Construction d'une application à trois couches avec

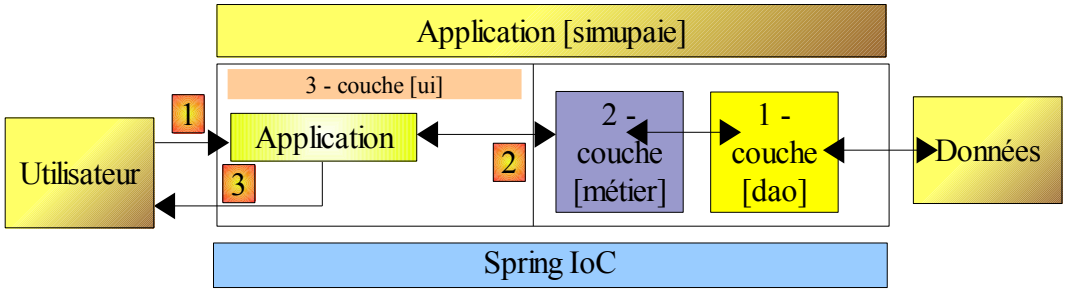
ASP.NET 2.0, C#,
Spring.Net et NHibernate

serge.tahe at istia.univ-angers.fr, juin 2010

1 Introduction

Nous souhaitons écrire une application .NET permettant à un utilisateur de faire des simulations de calcul de la paie des assistantes maternelles de l'association " Maison de la petite enfance " d'une commune. Nous nous intéresserons autant à l'organisation du code DotNet de l'application qu'au code lui-même.

L'application finale, que nous appellerons [SimuPaie] aura la structure à trois couches suivante :



- la couche [1-dao] (dao=Data Access Object) s'occupera de l'accès aux données. Celles-ci seront placées dans une base de données.
- la couche [2-métier] s'occupera de l'aspect métier de l'application, le calcul de la paie.
- la couche [3-ui] (ui=User Interface) s'occupera de la présentation des données à l'utilisateur et de l'exécution de ses requêtes. Nous appelons [Application] l'ensemble des modules assurant cette fonction. Elle est l'interlocuteur de l'utilisateur.
- les trois couches seront rendues indépendantes grâce à l'utilisation d'interfaces .NET
- l'intégration des différentes couches sera réalisée par **Spring IoC**

Le traitement d'une demande d'un client se déroule selon les étapes suivantes :

1. le client fait une demande à l'application.
2. l'application traite cette demande. Pour ce faire, elle peut avoir besoin de l'aide de la couche [métier] qui elle-même peut avoir besoin de la couche [dao] si des données doivent être échangées avec la base de données. L'application reçoit une réponse de la couche [métier].
3. selon celle-ci, elle envoie la vue (= la réponse) appropriée au client.

L'interface présentée à l'utilisateur peut avoir diverses formes :

1. une application console : dans ce cas, la vue est une suite de lignes de texte.
2. une application graphique windows : dans ce cas, la vue est une fenêtre windows
3. une application web : dans ce cas, la vue est une page HTML
4. ...

Nous écrirons différentes versions de cette application :

1. une version ASPNET comportant un unique formulaire et construite avec une architecture à une couche.
2. une version identique à la précédente mais avec des extensions Ajax
3. une version ASP.NET s'appuyant sur une architecture à trois couches où la couche d'accès aux données est implémentée avec le framework NHibernate. Elle aura toujours l'unique formulaire de la version 1.
4. une version 4 ASP.NET multi-vues et mono-page avec l'architecture trois couches de la version 3.
5. la partie serveur d'une application client / serveur où le serveur est implémenté par un service web s'appuyant sur l'architecture en couches de la version 3.
6. la partie client de l'application client / serveur précédente, implémentée par une couche ASP.NET.
7. une version 7 ASP.NET multi-vues et multi-pages avec l'architecture trois couches de la version 3.
8. une version 8 ASP.NET multi-vues et multi-pages cliente du service web de la version 5.
9. une version 9 ASP.NET multi-vues et multi-pages avec l'architecture trois couches de la version 3 où la couche d'accès aux données est implémentée par des classes de Spring qui facilitent l'utilisation du framework NHibernate.
10. une version 10 implémentée en FLEX et cliente du service web de la version 5.

Pré-requis

Dans une échelle [débutant-intermédiaire-avancé], ce document est dans la partie [intermédiaire]. Sa compréhension nécessite divers pré-requis qu'on pourra trouver dans certains des documents que j'ai écrits :

1. **Programmation ASP.NET** [http://tahe.developpez.com/dotnet/aspnet/vol1] et [http://tahe.developpez.com/dotnet/aspnet/vol2]
2. **Langage C# 2008** : [http://tahe.developpez.com/dotnet/csharp/] : classes, interfaces, héritage, polymorphisme
3. [**Spring IoC**], disponible à l'url [http://tahe.developpez.com/dotnet/springioc]. Présente les bases de l'inversion de contrôle (Inversion of Control) ou injection de dépendances (Dependency Injection) du framework **Spring.Net** [http://www.springframework.net].
4. [**Construction d'une application web à trois couches avec Spring et VB.NET - Parties 1 et 2**], disponibles aux url [http://tahe.developpez.com/dotnet/web3tier-part1] et [http://tahe.developpez.com/dotnet/web3tier-part2]. Ces deux articles présentent une application simplifiée d'achats de produits sur le web. Son architecture 3 couches implémente le modèle MVC.

Des conseils de lecture sont parfois donnés au début des paragraphes de ce document. Ils référencent les documents précédents.

Outils

Les outils utilisés dans cette étude de cas sont librement disponibles sur le web. Ce sont les suivants :

- **Visual C# 2008, Visual Web Developer Express 2008**, **SQL Server Express 2005** disponibles à l'url [http://www.microsoft.com/express/downloads/].
- **Spring IoC** pour l'instanciation des services nécessaires à l'application, disponible à l'url [http://www.springframework.net/] - voir également [3]
- **NHibernate** pour la couche d'accès aux données du SGBD [http://sourceforge.net/projects/nhibernate/]
- NUnit : [http://www.nunit.org] pour les tests unitaires.

2 Une rapide introduction à ASP.NET

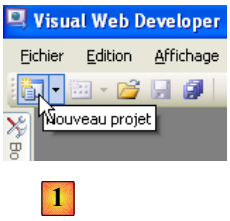
Nous nous proposons ici d'introduire, à l'aide de quelques exemples, les concepts d'ASP.NET qui nous seront utiles dans la suite du document. Cette introduction ne permet pas de comprendre les subtilités des échanges client / serveur d'une application web. Pour cela, on pourra lire :

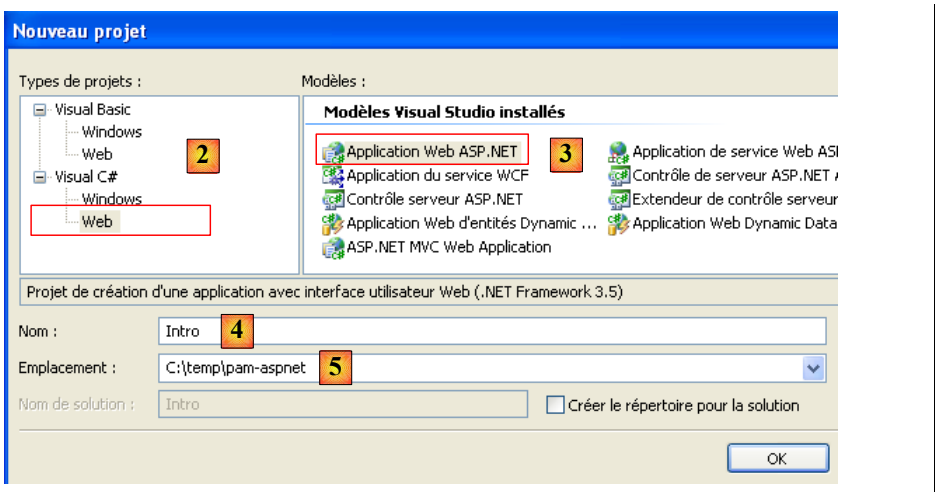
- **Programmation ASP.NET** [http://tahe.developpez.com/dotnet/aspnet/vol1] et [http://tahe.developpez.com/dotnet/aspnet/vol2]

Cette introduction est faite pour ceux qui voudraient aller vite en acceptant, dans un premier temps, de laisser dans l'ombre des points qui peuvent être importants. La suite du document permet d'approfondir ces derniers. Ceux qui connaissent ASP.NET peuvent passer directement au paragraphe 3, page 36.

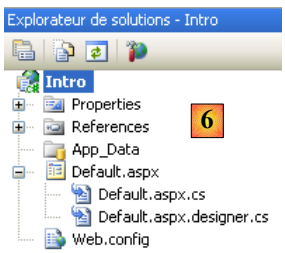
2.1 Un projet exemple

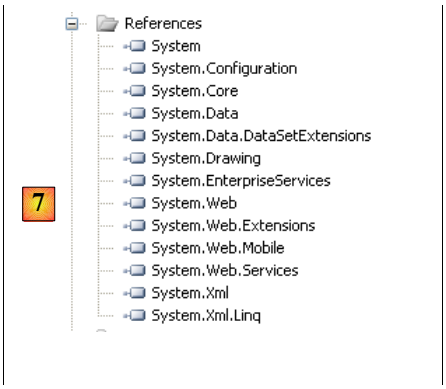
2.1.1 Création du projet

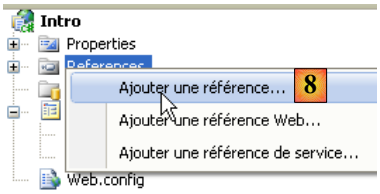




- en [1], on crée un nouveau projet avec Visual Web Developer
- en [2], on choisit un projet web en Visual C#
- en [3], on indique qu'on veut créer une application web ASP.NET
- en [4], on donne un nom à l'application. Un dossier sera créé pour le projet avec ce nom.
- en [5], on indique le dossier parent du dossier [4] du projet





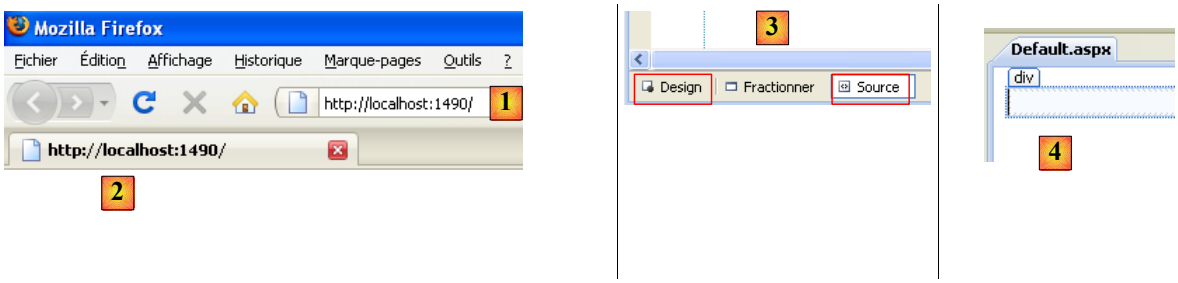


- en [6], le projet créé
- [Default.aspx] est une page web créée par défault. Elle contient des balises HTML et des balises ASP.NET

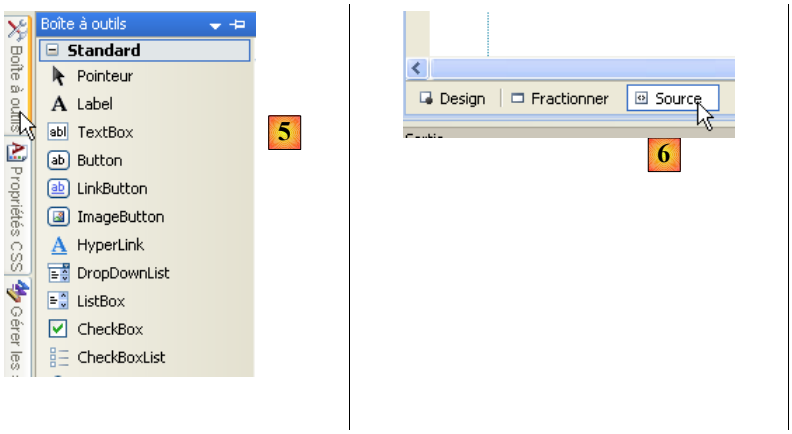
- [Default.aspx.cs] contient le code de gestion des événements provoqués par l'utilisateur sur la page [Defaul.aspx] affichée dans son navigateur
- [Default.aspx.designer.cs] contient la liste des composants ASP.NET de la page [Default.aspx]. Chaque composant ASP.NET déposé sur la page [Default.aspx] donne naissance à la déclaration de ce composant dans [Default.aspx.designer.cs].
- [Web.config] est le fichier de configuration du projet ASP.NET.
- [References] est la liste des Dll utilisées par le projet web. Ces Dll sont des bibliothèques de classes que le projet est amené à utiliser. En [7] la liste des Dll mises par défaut dans les références du projet. La plupart sont inutiles. Si le projet doit utiliser une Dll non listée en [7], celle-ci peut être ajoutée par [8].

2.1.2 La page [Default.aspx]

Si on exécute le projet par [Ctrl-F5], la page [Default.aspx] est affichée dans un navigateur :



- en [1], l'Url du projet web. Visual Web Developer a un serveur web intégré qui est lancé lorsqu'on demande l'exécution d'un projet. Il écoute sur un port aléatoire, ici 1490. Le port d'écoute est habituellement le port 80. En [1], aucune page n'est demandée. Dans ce cas, c'est la page [Default.aspx] qui est affichée, d'où son nom de page par défaut.
- en [2], la page [Default.aspx] est vide.
- dans Visual Web Developer, la page [Default.aspx] [3] peut être construite visuellement (onglet [Design]) ou à l'aide de balises (onglet [Source])
- en [4], la page [Defaul.aspx] en mode [Design]. On la construit en y déposant des composants que l'on trouve dans la boîte à outils [5].



Le mode [Source] [6] donne accès au code source de la page :

```
1.  <%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
    Inherits="Intro._Default" %>
2.
3.  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4.  <html xmlns="http://www.w3.org/1999/xhtml">
5.  <head runat="server">
6.    <title></title>
7.  </head>
8.  <body>
9.    <form id="form1" runat="server">
```

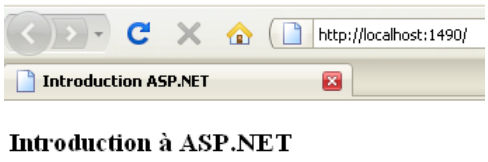
```
10.   <div>
11.   </div>
12.   </form>
13. </body>
14. </html>
```

- la ligne 1 est une directive ASP.NET qui liste certaines propriétés de la page
  - la directive *Page* s'applique à une page web. Il y a d'autres directives telles que *Application*, *WebService*, ... qui s'applique à d'autres objets ASP.NET
  - l'attribut *CodeBehind* indique le fichier qui gère les événements de la page
  - l'attribut *Language* indique le langage .NET utilisé par le fichier *CodeBehind*
  - l'attribut *Inherits* indique le nom de la classe définie à l'intérieur du fichier *CodeBehind*
  - l'attribut *AutoEventWireUp="true"* indique que la liaison entre un événement dans [Default.aspx] et son gestionnaire dans [Defaul.aspx.cs] se fait par le nom de l'événement. Ainsi l'événement *Load* sur la page [Default.aspx] sera traité par la méthode *Page_Load* de la classe *Intro._Default* définie par l'attribut *Inherits*.
- les lignes 4-14 décrivent la page [Defaul.aspx] à l'aide de balises :
  - Html classiques telles que la balise <*body*> ou <*div*>
  - ASP.NET. Ce sont les balises qui ont l'attribut *runat="server"*. Les balises ASP.NET sont traitées par le serveur web avant envoi de la page au client. Elles sont transformées en balises Html. Le navigateur client reçoit donc une page Html standard dans laquelle il n'existe plus de balises ASP.NET.

La page [Default.aspx] peut être modifiée directement à partir de son code source. C'est parfois plus simple que de passer par le mode [Design]. Nous modifions le code source de la façon suivante :

```
1.  <%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
    Inherits="Intro._Default" %>
2.
3.  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4.  <html xmlns="http://www.w3.org/1999/xhtml">
5.  <head runat="server">
6.    <title>Introduction ASP.NET</title>
7.  </head>
8.  <body>
9.    <h3>Introduction à ASP.NET</h3>
10.   <form id="form1" runat="server">
11.   <div>
12.   </div>
13.   </form>
14. </body>
15. </html>
```

En ligne 6, nous donnons un titre à la page grâce à la balise Html <title>. En ligne 9, nous introduisons un texte dans le corps (<body>) de la page. Si nous exécutons le projet (Ctrl-F5), nous obtenons le résultat suivant dans le navigateur :



Introduction à ASP.NET

2.1.3 Les fichiers [Default.aspx.designer.cs] et [Default.aspx.cs]

Le fichier [Default.aspx.designer.cs] déclare les composants de la page [Defaul.aspx] :

```
1.  //------------------------------------------------------------------------------
2.  // <auto-generated>
3.  //     Ce code a été généré par un outil.
4.  //     Version du runtime :2.0.50727.3603
5.  //
6.  //     Les modifications apportées à ce fichier peuvent provoquer un comportement incorrect et seront perdues si
7.  //     le code est régénéré.
8.  // </auto-generated>
9.  //------------------------------------------------------------------------------
```

```
10.
11. namespace Intro {
12.
13.
14.     public partial class _Default {
15.
16.         /// <summary>
17.         /// Contrôle form1.
18.         /// </summary>
19.         /// <remarks>
20.         /// Champ généré automatiquement.
21.         /// Pour modifier, déplacez la déclaration de champ du fichier de concepteur dans le fichier code-
    behind.
22.         /// </remarks>
23.         protected global::System.Web.UI.HtmlControls.HtmlForm form1;
24.     }
25. }
```

On trouve dans ce fichier la liste des composants ASP.NET de la page [Default.aspx] ayant un identifiant. Ils correspondent aux balises de [Default.aspx] ayant l'attribut *runat="server"* et l'attribut *id*. Ainsi le composant de la ligne 23 ci-dessus correspond à la balise

```
<form id="form1" runat="server">
```

de [Default.aspx].

Le développeur interagit peu avec le fichier [Default.aspx.designer.cs]. Néanmoins ce fichier est utile pour connaître la classe d'un composant particulier. Ainsi on voit ci-dessous que le composant *form1* est de type *HtmlForm*. Le développeur peut alors explorer cette classe pour en connaître les propriétés et méthodes. Les composants de la page [Default.aspx] sont utilisés par la classe du fichier [Default.aspx.cs] :

```
1.  using System;
2.  using System.Collections.Generic;
3.  using System.Linq;
4.  using System.Web;
5.  using System.Web.UI;
6.  using System.Web.UI.WebControls;
7.
8.  namespace Intro
9.  {
10.   public partial class _Default : System.Web.UI.Page
11.   {
12.     protected void Page_Load(object sender, EventArgs e)
13.     {
14.
15.     }
16.   }
17. }
```

On notera que la classe définie dans les fichiers [Default.aspx.cs] et [Default.aspx.designer.cs] est la même (ligne 10) : *Intro._Default*. C'est le mot clé *partial* qui rend possible d'étendre la déclaration d'une classe sur plusieurs fichiers, ici deux.

Ligne 10, ci-dessus, on voit que la classe [_Default] étend la classe [Page] et hérite de ses événements. L'un d'entre-eux est l'événement *Load* qui se produit lorsque la page est chargée par le serveur web. Ligne 12, la méthode *Page_Load* qui gère l'événement *Load* de la page. C'est généralement ici qu'on initialise la page avant son affichage dans le navigateur du client. Ici, la méthode *Page_Load* ne fait rien.

La classe associée à une page web, ici la classe *Intro._Default*, est créée au début de la requête du client et détruite lorsque la réponse au client a été envoyée. Elle ne peut donc servir à mémoriser des informations entre deux requêtes. Pour cela il faut utiliser la notion de *session utilisateur*.

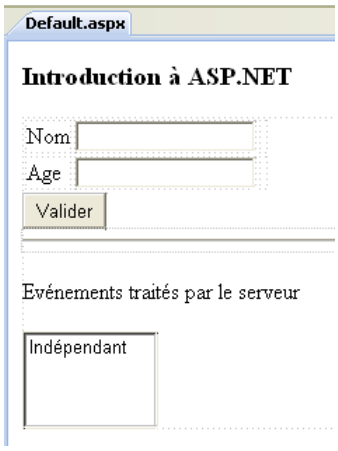
2.2 Les événements d'une page web ASP.NET

Nous construisons la page [Default.aspx] suivante :

```
1.  <%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
    Inherits="Intro._Default" %>
2.
3.  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4.  <html xmlns="http://www.w3.org/1999/xhtml">
```

```
<head runat="server">
  <title>Introduction ASP.NET</title>
</head>
<body>
  <h3>Introduction à ASP.NET</h3>
  <form id="form1" runat="server">
  <div>
    <table>
      <tr>
        <td>
          Nom</td>
        <td>
          <asp:TextBox ID="TextBoxNom" runat="server"></asp:TextBox>
        </td>
        <td>
           </td>
      </tr>
      <tr>
        <td>
          Age</td>
        <td>
          <asp:TextBox ID="TextBoxAge" runat="server"></asp:TextBox>
        </td>
        <td>
           </td>
      </tr>
    </table>
  </div>
  <asp:Button ID="ButtonValider" runat="server" Text="Valider" />
  <hr />
  <p>
    Evénements traités par le serveur</p>
  <p>
    <asp:ListBox ID="ListBoxEvts" runat="server"></asp:ListBox>
  </p>
  </form>
</body>
</html>
```

Le mode [Design] de la page est le suivant :



Le fichier [Default.aspx.designer.cs] est le suivant :

```
namespace Intro {
    public partial class _Default {
        protected global::System.Web.UI.HtmlControls.HtmlForm form1;
        protected global::System.Web.UI.WebControls.TextBox TextBoxNom;
        protected global::System.Web.UI.WebControls.TextBox TextBoxAge;
        protected global::System.Web.UI.WebControls.Button ButtonValider;
        protected global::System.Web.UI.WebControls.ListBox ListBoxEvts;
    }
}
```

On y retrouve tous les composants ASP.NET de la page [Default.aspx] ayant un identifiant.

Nous faisons évoluer le fichier [Default.aspx.cs] comme suit :

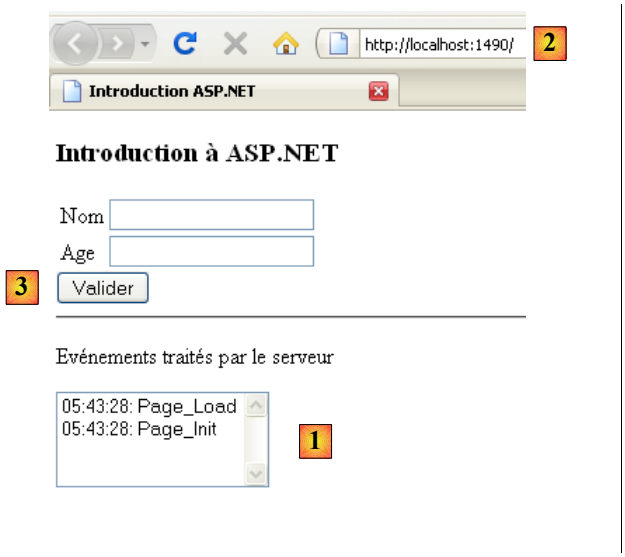
```
1.  using System;
2.
3.  namespace Intro
4.  {
5.    public partial class _Default : System.Web.UI.Page
6.    {
7.      protected void Page_Init(object sender, EventArgs e)
8.      {
9.        // on note l'événement
10.       ListBoxEvts.Items.Insert(0, string.Format("{0}: Page_Init",
    DateTime.Now.ToString("hh:mm:ss")));
11.     }
12.
13.     protected void Page_Load(object sender, EventArgs e)
14.     {
15.       // on note l'événement
16.       ListBoxEvts.Items.Insert(0, string.Format("{0}: Page_Load",
    DateTime.Now.ToString("hh:mm:ss")));
17.     }
18.
19.     protected void ButtonValider_Click(object sender, EventArgs e)
20.     {
21.       // on note l'événement
22.       ListBoxEvts.Items.Insert(0, string.Format("{0}: ButtonValider_Click",
    DateTime.Now.ToString("hh:mm:ss")));
23.     }
24.   }
25. }
```

La classe [_Default] (ligne 5) traite trois événements :

- l'événement **Init** (ligne 7) qui se produit lorsque la page a été initialisée
- l'événement **Load** (ligne 13) qui se produit lorsque la page a été chargée par le serveur web. L'évenement *Init* se produit avant l'événement *Load*.
- l'événement **Click** sur le bouton **ButtonValider** (ligne 19) qui se produit lorsque l'utilisateur clique sur le bouton [Valider]

La gestion de chacun de ces trois événements consiste à ajouter un message au composant *Listbox* nommé *ListBoxEvts*. Ce message affiche l'heure de l'événement et le nom de celui-ci. Chaque message est placé en début de liste. Aussi les messsages placés en haut de la liste sont les plus récents.

Lorsqu'on exécute le projet, on obtient la page suivante :

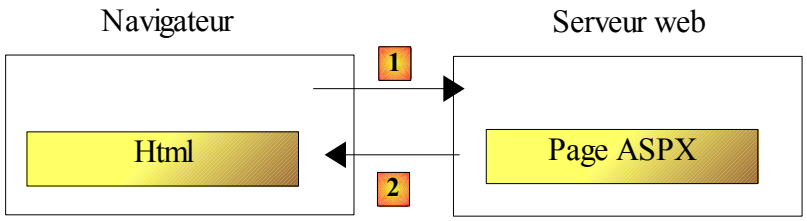


On voit en [1] que les événements *Page_Init* et *Page_Load* se sont produits dans cet ordre. On rappelle que l'événement le plus récent est en haut de la liste. Lorsque le navigateur demande la page [Default.aspx] directement par son Url [2], il le fait par une commande

HTTP (HyperText Transfer Protocol) appelée **GET**. Une fois la page chargée dans le navigateur, l'utilisateur va provoquer des événements sur le page. Par exemple il va cliquer sur le bouton [Valider] [3]. Les événements provoqués par l'utilisateur une fois la page chargée dans le navigateur, déclenchent une requête à la page [Default.aspx] mais cette fois-ci avec une commande HTTP appelée **POST**. Pour résumer :

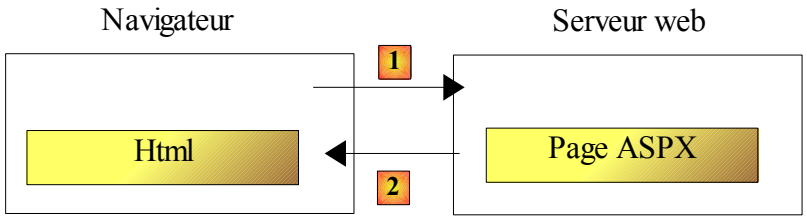
- le chargement initial d'une page P dans un navigateur est faite par une opération HTTP GET
- les événements qui se produisent ensuite sur la page produisent à chaque fois une nouvelle requête vers la même page P mais cette fois avec une commande HTTP POST. Il est possible pour une page P de savoir si elle a été demandée avec une commande GET ou une commande POST, ce qui lui permet de se comporter différemment si c'est nécessaire, ce qui est la plupart du temps le cas.

Demande initiale d'une page ASPX : GET



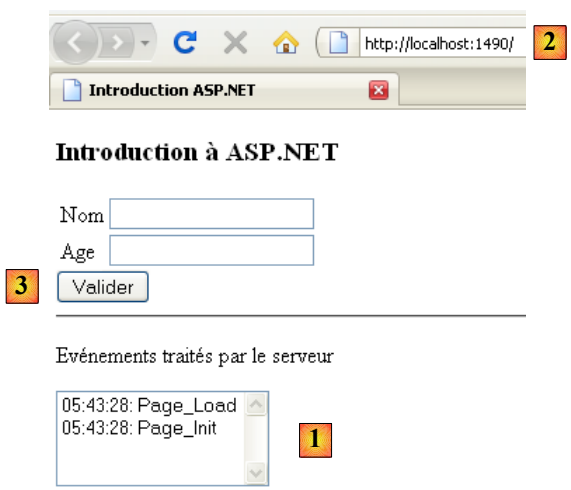
- en [1], le navigateur demande la page ASPX via une commande HTTP **GET** sans paramètres.
- en [2], le serveur web lui envoie en réponse le flux HTML traduction de la page ASPX demandée.

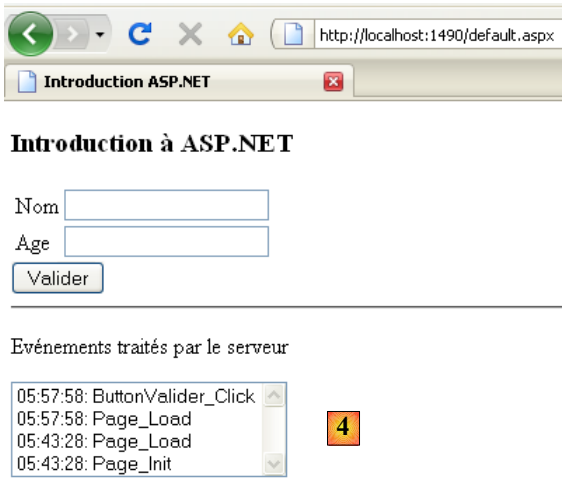
Traitement d'un événement produit sur la page affichée par le navigateur : POST



- en [1], lors d'un événement sur la page Html, le navigateur demande la page ASPX déjà acquise avec une opération GET, cette fois avec une commande HTTP **POST** accompagnée de paramètres. Ces paramètres sont les valeurs des composants qui se trouvent à l'intérieur de la balise <form> de la page HTML affichée par le navigateur. On appelle ces valeurs, les valeurs **postées** par le client. Elles vont être exploitées par la page ASPX pour traiter la demande du client.
- en [2], le serveur web lui envoie en réponse le flux HTML traduction de la page ASPX demandée initialement par le POST ou bien d'une autre page s'il y a eu **transfert** de page ou **redirection** de page.

Revenons à notre page exemple :





- en [2], la page a été obtenue par un GET.
- en [1], on voit les deux événements qui se sont produits lors de ce GET

Si, ci-dessus, l'utilisateur clique sur le bouton [Valider] [3], la page [Default.aspx] va être demandée avec un POST. Ce POST sera accompagné de paramètres qui seront les valeurs de tous les composants inclus dans la balise <form> de la page [Default.aspx] : les deux TextBox [TextBoxNom, TextBoxAge], le bouton [ButtonValider], la liste [ListBoxEvts]. Les valeurs postée pour les composants sont les suivantes :

- *TextBox* : la valeur saisie
- *Button* : le texte du bouton, ici le texte "Valider"
- *Listbox* : le texte du message sélectionné dans le ListBox

En réponse du POST, on obtient la page [4]. C'est de nouveau la page [Default.aspx]. C'est le comportement normal, à moins qu'il y ait transfert ou redirection de page par les gestionnaires d'événements de la page. On peut voir que deux nouveaux événements se sont produits :

- l'événement *Page_Load* qui s'est produit lors du chargement de la page
- l'événement *ButtonValider_Click* qui s'est produit à cause du clic sur le bouton [Valider]

On peut remarquer que :

- l'événement *Page_Init* ne s'est pas produit sur l'opération HTTP POST alors qu'il s'était produit sur l'événement HTTP GET
- l'événement *Page_Load* se produit tout le temps que ce soit sur un GET ou un POST. C'est dans cette méthode qu'on a en général besoin de savoir si on a affaire à un GET ou à un POST.
- à l'issue du POST, la page [Default.aspx] a été renvoyée au client avec les modifications apportées par les gestionnaires d'événements. C'est toujours ainsi. Une fois les événements d'une page P traités, cette même page P est renvoyée au client. Il y a deux façons d'échapper à cette règle. Le dernier gestionnaire d'événement exécuté peut
  - transférer le flux d'exécution à une autre page P2.
  - rediriger le navigateur client vers une autre page P2.

  Dans les deux cas, c'est la page P2 qui est renvoyée au navigateur. Les deux méthodes présentent des différences sur lesquelles nous reviendrons.
- l'événement *ButtonValider_Click* s'est produit après l'événement *Page_Load*. C'est donc ce gestionnaire qui peut prendre la décision du transfert ou de la redirection vers une page P2.
- la liste des événements [4] a gardé les deux événements affichés lors du chargement initial GET de la page [Default.aspx]. C'est surprenant lorsqu'on sait que la page [Default.aspx] a été recréée lors du POST. On devrait retrouver la page [Default.aspx] avec ses valeurs de conception avec donc un *ListBox* vide. L'exécution des gestionnaires *Page_Load* et *ButtonValider_Click* devrait y mettre ensuite deux messages. Or on en trouve quatre. C'est le mécanisme du **VIEWSTATE** qui explique cela. Lors du GET initial, le serveur web envoie la page [Default.aspx] avec une balise HTML *<input type="hidden" ...>* appelé **champ caché** (ligne 10 ci-dessous).

```
1.  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
2.  <html xmlns="http://www.w3.org/1999/xhtml">
3.  <head><title>
4.          Introduction ASP.NET
5.  </title></head>
```

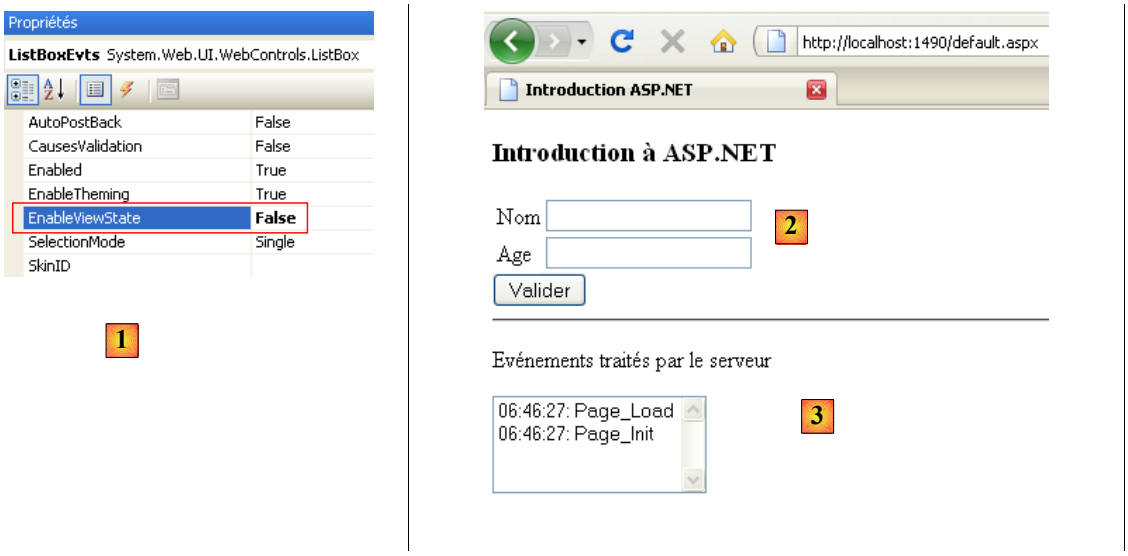
```
6.  <body>
7.    <h3>Introduction à ASP.NET</h3>
8.    <form name="form1" method="post" action="default.aspx" id="form1">
9.  <div>
10. <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
    value="/wEPDwUKLTMzMTEyNDMxMg9kFgICAw9kFgICBw8QZBAVAhMwNjoxNjozNjogUGFnZV9Mb2FkEzA2OjE2OjM2OiBQYWd
    lX0luaXQVAhMwNjoxNjozNjogUGFnZV9Mb2FkEzA2OjE2OjM2OiBQYWdlX0luaXQUKwMCZ2dkZGRW1AnTL8f/q7h2MXBLxctKD
    1UKfg==" />
11. </div>
12. ..............................
```

Dans le champ d'id "__VIEWSTATE" le serveur web met sous forme codée la valeur de tous les composants de la page. Il le fait aussi bien sur le GET initial que sur les POST qui suivent. Lorsqu'un POST sur une page P se produit :

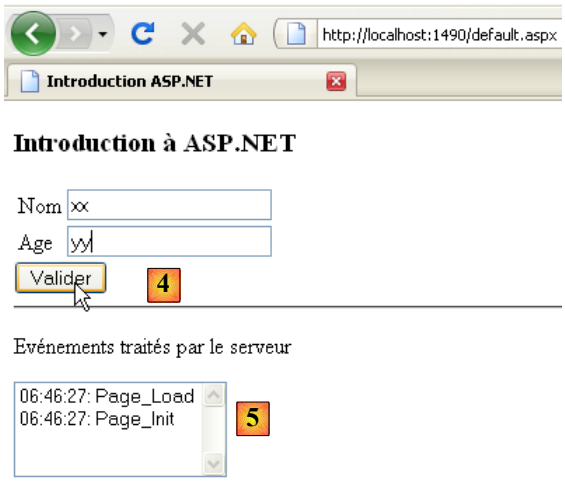
- le navigateur demande la page P en envoyant dans sa requête les valeurs de tous les composants qui sont à l'intérieur de la balise <form>. Ci-dessus, on peut voir que le composant "__VIEWSTATE" est à l'intérieur de la balise <form>. Sa valeur est donc envoyée au serveur lors d'un POST.
- la page P est instanciée et initialisée avec ses valeurs de construction
- le composant "__VIEWSTATE" est utilisé pour redonner aux composants la valeurs qu'ils avaient lorsque la page P avait été envoyée précédemment. C'est ainsi par exemple, que la liste des événements [4] retrouve les deux premiers messages qu'elle avait lorsqu'elle a été envoyée en réponse au GET initial du navigateur.
- les composants de la page P prennent ensuite les valeurs postées par le navigateur. A ce moment là, le formulaire de la page P est dans l'état où l'utilisateur l'a posté.
- l'événement *Page_Load* est traité. Ici il rajoute un message à la liste des événements[4].
- l'événement qui a provoqué le POST est traité. Ici *ButtonValider_Click* rajoute un message à la liste des événements[4].
- la page P est renvoyée. Les composants ont pour valeur :
  - soit la valeur postée, c.a.d. la valeur que le composant avait dans le formulaire lorsque celui a été posté au serveur
  - soit une valeur donnée par l'un des gestionnaires d'événements.

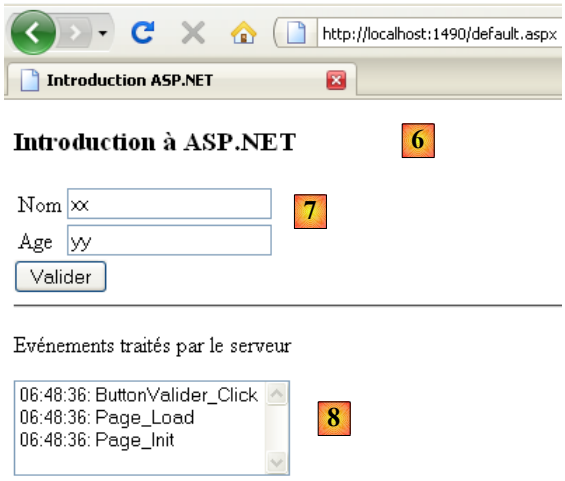
  Dans notre exemple,
  - les deux composants *TextBox* retrouveront leur valeur postée car les gestionnaires d'événements n'y touchent pas
  - la liste des événements [4] retrouve sa valeur postée, c.a.d. tous les événements déjà inscrits dans la liste, plus deux nouveaux événements créés par les méthodes *Page_Load* et *ButtonValider_Click*.

Le mécanisme du VIEWSTATE peut être activé ou inhibé au niveau de chaque composant. Inhibons-le pour le composant [ListBoxEvts] :



- en [1], le VIEWSTATE du composant [ListBoxEvts] est inhibé. Celui des TextBox [2] est activé par défaut.
- en [3], les deux événements renvoyés après le GET initial

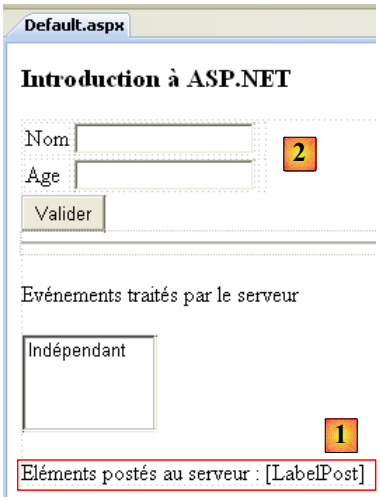




- en [4], on a rempli le formulaire et on clique sur le bouton [Valider]. Un POST vers la page [Default.aspx] va être fait.
- en [6], le résultat renvoyé après un Clic sur le bouton [Valider]
  - le mécanisme du VIEWSTATE activé explique que les TextBox [7] aient gardé leur valeur postée en [4]
  - le mécanisme du VIEWSTATE inhibé explique que le composant [ListBoxEvts] [8] n'ait pas gardé son contenu [5].

2.3 Gestion des valeurs postées

Nous allons nous intéresser ici aux valeurs postées par les deux *TextBox* lorsque l'utilisateur clique sur le bouton [Valider]. La page [Default.aspx] en mode [Design] évolue comme suit :



Le code source de l'élément rajouté en [1] est le suivant :

```
1.    <p>
2.      Eléments postés au serveur :
3.      <asp:Label ID="LabelPost" runat="server"></asp:Label>
4.  </p>
```

Nous utiliserons le composant [LabelPost] pour afficher les valeurs saisies dans les deux *TextBox* [2]. Le code du gestionnaire d'événements [Default.aspx.cs] évolue comme suit :

```
1.  using System;
2.
```

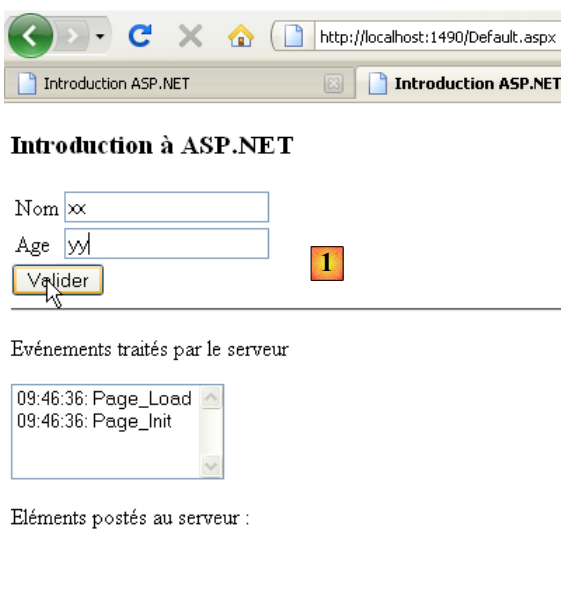
```
3.  namespace Intro
4.  {
5.    public partial class _Default : System.Web.UI.Page
6.    {
7.      protected void Page_Init(object sender, EventArgs e)
8.      {
9.        // on note l'événement
10.       ListBoxEvts.Items.Insert(0, string.Format("{0}: Page_Init",
    DateTime.Now.ToString("hh:mm:ss")));
11.     }
12.
13.     protected void Page_Load(object sender, EventArgs e)
14.     {
15.       // on note l'événement
16.       ListBoxEvts.Items.Insert(0, string.Format("{0}: Page_Load",
    DateTime.Now.ToString("hh:mm:ss")));
17.     }
18.
19.     protected void ButtonValider_Click(object sender, EventArgs e)
20.     {
21.       // on note l'événement
22.       ListBoxEvts.Items.Insert(0, string.Format("{0}: ButtonValider_Click",
    DateTime.Now.ToString("hh:mm:ss")));
23.       // on affiche le nom et l'âge
24.       LabelPost.Text = string.Format("nom={0}, age={1}", TextBoxNom.Text.Trim(),
    TextBoxAge.Text.Trim());
25.     }
26.   }
27. }
```

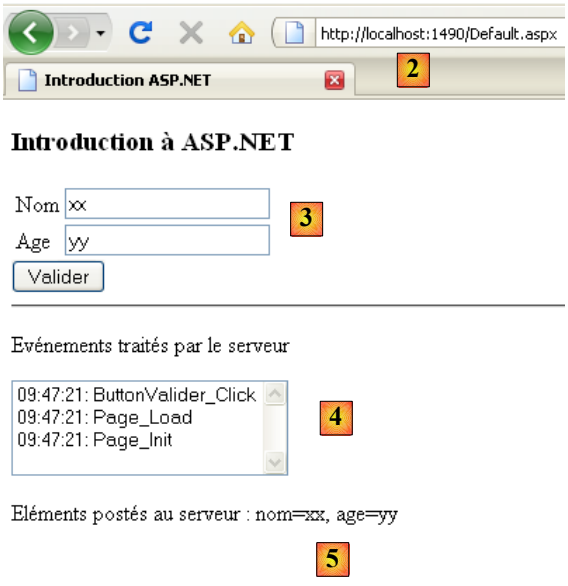
Ligne 24, on met à jour le composant *LabelPost* :

- *LabelPost* est de type [System.Web.UI.WebControls.Label] (cf Default.aspx.designer.cs). Sa propriété *Text* représente le texte affiché par le composant.
- *TextBoxNom* et *TextBoxAge* sont de type [System.Web.UI.WebControls.TextBox]. La propriété *Text* d'un composant *TextBox* est le texte affiché dans la zone de saisie.
- la méthode *Trim()* élimine les espaces qui peuvent précéder ou suivre une chaîne de caractères

Comme il a été expliqué précédemment, lorsque la méthode *ButtonValider_Click* est exécutée, les composants de la page ont la valeur qu'ils avaient lorsque la page a été postée par l'utilisateur. Les propriétés *Text* des deux *TextBox* ont donc pour valeur les textes saisis par l'utilisateur dans le navigateur.

Voici un exemple :





- en [1], les valeurs postées
- en [2], la réponse du serveur.
- en [3], les *TextBox* ont retrouvé leur valeur postée par le mécanisme du VIEWSTATE activé

- en [4], les messages du composant *ListBoxEvts* sont issus des méthodes *Page_Init*, *Page_Load*, *ButtonValider_Click* et d'un VIEWSTATE inhibé
- en [5], le composant *LabelPost* a obtenu sa valeur par la méthode *ButtonValider_Click*. On a bien récupéré les deux valeurs saisies par l'utilisateur dans les deux *TextBox* [1].

On voit ci-dessus que la valeur postée pour l'âge est la chaîne "yy", une valeur illégale. Nous allons rajouter à la page des composants appelés **validateurs**. Ils servent à vérifier la validité de données postées. Cette validité peut être vérifiée à deux endroits :

- **sur le client**. Une option de configuration du validateur permet de demander ou non que les tests soient faits sur le navigateur. Ils sont alors faits par du code JavaScript embarqué dans la page Html. Lorsque l'utilisateur fait un POST des valeurs saisies dans le formulaire, celles-ci sont tout d'abord vérifiées par le code Javascript. Si un des tests échoue, le POST n'est pas effectué. On évite ainsi un aller-retour avec le serveur rendant ainsi la page plus réactive.
- **sur le serveur**. Si les tests côté client peuvent être facultatifs, côté serveur ils sont obligatoires qu'il y ait eu vérification ou non côté client. En effet, lorsqu'une page reçoit des valeurs postées, elle n'a pas possibilité de savoir si elles ont été vérifiées par le client avant leur envoi. Côté serveur, le développeur doit donc **toujours** vérifier la validité des données postées.

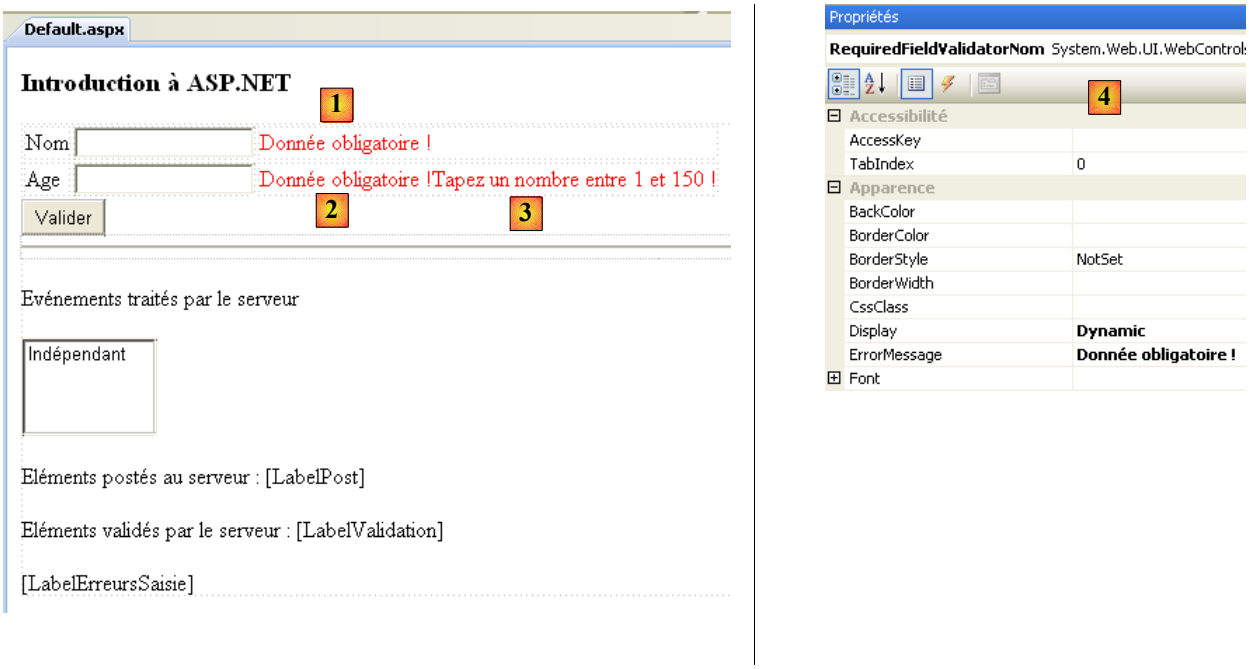
La page [Default.aspx] évolue comme suit :

```
1.  <%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
    Inherits="Intro._Default" %>
2.
3.  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4.  <html xmlns="http://www.w3.org/1999/xhtml">
5.  <head runat="server">
6.    <title>Introduction ASP.NET</title>
7.  </head>
8.  <body>
9.    <h3>Introduction à ASP.NET</h3>
10.   <form id="form1" runat="server">
11.   <div>
12.     <table>
13.       <tr>
14.         <td>
15.           Nom</td>
16.         <td>
17.           <asp:TextBox ID="TextBoxNom" runat="server"></asp:TextBox>
18.         </td>
19.         <td>
20.           <asp:RequiredFieldValidator ID="RequiredFieldValidatorNom" runat="server"
21.             ControlToValidate="TextBoxNom" Display="Dynamic"
22.             ErrorMessage="Donnée obligatoire !"></asp:RequiredFieldValidator>
23.         </td>
24.       </tr>
25.       <tr>
26.         <td>
27.           Age</td>
28.         <td>
29.           <asp:TextBox ID="TextBoxAge" runat="server"></asp:TextBox>
30.         </td>
31.         <td>
32.           <asp:RequiredFieldValidator ID="RequiredFieldValidatorAge" runat="server"
33.             ControlToValidate="TextBoxAge" Display="Dynamic"
34.             ErrorMessage="Donnée obligatoire !"></asp:RequiredFieldValidator>
35.           <asp:RangeValidator ID="RangeValidatorAge" runat="server"
36.             ControlToValidate="TextBoxAge" Display="Dynamic"
37.             ErrorMessage="Tapez un nombre entre 1 et 150 !" MaximumValue="150"
38.             MinimumValue="1" Type="Integer"></asp:RangeValidator>
39.         </td>
40.       </tr>
41.     </table>
42.   </div>
43.   <asp:Button ID="ButtonValider" runat="server" onclick="ButtonValider_Click"
44.     Text="Valider" CausesValidation="False"/>
45.   <hr />
46.   <p>
47.     Evénements traités par le serveur</p>
48.   <p>
49.     <asp:ListBox ID="ListBoxEvts" runat="server" EnableViewState="False">
50.     </asp:ListBox>
51.   </p>
```

```
52.   <p>
53.     Eléments postés au serveur :
54.     <asp:Label ID="LabelPost" runat="server"></asp:Label>
55.   </p>
56.   <p>
57.     Eléments validés par le serveur :
58.     <asp:Label ID="LabelValidation" runat="server"></asp:Label>
59.   </p>
60.    <asp:Label ID="LabelErreursSaisie" runat="server" ForeColor="Red"></asp:Label>
61.   </form>
62. </body>
63. </html>
```

Les validateurs ont été rajoutés aux lignes 20, 32 et 35. Ligne 58, un composant *Label* est utilisé pour afficher les valeurs postées valides. Ligne 60, un un composant *Label* est utilisé pour afficher un message d'erreur s'il y a des erreurs de saisie.

La page [Default.aspx] en mode [Design] est la suivante :



- les composants [1] et [2] sont de type *RequiredFieldValidator*. Ce validateur vérifie qu'un champ de saisie est non vide.
- le composant [3] est de type *RangeValidator*. Ce validateur vérifie qu'un champ de saisie contient une valeur entre deux bornes.
- en [4], les propriétés du validateur [1].

Nous allons présenter les deux types de validateurs au travers de leurs balises dans le code de la page [Default.aspx] :

```
        <asp:RequiredFieldValidator ID="RequiredFieldValidatorNom" runat="server"
          ControlToValidate="TextBoxNom" Display="Dynamic"
ErrorMessage="Donnée obligatoire !"></asp:RequiredFieldValidator>
```

- **ID** : l'identifiant du composant
- **ControlToValidate** : le nom du composant dont la valeur est vérifiée. Ici on veut que le composant *TextBoxNom* n'ait pas une valeur vide (chaîne vide ou suite d'espaces)
- **ErrorMessage** : message d'erreur à afficher dans le validateur en cas de donnée invalide.
- **EnableClientScript** : booléen indiquant si le validateur doit être exécuté également côté client. Cet attribut a la valeur *True* par défaut lorsqu'il n'est pas explicitement positionné comme ci-dessus.
- **Display** : mode d'affichage du validateur. Il y a deux modes :
  - *static* (défaut) : le validateur occupe de la place sur la page même s'il n'affiche pas de message d'erreur
  - *dynamic* : le validateur n'occupe pas de place sur la page s'il n'affiche pas de message d'erreur.

```
        <asp:RangeValidator ID="RangeValidatorAge" runat="server"
          ControlToValidate="TextBoxAge" Display="Dynamic"
          ErrorMessage="Tapez un nombre entre 1 et 150 !" MaximumValue="150"
          MinimumValue="1" Type="Integer"></asp:RangeValidator>
```

- **Type** : le type de la donnée vérifiée. Ici l'âge est un entier.
- **MinimumValue**, **MaximumValue** : les bornes dans lesquelles doit se trouver la valeur vérifiée

La configuration du composant qui provoque le POST joue un rôle dans le mode de validation. Ici ce composant est le bouton [Valider] :

```
  <asp:Button ID="ButtonValider" runat="server" onclick="ButtonValider_Click"  Text="Valider"
CausesValidation="True" />
```

- **CausesValidation** : fixe le mode automatique ou nom des validations côté serveur. Cet attribut a la valeur par défaut "*True*" s'il n'est pas explicitement mentionné. Dans ce cas,
  - côté client, les validateurs ayant *EnableClientScript* à *True* sont exécutés. Le POST n'a lieu que si tous les validateurs côté client réussissent.
  - côté serveur, tous les validateurs présents sur la page sont automatiquement exécutés avant le traitement de l'événement qui a provoqué le POST. Ici, ils seraient exécutés avant l'exécution de la méthode *ButtonValider_Click*. Dans cette méthode, il est possible de savoir si toutes les validations ont réussi ou non. **Page.IsValid** est "True" si elles ont toutes réussi, "False" sinon. Dans ce dernier cas, on peut arrêter le traitement de l'événement qui a provoqué le POST. La page postée est renvoyée telle qu'elle a été saisie. Les validateurs ayant échoué affichent alors leur message d'erreur (attribut *ErrorMessage*).

  Si **CausesValidation** a la valeur *False*, alors
  - côté client, aucun validateur n'est exécuté
  - côté serveur, c'est au développeur de demander lui-même l'exécution des validateurs de la page. Il le fait avec la méthode *Page.Validate()*. Selon le résultat des validations, cette méthode positionne la propriété **Page.IsValid** à "True" ou "False".

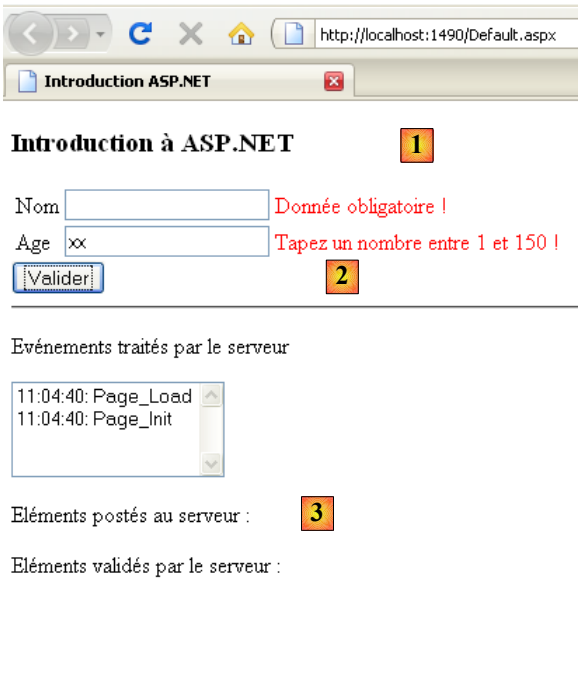
Dans [Default.aspx.cs] le code de traitement de *ButtonValider_Click* évolue comme suit :

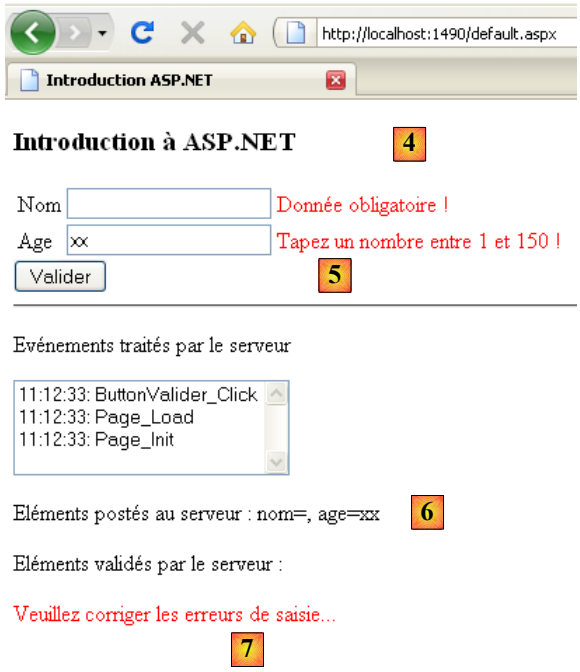
```
1.  protected void ButtonValider_Click(object sender, EventArgs e)
2.      {
3.        // on note l'événement
4.        ListBoxEvts.Items.Insert(0, string.Format("{0}: ButtonValider_Click",
    DateTime.Now.ToString("hh:mm:ss")));
5.        // on affiche le nom et l'âge
6.        LabelPost.Text = string.Format("nom={0}, age={1}", TextBoxNom.Text.Trim(),
    TextBoxAge.Text.Trim());
7.        // la page est-elle valide ?
8.        Page.Validate();
9.        if (!Page.IsValid)
10.       {
11.         // msg d'erreur global
12.         LabelErreursSaisie.Text = "Veuillez corriger les erreurs de saisie...";
13.         LabelErreursSaisie.Visible = true;
14.         return;
15.       }
16.       // on cache le msg d'erreur
17.       LabelErreursSaisie.Visible = false;
18.       // on affiche le nom et l'âge validés
19.       LabelValidation.Text = string.Format("nom={0}, age={1}", TextBoxNom.Text.Trim(),
    TextBoxAge.Text.Trim());
20. }
```

Dans le cas où le bouton [Valider] a son attribut *CausesValidation* à *True* et les validateurs leur attribut *EnableClientScript* à *True*, la méthode *ButtonValider_Click* n'est exécutée que lorsque les valeurs postées sont valides. On peut se demander alors le sens du code que l'on trouve à partir de la ligne 8. Il faut se rappeler qu'il est toujours possible d'écrire un client programmé qui poste des valeurs non vérifiées à la page [Default.aspx]. Donc celle-ci doit toujours refaire les tests de validité.

- ligne 8 : lance l'exécution de tous les validateurs de la page. Dans le cas où le bouton [Valider] a son attribut *CausesValidation* à *True*, ceci est fait automatiquement et il n'y a pas lieu de le refaire. Il y a ici redondance.
- lignes 9-15 : cas où l'un des validateurs a échoué
- lignes 16-19 : cas où tous les validateurs ont réussi

Voici deux exemples d'exécution :

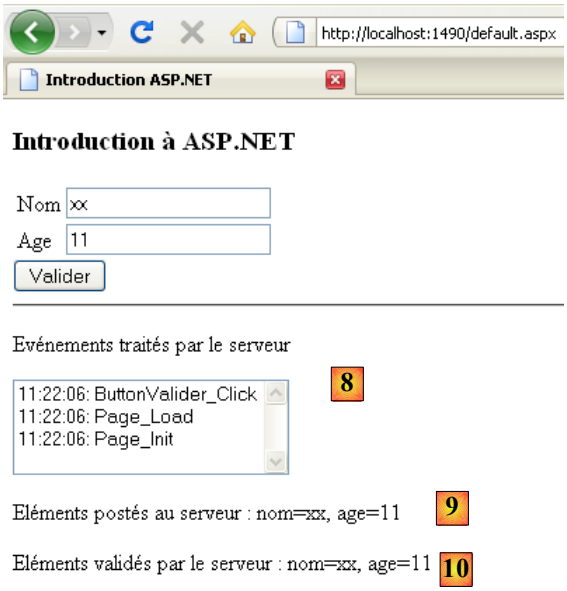




- en [1], un exemple d'exécution dans le cas où :
  - le bouton [Valider] a sa propriété *CausesValidation* à *True*
  - les validateurs ont leur propriété *EnableClientScript* à *True*

  Les messages d'erreurs [2] ont été affichés par les validateurs exécutés côté client par le code Javascript de la page. Il n'y a pas eu de POST vers le serveur comme le montre le label des éléments postés [3].
- en [4], un exemple d'exécution dans le cas où :
  - le bouton [Valider] a sa propriété *CausesValidation* à *False*
  - les validateurs ont leur propriété *EnableClientScript* à *False*

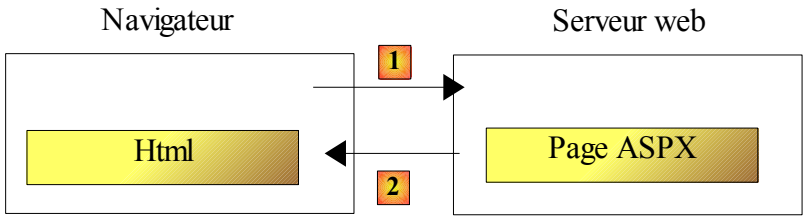
  Les messages d'erreurs [5] ont été affichés par les validateurs exécutés côté serveur. Comme le montre [6], il y a bien eu un POST vers le serveur. En [7], le message d'erreur affiché par la méthode [ButtonValider_Click] dans le cas d'erreurs de saisie.



- en [8] un exemple obtenu avec des données valides. [9,10] montrent que les éléments postés ont été validés. Lorsqu'on fait des tests répétés, il faut mettre à *False* la propriété *EnableViewState* du label [LabelValidation] afin que le message de validation ne reste pas affiché au fil des exécutions.

2.4 Gestion des données de portée Application

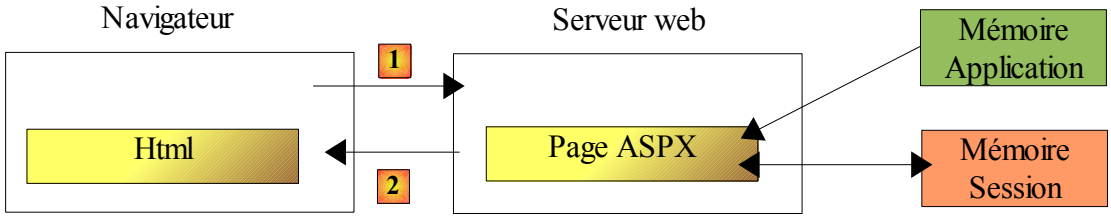
Revenons sur l'architecture d'exécution d'une page ASPX :



La classe de la page ASPX est instanciée au début de la requête du client et détruite à la la fin de celle-ci. Aussi elle ne peut servir à mémoriser des données entre deux requêtes. On peut vouloir mémoriser deux types de données :

- des données partagées par tous les utilisateurs de l'application web. Ce sont en général des données en lecture seule. Trois fichiers sont utilisés pour mettre en oeuvre ce partage de données :
  - [Web.Config] : le fichier de configuration de l'application
  - [Global.asax, Global.asax.cs] : permettent de définir une classe, appelée classe globale d'application, dont la durée de vie est celle de l'application, ainsi que des gestionnaires pour certains événements de cette même application.

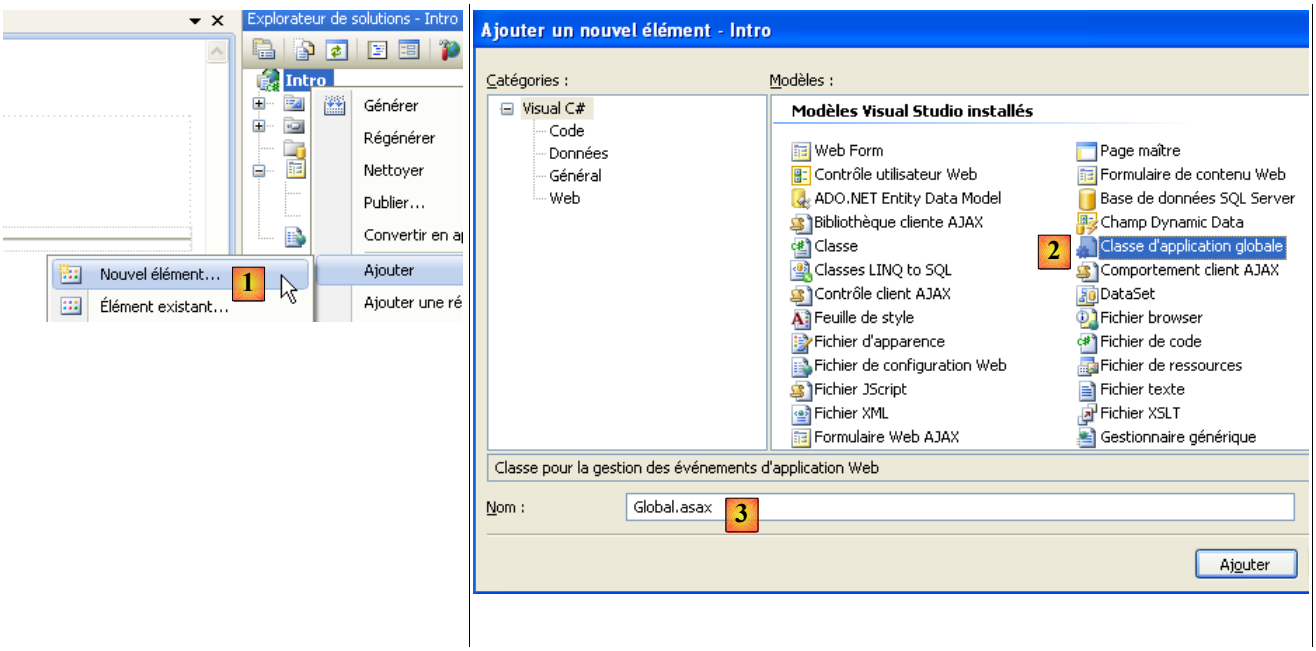
  La classe globale d'application permet de définir des données qui seront disponibles pour toutes les requêtes de tous les utilisateurs.
- des données partagées par les requêtes d'**un même client**. Ces données sont mémorisées dans un objet appelée **Session**. On parle alors de **session client** pour désigner la mémoire du client. Toutes les requêtes d'un client ont accès à cette session. Elles peuvent y stocker et y lire des informations



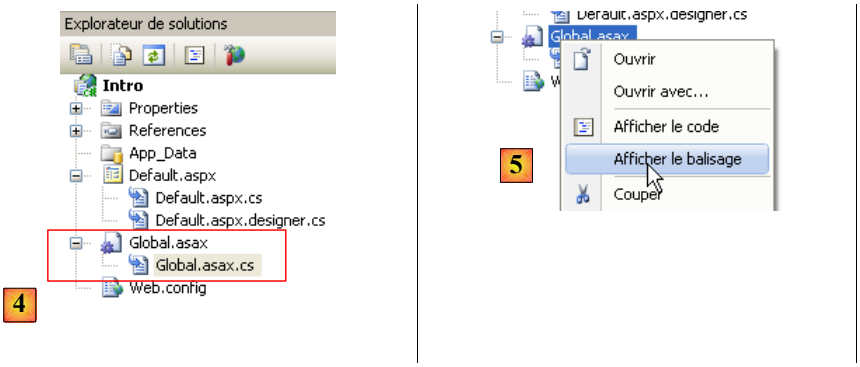
Ci-dessus, nous montrons les types de mémoire auxquels a accès une page ASPX :

- la mémoire de l'application qui contient la plupart du temps des données en lecture seule et qui est accessible à tous les utilisateurs.
- la mémoire d'un utilisateur particulier, ou session, qui contient des données en lecture / écriture et qui est accessible aux requêtes successives d'un même utilisateur.
- non représentée ci-dessus, il existe une mémoire de requête, ou contexte de requête. La requête d'un utilisateur peut être traitée par plusieurs pages ASPX successives. Le contexte de la requête permet à une page 1 de transmettre de l'information à une page 2.

Nous nous intéressons ici aux données de portée *Application*, celles qui sont partagées par tous les utilisateurs. La classe globale de l'application peut être créée comme suit :



- en [1], on ajoute un nouvel élément au projet
- en [2], on ajoute la classe d'application globale
- en [3], on garde le nom par défaut [Global.asax] pour le nouvel élément



- en [4], deux nouveaux fichiers ont été ajoutés au projet
- en [5], on affiche le balisage de [Global.asax]

```
<%@ Application Codebehind="Global.asax.cs" Inherits="Intro.Global" Language="C#" %>
```

- la balise *Application* remplace la balise *Page* qu'on avait pour [Default.aspx]. Elle identifie la classe d'application globale
- **Codebehind** : définit le fichier dans lequel est définie la classe d'application globale
- **Inherits** : définit le nom de cette classe

La classe **Intro.Global** générée est la suivante :

```
1.  using System;
2.
3.  namespace Intro
4.  {
5.    public class Global : System.Web.HttpApplication
6.    {
7.
8.      protected void Application_Start(object sender, EventArgs e)
9.      {
10.
```

```
11.     }
12.
13.     protected void Session_Start(object sender, EventArgs e)
14.     {
15.
16.     }
17.
18.     protected void Application_BeginRequest(object sender, EventArgs e)
19.     {
20.
21.     }
22.
23.     protected void Application_AuthenticateRequest(object sender, EventArgs e)
24.     {
25.
26.     }
27.
28.     protected void Application_Error(object sender, EventArgs e)
29.     {
30.
31.     }
32.
33.     protected void Session_End(object sender, EventArgs e)
34.     {
35.
36.     }
37.
38.     protected void Application_End(object sender, EventArgs e)
39.     {
40.
41.     }
42.   }
43. }
```

- ligne 5 : la classe globale d'application dérive de la classe *HttpApplication*

La classe est générée avec des squelettes de gestionnaires d'événements de l'application :

- lignes 8, 38 : gèrent les événements *Application_Start* (démarrage de l'application) et *Application_End* (fin de l'application lorsque le serveur web s'arrête ou lorsque l'administrateur décharge l'application)
- lignes 13, 33 : gèrent les événements *Session_Start* (démarrage d'une nouvelle session client à l'arrivée d'un nouveau client ou à l'expiration d'une session existante) et *Session_End* (fin d'une session client soit explicitement par programmation soit implicitement par dépassement de la durée autorisée pour une session).
- ligne 28 : gère l'événement *Application_Error* (apparition d'une exception non gérée par le code de l'application et remontée jusqu'au serveur)
- ligne 18 : gère l'événement *Application_BeginRequest* (arrivée d'une nouvelle requête).
- ligne 23 : gère l'événement *Application_AuhenticateRequest* (se produit lorsqu'un utilisateur s'est authentifié).

La méthode [Application_Start] est souvent utilisée pour initialiser l'application à partir d'informations contenues dans [Web.Config]. Celui généré à la création initiale d'un projet a l'allure suivante :

```
<?xml version="1.0" encoding="utf-8"?>

<configuration>
    <configSections>
...
    </configSections>

    <appSettings/>
    <connectionStrings/>

    <system.web>
...
    </system.web>

    <system.codedom>
....
    </system.codedom>

    <!--
        La section system.webServer est requise pour exécuter ASP.NET AJAX sur Internet
        Information Services 7.0.  Elle n'est pas nécessaire pour les versions précédentes d'IIS.
    -->
    <system.webServer>
```

```
...
    </system.webServer>

    <runtime>
....
    </runtime>

</configuration>
```

Pour notre application actuelle, ce fichier est inutile. Si on le supprime ou le renomme, l'application continue à fonctionner normalement. Nous allons nous intéresser aux balises des lignes 8 et 9 :

- **<appsettings>** permet de définir un dictionnaire d'informations
- **<connectionStrings>** permet de définir des chaînes de connexion à des bases de données

Considérons le fichier [Web.config] suivant :

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.
3.  <configuration>
4.      <configSections>
5.  ...
6.      </configSections>
7.
8.    <appSettings>
9.      <add key="cle1" value="valeur1"/>
10.     <add key="cle2" value="valeur2"/>
11.   </appSettings>
12.   <connectionStrings>
13.     <add connectionString="connectionString1" name="conn1"/>
14.   </connectionStrings>
15.
16.     <system.web>
17. ...
18.
```

Ce fichier peut être exploité par la classe globale d'application suivante :

```
1.  using System;
2.  using System.Configuration;
3.
4.  namespace Intro
5.  {
6.    public class Global : System.Web.HttpApplication
7.    {
8.      public static string Param1 { get; set; }
9.      public static string Param2 { get; set; }
10.     public static string ConnString1 { get; set; }
11.     public static string Erreur { get; set; }
12.
13.     protected void Application_Start(object sender, EventArgs e)
14.     {
15.       try
16.       {
17.         Param1 = ConfigurationManager.AppSettings["cle1"];
18.         Param2 = ConfigurationManager.AppSettings["cle2"];
19.         ConnString1 = ConfigurationManager.ConnectionStrings["conn1"].ConnectionString;
20.       }
21.       catch (Exception ex)
22.       {
23.         Erreur = string.Format("Erreur de configuration : {0}", ex.Message);
24.       }
25.     }
26.
27.     protected void Session_Start(object sender, EventArgs e)
28.     {
29.
30.     }
31.
32.   }
33. }
```

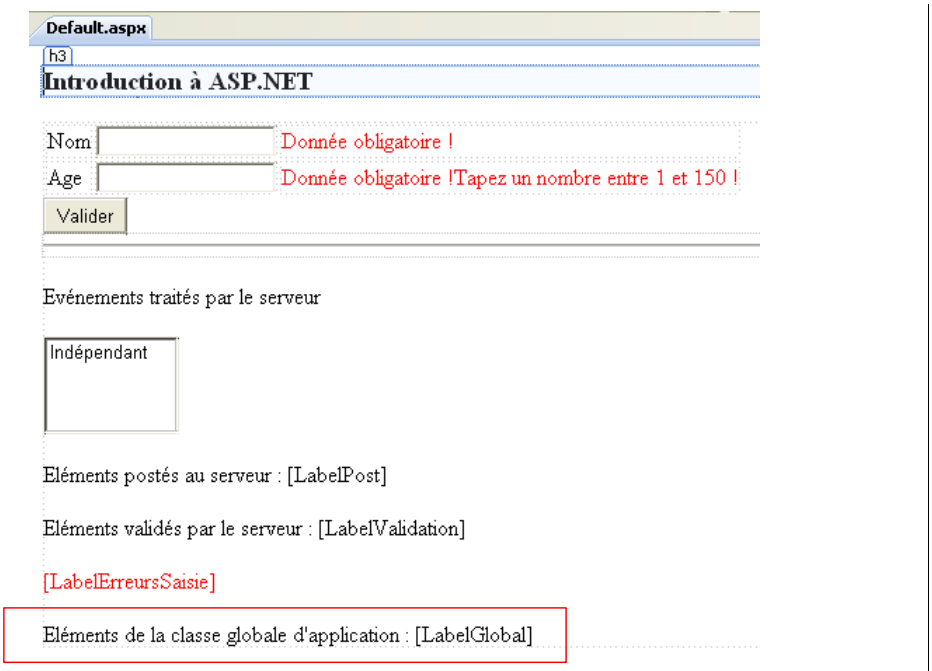
- lignes 8-11 : quatre propriétés statiques P. Comme la durée de vie de la classe *Global* est celle de l'application, toute requête faite à l'application aura accès à ces propriétés P via la syntaxe *Global.P*.

- lignes 17-19 : le fichier [Web.config] est accessible via la classe [System.Configuration.ConfigurationManager]
- lignes 17-18 : récupère les éléments de la balise <*appSettings*> du fichier [Web.config] via l'attribut *key*.
- ligne 19 : récupère les éléments de la balise <*connectionStrings*> du fichier [Web.config] via l'attribut *name*.

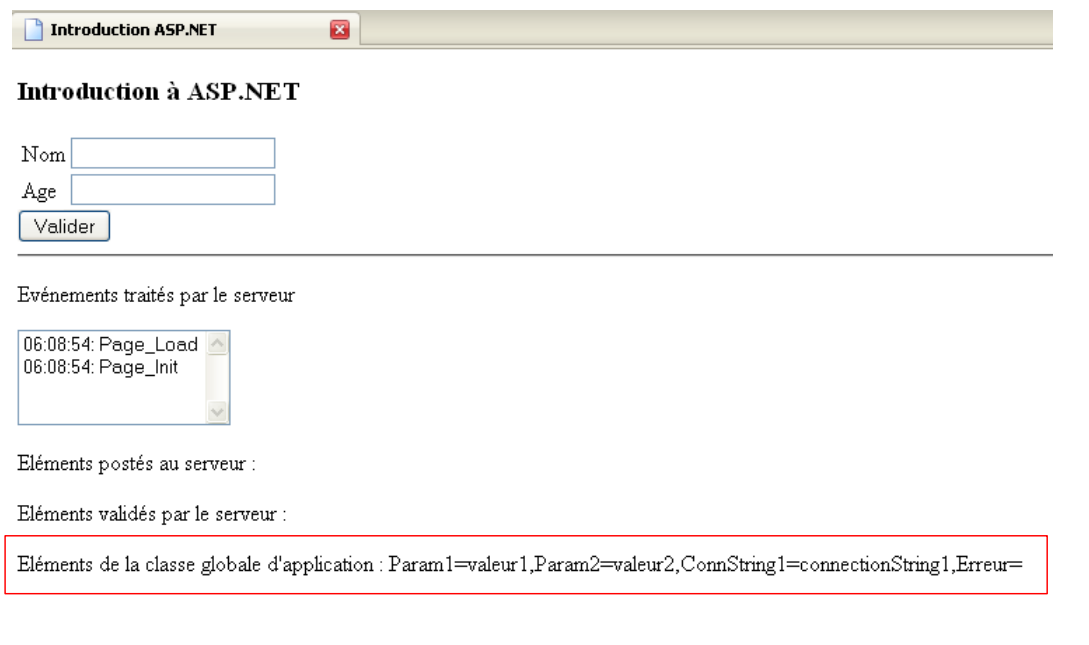
Les attributs statiques des lignes 8-11 sont accessibles de n'importe quel gestionnaire d'événement des pages ASPX chargées. Nous les utilisons dans le gestionnaire [Page_Load] de la page [Default.aspx] :

```
1.      protected void Page_Load(object sender, EventArgs e)
2.      {
3.        // on note l'événement
4.        ListBoxEvts.Items.Insert(0, string.Format("{0}: Page_Load", DateTime.Now.ToString("hh:mm:ss")));
5.        // on récupère les informations de la classe globale d'application
6.        LabelGlobal.Text = string.Format("Param1={0},Param2={1},ConnString1={2},Erreur={3}", Global.Param1,
    Global.Param2, Global.ConnString1, Global.Erreur);
7.  }
```

- ligne 6 : les quatre attributs statiques de la classe globale d'application sont utilisés pour alimenter un nouveau label de la page [Default.aspx]



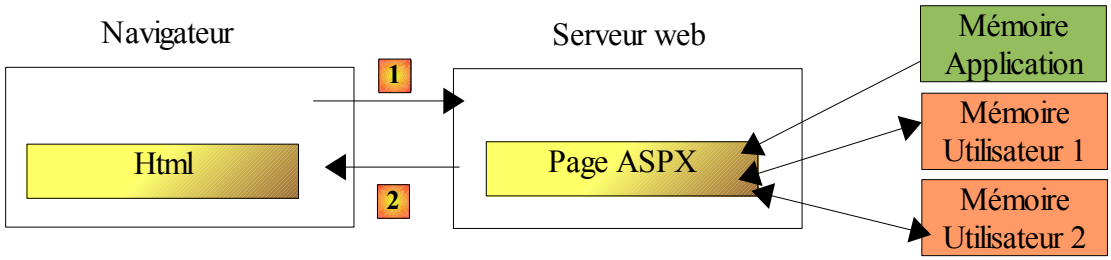
A l'exécution, nous obtenons le résultat suivant :



Ci-dessus, nous voyons que les paramètres de [web.config] ont été correctement récupérés. La classe globale d'application est le bon endroit pour stocker des informations partagées par tous les utilisateurs.

2.5 Gestion des données de portée Session

Nous nous intéressons ici à la façon de mémoriser des informations au fil des requêtes d'un utilisateur donné :



Chaque utilisateur a sa propre mémoire qu'on appelle sa session.

Nous avons vu que la classe d'application globale disposait de deux gestionnaires pour gérer les événements :

- *Session_Start* : début d'une session
- *Session_end* : fin d'une session

Le mécanisme de la session est mis en oeuvre de la façon suivante :

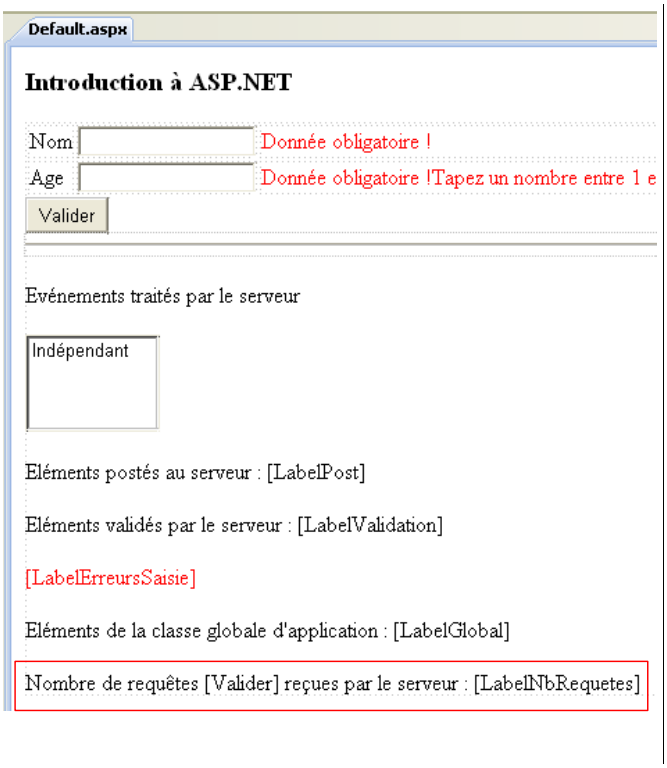
- lors de la première requête d'un utilisateur, le serveur web crée un jeton de session qu'il attribue à l'utilisateur. Ce jeton est une suite de caractères unique pour chaque utilisateur. Il est envoyé par le serveur dans la réponse faite à la première requête de l'utilisateur.
- lors des requêtes suivantes, l'utilisateur (le navigateur web) inclut dans sa requête le jeton de session qu'on lui a attribué. Aussi le serveur web est-il capable de le reconnaître.
- une session a une durée de vie. Lorsque le serveur web reçoit une requête d'un utilisateur, il calcule le temps qui s'est écoulé depuis la requête précédente. Si ce temps dépasse la durée de vie de la session, une nouvelle session est créée pour l'utilisateur. Les données de la précédente session sont perdues. Avec le serveur web IIS (Internet Information Server) de Microsoft, les sessions ont par défaut une durée de vie de 20 mn. Cette valeur peut être changée par l'administrateur du serveur web.
- le serveur web sait qu'il a affaire à la première requête d'un utilisateur parce que cette requête ne comporte pas de jeton de session. C'est la seule.

Toute page ASP.NET a accès à la session de l'utilisateur via la propriété **Session** de la page, de type [System.Web.SessionState.HttpSessionState]. Nous utiliserons les propriétés P et méthodes M suivantes de la classe *HttpSessionState* :

Nom	Type	Rôle
Item[String clé]	P	La session peut être construite comme un dictionnaire. *Item[clé]* est l'élément de la session identifié par *clé*. Au lieu d'écrire [*HttpSessionState*].*Item[clé]*, on peut également écrire [*HttpSessionState*].*[clé]*.
Clear	M	vide le dictionnaire de la session
Abandon	M	termine la session. La session n'est alors plus valide. Une nouvelle session démarrera avec la prochaine requête de l'utilisateur.

Comme exemple de mémoire utilisateur, nous allons compter le nombre de fois qu'un utilisateur clique sur le bouton [Valider]. Pour obtenir ce résultat, il faut maintenir un compteur dans la session de l'utilisateur.

La page [Default.aspx] évolue comme suit :



La classe globale d'application [Global.asax.cs] évolue comme suit :

