Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

Travail de diplôme 2008

Filière Informatique de gestion

WSDL to XAML

Etudiant : Olivier Vionnet

Professeur : David Russo
Michael Schumacher

www.hevs.ch

Sommaire

1. Introduction

Fin 2006, Microsoft a sorti la version 3.0 du framework .Net suivi une année plus tard de la version 3.5. Dans la version 3.0, on note entre autres l'arrivée de WPF (Windows Presentation Foundation) qui est le nouveau système destiné à l'interface utilisateur qui utilise le langage de description XAML et WCF (Windows Communication Foundation) qui est le nouveau système orienté service pour communiquer entre différents composants applicatifs.

Lors de la sortie du framework 3.0, il n'était pas très facile de développer des web services WCF sous Visual Studio 2005 patché pour l'occasion, les outils à disposition étant un peu lacunaire.
L'arrivée de Visual Studio 2008 a permis de résoudre ce problème en proposant un projet pour créer un web service WCF et une meilleure implémentation des aides au développement ce qui permet au final au développeur de gagner du temps dans le développement de son web service.

Malgré toutes les améliorations apportées dans Visual Studio 2008, il manque une fonctionnalité qui serait utile : la possibilité de développer rapidement un client pour le web service développé. En effet, il est toujours à la charge du développeur de devoir créer un client pour tester son web service, le projet WCF Service Application ne générant pas de client prêt à l'emploi.
Au mieux, il propose une page web indiquant comment créer un client en utilisant l'outil svcutil.exe pour créer un fichier de configuration et un fichier de code contenant la classe du client ainsi que le code à implémenter pour appeler le service.

Le développeur doit ensuite créer un projet, intégrer les différents fichiers cités ci-dessus, s'occuper de créer l'interface utilisateur pour pouvoir saisir facilement des données, développer le traitement au niveau du code pour utiliser le web service ce qui comprend la récupération des saisies utilisateurs, la création de différents objets nécessaires avec les données de l'utilisateur afin d'avoir à disposition toutes les informations structurées comme l'attend l'opération du web service, traiter le retour du web service en décomposant les objets en types standards pouvant être affiché dans l'interface utilisateur.

Si le développeur souhaite tester le web service en utilisant différents jeux de valeurs, il faudra qu'il les teste un à un ou bien qu'il développe un nouveau projet spécifique pour le traitement qu'il souhaite réaliser.

Au final, le client demande du temps pour être développé. Si en plus, le web service contient plusieurs opérations, le temps s'en trouve rallongé.

C'est pour résoudre ce problème que ce travail de diplôme a été réalisé. Il faut offrir au développeur un outil qui soit paramétrable et qui lui permette de créer une application cliente rapidement pour tester le web service qu'il a créé.

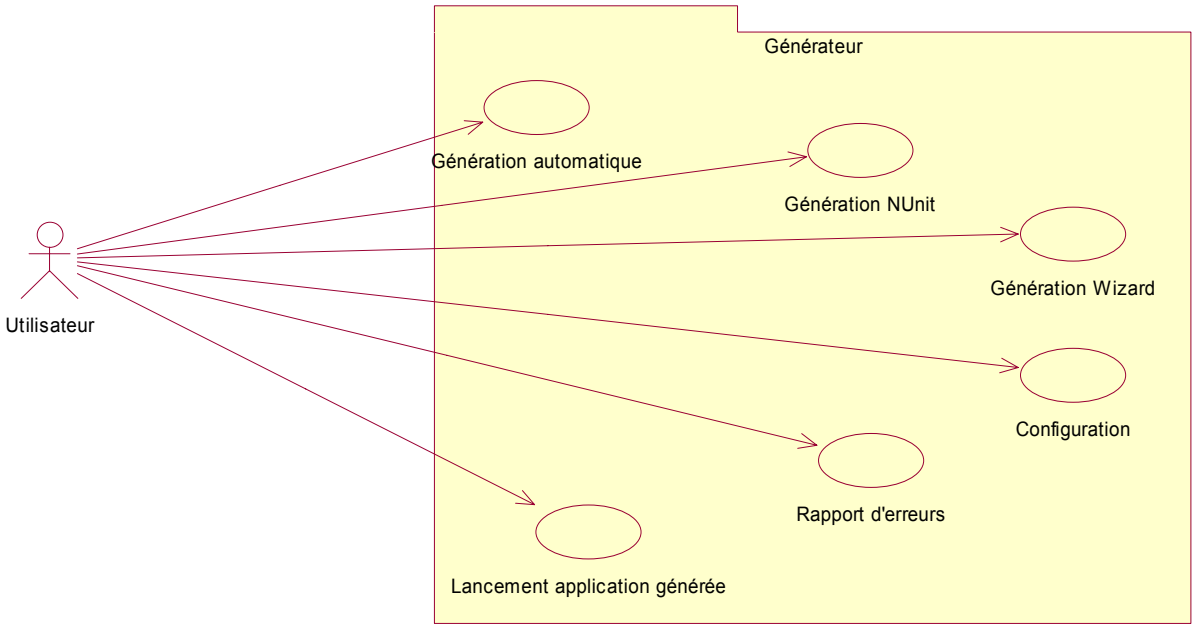
Cet outil doit s'occuper de générer une application cliente en se basant sur le document WSDL du web service afin de récupérer toutes les informations nécessaires telles que les opérations disponibles, les types présents,…, de générer le code XAML pour construire l'interface graphique et le code C# pour les traitements à effectuer et de compiler le tout en un exécutable. Le développeur n'aura alors plus qu'à utiliser l'application générée pour tester son web service.

Pour rendre cet outil encore plus attractif auprès du développeur et lui faciliter son travail, l'application se doit de proposer différents moyens de générer un client :

- De manière automatique, sans que l'utilisateur n'aie besoin de spécifier quoi que ce soit lors de la génération du client.
- Dans un style wizard où le développeur pourra choisir quelles opérations et quels contrôles XAML il souhaite utiliser lors du processus de génération du client.
- De type test où le développeur pourra rentrer des tests unitaires qui seront exécutés automatiquement et le résultat du traitement effectué clairement affiché.

Au final, cet outil réduira sensiblement le temps que le développeur devra consacrer à la création d'un client pour tester son web service et lui permettra également de faire des tests unitaires facilement et rapidement.

2. Fonctionnalités de l'application



2.1. Génération automatique

Cette fonctionnalité permet de générer sans intervention humaine un client pour un web service. Cette option est utilisée pour obtenir très rapidement un client et pouvoir tester rapidement le web service.
Toutes les opérations trouvées sont implémentées et les UserControls par défaut sont utilisés. Lorsque la génération est achevée, l'application affiche dans la fenêtre principale un bouton pour lancer l'exécutable créé.

2.2.Génération wizard

Cette fonctionnalité permet de générer un client en permettant à l'utilisateur d'effectuer des choix lors de la génération. Cette fonctionnalité est utilisée pour les cas où il y a beaucoup d'opérations disponibles et qu'on ne souhaite traiter que certaines ou encore pour utiliser d'autres UserControls que les UserControls standards afin de simplifier l'utilisation pour tester le web service.
Une fois la génération commencée, l'application demande à l'utilisateur quelles opérations il souhaite intégrer au client.
La génération continue le traitement pour préparer les différents UserControls et une fois terminé, l'application demande à l'utilisateur de choisir les UserControls qu'il souhaite utiliser. C'est ici que l'utilisateur pourra opter pour certains types des UserControls plus adaptés pour l'affichage et l'utilisation du client.
Une fois le choix fait, l'application termine la génération et affiche un bouton pour lancer l'exécutable créé.

2.3.Génération tests Nunit

Cette fonctionnalité permet de générer un client destiné à tester le web service avec un grand nombre de données différentes sans devoir à chaque fois rentrer ces données à la main.
Cette fonctionnalité est utilisée pour les cas où il y a beaucoup de données à tester et que l'utilisation d'un client graphique où il faut rentrer chaque donnée à la main prendrait trop de temps.
L'application s'occupe de construire le client en mode console et demande à l'utilisateur de rentrer la série de tests unitaires qu'il souhaite effectuer. Une fois les tests rentrés, la génération de l'application cliente est effectuée et les tests sont exécutés en utilisant NUnit. Le résultat est affiché sous forme de tableau en couleur de manière à identifier rapidement les erreurs qui se seraient produites.

2.4.Configuration

Cette fonctionnalité permet de paramétrer le générateur. Son contenu est divisé en 4 parties :
1ère : tout ce qui concerne les dossiers et chemin d'accès aux différents outils utilisés par l'application ainsi que les noms que différents fichiers doivent prendre et finalement l'url du WSDL du web service.
2ème : Les assemblies qui doivent être utilisées lors d'une génération automatique ou wizard
3ème : Les assemblies qui doivent être utilisées lors d'une génération pour des tests NUnit.
4ème : La possibilité de sauver les paramètres de l'application et de charger une nouvelle configuration sauvée (pratique si l'on développe plusieurs web services et que l'on souhaite travailler en parallèle.

2.5.Rapport d'erreur

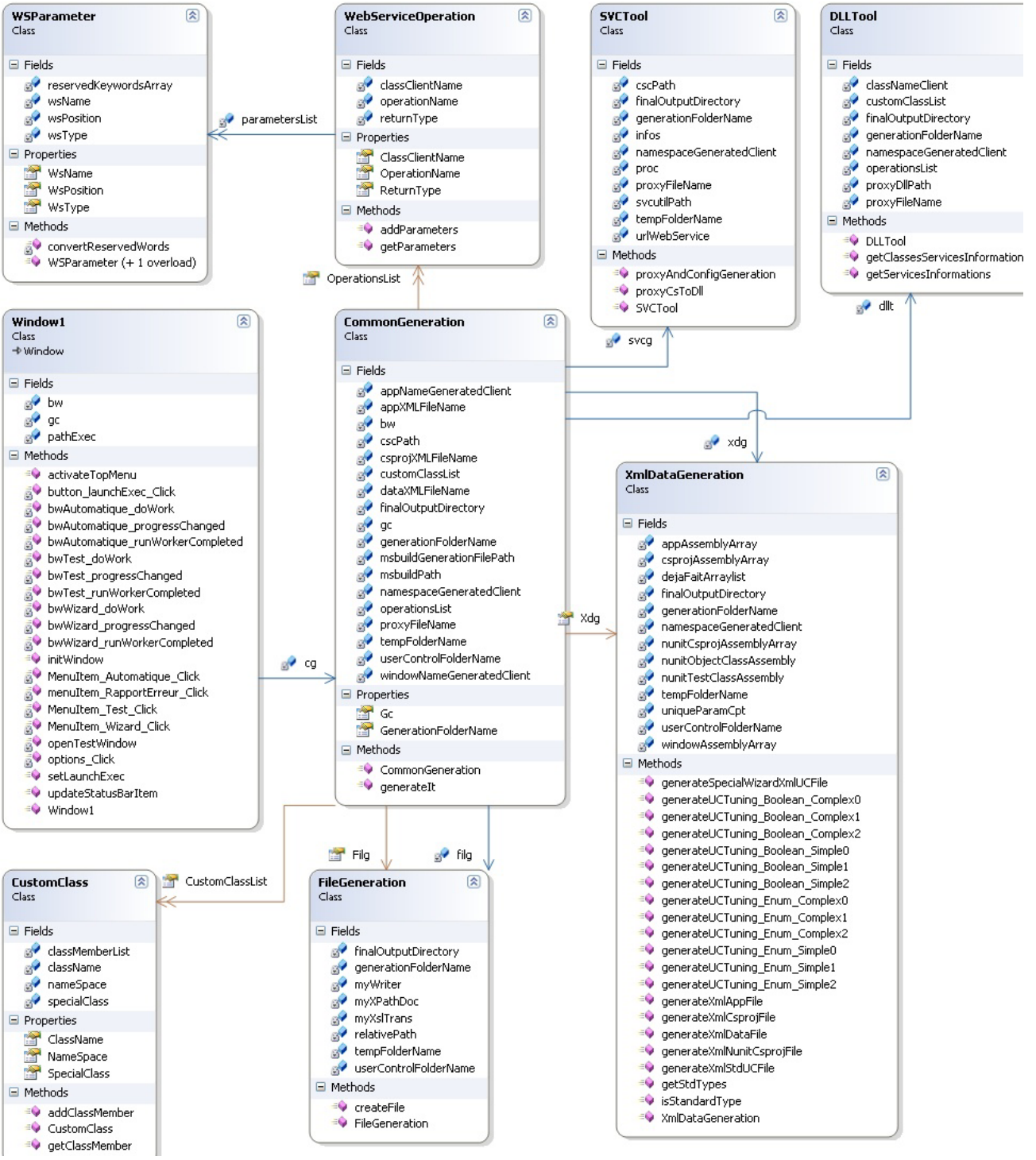
Lors d'une génération, il est possible de rencontrer des erreurs. Cette fonctionnalité permet de consulter le fichier de logs pour la dernière génération effectuée sur les erreurs qui seraient survenues.

2.6.Lancement d'une application générée

Dans le cas d'une génération réussie, cette fonctionnalité affiche un bouton permettant de lancer le client sans devoir parcourir l'arborescence de fichiers à la recherche du dernier client généré.

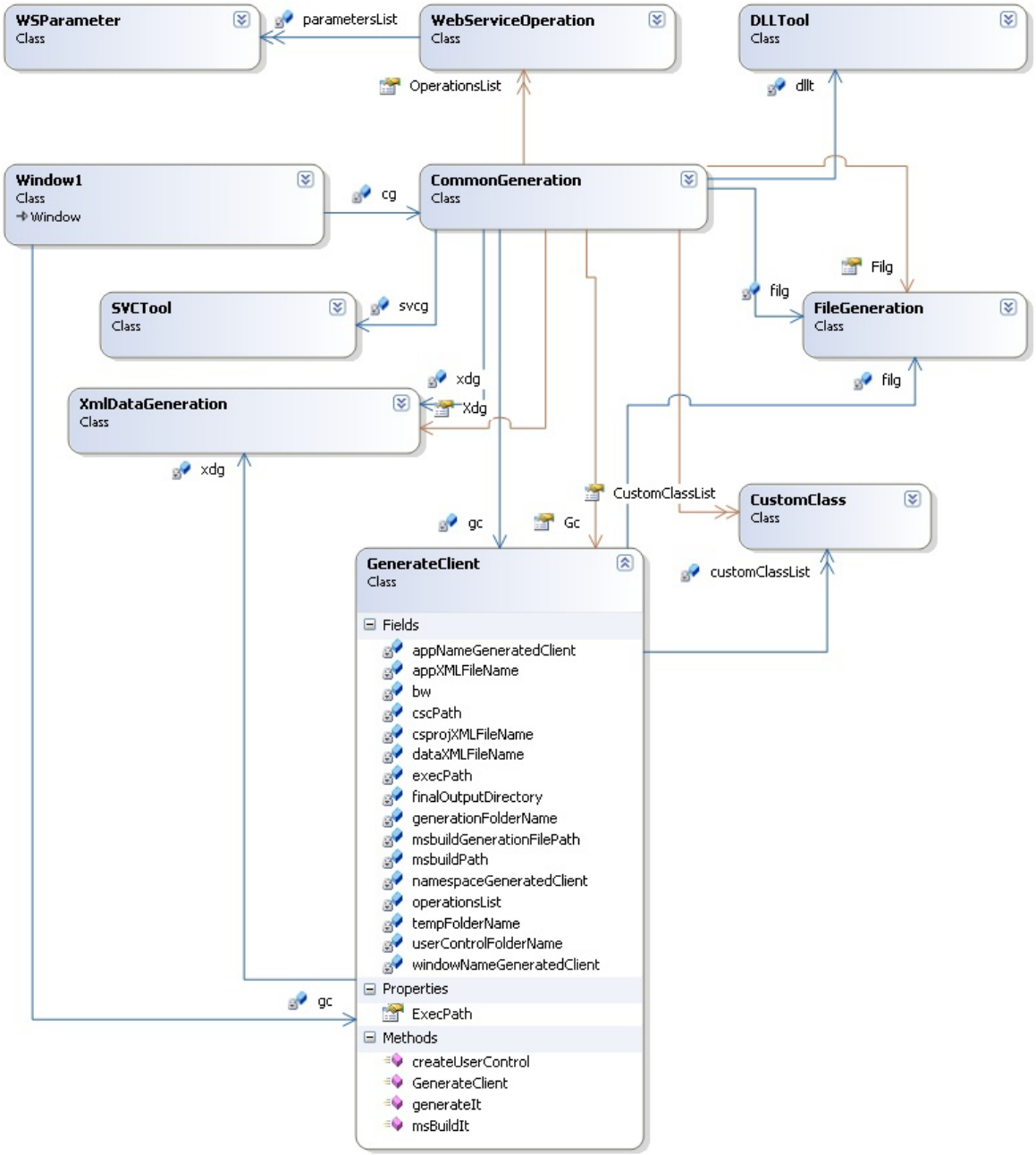
3. Architecture de l'application

3.1. Génération commune



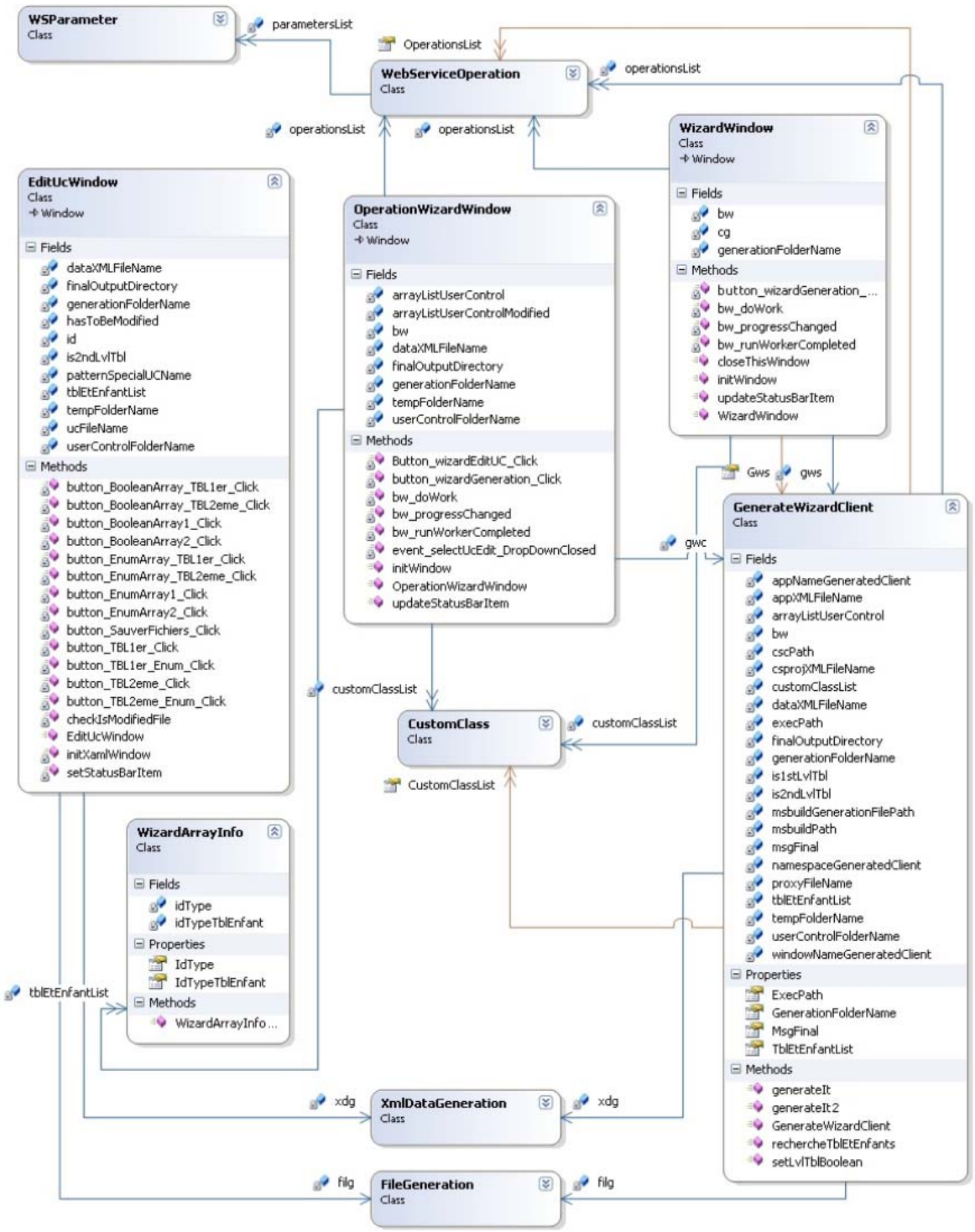
Les 3 différentes générations ont une partie commune pour récupérer et traiter les informations du WSDL. A la fin du traitement de la méthode generateIt, un switch permet de poursuivre le traitement spécifique de chaque génération.

3.2.Génération automatique



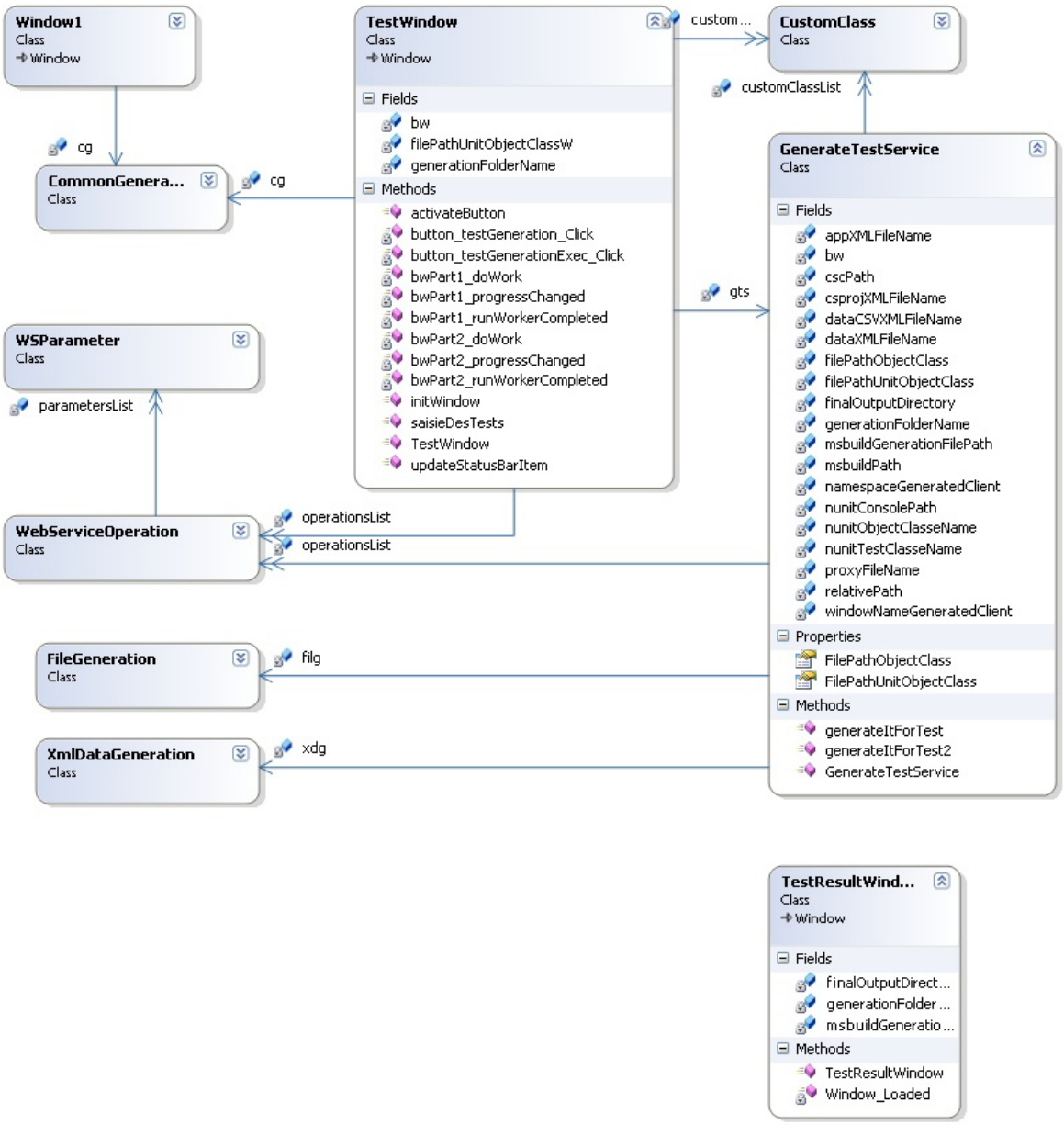
Une fois le traitement commun terminé, le traitement se poursuit dans GenerateClient pour construire l'application.

3.3.Génération wizard



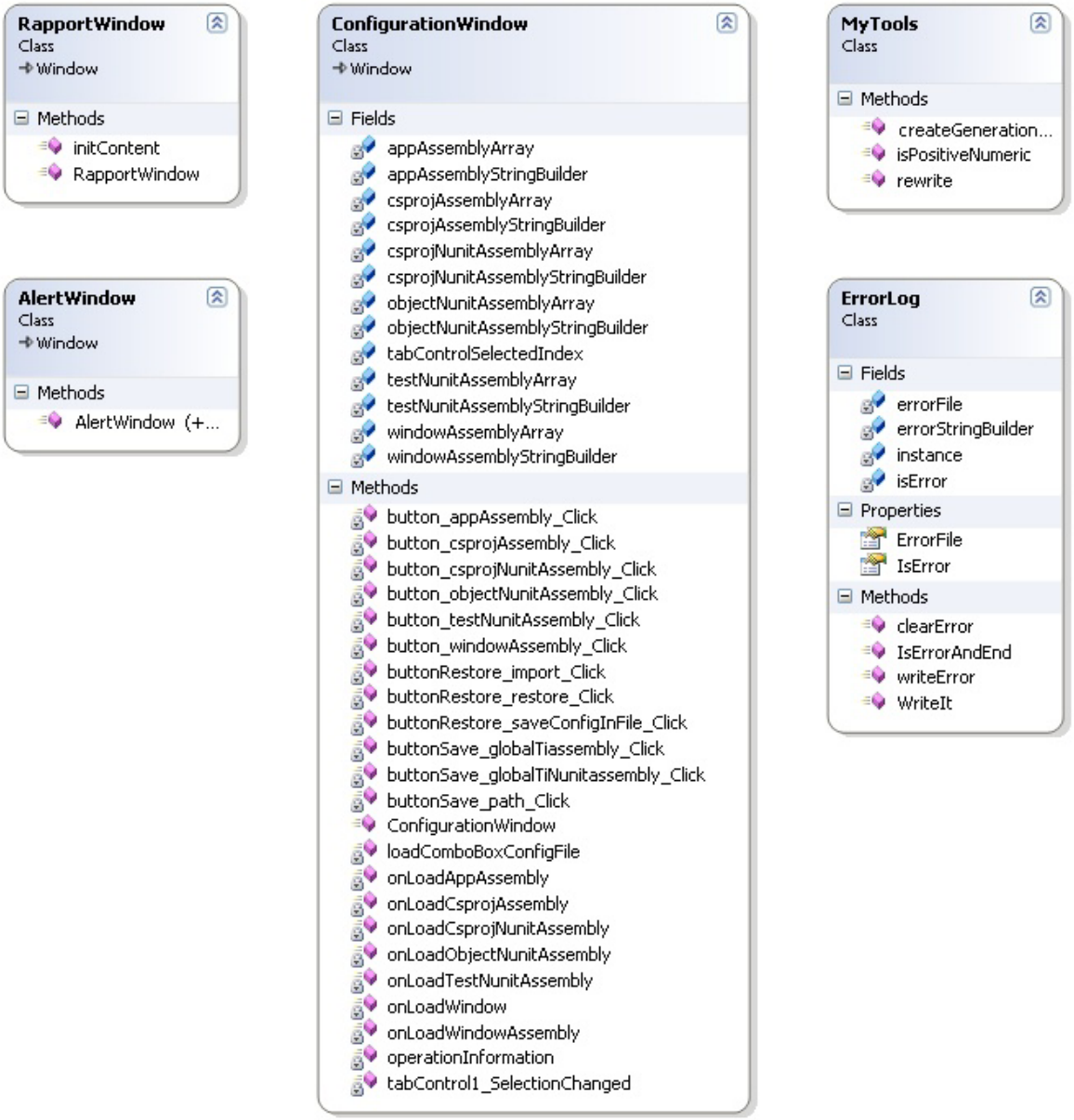
Une fois le traitement commun terminé, WizardWindow est instancié afin de choisir les opérations à traiter puis le traitement est continué dans la classe GenerateWizardClient avec la méthode generateItForTest pour la création des UserControls.
Ensuite, OperationWizardWindow est appelé pour afficher pour chaque opération les userControls disponibles, EditUcWindow permettant d'éditer le code des userControls.
Une fois le choix validé, le traitement se poursuit avec la méthode generateItForTest qui s'occupe de construire l'application.

3.4.Génération tests Nunit



Une fois le traitement commun terminé, TestWindow est instancié pour dans un premier temps choisir les opérations à utiliser. La méthode generateItForTest de GenerateTestService s'occupe de créer les fichiers de l'application et test. Ensuite, le contenu de TestWindow est modifié pour afficher le contenu des fichiers créés afin de saisir les tests puis la méthode du generateItForTest2 est appelée pour générer l'application, lancer les tests et finalement, afficher le résultat dans TestResultWindow.

3.5. Configuration et autres éléments



Cette partie s'occupe de regrouper les différentes classes restantes.
ConfigurationWindow et rapportWindow peuvent être appelés par Window1. AlertWindow se s'affiche qu'en cas d'erreur de traitement dans la fenêtre de configurationWindow.

3.6.Documentation sur contenu des classes C#

La documentation du code C# de l'application se trouve sur le cd dans le fichier documentation\documentation du code csharp\WpfClientGen_codeDoc.chm.

3.7.Documentation sur contenu des fichiers de transformation

La documentation concernant les fichiers de transformation de type XSLT n'étant pas prise en charge par SandCastle, elle est décrite ci-après.
Avant de regarder les différents fichiers de transformation, il faut parler de la façon dont je procède pour échapper certains caractères qui peuvent poser problème dans une transformation.

Très souvent, les fichiers de transformation contiennent des < ou > au lieu d'avoir < ou > ceci afin d'éviter que la transformation prenne en compte certains éléments (souvent certains éléments sont interprétés lors de la transformation et génère pour les balises qui sont à l'intérieur le nom du namespace ce qui provoque ensuite des erreurs pour compiler le fichier avec msbuild) ou bien si le résultat de la transformation est un template qui devra être une nouvelle fois transformée.
A noter que certains éléments sont encapsulés dans un <![CDATA[ ]]> toujours dans le but d'éviter d'avoir des problèmes avec les namespaces.
Lors de la première transformation, le CDATA n'est pas pris en compte, car ce n'est pas un élément connu dans XSLT à cause de l'ouverture et la fermeture de la balise (seul l'ouverture et la fermeture de la balise est transformée, on obtient après avoir transformé le contenu suivant <![CDATA[ ]]> qui sera un élément connu lors de la 2ème transformation).
Lors de la deuxième transformation, CDATA est connu et tout son contenu ne sera pas transformé

Exemple :

Avant 1ère transformation	Après 1ère transformation	Après 2ème transformation
<![CDATA[</UserControl>]]>	<![CDATA[</UserControl>]]>	</UserControl>

Pour que cet élément soit compris par le framework lors du build de l'application, il ne restera plus qu'au niveau de la programmation de parcourir l'entier du fichier du UserControl pour remplacer les < et > par < et >.

3.7.1. UserControls

La création des UserControls pour la génération automatique et wizard utilise le même fichier de transformation mais différents templates.

3.7.1.1. UC_xaml.xslt

Ce fichier de transformation s'occupe de générer la partie XAML de l'UserControl. Il est composé 5 templates :

root
Ce template est utilisé lorsqu'il faut créer un UserControl standard (la majorité des cas). Il est composé d'un Label pour afficher les informations du type, le nom et id du membre, d'un TextBox avec une couleur différente suivant le type et d'un Label d'erreur caché.

rootBoolean, rootBooleanArray, rootEnum, rootEnumArray
Ces templates sont destinés à la création de UserControls spécifiques (dans la partie wizard). La logique est commune au 4 templates.
La première étape consiste à écrire les informations de base tel que le contenu x:Class et les différents contrôles qui sont toujours identiques. Tout le reste contenu dans le template est destiné à la 2ème transformation.
La deuxième étape consiste à écrire les contrôles spécifiques suivants ce que l'utilisateur aura fourni comme information concernant l'emplacement du type. Il y a 3 cas de figures possibles : le type n'est pas contenu dans un tableau de type personnalisé ou il est contenu dans 1 ou 2 tableaux de type personnalisé. Suivant ce qu'il aura précisé (et qui aura été transposé dans le fichier de

données XML), le template écrira le bon nombre de contrôles en respectant la taille des tableaux fournis.

3.7.1.2. UC_xaml_cs.xslt

Ce fichier de transformation s'occupe de générer le code C# du UserControl. Pour pouvoir utiliser ce fichier de transformation aussi bien pour la génération automatique que wizard et que cela reste compatible ensuite avec le fichier de transformation AllWindow_xaml_cs2.xslt, les méthodes des UserControls sont identiques.
De plus, pour obtenir ou inscrire une valeur se fait toujours avec un String, le code de traitement de l'UserControl s'occupant de transformer si nécessaire le String pour remplir et afficher correctement les contrôles.
Pour chaque UserControl, il y a toujours les mêmes méthodes présentes :
- getUCValue() qui permet de récupérer la valeur du contrôle.
- setUCValue(String str) qui permet d'affecter une valeur au contrôle.
- cleanUCValue() qui permet d'effacer le contenu du contrôle.

Pour la création des UserControls spécifiques (Boolean ou Enum), il y a toujours 2 transformations.
La première transformation permet de configurer le nom de classe et namespace tandis que la deuxième transformation s'occupe d'implémenter les différentes méthodes avec le code de traitement approprié suivant que le type traité se trouve dans 0, 1 ou 2 tableaux de type personnalisé.
Tout comme le fichier de transformation UC_xaml.xslt, il est composé de 5 templates.

root
Ce template est utilisé lorsqu'il faut créer un UserControl standard. Il s'occupe d'écrire le bon namespace, si nécessaire d'importer le type et d'écrire le nom de classe du contrôle.

Il y a également un événement sur la perte de focus du contrôle. Cet événement est uniquement présent pour montrer qu'il est possible de contrôler des valeurs et d'afficher un message d'erreur le cas échéant.
Toutes ces méthodes n'ont pas besoin d'être modifiée pour fonctionner avec tous les types.

rootBoolean
Ce template est destiné à la création de UserControls spécifiques pour le type Boolean. Au lieu d'afficher un TextBox, il affiche un CheckBox.

- getUCValue() :

Aucun tableau d'objets personnalisés qui est parent

```
<!-- get Boolean -->
  &lt;xsl:for-each select="zeroDimension/input"&gt;
  public String getUCValue()
  {
    String returnValue = "";
    &lt;xsl:for-each select="BooleanC"&gt;
      returnValue += &lt;xsl:value-of select="@nameElement"/&gt;.IsChecked  ;
      return returnValue;
    &lt;/xsl:for-each&gt;
  }
  &lt;/xsl:for-each&gt;
```

Chaque for-each n'est exécuté qu'une fois vu que le type actuellement traité n'a pas de parent qui serait un tableau personnalisé.

1 tableau d'objets personnalisés qui est le parent

```
<!-- get Custom[] -> Boolean -->
  &lt;xsl:for-each select="uneDimension/input"&gt;
  public String getUCValue()
  {
    String returnValue = "";
    &lt;xsl:for-each select="BooleanC"&gt;
      returnValue += &lt;xsl:value-of select="@nameElement"/&gt;.IsChecked + ";;"
;
      &lt;xsl:if test="position() = last()"&gt;
        returnValue = returnValue.Substring(0, returnValue.Length - 2);
      &lt;/xsl:if&gt;
    &lt;/xsl:for-each&gt;
    return returnValue;
  }
  &lt;/xsl:for-each&gt;
```

Le premier for-each est exécuté une fois. Ensuite, pour chaque élément du tableau parent, la valeur du CheckBox est récupérée, stockée dans un String et séparée par un délimiteur. Un test est effectué à chaque nœud pour savoir si c'est le dernier et enlever le délimiteur de trop à la fin du String.

2 tableaux d'objets personnalisés qui sont les parents

```
<!-- get Custom[] -> Custom[] -> Boolean -->
  &lt;xsl:for-each select="deuxDimension/input"&gt;
  public String getUCValue()
  {
    String returnValue = "";
    &lt;xsl:for-each select="BooleanC2"&gt;
      returnValue += "{";
      &lt;xsl:for-each select="BooleanC2i"&gt;
        returnValue += &lt;xsl:value-of select="@nameElement"/&gt;.IsChecked +
";;"  ;
      &lt;/xsl:for-each&gt;
      returnValue = returnValue.Substring(0, returnValue.Length - 2);
      returnValue += "}";
      &lt;xsl:if  test="not(position() = last())"&gt;
        returnValue += ";;";
      &lt;/xsl:if&gt;
    &lt;/xsl:for-each&gt;

    return returnValue;
  }
  &lt;/xsl:for-each&gt;
```

Le principe est le même que le traitement avec un tableau, sauf qu'il y a 2 for-each pour récupérer les valeurs : BooleanC2 s'occupe du 1er tableau (qui est le parent du 2ème tableau) et BooleanC2i qui s'occupe de parcourir les contrôles pour le 2ème tableau pour chaque entrée du 1er tableau.

- setUCValue(String str) :

Le point important dans cette méthode est le fait que c'est sur la base du contenu du String que sera déterminé le nombre de CheckBoxs à mettre dans le StackPanel.

Aucun tableau d'objets personnalisés qui est parent

```
<!-- set Boolean -->
&lt;xsl:for-each select="zeroDimension/output"&gt;
public void setUCValue(String str)
{
```

```
  stackPanel_uc1.Children.Clear();

  &lt;xsl:for-each select="BooleanC_Output"&gt;

      CheckBox cb = new CheckBox();
      cb.Name = "ucCheckBox_saisie_0";
      cb.IsChecked = Boolean.Parse(str);
      cb.Content = "valeur 0";
      stackPanel_uc1.Children.Add(cb);

  &lt;/xsl:for-each&gt;
}
&lt;/xsl:for-each&gt;
```

Chaque for-each n'est exécuté qu'une fois vu que le type actuellement traité n'a pas de parent qui serait un tableau d'objets personnalisés.

1 tableau d'objets personnalisés qui est le parent

```
<!-- set Custom[] -> Boolean -->
&lt;xsl:for-each select="uneDimension/output"&gt;
public void setUCValue(String str)
{
  stackPanel_uc1.Children.Clear();

  String[] tmpArray = str.Split(new String[] { ";;" },
System.StringSplitOptions.RemoveEmptyEntries);
  Boolean[] boolArray = new Boolean[tmpArray.Length] ;
  for(int i = 0; i &lt;![CDATA[&lt;]]&gt; tmpArray.Length; i++)
  {
    boolArray[i] = Boolean.Parse(tmpArray[i]) ;
  }

  int cpt = 0 ;
  <!-- Output : génération des contrôles par rapport aux données -->
  &lt;xsl:for-each select="BooleanC_Output"&gt;
    for(int i=0; i &lt;![CDATA[&lt;]]&gt; boolArray.Length; i++)
    {
      CheckBox cb = new CheckBox();
      cb.Name = "ucCheckBox_saisie_" + i;
      cb.IsChecked = boolArray[cpt++];
      cb.Content = "tbl [" + i + "]";
      stackPanel_uc1.Children.Add(cb);
    }

  &lt;/xsl:for-each&gt;
}
&lt;/xsl:for-each&gt;
```

Le premier for-each est exécuté une fois. Ensuite, la première chose est d'effacer le contenu du StackPanel (pour le cas où l'on reçoit une nouvelle réponse du web service). Le String est décomposé en un tableau de Boolean. Ensuite, le for-each BooleanC_Output s'occupe de créer le contrôle, d'y mettre la valeur et de l'afficher dans le StackPanel et ce, pour tous les éléments du tableau parent.

2 tableaux d'objets personnalisés qui sont les parents

```
<!-- set Custom[] -> Custom[] -> Boolean -->
&lt;xsl:for-each select="deuxDimension/output"&gt;
public void setUCValue(String str)
{
  stackPanel_uc1.Children.Clear();

  int cpt = 0;
  List&lt;![CDATA[&lt;]]&gt;Boolean&lt;![CDATA[&gt;]]&gt; boolList = new
List&lt;![CDATA[&lt;]]&gt;Boolean&lt;![CDATA[&gt;]]&gt;() ;
  String[] tmpArray = str.Split(new String[] { "};;{" },
System.StringSplitOptions.RemoveEmptyEntries);
```

```
  List&lt;![CDATA[&lt;]]&gt;int&lt;![CDATA[&gt;]]&gt; cptList1 = new
List&lt;![CDATA[&lt;]]&gt;int&lt;![CDATA[&gt;]]&gt;() ;

    for(int i = 0; i &lt;![CDATA[&lt;]]&gt; tmpArray.Length; i++)
    {
      String[] tmp2Array = tmpArray[i].Split(new String[] { ";;" },
System.StringSplitOptions.RemoveEmptyEntries);
      cptList1.Add(tmp2Array.Length);

        for(int j = 0; j &lt;![CDATA[&lt;]]&gt; tmp2Array.Length; j++)
        {
          tmp2Array[j] = tmp2Array[j].Replace("{", "");
          tmp2Array[j] = tmp2Array[j].Replace("}", "");

          boolList.Add(Boolean.Parse(tmp2Array[j])) ;
        }
    }

    Boolean[] boolArray =
boolList.ToArray&lt;![CDATA[&lt;]]&gt;Boolean&lt;![CDATA[&gt;]]&gt;() ;
    int[] cptArray1 =
cptList1.ToArray&lt;![CDATA[&lt;]]&gt;int&lt;![CDATA[&gt;]]&gt;() ;

    &lt;xsl:for-each select="BooleanC2_Output"&gt;
    //faire une boucle pour ajouter les éléments de manière automatique au UC

    for(int i=0; i &lt;![CDATA[&lt;]]&gt; tmpArray.Length; i++)
    {
      Label lb = new Label();
      lb.Content = "tbl [" + i + "]" ;
      stackPanel_uc1.Children.Add(lb);

      for(int j=0; j &lt;![CDATA[&lt;]]&gt; cptArray1[i]; j++)
      {
        CheckBox cb = new CheckBox();
        cb.Name = "ucCheckBox_saisie_" + i + "_" + j;
        cb.IsChecked = boolArray[cpt++];
        cb.Content = "tbl [" + i + "," + j +"]";
        stackPanel_uc1.Children.Add(cb);
      }
    }
    &lt;/xsl:for-each&gt;
}
&lt;/xsl:for-each&gt;
```

Le principe est le même que le traitement avec un tableau mais un peu plus complexe. Je commence par créer une liste de Boolean nommée boolList dans laquelle je mets toutes valeurs trouvées dans le String passé en paramètre. Dans une autre liste de type int nommée cptList1, je mets pour chaque entrée du 1er tableau la taille de chaque 2ème tableau contenu ce qui me permet lors de la mise en place des éléments de savoir combien je dois prendre d'éléments de boolList pour chaque entrée du 1er tableau.

- cleanUCValue() :

Cette méthode est uniquement présente parce que par défaut, quel que soit l'UserControl utilisé, j'appelle toujours cette méthode. Dans ce cas de figure, la méthode ne fait rien.

rootBooleanArray

Ce template est destiné à la création de UserControls spécifiques pour le type de tableau de Boolean. Au lieu d'afficher un TextBox, il affiche une série de CheckBoxs (suivant ce qu'aura indiqué l'utilisateur).

Le code est calqué sur celui du template rootBoolean à la différence qu'il y a à la fin un tableau de Boolean à traiter au lieu d'un simple type de Boolean.

rootEnum
Ce template est destiné à la création de UserControls spécifiques pour un type qui est une énumération. Au lieu d'afficher un TextBox, il affiche un ComboBox dont le contenu est rempli avec les différentes énumérations du type.

- initUC_*() :

Pour pouvoir utiliser ce contrôle dans la partie input, il faut dans un premier temps initialiser les contrôles et mettre le nom de chaque énumération.

S'il n'y a pas de tableau d'objets personnalisés qui est parent :

```
<!-- init du contenu des ComboBoxs Enum -->
&lt;xsl:for-each select="zeroDimension/input"&gt;
private void initUC_zero()
{
    for (int i = 0; i &lt;![CDATA[&lt;]]&gt; enumStrArray.Length; i++)
    {
      ComboBoxItem cbi = new ComboBoxItem();
      cbi.Content = enumStrArray[i];
      ucComboBox_saisie_0.Items.Add(cbi);
    }
}
&lt;/xsl:for-each&gt;
```

S'il y a 1 tableau d'objets personnalisés qui est parent :

```
<!-- init du contenu des ComboBoxs Custom[] -> Enum -->
&lt;xsl:for-each select="uneDimension/input"&gt;
private void initUC_une()
{
  &lt;xsl:for-each select="EnumC"&gt;
    for (int i = 0; i &lt;![CDATA[&lt;]]&gt; enumStrArray.Length; i++)
    {
      ComboBoxItem cbi = new ComboBoxItem();
      cbi.Content = enumStrArray[i];
      &lt;xsl:value-of select="@nameElement"/&gt;.Items.Add(cbi);
    }
  &lt;/xsl:for-each&gt;
}
&lt;/xsl:for-each&gt;
```

Le for-each EnumC s'occupe pour chaque entrée du tableau parent de créer le contenu du ComboBox.

S'il y a 2 tableaux d'objets personnalisés qui sont les parents :

```
<!-- init du contenu des ComboBoxs Custom[] -> Custom[] -> Enum -->
&lt;xsl:for-each select="deuxDimension/input"&gt;
private void initUC_deux()
{
  &lt;xsl:for-each select="EnumC2"&gt;
    &lt;xsl:for-each select="EnumC2i"&gt;
    for (int i = 0; i &lt;![CDATA[&lt;]]&gt; enumStrArray.Length; i++)
    {
      ComboBoxItem cbi = new ComboBoxItem();
      cbi.Content = enumStrArray[i];
      &lt;xsl:value-of select="@nameElement"/&gt;.Items.Add(cbi);
    }
    &lt;/xsl:for-each&gt;
  &lt;/xsl:for-each&gt;
```

```
}
&lt;/xsl:for-each&gt;
```

Même opération à effectuer que pour 1 tableau à la différence qu'il faut traiter les 2 tableaux.

- getUCValue() :

S'il n'y a pas de tableau d'objets personnalisés qui est parent :

```
<!-- get Enum -->
&lt;xsl:for-each select="zeroDimension/input"&gt;
public String getUCValue()
{
  String returnValue = "";
  &lt;xsl:for-each select="EnumC"&gt;
    ComboBoxItem cbi = (ComboBoxItem)ucComboBox_saisie_0.SelectedItem;
    return cbi.Content.ToString();
  &lt;/xsl:for-each&gt;
}
&lt;/xsl:for-each&gt;
```

Il s'agit simplement de récupérer la valeur de l'index sélectionné.

S'il y a 1 tableau d'objets personnalisés qui est parent :

```
<!-- get Custom[] -> Enum -->
&lt;xsl:for-each select="uneDimension/input"&gt;
public String getUCValue()
{
  String returnValue = "";
  ComboBoxItem cbi = null ;
  &lt;xsl:for-each select="EnumC"&gt;
    cbi = (ComboBoxItem)&lt;xsl:value-of select="@nameElement"/&gt;.SelectedItem;
    returnValue += cbi.Content.ToString() + ";;";

    &lt;xsl:if test="position() = last()"&gt;
      returnValue = returnValue.Substring(0, returnValue.Length - 2);
    &lt;/xsl:if&gt;

  &lt;/xsl:for-each&gt;

  return returnValue;
}
&lt;/xsl:for-each&gt;
```

Il faut parcourir le for-each pour traiter toutes les entrées du tableau (qui équivaut au nombre de fois qu'apparaît EnumC dans le fichier de données XML).

S'il y a 2 tableaux d'objets personnalisés qui sont les parents :

```
<!-- get Custom[] -> Custom[] -> Enum -->
&lt;xsl:for-each select="deuxDimension/input"&gt;
public String getUCValue()
{
  String returnValue = "";
  ComboBoxItem cbi = null ;

  &lt;xsl:for-each select="EnumC2"&gt;
    returnValue += "{";
    &lt;xsl:for-each select="EnumC2i"&gt;
      cbi = (ComboBoxItem)&lt;xsl:value-of
select="@nameElement"/&gt;.SelectedItem;
      returnValue += cbi.Content.ToString() + ";;";
    &lt;/xsl:for-each&gt;
    returnValue = returnValue.Substring(0, returnValue.Length - 2);
```

```
    returnValue += "}";
    &lt;xsl:if  test="not(position() = last())"&gt;
      returnValue += ";;";
    &lt;/xsl:if&gt;
  &lt;/xsl:for-each&gt;
  return returnValue;
}
&lt;/xsl:for-each&gt;
```

Même opération que pour 1 tableau, sauf qu'il y en a 2 à traiter (le tableau parent du 2ème tableau est traité avec le for-each EnumC2 et pour chaque entrée du 1er tableau, le 2ème tableau est traité avec le for-each EnumC2i).

- setUCValue(String str) :

Cette méthode s'occupe de sélectionner la bonne énumération.

S'il n'y a pas de tableau d'objets personnalisés qui est parent :

```
<!-- set Enum -->
&lt;xsl:for-each select="zeroDimension/output"&gt;
public void setUCValue(String str)
{
  stackPanel_uc1.Children.Clear();
  <!-- Poser le ComboBox dans le panel avec ses valeurs -->
  &lt;xsl:for-each select="EnumC_Output"&gt;

      ComboBox cb = new ComboBox();
      cb.Name = "ucComboBox_saisie_0";

        for (int i = 0; i &lt;![CDATA[&lt;]]&gt; enumStrArray.Length; i++)
        {
            ComboBoxItem cbi = new ComboBoxItem();
            cbi.Content = enumStrArray[i];
            cb.Items.Add(cbi);
        }

      ItemCollection ic = cb.Items;
      stackPanel_uc1.Children.Add(cb);

  <!-- Sélectionner le bon index -->
  int cpt = 0;
  Boolean isSet = false;

  while (cpt &lt;![CDATA[&lt;]]&gt; ic.Count &lt;![CDATA[&amp;&amp;]]&gt; !isSet)
  {
      ComboBoxItem cbi = (ComboBoxItem)ic.GetItemAt(cpt);

      if (cbi.Content.ToString() == str)
      {
          cb.SelectedIndex = cpt;
          isSet = true;
      }
      cpt++;
  }
  &lt;/xsl:for-each&gt;
}
&lt;/xsl:for-each&gt;
```

Le premier for-each s'occupe de créer le ComboBox, d'ajouter les items et d'afficher le StackPanel le contrôle.
Il ne reste ensuite plus qu'à sélectionner le bon index en comparant le String reçu en paramètre avec les items du ComboBox.

S'il y a 1 tableau d'objets personnalisés qui est parent :

```
<!-- set Custom[] -> Enum -->
&lt;xsl:for-each select="uneDimension/output"&gt;
public void setUCValue(String str)
{
  stackPanel_uc1.Children.Clear();
  <!-- Poser les ComboBoxs dans le panel avec ses valeurs -->

  ComboBox cb = null ;
  &lt;xsl:for-each select="EnumC_Output"&gt;
    String[] tmpArray = str.Split(new String[] { ";;" },
System.StringSplitOptions.RemoveEmptyEntries);

    for(int i = 0; i &lt;![CDATA[&lt;]]&gt; tmpArray.Length; i++)
    {
      cb = new ComboBox();
      cb.Name = "<xsl:value-of select="@nameElement"/>";

        for (int j = 0; j &lt;![CDATA[&lt;]]&gt; enumStrArray.Length; j++)
        {
            ComboBoxItem cbi = new ComboBoxItem();
            cbi.Content = enumStrArray[j];
            cb.Items.Add(cbi);
        }

      ItemCollection ic = cb.Items;
      stackPanel_uc1.Children.Add(cb);
      <!-- Sélectionner le bon index -->
      int cpt = 0;
      Boolean isSet = false;

      while (cpt &lt;![CDATA[&lt;]]&gt; ic.Count &lt;![CDATA[&amp;&amp;]]&gt;
!isSet)
      {
          ComboBoxItem cbi = (ComboBoxItem)ic.GetItemAt(cpt);

          if (cbi.Content.ToString() == tmpArray[i])
          {
              cb.SelectedIndex = cpt;
              isSet = true;
          }
          cpt++;
      }
    }
  &lt;/xsl:for-each&gt;
}
&lt;/xsl:for-each&gt;
```

Même opération que précédemment sauf que l'on a un tableau parent.
Après avoir enlevé tous les contrôles que pourrait avoir dans le StackPanel, un objet de type ComboBox est créé pour éviter d'avoir par la suite un même nom de variable dans une boucle.
Le nombre de ComboBox à mettre est déterminé par le String passé en paramètre. Cette information est obtenue en splitant le String pour en faire un tableau et il est parcouru pour mettre à chaque entrée le contrôle.
A chaque fois, il faut mettre les items du contrôle puis l'ajouter au StackPanel et finalement sélectionner le bon index.
A noter que le nom du ComboBox n'est pas nécessaire (au final, ça donne un string vide).

S'il y a 2 tableaux d'objets personnalisés qui sont les parents :

```
<!-- set Custom[] -> Custom[] -> Enum -->
&lt;xsl:for-each select="deuxDimension/output"&gt;
public void setUCValue(String str)
{
  stackPanel_uc1.Children.Clear();
```

```
  <!-- Poser les ComboBoxs dans le panel avec ses valeurs -->

  ComboBox cb = null ;
  &lt;xsl:for-each select="EnumC2_Output"&gt;
    String[] tmpArray = str.Split(new String[] { "};;{" },
System.StringSplitOptions.RemoveEmptyEntries);

    for(int i = 0; i &lt;![CDATA[&lt;]]&gt; tmpArray.Length; i++)
    {
      Label lb = new Label();
      lb.Content = "tbl [" + i + "]" ;
      stackPanel_uc1.Children.Add(lb);

      String[] tmp2Array = tmpArray[i].Split(new String[] { ";;" },
System.StringSplitOptions.RemoveEmptyEntries);

      for(int j = 0; j &lt;![CDATA[&lt;]]&gt; tmp2Array.Length; j++)
      {
        cb = new ComboBox();
        cb.Name = "<xsl:value-of select="@nameElement"/>";

        tmp2Array[j] = tmp2Array[j].Replace("{", "");
        tmp2Array[j] = tmp2Array[j].Replace("}", "");

        for (int k = 0; k &lt;![CDATA[&lt;]]&gt; enumStrArray.Length; k++)
        {
            ComboBoxItem cbi = new ComboBoxItem();
            cbi.Content = enumStrArray[k];
            cb.Items.Add(cbi);
        }
        ItemCollection ic = cb.Items;
        stackPanel_uc1.Children.Add(cb);
        <!-- Sélectionner le bon index -->
        int cpt = 0;
        Boolean isSet = false;

        while (cpt &lt;![CDATA[&lt;]]&gt; ic.Count &lt;![CDATA[&amp;&amp;]]&gt;
!isSet)
        {
            ComboBoxItem cbi = (ComboBoxItem)ic.GetItemAt(cpt);

            if (cbi.Content.ToString() == tmp2Array[j])
            {
                cb.SelectedIndex = cpt;
                isSet = true;
            }
            cpt++;
        }
      }
    }
  &lt;/xsl:for-each&gt;
}
&lt;/xsl:for-each&gt;
```

Même opération que pour 1 tableau sauf qu'il y a 2 tableaux à traiter. Pour chaque entrée du 1er tableau, un Label est ajouté pour avoir une meilleure vue de comment sont organisés les éléments lors de l'utilisation de l'application.

- cleanUCValue() :

Cette méthode est uniquement présente parce que par défaut, quel que soit l'UserControl utilisé, j'appelle toujours cette méthode. Dans ce cas de figure, la méthode ne fait rien.

rootEnumArray

Ce template est destiné à la création de UserControls spécifiques pour le type de tableau d'énumération. Au lieu d'afficher un TextBox, il affiche une série de ComboBox (suivant ce qu'aura indiqué l'utilisateur).

Le code est calqué sur celui du template rootEnum à la différence qu'il y a la fin un tableau de type d'énumération à traiter au lieu d'un simple type d'énumération.

3.7.2. Csproj

Le fichier csproj est important pour la création de l'application générée, c'est sur lui que s'appuie msbuild. Il y a 2 fichiers de transformation : le premier destiné à la génération automatique et wizard et le 2ème pour la génération des tests Nunits. Ces fichiers proviennent d'un fichier csproj d'un projet VS et ont été adaptés et transformés pour en faire des fichiers de transformation.

3.7.2.1. manifest_csproj.xslt

C'est celui des deux fichiers qui contient le plus d'éléments (car il est destiné à générer une application graphique).
Les points importants sont les suivants :

- **Template match property**
  Il spécifie comment doit être fait la génération et l'emplacement où les fichiers seront mis.
- **Template match ref**
  Ce template s'occupe de référencer toutes les assemblies qui doivent être incluses ainsi que la dll du proxy.
- **Template starteritem**
  Ce template permet de spécifier par quel fichier l'application doit commencer (en général, app.xaml et app.caml.cs
- **Template item**
  Ce template référence tous les fichiers que doit contenir l'application, comment ils doivent être traités par le générateur, les dépendances entre fichiers.

3.7.2.2. manifest_nunit_csproj.xslt

Les points importants sont les suivants :

- **Template match property**
  Il spécifie comment doit être fait la génération et l'emplacement où les fichiers seront mis.
- **Template match ref**
  Ce template s'occupe de référencer toutes les assemblies qui doivent être incluses ainsi que la dll du proxy.
- **Template itemgroup2**
  Ce template référence tous les fichiers que doit contenir l'application, comment ils doivent être traités par le générateur, les dépendances entre fichiers.

3.7.3. App

Ces 2 fichiers de transformation s'occupent de créer le fichier XAML et C# pour le point d'entrée de l'application. Ces fichiers servent juste dans le cas où l'on changerait le nom des fichiers app.

3.7.4. XAML

Ces 2 fichiers de transformation s'occupent de la partie interface utilisateur pour la fenêtre principale de l'application générée.

3.7.4.1. Window_xaml_2.xslt

Ce fichier de transformation est utilisé pour créer l'interface utilisateur pour la génération automatique. Il s'occupe de créer la fenêtre XAML et d'y déposer les différents UserControls.

- **Template name=window**
  La transformation commence réellement avec ce template. La première chose à signaler, c'est l'utilisation d'un mappage de nom pour pouvoir utiliser les UserControls (pour qu'ils soient reconnus).

```
xmlns:my="clr-namespace:replace"
```

  Normalement, à la place de replace, il doit y avoir le namespace de l'application suivit du nom du dossier contenant les UserControls.
  Cette information est écrite après la transformation, car il m'a été impossible de le faire lors de la transformation sans que ça n'entraîne un échec de la transformation.
  Ensuite, il faut renseigner les valeurs pour le nom de la classe du code behind pour cette fenêtre et indiquer le nom qui doit apparaître dans le titre.
- **Template name=core**
  On rentre dans le contenu de la fenêtre. Il s'agit de place l'élément root pour tout le contenu et définir un Grid contenant 2 lignes.
  La 1ère ligne contiendra tous les UserControls pour les saisies de l'utilisateur ainsi que l'affichage du résultat obtenu de la part du web service le tout dans un TabControl.
  La 2ème ligne contiendra les éventuels messages d'erreur.
- **Template name=operation**
  Pour chaque opération du web service qui doit être traité, ce template est appelé.
  Il s'occupe de créer des TabItem avec comme Header le nom de l'opération et qui contient un ScrollViewer pour scroller verticalement si le contenu est trop grand et à l'intérieur un StackPanel vertical pour empiler les différents contrôles que l'on va mettre.
  Tout en bas de ce StackPanel, je place un bouton pour déclencher une action (sa mission sera d'interagir avec le web service).
- **Template match=//input et //output**
  Ce sont 2 templates intermédiaires. Ils sont présents uniquement pour pouvoir afficher un GroupBox afin de différencier la partie input de l'output du web service.
- **Template match=//parametre**
  Ce template s'occupe à chaque fois d'appeler le template inserElement pour ajouter un UserControl, puis fait un apply-templates pour rechercher d'autres paramètres et finalement, regarde si c'était le dernier élément et si ce n'est pas le petit-fils du nœud input pour refermer le StackPanel et le GroupBox
- **Template name=insertElement**
  C'est dans ce template que les UserControls sont écrits en se basant sur le type pour choisir quel UserControl doit être mis. Le choix est affiné

suivant que l'UserControl à mettre est dans input ou output (ils sont identiques actuellement mais on peut imaginer avoir des tests à réaliser spécifiquement pour l'UserControl dans la partie input qui n'aurait pas lieu d'être dans la partie output).

Dans le cas où le type n'est pas trouvé, il faut déterminer si c'est un type énuméré ou un tableau de type énuméré. Si c'est le cas, l'UserControl sera appliqué. Dans le cas contraire, le type est une classe qui contient des types, donc, il ne faut pas mettre d'UserControl mais ouvrir un nouveau GroupBox contenant un StackPanel afin de voir la différence à l'écran.

3.7.4.2. WizardWindow_xaml.xslt

Ce fichier de transformation est utilisé pour créer l'interface utilisateur pour la génération wizard. Le traitement effectué est pratiquement le même que celui effectué par Window_xaml_2.xslt.
La différence réside dans le template name=insertElement.
Si le type traité correspond à l'un des types standards, l'UserControl est appliqué mais utilise le nom de l'UserControl qui est créé pour ce type à la position x. Chaque UserControl est indépendant des autres. Si le type n'est pas standard et n'est pas une énumération, un GroupBox est également placé (indiquant visuellement que c'est une classe).

3.7.5. C#

3.7.5.1. AllWindow_xaml_cs_2.xslt

Ce fichier de transformation s'occupe de créer pour la génération automatique et wizard le code behind pour l'interface utilisateur qui contiendra les différentes méthodes pour interagir avec les opérations du web service.

- **Template match=using**
  Ce template écrit les différents assemblies nécessaires
- **Template match=namespace**
  Ce template permet de poser le namespace. A l'intérieur, il y a 2 appels de template : le template assembly, qui pour certains web services, permet de mettre la référence pour le web service et que l'objet client soit connu lors de son utilisation (pour certains web service, si la référence pour l'objet client est mise en dehors du namespace, l'objet n'est pas connu et entraîne une erreur lors de génération de l'application). Le template classe permet de continuer l'insertion du contenu.
- **Template match=classe**
  Ce template s'occupe d'écrire la classe qui contient le client et l'initialisation de la fenêtre. Il y a également l'appel à 2 templates : select=operation mode=init et select=operation mode=action qui permettent de mettre à jour le label pour chaque UserControl.
- **Template match=operation mode=action**
  Pour chaque opération, une méthode est créée pour l'événement Click du bouton et contient tout le traitement pour préparer les paramètres à fournir à l'opération ainsi que le traitement pour afficher le résultat.
  Le traitement fait appel à plusieurs templates :

1. **apply-templates select=output mode=outputInit**
   Application de différents templates dont le but est de :
   - effacer le message d'erreur s'il y a eu une utilisation précédente de l'application.
   - nettoyer le contenu des contrôles de saisie dans les UserControls
   - Déclarer pour chaque type standard une variable (ce sont ces variables de type String qui sont passés à la fin à chaque UserControl pour afficher le résultat de l'utilisation du web service.
2. **apply-templates select=input mode=input**
   Récupération des saisies de l'utilisateur et création des différents objets pour obtenir tous les éléments nécessaires qui seront passés en paramètres pour l'utilisation de l'opération.
3. **Ecrire la ligne qui appelle l'opération et récupère le retour**
   A la gauche du signe égal, le template nommé outputmode2 s'occupe de créer l'objet que l'opération retourne.
   A la droite du signe égal, la méthode de l'opération pour le client est appelée et les paramètres à passer sont traités par le template nommé mode2

```
<xsl:call-template name="outputmode2"/> = <xsl:value-of
select="@clientName" />.<xsl:value-of select="operationName/@name"
/>(<xsl:call-template name="mode2"/>);
```

4. **apply-templates select=output mode=output**
   Découper l'objet provenant de l'opération en types standards qui sont stockés dans des variables String.
5. **apply-templates select=output mode=outputResult**
   Pour chaque type, mettre la valeur dans le UserControl

- **Template match=parametre/type mode=input1**
  Ce template traite tous les noeuds parameter/type qui se trouve dans l'input pour récupérer les valeurs saisies dans les UserControls et les stocke dans des variables créées à ce moment.
  Ce template est appelé autant de fois qu'il y a de parametre/type à traiter. Une exception toutefois à cette règle pour le cas où il y a un tableau d'objets personnalisés. Le traitement de ce genre de cas n'ayant pas pu être résolu de la même manière que pour les autres types, dès que le traitement rentre dans ce cas de figure, il n'en sort plus pour tous les nœuds enfants qu'il contient.

  Un xsl:choose permet de sélectionner la bonne partie du template à utiliser suivant le type de données qu'il faut traiter (standard, complexe, énumération ou tableau d'énumération, objet personnalisé ou tableau d'objets personnalisés).

  Par exemple avec un nœud de type String :
  Le 1er xls:when est utilisé. A l'intérieur, il y a 2 possibilités :
  Si le nœud n'a pas de parent, une variable est créée (avec le type adéquat) et le contenu de l'UserControl y est mis (après avoir fait une conversion de type).
  Si le nœud à un parent, il faut s'assurer que ce ne soit pas un tableau

d'objets personnalisés (car ce traitement aura déjà été effectué dans la partie s'occupant de ce genre de type). Si c'est le cas, rien n'est fait. Dans le cas contraire, on se trouve dans le cas où le parent est un objet personnalisé. Il faut créer une variable avec le bon type et y mettre le contenu de l'UserControl dedans.
Ensuite, il faut mettre à jour le membre de l'objet parent (appelé par défaut custType_<id> avec la variable créée juste avant.

Le système est identique pour les cas de type complexe (un tableau de String par exemple) ainsi que les énumérations et tableaux d'énumérations, à la différence qu'il faut pour les cas comportant un tableau à traiter la donnée provenant de l'UserControl pour en faire un tableau.

Pour le cas où il y a un type personnalisé, créer un nouvel objet du type actuel (en s'assurant de ne pas avoir un parent qui est un tableau de type personnalisé) puis si le nœud courant est l'enfant d'un autre, il faut mettre à jour le membre du parent avec l'objet qui a été créé.

Finalement, le cas du tableau d'objets personnalisés. Une fois que traitement rentre dans cette partie, il n'en sort plus pour tout ce qui concerne le traitement des nœuds enfants. A l'intérieur, un for-each s'occupe de traiter les nœuds enfants et utilise des xsl:if pour appliquer le bon traitement à effectuer suivant le type rencontré.

Tout à la fin de cette partie, la construction des objets commence. Le traitement part depuis l'objet parent (déclaration d'une boucle for dans le code C# et création de l'objet). Le traitement est poursuivi en appelant le template select=parametre/type mode=inputArrayCust.

