RESTful Services

Building real "Web" services

Overview

- **Web programming models**
  - SOAP vs. POX vs. REST
- **WCF 3.5 support**
  - WebGet/WebInvoke
  - WebServiceHost
- **Web data formats**
  - RSS/Atom (syndication)
  - JSON (Ajax)

SOAP overkill?

- **Some believe SOAP is overkill at times**
  - When the Basic Profile is enough
  - When HTTPS is enough for security (no need for WS-Security)
- **Hence, many use SOAP without WS-***
  - Often referred to as simple SOAP
  - Increases interoperability potential
  - Frameworks/tools hide XML/HTTP

Understanding POX

- **Others just exchange XML messages over HTTP (without SOAP)**
  - Referred to as plain-old XML (POX)
  - Interoperability is virtually guaranteed
  - Requires direct XML/HTTP coding
- **POX applications can also be defined in a REST-ful way**
  - When actions defined by HTTP verbs

Understanding REST

- **In 2000, R. Fielding introduced REpresentational State Transfer**
  - Referred to as REST for short
  - An architecture for building services that build on HTTP
- **Fundamentally different from SOAP**
  - SOAP defines a transport-neutral model focused on custom operations
  - REST defines a transport-specific (HTTP) model focused on resources
- **Builds on a uniform interface and common data formats**
  - HTTP methods: GET, POST, PUT, DELETE, etc
  - Data formats: HTML, XML, JSON, etc

REST vs. SOAP

- SOAP emphasizes verbs while REST emphasizes nouns or "resources"

SOAP

```
getUser()
addUser()
removeUser()
...
getLocation()
addLocation()
...
```

With SOAP, you define custom operations (new verbs), tunnels through POST

REST

```
User { }
Location { }
```

With REST, you define resources, and use a uniform interface to operate on them (HTTP verbs)

XML representation

```
<user>
  <name>Jane User</name>
  <gender>female</gender>
  <location href="http://example.org/locations/us/ny/nyc"
  >New York City, NY, US</location>
</user>
```

Resource orientated architecture

- **With REST, you build resource-oriented architectures (ROA)**
  - You focus on identifying & naming resources (URIs)
  - How to represent them (XML format)
  - How to navigate between them (HREFs)
  - And you use a uniform interface to interact with them
- **HTTP defines the uniform interface (the HTTP verbs)**
  - Constraining operation set simplifies model
- **Primary benefits**
  - Widespread interoperability
  - Scalability!

The importance of GET

- **The Web is primarily built on GET**
  - GET simply means "retrieve" the representation of a resource
  - Should not cause any unsafe side-effects
  - Idempotency expected
  - POST, PUT, DELETE…provide additional semantics, but no guarantees
- **REST naturally embraces the importance of GET**
  - While SOAP largely ignored its importance

The importance of Web formats

- **Reach is also constrained by support for message format**
  - A few Web formats have become ubiquitous
- **Some of the most common formats are RSS and Atom**
  - RSS 0.9x, 1.0 and 2.0 are all common today
  - Atom is emerging as the syndication standard
  - Currently being standardized by IETF (RFC 4287)
  - Atom Publishing API is REST-based
- **Another common format in the Ajax space is JSON**
  - JSON = JavaScript Object Notation

Web programming model comparison

REST

- An architectural model that describes how to identify, represent, and operate on resources
- Embraces HTTP verbs, URIs, and Web data formats
- Action defined by HTTP verbs

POX/HTTP

- A simple ad-hoc model for exchanging XML over HTTP
- Typically GET/POST, but action conveyed in XML
- In some ways, a hybrid between REST & SOAP

