
# Model-Glue and You


Mark Drew
Railo Technologies
CFUnited
2009

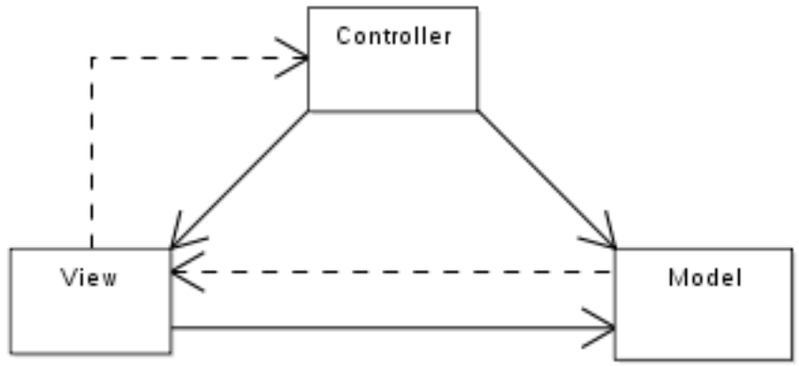
# What is this about?

- Design Patterns in General
- The Model View Controller Design Pattern
- Frameworks
- How Model-Glue can help you make your life easier
- How to develop Model-Glue Applications

# Who is Mark Drew?

- Railo UK's CEO
  - Fast Open Source CFML Engine
  - Development Consultancy
- CFEclipse's Lead Developer
- UK ColdFusion User Group's Co-Manager
- A CFML Developer since '97 and Web developer since '94 (showing my age now eh?)
- Reactor ORM Project Manager

# What is MVC?




- Stands for Model-View-Controller. A way of splitting your application
- Its the king of patterns! (Like Elvis!)
- Once you understand MVC, you are on your way to good Object Oriented practices
- It is applicable to all applications with a UI

Mark Drew - Railo

# Why is MVC hard to learn?

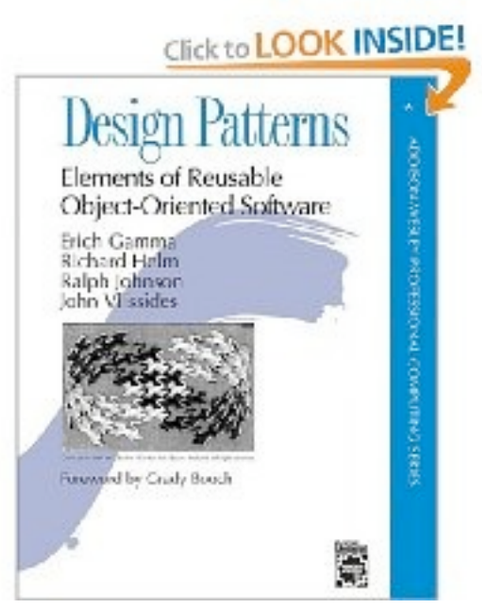
- People start with the framework (implementation) rather than on the pattern itself
- To truly grasp it we need to look at the patterns that are involved in MVC
- Learn it from the bottom up rather than top down! Much easier :)

# MVC: Not just a pretty pattern

- MVC isn't just a pattern it employs multiple patterns
- MVC is a *Compound Design Pattern*
- Rather than a single pattern its multiple patterns working together to do something cool!

# Hold your horses! What is a pattern then?

- Someone might have asked?
- A pattern is a "*tried and true solution to a recurring programming problem*"
- Coined by the "Gang of Four" in the book "*Design Patterns:Elements of Reusable Object-Oriented Software*"
- Lets look at it in a simple example



# Talking in tongues?

"Hey Bob, I come up with a way to split out our apps that we can all FINALLY understand!"

"Oh yeah?  What's that, Jim?"

"I split it out into Views that show stuff, a Controller that guides the requests and a Model that is Taking Care of Business (tm)"

Mark Drew - Railo

# The birth of a pattern

"I'm not sure that's a good name to present to management. Let's call it *Model View Controller* from now on."

"Oh, I've done that too. I call it Bob's Über Traffic Cop Ümlaut."

"Oh, ok. That's not as much fun, but now we can just say 'I wrote an MVC app!' and we both know the whole idea!"