- **Templates select=parametre/type mode=inputArrayCust**
  Ce template s'occupe si besoin de créer les objets et surtout de renseigner les membres de l'objet parent. Le traitement suit la même logique que le template match=parametre/type mode=input1.
- **Template name=conversion**
  Ce template s'occupe d'écrire le casting de la valeur contenue dans le UserControl. A noter l'utilisation d'un paramètre pour utiliser ce template. Ce paramètre est en fait le nom de l'UserControl et la méthode à appeler pour obtenir sa valeur.
- **Template name=mode2**
  Ce template s'occupe d'écrire les paramètres que l'opération attend pour fonctionner.
- **Template name=outputmode2**
  Ce template traite le retour de l'opération du web service en créant un objet du type du retour de l'opération.
- **Template match=parametre/type mode=output1**
  Ce template s'occupe de décomposer l'objet reçu en retour de l'opération pour obtenir les types standards. Le principe de traitement est le même que pour le template match=parametre/type mode=input1 sauf que l'objet est décomposé.
  Le xsl:choose s'occupe d'appliquer le bon traitement par rapport au type (ou nœud) rencontré pour placer les valeurs dans des variables String

(qui ont été déclarées grâce au template select=output mode=outputInit).

Pour le cas d'un nœud de type primitif (ou énumération), s'il n'y a pas d'objet parent, la valeur de retour est stockée dans une variable str_<id>. S'il y a un parent (et qui ne soit pas un tableau d'objets personnalisés), le contenu provient du membre de l'objet parent.

Pour le cas d'un nœud de type complexe (ou tableau d'énumération), s'il n'y a pas de parent, le tableau est parcouru, les valeurs sont stockées dans un String et séparées par un délimiteur. S'il y a un parent, c'est la même opération mais le contenu provient du membre de l'objet parent.

Pour le cas d'un nœud de type personnalisé, s'il y a un parent, le contenu pris est le membre du parent sinon il provient directement de l'opération du web service. Dans les 2 cas, l'objet est mis dans un nouvel objet du même type et il sera traité par une autre partie du template lors d'un nouveau passage.

Pour le cas des tableaux de types personnalisés, une fois que le traitement rentre à l'intérieur, il n'en ressort plus pour les nœuds enfants. A l'intérieur, un for-each s'occupe de traiter les nœuds enfants et utilise des xsl:if pour appliquer le bon traitement à effectuer suivant le type rencontré.
A noter qu'à chaque fois qu'un tableau de types personnalisés, il y a la création d'une liste pour pouvoir ensuite parcourir avec une boucle for le contenu du tableau (au niveau du code C#).

3.7.5.2. Test_ObjectClass_cs.xslt

Ce fichier de transformation s'occupe de créer la classe qui contiendra les différents appels aux opérations du web service pour la génération des tests NUnit.

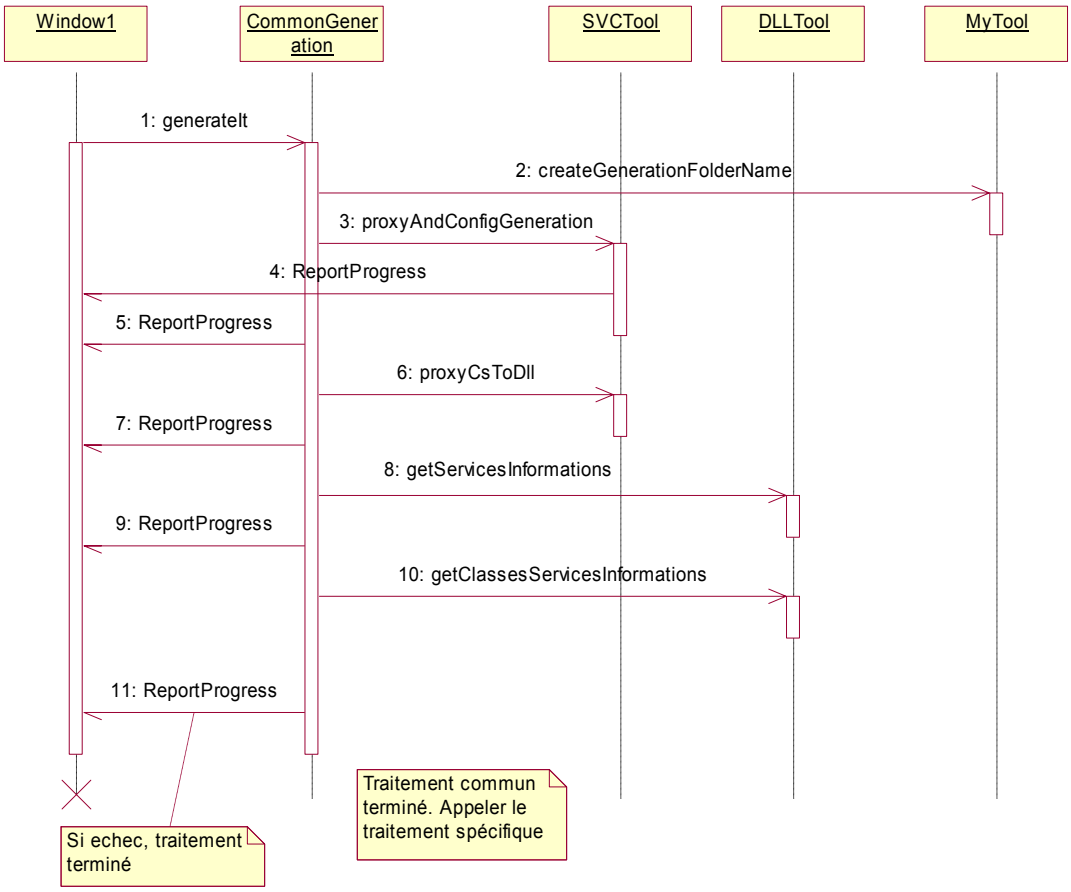
- **Template match=classe**
  Ce template écrit le client pour utiliser le web service.
- **Template match=classe/operation**
  Ce template écrit une méthode pour accéder à une opération du web service. Ce template est répété jusqu'à ce que toutes les opérations soient traitées.
- **Template name=returnTypeOutput**
  Ce template est appelé pour chaque méthode afin d'y mettre le type que doit retourner la méthode.
- **Template name=inputTypeFunctionParam**
  Ce template s'occupe d'écrire les paramètres que la méthode attend et qui sont en fait les paramètres à utiliser pour l'opération du web service (quel que soit le type. Si c'est un objet qu'il faut passer en paramètre, j'écris tel quel l'objet.).
  Pour chaque paramètre, le type et le nom est écrit.
- **Template name=inputTypeWSParam**
  Même opération que inputTypeFunctionParam à la différence que ce template est juste utilisé pour poser les paramètres de l'opération dans son appel.

3.7.5.3. Test_UnitObjectClass_cs.xslt

Ce fichier de transformation s'occupe de créer la classe qui contiendra les différents tests unitaires que l'utilisateur pourra saisir. Au niveau des templates, le template match= nunitObjectClassAssembly ajoute les assemblies nécessaires et le template match=namespace permet de spécifier le nom du namespace utilisé ainsi que le nom de la classe de tests.

4. Séquence des traitements de l'application

4.1. Récupération des informations provenant du WSDL



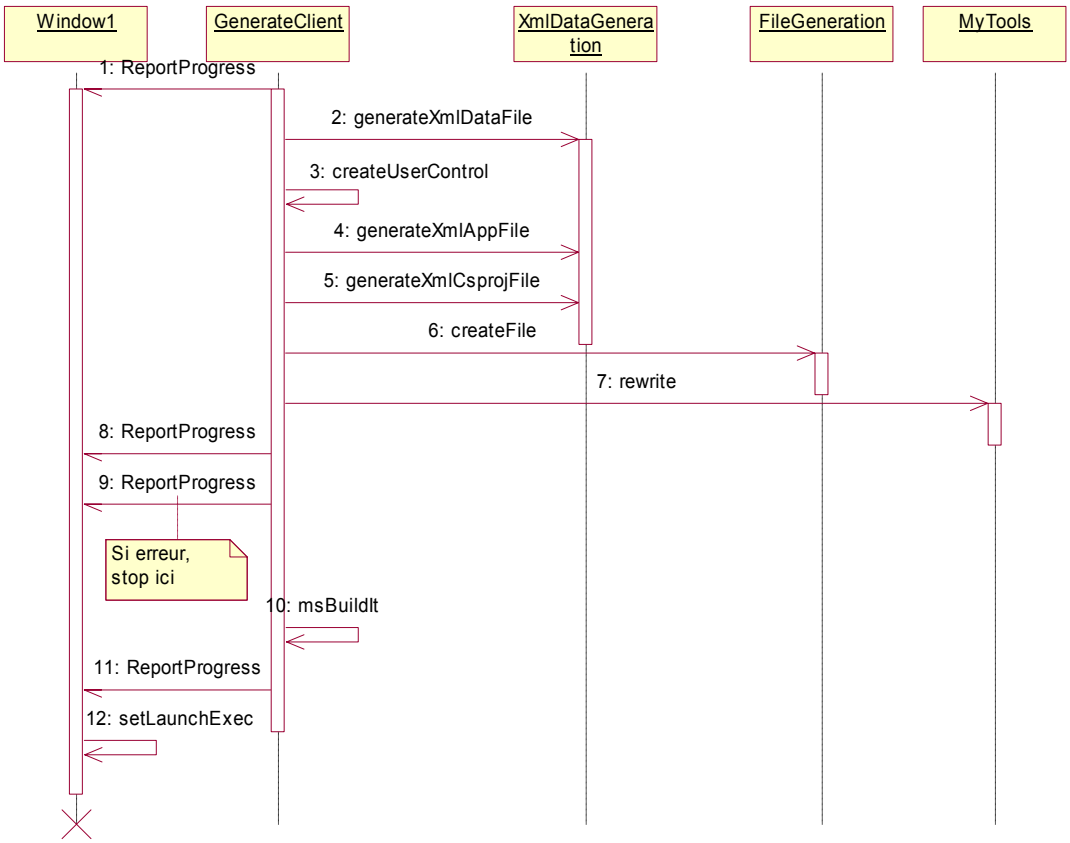
Tout ce traitement est effectué pour les différentes générations. Il est axé autour de la classe CommonGeneration qui se charge d'utiliser les différentes classes permettant de créer la base nécessaire (à savoir, obtenir toutes les informations utiles du WSDL) qui seront utilisés par la suite pour réaliser une application.

1. La méthode generateIt(int choix) est appelée lorsque l'utilisateur demande une génération (parmi les 3 disponibles, la variable int indiquant quel doit être la génération).
2. Création du dossier qui va contenir l'application générée, mais également tous les fichiers temporaires et les fichiers de code. C'est la méthode statique createGenerationFolderName(choix) de la classe MyTools qui est chargée du travail. A noter qu'ensuite, il y a une série d'objets qui sont initialisés.
3. Utilisation de la méthode proxyAndConfigGeneration(bw) de la classe SVCTool afin de créer le proxy et le fichier de configuration.
4. Un objet de type bw a été passé en paramètre dans la méthode précédente. Ceci permet de mettre à jour l'affichage en mettant un texte dans le statusBarItem de

la fenêtre. Affiche à l'utilisateur le message suivant : Web service en cours d'interrogation par svcutil. Ce message est important car le traitement peut prendre du temps suivant que le web service est en local ou à distance, que la connexion Internet est rapide ou non et le pc puissant ou non.

5. Idem qu'en 4. Informer l'utilisateur que la génération de la dll du proxy est en cours.
6. Utilisation de la méthode proxyCsToDll() de MyTools pour transformer le fichier proxy.cs en une dll afin de pouvoir y accéder en runtime pour obtenir les informations du web service.
7. Informer l'utilisateur que l'application récupère la liste des opérations
8. getServicesInformations() s'occupe de rechercher toutes les opérations qui sont disponibles dans la dll du proxy.
9. Informer l'utilisateur que l'application va récupérer les classes spéciales
10. getClassesServicesInformations() s'occupe de rechercher toutes classes qui sont des classes personnalisées et de connaître tous les membres présents (c'est un inventaire).
11. Si une erreur s'est produite, il faut informer l'utilisateur.
12. Dans le cas où il n'y a pas eu d'erreur, un switch permet de diriger le code sur la classe spécifique à utiliser pour réaliser la génération demandée par l'utilisateur.

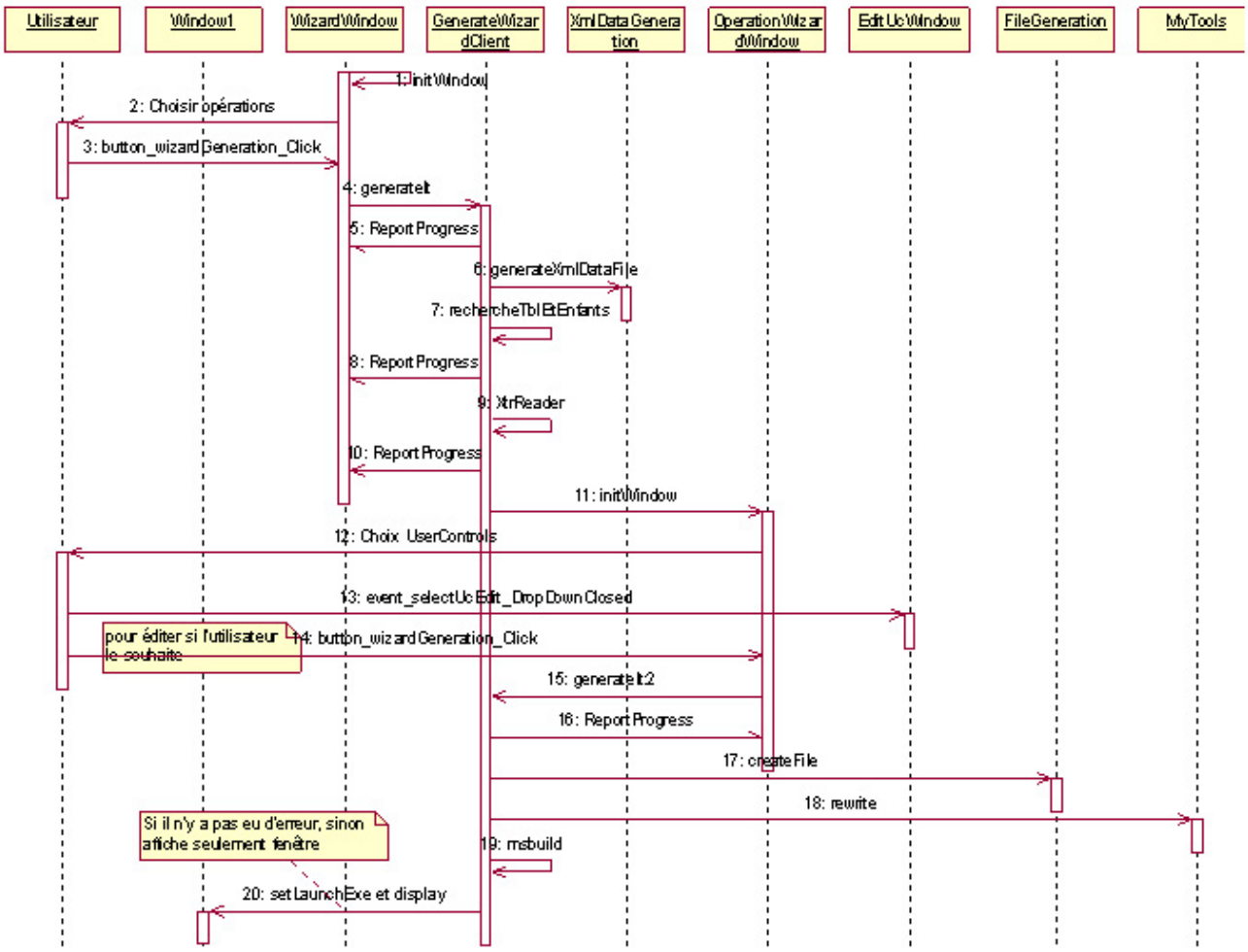
4.2.Générer du code pour la génération automatique



Le traitement pour produire une application de manière automatique ne demandant pas d'intervention de la part de l'utilisateur est effectué par la classe GenerateClient avec l'aide de différentes classes.
Lorsque le traitement effectué par CommonGeneration est terminé, un objet GenerateClient est instancié (en passant en paramètre les objets BackgroundWorker et CommonGeneration (ceci pour avoir la liste des opérations, des classes spéciales et le nom du dossier de génération entre autres)).

1. Informer l'utilisateur que l'application va créer les fichiers XAML et C#.
2. Création du fichier de données XML pour utiliser par la suite les transformations XSTL. Ce fichier contient les différents assemblies nécessaires, les informations sur l'objet client qui doit être créé et toutes les opérations disponibles contenant tous les types qui seront utilisés pour réaliser l'opération.
3. Création de tous les UserControls en utilisant le fichier de données XML créé dans l'étape 2.
4. Création du fichier de données XML pour le fichier App (point d'entrée de l'application générée).
5. Création du fichier de données XML pour le fichier csproj.
6. La méthode createFile de l'objet FileGeneration s'occupe de réaliser la transformation XSLT. La transformation est faite pour obtenir les fichiers finaux pour App, Window1 et csproj.
7. La méthode statique rewrite de MyTools permet de parcourir les fichiers créés pour convertir les caractères spéciaux de substitution utilisés lors de la transformation.
8. Informer l'utilisateur que la génération de l'application commence
9. Si il y a eu une erreur dans le traitement, informer l'utilisateur
10. Génération de l'application
11. Informer l'utilisateur sur le résultat de la génération
12. Si la génération s'est déroulée avec succès, afficher un bouton dans la fenêtre principale permettant de lancer l'application générée.

4.3.Générer du code pour la génération wizard



Le traitement pour produire une application wizard (avec intervention de l'utilisateur) est du ressort de la classe GenerateWizardClient avec l'aide de différentes classes. Les interactions avec l'utilisateur sont le travail des classes WizardWindow pour le choix des opérations et OperationWizardWindow pour le choix des UserControls.

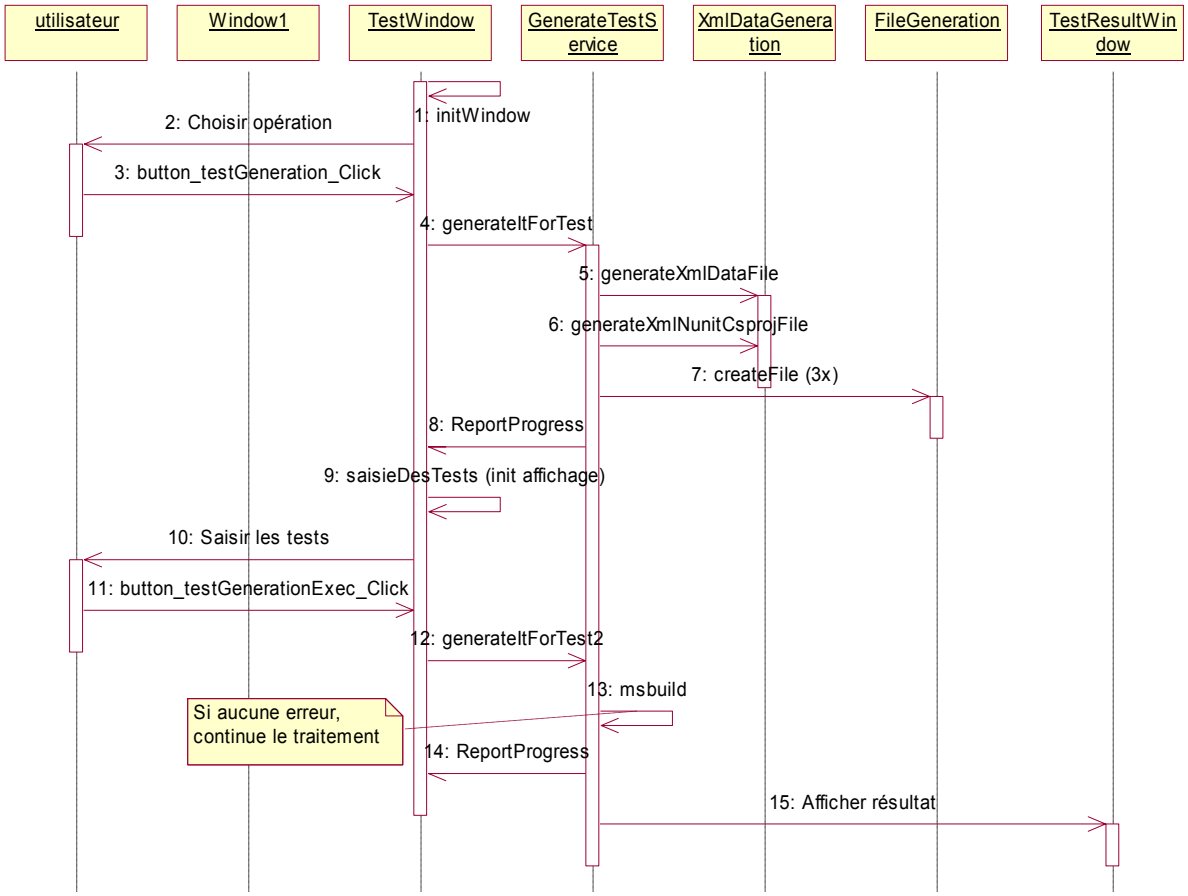
Lorsque le traitement effectué par CommonGeneration est terminé, un objet GenerateClient est instancié (en passant en paramètre les objets BackgroundWorker, la liste des opérations disponibles, la liste des classes spéciales et le nom du dossier de génération.

1. Initialisation de la fenêtre permettant à l'utilisateur de choisir les opérations qu'il souhaite traiter. Les opérations sont listées avec un CheckBox et un bouton pour valider le choix.
2. La fenêtre est affichée à l'utilisateur et il fait son choix.
3. Le choix est validé par l'utilisateur. Le code de l'événement du bouton s'occupe de récupérer les opérations sélectionnées.
4. Le BackgroundWorker est appelé par l'événement du bouton et s'occupe d'exécuter l'instanciation de l'objet de type GenerateWizardClient et la méthode generateIt est appelée pour lancer la création des fichiers.
5. Informer l'utilisateur que l'application prépare les données nécessaires pour la création des UserControls.
6. Création du fichier de données XML pour utiliser par la suite les transformations XSTL. Ce fichier contient les différents assemblies nécessaires, les informations sur l'objet client qui doit être créé et toutes les opérations disponibles contenant tous les types qui seront utilisés pour réaliser l'opération.
7. La méthode rechercheTblEtEnfants s'occupe de rechercher dans le fichier de données tous les tableaux de type personnalisé tous les enfants (types) qu'il contient.
8. Informer l'utilisateur que l'application crée les UserControls.
9. A ce stade, l'application va parcourir le fichier de données des opérations et pour chaque type trouvé, créer le ou les UserControls. C'est ici également que l'application contrôle pour chaque type s'il a des parents qui sont des tableaux personnalisés (si c'est le cas, la création de l'UserControl se fait différemment et utilise pour chaque cas un fichier de données pour l'UserControl différent).
10. Informer l'utilisateur sur le résultat de la création des UserControls. A ce stade, tous les types présents pour les opérations on au moins un UserControl standard et pour les types tels que Boolean, Boolean[], les énumérations et tableau d'énumérations, un UserControl spécial qui est personnalisé suivant la présence ou non de tableaux personnalisés comme parent.
11. Initialisation de la fenêtre permettant à l'utilisateur de choisir les UserControls à utiliser qui sont pour chaque type affiché dans un ComboBox.
12. La fenêtre est affichée à l'utilisateur pour qu'il puisse choisir les UserControls qu'il souhaite utiliser.
    A chaque fois que l'utilisateur choisi un UserControl, son choix est stocké dans un ArrayList ce qui permettra plus tard de savoir quels sont les UserControls qu'il faudra incorporer pour la génération.
13. Si l'utilisateur utilise des UserControls spéciaux pour la partie input, il doit impérativement les éditer (qu'ils contiennent ou non des tableaux personnalisés, car en les éditant, l'application effectue une transformation XSLT pour arriver à un fichier XAML et C# final et utilisable. Cette édition permet de spécifier pour les cas où il y a des tableaux personnalisés parents de saisir la taille.
14. Une fois le choix terminé, le code d'événement du bouton s'occupe de contrôler qu'il n'y a pas de UserControls spéciaux non transformés et s'occupe de lancer

un nouveau BackgroundWorker qui a pour tâche de lancer la méthode generateIt2 de l'objet GenerateWizardClient.

15. Lancement de la méthode generateIt2 pour générer l'application
16. Informer l'utilisateur que l'application s'occupe de prendre tous les UserControls qui ont été choisis.
17. Création des fichiers de données pour l'App, Window1 et le csproj avec à chaque fois la transformation effectuée par la méthode createFile de l'objet FileGeneration.
18. Chaque fichier est réécrit pour convertir les caractères spéciaux.
19. Génération de l'application avec msbuild
20. Affichage du résultat de la génération. Si elle s'est bien déroulée, un bouton est créé dans la fenêtre principale pour lancer l'application générée.

4.4.Générer du code pour la génération test Nunit



Le traitement pour produire une application de tests est effectué par la classe GenerateTestService avec l'aide de différentes classes. Les interactions avec l'utilisateur sont le travail de la classe TestWindow et le résultat est affiché dans TestResultWindow.

1. A la fin du traitement de CommonGeneration, la fenêtre TestWindow est appelée et initialisée pour lister les différentes opérations disponibles.
2. La fenêtre est affichée et l'utilisateur sélectionne les différentes opérations qu'il souhaite tester.
3. L'événement du bouton s'occupe de récupérer les opérations sélectionnées.
4. Le BackgroundWorker lance la méthde generateItForTest de l'objet GenerateTestService.
5. Création du fichier de données XML.
6. Création du fichier de données XML pour le csproj.

7. Transformation XSLT des fichiers Test_ObjectClass, Test_UnitObjectClass et du csproj.
8. Effacer le contenu du statusBarItem de la fenêtre TestWindow.
9. Appel de la méthode saisieDesTests qui réinitialise le contenu de la fenêtre et affiche le contenu des 2 classes C# pour les tests.
10. La fenêtre est prête pour que l'utilisateur saisisse les tests NUnits qui souhaite faire.
11. L'événement du bouton s'occupe de sauver les modifications faites dans pour les tests (le fichier Test_UnitObjectClass)
12. Le backgroundWorker s'occupe d'appeler la méthode generateItForTest2.
13. Génération de l'application.
14. S'il y a eu une erreur après la génération, le traitement s'arrête et un message informe l'utilisateur dans la fenêtre principale. Dans le cas contraire, les tests sont exécutés avec NUnit. Une fois finie, la fenêtre de résultat est affichée à l'utilisateur.

5. Exemples de fichiers générés

5.1. Génération automatique

Le web service WCFCalculator contient 4 opérations : add (addition), div (division), mult (multiplication) et soustr (soustraction). Chaque opération a en paramètre 2 nombres de type Double et renvoie le résultat également de type Double.

5.1.1. Window1.xaml & xaml.cs

On trouve dans le dossier de génération tous les fichiers de données XML pour construire l'application, les fichiers de code générés et le fichier csproj.

Les 2 fichiers intéressants ici sont Window1.xaml et Window1.xaml.cs (le fichier de la fenêtre de l'application générée) :

XAML :

```
<Window x:Class="WCFCalculator.Window1" Title="Window1" Height="300" Width="300"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" xmlns:my="clr-
namespace:WCFCalculator.userControls">
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="250*" />
        <RowDefinition Height="50" />
    </Grid.RowDefinitions>
```

Dans cette première partie, on trouve les informations concernant la fenêtre et le Grid avec sa définition pour les lignes (2 ici).

```
<TabControl Grid.Row="0" Name="tabControl1"><TabItem Header="add"
IsSelected="True"><ScrollViewer Name="scrollViewer_1"><StackPanel
CanVerticallyScroll="True"><GroupBox Header="Input"><StackPanel>

    <my:UC_Double_I x:Name="uc_1"> </my:UC_Double_I>


    <my:UC_Double_I x:Name="uc_2"> </my:UC_Double_I>
  </StackPanel></GroupBox><GroupBox Header="Output"><StackPanel>

    <my:UC_Double_O x:Name="uc_3"> </my:UC_Double_O>
  </StackPanel></GroupBox>
<Button Height="23" Width="120" Name="button_1" Click="button_1_Click">Générer
      </Button><TextBlock TextWrapping="WrapWithOverflow" Name="textBlockError_1"
```

```
/></StackPanel></ScrollViewer></TabItem>
```

Dans cette deuxième partie, il y a le TabControl qui va contenir autant de TabItem qu'il y a d'opérations.
Pour chaque TabItem, on trouve un ScrollViewer pour scroller verticalement s'il y a trop d'éléments puis un StackPanel qui va empiler le contenu.
Un GroupBox s'occupe de regrouper les éléments pour la partie input (contenant de nouveau un StackPanel qui contient les UserControls) et un autre GroupBox pour regrouper les éléments de la partie output.
Finalement, un bouton pour lancer l'exécution du traitement.

```
… ICI les 3 autres opérations dans le même style que le tabItem ci-dessus

  </TabControl>
  <ScrollViewer Grid.Row="1" Name="scrollViewerMsgError">
    <StackPanel Grid.Row="1" CanVerticallyScroll="True">
      <TextBlock Name="textBlockError" TextWrapping="WrapWithOverflow" />
    </StackPanel>
  </ScrollViewer>
</Grid>
</Window>
```

Pour terminer la fenêtre, la fin du TabControl et ensuite la place destinée à afficher les messages d'erreur pouvant survenir lors de l'utilisation du web service.

Code behind :

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Runtime.Serialization;
using System.Windows.Shapes;
using System.ServiceModel;
using System.Collections;

namespace WCFCalculator
{
    public partial class Window1 : Window
  {

        Service1Client client_Service1Client = new Service1Client();


      public Window1()
      {
          InitializeComponent();
          init();
      }

      public void init()
      {


  uc_1.ucLabel_info.Content += " - id : 1 - nameMember : a1" ;

  uc_2.ucLabel_info.Content += " - id : 2 - nameMember : a2" ;

  uc_3.ucLabel_info.Content += " - id : 3 - nameMember : " ;

  uc_4.ucLabel_info.Content += " - id : 4 - nameMember : a1" ;
```

```
  uc_5.ucLabel_info.Content += " - id : 5 - nameMember : a2" ;

  uc_6.ucLabel_info.Content += " - id : 6 - nameMember : " ;

  uc_7.ucLabel_info.Content += " - id : 7 - nameMember : a1" ;

  uc_8.ucLabel_info.Content += " - id : 8 - nameMember : a2" ;

  uc_9.ucLabel_info.Content += " - id : 9 - nameMember : " ;

  uc_10.ucLabel_info.Content += " - id : 10 - nameMember : a1" ;

  uc_11.ucLabel_info.Content += " - id : 11 - nameMember : a2" ;

  uc_12.ucLabel_info.Content += " - id : 12 - nameMember : " ;

      }
```

Dans cette première partie, on trouve le code que l'on trouve pour toutes les classes de Window (assemblies, namespace et nom de classe).

Vient ensuite la création de l'objet client qui sera utilisé pour appeler les différentes opérations du web service.

Puis l'initialisation du contenu du label d'information de chaque UserControl (le label ne contient que le nom du type). Il y a 4 opérations qui ont chacune 2 paramètres d'entrée et un retour.

```
private void button_1_Click(object sender, RoutedEventArgs e)
{
  try
  {
  //initialiser les éléments d'output et nettoyage du contenu

  uc_3.cleanUCValue() ;

  String str_3 = "" ;

textBlockError.Text = "" ;


  //Récupération des données saisies

    //primitif | match="parametre/type" mode="input1"
    Double str_1 = Double.Parse(uc_1.getUCValue()) ;

    //primitif | match="parametre/type" mode="input1"
    Double str_2 = Double.Parse(uc_2.getUCValue()) ;



  //Appel du WS et récupération du retour


  Double strO_3 = client_Service1Client.add(str_1,str_2);


  //traitement du retour WS

    //primitif

      if(strO_3 != null)
        str_3 = strO_3.ToString();


  //Afficher le retour du WS dans la fenêtre
```

```
  uc_3.setUCValue(str_3) ;

  }
  catch(Exception ex1)
  {
    textBlockError.Text += ex1.Message + "\n\n";
    textBlockError.Text += ex1.Source + "\n\n";
    textBlockError.Text += ex1.InnerException + "\n\n";
  }
}

… ICI se trouve les 3 autres opérations. Le principe est le même
  }

}
```

Le traitement qui doit être réalisé pour la 1ère opération (addition).
Tout le traitement est effectué dans un try-catch pour récupérer les éventuelles erreurs qui pourraient survenir. Si c'est le cas, le TextBlock en bas de la fenêtre sera rempli.

La 1ère étape est de nettoyer le contenu de l'UserControl qui s'occupe d'afficher la valeur de retour (pour le cas où l'application est utilisée plus d'une fois).

La 2ème étape s'occupe de créer un string pour accueillir la valeur de retour de l'opération. Cette variable string est importante car elle permet de réduire la complexité des templates XSLT (par défaut, quel que soit le type que l'UserControl doit afficher, il attend toujours un string pour mettre à jour le contenu de son TextBox. C'est à l'UserControl ensuite de s'occuper de traiter correctement ce qu'il reçoit). Ensuite, le TextBlock destiné aux messages d'erreurs est nettoyé.

La 3ème étape s'occupe de récupérer le contenu saisi par l'utilisateur et de stocker la valeur dans une variable du type attendu par le web service. Dans ce cas, ce sont des Doubles.
Vient ensuite l'appel au web service et la récupération de la valeur de retour. Convertir le résultat en string puis transmettre à l'UserControl qui s'occupe d'afficher le résultat.

5.1.2. userControls

Ce dossier contient tous les UserControls qui sont utilisés par l'application générée. Dans le cas de ce web service, il n'y a que 2 UC différents : un destiné pour l'input et l'autre pour l'output.

Exemple avec UC_Double_I :

XAML :

```
<UserControl x:Class="WCFCalculator.userControls.UC_Double_I"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
<StackPanel><Label Height="28" Name="ucLabel_info" VerticalAlignment="Top"
HorizontalAlignment="Left" MinWidth="120">Double</Label>

<TextBox Height="23" Name="ucTextBox_saisie" VerticalAlignment="Top"
HorizontalAlignment="Left" MinWidth="120" LostFocus="ucTextBox_saisie_LostFocus"
Background="#FFF8E2F1" />

<Label Height="28" Name="ucLabel_error" VerticalAlignment="Top"
HorizontalAlignment="Left" MinWidth="120" Visibility="Collapsed" Foreground="Red"
/></StackPanel>
```

```
</UserControl>
```

Code behind :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WCFCalculator.userControls
{


  public partial class UC_Double_I : UserControl
  {

    public UC_Double_I()
    {
      InitializeComponent();
    }

    private void ucTextBox_saisie_LostFocus(object sender, RoutedEventArgs e)
    {
      Boolean isError = false ;
      //uniquement pour montrer que l'on peut faire une gestion du contenu saisi
      if (ucTextBox_saisie.Text == "error")
      {
        isError = true ;
        ucLabel_error.Content = "erreur" ;
        ucLabel_error.Visibility =  System.Windows.Visibility.Visible;
//Visibility.Visible ;
      }


      //End traitement erreur
      if(!isError)
      {
        ucLabel_error.Content = "" ;
        ucLabel_error.Visibility = System.Windows.Visibility.Collapsed; //
Visibility.Collapsed ;
      }
    }

    public String getUCValue()
    {
      return ucTextBox_saisie.Text;
    }

    public void setUCValue(String str)
    {
      ucTextBox_saisie.Text = str ;
    }

    public void cleanUCValue()
    {
      ucTextBox_saisie.Text = "";
    }
  }
}
```

5.1.3. temp

Ce dossier contient le proxy en version C# et WCFCalculator.config qui est dans ce cas précis le fichier de configuration de l'application, ces 2 fichiers provenant de l'utilisation de Svcutil qui les a générés.

Le fichier tmpDataUCXmlFile.xml est un fichier temporaire de données XML pour la création des UserControls et est écrasé plusieurs fois lors d'une génération (chaque UserControl créé nécessitant un fichier de données XML légèrement modifié).

Exemple pour le dernier UserControl créé :

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <root>
3. <namespace xclasse="WCFCalculator.userControls.UC_Double_O"
   namespace="WCFCalculator.userControls" classeName="UC_Double_O" />
4. <importUsing type="" />
5. <data type="Double">
6. <testEntry />
7. </data>
8. </root>
```

Ligne 3 :
Namespace et classeName sont destinés pour la création du fichier C#.
Ligne 4 :
Est uniquement spécifié si l'UserControl qui doit être créé est un UC spécial qui est destiné à un type énuméré (sinon, le framework ne pourra pas construire l'application car il ne connaîtra pas ce que contient le type énuméré).
Ligne 5 :
Le type de données concerné.

Le dernier fichier est le fichier Window_xaml.xslt, un fichier temporaire de transformation qui est utilisé pour créer la fenêtre Window de l'application. Il contient par rapport au template Window_xaml_2.xslt l'information sur le namespace que les UserControls utilisent (cette modification est faite en réécrivant le fichier car c'était la solution la plus simple).

5.1.4. Dossier bin et obj

Ce dossier contient le résultat du msbuild (génération d'une application) en mode debug. C'est dans bin\Release que l'on trouve l'exécutable de l'application, la dll du proxy qui est utilisée par l'application et le fichier de configuration .config (qui porte le même nom que l'application.exe).

5.2.Génération wizard avec Enum et Boolean

Pour illustrer cette partie, j'ai pris le web service WCFPersonne (son architecture est détaillée dans la partie Manuel d'utilisation de l'application).
Les types qui sont des énumérations ou Boolean utilisent des UserControls spéciaux.
Pour cette génération, j'ai sélectionné uniquement l'opération getArrayPersonne qui prend en paramètre un tableau de Personnes et retourne un tableau de Personnes modifié.

5.2.1. Window1.xaml & xaml.cs

On trouve dans le dossier de génération tous les fichiers de données XML pour construire l'application, les fichiers de code générés et le fichier csproj.

Pour le fichier XAML de l'affichage, le principe est le même que pour une génération automatique. Des GroupBoxs symbolisent les objets personnalisés et à l'intérieur, on y trouve les UserControls.

Au niveau du code C#, il suit le même principe qu'une génération automatique. Ce qui est intéressant de voir ici sont la gestion des différents tableaux et comment créer les objets nécessaires pour arriver à la signature de l'opération du web service et la façon dont est décomposé le retour de l'opération pour l'afficher. Tout le traitement est effectué dans la méthode de l'événement du bouton.

La première chose qui est faite est l'initialisation des variables String qui vont récupérer les valeurs de retour du web service et l'effacement du contenu des UserControls recevant le retour du web service.

```
… ICI les assemblies, initialisation,

private void button_1_Click(object sender, RoutedEventArgs e)
{
  try
  {
  //initialiser les éléments d'output et nettoyage du contenu
  uc_11.cleanUCValue() ;
  String str_11 = "" ;
  String str_12 = "" ;
  uc_13.cleanUCValue() ;
  String str_13 = "" ;
  uc_14.cleanUCValue() ;
  String str_14 = "" ;
  uc_16.cleanUCValue() ;
  String str_16 = "" ;
  uc_17.cleanUCValue() ;
  String str_17 = "" ;
  uc_18.cleanUCValue() ;
  String str_18 = "" ;
textBlockError.Text = "" ;
…
```

L'étape suivante consiste à récupérer dans tous les UserControls les valeurs saisies par l'utilisateur et de construire les objets jusqu'à arriver à avoir tout à dispoition pour envoyer au web service.

Les commentaires dans le code permettent de savoir quelle partie de template a été utilisé. Les endroits où il n'y a que des commentaires sont des endroits où le contenu n'a pas été traité car il faisait partie d'un tableau d'objets personnalisés qui a déjà été traité.

Exemple de récupération d'une valeur provenant d'un contrôle se trouvant dans un tableau de type personnalisé :

La valeur saisie contient un délimiteur qui permet de spliter les valeurs dans un tableau de String. Le type attendu par le web service est un Int32. Donc, il faut parcourir le tableau de String et convertir chaque entrée en Int32 pour le mettre dans le tableau de Int32.

```
          //custom[] -> primitif | match="parametre/type" mode="input1"

          String[] tmpArray_2 = uc_2.getUCValue().Split(new String[] { ";;" },
System.StringSplitOptions.RemoveEmptyEntries);
          Int32[] strArray_2 = new Int32[tmpArray_2.Length] ;
```

```
            for(int i_2 = 0; i_2 < strArray_2.Length; i_2++)
            {
              strArray_2[i_2] = int.Parse(tmpArray_2[i_2]) ;
            }
            cptStrArray_1 = strArray_2.Length ;
…
```

Exemple de récupération d'une valeur provenant d'un contrôle se trouvant dans un tableau de type personnalisé lui-même se trouvant dans un autre tableau de type personnalisé :

Le premier split s'occupe du tableau de Personne. Ensuite est créée une liste de tableau de Boolean et une liste d'int pour la taille des différents tableaux (de type Ami).
Le traitement de récupération commence avec la première boucle for pour le tableau de Personne. Pour chaque entrée, il faut récupérer le contenu et créer un tableau de String temporaire (qui est en fait le tableau d'Amis). Il faut parcourir ce tableau temporaire pour convertir chaque valeur en un Boolean. A chaque fois qu'un tableau d'Ami est prêt, il faut le stocker dans la liste de tableau Boolean.

```
…
  //custom[] -> custom[] -> primitif
  String[] tmp2Array_7 = uc_7.getUCValue().Split(new String[] { "};;{" },
System.StringSplitOptions.RemoveEmptyEntries);
  List<Boolean[]> listArray_7 = new List<Boolean[]>();

  List<int> cpt_listArray_7 = new List<int>();

  for (int i_7 = 0; i_7 < tmp2Array_7.Length; i_7++)
  {
    String[] tmpArray_7 = tmp2Array_7[i_7].Split(new String[] { ";;" },
System.StringSplitOptions.RemoveEmptyEntries);
    Boolean[] strTmpArray_7 = new Boolean[tmpArray_7.Length];

    for (int j_7 = 0; j_7 < tmpArray_7.Length; j_7++)
    {
      if(tmpArray_7[j_7].IndexOf("{",0) == 0)
                  {
                    tmpArray_7[j_7] = tmpArray_7[j_7].Substring(1);
                  }

                  if(tmpArray_7[j_7].IndexOf("}",tmpArray_7[j_7].Length - 1) ==
tmpArray_7[j_7].Length - 1)
                  {
                    tmpArray_7[j_7] =
tmpArray_7[j_7].Substring(0,tmpArray_7[j_7].Length - 1);
                  }
      strTmpArray_7[j_7] = Boolean.Parse(tmpArray_7[j_7]) ;
    }
    listArray_7.Add(strTmpArray_7);
    cpt_listArray_7.Add(strTmpArray_7.Length);
  }

  cptStrArray_1 = tmp2Array_7.Length;
…
```

Une fois que tous les types ont été traités, il n'y a plus qu'à construire les objets et y mettre les données membres.
Je commence toujours de l'objet le plus haut (parent) pour arriver à l'objet le plus bas (enfant). Le for s'occupe de traiter le tableau de Personne. A chaque fois, un objet Personne est créé et les membres sont renseignés.

La 2ème boucle for s'occupe du tableau d'Amis. Le principe est le même, un objet de type Ami est créé, rempli et stocké dans une liste.
A la fin, la liste est mise dans le bon membre de l'objet Personne.

Une fois tout le traitement fini, il n'y a plus qu'à utiliser l'opération du web service.

```
…
  for(int i_1 = 0; i_1 < cptStrArray_1; i_1++)
  {
  Personne custType_1 = new Personne();

//inputArrayCust : primitif
custType_1.Age = strArray_2[i_1] ;

//inputArrayCust : Enum
custType_1.Genre = strArray_3[i_1] ;

//inputArrayCust : primitif
custType_1.Nom = strArray_4[i_1] ;

//inputArrayCust : primitif
custType_1.Prenom = strArray_5[i_1] ;

//inputArrayCust : custom[]

   for(int j_6 = 0; j_6 < cpt_listArray_7.ElementAt(i_1); j_6++)

{
  Ami custType_6 = new Ami();


        //inputArrayCust : custom[] -> primitif
        custType_6.AmiDenfance = listArray_7.ElementAt(i_1)[j_6] ;

        //inputArrayCust : custom[] -> primitif
        custType_6.Nom = listArray_8.ElementAt(i_1)[j_6] ;

        //inputArrayCust : custom[] -> primitif
        custType_6.Prenom = listArray_9.ElementAt(i_1)[j_6] ;


  listType_6.Add(custType_6);
}
custType_1.SesAmis = listType_6.ToArray();
listType_6.Clear();

  listType_1.Add(custType_1);
  }

  Personne[] arrayType_1 = listType_1.ToArray();

//primitif | match="parametre/type" mode="input1"

//Enum | match="parametre/type" mode="input1"

//primitif | match="parametre/type" mode="input1"

//primitif | match="parametre/type" mode="input1"

//custom[] | match="parametre/type" mode="input1"

//primitif | match="parametre/type" mode="input1"

//primitif | match="parametre/type" mode="input1"

//primitif | match="parametre/type" mode="input1"

//Appel du WS et récupération du retour
```

```
Personne[] arrayType_10 = client_Service1Client.getArrayPersonne(arrayType_1);
…
```

Le traitement qui reste à faire est de transformer le retour de l'opération pour afficher les valeurs dans les différents UserControls.

```
…
//traitement du retour WS
//custom[]

List<Personne> listType_10 = arrayType_10.ToList() ;

for(int i_10 = 0 ; i_10 < listType_10.Count; i_10++)
{
Personne custType_10 = listType_10.ElementAt(i_10);

    // custom[] -> primitif
    str_11 += custType_10.Age ;

    if(i_10 != listType_10.Count -1)
    {
      str_11 += ";;";
    }

    // custom[] -> enum
    str_12 += custType_10.Genre ;

    if(i_10 != listType_10.Count -1)
    {
      str_12 += ";;";
    }

    // custom[] -> primitif
    str_13 += custType_10.Nom ;

    if(i_10 != listType_10.Count -1)
    {
      str_13 += ";;";
    }


    // custom[] -> primitif
    str_14 += custType_10.Prenom ;

    if(i_10 != listType_10.Count -1)
    {
      str_14 += ";;";
    }

    //custom[] -> cutom[]

    List<Ami> listType_15 = custType_10.SesAmis.ToList() ;

        str_16 += "{";

        str_17 += "{";

        str_18 += "{";

      for(int j_15 = 0 ; j_15 < listType_15.Count; j_15++)
    {
      Ami custType_15 = listType_15.ElementAt(j_15);

          //custom[] -> cutom[] -> primitif
          str_16 += custType_15.AmiDenfance ;

          if(j_15 != listType_15.Count -1)
          {
```

```
            str_16 += ";;";
          }

          //custom[] -> cutom[] -> primitif
          str_17 += custType_15.Nom ;

          if(j_15 != listType_15.Count -1)
          {
            str_17 += ";;";
          }

          //custom[] -> cutom[] -> primitif
          str_18 += custType_15.Prenom ;

          if(j_15 != listType_15.Count -1)
          {
            str_18 += ";;";
          }
    }

        str_16 += "}";

        if(i_10 != listType_10.Count -1)
        {
        str_16 += ";;";
        }

        str_17 += "}";

        if(i_10 != listType_10.Count -1)
        {
        str_17 += ";;";
        }

        str_18 += "}";

        if(i_10 != listType_10.Count -1)
        {
        str_18 += ";;";
        }

      }

    //primitif

    //Enum

    //primitif

    //primitif

    //custom[]


    //primitif

    //primitif

    //primitif

  //Afficher le retour du WS dans la fenêtre

  uc_11.setUCValue(str_11) ;

  uc_12.setUCValue(str_12) ;

  uc_13.setUCValue(str_13) ;

  uc_14.setUCValue(str_14) ;
```

```
  uc_16.setUCValue(str_16) ;

  uc_17.setUCValue(str_17) ;

  uc_18.setUCValue(str_18) ;
…
```

5.2.2. userControls

Ce dossier contient tous les UserControls qui sont utilisés par l'application générée. Contrairement à la génération automatique, chaque type à son propre UserControl.

Le code pour les UserControls standards (avec un TextBox) étant identiques à ceux utilisés pour la génération commune, je ne m'y attarderai pas. Par contre, il est intéressant de regarder le contenu C# des UserControls spéciaux pour ce web service.

5.2.2.1. Input

UC_Genre_I_id3_ENU_S.xaml.cs :
Cet UserControl est contenu dans un tableau de Personne qui a une taille de 2.

```
… liste des usings …

namespace WCFPersonne.userControls
{
  using WcfService1;

  public partial class UC_Genre_I_id3_ENU_S : UserControl
  {
    String[] enumStrArray = Enum.GetNames(typeof(Genre));

    public UC_Genre_I_id3_ENU_S()
    {
      InitializeComponent();
      initUC_une();
    }
    …
```

La première chose que l'on remarque, c'est la présence dans le namespace d'une référence WcfService1. Cette référence est importante ici car sans elle, il ne serait pas possible de connaître le type énuméré Genre.
Lors de l'initialisation de l'UserControl, il faut appeler une méthode chargée de créer les contrôles ComboBox :

```
…
    private void initUC_une()
    {
        for (int i = 0; i < enumStrArray.Length; i++)
        {
          ComboBoxItem cbi = new ComboBoxItem();
                cbi.Content = enumStrArray[i];
                ucComboBox_saisie_0.Items.Add(cbi);
        }

        for (int i = 0; i < enumStrArray.Length; i++)
        {
          ComboBoxItem cbi = new ComboBoxItem();
                cbi.Content = enumStrArray[i];
                ucComboBox_saisie_1.Items.Add(cbi);
        }
    }
    …
```

La transformation XSLT s'est occupée d'écrire le code pour mettre 2 fois le contrôle (car le tableau parent est de taille 2). Pour chaque ComboBox, il faut parcourir le tableau contenant toutes les énumérations pour remplir les items.

```
…
    public String getUCValue()
    {
      String returnValue = "";
      ComboBoxItem cbi = null ;

        cbi = (ComboBoxItem)ucComboBox_saisie_0.SelectedItem;
        returnValue += cbi.Content.ToString() + ";;";

        cbi = (ComboBoxItem)ucComboBox_saisie_1.SelectedItem;
        returnValue += cbi.Content.ToString() + ";;";

          returnValue = returnValue.Substring(0, returnValue.Length - 2);

      return returnValue;
    }

    public void cleanUCValue()
    {
      //Do nothing
    }
  }
}
```

Comme cet UserControl n'est destiné qu'à l'input, il y a juste la méthode getUCValue qui permet de récupérer le contenu des contrôles. Il faut récupérer la valeur des index sélectionnés dans les ComboBoxs (séparé par ;;) et retourner la valeur.

UC_Boolean_I_id7_B_S.xaml.cs :
Cet UserControl est contenu dans 2 tableaux personnalisés. Le premier est le tableau de Personnes de taille 2. Pour Personne[0], il y a un tableau d'Amis de taille 3 et pour Personne[1], un tableau d'Amis de taille 5.

```
… liste des usings …

namespace WCFPersonne.userControls
{
  public partial class UC_Boolean_I_id7_B_S : UserControl
  {

    public UC_Boolean_I_id7_B_S()
    {
    InitializeComponent();
    }


    public String getUCValue()
    {
      String returnValue = "";

        returnValue += "{";

          returnValue += ucCheckBox_saisie_0_0.IsChecked + ";;"  ;

          returnValue += ucCheckBox_saisie_0_1.IsChecked + ";;"  ;

          returnValue += ucCheckBox_saisie_0_2.IsChecked + ";;"  ;

        returnValue = returnValue.Substring(0, returnValue.Length - 2);
        returnValue += "}";
```

```
            returnValue += ";;";

          returnValue += "{";

            returnValue += ucCheckBox_saisie_1_0.IsChecked + ";;"  ;

            returnValue += ucCheckBox_saisie_1_1.IsChecked + ";;"  ;

            returnValue += ucCheckBox_saisie_1_2.IsChecked + ";;"  ;

            returnValue += ucCheckBox_saisie_1_3.IsChecked + ";;"  ;

            returnValue += ucCheckBox_saisie_1_4.IsChecked + ";;"  ;

          returnValue = returnValue.Substring(0, returnValue.Length - 2);
          returnValue += "}";


        return returnValue;
      }

      public void cleanUCValue()
      {
        //Do nothing
      }
    }
}
```

Il n'y a qu'une méthode présente qui sert à récupérer le contenu des différents contrôles présents. Le code a été généré grâce à la transformation XSLT suivant ce que l'utilisateur a saisi comme taille pour les différents tableaux.

5.2.2.2. Output

UC_Genre_O_id12_ENU_S.xaml.cs :
La différence entre un UserControl pour l'input et l'ouput réside dans le fait que pour l'output, on ne connaît pas ce que l'on va recevoir au niveau de la taille des tableaux. C'est pourquoi, pour les UserControls spéciaux, la création des éléments graphiques est effectué en C#.

```
… liste des usings …

namespace WCFPersonne.userControls
{
  using WcfService1;

  public partial class UC_Genre_O_id12_ENU_S : UserControl
  {
    String[] enumStrArray = Enum.GetNames(typeof(Genre));

    public UC_Genre_O_id12_ENU_S()
    {
      InitializeComponent();
    }
…
```

Cette création est effectuée dans la méthode setUCValue. C'est les données qui vont déterminer combien de fois il faut mettre un ComboBox. Cette information est trouvée en splitant le string pour obtenir un tableau temporaire. Pour chaque entrée du tableau, je crée un contrôle et je remplis ses items.
Il ne reste plus qu'à sélectionner la valeur pour chaque contrôle.

```
…
    public void setUCValue(String str)
```

```
    {
      stackPanel_uc1.Children.Clear();


      ComboBox cb = null ;

        String[] tmpArray = str.Split(new String[] { ";;" },
System.StringSplitOptions.RemoveEmptyEntries);

        for(int i = 0; i < tmpArray.Length; i++)
        {
          cb = new ComboBox();
          cb.Name = "";

            for (int j = 0; j < enumStrArray.Length; j++)
            {
                ComboBoxItem cbi = new ComboBoxItem();
                cbi.Content = enumStrArray[j];
                cb.Items.Add(cbi);
            }

          ItemCollection ic = cb.Items;
          stackPanel_uc1.Children.Add(cb);

          int cpt = 0;
          Boolean isSet = false;

          while (cpt < ic.Count && !isSet)
          {
              ComboBoxItem cbi = (ComboBoxItem)ic.GetItemAt(cpt);

              if (cbi.Content.ToString() == tmpArray[i])
              {
                  cb.SelectedIndex = cpt;
                  isSet = true;
              }
              cpt++;
          }
        }
    }

    public void cleanUCValue()
    {
      //Do nothing
    }
  }
}
…
```

UC_Boolean_O_id16_B_S.xaml.cs :
Ici également, la création des contrôles se fait côté C#.
La transformation a construit le code en prenant en compte qu'il y a 2 tableaux personnalisés : un tableau de Personnes et pour chaque entrée de ce tableau, un tableau d'Amis contenant le type à traiter.
La liste de type Boolean boolList va s'occuper de stocker toutes les valeurs Boolean pour les différents tableaux. Dans ce web service, ce qui est envoyé au web service est retourné avec le même nombre de variables. Si l'on envoi un tableau de Personne de taille 2 et que Personne[0] contient un tableau de 3 Amis et Personne[1] un tableau de 5 Amis, on obtient au total 8 variables Boolean. Cette liste s'occupe de stocker ces 8 valeurs.

Ensuite, le string reçu est splité une 1ère fois (c'est le tableau de Personnes). Une liste de type int cptList1 est créée. Cette liste va s'occuper de stocker la taille de chaque tableau d'Amis.

Le tableau de Personnes est parcouru. Pour chaque entrée, il faut spliter le contenu pour avoir le tableau d'Amis. La taille du tableau d'Amis est stockée dans la liste d'int. Chaque entrée du tableau d'Amis est convertie en Boolean et stocké dans la liste de Boolean.

Les deux listes sont transformées en tableau. Il ne reste plus qu'à créer les contrôles. Pour chaque entrée du tableau de Personne, un Label est créé puis le contenu du tableau est mis en mettant le bon nombre de CheckBoxs.

```
… liste des usings …

namespace WCFPersonne.userControls
{
  public partial class UC_Boolean_O_id16_B_S : UserControl
  {

    public UC_Boolean_O_id16_B_S()
    {
    InitializeComponent();
    }


    public void setUCValue(String str)
    {
      stackPanel_uc1.Children.Clear();

      int cpt = 0;
      List<Boolean> boolList = new List<Boolean>() ;
      String[] tmpArray = str.Split(new String[] { "};;{" },
System.StringSplitOptions.RemoveEmptyEntries);

      List<int> cptList1 = new List<int>() ;

        for(int i = 0; i < tmpArray.Length; i++)
        {
          String[] tmp2Array = tmpArray[i].Split(new String[] { ";;" },
System.StringSplitOptions.RemoveEmptyEntries);
          cptList1.Add(tmp2Array.Length);

            for(int j = 0; j < tmp2Array.Length; j++)
            {
              tmp2Array[j] = tmp2Array[j].Replace("{", "");
              tmp2Array[j] = tmp2Array[j].Replace("}", "");

              boolList.Add(Boolean.Parse(tmp2Array[j])) ;
            }
        }

        Boolean[] boolArray = boolList.ToArray<Boolean>() ;
        int[] cptArray1 = cptList1.ToArray<int>() ;


        //faire une boucle pour ajouter les éléments de manière automatique au UC

        for(int i=0; i < tmpArray.Length; i++)
        {
          Label lb = new Label();
          lb.Content = "tbl [" + i + "]" ;
          stackPanel_uc1.Children.Add(lb);

          for(int j=0; j < cptArray1[i]; j++)
          {
            CheckBox cb = new CheckBox();
            cb.Name = "ucCheckBox_saisie_" + i + "_" + j;
            cb.IsChecked = boolArray[cpt++];
            cb.Content = "tbl [" + i + "," + j +"]";
            stackPanel_uc1.Children.Add(cb);
```

