Cursos de Extensión Universitaria
UNIVERSIDAD DE OVIEDO

PROGRAMACIÓN ORIENTADA A OBJETOS CON C# EN LA PLATAFORMA .NET

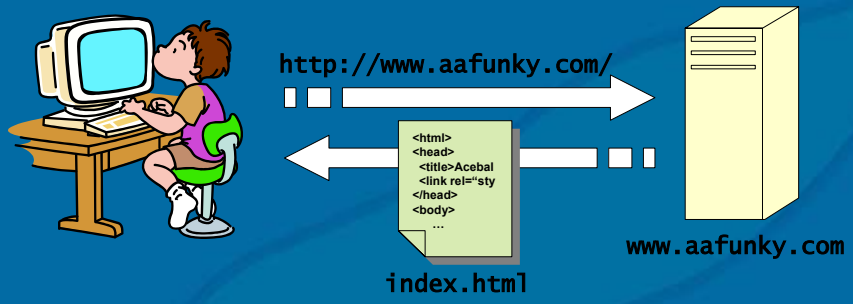
Servicios Web

César Fernández Acebal
acebal@ieee.org

Dpto. de Informática
OOTLab - Laboratorio de Tecnologías Orientadas a Objetos
www.ootlab.uniovi.es

Antecedentes del Web

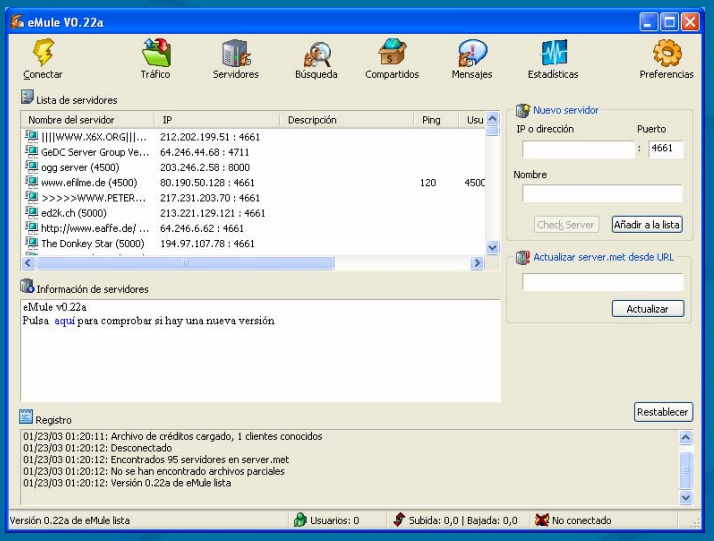
- Desde su creación, el Web está pensado para proporcionar información a los humanos
- Los servidores devuelven páginas HTML que son mostradas por los navegadores, pero que no pueden ser procesadas por otros programas



Antecedentes del Web

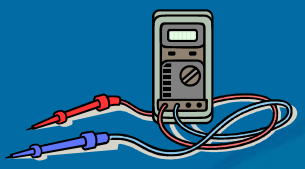
- ¿Por qué no aprovechar la infraestructura de Internet que otros programas puedan solicitar y procesar información?
  - Una interfaz de usuario nativa suele ser mucho mejor que una interfaz Web
    - Piénsese en el Outlook y en Hotmail
  - Además, mejoraría el rendimiento del servidor
  - Habrá programas que no necesiten interfaz de usuario
    - Y sin embargo puedan acceder a los datos a través de Internet
  - Hay multitud de dispositivos de Internet que no son PC
    - Teléfonos móviles, asistentes personales digitales, etcétera

Ejemplo de interfaz nativa: eMule



Web ≠ Navegador

- En definitiva, que no debemos confundir, como hasta ahora, el Web con el navegador
- A comienzos del siglo XX, cuando llegó la electricidad a los hogares estadounidenses, los motores se solían vender aparte de los electrodomésticos
  - El mismo motor servía para conectar, por ejemplo, la máquina de coser, la batidora y el ventilador
  - Por tanto, no podía estar optimizado para ninguno de ellos
  - Para utilizar el electrodoméstico era necesario configurar previamente el motor



Web ≠ Navegador

- En cuanto los motores se hicieron más pequeños y más baratos, comenzaron a introducirse en los propios electrodomésticos
  - Éstos son fáciles de utilizar porque el motor y su alimentación, conexiones, etc. están optimizados para cada tarea específica, y ocultos para el usuario
- Lo mismo está empezando a producirse en la programación Web
  - El acceso a Internet se construirá muy pronto en cualquier programa que se escriba
    - Ejemplo: *Napster*, *Microsoft Money*…
  - No se utilizará el navegador más que para eso: navegar

