Implementing SOA Design Patterns with WCF

Rob Daigneau

VSLive!
SAN FRANCISCO
March 25-29, 2007

Pre-requisites for this presentation:

1) Intermediate to advanced C#, ASP.Net, Web Services

2) Some exposure to Design Patterns

3) A Basic Understanding of SOA

Level: Advanced

Overview

- SOA Overview
  - SOA Myths
  - Why SOA?
- Patterns for Service Orientation
  - Anti-patterns and Best Practices
  - SOA Design Patterns in WCF
  - Versioning
- Review of the Web Service Software Factory
  - If time permits

About Me

- Rob Daigneau
  - Director of Architecture for Monster.com
  - Author of www.DesignPatternsFor.Net

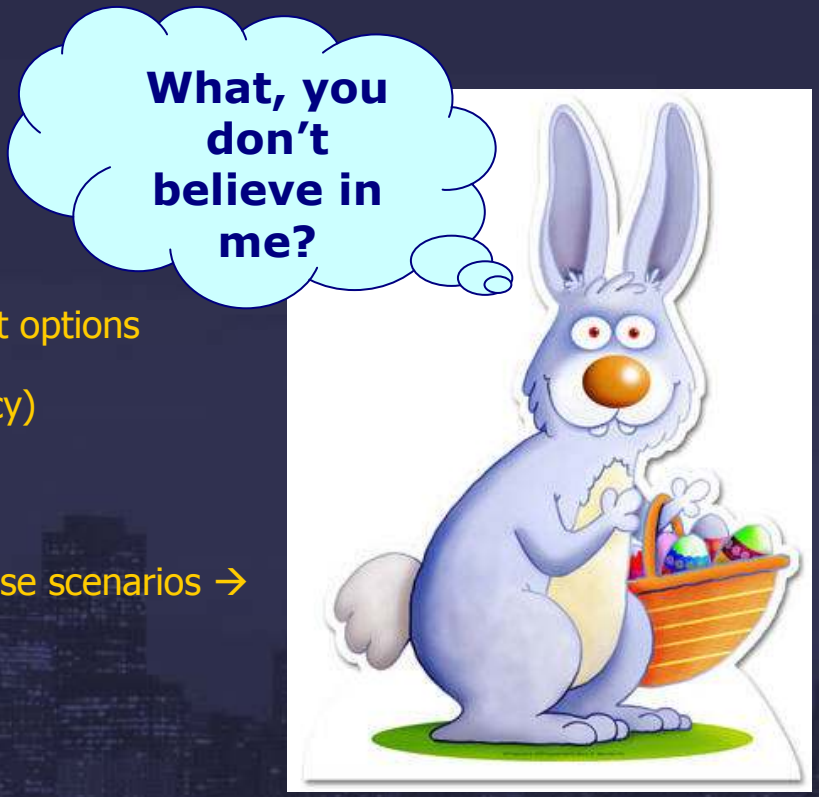
IUnknown@DesignPatterns.com

SOA Overview

SOA Myths

"There is nothing inherent in this architectural approach that ensures adaptability, re-use, productivity, and efficiency" *Me, May 6, 2006*

- Business Agility
  - This stuff ain't quick or easy
  - There's a lot to consider …
    - Design of contracts, versioning, governance
    - Forward and backward compatibility
    - Tradeoffs associated with various deployment options
    - Integration and regression testing
    - Security (authentication, authorization, privacy)



- Reuse

  Reuse implies suitability for a number of use-case scenarios → this suggests generic design for lowest common denominator

- High Availability
  - It depends on your Deployment Architecture

Why SOA?

I've come to believe that these benefits are real …

- Separation of Concerns
- Independent Evolution of Services and Consumers
  - Interface tolerance for consumers
  - Platform independence and interoperability
- Smaller builds and memory footprints for applications
- Highlights the Need for IT Governance
  - The contract is "right in your face"

    We're reminded of what might happen when contracts change
- WCF addresses performance issues for some
  - You don't always have to use HTTP web services

Patterns for Service Orientation

Anti-Patterns in SOA

"*Something that looks like a good thing, but which backfires badly when applied*", Jim Coplien

- Interfaces that lead to "chattiness"
  - Services that try to be like objects …
    Operations that look like properties
  - Database Table oriented operations
    These also expose the database schema and create tight coupling to it
- Instead …
  - Create "coarse-grained" operations
    Get more done with each service call
  - Create operations that are "use-case oriented"
    These may invoke operations on a number of business objects, which in turn execute database operations

Anti-Patterns in SOA

Service operations with "open contracts"

