Spring Framework Update: Introducing Spring 2.0

Rod Johnson

CEO
Interface21
www.interface21.com

TS-3744

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

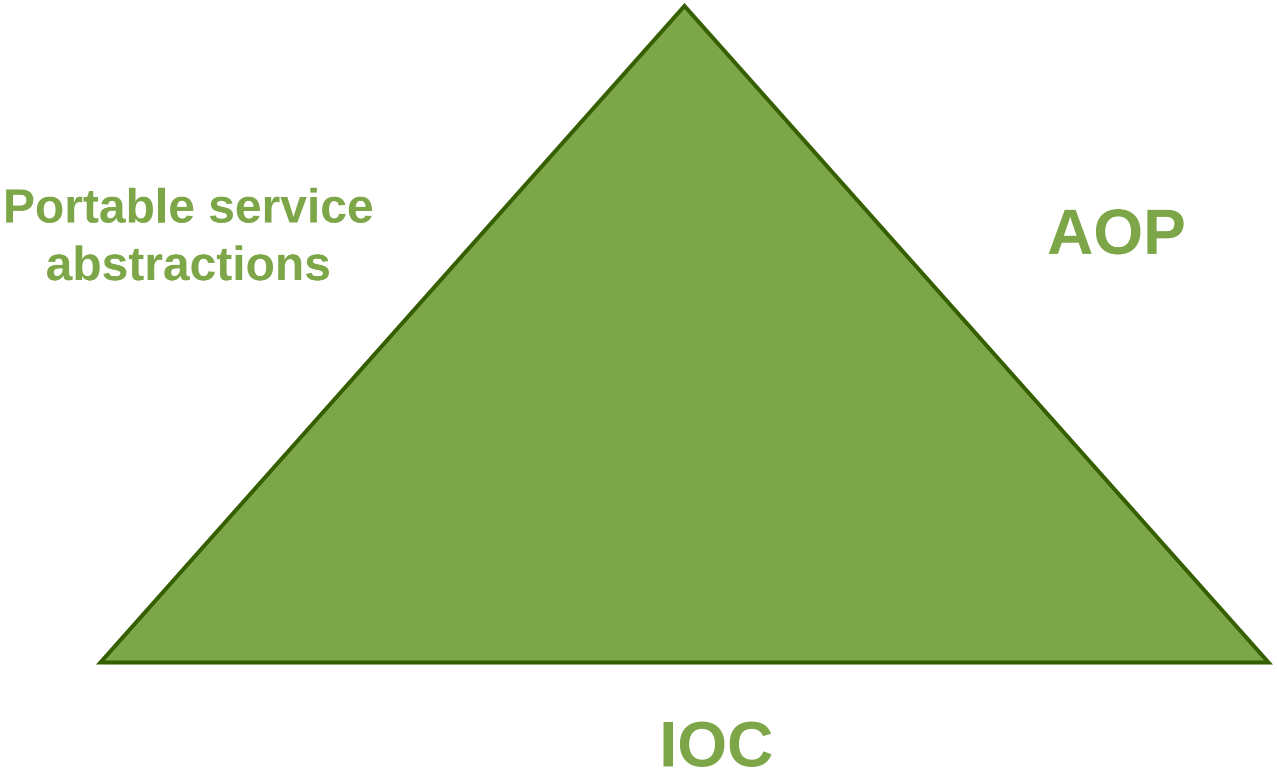
- **POJO** stands for **Plain Old Java based Object**
- A POJO is not bound to any environment
  - No imports of classes that bind it to a specific environment
  - Not dependent on a particular lookup mechanism
    - Collaborating instances are injected using plain Java constructors or setter methods
  - 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: The Spring Triangle

Power to the POJO



Portable service abstractions

AOP

IOC

Spring in Practice

- Solidity of core Spring abstractions has since 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

Who Uses Spring?

