

# Creating Custom WCF Behaviors

Rob Daigneau

**VS**Live!

SAN FRANCISCO

March 25-29, 2007

Pre-requisites for this presentation:

- 1) Intermediate to advanced C#, ASP.Net, Web Services
- 2) A Basic Understanding of WCF

*Level: Advanced*



# Overview

- Review concepts from Aspect-Oriented Programming
  - Correlate to features in WCF
- Focus on WCF Custom Behaviors
  - An extensibility mechanism
- Introduce the Interceptors
- Explore ways to inject logic into
  - Proxy and Dispatcher Pipelines

# About Me

- Rob Daigneau
  - Director of Architecture for Monster.com
  - Author of [www.DesignPatternsFor.Net](http://www.DesignPatternsFor.Net)  
[IUnknown@DesignPatterns.com](mailto:IUnknown@DesignPatterns.com)



# Introduction to Behaviors

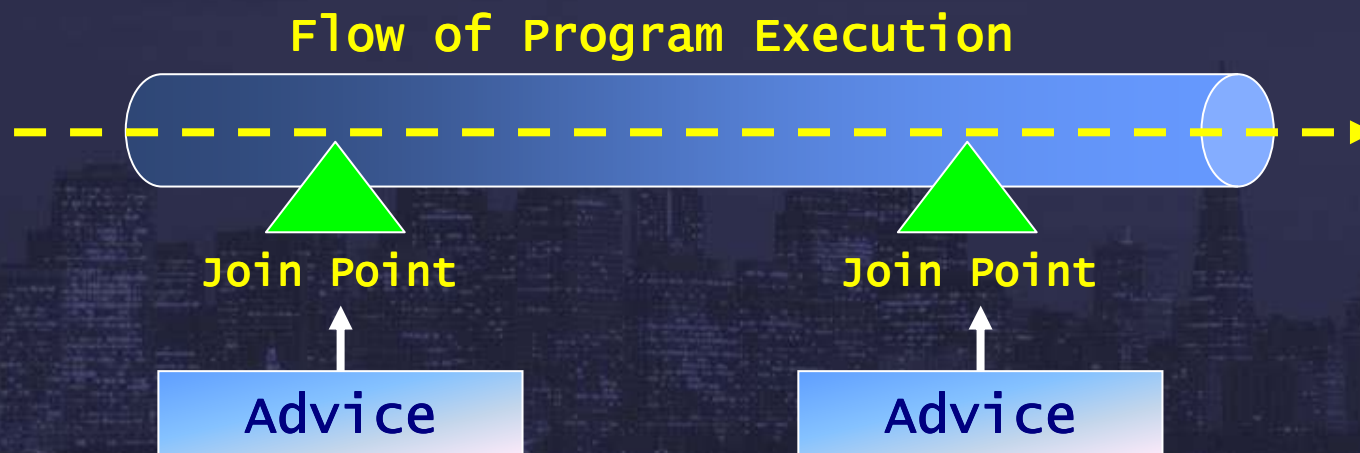
# Looking Back on AOP

- Aspect-Oriented Programming
  - Gregor Kiczales and Xerox PARC
  - AspectJ
    - **An AOP extension to Java, released in 2001**
- What was the motivation?
  - Recognized existence of “Common Logic” that cut across many modules or classes
    - a.k.a. Cross-Cutting Concerns**
  - e.g. Logging, persistence, security
  - Incorporating this common logic into class hierarchies didn’t work so well
    - Would have to copy this logic or use techniques like aggregation or composition across the object hierarchies**

# AOP Concepts Used in WCF

Aspects encapsulate the handling of some concern

- Contains *Advice*
  - Logic or **Behavior** that's applied at a given "join point"
  - May alter the execution of the code
- This code may Intercept program execution before or after the "join point"
- AOP languages provide developers the mean to define when Advice is applied
  - **WCF only allows you to define the Behaviors** (*you can't define join points*)
  - **WCF determines when the Behaviors are executed**



