

# Spring & Web Services

Alef Arendsen  
Interface21



# Some questions

- Who has built (SOAP) web services?
- The different toolkits:
  - ◆ Axis?
  - ◆ XFire?
  - ◆ XINS?
  - ◆ Glue?
- Who uses JMS?
  - ◆ When thinking about message-based services, why are JMS and SOAP web service APIs so different?

- Web services, the why and the what
- Sample web service
- Implementing a subsystem with Axis
- Implementing a subsystem with XFire
- Discussion of implementations
- Preview of Spring Web Services

# Why web services (based on SOAP)

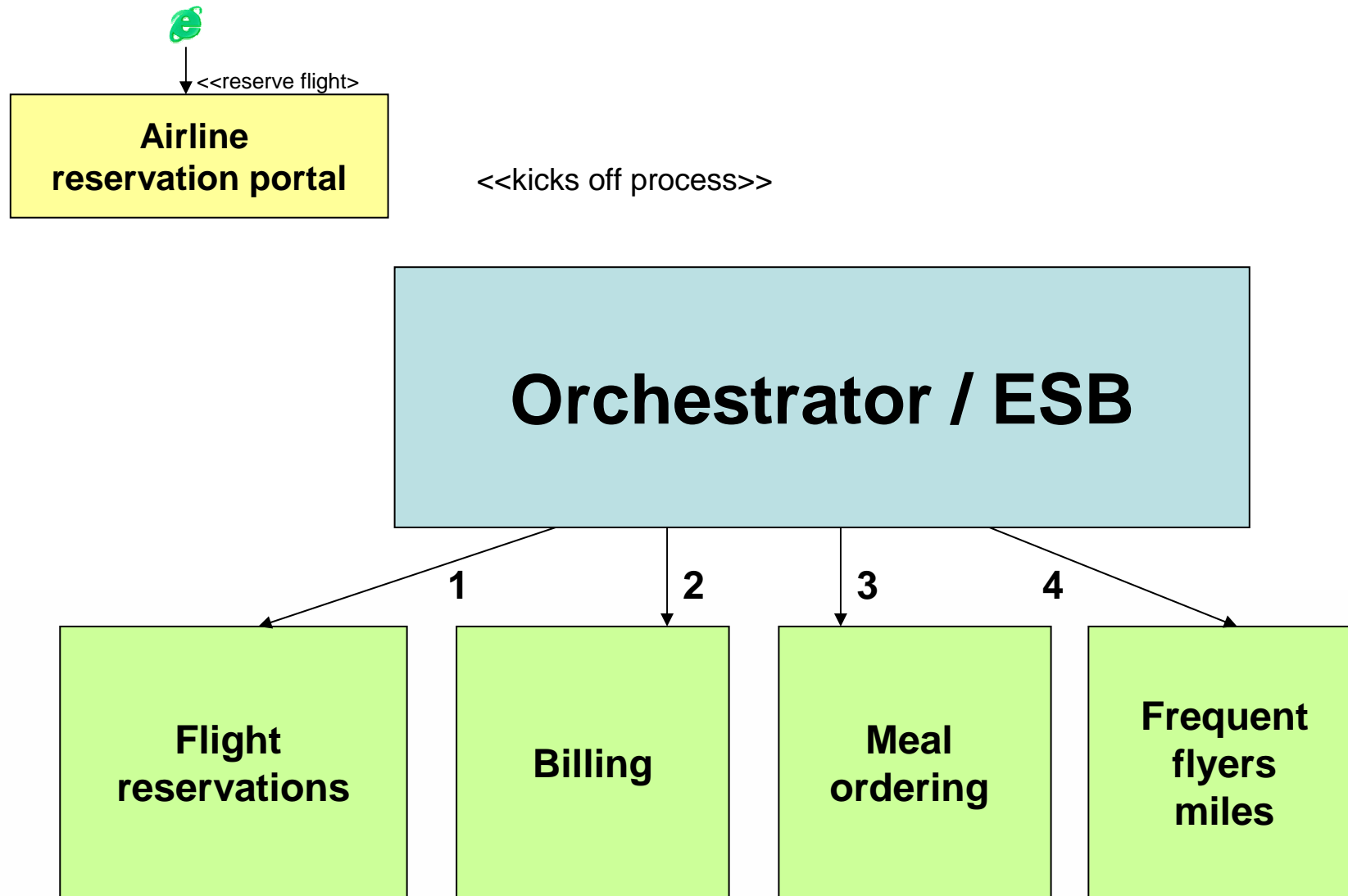
- To support Services Oriented Architecture
  - ◆ Transaction management
  - ◆ Correlation / callbacks
  - ◆ Asynchronous requests
- To overcome platform differences
  - ◆ Different platforms, different semantics
    - Procedural versus Object-Oriented
    - C++ / Java / C#, et cetera

