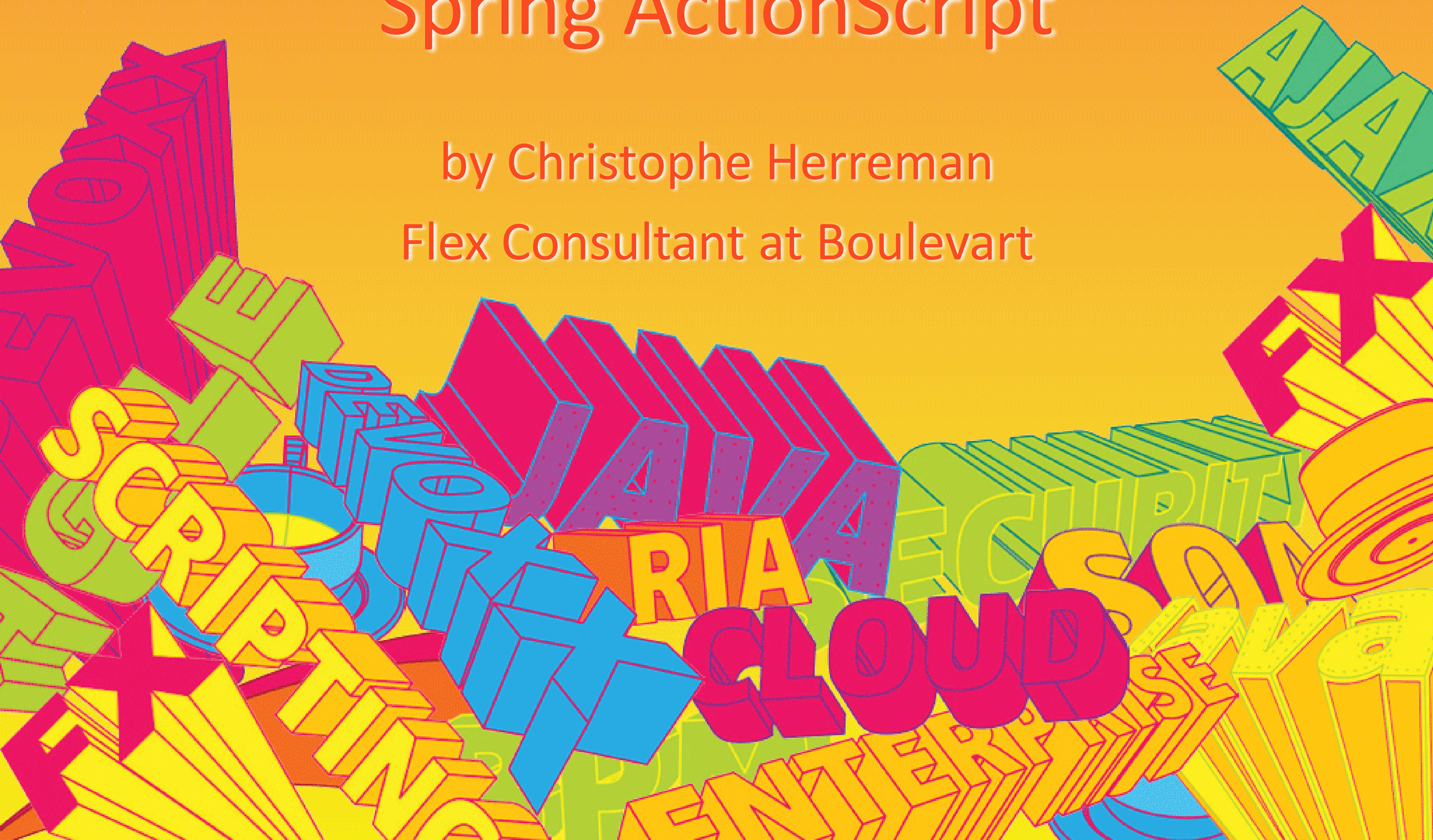
Spring ActionScript

by Christophe Herreman

Flex Consultant at Boulevart



Overall presentation

Introduction to Spring ActionScript:

what it does and

how you benefit from using it

www.devoxx.com

Agenda

- What is Spring ActionScript
- The IoC container
- Configuration
- MVC Architecture
- Demo
- Q&A

www.devoxx.com

Speaker's qualifications

- Certified Adobe Flex & AIR expert
- 10 years playing with Flash technology
- Founder and lead developer of AS3Commons & Spring ActionScript
- Tech reviewer for FoED/Apress

What is Spring ActionScript (1/2)

- Inversion of Control (IoC) for ActionScript 3.0
- Flash/Flex/AIR/Pure AS3
- IoC container
- MVC?

