

The Spring Framework

Features & Future

Alef Arendsen

&

Rod Johnson

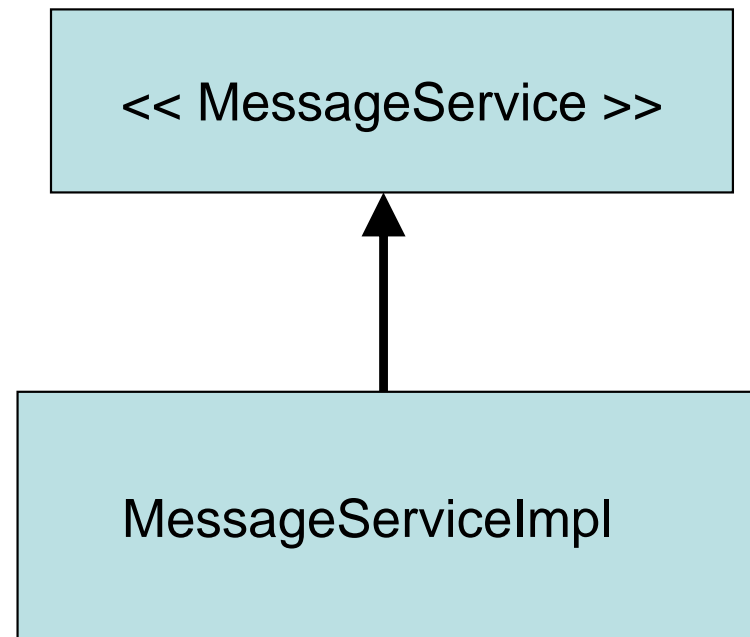
- Introduction
- A small recap
- Use Spring to create a web-service of your bean
- The MBeanExporter: JMX-enable your beans
- XML simplification
- Roadmap and future

- Interface21
 - Consulting and support services
 - The Spring Framework
 - Related technologies
- The Spring Framework
 - 1.0 March 2004
 - 1.1 September 2004
 - 1.2 May 2005
 - 1.3 July / August 2005

- Spring's vision with respect to your objects:
 - Everything can and *should* be a POJO
 - Infrastructural behavior should be applied transparently
 - No complex glue and bootstrapping code should have to be written
 - Allows you to focus on the actual problem at hand
- Spring: the backbone of your application

- Advanced *Dependency Injection* container
- Proxy-based AOP framework
- Integration with all leading persistence technologies
- Out-of-the-box platform-agnostic transaction management aspect
- Modern, flexible MVC framework
- Several possibilities for remoting objects
- Consistent design based on best practices

