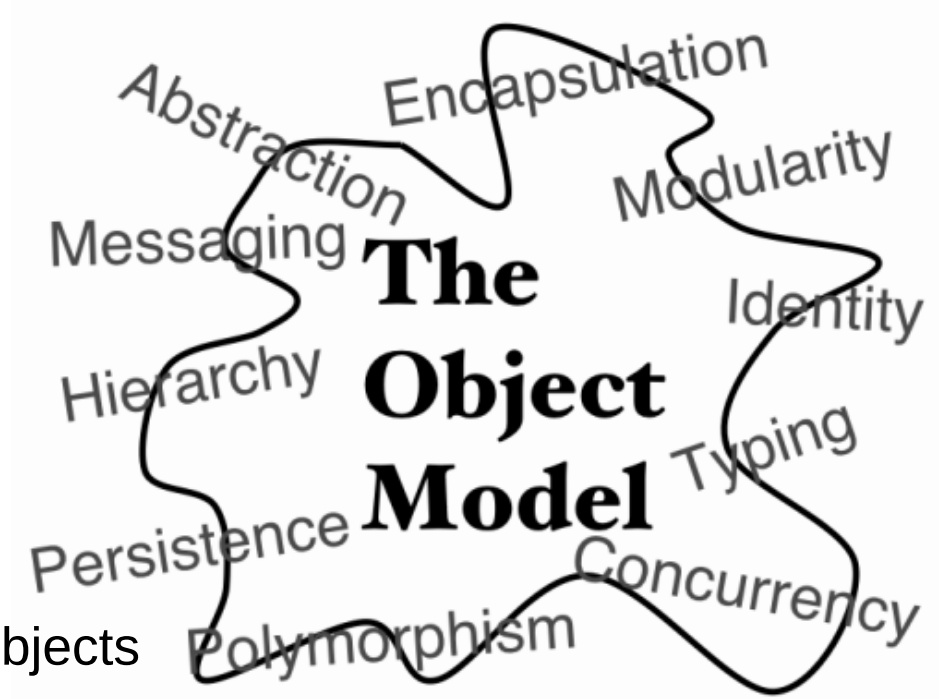
FRAMEWORKS 2007 CONFERENCE

Intro to Object Factories

By Rob Gonda

History

- Once upon a time…
- Procedural Code
  - Spaghetti Code
  - Organized
    - Includes
    - Modules
- OOP
  - Objects as containers
  - The Big Object
  - Scoped / Persistent Objects
  - Encapsulation?
  - Relations/ Wiring



OOP: The Object

- Basics
- Lifecycle
- Constructors
- Types

OOP: The Object: Basics

- Class: A class defines the abstract characteristics of a thing, including the thing's characteristics – its attributes or properties – and behaviors or methods
- Object: A particular instance of a class.
- Method: An object's abilities – Function

OOP: The Object: Lifecycle

- Gets created → runs pseudo-constructor
- Gets initialized → CF has no constructor method. init() method commonly used
- Use methods
- Destroy component → CF has no deconstructor method.

OOP: The Object: Types

- Business Model Layer
  - Transient – stateful
    - People, places, and things
- Service Layer
  - Persistent – stateless
    - Services and infrastructure

Problems

- Object Initialization
- Relationships – wiring
- Manageability – moving objects – keeping paths
- Loosely coupled
- Unit Testing

Design Patterns Defined

- In software engineering, a design pattern is a general repeatable solution to a commonly occurring problem in software design.

The Object Factory

- Typically, an object factory is quite simple and small. Its main role is to collect the information necessary to create an instance of the intended object's class and then to invoke that class's constructor.

The Factory Pattern

- The factory method pattern is an object-oriented design pattern. Like other creational patterns, it deals with the problem of creating objects without specifying the exact class of object that will be created.

Why Factories

- Object Factory creates objects
- Centralize repository for path and dependencies
- All objects depend on the factory
  - Factory is passed to all objects

Inversion of Control

- Inversion of Control, also known as IOC, is an important object-oriented programming principle that can be used to reduce coupling inherent in computer programmes.

Dependency Injection

- Dependency injection (DI) is a programming design pattern and architectural model, sometimes also referred to as inversion of control or IoC, although technically speaking, dependency injection specifically refers to an implementation of a particular form of IoC.

Dependency Injection II

- Dependency injection is a pattern in which responsibility for object creation and object linking is removed from the objects themselves and transferred to a factory. Dependency injection therefore is inverting the control for object creation and linking, and can be seen to be a form of IoC.
- There are three common forms of dependency injection: setter-, constructor- and interface-based injection.

Dependency Injection III

- Dependency injection is a way to achieve loose coupling. The technique results in highly testable objects, particularly when applying test-driven development using mock objects: Avoiding dependencies on the implementations of collaborating classes (by depending only on interfaces that those classes adhere to) makes it possible to produce controlled unit tests that focus on exercising the behavior of, and only of, the class under test.
- To achieve this, dependency injection is used to cause instances of the class under test to interact with mock collaborating objects, whereas, in production, dependency injection is used to set up associations with bona fide collaborating objects.

ColdSpring: what, why

- "Spring is the best thing to happen to **programming** in 20 years" – Antranig Bassman, RSF Lead, SEPP Conference Vancouver 2006

- ColdSpring was "inspired" by Spring – we're not porting all the functionality, but solving the same problems.

