Cst556 Distributed Applications for .NET on Mono

Last Modified February 6, 2019 12:40 pm

5. Web Services with ASP.NET
5.a Web Services: Objectives, Outcomes and References

5.a.1 Outcomes and References

- What are Web Services and how can I develop Web Services and their Clients with ASP.NET both manually and using Visual Studio?
- How is XML used to Describe, Discover and Invoke Web Services: Do I have to manually generate and manipulate these XML forms to create a Web client or Application client that uses Web Services?
- Section Outcomes
  - To know which XML languages (WSDL and SOAP) are used to realize Web Services; to understand each and the role it plays.
  - To know how to develop, deploy and consume Web Services manually and with the Visual Studio IDE.
  - To understand how to create proxies and clients for Web Services.
  - To understand the difference between stateless and state-full (session and application) Web Services; to know how to create each and performance implications of each.
  - To understand how to create and use Web Applications, and to know their relationship to Web Services.
5.a.2 References

- C# Web Services (course textbook) Chapter 3, 4, 5 - Web services with .NET, with ASP.NET and consuming ASP.NET Web Services
- GotDotNet.com tutorial on Web development with .NET
- The notes and examples use xsp4, which is the development web server provided with Mono complete installation.
- C# for Java Developers by Jones and Freeman, Microsoft Press, Chapter 19 Introduction to XML Web Services
5.a.3 Index of Example Web Service Programs

- Services only: Section 5.d.1 Sample Function Calculation Web Service
- Stateless: Section 5.d.2 Sample Stateless Web Service: SailboatCalculs
- Singleton: Section 5.d.6 Developing Web Services Using development webserver
- Session: Section 5.e.3 Session Object
- Web service proxy: Section 5.f.1 Creating a Proxy Class
- Web application client: Section 5.g.4 Example Web Application and Web Service
5.b Motivation

5.b.1 Motivation for Web Services

- **Web Services**: call methods of an object over the internet or through http
  - Services useful to serveral users or applications
  - Multiple cost models: free, subscription, per use
  - Available across multiple platforms, languages and applications

- **Two of the ways** .NET framework provides for building Web Services
  - **ASP.NET** - an active server page called through http
  - **.NET Remoting** - hosting web services in an application (not through IIS)

- **Example Web Services**
  - Credit card purchase
  - Mapping and directions
  - Travel related services: search, cost, schedules, and reservations
  - Shipping tracking
  - Retail and services information: Which vendors offer a product, at what price; what’s on sale?
5.b.2  Web Services in the Context of Components

- **Code reuse** was an original objective of components
  - The code used on one project is adapted for use in another project
  - Reuse on average costs about 70% of the cost of redeveloping
- **Interface** based programming and **Object Orientation**
  - A component publishes a well-defined interface that other project parts use to govern interaction
  - Ada, for example, included module specifications as separately compilable from the module body
  - Vendors produce COM components and distribute them to customers to install on their machines. Leads to multiple versions of the component (DLL file) on a single machine.
- Web Service component model **decreases the replication of code** to customers
  - Vendor exposes the functionality as a service rather than distributing DLL
  - Customers access the Web Service and are charged accordingly
  - The vendor may provide updates by modifying its service only
5.b.3  The Vision of Web Services

• From Roger L Costello (Aug 2002), see:

• “The vision of SOAP, WSDL, and UDDI may best be characterized by this example: a program needs to determine the energy consumed by a 1600 BTU air conditioner in a 10 hour period. The program will find, connect, and interact with a Web service to obtain this information. It will dynamically interact with a UDDI operator which will return a list of Web services that have implemented the A/C energy calculating functionality. Based on factors such as performance, reliability, and trust the program will choose one of the Web services. Using the WSDL document that UDDI (or some other registry) returned the program will dynamically learn how to connect to and use the selected Web service. The program will then interact with the chosen A/C energy calculating service by sending and receiving XML documents wrapped in a SOAP envelope.”
5.b.4 Web Services: Related and Competing Technologies

- **Distributed COM** by Microsoft
  - Multi-language, single platform
  - Constrained functionally by not including complete object model