Mark Drew - Railo

# What is a pattern? (finally!)

- Its not HOW you solve a problem
- Its an observation of a problem being repeatedly solved in the same way: a *Pattern* emerges instead of being *invented*

# A compound pattern is...?

- A Compound Pattern is where multiple patterns are used as a team to accomplish a goal unique to their combination

Mark Drew - Railo

# 3 Patterns that make up MVC

- The Strategy Pattern allows MVC to detach the Model from the View and Controller
- The Observer pattern allows different parts of your application to communicate without being coupled
- The Composite pattern often allows the View to be re-arranged at will (not very applicable in Web apps)

# MVC: Why these three?

- The three patterns used in MVC allow a application's UI to be decoupled from its business logic (via *Strategy Pattern*) but still communicate with it (via *Observer Pattern)* and within itself (via *Composite Pattern*)

Mark Drew - Railo

# MVC: Strategizing

- The Strategy pattern is a simple design pattern
- Algorithms (how your code does something) are encapsulated and made interchangeable
- While simple, it enforces a core tenet of Object Oriented Programming

# MVC: Strategizing

- …code to interfaces, not implementations.
- Translator example:

By employing the Strategy pattern, I can write two different CFCs, one  of which turns a string into its Pig-Latin equivalent, while the other reverses it

Mark Drew - Railo

# MVC: Strategizing

- If my client wants me to write a translator application that turns phrases into Pig Latin one day and reverses them the next, my application's UI code doesn't need to change:  only the CFC it uses for translation needs to be swapped.

Mark Drew - Railo

# MVC: Strategizing

- By employing the Strategy pattern, calling code becomes *decoupled* from the code it calls.
- *Loose coupling* is a **good thing!** It leads to flexible applications with long life-spans.

# MVC: Observer, Obviously

- It's onclick, but backwards!
- Instead of procedurally stating that something needs to happen when a button is clicked, the button *broadcasts* that it's been clicked, and any code *observing* the button responds.

# MVC: Observer, Obviously

- In an MVC application, something acts as a *broadcaster*, sending messages other code "listens" for and react to.
- This allows the broadcaster to be *decoupled* from its observers

# MVC: Clean Composition

- The Composition pattern is implemented when one object contains or "has" references to one or many other objects.
- To interact with the entire tree or collection of objects, a client only needs to interact with the top-level *(Compositor)* object

# MVC: Clean Composition

- In an MVC application using a typical Tree control, the Controller needs to simply tell the Tree that it contains new data. The Tree should then update all of its branches and leaves automatically.

# MVC: Clean Composition

- By decoupling (there it is again!) the Controller from as many portions of the View as possible, the View can more freely be re-arranged without changing the Controller

Mark Drew - Railo

# Railo MVC: The Story So Far

- We've learned about patterns.
- We've learned some vocabulary.
- We've learned some OOD concepts
- We've learned some specific patterns.
- We've learned that Model-View-Controller is made up of three patterns.

# Enough with the pattern talk!

- What is a framework then?
  - Its a standardized way to develop code
  - Allows developers to “speak the same language” (like we are doing here!)
  - Provides a nice abstraction to develop code
  - ~~Enforces~~ Suggests good practice

# Frameworks in CF

- There are a lot of frameworks in CF some developed for specific needs:
  - Model-Glue, Mach-II, Fusebox, ColdBox et al provide MVC capabilities
  - Reactor and Transfer provide ORM (Object Relationship Mapping) capabilities
  - ColdSpring and Lightwire provide Inversion of Control and Dependency Injection capabilities

# Say hello to Model Glue

- **Finally** get round to Model Glue
- Its an MVC framework.
- It uses ColdSpring for configuration and dependency injection
- It uses Reactor or Transfer for ORM capabilities
- Provides the MVC separation for you

# Model Glue cont...

- Configured by XML files rather than convention:
  - ColdSpring.xml : settings
  - ModelGlue.xml : events and message-listeners

# Some terms in Model-Glue

- Message-Listener: listens for your Broadcasts
- Broadcast: sends a message to your listener!
- Event-handlers: handle an EVENT! e.g. calling a url:
  *index.cfm?event=login.user*
  *or*
  *index.cfm?event=myEvent*
- View: the code to display
- Results: things that include the view or redirect to other events