# Sample requirements

- Airline reservation system involves
  - ◆ **Making a flight reservation**
  - ◆ Issuing an order for a meal
  - ◆ Prepare billing
  - ◆ Update frequent flyer miles account

(and probably a lot more)

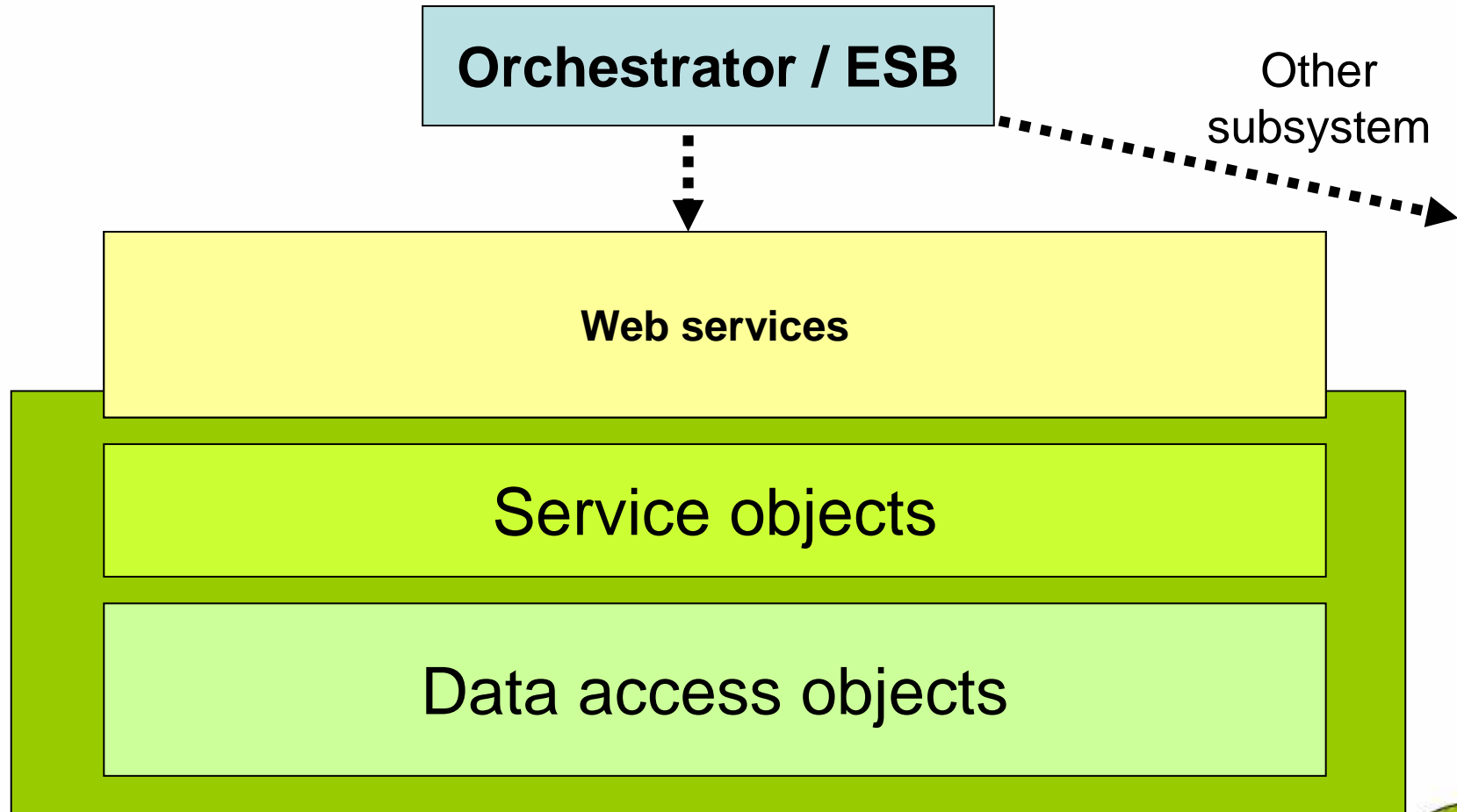
# Topology of the entire system



# Flight reservation system

```
public interface AirlineService {  
  
    List getFlightsInPeriod(  
        Calendar startOfPeriod,  
        Calendar endOfPeriod);  
  
    Reservation bookFlight(  
        String flightNumber,  
        long customerId);  
  
}
```

# Web services – exposing service objects



# Some SOAP frameworks

- Axis 1.x & Axis 2 (currently at 0.93)
  - ◆ [ws.apache.org/axis\(2\)](http://ws.apache.org/axis(2))
- XFire (currently at 1.0m6a)
  - ◆ [xfire.codehaus.org](http://xfire.codehaus.org)
- ActiveSOAP (pre-final)
  - ◆ [activesoap.codehaus.org](http://activesoap.codehaus.org)
- Spring Web Services (pre-final)
  - ◆ [springframework.org](http://springframework.org)
- XINS (1.3)
  - ◆ [xins.sf.net](http://xins.sf.net)

# Exposing services with Axis 1.x (1)

- Axis 1.x
  - ◆ Mature and popular SOAP stack
  - ◆ Provides basis for many SOAP solutions
  - ◆ Based on JAX-RPC
  - ◆ Leaning towards RPC
  - ◆ Virtually no support for exposing POJOs
  - ◆ Lots of tools (wsdl2java, java2wsdl, etcetera)

# Exposing services with Axis 1.x (2)



- Spring integrates with Axis through
  - ◆ ServletEndpointSupport
  - ◆ JaxRpcPortProxyFactoryBean & JaxRpcServicePostProcessor



## Exposing services with Axis 1.x (3)

1. Create your service (as usual)
2. Create a JAX-RPC servlet endpoint:
  - ◆ Implementing your service interface
  - ◆ Extending ServletEndpointSupport
3. Lookup appropriate bean from context
4. Delegate incoming calls
5. Wire newly created class in **server-config.wsdd**
6. Optionally create and register serializers

