ROYAUME DU MAROC

مكتب التكوين المهني وإنعاش الشغل

Office de la Formation Professionnelle et de la Promotion du Travail

Développer un service Web

OFPPT

DIRECTION RECHERCHE ET INGENIERIE DE FORMATION

SECTEUR NTIC

Sommaire

1. Introduction

1.1. Présentation

Auparavant pour mettre en place des applications distribuées, il fallait utiliser des technologies assez complexes telles que COM. Certes ces technologies étaient abordables pour un développeur, mais il fallait que le développeur passe du temps à établir un protocole de transmission.

Les WebServices sont alors apparus pour faciliter tout d'abord la tâche des développeurs. Avant toute chose, Microsoft, contrairement aux idées reçues, n'a pas créé les WebServices mais Microsoft a participé avec de grandes entreprises telles que IBM, SUN … à la standardisation des WebServices.

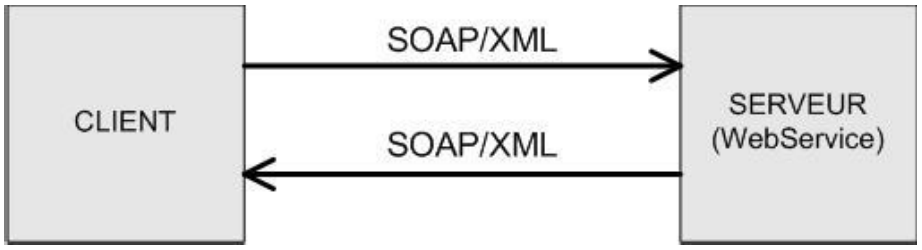
Ceci montre bien que la technologie des WebServices est une technologie très jeune, ce qui bien sûr peut être un inconvénient pour son intégration au sein des entreprises. Mais les plus grands spécialistes prévoient une « explosion » de l'utilisation des WebServices d'ici 2004 toutes technologies confondues (.NET, Java …).

1.2. Fonctionnement des WebServices

L'un des plus gros avantages des WebServices est qu'ils reposent sur des protocoles standardisés. Cela permet que cette technologie soit exploitable par de nombreux langages.

En effet, les WebServices se reposent sur des protocoles tels que XML et http, donc SOAP. Pour vulgariser ce dernier protocole, SOAP permet de faire circuler du XML via du HTTP. Donc lorsqu'on interroge un WebService, les données sont transmises en XML via le port 80 (HTTP). Rien de plus simple ensuite pour le développeur de traiter l'information reçue.

A l'heure actuelle, la quasi totalité des langages informatiques supporte ces protocoles : ils disposent en effet de fonctions pour lire un fichier XML (Parseur XML). Donc un WebService développé sous une plateforme .NET peut être utilisé via le langage Perl, PHP, Python, Delphi, Cobol …



1.3. Exemples d'utilisations

Nous allons voir à présent deux cas d'utilisation de WebServices sans rentrer dans les détails.

Pour notre premier exemple, prenons le cas simple d'une entreprise qui veut mettre en place un système de messagerie au sein de son Intranet, voir Internet par la suite. Elle pourra donc opter pour la technologie des WebServices en développant le Serveur de Messagerie en WebService .NET. L'avantage de cette solution et que l'application cliente pourra être développée sous n'importe quel

langage. L'entreprise pourra même développer plusieurs clients certains marchant sous Unix, d'autres sous Windows mais utilisant le même serveur de messagerie, c'est-à-dire le WebService qui se charge de la réception et de la transmission des messages. Dans notre cas, nous pourrons donc très bien avoir un client Windows qui communique avec un client Unix voir même un client Pocket PC !

Prenons à présent l'exemple d'une station de Météo qui veut revendre ses services. Elle veut par exemple partager une méthode avec une entreprise tierce : cette méthode permettrai de retrouver la température d'une région en entrant le Code Postal de celle-ci.
Rien de plus simple avec les WebServices ! La Station de Météo devra créer un WebService avec une
WebMethod (nous verrons la signification de ce mot dans le chapitre suivant) qui retournera un entier
(ou un float) suivant le Code Postal (donc un entier) qui sera transmis à la WebMethod.
Ensuite, si la station de Météo veut faire payer son Service, elle devra bien sûr protéger l'accès de la méthode par des moyens que nous aborderons par la suite.

2. Le protocole SOAP

Nous allons décrire dans ce chapitre le protocole SOAP et ses concurrents (COM, CORBA …). Nous détaillerons ensuite la structure d'un message SOAP pour bien comprendre comment les informations sont émises et reçues à partir d'un WebService. En effet, comme nous l'avons vu dans le chapitre précédent, la technologie des WebServices repose en autres sur le protocole SOAP.

2.1. Présentation

SOAP est un protocole adopté par le Consortium W3C. Le Consortium W3C crée des standards pour le Web : son but est donc de créer des standards pour favoriser l'échange d'information. Un standard veut tout simplement dire qu'il peut être accessible à tout le monde, et donc qu'il n'est pas propriétaire. Ce qui a pour conséquence qu'un protocole standard contrairement à un protocole propriétaire pourra être utilisé sous n'importe quelle plateforme.
Les spécifications du protocole SOAP sont disponibles à l'adresse suivante : http://www.w3.org/TR/SOAP/

SOAP veut dire : Simple Object Access Protocol. Si l'on voulait traduire cette définition en français cela donnerait Protocole Simple d'Accès aux Objets. En effet, le protocole SOAP consiste à faire circuler du XML via du http sur le port 80. Cela facilite grandement les communications, car le XML est un langage standard et le port utilisé est le port 80, qui ne pose donc pas de problèmes pour les firewalls de l'entreprise, contrairement à d'autres protocoles.

