SoftServe

Introduction to Web Services

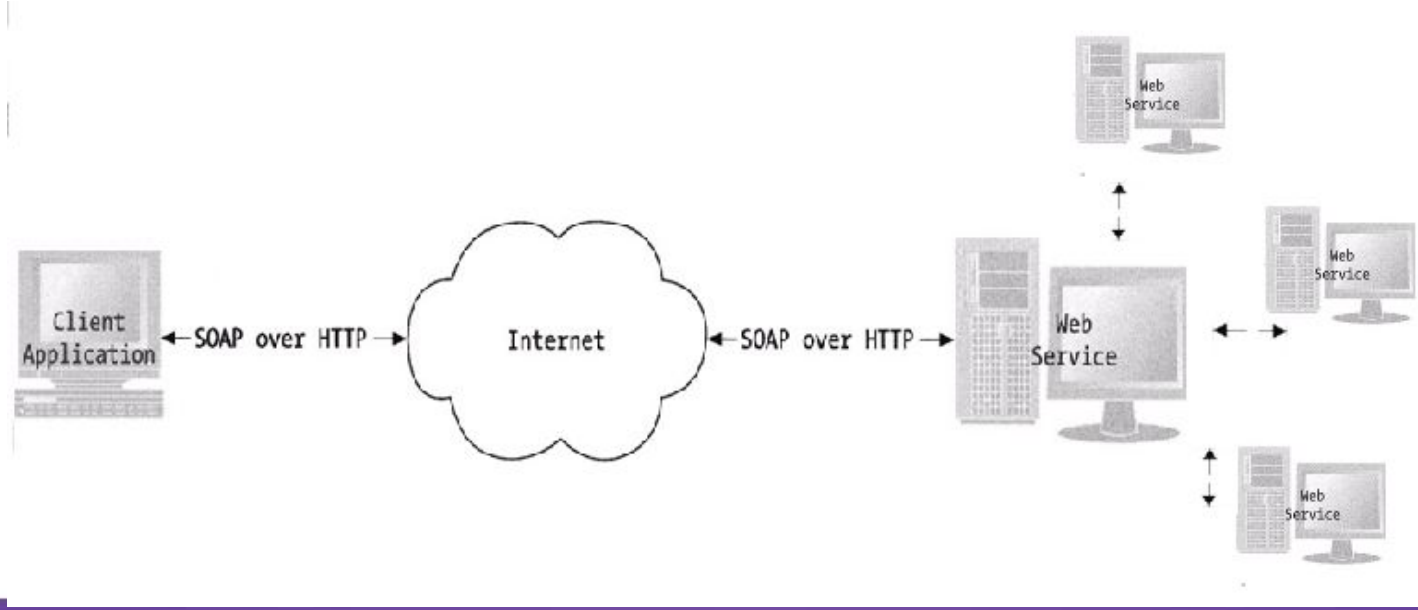
Last update: September 2014
Ihor Kohut
Reviewed by Oleksandr Mykhailyshyn

Agenda

- What is Web Service ?
- Web Services Architecture
- Standards
- Advantages
- Web API
- Styles of use
- Design methodologies
- Case studies

What is Web Service ?

- **Web service** - a **software system**, is identified by string URI, whose public interfaces are defined in XML.
- **Web services** - **software components** that can be accessed and executed remotely via a network by a client application using standard protocols such as Hypertext Transfer Protocol (HTTP) and Simple Object Access Protocol (SOAP)
- **Web service -** a **method of communication** between two electronic devices.



What is Web Service ?

SOA - Service Oriented Architecture of Web applications

- **Main task**: supporting interoperable **machine-to-machine interaction** over a network
- **Web Services** don't have a GUI but have **programming interface**
- **Web Services** are not user-oriented but **application-oriented**

Main Web Services protocols

- **SOAP** (Simple Object Access Protocol)
  - is based on XML, for messages exchanging between services
  - SOAP/WSDL/UDDI
- **REST** (Representational State Transfer)
  - for interacting with resources
- **XML-RPC** (XML Remote Procedure Call)
  - The early version of SOAP

RPC Remote Procedure Call

- Other approaches with nearly the same functionality as RPC are:
  - Object Management Group's (OMG)
  - Common Object Request Broker Architecture (CORBA)
  - Microsoft's Distributed Component Object Model (DCOM)
  - Sun Microsystems's Java/Remote Method Invocation (RMI).