```
1.  using System;
2.  using System.Configuration;
3.
4.  namespace Intro
5.  {
6.    public class Global : System.Web.HttpApplication
7.    {
8.      public static string Param1 { get; set; }
9.  ...
10.
11.     protected void Application_Start(object sender, EventArgs e)
12.     {
13. ...
14.     }
15.
16.     protected void Session_Start(object sender, EventArgs e)
17.     {
18.       // compteur de requêtes
19.       Session["nbRequêtes"] = 0;
20.     }
21.
```

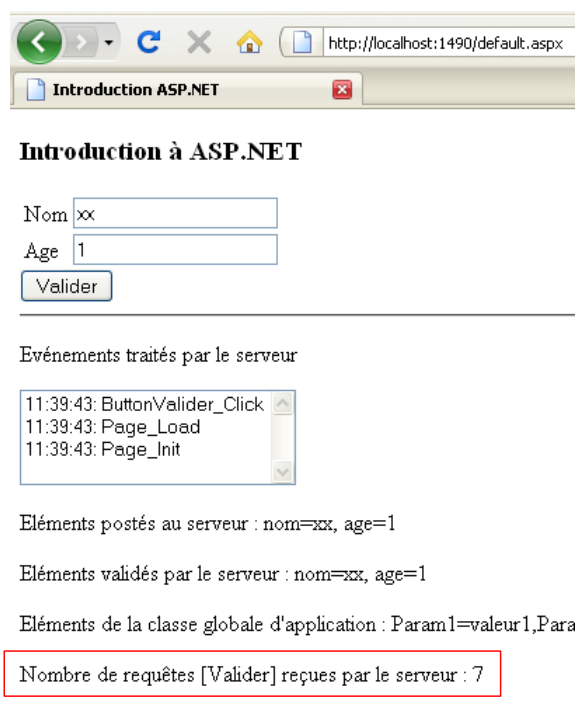
```
22.   }
23. }
```

Ligne 19, on utilise la session de l'utilisateur pour y stocker un compteur de requêtes identifié par la clé *"nbRequêtes"*. Ce compteur est mis à jour par le gestionnaire [ButtonValider_Click] de la page [Default.aspx] :

```
1.  using System;
2.
3.  namespace Intro
4.  {
5.    public partial class _Default : System.Web.UI.Page
6.    {
7.  ....
8.
9.      protected void ButtonValider_Click(object sender, EventArgs e)
10.     {
11.       // on note l'événement
12.       ListBoxEvts.Items.Insert(0, string.Format("{0}: ButtonValider_Click",
    DateTime.Now.ToString("hh:mm:ss")));
13.       // on affiche le nom et l'âge postés
14.       LabelPost.Text = string.Format("nom={0}, age={1}", TextBoxNom.Text.Trim(),
    TextBoxAge.Text.Trim());
15.       // nombre de requêtes
16.       Session["nbRequêtes"] = (int)Session["nbRequêtes"] + 1;
17.       LabelNbRequetes.Text = Session["nbRequêtes"].ToString();
18.       // la page est-elle valide ?
19.       Page.Validate();
20.       if (!Page.IsValid)
21.       {
22. ...
23.       }
24. ...
25.     }
26.   }
27. }
```

- ligne 16 : on incrémente le compteur de requêtes
- ligne 17 : le compteur est affiché sur la page

Voici un exemple d'exécution :



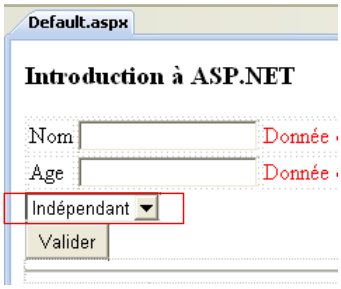
2.6 Gestion du GET / POST dans le chargement d'une page

Nous avons dit qu'il y avait deux types de requêtes vers une page ASPX :

- la requête initiale du navigateur faite avec une commande HTTP GET. Le serveur répond en envoyant la page demandée. Nous supposerons que cette page est un formulaire, c.a.d. que dans la page ASPX envoyée, il y a une balise <form runat="server"...>.
- les requêtes suivantes faites par le navigateur en réaction à certaines actions de l'utilisateur sur le formulaire. Le navigateur fait alors une requête HTTP POST.

Que ce soit sur une demande GET ou une demande POST, la méthode [Page_Load] est exécutée. Lors du GET, cette méthode est habituellement utilisée pour initialiser la page envoyée au navigateur client. Ensuite, par le mécanisme du VIEWSTATE, la page reste initialisée et n'est modifiée que par les gestionnaires des événements qui provoquent les POST. Il n'y a pas lieu de réinitialiser la page dans *Page_Load*. D'où le besoin pour cette méthode de savoir si la requête du client est un GET ou un POST.

Examinons l'exemple suivant. On ajoute une liste déroulante à la page [Default.aspx]. Le contenu de cette liste sera défini dans le gestionnaire *Page_Load* de la requête GET :



La liste déroulante est déclarée dans [Default.aspx.designer.cs] de la façon suivante :

```
        protected global::System.Web.UI.WebControls.DropDownList DropDownListNoms;
```

Nous utiliserons les méthodes M et propriétés P suivantes de la classe [DropDownList] :

Nom	Type	Rôle
Items	P	la collection de type *ListItemCollection* des éléments de type *ListItem* de la liste déroulante
SelectedIndex	P	l'index, partant de 0, de l'élément sélectionné dans la liste déroulante lorsque le formulaire est posté
SelectedItem	P	l'élément de type *ListItem* sélectionné dans la liste déroulante lorsque le formulaire est posté
SelectedValue	P	la valeur de type *string* de l'élément de type *ListItem* sélectionné dans la liste déroulante lorsque le formulaire est posté. Nous allons définir prochainement cette notion de valeur.

La classe **ListItem** des éléments d'une liste déroulante sert à générer les balises <*option*> de la balise Html <*select*> :

```
1.  <select ....>
2.     <option value="val1">texte1</option>
3.     <option value="val2">texte2</option>
4.  ....
5.  </select>
```

Dans la balise <*option*>

- *textei* est le texte affiché dans la liste déroulante
- *vali* est la valeur postée par le navigateur si *textei* est le texte sélectionné dans la liste déroulante

Chaque option peut être générée par un objet *LisItem* construit à l'aide du constructeur *ListItem(string texte, string valeur)*.

Dans [Default.aspx.cs], le code du gestionnaire [Page_Load] évolue comme suit :

```
1.      protected void Page_Load(object sender, EventArgs e)
2.      {
3.        // on note l'événement
```

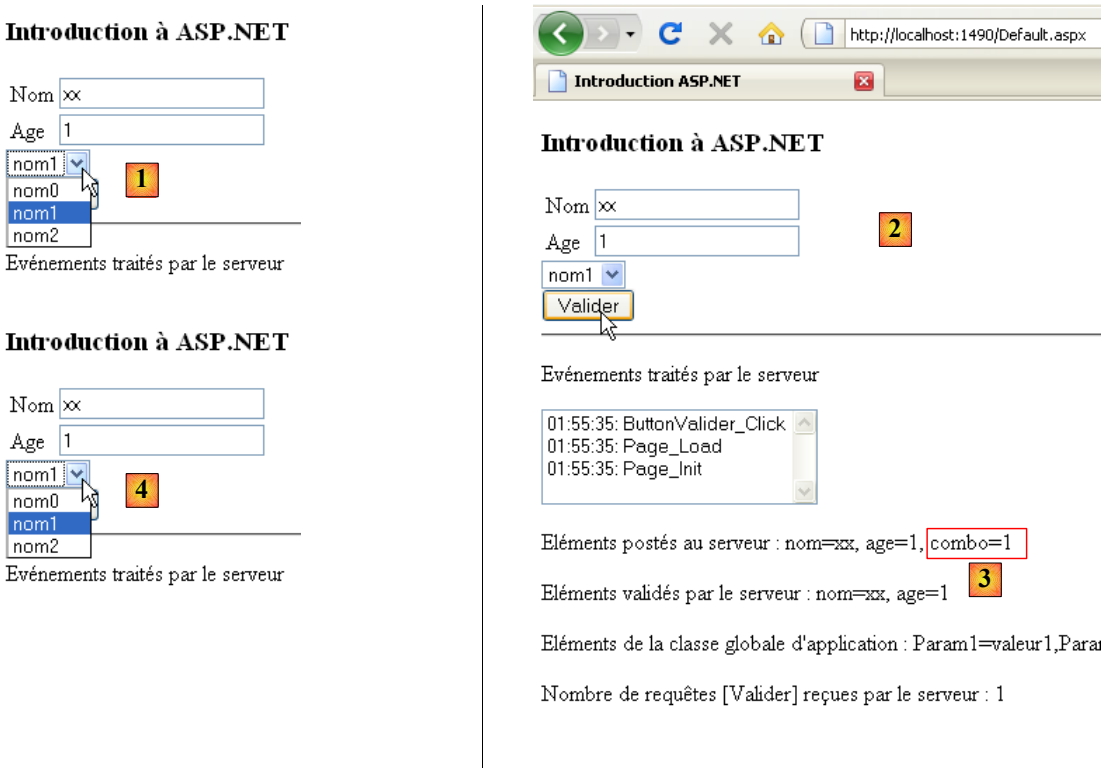
```
4.        ...
5.        // on récupère les informations de la classe globale d'application
6.        ...
7.        // initialisation du combo des noms uniquement lors du GET initial
8.        if (!IsPostBack)
9.        {
10.         for (int i = 0; i < 3; i++)
11.         {
12.           DropDownListNoms.Items.Add(new ListItem("nom"+i,i.ToString()));
13.         }
14.       }
15. }
```

- ligne 8 : la classe *Page* a un attribut *IsPostBack* de type booléen. A vrai, il signifie que la requête de l'utilisateur est un POST. Les lignes 10-13 ne sont donc exécutées que sur le GET initial du client.
- ligne 12 : on ajoute à la liste [DropDownListNoms], un élément de type *ListItem(string texte, string value)*. Le texte affiché pour le (i+1)ème élément sera *nomi* et la valeur postée pour cet élément s'il est sélectionné sera *i*.

Le gestionnaire [ButtonValider_Click] est modifié afin d'afficher la valeur postée par la liste déroulante :

```
1.      protected void ButtonValider_Click(object sender, EventArgs e)
2.      {
3.        // on note l'événement
4.  ...
5.        // on affiche les valeurs postées
6.        LabelPost.Text = string.Format("nom={0}, age={1}, combo={2}", TextBoxNom.Text.Trim(),
    TextBoxAge.Text.Trim(), DropDownListNoms.SelectedValue);
7.        // nombre de requêtes
8.  ...
9.  }
```

Ligne 6, la valeur postée pour la liste [DropDownListNoms] est obtenue avec la propriété *SelectedValue* de la liste. Voici un exemple d'exécution :



- en [1], le contenu de la liste déroulante après le GET initial et juste avant le 1er POST
- en [2], la page après le 1er POST.
- en [3], la valeur postée pour la liste déroulante. Correspond à l'attribut *value* du *ListItem* sélectionné dans la liste.

- en [4], la liste déroulante. Elle contient les mêmes éléments qu'après le GET initial. C'est le mécanisme du VIEWSTATE qui explique cela.

Pour comprendre l'interaction entre le VIEWSTATE de la liste *DropDownListNoms* et le test *if (! IsPostBack)* du gestionnaire *Page_Load* de [Default.aspx], le lecteur est invité à refaire le test précédent avec les configurations suivantes :

Cas	DropDownListNoms.EnableViewState	test if(! IsPostBack) dans Page_Load de [Default.aspx]
1	true	présent
2	false	présent
3	true	absent
4	false	absent

Les différents tests donnent les résultats suivants :

1. c'est le cas présenté plus haut
2. la liste est remplie lors du GET initial mais pas lors des POST qui suivent. Comme *EnableViewState* est à faux, la liste est vide après chaque POST
3. la liste est remplie aussi bien après le GET initial que lors des POST qui suivent. Comme *EnableViewState* est à *vrai*, on a 3 noms après le GET initial, 6 noms après le 1er POST, 9 noms après le 2ième POST, ...
4. la liste est remplie aussi bien après le GET initial que lors des POST qui suivent. Comme *EnableViewState* est à *faux*, la liste est remplie avec seulement 3 noms à chaque requête que celle-ci soit le GET initial ou les POST qui suivent. On retrouve le comportement du cas 1. Il y a donc deux façons d'obtenir le même résultat.

2.7 Gestion du VIEWSTATE des éléments d'une page ASPX

Par défaut, tous les éléments d'une page ASPX ont leur propriété *EnableViewState* à *True*. A chaque fois que la page ASPX est envoyée au navigateur client, elle contient le champ caché __VIEWSTATE qui a pour valeur une chaîne de caractères codant l'ensemble des valeurs des composants ayant leur propriété *EnableViewState* à *True*. Pour minimiser la taille de cette chaîne, on peut chercher à réduire le nombre de composants ayant leur propriété *EnableViewState* à *True*.

Rappelons comment les composants d'une page ASPX obtiennent leurs valeurs à l'issue d'un POST :

1. la page ASPX est instanciée. Les composants sont initialisés avec leurs valeurs de conception.
2. la valeur __VIEWSTATE postée par le navigateur est utilisée pour donner aux composants la valeur qu'ils avaient lorsque la page ASPX a été envoyée au navigateur la fois précédente.
3. les valeurs postées par le navigateur sont affectées aux composants
4. les gestionnaires d'événements sont exécutés. Ils peuvent modifier la valeur de certains composants.

De cette séquence, on déduit que les composants qui :

- ont leur valeur postée
- ont leur valeur modifiée par un gestionnaire d'événement

peuvent avoir leur propriété *EnableViewState* à *Faux* puisque leur valeur de VIEWSTATE (étape 2) va être modifiée par l'une des étapes 3 ou 4.

La liste des composants de notre page est disponible dans [Default.aspx.designer.cs] :

```
1.  namespace Intro {
2.      public partial class _Default {
3.          protected global::System.Web.UI.HtmlControls.HtmlForm form1;
4.          protected global::System.Web.UI.WebControls.TextBox TextBoxNom;
5.          protected global::System.Web.UI.WebControls.RequiredFieldValidator RequiredFieldValidatorNom;
6.          protected global::System.Web.UI.WebControls.TextBox TextBoxAge;
7.          protected global::System.Web.UI.WebControls.RequiredFieldValidator RequiredFieldValidatorAge;
8.          protected global::System.Web.UI.WebControls.RangeValidator RangeValidatorAge;
9.          protected global::System.Web.UI.WebControls.DropDownList DropDownListNoms;
10.         protected global::System.Web.UI.WebControls.Button ButtonValider;
11.         protected global::System.Web.UI.WebControls.ListBox ListBoxEvts;
12.         protected global::System.Web.UI.WebControls.Label LabelPost;
13.         protected global::System.Web.UI.WebControls.Label LabelValidation;
14.         protected global::System.Web.UI.WebControls.Label LabelErreursSaisie;
15.         protected global::System.Web.UI.WebControls.Label LabelGlobal;
16.         protected global::System.Web.UI.WebControls.Label LabelNbRequetes;
```

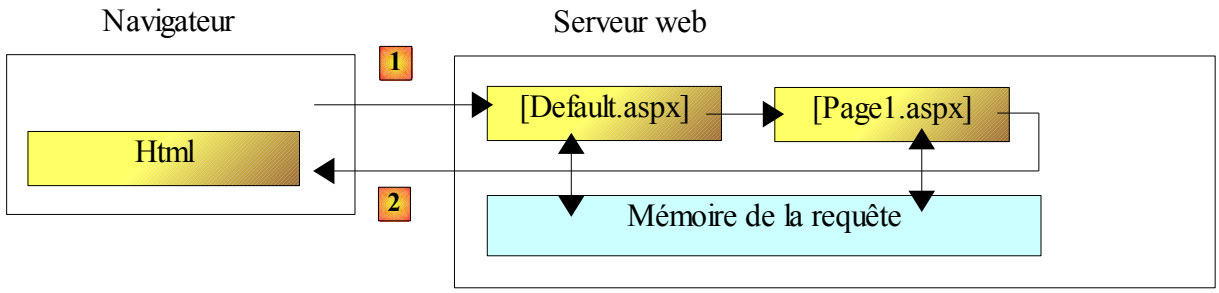
```
17.     }
18. }
```

La valeur de la propriété *EnableViewState* de ces composants pourrait être la suivante :

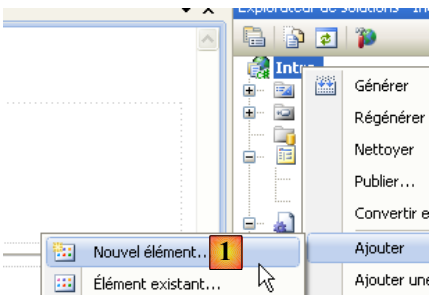
Composant	*Valeur postée*	*EnableViewState*	*Pourquoi*
TextBoxNom	valeur saisie dans le TextBox	False	la valeur du composant est postée
TextBoxAge	idem		
RequiredFieldValidatorNom	aucune	False	absence de notion de valeur du composant
RequiredFieldValidatorAge	idem		
RangeValidatorAge	idem		
LabelPost	aucune	False	obtient sa valeur par un gestionnaire d'événement
LabelValidation	idem		
LabelErreursSaisie	idem		
LabelGlobal	idem		
LabelNbRequetes	idem		
DropDownListNoms	"value" de l'élément sélectionné	True	on veut garder le contenu de la liste au fil des requêtes sans avoir à la régénérer
ListBoxEvts	"value" de l'élément sélectionné	False	le contenu de la liste est généré par un gestionnaire d'événement
ButtonValider	libellé du bouton	False	le composant garde sa valeur de conception

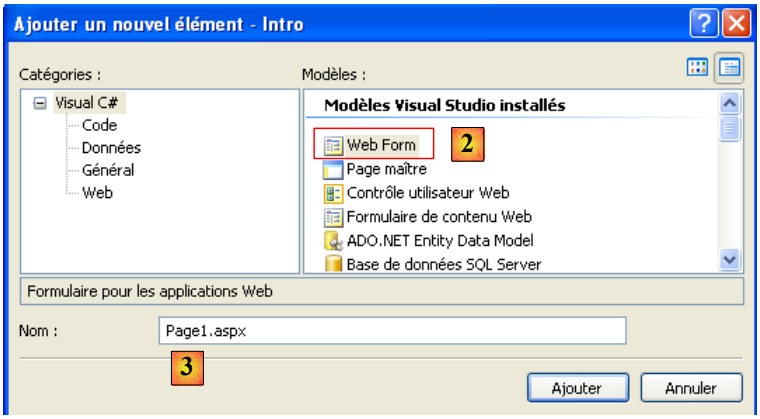
2.8 Forward d'une page vers une autre

Jusqu'à maintenant, les opérations GET et POST retournaient toujours la même page [Default.aspx]. Nous allons considérer le cas où une requête est traitée par deux pages ASPX successives, [Default.aspx] et [Page1.aspx] et où c'est cette dernière qui est retournée au client. Par ailleurs, nous verrons comment la page [Default.aspx] peut transmettre des informations à la page [Page1.aspx] via une mémoire qu'on appellera mémoire de la requête.

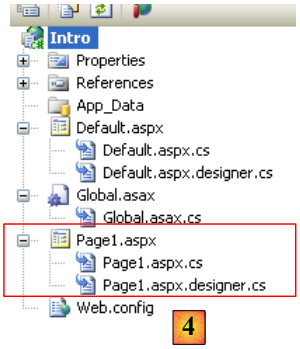


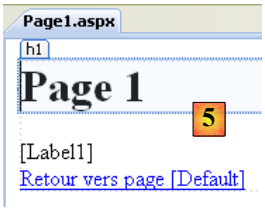
Nous construisons la page [Page1.aspx] :





- en [1], on ajoute un nouvel élément au projet
- en [2], on ajoute un élément [Web Form] nommé [Page1.aspx] [3]





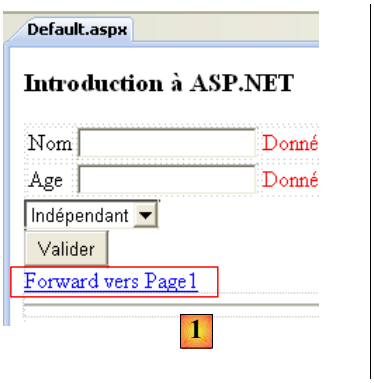
- en [4], la page ajoutée
- en [5], la page une fois construite

Le code source de [Page1.aspx] est le suivant :

```
1.  <%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Page1.aspx.cs" Inherits="Intro.Page1" %>
2.
3.  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4.  <html xmlns="http://www.w3.org/1999/xhtml">
5.  <head id="Head1" runat="server">
6.    <title>Page1</title>
7.  </head>
8.  <body>
9.    <form id="form1" runat="server">
10.   <div>
11.     <h1>
12.       Page 1</h1>
13.     <asp:Label ID="Label1" runat="server"></asp:Label>
14.     <br />
15.     <asp:HyperLink ID="HyperLink1" runat="server" NavigateUrl="~/Default.aspx">Retour
16.       vers page [Default]</asp:HyperLink>
17.   </div>
18.   </form>
19. </body>
20. </html>
```

- ligne 13 : un label qui servira à afficher une information transmise par la page [Default.aspx]
- ligne 15 : un lien Html vers la page [Default.aspx]. Lorsque l'utilisateur clique sur ce lien, le navigateur demande la page [Default.aspx] avec une opération GET. La page [Default.aspx] est alors chargée comme si l'utilisateur avait tapé directement son Url dans son navigateur.

La page [Default.aspx] s'enrichit d'un nouveau composant de type *LinkButton* :



Le code source de ce nouveau composant est le suivant :

```
<asp:LinkButton ID="LinkButtonToPage1" runat="server" CausesValidation="False"
  EnableViewState="False" onclick="LinkButtonToPage1_Click">Forward vers Page1</asp:LinkButton>
```

- CausesValidation="False" : le clic sur le lien va provoquer un POST vers [Defaul.aspx]. Le composant [LinkButton] se comporte comme le composant [Button]. Ici, on ne veut pas que le clic sur le lien déclenche l'exécution des validateurs.
- EnableViewState="False" : il n'y a pas lieu de conserver l'état du lien au fil des requêtes. Il garde ses valeurs de conception.
- onclick="LinkButtonToPage1_Click" : nom de la méthode qui, dans [Defaul.aspx.cs], gère l'événement *Click* sur le composant *LinkButtonToPage1*.

Le code du gestionnaire *LinkButtonToPage1_Click* est le suivant :

```
1.    // vers Page1
2.    protected void LinkButtonToPage1_Click(object sender, EventArgs e)
3.    {
4.      // on met des infos dans le contexte
5.      Context.Items["msg1"] = "Message de Default.aspx pour Page1";
6.      // on passe la requête à Page1
7.      Server.Transfer("Page1.aspx",true);
8.  }
```

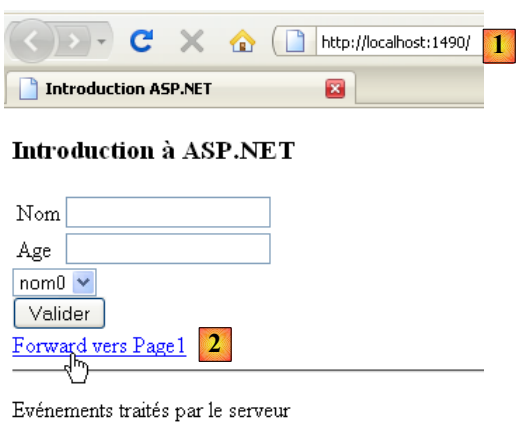
Ligne 7, la requête est passée à la page [Page1.aspx] au moyen de la méthode [Server.Transfer]. Le deuxième paramètre de la méthode qui est à *true* indique qu'il faut passer à [Page1.aspx] toute l'information qui a été envoyée à [Default.aspx] lors du POST. Cela permet par exemple à [Page1.aspx] d'avoir accès aux valeurs postées via une collection appelée *Request.Form*. La ligne 5 utilise ce qu'on appelle le contexte de la requête. On y a accès via la propriété *Context* de la classe *Page*. Ce contexte peut servir de mémoire entre les différentes pages qui traitent la même requête, ici [Default.aspx] et [Page1.aspx]. On utilise pour cela le dictionnaire **Items**.

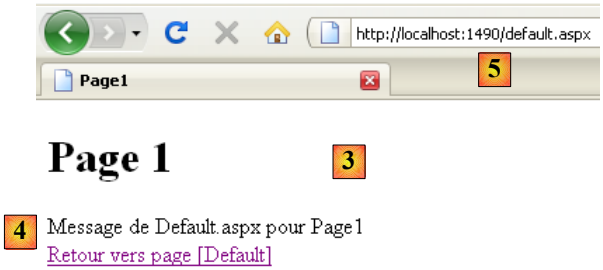
Lorsque [Page1.aspx] est chargée par l'opération *Server.Transfer("Page1.aspx",true)*, tout se passe comme si [Page1.aspx] avait été appelée par un GET d'un navigateur. Le gestionnaire *Page_Load* de [Page1.aspx] est exécuté normalement. Nous l'utiliserons pour afficher le message mis par [Default.aspx] dans le contexte de la requête :

```
1.  using System;
2.
3.  namespace Intro
4.  {
5.    public partial class Page1 : System.Web.UI.Page
6.    {
7.      protected void Page_Load(object sender, EventArgs e)
8.      {
9.        Label1.Text = Context.Items["msg1"] as string;
10.     }
11.   }
12. }
```

Ligne 9, le message mis par [Default.aspx] dans le contexte de la requête, est affiché dans *Label1*.

Voici un exemple d'exécution :

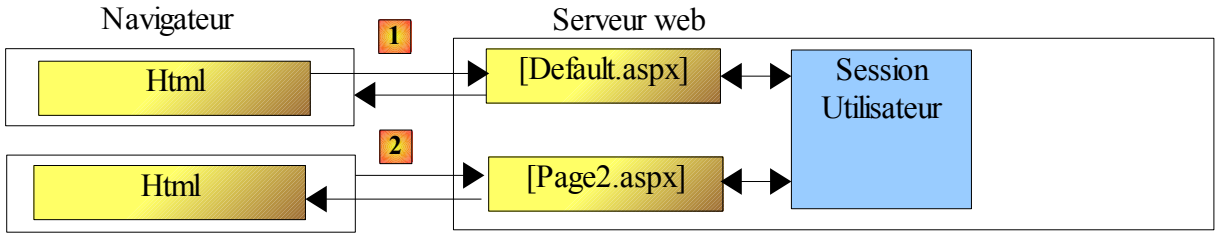




- dans la page [Default.aspx] [1], on clique sur le lien [2] qui nous mène vers la page *Page1*
- en [3], la page *Page1* est affichée
- en [4], le message créé dans [Default.aspx] et affiché par [Page1.aspx]
- en [5], l'Url affichée dans le navigateur est celle de la page [Default.aspx]

2.9 Redirection d'une page vers une autre

Nous présentons ici une autre technique fonctionnellement proche de la précédente : lorsque l'utilisateur demande la page [Default.aspx] par un POST, il reçoit en réponse une autre page [Page2.aspx]. Dans la méthode précédente, la requête de l'utilisateur était traitée successivement par deux pages : [Default.aspx] et [Page1.aspx]. Dans la méthode de la redirection de page que nous présentons maintenant, il y a deux requêtes distinctes du navigateur :



- en [1], le navigateur fait une requête POST à la page [Default.aspx]. Celle-ci traite la requête et envoie une réponse dite de redirection au navigateur. Cette réponse est un simple flux HTTP (des lignes de texte) demandant au navigateur de se rediriger vers une autre Url [Page2.aspx]. [Default.aspx] n'envoie pas de flux Html dans cette première réponse.
- en [2], le navigateur fait une requête GET à la page [Page2.aspx]. Celle-ci est alors envoyée en réponse au navigateur.
- si la page [Default.aspx] souhaite transmettre des informations à la page [Page2.aspx], elle peut le faire via la session de l'utilisateur. Contrairement à la méthode précédente, le contexte de la requête n'est pas utilisable ici, car il y a ici deux requêtes distinctes donc deux contextes distincts. Il faut alors utiliser la session de l'utilisateur pour faire communiquer les pages ensemble.

Comme il a été fait pour [Page1.aspx], nous ajoutons au projet la page [Page2.aspx] :

- en [1], [Page2.aspx] a été ajoutée au projet
- en [2], l'aspect visuel de [Page2.aspx]
- en [3], nous ajoutons à la page [Default.aspx] un composant *LinkButton* [4] qui va rediriger l'utilisateur vers [Page2.aspx].

Le code source de [Page2.aspx] est analogue à celui de [Page1.aspx] :

```
1.  <%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Page2.aspx.cs" Inherits="Intro.Page2" %>
2.
3.  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4.  <html xmlns="http://www.w3.org/1999/xhtml">
5.  <head id="Head1" runat="server">
6.    <title>Page2</title>
7.  </head>
8.  <body>
9.    <form id="form1" runat="server">
10.   <div>
11.     <h1>
12.       Page 2</h1>
13.     <asp:Label ID="Label1" runat="server"></asp:Label>
14.     <br />
15.     <asp:HyperLink ID="HyperLink1" runat="server" NavigateUrl="~/Default.aspx">Retour
16.       vers page [Default]</asp:HyperLink>
17.   </div>
18.   </form>
19. </body>
20. </html>
```

Dans [Default.aspx], l'ajout du composant *LinkButton* a généré le code source suivant :

```
<asp:LinkButton ID="LinkButtonToPage2" runat="server"
    onclick="LinkButtonToPage2_Click">Redirection vers Page2</asp:LinkButton>
```

C'est le gestionnaire [LinkButtonToPage2_Click] qui assure la redirection vers [Page2.aspx]. Son code dans [Defaul.aspx.cs] est le suivant :

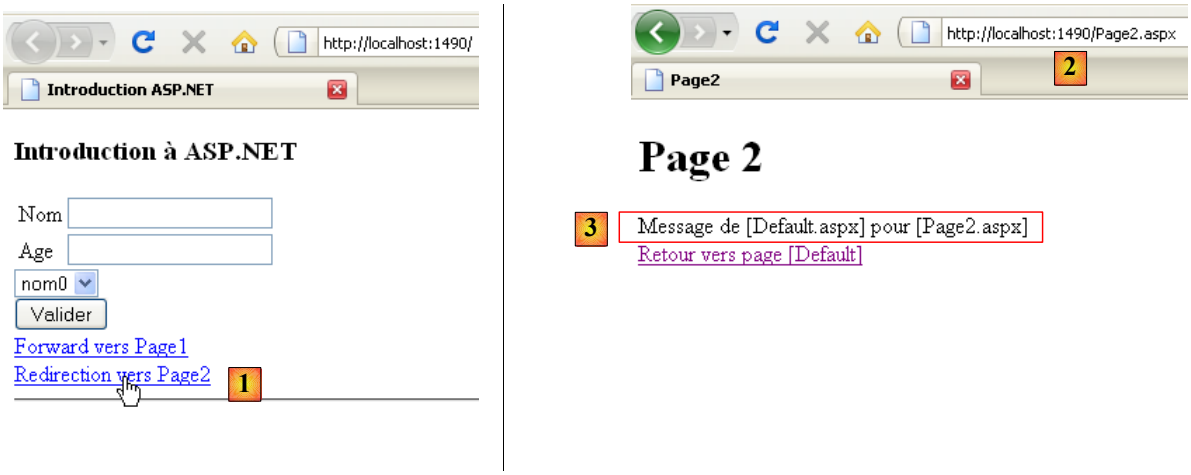
```
1.      protected void LinkButtonToPage2_Click(object sender, EventArgs e)
2.      {
3.        // on met un msg dans la session
4.        Session["msg2"] = "Message de [Default.aspx] pour [Page2.aspx]";
5.        // on redirige le client vers [Page2.aspx]
6.        Response.Redirect("Page2.aspx");
7.  }
```

- ligne 4 : on met un message dans la session de l'utilisateur
- ligne 5 : l'objet *Response* est une propriété de toute page ASPX. Elle représente la réponse faite au client. Elle possède une méthode *Redirect* qui fait que la réponse faite au client va être un ordre HTTP de redirection.

Lorsque le navigateur va recevoir l'ordre de redirection vers [Page2.aspx], il va faire un GET sur cette page. Dans celle-ci, la méthode [Page_Load] va s'exécuter. On va l'utiliser pour récupérer le message mis par [Default.aspx] dans la session et afficher celui-ci. Le code [Page2.aspx.cs] est le suivant :

```
1. using System;
2.
3. namespace Intro
4. {
5.   public partial class Page2 : System.Web.UI.Page
6.   {
7.     protected void Page_Load(object sender, EventArgs e)
8.     {
9.       // on affiche le msg mis dans la session par [Default.aspx]
10.      Label1.Text = Session["msg2"] as string;
11.    }
12.  }
13. }
```

A l'exécution, on obtient les résultats suivants :



- en [1], on clique sur le lien de redirection de [Default.aspx]. Un POST est fait vers la page [Default.aspx]
- en [2], le navigateur a été redirigé vers [Page2.aspx]. Cela se voit à l'Url affichée par le navigateur. Dans la méthode précédente, cette Url était celle de [Default.aspx] car l'unique requête faite par le navigateur l'était vers cette Url. Ici il y a un premier POST vers [Default.aspx], puis à l'insu de l'utilisateur un second GET vers [Page2.aspx].
- en [3], on voit que [Page2.aspx] a correctement récupéré le message mis par [Default.aspx] dans la session.

2.10 Conclusion

Nous avons introduit, à l'aide de quelques exemples, les concepts d'ASP.NET qui nous seront utiles dans la suite du document. Cette introduction ne permet pas de comprendre les subtilités des échanges client / serveur d'une application web. Pour cela, on pourra lire :

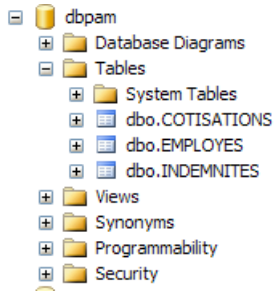
- **Programmation ASP.NET** [http://tahe.developpez.com/dotnet/aspnet/vol1] et [http://tahe.developpez.com/dotnet/aspnet/vol2]

3 L'étude de cas

Nous souhaitons écrire une application .NET permettant à un utilisateur de faire des simulations de calcul de la paie des assistantes maternelles de l'association " Maison de la petite enfance " d'une commune. Nous nous intéresserons autant à l'organisation du code DotNet de l'application qu'au code lui-même.

3.1 La base de données

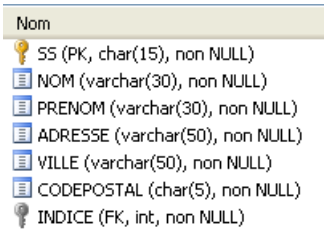
Les données statiques utiles pour construire la fiche de paie sont placées dans une base de données SQL Server Express nommée **dbpam** (**pam**=Paie Assistante Maternelle). Cette base a un administrateur appelé **sa** ayant le mot de passe **msde**.



La base a trois tables, **EMPLOYES, COTISATIONS** et **INDEMNITES,** dont la structure est la suivante :

Table **EMPLOYES** : rassemble des informations sur les différentes assistantes maternelles

Structure :



SS	numéro de sécurité sociale de l'employé - clé primaire
NOM	nom de l'employé
PRENOM	son prénom
ADRESSE	son adresse
VILLE	sa ville
CODEPOSTAL	son code postal
INDICE	son indice de traitement - clé étrangère sur le champ [INDICE] de la table [INDEMNITES]

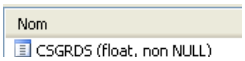
Son contenu pourrait être le suivant :

SS	NOM	PRENOM	ADRESSE	VILLE	CODEPOSTAL	INDICE
254104940426058	Jouveinal	Marie	5 rue des Oiseaux	St Corentin	49203	2
260124402111742	Laverti	Justine	la Brûlerie	St Marcel	49014	1

Table **COTISATIONS** : rassemble les taux des cotisations sociales prélevées sur le salaire

Structure :

S

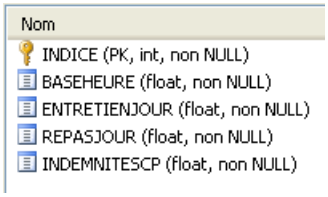


CSGRDS	pourcentage : contribution sociale généralisée + contribution au remboursement de la dette sociale

CSGRDS	CSGD	SECU	RETRAITE
3,49	6,15	9,39	7,88

Les taux des cotisations sociales sont independants du salarie. La table precedente n'a qu'une ligne.

Table **INDEMNITES** : rassemble les différentes indemnités dépendant de l'indice de l'employé



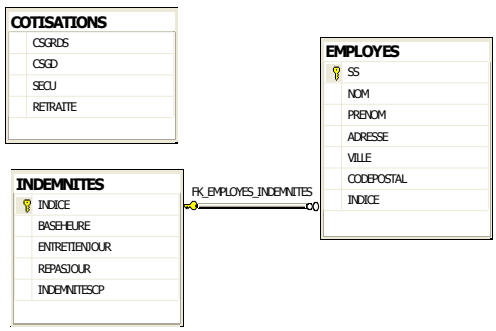
INDICE	indice de traitement - clé primaire
BASEHEURE	prix net en euro d'une heure de garde
ENTRETIENJOUR	indemnité d'entretien en euro par jour de garde
REPASJOUR	indemnité de repas en euro par jour de garde
INDEMNITESCP	indemnité de congés payés. C'est un pourcentage à appliquer au salaire de base.

Son contenu pourrait être le suivant :

INDICE	BASEHEURE	ENTRETIENJOUR	REPASJOUR	INDEMNITESCP
1	1,93	2	3	12
2	2,1	2,1	3,1	15

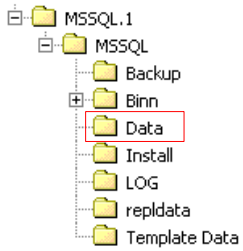
On notera que les indemnités peuvent varier d'une assistante maternelle à une autre. Elles sont en effet associées à une assistante maternelle précise via l'indice de traitement de celle-ci. Ainsi *Mme Marie Jouveinal* qui a un indice de traitement de 2 (table EMPLOYES) a un salaire horaire de 2,1 euros (table INDEMNITES).

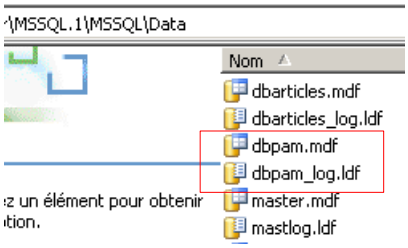
Les relations entre les trois tables sont les suivantes :



Il y a une relation de clé étrangère entre la colonne EMPLOYES(INDICE) et la colonne INDEMNITES(INDICE).

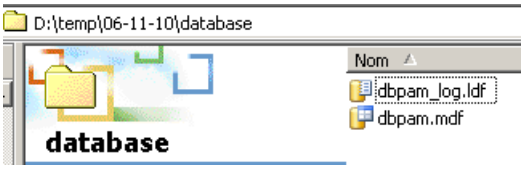
La base de données [dbpam] ainsi créée donne naissance à deux fichiers dans le dossier de SQL Server Express :



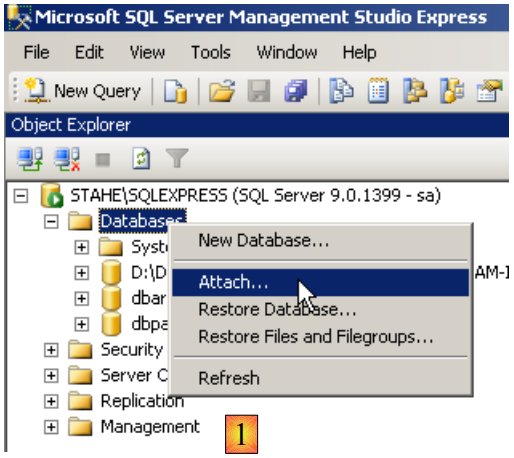


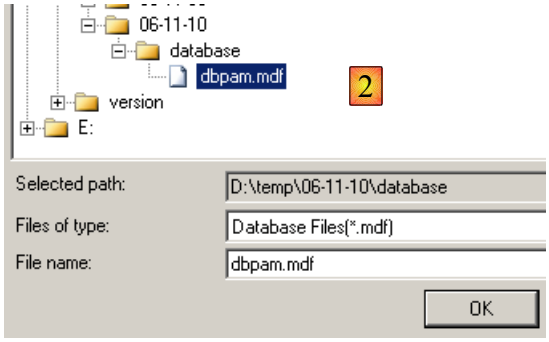
Les fichiers [dbpam.mdf, dbpam_log.ldf] peuvent être transportés sur une autre machine et être réattachés au SGBD SQL Server Express de cette machine. Voici comment procéder :

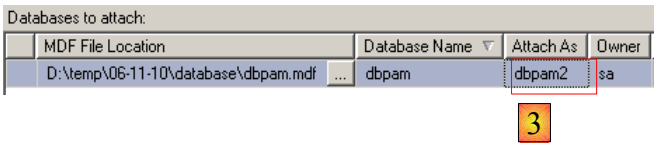
- les fichiers de la BD [dbpam] sont dupliqués dans un dossier

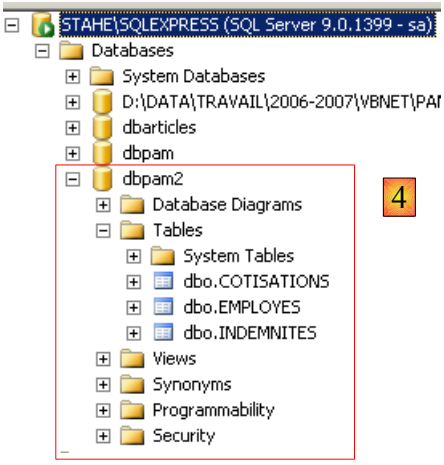


- on lance SQL Server Express
- avec le client SQL Server Management Studio Express, on attache le fichier [dbpam.mdf] au SGBD :









1. clic droit sur [Databases] / Attach
2. choix du fichier [dbpam.mdf] via un bouton [Add] non représenté
3. le fichier attaché va donner naissance à une BD qui ne doit pas déjà exister. Ici, dans le champ [Attach As] on a donné le nom [dbpam2] à la nouvelle BD.
4. on voit la nouvelle BD et ses tables

Cette technique de l'attachement d'une BD est utile pour transporter une BD d'un poste à un autre et nous l'utiliserons ici.

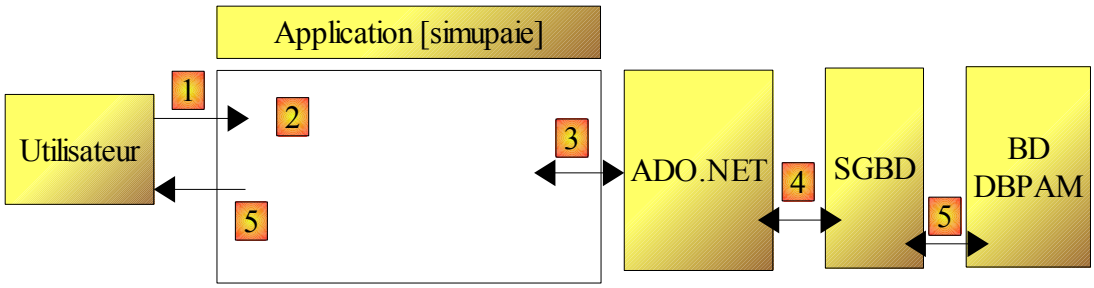
3.2 Mode de calcul du salaire d'une assistante maternelle

Nous présentons maintenant le mode de calcul du salaire mensuel d'une assistante maternelle. Nous prenons pour exemple, le salaire de Mme Marie Jouveinal qui a travaillé 150 h sur 20 jours pendant le mois à payer.

Les éléments suivants sont pris en compte :	[TOTALHEURES]: total des heures travaillées dans le mois [TOTALJOURS]: total des jours travaillés dans le mois	[TOTALHEURES]=150 [TOTALJOURS]= 20
Le salaire de base de l'assistante maternelle est donné par la formule suivante :	[SALAIREBASE]=([TOTALHEURES]*[BASEHEURE])*(1+[INDEMNITESCP]/100)	[SALAIREBASE]=(150*[2.1])*(1+0.15)= 362,25
Un certain nombre de cotisations sociales doivent être prélevées sur ce salaire de base :	Contribution sociale généralisée et contribution au remboursement de la dette sociale : [SALAIREBASE]*[CSGRDS/100] Contribution sociale généralisée déductible : [SALAIREBASE]*[CSGD/100] Sécurité sociale, veuvage, vieillesse : [SALAIREBASE]*[SECU/100] Retraite Complémentaire + AGPF + Assurance Chômage : [SALAIREBASE]*[RETRAITE/100]	CSGRDS : 12,64 CSGD : 22,28 Sécurité sociale : 34,02 Retraite : 28,55
Total des cotisations sociales :	[COTISATIONSSOCIALES]=[SALAIREBASE]*(CSGRDS+CSGD+SECU+RETRAITE)/100	[COTISATIONSSOCIALES]=97,48
Par ailleurs, l'assistante maternelle a droit, chaque jour travaillé, à une indemnité d'entretien ainsi qu'à une indemnité de repas. A ce titre elle reçoit les indemnités suivantes :	[INDEMNITÉS]=[TOTALJOURS]*(ENTRETIENJOUR+REPASJOUR)	[INDEMNITES]=104
Au final, le salaire net à payer à l'assistante maternelle est le suivant :	[SALAIREBASE]-[COTISATIONSSOCIALES]+[INDEMNITÉS]	[salaire NET]=368,77

3.3 Rappels ADO.NET

L'application de calcul de paie a besoin des informations de la base de données [dbpam]. Son architecture sera la suivante :



- en [1], l'utilisateur fait une demande
- en [2], l'application de paie la traite.

- elle peut alors avoir besoin des données de la base de données. Elle fait alors une requête au fournisseur (provider) ADO.NET du SGBD utilisé [4].
- celui-ci exploite la base de données [5] et rend ses résultats au fournisseur ADO.NET qui lui-même les remonte à l'application
- celle ci exploite ces résultats et élabore une réponse [5] pour l'utilisateur

Nous rappelons maintenant les principales interfaces présentées par un fournisseur ADO.NET à ses clients [3].

En mode **connecté**, l'application :
1. ouvre une connexion avec la source de données
2. travaille avec la source de données en lecture/écriture
3. ferme la connexion

Trois interfaces ADO.NET sont principalement concernées par ces opérations :

- **IDbConnection** qui encapsule les propriétés et méthodes de la connexion.
- **IDbCommand** qui encapsule les propriétés et méthodes de la commande SQL exécutée.
- **IDataReader** qui encapsule les propriétés et méthodes du résultat d'un ordre SQL Select.

L'interface IDbConnection

Sert à gérer la connexion avec la base de données. Parmi les méthodes **M** et propriétés **P** de cette interface on trouve les suivantes :

Nom	Type	Rôle
`ConnectionString`	P	**chaîne de connexion** à la base. Elle précise tous les paramètres nécessaires à l'établissement de la connexion avec une base précise.
`Open`	M	ouvre la connexion avec la base définie par *ConnectionString*
`Close`	M	ferme la connexion
`BeginTransaction`	M	démarre une transaction.
`State`	P	état de la connexion : *ConnectionState.Closed*, *ConnectionState.Open*, *ConnectionState.Connecting*, *ConnectionState.Executing*, *ConnectionState.Fetching*, *ConnectionState.Broken*

Si *Connection* est une classe implémentant l'interface *IDbConnection*, l'ouverture de la connexion peut se faire comme suit :

```
1.  IDbConnection connexion=new Connection();
2.  connexion.ConnectionString=...;
3.  connexion.Open();
```

L'interface IDbCommand

Sert à exécuter un ordre SQL ou une procédure stockée. Parmi les méthodes **M** et propriétés **P** de cette interface on trouve les suivantes :

Nom	Type	Rôle
`CommandType`	P	indique ce qu'il faut exécuter - prend ses valeurs dans une énumération : - *CommandType.Text* : exécute l'ordre SQL défini dans la propriété *CommandText*. C'est la valeur par défaut. - *CommandType.StoredProcedure* : exécute une procédure stockée dans la base
`CommandText`	P	- le texte de l'ordre SQL à exécuter si *CommandType= CommandType.Text* - le nom de la procédure stockée à exécuter si *CommandType= CommandType.StoredProcedure*
`Connection`	P	la connexion *IDbConnection* à utiliser pour exécuter l'ordre SQL
`Transaction`	P	la transaction *IDbTransaction* dans laquelle exécuter l'ordre SQL
`Parameters`	P	la liste des paramètres d'un ordre SQL paramétré. L'ordre *update articles set prix=prix*1.1 where id=@id* a le paramètre @id.
`ExecuteReader`	M	pour exécuter un ordre SQL *Select*. On obtient un objet *IDataReader* représentant le résultat du *Select*.
`ExecuteNonQuery`	M	pour exécuter un ordre SQL *Update, Insert, Delete*. On obtient le nombre de lignes affectées par l'opération (mises à jour, insérées, détruites).
`ExecuteScalar`	M	pour exécuter un ordre SQL *Select* ne rendant qu'un unique résultat comme dans : *select count(*) from articles*.

Nom	Type	Rôle
CreateParameter	M	pour créer les paramètres *IDbParameter* d'un ordre SQL paramétré.
Prepare	M	permet d'optimiser l'exécution d'une requête paramétrée lorsqu'elle est exécutée de multiples fois avec des paramètres différents.

Si *Command* est une classe implémentant l'interface *IDbCommand*, l'exécution d'un ordre SQL sans transaction aura la forme suivante :

```
1.  // ouverture connexion
2.  IDbConnection connexion=...
3.  connexion.Open();
4.  // préparation commande
5.  IDbCommand commande=new Command();
6.  commande.Connection=connexion;
7.  // exécution ordre select
8.  commande.CommandText="select ...";
9.  IDbDataReader reader=commande.ExecuteReader();
10. ...
11. // exécution ordre update, insert, delete
12. commande.CommandText="insert ...";
13. int nbLignesInsérées=commande.ExecuteNonQuery();
14. ...
15. // fermeture connexion
16. connexion.Close();
```

L'interface IDataReader

Sert à encapsuler les résultats d'un ordre SQL *Select*. Un objet *IDataReader* représente une table avec des lignes et des colonnes, qu'on exploite séquentiellement : d'abord la 1ère ligne, puis la seconde, ... Parmi les méthodes **M** et propriétés **P** de cette interface on trouve les suivantes :

Nom	Type	Rôle
FieldCount	P	le nombre de colonnes de la table *IDataReader*
GetName	M	*GetName(i)* rend le nom de la colonne n° i de la table *IDataReader*.
Item	P	*Item[i]* représente la colonne n° i de la ligne courante de la table *IDataReader*.
Read	M	passe à la ligne suivante de la table *IDataReader*. Rend le booléen *True* si la lecture a pu se faire, *False* sinon.
Close	M	ferme la table *IDataReader*.
GetBoolean	M	*GetBoolean*(i) : rend la valeur booléenne de la colonne n° i de la ligne courante de la table *IDataReader*. Les autres méthodes analogues sont les suivantes : *GetDateTime*, *GetDecimal*, *GetDouble*, *GetFloat*, *GetInt16*, *GetInt32*, *GetInt64*, *GetString*.
Getvalue	M	*Getvalue*(i) : rend la valeur de la colonne n° i de la ligne courante de la table *IDataReader* en tant que type *object*.
IsDBNull	M	*IsDBNull(i)* rend *True* si la colonne n° i de la ligne courante de la table *IDataReader* n'a pas de valeur ce qui est symbolisé par la valeur SQL NULL.

L'exploitation d'un objet *IDataReader* ressemble souvent à ce qui suit :

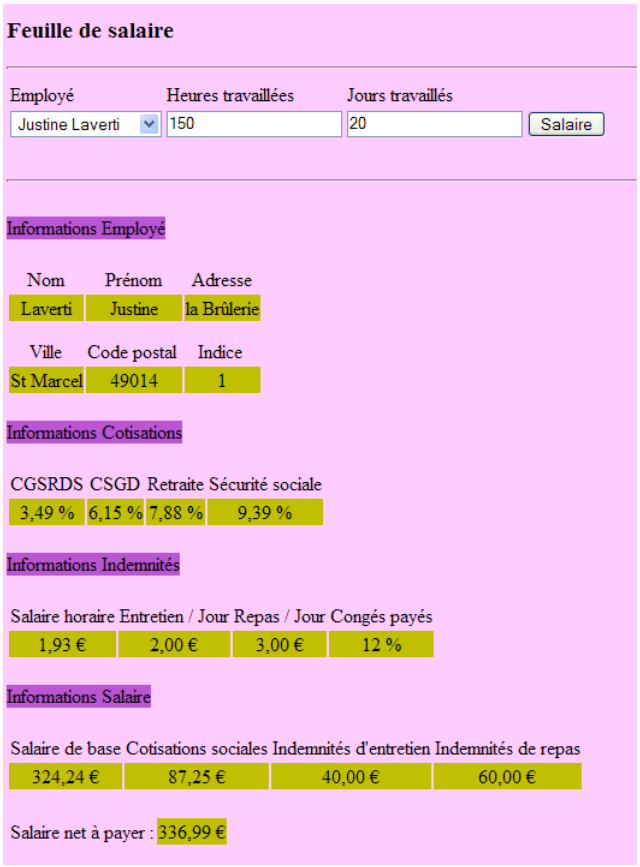
```
1.  // ouverture connexion
2.  IDbConnection connexion=...
3.  connexion.Open();
4.  // préparation commande
5.  IDbCommand commande=new Command();
6.  commande.Connection=connexion;
7.  // exécution ordre select
8.  commande.CommandText="select ...";
9.  IDataReader reader=commande.ExecuteReader();
10. // exploitation résultats
11. while(reader.Read()){
12.  // exploiter ligne courante
13.     ...
14. }
15. // fermeture reader
16. reader.Close();
17. // fermeture connexion
18. connexion.Close();
```

4 L'application [SimuPaie] – version 1 – ASP.NET

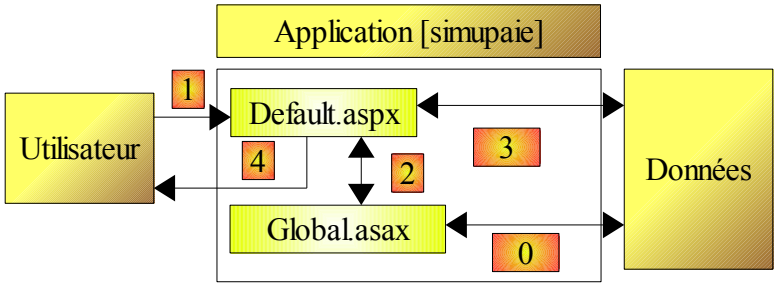
4.1 Introduction

Nous souhaitons écrire une application .NET permettant à un utilisateur de faire des simulations de calcul de la paie des assistantes maternelles de l'association " Maison de la petite enfance " d'une commune.

Le formulaire ASP.NET de calcul du salaire aura l'allure suivante :



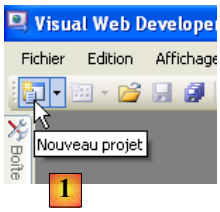
L'application ASP.NET aura l'architecture suivante :

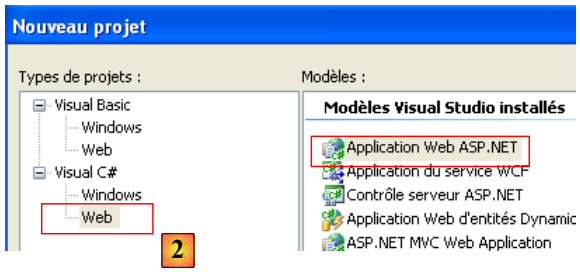


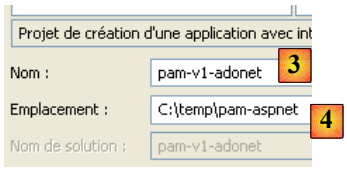
- lors de la première requête faite à l'application, un objet de type [Global] dérivé du type [System.Web.HttpApplication] est instancié. C'est lui qui va exploiter le fichier de configuration [web.config] de l'application web et mettre en cache certaines données de la base de données (opération 0 ci-dessus).

- lors de la première requête (opération 1) faite à la page [Default.aspx] qui est l'unique page de l'application, l'événement [Load] est traité. On y traite le remplissage du combo des employés. Les données nécessaires au combo sont demandées à l'objet [Global] qui les a mises en cache. C'est l'opération 2 ci-dessus. L'opération 4 envoie la page ainsi initialisée à l'utilisateur.
- lors de la demande du calcul du salaire (opération 1) faite à la page [Default.aspx], l'événement [Load] est de nouveau traité. Rien n'est fait car il s'agit d'une demande de type POST, et le gestionnaire [Pam_Load] a été écrit pour ne rien faire dans ce cas (utilisation du booléen *IsPostBack*). L'événement [Load] traité, c'est l'événement [Click] sur le bouton [Salaire] qui l'est ensuite. Celui-ci a besoin de données qui n'ont pas été mises en cache dans [Global]. Aussi le gestionnaire de l'événement les demande-t-il à la base de données. C'est l'opération 3 ci-dessus. Le calcul du salaire est ensuite fait et l'opération 4 envoie les résultats à l'utilisateur.

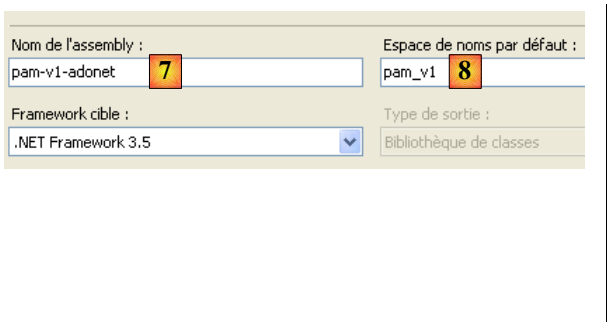
4.2 Le projet Visual Web Developer 2008







- en [1], on crée un nouveau projet
- en [2], de type [Web / Application Web ASP.NET]
- en [3], on donne un nom au projet
- en [4], on précise son nom et en [5] son emplacement. Un dossier [c:\temp\pam-aspnet\pam-v1-adonet] va être créé pour le projet.



- en [5], le projet Visual Web Developer
- en [6], les propriétés du projet (clic droit sur projet / Propriétés / Application).
- en [7], le nom de l'assembly qui sera produit par la génération du projet
- en [8], l'espace de noms par défaut que nous souhaitons avoir pour les classes du projet. La classe [_Default] définie dans les fichiers [Default.aspx.cs] et [Default.aspx.designer.cs] a elle été créée dans l'espace de noms [pam_v1_adonet] dérivé du nom du projet. On peut changer cet espace de noms directement dans le code de ces deux fichiers :

[Default.aspx.cs]

```
using System;
....

namespace pam_v1
{
  public partial class _Default : System.Web.UI.Page
  {
```

[Default.aspx.designer.cs]

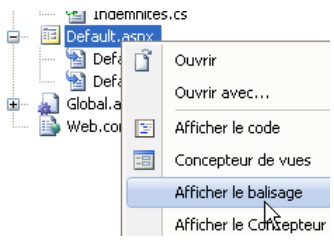
```
//------------------------------------------------------------------------------
// <auto-generated>
//     Ce code a été généré par un outil.
//     Version du runtime :2.0.50727.3603
//
//     Les modifications apportées à ce fichier peuvent provoquer un comportement incorrect et seront
perdues si
//     le code est régénéré.
// </auto-generated>
//------------------------------------------------------------------------------

namespace pam_v1 {


    public partial class _Default {

.........
```

Le balisage du fichier [Default.aspx] doit être également modifié :

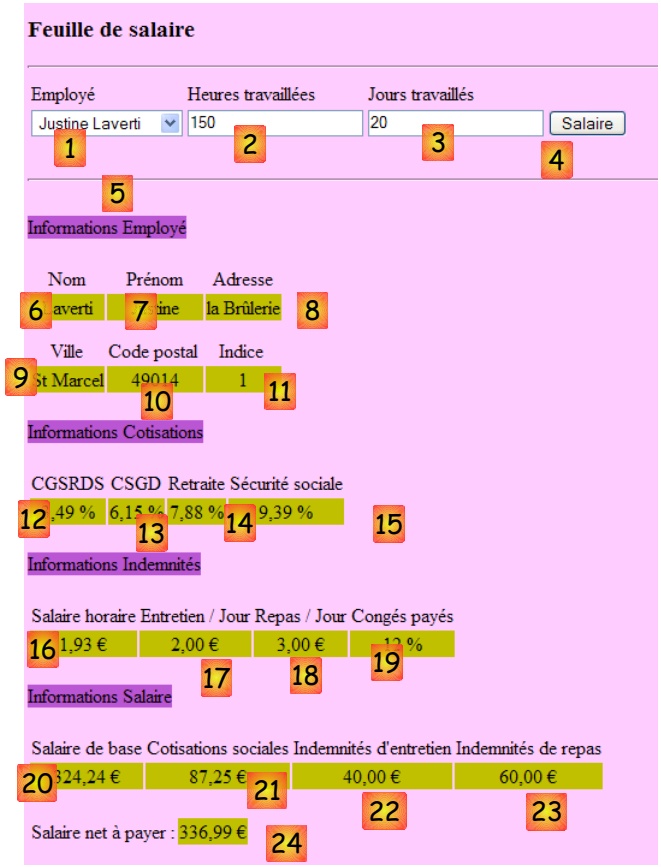



```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs" Inherits="pam_v1._Default" %>
...
```

L'attribut *Inherits* ci-dessus désigne la classe définie dans les fichiers [Default.aspx.cs] et [Default.aspx.designer.cs]. On y utilise l'espace de noms *pam_v1*.

4.2.1 Le formulaire [Default.aspx]

L'aspect visuel du formulaire [Default.aspx] est le suivant :



Les composants sont les suivants :

N°	Type	Nom	Rôle
1	DropDownList	ComboBoxEmployes	Contient la liste des noms des employés
2	TextBox	TextBoxHeures	Nombre d'heures travaillées – nombre réel
3	TextBox	TextBoxJours	Nombre de jours travaillés – nombre entier
4	Button	ButtonSalaire	Demande le calcul du salaire
5	TextBox	TextBoxErreur	Message d'information à destination de l'utilisateur *ReadOnly=true*, *TextMode=MultiLine*
6	Label	LabelNom	Nom de l'employé sélectionné en (1)
7	Label	LabelPrénom	Prénom de l'employé sélectionné en (1)
8	Label	LabelAdresse	Adresse
9	Label	LabelVille	Ville
10	Label	LabelCP	Code postal
11	Label	LabelIndice	Indice
12	Label	LabelCSGRDS	Taux de cotisation CSGRDS
13	Label	LabelCSGD	Taux de cotisation CSGD
14	Label	LabelRetraite	Taux de cotisation Retraite
15	Label	LabelSS	Taux de cotisation Sécurité Sociale
16	Label	LabelSH	Salaire horaire de base pour l'indice indiqué en (11)
17	Label	LabelEJ	Indemnité journalière d'entretien pour l'indice indiqué en (11)
18	Label	LabelRJ	Indemnité journalière de repas pour l'indice indiqué en (11)
19	Label	LabelCongés	Taux d'indemnités de congés payés à appliquer au salaire de base

N°	Type	Nom	Rôle
20	Label	LabelSB	Montant du salaire de base
21	Label	LabelCS	Montant des cotisations sociales à verser
22	Label	LabelIE	Montant des indemnités d'entretien de l'enfant gardé
23	Label	LabelIR	Montant des indemnités de repas de l'enfant gardé
24	Label	LabelSN	Salaire net à payer à l'employé(e)

Le formulaire contient en outre deux conteneurs de type [Panel] :

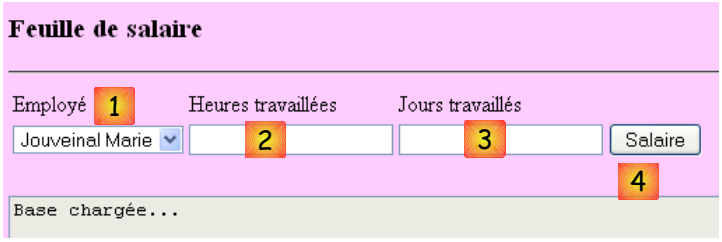
PanelErreurs	contient le composant (5) *TextBoxErreur*
PanelSalaire	contient les composants (6) à (24)

Un composant de type [Panel] peut être rendu visible ou non par programmation grâce à sa propriété booléenne *[Panel].Visible*.

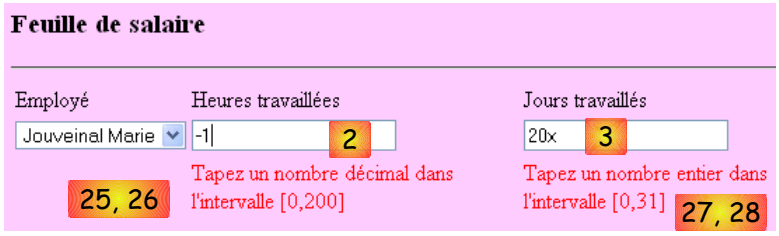
4.2.2 La vérification des saisies

Pour calculer un salaire, l'utilisateur :

- sélectionne un employé en [1]
- saisit le nombre d'heures travaillées en [2]. Ce nombre peut être décimal, comme 2,5 pour 2 h 30 mn.
- saisit le nombre de jours travaillés en [3]. Ce nombre est entier.
- demande le salaire avec le bouton [4]

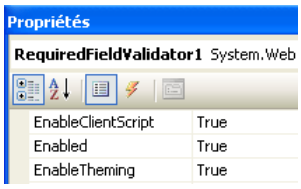


Lorsque l'utilisateur entre des données erronées dans [2] et [3], celles-ci sont vérifiées dès que l'utilisateur change de champ de saisie. Ainsi, la copie d'écran ci-dessous a été obtenue avant même que l'utilisateur ne clique sur le bouton [Salaire] :



Il faut deux conditions pour avoir le comportement précédent :

- les composants de validation doivent avoir leur propriété *EnableClientScript* à vrai :



- le navigateur affichant la page doit être capable d'exécuter le code Javascript embarqué dans une page HTML.

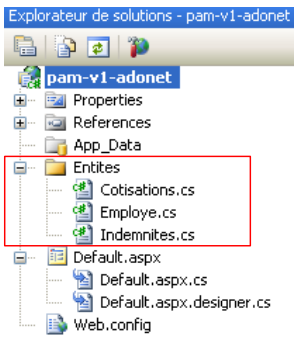
Si le navigateur client ne vérifie pas lui-même la validité des données, celles-ci ne seront vérifiées que lorsque le navigateur postera les saisies du formulaire au serveur. C'est alors le code situé sur ce dernier et qui traite la demande du navigateur qui vérifiera la validité des données. On rappelle que cette vérification doit être toujours faite même si la page affichée dans le navigateur client embarque du code javascript faisant cette même vérification. En effet, le serveur ne peut être assuré que la demande POST qui lui est faite vient réellement de cette page et que donc la vérification des données a été faite.

La liste des validateurs est la suivante :

N°	Type	Nom	Rôle
25	RequiredFieldValidator	RequiredFieldValidatorHeures	vérifie que le champ [2] [TextBoxHeures] n'est pas vide
26	RangeValidator	RangeValidatorHeures	vérifie que le champ [2] [TextBoxHeures] est un nombre réel dans l'intervalle [0, 200]
27	RequiredFieldValidator	RequiredFieldValidatorJours	vérifie que le champ [3] [TextBoxJours] n'est pas vide
28	RangeValidator	RangeValidatorJours	vérifie que le champ [3] [TextBoxJours] est un nombre entier dans l'intervalle [0,31]

Travail à faire : construire la page [Default.aspx]. On placera d'abord les deux conteneurs [PanelErreurs] et [PanelSalaire] pour pouvoir ensuite y placer les composants qu'ils doivent contenir.

4.2.3 Les entités de l'application



Une fois lues, les lignes des tables [cotisations], [employes], [indemnites] seront mises dans des objets de type [Cotisations], [Employe] et [Indemnites] définis comme suit :

```
1.  namespace Pam.Entites
2.  {
3.    public class Cotisations
4.    {
5.      // propriétés automatiques
6.      public double CsgRds { get; set; }
7.      public double Csgd { get; set; }
8.      public double Secu { get; set; }
9.      public double Retraite { get; set; }
10.
11.     // constructeurs
12.     public Cotisations()
13.     {
14.     }
15.
16.     public Cotisations(double csgRds, double csgd, double secu, double retraite)
17.     {
18.       CsgRds = csgRds;
19.       Csgd = csgd;
20.       Secu = secu;
```

```
21.       Retraite = retraite;
22.     }
23.
24.     // ToString
25.     public override string ToString()
26.     {
27.       return string.Format("[{0},{1},{2},{3}]", CsgRds, Csgd, Secu, Retraite);
28.     }
29.   }
30. }
```

```
1.  namespace Pam.Entites
2.  {
3.    public class Employe
4.    {
5.      // propriétés automatiques
6.      public string SS { get; set; }
7.      public string Nom { get; set; }
8.      public string Prenom { get; set; }
9.      public string Adresse { get; set; }
10.     private string Ville { get; set; }
11.     private string CodePostal { get; set; }
12.     private int Indice { get; set; }
13.
14.     // constructeurs
15.     public Employe()
16.     {
17.
18.     }
19.
20.     public Employe(string ss, string nom, string prenom, string adresse, string codePostal, string
    ville, int indice)
21.     {
22.       SS = ss;
23.       Nom = nom;
24.       Prenom = prenom;
25.       Adresse = adresse;
26.       CodePostal = codePostal;
27.       Ville = ville;
28.       Indice = indice;
29.     }
30.
31.     // ToString
32.     public override string ToString()
33.     {
34.       return string.Format("[{0},{1},{2},{3},{4},{5},{6}]", SS, Nom, Prenom, Adresse, Ville,
    CodePostal, Indice);
35.     }
36.   }
37. }
```

```
1.  namespace Pam.Entites
2.  {
3.    public class Indemnites
4.    {
5.
6.      // propriétés automatiques
7.      public int Indice { get; set; }
8.      public double BaseHeure { get; set; }
9.      public double EntretienJour { get; set; }
10.     public double RepasJour { get; set; }
11.     public double IndemnitesCP { get; set; }
12.
13.     // constructeurs
14.     public Indemnites()
15.     {
16.
17.     }
18.
19.     public Indemnites(int indice, double baseHeure, double entretienJour, double repasJour, double
    indemnitesCP)
20.     {
21.       Indice = indice;
22.       BaseHeure = baseHeure;
23.       EntretienJour = entretienJour;
```

```
24.       RepasJour = repasJour;
25.       IndemnitesCP = indemnitesCP;
26.     }
27.
28.     // identité
29.     public override string ToString()
30.     {
31.       return string.Format("[{0}, {1}, {2}, {3}, {4}]", Indice, BaseHeure, EntretienJour,
    RepasJour, IndemnitesCP);
32.     }
33.
34.   }
35. }
```

4.2.4 Configuration de l'application

Le fichier [Web.config] qui configure l'application sera le suivant :

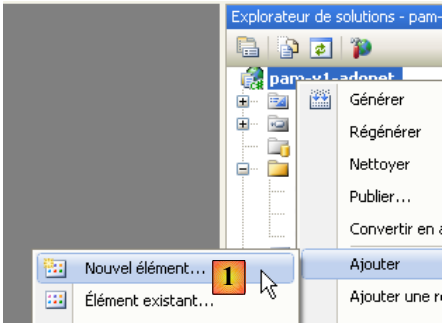
```
1.  <?xml version="1.0" encoding="utf-8"?>
2.
3.  <configuration>
4.      <configSections>
5.  ...
6.      </configSections>
7.
8.    <connectionStrings>
9.      <add name="dbpamSqlServer2005" connectionString="Data Source=.\SQLEXPRESS;AttachDbFilename=C:\
    data\...\dbpam.mdf;User Id=sa;Password=msde;Connect Timeout=30;"
    providerName="System.Data.SqlClient"/>
10.   </connectionStrings>
11.
12.   <appSettings>
13.     <add key="selectEmploye" value="select NOM,PRENOM,ADRESSE,VILLE,CODEPOSTAL,INDICE from
    EMPLOYES where SS=@SS"/>
14.     <add key="selectEmployes" value="select PRENOM, NOM, SS from EMPLOYES"/>
15.     <add key="selectCotisations" value="select CSGRDS,CSGD,SECU,RETRAITE from COTISATIONS"/>
16.     <add key="SelectIndemnites" value="select
    INDICE,BASEHEURE,ENTRETIENJOUR,REPASJOUR,INDEMNITESCP from INDEMNITES"/>
17.   </appSettings>
18.
19.   <system.web>
20.         <!--
21.             Définissez compilation debug="true" pour insérer des symboles
22.             de débogage dans la page compilée. Comme ceci
23.             affecte les performances, définissez cette valeur à true uniquement
24.             lors du développement.
25.         -->
26.         <compilation debug="false">
27. ...
28. </configuration>
```

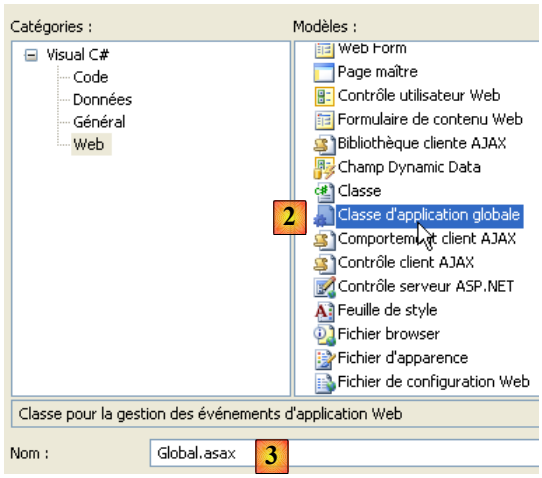
- ligne 9 : définit la chaîne de connexion à la base SQL Server précédente
- lignes 13-16 : définissent des requêtes SQL utilisées par l'application afin d'éviter de les mettre en dur dans le code.
- ligne 13 : la requête SQL est paramétrée. La notation du paramètre est propre à SQL Server.

Travail à faire : introduire ces paramètres dans le fichier [Web.config]. La ligne 9 sera adaptée au chemin réel de la base de données [dbpam.mdf].

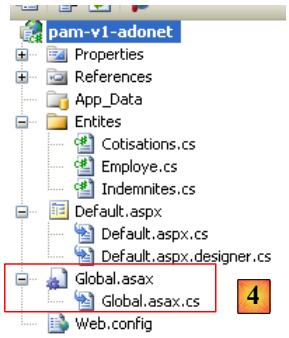
4.2.5 Initialisation de l'application

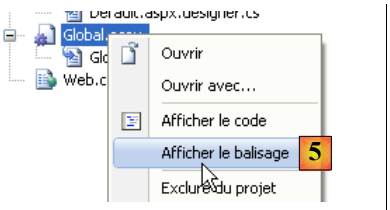
L'initialisation d'une application ASP.NET est faite par le fichier [Global.asax.cs]. Celui-ci est construit de la façon suivante :





- en [1], on ajoute un nouvel élément au projet
- en [2], on ajoute la classe d'application globale qui s'appelle par défaut [Global.asax] [3]





- en [4], le fichier [Global.asax] et la classe associée [Global.asax.cs]
- en [5], on fait afficher le balisage de [Global.asax]

```
<%@ Application Codebehind="Global.asax.cs" Inherits="pam_v1.Global" Language="C#" %>
```

La classe [pam_v1.Global] est définie dans le fichier [Global.asax.cs]. Pour notre problème, elle sera définie comme suit :

```
1.  using System;
2.  using System.Collections.Generic;
3.  using System.Configuration;
4.  using System.Data.SqlClient;
5.  using Pam.Entites;
6.
7.  namespace pam_v1
8.  {
9.    public class Global : System.Web.HttpApplication
10.   {
11.     // --- données statiques de l'application ---
12.     public static Employe[] Employes;
13.     public static Cotisations Cotisations;
14.     public static Dictionary<int, Indemnites> Indemnites = new Dictionary<int, Indemnites>();
15.     public static string Msg = string.Empty;
16.     public static bool Erreur = false;
17.     public static string ConnectionString = null;
18.
19.     // démarrage de l'application
20.     public void Application_Start(object sender, EventArgs e)
21.     {
```

```
22. ...
23.       try
24.       {
25.         // connexion
26.         ConnectionString = ...
27.         using (SqlConnection connexion = new SqlConnection(ConnectionString))
28.         {
29.           connexion.Open();
30.           // on récupère la liste des employés qu'on met dans le tableau statique [Employes]
31. ...
32.           // on récupère les taux de cotisation dans la variable statique [Cotisations]
33. ...
34.           // on récupère les indemnités dans le dictionnaire statique [Indemnites]
35. ...
36.           // on a réussi
37.           Msg = "Base chargée...";
38.         }
39.       }
40.       catch (Exception ex)
41.       {
42.         // on note l'erreur
43.         Msg = string.Format("L'erreur suivante s'est produite lors de l'accès à la base de données
    : {0}", ex);
44.         Erreur = true;
45.       }
46.     }
47.   }
48. }
```

- lignes 20: la méthode [Application_Start] est exécutée au démarrage de l'application web. Elle n'est exécutée qu'une fois.
- lignes 12-17 : champs publics et **statiques** de la classe. Un champ statique est partagé par toutes les instances de la classe. Ainsi, si plusieurs instances de la classe [Global] sont créées, elles partagent toutes le même champ statique [Employes], accessible via la référence [Global.Employes], c.a.d. *[NomDeClasse].ChampStatique*. [Global] est le nom d'une classe. C'est donc un type de donnée. Ce nom est arbitraire. La classe dérive toujours de [System.Web.HttpApplication].

  Revenons à notre classe [Global]. On peut se demander s'il est nécessaire de déclarer **statiques** ses champs. En fait, il semble que dans certains cas, on puisse avoir plusieurs exemplaires de la classe [Global], ce qui justifie le fait de rendre statiques les champs qui ont à être partagés par toutes ces instances.

  Il existe une autre façon de partager des données de portée " application " entre les différentes pages d'une application web. Ainsi le tableau *Employes* des employés pourrait être mémorisé dans la procédure *Application_Start* de la façon suivante :

```
Application.Add("employes",Employes);
```

  [Application] est par défaut dans toute application ASP.NET une référence sur une instance de la classe définie par [Global.asax.cs]. *Application* est un conteneur pouvant mémoriser des objets de tout type. Le tableau *Employes* pourrait ensuite être récupéré dans le code d'une page quelconque de l'application web de la façon suivante :

```
Employe[] employes=Application.Get["employes"] as Employe[];
```

  Parce que le conteneur *Application* mémorise des objets de tout type, on récupère un type *Object* qu'il faut ensuite transtyper. Cette méthode est toujours utilisable mais partager des données via des champs statiques typés de l'objet [Global] évite les transtypages et permet au compilateur de faire des vérifications de type qui aident le développeur. C'est la méthode qui sera utilisée ici.

Les données partagées par tous les utilisateurs sont les suivantes :

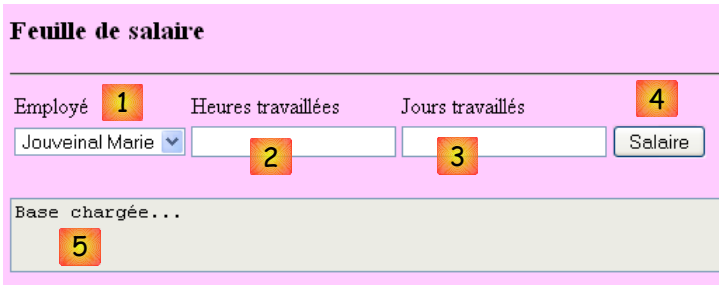
- ligne 12 : le tableau d'objets de type [Employe] qui mémorisera la liste simplifiée (SS, NOM, PRENOM) de tous les employés
- ligne 13 : l'objet de type [Cotisations] qui mémorisera les taux de cotisations
- ligne 14 : le dictionnaire qui mémorisera les indemnités liés aux différents indices des employés. Il sera indexé par l'indice de l'employé et ses valeurs seront de type [Indemnites]
- ligne 15 : un message indiquant comment s'est terminée l'initialisation (bien ou avec erreur)
- ligne 16 : un booléen indiquant si l'initialisation s'est terminée par une erreur ou non.
- ligne 17 : la chaîne de connexion à la base de données.

Question : compléter le code de la classe [Application_Start].

4.3 Les événements du formulaire [Default.aspx]

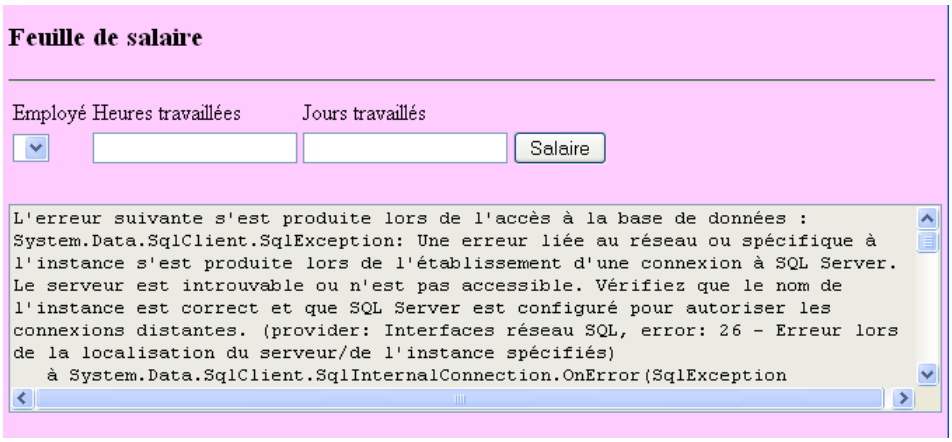
4.3.1 La procédure [Page_Load]

Lorsque le formulaire [Default.aspx] est chargé, il va chercher dans le tableau [Global.Employes] (ligne 12) les noms des employés pour les mettre dans la liste déroulante [1] :



- la liste déroulante [1] a été remplie
- le TextBox [5] indique que la base des données a été lue correctement

S'il s'est produit des erreurs d'initialisation lors du démarrage de l'application, le TextBox [5] l'indique :



Question : écrire la procédure [Page_Load] de la page web [Default.aspx] qui, exécutée au démarrage de l'application, garantit le fonctionnement précédent.

4.3.2 Le calcul du salaire

Le clic sur le bouton [4] provoque l'exécution du gestionnaire :

```
    protected void ButtonSalaire_Click(object sender, System.EventArgs e)
```

Ce gestionnaire commence par vérifier la validité des saisies faites en [2] et [3]. Si l'une des deux est incorrecte, l'erreur est signalée comme il a été montré précédemment. Une fois les saisies [2] et [3] vérifiées et trouvées valides, l'application doit afficher des données complémentaires sur l'utilisateur sélectionné en [1] ainsi que son salaire (cf copie d'écran page 43).

Question : écrire le code de la procédure [ButtonSalaire_Click].

5 L'application [SimuPaie] – version 2 – AJAX / ASP.NET

5.1 Introduction

La technologie AJAX (**A**synchronous **J**avascript **A**nd **X**ml) réunit un ensemble de technologies :

- **Javascript** : une page HTML affichée dans un navigateur peut embarquer du code Javascript. Ce code est exécuté par le navigateur si l'utilisateur n'a pas inhibé l'exécution du code Javascript sur son navigateur. Cette technologie est apparue dès les débuts du web et suscite depuis 2005 un regain d'intérêt grâce à AJAX.
- **DOM** : **D**ocument **O**bject **M**odel. Le code Javascript embarqué dans une page HTML a accès au document sous la forme d'un arbre d'objets, le DOM. Chaque élément du document (un champ de saisie, une balise nommée, une balise <select>,... est représenté par un noeud de l'arbre. Le code Javascript peut changer le document affiché en modifiant le DOM. Par exemple, il peut changer le texte affiché dans un champ de saisie via le noeud du DOM représentant ce champ.

AJAX utilise Javascript pour faire des échanges navigateur / serveur en tâche de fond. Pour réagir à un événement tel que le clic sur un bouton, du code Javascript va être exécuté par le navigateur pour envoyer une requête HTTP au serveur. Cette requête va contenir des paramètres indiquant au serveur ce qu'il doit faire. Celui-ci va exécuter l'action demandée. Il va envoyer en réponse au navigateur, un flux HTTP contenant des données permettant une mise à jour partielle de la page actuellement affichée. Celle-ci sera réalisée via le DOM du document et l'exécution de code Javascript embarqué dans le document. Les données renvoyées par le serveur peut l'être sous différents formats : XML, HTML, JSON, texte non formaté, ...

L'intérêt de la technologie réside dans cette mise à jour partielle du document affiché. Cela signifie :
- moins de données échangées entre le client et le serveur et donc une réponse plus rapide
- une interface graphique plus fluide à utiliser car les actions de l'utilisateur ne provoquent pas le rechargement complet de la page.

Dans un intranet où les échanges réseau sont rapides, AJAX permet de créer des applications web ayant un comportement qui se rapproche de celui des interfaces windows classiques. On peut en effet gérer des événements tels que le changement de sélection dans une liste et rafraîchir immédiatement et partiellement la page pour refléter ce changement de sélection. Le fait que la page ne soit pas totalement rechargée donne à l'utilisateur l'impression de fluidité qu'il a avec les applications windows. Pour des applications Internet, où les temps de réponse peuvent être longs, la technologie AJAX reste utilisable mais l'impression de fluidité est dépendante de celle du réseau.

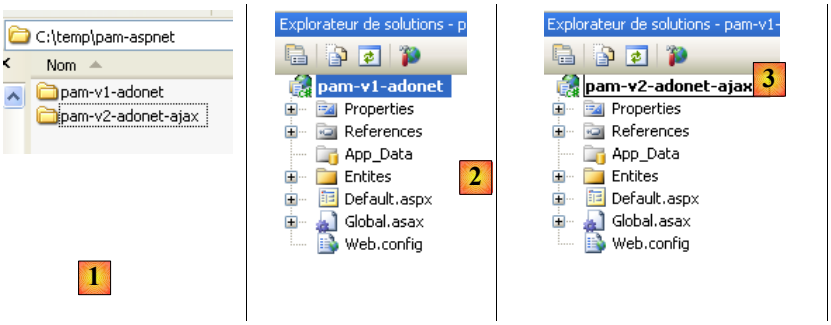
La principale difficulté dans AJAX est le langage Javascript. Créé dès les débuts du web,

- il s'avère peu pratique pour faire de la programmation orientée objet. Ce n'est pas, par exemple, un langage typé. On n'y déclare pas le type des données manipulées. On se rapproche là de langages tels que Perl ou PHP.
- il n'a pas un standard accepté par tous les navigateurs. Chacun d'eux a ses extensions "Javascript" propriétaires qui font qu'un code Javascript écrit pour IE pourra ne pas fonctionner dans Mozilla Firefox et vice-versa.
- il est difficile à déboguer, les navigateurs n'offrant pas d'outils performants de débogage du code Javascript qu'ils exécutent.

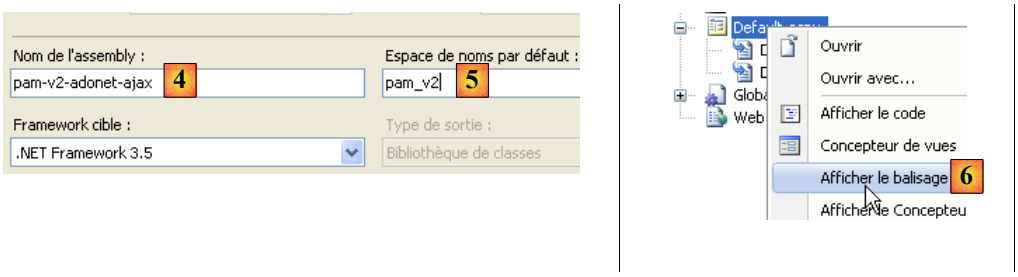
Tous ces maux du Javascript ont fait qu'il était peu utilisé avant l'arrivée d'AJAX. Une fois compris l'intérêt de faire exécuter du code Javascript en tâche de fond, pour faire des requêtes HTTP au serveur web et utiliser la réponse de celui-ci pour faire, grâce au DOM, une mise à jour partielle du document affiché, les équipes de développement se sont mises au travail et ont proposé des frameworks permettant de faire de l'AJAX sans que le développeur ait à écrire du code Javascript. Pour cela, des bibliothèques Javascript qui savent s'adapter au navigateur client ont été écrites. L'insertion de code Javascript dans la page HTML envoyée au navigateur est faite côté serveur, selon des techniques qui diffèrent selon le framework AJAX utilisé. Il existe des frameworks Ajax aussi bien pour Java, .NET, PHP, Ruby, ... AJAX est intégré dans Visual Web Developer 2008.

5.2 Le projet Visual Web Developer de la couche [web-ui-ajax]

La nouvelle version est obtenue par duplication de la version précédente. Seule la page [Default.aspx] va être modifiée.



- en [1] avec l'explorateur windows on duplique le dossier du projet [pam-v1-adonet] et on lui donne le nom [pam-v2-adonet-ajax]. Puis avec Visual Web Developer, on ouvre le projet (Fichier / Ouvrir un projet) de ce nouveau dossier [2]
- en [3], on lui donne un nouveau nom

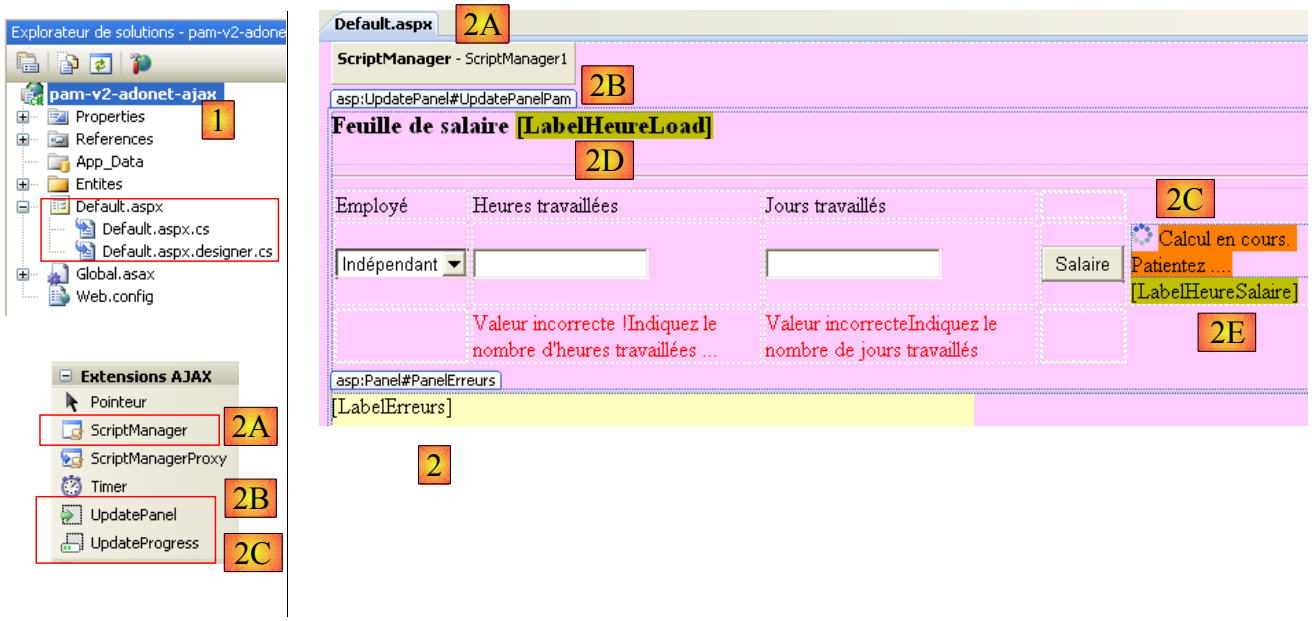


- dans les propriétés du nouveau projet, on change le nom de l'assembly [4] ainsi que l'espace de noms par défaut [5].

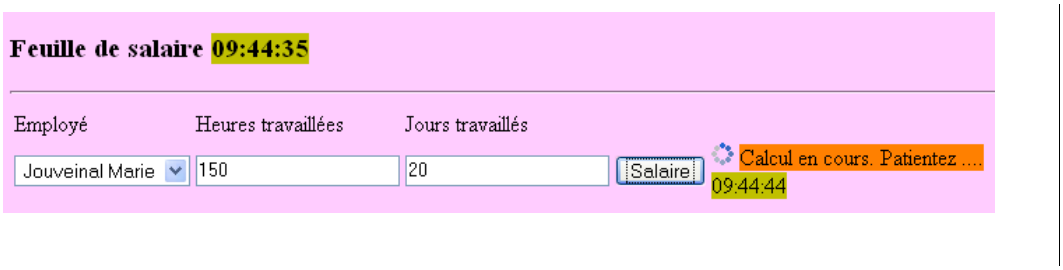
Ceci fait, on remplace dans les fichiers [Default.aspx, Default.aspx.cs, Default.aspx.designer.cs, Global.asax, Global.asax.cs] l'espace de noms [pam_v1] par l'espace de noms [pam_v2] afin d'être cohérent avec l'espace de noms par défaut. Par exemple, [Default.aspx] [6] devient :

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs" Inherits="pam_v2._Default" %>
...
```

La page [Default.aspx] va être modifiée de la façon suivante :



- ajout de composants Ajax dans le formulaire [Default.aspx] (cf [2] ci-dessus).
  - 2A : le composant <*asp:ScriptManager*> est nécessaire pour tout projet AJAX
  - 2B : le composant <*asp:UpdatePanel*> sert à délimiter la zone à rafraîchir lors d'un POST de l'utilisateur. Ce composant évite le rechargement total de la page.
  - 2C : le composant <*asp:UpdateProgress*> sert à afficher un texte ou une image pendant la durée de la mise à jour afin d'en avertir l'utilisateur comme montré ci-dessous :



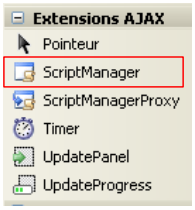
  - 2D : un type *Label* nommé *LabelHeureLoad* qui va afficher l'heure à laquelle s'exécute le gestionnaire de l'événement *Load* de la page.
  - 2E : un type *Label* nommé *LabelHeureSalaire* qui va afficher l'heure à laquelle s'exécute le gestionnaire de l'événement *Click* sur le bouton [Salaire]. On veut montrer qu'Ajax ne recharge pas toute la page lors du clic sur le bouton [Salaire]. C'est ce que montre la copie d'écran précédente où on peut voir deux heures différentes dans les deux *Labels*.

