Building complex and modular RIAs with OSGi™ and Flex®

francois.fornaciari@zenika.com
www.zenika.com

ApacheCon

Leading the Wave of Open Source

Content

- Introduction
- OSGi™ and Flex® interactions
- Application modularization
- Demo
- Conclusion

Bio

- Java EE / RIA consultant and trainer
  - Technical leader on the new temporary staff search engine at Manpower
- Board member of the OSGi™ Users' Group France
- Over the last 4 years
  - Participation in the refactoring of the new version of JOnAS based on OSGi™

What is Flex® (1/2)

- Framework for building RIA
  - Flash technology
  - Flash Player (VM)
  - Free Software Development Kit (SDK)
  - Strong integration with Java
  - Tooling
    - Flash Builder™ : IDE based on Eclipse
    - IntelliJ IDEA

What is Flex® (2/2)

- **MXML**: XML-based language
- **ActionScript**: ECMAScript-compliant scripting
- Library of pre-built components
- Compiler: create SWFs from MXML/ActionScript

MXML

- XML-based language
- Declarative way to build applications

```
<s:Application
   xmlns:fx="http://ns.adobe.com/mxml/2009"
   xmlns:s="library://ns.adobe.com/flex/spark"
   xmlns:mx="library://ns.adobe.com/flex/mx">
  <s:Panel>
    <s:Label text="Label" />
    <s:Button label="Button" />
  </s:Panel>
</s:Application>
```

ActionScript

- Core of the Flex® Framework
- ECMAScript-compliant scripting
- Familiar syntax

```
package com.zenika.flex {
  public class MyClass interface MyInterface {
    public function MyClass() {

    }
    public function doSomething():String {

   }
  }
}
```