# What Are WCF Behaviors?

- Mechanisms that may alter the runtime execution at the Service Model Layer
  - Proxy or Dispatcher
- Common Behaviors Built-in to WCF
  - Instance Management
    - PerCall, Single, PerSession, Shareable**
  - Concurrency Management, Threading
    - Single, Multiple, Reentrant**
- Custom Behaviors
  - Created by you
  - Injected into the runtime upon execution

# The WCF Architecture and Extensibility

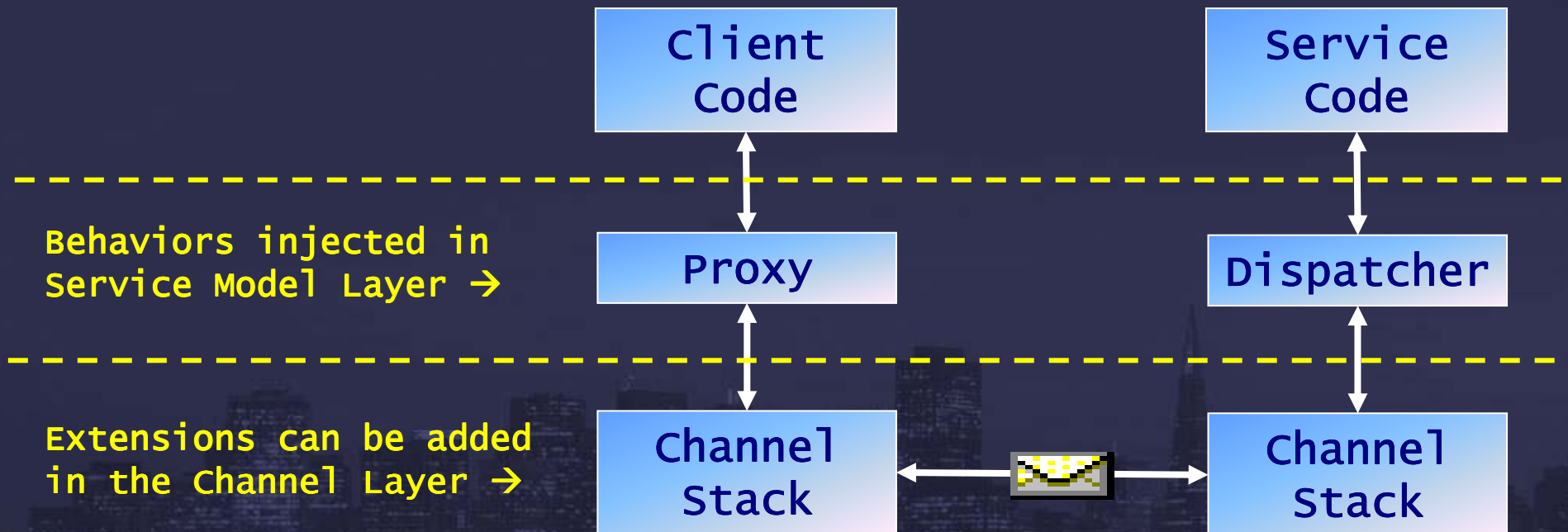
The WCF runtime provides hooks for extensibility at ...

