# Behaviour Chain Pattern / Russian Doll / BMVC

1. Introduce the concept – quickly outline those terms above
   1. It’s smart hybrid of decorator pattern and the chain of responsibility
   2. It’s a compisitonal patern – facilitates use of Open closed principle, SRP and other good principles
   3. Helps to reduce boilerplate code by moving it into the infrastructure
   4. Can be created, removed and combined in many ways
   5. Similar to a linked list
   6. Russian doll model – because each behaviour has control over the invocation of the next
   7. Can use IoC to compose the behaviours – so nasty new(new(new()))) stuff
   8. It is not chain of responsibility because multiple behaviours in the chain are likely to be invoked
   9. Easy to wrap the entire chain with a new behaviour – e.g. to close a session connection
   10. Allows you to add to the end of the chain or to just wrap the chain
2. Show the UML?
3. Talk about the runtime aspect
4. Talk about the configuration
5. Show a contrived example of the pattern
   1. In the search
      1. Have a series of conventional stages that will add behaviours based on the text
6. Show the pattern in action with FubuMVC
   1. There is no controller – you can compose the chain to find actions however you like
7. Show what it allows compared to ASP.NET MVC
8. Mention that it is similar to open rasta too
9. Links to good blog posts
10. Make slides

\*\* Show the order steps at the beginning of the session \*\*

\*\* Need to show conditional wrapping of chains based on some predicate \*\*

## Essential Features

* A behaviour chain is a collection of behaviours
  + The chain is an object and can also provide access and details about the behaviors

## Introduction

Behaviour chains / the Russian doll model, is also known as decorator pattern / CoR hybrid.

It’s similar in the fact that each behaviour holds a reference to the next. In CoR, a matching link is searched for and if it meets the criteria it is executed. So the responsibility is just passed down.

With a behaviour chain, it is a series of actions where multiple behaviours can execute. So whilst CoR is the slot machine, behaviour chains are a pipeline – e.g. process a http request in your favourite web framework – not the inheritance-restricted ASP.NET MVC.

Behaviours can execute around the around the action – before the inner and after the inner. So it wraps it. The flow goes all the way to the bottom, and comes all the way back up. [[[[ \*\*\* Add an image here \*\*\* ]]]]

## Configuration Model

**Wrap**

* “Policies” on the FubuRegistry
* Pass the type of behaviour into WrapBehaviorChainsWith
* \*\*\*\*\* Need to do more research here \*\*\*\*\*

**Enrich Calls With**

* addPolicy
  + Adds the **type** of behaviour to all chains (action calls) satisfying the given predicate
    - Adds it after the last action (Behaviour Node.Add After)
* **IActionBehavior**

## Runtime Model

IConfigurationActions are applied to setup the graph

**BehaviorAggregator**

* Finds all the actions
  + For each one
    - creates a new chain
    - Actions the actions to the end of it
    - Adds the chain to the graph
* **BehaviorChain**
  + Implement as IEnumerable and use decorator-style nested recursion to get each object in the graph
* **BehaviorNode**
* **IBehaviorFactory** “Asked to build the behaviour from the arguments and the chains unique id”
  + **StructureMapContainerFacility** – looks like the implementation
    - It just takes args and id of chain and wraps it in a custom implementation of IActionBehavior – **NestedStructureMapContainerBehaviour**
  + **BehaviorInvoker** then invokes the behaviour

**NestedStructureMapContainerBehavior**

**Invoke –** Get’s the appropriate behaviour and invokes it

* Get’s a nested structure map container – has all the original services from the original container
* Registers all the services with it from the passed in “arguments” of type ServiceArguments
* Get’s the instance of IActionBehavior using the behaviourid
* Invokes that behaviour

## Problem – How does the invoke partial work

* Object node returns a type
* Type is the behaviour
* Instance of behaviour is created
* Behaviour invokes partial
* How does behaviour get partial?
  + So the partial behaviour is just an enum telling what to happen when the partial is invoked
* The **“InsideBehavior”** is not explicity set in code – it must also be done by structuremap

## Working Application Plan

**Create a simple Demo showing just chains and nodes**

1. Create an instance of the behaviour
2. Create a configuration api and build up the chain
   1. Some at the start of the chain – prepend in the behaviour chain – but *has a wrapping effect at runtime* - **WRAP**
   2. Some at the end of the chain – append to the chain, *executes at the end of the model in runtime* – **ENRICH**
3. Create some kind of chain executor class
   1. Takes a behaviour chain and creates the runtime chain of linked Behaviors using dependency injection

**Create a more complex demo Showing the power of Configuration and Conditional Logic etc**

## Why is it Different To CoR

It is a CoR in form, but not necessarily in intent. CoR is usually used to   
find a class that can process a given request.

In Fubu Behaviors, it has the same form as a CoR, but the intent is not to   
find a responsible class, but to allow many classes in the chain to   
participate in the processing of a request (to add context and information   
to the request, or to process the request and contribute to the output).

So in that respect, it's a decorator pattern.  You might call Fubu   
Behaviors a "Chain of Decorators" pattern since it has elements of both   
patterns.  In fact, we originally called them decorators until Steve Harman   
suggested we use the term "Behaviors" and it stuck.

-Chad

## Checklist

* Shown a link of good posts

## Links

* <http://www.google.com/url?sa=D&q=http://lostechies.com/chadmyers/2011/06/23/cool-stuff-in-fubumvc-no-1-behaviors&usg=AFQjCNFX0PV8FO5sE0kFkvB9UzTVqYJg-Q>
* <http://lostechies.com/josharnold/2011/02/01/fubumvc-primer-configuration-vs-runtime/>