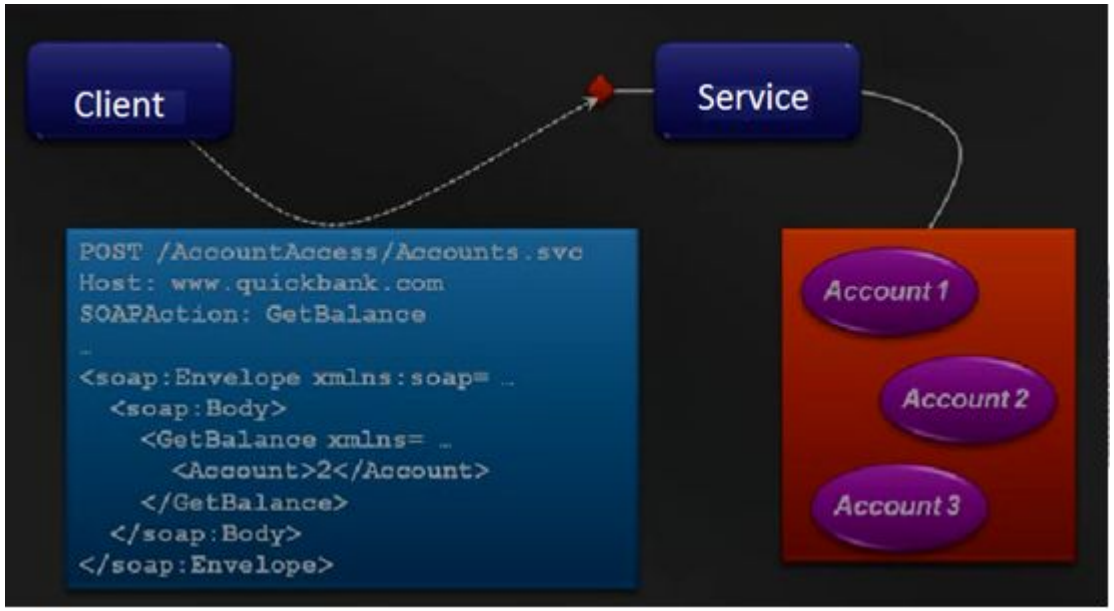
Main Web Services protocols

- **SOAP**–services
  - are focused on actions
  - WCF, ASMX-webservices

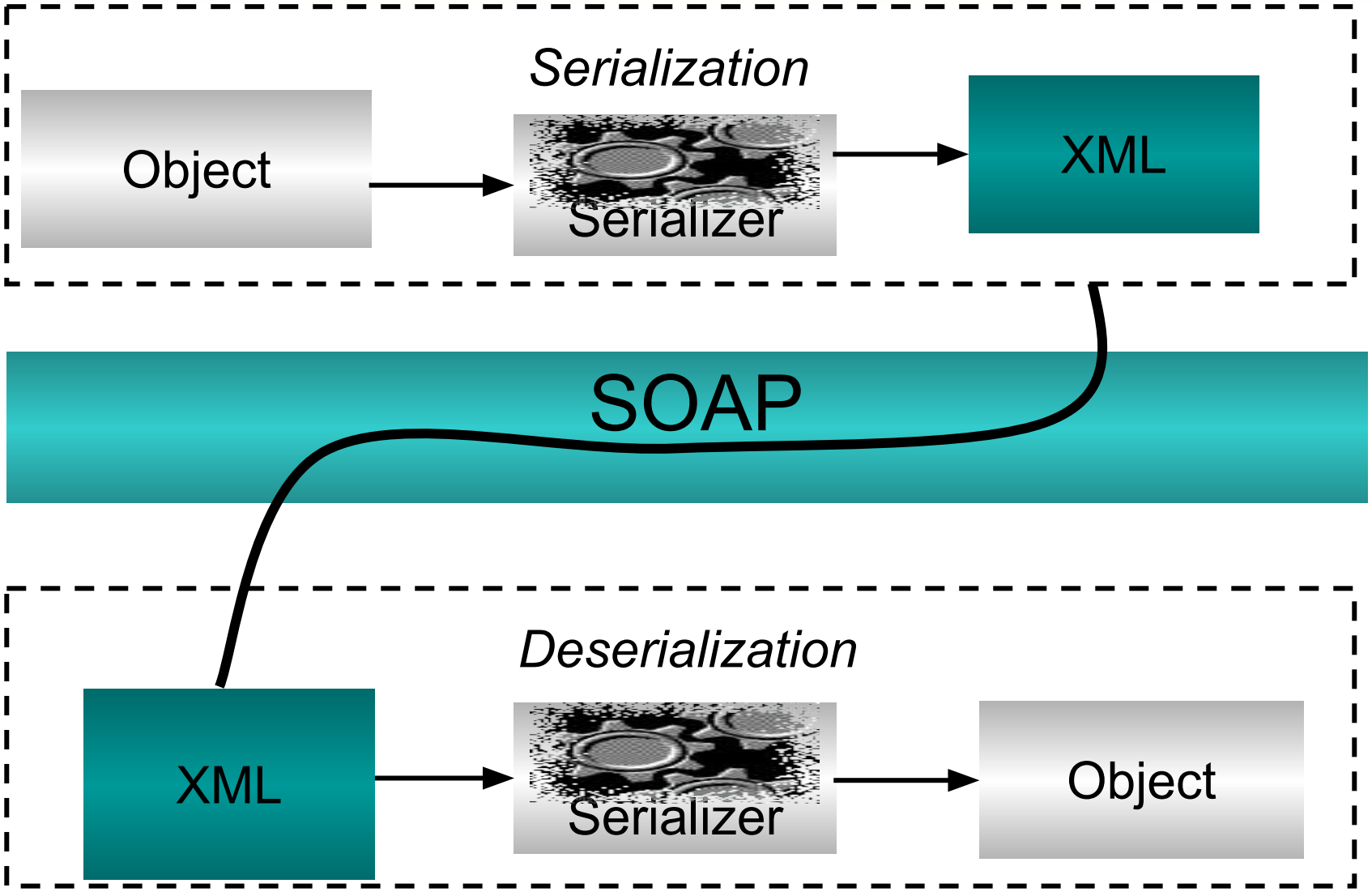
- **REST** -services
  - are focused on data
  - WCF REST, ADO.NET Services

SOAP - services

- SOAP-services **publish the "contract"** (set of methods, parameters and return values descriptions) in WSDL
- Clients know about contract – call methods (using XML), which **are executed on service**
