

ColdSpring and Model-GLue

Mark Drew
UKCFUG - June 2006

CFUG Sponsors



ColdFusion Hosting Professionals



For all your Adobe Software



Flex & ColdFusion
development and consulting
specialists

ColdSpring and Model-Glue

What is ColdSpring

- An Inversion of Control Framework
- or... A Dependency Injection Framework
- Based on the Java Spring Framework (not a port)

Inversion of Control and Dependency Injection

- Simply put: It INJECTS dependencies into your components
- Why? Code should be re-usable, although good, it means you have more to wire together (aka: coupling)
- Less coupling is better practice, your components shouldn't need to know how to create their dependencies

Example (ShoppingCart.cfc):

```
<cffunction name="init" returntype="ShoppingCart"...>  
    <cfargument name="maxItems" type="numeric"...>  
    <!--- create the dependency --->  
    <cfset variables.TaxCalculator = CreateObject("component",  
        "TaxCalculator").init() />  
    <cfset variables.maxItems = arguments.maxItems />  
  
</cffunction>
```

What happens if...

- BUT, what happens if TaxCalculator changes?
- Lets add a Tax Rate to the TaxCalculator

Example (ShoppingCart.cfc and TaxRate):

```
<cffunction name="init" returntype="ShoppingCart"...>
```

```
  <cfargument name="maxItems" type="numeric".../>
```

```
  <cfargument name="TaxRate" type="numeric" .../>
```

```
  <!--- create the dependency --->
```

```
  <cfset variables.TaxCalculator = CreateObject("component",  
  "TaxCalculator").init(arguments.TaxRate) />
```

```
  <cfset variables.maxItems = arguments.maxItems />
```

```
</cffunction>
```

Example (ShoppingCart.cfc with ColdSpring):

```
<cffunction name="init" returntype="ShoppingCart"...>  
    <cfargument name="maxItems" type="numeric".../>  
    <cfargument name="TaxCalculator" type="TaxCalculator" .../>  
  
    <cfset variables.TaxCalculator = arguments.TaxCalculator />  
    <cfset variables.maxItems = arguments.maxItems />  
  
</cffunction>
```

With ColdSpring...

- There is no change! The code remains the same in the ShoppingCart!
- ColdSpring would create the Dependencies for us and wire them together
- We dont need to change the code of our components.

Dependency Injection

Benefits

- Separation of Concerns: Components don't have to do things that are not related to them
- Components aren't tied to other components' implementations.
- Components are easier to configure and you don't have to change the code

Dependency Injection

Benefits

- Easier to test, you can create “stub” code and trick the component into running
- Overview of dependencies in one place.
- Your components are NOT tied to ColdSpring, only the calling code would use ColdSpring

Implementing Coldspring

- Setup with a simple XML file
- Based on the Java-beans spec (hence references to <beans>)
- Defines the “beans” (i.e.: Components) and what they need to be initialised.

Implementing ColdSpring

Simple Configuration:

```
<bean id="ShoppingCart"  
class="app.model.ShippingCart" />
```

Configure `app.model.ShippingCart` and
when I ask for "ShoppingCart" return an
initialised instance of the component

Implementing ColdSpring

As a constructor (...init(MaxItems))

```
<bean id="ShoppingCart"  
class="app.model.ShoppingCart">  
  <constructor-arg name="MaxItems">  
    <value>15</value>  
  </constructor-arg>  
</bean>
```

Implementing ColdSpring

As a property (...setMaxItems(MaxItems))

```
<bean id="ShoppingCart"  
class="app.model.ShoppingCart">  
  <property name="MaxItems">  
    <value>15</value>  
  </property>  
</bean>
```

Implementing our Example:

```
<!-- define TaxCalculator -->
<bean id="TaxCalculator" id="app.model.TaxCalculator">
  <constructor-arg name="TaxRate"><value>1.175</value></constructor-arg>
</bean>

<!-- define ShoppingCart -->
<bean id="ShoppingCart" id="app.model.ShoppingCart">
  <constructor-arg name="MaxItems"><value>15</value></constructor-arg>
  <constructor-arg name="TaxCalculator">
    <ref bean="TaxCalculator">
  </constructor-arg>
</bean>
```

What happened there?