What is Spring ActionScript (2/2)

- Started as an in-house library
- Formerly known as the Prana framework
- Incubated as a Spring Extension project
- Upcoming release 0.9
- AS3Commons: Lang, Logging, Reflect, …

The IoC Container (1/3)

- = Object Factory
- Creates and assembles objects
- Centralized dependency management
- Spring AS: configuration via XML or MXML

www.devoxx.com

The IoC Container (2/3)

- Object (bean): object managed and/or created by the container
- Object Factory: factory that creates and manages objects
  - factory.getObject("myObject")
- Object Definition: blueprint for an object
- Application Context: smarter Object Factory

www.devoxx.com

The IoC Container (3/3)

- Object Scopes
  - Singleton: only one instance in the container
    - Not linked to the Singleton Design Pattern
    - The default scope
    - factory.getObject("obj") == factory.getObject("obj")
  - Prototype: new instance created on each request
    - factory.getObject("obj") != factory.getObject("obj")

www.devoxx.com

Configuration (1/9)

MXML Configuration

```
// AppContext.mxml file, compiled into the application

<Objects>

    <app:ApplicationModel id="appModel" />

    <app:ApplicationController id="appController"
     applicationModel="{appModel}"/>

</Objects>
```

Configuration (2/9)

MXML Configuration: loading

```
var context:MXMLApplicationContext = new MXMLApplicationContext();
context.addConfig(AppContext);
context.addEventListener(Event.COMPLETE, context_completeHandler);
context.load();

function context_completeHandler(event:Event):void {
    var appModel:ApplicationModel = context.getObject("appModel");
    var appController:ApplicationController =
     context.getObject("appController");
}
```

Configuration (3/9)

MXML Configuration: alternative approach

```
// AppContext.mxml file, compiled into the application

<Objects>

    <Object id="appModel" clazz="{ApplicationModel}"/>

    <Object id="appController" clazz="{ApplicationController}">
         <Property name="applicationModel" ref="appModel"/>
    </Object>

</objects>
```

Configuration (4/9)

MXML Configuration: pros

- Editor support
- Simple to use

Configuration (5/9)

MXML Configuration: cons

- Compiled into the application
- Explicit declaration: limited to singletons, no external references, no prototypes -> use Spring AS MXML dialect

www.devoxx.com

Configuration (6/9)

XML Configuration

```
// external context.xml, loaded on application startup

<objects>

    <object id="appModel" class="com.domain.app.ApplicationModel"/>

    <object id="appController" class="com.domain.app.ApplicationController">
          <property name="applicationModel" ref="appModel"/>
    </object>

</objects>
```

www.devoxx.com

Configuration (7/9)

XML Configuration: loading

```
var context:XMLApplicationContext = new XMLApplicationContext();
context.addConfigLocation("context.xml");
context.addEventListener(Event.COMPLETE, context_completeHandler);
context.load();

function context_completeHandler(event:Event):void {
   var appModel:ApplicationModel = context.getObject("appModel");
   var appController:ApplicationController =
    context.getObject("appController");
}
```

Configuration (8/9)

XML Configuration: pros

- Richer dialect than MXML config
- Usable for non-Flex projects
- No need to recompile (in some cases)
- Familiar to Spring Java users

Configuration (9/9)

XML Configuration: cons

- Classes need to be compiled into the app, no class loading (!)
- No editor support, only XSD

MVC Architecture (1/2)

- No real prescriptive architecture
- Do you really need one?
- Roll your own because one does NOT fit all
- Many MVC architectures out there:
  - Cairngorm, PureMVC, Mate, …
  - … Spring AS wants to support them

MVC Architecture (2/2)