- Interaction with service - throw **endpoints**:
  - **URL-address** of web service
  - **binding** – interaction protocol, security parameters



How does a WebService work?



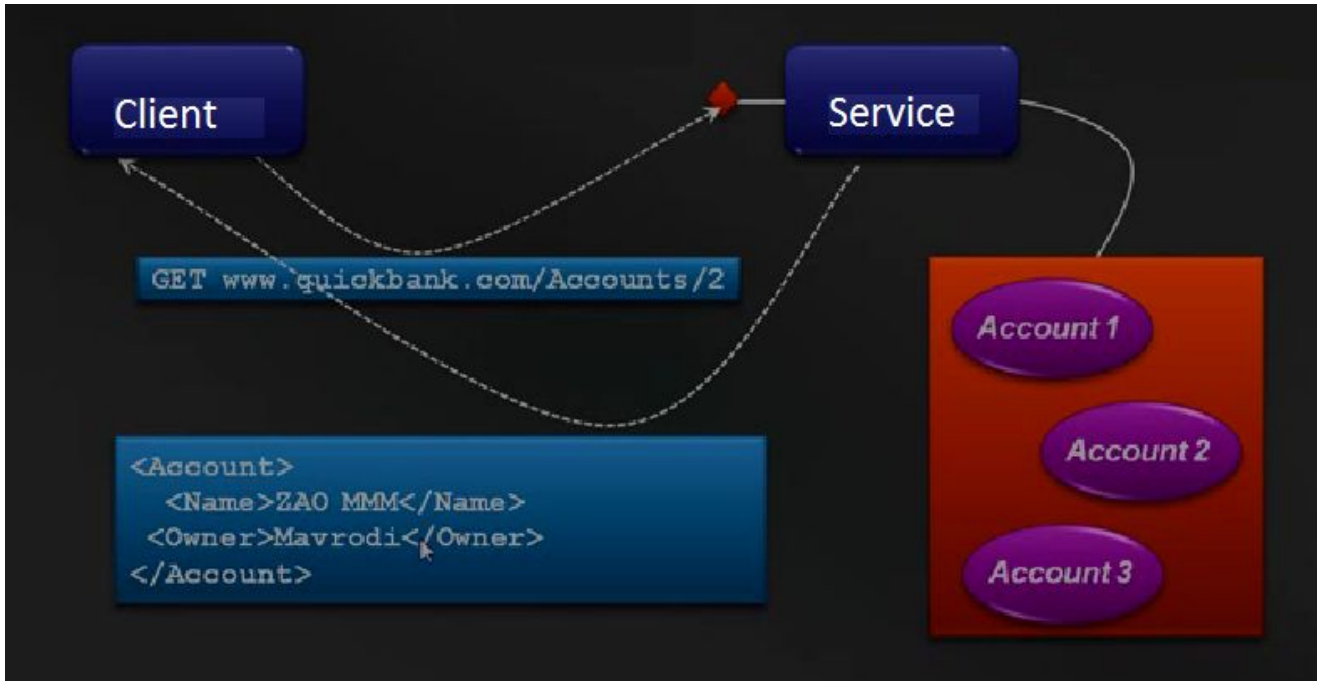
SOAP request and response

```
<SOAP-ENV:Envelope><br>
xmlns:SOAP-ENV="http://[soaporg]/envelope"
SOAP-ENV:encodingStyle="http://[soapporg]/encoding//"
<SOAP-ENV:BODY>
<m:GetStockRespense xmlns:m="SOME-URL">
<Symbol>HST</Symbol>
<m:GetLastStock>
<SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