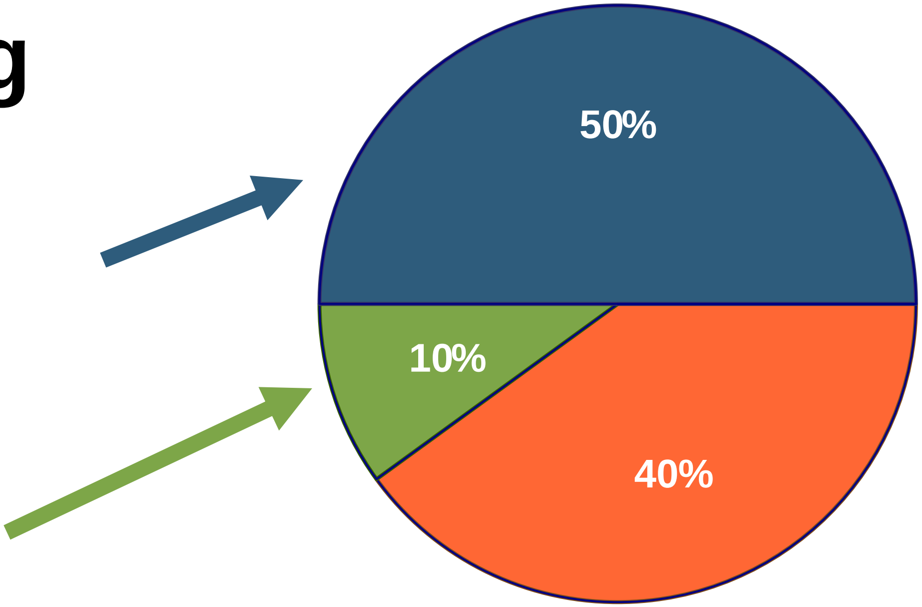
- Extensive and growing usage across many industries, including:
  - Retail and investment banks
  - Insurance companies (US and Europe)
  - Government
    - European Patent Office
    - French Online Taxation System
    - US, Canadian and Australian Government Agencies

Who Uses Spring?

- Scientific Research
  - CERN
- Airline industry
- Defence
- Media and online businesses
  - eBay
  - And many others
- Software
  - BEA Systems
    - WebLogic JEE 5 preview uses Spring internally
    - Spring helped speed time to market

Focus: Banking

- 5 out of the world’s 10 largest banks are Spring users and Interface21 clients
- At least one more is also using Spring on multiple projects



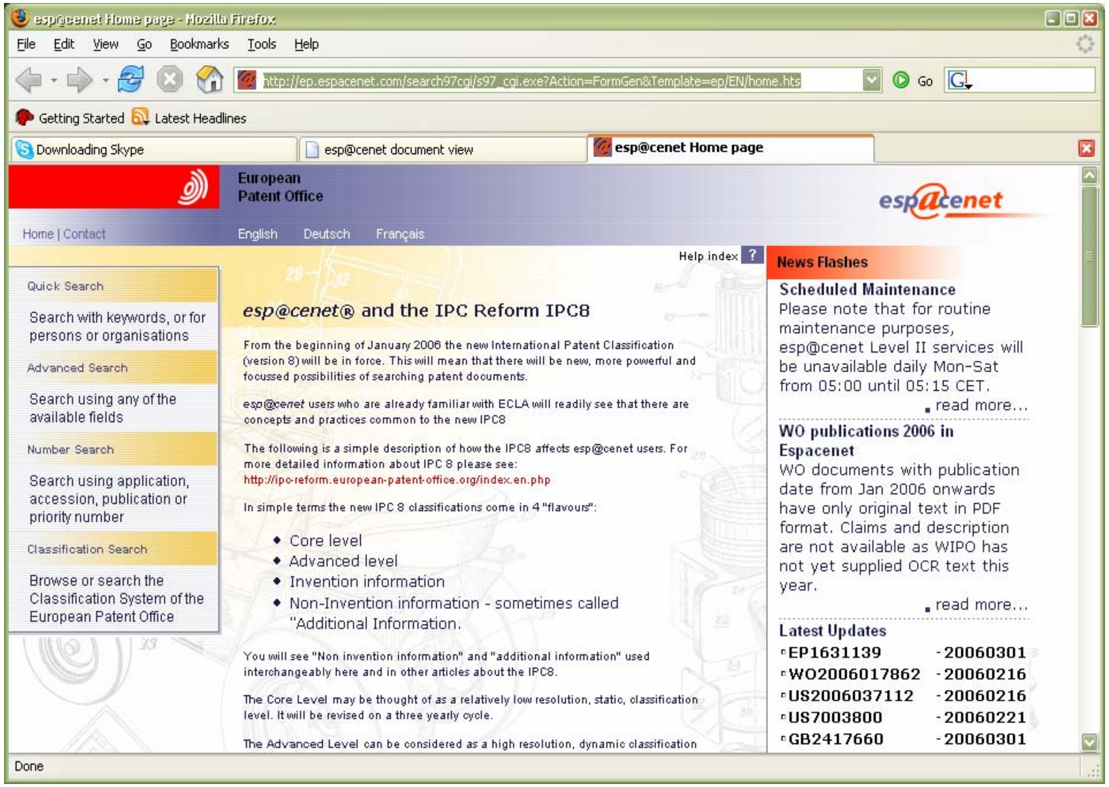
Source: *The Economist* – “The world’s biggest banks” (2005)