# Exposing services with Axis 1.x (4)

```
public class JaxRpcAirlineService
extends ServletEndpointSupport
implements AirlineService {

    private AirlineService service;

    protected void onInit() {
        service = (AirlineService)
            getWebApplicationContext().getBean("airlineService");
    }

    public List getFlightsInPeriod(Calendar start,
        Calendar end) {
        return service.getFlightsInPeriod(start,end);
    }
}

<service name="FlightManager" provider="java:RPC">
    <parameter name="className"
        value="com.airline.ws.JaxRpcFlightManager"/>
</service>
```



Let's have a look around the web service  
we've created with Axis

# What was that???

```
List getFlightsInPeriod(  
    Calendar startOfPeriod,  
    Calendar endOfPeriod);
```

VERSUS

```
Flight[] getFlightsInPeriod(  
    Calendar startOfPeriod,  
    Calendar endOfPeriod);
```



# Exposing services with XFire (1)

- XFire
  - ◆ Next generation SOAP stack
  - ◆ Less mature than Axis
  - ◆ More open architecture
  - ◆ Integration with Spring available

- Code service objects (as always)
- (Optionally create a delegate)
- Add JSR-181 annotations to service
  - ◆ @WebService
  - ◆ @WebMethod
  - ◆ @WebParam
- Wire XFire service using Spring XBean
  - ◆ Configuration format similar to new configuration features in 2.0

# Our FlightManager

```
@WebService(name="AirlineService")
public class AirlineServiceImpl
implements AirlineService

    @WebMethod(operationName="reserverFlight")
    public Flight[] getFlightsInPeriod(
        Calendar start, Calendar end) {

        // implementation of business method
    }
}

<beans xmlns="http://xfire.codehaus.org/config/1.0">
    <service>
        <serviceClass>
            com.airline.AirlineServiceImpl
        </serviceClass>
        <serviceFactory>jsr181</serviceFactory>
    </service>
</beans>
```