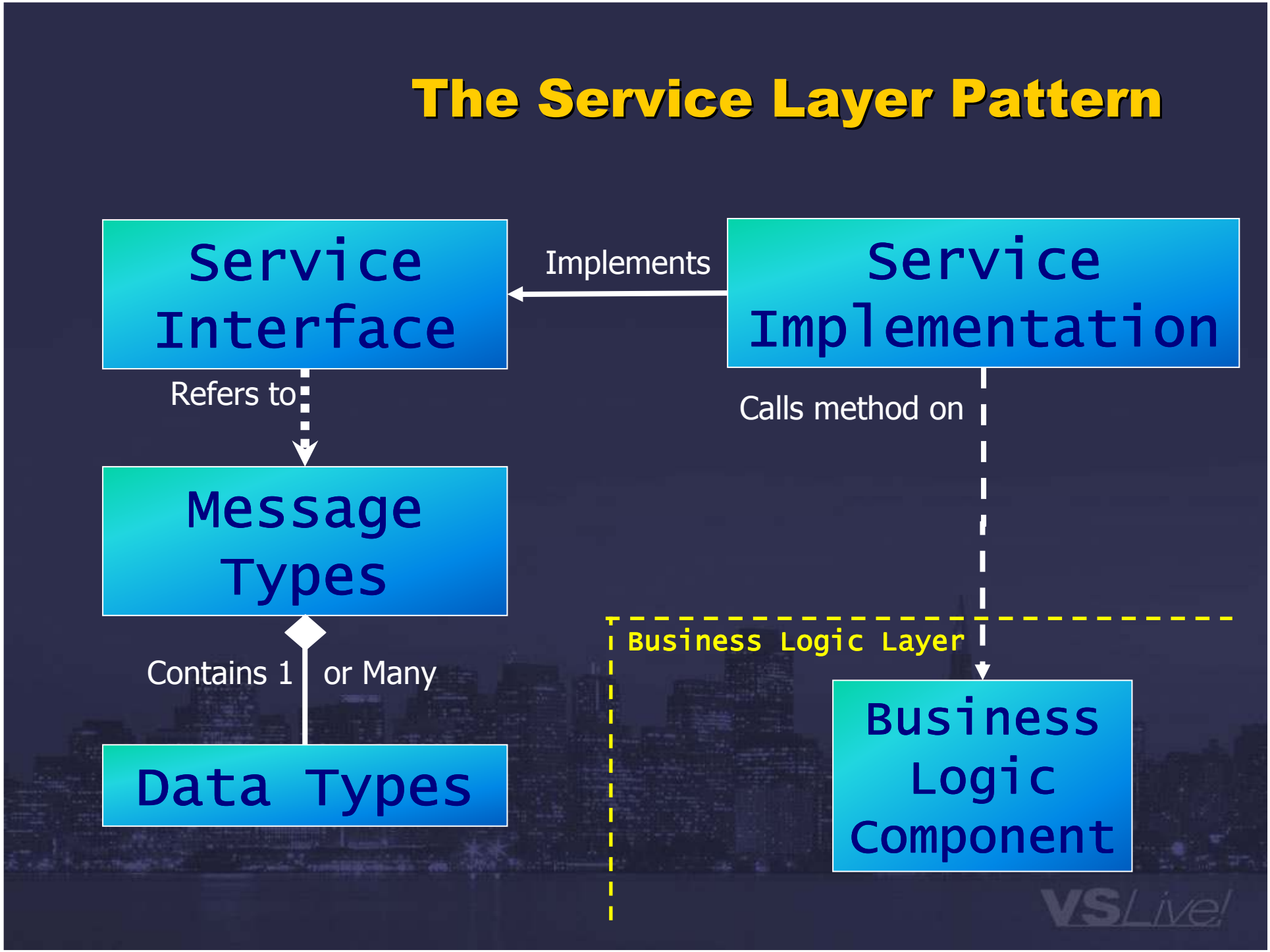
- All of these allow for the submission of member data that might not be known by the operation
  - Use of key-value pairs as parameters
    Unless keys are constrained and defined in the XSD
  - Use of string parameters that contain XML
    Permits unstructured data
  - Use of XMLElement parameters

Best Practices in 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
  (i.e. Service Data Types vs. Business Entities)
- Some applications might not want to use services to access business logic

The Service Layer Pattern



Why Use Message Types?

- Can version and extend messages
  - Always add new data types to end of message

    Think of these as "Contract Amendments"

    Mark these extensions as Optional
- Can be reused across service operations
- Messages can be queued for deferred processing

Let's Look at Some Code!