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



Search with keywords, or for persons or organisations

Search using any of the available fields

Search using application, accession, publication or priority number

Browse or search the Classification System of the European Patent Office

esp@cenet® and the IPC Reform IPC8

From the beginning of January 2006 the new International Patent Classification (version 8) will be in force. This will mean that there will be new, more powerful and focussed possibilities of searching patent documents.

esp@cenet users who are already familiar with ECLA will readily see that there are concepts and practices common to the new IPC8

The following is a simple description of how the IPC8 affects esp@cenet users. For more detailed information about IPC 8 please see:
http://ipc-reform.european-patent-office.org/index.en.php

In simple terms the new IPC 8 classifications come in 4 "flavours":

- Core level
- Advanced level
- Invention information
- Non-Invention information - sometimes called "Additional Information.

You will see "Non invention information" and "additional information" used interchangeably here and in other articles about the IPC8.

The Core Level may be thought of as a relatively low resolution, static, classification level. It will be revised on a three yearly cycle.

The Advanced Level can be considered as a high resolution, dynamic classification

News Flashes

Scheduled Maintenance
Please note that for routine maintenance purposes, esp@cenet Level II services will be unavailable daily Mon-Sat from 05:00 until 05:15 CET.
read more...

WO publications 2006 in Espacenet
WO documents with publication date from Jan 2006 onwards have only original text in PDF format. Claims and description are not available as WIPO has not yet supplied OCR text this year.
read more...

