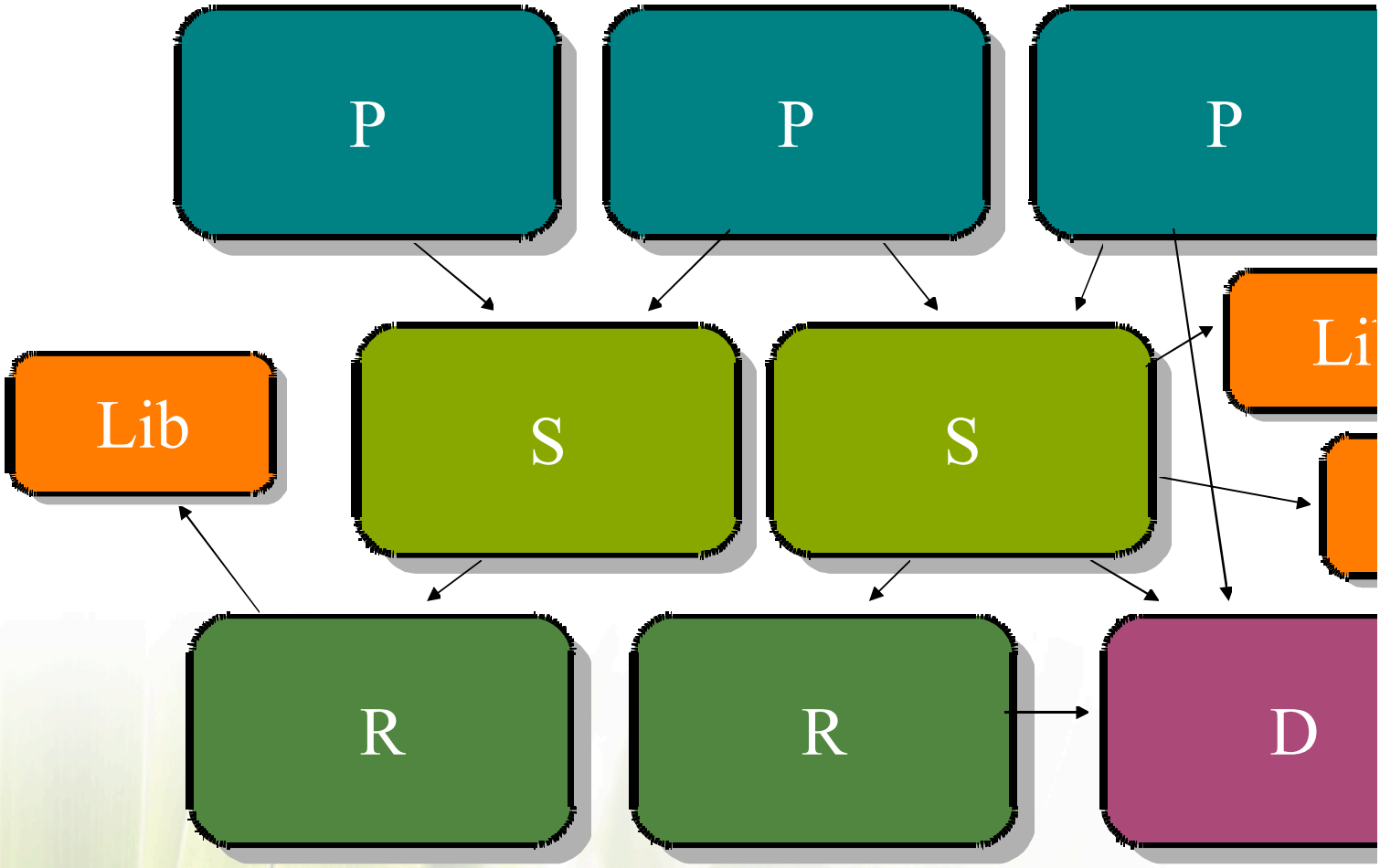
Nested OSGi



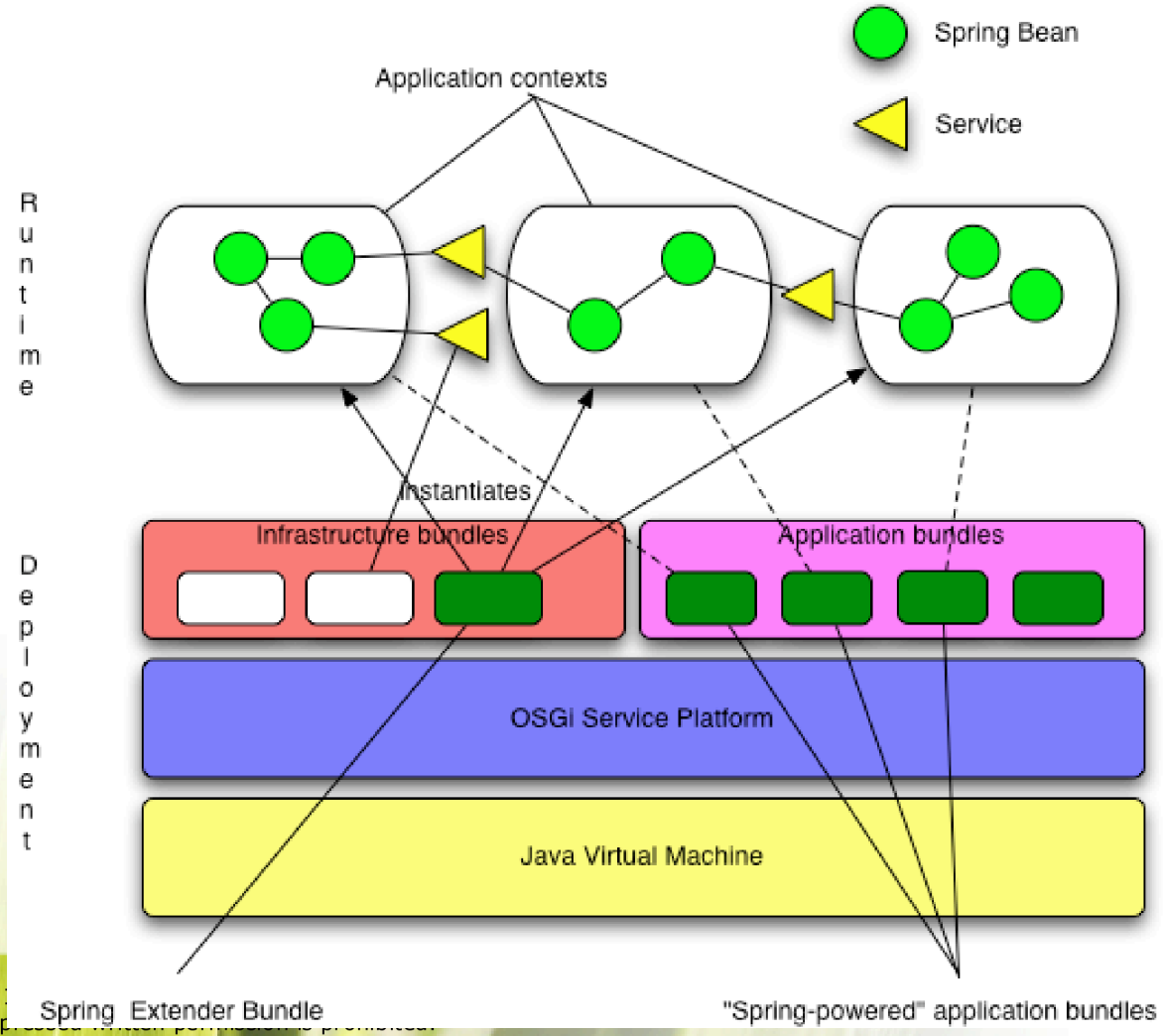
Application Design

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

Application wiring



Spring Dynamic Modules



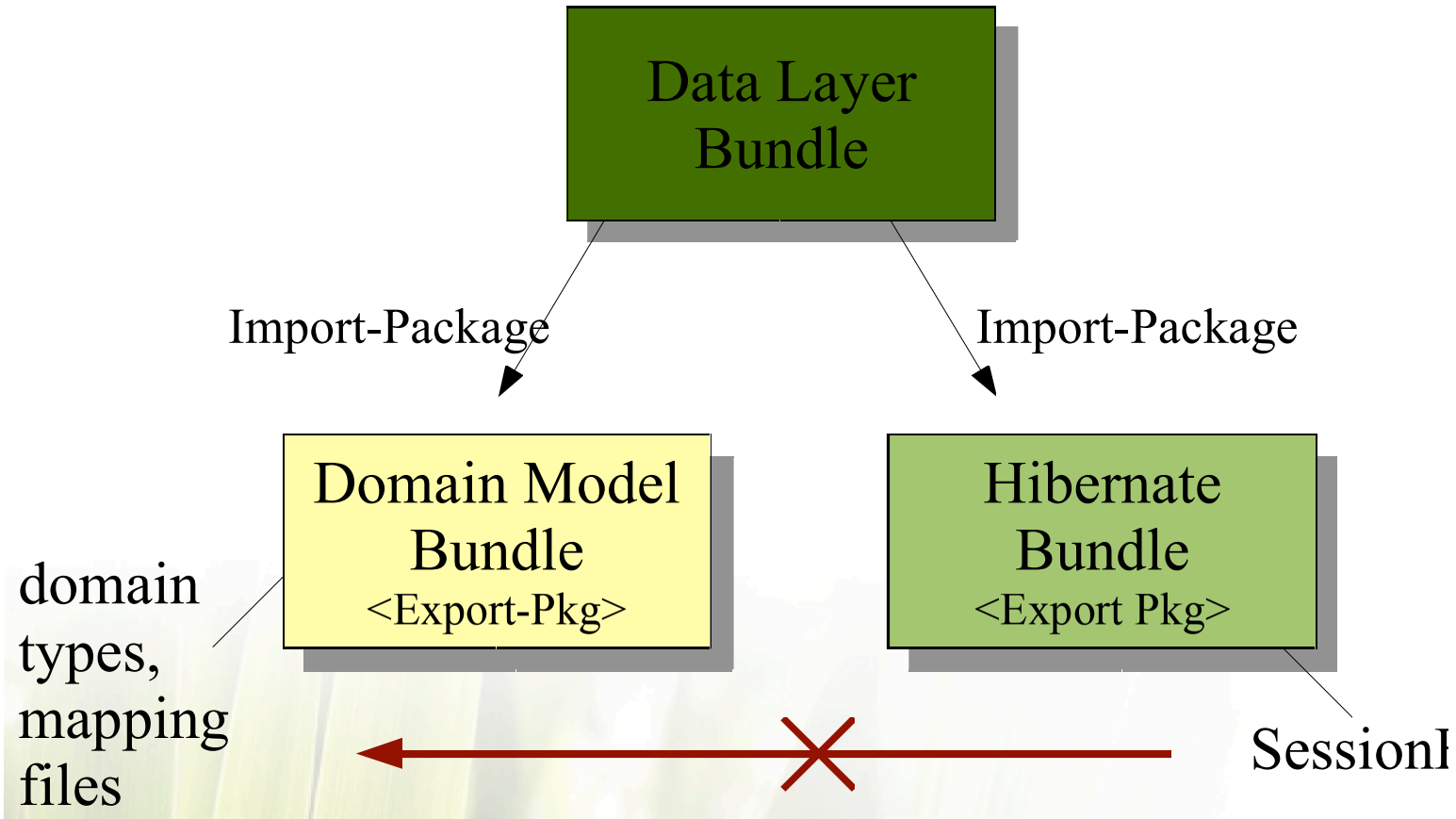
Design considerations

- Concurrency and thread management
- Asynchronous activation
  - service dependency management
- Platform dynamics
  - services may come and go at any time
  - ServiceTracker
- Testing

Enterprise Libraries under OSGi

- Code designed without OSGi in mind may ru class and resource-loading problems
  - class visibility
  - Class.forName
  - context class loader
- Spring 2.5 is OSGi-ready
  - modules shipped as bundles
  - all class loading behaves correctly under OSGi