- Service Model Layer

**Proxy and Dispatcher**

- Channel Layer

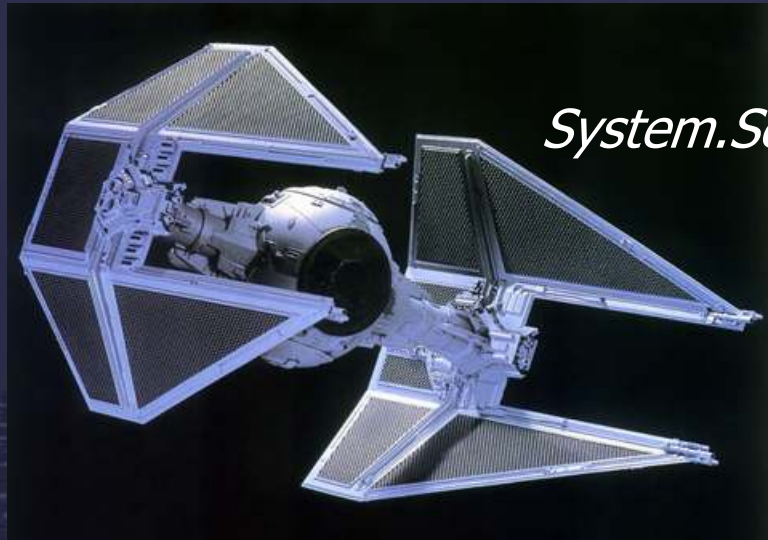
**Client or Service Side**



## Possible Use-Cases for Custom Behaviors

- Validate parameters and messages
- Alter or Transform parameters and messages
- Log information about messages
- Redirect operations to invoke
  - e.g. might want to call some other operation given the content of some body element rather than the normal Action that would be called
- Custom Error Handling
  - Can alter how faults are sent back
- Custom Authorization

# Meet the Interceptors



The Interfaces of the  
*System.ServiceModel.Dispatcher* Namespace

# Using the Interceptors

WCF defines several *Interfaces*

- Namespace = *System.ServiceModel.Dispatcher*



1. Create classes that implement these interfaces

- Provide your own implementation logic

2. Add these **Behaviors** to the WCF Runtime *(more on this later)*

- Before ServiceHost.Open

  - On the service side**

- Before ChannelFactory.Open

  - On the client side**



You have successfully extended the WCF Runtime !!!

- The flow of logic will be intercepted at well-defined “join points”

- The logic in your classes will be executed

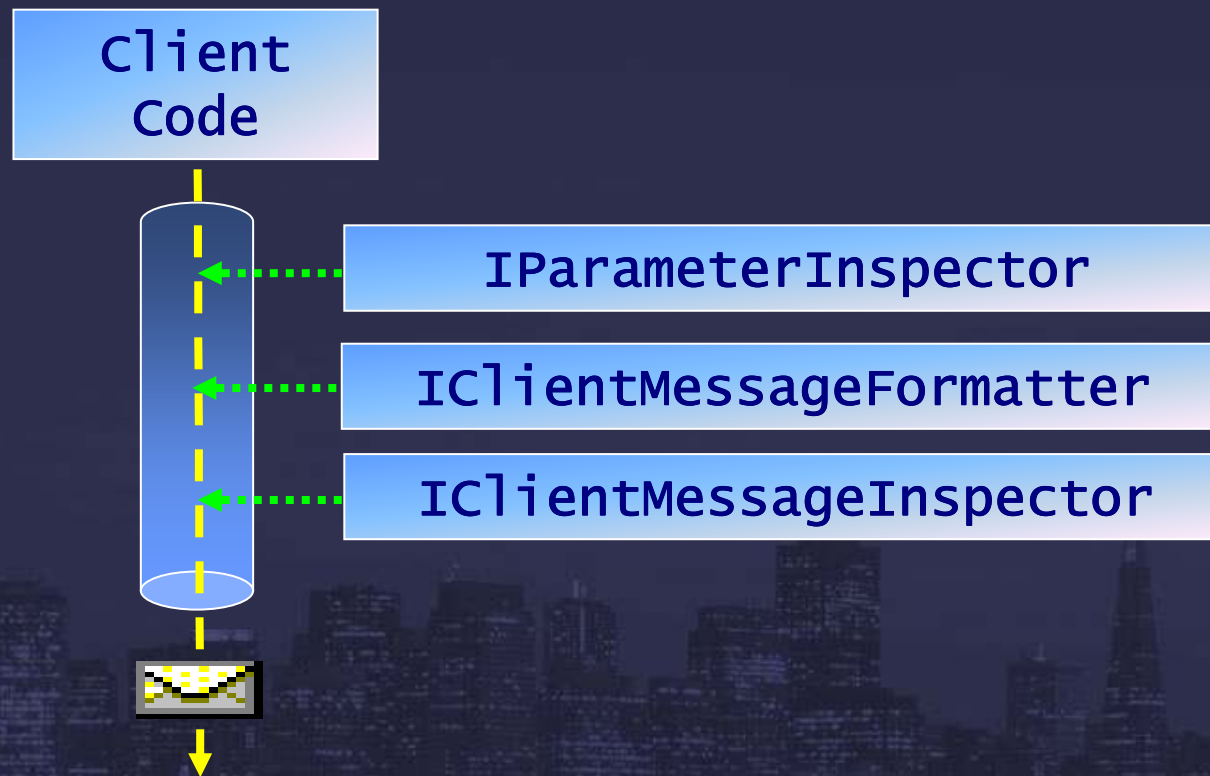
## Potential Use-Case Scenarios for the Interceptors

