ENTERPRISE DEVELOPMENT SERVICES DIRECT FROM THE CREATORS OF SPRING FRAMEWORK

INTERFACE21

What's New and Cool in Spring 2.0

Rod Johnson

CEO, Interface21

www.interface21.com

Spring

Goals of This Talk

Understand the major enhancements in Spring 2.0, the latest generation of the most popular application programming framework for the Java™/J2EE™ platforms

Agenda

- The story so far
- Goals of Spring 2.0
- Feature overview
- Extensible XML configuration
- AOP enhancements and AspectJ integration

Aims of Spring

- Starting goal of Spring (from 2002) was to help to reduce the complexity of J2EE™ based development
  - To simplify without sacrificing power
  - To facilitate best practices that were otherwise difficult to follow
  - Grew from practical experience of myself and other practising architects/developers of J2EE based applications
- *Simple things should be simple and complex things should be possible* — Alan Kay
- *Unless simple things are simple, complex things are impossible*

Technical Aims of Spring

- Enable applications to be coded from POJOs
  - Offer sophisticated configuration capabilities that scale to real-world complexity
  - Allow enterprise services to be applied to those POJOs in a declarative, non-invasive way
- Examples
  - POJOs can be made transactional without the need to know about transaction APIs
  - POJOs can be exposed for management via JMX without the need to implement an MBean interface
  - …

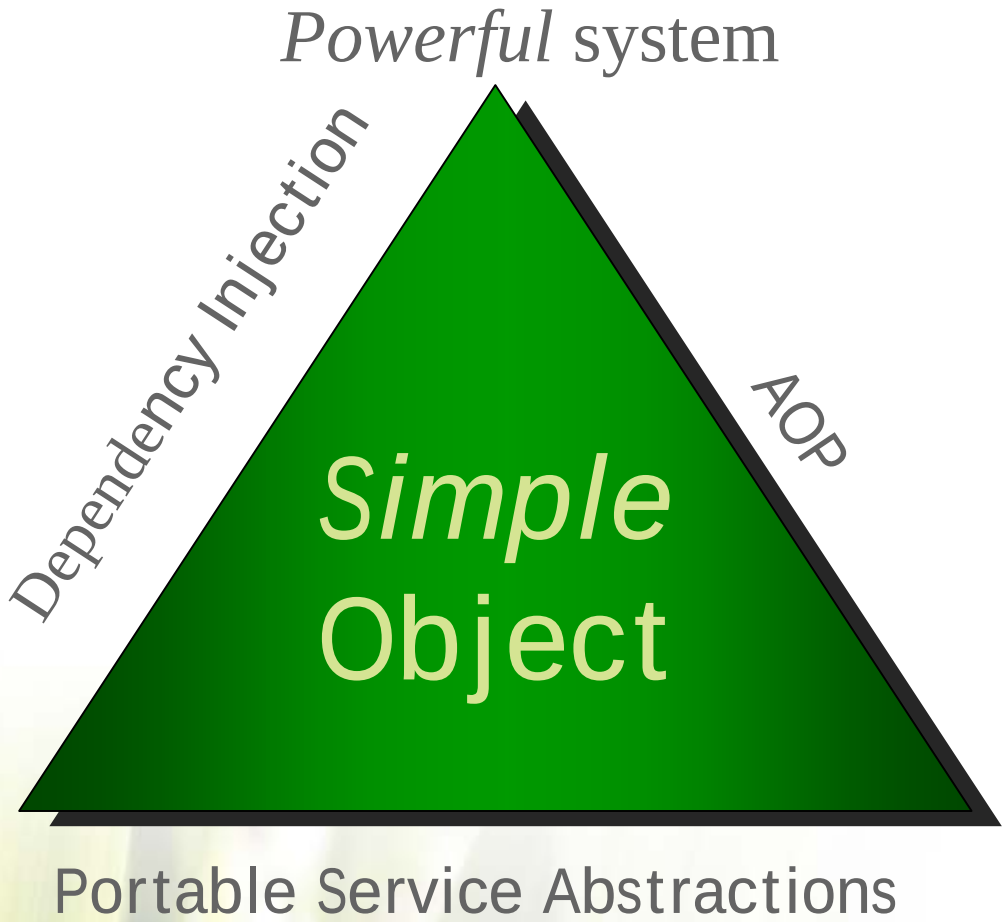
POJO Development

- **POJO** stands for **Plain Old Java based Object**
- A POJO is not bound to any environment
  - No imports of classes that bind it to a specific environment
  - Not dependent on a particular lookup mechanism
    - Collaborating instances are injected using plain Java constructors or setter methods
  - Prolongs the life of business logic by decoupling it from volatile infrastructure
- **True POJOs are testable in isolation**

Applying Services to POJOs Declaratively

- Decouples your application objects from their environment
  - Brings leverage, enables reuse
- Actually more powerful than traditional invasive component models
  - Lets you scale up or down without rewriting application code
- Examples
  - Switch to global transactions over Java TA
  - Export your business objects in different environments
    - Switch between SLSB, web service, write/take from JavaSpaces™ technology etc.