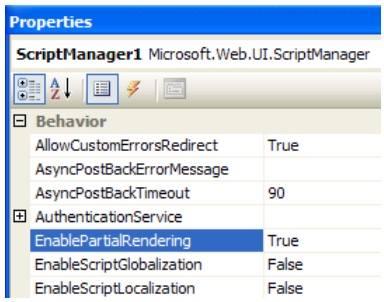
L'Ajaxification précédente du formulaire peut se faire directement dans le code source de [Default.aspx] par l'ajout de balises d'extensions Ajax :

```
1.  ...
2.  <body style="text-align: left" bgcolor="#ffccff">
3.    <h3>
4.      Feuille de salaire
5.      <asp:Label ID="LabelHeureLoad" runat="server" BackColor="#C0C000"></asp:Label></h3>
6.    <hr />
7.    <form id="form1" runat="server">
8.      <asp:ScriptManager ID="ScriptManager1" runat="server" EnablePartialRendering="true" />
9.      <asp:UpdatePanel runat="server" ID="UpdatePanelPam" UpdateMode="Conditional">
10.       <ContentTemplate>
11.         <div>
12.           <table>
13.             <tr>
14. ...
15.             <tr>
16.               <td>
17.                 <asp:DropDownList ID="ComboBoxEmployes" runat="server">
18.                 </asp:DropDownList></td>
19.               <td>
20.                 <asp:TextBox ID="TextBoxHeures" runat="server" EnableViewState="False">
21.                 </asp:TextBox></td>
22.               <td>
23.                 <asp:TextBox ID="TextBoxJours" runat="server" EnableViewState="False">
24.                 </asp:TextBox></td>
25.               <td>
26.                 <asp:Button ID="ButtonSalaire" runat="server" Text="Salaire"
    CausesValidation="False" /></td>
27.                 <td>   <asp:UpdateProgress ID="UpdateProgress1" runat="server">
28.                   <ProgressTemplate>
29.                     <img src="images/indicator.gif" />
30.                     <asp:Label ID="Label5" runat="server" BackColor="#FF8000"
    EnableViewState="False"
31.                       Text="Calcul en cours. Patientez ...."></asp:Label>
32.                   </ProgressTemplate>
33.                 </asp:UpdateProgress>
34.           <asp:Label ID="LabelHeureSalaire" runat="server" BackColor="#C0C000"></asp:Label></td>
35.             </tr>
36.             <tr>
37. ...
38.             </tr>
39.           </table>
40.         </div>
41.         <br />
```

```
42. ....
43.           <table>
44.             <tr>
45.               <td>
46.                 Salaire net à payer :
47.               </td>
48.               <td align="center" bgcolor="#C0C000" height="20px">
49.                 <asp:Label ID="LabelSN" runat="server" EnableViewState="False"></asp:Label></td>
50.             </tr>
51.           </table>
52.            </asp:Panel>
53.       </ContentTemplate>
54.     </asp:UpdatePanel>
55.   </form>
56. </body>
57. </html>
```

- ligne 8 : tout formulaire Ajaxifié doit inclure la balise <asp:ScriptManager>. C'est cette balise qui permet l'utilisation des nouveaux composants Ajax :



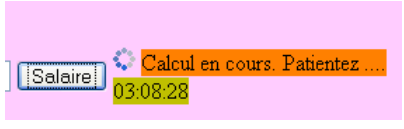


- la balise <*asp:ScriptManager*> peut être obtenue par double-clic sur le composant [ScriptManager] ci-dessus. Il faut prendre soin de vérifier que cette balise est contenue dans la balise <form> du code source. L'attribut [EnablePartialRendering= "true "] de la ligne 8 est absent pas défaut. La valeur " true " étant sa valeur par défaut, il n'est pas indispensable.

- ligne 9 : la balise <*asp:UpdatePanel*> permet de délimiter les zones de la page qui doivent être mises à jour, lors d'une mise à jour partielle de la page. L'attribut [*UpdateMode="Conditional"*] indique que la zone ne doit être mise à jour que sur un événement AJAX d'un composant de la zone. L'autre valeur est [*UpdateMode="Always"*] et c'est la valeur par défaut. Avec cet attribut, la zone *UpdatePanel* est mise à jour systématiquement même si l'événement Ajax qui s'est produit provient d'un composant d'une autre zone *UpdatePanel*. En général, ce comportement n'est pas désirable.

  La balise <*asp:UpdatePanel*> admet deux balises enfant : <*ContentTemplate*> et <*Triggers*>.

- les balises <*asp:ContentTemplate*>, lignes 10 et 53, délimitent la zone de la page à mettre à jour partiellement.
- lignes 27-33 : un composant Ajax <*asp:UpdateProgress*> permettant d'afficher un texte pendant toute la durée de la mise à jour de la page. Par exemple, le clic sur le bouton [Salaire] va provoquer un POST en arrière-plan. Le navigateur ne met alors pas le sablier et l'utilisateur peut être ainsi tenté de continuer à utiliser le formulaire. La balise <*asp:UpdateProgress*> permet d'afficher un texte avertissant qu'une mise à jour de la page est en cours. On peut également afficher une image. Ici, on affiche une image animée (ligne 29) ainsi qu'un texte (lignes 30-31) :



- lignes 28-32 : la balise <*ProgressTemplate*> délimite le contenu qui sera affiché pendant toute la durée de la mise à jour de la zone *UpdatePanel* dans laquelle se trouve la balise *UpdateProgress*.
- lignes 29-31 : l'image animée et le texte qui seront affichés pendant la mise à jour de la zone *UpdatePanel* .
- ligne 5 : le label *LabelHeureLoad* est mis à l'extérieur de la zone ajaxifiée

- ligne 34 : le label *LabelHeureSalaire* est mis dans la zone ajaxifiée

Le code de la page [Default.aspx.cs] évolue lui de la façon suivante :

```
1.      protected void Page_Load(object sender, System.EventArgs e)
2.      {
3.          // heure de chaque chargement de page
4.          LabelHeureLoad.Text = DateTime.Now.ToString("hh:mm:ss");
5.          // traitement requête initiale
6.          if (!IsPostBack)
7.          {
8.  ....
9.  }
```

- ligne 4 : mise à jour de l'heure de chargement de la page

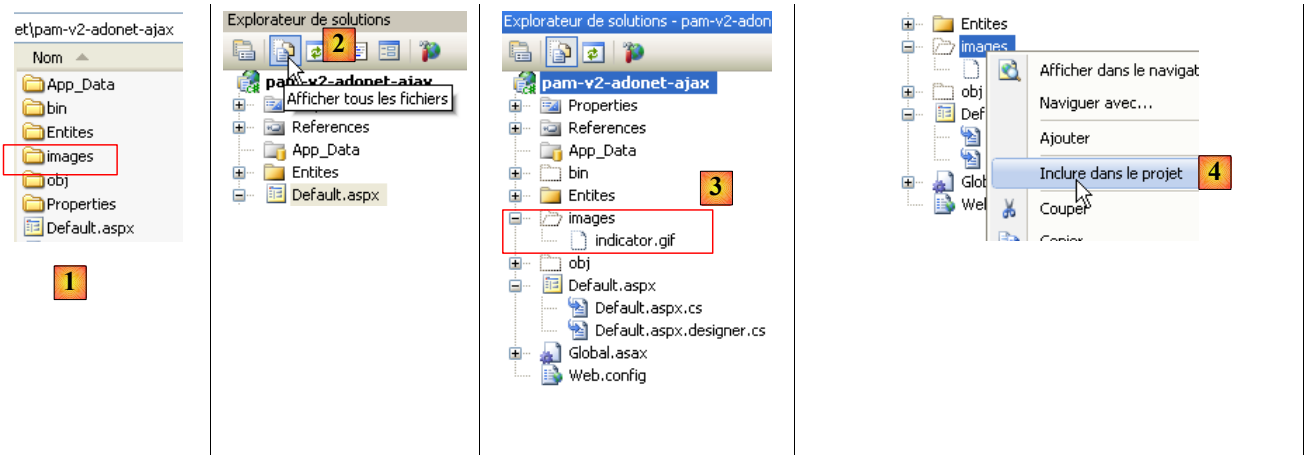
```
1.      protected void ButtonSalaire_Click(object sender, System.EventArgs e)
2.      {
3.          // heure calcul salaire
4.          LabelHeureSalaire.Text = DateTime.Now.ToString("hh:mm:ss");
5.          // on vérifie les données
6.  ....
7.  }
```

- ligne 4 : mise à jour de l'heure de calcul du salaire

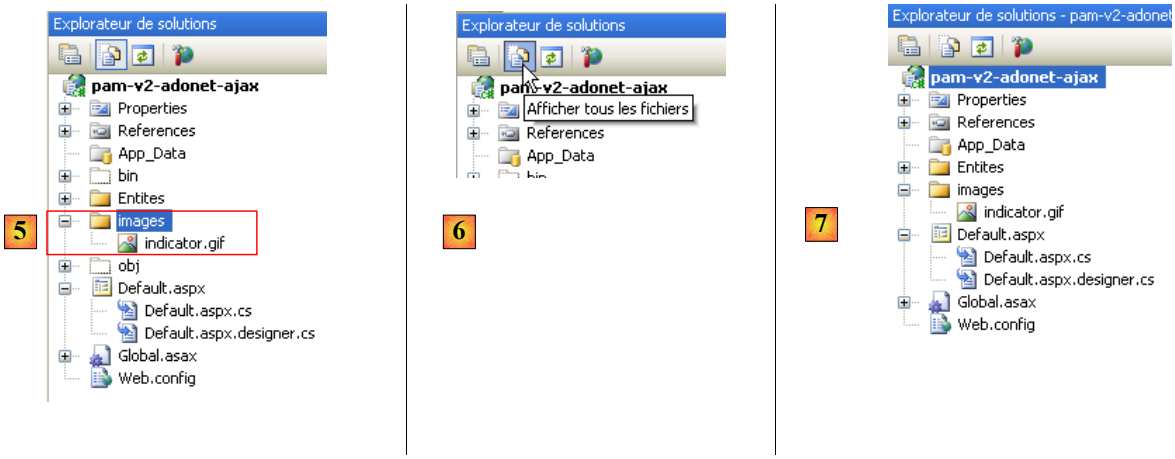
Pour terminer la page [Default.aspx], il nous faut inclure l'image animée référencée par la ligne 4 du code source suivant de la page :

```
1.              <td>
2.                <asp:UpdateProgress ID="UpdateProgress1" runat="server">
3.                  <ProgressTemplate>
4.                    <img src="images/indicator.gif" />
5.                    <asp:Label ID="Label5" runat="server" BackColor="#FF8000"
    EnableViewState="False"
6.                      Text="Calcul en cours. Patientez ...."></asp:Label>
7.                  </ProgressTemplate>
8.                </asp:UpdateProgress>
9.                <asp:Label ID="LabelHeureSalaire" runat="server" BackColor="#C0C000"></asp:Label>
10. </td>
```

Avec l'explorateur windows, nous rajoutons le fichier [images/indicator.gif] au dossier du projet [1] :



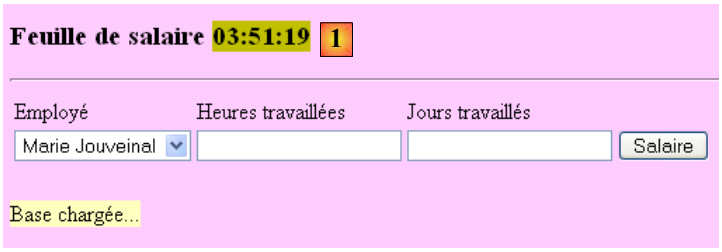
- en [2], on fait afficher tous les fichiers du projet
- en [3], le dossier [images]
- en [4], on inclut le dossier [images] dans le projet



- en [5], le dossier [images] a été inclus dans le projet
- en [6], on cache les fichiers qui ne font pas partie du projet
- en [7], le nouveau projet

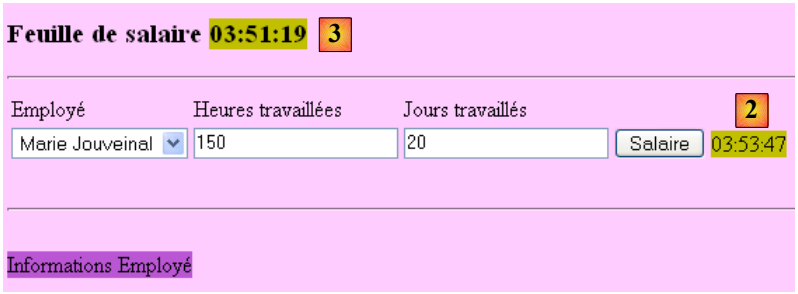
5.3 Tests de la solution Ajax

Lorsqu'on demande l'exécution du projet (CTRL-F5), on obtient la page suivante :

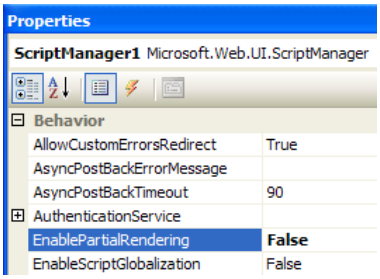


- en [1], l'heure de chargement de la page

Faites des essais et vérifiez que le bouton [Salaire] ne provoque pas le rechargement complet de la page. Vous pouvez le voir avec l'heure affichée pour le traitement du clic sur le bouton [Salaire] [2] qui n'est pas le même que l'heure du chargement initial de la page [3] :



Refaire les tests en mettant la propriété *EnablePartialRendering* du composant *ScriptManager1* à *False* :

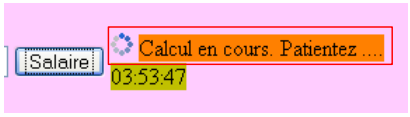


Constater qu'alors on retrouve le comportement d'une page non ajaxifiée. Il y a rechargement total de la page lors du clic sur le bouton [Salaire]. Refaire les tests avec un autre navigateur. Le test multi-navigateurs a pour but de montrer que le javascript généré par les composants serveur ASP / AJAX est correctement interprété par les différents navigateurs.

Pour finir, mettons en lumière le rôle de la balise <*asp:UpdateProgress*>. Dans le code de la procédure [ButtonSalaire_Click] de [Default.aspx.cs], ajoutons une instruction qui arrête la procédure pendant 3 secondes :

```
1.  using System.Threading;
2.      ...
3.  protected void ButtonSalaire_Click(object sender, System.EventArgs e)
4.      {
5.          // heure calcul salaire
6.          LabelHeureSalaire.Text = DateTime.Now.ToString("hh:mm:ss");
7.          // attente
8.          Thread.Sleep(3000);
9.          // on vérifie les données
10. ....
11. }
```

La ligne 8 fait attendre 5 secondes le thread qui exécute la procédure [ButtonSalaire_Click]. Ceci fait, refaisons des tests (après avoir remis l'attribut *EnablePartialRendering* du composant *ScriptManager1* à True). Cette fois, on voit le texte de [UpdateProgress] ainsi que l'image animée pendant le calcul du salaire :



5.4 Conclusion

L'étude précédente nous a montré qu'il était possible d'ajaxifier des applications ASP.NET existantes. Les extensions AJAX d'ASP.NET vont plus loin que ce qui vient d'être vu. Le lecteur est invité à consulter le site [http://ajax.asp.net].

6 Introduction à l'ORM NHIBERNATE

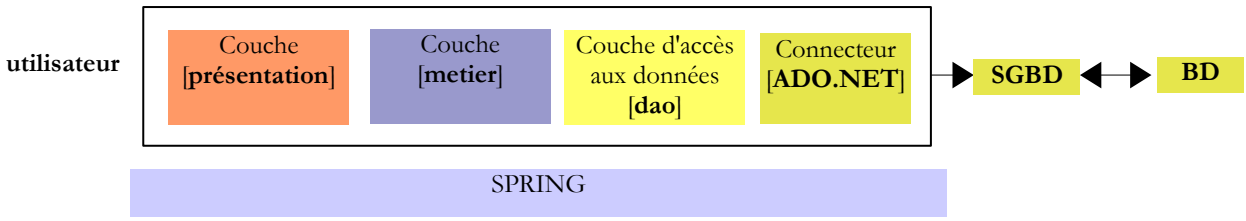
Note : ce chapitre est une introduction succincte à NHibernate, l'équivalent pour .Net du framework Java Hibernate. Pour une introduction complète on pourra lire :

Titre : NHibernate in Action, **Auteur** : Pierre-Henri Kuaté, **Editeur** : Manning, **ISBN-13** : 978-1932394924

Un ORM (Object Relational Mapper) est un ensemble de bibliothèques permettant à un programme exploitant une base de données d'exploiter celle-ci sans émettre d'ordres SQL explicites et sans connaître les particularités du SGBD utilisé.

6.1 La place de NHIBERNATE dans une architecture .NET en couches

Une application .NET utilisant une base de données peut être architecturée en couches de la façon suivante :



La couche [dao] communique avec le SGBD via l'API ADO.NET. Rappelons les principales méthodes de cette API.

En mode **connecté**, l'application :

4. ouvre une connexion avec la source de données
5. travaille avec la source de données en lecture/écriture
6. ferme la connexion

Trois interfaces ADO.NET sont principalement concernées par ces opérations :

- **IDbConnection** qui encapsule les propriétés et méthodes de la connexion.
- **IDbCommand** qui encapsule les propriétés et méthodes de la commande SQL exécutée.
- **IDataReader** qui encapsule les propriétés et méthodes du résultat d'un ordre SQL Select.

L'interface IDbConnection

Sert à gérer la connexion avec la base de données. Parmi les méthodes **M** et propriétés **P** de cette interface on trouve les suivantes :

Nom	Type	Rôle
`ConnectionString`	P	**chaîne de connexion** à la base. Elle précise tous les paramètres nécessaires à l'établissement de la connexion avec une base précise.
`Open`	M	ouvre la connexion avec la base définie par *ConnectionString*
`Close`	M	ferme la connexion
`BeginTransaction`	M	démarre une transaction.
`State`	P	état de la connexion : *ConnectionState.Closed*, *ConnectionState.Open*, *ConnectionState.Connecting*, *ConnectionState.Executing*, *ConnectionState.Fetching*, *ConnectionState.Broken*

Si *Connection* est une classe implémentant l'interface *IDbConnection*, l'ouverture de la connexion peut se faire comme suit :

```
1.  IDbConnection connexion=new Connection();
2.  connexion.ConnectionString=...;
3.  connexion.Open();
```

L'interface IDbCommand

Sert à exécuter un ordre SQL ou une procédure stockée. Parmi les méthodes **M** et propriétés **P** de cette interface on trouve les suivantes :

Nom	Type	Rôle
CommandType	P	indique ce qu'il faut exécuter - prend ses valeurs dans une énumération : - *CommandType.Text* : exécute l'ordre SQL défini dans la propriété *CommandText*. C'est la valeur par défaut. - *CommandType.StoredProcedure* : exécute une procédure stockée dans la base
CommandText	P	- le texte de l'ordre SQL à exécuter si *CommandType= CommandType.Text* - le nom de la procédure stockée à exécuter si *CommandType= CommandType.StoredProcedure*
Connection	P	la connexion *IDbConnection* à utiliser pour exécuter l'ordre SQL
Transaction	P	la transaction *IDbTransaction* dans laquelle exécuter l'ordre SQL
Parameters	P	la liste des paramètres d'un ordre SQL paramétré. L'ordre *update articles set prix=prix*1.1 where id=@id* a le paramètre @id.
ExecuteReader	M	pour exécuter un ordre SQL *Select*. On obtient un objet *IDataReader* représentant le résultat du *Select*.
ExecuteNonQuery	M	pour exécuter un ordre SQL *Update, Insert, Delete*. On obtient le nombre de lignes affectées par l'opération (mises à jour, insérées, détruites).
ExecuteScalar	M	pour exécuter un ordre SQL *Select* ne rendant qu'un unique résultat comme dans : *select count(*) from articles*.
CreateParameter	M	pour créer les paramètres *IDbParameter* d'un ordre SQL paramétré.
Prepare	M	permet d'optimiser l'exécution d'une requête paramétrée lorsqu'elle est exécutée de multiples fois avec des paramètres différents.

Si *Command* est une classe implémentant l'interface *IDbCommand*, l'exécution d'un ordre SQL sans transaction aura la forme suivante :

```
1.  // ouverture connexion
2.  IDbConnection connexion=...
3.  connexion.Open();
4.  // préparation commande
5.  IDbCommand commande=new Command();
6.  commande.Connection=connexion;
7.  // exécution ordre select
8.  commande.CommandText="select ...";
9.  IDbDataReader reader=commande.ExecuteReader();
10. ...
11. // exécution ordre update, insert, delete
12. commande.CommandText="insert ...";
13. int nbLignesInsérées=commande.ExecuteNonQuery();
14. ...
15. // fermeture connexion
16. connexion.Close();
```

L'interface IDataReader

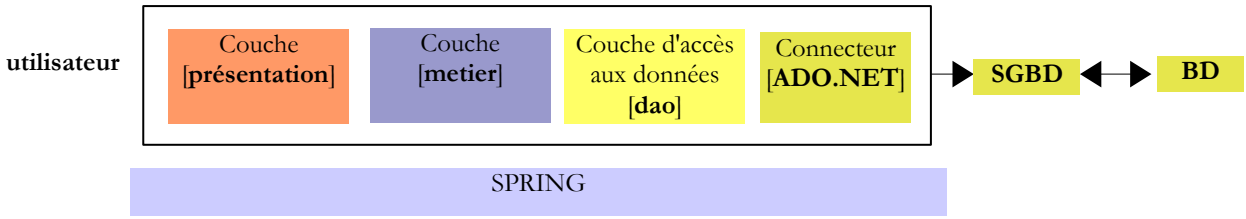
Sert à encapsuler les résultats d'un ordre SQL *Select*. Un objet *IDataReader* représente une table avec des lignes et des colonnes, qu'on exploite séquentiellement : d'abord la 1ère ligne, puis la seconde, ... Parmi les méthodes **M** et propriétés **P** de cette interface on trouve les suivantes :

Nom	Type	Rôle
FieldCount	P	le nombre de colonnes de la table *IDataReader*
GetName	M	*GetName(i)* rend le nom de la colonne n° i de la table *IDataReader*.
Item	P	*Item[i]* représente la colonne n° i de la ligne courante de la table *IDataReader*.
Read	M	passe à la ligne suivante de la table *IDataReader*. Rend le booléen *True* si la lecture a pu se faire, *False* sinon.
Close	M	ferme la table *IDataReader*.
GetBoolean	M	*GetBoolean*(i) : rend la valeur booléenne de la colonne n° i de la ligne courante de la table *IDataReader*. Les autres méthodes analogues sont les suivantes : *GetDateTime*, *GetDecimal*, *GetDouble*, *GetFloat*, *GetInt16*, *GetInt32*, *GetInt64*, *GetString*.
Getvalue	M	*Getvalue*(i) : rend la valeur de la colonne n° i de la ligne courante de la table *IDataReader* en tant que type *object*.
IsDBNull	M	*IsDBNull(i)* rend *True* si la colonne n° i de la ligne courante de la table *IDataReader* n'a pas de valeur ce qui est symbolisé par la valeur SQL NULL.

L'exploitation d'un objet *IDataReader* ressemble souvent à ce qui suit :

```
1.  // ouverture connexion
2.  IDbConnection connexion=...
3.  connexion.Open();
4.  // préparation commande
5.  IDbCommand commande=new Command();
6.  commande.Connection=connexion;
7.  // exécution ordre select
8.  commande.CommandText="select ...";
9.  IDataReader reader=commande.ExecuteReader();
10. // exploitation résultats
11. while(reader.Read()){
12.  // exploiter ligne courante
13.     ...
14. }
15. // fermeture reader
16. reader.Close();
17. // fermeture connexion
18. connexion.Close();
```

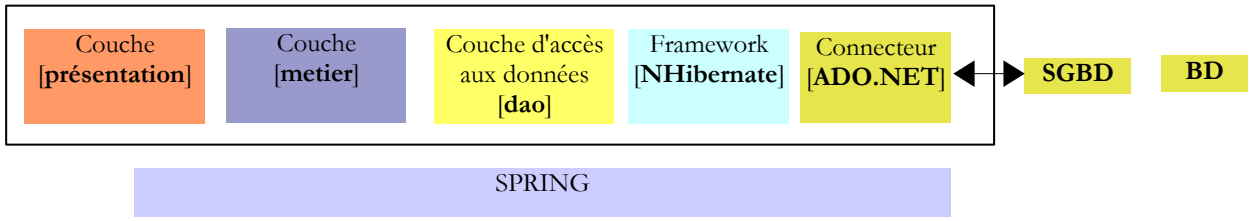
Dans l'architecture précédente,



le connecteur [ADO.NET] est lié au SGBD. Ainsi la classe implémentant l'interface [IDbConnection] est :

- la classe [MySQLConnection] pour le SGBD MySQL
- la classe [SQLConnection] pour le SGBD SQLServer

La couche [dao] est ainsi dépendante du SGBD utilisé. Certains frameworks (Linq, Ibatis.net, NHibernate) lèvent cette contrainte en ajoutant une couche supplémentaire entre la couche [dao] et le connecteur [ADO.NET] du SGBD utilisé. Nous utiliserons ici, le framework [NHibernate].



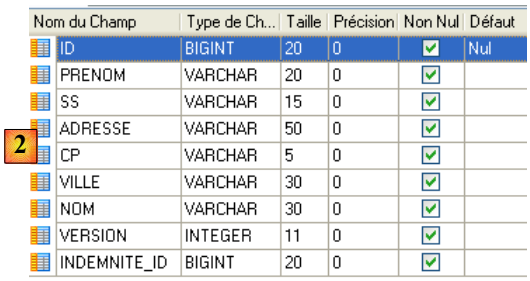
Ci-dessus, la couche [dao] ne s'adresse plus au connecteur [ADO.NET] mais au framework NHibernate qui va lui présenter une interface indépendante du connecteur [ADO.NET] utilisé. Cette architecture permet de changer de SGBD sans changer la couche [dao]. Seul le connecteur [ADO.NET] doit être alors changé.

6.2 La base de données exemple

Pour montrer comment travailler avec NHibernate, nous utiliserons la base de données MySQL [dbpam_nhibernate] suivante :

- en [1], la base a trois tables :
  - [employes] : une table qui enregistre les employées d'une crèche
  - [cotisations] : une table qui enregistre des taux de cotisations sociales
  - [indemnites] : une table qui enregistre des informations permettant de calculer la paie des employées

Table [employes]



ID	clé primaire de type autoincrement
VERSION	n° de version de l'enregistrement
PRENOM	prénom de l'employée
NOM	son nom
ADRESSE	son adresse
CP	son code postal
VILLE	sa ville
INDEMNITE_ID	clé étrangère sur INDEMNITES(ID)

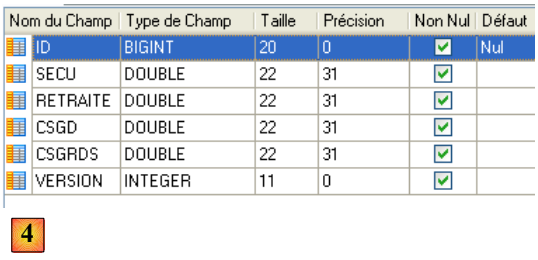
3

- en [2], la table des employés et en [3], la signification de ses champs

Le contenu de la table pourrait être le suivant :

ID	PRENOM	SS	ADRESSE	CP	VILLE	NOM	VERSION	INDEMNITE_ID
5	Marie	254104940426058	5 rue des oiseaux	49203	St Corentin	Jouveinal	1	7
6	Justine	260124402111742	La Brûlerie	49014	St Marcel	Laverti	1	8

Table [cotisations]



ID	clé primaire de type autoincrement
VERSION	n° de version de l'enregistrement
SECU	taux (pourcentage) de cotisation pour la sécurité sociale
RETRAITE	taux de cotisation pour la retraite
CSGD	taux de cotisation pour la contribution sociale généralisée déductible
CSGRDS	taux de cotisation pour la contribution sociale généralisée et la contribution au remboursement de la dette sociale

5

- en [4], la table des cotisations et en [5], la signification de ses champs

Le contenu de la table pourrait être le suivant :

ID	SECU	RETRAITE	CSGD	CSGRDS	VERSION
3	9,39	7,88	6,15	3,49	1

Table [indemnites]

Nom du Champ	Type de Cha...	Taille	Précision	Non Nul	Défaut
ID	BIGINT	20	0	☑	Nul
ENTRETIEN_JOUR	DOUBLE	22	31	☑	
REPAS_JOUR	DOUBLE	22	31	☑	
INDICE	INTEGER	11	0	☑	
INDEMNITES_CP	DOUBLE	22	31	☑	
BASE_HEURE	DOUBLE	22	31	☑	
VERSION	INTEGER	11	0	☑	

6

ID	clé primaire de type autoincrement
VERSION	n° de version de l'enregistrement
BASE_HEURE	coût en euro d'une heure de garde
ENTRETIEN_JOUR	indemnité en euro par jour de garde
REPAS_JOUR	indemnité de repas en euro par jour de garde
INDEMNITES_CP	indemnités de congés payés. C'est un pourcentage à appliquer au salaire de base.

7

- en [6], la table des indemnités et en [7], la signification de ses champs

Le contenu de la table pourrait être le suivant :

ID	ENTRETIEN_JOUR	REPAS_JOUR	INDICE	INDEMNITES_CP	BASE_HEURE	VERSION
7	2	3	1	12	1,93	1
8	2,1	3,1	2	15	2,1	1

L'exportation de la structure de la base vers un fichier SQL donne le résultat suivant :

```
#
# Structure for the `cotisations` table :
#

CREATE TABLE `cotisations` (
  `ID` bigint(20) NOT NULL auto_increment,
  `SECU` double NOT NULL,
  `RETRAITE` double NOT NULL,
  `CSGD` double NOT NULL,
  `CSGRDS` double NOT NULL,
  `VERSION` int(11) NOT NULL,
  PRIMARY KEY  (`ID`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=latin1;

#
# Structure for the `indemnites` table :
#

CREATE TABLE `indemnites` (
  `ID` bigint(20) NOT NULL auto_increment,
  `ENTRETIEN_JOUR` double NOT NULL,
  `REPAS_JOUR` double NOT NULL,
  `INDICE` int(11) NOT NULL,
  `INDEMNITES_CP` double NOT NULL,
  `BASE_HEURE` double NOT NULL,
  `VERSION` int(11) NOT NULL,
  PRIMARY KEY  (`ID`),
  UNIQUE KEY `INDICE` (`INDICE`)
) ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT CHARSET=latin1;

#
# Structure for the `employes` table :
#

CREATE TABLE `employes` (
  `ID` bigint(20) NOT NULL auto_increment,
  `PRENOM` varchar(20) NOT NULL,
  `SS` varchar(15) NOT NULL,
```

```
39.   `ADRESSE` varchar(50) NOT NULL,
40.   `CP` varchar(5) NOT NULL,
41.   `VILLE` varchar(30) NOT NULL,
42.   `NOM` varchar(30) NOT NULL,
43.   `VERSION` int(11) NOT NULL,
44.   `INDEMNITE_ID` bigint(20) NOT NULL,
45.   PRIMARY KEY  (`ID`),
46.   UNIQUE KEY `SS` (`SS`),
47.   KEY `FK_EMPLOYES_INDEMNITE_ID` (`INDEMNITE_ID`),
48.   CONSTRAINT `FK_EMPLOYES_INDEMNITE_ID` FOREIGN KEY (`INDEMNITE_ID`) REFERENCES `indemnites`
    (`ID`)
49. ) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=latin1;
```

On notera, lignes 6, 20 et 36 que les clés primaires ID ont l'attribut *autoincrement*. Ceci signifie que MySQL génèrera automatiquement les valeurs des clés primaires à chaque ajout d'un enregistrement. Le développeur n'a pas à s'en préoccuper.

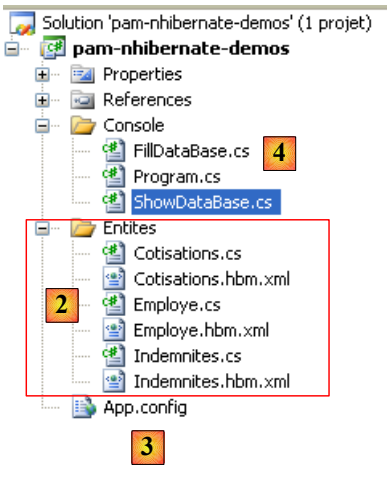
6.3 Le projet C# de démonstration

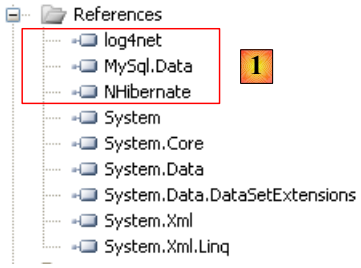
Pour introduire la configuration et l'utilisation de NHibernate, nous utiliserons l'architecture suivante :

Un programme console [1] manipulera les données de la base de données précédente [2] via le framework [NHibernate] [3]. Cela nous amènera à présenter :

- les fichiers de configuration de NHibernate
- l'API de NHibernate

Le projet C# sera le suivant :



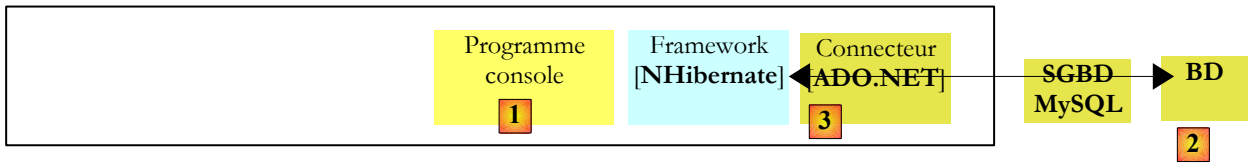


Les éléments nécessaires au projet sont les suivants :

- en [1], les DLL dont a besoin le projet :
  - [NHibernate] : la DLL du framework NHibernate
  - [MySql.Data] : la DLL du connecteur ADO.NET du SGBD MySQL
  - [log4net] : la DLL d'un outil permettant de générer des logs
- en [2], les classes images des tables de la base de données
- en [3], le fichier [App.config] qui configure l'application tout entière, dont le framework [NHibernate]
- en [4], des applications console de test

6.3.1 Configuration de la connexion à la base de données

Revenons à l'architecture de test :



Ci-dessus, [NHibernate] doit pouvoir accéder à la base de données. Pour cela, il a besoin de certaines informations :

- le SGBD qui gère la base (MySQL, SQLServer, Postgres, Oracle, ...). La plupart des SGBD ont ajouté au langage SQL des extensions qui leur sont propres. En connaissant le SGBD, NHibernate peut adapter les ordres SQL qu'il émet à ce SGBD. NHibernate utilise la notion de **dialecte SQL**.
- les paramètres de connexion à la base de données (nom de la base, nom de l'utilisateur propriétaire de la connexion, son mot de passe)

Ces informations peuvent être placées dans le fichier de configuration [App.config]. Voici celui qui sera utilisé avec une base MySQL :

```
1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <configuration>
3.    <!-- sections de configuration -->
4.    <configSections>
5.       <section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler,log4net" />
6.       <section name="hibernate-configuration" type="NHibernate.Cfg.ConfigurationSectionHandler,
    NHibernate" />
7.    </configSections>
8.
9.
10.   <!-- configuration NHibernate -->
11.   <hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">
12.      <session-factory>
13.         <property
    name="connection.provider">NHibernate.Connection.DriverConnectionProvider</property>
14.         <!--
15.         <property name="connection.driver_class">NHibernate.Driver.MySqlDataDriver</property>
16.         -->
17.         <property name="dialect">NHibernate.Dialect.MySQLDialect</property>
18.         <property name="connection.connection_string">
19.            Server=localhost;Database=dbpam_nhibernate;Uid=root;Pwd=;
20.         </property>
21.         <property name="show_sql">false</property>
22.         <mapping assembly="pam-nhibernate-demos"/>
23.      </session-factory>
24.   </hibernate-configuration>
25.
26.   <!-- This section contains the log4net configuration settings -->
27.   <!-- NOTE IMPORTANTE : les logs ne sont pas actifs par défaut. Il faut les activer par
    programme avec l'instruction log4net.Config.XmlConfigurator.Configure();
28.   ! -->
29.   <log4net>
30.      <!-- Define an output appender (where the logs can go) -->
31.      <appender name="LogFileAppender" type="log4net.Appender.FileAppender, log4net">
32.         <param name="File" value="log.txt" />
33.         <param name="AppendToFile" value="false" />
34.         <layout type="log4net.Layout.PatternLayout, log4net">
35.            <param name="ConversionPattern" value="%d [%t] %-5p %c [%x] &lt;%X{auth}&gt; - %m%n" />
36.         </layout>
37.      </appender>
38.      <appender name="LogDebugAppender" type="log4net.Appender.DebugAppender, log4net">
39.         <layout type="log4net.Layout.PatternLayout, log4net">
40.            <param name="ConversionPattern" value="%d [%t] %-5p %c [%x] &lt;%X{auth}&gt; - %m%n"/>
41.         </layout>
42.      </appender>
43.      <appender name="ConsoleAppender" type="log4net.Appender.ConsoleAppender, log4net">
44.         <layout type="log4net.Layout.PatternLayout, log4net">
45.            <param name="ConversionPattern" value="%d [%t] %-5p %c [%x] &lt;%X{auth}&gt; - %m%n"/>
46.         </layout>
47.      </appender>
```

```
48.
49.       <!-- Setup the root category, set the default priority level and add the appender(s) (where
    the logs will go) -->
50.       <root>
51.          <priority value="INFO" />
52.          <!--
53.          <appender-ref ref="LogFileAppender" />
54.          <appender-ref ref="LogDebugAppender"/>
55.          -->
56.          <appender-ref ref="ConsoleAppender"/>
57.       </root>
58.
59.       <!-- Specify the level for some specific namespaces -->
60.       <!-- Level can be : ALL, DEBUG, INFO, WARN, ERROR, FATAL, OFF -->
61.       <logger name="NHibernate">
62.          <level value="INFO" />
63.       </logger>
64.    </log4net>
65. </configuration>
```

- lignes 4-7 : définissent des sections de configuration dans le fichier [App.config]. Considérons la ligne 6 :

```
<section name="hibernate-configuration" type="NHibernate.Cfg.ConfigurationSectionHandler, NHibernate" />
```

Cette ligne définit la section de configuration de NHibernate dans le fichier [App.config]. Elle a deux attributs : *name* et *type*.

- l'attribut [name] nomme la section de configuration. Cette section doit être ici délimitée par les balises <name>...</name>, ici <hibernate-configuration>...</hibernate-configuration> des lignes 11-24.
- l'attribut [type=classe,DLL] indique le nom de la classe chargée de traiter la section définie par l'attribut [name] ainsi que la DLL contenant cette classe. Ici, la classe s'appelle [NHibernate.Cfg.ConfigurationSectionHandler] et se trouve dans la DLL [NHibernate.dll]. On se rappelle que cette DLL fait partie des références du projet étudié.

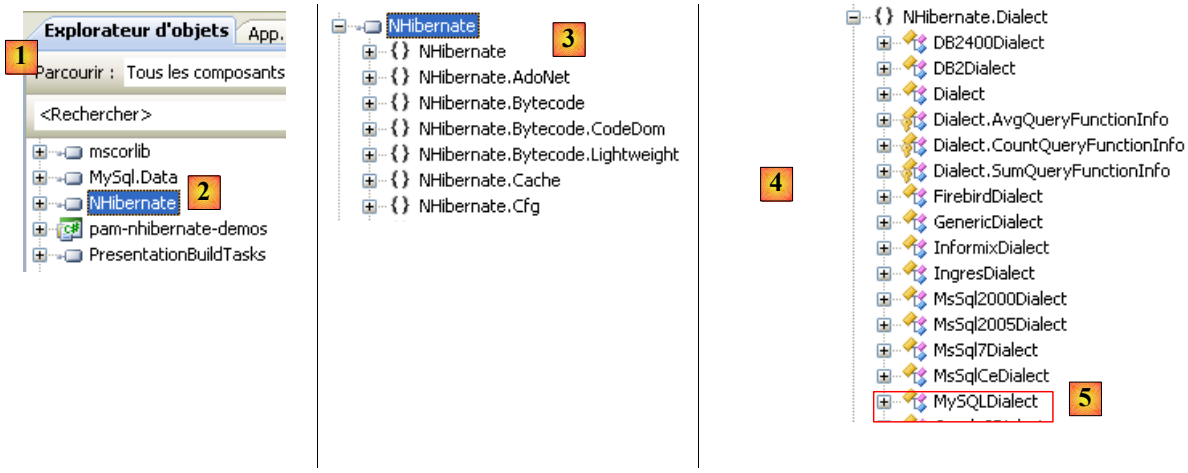
Considérons maintenant la section de configuration de NHibernate :

```
1.     <!-- configuration NHibernate -->
2.     <hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">
3.        <session-factory>
4.           <property name="connection.provider">NHibernate.Connection.DriverConnectionProvider</property>
5.           <!--
6.           <property name="connection.driver_class">NHibernate.Driver.MySqlDataDriver</property>
7.           -->
8.           <property name="dialect">NHibernate.Dialect.MySQLDialect</property>
9.           <property name="connection.connection_string">
10.             Server=localhost;Database=dbpam_nhibernate;Uid=root;Pwd=;
11.          </property>
12.          <property name="show_sql">false</property>
13.          <mapping assembly="pam-nhibernate-demos"/>
14.       </session-factory>
15. </hibernate-configuration>
```

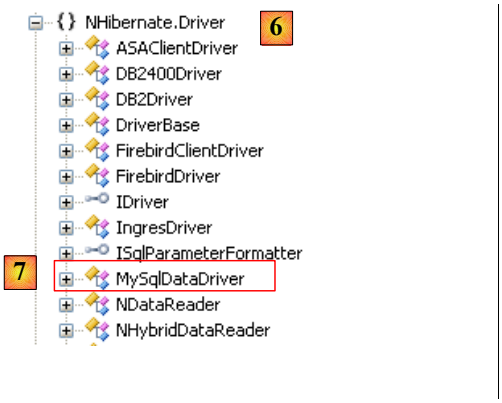
- ligne 2 : la configuration de NHibernate est à l'intérieur d'une balise <hibernate-configuration>. L'attribut xmlns (Xml NameSpace) fixe la version utilisée pour configurer NHibernate. En effet, au fil du temps, la façon de configurer NHibernate a évolué. Ici, c'est la version 2.2 qui est utilisée.
- ligne 3 : la configuration de NHibernate est ici tout entière contenue dans la balise <session-factory> (lignes 3 et 14). Une session NHibernate, est l'outil utilisé pour travailler avec une base de données selon le schéma :
  - ouverture session
  - travail avec la base de données via les méthodes de l'API NHibernate
  - fermeture session

  La session est créée par une *factory*, un terme générique désignant une classe capable de créer des objets. Les lignes 3-14 configurent cette *factory*.
- lignes 4, 6, 8, 9 : configurent la connexion à la base de données cible. Les principales informations sont le nom du SGBD utilisé, le nom de la base, l'identité de l'utilisateur et son mot de passe.
- ligne 4 : définit le fournisseur de la connexion, celui auprès duquel on demande une connexion vers la base de données. La valeur de la propriété [connection.provider] est le nom d'une classe NHibernate. Cette propriété ne dépend pas du SGBD utilisé.
- ligne 6 : le pilote ADO.NET à utiliser. C'est le nom d'une classe NHibernate spécialisée pour un SGBD donné, ici MySQL. La ligne 6 a été mise en commentaires, car elle n'est pas indispensable.
- ligne 8 : la propriété [dialect] fixe le dialecte SQL à utiliser avec le SGBD. Ici c'est le dialecte du SGBD MySQL.

Si on change de SGBD, comment trouve-t-on le dialecte NHibernate de celui-ci ? Revenons au projet C# précédent et double-cliquons sur la DLL [NHibernate] dans l'onglet [References] :



- en [1], l'onglet [Explorateur d'objets] affiche un certain nombre de DLL, dont celles référencées par le projet.
- en [2], la DLL [NHibernate]
- en [3], la DLL [NHibernate] développée. On y trouve les différents espaces de noms (namespace) qui y sont définis.
- en [4], l'espace de noms [NHibernate.Dialect] où l'on trouve les classes définissant les différents dialectes SQL utilisables.
- en [5], la classe du dialecte du SGBD MySQL.

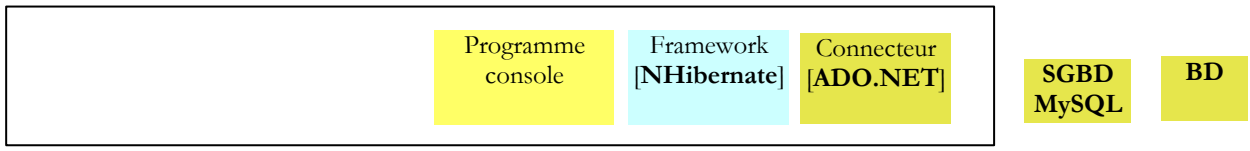


- en [6], l'espace de noms de la classe [MySqlDataDriver] utilisé ligne 6 ci-dessous :

```
1.     <!-- configuration NHibernate -->
2.     <hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">
3.         <session-factory>
4.             <property name="connection.provider">NHibernate.Connection.DriverConnectionProvider</property>
5.             <!--
6.             <property name="connection.driver_class">NHibernate.Driver.MySqlDataDriver</property>
7.             -->
8.             <property name="dialect">NHibernate.Dialect.MySQLDialect</property>
9.             <property name="connection.connection_string">
10.                Server=localhost;Database=dbpam_nhibernate;Uid=root;Pwd=;
11.            </property>
12.            <property name="show_sql">false</property>
13.            <mapping assembly="pam-nhibernate-demos"/>
14.        </session-factory>
15. </hibernate-configuration>
```

- lignes 9-11 : la chaîne de connexion à la base de données. Cette chaîne est de la forme "param1=val1;param2=val2; ...". L'ensemble des paramètres ainsi définis permet au pilote du SGBD de créer une connexion. La forme de cette chaîne de connexion est dépendante du SGBD utilisé. On trouve les chaînes de connexion aux principaux SGBD sur le site [http://www.connectionstrings.com/]. Ici, la chaîne "*Server=localhost;Database=dbpam;Uid=root;Pwd=;*" est une chaîne de connexion pour le SGBD MySQL. Elle indique que :
  - *Server=localhost;* : le SGBD est sur la même machine que le client qui cherche à ouvrir la connexion
  - *Database=dbpam;* : la base de données MySQL visée
  - *Uid=root;* : l'utilisateur qui ouvre la connexion est l'utilisateur root
  - *Pwd=;* : cet utilisateur n'a pas de mot de passe (cas particulier de cet exemple)

- ligne 12 : la propriété [show_sql] indique si NHibernate doit afficher dans ses logs, les ordres SQL qu'il émet sur la base de données. En phase de développement, il est utile de mettre cette propriété à [true] pour savoir exactement ce que fait NHibernate.
- ligne 13 : pour comprendre la balise <mapping>, revenons à l'architecture de l'application :



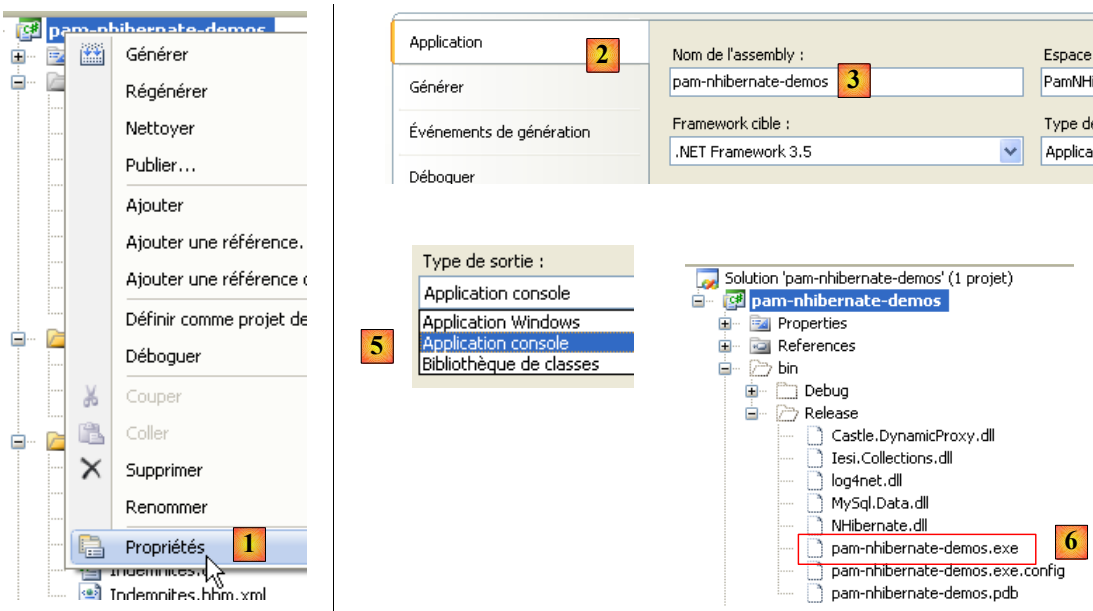
Si le programme console était un client direct du connecteur ADO.NET et qu'il voulait la liste des employés, il ferait exécuter au connecteur un ordre SQL *Select*, et il recevrait en retour un objet de type *IDataReader* qu'il aurait à traiter pour obtenir la liste des employés désirée initialement.

Ci-dessus, le programme console est le client de NHibernate et NHibernate est le client du connecteur ADO.NET. Nous verrons ultérieurement que l'API de NHibernate va permettre au programme console de demander la liste des employés. NHibernate va alors traduire cette demande en un ordre SQL *Select* qu'il va faire exécuter au connecteur ADO.NET. Celui-ci va alors lui rendre un objet de type *IDataReader* . A partir de cet objet, Nhibernate doit être capable de construire la liste des employés qui lui a été demandée. C'est par configuration que cela est rendu possible. A chaque table de la base de données est associé une classe C#. Ainsi à partir des lignes de la table [employes] renvoyées par le *IDataReader*, NHibernate va être capable de construire une liste d'objets représentant des employés et rendre celle-ci au programme console. Ces relations **tables <--> classes** sont créées dans des fichiers de configuration. NHibernate utilise le terme "mapping" pour définir ces relations.

Revenons à la ligne 13 ci-dessous :

```
1.     <!-- configuration NHibernate -->
2.     <hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">
3.         <session-factory>
4.             <property name="connection.provider">NHibernate.Connection.DriverConnectionProvider</property>
5.             <!--
6.             <property name="connection.driver_class">NHibernate.Driver.MySqlDataDriver</property>
7.             -->
8.             <property name="dialect">NHibernate.Dialect.MySQLDialect</property>
9.             <property name="connection.connection_string">
10.                Server=localhost;Database=dbpam_nhibernate;Uid=root;Pwd=;
11.            </property>
12.            <property name="show_sql">false</property>
13.            <mapping assembly="pam-nhibernate-demos"/>
14.        </session-factory>
15. </hibernate-configuration>
```

La ligne 13 indique que les fichiers de configuration **tables <--> classes** seront trouvés dans l'assembly [pam-nhibernate-demos]. Un assembly est l'exécutable ou la DLL produit par la compilation d'un projet. Ici, les fichiers de mapping seront placés dans l'assembly du projet exemple. Pour connaître le nom de cet assembly, il faut regarder les propriétés du projet :

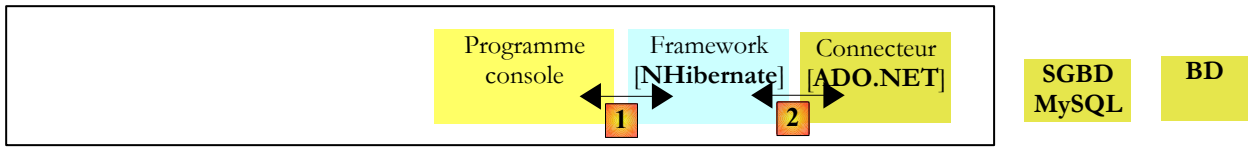


- en [1], les propriétés du projet
- dans l'onglet [Application] [2], le nom de l'assembly [3] qui va être généré.
- parce que le type de sortie est [Application console] [4], le fichier généré à la compilation du projet s'appellera [pam-nhibernate-demos.exe]. Si le type de sortie était [Bibliothèque de classes] [5], le fichier généré à la compilation du projet s'appellerait [pam-nhibernate-demos.dll]
- l'assembly est généré dans le dossier [bin/Release] du projet [6].

On retiendra de l'explication précédente que les fichiers de mapping **tables <--> classes** devront être dans le fichier [pam-nhibernate-demos.exe] [6].

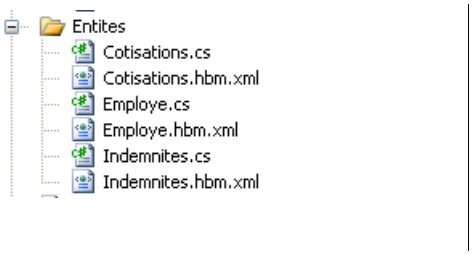
6.3.2 Configuration du mapping tables <-->classes

Revenons à l'architecture du projet étudié :



- en [1] le programme console utilise les méthodes de l'API du framework NHibernate. Ces deux blocs échangent des objets.
- en [2], NHibernate utilise l'API d'un connecteur .NET. Il émet des ordres SQL vers le SGBD cible.

Le programme console va manipuler des objets reflétant les tables de la base de données. Dans ce projet, ces objets et les liens qui les unissent aux tables de la base de données ont été placés dans le dossier [Entites] ci-dessous :

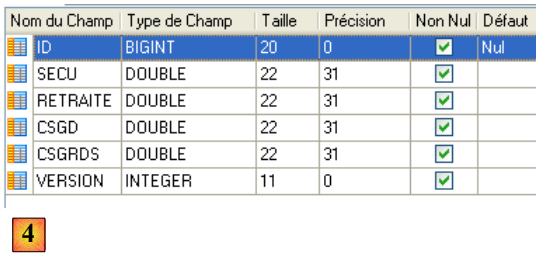


- chaque table de la base de données fait l'objet d'une classe et d'un fichier de mapping entre les deux

Table	Classe	Mapping
cotisations	Cotisations.cs	Cotisations.hbm.xml
employes	Employe.cs	Employe.hbm.xml
indemnites	Indemnites.cs	Indemnites.hbm.xml

6.3.2.1 Mapping de la table [cotisations]

Considérons la table [cotisations] :



Nom du Champ	Type de Champ	Taille	Précision	Non Nul	Défaut
ID	BIGINT	20	0	☑	Nul
SECU	DOUBLE	22	31	☑	
RETRAITE	DOUBLE	22	31	☑	
CSGD	DOUBLE	22	31	☑	
CSGRDS	DOUBLE	22	31	☑	
VERSION	INTEGER	11	0	☑	

4

ID	clé primaire de type autoincrement
VERSION	n° de version de l'enregistrement
SECU	taux (pourcentage) de cotisation pour la sécurité sociale
RETRAITE	taux de cotisation pour la retraite
CSGD	taux de cotisation pour la contribution sociale généralisée déductible
CSGRDS	taux de cotisation pour la contribution sociale généralisée et la contribution au remboursement de la dette sociale

5

Une ligne de cette table peut être encapsulée dans un objet de type [Cotisations.cs] suivant :

```
1.  namespace PamNHibernateDemos {
2.     public class Cotisations {
3.        // propriétés automatiques
4.        public virtual int Id { get; set; }
5.        public virtual int Version { get; set; }
6.        public virtual double CsgRds { get; set; }
7.        public virtual double Csgd { get; set; }
8.        public virtual double Secu { get; set; }
9.        public virtual double Retraite { get; set; }
10.
11.       // constructeurs
12.       public Cotisations() {
13.       }
14.       // ToString
15.       public override string ToString() {
16.          return string.Format("[{0}|{1}|{2}|{3}]", CsgRds, Csgd, Secu, Retraite);
17.       }
18.    }
19.
20. }
```

On a créé une propriété automatique pour chacune des colonnes de la table [cotisations]. Chacune de ces propriétés doit être déclarée virtuelle (*virtual*) car NHibernate est amené à dériver la classe et à redéfinir (override) ses propriétés. Celles-ci doivent donc être virtuelles.

On notera, ligne 1, que la classe appartient à l'espace de noms [PamNHibernateDemos].

Le fichier de mapping [Cotisations.hbm.xml] entre la table [cotisations] et la classe [Cotisations] est le suivant :

```
1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
3.  namespace="PamNHibernateDemos" assembly="pam-nhibernate-demos">
4.     <class name="Cotisations" table="COTISATIONS">
5.        <id name="Id" column="ID" unsaved-value="0">
6.           <generator class="native" />
7.        </id>
8.        <version name="Version" column="VERSION"/>
9.        <property name="CsgRds" column="CSGRDS"/>
10.       <property name="Csgd" column="CSGD"/>
11.       <property name="Retraite" column="RETRAITE"/>
12.       <property name="Secu" column="SECU"/>
13.    </class>
14. </hibernate-mapping>
```

- le fichier de mapping est un fichier Xml défini à l'intérieur de la balise <hibernate-mapping> (lignes 2 et 14)
- ligne 4 : la balise <class> fait le lien entre une table de la base de données et une classe. Ici, la table [COTISATIONS] (attribut *table*) et la classe [Cotisations] (attribut *classe*). En .NET, une classe doit être définie par son nom complet (espace de noms inclus) et par l'assembly qui la contient. Ces deux informations sont données par la ligne 3. La première (namespace) peut être trouvée dans la définition de la classe. La seconde (assembly) est le nom de l'assembly du projet. Nous avons déjà indiqué comment trouver ce nom.
- lignes 5-7 : la balise <id> sert à définir le mapping de la clé primaire de la table [cotisations].
    - ligne 5 : l'attribut *name* désigne le champ de la classe [Cotisations] qui va recevoir la clé primaire de la table [cotisations]. L'attribut *column* désigne la colonne de de la table [cotisations] qui sert de clé primaire. L'attribut *unsaved-value* sert à définir une clé primaire non encore générée. Cette valeur permet à NHibernate de savoir comment sauvegarder un objet [Cotisations] dans la table [cotisations]. Si cet objet à un champ Id=0, il fera une opération SQL INSERT, sinon il fera une opération SQL UPDATE. La valeur de *unsaved-value* dépend du type du champ *Id* de la classe [Cotisations]. Ici, il est de type *int* et la valeur par défaut d'un type *int* est 0. Un objet [Cotisations] encore non sauvegardé (sans clé primaire donc) aura donc son champ Id=0. Si le champ *Id* avait été de type *Object* ou dérivé, on aurait écrit *unsaved-value=null*.
    - ligne 6 : lorsque NHibernate doit sauvegarder un objet [Cotisations] avec un champ Id=0, il doit faire sur la base de données une opération INSERT au cours de laquelle il doit obtenir une valeur pour la clé primaire de l'enregistrement. La plupart des SGBD ont une méthode propriétaire pour générer automatiquement cette valeur. La balise <generator> sert à définir le mécanisme à utiliser pour la génération de la clé primaire. La balise <generator class="native"> indique qu'il faut utiliser le mécanisme par défaut du SGBD utilisé. Nous avons vu page 66 que les clés primaires des nos trois tables MySQL avaient l'attribut *autoincrement*. Lors de ses opérations INSERT, NHibernate ne fournira pas de valeur à la colonne ID de l'enregistrement ajouté, laissant MySQL générer cette valeur.
- ligne 8 : la balise <version> sert à définir la colonne de la table (ainsi que le champ de la classe qui va avec) qui permet de "versionner" les enregistrements. Au départ, la version vaut 1. Elle est incrémentée à chaque opération UPDATE. D'autre part, toute opération UPDATE ou DELETE est faite avec un filtre WHERE ID= id AND VERSION=v1. Un utilisateur ne peut donc modifier ou détruire un objet que s'il a la bonne version de celui-ci. Si ce n'est pas le cas, une exception est remontée par NHibernate.
- ligne 9 : la balise <property> sert à définir un mapping de colonne normale (ni clé primaire, ni colonne de version). Ainsi la ligne 9 indique que la colonne *CSGRDS* de la table [COTISATIONS] est associée à la propriété *CsgRds* de la classe [Cotisations].

6.3.2.2 Mapping de la table [indemnites]

Considérons la table [indemnites] :

Nom du Champ	Type de Cha...	Taille	Précision	Non Nul	Défaut
ID	BIGINT	20	0	☑	Nul
ENTRETIEN_JOUR	DOUBLE	22	31	☑	
REPAS_JOUR	DOUBLE	22	31	☑	
INDICE	INTEGER	11	0	☑	
INDEMNITES_CP	DOUBLE	22	31	☑	
BASE_HEURE	DOUBLE	22	31	☑	
VERSION	INTEGER	11	0	☑	

ID	clé primaire de type autoincrement
VERSION	n° de version de l'enregistrement
BASE_HEURE	coût en euro d'une heure de garde
ENTRETIEN_JOUR	indemnité en euro par jour de garde
REPAS_JOUR	indemnité de repas en euro par jour de garde
INDEMNITES_CP	indemnités de congés payés. C'est un pourcentage à appliquer au salaire de base.

Une ligne de cette table peut être encapsulée dans un objet de type [Indemnites] suivant :

```
1.  namespace PamNHibernateDemos {
2.     public class Indemnites {
3.
4.        // propriétés automatiques
5.        public virtual int Id { get; set; }
6.        public virtual int Version { get; set; }
7.        public virtual int Indice { get; set; }
8.        public virtual double BaseHeure { get; set; }
9.        public virtual double EntretienJour { get; set; }
10.       public virtual double RepasJour { get; set; }
11.       public virtual double IndemnitesCp { get; set; }
12.
13.       // constructeurs
14.       public Indemnites() {
15.       }
16.
17.       // identité
18.       public override string ToString() {
19.          return string.Format("[{0}|{1}|{2}|{3}|{4}]", Indice, BaseHeure, EntretienJour, RepasJour,
    IndemnitesCp);
20.       }
21.
22.    }
23. }
```

Le fichier de mapping table [indemnites] <--> classe [Indemnites] pourrait être le suivant (Indemnites.hbm.xml) :

```
1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
3.  namespace="PamNHibernateDemos" assembly="pam-nhibernate-demos">
4.     <class name="Indemnites" table="INDEMNITES">
5.        <id name="Id" column="ID" unsaved-value="0">
6.           <generator class="native" />
7.        </id>
8.        <version name="Version" column="VERSION"/>
9.        <property name="Indice" column="INDICE" unique="true"/>
10.       <property name="BaseHeure" column="BASE_HEURE" />
11.       <property name="EntretienJour" column="ENTRETIEN_JOUR" />
12.       <property name="RepasJour" column="REPAS_JOUR" />
13.       <property name="IndemnitesCp" column="INDEMNITES_CP" />
14.    </class>
15. </hibernate-mapping>
```

On ne trouve là rien de neuf vis à vis du fichier de mapping expliqué précédemment. La seule différence se trouve ligne 9. L'attribut unique="true" indique qu'il y a dans la table [indemnites] une contrainte d'unicité sur la colonne [INDICE] : il ne peut pas y avoir deux lignes avec la même valeur pour la colonne [INDICE].

6.3.2.3 Mapping de la table [employes]

Considérons la table [employes] :

Nom du Champ	Type de Ch...	Taille	Précision	Non Nul	Défaut
ID	BIGINT	20	0	☑	Nul
PRENOM	VARCHAR	20	0	☑	
SS	VARCHAR	15	0	☑	
ADRESSE	VARCHAR	50	0	☑	
CP	VARCHAR	5	0	☑	
VILLE	VARCHAR	30	0	☑	
NOM	VARCHAR	30	0	☑	
VERSION	INTEGER	11	0	☑	
INDEMNITE_ID	BIGINT	20	0	☑	

ID	clé primaire de type autoincrement
VERSION	n° de version de l'enregistrement
PRENOM	prénom de l'employé
NOM	son nom
ADRESSE	son adresse
CP	son code postal
VILLE	sa ville
INDEMNITE_ID	clé étrangère sur INDEMNITES(ID)

La nouveauté vis à vis des tables précédentes est la présence d'une **clé étrangère** : la colonne [INDEMNITE_ID] est une clé étrangère sur la colonne [ID] de la table [INDEMNITES]. Ce champ référence la ligne de la table [INDEMNITES] à utiliser pour calculer les indemnites de l'employé.

La classe [Employe] image de la table [employes] pourrait être la suivante :

```
1.  namespace PamNHibernateDemos {
2.     public class Employe {
3.        // propriétés automatiques
4.        public virtual int Id { get; set; }
5.        public virtual int Version { get; set; }
6.        public virtual string SS { get; set; }
7.        public virtual string Nom { get; set; }
8.        public virtual string Prenom { get; set; }
9.        public virtual string Adresse { get; set; }
10.       public virtual string Ville { get; set; }
11.       public virtual string CodePostal { get; set; }
12.       public virtual Indemnites Indemnites { get; set; }
13.
14.       // constructeurs
15.       public Employe() {
16.       }
17.
18.       // ToString
19.       public override string ToString() {
20.          return string.Format("[{0}|{1}|{2}|{3}|{4}|{5}|{6}]", SS, Nom, Prenom, Adresse, Ville,
    CodePostal, Indemnites);
21.       }
22.    }
23. }
```

Le fichier de mapping [Employe.hbm.xml] pourrait être le suivant :

```
1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
3.  namespace="PamNHibernateDemos" assembly="pam-nhibernate-demos">
4.     <class name="Employe" table="EMPLOYES">
5.        <id name="Id" column="ID" unsaved-value="0">
6.           <generator class="native" />
7.        </id>
8.        <version name="Version" column="VERSION"/>
9.        <property name="SS" column="SS"/>
10.       <property name="Nom" column="NOM"/>
11.       <property name="Prenom" column="PRENOM"/>
12.       <property name="Adresse" column="ADRESSE"/>
13.       <property name="Ville" column="VILLE"/>
14.       <property name="CodePostal" column="CP"/>
15.       <many-to-one name="Indemnites" column="INDEMNITE_ID" cascade="save-update"/>
16.    </class>
17. </hibernate-mapping>
```

La nouveauté réside ligne 15 avec l'apparition d'une nouvelle balise : <many-to-one>. Cette balise sert à mapper une colonne clé étrangère [INDEMNITE_ID] de la table [EMPLOYES] vers la propriété [Indemnites] de la classe [Employe] :

```
1.  namespace PamNHibernateDemos {
2.     public class Employe {
3.        // propriétés automatiques
4.  ..
5.        public virtual Indemnites Indemnites { get; set; }
6.
7.  ...
8.     }
9.  }
```

La table [EMPLOYES] a une clé étrangère [INDEMNITE_ID] qui référence la colonne [ID] de la table [INDEMNITES]. Plusieurs (many) lignes de la table [EMPLOYES] peuvent référencer une même ligne (one) de la table [INDEMNITES]. D'où le nom de la balise <many-to-one>. Cette balise a ici les attributs suivants :

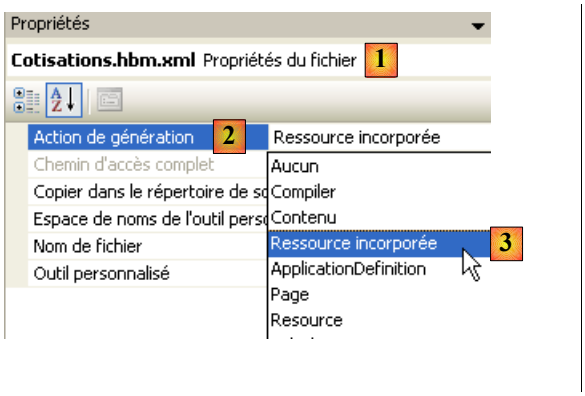
- **column** : indique le nom de la colonne de la table [EMPLOYES] qui est clé étrangère sur la table [INDEMNITES]
- **name** : indique la propriété de la classe [Employe] associée à cette colonne. Le type de cette propriété est nécessairement la classe associée à la table cible de la clé étrangère, ici la table [INDEMNITES]. On sait que cette classe est la classe [Indemnites] déjà décrite. C'est ce que reflète la ligne 5 ci-dessus. Cela signifie que lorsque NHibernate ramènera de la base un objet [Employe], il ramènera également l'objet [Indemnites] qui va avec.
- **cascade** : cet attribut peut avoir diverses valeurs :
  - *save-update* : une opération d'insertion (save) ou de mise à jour (update) sur l'objet [Employe] doit être propagée sur l'objet [Indemnites] qu'il contient.

- *delete* : la suppression d'un objet [Employe] doit être propagée à l'objet [Indemnites] qu'il contient.
- *all* : propage les opérations d'insertion (save), de mise à jour (update) et de suppression (delete).
- *none* : ne propage rien

Pour terminer, rappelons la configuration de NHibernate dans le fichier [App.config] :

```
1.     <!-- configuration NHibernate -->
2.     <hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">
3.         <session-factory>
4.             <property name="connection.provider">NHibernate.Connection.DriverConnectionProvider</property>
5.             <!--
6.             <property name="connection.driver_class">NHibernate.Driver.MySqlDataDriver</property>
7.             -->
8.             <property name="dialect">NHibernate.Dialect.MySQLDialect</property>
9.             <property name="connection.connection_string">
10.                Server=localhost;Database=dbpam_nhibernate;Uid=root;Pwd=;
11.            </property>
12.            <property name="show_sql">false</property>
13.            <mapping assembly="pam-nhibernate-demos"/>
14.        </session-factory>
15. </hibernate-configuration>
```

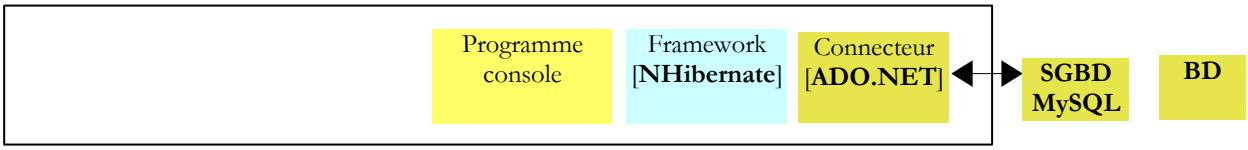
La ligne 13 indique que les fichiers de mapping ***.hbm.xml** seront trouvés dans l'assembly [pam-nhibernate-demos]. Ceci n'est pas fait par défaut. Il faut le configurer dans le projet C# :



- en [1], on sélectionne les propriétés d'un fichier de mapping
- en [2], l'action de génération doit être [Ressource incorporée] [3]. Cela signifie qu'à la génération du projet, le fichier de mapping doit être incorporé dans l'assembly généré.

6.4 l'API de NHibernate

Revenons à l'architecture de notre projet exemple :



Dans les paragraphes précédents, nous avons configuré NHibernate de deux façons :

- dans [App.config], nous avons configuré la connexion à la base de données
- nous avons écrit pour chaque table de la base, la classe image de cette table et le fichier de mapping qui permet de passer de la classe à la table et vice-versa.

Il nous reste à découvrir les méthodes offertes par NHibernate pour manipuler les données de la base : insertion, mise à jour, suppression, liste.

6.4.1 L'objet SessionFactory

Toute opération NHibernate se fait à l'intérieur d'une session. Une séquence typique d'opérations NHibernate est la suivante :

- ouvrir une session NHibernate
- commencer une transaction dans la session
- faire des opérations de persistance avec la session (Load, Get, Find, CreateQuery, Save, SaveOrUpdate, Delete)
- valider (commit) ou invalider (rollback) la transaction
- fermer la session NHibernate

Une session est obtenue auprès d'une factory de type [SessionFactory]. Cette factory est celle configurée par la balise <session-factory> dans le fichier de configuration [App.config] :

```
1.     <!-- configuration NHibernate -->
2.     <hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">
3.        <session-factory>
4.           <property name="connection.provider">NHibernate.Connection.DriverConnectionProvider</property>
5.           <!--
6.           <property name="connection.driver_class">NHibernate.Driver.MySqlDataDriver</property>
7.           -->
8.           <property name="dialect">NHibernate.Dialect.MySQLDialect</property>
9.           <property name="connection.connection_string">
10.             Server=localhost;Database=dbpam_nhibernate;Uid=root;Pwd=;
11.          </property>
12.          <property name="show_sql">false</property>
13.          <mapping assembly="pam-nhibernate-demos"/>
14.       </session-factory>
15. </hibernate-configuration>
```

Dans un code C#, la SessionFactory peut être obtenue de la façon suivante :

```
ISessionFactory sessionFactory = new Configuration().Configure().BuildSessionFactory();
```

La classe **Configuration** est une classe du framework NHibernate. L'instruction précédente exploite la section de configuration de NHibernate dans [App.config]. L'objet [ISessionFactory] obtenu a alors les :

- informations pour créer une connexion à la base de données cible
- les fichiers de mapping entre tables de la base de données et classes persistantes manipulées par NHibernate.

6.4.2 La session NHibernate

Une fois la *SessionFactory* créée (cela se fait une unique fois), on peut en obtenir les sessions permettant de faire des opérations de persistance NHibernate. Un code usuel est le suivant :

```
1.  try{
2.        // ouverture session
3.        using (ISession session = sessionFactory.OpenSession())
4.        {
5.          // début transaction
6.          using (ITransaction transaction = session.BeginTransaction())
7.          {
8.  ... opérations de persistance
9.            // validation de la transaction
10.           transaction.Commit();
11.         }
12.       }
13. }catch (Exception ex){
14. ....
15. }
```

- ligne 3 : une session est créée à partir de la *SessionFactory* à l'intérieur d'une clause **using**. A la sortie de la clause *using*, la session sera automatiquement fermée. Sans la clause *using*, il faudrait fermer la session explicitement (*session.Close()*).
- ligne 6 : les opérations de persistance vont se faire à l'intérieur d'une transaction. Soit elles réussissent toutes, soit aucune ne réussit. A l'intérieur de la clause *using*, la transaction est validée par un *Commit* (ligne 10). Si dans la transaction, une opération de persistance lance une exception, la transaction sera automatiquement invalidée par un *Rollback* à la sortie du *using*.
- le try / catch des lignes 1 et 13 permet d'intercepter une éventuelle exception lancée par le code à l'intérieur du try (session, transaction, persistance).

6.4.3 L'interface ISession

Nous présentons maintenant certaines des méthodes de l'interface **ISession** implémentée par une session NHibernate :

`ITransaction BeginTransaction()`	démarre une transaction dans la session *ITransaction tx=session.BeginTransaction();*
`void Clear()`	vide la session. Les objets qu'elle contenait deviennent détachés. *session.Clear();*
`void Close()`	ferme la session. Les objets qu'elle contenait sont synchronisés avec la base de données. Cette opération de synchronisation est également faite à la fin d'une transaction. Ce dernier cas est le plus courant. *session.Close();*
`IQuery CreateQuery(string queryString)`	crée une requête HQL (Hibernate Query Language) pour une exécution ultérieure. *IQuery query=session.createQuery("select e from Employe e);*
`void Delete(object id)`	supprime un objet. Celui-ci peut appartenir à la session (attaché) ou non (détaché). Le paramètre de la méthode est la clé primaire de l'objet à détruire. Lors de la synchronisation de la session avec la base de données, une opération SQL DELETE sera faite sur cet objet. *// on charge un employé de la BD* *Employe e=session.Get<Employe>(143);* *// on le supprime* *session.Delete(e);*
`void Flush()`	force la synchronisation de la session avec la base de données. Le contenu de la session ne change pas. *session.Flush();*
`T Get<T>(object id)`	va chercher dans la base l'objet **T** de clé primaire **id**. Si cet objet n'existe pas, rend le pointeur **null**. *// on charge un employé de la BD* *Employe e=session.Get<Employe>(143);*
`object Save(object obj)`	met l'objet **obj** dans la session. Cet objet n'a pas de clé primaire avant le *Save*. Après le *Save*, il en a une. Lors de la synchronisation de la session, une opération SQL INSERT sera faite sur la base. *// on crée un employé* *Employe e=new Employe(){...};* *// on le sauvegarde* *e=session.Save(e);*
`SaveOrUpdate(object obj)`	fait une opération *Save* si **obj** n'a pas de clé primaire ou une opération *Update* s'il en a déjà une.
`void Update(object obj)`	met à jour dans la base de données, l'objet **obj**. Une opération SQL UPDATE est alors faite sur la base. *// on charge un employé de la BD* *Employe e=session.Get<Employe>(143);* *// on change son nom*

```
e.Nom=...;
// on le met à jour dans la base
Update(e);
```

6.4.4 L'interface IQuery

L'interface **IQuery** permet de requêter la base de données pour en extraire des données. Nous avons vu comment en créer une instance :

```
IQuery query=session.createQuery("select e from Employe e);
```

Le paramètre de la méthode *createQuery* est une requête HQL (Hibernate Query Language), un langage analogue au langage SQL mais requêtant des classes plutôt que des tables. La requête ci-dessus demande la liste de tous les employés. Voici quelques exemples de requêtes HQL :

```
select e from Employe e where e.Nom like 'A%'
select e from Employe order by e.Nom asc
select e from Employe e where e.Indemnites.Indice=2
```

Nous présentons maintenant certaines des méthodes de l'interface **IQuery** :

`IList<T> List<T>()` rend le résultat de la requête sous la forme d'une liste d'objets T

```
IList<Employe> employes=session.createQuery("select e from Employe e order by e.Nom asc").List<Employe>();
```

`IList List()` rend le résultat de la requête sous la forme d'une liste où chaque élément de la liste représente une ligne résultat du Select sous la forme d'un tableau d'objets.

```
IList lignes=session.createQuery("select e.Nom, e.Prenom, e.SS").List();
```

lignes[i][j] représente la colonne j de la ligne i dans un type **object**. Ainsi lignes[10][1] est un type **object** représentant le prénom d'une personne. Des transtypages sont en général nécessaires pour récupérer les données dans leur type exact.