- IParameterInspector  
**Modify or capture the contents of parameters on client and service side**
- IClientMessageFormatter  
**Custom serialization and deserialization on client side**
- IClientMessageInspector  
**Modify or capture the contents of a message on the client**
- IDispatchMessageInspector  
**Modify or capture the contents of a message on the service**
- IDispatchOperationsSelector  
**Select an alternate operation for a given message**
- IDispatchMessageFormatter  
**Custom serialization and deserialization on service side**

# Behaviors and the Proxy Pipeline

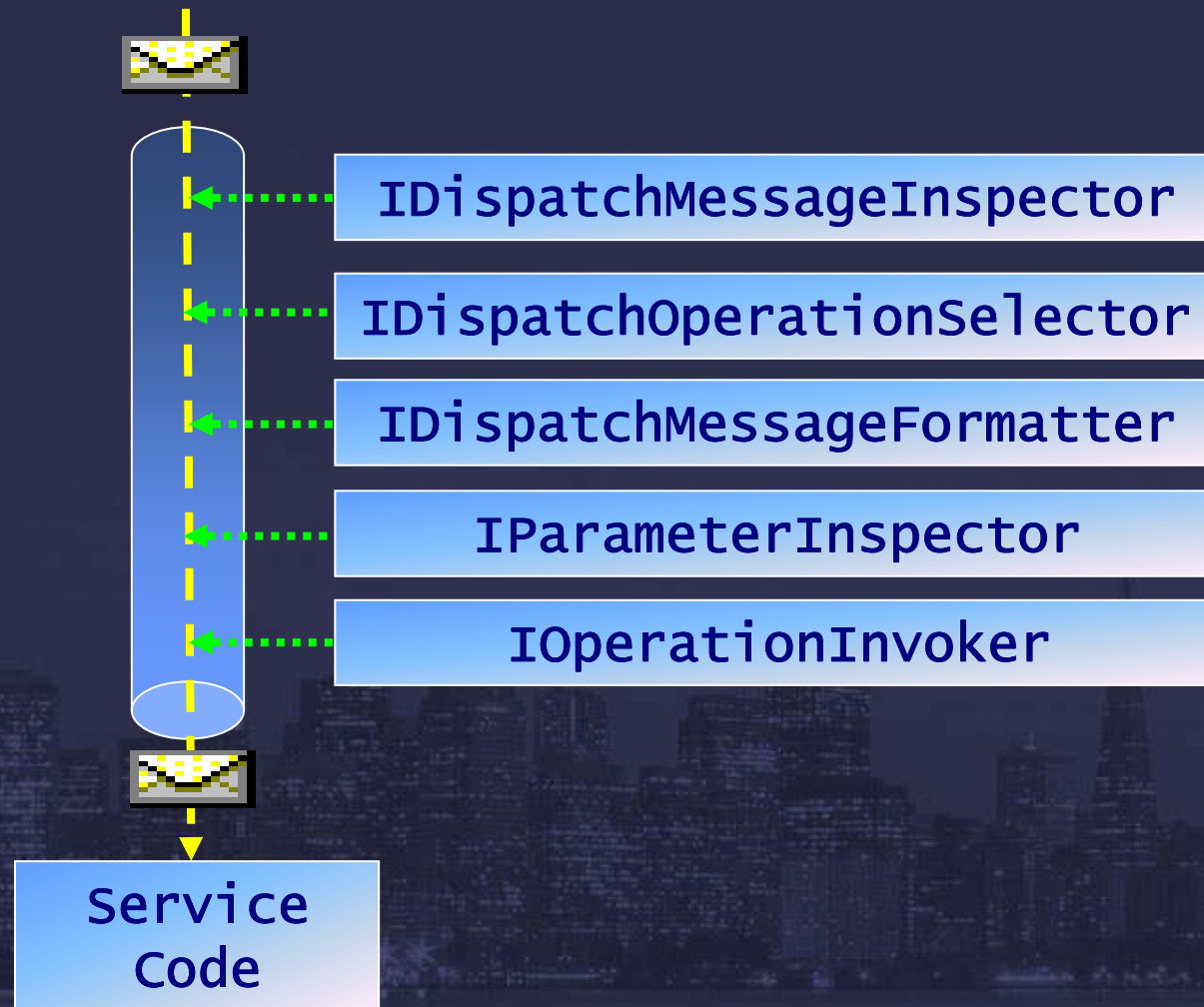
Interceptors can be injected at specific points on the Proxy

