Spring / BlazeDS Integration

Agenda

- Spring + Flex Intro
- BlazeDS and LiveCycle Data Services Overview
- Remoting Review
- Spring BlazeDS Integration
- Demo
- Roadmap
- Q&A

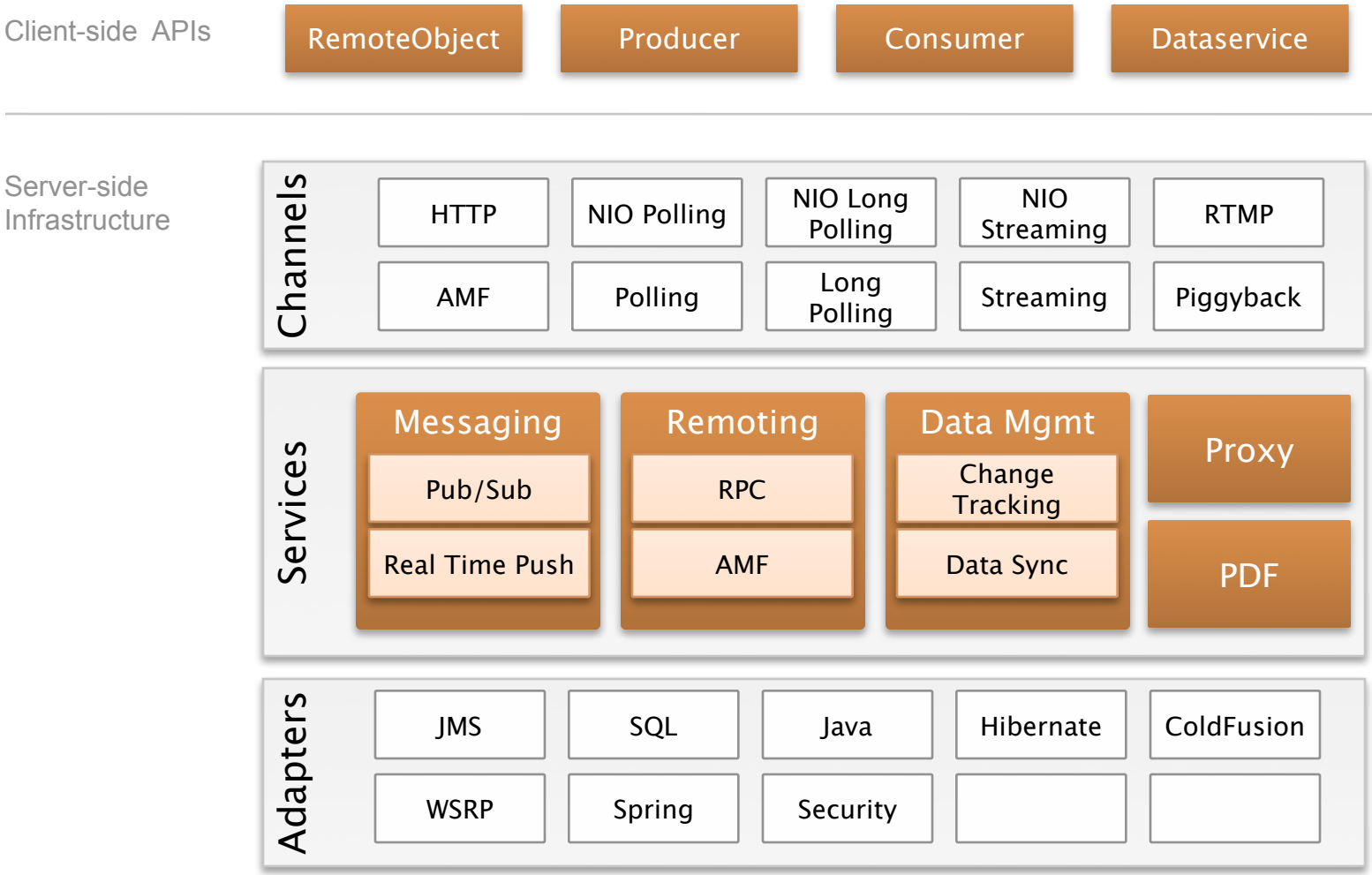
Why Flex + Spring?

- Spring has emerged as the de facto standard for the business tier of Java Enterprise applications.
- Spring aims to be agnostic to the chosen client technology.
- Flex is the obvious choice when a Spring developer is looking at RIA
- Don't have to abandon your server-side Spring investment to move into RIA

