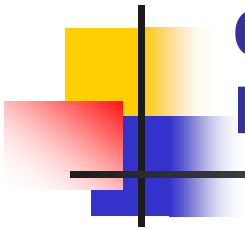
UE Systèmes à objets répartis

Objets/services répartis avec MICROSOFT .NET

Olivier Nedelec, Frank Singhoff

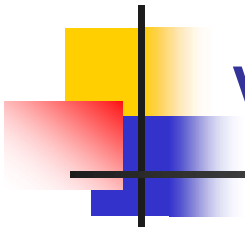


Objets/services répartis avec MICROSOFT .NET

1. **Introduction à C#**
2. .NET Remoting
3. Web Services .NET

Visual studio (1/2)

- Projet vs solutions vs références
  - Projet : produit un exécutable ou une bibliothèque
    - 1 répertoire
  - Solutions : ensemble d'exécutables ou de bibliothèques constituant une application
    - 1 sous répertoire par solution
  - Références : dépendances entre exécutables/bibliothèques



Visual studio (2/2)

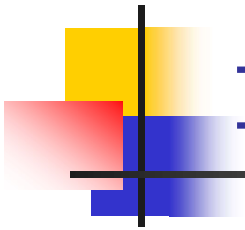
- Créer une solution et des projets
- Editer les sources, spécifier les références entre les projets
- Générer la solution
- Exécuter avec ou sans mise au point

Introduction C#

- Proche de Java (langage concurrent)

1) Structure

- Namespace : espace de visibilité des symboles/noms/déclarations
- Classe C#, méthode Main



Introduction C#

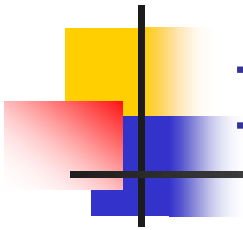
2) Entrées/sorties

- Offert par la classe System.Console
- Principales primitives utiles pour les TP:

```
public static ConsoleKeyInfo ReadKey();
public static string ReadLine();
public static void WriteLine(int, double,
        string, …);
```

Premier programme

```
using System;
namespace HelloWorld
{
    class Hello
    {
        static void Main()
        {
            System.Console.WriteLine("Hello World!");

            System.Console.WriteLine("Press any key to exit.");
            System.Console.ReadKey();
        }
    }
}
```



Introduction C#

3) Modèle objet

- Modèle similaire à Java
- Héritage simple
- Opérateur ':' => héritage, implantation d'interface

```
public class etudiant : Ietudiant, classe_mere
```

4) Type et passages d'arguments

- Objets passés par référence
- Types primitifs passés par copie
- « ref » : permet de passer tout argument par référence
- Types primitifs: int, double, string (+, .equals), …
- Tableau :

```
int [] a = new int [10];
```

Exemple avec les strings

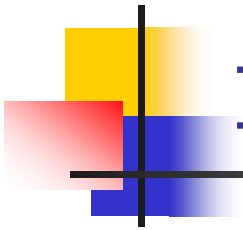
```
static void Main(string[] args)     {
        string s1 = "coucou ";
        string s2 = "";
        s2 = "coucou ";

        System.Console.WriteLine(s2);
        if (s2.Equals(s1))
            System.Console.WriteLine("equals");

        s2 = s2 + s2;
        System.Console.WriteLine(s2);
        if (s2.Equals(s1))
            System.Console.WriteLine("equals");

        System.Console.ReadKey();
        …
}
```

Passage arguments (1/2)

```
class A     {
        public int a = 100;
    }

class Program    {
        static void proc (A a1, int a2, ref int a3) {
            a1.a = 200;
            a2 = 200;
            a3 = 200;
     }
```

Passage arguments (2/2)

```
static void Main(string[] args)        {
    A a1 = new A();
    int a2 = 100;
    int a3 = 100;
    proc(a1, a2, ref a3);
    System.Console.WriteLine("a1 = " + a1.a);
    System.Console.WriteLine("a2 = " + a2);
    System.Console.WriteLine("a3 = " + a3);
    System.Console.ReadKey();
    …
}
```



Introduction C#

5) Exception

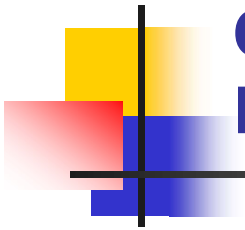
- Etendre la classe « Exception »
- try/catch
- new/throw
- Pas de déclaration des exceptions qu'une méthode peut lever

