Spring Framework

Enhancing Productivity, Powering Production

Nik Trevallyn-Jones

SpringSource

Folientitel

- Font: Microsoft Sans Serif, 32 pt
  - Font: Microsoft Sans Serif, 28 pt
    - Font: Microsoft Sans Serif, 24 pt
      - Font: Microsoft Sans Serif, 20 pt

- Lorem ipsum dolor sit amet, consectetur adipscing elit, sed diam nonnumy eiusmod tempor incidunt ut labore et dolore magna aliquam erat volupat.

Why Spring?

- Fix commonly seen problems in Enterprise Application development
  - Boilerplate code
  - Leaky abstractions
  - Code intrusion

Common Problems

- Boilerplate code
  - Same code appears in multiple places
  - Re-implemented for each project
  - Inconsistent; error-prone; maintenance nightmare
  - Egs: JDBC code, JMS code, etc.

Common Problems

- Leaky Abstractions
  - Technology-specific exceptions
  - Checked exceptions being thrown to the wrong layer
  - Egs: JDBC, JMS, RMI

Common Problems

- Code intrusion
  - Object in one layer “seeks” other objects it depends on
    - These dependencies may be in other layers
  - Using inheritance for system services
  - Egs: static find() methods; EJB; JavaBeans, RMI

So What?

- Reduced benefits of Object-Orientation
  - Code reuse
    - Primary benefit of inheritance
  - Code cohesion
    - Primary benefit of encapsulation
  - Modularity
    - Primary benefit of encapsulation
  - Deployment independence
    - Major benefit of polymorphism

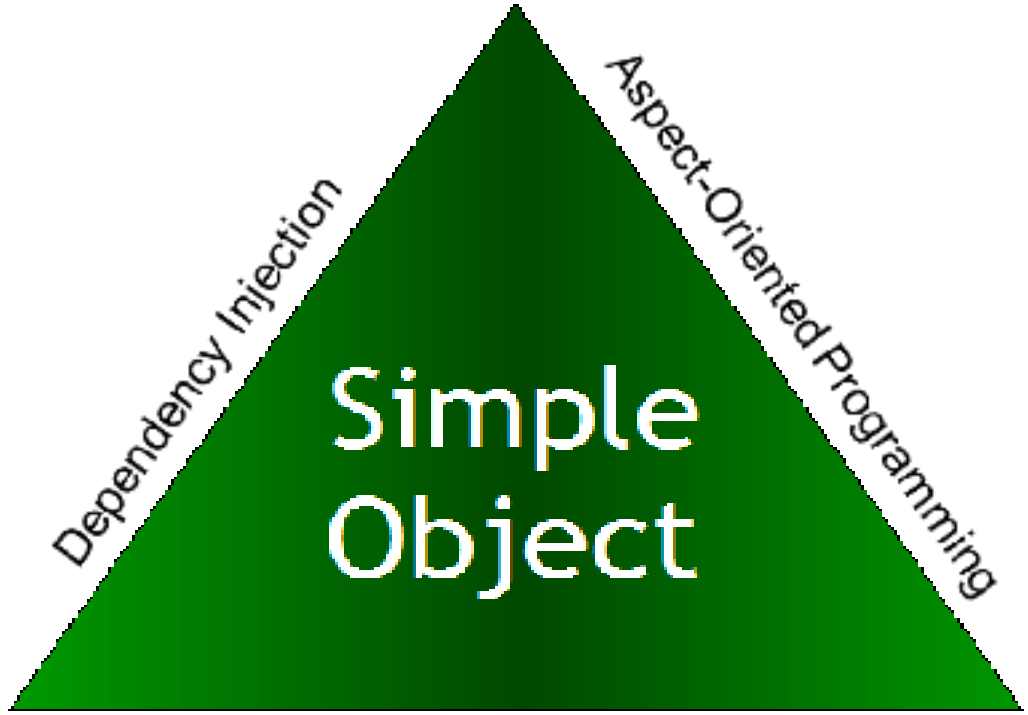
Underlying Causes

- Services lacking a consolidated interface
  - Programmer must call multiple methods on multiple objects
- Misuse of O-O features
  - Inheritance for system behaviour
  - Checked exceptions in infrastructure layers
- O-O has no tool to solve the problem
  - Eg cross-cutting concerns

From this, came Spring…

- Eliminate boilerplate code
  - Don’t Repeat Yourself
- Avoid code intrusion
  - Separation of Concerns
- DRY SOCS
  - Helping programmers feel warm and fuzzy…

The Spring Triangle



Enterprise Service Abstractions

Enhancing Productivity

- The evolution of Spring to version 2.5
  - Support for new features of Java
  - Further simplification of use
  - Reduced artifacts from Spring

A few observations ...