Tout comme la technologie des WebServices, le protocole SOAP est très jeune. Le protocole SOAP a été crée en septembre 98, avec la version 0.9, par trois grandes entreprises : Microsoft, UserLand et DevelopMentor. IBM n'a participé au protocole SOAP qu'à partir de la version 1.1 en avril 2000. C'est cette même année que SOAP a été soumis au W3C.
Depuis septembre 2000, SOAP 1.1 est en refonte complète pour donner jour à la version 1.2 avec un groupe de travail de plus de 40 entreprises ! Parmi ces 40 entreprises, on retrouve bien sûr Microsoft, IBM mais aussi HP, Sun, Intel …)

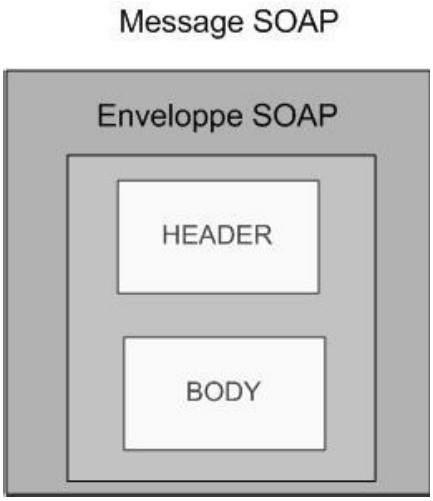
2.2. Messages SOAP

2.2.1. Structure d'un message SOAP

La syntaxe d'un message SOAP est décrite sur le site officiel W3C traitant du protocole SOAP :
http://www.w3.org/TR/SOAP/

Tout d'abord un message SOAP est un document XML qui doit avoir la forme suivante :

- Une déclaration XML qui n'est pas obligatoire
- Une enveloppe SOAP qui est composée de :
- Un en-tête SOAP (HEADER)
- Un corps SOAP (BODY)



2.2.2. Messages SOAP dans l'exemple

Voyons à présent un exemple concret d'un message SOAP de requête et le message de réponse.
Dans notre exemple, nous allons prendre une méthode qui retourne le nombre entré en paramètre. Ce type de méthode n'a bien sûr aucun intérêt dans la pratique, mais plus la méthode est simple, mieux vous comprendrez la structure d'un message SOAP.

Voici la signature de notre méthode :
GetNombre (Nombre as integer) as integer

La structure du message de requête sera du type :

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/ xmlns:SOAP-
ENC=http://schemas.xmlsoap.org/soap/encoding/
xmlns:xsi=http://www.w3.org/1999/XMLSchema-instance
xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<SOAP-ENV:Body>
    <ns1:GetNombre
     xmlns:ns1="urn:MySoapServices">
          <param1 xsi:type="xsd:int">10</param1>
     </ns1:GetNombre>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Comme vous pouvez le voir, la première ligne du message SOAP contient une déclaration XML qui n'est pas obligatoire. L'enveloppe SOAP est ensuite déclarée via la balise <SOAP-ENV:Envelope>. Cette enveloppe est composée d'un corps (<SOAP-ENV:Body>).
Dans le corps de ce message, vous pouvez très bien voir notre méthode « GetNombre » et son paramètre qui est égale à 10 dans notre exemple.

Voyons à présent le message de réponse :

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
xmlns:xsi=http://www.w3.org/1999/XMLSchema-instance
xmlns:xsd="http://www.w3.org/1999/XMLSchema">

<SOAP-ENV:Body>
    <ns1:GetNombreResponse
    xmlns:ns1="urn:MySoapServices"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
          <return xsi:type="xsd:int">10</return>
   </ns1:GetNombreResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Le message de réponse a la même structure que celle du message envoyé. Ceci veut dire qu'il contient une déclaration XML, ainsi qu'une enveloppe SOAP composée d'un corps.
Dans le corps du message de réponse, nous pouvons toujours voir notre méthode « GetNombre » : le mot « Response » a été rajouté à la fin du nom de la méthode pour bien préciser qu'il s'agit du retour d'une requête sur la méthode.
La valeur renvoyée par la méthode « GetNombre » est égale à 10.
Dans notre exemple, l'enveloppe SOAP n'était pas composée d'un en-tête : en effet, l'en-tête d'un message SOAP est optionnelle et est utilisée pour transmettre des données d'authentification ou de gestion de sessions. Nous verrons plus loin en détail l'utilité de cet en-tête.

L'exemple que nous venons de voir est un exemple très simple. Pour décrire aux mieux le protocole SOAP, il faudrait voir des exemples avec des types plus complexes. Pour cela nous vous renvoyons sur le site officiel du protocole SOAP :
http://www.w3.org/TR/SOAP/

2.3. Sécurité et SOAP

La sécurité est un des grands défauts du protocole SOAP. En effet, lors de la création du protocole SOAP, des groupes ont voulu faire pression pour insérer dans le protocole SOAP des mécanismes de sécurité. Mais le projet a été abandonné, car le protocole SOAP devait rester facile à mettre en œuvre. De plus, comme vous avez pu le voir dans notre section précédente, les messages SOAP peuvent être lus sans problèmes et il suffit donc qu'une tierce personne lise les messages SOAP émis et reçus par le WebService pour avoir l'intégralité des informations sans aucune difficulté.

Le cryptage devient donc nécessaire pour des services transmettant des données sensibles. Le protocole HTTPS, qui propose un chiffrement jusqu'à 128 bits, pourra donc être utilisé pour mettre un place un système de communication sécurisé.

Nous verrons dans un chapitre ultérieur les différents moyens qui peuvent être mis en œuvre pour mettre en place un système d'information sécurisé

3. WSDL et WebMethod

Avant de passer à la pratique, nous allons voir ensemble quelques termes techniques dont vous devrez connaître la signification avant de commencer à réaliser votre premier WebService.