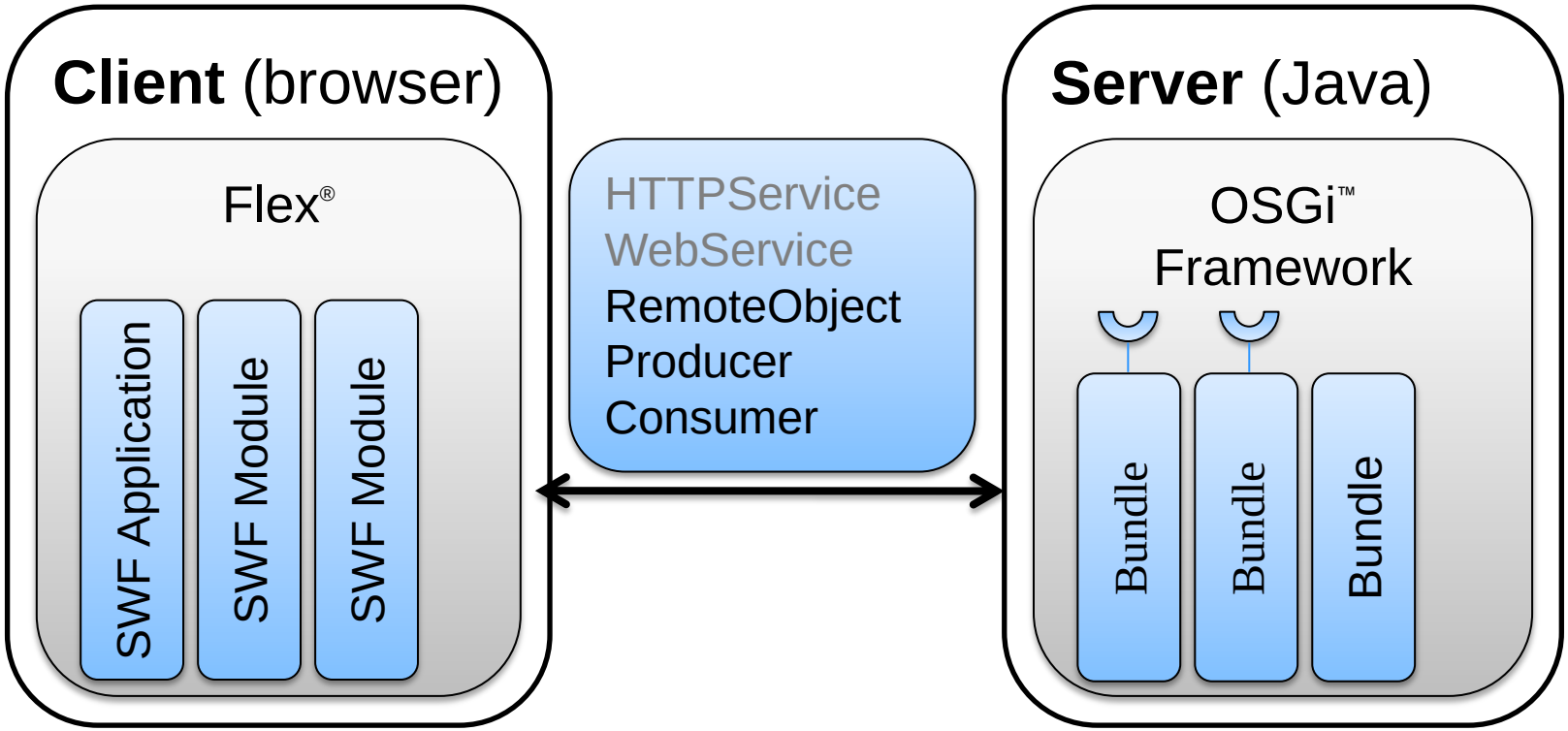
What is OSGi™

- OSGi™ framework specifications (4.2)
  - Core Specification
  - Compendium Specification
  - Enterprise Specification
- A **module system** and **service platform** for the Java programming language

Purpose

- Presentation of the main design patterns for building complex and modular applications with OSGi™ and Flex®
- Two axes
  - Communication between Flex® and OSGi™
  - Application modularization

General architecture



Handling remote calls

- HTTPService, WebService, Remote Objects, Consumer/Producer
  - Asynchronous calls
    - Result, Fault or Message handlers
  - Arguments and result types
    - Strongly typed or generic objects
    - XML format
  - Properties
    - URL, destination, endpoint, id, …

Service call example

```
private function callService():void {
  var service:MyService = new MyService(destination);
  service.addEventListener(ResultEvent.RESULT, onResult);
  service.addEventListener(FaultEvent.FAULT, onFault);
  service.send();
}

private function onResult(event:ResultEvent):void {
  var result:MyObject = event.result as MyObject;
  ...
}

private function onFault(event:FaultEvent):void {...}
```

Accessing web services (1/2)

- “Non-OSGi™” services
- Flex consumer objects
  - HTTPService
    - Supported methods: GET | POST
    - HEAD | OPTIONS | PUT | TRACE | DELETE only through the server-based proxy service
  - WebService
    - Provides access to SOAP-based web services on remote servers

Accessing web services (2/2)

- Expose OSGi™ services through the "Remote Services Specification"
  - Development of OSGi™ standard services
  - Service properties for defining
    - SOAP-based services
    - RESTful JAXRS-based endpoints
  - Implementation: Distributed OSGi™
    - Sub-project of Apache CXF

Accessing OSGi™ services

- In a transparent way
- Two communication types
  - RemoteObject
    - Simple call to an existing OSGi™ service
    - Strongly-typed arguments and result or …
    - … generic objects
  - Producer/Consumer
    - Topic subscription

AMF Protocol

- **A**ction **M**essage **F**ormat (v3)
  - Binary format used to serialize ActionScript objects
    - Optimized transferred data amount
  - Primarily used to exchange data between Adobe Flash applications and remote services
  - Specification since 2007
    - Supported by many server-side languages and technologies: Java™, .NET, PHP, …

Integration frameworks

- Existing open source frameworks

Name	OSGi™ support	RemoteObject	OSGi™ EventAdmin bridge
AMF3 for OSGi™ (LGPL)	yes	yes	yes
GraniteDS (LGPL)	yes (prototype)	yes (unstable)	no
Spring Flex (ASL)	yes (Virgo)	yes	no

- AMF3 for OSGi™ (Christopher Brind)
  - Robust, OSGi™ compliant and easy to use
    - Only 3 bundles to deploy

AMF3 for OSGi™

- OSGi™ bundle description
  - uk.co.arum.osgi.amf3
    - Serialization-deserialization of AMF3 objects
  - uk.co.arum.osgi.amf3.http
    - Registers an HTTP servlet that is capable of reading and writing AMF3 objects
  - uk.co.arum.osgi.amf3.flex.remoting
    - Implements FlexRemoting over HTTP
    - Provides support for channel based messaging

