

Event-Driven Programming in ColdFusion

Sean Corfield

November 19th, 2008



- A ColdFusion developer...
- ...who either:
 - wants to learn Flex
 - but thinks the event-driven style is too hard
 - already uses Flex
 - and wants to use that same event-driven style in CFML

- 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)
- Recently started doing Flex / AIR / Groovy development

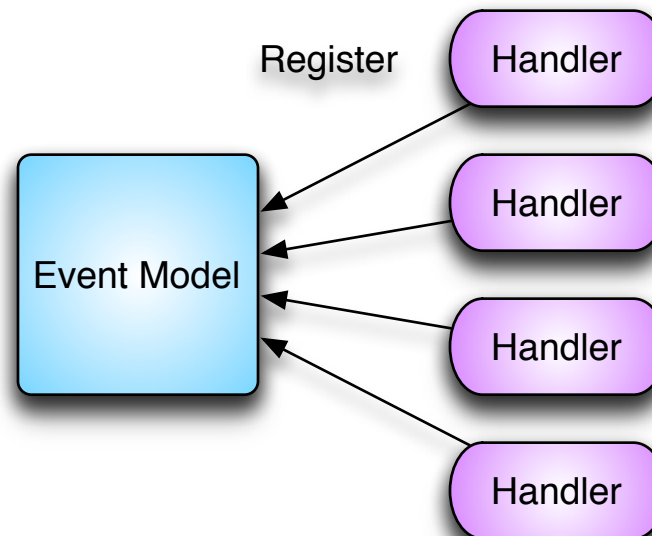
- Introduction to event-driven architecture
- External events vs. internal events
- The "Flex style" of event-driven programming
- ColdFusion and "external events" - what we can do today
- ColdFusion and "internal events" - what we can do today
- ColdFusion and "internal events" - what Flex can do

Event-Driven Architecture

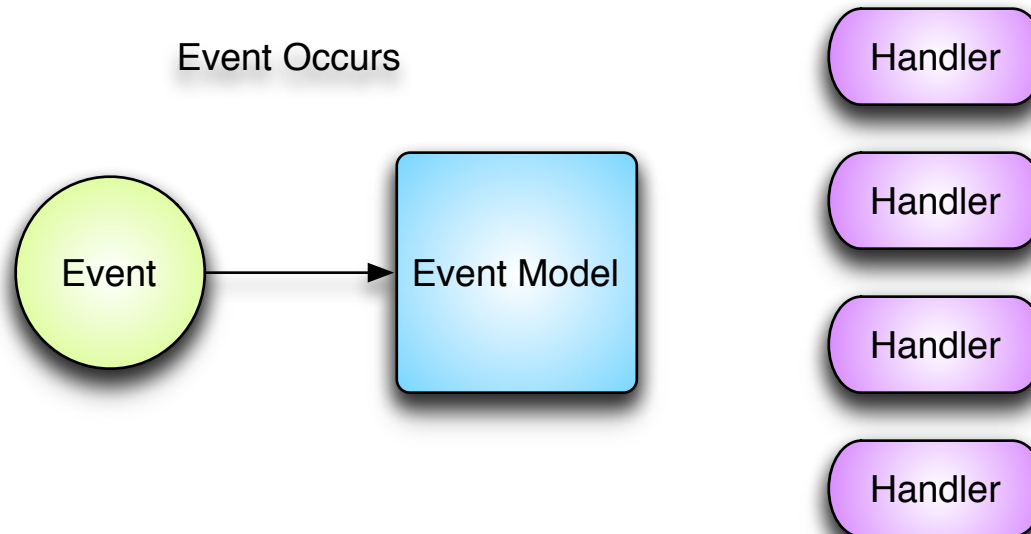
- Wikipedia.org defines an event as:
 - "a significant change in state"
- In Flex, the common idiom is a state:
 - The UI may be in list view state, detail view state or edit state
 - Data may be synchronized (with the server) or "dirty"
 - "Changes in state" are therefore common and
 - So, in Flex, events are common
- Wikipedia.org defines event-driven architecture as:
 - "a software architecture pattern promoting the production, detection, consumption of and reaction to events"