- **Common Object Request Broker** by Object Management Group
  - Multi-language, multi-platform,
  - Sacrificed ease of use and functionality

- **Remote Method Invocation** in SUN’s Java
  - Single language, multiple platform
  - Easy to use and conceptually clean, although with dynamic class loading security mechanisms must be carefully addressed
  - Incurs burden of internet-wide reference counting and garbage collection
5.b.5 ASP.NET Web Services versus .NET Remoting

• .NET Framework includes **two mechanisms for distributed application** development:

  - ASP.NET **Web Services**
  - .NET **Remoting**

• **Remoting** can use different transport mechanisms

  - A native binary TCP based protocol
  - SOAP and HTTP
  - Object references and a proxy object are utilized by the client
  - Remoting is a fast (binary formatted remoting is twice as fast as web services) solution for .NET clients and servers (no interoperability).

• **ASP.NET Web Services** expose web methods using SOAP and HTTP through an IIS web server.

  - Developed based on open standards (**WSDL**, **SOAP**)
  - Intended to allow across development platforms (for example, Java client accessing an ASP.NET Web Service).
5.b.6 Key Terminology and Supporting Standards

- **XML Web Services** - An Object component exposed to the internet via http.
- **SOAP** - Simple Object Access Protocol; XML based protocol with 3 parts
  - Envelope describing what is in a message and how to process it,
  - Encoding rules for transporting user-defined datatypes,
  - Convention for describing remote method calls and returns.
  - See: [http://www.w3.org/TR/SOAP/](http://www.w3.org/TR/SOAP/)
- **WSDL** - Web Services Description Language is a schema defining:
  - Whats and wheres of the service (URL, Name and Documentation)
  - Methods supported by the service, including parameters and return types
  - Details for accessing the service via SOAP, HttpGet and HttpPost
  - See: [http://www.w3.org/TR/wsd](http://www.w3.org/TR/wsd)
- **UDDI** - Universal Description, Discovery and Integration
  - Registers web services so clients can discover, connect to and use them.
  - Includes: white (detail), yellow (advertise) and green (reference) pages
  - See: [http://uddi.xml.org/](http://uddi.xml.org/)
5.c Overview of Web Services

5.c.1 View of ASP.NET Web Service from Server Side

1. http request

2. ASP.NET creates object specified in ASMX file

3. ASP.NET calls appropriate method

4. object returns value to ASP.NET

5. results returned via http and XML

Object with web methods

MyWebMethod()
5.c.2 Client Call to a Web Service

- Conceptually, **proxy** classes work for **WebServices** the same as they do for **.NET Remoting**.
- ASP.NET supports three protocols for calling Web Services
  - **SOAP**: parameters are passed with a SOAP request and results are returned with a SOAP result
  - **HttpGet**: parameters are passed in the url. The return is XML
  - **HttpPost**: parameters are passed in the http request message
- **Parameters** types for a method can determine which protocol is best to use.
  - If the WebService takes **complex** parameters then only **SOAP** can be used. **http request** sends parameters as name value pairs - quoted strings.
5.c.3 Example Soap Request and Response

• Here is an example of a request for a SOAP WS request

POST /SailboatCalcsWS.asmx
SOAPAction: http://pooh.poly.asu.edu:2330/hullSpeed
Content-Type: text/xml; charset=utf-8
Content-Length: string
Host: string

<?xml version="1.0" encoding="utf-16"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <hullSpeed xmlns="http://pooh.poly.asu.edu:2330/">
      <lwl>double</lwl>
    </hullSpeed>
  </soap:Body>
</soap:Envelope>

• Here is an example of a response for a SOAP WS request

HTTP/1.0 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: string

<?xml version="1.0" encoding="utf-16"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <hullSpeedResponse xmlns="http://pooh.poly.asu.edu:2330/">
      <hullSpeedResult>double</hullSpeedResult>
    </hullSpeedResponse>
  </soap:Body>
</soap:Envelope>
5.c.4 Example of HTTP GET Request and Response

- Using the GET protocol, the method name and parameters to a web service can be passed in the URL. Note that not all web servers process this format, e.g., Cassini.