Apache Felix projects (1/2)

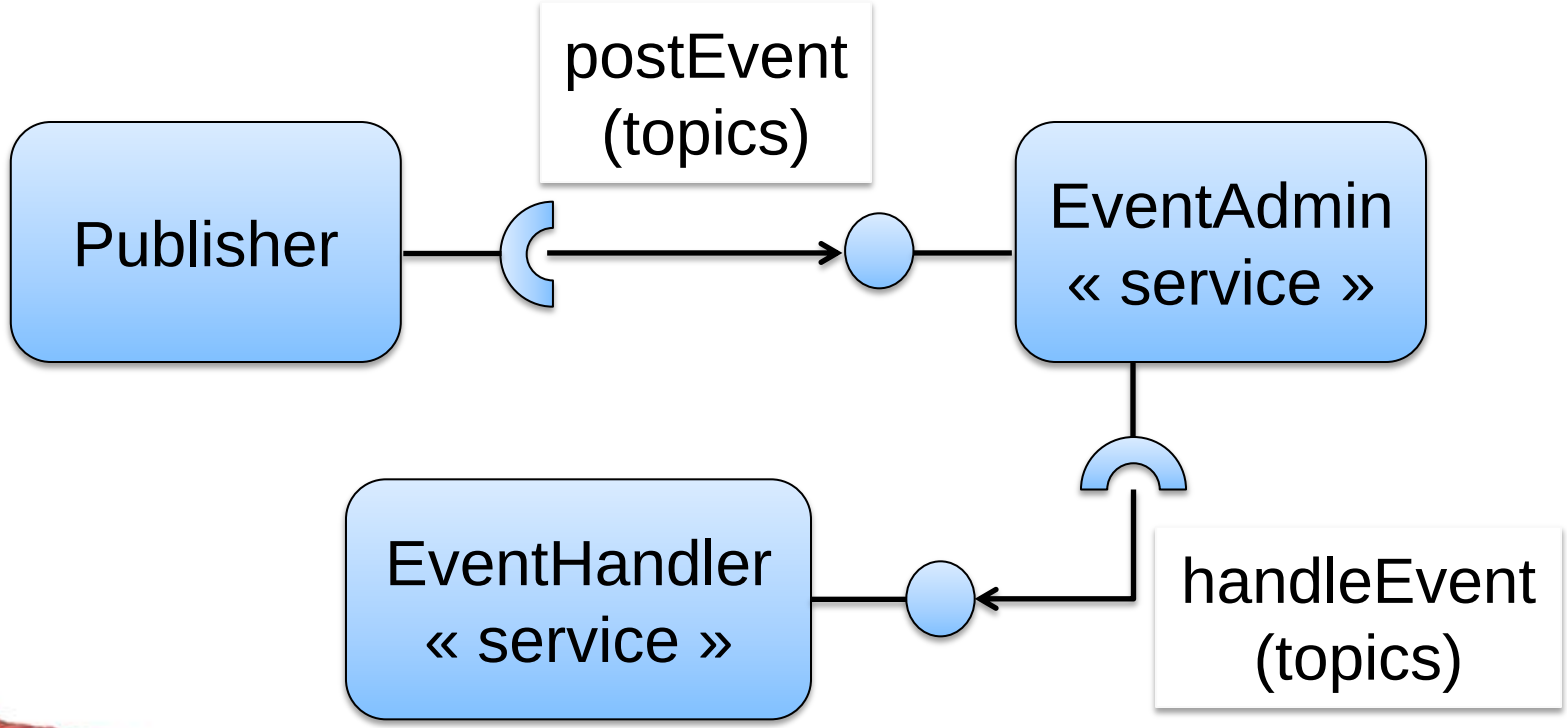
- Core of the server-side application
  - Framework
  - File Install
    - Directory based OSGi™ management agent
- Used internally by AMF3 for OSGi ™
  - Declarative Service (SCR)
  - Event Admin Service
  - Log Service
  - HTTP Service (Jetty)

Apache Felix projects (2/2)