SOAP

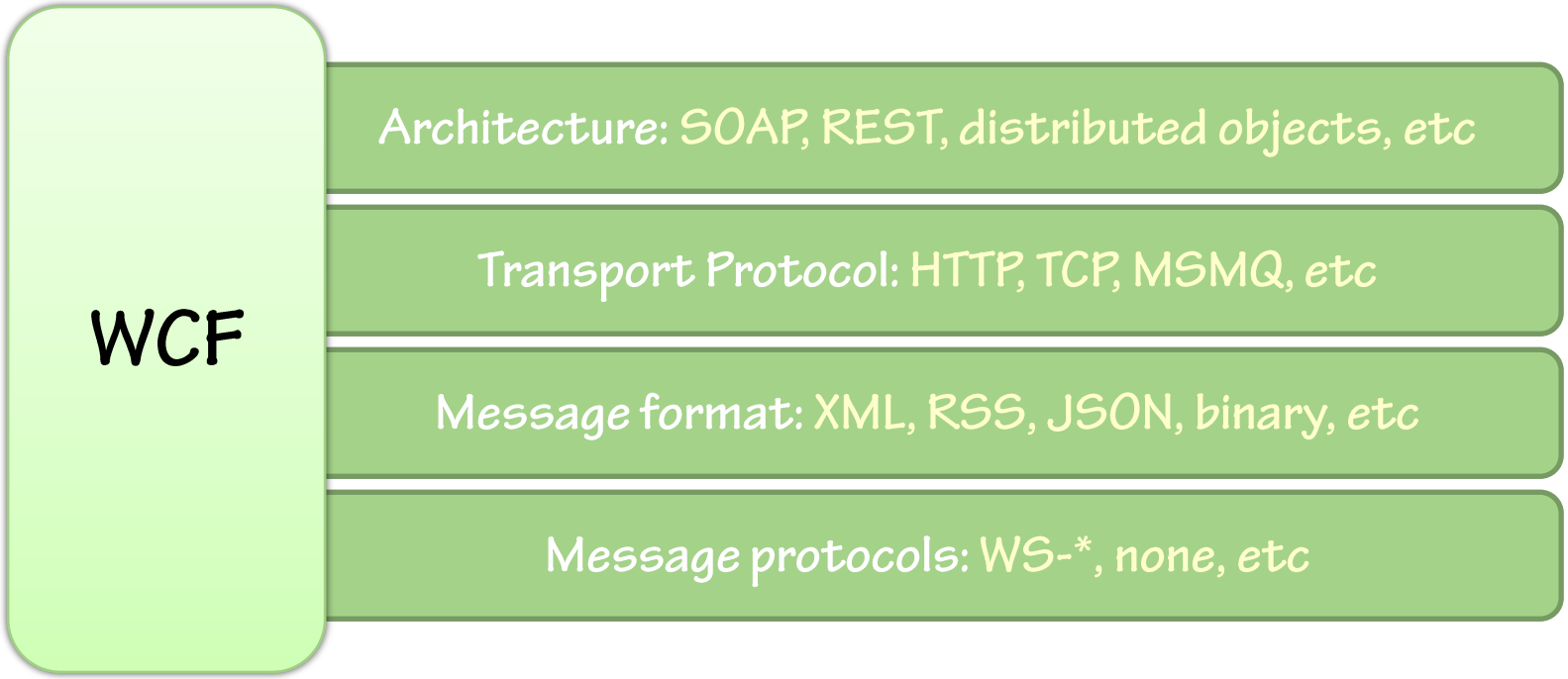
- A model for defining higher-level application protocols
- Can be used with any transport (HTTP, TCP, MSMQ, etc)
- Only POST when used w/HTTP, action conveyed in SOAP

Tradeoffs and guidance

- **Each architecture, style, and format has it's pros & cons**
  - No clear-cut winner all around
- **SOAP + WS-* is largely seen within enterprise scenarios today**
  - In complex situations where you need COM+-like capabilities
  - Can be overkill in many distributed scenarios today
  - Good tool support provided by the big SOAP vendors
- **REST is largely seen in highly-scalable Web-facing scenarios**
  - When you need interoperable, scalable, transfer of information
  - Large Web vendors adopting (Amazon, Yahoo, Flickr, etc)
  - Limited tool support beyond HTTP & XML stacks