- To call the **hullSpeed** method of the **SailboatCalcsWS** Web service, providing it with a double value for its **lwl** parameter use:

- The response from this call would be:

  HTTP/1.0 200 OK
  Content-Type: text/xml; charset=utf-8
  Content-Length: string

  <double xmlns="http://pooh.poly.asu.edu:2330/">6.242779393472</double>
5.d Manually Building and Deploying a Web Service

5.d.1 Sample Function Calculation Web Service

• **Two** primary **aspects** of creating a Web Service
  - **Marking** the appropriate parts of a C# implementation with attributes
  - **Deploying** the service with IIS by creating the appropriate description and library files. See the examples in `webSvcOnly.jar`

• Steps to creating the Web Service
  - Create the C# code for the service.
  - Create the **.asmx** file
  - Add **WebService** and **WebMethod** directives to the code as appropriate
  - Deploy the web service by creating a virtual IIS directory that points to the Web Service deployment directory, or using Cassini.

• The **.asmx** file.
  - May contain one or more classes in any .NET language or may refer to those classes (**Codebehind** element) as defined in other files.
  - Deployment directory must contain the **.asmx** file (and C# file), a bin subdirectory must contain a **dll** assembly of the WebService.
5.d.2 Sample Stateless Web Service: SailboatCalcs

- C# source. See: sailboatCalcsWS.jar

```csharp
[WebService(Namespace="http://localhost",
    Description="Sailboat performance calculation service."])
    public SailboatCalcsWS() {

    }

    [WebMethod(Description="Calculate the displacement to waterline length ratio D/L.")]
    public double displacementToLength(double disp, double lwl) {
        double dOverL = ((disp/2240.0)/Math.Pow((0.01*lwl),3.0));
        return dOverL;
    }
}
```

- ASMX file - the Web Service Directive

```
<%@ WebService Language="c#" Codebehind="SailboatCalcsWS.cs" Class="SailboatCalcsWS"%>
```

- Compile into a library manually or use ant buildservice

```
csc /out:bin\SailboatCalcsWS.dll /target:library SailboatCalcsWS.cs
```

- Deploy - configure the web-server IIS so that it recognizes and serves the web Service. (Manually create a Virtual Directory under IIS Manager.)

  - Developing, deploying, testing and publishing Web Services requires an approach to software engineering with particular characteristics.
5.d.3 Approaches for Deploying the Web Service

• Configure the Windows web-server IIS so that it recognizes and serves the web Service. (Manually create a Virtual Directory under IIS Manager.)
  - With the World Wide Web publishing service running, use: ControlPanel --> AdministrativeTools --> Internet Information Svcs Add a new Virtual Directory to the Default Web Site that points to the deployment directory.

• On Linux, Windows or MacOS, use the Mono open source .NET platform. Mono’s xsp is a development web server. See: http://www.mono-project.com/
  - Use the apache web server together with mod-mono extension.
  - Command line: xsp4 --port=2330 --applications=/./deploy
5.d.4 Web Service Attribute

  - Used to mark a class containing web methods. Need not be present.
  - Describes the service that will be offered to clients.

- Properties of `WebServiceAttribute`
  - **Description** - a string characterizing the XML Web Service functionality.
  - **Name** - the name of the XML Web Service
  - **Namespace** - the default XML namespace for the web-service. Should specify the web address of the deployed web service parent directory. Default is: `http://tempuri.org`
  - Although many namespaces look like URLs, they don’t need to point to Web resources.

```csharp
[WebService(Namespace="http://localhost/Cst556WebSvcEgs",
    Description="Sailboat performance calculation service.",
    Name="SailboatCalcs")]
public class SailboatCalcs : System.Web.Services.WebService { ... }
```

- **Note:** the syntax means that there is a parameterless constructor for `WebServiceAttribute` and the class has 3 public properties: `Namespace`, `Description` and `Name`. 
5.d.5 Web Method Attribute

  - Used to mark a `instance methods` of a class to indicate they should be exposed as web methods.

