Programación distribuida en .NET (y III)

DIEGO LZ. DE IPIÑA GZ. DE ARTAZA (Profesor del departamento de Ingeniería del Software de la facultad de Ingeniería de la Universidad de Deusto (ESIDE)).

Acabamos esta serie sobre programación distribuida en .NET, estudiando cómo utilizar .NET Remoting para generar servicios web, analizando el mecanismo de mensajería MSMQ y revisando Enterprise Services, el mecanismo .NET para el desarrollo de aplicaciones de empresa.

Introducción

Como se describió en la anterior entrega, el espacio de nombres "System.Runtime.Remoting" provee la infraestructura .NET para el desarrollo de aplicaciones fuertemente acopladas, acorde con el mecanismo de comunicación RPC (Remote Procedure Call). En esencia, .NET Remoting combina estándares ya existentes como SOAP (Simple Object Access Protocol), para codificación de mensajes, o HTTP y TCP, para la comunicación de mensajes, con formateadores binarios propietarios destinados a conseguir un alto rendimiento en las comunicaciones entre objetos distribuidos. Consiste en un conjunto de servicios y canales de comunicación que transmiten mensajes entre aplicaciones remotas, con la ayuda de formateadores que codifican y decodifican esos mensajes.

En esta entrega continuamos nuestro estudio de .NET Remoting mostrando su capacidad poco conocida de servir como infraestructura no sólo para la generación de servidores y clientes fuertemente acoplados sino también para la creación de servicios web. Además, analizamos un mecanismo de comunicación alternativo más robusto que .NET Remoting, basado en colas de mensajes, denominado MSMQ. Finalmente, introducimos Enterprise Services, la solución .NET para el desarrollo de aplicaciones escalables con altas demandas de rendimiento. Enterprise Services representa la proposición de Microsoft para competir con la solución Java para el desarrollo de aplicaciones de empresa, J2EE.

Recomendamos al lector mantenerse atento a los progresos de Indigo, el futuro de la programación distribuida en .NET

.NET Remoting y servicios web

Una característica de .NET Remoting muy interesante, aunque desconocida por muchos, es que un servidor en .NET Remoting puede ser exportado como si se tratase de un servicio web. En definitiva, podemos programar servicios web en .NET no sólo por medio de ASP.NET, si no que también con .NET Remoting. Esto es debido a que podemos acceder a un servidor .NET Remoting a través de un canal HTTP capaz de procesar peticiones a métodos remotos enviadas en formato SOAP. Los requisitos que deberá cumplir nuestro servidor de .NET Remoting para aparecer externamente como un servicio web serán:

1- Hacer uso de "HttpChannel".
2- Estar activado en la parte servidora.

A continuación, mostramos los pasos para generar tanto la parte servidora como cliente de un servicio web desarrollado con .NET Remoting, para nuestro conocido conversor de números romanos y árabes. Como prerrequisito es necesario tener instalado en nuestra máquina y en ejecución el servidor web IIS.

Parte servidora del servicio web con .NET Remoting

El código de la parte servidora sería idéntico al que mostramos en la entrega anterior. A modo de recordatorio, en el listado 1 se muestra el interfaz, o contrato, disponible para su consumo por los potenciales clientes. También se muestra en ese mismo listado el comienzo de la implementación del mismo en la clase "ConversorRomanoArabe". El código de la parte servidora se ha incluido en el CD-ROM número 125 que acompaña a la revista, en el fichero "RemotingHTTPWSDL/ConversorRomanoArabeServer.cs".

Efectuamos el despliegue de este servidor/servicio web, ayudados por la capacidad de activación de servidores .NET Remoting provista por IIS. En primer lugar, usamos el gestor de IIS ("inetmgr") para crear un directorio virtual de nombre "ConversorRomanoArabe". Este mismo proceso lo realizamos, de manera más pormenorizada, en la anterior entrega de esta serie. A continuación, crearemos un fichero "web.config" en ese directorio

LISTADO 1 Fragmento del fichero RemotingHTTPWSDL/ConversorRomanoArabeServer.cs

```
public interface IConversorRomanoArabe
{
    bool Login(Usuario usuario, out string token);
    bool Logoff(Usuario usuario, string token);
    string ArabeARomano(string token, int num);
    int RomanoAArabe(string token, string numRomano);
}

public class ConversorRomanoArabe: MarshalByRefObject, IConversorRomanoArabe
{ ... }
```

LISTADO 2 Fichero de configuración web.config

```
<configuration>
    <system.runtime.remoting>
        <application>
            <service>
                <wellknown mode="Singleton"
type="ConversorRomanoArabe.ConversorRomanoArabe, ConversorRomanoArabeServer"
                    objectUri="ConversorRomanoArabe.rem"/>
            </service>
        </application>
    </system.runtime.remoting>
</configuration>
```

con el contenido mostrado en el listado 2. Este fichero indica a IIS que debe activar en modo Singleton un servidor .NET Remoting de tipo "ConversorRomanoArabe.ConversorRomanoArabe", implementado en el ensamblado "CoversorRomanoArabeServer.dll".