- Application-specific projects
  - Event Admin Service
    - OSGi™ EventHandler receives messages from Flex Producers
    - Messages posted by the OSGi™ EventAdmin service are received by Flex Consumers
  - iPOJO
    - Used to declare OSGi™ components
    - Support of annotations
    - Extensible handlers

Event Admin

- Based on the *Publish-Subscribe* pattern



Publishing OSGi™ services

- Additional service property
  - AMF_SERVICE_NAME
    - Instantiated with the service class name

```
@Component(name="MyService")
@Provides
public class MyServiceImpl implements MyService {

  @ServiceProperty(name="AMF_SERVICE_NAME")
  private String serviceName = MyService.class.getName();

  ...
}
```

Consuming OSGi™ services

- Flex RemoteObject
  - Endpoint: AMF3 servlet path

    ex: http://localhost:8080/amf3osgi
  - Destination: OSGi™ service class name

```
<s:RemoteObject id="myService"
                endpoint="/amf3osgi"
                destination="com.zenika.service.MyService">
   <s:method name="doSometing"
             result="onResult(event)"
             fault="onFault(event)" />
</s:RemoteObject>
```

Producer / Consumer (1/2)

- Bridge between Flex® events and EventAdmin service
- Elements attributes
  - Consumer: callback when a message is received
  - Consumer and producer: destination

```
<s:Consumer message="onMessage(event)"
            destination="events"
            fault="onFault(event)" />
<s:Producer destination="events"
            fault="onFault(event)" />
```

Producer / Consumer (2/2)

- Destination configuration
  - Set of channels used to send messages to a target destination
    - QoS: many channels for network failure tolerance

```
var channelSet:ChannelSet = new ChannelSet();
var channel:AMFChannel =
                new AMFChannel("events","/amf3osgi");
channel.pollingInterval = 5000;
channelSet.addChannel(channel);

consumer.channelSet = channelSet;
```

Sending messages (1/2)

- From OSGi™
  - PublishedObjectEvent (channelID, object)
  - Extends org.osgi.service.event.Event

```
@Bind
public void bindEventAdmin(EventAdmin eventAdmin) {
  this.eventAdmin = eventAdmin;
}

private void sendMessage() {
  eventAdmin.postEvent(
    new PublishedObjectEvent("events", object));
}
```

Sending messages (2/2)

- From Flex®
  - Sending AsyncMessage through the producer
    - Body: object to send

```
var message:AsyncMessage = new AsyncMessage();
message.body = messageObject;
producer.send(message);
```

Receiving message (1/2)

```
@Component(name="FlexEventHandler")
@Provides
public class FlexEventHandler implements EventHandler {
  @ServiceProperty(name="event.topics")
  private String eventTopics =
   ".../amf3/flex/remoting/events/PublishedObjectEvent";

  @ServiceProperty(name="event.filter")
  private String eventFilter = "(channel.id=events)";

  public void handleEvent(Event event) {
    if (event instanceof PublishedObjectEvent) { ...}
  }
}
```

Receiving message (2/2)

- Start / cancel subscription
- Message: typed-object

```
private function start():void {
    consumer.subscribe();
}
private function stop():void {
    consumer.unsubscribe();
}
private function onMessage(event:MessageEvent):void {
  var msg:MyMessage = event.message.body as MyMessage;
  ...
}
```

Flex® modulary

- Flex® modules
  - SWF files that can be loaded and unloaded at runtime by an application
    - Free up memory and resources if needed
  - Better encapsulation
  - No need to load all modules at startup
    - Smaller initial download size
  - Cannot be run independently

Main goal

- OSGi™ is the Java modularity
- Flex® supports modules

- How can we build a complete modular application?