Exemple avec les exceptions (1/2)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace exceptions  {

    class MonEx : Exception   {
        public MonEx(string message) { }
    }
     class Program    {
        static double proc(int a)     {
                MonEx ex =
                    new MonEx("mon message " + a);
                throw ex;
        }
    …
}
```

Exemple avec les exceptions (2/2)

```
class Program    {

 static void Main(string[] args)
      {
          try          {
              proc(100);
              Console.ReadKey();
          }
          catch (MonEx ex)          {
              Console.WriteLine("catch de mon exception");
              Console.ReadKey();
          }
          catch (Exception ex)        {
              Console.WriteLine("\nMessage ---\n{0}", ex.Message);
              Console.ReadKey();
          }
}
```



Objets/services répartis avec MICROSOFT .NET

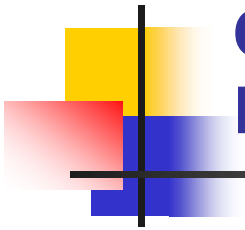
1. Introduction à C#
2. **.NET Remoting**
3. **Web Services .NET**

Objets/services répartis avec MICROSOFT .NET

- .NET Remoting:
  - TCP ou HTTP
  - Avec ou sans état : objets accessibles à distance et objets sérialisés
  - Souche, activation, …

- Web Services .NET :
  - Modèle producteur/consommateur
  - Les services web ne fonctionnent qu'à travers HTTP
  - Les services web fonctionnent en mode sans état : pas de conservation de l'état d'un objet d'une invocation à l'autre
    - Notion de session, architecture N-tiers

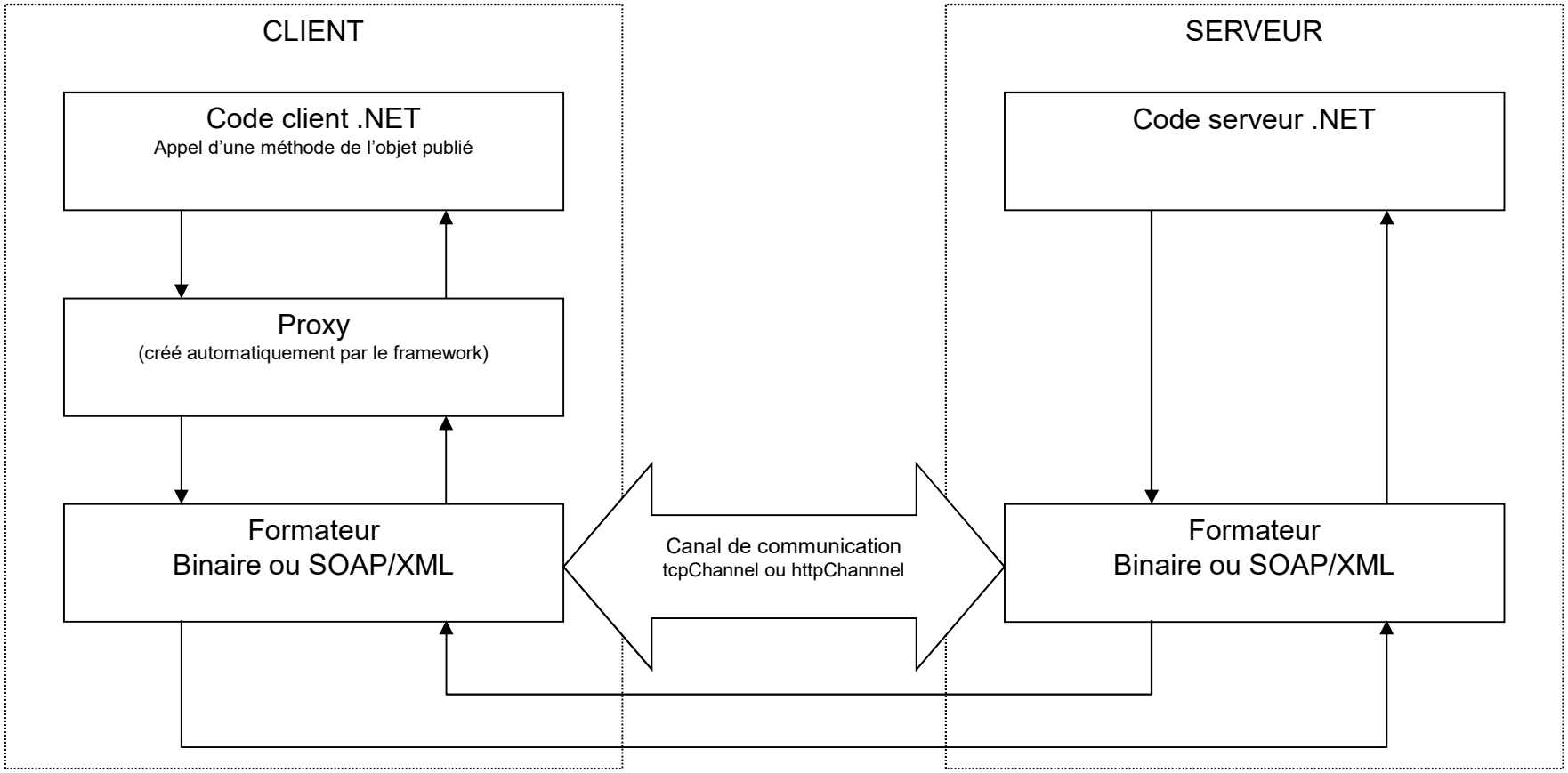
- Utilisation de protocoles similaires, basés sur une sérialisation XML



Objets/services répartis avec MICROSOFT .NET

1. Introduction à C#
2. **.NET Remoting**
3. Web Services .NET

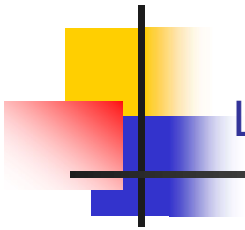
Objets/services répartis avec MICROSOFT .NET



La technologie .NET REMOTING

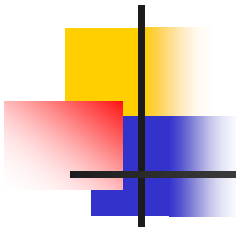
- Limitations
  - Pas de services de noms
  - Pas de transparence à la localisation
  - Obligation de déployer côté client une copie du code de l'interface du serveur pour la génération du proxy
    ➔ Mise à jour de l'objet difficile