Cursos de Extensión Universitaria
UNIVERSIDAD DE OVIEDO

Servicios Web
César F. Acebal - OOTLab

El futuro

- Los desarrolladores podrán crear sus aplicaciones a partir de servicios disponibles en el Web
  - Ejemplo: que el revisor ortográfico del procesador de textos sea el del *Oxford English Dictionary*
    - Actualmente, su sitio Web (www.oed.com) sólo tiene una interfaz genérica, para humanos
    - Seguramente tendrían muchas más suscripciones de proporcionar ese tipo de acceso a los programas

¿El futuro? ¿Y qué pasa con RPC, DCOM, CORBA, RMI…?

- En efecto, siempre ha habido técnicas de programación distribuida
- Las tecnologías mentadas adolecen de varios problemas:
  - Falta de interoperabilidad (RPC, RMI…)
  - Complejidad (CORBA)

Requisitos de la solución

- Se necesita un acceso universal, una caja negra de acceso a Internet
  - Independiente del sistema operativo, lenguaje de programación, etcétera
- Hay que resolver dos cosas:
  - El formato de los datos a intercambiar
  - El transporte de esos datos

Transporte

- El protocolo de transporte más habitual en Internet es el HTTP (*Hypertext Transfer Protocol*)
- ¿Por qué no utilizarlo para la comunicación entre aplicaciones?
- ¡Ojo!, que no tiene por qué ser éste obligatoriamente
  - Los servicios Web podrían hacer uso de cualquier otro protocolo (SMTP, etc.)

Formato de intercambio de datos

- XML
- Lenguaje de etiquetas, de sintaxis muy sencilla
  - Legible tanto por los humanos como por la máquina (fácilmente procesable)
- Adecuado para representar información *semi-estructurada*

Introducción a XML

Aunque no es el propósito del curso, es necesario ofrecer una introducción al lenguaje XML para el alumno desconocedor del mismo.

No obstante, pasaremos de puntillas sobre su sintaxis y nos centraremos en la filosofía subyacente, y en cómo ésta encaja a la perfección con la idea de los servicios Web.

¿Qué pasa con HTML?

- HTML (HyperText Markup Language) es un lenguaje de estructuración de páginas
  - (aunque se haya desvirtuado mezclándolo con la presentación)
  - Sirve para indicar qué es un párrafo, estructurar el documento en varios niveles de títulos, etc.

```
<html>
  <head>
    <title>AA Funky Software</title>
  </head>
  <body>
    <h1>AA Funky Software</h1>
      <div class="menu">
        <a href="who/" title="Who we are">Who</a>
        ...
  </body>
</html>
```

Caso de estudio: página de temperaturas

- Supongamos una página Web de un centro meteorológico que provee información acerca de la temperatura actual en varias ciudades
- Un centro comercial desea conectarse cada hora a dicha página para regular la temperatura del local en función de la temperatura exterior
- ¿Cómo puede extraer la información de la página?

Página Web meteorológica

```
<html>
<head>
  <title>Parte meteorol&oacute;gico</title>
</head>
<body>
  <h1>Parte meteorol&oacute;gico - Oviedo (Asturias)</h1>
  <table border="1">
    <tr>
      <td>Fecha</td>
      <td align="center">25 noviembre 2000</td>
    </tr>
    <tr>
      <td>Temperatura</td>
      <td align="center">16&#176;</td></tr>
    </tr>
    ...
```

Página Web meteorológica (Cont.)

```
    ...
    <tr>
      <td>M&aacute;xima prevista para hoy</td>
      <td align="center">21&#176;</td>
    </tr>
    <tr>
      <td>M&iacute;nima prevista para hoy</td>
      <td align="center">11&#176;</td>
    </tr>
  </table>
</body>
</html>
```

Estrategia 1

- Ir a la línea 14, columna 32 de la página HTML y extraer el número que haya allí hasta llegar al *ampersand* (&)
- Totalmente *inmantenible*: cualquier mínimo cambio en la página HTML invalidaría esta estrategia, produciendo resultados impredecibles

Estrategia 2

- Ir a la primera etiqueta `<table>`, después a la segunda etiqueta `<tr>` dentro de la tabla y, por último, a la segunda etiqueta `<td>` dentro de la fila.
- Parecido a la anterior; resistirá cambios nimios como insertar una línea en blanco, pero, ¿y si el diseñador de la página añade una tabla antes de la actual?

¿Cuál es la raíz del problema?