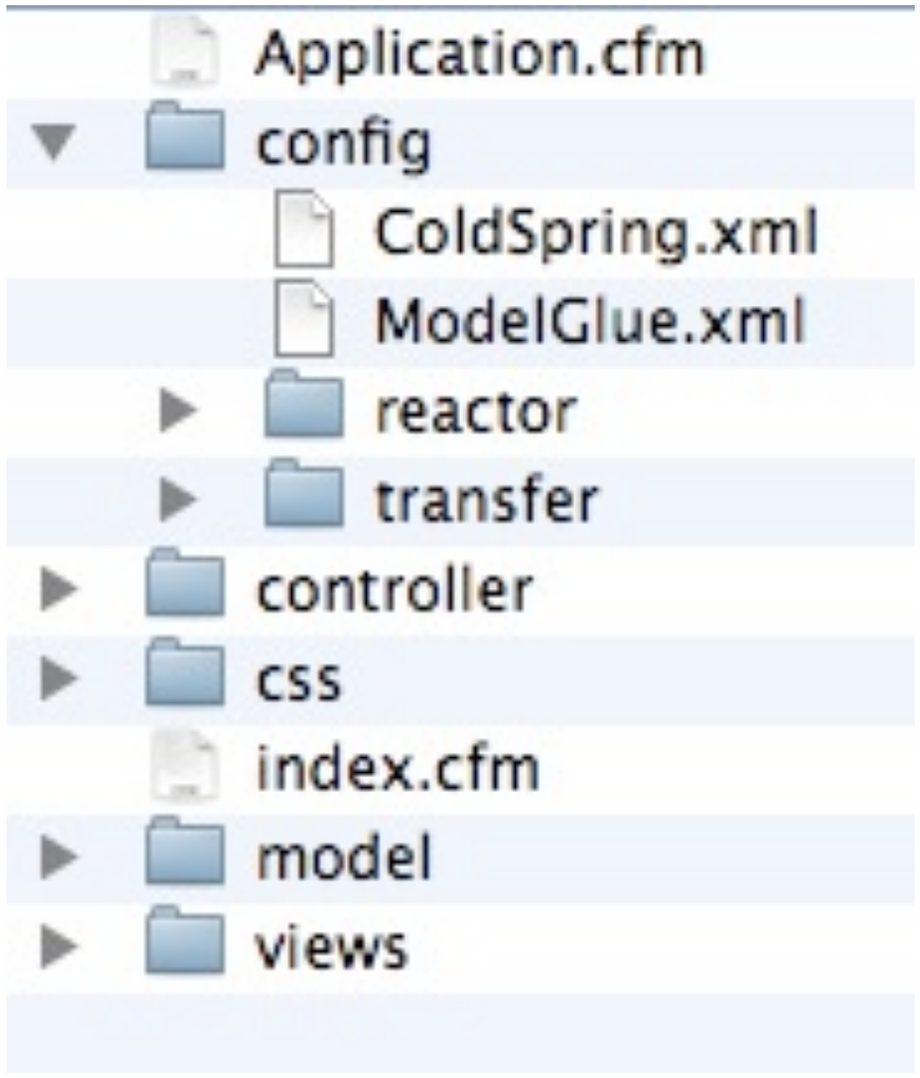
# Installing Model-Glue

- Download it from: http://www.model-glue.com
- Put it in your web root or create a mapping called */ModelGlue* to it
- Needs ColdSpring: http://www.coldspringframework.org/
- Put it in your web root or create a mapping called */coldspring* to it
- Install an ORM: such as Transfer or Reactor (same way as above... you are getting the idea!)
-

# MG Application template

- You can run an ant script from the */modelglueapplicationtemplate* folder or:
- Copy the the contents into a new folder (then we can configure it ourselves! and learn!)

