

Chapitre 8

Premiers pas .Net

L'objectif de cette annexe est de permettre à chacun de se familiariser un peu avec l'environnement de programmation `.Net`. Pour cela nous allons développer une petite application permettant à chacun de gérer ses cartes de visite (ajout, suppression et consultation) et de les rendre accessibles aux autres utilisateurs (uniquement consultation). Pour atteindre ce but, l'application se compose de deux parties : le service d'annuaire qui doit pouvoir être utilisée à distance et différentes applications clientes de type client léger ou client lourd graphique ou en ligne de commande. Pour des raisons de simplicité nous ne nous intéresserons pas aux problèmes liés à l'authentification des différentes classes d'utilisateurs.

Une version html de ce texte est disponible à l'adresse <http://rangiroa.polytech.unice.fr/riveill/enseignement/tp/dotnet.html>. Elle contiendra les évolutions futures du sujet.

8.1 Les outils à utiliser

Selon votre sensibilité ou votre OS préféré, vous pourrez exécuter vos applications `.Net` soit avec le `framework .Net` (disponible uniquement pour les plates-formes WINDOWS), soit avec Rotor (disponible pour WINDOWS, FREEBSD et MAC OS) ou Mono (disponible pour LINUX, SOLARIS, MAC OS X et WINDOWS).

8.1.1 Configuration d'un mode Emacs pour C#

Le code à écrire sera simple et afin de mettre en évidence les différentes étapes nous nous passerons dans la mesure du possible d'un IDE et utiliserons `emacs`. Il existe plusieurs modes `C#`¹ pour `Emacs` qui permettent de colorer le code source et l'indenter. Un d'entre eux, nécessite l'installation de la dernière version du `cc-mode`² sur lequel elle est construite.

¹csharp-mode : <http://mfgames.com/linux/releases/csharp-mode-0.5.0.tar.bz2>

²cc-mode : <http://ovh.dl.sourceforge.net/sourceforge/cc-mode/cc-mode-5.31.3.tar.gz>

```

MONO:~> cd \usr\share\emacs21\site-lisp\
MONO:emacs21/site-lisp> sudo tar zxvf ~/cc-mode-5.31.3.tar.gz
MONO:emacs21/site-lisp> cd cc-mode-5.31.3
MONO:site-lisp/cc-mode-5.31.3> sudo \
    emacs -batch -no-site-file -q -f batch-byte-compile cc-*.el

MONO:~> cd /usr/share/emacs21/site-lisp/
MONO:emacs21/site-lisp> sudo tar jxvf ~/csharp-mode-0.5.0.tar.bz2
MONO:emacs21/site-lisp> sudo mv csharp-mode-0.5.0.el csharp-mode.el

```

Pensez à configurer ensuite Emacs en ajoutant les lignes suivantes dans le fichier de configuration (`$HOME/.emacs`) :

```

(autoload 'csharp-mode "csharp-mode" "C# code." t)
(setq auto-mode-alist
  (append '(("\\.cs$" . csharp-mode)) auto-mode-alist))

```

Si la coloration syntaxique n'est pas visible lors de l'édition d'un document `*.cs`, c'est qu'elle n'est pas activée dans Emacs ! On l'active depuis le menu `Options/Syntax Highlighting`. On peut aussi en profiter pour activer l'`Active Region Highlighting` et le `Parent Match Highlighting`. Une fois les choix fait, ne pas oublier de sauvegarder les options choisies par un `Options/Save Options`.

8.1.2 Les outils .Net du monde Microsoft

Comme nous allons faire des développements simples nous allons utiliser les outils mis 'gracieusement' en ligne par Microsoft et les installer les uns après les autres :

1. Le *framework .Net*³ est généralement installé à l'adresse `C:/Windows/Microsoft.Net/Framework` et contient les principales classes nécessaires à l'exécution des programmes ainsi que le compilateur (`csc.exe`).
2. Le *kit de développement (SDK)*⁴ est généralement installé à l'adresse `C:/ProgramFiles/Microsoft.Net/SDK` et contient les principaux utilitaires : `xsd.exe` (pour générer pour compléter la sérialisation xml), `ildasm.exe` (le désassembleur) ou `wsdl.exe` (le générateur de talon pour les services web).

TODO vérifier le role de `xsd.exe` : ne permet pas la génération de la documentation mais uniquement de vérifier des schémas XML

3. L'environnement de programmation WEB MATRIX⁵ a comme principaux avantages d'avoir une faible empreinte mémoire, d'être gratuit, simple d'utilisation, d'inclure un serveur HTTP et de permettre la création et l'accès à des bases de données simples au format *Microsoft .Net Framework* et du *SDK* qui est généralement à l'adresse `C:/ProgramFiles/Microsoft.Net/SDK`. Pour tester la mise à jour de ces variables vous pouvez appeler le compilateur (`csc`), le désassembleur (`ildasm`) et le générateur de proxy web (`wsdl`).

³Framework .Net : <http://www.microsoft.com/france/msdn/netframework/default.aspx>

⁴SDK : <http://www.microsoft.com/france/msdn/netframework/default.aspx>

⁵Web Matrix : <http://www.asp.net/webmatrix/>

Si vous le souhaitez, MICROSOFT met à disposition des développeurs des versions gratuites de ses environnements de développement commerciaux⁶.

L'offre *MSDN Academic Alliance*⁷ permet aux universités d'avoir accès à VISUAL STUDIO, SQL SERVEUR et à l'ensemble des outils de développement commercialisés par MICROSOFT.

8.1.3 Les outils .Net du monde libre

Il est aussi possible de développer en .Net en tournant le dos aux logiciels Microsoft. Voici quelques pistes :

1. MONO⁸ est une implémentation complète du **framework .Net** et l'ensemble des binaires produit avec MONO tourne aussi sur la machine officielle .NET. MONO existe pour LINUX, SOLARIS, MAC OS X et WINDOWS. Le processus d'installation est très simple et est indiqué dans le README de chaque plate-forme. Les utilisateurs Linux trouveront un paquetage APT et les utilisateurs Mac, une *dmg*. En installant le framework, on installe aussi un IDE appelé MONODEVELOP qui est un portage Linux de SHARPDEVELOP et *gmc*s, qui est le compilateur C# 2.0 du framework MONO.
2. SHARPDEVELOP⁹ est un excellent environnement de programmation. *Note pour les utilisateurs de Windows* : Il est aussi possible d'utiliser SHARPDEVELOP comme environnement de développement, MONO comme compilateur et le framework .Net comme plate-forme d'exécution.
3. XSP est le micro-serveur Web fourni avec MONO. Par défaut, le serveur XSP se lancent en mode *stand-alone* sur le port 8080. Si on souhaite les démarrer à la demande, il suffit de retirer les scripts *ad-hoc* du repertoire */etc/init.d*. Pour des pages ou services en production, il est préférable d'utiliser APACHE avec le module *mod_apache_mono*.
4. MYSQL¹⁰ est très certainement le serveur de base de données libre le plus connu et le plus simple à administrer mais pas le plus efficace ni le plus complet.

8.2 Architecture de l'application "Cartes de Visite"

Nous proposons dans différents ateliers de concevoir une application simple permettant de consulter et mettre à jour un annuaire de cartes de visite. L'application doit pouvoir aussi bien être utilisée depuis le web que depuis un client lourd qu'il soit graphique ou en ligne de commande. Cette application se compose de trois parties : le service d'annuaire, les différentes applications clientes et les différents systèmes de persistances.

Pour des raisons de simplification, on considère que ce service est public et qu'il n'est pas nécessaire de s'authentifier pour pouvoir accéder à l'annuaire. Nous développerons cette application en C#. L'architecture finale de l'application est décrite dans la figure 8.1.