```
<SOAP-ENV:Envelope><br>
xmlns:SOAP-ENV="http://[soaporg]/envelope"
SOAP-ENV:encodingStyle="http://[soapporg]/encoding//"
<SOAP-ENV:BODY>
<m:GetStockRespense xmlns:m="SOME-URL">
<price>48.6</price>
<m:GetLastStockResprnse>
<Soap-ENV:Body>
</SOAP-ENV:Envelope>
```

REST - services

- REST-services **publish the data source**
- Client sends the request (GET, PUT, POST, DELETE) – not XML
- Service returns the part of data
- Each unit is uniquely **determined** by the URL



REST

- **REST** (Representational state transfer) - very **simple interface** without any additional internal layers.
- **Each unit of information** is uniquely determined by a global identifier such as a **URL**:
  - URL is actually a primary key for the data unit.
  - For example: the third book from the bookshelf will look: `/book/3`
    35 pages in this book : `/book/3/page/35`
  - returns strictly specified format.
  - it doesn't matter what format the data resides at `/book/3/page/35`- HTML file or jpeg-file, MW document
- **Just give the data.** Don't wrap the data in XML.
- **Interaction is based** on the communication protocol - **HTTP**.
  - The actions of **CRUD** (Create-Read-Update-Delete)
  - **GET**, **PUT** (add, replace), **POST** (add, change, delete), **DELETE** (to delete).
  - For Example:

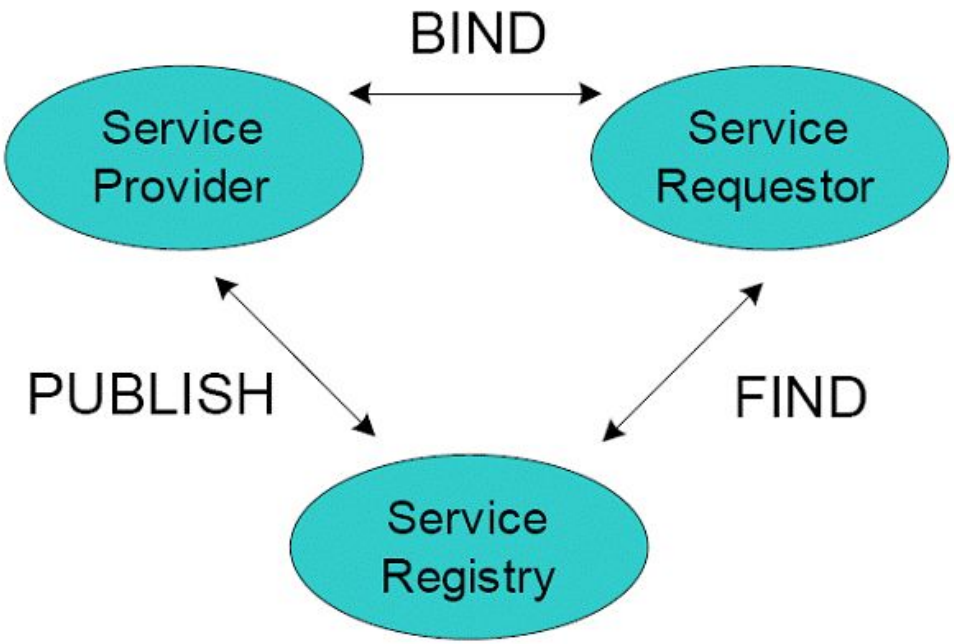
GET `/book/3/` - to get a book number 3
PUT `/book/` - add a book (the data in the request body)
POST `/book/3` - change the book (the data in the request body)
DELETE `/book/3` - remove a book

Windows Communication Foundation