`T UniqueResult<T>()` rend le premier objet du résultat de la requête

```
Employe e=session.createQuery("select e from Employe e where e.Nom='MARTIN'").UniqueResult<Employe>();
```

Une requête HQL peut être **paramétrée** :

```
1.  string numSecu;
2.  ...
3.  Employe e=session.createQuery("select e from Employe e where
    e.SS=:num").SetString("num",numSecu).UniqueResult<Employe>();
```

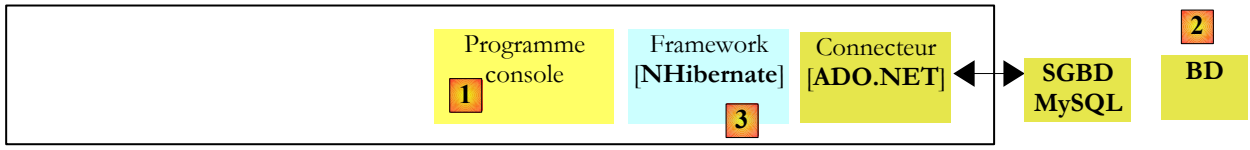
Dans la requête HQL de la ligne 3, *:num* est un paramètre qui doit recevoir une valeur avant que la requête ne soit exécutée. Ci-dessus, c'est la méthode *SetString* qui est utilisée pour cela. L'interface *IQuery* dispose de diverses méthodes **Set** pour affecter une valeur à un paramètre :

- **SetBoolean**(string name, bool value)
- **SetSingle**(string name, single value)
- **SetDouble**(string name, double value)
- **SetInt32**(string name, int32 value)

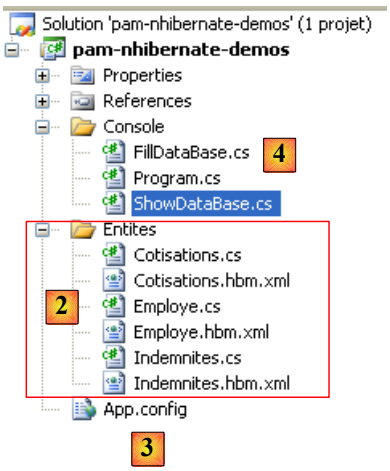
..

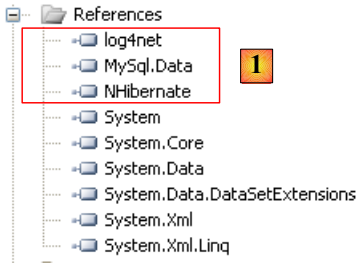
6.5 Quelques exemples de code

Les exemples qui suivent s'appuient sur l'architecture étudiée précédemment et rappelée ci-dessous. La base de données est la base de donnée MySQL [dbpam_nhibernate] également présentée. Les exemples sont des programmes console [1] utilisant le framework NHibernate [3] pour manipuler la base de données [2].



Le projet C# dans lequel s'insèrent les exemples qui vont suivre est celui déjà présenté :





- en [1], les DLL dont a besoin le projet :
  - [NHibernate] : la DLL du framework NHibernate
  - [MySql.Data] : la DLL du connecteur ADO.NET du SGBD MySQL
  - [log4net] : la DLL d'un outil permettant de générer des logs
- en [2], les classes images des tables de la base de données
- en [3], le fichier [App.config] qui configure l'application tout entière, dont le framework [NHibernate]
- en [4], des applications console de test. Ce sont celles-ci que nous allons présenter partiellement.

6.5.1 Obtenir le contenu de la base

Le programme [ShowDataBase.cs] permet d'afficher le contenu de la base :

```
1.  using System;
2.  using System.Collections;
3.  using System.Collections.Generic;
4.  using NHibernate;
5.  using NHibernate.Cfg;
6.
7.
8.  namespace PamNHibernateDemos
9.  {
10.   public class ShowDataBase
11.   {
12.
13.     private static ISessionFactory sessionFactory = null;
14.
15.     // programme principal
16.     static void Main(string[] args)
17.     {
18.       // initialisation factory NHibernate
19.       sessionFactory = new Configuration().Configure().BuildSessionFactory();
20.       try
21.       {
22.         // affichage contenu de la base
23.         Console.WriteLine("Affichage base ------------------------------------");
24.         ShowDataBase1();
25.       }
26.       catch (Exception ex)
```

```
27.       {
28.         // on affiche l'exception
29.         Console.WriteLine(string.Format("L'erreur suivante s'est produite : [{0}]",
    ex.ToString()));
30.       }
31.       finally
32.       {
33.         if (sessionFactory != null)
34.         {
35.           sessionFactory.Close();
36.         }
37.       }
38.       // attente clavier
39.       Console.ReadLine();
40.     }
41.
42.     // test1
43.     static void ShowDataBase1()
44.     {
45.       // ouverture session
46.       using (ISession session = sessionFactory.OpenSession())
47.       {
48.         // début transaction
49.         using (ITransaction transaction = session.BeginTransaction())
50.         {
51.           // on récupère la liste des employés
52.           IList<Employe> employes = session.CreateQuery(@"select e from Employe e order by e.Nom
    asc").List<Employe>();
53.           // on l'affiche
54.           Console.WriteLine("--------------- liste des employés");
55.           foreach (Employe e in employes)
56.           {
57.             Console.WriteLine(e);
58.           }
59.           // on récupère la liste des indemnites
60.           IList<Indemnites> indemnites = session.CreateQuery(@"select i from Indemnites i order by
    i.Indice asc").List<Indemnites>();
61.           // on l'affiche
62.           Console.WriteLine("--------------- liste des indemnités");
63.           foreach (Indemnites i in indemnites)
64.           {
65.             Console.WriteLine(i);
66.           }
67.           // on récupère la liste des cotisations
68.           Cotisations cotisations = session.CreateQuery(@"select c from Cotisations
    c").UniqueResult<Cotisations>();
69.           Console.WriteLine("--------------- tableau des taux de cotisations");
70.           Console.WriteLine(cotisations);
71.           // commit transaction
72.           transaction.Commit();
73.         }
74.       }
75.     }
76.   }
77. }
```

Question : expliquer ce code

Affichage écran obtenu :

```
1.  Affichage base -------------------------------------
2.  --------------- liste des employés
3.  [254104940426058|Jouveinal|Marie|5 rue des oiseaux|St Corentin|49203|[1|1,93|2|3|12]]
4.  [260124402111742|Laverti|Justine|La Brûlerie|St Marcel|49014|[2|2,1|2,1|3,1|15]]
5.
6.  --------------- liste des indemnités
7.  [1|1,93|2|3|12]
8.  [2|2,1|2,1|3,1|15]
9.  --------------- tableau des taux de cotisations
10. [3,49|6,15|9,39|7,88]
```

On notera lignes 3 et 4 qu'en demandant un employé, on a également obtenu son indemnité.

6.5.2 Insérer des données dans la base

Le programme [FillDataBase.cs] permet d'insérer des données dans la base :

```
1.  using System;
2.  using System.Collections;
3.  using System.Collections.Generic;
4.  using NHibernate;
5.  using NHibernate.Cfg;
6.
7.
8.  namespace PamNHibernateDemos
9.  {
10.   public class FillDataBase
11.   {
12.
13.     private static ISessionFactory sessionFactory = null;
14.
15.     // programme principal
16.     static void Main(string[] args)
17.     {
18.       // initialisation factory NHibernate
19.       sessionFactory = new Configuration().Configure().BuildSessionFactory();
20.       try
21.       {
22.         // suppression du contenu de la base
23.         Console.WriteLine("Effacement base -----------------------------------");
24.         ClearDataBase1();
25.         Console.WriteLine("Affichage base -----------------------------------");
26.         ShowDataBase();
27.         Console.WriteLine("Remplissage base -----------------------------------");
28.         FillDataBase1();
29.         Console.WriteLine("Affichage base -----------------------------------");
30.         ShowDataBase();
31.       }
32.       catch (Exception ex)
33.       {
34.         // on affiche l'exception
35.         Console.WriteLine(string.Format("L'erreur suivante s'est produite : [{0}]",
    ex.ToString()));
36.       }
37.       finally
38.       {
39.         if (sessionFactory != null)
40.         {
41.           sessionFactory.Close();
42.         }
43.       }
44.       // attente clavier
45.       Console.ReadLine();
46.     }
47.
48.     // test1
49.     static void ShowDataBase()
50.     {
51.         // voir exemple précédent
52.     }
53.
54.     // ClearDataBase1
55.     static void ClearDataBase1()
56.     {
57.       // ouverture session
58.       using (ISession session = sessionFactory.OpenSession())
59.       {
60.         // début transaction
61.         using (ITransaction transaction = session.BeginTransaction())
62.         {
63.           // on récupère la liste des employés
64.           IList<Employe> employes = session.CreateQuery(@"select e from Employe
    e").List<Employe>();
65.           // on supprime tous les employés
66.           Console.WriteLine("--------------- suppression des employés associées");
67.           foreach (Employe e in employes)
68.           {
69.             session.Delete(e);
70.           }
71.           // on récupère la liste des indemnités
72.           IList<Indemnites> indemnites = session.CreateQuery(@"select i from Indemnites
    i").List<Indemnites>();
73.           // on supprime les indemnités
```

```
74.           Console.WriteLine("--------------- suppression des indemnités");
75.           foreach (Indemnites i in indemnites)
76.           {
77.             session.Delete(i);
78.           }
79.           // on récupère la liste des cotisations
80.           Cotisations cotisations = session.CreateQuery(@"select c from Cotisations
    c").UniqueResult<Cotisations>();
81.           Console.WriteLine("--------------- suppression des taux de cotisations");
82.           if (cotisations != null)
83.           {
84.             session.Delete(cotisations);
85.           }
86.           // commit transaction
87.           transaction.Commit();
88.         }
89.       }
90.     }
91.
92.     // FillDataBase
93.     static void FillDataBase1()
94.     {
95.       // ouverture session
96.       using (ISession session = sessionFactory.OpenSession())
97.       {
98.         // début transaction
99.         using (ITransaction transaction = session.BeginTransaction())
100.          {
101.            // on crée deux indemnités
102.            Indemnites i1 = new Indemnites() { Id = 0, Indice = 1, BaseHeure = 1.93,
    EntretienJour = 2, RepasJour = 3, IndemnitesCp = 12 };
103.            Indemnites i2 = new Indemnites() { Id = 0, Indice = 2, BaseHeure = 2.1, EntretienJour
    = 2.1, RepasJour = 3.1, IndemnitesCp = 15 };
104.            // on crée deux employés
105.            Employe e1 = new Employe() { Id = 0, SS = "254104940426058", Nom = "Jouveinal",
    Prenom = "Marie", Adresse = "5 rue des oiseaux", Ville = "St Corentin", CodePostal = "49203",
    Indemnites = i1 };
106.            Employe e2 = new Employe() { Id = 0, SS = "260124402111742", Nom = "Laverti", Prenom
    = "Justine", Adresse = "La Brûlerie", Ville = "St Marcel", CodePostal = "49014", Indemnites =
    i2 };
107.            // on crée les taux de cotisations
108.            Cotisations cotisations = new Cotisations() { Id = 0, CsgRds = 3.49, Csgd = 6.15,
    Secu = 9.39, Retraite = 7.88 };
109.            // on sauvegarde le tout
110.            session.Save(e1);
111.            session.Save(e2);
112.            session.Save(cotisations);
113.            // commit transaction
114.            transaction.Commit();
115.          }
116.        }
117.      }
118.
119.    }
120.  }
```

Question : expliquer ce code

Affichage écran obtenu :

```
Effacement base -----------------------------------
--------------- suppression des employés et des indemnités associées
--------------- suppression des indemnités restantes
--------------- suppression des taux de cotisations
Affichage base ------------------------------------
--------------- liste des employés
--------------- liste des indemnités
--------------- tableau des taux de cotisations

Remplissage base ----------------------------------
Affichage base ------------------------------------
--------------- liste des employés
[254104940426058|Jouveinal|Marie|5 rue des oiseaux|St Corentin|49203|[2|2,1|2,1|3,1|15]]
[260124402111742|Laverti|Justine|La Brûlerie|St Marcel|49014|[1|1,93|2|3|12]]
--------------- liste des indemnités
[1|1,93|2|3|12]
[2|2,1|2,1|3,1|15]
```

```
--------------- tableau des taux de cotisations
[3,49|6,15|9,39|7,88]
```

6.5.3 Recherche d'un employé

Le programme [Program.cs] a diverses méthodes illustrant l'accès et la manipulation des données de la base. Nous en présentons quelques-unes.

La méthode [FindEmployee] permet de trouver un employé d'après son n° de sécurité sociale :

```
1.  // FindEmployee
2.      static void FindEmployee()
3.      {
4.        // ouverture session
5.        using (ISession session = sessionFactory.OpenSession())
6.        {
7.          // début transaction
8.          try
9.          {
10.           using (ITransaction transaction = session.BeginTransaction())
11.           {
12.             // on recherche un employé à partir de son n° SS
13.             String numSecu = "254104940426058";
14.             IQuery query=session.CreateQuery(@"select e from Employe e where e.SS=:numSecu");
15.             Employe employe = query.SetString("numSecu", numSecu).UniqueResult<Employe>();
16.             if (employe != null)
17.             {
18.               Console.WriteLine("Employe[" + numSecu + "]=" + employe);
19.             }
20.             else
21.             {
22.               Console.WriteLine("Employe[" + numSecu + "] non trouvé...");
23.             }
24.             // on recherche un employé inexistant
25.             numSecu = "xx";
26.             employe = query.SetString("numSecu", numSecu).UniqueResult<Employe>();
27.             if (employe != null)
28.             {
29.               Console.WriteLine("Employe[" + numSecu + "]=" + employe);
30.             }
31.             else
32.             {
33.               Console.WriteLine("Employe[" + numSecu + "] non trouvé...");
34.             }
35.
36.             // commit transaction
37.             transaction.Commit();
38.           }
39.         }
40.         catch (Exception e)
41.         {
42.           Console.WriteLine("L'exception suivante s'est produite : " + e.Message);
43.         }
44.       }
45.     }
```

Affichage écran obtenu :

```
Recherche d'un employé -------------------------------------
Employe[254104940426058]=[254104940426058|Jouveinal|Marie|5 rue des oiseaux|St Corentin|49203|[2|2,1|2,1|
3,1|15]]
Employe[xx] non trouvé...
```

6.5.4 Insertion d'entités invalides

La méthode suivante tente de sauvegarder une entité [Employe] non initialisée.

```
1.  // SaveEmptyEmployee
2.       static void SaveEmptyEmployee() {
3.          // ouverture session
4.          using (ISession session = sessionFactory.OpenSession()) {
```

```
5.             // début transaction
6.             try {
7.                using (ITransaction transaction = session.BeginTransaction()) {
8.                   // on crée un employe vide
9.                   Employe e = new Employe();
10.                  // on crée une indemnité encore non existante
11.                  Indemnites i = new Indemnites() { Id = 0, Indice = 3, BaseHeure = 1.93,
    EntretienJour = 2, RepasJour = 3, IndemnitesCp = 12 };
12.                  // qu'on associe à l'employé
13.                  e.Indemnites = i;
14.                  // on sauvegarde l'employé en laissant vides les autres champs
15.                  session.Save(e);
16.                  // commit transaction
17.                  transaction.Commit();
18.               }
19.            } catch (Exception e) {
20.               Console.WriteLine("L'exception suivante s'est produite : " + e.Message);
21.            }
22.         }
23.      }
```

Rappelons le code de la classe [Employe] :

```
1.  namespace PamNHibernateDemos {
2.    public class Employe {
3.       // propriétés automatiques
4.       public virtual int Id { get; set; }
5.       public virtual int Version { get; set; }
6.       public virtual string SS { get; set; }
7.       public virtual string Nom { get; set; }
8.       public virtual string Prenom { get; set; }
9.       public virtual string Adresse { get; set; }
10.      public virtual string Ville { get; set; }
11.      public virtual string CodePostal { get; set; }
12.      public virtual Indemnites Indemnites { get; set; }
13.
14.      // constructeurs
15.      public Employe() {
16.      }
17.
18.      // ToString
19.      public override string ToString() {
20.         return string.Format("[{0}|{1}|{2}|{3}|{4}|{5}|{6}]", SS, Nom, Prenom, Adresse, Ville,
    CodePostal, Indemnites);
21.      }
22.   }
23. }
```

Un objet [Employe] non initialisé, aura la valeur **null** pour tous ses champs de type **string**. Lors de l'insertion de l'enregistrement dans la table [employes], NHibernate laissera vides les colonnes correspondant à ces champs. Or dans la table [employes], toutes les colonnes ont l'attribut **not null**, ce qui interdit les colonnes sans valeur. Le pilote ADO.NET lancera alors une exception :

```
1.  sauvegarde d'un employé vide -------------------------------------
2.  L'exception suivante s'est produite : could not insert: [PamNHibernateDemos.Employe][SQL: INSERT
    INTO EMPLOYES (VERSION, SS, NOM, PRENOM, ADRESSE, VILLE, CP, INDEMNITE_ID) VALUES
    (?, ?, ?, ?, ?, ?, ?, ?)]
```

6.5.5 Création de deux indemnités de même indice à l'intérieur d'une transaction

Dans la table [indemnites], la colonne [indice] a été déclarée avec l'attribut **unique**, ce qui interdit d'avoir deux lignes avec le même indice. La méthode suivante crée deux indemnités de même indice à l'intérieur d'une transaction :

```
1.  // CreateIndemnites1
2.       static void CreateIndemnites1()
3.       {
4.         // ouverture session
5.         using (ISession session = sessionFactory.OpenSession())
6.         {
7.           // début transaction
8.           try
9.           {
10.            using (ITransaction transaction = session.BeginTransaction())
11.            {
```

```
12.             // on crée deux indemnités de même indice
13.             Indemnites i1 = new Indemnites() { Id = 0, Indice = 1, BaseHeure = 1.93, EntretienJour
    = 2, RepasJour = 3, IndemnitesCp = 12 };
14.             Indemnites i2 = new Indemnites() { Id = 0, Indice = 1, BaseHeure = 1.93, EntretienJour
    = 2, RepasJour = 3, IndemnitesCp = 12 };
15.             // on les sauvegarde
16.             session.Save(i1);
17.             session.Save(i2);
18.             // commit transaction
19.             transaction.Commit();
20.           }
21.         }
22.         catch (Exception e)
23.         {
24.           Console.WriteLine("L'exception suivante s'est produite : " + e.Message);
25.         }
26.       }
27.     }
```

Le résultat obtenu est le suivant :

```
1.  Effacement base -----------------------------------
2.  --------------- suppression des employés
3.  --------------- suppression des indemnités
4.  --------------- suppression des taux de cotisations
5.  Création de deux indemnités de même indice dans une transaction --------------
6.  L'exception suivante s'est produite : could not insert: [PamNHibernateDemos.Indemnites][SQL:
    INSERT INTO INDEMNITES (VERSION, INDICE, BASE_HEURE, ENTRETIEN_JOUR, REPAS_JOUR, INDEMNITES_CP)
    VALUES (?, ?, ?, ?, ?, ?)]
7.  Affichage base -----------------------------------
8.  --------------- liste des employés
9.  --------------- liste des indemnités
10. --------------- tableau des taux de cotisations
```

Ligne 9, on peut voir que la table [indemnites] est vide.

Question : expliquez ce qui s'est passé

6.5.6 Création de deux indemnités de même indice hors transaction

La méthode suivante crée deux indemnités de même indice sans utiliser de transaction :

```
1.  // CreateIndemnites2
2.      static void CreateIndemnites2()
3.      {
4.        // ouverture session
5.        using (ISession session = sessionFactory.OpenSession())
6.        {
7.          try
8.          {
9.            // on crée deux indemnités de même indice
10.           Indemnites i1 = new Indemnites() { Id = 0, Indice = 1, BaseHeure = 1.93, EntretienJour =
    2, RepasJour = 3, IndemnitesCp = 12 };
11.           Indemnites i2 = new Indemnites() { Id = 0, Indice = 1, BaseHeure = 1.94, EntretienJour =
    2, RepasJour = 3, IndemnitesCp = 12 };
12.           // on les sauvegarde
13.           session.Save(i1);
14.           session.Save(i2);
15.         }
16.         catch (Exception e)
17.         {
18.           Console.WriteLine("L'exception suivante s'est produite : " + e.Message);
19.         }
20.       }
21.     }
```

Le résultat obtenu est le suivant :

```
1.  Création de deux indemnités de même indice sans transaction --------------
2.  L'exception suivante s'est produite : could not insert: [PamNHibernateDemos.Indemnites][SQL:
    INSERT INTO INDEMNITES (VERSION, INDICE, BASE_HEURE, ENTRETIEN_JOUR, REPAS_JOUR, INDEMNITES_CP)
    VALUES (?, ?, ?, ?, ?, ?)]
```

```
3.  Affichage base ------------------------------------
4.  --------------- liste des employés
5.  --------------- liste des indemnités
6.  [1|1,93|2|3|12]
7.  --------------- tableau des taux de cotisations
```

La base était vide avant l'exécution de la méthode. Ligne 6, on peut voir que la table [indemnites] a une ligne.

Question : expliquez ce qui s'est passé

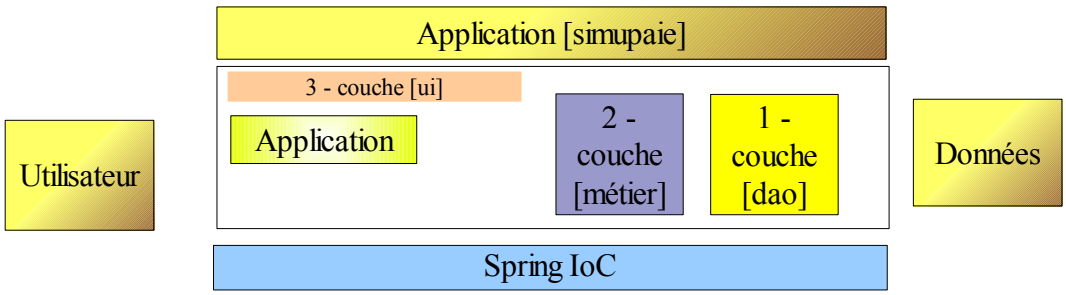
---

7 L'application [SimuPaie] – version 3 – architecture 3 couches avec NHibernate

Lectures conseillées : **"Langage C# 2008, Chapitre 4 : Architectures 3 couches, tests NUnit, framework Spring"**.

7.1 Architecture générale de l'application

L'application [SimuPaie] aura maintenant la structure à trois couches suivante :

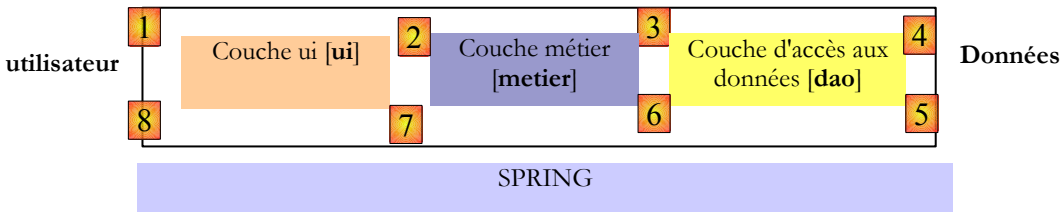


- la couche [1-dao] (dao=Data Access Object) s'occupera de l'accès aux données.
- la couche [2-métier] s'occupera de l'aspect métier de l'application, le calcul de la paie.
- la couche [3-ui] (ui=User Interface) s'occupera de la présentation des données à l'utilisateur et de l'exécution de ses requêtes. Nous appelons [Application] l'ensemble des modules assurant cette fonction. Elle est l'interlocuteur de l'utilisateur.
- les trois couches seront rendues indépendantes grâce à l'utilisation d'interfaces .NET
- l'intégration des différentes couches sera réalisée par **Spring IoC**

Le traitement d'une demande d'un client se déroule selon les étapes suivantes :

1. le client fait une demande à l'application.
2. l'application traite cette demande. Pour ce faire, elle peut avoir besoin de l'aide de la couche [métier] qui elle-même peut avoir besoin de la couche [dao] si des données doivent être échangées avec la base de données.
3. l'application reçoit une réponse de la couche [métier]. Selon celle-ci, elle envoie la vue (= la réponse) appropriée au client.

Prenons l'exemple du calcul de la paie d'une assistante maternelle. Celui-ci va nécessiter plusieurs étapes :



(a) la couche [ui] va devoir demander à l'utilisateur
- l'identité de la personne dont on veut faire la paie
- le nombre de jours travaillés de celle-ci
- le nombre d'heures travaillées

(b) Pour cela elle va devoir présenter à celui-ci la liste des personnes (nom, prénom, SS) présentes dans la table [EMPLOYES] afin que l'utilisateur choisisse l'une d'elles. La couche [ui] va utiliser le chemin [2, 3, 4, 5, 6, 7] pour les obtenir. L'opération [2] est la demande de la liste des employés, l'opération [7] la réponse à cette demande. Ceci fait, la couche [ui] peut présenter la liste des employés à l'utilisateur par [8].

(c) l'utilisateur va transmettre à la couche [ui] le nombre de jours travaillés ainsi que le nombre d'heures travaillées. C'est l'opération [1] ci-dessus. Au cours de cette étape, l'utilisateur n'interagit qu'avec la couche [ui]. C'est celle-ci qui va notamment vérifier la validité des données saisies. Ceci fait, l'utilisateur va demander le calcul de la paie.

(d) la couche [ui] va demander à la couche métier de faire ce calcul. Pour cela elle va lui transmettre les données qu'elle a reçues de l'utilisateur. C'est l'opération [2].

la couche [metier] a besoin de certaines informations pour mener à bien son travail :

- des informations plus complètes sur la personne (adresse, indice, ...)
- les indemnités liées à son indice
- les taux des différentes cotisations sociales à prélever sur le salaire brut

Elle va demander ces informations à la couche [dao] avec le chemin [3, 4, 5, 6]. [3] est la demande initiale et [6] la réponse à cette demande.

(e) ayant toutes les données dont elle avait besoin, la couche [metier] calcule la paie de la personne choisie par l'utilisateur.
(f) la couche [metier] peut maintenant répondre à la demande de la couche [ui] faite en (d). C'est le chemin [7].
(g) la couche [ui] va mettre en forme ces résultats pour les présenter à l'utilisateur sous une forme appropriée puis les présenter. C'est le chemin [8].
(h) on peut imaginer que ces résultats doivent être mémorisés dans un fichier ou une base de données. Cela peut être fait de façon automatique. Dans ce cas, après l'opération (f), la couche [metier] va demander à la couche [dao] d'enregistrer les résultats. Ce sera le chemin [3, 4, 5, 6]. Cela peut être fait également sur demande de l'utilisateur. Ce sera le chemin [1-8] qui sera utilisé par le cycle demande - réponse.

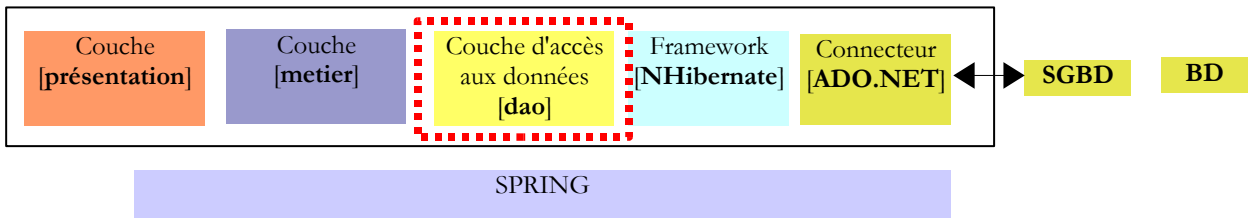
On voit dans cette description qu'une couche est amenée à utiliser les ressources de la couche qui est à sa droite, jamais de celle qui est à sa gauche.

Notre première implémentation de cette architecture 3 couches sera une application ASP.NET où

- les couches [dao] et [metier] seront implémentées par des DLL
- la couche [ui] sera implémentée par le formulaire web de la version 1 (cf page 45).

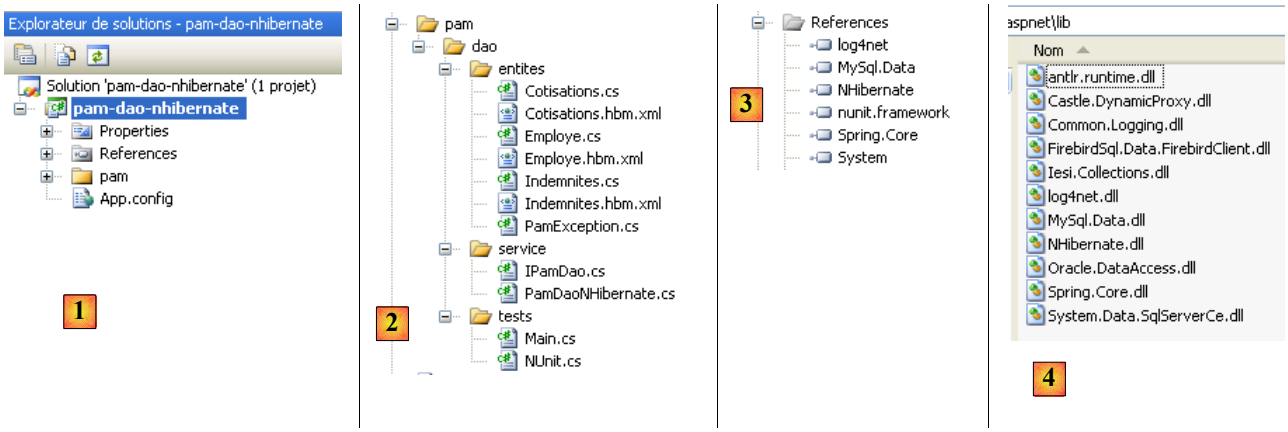
Nous commençons par implémenter la couche [dao] avec le framework NHibernate.

7.2 La couche [dao] d'accès aux données



7.2.1 Le projet Visual Studio C# de la couche [dao]

Le projet Visual Studio de la couche [dao] est le suivant :



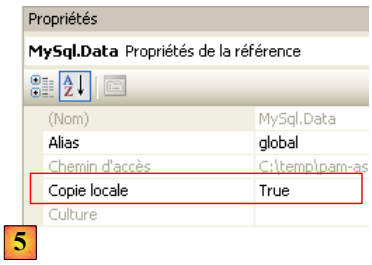
- en [1], le projet dans sa globalité
- en [2], les différentes classes du projet
- en [3], les références du projet.

- en [4], un dossier [lib] dans lequel ont été réunies les DLL nécessaires aux différents projets qui vont suivre

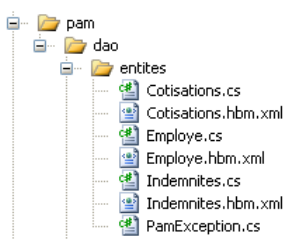
Dans les références [3] du projet, on trouve les DLL suivantes :

- *NHibernate* : pour l'ORM NHibernate
- *MySql.Data* : le pilote ADO.NET du SGBD MySQL
- *Spring.Core* : pour le framework Spring
- *log4net* : une bibliothèque de logs
- *nunit.framework* : une bibliothèque de tests unitaires

Ces références ont été prises dans le dosssier [lib] [4]. On prendra soin que toutes ces références aient leur propriété "Copie locale" à "True" [5] :



7.2.2 Les entités de la couche [dao]



Les entités (objets) nécessaires à la couche [dao] ont été rassemblées dans le dossier [entites] du projet. Certaines nous sont déjà connues : [Cotisations] décrite page 72, [Employe] décrite page 75, [Indemnites] décrite page 74. Elles sont toutes dans l'espace de noms [Pam.Dao.Entites].

La classe [Employe] évolue de la façon suivante :

```
1.  namespace Pam.Dao.Entites {
2.    public class Employe {
3.       // propriétés automatiques
4.       public virtual int Id { get; set; }
5.       public virtual int Version { get; set; }
6.       public virtual string SS { get; set; }
7.       public virtual string Nom { get; set; }
8.       public virtual string Prenom { get; set; }
9.       public virtual string Adresse { get; set; }
10.      public virtual string Ville { get; set; }
11.      public virtual string CodePostal { get; set; }
12.      public virtual Indemnites Indemnites { get; set; }
13.
14.      // constructeurs
15.      public Employe() {
16.      }
17.
18.      // ToString
19.      public override string ToString() {
20.         return string.Format("[{0},{1},{2},{3},{4},{5},{6}]", SS, Nom, Prenom, Adresse, Ville,
    CodePostal, Indemnites);
21.      }
```

```
22.    }
23. }
```

7.2.3 La classe [PamException]

La couche [dao] est chargée d'échanger des données avec une source extérieure. Cet échange peut échouer. Par exemple, si les informations sont demandées à un service distant sur Internet, leur obtention échouera sur une panne quelconque du réseau. Sur ce type d'erreurs, il est classique en Java de lancer une exception. Si l'exception n'est pas de type [RunTimeException] ou dérivé, il faut indiquer dans la signature de la méthode que celle-ci lance (throws) une exception. En .NET, toutes les exceptions sont non contrôlées, c.a.d. équivalentes au type [RunTimeException] de Java. Il n'y a alors pas lieu de déclarer que les méthodes [GetAllIdentitesEmployes, GetEmploye, GetCotisations] sont susceptibles de lancer une exception.

Il est cependant intéressant de pouvoir différencier les exceptions les unes des autres car leur traitement peut différer. Ainsi le code gérant divers types d'exceptions peut être écrit de la façon suivante :

```
try{
  ... code pouvant générer divers types d'exceptions
}catch (Exception1 ex1){
...on gère un type d'exceptions
}catch (Exception2 ex2){
...on gère un autre type d'exceptions
}finally{
...
}
```

Nous créons donc un type d'exceptions pour la couche [dao] de notre application. C'est le type [PamException] suivant :

```
1.  using System;
2.  namespace Pam.Dao.Entites {
3.
4.   public class PamException : Exception {
5.
6.      // le code de l'erreur
7.      public int Code { get; set; }
8.
9.      // constructeurs
10.     public PamException() {
11.     }
12.
13.     public PamException(int Code)
14.        : base() {
15.        this.Code = Code;
16.     }
17.
18.     public PamException(string message, int Code)
19.        : base(message) {
20.        this.Code = Code;
21.     }
22.
23.     public PamException(string message, Exception ex, int Code)
24.        : base(message, ex) {
25.        this.Code = Code;
26.     }
27.  }
28. }
```

- ligne 2 : la classe appartient à l'espace de noms [Pam.Dao.Entites]
- ligne 4 : la classe dérive de la classe [Exception]
- ligne 7 : elle a une propriété publique [Code] qui est un code d'erreur
- nous utiliserons dans notre couche [dao] deux sortes de constructeur :
  - celui des lignes 18-21 qu'on peut utiliser comme montré ci-dessous :

```
throw new PamException("Problème d'accès aux données",5);
```

  - ou celui des lignes 23-26 destiné à faire remonter une exception déjà survenue en l'encapsulant dans une exception de type [PamException] :

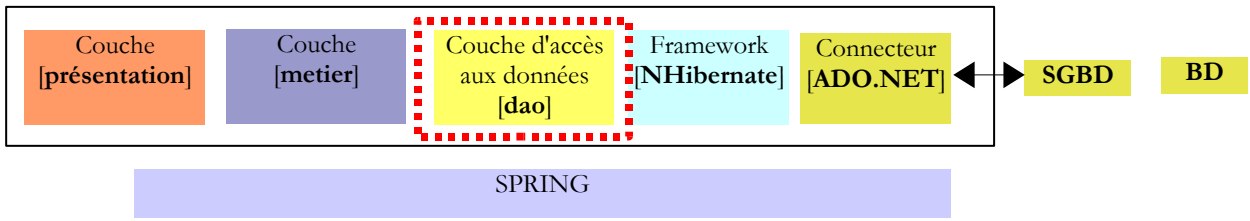
```
try{
....
}catch (IOException ex){
```

```
    // on encapsule l'exception
    throw new PamException("Problème d'accès aux données",ex,10);
}
```

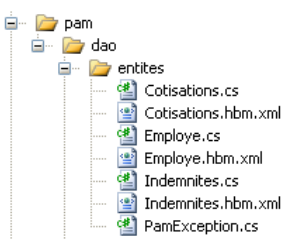
Cette seconde méthode a l'avantage de ne pas perdre l'information que peut contenir la première exception.

7.2.4 Les fichiers de mapping tables <--> classes de NHibernate

Revenons à l'architecture de l'application :



En lecture, le framework NHibernate exploite des données de la base de données et les transforme en objets dont nous venons de présenter les classes. En écriture, il fait l'inverse : à partir d'objets, il crée, met à jour, supprime des lignes dans les tables de la base de données. Les fichiers assurant la transformation tables <--> classes ont déjà été présentés :



- le fichier [Cotisations.hbm.xml] présenté page 72 fait la correspondance entre la table [COTISATIONS] et la classe [Cotisations]

```
1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
3.  namespace="Pam.Dao.Entites" assembly="pam-dao-nhibernate">
4.     <class name="Cotisations" table="COTISATIONS">
5.        <id name="Id" column="ID">
6.           <generator class="native" />
7.        </id>
8.        <version name="Version" column="VERSION"/>
9.        <property name="CsgRds" column="CSGRDS" not-null="true"/>
10.       <property name="Csgd" column="CSGD" not-null="true"/>
11.       <property name="Retraite" column="RETRAITE" not-null="true"/>
12.       <property name="Secu" column="SECU" not-null="true"/>
13.    </class>
14. </hibernate-mapping>
```

- le fichier [Employe.hbm.xml] présenté page 74 fait la correspondance entre la table [EMPLOYES] et la classe [Employe]

```
1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
3.  namespace="Pam.Dao.Entites" assembly="pam-dao-nhibernate">
4.     <class name="Employe" table="EMPLOYES">
5.        <id name="Id" column="ID">
6.           <generator class="native" />
7.        </id>
8.        <version name="Version" column="VERSION"/>
9.        <property name="SS" column="SS" length="15" not-null="true" unique="true"/>
10.       <property name="Nom" column="NOM" length="30" not-null="true"/>
11.       <property name="Prenom" column="PRENOM" length="20" not-null="true"/>
12.       <property name="Adresse" column="ADRESSE" length="50" not-null="true" />
```

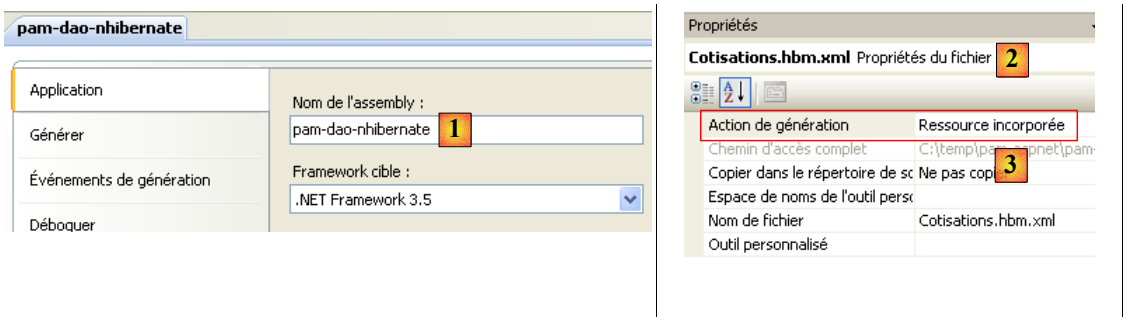
```
13.       <property name="Ville" column="VILLE" length="30" not-null="true"/>
14.       <property name="CodePostal" column="CP" length="5" not-null="true"/>
15.       <many-to-one name="Indemnites" column="INDEMNITE_ID" cascade="all" lazy="false"/>
16.    </class>
17. </hibernate-mapping>
```

- le fichier [Indemnites.hbm.xml] présenté page 73 fait la correspondance entre la table [INDEMNITES] et la classe [Indemnites]

```
1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
3.  namespace="Pam.Dao.Entites" assembly="pam-dao-nhibernate">
4.     <class name="Indemnites" table="INDEMNITES">
5.        <id name="Id" column="ID">
6.           <generator class="native" />
7.        </id>
8.        <version name="Version" column="VERSION"/>
9.        <property name="Indice" column="INDICE" not-null="true" unique="true"/>
10.       <property name="BaseHeure" column="BASE_HEURE" not-null="true"/>
11.       <property name="EntretienJour" column="ENTRETIEN_JOUR" not-null="true"/>
12.       <property name="RepasJour" column="REPAS_JOUR" not-null="true" />
13.       <property name="IndemnitesCp" column="INDEMNITES_CP" not-null="true"/>
14.    </class>
15. </hibernate-mapping>
```

On notera que dans la balise <hibernate-mapping> de ces fichiers (ligne 2), on a les attributs suivants :

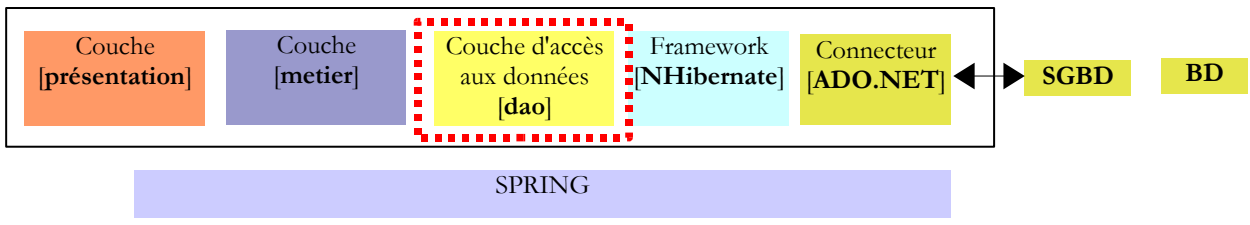
- *namespace* : *Pam.Dao.Entites*. Les classes [Cotisations], [Employe] et [Indemnites] doivent se trouver dans cet espace de noms.
- *assembly* : *pam-dao-nhibernate*. Les fichiers de mapping [*.hbm.xml] doivent être encapsulés dans une DLL nommée [pam-dao-nhibernate]. Pour obtenir ce résultat, le projet C# est configuré comme suit :



- en [1], l'assembly du projet a pour nom [pam-dao-nhibernate]
- en [2], les fichiers de mapping [*.hbm.xml] sont intégrés [3] à l'assembly du projet

7.2.5 L'interface [IPamDao] de la couche [dao]

Revenons à l'architecture de notre application :



Dans les cas simples, on peut partir de la couche [metier] pour découvrir les interfaces de l'application. Pour travailler, elle a besoin de données :

- déjà disponibles dans des fichiers, bases de données ou via le réseau. Elles sont fournies par la couche [dao].

- pas encore disponibles. Elles sont alors fournies par la couche [ui] qui les obtient auprès de l'utilisateur de l'application.

Quelle interface doit offrir la couche [dao] à la couche [metier] ? Quelles sont les interactions possibles entre ces deux couches ? La couche [dao] doit fournir les données suivantes à la couche [metier] :

- la liste des assistantes maternelles afin de permettre à l'utilisateur d'en choisir une en particulier
- des informations complètes sur la personne choisie (adresse, indice, ...)
- les indemnités liées à l'indice de la personne
- les taux des différentes cotisations sociales

Ces informations sont en effet connues avant le calcul de la paie et peuvent donc être mémorisées. Dans le sens [metier] -> [dao], la couche [metier] peut demander à la couche [dao] d'enregistrer le résultat du calcul de la paie. Nous ne le ferons pas ici.

Avec ces informations, on pourrait tenter une première définition de l'interface de la couche [dao] :

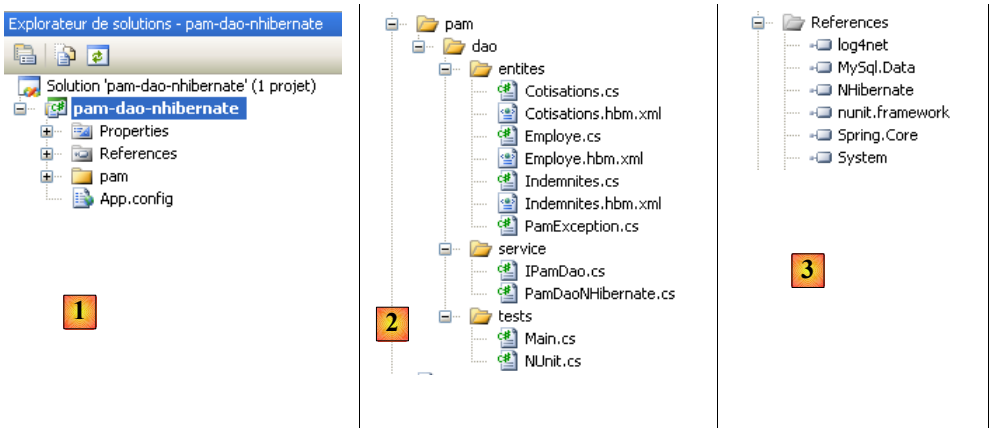
```
1.  using Pam.Dao.Entites;
2.
3.  namespace Pam.Dao.Service {
4.   public interface IPamDao {
5.      // liste de toutes les identités des employés
6.      Employe[] GetAllIdentitesEmployes();
7.      // un employé particulier avec ses indemnités
8.      Employe GetEmploye(string ss);
9.      // liste de toutes les cotisations
10.     Cotisations GetCotisations();
11.  }
12. }
```

- ligne 1 : on importe l'espace de noms des entités de la couche [dao].
- ligne 3 : la couche [dao] est dans l'espace de noms [Pam.Dao.Service]. Les éléments de l'espace de noms [Pam.Dao.Entites] peuvent être créés en plusieurs exemplaires. Les éléments de l'espace de noms [Pam.Dao.Service] sont créés en un unique exemplaire (singleton). C'est ce qui a justifié le choix des noms des espaces de noms.
- ligne 4 : l'interface s'appelle [IPamDao]. Elle définit trois méthodes :
  - ligne 6, [GetAllIdentitesEmployes] rend un tableau d'objets de type [Employe] qui représente la liste des assistantes maternelles sous une forme simplifiée (nom, prénom, SS).
  - ligne 8, [GetEmploye] rend un objet [Employe] : l'employé qui a le n° de sécurité sociale passé en paramètre à la méthode avec les indemnités liées à son indice.
  - ligne 10, [GetCotisations] rend l'objet [Cotisations] qui encapsule les taux des différentes cotisations sociales à prélever sur le salaire brut.

7.3 Implémentation et tests de la couche [dao]

7.3.1 Le projet Visual Studio

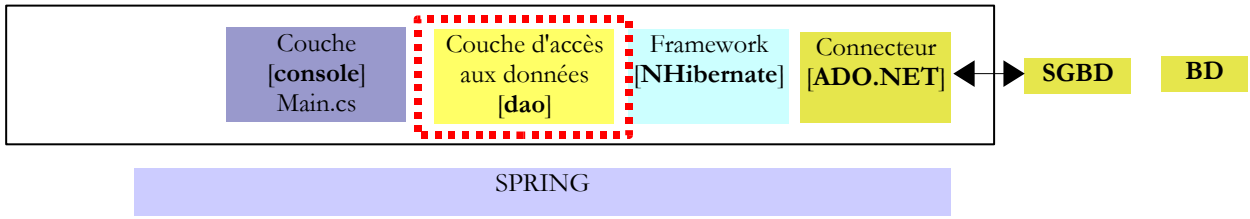
Le projet Visual Studio a déjà été présenté. Rappelons-le :



- en [1], le projet dans sa globalité
- en [2], les différentes classes du projet. Le dossier [entites] contient les entités manipulées par la couche [dao] ainsi que les fichiers de mapping NHibernate. Le dossier [service] contient l'interface [IPamDao] et son implémentation [PamDaoNHibernate]. Le dossier [tests] contient un test console [Main.cs] et un test unitaire [NUnit.cs].
- en [3], les références du projet.

7.3.2 Le programme de test console [Main.cs]

Le programme de test [Main.cs] est exécuté dans l'architecture suivante :



Il est chargé de tester les méthodes de l'interface [IPamDao]. Un exemple basique pourrait être le suivant :

```
1.  using System;
2.  using Pam.Dao.Entites;
3.  using Pam.Dao.Service;
4.  using Spring.Context.Support;
5.
6.  namespace Pam.Dao.Tests {
7.   public class MainPamDaoTests {
8.      public static void Main() {
9.         try {
10.           // instanciation couche [dao]
11.           IPamDao pamDao = (IPamDao)ContextRegistry.GetContext().GetObject("pamdao");
12.           // liste des identités des employés
13.           foreach (Employe Employe in pamDao.GetAllIdentitesEmployes()) {
14.              Console.WriteLine(Employe.ToString());
15.           }
16.           // un employé avec ses indemnités
17.           Console.WriteLine("-------------------------------------");
18.           Console.WriteLine(pamDao.GetEmploye("254104940426058"));
19.           Console.WriteLine("-------------------------------------");
20.           // liste des cotisations
21.           Cotisations cotisations = pamDao.GetCotisations();
22.           Console.WriteLine(cotisations.ToString());
23.        } catch (Exception ex) {
24.           // affichage exception
25.           Console.WriteLine(ex.ToString());
26.        }
27.        //pause
28.        Console.ReadLine();
29.     }
30.  }
31. }
```

- ligne 11 : on demande à Spring une référence sur la couche [dao].
- lignes 13-15 : test de la méthode [GetAllIdentitesEmployes] de l'interface [IPamDao]
- ligne 18 : test de la méthode [GetEmploye] de l'interface [IPamDao]
- ligne 21 : test de la méthode [GetCotisations] de l'interface [IPamDao]

Spring, NHibernate et log4net sont configurés par le fichier [App.config] suivant :

```
1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <configuration>
3.     <!-- sections de configuration -->
4.     <configSections>
5.        <section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler,log4net" />
6.        <sectionGroup name="spring">
7.           <section name="objects" type="Spring.Context.Support.DefaultSectionHandler, Spring.Core" />
8.           <section name="context" type="Spring.Context.Support.ContextHandler, Spring.Core" />
9.        </sectionGroup>
```

```
10.         <section name="hibernate-configuration" type="NHibernate.Cfg.ConfigurationSectionHandler, NHibernate" />
11.     </configSections>
12.
13.
14.     <!-- configuration Spring -->
15.     <spring>
16.         <context>
17.             <resource uri="config://spring/objects" />
18.         </context>
19.         <objects xmlns="http://www.springframework.net">
20.             <object id="pamdao" type="Pam.Dao.Service.PamDaoNHibernate, pam-dao-nhibernate" init-method="init"
    destroy-method="destroy"/>
21.         </objects>
22.     </spring>
23.
24.     <!-- configuration NHibernate -->
25.     <hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">
26.         <session-factory>
27.             <property name="connection.provider">NHibernate.Connection.DriverConnectionProvider</property>
28.             <property name="connection.driver_class">NHibernate.Driver.MySqlDataDriver</property>
29.             <property name="dialect">NHibernate.Dialect.MySQLDialect</property>
30.             <property name="connection.connection_string">
31.                 Server=localhost;Database=dbpam_nhibernate;Uid=root;Pwd=;
32.             </property>
33.             <property name="show_sql">false</property>
34.             <mapping assembly="pam-dao-nhibernate"/>
35.         </session-factory>
36.     </hibernate-configuration>
37.
38.     <!-- This section contains the log4net configuration settings -->
39.     <!-- NOTE IMPORTANTE : les logs ne sont pas actifs par défaut. Il faut les activer par programme
40.     avec l'instruction log4net.Config.XmlConfigurator.Configure();
41.     ! -->
42.     <log4net>
43.     ...
44.     </log4net>
45.
46. </configuration>
```

La configuration de NHibernate (ligne 10, lignes 25-36) a été expliquée page 67. On notera la ligne 34 qui indique que les fichiers de mapping se trouvent dans l'assembly [pam-dao-nhibernate]. C'est l'assembly du projet.

La configuration de Spring est faite aux lignes 6-9, 15-22. La ligne 20 définit l'objet [pamdao] utilisé par le programme console [Main.cs]. La balise <object> a ici les attributs suivants :

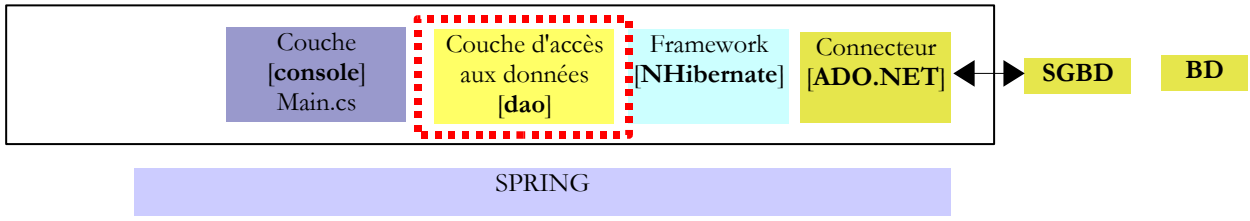
- *type* : fixe la classe à instancier. C'est la classe [PamDaoNHibernate] qui implémente l'interface [IPamDao]. Elle sera trouvée dans la DLL [pam-dao-nhibernate] du projet.
- *init-method* : la méthode de la classe [PamDaoNHibernate] à exécuter après instanciation de la classe
- *destroy-method* : la méthode de la classe [PamDaoNHibernate] à exécuter lorsque le conteneur Spring est détruit à la fin de l'exécution du projet.

L'exécution faite avec la base de données décrite au paragraphe 6.2, page 63, donne le résultat console suivant :

```
1. [254104940426058,Jouveinal,Marie,,,,]
2. [260124402111742,Laverti,Justine,,,,]
3. ------------------------------------
4. [254104940426058,Jouveinal,Marie,5 rue des oiseaux,St Corentin,49203,[2, 2,1, 2,1, 3,1, 15]]
5. ------------------------------------
6. [3,49,6,15,9,39,7,88]
```

- lignes 1-2 : les 2 employés de type [Employe] avec les seules informations [SS, Nom, Prenom]
- ligne 4 : l'employé de type [Employe] ayant le n° de sécurité sociale [254104940426058]
- ligne 5 : les taux de cotisations

7.3.3 Écriture de la classe [PamDaoNHibernate]



L'interface [IPamDao] implémentée par la couche [dao] est la suivante :

```
1.  using Pam.Dao.Entites;
2.
3.  namespace Pam.Dao.Service {
4.   public interface IPamDao {
5.      // liste de toutes les identités des employés
6.      Employe[] GetAllIdentitesEmployes();
7.      // un employé particulier avec ses indemnités
8.      Employe GetEmploye(string ss);
9.      // liste de toutes les cotisations
10.     Cotisations GetCotisations();
11.  }
12. }
```

Question : écrire le code de la classe [PamDaoNHibernate] implémentant l'interface [IPamDao] ci-dessus à l'aide du framework NHibernate configuré tel qu'il a été présenté précédemment. On implémentera également les méthodes **init** et **destroy** exécutées par Spring. La méthode *init* créera la *SessionFactory* auprès de laquelle on obtiendra des objets *Session*. La méthode *destroy* fermera cette *SessionFactory*. On s'aidera des exemples du paragraphe 6.5, page 79.

Contraintes :

On supposera que certaines données demandées à la couche [dao] peuvent entièrement tenir en mémoire. Ainsi, pour améliorer les performances, la classe [PamDaoNHibernate] mémorisera :

- la table [EMPLOYES] sous la forme (SS, NOM, PRENOM) nécessitée par la méthode [GetAllIdentitesEmployes] sous la forme d'un tableau d'objets de type [Employe]
- la table [COTISATIONS] sous la forme d'un unique objet de type [Cotisations]

Cela sera fait dans la méthode [init] de la classe. Le squelette de la classe [PamDaoNHibernate] pourrait être le suivant :

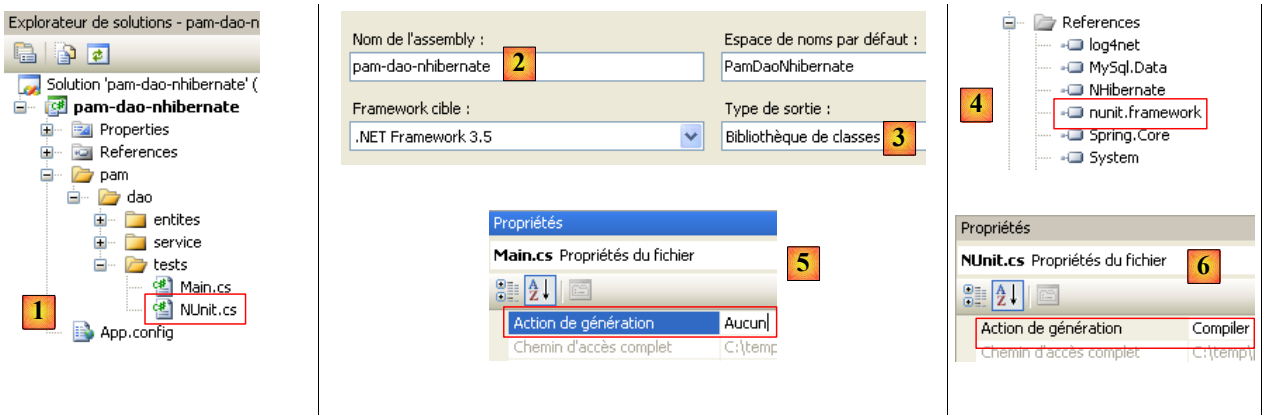
```
1.  using System;
2.  ...
3.
4.  namespace Pam.Dao.Service {
5.    class PamDaoNHibernate : IPamDao {
6.       // champs privés
7.       private Cotisations cotisations;
8.       private Employe[] employes;
9.       private ISessionFactory sessionFactory = null;
10.
11.      // init
12.      public void init() {
13.         try {
14.            // initialisation factory
15.            sessionFactory = new Configuration().Configure().BuildSessionFactory();
16.            // on récupère les taux de cotisations et les employés pour les mettre en cache
17. ......................
18.      }
19.
20.      // fermeture SessionFactory
21.      public void destroy() {
22.         if (sessionFactory != null) {
23.            sessionFactory.Close();
24.         }
25.      }
26.
27.      // liste de toutes les identités des employés
28.      public Employe[] GetAllIdentitesEmployes() {
29.         return employes;
30.      }
31.
32.      // un employé particulier avec ses indemnités
33.      public Employe GetEmploye(string ss) {
34. ..............................
35.      }
36.
37.      // liste des cotisations
38.      public Cotisations GetCotisations() {
39.         return cotisations;
40.      }
41.   }
42. }
```

7.3.4 Tests unitaires avec NUnit

Lectures conseillées : **"Langage C# 2008, Chapitre 4 : Architectures 3 couches, tests NUnit, framework Spring"**.

Le test précédent avait été visuel : on vérifiait à l'écran qu'on obtenait bien les résultats attendus. C'est une méthode insuffisante en milieu professionnel. Les tests doivent toujours être automatisés au maximum et viser à ne nécessiter aucune intervention humaine. L'être humain est en effet sujet à la fatigue et sa capacité à vérifier des tests s'émousse au fil de la journée. L'outil [NUnit] aide à réaliser cette automatisation. Il est disponible à l'Url [http://www.nunit.org/].

Le projet Visual Studio de la couche [dao] va évoluer de la façon suivante :



- en [1], le programme de test [NUnit.cs]
- en [2,3], le projet va générer une DLL nommé [pam-dao-nhibernate.dll]
- en [4], la référence à la DLL du framework NUnit : [nunit.framework.dll]
- en [5], la classe [Main.cs] ne sera pas incluse dans la DLL [pam-dao-nhibernate]
- en [6], la classe [NUnit.cs] sera incluse dans la DLL [pam-dao-nhibernate]

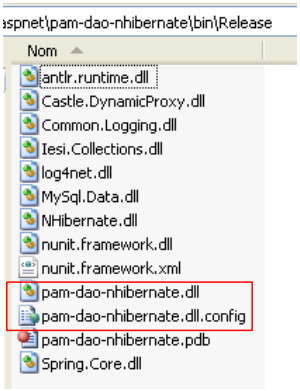
La classe de test NUnit est la suivante :

```
1.  using System.Collections;
2.  using NUnit.Framework;
3.  using Pam.Dao.Service;
4.  using Pam.Dao.Entites;
5.  using Spring.Objects.Factory.Xml;
6.  using Spring.Core.IO;
7.  using Spring.Context.Support;
8.
9.  namespace Pam.Dao.Tests {
10.
11.  [TestFixture]
12.  public class NunitPamDao : AssertionHelper {
13.     // la couche [dao] à tester
14.     private IPamDao pamDao = null;
15.
16.     // constructeur
17.     public NunitPamDao() {
18.        // instanciation couche [dao]
19.        pamDao = (IPamDao)ContextRegistry.GetContext().GetObject("pamdao");
20.     }
21.
22.     // init
23.     [SetUp]
24.     public void Init() {
25.
26.     }
27.
28.     [Test]
29.     public void GetAllIdentitesEmployes() {
30.        // vérification nbre d'employes
31.        Expect(2, EqualTo(pamDao.GetAllIdentitesEmployes().Length));
32.     }
33.
```

```
34.     [Test]
35.     public void GetCotisations() {
36.        // vérification taux de cotisations
37.        Cotisations cotisations = pamDao.GetCotisations();
38.        Expect(3.49, EqualTo(cotisations.CsgRds).Within(1E-06));
39.        Expect(6.15, EqualTo(cotisations.Csgd).Within(1E-06));
40.        Expect(9.39, EqualTo(cotisations.Secu).Within(1E-06));
41.        Expect(7.88, EqualTo(cotisations.Retraite).Within(1E-06));
42.     }
43.
44.     [Test]
45.     public void GetEmployeIdemnites() {
46.        // vérification individus
47.        Employe employe1 = pamDao.GetEmploye("254104940426058");
48.        Employe employe2 = pamDao.GetEmploye("260124402111742");
49.        Expect("Jouveinal", EqualTo(employe1.Nom));
50.        Expect(2.1, EqualTo(employe1.Indemnites.BaseHeure).Within(1E-06));
51.        Expect("Laverti", EqualTo(employe2.Nom));
52.        Expect(1.93, EqualTo(employe2.Indemnites.BaseHeure).Within(1E-06));
53.     }
54.
55.     [Test]
56.     public void GetEmployeIdemnites2() {
57.        // vérification individu inexistant
58.        bool erreur = false;
59.        try {
60.           Employe employe1 = pamDao.GetEmploye("xx");
61.        } catch {
62.           erreur = true;
63.        }
64.        Expect(erreur, True);
65.     }
66.  }
67. }
```

- ligne 11 : la classe a l'attribut [*TestFixture*] qui en fait une classe de test [NUnit].
- ligne 12 : la classe dérive de la classe utilitaire *AssertionHelper* du framework *NUnit* (à partir de la version 2.4.6).
- ligne 14 : le champ privé [pamDao] est une instance de l'interface d'accès à la couche [dao]. On notera que le type de ce champ est une **interface** et non une **classe**. Cela signifie que l'instance [pamDao] ne rend accessibles que des méthodes, celles de l'interface [IPamDao].
- les méthodes testées dans la classe sont celles ayant l'attribut [*Test*]. Pour toutes ces méthodes, le processus de test est le suivant :
  - la méthode ayant l'attribut [*SetUp*] est tout d'abord exécutée. Elle sert à préparer les ressources (connexions réseau, connexions aux bases de données, ...) nécessaires au test.
  - puis la méthode à tester est exécutée
  - et enfin la méthode ayant l'attribut [*TearDown*] est exécutée. Elle sert généralement à libérer les ressources mobilisées par la méthode d'attribut [*SetUp*].
- dans notre test, il n'y a pas de ressources à allouer avant chaque test et à désallouer ensuite. Aussi n'avons-nous pas besoin de méthode avec les attributs [*SetUp*] et [*TearDown*]. Pour l'exemple, nous avons présenté, lignes 23-26, une méthode avec l'attribut [*SetUp*].
- lignes 17-20 : le constructeur de la classe initialise le champ privé [pamDao] à l'aide de Spring et [App.config].
- lignes 29-32 : testent la méthode [GetAllIdentitesEmployes]
- lignes 35-42 : testent la méthode [GetCotisations]
- lignes 45-53 : testent la méthode [GetEmploye]
- lignes 56-65 : testent la méthode [GetEmploye] lors d'une exception.

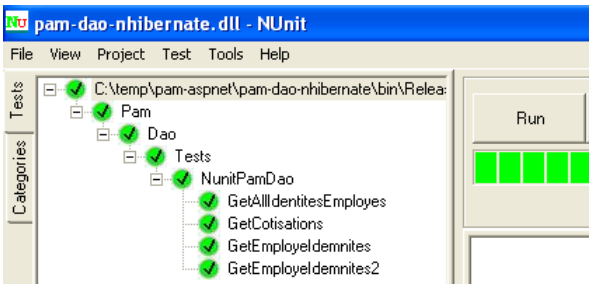
La génération du projet crée la DLL [pam-dao-nhibernate.dll] dans le dossier [bin/Release].



Le dossier [bin/Release] contient en outre :

- les DLL qui font partie des références du projet et qui ont l'attribut [Copie locale] à vrai : [Spring.Core, MySql.data, NHibernate, log4net]. Ces DLL sont accompagnées des copies des DLL qu'elles utilisent elles-mêmes :
  - [CastleDynamicProxy, Iesi.Collections] pour l'outil NHibernate
  - [antlr.runtime, Common.Logging] pour l'outil Spring
- le fichier [pam-dao-nhibernate.dll.config] est une copie du fichier de configuration [App.config]. C'est VS qui opère cette duplication. A l'exécution c'est le fichier [pam-dao-nhibernate.dll.config] qui est utilisé et non [App.config].

On charge la DLL [pam-dao-nhibernate.dll] avec l'outil [NUnit-Gui], version 2.4.6 et on exécute les tests :



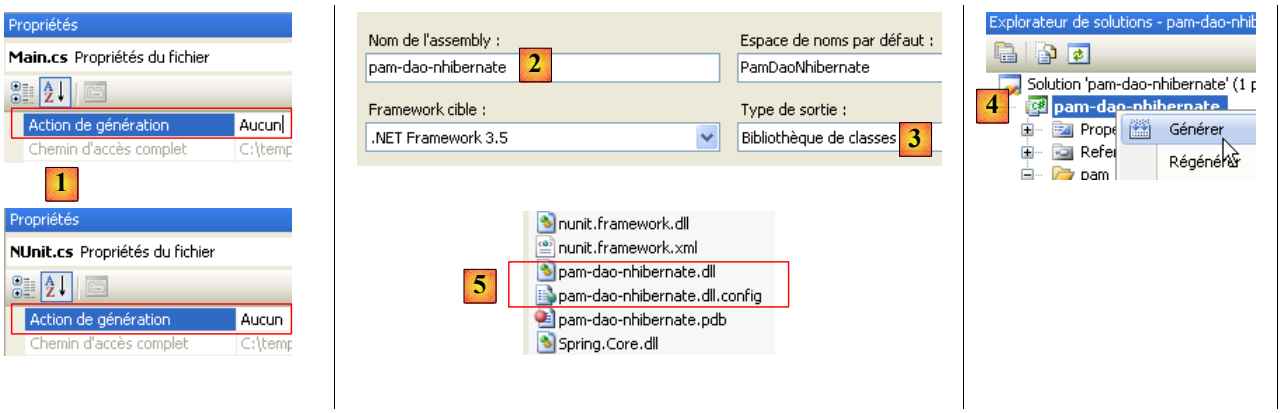
Ci-dessus, les tests ont été réussis.

Travail pratique :

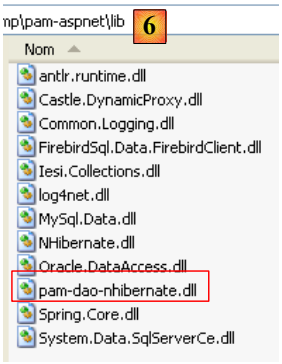
- mettre en oeuvre sur machine les tests de la classe [PamDaoNHibernate].
- utiliser différents fichiers de configuration [App.config] afin d'utiliser des SGBD différents (Firebird, MySQL, Postgres, SQL Server)

7.3.5 Génération de la DLL de la couche [dao]

Une fois écrite et testée la classe [PamDaoNHibernate], on génèrera la DLL de la couche [dao] de la façon suivante :

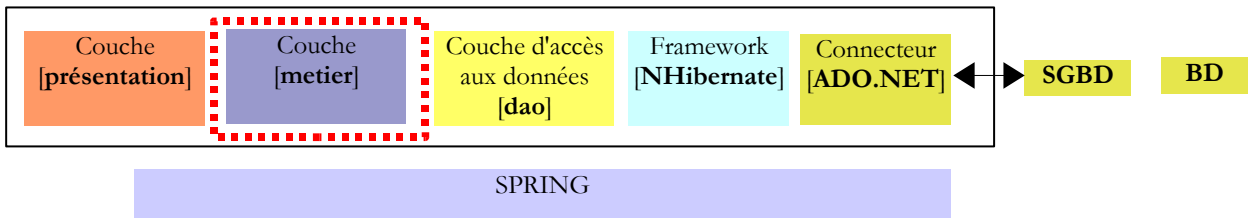


- [1], les programmes de test sont exlus de l'assembly du projet
- [2,3], configuration du projet
- [4], génération du projet
- la DLL est générée dans le dossier [bin/Release] [5]. Nous l'ajoutons aux DLL déjà présentes dans le dossier [lib] [6] :



7.4 La couche métier

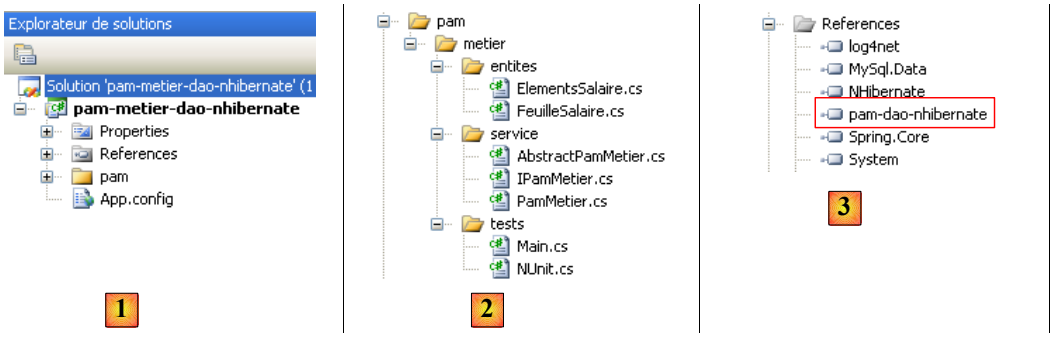
Revenons sur l'architecture générale de l'application [SimuPaie] :



Nous considérons désormais que la couche [dao] est acquise et qu'elle a été encapsulée dans la DLL [pam-dao-nhibernate.dll]. Nous nous intéressons maintenant à la couche [metier]. C'est elle qui implémente les règles métier, ici les règles de calcul d'un salaire.

7.4.1 Le projet Visual Studio de la couche [metier]

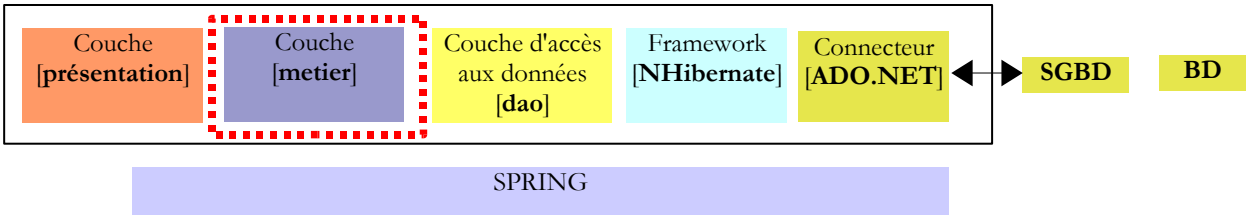
Le projet Visual Studio de la couche métier pourrait ressembler à ce qui suit :



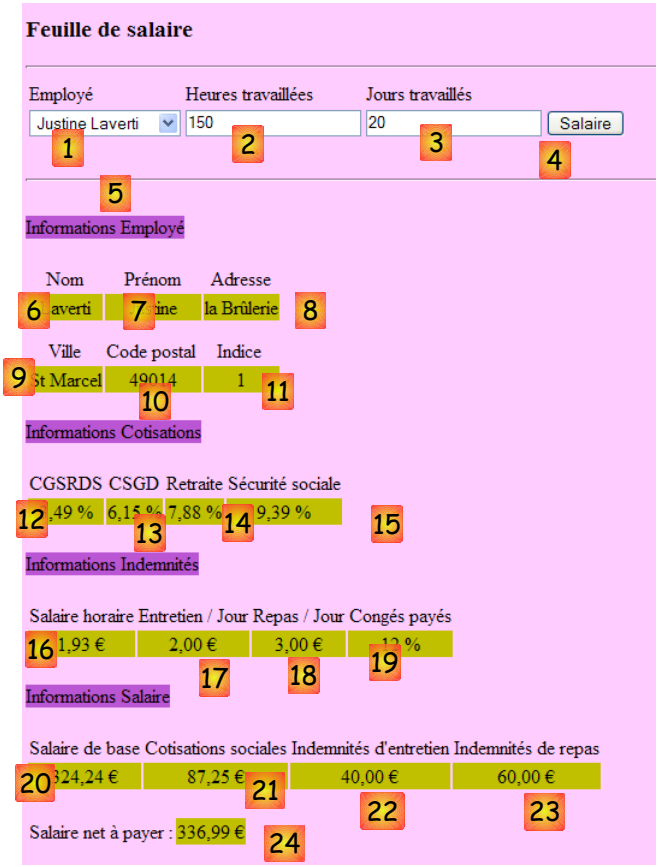
- en [1] l'ensemble du projet configuré par le fichier [App.config]
- en [2], la couche [metier] est formée des deux dossiers [entites, service]. Le dossier [tests] contient un programme de test console (Main.cs) et un programme de test NUnit (NUnit.cs).
- en [3] les références utilisées par le projet. On notera la DLL [pam-dao-nhibernate] de la couche [dao] étudiée précédemment.

7.4.2 L'interface [IPamMetier] de la couche [metier]

Revenons à l'architecture générale de l'application :



Quelle interface doit offrir la couche [metier] à la couche [ui] ? Quelles sont les interactions possibles entre ces deux couches ? Rappelons-nous l'interface web qui sera présentée à l'utilisateur :



1. à l'affichage initial du formulaire, on doit trouver en [1] la liste des employés. Une liste simplifiée suffit (Nom, Prénom, SS). Le n° SS est nécessaire pour avoir accès aux informations complémentaires sur l'employé sélectionné (informations 6 à 11).
2. les informations 12 à 15 sont les différents taux de cotisations.
3. les informations 16 à 19 sont les indemnités liées à l'indice de l'employé
4. les informations 20 à 24 sont les éléments du salaire calculés à partir des saisies 1 à 3 faites par l'utilisateur.

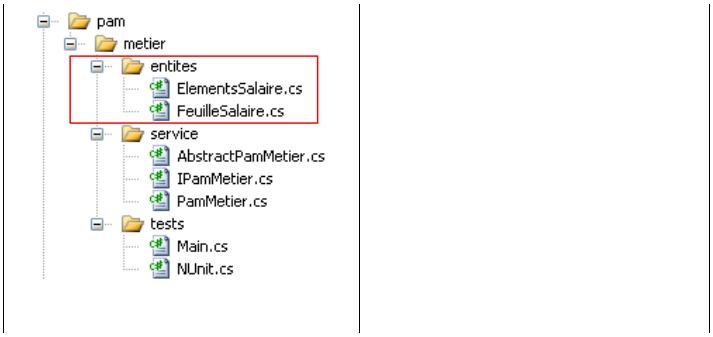
L'interface [IPamMetier] offerte à la couche [ui] par la couche [metier] doit répondre aux exigences ci-dessus. Il existe de nombreuses interfaces possibles. Nous proposons la suivante :

```
1.  using Pam.Dao.Entites;
2.  using Pam.Metier.Entites;
3.
4.  namespace Pam.Metier.Service {
5.   public interface IPamMetier {
6.      // liste de toutes les identités des employés
7.      Employe[] GetAllIdentitesEmployes();
8.
9.      // ------- le calcul du salaire
10.     FeuilleSalaire GetSalaire(string ss, double heuresTravaillées, int joursTravaillés);
11.  }
12. }
```

- ligne 7 : la méthode qui permettra le remplissage du combo [1]
- ligne 10 : la méthode qui permettra d'obtenir les renseignements 6 à 24. Ceux-ci ont été rassemblés dans un objet de type [FeuilleSalaire].

7.4.3 Les entités de la couche [metier]

Le dossier [entites] du projet Visual Studio contient les objets manipulés par la classe métier : [FeuilleSalaire] et [ElementsSalaire].



La classe [FeuilleSalaire] encapsule les informations 6 à 24 du formulaire précédent :

```
1.  using Pam.Dao.Entites;
2.
3.  namespace Pam.Metier.Entites {
4.
5.     public class FeuilleSalaire {
6.
7.        // propriétés automatiques
8.        public Employe Employe { get; set; }
9.        public Cotisations Cotisations { get; set; }
10.       public ElementsSalaire ElementsSalaire { get; set; }
11.
12.       // ToString
13.       public override string ToString() {
14.          return string.Format("[{0},{1},{2}", Employe, Cotisations, ElementsSalaire);
15.       }
16.    }
17. }
```

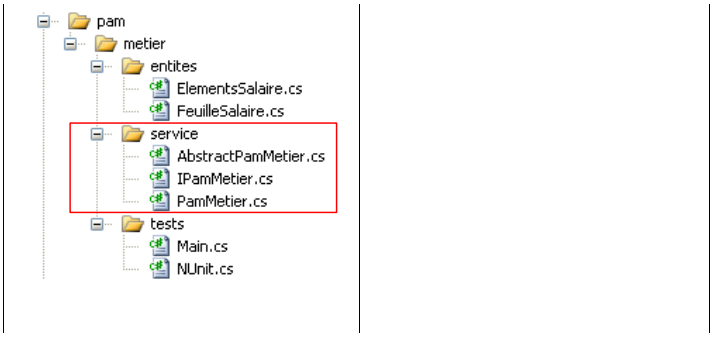
- ligne 8 : les informations 6 à 11 sur l'employé dont on calcule le salaire et les informations 16 à 19 sur ses indemnités. Il ne faut pas oublier ici qu'un objet [Employe] encapsule un objet [Indemnites] représentant les indemnités de l'employé.
- ligne 9 : les informations 12 à 15
- ligne 10 : les informations 20 à 24
- lignes 13-15 : la méthode [ToString]

La classe [ElementsSalaire] encapsule les informations 20 à 24 du formulaire :

```
1.  namespace Pam.Metier.Entites {
2.     public class ElementsSalaire {
3.        // propriétés automatiques
4.        public double SalaireBase { get; set; }
5.        public double CotisationsSociales { get; set; }
6.        public double IndemnitesEntretien { get; set; }
7.        public double IndemnitesRepas { get; set; }
8.        public double SalaireNet { get; set; }
9.
10.
11.       // ToString
12.       public override string ToString() {
13.          return string.Format("[{0} : {1} : {2} : {3} : {4} ]", SalaireBase, CotisationsSociales,
    IndemnitesEntretien, IndemnitesRepas, SalaireNet);
14.       }
15.    }
16. }
```

- lignes 4-8 : les éléments du salaire tels qu'expliqués dans les règles métier décrites page 39.
- ligne 4 : le salaire de base de l'employé, fonction du nombre d'heures travaillées
- ligne 5 : les cotisations prélevées sur ce salaire de base
- lignes 6 et 7 : les indemnités à ajouter au salaire de base, fonction de l'indice de l'employé et du nombre de jours travaillés
- ligne 8 : le salaire net à payer
- lignes 12-15 : la méthode [ToString] de la classe.

7.4.4 Implémentation de la couche [metier]



Nous allons implémenter l'interface [IPamMetier] avec deux classes :

- [AbstractBasePamMetier] qui est une classe abstraite dans laquelle on implémentera l'accès aux données de l'interface [IPamMetier]. Cette classe aura une référence sur la couche [dao].
- [PamMetier] une classe dérivée de [AbstractBasePamMetier] qui elle, implémentera les règles métier de l'interface [IPamMetier]. Elle sera ignorante de la couche [dao].

La classe [AbstractBasePamMetier] sera la suivante :

```
1.  using Pam.Dao.Entites;
2.  using Pam.Dao.Service;
3.  using Pam.Metier.Entites;
4.
5.  namespace Pam.Metier.Service {
6.   public abstract class AbstractBasePamMetier : IPamMetier {
7.
8.      // l'objet d'accès aux données
9.      public IPamDao PamDao { get; set; }
10.
11.     // liste de toutes les identités des employés
12.     public Employe[] GetAllIdentitesEmployes() {
13.        return PamDao.GetAllIdentitesEmployes();
14.     }
15.
16.     // un employé particulier avec ses indemnités
17.     protected Employe GetEmploye(string ss) {
18.        return PamDao.GetEmploye(ss);
19.     }
20.
21.     // les cotisations
22.     protected Cotisations GetCotisations() {
23.        return PamDao.GetCotisations();
24.     }
25.
26.     // le calcul du salaire
27.     public abstract FeuilleSalaire GetSalaire(string ss, double heuresTravaillées, int
    joursTravaillés);
28.  }
29. }
```

- ligne 5 : la classe appartient à l'espace de noms [Pam.Metier.Service] comme toutes les classes et interfaces de la couche [metier].
- ligne 6 : la classe est abstraite (attribut *abstract*) et implémente l'interface [IPamMetier]
- ligne 9 : la classe détient une référence sur la couche [dao] sous la forme d'une propriété publique
- lignes 12-14 : implémentation de la méthode [GetAllIdentitesEmployes] de l'interface [IPamMetier] – utilise la méthode de même nom de la couche [dao]
- lignes 17-19 : méthode interne (protected) [GetEmploye] qui fait appel à la méthode de même nom de la couche [dao] – déclarée *protected* pour que les classes dérivées puissent y avoir accès sans qu'elle soit publique.
- lignes 22-24 : méthode interne (protected) [GetCotisations] qui fait appel à la méthode de même nom de la couche [dao]
- ligne 27 : implémentation abstraite (attribut *abstract*) de la méthode [GetSalaire] de l'interface [IPamMetier].

Le calcul du salaire est implémenté par la classe [PamMetier] suivante :

```
1.  using System;
2.  using Pam.Dao.Entites;
3.  using Pam.Metier.Entites;
4.
```

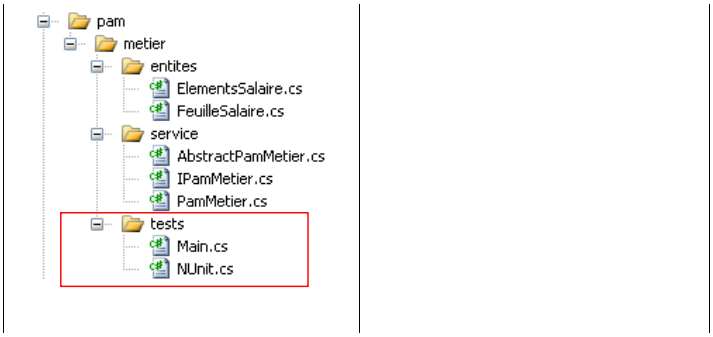
```
5.  namespace Pam.Metier.Service {
6.
7.   public class PamMetier : AbstractBasePamMetier {
8.
9.      // calcul du salaire
10.     public override FeuilleSalaire GetSalaire(string ss, double heuresTravaillées, int
    joursTravaillés) {
11.        // SS : n° SS de l'employé
12.        // HeuresTravaillées : le nombre d'heures travaillés
13.        // Jours Travaillés : nbre de jours travaillés
14.        // on récupère l'employé avec ses indemnités
15.        ...
16.        // on récupère les divers taux de cotisation
17.        ...
18.        // on calcule les éléments du salaire
19.        ...
20.        // on rend la feuille de salaire
21.        return ...;
22.     }
23.  }
24. }
```

- ligne 7 : la classe dérive de [AbstractBasePamMetier] et donc implémente de ce fait l'interface [IPamMetier]
- ligne 10 : la méthode [GetSalaire] à implémenter

Question : écrire le code de la méthode [GetSalaire].