⁶Visual Studio Express : <http://www.microsoft.com/france/msdn/vstudio/express/default.msp>

⁷MSDN AA : <http://www.microsoft.com/france/msdn/abonnements/academic/default.msp>

⁸Mono : <http://www.mono-project.com/>

⁹SharpDevelop : <http://sourceforge.net/projects/sharpdevelop/>

¹⁰MySql : <http://dev.mysql.com/downloads/>

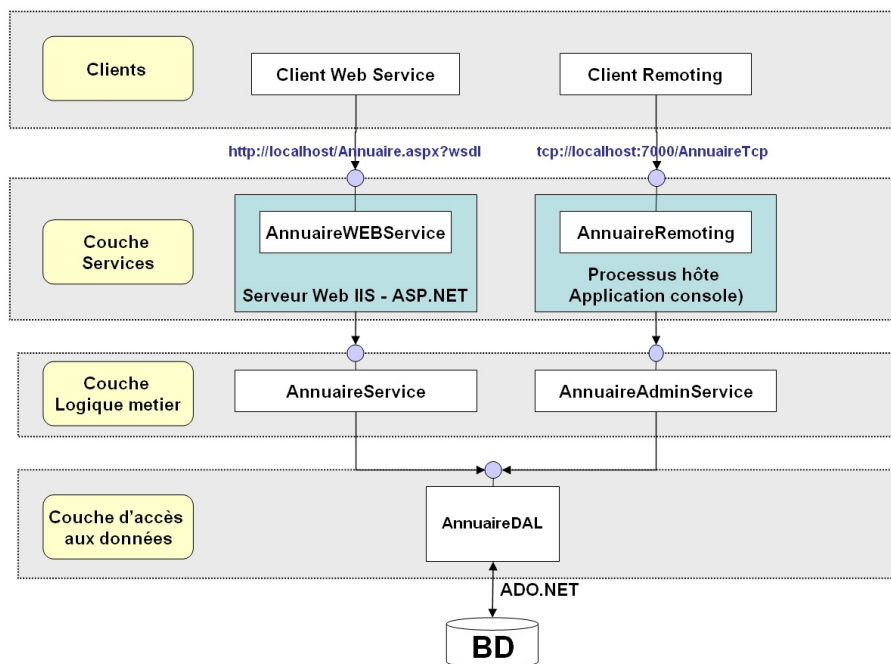


Figure 8.1 – Architecture de l'application

8.3 L'objet métier - la CarteDeVisite

Nous choisissons de représenter les données de chaque carte dans des instances d'une classe `CarteDeVisite` dont le code C# est donné ci-dessous :

```

using System;
2 using System.Xml;
using System.Xml.Serialization;

namespace MonPremierTP
{
7 [Serializable]
[XmlType("carte-de-visite")]
public class CarteDeVisite
{
12 [XmlElement("nom")] public string Nom;
[XmlElement("prenom")] public string Prenom;
[XmlElement("email")] public string Email;
[NonSerialized()] public string Telephone;
}
}

```

Listing 8.1 – Fichier `CarteDeVisite.cs`

Nous verrons ultérieurement comment les différentes fiches seront initialisées par des données issues de la base de données. La description et l'utilisation des différents attributs sont présentés dans la section 8.6.4.

Pour générer une bibliothèque (DLL) qui contiendra le résultat de la compilation du fichier précédemment décrit, la commande est la suivante :

```
WIN32> csc /target:library CarteDeVisite.cs
MONO:~/TP> gmcs /target:library CarteDeVisite.cs
```

La librairie créée aura son nom par défaut : `CarteDevisite.dll`.

8.4 La couche d'accès aux données (*DAL, Data Access Layer*)

Cette couche permet d'alimenter en données les différents objets métiers manipulés par l'application cartes de visite. Nous donnons uniquement ici une version vide (v0.1) de la classe. Elle sera complétée ultérieurement :

```
using System;
2 using System.Reflection; /** Pour accéder aux attributs **/
[assembly:AssemblyVersion("0.1.0.0")]

namespace MonPremierTP
{
7   public class AnnuaireDAL
   {
       /** Constructeur **/
       public AnnuaireDAL() {}

12      /** operation de recherche d une carte dans la BD **/
       public CarteDeVisite ExtraireCarte(string nomRecherche)
       { return null; }

       /** operation d insertion d une nouvelle carte **/
17      public void AjouterCarte(CarteDeVisite carte) {}
   }
}
```

Listing 8.2 – Fichier `AnnuaireDAL.cs`

Pour généré la bibliothèque `AnnuaireDAL.dll`, la commande est la suivante :

```
1 [csc ou gmcs] /target:library /reference:CarteDeVisite.dll AnnuaireDAL.cs
```

8.5 La couche métier

Quand nous aurons implémenté la classe `AnnuaireDAL` décrite dans la section précédente, nous disposerons d'un moyen d'accéder aux données persistantes du service d'annuaire sans avoir à connaître la technologie de stockage employée. L'interrogation et la mise à jour des données seront faites au travers des interfaces de la *DAL (Data Access Layer)* en ne manipulant que des entités liées au domaine métier.

Pour créer l'indépendance entre conservation des données et manipulation des données, nous allons créer la couche contenant la logique du service d'annuaire. Il s'agit d'implanter

deux services, l'un pour consulter les cartes de visites existantes et un autre pour en insérer de nouvelles. L'idée est ici de spécifier clairement les services sous la forme d'interface. Cette couche ne sera manipulée qu'au travers de ces seules interfaces et non par un accès direct à leur implantation. Voici les interfaces des deux services :

8.5.1 Interface vers la *DAL*

```

using System;

namespace MonPremierTP
4 {
    /** Le service d annuaire **/
    public interface IAnnuaireService {
        CarteDeVisite Rechercher(string nom);
    }
9
    /** Administration de l annuaire **/
    public interface IAnnuaireAdminService {
        CarteDeVisite Rechercher(string nom);
        void Ajouter(CarteDeVisite carte);
14 }
}

```

Listing 8.3 – Fichier IAnnuaireService.cs

8.5.2 Implantation des interfaces

Une fois ces interfaces définies, il ne nous reste qu'à les implanter en utilisant la couche d'accès aux données décrite précédemment.

```

using System;
namespace MonPremierTP
{
5   public class AnnuaireService : IAnnuaireService
    {
        /** constructeur de la classe **/
        public AnnuaireService() {}

        /** implementation de la methode metier **/
10   public CarteDeVisite Rechercher(string nom) {
            AnnuaireDAL dal=new AnnuaireDAL();
            return dal.ExtraireCarte(nom);
        }
    }
15
    public class AnnuaireAdminService : IAnnuaireAdminService
    {
        /** constructeur de la classe **/
        public AnnuaireAdminService(){ }
20
        /** implementation des methodes metiers **/
        public CarteDeVisite Rechercher(string nom)
        {

```

```
25     AnnuaireDAL dal=new AnnuaireDAL();
        return dal.ExtraireCarte(nom);
    }

    public void Ajouter(CarteDeVisite carte)
    {
30     AnnuaireDAL dal=new AnnuaireDAL();
        /** on verifie que la carte n existe pas deja **/
        CarteDeVisite test=dal.ExtraireCarte(carte.Nom);
        if (test==null)
            dal.AjouterCarte(carte);
35     else
            throw new Exception("La carte existe déjà");
    }
}
}
```

Listing 8.4 – Fichier AnnuaireService.cs

8.6 Quelques petites expériences programmatiques

Ces expérimentations ne sont pas directement liées à la mise en œuvre de l'application cartes de visite mais permettent de se familiariser avec le langage C# et le framework .Net.

8.6.1 Construire le diagramme UML

A cette étape du TP, nous avons écrit 4 fichiers : `CarteDeVisite.cs`, `AnnuaireDAL.cs`, `IAnnuaireService.cs` et `AnnuaireService.cs` et il peut être souhaitable de construire le diagramme UML de l'ensemble.

8.6.2 Générer les fichiers de documentation

Comme pour tout langage de programmation, il est possible de commenter des programmes C#. Au delà des commentaires, introduits par `//` et par `/*` qui fonctionnent comme en C ou C++, il existe un troisième type de commentaire introduit par `///`. Pour ce dernier type, le compilateur C# peut extraire la documentation présente dans le code pour les placer dans un fichier XML. Ces commentaires sont particulièrement indiqués pour commenter tous les types utilisateurs (*classe*, *délégué* ou *interface*), les membres (*champ*, un *événement*, une *propriété* ou une *méthode*) ou une déclaration d'espace de noms. Dans *Visual Studio* ou *SharpDevelop*, le fait de mettre `///` dans un endroit valide fait apparaître un menu avec les balises XML de documentation autorisées. Nous présentons ici les principales balises :

- La balise *summary* sert à donner la description complète de l'élément que l'on souhaite documenter. Il peut être utilisé sur une classe, une méthode, une propriété ou une variable.
- La balise *param* permet de documenter les paramètres d'une méthode ou d'une propriété. Il prend en complément uniquement le nom de la variable, le type de la variable est automatiquement déterminé par le générateur de documentation.