Latest Updates

EP1631139	- 20060301
WO2006017862	- 20060216
US2006037112	- 20060216
US7003800	- 20060221
GB2417660	- 20060301

A User’s Perspective

French Taxation Office

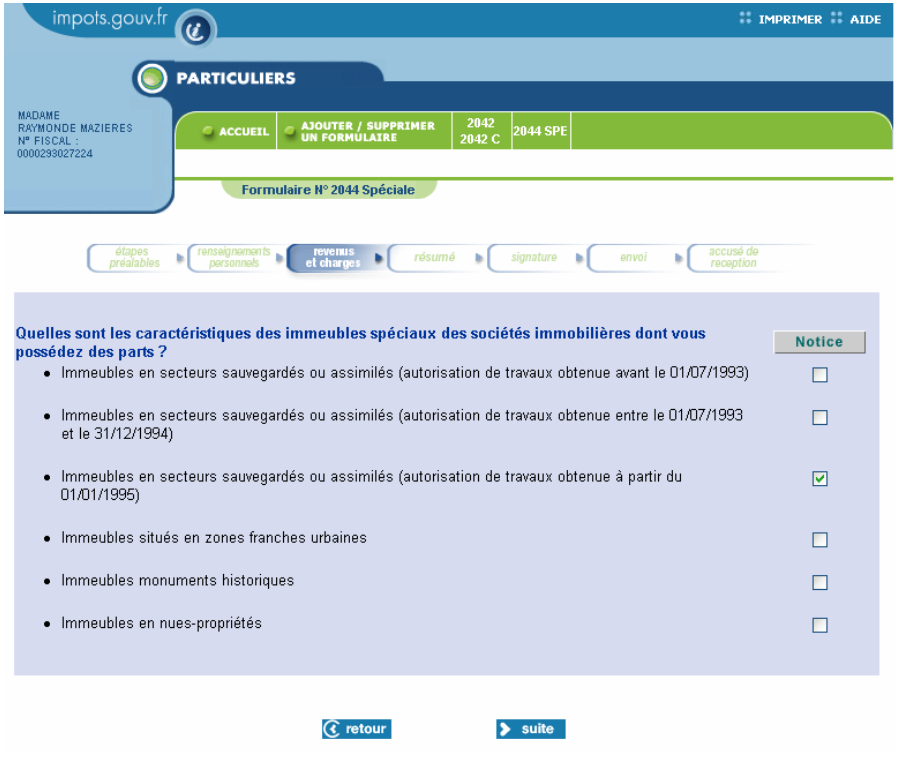
Online Tax Submission System

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



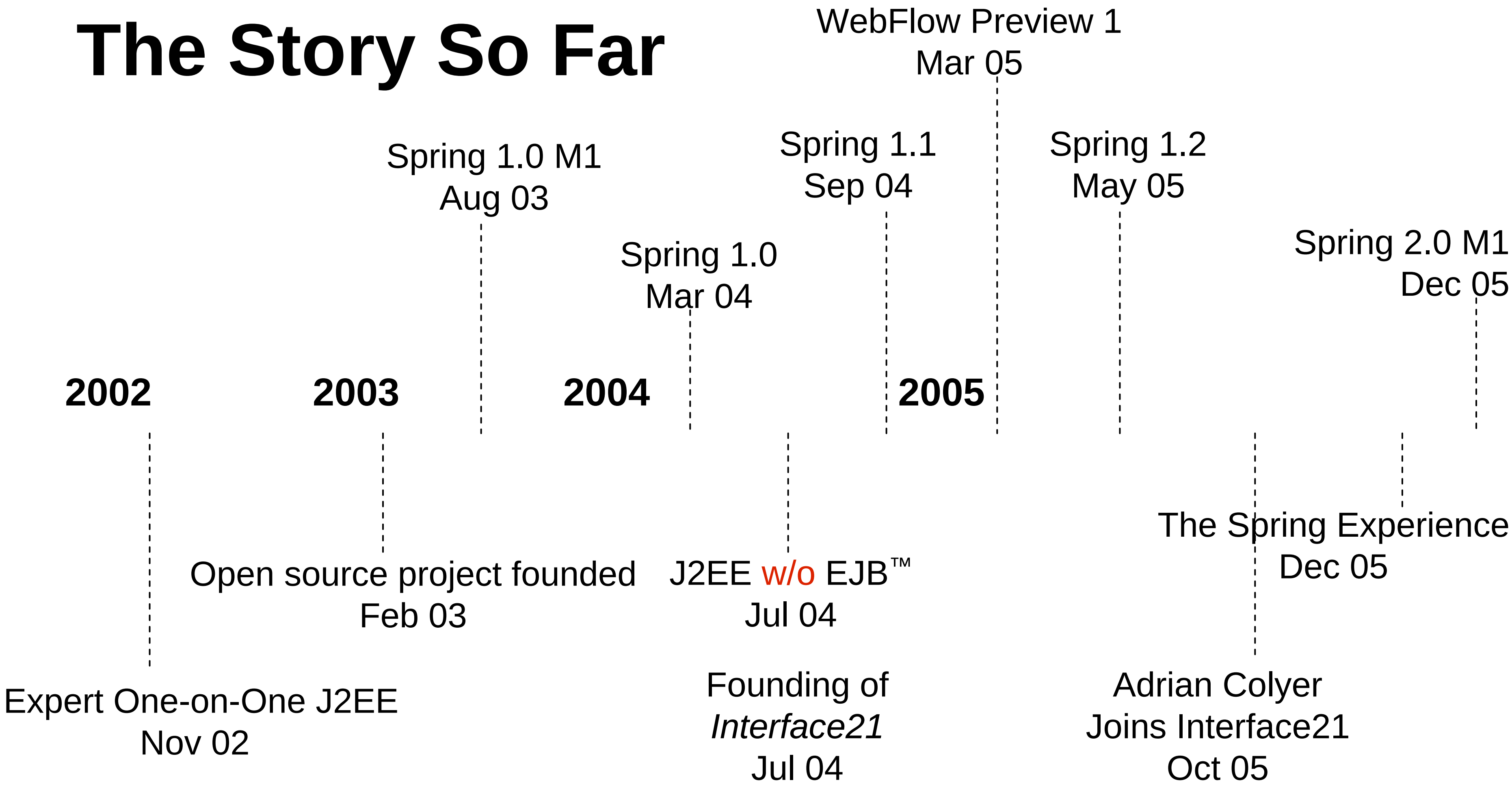
MADAME
RAYMONDE MAZIERES
N° FISCAL :
0000293027224

Formulaire N° 2044 Spéciale

Quelles sont les caractéristiques des immeubles spéciaux des sociétés immobilières dont vous possédez des parts ?