# Issues with remoting and web services



## Issues with remote services in general

- Latency, network issues
- Concurrency control
  
- A Remote Service **REALLY DOES** differ from a local service

# FlightManager cont.

```
public interface AirlineService {  
    public List getFlightsInPeriod(  
        Calendar start, Calendar end, Integer serviceLevel);  
}
```

```
public interface ServiceLevel {  
    public static final Integer ECONOMY = new Integer(1);  
    public static final Integer BUSINESS = new Integer(2);  
    public static final Integer FIRST = new Integer(3);  
}
```

# when no service level required, specify *null*



## Issues with updated version

- To support the old version we have to
  - ◆ Keep old version deployed separately
  - ◆ Keep old functionality around
- Specifying in .NET
  - autoboxing, *null* not supported
- What if we'd like to specify the service level as XML Schema enumeration

# Issues with XML marshalling

- XML Schema is richer than Java
- Incompatible naming
- `xsd:enumeration`
- Unserializable types
- XML=hierarchical, Java=cyclic graph
- Independent namespaces

# Issues with WSDL generation

- Cannot ensure published interface remains the same
- Every redeployment can change it!
- Varying contract is worthless
- Therefore:
  - ◆ Contract-first instead of contract-last

# The net result

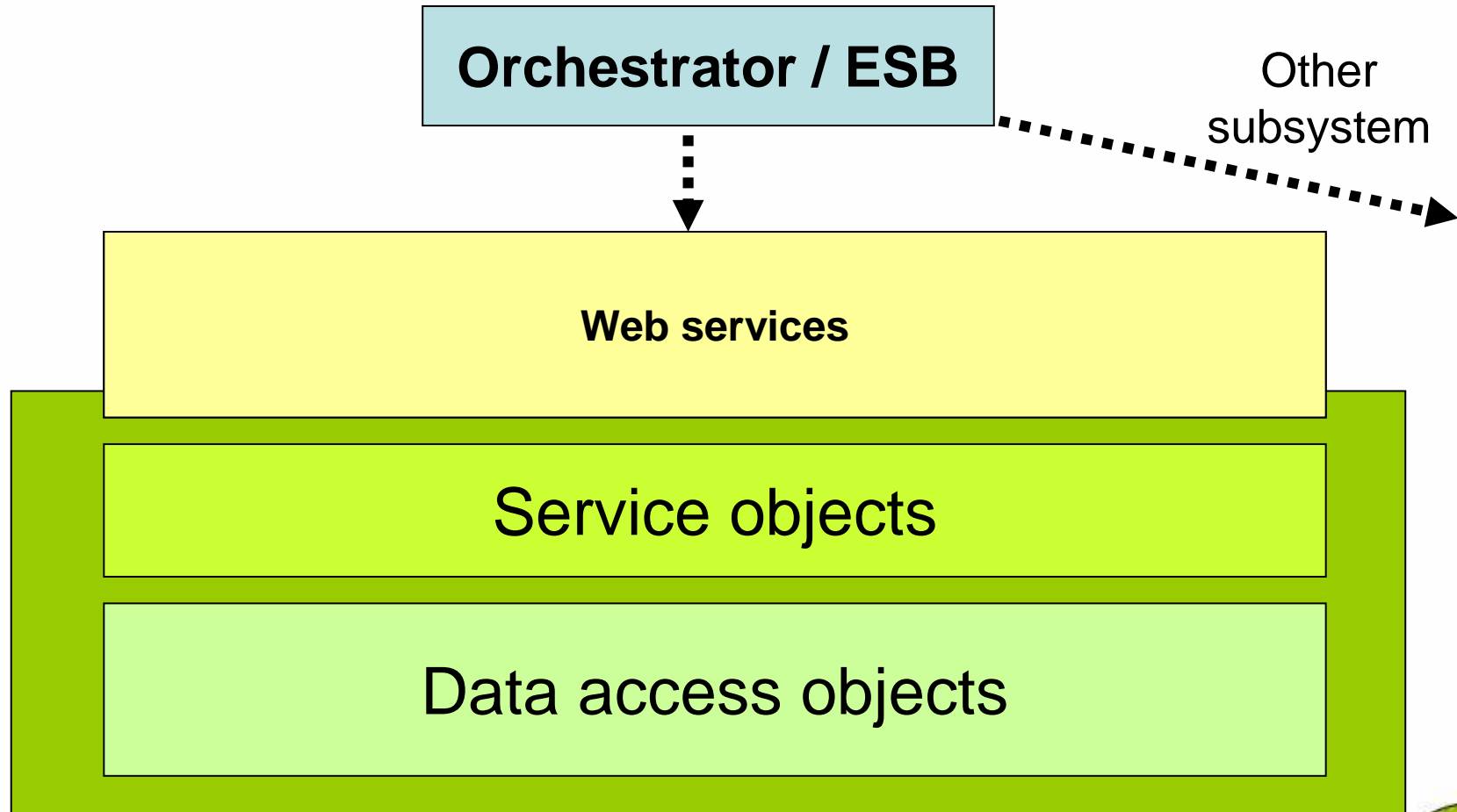
- Tight coupling to our service interface
  - ◆ Decreases interoperability, XML-Java-hassle
  - ◆ Version management is a pain
- RPC-based web service
  - ◆ Document-based is possible (message-style), but less obvious
- Contract-last development
  - ◆ Data is important, not the interface
  - ◆ Changes to interface → changes to WSDL
  - ◆ Less interoperability
- Easy to deploy simple service though
- See resources slide