The Spring Experience

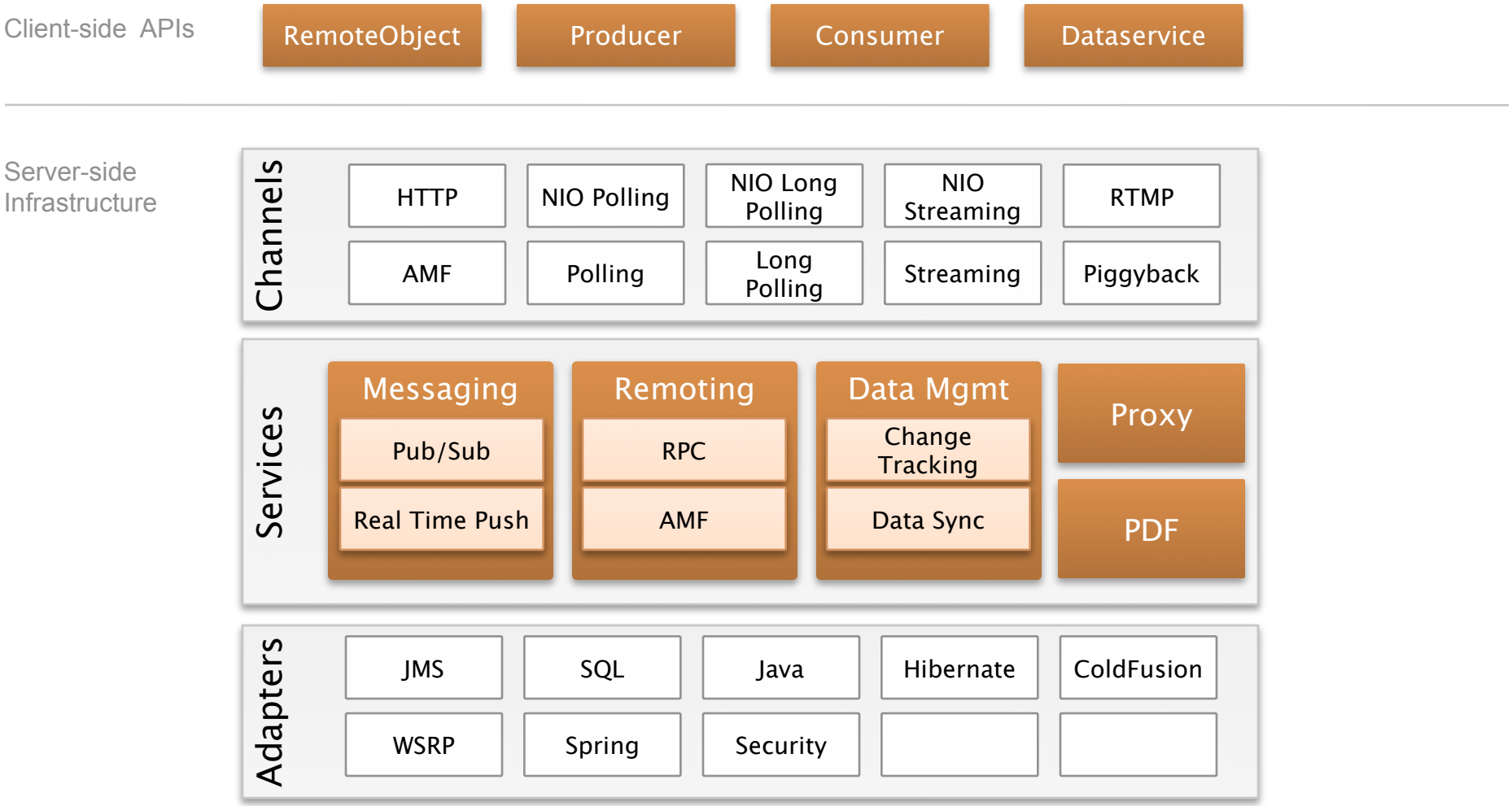
- Spring aims to ease integration of a multitude of Java technologies
- A Spring developer is attracted to the “Spring Way” of doing things
  - Common configuration approach for many disparate technologies
  - Easy transition from “simple” to “enterprise”
    - i.e., Local transactions to full-blown JTA requires no code changes

LiveCycle Data Services



Adobe

Open Source BlazeDS



Adobe

Remoting Service

- Enables client applications to remotely invoke methods of objects deployed in your application server
- Type information is maintained (see Java – ActionScript data type mapping)
- Binary encoding of data (AMF: Action Message Format)
- Benefits
  - Straightforward programming model (avoid double XML transformation)
  - Significant performance and bandwidth advantages

Remoting 101

```
<mx:RemoteObject id="srv" destination="product"/>
<mx:Button label="Get Data" click="srv.getProducts()"/>
<mx:DataGrid dataProvider="{srv.getProducts.lastResult} "/>
```

- **remoting-config.xml**

```
<destination id="product">
        <properties>
            <source>flex.samples.ProductService</source>
        </properties>
</destination>
```

Class Mapping and Serialization/Deserialization Process

- RemoteClass annotation provides class mapping information
- Without explicit mapping information, Java objects are deserialized into dynamic AS objects, and AS objects are deserialized into Java HashMaps

Product.as

```
[RemoteClass(alias="flex.samples.Product"
    )]
public class Product {
      public var id:int;
      public var name:String;
      public var description:String;
      public var price:Number;
      public var qtyInStock:int;
}
```

Product.java

```
public class Product {
    private int id;
            private String name;
            private String description;
    private double price;
            private int qtyInStock;
    //  getters and setters
}
```