- Spring is one of the leading Java frameworks in use today
- Many find the XML configuration verbose or tedious
- Spring and Tomcat is our leading production platform

What we did ...

- Made Spring easier to use
  - Annotations
- Made it more powerful in production
  - New platforms
  - New tools
  - OSGi

Why?

- Less time configuring Spring
- More time writing your code – the interesting stuff!
- Easier to deploy production applications

More power, less complexity

Agenda

- Configuration
  - **Annotations**
- Production
  - Supported platforms
  - Application Monitoring
  - OSGi

Configuration

From JSR-250 to @Controller

Annotation-driven configuration

Spring 2.5 embraces annotations:

- JSR-250 Common Annotations
- Spring Auto-wiring Annotations
- Auto-detection of Components
- MVC Annotations
- Testing

Annotation-driven configuration

Additional configuration option

- XML is *in no way* deprecated!
- You can mix and match
- Different from Spring JavaConfig

JSR-250 Common Annotations

- Part of Java EE 5 and JDK 1.6
  - Available as extra jar for JDK 1.5
- Annotations for lifecycle and DI
  - `@PostConstruct` & `@PreDestroy`
    - equiv. to init- and destroy-methods
    - equiv. to InitializingBean, FinalizingBean
  - `@Resource`
    - Injection of named beans or JNDI resources
    - Can also be used on fields
- Fully supported by Spring 2.5!

Spring Configuration Example

Before

```
public class MyService implements MyServiceInterface
{
  private DataSource dataSource;
  private Processor processor;
  public void setDataSource(DataSource dataSource) { … }
  public void setProcessor(Processor processor) { … }
  public void initialize() { … }
  public void shutdown() { … }
}
<!-- Declare and initialize the beans -->
<bean id="myService" class="mypackage.MyService"
      init-method="initialize" destroy-method="shutdown">
   <property name="dataSource" ref="dataSource"/>
   <property name="processor" ref="processor"/>
</bean>
```

JSR-250 Annotations Example

After

```
public class MyService implements MyServiceInterface
{
  @Resource
  private DataSource dataSource;
  private Processor processor;
  @Resource(name="myProcessor")
  public void setProcessor(Processor processor) { … }
  @PostConstruct
  public void initialize() { … }
  @PreDestroy
  public void shutdown() { … }
}
```

```
<!-- Just declare the beans - no other config is necessary! -->
<bean id="myService" class="mypackage.MyService" />
<!-- Turn annotations on - off by default -->
<context:annotation-config/>
```

Annotation-driven configuration

- JSR-250 Common Annotations
- **Spring Autowiring Annotations**
- Component scanning
- MVC Annotations

Autowiring Annotation

- New `@Autowired` annotation
  - Autowiring by type
  - Of fields, methods and constructors
- Sweet spot for autowiring:
  - *By name* often too simplistic
  - *By type* often too extensive
  - **Specific** by type works a lot better!

Autowiring Annotation Example

```
public class MyService1 implements MyServiceInterface {
  @Autowired
  private DataSource dataSource;
  @Autowired
  public void injectServices(ServiceA a, ServiceB b) { … }
}
public class MyService2 implements MyServiceInterface {
  ...
  @Autowired
  public MyService(DataSource dataSource, ServiceA a) { … }
}
<!-- No specific configuration necessary: driven by annotations -->
<bean id="myService" class="mypackage.MyService1" />
<!-- Turn annotations on - off by default -->
<context:annotation-config/>
```

More Autowiring

- Autowiring of typed Collections:

```
@Autowired
private List<MovieCatalog> allCatalogs;
```

- Replacing ***Aware** interfaces:

```
@Autowired
private MessageSource messageSource;

@Autowired
private ResourceLoader resourceLoader;

@Autowired
private ApplicationContext applicationContext;
```

The context XML Namespace

New configuration namespace: **`context`**

- **<context:property-placeholder location="…">**
  - invokes PropertyPlaceholderConfigurer
- **<context:annotation-config>**
  - activating JSR-250, all common autowiring annotations, Spring 2.0's **`@Required`**
- **<context:mbean-export>**
  - activating annotation-driven MBean export (**`@ManagedResource`**, **`@ManagedOperation`**)
- *We'll see more later on*

Should you do it?

- Pros:
  - Self-contained: no XML config needed
  - Works in much more cases than generic autowiring (any m method or field)
  - JSR-250 or custom annotations keep your code from depending on Spring
- Cons:
  - Requires classes to be annotated
  - Configuration only per class, not per instance
  - Changes require recompilation

Bottom Line: It's your choice, pick one!