Creating modules

- “Module” as root element

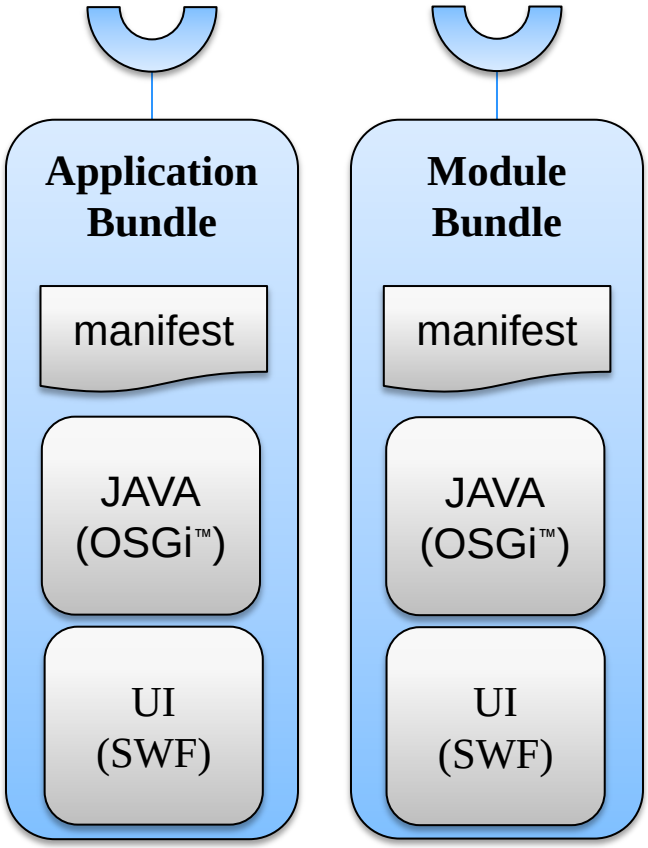
```
<mx:Module xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    implements="...api.ModuleInterface"
    creationComplete="start()">

  <fx:Script>
    <![CDATA[
      public function start():void {...}
      public function stop():void {...}
    ]]>
  </fx:Script>

  <s:Label text="Label" />
</mx:Module>
```

Module lifecycle

- Start method is called when the module is loaded
- Stop method is called before unloading the module
  - Common interface between the application and the module

```
package com.zenika.systemmanager.api {
  public interface ModuleInterface {
    function stop():void;
  }
}
```

Packaging



- Each bundle contains
  - Business logic
    - OSGi™ services
  - User interface
    - SWF file
- Building tools like Maven, Ant or Gradle can ease the packaging phase

Application bundle

- User Interface skeleton
- Able to retrieve list of available modules
  - OSGi™ service for getting module URLs
- Flex® client notified when a module appears or disappears
  - Flex consumer is aware of events posted by the EventAdmin

Technical view (1/4)

- At startup, registers bundle resources by using the HTTP Service
  - Main SWF, HTML files, …
- Gets registered modules and listens to bundle arrival / departure
  - Implementation of the extender pattern

Technical view (2/4)

- Extender pattern
  - Use of a BundleTracker in order to be notified of ACTIVE bundles declaring Flex® modules
    - MANIFEST.MF :
      - SWF-Modules: [path]=[SWF name]
  - Register / unregister SWFs into path
    - Creation of a resource-aware context
  - Post event to notify Flex® client

Technical view (3/4)

```
var modules:ArrayCollection = event.result as
ArrayCollection;

for each(var module:SWFModule in modules) {
  var loader:ModuleLoader = new ModuleLoader(module.url);

  // Register event listeners
  loader.addEventListener(ModuleEvent.READY,
                          handleModuleReady);
  loader.addEventListener(ModuleEvent.ERROR,
                          handleModuleError);

  // Start the module loading
  loader.load();
}
```

Technical view (4/4)

- After bundle undeployment
  - Call the stop method
    - Stop properly on-going processes
    - Warning: OSGi™ services have already been unregistered
  - Unload associated modules
    - Free-up memory and resources

Module bundle

- Autonomous module
  - Contains its own UI and logic
  - Good isolation
  - Flex® modules have to be based upon their own OSGi™ services
    - Avoid using unregistered services or managing service dynamicity on the client-side

Demo

Conclusion (1/2)

- Flex®
  - Development of rich interfaces
  - Native support of modules
    - Loading / unloading modules at runtime
- OSGi™
  - The Dynamic Module System for Java™
  - Service Oriented Architecture
    - Service availability to Flex®

Conclusion (2/2)

- Seamless communications
  - Asynchronous remote calls
  - Notifications
- Integration
  - Application: set of autonomous bundles with their own interface and business logic

References

- OSGi™ : www.osgi.org
- Apache Felix: felix.apache.org
- Adobe Flex® : www.adobe.com/fr/products/flex
- AMF3 for OSGi™ : www.arum.co.uk/amf3osgi.php

Questions

francois.fornaciari@zenika.com
www.zenika.com