```
            }
        }
    }

    public void cleanUCValue()
    {
      //Do nothing
    }
  }
}
```

5.2.3. temp

Ce dossier contient les mêmes fichiers décrits dans la partie génération automatique. On y trouve 1 fichier et un dossier supplémentaires :
Le fichier tmpDataSpecialWizardUCXmlFile.xml est un fichier de données XML temporaire destiné à convertir les informations de taille des tableaux que l'utilisateur a saisies pour pouvoir ensuite être utilisées pour effectuer la transformation XSLT des UserControls spéciaux. Ce fichier est écrasé au fur et à mesure des différents UserControls qui doivent être transformés.
Le dossier userControls s'occupe de stocker tous les UserControls que l'application génère en vrac avant que l'utilisateur choisisse ceux qu'il souhaite utiliser. Une fois que l'utilisateur a validé ses choix, tous les UserControls à utiliser sont retirés de ce dossier.

6. Assurance qualité

Tout au long du développement, différents tests ont été effectués afin d'identifier les bugs et si possible les corriger. Une grande partie des tests est constituée de web services WCF que j'ai créés afin de pouvoir tester la récupération des informations du WSDL ainsi que la génération de code traitée par les templates de transformation XSLT.
Des web services issus d'Internet ont également été utilisés pour tester les mêmes points que les web services que j'ai développés mais également pour mettre en lumière des cas de figures auxquels je n'avais pas pensés.

6.1. WS développés

Les web services listés ci-dessous sont disponibles sur le cd.
Chaque web service est constitué de son nom en gras et sur la ligne suivante, l'opération et sa signature.
Le retour du web service est à gauche suivi du nom de l'opération puis finalement des paramètres de l'opération entre parenthèses.

1. **WcfService1**
   int wcfs1(string, int)
2. **WcfService2**
   int[] wcfs2(string[], int[])
3. **WcfService3**
   string wcfs3(string, int[])
4. **WcfService4**
   Custom wcfs4(Custom(string, int))
5. **WcfService5**
   Custom wcfs5(Custom(string, Int32[]))
6. **WcfService6**
   Custom wcfs6(Custom(string, Custom2(string, Int32[])))
7. **WcfService7**
   Custom wcfs7(Custom(string[], Custom2(string, Int32[])))

8. **WcfService8**
   Custom wcfs8(Custom(string[], Boolean[], Custom2(string[], Double[])))
9. **WcfService9**
   Custom[] wcfs9(Custom[](string, Int32)
10. **WcfService10**
    Custom[] wcfs10(Custom[](string, Int32[])
11. **WcfService11**
    Custom[] wcfs11(Custom[](string, Custom2(string, Double))
12. **WcfService12**
    Custom wcfs12(Custom(string, Custom2[](string, Double)))
13. **WcfService13**
    Custom wcfs13(Custom(string, Custom2[](string[], Double)))
14. **WcfService14**
    Custom wcfs14(Custom(string, Double, Custom2 (string, Double, Costum3(string, double))))
15. **WcfService15**
    Custom wcfs15(Custom(string[], Double, Custom2 (string[], Double, Costum3(string[], double))))
16. **WcfService16**
    Custom wcfs16(Custom[](string[], Custom2[](Double)))
17. **WcfService17**
    Custom wcfs17(Custom[](string, Double, Custom2[](Double, string)))
18. **WcfService18**
    Custom wcfs18(Custom[](string[], Double[], Custom2[](Double, string)))
19. **WcfService19**
    Custom wcfs19(Custom[](string[], Custom2[](Double[])))
20. **WcfService20**
    Custom wcfs20(Custom[](Custom2[](Double)))
21. **WcfService21**
    Custom wcfs21(Custom[](string, Custom2[](Double[], String[])))
22. **WcfService22**
    Custom wcfs22(Custom[](string, Custom2[](Double[], String[], Boolean)))
23. **WcfService23**
    Custom wcfs23(Custom[](string, Custom2[](Custom3(String))))
24. **WcfService24**
    Custom wcfs24(Custom[](string, Custom2[](Custom3(String, int, Boolean))))
25. **WcfService25**
    Custom wcfs25(Custom[](string, Custom2[](Custom3(String[]))))
26. **WcfService26**
    String wcfs26()
27. **WcfService27**
    Custom wcfs27(Custom[](string, Custom2(string. Custom3(int))))
28. **WcfService28**
    Custom wcfs28(Custom[](string, Custom2(string. Custom3(int[]))))
29. **WcfService29**
    Custom wcfs29(Custom[](string, Custom2(string[]. Custom3(int[]))))
30. **WcfService30**
    Custom wcfs30(Custom[](string, Custom2(string[]. Custom3[](string))))
31. **WcfService31**
    Custom wcfs31(Custom[](string, Custom2(string[], Custom3[](string[]))))
32. **WcfService32**
    Custom wcfs32(Custom (string, Custom2(string, Custom3[](string, Double))))

33. **WcfService33**
    Custom wcfs33(Custom (string, Custom2(string, Custom3[](string[], Double[]))))
34. **WcfService34**
    Enum1 wcfs34(string, Enum1)
35. **WcfService35**
    Custom wcfs35(Custom(string, Enum1))
36. **WcfService36**
    Custom wcfs36(Custom(string, Enum1, Custom2(string, Enum2)))
37. **WcfService37**
    Custom[] wcfs37(Custom[](string, Enum1))
38. **WcfService38**
    Custom[] wcfs38(Custom[](string, Custom2(string, Enum1)))
39. **WcfService39**
    Custom[] wcfs39(Custom[](string, Custom2(string, Custom3(string, Enum1))))
40. **WcfService40**
    Custom[] wcfs40(Custom[](string, Custom2(string, Custom3[](string, Enum1))))
41. **WcfService41**
    Custom[] wcfs41(Custom[](string, Custom2[](string, Enum1)))
42. **WcfService42**
    Custom[] wcfs42(Custom[](string, Custom2[](string, Custom3(string, Enum1))))
43. **WcfService43**
    Enum1[] wcfs43(string, Enum1[])
44. **WcfService44**
    Custom wcfs44(Custom(string, Enum1[]))
45. **WcfService45**
    Custom wcfs45(Custom(string, Enum1[], Custom2(string, Enum1[])))
46. **WcfService46**
    Custom wcfs46(Custom[](string, Enum1[]))
47. **WcfService47**
    Custom[] wcfs47(Custom[](string, Custom2(string, Custom3(string, Enum1[]))))
48. **WcfService48**
    Custom[] wcfs48(Custom[](string, Custom2(string, Custom3(string, Enum1[]))))
49. **WcfService49**
    Custom[] wcfs49(Custom[](string, Custom2(string, Custom3[](string, Enum1[]))))
50. **WcfService50**
    Custom[] wcfs50(Custom[](string, Custom2[](string, Enum1[]))
51. **WcfService51**
    Custom[] wcfs51(Custom[](string, Custom2[](string, Custom3(string, Enum1[]))))
52. **WcfService52**
    Boolean wcfs52(Boolean, Boolean[])
    Boolean[] wcfs52A(Boolean, Boolean[])
    Double[] wcfs52B(Int32, Double[])
    Boolean wcfs52C(Boolean)
    int wcfs52a1(int)
    String wcfs52a2(string)
    Double wcfs52a3(Double)
    Double[] wcfs52D(Double[])
    String[] wcfs52E(String[])
    Boolean[] wcfs52D(Boolean[])
53. **WcfService53**
    Custom wcfs53(Custom(Boolean, int)

54. **WcfService54**
    Custom wcfs54(Custom(Boolean, Boolean[])
55. **WcfService55**
    Custom wcfs55(Custom(Enum1, Enum1[])
56. **WcfService56**
    Custom[] wcfs56(Custom[](Boolean, Boolean[], Custom2[](Boolean, Boolean[]))
57. **WcfService57**
    Boolean wcfs57(Boolean)
58. **WcfService58**
    Custom[] wcfs58(Boolean, Custom[](Boolean))
59. **WcfService59**
    Custom[] wcfs59(Boolean, Custom[](Boolean, Custom2[](Boolean)))
60. **WcfService60**
    Boolean[] wcfs60(Boolean[])
61. **WcfService61**
    Custom[] wcfs61(Custom[](Boolean[]))
62. **WcfService62**
    Custom[] wcfs62(Custom[](Boolean[], Custom2[](Boolean[])))
63. **WcfService63**
    Custom[] wcfs63(Custom[](Boolean, Custom2 (Boolean, Custom3[](Boolean))))
64. **WcfService64**
    Enum1 wcfs64(string, Enum1)
65. **WcfService65**
    Custom[] wcfs65(Custom[](Enum1))
66. **WcfService66**
    Custom[] wcfs66(Custom[](Enum1, Custom2[](Enum1))
67. **WcfService67**
    Enum1[] wcfs67(Enum1[])
68. **WcfService68**
    Custom[] wcfs68(Custom[](Enum1[]))
69. **WcfService69**
    Custom[] wcfs69(Custom[](Custom2[](Enum1[])))
70. **WcfService70**
    Custom[] wcfs70(Custom[](string, Custom2 (string[],
    Custom3[](string[]))),Custom[](string, Custom2 (string[], Custom3[](string[]))))

6.2.WS Internet

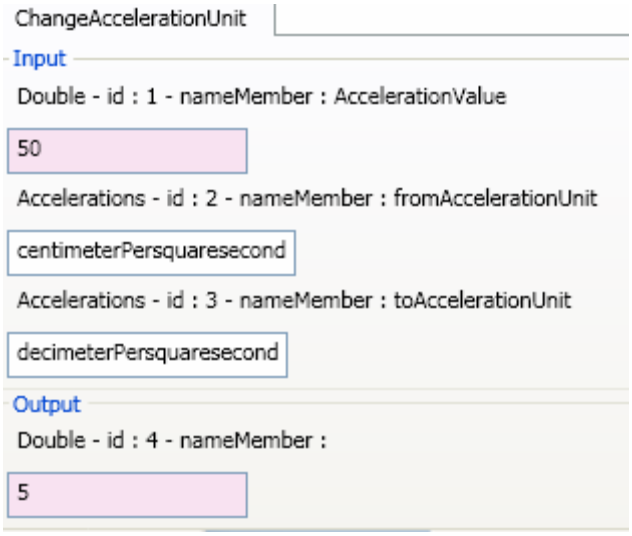
6.2.1. Acceleration Unit Convertor

Convertir une valeur dans une certaine unité d'accélération dans une autre.
Source : http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=53
WSDL :http://www.webservicex.net/ConvertAcceleration.asmx?wsdl

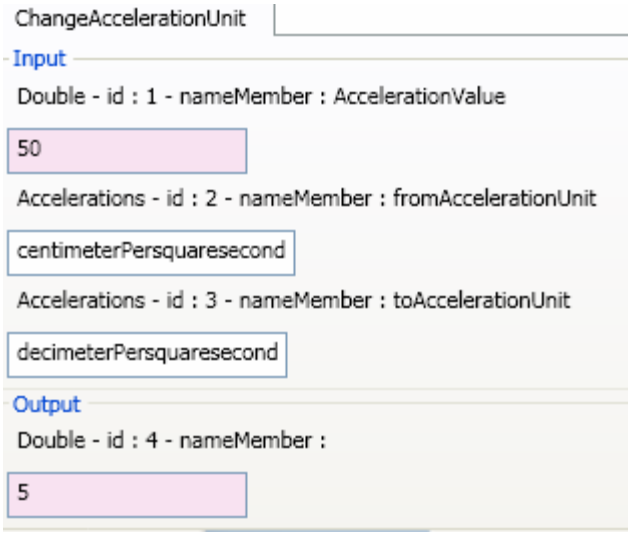
Exemple :
Convertir 50 centimeterPersquaresecond en decimeterPersquaresecond.
Résultat attendu : 5

Rapport d'erreur :
Erreur SvcUtil binding http get et post : erreur non fatale, le binding soap fonctionne

Résultat génération automatique :
Génération de l'application : réussie
Utilisation de l'application : réussie



Résultat génération wizard :
Génération de l'application : réussie
Utilisation de l'application : réussie



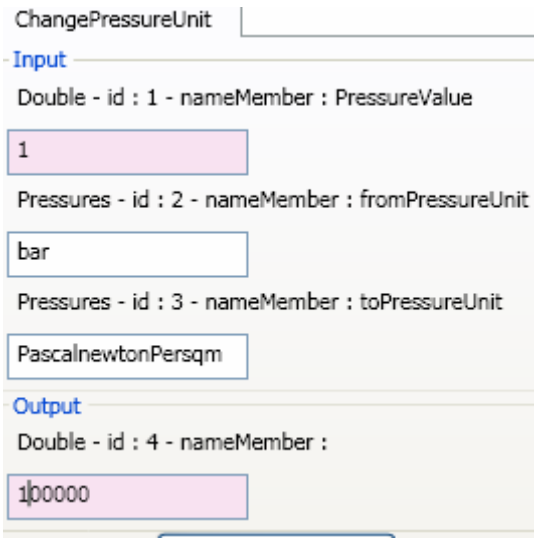
6.2.2. Pressure

Convertir une valeur dans une autre unité de mesure de pression.
Source : http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=54
WSDL : http://www.webservicex.net/CovertPressure.asmx?wsdl

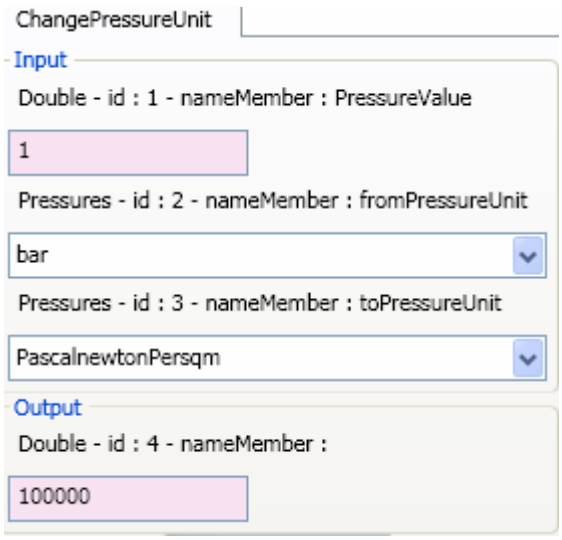
Exemple :
Convertir 1 bar en Pascal (PascalnewtonPersqm)
Résultat attendu : 100000

Rapport d'erreur :
Erreur svcutil binding http get et post : erreur non fatale, le binding soap fonctionne

Résultat génération automatique :
Génération de l'application : réussie
Utilisation de l'application : réussie



Résultat génération wizard :
Génération de l'application : réussie
Utilisation de l'application : réussie



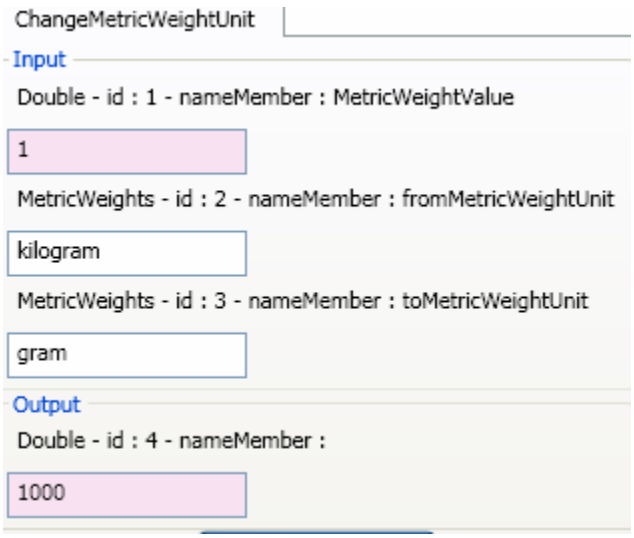
6.2.3. Metric Weight

Convertir une valeur dans une autre unité de mesure de poids.
Source : http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=59
WSDL : http://www.webservicex.net/convertMetricWeight.asmx?wsdl

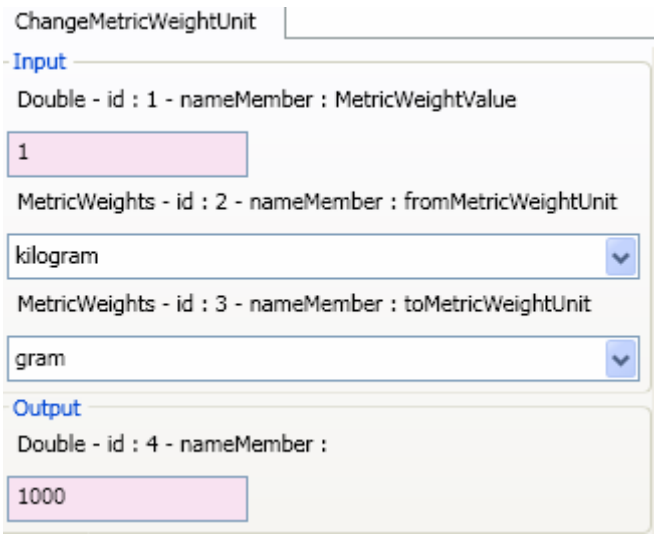
Exemple :
Convertir 1 kilo (kilogram) en gramme (gram)
Résultat attendu : 1000

Rapport d'erreur :
Erreur svcutil binding http get et post : erreur non fatale, le binding soap fonctionne

Résultat génération automatique :
Génération de l'application : réussie
Utilisation de l'application : réussie



Résultat génération wizard :
Génération de l'application : réussie
Utilisation de l'application : réussie



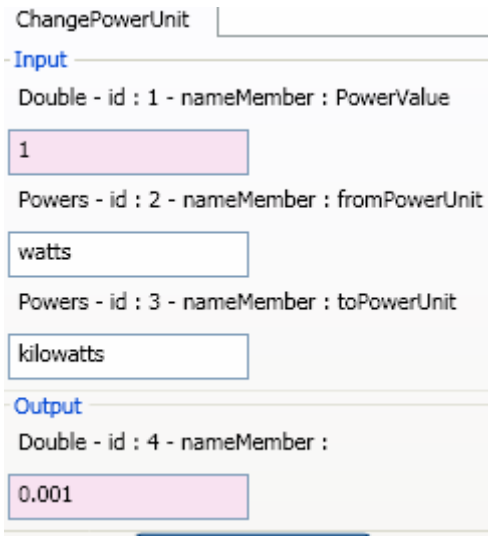
6.2.4. Power

Convertir une valeur dans une autre unité de d'énergie.
Source : http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=56
WSDL : http://www.webservicex.net/ConverPower.asmx?wsdl

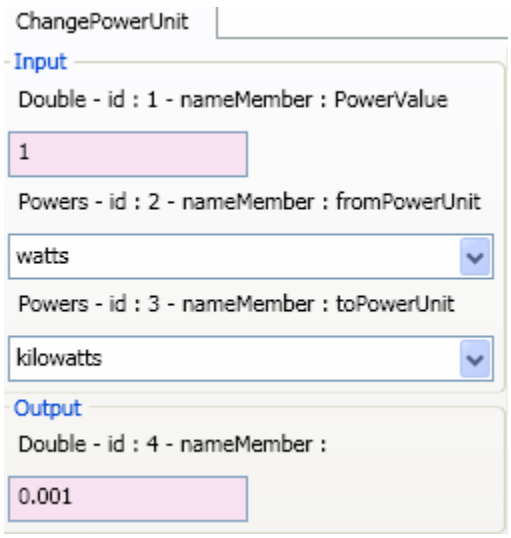
Exemple :
Convertir 1 watt (watts) en kilowatt (kilowatts)
Résultat attendu : 0.001

Rapport d'erreur :
Erreur svcutil binding http get et post : erreur non fatale, le binding soap fonctionne

Résultat génération automatique :
Génération de l'application : réussie
Utilisation de l'application : réussie



Résultat génération wizard :
Génération de l'application : réussie
Utilisation de l'application : réussie



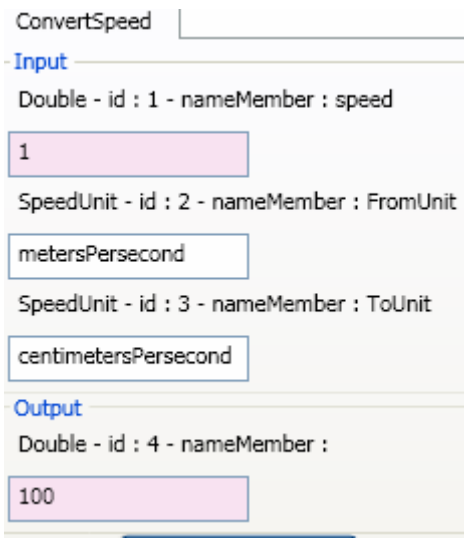
6.2.5. Speed

Convertir une valeur dans une autre unité de vitesse.
Source : http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=70
WSDL : http://www.webservicex.net/ConvertSpeed.asmx?wsdl

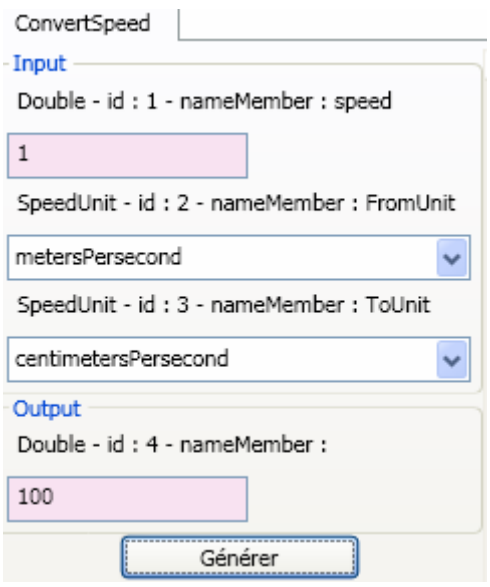
Exemple :
Convertir 1 mètre/seconde (metersPersecond) en centimètre/seconde (centimetersPersecond)
Résultat attendu : 100

Rapport d'erreur :
Erreur svcutil binding http get et post : erreur non fatale, le binding soap fonctionne

Résultat génération automatique :
Génération de l'application : réussie
Utilisation de l'application : réussie



Résultat génération wizard :
Génération de l'application : réussie
Utilisation de l'application : réussie



6.2.6. Real Time Market Data

Web service permettant de récupérer des informations de la bourse.
Source : http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=7
WSDL : http://www.webservicex.net/RealTimeMarketData.asmx?wsdl

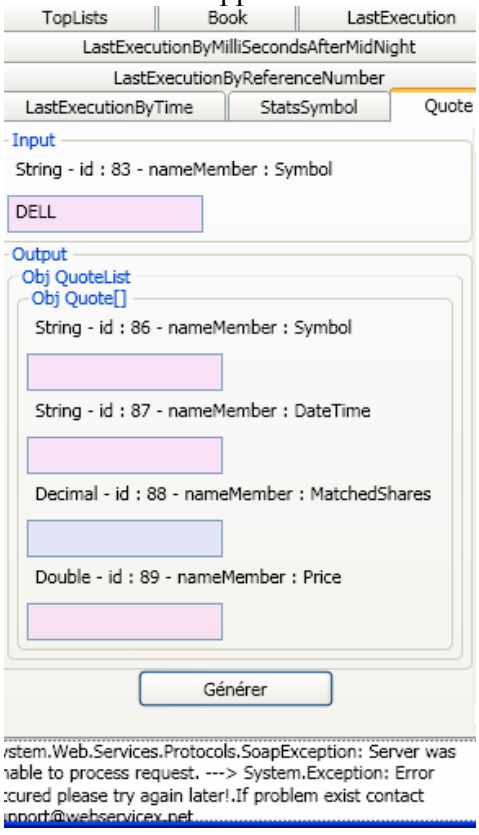
Le web service ne semble pas bien fonctionner. Par exemple, en utilisant l'opération Quote qui à partir d'un symbole permet de récupérer le nom du symbole, la date, les actions et les prix, j'obtiens le message d'erreur suivant : System.Web.Services.Protocols.SoapException: Server was unable to process request. ---> System.Exception: Error occured please try again later!.If problem exist contact support@webservicex.net
at RealTimeQuote.RealTimeMarketData.Quote(String Symbol)
--- End of inner exception stack trace ---

J'ai également testé le web service en créant un client à la main dans Visual Studio et j'ai obtenu le même message d'erreur. Je pense que l'erreur vient du serveur. Si les symboles que j'ai utilisés n'existent pas, je pense que j'aurai dû recevoir un autre message.

Rapport d'erreur :
Erreur svcutil binding http get et post : erreur non fatale, le binding soap fonctionne

Résultat génération automatique :
Génération de l'application : réussie
Utilisation de l'application : échec

Résultat génération wizard :
Génération de l'application : réussie
Utilisation de l'application : échec



6.2.7. Statistics

Effectuer des opérations de statistiques.
Source : http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=11
WSDL : http://www.webservicex.net/Statistics.asmx?wsdl

Rapport d'erreur :
Erreur svcutil binding http get et post : erreur non fatale, le binding soap fonctionne.

Problème de génération du fichier XAML pour l'interface utilisateur. Le nom de certains types contiennent & (Double& au lieu de Double).
Ca donne comme résultat que la construction des GroupBoxs s'effectue mal. Le type devrait être connu, mais comme il y a & à la fin, il est considéré comme n'étant pas un type standard et cela entraine la création de GroupBox (alors que ça ne devrait être qu'uniquement un userControl) et ces GroupBoxs en plus ne sont jamais refermés (ce qui donne une erreur lors de la génération avec msbuild).

A noter que sous Visual Studio, lors de la création d'un client manuellement, lors de l'ajout de la référence, il n'arrive pas à ajouter la référence du service (ce qui est normal vu le problème).

Résultat génération automatique :
Génération de l'application : échec
Utilisation de l'application : échec

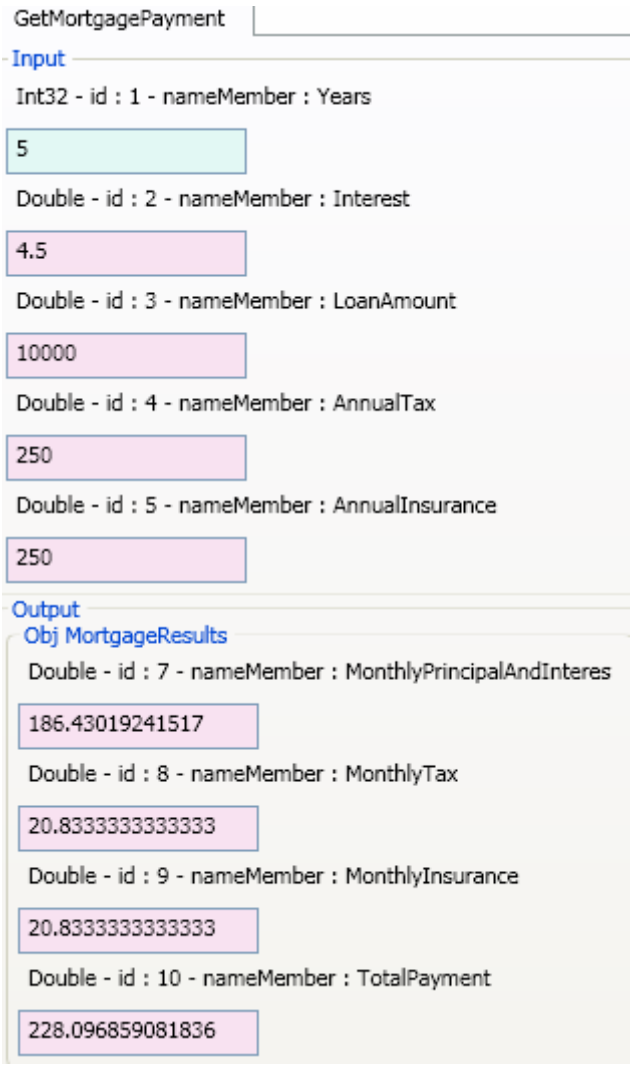
6.2.8. Mortgage Web service

Combien une hypothèque coûte par mois.
Source : http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=13
WSDL : http://www.webservicex.net/mortgage.asmx?wsdl

Exemple :
Un prêt de 10'000 pour une durée de 5 ans à un taux de 4.5% avec une taxe annuelle de 250 et une assurance annuelle de 250.

Rapport d'erreur :
Erreur svcutil binding http get et post : erreur non fatale, le binding soap fonctionne

Résultat génération automatique :
Génération de l'application : réussie
Utilisation de l'application : réussie



Résultat génération wizard :
L'affichage est le même que pour la génération automatique.

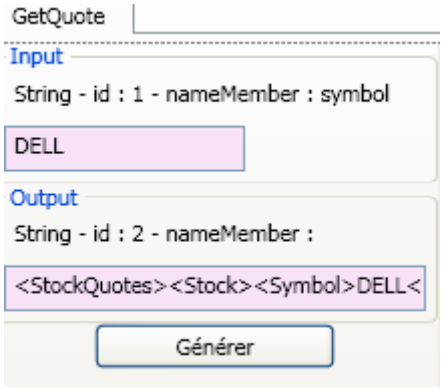
6.2.9. Stock Quote

Obtenir la cote d'une action.
Source : http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=19
WSDL : http://www.webservicex.net/stockquote.asmx?wsdl

Exemple :
Exemple avec DELL.
A noter que le résultat reçu est un string (les données sont du XML).

Rapport d'erreur :
Erreur svcutil binding http get et post : erreur non fatale, le binding soap fonctionne

Résultat génération automatique :
Génération de l'application : réussie
Utilisation de l'application : réussie



Contenu de l'output :
<StockQuotes><Stock><Symbol>DELL</Symbol><Last>23.06</Last><Date>5/30/2008</Date><Time>4:00pm</Time><Change>+1.25</Change><Open>23.50</Open><High>23.945</High><Low>23.05</Low><Volume>78497624</Volume><MktCap>47.095B</MktCap><PreviousClose>21.81</PreviousClose><PercentageChange>+5.73%</PercentageChange><AnnRange>18.13 - 30.77</AnnRange><Earns>1.312</Earns><P-E>16.62</P-E><Name>DELL INC</Name></Stock></StockQuotes>

Résultat génération wizard :
L'affichage est le même que pour la génération automatique

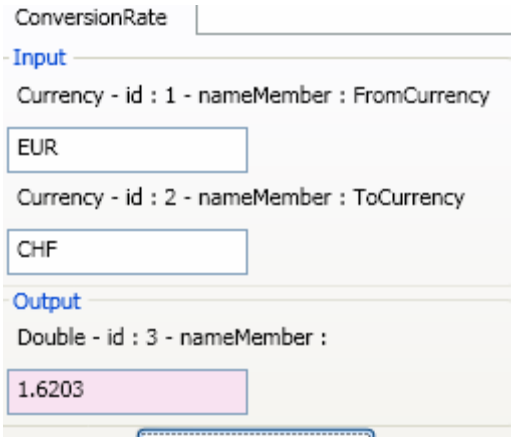
6.2.10. Currency Convertor

Taux de change.
Source : http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=18
WSDL : http://www.webservicex.net/CurrencyConvertor.asmx?wsdl

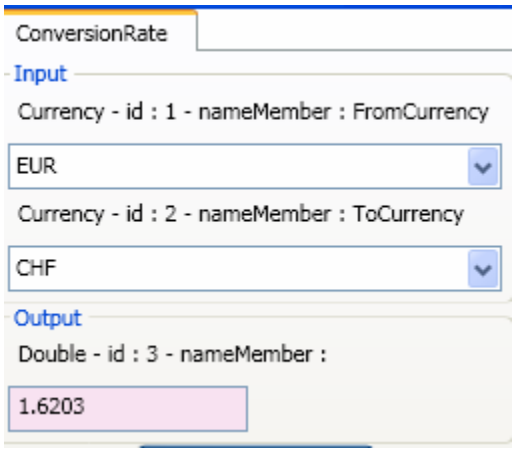
Exemple :
Quel est le taux de change entre des euros (EUR) et des francs suisse (CHF).

Rapport d'erreur :
Erreur svcutil binding http get et post : erreur non fatale, le binding soap fonctionne

Résultat génération automatique :
Génération de l'application : réussie
Utilisation de l'application : réussie



Résultat génération wizard :
Génération de l'application : réussie
Utilisation de l'application : réussie



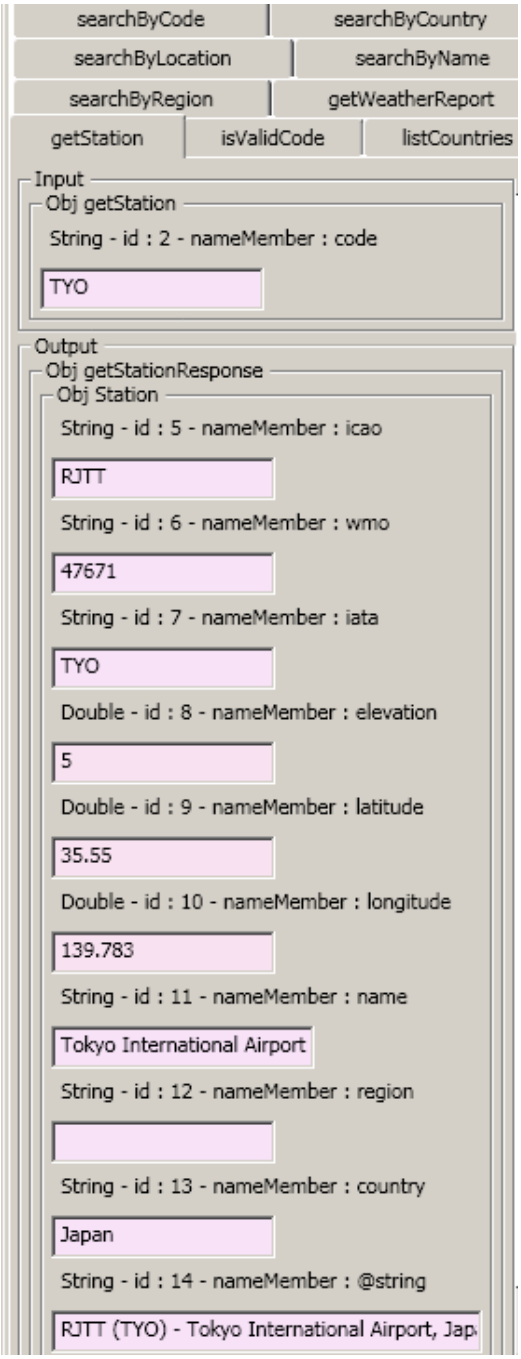
6.2.11. Global Weather

Obtenir un rapport de météo pour une station
Source : http://developer.capeclear.com/?q=webservices/globalweather/index.shtml
WSDL : http://live.capeclear.com/ccx/GlobalWeather?wsdl

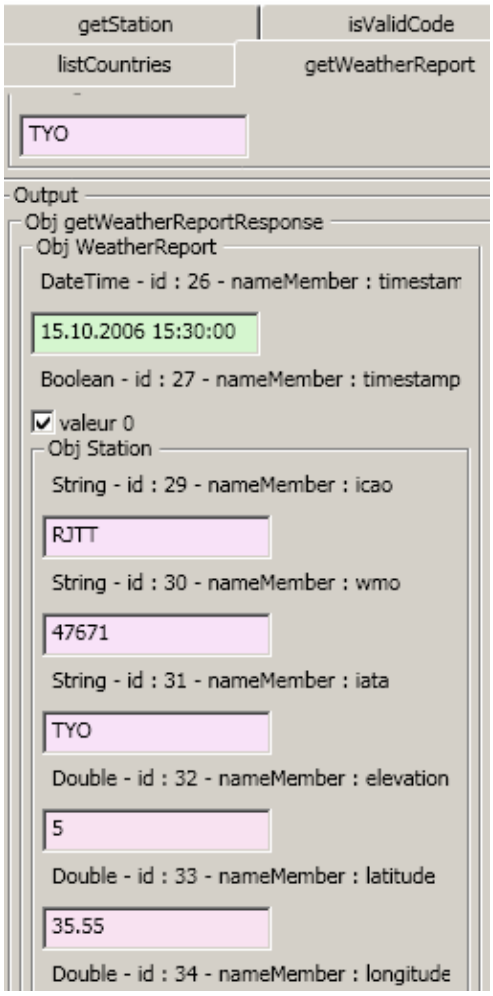
Ce web service est compose de différentes operations :
getStation, isValidCode, listCountries, searchByCode, searchByCountry, searchByLocation, searchByName, searchByRegion, getWeatherReport

Rapport d'erreur :
Erreur svcutil binding http get et post : erreur non fatale, le binding soap fonctionne

Résultat génération automatique :
Génération de l'application : réussie
Utilisation de l'application : réussie



Résultat génération wizard :
Génération de l'application : réussie
Utilisation de l'application : réussie



Remarques :
Les noms de certaines données membres contiennent des mots qui ont une signification pour le framework (par exemple string, return,…). Dans de tel cas, cela provoque une erreur de génération. Pour éviter ce problème, je rajoute un @ devant le nom du membre.
Ce web service contient 2 clients : GlobalWeather et StationInfo. Il m'a fallu modifier le code pour prendre ce genre de cas de figure en compte.
A noter également que pour l'opération listCountries, l'objet ListCountries passé en paramètre est vide (il n'y a que la déclaration de la classe). Il a fallu modifier le template de génération du XAML pour prendre ce cas de figure en compte et éviter que lors de la construction des éléments XAML, le template ajoute des balises de fermeture inutile parce qu'il n'y a rien dans cette classe qui font échouer la génération.

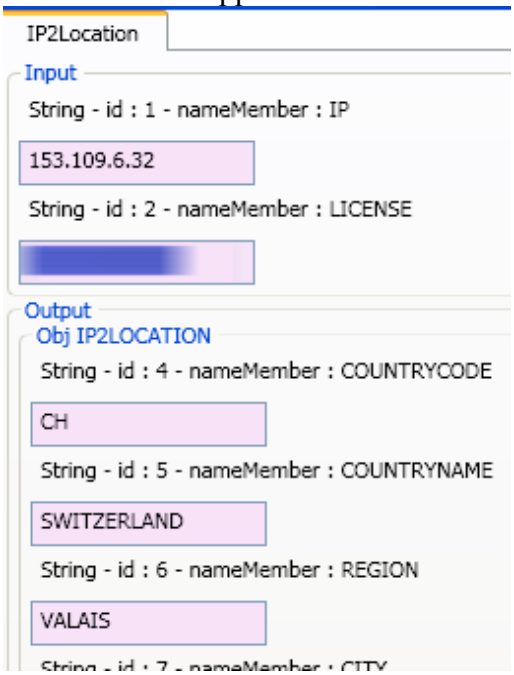
6.2.12. IP2Location

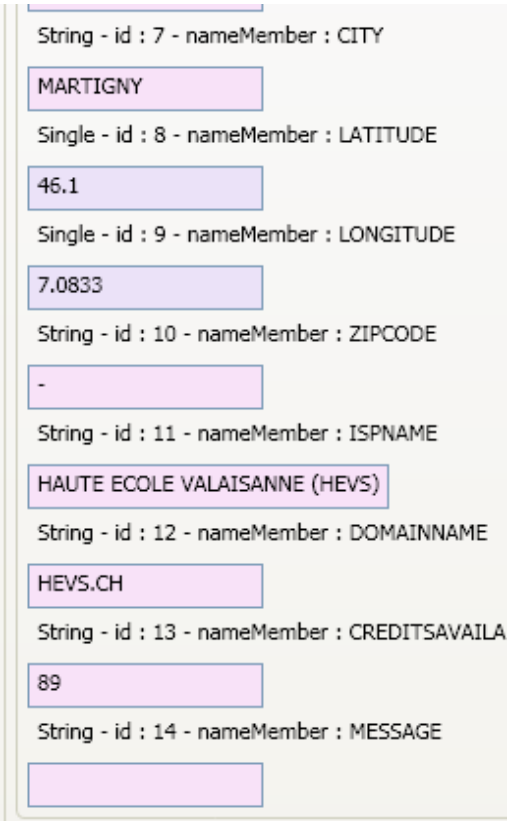
Pour une adresse IP donnée, obtenir les informations sur le pays, ville, nom de l'ISP,… Ce web service demande une licence pour être utilisé (possibilité de tester gratuitement).
Source : http://www.fraudlabs.com/ip2location.aspx
WSDL : http://ws.fraudlabs.com/ip2locationwebservice.asmx?wsdl

Exemple :
Obtenir les informations de l'adresse IP 153.109.6.32 (www.hevs.ch).

Rapport d'erreur :
Erreur svcutil binding http get et post : erreur non fatale, le binding soap fonctionne

Résultat génération automatique :
Génération de l'application : réussie
Utilisation de l'application : réussie





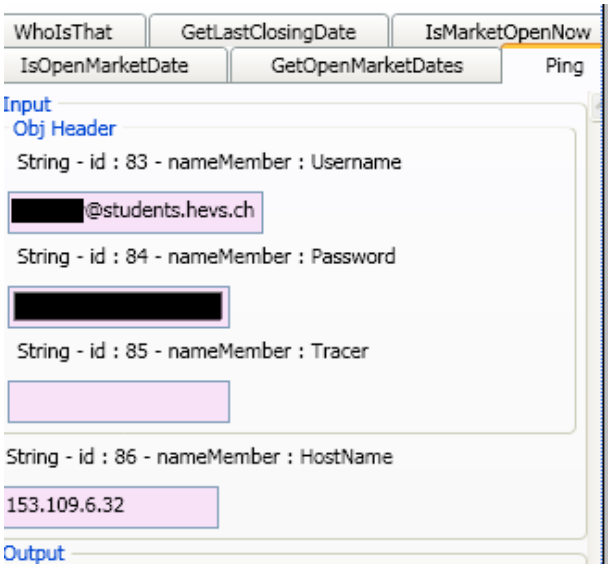
6.2.13. Web Service XigniteTools

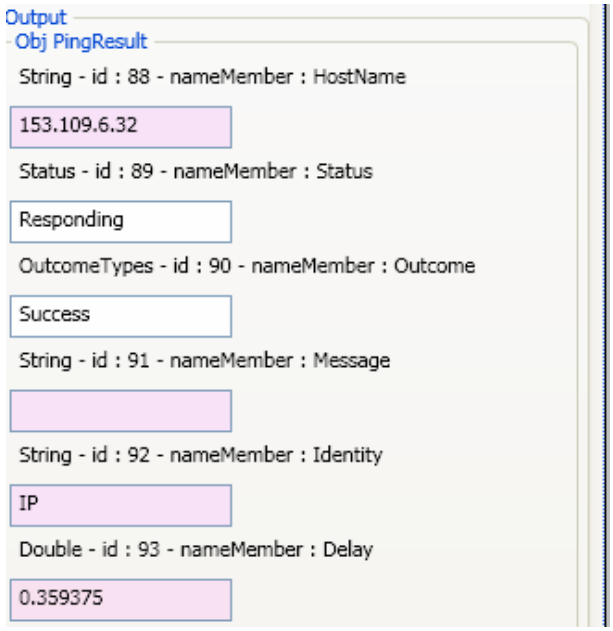
Quelques outils mis à disposition dans un web service.
Ce web service demande une licence pour être utilisé (possibilité de tester gratuitement).
Source : http://preview.xignite.com/xtools.asmx
WSDL : http://www.xignite.com/xtools.asmx?WSDL

Ce web service contient différentes opérations. Je n'ai testé que ceux qui me paraissaient le plus intéressant :
Ping pour tester le temps de réponse d'une adresse IP
WhoIsThat pour obtenir un whois d'un serveur
GetLastClosingDate pour obtenir le dernier jour avant la femeture

Rapport d'erreur :
Erreur svcutil binding http get et post : erreur non fatale, le binding soap fonctionne

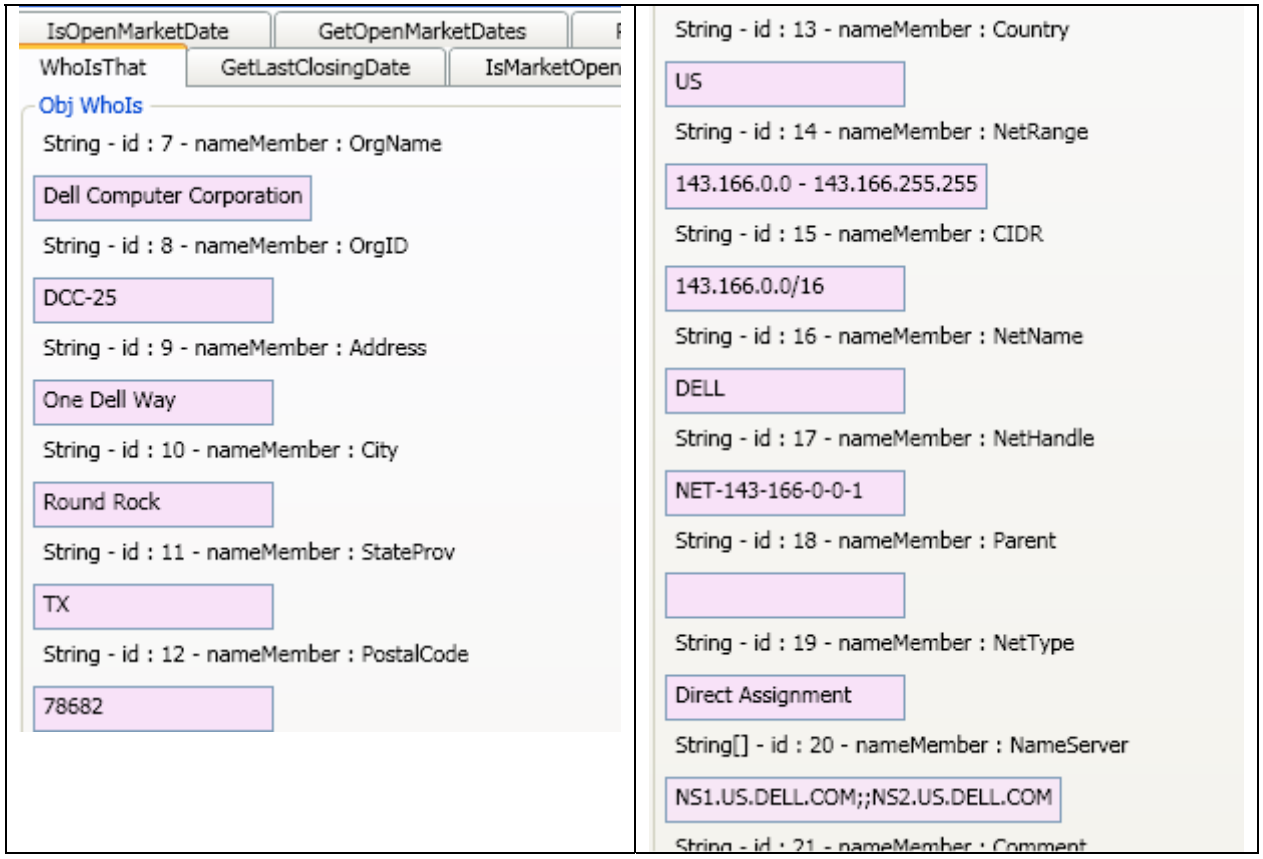
Résultat génération automatique :
Génération de l'application : réussie
Utilisation de l'application (opération Ping) : réussie



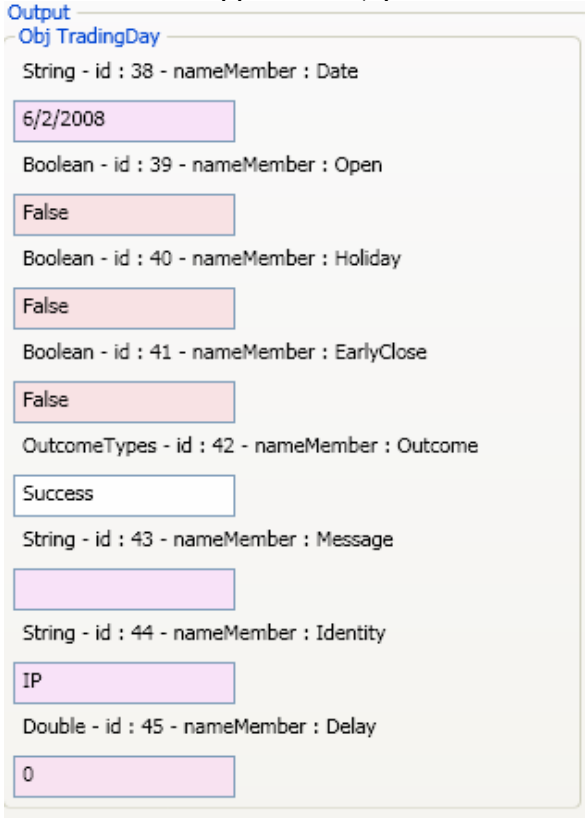


Utilisation de l'application (opération WhoIsThat) : réussie
Une erreur s'est produite lors de l'utilisation de cette opération pour l'adresse IP 153.109.6.32 (www.hevs.ch).
Le message d'erreur obtenu lors de l'utilisation est le suivant : La référence d'objet n'est pas définie à une instance d'un objet.
Cette erreur provient du fait que l'objet WhoIs retourné est nul. Après avoir recherché dans le code C#, il s'avère que le problème se situe lors du traitement du tableau de String[] (id20). Il n'y a pas de vérification qui est faite pour savoir si le membre de l'objet est nul (alors que pour tous les types simples, il y a cette vérification).

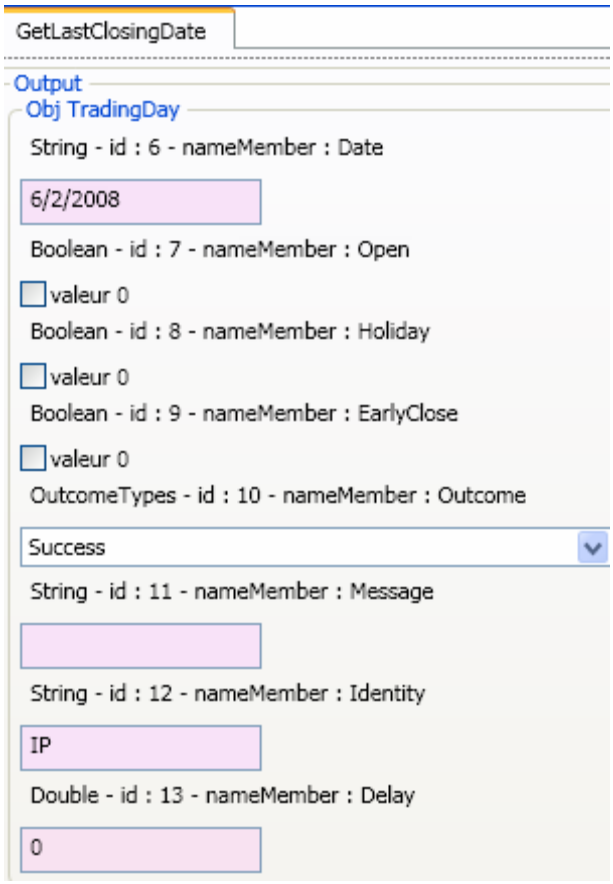
En utilisant l'IP 143.166.83.38 (www.dell.com), l'opération a réussie



Utilisation de l'application (opération GetLastClosingDate) : réussie



Résultat génération wizard :
Génération de l'application : réussie
Utilisation de l'application : réussie
Le test a porté uniquement sur l'opération GetLasClosingDate



6.2.14. Services de géolocalisation

Service payant, n'a pu être que généré.
Source : http://www.w-service.fr/servicesList.aspx
WSDL : http://www.w-service.fr/GeoGraphy/Localization.asmx?WSDL

Résultat génération automatique :
Génération de l'application : échec

Ce web service contient un tableau de type XmlAttribute (System.Xml.XmlAttribute). Ce type n'est pas implémenté dans l'application ce qui provoque une erreur de génération.

6.2.15. Amazon web services

Source : http://aws.amazon.com/
WSDL :
http://www.amazon.com/gp/redirect.html/ref=sc_fe_c_0_12738641_10?location=http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl%3f&token=1B53B9A1D3B078A3877A564512482AE82DCB7FC2

Je n'ai testé que 2 opérations du web service mais ça m'a permis de constater qu'il ne m'est pas possible de traiter ce web service correctement.

Pour l'opération ItemSearch. Il semblerait que le traitement soit beaucoup trop gros pour que mon application arrive à traiter les opérations (il est également possible qu'il y ait des bugs dans la génération du fichier XML : arriver à une taille dépassant les 10Go, c'est anormal).

En générant l'opération help, la création du fichier de données XML c'est bien passée. A noter qu'il y a tout de même près de 250 éléments (en comptant les objets et les types standards). Par contre, la génération de l'application a échouée.
Le problème survient à partir de la ligne 3633 du fichier C#. Je tente d'accéder à des objets qui n'existent pas (ces objets sont déclarés dans une boucle for).
En mettant en commentaire les lignes 3633 à 4074 puis en générant à la main l'exécutable, la génération a réussi (mais je ne sais pas si l'utilisation fonctionne, je n'ai pas compris comment fonctionne cette opération).
Je pense que le problème se situe dans le template de transformation :
AllWindow_xaml_cs_2.xslt -> template match : parametre/type, mode : output1
Ligne : 259 dans le when tbl custom //custom.

Le test qui est fait doit afficher la suite du contenu que s'il n'y a pas de noeuds parents contenant un tableau et je pense que c'est justement la recherche de tableau dans les parents qui est fausse.

Une suggestion de correction serait (qui n'a pas été faite dans l'application car il faudrait retester beaucoup de web services pour s'assurer que cette modification n'apporte pas d'autres bugs):

