

Implementing SOA Design Patterns with WCF

Rob Daigneau
Chief Architect, SynXis

Prerequisites for this presentation:

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

Level: Advanced



Overview

- Introducing Domain Services
- Anti-Patterns in Domain Service Design
- Message Exchange Patterns
- Patterns for Flexible WCF Services

LAS VEGAS
NEVADA

About Me

- Rob Daigneau
 - Chief Architect for SynXis



- Former Director of Architecture for Monster.com



- Author of forthcoming "SOA Design Patterns" book for Addison Wesley

<http://www.aw-bc.com/catalog/academic/course/0,1143,70071,00.html>

- Release date TBD

- Host of www.DesignPatternsFor.Net

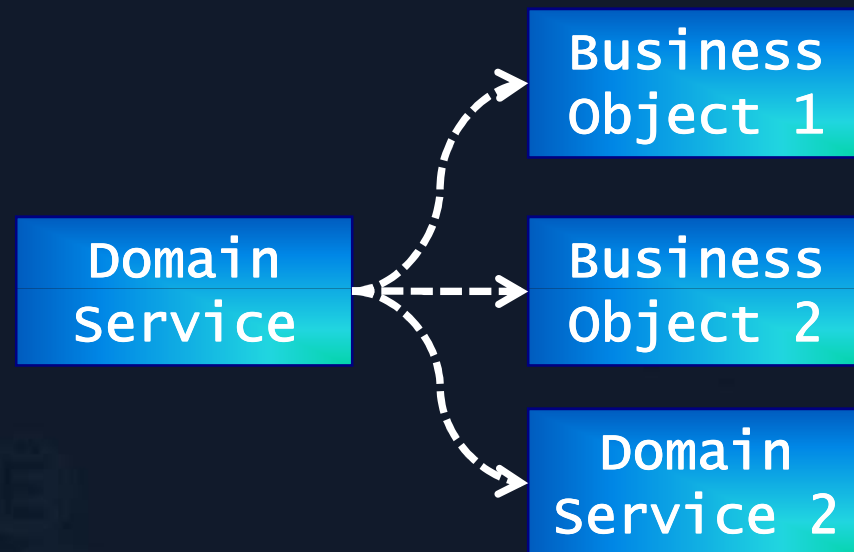
- IUnknown@DesignPatterns.com

Introducing Domain Services

The Foundational Elements of a
Service Oriented Architecture

Domain Services

- Two roles ...
 - Mediator
 - Encapsulates business logic and data management for a specific business problem domain
 - Invokes logic on business objects
 - Façade
 - Simplified and higher-level interface (i.e. coarse-grained) to fine-grained business objects



Best Practices in Domain Service Design

Don't Put Business Logic in the Service!

- Think of the Service Layer as a doorway to your business logic, nothing more
- To allow for independent evolution of the "Service Gateway" from the business logic and data
- Some applications might not want to use services to access business logic



Anti-Patterns in Domain Service Design : Fine-Grained Services Code Demo

- Services that try to be like objects
- One Search Operation for each Search Permutation
- Multiple service calls required to complete a “unit of work”

Message Exchange Patterns

Message Exchange Patterns

- Here are the most common ...