- Immeubles en secteurs sauvegardés ou assimilés (autorisation de travaux obtenue avant le 01/07/1993) □
- Immeubles en secteurs sauvegardés ou assimilés (autorisation de travaux obtenue entre le 01/07/1993 et le 31/12/1994) □
- Immeubles en secteurs sauvegardés ou assimilés (autorisation de travaux obtenue à partir du 01/01/1995) ☑
- Immeubles situés en zones franches urbaines □
- Immeubles monuments historiques □
- Immeubles en nues-propriétés □

The Story So Far



2002

Expert One-on-One J2EE
Nov 02

2003

Open source project founded
Feb 03

Spring 1.0 M1
Aug 03

2004

Spring 1.0
Mar 04

J2EE w/o EJB™
Jul 04

Founding of
Interface21
Jul 04

Spring 1.1
Sep 04

2005

WebFlow Preview 1
Mar 05

Spring 1.2
May 05

Adrian Colyer
Joins Interface21
Oct 05

The Spring Experience
Dec 05

Spring 2.0 M1
Dec 05

Spring 2.0

- Builds on this solid base
- 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

- Final release June 2006
- Spring 2.0 RC1 released for JavaOne
  - Feature complete

Spring 2.0

- Numerous major enhancements and new features, especially…
  - Simpler, more extensible XML configuration
  - Enhanced integration with AspectJ
  - 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
  - Used by Sony (major Spring users)
  - 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

Spring 2.0: New Features

- Customizable task execution framework for asynchronous task execution
- CommonJ TimerManager implementation
  - Great for WebLogic/Websphere users
- 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 dynamic reloading
- Spring MVC enhancements
  - More intelligent defaulting to reduce configuration in typical cases
  - Beneficiary from scripting support
  - 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™ specification usage