```
<xsl:if test="not(contains(../../@name,'[]')) and
not(ancestor::*[(contains(@name,'[]'))])">
```

6.2.16. Euro 2008 Football Championships

Traite de l'Euro 2008. Composé de 45 opérations (joueurs, stades, goal,…).
Source : http://www.xmethods.net
WSDL : http://euro2008.dataaccess.eu/footballpoolwebservice.wso?WSDL

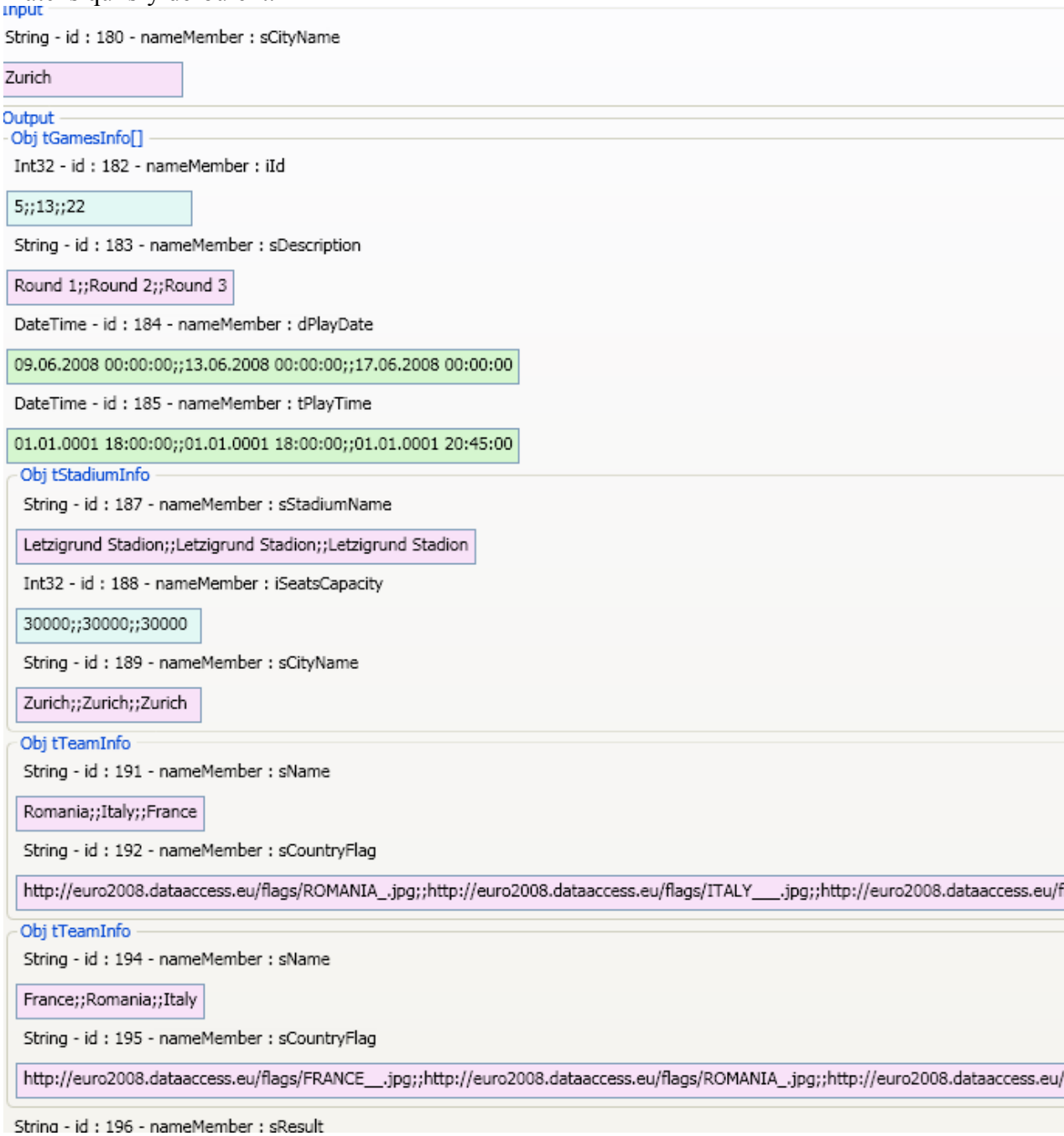
Rapport d'erreur :
Aucune erreur

Résultat génération automatique :
Génération de l'application : réussie sous conditions
Utilisation de l'application : réussie

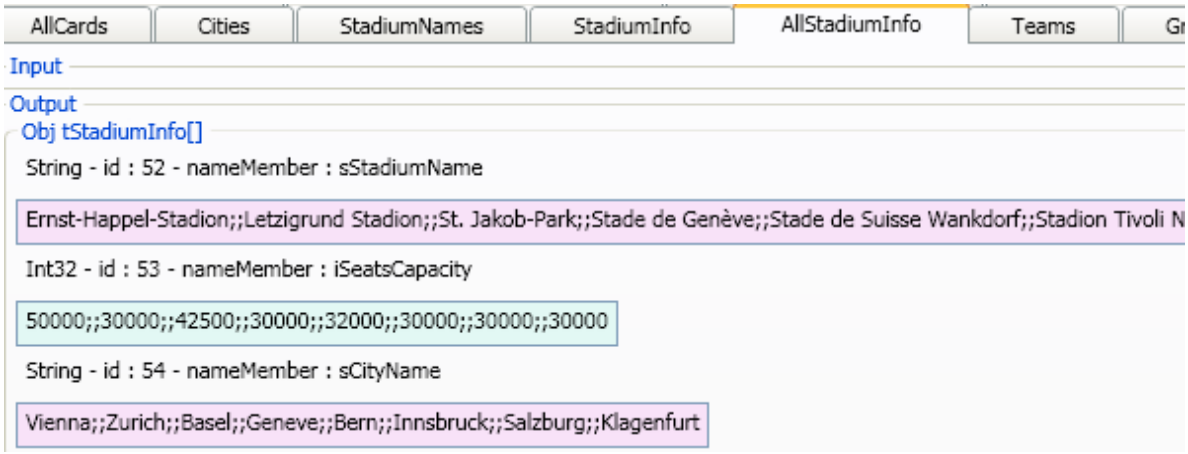
Il y a un problème pour récupérer par défaut les types avec Svcutil. Pour les types complexes, au lieu d'avoir un String[] comme type, je reçois un ArrayOfString et ce type inscrit de cette façon n'est pas traité correctement par l'application.
Après quelques tests, il s'avère que lors de l'utilisation du Svcutil, il faille mettre l'attribut de sérialisation XmlSerializer pour que le type soit correctement écrit et traité par l'application.
Ca semblait une bonne idée de mettre cette option par défaut mais au final, ça provoque un problème au niveau des web services WCF que j'ai développés : j'obtiens dans un type personnalisé pour chaque type un type doublon : Boolean et BooleanSpecified par exemple. Le résultat est qu'au final, ça ne fonctionne pas correctement.
Finalement, j'ai opté pour rajouter une option dans la configuration de l'application permettant à l'utilisateur de choisir la sérialisation qui doit être appliquée (Auto, XmlSerializer ou DataContractSerializer).
Par défaut, elle est sur Auto (svcutil prenant en premier l'option DataContractSerializer et si elle échoue, il utilisera XmlSerializer).

Pour ce web service, je n'ai testés que quelques opérations.
AllDefenders : pour un pays donné, obtenir la liste des défenseurs :

GamesPerCity : pour une ville donnée, obtenir toutes les informations sur les matchs qui s'y déroulent.



AllStadiumInfo : information sur tous les stades



Résultat génération wizard :
Génération de l'application : réussie
Utilisation de l'application : échec

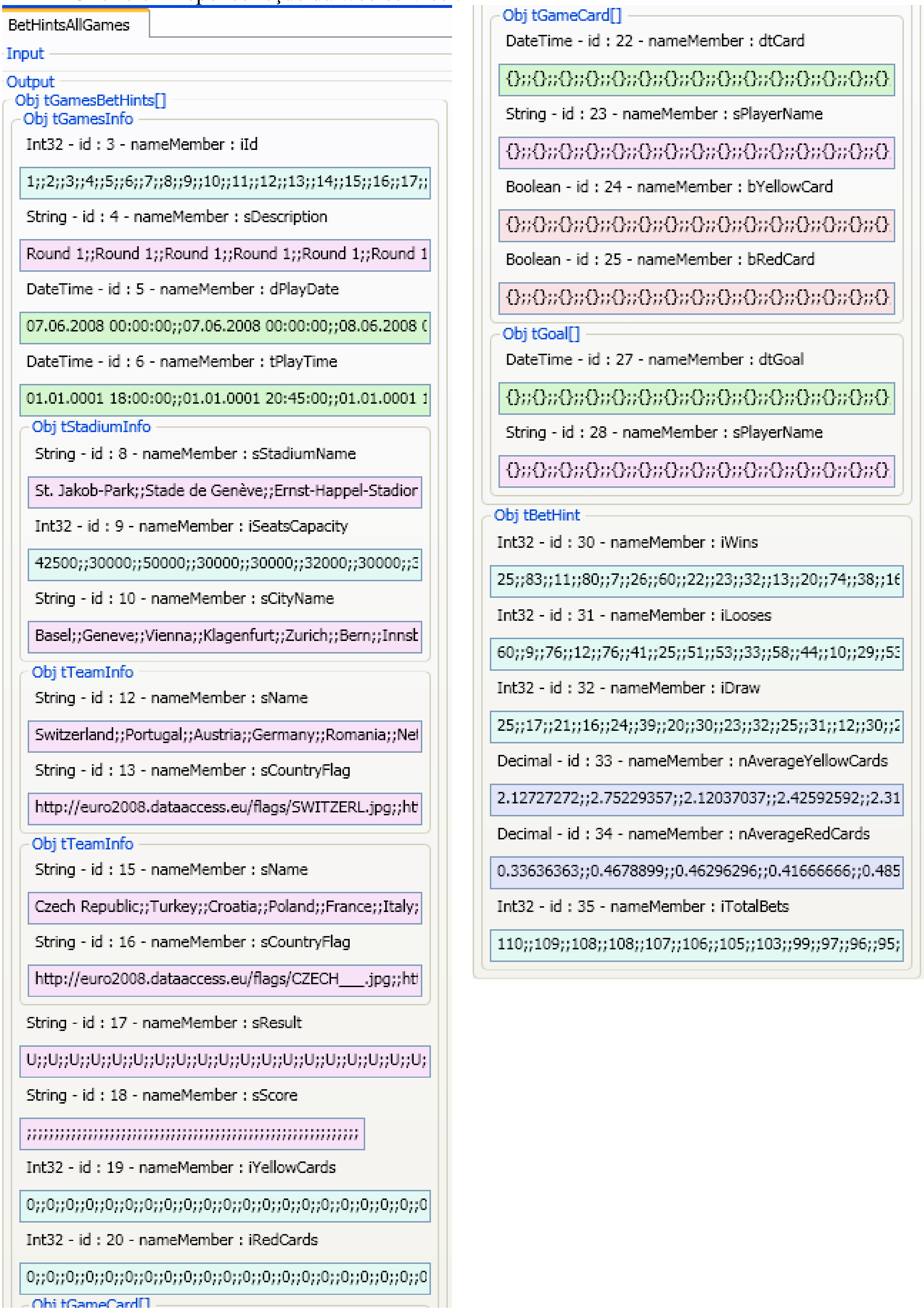
Pour la génération du wizard, je n'ai testé que l'opération BetHintsAllGames (c'est l'une des opérations disponibles qui présente la plus grande complexité au niveau de la structure et des types utilisés).
L'échec de l'utilisation provient de l'utilisation d'UserControls spéciaux pour l'affichage des types Boolean (sous forme de CheckBoxs). Le résultat reçu du web service pour ces types ne contenait rien comme donnée et le code de l'application n'arrive pas à convertir rien en Boolean.
Un autre problème (non fatal) a été identifié dans l'écran affichant toutes les opérations disponibles. Il n'y a pas de scroll disponible pour le cas où le nombre d'opération dépasserait la taille d'affichage.
Pour éviter le problème ci-dessus, j'ai généré une nouvelle version en utilisant que des userControls standards.
Cette opération retourne un tableau contenant des informations sur les matchs ainsi que ce que les gens ont pronostiqués (une entrée du tableau égal un match).
On distingue 2 objets principaux : tGamesInfo qui contient les informations réelles sur le match et tBetHint pour les pronostiques.

Objet tGamesInfo :
information sur l'id de la rencontre, à quel tour elle appartient, la date et heure du match. Cet objet contient d'autres objets :
Objet tStadiumInfo : information sur le lieu de la rencontre, la capacité du stade et la ville.
Objet tTeamInfo : information sur le nom de l'équipe et une image du drapeau. Cet objet est présent deux fois (un par équipe)
Objet tGameCard : C'est un tableau d'objet qui permet de lister pour la rencontre les joueurs qui ont eu un carton et si le carton était jaune ou rouge
Objet tGoal : C'est un tableau d'objet qui permet de lister pour la rencontre à quel temps à été marqué les buts et par quels joueurs.
Au niveau des membres de l'objet, on y trouve le nombre de cartons jaunes et rouges, le score et le résultat

Objet tBetHint :
Information sur les pronostiques. Contient le nombre de personnes qui ont émis un pronostic sur la victoire, défaite ou match nulle, la moyenne de carte jaune et rouge

qu'aura ce match et le total des gens qui ont pronostiqué.Au lancement de l'application, l'affichage est le suivant :

Une fois la réponse reçue du web service :



Exemple avec le premier match (web service testé le 3 juin) :
Suisse – République Tchèque le 7 juin à 18:00.
Le match se déroule au stade Jackob-Park à Bâle (contenance 425000 places).
Les informations sur le résultat, le score, les cartons jaunes et rouges ne contiennent rien. Au niveau des pronostics, sur 108 personnes, 24 personnes pensent que la Suisse va gagner, 59 qu'elle va perdre et 25 qu'il y aura des prolongations. La moyenne estimée des cartons jaunes est de 2.08 et de 0.36 pour les cartons rouges.

6.2.17. Weather data

Récupération des données météos. Nécessite d'être enregistré pour pouvoir utiliser ce service. Pour chaque opération, il faut toujours renseigner les champs UserName et Password.
Source : http://www.ejse.com/weather_data.htm
WSDL : http://www.ejse.com/WeatherService/Service.asmx?WSDL

Rapport d'erreur :
Erreur svcutil binding http get et post : erreur non fatale, le binding soap fonctionne

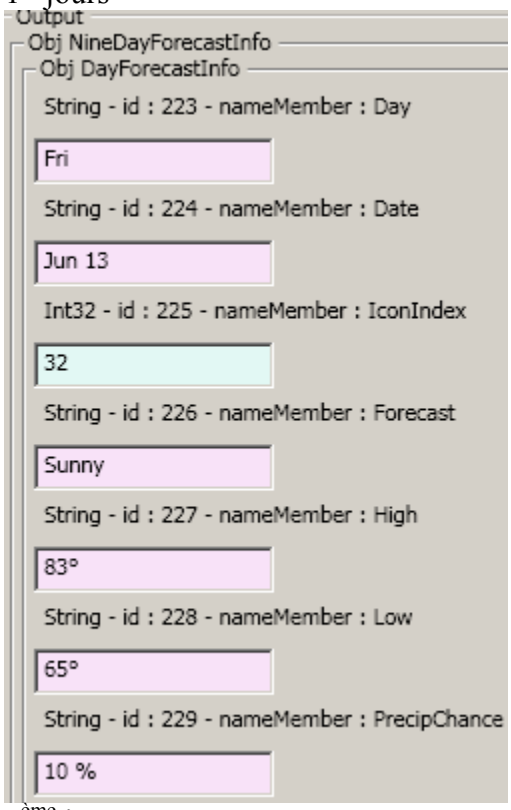
Résultat génération automatique :
Génération de l'application : réussie
Utilisation de l'application : réussie

J'ai rencontré un problème lors de l'utilisation du web service au niveau du temps imparti pour recevoir une réponse de la part du web service. Le délai d'expiration est de 1 minute par défaut ce qui largement suffisant pour des web services en local et qui jusqu'à ce test l'était également pour les web services distants.

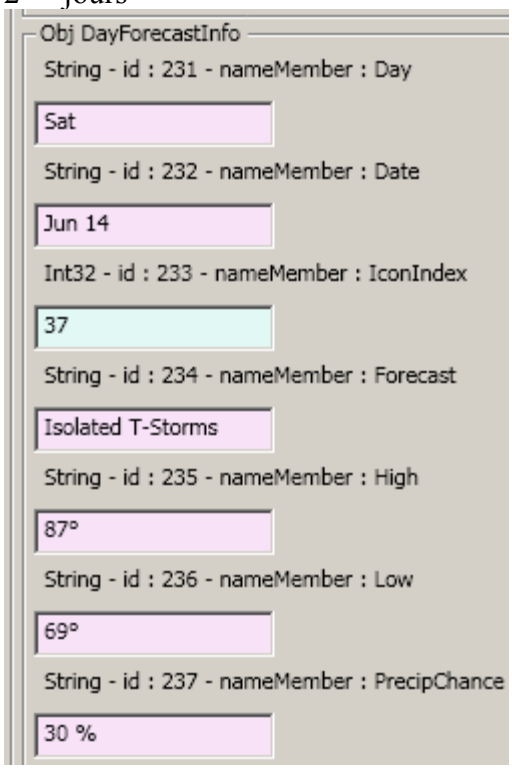
Pour résoudre ce problème, il faudrait que l'utilisateur puisse paramétrer dans le fichier de config les timeouts du binding et ainsi changer facilement les valeurs par défaut.
Lors du test, le web service a également été indisponible (peut-être que ces deux problèmes sont liés).
Finalement, j'ai tout de même réussi à utiliser le web service (pour les opérations qui ne concernent pas l'Irak) sans avoir modifié quoi que ce soit.

GetNineDayForecastInfo
Pour un numéro postal donné, donne la prévision à 9 jours. Chaque prévision de jour contient : le nom du jour, la date, l'IconIndex (qui je pense doit être une information concernant une image), la prévision (soleil, pluie,…), les températures maximale et minimal, le pourcentage de chances de précipitation. Exemple avec New York (je n'ai mis que les 4 premiers jours) :

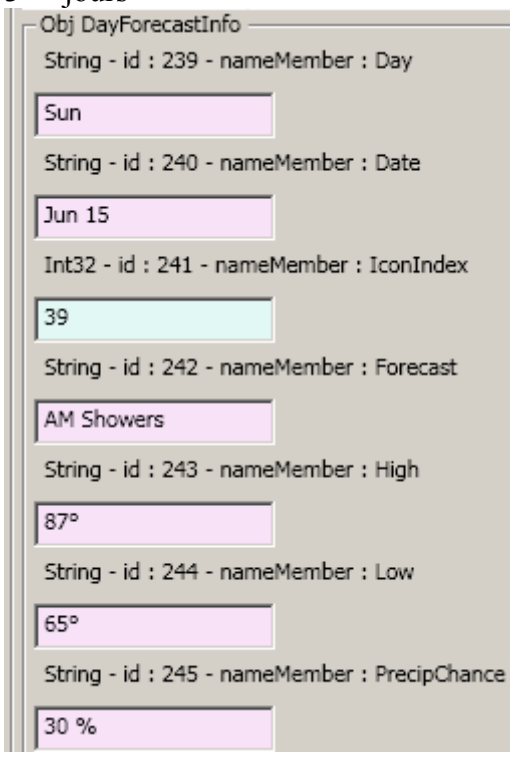
1er jours



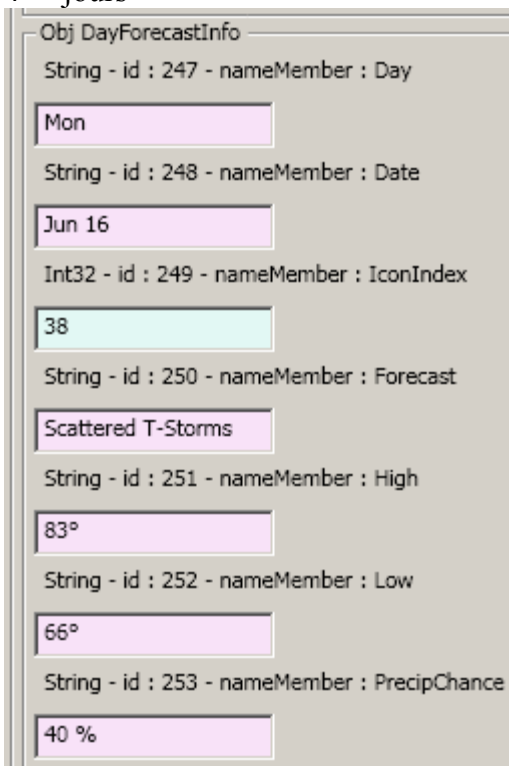
2ème jours



3ème jours



4ème jours



7. Manuel d'utilisation de l'application

Pour expliquer comment utiliser l'application, je vais utiliser 2 web services dont les wsdl sont atteignables à l'adresse http://localhost:1999/Service1.svc?wsdl :

WCFCalculator :

Ce web service contient 4 opérations :

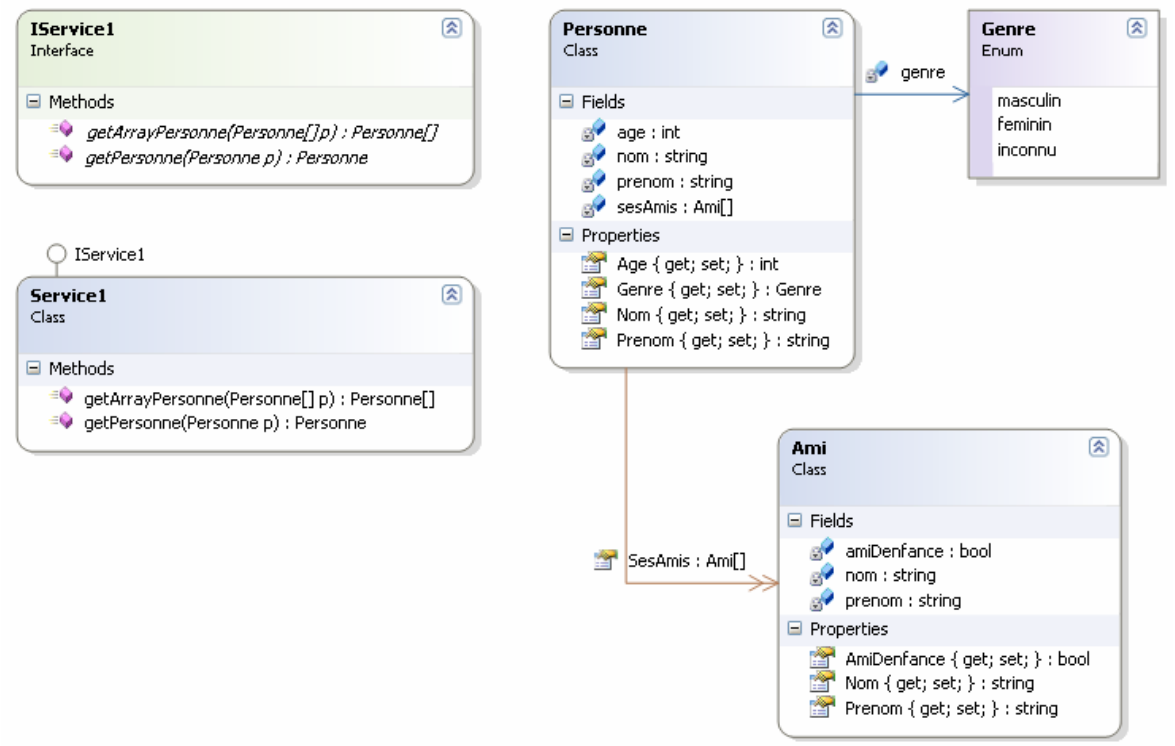
- double add(double a1, double a2);
- double div(double a1, double a2);
- double mult(double a1, double a2);
- double soustr(double a1, double a2);

WCFPersonne :

Ce web service contient 2 opérations :

- Personne getPersonne(Personne p);
- Personne[] getArrayPersonne(Personne[] p);

Ci-dessous le diagramme de classe :



Un objet Personne est composé de l'âge, nom, prénom et de son genre (un type Enum). Il a également un tableau d'objet de type Ami. Ce tableau pourra avoir une taille différente suivant les objets Personne (une personne pouvant avoir 1 ami alors qu'une autre personne aurait 3 Amis).

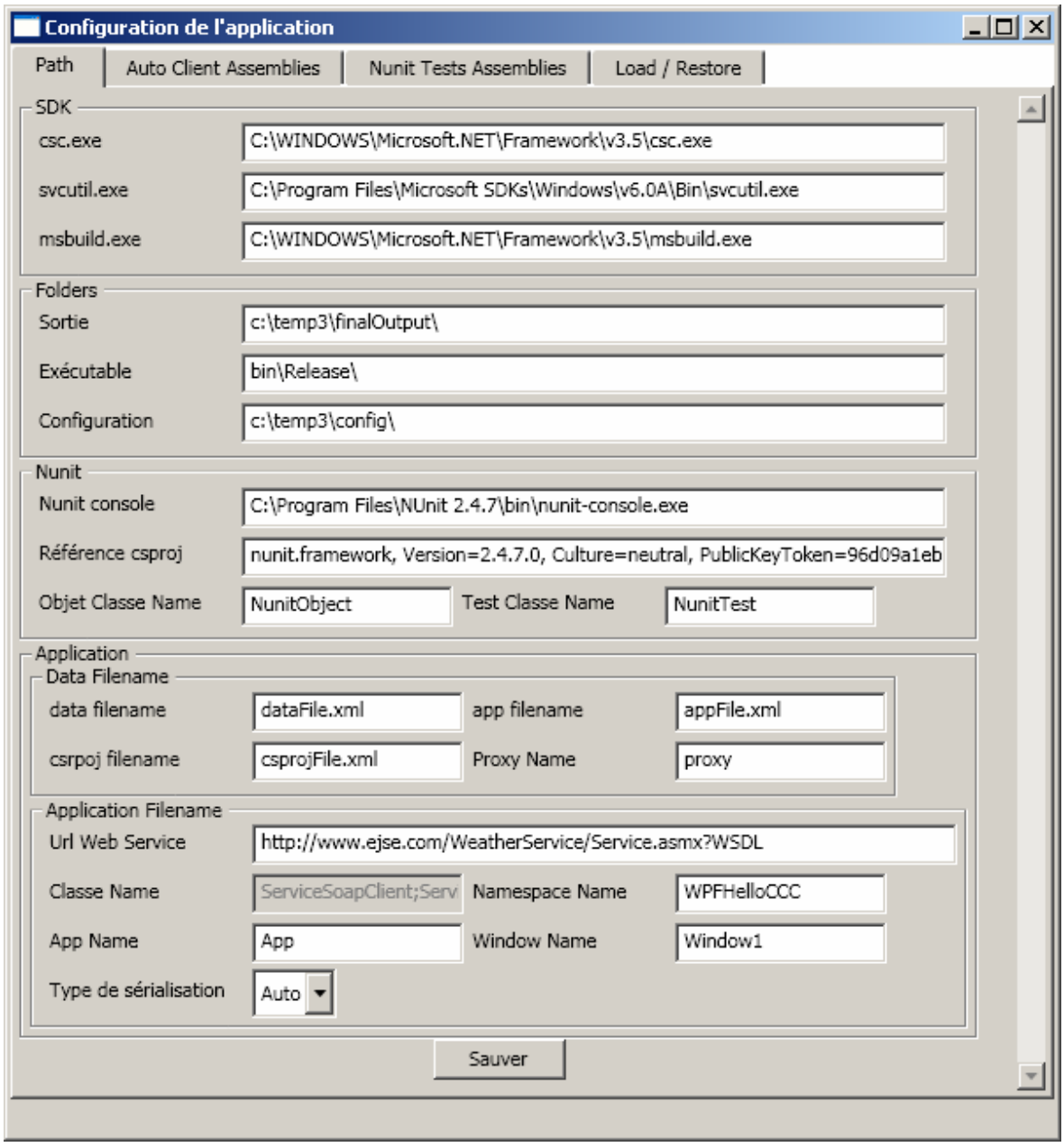
Un objet Ami est composé de son nom et prénom ainsi qu'un Boolean indiquant si cet ami est un ami d'enfance.

7.1. Configuration de l'application

Avant de lancer une génération, il faut commencer par paramétrer le générateur en se rendant dans la partie option. Elle est composée de 4 parties.

7.1.1. Path

Cet onglet permet de définir les chemins d'accès aux différents outils utilisés, les noms et chemins des dossiers, les noms des fichiers utilisés, la façon de sérialiser les objets sans oublier l'url du WSDL.
Pour sauver les modifications, cliquez sur Sauver.



7.1.1.1. SDK

Cette partie s'occupe de définir le chemin des programmes. Lors de la première utilisation de l'application, il faut contrôler que les chemins correspondent à ce qui se trouve sur la machine.
Csc.exe : est utilisé pour générer les dll du fichier proxy
Svcutil.exe : est utilisé pour créer un fichier proxy et configuration
Msbuild.exe : est utilisé pour construire l'application générée.

7.1.1.2. Folders

Cette partie s'occupe des dossiers dans lesquels sont stockés la sortie, l'application générée et l'emplacement des fichiers de configuration.
Sortie : dossier qui contiendra toutes les générations effectuées
Exécutable : dossier où doit être généré l'exécutable (chemin relatif)
Configuration : dossier contenant les différents fichiers de configuration XML permettant de recharger un projet de génération.

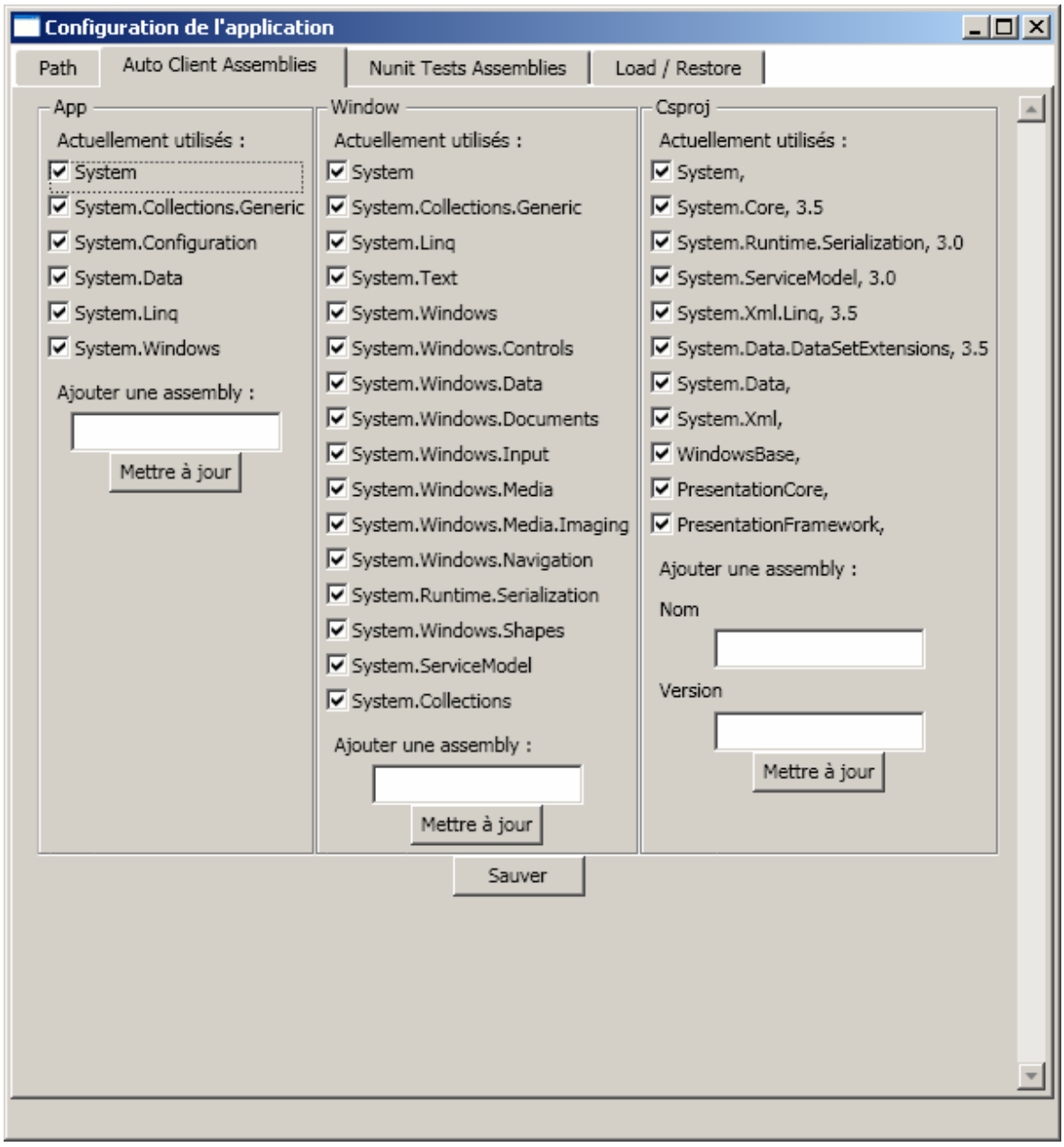
7.1.1.3. Nunit

Cette partie est destinée à la gestion du programme NUnit.
Nunit Console : chemin d'accès à la version console de NUnit
Référence csproj : permet de spécifier la référence de Nunit pour que les éléments de code spécifique soient reconnus lors du build de l'application.
Objet Classe Name : nom du fichier C# qui contient les différentes méthodes pour appeler les opérations du web service.
Test Classe Name : nom du fichier C# qui contient les tests unitaires à effectuer.

7.1.1.4. Application

Cette partie s'occupe de définir les noms des fichiers, namespace, l'adresse du WSDL du web service et la façon de sérialiser
Csv filename : nom du fichier de données xml contenant pour chaque operation tous les types.
App filename : nom du fichier de données xml contenant les informations pour la generation du fichier app.
Csproj filename : nom du fichier de données xml contenant les informations pour la génération du fichier csproj (fichier de configuration pour lancer une génération avec MSBuild).
Proxy Name : nom du fichier proxy
Url Web Service : emplacement du WSDL du web service que l'on souhaite utiliser.
Namespace Name : nom de l'espace de nom de l'application générée.
App Name : nom du fichier app.xaml/cs
Window Name : nom du fichier window.xaml/cs
Type de serialisation : choix de la façon dont les objets doivent être sérialisés : AUTO pour laisser l'application se débrouiller, DataContract (WCF), XMLserializer pour les cas où les données reçues sont mal traitées (par exemple, ArrayOfString au lieu de String[]).

7.1.2. Auto Client Assemblies

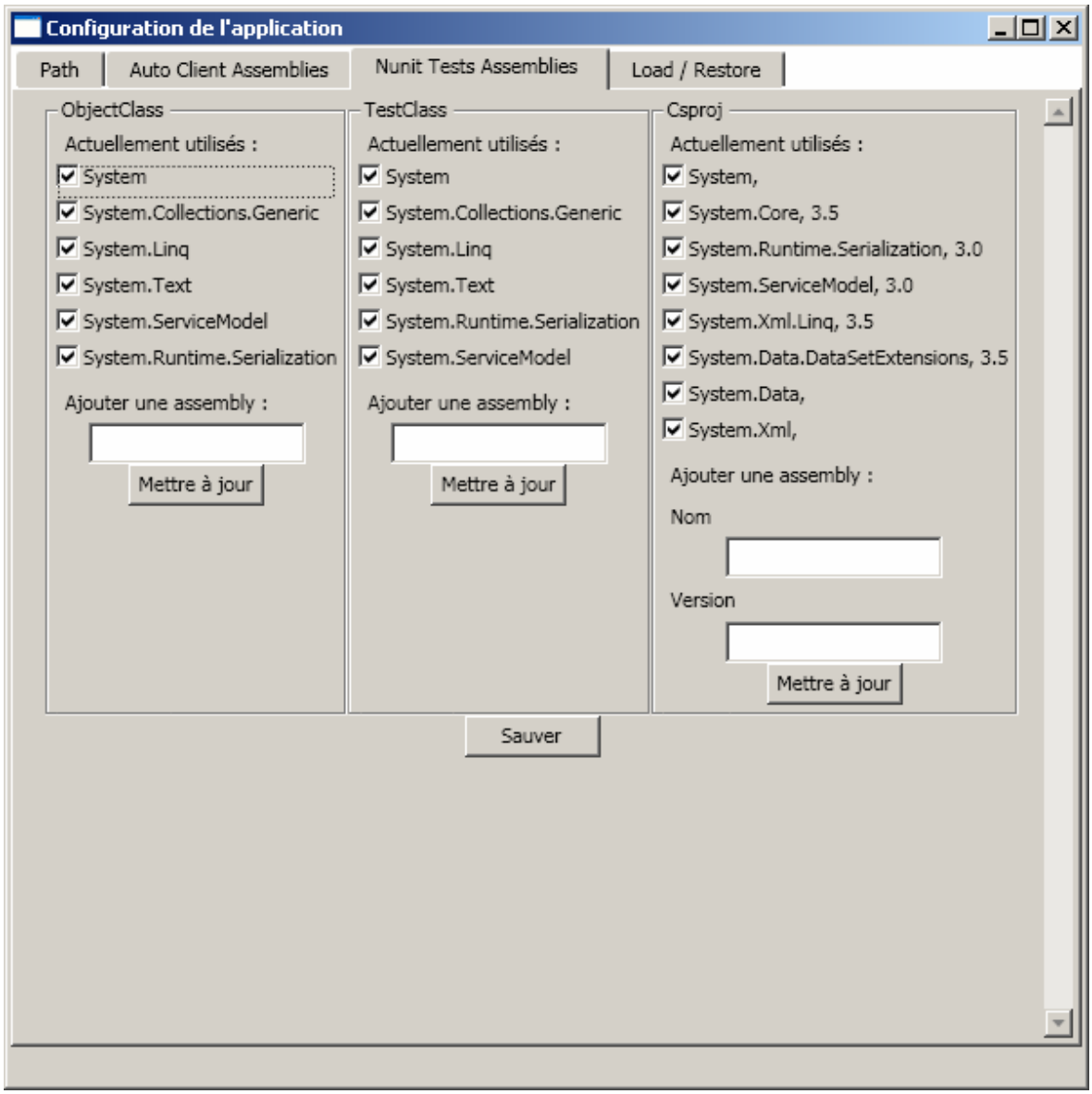


Cet onglet permet de gérer les assemblies qui doivent être utilisées dans le code behind des fichiers App et Window ainsi que dans le fichier csproj (il faut référencer les assemblies).

Pour enlever un assembly, il suffit de le décocher et dans la même colonne, de cliquer sur le bouton Mettre à jour (l'assembly disparaît à ce moment) et de cliquer sur Sauver pour que le changement soit réellement pris en compte.

Pour ajouter un assembly, entrez le nom de l'assembly, cliquez sur Mettre à jour puis sur Sauver.

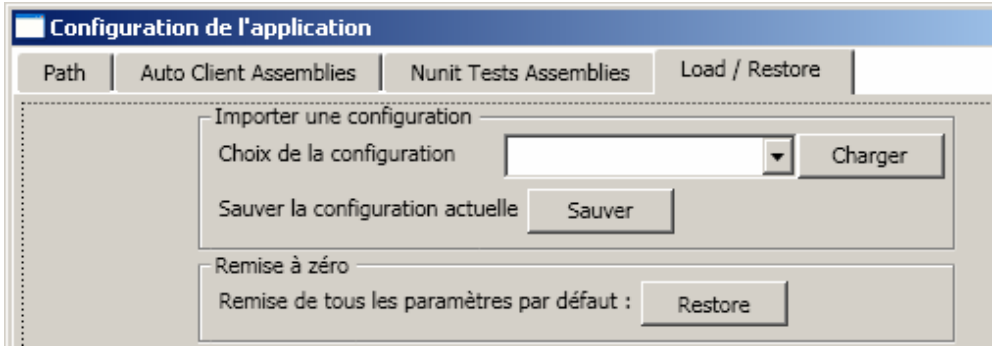
7.1.3. Nunit tests Assemblies



Cet onglet permet de gérer les assemblies qui doivent être utilisés lors de l'utilisation de la génération des tests unitaires.
Le principe de fonctionnement est identique au point 7.1.2 Auto Client Assemblies.

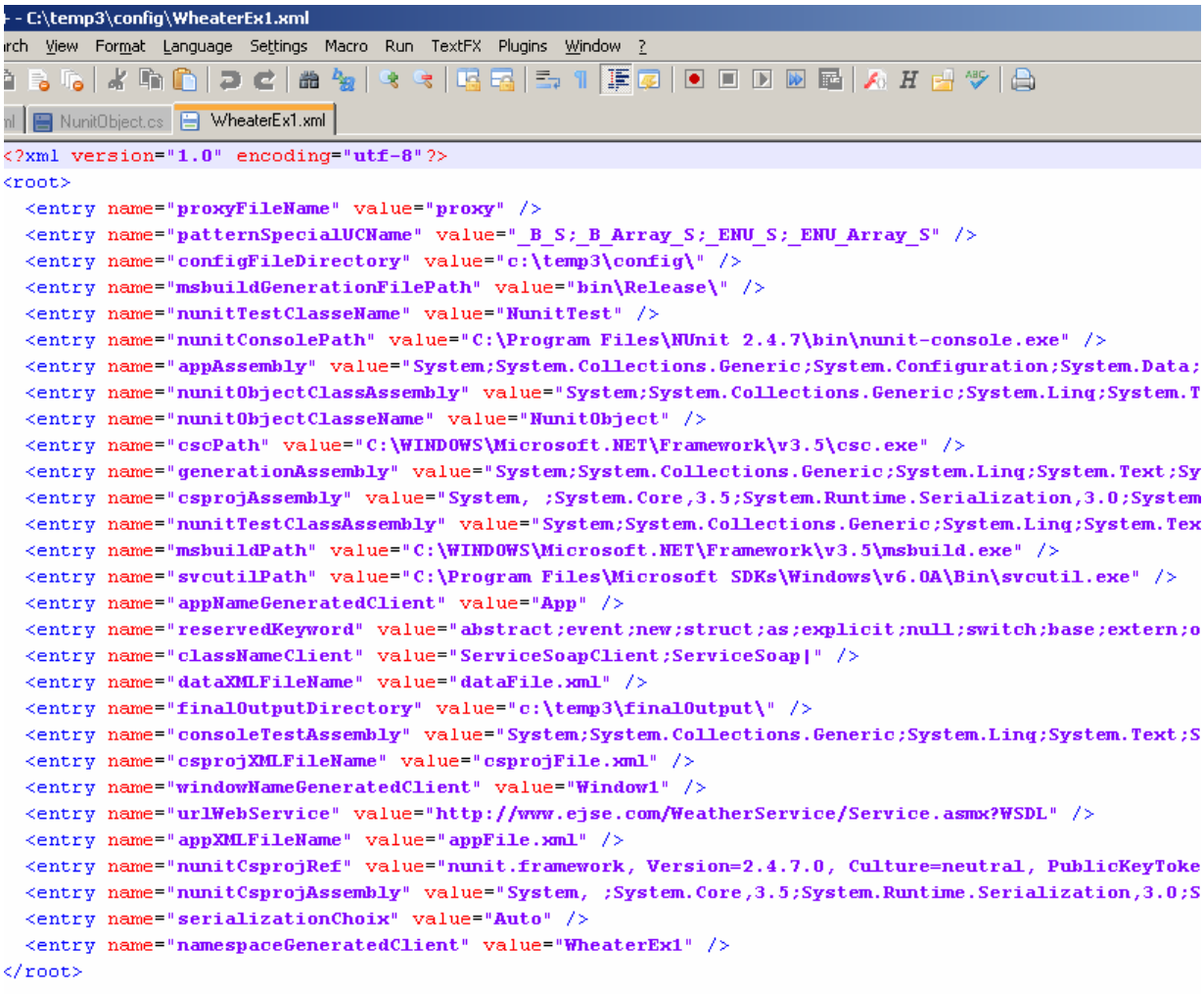
7.1.4. Load / Restore

Cet onglet permet de gérer les options sous forme de projet. Il est possible de sauver la configuration actuelle et d'en charger une autre, voir de remettre tout à zéro.



Sauver la configuration actuelle : Il suffit de cliquer sur le bouton Sauver pour qu'un fichier xml contenant tout le contenu du fichier ApplicationSettings.settings soit créé (ce fichier ApplicationSettings comprend entre autres toutes les options de configuration de cette fenêtre).

Voici le résultat créé :



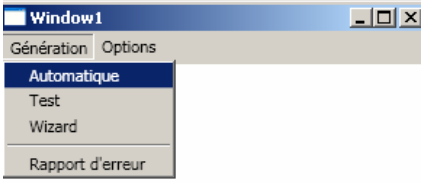
Attention : le namespace (défini dans l'onglet Path) de l'application est utilisé comme nom pour le fichier xml.

Restaurer une configuration : Pour restaurer une configuration, il suffit de choisir dans la liste déroulante le fichier à importer (ces fichiers xml sont normalement stockés dans c:\temp3\config)

Remise à zéro : permet de remettre les paramètres originaux de l'application.

7.2.Génération automatique

Pour n'importe quel web service dont il faut générer le client de façon automatique, il faut prendre le choix Automatique.

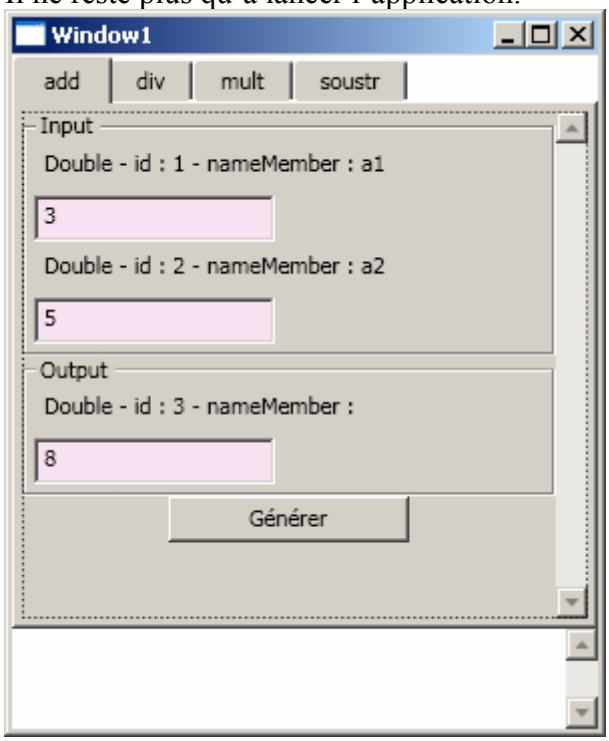


7.2.1. WCFCalculator

Lancer la génération en choisissant la version automatique. Au fur et à mesure de l'avancement du traitement, les informations sur ce qui se passe sont affichés dans la barre de status.
Si la génération c'est bien déroulée, l'application doit afficher un bouton pour lancer l'application générée.

Si la génération a échouée, le rapport d'erreur contiendra les informations sur les problèmes rencontrés.

Il ne reste plus qu'à lancer l'application.

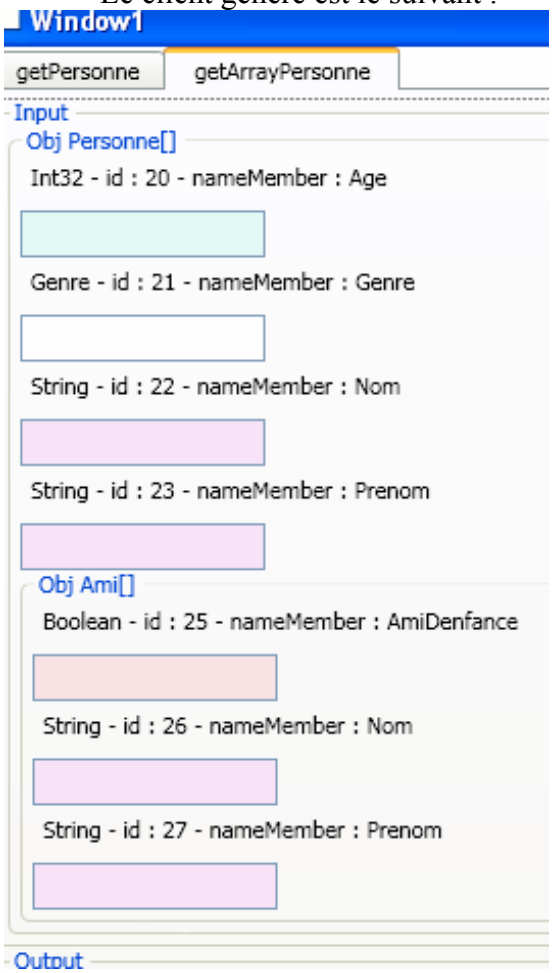


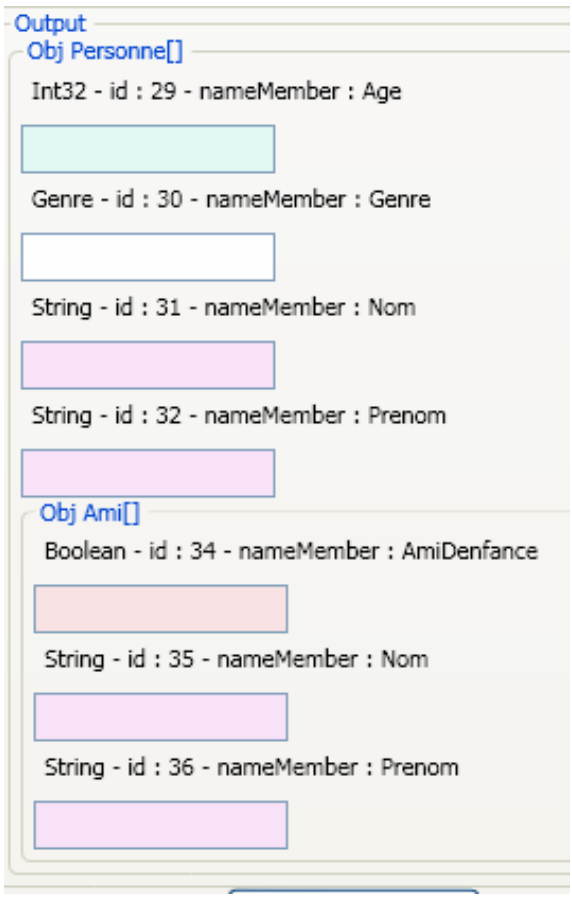
Il suffit de choisir l'opération que l'on souhaite essayer (chaque opération correspondant à un onglet), de rentrer les données pour tout le contenu du GroupBox Input et d'appuyer sur le bouton Générer pour obtenir soit le résultat, soit un message d'erreur dans la partie du bas en blanc de l'application.

7.2.2. WCFPersonne

Le déroulement pour générer le client de manière automatique est identique à WCFCalculator.

Le client généré est le suivant :





La première chose qu'il faut remarquer : la partie input est identique à la partie output (ce qui est normal). A l'intérieur, un GroupBox dont le header est Obj Personne[]. Cela indique que l'on est en présence d'un tableau de type Personne. Tout ce qui est contenu dans le GroupBox (à l'intérieur de la ligne grise) fait parti de l'objet Personne.
A l'intérieur, on retrouve bien l'âge de type int, le nom et prénom qui sont des strings. Pour le genre, c'est le type Genre qui est un énum. Comment savoir si c'est bien un énum et pas un simple objet ? Simplement parce que si c'est un objet, il ne peut pas avoir d'UserControl, il serait un GroupBox.
C'est le cas du dernier membre (un objet) : le tableau d'Ami.
A l'intérieur se trouve le nom et prénom de type string et la propriété AmiDenfance de type Boolean.

Comment utiliser l'application générée ?
Lorsqu'il y a des tableaux, il faut utiliser des délimiteurs (sans espaces). Cela dépend s'il y a uniquement un tableau ou s'il y a un tableau dans un tableau.
Par exemple :
1 tableau : ;;
a;;c;;a (taille 3)
2 tableaux : {};;{}
{a;;b};;{c;;d;;e;;f}
Le tableau parent (TblParent) est de taille 2 : {a;;b} et {c;;d;;e;;f}.
Pour chaque entrée du tableau parent, il y a un autre tableau
TblParent[0] : a;;b

TblParent[1] : c;;d;;e;;f

3 tableaux : {{};;{}};;{{};;{}}
{{a};;{b;;c}};;{{d;;e};;{f;;g;;h};;{i;;j}}
Le tableau parent (TblParent) est de taille 2 : {{a};;{b;;c}} et
{{d;;e};;{f;;g;;h};;{i;;j}}

Pour chaque entrée du tableau parent, il y a un autre tableau
TblParent[0] : {a};;{b;;c}
TblParent[1] : {d;;e};;{f;;g;;h};;{i;;j}

TblParent[0] contient 2 tableaux enfant (TblEnfant_0) : {a} et {b;;c}
TblEnfant_0[0] : a (un tableau avec une seule entrée)
TblEnfant_0[1] : b;;c qui est un tableau avec 2 entrées.

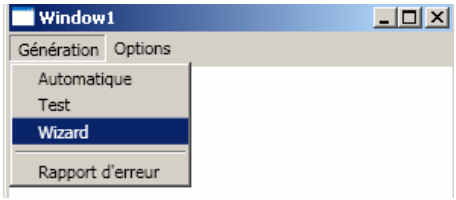
TblParent[1] contient 3 tableaux enfant (TblEnfant_1) : {d;;e} et {f;;g;;h} et {i;;j}
TblEnfant_1[0] : d;;e un tableau avec 2 entrées
TblEnfant_1[1] : f;;g;;h un tableau avec 3 entrées
TblEnfant_1[2] : i;;j un tableau avec 2 entrées

Exemple concret de l'utilisation du client :
Pour illustrer, je vais prendre 2 grands personnages de Disney pour le tableau de Personnes : Mickey Mouse et Donald Duck.
Mickey Mouse a les amis suivants : Minnie Mouse (amie d'enfance), Goofy Dingo et Pluto le chien (qui est aussi un ami d'enfance).
Donald Duck a lui les amis suivants : Riri Duck, Fifi Duck, Loulou Duck, Géo Trouvetou et Balthazar Picsou (aucun ne sont des amis d'enfance).

Il y a 2 Personnes à rentrer, donc, les valeurs sont séparées par un double ; pour les ids 20 à 23.
Pour le tableau, c'est un tableau d'Ami par Personne. Un tableau dans un tableau, il faut délimiter avec {};;{}.
Pour Mickey, on a bien les amis {Minnie;;Dingo;;Pluto} et Donald à les amis {Riri;;Fifi;;Loulou;;Géo;;Balthazar}.

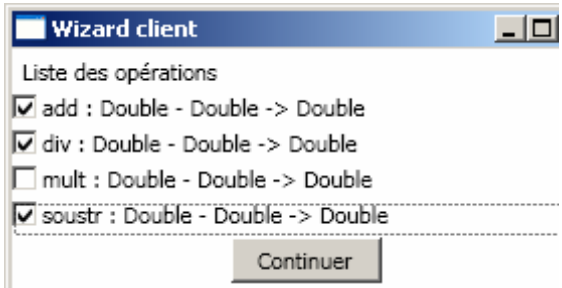
7.3.Génération wizard

Pour n'importe quel web service dont il faut générer le client et que l'on souhaite ne prendre qu'une partie des opérations disponibles ou que l'on souhaite utiliser des contrôles spéciaux pour les types Boolean ou des énumérations, il faut prendre le choix Wizard.

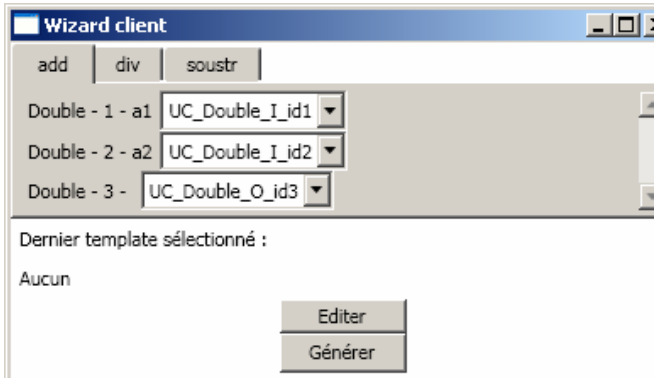


7.3.1. WCFCalculator

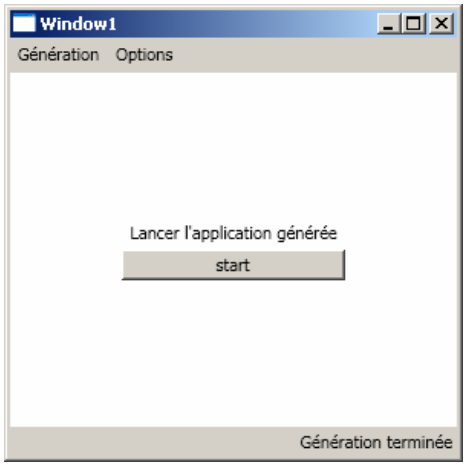
Le premier choix qui s'offre à l'utilisateur est celui des opérations qu'il souhaite utiliser en les cochant. Une fois le choix terminé, il suffit d'appuyer sur Continuer.



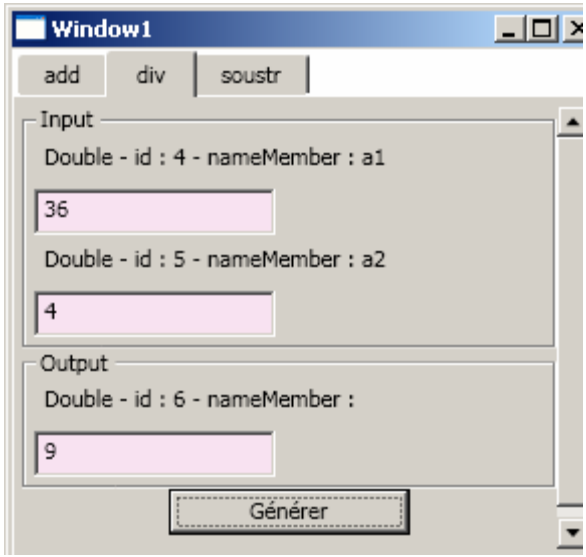
Le deuxième choix concerne les UserControls que l'on souhaite utiliser.
Seul les types Boolean, Boolean[], les énumérations et les tableaux d'énumérations peuvent avoir un UserControl spécial en plus. Dans le cas de WCFCalculator, il n'y a que des Double, donc, pour cette partie, il n'y a rien de particulier à faire et l'on peut sans autre cliquer sur Générer pour générer l'application cliente.



Si la génération s'est bien déroulée, dans la fenêtre principale apparaît le bouton permettant de lancer le client généré.

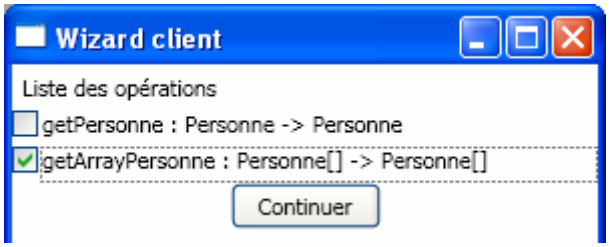


En lançant l'application, on retrouve bien que les 3 opérations que l'on a sélectionnées.

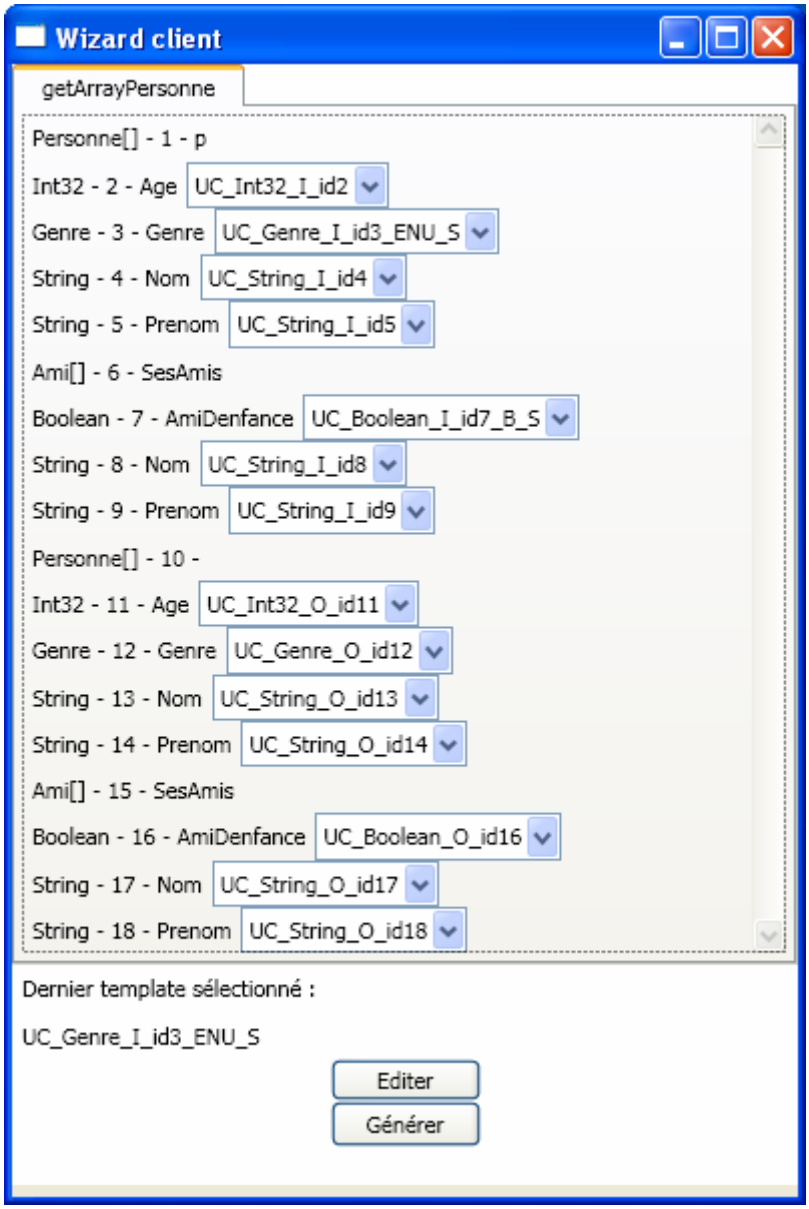


7.3.2. WCFPersonne

Le premier choix qui s'offre à l'utilisateur est celui des opérations qu'il souhaite utiliser en les cochant. Une fois le choix terminé, il suffit d'appuyer sur Continuer.

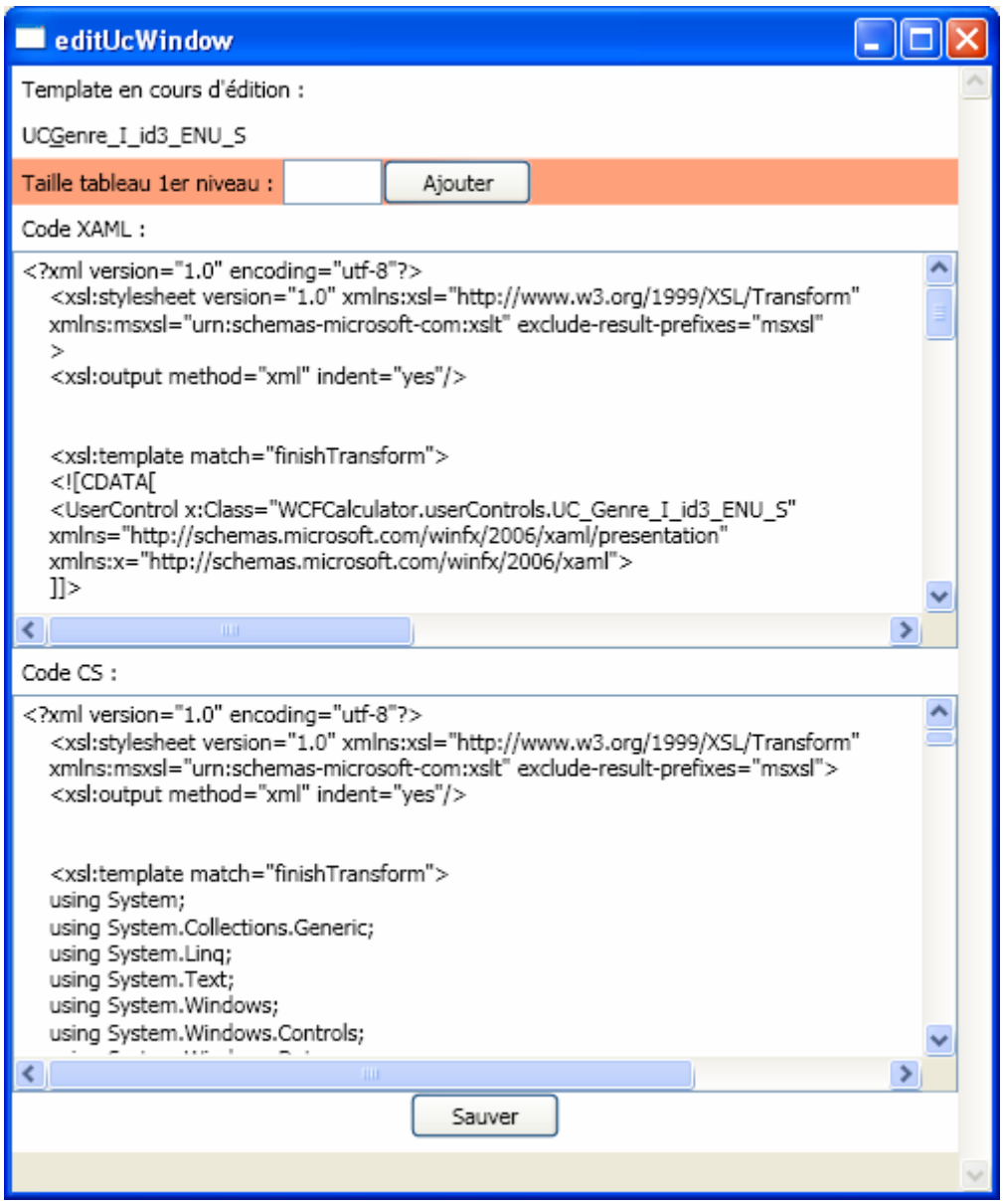


Le deuxième choix concerne les UserControls que l'on souhaite utiliser.

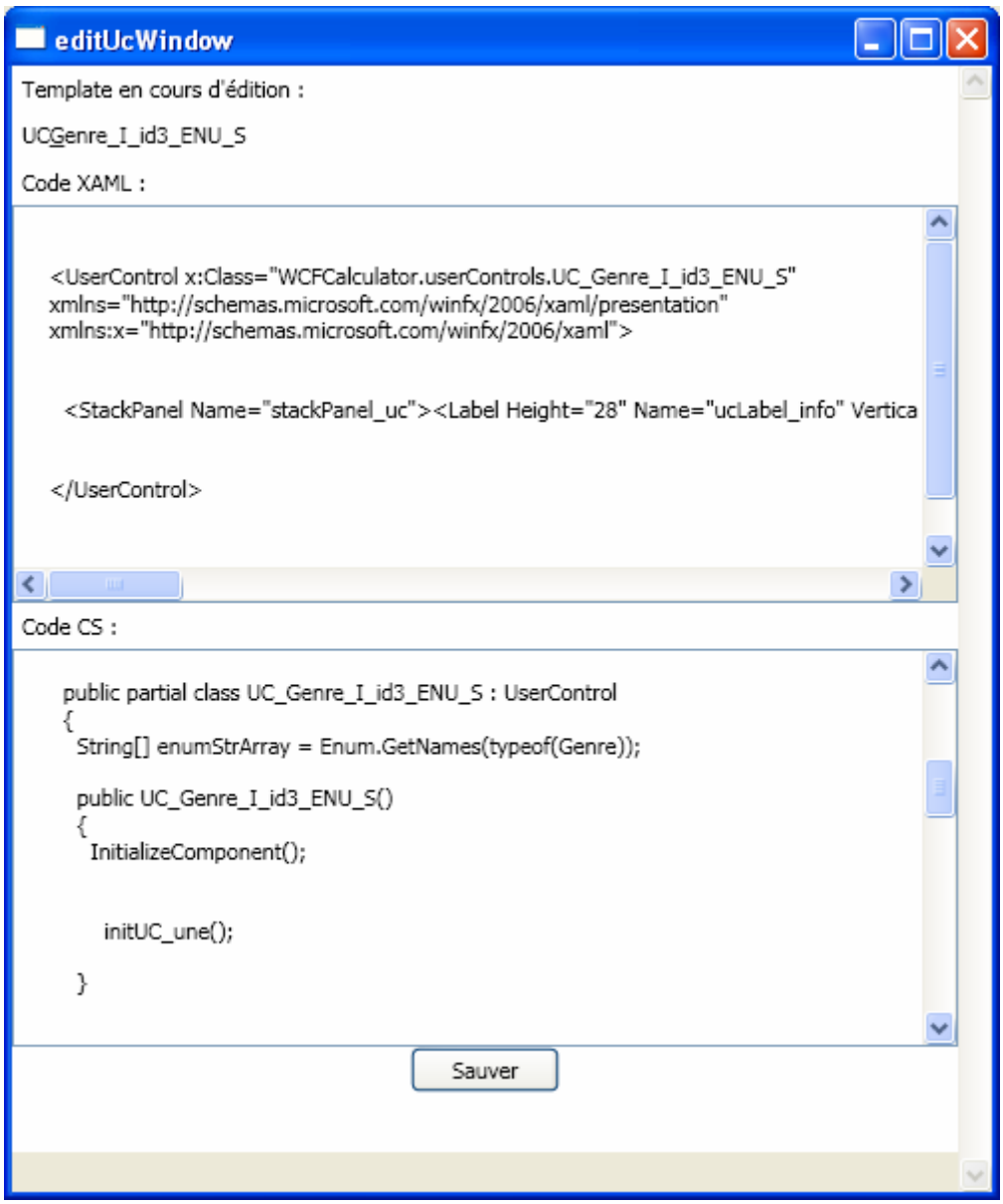


Dans ce web service, il y a un type énum et des Booleans. C'est la bonne occasion d'utiliser les UserControls spéciaux.
Lorsque l'on sélectionne un UserControl spécial dans la partie input, il faut toujours l'éditer pour que la transformation XSLT puisse être achevée correctement. Si l'on tente de générer alors que tous les UserControls ne sont pas transformés, un message en bas de la fenêtre en rouge indique les UserControls spéciaux qui doivent encore être modifiés.
Une remarque importante : une fois qu'une transformation est faite, il n'est plus possible de relancer une transformation pour l'UserControl déjà transformé.

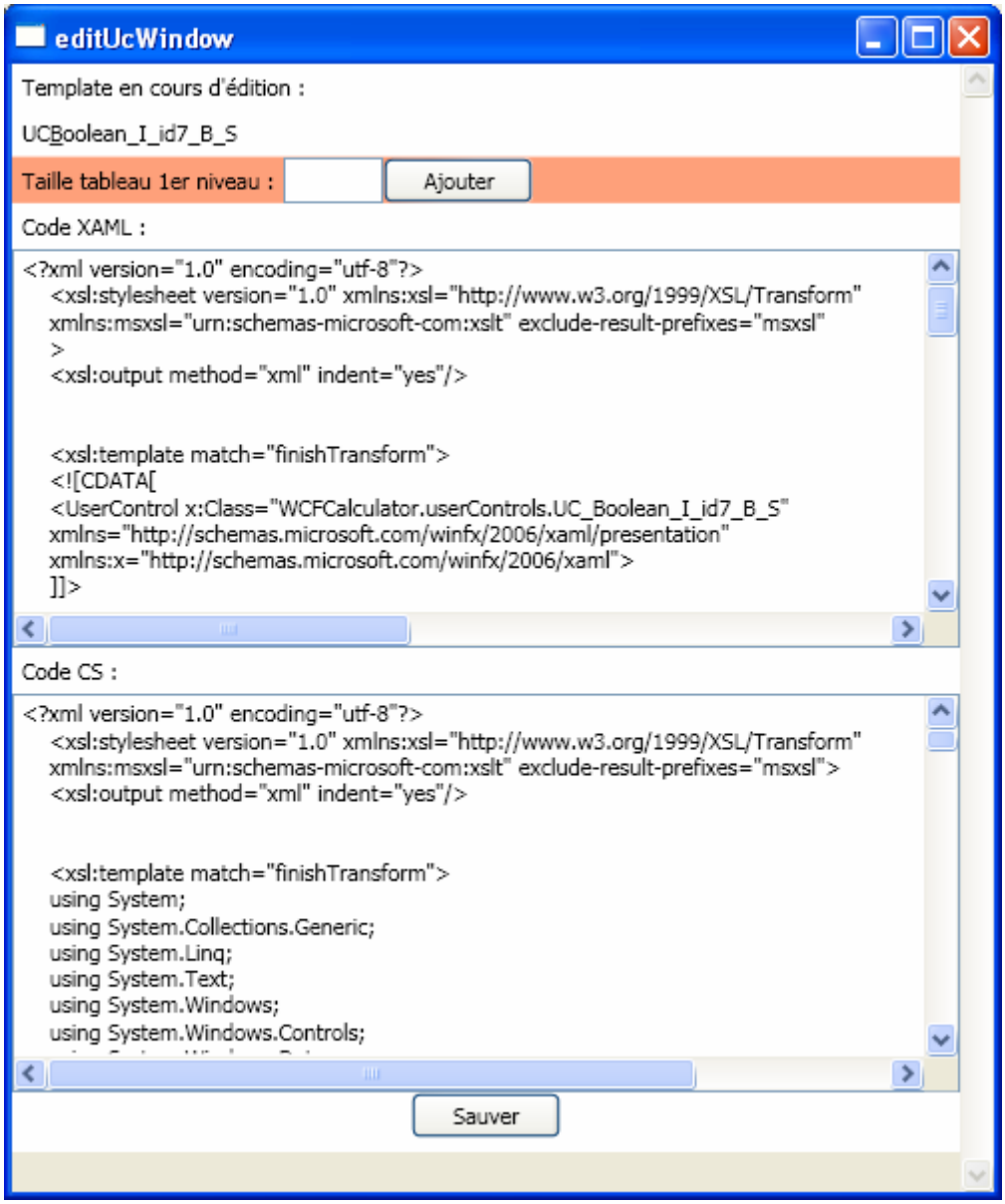
Il faut commencer par le champ Genre (id3) et éditer l'UserControl.



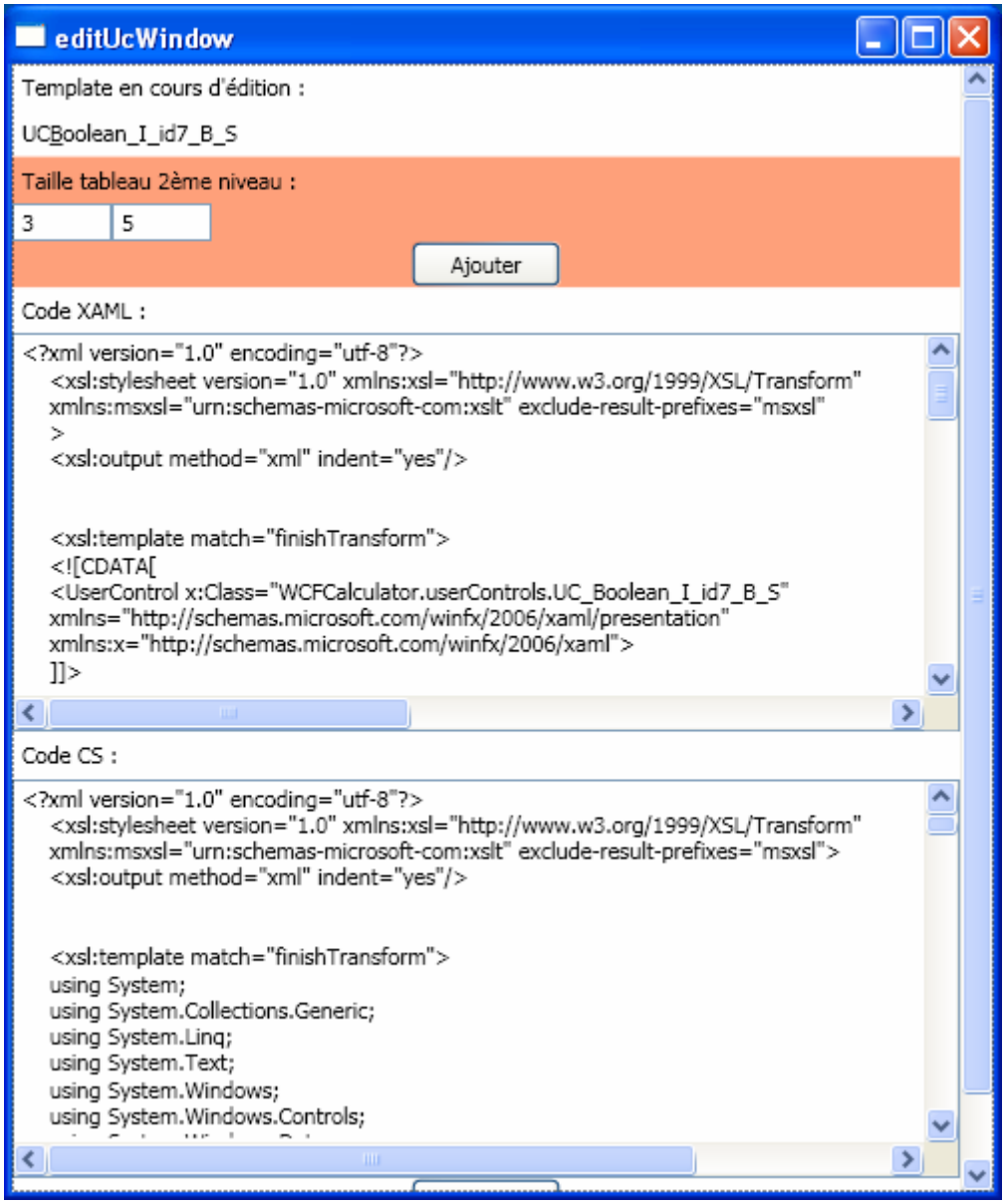
Dans cette fenêtre d'édition, il s'agit de spécifier la taille du tableau de Personne (qui est le tableau de 1er niveau). Il y a 2 Personnes à mettre (Mickey et Donald). Comme il n'y a pas d'autres tableaux à spécifier pour cet UserControl, la transformation XSLT est faite et le code du UserControl est finalisé (il n'y a plus de balise xml) et il ne reste plus qu'à sauver les modifications.



Vient ensuite l'UserControl pour le champ AmiDenfance qui est de type Boolean, id7 :



La taille du tableau du 1er niveau est identique à celui du contrôle précédent (tableau 1er niveau étant le tableau de Personnes).
Il faut ensuite spécifier les tailles des tableaux de 2ème niveau pour les tableaux d'objets de type Ami.
Pour Mickey, il y a 3 Amis (Minnie, Dingo et Pluto) et pour Donald, il y a 5 Amis.



Une fois les tailles mises et validées en cliquant sur Ajouter, il ne reste plus qu'à sauver pour terminer.

La partie Input est prête. On peut soit lancer la génération ou bien également demander à ce que dans la partie Ouput soit également affiché des UserControls spéciaux. C'est ce que l'on va faire.

Pour l'output, il n'y a pas d'intervention particulière à faire pour utiliser les UserControls spéciaux. Il suffit juste de les sélectionner. Le nombre de contrôles que chaque UserControl doit afficher sera déterminé par les données reçues du web service lors de l'utilisation de l'application.

Voici ce que donne la fenêtre avant la génération :

Si la génération s'est bien déroulée, dans la fenêtre principale apparaît le bouton permettant de lancer le client généré.

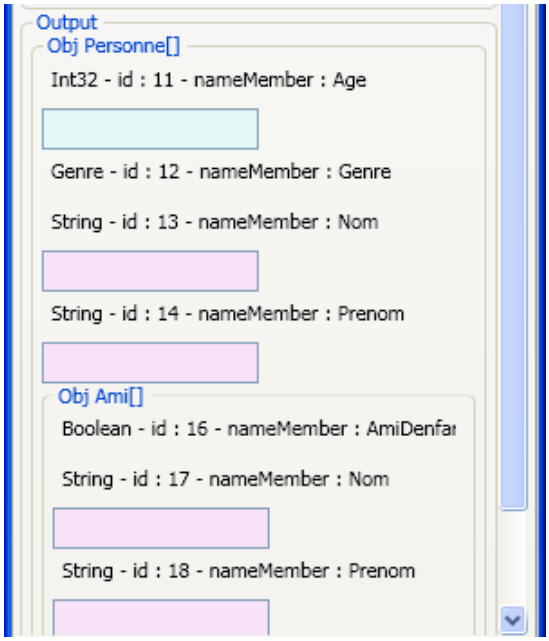
En lançant l'application, on retrouve bien l'opération que l'on a sélectionnée.

Le champ Genre contient 2 ComboBox contenant les types énumérés :
masculin
féminin
inconnu

Pour le champ AmiDenfance, on a les CheckBoxs qui ont été créées.

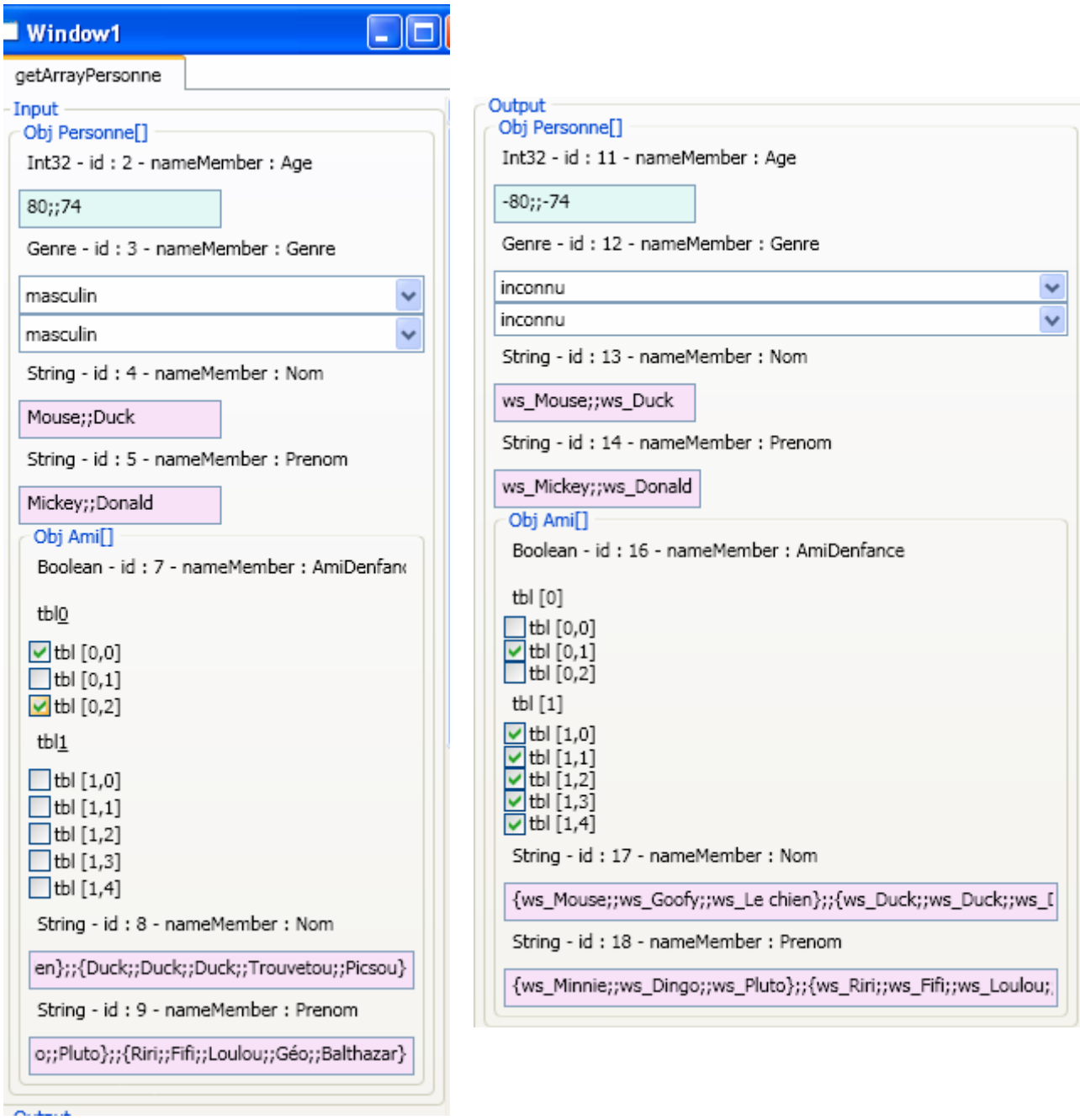
Pour la 1ère Personne (Mickey), il y a 3 Amis.

Pour la 2ème Personne (Donald), il y a 5 Amis.



Pour l'output, on remarque qu'il n'y a pas d'UserControls présents pour les champs Genre et AmiDenfance. Ils seront mis lorsque l'application sera utilisée.

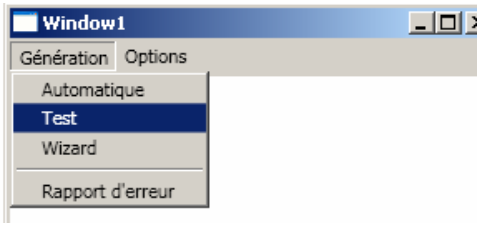
Voici le résultat obtenu :



Le traitement effectué par le web service est le suivant : l'âge retourné est négatif, tous les champs de type string sont préfixés avec ws_. Le Genre est mis à inconnu et les Booleans AmiDenfance sont inversés.
On constate que l'application a mis le bon nombre de ComboBoxs pour le champ Genre ainsi que le bon nombre de CheckBoxs pour chaque entrée du tableau de Personnes.

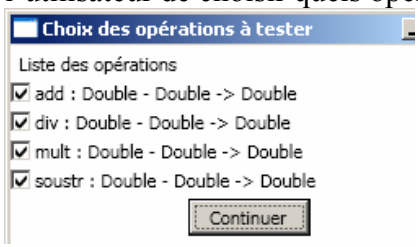
7.4.Génération test NUnit

Si l'on souhaite tester une plus grande quantité de données sans devoir à chaque fois devoir les rentrer depuis l'interface graphique, il faut prendre le choix Test.

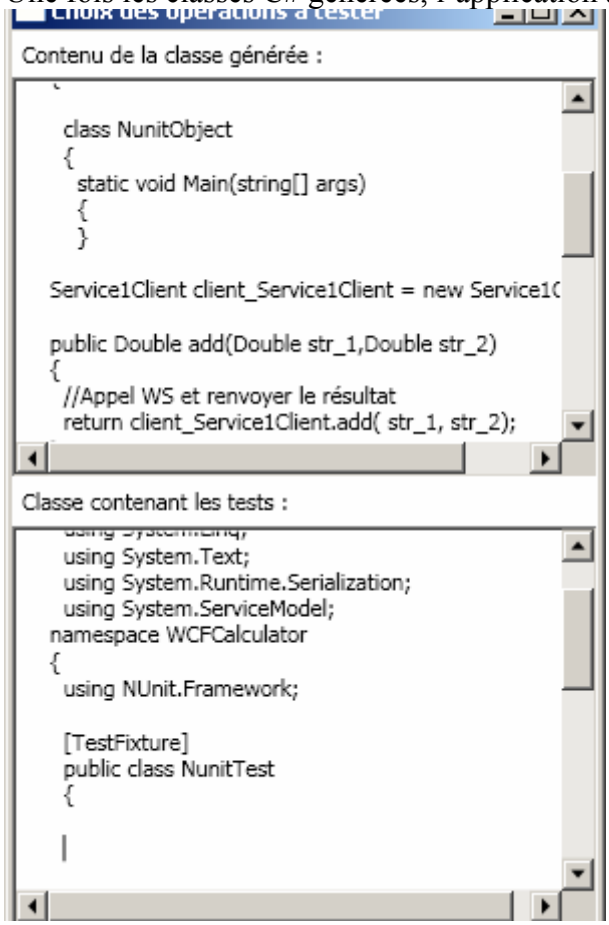


7.4.1. WCFCalculator

Après avoir lancé la génération des tests, la première étape consiste pour l'utilisateur de choisir quels opérations il souhaite tester.



Une fois les classes C# générées, l'application affiche les 2 fichiers de code :



Le TextBox du haut contient les différentes opérations que l'on peut utiliser pour tester et est juste présent comme rappel de ce que l'on peut faire et de l'objet qu'il faudra utiliser pour réaliser les tests.

Le TextBox du bas est nettement plus intéressant et utile pour ce que l'on veut faire. C'est ici qu'il va falloir rentrer les tests que l'on désire réaliser.
La classe est prête à être utilisée : il suffit juste de rentrer les tests unitaires. Voici un exemple :

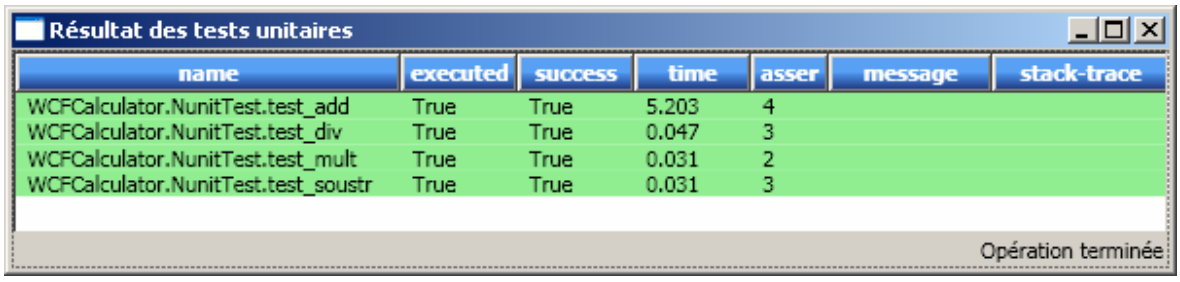
```
[Test]
public void test_add()
{
  NunitObject objC = new NunitObject();
  Assert.IsInstanceOfType(Type.GetType("System.Double"), (objC.add(4, 2)));
  Assert.AreEqual(6, objC.add(4, 2));
  Assert.AreEqual(10, objC.add(8, 2));
  Assert.Greater(20, objC.add(9, 2));
}