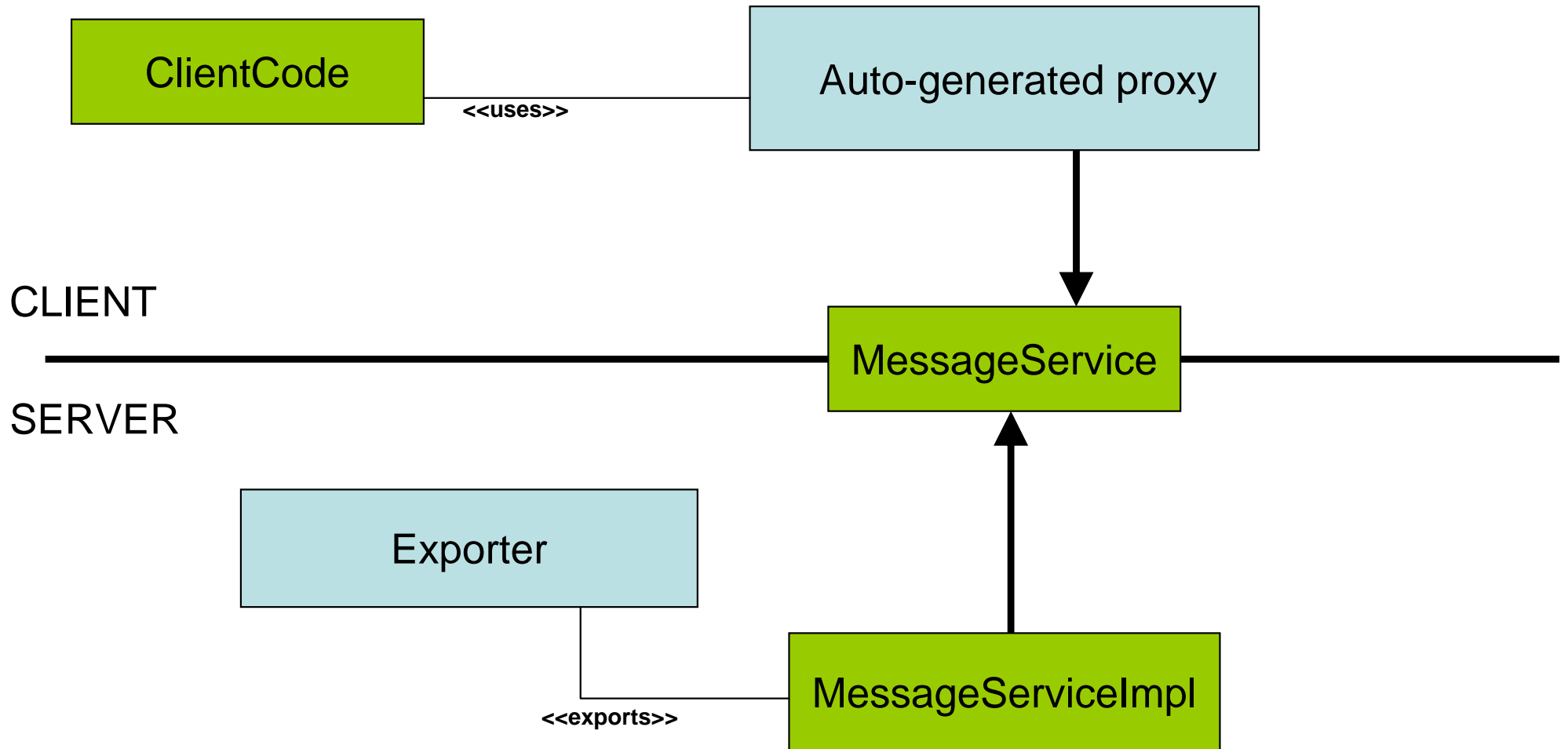
The case: transparently expose a service object through SOAP & JMX



```
public class MessageServiceImpl implements MessageService {  
    private String message;  
    public void setMessage(String message) { . . . }  
    public String getMessage() {  
        return this.message;  
    }  
    public String reverseMessage() {  
        char[] chars = this.message.toCharArray();  
        char[] reversed = new char[chars.length];  
        for (int i = chars.length - 1, x = 0; i >= 0; i--, x++) {  
            reversed[x] = chars[i];  
        }  
        return new String(reversed);  
    }  
}
```

- Spring exporters
 - Consistent approach to
 - transparently remote *any* of your beans,
 - for access using a variety of protocols,
 - no need for infrastructural code in your business object
 - Reduced development effort
 - Enhanced testability