3.1. WSDL

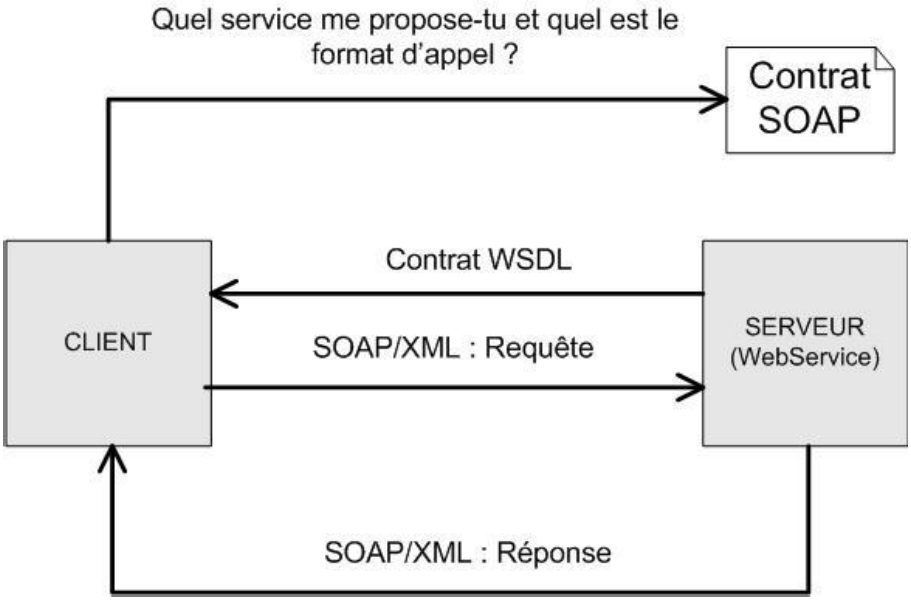
3.1.1. Présentation

Il faut savoir avant toute chose que chaque WebService possède un fichier de spécification : c'est ce qu'on appelle la WSDL (Web Services Description Language). C'est à partir de ce fichier que le développeur pourra savoir comment interroger le WebService, quels sont ses différentes WebMethods, quels types ces WebMethods retournent t-elles …

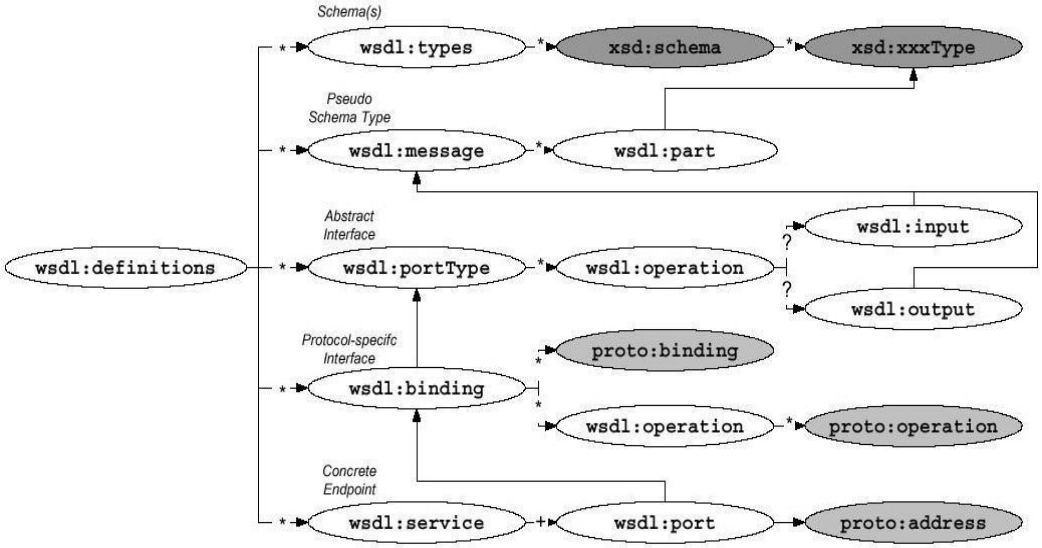
La WSDL décrit en fait tout le fonctionnement d'un WebService et il est donc indispensable pour pouvoir utiliser un WebService. Ce fichier est

structuré en XML et il est assez complexe quand le développeur ne connaît pas XML. Mais nous verrons par la suite que le Framework génère automatiquement la WSDL d'un WebService ce qui facilite grandement la tâche du développeur.

Enfin, pour que votre application puisse utiliser un WebService, il faut qu'elle dispose d'une classe Proxy appelé Proxy Web qui sera créée à partir du Contrat du WebService, c'est-à-dire le fichier de description WSDL.
Le Proxy va aider le client à savoir où trouver le WebService. Le Proxy contiendra également les détails des communications avec le WebService.
Nous verrons ultérieurement que l'utilitaire « wsdl.exe » fourni avec le Framework SDK .NET
permettra de générer notre classe Proxy à partir d'un fichier de description WSDL.



3.1.2. Schéma fichier WSDL



3.2. WebMethod

Nous verrons dans le chapitre suivant que lorsque nous devons créer un WebService se nommant par exemple « MyWebService » nous devons créer une classe « MyWebService ». En effet tout est objet en .NET et tout code doit se trouver dans une classe.

Comme vous devez le savoir, une classe dispose de valeurs ainsi que de méthodes. Ceci ne change pas avec les WebServices, mais pour qu'une méthode soit utilisable via votre WebService, il faut donner l'attribut « WebMethod » à cette même méthode.

Examinons à présent le code suivant :

```
<WebMethod()> _
  Public Function GetName(Prenom as String) As String
      Return "Hello World"
  End Function

    Private Sub NoReturn()

  End Sub
```

Nous avons ici deux méthodes (Nous n'avons pas représenter ici la classe). La première méthode « GetName » possède l'attribut « WebMethod ». La méthode GetName est donc une WebMethod qui retourne un type « string » et qui prend type « string » comme argument.

Attention : Toutes WebMethods doivent être déclarées en public et ne peuvent pas être « private » ou « protected », sinon la WebMethod ne pourra pas être atteinte depuis l'extérieur.
La deuxième méthode n'est pas une WebMethod car elle ne possède pas l'attribut « WebMethod » et ne pourra donc pas être atteinte via le WebService.