[Test]
public void test_div()
{
  NunitObject objC = new NunitObject();
  Assert.AreEqual(2, objC.div(4, 2));
  Assert.AreEqual(4, objC.div(8, 2));
  Assert.IsNaN(objC.div(8, 0));
}

[Test]
public void test_mult()
{
  NunitObject objC = new NunitObject();
  Assert.AreEqual(8, objC.mult(4, 2));
  Assert.AreEqual(16, objC.mult(8, 2));
}

[Test]
public void test_soustr()
{
  NunitObject objC = new NunitObject();
  Assert.AreEqual(2, objC.soustr(4, 2));
  Assert.AreEqual(6, objC.soustr(8, 2));
  Assert.AreEqual(1, objC.soustr(8, 7));
}
```

Il faut faire attention à ce que chaque opération (méthode) testée ait le tag [Test] et utilise la bonne classe pour instancier l'objet objC.

Si l'on doit comparer un type avec un autre en utilisant IsInstanceOfType, il faut toujours préciser le type en entier (Double ne sera pas la même chose que System.Double).

Une fois que tous les tests sont rentrés, il ne reste plus qu'à lancer la génération en cliquant sur Générer le test.

On obtient le résultat suivant :

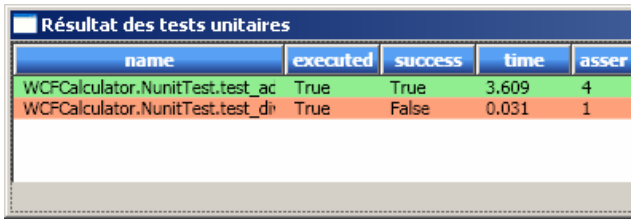


Il se peut très bien qu'un ou plusieurs tests unitaires échouent. Voici un exemple :

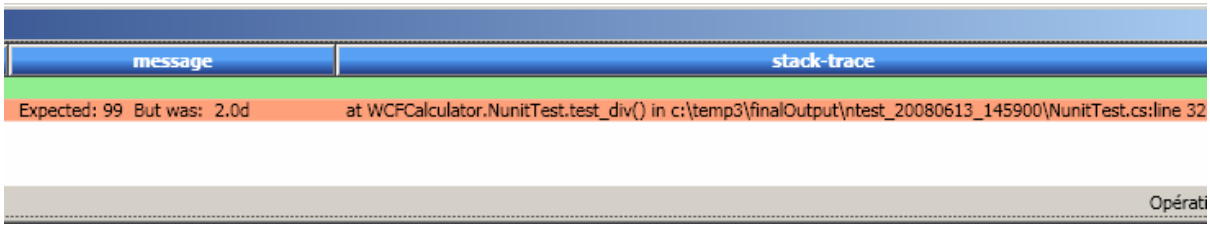
```
[Test]
public void test_add()
{
  NunitObject objC = new NunitObject();
  Assert.IsInstanceOfType(Type.GetType("System.Double"), (objC.add(4, 2)));
```

```
  Assert.AreEqual(6, objC.add(4, 2));
  Assert.AreEqual(10, objC.add(8, 2));
  Assert.Greater(20, objC.add(9, 2));
}

[Test]
public void test_div()
{
  NunitObject objC = new NunitObject();
  Assert.AreEqual(99, objC.div(4, 2));  //FAUX !!!
  Assert.AreEqual(4, objC.div(8, 2));
  Assert.IsNaN(objC.div(8, 0));
}

[Test]
public void test_mult()
{
  NunitObject objC = new NunitObject();
  Assert.AreEqual(8, objC.mult(4, 2));
  Assert.AreEqual(16, objC.mult(8, 2));
}

[Test]
public void test_soustr()
{
  NunitObject objC = new NunitObject();
  Assert.AreEqual(2, objC.soustr(4, 2));
  Assert.AreEqual(6, objC.soustr(8, 2));
  Assert.AreEqual(1, objC.soustr(8, 7));
}
```

Le résultat affiché est le suivant :



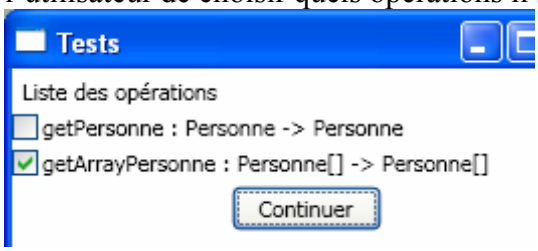
Nunit traite les assertions jusqu'à ce qu'il ait fini ou qu'une erreur s'est produite.



Si le résultat paraît étrange, il est possible d'aller consulter le fichier de résultat XML généré par NUnit. Il se trouve dans le dossier :
C:\temp3\finalOutput\<dossier de génération>\bin\Release\NunitResult.xml

7.4.2. WCFPersonne

Après avoir lancé la génération des tests, la première étape consiste pour l'utilisateur de choisir quels opérations il souhaite tester.



Une fois les classes C# générées, l'application affiche les 2 fichiers de code. Le principe est le même que pour WCFCalculator.

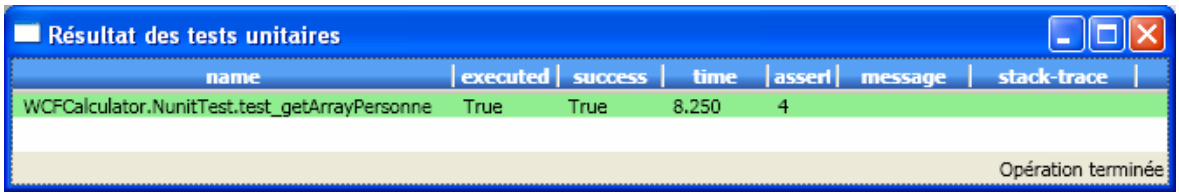
Voici un exemple pour tester le web service :

```
      [Test]
public void test_getArrayPersonne()
{
  NunitObject objC = new NunitObject();

  Personne[] pArray = new Personne[2];

  Personne p0 = new Personne();
  p0.Nom = "Mouse";
  p0.Prenom = "Mickey" ;
  p0.Age = 80;
  p0.Genre = Genre.masculin;

  Ami[] sesAmisArray0 = new Ami[3] ;
  Ami ami0_0 = new Ami();
  ami0_0.Nom = "Mouse";
  ami0_0.Prenom = "Minnie";
  ami0_0.AmiDenfance = true;
  sesAmisArray0[0] = ami0_0;
  Ami ami0_1 = new Ami();
  ami0_1.Nom = "Goofy";
  ami0_1.Prenom = "Dingo";
  ami0_1.AmiDenfance = false;
  sesAmisArray0[1] = ami0_1;
  Ami ami0_2 = new Ami();
  ami0_2.Nom = "Le chien";
  ami0_2.Prenom = "Pluto";
  ami0_2.AmiDenfance = true;
  sesAmisArray0[2] = ami0_2;
  p0.SesAmis = sesAmisArray0 ;

  pArray[0] = p0 ;


  Personne p1 = new Personne();
  p1.Nom = "Duck";
  p1.Prenom = "Donald" ;
  p1.Age = 74;
  p1.Genre = Genre.masculin;

  Ami[] sesAmisArray1 = new Ami[5] ;
  Ami ami1_0 = new Ami();
  ami1_0.Nom = "Duck";
  ami1_0.Prenom = "Riri";
  ami1_0.AmiDenfance = false;
  sesAmisArray1[0] = ami1_0;
  Ami ami1_1 = new Ami();
  ami1_1.Nom = "Duck";
  ami1_1.Prenom = "Fifi";
```

```
  ami1_1.AmiDenfance = false;
  sesAmisArray1[1] = ami1_1;
  Ami ami1_2 = new Ami();
  ami1_2.Nom = "Duck";
  ami1_2.Prenom = "Loulou";
  ami1_2.AmiDenfance = false;
  sesAmisArray1[2] = ami1_2;
    Ami ami1_3 = new Ami();
  ami1_3.Nom = "Trouvetou";
  ami1_3.Prenom = "Géo";
  ami1_3.AmiDenfance = false;
  sesAmisArray1[3] = ami1_3;
    Ami ami1_4 = new Ami();
  ami1_4.Nom = "Picsou";
  ami1_4.Prenom = "Balthazar";
  ami1_4.AmiDenfance = false;
  sesAmisArray1[4] = ami1_4;

  p1.SesAmis = sesAmisArray1 ;

  pArray[1] = p1 ;

  Personne[] pArrayWs = objC.getArrayPersonne(pArray);

  Assert.AreEqual(-80, pArrayWs[0].Age);
  Assert.AreEqual(-74, pArrayWs[1].Age);
  Assert.AreEqual("ws_Mickey", pArrayWs[0].Prenom);
  Assert.AreEqual("ws_Donald", pArrayWs[1].Prenom);

}
```

La différence qu'il y a par rapport à WCFCalculator est qu'il faut d'abord construire les objets puis appeler le web service et c'est seulement une fois le retour obtenu que l'on peut tester les valeurs.

Il y a 4 assertions : les 2 premières concernent l'âge et les 2 suivantes le prénom.
Le résultat obtenu est le suivant :



8. Limitation de l'application

8.1.Profondeur des tableaux de type personnalisé

Alors que l'application supporte différents niveaux d'objets personnalisés, ce n'est pas le cas lorsque ces objets sont des tableaux de type personnalisé. Je n'ai pas réussi à coder les parties de template pour ce genre de tableaux afin qu'ils puissent être appelés avec un apply-template (et donc, supporter n'importe quel niveau de profondeur). J'ai décidé de limiter le nombre de niveaux pouvant être traité en n'appliquant pas de template lorsque la transformation commence le traitement d'un tableau de type personnalisé. Dès que le traitement rentre dans cette partie, il ne peut plus en sortir. A l'intérieur, les templates sont écrits à chaque fois pour les différents types et ainsi de suite pour le contenu du prochain tableau de type personnalisé. Le tableau ci-dessous permet de voir les limites de profondeur des types dès qu'un tableau personnalisé (nommé Custom) est traité :

Current	**1er niveau**	**2ème niveau**	**3ème niveau**
Custom[]	Primitif		
Custom[]	Complex		
Custom[]	Custom	Primitif	
Custom[]	Custom	Complexe	
Custom[]	Custom	Custom	Primtif
Custom[]	Custom	Custom	Complexe
Custom[]	Custom	Custom[]	Primitif
Custom[]	Custom	Custom[]	Complexe
Custom[]	Custom[]	Primitif	
Custom[]	Custom[]	Complexe	
Custom[]	Custom[]	Custom	Primitif
Custom[]	Custom[]	Custom	Complexe

8.2.Web service avec état

L'application ne peut gérer que des web services sans états.
Un web service avec état serait par exemple une caisse enregistreuse : on saisit les prix des articles et à chaque fois, le web service garde le résultat des additions précédentes pour ajouter le prochain prix. Une fois que l'on souhaite le résultat, on demandera au web service la somme totale.

8.3. Types supportés

L'application supporte les types les plus courants (simple ou complex (tableau)) (String, Double, Boolean, Single, Decimal, Int64, Int32, Int16, Byte, UInt64, UInt32, UInt16, SByte, DateTime). Elle supporte également les types qui sont des énumérations (ou tableau d'énumérations) ainsi que les types personnalisés.
Le type XmlAttribute n'est pas implémenté.

8.4.tests NUnits codés à la main

Les tests doivent être développés et écrits par l'utilisateur. Cela comprend les tests en eux-même et également la préparation des différents objets nécessaires pour utiliser le web service et le traitement du retour pour ensuite tester les valeurs.
Il aurait été préférable que ce soit l'application qui s'occupe de générer le code pour la préparation des différents objets et les tests, mais cela aurait demandé énormément de

temps pour développer cette partie, donc la solution d'offrir un point d'entrée pour pouvoir ensuite étendre les fonctionnalités pour cette partie était la meilleure solution.

8.5. Retransformer un UserControl spécial

Dans la partie wizard, il n'est pas possible une fois un UserControl spécial transformé de refaire une transformation (pour changer par exemple la taille d'un tableau).

8.6. Taille des tableaux dans un UserControl spécial

Toujours dans la partie wizard, s'il y a un tableau avec 2 membres pour lesquels il faut utiliser des UserControls spéciaux, la taille du tableau doit être indiquée pour chaque UserControl (ce qui peut provoquer des erreurs si l'utilisateur rentre une taille de tableau différent pour les 2 UserControls).

8.7. TimeOut des réponses du web service pour l'application générée

Par défaut, les paramètres de timeout pour l'ouverture, la fermeture et l'envoi sont à 1 minutes (et 10 pour la réception) et il n'y a pas de moyen de modifier ceci qu'en modifiant le fichier de configuration à la main.

9. Bugs ou problèmes connus

9.1. Génération de code C#

Il y a un problème de génération lorsqu'un type d'objet personnalisé contient un ou des tableaux d'objets personnalisés (des portions de code sont écrites alors qu'ils ont déjà été écrits). Dans la partie servant à traiter les tableaux personnalisés, il y a un test qui sert à déterminer si le type actuellement traité ne contient pas un ancêtre qui serait un tableau personnalisé.
Exemple dans match="parametre/type" mode="output1" :

```
<!-- tbl custom -->
      <xsl:when test="contains(@name,'[]') and not(@name='String[]') and
not(@name='Double[]') and not(@name='Boolean[]') and
not(@name='Single[]') and not(@name='Decimal[]') and not(@name='Int64[]')
and not(@name='Int32[]') and not(@name='Int16[]') and not(@name='Byte[]')
and not(@name='UInt64[]') and not(@name='UInt32[]') and
not(@name='UInt16[]') and not(@name='SByte[]') and
not(@name='DateTime[]') and not(@name='Object[]')">
        //custom[]

        <xsl:if test="not(contains(../../@name,'[]')) and
not(contains(ancestor::parametre/type/../../@name,'[]'))">
```

Ce test doit permettre d'éviter que le code de traitement qui suit soit écrit car lorsque l'on arrive la première fois dans cette partie, il n'y a jamais d'ancêtre qui est un tableau d'objets personnalisés. Tout le traitement pour les enfants se fait à l'intérieur.
Mais comme il y a des apply-templates effectués, tôt ou tard, le traitement va s'occuper des enfants du tableau personnalisé. Or ces enfants auront déjà été traités et il ne faut pas les retraiter une seconde fois. Ce test doit permettre d'éviter ce problème. Le bug se situe sur la façon de contrôler s'il y a un ancêtre qui est un tableau personnalisé.

Il faudrait modifier comme suit :

```
<xsl:if test="not(contains(../../@name,'[]')) and
```

```
not(ancestor::*[(contains(@name,'[]'))])">
```

Cette modification n'a pas été faite car le problème a été identifié tardivement et que modifier un template XSLT peut provoquer d'autres dysfonctionnement ensuite dans la génération et il aurait fallu refaire tous les tests (ce qui représente de nombreuses heures pour tout retester).

9.2.Code pour les tests NUnits

Les assemblies sont écrits en dur…

9.3.Comportement de NUnit

NUnit se comporte de façon étrange lorsqu'il y a beaucoup de tests. Il semble se bloquer. Je ne pense pas que le problème vienne de mon application ou du code que je mets (si c'est le cas, ça ne devrait soit pas générer, soit mettre une erreur dans le rapport de NUnit mais certainement pas traiter correctement les 2 premières fois puis bloquer ensuite).

9.4.Différents clients dans le code généré

Pour certains web services, il y a une partie des opérations qui utilise un certain client et le reste des opérations un autre client. Pour la génération automatique, les 2 sont écrits. Pour la génération wizard, même si un des clients n'est pas utilisé, il est tout de même écrit. Ca ne gêne pas l'utilisation et ça fonctionne correctement, mais c'est une ligne de code inutile.

9.5.Des objets nuls

C'est un cas de figure qui n'a pas été pris en compte. Si ça se produit, cela peut provoquer des erreurs lors de l'utilisation de l'application. Il y a néanmoins un petit contrôle effectué sur les objets personnalisés reçu du web service. S'ils sont nuls, ils ne sont pas traités.
Cela m'amène ensuite au problème d'afficher le résultat de l'objet nul. Les contrôles des UserControls affichent les valeurs par défaut (dans le cas d'un string, rien, pour un int, c'est un zéro qui est reçu par l'UserControl et affiché). Mais rien n'indique que l'objet soit nul.
D'autre part, lors de l'utilisation d'un UserControl spécial pour le type Boolean, si les valeurs reçues en retour sont nulles, l'application générera une erreur avec les CheckBoxs car il n'y a pas d'état nul pour des CheckBoxs.

9.6.Fenêtre de sélection des opérations

Il n'y a pas de scroll prévu pour le cas où les éléments de la liste dépasseraient la résolution d'affichage de l'écran.

9.7.Fenêtre de sélection des UserControls dans le wizard

L'affichage n'est pas des plus pratiques. Visuellement, rien n'indique si le type est contenu dans un objet ou un tableau d'objets personnalisés ce qui complique pour l'utilisateur ce qu'il doit rentrer comme taille de tableau pour les UserControls spéciaux.
Il manque un système de GroupBoxs qui permettrait de délimiter le contenu des objets comme c'est le cas pour l'application générée.

9.8.Choix des bindings

Il y a 2 problèmes pour les bindings. Le premier vient de l'outil svcutil, il envoie une erreur lorsqu'il faut traiter des bindings de type http. Vient ensuite le cas où il y a plusieurs bindings possibles (SOAP 1.1 et SOAP 1.2). Par défaut, l'application ne prend en compte que le premier qu'il trouve. Rien n'a été prévu pour les cas où il y aurait plusieurs bindings. Il aurait fallu faire en sorte de demander à l'utilisateur le binding qu'il souhaite utiliser.

10. Evaluation de l'application

Dans l'ensemble, l'application répond aux objectifs qui ont été fixés.
Elle permet de générer une application dont l'interface utilisateur est composée d'éléments XAML ainsi que le code behind en C# basée sur la description WSDL de web services.
Il est possible de personnaliser le client généré en choisissant les opérations du web service que l'on souhaite intégrer et pour certains types, d'utiliser un UserControl différent pour personnaliser l'affichage.
Et finalement, elle offre la possibilité d'utiliser des tests unitaires en proposant les fichiers de bases pour la création d'un client destiné aux tests et dont l'utilisateur n'a plus qu'à rentrer les tests qu'il souhaite effectuer (l'application se chargeant ensuite d'exécuter ces tests à l'aide de NUnit et d'afficher le résultat).

Malgré tout, l'application n'est de loin pas parfaite.

Elle souffre d'une trop grande limitation sur la structure des données que peuvent avoir certains web services. Même si dans une grande partie des cas, pouvoir gérer 2 niveaux de profondeurs contenant un ou 2 tableaux d'objets personnalisés sera suffisant, c'est tout de même restreignant. S'ajoute ensuite l'impossibilité de pouvoir choisir le binding qui doit être utilisé. On peut certes modifier les fichiers générés pour choisir l'endpoint à utiliser puis refaire une compilation des fichiers, mais cela aurait été mieux d'y avoir pensé avant et de l'avoir inclus dans l'application.
Les TextBoxs qui peuvent servir à l'édition des fichiers de codes sont une idée mal exploitée qui n'apportent pas de confort et d'aide à l'utilisateur. Il aurait été plus judicieux de lui laisser le choix sur le programme à utiliser pour l'édition.
La partie wizard permettant de choisir les UserControls manque de clareté au niveau de l'affichage de la structure des données. Séparer visuellement les différents types par objet comme c'est le cas pour l'application générée aurait été un grand plus pour s'y retrouver, surtout si le web service à une structure de données complexe et volumineuse.
Toujours dans la partie wizard, la gestion de la taille des tableaux pour les UserControls enfants n'est pas bien conçue car devoir saisir plusieurs fois la taille d'un même tableau pour différents UserControls est une perte de temps et un risque d'erreurs.

Finalement, l'architecture pour la création d'UserControls spéciaux n'est pas bonne. Elle est dépendante de la limitation de la génération de code. Si cette limitation peut être résolue, il faudra également modifier le contenu de la méthode generateIt de la classe GenerateWizardClient.
De plus, le choix des UserControls spéciaux n'est pas très grand. Si l'on souhaite créer de nouveaux UserControls spéciaux, il y aura pas mal de modifications à faire dans cette partie et il faudra également éditer les fichiers de transformation XSLT. Il aurait été préférable de faire un système permettant de créer plus facilement des UserControls en mettant en place soit un petit wizard pour les créer ou un système modulaire dans lequel on poserait le nouveau template de transformation pour créer l'UserControl ainsi que le fichier de données XML.

Bien que cette application puisse être utilisée pour de nombreux web services et simplifie la vie du développeur, elle n'est pas suffisamment bonne pour devenir LA référence incontournable que tous les développeurs voudront posséder.

11. Evolution future de l'application

Si le projet devrait être repris après ce travail de diplôme, il faudrait se pencher sur les points suivants pour améliorer l'application :

- Résoudre la limitation de la profondeur des types de tableaux personnalisés. L'idéal serait d'arriver à gérer jusqu'à 5 niveaux de profondeur (le 5ème niveau n'ayant que des types primitifs, complexe, énums ou tableau d'énums) ce qui permettrait de traiter la majorité des cas.
- Résoudre le problème pouvant survenir avec des objets nuls et la façon d'afficher une valeur nulle.
- Pouvoir choisir le binding (par exemple du SOAP 1.1 ou SOAP 1.2)
- Remplacer les TextBoxs servant à l'édition de code par la possibilité d'éditer les fichiers en utilisant un éditeur par défaut choisi par l'utilisateur (ça peut être notepad ou Visual Studio par exemple).
- Résoudre le problème pour les UserControls spéciaux dans la partie wizard de la saisie des tailles des tableaux. Il faut éviter que pour 2 UC enfants d'un tableau parent, il faille saisir lors de l'édition de ces 2 UCs à chaque fois la taille (une fois saisie pour le 1er UC, l'autre devrait pouvoir bénéficier de l'information.
- Il n'y a pas assez de choix pour personnaliser les contrôles.
- Modifier la partie de création des UserControls (méthode generateIt() de la classe GenerateWizardClient). Il faudrait pouvoir rendre cette partie plus facilement modifiable et se pencher sur un système permettant à l'utilisateur de pouvoir créer ses propres UserControls et les mettre à disposition de l'application sans devoir modifier une ligne de code.
- Intégrer l'application à Visual Studio sous forme d'un AddIn.
- Développer un système automatique permettant de créer des tests unitaires. Cela doit comprendre la création des objets, l'interaction avec le web service et le traitement du retour du web service ainsi que la génération des tests unitaires.

12. Conclusion

L'analyse de risque que j'ai faite au début du projet a été trop optimiste. J'ai largement sous-estimé la complexité que représentait la génération de code pour créer des fichiers C# et XAML.

La moindre erreur de code dans un fichier généré entraîne dans le meilleur des cas un échec lors de la compilation des fichiers et dans le pire des cas, la compilation passera mais c'est lors de l'utilisation que les problèmes surviendront (plantage ou comportement faussé de l'application générée). Ces problèmes peuvent très vite devenir coûteux en temps et plus la génération est complexe, plus le risque est grand. D'autre part, plus la génération est complexe et plus il y aura de chances que le changement d'une condition de tests dans un template entraîne des erreurs ailleurs. Même en ayant créé beaucoup de web services pour tester l'application et utilisé des web services trouvés sur Internet, cela n'a pas empêché la découverte de bugs tardivement dans des endroits que je considérai comme bons.

Je n'ai également pas assez analysé la génération de code avant de commencer à créer les templates. J'ai payé cette erreur en cours de projet lors que j'ai été confronté aux tableaux

de type d'objets personnalisés. Je n'avais pas prévu ce cas de figure et les templates créés ne pouvaient pas être simplement modifiés pour supporter ce genre de type. Il a fallu repartir de zéro.

Mon manque d'expérience dans la création d'une application s'est ressenti lors du développement. J'ai passé du temps à chercher comment résoudre des problèmes courants tels que le fait de ne pas avoir une application qui reste figée lors d'un long traitement. Ce manque d'expérience a également affecté l'architecture et de l'ergonomie de l'application (notamment pour la partie wizard qui n'est pas assez intuitive et efficace).
De plus, je n'ai pas assez pensé à l'utilisateur final, l'exemple étant l'utilisation de TextBoxs pour l'édition du code.

Si le travail était à refaire, je m'y prendrai différemment. Je commencerai comme je l'ai fait par étudier le XAML et le XSLT en faisant des tests puis étudier et comprendre le fonctionnement des web services. Je prendrai plus de temps sur ce que l'on peut faire comme traitement et ainsi éviter d'arriver vers la fin du projet et de remarquer qu'un cas de figure comme les web services avec état ne soit pas supporté.

Ensuite, je ferai une recherche exhaustive des web services se trouvant sur le web. Cela permettrait d'éviter de découvrir tardivement des types de données et de devoir tenter des les intégrer aux templates déjà existants.
Créer des web services pour tester plus facilement les différents types et structures de données possibles et réfléchir plus en détail sur la façon de créer ces web services pour pouvoir tester plus facilement l'application (charger à chaque fois un web service, le tester et passer au suivant prenant beaucoup de temps).

Finalement, avec toutes ces informations, travailler sur papier pour créer l'architecture de l'application et se mettre à la place de l'utilisateur : qu'est-ce que j'attends d'une application comme ça, comment faudrait-il faire pour gagner un maximum de temps lorsque j'utilise l'application, quel sera le cheminement à suivre pour exécuter une fonctionnalité, est-ce que je peux utiliser mes outils préférés…

Une fois l'architecture définie, la créer afin de se rendre compte de comment fonctionne l'ensemble et ensuite, se pencher sur la partie génération de code en développant les templates et en testant au fur et à mesure les différents éléments des templates.

Au final, ce travail de diplôme fut enrichissant. Bien que cela m'ait permis d'accroître mes connaissances au niveau du XAML, de la transformation XSLT et d'explorer un peu plus les web services, il me reste encore indéniablement un très long chemin à parcourir avant d'avoir une meilleure maîtrise de ces éléments et l'expérience nécessaire pour réaliser une application bien construite répondant parfaitement aux attentes de l'utilisateur.

13. Annexes

13.1. Définitions

13.1.1. Web service

Les web services sont des petits éléments de code qui sont construits pour s'occuper d'une tâche bien précise. Ils s'appuient sur des protocoles de communication qui utilisent le XML. Ils sont indépendants du système d'exploitation qui les exécute et du langage de programmation utilisé. Leur tâche est de connecter aussi bien des gens que des systèmes ou des appareils entre eux. Les avantages d'utiliser les web services sont de rendre plus facile la communication entre applications, la réutilisation de services existants, la distribution d'informations à beaucoup de clients ou utilisateurs du service et sont rapides à développer.

13.1.2. W3C

Le World Wide Web Consortium s'occupe de développer et de mettre en place des spécifications et standards pour le web pour une meilleure interopérabilité entre les différents services possibles.

13.1.3. WSDL

Le WSDL (Web Services Descriptions Language) est un document écrit en XML qui décrit un web service. Il spécifie l'adresse du service et les différentes méthodes exposées. Il représente un contrat entre le fournisseur du service et le consommateur de ce service. De plus, le WSDL est indépendant de la plateforme et du langage utilisé.
Une recommandation (ou guide de spécification) a été émise en 2001 par le W3C. Cette recommandation spécifie la version 1.1 du WSDL et est suivie par Microsoft pour le WCF.
En juin 2007, une nouvelle recommandation a été émise faisant évoluer le WSDL vers la version 2.0 (avec comme changement entre autres des modifications sur les noms de différents composants).
A noter que cette recommandation n'est pas supportée dans WCF 3.0/3.5 pour une utilisation telle quelle mais qu'il existe quelques possibilités pour prendre en charge certains points. Cette recommandation est à l'étude pour une éventuelle implémentation dans des versions futures selon Dave Cliffe, modérateur du forum MSDN.

13.1.4. Namespaces

L'espace de nom correspond à une collection de noms qui est identifiée par un URI et que l'on utilise entre autres dans des documents XML. Cela permet de définir pour un nom une certaine signification par rapport à un certain namespace.

13.1.5. WCF

WCF (Windows Communication Foundation) est la nouvelle couche de communication du framework 3.0. Elle permet de faire communiquer des applications entre elles sur la même machine ou sur différentes machines. Ceci existe déjà depuis longtemps (remoting, web services…) mais apporte au développeur plus de facilité pour développer ses applications sans se soucier du moyen de communiquer ou de la technologie utilisée.

Le code développé ne devra pas être changé pour utiliser un autre moyen de transport par exemple. C'est WCF qui se charge de faire cette transformation pour envoyer les données.

13.1.6. XAML

XAML (eXtensible Application Markup Language) est un langage déclaratif qui permet construire les interfaces graphiques des applications. L'énorme avantage est de pouvoir séparer les éléments de description d'interface des éléments de code, ce qui permet de mieux répartir les tâches entre plusieurs développeurs/designers et simplifie le développement de l'application.

13.1.7. CodeDom (Code Document Object Model):

Le CodeDOM est une API qui permet de générer du code source (structure, objets,…) dans plusieurs langages de programmation lors de l'exécution. Cette API est disponible dans le SDK du framework .Net et permet de générer du C#, JScript et du Visual Basic.

13.1.8. XSLT (EXtensible Stylesheet Language):

Le XSLT est un langage de template qui permet de convertir un fichier XML dans un autre format (par exemple du HTML) en utilisant un fichier de transformation.

13.2. Gestion de projet

13.2.1. Cahier des charges

Le cahier des charges pour ce projet est le suivant :

- Développer un générateur de fichiers XAML à partir de descriptions de Web services en WSDL.
- Développer une application qui intègre ce générateur et qui permette de créer de manière configurable des écrans personnalisables.
- Créer des fichiers de tests de base pour ces Web services, qu'un développeur pourra étendre pour ses propres besoins.
- Ecrire une documentation de projet qui inclue: i) une description des technologies utilisées, des requis (requirements) ainsi que choix entrepris; ii) une description du design de l'architecture du générateur et de l'application; iii) un tutorial avec des Web services existants; iv) une analyse des résultats atteints et des améliorations à donner.

13.2.2. Analyse des risques

Nature du risque	**Degré du risque pour ce projet**						
		0	**1**	**2**	**3**	**4**	**5**
Taille du projet			X				
Difficulté technique				X			
Degré d'intégration			X				
Configuration organisationnelle			X				
Changement			X				
Instabilité de l'équipe de projet			X				

Taille du projet : 1
Le projet est de taille normale. A la vue des différentes fonctionnalités qu'il faut implémenter, je devrais pouvoir le séparer en 3 morceaux. Commencer la génération automatique serait une bonne chose, car ça me permettrait ensuite de m'appuyer sur cette partie développée pour la partie wizard (car en gros, c'est un traitement automatique avec des choix de l'utilisateur) et dans une moindre mesure pour la partie destinée aux tests.

Difficulté technique : 2
Les difficultés résident dans le manque de connaissances au niveau du XAML et WCF, des web services (notamment au niveau du WSDL), le peu d'expérience dans la création d'application et dans une moindre mesure la génération de code.
Pour réduire le risque :
Au niveau du XAML, il faut apprendre comment se construit une fenêtre, quels sont les contrôles importants et comment générer du contenu XAML depuis le code.
Pour le WCF, consulter le site MSDN sur comment fonctionnent les web services.
Au niveau du WSDL, il faut comprendre les différentes parties qui constituent le document, où se trouvent les informations dont j'ai besoin et quels sont les moyens pour que je puisse les obtenir dans mon application. La complexité pourrait être réduite s'il y a un moyen simple d'obtenir les informations du WSDL.
Au niveau de la génération de code, faire des tests afin de comprendre mieux comment utiliser les différents éléments et comment se déplacer dans une arborescence XML, mais je ne pense pas que ce soit trop difficile de générer du code.
Il faudra penser à faire de nombreux tests (développer des web services pour tester l'application) et également tester quelques web services sur Internet.
En ce qui concerne le peu d'expérience dans la création d'application, malheureusement, à part faire des tests et lire des articles sur Internet, je ne vois pas ce qui pourrait réduire le risque de problèmes.

Degré d'intégration : 1
Il y a des chances pour que j'utilise des outils externes pour réaliser certaines opérations. Il faudra prendre du temps pour lire la documentation des outils qui seront utilisés et faire des tests pour voir comment l'application que je crée pourrait interagir avec et utiliser les résultats provenant des traitements des outils externes à l'application. La piste à suivre sera certainement de faire des appels en console.

Configuration organisationnelle : 1
Le nombre d'intervenants se limite à 2 personnes dans le cadre du développement. Pour gagner en efficacité et éviter de multiplier les séances, un blog sera créé pour le suivi des rapports hebdomadaires ce qui permet de centraliser à une place l'information sur le projet et qui permet de retrouver rapidement ce qui a été fait les semaines précédentes.

Changement : 1
Actuellement, il n'y a rien qui permet de faire simplement et rapidement un client pour un web service, donc, l'utilisation de cette application ne devrait pas rencontrer d'hostilités quand à son utilisation, elle devrait même susciter un certain engouement. Par contre, cet engouement pourrait être réduit suivant les limitations que l'application pourrait avoir.

Instabilité de l'équipe de projet : 1
Le risque se situe principalement sur la transmission des connaissances acquises durant ce travail. Afin de réduire au maximum cette perte, le rapport devra être le plus complet possible et offrir aux futurs développeurs une solution agréable pour trouver rapidement les informations recherchées (par exemple, ce que fait une classe ou une méthode).

13.2.3. MS Project

Date de début du projet : Ven 24.08.07
Date de fin du projet : Dim 22.06.08

Détails du projet

Code WBS	Tâche	Durée planifiée	Durée	Date début	Date Fin	Travail planifié	Travail réel
1	**Projet WSDL & XAML**	67.94 jours	89.06 jours?	Ven 24.08.07	Dim 22.06.08	543.5 hr	706 hr
1.1	**Pré-Projet**	3 jours	3 jours	Ven 24.08.07	Ven 07.09.07	24 hr	24 hr
1.1.1	Tests divers pour identifier les besoins	24 hr	24 hr	Ven 24.08.07	Ven 07.09.07	24 hr	24 hr
1.2	**Mise en route du Projet**	3.19 jours	11.19 jours?	Mar 16.10.07	Dim 13.01.08	25.5 hr	105.5 hr
1.2.1	Séance début projet	1.5 hr	1.5 hr	Mar 16.10.07	Mar 16.10.07	1.5 hr	1.5 hr
1.2.2	Etablissement Cahier des charges + blog	12 hr	12 hr	Dim 21.10.07	Mar 23.10.07	12 hr	12 hr
1.2.3	Etablissement Fichier de suivi de projet (MS Project)	4 hr	4 hr	Mar 30.10.07	Mar 30.10.07	4 hr	4 hr
1.2.4	**Mise en place des outils (pc maison)**	1 jour	9 jours?	Mer 31.10.07	Dim 13.01.08	8 hr	88 hr
1.2.4.1	Installation Framework 3.5	1 hr	0.33 hr	Mer 31.10.07	Mer 31.10.07	1 hr	0.33 hr
1.2.4.2	Mise en place suivi de projet sous MS Project	0 jour?	4 hr	Mer 31.10.07	Jeu 01.11.07	0 hr	4 hr
1.2.4.3	Installation VS 2008	3 hr	3 hr	Dim 04.11.07	Dim 04.11.07	3 hr	3 hr
1.2.4.4	Recherche et documentation sur wsdl, outils, XAML	0 jour?	74 hr	Dim 04.11.07	Dim 13.01.08	0 hr	74.67 hr
1.2.4.5	Recherche et mis en place des exemples wsdl (MS)	4 hr	2 hr	Dim 13.01.08	Dim 13.01.08	4 hr	2 hr
1.2.4.6	Installation VS 2008 version finale	0 jour?	0.88 jour?	Sam 05.01.08	Dim 06.01.08	0 hr	4 hr
1.3	**Récupérations Informations du WSDL**	9.5 jours	2.75 jours	Jeu 24.01.08	Lun 28.01.08	76 hr	22 hr
1.3.1	Recherche documentation outils MS du framework 3.5	8 hr	4 hr	Jeu 24.01.08	Jeu 24.01.08	8 hr	4 hr
1.3.2	Récupérer informations sur les méthodes exposées	16 hr	10 hr	Jeu 24.01.08	Lun 28.01.08	16 hr	10 hr

1.3.3	Récupérer informations sur les types utilisés	16 hr	2 hr	Lun 28.01.08	Lun 28.01.08	16 hr	2 hr
1.3.4	Projet .net regroupant les fonctionnalités de récupération du WSDL	10 hr	6 hr	Lun 28.01.08	Lun 28.01.08	10 hr	6 hr
1.4	**Compilation (génération) automatique**	2.5 jours	0.5 jour	Lun 28.01.08	Mar 29.01.08	20 hr	4 hr
1.4.1	Rechercher documentation sur compilation (cmd)	8 hr	3 hr	Lun 28.01.08	Mar 29.01.08	8 hr	3 hr
1.4.2	Projet .net permettant de compiler un projet et intégration. Résultat exec console	12 hr	1 hr	Mar 29.01.08	Mar 29.01.08	12 hr	1 hr
1.5	**Génération de code**	0 jour?	2.88 jours	Mer 30.01.08	Dim 03.02.08	0 hr	23 hr
1.5.1	recherche de moyens pour générer le code	0 jour?	4 hr	Mer 30.01.08	Mer 30.01.08	0 hr	4 hr
1.5.2	apprentissage et documentation sur le moyen trouvé	0 jour?	12 hr	Ven 01.02.08	Sam 02.02.08	0 hr	12 hr
1.5.3	implémentation dans un projet xaml	0 jour?	7 hr	Sam 02.02.08	Dim 03.02.08	0 hr	7 hr
1.6	**Projet .net XAML**	0 jour?	9.5 jours	Dim 03.02.08	Mer 20.02.08	0 hr	71 hr
1.6.1	Projet .Net pour générer une application XAML (que pour le WS du sdk)	0 jour?	24 hr	Dim 03.02.08	Dim 10.02.08	0 hr	24 hr
1.6.2	Correction bug, tests, types costumes et complex, options	0 jour?	47 hr	Ven 15.02.08	Mer 20.02.08	0 hr	47 hr
1.7	**Gestion des options et tests**	9.75 jours	5.5 jours	Ven 22.02.08	Dim 02.03.08	78 hr	44 hr
1.7.1	Mise en place d'éléments pour paramètrer l'application de génération et résolution des bugs	24 hr	20 hr	Ven 22.02.08	Dim 24.02.08	24 hr	20 hr
1.7.2	Développement d'une partie "fichiers tests"	24 hr	24 hr	Dim 24.02.08	Dim 02.03.08	24 hr	24 hr
1.8	Tests d'un web service et corrrection de bug	0 jour?	19 hr	Dim 02.03.08	Sam 08.03.08	0 hr	19 hr
1.9	séance	0 jour?	1.5 hr	Lun 10.03.08	Lun 10.03.08	0 hr	0 hr
1.10	**Nouvelle orientation**	0 jour?	29.69 jours	Mar 11.03.08	Sam 10.05.08	0 hr	237.5 hr
1.10.1	Corrections génération de bugs	0 jour?	11.5 hr	Mar 11.03.08	Mer 12.03.08	0 hr	11.5 hr
1.10.2	**Génération code : redéveloppement**	0 jour?	9.5 jours	Jeu 13.03.08	Sam 29.03.08	0 hr	76 hr

1.10.2.1	pseudo code + trace	0 jour?	24 hr	Jeu 13.03.08	Dim 16.03.08	0 hr	24 hr
1.10.2.2	Coder le pseudo code	0 jour?	18 hr	Dim 16.03.08	Sam 22.03.08	0 hr	18 hr
1.10.2.3	Debug et essais	0 jour?	34 hr	Sam 22.03.08	Sam 29.03.08	0 hr	34 hr
1.10.3	Wizard : identifier les éléments à rendre paramétrable	0 jour?	4 hr	Dim 30.03.08	Dim 30.03.08	0 hr	4 hr
1.10.4	Gestion des tests : exploration de nunit	0 jour?	4 hr	Dim 30.03.08	Dim 30.03.08	0 hr	4 hr
1.10.5	Expression : essayer et regarder comment sont faits les éléments (grid)	0 jour?	4 hr	Jeu 03.04.08	Jeu 03.04.08	0 hr	4 hr
1.10.6	**Web services sur le web**	0 jour?	2.76 jours	Ven 04.04.08	Dim 06.04.08	0 hr	23 hr
1.10.6.1	tests	0 jour?	6 hr	Ven 04.04.08	Ven 04.04.08	0 hr	6 hr
1.10.6.2	Implémenté type Enum et tbl Enum	0 jour?	8 hr	Ven 04.04.08	Sam 05.04.08	0 hr	8 hr
1.10.6.3	Corrections de divers bugs	0 jour?	8.1 hr	Sam 05.04.08	Dim 06.04.08	0 hr	9 hr
1.10.7	Modification UserControls	0 jour?	8 hr	Ven 11.04.08	Ven 11.04.08	0 hr	8 hr
1.10.8	Création d'un exemple concret	0 jour?	6 hr	Sam 12.04.08	Sam 12.04.08	0 hr	6 hr
1.10.9	**Nouvelle version de la partie test**	0 jour?	6 jours	Sam 12.04.08	Sam 26.04.08	0 hr	48 hr
1.10.9.1	Implémenter test avec utilisation de Nunit	0 jour?	36 hr	Sam 12.04.08	Ven 25.04.08	0 hr	36 hr
1.10.9.2	tests et exemple	0 jour?	12 hr	Ven 25.04.08	Sam 26.04.08	0 hr	12 hr
1.10.10	**Partie wizard**	0 jour?	6.63 jours	Sam 26.04.08	Sam 10.05.08	0 hr	53 hr
1.10.10.1	Création du code	0 jour?	53 hr	Sam 26.04.08	Sam 10.05.08	0 hr	53 hr
1.11	Refactoring	0 jour?	48 hr	Sam 10.05.08	Sam 24.05.08	0 hr	48 hr
1.12	Correction de bugs	0 jour?	18 hr	Sam 24.05.08	Ven 30.05.08	0 hr	18 hr
1.13	Tests	0 jour?	12 hr	Ven 30.05.08	Sam 31.05.08	0 hr	12 hr
1.14	SandCastle	0 jour?	6 hr	Dim 01.06.08	Dim 01.06.08	0 hr	6 hr
1.15	Documentation	0 jour?	72 hr	Dim 01.06.08	Dim 22.06.08	0 hr	72 hr

13.3. Outils utilisés

13.3.1. Visual Studio 2005

Le projet ayant commencé en septembre 2007, Visual Studio 2008 n'était pas encore disponible, c'est pourquoi j'ai utilisé Visual Studio 2005 avec l'installation du patch dotNetFx35setup.exe pour prendre en charge le framework 3.0 (les différences entre la version 3.0 et 3.5 ne posant pas de problème au niveau des web services), ceci afin de pouvoir créer des web services avec les connaissances apprises en cours. D'autre part, les exemples fournis dans le sdk utilisent cette version de Visual Studio. Il était préférable de continuer sur cette voie que de convertir les projets pour la version 2008 avec les problèmes que cela aurait pu apporter.
L'inconvénient de cette méthode est que toutes les fonctionnalités pour faciliter le développement du XAML par exemple ne sont pas présentes. Ainsi, un double click sur un bouton ne génère pas la méthode Click dans le code behind.
C'est pourquoi dès que Visual Studio 2008 beta 2 est sorti, j'ai l'ai utilisé suivi de la version finale (Visual Studio 2008 Professional).
La beta 2 (avec le framework 3.5 beta) a permis de se familiariser avec le nouvel environnement de développement (très semblable à la version 2005 patchée) et pouvoir tester plus facilement l'utilisation des éléments XAML.

13.3.2. Visual Studio 2008 beta 2

Au niveau de l'installation, rien de particulier à signaler, il suffit de suivre le wizard d'installation.
Problèmes rencontrés :
Outre quelques bugs dérangeants dont la fermeture intempestive du programme par moments, j'ai rencontré un problème lors de la création de nouveaux projets. Le message d'erreur était le suivant :
The build was aborted because the "HostLogger" logger failed unexpectedly during initialization.
La réparation n'a pas permis de résoudre le problème.

En cherchant sur Internet, 2 possibilités pour résoudre ce problème :

- Désinstaller Microsoft Windows Software Development Kit (6000.0.0) puis faire une réparation (http://207.46.236.188/MSDN/ShowPost.aspx?PostID=2491702&SiteID=1)
- Désinstaller AddOn Studio for WOW qui demande une version RTM de VS 2008 (Les versions RTM et beta sont incompatibles) et refaire une réparation de la beta (http://www.codeplex.com/WarcraftAddOnStudio/WorkItem/View.aspx?WorkItemId=724)

Ces 2 solutions n'ayant pas résolu le problème, j'ai désinstallé tout ce qui était en version beta (framework, Visual Studio, MSDN library) et installé la version finale de Visual Studio 2008.
Finalement, la création de nouveaux projets a fonctionné mais un autre problème est survenu : impossible de faire un drag & drop depuis la toolbox.
Impossible également de compiler sans obtenir dans la fenêtre de design le message d'erreur suivant :

> Could not load file or assembly 'Microsoft.Windows.Design.Developer, Ver-sion=0.0.4032.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a' or one of its dependencies. The system cannot find the file specified.

La solution a été trouvée sur le forum MSDN (http://forums.microsoft.com/MSDN/ShowPost.aspx?PostID=2633726&SiteID=1):

- Sortir de Visual Studio
- Se rendre dans le répertoire %USERPROFILE%\AppData\Local\Microsoft\VisualStudio\9.0
- Effacer tous les fichiers portants l'extension TBD (par défaut, ils sont cachés). Redémarrer Visual Studio et ouvrir une application WPF. La toolbox sera réinitialisée automatiquement.

13.3.3. Visual Studio 2008 Professional

Au niveau de l'installation, rien de particulier à signaler, il suffit de suivre le wizard d'installation. Au préalable, il a fallu désinstaller tout ce qui était issu de la version beta de Visual Studio.

13.3.4. MS Project

La version 2003 a été utilisée.
Problème rencontré :
Des problèmes pour sauver le fichier dans d'autres formats (par exemple en .htm). Cela vient du fait que Windows à le SP 3 et que MS Project n'avait que le SP 1 d'installé et certains formats sont bloqués. Pour résoudre le problème, j'ai dû installer le SP 3 pour MS Project et ensuite, j'ai pu sauver les fichiers dans le format que je souhaitais.

13.3.5. SDK 6.0A et 6.0

Le kit de développement qui contient svcutil. Il est installé en même temps que Visual Studio 2008.

13.3.6. Notepad++

Petit éditeur de code source qui supporte différents langages et permet d'avoir une meilleure vue des éléments que dans le traditionnel Notepad.
Site officiel :
http://notepad-plus.sourceforge.net/fr/site.htm

13.3.7. Microsoft Expression 2.0 beta

Cette solution permet de designer des écrans, l'arrangement et le contenu faisant un peu penser à Photoshop d'Adobe. Ce programme a été utilisé dans l'optique de découvrir comment faire des tableaux avec différentes couleurs. Je pensai qu'il y aurait une déclinaison des éléments XAML de base et qu'il y aurait toute une série de templates prêt à l'emploi ce qui m'aurait permis de voir au niveau du code comment c'est construit. Malheureusement, à ce niveau, l'application est vide. Est-ce que cela vient du fait que cette version est en beta et que tout n'est pas encore présent ?
Quoi qu'il en soit, on retrouve les contrôles de base mais sans plus. Certes, il est plus agréable de travailler sur ce logiciel que sous Visual Studio mais il faudra néanmoins mettre la main dans le code pour réaliser quelque chose : par exemple, je n'ai pas trouvé comment faire depuis l'interface graphique afficher les lignes des cellules du grid et y mettre une couleur. De plus, l'Intellisense semble absent.
Au final, ce programme n'est pas d'une grande nécessité pour mes besoins.

13.3.8. NUnit

NUnit est un programme de tests unitaires pour tous les langages .Net. Il peut être exécuté aussi bien en mode graphique qu'en mode console. Au niveau de l'installation, rien de particulier à noter, il suffit de suivre le wizard pour l'installer.
Site officiel :
http://www.nunit.org/index.php

13.3.9. SandCastle & DocProject 2008

SandCastle et DocProject permettent de générer une documentation de code basée sur les commentaires contenus dans les fichiers C# du projet. La documentation est produite sous la forme d'un fichier .chm et permet de trouver plus rapidement des informations sur une classe ou une méthode qu'en consultant un document Word.
A noter que la prise en charge de la documentation pour les fichiers XSLT n'existe pas (et c'est pourquoi, cette documentation a été mise dans le rapport).
Site officiels :
http://www.codeplex.com/DocProject
http://www.codeplex.com/Sandcastle
Au moment de la réalisation de ce rapport, SandCastle a été retiré de codeplex.com. (https://blogs.msdn.com/sandcastle/)

Pour l'installation, il faut commencer par installer SandCastle puis DocProject.

Pour créer une documentation, 2 cas possibles :

- Si la solution à documenter contient déjà un projet DocProject, il n'y a rien de particulier à effectuer. A chaque fois qu'une modification dans le projet de l'application est faite et qu'un build est lancé, la documentation sera remise à jour.
  Vu que cela peut prendre de 5 à 10 minutes pour construire ou reconstruire la documentation, il est préférable soit de mettre la génération de la documentation une fois que le projet est terminé ou de lancer les builds en sélectionnant uniquement le projet de l'application (et ainsi, la documentation ne sera pas buildée).
- Si la solution à documenter ne contient pas de projet DocProject, il faut en créer un.
  Dans Visual Studio, File -> Add -> New Project et choisir DocProject comme template (il est préférable de laisser le nom proposé par défaut pour éviter d'éventuels bugs).
  Maintenant que le projet est présent dans la solution, un wizard permet de choisir les options suivantes :
  Choisir le moteur de génération : prendre SandCastle (sélectionné par défaut).
  Au niveau de la présentation, prendre Hana.
  Choisir comment l'aide doit être compilé (en général, la version1.x).
  Si besoin, faire les modifications que l'on souhaite pour le contenu partagé (pas indispensable).
  Finalement, choisir le projet qui doit être documenté.

  Il ne reste plus qu'à contrôler dans les propriétés de l'application que la documentation XML lors d'un build est faite. Avant de lancer un build, fermez Visual Studio et relancez-le, ceci afin d'éviter un problème au démarrage de la génération.

13.4. Choix techniques

13.4.1. Comment récupérer les informations du WSDL

J'ai effectué des recherches sur Google, MSDN Online et Codezone Community afin de trouver quels sont produits (open source ou commercial) actuellement disponible et répondant de près ou de loin à l'objectif général de ce travail de diplôme.

13.4.1.1. Visual Studio 2005

Par défaut, VS 2005 ne supporte pas le XAML. Il faut installer Microsoft .NET framework 3.0, Windows SDK pour la documentation, exemples et outils de développement et finalement Visual Studio 2005 extensions pour intégrer les nouvelles fonctionnalités de WPF (Windows Presentation Foundation) et WCF (Windows Communication Foundation) dans Visual Studio.
En consultant les différents types de projet possible, une nouvelle entrée nommée .NET Framework 3.0 est disponible mais rien n'est prévu concernant la génération d'un client graphique en ne fournissant que le WSDL.
Dans les autres types de projet, il n'y a également rien de prévu.

13.4.1.2. Visual Studio 2008

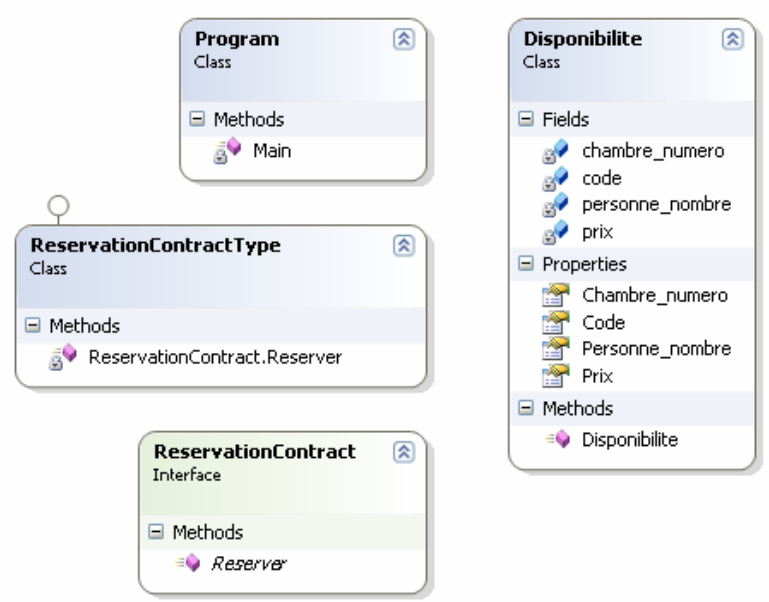
Cette nouvelle version de VS nécessite la nouvelle version du framework .net 3.5 et entre autres nouveautés, elle bénéficie d'un designer XAML nommé cider. En consultant sur le site de MSDN à propos des nouveautés apportées par le nouvel environnement de développement et le nouveau framework, rien ne permet de dire qu'il y a des outils permettant de réaliser mon objectif.

13.4.1.3. Programmes et plugins tiers

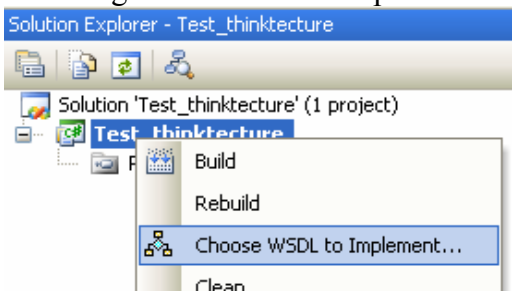
Je n'ai trouvé qu'un seul projet répondant à une partie de ma problématique : WSCF - Web Services Contract First créé par la société thinktecture.
Ce plugin est destiné à être intégré dans Visual Studio 2005 (un exécutable est également disponible) et permet de réaliser les opérations suivantes en suivant des wizards :

- Générer le WSDL à partir des définitions metadata pour les données, messages et les interfaces.
- Générer le code (C# ou VB .Net) côté serveur
- Générer le code côté client

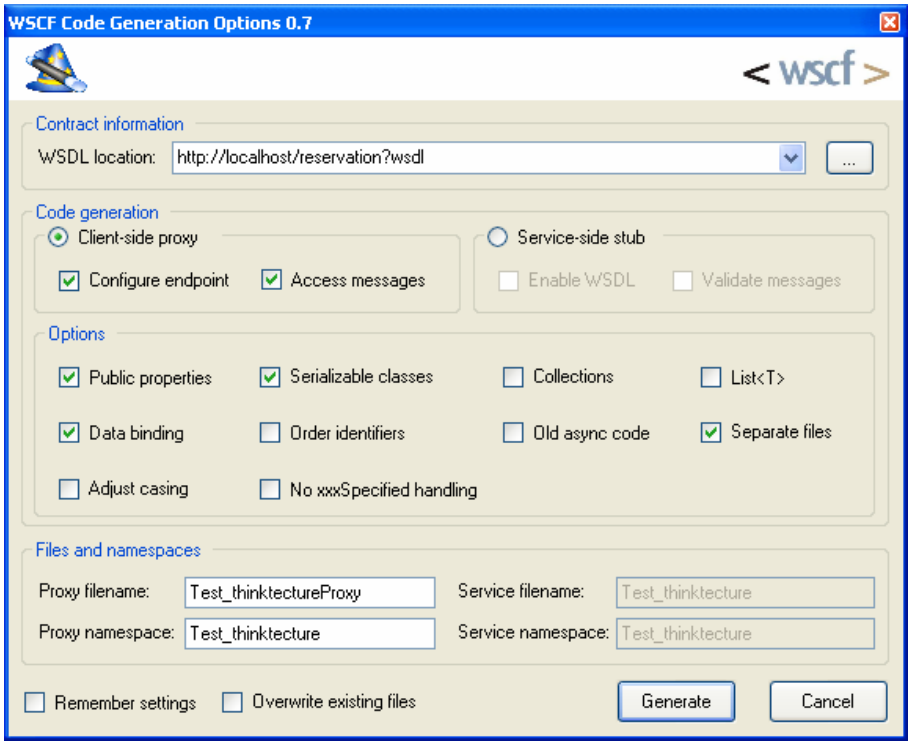
Afin d'avoir une meilleure idée de quoi est capable ce plugin, j'ai effectué un petit test avec un webservice que j'avais créé pour un test : ReservationService.



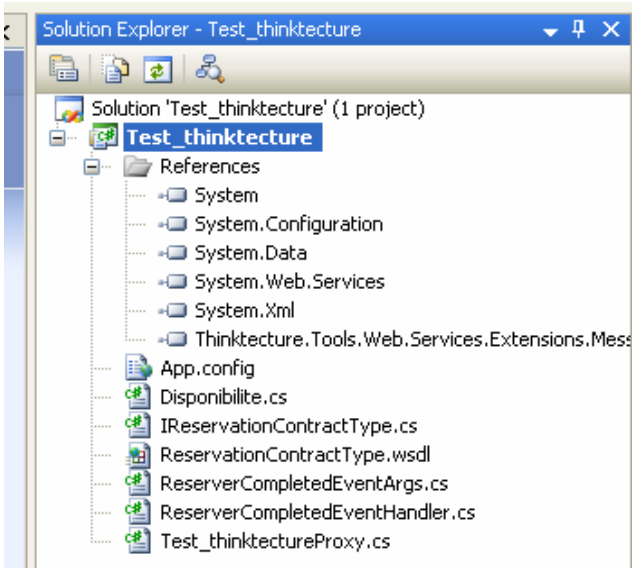
J'ai créé un nouveau projet C# (Windows Application WPF sinon il n'y aura pas d'initialisation de la fenêtre Windows) nommé Test_thinktecture et j'ai lancé la génération du client par le WSDL.



Dans la fenêtre d'options affichées, j'ai spécifié les éléments suivants puis générer :



Le plugin a créé les éléments suivants (l'option separate files permettant de séparer les éléments créés dans chaque fichier) et a ajouté les références nécessaires :

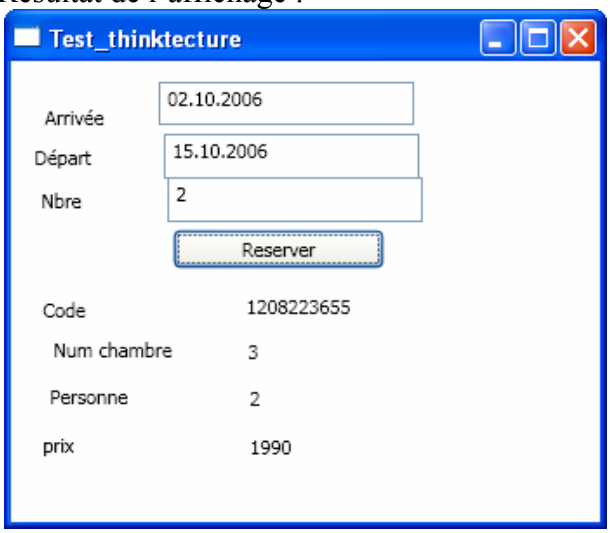


Ensuite, j'ai codé ma fenêtre d'affichage et fait un simple run pour voir si le client arrive à communiquer avec le web service et s'il reçoit une réponse.

Code .cs pour la fenêtre d'affichage :

```
namespace Test_thinktecture
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>
    public partial class Window1 : System.Windows.Window
    {
        ReservationContractType rct;
        public Window1()
        {
            InitializeComponent();
        }
        public void clickIt(object sender, EventArgs e)
        {
            rct = new ReservationContractType();

            Disponibilite d = rct.Reserver(
                DateTime.Parse(this.textBox1.Text),
                true,
                DateTime.Parse(this.textBox2.Text),
                true,
                Int32.Parse(this.textBox3.Text),
                true
                );
            this.a1.Content = d.Code;
            this.a2.Content = d.Chambre_numero;
            this.a3.Content = d.Personne_nombre;
            this.a4.Content = d.Prix;
        }
    }
}
```

Résultat de l'affichage :



Au final, tout semble fonctionner et de manière assez simple. Je note toutefois une différence dans les paramètres passés à la méthode Reserver par rapport à la version du client dont le proxy a été généré avec Visual Studio :

```
localhost.Disponibilite d = a.Reserver(
    DateTime.Parse(this.textBox1.Text),
    DateTime.Parse(this.textBox2.Text),
    Int32.Parse(this.textBox3.Text)
    );
```

Je constate que d'une part, il n'y a pas d'interface prévue pour pouvoir directement utiliser le client et qu'il y a une référence concernant l'outil qui a généré le code.
Malgré tout, cela reste intéressant dans la mesure où l'on a les fichiers nécessaires pour construire une application graphique sur cette base et qu'il y a également la possibilité d'utiliser le mode console pour automatiser la génération.

Avantages et désavantages d'utiliser WSCF :
Pour :

- Toute la problématique concernant la création des fichiers destinés à l'application client par rapport au WSDL pourrait être résolue en utilisant ce service en mode console et ensuite puiser les éléments pour pouvoir construire l'application générée.

Contre :

- Dépendant d'un service extérieur sur un élément de clé du projet, la récupération des informations du WSDL.
- Avenir du produit : quels seront les prochains développements ? Compatibilité avec VS 2008 et le framework .Net 3.5 ?
- Contrat de licence : libre d'utilisation mais dans le cas où l'on souhaite distribuer notre application en incluant WSCF, il faudra faire la demande à thinklecture.
- Pourrait poser des problèmes pour déployer l'application finale de génération.

Mon avis :
Bien que l'utilisation de WSCF est séduisante et permettrait d'avancer plus rapidement dans mon travail de diplôme, je n'y suis pas très favorable.

Ne pas l'utiliser permet de rester libre sur le futur de l'application et de ce que l'on veut en faire une fois le travail terminé.
Ensuite, vient le problème de la réutilisation des éléments fournis par WSCF dans un projet final. Il serait bien que l'on puisse facilement récupérer les portions de code (ou l'entier du code) afin de pouvoir les intégrer dans une autre application que celle générée. Or il y a des références spéciales qui sont faites pour WSCF afin de pouvoir communiquer avec le WSDL (entre autres). Cela voudrait dire qu'il faudra également intégrer au projet final ces références ce qui n'est peut-être pas la meilleure solution.
Cette solution a été laissée de côté.

13.4.1.4. Parser le fichier WSDL

N'ayant rien trouvé qui soit réellement intéressant pour obtenir les informations du WSDL, je me suis tourné vers la solution de parcourir le fichier WSDL (comme il est basé sur du XML, ce ne devait pas être trop difficile de récupérer les informations dont j'ai besoin). Après quelques essais, le résultat était mitigé. Certes, j'ai réussi à récupérer quelques informations mais le travail à effectuer est immense et obtenir à la fin une solution qui fonctionne à tous les coups me semble difficile. Cette solution a été laissée de côté.

13.4.1.5. wsdl.exe & svcutil.exe

Etant toujours à la recherche d'une solution pas trop compliquée et qui ne prenne pas trop de temps, je me suis posé la question suivante : Comment Visual Studio 2008 fait pour générer un proxy lorsque l'on utilise la commande Add Service Reference?
La réponse : il utilise Svcutil.exe.
Cet outil permet de générer un fichier proxy.cs (qui contient toutes les informations du WSDL) et un fichier de configuration.
En consultant la documentation du framework, il y a 2 outils permettant de créer le proxy et le fichier de configuration.

Wsdl.exe
Wsdl.exe permet de générer les fichiers nécessaires pour faire un proxy à partir du contrat WSDL et des schémas XSD. Contrairement à SvcUtil.exe, Wsdl.exe n'est pas un outil WCF et n'est à utiliser que dans le cas où les classes du proxy vont être intégrées dans un client qui n'est pas un client WCF.

Svcutil.exe
WCF a introduit ce nouvel outil pour générer un proxy. Les fichiers (classes et le fichier de configuration) qui sont générés sont conçus pour l'infrastructure WCF.
Tout comme Wsdl.exe, il est possible d'utiliser Svcutil.exe en mode console.

13.4.1.6. Solution retenue

L'utilisation de Svcutil a été retenue pour les raisons suivantes :

- Il génère de manière fiable les fichiers proxy et config (certainement bien plus que si je parse moi-même le fichier WSDL).
- Il est possible de l'utiliser en mode console, donc, je pourrai l'appeler dans mon application pour générer au bon moment les fichiers dont j'ai besoin.
- Je développe une application qui utilise le framework 3.0+, donc, svcutil doit être utilisé

- Réduit sensiblement le temps et la complexité de la récupération des informations du WSDL.

13.4.2. Comment générer du code

Pour générer le code, il y a 3 possibilités : utiliser le StreamWriter, CodeDom et XSLT.

13.4.2.1. StreamWriter

Cette solution consiste à écrire au fur et à mesure le code dans un fichier texte. Cette méthode va très bien pour générer rapidement une petite portion de code à des fins de tests mais cela va très vite devenir difficile pour écrire dynamiquement des éléments de code et n'est de loin pas facile à structurer et à comprendre la séquence de traitement (et encore plus si le projet est délaissé pendant un certain temps).

13.4.2.2. CodeDom

Pour générer du code répétitif, le CodeDom semble être un bon outil. Il a l'énorme avantage de pouvoir générer du code dans différents langages mais je ne suis pas convaincu que ce choix soit judicieux dans le cadre de ce projet.
La génération du code que je dois faire est basée sur des web services que l'utilisateur a choisis. Chacun de ces web services est la plupart du temps différent, donc cela donnera des fichiers de code différents.

En ayant consulté quelques exemples qui utilisent du CodeDom, tout le traitement pour construire une méthode par exemple, est faite dans le code C#. Cela fait beaucoup de lignes de code pour générer quelque chose mais le plus dérangeant, c'est la lecture de ce code : il est très difficile de se représenter le code qui sera généré en regardant le code de construction.
D'autre part, cela semble assez compliqué à mettre en place pour avoir à disposition un générateur qui puisse assembler différents bouts de code pour construire une classe spécifique à un web service.
Et finalement, il faut également coder la partie graphique XAML qui ne contient que du XML, donc CodeDom ne conviendrait pas.

La preuve en code avec un exemple tiré du site techheadbrothers.com pour générer une méthode :

```
//Creation d un constructeur avec parametres
CodeConstructor ParamCtor = new CodeConstructor();
ParamCtor.Comments.Add(new System.CodeDom.CodeCommentStatement("le constructeur
avec paramtres"));
//definition de l'attriubt d'acces
ParamCtor.Attributes = MemberAttributes.Public;
//Definition des parametres d'entrees age et nom
CodeParameterDeclarationExpression cpd = new
CodeParameterDeclarationExpression(typeof(string), "nom");
CodeParameterDeclarationExpression cpd1 = new
CodeParameterDeclarationExpression(typeof(int), "age");
ParamCtor.Parameters.Add(cpd);
ParamCtor.Parameters.Add(cpd1);

//Produit this.m_age=age et this.m_name=nom
CodeFieldReferenceExpression fieldRefName = new CodeFieldReferenceExpression(new
CodeThisReferenceExpression (),codeMemberName.Name);
CodeFieldReferenceExpression fieldRefAge = new CodeFieldReferenceExpression(new
CodeThisReferenceExpression (),codeMemberAge.Name);
CodeAssignStatement codeAssignName = new CodeAssignStatement(fieldRefName, new
CodeArgumentReferenceExpression(cpd.Name));
```

```
CodeAssignStatement codeAssignAge = new CodeAssignStatement(fieldRefAge, new
CodeArgumentReferenceExpression(cpd1.Name));

ParamCtor.Statements.Add(codeAssignName);
ParamCtor.Statements.Add(codeAssignAge);

//j'ajoute le constructeur a ma classe
codeClass.Members.Add(ParamCtor);
```

Le code qui sera généré :

```
// le constructeur avec paramtres
public Humain(string nom, int age)
{
    this.m_name = nom;
    this.m_age = age;
}
```

13.4.2.3. XSLT

Le XSLT est un langage de template qui permet de créer n'importe quel type de texte, XML ou HTML. Vu que les codes sources sont du texte, XSLT peut être utilisé pour les produire (même le XAML qui est au final du XML).
En utilisant le XSLT, on simplifie d'une certaine manière la vie du développeur et la maintenance future de l'application. Ceci parce qu'il est plus simple que codeDOM pour éditer ou rechercher un élément dans le contenu. Il y a également une vision plus parlante lorsque l'on consulte le fichier XML (qui contient les données) et le fichier XSLT contenant la structure utilisée pour créer le nouveau document, ce qui permet de savoir plus rapidement à quoi correspond une partie du code ou pour identifier un problème.
Cependant, tout n'est pas parfait. Alors que codeDOM peut produire différents langages de code depuis un fichier, ce ne sera pas le cas avec le XSLT (il faudrait créer un fichier xslt par langage). De plus, il faudra prendre du temps pour apprendre ce nouveau langage (et dans l'idéal avoir un peu d'expérience pour utiliser au mieux les capacités du XSLT).

13.4.2.4. Choix retenu

Malgré ces quelques petits « défauts », XSLT semble la meilleure solution pour générer le code des clients des web services. C'est donc ce moyen qui a été retenu pour générer du code.

13.4.3. Comment créer un exécutable

Mon choix s'est porté naturellement sur l'utilisation de MSBuild.exe après avoir regardé comment Visual Studio fonctionne pour créer un exécutable. Il présente l'avantage de pouvoir générer aussi bien une application WPF qu'une application console de la même manière mais nécessite un fichier csproj pour lui indiquer quels sont les fichiers qui doivent être utilisés, quel type de génération il faut faire et si l'on veut obtenir une application console ou Window. La complexité d'utilisation est très faible et le fichier csproj peut être assez facilement repris et modifié d'un projet existant pour créer un template afin de pouvoir générer une application de différente manière.
De plus, il est possible de pouvoir le piloter en mode console.

13.4.4. Comment créer des tests unitaires

Pour pouvoir tester des web services, il faut soit créer son propre système de tests, soit utiliser un système existant.
La solution de créer son propre système a été mise de côté. Beaucoup de travail

pour le réaliser et au final, le résultat sera inférieur à ce que d'autres outils peuvent faire.

Mon choix s'est porté sur l'utilisateur de NUnit présent depuis de nombreuses années car il propose un model d'assert permettant d'effectuer les opérations les plus communes telles que les égalités, identités, comparaisons, types, conditions.

De plus, il est simple à utiliser et à préparer le code pour effectuer des tests, il dispose d'une version console permettant le lancement de tests par d'autres programmes et finalement, il propose le résultat des tests sous forme d'un fichier XML qui peut être parsé pour en retirer les résultats obtenus.

13.5. Créer un web service et un client avec VS 2008

13.5.1. Web service

L'opération est assez simple. Dans Visual Studio, créer un nouveau projet en utilisant le template WCF service (dans la partie web).

Par défaut, 2 fichiers importants sont créés :
Service1.svc :
Contient pour chaque opération du web service le traitement qui doit être effectué
IService1.cs :
Contient 2 éléments. Le premier est l'interface contenant le ServiceContract. C'est ici qu'il faut lister toutes les opérations que peut faire le web service (un peu à la manière d'un fichier .h en cpp).
Chaque opération doit être précédée de l'information OperationContract (sinon, l'opération ne sera pas disponible).
Le 2ème élément est optionnel. Il s'agit du DataContract qui définit pour chaque classe personnalisée comment elle est structurée. Chaque propriété doit être précédée de l'information DataMember afin qu'elle puisse être sérialisée.

Pour faciliter l'utilisation, modifier dans les propriétés du projet le port du serveur de développement sur lequel on peut atteindre le WSDL. J'utilise par défaut le port 1999.

Exemple :
Une opération getPersonne prend comme paramètre un objet Personne qui est transformé par le web service avant de renvoyer l'objet Personne.
Cet objet Personne contient un nom et prénom de type string, l'âge de type int et son genre qui est un type Genre (énume).

Service.svc.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace WcfService1
{
    public class Service1 : IService1
    {
        public Personne getPersonne(Personne p)
        {
```

```
            p.Nom = "ws_" + p.Nom;
            p.Prenom = "ws_" + p.Prenom;
            p.Age = p.Age * -1;
            p.Genre = Genre.inconnu;
            return p;
        }
    }
}
```

La méthode getPersonne s'occupe de rajouter un préfix ws_ pour le nom et prénom, de rendre l'âge négatif et de mettre le genre à inconnu.