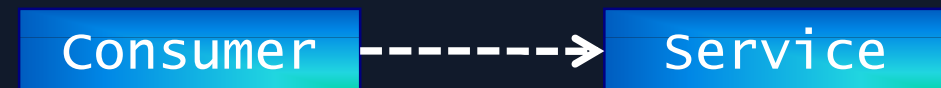
- Request-Response

- Synchronous, consumer waits



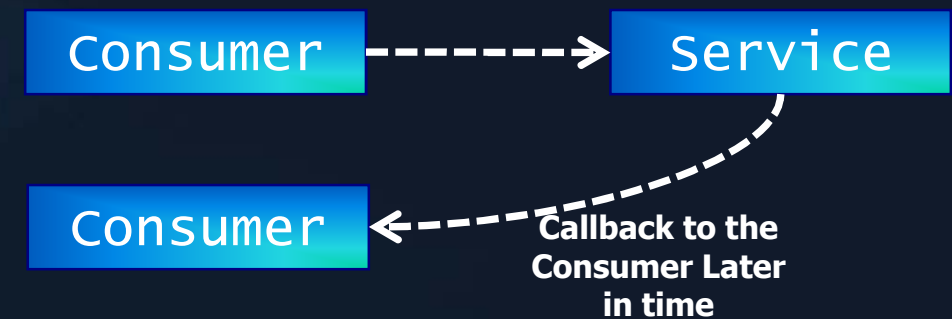
- In Only

- Asynchronous, Input-Only
- Fire & Forget



- Duplex

- Asynchronous with callbacks

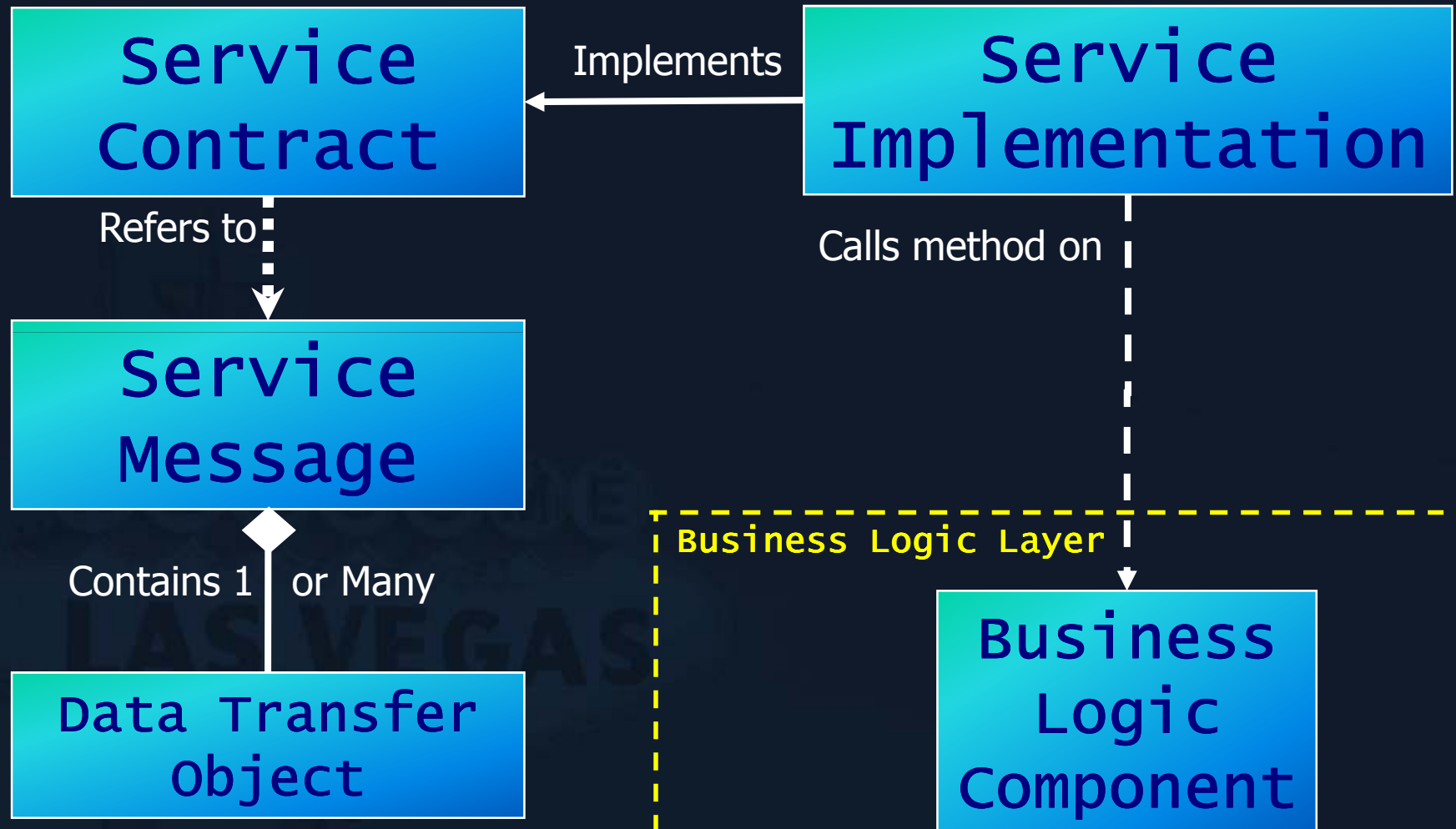


Message Exchange Patterns Code Demo

In-Only,
Duplex

Patterns for Flexible WCF Services

Patterns for Flexible WCF Services



Why Use Service Messages?

- Why not just have a bunch of parameters in the signature of a service operation ?
- Messages should be used because ...
 - Can be extended without forcing clients to get new proxy
 - Always add new data types into "Extension Element" at the end of message
 - Can also extend each DTO
 - Mark new items added to Extension Element as Optional
 - Can be reused across service operations
 - Can be queued for deferred processing

Let's Look at Some Code!

Data Transfer Objects,
Service Messages,
Extension Element,
Service Contract,
Service Implementation Class,
Versioning in Action

Conclusion

- Domain Services
 - Act as Mediators and Facades
- Anti-Patterns in Domain Service Design
