cfunited

the premier coldfusion technical conference

June 18 - 21, 2008
Washington DC

Event-Driven Programming in ColdFusion

Sean Corfield
Chief Systems Architect & VP Engineering
Broadchoice, Inc.

What is he on about?

What is he on about?

- We're used to simple method calls

What is he on about?

- We're used to simple method calls
- We're used to responding to clicks (link/submit)

What is he on about?

- We're used to simple method calls
- We're used to responding to clicks (link/submit)
- Richer user interfaces require us to respond to more interactions in less structured workflows

What is he on about?

- We're used to simple method calls
- We're used to responding to clicks (link/submit)
- Richer user interfaces require us to respond to more interactions in less structured workflows
- Flex allows for very rich user experiences based on an event model and asynchronous execution

What is he on about?

- We're used to simple method calls
- We're used to responding to clicks (link/submit)
- Richer user interfaces require us to respond to more interactions in less structured workflows
- Flex allows for very rich user experiences based on an event model and asynchronous execution
- Asynchronous event handling is pretty alien to most ColdFusion developers...

What is he on about?

- We're used to simple method calls
- We're used to responding to clicks (link/submit)
- Richer user interfaces require us to respond to more interactions in less structured workflows
- Flex allows for very rich user experiences based on an event model and asynchronous execution
- Asynchronous event handling is pretty alien to most ColdFusion developers...
- ...so let's try to get familiar with it!

Who is this man?

- Chief Systems Architect / VP Engineering @ Broadchoice, Inc.
- Adobe Community Expert for ColdFusion
- Manager of Bay Area ColdFusion User Group
- Formerly Senior Architect for Macromedia IT
- ColdFusion developer since 2001 ("Neo")
- Previously Java developer (since 1997) and C++ developer (since 1992)

Agenda

- Event-Driven Architecture & Lifecycle
- Common Event Handling Techniques
  - Configuration
  - Convention
  - Code (Dynamic)
- Doing more with events in ColdFusion
- Ready for Flex?

Event-Driven Architecture (I)

- "a software architecture pattern promoting the production, detection, consumption of and reaction to events"
  - wikipedia.org
- event: "a significant change in state"
  - wikipedia.org

Event-Driven Architecture (II)

- Event Model:
  - Detects events from producers
  - Dispatches events to consumers that have registered an interest
- **Goal: decouple components**

Event-Driven Lifecycle

Event-Driven Lifecycle

- Components are registered as event handlers
  - They express interest in specific events

Event-Driven Lifecycle

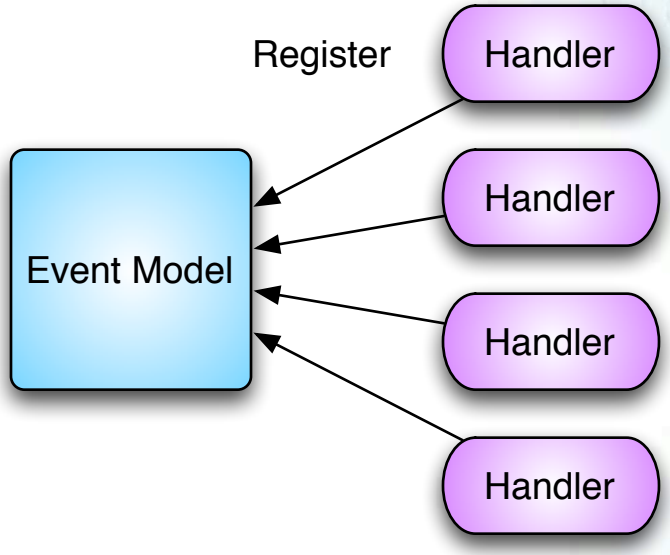
- Components are registered as event handlers
  - They express interest in specific events
- Events enter the system
  - From external sources
  - As a result of internal operations

Event-Driven Lifecycle

- Components are registered as event handlers
  - They express interest in specific events
- Events enter the system
  - From external sources
  - As a result of internal operations
- Events are dispatched to interested handlers
  - The handlers registered for a given event (or event type) are executed, synchronously or asynchronously