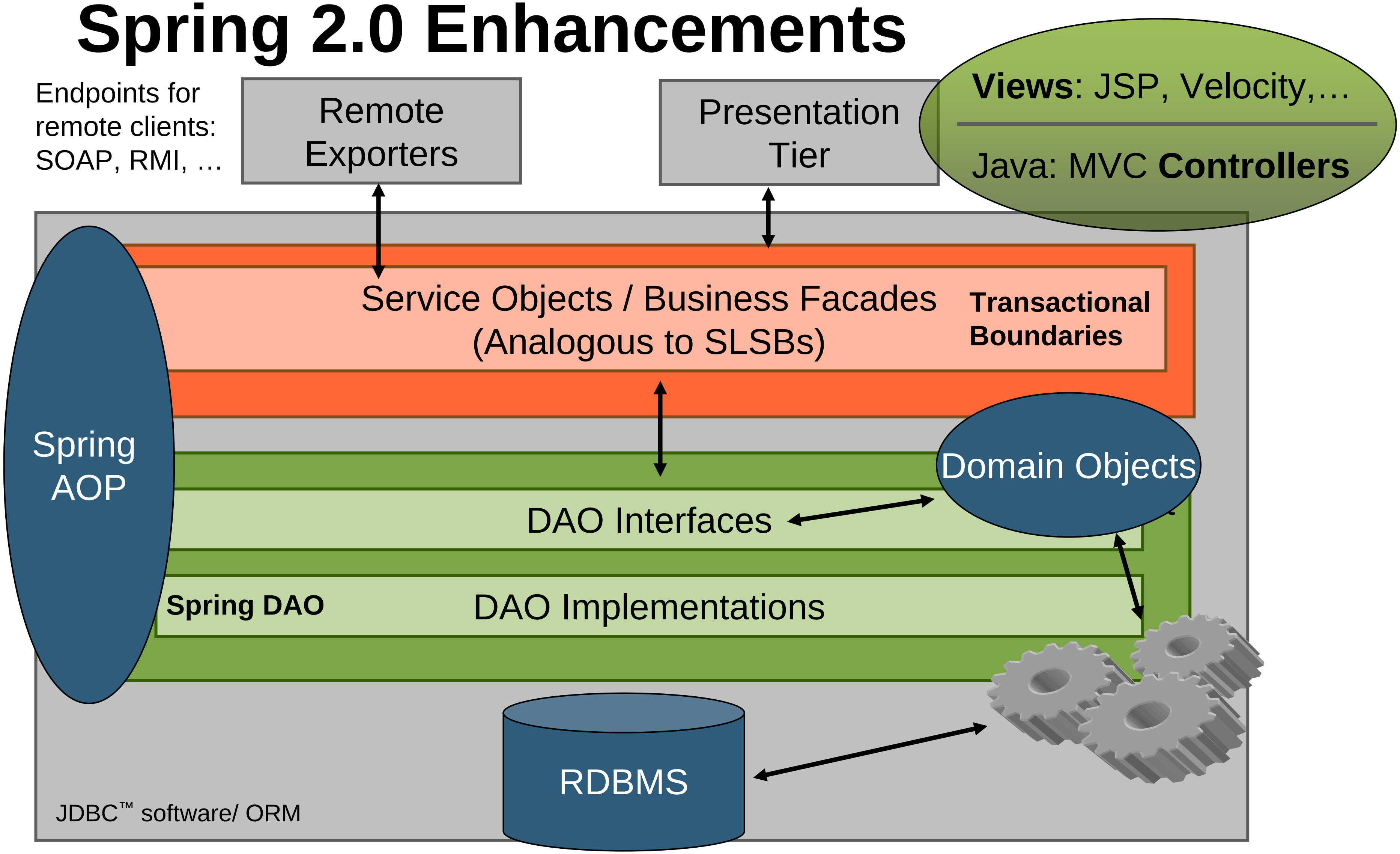
Spring 2.0: Ease of Use

- 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 and Java Persistence API

- Java Persistence API 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 will offer
  - 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 Enhancements



Agenda

The Story So Far

Goals of Spring 2.0

Feature Overview

Extensible XML Configuration

AOP Enhancements and Aspectj 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

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
- Both parts of the same big picture
- Let’s step back

Information Hiding Makes It Possible to Build Large Systems

"Every module in the decomposition is characterized by its knowledge of a design decision which it hides from all others…Its interface or definition was chosen to reveal as little as possible about its inner workings."

—On the criteria to be used in decomposing a system into modules

Parnas, 1972

Separation of Concerns

- A software engineering principle, each module
  - Does one thing
  - Knows one thing
  - keeps the secret of how it does it hidden

“But nothing is gained—on the contrary!—by tackling these various aspects simultaneously. It is what I sometimes have called “the separation of concerns”, which, even if not perfectly possible, is yet the only available technique for effective ordering of one's thoughts, that I know of.”

—On the role of scientific thought

Edsger W. Dijkstra, 1974

DI as Information Hiding

- Where the dependencies come from
- What particular implementation
- Singleton or not

Dependency Injection and Aspects

- Information hiding
  - DI hides the details of resource and collaborator discovery
- Modularity
  - Aspects hide the detail of service invocation
- Separation of concerns
  - Aspects can encapsulate rules and policies shared by implementation constructs
- A complementary partnership

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 real 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 Chief Scientist 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
  - “Father of AOP” Gregor Kiczales is giving a keynote at SpringOne in June

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

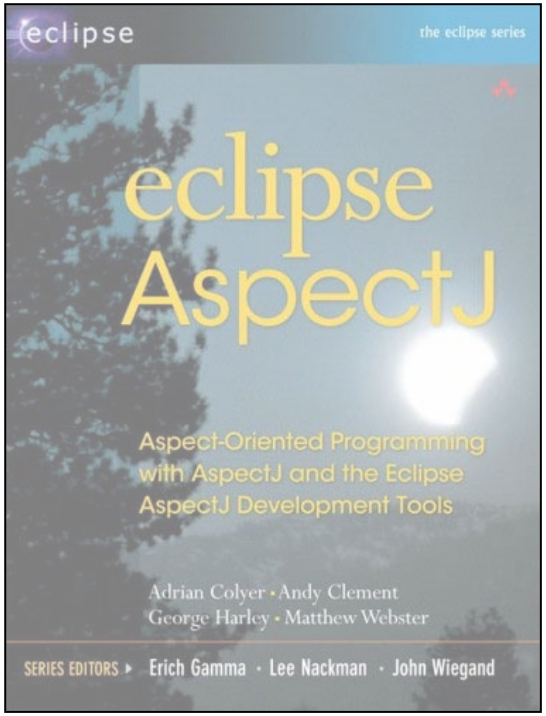
Pointcut Expressions

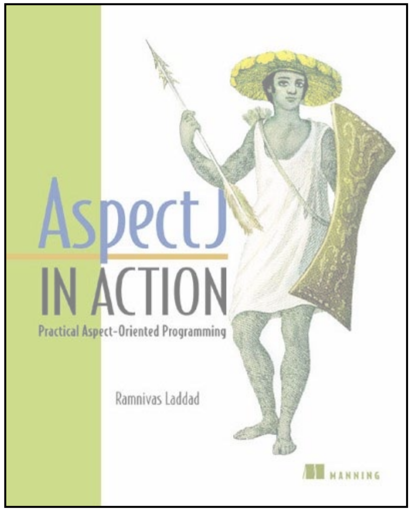
- Spring can use AspectJ pointcut expressions
  - In Spring XML
  - In @AspectJ aspects
  - In Java code (with Spring **`ProxyFactory`**)
- New **`AspectJExpressionPointcut`** will become the most used Spring AOP Pointcut implementation