- La balise *returns* permet de documenter la valeur de retour d'une fonction seulement.
- La balise *value* permet de décrire la valeur de retour ou d'entrée d'une propriété. Elle joue le même rôle que la balise *param* pour une fonction.
- La balise *paramref* permet d'indiquer que le mot dans le commentaire est un paramètre de la fonction afin que le générateur de documentation puis mettre un lien vers le commentaire du paramètre.
- La balise *exception* permet d'informer sur le(s) type(s) d'exception(s) que la fonction peut lever. La propriété *cref* du tag permet de spécifier le type d'exception documenté.
- Il existe bien d'autres balises : `<c>`, `<code>`, `<example>`, `<include>`, `<list>`, `<para>`, `<permission>`, `<remarks>`, `<see>`, `<seealso>`. A vous de découvrir leur utilisation en lisant la documentation.

Voici la classe `AnnuaireDAL` commentée :

```

1 using System;
using System.Reflection; /** Pour acceder aux attributs **/
[assembly:AssemblyVersion("0.2.0.0")]

namespace MonPremierTP
6 {

    /// texte pour la classe AnnuaireDAL
    /// <summary>
    /// Description de la classe AnnuaireDAL .
11    /// </summary>
    public class AnnuaireDAL
    {
        /// <remarks>
        /// Des commentaires plus longs peuvent être associés à un type
16    /// ou un membre grâce à la balise remarks</remarks>
        /// <example> Cette classe permet d isoler la partie métier de la
        /// persistance des données, elle comporte essentiellement deux
        /// méthodes.
        /// <code>
21    /// public CarteDeVisite ExtraireCarte(string nomRecherche)
        /// public void AjouterCarte(CarteDeVisite carte)
        /// </code>
        /// </example>
        /// <summary>
26    /// Constructeur de la classe AnnuaireDAL.
        /// </summary>
        public AnnuaireDAL() {} /** constructeur **/

        /// texte pour la méthode ExtraireCarte
31    public CarteDeVisite ExtraireCarte(string nomRecherche)
        {
            /** operation de recherche d une carte dans la BD **/
            return null;
        }

36    /// texte pour la méthode AjouterCarte
        /// <summary>
        /// Description de AjouterCarte.</summary>

```

```

41  /// <param name="carte"> Description du paramètre carte </param>
    /// <seealso cref="String">
    /// Vous pouvez utiliser l attribut cref sur n importe quelle balise
    /// pour faire référence à un type ou un membre, et le compilateur
    /// vérifiera que cette référence existe.
    /// </seealso>
46  public void AjouterCarte(CarteDeVisite carte)
    {
        /** operation d insertion d une nouvelle carte **/
    }
51 }

```

Listing 8.5 – Fichier AnnuaireDAL_COMMENT.cs commenté

La génération de la documentation fait partie intégrante du processus de compilation et ces deux processus ne sont pas dissociable. Pour compiler et générer la documentation, il suffit d'appeler le compilateur avec les options adéquates¹¹ :

```

[csc ou gmcs] \
  /doc:AnnuaireDAL_doc.xml /r:CarteDeVisite.dll /t:library \
  AnnuaireDAL_COMMENT.cs

```

Examinez le fichier XML produit.

TODO *J'ai 3 warning sous mono que je n'avais pas sous la version officielle*

8.6.3 Désassembleur

Commençons par compiler les différents fichiers pour obtenir une DLL.

```

2 [csc ou gmcs] /t:library /out:Annuaire.dll \
  /r:CarteDeVisite.dll /r:AnnuaireDAL.dll \
  IAnnuaireService.cs AnnuaireService.cs

```

La DLL `Annuaire.dll` contient :

- Le code intermédiaire (CIL) des fichiers `IAnnuaireService.cs` et `AnnuaireService.cs`.
- Deux références vers des DLL tierces : `CarteDeVisite.dll` et `AnnuaireDAL.dll`.

Conséquence : Si lors d'une compilation ultérieure on référence `Annuaire.dll`, on ne pourra accéder qu'à son contenu. Les types définis dans les DLL tierces référencées par `Annuaire.dll` ne seront pas accessible directement et il faudra référencer la DLL.

Pour observer le contenu d'un assemblage .Net depuis WINDOWS, on utilise l'utilitaire `ildasm` et depuis MONO, l'utilitaire `monodis`.

```

2 WIN32> ildasm Annuaire.dll
MONO:~/TP> monodis Annuaire.dll

```

Observez la DLL produite et commentez. Qu'observez-vous : nombre de classes, fonctions membres ?

¹¹Nous avons remplacer `/target` et `/reference` par leur format abrégé `/t` et `/r`.

8.6.4 Attributs utilisateurs et exemples d'utilisation

TODO A remettre après avoir trouvé de de bons exemples d'utilisation

TODO SOAPextension qui permettent de modifier la sérialisation XML par défaut

TODO CastleProjet qui utilise les attributs pour spécifier les points de coupe du tisseur d'aspects

TODO .Net 3.0 qui permet de choisir .Net Remoting ou service web au dernier moment

8.6.5 Réflexivité et introspection

Nous allons utiliser la réflexivité pour avoir un aperçu des capacités introspectives de .Net et découvrir les fonctions membres héritées.

```

using System;
2 using System.IO;
using System.Reflection;

public class Meta
{
7   public static int Main ()
   {
       /** 1. Lire l assemblage **/
       Assembly a = Assembly.LoadFrom ("Annuaire.dll");

12      /** 2. Lire tous les modules de l assembly **/
       Module[] modules = a.GetModules();

       /** 3. Inspecter le premier module **/
       Module module = modules[0];

17      /** 4. Lire tous les types du premier module **/
       Type[] types = module.GetTypes();

       /** 5. Inspecter tous les types **/
22      foreach (Type type in types)
       {
           Console.WriteLine("Le type {0} a cette(ces) methode(s) : ",
                               type.Name);

27          /** 5.1 Inspecter toutes les methodes du type **/
           MethodInfo[] mInfo = type.GetMethods();
           foreach (MethodInfo mi in mInfo)
               Console.WriteLine (" {0}", mi);
       }

32      return 0;
   }
}

```

Listing 8.6 – Fichier Meta.cs

Il ne reste plus qu'à compiler la classe Meta.cs et exécuter le code CIL produit.

```

1 WIN32> csc /t:exe Meta.cs
WIN32> Meta.exe