- The **main technology** for building Web Services .NET (Framework 3.5)
- **Is based on layers:**
  - Standard layers: code, codding (message), transport, …

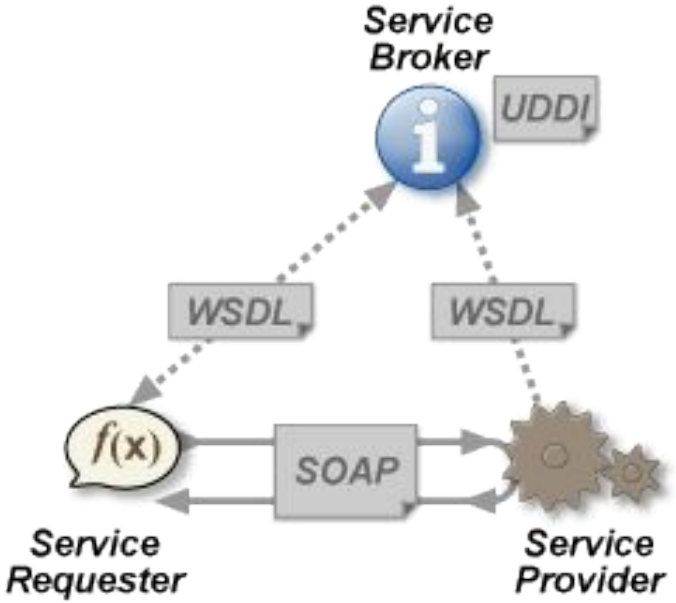
Web Services Architecture

- The architecture allows multiple web services to be combined to create new functionality.



Web Services Architecture

- The web services architecture has three distinct roles:
  - **Provider** creates the web service and makes it available to clients who want to use it
  - **Requestor** is a client application that consumes the web service. The requested web service can also be a client of other web services.
  - **Broker**, such as a service registry, provides a way for the provider and the requestor of a web service to interact.



Standards

- **Web Services based on 3 main standarts:**
- **SOAP -** messaging protocol based on XML;
- **WSDL** (Web Service Definition Language) - The language describing web services interface based on XML;

- **UDDI** (Universal Discovery, Description and Integration)
  - universal interface identification, description and integration.
  - Catalogue of Web services, and information about companies providing Web services into general use or specific companies.
  - is similar to a telephone directory: Business Entity, Business Service, Binding Template and Technology Model (“white, yellow and green” pages)

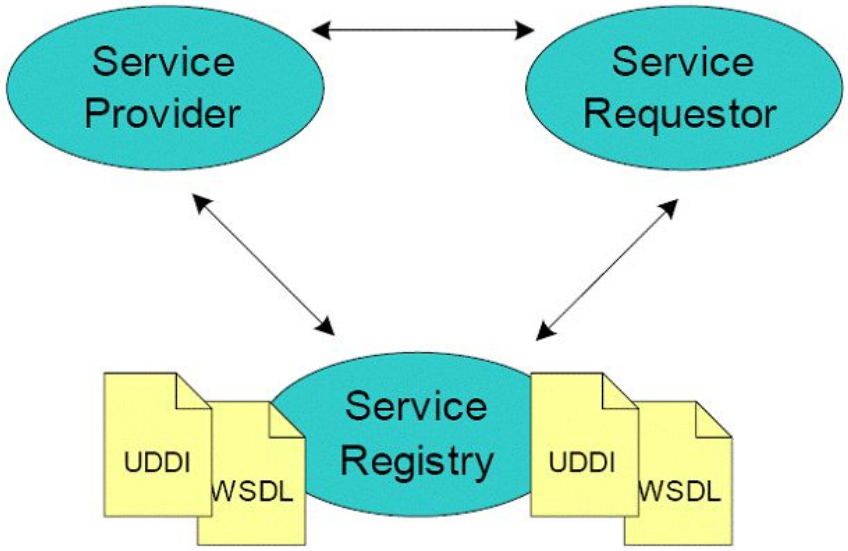
Web service developing -> WSDL document creating ->web service publishing in UDDI registry -> searching and using by clients

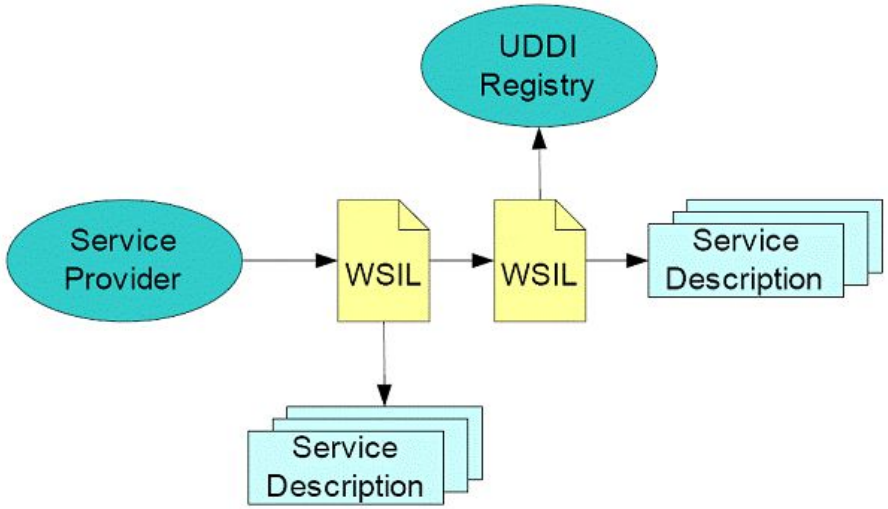
Example: MSDI description of web service