Enabling Technologies



Spring in Practice

- Solidity of core Spring abstractions has seen Spring demonstrate value in a wide range of environments beyond J2EE™ technology
- Strategic adoption in many enterprises moving away from traditional costly, inefficient J2EE technology approaches
- Used as the basis for server technology and further products from several vendors, including…
  - BEA WebLogic 10, WebLogic RealTime Server
  - Oracle Fusion middleware products

What's in it for you?

- Proven solutions for common problems
- Makes it easy to follow best practice
- Increases potential for reuse
  - Decouples business logic from underlying platfom
- Simple, consistent programming model that scales to handle real-world complexity and runs in any environment
- Facilitates testability
  - Round trips become much quicker during development

Who Uses Spring?

- **Over 1.1 million downloads and gaining further momentum**
- Extensive and growing usage across many industries, including…
- Retail and investment banks
  - Most of the worlds top 10 banks are Spring users and Interface21 customers
- Insurance companies (US and Europe)
- Government
  - European Patent Office
  - French Online Taxation System
  - US, Canadian and Australian Government Agencies

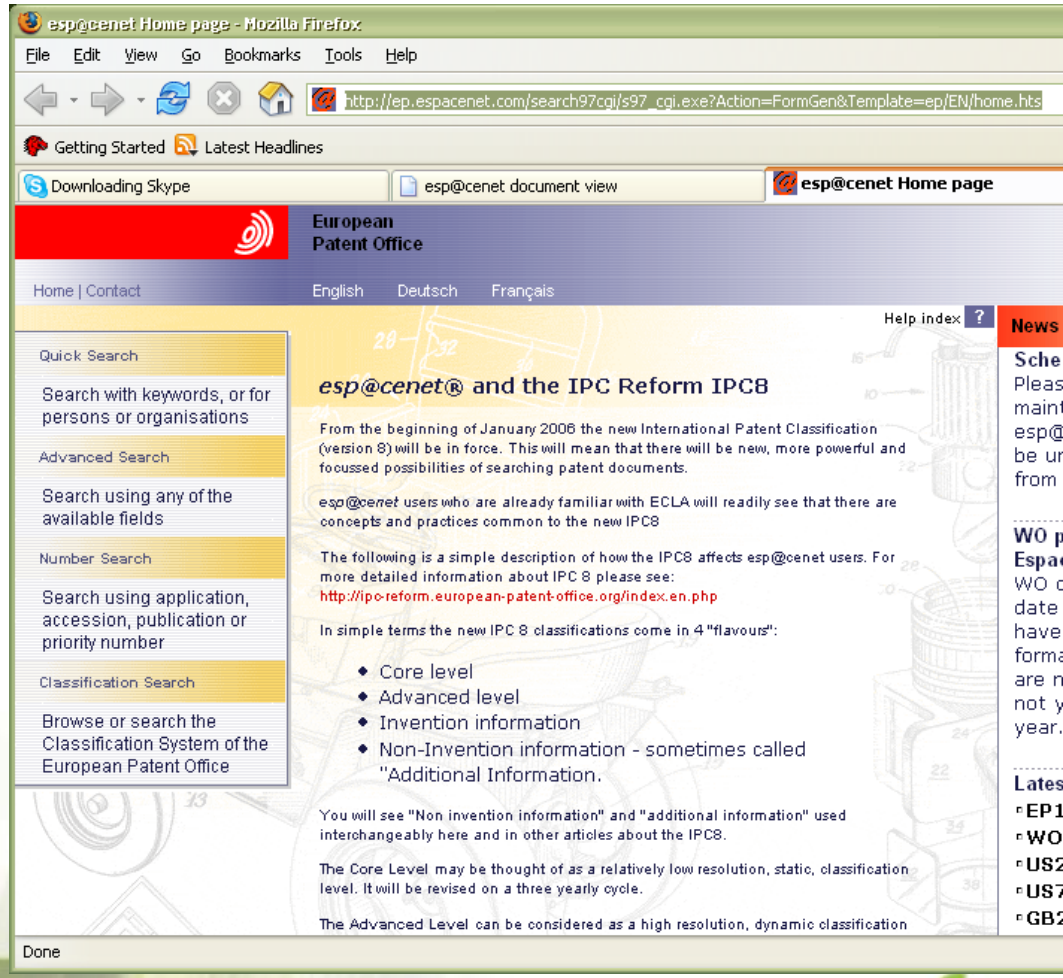
Who Uses Spring?

- Scientific Research
  - CERN
- Airline industry
- Defence
- Media and online businesses
  - eBay
  - And many others…
- Software vendors
  - Including Oracle and BEA

A User's Perspective

Ever since the introduction of Spring I have been able to focus on what really matters: the business focus of an enterprise application. "Low-Level plumbing" is a thing of the past and as an architect I can tie modules and functionality together – injecting concerns "I" think matter, where they matter

What has previously taken numerous man-years to compose/understand and implement has been achieved in a few months using Spring and Spring Modules