4. WebService avec Visual .NET

4.1. Avantages de Visual .NET

Utiliser Visual Studio .NET facilite le travail du développeur et rend plus productif le projet. En effet, lorsque vous commencez un nouveau WebService, Visual Studio .NET vous crée tous les fichiers nécessaires au bon fonctionnement de votre WebService et fournit quelques exemples de code.

Il est vrai que Visual .NET n'est pas donné du point de vue financier, mais il y a une autre alternative qui est elle entièrement gratuite. Cette solution s'appelle « WebMatrix » et c'est l'éditeur d'applications Web et WebService de Microsoft. « WebMatrix » est bien sûr beaucoup plus léger que Visual .NET et n'apporte donc pas toutes ses fonctionnalisés.

4.2. Création d'un WebService

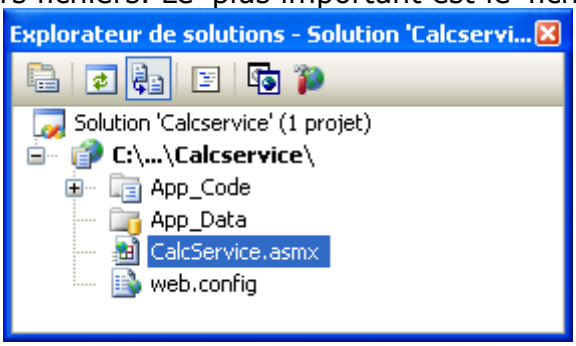
Nous allons apprendre a créer un WebService en utilisant Visual Studio .NET et qu'on nommera CalcService.

4.2.1. Utilisation de l'interface de VS

Lancez Visual Studio .NET.
Une fois Visual lancé,
Cliquez sur : File -> New web site -> ASP.NET Web Service.

Choisissez son emplacement et le langage

Dans le « solution explorer » de votre projet, vous pouvez voir que Visual vous a créé plusieurs fichiers. Le plus important est le fichier «*.asmx » :



Vous pouvez alors éditer votre WebService
Les autres fichiers et dossiers vous sont connus, comme vous avez déjà sûrement programmé des applications Web ou Windows avec .NET, mais en voici une rapide description :

- **CalcService.asmx** : C'est le fichier qui représente votre WebService.
- **Web.config** : C'est le fichier de configuration de votre WebService.
- **App_Code** : C'est le dossier qui contient les classes métier .
- **App_Data** : C'est le dossier qui contient les classes d'accès aux données.

4.2.2. Accès au code d'un WebService

Vous êtes à présent dans l'éditeur de code de votre fichier « CalcService.asmx », c'est-à-dire le WebService modifier ce code pour avoir le contenu suivant.

```
Imports System.Web
Imports System.Web.Services
Imports System.Web.Services.Protocols

<WebService(Namespace:="http://tempuri.org/")> _
<WebServiceBinding(ConformsTo:=WsiProfiles.BasicProfile1_1)> _
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
Public Class CalcService
    Inherits System.Web.Services.WebService

<WebMethod()> _
Public Function Ajouter(ByVal valeur1 As Integer, ByVal valeur2 As
Integer) As Integer
        Return valeur1 + valeur2
End Function
<WebMethod()> _
Public Function Soustraire(ByVal valeur1 As Integer, ByVal valeur2 As
Integer) As Integer
        Return valeur1 - valeur2
    End Function
End Class
```

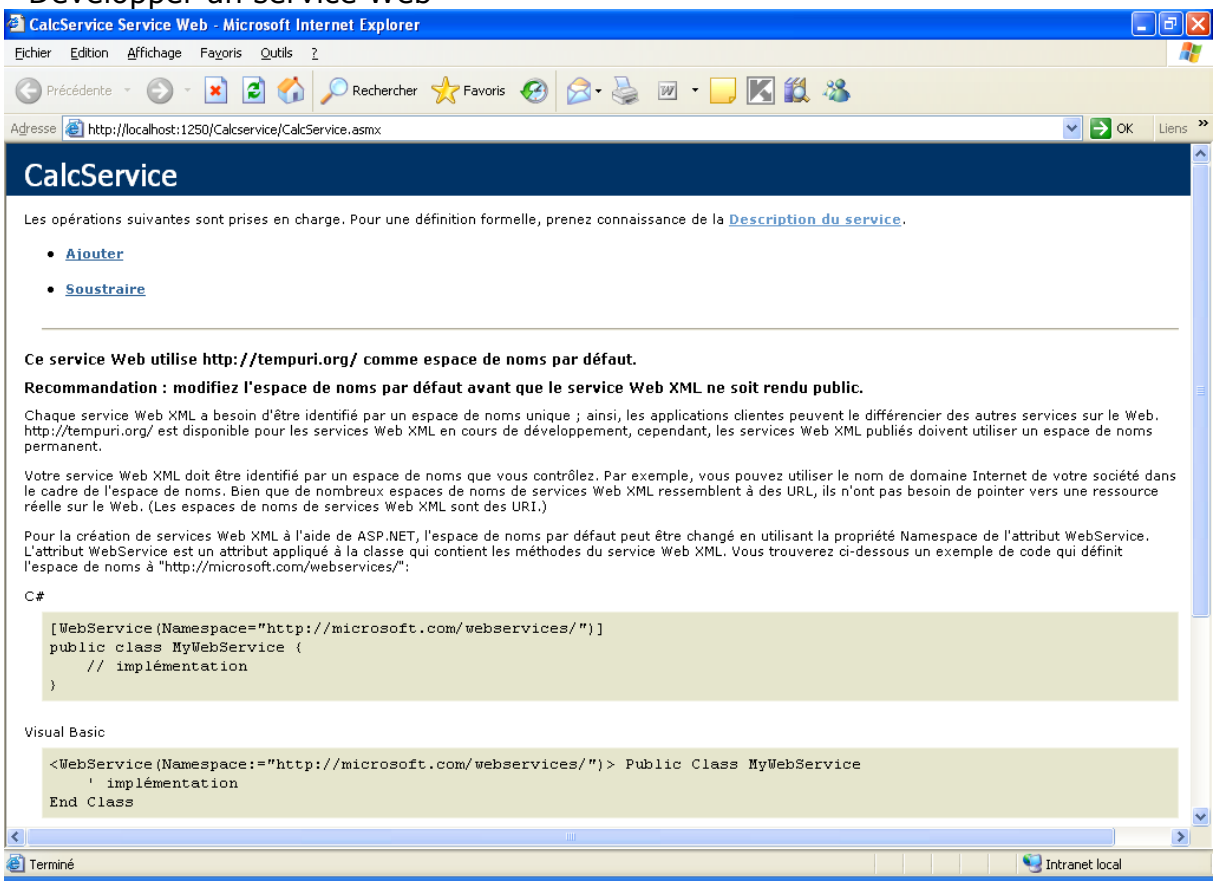
Vous avez à présent créé votre premier WebService avec Visual Studio .NET. Nous allons à présent voir comment nous pouvons tester notre WebService, atteindre la WSDL, et enfin créer une application utilisant notre WebService.