- **Properties**
  - **BufferResponse** - If true, the response to an XML Web service request will be buffered until executing the method completes, sending back a single response. Default is buffered.
  - **CacheDuration** - How long (seconds) is the response to a Web service request kept in memory waiting for another request with the same arguments.
  - **Description** - a string characterizing the XML Web Method functionality.
  - **EnableSession** - `true` or `false` (default is false) indicating whether there is any session context available for this Web Method.
  - **MessageName** - the name exposed to clients for the method. Default is the name of the method from the C# code.
5.d.6  Developing Web Services Using development webserver

- Example of Hashtable web service with a variety of client types
  - **Java Swing** application under Tomcat and Axis. For references and installation information for Tomcat and Axis see Cst427 notes: [http://pooh.poly.asu.edu/Cst427/ClassNotes/WebSvcs/cnWebSvcsInJava-3.html](http://pooh.poly.asu.edu/Cst427/ClassNotes/WebSvcs/cnWebSvcsInJava-3.html)
  - C# .NET Windows application
  - C# ASP.NET web application (.aspx) that is loadable through the browser
  - Browser test form that is supported by **IIS** and **webdev.webserver**.
- Hashtable web service demonstrates application and session state in addition to single-call state for ASP.NET web services.
- Hashtable web service example uses two instances of **xsp4** which is the development web server installed as part of Mono. One to serve the web service and one to serve the web application.
  - download: **hashtableWS.jar**
5.e Adding State to Web Services

5.e.1 Web Services are Stateless

- Between invocations by clients, Web Services loose values of properties.
  - Objects of the class are re-created (and the code is reloaded) for each call, depending upon configuration of the server and ASP.NET.
  - Provides better scalability in high-capacity load-balanced applications - connections can be pooled and/or server objects can be pooled.

- By virtue of extending the System.Web.Services.WebService class, a web service has access to Application and Session objects
  - Application Object can be used to retain values that should persist calls by different clients.
  - Session Object can be used to retain values that should persist multiple calls by the same client (cookies or shopping cart applications).
5.e.2 Application Object

• Application specific storage that persists calls by different clients.
  - WebService class defines an Application property that returns an instance of System.Web.HttpApplicationState
  - The application object is shared across all requests to the Web service: Common to all methods of the service and all clients who call a method on the service.

• HttpApplicationState is a key, value collection, like a Hashtable.
  - Application[“anyName”]=Application[“anyName”] + newValue;

public void addHashEntry(string key, string value) {
    incrCallCount();
    /// overwrite if there is already an entry for key.
    Application[key] = value;
    /// save all of the keys that have been entered by all clients
    if(Application["keys"] == null) {
        Application["keys"] = "I have added the following keys: " + key;
    } else {
        Application["keys"] = Application["keys"] + " +key;
    }
}
5.e.3 Session Object

- Storage that persists calls by a single client, and is unique to the client.
  - WebService class defines a Session property that returns an instance of System.Web.HttpSessionState
  - The **session object** is shared across all requests a single client makes to the Web service: Common to all methods and unique for each client.

- HttpSessionState is a key, value collection, like a Hashtable.
  - Session[“newCartValue”] = Session[“oldCartValue”]+newValue
  - To use the session object, the **EnableSession** property of WebMethodAttribute must be set to **true**, default is false.

```csharp
[WebMethod(EnableSession=true)]
public void rememberClientKeys(string key){
    if(Session[“keys”]==null){
        Session[“keys”] = “keys entered by this client are: “+key;
    }else{
        Session[“keys”]=Session[“keys”]+” “+key;
    }
}
```

- Example of a state-full WebService:
  - Hashtable web service, save key, value pairs by application and session.
  - See: hashtableWS.jar
5.e.4 Sessions in Clients

• When using the browser to exercise a session enabled web service, the session state is properly handled
  - Depending on browser configuration, you can configure sessions to use cookies or cookie-less (passed via url).

• More commonly, clients are separate applications.
  - Enabling sessions in clients is treated in the next section.
5.f Web Service Clients: Cmd Line and Win Apps

5.f.1 Creating a Proxy Class

- Conceptually, proxy classes work for WebServices the same as they do for .NET Remoting.
  - The proxy takes the form of a source file that can be compiled with the client and used as a part of the client application.
  - The proxy is local to the client and accesses the Web Service transparently

- To manually create the proxy class, use the wsdl command line tool.
  