- Note: The sequence of execution flows from top to bottom



# Behaviors and the Dispatcher Pipeline

Interceptors can also be injected at specific points on the dispatcher



# Code Review

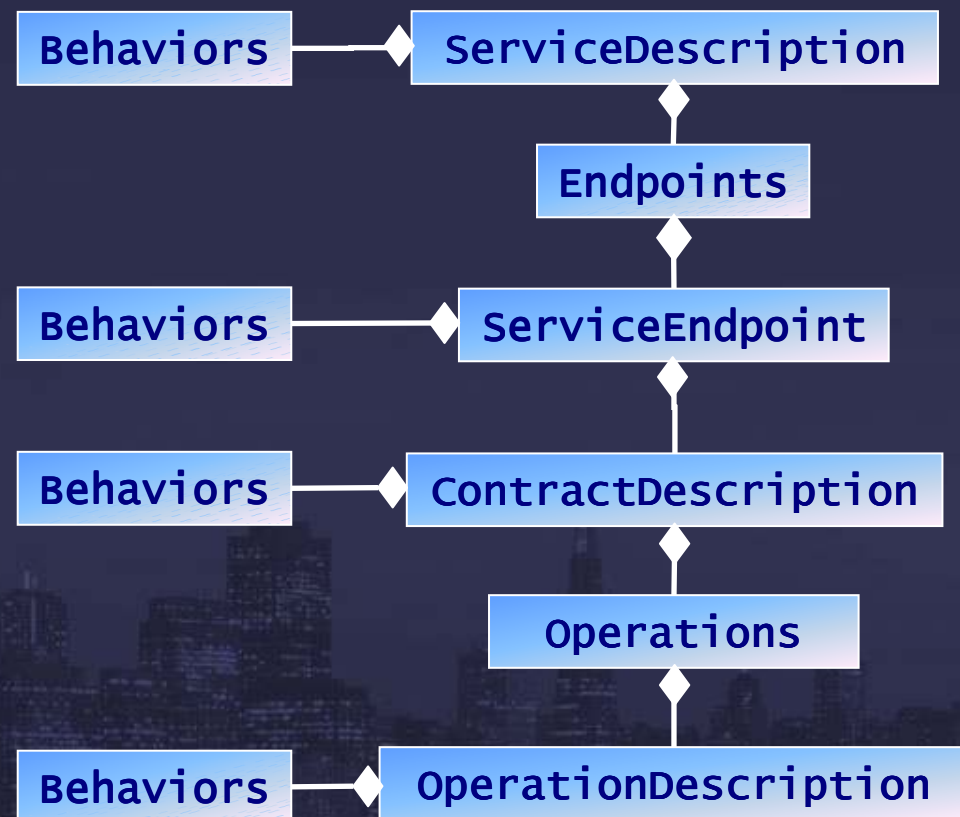
# Loading the Behaviors into the WCF Runtime



