Building WCF REST Services

Strategies for communicating with all kind of clients

developmentor

Objectives

- **Why REST?**
- **Expose REST services**
  - map operations to http verbs
  - utilize URI templates

SOAP == SOA?

- **Systems built on SOA often use SOAP**
  - defined standard
  - built in extensibility infrastructure
  - higher order protocols agreed
  - agreed metadata formats
  - supports arbitrary network protocols

SOAP != SOA?

- **SOAP has issues**
- **Plumbing can be highly complex**
  - e.g. WS-Security
- **Service operations at single endpoint**
  - scaling out problematic
  - sequence of multiple operations not defined
- **“Runs on web” not “part of web”**
  - all messages use POST
  - HTTP caching not supported
- **Client needs special coding to remember place in series of message exchanges**
  - nothing inherent in exchange tells client where they had got to

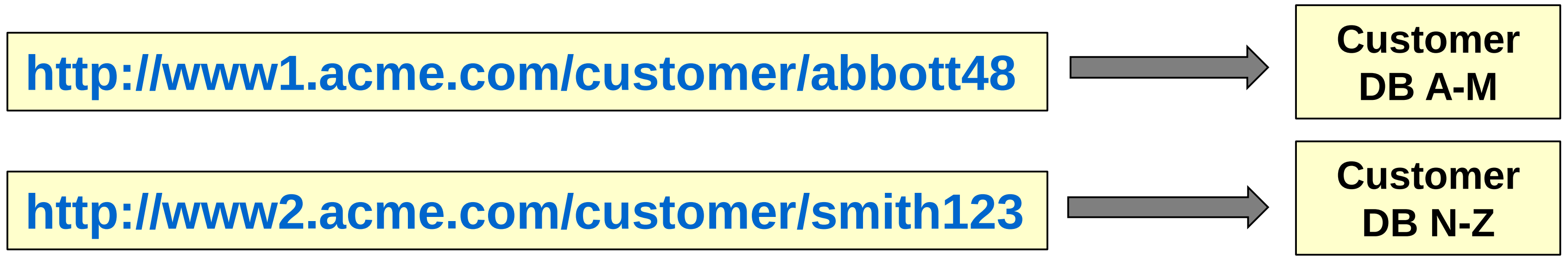
REST = REpresentational State Transfer

- **Alternate way to define services**
  - all operations identified by resource URI and HTTP verb
    - GET = read
    - PUT = insert/update
    - DELETE = delete
    - POST = anything else that doesn't fit the first three
- **Many large scale systems built using REST approach**
  - Amazon S3
  - Google Search API

GET http://www.acme.com/widgets/bypartno?partno=456

REST: Part of the Web

- **Resources identified by URIs**
  - http://www.google.com/search?hl=en&q=REST
  - http://news.bbc.co.uk/2/hi/africa/7322468.stm
- **GET is commonly cached on the client, proxy server or web server**
- **Link from one place to another not necessarily on the same machine**
  - allows expensive operations to be dealt with by different servers/databases
  - allows simple horizontal partitioning of data


http://www1.acme.com/customer/abbott48
Customer DB A-M
http://www2.acme.com/customer/smith123
Customer DB N-Z

REST: Defining the Application Protocol

- **No defined order for SOAP operations**
  - InvalidOperationException
- **REST response message defines the next valid URIs for message exchange**
  - URIs may be data dependent

```
<product>
  <id>123</id>
  <desc>Infinite Improbability Drive</desc>
  <actions>
    <action name="techdetails"
            uri="http://www.heartofgold.com/product/123/techdetails"
            verb="GET"/>
    <action name="purchase"
            uri="http://www.heartofgold.com/basket/add"
            verb="PUT"/>
  </actions>
</product>
```

REST: URI is the State of the System

- **URIs change during message exchange**
  - next possible operations contained in response message
- **Client can stop exchange and continue later**
  - URI contains all contextual information
  - may not be possible in all circumstances
    - e.g. loan offer only valid for 48 hours

REST: Flexible Message Types

- **REST is not bound to XML**
  - URI may contain all data operation requires
  - XML and JSON common for sending complex data
- **Response message can be any HTTP content type**
  - XML
  - JSON
  - JPEG
  - MPEG