5. Test d'un WebService

5.1. Présentation Page de test

Une fois notre WebService fini, nous pouvons le tester immédiatement, car le Framework génère une page qui permet de tester toutes ses méthodes. Pour accéder à cette page, il vous suffit d'entrer le chemin Web de votre fichier .asmx dans Internet Explorer ou autres… Si vous utilisez Visual Studio
.NET, vous pouvez directement avoir accès à votre WebService en appuyant sur la touche « F5 ». Visual va alors compiler votre application et lancer Internet Explorer.

Vous tomberez alors sur une page de ce type :



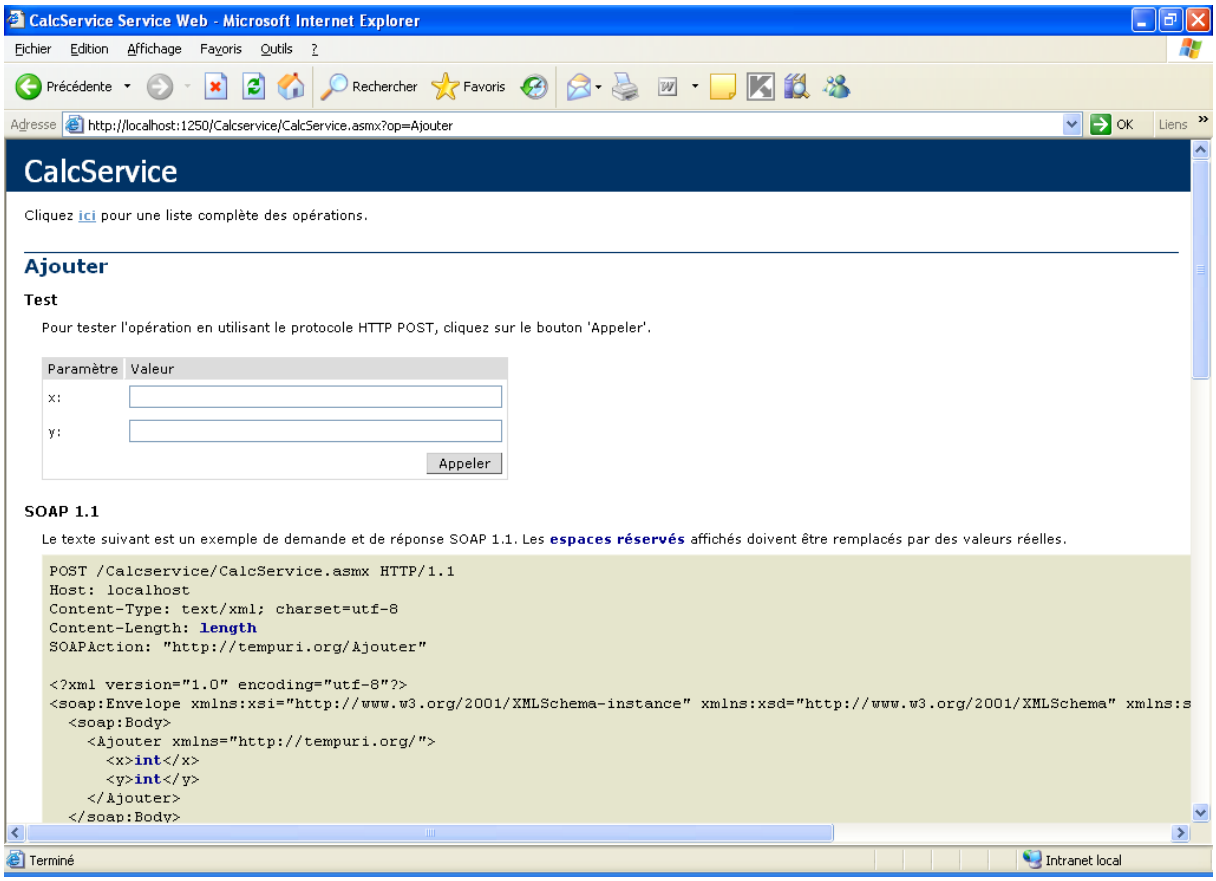
Notez tout d'abord que la page vous indique que votre WebService a un namesapce par défaut « http://tempuri.org ». Nous vous montrerons dans une section ultérieure comment changer le namespace de votre WebService.

Les deux méthodes de notre WebService, « Ajouter » et « Soustraire », sont présentes sur la page en tant qu'hyperlien. Vous allez pouvoir, en cliquant par exemple sur « Ajouter », tester la méthode et voir quelle méthode elle retourne.

5.2. Analyse de la méthode « Ajouter »

5.2.1. Test de la méthode

Je vous propose donc de cliquer sur « Ajouter » pour tester et analyser la méthode « Ajouter ». Vous devez alors avoir la page suivante qui apparaît:



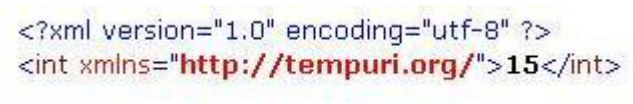
Note : Nous n'avons pas affiché pour l'instant les informations qui se trouvent en bas, car nous parlerons de ces informations juste après le test de la méthode « Ajouter ».