  - Pas de mécanisme de sécurité/authentification

  ➔ Usage de .NET Remoting limité à un réseau local



La technologie .NET REMOTING

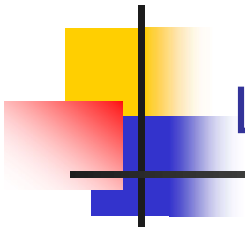
- Accès à un objet à distance :
  - Utilisation d'un objet MarshalByRefObject
- Notion de proxy/souche
  - Proxy = représentation locale d'un objet distant
    ➔ prend en charge la communication avec le serveur
    ➔ sérialisation/désérialisation des données
- Passage d'un objet par copie :
  - Utilisation d'un objet « Marshal By Value »: Serializable
- Canal de communication
  - httpChannel : échanges client/serveur par HTTP
    ➔ traverse les firewalls
  - tcpChannel : échanges TCP selon protocole spécifique
    ➔ protocole plus efficace que httpChannel
    ➔ à utiliser uniquement en réseau local



La technologie .NET REMOTING

- Types de connexion:
  - Singleton : l'objet est conservé pendant une certaine durée. Etat conservé entre plusieurs appels à l'objet
  - SingleCall : pas d'état, l'objet est ré-alloué à chaque invocation.

- Activation d'un objet par le client :
  - Crée l'instance de l'objet distant et activation sur l'instanciation de la souche chez le client



La technologie .NET REMOTING

- Durée de vie d'un objet distant:
  - Problème
    - Comment libérer les ressources d'un objet partagé ?
    - Le Garbage Collector ne peut pas savoir si l'objet est utilisé ou non par un client ➔ Utilisation du principe de bail (*lease*)
  - Mode SingleCall
    - Non concerné : objet créé à chaque appel de méthode
  - Mode Singleton
    - Définition d'un objet Lease coté Serveur:
      - *InitialLeaseTime* : durée de vie de l'objet après activation et avant le premier appel de méthode (par défaut : 5 min)
      - *RenewOnCallTime* : étend la durée de vie de l'objet à chaque appel de méthode (par défaut : 2 min)
      - *CurrentLeaseTime* : durée de vie restante avant destruction de l'objet si aucun appel reçu (par défaut : 5 min)

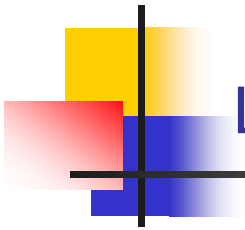
La technologie .NET REMOTING

- Exemples :
  - Redéfinir la méthode InitializeLifetimeService de l'objet **MarshalByRefObject**