7.4.5 Le test console de la couche [metier]

Rappelons le projet Visual Studio de la couche [metier] :



Le programme de test [Main] ci-dessus teste les méthodes de l'interface [IPamMetier]. Un exemple basique pourrait être le suivant :

```
1.  using System;
2.  using Pam.Dao.Entites;
3.  using Pam.Metier.Service;
4.  using Spring.Context.Support;
5.
6.  namespace Pam.Metier.Tests {
7.   class MainPamMetierTests {
8.      public static void Main() {
9.         try {
10.           // instanciation couche [metier]
11.           IPamMetier pamMetier = ContextRegistry.GetContext().GetObject("pammetier") as
    IPamMetier;
12.           // calculs de feuilles de salaire
13.           Console.WriteLine(pamMetier.GetSalaire("260124402111742", 30, 5));
14.           Console.WriteLine(pamMetier.GetSalaire("254104940426058", 150, 20));
15.           try {
16.              Console.WriteLine(pamMetier.GetSalaire("xx", 150, 20));
17.           } catch (PamException ex) {
18.              Console.WriteLine(string.Format("PamException : {0}", ex.Message));
19.           }
20.        } catch (Exception ex) {
21.           Console.WriteLine(string.Format("Exception : {0}", ex.ToString()));
22.        }
23.        // pause
```

```
24.         Console.ReadLine();
25.     }
26.  }
27. }
```

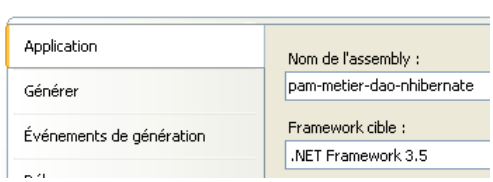
- ligne 11 : instanciation par Spring de la couche [metier].
- lignes 13-14 : tests de la méthode [GetSalaire] de l'interface [IPamMetier]
- lignes 15-22 : test de la méthode [GetSalaire] lorsqu'il se produit une exception

Le programme de test utilise le fichier de configuration [App.config] suivant :

```
1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <configuration>
3.     <!-- sections de configuration -->
4.     <configSections>
5.        <section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler,log4net" />
6.        <sectionGroup name="spring">
7.           <section name="objects" type="Spring.Context.Support.DefaultSectionHandler, Spring.Core" />
8.           <section name="context" type="Spring.Context.Support.ContextHandler, Spring.Core" />
9.        </sectionGroup>
10.       <section name="hibernate-configuration" type="NHibernate.Cfg.ConfigurationSectionHandler, NHibernate" />
11.    </configSections>
12.
13.
14.    <!-- configuration Spring -->
15.    <spring>
16.       <context>
17.          <resource uri="config://spring/objects" />
18.       </context>
19.       <objects xmlns="http://www.springframework.net">
20.          <object id="pamdao" type="Pam.Dao.Service.PamDaoNHibernate, pam-dao-nhibernate" init-method="init"
    destroy-method="destroy"/>
21.          <object id="pammetier" type="Pam.Metier.Service.PamMetier, pam-metier-dao-nhibernate" >
22.             <property name="PamDao" ref="pamdao"/>
23.          </object>
24.       </objects>
25.    </spring>
26.
27.    <!-- configuration NHibernate -->
28.    <hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">
29. ....
30.    </hibernate-configuration>
31.
32.    <!-- This section contains the log4net configuration settings -->
33.    <!-- NOTE IMPORTANTE : les logs ne sont pas actifs par défaut. Il faut les activer par programme
34.    avec l'instruction log4net.Config.XmlConfigurator.Configure();
35.    ! -->
36.    <log4net>
37. ...
38.    </log4net>
39.
40. </configuration>
```

Ce ficheir est identique au fichier [App.config] utilisé pour le projet de la couche [dao] (cf page 95) aux détails près suivants :

- ligne 20 : l'objet d'id "pamdao" a le type [Pam.Dao.Service.PamDaoNHibernate] et est trouvé dans l'assembly [pam-dao-nhibernate]. La couche [dao] est celle étudiée précédemment.
- lignes 21-23 : l'objet d'id "pammetier" a le type [Pam.Metier.Service.PamMetier] et est trouvé dans l'assembly [pam-metier-dao-nhibernate]. Il faut configurer le projet en ce sens :



- ligne 22 : l'objet [PamMetier] instancié par Spring a une propriété publique [PamDao] qui est une référence sur la couche [dao]. Cette propriété est initialisée avec la référence de la couche [dao] créée ligne 20.

L'exécution faite avec la base de données décrite au paragraphe 6.2, page 63, donne le résultat console suivant :

```
1.  [[260124402111742,Laverti,Justine,La Brûlerie,St Marcel,49014,[1, 1,93, 2, 3, 12]],
    [3,49,6,15,9,39,7,88],[1, 1,93, 2, 3, 12],[64,85 : 17,45 : 10 : 15 : 72,4 ]
```

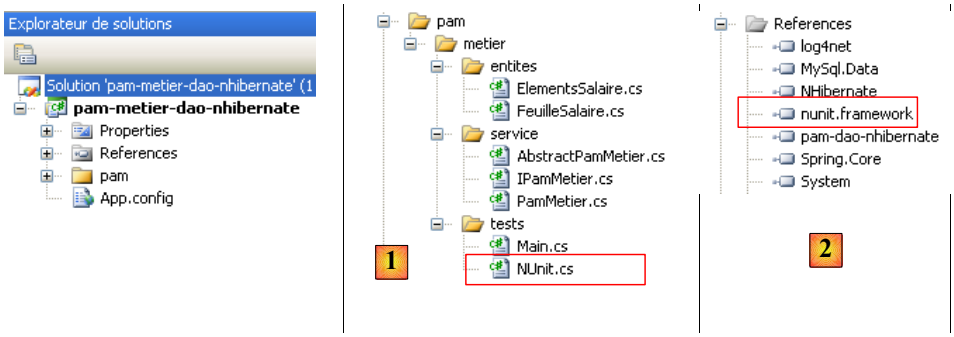
```
2.  [[254104940426058,Jouveinal,Marie,5 rue des oiseaux,St Corentin,49203,[2, 2,1, 2,1, 3,1, 15]],
    [3,49,6,15,9,39,7,88],[2, 2,1, 2,1, 3,1, 15],[362,25 : 97,48 : 42: 62 : 368,77 ]
3.  PamException : L'employé de n° ss [xx] n'existe pas
```

- lignes 1-2 : les 2 feuilles de salaire demandées
- ligne 3 : l'exception de type [PamException] provoquée par un employé inexistant.

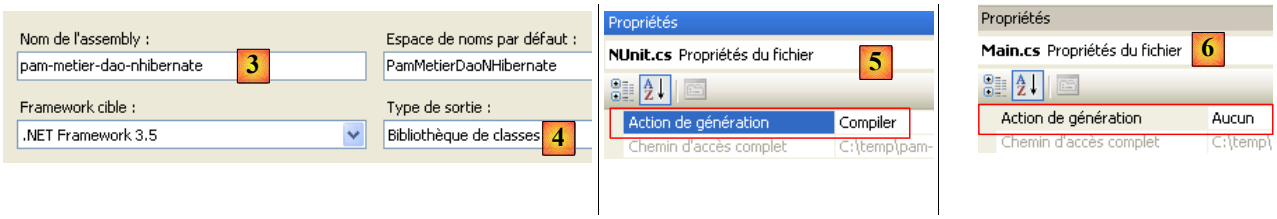
7.4.6 Tests unitaires de la couche métier

Le test précédent était visuel : on vérifiait à l'écran qu'on obtenait bien les résultats attendus. Nous passons maintenant aux tests non visuels NUnit.

Revenons au projet Visual Studio du projet [metier] :



- en [1], le programme de test NUnit
- en [2], la référence sur la DLL [nunit.framework]



- en [3,4], la génération du projet va produire la DLL [pam-metier-dao-nhibernate.dll].
- en [5], le fichier [NUnit.cs] sera inclus dans l'assembly [pam-metier-dao-nhibernate.dll] mais pas [Main.cs] [6]

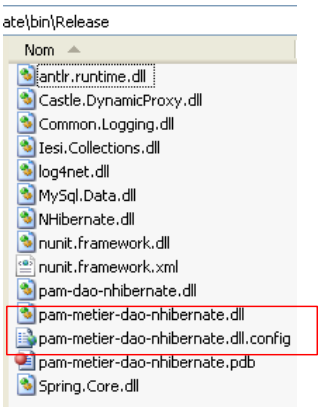
La classe de test NUnit est la suivante :

```
1.  using NUnit.Framework;
2.  using Pam.Dao.Entites;
3.  using Pam.Metier.Entites;
4.  using Pam.Metier.Service;
5.  using Spring.Context.Support;
6.
7.  namespace Pam.Metier.Tests {
8.
9.   [TestFixture()]
10.  public class NunitTestPamMetier : AssertionHelper {
11.
12.     // la couche [metier] à tester
13.     private IPamMetier pamMetier;
14.
15.     // constructeur
16.     public NunitTestPamMetier() {
17.        // instanciation couche [dao]
18.        pamMetier = ContextRegistry.GetContext().GetObject("pammetier") as IPamMetier;
19.     }
20.
```

```
21.
22.     [Test]
23.     public void GetAllIdentitesEmployes() {
24.        // vérification nbre d'employes
25.        Expect(2, EqualTo(pamMetier.GetAllIdentitesEmployes().Length));
26.     }
27.
28.     [Test]
29.     public void GetSalaire1() {
30.        // calcul d'une feuille de salaire
31.        FeuilleSalaire feuilleSalaire = pamMetier.GetSalaire("254104940426058", 150, 20);
32.        // vérifications
33.        Expect(368.77, EqualTo(feuilleSalaire.ElementsSalaire.SalaireNet).Within(1E-06));
34.        // feuille de salaire d'un employé inexistant
35.        bool erreur = false;
36.        try {
37.           feuilleSalaire = pamMetier.GetSalaire("xx", 150, 20);
38.        } catch (PamException) {
39.           erreur = true;
40.        }
41.        Expect(erreur, True);
42.     }
43.
44.  }
45. }
```

- ligne 13 : le champ privé [pamMetier] est une instance de l'interface d'accès à la couche [metier]. On notera que le type de ce champ est une **interface** et non une **classe**. Cela signifie que l'instance [PamMetier] ne rend accessibles que des méthodes, celles de l'interface [IPamMetier].
- lignes 16-19 : le constructeur de la classe initialise le champ privé [pamMetier] à l'aide de Spring et du fichier de configuration [App.config].
- lignes 23-26 : testent la méthode [GetAllIdentitesEmployes]
- lignes 29-42 : testent la méthode [GetSalaire]

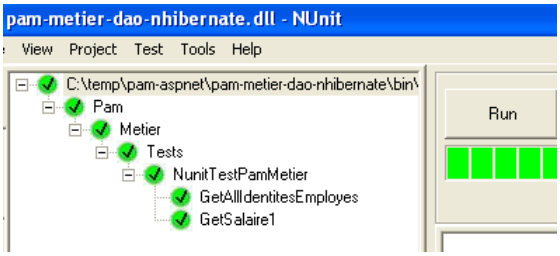
Le projet ci-dessus génère la DLL [pam-metier.dll] dans le dossier [bin/Release].



Le dossier [bin/Release] contient en outre :

- les DLL qui font partie des références du projet et qui ont l'attribut [Copie locale] à vrai : [Spring.Core, MySql.data, NHibernate, log4net, pam-dao-nhibernate]. Ces DLL sont accompagnées des copies des DLL qu'elles utilisent elles-mêmes :
  - [CastleDynamicProxy, Iesi.Collections] pour l'outil NHibernate
  - [antlr.runtime, Common.Logging] pour l'outil Spring
- le fichier [pam-metier-dao-nhibernate.dll.config] est une copie du fichier de configuration [App.config].

On charge la DLL [pam-metier-dao-nhibernate.dll] avec l'outil [NUnit-Gui, version 2.4.6] et on exécute les tests :



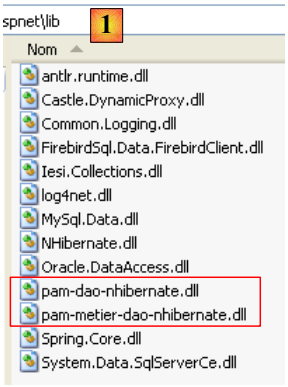
Ci-dessus, les tests ont été réussis.

Travail pratique :

- mettre en oeuvre sur machine les tests de la classe [PamMetier].
- utiliser différents fichiers de configuration App.config afin d'utiliser des SGBD différents (Firebird, MySQL, Postgres, SQL Server)

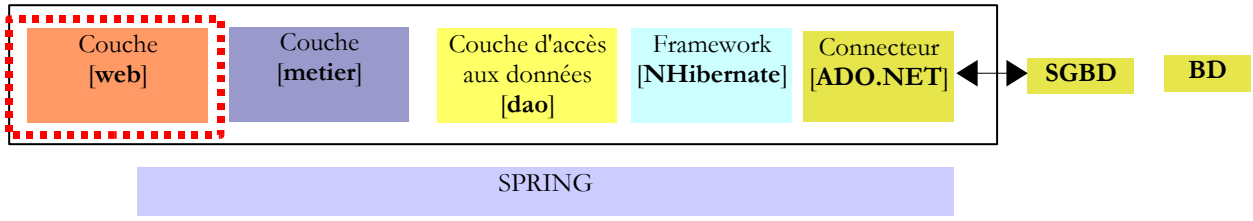
7.4.7 Génération de la DLL de la couche [metier]

Une fois écrite et testée la classe [PamMetier], on génèrera la DLL [pam-metier-dao-nhibernate.dll] de la couche [metier] en suivant la méthode décrite au paragraphe 7.3.5, page 100. On prendra soin de ne pas inclure dans la DLL les programmes de test [Main.cs] et [NUnit.cs]. On la placera ensuite dans le dossier [lib] des DLL [1].



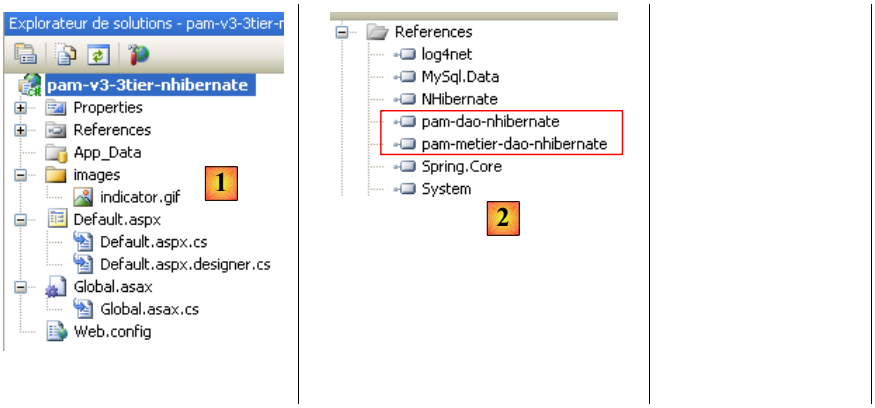
7.5 La couche [web]

Revenons sur l'architecture générale de l'application [SimuPaie] :



Nous considérons que les couche [dao] et [métier] sont acquises et encapsulées dans les DLL [pam-dao-nhibernate, pam-metier-dao-nhibernate.dll]. Nous décrivons mainatenant la couche web.

7.5.1 Le projet Visual Web Developer de la couche [web]



- en [1], le projet dans son ensemble :
  - [Global.asax] : la classe instanciée au démarrage de l'application web et qui assure l'initialisation de l'application
  - [Default.aspx] : la page du formulaire web
- en [2], les DLL nécessaires à l'application web. On notera les DLL des couches [dao] et [metier] construites précédemment.

7.5.2 Configuration de l'application

Le fichier [Web.config] qui configure l'application définit les mêmes données que le fichier [App.config] configurant la couche [metier] étudiée précédemment. Celles-ci doivent prendre place dans le code pré-généré du fichier [Web.config] :

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.
3.  <configuration>
4.
5.    <configSections>
6.      <sectionGroup name="system.web.extensions" type="System.Web.Configuration.SystemWebExtensionsSectionGroup,
    System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35">
7.  ..........
8.      </sectionGroup>
9.      <sectionGroup name="spring">
10.       <section name="context" type="Spring.Context.Support.ContextHandler, Spring.Core" />
11.       <section name="objects" type="Spring.Context.Support.DefaultSectionHandler, Spring.Core" />
12.     </sectionGroup>
13.     <section name="hibernate-configuration" type="NHibernate.Cfg.ConfigurationSectionHandler, NHibernate" />
14.     <section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler,log4net" />
15.   </configSections>
16.
17.   <!-- configuration Spring -->
18.   <spring>
19.     <context>
20.       <resource uri="config://spring/objects" />
21.     </context>
22.     <objects xmlns="http://www.springframework.net">
23.       <object id="pamdao" type="Pam.Dao.Service.PamDaoNHibernate, pam-dao-nhibernate" init-method="init"
    destroy-method="destroy"/>
24.       <object id="pammetier" type="Pam.Metier.Service.PamMetier, pam-metier-dao-nhibernate" >
25.         <property name="PamDao" ref="pamdao"/>
26.       </object>
27.     </objects>
28.   </spring>
29.
30.   <!-- configuration NHibernate -->
31.   <hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">
32.     <session-factory>
33.       <property name="connection.provider">NHibernate.Connection.DriverConnectionProvider</property>
34.       <!--
35.           <property name="connection.driver_class">NHibernate.Driver.MySqlDataDriver</property>
36.           -->
37.       <property name="dialect">NHibernate.Dialect.MySQLDialect</property>
38.       <property name="connection.connection_string">
39.         Server=localhost;Database=dbpam_nhibernate;Uid=root;Pwd=;
40.       </property>
41.       <property name="show_sql">false</property>
```

```
42.       <mapping assembly="pam-dao-nhibernate"/>
43.     </session-factory>
44.   </hibernate-configuration>
45.
46.   <!-- This section contains the log4net configuration settings -->
47.   <!-- NOTE IMPORTANTE : les logs ne sont pas actifs par défaut. Il faut les activer par programme
48.    avec l'instruction log4net.Config.XmlConfigurator.Configure();
49.    ! -->
50.   <log4net>
51. ....
52.   </log4net>
53.
54.   <appSettings/>
55.   <connectionStrings/>
56.
57.   <system.web>
58. ....
59. ....
60.
61. </configuration>
```

On retrouve lignes 9-12, 18-28 et 31-44 la configuration Spring et NHibernate décrite dans le fichier [App.config] de la couche [metier] (cf page 107).

Global.asax.cs

```
1.  using System;
2.  using Pam.Dao.Entites;
3.  using Pam.Metier.Service;
4.  using Spring.Context.Support;
5.
6.  namespace pam_v3
7.  {
8.    public class Global : System.Web.HttpApplication
9.    {
10.     // --- données statiques de l'application ---
11.     public static Employe[] Employes;
12.     public static IPamMetier PamMetier = null;
13.     public static string Msg;
14.     public static bool Erreur = false;
15.
16.     // démarrage de l'application
17.     public void Application_Start(object sender, EventArgs e)
18.     {
19.       // exploitation du fichier de configuration
20.       try
21.       {
22.         // instanciation couche [metier]
23.         PamMetier = ContextRegistry.GetContext().GetObject("pammetier") as IPamMetier;
24.         // liste simplifiée des employés
25.         Employes = PamMetier.GetAllIdentitesEmployes();
26.         // on a réussi
27.         Msg = "Base chargée...";
28.       }
29.       catch (Exception ex)
30.       {
31.         // on note l'erreur
32.         Msg = string.Format("L'erreur suivante s'est produite lors de l'accès à la base de données
    : {0}", ex);
33.         Erreur = true;
34.       }
35.     }
36.   }
37. }
```

On rappelle que :

- la classe [Global.asax.cs] est instanciée au démarrage de l'application et que cette instance est accessible à toutes les requêtes de tous les utilisateurs. Les champs statiques des lignes 11-14 sont ainsi partagés entre tous les utilisateurs.
- que la méthode [Application_Start] est exécutée une unique fois après l'instanciation de la classe. C'est la méthode où est faite en général l'initialisation de l'application.

Les données partagées par tous les utilisateurs sont les suivantes :

- ligne 11 : le tableau d'objets de type [Employe] qui mémorisera la liste simplifiée (SS, NOM, PRENOM) de tous les employés
- ligne 12 : une référence sur la couche [metier] encapsulée dans la DLL [pam-metier-dao-nhibernate.dll]

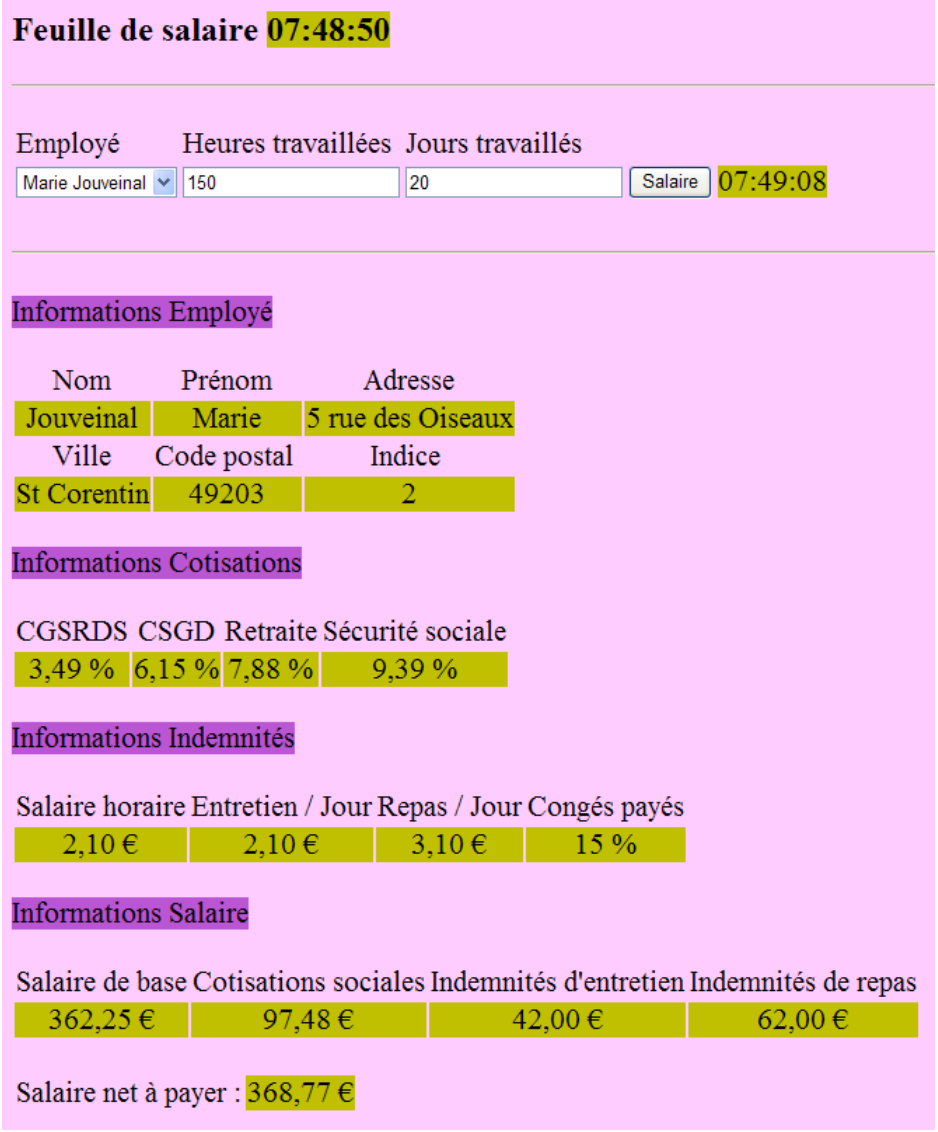
- ligne 13 : un message indiquant comment s'est terminé l'initialisation (bien ou avec erreur)
- ligne 14 : un booléen indiquant si l'initialisation s'est terminée par une erreur ou non.

Dans [Application_Start] :

- ligne 23 : Spring instancie les couches [metier] et [dao] et rend une référence sur la couche [metier]. Celle-ci est mémorisée dans le champ statique [PamMetier] de la ligne 12.
- ligne 25 : le tableau des employés est demandé à la couche [metier]
- ligne 27 : le message en cas de réussite
- ligne 32 : le message en cas d'erreur

7.5.3 Le formulaire [Default.aspx]

Le formulaire est celui de la version 2.



Question : En vous inspirant du code C# de la page [Default.aspx.cs] de la version 2, écrire le code [Default.aspx.cs] de la version 3. L'unique différence est dans le calcul du salaire. Alors que dans la version 2, on utilisait l'API ADO.NET pour récupérer des informations dans la base de données, ici on utilisera la méthode *GetSalaire* de la couche [metier].

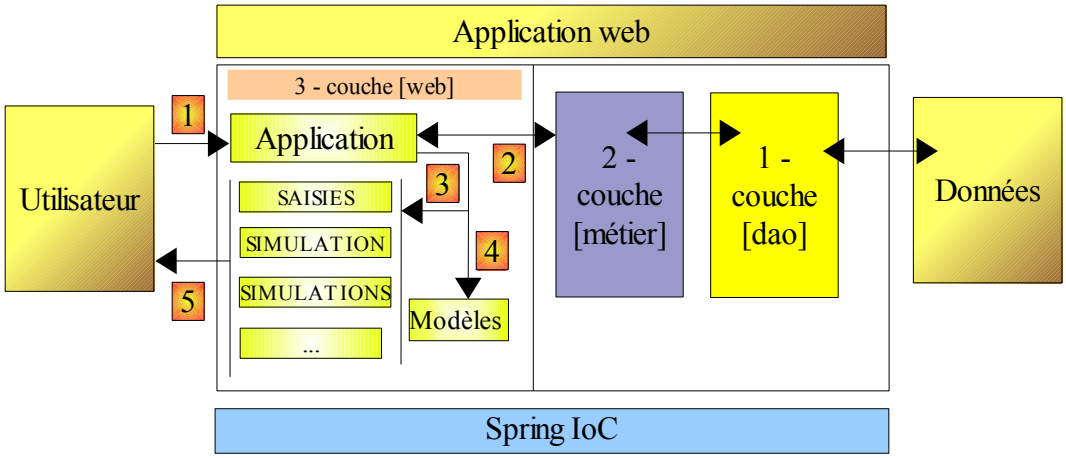
Travail pratique :

- mettre en oeuvre sur machine l'application web précédente
- utiliser différents fichiers de configuration [Web.config] afin d'utiliser des SGBD différents (Firebird, MySQL, Postgres, SQL Server)

8 L'application [SimuPaie] – version 4 – ASP.NET / multi-vues / mono-page

Lectures conseillées : référence [1], **programmation ASP.NET vol2**, paragraphes :
- 1.1.3 : Composants serveur et contrôleur d'application
- 1.1.4 : Exemples d'applications MVC avec composants serveurs ASP

Nous étudions maintenant une version dérivée de l'application ASP.NET à trois couches étudiée précédemment et qui lui ajoute de nouvelles fonctionnalités. L'architecture de notre application évolue de la façon suivante :



Le traitement d'une demande d'un client se déroule selon les étapes suivantes :

1. le client fait une demande à l'application.
2. l'application traite cette demande. Pour ce faire, elle peut avoir besoin de l'aide de la couche [métier] qui elle-même peut avoir besoin de la couche [dao] si des données doivent être échangées avec la base de données. L'application reçoit une réponse de la couche [métier].
3. selon celle-ci, elle choisit (3) la vue (= la réponse) à envoyer au client en lui fournissant (4) les informations (le modèle) dont elle a besoin.
4. la réponse est envoyée au client (5)

On a ici une architecture web dite MVC (Modèle – Vue – Contrôleur) :

- [Application] est le **contrôleur**. Il voit passer toutes les requêtes du client.
- [Saisies, Simulation, Simulations, ...] sont les **vues**. Une vue en .NET est du code ASP / HTML standard qui contient des composants qu'il convient d'initialiser. Les valeurs qu'il faut fournir à ces composants forment le **modèle** de la vue.

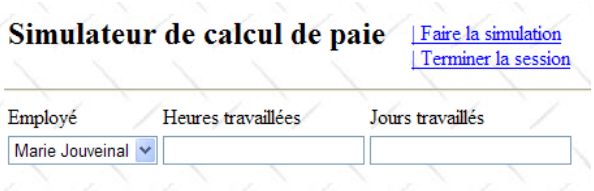
Nous allons ici implémenter le modèle (design pattern) MVC de la façon suivante :
- les vues seront des composants [View] au sein d'une page unique [Default.aspx]
- le contrôleur est alors le code [Default.aspx.cs] de cette page unique.

Seules des applications basiques peuvent supporter cette implémentation MVC. En effet, à chaque requête, tous les composants de la page [Default.aspx] sont instanciés, donc toutes les vues. Au moment d'envoyer la réponse, l'une d'elles est choisie par le code de contrôle de l'application simplement en rendant visible le composant [View] correspondant et en cachant les autres. Si l'application a de nombreuses vues, la page [Default.aspx] aura de nombreux composants et son coût d'instanciation peut devenir prohibitif. Par ailleurs, le mode [Design] de la page risque de devenir ingérable parce qu'ayant trop de vues. Ce type d'architecture convient pour des applications avec peu de vues et développées par une unique personne. Lorsqu'elle peut être adoptée, elle permet de développer une architecture MVC très simplement. C'est ce que nous allons voir dans cette nouvelle version.

8.1 Les vues de l'application

Les différentes vues présentées à l'utilisateur seront les suivantes :

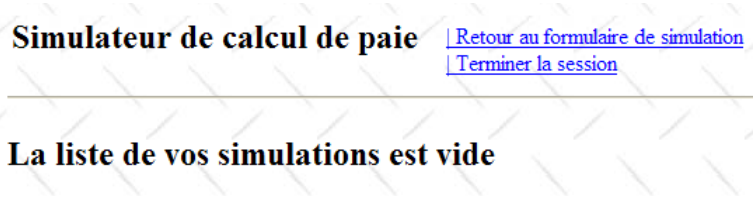
- la vue [VueSaisies] qui présente le formulaire de simulation



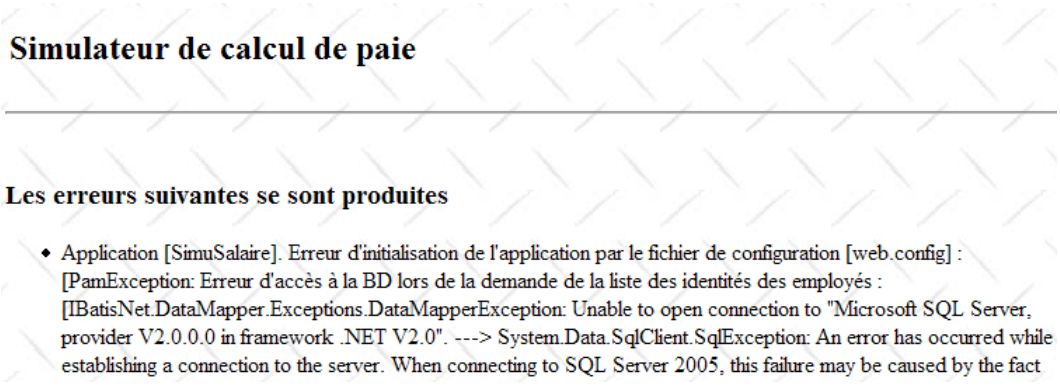
- la vue [VueSimulation] utilisée pour afficher le résultat détaillé de la simulation :

- la vue [VueSimulations] qui donne la liste des simulations faites par le client

- la vue [VueSimulationsVides] qui indique que le client n'a pas ou plus de simulations :

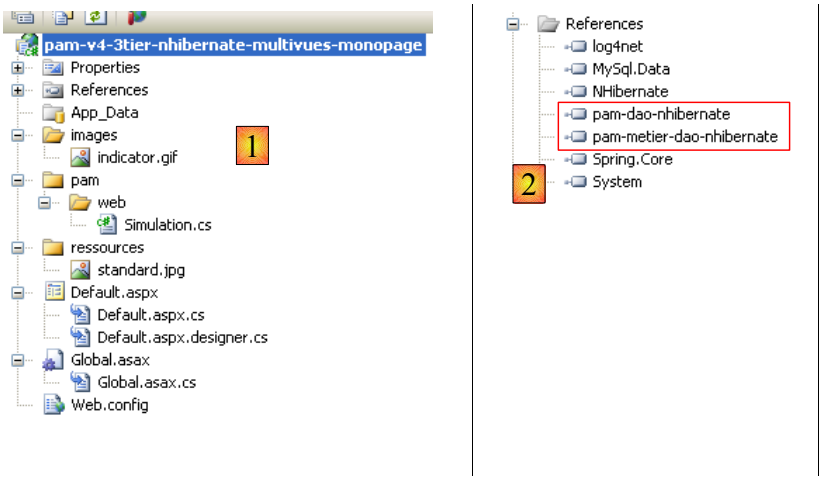


– la vue [VueErreurs] qui indique une ou plusieurs erreurs :



8.2 Le projet Visual Web Developer de la couche [web]

Le projet Visual Web Developer de la couche [web] est le suivant :



- en [1] on trouve :
  - le fichier de configuration [Web.config] de l'application – est identique à celui de l'application précédente.
  - le fichier [Global.cs] qui gère les événements de l'application web, ici son démarrage - est identique à celui de l'application précédente si ce n'est qu'il gère également le démarrage de la session utilisateur.
  - le formulaire [Default.aspx] de l'application – contient les différentes vues de l'application.

- en [2] on trouve les références du projet – sont identiques à celles de la version précédente

8.3 Le fichier [Global.cs]

Le fichier [Global.cs] qui gère les événements de l'application web, est identique à celui de l'application précédente si ce n'est qu'il gère en plus le démarrage de la session utilisateur :

Global.cs

```
1.  using System;
2.  using System.Web;
3.  using Pam.Dao.Entites;
4.  using Pam.Metier.Service;
5.  using Spring.Context.Support;
6.  using System.Collections.Generic;
7.  using istia.st.pam.web;
8.
9.  namespace pam_v4
10. {
11.     public class Global : HttpApplication
12.     {
13.
14.         // --- données statiques de l'application ---
15.         public static Employe[] Employes;
16.         public static string Msg = string.Empty;
17.         public static bool Erreur = false;
18.         public static IPamMetier PamMetier = null;
19.
20.         // démarrage de l'application
21.         public void Application_Start(object sender, EventArgs e)
22.         {
23. ...
24.         }
25.
26.         // démarrage de la session d'un utilisateur
27.         public void Session_Start(object sender, EventArgs e)
28.         {
29.             // on met une liste de simulations vide dans la session
30.             List<Simulation> simulations = new List<Simulation>();
31.             Session["simulations"] = simulations;
32.         }
33.
34.     }
35. }
```

- lignes 27-34 : on gère le démarrage de la session. Nous mettrons dans celle-ci la liste des simulations faites par l'utilisateur.
- ligne 30 : une liste de simulations vide est créée. Une simulation est un objet de type [Simulation] que nous allons détailler prochainement.
- ligne 31 : la liste de simulations est placée dans la session associée à la clé " simulations "

8.4 La classe [Simulation]

Un objet de type [Simulation] sert à encapsuler une ligne du tableau des simulations :

Son code est le suivant :

```
namespace Pam.Web
{

    public class Simulation
    {
        // données d'une simulation
        public string Nom { get; set; }
        public string Prenom { get; set; }
        public double HeuresTravaillees { get; set; }
        public int JoursTravailles { get; set; }
        public double SalaireBase { get; set; }
        public double Indemnites { get; set; }
        public double CotisationsSociales { get; set; }
        public double SalaireNet { get; set; }

        // constructeurs
        public Simulation()
        {

        }

        public Simulation(string nom, string prenom, double heuresTravailllees, int joursTravailles, double salaireBase, double indemnites, double cotisationsSociales, double salaireNet)
        {
            {
                this.Nom = nom;
                this.Prenom = prenom;
                this.HeuresTravaillees = heuresTravailllees;
                this.JoursTravailles = joursTravailles;
                this.SalaireBase = salaireBase;
                this.Indemnites = indemnites;
                this.CotisationsSociales = cotisationsSociales;
                this.SalaireNet = salaireNet;
            }
        }
    }
}
```

Les champs de la classe correspondent aux colonnes du tableau des simulations.

8.5 La page [Default.aspx]

8.5.1 Vue d'ensemble

La page [Default.aspx] contient plusieurs composants [View], un pour chaque vue. Son squelette est le suivant :

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="pam_v4.PagePam" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>Simulateur de paie</title>
</head>
<body background="ressources/standard.jpg">
    <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server" EnablePartialRendering="true" />
    <asp:UpdatePanel runat="server" ID="UpdatePanelPam" UpdateMode="Conditional">
        <ContentTemplate>
            <table>
                <tr>
                    <td>
                        <h2>
                            Simulateur de calcul de paie</h2>
                    </td>
                    <td>
                        <label>
                               </label>
                        <asp:UpdateProgress ID="UpdateProgress1" runat="server">
                            <ProgressTemplate>
                                <img src="images/indicator.gif" />
                                <asp:Label ID="Label5" runat="server" BackColor="#FF8000"
                                 EnableViewState="False" Text="Calcul en cours. Patientez ....">
                                </asp:Label>
                            </ProgressTemplate>
                        </asp:UpdateProgress>
```

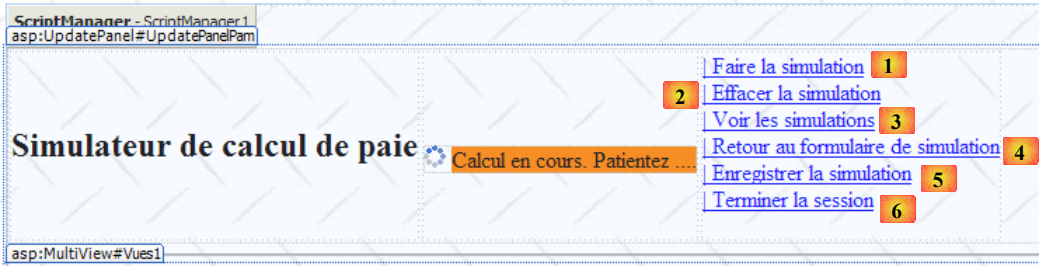
```
30.                     </td>
31.                     <td>
32.                        <asp:LinkButton ID="LinkButtonFaireSimulation" runat="server"
33.                         CausesValidation="False" OnClick="LinkButtonFaireSimulation_Click">
34.                             | Faire la simulation<br /></asp:LinkButton>
35.                        <asp:LinkButton ID="LinkButtonEffacerSimulation" runat="server"
36.                                 CausesValidation="False" OnClick="LinkButtonEffacerSimulation_Click">
37.                             | Effacer la simulation<br /></asp:LinkButton>
38.                        <asp:LinkButton ID="LinkButtonVoirSimulations" runat="server"
39.                         CausesValidation="False" OnClick="LinkButtonVoirSimulations_Click">
40.                             | Voir les simulations<br /></asp:LinkButton>
41.                         <asp:LinkButton ID="LinkButtonFormulaireSimulation" runat="server"
42.                         CausesValidation="False" OnClick="LinkButtonFormulaireSimulation_Click">
43.                         | Retour au formulaire de simulation<br /></asp:LinkButton>
44.                         <asp:LinkButton ID="LinkButtonEnregistrerSimulation" runat="server"
45.                         CausesValidation="False" OnClick="LinkButtonEnregistrerSimulation_Click">
46.                             | Enregistrer la simulation<br /></asp:LinkButton>
47.                        <asp:LinkButton ID="LinkButtonTerminerSession" runat="server"
48.                         CausesValidation="False" OnClick="LinkButtonTerminerSession_Click">
49.                             | Terminer la session<br /></asp:LinkButton>
50.                     </td>
51.             </table>
52.             <hr />
53.             <asp:MultiView ID="Vues1" ActiveViewIndex="0" runat="server">
54.                 <asp:View ID="VueSaisies" runat="server">
55. ...
56.                 </asp:View>
57.             </asp:MultiView>
58.             <asp:MultiView ID="Vues2" runat="server">
59.                 <asp:View ID="VueSimulation" runat="server">
60. ...
61.                 </asp:View>
62.                 <asp:View ID="VueSimulations" runat="server">
63. ...
64.                 </asp:View>
65.                 <asp:View ID="VueSimulationsVides" runat="server">
66. ...
67.                 </asp:View>
68.                 <asp:View ID="VueErreurs" runat="server">
69. ...
70.                 </asp:View>
71.             </asp:MultiView>
72.         </ContentTemplate>
73.     </asp:UpdatePanel>
74.     </form>
75. </body>
76. </html>
```

- ligne 10 : la balise pour disposer des extensions Ajax
- lignes 11-73 : le conteneur *UpdatePanel* mis à jour par des appels Ajax
- lignes 12-72 : le contenu du conteneur *UpdatePanel*
- lignes 13-52 : l'entête qui sera présent dans chaque vue. Il présente à l'utilisateur la liste des actions possibles sous la forme d'une liste de liens.
- ligne 33 : on notera l'attribut *CausesValidation="False"* qui fait que les validateurs de la page ne seront pas exécutés **implicitement** lorsque le lien sera cliqué. Lorsque cet attribut est absent, sa valeur par défaut est *True*. La validation de la page pourra être faite **explicitement** dans le code exécuté côté serveur par l'opération *Page.Validate*.
- lignes 53-57 : un composant [MultiView] avec une unique vue [VueSaisies]. Pour cette raison, on a mis en dur, ligne 53, le n° de la vue à afficher : *ActiveViewIndex="0"*
- lignes 53-67 : un composant [MultiView] avec quatre vues : la vue [VueSimulation] lignes 59-61, la vue [VueSimulations] lignes 62-64, la vue [VueSimulationsVides] lignes 65-67, la vue [VueErreurs] lignes 68-70. Un composant [MultiView] n'affiche qu'une vue à la fois. Pour afficher la vue n° i du composant *Vues2*, on écrira le code :

```
Vues2.ActiveViewIndex=i
```

8.5.2 L'entête

L'entête est formé des composants suivants :



N°	Type	Nom	Rôle
1	LinkButton	LinkButtonFaireSimulation	demande le calcul de la simulation
2	LinkButton	LinkButtonEffacerSimulation	efface le formulaire de saisie
3	LinkButton	LinkButtonVoirSimulations	affiche la liste des simulations déjà faites
4	LinkButton	LinkButtonFormulaireSimulation	ramène au formulaire de saisie
5	LinkButton	LinkButtonEnregistrerSimulation	enregistre la simulation courante dans la liste des simulations
6	LinkButton	LinkButtonTerminerSession	abandonne la session courante

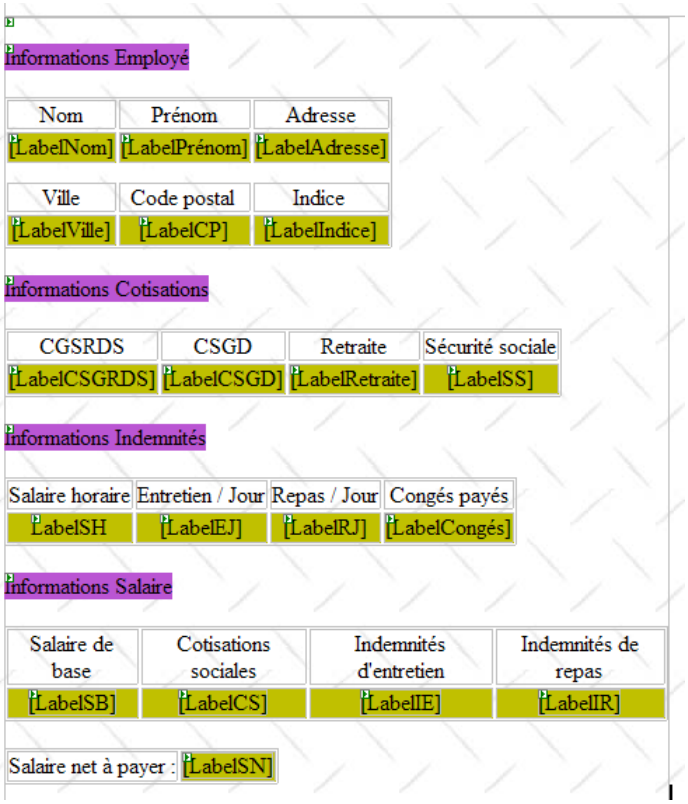
8.5.3 La vue [Saisies]

Le composant [View] nommé [VueSaisies] est le suivant :

N°	Type	Nom	Rôle
1	DropDownList	ComboBoxEmployes	Contient la liste des noms des employés
2	TextBox	TextBoxHeures	Nombre d'heures travaillées – nombre réel
3	TextBox	TextBoxJours	Nombre de jours travaillés – nombre entier
4	RequiredFieldValidator	RequiredFieldValidatorHeures	vérifie que le champ [2] [TextBoxHeures] n'est pas vide
5	RegularExpressionValidator	RegularExpressionValidatorHeures	vérifie que le champ [2] [TextBoxHeures] est un nombre réel >=0
6	RequiredFieldValidator	RequiredFieldValidatorJours	vérifie que le champ [3] [TextBoxJours] n'est pas vide
7	RegularExpressionValidator	RegularExpressionValidatorJours	vérifie que le champ [3] [TextBoxJours] est un nombre entier >=0

8.5.4 La vue [Simulation]

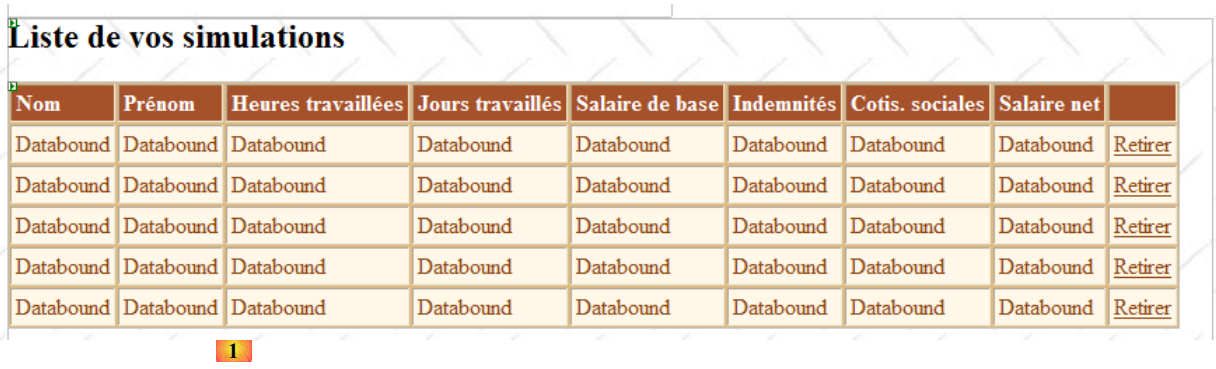
Le composant [View] nommé [VueSimulation] est le suivant :



Il n'est composé que de composants [Label] dont les ID sont indiqués ci-dessus.

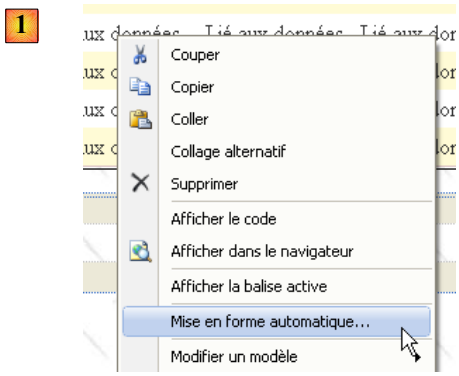
8.5.5 La vue [Simulations]

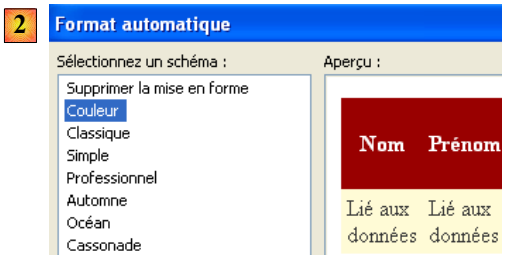
Le composant [View] nommé [VueSimulations] est le suivant :



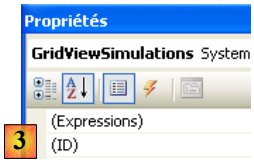
N°	Type	Nom	Rôle
1	GridView	GridViewSimulations	Contient la liste des simulations

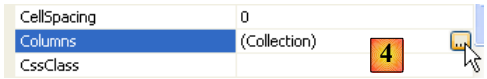
Les propriétés du composant [GridViewSimulations] ont été définies de la façon suivante :

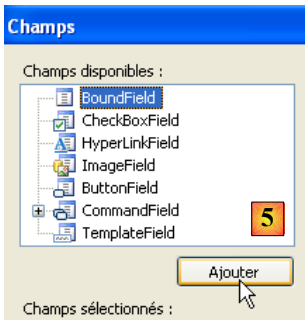




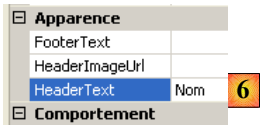
- en [1] : clic droit sur le [GridView] / option [Mise en forme automatique]
- en [2] : choisir un type d'affichage pour le [GridView]







- en [3] : sélectionner les propriétés du [GridView]
- en [4] : éditer les colonnes du [GridView]
- en [5] : ajouter une colonne de type [BoundField] qui sera liée (bound) à l'une des propriétés publiques de l'objet à afficher dans la ligne du [GridView]. L'objet affiché ici, sera un objet de type [Simulation].



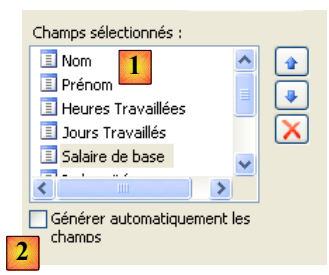
- en [6] : donner le titre de la colonne
- en [7] : donner le nom de la propriété de la classe [Simulation] qui sera associée à cette colonne.
- [DataFormatString] indique comment doivent être formatées les valeurs affichées dans la colonne.

Les colonnes du composant [GridViewSimulations] ont les propriétés suivantes :

N°	Propriétés
2	Type : **BoundField**, HeaderText : **Nom**, DataField : **Nom**
3	Type : **BoundField**, HeaderText : **Prénom**, DataField : **Prenom**
4	Type : **BoundField**, HeaderText : **Heures travaillées**, DataField : **HeuresTravaillees**
5	Type : **BoundField**, HeaderText : **Jours travaillés**, DataField : **JoursTravailles**
6	Type : **BoundField**, HeaderText : **Salaire de base**, DataField : **SalaireBase**, DataFormatString : **{0:C}** (format monétaire, C=Currency) – affichera le sigle de l'euro.
7	Type : **BoundField**, HeaderText : **Indemnités**, Data Field : **Indemnites**, DataFormatString : **{0:C}**
8	Type : **BoundField**, HeaderText : **Cotis. sociales**, DataField : **CotisationsSociales**, DataFormatString : **{0:C}**

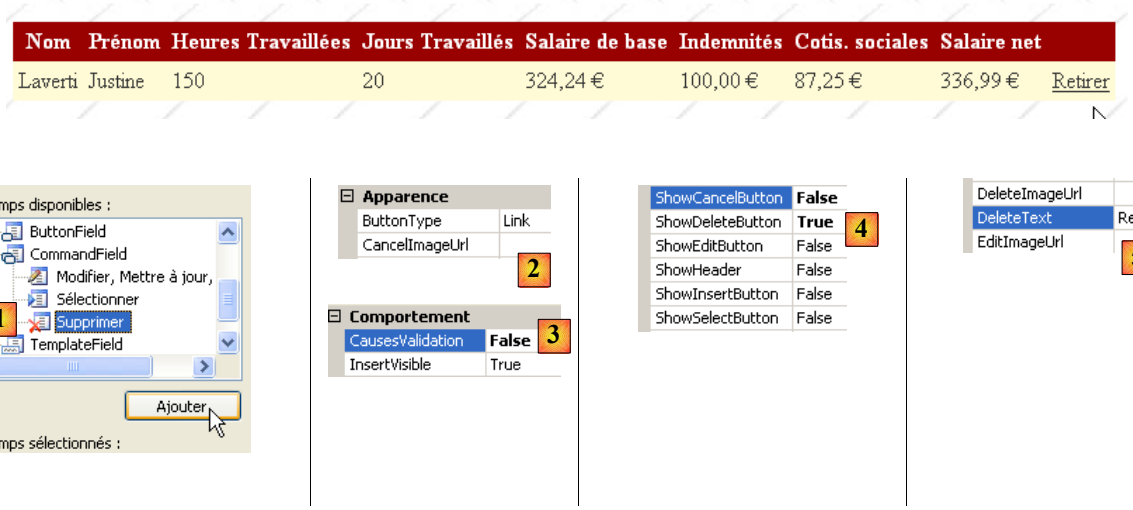
N°	Propriétés
9	Type : **BoundField**, HeaderText : **Salaire net**, DataField : **SalaireNet**, DataFormatString : **{0:C}**

On prêtera attention au fait que le champ [DataField] doit correspondre à une **propriété existante** de la classe [Simulation]. A l'issue de cette phase, toutes les colonnes de type [BoundField] ont été créées :



- en [1] : les colonnes créées pour le [GridView]
- en [2] : la génération automatique des colonnes doit être inhibée lorsque c'est le développeur qui les définit lui-même comme nous venons de le faire.

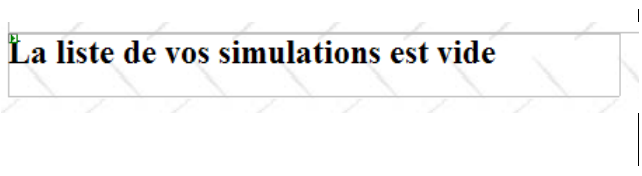
Il nous reste à créer la colonne des liens [Retirer] :



- en [1] : ajouter une colonne de type [CommandField / Supprimer]
- en [2] : *ButtonType=Link* pour avoir un lien dans la colonne plutôt qu'un bouton
- en [3] : *CausesValidation=False*, un clic sur le lien ne provoquera pas l'exécution des contrôles de validation qui peuvent se trouver sur la page. En effet, la suppression d'une simulation ne nécessite aucune vérification de données.
- en [4] : seul le lien de suppression sera visible.
- en [5] : le texte de ce lien

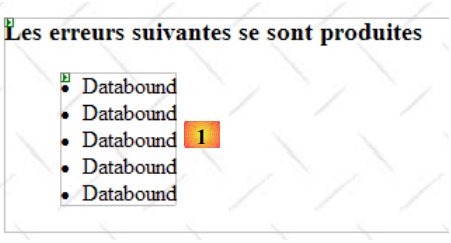
8.5.6 La vue [SimulationsVides]

Le composant [View] nommé [VueSimulationsVides] contient simplement du texte :



8.5.7 La vue [Erreurs]

Le composant [View] nommé [VueErreurs] est le suivant :



N°	Type	Nom	Rôle
1	Repeater	RptErreurs	affiche une liste de messages d'erreur

Le composant [Repeater] permet de répéter un code ASP.NET / HTML pour chaque objet d'une source de données, généralement une collection. Ce code est défini directement dans le code source ASP.NET de la page :

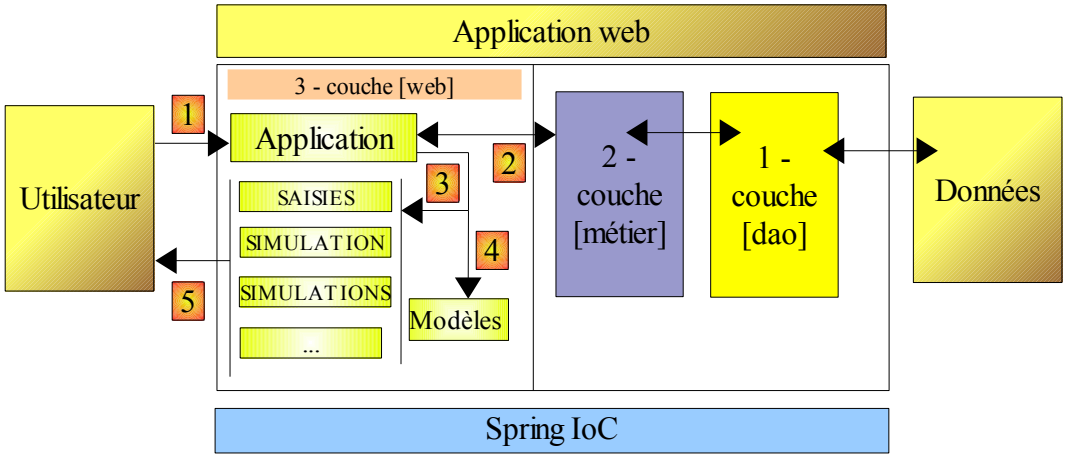
```
1.                          <asp:Repeater ID="RptErreurs" runat="server">
2.                                  <ItemTemplate>
3.                                          <li>
4.                                                  <%# Container.DataItem %>
5.                                          </li>
6.                                  </ItemTemplate>
7.  </asp:Repeater>
```

- ligne 2 : <*ItemTemplate*> définit le code qui sera répété pour chaque élément de la source de données.
- ligne 4 : affiche la valeur de l'expression *Container.DataItem* qui désigne l'élément courant de la source de données. Cet élément étant un objet, c'est la méthode *ToString* de cet objet qui est utilisée pour inclure celui-ci dans le flux HTML de la page. Notre collection d'objets sera une collection *List(Of String)* contenant des messages d'erreur. Les lignes 3-5 incluront des séquences <*li*>*Message*</*li*> dans le flux HTML de la page.

8.6 Le contrôleur [Default.aspx.cs]

8.6.1 Vue d'ensemble

Revenons à l'architecture MVC de l'application :



- [Default.aspx.cs] qui est le code de contrôle de la page unique [Default.aspx] est le contrôleur de l'application.
- [Global] est l'objet de type [HttpApplication] qui initialise l'application et qui dispose d'une référence sur la couche [métier].

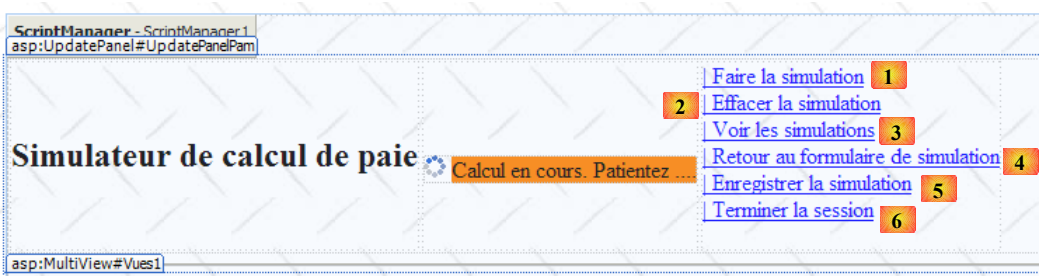
Le squelette du code du contrôleur [Default.aspx.cs] est le suivant :

```
1.  using System.Collections.Generic;
2.  ...
3.
4.  public partial class PagePam : Page
5.  {
6.
7.      private void setVues(bool boolVues1, bool boolVues2, int index)
8.      {
9.          // on affiche les vues demandées
10.         // boolVues1 : true si le multivues Vues1 doit être visible
11.         // boolVues1 : true si le multivues Vues2 doit être visible
12.         // index : index de la vue de Vues2 à afficher
13. ...
14.     }
15.
16.     private void setMenu(bool boolFaireSimulation, bool boolEnregistrerSimulation, bool
    boolEffacerSimulation, bool boolFormulaireSimulation, bool boolVoirSimulations, bool
    boolTerminerSession)
17.     {
18.         // on fixe les options de menu
19.         // chaque booléen est affecté à la propriété Visible du lien correspondant
20. ...
21.     }
22.
23.     // chargement de la page
24.     protected void Page_Load(object sender, System.EventArgs e)
25.     {
26.         // traitement requête initiale
27.         if (!IsPostBack)
28.         {
29. ...
30.         }
31.     }
32.
33.     protected void LinkButtonFaireSimulation_Click(object sender, System.EventArgs e)
34.     {
35. ...
36.     }
37.
38.     protected void LinkButtonEffacerSimulation_Click(object sender, System.EventArgs e)
39.     {
40. ....
41.     }
42.
43.     protected void LinkButtonVoirSimulations_Click(object sender, System.EventArgs e)
44.     {
45. ...
46.     }
47.
48.     protected void LinkButtonEnregistrerSimulation_Click(object sender, System.EventArgs e)
49.     {
50. ...
51.     }
52.
53.     protected void LinkButtonTerminerSession_Click(object sender, System.EventArgs e)
54.     {
55. ...
56.     }
57.
58.     protected void LinkButtonFormulaireSimulation_Click(object sender, System.EventArgs e)
59.     {
60. ...
61.     }
62.
63.
64.     protected void GridViewSimulations_RowDeleting(object sender, GridViewDeleteEventArgs e)
65.     {
66. ...
```

```
67.     }
68. }
```

A la requête initiale (GET) de l'utilisateur, seul l'événement *Load* des lignes 24-31 est traité. Aux requêtes suivantes (POST) faites via les liens du menu, deux événements sont traités :

1. l'événement *Load* (24-31) mais le test du booléen *Page.IsPostback* (ligne 27) fait que rien ne sera fait.
2. l'événement lié au lien qui a été cliqué :



- lignes 33-36 : traitent le clic sur le lien [1]
- lignes 38-41 : traitent le clic sur le lien [2]
- lignes 43-46 : traitent le clic sur le lien [3]
- lignes 58-61 : traitent le clic sur le lien [4]
- lignes 48-51 : traitent le clic sur le lien [5]
- lignes 53-56 : traitent le clic sur le lien [6]

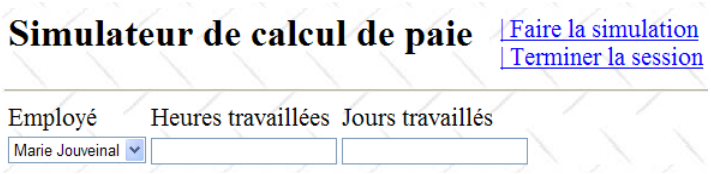
Pour factoriser des séquences de code revenant souvent, deux méthodes internes ont été créées :

- **setVues**, lignes 7-14 : fixe la ou les vues à afficher
- **setMenu**, lignes 16-21 : fixe les options de menu à afficher

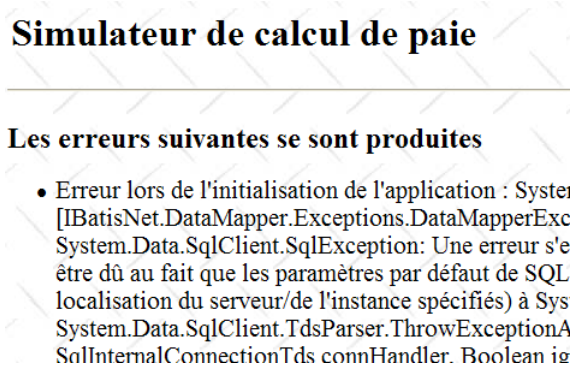
8.6.2 L'événement Load

Lectures conseillées : référence [1], **programmation ASP.NET** vol2 :
- paragraphe 2.8 : Composant [Repeater] et liaison de données.

La première vue présentée à l'utilisateur est celle du formulaire vide :



L'initialisation de l'application nécessite l'accès à une source de données qui peut échouer. Dans ce cas, la première page est une page d'erreurs :



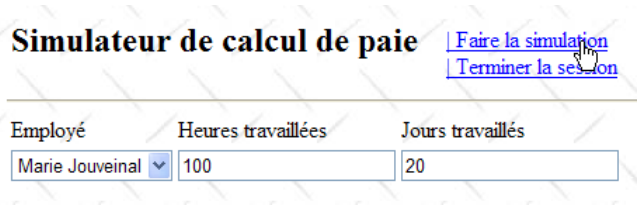
L'événement [Load] est traité de façon analogue à celle des versions ASP.NET précédentes :

```
1.      // chargement de la page
2.      protected void Page_Load(object sender, System.EventArgs e)
3.      {
4.          // traitement requête initiale
5.          if (!IsPostBack)
6.          {
7.              // des erreurs d'initialisation ?
8.              if (Global.Erreur)
9.              {
10.                 // affichage vue [erreurs]
11. ...
12.                 // positionnement menu
13.                 ...
14.                 return;
15.             }
16.             // chargement des noms des employés dans le combo
17. ...
18.             // positionnement menu
19. ...
20.             // affichage vue [saisie]
21.  ...
22.         }
23. }
```

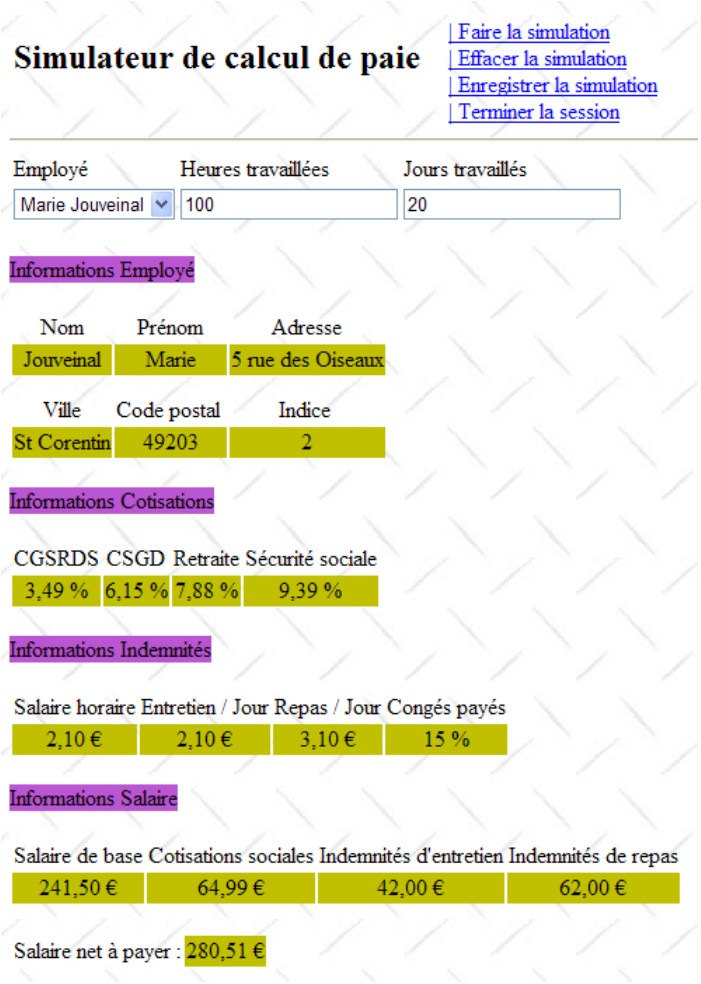
Question : compléter le code ci-dessus

8.6.3 Action : Faire la simulation

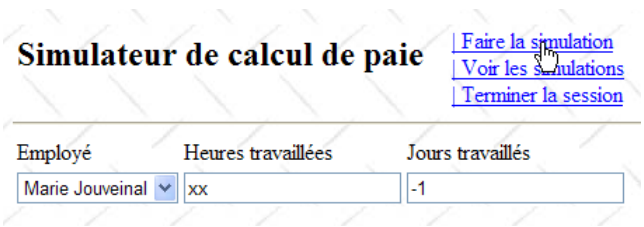
Dans ce qui suit, l'écran noté (1) est celui de la demande de l'utilisateur, l'écran noté (2) la réponse qui lui est envoyée par l'application web. A partir de l'écran d'accueil, l'utilisateur peut commencer une simulation :



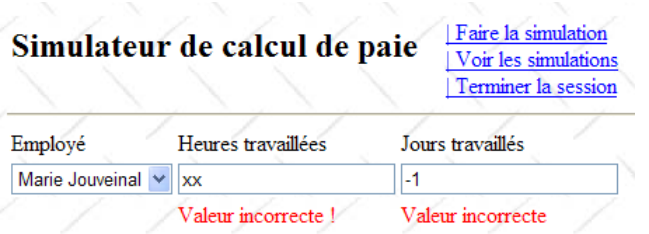
- (1) : l'utilisateur demande une simulation



- (2) : résultat de la simulation



- (1) : on entre des données erronées et on demande la simulation



- (2) : les erreurs ont été signalées

La procédure qui traite cette action pourrait ressembler à ce qui suit :

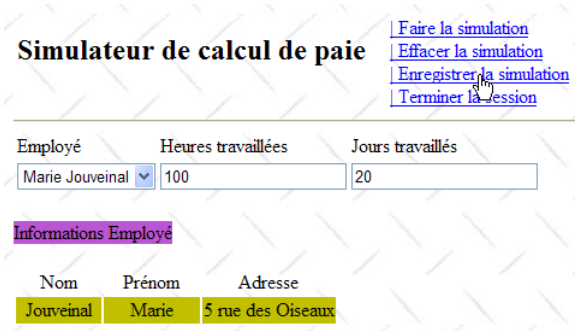
```
1.      protected void LinkButtonFaireSimulation_Click(object sender, System.EventArgs e)
2.      {
3.          // calcul du salaire
4.          // page valide ?
5.          Page.Validate();
6.          if (!Page.IsValid)
7.          {
8.              // affichage vue [saisie]
9.              ...
10.             return;
11.         }
```

```
12.         // la page est valide - on récupère les saisies
13.         double HeuresTravaillées = ...;
14.         int JoursTravaillés = ...;
15.         // on calcule le salaire de l'employé
16.         FeuilleSalaire feuillesalaire;
17.         try
18.         {
19.             feuillesalaire = ...
20.         }
21.         catch (PamException ex)
22.         {
23.             // affichage vue [erreurs]
24. ...
25.             return;
26.         }
27.         // on met le résultat dans la session
28.         Session["simulation"] = ...
29.         // affichage résultats
30. ...
31.         // affichage vues [saisie, employe, salaire]
32.         ...
33.         // affichage menu
34.         ...
35. }
```

Question : compléter le code ci-dessus

8.6.4 Action : Enregistrer la simulation

Une fois la simulation faite, l'utilisateur peut demander son enregistrement :



- (1) : demande d'enregistrement de la simulation courante

- (2) : la simulation est enregistrée et la liste des simulations faites est présentée

La procédure qui traite cette action pourrait ressembler à ce qui suit :

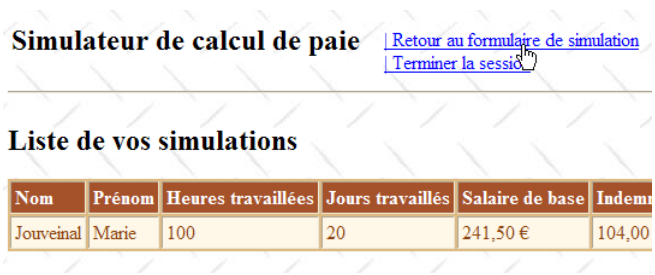
```
1.      protected void LinkButtonEnregistrerSimulation_Click(object sender, System.EventArgs e)
2.      {
3.          // on enregistre la simulation courante dans la liste des simulations présente dans la
    session
4.          ...
5.          // on affiche la vue [simulations]
6.          ...
7.  }
```

Question : compléter le code ci-dessus

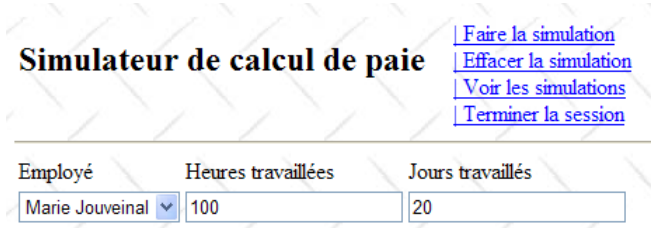
8.6.5 Action : Retour au formulaire de simulation

Lectures conseillées : référence [1], **programmation ASP.NET** vol2 :
- paragraphe 1.6.3 : Le rôle du champ caché _VIEWSTATE

Une fois la liste des simulations présentée, l'utilisateur peut demander à revenir au formulaire de simulation :



- (1) : retour au formulaire de simulation



- (2) : le formulaire de simulation tel qu'il a été saisi initialement

On notera que l'écran (2) présente le formulaire tel qu'il a été saisi. Il faut se rappeler ici que ces différentes vues appartiennent à une même page. Entre les différentes requêtes, les valeurs des composants sont maintenues par le mécanisme du *ViewState* si ces composants ont leur propriété *EnableViewState* à *true*.

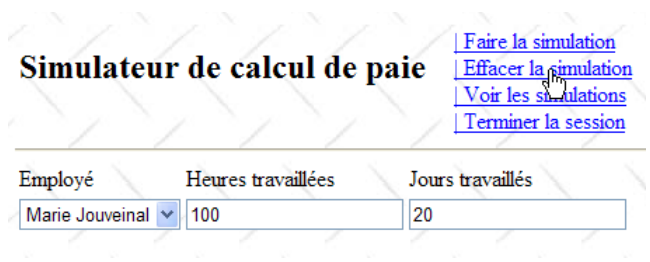
La procédure qui traite cette action pourrait ressembler à ce qui suit :

```
1.      protected void LinkButtonFormulaireSimulation_Click(object sender, System.EventArgs e)
2.      {
3.          // affichage vue [saisie]
4.          ...
5.          // positionnement menu
6.          ...
7.  }
```

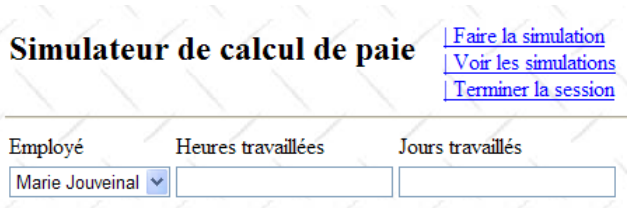
Question : compléter le code ci-dessus

8.6.6 Action : Effacer la simulation

Une fois revenu au formulaire de simulation, l'utilisateur peut demander à effacer les saisies présentes :



- (1) : on efface les données du formulaire



- (2) : le formulaire a été réinitialisé

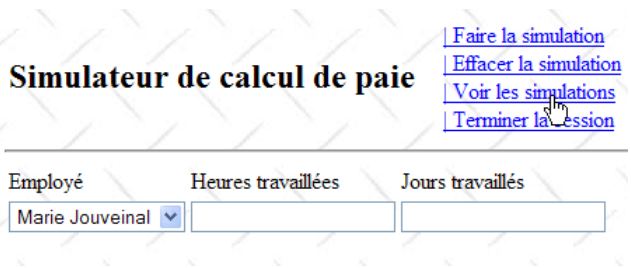
La procédure qui traite cette action pourrait ressembler à ce qui suit :

```
1.      protected void LinkButtonEffacerSimulation_Click(object sender, System.EventArgs e)
2.      {
3.          // RAZ du formulaire
4.          ...
5.  }
```

Question : compléter le code ci-dessus

8.6.7 Action : Voir les simulations

L'utilisateur peut demander à voir les simulations qu'il a déjà faites :



- (1) : on demande à voir les simulations

- (2) : la liste des simulations

La procédure qui traite cette action pourrait ressembler à ce qui suit :

```
1.  protected void LinkButtonVoirSimulations_Click(object sender, System.EventArgs e)
2.      {
3.          // on récupère les simulations dans la session
4.          ...
5.          // y-a-t-il des simulations ?
6.          if (...)
7.          {
8.              // vue [simulations] visible
9.  ...
10.         }
11.         else
12.         {
13.             // vue [SimulationsVides]
14.             ...
15.         }
16.         // on fixe le menu
17.         ...
18.     }
```

Question : compléter le code ci-dessus

8.6.8 Action : Supprimer une simulation

L'utilisateur peut demander à supprimer une simulation :

- (1) : on peut retirer des simulations de la liste

- (2) : la simulation a été retirée

La procédure [GridViewSimulations_RowDeleting] qui traite cette action pourrait ressembler à ce qui suit :

```
1.      protected void GridViewSimulations_RowDeleting(object sender, GridViewDeleteEventArgs e)
2.      {
```

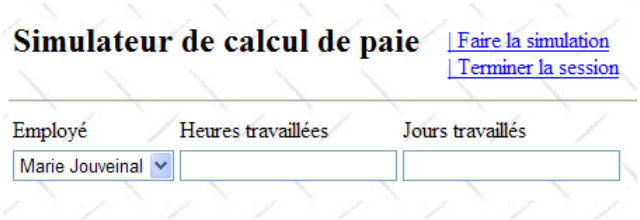
```
3.          // on récupère les simulations dans la session
4.          ...
5.          // on supprime la simulation désignée (e.RowIndex est le n° de la ligne supprimée)
6.          ...
7.          // reste-t-il des simulations ?
8.          if (...)
9.          {
10.             // on remplit le GridView
11.  ...
12.         }
13.         else
14.         {
15.             // vue [SimulationsVides]
16.             ...
17.         }
18. }
```

Question : compléter le code ci-dessus

8.6.9 Action : Terminer la session

L'utilisateur peut demander à terminer sa session de simulations. Cela abandonne le contenu de sa session et présente un formulaire vide :

- (1) : on invalide la session courante



- (2) : on revient à la page d'accueil

La procédure qui traite cette action pourrait ressembler à ce qui suit :

```
1.      protected void LinkButtonTerminerSession_Click(object sender, System.EventArgs e)
2.      {
3.          // on abandonne la session
4.          ...
5.          // afficher la vue [saisies]
6.          ...
7.          // positionnement menu
8.          ...
9.  }
```

Question : compléter le code ci-dessus

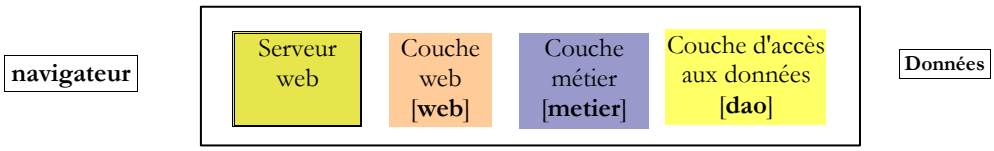
Travail pratique : mettre en oeuvre sur machine l'application web précédente. Ajoutez-lui un comportement Ajax.

9 L'application [SimuPaie] – version 5 – ASP.NET / service web

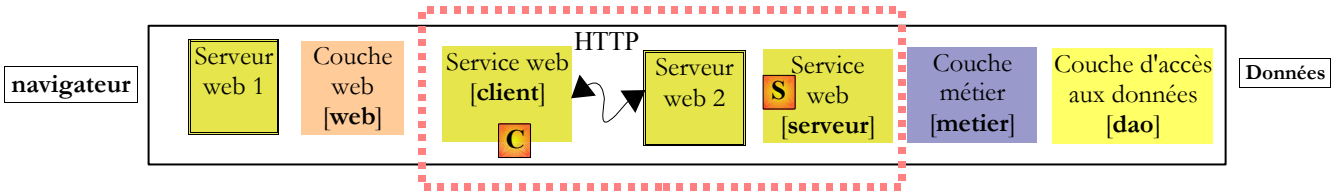
Lectures conseillées : référence [2], **Introduction à C# 2008**, chapitre 10 **"Services Web"**

9.1 La nouvelle architecture de l'application

L'architecture en couches de l'application *Pam* est actuellement la suivante :



Nous allons la faire évoluer comme suit :



Alors que dans l'architecture précédente, les couches [web], [metier], [dao] s'exécutaient dans une même machine virtuelle .NET, dans la nouvelle architecture, la couche [web] s'exécutera dans une autre machine virtuelle que les couches [metier] et [dao]. Ce sera le cas notamment si la couche [web] est sur une machine M1 et les couches [metier] et [dao] sur une machine M2. On a ici une architecture client / serveur :

- le serveur est constitué des couches [metier] et [dao]. Parce que c'est un service web, il a besoin du serveur web n° 2 pour s'exécuter.
- le client est constitué de la couche [web]. Pour s'exécuter, il a besoin du serveur web n° 1.
- le client et le serveur communiquent par le réseau Tcp / Ip avec le protocole HTTP / SOAP. Pour cela, deux nouvelles couches doivent être ajoutées à l'architecture :
  - la couche [S] qui sera un **service web**. Le service web reçoit les requêtes des clients distants et utilise les couches [metier] et [dao] pour les satisfaire. Il y a de nombreuses façons de construire un service Tcp / Ip. L'avantage du **service web** est double :
    - il utilise le protocole HTTP que laissent passer les pare-feux des entreprises et administrations
    - il utilise un sous-protocole HTTP / SOAP standard, implémenté par de nombreuses plate-formes de développement : .Net, Java, Php, Flex, ... Ainsi un service web peut être "consommé" (c'est le terme usuel) par des clients .Net, Java, Php, Flex, ...
  - la couche [C] qui sera le **client** du service web distant. Elle aura pour rôle de communiquer avec le service web [S].

Cette nouvelle architecture peut être dérivée des précédentes sans trop d'efforts :
- les couches [metier] et [dao] restent inchangées
- la couche [web] évolue légèrement, essentiellement pour référencer des entités telles *Employe*, *FeuilleSalaire* qui sont devenues des entités de la couche client [C]. Ces entités sont analogues à celles des couches [metier] ou [dao] mais elles appartiennent à des espace de noms différents.
- la couche serveur [S] est une classe qui implémente l'interface *IPamMetier* de la couche [metier]. Cette implémentation se contente de faire appel aux méthodes correspondantes de la couche [metier]. Les méthodes implémentées par la couche serveur [S] vont être "exposées" aux clients distants qui vont pouvoir les appeler.
- la couche client [C] sera générée par Visual Studio.

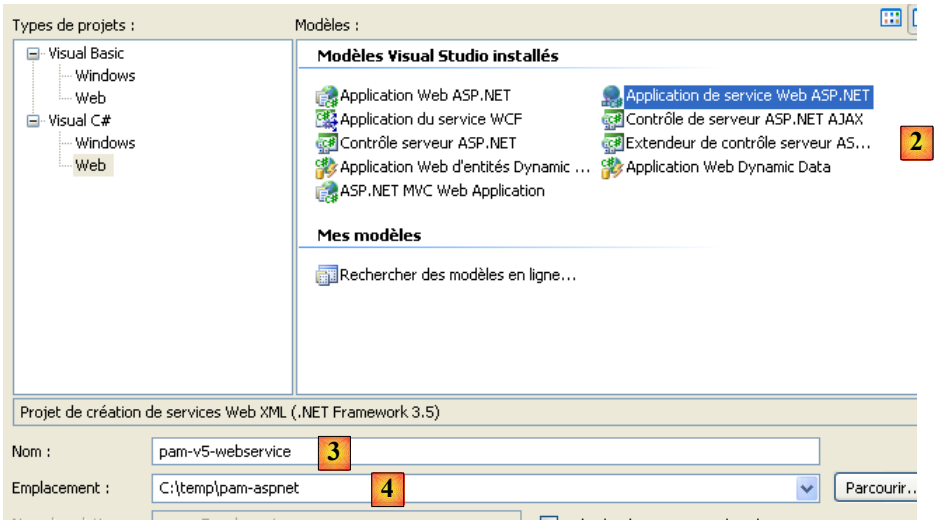
Les principes de la nouvelle architecture sont les suivants :
- la couche [web] continue à communiquer avec la couche [metier] comme si celle-ci était locale. Pour cela, la couche client [C] implémente l'interface *IPamMetier* de la couche [metier] réelle et se présente à la couche [web] comme une couche [metier] locale. En-dehors du problème des espaces de noms évoqué précédemment, la couche [web] ne change pas. C'est

l'avantage d'avoir travaillé en couches. Si on avait construit une application mono-couche, il faudrait la remanier très profondément.

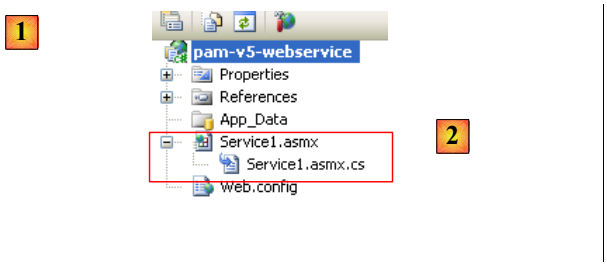
- la couche client [C] transmet, de façon transparente pour la couche [web], les requêtes de celle-ci au service web distant [S]. Elle prend en charge tout l'aspect "communication réseau". Elle reçoit une réponse du service web distant qu'elle met en forme pour la rendre à la couche [web] sous la forme que celle-ci attend.
- côté serveur, le service web [S] reçoit des commandes de ses clients distants. Il les met en forme afin d'appeler les méthodes de l'interface *IPamMetier* de la couche [metier]. Lorsqu'il a reçu la réponse de la couche [metier], il met en forme celle-ci pour la transmettre via le réseau au client [C]. Les couches [metier] et [dao] n'ont pas à être modifiées.