Nous retrouvons sur la page d'information de la méthode « Ajouter » ses deux paramètres :

- valeur1
- valeur2

Il y a aussi un bouton « Invoke » qui va vous permettre d'appeler la méthode avec la valeur des paramètres que vous aurez insérés des les textbox appropriées.
Entrons par exemple la valeur « 10 » pour « valeur1 » et « 5 » pour « valeur2 ». Nous ne le savons pas encore mais théoriquement, si notre méthode fonctionne normalement, la méthode « Ajouter » devrait retourner la valeur « 15 ».

Pour savoir si votre méthode « Ajouter » retourne un résultat cohérent, cliquez sur le bouton « Invoke ».



La valeur retournée est bien sûr envoyée sous format XML et nous pouvons voir que cette valeur est bien égale à « 15 ». Notre méthode « Ajouter » a donc bien fonctionné.

5.2.2. Différents protocole d'accès :

Retournons à présent sur la page d'information de la méthode « Ajouter ». Vous pouvez voir en dessous du bouton « Invoke » plusieurs informations qui sont réparties en trois chapitres qui représentent en fait les différents protocoles d'accès au WebService. Nous n'avons vu jusqu'à présent qu'un seul protocole d'accès à un WebService, le protocole SOAP. Il existe en fait trois protocoles d'accès :

- SOAP
- HTTP GET
- HTTP POST

5.2.3. SOAP

Le premier protocole traité par la page d'information de la méthode « Ajouter » est le protocole SOAP que nous avons détaillé dans un chapitre précédent.

Comme le montre la figure de la page suivante, vous pouvez voir deux messages. Le premier est le message de requête et le deuxième est le message de réponse.
Les deux messages sont bien constitués d'une enveloppe SOAP et d'un corps appelé « BODY ».

Le corps du message de requête contient bien :

- La méthode « Ajouter » : <Ajouter xmlns="http://tempuri.org/">
- Le premier paramètre : <valeur1>**int**</valeur1>
- Le deuxième paramètre : <valeur2>**int**</valeur2>

Le corps du message de réponse contient :

- La valeur de retour : <AjouterResult>**int**</AjouterResult>

SOAP

The following is a sample SOAP request and response. The **placeholders** shown need to be replaced with actual values.

```
POST /CalcService/CalcService.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.w3.org/
  <soap:Body>
    <Ajouter xmlns="http://tempuri.org/">
      <valeur1>int</valeur1>
      <valeur2>int</valeur2>
    </Ajouter>
  </soap:Body>
</soap:Envelope>
```

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

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

5.2.4. HTTP GET

Nous pouvons en effet interroger notre WebService directement à partir de son adresse HTTP et en ajoutant les paramètres nécessaires à la méthode :

HTTP GET

The following is a sample HTTP GET request and response. The **placeholders** shown need to be replaced with actual values.

```
GET /CalcService/CalcService.asmx/Ajouter?valeur1=string&valeur2=string HTTP/1.1
Host: localhost
```

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<int xmlns="http://tempuri.org/">int</int>
```

Comme indiqué dans la figure précédente, il nous suffit d'entrez l'adresse «http://localhost/CalcService/CalcService.asmx/Ajouter?valeur 1=10&valeur2=5 »
pour pouvoir interroger notre méthode :


```
<?xml version="1.0" encoding="utf-8" ?>
<int xmlns="http://tempuri.org/">15</int>
```

5.2.5. HTTP POST

Vous pouvez accéder au WebService via une simple requête POST à partir d'un formulaire :

HTTP POST

The following is a sample HTTP POST request and response. The **placeholders** shown need to be replaced with actual values.

```
POST /CalcService/CalcService.asmx/Ajouter HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded
Content-Length: length

valeur1=string&valeur2=string
```

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<int xmlns="http://tempuri.org/">int</int>
```

5.3. WSDL

Vous pouvez accéder au fichier de description de votre WebService en ajoutant « ?WSDL » à l'adresse initiale de votre WebService :

Address http://localhost/CalcService/CalcService.asmx?WSDL

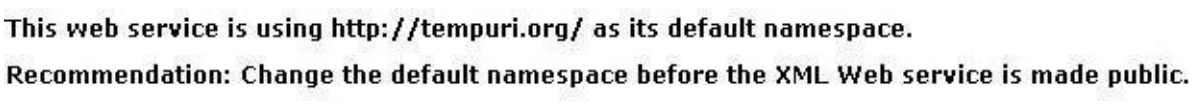
```
<?xml version="1.0" encoding="utf-8" ?>
- <definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:soap="http://schemas
    xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:s0="http://tempuri.org/" xmlns:soape
    xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/" xmlns:mime="http://schemas.x
    xmlns="http://schemas.xmlsoap.org/wsdl/">
  - <types>
    - <s:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/">
      - <s:element name="Ajouter">
        - <s:complexType>
          - <s:sequence>
              <s:element minOccurs="1" maxOccurs="1" name="valeur1" type="s:int" />
              <s:element minOccurs="1" maxOccurs="1" name="valeur2" type="s:int" />
            </s:sequence>
          </s:complexType>
        </s:element>
      - <s:element name="AjouterResponse">
        - <s:complexType>
          - <s:sequence>
              <s:element minOccurs="1" maxOccurs="1" name="AjouterResult" type="s:int" />
            </s:sequence>
          </s:complexType>
        </s:element>
```

Nous n'avons bien sûr pas affiché tout le fichier, car un fichier WSDL est très long même pour un
WebService très simple.
Mais la complexité d'un fichier WSDL ne gêne en aucun cas le développeur, car ce fichier de description est automatiquement généré par le Framework.
La WSDL permettra donc aux développeurs de savoir comment utiliser le WebService, c'est-à-dire quelles sont ses méthodes, comment les interroger …

5.4. Personnalisation du WebService

5.4.1. Changement du namespace

Si vous n'avez pas attribué de namespace à votre WebService, un namesapce par défaut lui sera attribué et la page d'accueil de test vous conseillera alors de donner un nom à votre WebService :



Vous devez alors attribuer un namepsace à votre WebService. Pour notre exemple, nous le nommerons
« http://www.OFPPT.org.ma ».
Pour arriver à cela, il vous suffit d'ajouter l'attribut suivant à la classe de votre WebService :

```
[WebService(Namespace=" http://www.OFPPT.org.ma ")
```

5.4.2. Description

Vous pouvez ajouter une description à vos méthodes et à votre WebService. Ces descriptions seront alors incluses dans le fichier de description WSDL et affichées sur la page de test du WebService.