- 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

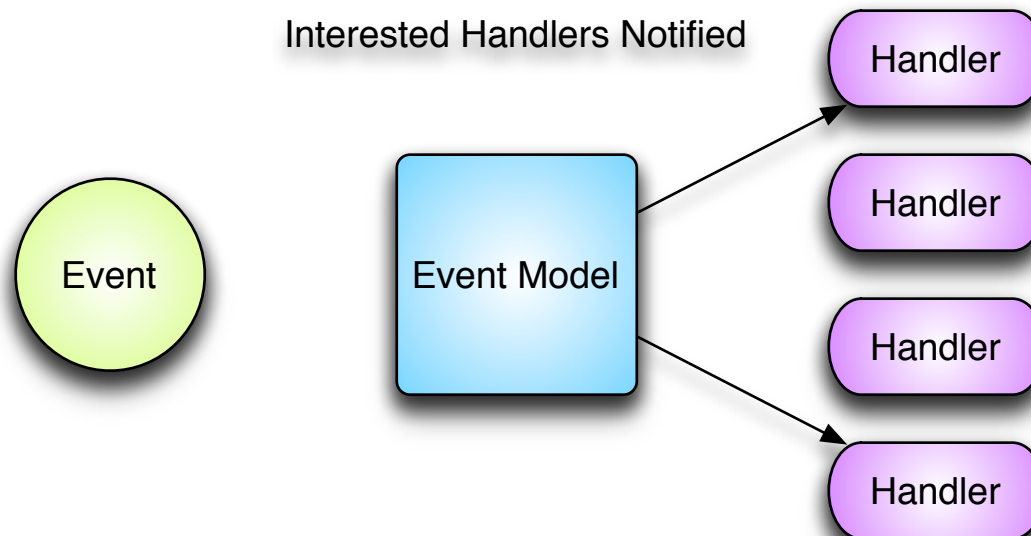
- Event Registration



- Event Arrival



- Event Dispatching



- Event Model:
 - Detects events from producers
 - Dispatches events to consumers that have registered an interest
- Goal: **decouple components**
 - Producers and consumers do not know about each other

External vs. Internal

- User clicks link / button (ColdFusion / Flex)
 - The most traditional sort of external event
- Application "events" (ColdFusion)
 - Application / session / request - start / end
- Anything detected by an event gateway (ColdFusion)
 - JMS, XMPP, SMS, Flex Messaging / Data Services etc
- Result / error callback (Flex / AJAX)
 - Server response is "external" to the Flex / AJAX client

- ColdFusion has almost no native examples
- In Flex, internal events are everywhere:
 - Component lifecycles have creation and initialization events
 - Data binding relies on events triggered by changes to data
 - Events are dispatched for state changes
 - Within components
 - Programmatically for application state changes

Event-Driven Programming in Flex

- External events are often handled declaratively:

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

- Data binding is also declarative:

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

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

- 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

- Registering a listener for an event:

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

- Can add many listeners for same event type

- Announcing an event

```
dispatchEvent( new MyEvent(...) )
```

- Can announce any event containing any data

- ContentController.as from Broadchoice Workspace AIR (Flex)
 - Listener for SAVE_CONTENT
 - UI broadcasts SAVE_CONTENT when Save button clicked
 - Broadcasts event when starting save process
 - Broadcasts event when save process is complete
 - Broadcasts event if save process fails

External Events in ColdFusion

- User clicks link / button
 - URL determines which page ColdFusion will execute
 - URL / form scope contain data for that event
- Most frameworks use index.cfm for all URLs
 - URL / form scope data combined into "event" CFC
 - XML (or convention) describes how to process the external event based on a specific URL (or form) variable
 - e.g., </index.cfm?do=task.list>

- Application.cfc has "builtin" event handlers for the start and end of each request, each session and each "application"
 - Write `onWhateverStart()` / `onWhateverEnd()` methods
 - Called automatically (based on external event type)
- Event gateways listen for external "events"
 - Configure a CFC (and configuration file) for the gateway
 - ColdFusion calls methods automatically when event fires
 - e.g., `onMessage()` passed data about the event

Internal Events in ColdFusion (Frameworks)

- Mach-II, Model-Glue - event-based "implicit invocation"
 - Use XML to define event handler actions
- ColdBox - event-based but uses convention for handler actions
- Fusebox - supports both XML and convention, not "implicit"
- Mostly used to handle external events in a consistent way
- Mach-II, Model-Glue support broadcast / multiple listener