- The TaxCalculator has been setup
- The ShoppingCart has been setup and constructed (injected) with the TaxCalculator
- No code was needed in the ShoppingCart to setup the TaxCalculator
- Your code is now more reusable, testable and maintainable

The BeanFactory

- Holds all your components
- Instantiates, configures and resolves dependencies for your components (beans)

Installation

- Download
- Copy to webroot
- or
- Create a mapping to coldpsring folder

Implementation

- Create a bean factory:

```
<cfset myBeanFactory = CreateObject("component",  
"coldspring.beans.DefaultXmlBeanFactory").init()>
```

Configuration

- The `DefaultXmlBeanFactory` can only read XML configuration
- Currently no implementation for runtime addition of bean settings
- But: XML can be constructed and passed in

Passing a File

- Pass a fully qualified path to the xml file:

```
<cfset myBeanFactory = CreateObject("component",  
"coldspring.beans.DefaultXmlBeanFactory").init()>
```

```
<cfset myBeanFactory.loadBeansFromXmlFile("/path/to/xml",  
true)>
```

true = Lazy Initialisation, e.g. dont load until asked to do so.

Passing a Raw XML String

- Pass a raw xml string:

```
<cfset myBeanFactory = CreateObject("component",  
"coldspring.beans.DefaultXmlBeanFactory").init(>
```

```
<cfcontent variable="beanConfig">
```

```
<beans>
```

```
  <bean id="aBean" class="app.model.aBean" />
```

```
</beans>
```

```
</cfcontent>
```

```
<cfset myBeanFactory.loadBeansFromXmlRaw(beanConfig,  
true)>
```

Passing a XML Object

- Pass a parsed Coldfusion xml object:

```
<cfset myBeanFactory = CreateObject("component",  
"coldspring.beans.DefaultXmlBeanFactory").init()>
```

```
<cffile action="read" file="/path/to/xml"  
variable="xmlContent">
```

```
<cfset beanXML = XMLParse(xmlContent)>
```

```
<cfset myBeanFactory.loadBeansFromXmlObj(beanXML, true)>
```

Getting your objects

```
<cfset myBeanFactory = CreateObject("component",  
"coldspring.beans.DefaultXmlBeanFactory").init()>
```

```
<cfset myBeanFactory.loadBeansFromXmlFile("/path/to/xml",  
true)>
```

```
<cfset myShoppingCart = myBeanFactory.getBean  
("ShoppingCart") />
```

Resources

- <http://www.coldspringframework.org>
- MDeamon@coldspringframework.org

Model-Glue

Model-Glue 2.0 : Unity

What's new?

- Includes Reactor and uses ColdSpring for ALL its configuration, you can change how it behaves
- Major speed improvements
- Scaffolding: Makes your List-Edit-Add pages for you
- ActionPacks: Modules that can be dropped into your applications
- Documentation!

Configuration Changes

ModelGlue 1.1

ModelGlue.xml

- Config
- Controllers
- Event Handlers

Optionally...

