Architecting ColdFusion applications – a framework overview

Kai Koenig, Digital Solutions Architect

Kudos

- Thank you to Adobe AsiaPacific for the support of this series of talks !!!
  - Access to the Breeze server
  - Registration module
  - Promotion

Agenda

- About Kai
- Architectures and frameworks - definition and some general thoughts
- Benefits of dealing with architecture and frameworks for ColdFusion developers

Agenda

- Some common CF frameworks
  - Fusebox
  - Mach-II
  - ModelGlue and Unity: MG 2 + ColdSpring + Reactor
- Frameworks and RIA development
- Resources

About Kai

- Digital Solutions Architect at ZeroOne (NZ) Ltd.
- Fields of work:
  - Development, Training, Architectures
  - ColdFusion, Flex, Java, Flash
- Adobe Certified Professional
- Adobe Certified Master Instructor

Definitions

- Software Architecture
  - “Software architecture is the set of design decisions which, if made incorrectly, may cause your project to be cancelled.” (Eoin Woods)
  - “Architecture is the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.” (ANSI/IEEE)

Definitions

- Framework
  - “In software development a framework is a defined support structure in which another software project can be organized and developed. A framework may include support programs, code libraries, a scripting language, or other software to help develop and glue together the different components of a software project.”

Why?

- Why should you use a framework or even bother with architecture?
  - Idea: To provide a structure for your software development that allows a robust and well-defined evolution of your applications
  - Frameworks and architectural patterns are designed with the intention of facilitating software development
  - Eventually developers should spend more time on meeting the software’s business requirements rather than dealing with providing a working foundation

Benefits for CF developers

- ColdFusion started pretty much as a proprietary system and each developer developed in their own style -> scripting HTML pages
- With CFMX and putting the product closer to Java, a lot of people started to think about bigger structures and noticed the benefits of ColdFusion Components and the possibilities of Java.

Benefits for CF developers

- In Java development it is pretty common to work with frameworks and mechanisms of making the developer's work easier – due to the complexity of the language and the API.
- With CFMX and the possibilities of CFCs, the development of frameworks itself became easier as well (object-like components, inheritance, typing of "CFC-Objects" etc.)

Benefits for CF developers

- It is – obviously – not mandatory to use a framework for your CF development work, so – why would you want to though?

Benefits for CF developers

- Some answers:
  - Frameworks could make your life easier
    - Well-defined structures provide an easy set up process for new applications and save maintenance time as all your applications are built in an identical and/or very similar way.
    - A good framework is well-documented and follows existing best practices in software and web development. That makes it easier for your developers to create good software.
  - Frameworks allow the standardization of development processes
    - New developers or new members of a team save time by not having to dig through tons of legacy code
    - A good framework provides functionality out of the box
    - Frameworks restrict developers to a standard way of coding

Fusebox - general

- Fusebox is a rather old framework; the first public version that was used widely was Fusebox 2 (roughly in 1998-2000).
- Fusebox as a framework consists of a few core files (in Fusebox 4.x those are .cfm files and some UDF libraries).
- Fusebox is available for CF and PHP and provides a very mature framework which is used by a lot of developers throughout the world.

Fusebox - general

- The current version is 4.1, currently there is a version 5 in beta testing, which sounds very interesting and promising, but I haven't worked with it yet.
- Fusebox uses a metaphor of terms that fits into the language of electrical circuits:
  - "Circuits" are the major sections of an application
  - "Fuseactions" are the individual actions within a circuit
  - "Fuses" are individual files that are used to process a fuseaction and could be query, display, layout or action files

Fusebox - theory

- The framework follows the idea of encapsulation of responsibilities, so that single functional circuits could be taken out of an application and plugged into different applications for the purpose of reusing code.
- All requests are piped through a central controller, usually the index.cfm file. This approach is described by the FrontControllor design pattern.

Fusebox - processing

- The index.cfm and a configuration file called fusebox.xml are responsible for the processing of a request made to a Fusebox application.
- Typical xml tags of the fusebox.xml could be:
  - <do action="…" />
  - <include template="…" />
  - <relocate … />

Fusebox - processing

- Each circuit has its own circuit.xml to define its configuration independently from the application and/or other circuits.
- The XML files are translated into native ColdFusion code during the first request and could be cached rather effectively by the application server and by the web server for further requests.