Windows Communication Foundation

- **WCF doesn't take sides – it provides a unified programming model**
  - And allows you to use SOAP, POX, REST, whatever data format, etc…



WCF programming styles

- **Most of the built-in WCF bindings use SOAP & WS-* by default**
  - You have to configure them to disable SOAP
- **WCF 3.5 comes with a new Web (REST) programming model**
  - Found in System.ServiceModel.Web.dll
  - Allows you to map HTTP requests to methods via URI templates
- **You enable the Web model with a new binding/behavior**
  - Apply to messaging layer using WebHttpBinding
  - Apply to dispatcher using WebHttpBehavior

Configuring Web-based services

- **WebHttpBinding produces an appropriate HTTP-based channel**
  - Produces an HTTP transport channel
  - You can customize certain settings (cookies, proxy, security, etc)
  - Security modeled by WebHttpSecurity (HTTP vs HTTPS)
  - Produces a WebMessageEncoder (supports XML & JSON)
- **WebHttpBehavior customizes the HTTP-based dispatching logic**
  - Overrides operation selection, serialization, and invocation
- **Both are required to use the new Web features that follow…**

Wiring-up a Web-based service

add the Web binding →

add the Web behavior →

```
...
ServiceHost host = new ServiceHost(typeof(EvalService),
new Uri("http://localhost:8080/evals"));

host.AddServiceEndpoint(typeof(IEvals),
    new WebHttpBinding(), "");
host.Description.Endpoints[0].Behaviors.Add(
    new WebHttpBehavior());
host.Open();  // service is up and running

Console.ReadLine(); // hold process open
...
```

WebServiceHost

- **WebServiceHost simplifes hosting Web-based services**
  - Derives from ServiceHost and overrides OnOpening
  - Automatically adds endpoint for the base address using WebHttpBinding
  - Automatically adds WebHttpBehavior to the endpoint

this host adds the Web binding & behavior →

```
...
WebServiceHost host = new WebServiceHost(
    typeof(EvalService),
    new Uri("http://localhost:8080/evals"));
host.Open();  // service is up and running
Console.ReadLine(); // hold process open
...
```

WebGetAttribute

- **WebGetAttribute maps HTTP GET requests to a WCF method**
  - Supply a UriTemplate to defining URI mapping
  - The UriTemplate variables map to method parameters by name
  - BodyStyle property controls bare vs. wrapped
  - Request/ResponseFormat control body format

URI template

```
[ServiceContract]
public interface IEvalService
{
    [WebGet(UriTemplate="evals?name={name}&score={score}")]
    [OperationContract]
    List<Eval> GetEvals(string name, int score);
...
```

WebInvokeAttribute

- **WebInvokeAttribute maps all other HTTP verbs to WCF methods**
  - Adds a Method property for specifying the verb (default is POST)
  - Allows mapping UriTemplate variables
  - Body is deserialized into remaining parameter

Specify which HTTP verb this responds to

```
[ServiceContract]
public interface IEvals
{
    [WebInvoke(UriTemplate ="/evals?name={name}",Method="PUT")]
    [OperationContract]
    void SubmitEval(string name, Eval eval /* body */);
...
```

UriTemplate

- **System.UriTemplate implements a URI template syntax**
  - Template syntax allows you to specify variables in URI space
  - UriTemplate.Bind fills variables with actual values
  - UriTemplate.Match verifies match and extracts actual values
  - Can use wildcard character "*" to match anything

```
/services/evals?name={name}&detailed={detailed}
```

variables

WebOperationContext

- **Use WebOperationContext to access HTTP specifics within methods**
  - To retreived the current context use WebOperationContext.Current
  - Provides properties for incoming/outgoing request/response context
- **Each context object surfaces most common HTTP details**
  - URI, ContentLength, ContentType, Headers, ETag, etc