IService.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace WcfService1
{
    [ServiceContract]
    public interface IService1
    {
        [OperationContract]
        Personne getPersonne(Personne p);
    }

    [DataContract]
    public class Personne
    {
        string nom = "";
        string prenom = "";
        int age = 0;
        Genre genre ;

        [DataMember]
        public string Nom
        {
            get { return nom; }
            set { nom = value; }
        }
        [DataMember]
        public string Prenom
        {
            get { return prenom; }
            set { prenom = value; }
        }
        [DataMember]
        public int Age
        {
            get { return age; }
            set { age = value; }
        }
        [DataMember]
        public Genre Genre
        {
            get { return genre; }
            set { genre = value; }
        }
    }
    [DataContract]
    public enum Genre
    {
        homme,
        femme,
```

```
            inconnu
        }
}
```

Lorsque le web service tourne, une page web est ouverte dans le navigateur IExplorer.
Le WSDL est atteignable à l'adresse suivante :
http://localhost:1999/Service1.svc?wsdl

13.5.2. Client

Dans Visual Studio, créer un nouveau projet WPF Application.
Dans le projet, faire un Add Service Reference en utilisant l'url du WSDL du web service (http://localhost:1999/Service1.svc?wsdl), VS s'occupant d'utiliser le Svcutil pour créer le proxy et le fichier de configuration app.config.

Créer l'interface graphique. Simplement déposer les éléments XAML dans la fenêtre Window1.

Window1.xaml

```
<Window x:Class="WpfApplicationPersonne.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Window1" Height="300" Width="300">
    <Grid>
        <Label Height="28" Name="label1" VerticalAlignment="Top"
HorizontalAlignment="Left" Width="120">Nom</Label>
        <Label Height="28" Name="label3" VerticalAlignment="Top"
HorizontalAlignment="Right" Width="120">Prénom</Label>
        <Label Height="28" Margin="0,51,0,0" Name="label2"
VerticalAlignment="Top" HorizontalAlignment="Left" Width="120">Age</Label>
        <Label Height="28" HorizontalAlignment="Right" Margin="0,51,0,0"
Name="label4" VerticalAlignment="Top" Width="120">Genre</Label>
        <TextBox Height="23" Margin="0,28,0,0" Name="textBox_nom"
VerticalAlignment="Top" HorizontalAlignment="Left" Width="120" />
        <TextBox Height="23" HorizontalAlignment="Right" Margin="0,28,0,0"
Name="textBox_prenom" VerticalAlignment="Top" Width="120" />
        <TextBox Height="23" Margin="0,79,0,0" Name="textBox_age"
VerticalAlignment="Top" HorizontalAlignment="Left" Width="120" />
        <TextBox Height="23" HorizontalAlignment="Right" Margin="0,79,0,0"
Name="textBox_genre" VerticalAlignment="Top" Width="120" />
        <Label Height="28" HorizontalAlignment="Left" Margin="0,0,0,74"
Name="label5" VerticalAlignment="Bottom" Width="120">Nom WS</Label>
        <Label Height="28" HorizontalAlignment="Right" Margin="0,0,0,74"
Name="label7" VerticalAlignment="Bottom" Width="120">Prénom WS</Label>
        <Label Height="28" HorizontalAlignment="Left" Margin="0,0,0,23"
Name="label6" VerticalAlignment="Bottom" Width="120">Age WS</Label>
        <Label Height="28" HorizontalAlignment="Right" Margin="0,0,0,23"
Name="label8" VerticalAlignment="Bottom" Width="120">Genre WS</Label>
        <TextBox Height="23" HorizontalAlignment="Left" Margin="0,0,0,51"
Name="textBox_nomWS" VerticalAlignment="Bottom" Width="120" />
        <TextBox Height="23" HorizontalAlignment="Right" Margin="0,0,0,51"
Name="textBox_prenomWS" VerticalAlignment="Bottom" Width="120" />
        <TextBox Height="23" HorizontalAlignment="Left" Name="textBox_ageWS"
VerticalAlignment="Bottom" Width="120" />
        <TextBox Height="23" HorizontalAlignment="Right" Name="textBox_genreWS"
VerticalAlignment="Bottom" Width="120" />
        <Button Margin="96,126,107,113" Name="button1"
Click="button1_Click">Start</Button>
    </Grid>
</Window>
```

Window1.xaml.cs

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using WpfApplicationPersonne.ServiceReference1;

namespace WpfApplicationPersonne
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>
    public partial class Window1 : Window
    {
        public Window1()
        {
            InitializeComponent();
        }

        Service1Client client = new Service1Client();

        private void button1_Click(object sender, RoutedEventArgs e)
        {
            Personne p = new Personne();
            p.Nom = textBox_nom.Text;
            p.Prenom = textBox_prenom.Text;
            p.Age = int.Parse(textBox_age.Text);
            p.Genre = (Genre) Enum.Parse(typeof(Genre),textBox_genre.Text);
            Personne pWs = client.getPersonne(p);
            textBox_nomWS.Text = pWs.Nom;
            textBox_prenomWS.Text = pWs.Prenom;
            textBox_ageWS.Text = pWs.Age.ToString();
            textBox_genreWS.Text = pWs.Genre.ToString();
        }
    }
}
```

La référence ServiceReference1 est spécifiée dans la liste des assemblies et l'objet client est créé. L'action du bouton contient la récupération du contenu des différents TextBoxs pour remplir l'objet Personne puis la méthode getPersonne du client est appeléee et le retour est stocké dans un nouvel objet Personne dont les membres sont affichés dans les TextBoxs.

Résultat de l'utilisation du client :

13.6. Technologies utilisées

13.6.1. Web Services

Un web service implémente un moyen de communiquer, un moyen de décrire les services exposés et un nom ainsi qu'un répertoire qui permet de publier sur un serveur de catalogue des web services.

Communication : SOAP

Dans les web services, les interactions sont basées sur le SOAP ce qui leur permet d'échanger des messages avec une norme standardisée en XML et de lire les messages XML et de les traduire en action à effectuer.

Description des services : WSDL

Le WSDL permet de décrire les services de manière standardisée. En l'utilisant, cela permet de définir l'interface de programmation du web service qui est traduit en méthodes. Chacune de ces méthodes peut prendre un message en entrée et retourner un autre message comme réponse.

Le WSDL peut être compilé dans un langage de programmation pour générer les éléments intermédiaires et rendre de façon plus transparente l'utilisation du web service : c'est le proxy. Cela permet au développeur d'avoir tous les éléments en local et de pouvoir simplement appeler une méthode très simplement dans son code.

Publication des services : UDDI

Pour pouvoir informer le monde de l'existence du web service, il faut l'enregistrer dans un catalogue de manière standardisée grâce à l'UDDI et c'est à cette même place que les utilisateurs iront rechercher le web service qu'ils recherchent. A noter que ce n'est pas obligatoire.

13.6.1.1. SOAP

SOAP définit comment organiser les informations en utilisant le XML pour qu'elles puissent être échangées. Il utilise des messages pour échanger des informations qui sont utilisées comme enveloppe (constituée d'un header (optionnel) et d'un corps). C'est dans cette enveloppe que l'application s'occupe de mettre les informations devant être échangées entre le client et le service.

13.6.1.2. structure d'un document WSDL

Les spécifications du WSDL sont en général composées d'une partie concrète (bindings, services et ports) et d'une partie abstraite (types, messages, opérations, port types). On parle d'abstrait parce qu'il n'y a jamais un lien concret ou un encodage spécifique pour la construction ou une définition pour un service qui implémente un jeu de port types.

```
- <wsdl:definitions name="HelloWorldContractType" targetNamespace="http://tempuri.org/">
  + <wsdl:types></wsdl:types>
  + <wsdl:message name="HelloWorldContract_HelloWorld_InputMessage"></wsdl:message>
  + <wsdl:message name="HelloWorldContract_HelloWorld_OutputMessage"></wsdl:message>
  + <wsdl:message name="HelloWorldContract_MyHello_InputMessage"></wsdl:message>
  + <wsdl:message name="HelloWorldContract_MyHello_OutputMessage"></wsdl:message>
  + <wsdl:portType name="HelloWorldContract"></wsdl:portType>
  + <wsdl:binding name="BasicHttpBinding_HelloWorldContract" type="tns:HelloWorldContract"></wsdl:binding>
  + <wsdl:service name="HelloWorldContractType"></wsdl:service>
  </wsdl:definitions>
```

Pour mieux comprendre, prenons le cas d'un web service qui expose 2 fonctionnalités :
- string HelloWord(); qui ne prend pas de paramètres en entrée et renvoie un string
- Hello MyHello(int njoursSemaine); qui prend un integer comme paramètre et renvoie un objet Hello (l'objet est créé lors de l'appel de la méthode et un switch sélectionne le jours de la semaine à renvoyer).

13.6.2. Contenu du WSDL

13.6.2.1. definitions

C'est l'élément root pour tous les documents WSDL. Il définit le nom du service, déclare les namespaces qui seront utilisés dans tout le document et contient tous les éléments tels que portType, message, types, binding et service.

13.6.2.2. portType

Il décrit le web service, les opérations qui peuvent être réalisées et pour chacune des opérations, comment l'échange des messages doit se faire, le tout étant groupé dans le port type.
Il y a plusieurs types d'opérations :

- **One-way :** L'opération peut recevoir un message mais ne donne pas de réponse
- **Request-response :** L'opération peut recevoir une demande et va retourner une réponse
- **Solicit-response :** L'opération peut envoyer une demande et va attendre une réponse
- **Notification :** L'opération va envoyer un message mais n'attendra pas pour une réponse.

Chaque opération est répertoriée et pour chaque input / output, les actions et messages sont spécifiés (Pour savoir à quoi correspond par exemple tns:HelloWorldContract_HelloWorld_InputMessage, il faut se rendre dans la partie message du WSDL).

```
- <wsdl:portType name="HelloWorldContract">
  - <wsdl:operation name="HelloWorld">
      <wsdl:input wsaw:Action="http://tempuri.org/HelloWorldContract/HelloWorld"
      message="tns:HelloWorldContract_HelloWorld_InputMessage"/>
      <wsdl:output wsaw:Action="http://tempuri.org/HelloWorldContract/HelloWorldResponse"
      message="tns:HelloWorldContract_HelloWorld_OutputMessage"/>
    </wsdl:operation>
  - <wsdl:operation name="MyHello">
      <wsdl:input wsaw:Action="http://tempuri.org/HelloWorldContract/MyHello"
      message="tns:HelloWorldContract_MyHello_InputMessage"/>
      <wsdl:output wsaw:Action="http://tempuri.org/HelloWorldContract/MyHelloResponse"
      message="tns:HelloWorldContract_MyHello_OutputMessage"/>
    </wsdl:operation>
  </wsdl:portType>
```

Il peut également y avoir un élément optionnel wsdl:fault pour s'occuper de messages d'erreur qui surviendrait à l'exécution de l'opération.

13.6.2.3. types

Permet de définir un système de types pour que les données qui sont échangées soient correctement interprétées entre le client et le service. En général, c'est le

schéma XML qui est utilisé et qui permet à l'utilisateur de définir des données de type personnalisé tel que des structures.

La définition des types de données se trouve à l'adresse contenue dans schemaLocation.

- La 1ère adresse s'occupe de définir pour chaque élément défini dans les messages la séquence et le type de données (séquence parce qu'une méthode qui prendrait un string en premier puis un int n'a pas la même signature que celle qui prend le int en premier puis le string).
- La 2ème adresse s'occupe de la sérialisation des données.
- La 3ème adresse s'occupe de définir les types personnalisés. Ces types sont des classes créées qui portent dans le code .net la mention [DataContract] (et qui équivaut au nom de la classe) et les données membres qui dans le code .net ont la mention [DataMember].

```
- <wsdl:types>
  - <xsd:schema targetNamespace="http://tempuri.org/Imports">
      <xsd:import schemaLocation="http://localhost/helloworld?xsd=xsd0" namespace="http://tempuri.org/"/>
      <xsd:import schemaLocation="http://localhost/helloworld?xsd=xsd1"
      namespace="http://schemas.microsoft.com/2003/10/Serialization/"/>
      <xsd:import schemaLocation="http://localhost/helloworld?xsd=xsd2"
      namespace="http://schemas.datacontract.org/2004/07/HelloWorldExample"/>
    </xsd:schema>
  </wsdl:types>
```

Contenu xsd=xsd0 :

```
- <xs:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/">
    <xs:import schemaLocation="http://localhost/helloworld?xsd=xsd2"
    namespace="http://schemas.datacontract.org/2004/07/HelloWorldExample"/>
  - <xs:element name="HelloWorld">
    - <xs:complexType>
        <xs:sequence/>
      </xs:complexType>
    </xs:element>
  - <xs:element name="HelloWorldResponse">
    - <xs:complexType>
      - <xs:sequence>
          <xs:element minOccurs="0" name="HelloWorldResult" nillable="true" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  - <xs:element name="MyHello">
    - <xs:complexType>
      - <xs:sequence>
          <xs:element minOccurs="0" name="njoursSemaine" type="xs:int"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  - <xs:element name="MyHelloResponse">
    - <xs:complexType>
      - <xs:sequence>
          <xs:element minOccurs="0" name="MyHelloResult" nillable="true" type="q1:Hello"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
</xs:schema>
```

Contenu xsd=xsd2 :

```
- <xs:schema elementFormDefault="qualified"
  targetNamespace="http://schemas.datacontract.org/2004/07/HelloWorldExample">
   - <xs:complexType name="Hello">
      - <xs:sequence>
           <xs:element minOccurs="0" name="Jours" nillable="true" type="xs:string"/>
           <xs:element minOccurs="0" name="JoursSemaine" type="xs:int"/>
        </xs:sequence>
     </xs:complexType>
     <xs:element name="Hello" nillable="true" type="tns:Hello"/>
  </xs:schema>
```

13.6.2.4. message

La définition des messages se base sur ce qui a été défini dans le Type et permet de définir les données pour une opération. Chaque message peut contenir une ou plusieurs données. On peut comparer ces données aux paramètres d'une fonction.
Il fait le lien entre le nom utilisé pour le message dans le portType et l'élément auquel il correspond dans le schéma.
L'attribut name du message est unique par rapport à tous les autres messages définis.
L'attribut name du part est unique pour toutes les parties qui composent le message (il peut y avoir plusieurs parts constituant 1 message).

```
- <wsdl:message name="HelloWorldContract_HelloWorld_InputMessage">
     <wsdl:part name="parameters" element="tns:HelloWorld"/>
  </wsdl:message>
- <wsdl:message name="HelloWorldContract_HelloWorld_OutputMessage">
     <wsdl:part name="parameters" element="tns:HelloWorldResponse"/>
  </wsdl:message>
- <wsdl:message name="HelloWorldContract_MyHello_InputMessage">
     <wsdl:part name="parameters" element="tns:MyHello"/>
  </wsdl:message>
- <wsdl:message name="HelloWorldContract_MyHello_OutputMessage">
     <wsdl:part name="parameters" element="tns:MyHelloResponse"/>
  </wsdl:message>
```

13.6.2.5. binding

Le binding permet de spécifier l'encodage du message et le protocole de liaison pour toutes les opérations et messages définis pour un port type donné. Il y a 2 modes d'opérations possibles : RPC-style et document-style. Dans le RPC-style, les messages d'entrée et de sortie transporte les paramètres d'entrée et de sortie des appels aux procédures tandis que dans un document-style, les messages transportent des documents connus du client et du service.
C'est également dans cette partie que l'on spécifie que la communication des messages doit se faire en utilisant SOAP, quel protocole de transport doit être utilisé et quels sont les règles d'encodage utilisé pour la sérialisation les différentes parties du message.
On trouve 2 possibilités pour l'encodage : literal prend les types WSDL défini dans le schéma XML (que l'on utilise en général pour le document-style) tandis que le SOAP encoding prends les définitions abstraites du schéma XML et les traduit en XML en utilisant les règles d'encodage de SOAP (utilisé en général avec le RPC-style).

Soap:binding permet d'indiquer que la connexion est disponible par SOAP.
L'attribut transport indique quel protocole va être utilisé pour la circulation des messages, dans notre cas SOAP HTTP.

Soap:operation indique la connexion à une opération. L'attribut soapAction indique le type de header pour identifier le service (HTTP).
L'attribut de style indique de façon générale le format des messages SOAP. Dans notre cas, document spécifie que les demandes et les réponses seront des documents XML.

Soap:body indique comment les parties logiques d'un message apparaissent dans l'élément soap:body.
L'attribut use indique si les parties du message sont codées avec des règles d'encodage (comme valeur encoded et l'attribut encodingStyle doit être spécifié) ou s'il définit le schéma concret du message (dans ce cas, la valeur sera literal).

```
- <wsdl:binding name="BasicHttpBinding_HelloWorldContract" type="tns:HelloWorldContract">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
  - <wsdl:operation name="HelloWorld">
      <soap:operation soapAction="http://tempuri.org/HelloWorldContract/HelloWorld" style="document"/>
    - <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
    - <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  - <wsdl:operation name="MyHello">
      <soap:operation soapAction="http://tempuri.org/HelloWorldContract/MyHello" style="document"/>
    - <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
    - <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
```

13.6.2.6. Ports

Il fait le lien entre l'interface de liaison (binding) et l'adresse réseau d'où se trouve l'implémentation de ce type de port pouvant être accédé.

13.6.2.7. service

C'est un groupement logique de port. On peut par exemple trouver différents ports qui représentent différents types de connexion pour un même port type (une même fonctionnalité disponible au travers de différents protocoles de transport). Cet élément fourni l'adresse physique par laquelle on peut accéder au web service.

```
- <wsdl:service name="HelloWorldContractType">
  - <wsdl:port name="BasicHttpBinding_HelloWorldContract" binding="tns:BasicHttpBinding_HelloWorldContract">
      <soap:address location="http://localhost/helloworld"/>
    </wsdl:port>
  </wsdl:service>
```

13.6.3. XAML

Avec la version 2008 de Visual Studio, utiliser le XAML devient plus facile. Certaines actions ne nécessitent plus d'être codées à la main (par exemple, maintenant, un bouton posé dans la fenêtre .xaml peut être double cliqué pour arriver directement dans la fenêtre .xaml.cs avec la méthode déjà écrite).

13.6.3.1. Quelques Informations générales

Dans le code source d'un fichier XAML, on ne peut mettre que des objets de type UIElement (donc, pas possible d'écrire un hello world si ce n'est pas un objet, il faut passer par un Label par exemple). A noter que XAML est sensible à la casse.
Chaque élément déclaré commence par une balise d'ouverture dans laquelle on peut y mettre des propriétés et une balise fermante (si l'élément ne contient pas d'éléments enfants, la balise de fermeture peut être remplacée par /> dans la balise d'ouverture).
Comme en XML, l'ordre de fermeture des éléments doit correspondre au sens inverse de leur ouverture, la dernière balise ouverte étant fermée en premier :

```
<Balise1>
  <Balise2>
    <Balise 3 />
  </Balise2>
</Balise1>
```

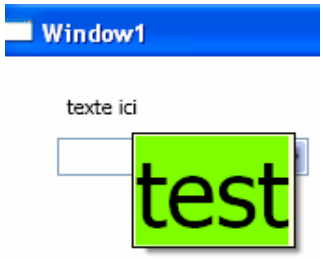
Les propriétés des éléments XAML sont spécifiées en utilisant dans la balise de l'élément un attribut (Height par exemple) avec sa valeur (23) :

```
<Label Height="23" Margin="30,16,88,0" Name="label1"
VerticalAlignment="Top">texte ici</Label>
```

Pour certaines propriétés des éléments, la syntaxe par attribut n'est pas possible, parce que la valeur ne peut pas être écrite comme un simple string. La solution est alors d'utiliser l'élément.propriété.

```
<Label Height="23" Margin="30,16,88,0" Name="label1"
    VerticalAlignment="Top">
  <Label.ToolTip>
    <AccessText FontSize="50" Background="Chartreuse"> test
    </AccessText>
  </Label.ToolTip> texte ici
</Label>
```

Dans la propriété ToolTip du label, un texte a été mis lors du survol de la souris sur le label avec la taille de la fonte et la couleur d'arrière plan ce qui donne le résultat suivant :



13.6.3.2. Markup Extension :

Par défaut, le moteur XAML interprète une valeur d'attribut comme un string littéral ou le converti en objet sur la base du type de l'attribut. Il se peut

toutefois que la valeur de l'attribut soit une référence d'un objet déjà présent (statique ou non). Pour ces cas-là, il faut informer le moteur que ce n'est pas quelque chose à traiter de façon habituelle en mettant des accolades dans la valeur de l'attribut ({ et }).
Dans cet exemple, lorsque le markup extension est traité, il retourne une référence d'un style qui a déjà été instancié.

```
<Window.Resources>
  <Style TargetType="Border" x:Key="PageBackground">
    <Setter Property="Background" Value="Green"/>
  </Style>
  <Style TargetType="Label" x:Key="LabelBackground">
    <Setter Property="Background" Value="Red"/>
  </Style>
</Window.Resources>
```

La fenêtre dans laquelle on se trouve est de type Window, c'est pourquoi on utilise le tag Window.Resources (dans le cas d'une page, ce serait Page.Resources).

```
<Label Height="23" Margin="30,16,88,0" Name="label1" VerticalAlignment="Top"
Style="{StaticResource LabelBackground}">
<Border Margin="20,89,0,123" Name="border1" Style="{StaticResource
PageBackground}" />
```

Lorsque l'on souhaite utiliser le style spécial défini, il suffit de mettre dans la valeur de l'attribut Style les accolades avec l'information qu'il s'agit d'une ressource statique et le nom de la ressource que l'on souhaite utiliser.

Les cas les plus courants sont :
StaticResource : Fourni une valeur pour une propriété utilisant une valeur d'une ressource déjà définie.
A noter qu'il faut que la déclaration de la ressource soit fait avant son utilisation dans le code xaml (bien que cela ne donne pas une erreur, cela réduit les performances). Si néanmoins il ne peut pas être fait autrement, il faut dans ce cas utiliser une référence dynamique.

DynamicResource : Fourni une valeur pour une propriété en utilisant une valeur provenant provenant d'une ressource lors de l'exécution (à chaque fois qu'il est utilisé, il faut accéder à la ressource pour savoir ce qu'il y a dedans).

Binding : permet de faire le lien entre la valeur et les données

13.6.3.3. Couleurs

Pour définir le background par exemple, jusqu'à présent, on a utilisé uniquement des couleurs disponibles avec des mots clés (Green, Red).

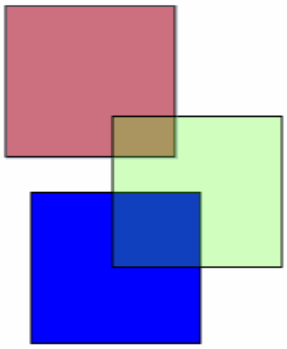
Il est également possible de définir d'autres couleurs en utilisant un codage à 8 hexadécimal car il y a également l'opacité (alpha) à spécifier.
FF = 100% de la couleur (ou opacité)
00 = 0% de la couleur (ou opacité)
Structure du codage :
Alpha Red Green Blue

Exemple :

Rouge normal	Rouge avec 0% d'alpha	Rouge avec 0% de rouge
<Setter Property="Background" Value="#FFFF0033"/>	<Setter Property="Background" Value="#00FF0033"/>	<Setter Property="Background" Value="#FF000033"/>

Il faut encore noter qu'il est possible de jouer sur les couleurs avec la superposition d'éléments dont l'opacité n'est pas de 100% (le point d'intersection donnant une nouvelle couleur) :

```
<Rectangle Margin="22,28,0,0" Name="rectangle1" Stroke="Black" Fill="#FFCC7080"
Height="98" Width="107" HorizontalAlignment="Left" VerticalAlignment="Top" />
<Rectangle Margin="38,0,133,16" Name="rectangle2" Stroke="Black" Fill="#FF0000FF"
Height="98" VerticalAlignment="Bottom" />
<Rectangle Margin="89,99,82,65" Name="rectangle3" Stroke="Black" Fill="#4045FF00"
/>
```



13.6.3.4. Evénements et Code-behind :

Dans les attributs de la balise root (window dans notre cas), on trouve l'attribut x:Class qui indique le fichier qui contient le code-behind pour ce fichier XAML.

```
<Window x:Class="WpfApplicationXaml.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Window1" Height="300" Width="300">
```

C'est dans ce fichier .xaml.cs que vont être traités entre autres les événements qui surviennent

13.6.3.5. Layout

Pour positionner ces éléments dans une fenêtre, il y a 2 moyens à disposition : relatif et absolu.

Relatif :
Les éléments comme le StackPanel, DockPanel ou le Grid fournissent un positionnement relatif. Tout le contenu n'est pas figé à sa place mais navigue lorsque l'on redimensionne la fenêtre.

```
<Grid>
    <StackPanel Name="stackPanel1" Margin="20,30,40,120">
        <Label Height="23" Name="label1" Width="120"
            VerticalAlignment="Top">Label</Label>
        <Button Height="23" Name="button1"
            Width="75">Button</Button>
        <TextBox Height="21" Name="textBox1" Width="120"
            Text="TextBox"/>
    </StackPanel>
</Grid>
```

Absolu :
Le positionnement absolu que l'élément Canvas utilise nécessite par contre de spécifier exactement la position de chaque élément par rapport à l'élément parent (en haut à gauche comme point de départ). Les éléments resteront à la même place même si la fenêtre est redimensionnée.

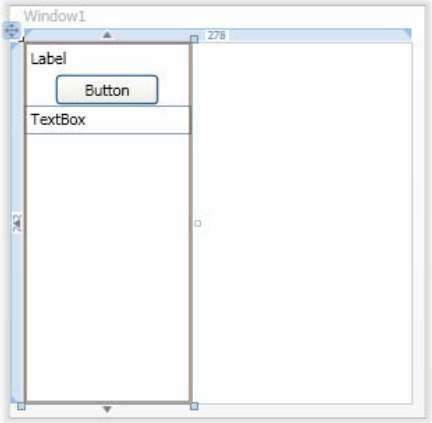
```
<Canvas>
    <Label Canvas.Left="0" Canvas.Top="0" Name="label1" >Left :
        0 ; Top : 0</Label>
</Canvas>
```

13.6.3.6. StackPanel

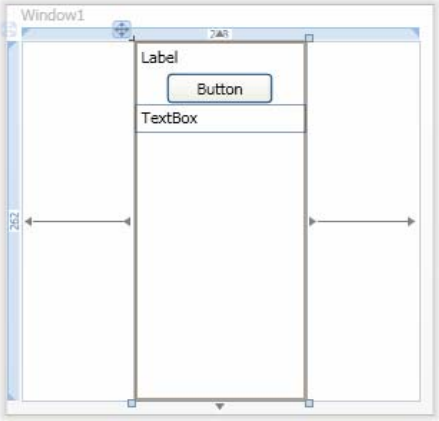
Les éléments sont disposés par défaut verticalement au milieu du StackPanel. L'ordre d'affichage est déterminé par l'ordre de déclaration dans le code XAML (1er déclaré = 1er affiché). Il est possible de changer l'alignement vertical et horizontal du StackPanel pour pas qu'il prenne toute la place (stretch) en modifiant les paramètres HorizontalAlignment et VerticalAlignment.
Dans l'exemple suivant, dans l'objet Window, le Grid contient le StackPanel dans lequel on change le layout :

HorizontalAlignment :

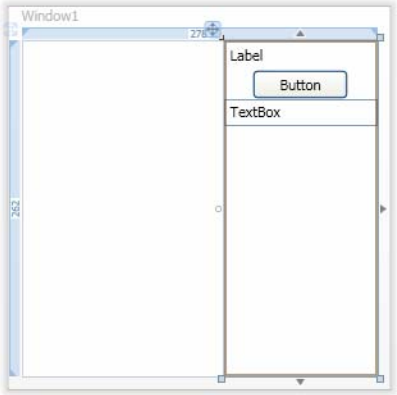
Left



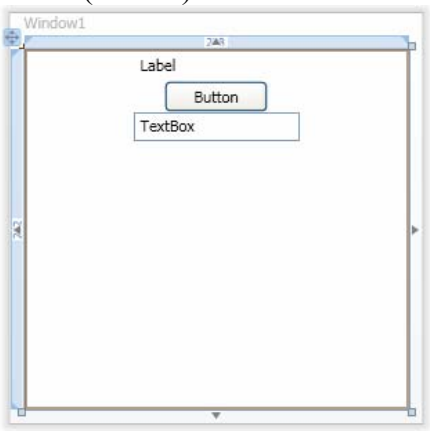
Center



Right

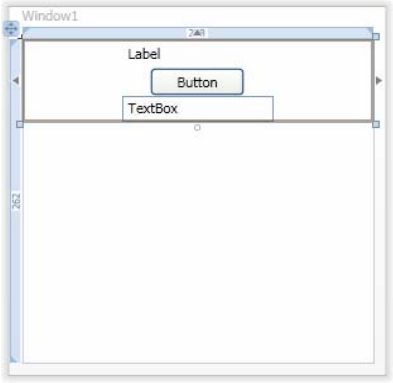


Stretch (default)

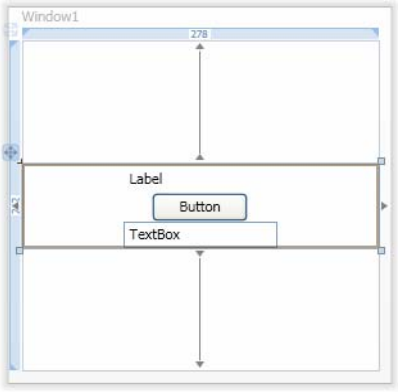


VerticalAlignment :

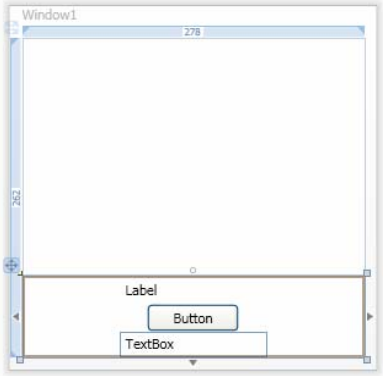
Top



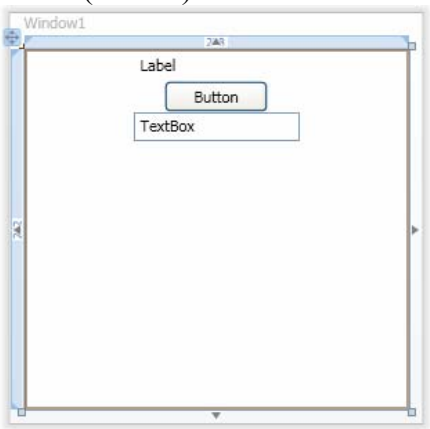
Center



Bottom



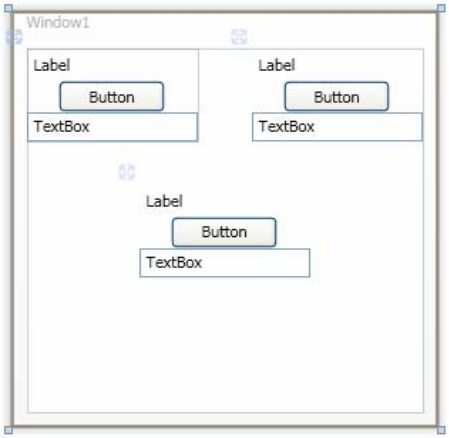
Stretch (default)

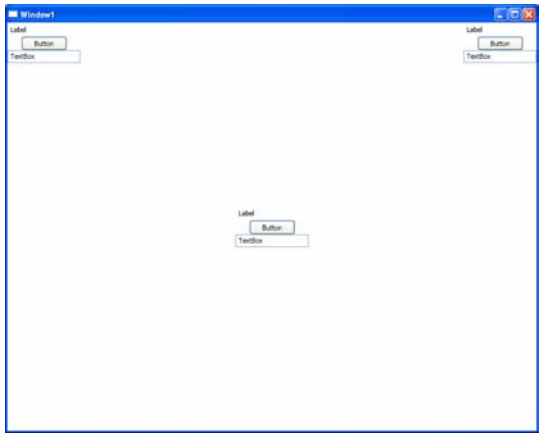


Le Grid peut également contenir plusieurs objets de type StackPanel comme le montre l'exemple suivant :

```
        <StackPanel Name="stackPanel1" Grid.ColumnSpan="1" Grid.RowSpan="1"
HorizontalAlignment="Left" VerticalAlignment="Top">
            <Label Height="23" Name="label1" Width="120"
VerticalAlignment="Top">Label</Label>
            <Button Height="23" Name="button1" Width="75">Button</Button>
            <TextBox Height="21" Name="textBox1" Width="120" Text="TextBox"/>
        </StackPanel>

        <StackPanel Name="stackPanel2" Grid.ColumnSpan="1" Grid.RowSpan="1"
HorizontalAlignment="right" VerticalAlignment="Top">
…
```

```
        </StackPanel>

        <StackPanel Name="stackPanel3" Grid.ColumnSpan="1" Grid.RowSpan="1"
HorizontalAlignment="Center" VerticalAlignment="Center">
…
        </StackPanel>
    </Grid>
```



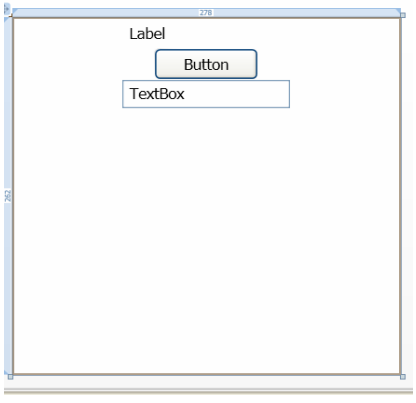


Lorsque l'on agrandit la fenêtre, les éléments restent collés à leur position.
Il est possible de disposer l'élément relativement par rapport à un autre élément ou par rapport à l'élément parent en utilisant la propriété Margin.
Le contenu de Margin fonctionne de la façon suivante :

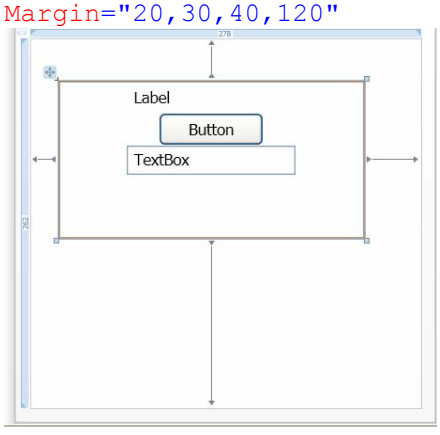
1. distance entre le bord gauche de l'élément parent et le bord gauche de l'élément contenant l'attribut Margin
2. distance entre le haut de l'élément parent et le haut de l'élément contenant l'attribut Margin
3. distance entre le bord droite de l'élément parent et le bord droite de l'élément contenant l'attribut Margin
4. distance entre le bas de l'élément parent et le bas de l'élément contenant l'attribut Margin

Exemple avec un StackPanel avec les propriétés suivantes de Margin :

`Margin="0,0,0,0"`



`Margin="20,30,40,120"`




```
<StackPanel Name="stackPanel1"
Margin="0,0,0,0">
    <Label Height="23"
Name="label1" Width="120"
VerticalAlignment="Top">
        Label</Label>
    <Button Height="23"
Name="button1"
Width="75">Button</Button>
```

```
<StackPanel Name="stackPanel1"
Margin="20,30,40,120">
    <Label Height="23"
Name="label1" Width="120"
VerticalAlignment="Top">
        Label</Label>
    <Button Height="23"
Name="button1"
Width="75">Button</Button>
```

```
    <TextBox Height="21"
Name="textBox1" Width="120"
Text="TextBox"/>
</StackPanel>
```

```
    <TextBox Height="21"
Name="textBox1" Width="120"
Text="TextBox"/>
</StackPanel>
```

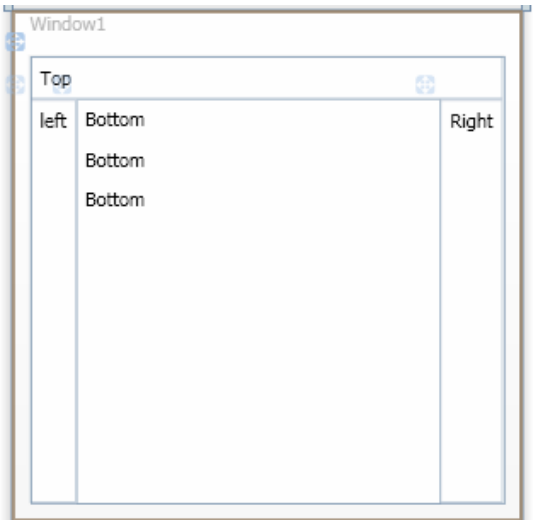
Lorsqu'on modifie la taille de la fenêtre lors de l'exécution, le contenu se déplace toujours mais prend en compte les informations de la propriété Margin.

13.6.3.7. DockPanel

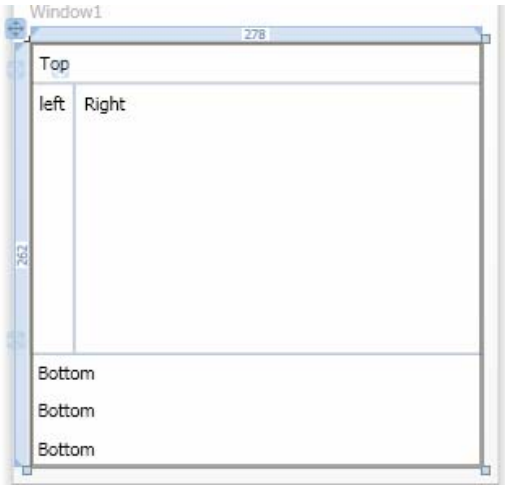
Le DockPanel permet de positionner les éléments enfants horizontalement ou verticalement. Les éléments enfants utilisent la propriété DockPanel.Dock pour se positionner à droite, gauche, haut ou bas. A noter que la déclaration des éléments dans le fichier XAML influence l'affichage :

```
<Grid>
    <DockPanel>
        <Border Name="border1" DockPanel.Dock="Top" >
            <Label Name="label1" >Top</Label>
        </Border>
        <Border Name="border2" DockPanel.Dock="Left" >
            <Label Name="label2" >left</Label>
        </Border>
        <Border Name="border3" DockPanel.Dock="Right" >
            <Label Name="label3" >Right</Label>
        </Border>
        <StackPanel Name="border4" DockPanel.Dock="Bottom" >
            <Label Name="label4" >Bottom</Label>
            <Label Name="label5" >Bottom</Label>
            <Label Name="label6" >Bottom</Label>
        </StackPanel>
    </DockPanel>
</Grid>
```

Déclaration : Top, Left, Right, Bottom



Déclaration : Top, Bottom, Left, Right



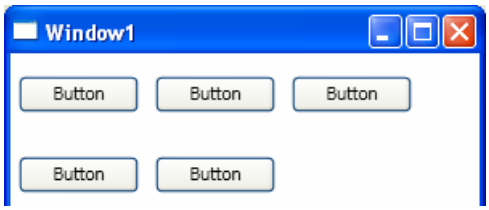
Tout comme le StackPanel, on peut spécifier un alignement vertical ou horizontal.

13.6.3.8. WrapPanel

Le WrapPanel permet de poser des éléments dans un ordre de gauche à droite par défaut. Si la largeur n'est pas suffisante, les éléments restants sont mis dans une nouvelle ligne.

Les propriétés ItemHeight et ItemWidth permettent de gérer l'espacement des différents éléments. Si la taille de l'élément est plus grande que celle définies dans le WrapPanel, les éléments seront coupés par les éléments suivants.

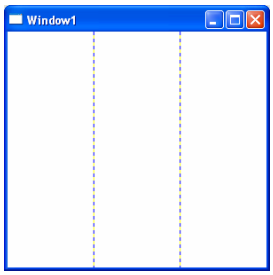
```
<WrapPanel Name="wrapPanel1" FlowDirection="LeftToRight" ItemHeight="50"
ItemWidth="85">
    <Button Height="23" Name="button1" Width="75">Button</Button>
    <Button Height="23" Name="button2" Width="75">Button</Button>
    <Button Height="23" Name="button3" Width="75">Button</Button>
    <Button Height="23" Name="button4" Width="75">Button</Button>
    <Button Height="23" Name="button5" Width="75">Button</Button>
</WrapPanel>
```



13.6.3.9. Grid

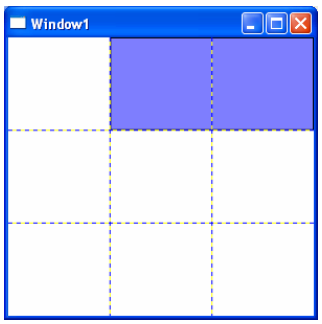
Avec le Grid, il est possible de créer un tableau dans la fenêtre en utilisant les propriétés ColumnDefinitions et RowDefinitions :

```
<Grid ShowGridLines="true">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Name="col1" />
        <ColumnDefinition Name="col3" />
        <ColumnDefinition Name="col2" />
    </Grid.ColumnDefinitions>
</Grid>
```



Il n'est pas possible de fusionner des cellules comme on pourrait le faire en HTML avec l'attribut rowspan ou colspan mais on peut placer des éléments par-dessus qui occupent plusieurs cellules :

```
<Grid ShowGridLines="true">
    <Grid.RowDefinitions>
        <RowDefinition Name="row1" />
        <RowDefinition Name="row2" />
        <RowDefinition Name="row3" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Name="col1" />
        <ColumnDefinition Name="col3" />
        <ColumnDefinition Name="col2" />
</Grid.ColumnDefinitions>
    <Rectangle Grid.Column="1" Grid.ColumnSpan="2" Margin="0,0,0,0"
Name="rectangle1" Stroke="Black" Fill="#800000FF" />
</Grid>
```



Dans l'élément Rectangle, Grid.Column indique la colonne dans laquelle commence l'élément Rectangle. Grid.ColumnSpan indique combien de colonnes sont utilisées par l'élément Rectangle. Si on avait voulu avoir toutes les colonnes de la 1ère ligne occupée par l'élément Rectangle, Grid.Column aurait dû prendre comme valeur 0 (pour la 1ère colonne) et Grid.ColumnSpan 3 (vu que l'on a 3 colonnes).
En ayant mis 0 dans la propriété Margin, cela permet à l'élément Rectangle d'occuper toute la place de la cellule.

Width et Height
Il est toujours possible de mettre la taille en pixel de la largeur d'une colonne ou d'une ligne (ces informations étant mises dans la définition (`Width="10"`)). Lorsque la fenêtre est agrandie ou rétrécie, la ligne ou colonne ne change pas de taille.
Si l'on souhaite ne pas mettre une taille fixe mais un ratio de taille, il faut utiliser le système de fractionnement décimal (terminé par un astérisque). Ce système se base sur la taille de la ligne ou de la colonne et non pas sur le total de l'espace disponible :

`Width="1.0*"` : occupe 100% de sa taille
`Width="2.0*"` : occupe 200% de sa taille
`Width="0.5*"` : occupe 50% de sa taille

Dans une fenêtre Windows contenant 9 cellules dont la largeur est de 93 pixels (dans le volet d'édition par défaut de la fenêtre), cela donne avec les différents Width pour la colonne du milieu les tailles suivantes :

`Width="1.0*"` : 93 , 93, 93
`Width="2.0*"` : 70, 139, 70
`Width="0.5*"` : 111, 56, 111

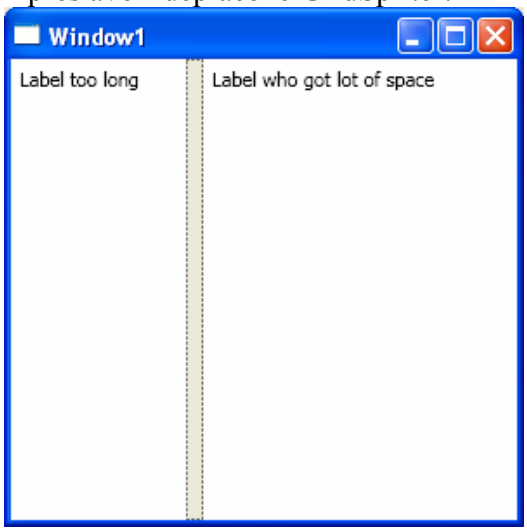
GridSplitter
Il peut être intéressant d'offrir à l'utilisateur la possibilité de redimensionner la largeur d'une colonne et c'est là que le GridSplitter intervient. Il met en place une zone entre 2 colonnes avec laquelle l'utilisateur peut interagir. On peut l'utiliser par exemple pour un menu qui est sur la gauche et les informations sur la droite. Si l'arborescence du menu dépasse la largeur de la colonne, le texte ne sera pas affiché mais l'utilisateur pourra agrandir la zone pour tout afficher (un exemple parlant avec le site MSDN : le menu à gauche pouvant prendre beaucoup de place en largeur, on peut déplacer la colonne pour afficher plus de texte).

On peut mettre le GridSplitter directement dans la colonne qui suit celle que l'on souhaite rendre redimensionnable mais le contenu de la colonne dans laquelle il se trouve peut se superposer. Pour éviter ceci, soit il faut ajouter une colonne de la même largeur que le GridSplitter entre les 2 colonnes et d'y mettre le GridSplitter ou dans la colonne qui contient le GridSplitter, mettre un Border avec un espace entre le début de la colonne et le début de l'élément.

```
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="50" />
        <ColumnDefinition Width="10"/>
        <ColumnDefinition Width="228" />
    </Grid.ColumnDefinitions>
    <GridSplitter Name="GridSplitter1" Grid.Column="1" Margin="0,0,0,0"
        HorizontalAlignment="Left" Width="10" />
    <Label Height="23" Margin="0,0,0,0" Name="label1" VerticalAlignment="Top">Label
too long</Label>
    <Label Grid.Column="2" Height="23" Margin="0,0,0,0" Name="label2"
VerticalAlignment="Top">Label who got lot of space</Label>
</Grid>
```

Etat Initiale de la fenêtre :

Après avoir déplacé le GridSpliter:

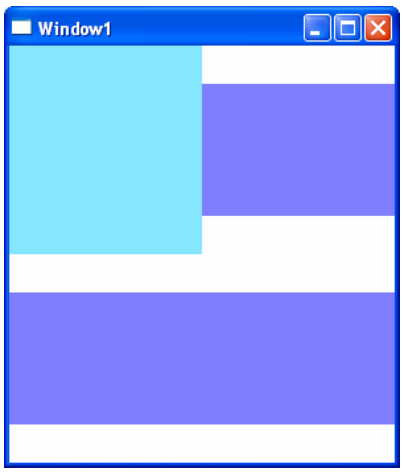


13.6.3.10. UniformGrid

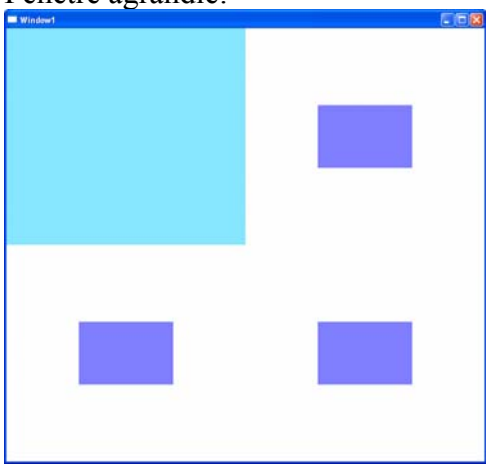
Cet élément fait penser à un Grid mais présente l'avantage d'être plus simple à utiliser. En effet, pas besoin de définir les lignes et colonnes, chaque élément placé dans ce contrôle équivaut à une cellule.
L'inconvénient de cette méthode tient dans le fait que la taille des cellules est uniforme (d'où le nom du contrôle) et que le layout des cellules est prédéfinie sans possibilité de changement. Si les éléments débordent, ils ne seront pas affichés entièrement (sauf si la taille de la fenêtre est agrandie).

```
<UniformGrid>
    <Border Height="500" Name="border1" Width="500" Background="#800DD0FF"></Border>
    <Border Height="100" Name="border2" Width="150" Background="#800000FF" />
    <Border Height="100" Name="border3" Width="150" Background="#800000FF" />
    <Border Height="100" Name="border4" Width="150" Background="#800000FF" />
</UniformGrid>
```

Etat Initiale de la fenêtre :



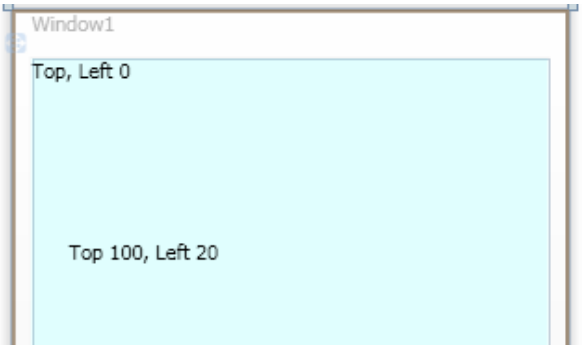
Fenêtre agrandie:



13.6.3.11. Canvas

Lorsque l'on pose un Canvas dans une fenêtre, si on ne spécifie pas les attributs Height et Width, il prendra toute la place dans la fenêtre. Si l'on spécifie la même taille que la fenêtre, le Canvas débordera sur la droite et en bas de la fenêtre.
Le positionnement d'éléments dans un Canvas se fait de manière absolue. Pour positionner l'élément, il faut spécifier les attributs Top et Left, 0,0 étant le coin en haut à gauche de la fenêtre.
Dans le cas où ce n'est pas des Canvas à positionner (par exemple des rectangles), l'emplacement est fait en utilisant les attributs Canvas.Top et Canvas.Left.
Si l'un des attributs n'est pas spécifié, il prend par défaut la valeur de 0.

```
<Canvas Background="LightCyan">
    <TextBlock Canvas.Top="0" Canvas.Left="0" Text="Top, Left 0"/>
    <TextBlock Canvas.Top="100" Canvas.Left="20" Text="Top 100, Left 20" />
</Canvas>
```

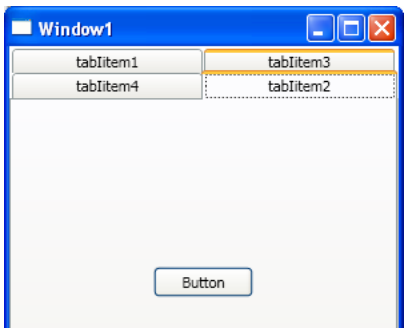


13.6.3.12. TabControl

Ce contrôle permet de gérer les éléments affichés à l'écran par le biais d'onglet. Un tabControl contient des TabItem. L'attribut Header du TabItem sert à afficher le texte sur l'onglet. A l'intérieur, on peut soit se contenter d'utiliser un TabItem.Content pour afficher du texte ou un contrôle. S'il faut afficher plus de contenu, il faudra se tourner vers un autre Layout (comme le Canvas par exemple).

```
<TabControl  Margin="0, 0, 0,0">
    <TabItem Name="tabIitem1" Header="tabIitem1">
```

```
        <StackPanel>
            <Label >texte ici</Label>
        </StackPanel>
    </TabItem>
    <TabItem Name="tabIitem3" Header="tabIitem3">
        <TabItem.Content>information 3</TabItem.Content>
    </TabItem>
    <TabItem Name="tabIitem4" Header="tabIitem4">
        <TabItem.Content>information 4</TabItem.Content>
    </TabItem>
    <TabItem Name="tabIitem2" Header="tabIitem2">
        <TabItem.Content><Button Height="23" Name="button1"
Width="75">Button</Button>
        </TabItem.Content>
    </TabItem>
</TabControl>
```



13.6.3.13. Saisie utilisateur

Pour saisir du texte, 3 éléments sont à disposition : le TextBox, le RichTextBox et le PasswordBox

TextBox

Le TextBox est l'élément utilisé dans la majorité des cas pour saisir du texte simple.

RichTextBox

Cet élément est plus évolué que le TextBox. Il permet entre autres de pouvoir formater le texte (mettre en gras un élément du texte avec le raccourci Ctr+B), d'y faire des paragraphes, inclure des éléments tels qu'un bouton ou une image. On parle dans ce cas de texte riche (rich cotnent).

PasswordBox

Comme on pourrait s'en douter, le PasswordBox permet de faire de la saisie masquée.

```
<StackPanel>
    <Label Height="23" Name="label1" Width="120">TextBox</Label>
    <TextBox Height="21" Name="textBox1" Width="120"  />
    <Label Height="23" Name="label2" Width="120">RichTextBox</Label>
    <RichTextBox Height="200" Name="richTextBox1" Width="200">
        <FlowDocument >
            <Paragraph>
                <Hyperlink>google</Hyperlink> This is flow content and you can
                <Bold>edit me!</Bold>
                <Button Height="23" Name="button1" Width="75">Button</Button>
            </Paragraph>
            <Paragraph>
                <Image Height="150" Name="image1" Width="200" Source="Collines.jpg"
/>
            </Paragraph>
        </FlowDocument>
    </RichTextBox>
    <Label Height="23" Name="label3" Width="120">PasswordBox</Label>
```

```
    <PasswordBox Height="21" Name="passwordBox1" Width="120" />
</StackPanel>
```

13.6.4. Affichage de texte

13.6.4.1. Label

Pour n'afficher qu'une ligne, le meilleur élément est le Label.
A noter une chose étrange avec l'affichage lorsque l'on utilise l'attribut FlowDirection dans VS 2008 beta 2, lorsque la propriété est mise en RightToLeft (le texte commence sur la droite) et que l'on repasse en LeftToRight, le designer ne fait pas le changement (par contre, une fois compilé, le texte est à la bonne place).

13.6.4.2. TextBlock

Comme son nom l'indique, il permet de mettre des blocs de texte. Pour éviter que le texte ne soit rogné, il faut préciser 2 attributs :

- TextWrapping avec comme propriété WrapWithOverflow qui va mettre à la ligne le reste du texte s'il dépasse la largeur de l'élément.
- Ne pas spécifier de hauteur (Height) pour que l'élément prenne la hauteur nécessaire pour afficher tout le texte.

Il est également possible de faire des sauts de lignes en utilisant l'élément LineBreak dans le TextBlock. Il n'est pas possible de le mettre dans la propriété de l'attribut Text (erreur de parsing), par contre, il peut être mis après avoir écrit la balise d'ouverture de l'élément TextBox (un élément LineBreak ne peut être suivi par un autre élément LineBreak).
Pour mettre le texte qui suit, il suffit d'utiliser l'élément Run et d'y spécifier le texte entre la balise d'ouverture et de fermeture.

```
<StackPanel>
    <TextBlock Name="textBlock1" Width="120" Text="Ceci est un texte"
TextWrapping="WrapWithOverflow">
        <LineBreak/>
        <Run>qui est placé dans un TextBlock. On peut même faire plusieurs lignes de
texte dedans</Run>
        <LineBreak/>
        <Run>On peut même faire plusieurs lignes de texte dedans</Run>
    </TextBlock>
</StackPanel>
```

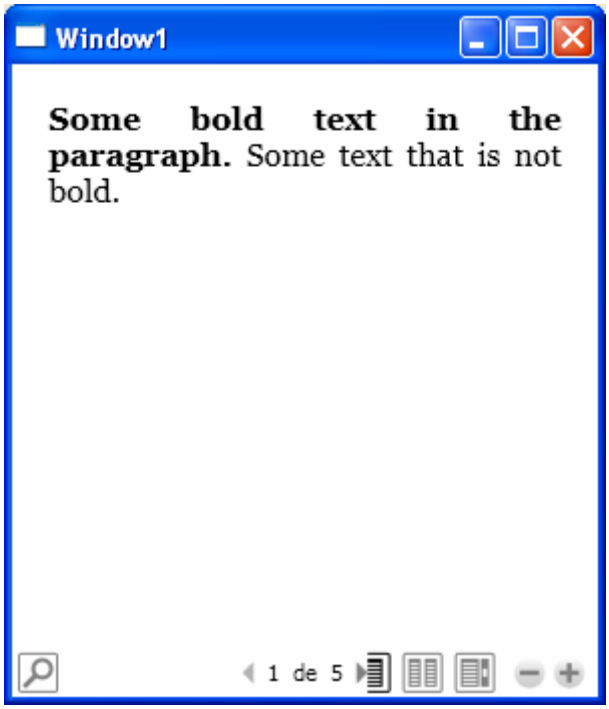
13.6.4.3. Flow Document

Lorsqu'il y a beaucoup de texte à afficher, les contrôles présentés ci-dessus risquent de ne pas être suffisants. C'est là qu'intervient le Flow Document.
Le Flow Document permet de reformater le contenu par rapport à la taille d'une fenêtre, de la résolution du matériel sur lequel tourne l'application ainsi que d'autres environnements. De plus, il embarque des fonctionnalités prédéfinies incluant la recherche, des modes d'affichage pour améliorer la lecture et la possibilité de changer la taille et la fonte du texte.

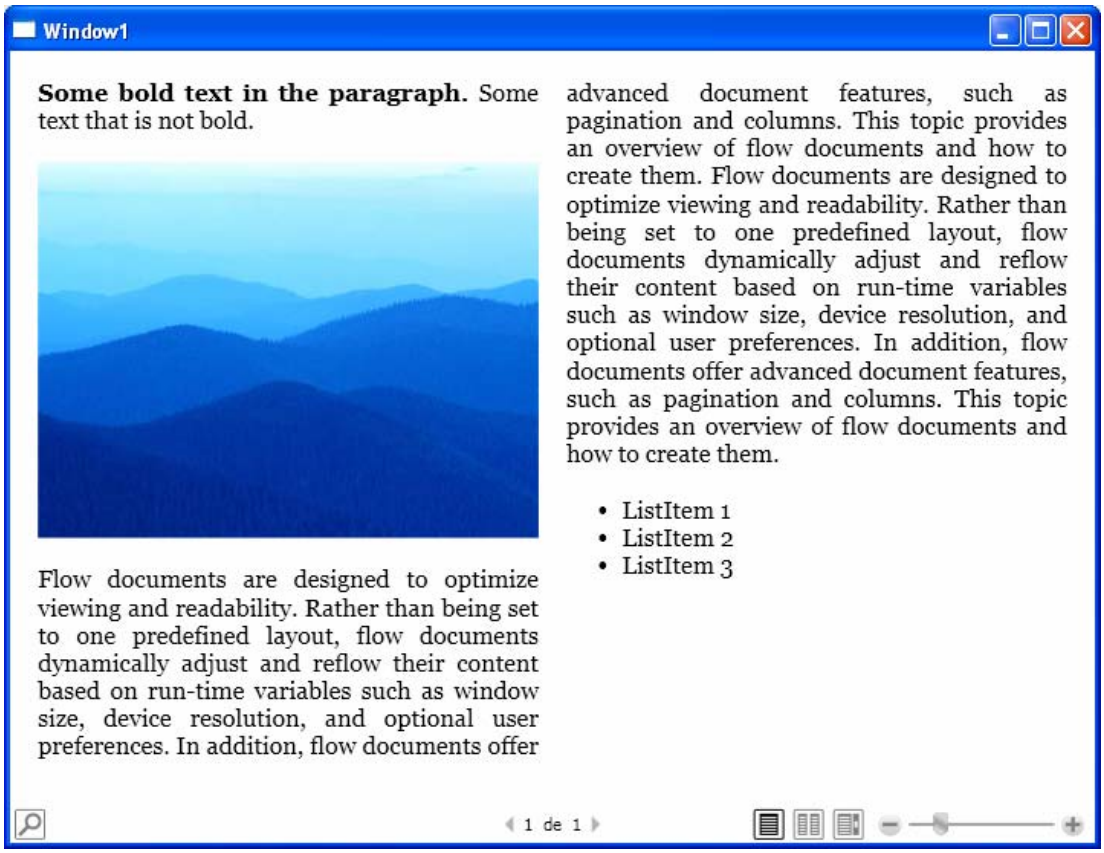
Voici un exemple qui prend pour base un exemple (ainsi que le texte) provenant du site MSDN (http://msdn2.microsoft.com/fr-fr/library/aa970909(VS.90).aspx) :

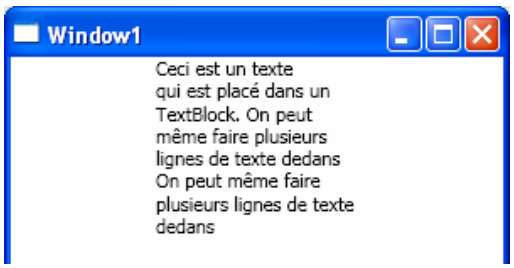
```
<FlowDocumentReader
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <FlowDocument>
        <Paragraph>
            <Bold>Some bold text in the paragraph.</Bold>
            Some text that is not bold.
        </Paragraph>
        <BlockUIContainer>
            <Image Source="Collines.jpg" />
        </BlockUIContainer>
        <Paragraph>
            Flow documents are designed to optimize viewing and readability. Rather
than being set to one predefined layout, flow documents dynamically adjust and
reflow their content based on run-time variables such as window size, device
resolution, and optional user preferences. In addition, flow documents offer
advanced document features, such as pagination and columns. This topic provides an
overview of flow documents and how to create them.
            Flow documents are designed to optimize viewing and readability. Rather
than being set to one predefined layout, flow documents dynamically adjust and
reflow their content based on run-time variables such as window size, device
resolution, and optional user preferences. In addition, flow documents offer
advanced document features, such as pagination and columns. This topic provides an
overview of flow documents and how to create them.
        </Paragraph>
        <List>
            <ListItem>
                <Paragraph>ListItem 1</Paragraph>
            </ListItem>
            <ListItem>
                <Paragraph>ListItem 2</Paragraph>
            </ListItem>
            <ListItem>
                <Paragraph>ListItem 3</Paragraph>
            </ListItem>
        </List>
    </FlowDocument>
</FlowDocumentReader>
```

Lors de l'exécution de l'application, on obtient l'affichage par défaut suivant :



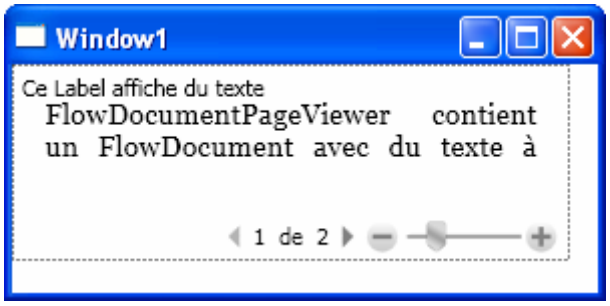
On trouve le 1er paragraphe du texte et en bas de la fenêtre, les différentes fonctionnalités par défaut du FlowDocumentReader.
En agrandissant la fenêtre, l'image est affichée ainsi que tout le texte défini :

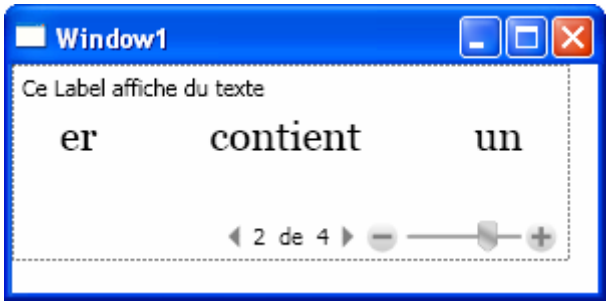




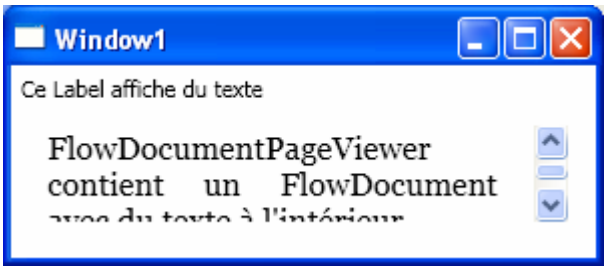
Il est également possible d'afficher dans un canvas (ou tout autre élément de layout) du texte avec le FlowDocument en utilisant un conteneur.