 - Fine-grained services leading to chattiness and maintenance issues
- Message Exchange Patterns
 - Request-Response
 - In-Only, Duplex
- Patterns for Flexible WCF Services
 - Data Transfer Objects
 - Service Messages
 - Extension Elements
 - Service Contract
 - Service Implementation Class

Resources

- Introducing Domain Services
 - <http://www.designpatternsfor.net/default.aspx?pid=104>
- Patterns for Flexible WCF Services
 - <http://www.designpatternsfor.net/default.aspx?pid=99>
- When Flexibility in Service Contracts Goes Awry
 - <http://www.designpatternsfor.net/default.aspx?pid=69>
- Looking at Coupling Using the Old Definition
 - <http://www.designpatternsfor.net/default.aspx?pid=38>
- Message Exchange Patterns
 - <http://www.w3.org/2002/ws/cg/2/07/meps.html>
 - http://en.wikipedia.org/wiki/Message_Exchange_Pattern
- Web Service Software Factory
 - <http://msdn2.microsoft.com/en-us/library/aa480534.aspx>
- Patterns of Enterprise Application Architecture
 - Martin Fowler, Addison Wesley

Extras

WELCOME

LAS VEGAS

NEVADA

VSLive!

Domain Services and Enterprise Services

WELCOME

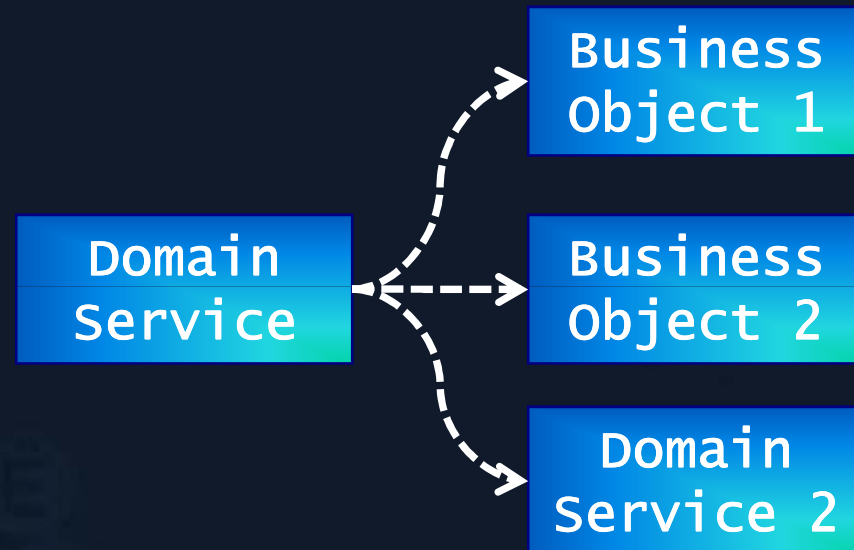
LAS VEGAS

NEVADA

VSLive!