```

```
MONO:~/TP> gmcs /t:exe Meta.cs
MONO:~/TP> mono Meta.exe
```

8.6.6 Sérialisation binaire et XML

Voici un exemple permettant de sérialiser et désérialiser un objet selon différents formats.

```
using System;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;
using System.Xml;
5 using System.Xml.Serialization;
using MonPremierTP;

class Serialisation
{
10 static void Main(string[] args)
    {
        CarteDeVisite uneCarte = new CarteDeVisite();

        uneCarte.Nom = "JOFFROY";
15 uneCarte.Prenom = "Cedric";
        uneCarte.Email = "joffroy@polytech.unice.fr";
        uneCarte.Telephone = "04 93 XX XX XX";

        /** on serialise en binaire et on sauvegarde dans un fichier **/
20 Stream stream = File.Open("Demo.bin", FileMode.Create);
        BinaryFormatter formatter = new BinaryFormatter ();
        formatter.Serialize (stream, uneCarte);
        stream.Close();

        /** on serialise en XML et on sauvegarde dans un fichier **/
25 Stream stream2 = File.Open("Demo.xml", FileMode.Create);
        XmlSerializer serializer = new XmlSerializer (typeof(CarteDeVisite));
        serializer.Serialize(stream2, uneCarte);
        stream2.Close();

30 /** on deserialise depuis un fichier (mode binaire) **/
        stream = File.Open ("Demo.bin", FileMode.Open);
        formatter = new BinaryFormatter();
        CarteDeVisite obj = (CarteDeVisite) formatter.Deserialize (stream);
35 stream.Close ();

        /** on deserialise depuis un fichier (mode XML) **/
        stream2 = File.Open ("Demo.xml", FileMode.Open);
        serializer = new XmlSerializer (typeof(CarteDeVisite));
40 CarteDeVisite obj2 = (CarteDeVisite) serializer.Deserialize (stream2);
        stream2.Close ();

        /** On affiche les resultats du jeu de test **/
45 Console.WriteLine ("\t\t origine \t\t binaire \t\t Xml");

        Console.WriteLine ("uneCarte.nom a ete serialisee");
```

```

    Console.WriteLine ("\t\t {0} \t\t {1} \t\t {2}",
        uneCarte.Nom, obj.Nom, obj2.Nom);

50 Console.WriteLine ("uneCarte.prenom a ete serialisee");
    Console.WriteLine ("\t\t {0} \t\t {1} \t\t {2}",
        uneCarte.Prenom, obj.Prenom, obj2.Prenom);

    Console.WriteLine ("uneCarte.email a ete serialisee");
55 Console.WriteLine ("\t\t {0} \t\t {1} \t\t {2}",
        uneCarte.Email, obj.Email, obj2.Email);

    /** Telephone n a pas ete serialise en binaire, mais l a ete en Xml */
    Console.WriteLine ("uneCarte.telephone n a pas ete serialisee");
60 Console.WriteLine ("\t\t {0} \t\t {1} \t\t {2}",
        uneCarte.Telephone, obj.Telephone, obj2.Telephone);
}
}

```

Listing 8.7 – Fichier SerialisationTest.cs

Compilation et exécution

```

WIN32> csc /t:exe /r:CarteDeVisite.dll SerialisationTest.cs
2 WIN32> SerialisationTest.exe

MONO:~/TP> gmcs /t:exe /r:CarteDeVisite.dll SerialisationTest.cs
MONO:~/TP> mono SerialisationTest.exe

```

Remarque : si vous sérialisez un tableau ou une collection, tous les objets du tableau ou de la collection sont sérialisés.

8.7 La couche de stockage physique des données

On reprend la conception de l'application. Les données stockées dans l'annuaire sont des cartes de visite contenant un nom, un prénom, un courriel et un numéro de téléphone. Selon la plate-forme utilisée la manière de créer la base de données diffère.

8.7.1 Utilisation d'une base de données Access

Bien entendu, cette solution est réservée aux utilisateurs WINDOWS.

Création de la base de données

Pour que ceux qui ne sont pas très aguerris avec la manipulation des bases de données, WEB MATRIX possède un mode 'création de base de données' qui permet entre autre de créer une base de données Access. Voici la manière de procéder :

1. ouvrir *web matrix*
2. aller dans la fenêtre 'Workspace'
3. sélectionner l'onglet 'Data'

4. cliquer sur l'icone 'Nouvelle Base de Données'
5. créer une base de données Access
6. créer la table en sélectionnant table, dans la BD créée
7. cliquer sur l'icone 'Nouvelle Table'
8. pour finir, créer les entrées de la table qui doivent au moins contenir les champs suivants ainsi qu'une clé primaire non décrite ici :

```
string Nom;
string Prenom;
string Email;
string Telephone;
```

Par exemple la table obtenue peut avoir le format décrit dans la figure 8.2.

Column Name	Data Type	Size	Allow Nulls
key	AutoNumber	0	False
nom	Text	50	False
prenom	Text	50	True
email	Text	50	True
telephone	Text	50	True

Figure 8.2 – Format de la base de données

Initialisation de la base de données

Il est alors possible d'initialiser la base de données en sélectionnant l'onglet 'Data'. Un exemple de contenu est donné dans la figure 8.3.

	key	nom	prenom	email	telephone
▶	1	RIVEILL	Michel	riveill@unicef	(null)
	2	BLAY	Mireille	(null)	(null)
	3	PINNA	Anne-Marie	(null)	(null)
	4	TIGLI	Jean-Yves	(null)	(null)
	5	HUGUES	Anne-Marie	(null)	(null)
*					

Figure 8.3 – Contenu de la base de données

8.7.2 Création et connexion à une base de données MySql

Création de la base de donnée

L'utilisation d'une base de données MYSQL s'impose aux utilisateurs de MONO mais peut aussi être choisi par les utilisateur du Framework .Net. Pour créer et initialiser la base de données `mon_premier_tp_dot_net` du moteur MYSQL nous allons utiliser le script suivant :

```

-- Creation de la base de donnees
DROP DATABASE IF EXISTS mon_premier_tp_dot_net;
CREATE DATABASE mon_premier_tp_dot_net;
USE mon_premier_tp_dot_net;
5
-- Creation de la relation
CREATE TABLE mon_annuaire (
  id          INTEGER NOT NULL PRIMARY KEY auto_increment,
  nom         VARCHAR (255) NOT NULL,
10  prenom     VARCHAR (255) NOT NULL,
  email      VARCHAR (255) NOT NULL,
  telephone  VARCHAR (255) );

-- Insertion des tuples initiaux
15 INSERT INTO 'mon_annuaire' VALUES
  (NULL, 'MOSSER', 'Sebastien', 'mosser@polytech.unice.fr', NULL),
  (NULL, 'JOFFROY', 'Cedric', 'joffroy@polytech.unice.fr', NULL),
  (NULL, 'RIVEILL', 'Michel', 'riveill@unice.fr', NULL);

20 -- Fin du script

```

Listing 8.8 – Fichier DatabaseCreate.sql

La création de la base se fait de la manière suivante :

```
MONO:~/TP> mysql -u root < Database.sql
```

Connexion à la base de données

Le framework ne fournit pas en standard un moyen d'accéder à un système de gestion de base de données de type MySQL, il convient donc de récupérer la DLL ¹². Ecrite en .Net, la DLL est portable sur les différentes plates-formes.

Pour la rendre disponible, il convient de la publier dans le GAC.

```
MONO:~/TP> sudo gacutil -i MySql.Data.dll -package MySql
```

8.8 Accès direct à la base de données par un client léger

Dans un premier temps et pour nous familiariser avec l'environnement, on va rompre l'architecture *n*-tiers mise en place et accéder directement aux données depuis un client léger en utilisant une page ASP.Net.

Le principe de la page ASP.Net que l'on va écrire est simple :

1. L'utilisateur saisit un nom dans une TextBox
2. Il valide sa requête en cliquant sur un Button
3. La page effectue une requête sur le moteur de base de données et affiche les résultats obtenus dans une DataGridView.

¹²Connecteur MySql : <http://dev.mysql.com/downloads/connector/net/5.1.html>, prendre la version 'Windows Binaries with no installer (ZIP)'

8.8.1 Utilisateurs de Web Matrix

L'environnement WEB MATRIX simplifie, comme tout environnement de programmation l'écriture de type de page contenant des éléments graphiques, une réactions à un clic souris sur un bouton, la connexion à une base de données et l'exécution d'une requête Sql. La manière de procéder est la suivante :

1. ouvrir WEB MATRIX
2. créer un nouveau fichier ('File', 'New File') puis 'General', 'ASP.Net Page', et finir de remplir le formulaire (le suffixe du fichier sera aspx).
3. une fenêtre d'édition comportant 3 modes : 'dessin', 'html' et 'code' est à votre disposition ('all' est la fusion du mode 'html' et 'code'). En mode 'dessin' mettez sur la page une `TextBox` (saisie de chaine de caractère), un `DataGrid` et un `Button`.
4. en mode 'html' vous pouvez renommer les identificateurs des différents éléments. Par exemple pour moi, le `TextBox` a "nom" comme identificateur, le `DataGrid` a "listeCartes" comme identificateur et le `Button` a "Button" comme id et "Chercher" comme Text.
5. écrire la méthode `ExtraireCarteSQL` à l'aide de l'utilitaire fournit par WEB MATRIX (procédure décrite ci-après).