Fusebox - processing

- This allows a multi-step processing algorithm:
  - Figure out which sub-actions and fuses are involved calling a particular fuseaction.
  - Parsing the necessary XML for the processing and then storing and caching it.
  - Providing the parsed information to the Fusebox runtime for execution – if this particular fuseaction is unchanged since the last execution, it results in a very fast execution.

Fusebox - additional info

- There are different approaches of developing Fusebox applications besides the one described here.
- Some books suggest using a MVC approach, this would result in having 3 circuits in the root of the application:
    - Model: contains action and query files
    - View: contains display and layout files
    - Controller: provides public fuseactions to be called from external sources

Fusebox - additional info

- Another approach could be replacing the query files with ColdFusion components and then calling them from the action files, respectively from fuses within the model circuit.
  - The benefit of this: kicks Fusebox into the “pseudo-OO” world of using ColdFusion Components and makes Fusebox a more modern approach.

Mach-II - general

- Mach-II is a framework developed by Ben Edwards and is based on the concept of implicit invocation. Its first release was published in 2003.
- Event-driven implicit invocation usually leads to two often requested features of a software architecture:
  - High cohesion – means that a component has a very focussed purpose
  - Loosely coupled – the components of the architecture are independent and reusable

Mach-II - general

- Implicit invocation
  - “The idea behind implicit invocation is that instead of invoking a procedure directly, a component can announce (or broadcast) one or more events. Other components in the system can register an interest in an event by associating a procedure with the event. When the event is announced the system itself invokes all of the procedures that have been registered for the event. Thus an event 'implicitly' causes the invocation of procedures in other modules.” (D.Garland and M.Shaw)
- II is pretty much how a lot of software systems work today!

Mach-II - general

- The Mach-II framework for ColdFusion consists of a set of CFCs (around 20) which are used to deliver the core functionalities of the framework – the framework itself IS object-oriented.
- Current version is 1.1.0 (from end of 2005) and the release cycle has pretty much slowed down, as the purpose the framework was built for is pretty much reached.

Mach-II - processing

- A Mach-II application usually comprises:
  - A central configuration/controller file: mach-ii.xml
  - Model CFCs, which form the business logic/domain model layer.
  - View templates, the .cfm files that are displayed to the user and that the user interacts with.
  - Plugins and filters contain code that could either filter specific events (for example to make sure someone is logged in) or run prior to every event.

Mach-II - processing

- Mach-II also makes use of the concepts of leveraging a FrontController to route all requests (index.cfm).
- Other terms that are commonly used:
  - Event: core part of the framework
  - Listener: extends from a generic Listener CFC and is used to provide functionality on a notification basis

Mach-II - processing

- A typical request looks like:
  - Form submit or URL request
  - Event is created
  - XML configuration file is loaded and parsed
  - The specified event listener is invoked and all necessary details are passed in
  - The listener spawns new event(s) based on results

ModelGlue - general

- ModelGlue is a rather new framework, developed and driven by Joe Rinehart.
- The approach of MG is a bit different from both Fusebox and Mach-II:
  - MG is a lightweight framework with the particular purpose of providing a powerful and easy way to connect presentation layer to business logic.

ModelGlue - general

- That’s an important distinction to the other frameworks as MG’s approach is NOT to deliver a global solution, but to focus on being the glue between view and model.
  - Strongly expressed by the fact that there are no restrictions in how one prefers to build the business logic and data access layer.
  - It doesn’t have to be part of the framework structure, could be even build by using a different framework.

ModelGlue - processing

- MG is FrontController- and event-driven, but by far not as heavyweight as Mach-II.
- The framework and application configuration is held in the server memory (application scope).
- A request is piped through index.cfm and broadcasts messages to controller components. They then call business logic, which doesn’t have to be part of the framework, and report back with results.

ModelGlue - processing

- ModelGlue applications follow the principle of pre-rendered and nested presentation layer elements.
  - Very handy feature, as your event handler can consist of various view templates which are rendered during the execution of the event and then used to form a global layout

ModelGlue - Unity

- ModelGlue 2.0, codename “Unity” is currently in public beta.
- MG 2.0 is fully backwards-compatible with existing ModelGlue 1.x applications! Well, to ensure this is at least one of the goals of the public beta…
- Unity incorporates two additional frameworks: Reactor and ColdSpring

ModelGlue - Unity

- Benefits of this approach are:
  - ColdSpring is a CF port of the famous Java Spring framework and supports you in using nearly any existing business logic encapsulated in CFCs immediately with MG 2
  - Reactor is an Object-Relational Modelling framework, which allows you to create database abstractions on the fly.
- The combination of both makes Unity incredibly powerful !