- HTML fue diseñado originariamente para representar una estructura de presentación de un documento
  - Títulos, párrafos, enlaces...
- No contiene etiquetas para representar datos lógicos, como la temperatura
  - *No aporta información semántica*
- Otro problema adicional es la laxitud de los navegadores con el estándar HTML: tragan con casi todo

Ejemplo de documento XML

parte-meteorologico.xml

```
<?xml version="1.0"?>
<!DOCTYPE parte-meteorologico SYSTEM
          "parte-meteorologico.dtd">
<parte-meteorologico>
  <ciudad>Oviedo</ciudad>
  <fecha>25 de noviembre de 2000</fecha>
  <temperatura-actual unit="Celsius">16</temperatura-
actual>
  <maxima unit="Celsius">21</maxima>
  <minima unit="Celsius">11</minima>
</parte-meteorologico>
```

¿Qué aporta XML?

- XML es una representación lógica de los datos meteorológicos, independiente de la presentación de los mismos en la página
- Para ello se define la etiqueta `<temperatura-actual>`

Estrategia 3

- Ir a la etiqueta `<temperatura-actual>`
- Pero, ¿cómo podemos garantizar que dicha etiqueta existe en el documento y que se llama así?
- Porque vendrá definida en la gramática de la aplicación, de una de estas dos formas:
  - DTD (Document Type Definition)
  - Esquema XML
- La gramática deberá ser pública y accesible

Ejemplo de gramática (DTD)

- El DTD para nuestro ejemplo de temperaturas sería el siguiente:

parte-meteorologico.dtd

```
<!ELEMENT parte-meteorologico (ciudad, fecha, temperatura-
actual, maxima, minima)>
<!ELEMENT ciudad (#PCDATA)>
<!ELEMENT fecha (#PCDATA)>
<!ELEMENT temperatura-actual (#PCDATA)>
<!ELEMENT maxima (#PCDATA)>
<!ELEMENT minima (#PCDATA)>
<!ATTLIST temperatura-actual unit (Farenheit|Celsius)
#REQUIRED>
<!ATTLIST maxima unit (Farenheit|Celsius) #REQUIRED>
<!ATTLIST minima unit (Farenheit|Celsius) #REQUIRED>
```


Problemas de los DTD

- No son XML
  - ¿Por qué no aprovechar el propio lenguaje para especificar su gramática?
  - Así, podríamos beneficiarnos de las numerosas herramientas de procesamiento de XML
  - E incluirlos dentro de otro documento XML
- Sólo permiten definir la *estructura* del documento
  - No sus tipos de datos (más allá de unos pocos tipos básicos)
    - Por ejemplo, #PCDATA puede ser cualquier cosa

Esquema XML

- Para solucionar esos problemas, surge el Esquema XML
  - Al igual que el lenguaje, un estándar del W3C (www.w3.org)
    - XML Schema Part 1: Structures
    - XML Schema Part 2: Datatypes
  - Tiene un rico conjunto de tipos de datos, y también permite especificar cosas como claves ajenas o integridad referencial

Ejemplo de esquema

book.xsd

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="book">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="title" type="xs:string"/>
        <xs:element name="author" type="xs:string"/>
        <xs:element name="date" type="xs:date"/>
      </xs:sequence>
      <xs:attribute name="isbn" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Ejemplo de esquema

- Un posible documento XML que satisfaga el esquema anterior sería el siguiente:

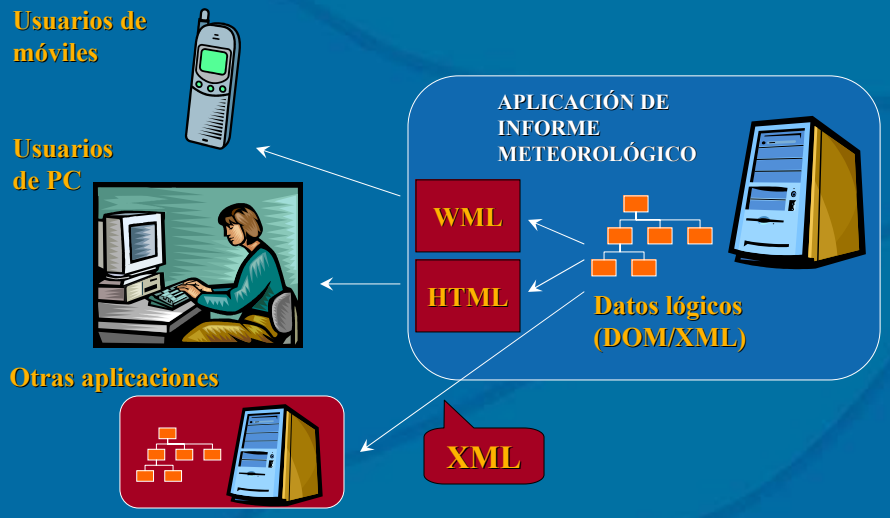
xp.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<book isbn="02016164162">
 <title>
  Extreme Programming Explained: Embrace Change
 </title>
 <author>Kent Beck</author>
 <date>1999-10-5</date>
</book>
```


Aclaración

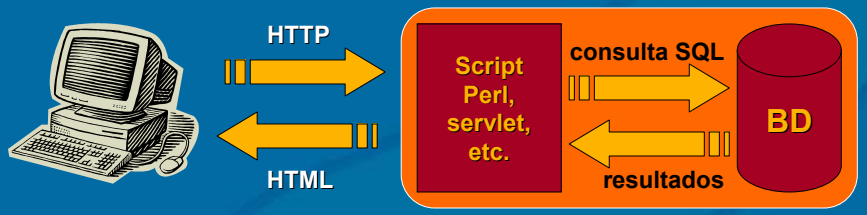
- No siempre es obligatoria una gramática
- Lo es si queremos *validar* el documento
  - No para comprobar que está bien formado
- Habrá aplicaciones para las que la utilidad de XML vendrá dada precisamente por la falta de gramática
  - Fundamentalmente, en entornos complejos (CAS, *Complex Adaptive Systems*), en los que es imposible realizar un análisis previos de la información requerida, pues ésta… ¡muta continuamente!

Escenario de uso de XML



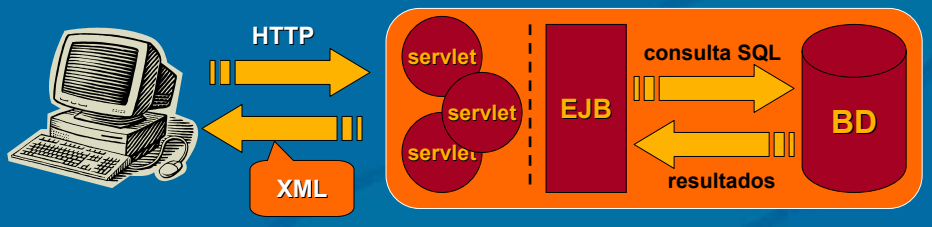
XML y arquitecturas en 3 capas

- Muchas aplicaciones Web actuales acceden a algún programa en el servidor (CGI, *servlets*...) que a su vez ejecuta una petición a una base de datos y devuelve unos resultados formateados en HTML



XML y arquitecturas en 3 capas

- Este enfoque típico en tres capas, sólo vale para la presentación
- Pero... ¿por qué no devolver los datos en formato XML y que el cliente haga con ellos lo que quiera? (p. ej. importarlos en Excel como una hoja de cálculo)



Introducción a la sintaxis de XML

- *"Un objeto de datos es un documento XML si está bien formado. Un documento XML bien formado puede, además, ser válido, si cumple otras restricciones."*

Documento bien formado

- Un documento XML está bien formado si cumple las reglas sintácticas de la especificación
- Ejemplos:
  - Que no haya etiquetas sin cerrar
  - Que no se entremezclen etiquetas: <a>
  - etc.

Documento válido

- Para que un documento XML sea válido debe hacer referencia a un DTD o a uno o varios esquemas:
  - Declaración <!DOCTYPE> al principio, donde viene especificada el DTD
  - O indicando el esquema o esquemas a los que se adhiere, mediante atributos `xmlns`
- El documento debe adecuarse a la estructura impuesta
- Si no, el procesador XML no realizaría la comprobación de validez

Aspectos generales de la sintaxis

- Los documentos XML distinguen entre mayúsculas y minúsculas
- Todos los espacios, tabuladores y saltos de línea fuera de las etiquetas se tienen en cuenta
- Hay algunos caracteres especiales reservados, que forman parte de la sintaxis de XML: <, >, &, " y '. En su lugar deberemos usar las entidades <, >, &, " y ' respectivamente.
- Los valores de los atributos de todas las etiquetas deben ir siempre entrecomillados. Son válidas las dobles comillas (") y la comilla simple (').

Aspectos generales de la sintaxis

- Elementos
  - Pueden tener contenido, en cuyo caso necesitan una etiqueta de inicio y otra de cierre:

```
<alumno>Paco</alumno>
```

  - O pueden ser vacíos (dos posibilidades):