Remote Objects Instantiation and Life Cycle

- By default, BlazeDS takes care of remote objects instantiation using no-arg constructor
- Scope can be **request** (default), **session**, or **application**

```
<destination id="contacts">
  <properties>
      <source>flex.samples.ProductService</source>
      <scope>application</scope>
  </properties>
</destination>
```

- Instantiation process can be delegated using factories
- In the “old” Spring/BlazeDS integration approach, a SpringFactory was used to let Spring instantiate remote objects with appropriate dependency injection

Old SpringFactory Approach

- **services-config.xml**

```
<factories>
  <factory id="spring" class="flex.samples.factories.SpringFactory"/>
</factories>
```

- **remoting-config.xml**

```
<destination id="productService">
  <properties>
      <factory>spring</factory>
      <source>productBean</source>
  </properties>
</destination>
```

Spring -> Flex

- The current path is overly complex
  - Using the “dependency lookup” approach of the SpringFactory feels antithetical to the “Spring Way”
  - The burden of configuration is multiplied
  - Potential for deep integration beyond just remoting is limited
  - Ultimately acts as a potential barrier to adoption of Flex by the Spring community

The Way Forward

- Ideally lower the barrier to adoption of Flex by the Spring community
  - Configuration using the “Spring Way”
  - Deeper integration beyond remoting
  - Make Flex the obvious and easy choice for a Spring-powered RIA
- SpringSource and Adobe have formed a joint partnership to turn this idea into reality

Open Source Foundation

- The foundations of this new integration are available as open source
  - A new Spring subproject in the web portfolio:

    Spring BlazeDS Integration

- Focus on integrating the open source BlazeDS with Spring

Spring BlazeDS Integration

- Bootstrap the BlazeDS MessageBroker as a Spring-managed bean (no more web.xml MessageBrokerServlet config needed)
- Route http-based Flex messages to the MessageBroker through the Spring DispatcherServlet
- Expose Spring beans for remoting using typical Spring remoting exporter configuration
  - Using XML namespace tags or Java annotations
  - No more remoting-config.xml

Spring BlazeDS Integration – Security

- Spring Security integration
  - Easily enabled through simple XML namespace tags
  - SpringSecurityLoginManager enables use of Spring Security for Authentication through the Flex API
    - optionally supports per-client authentication
  - Gives access to the GrantedAuthorities for conditional UI logic
  - Destinations (which are just Spring beans) are secured using existing Spring Security Authorization mechanisms
  - Endpoints can be secured by URL or Endpoint Id
  - Improved SecurityException translation

Spring BlazeDS Integration – Messaging

- Messaging integration
  - Integration with the BlazeDS MessageService
    - No more need for messaging-config.xml
  - Use Spring configuration to manage BlazeDS MessageDestinations
  - MessageTemplate provides simple server-push capabilities
  - Adapters provided for Spring JMS and Spring Integration
    - Allows easy communication from Flex clients to Spring message-driven POJOs

Spring BlazeDS Integration – Advanced Customization

- Several hooks are provided for advanced customization
  - ExceptionTranslator
  - MessageInterceptor
  - ManageableComponentFactoryBean
    - for integrating 3rd-party adapters (i.e., dpHibernate, Gilead)

Demo

Spring BlazeDS Integration – Upcoming Features (post-1.0)

- Spring 3.0 REST integration
  - Provides support for multiple client-types
  - Flex apps can already consume Spring 3.0 RESTful endpoints through HTTPService
  - Additional value could be realized by providing an AMFView implementation
    - Response for HTTP requests with a Content-Type=application/actionscript
- Hibernate serialization support
  - Make it easier to serialize Hibernate entities to AMF without LazyInitializationException, etc.

LCDS Integration

- Build on the BlazeDS Integration foundation to provide additional benefit to using LCDS with Spring

- An add-on for LCDS available as an optional module of Spring BlazeDS Integration:

  Spring Adapters for LCDS

LCDS Integration – Upcoming Features

- Enable users to easily take advantage of the advanced data synchronization features of LCDS using their existing service infrastructure
- Ensure smooth integration of existing BlazeDS Integration features with the LCDS NIO-based SocketServer

LCDS Integration – Upcoming Features

- LCDS data Assemblers configured as Spring beans
  - Enables them for declarative transaction control
- SpringHibernateAssembler that uses a Spring-managed Hibernate SessionFactory
  - equivalent assembler for JPA
- Declarative annotation-based adaptation of existing Spring-managed DAOs to the Assembler interface

Spring BlazeDS Integration – Goals

- Make the transition to using Flex for developers in the Spring community a smooth and painless experience
  - Able to migrate portions of your application gradually since the infrastructure does not change
  - Continue using the Spring programming model you've come to know and love

Summary

- Flex and Spring together provide a powerful solution for building enterprise class RIAs
- BlazeDS provides a proven open source foundation to build on for Flex->Java communication
- LCDS takes up where BlazeDS leaves off to provide a first-class supported enterprise solution
- Spring BlazeDS Integration and the Spring Adapters for LCDS make Flex the obvious and easy choice for building a Spring-powered RIA

Questions?

Adobe