Utilisation d'interfaces et de fichiers de configuration

Reprendre l'exemple de WKO vu en cours. Compléter le fichier d'interface par deux autres interfaces *IMultiplicateur* et *IFabrique*. *IFabrique* est une fabrique de divieurs.

1. Placer les interfaces dans un assemblage qui sera référencé par le client et le serveur
2. Ecrire un fichier de configuration du client
3. Utilisez, dans le programme du client, une table *dicoTypes* qui associe les services WKO configurés par le serveur à des interfaces configurées dans le client. Le contenu du fichier de configuration est récupéré l'aide de la méthode *RemotingConfiguration.GetRegisteredWellKnownClientTypes*().
4. Ecrire le fichier de configuration du serveur ainsi que le programme du serveur.

Question 1 :
Interface.cs

```
Compilation
csc.exe /out:Interface.dll  /target:library Interface.cs

namespace NommageInterface {
   public interface IAdditionneur {
      double Add(double d1, double d2);
   }
   public interface IMultiplicateur {
      double Mult(double d1, double d2);
   }
   public interface IDiviseur {
      double Div(double d1, double d2);
   }
   public interface IFabrique {
      IAdditionneur   FabriqueNouvelAdditionneur();
      IMultiplicateur FabriqueNouvelMultiplicateur();
      IDiviseur       FabriqueNouvelDiviseur();
   }
}
```

Question 2 :

Fichier de configuration du client:

```
<configuration>
   <system.runtime.remoting>
      <application name = "Client">
         <client>
            <wellknown type="NommageInterface.IAdditionneur,Interface"
                       url="http://localhost:65100/Service1.rem" />
            <wellknown type="NommageInterface.IMultiplicateur,Interface"
                       url="http://localhost:65100/Service2.rem" />
            <wellknown type="NommageInterface.IFabrique,Interface"
                       url="http://localhost:65100/Service3.rem" />
         </client>
      </application>
   </system.runtime.remoting>
</configuration>
```

Le programme du client :

```
Compilation:  csc.exe /out:Client.exe  /target:exe
Exemple_22_21_a_renommer_Client.cs /r:Interface.dll
```

```
//
*****************************************************************************
***************

using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Http;
using System.Runtime.Remoting.Activation;
using System.Collections;

using NommageInterface;

namespace NommageClient {
   class NotreActivateur {
      private static bool bInit;
      // Table d'association : interfaces/services distants WKO.
      private static IDictionary dicoTypes;

      public static Object GetObject(Type type) {
         if (!bInit)
            InitdicoTypes();
         WellKnownClientTypeEntry entry = (WellKnownClientTypeEntry)
                                                      dicoTypes[type];
         return Activator.GetObject(entry.ObjectType, entry.ObjectUrl);
      }

      private static void InitdicoTypes() {
         bInit = true;
         dicoTypes = new Hashtable();
         foreach ( WellKnownClientTypeEntry entry in
            RemotingConfiguration.GetRegisteredWellKnownClientTypes() )
               dicoTypes.Add(entry.ObjectType, entry);
      }
   }

   class Program {
      static void Main() {
         RemotingConfiguration.Configure("Client.config", false);

         IAdditionneur objA = (IAdditionneur)
            NotreActivateur.GetObject(typeof(IAdditionneur));
         double dA = objA.Add(3.0, 4.0);

         IMultiplicateur objM = (IMultiplicateur)
            NotreActivateur.GetObject(typeof(IMultiplicateur));
         double dM = objM.Mult(3.0, 4.0);

         // Utilisation du design pattern factory pour un objet CAO.
         IFabrique objFabrique = (IFabrique)
            NotreActivateur.GetObject(typeof(IFabrique));
         IDiviseur objD = objFabrique.FabriqueNouvelDiviseur();
         double dD = objD.Div(3.0, 4.0);
      }
   }
}
```

Le fichier de configuration du serveur :

```
<configuration>
   <system.runtime.remoting>
```

```
      <application name = "Serveur">
         <service>
            <wellknown type="NommageServer.CAdditionneur,Serveur"
                       mode ="Singleton" objectUri="Service1.rem" />
            <wellknown type="NommageServer.CMultiplicateur,Serveur"
                       mode ="SingleCall" objectUri="Service2.rem" />
            <wellknown type="NommageServer.CFabrique,Serveur"
                       mode ="SingleCall" objectUri="Service3.rem" />
         </service>
         <channels>
            <channel port="65100" ref ="http" />
         </channels>
      </application>
   </system.runtime.remoting>
</configuration>
```

Le programme du serveur :

```
Compilation:
csc.exe /out:Serveur.exe  /target:exe Serveur.cs /r:Interface.dll

using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Http;
using NommageInterface;

namespace NommageServer {
   public class CAdditionneur : MarshalByRefObject, IAdditionneur {
      public CAdditionneur() {
         Console.WriteLine("CAdditionneur ctor");
      }
      public double Add(double d1, double d2) {
         Console.WriteLine("CAdditionneur Add( {0} + {1} )", d1, d2);
         return d1 + d2;
      }
   }
   public class CMultiplicateur : MarshalByRefObject, IMultiplicateur {
      public CMultiplicateur() {
         Console.WriteLine("CMultiplicateur ctor");
      }
      public double Mult(double d1, double d2) {
         Console.WriteLine("CMultiplicateur Mult( {0} + {1} )", d1, d2);
         return d1 * d2;
      }
   }
   public class CDiviseur : MarshalByRefObject, IDiviseur {
      public CDiviseur() {
         Console.WriteLine("CDiviseur ctor");
      }
      public double Div(double d1, double d2) {
         Console.WriteLine("CDiviseur Div( {0} + {1} )", d1, d2);
         return d1 + d2;
      }
   }
   public class CFabrique : MarshalByRefObject, IFabrique {
      public IAdditionneur FabriqueNouvelAdditionneur() {
         return new CAdditionneur();
      }
      public IMultiplicateur FabriqueNouvelMultiplicateur () {
         return new CMultiplicateur();
      }
```

```
      public IDiviseur FabriqueNouvelDiviseur() {
         return new CDiviseur();
      }
   }
   class Program {
      static void Main() {
         HttpChannel canal = new HttpChannel(65100);
         ChannelServices.RegisterChannel(canal, false);

         RemotingConfiguration.RegisterWellKnownServiceType(
                typeof(CFabrique),
               "ServiceFabrique",
               WellKnownObjectMode.Singleton);
         Console.WriteLine(
            "Pressez une touche pour stopper le serveur.");
         Console.Read();
      }
   }
}
```

Ces programmes ont été repris du livre «Pratique de .NET2 et C#2, Patrick Smacchia (O'Reilly 2005)».