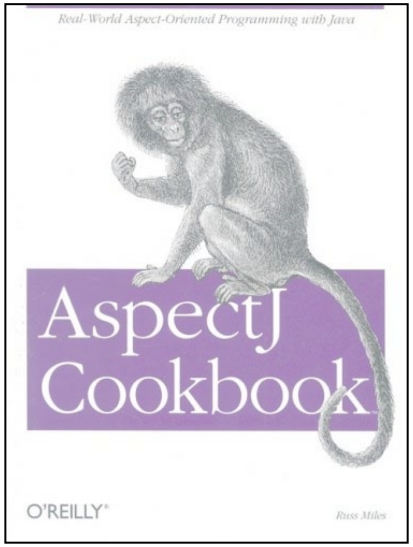
What’s So Good About AspectJ Pointcut Expressions?

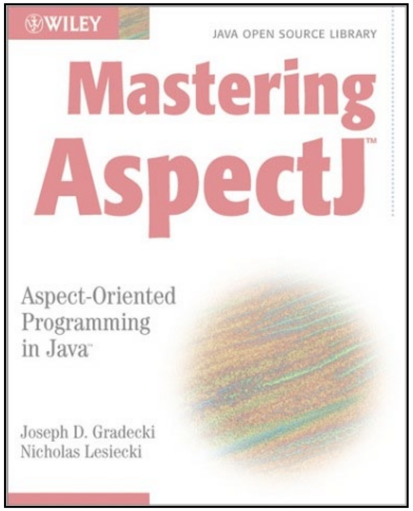
- Go far beyond simple wildcarding
- AspectJ views pointcuts as first-class language constructs
    - Can compose pointcuts into expressions
    - Can reference named pointcuts, enabling reuse
    - Can perform argument binding
    - Can express complex matching logic concisely

AspectJ Is Well Documented…


eclipse
the eclipse series
eclipse AspectJ
Aspect-Oriented Programming with AspectJ and the Eclipse AspectJ Development Tools
Adrian Colyer • Andy Clement
George Harley • Matthew Webster
SERIES EDITORS ▸ Erich Gamma • Lee Nackman • John Wiegand







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.");
    }
}
```

@AspectJ-style Aspects

```
<aop:aspectj-autoproxy/>

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

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

AOP Is More Than Interception

- Interception is merely one implementation strategy for AOP
- It is not a complete conceptual model of AOP
- Spring 2.0 aligns on AspectJ semantics
  - But is still wholly backward compatible
- Allows far more sophisticated constructs
  - Pointcuts are a first-class construct