Pour écrire la méthode `ExtraireCarteSQL` à l'aide de l'utilitaire présent dans WEB MATRIX il faut être en mode 'code', puis faire glisser la zone `SELECT Data Method` de la colonne `ToolBox` à l'endroit ou vous voulez insérer la méthode. Une boite de dialogue s'ouvre. Elle permet de sélectionner la base de données, puis de construire la requête `SELECT` souhaitée.

Pour cela, sélectionnez les bonnes colonnes, remplissez la clause `WHERE` pour obtenir la requête décrite dans la figure 8.4.

Après avoir cliqué sur le bouton suivant, testez la requête écrite. Si elle convient, donnez le nom de la méthode et demandez le résultat sous la forme d'un `DataSet`.

Pour terminer l'exemple, il reste à programmer la méthode appelée lorsque le bouton sera sélectionné. Pour cela, en mode 'dessin', cliquez 2 fois sur le `Button`. La fenêtre d'édition bascule en mode 'code' et l'on peut saisir le code associé à l'évènement 'clic sur le `Button`'. Le code de cette méthode est le suivant :

```
listeCartes.DataSource = ExtraireCarteSQL (nom.Text);  
listeCartes.DataBind();  
listeCartes.Visible = true;
```

Listing 8.9 – Dans la page .aspx

Petites explications : la première ligne met à jour la liste des données du `DataSet` en appelant la méthode `ExtraireCarteSQL` construite dans l'étape 6 précédente, le paramètre est le texte de la `TextBox` contenue dans la page. Les deux lignes mettent à jour le `DataSet` et le rendent visible.

Attention : cette méthode, certes très efficace en terme de programmation ne respecte pas du tout l'architecture initiale, elle a juste permis d'utiliser la base de données créée et de nous familiariser avec WEB MATRIX pour la création de page web dynamique. La suite du TP reprend la construction pas à pas des différentes couches de l'application cartes de visite.

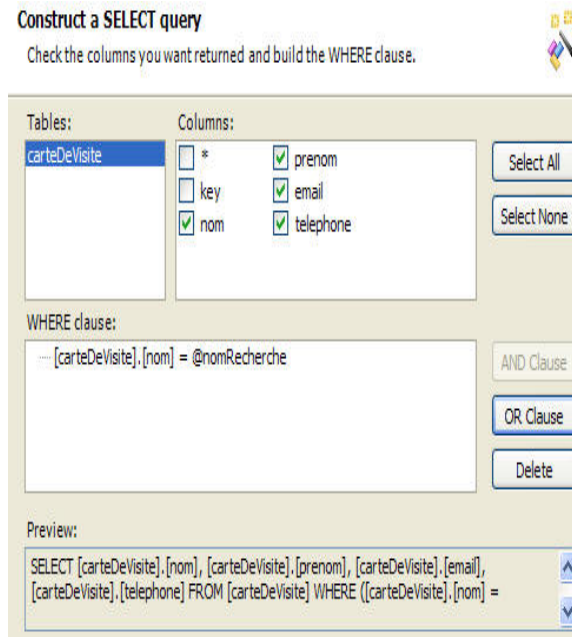


Figure 8.4 – Clause WHERE

8.8.2 Utilisateurs de Mono

Création du répertoire de travail

On crée un répertoire racine pour XSP. Ce répertoire doit contenir un sous répertoire bin.

```
2  .
   |-- www
   |   |-- bin
```

Création de la page aspx en *code-behind*

```
2  <%@ Page Language="C#" Inherits="CarteDeVisiteAccess"
   src="databaseAccess.aspx.cs" %>
   <html>
     <head>
       <title> Acces direct a la base de donnees </title>
     </head>
7   <body>
     <form runat="server">
       <p>
         <asp:TextBox id="TextBox1" runat="server"></asp:TextBox>
         <asp:Button id="Button1" onclick="Button1_Click"
12        runat="server" Text="Search"></asp:Button>
       </p>
       <p>
         <asp:DataGrid id="DataGrid1" runat="server"></asp:DataGrid>
```

```
17     </p>
    <p>
        <asp:Label id="Label1" runat="server"></asp:Label>
    </p>
</form>
</body>
22 </html>
```

Listing 8.10 – Fichier databaseAccess.aspx

Création de la classe *code-behind*

```
using System;
using System.Data;
3 using System.Web.UI.WebControls;
using MySql.Data.MySqlClient;

public class CarteDeVisiteAccess : System.Web.UI.Page
{
8     public Button Button1;
    public TextBox TextBox1;
    public DataGrid DataGrid1;
    public Label Label1;

13     DataSet ExtraireCarteSQL(string arguments)
    {
        string connectionString =
            "Server=localhost;" +
            "Database=mon_premier_tp_dot_net;" +
18         "User ID=root;" +
            "Password=" +
            "Pooling=false";

        IDbConnection dbcon;
23         dbcon = new MySqlConnection(connectionString);
        dbcon.Open();

        IDbCommand dbcmd = dbcon.CreateCommand();
        string sql =
28         String.Format(
            "SELECT nom, prenom, email, telephone \
                FROM mon_annuaire \
                WHERE nom = '{0}'",
            arguments);

33         dbcmd.CommandText = sql;
        IDbDataAdapter dataAdapter = new MySqlDataAdapter();
        dataAdapter.SelectCommand = dbcmd;
        DataSet dataSet = new DataSet();
38         dataAdapter.Fill(dataSet);

        dbcmd.Dispose();
        dbcmd = null;
        dbcon.Close();
43         dbcon = null;
```

```

    return dataSet;
}
48 public void Button1_Click(object sender, EventArgs e) {
    DataGrid1.DataSource = ExtraireCarteSQL(TextBox1.Text);
    DataGrid1.DataBind();
    DataGrid1.Visible = true;
}
53 }

```

Listing 8.11 – Fichier databaseAccess.aspx.cs

Configuration de l'application Web

La page utilise un assemblage non-standard pour accéder au moteur de base de données MySQL. Cet assemblage a beau être présent dans le GAC, il n'est pas chargé par le framework lors de l'exécution de pages `aspx`.

On va donc *configurer* l'application créée pour accéder à cet assemblage.

```

<configuration>
2 <system.web>
    <compilation>
        <assemblies>
            <add assembly="MySql.Data" />
        </assemblies>
7 </compilation>
</system.web>
</configuration>

```

Listing 8.12 – Fichier web.config

Remarque : par défaut, la page `aspx` référence automatiquement toutes les DLLs présentes dans son sous-répertoire `bin` de la racine du serveur web.

Lancement du serveur XSP

Le contenu du répertoire virtuel est le suivant :

```

MONO:~/TP/www> tree .
.
|-- bin
|-- <rep>
    |-- databaseAccess.aspx
    |-- databaseAccess.aspx.cs
    |-- web.config

```

Pour exécuter la page ASP, il est nécessaire de démarrer le serveur XSP¹³ :

```

1 MONO:~/TP/asp-test> xsp2

```

¹³ xsp démarre le serveur avec la version 1 de .Net et xsp2 avec la version 2 du framework

On peut maintenant tester la page ASP depuis son butineur web préféré :

```
MONO:~> firefox http://localhost:8080/<rep>/databaseAccess.aspx
```

8.9 Couche d'accès aux données

Cette couche a pour rôle de faire abstraction de la technologie utilisée pour stocker les données. Elle doit masquer le moteur de base de données utilisé à la couche suivante, la couche métier que nous avons déjà écrite. L'avantage de cette abstraction est qu'il est possible de remplacer le moteur de SGBD utilisé par tout autre moteur de base de données ou encore par un système de stockage simple à l'aide d'un système de gestion de fichier (sérialisation binaire ou XML). Si une telle modification avait lieu, seule cette couche d'accès aux données devrait être réécrite et la modification serait transparente au reste de l'application.