Domain Services

- Two roles ...
 - Mediator
 - Encapsulates business logic and data management for a specific business problem domain
 - Invokes logic on business objects
 - Façade
 - Simplified and higher-level interface (i.e. coarse-grained) to fine-grained business objects



Domain Service Types

- Web Services

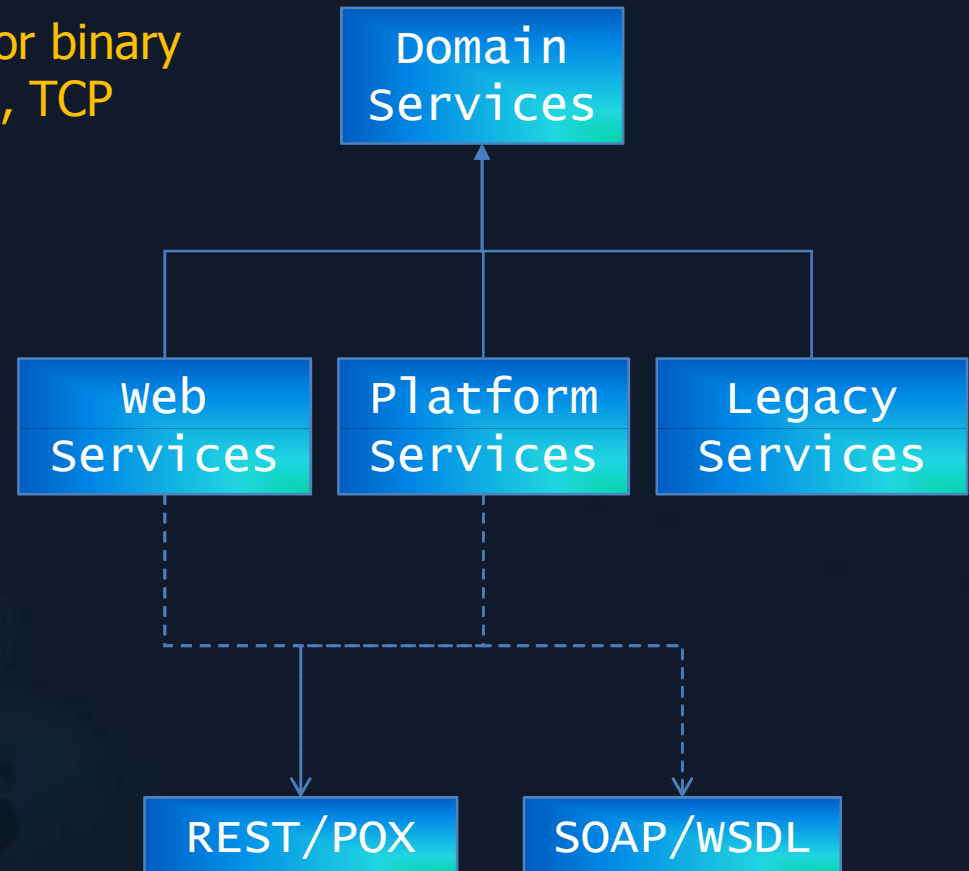
- **Payload** : XML encoded in text or binary
- **Protocols**: HTTP, HTTPS, SMTP, TCP
- **Goal**: Platform Interoperability

- Platform Services

- **Payload** : XML encoded in text or binary
- **Protocols**: non-proprietary or proprietary
- **Goal**: Performance, ability to leverage unique features of the platform