When should I do it?

- Annotations
  - When annotation binds tightly to code purpose
    - Testing
    - Web-layer artifacts (Controller, Endpoint)
    - Transactions
  - Looser binding makes a weaker case
    - Domain POJOs should be reusable
    - But remember: annotations are *passive*

Annotation-driven configuration

- JSR-250 Common Annotations
- Spring Autowiring Annotations
- **Component scanning**
- MVC Annotations

Annotated Components

- Annotate a type to make it a Spring bean
  - with **@Component**
  - or annotation which is itself marked @Component
    - Spring has @Repository, @Service and @Controller stereotypes
    - You can create your own
- No <bean> tag needed anymore!
- Enabled with <context:component-scan>
  - Scans the classpath for beans

Annotated Component Example

```
package mypackage.services;
@Service
public class MyService implements MyServiceInterface {
  @@Resource(name="myDataSource")
   private DataSource dataSource;
   @Autowired
   public void injectServices
        (ServiceA a, ServiceB b) { … }
   @PostConstruct
   public void initialize() { … }
   @PreDestroy
   public void shutdown() { … }
}
<!-- Not even a plain XML <bean> tag is necessary! -->
<context:component-scan
     base-package="mypackage.services" />
```

Spring automatically:

- *Creates an instance*
- *Sets fields marked as Resource or Autowired*
- *Calls Autowired and PreConstruct methods*

Bean Name and Scope

- Provide bean name using value element:
  - @Component("service")
    - Defaults to uncapitalized short class name:
      `mypackage.services.MyService`
      => *myService*

- Use @Scope to define bean scope

```
package mypackage.services;

@Component("service")
@Scope("session")
public class MyService implements MyServiceInterface
{
    ...
}
```

Component Scanning Pros

- No need for XML unless you need the greater sophistication it allows
- Changes are picked up automatically
- Works great with Annotation Driven Injection
  - picking up further dependencies with **`@Autowired`**
- Highly configurable

Component Scanning Cons

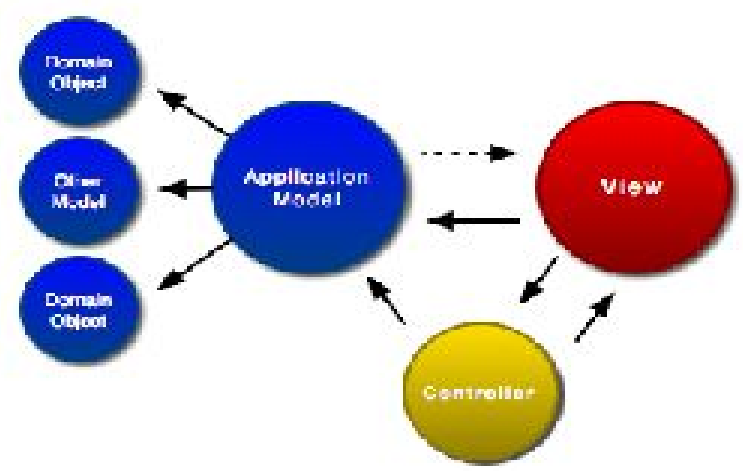
- Not a 100% solution
- Requires classes to be annotated
- Don't scan a huge number of classes!
  - use Spring’s filtering mechanism

Annotation-driven configuration

- JSR-250 Common Annotations
- Spring Autowiring Annotations
- Component scanning
- **MVC Annotations**

@Controller

@RequestMethod



Annotation-driven Controllers

- Java5 variant of MultiActionController
  - Including form handling capabilities
- POJO-based
  - Just annotate your class
  - Works in servlet and portlet container
- Several annotations:
  - @Controller
  - @RequestMapping
  - @RequestMethod
  - @RequestParam
  - @ModelAttribute
  - @SessionAttributes
  - @InitBinder

Example of Annotated Controller

```
@Controller
@RequestMapping("/order/*")
public class OrderController {
  @Autowired
  private OrderService orderService;
  @RequestMapping("/print.*")
  public void printOrder(HttpServletRequest request, OutputStream responseStream) {
    // write directly to the OutputStream:
    orderService.generatePdf(responseStream);
  }
  @RequestMapping("/display.*")
  public String displayOrder(@RequestParam("id") int orderId, Model model) {
    model.addAttribute(...);
    return "displayOrder";
  }
}
```

Annotated Controller from PetClinic (1)

- Session-based form setup:

```
@Controller
@RequestMapping("/editPet.do")
@SessionAttributes("pet")
public class EditPetForm {
  // ...
  @RequestMapping(method = RequestMethod.GET)
  public String setupForm(
    @RequestParam("petId") int petId, ModelMap model)
  {
    Pet pet = this.clinic.loadPet(petId);
    model.addAttribute("pet", pet);
    return "petForm";
  }
```