L'API ADO.Net est utilisée pour dialoguer avec le serveur de base de données utilisé. Il faut maintenant implémenter la classe en charge du dialogue avec la base de données pour lire et insérer des cartes de visite. Il faut en particulier écrire les fonctions `ExtraireCarteSQL` et `InsererCarteSQL` qui seront utilisées dans la classe `AnnuaireDal`. Le code à écrire est proche de celui de la section 8.8) et complète le code suivant :

```
public CarteDeVisite ExtraireCarte(string nomRecherche) {
    CarteDeVisite carte = new CarteDeVisite ();
    System.Data.DataSet requete = ExtraireCarteSQL (nomRecherche);
4   carte.Nom = (string) requete.Tables["Table"].Rows[0]["nom"];
    carte.Prenom = (string) requete.Tables["Table"].Rows[0]["prenom"];
    carte.Email = (string) requete.Tables["Table"].Rows[0]["email"];
    try {
        carte.Telephone = (string) requete.Tables["Table"].Rows[0]["telephone"];
9   } catch (Exception e) {
        carte.Telephone = "";
    }
    return carte;
}
14
public void AjouterCarte(CarteDeVisite carte) {
    AjouterCarteSQL (carte.Nom, carte.Prenom, carte.Email, carte.Telephone);
}
```

Listing 8.13 – Fichier `AnnuaireDAL.cs`

8.10 La logique métier

Nous avons maintenant les différents éléments pour construire la bibliothèque `AnnuaireService.dll` par compilation du fichier `annuaireService.cs`. Voici un exemple de *compilation séparée* :

```
WIN32> REM production des différents modules
WIN32> csc /t:module CarteDeVisite.cs
3 WIN32> csc /t:module /addModule:CarteDeVisite.netmodule AnnuaireDAL.cs
WIN32> csc /t:module /addModule:CarteDeVisite.netmodule IAnnuaireService.cs
```

```

WIN32> REM liaison des modules produits avec la logique métier
WIN32> csc /out:AnnuaireService2.dll /t:library \
8 /addModule:IAnnuaireService.netmodule \
  /addModule:CarteDeVisite.netmodule \
  /addmodule:AnnuaireDAL.netmodule AnnuaireService.cs

```

Pour les utilisateurs de MONO, la génération des modules provoque souvent une exception lors de la phase de compilation. Il est préférable d'effectuer une *compilation globale*, possible aussi pour les utilisateurs WINDOWS.

```

WIN32> csc /out:AnnuaireService.dll /t:library \
  IAnnuaireService.cs AnnuaireService.cs \
  CarteDeVisite.cs AnnuaireDAL.cs
5 MONO:~TP/> gmcs /out:AnnuaireService.dll \
  /t:library \
  /r:System.Data.dll /r:/path-installation/MySQL.Data.dll \
  IAnnuaireService.cs AnnuaireService.cs \
  CarteDeVisite.cs AnnuaireDAL.cs

```

8.11 Utilisation depuis un client local

Nous allons maintenant, construire un client local capable d'ajouter une nouvelle fiche et de consulter les fiches présentes dans la base. Voici un client rudimentaire :

```

1 using System;
  using MonPremierTP;

  public class ClientLocal
  {
6     public ClientLocal() {}

    static int Main()
    {
11     try
        {
            IAnnuaireAdminService annuaire = new AnnuaireAdminService ();
            CarteDeVisite carte = annuaire.Rechercher ("MOSSER");
            Console.WriteLine ("le prenom de MOSSER est {0}", carte.Prenom);

16     CarteDeVisite carteICAR = new CarteDeVisite ();
            carteICAR.Nom = "ICAR";
            carteICAR.Prenom = "Ecole";
            carteICAR.Email = "";
            carteICAR.Telephone = "";
21     annuaire.Ajouter(carteICAR);

            CarteDeVisite carteRecherche = annuaire.Rechercher ("ICAR");
            Console.WriteLine ("l email de ICAR est {0}",
                                carteRecherche.Email);
26     } catch (Exception e)
        {
            Console.WriteLine (e.ToString());

```

```

31     }
        return 0;
    }
}

```

Listing 8.14 – Fichier ClientLocal.cs

Pour compiler le client, aucune difficulté, il suffit d'appeler le compilateur et de lui demander de générer un exécutable.

```
csc/gmcs /t:exe /r:AnnuaireService.dll ClientLocal.cs
```

Note : Attention, dans les exemples donnés, afin d'alléger le code, toutes les exceptions n'ont pas été traitées.

8.12 Accès à distance

Il s'agit de publier maintenant le service d'annuaire afin de le rendre accessible à des clients distants. Nous allons publier ce services de deux manières différentes : via le *.Net Remoting* puis au travers d'un *service web*.

Dans le cas de cet exemple d'annuaire, il s'agit d'enrober la logique métier de l'application dans des conteneurs permettant l'accès distant au service. La figure 8.5 présente les différents objets qui seront créés dans le cas de l'approche *.Net Remoting* et dans le cas de l'approche par *service web*.

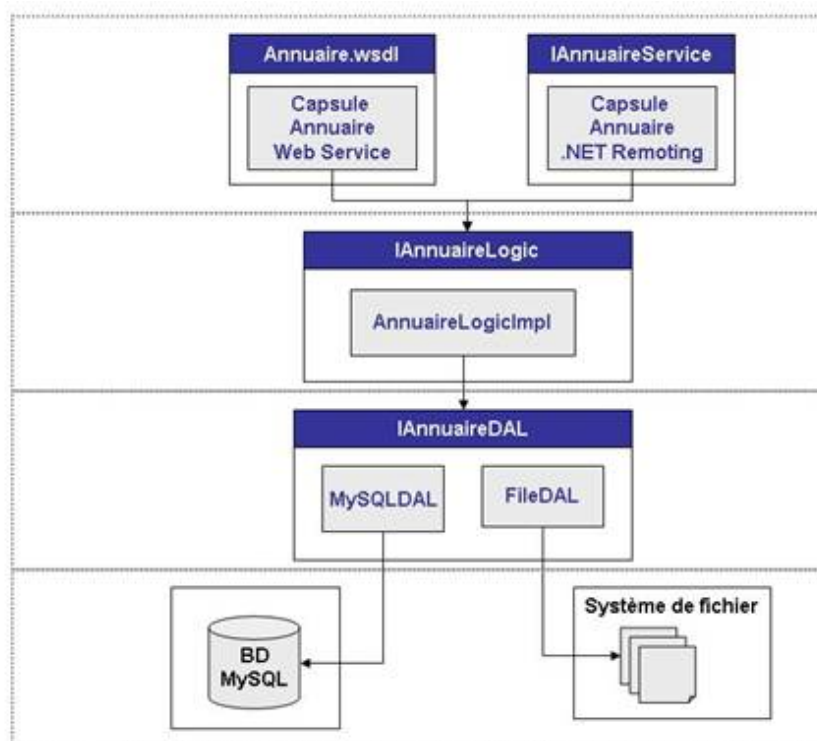


Figure 8.5 – Objets utilisés pour les différents appel à distance

8.12.1 .Net Remoting

Partie serveur

Pour exposer un service via le .Net Remoting il faut créer un conteneur (objet servant dans la terminologie de l'OMG) capable de distribuer cet objet. Pour cela nous allons construire un exécutable qui publiera l'objet sur le port 7000 via un canal TCP en mode singleton (un seul objet serveur actif à la fois partagé par tous les clients). On définit pour cela deux entités :

- `AnnuaireServiceRemoting` : l'objet singleton publié,

```

using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
4 using System.Runtime.Remoting.Channels.Tcp;
using MonPremierTP;

namespace MyAnnuaireRemoting
{
9   public class AnnuaireServiceRemoting: MarshalByRefObject,
                                   IAnnuaireService
   {
       IAnnuaireAdminService annuaire = new AnnuaireAdminService ();
       public CarteDeVisite Rechercher(string nom)
14   {
           return annuaire.Rechercher(nom);
       }
       public void Ajouter(CarteDeVisite carte)
19   {
           annuaire.Ajouter(carte);
       }
   }
}

```

Listing 8.15 – Fichier `AnnuaireServiceRemoting.cs`

- `Server` : le serveur qui va publier l'objet.

```

using System;
using System.Runtime.Remoting;
3 using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;
using MyAnnuaireRemoting;

public class Server {
8   [STAThread]
   static void Main(string[] args)
   {
       IChannel chan = new TcpChannel (7000);
       ChannelServices.RegisterChannel (chan);
13
       RemotingConfiguration.RegisterWellKnownServiceType(
           typeof (AnnuaireServiceRemoting),
           "TcpService",
           WellKnownObjectMode.Singleton);
18
   }
}

```

```

        Console.WriteLine("Press Enter to disconnect the annuaire.");
        Console.ReadLine();
    }
}

```

Listing 8.16 – Fichier `Serveur.cs`

TODO *.Net singletonne les objets distants tout seul ? Le pattern n'est pas implémenté dans la classe =; c'est bizarre de parler de singleton sans un constructeur protected, une instance statique de soi même et une opération 'GetSingleton' (if (this.MySelf == null) this.MySelf = new MySelf(); return this.MySelf;*

```

3 [csc ou gmcs] /t:exe /out:Serveur.exe /r:AnnuaireService.dll \
                                     /r:System.Runtime.Remoting.dll \
                                     AnnuaireServiceRemoting.cs Serveur.cs

WIN32> Serveur.exe
Press Enter to disconnect the annuaire.

8 MONO:~TP/> mono Serveur.exe
Press Enter to disconnect the annuaire.