Reactor.xml

- Config
- Objects

ModelGlue 2.0

ColdSpring.xml

- MG config
- Reactor Config

ModelGlue.xml

- Controllers
- Event Handlers

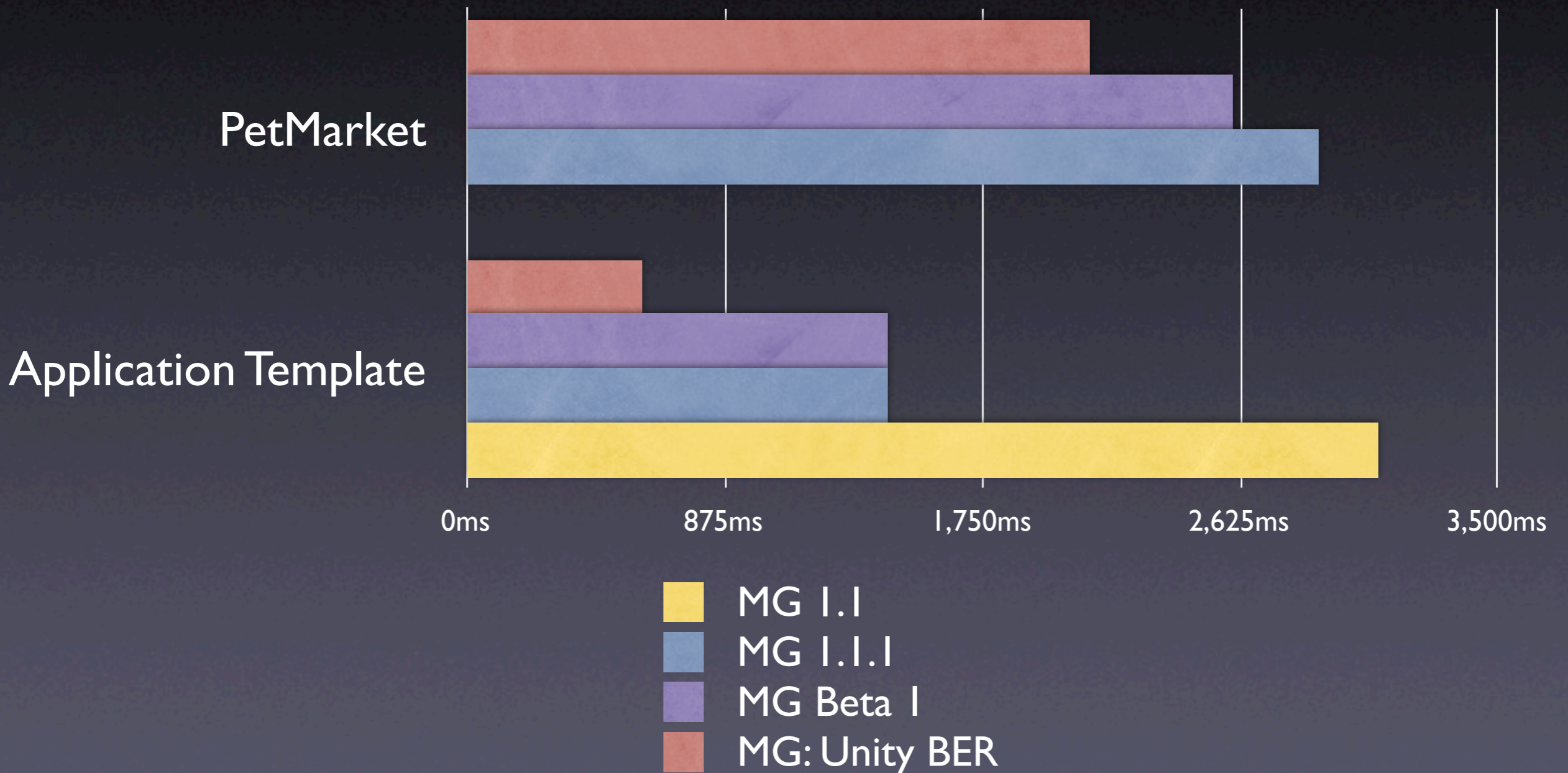
Reactor.xml

- Objects

Configuration Changes

- All configuration for your application in one file loaded automatically by Coldpsring
- ModelGlue file takes care of your Controllers and Event Handlers
- Reactor file takes care of your objects and relations

Speed Improvements



Scaffolding

- Creates all your List, Edit, View and Delete pages for you
- Uses Reactor to introspect database
- Very little change in the config: only addition of `<scaffold object="reactorObject">` tag

Example of Scaffolding

- Demo...

Action Packs

- Include pre-made MG Applications within a main application

```
<modelglue>  
  <include template="/mgwiki/config/Wiki.xml" />  
  
  <!--- Application stuff...-->  
</modelglue>
```

Resources

- <http://www.model-glue.com>
- <http://clearsoftware.net>
- Mailing list (at www.model-glue.com)

And Finally...

CFEclipse 1.3 Beta

- Out next week! (maybe)
- Lots of new features (not new if you have used the nightly)
- Looking for Beta Testers as from this weekend!

Q&A and/or Pub?