Para acabar de desplegar nuestro servicio web desarrollado con .NET Remoting, tan sólo deberemos compilar el fichero "ConversorRomanoArabeServer.cs", generando una dll que colocaremos en el subdirectorio "bin" del directorio donde está colocado el "web.config", ya que es donde lo busca IIS. Es decir, asumiendo que el fichero .cs está en el mismo directorio que el "web.config", lo compilaremos con:

```
csc /t:library /out:bin\ ConversorRomanoArabeServer.dll ConversorRomanoArabeServer.cs
```

Parte cliente del servicio web con .NET Remoting

Para generar un cliente capaz de consumir este servicio web procederemos de la manera ordinaria seguida al crear un cliente de un servicio web implementado con ASP.NET.

Recordemos que, por definición, en los servicios web XML, los clientes y servidores se comunican por red intercambiando mensajes en formato SOAP. Estos mensajes encapsulan los parámetros de entrada y salida de los métodos remotos como contenido XML. Para simplificar la construcción de clientes de servicios web, se suele generar una clase proxy que mapea los parámetros a elementos XML y luego envía los mensajes SOAP a través de la red. Mientras exista una descripción de un servicio en formato WSDL (Web Services Description Language), se podrá generar una clase proxy con la herramienta de .NET "wsdl.exe". Para nuestro servicio web de conversión de números romanos a árabes, y viceversa, el comando a utilizar será:

```
wsdl http://localhost/ConversorRomanoArabe/ConversorRomanoArabe.rem?wsdl
```

Para crear un cliente que consuma este servicio web simplemente necesitamos crear una clase que se sirve del proxy como intermediario para efectuar las invocaciones remotas. El listado 3 muestra cómo se instancia la clase "ConversorRomanoArabeService", resultante de aplicar el generador de proxies (wsdl) a la url anterior. Tras crear la instancia de "ConversorRomanoArabeService" efectuamos login para obtener un token de autorización que es pasado en todas las siguientes peticiones. Por ejemplo:

```
conversor.ArabeARomano(token, 1295)
```

LISTADO 3 Fichero RemotingHTTPWSDL/ConversorRomanoArabeClient.cs

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;