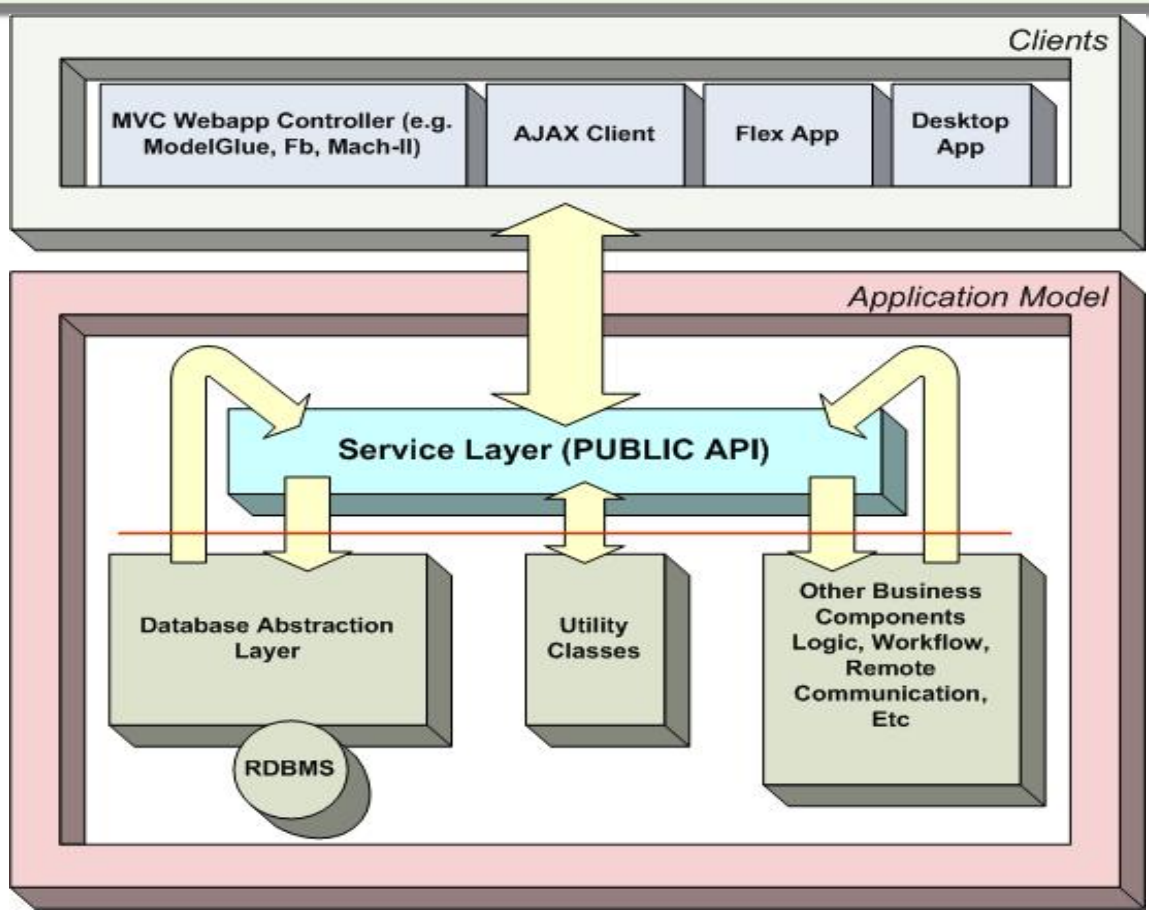
ColdSpring II: Definition

- ColdSpring is a inversion-of-control framework/container for CFCs (ColdFusion Components). Inversion of Control, or IoC, is synonymous with Dependency Injection, or DI. Dependency Injection is an easier term to understand because it's a more accurate description of what ColdSpring does. ColdSpring borrows its XML syntax from the java-based Spring Framework, but ColdSpring is not necessarily a "port" of Spring.

ColdSpring III: Polymorphism

- Components don't care about their collaborators implementation, we have a perfect environment for "swap-ability".
- Swap one implementation of a component out for another and collaborators wouldn't know the difference.
- Perfect for Unit testing: swap dependencies per abstract classes – based on interfaces

ColdSpring: Tiers, Layers



Code Examples: plain

- Scope components in application
- Since we don't want to break encapsulation, each component re-initialize all dependencies
  - Needs to know how to initialize dependencies
  - Multiple instances of the class in memory

Code Examples: setters

- Scope components in application
- Use setters to pass instances of dependencies to each other.

Code Examples: factory

- Scope the factory in application
- Get components from factory
- Only factory knows how to initialize components
- Factory passes itself to components
- Components use factory to retrieve dependencies

Code Examples: factory DI

- Scope the factory in application
- Get components from factory
- Only factory knows how to initialize components
- Factory injects dependencies to components
- Components don’t even know the factory exists

ColdSpring: The Config File

- Beans
  - ID
  - Class
- Dependencies
  - Property
  - Constructor

Code Examples: ColdSpring

- Scope ColdSpring in application
- Pass configuration file to ColdSpring
- Get components from ColdSpring
- Only ColdSpring knows how to initialize components
- ColdSpring injects dependencies to components
- Components don't even know the ColdSpring exists

Code Examples: ColdSpring Autowiring

- Scope ColdSpring in application
- Pass configuration file to ColdSpring – Introduction to Autowiring
- Get components from ColdSpring
- Only ColdSpring knows how to initialize components – Built-in autowiring logic
- ColdSpring injects dependencies to components
- Components don't even know the ColdSpring exists

ColdSpring: Auto wiring

- ColdSpring will inspect constructor and setter methods and identify all dependencies by either type or name
- Pros:
  - Less XML
- Cons:
  - No documentation

ColdSpring: AOP

- Aspect Oriented Programming
- Chris is da man

Thank You

- Questions / Comments?
- Blog: http://www.robgonda.com
- Corp: http://www.ichameleongroup.com
- email: rob@robgonda.com