```
<becado></becado>
<becado/>
```

Servicios Web

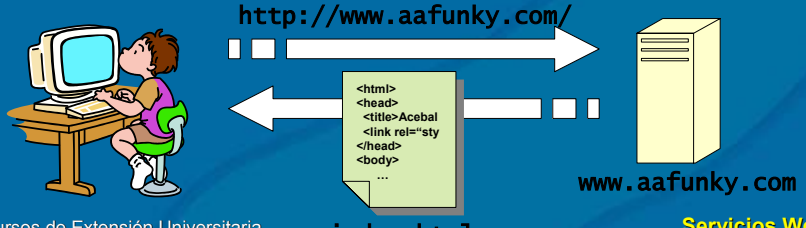
Hemos visto los antecedentes del Web, y cómo es una pena usar toda esa infraestructura únicamente para mostrar páginas HTML.

Hemos echado un vistazo a XML, un formato extensible para el intercambio de información.

Veamos ahora qué demonios es eso de los servicios Web que promete ser el *middleware* del futuro (frente a otros intentos fallidos como RPC, DCOM, CORBA, RMI…).

Introducción

- Un servicio Web no es más que la quintaesencia del Web, tal y como lo conocemos
- Aunque el término es nuevo, no lo es el concepto
- En el fondo, en un sentido lato, la invocación de un URL desde el navegador para obtener una página HTML puede considerarse un *servicio Web*
  - Hay un módulo del servidor que se encarga de recoger dicha petición y de devolver un documento HTML sobre HTTP

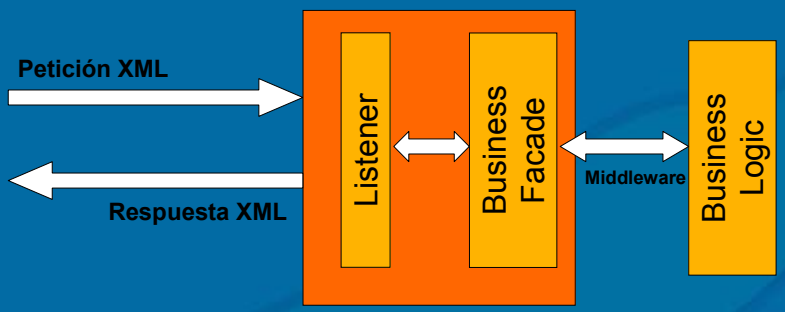


Servicios Web

- Llevando esa idea un poco más lejos, un servicio Web es un programa que puede ser accedido *por otro programa* a través del Web
  - Todo lo que se necesita para localizar y acceder a un servicio Web es un URL
  - En la práctica, el protocolo es siempre HTTP
  - En vez un único punto de acceso para acceder a un URL y obtener HTML, se pueden usar varios métodos con sus correspondientes signaturas para acceder al servicio

Servicio Web: una definición

Un servicio Web es un programa servidor que acepta peticiones entrantes de los clientes e intercambia XML sobre HTTP



Arquitectura de un servicio Web genérico

Estándares abiertos

- No importa con qué se cree un servicio Web, la forma de exponerlo públicamente es la misma
- Pueden ser accedidos por cualquier aplicación que entienda XML y use HTTP
- La escasa infraestructura necesaria la proveen automáticamente los entornos de desarrollo

Claves del éxito

- Fundamentalmente, dos:
  - Simplicidad
  - Ubicuidad

  } **Interoperabilidad**
- En definitiva, son muy "ligeros":

XML + **HTTP**

- Proveen una interfaz de acceso uniforme y ampliamente disponible a los servicios

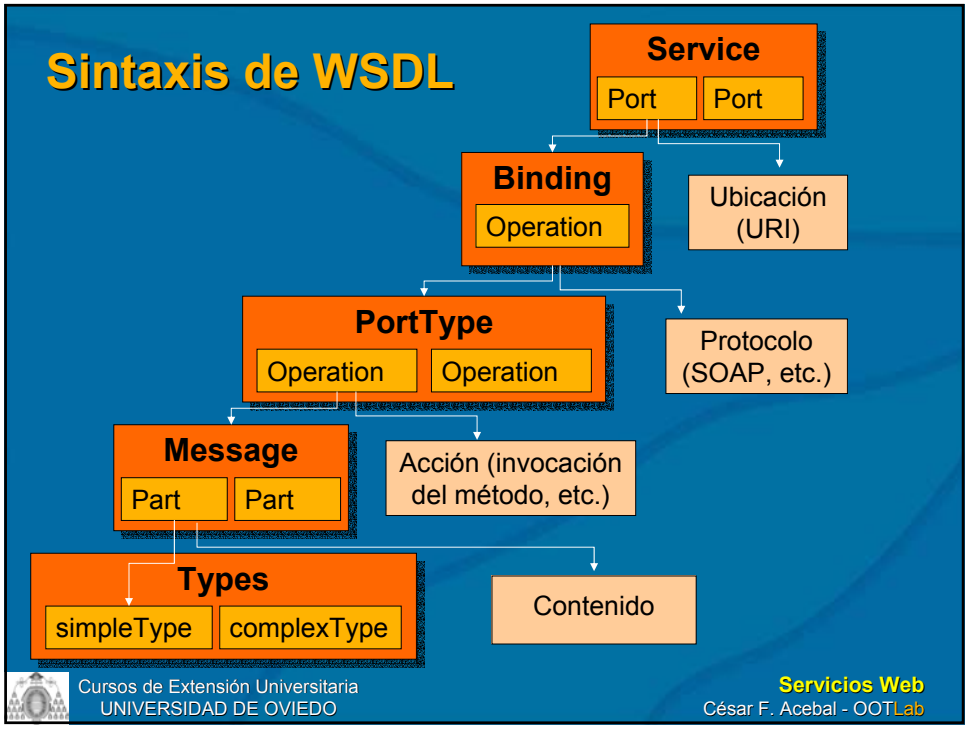
Cómo se describe un servicio Web

- Es decir, si puede tener varios métodos, cómo saben los clientes cuáles son, qué parámetros toman, qué devuelven…
- Además, lo ideal sería que una herramienta pudiera entenderlo y generar en el cliente el código necesario para invocar tales métodos, a través de un *proxy*
- Aparte de los métodos, hay que especificar:
  - La dirección a la que enviar el mensaje XML
  - El protocolo mediante el que se envía (HTTP, SMTP…)

WSDL

- Es el lenguaje de descripción de servicios Web (*Web Service Description Language*)
- Es también un dialecto de XML
- Utiliza el esquema XML

Sintaxis de WSDL



Ejemplo de documento WSDL