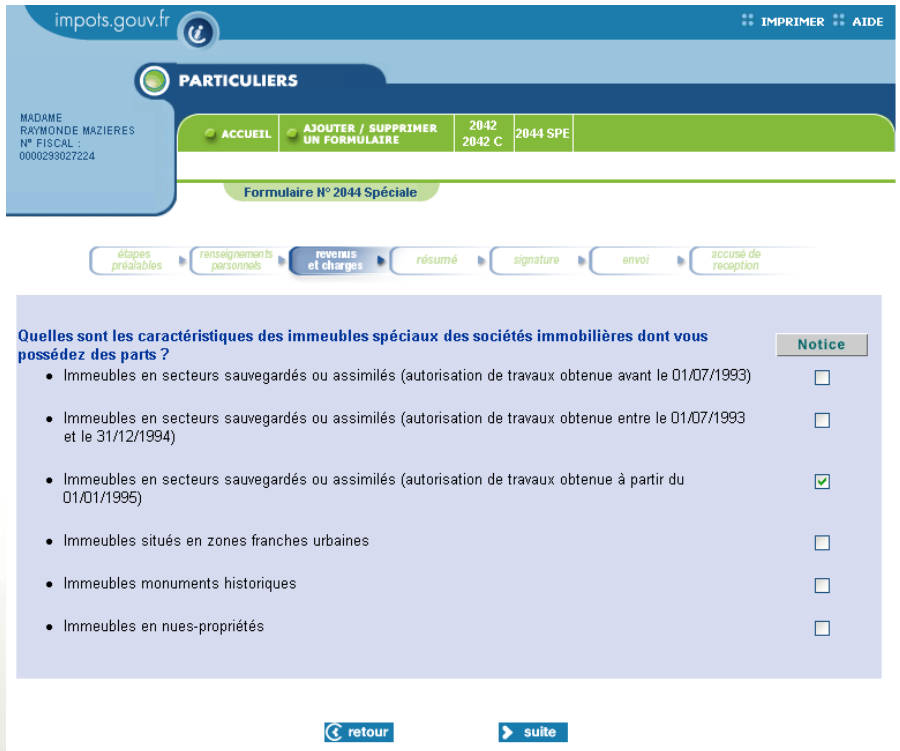
Roland Nelson
European Patent Office

A User's Perspective

French Taxation Office
Online Tax Submission System
Build by Accenture
Based on Spring

Spring has had a significant impact on the productivity of our J2EE technology developments. Thanks to its simple yet powerful programming model we were able to significantly improve time to market and build better quality solutions.

Thomas van de Velde
Lead Java Architect
Accenture Delivery Architectures



About Voca

Part of the Critical National Infrastructure for the world's fourth largest economy

Voca process Direct Debits, Direct Credits and Standing Orders to move money between banks

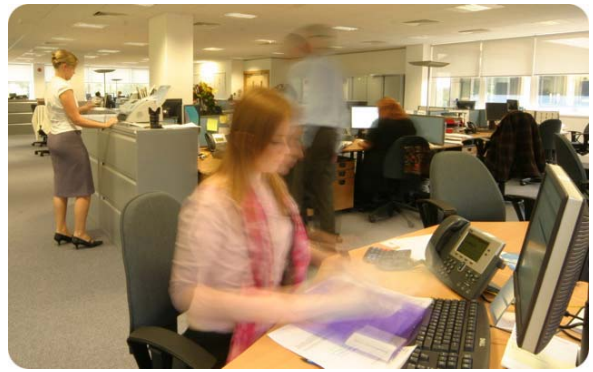
Over 5 billion transactions worth €4.5 trillion in 2005

Some 15% of Europe's Direct Debits and Direct Credits are handled by Voca

Over 70% of the UK population use Direct Debits to pay household bills; Direct Credits are used to pay over 90% of UK salaries

Over 80 million items on a peak day

In 30 years, Voca have never lost a payment



Voca’s Experience with Spring and Interface21

Increase in developer productivity without impact on system performance

Increase in testability of application components

Encourages good development practices

Strong support available

Processed a record number of transactions using Spring

Spring 2.0

- Builds on this solid base of proven technology
- Pursues vision of POJO-based development
- Adds new capabilities and makes many tasks more elegant

Spring 2.0 Goals

- Simplify common tasks
- Make Spring more powerful

Spring 2.0

- Numerous major enhancements and new features, especially…
  - Simpler, more extensible XML configuration
  - Enhanced integration with AspectJ
  - Introduces cross-language component model
  - Integration with JPA (EJB 3.0 Java Persistence API)
- Further stretches Spring's leadership in POJO programming model

Agenda

The story so far

Goals of Spring 2.0

Spring 2.0 feature overview

Extensible XML configuration

AOP enhancements and AspectJ integration

Spring 2.0: New Features

- Additional scoping options for beans
  - Backed by HttpSession etc.
  - Pluggable backing store
    - Not tied to web tier
  - Extensible and easy to use
- Numerous features in core IoC container and elsewhere to take advantage of language improvements in Java 5
  - Type inference for collections
  - New APIs leveraging Java 5 features

Type Inference: Java

```
public class DependsOnLists {

   private List plainList;

   private List<Float> floatList;

   public void setFloatList(List<Float> floatList) {
      this.floatList = floatList;
   }

   public void setPlainList(List plainList) {
      this.plainList = plainList;
   }

}
```

Type Inference: Configuration

```
<bean class="com.interface21.spring2.ioc.DependsOnLists">
   <property name="plainList">
      <list>
         <value>1</value>
         <value>2</value>
         <value>3</value>
      </list>
   </property>
   <property name="floatList">
      <list>
         <value>1</value>
         <value>2</value>
         <value>3</value>
      </list>
   </property>
</bean>
```