Annotated Controller from PetClinic (2)

- Session-based form processing:

```
@ModelAttribute("types")
  public Collection<PetType> populatePetTypes() {
    return this.clinic.getPetTypes();
  }
  @RequestMapping(method = RequestMethod.POST)
  public String processSubmit(@ModelAttribute("pet") Pet pet,
   BindingResult result, SessionStatus status)
  {
    new PetValidator().validate(pet, result);
    if (result.hasErrors()) { return "petForm"; }
    else {
      this.clinic.storePet(pet);
      status.setComplete();
      return "redirect:owner.do?ownerId=" + pet.getOwner().getId();
    }
  }
```

Spring Test Context Framework

- Revised, annotation-based test framework
  - Convention over configuration
- Supports JUnit 4.4, TestNG as well as JUnit 3.8
- Supersedes older JUnit 3.8 base classes
  - AbstractDependencyInjection-SpringContextTests & friends
  - They're still there for JDK 1.4

Annotated Test Class Example

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("test-config.xml")
// or default to MyTests-context.xml in same package
public class MyTests {
  @Autowired
  private MyService myService;
  @Test
  public void mySimpleTest() {...}
  @Test
  @Transactional
  public void myTransactionalTest() { ... }
}
```

Test Context Annotations

- TestExecutionListeners
  - `@TestExecutionListeners`
- Application Contexts