class ConversorRomanoArabeClient
{
    static void Main()
    {
        ConversorRomanoArabeService conversor = new
ConversorRomanoArabeService();
        string token;
        Usuario usuario = new Usuario();
        usuario.nombreUsuario = "solop";
        usuario.contraseña = "solop";
        if (conversor.Login(usuario, out token))
        {
            string numRomano = conversor.ArabeARomano(token, 1295);
            Console.WriteLine("El número árabe 1295 convertido a romano es:
                                                                " + numRomano);
            int numArabe = conversor.RomanoAArabe(token, numRomano);
            Console.WriteLine("El número romano " + numRomano +
                                                " convertido a árabe es: " + numArabe);
            conversor.Logoff(usuario, token);
        }
    }
}
```

MSMQ

Microsoft Message Queuing (MSMQ) es una herramienta MOM (Message Oriented Middleware) que ofrece un mecanismo alternativo a .NET Remoting, menos acoplado, para la comunicación con aplicaciones remotas. MSMQ se diferencia principalmente de .NET Remoting en que garantiza el envío de mensajes. Los mensajes que no pueden ser enviados se guardan en una cola y permanecen allí hasta que el destinatario vuelve a estar activo. Es una solución destinada a sistemas que requieren mayor robustez que el modelo de mensajería petición/respuesta de RPC. Además, permite a varios mensajes relacionados ser combinados en una transacción única para asegurar así que sean enviados en orden, y solamente una vez, y sean recibidos exitosamente desde sus colas de mensajes por las aplicaciones remotas. Si ocurre un error, toda la transacción puede ser deshecha. MSMQ puede ser utilizado para implementar soluciones tanto para escenarios de comunicación síncronos como asíncronos. La infraestructura MSMQ funciona simplemente como un broker de mensajes, es decir, permite la lectura y escritura de mensajes, y se

encarga del correcto enrutamiento y envío de los mismos.

Así como .NET Remoting, o el paradigma de comunicación RPC, está orientado a los sistemas que se encuentran altamente acoplados, MSMQ es una solución para sistemas poco o nada acoplados (loosely-coupled). Como los mensajes de e-mail que recibimos en nuestra bandeja de entrada, en MSMQ los mensajes pueden ser enviados a sistemas muy diferentes (diferente plataforma) que no necesitan ni siquiera estar conectados directamente unos con otros, o incluso saber de su existencia, y además realizarlo de manera atemporal, es decir, el emisor y receptor del mensaje no tienen por qué encontrarse simultáneamente activos.

Con MSMQ, se pueden escribir aplicaciones que no requieren respuesta inmediata ni de clientes ni de servidores, que otorgan la flexibilidad de manejar pausas entre procesos de negocio. A través de este mecanismo, se pueden desarrollar aplicaciones más robustas y escalables, que pueden intercambiar mensajes incluso a través de redes poco fiables como las inalámbricas.

Algunos ejemplos de sistemas donde MSMQ es utilizado son:

- Servicios financieros que garanticen la integridad de datos intercambiados, por ejemplo en comercio electrónico.
- Dispositivos y aplicaciones empotradas.
- Sistemas de workflow en los cuales diferentes agentes software realizan distintas funciones de manera desacoplada, sin depender unos de otros.

Colas de mensajes

Las colas de mensajes son el componente fundamental de MSMQ, y el concepto principal en el que se basa el middleware de tipo MOM. Las ventajas del enfoque MOM sobre la comunicación RPC tradicional, tal es el caso de .NET Remoting, son:

- **Robustez:** Los mensajes no son afectados por fallos de los componentes ya que son guardados en colas y permanecen allí hasta ser correctamente procesados.
- **Prioridad de mensajes:** Los mensajes más urgentes o importantes pueden ser recibidos antes que otros mensajes, garantizando un tiempo de respuesta más adecuado para aplicaciones críticas.
- **Capacidad de trabajo off-line:** Los mensajes son enviados a colas y permanecen allí hasta que son finalmente entregados. Un emisor de un mensaje puede proseguir su trabajo una vez lo ha entregado a la cola, independientemente de la velocidad del proceso receptor o incluso si está o no activo.
- **Mensajería transaccional:** Se pueden agrupar varios mensajes en una sola transacción, garantizando que se entregan en orden, se envían una sola vez, y son recuperados exitosamente de su cola de destino.
- **Seguridad:** Los mensajes enviados pueden ser encriptados y recibidos sólo por usuarios autorizados.

Podemos utilizar los siguientes tipos de colas:

- Una cola pública es visible a través de la red y puede ser accedida potencialmente por todas las máquinas en Internet.
- Una cola privada es visible sólo en la máquina local y accesible por aplicaciones que conocen el camino o nombre de la cola.
- Una cola del sistema es aquella utilizada por el sistema operativo, pueden ser journal queues que guardan copias de mensajes, dead letter queues que guardan copias de mensajes no enviados o expirados o colas de informes.

Instalando Message Queuing

Para poder utilizar MSMQ es necesario tenerlo instalado tanto en las máquinas emisoras de mensajes como en las receptoras. Este componente puede ser instalado utilizando el CD de instalación de Windows 2000/XP Professional. Para ello abriremos el "Panel De Control" y haremos clic sobre "Añadir/Eliminar Componentes Windows". Nos aseguraremos de tener seleccionada la casilla correspondiente a "Message Queuing Services", como se muestra en la figura 1. En caso de que esta casilla no hubiera sido seleccionada con anterioridad se procederá a su instalación con la ayuda del CD de instalación de Windows.

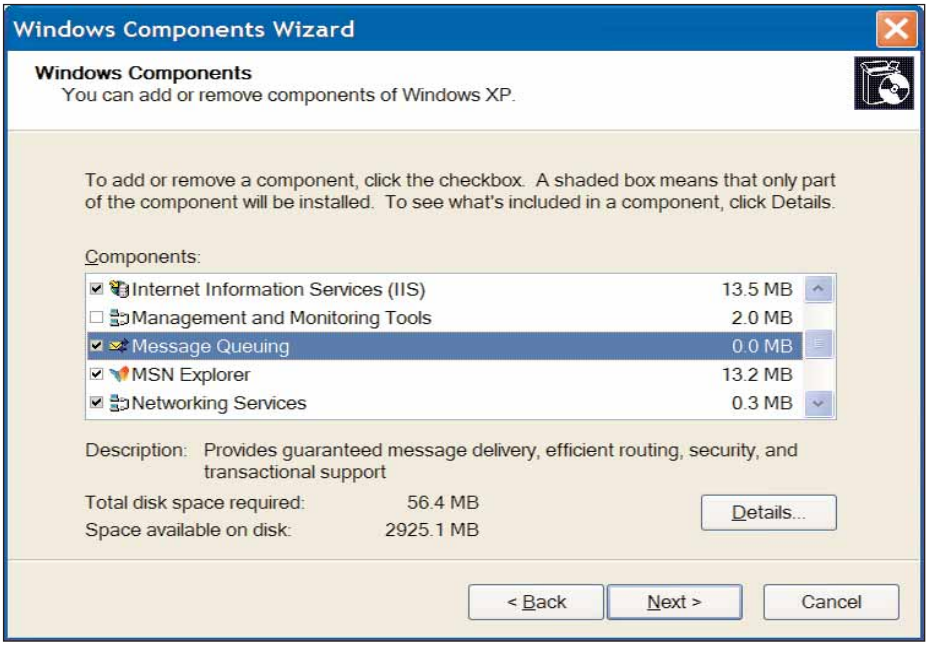

Figura 1. Pantalla de selección de componentes Windows.

MSMQ en .NET

El espacio de nombres "System.Messaging" define una interfaz para utilizar desde .NET MSMQ. "System.Messaging" provee clases que permiten conectarse, monitorizar y administrar colas de mensajes en una red, así como enviar, recibir o hacer un peek de los mensajes de las mismas.

La clase "System.Messaging.MessageQueue" incluye los siguientes métodos para leer y escribir mensajes a una cola:

- El método "Send" escribe un mensaje en una cola. Normalmente recibe mensajes de tipo "System.Messaging.Message".
- Los métodos "Receive", "ReceiveById" y "ReceiveByCorrelationId" proveen funcionalidad para leer mensajes de una cola. Como con los métodos de "Send" se facilitan varias overloads de los mismos que soportan procesamiento transaccional de las colas. Estos métodos bloquearán al proceso emisor hasta que se reciba un mensaje o expire un time-out indicado opcionalmente.
- El método "Peek" es similar a "Receive", pero hace que el mensaje no sea eliminado de la cola cuando es leído.

- Los métodos "BeginPeek", "EndPeek", "BeginReceive", y "EndReceive" proveen mecanismos para leer mensajes de manera asíncrona de la cola, sin bloquearse el hilo que invoca los métodos.

Otros métodos de "MessageQueue" permiten recuperar las listas de colas o determinar si una cola existe, así como crear o eliminar una cola:

- El método "GetPrivateQueuesByMachine" permite recuperar las colas privadas de un ordenador.
- Los métodos "GetPublicQueuesBy Category", "GetPublicQueuesByLabel", y "Get PublicQueuesByMachine" proveen mecanismos para recuperar colas públicas a partir de unos criterios.
- Para crear una cola usaremos el método "Create" y para eliminarla "Delete".

La clase "System.Messaging.Message" provee control total sobre la información enviada a una cola. Aparte del cuerpo del mensaje, algunas de las propiedades incluidas en un mensaje son: selección del formateador, mecanismo de reconocimiento preferido, identificación, autenticación y encriptación del contenido, marca de tiempo o información transaccional.

Una cola de mensajes (MessageQueue) está asociada con tres formateadores que permiten serializar y deserializar los mensajes enviados y recibidos por las colas:

- El "XmlMessageFormatter" provee un mecanismo de comunicación poco acoplado, permitiendo versiones diferentes de tipos serializados en la parte cliente y en la servidora.
- El "ActiveXMessageFormatter" es compatible con el control MSMQ COM.
- El "BinaryMessageFormatter" provee una alternativa más rápida que el "XmlMessageFormatter", pero sin los beneficios de bajo acoplamiento.

Por ejemplo, para crear una cola privada haríamos lo siguiente:

```
using System.Messaging;
…
string qPath=@".\Private$\PruebaQueue";
if (!MessageQueue.Exists(qPath))
    MessageQueue.Create(qPath); // crea
    la cola
MessageQueue q = new MessageQueue
(qPath); // instancia un objeto cola
```

La sintaxis para el path de una cola es "nombreMáquina\nombreCola" para las colas públicas y "nombreMáquina\Private$\nombreCola" para las privadas. Se puede utilizar el "." para representar la máquina local y también se puede modificar la propiedad "Path" de una cola en tiempo de ejecución. Por ejemplo:

```
q.Path = @".\Private$\PruebaQueue"
```

Para poder enviar y recibir nuestros propios tipos de datos, asignamos los nombres de los tipos a enviar a la propiedad "Formatter" de un MessageQueue. Por ejemplo:

```
q.Formatter = new XmlMessageFormatter
(new string[] {"Usuario"});
...
// recibir un objeto Usuario
Usuario user = (Usuario)q.Receive().Body;
...
// enviar un objeto Usuario
q.Send(user);
```

ConversorRomanoArabe implementado con MSMQ

Para demostrar la funcionalidad de MSMQ hemos adaptado nuestra implementación del conversor de números romanos a árabes y viceversa. Para ello hemos dividido nuestro código en cuatro ficheros:

- **MSMQRomanos\ConversorRomanoArabe.cs:** Implementa la librería compartida "ConversorRomanoArabe.dll" que encapsula el proceso de conversión entre formatos numéricos y ofrece tres métodos públicos a los usuarios de esta librería (mostrados a continuación): el constructor de la clase y los métodos "ArabeARomano" y "RomanoAArabe". El código de esta clase se puede encontrar en el fichero "MSMQRomanos\Conversor RomanoArabe.cs" del CD-ROM número 125 que acompaña a la revista:

```
public ConversorRomanoArabe();
public string ArabeARomano(int num);
public int RomanoAArabe
(string numRomano);
```

- **MSMQRomanos\ConvertQueue.cs:** En este fichero se definen dos clases útiles tanto para los procesos clientes, aquellos que solicitan una conversión, como los procesos servidores, aquellos que las efectúan:
  - La clase "ConvertQueue" es un wrapper de un objeto "System.Messaging. MessageQueue". Básicamente, crea una cola privada a la que se asocia un formateador XML capaz de serializar objetos de tipo "ConvertMsg". Además de un constructor al que se pasa como parámetro el nombre de la cola a crear, define los métodos "Send" y "Receive"

LISTADO 4 — Clase ConvertQueue

```
public class ConvertQueue
{
    private string qPath;
    private MessageQueue q;

    public ConvertQueue(string qPath)
    {
        this.qPath = @".\Private$\" + qPath;
        // crear e instanciar la cola ...
        if (!MessageQueue.Exists(this.qPath)) MessageQueue.Create(this.qPath);
        this.q = new MessageQueue(this.qPath);
        // formatear la cola para que guarde mensajes de tipo ConvertMsg
        // del ensamblado ConvertQueue.dll
        this.q.Formatter = new XmlMessageFormatter(
            new string[]{"ConversorRomanoArabe.ConvertMsg, ConvertQueue"});
    }

    public void Send(ConvertMsg msg)
    {
        this.q.Send(msg); // enviar un objeto ConvertMsg
    }

    public ConvertMsg Receive()
    {
        return (ConvertMsg)this.q.Receive().Body; // recibir un objeto ConvertMsg
    }

    public void Kill()
    {
        this.q.Purge(); // eliminar todos los mensajes encolados
        MessageQueue.Delete(this.qPath); // eliminar la cola
    }
}
```

LISTADO 5 Clase ConvertMsg