REST: Issues

- **No metadata standard**
  - message “specifications” bespoke
  - WADL in early stages
- **No standard for “actions”**
  - format for next available operations bespoke
- **No integration with complex protocols**
  - federated security
- **Currently very little tool support**
- **Wedded to HTTP**
  - not formally but in practical terms
- **Building a good REST API harder than first seems**
  - very easy to end up with RPC like API rather than relying on URIs

Creating REST services in WCF

- **WCF 3.5 fully supports creating REST services**
  - URI templates for building, parsing URI’s
  - new attributes: [WebGet] and [WebInvoke]
  - new binding and behavior: WebHttpBinding, WebHttpBehavior
  - new service host: WebServiceHost
  - config-free deployment: WebServiceHostFactory
  - access headers, set content-type: WebOperationContext
  - support AJAX-style web apps: JSON format
  - new syndication API for RSS and ATOM feeds

Building and parsing URI's

- **UriTemplate to the rescue**
  - accepts string with placeholders in {curly} {braces}
  - Bind methods create a Uri by supplying values
  - Match method lets you extract values
  - also useful apart from WCF

build Uri's based on templates

```
Uri baseAddress = new Uri("http://northwind.com");
UriTemplate template =
   new UriTemplate("customers?id={custId}");
Uri boundUri = template.BindByPosition
    (baseAddress, "ANATR");
```

parse Uri's

```
UriTemplateMatch match =
   template.Match(baseAddress, boundUri);
string id = match.BoundVariables["custId"];
```

Extending the contract [WebGet attribute]

- **Allows mapping HTTP GET to an operation**
  - used to retrieve some resource
  - UriTemplate property maps placeholders in {curly} {braces} to method parameters

maps uri elements to params

```
[ServiceContract(Namespace="http://foo.com")]
interface ICustomerService
{
    [OperationContract]
    [WebGet(UriTemplate="customers?id={custId}")]
    Customer GetCustomer(string custId);
}
```

Extending the contract [WebInvoke attribute]

- **Allows mapping other HTTP verbs to an operation**
  - used to execute some operation
  - Method property specifies HTTP verb (defaults to POST)
    - POST, PUT, DELETE, etc.

maps operation to HTTP verb

```
[ServiceContract(Namespace="http://foo.com")]
interface ICustomerService
{
   [OperationContract]
   [WebInvoke(Method="PUT",
      UriTemplate = "customers")]
   void SaveCustomer(Customer cust);
}
```

Access web specifics [WebOperationContext]

- **Allows access to HTTP headers, content-type, status codes**
  - set status code (NotFound, Forbidden, etc.)
  - set content-type (text/html, image/jpeg, etc.)

```
public Stream GetCustomerPhoto(string custId)
{
   if (custId != "ANATR")
   {  WebOperationContext.Current.OutgoingResponse.
         SetStatusAsNotFound();
      return null;
   }
   WebOperationContext.Current.OutgoingResponse.
      ContentType = "image/jpeg";
   ...
}
```

New binding [WebHttpBinding]

- **Exposes an endpoint as Plain Old XML (POX)**
  - no SOAP envelope in the payload: webHttpBinding
  - was possible in WCF 3.0 but now made easier
  - endpoint behavior also required: webHttp behavior

```
<service name="CustomerService">
  <endpoint contract="ICustomerService"
            binding="webHttpBinding"
            address="http://localhost:1234/..."
            behaviorConfiguration="rest"/>
</service>

<endpointBehaviors>
  <behavior name="rest">
    <webHttp/>
  </behavior>
</endpointBehaviors>
```

endpoint binding

endpoint behavior

New service host [WebServiceHost]

- **Intended for non-SOAP WCF services**
  - extends ServiceHost
  - adds WebHttpBehavior to all endpoints
  - validates compatibility of each endpoint with WebHttpBehavior

```
Type svcType = typeof(CustomerService);
using (WebServiceHost host = new WebServiceHost(svcType))
{
    host.Open();
    Console.ReadKey(true);
}
```

Eliminates need for webHttp endpoint behavior.

Config-free deployment [WebServiceHostFactory]

- **Intended for WCF services hosted in IIS**
  - eliminates need for <system.ServiceModel> in web.config
  - add Factory property to .svc file's ServiceHost directive

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

Eliminates need for system.ServiceModel element in config.

Summary

- **Expose REST services**
  - support non-soap clients (for example, AJAX-style apps)
  - no need for WS-* protocols (security, reliable messaging, etc.)
  - use POX or JSON formatting