  - `@ContextConfiguration` and `@DirtiesContext`
- Dependency Injection
  - `@Autowired`, `@Qualifier`, `@Resource`, `@Required`, etc.
- Transactions
  - `@Transactional`, `@NotTransactional`, `@TransactionConfiguration`, `@Rollback`, `@BeforeTransaction`, and `@AfterTransaction`
- Testing Profiles (JUnit only)
  - `@IfProfileValue` and `@ProfileValueSourceConfiguration`
- JUnit extensions
  - `@ExpectedException`, `@Timed`, `@Repeat`

Production

Enhancing deployment

Support for new Platforms

New Platform support:

- JJava 6 (JDK 1.6)
- Java EE 5
- OSGi

Java 6 Support

- New JDK 1.6 API's supported:
  - JDBC 4.0
  - JMX MXBeans
  - JDK ServiceLoader API
- JDK 1.4 and 5 still fully supported
  - Last Spring release compatible with 1.4
  - Even JDK 5 (1.5) is end-of-life in 2009
  - JDK 1.3 no longer supported
    - Declared end-of-life by Sun a year ago

Note: Extended Spring support via *Subscriptions*

Java EE 5 support

- Support for Java EE 5
  - Integrates seamlessly
- New Java EE 5 API's supported:
  - Servlet 2.5, JSP 2.1 & JSF 1.2
  - JTA 1.1, JAX-WS 2.0 & JavaMail 1.4
- 2.5 officially supported on IBM WAS 6.x
  - Custom WebSphere transaction manager
- J2EE 1.4 still fully supported
  - For BEA WebLogic and IBM WebSphere releases still using 1.4

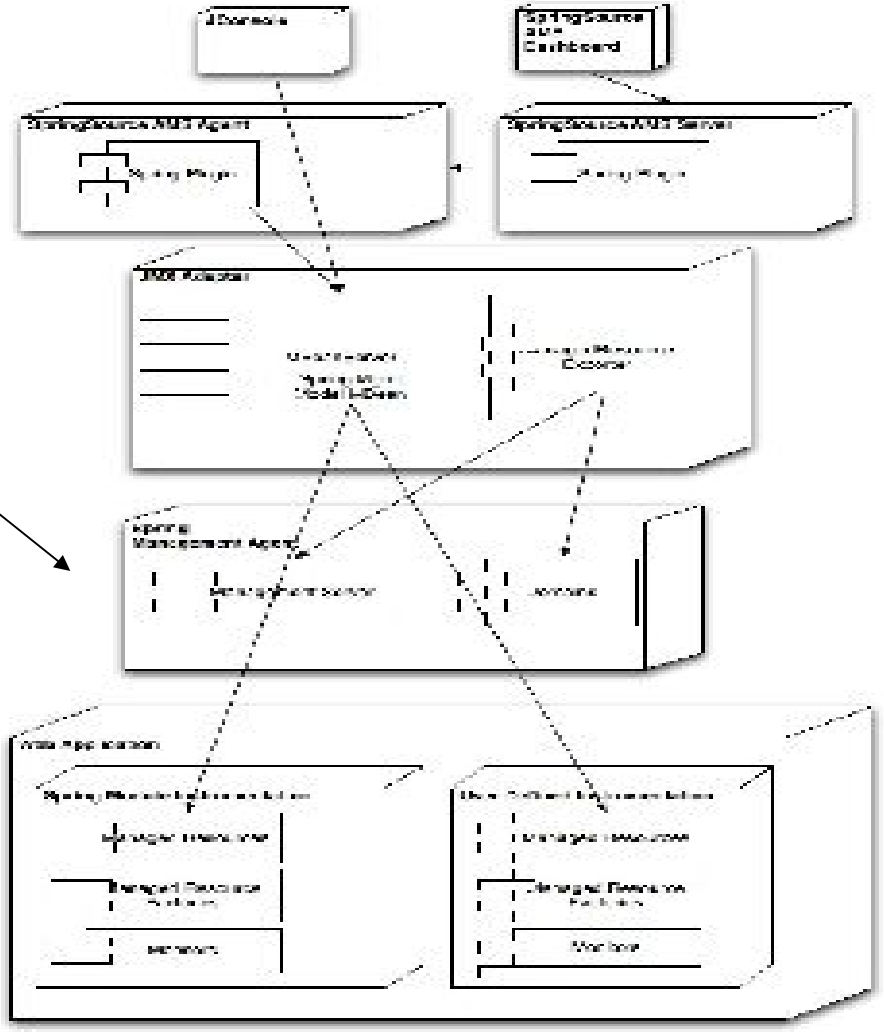
Application Monitoring

Spring Application Management Suite (AMS)