```
public class ConvertMsg
{
   public string format;
   public string number;
   public string QID;
}
```

LISTADO 6 Clase ConversorRomanoArabeMSMQServer

```
using System;
using System.Messaging;

namespace ConversorRomanoArabe
{
   public class ConversorRomanoArabeMSMQServer
   {
      public static void Main()
      {
         ConversorRomanoArabe conversor = new ConversorRomanoArabe();
         ConvertQueue queue = new ConvertQueue("conversor");
         Console.WriteLine("Processing Conversiones de Romano a Árabe y viceversa");
         while(true)
         {
            ConvertMsg cMsg = queue.Receive();
            switch (cMsg.format.ToLower().Trim())
            {
               case ConversorRomanoArabe.ARAB:
                  cMsg.number = conversor.ArabeARomano(Int32.Parse(cMsg.number));
                  cMsg.format = ConversorRomanoArabe.ROMAN;
                  break;
               case ConversorRomanoArabe.ROMAN:
                  cMsg.number = conversor.RomanoAArabe(cMsg.number).ToString();
                  cMsg.format = ConversorRomanoArabe.ARAB;
                  break;
               default:
                  cMsg.number = "ERROR: el parámetro format debe tener el valor
                  'roman' o 'arab'";
                  cMsg.format = ConversorRomanoArabe.ERROR;
                  break;
            }
            ConvertQueue outQ = new ConvertQueue(cMsg.QID);
            cMsg.QID = null;
            outQ.Send(cMsg);
         }
      }
   }
}
```

para enviar y recibir mensajes de tipo "ConvertMsg" y el método Kill para eliminar la cola. El listado 4 muestra la definición de esta clase.

- La clase "ConvertMsg" representa los mensajes intercambiados entre solicitantes y conversores de números romanos a árabes y viceversa. Los datos que encapsula son: (1) El formato del número a convertir ("format", que será "roman" o "arab"), (2) El número al que se aplicará la conversión ("number") y (3) El identificador de la cola privada al que enviar la respuesta ("QID"). El listado 5 muestra la definición de esta clase.

- **MSMQRomanos\ConversorRomanoArabeMSMQServer.cs:** Esta clase utiliza la librería anteriormente definida para crear un objeto de tipo "Conversor RomanoArabe" al cual delegar la traducción de las solicitudes recibidas de los clientes en forma de mensajes de tipo "ConvertMsg". Las respuestas enviadas a estos clientes son también en este mismo formato. El listado 6 muestra la definición de esta clase.

- **MSMQRomanos\ConversorRomanoArabeMSMQClient.cs:** Define una clase cliente que en su bloque Main acepta dos parámetros como entrada: el formato del número a convertir y el número a transformar. Con esos dos parámetros genera un objeto de tipo "ConvertMsg" y crea una cola donde recibir la respuesta con nombre en el formato:

```
"convert_client_" + new System.
Random ().Next(1, 1000000)
```

A continuación, se queda bloqueado con una llamada a "Receive" hasta que se recibe la respuesta del servidor. El listado 7 muestra la definición de esta clase.

Para compilar y ejecutar tanto la parte servidora como cliente de esta versión del conversor realizada mediante MSMQ, seguiremos las instrucciones indicadas en el fichero "MSMQRomanos\LEEME.txt" incluido en el CD-ROM número 125. A continuación, si en nuestro sistema vamos a "Panel De Control/Herramientas Administrativas/Gestión de la Máquina/Servicios y Aplicaciones/Message Queuing" se podrá observar que en el listado de colas privadas aparecerá una nueva de nombre "conversor" tal como se muestra en la figura 2. Para probar la gran robustez otorga-

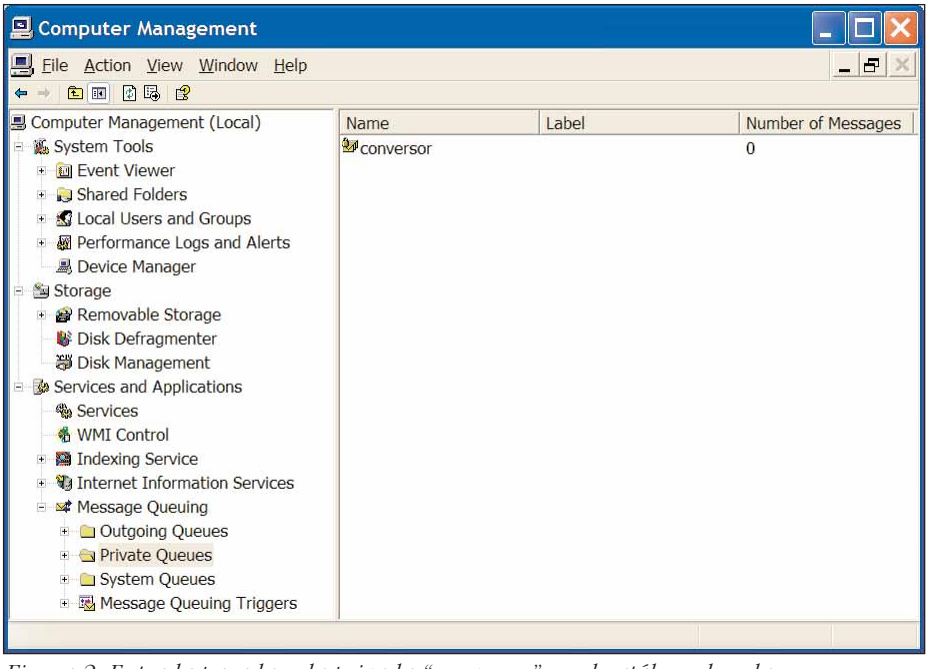

Figura 2. Entrada para la cola privada "conversor" en el catálogo de colas.

LISTADO 7 Clase ConversorRomanoArabeMSMQClient