```
<?xml version="1.0" ?>
<definitions name="Stocks" tsrgetNamespace=url
xmlns:soap="http://(soaporg)/wsdl/soap"
xmlns="http://(soaporg)/wsdl/">
<types>
<elements>...</elements>
</types>
<message>...</message>
<portType>...</portType>
<binding>
<operation>
<input>...</input>
<output>...</output>
</operation>
</binding>
<service>...</service>
</definitions>
```

WS Inspection

- **WS Inspection** is based on WSIL (Web Services Inspection Language), like UDDI, provides a method of service discovery for web services.
  - Unlike UDDI, WSIL uses a **de-centralized**, distributed model, rather than a centralized model.
  - The WSIL specification provides standards for using XML-formatted documents to inspect a site for services and a set of rules for how the information is mading available.
  - The WSIL document is then hosted by the provider of the service, so consumers can find out about available services.





WSIL document

```
<?xml version="1.0"?>
<inspection xmlns="http://schemas.xmlsoap.org/ws/2001/10/inspection/">
<service>
<name>MeteoService</name>
<description referencedNamespace="http://schemas.xmlsoap.org/wsdl/"
    location="http://www.meteo.com/wsdl/MeteoService.wsdl" />
</service>
</inspection>
```

Advantages and disadvantages

- **Advantages** of Web Services
  - Web services provide cooperation between software systems regardless of platform;
  - Web services are based on open standards and protocols. Using XML provides ease development and debugging Web services;
  - Using the Internet Protocol provides HTTP-interaction software systems through a firewall.
- **Disadvantages** of Web services:
  - Lower performance and larger network traffic compared with the technologies RMI, CORBA, DCOM through the use of text XML-based messages. However, some Web servers can configure the compression of network traffic.

Design methodologies

- **Automated tools** can help in the creation of a web service:
  - **bottom up** method: developer writes implementing classes first (in some programming language), and then uses a WSDL generating tool to expose methods from these classes as a web service.
  - **top down** method: developer writes the WSDL document first and then uses a code generating tool to produce the class skeleton, to be completed as necessary. This way is generally considered more difficult but can produce cleaner designs.

Example

- Web Services **for the developer**:
  - File web-service has an **extension** of **asmx**;
  - The creation of web-service is not much different from creating a **web form** in. NET Framework;
  - File the Web service must begin with a **directive WebService**;
  - The web-service class may (but dosen't need) be inherited from `System.Web.Services.Webservice`.
  - A **method** that is called through the web, must have the **attribute WebMethod**.

Example

- Create a new application in VS.NET and **add** to it File the web service.
- nw.asmx file contains the line - a **directive WebService**, which states that this file - is really a web service.