Spring 2.0: New Features

- Customizable task execution framework for asynchronous task execution
- CommonJ TimerManager implementation
- Portlet MVC framework
  - Analogous to Spring MVC

Spring 2.0: New Features

- Ability to define any named bean in a scripting language such as Groovy or JRuby
  - Named bean conceals both configuration and implementation language
  - Allows for DI, AOP and full range of Spring services
  - Allows dynamic reloading while honoring references
- **Spring component model is now cross-language**

Groovy components: Example

Spring 2.0: New Features

- Spring Web MVC enhancements
  - More intelligent defaulting to reduce configuration in typical cases
  - Beneficiary from dynamic language support
    - Author controllers in Groovy or JRuby, modify them on the fly during development
  - New custom tag library to simplify working with common controls
    - Analogous to Struts tag library

Spring 2.0: New Features

- **Message-driven POJOs**
  - Support for asynchronous reception of Java Message Service (JMS) API messages
    - Full support for XA-transactional receive
    - Usual Spring value proposition
      - Works in J2EE and J2SE™ platforms
  - Closes off one of the remaining corner cases justifying EJB™ usage

Spring 2.0: Ease of Use

- Configuration simplification
- MVC simplification
  - Greater use of intelligent defaulting
- **`SimpleJdbcTemplate`**
  - Designed to take advantage of generics, varargs and autoboxing on Java EE 5 platform
- **And much, much more…**

Spring 2.0: SimpleJdbcTemplate

- Motivations
  - Use new Java 5 features that can simplify usage
    - Varargs
    - Autoboxing
    - Parameterized methods
  - Offer only methods that are commonly used
    - Most commonly used callbacks
    - Fewer overloaded methods
- Offers access to a wrapped `JdbcTemplate` for more advanced operations
- `Provides the SimpleJdbcDaoSupport` class

Varargs and Autoboxing

JdbcTemplate, <= Java 1.4

```
jdbcTemplate.queryForInt("SELECT COUNT(0) FROM
   T_CLIENT WHERE TYPE=? AND CURRENCY=?",
   new Object[] { new Integer(13), "GBP" }
);
```

JdbcTemplate, autoboxing

```
jdbcTemplate.queryForInt("SELECT COUNT(0) FROM
   T_CLIENT WHERE TYPE=? AND CURRENCY=?",
   new Object[] { 13, "GBP" }
);
```

SimpleJdbcTemplate, available on Java 5

```
simpleJdbcTemplate.queryForInt("SELECT COUNT(0) FROM
   T_CLIENT WHERE TYPE=? AND CURRENCY=?",
   13, "GBP"
);
```