9.2 Le projet Visual Web Developer du service web

Nous construisons un nouveau projet avec Visual Web Developer :



- en [1], nous choisissons un projet web en C#
- en [2], nous choisissons "Application de service Web ASP.NET"
- en [3], nous donnons un nom au projet web
- en [4], nous indiquons un emplacement pour ce projet



- en [1] le projet généré. C'est un projet web classique aux détails suivants près :
  - on a indiqué que le projet était de type "service web". Un service web n'envoie pas des pages web HTML à ses clients mais des données au format XML. Aussi la page [Default.aspx] habituellement générée ne l'a pas été.
- en [2], un fichier [Service1.asmx] a été généré avec le contenu suivant :

```
<%@ WebService Language="C#" CodeBehind="Service1.asmx.cs" Class="pam_v5_webservice.Service1" %>
```

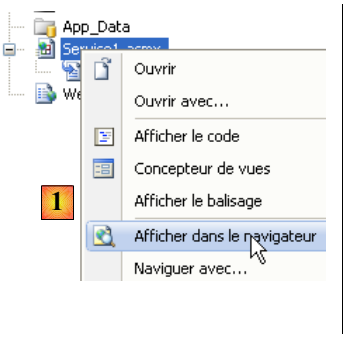
- la balise *WebService* indique que [Service.asmx] est un service web
- l'attribut *CodeBehind* indique l'emplacement du code source de ce service web
- l'attribut *Class* indique le nom de la classe implémentant le service web dans le code source

Le code source [Service.asmx.cs] du service web généré par défaut est le suivant :

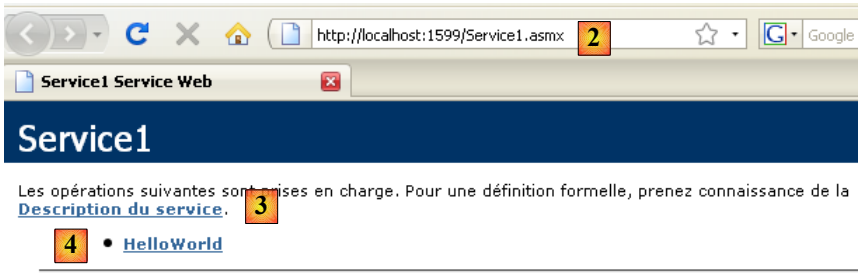
```
1.  using System.Web.Services;
```

```
2.
3.  namespace pam_v5_webservice
4.  {
5.    /// <summary>
6.    /// Description résumée de Service1
7.    /// </summary>
8.    [WebService(Namespace = "http://tempuri.org/")]
9.    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
10.   [System.ComponentModel.ToolboxItem(false)]
11.   // Pour autoriser l'appel de ce service Web depuis un script à l'aide d'ASP.NET AJAX, supprimez
    les marques de commentaire de la ligne suivante.
12.   // [System.Web.Script.Services.ScriptService]
13.   public class Service1 : System.Web.Services.WebService
14.   {
15.
16.     [WebMethod]
17.     public string HelloWorld()
18.     {
19.       return "Hello World";
20.     }
21.   }
22. }
```

- ligne 8 : l'annotation *WebService* qui fait que la classe *Service1* de la ligne 13 va être exposée comme un service web. Un service web appartient à un espace de noms afin d'éviter que deux services web dans le monde portent le même nom. Nous serons amenés à changer cet espace de noms ultérieurement.
- ligne 13 : la classe *Service1* dérive de la classe *WebService* du framework .NET.
- ligne 16 : l'annotation *WebMethod* fait que la méthode ainsi annotée va être exposée aux clients distants qui pourront ainsi l'appeler.
- lignes 17-20 : la méthode *HelloWorld* est une méthode de démonstration. Nous la supprimerons plus tard. Elle nous permet de faire les premiers tests et de découvrir les outils de Visual Studio ainsi que certains éléments à connaître à propos des services web.

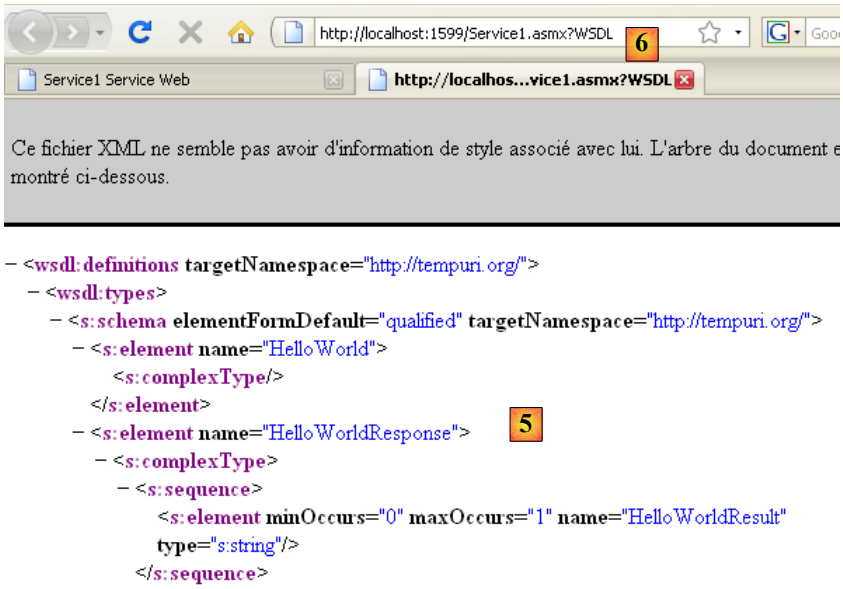


- en [1], nous exécutons le service web [Service.asmx]

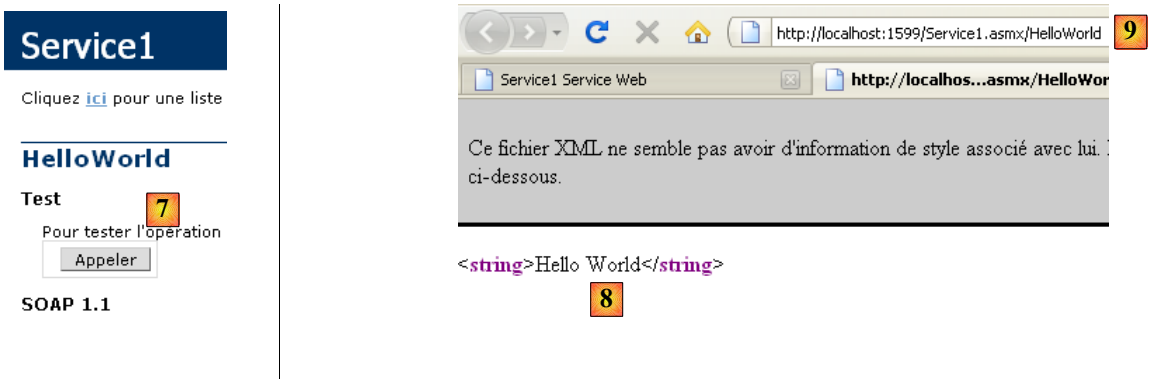


- VS Web Developer a lancé son serveur web intégré et le fait écouter sur un port aléatoire, ici 1599. L'Url [2] a été ensuite demandée au serveur web. C'est celle d'une page de test du service web.
- en [3], un lien qui permet de visualiser le fichier de description du service web. Ce fichier appelé fichier WSDL (**W**eb**S**ervice **D**escription **L**anguage) à cause de son suffixe (.wsdl) est un fichier XML décrivant les méthodes exposées par le service web. C'est à partir de ce fichier WSDL que les clients peuvent connaître :
  - l'espace de noms du service web
  - la liste des méthodes exposées par le service web
  - les paramètres attendus par chacune d'elles

- la réponse renvoyée par chacune d'elles
- en [4], l'unique méthode exposée par le service web.



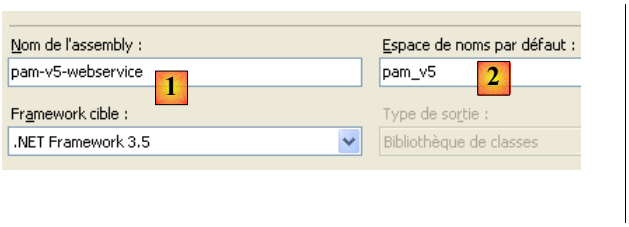
- en [5], le contenu du fichier WSDL obtenu via le lien [3]. On notera l'Url [6]. Sa connaissance est nécessaire aux clients du service web.



- en [7], la page obtenue en suivant le lien [4] permet d'appeler la méthode [HelloWorld] du service web
- en [8], le résultat obtenu : une réponse XML. On notera l'Url [9] de la méthode.

L'étude des pages précédentes permet de comprendre comment est appelée une méthode d'un service web et quel type de réponse elle renvoie. Cela permet d'écrire des clients HTTP capables de dialoguer avec le service web. La plupart des IDE actuels permet la génération automatique de ce client HTTP évitant ainsi au développeur de l'écrire. C'est le cas notamment de Visual Studio Express.

Avant de continuer dans ce projet, nous allons changer l'espace de noms utilisé par défaut lors de la génération des classes :



Lorsque nous sélectionnons les propriétés du projet (clic droit sur projet / Propriétés), nous avons en [1] le nom de *l'assembly* du projet et en [2] son espace de noms par défaut.

Ceci fait,

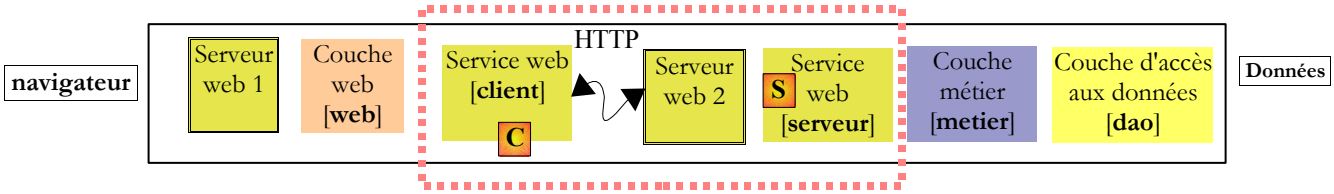
- dans [Service1.asmx.cs], nous changeons l'espace de noms de la classe :

```
1.  using System.Web.Services;
2.
3.  namespace pam_v5
4.  {
5.  ...
6.    public class Service1 : System.Web.Services.WebService
7.    {
8.  ...
9.    }
10. }
```

- dans [Service.asmx], nous changeons également l'espace de noms utilisé pour la classe [Service1] (clic droit / afficher le balisage) :

```
<%@ WebService Language="C#" CodeBehind="Service1.asmx.cs" Class="pam_v5.Service1" %>
```

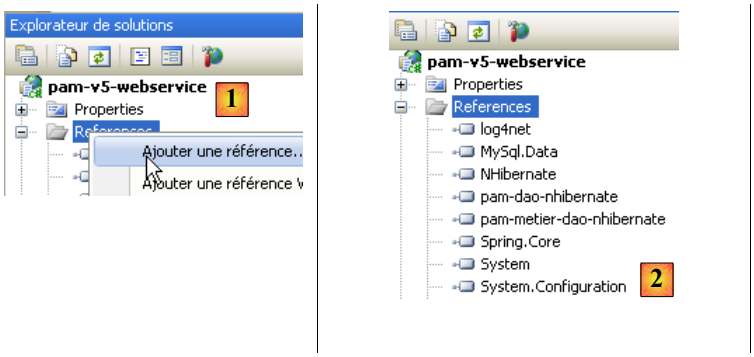
Revenons à l'architecture de notre application :



- la couche [S] est le service web. Elle se contente d'exposer les méthodes de la couche [metier] à des clients distants. C'est cette couche que nous sommes en train de construire.
- la couche [C] est le client HTTP du service web. C'est cette couche que les IDE savent générer automatiquement.
- la couche [web] voit la couche [C] comme une couche [metier] locale si on fait en sorte que la couche [C] implémente l'interface de la couche [metier] distante.

Nous voyons ci-dessous que notre service web va :
- exposer les méthodes de la couche [metier]
- dialoguer avec cette dernière qui elle-même va dialoguer avec la couche [dao].

Le projet doit donc utiliser les DLL des couches [metier] et [dao]. Il évolue de la façon suivante :

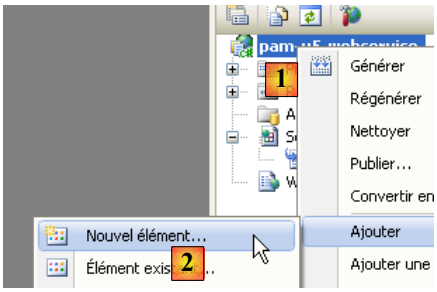


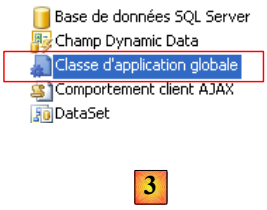
- en [1], on ajoute des références au projet
- en [2], on sélectionne les DLL habituelles du dossier [lib]. On prendra soin qu'elles aient toutes leur propriété "Copie locale" à *True*. Les Dll sélectionnées sont celles qui implémentent les couches [metier] et [dao] avec un support NHibernate.

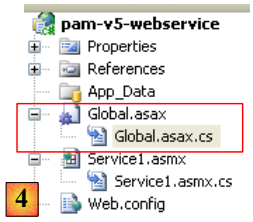
Une application web de type "service Web ASP.NET" peut avoir une classe d'application globale "Global.asax" comme une application "site Web ASP.NET" classique. Nous avons vu l'intérêt d'une telle classe :

- elle est instanciée au démarrage de l'application et reste en mémoire

- elle peut ainsi mémoriser des données partagées par tous les clients et qui sont en lecture seule. Dans notre application elle mémorisera comme dans les précédentes, la liste simplifiée des employés. Cela évitera d'aller chercher cette liste dans la base de données lorsqu'un client la demandera.







- en [1], cliquer droit sur le projet
- en [2], choisir l'option [Ajouter un nouvel élément]
- en [3], choisir [Classe d'application globale]
- en [4], le fichier [Global.asax] a été ajouté au projet

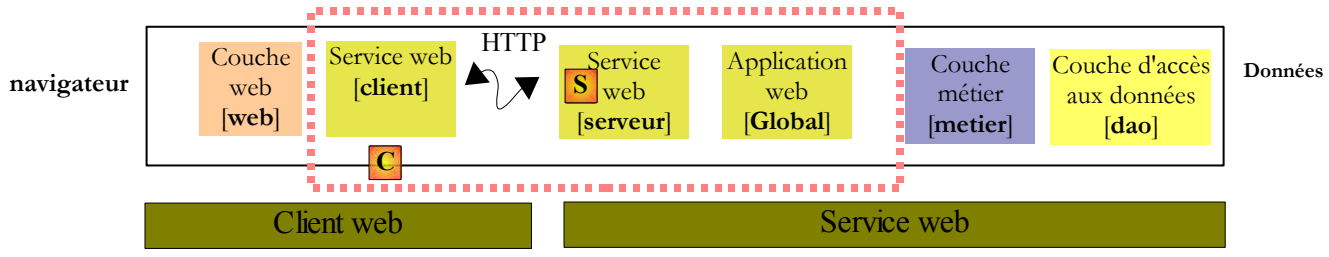
Le contenu du fichier [Global.asax] est le suivant :

```
<%@ Application Codebehind="Global.asax.cs" Inherits="pam_v5.Global" Language="C#" %>
```

Le contenu du fichier [Global.asax.cs] est le suivant :

```
1.  using System;
2.
3.  namespace pam_v5
4.  {
5.    public class Global : System.Web.HttpApplication
6.    {
7.
8.      protected void Application_Start(object sender, EventArgs e)
9.      {
10.
11.     }
12. ...
13.   }
14. }
```

Que devons-nous faire dans la méthode *Application_Start* ? Exactement la même chose que dans les applications web précédentes. Revenons à l'architecture de l'application et positionnons-y la classe [Global] :



Dans le schéma ci-dessus,

- la classe [Global] est instanciée lorsque le service web est démarré. Elle reste en mémoire tant que le service web est actif.
- la classe [Global] instancie les couches [metier] et [dao] dans sa méthode [Application_Start]
- pour améliorer les performances, la classe [Global] met la liste simplifiée des employés dans un champ interne. Elle délivrera la liste des employés à partir de ce champ.
- le service web est lui instancié à chaque requête d'un client. Il disparaît après avoir servi celle-ci. Il ne s'adressera pas directement à la couche [metier] mais à la classe [Global]. Celle-ci implémentera l'interface de la couche [metier].

La classe [Global] est analogue à celle déjà construite pour les applications précédentes :

```
1.  using System;
2.  using Pam.Dao.Entites;
3.  using Pam.Metier.Entites;
4.  using Pam.Metier.Service;
5.  using Spring.Context.Support;
6.
7.  namespace pam_v5
8.  {
9.    public class Global : System.Web.HttpApplication
10.   {
11.     // --- données statiques de l'application ---
12.     public static Employe[] Employes;
13.     public static IPamMetier PamMetier = null;
14.
15.     protected void Application_Start(object sender, EventArgs e)
16.     {
17.       // instanciation couche [metier]
18.       PamMetier = ContextRegistry.GetContext().GetObject("pammetier") as IPamMetier;
19.       // on récupère le tableau simplifié des employés
20.       Employes = PamMetier.GetAllIdentitesEmployes();
21.     }
22.
23.     // liste simplifiée des employés
24.     static public Employe[] GetAllIdentitesEmployes()
25.     {
26.       return Employes;
27.     }
28.
29.     // salaire d'un employé
30.     static public FeuilleSalaire GetSalaire(string SS, double heuresTravaillées, int
    joursTravailles)
31.     {
32.       return PamMetier.GetSalaire(SS, heuresTravaillées, joursTravailles);
33.     }
34.   }
35. }
```

La classe [Global] implémente l'interface [IPamMetier] mais ce n'est pas dit explicitement par la déclaration :

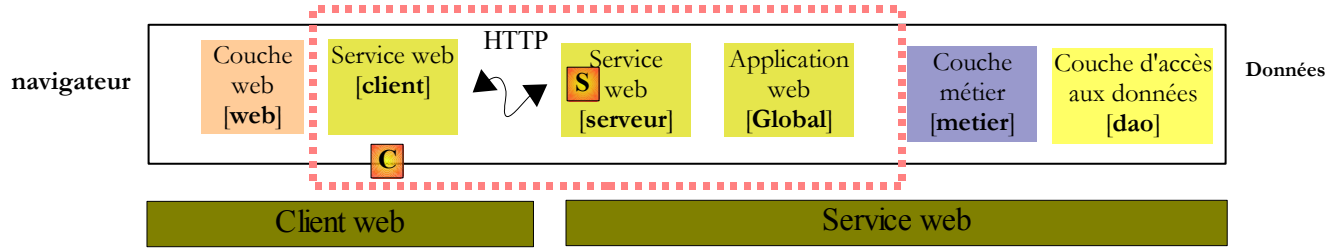
```
public class Global : System.Web.HttpApplication, IPamMetier
```

En effet, les méthodes *GetAllIdentitesEmployes* (ligne 24) et *GetSalaire* (ligne 30) sont statiques alors que les méthodes de l'interface *IPamMetier* ne le sont pas. Aussi la classe *Global* ne peut-elle implémenter l'interface *IPamMetier*. Par ailleurs, il n'est pas possible de déclarer les méthodes *GetAllIdentitesEmployes* et *GetSalaire* non statiques. En effet, elles sont accédées via le nom de la classe et non via une instance de celle-ci.

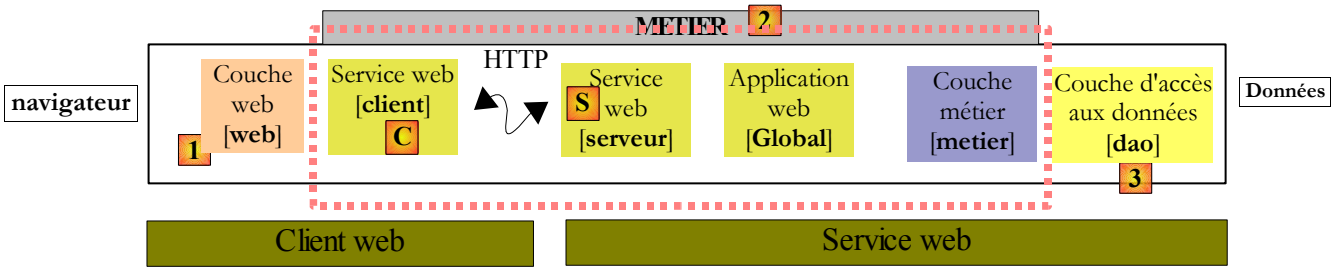
- ligne 15 : la méthode *Application_Start* est analogue à celle des classes [Global] étudiées dans les versions précédentes. Elle instancie la couche [metier] (ligne 18) puis initialise (ligne 20) le tableau des employés de la ligne 12.
- ligne 24 : la méthode *GetAllIdentitesEmployes* se contente de rendre le tableau des employés de la ligne 12. C'est là l'intérêt de l'avoir mémorisé dès le démarrage de l'application.
- ligne 30 : la méthode *GetSalaire* fait appel à la méthode du même nom de la couche [metier].

Pour instancier la couche [metier] (ligne 18), la classe [Global] utilise le framework Spring. Celui-ci est configuré par le fichier [Web.config] qui est identique à celui du projet précédent : il configure Spring et NHibernate afin d'instancier les couches [metier] et [dao] du service web.

Revenons à l'architecture de notre application client / serveur :



Côté serveur, il ne reste plus qu'à écrire le service web [S] lui-même. Si nous revenons à l'architecture de l'application :



nous voyons que côté serveur, toutes les couches qui précèdent la couche [metier] implémentent l'interface de celle-ci *IPamMetier*. Ce n'est pas obligatoire mais c'est une démarche qui paraît logique. Ce raisonnement pourra être appliqué côté client, au client [C] du service web [S]. Ainsi toutes les couches séparant la couche [web] de la couche [metier] implémentent-elles alors l'interface *IPamMetier*. On peut dire ainsi qu'on est revenu à une application à trois couches :

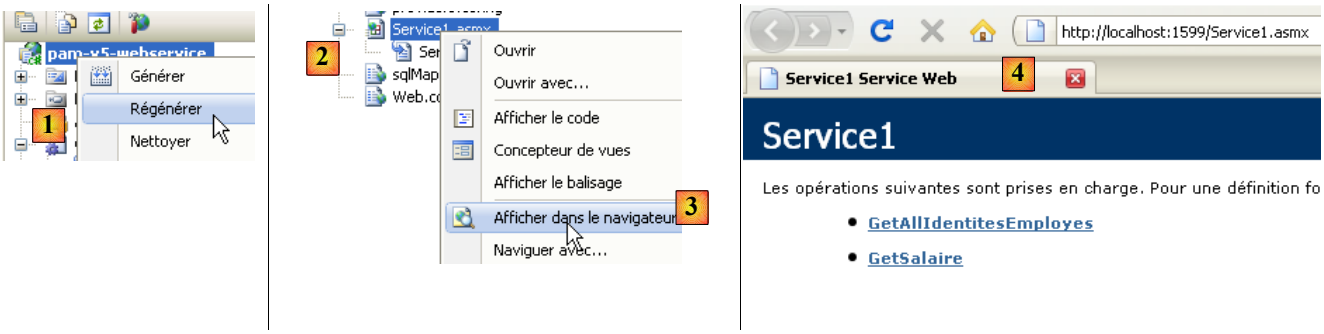
- la couche de présentation [web] [1]
- la couche [metier] [2]
- la couche d'accès aux données [3]

L'implémentation du service web [Service1.asmx.cs] pourrait être la suivante :

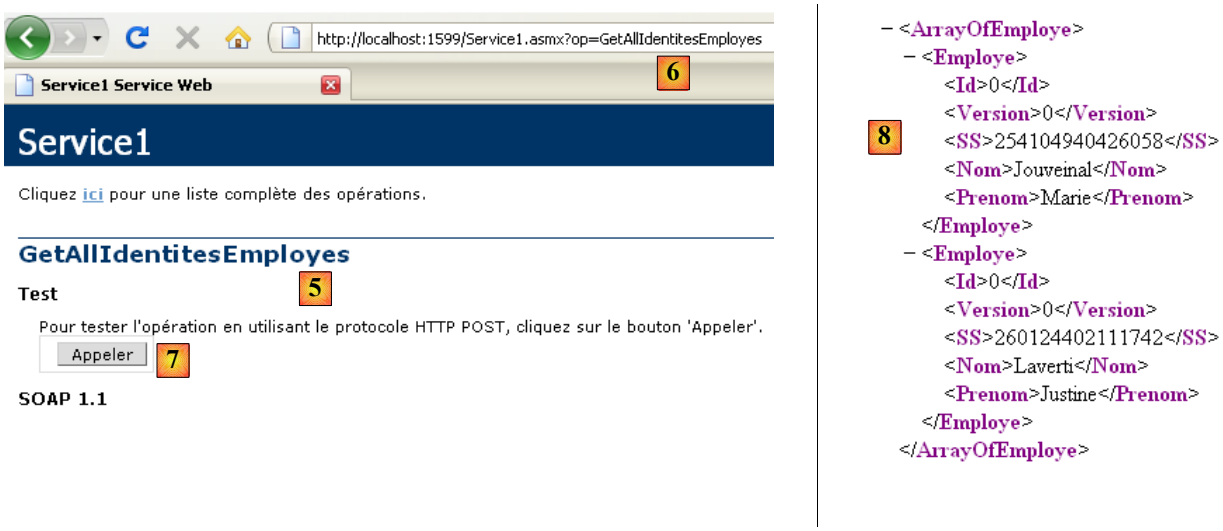
```
1.  using System.Web.Services;
2.  using Pam.Dao.Entites;
3.  using Pam.Metier.Entites;
4.  using Pam.Metier.Service;
5.
6.  namespace pam_v5
7.  {
8.    [WebService(Namespace = "http://st.istia.univ-angers.fr/")]
9.    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
10.   [System.ComponentModel.ToolboxItem(false)]
11.   public class Service1 : System.Web.Services.WebService, IPamMetier
12.   {
13.
14.     // liste de toutes les identités des employés
15.     [WebMethod]
16.     public Employe[] GetAllIdentitesEmployes()
17.     {
18.       return Global.GetAllIdentitesEmployes();
19.     }
20.
21.     // ------- le calcul du salaire
22.     [WebMethod]
23.     public FeuilleSalaire GetSalaire(string ss, double heuresTravaillees, int joursTravailles)
24.     {
25.       return Global.GetSalaire(ss, heuresTravaillees, joursTravailles);
26.     }
27.   }
28. }
```

- ligne 8 : la classe est annotée avec l'attribut [WebService] et nous donnons un nom à l'espace de noms du service web
- ligne 11 : la classe [Service1] hérite de la classe [WebService] et implémente l'interface [IPamMetier]
- lignes 15 et 22 : chaque méthode de la classe est annotée avec l'attribut [WebMethod] afin d'être exposée aux clients distants. Par défaut toutes les méthodes publiques d'un service web sont exposées. Les attributs des lignes 15 et 22 sont donc facultatifs ici. Pour implémenter l'interface [IPamMetier] chaque méthode se contente d'appeler la méthode de même nom de la classe [Global].

Nous sommes prêts pour l'exécution du service web :

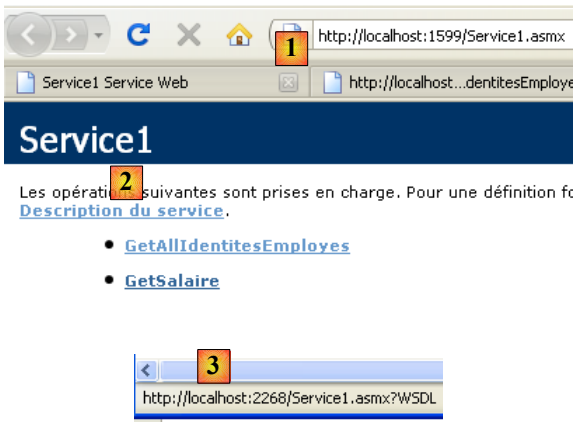


- en [1], le projet est régénéré
- en [2], on sélectionne le service web [Service1.asmx] et on l'affiche dans le navigateur [3]
- en [4], la page web affichée. Elle présente les méthodes du service web.



- en [4], nous suivons le lien [GetAllIdentitesEmployes] et nous obtenons en [5] la page de test de cette méthode.
- en [6], l'Url de la méthode
- en [7], le bouton [Appeler] permettant de tester la méthode. Celle-ci ne demande aucun paramètre.
- en [8], le résultat Xml renvoyé par le service web. Dans celui-ci, seuls les propriétés *SS*, *Nom*, *Prenom* des objets *Employe* sont significatifs car la méthode [GetAllIdentitesEmployes] ne demande que ces propriétés. Cependant cette méthode rend un tableau d'objets *Employe*. On voit en [8] que les propriétés numériques *Id*, *Version* sont dans le flux Xml renvoyé mais pas les propriétés ayant une valeur *null* : *Adresse*, *Ville*, *CodePostal*, *Indemnites*.

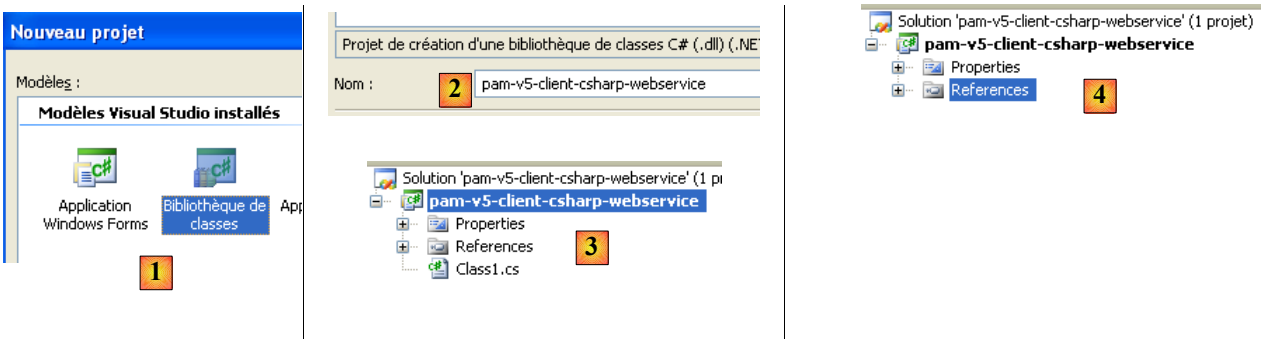
Nous avons un service web actif. Nous allons maintenant lui écrire un client C#. Pour cela, nous aurons besoin de l'Uri du fichier WSDL du service web. Nous l'obtenons dans la page initialement affichée lors de l'exécution de [Service.asmx] :



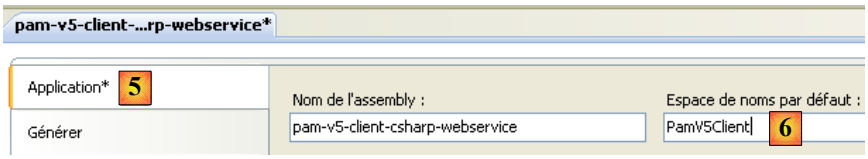
- en [1], l'Uri du service web
- en [2], le lien qui mène à son fichier WSDL
- en [3], la valeur de ce lien

9.3 Le projet C# d'un client NUnit du service web

Nous créons un projet C# (avec Visual C# et non pas Visual Web Developer) pour le client du service web. Ce sera un client de test NUnit. Aussi le projet sera-t-il de type "Bibilothèque de classes".

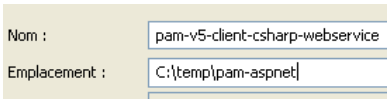


- en [1], nous créons un projet C# de type "Bibliothèque de classes"
- en [2], nous donnons un nom au projet
- en [3], le projet. Nous supprimons [Class1.cs].
- en [4], le nouveau projet.

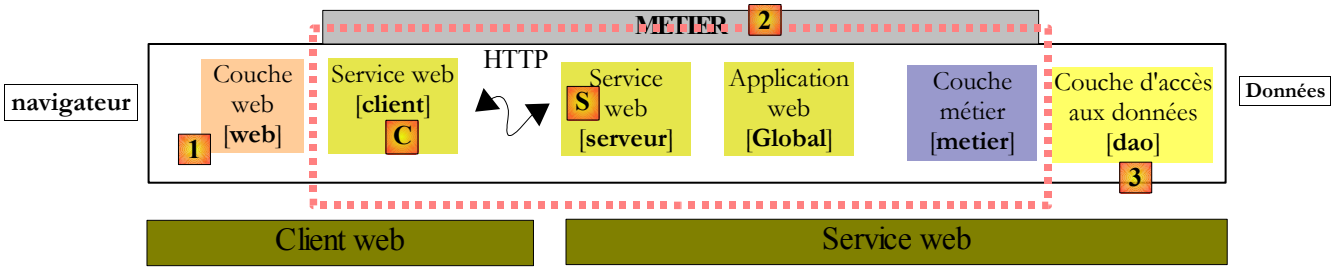


- dans les propriétés du projet, dans l'onglet [Application] [5], nous fixons l'espace de noms du projet. Chaque classe générée par l'IDE le sera dans cet espace.

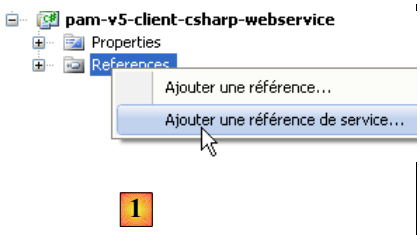
Nous sauvegardons notre nouveau projet à un endroit qui nous convient :

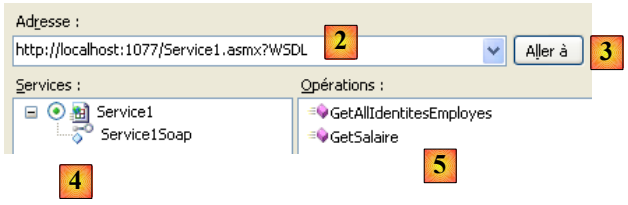


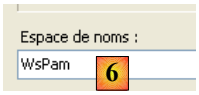
Ceci fait, nous générons le client du service web distant. Pour comprendre ce que nous allons faire, il faut revenir à l'architecture client / serveur en cours de construction :



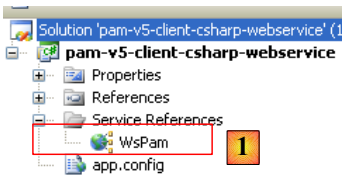
L'IDE va générer la couche client [C] à partir de l'Uri du fichier WSDL du service web [S]. Rappelons que l'Uri de ce fichier a été notée à la page 145. Nous procédons de la façon suivante :

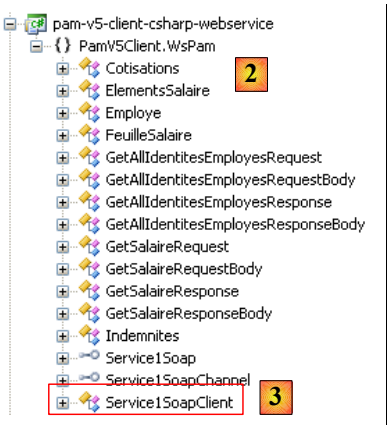


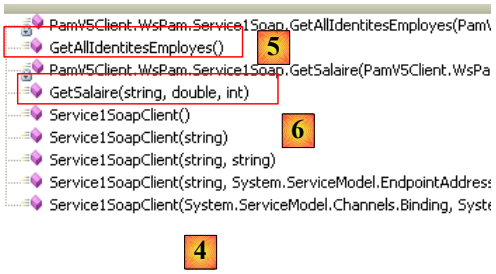




- en [1], cliquer droit sur la branche *References* et ajouter une *référence de service*
- en [2], indiquer l'Url du fichier WSDL du service web (cf page 145). Celui-ci doit être lancé auparavant s'il ne l'est pas.
- en [3], demander la découverte du service web au travers de son fichier WSDL
- en [4], le service web découvert
- en [5], les méthodes exposées par le service web.
- en [6], l'espace de noms dans lequel on veut placer les classes et interfaces du client qui va être généré.
- on valide l'assistant





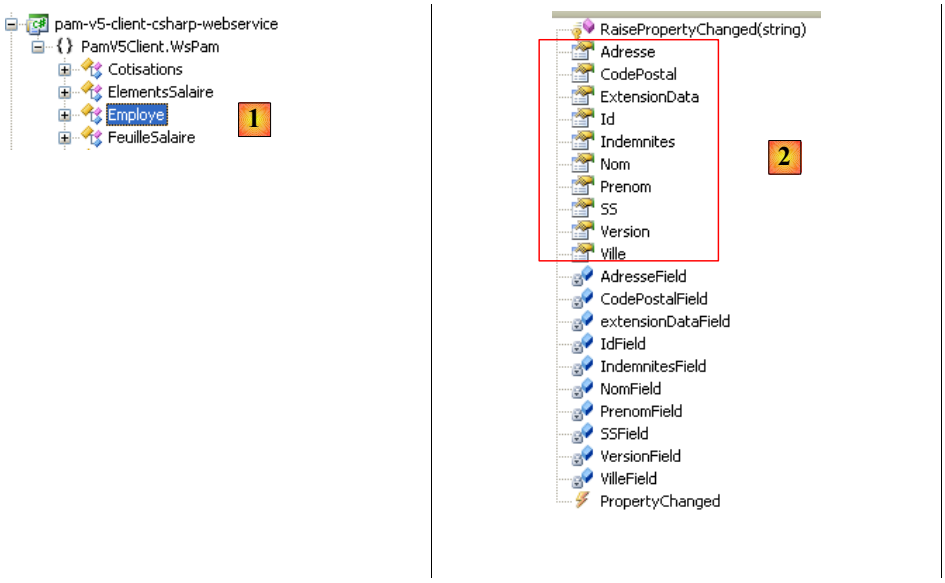


- en [1], le client généré. On double-clique dessus pour avoir accès à son contenu.

- en [2], dans l'explorateur d'objets, les classes et interfaces de l'espace de noms *Client.WsPam* sont affichées. C'est l'espace de noms du client généré.
- en [3], la classe qui implémente le client du service web.
- en [4], les méthodes implémentées par le client [Service1SoapClient]. On y retrouve les deux méthodes du service web distant [5] et [6].
- en [2], on retrouve les images des entités des couches :
  - [metier] : *FeuilleSalaire*, *ElementsSalaire*
  - [dao] : *Employe*, *Cotisations*, *Indemnites*

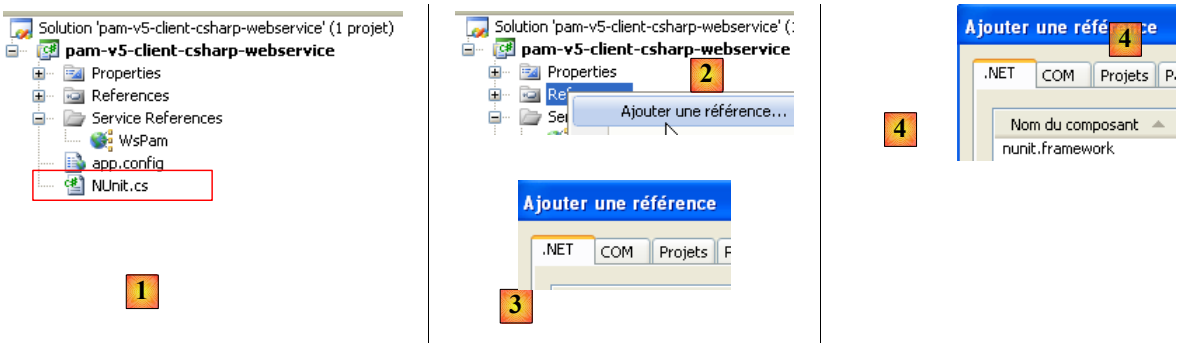
  Dans la suite, il faut se rappeler que ces images des entités distantes se trouvent côté client et dans l'espace de noms *PamV5Client.WsPam*.

Examinons les méthodes et propriétés exposées par l'une d'elles :



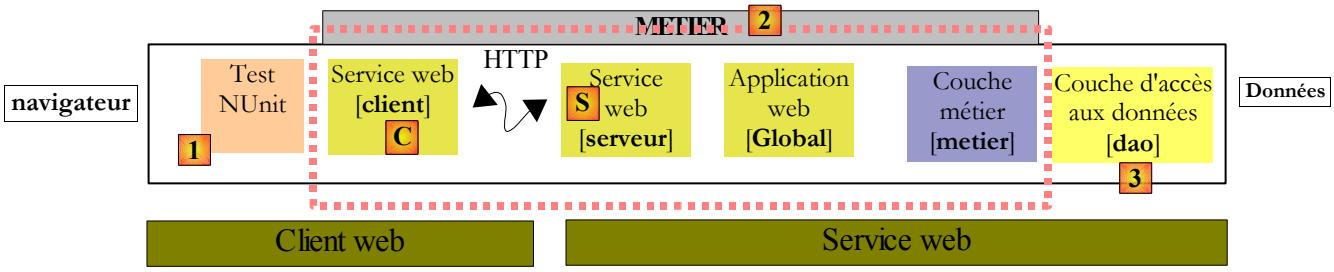
- en [1], on sélectionne la classe [Employe] locale
- en [2], on retrouve les propriétés de l'entité [Employe] distante ainsi que des champs privés utilisés pour les besoins propres de l'entité locale.

Revenons à notre application C#. Nous lui ajoutons une classe de test NUnit :



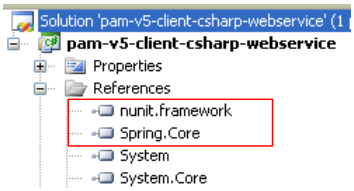
- en [1], la classe [NUnit] rajoutée. La classe [NUnit] va avoir besoin du framework NUnit et donc d'une référence sur la DLL de celui-ci. Nous supposons ici que le framework NUnit a été installé sur le poste (http://nunit.org/).
- en [2], on ajoute une référence au projet
- dans l'onglet [3] .NET qui rassemble les DLL enregistrées sur le poste, nous sélectionnons [4], la DLL [nunit.framework] version 2.4.6 minimale.

Par ailleurs, nous allons utiliser Spring pour instancier le client local [C] du service web [S] :

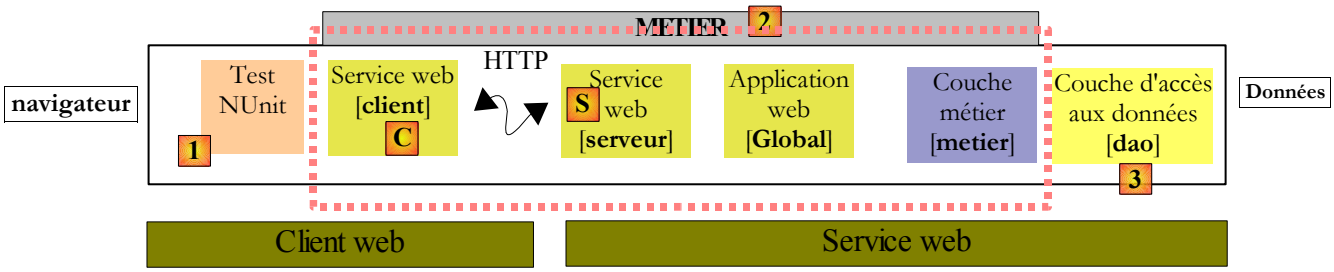


La référence sur la DLL de Spring peut être rajoutée comme il a été fait avec le framework NUnit si les DLL ont été au préalable enregistrées sur le poste (http://www.springframework.net/download.html).

Nous procédons différemment. Nous utilisons le dossier [lib] des projets précédents qui contenait les DLL nécessaires à Spring et nous rajoutons la référence de Spring au projet :



Revenons sur l'architecture du client en cours de construction :



Ci-dessus, nous voyons que le client de test [1] s'interface avec une couche [metier] [2] étendue. Celle-ci présente les mêmes méthodes que la couche [metier] distante. Nous pouvons donc utiliser la classe de test déjà rencontrée lors du test de la couche [metier] dans le projet C# [pam-metier-dao-nhibernate] :

```
1.  using NUnit.Framework;
2.  using Pam.Dao.Entites;
3.  using Pam.Metier.Entites;
4.  using Pam.Metier.Service;
5.  using Spring.Context.Support;
6.
7.  namespace Pam.Metier.Tests {
8.
9.     [TestFixture()]
10.    public class NunitTestPamMetier : AssertionHelper {
11.
12.       // la couche [metier] à tester
13.       private IPamMetier pamMetier;
14.
15.       // constructeur
16.       public NunitTestPamMetier() {
17.          // instanciation couche [dao]
18.          pamMetier = ContextRegistry.GetContext().GetObject("pammetier") as IPamMetier;
19.       }
20.
21.
22.       [Test]
23.       public void GetAllIdentitesEmployes() {
```

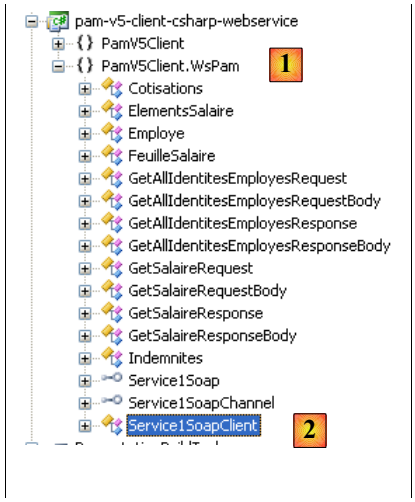
```
24.          // vérification nbre d'employes
25.          Expect(2, EqualTo(pamMetier.GetAllIdentitesEmployes().Length));
26.       }
27.
28.       [Test]
29.       public void GetSalaire1() {
30.          // calcul d'une feuille de salaire
31.          FeuilleSalaire feuilleSalaire = pamMetier.GetSalaire("254104940426058", 150, 20);
32.          // vérifications
33.          Expect(368.77, EqualTo(feuilleSalaire.ElementsSalaire.SalaireNet).Within(1E-06));
34.          // feuille de salaire d'un employé inexistant
35.          bool erreur = false;
36.          try {
37.             feuilleSalaire = pamMetier.GetSalaire("xx", 150, 20);
38.          } catch (PamException) {
39.             erreur = true;
40.          }
41.          Expect(erreur, True);
42.       }
43.
44.    }
45. }
```

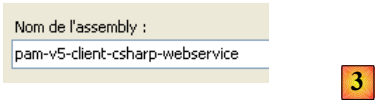
Il y a quelques modifications à faire :

- ligne 18, on instancie la couche [metier] avec le framework Spring. La classe n'est pas la même dans les deux cas. Ici, la couche [metier] locale est une instance de la classe [PamV5Client.WsPam.Service1SoapClient], la classe générée par l'IDE. Aussi Spring est-il configuré de la façon suivante dans le fichier [app.config] du projet C# :

```
1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <configuration>
3.
4.     <configSections>
5.        <sectionGroup name="spring">
6.           <section name="context" type="Spring.Context.Support.ContextHandler, Spring.Core" />
7.           <section name="objects" type="Spring.Context.Support.DefaultSectionHandler, Spring.Core" />
8.        </sectionGroup>
9.     </configSections>
10.
11.    <spring>
12.       <context>
13.          <resource uri="config://spring/objects" />
14.       </context>
15.       <objects xmlns="http://www.springframework.net">
16.          <object id="pammetier" type="PamV5Client.WsPam.Service1SoapClient, pam-v5-client-csharp-webservice"/>
17.       </objects>
18.    </spring>
19.
20.
21.    <system.serviceModel>
22. ...
```

  - ligne 16 ci-dessus, l'objet [pammetier] est une instance de la classe [PamV5Client.WsPam.Service1SoapClient] qui se trouve dans l'assemblage (assembly) [pam-v5-client-csharp-webservice]. Pour avoir la première information, il suffit de revenir sur la définition de la classe [Service1SoapClient] dans l'explorateur d'objets (page 146) :





- en [2], la classe d'implémentation de la couche [metier] locale et en [1] son espace de noms
- en [3], dans les propriétés du projet, le nom de l'assembly, la deuxième information nécessaire à la configuration de l'objet Spring [pammetier].

Revenons au code de l'instanciation de la couche [metier] locale dans [NUnit.cs] :

```
1.        // la couche [metier] à tester
2.        private IPamMetier pamMetier;
3.
4.        // constructeur
5.        public NunitTestPamMetier() {
6.           // instanciation couche [dao]
7.           pamMetier = ContextRegistry.GetContext().GetObject("pammetier") as IPamMetier;
8.        }
9.
```

Ligne 7, la couche [metier] distante était de type *IPamMetier*. Ici la couche [metier] est de type [Service1SoapClient] :

```
public class Service1SoapClient : System.ServiceModel.ClientBase<Service1Soap>
```

Nous voyons que la classe *Service1SoapClient* n'implémente pas l'interface *IPamMetier* même si elle expose des méthodes de même nom. Il nous faut donc écrire l'instanciation de la couche [metier] locale de la façon suivante :

```
1.        // la couche [metier] à tester
2.        private Service1SoapClient pamMetier;
3.
4.        // constructeur
5.        public NunitTestPamMetier() {
6.           // instanciation couche [metier]
7.           pamMetier = ContextRegistry.GetContext().GetObject("pammetier") as Service1SoapClient;
8.     }
```

Autre modification à faire :

```
1.        [Test]
2.        public void GetSalaire1() {
3.     ...
4.           try {
5.              feuilleSalaire = pamMetier.GetSalaire("xx", 150, 20);
6.           } catch (PamException) {
7.              erreur = true;
8.           }
9.           Expect(erreur, True);
10.       }
11.
```

Le code ci-dessus utilise, ligne 6, le type *PamException* qui n'existe pas côté client. On le remplacera par sa classe parent, le type *Exception*.

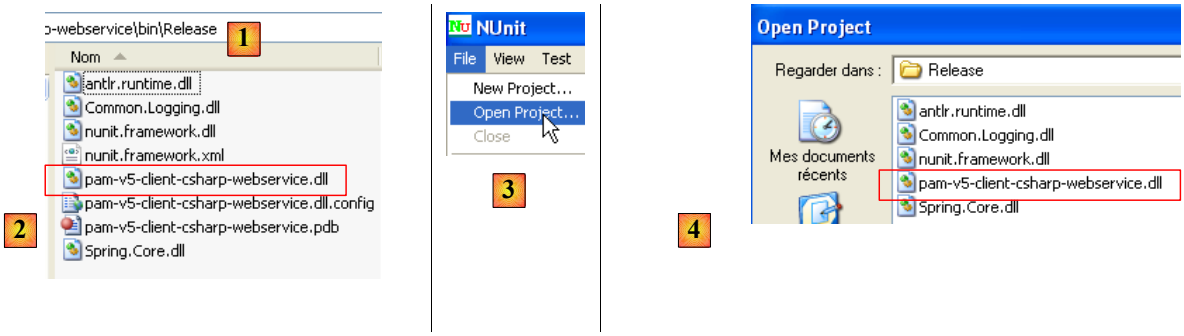
```
1.        [Test]
```

```
2.        public void GetSalaire1() {
3.  ...
4.           try {
5.              feuilleSalaire = pamMetier.GetSalaire("xx", 150, 20);
6.           } catch (Exception) {
7.              erreur = true;
8.           }
9.           Expect(erreur, True);
10.       }
11.
```

Enfin, les paquetages importés ne sont plus les mêmes :

```
using System;
using PamV5Client.WsPam;
using NUnit.Framework;
using Spring.Context.Support;
```

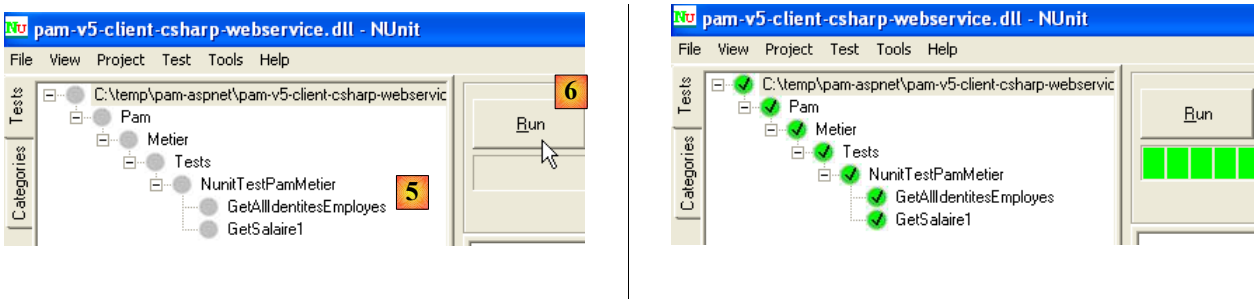
Ceci fait, le projet de type "Bibliothèque de classes" peut être généré. La DLL suivante est créée :



- en [1], le dossier [bin/Release] du projet C#
- en [2], la DLL du projet.

Le test NUnit est ensuite exécuté par le framework NUnit (la base MySQL *dbpam_nhibernate* doit être active pour le test) :

- en [3] et [4], la DLL [2] est chargée dans l'application de test NUnit



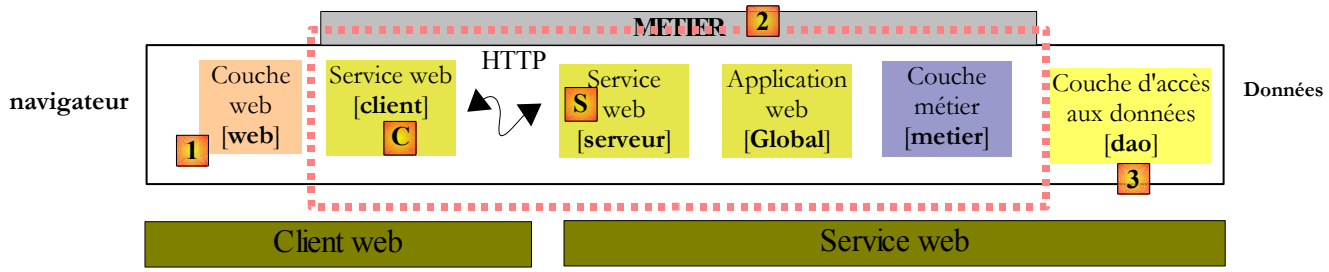
- en [5], la classe de test est sélectionnée et exécutée [6]
- en [7], les résultats d'un test réussi

Nous avons désormais un service web opérationnel.

10 L'application [SimuPaie] – version 6 – client ASP.NET d'un service web

10.1 L'architecture de l'application

Nous faisons évoluer l'architecture client / serveur du test NUnit de la façon suivante :

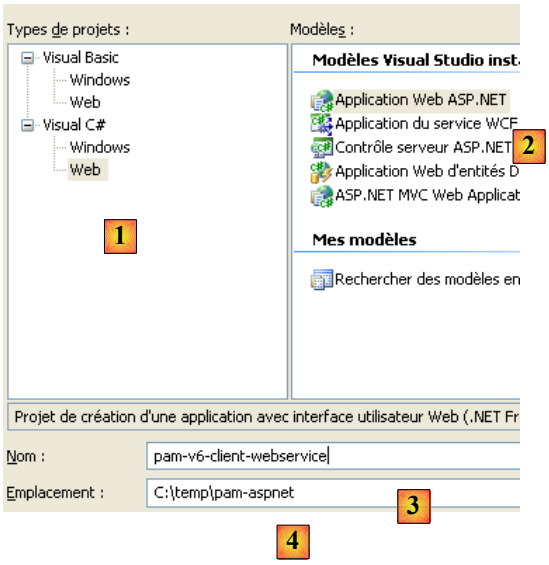


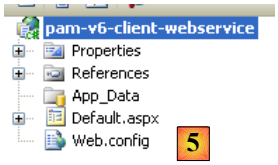
En [1], le test NUnit est remplacé par l'application web de la version 8. Il faut se rappeler ici que cette architecture comporte deux serveurs web non représentés :

- un serveur web qui exécute le service web [S]. Il s'exécutera dans une première instance de Visual Web Developer.
- un serveur web qui exécute le client web [1]. Il s'exécutera dans une deuxième instance de Visual Web Developer.

10.2 Le projet Visual Web Developer du client [web]

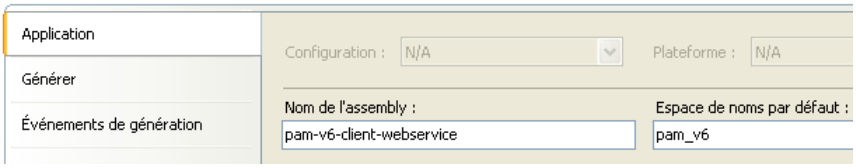
Nous créons un nouveau projet Web ASP.NET :





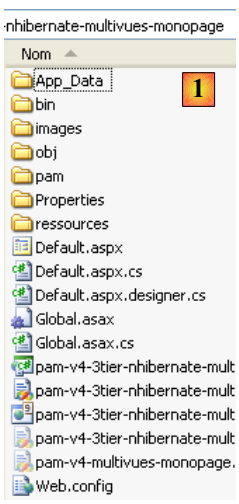
- en [1], nous choisissons un projet web en C#
- en [2], nous choisissons "Application Web ASP.NET"
- en [3], nous donnons un nom au projet web
- en [4], nous indiquons un emplacement pour ce projet
- en [5], le projet créé

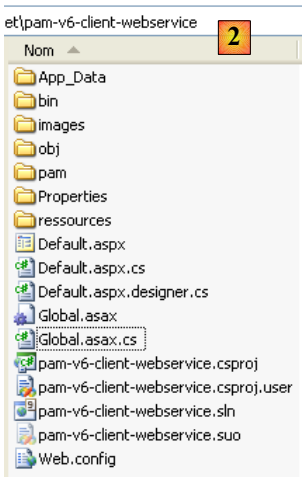
Nous modifions quelques propriétés du projet :

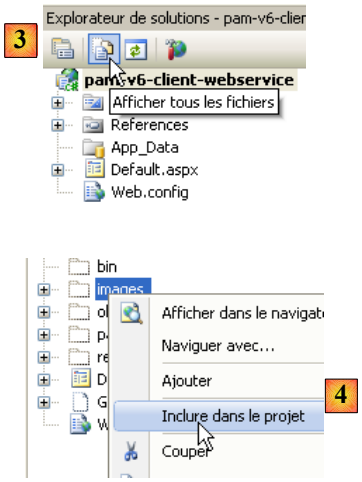


- en [1], le nom de l'assemblage qui sera généré
- en [2], l'espace de noms par défaut des classes et interfaces qui seront créées

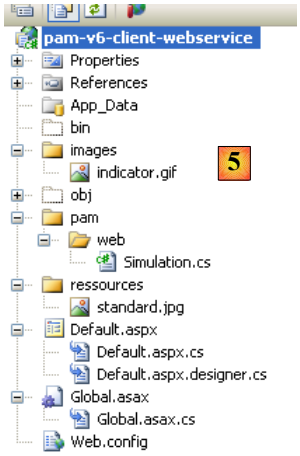
Pour construire le projet [pam-v6-client-webservice] on peut partir de la couche [web] du projet [pam-v4-3tier-nhibernate-multivues-monopage]. On peut récupérer les fichiers [Global.asax] et [Default.aspx] du projet [pam-v4-3tier-nhibernate-multivues-monopage] et les copier dans le projet [pam-v6-client-webservice]. On fait cela avec l'explorateur windows tout d'abord :

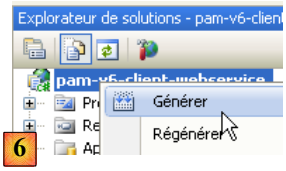






- en [1], le dossier du projet [pam-v4-3tier-nhibernate-multivues-monopage]
- en [2], le dossier du projet [pam-v6-client-webservice] après recopie
  - des dossiers [images, pam, ressources]
  - des fichiers [Global.asax, Global.asax.cs]
  - des fichiers [Default.aspx, Default.aspx.cs, Default.aspx.designer.cs]
- en [3], dans Visual Studio Express, on fait afficher tous les fichiers du projet, pour faire apparaître les fichiers nouvellement ajoutés
- en [4], on inclut dans le projet les dossiers et fichiers ajoutés



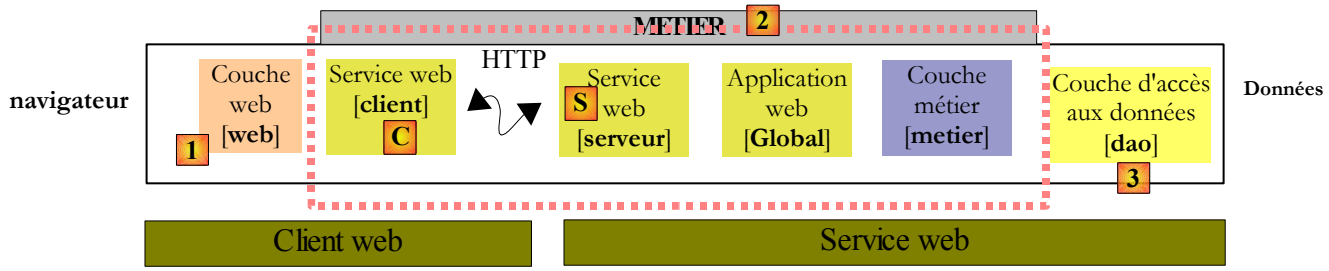


- en [5] le nouveau projet [pam-v6-client-webservice]

A ce stade, on peut générer le projet une première fois [6]. On obtient les erreurs suivantes :

```
1.  Erreur   1  Le type ou le nom d'espace de noms 'Dao' n'existe pas dans l'espace de noms 'Pam' (une
    référence d'assembly est-elle manquante ?)  C:\temp\pam-aspnet\pam-v6-client-
    webservice\Global.asax.cs 3  11 pam-v6-client-webservice
2.  Erreur   2  Le type ou le nom d'espace de noms 'Metier' n'existe pas dans l'espace de noms 'Pam'
    (une référence d'assembly est-elle manquante ?)     C:\temp\pam-aspnet\pam-v6-client-
    webservice\Global.asax.cs 4  11 pam-v6-client-webservice
3.  Erreur   3  Le type ou le nom d'espace de noms 'Dao' n'existe pas dans l'espace de noms 'Pam' (une
    référence d'assembly est-elle manquante ?)  c:\temp\pam-aspnet\pam-v6-client-
    webservice\default.aspx.cs5  11 pam-v6-client-webservice
4.  Erreur   4  Le type ou le nom d'espace de noms 'Spring' est introuvable (une directive using ou une
    référence d'assembly est-elle manquante ?)  C:\temp\pam-aspnet\pam-v6-client-
    webservice\Global.asax.cs 6  7  pam-v6-client-webservice
5.  Erreur   5  Le type ou le nom d'espace de noms 'Metier' n'existe pas dans l'espace de noms 'Pam'
    (une référence d'assembly est-elle manquante ?)     c:\temp\pam-aspnet\pam-v6-client-
    webservice\default.aspx.cs6  11 pam-v6-client-webservice
6.  Erreur   6  Le type ou le nom d'espace de noms 'Employe' est introuvable (une directive using ou
    une référence d'assembly est-elle manquante ?)      C:\temp\pam-aspnet\pam-v6-client-
    webservice\Global.asax.cs 13 19 pam-v6-client-webservice
7.  Erreur   7  Le type ou le nom d'espace de noms 'IPamMetier' est introuvable (une directive using ou
    une référence d'assembly est-elle manquante ?)      C:\temp\pam-aspnet\pam-v6-client-
    webservice\Global.asax.cs 14 19 pam-v6-client-webservice
```

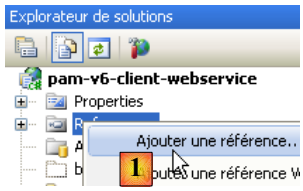
Pour comprendre ces erreurs et les corriger, il faut revenir à l'architecture du projet web en construction :

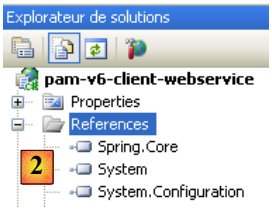


Le projet [pam-v6-client-webservice] est la couche [web] [1] dans le schéma ci-dessus. On voit que cette couche communique avec le client [C] du service web, un client que nous allons générer prochainement.

- les erreurs 2, 5, 7 viennent du fait que le code de [pam-v4] référençait la DLL de la couche [metier], DLL qui est maintenant du côté serveur et non du côté client.
- les erreurs 1, 3 ont la même cause, mais cette fois pour la DLL de la couche [dao]
- l'erreur 4 vient du fait que le code de [Global.asax] utilise Spring et que notre projet ne référence pas la DLL de Spring. Nous allons ajouter cette référence.
- l'erreur 6 vient du fait que la classe [Employe] est définie dans la couche [dao] qui n'existe pas côté client.

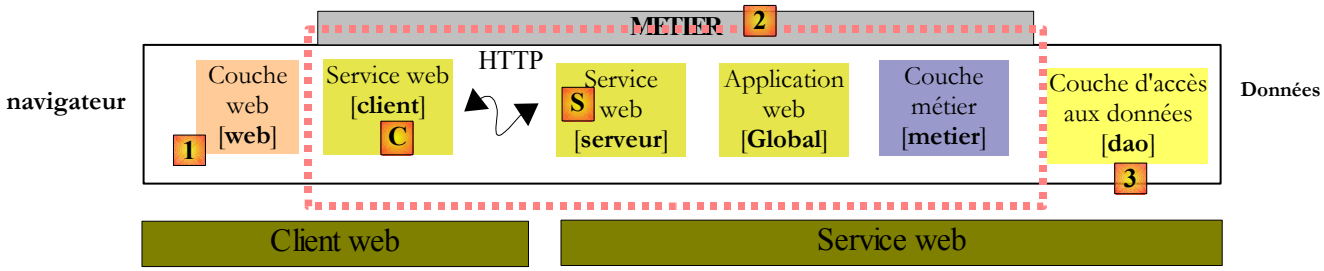
Les erreurs d'espace de noms 1, 2, 4, 5, 6, 7 vont être résolues par la génération du client C du service web. L'erreur 3 est résolue en ajoutant au projet une référence sur la DLL de Spring. Nous pouvons procéder comme suit :





- en [1], on ajoute une référence au projet [pam-v6-client-webservice]
- en [4], on a ajouté une référence sur la DLL [Spring.Core] du dossier [lib].

Après l'ajout de cette référence sur Spring, la génération du projet présente une erreur en moins. Toutes les autres erreurs sont dues à des lignes de code qui référencent des objets des couches [metier] et [dao] qui sont maintenant sur le serveur. Nous verrons que ces objets manquants seront générés dans le client [C] du service web.

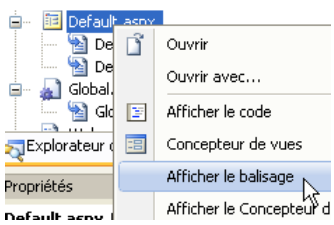


Avant de générer le client [C] du service web, nous allons modifier l'espace de noms des différentes classes présentes. Elles sont actuellement dans l'espace de noms [pam-v4]. Nous changeons cet espace de noms en [pam-v6]. Les fichiers concernés sont les suivants : *Default.aspx.cs, Default.aspx.designer.cs, Global.asax.cs*. Par exemple :

```
1.  ....
2.  namespace pam_v6
3.  {
4.    public class Global : System.Web.HttpApplication
5.    {
6.      // --- données statiques de l'application ---
7.      public static Employe[] Employes;
8.      public static IPamMetier PamMetier = null;
9.  ....
```

Ligne 2, l'espace de noms *pam_v4* a été remplacé par l'espace de noms *pam_v6*.

Par ailleurs, il faut modifier le balisage des fichiers : *Default.aspx* et *Global.asax* :



Le balisage de [Default.aspx] devient :

```
1.  <%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="pam_v6.PagePam"
    %>
```

```
2.
3.
4.  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
5.  .....
```

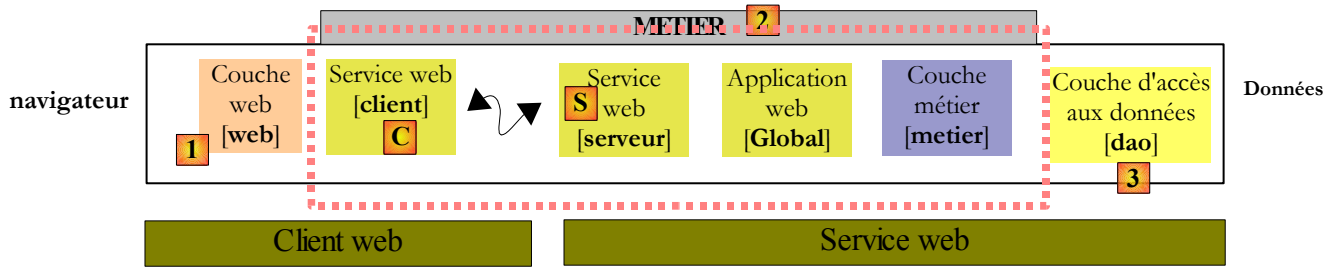
Ligne 1, l'attribut *Inherits* désigne le nom de la classe du fichier [Default.aspx.cs]. On change l'espace de noms.

Le balisage de [Global.asax] devient :

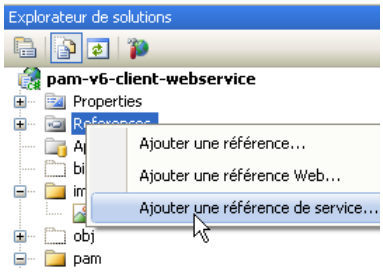
```
<%@ Application Codebehind="Global.asax.cs" Inherits="pam_v6.Global" Language="C#" %>
```

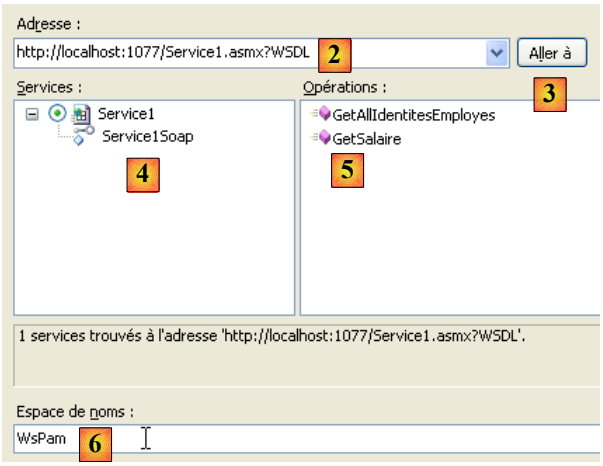
Ci-dessus, on change l'espace de noms de l'attribut *Inherits*.

Nous générons maintenant le client [C] du service web.

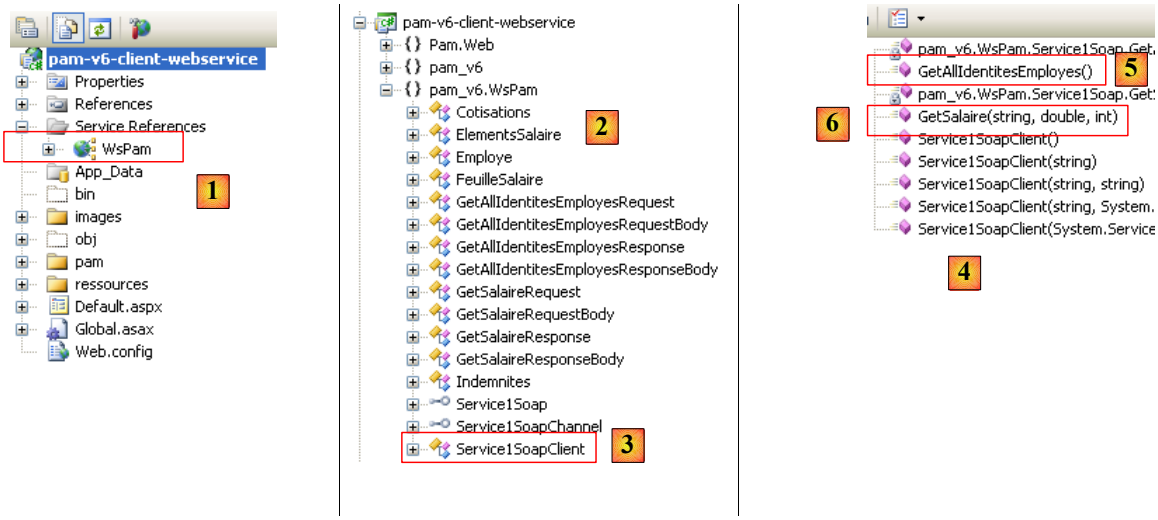


Pour faire les opérations qui suivent, il faut que le service web [S] [pam-v5-webservice] soit lancé dans une autre instance de Visual Web Developer.





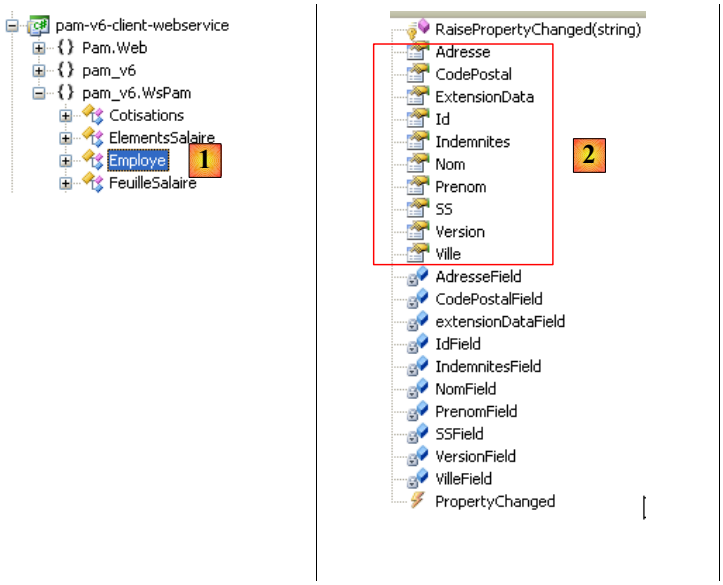
- en [1], on ajoute au projet [pam-v6-client-webservice], une référence au service web [pam-v5-webservice]
- en [2], l'Uri du service web [pam-v5-webservice]. C'est l'Url du fichier WSDL de ce service web. Dans la construction du service web [pam-v5-webservice], on a indiqué comment connaître cette Url.
- en [3], on demande à l'assistant d'exploiter le fichier WSDL indiqué en [2]
- en [4], le service web [Service1Soap] découvert et en [5] les méthodes distantes qu'il expose.
- en [6], on fixe l'espace de noms dans lequel les classes et interfaces du client [C] vont être générées
- on lance la génération du client [C]



- en [1], le client généré. On double-clique dessus pour avoir accès à son contenu.
- en [2], dans l'explorateur d'objets, les classes et interfaces de l'espace de noms *pam_v6.WsPam* sont affichées. C'est l'espace de noms du client généré.
- en [3], la classe qui implémente le client du service web.
- en [4], les méthodes implémentées par le client [Service1SoapClient]. On y retrouve les deux méthodes du service web distant [5] et [6].
- en [2], on retrouve les images des entités des couches :
    - [metier] : *FeuilleSalaire*, *ElementsSalaire*
    - [dao] : *Employe*, *Cotisations*, *Indemnites*

  Dans la suite, il faut se rappeler que ces images des entités distantes se trouvent côté client et dans l'espace de noms *pam_v6.WsPam*.

Examinons les méthodes et propriétés exposées par l'une d'elles :



- en [1], on sélectionne la classe [Employe] locale
- en [2], on retrouve les propriétés de l'entité [Employe] distante ainsi que des champs privés utilisés pour les besoins propres de l'entité locale.

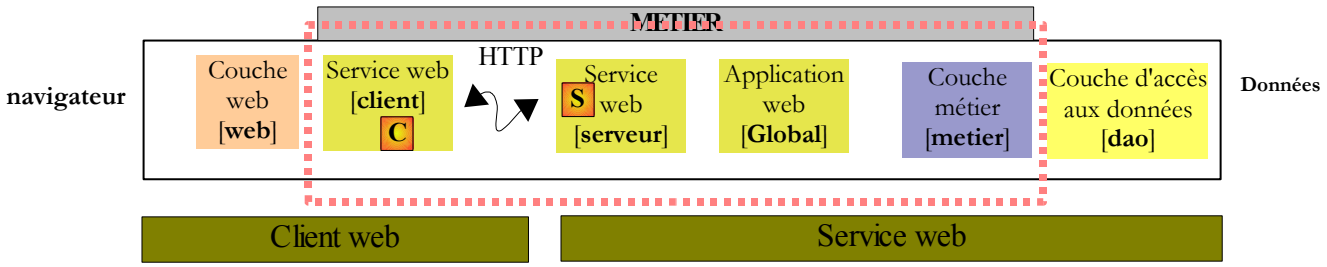
Examinons le code de [Global.asax.cs] qui présente des erreurs :

```
1  using System;
2  using System.Collections.Generic;
3  using Pam.Dao.Entites;
4  using Pam.Metier.Service;
5  using Pam.Web;
6  using Spring.Context.Support;
7
8  namespace pam_v6
9  {
10   public class Global : System.Web.HttpApplication
11   {
12     // --- données statiques de l'application ---
13     public static Employe[] Employes;
14     public static IPamMetier PamMetier = null;
15     public static string Msg;
16     public static bool Erreur = false;
```

- les lignes 3 et 4 utilisent des espaces de noms qui existent sur le serveur mais pas chez le client.
- la ligne 13 utilise une classe [Employe] dont on n'a pas déclaré le bon espace de noms
- la ligne 14 utilise une interface *IPamMetier* inconnue chez le client.

Nous avons déjà rencontré des problèmes similaires dans le client C# étudié précédemment.

Dans l'architecture :



- la couche [metier] locale est implémentée par le client [C] de type *pam_v6.WsPam.Service1SoapClient* qui n'implémente pas l'interface *IPamMetier* même si par ailleurs elle présente des méthodes de mêmes noms.
- les entités manipulées (*Employe*, *Indemnites*, *Cotisations*) par le client [C] généré sont dans l'espace de noms *pam_v6.WsPam*

Le code de [Global.asax.cs] évolue comme suit :

```
1.  using System;
2.  using System.Collections.Generic;
3.  using Pam.Web;
4.  using Spring.Context.Support;
5.  using pam_v6.WsPam;
6.
7.  namespace pam_v6
8.  {
9.    public class Global : System.Web.HttpApplication
10.   {
11.     // --- données statiques de l'application ---
12.     public static Employe[] Employes;
13.     public static Service1SoapClient PamMetier = null;
14.     public static string Msg;
15.     public static bool Erreur = false;
16.
17.     // démarrage de l'application
18.     public void Application_Start(object sender, EventArgs e)
19.     {
20.       // exploitation du fichier de configuration
21.       try
22.       {
23.         // instanciation couche [metier]
```

```
24.         PamMetier = ContextRegistry.GetContext().GetObject("pammetier") as Service1SoapClient;
25.         // liste simplifiée des employés
26.         Employes = PamMetier.GetAllIdentitesEmployes();
27.         // on a réussi
28.         Msg = "Base chargée...";
29.       }
30.       catch (Exception ex)
31.       {
32.         // on note l'erreur
33.         Msg = string.Format("L'erreur suivante s'est produite lors de l'accès à la base de données
    : {0}", ex);
34.         Erreur = true;
35.       }
36.     }
37.
38.     public void Session_Start(object sender, EventArgs e)
39.     {
40.       // on met une liste de simulations vide dans la session
41.       List<Simulation> simulations = new List<Simulation>();
42.       Session["simulations"] = simulations;
43.     }
44.   }
45. }
```

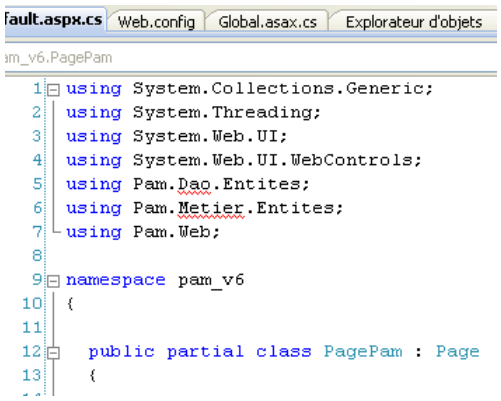
La ligne 24 instancie la couche [C] grâce à Spring. La configuration nécessaire à cette instanciation est faite dans [web.config] :

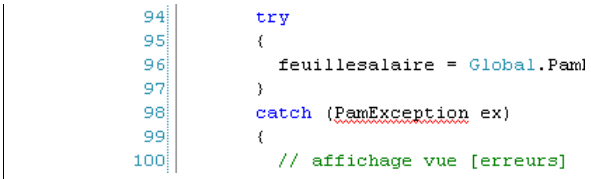
```
<configSections>
      <sectionGroup name="system.web.extensions"
type="System.Web.Configuration.SystemWebExtensionsSectionGroup, System.Web.Extensions, Version=3.5.0.0,
Culture=neutral, PublicKeyToken=31BF3856AD364E35">
 ...
      </sectionGroup>
      <sectionGroup name="spring">
        <section name="context" type="Spring.Context.Support.ContextHandler, Spring.Core" />
        <section name="objects" type="Spring.Context.Support.DefaultSectionHandler, Spring.Core" />
      </sectionGroup>
    </configSections>

  <spring>
    <context>
      <resource uri="config://spring/objects" />
    </context>
      <objects xmlns="http://www.springframework.net">
        <object id="pammetier" type="pam_v6.WsPam.Service1SoapClient,pam-v6-client-webservice"/>
      </objects>
  </spring>
```

Ligne 16, la couche [metier] est une instance de la classe [pam_v6.WsPam.Service1SoapClient] qui sera trouvée dans la DLL [pam-v6-client-webservice]. On rappelle que nous avons configuré le projet [pam-v6-client-webservice] pour qu'il génère cette DLL.

Il reste encore quelques erreurs dans [Default.aspx.cs] :





- lignes 5, 6 : ces espaces de noms n'existent plus. Les entités sont désormais dans l'espace de noms *pam_v6* ou *pam_v6.WsPam*.

- ligne 98 : le client généré [C] n'a pas repris la classe [PamException] de la couche [dao]. Il ne le pouvait pas puisque le service web n'expose pas cette exception. On choisit de remplacer *PamException* par sa classe parent *Exception*.

Le code devient :

```
1.  ...
2.  using System.Web.UI.WebControls;
3.  using pam_v6.WsPam;
4.
5.  namespace pam_v6
6.  {
7.
8.    public partial class PagePam : Page
9.    {
10. ...
```

```
1.  ...
2.        try
3.        {
4.          feuillesalaire = Global.PamMetier.GetSalaire(DropDownListEmployes.SelectedValue,
    HeuresTravaillées, JoursTravaillés);
5.        }
6.        catch (Exception ex)
7.        {
8.  ...
```

Une fois ces erreurs corrigées, nous pouvons exécuter l'application web :

- en [1], l'Url du client web du service web distant
- en [2], le combo des employés a été rempli. Les données qu'il contient proviennent du service web.

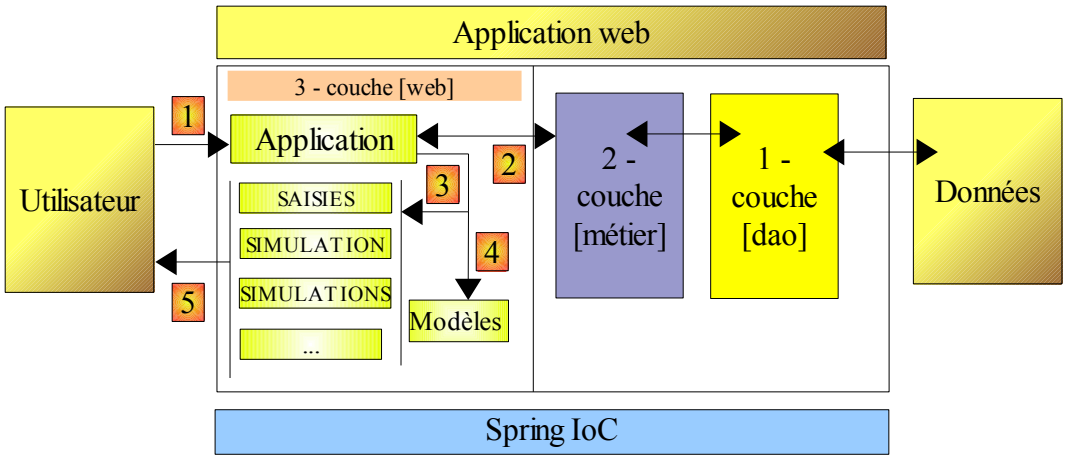
Nous invitons le lecteur à tester cette version 6 copie de la version 4.

11 L'application [SimuPaie] – version 7 – ASP.NET / multi-vues / multi-pages

Lectures conseillées : référence [1], **programmation ASP.NET vol1**, paragraphe 5 : **Exemples**

Nous étudions maintenant une version fonctionnellement identique à l'application ASP.NET à trois couches [pam-v4-3tier-nhibernate-multivues-monopage] étudiée précédemment mais nous modifions l'architecture de cette dernière de la façon suivante : là où dans la version précédente, les vues étaient implémentées par une unique page ASPX, ici elles seront implémentées par trois pages ASPX.

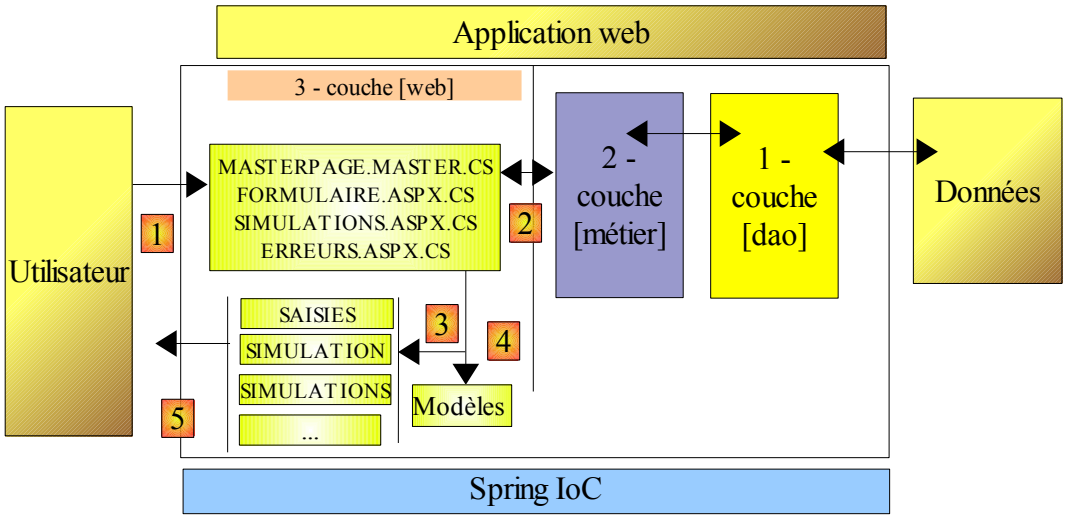
L'architecture de l'application précédente était la suivante :



On a ici une architecture MVC (Modèle – Vue – Contrôleur) :

- [Default.aspx.cs] contient le code du **contrôleur**. La page [Default.aspx] est l'unique interlocuteur du client. Elle voit passer toutes les requêtes de celui-ci.
- [Saisies, Simulation, Simulations, ...] sont les **vues**. Ces vues sont implémentées ici, par des composants [View] de la page [Default.aspx].