```
public override object InitializeLifeTimeService()
{
    ILease lease = (ILease)base.InitializeLifetimeService();
    // Tester si l'objet n'est pas encore activé. Une fois activé, on ne peut plus modifier le Lease
    if (lease.CurrentState == LeaseState.Initial)
    {
      lease.InitialLeaseTime = TimeSpan.FromMinutes(20);
      lease.RenewOnCallTime = TimeSpan.FromMinutes(10);
    }
   return lease;
}
```

  - Ne jamais désactiver l'objet publié

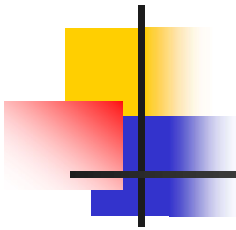
```
public override object InitializeLifeTimeService()
{
    return null;
}
```



La technologie .NET REMOTING

Création d'une solution contenant 3 projets :

- Interface
  - Signature du service mis en œuvre par les objets distants
    - Interface *Ihelllo*, méthode *hello*
  - Bibliothèque de classes
- Serveur
  - Application Console
  - Mise en œuvre du service : classe dérivée de MarshalByRefObject et mise en œuvre de la méthode publique *hello*
  - Instancier et activer l'objet distant
- Client
  - Application WinForm ou Console
  - Référencer l'interface
  - Appel de la méthode *hello* de l'objet distant



La technologie .NET REMOTING

- Exemple d'interface :

```
namespace CoteServeur {

    interface IHello {
            string Hello();
    }
}
```

La technologie .NET REMOTING

- Exemple de mise en œuvre de l'objet distant coté Serveur :

```
namespace CoteServeur
{
   public class HelloImpl : MarshalByRefObject , IHello
     {

       public string Hello()
       {
           return "hello";
       }
   }
}
```

La technologie .NET REMOTING

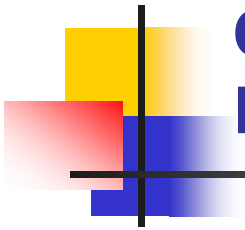
- Exemple de mise en place de l'objet :

```
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp; //  ou Http;
using CoteServeur;

namespace Serveur
{
   class Program
   {
      static void Main(string[] args)
      {
         // On crée un canal de communication tcp
         TcpChannel canal = new TcpChannel(3000);
         // Si on préfère utiliser un canal http (plus verbeux, mais passant les firewalls) :
         // déclarer le canal comme ceci : HttpChannel canal = new HttpChannel(<numéro de port optionnel : par défaut 80>);

         // On le déclare dans le système
         ChannelServices.RegisterChannel(canal);

         // Activation de l'objet par le serveur
         // Mode SingleCall : 1 instance par invocation du client  -  Mode Singleton : 1 instance par serveur (état partagé entre clients)
         RemotingConfiguration.RegisterWellKnownServiceType(typeof(CoteServeur.HelloImpl),
                                                            "monServ", // nom de l'objet
                                                            WellKnownObjectMode.SingleCall); // ou Singleton