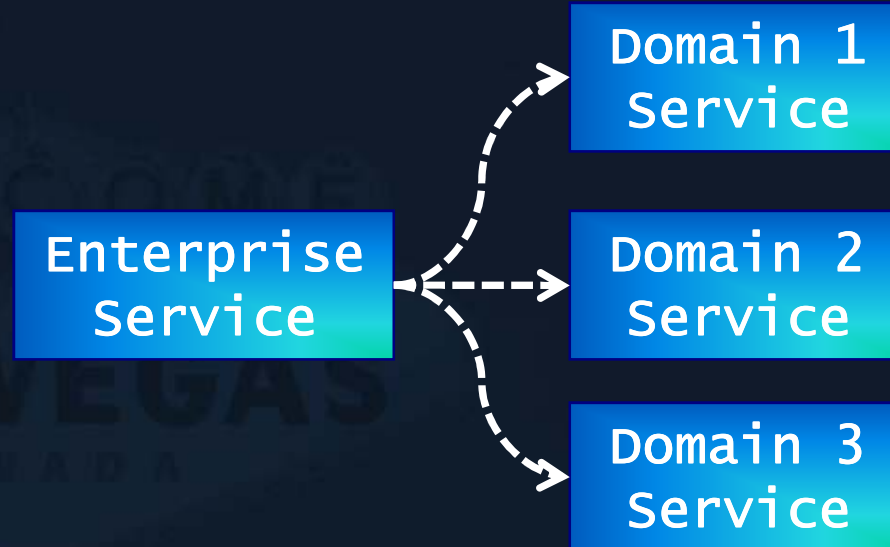
- Legacy Services

- e.g. CORBA, DCOM, RMI, .Net Remoting



Enterprise Services

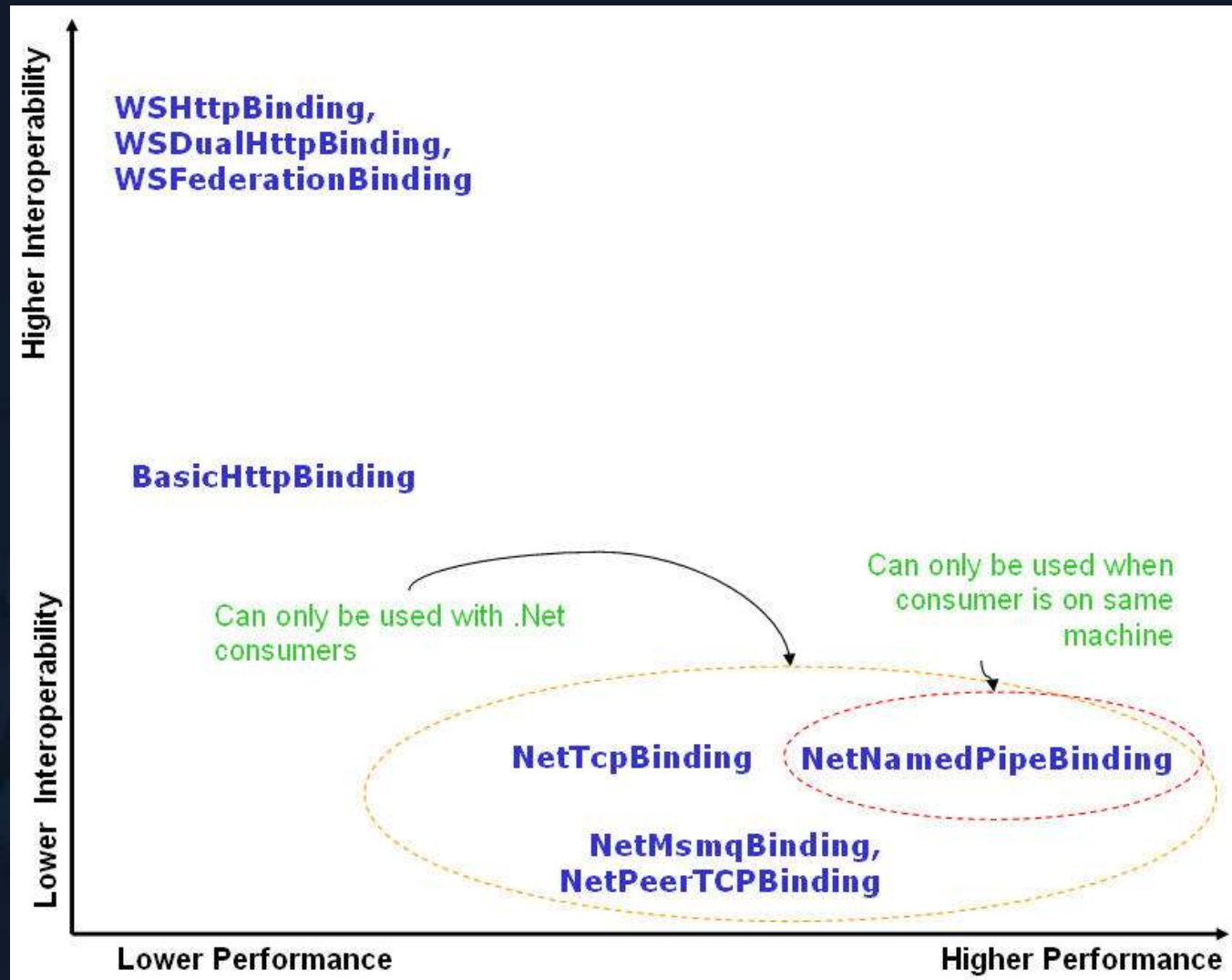
- Composed by assembling multiple Domain Services from multiple domains
 - May leverage Enterprise Service Bus (e.g. BizTalk)
 - **Great for asynchronous communications**
 - **Enables creation of “Event Driven Architectures”**



