

Tema 9. Comunicaciones IP

Formación específica, cursos verano 2008
ETS de Informática Aplicada
Universidad Politécnica de Valencia

- ◆ Consumo de Web Services
- ◆ Comunicaciones HTTP
- ◆ Comunicaciones por sockets

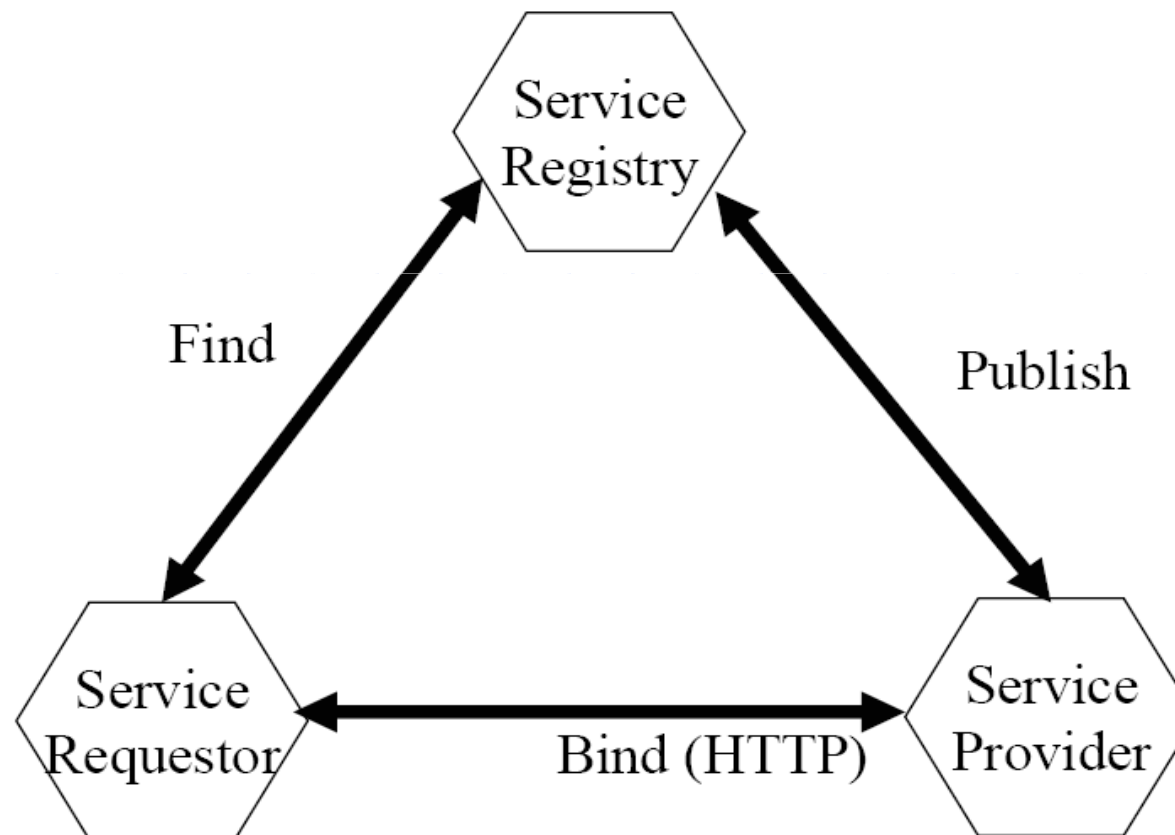
Consumo se Web Services

- ◆ Arquitectura de un Web Service
- ◆ Clientes .NET CF que consuman Web Services

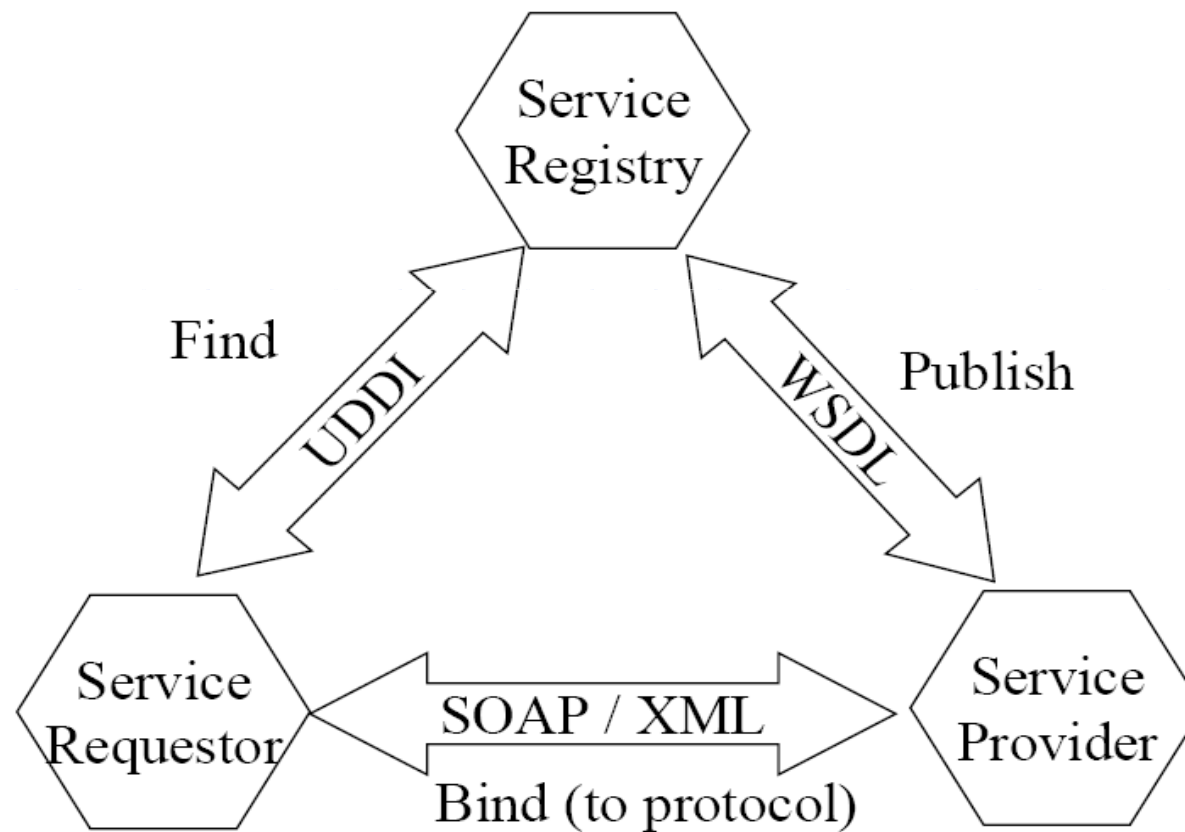
Web Service: Definición

- ◆ Un Web Service es una aplicación Web identificada por un URI y cuya interfaz está definida y puede ser descubierta mediante XML
- ◆ Las operaciones de su interfaz son accesibles de forma remota usando mensajes XML mediante protocolos estándar de Internet
- ◆ Facilitan la integración de tecnologías
 - Interacción entre aplicaciones programadas con distintos lenguajes y funcionando sobre distintos sistemas operativos
 - Uso de soluciones desarrolladas por terceros
 - Desacoplamiento entre clientes y bases de datos

Arquitectura orientada a servicios



Arquitectura orientada a servicios

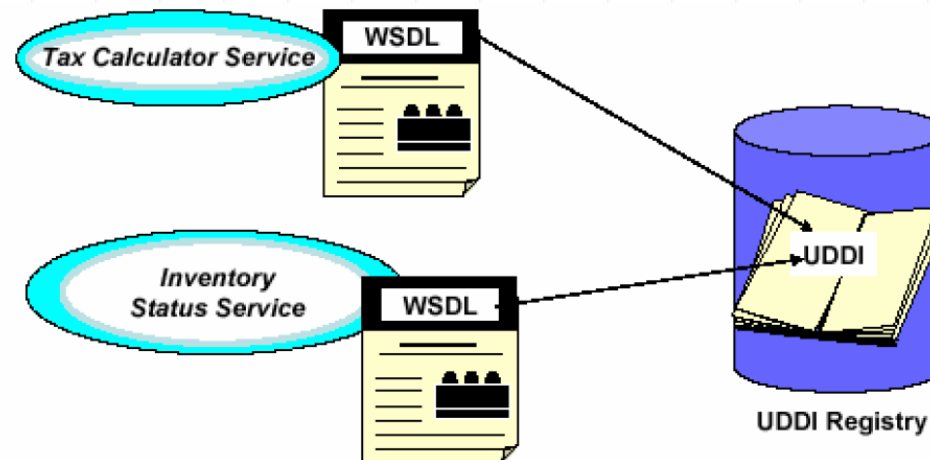


WSDL

- ◆ Web Services Description Language, es un documento XML que representa la definición de las interfaces del Servicio Web
- ◆ Documento XML que describe servicios Web
 - Qué operaciones puede realizar, cuál es su formato y cómo se debe invocar

UDDI

- ◆ Universal Description, Discovery and Integration, define un registro de protocolos para localizar y acceder a los servicios publicados
- ◆ Permite publicar y descubrir información sobre proveedores y los servicios que ofrecen
- ◆ La información contenida puede incluir enlaces hacia la interfaz WSDL, documentos de especificación del servicios, etc.