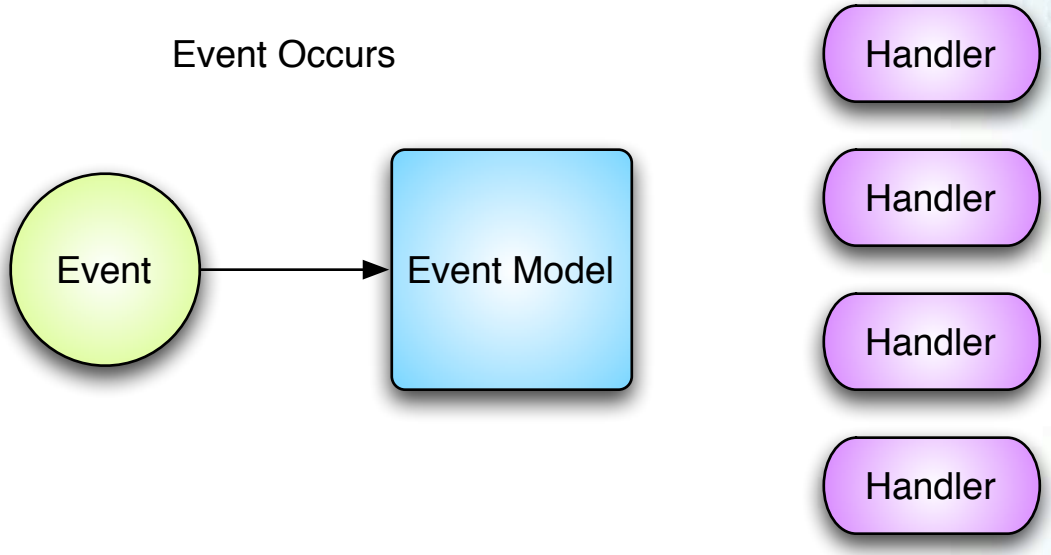
Event-Driven Lifecycle (I)

- Event Registration



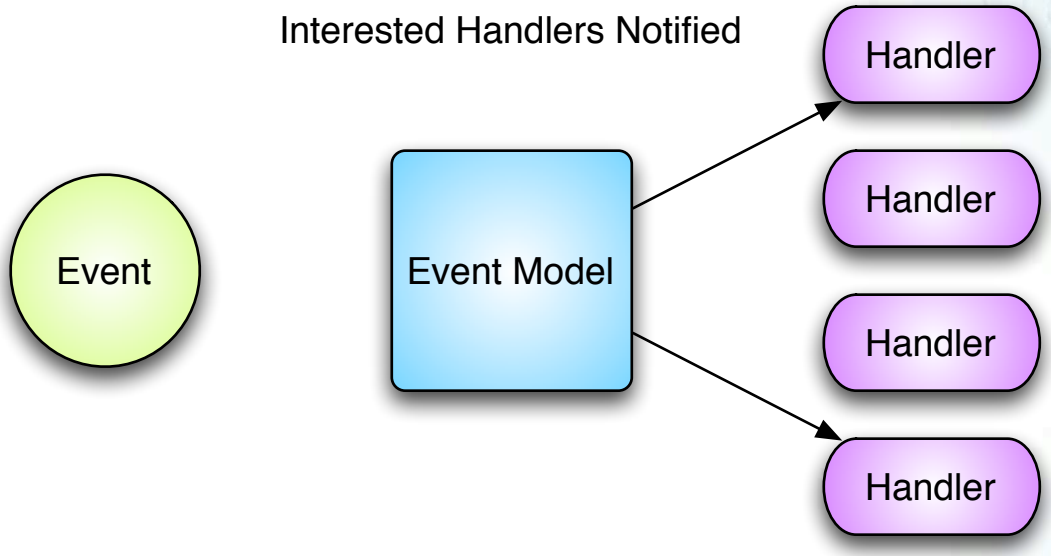
Event-Driven Lifecycle (II)

- Event Arrival



Event-Driven Lifecycle (III)

- Event Dispatching



Event Handling by Configuration

- Explicit declaration of handlers for events
  - Mach-II (XML)
  - Model-Glue (XML)
  - HTML/JavaScript (tag attributes)
  - Flex (tag attributes)

Mach-II

- Event handlers, listeners and subscribers declared in XML
- Event handlers can notify listeners and publish messages
- Event handlers also set up event data, event mappings, render views and announce events
- Events can be queued
- <notify> is really a direct method call
- Publish / subscribe can be asynchronous (1.6)

Mach-II (continued)

```
<event-handler event="customer.add.form">
    <notify listener="user" method="fetchUser"/>
    <event-arg name="xe.submit"
                        value="customer.add.submit"/>
    <view-page name="customer.add.form"
                        contentArgs="body"/>
    <announce event="main.layout"/>
</event-handler>
```

Model-Glue

- Event handlers and message listeners declared in XML
- Event handlers can broadcast messages, render views and announce events
- Events can be queued (but rarely are)
- Listeners can be executed asynchronously (rarely used functionality)

Model-Glue (continued)

```
<controller name="clipboard"
                    bean="ClipboardController">
   <message-listener
                    message="needClipboardData"
                    function="loadClipboardData"/>
</controller>
```

Model-Glue (continued)

```
<event-handler name="clipboard.main">
    <broadcasts>
        <message name="needClipboardData"/>
    </broadcasts>
    ...
    <results>
        <result do="view.template"/>
    </results>
</event-handler>
```