- Spring AS MVC ingredients
  - Operation API
  - Event Bus
  - Autowiring
- Recommendations
  - Presentation Model (Fowler)
  - Layered Architecture (DDD – Evans)

The Operation API (1/7)

- Asynchronous behavior in Flash Player: loading external resources, RMI, sqlite, Webservices, HTTPServices, …
- No threading API
- Many different APIs: AsyncToken, Responders, Callbacks, Events… we need a unified approach!

www.devoxx.com

The Operation API (2/7)

- Problem: a user repository (or service)

```
interface IUserRepository {
  function getUser(id:int): ???
}
```

- What does the getUser method return? AsyncToken (Flex/RemoteObject), User (In memory), void (events)

www.devoxx.com

The Operation API (3/7)

- Spring AS solution: return a IOperation

```
interface IUserRepository {
  function getUser(id:int):IOperation;
}
```

- Can now be used in any ActionScript 3 context and easily mocked for testing

www.devoxx.com

The Operation API (4/7)

- In Spring AS, an "operation" is used to indicate an asynchronous execution
- `IOperation` interface with "complete" and "error" events
- `IProgressOperation` with "progress" event
- `OperationQueue` bundles operations (a composite operation)

www.devoxx.com

The Operation API (5/7)

Defining an operation:

```
public class GetUserOperation extends AbstractOperation {

    function GetUserOperation(remoteObject:RemoteObject, id:String) {
        var token:AsyncToken = remoteObject.getUser(id);
        token.addResponder(new Responder(resultHandler, faultHandler));
    }

    private function resultHandler(event:ResultEvent):void {
        dispatchCompleteEvent(User(event.result));
    }

    private function faultHandler(event:FaultEvent):void {
        dispatchErrorEvent(event.fault);
    }
}
```

www.devoxx.com

The Operation API (6/7)

Usage:

```
var operation:IOperation = new GetUserOperation(remoteObject, 13);

operation.addCompleteListener(function(event:OperationEvent):void {
   var user:User = event.result;
});

operation.addErrorListener(function(event:OperationEvent):void {
   Alert.show(event.error, "Error");
});
```

The Operation API (7/7)

A progress operation:

```
var operation:IOperation = new SomeProgressOperation(param1, param2);

operation.addCompleteListener(function(event:OperationEvent):void {
});

operation.addErrorListener(function(event:OperationEvent):void {
});

operation.addProgressListener(function(event:OperationEvent):void {
    var op:IProgressOperation = IProgressOperation(event.operation);
    trace("Progress:" + op.progress + ", total: " + op.total);
});
```

The Event Bus (1/3)

- Publish/subscribe event system
- Used for global application events
- Promotes loose coupling
- Works with standard Flash Events

The Event Bus (2/3)

Listening/subscribing to events:

```
// listen for all events
EventBus.addListener(listener:IEventBusListener);

function onEvent(event:Event):void {
}

// listen for specific events
EventBus.addEventListener("anEvent", handler);

function handler(event:Event):void {
}
```

www.devoxx.com

The Event Bus (3/3)

Dispatching/publishing events:

```
// dispatch standard event
EventBus.dispatchEvent(new Event("someEvent"));

// dispatch custom event
class UserEvent extends Event {
    ...
}

EventBus.dispatchEvent(new UserEvent(UserEvent.DELETE, user));
```

www.devoxx.com

Autowiring (1/2)

- Auto Dependency Injection via metadata
- Autowire by type, name, constructor, autodetect
- Works for objects managed by the container and for view components
- Be careful: magic happens, no explicit configuration

Autowiring (2/2)

- Annotate a property with [Autowired]

```
class UserController {

  [Autowired]
  public var userRepository:IUserRepository;
}

[Autowired(name="myObject", property="prop")]
```

www.devoxx.com

Summary

- Dependency Management
- Loose coupling, type to interfaces
- Use with other frameworks
- Spring mentality: provide choice
- Promote best practices

www.devoxx.com

BLENDED FOR THE JAVA COMMUNITY

DEMO

A sample application

Thanks for your attention!

- www.herrodius.com

- info@herrodius.com

- www.springactionscript.org

- www.as3commons.org

www.devoxx.com