Data Types,
Message Types,
Service Interface,
Service Implementation Class,
Exposing a Service Endpoint

Demonstration: Versioning in Action

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 data or message types**
  - **Changing the order of types in data or message types**
  - **Changing the types on data or message types**
  - **Removing/Renaming operations on services**
  - **Removing/Renaming parameters on operations**
  - **Changing return types of operations**
  - **Changing namespace, endpoint address, or bindings**
- Recommend Major Release for breaking changes
  - Try to avoid, but if you can't …
    - **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 Data and Message Types
  - You can create new types at any time
  - Add new types to the end of the Data or Message Type

    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

Demonstration:
Using the Web Service Software Factory

Data Types,
Message Types,
Service Interface,
Service Implementation Class,

Conclusion

- SOA Myths and Benefits
  - Business agility is a bogus claim
  - Reuse, availability are possible, but not guaranteed
  - Primary benefits include:
    - Independent Evolution of Services and Consumers
    - Highlights the Need for IT Governance
- Anti-patterns
  - Object-like services, exposing the database, "open contracts", business logic in the service
- Patterns and Practices for Service Orientation
  - Group operations by functional area, get more done with each call
  - Service Layer

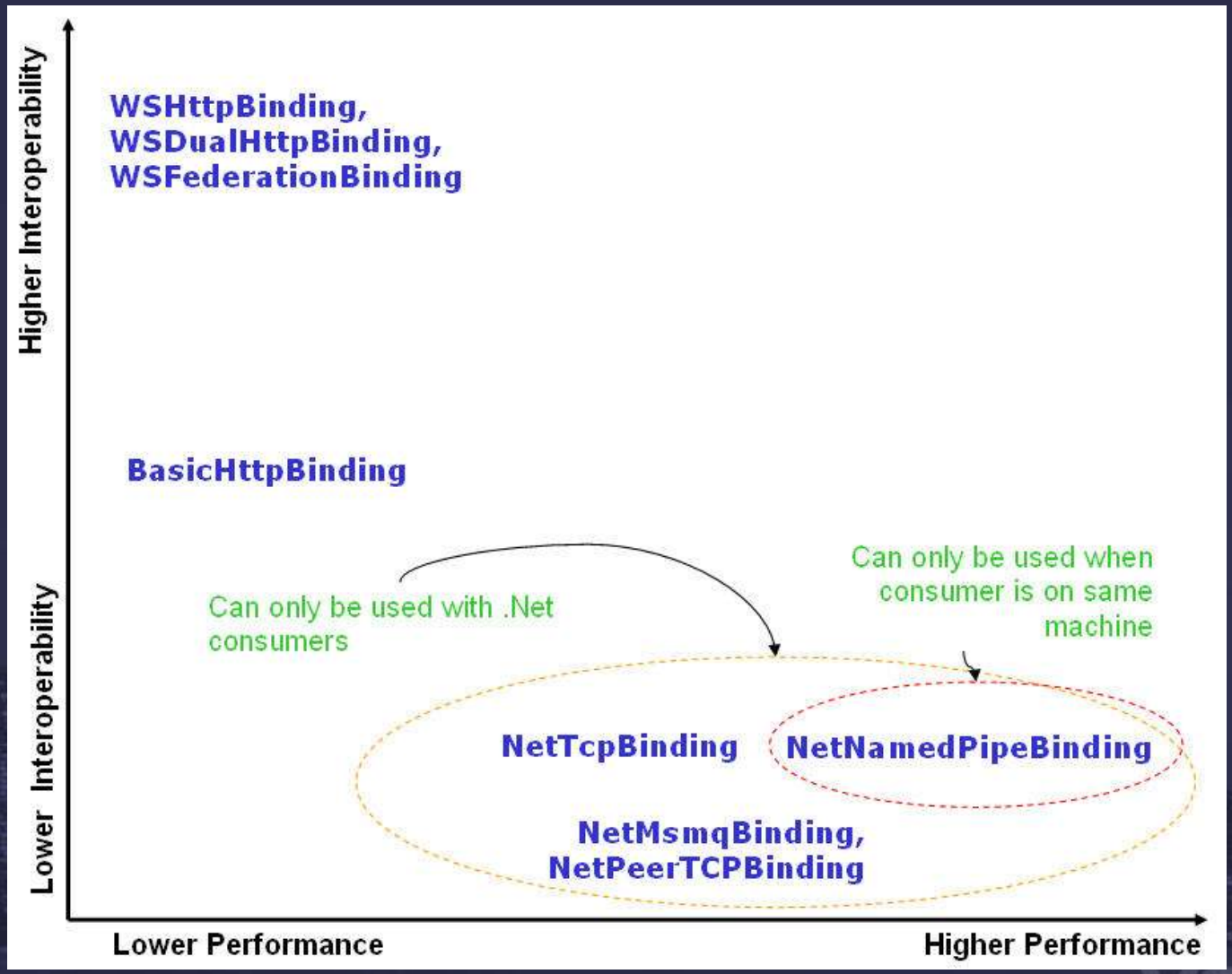
    Data Type, Message Type, Service Interface, Service Implementation
- Versioning
  - Don't cause breaking changes, add types to the end

Resources

- www.DesignPatternsFor.Net
- Web Service Software Factory
  http://msdn2.microsoft.com/en-us/library/aa480534.aspx
- Patterns of Enterprise Application Architecture
  – Martin Fowler, Addison Wesley

Extras

Tradeoffs Between Performance and Interoperability



WCF Deployment Patterns

You can't have it all

- 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