WebMessageBodyStyle

- **WCF provides support for two message styles: wrapped and bare**
  - Controlled via WebMessageBodyStyle
- **Wrapped includes a wrapper element named after the method**
  - [MethodName]Response containing [MethodName]Result
- **Bare does not include a wrapper element**
  - The root element is named after the type

Syndication programming model

- **WCF comes with a simplified syndication programming model**
  - The logical data model for a feed is SyndicationFeed
  - You can use it to produce/consume feeds in a variety of formats
- **You use a SyndicationFeedFormatter to work with a specific format**
  - Use Atom10FeedFormatter or RSS20FeedFormatter today

```
[ServiceKnownType(typeof(Atom10FeedFormatter))]
[ServiceKnownType(typeof(Rss20FeedFormatter))]
[ServiceContract]
public interface IEvalService {
    [WebGet(UriTemplate = "evalsfeed")]
    [OperationContract]
    SyndicationFeedFormatter GetEvalsFeed();
...
}
```

allows you to return an Atom or RSS feed

Returning a formatted feed

```
public class EvalService : IEvalService {
    public SyndicationFeedFormatter GetEvalsFeed() {

        List<Eval> evals = this.GetEvals();
        SyndicationFeed feed = CreateSyndicationFeed(evals);

        // figure out what format the client wants
        WebOperationContext ctx = WebOperationContext.Current;
        string format =
            ctx.IncomingRequest.UriTemplateMatch.QueryParameters["format"];

        // return the right type of formatted feed
        if (format != null && format.Equals("atom"))
           return new Atom10FeedFormatter(feed);
        else
           return new Rss20FeedFormatter(feed);
    }
...
```

WebMessageFormat

- **WCF provides support for two primary Web formats: XML & JSON**
  - You control the format via RequestFormat and ResponseFormat

```
[ServiceContract]
public interface IEvals
{
    [WebGet(UriTemplate = "/evals?name={nameFilter}",
        ResponseFormat = WebMessageFormat.Json)]
    [OperationContract]
    List<Eval> GetCurrentEvals(string nameFilter);
...
```

Specify the message format

Enabling Ajax integration

- **Use the WebScriptEnablingBehavior to Ajax-enable a WCF service**
  - Makes JSON the default message format
  - Enables Ajax-style invocations for each [OperationContract]
  - You don't need to use [WebGet] or [WebInvoke]
  - Produces a Javascript proxy for Ajax clients (via base address + "/js")
- **Use WebScriptServiceHostFactory to simplify Ajax hosting**
  - Automatically adds the WebScriptEnablingBehavior

evals.svc (Ajax-compatible)

```
<%@ ServiceHost Language="C#" Service="EvalService" Factory=
    "System.ServiceModel.Activation.WebScriptServiceHostFactory" %>
```

System.Web limitations today

- **UriTemplate isn't capable of handling more complex URI's**
  - E.g., matrix Uris with data separated by commas/semicolons
- **You cannot map UriTemplate variables to [DataContract] fields**
  - UriTemplate only currently supports strings
- **You can't mix the WebScriptEnablingBehavior with UriTemplate**
  - They did not anticipate you using these two features together
- **Server faults always return a "400 Bad Request" to clients**
  - There is no way to change this short of a custom behavior

Summary

- **There are several architectural styles for building services today**
  - SOAP, POX, and REST
- **There are several data formats used by services today**
  - XML, RSS, JSON, MTOM, etc
- **Each architectural style and format has its pros & cons**
  - SOAP is usually better in the enterprise
  - REST is usually better in Web-centric scenarios
- **WCF provides a programming model that accommodates each style**
  - One way to write the code, different ways to wire things up

References

- **HTTP Programming with WCF and the .NET Framework 3.5**
  - http://msdn.microsoft.com/msdnmag/issues/08/01/WCFinOrcas/
- **Creating RESTful Web Services with WCF 3.5**
  - http://www.developer.com/net/article.php/3695436
- **Publishing RSS and Atom Feeds using WCF 3.5 Syndication Libraries**
  - http://www.codeproject.com/KB/WCF/WCF35RSSATOM.aspx