# RailoMG directory structure




Mark Drew - Railo

# MG configuration (ColdSpring.xml)

```
<bean id="modelGlueConfiguration" class="ModelGlue.unity.framework.ModelGlueConfiguration">
<!-- Be sure to change reload to false when you go to production! -->
<property name="reload"><value>true</value></property>
    <!-- Rescaffold is overridden by reload - if reload is false, rescaffold's setting doesn't matter -->
<property name="rescaffold"><value>true</value></property>
<!-- Be sure to change debug to false when you go to production! -->
<property name="debug"><value>true</value></property>
<property name="defaultEvent"><value>page.index</value></property>
<property name="reloadPassword"><value>true</value></property>
<property name="viewMappings"><value>/modelglueapplicationtemplate/views</value></property>
<property name="generatedViewMapping"><value>/modelglueapplicationtemplate/views/generated</value></property>
<property name="configurationPath"><value>config/ModelGlue.xml</value></property>
<property name="scaffoldPath"><value>config/scaffolds/Scaffolds.xml</value></property>
<property name="statePrecedence"><value>form</value></property>
<property name="reloadKey"><value>init</value></property>
<property name="eventValue"><value>event</value></property>
<property name="defaultTemplate"><value>index.cfm</value></property>
<property name="defaultExceptionHandler"><value>exception</value></property>
<property name="defaultCacheTimeout"><value>5</value></property>
<property name="defaultScaffolds"><value>list,edit,view,commit,delete</value></property>
</bean>
```

Mark Drew - Railo

# Configuring your events
# ModelGlue.xml

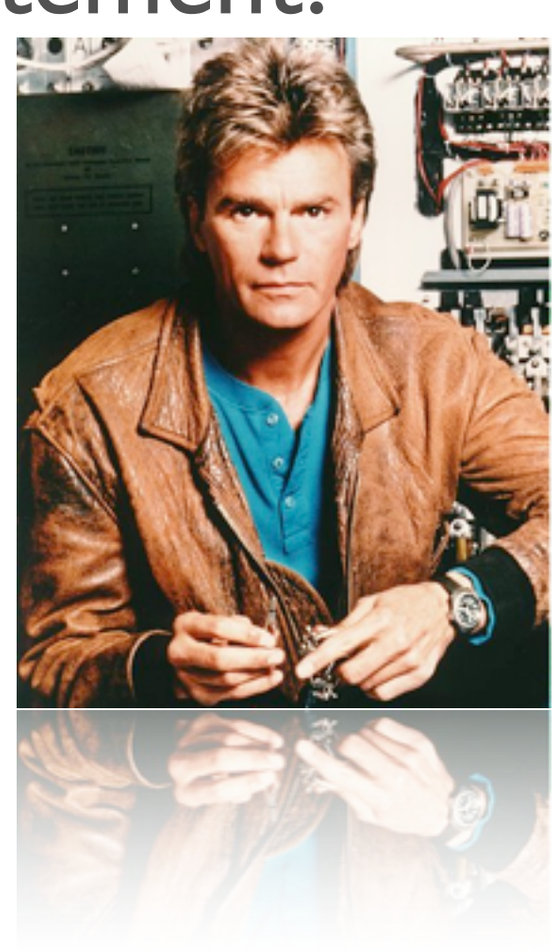
```
<controllers>
    <controller name="MyController" type="modelglueapplicationtemplate.controller.Controller">
        <message-listener message="OnRequestStart" function="OnRequestStart" />
        <message-listener message="OnQueueComplete" function="OnQueueComplete" />
        <message-listener message="OnRequestEnd" function="OnRequestEnd" />
        <message-listener message="sayHello" function="doSayHello" />
    </controller>
</controllers>

<event-handlers>
    <event-handler name="page.index">
        <broadcasts>
            <message name="sayHello"/>
        </broadcasts>
        <results>
            <result do="view.template" />
        </results>
        <views>
            <include name="body" template="dspIndex.cfm" />
        </views>
    </event-handler>
```

Mark Drew - Railo

# Lets get playing with the code!

- Lets actually set all of this up as an example.
- This is live, with no safety net! The excitement!



Paperclip and bubblegum not required...

Mark Drew - Railo

# Scaffolding

- Now we have seen all how it all fits together, what is this scaffolding stuff?
- With an ORM you can connect directly to your model. ModelGlue will generate the connections for you!
- Lets setup reactor....

# Reactor

- Download it from: http://www.reactorframework.com/
- Put it under your web root or .... you remember this! Create a mapping called */reactor* !

