

.NET and REST

Richard Blewett

richardb@develop.com

<http://www.dotnetsonconsult.co.uk/weblog2>



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



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
- **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



SOAP and WCF

- WCF was originally designed around SOAP

```
[ServiceContract]
interface IGetBooks
{
    [OperationContract(Action="getbooks",
                       ReplyAction="getbooksResponse")]
    Book[] GetBooks();

    [OperationContract]
    void AddBook(Book book);
}
```



REST and WCF

- **WCF is extremely pluggable**
 - many points of extensibility
- **REST support layered over SOAP infrastructure**
 - encoder strips off SOAP envelope
 - attributes used to associate URI to operation
 - at runtime behavior maps URI to correct contract operation

```
[ServiceContract]
interface IGetBooks
{
    [OperationContract]
    [WebGet(UriTemplate = "books/")]
    Book[] GetBooks();
    ...
}
```



Data Dependent URIs

- UriTemplate used to bind parts of URI to variables using placeholders

```
UriTemplate temp = new UriTemplate("books/{isbn}/");  
Uri baseAddr = new Uri("http://amazonlite.com/");  
  
Uri uri = temp.BindByPosition(baseAddr, "111-222-333");
```

```
Uri uri = new Uri("http://amazonlite.com/books/111-222-333");  
UriTemplate temp = new UriTemplate("books/{isbn}/");  
Uri baseAddr = new Uri("http://amazonlite.com/");  
  
UriTemplateMatch match = temp.Match(baseAddr, uri);  
string isbn = match.BoundVariables["isbn"];
```



Binding Uri to Operation Parameters

- REST Attributes can use UriTemplate for mapping parameters

```
[ServiceContract]
interface IGetBooks
{
    [OperationContract]
    [WebGet( UriTemplate= "books/{isbn}")]
    Book GetBookByISBN(string isbn);
}
```



Enabling PUT and DELETE

- **WebGet** is for **GET** only
- **WebInvoke** used for **PUT**, **DELETE** and other verbs
 - verb specified in `Method` parameter

```
[ServiceContract]
interface IGetBooks
{
    [OperationContract]
    [WebInvoke(Method="PUT", UriTemplate="books/")]
    void AddBook(Book book);
}
```



Beyond the Plumbing

- **REST is about more than plumbing its an architectural style**
 - integrating with the web infrastructure
- **HTTP Status Codes indicate operation result**
- **Content Types beyond XML supported**
- **Security based on HTTP constructs and more**



Status Codes

- **Errors handled with HTTP status codes**
 - 404 resource not found
 - 500 server error
- **Success conditions handled correctly with status codes**
 - 200 OK
 - 201 Created (returning the Uri of the created resource)
- **Support conditional GET and PUT**
 - Supporting ETAGs
 - 304 Not modified
 - 409 resource conflict

```
WebOperationContext ctx = WebOperationContext.Current;  
ctx.OutgoingResponse.StatusCode = HttpStatusCode.Conflict;
```



Content Types

- **POX and JSON supported directly**
 - Attribute of `WebGet` and `WebInvoke`
- **Other data formats can be set directly on response**

```
public Stream GetImage()  
{  
    FileStream fs = File.OpenRead(@"..\..\image.jpg");  
    WebOperationContext ctx = WebOperationContext.Current;  
    ctx.OutgoingResponse.ContentType = "image/jpeg";  
    return fs;  
}
```



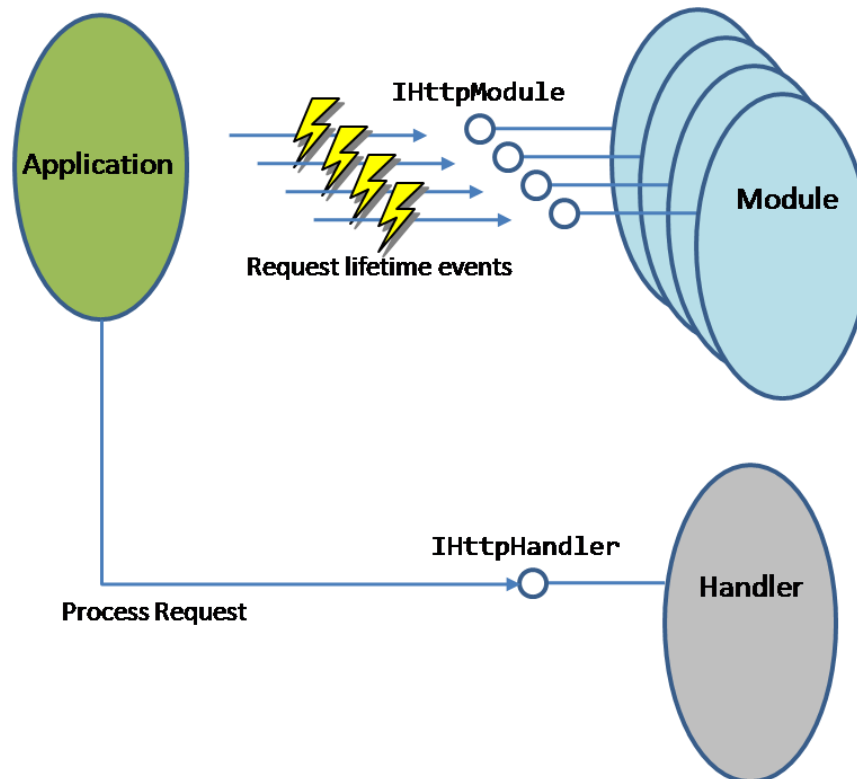
Security

- **Standard HTTP model**
 - SSL
 - HTTP Authentication
- **Website access can integrate with Forms authentication**
- **OpenID can be used for single sign-on scenarios**



ASP.NET HTTP Pipeline Refresher

- Application object for application wide functionality
- Modules for interception based processing
- Handlers to process actual request



Implementing REST with ASP.NET

- **Traditional WebForms not suitable for REST**
 - all requests targeted at .aspx files
- **ASP.NET MVC introduces routing infrastructure**
 - binds handlers to arbitrary URIs
 - routing infrastructure shipped in .NET 3.5 SP1
 - based around `IRouteHandler`

```
public interface IRouteHandler
{
    IHttpHandler GetHttpHandler(RequestContext ctx);
}
```



Mapping Handlers to URIs

- `RouteTable` maps **URI** to **route handler**
 - route handler returns HTTP handler to process request
 - normally set up in `Application_Start`

```
void Application_Start(object sender, EventArgs e)
{
    Route r = new Route("books",
                       new BooksRouteHandler());
    RouteTable.Routes.Add(r);

    r = new Route("books/isbn/{isbn}",
                 new BooksRouteHandler());
    RouteTable.Routes.Add(r);
}
```



Processing the RouteTable

- Module used to process route table and identify HTTP handler from route handler
- Added in web.config

```
<system.web>
  <httpModules>
    <add name="Routing"
        type="System.Web.Routing.UrlRoutingModule,
            System.Web.Routing,
            Version=3.5.0.0,
            Culture=neutral,
            PublicKeyToken=31bf3856ad364e35" />
  </httpModules>
</system.web>
```



Summary

- **REST Supports massively scalable services**
- **REST integrates with how the web works**
- **WCF supports REST based services**
- **ASP.NET supports REST based services**



Contact

Richard Blewett

Email: richardb@develop.com

Blog: <http://www.dotnetconsult.co.uk/weblog2>