```

Partie cliente

Dans le cas de `.Net Remoting`, le client a besoin de connaître uniquement l'interface du service à partir de laquelle un proxy sera dynamiquement construit. Sa construction est relativement simple :

```

1 using MonPremierTP;
  using System;
  using System.Runtime.Remoting;
  using System.Runtime.Remoting.Channels;
  using System.Runtime.Remoting.Channels.Tcp;
6
  class ClientRemoting
  {
    static void Main (string[] args) {
11      TcpChannel channel = new TcpChannel ();
      ChannelServices.RegisterChannel (channel);

      IAnnuaireService annuaire =
          (IAnnuaireService) Activator.GetObject(typeof (IAnnuaireService),
16          "tcp://127.0.0.1:7000/TcpService");
      CarteDeVisite c = annuaire.Rechercher ("MOSSER");
      Console.WriteLine ("1 email de MOSSER est {0}", c.Email);
    }
  }

```

Listing 8.17 – Fichier `ClientRemote.cs`

La classe `CarteDeVisite` n'est pas exposée automatiquement en `.Net Remoting`. Le client doit donc référencer l'assemblage la contenant.

Remarque : pour fonctionner, le client doit être lancé après le serveur ...

TODO on a uniquement besoin de l'interface et des types par de la dll entière - revoir l'exemple pour ne mettre que le minimum (et pas toute la DLL)

```

1 WIN32:~TP/> csc /r:AnnuaireService.dll ClientRemote.cs
WIN32:~TP/> ClientRemote.exe

MONO:~TP/> gmcs /r:System.Runtime.Remoting.dll
                                     /r:AnnuaireService.dll ClientRemote.cs
6 MONO:~TP/> mono ClientRemote.exe

```

8.12.2 Web Service

Partie serveur

La publication d'un service web nécessite l'écriture d'une page `.asmx` décrivant le code du service. Comme pour les pages `.aspx`, deux modes sont possibles : *code-behind* et *code-in line*. Voici ci-dessous le code de la page `.asmx` en mode *code-in line*.

```

<%@ WebService language="C#" class="AnnuaireWebService" Debug="true" %>

using System;
4 using System.Web.Services;
using System.Xml.Serialization;
using MonPremierTP;

public class AnnuaireWebService : System.Web.Services.WebService,
9                                     IAnnuaireService
{
    private IAnnuaireService service;
    public AnnuaireWebService()
    {
14         service=new AnnuaireService();
    }

    [WebMethod]
    public CarteDeVisite Rechercher(string nom)
19    {
        return service.Rechercher(nom);
    }
}

```

Listing 8.18 – Fichier `AnnuaireWebService.asmx`

Déploiement Que vous utilisiez le serveur intégré à Web Matrix ou celui de Mono, la page `.asmx` doit être déposé dans un répertoire du serveur Web et l'assemblage doit être déposé dans le sous-répertoire `bin`.

```

MONO:~/TP/www> tree .
3 .
  |-- bin
    |-- AnnuaireService.dll
  |-- <rep>
    |-- AnnuaireWebService.asmx

```

Utilisateurs Web Matrix Pour pouvoir exécuter un service web, il faut un serveur Web. Pour cela, nous allons utiliser Web Matrix¹⁴ qui permet d'activer un serveur Web interne. Le tutorial de Web Matrix (<http://rangiroa.essi.fr/cours/tutorial-webmatrix>) explique en détail la marche à suivre. Voici un court résumé :

1. Créer la classe `AnnuaireWebService` dans le fichier `AnnuaireWebService.asmx` (File, New File) puis 'General', 'XML Web Service', et finir de remplir la boîte de dialogue.
2. Un canevas de service web est créé. Il vous reste à compléter le code selon le modèle présenté ci-dessus pour qu'il accède à l'annuaire. Le code est presque le même que celui du servant dans le cadre du 'Remoting'. Seules les méthodes devant être accédées à distance sont précédées de l'attribut `[WebMethod]`.
3. Copier la DLL `AnnuaireService.dll` dans un répertoire `bin`.
4. Sauvegarder et exécuter le service web (appuyer sur la touche F5). Une boîte de dialogue vous demande le port d'activation du Web Service (8080 par défaut)
5. Un navigateur s'ouvre avec une page vous permettant d'interroger le service web :
 - (a) `http://localhost:8080/<rep>/AnnuaireWebService.asmx` pour avoir accès au service web,
 - (b) `http://localhost:8080/<rep>AnnuaireWebService.asmx?WSDL` pour avoir accès à la description WSDL de service.

Utilisateur de Mono Comme le serveur web de Web Matrix, XSP produit automatiquement un client permettant de tester le service qu'il expose. Pour y accéder, il suffit d'ouvrir son butineur préféré et de se rendre à l'adresse : `http://localhost:8080/AnnuaireWebService.asmx` (cf. figure 8.6). Comme pour tout serveur Web, la description WSDL est disponible à la même adresse en utilisant le suffixe `?WSDL`.

Partie Client

Dans le cas de la consommation d'un *service web*, il est nécessaire de construire au préalable un proxy C# du service web désiré. Celle utilise l'outil `wSDL`¹⁵ pour générer un proxy local au service distant que l'on souhaite invoquer (et qui doit être accessible ...).

```
wSDL2 /out:proxy_webservice.cs /n:proxy \
      http://localhost:8080/AnnuaireWebService.asmx?WSDL
```

Le code du client est le suivant :

```
using System;
class ClientWS
{
    static void Main (string[] args)
    {
        proxy.AnnuaireWebService annuaire = new proxy.AnnuaireWebService();
    }
}
```

¹⁴il est aussi possible d'utiliser le serveur IIS depuis Web Matrix

¹⁵la directive `/n` permet de préciser l'espace de nom. `wSDL` génère du code pour le framework 1.0 et `wSDL2` génère du code pour le framework 2.0