Example: Class visibility

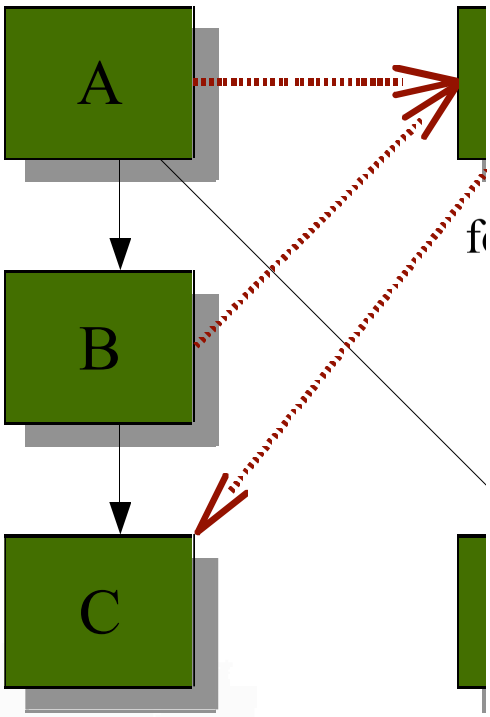


Class visibility solutions

- Dynamic-ImportPackage
  - a last resort
  - does not affect module resolution
  - too broad a scope
- Equinox Buddy Policy
  - In Hibernate bundle manifest:
    - Eclipse-BuddyPolicy : registered
  - In domain model bundle manifest:
    - Eclipse-RegisterBuddy : org.hibernate
    - Import-Package: org.hibernate

Class.forName

- Caches the returned class in the initiating class loader
  - native, vm-level cache
- Can cause class loading errors
- Prefer ClassLoader.loadClass



Context Class Loader

- Heavily used in enterprise Java
- Expected to have visibility of application typ classpath
- ContextClassLoader is undefined in OSGi!
  - No notion of “context”
  - No notion of “application”
- Solutions:
  - Eclipse Equinox: Context Finder
  - Spring Dynamic Modules : CCL management

Web Applications

- OSGi HttpService
  - registerServlets and resources under aliases
  - programmatic configuration
- Equinox Http Registry bundle
  - register servlets and resources using eclipse exte registry

Extension Registry

```
<plugin>

  <extension point="org.eclipse.equinox.http.registry.resources'
    <resource
      alias="/files"
      base-name="/web_files"/>
  </extension>

  <extension point="org.eclipse.equinox.http.registry.servlets">
    <servlet
      alias="/test"
      class="com.example.servlet.MyServlet"/>
  </extension>

</plugin>
```

Spring Dynamic Modules for OSGi Service Platforms

Enterprise applications on OSGi?

- OSGi offers an excellent foundation
- What if we want to build enterprise applicat on top of it?
- Need to exploit the power and sophisticatic OSGi
  - without adding complexity
  - retaining ability to use familiar enterprise librarie approaches
- Split application into a number of OSGi bun

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 s
- …and wire service references between bundle
- 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 back
  - new services are published, old service ren
  - broadcast operations
  - etc.

Preserve ability to test

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

Spring Dynamic Modules

- A new member of the Spring family
  www.springframework.org/osgi

- The simplicity and power of Spring…
- …with the dynamic module system of OSGi
- Heading towards 1.0 rc1

Project collaborators

- Led by Interface21
- Committers from BEA and Oracle also active on the
- Input to the specification and direction from
  - OSGi Alliance Enterprise Expert Gr
  - BEA, Oracle, IBM
  - Eclipse Equinox
  - Felix
  - and many individuals

OsgiBundleXmlApplicationConte

- A Spring application context based on an O bundle
- uses bundle context and classloader to loa resources
- implements Spring's resource abstraction f OSGi
  - relative resource paths resolved to bundle entrie
  - "bundle:" prefix for explicit specification

Creating a bundle application con

- Possible to create a bundle application cont programmatically…
- …but you don't normally need to

- Deploy the org.sfw.osgi.extender bundle
  - acts like "ContextLoaderListener"
  - automatically creates Spring application context a bundle when a bundle is started
    - no code or dependence on Spring APIs required!

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/MANIFES
  - Bundle-SymbolicName: org.xyz.myapp.mymodu
  - Bundle-Version: 1.0
  - Bundle-ManifestVersion : 2
- Place configuration files in META-INF/sprin

Exporting a Service

```
<osgi:service id="simpleServiceOsgi"
  ref="simpleService"
  interface=
     "org.sfw.osgi.samples.ss.MyService"/>
```

Importing a Service

```
<osgi:reference id="aService"
  interface=
     "org.sfw.osgi.samples.ss.MyService"/>
```

- optional "filter" attribute

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?

Other Spring DM features

- Context Class Loader management
- Configuration Admin service integration
- Integration testing support
- Bundle lifecycle management
- and more!

- See the reference guide online for full detail

Summary

- OSGi is a dynamic module system for Java
  - proven
  - scalable (up and down)
- Offers benefits in terms of
  - modularity
  - versioning
  - operational control
- Programming model needs simplifying
  - Spring Dynamic Modules combines the simplicity and of Spring with the sophistication of the OSGi platform