         Console.WriteLine("Appuyer sur une touche pour terminer...");
         Console.ReadLine();
         ChannelServices.UnregisterChannel(canal);
      }
   }
}
```

La technologie .NET REMOTING

- Exemple de client :

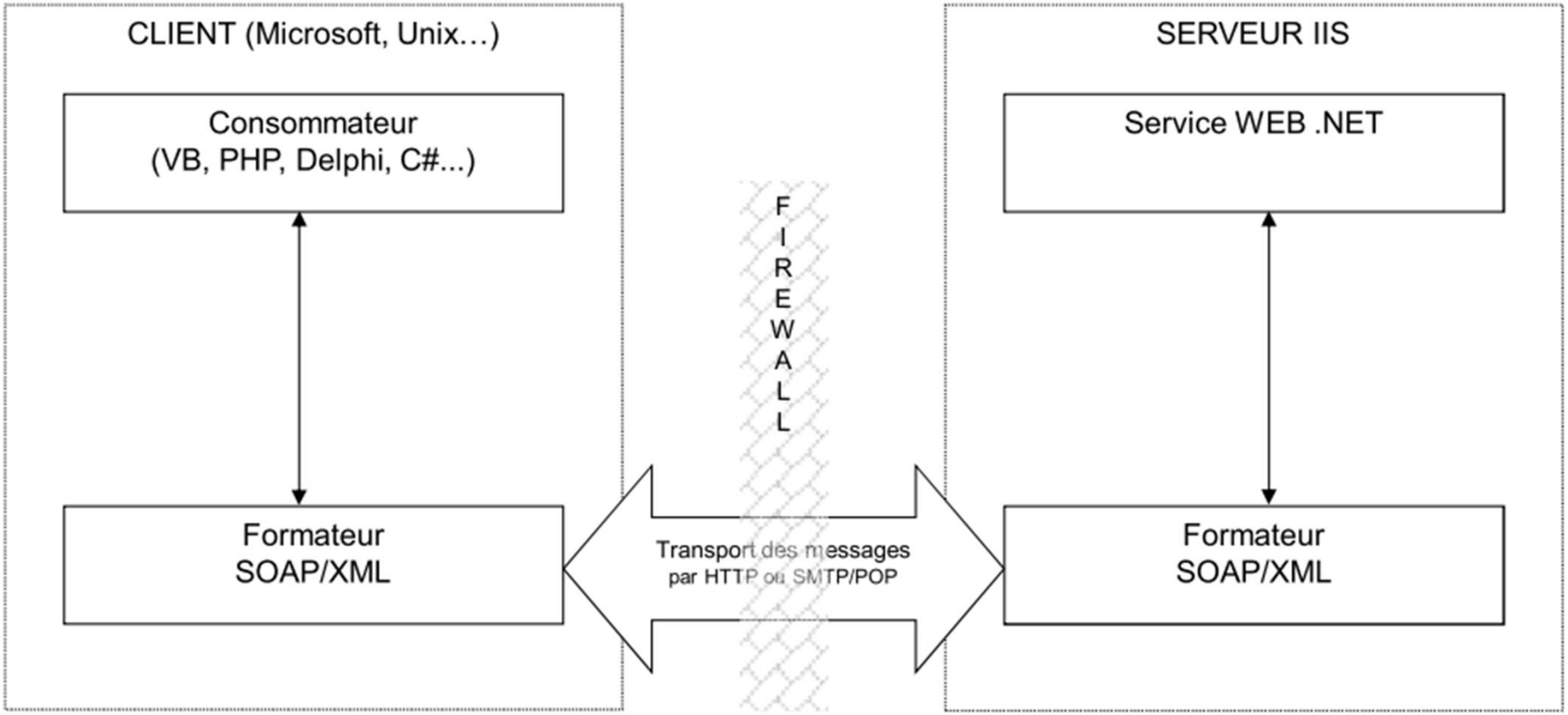
```
using CoteServeur;
private void AppelService()
 {
     // Connexion à l'objet distant, création d'une souche
     CoteServeur.Serveur proxy =
(CoteServeur.Serveur)Activator.GetObject(
                                        typeof(CoteServeur.Serveur),
                                        "tcp://localhost:3000/monServ");

    // Invocation de l'objet distant via la souche
     string result=proxy.Hello();
 }
```



Objets/services répartis avec MICROSOFT .NET

1. Introduction à C#
2. .NET Remoting
3. **Web Services .NET**

La technologie .NET Web-services



La technologie .NET Web-services

- Description d'un service (sans état) :

```
public class WebServiceCalculatrice : System.Web.Services.WebService
  {

      [WebMethod]
      public void decrementer(ref int data) {
          data = data - 1;
      }

      [WebMethod]
      public int ajouter(int a, int b)   {
          return a + b;
      }

…
```

La technologie .NET Web-services

- Exemple d'un client (test du service):

Cliquez ici pour une liste complète des opérations.

ajouter

Test

Pour tester l'opération en utilisant le protocole HTTP POST, cliquez sur le bouton 'Appeler'.

Paramètre	Valeur
a:	12
b:	6

Appeler

SOAP 1.1

Le texte suivant est un exemple de demande et de réponse SOAP 1.1. Les espaces réservés affich