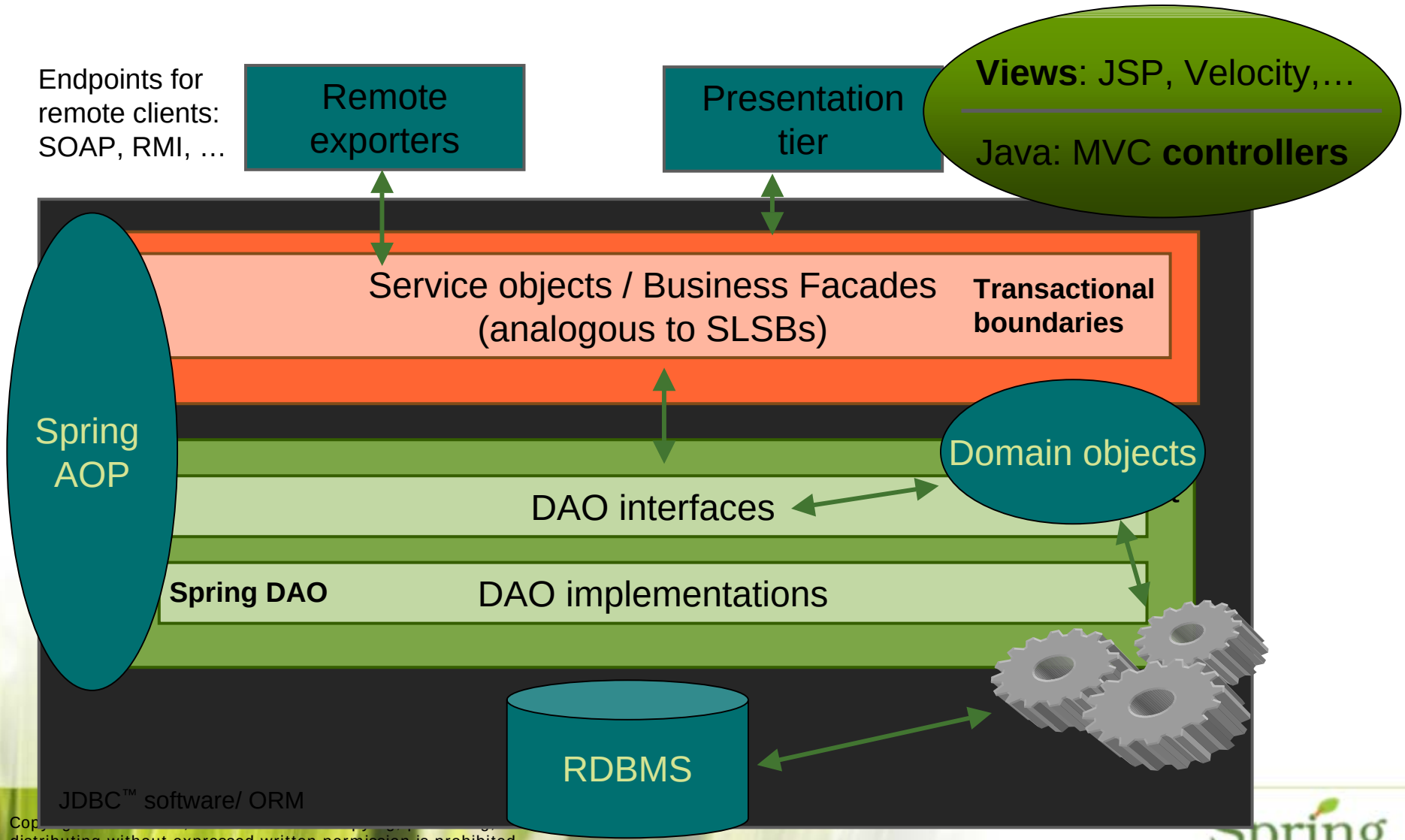
SimpleJdbcTemplate: Generics

Generics make signatures clearer and eliminate casts

```
public Map<String, Object>
   queryForMap(String sql, Object... args)
            throws DataAccessException

public List<Map<String, Object>>
   queryForList(String sql, Object ... args)
            throws DataAccessException
```

Spring 2.0 Enhancements



Agenda

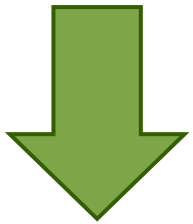
- The story so far
- Goals of Spring 2.0
- Feature overview
- **Extensible XML configuration**
- AOP enhancements and AspectJ integration
- JPA integration

XML Configuration in Spring 2.0

- Ability to define new XML tags to produce one or more Spring bean definitions
- Tags out of the box for common configuration tasks
- Problem-specific configuration
  - Easier to write and to maintain
- XML schema validation
  - Better out of the box tool support
  - Code completion for free
- Exploits the full power of XML
  - Namespaces, schema, tooling
- Backward compatibility
  - Full support for <beans> DTD

XML Configuration in Spring 2.0

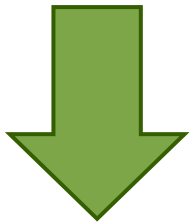
```
<bean id="dataSource" class="...JndiObjectFactoryBean">
  <property name="jndiName" value="jdbc/StockData"/>
</bean>
```



```
<jee:jndi-lookup id="dataSource"
       jndiName="jdbc/StockData"/>
```

XML Configuration in Spring 2.0

```
<bean id="properties" class="...PropertiesFactoryBean">
  <property name="location" value="jdbc.properties"/>
</bean>
```



```
<util:properties id="properties"
                 location="jdbc.properties"/>
```

Transaction Simplification

- Specialized tags for making objects transactional
  - Benefit from code assist
- **`<tx:annotation-driven />`**
- Code assist for transaction attributes

Extended Configuration Options

- Java Management Extensions
- Remoting
- Scheduling
- MVC
- Suggestions and contributions welcome
  - A rich library will build over time

Authoring custom extensions: Step 1

- Write an XSD to define element content
  - Allows sophisticated validation, well beyond DTD
  - Amenable to tool support during development
  - Author with XML tools
    - XML Spy

Authoring custom extensions: Step 2

- Implement a NamespaceHandler to generate Spring BeanDefinitions from element content
- Helper classes such as BeanDefinitionBuilder to make this easy

```
public interface NamespaceHandler {

    BeanDefinitionParser findParserForElement(
                                            Element element);
    BeanDefinitionDecorator findDecoratorForElement(
                                            Element element);
}


public interface BeanDefinitionParser {

    void parse(Element element,
                        BeanDefinitionRegistry registry);
}
```

Authoring custom extensions: Step 3

- Add a mapping in META-INF/spring.handlers
- Can add or hide handlers

```
http\://www.springframework.org/schema/util=org.springfram
   ework.beans.factory.xml.UtilNamespaceHandler
http\://www.springframework.org/schema/aop=org.springframe
   work.aop.config.AopNamespaceHandler
http\://www.springframework.org/schema/jndi=org.springfram
   ework.jndi.config.JndiNamespaceHandler
http\://www.springframework.org/schema/tx=org.springframew
   ork.transaction.config.TxNamespaceHandler
http\://www.springframework.org/schema/mvc=org.springframe
   work.web.servlet.config.MvcNamespaceHandler
```