```
<?xml version="1.0" encoding="utf-8"?>
<definitions targetNamespace="http://somedomain/Calculator/wsdl"
                    xmlns:tns="http://somedomain/Calculator/wsdl"
                    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                    xmlns:s="http://somedomain/Calculator/schema"
                    xmlns="http://schemas.xmlsoap.org/wsdl/">

  <types>
    <!- Definiciones de los mensajes (nombre, tipos de parámetros y tipo de retorno) -->
  </types>

  <message name="AddMsgIn">
    <part name="parameters" element="s:Add"/>
  </message>
  <message name="AddMsgOut">
    <part name="parameters" element="s:SubtractResult"/>
  </message>
  ...

  <portType name="CalculatorPortType">...</portType>

  <binding name="CalculatorBinding" type="tns:CalculatorPortType>...</binding>

  <service name="CalculatorService">
    <ext:SomeExtElement/>
    <port name="CalculatorPort" binding="tns:CalculatorBinding">
      <ext:SomeExtElement/>
    </port>
  </service>

</definitions>
```

Elemento 'types'

```
<types>
  <schema attributeFormDefault="qualified" elementFormDefault="qualified"
                xmlns="http://www.w3.org/2001/XMLSchema"
                targetNamespace="http://somedomain/Calculator/schema">

    <!- Definiciones de los mensajes -->

    <element name="Add">
      <complexType>
        <all>
          <element name="x" type="int"/>
          <element name="y" type="int"/>
        </all>
      </complexType>
    </element>
    <element name="AddResult">
      <complexType>
        <all>
          <element name="result" type="int"/>
        </all>
      </complexType>
    </element>