```
POST /Service1.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/ajouter"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.
  <soap:Body>
    <ajouter xmlns="http://tempuri.org/">
      <a>int</a>
      <b>int</b>
    </ajouter>
  </soap:Body>
```

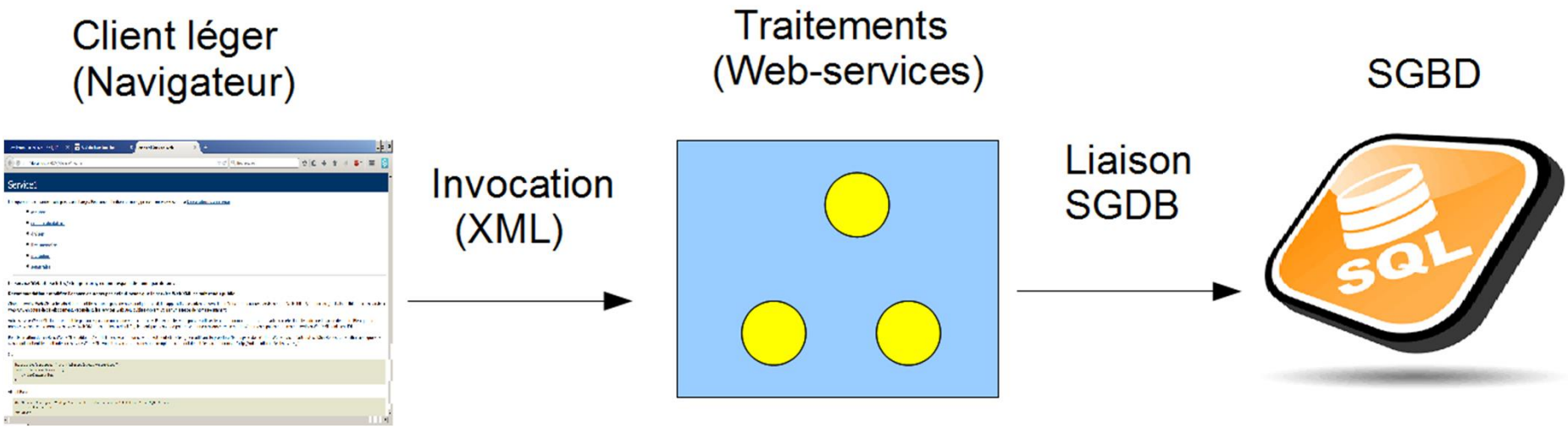
La technologie .NET Web-services

- Exemple d'un client :

```
static void Main(string[] args)
    {
        try
        {

            ServiceCalculatrice.Service1SoapClient monService = new
                    ServiceCalculatrice.Service1SoapClient();

            int res = monService.ajouter(1, 2);

            Console.WriteLine("Resultat ajouter(1,2) = ");
            Console.WriteLine(res.ToString());

           …
```

La technologie .NET REMOTING

- Architectures N-tiers :



- Comment mettre en oeuvre la persistance si le Web-service est sans état et si plusieurs interactions client/traitements ?

La technologie .NET Web-services

- Notion de service avec session (association clef/objet):

```
[WebMethod(EnableSession=true)]
public string HelloWorldAvecSession()     {
   MaSession ma;

   if (Session["compteur"] == null)     {
      ma = new MaSession();
      Session["compteur"]=ma;
   }
   else     {
      ma = (MaSession)Session["compteur"];
      ma.nb++;
   }

   return "Hello World avec session : nb_invocation = " + ma.nb.ToString();
```

La technologie .NET Web-services

- Persistance :
  - Accès à une base de données
  - Voire plus simplement des fichiers (XML):
    - Balises : start element, end element
    - Attributs
    - Sauvegarde et chargement en début fin de session
    - Parsers SAX et DOM
    - Exemple de parsers C#, la classe XmlReader :
      - Read()
      - IsStartElement()
      - Attributs Value et Name

```
<?xml version="1.0" encoding="utf-8" ?>
<annuaire>
  <abonne id=90>
    <nom> legall </nom>
    <prenom> jacques </prenom>
    <numero>0298000001</numero>
    <ville> plouzane </ville>
  </abonne>
  <abonne>
    <nom> logall </nom>
    <prenom> pierre </prenom>
    <numero>029000002</numero>
    <ville> brest </ville>
  </abonne>
</annuaire>
```