```
using System;
using System.Messaging;

namespace ConversorRomanoArabe
{
   class ConversorRomanoArabeMSMQClient
   {
      public static void Main(string[] argv)
      {
         if (argv.Length != 2)
         {
            Console.WriteLine("Usage: ConversorRomanoArabeMSMQClient <formato-
            de-número-a-convertir> número\n\tdonde formato-de-número-a-
            convertir es [roman|arab]");
            return;
         }
         ConvertMsg convertRequest = new ConvertMsg();
         convertRequest.format = argv[0];
         convertRequest.number = argv[1];
         convertRequest.QID = "convert_client_" + new System.Random().Next(1, 1000000);

         ConvertQueue inQ = new ConvertQueue(convertRequest.QID);
         ConvertQueue outQ = new ConvertQueue("conversor");
         outQ.Send(convertRequest);

         ConvertMsg convertResponse = inQ.Receive();
         Console.WriteLine("El resultado de convertir el número {0} es {1}",
         argv[1], convertResponse.number);
         inQ.Kill();
      }
   }
}
```

da a nuestro servicio de conversión por el uso de MSMQ se recomienda arrancar varios clientes antes de arrancar uno o varios servidores y comprobar como todos los mensajes llegan eventualmente a alguno de los servidores y la respuesta es recibida por los clientes.

Enterprise Services

Los Enterprise Services son un mecanismo que permite a las clases de .NET acceder a los servicios COM+. Una aplicación .NET accede a los servicios COM+ a través del espacio de nombres "System.EnterpriseServices". De esa manera, los objetos .NET se benefician de la infraestructura COM+ para así adecuarse a las demandas de robustez y escalabilidad de las aplicaciones de empresa. COM+ es una tecnología que provee una arquitectura de servicios para la programación basada en componentes de aplicaciones de empresa. Para obtener una documentación más detallada sobre .NET Enterprise Services se recomienda consultar la url http://support.microsoft.com/default.aspx?scid=kb;en-us;308672#2.

Algunos ejemplos de los servicios COM+ disponibles a las aplicaciones .NET a través de "System.EnterpriseServices" son:

- **Transacciones:** Aplica características de procesamiento de transacciones a una aplicación tanto de modo declarativo como programático.
- **Activación Just-in-Time (JIT):** Activa un objeto cuando se invoca un método y lo desactiva cuando la llamada es devuelta. Ayuda a tu servidor a utilizar los recursos del servidor de manera más eficiente, particularmente cuando se escala una aplicación para realizar un alto volumen de transacciones.
- **Pools de Objetos:** Configura un componente para que mantenga instancias de si mismo activas en un repositorio (pool), para así ser reutilizadas cuando un cliente realice una petición a ese componente.
- **Eventos:** El servicio de eventos de COM+ permite la colaboración de componentes desacoplados mediante la gestión de eventos de diferentes emisores. Los suscriptores de esos eventos pueden a través de este componente seleccionar los eventos que desean recibir.
- **Sincronización:** Garantiza la exclusión mutua en el acceso a un componente.
- **Gestión de propiedades compartidas:** Se utiliza para mantener estado compartido entre múltiples objetos dentro de un proceso servidor.
- **Seguridad:** Aplica permisos de seguridad basados en roles tanto de manera declarativa como programática.

El espacio de nombres "System.EnterpriseServices" provee acceso a atributos propietarios y clases para acceder a estos servicios desde código manejado en .NET. Todo servicio de componentes es una clase escrita en un lenguaje CLS (Common Language Specification) que deriva de la clase "System.EnterpriseServices.ServicedComponent". Para usar estos servicios es necesario aplicar atributos relacionados a servicios y crear instancias de clases que usan esos servicios. El catálogo COM+ mantiene información sobre la configuración aplicada a la implementación de una clase. COM+ crea una capa de contexto de servicio en base a estos atributos.

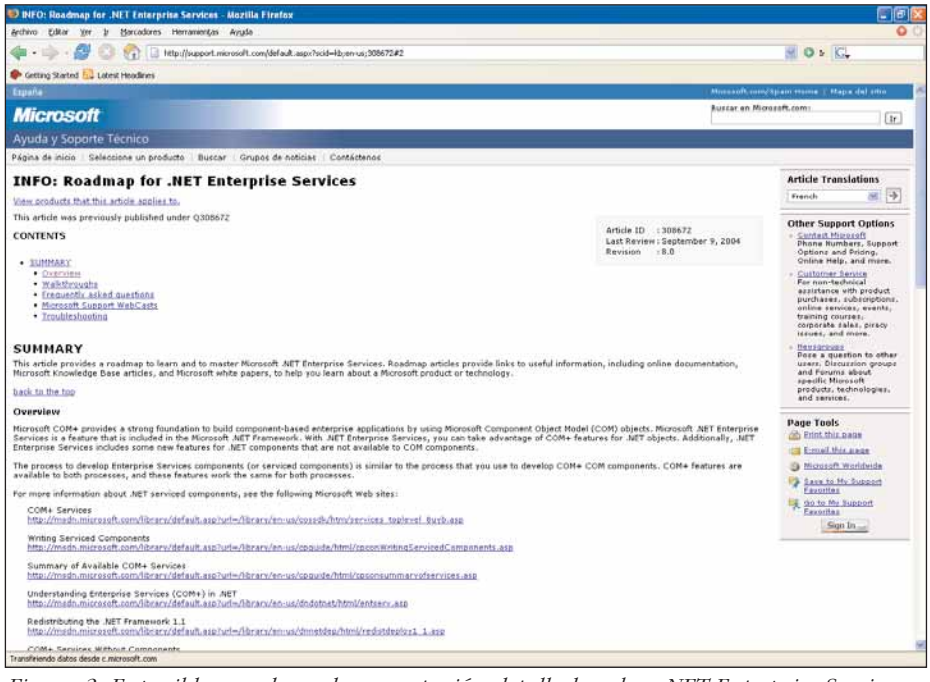

Figura 3. Es posible acceder a documentación detallada sobre .NET Enterprise Services en http://support.microsoft.com/default.aspx?scid=kb;en-us;308672#2.

Creando un ConversorRomanoArabe escalable

Para ilustrar el uso de Enterprise Services vamos a crear una versión de nuestro archiconocido conversor de números romanos a árabes, enriquecido con el mecanismo Object Pooling de COM+. De ese modo, nuestro conversor será un componente altamente escalable que pueda resolver las solicitudes concurrentes de conversión provenientes de múltiples clientes.

El servicio COM+ Object Pooling permite reducir la sobrecarga de crear objetos desde cero. Cuando un objeto se activa se recupera del pool. Cuando se desactiva, se encola en el pool para esperar a la siguiente petición. Para configurar Object Pooling se utiliza el atributo "ObjectPoolingAttribute" en una clase derivada de "System.EnterpriseServices.ServicedComponent".

En el listado 8 mostramos el esqueleto de la implementación del conversor romano a árabe aumentado con Object Pooling. La versión completa de este código esta disponible en el CD-ROM número 125, en el fichero "RomanosObjectPooling/ConversorRomanoArabePoolingService.cs". Obsérvese el uso de atributos .NET, es decir, la configuración declarativa empleada para: (1) Dar un nombre a la aplicación con el que registrarla en el catálogo COM+ (véase la figura 4), (2) Asociar la aplicación a un nombre fuerte, solamente las clases strong-named pueden ser registradas en el catálogo COM+, y (3) Para declarar que el servicio COM+ Object Pooling será utilizado por esta aplicación. La línea:

LISTADO 8 — Clase ConversorRomanoArabePoolingService