  </schema>
</types>
```

Elemento 'types'

- La definición de tipos por omisión de WSDL es el Esquema XML, pero no es obligatorio
- Dentro de `types`, van las definiciones de los métodos
  - Hacen referencia al espacio de nombres definido previamente en `definitions`
- WSDL no está limitado a describir formatos de *serialización* basados en XML
  - Podría usarse para describir un servicio DCOM, por ejemplo (es decir, incluso con formatos binarios)

El “Hola, mundo” remozado :-)

Pues sí, el “Hola, mundo” original de Kernighan y Ritchie con el aspecto mucho más lozano de… ¡un servicio Web!

Mi primer servicio Web en .NET

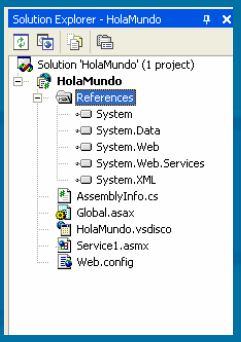
- Para hacer el servicio Web, hay que crear un proyecto de tipo *ASP .NET Web Service*
- Nótese que en ubicación aparecerá:
  - `http://localhost/<nombre del proyecto>`
- Y es que, para poder probar nuestro servicio Web, necesitaremos tener *Internet Information Server* (IIS) instalado en nuestra máquina
  - Así, pues, la ubicación real del servicio Web sería:
    `c:\inetpub\wwwroot\HolaMundo`

Internet Information Server (IIS)

- Es un servidor Web
  - Como lo es Apache
- Un servidor Web es un programa que escucha peticiones HTTP (normalmente, en el puerto 80) y que devuelve una página HTML
- Al instalarlo, permitimos que nuestra máquina sea un servidor Web como cualquiera a los que accedemos en Internet

¿Qué tenemos?

- Visual Studio .NET ha creado automáticamente una serie de ficheros
  - `AssemblyInfo.cs`
  - `Global.asax`
  - `HolaMundo.vsdisco`
  - `Service1.asmx`
  - `Web.config`



- Además, ha añadido las referencias necesarias:
  - `System, System.Data, System.Web, System.Web.Services, System.XML`

Service1.asmx

- Elegimos ver el código del fichero `Service1.asmx`
- Cambiamos el nombre de la clase
  - Por ejemplo, en vez de `Service1`, la llamamos `HolaMundo`
  - Naturalmente, habrá que cambiar también el nombre del constructor para que coincida con el de la clase
- Es buena idea cambiar también el nombre del propio fichero para que coincida con el de la clase que implementa el servicio Web
  - `HolaMundo.asmx`

Definición de métodos del servicio Web. Atributo WebMethod

- Implementamos el siguiente método:

```
[WebMethod]
public string Saludar()
{
    return "¡Hola, Mundo!";
}
```

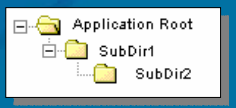
- `[WebMethod]` es un atributo que indica al motor de ejecución de ASP .NET que dicho método pertenece al servicio Web
  - Entre otras cosas, generará el WSDL para él

Fichero Web.config

- ASP .NET proporciona muchos servicios prefabricados
- Cada aplicación individual dispone de muchas opciones de configuración
  - Esta información de configuración se guarda en el fichero `Web.config`
    - Se almacena en el propio directorio de la aplicación
    - Puede haber uno por cada subdirectorio
    - La información de configuración está en XML

Fichero Web.config

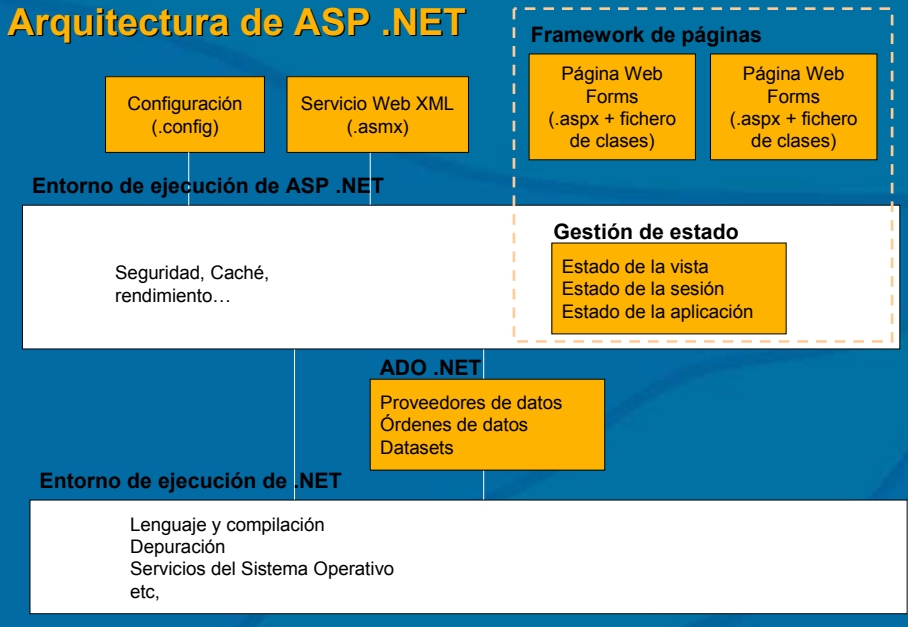
- El fichero maestro que especifica los valores predeterminados de todo el sistema .NET se llama `Machine.config`
  - `%windir%\Microsoft .NET\Framework\version\CONFIG`
- Dicha configuración se puede sobrescribir por los ficheros `Web.config` de las diferentes aplicaciones (y éstos, por los de sus subdirectorios)



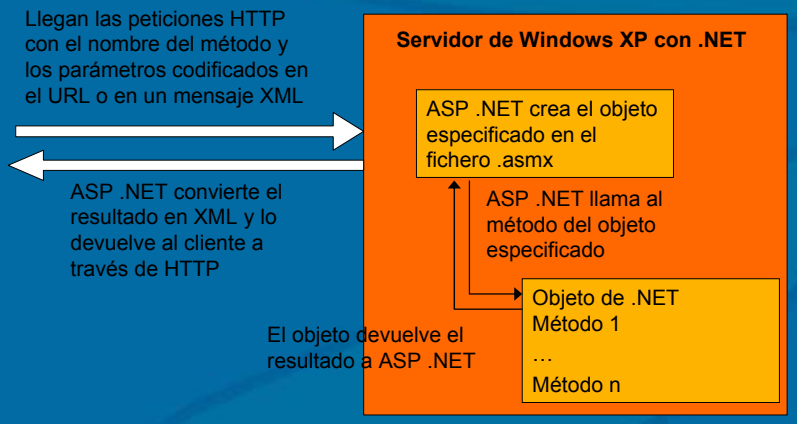
Visión general de ASP .NET

- No confundir ASP .NET con *Active Server Pages*
- Son sólo una parte de una plataforma para el desarrollo y ejecución de aplicaciones en un servidor Web
  - Controles y objetos de tiempo de diseño
  - Entorno en tiempo de ejecución
- Es parte de la plataforma .NET (.NET Framework), por lo que puede acceder a todas las posibilidades de ésta:
  - Lenguajes de programación, depuración, acceso a datos mediante ADO .NET, servicios del SO, etc.

Arquitectura de ASP .NET



Ejecución de un servicio Web desde el punto de vista del servidor



Volviendo al ejemplo…

- Retomemos el Servicio Web `HolaMundo`
- Hacemos `Build`
- Y lo ejecutamos… (`Ctrl + F5`)
- En ese momento ASP .NET detectará que la petición es a un servicio Web e Internet Information Server abrirá la página generada automáticamente para él
  - Nota: debemos tener los permisos adecuados

Algunos servicios Web

- Google
  - www.google.com/apis
  - Proporciona una interfaz para realizar búsquedas en Google
  - Podríais implementar una interfaz completamente nueva

- Amazon

Amazon.com Web Services 2.0

  - http://associates.amazon.com/exec/panama/associates/ntg/browse/-/1067662/086-9135833-2836249
  - Lo mismo: permite buscar libros en su base de datos
- X Methods
  - www.xmethods.net

Conclusiones

Recapitulemos lo visto hasta aquí y anticipemos qué nos falta por ver en la próxima clase.

Qué falta por ver

- Lo más importante de lo visto hasta aquí es comprender la idea que está detrás de los servicios Web.
- Aparte de eso, hemos visto los servicios Web desde el lado del servidor
  - Es decir, desde el punto de vista del propio servicio
  - Concretamente utilizando la plataforma .NET y el entorno de desarrollo Visual Studio .NET.
- Nos falta por ver los servicios Web desde el punto de vista del cliente
  - Esto es, ¿cómo hacemos un cliente de un servicio Web?
  - Y, más concretamente, ¿qué facilidades proporciona Visual Studio .NET para ello?
- También habrá que comentar brevemente una faceta de los servicios Web de la que no hemos hablado:
  - Cómo descubrir servicios Web (UDDI)

¿Y SOAP?

- Quien ya hubiera leído algo sobre el tema –y dado el boom editorial, raro sería lo contrario–, estará a estas alturas preguntándose: ¿qué pasa con SOAP?
- Pues pasa que, si bien es un estándar de facto, los servicios Web no tienen por qué estar ligados a SOAP
  - Por eso siempre hemos hablado de peticiones y respuestas XML
  - SOAP es un protocolo de comunicación en XML
  - Es cierto que es un estándar de facto en lo referente a servicios Web
  - Pero no tiene por qué ser siempre así
    - Existen estilos alternativos, como REST
  - Por eso hemos evitado referirnos a él hasta aquí

¿Y SOAP?

- Si bien SOAP resulta apropiado para servicios Web del estilo RPC (*Remote Procedure Call*), como los que hemos visto, puede no serlo tanto si lo único que queremos es intercambiar precisamente un mensaje XML tal cual
- En cualquier caso, esto lo veremos mejor cuando veamos cómo invocar a un servicio Web
  - Diferentes posibilidades: GET, POST, SOAP…

Conclusiones

- ¿Por qué pueden triunfar los servicios Web?
  - No será por que constituyan una revolución tecnológica, sino por su simplicidad

Servicios Web = La belleza de la simplicidad

  - Lo realmente bueno es la idea
  - Recordemos: aprovechar la infraestructura existente (HTTP, *routers*, *firewalls*, servidores Web, XML…en definitiva, el Web) de un modo novedoso