Comparison

- Core structure
  - Fusebox: .cfm files, developers can code CF apps without using components and object technology.
  - Mach-II: consists of CFCs, applications in Mach-II have to be developed with the Mach-II base components in mind. Mach-II enforces MVC-style application development.
  - MG and Unity: the framework itself is a mixture of .cfm files and CFCs, but enforces MVC-style development as well.

Comparison

- Configuration/Controller
  – XML based for all of the frameworks, Fusebox doesn’t have a monolithic XML controller file, but offers XML configuration files per circuit.
  – All frameworks pre-parse/cache the configuration in a very efficient way during runtime.

Comparison

- Event handling
  - – Fusebox: static event handling, explicit invocation
  - – Mach-II: very dynamic event handling, implicit invocation
  - – ModelGlue: leaned towards a static event handling, but allows flexibility in execution paths, implicit invocation

Comparison

- The big question: which framework is the best?
- Or maybe another question even before that one: Do I really need a framework? Will I be a better CF developer after adopting a framework?
  - The answer to this is: no, not necessarily. You might have a very good and standardized approach already, for example by using CFCs and OO-techniques.
  - But frameworks help developers of all levels to structure their development and to make their life and the lives of their development teams easier.

Comparison

- Answer to the first question:
  - There is no “best” framework, different frameworks support different approaches and have different goals, try to find the one that you as a developer could benefit from the most.

Comparison

- Some guidelines:
  - Fusebox is probably the easiest approach for CF developers who haven't been exposed to OO-like technologies such as CFCs.
  - Mach-II and ModelGlue are both full of OO, so they might be harder to learn, but are eventually a better approach for people looking into an OO-style for their development. ModelGlue is the more lightweight approach, as it allows more flexibility in dealing with the surrounding environment.

Comparison

- Fusebox and ModelGlue are very healthy frameworks in term of community and ongoing development, Mach-II is rather static and there’s not really a lot of ongoing development.
- Particularly ModelGlue 2 (Unity) seems to be powerful – without having looked at Fusebox 5 much yet

Frameworks and RIA

- All covered frameworks are CF-/HTML-based - what if someone wants to develop a CF-driven Rich Internet Application in Flash or Flex?
- Apparently we need some different concepts and approaches then:
  – Asynchronous calls to business logic
  – View rendering to Flash has to be handled differently
- Has to integrate with existing RIA-frameworks

Frameworks and RIA

- Cairngorm 2.0 (for Flex 2.0)
  - Makes use of backend CFCs to provide Data Access Objects. The CFCs have to follow some basic rules to be useable for Flex.
- ARP (for Flash applications, port to make it useable for Flex 2.0 apps is on the way)
  - Very flexible, it's possible to dock a lot of different approaches to ARP's server side

Frameworks and RIA

- A good solution for ARP seems to be Tartan, a framework which is just dealing with a way to structure the business logic and server layer of an application.
- I’ve got reports from a few people using ARP for their Flash development that they have achieved awesome results by using Tartan as their CFC backend framework.

Personal flavour

- My choice?
  - Use a framework!
  - Use a framework!
  - From today’s perspective ModelGlue 2 is my personal favourite; it totally suits my style of development, and ModelGlue 1.x became the standard framework for any new ZeroOne development work some months ago.
  - Did I mention: “Use a framework”?
  - Use a framework!

Resources

- Further information on the front controller pattern (warning: Java stuff!)
  - http://java.sun.com/blueprints/corej2eepatterns/Patterns/FrontController.html
- Framework sites:
  - Fusebox: http://www.fusebox.org
  - Mach-II: http://www.mach-ii.com
  - ModelGlue: http://www.model-glue.com
  - Unity public beta 1: http://www.model-glue.com/modelglue_2.0.161.zip
  - Reactor: http://doughughes.net/index.cfm
  - ColdSpring: http://www.coldspringframework.com
  - Tartan: http://www.tartanframework.org/tartan/?page=TartanFramework
  - ARP: http://osflash.org/arp
  - Cairngorm Flex 2: http://www.richinternetapps.com/archives/000143.html

Contact

Kai Koenig

email:
kai@zeroone.co.nz
kai@bloginblack.de

ZeroOne (NZ) Ltd.
Level 4 / 107 Customhouse Quay
Wellington

DDI: +64 4 471 4448