L'architecture de la nouvelle version sera elle la suivante :



- seule la couche [web] évolue
- les vues (ce qui est présenté à l'utilisateur) ne changent pas.

- le code du contrôleur, qui était dans la version précédente, tout entier dans [Default.aspx.cs] est désormais réparti sur plusieurs pages :
  - [MasterPage.master] : une page qui factorise ce qui est commun aux différentes vues : le bandeau supérieur avec ses options de menu
  - [Formulaire.aspx] : la page qui présente le formulaire de simulation et gère les actions qui ont lieu sur ce formulaire
  - [Simulations.aspx] : la page qui présente la liste des simulations et gère les actions qui ont lieu sur cette même page
  - [Erreurs.aspx] : la page qui est affichée lors d'une erreur d'initialisation de l'application. Il n'y a pas d'actions possibles sur cette page.

On peut considérer qu'on a là une architecture MVC à contrôleurs multiples alors que l'architecture de la version précédente était une architecture MVC à contrôleur unique.

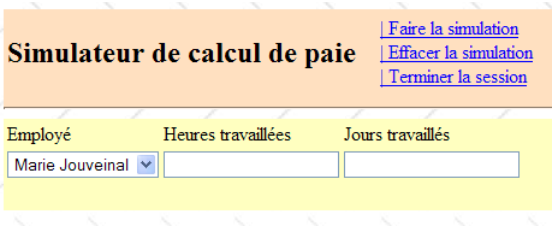
Le traitement d'une demande d'un client se déroule selon les étapes suivantes :

A) le client fait une demande à l'application. Il la fait à l'une des deux pages [Formulaire.aspx, Simulations.aspx].
B) la page demandée traite cette demande. Pour ce faire, elle peut avoir besoin de l'aide de la couche [métier] qui elle-même peut avoir besoin de la couche [dao] si des données doivent être échangées avec la base de données. L'application reçoit une réponse de la couche [métier].
C) selon celle-ci, elle choisit (3) la vue (= la réponse) à envoyer au client en lui fournissant (4) les informations (le modèle) dont elle a besoin.
D) la réponse est envoyée au client (5)

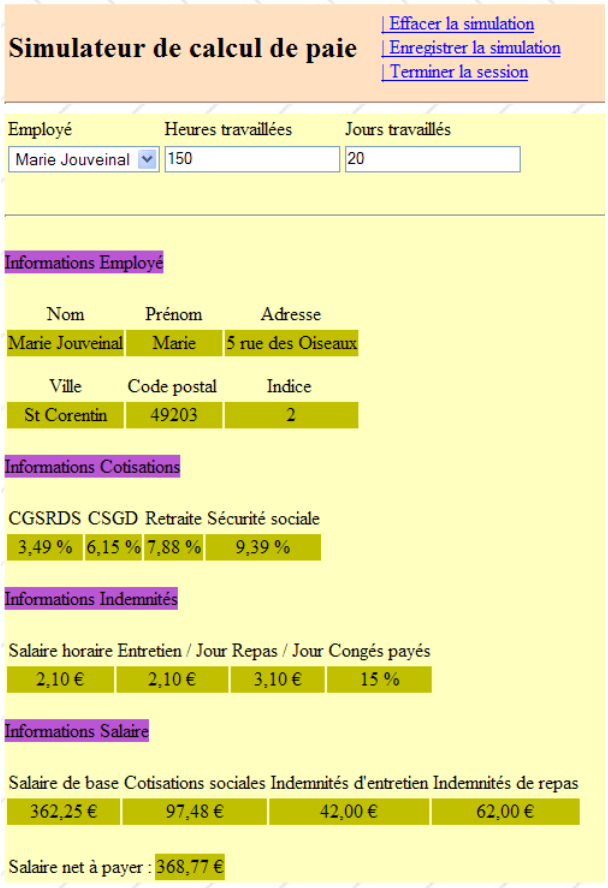
11.1 Les vues de l'application

Les différentes vues présentées à l'utilisateur sont les suivantes :

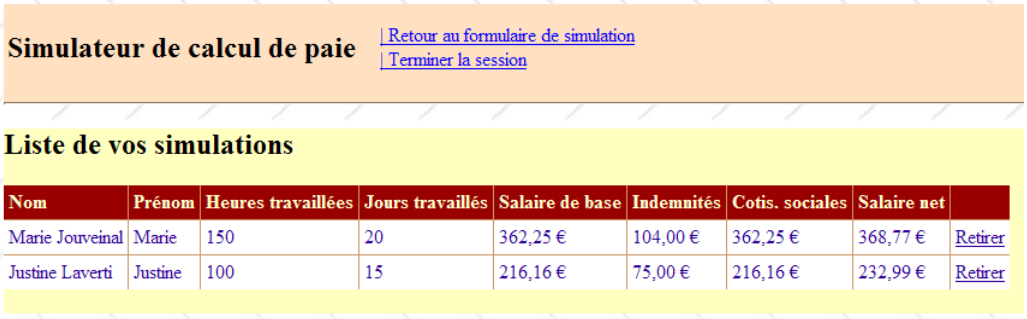
- la vue [VueSaisies] qui présente le formulaire de simulation



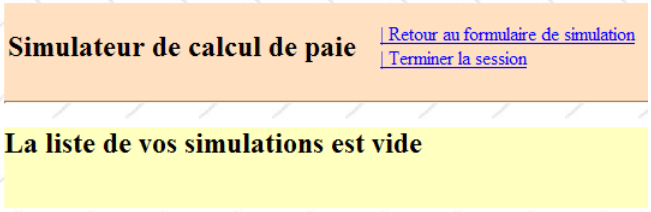
- la vue [VueSimulation] utilisée pour afficher le résultat détaillé de la simulation :



- la vue [VueSimulations] qui donne la liste des simulations faites par le client



- la vue [VueSimulationsVides] qui indique que le client n'a pas ou plus de simulations :



- la vue [VueErreurs] qui indique une erreur d'initialisation de l'application :

Simulateur de calcul de paie

Les erreurs suivantes se sont produites au démarrage de l'application

- Spring.Objects.Factory.ObjectCreationException: Error thrown by a dependency of object 'pammetier' defined in 'file [C:\data\2006-2007\aspnet\pam\v1-3tier-multipages\spring-config.xml]' : Initialization of object failed : Cannot instantiate Type [istia.st.pam.dao.service.PamDaoSqlMap] using ctor [Void .ctor()] : 'Exception has been thrown by the target of an invocation.' while resolving 'PamDao' to 'pamdao' defined in 'file [C:\data\2006-2007\aspnet\pam\v1-3tier-multipages\spring-config.xml]' ---> Spring.Objects.FatalObjectException: Cannot instantiate Type [istia.st.pam.dao.service.PamDaoSqlMap] using ctor [Void .ctor()] : 'Exception has been thrown by the target of an invocation.' ---> System.Reflection.TargetInvocationException: Exception has been thrown by the target of an invocation. ---> istia.st.pam.dao.entites.PamException: Erreur d'accès à la BD lors de la demande des cotisations: [IBatisNet.DataMapper.Exceptions.DataMapperException: Unable to open connection to "Microsoft SQL Server, provider V2.0.0.0 in framework .NET V2.0". ---> System.Data.SqlClient.SqlException: An error has occurred while establishing a

11.2 Génération des vues dans un contexte multi-contrôleurs

Dans la version précédente, toutes les vues étaient générées à partir de l'unique page [Default.aspx]. Celle-ci contenait deux composants [MultiView] et les vues étaient composées d'une réunion d'un ou deux composants [View] appartenant à ces deux composants [MultiView].

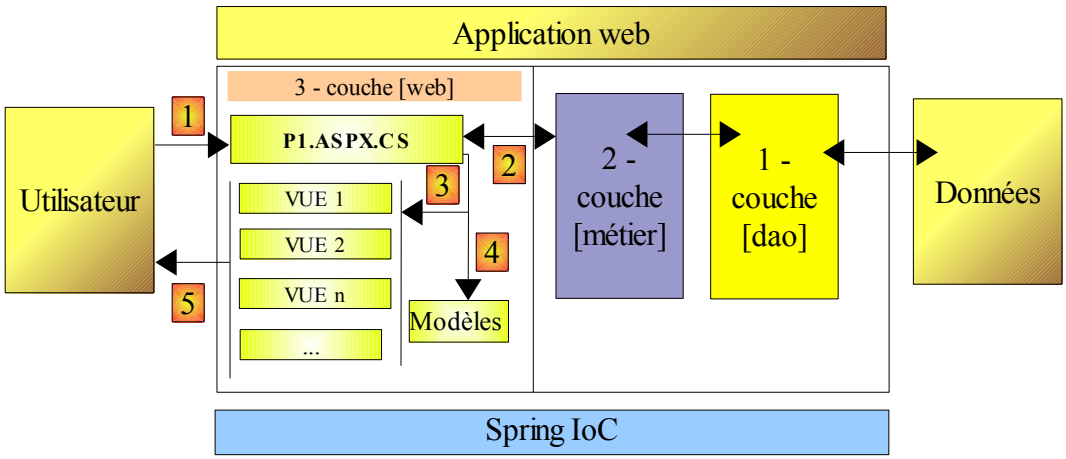
Efficace lorsqu'il y a peu de vues, cette architecture atteint ses limites dès que le nombre des composants formant les différentes vues devient important : en effet, à chaque requête qui est faite à l'unique page [Default.aspx], tous les composants de celle-ci sont instanciés alors même que seuls certains d'entre-eux vont être utilisés pour générer la réponse à l'utilisateur. Un travail inutile est alors fait à chaque nouvelle requête, travail qui devient pénalisant lorsque le nombre total de composants de la page est important.

Une solution est alors de répartir les vues sur différentes pages. C'est ce que nous faisons ici. Etudions deux cas différents de génération de vues :

1. la requête est faite à une page P1 et celle-ci génère la réponse
2. la requête est faite à une page P1 et celle-ci demande à une page P2 de générer la réponse

11.2.1 Cas 1 : une page contrôleur / vue

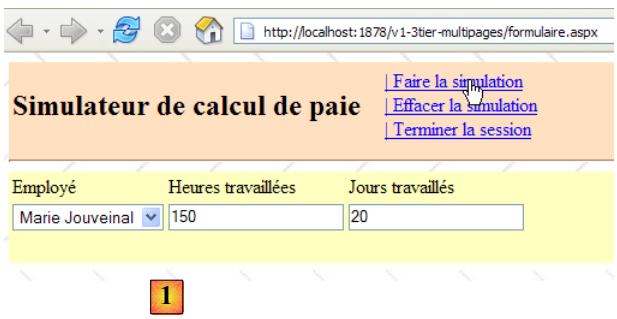
Dans le cas 1, on retombe sur l'architecture mono-contrôleur de la version précédente, où la page [Default.aspx] est la page P1 :



A) le client fait une demande à la page P1 (1)
B) la page P1 traite cette demande. Pour ce faire, elle peut avoir besoin de l'aide de la couche [métier] (2) qui elle-même peut avoir besoin de la couche [dao] si des données doivent être échangées avec la base de données. L'application reçoit une réponse de la couche [métier].
C) selon celle-ci, elle choisit (3) la vue (= la réponse) à envoyer au client en lui fournissant (4) les informations (le modèle) dont elle a besoin. Il s'agit ici, de choisir dans la page P1 les composants [Panel] ou [View] à afficher et d'initialiser les composants qu'ils contiennent.
D) la réponse est envoyée au client (5)

Voici deux exemples pris dans l'application étudiée :

[**page Formulaire.aspx**]



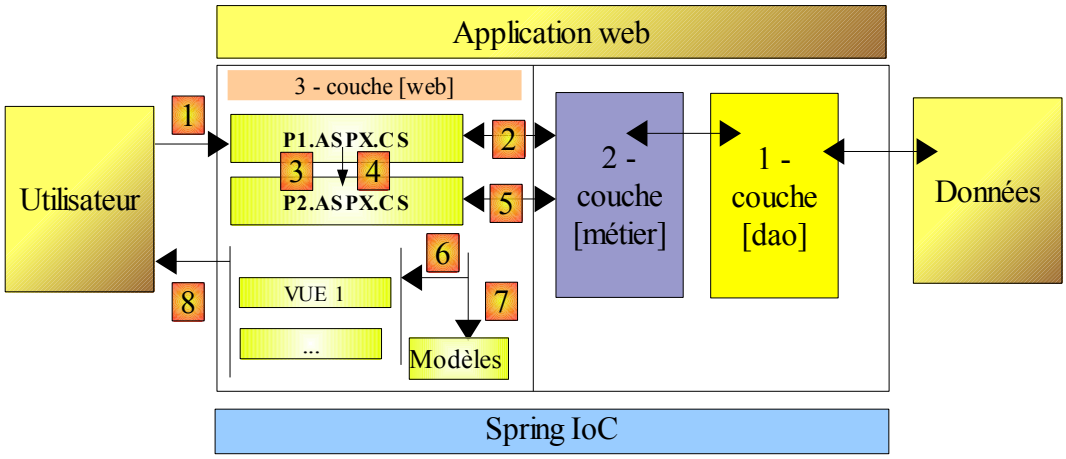
- en [1] : l'utilisateur, après avoir demandé la page [Formulaire.aspx], demande une simulation
- en [2] : la page [Formulaire.aspx] a traité cette demande et généré elle-même la réponse en affichant un composant [View] qui n'avait pas été affiché en [1]

[**page Simulations.aspx**]

- en [1] : l'utilisateur, après avoir demandé la page [Simulations.aspx], veut retirer une simulation
- en [2] : la page [Simulations.aspx] a traité cette demande et généré elle-même la réponse en réaffichant la nouvelle liste de simulations.

11.2.2 Cas 2 : une page 1 contrôleur, une page 2 controleur / vue

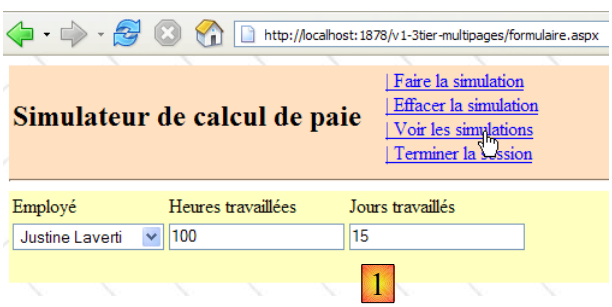
Le cas 2 peut recouvrir diverses d'architectures. Nous choisirons la suivante :

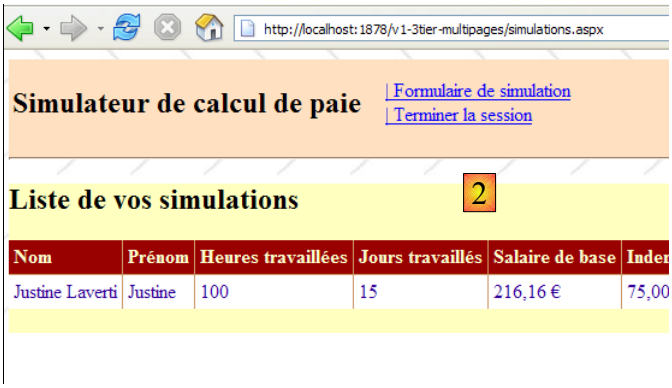


A) le client fait une demande à la page P1 (1)
B) la page P1 traite cette demande. Pour ce faire, elle peut avoir besoin de l'aide de la couche [métier] (2) qui elle-même peut avoir besoin de la couche [dao] si des données doivent être échangées avec la base de données. L'application reçoit une réponse de la couche [métier].
C) selon celle-ci, elle choisit (3) la vue (= la réponse) à envoyer au client en lui fournissant (4) les informations (le modèle) dont elle a besoin. Il se trouve qu'ici, la vue à générer doit l'être par une autre page que P1, la page P2. Pour faire les opérations (3) et (4), la page P1 a deux possibilités :
   - ✗ **faire un transfert d'exécution à la page P2** par l'opération [**Server.Transfer(" P2.aspx ")**]. Dans ce cas, elle peut mettre le modèle destiné à la page P2 dans le contexte de la requête [*Context.Items[" clé "]=valeur*] ou dans la session de l'utilisateur [*Session.[" clé "]=valeur*]. La page P2 sera alors instanciée et lors du traitement de son événement *Load* par exemple, elle pourra récupérer les informations transmises par la page P1 par les opérations [*valeur=(Type)Context.Items[" clé "]*] ou bien [*valeur=(Type)Session[" clé "]*] selon les cas, où *Type* est le type de la valeur associée à la clé. La transmission de valeurs par le contexte *Context* est la plus appropriée s'il n'y a pas utilité à ce que les valeurs du modèle soient conservées pour une future requête du client.
   - ✗ demander au client de **se rediriger vers la page P2** par l'opération [**Response.Redirect(" P2.aspx ")**]. Dans ce cas, la page P1 mettra le modèle destiné à la page P2 dans la session, car le contexte *Context* de requête est supprimé à la fin de chaque requête. Or ici, la redirection va provoquer la fin de la 1ère requête du client vers P1 et l'émission d'une seconde requête de ce même client, vers P2 cette fois. Il y a deux requêtes successives. On sait que la session est l'un des moyens de conserver de la " mémoire " entre requêtes. Il y a d'autres solutions que la session.

D) quelque soit la façon dont P2 prend la main, on retombe ensuite dans le cas 1 : P2 a reçu une requête qu'elle va traiter (5) et elle va générer elle-même la réponse (6, 7). On peut aussi imaginer que la page P2 va après traitement de la requête, passer la main à une page P3, et ainsi de suite.

Voici un exemple pris dans l'application étudiée :

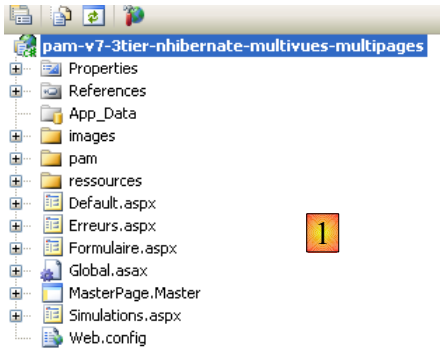


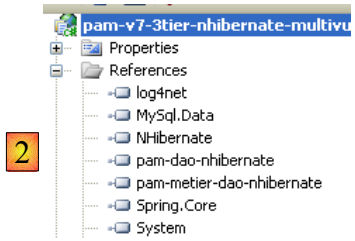


- en [1] : l'utilisateur qui a demandé la page [Formulaire.aspx] demande à voir la liste des simulations
- en [2] : la page [Formulaire.aspx] traite cette demande et redirige le client vers la page [Simulations.aspx]. C'est cette dernière qui fournit la réponse à l'utilisateur. Au lieu de demander au client de se rediriger, la page [Formulaire.aspx] aurait pu transférer la demande du client vers la page [Simulations.aspx]. Dans ce cas en [2], on aurait vu la même Url qu'en [1]. En effet, un navigateur affiche toujours la dernière Url demandée :
  - l'action demandée en [1] est destinée à la page [Formulaire.aspx]. Le navigateur fait un POST vers cette page.
  - si la page [Formulaire.aspx] traite la demande puis la transfère par [Server.Transfer(" Simulations.aspx ")] à la page [Simulations.aspx], on reste dans la même requête. Le navigateur affichera alors en [2], l'Url de [Formulaire.aspx] vers qui a eu lieu le POST.
  - si la page [Formulaire.aspx] traite la demande puis la redirige par [Response.Redirect(" Simulations.aspx ")] vers la page [Simulations.aspx], le navigateur fait alors une 2ième requête, un GET vers [Simulations.aspx]. Le navigateur affichera alors en [2], l'Url de [Simulations.aspx] vers qui a eu lieu le GET. C'est ce que la copie d'écran [2] ci-dessus nous montre.

11.3 Le projet Visual Web Developer de la couche [web]

Le projet Visual Web Developer de la couche [web] est le suivant :



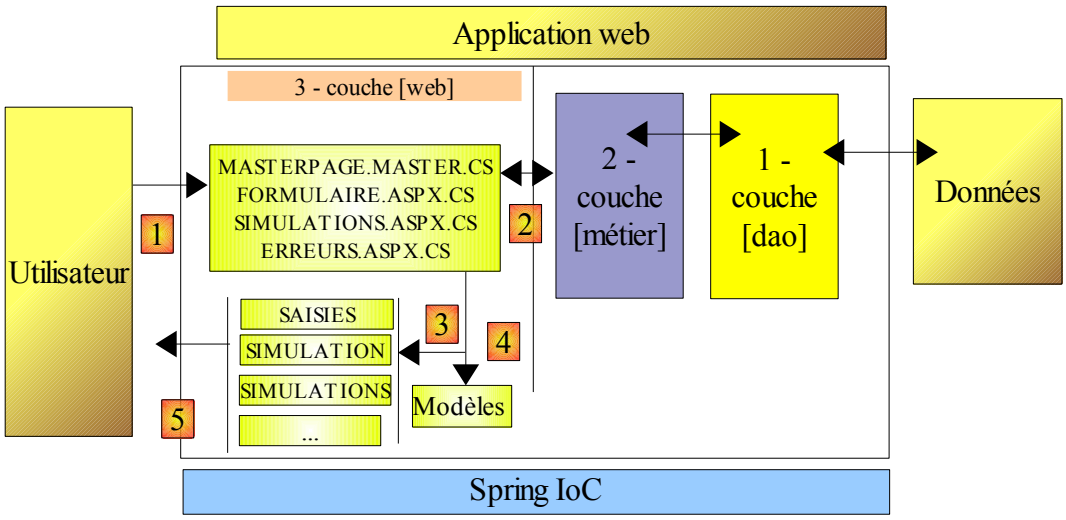


- en [1] on trouve :
  - le fichier de configuration [Web.config] de l'application – est identique à celui de l'application [pam-v4-3tier-nhibernate-multivues-monopage].
  - la page [Default.aspx] – se contente de rediriger le client vers la page [Formulaire.aspx]
  - la page [Formulaire.aspx] qui présente à l'utilisateur le formulaire de simulation et traite les actions liées à ce formulaire
  - la page [Simulations.aspx] qui présente à l'utilisateur la liste de ses simulations et traite les actions liées à cette page

- la page [Erreurs.aspx] qui présente à l'utilisateur une page signalant une erreur rencontrée au démarrage de l'application web.

– en [2] on voit les références du projet.

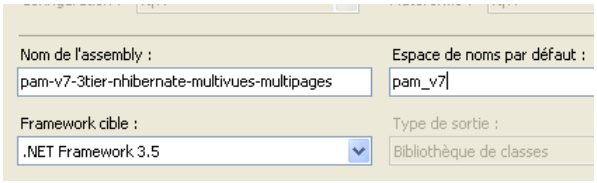
Revenons à l'architecture du nouveau projet :



Vis à vis du projet [pam-v4-3tier-nhibernate-multivues-monopage], seules changent les vues. C'est ainsi que le nouveau projet reprend certains des fichiers de ce projet :

- le fichier de configuration [Web.config]
- les Dll référencées [pam-dao-nhibernate, pam-metier-dao-nhibernate, Spring.Core, NHibernate]
- la classe globale d'application [Global.asax]
- les dossiers [images, ressources, pam]

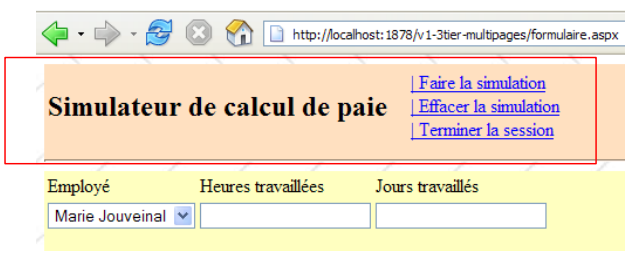
Pour être cohérent avec le projet en cours de construction, on fera en sorte que l'espace de noms des vues et de la classe globale d'application soit [pam-v7] :



11.4 Le code de présentation des pages

11.4.1 La page maître [MasterPage.master]

Les vues de l'application présentées au paragraphe 11.1, page 162, ont des parties communes qu'on peut factoriser dans une page Maître, appelée la **Master Page** dans Visual Studio. Prenons par exemple, les vues [VueSaisies] et [VueSimulationsVides] ci-dessous, générées respectivement par les pages [Formulaire.aspx] et [Simulations.aspx] :

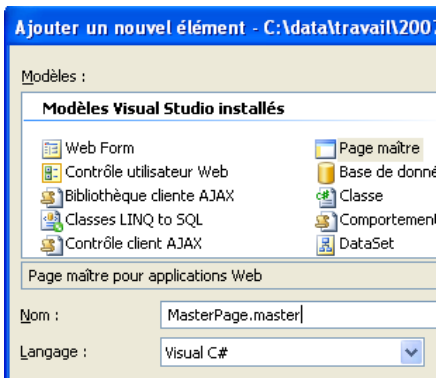


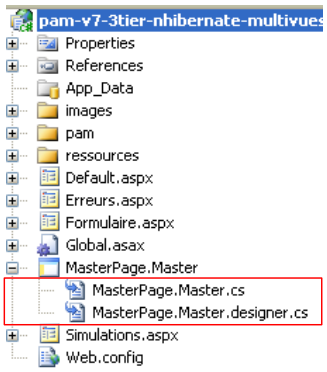
Ces deux vues possèdent en commun, le bandeau supérieur (Titre et Options de menu). Il en est ainsi de toutes les vues qui seront présentées à l'utilisateur : elles auront toutes le même bandeau supérieur. Pour que différentes pages partagent un même fragment de présentation, il existe diverses solutions dont les suivantes :

- mettre ce fragment commun dans un composant utilisateur. C'était la principale technique avec ASP.NET 1.1
- mettre ce fragment commun dans une page Maître. Cette technique est apparue avec ASP.NET 2.0. C'est celle que nous utilisons ici.

Pour créer une page Maître dans une application web, on peut procéder ainsi :

- clic droit sur le projet/ Ajouter un nouvel élément / Page maître :





L'ajout d'une page maître ajoute par défaut trois fichiers à l'application web :

- [MasterPage.master] : le code de présentation de la page Maître
- [MasterPage.master.cs] : le code de contrôle de la page Maître
- [Masterpage.Master.designer.cs] : la déclaration des composants de la page maître

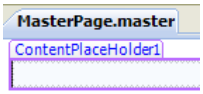
Le code généré par Visual Studio dans [MasterPage.master] est le suivant :

```
1.  <%@ Master Language="C#" AutoEventWireup="true" CodeBehind="MasterPage.master.cs"
    Inherits="pam_v7.MasterPage" %>
2.
3.  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4.
5.  <html xmlns="http://www.w3.org/1999/xhtml">
6.  <head runat="server">
7.      <title>Untitled Page</title>
8.      <asp:ContentPlaceHolder id="head" runat="server">
9.      </asp:ContentPlaceHolder>
10. </head>
11. <body>
12.     <form id="form1" runat="server">
13.     <div>
14.         <asp:ContentPlaceHolder id="ContentPlaceHolder1" runat="server">
15.
16.         </asp:ContentPlaceHolder>
17.     </div>
18.     </form>
19. </body>
20. </html>
```

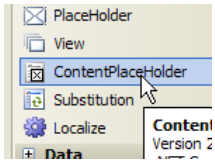
- ligne 1 : la balise <%@ Master ... %> sert à définir la page comme une page Maître. Le code de contrôle de la page sera dans le fichier défini par l'attribut *CodeBehind*, et la page héritera de la classe définie par l'attribut *Inherits*.
- lignes 12-18 : le formulaire de la page Maître
- lignes 14-16 : un conteneur vide qui contiendra dans notre application, l'une des pages [Formulaire.aspx, Simulations.aspx, Erreurs.aspx]. Le client reçoit en réponse, toujours la même page, la page Maître, dans laquelle le conteneur [ContentPlaceHolder1] va recevoir un flux HTML fourni par l'une des pages [Formulaire.aspx, Simulations.aspx, Erreurs.aspx]. Ainsi pour changer l'aspect des pages envoyées aux clients, il suffit de changer l'aspect de la page Maître.
- lignes 8-9 : un conteneur vide avec lequel les pages "fille" pourront personnaliser l'entête <head>...</head>.

La représentation visuelle (onglet Design) de ce code source est présentée en (1) ci-dessous. Par ailleurs, il est possible d'ajouter autant de conteneurs que souhaité, grâce au composant [ContentPlaceHolder] (2) de la barre d'outils [Standard].

1



2



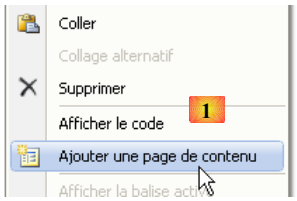
Le code de contrôle généré par Visual Studio dans [MasterPage.master.cs] est le suivant :

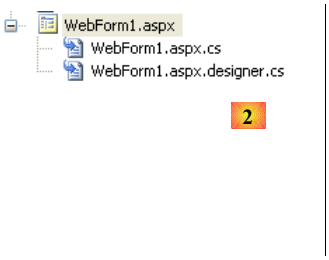
```
1.  using System;
2.
3.  public partial class MasterPage : System.Web.UI.MasterPage
4.  {
5.      protected void Page_Load(object sender, EventArgs e)
6.      {
7.
8.      }
9.  }
```

- ligne 3 : la classe référencée par l'attribut [Inherits] de la directive <%@ Master ... %> de la page [MasterPage.master] dérive de la classe [System.Web.UI.MasterPage]

Ci-dessus, nous voyons la présence de la méthode *Page_Load* qui gère l'événement *Load* de la page maître. La page maître contiendra en son sein une autre page. Dans quel ordre se produisent les événements *Load* des deux pages ? Il s'agit là d'une règle générale : l'événement *Load* d'un composant se produit avant celui de son conteneur. Ici, l'événement *Load* de la page insérée dans la page maître aura donc lieu avant celui de la page maître elle-même.

Pour générer une page qui a pour page maître la page [MasterPage.master] précédente, on pourra procéder comme suit :





– en [1] : clic droit sur la page maître puis option [Ajouter une page de contenu]
– en [2] : une page par défaut, ici [WebForm1.aspx] est générée.

Le code de présentation [WebForm1.aspx] est le suivant :

```
1.  <%@ Page Title="" Language="C#" MasterPageFile="~/MasterPage.Master" AutoEventWireup="true"
    CodeBehind="WebForm1.aspx.cs" Inherits="pam_v7.WebForm1" %>
2.  <asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">
3.  </asp:Content>
```

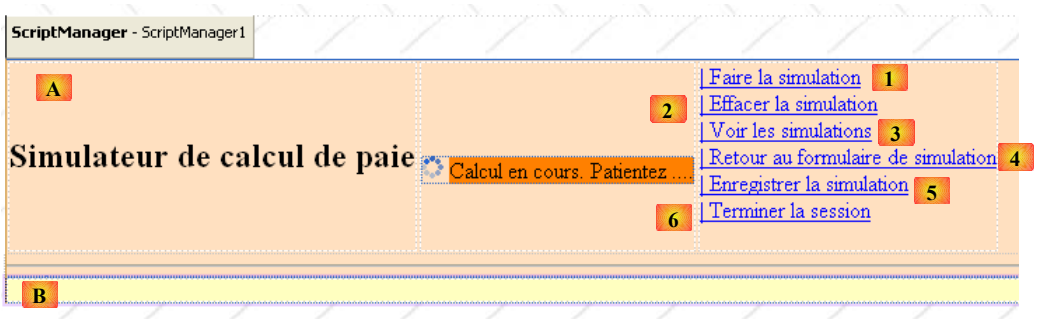
- ligne 1 : la directive **Page** et ses attributs

- *MasterPageFile* : désigne le fichier de la page maître de la page décrite par la directive. Le signe ~ désigne le dossier du projet.
- les autres paramètres sont ceux habituels d'une page web ASP

- lignes 2-3 : les balises <*asp:Content*> sont reliées une à une aux directives <*asp:ContentPlaceHolder*> de la page maître via l'attribut *ContentPlaceHolderID*. Les composants placés entre les lignes 2-3 ci-dessus, seront à l'exécution, placés dans le conteneur d'ID *ContentPlaceHolder1* de la page maître.

En renommant la page [WebForm1.aspx] ainsi générée, on peut construire les différentes pages ayant [MasterPage.master] comme page maître.

Pour notre application [SimuPaie], l'aspect visuel de la page maître sera la suivante :



N°	Type	Nom	Rôle
A	Panel (rose ci-dessus)	entete	entête de la page
B	Panel (jaune ci-dessus)	contenu	contenu de la page
1	LinkButton	LinkButtonFaireSimulation	demande le calcul de la simulation
2	LinkButton	LinkButtonEffacerSimulation	efface le formulaire de saisie
3	LinkButton	LinkButtonVoirSimulations	affiche la liste des simulations déjà faites
4	LinkButton	LinkButtonFormulaireSimulation	ramène au formulaire de saisie
5	LinkButton	LinkButtonEnregistrerSimulation	enregistre la simulation courante dans la liste des simulations
6	LinkButton	LinkButtonTerminerSession	abandonne la session courante

Le code source correspondant est le suivant :

```
1.  <%@ Master Language="C#" AutoEventWireup="true" CodeBehind="MasterPage.master.cs"
    Inherits="pam_v7.MasterPage" %>
2.
3.  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4.  <html xmlns="http://www.w3.org/1999/xhtml">
5.  <head id="Head1" runat="server">
6.    <title>Application PAM</title>
7.  </head>
8.  <body background="ressources/standard.jpg">
9.    <form id="form1" runat="server">
10.   <asp:ScriptManager ID="ScriptManager1" runat="server" EnablePartialRendering="true" />
11.   <asp:UpdatePanel runat="server" ID="UpdatePanelPam" UpdateMode="Conditional">
12.     <ContentTemplate>
13.       <asp:Panel ID="entete" runat="server" BackColor="#FFE0C0">
14.         <table>
15.           <tr>
16.             <td>
17.               <h2>
18.                 Simulateur de calcul de paie</h2>
19.             </td>
20.             <td>
21.               <label>
22.                    </label>
23.               <asp:UpdateProgress ID="UpdateProgress1" runat="server">
24.                 <ProgressTemplate>
```

```
25.                   <img alt="" src="images/indicator.gif" />
26.                   <asp:Label ID="Label5" runat="server" BackColor="#FF8000"
27.                    EnableViewState="False" Text="Calcul en cours. Patientez ....">
28.                  </asp:Label>
29.                  </ProgressTemplate>
30.                </asp:UpdateProgress>
31.              </td>
32.              <td>
33.                <asp:LinkButton ID="LinkButtonFaireSimulation" runat="server"
34.                     CausesValidation="False">| Faire la simulation<br />
35.               </asp:LinkButton>
36.                <asp:LinkButton ID="LinkButtonEffacerSimulation" runat="server"
37.                     CausesValidation="False">| Effacer la simulation<br />
38.               </asp:LinkButton>
39.                <asp:LinkButton ID="LinkButtonVoirSimulations" runat="server"
40.                 CausesValidation="False">| Voir les simulations<br />
41.               </asp:LinkButton>
42.                <asp:LinkButton ID="LinkButtonFormulaireSimulation" runat="server"
43.                     CausesValidation="False">| Retour au formulaire de simulation<br />
44.               </asp:LinkButton>
45.                <asp:LinkButton ID="LinkButtonEnregistrerSimulation" runat="server"
46.                     CausesValidation="False">| Enregistrer la simulation<br />
47.               </asp:LinkButton>
48.                <asp:LinkButton ID="LinkButtonTerminerSession" runat="server"
49.                     CausesValidation="False">| Terminer la session<br />
50.               </asp:LinkButton>
51.              </td>
52.            </tr>
53.          </table>
54.          <hr />
55.        </asp:Panel>
56.        <div>
57.          <asp:Panel ID="contenu" runat="server" BackColor="#FFFFC0">
58.            <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
59.            </asp:ContentPlaceHolder>
60.          </asp:Panel>
61.        </div>
62.      </ContentTemplate>
63.    </asp:UpdatePanel>
64.    </form>
65. </body>
66. </html>
```

- ligne 1 : on notera le nom de la classe de la page maître : **MasterPage**
- ligne 8 : on définit une image de fond pour la page.
- lignes 9-64 : le formulaire
- ligne 10 : le composant ScriptManager nécessaire aux effets Ajax
- lignes 11-63 : le conteneur AJax
- lignes 12-62 : le contenu ajaxifié
- lignes 13-55 : le composant Panel [entete]
- lignes 57-60 : le composant Panel [contenu]
- lignes 58-59 : le composant d'ID [ContentPlaceHolder1] qui contiendra la page encapsulée [Formulaire.aspx, Simulations.aspx, Erreurs.aspx]

Pour construire cette page, on pourra insérer dans le panel [entete], le code ASPX de la vue [VueEntete] de la page [Default.aspx] de la version [pam-v4-3tier-nhibernate-multivues-monopage], décrite au paragraphe 8.5.2, page 120.

11.4.2 La page [Formulaire.aspx]

Pour générer cette page, on suivra la méthode exposée page 170 et on renommera [Formulaire.aspx] la page [WebForm1.aspx] ainsi générée. L'aspect visuel de la page [Formulaire.aspx] en cours de construction sera le suivant :

L'aspect visuel de la page [Formulaire.aspx] a deux éléments :

- en [1] la page maître avec son conteneur [ContentPlaceHolder1] (2)
- en [2] les composants placés dans le conteneur [ContentPlaceHolder1]. Ceux-ci sont identiques à ceux de l'application précédente.

Le code source de cette page est le suivant :

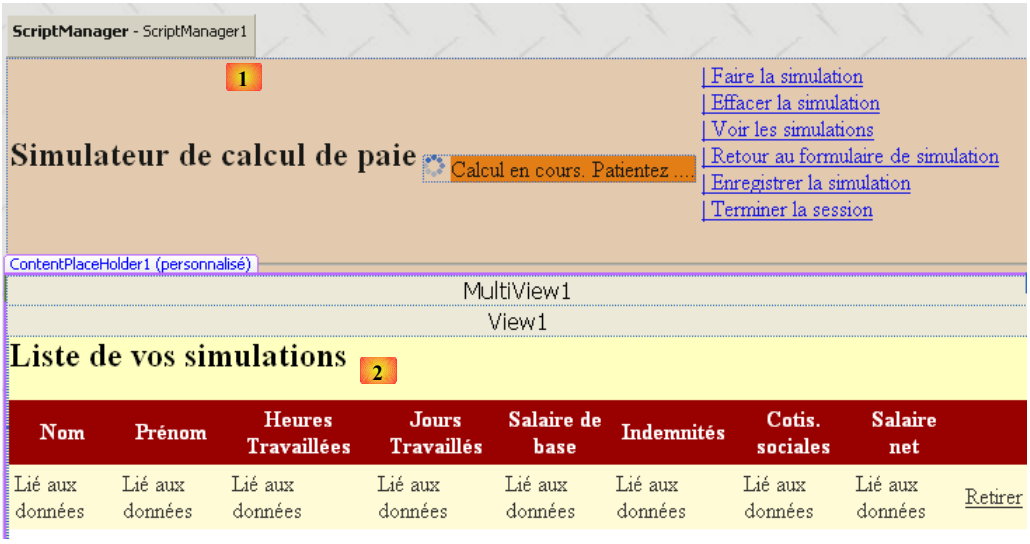
```
1.  <%@ Page Language="C#" MasterPageFile="~/MasterPage.master" AutoEventWireup="true"
2.    CodeBehind="Formulaire.aspx.cs" Inherits="pam_v7.PageFormulaire" Title="Simulation de calcul de
    paie : formulaire" %>
3.
4.  <%@ MasterType VirtualPath="~/MasterPage.master" %>
5.  <asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" runat="Server">
6.    <div>
7.      <table>
8.        <tr>
9.          <td>
10.           Employé
11.         </td>
12.         <td>
13.           Heures travaillées
14.         </td>
15.         <td>
16.           Jours travaillés
17.         </td>
18.         <td>
19.         </td>
20.       </tr>
21. ...
22. </asp:Content>
```

- ligne 1 : la directive *Page* avec son attribut *MasterPageFile*
- ligne 4 : la classe de contrôle de la page maître peut exposer des champs et propriétés **publics**. Ceux-ci sont accessibles aux pages encapsulées avec la syntaxe **Master.[champ]** ou **Master.[propriété]**. La propriété **Master** de la page désigne la page maître sous la forme d'une instance de type [System.Web.UI.MasterPage]. Aussi dans notre exemple, faudrait-il écrire en réalité *(MasterPage)(Master).[champ]* ou *(MasterPage)(Master).[propriété]*. On peut éviter ce transtypage en insérant dans la page la directive *MasterType* de la ligne 4. L'attribut *VirtualPath* de cette directive indique le fichier de la page maître. Le compilateur peut alors connaître les champs, propriétés et méthodes publics exposés par la classe de la page maître, ici de type [MasterPage].
- lignes 5-22 : le contenu qui sera inséré dans le conteneur [ContentPlaceHolder1] de la page maître.

On pourra construire cette page en mettant comme contenu (lignes 6-21), celui de la vue [VueSaisies] décrite au paragraphe 8.5.3 de la page 121 et celui de la vue [VueSimulation] décrite au paragraphe 8.5.4 de la page 121.

11.4.3 La page [Simulations.aspx]

Pour générer cette page, on suivra la méthode exposée page 170 et on renommera [Simulations.aspx] la page [WebForm1.aspx] ainsi générée. L'aspect visuel de la page [Simulations.aspx] en cours de construction est le suivant :



L'aspect visuel de la page [Simulations.aspx] a deux éléments :

- en [1] la page maître avec son conteneur [ContentPlaceHolder1]
- en [2] les composants placés dans le conteneur [ContentPlaceHolder1]. Ceux-ci sont identiques à ceux de l'application précédente.

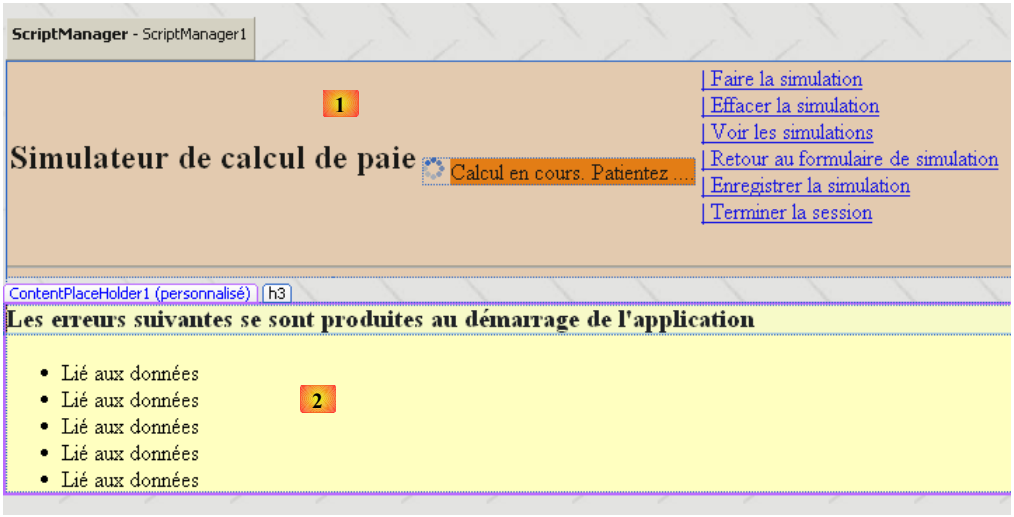
Le code source de cette page est le suivant :

```
1.  <%@ Page Language="C#" MasterPageFile="~/MasterPage.master" AutoEventWireup="true"
2.    CodeBehind="Simulations.aspx.cs" Inherits="pam_v7.PageSimulations" Title="Pam : liste des
    simulations" %>
3.
4.  <%@ MasterType VirtualPath="~/MasterPage.master" %>
5.  <asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" runat="Server">
6.    <asp:MultiView ID="MultiView1" runat="server">
7.      <asp:View ID="View1" runat="server">
8.        <h2>
9.          Liste de vos simulations</h2>
10.       <p>
11.         <asp:GridView ID="GridViewSimulations" runat="server" ...>
12. ...
13.         </asp:GridView>
14.       </p>
15.     </asp:View>
16.     <asp:View ID="View2" runat="server">
17.       <h2>
18.         La liste de vos simulations est vide</h2>
19.     </asp:View>
20.   </asp:MultiView><br />
21. </asp:Content>
```

On pourra construire cette page en mettant comme contenu (lignes 5-21), celui de la vue [VueSimulations] décrite au paragraphe 8.5.5 de la page 122 et celui de la vue [VueSimulationsVides] décrite au paragraphe 8.5.6 de la page 124.

11.4.4 La page [Erreurs.aspx]

Pour générer cette page, on suivra la méthode exposée page 170 et on renommera [Erreurs.aspx] la page [WebForm1.aspx] ainsi générée. L'aspect visuel de la page [Erreurs.aspx] en cours de construction est le suivant :



L'aspect visuel de la page [Erreurs.aspx] a deux éléments :

- en [1] la page maître avec son conteneur [ContentPlaceHolder1]
- en [2] les composants placés dans le conteneur [ContentPlaceHolder1]. Ceux-ci sont identiques à ceux de l'application précédente.

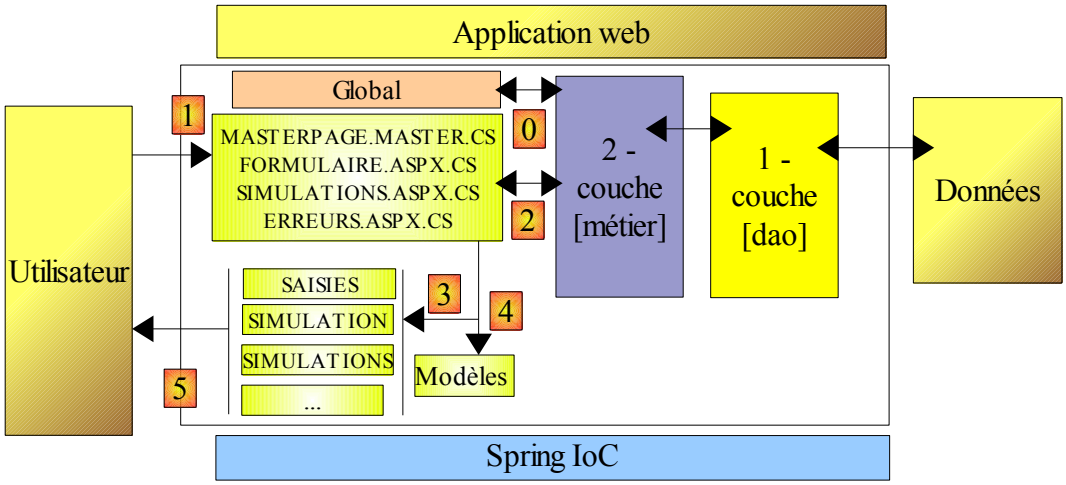
Le code source de cette page est le suivant :

```
1.  <%@ Page Language="C#" MasterPageFile="~/MasterPage.master" AutoEventWireup="true"
2.    CodeBehind="Erreurs.aspx.cs" Inherits="pam_v7.PageErreurs" Title="Pam : erreurs" %>
3.
4.  <%@ MasterType VirtualPath="~/MasterPage.master" %>
5.  <asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
6.      <h3>Les erreurs suivantes se sont produites au démarrage de l'application</h3>
7.      <ul>
8.          <asp:Repeater id="rptErreurs" runat="server">
9.              <ItemTemplate>
10.                 <li>
11.                     <%# Container.DataItem %>
12.                 </li>
13.             </ItemTemplate>
14.         </asp:Repeater>
15.     </ul>
16. </asp:Content>
```

11.5 Le code de contrôle des pages

11.5.1 Vue d'ensemble

Revenons à l'architecture de l'application :



- [Global] est l'objet de type [HttpApplication] qui initialise (étape 0) l'application. Cette classe est identique à celle de la version précédente.
- le code du contrôleur, qui était dans la version précédente, tout entier dans [Default.aspx.cs] est désormais réparti sur plusieurs pages :
  - [MasterPage.master] : la page maître des pages [Formulaire.aspx, Simulations.aspx, Erreurs.aspx]. Elle contient le menu.
  - [Formulaire.aspx] : la page qui présente le formulaire de simulation et gère les actions qui ont lieu sur ce formulaire
  - [Simulations.aspx] : la page qui présente la liste des simulations et gère les actions qui ont lieu sur cette même page
  - [Erreurs.aspx] : la page qui est affichée lors d'une erreur d'initialisation de l'application. Il n'y a pas d'actions possibles sur cette page.

Le traitement d'une demande d'un client se déroule selon les étapes suivantes :

A) le client fait une demande à l'application. Il la fait normalement à l'une des deux pages [Formulaire.aspx, Simulations.aspx], mais rien ne l'empêche de demander la page [Erreurs.aspx]. Il faudra prévoir ce cas.
B) la page demandée traite cette demande (étape 1). Pour ce faire, elle peut avoir besoin de l'aide de la couche [métier] (étape 2) qui elle-même peut avoir besoin de la couche [dao] si des données doivent être échangées avec la base de données. L'application reçoit une réponse de la couche [métier].
C) selon celle-ci, elle choisit (étape 3) la vue (= la réponse) à envoyer au client et lui fournit (étape 4) les informations (le modèle) dont elle a besoin. Nous avons vu trois possibilités pour générer cette réponse :
   - la page (D) demandée est également la page (R) envoyée en réponse. Construire le modèle de la réponse (R) consiste alors à donner à certains des composants de la page (D), la valeur qu'ils doivent avoir dans la réponse.
   - la page (D) demandée n'est pas la page (R) envoyée en réponse. La page (D) peut alors :
     - transférer le flux d'exécution à la page (R) par l'instruction *Server.Transfer(" R ")*. Le modèle peut alors être placé dans le contexte par *Context.Items(" clé ")=valeur* ou plus rarement dans la session par *Session.Items(" clé ")=valeur*
     - rediriger le client vers la page (R) par l'instruction *Response.redirect(" R ")*. Le modèle peut alors être placé dans la session mais pas dans le contexte.
D) la réponse est envoyée au client (étape 5)

Chacune des pages [MasterPage.master, Formulaire.aspx, Simulations.aspx, Erreurs.aspx] répondra à un ou plusieurs des événements ci-dessous :

- **Init** : premier événement dans le cycle de vie de la page
- **Load** : se produit au chargement de la page
- **Click** : le clic sur l'un des liens du menu de la page maître

Nous traitons les pages les unes après les autres en commençant par la page maître.

11.5.2 Code de contrôle de la page [MasterPage.master]

11.5.2.1 Squelette de la classe

Le code de contrôle de la page maître a le squelette suivant :

```
1.  using System.Web.UI.WebControls;
2.
3.  namespace pam_v7
4.  {
5.    public partial class MasterPage : System.Web.UI.MasterPage
6.    {
7.
8.      // le menu
9.      public LinkButton OptionFaireSimulation
10.     {
11.       get { return LinkButtonFaireSimulation; }
12.     }
13. ...
14.
15.     // fixer le menu
16.     public void SetMenu(bool boolFaireSimulation, bool boolEnregistrerSimulation, bool
    boolEffacerSimulation, bool boolFormulaireSimulation, bool boolVoirSimulations, bool
    boolTerminerSession)
17.     {
18. ....
19.     }
20.
21.     // gestion de l'option [Terminer la session]
22.     protected void LinkButtonTerminerSession_Click(object sender, System.EventArgs e)
23.     {
24. ....
25.     }
26.
27.     // init master page
28.     protected void Page_Init(object sender, System.EventArgs e)
29.     {
30. ....
31.       }
32.     }
33.   }
34. }
```

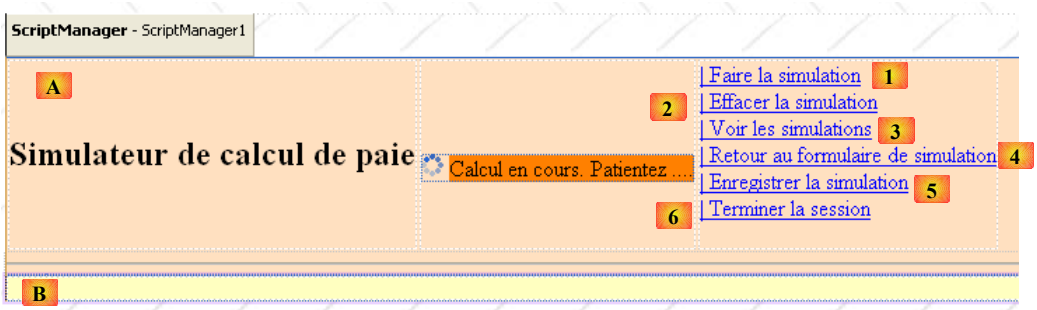
- ligne 5 : la classe s'appelle [MasterPage] et dérive de la classe système [System.Web.UI.MasterPage].
- lignes 9-14 : les 6 options du menu sont exposées comme propriétés publiques de la classe
- lignes 16-19 : la méthode publique *SetMenu* va permettre aux pages [Formulaire.aspx, Simulations.aspx, Erreurs.aspx] de fixer le menu de la page maître
- lignes 22-25 : la procédure qui va gérer le clic sur le lien [LinkButtonTerminerSession]
- lignes 28-31 : la procédure de gestion de l'événement *Init* de la page maître

11.5.2.2 Propriétés publiques de la classe

```
1.  using System.Web.UI.WebControls;
2.
3.  namespace pam_v7
4.  {
5.    public partial class MasterPage : System.Web.UI.MasterPage
6.    {
7.
8.      // le menu
9.      public LinkButton OptionFaireSimulation
10.     {
11.       get { return LinkButtonFaireSimulation; }
12.     }
13.
14.     public LinkButton OptionEffacerSimulation
15.     {
16.       get { return LinkButtonEffacerSimulation; }
17.     }
18.
19.     public LinkButton OptionEnregistrerSimulation
20.     {
21.       get { return LinkButtonEnregistrerSimulation; }
22.     }
23.
24.     public LinkButton OptionVoirSimulations
25.     {
26.       get { return LinkButtonVoirSimulations; }
27.     }
```

```
28.
29.     public LinkButton OptionTerminerSession
30.     {
31.       get { return LinkButtonTerminerSession; }
32.     }
33.
34.     public LinkButton OptionFormulaireSimulation
35.     {
36.       get { return LinkButtonFormulaireSimulation; }
37.     }
38.
39. ...
40.   }
41. }
```

Pour comprendre ce code, il faut se rappeler les composants qui forment la page maître :



N°	Type	Nom	Rôle
A	Panel (rose ci-dessus)	entete	entête de la page
B	Panel (jaune ci-dessus)	contenu	contenu de la page
1	LinkButton	LinkButtonFaireSimulation	demande le calcul de la simulation
2	LinkButton	LinkButtonEffacerSimulation	efface le formulaire de saisie
3	LinkButton	LinkButtonVoirSimulations	affiche la liste des simulations déjà faites
4	LinkButton	LinkButtonFormulaireSimulation	ramène au formulaire de saisie
5	LinkButton	LinkButtonEnregistrerSimulation	enregistre la simulation courante dans la liste des simulations
6	LinkButton	LinkButtonTerminerSession	abandonne la session courante

Les composants 1 à 6 ne sont pas accessibles en-dehors de la page qui les contient. Les propriétés des lignes 9 à 37 visent à les rendre accessibles aux classes externes, ici les classes des autres pages de l'application.

11.5.2.3 La méthode SetMenu

La méthode publique *SetMenu* permet aux pages [Formulaire.aspx, Simulations.aspx, Erreurs.aspx] de fixer le menu de la page maître. Son code est basique :

```
1.          // fixer le menu
2.          public void SetMenu(bool boolFaireSimulation, bool boolEnregistrerSimulation, bool
    boolEffacerSimulation, bool boolFormulaireSimulation, bool boolVoirSimulations, bool
    boolTerminerSession)
3.          {
4.              // on fixe les options de menu
5.              LinkButtonFaireSimulation.Visible = boolFaireSimulation;
6.              LinkButtonEnregistrerSimulation.Visible = boolEnregistrerSimulation;
7.              LinkButtonEffacerSimulation.Visible = boolEffacerSimulation;
8.              LinkButtonVoirSimulations.Visible = boolVoirSimulations;
9.              LinkButtonFormulaireSimulation.Visible = boolFormulaireSimulation;
10.             LinkButtonTerminerSession.Visible = boolTerminerSession;
11. }
```

11.5.2.4 La gestion des événements de la page maître

La page maître va gérer deux événements :

- l'événement *Init* qui est le premier événement du cycle de vie de la page
- l'événement *Click* sur le lien [LinkButtonTerminerSession]

La page maître a cinq autres liens : [LinkButtonFaireSimulation, LinkButtonEnregistrerSimulation, LinkButtonEffacerSimulation, LinkButtonVoirSimulations, LinkButtonFormulaireSimulation]. Comme exemple, examinons ce qu'il faudrait faire lors d'un clic sur le lien [LinkButtonFaireSimulation] :

1. vérifier les données saisies (heures, jours) dans la page [Formulaire.aspx]
2. faire le calcul du salaire
3. afficher les résultats dans la page [Formulaire.aspx]

Les opérations 1 et 3 impliquent d'avoir accès aux composants de la page [Formulaire.aspx]. Ce n'est pas le cas. En effet, la page maître n'a aucune connaissance des composants des pages susceptibles d'être insérées dans son conteneur [ContentPlaceHolder1]. Dans notre exemple, c'est à la page [Formulaire.aspx] de gérer le clic sur le lien [LinkButtonFaireSimulation] car c'est elle qui est affichée lorsqu'a lieu cet événement. Comment peut-elle être avertie de celui-ci ?

- le lien [LinkButtonFaireSimulation] ne faisant pas partie de la page [Formulaire.aspx], on ne peut pas écrire dans [Formulaire.aspx] la procédure habituelle :

```
1.      private void LinkButtonFaireSimulation_Click(object sender, System.EventArgs e)
2.      {
3.  ...
4.  }
```

  On peut contourner le problème avec le code suivant dans [Formulaire.aspx] :

```
1.  using System.Collections.Generic;
2.  ...
3.
4.  namespace pam_v7
5.  {
6.     public partial class Formulaire : System.Web.UI.Page
7.     {
8.  // chargement de la page
9.      protected void Page_Load(object sender, System.EventArgs e)
10.     {
11.         // gestionnaire d'évts
12.         Master.OptionFaireSimulation.Click += OptFaireSimulation_Click;
13.         Master.OptionEffacerSimulation.Click += OptEffacerSimulation_Click;
14.         Master.OptionVoirSimulations.Click += OptVoirSimulations_Click;
15.         Master.OptionEnregistrerSimulation.Click += OptEnregistrerSimulation_Click;
16. ...
17.     }
18.
19.     // calcul de la paie
20.     private void OptFaireSimulation_Click(object sender, System.EventArgs e)
21.     {
22. ....
23.     }
24.
25.     // effacer la simulation
26.     private void OptEffacerSimulation_Click(object sender, System.EventArgs e)
27.     {
28. ...
29.     }
30.
31.     protected void OptVoirSimulations_Click(object sender, System.EventArgs e)
32.     {
33. ...
34.     }
35.
36.     protected void OptEnregistrerSimulation_Click(object sender, System.EventArgs e)
37.     {
38. ...
39.     }
40.  }
41. }
```

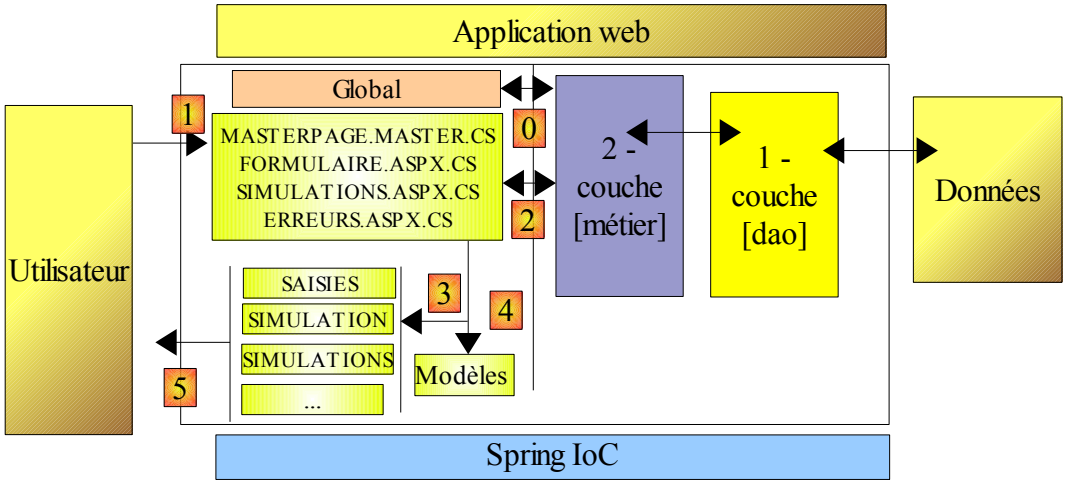
- lignes 12-15 : lorsque l'événement *Load* de la page [Formulaire.aspx] se produit, la classe [MasterPage] de la page maître a été instanciée. Ses propriétés publiques *Optionxx* sont accessibles et sont de type *LinkButton*, un composant qui supporte l'événement *Click*. Nous associons à ces événements *Click* les méthodes :
  - *OptFaireSimulation_Click* pour l'événement *Click* sur le lien *LinkButtonFaireSimulation*
  - *OptEffacerSimulation_Click* pour l'événement *Click* sur le lien *LinkButtonEffacerSimulation*
  - *OptVoirSimulations_Click* pour l'événement *Click* sur le lien *LinkButtonVoirSimulations*
  - *OptEnregistrerSimulation_Click* pour l'événement *Click* sur le lien *LinkButtonEnregistrerSimulation*

La gestion des événements *Click* sur les six liens du menu sera répartie de la façon suivante :
- la page [Formulaire.aspx] gèrera les liens [LinkButtonFaireSimulation, LinkButtonEnregistrerSimulation, LinkButtonEffacerSimulation, LinkButtonVoirSimulations]
- la page [Simulations.aspx] gèrera le lien [LinkButtonFormulaireSimulation]
- la page maître [MasterPage.master] gèrera le lien [LinkButtonTerminerSession]. Pour cet événement, elle n'a en effet pas besoin de connaître la page qu'elle encapsule.

11.5.2.5 L'événement Init de la page maître

Les trois pages [Formulaire.aspx, Simulations.aspx, Erreurs.aspx] de l'application ont [MasterPage.master] pour page maître. Appelons M la page maître, E la page encapsulée. Lorsque la page E est demandée par le client, les événements suivants se produisent dans l'ordre :
- E.Init
- M.Init
- E.Load
- M.Load
- ...

Nous allons utiliser l'événement *Init* de la page M pour exécuter du code qu'il serait intéressant d'exécuter le plus tôt possible, ceci quelque soit la page cible E. Pour découvrir ce code, revoyons l'image d'ensemble de l'application :



Ci-dessus, [Global] est l'objet de type [HttpApplication] qui initialise l'application. Cette classe est la même que dans la version [pam-v4-3tier-nhibernate-multivues-monopage] :

```
1.  using System;
2.  ...
3.
4.  namespace pam_v7
5.  {
6.    public class Global : System.Web.HttpApplication
7.    {
8.      // --- données statiques de l'application ---
9.      public static Employe[] Employes;
10.     public static IPamMetier PamMetier = null;
11.     public static string Msg;
12.     public static bool Erreur = false;
13.
```

```
14.     // démarrage de l'application
15.     public void Application_Start(object sender, EventArgs e)
16.     {
17. ...
18.     }
19.
20.     public void Session_Start(object sender, EventArgs e)
21.     {
22. ...
23.     }
24.   }
25. }
```

Si la classe [Global] ne réussit pas à initialiser correctement l'application, elle positionne deux variables publiques statiques :

- le booléen **Erreur** de la ligne 12 est mis à *vrai*
- la variable **Msg** de la ligne 11 contient un message donnant des détails sur l'erreur rencontrée

Lorsque l'utilisateur demande l'une des pages [Formulaire.aspx, Simulations.aspx] alors que l'application ne s'est pas initialisée correctement, cette demande doit être transférée ou redirigée vers la page [Erreurs.aspx], qui affichera le message d'erreur de la classe [Global]. On peut gérer ce cas de diverses façons :

- faire le test d'erreur d'initialisation dans le gestionnaire des événements *Init* ou *Load* de chacune des pages [Formulaire.aspx, Simulations.aspx]
- faire le test d'erreur d'initialisation dans le gestionnaire des événements *Init* ou *Load* de la page maître de ces deux pages. Cette méthode a l'avantage de placer le test d'erreur d'initialisation à un unique endroit.

Nous choisissons de faire le test d'erreur d'initialisation dans le gestionnaire de l'événement *Init* de la **page maître** :

```
1.          protected void Page_Init(object sender, System.EventArgs e)
2.          {
3.              // gestionnaire d'évts
4.              LinkButtonTerminerSession.Click += LinkButtonTerminerSession_Click;
5.              // des erreurs d'initialisation ?
6.              if (Global.Erreur)
7.              {
8.                  // la page encapsulée est-elle la page d'erreurs ?
9.                  bool isPageErreurs =...;
10.                 // si c'est la page d'erreurs qui s'affiche, on laisse faire sinon on redirige le
    client vers la page d'erreurs
11.                 if (!isPageErreurs)
12.                     Response.Redirect("Erreurs.aspx");
13.                 return;
14.             }
15. }
```

Le code ci-dessus va s'exécuter dès que l'une des pages [Formulaire.aspx, Simulations.aspx, Erreurs.aspx] va être demandée. Dans le cas où la page demandée est [Formulaire.aspx, Simulations.aspx], on se contente (ligne 12) de rediriger le client vers la page [Erreurs.aspx], celle-ci se chargeant d'afficher le message d'erreur de la classe [Global]. Dans le cas où la page demandée est [Erreurs.aspx], cette redirection ne doit pas avoir lieu : il faut laisser la page [Erreurs.aspx] s'afficher. Il nous faut donc savoir dans la méthode [Page_Init] de la page maître, quelle est la page que celle-ci encapsule.

Revenons sur l'arbre des composants de la page maître :

```
1.  ...
2.  <body background="ressources/standard.jpg">
3.          <form id="form1" runat="server">
4.                  <asp:Panel ID="entete" runat="server" BackColor="#FFE0C0" Width="1239px" >
5.  ...
6.                  </asp:Panel>
7.                  <div>
8.                          <asp:Panel ID="contenu" runat="server" BackColor="#FFFFC0">
9.                                  <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
10.                                 </asp:ContentPlaceHolder>
11.                         </asp:Panel>
12.                 </div>
13.         </form>
14. </body>
15. </html>
```

- lignes 1-13 : le conteneur d'id "form1"

- lignes 4-6 : le conteneur d'id "entete", inclus dans le conteneur d'id "form1"
- lignes 8-11 : le conteneur d'id "contenu", inclus dans le conteneur d'id "form1"
- lignes 9-10 : le conteneur d'id "ContentPlaceHolder1", inclus dans le conteneur d'id "contenu"

Une page E encapsulée dans la page maître M, l'est dans le conteneur d'id "ContentPlaceHolder1". Pour référencer un composant d'id C de cette page E, on écrira :

```
this.FindControl("form1").FindControl("contenu").FindControl("ContentPlaceHolder1").FindControl("C");
```

L'arbre des composants de la page [Erreurs.aspx] est lui, le suivant :

```
1.  <%@ Page Language="C#" MasterPageFile="~/MasterPage.master" AutoEventWireup="true"
2.    CodeBehind="Erreurs.aspx.cs" Inherits="pam_v7.PageErreurs" Title="Pam : erreurs" %>
3.
4.  <%@ MasterType VirtualPath="~/MasterPage.master" %>
5.  <asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
6.      <h3>Les erreurs suivantes se sont produites au démarrage de l'application</h3>
7.      <ul>
8.         <asp:Repeater id="rptErreurs" runat="server">
9.            <ItemTemplate>
10.              <li>
11.                 <%# Container.DataItem %>
12.              </li>
13.           </ItemTemplate>
14.        </asp:Repeater>
15.     </ul>
16. </asp:Content>
```

Lorsque la page [Erreurs.aspx] est fusionnée avec la page maître M, le contenu de la balise <*asp:Content*> ci-dessus (lignes 5-16), est intégré dans la balise <*asp:ContentPlaceHolder*> d'id "ContentPlaceholder1" de la page M, l'arbre des composants de celle-ci devenant alors :

```
1.  ...
2.  <body background="ressources/standard.jpg">
3.          <form id="form1" runat="server">
4.                  <asp:panel ID="entete" runat="server" BackColor="#FFE0C0" Width="1239px" >
5.  ...
6.                  </asp:Panel>
7.                  <div>
8.                          <asp:Panel ID="contenu" runat="server" BackColor="#FFFFC0">
9.                                  <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
10.                 <h3>Les erreurs suivantes se sont produites au démarrage de l'application</h3>
11.                 <ul>
12.                         <asp:Repeater id="rptErreurs" runat="server">
13.                                 <ItemTemplate>
14.                                         <li>
15.                                                 <%# Container.DataItem %>
16.                                         </li>
17.                                 </ItemTemplate>
18.                         </asp:Repeater>
19.                 </ul>
20.                                 </asp:ContentPlaceHolder>
21.                         </asp:Panel>
22.                 </div>
23.         </form>
24. </body>
25. </html>
```

- ligne 12 : le composant [rptErreurs] peut être utilisé pour savoir si la page maître M contient ou non la page [Erreurs.aspx]. En effet, ce composant n'existe que dans cette page.

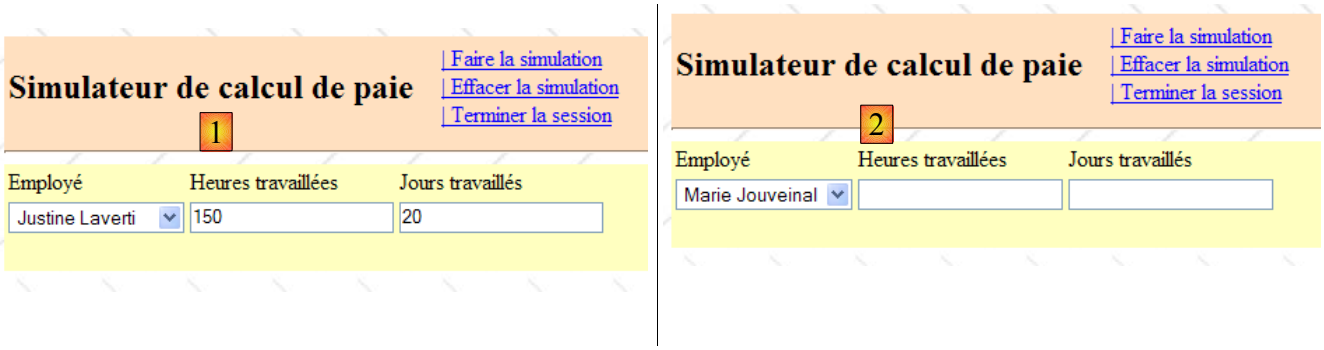
Ces explications suffisent pour comprendre le code de la procédure [Page_Init] de la page maître :

```
1.  protected void Page_Init(object sender, System.EventArgs e)
2.          {
3.              // gestionnaire d'évts
4.              LinkButtonTerminerSession.Click += LinkButtonTerminerSession_Click;
5.              // des erreurs d'initialisation ?
6.              if (Global.Erreur)
7.              {
8.                  // la page encapsulée est-elle la page d'erreurs ?
```

```
9.                  bool isPageErreurs =
    this.FindControl("form1").FindControl("contenu").FindControl("ContentPlaceHolder1").FindControl("
    rptErreurs") != null;
10.                 // si c'est la page d'erreurs qui s'affiche, on laisse faire sinon on redirige le
    client vers la page d'erreurs
11.                 if (!isPageErreurs)
12.                     Response.Redirect("Erreurs.aspx");
13.                 return;
14.             }
15.         }
```

- ligne 4 : on associe un gestionnaire d'événement à l'événement *Click* sur le lien *LinkButtonTerminerSession*. Ce gestionnaire est dans la classe *MasterPage*.
- ligne 6 : on vérifie si la classe [Global] a positionné son booléen *Erreur*
- ligne 9 : si oui, le booléen *IsPageErreurs* indique si la page encapsulée dans la page maître est la page [Erreurs.aspx]
- ligne 12 : si la page encapsulée dans la page maître n'est pas la page [Erreurs.aspx], alors on redirige le client vers cette page, sinon on ne fait rien.

11.5.2.6 L'événement Click sur le lien [LinkButtonTerminerSession]



Lorsque l'utilisateur clique sur le lien [Terminer la session] dans la vue (1) ci-dessus, il faut vider la session de son contenu et présenter un formulaire vide (2).

Le code du gestionnaire de cet événement pourrait être le suivant :

```
1.          protected void LinkButtonTerminerSession_Click(object sender, System.EventArgs e)
2.          {
3.              // on abandonne la session
4.              Session.Abandon();
5.              // on affiche la vue [formulaire]
6.              Response.Redirect("Formulaire.aspx");
7.  }
```

- ligne 4 : la session courante est abandonnée
- ligne 6 : le client est redirigé vers la page [Formulaire.aspx]

On voit que ce code ne fait intervenir aucun des composants des pages [Formulaire.aspx, Simulations.aspx, Erreurs.aspx]. L'événement peut donc être géré par la page maître elle-même.

11.5.3 Code de contrôle de la page [Erreurs.aspx]

Le code de contrôle de la page [Erreurs.aspx] pourrait être le suivant :

```
1.  using System.Collections.Generic;
2.
3.  namespace pam_v7
4.  {
5.    public partial class Erreurs : System.Web.UI.Page
6.    {
7.      protected void Page_Load(object sender, System.EventArgs e)
8.      {
9.        // des erreurs d'initialisation ?
10.       if (Global.Erreur)
```

```
11.       {
12.         // on prépare le modèle de la page [erreurs]
13.         List<string> erreursInitialisation = new List<string>();
14.         erreursInitialisation.Add(Global.Msg);
15.         // on associe la liste d'erreurs à son composant
16.         rptErreurs.DataSource = erreursInitialisation;
17.         rptErreurs.DataBind();
18.       }
19.       // on fixe le menu
20.       Master.SetMenu(false, false, false, false, false, false);
21.     }
22.   }
23. }
```

Rappelons que la page [Erreurs.aspx] a pour unique rôle d'afficher une erreur d'initialisation de l'application lorsque celle-ci se produit :

- ligne 10 : on teste si l'initialisation s'est terminée par une erreur
- lignes 13-14 : si oui, le message d'erreur (Global.Msg) est placé dans une liste [ErreursInitialisation]
- lignes 16-17 : on demande au composant [rptErreurs] d'afficher cette liste
- ligne 20 : dans tous les cas (erreur ou pas), les options du menu de la page maître ne sont pas affichées, de sorte que l'utilisateur ne peut débuter aucune nouvelle action à partir de cette page.

Que se passe-t-il si l'utilisateur demande directement la page [Erreurs.aspx] (ce qu'il n'est pas supposé faire dans une utilisation normale de l'application) ? En suivant le code de [MasterPage.master.cs] et de [Erreurs.aspx.cs], on s'apercevra que :

- s'il y a eu erreur d'initialisation, celle-ci est affichée
- s'il n'y a pas eu erreur d'initialisation, l'utilisateur reçoit une page ne contenant que l'entête de [MasterPage.master] avec aucune option de menu affichée.

11.5.4 Code de contrôle de la page [Formulaire.aspx]

11.5.4.1 Squelette de la classe

Le squelette du code de contrôle de la page [Formulaire.aspx] pourrait être le suivant :

```
1.  using Pam.Metier.Entites;
2.  ...
3.
4.  partial class PageFormulaire : System.Web.UI.Page
5.  {
6.
7.      // chargement de la page
8.      protected void Page_Load(object sender, System.EventArgs e)
9.      {
10.         // gestionnaire d'évts
11.         Master.OptionFaireSimulation.Click += OptFaireSimulation_Click;
12.         Master.OptionEffacerSimulation.Click += OptEffacerSimulation_Click;
13.         Master.OptionVoirSimulations.Click += OptVoirSimulations_Click;
14.         Master.OptionEnregistrerSimulation.Click += OptEnregistrerSimulation_Click;
15. ....
16.     }
17.
18.     // calcul de la paie
19.     private void OptFaireSimulation_Click(object sender, System.EventArgs e)
20.     {
21. ....
22.     }
23.
24.     // effacer la simulation
25.     private void OptEffacerSimulation_Click(object sender, System.EventArgs e)
26.     {
27. ...
28.     }
29.
30.     protected void OptVoirSimulations_Click(object sender, System.EventArgs e)
31.     {
32. ....
33.     }
34.
35.     protected void OptEnregistrerSimulation_Click(object sender, System.EventArgs e)
36.     {
37. ...
```

```
38.     }
39.
40. }
```

Le code de contrôle de la page [Formulaire.aspx] gère cinq événements :

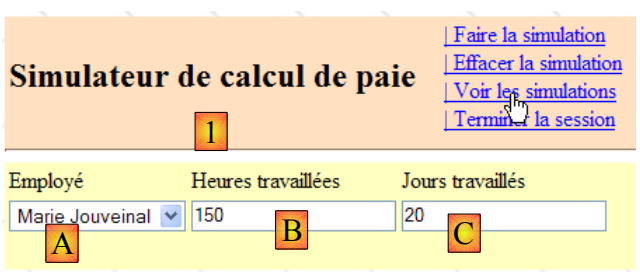
1. l'événement *Load* de la page
2. l'événement *Click* sur le lien [LinkButtonFaireSimulation] de la page maître
3. l'événement *Click* sur le lien [LinkButtonEffacerSimulation] de la page maître
4. l'événement *Click* sur le lien [LinkButtonEnregistrerSimulation] de la page maître
5. l'événement *Click* sur le lien [LinkButtonVoirSimulations] de la page maître

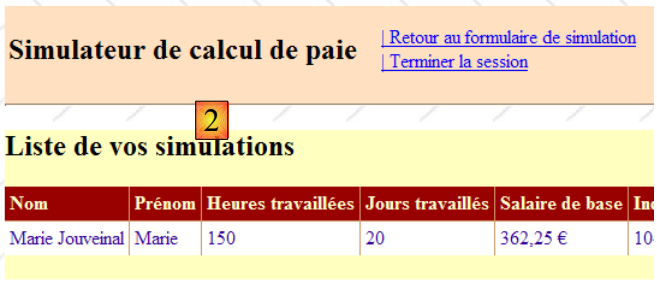
11.5.4.2 Evénement Load de la page

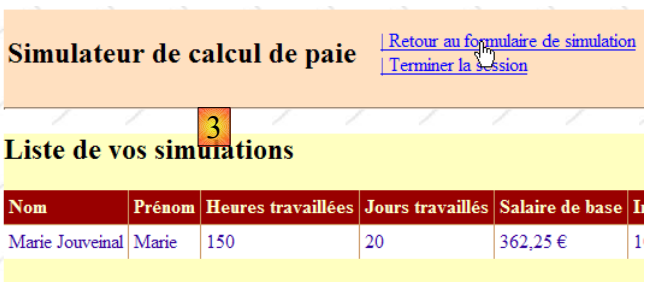
Le squelette du gestionnaire de l'événement *Load* de la page pourrait être le suivant :

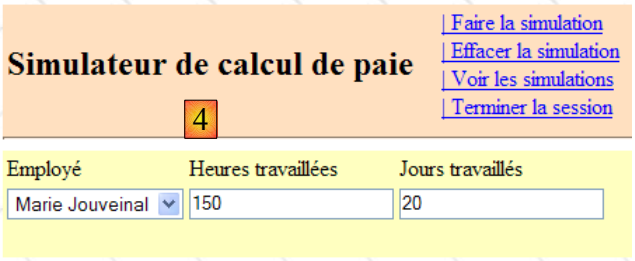
```
1.      protected void Page_Load(object sender, System.EventArgs e)
2.      {
3.          // gestionnaire d'évts
4.          Master.OptionFaireSimulation.Click += OptFaireSimulation_Click;
5.          Master.OptionEffacerSimulation.Click += OptEffacerSimulation_Click;
6.          Master.OptionVoirSimulations.Click += OptVoirSimulations_Click;
7.          Master.OptionEnregistrerSimulation.Click += OptEnregistrerSimulation_Click;
8.          // affichage vue [saisies]
9.          ...
10.         // positionnement menu page maître
11.         ...
12.         // traitement requête GET
13.         if (!IsPostBack)
14.         {
15.             // chargement des noms des employés dans le combo
16. ...
17.             // init vue [saisies] avec saisies mémorisées dans la session si elles existent
18. ....
19.         }
20. }
```

Un exemple pour éclaircir le commentaire de la ligne 17 pourrait être celui-ci :









- en [1], on demande à voir la liste des simulations. Des saisies ont été faites en [A, B, C].
- en [2], on voit la liste
- en [3], on demande à retourner au formulaire
- en [4], on retrouve le formulaire tel qu'on l'a laissé. Comme il y a eu deux requêtes, (1,2) et (3,4), cela signifie que :
  - lors du passage de [1] à [2], les saisies de [1] ont été mémorisées
  - lors du passage de [3] à [4], elles ont été restituées. C'est la procédure [Page_Load] de [Formulaire.aspx] qui opére cette restitution.