Tecnologías para la transmisión de las invocaciones

◆ XML

- Describe la información a ser usada

◆ SOAP + HTTP

- Empaqueta la información y la transporta entre el cliente y el servidor

XML

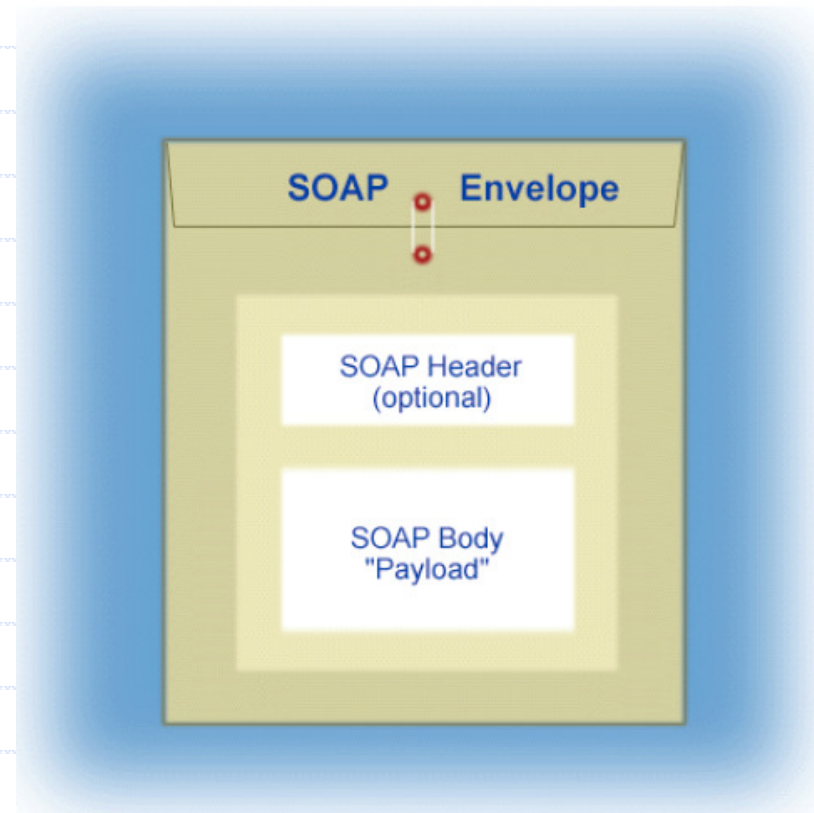
- ◆ XML puede describir tanto datos, metadatos y servicios
- ◆ Proporciona una forma sencilla de estructurar información compleja
- ◆ Flexibilidad, facilidad de procesamiento
- ◆ Independencia de plataformas y arquitecturas

SOAP

- ◆ SOAP es un protocolo basado en XML para el intercambio de información de forma descentralizada sobre entornos distribuidos
- ◆ Define un mecanismo para el paso de instrucciones (comandos) y parámetros entre clientes y servidores.
- ◆ Es totalmente independiente de la plataforma, el modelo de datos y el lenguaje de programación usado.

Estructura de SOAP

- ◆ Requiere un cuerpo (body) soap y un sobre (envelop)
- ◆ Cabeceras (headers) opcionales



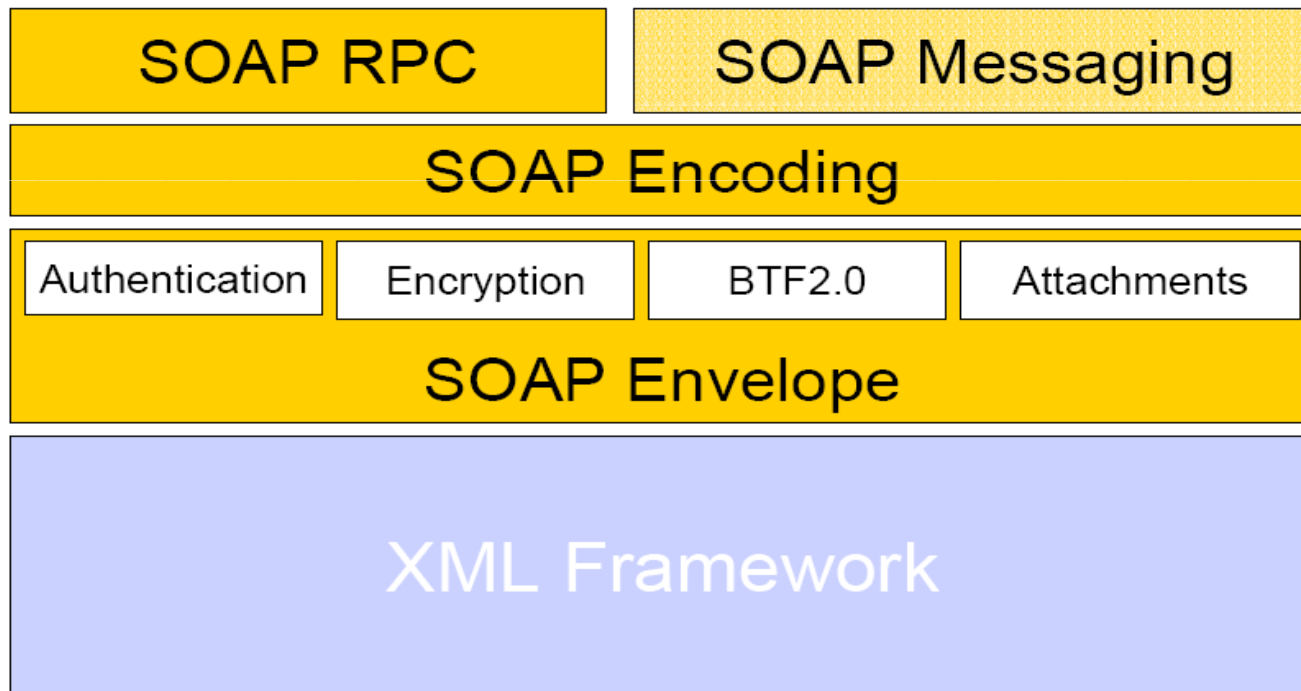
Estructura de SOAP

- ◆ Requiere un cuerpo (body) soap y un sobre (envelop)
- ◆ Cabeceras (headers) opcionales