# What are we looking for?

- Access to raw message
- XML marshalling
  - ◆ Optional
  - ◆ Customizable
  - ◆ Schema based
- Contract-first development

The container is nice and all, but we want something to work with, not work for...

# Web services – exposing service objects



## Develop web services on top of...



- Web services are positioned *on top of your services*
- To overcome the interop problems we've seen, create web services using:
  - ◆ A thin extra layer on top
  - ◆ Clean, simple processing of messages
  - ◆ Complete control over everything

→ Compare with Spring MVC Goals



# Design Goals Spring-WS

Spring  
*from the source*

- Make WS part of the application architecture
- Framework, not container
  - ◆ Something that clearly helps us to overcome the nasty bits of SOAP and WS
  - ◆ Without hiding the details



# Design Goals Spring-WS

- Spring-based
- Pluggability
- Focus on SOAP
- Use available implementations
- Sensible defaults
- Fully message based

# Features

- Current WS standards
- Multiple transports
- WS-\* standards
- Pluggable Object-XML Mapping
- Flexible Exception-Fault Mapping
- Integration with Java standards
- Integration with SOAP containers

→ And a strong resemblance to Spring MVC

## Other features

- SOAP version 1.1
- WSDL 1.1
- WS-I basic profile
  
- Multiple transports (at least JMS & HTTP)
  
- WS-Addressing and WS-Security  
(with the latter by using Acegi)

- O/X Mapping through:
  - ◆ Castor
  - ◆ JAXB
  - ◆ XMLBeans
- O/X Mapping Exception hierarchy
  - ◆ Uniform
  - ◆ Runtime exceptions
- `org.springframework.oxm.Marshaller`
- `org.springframework.oxm.Unmarshaller`
- `org.springframework.oxm.XmlMappingException`

# Exception handling

- Exception != Fault
- Customizable Exception-Fault mapping
  - ◆ Similar to Servlets, Spring-MVC
  - ◆ Maps classes of exceptions to faults

- SAAJ
- JAX-RPC (although not recommended)
- JAX-WS (partly, for now)

- **MessageHandler**
  - ◆ `SOAPMessage handleMessage(SOAPMessage message);`
- **PayloadEndpoint**
  - ◆ `Source invoke(Source request);`
- **Method Invoking Endpoint**
  - ◆ Any method
  - ◆ Marshalling
- Possibly a *message backing object* endpoint, similar to `SimpleFormController`
  - ◆ Binding results from XPath queries