```
using System;
using System.EnterpriseServices;
using System.Reflection;
using System.Collections;

// El nombre de la aplicación COM+ creada
[assembly: ApplicationName("ConversorRomanoArabePoolingService")]
// El strong-name del ensamblado
[assembly: AssemblyKeyFileAttribute("ConversorRomanoArabePoolingService.snk")]

// Configuración declarativa del Object Pooling
[ObjectPooling(Enabled=true, MinPoolSize=2, MaxPoolSize=5,
CreationTimeout=20000)]
public class ConversorRomanoArabePoolingService : ServicedComponent
{
   // ...
   public ConversorRomanoArabePoolingService()
   {
        // ...
   }
   protected override void Activate()
   {
      Console.WriteLine("Instancia solicitada del pool");
   }
   protected override void Deactivate()
   {
      Console.WriteLine("Instancia devuelta a pool");
   }
   protected override bool CanBePooled()
   {
      Console.WriteLine("Puedo desactivarme");
      return true;
   }
   public string ArabeARomano(int num)
   {
      // ...
   }
   public int RomanoAArabe(string numRomano)
   {
      // ...
   }
}
```

```
[ObjectPooling(Enabled=true,
MinPoolSize=2, MaxPoolSize=5,
CreationTimeout=20000)]
```

Indica que en el pool habrá como mínimo dos instancias y como máximo cinco de "ConversorRomanoArabePoolingService", además indica el máximo tiempo (2 segundos) que se deberá esperar a que una instancia de un objeto sea obtenido desde el pool antes de lanzar una excepción. Finalmente, apreciar que la clase "ConversorRomanoArabePoolingService" definida hereda de "ServicedComponent".

El proceso de compilación de una aplicación que utiliza Enterprise Services es más complejo que el de una aplicación .NET estándar. A continuación, mostramos la secuencia de comandos que serían necesarios para la compilación satisfactoria del "Conversor RomanoArabePoolingService":

1- **sn -k ConversorRomanoArabePoolingService.snk:** Generamos un nombre seguro para el ensamblado correspondiente al conversor romano árabe con Object Pooling.
2- **csc /t:library /r:System.EnterpriseServices.dll ConversorRomanoArabePoolingService.cs:** Generamos el ensamblado del conversor en forma de una dll.
3- **gacutil.exe -i ConversorRomanoArabePoolingService.dll:** Instalamos el ensam-

LISTADO 9 — Clase cliente de ConversorRomanoArabePoolingService