```
<%@ WebService Language="c#" Class="WebServicesExample.nw" %>
```

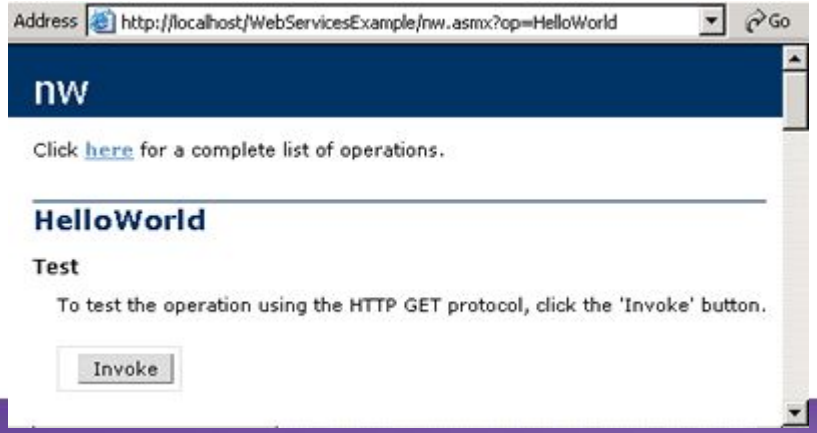
- **Code** for the Web service will be located in the codebehind file nw.asmx.cs.

```
using System;
. . .
using System.Web.Services;

[WebService(Namespace="http://www.aspnetmania.com/webservices")]
namespace WebServicesExample
{
    public class nw : System.Web.Services.WebService
    {
        public nw()
        {
            . . .
        }

        [WebMethod]
        public string HelloWorld()
        {
            return "Hello World";
        }
    }
}
```

Example

- On the Web service page:
  - the name of a Web service (marked 1),
  - a reference to the description of the service (2) (this link will continue to interest us in creating a Web service client)
  - and a list of Web methods declared in a Web service (3).

- Just click on the link on the description page SayHello web-service



WebMethod attribute

- **WebMethod attribute** has six **properties** that affect the web-method:
  - **Description**. This property(string) is for general description of the web-method. Description property value is displayed on the page describing web-service.

```
[WebMethod(Description = "Returns a list of orders for a
specific client")]
public DataSet GetCustOrders(string CustomerID) {...}
```

  - **EnableSession**. This feature allows you to enable sessions. To enable it, specify the web-method as follow:

```
[WebMethod(EnableSession=true)]
public DataSet GetCustOrders(string CustomerID) {...}
```

  - **MessageName**. This property allows you to assign web-method name that is different from a class web-service method.

```
[WebMethod(Description = "Returns a list of client")]
public DataSet GetCustOrders(string CustomerID) {...}

[WebMethod(MessageName = "GetCustOrdersByDate")]
public DataSet GetCustOrders(string CustomerID, DateTime startDate) {...}
```

WebMethod attribute

- **TransactionOption**. Web-service **limits** support transactions. With this property we can control how our method uses the transaction. It may take the following values:
  - **Disabled**. Web method is executed outside the transaction;
  - **Supported**. If a transaction exists - the method is executed in the context of this transaction, but if not - performance goes beyond the transaction;
  - **Required**. Method requires a transaction to be executed. It always creates a new transaction (similar RequiresNew);
  - **RequiresNew**. Method requires the creation of a new transaction. Each time you call the method it is creating a new transaction.

- **CacheDuration**. Caching of web services with an indication of time period in seconds, at which cached web service.

```
[WebMethod(CacheDuration=600)]
public DataSet GetCustOrders(string CustomerID) {...}
```

WebMethod attribute

- **BufferResponse**. BufferResponse property allows you to manage web-buffered response method. Default output is buffered and **sent** to the client only after it is **fully formed**. However, if your web-method is very long runs, perhaps it makes sense to disable the buffering effect.

```
[WebMethod(BufferResponse=false)]
public DataSet GetCustOrders(string CustomerID) {...}
```

Questions?