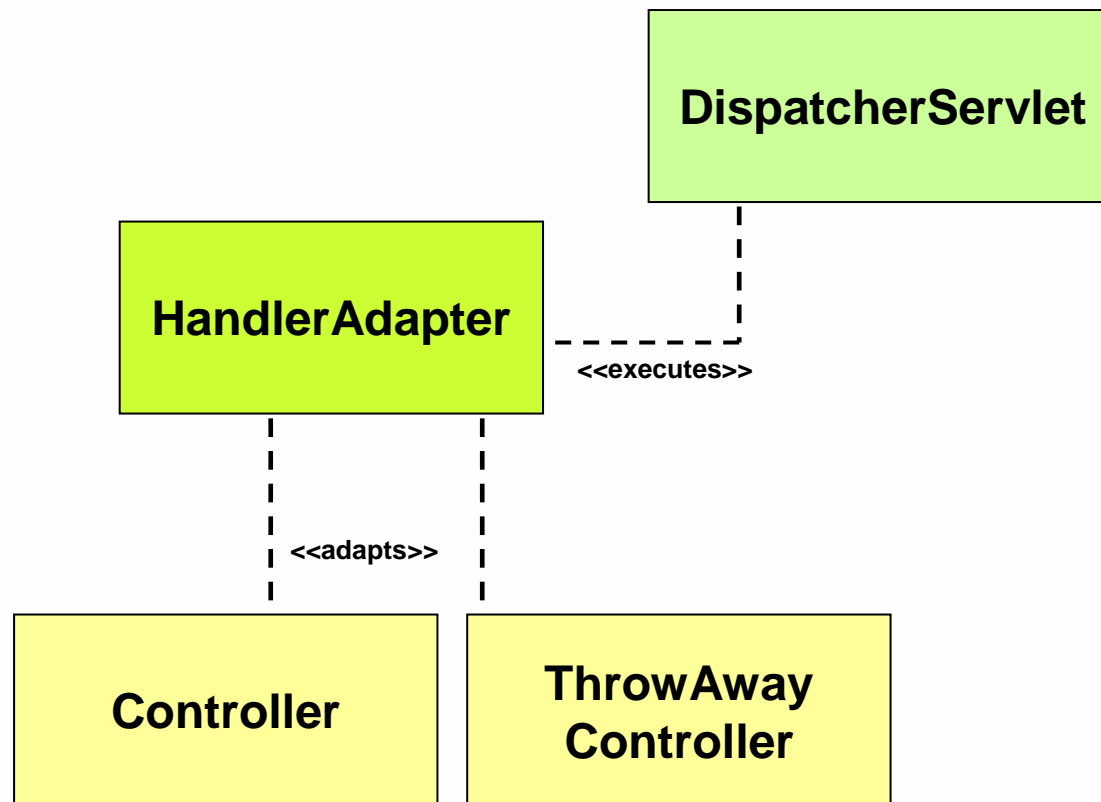
# MVC versus Web Services



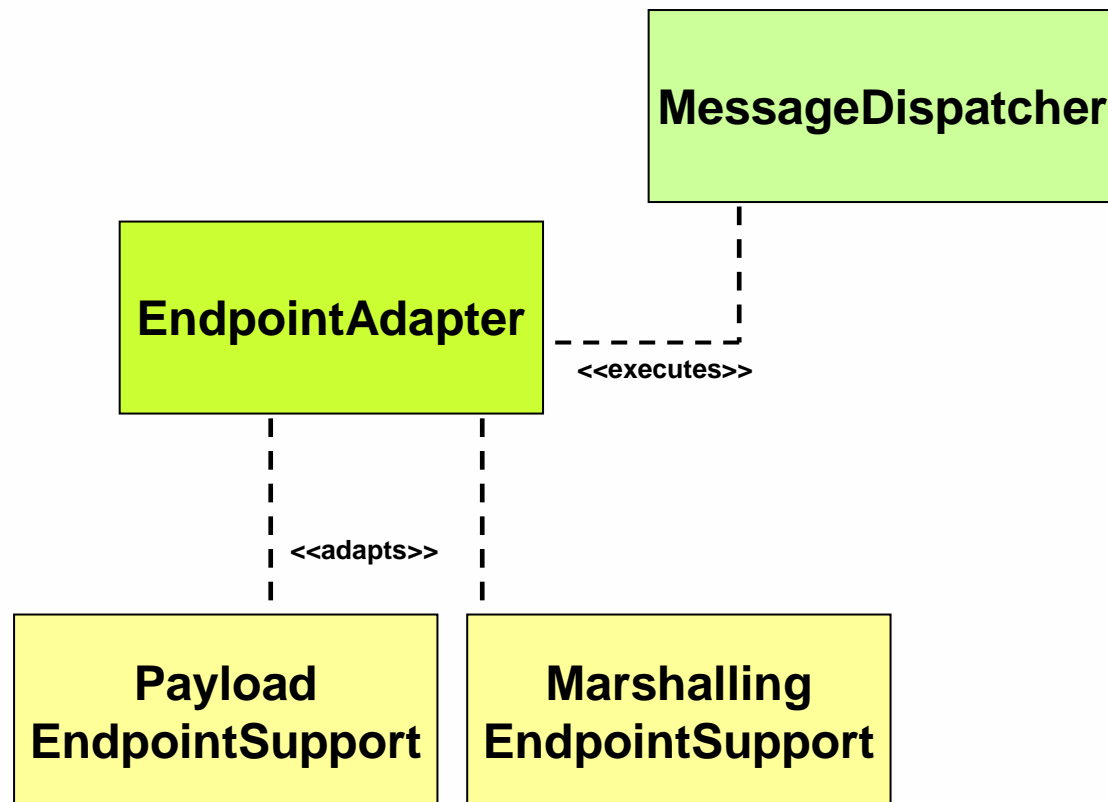
Spring MVC	versus	Spring WS
HandlerAdapter		EndpointAdapter
HandlerMapping		EndpointMapping
handler		endpoint
HandlerInterceptor		EndpointInterceptor
HandlerExceptionResolver		EndpointExceptionResolver
DispatcherServlet		MessageDispatcher