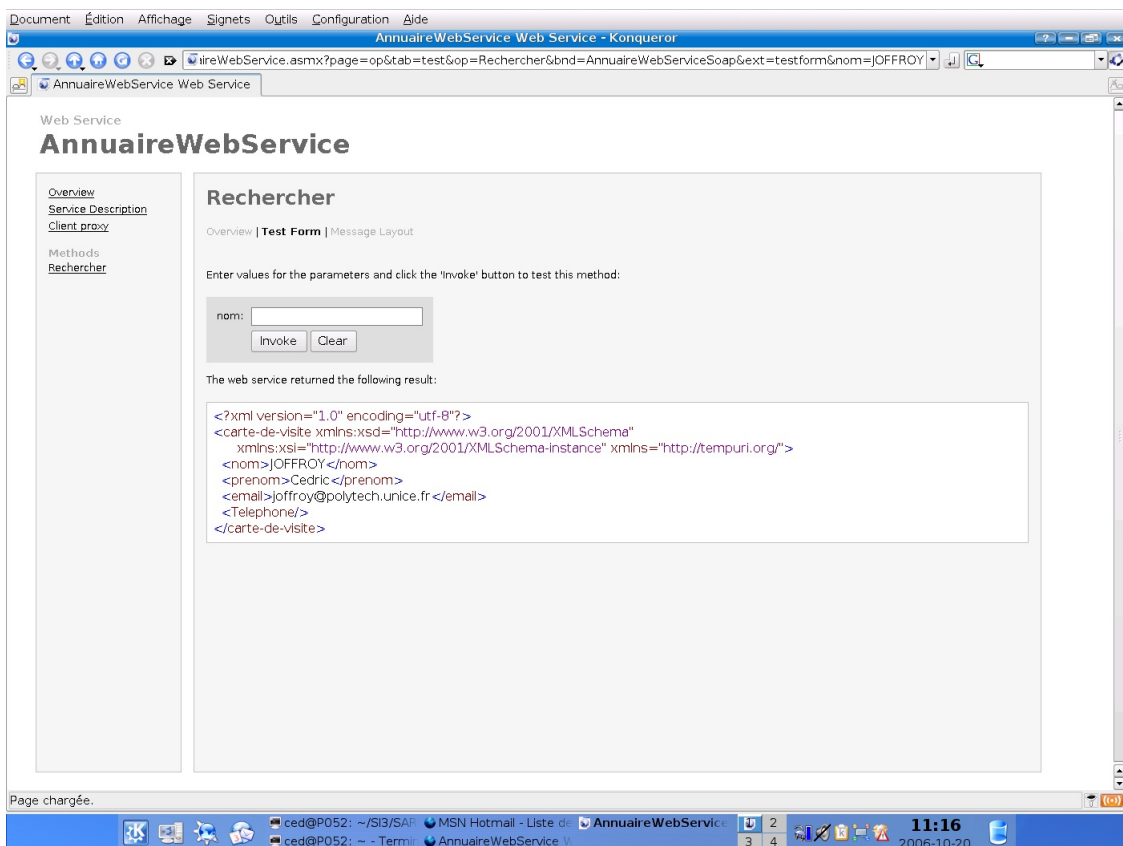


Figure 8.6 – Affichage de la pages ASMX dans un butineur

```

8 |     proxy.cartedevsite c = annuaire.Rechercher("MOSSER");
    |     Console.WriteLine ("1 email de MOSSER est {0}", c.email);
    | }
    | }

```

Listing 8.19 – Fichier ClientWS.cs

```

WIN32> csc /t:exe /out:ClientWS.exe ClientWS.cs proxy_webservice.cs
WIN32> ClientWS.exe
4 | MONO:~TP/> gmcs /t:exe /out:ClientWS.exe \
    |                               /r:System.Web.Services \
    |                               ClientWS.cs proxy_webservice.cs
MONO:~TP/> mono ClientWS.exe

```

8.13 Utilisation du code métier depuis un client léger

Nous arrivons bientôt à la fin de nos différents scénarios. La construction d'un client léger a déjà été vu dans la section 8.8 même si nous n'avions pas alors respecté l'architecture *n*-tiers de l'application. Pour respecter celle-ci, nous avons deux possibilités :

- créer un client léger qui appelle le service web préalablement créé en utilisant le proxy généré,
- créer un client léger qui utilise directement la DLL précédemment produite.

8.13.1 Appel d'un service web depuis la page ASP

La suite de cette section détaille la manière de créer un client léger qui appelle le service web préalablement créé, via le proxy. Nous détaillons successivement l'entête, la partie code et la partie html de cette page.

```

<%@ Page Language="C#" Debug="true" %>
<%@ assembly Src="proxy_webservice.cs" %>

```

Listing 8.20 – Fichier AnnuaireAspWs.aspx – partie entête

Remarque : pour pouvoir mettre au point le programme, l'entête de la page doit contenir la directive debug ainsi que le nom du fichier contenant le proxy du service web. Le code du proxy, comme celui de la page ASP .Net sera compilé dynamiquement.

```

<script runat="server">
    void Button_Click(object sender, EventArgs e)
3 | {
    |     // création du proxy et appel du service web
    |     proxy.AnnuaireWebService annuaire = new proxy.AnnuaireWebService ();
    |     proxy.cartedevsite carte = annuaire.Rechercher (nom.Text);
    |
8 |     // mise à jour des 'Labels' contenu dans la page
    |     LabelPrenomValeur.Text = carte.prenom;
    |     LabelEmailValeur.Text = carte.email;
    |
    |     // rendre visible les 'Labels' de la page
13 |     LabelPrenom.Visible = true;

```

```

    LabelPrenomValeur.Visible = true;
    LabelEmail.Visible = true;
    LabelEmailValeur.Visible = true;
}
18 </script>

```

Listing 8.21 – Fichier AnnuaireAspWs.aspx – partie code

La partie 'html' de la page contient un TextBox pour saisir le nom et quatre Labels : deux pour afficher le texte "prénom" et "email" et deux autres pour afficher les valeurs associées. Cette partie est bien entendue directement générée par WEB MATRIX à l'aide du mode 'dessin'.

```

<html>
2 <head>
  </head>
  <body>
    <form runat="server">
      <p>
7      <asp:TextBox id="nom" runat="server"></asp:TextBox>
      <asp:Button id="Button" onclick="Button_Click" runat="server"
        Text=" Search"></asp:Button>
      </p>
      <p>
12      <asp:Label id="LabelPrenom" runat="server" text="Prénom : "
        visible="False"></asp:Label>
      <asp:Label id="LabelPrenomValeur" runat="server" text="Label "
        visible="False"></asp:Label>
      </p>
17      <p>
      <asp:Label id="LabelEmail" runat="server" text="Email : "
        visible="False"></asp:Label>
      <asp:Label id="LabelEmailValeur" runat="server" text="Label "
        visible="False"></asp:Label>
22      </p>
    </form>
  </body>
</html>

```

Listing 8.22 – Fichier AnnuaireAspWs.aspx – partie html

Déploiement (toutes plates-formes) :

Quelque soit votre plate-forme de développement, le source C# du proxy et la page ASP doivent se trouver dans un répertoire accessible par le serveur web utilisé.

L'accès à la page se fait depuis votre butineur préféré à l'url <http://localhost:8080/AnnuaireAspWs.aspx>.

8.13.2 Appel du code métier contenu dans une DLL depuis la page ASP

Créer un client léger qui utilise directement la DLL précédemment produite. Ceci peut être fait très facilement en déplaçant la DLL dans un répertoire bin créé à partir du répertoire qui héberge la page. La suite présente successivement l'entête et le code contenu de la page ASP. La partie html de la page est identique à l'approche précédente et ne sera pas répétée.

```
<%@ import Namespace="MonPremierTP" %>
```

Listing 8.23 – Fichier AnnuaireAspD11.aspx -partie entête

Remarque : la directive import a un rôle équivalent au 'using' de C#. Si votre DLL utilise des modules, il faut aussi déployer dans ce répertoire les différents modules.

La partie html est évidemment la même. Le code, très voisin de l'étape précédente, est le suivant :

```
<script runat="server">
    void Button_Click(object sender, EventArgs e)
    {
4       IAnnuaireService annuaire = new AnnuaireService ();
        CarteDeVisite carte = annuaire.Rechercher (nom.Text);

        // mise a jour des 'Labels' contenu dans la page
        LabelPrenomValeur.Text = carte.Prenom;
9       LabelEmailValeur.Text = carte.Email;

        // rendre visible les 'Labels' de la page
        LabelPrenom.Visible = true;
        LabelPrenomValeur.Visible = true;
14       LabelEmail.Visible = true;
        LabelEmailValeur.Visible = true;
    }
</script>
```

Listing 8.24 – Fichier AnnuaireAspD11.aspx -partie code

Remarque : si l'entête ne possède pas la directive 'import', il est alors nécessaire de préfixer les noms de classes par le nom de leur namespace.

Déploiement Bien évidemment, la page .aspx doit être placée dans un répertoire accessible depuis le serveur web utilisé et la DLL contenant le code métier de l'application doit être présente dans le sous-répertoire 'bin' du répertoire pointé par le serveur web.

Il suffit ensuite d'afficher la page `http://localhost:8080/<repertoire>/AnnuaireAspD11.aspx` dans votre butineur préféré.

8.14 Bientôt de nouvelles rubriques...

- Déploiement et la gestion de versions
- Gestion des exceptions
- Utilisation de delegate et appel asynchrone avec call back
- Sauvegarde de la base dans un fichier XML et dans un fichier binaire
- Gestion des transactions
- Authentification et sécurité : en particulier faire en sorte que seul un administrateur possède puisse modifier la BD.