# The ServiceDescription Class

The *ServiceDescription* class contains an in-memory representation of the entire service (*a.k.a. Description Tree*)

- Created when ServiceHost opened
  - The *ChannelFactory* on the client side has a similar structure
- Behaviors can be added to various scopes ...
  - Service, Endpoint, Contract, Description
- Behaviors must be loaded
  - On Service Side
    - Before ServiceHost is opened**
  - On Client Side
    - Before ChannelFactory is opened**



# Methods Used to Inject Behaviors

- *System.ServiceModel.Description* contains
  - IServiceBehavior, IEndpointBehavior, IContractBehavior, IOperationBehavior
- These contain methods used to inject behaviors into the WCF Runtime
  - ApplyDispatchBehavior
    - Injects your extensions on the service side**
  - ApplyClientBehavior
    - Injects your extensions on the client side**
- Classes that provide implementation logic for these interfaces function as your loaders

# Scopes

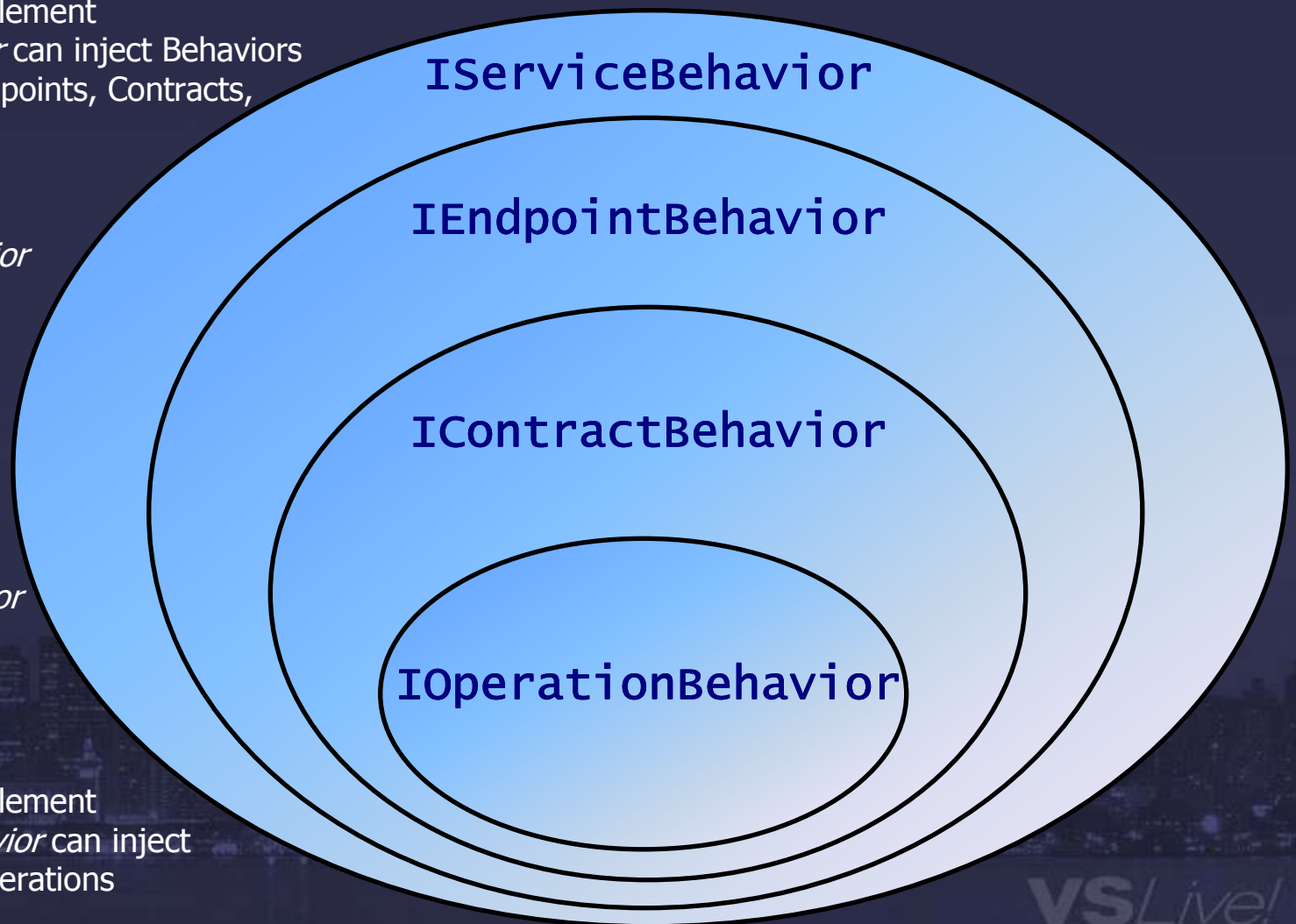
- Classes that implement the *System.ServiceModel.Description* Interfaces can load behaviors whose **scope** may affect one or many levels in the "Description Tree", i.e ...