- **FlowDocumentPageViewer** pour un affichage de type page. Il affiche en bas un slider pour augmenter ou diminuer la taille du texte ainsi qu'une petite navigation permettant de passer d'une « page » à l'autre.





- **FlowDocumentScrollViewer** pour un affichage avec une scrollbar (exemple : un contrat de licence).



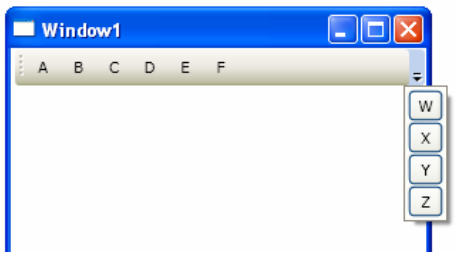
13.6.5. ToolBar

13.6.5.1. ToolBar, ToolBarPanel et ToolBarTray

En général, ces éléments s'emboîtent de la façon suivante : un ToolBarTray contient un ou plusieurs Toolbars qui peuvent contenir un ToolBarPanel.
Le ToolBar est un conteneur pour stocker des commandes ou des contrôles (la barre d'outils dans office par exemple).
Si la taille est plus petite que la taille totale nécessaire pour afficher tous les contrôles, les contrôles restants peuvent être appelés en cliquant sur la flèche à la fin du contrôle.

Il est également possible mettre des contrôles spécifiques pour le ToolBar dans un ToolBarPanel qui seront affichés après avoir cliqué sur la flèche à la fin du contrôle ToolBar. Il faut également préciser l'attribut ToolBar.OverflowMode avec comme propriété Always pour que les contrôles n'apparaissent pas s'il y a assez de place dans la fenêtre pour tout afficher.
Dans le ToolBarPanel, l'attribut Orientation permet d'afficher la série de contrôle masqué de façon horizontal ou vertical.

```
<StackPanel>
        <ToolBar Band="1" BandIndex="1">
        <Button Height="23" Name="button1" Width="25">A</Button>
        <Button Height="23" Name="button2" Width="25">B</Button>
        <Button Height="23" Name="button3" Width="25">C</Button>
        <Button Height="23" Name="button4" Width="25">D</Button>
        <Button Height="23" Name="button5" Width="25">E</Button>
        <Button Height="23" Name="button6" Width="25">F</Button>
        <ToolBarPanel Orientation="Vertical" ToolBar.OverflowMode="Always">
            <Button Height="23" Name="button7" Width="25">W</Button>
            <Button Height="23" Name="button8" Width="25">X</Button>
            <Button Height="23" Name="button9" Width="25">Y</Button>
            <Button Height="23" Name="button10" Width="25">Z</Button>
        </ToolBarPanel>
    </ToolBar>
</StackPanel>
```

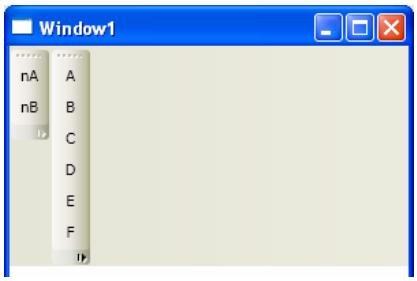


Quelques attributs intéressants dans l'élément ToolBar sont à signaler :
BandIndex qui permet de spécifier l'ordre d'affichage de plusieurs ToolBar dans un ToolbarTray (celui qui est le plus petit s'affichant en premier).
Band permet de choisir sur quelle ligne afficher le ToolBar. 1 pour la 1ère ligne, 2 pour la 2ème ligne et ainsi de suite.

Un autre exemple affichant le contenu d'un ToolBarTray verticalement contenant 2 ToolBars :

```
<StackPanel>
    <ToolBarTray Orientation="Vertical">
        <ToolBar Band="2" BandIndex="2">
            <Button Height="23" Name="button1" Width="25">A</Button>
            <Button Height="23" Name="button2" Width="25">B</Button>
            <Button Height="23" Name="button3" Width="25">C</Button>
            <Button Height="23" Name="button4" Width="25">D</Button>
            <Button Height="23" Name="button5" Width="25">E</Button>
            <Button Height="23" Name="button6" Width="25">F</Button>
            <ToolBarPanel Orientation="Vertical" ToolBar.OverflowMode="Always">
                <Button Height="23" Name="button7" Width="25">W</Button>
                <Button Height="23" Name="button8" Width="25">X</Button>
                <Button Height="23" Name="button9" Width="25">Y</Button>
                <Button Height="23" Name="button10" Width="25">Z</Button>
            </ToolBarPanel>
        </ToolBar>
        <ToolBar Band="1" BandIndex="1">
            <Button Height="23" Name="buttona" Width="25">nA</Button>
            <Button Height="23" Name="buttonb" Width="25">nB</Button>
```

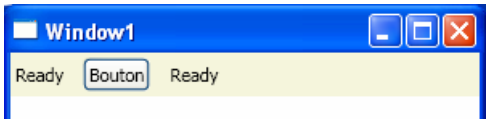
```
        </ToolBar>
    </ToolBarTray>
</StackPanel>
```



13.6.5.2. Statusbar

Pour afficher des informations telles que du texte, des images ou des contrôles en bas de fenêtre (il est également possible de spécifier un autre emplacement grâce à l'attribut VerticalAlignment), l'élément StatusBar est l'idéal.
Il est constitué de StatuBarItem dans lesquels on place les informations à afficher.
Un exemple avec la barre de status placée volontairement en haut de la fenêtre :

```
<StatusBar Name="sbar"
       VerticalAlignment="Top" Background="Beige" >
    <StatusBarItem>
        <TextBlock>Ready</TextBlock>
    </StatusBarItem>
    <StatusBarItem>
        <Separator/>
    </StatusBarItem>
    <StatusBarItem>
        <Button >Bouton</Button>
    </StatusBarItem>
    <StatusBarItem>
        <Separator/>
    </StatusBarItem>
    <StatusBarItem>
        <TextBlock>Ready</TextBlock>
    </StatusBarItem>
</StatusBar>
```

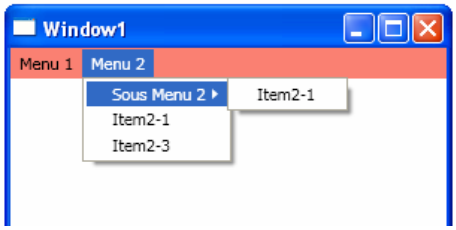


13.6.5.3. Menus

Grâce à l'élément Menu, il est possible de créer ses propres menus contenant des MenuItem ainsi que la possibilité pour ces MenuItem de contenir d'autres éléments.

```
<StackPanel>
    <Menu Background="Salmon">
        <MenuItem Header="Menu 1" >
            <MenuItem Header="Item1" Name="menuItem1" />
            <MenuItem Header="Item2" Name="menuItem2"  />
            <MenuItem Header="Item3" Name="menuItem3"/>
        </MenuItem>
        <MenuItem Header="Menu 2" >
            <MenuItem Header="Sous Menu 2" >
                <MenuItem Header="Item2-1" Name="menuItem4" />
            </MenuItem>
            <MenuItem Header="Item2-1" Name="menuItem5"  />
```

```
            <MenuItem Header="Item2-3" Name="menuItem6"/>
        </MenuItem>
    </Menu>
</StackPanel>
```



13.6.6. Images, Border, GroupBox

13.6.6.1. Image

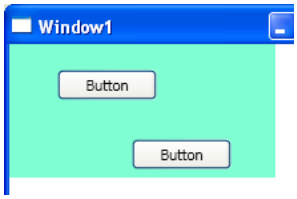
Pour ajouter une image dans un projet, il faut avoir au préalable avoir mis l'image dans le dossier contenant le projet. Il suffit ensuite dans la fenêtre Solution Explorer faire un click droit sur le projet et choisir Add -> Existing Item. Dans la nouvelle fenêtre, mettre le type d'objets à All Files (*.*) et sélectionner l'image.
Il n'y a plus qu'à poser un élément Image dans la fenêtre de design et dans l'attribut Source dans la fenêtre de propriété sélectionner l'image à appliquer.

```
<WrapPanel Name="wrapPanel1">
    <Image Name="image1" Source="/layout_StackPanel_Wpf;component/Collines.jpg" />
</WrapPanel>
```

13.6.6.2. Border

L'élément Border permet de mettre une bordure autour d'un autre contrôle, par exemple un bouton. Ce bouton est mis entre les balises d'ouverture et de fermeture de l'élément Border. Il ne peut contenir qu'un enfant (donc, pas possible de mettre 2 boutons dans un Border). Pour contourner cette limitation, il suffit de mettre un Layout dans le Border (et ce Layout pourra contenir autant d'enfants que nécessaire).

```
<WrapPanel Name="wrapPanel1">
    <Border Height="100" Name="border1" Width="200" Background="Aquamarine">
        <Canvas Height="100" Name="canvas1" Width="200" >
            <Button Canvas.Left="36" Canvas.Top="19" Height="23" Name="button1"
Width="75">Button</Button>
            <Button Canvas.Left="92" Canvas.Top="71" Height="23" Name="button2"
Width="75">Button</Button>
        </Canvas>
    </Border>
</WrapPanel>
```



13.6.6.3. GroupBox

Le GroupBox est un conteneur qui possède une bordure et un header.
Le header est défini en utilisant `<GroupBox.Header>` et on y place un label par exemple.

```
<StackPanel>
    <GroupBox Width="150" Height="52">
        <GroupBox.Header>
            <Label>Boutons</Label>
        </GroupBox.Header>
        <Button>bouton</Button>
    </GroupBox>

    <GroupBox Width="150" Height="101">
        <GroupBox.Header>
            <Label>Label</Label>
        </GroupBox.Header>
        <Label>label</Label>
    </GroupBox>
</StackPanel>
```

13.6.7. Button, Checkbox, ListBox & RadioButton

13.6.7.1. Button

Le bouton permet à l'utilisateur de déclencher l'exécution du code déterminé dans l'attribut Click. 2 attributs intéressants de citer :

- Cursor permet de définir comment doit s'afficher le curseur de la souris lors d'un survol du bouton.
- ClickMode permet de déterminer quand l'action du bouton doit être déclenchée : lors du click, lors de relâchement du click ou lors du survol de la souris.

Alors que dans la version de Visual Studio 2005 avec le patch pour la prise en charge du framework 3.0 la création de l'événement du click sur un bouton devait se coder totalement manuellement, la version 2008 de VS permet de créer l'événement du click en double cliquant sur le contrôle dans la fenêtre de design et affiche la nouvelle méthode qui va être exécutée au click sur le bouton. Le nom de la méthode doit correspondre à celle définie dans l'attribut Click dans le fichier XAML.

Exemple de l'action d'un bouton dans le code-behind :

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    this.Close();
}
```

Cette action va provoquer la fermeture de la fenêtre.

13.6.7.2. CheckBox

Le contrôle CheckBox dispose de 2 états : non sélectionné et sélectionné en utilisant l'attribut IsChecked dans le code XAML.
Grâce à l'attribut IsThreeState à true, ce contrôle peut afficher un 3ème état dit intermédiaire. Pour afficher directement cet état, IsChecked prend alors la valeur `{x:Null}`.
Dans l'exemple fourni par Microsoft sur la gestion en lecture seule, l'utilisation du stade intermédiaire peut être intéressante dans le cas où le contenu ne correspond pas à true ou false : c'est le cas lorsque l'on regarde les propriétés pour un dossier qui contient à la fois des fichiers qui sont en lecture seule et d'autres qui peuvent être modifiés.
Il est également possible de réagir par rapport à l'état dans lequel se trouve le checkBox lorsque l'utilisateur a cliqué. Comme pour le bouton, le principe est le même, il faut définir dans le code-behind les méthodes qui doivent être appelées.
Checked : lorsque le checkBox est sélectionné
Unchecked : lorsque le checkBox est désélectionné
Interminate : lorsque le checkBox est dans l'état intermédiaire.

13.6.7.3. ListBox

Ce contrôle affiche une liste de choix pour une sélection. Il est constitué de ListBoxItem. Il y a 3 modes de sélection possible :
Single : un seul élément peut être sélectionné à la fois
Multiple : plusieurs éléments peuvent être sélectionnés à la fois
Extended : plusieurs éléments peuvent être sélectionnés à la fois. Pour sélectionner plusieurs éléments, il faut utiliser la touche ctrl lors des clicks.
L'événement lors du changement de sélection peut être récupéré en spécifiant le nom de la méthode à appeler dans l'attribut SelectionChanged.

```
<Grid>
    <StackPanel>
    <TextBox Name="tb" Width="140" Height="30"></TextBox>
    <ListBox Name="lb" Width="100" Height="55" SelectionChanged="PrintText"
SelectionMode="Single">
        <ListBoxItem>Item 1</ListBoxItem>
        <ListBoxItem>Item 2</ListBoxItem>
        <ListBoxItem>Item 3</ListBoxItem>
        <ListBoxItem>Item 4</ListBoxItem>
        <ListBoxItem>Item 5</ListBoxItem>
        <ListBoxItem>Item 6</ListBoxItem>
        <ListBoxItem>Item 7</ListBoxItem>
        <ListBoxItem>Item 8</ListBoxItem>
        <ListBoxItem>Item 9</ListBoxItem>
        <ListBoxItem>Item 10</ListBoxItem>
    </ListBox>
    </StackPanel>
</Grid>
```

Dans l'exemple fourni par Microsoft, la méthode PrintText est appelée à chaque changement de sélection. Pour récupérer l'élément sélectionné (qui est un

ListBoxItem), l'argument sender est casté en ListBox puis l'appel à la méthode SelectedItem dont le résultat est casté en ListBoxItem.

```
void PrintText(object sender, SelectionChangedEventArgs args)
{
    ListBoxItem lbi = ((sender as ListBox).SelectedItem as ListBoxItem);
    tb.Text = "   You selected " + lbi.Content.ToString() + ".";
}
```

Une version en n'utilisant pas de cast avec le as :

```
void PrintText(object sender, SelectionChangedEventArgs args)
{
    ListBox lb = (ListBox)sender;
    ListBoxItem lbi = (ListBoxItem)lb.SelectedItem;
    tb.Text = "   You selected " + lbi.Content.ToString() + ".";
}
```

13.6.7.4. ComboBox

Ce contrôle permet d'afficher une liste déroulante de sélection. Il reprend le même système que le ListBox pour afficher des éléments dans la liste (ComboBoxItem).
Les attributs IsEditable et IsReadOnly permettent de modifier le comportement du contrôle :

IsEditable	IsReadOnly	
False	False	Saisir un string pour sélectionner un item (si le début du string saisi ne correspond à aucun élément de la liste, rien ne se passe). Ne peut pas sélectionner une partie du texte sélectionné
False	True	Idem que False, False
True	False	Curseur lors du click sur l'item sélectionné. Saisir un string pour sélectionner un item. Copier/Coller le contenu sélectionné
True	True	Possible uniquement de copier le contenu sélectionné. Impossible d'écrire et de sélectionner un item en écrivant les premières lettres de l'item.

13.6.7.5. RadioButton

Ce contrôle ressemble à un CheckBox à la différence qu'une fois le contrôle sélectionné, il n'est pas possible de le désélectionner. En général, le RadioButton est utilisé dans un groupe de contrôle (la sélection étant exclusive).
Il y a deux façons pour faire en sorte que les RadioButton soient groupés :

- S'ils sont dans le même conteneur.
- S'ils portent la même valeur dans l'attribut GroupName.

L'exemple ci-dessous montre les 3 cas de figure :

```
<Grid>
    <StackPanel>
        <GroupBox>
            <GroupBox.Header>Groupé par containeur</GroupBox.Header>
            <StackPanel>
                <RadioButton Name="rb1">Yes</RadioButton>
                <RadioButton Name="rb2">No</RadioButton>
                <RadioButton Name="rb3">No opinion</RadioButton>
            </StackPanel>
```

```
        </GroupBox>
        <GroupBox>
            <GroupBox.Header>Groupé avec GroupName</GroupBox.Header>
            <StackPanel>
                <RadioButton GroupName="rbgrp">Yes</RadioButton>
                <RadioButton GroupName="rbgrp">No</RadioButton>
                <RadioButton GroupName="rbgrp">No opinion</RadioButton>
            </StackPanel>
        </GroupBox>
        <GroupBox>
            <GroupBox.Header>Non groupé car dans différents
containeurs</GroupBox.Header>
            <StackPanel>
                <StackPanel>
                    <RadioButton Name="nrb1">Yes</RadioButton>
                </StackPanel>
                <StackPanel>
                    <RadioButton Name="nrb2">No</RadioButton>
                </StackPanel>
            </StackPanel>
        </GroupBox>
    </StackPanel>
</Grid>
```

13.6.7.6. Ajouter des éléments XAML depuis C#

Jusqu'à présent, nous avons vu comment placer des éléments XAML. Ceci suppose que l'application ne changera pas d'aspect tout au long de son exécution. Mais il se peut également que suivant les choix de l'utilisateur, il faille ajouter des éléments.
Pour ajouter des éléments, c'est très simple :
Dans le code behind de la fenêtre XAML, créer les contrôles à ajouter puis ajouter au conteneur dans lequel on souhaite afficher le contrôle.

Exemple de l'ajout d'un CheckBox dans le code-behind :

```
public Window1()
{
    InitializeComponent();
    CheckBox cb = new CheckBox();
    cb.Height = 16 ;
    cb.Width = 180 ;
    cb.Content = "C# CheckBox";
    cb.Name = "test";
    stackPanel1.Children.Add(cb);
}
```

Important :
Etant donné que l'ajout d'éléments se fait par rapport au parent, il faut pouvoir connaître le nom du parent. Donc, chaque élément déjà présent doit avoir un

nom et chaque item également pour le cas où il faut ajouter des éléments à l'intérieur.

13.6.8. XSLT & XPath

L'utilisation du XSLT nécessite 2 éléments :
Un fichier XML contenant les valeurs qui doivent être utilisées et un fichier XSLT pour appliquer la structure souhaitée aux données.
Ces deux fichiers peuvent être créés sous Visual Studio avec les déclarations par défaut (dans le cas du XSLT, tout le contenu après l'élément output peut être supprimé).
Pour bien pouvoir utiliser le XSLT, il faut connaître un minimum d'éléments utilisés couramment, le XPath (navigation dans les nœuds) et comment effectuer des tests.

13.6.8.1. Les éléments importants du XSLT

<xsl:template>
Avant de pouvoir commencer à traiter le document XML, il faut sélectionner la partie du document qui doit être traitée. La sélection s'effectue par rapport à un nœud : c'est là qu'intervient l'attribut match.
Pour sélectionner le root, soit on spécifie "/" ou le nom du root. Chaque fois qu'il faut descendre dans un nom enfant, on utilise / : /node/node2 -> <root><node><node2>…
On peut également demander à sélectionner des éléments quel que soit leur emplacement : //element sélectionne tous les éléments qui sont n'importe où dans le document.
x//element sélectionne tous les éléments qui descendent de x.

<xsl:apply-templates>
Pour que les nœuds enfants soient également traités, il faut utiliser l'apply-templates.
Il est possible d'avoir besoin de traiter un nœud plusieurs fois mais de différentes manières, c'est à ce moment qu'intervient l'attribut mode.
L'apply-templates prend alors dans l'attribut select le chemin du nœud à traiter et l'attribut mode contient le nom permettant d'identifier le bon template à prendre. Le mode doit également être défini dans xsl:template.

<xsl:value-of>
Extrait la valeur du nœud défini dans le select.

<xsl:element>
Comme son nom l'indique, il permet de générer des éléments (balises) par rapport au contenu du fichier XML. name="{@id}" donnera <@id />.

<xsl:attribute>
On trouve en général cet élément dans un xsl:element ou après avoir déclaré une balise en dur dans le XSLT (un TD pour une sortie HTML). Il permet de mettre à la balise des attributs.
Name permet de définir le nom de l'attribut. A l'intérieur de la balise, on trouve soit un élément text ou value-of.

```
<xsl:template match="//*">
  <xsl:for-each select="//node">
    <xsl:element name="{@id}">
```

```
      <xsl:attribute name="height">
        <xsl:value-of select="@height"/>
        <xsl:text>px</xsl:text>
      </xsl:attribute>
      test
    </xsl:element>
  </xsl:for-each>
</xsl:template>
```

Ce qui donnera par exemple comme sortie :

```
<a1 height="50px">test</a1>
```

<xsl:if>

Permet de faire un test. Si le test renvoie une liste d'éléments non vide, le contenu du if est utilisé. Il n'existe pas de else.
L'exemple suivant prend tous les enfants du nœud root et affiche une virgule entre chacun. Le test s'occupe de contrôler que pour mettre une virgule, le nœud courant ne soit pas le dernier.
Très pratique pour créer le code d'une fonction avec plusieurs arguments séparés par une virgule.

```
  <xsl:template match="//*">
    <xsl:for-each select="node">
      <xsl:value-of select="@id"/>
      <xsl:if test="not (position()=last())">
        <xsl:text>, </xsl:text>
      </xsl:if>
    </xsl:for-each>
  </xsl:template>
```

<xsl:choose>

Cet élément permet de faire un choix sur la transformation à effectuer parmi plusieurs possibilités (cela fait penser au switch). On peut mettre autant de possibilités que l'on souhaite en mettant à la suite l'élément <xsl:when> sans oublier de préciser dans l'attribut test l'expression à tester et mettant le contenu de la transformation à effectuer.
Il est également possible de préciser une opération par défaut à effectuer en utiliser l'élément <xsl:otherwise>.

```
<xsl:template match="/" >
  <xsl:for-each select="//*">
    <xsl:value-of select="@id"/>
    <xsl:choose>
      <xsl:when test="node[@id='a1']">
        AAAA
      </xsl:when>
      <xsl:when test="node[@id='a2']">
        BBBB
      </xsl:when>
      <xsl:otherwise>
        CCCC
      </xsl:otherwise>
    </xsl:choose>
 </xsl:for-each>
</xsl:template>
```

Le résultat de la transformation :

```
        AAAA
  a1
        BBBB
  a2
        CCCC
  a3
        CCCC
  a4
        CCCC
  a5
        CCCC
  a6
        CCCC
```

<xsl:variable>
Cet élément permet de stocker un contenu. Pour récupérer la valeur, il faut utiliser le nom de la variable précédé du signe $.

```
   <xsl:variable name="var1">
     valeurVar1
   </xsl:variable>
…
<xsl:value-of select="$var1" />
```

13.6.8.2. XPath

Il existe 2 moyens de déterminer dans quel contexte (emplacement) on se trouve (complète ou abrégée). J'utiliserai uniquement la version abrégée qui est la plus simple à utiliser.

Dans le cas d'un fichier XML avec la structure suivante :

```
<xml1>
  <xml2>
     … contenu dont il faut afficher les valeurs
  </xml2>
</xml1>
```

Pour parcourir, on utilise un for-each avec comme select xml1/xml2.
Si l'on rajoute maintenant entre xml1 et xml2 un nouveau niveau, la recherche ne fonctionnera plus :

```
<xml1>
  <xml1.5>
    <xml2>
       … contenu dont il faut afficher les valeurs
    </xml2>
  </xml1.5>
</xml1>
```

Soit on spécifie tous les nœuds à parcourir xml1/xml1.5/xml2, soit on peut omettre un nœud (celui du milieu) : xml1//xml2. On peut également uniquement préciser le dernier nœud : //xml2.
Attention toutefois dans ce dernier cas : il sélectionnera tous les nœuds portant le nom xml2, quel que soit la profondeur dans laquelle ils se trouvent.
En utilisant un value-of, on peut afficher le nœud courant ou d'autres informations. Voici quelques possibilités :

self :
Si l'on se trouve dans le nœud qui contient la valeur à reprendre, le select prend comme information ".".
attribute :
Si c'est la valeur d'un attribut du nœud qu'il faut récupérer, on utilise @ puis le nom de l'attribut.
parent :
Cela sélectionne le père du nœud courant (../)

13.6.8.3. Tester des expressions

XSLT offre la possibilité de tester des expressions afin de ne traiter que les nœuds qui contiennent une certaine valeur. La syntaxe est composée du nom du nœud suivi de l'expression entre crochet.
2 exemples pour un template ou un for-each :
node[@arg] :
Cherche un node qui a un attribut arg. A noter qu'il est possible de prendre que les nœuds qui ne contiennent pas cet attribut en utilisant le mot clé not et le test entre accolade : node[not(@arg)].
node[exemple='eee'] :
Pour chaque node trouvé, s'il contient un nœud enfant nommé « exemple » dont la valeur est « eee », le test sera vrai :

XML:
```
<root>
  <node id="a1" height="50">a
    <exemple>eee</exemple>
…
```

XSLT:
```
<xsl:template match="//*" >
  <xsl:for-each select="node[exemple='eee']">
    <xsl:value-of select="@id"/> - C'est vrai !
  </xsl:for-each>
</xsl:template>
```

Résultat:
```
a1 - C'est vrai !
```

13.6.8.4. Position et famille

Il est possible de connaître la position d'un noeud par rapport à ses frères grâce à position() et last() (attention, les méthodes first-of-type(), first-of-any(), last-of-type(), last-of-any() ne sont plus d'actualités).
Il est également possible d'accéder directement à un nœud précis en utilisant des crochets [].
L'exemple suivant montre l'utilisation des 3 possibilités :

XML:
```
<root>
  <node id="a1" height="50">a</node>
  <node id="a2" height="30">b</node>
  <node id="a3" height="50">c</node>
  <node id="a4" height="10">d</node>
  <node id="a5" height="20">e</node>
  <node id="a6" height="50">f</node>
</root>
```

XSLT:

```
<xsl:template match="/" >
  <xsl:for-each select="//node">
    <xsl:if test="position() = 1">
      Premier : <xsl:value-of select="@id"/>
    </xsl:if>
    <xsl:if test="position()=last()">
      Dernier : <xsl:value-of select="@id"/>
    </xsl:if>
  </xsl:for-each>
  contenu du 2eme node : <xsl:value-of select="//node[2]"/>
</xsl:template>
```

Résultat:

```
Premier : a1
Dernier : a6
contenu du 2eme node : b
```

Il peut être intéressant de savoir par rapport au nœud courant quels sont ses enfants, descendants, parents ou encore ses ancêtres.
Pour illustrer, voici le fichier XML qui a été utilisé :

```
<root>
  <nodeA1 id="a1">
    <nodeA2 id="a2">
      <nodeA3 id="a3"></nodeA3>
    </nodeA2>
    <nodeA2 id="a4"></nodeA2>
  </nodeA1>
  <nodeB1 id="a5"></nodeB1>
  <nodeB1 id="a6"></nodeB1>
</root>
```

Le template XSLT suivant a été utilisé.

```
  <xsl:template match="/">
    <!-- /root//* : prends tous les nœuds contenu dans root, peu importe leur
position -->
    <xsl:for-each select="/root//*">
      <!-- Appliquer le template portant le nom print -->
      <xsl:call-template name="print" />
    </xsl:for-each>
  </xsl:template>

  <xsl:template name="print">
    <!-- Afficher le parent : son nom et l'attribut id (. = le noeud en cours, @
pour l'argument -->
    Parent : <xsl:value-of select="name()"/> (<xsl:value-of select="./@id"/>)
    Enfant :
    <!-- boucle pour parcourir tous les enfants du noeud courant (les petits
enfants apparaissent parce que l'on parcours ensuite chaque enfant) -->
    <xsl:for-each select="child::*">
      <xsl:value-of select="name()"/> (<xsl:value-of select="./@id"/>) -
      <xsl:text/>
    </xsl:for-each>
  </xsl:template>
```

Entre les différents exemples qui suivent, seul l'élément souligné et surligné en jaune a été modifié ainsi que les textes écrits en dur (Parent : et Enfant :).

Child :
Permet de trouver les enfants du nœud courant. Pour les afficher, une boucle for-each avec comme select child::* permet de réaliser cette opération.

```
Sortie:
    Parent : nodeA1 (a1)
    Enfant :
    nodeA2 (a2) -
      nodeA2 (a4) -

    Parent : nodeA2 (a2)
    Enfant :
    nodeA3 (a3) -

    Parent : nodeA3 (a3)
    Enfant :

    Parent : nodeA2 (a4)
    Enfant :

    Parent : nodeB1 (a5)
    Enfant :

    Parent : nodeB1 (a6)
    Enfant :
```

Descendant :
Permet de trouver tous les descendants du nœud courant. Dans ce cas également, une boucle for-each est intéressant pour parcourir les nœuds et le select prendra la valeur descendant::*.

```
Sortie:
    Parent : nodeA1 (a1)
    Enfant :
    nodeA2 (a2) -
      nodeA3 (a3) -
      nodeA2 (a4) -

    Parent : nodeA2 (a2)
    Enfant :
    nodeA3 (a3) -

    Parent : nodeA3 (a3)
    Enfant :

    Parent : nodeA2 (a4)
    Enfant :

    Parent : nodeB1 (a5)
    Enfant :

    Parent : nodeB1 (a6)
    Enfant :
```

Parent :
Permet de trouver le parent du nœud courant en utilisant parent::*

```
Sortie:
    Enfant : nodeA1 (a1)
    Parent :
    root () -

    Enfant : nodeA2 (a2)
    Parent :
    nodeA1 (a1) -

    Enfant : nodeA3 (a3)
    Parent :
    nodeA2 (a2) -
```

```
Enfant : nodeA2 (a4)
Parent :
nodeA1 (a1) -

Enfant : nodeB1 (a5)
Parent :
root () -

Enfant : nodeB1 (a6)
Parent :
root () -
```

Ancestor :
Permet de trouver tous les ancêtres du nœud courant en utilisant ancestor::* :

```
Enfant : nodeA1 (a1)
Parent :
root () -

Enfant : nodeA2 (a2)
Parent :
root () -
  nodeA1 (a1) -

Enfant : nodeA3 (a3)
Parent :
root () -
  nodeA1 (a1) -
  nodeA2 (a2) -

Enfant : nodeA2 (a4)
Parent :
root () -
  nodeA1 (a1) -

Enfant : nodeB1 (a5)
Parent :
root () -

Enfant : nodeB1 (a6)
Parent :
root () -
```

On remarque que pour le node avec l'id a3, le node portant l'id a4 n'est pas un ancêtre (c'est un oncle).

Following-sibling :
Permet de trouver les frères qui suivent en utilisant following-sibling::* :

```
Moi : nodeA1 (a1)
Mes frères suivants :
nodeB1 (a5) -
  nodeB1 (a6) -

Moi : nodeA2 (a2)
Mes frères suivants :
nodeA2 (a4) -

Moi : nodeA3 (a3)
Mes frères suivants :

Moi : nodeA2 (a4)
Mes frères suivants :

Moi : nodeB1 (a5)
```

```
Mes frères suivants :
nodeB1 (a6) -

Moi : nodeB1 (a6)
Mes frères suivants :
```

On constate que le nœud avec l'id a1 à 2 frères qui le suit tandis que le nœud avec l'id a6 n'a personne qui le suit (c'est le cadet !).

preceding-sibling :
Permet de trouver les frères qui précèdent en utilisant preceding-sibling::*.
Le nœud avec l'id a1 n'aura pas de précédents frères (c'est l'aîné) tandis que le nœud avec l'id a6 aura comme frères précédents les nœuds avec les id a5 et a1.

Following :
Permet de trouver tous les nœuds qui suivent (les enfants du nœud courrant ne sont pas compris mais les enfants des autres nœuds oui) en utilisant following::*.

```
Moi : nodeA1 (a1)
Noeuds suivants :
nodeB1 (a5) -
  nodeB1 (a6) -

Moi : nodeA2 (a2)
Noeuds suivants :
nodeA2 (a4) -
  nodeB1 (a5) -
  nodeB1 (a6) -

Moi : nodeA3 (a3)
Noeuds suivants :
nodeA2 (a4) -
  nodeB1 (a5) -
  nodeB1 (a6) -

Moi : nodeA2 (a4)
Noeuds suivants :
nodeB1 (a5) -
  nodeB1 (a6) -

Moi : nodeB1 (a5)
Noeuds suivants :
nodeB1 (a6) -

Moi : nodeB1 (a6)
Noeuds suivants :
```

Preceding :
Permet de trouver tous les nœuds qui précèdent le nœud courant en utilisant preceding::*. Attention, le parent, grand-parent et ainsi de suite n'apparaissent pas.

```
Moi : nodeA1 (a1)
Noeuds précédents :

Moi : nodeA2 (a2)
Noeuds précédents :

Moi : nodeA3 (a3)
Noeuds précédents :
```

```
Moi : nodeA2 (a4)
Noeuds précédents :
nodeA2 (a2) -
  nodeA3 (a3) -

Moi : nodeB1 (a5)
Noeuds précédents :
nodeA1 (a1) -
  nodeA2 (a2) -
  nodeA3 (a3) -
  nodeA2 (a4) -

Moi : nodeB1 (a6)
Noeuds précédents :
nodeA1 (a1) -
  nodeA2 (a2) -
  nodeA3 (a3) -
  nodeA2 (a4) -
  nodeB1 (a5) -
```

descendant-or-self :
Se comporte comme descendant mais en incluant le nœud courant.

ancestor-or-self :
Se comporte comme ancestor mais en incluant le nœud courant.

Un schéma visuel qui regroupe les différentes méthodes pour connaître les enfants, parents,… est disponible à l'adresse :
http://nwalsh.com/docs/tutorials/xsl/xsl/slides.html (Slide 22: Axis Specifiers)

13.7. Rapports hebdomadaires

Sujet du travail de diplôme :
WSDL & XAML

Etudiant :
Vionnet Olivier

Professeur responsable :
Schumacher Michael Ignaz
Russo David

Semaine : *1-3*
Woche :

Décompte des heures hebdomadaires
Deklaration Wochenstunden

Tâche/ Aufgabe	Heures/ Stunden
Recherche d'information et divers essais avant le début du projet (armée lors du début) Etablissement du cahier des charges, MS-Project Installation des outils	54h

Sujet du travail de diplôme :
WSDL & XAML

Etudiant :
Vionnet Olivier

Professeur responsable :
Schumacher Michael Ignaz
Russo David

Semaine : *4-7*
Woche : *29.11.2007*

Décompte des heures hebdomadaires
Deklaration Wochenstunden

Tâche/ Aufgabe	Heures/ Stunden
Explications sur différentes définitions Exemple et structure d'un document WSDL Création d'un client consommant un web service Recherche sur les outils permettant de récupérer les informations d'un web service. XAML : apprentissage des différents éléments qui le constituent (de loin pas fini)	24h

Divers/ Diverses

Je pensai que l'apprentissage de XAML irait très vite mais ce n'est pas le cas. Ca prend pas mal de temps de lire la documentation (et de comprendre) et de coder des exemples pour illustrer les différentes fonctionnalités.

Sujet du travail de diplôme :
WSDL & XAML

Etudiant :
Vionnet Olivier

Professeur responsable :
Schumacher Michael Ignaz
Russo David

Semaine : *8-13*
Woche : *23.01.2008*

Décompte des heures hebdomadaires
Deklaration Wochenstunden

Tâche/ Aufgabe	Heures/ Stunden
Continué et terminé la recherche sur les différents éléments de XAML.	42 h
Continué et terminé la recherche sur les web services et le WSDL.	
Documentation sur les différentes recherches et modification du fichier	

Problèmes rencontrés / Solutions trouvées
Probleme / Lösungen

Description du problème Beschreibung des Problems	Description de la solution (si trouvée…) Beschreibung der Lösung (wenn gefunden…)
Quelques problèmes avec la beta de Visual Studio 2008. Impossible de créer un nouveau projet.	Après quelques recherches sur Internet et réparation, finallement tout a été désinstallé et la version finale a été installée. http://207.46.236.188/MSDN/ShowPost.aspx?PostID=2491702&SiteID=1 http://www.codeplex.com/WarcraftAddOnStudio/WorkItem/View.aspx?WorkItemId=724
Dans la version finale de VS, impossible d'utiliser des éléments XAML drag et drop...	Ca vient du fait d'avoir eu la beta 2 installée sur le pc. Il a fallu se rendre dans %USERPROFILE%\AppData\Local\Microsoft\VisualStudio\9.0 pour effacer tous les fichiers portants l'extension TBD ce qui a permit de réinitialiser la toolbox. http://forums.microsoft.com/MSDN/ShowPost.aspx?PostID=2633726&SiteID=1

Divers/ Diverses

Je pensai que l'apprentissage de XAML irait très vite mais ce n'est pas le cas. Ca prend pas mal de temps de lire la documentation (et de comprendre) et de coder des exemples pour illustrer les différentes fonctionnalités.

Sujet du travail de diplôme :
WSDL & XAML

Etudiant :
Vionnet Olivier

Professeur responsable :
Schumacher Michael Ignaz
Russo David

Semaine : *14*
Woche : *27.01.2008*
31.01.2008

Décompte des heures hebdomadaires
Deklaration Wochenstunden

Tâche/ Aufgabe	Heures/ Stunden

Création d'un projet en ligne de commande sous VS 2008: générer une application console en utilisant le web service Service Model Sample (fourni dans le sdk de Microsoft). Contient 4 opérations (calculatrice)

Récupération des informations du wsdl :

- Génération du proxy et fichier config (svcutil).
- Génération du proxy sous forme de dll.
- Accès à la dll (runtime) pour récupérer toutes les méthodes et leur signature.
- Affichage console des informations trouvées dans la dll.

Génération du fichier MyProgram.cs qui doit s'occuper d'utiliser les méthodes du web service avec des valeurs par défaut (tout n'est de loin pas généré « dynamiquement », ceci parce qu'il risque d'avoir beaucoup de différences entre du code console et du code dans un environnement XAML).

Génération d'un exécutable en utilisant le fichier MyProgram.cs et mettre tous les fichiers nécessaires dans un même répertoire (exe, proxy.dll, .config)

Recherche sur les possibilités de générer du code (qu'est ce qu'il faut utiliser).
Choix retenu : XSLT

Points à éclaircir pour le futur :

- Utilisation d'un autre binding que wsHttpBinding (théoriquement, ça ne devrait pas poser de problème car cette partie est déterminée par le

fichier .config généré par rapport au WSDL

Garder un œil sur les assemblies à utiliser dans le fichier MyProgram.cs

Documentation : définitions pour CodeDom, XSLT, mise à jour des références, chapitre 9 dans documentation :

- Etape 1 : générer un client en mode console
- Etape 2 : Choix de méthode pour générer du code

Prochaine étape :

- recherche d'information sur XSLT pour C# et apprentissage rapide du langage pour ensuite implémenter de le projet de test en cours.
- Créer un fichier xml pour stocker les valeurs du web service
- Créer un template XSLT pour traiter les données

Tester la génération et exécuter le résultat

Total : 30h

Problèmes rencontrés / Solutions trouvées
Probleme / Lösungen

Description du problème Beschreibung des Problems	Description de la solution (si trouvée…) Beschreibung der Lösung (wenn gefunden…)
Parti sur une très mauvaise piste : parser à la main les fichiers créés par svcutil. Beaucoup de problèmes, difficile, long et au final, pas terrible.	En ajoutant dans un projet C# le fichier du proxy généré, j'ai pu accéder aux méthodes exposées. Ce qui m'a amené à penser que pour faire ceci en runtime, il fallait juste trouver le moyen d'intégrer le proxy (et c'est pourquoi j'ai fait une dll avec le proxy pour pouvoir l'utiliser).
Stocker les chemins d'accès à certaines dll ou programmes utilisés (csc, svcutil) ainsi que pour les sorties. Utiliser un .resx n'est pas le mieux.	Finalement, opté pour un fichier settings. A noter qu'il faut mettre pour chaque entrée le scope en User, sinon impossible de sauver un changement de valeur pour une entrée...
Quelques problèmes pour générer l'exécutable final (qui utilise le code de MyProgram.cs pour utiliser les méthodes du web service) et faire en sorte qu'il fonctionne lorsqu'il est utilisé.	La génération a été trouvée après plusieurs tests. Le problème du plantage lors de l'utilisation là également, trouvé après pas mal d'essais: Je pensai que la dll était incluse dans l'exe. Et non, il faut qu'elle soit dans le même dossier que l'exe. L'utilisation du fichier de configuration (.config). Pour que l'exe l'utilise, il faut qu'il porte le nom de l'exe .config (<nom exe>.exe.config)
MyProgram.cs est généré avec un StreamWriter (chaque ligne est mise en utilisant un WriteLine. Ca va très bien pour un test mais ça va vite devenir un problème pour écrire de façon dynamique les éléments de code (et le fichier XAML).Trouver un	Recherche des différents moyens possibles : bruteforce (StreamWriter) et codeDOM écarté (pas adapté). Choix retenu : XSLT -> peut produire tout comme code

autre moyen pour générer le code	(car txt en sortie), lecture facile des templates,...

Divers/ Diverses

Prochaine étape : Se mettre au XSLT, générer fichier de données XML et un template XSLT pour produire le fichier permettant d'utiliser le service web.

Sujet du travail de diplôme :
WSDL & XAML

Etudiant :
Vionnet Olivier

Professeur responsable :
Schumacher Michael Ignaz
Russo David

Semaine : *15*
Woche : *01.02.2008*
07.01.2008

Décompte des heures hebdomadaires
Deklaration Wochenstunden

Tâche/ Aufgabe	Heures/ Stunden

Recherche d'information sur XSLT et C#

Apprentissage du XSLT et XPath ainsi que documentation

Compilation d'un projet XAML (avec msbuild en console à la main sur un projet existant).

Création d'un projet pour générer une application XAML

Préparation des fichiers codes nécessaires pour produire un projet (sans .csproj) : Reprendre tous les fichiers XAML, C# d'un projet HelloWorld (juste afficher un string dans le Window) et transformer chaque fichier pour qu'il puisse être transformé avec XSLT.

Pour le moment, pour chaque fichier, il y a un fichier xml (qui ne contient qu'un nœud root) et un fichier XSLT qui contient en dur le code XAML ou C#. L'objectif est de pouvoir transformer chacun de ses fichiers et de voir s'il y a des problèmes dans le XSLT puis faire à la main un msbuild.

Prochaine étape :

- Modifier chaque fichier XSLT pour pouvoir générer du code selon le fichier XML contenant les données (avec les informations provenant du web service de la calculatrice).
- Générer pour chaque opération un écran (tabcontrol) contenant pour chaque paramètre d'entrée et le paramètre de sortie les éléments XAML (un label et un textbox) et un bouton pour lancer une action.
- Modifier le fichier XSLT qui contient le code source pour utiliser la fenêtre et réaliser l'action (concaténer du texte et l'afficher).
- Modifications pour pouvoir interagir avec le web service (mettre le proxy

et modifier le code d'action.

- Se pencher sur les éléments devant être dynamique et mettre en place une fenêtre permettant de configurer de façon basique l'application parent (chemins d'accès aux outils) et enfants (nom, namespace de la nouvelle application…).
- Se pencher sur le fichier csproj indispensable à msbuild pour générer l'application.
- Au final, je devrais pouvoir générer sans intervention manuelle le client pour le web service de la calculatrice.

Total : 24h

Problèmes rencontrés / Solutions trouvées
Probleme / Lösungen

Description du problème Beschreibung des Problems	Description de la solution (si trouvée…) Beschreibung der Lösung (wenn gefunden…)
Pas mal de problèmes avec le XSLT : Pas trouvé de moyen pour fixer correctement l'indentation du code généré. Dans les templates, ça ne suit pas l'indentation Saut de ligne allant d'une à plusieurs lignes...	Pas trouvé de solution miracle pour la mise en page du code produit. La meilleure solution est de bien disposer les éléments dans les templates XSLT (plusieurs balises sur une ligne pour éviter le retour de ligne). Pour l'indentation, je ne sais pas trop... Faire à la main pour chaque ligne avec un espace, c'est pas terrible. Vu que ça ne dérange que pour la lecture du code produit, si j'arrive à avoir des blocs de code avec le même retrait, ça sera déjà pas mal. Et il y a toujours la possibilité lorsque l'on reprend le code produit pour l'implémenter dans un autre projet par exemple de le formater.
De la peine à me retrouver avec le Xpath (ce qui donne souvent comme résultat qu'un bout de template n'est pas exécuté parce que le chemin est faux par rapport au context).	

Toujours en XSLT, pour produire le code d'une fenêtre XAML, un problème au niveau du code généré : certaines balises ont un attribut supplémentaire xmlns="" ce qui donne des erreurs lors du build avec msbuild	Après pas mal d'essais et de recherche de solutions sur le net (sans succès), j'ai trouvé la cause.
	Pour avoir une fenêtre, il faut avoir une déclaration dans ce style (urls supprimés) :
	`<Window x:Class="WPFHello.Window1" xmlns="…" xmlns:x="…" Title="Window1" Height="300" Width="300">`
	Il semblerait que le namespace xmlns:x (et xmlns) influencent le XSLT et que dans les templates qui suivent, il n'y a pas d'informations.En essayant, j'ai trouvé une solution qui semble être la bonne :
	Dans la déclaration du stylesheet du fichier XSLT, j'ai rajouté les références qui sont utilisées par Window :
	`xmlns:x="http://schemas.microsoft.com`

	/winfx/2006/xaml" xmlns="http://schemas. microsoft.com /winfx/2006/xaml/presentation"
Jusqu'à présent, je n'ai pas trouvé d'exemples sur la génération de code XAML.	Pas beaucoup de choix que d'essayer et de modifier les fichiers au fur et à mesure que les problèmes arrivent

Sujet du travail de diplôme :
WSDL & XAML

Etudiant :
Vionnet Olivier

Professeur responsable :
Schumacher Michael Ignaz
Russo David

Semaine : *16*
Woche : *08.02.2008*
14.02.2008

Décompte des heures hebdomadaires
Deklaration Wochenstunden

Tâche/ Aufgabe	Heures/ Stunden

Modification de tous les fichiers de génération de code : le code est maintenant créé par rapport aux informations des fichiers XML provenant du web service Calculatrice. Chaque opération = un tabitem
Mise en place d'une partie du fichier de configuration
Le projet peut être maintenant exécuté et généré l'application finale sans intervention manuelle (mis appart la compilation du projet XAML avec msbuild).

Prochaine étape :
Le fichier csproj doit pouvoir être généré de façon automatique (écrit en dur dans le xslt avec certains champs pouvant être modifié par les paramètres de configuration)
Terminer la partie configuration : Pouvoir mettre les assemblies que l'on souhaite utiliser puis debugger (s'assurer que les changements faits sont utilisés dans l'application lors de la génération de l'application finale).
Implémenter pour la génération tous les types standard
Implémenter pour la génération les types costum (par exemple une classe Personne)

Total : 24h

Problèmes rencontrés / Solutions trouvées
Probleme / Lösungen

Description du problème Beschreibung des Problems	Description de la solution (si trouvée…) Beschreibung der Lösung (wenn gefunden…)
Je commence à galérer...	
Beaucoup de temps pris pour trouver toutes les erreurs qui empêchent la génération de l'application finale avec msbuild.	S'appuyer sur les erreurs signaler par msbuild pour ensuite modifier la génération des fichiers qui posent problèmes.
Plantage de l'application finale lors de l'utilisation	Je me suis de nouveau fait avoir... ça venait du fait que soit le fichier de configuration .exe.config n'était pas avec l'exécutable ou qu'il portait un mauvais nom.

Sujet du travail de diplôme :
WSDL & XAML

Etudiant :
Vionnet Olivier

Professeur responsable :
Schumacher Michael Ignaz
Russo David

Semaine : *17*
Woche : *15.02.2008*
21.02.2008

Décompte des heures hebdomadaires
Deklaration Wochenstunden

Tâche/ Aufgabe	Heures/ Stunden

Génération automatique du fichier csproj (les noms sont mis dynamiquement ainsi que les références)

Fenêtre de configuration : les assemblies / références peuvent maintenant être mises et modifiées. Debug de l'application par rapport aux différents changements dans les noms de fichiers, namespace lorsque l'on les change dans la fenêtre de configuration

Implémenté les types standards

Implémenté les types costums

Implémenté les types complex

Test sur les différents bindings : WSHttpBinding, BasicBinding, netTcpBinding et WS Dual http bindings (dual pose problème).

Prochaine étape : (discuté en séance)

- Modifier le code pour éviter les bugs pouvant survenir lors d'une regénération (2 générations sans fermer l'application)
- Chaque génération doit être mise dans un dossier séparé (il y aura toujours le dossier de sortie standard dans lequel je vais écrire le nouveau dossier et mettre tous les fichiers
- Configuration de l'application avec un fichier externe : pouvoir charger un fichier qui va s'occuper de modifier le contenu de la fenêtre de configuration
- Fichier log de génération : doit pouvoir récupérer les messages d'erreur qui surviennent lors de la génération de l'application (outils de génération tels que svcutil, msbuild)

- Système de test de web service : donner à l'utilisateur un fichier contenant la structure permettant de rentrer des données à injecter ensuite dans le web service et écrire le résultat.

Séance avec Michael Schumacher

Total : 47h

Problèmes rencontrés / Solutions trouvées
Probleme / Lösungen

Description du problème Beschreibung des Problems	Description de la solution (si trouvée…) Beschreibung der Lösung (wenn gefunden…)
WS Dual http bindings (duplex) : La génération du code pour ce binding ne fonctionne pas. La structure du code est différente par rapport au binding standard (l'envoi des informations au web service se fait sur une interface et le web service renvoie la réponse depuis une autre interface).	Laissé de côté ce cas (je gère les 3 types les plus courant de binding) pour ce qui est de la génération automatique. Par contre, c'est un point à garder à l'esprit pour la partie génération avec un wizard (l'utilisateur devra faire des choix)
Gestion des erreurs : Actuellement, un try-catch englobe tout le contenu du bouton qui récupère les informations saisies et qui écrit les informations provenant du web service. Si l'utilisateur rentre un string dans un champ de int, une erreur est affichée (stack de l'exception) mais pas d'information sur le champ où se trouve l'erreur	Modifier la gestion d'erreur : faire pour chaque champ un contrôle (A faire)

Sujet du travail de diplôme :
WSDL & XAML

Etudiant :
Vionnet Olivier

Professeur responsable :
Schumacher Michael Ignaz
Russo David

Semaine : *18*
Woche : *22.02.2008*
28.02.2008

Décompte des heures hebdomadaires
Deklaration Wochenstunden

Tâche/ Aufgabe	Heures/ Stunden
Modification du code pour éviter les bugs de regénération Chaque génération est faite dans un dossier différent (date-heure-minutes-secondes) Modification des options : ajout, débug et implémenté la possibilité de sauver la configuration et de la recharger (fichiers (xml) stockés dans un dossier, chargement se fait par liste déroulante) Fichier de log de génération : pour le moment que sur la génération automatique. Récupère les erreurs survenant lors de l'utilisation des outils en mode console Modification de l'affichage lors de la génération : les fenêtres consoles n'apparaissent plus. Les informations sur ce que fait l'application lors de la génération sont affichées dans un statusbar. Actuellement en cours : Système de test d'un web service avec un fichier de type csv. Déroulement : L'utilisateur clique sur le menu de test. L'application va générer : Un dossier de test Générer les fichiers proxys et configuration créer une dll et récupérer les informations du web service Ouverture d'une nouvelle fenêtre affichant la liste des opérations disponibles avec leur signature. L'utilisateur ne peut sélectionner qu'une opération à la fois. Une fois le choix validé, l'application va générer les fichiers de données xml : 1 qui va servir à construire le fichier csv (ne contient que le contenu de l'opération du web service) et un fichier de données (avec les namespaces et autres informations nécessaires pour construire l'exécutable.	

Construction du fichier csv :
1ère ligne : les noms des champs préfixés avec I_ pour input, O_ pour output, OE_ pour le résultat espéré. Points virgule comme séparateur.

Nouvelle fenêtre demandant si l'utilisateur veut lancer le test (ceci afin qu'il puisse mettre les données à tester dans le fichier csv

Lancer les tests et affichage du résultat

Prochaine étape :

- Continuer la partie des tests
- Résoudre un bug de génération : si la valeur de retour du web service est un type array, la génération du code est totalement fausse
- Modifier la gestion des erreurs des types saisis lors de l'utilisation de l'application générée
- Développer des web services « simples » pour tester la génération de l'application
- Tester l'application avec des web services trouvés sur Internet

Total : 24h

Problèmes rencontrés / Solutions trouvées
Probleme / Lösungen

Description du problème Beschreibung des Problems	Description de la solution (si trouvée…) Beschreibung der Lösung (wenn gefunden…)
Problème de threads avec l'affichage lors de la génération : Lorsque le traitement est en cours, le bouton de click reste enfoncé et rien ne bouge tant que tout le traitement n'a pas été entièrement fait. 2 threads ne suffisent pas (un pour le traitement et un pour actualisé l'affichage dans le statusbar). Lors de l'exécution à 2 threads, j'obtiens l'exception System.InvalidOperationException : thread non propriétaire.	En consultant le blog de Simon Boigelot (http://www.simonboigelot.com/post/SystemInvalidOperationException---thread-non-propri%C3%A9taire.aspx), j'ai appris que dans WPF, chaque thread qui crée un objet est le propriétaire. Pour pouvoir faire une modification, il faut pouvoir intéragir avec le thread qui a créé l'interface. C'est là qu'intervient le Dispatcher : il permet d'ajouter une opération à faire dans le thread. Je me suis basé sur cet exemple pour résoudre le problème et mettre à jour le statusbar

Sujet du travail de diplôme :
WSDL & XAML

Etudiant :
Vionnet Olivier

Professeur responsable :
Schumacher Michael Ignaz
Russo David

Semaine : *19*
Woche : *29.02.2008*
09.03.2008

Décompte des heures hebdomadaires
Deklaration Wochenstunden

Tâche/ Aufgabe	Heures/ Stunden

Modification pour le code utilisant le delegate et dispatcher pour afficher un texte dans le statusbar. Maintenant, il n'y a plus de threads inutiles (avant, dans la fenêtre xaml, chaque type de modification était fait en utilisant un thread). Lorsqu'une fonction prend du temps à être traitée, elle est contenue dans un thread et l'affichage est modifié au travers d'un delegate contenu dans la fenêtre xaml (objet passé en paramètre). Nettement mieux.

Modification de la gestion des erreurs lors de la génération automatique :
Avant, un try-catch global enregistrait l'erreur et le stack était renvoyé.
Maintenant, chaque traitement parse est entouré d'un try-catch. En bas du programme, un textbloc empile les erreurs avec le message d'erreur ainsi que le champ qui a levé l'exception.
Si l'opération est contenue dans une ligne de code avec d'autres opérations, le try-catch est effectué avant la ligne de code. La ligne de code reste néanmoins traitées (et génèrera une erreur qui sera prise dans le try-catch global).
C'est la solution la plus simple que j'ai trouvée pour régler ce problème sans devoir repartir à 0 avec les templates.

Mis en place la partie « test » : ça fonctionne mais le traitement est fait sur une ancienne version du code (tiré de la partie génération automatique).
Pas une très bonne idée car ça me fait maintenant 2 endroits différents à modifier lorsque je découvre des bugs de génération mais en contre partie, ça permet d'avoir quelque chose à montrer sur comment je compte faire les tests.

Génération automatique :
Découverte de 2 problèmes qui sont en plus ou moins corrigés…

Tableau de type costum :
En testant le web service CDYNE (http://ws.cdyne.com/SpellChecker/check.asmx), j'ai constaté que mon application ne traite pas les tableaux de type costum. Après pas mal d'essais, ça fonctionne pour les cas où le tableau est en sortie, en paramètres d'entrée (pas encapsulé dans une classe).
C'est assez difficile de faire les modifications dans les templates xslt par rapport aux différents éléments qu'il faut traiter.
Je ne sais pas si j'arriverai à traiter les cas imbriqués (un tableau dans un tableau par exemple) de manière récursive sans que ça génère d'autres erreurs.

Si j'arrive à trouver une solution qui fonctionne, il faudra encore faire les modifications dans la partie test.

Problème de binding :
Il peut y avoir plusieurs façons d'utiliser un web service (binding). Jusqu'à présent, je n'avais qu'un moyen défini dans le fichier de configuration. Avec le web service CDYNE, il y en a 3 différents ce qui pose quelques problèmes : lorsque j'utilise l'application générée, il y a une erreur de binding (il ne sait pas quelle méthode doit être utilisée pour utiliser le web service car il y en a plusieurs). Pour résoudre ce cas, lorsque je crée l'objet client (checkSoapClient client = new checkSoapClient();), il faut que je passe en paramètre le nom du endpoint à utiliser.

D'autre part, lors de la génération du fichier de configuration, svcutil renvoie des erreurs (http://forums.microsoft.com/MSDN/ShowPost.aspx?PostID=2893137&SiteID=1&pageid=0). Il n'arrive pas à traiter 2 bindings (Get et Post) et donc, dans le fichier de configuration, je n'ai que les bindings pour le soap qui sont présents.
J'ai modifié le code qui se charge de récupérer le nom de la classe pour créer le client. Je récupère maintenant tous les bindings.
Ensuite, je contrôle que les bindings sont bien présents dans le fichier de configuration (je me base sur l'attribut contract du nœud endpoint). Dès qu'un est présent, j'arrête les tests et je l'utilise. Si je n'en trouve aucun, c'est qu'il n'y a pas l'attribut contract et j'utilise celui « par défaut ».
C'est la seule solution que j'ai trouvée jusqu'à présent qui me permette d'utiliser le web service CDYNE et les web services que j'ai créés.

Problèmes actuellement non résolus :

Si j'envoie un Hello au web service CDYNE, mon application génère une exception générale (par contre, si je marque n'importe quoi qui n'est pas un mot connu du dictionnaire, j'ai une réponse en retour sans erreur). Je ne sais pas encore pourquoi ça fait ceci, si c'est normal et si ce n'est pas normal, s'il y a moyen de corriger.

Remise à zéro des champs de sortie dans l'application générée : actuellement, si j'utilise plusieurs fois la méthode du web service, les champs ne sont pas remis avec aucune valeur.
Je ne sais pas encore comment je vais résoudre ce problème

Prochaine étape :

Essayer de résoudre le plus rapidement les problèmes non résolus et tester avec d'autres web service puis ensuite faire les modifications dans la partie test.

Total : 39h

Sujet du travail de diplôme :
WSDL & XAML

Etudiant :
Vionnet Olivier

Professeur responsable :
Schumacher Michael Ignaz
Russo David

Semaine : *20*
Woche : *10.03.2008*
13.03.2008

Décompte des heures hebdomadaires
Deklaration Wochenstunden

Tâche/ Aufgabe	Heures/ Stunden

Suite à la séance du 10 avril, voici le programme prévu :

1. Sortir quel serait les éléments à rendre paramétrable dans la partie wizard qui ne le sont pas actuellement dans la génération automatique (sans tomber dans la création d'un générateur graphique).
2. Gestion des tests : explorer la piste Nunit et voir s'il est possible de l'utiliser pour faire les tests des web services.
3. Expression : essayer ce programme pour voir comment sont construits certains composants (notamment ce qui touche au grid).
4. Remplacer les textBox par des userControls
5. Restructurer le XSLT pour mieux s'y retrouver et qu'il puisse également être le résultat d'une transformation XSLT précédente

Mais avant de pouvoir s'investir dans les points ci-dessus, il faut absolument que je résolve les problèmes de tableau costum et pour le moment, je n'y arrive pas (les modifications que je fais génèrent d'autres bugs, l'instanciation des tableaux posent également des problèmes...)
De plus, j'ai identifié un autre problème :
S'il y a un tableau de costum (tbl2) dans un tableau de costum (tbl1), le contenu de tbl2 envoyé au web service est faux. Lorsque je crée par exemple les paramètres à envoyer au web service, pour chaque ligne de tbl1, il y a tout le contenu de tbl2 qui est mis.
Vu les difficultés que j'ai à faire les modifications, je vais repartir à 0 pour la génération du code (sans gestion des erreurs et je modifierai ensuite la partie XAML pour utiliser des usercontrols) afin de réduire le risque d'avoir une génération de bugs au lieu d'une génération de code.

Avant de coder, je vais faire du pseudo-code XSLT et en même temps faire des traces papiers ce qui j'espère me permettra d'avoir une transformation qui fonctionnera mieux qu'actuellement et qui soit également plus simple.

Planing nouvelle orientation (commencé le 11mars):

- Tentative de correction des bugs de génération custom[] (non concluant) : 11.5h

- Génération code : redéveloppement
  - pseudo code + trace : 24h

- Coder le pseudo code : 18h
- Debug et essais : 24h

- Wizard : identifier les éléments à rendre paramétrable : 4h
- Gestion des tests : exploration de nunit : 4h
- Expression : essayer et regarder comment sont construits certains contrôles : 4h

Si le travail se passe selon l'estimation (j'ai un doute tout de même), je devrais avoir terminé ceci le dimanche 30 mars (les 2 semaines de vacances que j'avais planifiées sont remplacées par 48h de travail).

Total : 20h

Sujet du travail de diplôme :
WSDL & XAML

Etudiant :
Vionnet Olivier

Professeur responsable :
Schumacher Michael Ignaz
Russo David

Semaine : *21*
Woche : *14.03.2008*
27.03.2008

Décompte des heures hebdomadaires
Deklaration Wochenstunden

Tâche/ Aufgabe	Heures/ Stunden

Redéveloppement du code (pseudo code et trace) :

Je n'ai pas encore fini cette partie. J'ai terminé (sous réserve) la partie input et je travaille sur la partie output. Une fois terminée, j'aurai encore des tests de code supplémentaires à faire pour la partie tableau costum (pour être certain que ça passe à 2 niveaux).

J'ai passé beaucoup de temps sur le problème des tableaux de type costum et je ne suis pas arrivé à trouver une solution qui fonctionne quel que soit le nombre de tableau costum imbriqué (du style : Personne[] contenant Chien[] et ainsi de suite. J'ai des problèmes pour savoir pour le 1er niveau combien j'ai d'éléments dans le tableau (et ensuite pour chaque sous niveau) afin de pouvoir parcourir l'ensemble et mettre les bonnes valeurs au bon endroit, pour les enfants, si ce sont des types primitifs ou complexes, par rapport à leur profondeur, quel est le délimiteur de séparation qui doit être utilisé pour recréer un tableau (si c'est un simple tableau, le délimiteur est ;; , un tableau 2 dimension, le délimiteur est } ;;{ et ainsi de suite. De plus, il y a également des problèmes de déroulement de code, par exemple, un bout de traitement de code qui est effectué alors qu'une variable n'a pas encore été instanciée.

Vu que c'est un élément important et qui peut revenir régulièrement dans des web services, il faut tout de même pouvoir traiter ce genre de cas.
Comme je n'arrive pas en rappelant le template pour construire le code, je vais coder de manière spécifique dès qu'un tableau de costum est trouvé : il n'y a plus d'appel à d'autres templates : le traitement pour les enfants se fait à la suite.
Ca présente la limitation suivante : une fois rentré dans le type de tableau costum, il ne pourra avoir qu'un nombre limité de nœuds enfants pouvant être traités.

L'idéal serait de pouvoir traiter 2 niveaux de tableau costum :
custom[] (xxx, custom[] (xxx)) où xxx peut être prendre le type primitif, complex[], custom().

Ce n'est pas terrible comme solution mais à défaut de mieux, c'est toujours mieux que de ne pas supporter ce genre de types. Ca me permettra au moins de prendre en charge les cas « simples ». De plus, je ne pense pas qu'il y ait souvent beaucoup de tableaux customs imbriqués. Sans compté que ça devient très vite difficile pour remplir les données dans un textbox :

par exemple pour custom[] (string ; custom2[] (double[])), le tableau de double doit être entré dans le format suivant : {{};;{}};;{{};;{};;{}};;{{}}.

Exemple :
{{1;;3;;2};;{4;;8}};;{{3;;9;;0};;{2;;7};;{2;;9;;0}};;{{2;;8;;9}}
costum[] contient 3 éléments custom() :
1 : {{1;;3;;2};;{4;;8}}
2 : {{3;;9;;0};;{2;;7};;{2;;9;;0}}
3 : {{2;;8;;9}}

Pour l'entrée 1, custom2[] contient 2 éléments :
1 : {1;;3;;2}
2 : {4;;8}

Et à l'intérieur, on trouve le tableau de double dont les valeurs sont séparées par un double point-virgule.

Pour assurer mes arrières quant aux bugs, je développe en parallèle toute une série de web services pour tester et je teste également fréquemment les éléments considérés comme bon car avec la génération de code, une petite modification peut poser beaucoup de problèmes en cascade, donc, il faut que je puisse détecter les problèmes au plus tôt avant que ça ne se transforme en montagne.

Pour la création des web services, j'ai mis de côté VS 2005 et je développe maintenant avec VS 2008 en utilisant le template WCF Service Application pour rester sur la ligne WCF (composé d'une interface (ServiceContract) et des classes de données (DataCotnract) et une classe Service contenant le traitement des opérations). C'est pratique à utiliser, ça va assez vite à monter et par rapport à avant où je modifiais un web service au fur et à mesure des tests, je peux réutiliser facilement pour retester.

Vu que j'étais bloqué avec les tableaux custom, j'ai profité de faire quelques pauses sur cette partie en regardant comment faire pour utiliser des Usercontrols.

Pour les utiliser, il faut avoir une référence pour y avoir accès. Cette référence est faite dans l'élément Window :
```
xmlns:my="clr-namespace:WpfApplication4"
```
L'UserControl peut ensuite être appelé dans le code de la manière suivante :
```
<my:UserControl_final
x:Name="uc1"></my:UserControl_final>
```

Cela pose quelques problèmes pour utiliser ce système :

Il faut la référence du namespace de l'application pour pouvoir utiliser les UserControls : ce qui suit clr-namespace doit être obligatoirement dynamique (selon le choix de l'utilisateur défini dans les options).
Le problème, c'est que pour éviter d'avoir des informations parasites dans certains éléments XAML générés, il faut que je spécifie le namespace dans la balise stylesheet et dans la balise Window :
```
xmlns:my="clr-namespace:<Le namespace dynamique>"
```
Mais je ne peux pas accéder par la transformation à la balise stylesheet (et mettre un attribut dynamique).
Après divers essais, je suis arrivé à la solution suivante :
Dans le fichier de base, j'ai la ligne de code suivante :
```
xmlns:my="clr-namespace:replace"
```
Dans le code, lors de la génération, j'ouvre le fichier xslt, je le parcours entièrement et à chaque occurrence, je remplace le contenu de la ligne avec le bon namespace et je stocke le résultat dans un dossier temp.

Vu que je n'ai pas beaucoup à modifier, cette solution est acceptable. Ensuite, je peux utiliser le fichier écrit pour effectuer la transformation normale.

Le 2ème problème concerne les fichiers XAML et XAML.cs de chaque Usercontrol. Pour pouvoir les utiliser dans l'application finale, il faut que je les aie à disposition.
J'utilise un fichier de base XSLT et je regarde quels sont les types que l'application va rencontrer. Pour chaque type standard, je génère un fichier XAML et XAML.cs. Pour éviter les problèmes de namespace du code XAML par rapport à ceux du stylesheet, les balises du Usercontrol sont converties en > et < afin que le contenu ne soit pas interprété et une fois généré, je remplace chaque occurrence par < ou >.

les fichiers des Usercontrols sont stockés dans un dossier Usercontrols et le fichier csproj est mis à jour pour inclure les Usercontrols qui vont être utilisés.

Au niveau de la génération des fichiers de code, cela implique deux changements : au niveau du XAML, le code est simplifié. C'est juste une ligne par type trouvé pour poser l'élément.
Pour la partie cs, après avoir initialisé le composant, il faut mettre à jour chaque label d'information de chaque UserControl (par défaut, il n'y a que le type qui est présent de base). Il faut rajouter l'identifiant et le nom membre provenant du web service.

A noter que le problème de remise à zéro des TextBox après utilisation devrait être corrigé.

Je commence à perdre ma motivation…

Total : 53h

Sujet du travail de diplôme :
WSDL & XAML

Etudiant :
Vionnet Olivier

Professeur responsable :
Schumacher Michael Ignaz
Russo David

Semaine : *22*
Woche : *28.03.2008*
03.04.2008

Décompte des heures hebdomadaires
Deklaration Wochenstunden

Tâche/ Aufgabe — Heures/ Stunden

- Redéveloppement de la génération de code terminée.
  Choix d'une limitation pour les tableaux de type custom :

Current	**1er niveau**	**2ème niveau**	**3ème niveau**
Custom[]	Primitif		
Custom[]	Complex		
Custom[]	Custom	Primitif	
Custom[]	Custom	Complexe	
Custom[]	Custom	Custom	Primtif
Custom[]	Custom	Custom	Complexe
Custom[]	Custom	Custom[]	Primitif
Custom[]	Custom	Custom[]	Complexe
Custom[]	Custom[]	Primitif	
Custom[]	Custom[]	Complexe	
Custom[]	Custom[]	Custom	Primitif
Custom[]	Custom[]	Custom	Complexe

- Création d'une 30aines de services WCF pour tester la génération de code.

- Microsoft Expression 2 : installé la beta fév. 08 pour connaître comment Microsoft construit des grids jolis (pour rappel, trouver quelque chose de comparable au datagrid en .Net 2.0).
  Il est un peu vide ce programme… et surtout, j'espérai trouver des templates ou des wizards pour produire plus facilement des éléments au niveau graphique.
  Au final, je vais partir sur la même structure que trouvé dans les exemples MSDN pour la classe grid : mettre des éléments XAML comme un Border pour faire des contours dans les cellules par exemple.

- Gestion des tests : installé et fait quelques essais avec NUnit 2.4.7

- Réfléchir sur la partie de génération par le biais d'un wizard : qu'est ce qu'il faudrait faire.

- Une séance jeudi

Suite à la séance du jeudi 3 avril, voici le programme :

- Tester des web services externes et corriger les bugs :
  Après avoir pris du temps pour comprendre les erreurs dans de volumineux fichiers de code, j'ai découvert 2 problèmes :

  Problème d'importation avec svcutil : certains services ont un binding HTTPPost ou HTTPGet. Svcutil n'arrive pas à générer ces parties (par contre, le SOAP passe).
  Je me retrouve avec un fichier de configuration qui prend uniquement en charge la partie SOAP mais rien pour le HTTP (par contre, mon application génère tout le code également pour le HTTP). Ca provoque des erreurs lors de la création de l'application : par défaut, je me retrouve qu'avec le binding SOAP pour traiter les cas HTTP et les paramètres que je tente d'envoyer sont des strings (le traitement de récupération des types est juste) mais la partie SOAP attend des paramètres qui sont d'un autre type. Au final, je me retrouve avec des paramètres strings et je ne sais pas en quoi il faudrait les convertir.
  La solution (et encore plus vu que svcutil n'arrive pas générer les parties HTTP), c'est de ne pas prendre en compte les bindings HTTP.

  Le 2ème problème est plus ennuyeux : Je suis encore tombé sur un type de données auquel je n'avais pas pensé : les énumérations !
  C'est ce qui fait planter la génération de code XAML. A noter que l'on peut avoir des Enumérations ou encore des tableaux d'énumération. Bref, c'est un gros morceau qui va prendre du temps à implémenter (et vu que l'on approche de la fin, ça va être short).

  Ce que je vais faire :
  Corriger le code de récupération des types pour prendre en charge les types Enum et Enum[]

  Etant donné qu'un Enum peut prendre n'importe quel nom, il faut que je puisse faire la différence entre une énumération et un type custom. Pour le fichier xml data, si je rencontre un Enum, il aura un attribut supplémentaire specialClass noté Enum ou Enum[].
  <type id="2" name="Enum1" nameMember="En1" specialClass="Enum" />

  Au niveau du template pour le XAML, les changements devraient être minimes, donc je vais implémenter Enum et Enum[] pour prendre les bons UserControl.

  En ce qui concerne le template du code behind, ça va être long à faire les modifications et tout retester (pour éviter qu'à un endroit, un Enum soit considéré comme un type custom). Je pense laisser de côté les tableaux d'Enum et n'implémenter que les Enum.

  Le plus simple est de se baser sur le model actuel : on peut avoir lors du traitement le choix entre : primitif, complexe, custom et custom[]. Je vais mettre pour chaque cas le choix Enum.

  Temps estimé : 24h (si pas de problèmes, actuellement, j'en suis à 10h et j'ai commencé à modifier le template de code behind).

- Modification de l'utilisation des UserControls : actuellement, que ce soit pour de l'input ou de l'output, il n'y a pas de différences pour chaque type de UserControl (cela sera problématique pour la partie wizard). Il faut modifier les templates et la création des fichiers UserControls pour

qu'il y ait pour chaque type un UserControl pour l'input et l'output.

Temps estimé : 12h

- Création de 1 ou 2 exemples concrets de web service par rapport à l'informatique de gestion afin de mieux pouvoir expliquer le fonctionnement du client généré et être moins abstrait que les services WCF de test que j'ai développés.

  Temps estimé : 12h

- Développer la partie test :
  mettre en place le choix de l'opération à tester
  préparer le projet d'application de test : créer une classe qui utilise le web service (une méthode pour l'appel et de quoi pouvoir accéder aux variables).
  Créer une classe de test (classe vierge). L'utilisateur aura un grand TextBlock pour rentrer le code des tests
  Construction du projet de test et lancement de l'application en utilisant NUnit avec la console.
  Traiter le fichier de résultat de NUnit pour les tests et afficher le résultat dans un Grid (creuser le datacontext pour mettre les données directement dans le Grid)

  Ca fait un moment que je n'ai pas retouché cette partie, donc il y aura énormément de choses à implémenter.

  Temps estimé : 48h

  A ce stade, les éléments ci-dessus devraient être terminés pour début mai.

- Wizard :
  Choisir la ou les opérations que l'on souhaite traiter
  Choix de l'UserControl à utiliser pour chaque type (différent entre l'input et l'outpu)
  Traitement de web service avec state (envoi de données x fois puis demande d'un retour de la part du web service)

  Temps estimé : le reste du temps… à voir

- Pour la prochaine séance, ça dépend de comment j'ai avancé. J'en saurai plus à la mi-avril (et j'enverrai un mail).

Total : 28h

Sujet du travail de diplôme :
WSDL & XAML

Etudiant :
Vionnet Olivier

Professeur responsable :
Schumacher Michael Ignaz
Russo David

Semaine : *23*
Woche : *04.04.2008*
10.04.2008

Décompte des heures hebdomadaires
Deklaration Wochenstunden

Tâche/ Aufgabe	Heures/ Stunden

- Correction du problème si le Web service contient un binding http post ou get. Ils ne sont plus pris en compte (vu que svcutil ne traite pas ces cas)

- Implémenté les types Enum et Enum[] (ça a été moins difficile que prévu, donc j'ai profité d'inclure les tableaux d'énumération)

- Correction d'un bug dans le template parametre/type, mode inputArrayCust qui sert à la création d'objets pour l'envoi au web service : une condition de test pour la partie contenant un tableau pour coder une boucle for prenait que le premier élément enfant trouvé pour déterminer quel compteur devait être utilisé pour la limite maximum d'itération.
  Maintenant, un count est fait sur tous les éléments enfants pour les types primitifs et complexes. Si ça correspond au nombre de nœuds enfants, l'id du compteur utilisé sera le premier enfant. Dans le cas contraire, ça prend l'id de l'enfant de l'enfant (car l'un des enfants est un type costum).

- Correction d'un bug dans la partie option : si les dossiers par défaut n'existent pas, l'application plante. Ajouté une méthode d'initialisation lors de l'appel de la fenêtre Window1 (point d'entrée dans l'application). Test si les 3 dossiers existent (par rapport aux informations contenues dans ApplicationSettings.settings et si ce n'est pas le cas, crée les dossiers manquants.

- Au final, les tests que j'ai créés pour tester l'application (si le retour n'est pas précisé, c'est qu'il est identique aux paramètres envoyés au web service :

string, int retour : int
string[], int[] retour : int[]
string, int[] retour string
custom(string,int)
custom(string,int[])
custom(string,custom2(string,int32[]))

custom(string[], custom2(string, int32[]))
custom(string[], boolean[], custom2(string[], double[]))
custom[] (string, int32)
custom[](string, int32[])
custom[](string, custom2(string, double))
custom(string, custom2[](string, double))
custom(string, custom2[](string[], double))
custom(string, double, custom2(string, double, custom3(string, double)))
custom(string[], double, custom2(string[], double, custom3(string[], double)))
custom[](string, custom2[](double))
custom[](string, double, custom2[](string, double))
custom[](string[], double, custom2[](string, double))
custom[](string, custom2[](double[]))
custom[](custom2[](double))
custom[](string, custom2[](double[], string[]))
custom[](string, custom2[](double[], Boolean, string[]))
custom[](string, custom2[](custom3 (string)))
custom[](string, custom2[](custom3(string, int, Boolean)))
custom[](string, custom2[](custom3(string[])))
void retour : string
custom[](string, custom2(string, custom3(int)))
custom[](string, custom2(string, custom3(int[])))
custom[](string, custom2(string[], custom3(int[])))
custom[](string, custom2(string[], custom3[](string)))
custom[](string, custom2(string[], custom3[](string[])))
custom(string, custom2 (string, custom3[](string, double)))
custom(string, custom2(string, custom3[](string[], double[])))
string, enum retour : enum
custom(string, enum)
custom(string, enum, custom2(string, enum2))
custom[](string, enum)
custom[](string, custom2(string, enum))
custom[](string, custom2(string, custom3(string, enum)))
custom[](string, custom2(string, custom3[](string, enum)))
custom[](string, custom2[](string, enum))
custom[](string, custom2[](string, custom3(string, enum)))
string, enum[] retour : enum
custom(string, enum[])
custom(string, enum[], custom2(string, enum[]))
custom[](string, enum[])
custom[](string, custom2(string, enum[]))
custom[](string, custom2(string, custom3(string, enum[])))
custom[](string, custom2(string, custom3[](string, enum[])))
custom[](string, custom2[](string, enum[]))
custom[](string, custom2[](string, custom3(string, enum)))

- Sur une durée aussi longue, il était à prévoir qu'il y aurait un problème informatique : c'est chose faite.
  Bloc d'alimentation grillé, ventilateur de la carte graphique out et en réparation…
  Cela aurait pu être pire… je croise les doigts pour que ce soit les seuls problèmes de ce type rencontrés durant ce projet.

Planning prévu pour la semaine prochaine :

- Modification de l'utilisation des UserControls :
  J'ai déjà commencé à regarder le code pour voir comment je vais changer la création des UserControls pour avoir un UC par type et par input/ouput pour la génération automatique (au final, ça ne fera pas de différence dans l'utilisation mais ça sera déjà un problème plus ou moins résolu

lorsque j'attaquerai la partie Wizard). Il semblerait que ce soit plus simple à modifier que ce que je pensai en planifiant. A suivre…
Temps estimé : 12h

- Création de 1 ou 2 exemples concrets de web service par rapport à l'informatique de gestion afin de mieux pouvoir expliquer le fonctionnement du client généré et être moins abstrait que les services WCF de test que j'ai développé.

  Je pense faire un service simple de gestion de vente de livres avec une petite base de données contenant les tables Article, Client, commande et détail de commande.
  Pour illustrer, je vais faire les OperationContracts :

  - Lister les clients (retourne un tableau de client)
  - Récupérer les informations d'un client en passant un identifiant
  - Lister les articles (retourne un tableau d'Article)
  - Récupérer un article en passant le nom de l'article et le nom de l'auteur
  - Récupérer une commande avec un identifiant
  - Récupérer toutes les commandes pour un client en passant sont identifiant (ça donnerait comme retour un objet Client qui contient un tableau de commande et ce tableau de commande contient pour chaque commande un tableau de détailcommande ce qui me permettra de montrer comment sont traités les données tableaux.

  Je vais voir également pour créer quelques opérations qui passent en paramètre autre chose que du string ou un int :
  Notamment la création d'un client (en passant l'objet client) et la création d'un article. Pour illustrer les tableaux en input, je ferai une version de ces 2 opérations simple (1 client, 1 article) et une version tableau (pourra ajouter x clients ou x articles en même temps).

  La gestion des erreurs sera sommaire et les appels à la bd et la création des objets sera fait directement dans la classe Service qui contient le traitement des opérations du web service.
  Temps estimé : 12h

Total : 23h

Sujet du travail de diplôme :
WSDL & XAML

Etudiant :
Vionnet Olivier

Professeur responsable :
Schumacher Michael Ignaz
Russo David

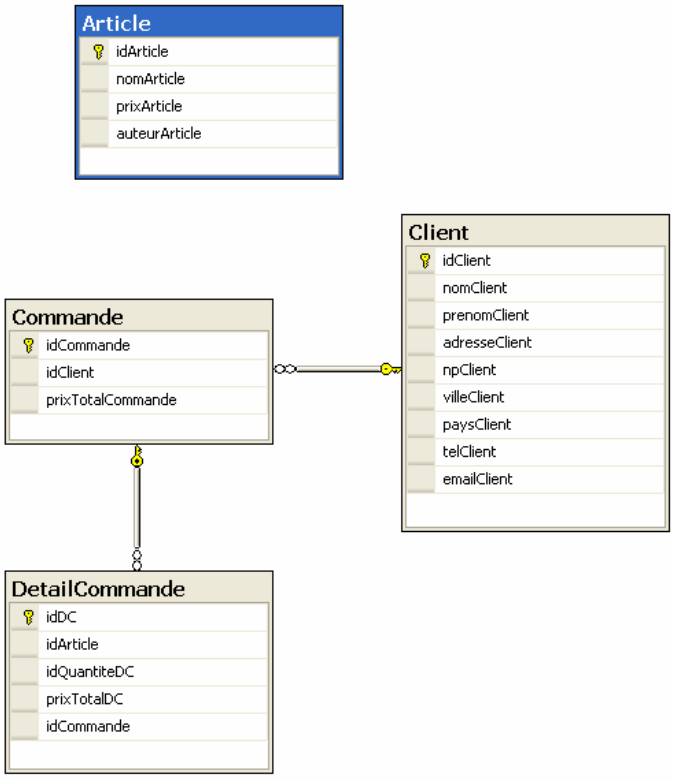
Semaine : *24*
Woche : *11.04.2008*
17.04.2008

Décompte des heures hebdomadaires
Deklaration Wochenstunden

Tâche/ Aufgabe	Heures/ Stunden

- UserControls : le code a été modifié pour la génération automatique. Maintenant, il y a pour chaque type de données un UserControl différent pour l'input et l'output.

- Création d'un exemple de web service exemple : une mini gestion de client vente de livre.
  Le diagramme de la base de données :



  J'ai défini les opérations du web service suivantes :

Client[] listClient();
Afficher tous les clients présents

Client getOneWithIdClient(int id);
Afficher un client en utilisant son identifiant
int insertNewClient(ClientInsert cli);
Ajouter un nouveau client
int insertNewClients(ClientInsert[] cli);
Ajouter des nouveaux clients (un tableau contenant plusieurs objets de type client est passé en paramètre)

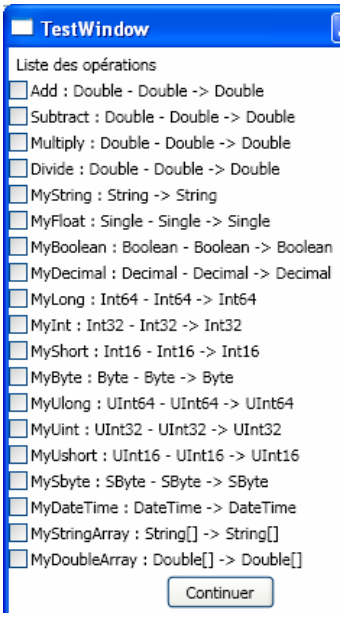
Article[] listArticle();
Afficher tous les articles présents
Article getOneWithNomAndAuteurArticle(string nomArticle, string auteurArticle);
Afficher un article en utilisant le titre du livre et l'auteur
int insertNewArticle(ArticleInsert artI);
Ajouter un nouvel article, retourne le nombre d'enregistrements effectués dans la base de donnée
int insertNewArticles(ArticleInsert[] artI);
Ajouter plusieurs articles en même temps (tableau d'articles)

Commande getCommandeByIdCommande(int idCommande);
Obtenir les informations sur une commande précise avec le contenu de la commande (le client a qui appartient la commande n'est qu'un int).

ClientCommande getClientCommandeById(int idClient);
Permet de récupérer toutes les commandes d'un client. Cette méthode renvoi les informations du client, la liste de toutes ses commandes ainsi que le détail de chacune des commandes.

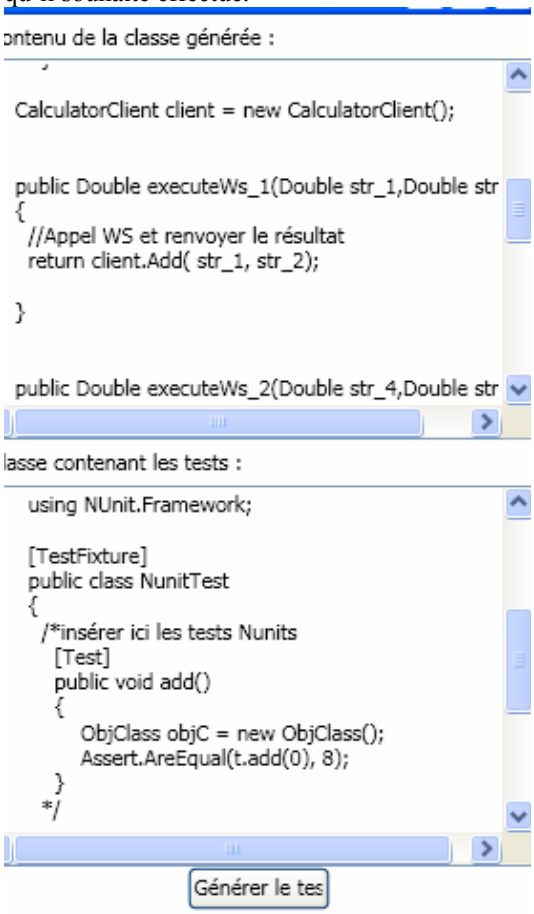
La gestion des erreurs est sommaire et les appels à la bd et la création des objets sera fait directement dans la classe Service qui contient le traitement des opérations du web service.

- Partie tests Nunit de l'application, je suis en train de terminer d'implémenter le code pour lancer les tests. Il y aura plusieurs étapes :
  1 : Lancer l'application pour les tests (crée les fichiers nécessaires pour connaître les opérations possibles)
  2 : Choix des opérations que l'on veut tester : J'ai opté pour utiliser des CheckBox au lieu de RadioButton. Maintenant, l'utilisateur peut choisir plusieurs opérations en même temps à traiter.



3 : Création des fichiers de codes : la classe objet qui contient les appels

au web service et la classe de test (qui contient juste un exemple en commentaire et que l'utilisateur devra remplir à la main pour les tests qu'il souhaite effectué.



4 : Générer le projet final et lancer les tests Nunit (en utilisant la console)
5 : Afficher le résultat par rapport au fichier de résultat xml généré par Nunit (TODO)

Ce qui est prévu du 18 au 24 avril : Terminer la partie Nunit test :

- Rendre certaines variables ou assembly paramétrables dans le menu des options.
- Implémenter l'affichage des résultats dans un grid
- Tester les tests : voir comment ça se comporte avec différentes opérations en même temps.
- Regarder les asserts possibles et faire un petit exemple de tests avec un web service de type calculatrice (+,-,*,/)
- Contrôler que l'affichage soit clair entre les différents tests.
  Théoriquement, le 24 avril, cette partie devrait être terminée

Total : 24h

Sujet du travail de diplôme :
WSDL & XAML

Etudiant :
Vionnet Olivier

Professeur responsable :
Schumacher Michael Ignaz
Russo David

Semaine : *25*
Woche : *18.04.2008*
24.04.2008

Décompte des heures hebdomadaires
Deklaration Wochenstunden

Tâche/ Aufgabe	Heures/ Stunden

- Création de la partie affichage des tests Nunits : j'ai utilisé l'élément ListView et GridView. C'est un peu étrange : ListView contient un GridView qui contient les données.

  Je me suis basé sur 2 exemples pour construire mon affichage :
  http://www.codeproject.com/KB/miscctrl/GridView_WPF.aspx
  http://msdn2.microsoft.com/fr-fr/library/ms771778.aspx

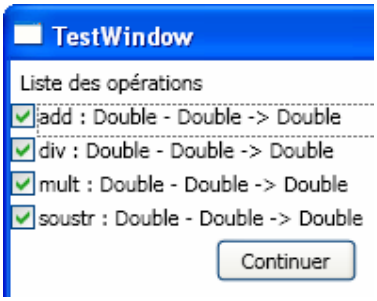
  Je n'ai pas pu utiliser le fichier de résultat de Nunit tel quel (il y a un problème de binding à cause de nœuds portant le même nom qui sont imbriqués).
  J'ai fait au final un fichier xml temporaire pour ne garder que les résultats importants et ça fonctionne.
  A noter que les colonnes peuvent être réarrangées par défaut.

- Chaque méthode de test représente une ligne. Chacune des méthodes peut contenir plusieurs asserts (mais Nunit ne renvoie qu'une ligne pour tous les asserts d'une méthode). Dès qu'une assert est fausse, les tests des asserts suivants ne sont pas effectués.

- Un exemple avec une calculatrice :

Opérations disponibles sur le web service :
double add(double a1, double a2);
double div(double a1, double a2);
double mult(double a1, double a2);
double soustr(double a1, double a2);

Choix des opérations à tester :



Code généré automatiquement pour la classe Object (s'occupe de faire les appels au web service)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.ServiceModel;
using System.Runtime.Serialization;

namespace WPFHelloCCC
{
    class NunitObject
     {

       static void Main(string[] args)
        {
        }

   Service1Client client = new Service1Client();

    public Double add(Double str_1,Double str_2)
   {
     //Appel WS et renvoyer le résultat
     return client.add( str_1, str_2);
   }

    public Double div(Double str_4,Double str_5)
   {
     //Appel WS et renvoyer le résultat
     return client.div( str_4, str_5);
   }

   public Double mult(Double str_7,Double str_8)
   {
     //Appel WS et renvoyer le résultat
     return client.mult( str_7, str_8);
   }

   public Double soustr(Double str_10,Double str_11)
   {
     //Appel WS et renvoyer le résultat
     return client.soustr( str_10, str_11);
   }
     }
   }
```