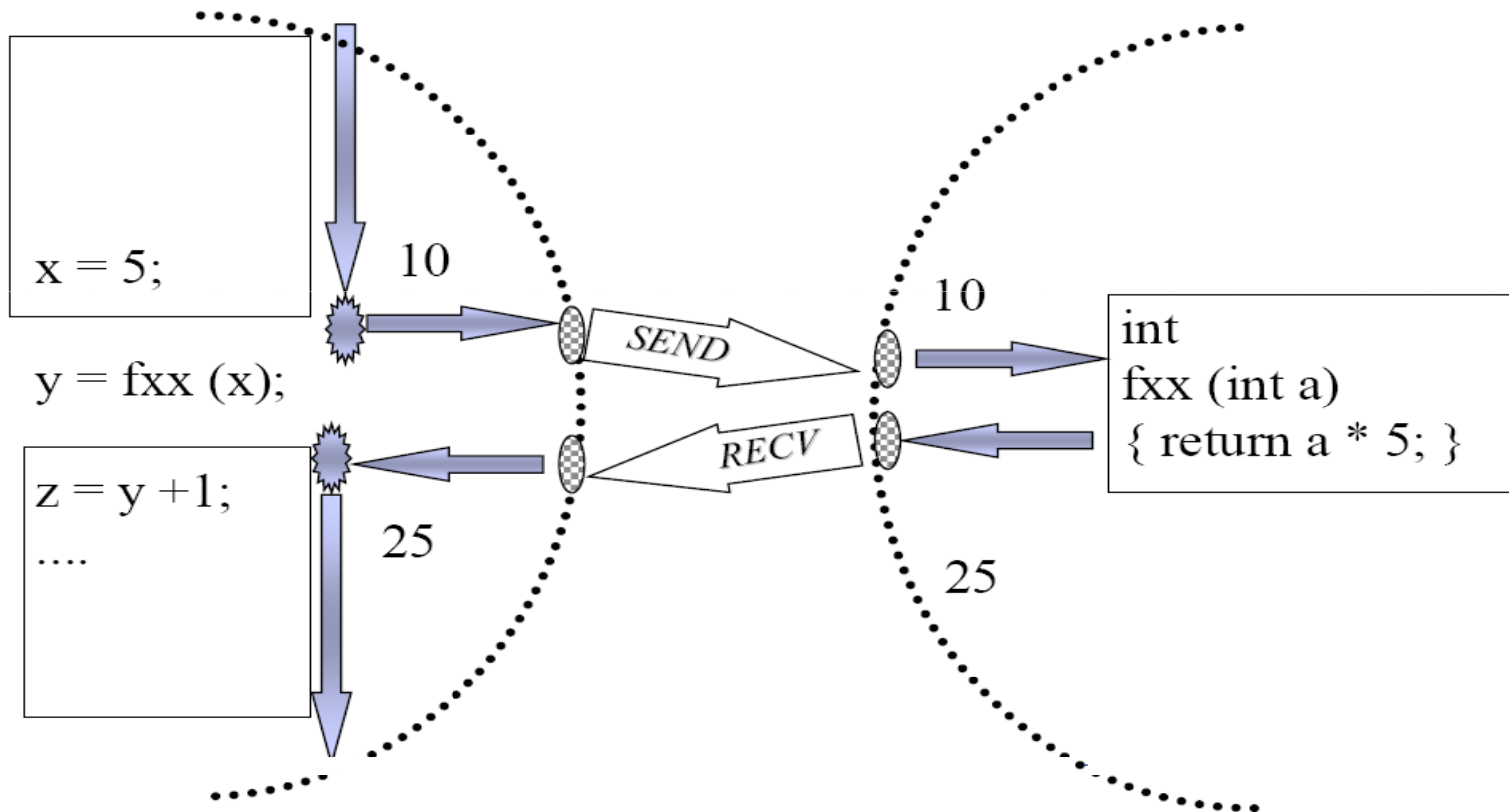


SOAP como protocolo

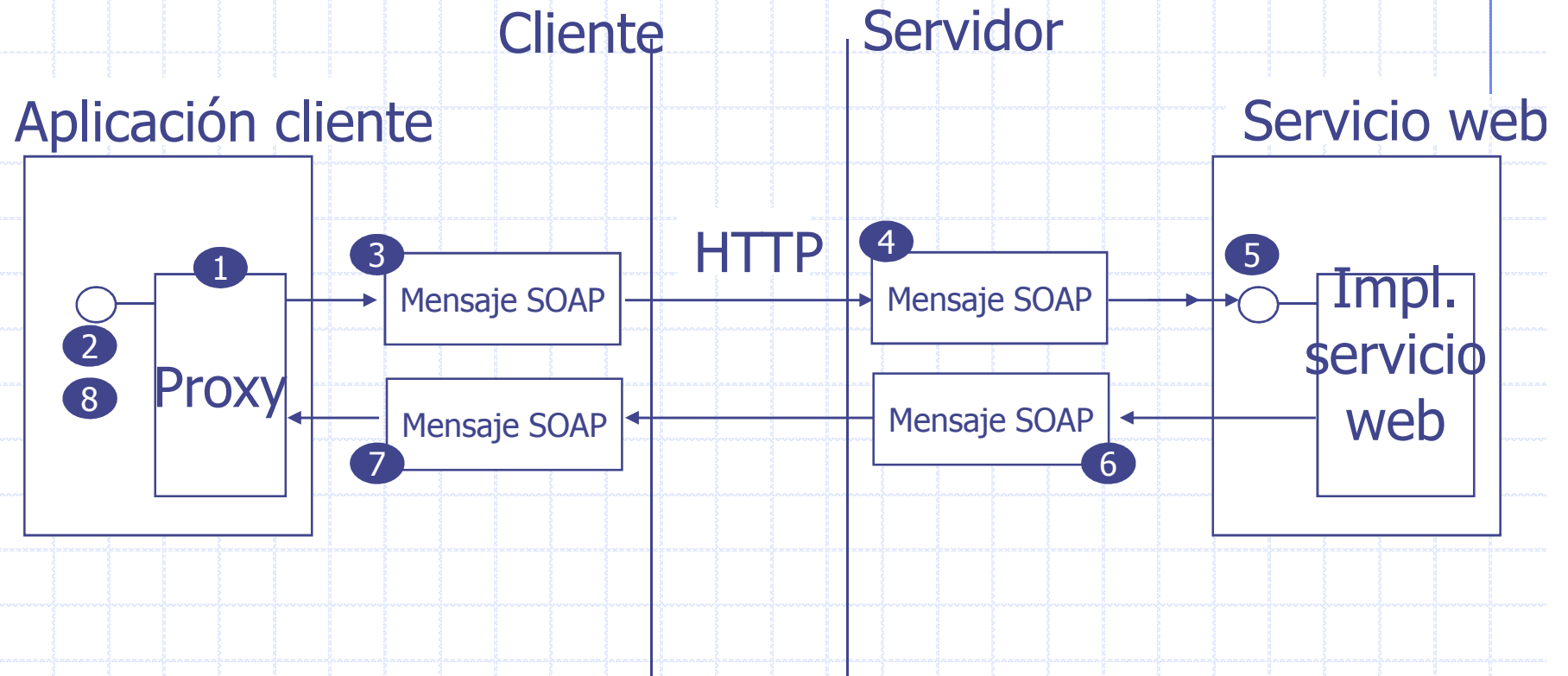
SOAP Se basa sobre XML



Modelo de llamada a procedimiento remoto (RPC)