- Monitors applications running on Spring at runtime
  - JMX based
  - Metrics on bean usage, performance
  - Real-time visibility of Spring components
- Plus common application components
  - Operating system, hardware ...
  - Databases, web/JEE container ...

Architecture

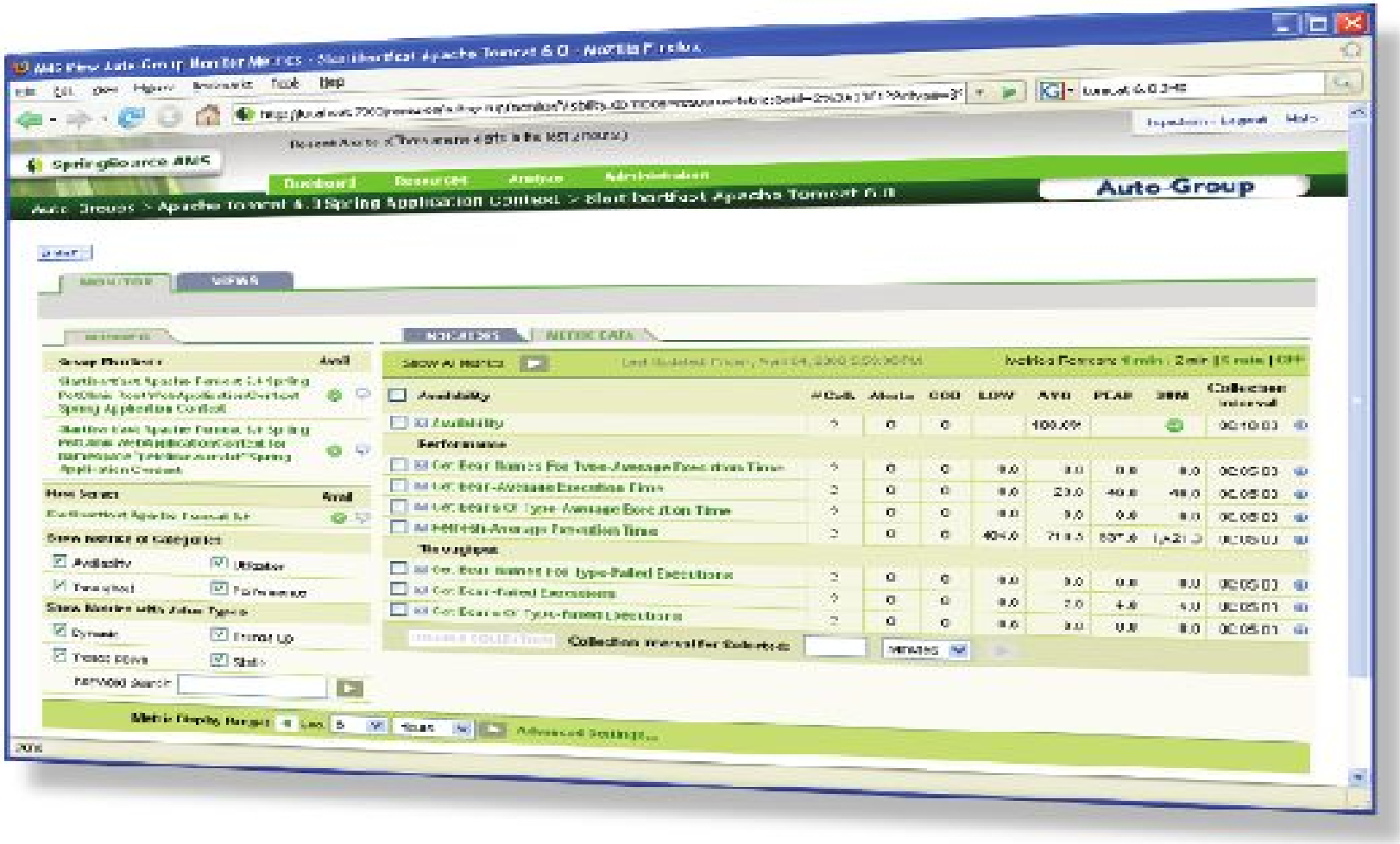
- Instrumented Application
  - JMX based
- Agent to collect metrics
- Server to aggregate and report
- Web-based Dashboard



Essential for ...

- Pre-deployment
  - Performance testing, resource usage
- Operations management
  - Alarms
  - Monitor application performance
  - Reduce downtimes
- ITIL service-delivery
  - enforce service level agreements
  - analyse trends over time

Sample Dashboard



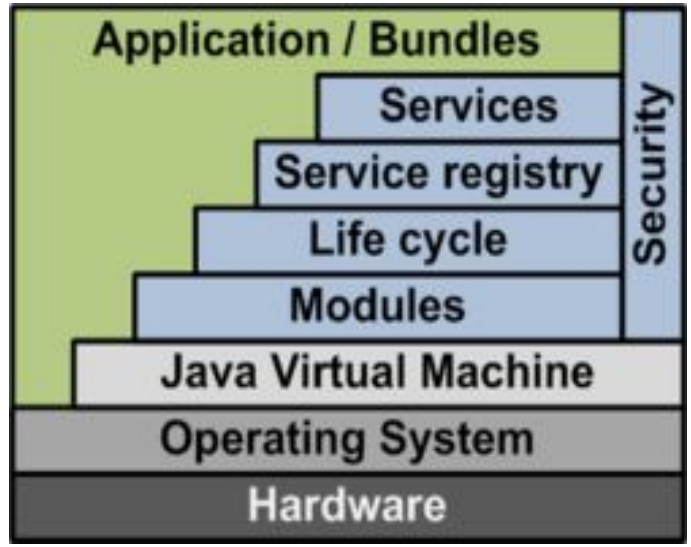
OSGi Support

- **Who knows what OSGi is?**
- **Who is using it?**

- **You are if you use Eclipse**
  - The *Equinox* framework u
    Eclipse plug-ins

What is OSGi?

- Open Services Gateway initiative
- Dynamic module system
- *Bundle* as central packaging unit
  - Versioned jar
  - Exports types to expose
  - Started, stopped and updated *at runtime*



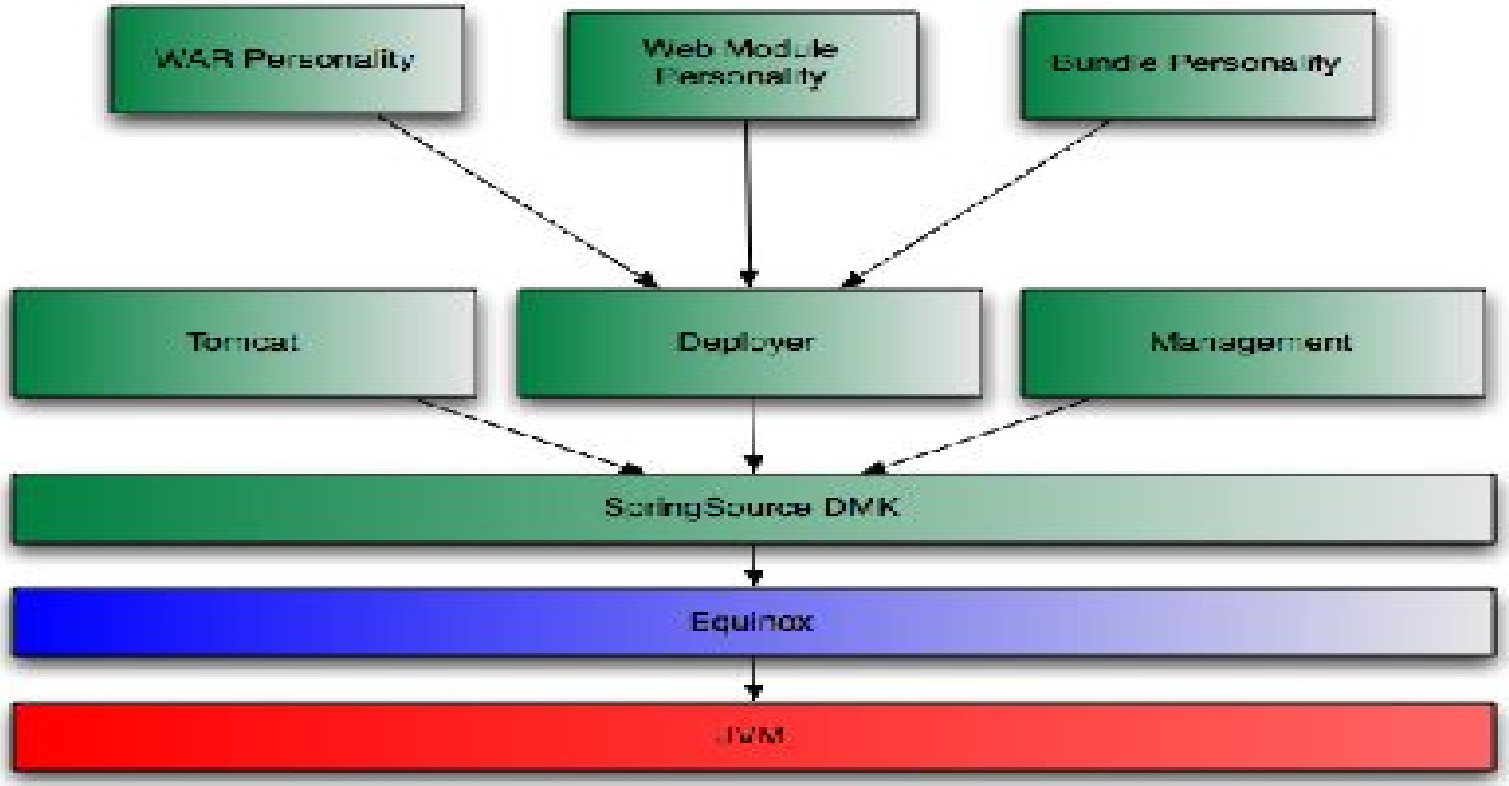
Spring & OSGi

- Integration with OSGi provided by the ***Spring Dynamic Modules for OSGi™ Service Platforms*** project
  - Now on version 1.0.2 (at April 2008)
- 2.5 jars now are compliant bundles
  - Headers in MANIFEST.MF