```
using System;
using System.EnterpriseServices;
using System.Threading;

public class App
{
   public static void Main(string[] args)
   {
      for (int i=0; i<1000;i++)
      {
         ConversorRomanoArabePoolingService conversor =
            new ConversorRomanoArabePoolingService();
         string romano1234 = conversor.ArabeARomano(1234);
         Console.WriteLine("El número {0} en romano es {1}", 1234, romano1234);
         /* Devuelve el objeto al pool y permite que sea reutilizado. Si no
          * se indica explícitamente el recolector de basura todavía se
          * encarga de devolverlo al pool, aunque podría afectar el rendimiento
          * global de la aplicación */
         ServicedComponent.DisposeObject (conversor);
         Thread.Sleep(1000);
      }
   }
}
```

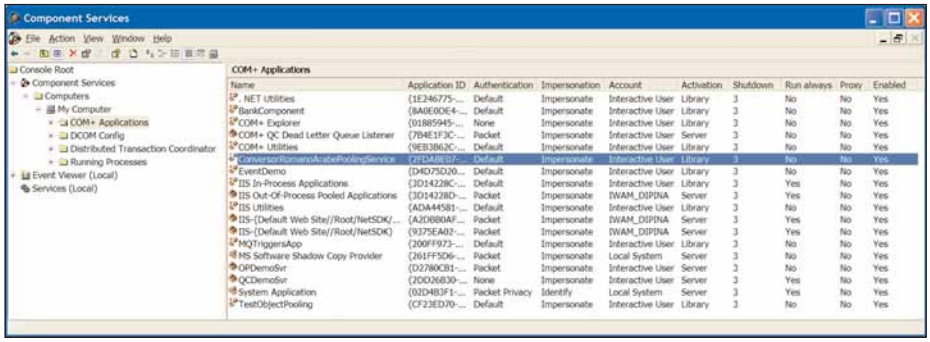

Figura 4. Entrada "ConversorRomanoArabePoolingService" en el catálogo COM+.

blado con nombre fuerte en el Global Assembly Cache, que es un repositorio definido en .NET para ensamblados compartidos por aplicaciones.

4- **regsvcs.exe ConversorRomanoArabePoolingService.dll:** Registramos de manera explícita el ensamblado con el catalogo COM+. El resultado de tal registro del servicio se refleja en la figura 4.

Finalmente, en el listado 9 mostramos el código de una aplicación cliente que utiliza el "ConversorRomanoArabePoolingService". Al adoptar el servicio COM+ Object Pooling, el hecho de incluir en el código del cliente la sentencia:

```
ConversorRomanoArabePoolingService
conversor = new ConversorRomanoArabe
PoolingService();
```

No significa que se cree una nueva instancia de este objeto por cada llamada al constructor; simplemente se reutilizarán continuamente las instancias disponibles en el pool de objetos. Para compilar este código escribiríamos la sentencia:

```
csc /r:ConversorRomanoArabePooling
Service.dll ConversorRomanoArabe
PoolingClient.cs
```

Indigo

No sería justo acabar esta serie sobre programación distribuida en .NET sin mencionar Indigo, o lo que es lo mismo: el futuro de la programación distribuida en .NET. Indigo combina y extiende la funcionalidad de las tecnologías de programación distribuida existentes en .NET como ASMX, .NET Remoting, .NET Enterprise Services, Web Services Enhancements, y System. Messaging con el objetivo de ofrecer una infraestructura simple y altamente productiva de aplicaciones distribuidas. Indigo es una infraestructura de comunicación construida alrededor de la arquitectura de servicios web. El soporte de servicios web avanzados (los llamados WS-*) en Indigo garantiza mensajería segura, robusta y transaccional, además de interoperabilidad. Indigo estará disponible con Windows Longhorn, aunque muy pronto podremos descargar versiones de evaluación para Windows XP y Windows Server 2003.

En algunos foros de Internet se ha extendido el rumor que la aparición de Indigo acabará con .NET Remoting. Eso no es cierto en absoluto. Para una explicación más detallada remito al lector a http://blogs.msdn.com/richardt/archive/2004/03/05/84771.aspx.

Si se desea información más detallada sobre Indigo o cómo acceder a versiones pre-beta de este sistema, se recomienda consultar la url http://msdn.microsoft.com/Longhorn/understanding/pillars/Indigo/default.aspx.

Conclusiones

En esta serie de tres entregas hemos realizado un recorrido sobre las principales herramientas provistas por .NET para la programación de sistemas distribuidos: sockets, multithreading, XML, .NET Remoting, MSMQ y Enterprise Services. Según ha ido avanzando la serie también lo ha hecho la sofisticación de las herramientas estudiadas. Confío que tras revisar estos artículos el lector haya podido comprender el potencial de cada una de las herramientas expuestas y sea capaz de identificar las situaciones en las que la aplicación de cada una de ellas (o su combinación) sea más adecuada. De cara al futuro se recomienda mantenerse atento a los progresos de Indigo, cuya versión definitiva es posible aparezca en menos de un año.