Using custom extensions

- Import relevant XSD
- Use the new elements

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop"

    xsi:schemaLocation="http://www.springframework.org/schema/b
    eans http://www.springframework.org/schema/beans/spring-
    beans.xsd
        http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop.xsd">
```

XML Configuration Best Practices

- Standard **<bean>** tags
  - Still a great solution
  - General configuration tasks
  - Application-specific components
    - DAOs, Services, Web Tier
- Custom tags
  - Infrastructure tasks
    - Java Naming and Directory Interface™ API , Properties, AOP, Transactions
  - 3rd party packages

Agenda

The story so far

Goals of Spring 2.0

Feature overview

Extensible XML configuration

AOP enhancements and AspectJ integration

Recap: Why AOP Matters

- Essential complement to DI to enable a POJO programming model
- Provides the tools to think about application structure in a new way
- Both parts of the same big picture
- Let's step back…

A Common Vocabulary

Enterprise application vocabulary

the vocabulary of enterprise applications

business service

service layer

repository

dao

controller

web layer

data access layer

Requirements

- Many requirements are expressed *in terms of this vocabulary*
  - the service layer should be transactional
  - when a dao operation fails the SQL exception should be translated
  - service layer objects should not call the web layer
  - a business service that fails with a concurrency related failure can be retried

Meaningful abstractions

It would be simpler…

and more *powerful*

Meaningful abstractions

if we could use these

abstractions

directly in the

implementation

Faithfulness to requirements

It would be simpler…

and more *powerful*

Faithfulness to requirements

if the implementation looked like the requirement

- One requirement, one code module
- No need to modify many classes to implement one requirement

System Architecture

```
@Aspect
public class SystemArchitecture {

  @Pointcut("within(a.b.c.service..*")
  public void inServiceLayer() {}

  @Pointcut("within(a.b.c.dao..*")
  public void inDataAccessLayer() {}

  @Pointcut("execution(* a.b.c.service.*.*(..))")
  public void businessService() {}

  …
}
```

AOP in Spring 2.0

- So AOP is important…
  - How do we make Spring AOP better?
- Simplified XML configuration using **<aop:*/>** tags
- Closer AspectJ integration
  - Pointcut expression language
  - AspectJ-style aspects in Spring AOP
  - @AspectJ-style aspects in Spring AOP
    - Fully interoperable with ajc compiled aspects
- Spring ships with AspectJ aspects for Spring/AspectJ users
  - Dependency injection on any object even if it isn't constructed by the Spring IoC container

Spring AOP (1.2.x): Pros and Cons

- Pros
  - Solid proxy-based model
  - Highly extensible
  - Ease of adoption
    - Zero impact on development process and server environment
- Cons
  - No pointcut expression language
  - XML configuration can be verbose
  - Highly extensible, but only in Java technology

Spring 2.0 Aims for Spring AOP

- Build on strengths, eliminate weaknesses
- Preserve ease of adoption
  - Still zero impact on development process, deployment
  - Easier to adopt
- Benefit from the power of AspectJ
- Provide a comprehensive AOP roadmap for Spring users, spanning
  - Spring AOP
  - AspectJ

Spring 2.0 Aims for Spring AOP

- Solution
  - Work with AspectJ, the de facto standard for full-featured AOP
  - AspectJ lead Adrian Colyer is now CTO at Interface21
  - Adrian is now working on Spring as well as AspectJ
  - Take advantage of XML configuration extensions

Benefits for Spring AOP

- Benefits from industry leading pointcut expression language
- Benefits from well thought-out semantics behind @AspectJ model
- Gains ability to have type-safe advice
- Benefits from input from leading AOP thinkers

Benefits for AspectJ

- AspectJ is a language, not a framework
  - Benefits from a framework offering DI and service abstractions
  - DI is as compelling for aspects as for objects
- AspectJ gains an incremental adoption path

Benefits for You

- You can use the same knowledge in Spring and AspectJ
- Exciting possibilities around rich domain models

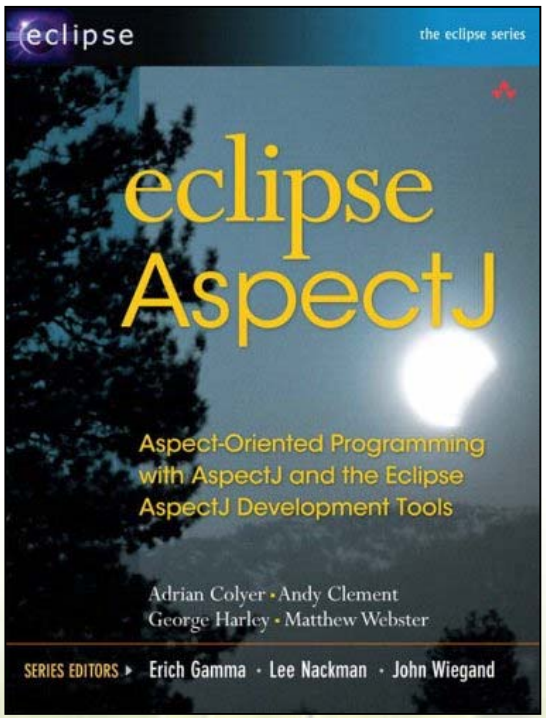
Pointcut Expressions

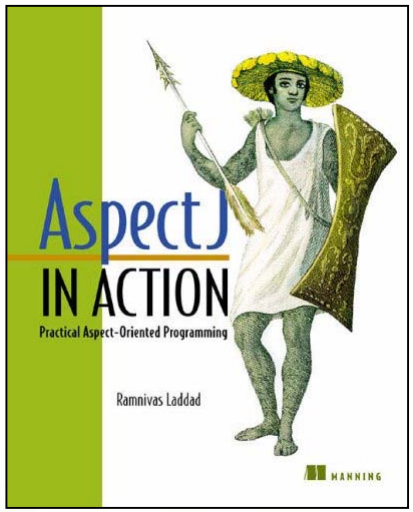
- Spring can use AspectJ pointcut expressions
  - In Spring XML
  - In @AspectJ aspects
  - In Java code (with Spring **`ProxyFactory`**)

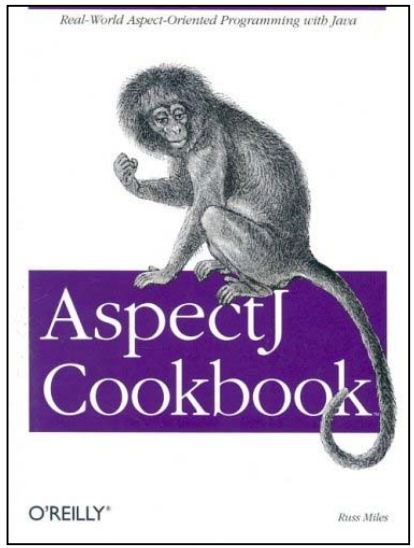
What's So Good About AspectJ Pointcut Expressions?

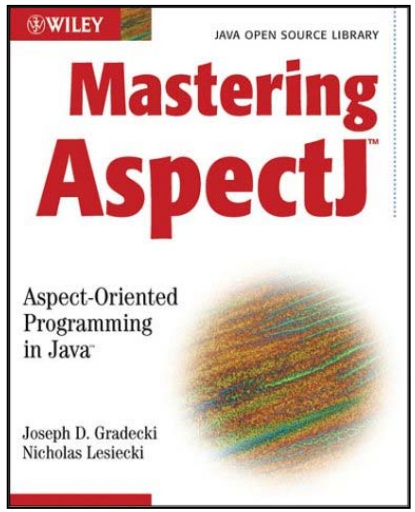
- Go far beyond simple wildcarding
- AspectJ views pointcuts as first-class language constructs
  - Can compose pointcuts into expressions
  - Can reference named pointcuts, enabling reuse
  - Can perform argument binding
  - Can express complex matching logic concisely

AspectJ Is Well Documented...









AOP Is About Pointcuts

- Pointcuts give us the tool to think about program structure in a different way to OOP
- Without a true pointcut model we have only trivial interception
  - Does not achieve aim of modularizing crosscutting logic
  - DRY (Don't repeat yourself) Principle
- Spring AOP has always had true pointcuts
  - But now they are dramatically improved

POJO Methods as Advice