- Event listeners are all registered declaratively (in XML or as code)
- Cannot add new listeners programmatically
- Mostly synchronous
 - Mach-II, Model-Glue have some asynchronous support
- All support programmatic triggering of events
 - `announceEvent()` in Mach-II
 - `addResult()` in Model-Glue (declaratively mapped to an event)
 - `do()` in Fusebox
 - ? in ColdBox

Internal Events in ColdFusion (like Flex)

- Programmatically register event listener:
`model.addEventListener("someEvent",listenerObject);`
- (`listenerObject.handleEvent()` is the method that will be called)
- Create a custom event:
`e = model.new("events.MyEvent");`
- (`events.MyEvent` would extend a standard event type)
`model.dispatchEvent(e);`

- Programmatically register event listener:
`component.addEventListener("someEvent",listenerFunction);`
- (local `listenerFunction()` is the method that will be called)
- Create a custom event:
`e = new MyEvent();`
- (`MyEvent` would extend a standard event type)
`dispatchEvent(e);`

- Flex / ColdFusion variable binding in objects is slightly different
 - Flex allows a method to retain its context
 - ColdFusion treats methods as independent of CFC
 - ColdFusion version must use CFC instances not bare methods

- Object creation syntax is slightly different
 - Flex uses `new` keyword and object type (and `import`)
 - ColdFusion uses `createObject()` function and type "name"
 - ColdFusion version wraps event object creation
 - Maybe Centaur will address this

- Event dispatch syntax is slightly different
 - Flex has `dispatchEvent()` and `addEventListener()` as methods on any `EventDispatcher` and most contexts are within components that extend that class
 - These are methods on the "event model" in ColdFusion (so **if** you extend the "event model" CFC, you can call these methods directly, just like in Flex)

Introducing 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
- Intended to be very flexible and not to force any particular style on you - so there are several ways to do most things!

- Originally intended to solve a Flex / ColdFusion integration problem that current CFML frameworks did not address
- That turned out to be a harder problem than I thought (and I couldn't solve it any better than Joe or Luis or Matt / Peter!)
- Edmund evolved to provide Flex-style event handling to provide a common idiom between client side and server side code
- Edmund also evolved a workflow sub-framework
 - Not covered in this presentation - talk to me offline :)

- Registering a listener for an event

```
edmund.addEventListener("eventName",someObj);  
// note: we pass an object, not a method (like Flex)
```

or

```
edmund.register("eventName",someObj);
```

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

- Can specify additional arguments

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

([addEventListener\(\)](#) is identical - for convenience)

- Announcing an event

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

```
anEvent = edmund.newEvent("anotherEvent");
```

```
edmund.dispatchEvent(anEvent); // like Flex
```

```
edmund.newEvent("oneMoreEvent").dispatch();
```

- Can specify data for events

```
e = edmund.newEvent("PersonRegistered")  
    .values( firstName="Sean", age=45 );
```

```
e.value( "lastName", "Corfield" );
```

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

- Creating custom events

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

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

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

- TaskController.cfc (just an example)
 - External event `task.save` - form button click
 - Model-Glue event handler broadcasts `task.save` message
 - That causes `save()` method of controller to be called
 - Broadcasts event when starting save process
 - Broadcasts event when save process is complete
 - Broadcasts event if save process fails

- Edmund is listed on RIAforge (link at end of presentation)
- Source code is on Google Code
- Mailing list is on Google Groups
- Documentation is minimal
 - Mostly as a few sample code files

Summary

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

- http://en.wikipedia.org/wiki/Event_Driven_Architecture
 - Good starting point to learn about the concepts
- <http://edmund.riaforge.org>
 - Edmund framework project on RIAforge
- <http://corfield.org/cat/edmund>
 - My blog posts about Edmund
- <http://developer.adobe.com/flex>
 - Learn about Flex on the ADC

Speaker Evaluation Forms!

ColdFusion Evangelism Pack

Fusion Authority Quarterly Update

Q & A

Contacting Me!

sean@corfield.org

<http://corfield.org/>

scorfield@broadchoice.com

<http://www.broadchoice.com/>