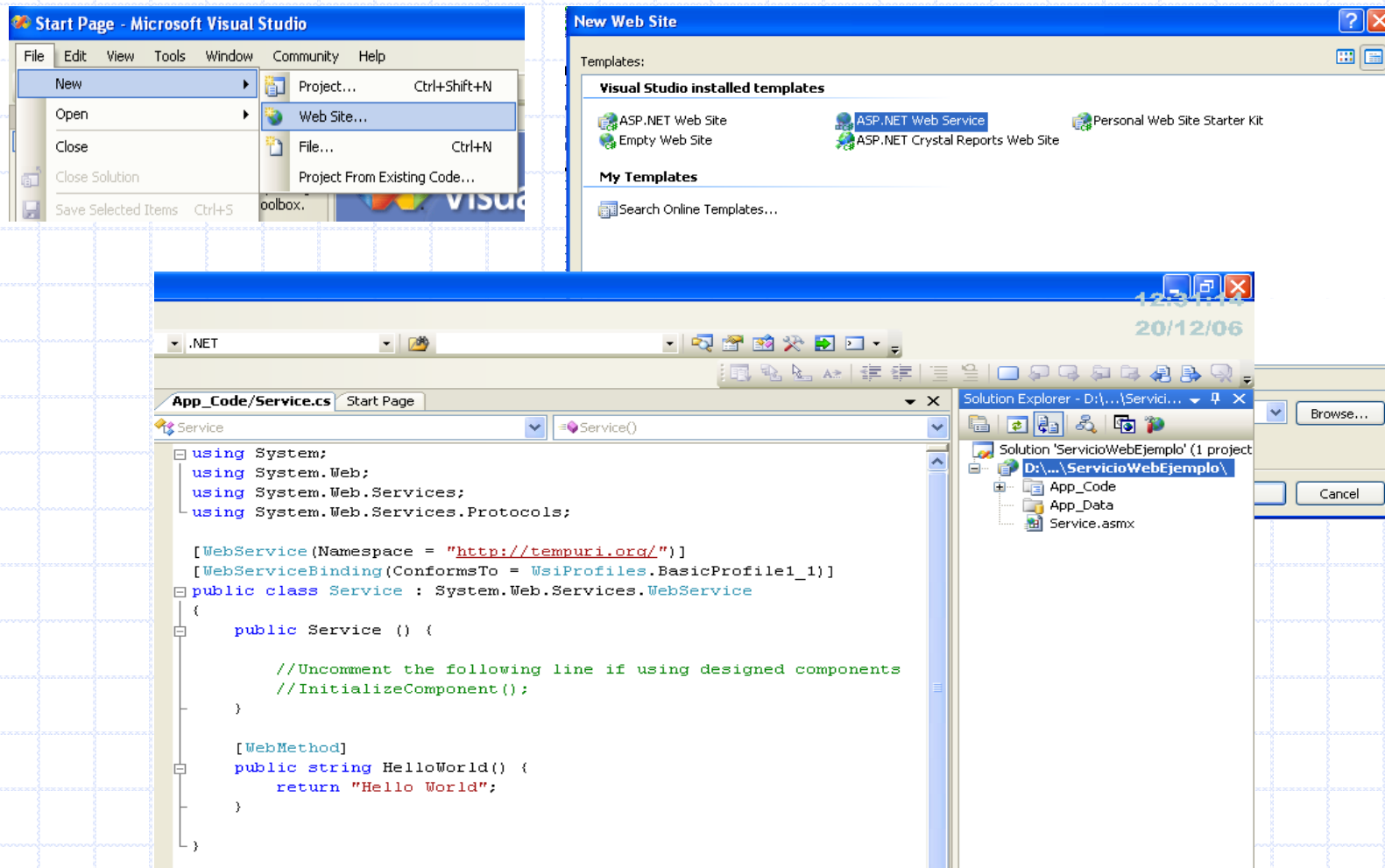
Interacciones



Limitaciones del .NET CF

Limitación	Descripción
DataSets tipados	No hay se soportan
Serialización y deserialización	La serialización binaria no se soporta a través de las clases <i>BinaryFormatter</i> o <i>SOAPFormatter</i> . Sin embargo la infraestructura de los servicios web soporta la transmisión de objetos utilizando SOAP

Un servicio web sencillo





CREACIÓN DE UN SERVICIO WEB SENCILLO

Un servicio web sencillo

The image illustrates the process of running a simple web service. It shows three main components:

- Solution Explorer:** A context menu is open over the project, with the 'View in Browser' option selected. Other options include 'Build Web Site', 'Publish Web Site', 'Add New Item...', 'Add Existing Item...', 'New Folder', 'Add ASP.NET Folder', 'Add Reference...', 'Add Web Reference...', 'View Class Diagram', 'Copy Web Site...', 'Start Options...', 'Set as StartUp Project', 'Browse With...', and 'Refresh Folder'.
- ASP.NET Development Server - Port 3099:** A configuration window showing the server settings:
 - Physical Path: D:\Usuarios\jcriguez\Visual Studio 2005\WebSites\ServicioW
 - Virtual Path: /ServicioWebEjemplo
 - Port: 3099
 - Root URL: <http://localhost:3099/ServicioWebEjemplo>A 'Stop' button is visible at the bottom right.
- Service Servicio Web - Mozilla Firefox:** A browser window displaying the web service page. The address bar shows <http://localhost:3099/ServicioWebEjemplo/Service.asmx>. The page content includes:
 - Service
 - Las operaciones siguientes son compatibles. Para una definición formal, revise la [descripción de servicios](#).
 - ◆ [HelloWorld](#)

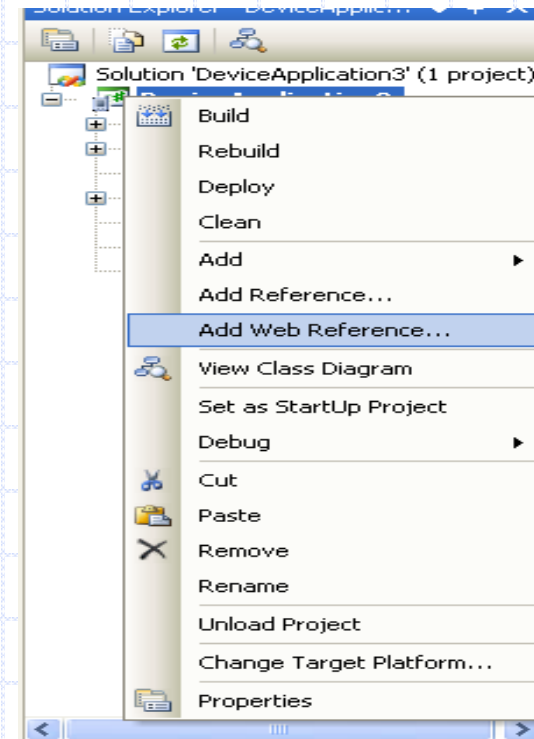
Arrows indicate the flow from the 'View in Browser' menu item to the development server window, and from the 'Open in Web Browser' option in the context menu to the browser window.



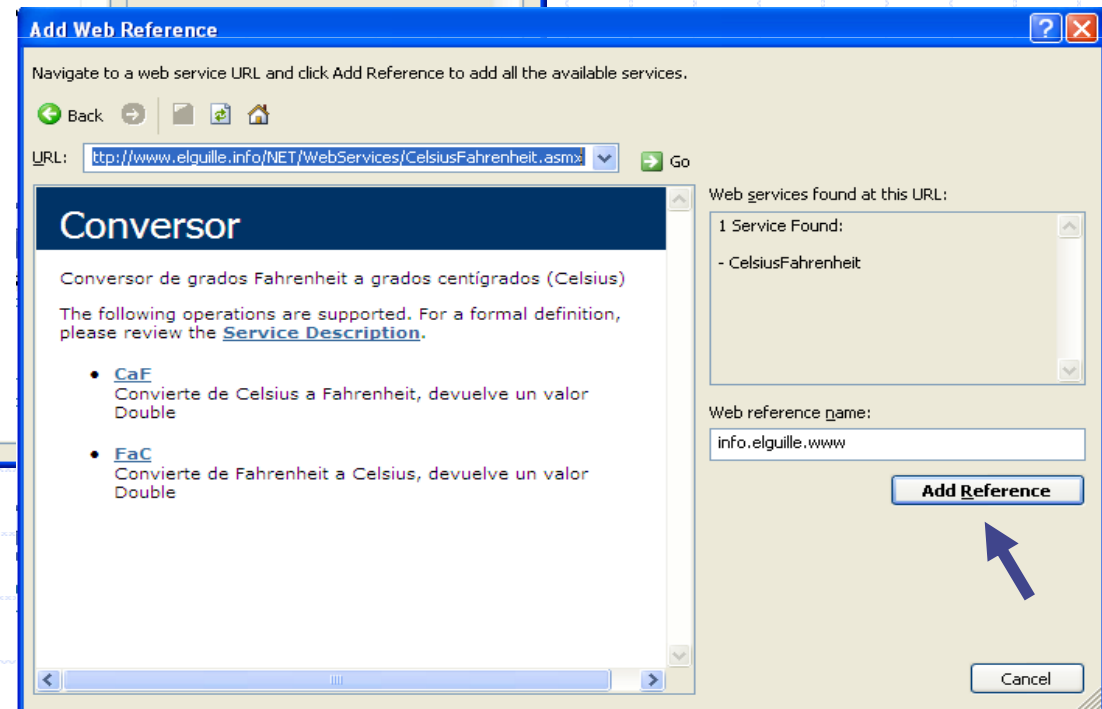
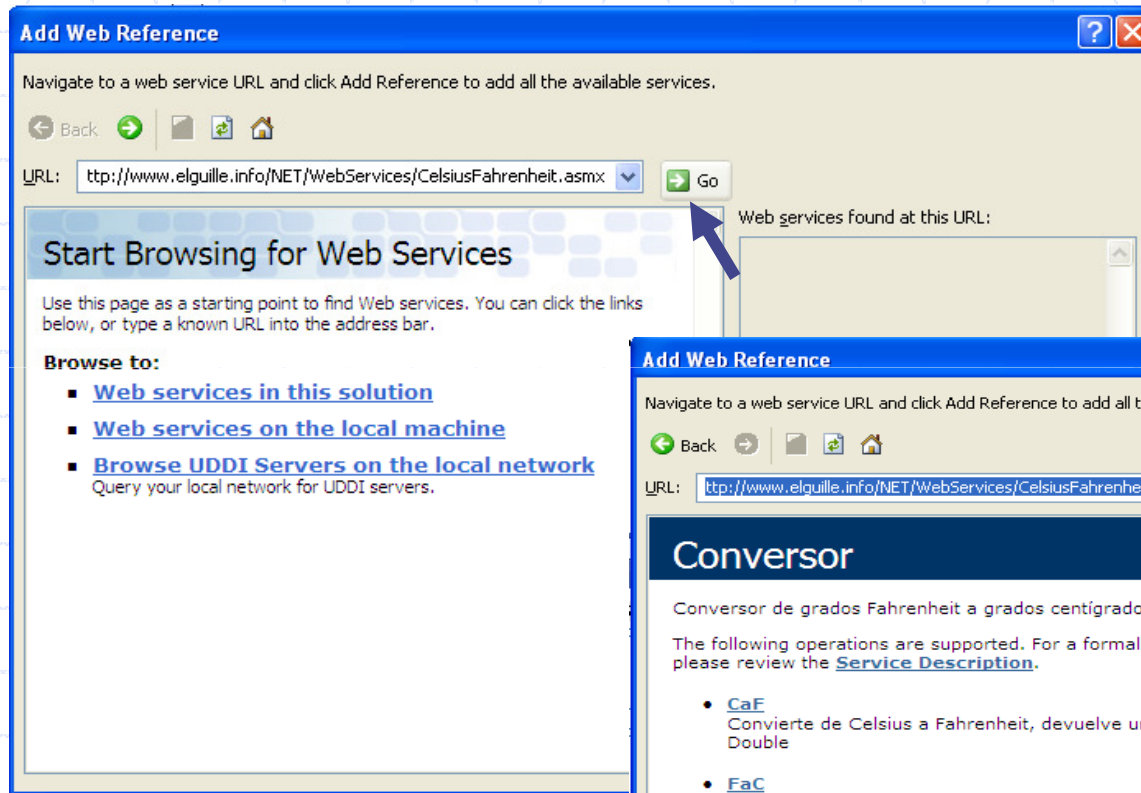
CREACIÓN DE UN CLIENTE .NET CF PARA EL SERVICIO WEB