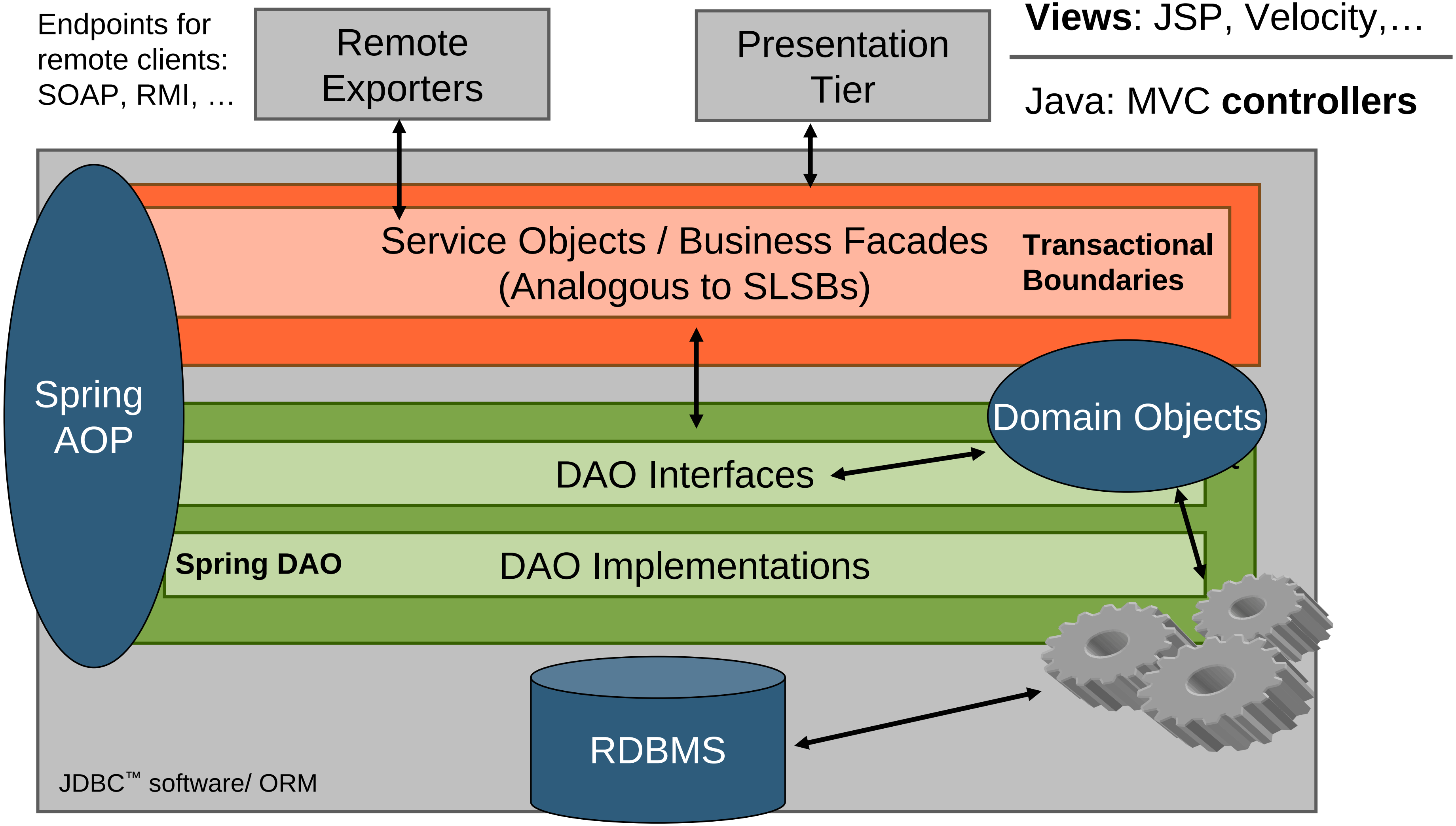
Why Not Just Interception?

- EJB 3.0 specification interception is an improvement compared to EJB 2 specification, but is essentially 2003-vintage, “AOP lite” technology
    - No real pointcut model
    - Every method or class (via annotation or corresponding XML deployment descriptor) needs to be changed to define new crosscutting behaviour
    - Fails to deliver a new structural way of thinking
    - Fails to achieve core goal of AOP of preventing crosscutting
        - Still need to change in many places to make one logical change (such as introduce auditing)
        - Concerns pile up as you have more and more annotations or interceptor definitions, reducing maintainability
        - Classes know about the interceptors that apply to them
            - Wrong way around

Spring 2.0: 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: What Breaks?



Endpoints for remote clients: SOAP, RMI, …

Remote Exporters

Presentation Tier

Views: JSP, Velocity,…

Java: MVC **controllers**

Service Objects / Business Facades (Analogous to SLSBs)

Transactional Boundaries

Spring AOP

Domain Objects

DAO Interfaces

Spring DAO

DAO Implementations

RDBMS

JDBC™ software/ ORM

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 EE 5 with Spring 2.0?

- No, but you’ll get an increasing amount of cool stuff if you are able to use Java EE 5
  - Spring 1.2 already introduced value adds on Java EE 5, such as @Transactional
  - AspectJ integration requires Java EE 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

For More Information

- Home of the Spring Framework
  - http://www.springframework.org/
- Interface21 Web Site
  - http://www.interface21.com/
- SpringOne conference in June 2006
  - http://www.springone.com

Other Spring Framework Sessions

- Keith Donald
  - Spring Web Flow
    - Powerful next generation web technology based on the Spring Framework
- Rod Johnson
  - BOF on testing with Spring (8:30 pm tonight)
  - Will cover new Spring JPA integration
  - Hands on, code-centric
  - Come along and ask questions!

Q&A

Rod Johnson

Spring Framework Update: Introducing Spring 2.0

Rod Johnson

CEO
Interface21
www.interface21.com

TS-3744