  - Command would generate a C# source file named SailboatProxy.cs

- Construct the client using the Web Service class(es) as though they were local objects. Compiling the client:
  
  msc SailboatCalcsClient.cs SailboatProxy.cs
  - Here is a simple example client for the SailboatCalcs Web Service
    
    See: sailboatCalcsWS.jar
5.2 Enabling Session in Web Service Clients

• For windows applications that are web service clients the following must be done to enable sessions.

  - The client must support a client sessionId this is done in .NET through a CookieContainer.

  - In the constructor of the Windows client application, create a new cookie container via:

    - cookie = new System.Net.CookieContainer();

  - Note that cookie could be defined as a private instance field (property)

• Each time your client constructs a web service proxy object, you must associate the cookie container (and sessionId) with the proxy.

  - HashtableWebService hws = new HashtableWebService();
    hws.CookieContainer = cookie;

• Since a new sessionId is created each new CookieContainer instance, its important to reuse the same HashtableWebService object for all calls that the client makes where the session should persist.
5.3.3  Asynchronous Calls to a Web Service

- Proxy generated by \texttt{wsdl.exe} provides asynchronous calls to Web Methods
  - Begin\texttt{<Web Service Method Name>}
  - End\texttt{<Web Service Method Name>}

- Asynchronous calls allow the client to continue with other activities while the Web Service provides return values. The Web Service calls the call-back method when it completes generating the result.

- For example,

  \begin{verbatim}
  var SailboatCalcs sc = new SailboatCalcs();
  AsyncCallback callBack = new AsyncCallback(SailboatCalcsClient.DOverLCallback);
  cs.BeginCalculateDOverL(5250, 21.9);
  // do something else
  Console.ReadLine();

  ***
	static void DOverLCallback(IAsyncResult callback) {
	SailboatCalcs cs = (SailboatCalcs)callback.AsyncState;
	Console.WriteLine("A sailboat with displ 5250 and lwl 21.9 has d/l of:{0,7:f2}",	
cs.EndCalculateDOverL(callback));
  }
  \end{verbatim}
5.g Web Service Clients: Web Applications

5.g.1 Approaches

- What are the different ways Web Service clients can be realized?

  - **Windows Application** - a resident program that may use Forms controls in interacting with the Web Service on behalf of the user.

  - **Web Application** - An application rendered in the browser with underlying logic executed on the server. The .NET extension of Active Server Pages (ASP) is what we look at in this section.

  - **Console Application** or about any other approach where XML/SOAP are available.

- The previous section looked at manually realizing a Web Service Client as a command line application. What about a Web Client?

  - **Visual Studio** and .NET provide mechanisms to design and realize a Web Application (that may consume Web Services) that presents a User Interface through a Browser.

- Each approach to consuming Web Services has strengths and weaknesses. Performance, distribution, installation, and updating applications are considerations when evaluating different approaches.
5.g.2 Web Applications with Visual Studio .NET

- Using Visual Studio .NET, you create .aspx and .cs codebehind files for a web Application. These are registered with IIS and ASP.NET

- The sequence diagram below shows a Web App using a Web Service.

- The Web method soap request originates from the Web Application (6) (running in ASP.NET) and is executed by the Web Service (7). The results are returned to the Web Application (8 and 9).
5.g.3  Notes on Constructing a Web Application

• Composite user interface web controls, such as **DropDownList** and **ListBox** objects have underlying data models (for loading from a database, for example) through the **DataView** and **DataTable** classes.

• The **aspx code behind** class object is conceptually constructed each time one of its methods is needed.
  - Persistent information must be realized through **Session** and **Application** state objects, as is the case with **Web Services**.

• **AutoPostBack** (and **Button** submits) can be used to notify the codebehind when user events occur, such as selecting from a drop down list.
  - Each time one of these events occurs, it includes a **Page_Load**, meaning that the **Page_Load** method as well as any selection or button handling methods are **all** called.

