Desarrollo de Aplicaciones Distribuidas

AUTORES:
Alejandro Calderón Mateos
Javier García Blas
David Expósito Singh
Laura Prada Camacho

Departamento de Informática
Universidad Carlos III de Madrid
Julio de 2012

DESARROLLO DE APLICACIONES DISTRIBUIDAS CON .NET:

ESTRUCTURA .NET

.NET remoting

- Tipos de serialización de objetos:
  - Objetos no serializables.
  - Serializables por valor: `Serializable`.
  - Serializables por referencia. `MarshalByRefObjects`.

- .NET Remoting emplea *proxies* transparentes.

- Tipos de canales:
  - TcpChannel.
  - HttpChannel.

.NET remoting

- Serialización de objetos.
  - Dos niveles de serialización: LOW y FULL.

```
<configuration>
<system.runtime.remoting>
<application>
<service>
<wellknown
type="ServiceType, common"
objectUri="ServiceType.soap"
mode="Singleton"
/>
</service>
<channels>
<channel ref="http">
<serverProviders>
<provider ref="wsdl" />
<formatter ref="soap" typeFilterLevel="Full" />
<formatter ref="binary" typeFilterLevel="Full" />
</serverProviders>
</channel>
</channels>
</application>
</configuration>
```

.NET remoting

- Proceso de invocación remota:
  - Objeto cliente: realiza la invocación.
  - Objeto *proxy* transparente: crea objeto que implementa la interfaz *IMessage*.
  - Objeto *proxy* real: la implementación puede ser sobrecargada.
  - *Sink objects:* procesado y formateado.
  - *Channel object*: Codifica y envía el objeto.
  - *Stack Builder:* desempaqueta el *IMessage* e invoca el método real.

.NET remoting

- *Delegates*.
  - Equivalen a punteros a funciones en C.
  - Están orientados a objetos.
  - Son completamente *type safe*.

- Uso de *delegates*:
  - ```
    static OutputMessageDelegate OutputMessage = new
    OutputMessageDelegate(OutputToConsole);
    ```

- *Delegates* admiten sobrecarga.
  - ```
    OutputMessage += new OutputMessageDelegate(OutputToMessageBox)
    ```
  - ```
    OutputMessage -= new OutputMessageDelegate(OutputToMessageBox)
    ```
    ```

.NET remoting: delegates

```
using System;
using System.Windows.Forms;

namespace hello1
{
  class Class1
  {
    private static string HelloMessage = "Hello, world";
    private static bool outputConsole = true;

    [STAThread]
    static void Main(string[] args)
    {
      if ((args.Length > 0) && (args[0] == "M"))
        outputConsole = false;
      OutputMessage(HelloMessage);
    }
```

.NET remoting

```
static void OutputMessage(string msg)
    {
      if (outputConsole)
        OutputToConsole(msg);
      else
        OutputToMessageBox(msg);
    }

    static void OutputToConsole(string msg)
    {
      Console.WriteLine(HelloMessage);
    }

    static void OutputToMessageBox(string msg)
    {
      MessageBox.Show(HelloMessage);
    }
  }
}
```

.NET remoting

```
using System;
using System.Windows.Forms;
namespace hello2
{
 delegate void OutputMessageDelegate(string msg);
  class helloClass
  {
    private static string HelloMessage = "Hello, world";

      static OutputMessageDelegate OutputMessage  =  New OutputMessageDelegate(OutputToConsole);

    static void Main(string[] args)
    {
      if ((args.Length > 0) && (args[0] == "M"))
      {
      OutputMessage =           new OutputMessageDelegate(OutputToMessageBox);
      }
      OutputMessage(HelloMessage);
    }
     static void OutputToConsole(string msg)
    {
      Console.WriteLine(HelloMessage);
    }
    static void OutputToMessageBox(string msg)
    {
      MessageBox.Show(HelloMessage);
    }
  }
}
```

.NET remoting

- *Asynchronous Remoting*
  - Usa *asynchronous delegates* y un objeto *ManualResetEvent* para invocar el método remoto y esperar la respuesta.

.NET remoting

1. Cree una instancia de un objeto que pueda recibir una llamada remota a un método.
2. Englobar el método de instancia en un objeto **AsyncDelegate**.
3. Englobar el método remoto en otro delegado.
4. Llame al método **BeginInvoke** en el segundo delegado pasándole todos los argumentos, al objeto **AsyncDelegate** y a un objeto que contenga el estado (o una referencia nula; **Nothing** en Visual Basic).
5. Espere a que el objeto de servidor llame a su método de devolución de llamada.

.NET remoting

- Comparación: Java vs C# .NET. http://www.veridicus.com/tummy/programming/java_vs_csharp.asp

- Pros de Java:

  - Fully cross platform APIs including support for Graphics, Windowing and Multimedia. Java has even more APIs than Windows. LOTS AND LOTS of libraries available.

  - High density of OO design patterns put to practise compared to C#/.NET.

  - Currently supported by multiple commercial vendors (not one).

  - Although not recognised by any international standards committee, Java is an open standard governed by Sun and the JCP members (or which there are MANY). C# is an ECMA standard but without support from Microsoft, very few third parties have been willing to implement it. Sun also tends to be less secretive than Microsoft about their products.

.NET remoting

- Comparación: Java vs C# .NET. http://www.veridicus.com/tummy/programming/java_vs_csharp.asp

- Pros de C# .NET
  - The CLR is well designed and arguably better than the JVM.
  - The CLR supports execution of both managed and unmanaged code.
  - The **best** way to write Windows desktop and web applications.
  - The remoting framework is advanced and well designed.
  - Good support for taking advantage of the native OS.
  - Metadata attributes are a VERY,VERY powerful feature. Attributes allow classes to be much more expressive.
  - Supports ValueTypes (structs) as first class citizens. "Int" is considered an object.