Desarrollo de un cliente

- ◆ Primero generamos el proxy y luego lo utilizamos
- ◆ Para generar el proxy hay que añadir al proyecto la referencia del servicio web



Añadir referencia



Conf. del emulador y la PDA

- ◆ Configurar el emulador de la PDA para que salga a Internet
 - Utilizar como pasarela ActiveSync
- ◆ Configurar la PDA para que se conecte a Internet vía WIFI
 - Mirar el manual de conexión disponible en <http://infoacceso.upv.es>
 - Conexión a UPVNET2G (de preferencia)

Ventajas de utilizar servicios web

- ◆ Permite crear nuevos desarrollos más rápidamente y a un menor costo, debido a la reusabilidad
- ◆ Se pueden reutilizar los sistemas desarrollados por terceros encapsulándolos como servicios web
- ◆ Integra el proceso de negocio entre proveedores y socios a un menor costo.
- ◆ Se puede ampliar el mercado de negocios, al integrar los SW disponibles para nuevos desarrollos

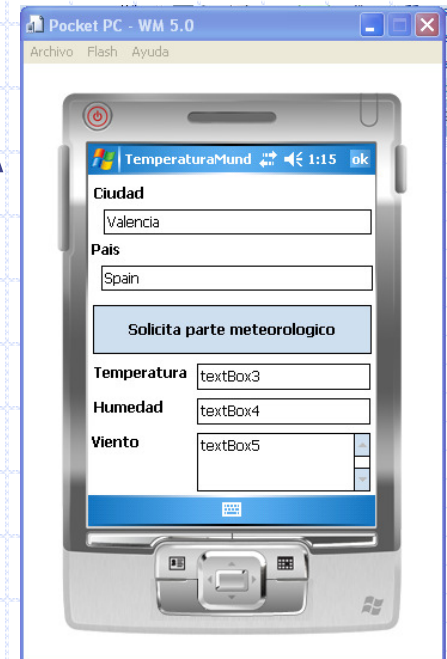
Algunos Web Services disponibles

- ◆ <http://www.elguille.info/NET/WebServices/HolaMundoWebS.aspx>
- ◆ <http://www.elguille.info/NET/WebServices/conversor.aspx>

=> Vamos a hacer un ejemplo en directo

Ejercicio

- ◆ Desarrollar una aplicación que utilice el web service de inf. meteorológica
 - <http://www.webservicex.com/globalweather.asmx>
- ◆ Probar el web service en la PDA



COMUNICACIONES HTTP

Espacio de nombres *System.Net*

- ◆ Contiene las clases necesarias para la gestión de comunicaciones en red
 - Dos clases básicas *WebRequest* y *WebResponse*
- ◆ Referencia los recursos disponibles en la red a través de URIs (*Uniform Resource Identifiers*)
 - Protocolo o mecanismo de acceso + `://`
Máquina + `[: + puerto] + /`
Identificador del recurso
 - Ejemplo:
`http://teraserver.microsoft.net/TerraService.asmx`

WebRequest yWebResponse

- ◆ Son clases abstractas → No instanciables
- ◆ Invocar el método estático *WebRequest.Create* con el URI del recurso web a utilizar para crear una instancia
 - La instancia creada depende del URI utilizado
 - ◆ *http://...* o *https://...* → *HttpWebRequest*
- ◆ El uso del método *GetResponse* sobre la instancia creada
 - Realiza la petición al recurso indicado
 - Retorna un objeto que descende de *WebResponse*
 - ◆ Objeto *HttpWebResponse* si se invoca *GetResponse* sobre una instancia de tipo *HttpWebRequest*

Uso *WebRequest/WebResponse*

```
using System.Net;
using System.IO;
...
WebRequest pet = WebRequest.Create(uri);
WebResponse res = pet.GetResponse();
//La llamada a GetResponse bloquea la aplicación hasta que
//se recibe la respuesta
Stream flujoRec = res.GetResponseStream();
string cadenaRec = flujoRec.ReadToEnd();
```

Comunicaciones por HTTP

Ejemplo: *screen scraper*

```
using System ;
using System.IO;
using System.Net;

public String DescargaPaginaWeb(String url){ //url es de tipo "http://..."
    String codigoHTML = null;
    HttpWebResponse respuesta = null; //respuesta podría ser de tipo WebResponse
    try{
        HttpWebRequest peticion = (HttpWebRequest) WebRequest.Create(url);
        respuesta = peticion.GetResponse() as HttpWebResponse;
        Stream flujoDeRecepcion = respuesta.GetResponseStream();
        StreamReader sr = new StreamReader(flujoDeRecepcion);
        codigoHTML = sr.ReadToEnd();
    }catch(Exception exp){
        MessageBox.Show(exp.Message);
    }finally{
        if (codigoHTML !=null) respuesta.Close();
    }
}
```

Gestión de errores

- ◆ El bloque *finally* nos asegura que el objeto *WebResponse* se libera independientemente del resultado de la petición → Liberación sistemática de recursos
- ◆ Es posible una gestión más refinada de los errores con las excepciones
 - *UriFormatException* → URI mal construida
 - *WebException* → Errores de protocolo
 - ◆ Uso de la propiedad *Status* de la excepción recogida para obtener más información

Gestión de errores

```
try{
    HttpWebRequest peticion = (HttpWebRequest) WebRequest.Create(url);
    ...
} catch(UriFormatException uriExcp){
    MessageBox.Show("URL incorrecta. Detalle: " + uriExcp.Message);
} catch(WebException webExcp){
    // WebExceptionStatus y webExcp.Status permiten determinar el motivo del problema
    // p.ej.: if (webExcp.Status == WebExceptionStatus.SendFailure){ ...}
    //         if (webExcp.Status == WebExceptionStatus.RequestCanceled) {...}
    if (webExcp.Status == WebExceptionStatus.ProtocolError){
        MessageBox.Show("Codigo del error: " +
            ((HttpWebResponse)webExcp.Response).StatusCode);
        MessageBox.Show("Descripción del error: " +
            ((HttpWebResponse)webExcp.Response).StatusDescription);
    }
} catch(Exception excp){
    MessageBox.Show("Problema desconocido. Detalle: " + excp.Message);
} finally{
    if (respuesta!=null) respuesta.Close();
}
```