• While some may question how well the IDE does for developing all code, it is useful to **develop** and **debug** aspx files (**Web Applications**).
  - Browsers are poor debugging platforms; often they have no way to give you an indication of underlying code failures.
  - Visual Studio and ASP.NET work together to avoid reloading libraries.
5.g.4 Example Web Application and Web Service

• Here is an example of a Hashtable Web Service together with clients which are a Windows Form C# client, a Java Swing client, and a C# ASP.NET Web Application client.
  - Hashtable Web Service, supports typical key, value pair associations.
  - For the code, see: hashtableWS.jar
  - The example includes an Ant build file that compiles and executes all components of the example.

• The example Web Application that acts as a client for the HashtableWS Web Service.
  - The Web Client presents a DropDownList, TextBoxes and a Button so the user can select a Web Service operation to call. Provide an input value and observe the result.

• Read the code in C# clients. Note the methods:
  - OnInit overridden from parent class: System.Web.UI.Page
  - InitializeComponent, Page_Load, and buttonClick
5.h Web Service Security

5.h.1 WS-Security

- Invoking web methods over https may not always be the best solution to adding security to web method calls.
  - **https** (http on top of **SSL**) is a transport level approach.
  - Once the message reaches the server, its converted to clear text. If the message is passed to other servers, it may not be encrypted.
  - An **XML** based approach allows the user to use security capabilities that match the needs of the application and message.

- **WS-Security** is an OASIS standard that allows integrity, authentication and confidentiality in SOAP messages
  - **User-Name Token Authentication.** Supports passing user name, password tokens in the SOAP message for the purpose of authentication.
  - **XML-Encryption.** Supports passing the data of a SOAP message in encrypted form.
  - **XML-Signature.** Supports digitally signing a SOAP message and passing the signature as part of the message.
5.h.2 Digital Signature

- **Used for** [authentication](#) and [integrity](#) assurance of digital data.
  - “Verify who it came from and that its not changed in transit”
  - **Integrity** means the data has not been modified or tampered with
  - **Authenticity** means the data comes from whoever claims to have created and signed it.

- **Sender** transfers both [Message](#) and [Signature](#).

- **Receiver** Verifies that the received [Message](#) matches [Signature](#) using [Public Key](#).

- **Asymmetric Keys** are created in matching pairs
  - **Private key** - used to generate signature (or decrypt cipher) and kept confidential to whoever is doing the signing.
  - **Public key** - used to verify authenticity of the message (or encrypt clear text to create a cipher). Distribute the public key to anyone who will receive signed information, or encrypt messages to be read by private key.
5.h.3 Digital Signatures

- Reciver wants assurance of the message sender’s identity (authentication) and that the message has not been altered in transit (integrity).

- **Signing**
  - Generate a digest, noted as $H(m)$, of the message to be sent: $m \Rightarrow H(m)$
  - Generate a signature (encryption of the digest) using the Private Key and an encryption algorithm: $s=\text{Encryption}_{PvK}(H(m))$
  - Send the message $m$ and the signature $s$ to the receiver

- **Verifying**
  - The receiver already has the matching public key (certificate); receives the message $m$ and digital signature $s$.
  - Decrypt the signature using the public key and matching algorithm to get a digest: $H(m')=\text{Decryption}_{PbK}(s)$
  - Generate a digest of the message received: $m \Rightarrow H(m)$
  - If $H(m) = H(m')$ then the message if verified unchanged and from the private key owner; otherwise unverified.

- WS-Security supports placing signature information into the SOAP Header.
5.h.4 XML Encryption

- An Encryption algorithm is used to produce **cypher text** of a portion of the XML document. This encrypted data is placed within an EncryptedData element in the resulting SOAP body.

- In addition to the cypher data, the encryption method, key information and reference data are also passed.

- Encryption can be either via shared key or asymmetric key
  - **Asymmetric key** is performance constrained, but key management is simpler.
  - **Shared key** has the added burden of managing the key.

- With Asymmetric keys:
  - **Private Key** and **Public Key** pairs are generated. The public key can be distributed freely, but the private key is kept secret.
  - The message is encrypted by the sender using the receiver’s public key.
  - Decryption can only be performed using the receiver’s private key.