```
public class JavaBeanPropertyMonitor {

  private int getterCount = 0;
  private int setterCount = 0;

  public void beforeGetter() {
    this.getterCount++;
  }

  public void afterSetter() {
    this.setterCount++;
  }

  ...
```

Applying Pointcuts

```
<aop:config>
  <aop:aspect bean="javaBeanMonitor">
    <aop:before
       pointcut=
        "execution(public !void get*())"
       method="beforeGetter"
    />
    <aop:afterReturning
       pointcut=
        "execution(public void set*(*))"
       method="afterSetter"
    />
   </aop:aspect>
</aop:config>
```

@AspectJ-style Aspects

```
@Aspect
public class AjLoggingAspect {
    @Pointcut("execution(* *..Account.*(..))")
    public void callsToAccount(){}

    @Before("callsToAccount()")
    public void before(JoinPoint jp) {
       System.out.println("Before [" +
                   jp.toShortString() + "].");
    }

    @AfterReturning("callsToAccount()")
    public void after() {
        System.out.println("After.");
    }
}
```

Setting up @AspectJ-style Aspects in Spring

```
<aop:aspectj-autoproxy/>

<bean id="account" class="demo.Account"/>

<bean id="aspect" class="demo.ataspectj.AjLoggingAspect"/>
```

Argument binding

```
@Aspect
public class TrackStringSetters {

    // Of course saving all names is not suitable for
    // production use, but this is a simple sample
    private List<String> namesRequested = new LinkedList();


    @Before("execution(* *.set*(String)) && args(name) && this(mytype)")
    public synchronized void onStringSetter(String name, Mytype mytype) {
          if (namesRequested.size() > historySize) {
                    namesRequested.remove(0);
          }
          namesRequested.add(name);
    }