- **`ApplicationContext`** per bundle
- Integration with OSGi service registry

Spring Source Appl^{n} Platform

- An OSGi Container
  - Spring and Tomcat as out-of-the-box OSGi bundles
  - Can deploy an existing WAR
  - Run different versions of same libraries in different plug-ins (Maven-like)
  - Incremental deployment/upgrade
  - Hot patching of bundles whilst server is still running
  - Free to download and develop

SSAP Architecture



Deployment Options

- Three ways to deploy
  - *Standard WAR file*
    - Will deploy straight inside an OSGi enabled Tomcat bundle.
  - *Web Module*
    - Similar to WAR, but *without* third-party jars. Necessary jars found and included during deployment.
  - Bundle
    - Best option. Allows full use of deployment repository to resolve jar dependencies.

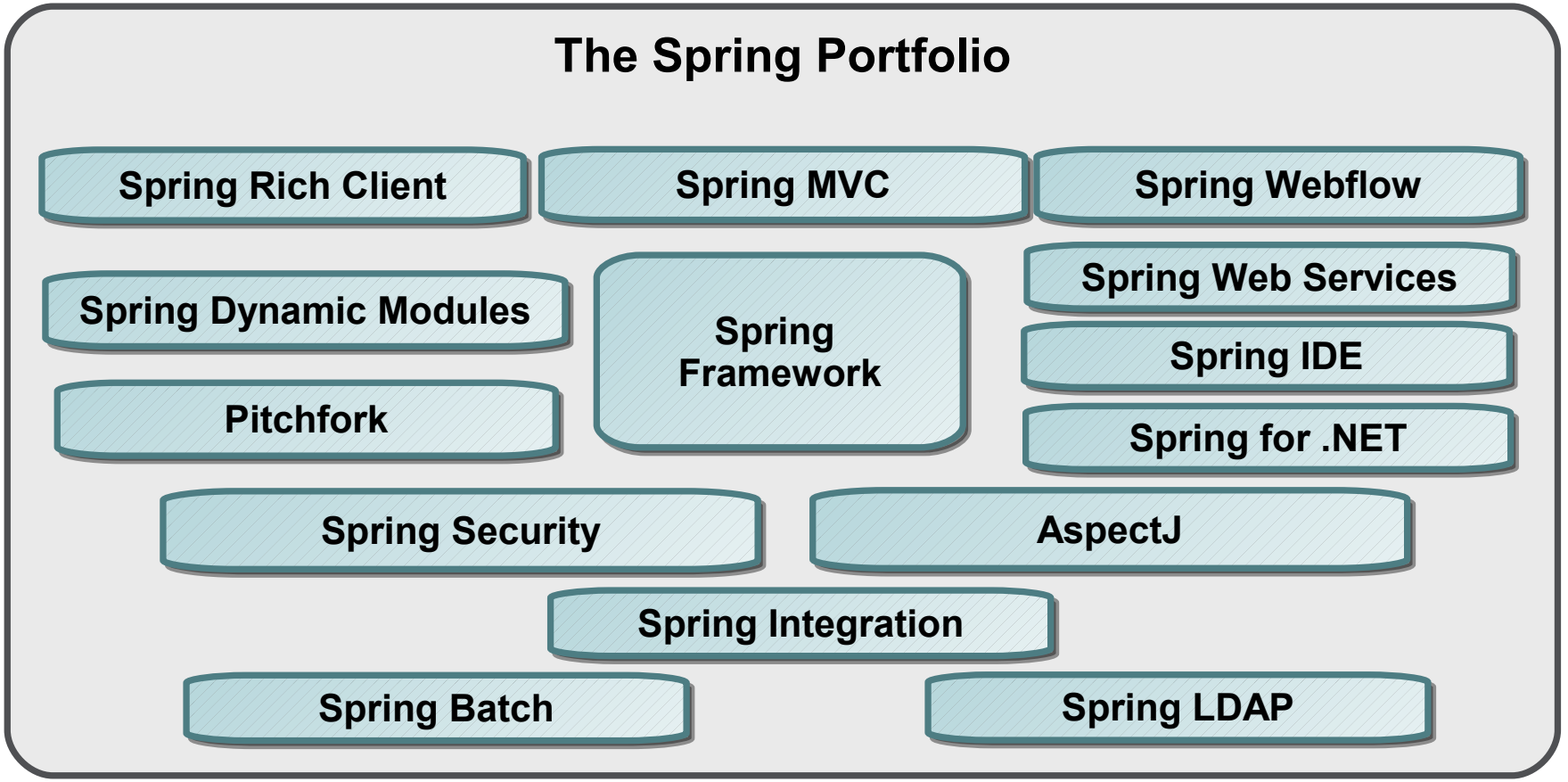
Spring on OSGi vs. Spring on Java EE

- Similar programming models
  - No dependencies on environment: just POJO's
  - Services are Spring managed components
- Very different deployment environment
- Currently alternative runtimes
  - Hard to mix
  - Might change in the future (Glassfish, Jetty)
  - Spring provides the common ground

Is Spring Still Open-Source?

- Absolutely, and emphatically YES!
  - Everything that was ever free stays free
  - *Always*
- In addition ...
  - Extra products to facilitate deployment