Code des tests Nunits (le squelette de la classe est généré automatiquement. L'utilisateur doit juste saisir à la main les éléments à coder). Pour cet exemple, j'ai également utilisé IsInstanceOfType qui permet de savoir si le type est bien celui attendu ainsi que Greater (la valeur attendue doit être plus grand). La première

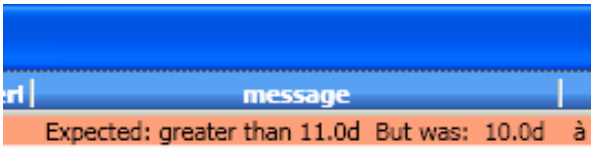
partie de l'assert contient la valeur attendue et la 2ème la valeur à tester :

```
[Test]
public void test_add()
{
        NunitObject objC = new NunitObject();
        Assert.IsInstanceOfType(Type.GetType("System.Double"),
(objC.add(4, 2)));
        Assert.AreEqual(6, objC.add(4, 2));
        Assert.AreEqual(10, objC.add(8, 2));
        Assert.Greater(20, objC.add(9, 2));
}

[Test]
public void test_div()
{
        NunitObject objC = new NunitObject();
        Assert.AreEqual(2, objC.div(4, 2));
        Assert.IsNaN(objC.div(8, 0));
}

[Test]
public void test_mult()
{
        NunitObject objC = new NunitObject();
        Assert.AreEqual(8, objC.mult(4, 2));
        Assert.AreEqual(16, objC.mult(8, 2));
}

[Test]
public void test_soustr()
{
        NunitObject objC = new NunitObject();
        Assert.AreEqual(2, objC.soustr(4, 2));
        Assert.AreEqual(6, objC.soustr(8, 2));
        Assert.AreEqual(1, objC.soustr(8, 7));
}
```

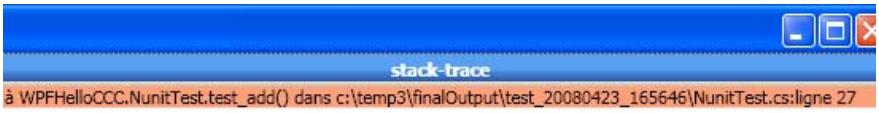
Résultat affiché après l'exécution des tests :

Cas où une assert contient une erreur : Dans test_add(), Assert.Greater(20, objC.add(9, 2)); est remplacé par Assert.Greater(10, objC.add(9, 2));

Les tests ont été réalisés jusqu'à ce qu'une erreur est arrivée.



Le message d'erreur : la valeur retournée 4 et celle que l'on attendait 2.



Emplacement du test qui a généré une erreur.

- Problème étrange :
  Il y a quelque chose de bizarre : En définissant 4 méthodes de tests contenant chacun 6 asserts, je peux exécuter 2x les mêmes tests (Cliquer sur le bouton en utilisant les mêmes codes de tests) sans problème particulier. Le résultat est affiché rapidement.
  A la 3ème, il y a un blocage au milieu de l'envoi des données (dans la fenêtre console du web service qui affiche ce qu'il reçoit, je n'ai que la moitié des données qui arrive et rien de plus).
  Si je restart le web service et que je relance le test (toujours sans avoir relancé une nouvelle génération), ça repasse de nouveau 2x.
  Si je ne mets que 2 méthodes de tests, la 3ème exécution passe sans problème.

  J'ai également testé en utilisant Nunit en mode graphique en chargeant l'exécutable créé par mon application). J'ai également le même problème. Il bloque lors de la 3ème exécution des tests.

  J'obtiens le message d'erreur suivant :
  WPFHelloCCC.NunitTest.multiply:
  System.TimeoutException : Le canal de demande a dépassé le délai imparti en attendant une réponse après 00:01:00. Augmentez la valeur du délai d'expiration transmise à l'appel à Request ou augmentez la valeur SendTimeout sur Binding. Le temps alloué à cette opération fait peut-être partie d'un délai d'attente plus long.
  ----> System.TimeoutException : La demande HTTP à « http://localhost:8000/ServiceModelSamples/service » a dépassé le délai d'expiration alloué de 00:01:00. Le temps alloué à cette opération peut avoir été une partie d'un délai d'expiration plus long.
  ----> System.Net.WebException : Le délai d'attente de l'opération a expiré.

  Pour le moment, je n'ai pas d'idées précises de ce qui pourrait être à l'origine de ce problème. Je pencherai pour le pc qui est vieillissant (et à de la peine avec tous les programmes ouverts en même temps) et le fait que le web service soit exécuté en débug sur la même machine. Ou alors Visual Studio qui aurait des ratés ? (il n'est pas rare que lors d'une exécution, j'ai tout d'un coup tout le layout des fenêtres qui change).

- Correction de bug dans la partie s'occupant d'afficher les erreurs :
  Maintenant, les erreurs MSBUILD sont prises en compte et affichées dans le rapport d'erreur.

- Modification du code concernant la réécriture de fichiers (pour changer des portions de code par exemple).
  Maintenant, j'ai une classe MyTools statique qui contient une méthode unique. Prends également en compte si le fichier doit être écrasé ou sauvé avec un .tmp

- Options assemblies pour Nunit :
  Les assemblies peuvent être définies maintenant dans les options pour les fichiers csproj, ObjetClasse et TestClasse.
  Modifié les fichiers de génération xml des données pour prendre en compte ce nouveau cas de figure ainsi que les templates XSLT pour la génération du code pour les classes utilisées pour le test.

  Pour plus de clarté, j'ai fait un tabItem pour les assemblies du client généré automatiquement ainsi qu'un autre pour les assemblies destinées aux tests Nunit.

- Le bouton pour lancer la génération de l'application pour les tests est maintenant désactivé après avoir cliqué dessus et réactivé une fois la fenêtre de résultat affichée.

Suite du travail : réaliser la partie wizard.

- Comme pour la partie des tests, offrir le choix des opérations que l'on souhaite traiter.

- Identifier pour chaque type de données quels sont les contrôles pouvant être utilisés. Les contrôles seront mis dans des UserControls et il y aura pour un type un UC pour l'input et un autre pour l'output.

  Mais là où ça coince : les types (données) peuvent être membres d'un objet ou d'un tableau d'objets. Ca complique tout de suite la façon de proposer les contrôles pour un type donné : il faut connaître ce qu'il y a au-dessus pour pouvoir ensuite construire correctement l'UserControl avec le nombre de contrôles nécessaires. Un exemple :

  Si le type Boolean n'a pas de parent, je peux poser sans problème un contrôle CheckBox dans l'UserControl.

  Si ce type est une donnée membre d'un objet et que cet objet est dans un tableau, je ne peux pas simplement mettre un CheckBox (sinon, tous les objets auront la même valeur pour le membre de type Boolean).
  Il faut que je construise l'UserControl avec le même nombre de CheckBox qu'il y a d'objets dans le tableau (comme ça, chaque objet pourra avoir le membre de type Boolean différent).

  Et ça se complique encore s'il y a un tableau d'objets dans un tableau d'objets…

  Donc, je ne peux pas définir un UserControl spécial pour certains cas qui passera partout sans que l'utilisateur doive le modifier.
  De plus, il faut également que les éléments que je souhaite intégrer ne nécessite pas de modifier en profondeur les templates de génération de code (sinon, ça va prendre beaucoup trop de temps).

  Processus pour la génération wizard (input uniquement) :
  Pour commencer, il me semble que le plus simple est de s'occuper dans un premier temps de la partie input (l'output sera traité pour le moment comme dans la partie générée automatiquement).

  Afficher les opérations à l'utilisateur. Il sélectionne celui ou ceux qu'il souhaite utiliser.

  Préparer les templates des UserControls par défaut (en tenant compte des options, namespace) et écrire les fichiers. Il me semble qu'il est plus simple de tout écrire d'un coup et ensuite, on garde que ceux qui ont été retenus :

Types primitifs :
- Pour tous les types :
UC contenant un TextBox (idem qu'actuellement pour la partie génération automatique)

- Boolean :
UC avec un CheckBox

- Enum :
UC avec un ListBox (vide)

Types complexes :
- Pour tous les types [] :
UC contenant un TextBox (idem qu'actuellement pour la partie génération automatique avec un délimiteur).

- Boolean[] :
UC avec un TextBox pour la taille du tableau. Une fois le chiffre rentré, affichage des contrôles CheckBox.

- Enum[] :
UC avec un TextBox pour la taille du tableau. Une fois le chiffre rentré, affichage des contrôles ListBox (vide).

Remarque importante :
S'il y a plusieurs types identiques (2 strings), chacun aura son propre UserControl (car il se peut très bien qu'un soit dans un tableau d'objet alors que l'autre non).

Pour chaque opération, afficher toute la structure des données input et output.
Pour chaque type trouvé, afficher les informations du fichier dataFile.xml (id, type, nom du membre).

Il faut également mettre pour chaque type les différents UserControls disponibles (une liste) qui va permettre de sélectionner l'UserControl qui sera utilisé.

Un bouton va permettre de lancer une fenêtre pour afficher le code XAML et CS du contrôle sélectionné pour pouvoir modifier le code.

Comment ça se passe lorsque le type est dans un tableau d'objets ?
C'est là que l'utilisateur devra intervenir au niveau du code si l'UserControl utilisé n'est pas le standard.
Il devra mettre autant de contrôles que la contenance du tableau.

Une fois toutes les opérations effectuées, créer l'application (attention aux éléments mis dans les UserControls et que les assemblies ne sont pas présents dans le csprojet par exemple).

Il y aura certainement des problèmes qui vont surgir et auxquels je n'ai pas pensés, mais il me semble que mon idée tienne la route.

Total : 24h

Sujet du travail de diplôme :
WSDL & XAML

Etudiant :
Vionnet Olivier

Professeur responsable :
Schumacher Michael Ignaz
Russo David

Semaine : *26*
Woche : *25.04.2008*
1.05.2008

Décompte des heures hebdomadaires
Deklaration Wochenstunden

Tâche/ Aufgabe	Heures/ Stunden

- Commencé la partie wizard :
  Choix des opérations à traiter OK

- Pour la partie Input, mis en place la génération des UserControls standard + Boolean (non testé).
  Il y a un UserControl pour chaque donnée (s'il y a 2 paramètres strings, il y aura 2 UserControls différents).

- Mise en place de la sélection des UserControls à utiliser :
  Pour chaque donnée, une liste déroulante avec le choix des UserControls (en général, il n'y a que celui par défaut).

  En sélectionnant une liste déroulante, on affiche le nom du dernier template sélectionné et un bouton pour l'éditer (ouvre une nouvelle fenêtre. Note : j'étais parti dans un premier temps en affichant au-dessous l'édition mais ça faisait trop chargé ensuite).

  Dans cette nouvelle fenêtre, 2 TextBox (xaml, cs) pour éditer le contenu :
  Par exemple pour le cas des Enums, mettre dans les ComboBoxItems les différentes valeurs de l'Enum.

  ```
  <StackPanel><Label Height="28" Name="ucLabel_info"
  VerticalAlignment="Top" HorizontalAlignment="Left"
  MinWidth="120">Enum1</Label><ComboBox Height="23"
  Name="ucComboBox_saisie" VerticalAlignment="Top">
  <ComboBoxItem>a1</ComboBoxItem>
  <ComboBoxItem>a2</ComboBoxItem>
  <ComboBoxItem>a3</ComboBoxItem>
  </ComboBox><Label Height="28" Name="ucLabel_error"
  VerticalAlignment="Top" HorizontalAlignment="Left"
  MinWidth="120" Visibility="Collapsed" Foreground="Red"
  /></StackPanel>
  ```

- Je modifie au fur et à mesure les templates XSLT :
  Récupérer les données saisies par l'utilisateur. Avec le système actuel, la récupération des données saisies est écrite comme suit :
  uc_1.ucTextBox_saisie.Text ;

Ca va très bien dans la génération automatique car tous les UserControls sont identiques (c'est un TextBox qui reçoit la saisie).
Pour la partie wizard, ça ne fonctionnera pas car il n'y a pas 100% de chances que ce soit un TextBox qui s'occupe d'accueillir la saisie.
Cette partie a été modifiée pour faire en sorte que l'UserControl aie une méthode standard qui permette de récupérer la valeur saisie (le nom ne changera pas mais le contenu pour traiter la récupération oui).

- **Templates implémentés :**
  UC standard (idem génération automatique)
  UC Boolean (TextBox remplacé par un CheckBox)
  UC Enum (TextBox remplacé par un ComboBox).
  Nécessite à l'utilisateur de rentrer manuellement les ComboBoxItems (code XAML) ou d'utiliser le TextBox pour rentrer une série de valeurs séparées par des points virgules.

- Problèmes rencontrés :
  perdu un peu de temps à comprendre pourquoi lorsque je sauvais le contenu d'un TextBox, je n'avais qu'une partie du texte sauvé (problème de buffer. Par défaut, 1024. J'ai mis à la place la taille du contenu du TextBox).
  Tourné un peu en rond jusqu'à ce que j'aie compris comment fonctionne les ListBoxs pour récupérer le contenu sélectionné (un ToString() par défaut donne le contenu mais également le type…)

Suite du travail :

- S'occuper des types complexes :
  Avec le processus que j'ai mis en place, il y a un problème : je génère des fichiers avant de demander des informations à l'utilisateur. Ca va bien pour les types simples mais pour les complexes, c'est un problème : si l'utilisateur choisi d'afficher des CheckBoxs ou des ComboBoxs, il faut les coder et leur nombre est déterminé par la taille du tableau (information que je n'ai pas lorsque je génère). Demander cette information lors de la génération des fichiers ne me semble pas une bonne idée (il est préférable que l'utilisateur puisse avoir la vue d'ensemble des paramètres de l'opération pour déterminer ce qu'il veut).
  Donc, il faut que je fasse de la modification de code lorsqu'il va choisir un template spéciale.
  Voici ce que je pense faire :
  Le code des fichiers que je vais générer va contenir du code XSLT qui ne sera pas transformé lors de la première opération de transformation.

  Lorsque l'utilisateur va choisir le template qu'il souhaite utiliser, j'ouvre comme pour les types simples une nouvelle fenêtre et j'affiche le code des 2 fichiers.
  Je rajoute des champs de saisie supplémentaire (la taille du tableau par exemple). L'utilisateur rentre les informations, valide et l'application va s'occuper de refaire une transformation pour implémenter le code nécessaire pour arriver au final dans le fichier XAML à avoir le bon nombre de contrôles et dans le fichier CS le code pour traiter les saisies et les méthodes pour interroger l'UserControl.

  L'utilisateur peut également directement modifier les fichiers en supprimant le code XSLT et en modifiant les codes pour arriver au même résultat.

Total : 24h

Sujet du travail de diplôme :
WSDL & XAML

Etudiant :
Vionnet Olivier

Professeur responsable :
Schumacher Michael Ignaz
Russo David

Semaine : *27*
Woche : *2.05.2008*
8.05.2008

Décompte des heures hebdomadaires
Deklaration Wochenstunden

Tâche/ Aufgabe	Heures/ Stunden
• Séance	
• En train de travailler la partie wizard pour implémenter les types qui peuvent utiliser d'autres contrôles que le TextBox par défaut. Je rencontre pas mal de difficulté pour mettre en place les éléments, identifier et corriger les bugs (dont une bonne partie survienne lors de la 2ème transformation XSLT). **Taille des tableaux :** J'avais pensé dans un premier temps gérer la taille des tableaux dans la fenêtre permettant de sélectionner les UserControls à utiliser mais je suis resté bloqué sur la façon de faire pour saisir les tailles des sous-tableaux (les tailles pouvant être différentes, cela implique d'avoir un nombre différent de TextBox pour saisir les tailles et comme la majorité du contenu affiché est créé en C#, c'est très difficile de s'y retrouver et de récupérer des valeurs). J'ai finalement pris la décision de traiter ceci dans la partie édition des UserControls spéciaux (enum et boolean) pour éviter de rester coincer sur ce problème. Mais ça présente un inconvénient : l'utilisateur est responsable de mettre les bonnes tailles du (ou des) tableau(x) qui contiennent le type pour chaque UserControl spécial.	
Suite du travail :	
• Terminer la partie wizard dans le courant de la semaine prochaine	
Total :	33h

Sujet du travail de diplôme :
WSDL & XAML

Etudiant :
Vionnet Olivier

Professeur responsable :
Schumacher Michael Ignaz
Russo David

Semaine : *25*
Woche : *9.05.2008*
15.05.2008

Décompte des heures hebdomadaires
Deklaration Wochenstunden

Tâche/ Aufgabe	Heures/ Stunden

- Partie wizard terminée et début du refactoring

- Ce n'est pas parfait et comme j'aurai voulu que ce soit (voir ci-dessous le point : modification pour une version 2). Après avoir fait une série de tests, tout semble fonctionner correctement.

  Au final, on peut utiliser les UserControls standards (qui ne nécessitent aucune édition).
  Pour les types Boolean, Boolean[], Enum et Enum[], il y a la possibilité d'utiliser un UserControl spécial. Pour les Booleans, il est composé de CheckBox et pour les Enums de ComboBox.
  A noter que le contenu des ComboBox est initialisé lors de l'exécution de l'application générée.
  Ces UCs spéciaux doivent être édités (et sauvés) pour la partie Input : rentrer les tailles des tableaux d'objets dont ils sont membres (ceci afin d'afficher le bon nombre de contrôles). Lorsque toutes les informations sont saisies, la transformation XSLT de 2ème niveau est faite automatiquement et le contenu des TextBox qui affichent le code est mis à jour. Le bouton sauver sauve le contenu des TextBoxs et ferme la fenêtre.

- Correction de 2 bugs :
  UCs spéciaux Ouput : contrôles générés lors du runtime de l'application générées
  Pour la partie Output des UCs spéciaux, il n'y a rien à faire. Le processus a été modifié et maintenant, les contrôles sont affichés lorsque l'application générée reçoit la réponse du web service (Donc, au démarrage de l'application générée, le contenu de l'output pour les UCs spéciaux est vide. C'est d'après les données que sera construit l'affichage).

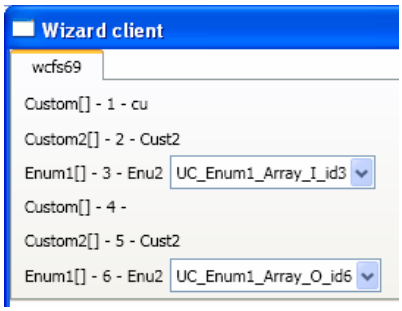
- Editer une 2ème fois un UC spécial :
  Lorsque l'on édite un UC spécial pour mettre la taille des tableaux, le code de transformation est écrasé et il ne reste plus que le code final servant à construire l'UC. Si l'on édite une 2ème fois le contrôle, il y aura une erreur.
  Je contrôle maintenant s'il y a la présence de code XSLT. S'il y a quelque chose, la transformation pourra être faite. Si elle a déjà été faite,

aucun traitement ne sera possible (par contre, le code pourra toujours être édité à la main).

Modifications pour une version 2 :
Si le projet doit être continué après la remise, voici les points qu'il faudrait modifier :

- L'utilisateur doit faire attention aux tailles des tableaux qu'il saisit dans l'édition des templates spéciaux (si un tableau d'objet contient 2 types spéciaux, il faudra que ces 2 types aient le même nombre dans la taille du tableau).

- Dans la fenêtre qui affiche pour chaque opération les paramètres et le retour, il n'y a pas d'éléments visuels indiquant si un type fait partie d'un tableau d'objets ou non :

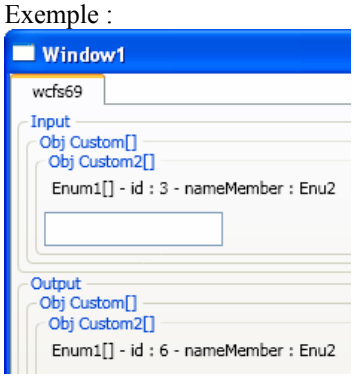


Est-ce que Enu2 est contenu uniquement dans Custom[] ou bien dans Custom2[] ?

Pour construire cet affichage, j'utilise un XmlTextReader et je parcours le fichier dataFile.xml qui contient toutes les informations sur les opérations disponibles. A chaque fois que je trouve un nœud qui porte le nom type, je crée une nouvelle ligne (un StackPanel horizontal) dans lequel je mets les informations du paramètre actuel et si c'est un type « final », un comboBox. Une fois fait, cette ligne est ajoutée dans le StackPanel vertical principal.

Avec ce système, je suis embêté pour faire ressortir les éléments qui en contiennent d'autres (dans l'image ci-dessus, chaque objet cu contient un tableau de Custom2 et chaque élément de ce tableau contient un tableau d'Enum1.

Je n'ai pas de solution rapide permettant d'améliorer le visuel. Si c'était à refaire, je pencherai plus pour faire une transformation XSLT pour créer le code XAML de la fenêtre (et obtenir au final le même type d'affichage que celui que j'obtiens pour l'application générée, c'est-à-dire qu'à chaque fois qu'il n'y a pas un type simple, complex ou « spécial », un GroupBox soit affiché et les membres à l'intérieur).
Exemple :



24h

Sujet du travail de diplôme :
WSDL & XAML

Etudiant :
Vionnet Olivier

Professeur responsable :
Schumacher Michael Ignaz
Russo David

Semaine : *25*
Woche : *16.05.2008*
22.05.2008

Décompte des heures hebdomadaires
Deklaration Wochenstunden

Tâche/ Aufgabe	Heures/ Stunden

- **Point sur le refactoring :**
  Pas mal de modifications ont été faites :
  Maintenant, j'utilise le BackGroundWorker pour faire des traitements long et qui me permet tout de même de mettre à jour la partie UserInterface (tout en ne bloquant pas l'application).
  C'est nettement plus simple à utiliser (je ne m'occupe plus de savoir si je dois être sur un thread UI ou non pour mettre à jour l'UI, les méthodes progressChanged et runWorkerCompleted sont sur le thread UI.

  Tous les fichiers de l'application sont maintenant dans des dossiers. Il ne reste plus que app et Window1 (les éléments par défaut pour lancer l'application) qui sont à la racine du projet.

  Génération de code :
  Une nouvelle classe (CommonGeneration) a été crée. Elle permet de regrouper à une place le traitement commun à la génération automatique, Wizard et Nunit (création des dossiers pour la génération en cours, génération proxy et config, transformation proxy en dll, récupération des opérations du web service et récupération des types spéciaux).

  Transformation XSLT :
  Template pour la création des UserControls : maintenant, il n'y a plus qu'un jeu de template pour la génération automatique et wizard.
  La génération du code cs de la fenêtre se base également plus que sur le template xslt du wizard.
  Par contre, la génération du code XAML a un template séparé pour la génération automatique et wizard.

- **Bugs corrigés :**
  Dans la partie Wizard, choix des UserControls à appliquer :
  Lorsqu'il y a plus de 9 UserControls (sur 1 ou plusieurs opérations), le champ pour l'UserControl 1 contient de mauvais template (le template 1 ce qui est normal mais également ceux du type 1x (genre 10, 11,…) qui ne doivent pas apparaître pour le 1.
  L'erreur provient de parce que je fais simplement un contrôle en utilisant contains id<nbre>. Donc si je fais un contain id1, id10 contient id1 et c'est pourquoi j'ai des templates qui appraissent. Le problème a été corrigé en contrôlant de la façon suivante : id<nbre>. .Vu que l'id est toujours à la fin du nom de fichier avant l'extension, le point

me permet d'éviter de prendre en compte id10.
Remarque : j'ai également du ajouter l'option id<nbre>_ pour le cas où il y a un UC spécial à mettre (il y a un underscore et du texte avant le point).

Génération du proxy et .config :
Ce n'était pas une très bonne idée de mettre ces 2 fichiers générés dans un dossier commun à toutes les générations (si le contact avec un web service échoue, l'application va prendre le proxy et config qui s'y trouve et au final, ça donnera un résultat erroné.
Ces 2 fichiers sont maintenant stockés dans le dossier temp propre à chaque génération.
Les entrées pour la création et l'utilisation du dossier svcutilFiles ont été supprimées.

Gestion des erreurs :
Le système est mal pensé. Lorsqu'il y a une erreur grave (par exemple, l'adresse du web service est fausse), la génération du proxy ne se fait pas (il y aura bien une entrée dans le Log des erreurs mais l'application va continuer son traitement et déclencher une exception plus loin).

J'ai modifié de façon à stopper le traitement de l'application lorsqu'il y a des erreurs clés (là où l'erreur entraine un plantage plus loin dans l'application).
Dans certains cas, je ne peux pas me baser sur les erreurs provenant d'un traitement de processus, par exemple, le cas du svcutil : si le web service contient des bindings http (get ou post), j'ai des erreurs qui sont loggées mais la génération du proxy a très bien pu se passer correctement et ces erreurs n'entrainent pas ensuite un plantage de l'application.
Dans ce cas, l'erreur est notée mais n'est pas considérée comme étant fatale.

- **Améliorations :**
  Pour la génération automatique et Wizard, si la génération réussit, j'affiche dans la fenêtre principale un bouton pour lancer l'application générée (il n'apparaît que si'l y a l'exécutable qui a été trouvé physiquement).

Suite du travail :

- **Continuer le refactoring :**
  Terminer de documenter tout le code (il me reste encore une 60aines de méthodes/constructeurs/classes à traiter sans compter la partie xslt).
  J'ai cette information parce que j'ai modifié dans les propriétés du projet la partie build : je génère maintenant également à chaque build la documentation xml dans le dossier bin\Debug\Wpf_ClientGen.XML.

  Il me reste encore à contrôler la documentation des fichiers xslts.

  Une fois finie, je relance tous les tests des web services que j'ai créés pour être certain que les modifications apportées au code n'ont pas créé de problèmes.

  A ce stade, l'application sera considérée comme terminée. Ca sera la version 1.0.

- **Outil pour lire la documentation du code :**
  C'est une chose à laquelle je n'avais pas pensé mais qu'il me semble intéressante de faire : utiliser pleinement la documentation du code en proposant un outil de type .chm qui permettrait de consulter la documentation plus facilement que de naviguer dans les fichiers de code (ce qui pourrait rendre la compréhension plus facile du contenu de l'application pour les personnes qui seraient amenées à continuer ce projet).

  Pour réaliser cet outil, je pense utiliser les programmes suivants :
  Sandcastle :
  création de documentation dans un style MSDN
  http://www.codeplex.com/Sandcastle

  (http://www.microsoft.com/downloads/details.aspx?FamilyId=E82EA71D-DA89-42EE-A715-696E3A4873B2&displaylang=en)

DocProject for Sandcastle :
Facilite l'administration et le développement de documentation faite avec Sandcastle
http://www.codeplex.com/DocProject/

Je n'ai encore jamais utilisé ce genre de programmes, donc c'est l'occasion de découvrir et de tester tout ça (je pense que ça devrait me prendre entre 6 et 12 heures, suivants les problèmes rencontrés (les programmes semblent être encore en développement) et la facilité avec laquelle je vais comprendre comment ça marche.

Cette documentation sera intégrée à la solution (mais pour éviter tout problème, j'utiliserai une copie de la version 1.0 pour intégrer la solution de documentation).

Total : 24h

Sujet du travail de diplôme :
WSDL & XAML

Etudiant :
Vionnet Olivier

Professeur responsable :
Schumacher Michael Ignaz
Russo David

Semaine : *30*
Woche : *23.05.2008*
29.05.2008

Décompte des heures hebdomadaires
Deklaration Wochenstunden

Tâche/ Aufgabe	Heures/ Stunden

- **refactoring terminé :**
  A force de vouloir éliminer tout ce qui est inutile, je me suis retrouvé avec quelques bugs (qui ont été corrigés).

- **Essais de différents web service trouvés sur le web :**
  J'ai refait des tests avec des web services développés par d'autres personnes.
  GlobalWeather service
  source :
  http://developer.capeclear.com/?q=webservices/globalweather/index.shtml
  wsdl :
  http://live.capeclear.com/ccx/GlobalWeather?wsdl

- **Correction de dernière minute :**

  En testant l'application

  Pas mal de modifications ont été faites :
  Maintenant, j'utilise le BackGroundWorker pour faire des traitements long et qui me permet tout de même de mettre à jour la partie UserInterface (tout en ne bloquant pas l'application).
  C'est nettement plus simple à utiliser (je ne m'occupe plus de savoir si je dois être sur un thread UI ou non pour mettre à jour l'UI, les méthodes progressChanged et runWorkerCompleted sont sur le thread UI.

  Tous les fichiers de l'application sont maintenant dans des dossiers. Il ne reste plus que app et Window1 (les éléments par défaut pour lancer l'application) qui sont à la racine du projet.

  Génération de code :
  Une nouvelle classe (CommonGeneration) a été crée. Elle permet de regrouper à une place le traitement commun à la génération automatique, Wizard et Nunit (création des dossiers pour la génération en cours, génération proxy et config, transformation proxy en dll, récupération des opérations du web service et récupération des types spéciaux).

Transformation XSLT :
Template pour la création des UserControls : maintenant, il n'y a plus qu'un jeu de template pour la génération automatique et wizard.
La génération du code cs de la fenêtre se base également plus que sur le template xslt du wizard.
Par contre, la génération du code XAML a un template séparé pour la génération automatique et wizard.

Documentations et divers

Total : 24h

Sujet du travail de diplôme :
WSDL & XAML

Etudiant :
Vionnet Olivier

Professeur responsable :
Schumacher Michael Ignaz
Russo David

Semaine : *31-34*
Woche : *30.05.2008*
22.06.2008

Décompte des heures hebdomadaires
Deklaration Wochenstunden

Tâche/ Aufgabe	Heures/ Stunden
• **documentation :** Ecrire le rapport final, mettre à jour la documentation du code pour la partie C# qui sera générée avec SandCastle. Effectué également une nouvelle série de tests et quelques bugs ont été identifiés (et certains corrigés).	
Total :	90h

13.8. Références, sources, livres

13.8.1. Définitions

Site Internet du W3C
http://www.w3.org

Recommandation du W3C pour le WSDL 1.1
http://www.w3.org/TR/2001/NOTE-wsdl-20010315

Recommandation du W3C pour le WSDL 2.0
http://www.w3.org/TR/wsdl20

Article InfoQ parlant l'approbation du W3C pour faire du WSDL 2.0 une recommandation
http://www.infoq.com/news/2007/07/wsdl-2-recommendation

Post sur le forum MSDN concernant le support WSDL 2.0 dans WCF 3.0/3.5
http://forums.microsoft.com/MSDN/ShowPost.aspx?PostID=2243560&SiteID=1

Articles parlant du WSDL
http://searchsoa.techtarget.com/tip/0,289483,sid26_gci811272,00.html
http://www.developer.com/services/article.php/10928_1602051_1

Définition du terme Namespace
http://en.wikipedia.org/wiki/Namespace
http://microsoft.supinfo.com/def/12306/

What Is Windows Communication Foundation?
http://msdn2.microsoft.com/en-us/library/ms731082.aspx

Introduction à Windows Communication Foundation
http://webman.developpez.com/articles/dotnet/wcf/intro/

Introduction à Windows Communication Foundation
http://vincentlaine.developpez.com/tuto/dotnet/wcf/

Programmez !, Hors-Série .NET, Octobre – Novembre 2007

Définition du XAML
http://fr.wikipedia.org/wiki/XAML

13.8.2. Applications/plugins permettant récupérer le contenu WSDL

Feature Specifications for Visual Studio 2008 and .NET Framework 3.5
http://msdn2.microsoft.com/en-us/vstudio/aa948851.aspx

WSCF - Web Services Contract First
http://www.thinktecture.com/resourcearchive

WSCF – Démo sur le fonctionnement et les possibilités du plugin

http://www.thinktecture.com/resourcearchive/tools-and-software/wscf/wscf-walkthrough

Contrat de licence WSCF
thinktecture WSCF (WsContractFirst) Licence Agreement

Copyright
The software, the documentation and all other content of the distributed pack-age, if not otherwise stated, is copyright 2003-2006 thinktecture (http://www.thinktecture.com/). All rights reserved.

The WizardControl control is copyright Tim Dawson (http://www.divil.co.uk/net/controls/wizardcontrol/).
The WRM.PropertyTree control is copyright Russell Morris (http://www.codeproject.com/cs/miscctrl/propertytree.asp).

Terms of Use
Permission is hereby granted to use this software, for both commercial and non-commercial purposes, without limitations and free of charge.
Permission is hereby granted to copy and distribute the software for non-commercial purposes. A commercial distribution is NOT allowed without prior written permission of the authors.

Warranty
This software is supplied "AS IS". The authors disclaim all warranties, ex-pressed or implied, including, without limitation, the warranties of merchant-ability and of fitness for any purpose. The authors assume no liability for di-rect, indirect, incidental, special, exemplary, or consequential damages, which may result from the use of this software, even if advised of the possibility of such damage.

Submissions
The authors encourage the submission of comments and suggestions concern-ing this software. All suggestions will be given serious technical considera-tion. By submitting material to the authors, you are granting the right to make any use of the material deemed appropriate, i.e. any communication or mate-rial that you transmit to the authors by electronic mail or otherwise, including any data, questions, comments, suggestions or the like, is, and will be treated as, non-confidential and nonproprietary information. The authors may use such communication or material for any purpose whatsoever, including, but not limited to, reproduction, disclosure, transmission, publication, broadcast and further posting. Further, the authors are free to use any ideas, concepts, know-how or techniques contained in any communication or material you send for any purpose whatsoever, including, but not limited to, developing, manufacturing and marketing products.

13.8.3. Web Services

Information et définition de webservices
http://www.w3schools.com/ngws/ngws_webservices.asp

Defining the web service with WSDL
http://www.ws-standards.com/wsdl.asp

WSDL Tutorial

http://www.w3schools.com/wsdl/default.asp

Describing Web Services using DAML-S and WSDL
http://www.daml.org/services/daml-s/0.7/daml-s-wsdl.html

Language WSDL 1.0
http://msdn2.microsoft.com/fr-fr/library/bb469923.aspx

Introduction à Windows Communication Foundation
http://www.microsoft.com/france/msdn/netframework/3/wcf/introduction-a-windows-communication-framework.mspx

Description du langage WSDL
http://msdn2.microsoft.com/fr-fr/library/bb469924.aspx

Web Services (Concepts, Architectures and Applications)
Gustavo Alonso, Fabio Casati, Harumi Kuno, Vijay Machiraju
Edition Springer, 2004, ISBN 3-540-44008-9

13.8.4. Création d'un client consommant un webservice

Tutorial WCF, Fernandes Bruno, HEVS décembre 2006

13.8.5. XAML

TabControl Overview
http://msdn2.microsoft.com/en-us/library/ms749171.aspx

RichTextBox Overview :
http://msdn2.microsoft.com/en-us/library/aa970779.aspx

Affichage de contenu flexible avec des documents dynamiques :
http://msdn.microsoft.com/msdnmag/issues/07/08/WPF/Default.aspx?loc=fr

FlowDocumentPageViewer Class
http://msdn2.microsoft.com/en-us/library/system.windows.controls.flowdocumentpageviewer.aspx

FlowDocumentScrollViewer Class :
http://msdn2.microsoft.com/en-us/library/system.windows.controls.flowdocumentscrollviewer.aspx

ToolBars in Avalon
http://nayyeri.net/blog/ToolBars-in-Avalon/

StatusBar Class
http://msdn2.microsoft.com/en-us/library/system.windows.controls.primitives.statusbar.aspx

Menu Overview
http://msdn2.microsoft.com/en-us/library/ms747430.aspx

Creating Buttons with XAML and C#

http://blog.paranoidferret.com/index.php/2007/06/29/creating-buttons-with-xaml-and-csharp/

CheckBox Overview
http://msdn2.microsoft.com/en-us/library/ms743611(VS.85).aspx

ListBox Class
http://msdn2.microsoft.com/en-us/library/system.windows.controls.listbox.aspx

ComboBox Class
http://msdn2.microsoft.com/en-us/library/system.windows.controls.combobox.aspx

RadioButton Overview
http://msdn2.microsoft.com/en-us/library/ms745072(VS.85).aspx

Pixel Snapping in WPF Applications
http://msdn2.microsoft.com/en-us/library/aa970908.aspx

Exemple sur la propriété ClipToBounds
http://wpf.netfx3.com/files/folders/code_snippets/entry6490.aspx

Visibility Enumeration
http://wpf.netfx3.com/files/folders/code_snippets/entry6490.aspx

WPF Layout - The WrapPanel
http://blogs.vbcity.com/xtab/archive/2007/08/07/8583.aspx

Little Boxes : WPF's UniformGrid
http://blogs.vbcity.com/xtab/archive/2007/08/02/8569.aspx

XAML StaticResource Markup Extension
http://msdn2.microsoft.com/en-us/library/ms750950(VS.90).aspx

XAML DynamicResource Markup Extension
http://msdn2.microsoft.com/en-us/library/ms748942(VS.90).aspx

XAML Binding
http://msdn2.microsoft.com/en-us/library/ms750413(VS.90).aspx

Start experimenting with XAML on your XP machine in under 15 minutes!
http://geekswithblogs.net/lorint/archive/2006/07/06/84287.aspx

Comparing HTML Tables to WPF Grids
http://geekswithblogs.net/lorint/archive/2006/08/01/86857.aspx

Learn WPF
http://learnwpf.com/Posts/PostsByCategory.aspx?categoryId=851b9683-69b8-4d66-9381-1944b3a3a854

13.8.6. XSLT & XPath

Recommandation W3C sur le XSLT
http://www.w3.org/TR/xslt

Recommandation W3C sur le XPath
Intéressant pour la série d'exemples de Location Paths
http://www.w3.org/TR/xpath

Tutorial sur le XSLT et XPath avec de nombreux exemples (transformation XML vers HTML). Excellent site
http://www.zvon.org/xxl/XSLTutorial/Output_fre/contents.html

Tutorial sur le XSLT et XPath
http://www.topxml.com/xsl/tutorials/intro/default.asp

Valeur, boucle et élément courant en XSLT 1.0. Contient pas mal d'exemples
http://erwy.developpez.com/cours/langage/xslt/boucle/

Référence XSLT
http://msdn2.microsoft.com/fr-fr/library/ms256069.aspx

Référence XPath
http://msdn2.microsoft.com/fr-fr/library/ms256115.aspx

Construire une application XML
Jean-Christophe Bernadac, François Knab
Edition Eyrolles, 1999, ISBN 2-212-09081-1

XML
Benoît Marchal
Edition CampusPress, 2000, ISBN 2-7440-0924-5

Introduction à XPath
http://tecfa.unige.ch/guides/tie/html/xml-xpath/xml-xpath.html

Eléments de programmation XSLT
http://tecfa.unige.ch/guides/tie/html/xml-xslt2/xml-xslt2.html

Predicates & Conditionals in XPath & XSLT
http://www2.sims.berkeley.edu/academics/courses/is290-14/s05/lecture-7/allslides.html

XSL Concepts and Practical Use
http://nwalsh.com/docs/tutorials/xsl/xsl/slides.html

Maîtriser les sauts de lignes en XSLT
http://xmlfr.org/documentations/faq/010703-0001

13.8.7. Déroulement du développement de l'application

Comment créer, copier, déplacer, supprimer un fichier ?
http://faqcsharp.developpez.com/?page=file#file_file

Problème concernant la modification de donnée dans le fichier .settings. La solution n'est pas très bonne (modifier un fichier généré), par contre, ça m'a permis de découvrir qu'il faut mettre le scoop en user pour pouvoir modifier une valeur.

http://forums.microsoft.com/MSDN/ShowPost.aspx?PostID=569651&SiteID=1

Information sur la classe Assembly
http://msdn2.microsoft.com/en-us/library/system.reflection.assembly.aspx

Introduction à CodeDom
http://www.techheadbrothers.com/Articles.aspx/introduction-codedom

CodeDom et XAML
http://www.thescripts.com/forum/thread622936.html

Petite définition du CodeDom
http://www.labo-microsoft.com/quickstart/howto/doc/ListBuilder.aspx

Un peu plus d'information sur ce que propose le CodeDom chez MSDN
http://msdn2.microsoft.com/fr-fr/library/650ax5cx.aspx

Introduction à la génération de code
http://www.aspfree.com/c/a/.NET/Introducing-Code-Generation/11/

Introducing Code Generation (quelques pages tirés du livre *Code Generation in Microsoft .NET*)
http://www.aspfree.com/c/a/.NET/Introducing-Code-Generation/11/

Using CodeDOM Code Generation to Implement Repetitive Patterns
http://www.devx.com/dotnet/Article/28193/1954?pf=true

XSLT for MSXML
http://msdn2.microsoft.com/en-us/library/ms759204(VS.85).aspx

Creating XML Writers
http://msdn2.microsoft.com/en-us/library/kkz7cs0d.aspx

XML transformation using Xslt in C# (pratique pour avoir une petite idée de comment faire au niveau du code C# pour transformer avec le XSLT.)
http://www.csharpfriends.com/Articles/getArticle.aspx?articleID=63

File path in C# (chemin relatif pour arriver sur un fichier contenu dans le projet)
http://geekswithblogs.net/vkamat/archive/2006/02/01/67868.aspx

CanVerticallyScroll Property : une propriété à ne pas utiliser. Pour le scroll, utiliser un ScrollViewer qui englobe tout ce qui doit pouvoir être scrollé.
http://msdn2.microsoft.com/en-us/library/system.windows.controls.stackpanel.canverticallyscroll.aspx

13.8.8. Divers

Quand utiliser de SvcUtil.exe
http://forums.microsoft.com/MSDN/ShowPost.aspx?PostID=1386361&SiteID=1

Outil Web Services Description Language Tool (Wsdl.exe)
http://msdn2.microsoft.com/fr-fr/library/7h3ystb6.aspx

Commentaire sur le fait que le Add Service Reference sous VS invoque SvcUtil.exe
http://forums.microsoft.com/MSDN/ShowPost.aspx?PostID=1026797&SiteID=1

Mon premier service WCF : consommateur (part II)
http://www.c2i.fr/code.aspx?IDCode=667

svcutil.exe busted in Beta2
http://blog.rolpdog.com/2007/07/svcutilexe-busted-in-beta2.html

13.9. Remerciements

Un grand merci à David Russo et Michael Schumacher pour le suivi de ce travail de diplôme.

14. Table des matières