Pour ajouter une description de la méthode « Ajouter », il vous suffit de détailler l'attribut
« WebMethod » comme ceci :

```
[WebMethod (Description="Description de la méthode …")]
public int Ajouter(int valeur1,int valeur2)
...
```

6. Créer une application cliente

Nous allons dans une chapitre créer une application Windows utilisant notre WebService - CalcService –

6.1. Création de l'application

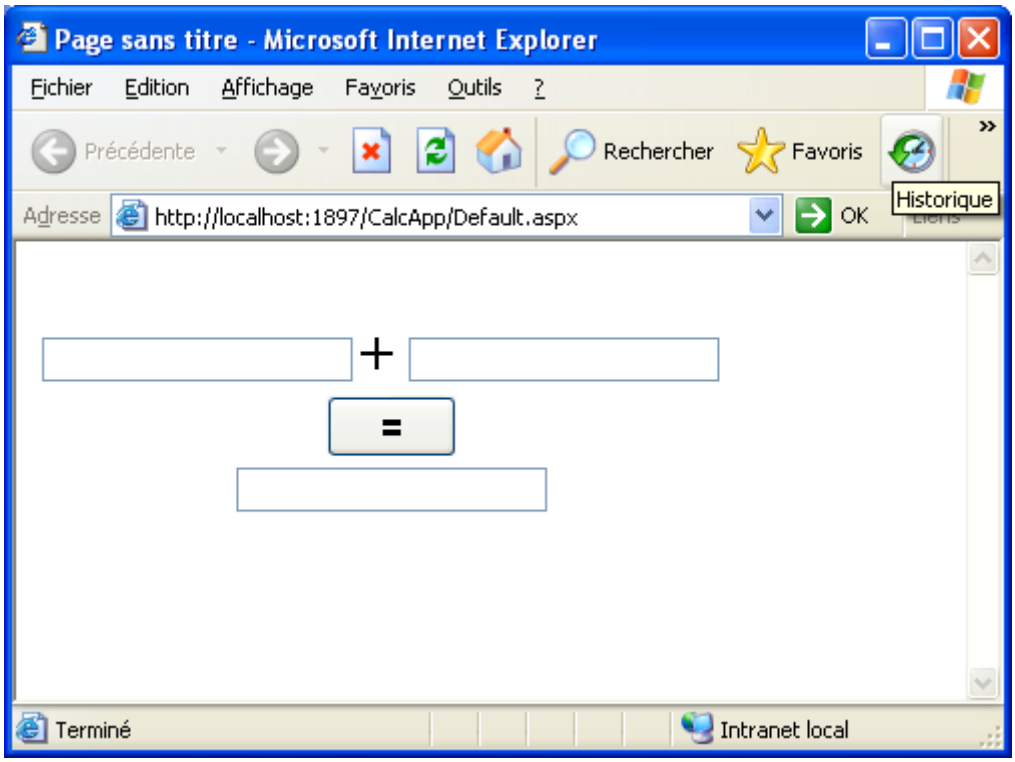
Nous allons créer une application Web « CalcApp » qui utilisera la méthode « Ajouter » de notre WebService.

6.2. Interface

Pour notre exemple, nous allons réaliser une simple Web Form contenant :

- « txtValeur1 » : Textbox contenant la valeur du premier paramètre de la méthode « Ajouter ».
- « txtValeur2 » : Textbox contenant la valeur du deuxième paramètre de la méthode « Ajouter ».
- « txtResultat » : Textbox contenant le résultat qui sera envoyé par la méthode « Ajouter ».
- « btnAjouter » : Bouton permettant d'appeler la méthode « Ajouter ».

L'interface reste simple comme vous pouvez le voir :



6.3. Classe Proxy

6.3.1. Présentation

Vous voulez à présent utiliser le WebService « CalcService » dans votre application Windows. Il faut vous créer une classe Proxy qui va vous permettre de faire la relation entre votre application et le WebService. Pour générer votre classe Proxy, vous avez deux solutions :

- WSDL.EXE
- Solution intégrée de Visual .NET « Add Web Reference ».

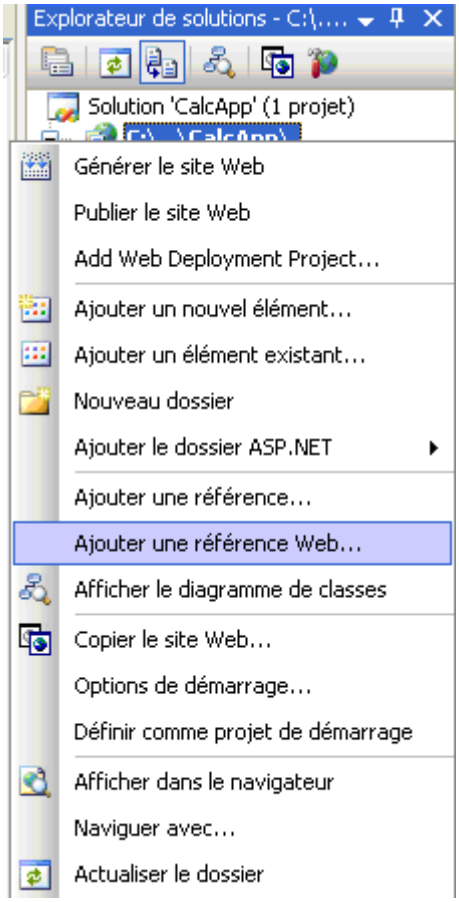
L'utilitaire « wsdl.exe », qui est fournit avec le Framework SDK, va vous permettre de générer votre classe Proxy à partir d'un fichier de description WSDL. La solution intégrée de Visual .NET utlise elle aussi « wsdl.exe », mais vous évite de passer en ligne de commande pour générer votre classe

Proxy. En effet, Visual .NET va générer automatiquement tous les fichiers nécessaires pour que votre application puisse communiquer avec le WebService.