    public List<String> getNamesRequested() {
          return namesRequested;
    }

}
```

Spring 2.0: Unique AOP Unification

- Brings same programming model (based on AspectJ) to proxy-based and class weaving based AOP
  - Choice of implementation strategies
  - Consistent programming model
  - Based on AspectJ, proven de facto standard for AOP
- Can compile aspects or use AspectJ load-time weaving, preserving the same programming model
- Again, no conflict between simplicity and power
  - Less powerful, less general mechanisms are simplistic, rather than simple

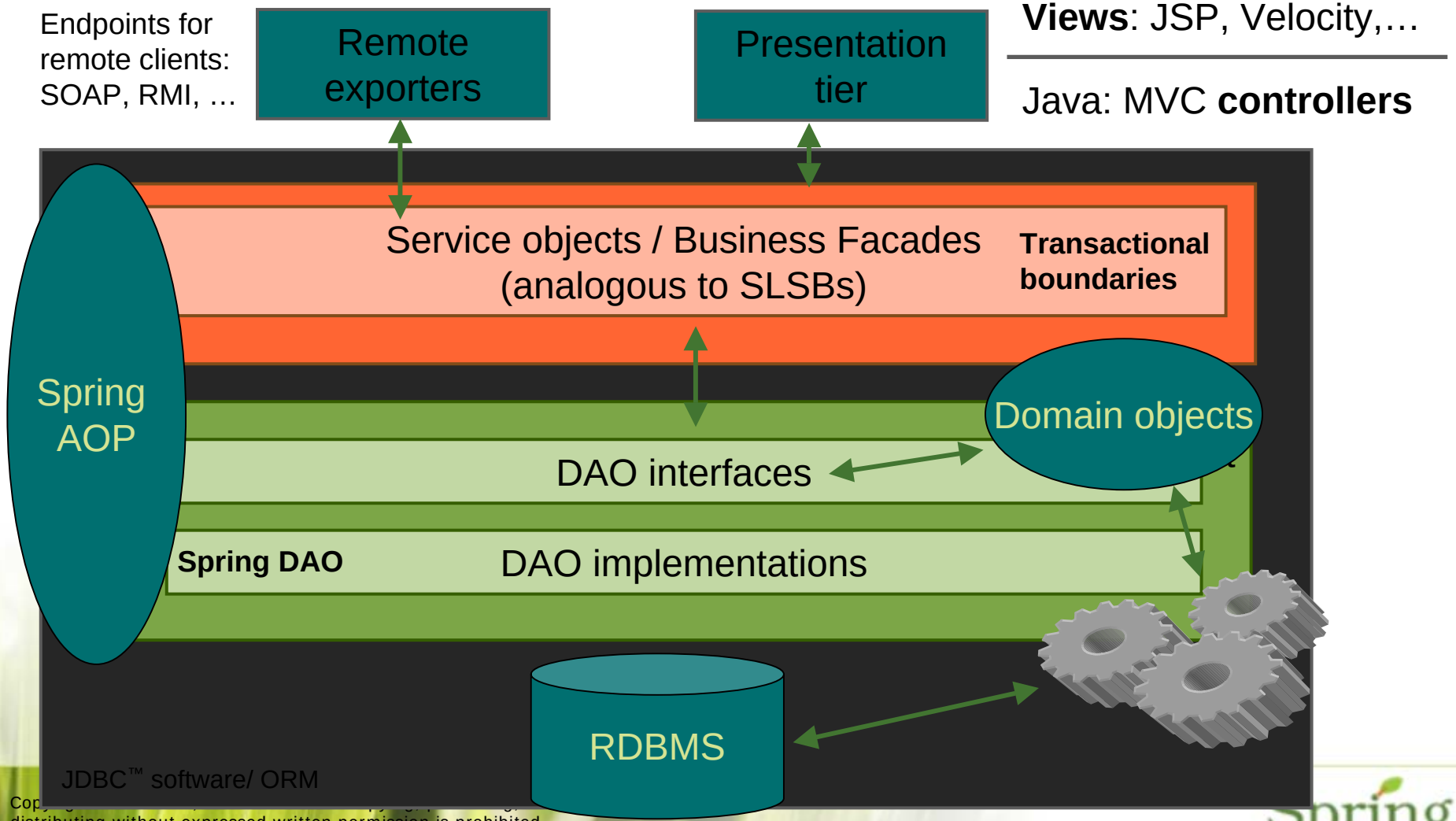
Spring 2.0 AOP: Demo

Spring and Java Persistence API

- Java Persistence API (JPA) is the persistence part of the Enterprise JavaBeans™ 3.0 specification
  - Finally standardizes real-world O/R mapping functionality
- Spring 2.0 integrates Java Persistence API in its consistent data access abstraction
- As always, Spring offers
  - Unified programming model for Java EE and Java SE platforms
  - Ease of testing (without need to deploy to an application server)
- Spring allows access to full JPA functionality without an EJB container
- Value adds beyond the JPA 1.0 specification *that work portably across all leading persistence providers*

Spring 2.0 JPA: Demo

Spring 2.0: What Breaks?



Spring 2.0: What Breaks?

- Spring 2.0 is fully backward compatible
- Enterprise-class technologies can't remain credible if they break existing application code
- POJO-based technology offers the stability in programming model J2EE technology has lacked
  - Spring offers the mature, proven realisation
    - Across J2EE and J2SE platforms

Do I Need Java 5 with Spring 2.0?

- No, but you'll get an increasing amount of cool stuff if you are able to use Java 5
  - Spring 1.2 already introduced value adds on Java 5, such as @Transactional
  - AspectJ integration requires Java 5 for full range of pointcut expressions
- Spring 2.x series will run on Java platform 1.3 and above
- Continues to run on all leading application servers, web containers
  - Or without any other container

Summary (1)

- Spring 2.0 Aims
  - Build on core Spring aim of offering a POJO programming model
  - Make Spring both simpler to use and more powerful
- Spring 2.0 introduces simplified, extensible XML configuration
  - Custom tags for Java Naming and Directory Interface API, AOP, transactions and more
- Significant improvements in Spring AOP
  - Pointcut expression support
  - AspectJ-style aspect support
  - @AspectJ aspect support

Summary (2)

- Many other enhancements, including…
  - TaskExecutor abstraction
  - Adds asynchronous JMS API to complement existing synchronous JMS API support
    - Message-driven POJOs
      - Message reception within XA transaction
  - Ease-of-use improvements for Spring MVC
  - Portlet MVC framework

Questions