→ Power to the POJO



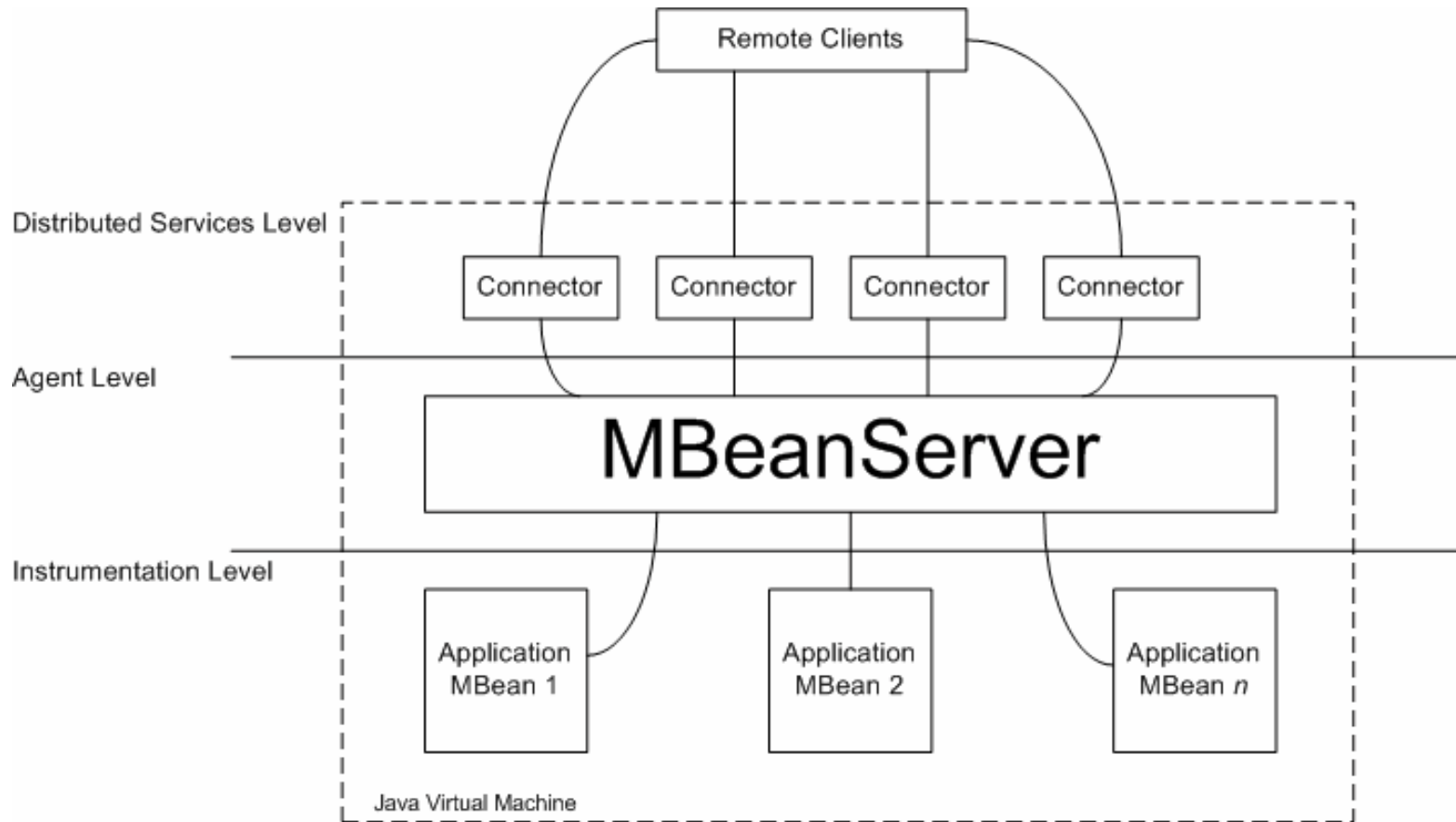
```
<bean name="/messageService"
      class="org.codehaus.xfire.spring.XFireExporter">
  <property name="service" ref="messageService"/>
  <property name="serviceInterface"
            value="example.MessageService"/>
</bean>

<bean id="messageService"
      class="example.MessageServiceImpl"/>
```

SMALL DEMO

- JMX doesn't need introduction does it?
 - Previous presentation
 - Java 5.0 introduces JMX in J2SE
- Part of the Java 5.0 J2SE distribution
- JSR-160 connectors offer access to remote clients using RMI/Hessian/SOAP