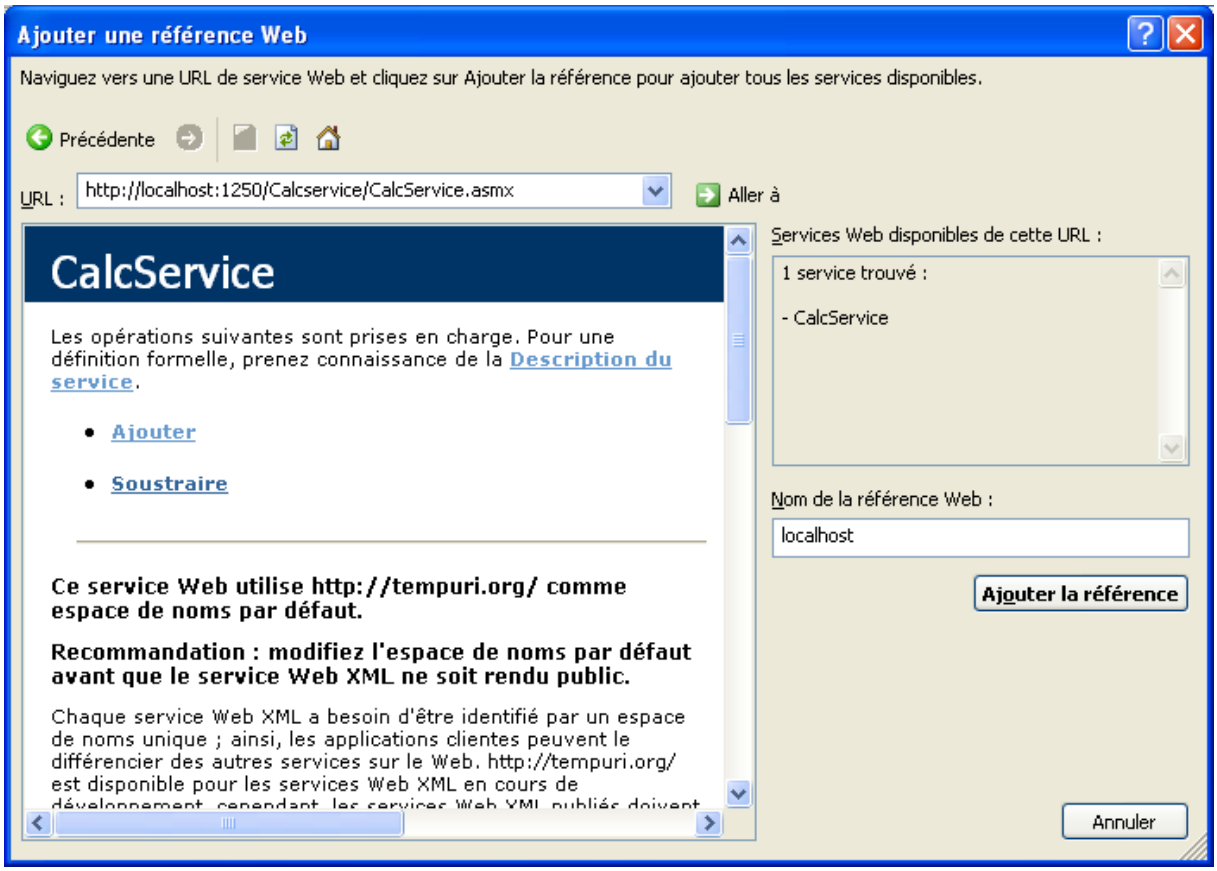
Nous verrons dans le chapitre suivant comment utiliser l'utilitaire « wsdl.exe » en détail…

6.3.2. Génération de classe Proxy

Nous allons donc utiliser Visual Studio .NET pour générer la classe Proxy du WebService « CalcService ».
Pour cela, allez dans le « Solution Explorer » et ajouter une référence Web :



Un Explorateur va alors s'afficher.
Entrez l'adresse du WebService et cliquez sur « Ajouter une Référence Web » :



Visual.NET va alors créer plusieurs fichiers dans un dossier «App_WebReferences »
Notre WebService sera accessible dans le namepsace «localhost».
Dans le dossier « localhost », nous trouvons trois fichiers :

- «CalcService.wsdl» : Fichier de description du WebService « CalcService ».
- «CalcService.disco» et «CalcService.discomap» : définissant les références aux WebServices

Appel au WebService
Nous voulons à présent appeler la méthode « Ajouter » lorsqu'on clique sur le bouton « = » de l'interface.
Voici les différentes étapes :

- Créer une nouvelle méthode et associer cette méthode à l'événement « Click » du bouton.
- Créer une instance de la classe « CalcService » qui représente notre WebService. Attention ! Comme indiqué ci-dessus, par défaut le WebService se trouvera dans le namesapce « localhost »
- Appeler la méthode « Ajouter » à partir de l'instance que vous venez de créer et insérer le résultat dans une TextBox.

Et voici le code correspondant à l'événement « Click » du bouton :

```
Protected Sub btnAjouter_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Button1.Click
  'On crée une instance de la classe CalcService
  Dim myWebService As New localhost.CalcService()
  'On appelle la méthode "Ajouter"
  'et on insère le résultat dans txtResultat
  txtResultat.Text =
myWebService.Ajouter(Integer.Parse(txtValeur1.Text),
Integer.Parse(txtValeur2.Text)).ToString()
End Sub
```

L'application est à présent terminée et nous pouvons la tester :