HTML/JavaScript

- A specific set of events are available
- Event handlers are usually declared with on{event} attributes on HTML elements
- Event handlers are JavaScript code

```
<p id="clickable" onclick="doSomething();">This paragraph of text is clickable.</p>
```

- JavaScript frameworks allow more sophistication

Flex

- Like HTML/JavaScript, some specific events are available declaratively

  ```
  <mx:Button label="Update"
  click="admin.getProviders()"/>
  ```

- Allows declarative binding between components

  ```
  <mx:ComboBox id="provider"
  dataProvider="{providers}" />
  ```

- This is an implicit event handler - the ComboBox listens for updates on the providers object

Event Handling by Convention

- Implicit association of handlers with events
  - ColdBox
  - Fusebox 5.5 (no-XML)
- Event handlers are methods on CFCs that are auto-discovered via naming and location conventions
  - user.list -> user.cfc and list() method
- See also Ruby on Rails, Grails etc

Event Handling Dynamically

- Association of handlers and events at runtime
  - AJAX
  - Flex
- AJAX is fairly limited

```
element.onfocus = someFunction;
```

- Flex is much more flexible(!)
  - Can define new event types

Flex

- Creating a new event

```
import flash.events.Event;
public class MyEvent extends Event {
    public static const EVENT_TYPE:String = "...";
    ...
}
```

- Can add "smart" events with data and behavior

Flex (continued)

- Registering a listener for an event

```
component.addEventListener(
    MyEvent.EVENT_TYPE,
    someListenerFunction
);
```

- Can add many listeners for same event type

Flex (continued)

- Announcing an event

  dispatchEvent(new MyEvent(...))

- Can announce any event containing any data

Back to ColdFusion

- The application frameworks get us part of the way there:
  - We can declare event listeners in XML (but not in code, nor dynamically add listeners)
  - We can announce new events dynamically
  - Events can contain arbitrary data
  - Event handling is synchronous (mostly)

Edmund

- Event-Driven Model framework focused on event handling with asynchronous execution

- Event handlers
  - Can be declared in XML
  - Can be registered dynamically
  - Can be synchronous or asynchronous

Edmund (continued)

- Registering a listener for an event

  edmund.register("eventName",someObj);

- someObj.handleEvent(event) is called, event.name() is "eventName"

Edmund (continued)

- Can specify additional arguments

```
edmund.register(
    eventName - string,
    object - any component,
    methodName - string - default "handleEvent",
    isAsynchronous - boolean - default false
);
```

Edmund (continued)

- Announcing an event

```
edmund.dispatch("eventName");

anEvent = edmund.new().name("anotherEvent");
edmund.dispatchEvent(anEvent);

edmund.new().name("evt").dispatch();
```

Edmund (continued)

- Can specify data for events

```
e = edmund.new().name("person")
                .values(firstName="Sean",age=45);
e.value("lastName","Corfield");

age = e.value("age");
```

Edmund (continued)

- Creating custom events

```
<cfcomponent
        extends="edmund.framework.Event">
</cfcomponent>
```

```
e = edmund.new("events.MyEvent");
```

e.name() is "MyEvent" based on the component

Code Example

- Show VistorLogin asynchronous event example

Edmund speaks Mach-II (sort of)

- Can use "Mach-II" XML
  - Declare <listeners>, <message-subscribers>
  - Declare <event-handlers>
  - Event-handlers contain <notify>, <publish>
- Other verbs are ignored
  - May support <event-arg>, <event-mapping> and <announce> in future

Edmund speaks Model-Glue (sort of)

- Can use "Model-Glue" XML
  - Declare <controllers>, including async="true"
  - Declare <event-handlers>
  - Event-handlers contain <broadcasts>
- Other sections are ignored
  - May support <value> (under <message>) and <results> in future

Code Example

- Show edmund.xml with Mach-II and Model-Glue style event handlers, listeners and subscribers

Edmund Workflow

- Has workflow components that implement handleEvent() method
  - decision(condition,ifTrue[,else])
  - foreach(condition,body)
  - seq(command[])
  - ...
- This makes it easy to build complex event handlers declaratively (using ColdSpring)

Edmund Demo

- Show me the code!

Summary

- Event-Driven Programming allows us to create flexible systems that are very loosely coupled
- We can apply the same approach in ColdFusion development as we see in Flex's client-side
- This helps us get more familiar with the event-based idioms required to develop Flex apps

Resources

- http://en.wikipedia.org/wiki/Event_Driven_Architecture
- http://edmund.riaforge.org
- http://corfield.org/cat/edmund
- http://developer.adobe.com/flex

Q & A

- Sean Corfield

- sean@corfield.org
- http://corfield.org

- scorfield@broadchoice.com
- http://broadchoice.com