    - Aimed primarily at large customers
    - Want certainty and support
    - Enhanced deployment capabilities
  - Hence *subscriptions*

Spring Open-Source Portfolio



Summary

- Spring 2.5 builds on strong 2.0 foundation
  - With updated support for standards
  - And focus on ease of configuration
- Embraces Java 6, Java EE 5 as well as OSGi
  - Spring as 'traditional' Java EE framework
  - Or as application framework in OSGi environment
  - Same programming model!
- Embraces annotations for configuration
  - As an addition to existing alternatives
- Enhances XML configuration

Want to learn more?

- Spring Training
  - Not yet available in India - we are working on it
  - Currently available in Australia, Hong Kong, Singapore
- SpringOne America
  - December 1st, Florida, USA
- Spring Exchange
  - January 29th, 2009, London, England

Want to learn more?

- Certification
  - Available and recognised worldwide
  - Register and take on-line exam
  - One exam included in every Core Spring course place
  - “Grandfathering” for experienced Spring developers

Want to learn more?

- Spring Reference Manual
  - Updated sample applications
  - *www.springframework.org*
- Other online resources
  - Spring Forums
  - *blog.springsource.com*
  - *www.springsource.com*

Ask me a Question!

Nik Trevallyn-Jones
Consultant

Q & A

Nik Trevallyn-Jones
Consultant