# Reactor configuration (ColdSpring.xml)

```
<alias alias="ormAdapter" name="ormAdapter.Reactor" />
<alias alias="ormService" name="ormService.Reactor" />
<bean id="reactorConfiguration" class="reactor.config.config">
    <constructor-arg name="pathToConfigXml">
        <value>/myProject/config/reactor/Reactor.xml</value>
    </constructor-arg>
    <property name="project"><value>myProject</value></property>
    <property name="dsn"><value>cfcamp08</value></property>
    <property name="type"><value>mysql</value></property>
    <property name="mapping"><value>/myProject/model/data/reactor</value></property>
    <property name="mode"><value>development</value></property>
</bean>
```

# Lets create a blog!

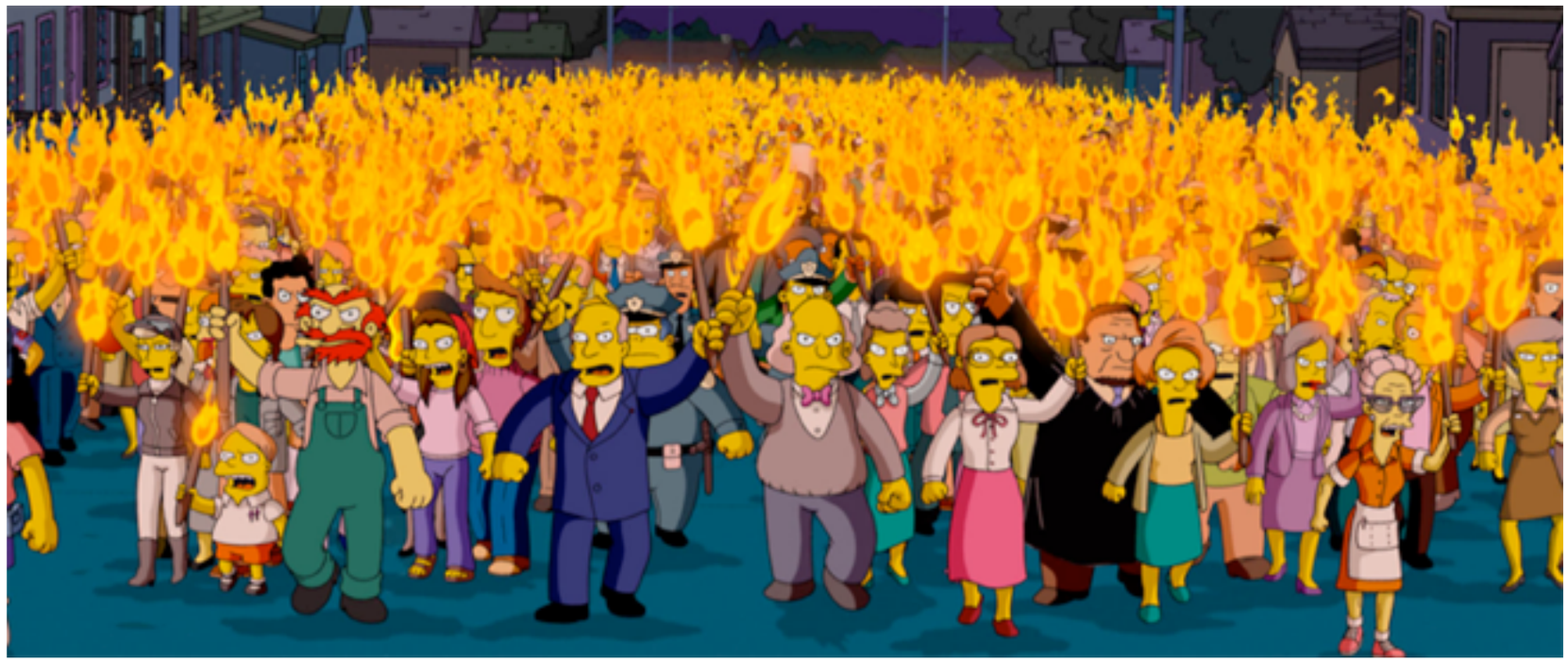
- Because there aren't enough blogs out there, lets create one in 10 minutes or... so... maybe?

Mark Drew - Railo

# Summary

- Learned about Design Patterns and MVC
- Learned some other frameworks
- Saw how you can develop with Model Glue
- Scaffolded your application pretty quickly!

# Questions?



Mark Drew - Railo