- HandlerAdapters
  - ◆ Adapt to arbitrary controller hierarchy



- EndpointAdapters
  - ◆ Adapt to arbitrary endpoint



- BeanNameUrlHandlerMapping
  - ◆ Bean names mapped to request paths
- SimpleUrlHandlerMapping
  - ◆ Properties mapped to request paths
- The criteria in this case: the request path

- PayloadRootQNameEndpointMapping
  - ◆ Qualified name of payload root mapped to web services endpoint
- MimeHeaderEndpointMapping
  - ◆ Configurable MIME header mapped to web services endpoint
  - ◆ SOAPAction (used by .NET)

# EndpointExceptionResolver

- Maps exceptions to SOAP response
  - ◆ Customizable, doesn't need to be fault
- One implementation at the moment
  - ◆ FaultMappingExceptionResolver
  - ◆ With PropertyEditor

- Implement the service (as always)
- Implement one of the endpoints
  - ◆ Marshalling endpoint
  - ◆ Endpoint with access to raw message
  - ◆ Endpoint with message backing object
- Map the endpoint with a `EndpointMapping`
- (Optionally use `EndpointInterceptors`)
- Wire using a normal `DispatcherServlet` application context

# Implementing our AirlineService with Spring WS

# Spring WS Release Schedule

Spring  
*from the source*

- First milestone early January
- 1.0 expected early Q2 2006



# Questions

Spring  
*from the source*

?



# Resources

Spring  
*from the source*

1. 'Philosophical Split Hurts Web Services Adoption' – Michael Daconta, July 2003  
[www.devchannel.org/webserviceschannel/03/07/11/2122220.shtml?tid=25&tid=38](http://www.devchannel.org/webserviceschannel/03/07/11/2122220.shtml?tid=25&tid=38)
2. 'J2EE: JAX-WS Improves JAX-RPC with better O/X Mapping' – Bobby Woolf, June 2005  
[www-128.ibm.com/developerworks/blogs/dw\\_blog\\_comments.jspa?blog=392&entry=83694](http://www-128.ibm.com/developerworks/blogs/dw_blog_comments.jspa?blog=392&entry=83694)
3. 'Rethinking the Java SOAP Stack' - Steven Loughran and Edmund Smith, June 2005  
[www.hpl.hp.com/techreports/2005/HPL-2005-83.pdf](http://www.hpl.hp.com/techreports/2005/HPL-2005-83.pdf)
4. JAX-RPC is Bad, Bad, Bad! – Richard Monson-Haefel, June 2005  
[rmh.blogs.com/weblog/2005/06/jaxrpc\\_is\\_bad\\_b.html](http://rmh.blogs.com/weblog/2005/06/jaxrpc_is_bad_b.html)
5. Practical data binding: XPath as data binding tool – Brett McLaughlin, November 2005  
<http://www-128.ibm.com/developerworks/xml/library/x-pracdb8.html>