- Spring 1.2 introduces JMX exporter for POJOs
- Lead developer, Rob Harrop (Interface21) part of JMX Expert Group
- Spring **MBeanExporter** transparently MBean'ifies your POJOs



- Remember our MessageService?

```

<bean id="mbeanExporter"
  class="org.springframework.jmx.export.MBeanExporter">
  <property name="beans">
    <map>
      <entry
        key="i21:service=messageService"
        value-ref="messageService" />
    </map>
  </property>
</bean>

<bean id="messageService"
  class="example.MessageServiceImpl" />
  
```

MBean ObjectName



Let's have a look at this

DEMO

- The usual hooks to customize everything:
 - NamingStrategy
 - Defines the JMX ObjectName
 - Using either metadata, JVM object identity or the identifier from the Spring configuration
 - MBeanInfoAssembler
 - Creates the MBeanInfo object, describing your MBean
 - Using reflection, declaration of method names in Spring config or metadata (annotations)

But wait! I saw a different XML format in the code examples! What are those?

Simplifications to the
XML format added in Spring 1.2

`<property name="xxx" value="yyy" />` and
`<property name="xxx" ref="yyy" />`

in addition to

```
<property name="xxx">
```

```
  <value>yyy</value>
```

```
</property>
```

```
<property name="xxx">
```

```
  <ref bean="yyy" />
```

```
</property>
```

```
<map>
```

```
  <entry key-ref="xxx" value-ref="yyy" />
```

```
</map>
```

in addition to

```
<map>
```

```
  <entry>
```

```
    <key><ref bean="xxx" /></key>
```

```
    <ref bean="yyy" />
```

```
  </entry>
```

```
</property>
```

- Some other small enhancements *and*
- In 1.3 – namespace support in the XML
 - Facilitating third-party integration
 - Drastically simplifying the XML

```
<j2ee:jndi id="dataSource"  
  jndiName="java:comp/env/jdbc/jpetstore" />
```

INSTEAD OF

```
<bean id="dataSource"  
  class="org.springframework.jndi.JndiObjectFactoryBean">  
  <property name="jndiName">  
    <value>java:comp/env/jdbc/jpetstore</value>  
  </property>  
</bean>
```

FOR EXAMPLE

```
<mvc:controller  
  class="com.mycompany.EditAccountController"  
  url="/editAccount.html"  
  formView="myForm"  
  successView="myConfirmation"  
  commandClass="com.mycompany.Account" />
```

AND

```
<remote:http interface="example.MessageService"  
  ref="messageService"  
  url="/MessageService" />
```

- Building on Spring's key abstractions
- Extending the POJO model to new paradigms
 - Grid computing with JavaSpaces
 - Spring's remoting abstraction is powerful enough to support this *transparently*
 - Mixed implementation languages
 - Java
 - Groovy/Jython/etc

- Dynamic reconfiguration within POJO model
 - Some configuration values can be resolved dynamically by the container
 - E.g. from a database query
 - Specify caching policy, allowing reconfiguration across a cluster
 - Possible integration with JSR-170 content repository API
 - Offers versioning, branching etc.

- Spring already integrates well with AspectJ
 - Since Spring 1.1 ability to configure AspectJ aspects using dependency injection
 - Makes AspectJ aspects first class citizens
- With AspectJ 5 use `@SpringConfigured` annotation
 - Allows objects created using `new` operator to be dependency injected
- Aim: **Seamless integration of AspectJ and Spring so that they act like one framework**

- Integration with rules engines
- Integration with workflow engines
- JSR-220 persistence integration within Spring's consistent data access abstraction
- Enhanced integration with specific environments *without sacrificing portability*

- First public Spring training
 - Amsterdam: June 7 – 10
 - Followed by training events in London and Atlanta
 - Covers core Spring, WebFlow, Acegi and much more
- *Professional Java Development with the Spring Framework* – to appear in 5-6 wks.

Q & A