- Classes that implement *IServiceBehavior* can inject Behaviors on Services, Endpoints, Contracts, and Operations

- Classes that implement *IEndpointBehavior* can inject Behaviors on Endpoints, Contracts, and Operations

- Classes that implement *IContractBehavior* can inject Behaviors on Contracts and Operations

- Classes that implement *IOperationBehavior* can inject Behaviors on Operations



## Behaviors May Injected Through Various Means

Depending upon the interface selected ...

- You can inject the behaviors through configuration, attributes, or programmatic means
- With the exception of *IServiceBehavior*, you can inject the behavior on both the client and service

Name	Applies To	Inject via Configuration File *	Inject via Attribute +	Inject Programmatically
<i>IServiceBehavior</i>	Service	Y	Y	Y
<i>IEndpointBehavior</i>	Service, Client	Y		Y
<i>IContractBehavior</i>	Service, Client		Y	Y
<i>IOperationBehavior</i>	Service, Client		Y	Y

\* If you want a configured behavior, must inherit from *BehaviorExtensionElement*

+ Inherit from *Attribute* if you want to apply a behavior via attribution

# Code Demo

# How Should Behaviors Be Added?

Behaviors can be added in 3 ways ...

- Configuration files
  - Good for adding/removing behaviors after deployment
    - e.g. optional behaviors**
  - Can also insert via *machine.config* files
- Attributes
  - Good when always want to inject a behavior
    - i.e. behavior is required**
  - Can't add, remove, or modify at runtime
- Programmatic
  - Provides a means to inject behaviors conditionally based upon run-time conditions

# Conclusion

## Conclusion

- Showed how some concepts from AOP are implemented in WCF
- Explored WCF Custom Behaviors as a means to
  - Extend the Proxy and Dispatcher Pipelines
  - Apply Logic to Services, Endpoints, Contracts, and Operations

# Resources

- [www.DesignPatternsFor.Net](http://www.DesignPatternsFor.Net)
- Extending WCF
  - <http://msdn2.microsoft.com/en-us/library/ms733848.aspx>