Gestión de cabeceras HTTP

- ◆ El protocolo HTTP define una serie de cabeceras → RFC2616 (cabeceras HTTP 1.1)
- ◆ Las cabeceras indican al cliente y al servidor como interpretar la información que reciben
- ◆ *HttpWebRequest* y *HttpWebResponse* permiten consultarlas y gestionarlas
 - Uso de la propiedad *Headers* de tipo *System.Net.WebHeaderCollection*
 - Ciertas cabeceras están disponibles como propiedades de las instancias de *HttpWebRequest* y *HttpWebResponse*

Gestión de cabeceras HTTP (II)

```
HttpWebResponse respuesta = null;
try{
    HttpWebRequest peticion = (HttpWebRequest) WebRequest.Create(url);
    peticion.Headers.Add("User-Agent", "Ejemplo, C#, .NETCF, WM5");
    //El usuario puede definirse sus propias cabeceras
    peticion.Headers.Add("Mi_Cabecera", "Mi_valor");

    HttpWebResponse res = peticion.GetResponse() as HttpWebResponse;
    String lonMensaje = res.Headers["Content-Length"];
}
...
```

Codificación

- ◆ Toda la información transmitida por la red está codificada
- ◆ Windows CE utiliza Unicode (al menos 2 bytes) para representar cada caracteres
 - Los caracteres US ASCII sólo necesitan 1 byte
 - El uso de 1 byte implica no poder representar caracteres utilizados en otros lenguajes
- ◆ La codificación empleada se refleja en el atributo Content.Encoding de la cabecera HTTP
 - Por defecto uso de UTF8 → Codificación multibyte de longitud variable

Codificación

- ◆ El tipo de codificación utilizado en un objeto *HttpWebResponse* puede determinarse utilizando la propiedad *CharacterSet*
 - IMP: En la petición hay que indicar mediante el atributo *MediaType* el tipo de datos esperados
 - ◆ `pet.MediaType = "text/html"`
 - Si el tipo no coincide o *MediaType* no se ha indicado → *CharacterSet* será null
 - ◆ `string contentType = res.ContentType;`
`//Debería ser "text/html; charset=utf-8"`
 - ◆ `string charsetEncoding = res.CharacterSet;`
`//Debería ser utf-8, pero podría ser utf-7, iso-10646, us-ascii,`
`//etc...`

Codificación (IV)

```
...
string codificacionCaracteres = respuesta.CharacterSet;

Stream flujoDeRecepcion = respuesta.GetResponseStream();
//Utilizando la informacion recabada para la codificación de caracteres es
//posible instanciar un StreamReader que lea la información atendiendo a su
//codificación
StreamReader sr = null;
if (codificacionCaracteres.Length>0){
    System.Text.Encoding encode =
        System.Text.Encoding.GetEncoding(codificacionCaracteres);
    sr = new StreamReader(flujoDeRecepcion, encode);
}else {
    //No hay una codificación específica a respetar
    sr = new StreamReader(flujoDeRecepcion);
}
//Almacenamos el contenido del flujo de recepción en un String
string codigoHTML = sr.ReadToEnd();
```

HTTP GET

- ◆ Las peticiones HTTP GET se codifican junto al URI
 - Formato: URI?clave1=valor1&...&claven=valorn
 - Ejemplo
 - ◆ http://www.optize.es/servlet/navigation?nombre=PDA_QTEK&category=16408&fabri=1072
 - En la petición no pueden aparecer caracteres típicamente utilizados en URLs (:,/,?), ni espacios
 - ◆ La clase System.Web.HttpUtility permite codificar las URL para evitar problemas en .NET, pero NO ESTA DISPONIBLE EN .NET CF

Solución al problema

```
public static string UriEncode(string cadenaEntrada){
    StringReader strRdr = new StringReader(cadenaEntrada);
    StringWriter strWtr = new StringWriter();
    int valorCar = strRdr.Read();
    while (valorCar != -1){
        if (((valorCar >= 48) && (valorCar<=57))           // 0-9
            ||((valorCar >= 65) && (valorCar<=90))       // A-Z
            ||(valorCar >= 97) && (valorCar<=122)))      // a-z
            strWtr.Write((char) valorCar);
        else if (valorCar == 32)                          // espacio
            strWtr.Write(' ');
        else
            strWtr.Write("{0:x2}", valorCar);
        valorCar = strRdr.Read();
    }
    return strWtr.ToString();
}
```

HTTP POST

- ◆ La petición se codifica en el cuerpo del mensaje HTTP
 - Uso típico del formato de parejas clave=valor
 - Si el cliente y el servidor se ponen de acuerdo el formato puede ser cualquiera (bytes, XML, etc.)
- ◆ La propiedad *HttpWebRequest.Method* debe ser "POST"
- ◆ Gestión del contenido transferido
 - Los datos deben convertirse en un array de bytes
 - Posicionar adecuadamente la propiedad *ContentLength*
 - Posicionar adecuadamente la propiedad *ContentType*
 - ◆ {text/plain, text/xml, application/x-www-form-urlencoded (tipo MIME)};
charset={utf-8, utf-7, ISO-8859-1}
 - Uso de *GetRequestStream* para poder obtener el flujo de comunicación y escribir en él los datos

HTTP POST (II)

```
HttpRequest peticion = null;
HttpResponse respuesta = null;
string stringRespuesta = null;
try{
    peticion = (HttpRequest) WebRequest.Create("http://www.optize.es/servlet/navigation");
    peticion.Method = "POST";
    //Preparar datos y posicionar propiedades en la petición
    string mensajeAEnviar = "nombre=PDA_QTEK&category=16408&fabri=1072";
    byte[] bytesCodificados = System.Text.Encoding.UTF8.GetBytes(mensajeAEnviar);
    peticion.ContentLength = bytesCodificados.Length;
    peticion.ContentType = "text/plain; charset=utf-8";
    //Enviar los datos
    Stream s = peticion.GetRequestStream();
    s.Write(bytesCodificados, 0, bytesCodificados.Length);
    s.Close();
    //Gestionar la respuesta
    respuesta = peticion.GetResponse() as HttpResponse;
    StreamReader sr = new StreamReader(respuesta.GetResponseStream());
    stringRespuesta = sr.ReadToEnd();
} catch (Exception excp) {
    MessageBox.Show("Problema: " + excp.Message);
} finally {
    if (respuesta != null) respuesta.Close();
}
```

Ejemplo HTTP

- ◆ El servidor accuweather ofrece vía GET previsiones meteorológicas a 5 días
 - La URL a utilizar para conocer el tiempo en Valencia es
 - ◆ http://htc.accuweather.com/widget/htc/forecast-data_v3.asp?ac=TR2cra9U&locCode=EUR|ES|SP016|VALENCIA
 - La respuesta a esta llamada está codificada en XML
- ◆ Diseñar una aplicación .NET CF que realice la información y muestre la previsión

Peticiones Web asíncronas

- ◆ Las peticiones web pueden tener un coste temporal importante
- ◆ Peticiones asíncronas
 - *WebRequest.BeginGetResponse*
(método de callback, objeto de estado)
 - El thread principal de la aplicación puede continuar su ejecución mientras que el encargado de la petición queda en segundo plano

Peticiones Web asíncronas (II)

- El objeto de estado sirve de conexión lógica entre el thread que efectúa inicialmente la petición y el que finalmente recibe la respuesta
- El objeto de estado puede tener la estructura que deseemos, pero siempre debe contener una referencia a la petición inicial (objeto *HttpRequest*)