Tradeoff Decisions

Interoperability vs. Performance

Tradeoffs Between Performance and Interoperability



WCF Deployment Patterns

You can't have it all

Performance

Interoperability



- Want the best performance?
 - Co-locate consumers and services
 - Avoids expensive cross-machine calls
 - Use NetNamedPipes binding
 - Avoids expensive cross-process calls BUT
 - You need to host in either a Windows Service or “Windows Activation Services”
 - The former may not be fault-tolerant, the latter isn't available as of this writing
 - **Consider using HTTP binding with binary encoding**
- Want more interoperability?
 - Use WSHttpBinding or BasicHttpBinding
 - Trade-off on performance

Versioning Service Contracts

Versioning Issues

- Do you want strict or lax validation?
 - Do clients validate the schema? Will they tolerate changes?
 - Should service tolerate missing Data Members?
- Breaking Changes typically caused by ...
 - **Removing/Renaming properties on DTOs or Message types**
 - **Changing the order of types in DTOs or Message types**
 - **Changing the types on DTOs or Message types**
 - **Removing/Renaming operations on services**
 - **Removing/Renaming parameters on operations**
 - **Changing return types of operations**
 - **Changing namespace, endpoint address, or bindings**
- Must create a Major Release when breaking changes occur
 - If you can't avoid breaking changes ...
 - **Create new namespace**
eg. <http://CompanyName/FunctionalArea/ServiceName/Date>
 - **Create new endpoint (i.e. Address, Binding, and Contract)**
 - **May want to deprecate older operations**
 - **Distribute new proxy**

How to Version for Minor Releases: Preserving Forward/Backward Compatibility

- Don't do anything that might cause a breaking change
- Extending DTOs and Service Messages
 - You can create new types at any time
 - Add new types to the end of the DTO or in the Service Message's Extension Element
 - Think "Contract Amendments"**
 - Assign incremental values to the Order Attribute
 - Set the Optional Attribute = True
 - Clients won't need new proxy, won't need to be altered
UNLESS the client uses strict validation
If clients validate, they must ignore new types**
- Extending interfaces
 - You can extend existing interfaces by adding new operations
 - Clients won't need new proxy, won't need to be altered**