Peticiones Web asíncronas (III)

```
public class GET_Asyncrono{
    ...
    public void RealizaGET(String url) {
        try {
            HttpWebRequest pet = (HttpWebRequest) WebRequest.Create(url);
            // Creamos el objeto que nos servirá para tratar el estado de la
            // petición y los asociamos
            RequestState ep = new EstadoPeticion();
            ep.Request = pet;
            pet.BeginGetResponse(new AsyncCallback(this.MetodoDeRespuesta), ep);
        }catch(Exception exp) {
            Message.Box(exp.Message);
        }
    }
}
```

```
public class EstadoPeticion{
    public HttpWebRequest Peticion;
    public EstadoPeticion() {
        Peticion = null;
    }
}
```

Peticiones Web asíncronas (IV)

```
...
private void MetodoDeRespuesta(IAAsyncResult ar){
    // Obtenemos el estado de la petición (objeto EstadoPeticon)
    // a través del resultado de la llamada asíncroa (objeto ar de tipo IAAsyncResult)
    EstadoPeticon rs = (EstadoPeticon) ar.AsyncState;
    HttpRequest req = rs.Peticon;

    // ... y a través de dicha petición se obtiene el objeto de respuesta (tipo HttpResponseMessage).
    HttpResponseMessage resp = (HttpResponseMessage) req.EndGetResponse(ar);

    // A partir del objeto de respuesta a la petición producimos un flujo de datos ...
    Stream flujoDeDatosRespuesta = resp.GetResponseStream();
    StreamReader sr = new StreamReader( flujoDeDatosRespuesta, Encoding.UTF8);
    string respuestaRecibida = sr.ReadToEnd();
    flujoDeDatosRespuesta.Close();
    PeticonCompletada();
}

public delegate void ManejadorPeticonCompletada();
public event ManejadorPeticonCompletada PeticonCompletada;
```

Peticiones Asíncronas (V)

```
class PeticionesAsincronas{

    private bool peticionCompletada = false;
    public Get_Asincrono ga = new Get_Asincrono(...);

    public static void Main(string[] args){
        PeticionesAsincronas pa = new PeticionesAsincronas();
        pa.Ejecuta();
    }
    public void Ejecuta(){
        // Asociamos el manejador al evento GetComplete
        ga.PeticionCompletada += new GET_Asincrono.ManejadorPeticionCompletada(this.NotificarFinPeticion);
        ga.RealizaGET(...);
        // Esta variable booleana nos sirve para indicar cuando una peticion se completa
        this.peticionCompletada = false;
        while (!peticionCompletada){
            //escribimos por pantalla un punto
            this.textBox2.Text += ".";
            //... y nos dormimos durante medio segundo
            Thread.Sleep(500);
        }
        //Si salimos del bucle es porque se ha recibido la respuesta y podemos terminar
        this.textBox2.Text += "\n Respuesta recibida";
        return;
    }

    protected void NotificarFinPeticion(){
        // Se ha recibido la respuesta
        peticionCompletada = true;
    }
}
```

Determinar si \exists conexión

- ◆ ¿Cómo saber si un dispositivo está o no conectado a la red?
 - No hay ninguna clase que nos notifique sobre la pérdida o recuperación de la conexión
 - Crear una instancia de *System.Threading.Timer* y sobrecargar su método *TimerEvent*
 - ◆ Más información en: <http://msdn2.microsoft.com/en-us/library/system.threading.timer.aspx>
 - A cada evento realizar una *WebRequest* de tipo GET al servidor. Si la propiedad *HttpWebResponse.StatusCode* es *HttpStatusCode.OK* se confirma la conectividad

Programación con Timers

```
using System;
using System.Threading;
using System.Windows.Forms;
using Timer = System.Threading.Timer;

public enum ConnectionState{ Unknown, Connected, Disconnected };
public class Connection{
    private string url;
    public Connection(string url){ this.url = url;}
    public IsAvailable(){ ... }
}
public class ConnectionMonitor{

    private ConnectionState _state;
    private System.Threading.Timer _timer;
    private Int32 _waitTime = 2500; // 2.5 seconds.

    public int WaitTime {
        get{ return _waitTime; }
        set{ if( !_waitTime.Equals( value ) ) _timer.Change( value, Timeout.Infinite ); }
    }

    public ConnectionMonitor(string url ) {
        _state = ConnectionState.Unknown;
        _timer = new Timer( new TimerCallback( TimerTick ), new Connection(url), WaitTime, Timeout.Infinite );
    }

    private void TimerTick( Object obj ) {

        bool connected = ((Connection) obj).IsAvailable();

        if ((connected) && ( _state != ConnectionState.Connected )) _state = ConnectionState.Connected;
        else if( _state != ConnectionState.Disconnected) _state = ConnectionState.Disconnected;
    }
}
```



COMUNICACIONES A TRAVES DE SOCKETS

El espacio *System.Net.Sockets*

- ◆ Proporciona una implementación gestionada para .NET de la librería *Winsock*
- ◆ Com. orientadas a conexión (TCP/IP)
 - Objeto a la espera de peticiones TCP sobre un socket (instancia de la clase *TcpListener*)
 - Objeto que realiza peticiones TCP a través de un socket (instancia de la clase *TcpClient*)
- ◆ Com. sin conexión (UDP/IP)
 - Envío de datagramas a través de un socket (instancia de la clase *UdpClient*)

Gestión de direcciones IP

◆ Análisis sintáctico de IPs

- Uso de la clase `System.Net.IPAddress`

```
System.Net.IPAddress dirIPLocal =  
    System.Net.IPAddress.Parse("127.0.0.1");
```

◆ Gestión combinada de IPs y puertos

- Uso de la clase `System.Net.IPEndPoint`

```
System.Net.IPEndPoint localIPEndpoint =  
    new System.Net.IPEndPoint(localIPAddress, 80);
```

Resolución de nombres y direcciones

◆ Uso de la clase *System.Net.Dns*

```
System.Net.IPHostEntry servidorGooglePorNombre =  
    System.Net.Dns.GetHostByName("www.google.es");
```

.NET CF 1.x

```
System.Net.IPHostEntry servidorGooglePorIP =  
    System.Net.Dns.GetHostByAddress("64.233.183.103");
```

```
System.Net.IPHostEntry servidorGooglePorNombre =  
    System.Net.Dns.GetHostEntry("www.google.es");
```

.NET CF 2.0

```
System.Net.IPHostEntry servidorGooglePorIP =  
    System.Net.Dns.GetHostEntry("64.233.183.103");
```

```
string nombreMaquina = System.Net.Dns.GetHostName();
```

.NET CF X.X

Lado del servidor

- ◆ Uso de la clase *TcpListener*
- ◆ Escucha en una IP y un puerto dados
 - Máquinas con una única interfaz de red

```
//1. La IP es la de la máquina y se especifica el puerto
TcpListener servidorTCP = new TcpListener(695);

//2. La IP es la de la máquina y se deja en el sistema la
// responsabilidad de asignar un puerto libre al servicio
TcpListener servidorTCP = new TcpListener(0);
servidorTCP.Start(); //AQUI se asigna el puerto al servicio
System.Net.IPEndPoint epAsignado =
    (System.Net.IPEndPoint) servidorTCP.LocalEndpoint;
int puerto = epAsignado.Port; //Averiguamos cuál es el puerto
```

Lado del servidor (II)

- Máquinas con varias interfaces de red

```
//1. Especificamos la interfaz que deseamos utilizar
IPEndPoint endPt =
    new IPEndPoint(IPAddress.Parse("192.168.0.1"), 695);
TcpListener servidorTCP = new TcpListener(endPt);
```

```
//2. La interfaz a utilizar no importa, la elige el sistema
IPEndPoint endPt =
    new IPEndPoint(IPAddress.Any /*0.0.0.0*/, 695);
TcpListener servidorTCP = new TcpListener(endPt);
```

```
//Averiguar en a través de qué interfaz y puerto ofrecemos servicio
IPHostEntry miHost = Dns.GetHostEntry(Dns.GetHostName());
Console.WriteLine("Máquina {0} escuchando en {1}, puerto {2}",
    miHost.HostName, endPt.Adress, endPt.Port);
```

Lado del servidor (III)

- ◆ Comenzar/Parar la escucha de peticiones
 - `objTcpListener.Start()` / `objTcpListener.Stop()`
 - Gestión de las peticiones a través de una cola interna
- ◆ Aceptar conexiones (llamadas con bloqueo)
 - Opción 1: `objTcpListener.AcceptSocket()`
 - Opción 2: `objTcpListener.AcceptTcpClient()`
- ◆ Aceptar conexiones (llamadas sin bloqueo)
 - `objTcpListener.Pending()`
 - ◆ Devuelve *true* si hay alguna petición pendiente en la cola interna
- ◆ Intercambio de información
 - Opción 1: `Socket.Send` / `Socket.Receive`
 - Opción 2: `TcpClient.GetStream`
- ◆ Las conexiones NO se cierran automáticamente

Ejemplo

```
using System;  
using System.Net;  
using System.Net.Sockets;  
using System.Text;
```

```
namespace ProgramasConSockets{  
    class EjemploUsoTcpListener{  
        public static void Main() {  
            String repuesta;  
            try{  
                IPEndPoint miHost = Dns.GetHostEntry(Dns.GetHostName());  
                System.Net.IPEndPoint miEndPoint =  
                    new System.Net.IPEndPoint(miHost.AddressList[0], 0);  
                TcpListener servidorTCP = new TcpListener(miEndPoint);  
                System.Net.IPEndPoint epAsignado =  
                    (System.Net.IPEndPoint)tcpServer.LocalEndpoint;  
                int puerto = epAsignado.Port;  
                Console.WriteLine("Máquina {0} escuchando en {1}, puerto {2}",  
                    miHost.HostName, miHost.AddressList[0].ToString(), puerto);  
                servidorTCP.Start();  
                Console.WriteLine("Esperando alguna conexión ...");  
                TcpClient clientConn = tcpServer.AcceptTcpClient();
```

```
Máquina juankarp escuchando en 192.168.123.137, puerto 1198  
Esperando alguna conexión ...  
-
```

Ejemplo

```
Stream strm = clientConn.GetStream();
byte[] bufEntrada = new Byte[4];
int bytes = bytes = strm.Read(bufEntrada, 0, bufEntrada.Length);
string mens += Encoding.ASCII.GetString(bufEntrada, 0, bytes);
respuesta = "Mensaje: " + mens + ", recibido y retornado a las "
            + DateTime.Now.ToLongTimeString();
byte[] bytesRespuesta =
    Encoding.ASCII.GetBytes(respuesta.ToCharArray());
clientConn.GetStream().Write(bytesRespuesta, 0, bytesRespuesta.Length);
clientConn.Close();
}catch (SocketException socketError){
    if (socketError.ErrorCode == 10048){
        Console.WriteLine("Error, puerto en uso");
    }
}
} //Main
} //clase EjemploUsoTcpListener
} //namespace ProgramasConSockets
```

El lado del cliente

◆ Uso de la clase *TCPClient*

```
private void button1_Click(object sender, System.EventArgs e){
    IPAddress serverIP;
    int puerto = Int16.Parse(ServerPortBox.Text);
    try{
        serverIP = IPAddress.Parse(ServerTextBox.Text);
    }catch (FormatException fexp){
        this.ResponseTextBox.Text = "Dir. IP inválida – se pide n.n.n.n, por ejemplo 192.168.0.103";
        return;
    }
    IPEndPoint remoteHost; //Verificamos si el host especificado está o no accesible
    try{
        remoteHost = Dns.GetHostEntry(ServerTextBox.Text);
    }catch (SocketException sexp){
        if (sexp.ErrorCode == 11001) // Host desconocido
            ResponseTextBox.Text = String.Format("Server {0} not known", ServerTextBox.Text);
        else
            ResponseTextBox.Text = String.Format("DNS lookup of {0} caused error: {1}",
                ServerTextBox.Text, sexp.Message);
    }
    return;
}
```



Ejemplo: Cliente

```
TcpClient client = new TcpClient();
client.Connect(new IPEndPoint(serverIP, puerto));
Stream strm;
try {
    strm = client.GetStream();
} catch (InvalidOperationException) {
    ResponseTextBox.Text = String.Format("No puedo conectar con: {0}", ServerTextBox.Text);
    return;
} catch (SocketException exc) {
    StringBuilder strB = new StringBuilder("");
    strB.Append(String.Format("No puedo conectar con: {0}\r\n", ServerTextBox.Text));
    strB.Append(exc.Message + "\r\n");
    strB.Append("Código de error del Socket: " + exc.ErrorCode.ToString());
    ResponseTextBox.Text = strB.ToString();
    return;
}
String str = textBox2.Text + '\0';
Byte[] bytes = Encoding.ASCII.GetBytes(str);
strm.Write(bytes, 0, bytes.Length);
Byte[] inputBuffer = new Byte[256];
int nbytes = strm.Read(inputBuffer, 0, inputBuffer.Length);
string response = Encoding.ASCII.GetString(inputBuffer, 0, nbytes);
ResponseTextBox.Text = response;
client.Close();
}
```

Información de los ejemplos

- ◆ Directorio: ConexionTCPporSockets
 - Cliente: TCPClientSample
 - Servidor: TCPServerSample

Otro ejemplo

- ◆ Variante del anterior en el que hay 2 iteraciones:
 - 1ª App1 (cliente) → App2 (servidor)
 - 2ª App1 (servidor) ← App2 (cliente)
- ◆ Ejemplo en proyectos
 - TCPClientSampleConRespuesta
 - TCPServerSampleConRespuesta

La problemática de configurar el emulador de la PDA

- ◆ Solución 1: Uso de ActiveSync → La PDA no se encuentra en ninguna red

La problemática de configurar el emulador de la PDA

- ◆ Solución 2: Uso de Virtual Ethernet Network Adapter
 - Requiere instalar el Virtual PC, disponible en <http://www.microsoft.com/downloads/details.aspx?displaylang=es&FamilyID=04d26402-3199-48a3-afa2-2dc0b40a73b6>
 - Procedimiento de configuración del emulador disponible en <http://netcf2.blogspot.com/2005/11/networking-ping-emulator.html>
 - Permite al emulador ser una máquina más de la red sin necesidad de una interfaz física adicional a la que ofrece la tarjeta de red → ii Es posible hacerle PING y todo!!
 - ◆ En caso de problemas a la hora de indicar la interfaz física a utilizar ir a Herramientas → Opciones → Herramientas de dispositivo → Dispositivo → Elegir el emulador → Propiedades → Opciones del emulador → Red → Habilitar adaptador de red PCMCIA NE2000 y enlazar a → elegir la interfaz → Aceptar (3 veces) → Lanzar el emulador
 - ◆ Si el estado del emulador se guardó puede que el cambio no surta efecto. Herramientas → Opciones → Herramientas de dispositivo → Dispositivo → Elegir emulador → Guardar como (dar otro nombre) → Repetir los pasos de configuración del punto anterior → Lanzar ahora el emulador con el nuevo nombre dado
 - ◆ iii Ojo con las direcciones 169.254.X.X !!! => No son buenas

La problemática de configurar el emulador de la PDA

- ◆ Solución 3: En ocasiones, y debido a cuestiones de seguridad, no es posible que el emulador integre la red (el servidor de DHCP no le asigna una IP). En ese caso:
 - Instalar el adaptador de bucle invertido
 - ◆ <http://support.microsoft.com/kb/839013/es>
 - Inicio→Conectar a→Mostrar todas las conexiones
 - Propiedades del interfaz de red conectado a internet → Opciones avanzadas → Habilitar "Permitir a usuarios de otras redes conectarse a través de la conexión a Internet de este equipo" -> Seleccionar la interfaz de red asociada al adaptador de bucle invertido
 - Siguiendo el procedimiento de la solución 2 asociar al emulador de la PDA el adaptador de bucle invertido como interfaz de red → Apuntar cuál es la dirección IP asignada (192.168.X.X)
 - Ir a Propiedades del interfaz de red conectado a internet → Opciones avanzadas → Configuración avanzada → Agregar → Descripción del servicio (A elegir) + IP (la del emulador, 192.168.X.X) ++ puertos (el que asignemos al emulador, p.ej. 8888) +Aceptar
- ◆ En este caso el emulador pertenece a una red de área local virtual y la tarjeta de red física redirecciona sus comunicaciones a internet y todas las que provengan del exterior y se dirijan al puerto 8888 se reenviarán a la PDA

Consejo

- ◆ Servisar los distintos ejemplos proporcionados con las distintas configuraciones del emulador descritas
 - Uso de ActiveSync siempre excepto cuando la PDA deba ser direccionada desde el exterior
 - En ese caso, la solución 2 es aconsejable, pero si no es aplicable, usar la solución 3