Question : compléter la procédure Page_Load en vous aidant des commentaires et du code de la version [pam-v4-3tier-nhibernate-multivues-monopage]

11.5.4.3 Gestion des événements Click sur les liens du menu

Le squelette des gestionnaires des événements *Click* sur les liens de la page maître est le suivant :

```
1.  // calcul de la paie
2.      private void OptFaireSimulation_Click(object sender, System.EventArgs e)
3.      {
4.          // effet Ajax
5.          Thread.Sleep(3000);
6.          // page valide ?
7.          Page.Validate();
8.          if (!Page.IsValid)
9.          {
10.             // affichage vue [saisie]
11. ...
12.         }
13.         // la page est valide - on récupère les saisies
14. ...
15.         // on calcule le salaire de l'employé
16.         FeuilleSalaire feuillesalaire;
17.         try
18.         {
19.             feuillesalaire = ...;
20.         }
21.         catch (PamException ex)
22.         {
23.             // on a rencontré un problème
24. ...
25.             return;
26.         }
27.         // on met le résultat dans la session
28.         Session["simulation"] = ...;
29.         // on met les saisies dans la session
30. ...
31.         // affichage
32. ...
33.         // affichage vues
34. ...
35.         // affichage menu MasterPage
36. ...
37.     }
38.
39.     // effacer la simulation
40.     private void OptEffacerSimulation_Click(object sender, System.EventArgs e)
41.     {
42.         // affichage panel [saisie]
43. ...
44.         // sélection 1er employé
45. ...
46.     }
47.
48.     protected void OptVoirSimulations_Click(object sender, System.EventArgs e)
49.     {
50.         // on met les saisies dans la session
51. ...
52.         // on affiche la vue [simulations]
53.         Response.Redirect("simulations.aspx");
54.     }
55.
56.     protected void OptEnregistrerSimulation_Click(object sender, System.EventArgs e)
57.     {
58.         // on enregistre la simulation courante dans la session de l'utilisateur
```

```
59. ...
60.         // on affiche la vue [simulations]
61.         Response.Redirect("simulations.aspx");
62. }
```

Question : compléter le code des procédures ci-dessus en vous aidant des commentaires et du code de la version [pam-v4-3tier-nhibernate-multivues-monopage]

11.5.5 Code de contrôle de la page [Simulations.aspx]

Le squelette du code de contrôle de la page [Simulations.aspx] pourrait être le suivant :

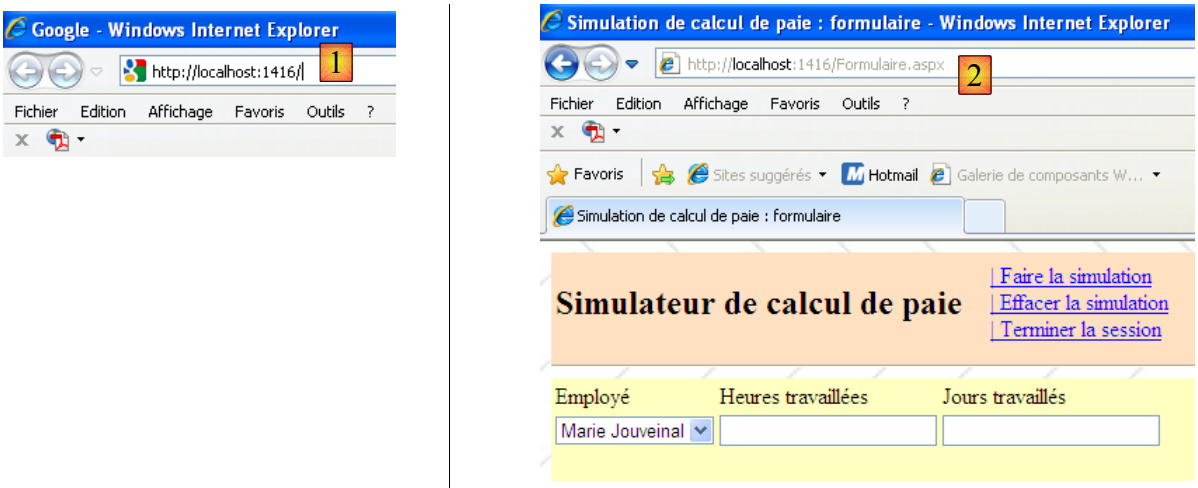
```
1.  using System.Collections.Generic;
2.  using Pam.Web;
3.  using System.Web.UI.WebControls;
4.
5.  partial class PageSimulations : System.Web.UI.Page
6.  {
7.
8.      // les simulations
9.      private List<Simulation> simulations;
10.
11.     // chargement de la page
12.     protected void Page_Load(object sender, System.EventArgs e)
13.     {
14.         // gestionnaire d'évts
15.         Master.OptionFormulaireSimulation.Click += OptFormulaireSimulation_Click;
16.         GridViewSimulations.RowDeleting += GridViewSimulations_RowDeleting;
17.         // on récupère les simulations dans la session
18.         simulations = ...;
19.         // y-a-t-il des simulations ?
20.         if (simulations.Count != 0)
21.         {
22.             // première vue visible
23.             ...
24.             // on remplit le gridview
25.             ...
26.         }
27.         else
28.         {
29.             // seconde vue
30.             ...
31.         }
32.         // on fixe le menu
33.         ...
34.     }
35.
36.     protected void GridViewSimulations_RowDeleting(object sender,
    System.Web.UI.WebControls.GridViewDeleteEventArgs e)
37.     {
38.         // on récupère les simulations dans la session
39.         List<Simulation> simulations = ...;
40.         // on supprime la simulation désignée (e.RowIndex représente le n° de la ligne supprimée
    dans le gridview)
41.         ..
42.         // reste-t-il des simulations ?
43.         if (simulations.Count != 0)
44.         {
45.             // on remplit le gridview
46.             ...
47.         }
48.         else
49.         {
50.             // vue [SimulationsVides]
51.             ...
52.         }
53.     }
54.
55.     protected void OptFormulaireSimulation_Click(object sender, System.EventArgs e)
56.     {
57.         // on affiche la vue [formulaire]
58.         Response.Redirect("formulaire.aspx");
59.     }
60.
```

```
61. }
```

Question : compléter le code des procédures ci-dessus en vous aidant des commentaires et du code de la version [pam-v4-3tier-nhibernate-multivues-monopage]

11.5.6 Code de contrôle de la page [Default.aspx]

On peut prévoir une page [Default.aspx] dans l'application, afin de permettre à l'utilisateur de demander l'url de l'application sans préciser de page, comme ci-dessous :



La demande [1] a reçu en réponse la page [Formulaire.aspx] (2). On sait que la demande (1) est traitée par défaut par la page [Default.aspx] de l'application. Pour obtenir (2), il suffit que [Default.aspx] redirige le client vers la page [Formulaire.aspx]. Cela peut être obtenu avec le code suivant :

```
1.  partial class _Default : System.Web.UI.Page
2.  {
3.
4.      protected void Page_Init(object sender, System.EventArgs e)
5.      {
6.          // on redirige vers le formulaire de saisie
7.          Response.Redirect("Formulaire.aspx");
8.      }
9.  }
```

La page de présentation [Default.aspx] ne contient elle que la directive qui la relie à [Default.aspx.cs] :

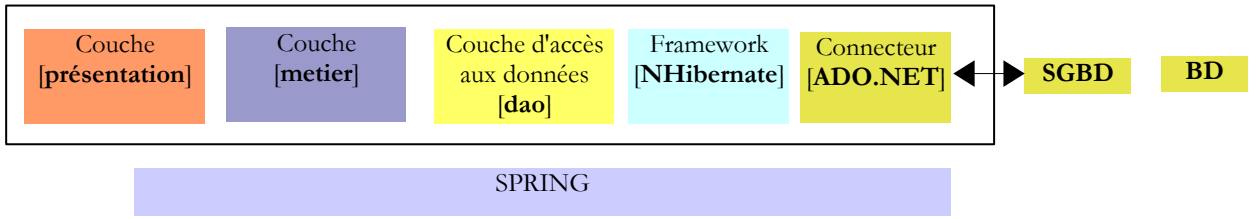
```
<%@ Page Language="C#" AutoEventWireup="true"
  CodeBehind="Default.aspx.cs" Inherits="pam_v7._Default" Title="Untitled Page" %>
```

12 L'application [SimuPaie] – version 8 – client ASP.NET d'un service web

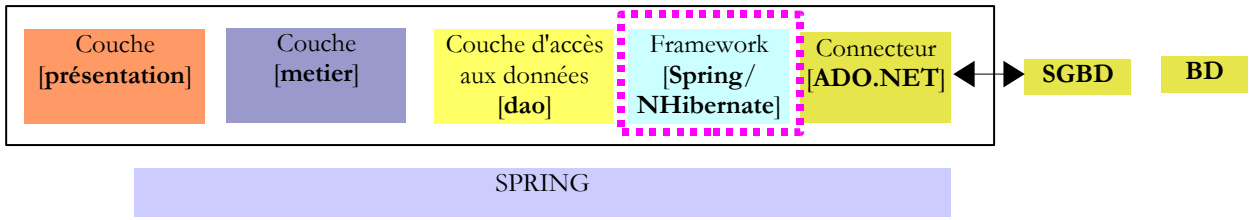
Question : en suivant ce qui a été fait pour la version 6, transformez la version 7 [pam-v7-3tier-nhibernate-multivues-multipages] en version 8 [pam-v8-multivues-multipages-client-webservice] où l'application web est cliente du service web distant.

13 L'application [SimuPaie] – version 9 – intégration Spring / NHibernate

Nous nous proposons ici de reprendre l'application ASP.NET à trois couches de la version 7 [pam-v7-3tier-nhibernate-multivues-multipages]. L'architecture en couches de l'application était la suivante :



Ci-dessus, la couche [dao] avait été implémentée avec le framework NHibernate. Le framework Spring n'avait été utilisé que pour l'intégration des couches entre-elles. Le framework Spring offre des classes utilitaires pour travailler avec le framework Nhibernate. L'utilisation de ces classes rend le code de la couche [dao] plus simple à écrire. L'architecture précédente évolue comme suit :

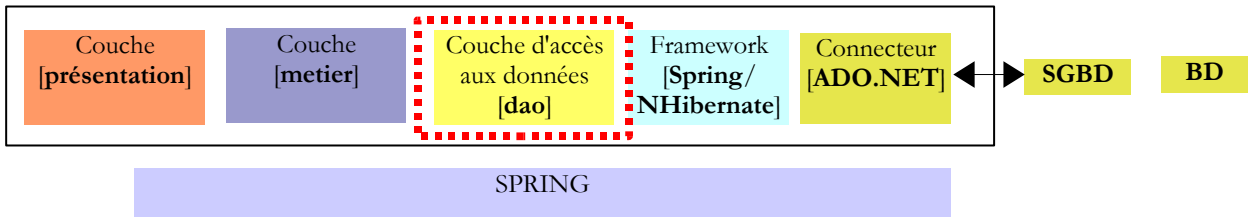


A cause de la structure en couches utilisée, l'utilisation de l'intégration **Spring / NHibernate** entraîne la modification de la seule couche [dao]. Les couches [presentation] (web / ASP.NET) et [metier] n'auront pas à être modifiées. C'est là le principal avantage des architectures en couches intégrées par Spring.

Dans la suite, nous allons construire la couche [dao] avec [Spring / NHibernate] en commentant le code d'une solution fonctionnelle. Nous ne chercherons pas à donner toutes les possibilités de configuration ou d'utilisation du framework [Spring / Nhibernate]. Le lecteur pourra adapter la solution proposée à ses propres problèmes en s'aidant de la documentation de Spring.NET [http://www.springframework.net/documentation.html] (juin 2010).

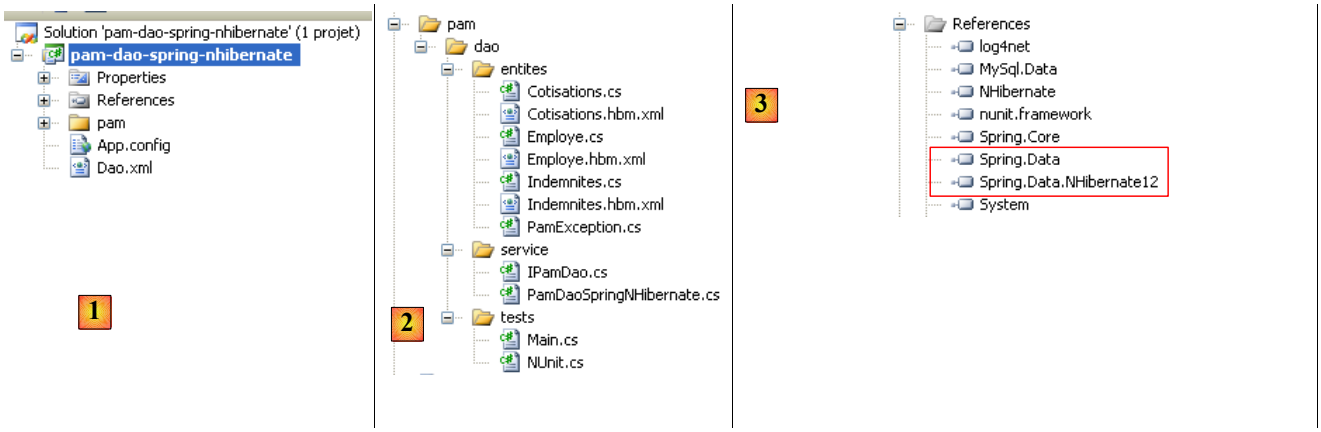
La démarche suivie pour construire les couches [dao] et [metier] est celle de la version 3 décrite au paragraphe 7, page 88. Celle suivie pour la couche [présentation] est celle de la version 7 décrite au paragraphe 11, page 161.

13.1 La couche [dao] d'accès aux données

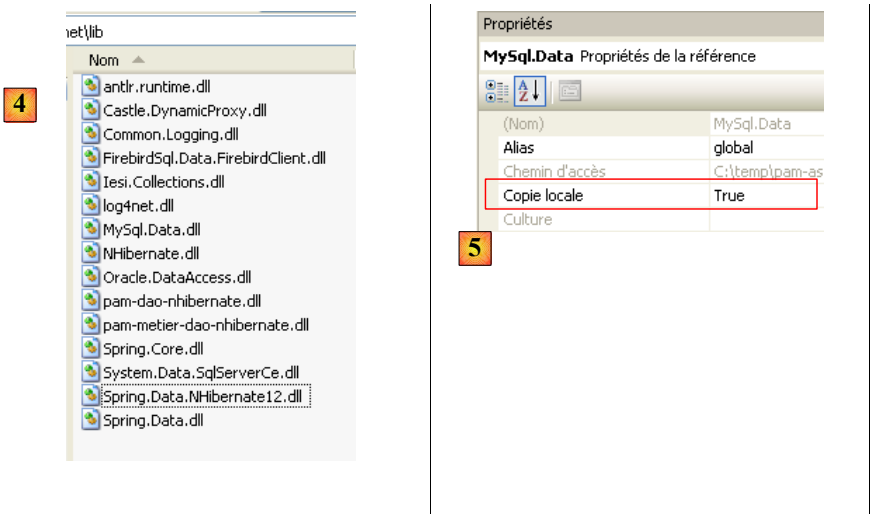


13.1.1 Le projet Visual Studio C# de la couche [dao]

Le projet Visual Studio de la couche [dao] est le suivant :



- en [1], le projet dans sa globalité
  - le dossier [pam] contient les classes du projet ainsi que la configuration des entités NHibernate
  - les fichiers [App.config] et [Dao.xml] configurent le framework Spring / NHibernate. Nous aurons à décrire le contenu de ces deux fichiers.
- en [2], les différentes classes du projet
  - dans le dossier [entites] nous retrouvons les entités NHibernate étudiées dans le projet [pam-dao-nhibernate]
  - dans le dossier [service], nous trouvons l'interface [IPamDao] et son implémentation avec le framework Spring / NHibernate [PamDaoSpringNHibernate]. Nous aurons à écrire cette nouvelle implémentation de l'interface [IPamDao]
  - le dossier [tests] contient les mêmes tests que le projet [pam-dao-nhibernate]. Ils testent la même interface [IPamdao].
- en [3], les références du projet. L'intégration Spring / NHibernate nécessite deux nouvelles Dll [Spring.Data] et [Spring.Data.NHibernate12]. Ces Dll sont disponibles dans le framework Spring.Net. Elles ont été ajoutées au dossier [lib] des Dll [4] :



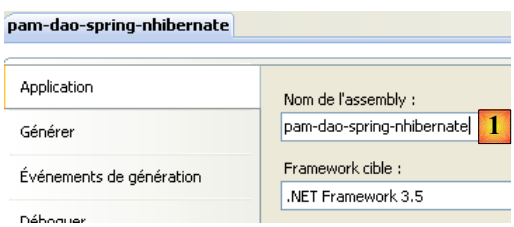
Dans les références [3] du projet, on trouve les DLL suivantes :

- *NHibernate* : pour l'ORM NHibernate
- *MySql.Data* : le pilote ADO.NET du SGBD MySQL
- *Spring.Core* : pour le framework Spring qui assure l'intégration des couches
- *log4net* : une bibliothèque de logs
- *nunit.framework* : une bibliothèque de tests unitaires
- *Spring.Data* et *Spring.Data.NHibernate12* : assurent le support Spring / NHibernate.

Ces références ont été prises dans le dosssier [lib] [4]. On prendra soin que toutes ces références aient leur propriété "Copie locale" à "True" [5] :

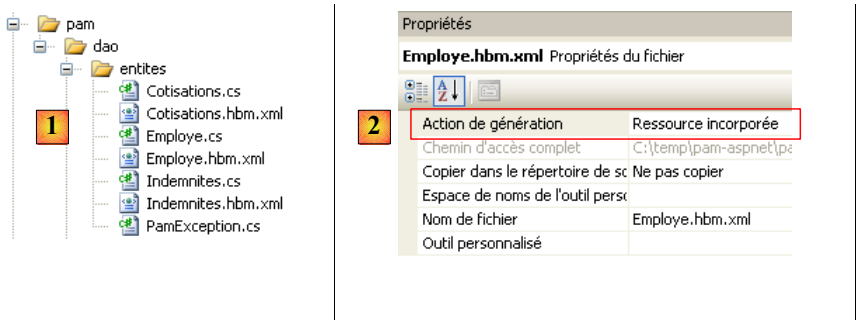
13.1.2 La configuration du projet C#

Le projet est configuré de la façon suivante :



- en [1], le nom de l'assembly du projet est [pam-dao-spring-nhibernate]. Ce nom intervient dans divers fichiers de configuration du projet.

13.1.3 Les entités de la couche [dao]



Les entités (objets) nécessaires à la couche [dao] ont été rassemblées dans le dossier [entites] [1] du projet. Ces entités sont celles du projet [pam-dao-nhibernate] à une différence près dans les fichiers de configuration NHibernate. Prenons par exemple, le fichier [Employe.hbm.xml] :

- en [2], le fichier est configuré pour être incorporé dans l'assembly du projet

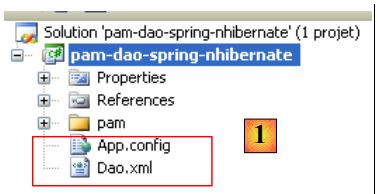
Son contenu est le suivant :

```
1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
3.  namespace="Pam.Dao.Entites" assembly="pam-dao-spring-nhibernate">
4.     <class name="Employe" table="EMPLOYES">
5.        <id name="Id" column="ID">
6.           <generator class="native" />
7.        </id>
8.        <version name="Version" column="VERSION"/>
9.        <property name="SS" column="SS" length="15" not-null="true" unique="true"/>
10.       <property name="Nom" column="NOM" length="30" not-null="true"/>
11.       <property name="Prenom" column="PRENOM" length="20" not-null="true"/>
12.       <property name="Adresse" column="ADRESSE" length="50" not-null="true" />
13.       <property name="Ville" column="VILLE" length="30" not-null="true"/>
14.       <property name="CodePostal" column="CP" length="5" not-null="true"/>
15.       <many-to-one name="Indemnites" column="INDEMNITE_ID" cascade="all" lazy="false"/>
16.    </class>
17. </hibernate-mapping>
```

- ligne 2 : l'attribut assembly indique que le fichier [Employe.hbm.xml] sera trouvé dans l'assembly [pam-dao-spring-nhibernate]

13.1.4 Configuration Spring / NHibernate

Revenons au projet Visual C# :



- en [1], les fichiers [App.config] et [Dao.xml] configurent l'intégration Spring / NHibernate

13.1.4.1 Le fichier [App.config]

Le fichier [App.config] est le suivant :

```
1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <configuration>
3.     <!-- sections de configuration -->
4.     <configSections>
5.        <sectionGroup name="spring">
6.           <section name="parsers" type="Spring.Context.Support.NamespaceParsersSectionHandler, Spring.Core" />
7.           <section name="objects" type="Spring.Context.Support.DefaultSectionHandler, Spring.Core" />
8.           <section name="context" type="Spring.Context.Support.ContextHandler, Spring.Core" />
9.        </sectionGroup>
10.       <section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler,log4net" />
11.    </configSections>
12.
13.
14.    <!-- configuration Spring -->
15.    <spring>
16.       <parsers>
17.          <parser type="Spring.Data.Config.DatabaseNamespaceParser, Spring.Data" />
18.       </parsers>
19.       <context>
20.          <resource uri="Dao.xml" />
21.       </context>
22.    </spring>
23.
24.    <!-- This section contains the log4net configuration settings -->
25.    <!-- NOTE IMPORTANTE : les logs ne sont pas actifs par défaut. Il faut les activer par programme
26.    avec l'instruction log4net.Config.XmlConfigurator.Configure();
27.    ! -->
28.    <log4net>
29.       <appender name="ConsoleAppender" type="log4net.Appender.ConsoleAppender">
30.          <layout type="log4net.Layout.PatternLayout">
31.             <conversionPattern value="%-5level %logger - %message%newline" />
32.          </layout>
33.       </appender>
34.
35.       <!-- Set default logging level to DEBUG -->
36.       <root>
37.          <level value="DEBUG" />
38.          <appender-ref ref="ConsoleAppender" />
39.       </root>
40.
41.       <!-- Set logging for Spring.  Logger names in Spring correspond to the namespace -->
42.       <logger name="Spring">
43.          <level value="INFO" />
44.       </logger>
45.
46.       <logger name="Spring.Data">
47.          <level value="DEBUG" />
48.       </logger>
49.
50.       <logger name="NHibernate">
51.          <level value="DEBUG" />
52.       </logger>
53.    </log4net>
54.
55. </configuration>
```

Le fichier [App.config] ci-dessus configure Spring (lignes 5-9, 15-22), log4net (ligne 10, lignes 28-53) mais pas NHibernate. Les objets de Spring ne sont pas configurés dans [App.config] mais dans le fichier [Dao.xml] (ligne 20). La configuration de Spring / NHibernate qui consiste à déclarer des objets Spring particuliers sera donc trouvée dans ce fichier.

13.1.4.2 Le fichier [Dao.xml]

Le fichier [Dao.xml] qui rassemble les objets gérés par Spring est le suivant :

```
1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <objects xmlns="http://www.springframework.net"
3.           xmlns:db="http://www.springframework.net/database">
4.
5.     <!-- Referenced by main application context configuration file -->
6.     <description>
7.        Application Spring / NHibernate
8.     </description>
9.
10.    <!-- Database and NHibernate Configuration -->
11.    <db:provider id="DbProvider"
12.                 provider="MySql.Data.MySqlClient"
13.                 connectionString="Server=localhost;Database=dbpam_nhibernate;Uid=root;Pwd=;"/>
14.
15.    <object id="NHibernateSessionFactory" type="Spring.Data.NHibernate.LocalSessionFactoryObject,
    Spring.Data.NHibernate12">
16.       <property name="DbProvider" ref="DbProvider"/>
17.       <property name="MappingAssemblies">
18.          <list>
19.             <value>pam-dao-spring-nhibernate</value>
20.          </list>
21.       </property>
22.       <property name="HibernateProperties">
23.          <dictionary>
24.             <entry key="hibernate.dialect" value="NHibernate.Dialect.MySQLDialect"/>
25.             <entry key="hibernate.show_sql" value="false"/>
26.          </dictionary>
27.       </property>
28.       <property name="ExposeTransactionAwareSessionFactory" value="true" />
29.    </object>
30.
31.    <!-- gestionnaire de transactions -->
32.    <object id="transactionManager"
33.         type="Spring.Data.NHibernate.HibernateTransactionManager, Spring.Data.NHibernate12">
34.       <property name="DbProvider" ref="DbProvider"/>
35.       <property name="SessionFactory" ref="NHibernateSessionFactory"/>
36.    </object>
37.
38.    <!-- Hibernate Template -->
39.    <object id="HibernateTemplate" type="Spring.Data.NHibernate.Generic.HibernateTemplate">
40.       <property name="SessionFactory" ref="NHibernateSessionFactory" />
41.       <property name="TemplateFlushMode" value="Auto" />
42.       <property name="CacheQueries" value="true" />
43.    </object>
44.
45.    <!-- Data Access Objects -->
46.    <object id="pamdao" type="Pam.Dao.Service.PamDaoSpringNHibernate, pam-dao-spring-nhibernate" init-
    method="init" destroy-method="destroy">
47.       <property name="HibernateTemplate" ref="HibernateTemplate"/>
48.    </object>
49. </objects>
```

- les lignes 11-13 configurent la connexion à la base de données [dbpam_nhibernate]. On y trouve :
  - le provider ADO.NET nécessaire à la connexion, ici le provider du SGBD MySQL. Cela implique d'avoir la Dll [Mysql.Data] dans les références du projet.
  - la chaîne de connexion à la base de données (serveur, nom de la base, propriétaire de la connexion, son mot de passe)
- les lignes 15-29 configurent la *SessionFactory* de NHibernate, l'objet qui sert à obtenir des sessions *NHibernate*. On rappelle que toute opération sur la base de données se fait à l'intérieur d'une session NHibernate. Ligne 15, on peut voir que la *SessionFactory* est implémentée par la classe Spring *Spring.Data.NHibernate.LocalSessionFactoryObject* trouvée dans la Dll *Spring.Data.NHibernate12*.
- ligne 16 : la propriété *DbProvider* fixe les paramètres de la connexion à la base de données (provider ADO.NET et chaîne de connexion). Ici cette propriété référence l'objet *DbProvider* défini précédemment aux lignes 11-13.
- lignes 17-20 : fixent la liste des assembly contenant des fichiers [*.hbm.xml] configurant des entités gérées par NHibernate. La ligne 19 indique que ces fichiers seront trouvés dans l'assembly du projet. Nous rappelons que ce nom est trouvé dans les propriétés du projet C#. Nous rappelons également que tous les fichiers [*.hbm.xml] ont été configurés pour être incorporés dans l'assembly du projet.
- lignes 22-27 : propriétés spécifiques de NHibernate.
  - ligne 24 : le dialecte SQL utilisé sera celui de MySQL
  - ligne 25 : le SQL émis par NHibernate n'apparaîtra pas dans les logs de la console. Mettre cette propriété à *true* permet de connaître les ordres SQL émis par NHibernate. Cela peut aider à comprendre par exemple pourquoi une application est lente lors de l'accès à la base.

- ligne 28 : la propriété *ExposeTransactionAwareSessionFactory* à *true* va faire que Spring va gérer les annotations de gestion des transactions qui seront trouvées dans le code C#. Nous y reviendrons lorsque nous écrirons la classe implémentant la couche [dao].
- les lignes 32-36 définissent le gestionnaire de transactions. Là encore ce gestionnaire est une classe Spring de la Dll *Spring.Data.NHibernate12*. Ce gestionnaire a besoin de connaître les paramètres de connexion à la base de données (ligne 34) ainsi que la *SessionFactory* de NHibernate (ligne 35).
- les lignes 39-43 définissent les propriétés de la classe *HibernateTemplate*, là encore une classe de Spring. Cette classe sera utilisée comme classe utilitaire dans la classe implémentant la couche [dao]. Elle facilite les interactions avec les objets NHibernate. Cette classe a certaines propriétés qu'il faut initialiser :
  - ligne 40 : la *SessionFactory* de NHibernate
  - ligne 41 : la propriété *TemplateFlushMode* fixe le mode de synchronisation du contexte de persistance NHibernate avec la base de données. Le mode *Auto* fait qu'il y aura synchronisation :
    - à la fin d'une transaction
    - avant une opération select
  - ligne 42 : les requêtes HQL (Hibernate Query Language) seront mises en cache. Cela peut amener un gain de performance.
- les lignes 46-48 définissent la classe d'implémentation de la couche [dao]
  - ligne 46 : la couche [dao] va être implémentée par la classe [PamdaoSpringNHibernate] de la Dll [pam-dao-spring-nhibernate]. Après instanciation de la classe, la méthode *init* de la classe sera immédiatement exécutée. A la fermeture du conteneur Spring, la méthode *destroy* de la classe sera exécutée.
  - ligne 47 : la classe [PamDaoSpringNHibernate] aura une propriété *HibernateTemplate* qui sera initialisée avec la propriété *HibernateTemplate* de la ligne 39.

13.1.5 Implémentation de la couche [dao]

13.1.5.1 Le squelette de la classe d'implémentation

L'interface [IPamDao] est la même que dans le projet [pam-dao-nhibernate] :

```
1.  using Pam.Dao.Entites;
2.
3.  namespace Pam.Dao.Service {
4.   public interface IPamDao {
5.      // liste de toutes les identités des employés
6.      Employe[] GetAllIdentitesEmployes();
7.      // un employé particulier avec ses indemnités
8.      Employe GetEmploye(string ss);
9.      // liste de toutes les cotisations
10.     Cotisations GetCotisations();
11.  }
12. }
```

- ligne 1 : on importe l'espace de noms des entités de la couche [dao].
- ligne 3 : la couche [dao] est dans l'espace de noms [Pam.Dao.Service]. Les éléments de l'espace de noms [Pam.Dao.Entites] peuvent être créés en plusieurs exemplaires. Les éléments de l'espace de noms [Pam.Dao.Service] sont créés en un unique exemplaire (singleton). C'est ce qui a justifié le choix des noms des espaces de noms.
- ligne 4 : l'interface s'appelle [IPamDao]. Elle définit trois méthodes :
  - ligne 6, [GetAllIdentitesEmployes] rend un tableau d'objets de type [Employe] qui représente la liste des assistantes maternelles sous une forme simplifiée (nom, prénom, SS).
  - ligne 8, [GetEmploye] rend un objet [Employe] : l'employé qui a le n° de sécurité sociale passé en paramètre à la méthode avec les indemnités liées à son indice.
  - ligne 10, [GetCotisations] rend l'objet [Cotisations] qui encapsule les taux des différentes cotisations sociales à prélever sur le salaire brut.

Le squelette de la classe d'implémentation de cette interface avec le support Spring / NHibernate pourrait être le suivant :

```
1.  using System;
2.  using System.Collections;
3.  using System.Collections.Generic;
4.  using Pam.Dao.Entites;
5.  using Spring.Data.NHibernate.Generic.Support;
6.  using Spring.Transaction.Interceptor;
7.
8.  namespace Pam.Dao.Service {
9.    public class PamDaoSpringNHibernate : HibernateDaoSupport, IPamDao {
```

```
10.       // champs privés
11.       private Cotisations cotisations;
12.       private Employe[] employes;
13.
14.       // init
15.       [Transaction(ReadOnly = true)]
16.       public void init() {
17. ...
18.       }
19.
20.       // suppression objet
21.       public void destroy() {
22.          if (HibernateTemplate.SessionFactory != null) {
23.             HibernateTemplate.SessionFactory.Close();
24.          }
25.       }
26.
27.       // liste de toutes les identités des employés
28.       public Employe[] GetAllIdentitesEmployes() {
29.          return employes;
30.       }
31.
32.
33.       // un employé particulier avec ses indemnités
34.       [Transaction(ReadOnly = true)]
35.       public Employe GetEmploye(string ss) {
36. ....
37.       }
38.
39.       // liste des cotisations
40.       public Cotisations GetCotisations() {
41.          return cotisations;
42.       }
43.    }
44. }
```

- ligne 9 : la classe [PamDaoSpringNHibernate] implémente bien l'interface de la couche [dao] [IPamDao]. Elle dérive également de la classe Spring [HibernateDaoSupport]. Cette classe a une propriété [HibernateTemplate] qui est initialisée par la configuration Spring qui a été faite (ligne 2 ci-dessous) :

```
1.    <object id="pamdao" type="Pam.Dao.Service.PamDaoSpringNHibernate, pam-dao-spring-nhibernate"
    init-method="init" destroy-method="destroy">
2.       <property name="HibernateTemplate" ref="HibernateTemplate"/>
3. </object>
```

- ligne 1 ci-dessus, on voit que la définition de l'objet [pamdao] indique que les méthodes *init* et *destroy* de la classe [PamDaoSpringNHibernate] doivent être exécutées à des moments précis. Ces deux méthodes sont bien présentes dans la classe aux lignes 16 et 21.
- lignes 15, 34 : annotations qui font que la méthode annotée sera exécutée dans une transaction. L'attribut *ReadOnly=true* indique que la transaction est en lecture seule. La méthode qui est exécutée dans la transaction peut lancer une exception. Dans ce cas, un *Rollback* automatique de la transaction est fait par Spring. Cette annotation élimine le besoin de gérer une transaction au sein de la méthode.
- ligne 16 : la méthode *init* est exécutée par Spring immédiatement après l'instanciation de la classe. Nous verrons qu'elle a pour but d'initialiser les champs privés des lignes 11 et 12. Elle se déroulera dans une transaction (ligne 15).
- les méthodes de l'interface [IPamDao] sont implémentées aux lignes 28, 35 et 40.
- lignes 28-30 : la méthode [GetAllIdentitesEmployes] se contente de rendre l'attribut de la ligne 12 initialisé par la méthode *init*.
- lignes 40-42 : la méthode [GetCotisations] se contente de rendre l'attribut de la ligne 11 initialisé par la méthode *init*.

13.1.5.2 Les méthodes utiles de la classe HibernateTemplate

Nous utiliserons les méthodes suivantes de la classe *HibernateTemplate* :

IList<T> Find<T>(string requete_hql)	exécute la requête HQL et rend une liste d'objets de type T
IList<T> Find<T>(string requete_hql, object[])	exécute une requête HQL paramétrée par des ?. Les valeurs de ces paramètres sont fournis par le tableau d'objets.
IList<T> LoadAll<T>()	rend toutes les entités de type *T*

Il existe d'autres méthodes utiles que nous n'aurons pas l'occasion d'utiliser qui permettent de retrouver, sauvegarder, mettre à jour et supprimer des entités :

T Load<T>(object id)	met dans la session NHibernate l'entité de type T ayant la clé primaire *id*.
void SaveOrUpdate(object entité)	insère (INSERT) ou met à jour (UPDATE) l'objet *entité* selon que celui-ci a une clé primaire (UPDATE) ou non (INSERT). L'absence de clé primaire peut être configuré par l'attribut *unsaved-values* du fichier de configuration de l'entité. Après l'opération *SaveOrUpdate*, l'objet *entité* est dans la session NHibernate.
void Delete(object entité)	supprime l'objet *entité* de la session NHibernate.

13.1.5.3 Implémentation de la méthode init

La méthode *init* de la classe [PamDaoSpringNHibernate] est par configuration la méthode exécutée après instanciation par Spring de la classe. Elle a pour but de mettre en cache local, les identités simplifiées des employés (nom, prénom, SS) et les taux de cotisations. Son code pourrait être le suivant.

```
1.  [Transaction(ReadOnly = true)]
2.       public void init() {
3.          try {
4.             // on récupère la liste simplifiée des employés
5.             IList<object[]> lignes = HibernateTemplate.Find<object[]>("select e.SS,e.Nom,e.Prenom
    from Employe e");
6.             // on la met dans un tableau
7.             employes = new Employe[lignes.Count];
8.             int i = 0;
9.             foreach (object[] ligne in lignes) {
10.               employes[i] = new Employe() { SS = ligne[0].ToString(), Nom = ligne[1].ToString(),
    Prenom = ligne[2].ToString() };
11.               i++;
12.            }
13.            // on met les taux de cotisations dans un objet
14.            cotisations = (HibernateTemplate.LoadAll<Cotisations>())[0];
15.         } catch (Exception ex) {
16.            // on transforme l'exception
17.            throw new PamException(string.Format("Erreur d'accès à la BD : [{0}]", ex.ToString()),
    43);
18.         }
19.      }
```

- ligne 5 : une requête HQL est exécutée. Elle demande les champs SS, Nom, Prenom de toutes les entités *Employé*. Elle rend une liste d'objets. Si on avait demandé l'intégralité de l'employé sous la forme "select e from Employe e", on aurait récupéré une liste d'objets de type *Employe*.
- lignes 7-12 : cette liste d'objets est copiée dans un tableau d'ojets de type *Employe*.
- ligne 14 : on demande la liste de toutes les entités de type *Cotisations*. On sait que cette liste n'a qu'un élément. On récupère donc le premier élément de la liste pour avoir les taux de cotisations.
- les lignes 7 et 14 initialisent les deux champs privés de la classe.

13.1.5.4 Implémentation de la méthode GetEmploye

La méthode *GetEmploye* doit rendre l'entité Employé ayant un n° SS donné. Son code pourrait être le suivant :

```
1.  [Transaction(ReadOnly = true)]
2.       public Employe GetEmploye(string ss) {
3.          IList<Employe> employés = null;
4.          try {
5.             // requête
6.             employés = HibernateTemplate.Find<Employe>("select e from Employe e where e.SS=?", new
    object[]{ss});
7.          } catch (Exception ex) {
8.             // on transforme l'exception
9.             throw new PamException(string.Format("Erreur d'accès à la BD lors de la demande de
    l'employé de n° ss [{0}] : [{1}]", ss, ex.ToString()), 41);
10.         }
11.         // a-t-on récupéré un employé ?
```

```
12.          if (employés.Count == 0) {
13.             // on signale le fait
14.             throw new PamException(string.Format("L'employé de n° ss [{0}] n'existe pas", ss), 42);
15.          } else {
16.             return employés[0];
17.          }
18.       }
```

- ligne 6 : obtient la liste des employés ayant un n° SS donné
- ligne 12 : normalement si l'employé existe on doit obtenir une liste à un élément
- ligne 14 : si ce n'est pas le cas, on lance une exception
- ligne 16 : si c'est le cas, on rend le premier employé de la liste

13.1.5.5 Conclusion

Si on compare le code de la couche [dao] dans le cas d'une utilisation

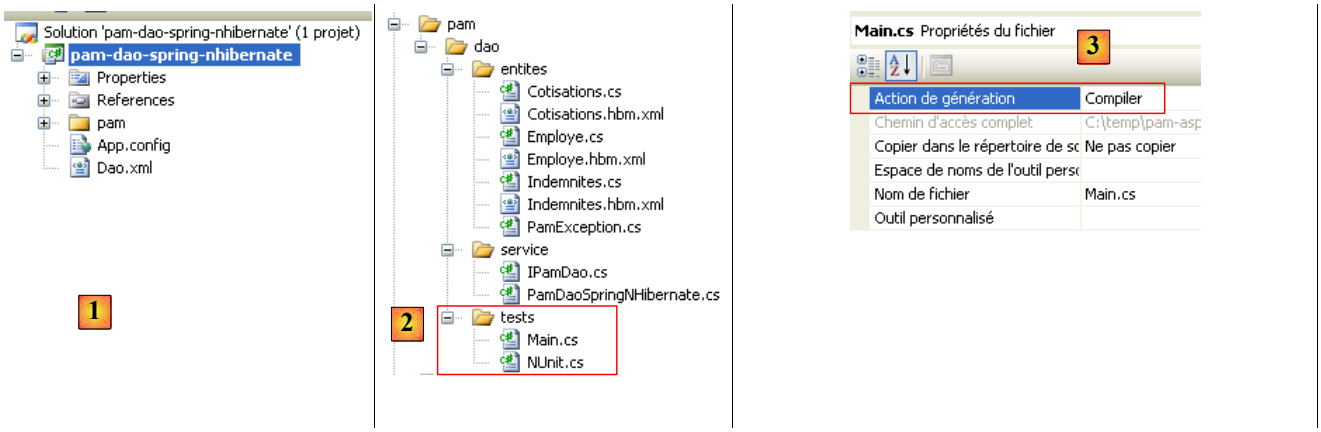
1. du seul framework NHibernate
2. du framework Spring / NHibernate

on réalise que la seconde solution a permis d'écrire du code plus simple.

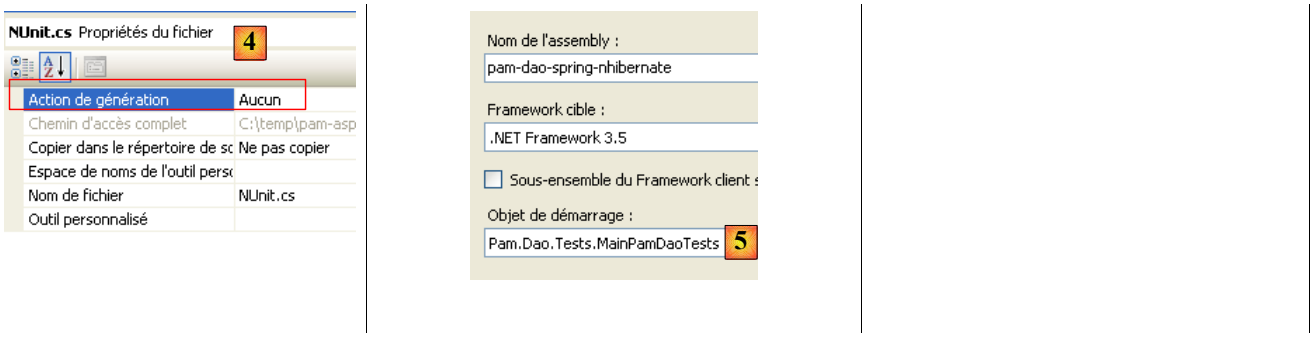
13.2 Tests de la couche [dao]

13.2.1 Le projet Visual Studio

Le projet Visual Studio a déjà été présenté. Rappelons-le :



- en [1], le projet dans sa globalité
- en [2], les différentes classes du projet. Le dossier [tests] contient un test console [Main.cs] et un test unitaire [NUnit.cs].
- en [3], le programme [Main.cs] est compilé.

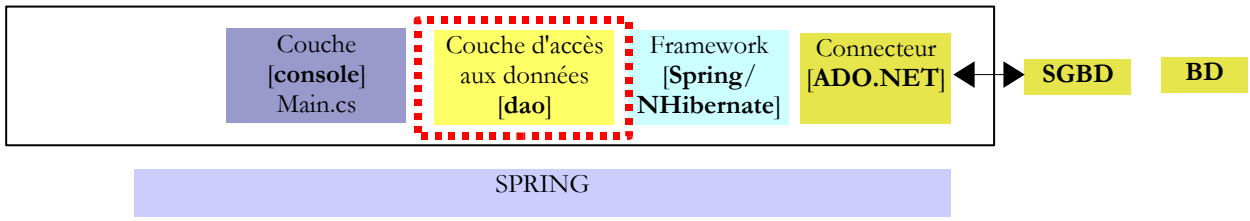


- en [4], le fichier [NUnit.cs] n'est pas généré.

- le projet est une application console. La classe exécutée est celle précisée en [5], la classe du fichier [Main.cs].

13.2.2 Le programme de test console [Main.cs]

Le programme de test [Main.cs] est exécuté dans l'architecture suivante :



Il est chargé de tester les méthodes de l'interface [IPamDao]. Un exemple basique pourrait être le suivant :

```
1.  using System;
2.  using Pam.Dao.Entites;
3.  using Pam.Dao.Service;
4.  using Spring.Context.Support;
5.
6.  namespace Pam.Dao.Tests {
7.   public class MainPamDaoTests {
8.      public static void Main() {
9.         try {
10.           // instanciation couche [dao]
11.           IPamDao pamDao = (IPamDao)ContextRegistry.GetContext().GetObject("pamdao");
12.           // liste des identités des employés
13.           foreach (Employe Employe in pamDao.GetAllIdentitesEmployes()) {
14.              Console.WriteLine(Employe.ToString());
15.           }
16.           // un employé avec ses indemnités
17.           Console.WriteLine("-----------------------------------");
18.           Console.WriteLine(pamDao.GetEmploye("254104940426058"));
19.           Console.WriteLine("-----------------------------------");
20.           // liste des cotisations
21.           Cotisations cotisations = pamDao.GetCotisations();
22.           Console.WriteLine(cotisations.ToString());
23.        } catch (Exception ex) {
24.           // affichage exception
25.           Console.WriteLine(ex.ToString());
26.        }
27.        //pause
28.        Console.ReadLine();
29.     }
30.  }
31. }
```

- ligne 11 : on demande à Spring une référence sur la couche [dao].
- lignes 13-15 : test de la méthode [GetAllIdentitesEmployes] de l'interface [IPamDao]
- ligne 18 : test de la méthode [GetEmploye] de l'interface [IPamDao]
- ligne 21 : test de la méthode [GetCotisations] de l'interface [IPamDao]

Spring, *NHibernate* et *log4net* sont configurés par le fichier [App.config] étudié au paragraphe 13.1.4.1, page 193.

L'exécution faite avec la base de données décrite au paragraphe 6.2, page 63, donne le résultat console suivant :

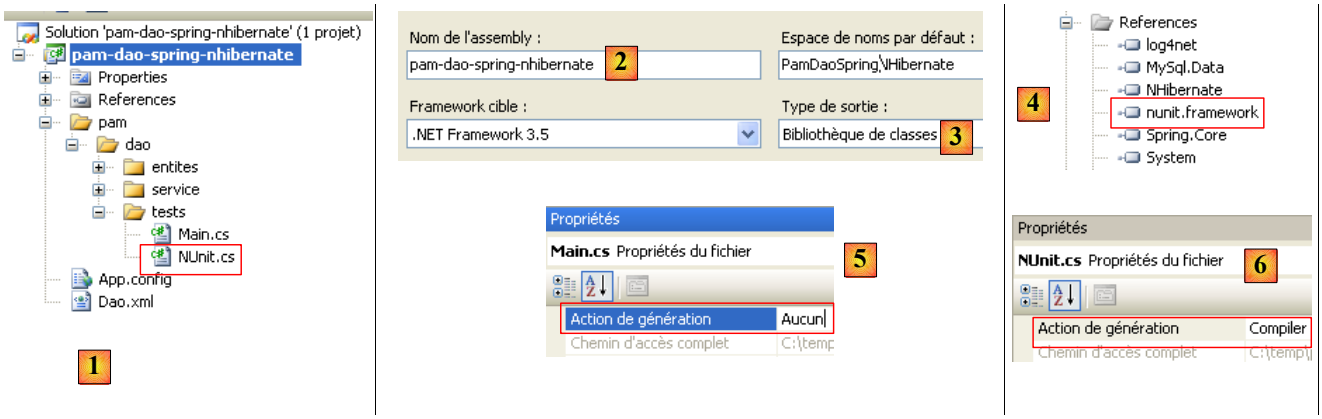
```
1.  [254104940426058,Jouveinal,Marie,,,,]
2.  [260124402111742,Laverti,Justine,,,,]
3.  -----------------------------------
4.  [254104940426058,Jouveinal,Marie,5 rue des oiseaux,St Corentin,49203,[2, 2,1, 2,1, 3,1, 15]]
5.  -----------------------------------
6.  [3,49,6,15,9,39,7,88]
```

- lignes 1-2 : les 2 employés de type [Employe] avec les seules informations [SS, Nom, Prenom]
- ligne 4 : l'employé de type [Employe] ayant le n° de sécurité sociale [254104940426058]

- ligne 5 : les taux de cotisations

13.2.3 Tests unitaires avec NUnit

Nous passons maintenant à un test unitaire NUnit. Le projet Visual Studio de la couche [dao] va évoluer de la façon suivante :



- en [1], le programme de test [NUnit.cs]
- en [2,3], le projet va générer une DLL nommé [pam-dao-spring-nhibernate.dll]
- en [4], la référence à la DLL du framework NUnit : [nunit.framework.dll]
- en [5], la classe [Main.cs] ne sera pas incluse dans la DLL [pam-dao-spring-nhibernate]
- en [6], la classe [NUnit.cs] sera incluse dans la DLL [pam-dao-spring-nhibernate]

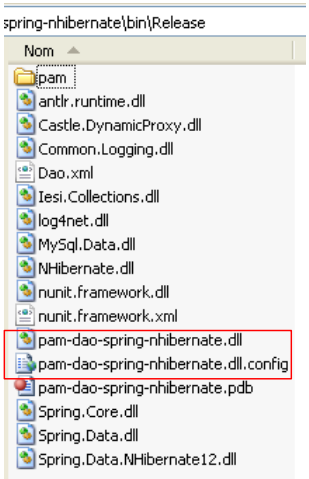
La classe de test NUnit est la suivante :

```
1.  using System.Collections;
2.  using NUnit.Framework;
3.  using Pam.Dao.Service;
4.  using Pam.Dao.Entites;
5.  using Spring.Objects.Factory.Xml;
6.  using Spring.Core.IO;
7.  using Spring.Context.Support;
8.
9.  namespace Pam.Dao.Tests {
10.
11.  [TestFixture]
12.  public class NunitPamDao : AssertionHelper {
13.     // la couche [dao] à tester
14.     private IPamDao pamDao = null;
15.
16.     // constructeur
17.     public NunitPamDao() {
18.        // instanciation couche [dao]
19.        pamDao = (IPamDao)ContextRegistry.GetContext().GetObject("pamdao");
20.     }
21.
22.     // init
23.     [SetUp]
24.     public void Init() {
25.
26.     }
27.
28.     [Test]
29.     public void GetAllIdentitesEmployes() {
30.        // vérification nbre d'employes
31.        Expect(2, EqualTo(pamDao.GetAllIdentitesEmployes().Length));
32.     }
33.
34.     [Test]
35.     public void GetCotisations() {
36.        // vérification taux de cotisations
37.        Cotisations cotisations = pamDao.GetCotisations();
38.        Expect(3.49, EqualTo(cotisations.CsgRds).Within(1E-06));
39.        Expect(6.15, EqualTo(cotisations.Csgd).Within(1E-06));
```

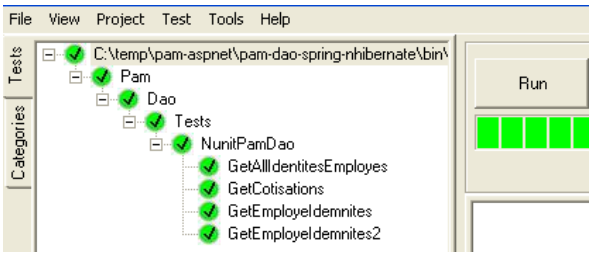
```
40.         Expect(9.39, EqualTo(cotisations.Secu).Within(1E-06));
41.         Expect(7.88, EqualTo(cotisations.Retraite).Within(1E-06));
42.     }
43.
44.     [Test]
45.     public void GetEmployeIdemnites() {
46.         // vérification individus
47.         Employe employe1 = pamDao.GetEmploye("254104940426058");
48.         Employe employe2 = pamDao.GetEmploye("260124402111742");
49.         Expect("Jouveinal", EqualTo(employe1.Nom));
50.         Expect(2.1, EqualTo(employe1.Indemnites.BaseHeure).Within(1E-06));
51.         Expect("Laverti", EqualTo(employe2.Nom));
52.         Expect(1.93, EqualTo(employe2.Indemnites.BaseHeure).Within(1E-06));
53.     }
54.
55.     [Test]
56.     public void GetEmployeIdemnites2() {
57.         // vérification individu inexistant
58.         bool erreur = false;
59.         try {
60.             Employe employe1 = pamDao.GetEmploye("xx");
61.         } catch {
62.             erreur = true;
63.         }
64.         Expect(erreur, True);
65.     }
66.  }
67. }
```

Cette classe a déjà été étudiée au paragraphe 7.3.4, page 98.

La génération du projet crée la DLL [pam-dao-spring-nhibernate.dll] dans le dossier [bin/Release].



On charge la DLL [pam-dao-spring-nhibernate.dll] avec l'outil [NUnit-Gui], version 2.4.6 et on exécute les tests :



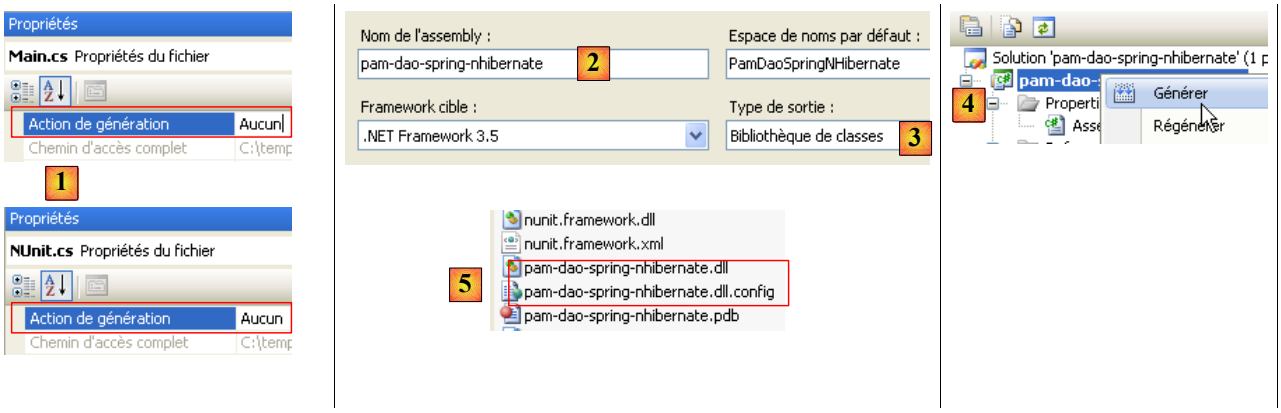
Ci-dessus, les tests ont été réussis.

Travail pratique :

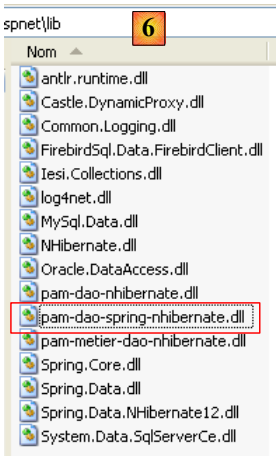
- mettre en oeuvre sur machine les tests de la classe [PamDaoSpringNHibernate].
- utiliser différents fichiers de configuration [Dao.xml] afin d'utiliser d'autres SGBD (Firebird, MySQL, Postgres, SQL Server)

13.2.4 Génération de la DLL de la couche [dao]

Une fois écrite et testée la classe [PamDaoNHibernate], on génèrera la DLL de la couche [dao] de la façon suivante :

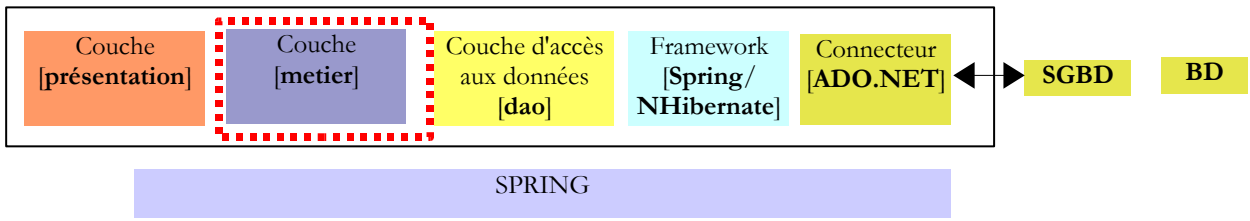


- [1], les programmes de test sont exlus de l'assembly du projet
- [2,3], configuration du projet
- [4], génération du projet
- la DLL est générée dans le dossier [bin/Release] [5]. Nous l'ajoutons aux DLL déjà présentes dans le dossier [lib] [6] :



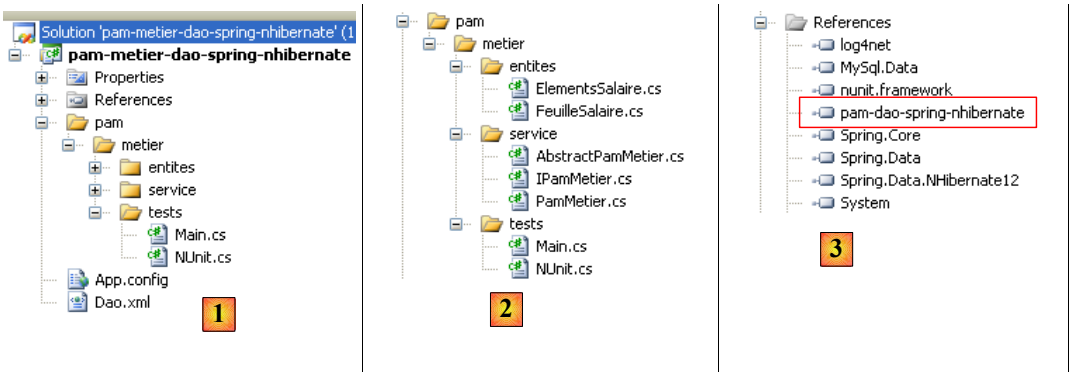
13.3 La couche métier

Revenons sur l'architecture générale de l'application [SimuPaie] :



Nous considérons désormais que la couche [dao] est acquise et qu'elle a été encapsulée dans la DLL [pam-dao-spring-nhibernate.dll]. Nous nous intéressons maintenant à la couche [metier]. C'est elle qui implémente les règles métier, ici les règles de calcul d'un salaire.

Le projet Visual Studio de la couche métier pourrait ressembler à ce qui suit :

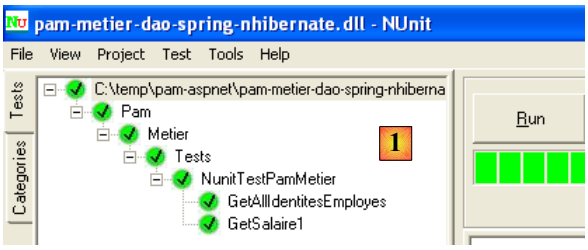


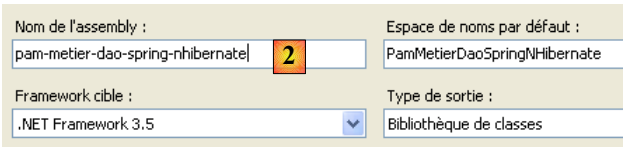
- en [1] l'ensemble du projet configuré par les fichiers [App.config] et [Dao.xml]. Le fichier [App.config] est identique à ce qu'il était dans le projet de la couche [dao] [pam-dao-spring-nhibernate]. Il en est de même pour le fichier [Dao.xml] si ce n'est qu'il déclare un objet Spring supplémentaire d'id *pammetier*. La déclaration de ce dernier est identique à ce qu'elle était dans le fichier [App.config] du projet [pam-metier-dao-nhibernate].
- en [2], le dossier [pam] est identique à ce qu'il était dans la couche [metier] du projet [pam-metier-dao-nhibernate]
- en [3] les références utilisées par le projet. On notera la DLL [pam-dao-spring-nhibernate] de la couche [dao] étudiée précédemment.

Question : construire le projet [pam-metier-dao-spring-nhibernate] ci-dessus. Il sera testé séparément :
- en mode console par le programme console [Main.cs]
- par le test unitaire [NUnit.cs] exécuté par le framework NUnit

Le nouveau projet [pam-metier-dao-spring-nhibernate] peut être construit par simple recopie du projet [pam-metier-dao-nhibernate] puis par modification des éléments qui doivent changer.

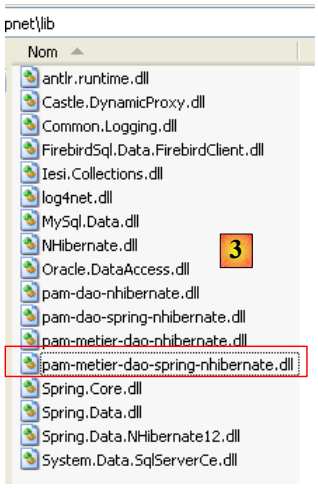
Après les tests, on génèrera la Dll de la couche [metier] qu'on appellera [pam-metier-dao-spring-nhibernate] :





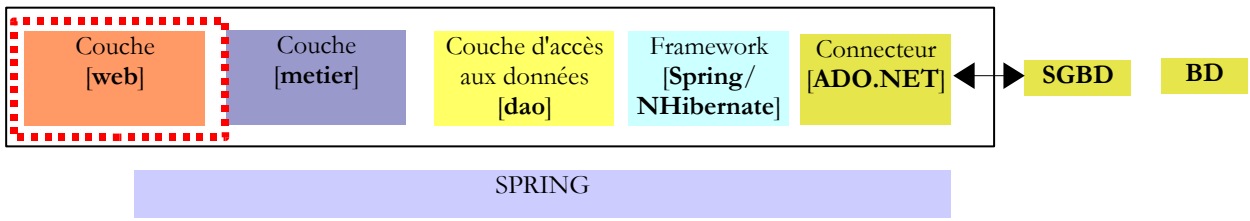
- en [1], le test NUnit réussi
- en [2], la Dll générée par le projet

On rajoutera la Dll de la couche [metier] aux Dll déjà présentes dans le dossier [lib] [3] :



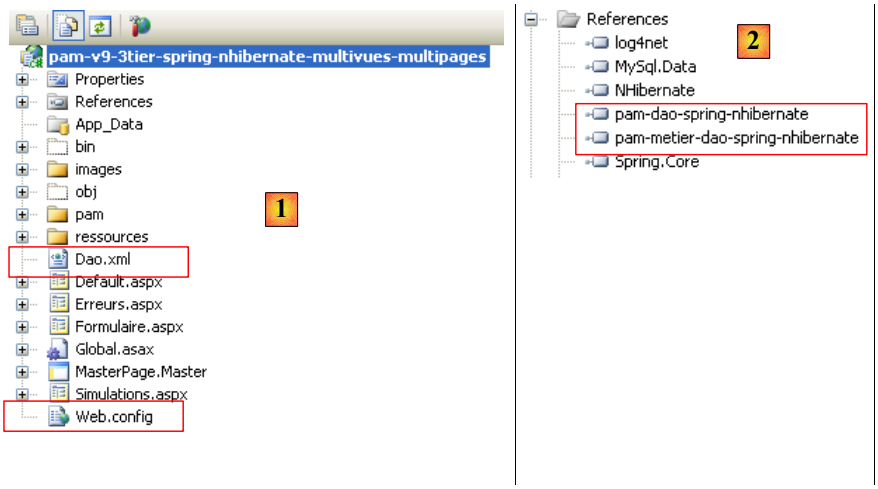
13.4 La couche [web]

Revenons sur l'architecture générale de l'application [SimuPaie] :



Nous considérons que les couches [dao] et [métier] sont acquises et encapsulées dans les DLL [pam-dao-spring-nhibernate, pam-metier-dao-spring-nhibernate]. Nous décrivons maintenant la couche web.

Le projet Visual Web Developer de la couche [web] est tout d'abord obtenu par simple recopie du dossier du projet web [pam-v7-3tier-nhibernate-multivues-multipages]. Le projet est ensuite renommé [pam-v9-3tier-spring-nhibernate-multivues-multipages] :



Le nouveau projet web [pam-v9-3tier-spring-nhibernate-multivues-multipages] diffère du projet [pam-v7-3tier-nhibernate-multivues-multipages] sur les points suivants :

- en [1], il est configuré par les fichiers [Dao.xml] et [Web.config]. [Dao.xml] n'existait pas dans [pam-v7] et le fichier [Web.config] doit intégrer la configuration Spring / NHibernate alors que dans [pam-v7], il ne configurait que NHibernate.
- en [2], les Dll des couches [dao] et [metier] sont celles que nous venons de construire.

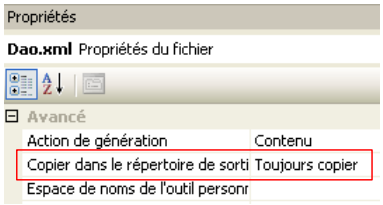
Le fichier [Dao.xml] est celui utilisé dans la construction de la couche [metier]. Le fichier [Web.config] est celui de [pam-v7] auquel on rajoute la configuration Spring / NHibernate que l'on trouvait dans les fichiers [App.config] des couches [dao] et [metier]. Le fichier [Web.config] de [pam-v9] est le suivant :

```
1.  <configuration>
2.
3.    <configSections>
4.      <sectionGroup name="system.web.extensions" type="System.Web.Configuration.SystemWebExtensionsSectionGroup,
    System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35">
5.  ........
6.      </sectionGroup>
7.      <sectionGroup name="spring">
8.        <section name="parsers" type="Spring.Context.Support.NamespaceParsersSectionHandler, Spring.Core" />
9.        <section name="objects" type="Spring.Context.Support.DefaultSectionHandler, Spring.Core" />
10.       <section name="context" type="Spring.Context.Support.ContextHandler, Spring.Core" />
11.     </sectionGroup>
12.     <section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler,log4net" />
13.   </configSections>
14.
15.   <!-- configuration Spring -->
16.   <spring>
17.     <parsers>
18.       <parser type="Spring.Data.Config.DatabaseNamespaceParser, Spring.Data" />
19.     </parsers>
20.     <context>
21.       <resource uri="~/Dao.xml" />
22.     </context>
23.   </spring>
24. ............. le reste est identique au fichier [Web.config] de [pam-v7]
```

Lignes 7-11 et 16-23 on retrouve la configuration Spring que l'on avait dans les fichiers [App.config] des couches [dao] et [metier] construites précédemment à une différence près : dans les fichiers [App.config], la ligne 17 était écrite comme suit :

```
      <resource uri="Dao.xml" />
```

Avec la configuration suivante :



le fichier [Dao.xml] est copié dans le dossier [bin] du dossier du projet web. Avec la syntaxe

```
<resource uri="Dao.xml" />
```

le fichier [Dao.xml] sera cherché dans le dossier courant du processus exécutant l'application web. Il se trouve que ce dossier n'est pas le dossier [bin] du dossier du projet web exécuté. Il faut écrire :

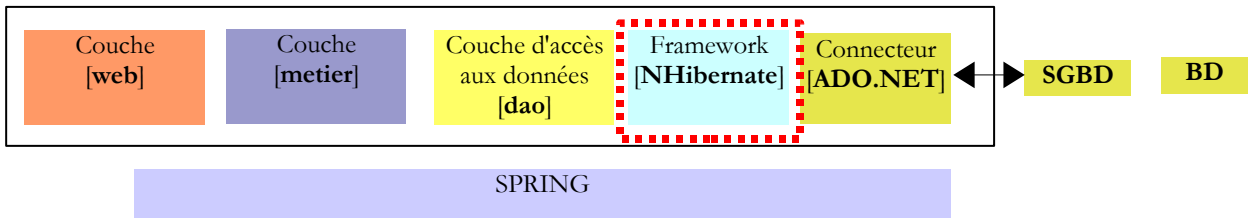
```
<resource uri="~/Dao.xml" />
```

pour que le fichier [Dao.xml] soit cherché dans le dossier [bin] du dossier du projet web exécuté.

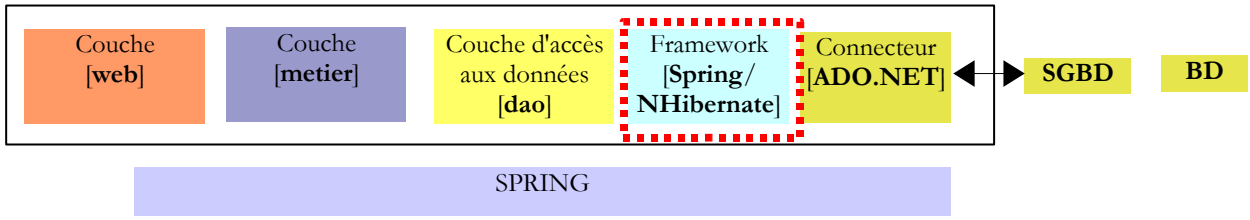
Question : mettre en oeuvre sur machine cette application web.

13.5 Conclusion

Nous sommes passés de l'architecture :



à l'architecture :



Il s'agissait d'implémenter la couche [dao] en profitant de facilités offertes par l'intégration de NHibernate par Spring.

Nous avons pu voir que cela :

- affectait la couche [dao]. Celle-ci a été plus simple à écrire mais a nécessité une configuration Spring plus complexe.
- affectait à la marge les couches [metier] et [web]

On a eu là un autre exemple de l'intérêt des architectures en couches.

14 L'application [SimuPaie] – version 10 – client Flex d'un service web ASP.NET

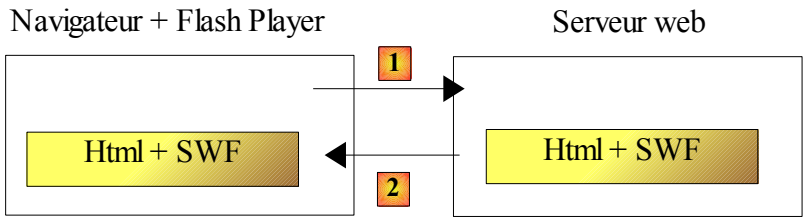
Nous présentons maintenant un client **Flex** du service web ASP.NET de la version 5. L'IDE utilisé est Flex Builder 3. Une version de démonstration de ce produit est téléchargeable à l'Url [https://www.adobe.com/cfusion/tdrc/index.cfm?loc=fr_fr&product=flex]. Flex Builder 3 est un IDE Eclipse. Par ailleurs, pour exécuter le client Flex, nous utilisons un serveur web Apache de l'outil **Wamp** [http://www.wampserver.com/]. N'importe quel serveur **Apache** fait l'affaire. Le navigateur qui affiche le client Flex doit disposer du plugin **Flash Player** version 9 minimale.

Les applications Flex ont la particularité de s'exécuter au sein du plugin Flash Player du navigateur. En cela elles se rapprochent des applications Ajax qui embarquent dans les pages envoyées au navigateur des scripts JavaScript qui sont ensuite exécutés au sein du navigateur. Une application Flex n'est pas une application web au sens où on l'entend habituellement : c'est une application cliente de services délivrés par des serveurs web. En cela, elle est analogue à une application de bureau qui serait cliente de ces mêmes services. Elle diffère cependant en un point : elle est téléchargée initialement depuis un serveur web au sein d'un navigateur disposant du plugin Flash Player capable de l'exécuter.

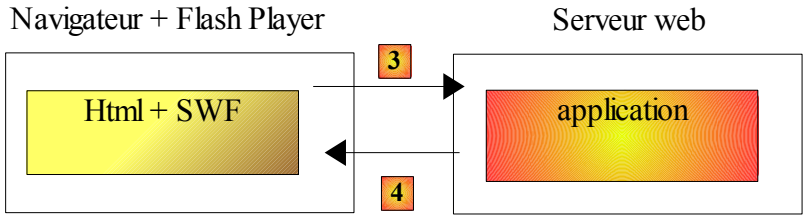
Comme une application de bureau, une application Flex est composée principalement de deux éléments :

- une partie **présentation** : les vues affichées dans le navigateur. Ces vues ont la richesse des fenêtres des applications de bureau. Une vue est décrite à l'aide d'un langage à balises appelé **MXML**.
- une partie **code** qui gère principalement les événements provoqués par les actions de l'utilisateur sur la vue. Ce code peut être écrit également en **MXML** ou avec un langage orienté objet appelé **ActionScript**. Il faut distinguer deux types d'événements :
  - l'événement qui nécessite d'avoir un échange avec le serveur web : remplissage d'une liste par des données fournies par une application web, envoi des données d'un formulaire au serveur, ... Flex fournit un certain nombre de méthodes pour communiquer avec le serveur de façon transparente pour le développeur. Ces méthodes sont par défaut **asynchrones** : l'utilisateur peut continuer à interagir avec la vue pendant la requête au serveur.
  - l'événement qui modifie la vue affichée sans échange de données avec le serveur, par exemple tirer un élément d'un arbre pour le déposer dans une liste. Ce type d'événement est entièrement traité localement au sein du navigateur.

Une application Flex est souvent exécutée de la façon suivante :



- en [1], une page Html est demandée
- en [2], elle est envoyée. Elle embarque avec elle un fichier binaire **SWF** (**S**hock**W**ave **F**lash) contenant l'intégralité de l'application Flex : toutes les vues et le code de gestion des événements de celles-ci. Ce fichier sera exécuté par le plugin Flash Player du navigateur.



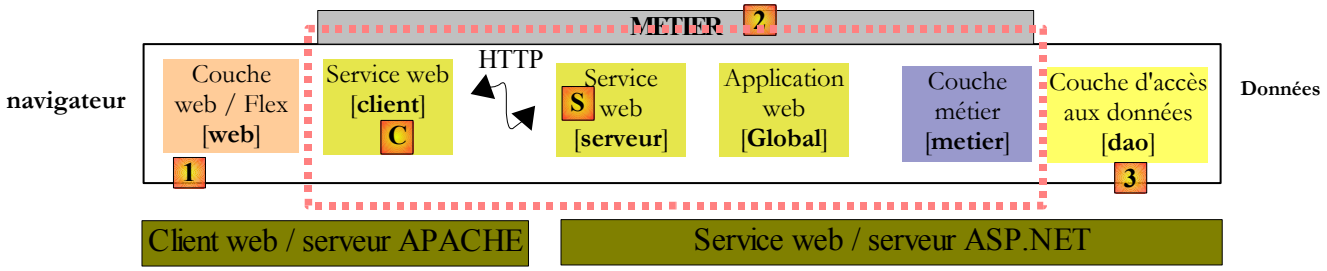
- l'exécution du client Flex se fait en local sur le navigateur sauf lorsqu'il a besoin de données externes. Dans ce cas, il les demande au serveur [3]. Il les reçoit en [4] selon des formats divers : XML ou binaire. L'application interrogée sur le serveur web peut être écrite dans un langage quelconque. Seul compte le format de la réponse.

Nous avons décrit l'architecture d'exécution d'une application Flex afin que le lecteur perçoive la différence entre celle-ci et celle d'une application Web classique, où les pages n'embarquent pas de code (Javascript, Flex, Silverlight, ...) que le navigateur

exécuterait. Dans cette dernière, le navigateur est passif : il affiche simplement des pages Html construites sur le serveur web qui les lui envoie.

14.1 Architecture de l'application client / serveur

L'architecture client / serveur mise en place est analogue à celles des versions 6 et 8 :

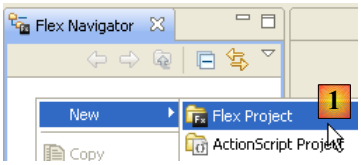


En [1], la couche web ASP.NET est remplacée par une couche web Flex écrite en MXML et ActionScript. Le client [C] sera généré par l'IDE Flex Builder. Il faut rappeler ici que cette architecture comporte deux serveurs web non représentés :

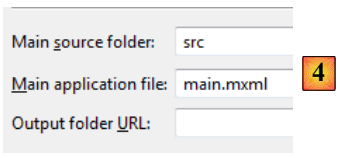
- un serveur web ASP.NET qui exécute le service web [S]
- un serveur web APACHE qui exécute le client web [1]

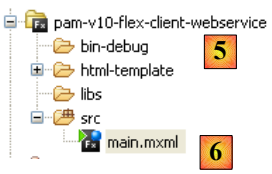
14.2 Le projet Flex 3 du client

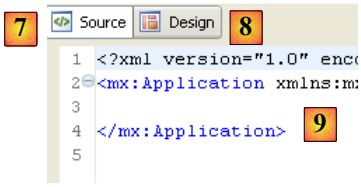
Nous construisons le client Flex avec l'IDE Flex Builder 3 :



- dans Flex Builder 3, on crée un nouveau projet en [1]
- on lui donne un nom en [2] et on précise en [3] dans quel dossier le générer







- en [4], on donne un nom à l'application principale (celle qui va être exécutée)
- en [5], le projet une fois généré
- en [6], le fichier MXML principal de l'application
- un fichier MXML comporte une vue et le code de gestion des événements de celle-ci. L'onglet [Source] [7] donne accès au fichier MXML. On y trouvera des balises <*mx*> décrivant la vue ainsi que du code ActionScript.
- la vue peut être construite graphiquement en utilisant l'onglet [Design] [8]. Les balises MXML décrivant la vue sont alors générées automatiquement dans l'onglet [Source]. L'inverse est vrai : les balises MXML ajoutées directement dans l'onglet [Source] sont reflétées graphiquement dans l'onglet [Design].

14.3 La vue n° 1

Nous allons construire progressivement une interface web analogue à celle de la version 1 (cf page 43). Nous construisons tout d'abord l'interface suivante :

- en [1], la vue lorsque la connexion au service web a pu se faire. Le combo des employés est alors rempli.
- en [2], la vue lorsque la connexion au service web n'a pu se faire. Un message d'erreur est alors affiché.

Le fichier principal [main.xml] du client est le suivant :

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
3.     creationComplete="init()">
4.     <mx:VBox width="100%">
5.        <mx:Label text="Feuille de salaire" fontSize="30"/>
6.        <mx:HBox>
7.           <mx:VBox>
8.              <mx:Label text="Employés"/>
9.              <mx:ComboBox id="cmbEmployes" dataProvider="{employes}" labelFunction="displayEmploye"/>
10.          </mx:VBox>
11.          <mx:VBox>
12.             <mx:Label text="Heures travaillées"/>
13.             <mx:TextInput id="txtHeuresTravaillees"/>
14.          </mx:VBox>
15.          <mx:VBox>
16.             <mx:Label text="Jours travaillés"/>
17.             <mx:NumericStepper id="joursTravailles" minimum="0" maximum="31" stepSize="1"/>
18.          </mx:VBox>
19.          <mx:VBox>
20.             <mx:Label text=""/>
21.             <mx:Button id="btnSalaire" label="Salaire"/>
22.          </mx:VBox>
23.       </mx:HBox>
24.       <mx:TextArea id="msg" minWidth="400" minHeight="100" editable="false" visible="true" enabled="true"
    horizontalScrollPolicy="auto" verticalScrollPolicy="auto" x="0" y="0" maxHeight="100" maxWidth="400"/>

25.    </mx:VBox>
26.
27.    <mx:WebService ...>
28.       ...
29.    </mx:WebService>
30.
31.    <mx:Script>
32.       <![CDATA[
33. ...
34.          // données
35.          [Bindable]
36.          private var employes : ArrayCollection;
37.
38.          private function init():void{
39. ...
40.          }
41.       ]]>
42.    </mx:Script>
43. </mx:Application>
```

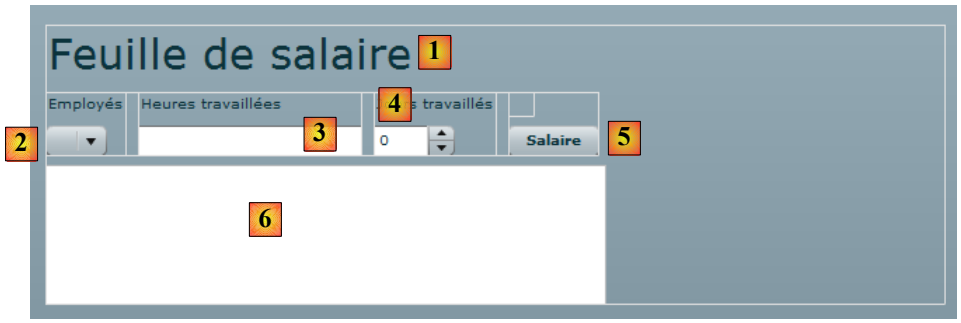
Dans ce code, il faut distinguer diverses choses :

- la définition de l'application (lignes 2-3)
- la description de la vue de celle-ci (lignes 4-25)
- les gestionnaires d'événements en langage ActionScript au sein de la balise <mx:Script> (lignes 31-42).
- la définition du service web distant (lignes 27-29)

Commentons pour commencer, la définition de l'application elle-même et la description de sa vue :

- lignes 2-3 : définissent :
  - le mode de disposition des composants dans le conteneur de la vue. L'attribut *layout="vertical"* indique que les composants seront les uns sous les autres.
  - la méthode à exécuter lorsque la vue aura été instanciée, c.a.d. le moment où tous ses composants auront été instanciés. L'attribut *creationComplete="init();"* indique que c'est la méthode *init* de la ligne 38 qui doit être exécutée. *creationComplete* est l'un des événements que peut émettre la classe *Application*.
- les lignes 4-25 définissent les composants de la vue
- lignes 4-25 : un conteneur vertical : les composants y seront placés les uns sous les autres
- ligne 5 : définit un texte
- lignes 6-23 : un conteneur horizontal : les composants seront placés horizontalement dans celui-ci.
- lignes 7-10 : un conteneur vertical qui contiendra un texte et une liste déroulante
- ligne 8 : le texte
- ligne 9 : la liste déroulante dans laquelle on mettra la liste des employés. La balise *dataProvider="{employes}"* indique la source des données qui doivent remplir la liste. Ici, la liste sera remplie avec l'objet *employes* défini ligne 36. Pour pouvoir écrire *dataProvider="{employes}"*, il faut que le champ *employes* ait l'attribut [Bindable] (ligne 35). Cet attribut permet à une variable ActionScript d'être référencée à l'extérieur de la balise <*mx*::*Script*>. Le champ *employes* est de type *ArrayCollection*, un type ActionScript qui permet de stocker des listes d'objets, ici une liste d'objets de type *Employe*.
- lignes 11-14 : un conteneur vertical qui contiendra un texte et une zone de saisie
- ligne 12 : le texte
- ligne 13 : la zone de saisie des heures travaillées.
- lignes 15-18 : un conteneur vertical qui contiendra un texte et un incrémenteur
- ligne 16 : le texte
- ligne 17 : l'incrémenteur qui permettra la saisie des jours travaillés.
- lignes 19-22 : un conteneur vertical qui contiendra un texte et un bouton qui déclenchera le calcul du salaire de la personne sélectionnée dans le combo.
- ligne 20 : le texte
- ligne 21 : le bouton.
- ligne 23 : fin du conteneur horizontal commencé ligne 6
- ligne 24 : une zone de texte dans un composant de type *TextArea*. Il affichera les messages d'erreur.
- ligne 25 : fin du conteneur vertical commencé ligne 4

Les lignes 4-25 génèrent la vue suivante dans l'onglet [Design] :



- [1] : a été généré par le composant *Label* de la ligne 5
- [2] : a été généré par le composant *ComboBox* de la ligne 9
- [3] : a été généré par le composant *TextInput* de la ligne 13
- [4] : a été généré par le composant *NumericStepper* de la ligne 17
- [5] : a été généré par le composant *Button* de la ligne 21
- [6] : a été généré par le composant *TextArea* de la ligne 24

Examinons maintenant la déclaration du service web distant :

```
1.  <mx:WebService id="pam"
2.        wsdl="http://localhost:1077/Service1.asmx?WSDL"
3.        fault="wsFault(event);"
```

```
4.       showBusyCursor="true">
5.       <mx:operation
6.          name="GetAllIdentitesEmployes"
7.          result="loadEmployesCompleted(event)"
8.          fault="loadEmployesFault(event);">
9.          <mx:request/>
10.      </mx:operation>
11.   </mx:WebService>
12.
```

- ligne 1 : le service web est un composant d'identifiant *pam* (attribut id)
- ligne 2 : l'Uri du fichier WSDL du service web (cf page 139)
- ligne 3 : la méthode à exécuter en cas d'erreur lors des échanges avec le service web : la méthode *wsFault*.
- ligne 4 : demande à ce qu'un indicateur soit affiché pour montrer à l'utilisateur qu'un échange avec le service web est en cours.
- lignes 5-10 : l'une des opérations offertes par le service web distant. Ici, la méthode *GetAllIdentitesEmployes*.
- ligne 7 : la méthode à exécuter lorsque l'appel de cette méthode se termine normalement, c.a.d. lorsque le service web renvoie bien la liste des employés
- ligne 8 : la méthode à exécuter lorsque l'appel de cette méthode se termine sur une erreur.
- ligne 9 : les paramètres de l'opération *GetAllIdentitesEmployes*. On sait que cette méthode n'attend pas de paramètres. Aussi laisse-t-on la balise <*mx:request*> vide.

Examinons maintenant le code ActionScript lié au service web :

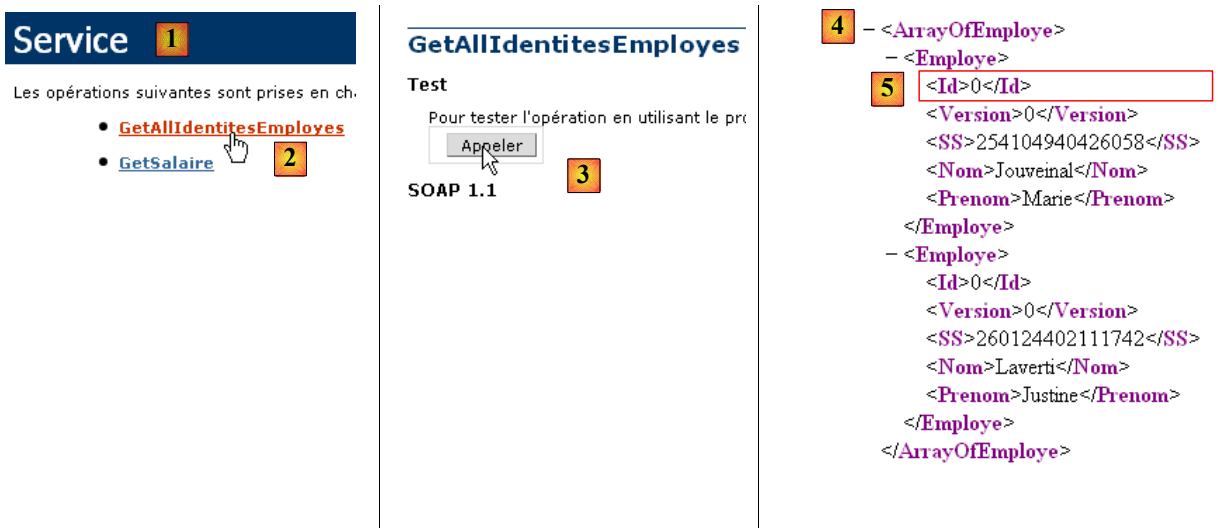
```
1.  <mx:Script>
2.       <![CDATA[
3.          import mx.rpc.events.FaultEvent;
4.          import mx.collections.ArrayCollection;
5.          import mx.rpc.events.ResultEvent;
6.
7.          // données
8.          [Bindable]
9.          private var employes : ArrayCollection;
10.
11.         private function init():void{
12.            // on note les coordonnées de la zone de message
13.            msgHeight=msg.height;
14.            msgWidth=msg.width;
15.            // on cache la zone de message
16.            hideMsg();
17.            // requête au service web distant pour avoir la liste simplifiée des employés
18.            pam.GetAllIdentitesEmployes.send();
19.         }
20.
21.         private function wsFault(event:Event):void{
22.               // on signale l'erreur
23.               msg.text="Service distant indisponible";
24.               showMsg();
25.         }
26.
27.         private function loadEmployesCompleted(event:ResultEvent):void{
28.            // remplissage combo des employés
29.            employes=event.result as ArrayCollection;
30.         }
31.
32.         private function displayEmploye(employe:Object):String{
33.            // identité d'un employé
34.            return employe.Prenom + " " + employe.Nom;
35.         }
36.
37.         private function loadEmployesFault(event:FaultEvent):void{
38.            // affichage msg d'erreur
39.            msg.text=event.fault.message;
40.            // formulaire
41.            showMsg();
42.         }
43.
44.   // gestion des blocs
45.      private var msgWidth:int;
46.      private var msgHeight:int;
47.
48.      private function hideMsg():void{
49.         msg.height=0;
50.         msg.width=0;
51.      }
```

```
52.
53.       private function showMsg():void{
54.          msg.height=msgHeight;
55.          msg.width=msgWidth;
56.       }
57.
58.
59.       ]]>
60.    </mx:Script>
```

- ligne 11 : la méthode *init* est exécutée au démarrage de l'application parce qu'on a écrit :

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
  creationComplete="init()">
```

- lignes 13-14 : on mémorise les hauteur et largeur de la zone de message. On utilise deux méthodes *hideMsg* (lignes 48-51) et *showMsg* (lignes 53-56) pour respectivement cacher / montrer la zone de message selon qu'il y a eu erreur ou non. La méthode *hideMsg* cache la zone de message en mettant ses hauteur / largeur à 0. La méthode *showMsg* affiche la zone de message en lui restituant ses hauteur / largeur mémorisées dans la méthode *init*.
- ligne 16 : on cache la zone de message. Au départ, il n'y a pas d'erreur.
- ligne 18 : la méthode *GetAllIdentitesEmploye* (ligne 6 du service web) du service web *pam* (ligne 1 du service web) est appelée. L'appel est asynchrone. La ligne 7 du service web indique que la méthode *loadEmployesCompleted* sera exécutée si cet appel asynchrone se termine bien. La ligne 8 du service web indique que la méthode *loadEmployesFault* sera exécutée si cet appel asynchrone se termine mal.
- ligne 27 : la méthode *loadEmployesCompleted* qui est exécutée si l'appel au service web de la ligne 18 se termine bien.
- ligne 29 : on sait que le service web renvoie une réponse XML. Il est bon de revenir à celle-ci pour comprendre le code ActionScript :



- en [1], page du service web [Service.asmx]
- en [2], le lien vers la page de test de la méthode [GetAllIdentitesEmployes]
- en [3], le test est fait. Aucun paramètre n'est attendu.
- en [4] : la réponse XML contient un tableau d'employés. Pour chacun d'entre-eux, on a cinq informations encapsulées dans les balises <Id>, <Version>,<*SS*>, <*Nom*>, <*Prenom*>. Si la réponse XML est mise dans un tableau *employes* de type *ArrayCollection* :
  - *employes.getItemAt(i)* : est l'élément n° i du tableau
  - *employes.getItemAt(i).SS* : est le n° de sécurité sociale de cet employé.
  - *employes.getItemAt(i).Nom* : est le nom de cet employé
  - ...

Revenons au code ActionScript :

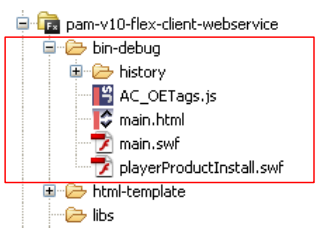
- ligne 29 : *event.result* représente la réponse XML du service web. La méthode *GetAllIdentitesEmployes* renvoie un tableau d'employés. *event.result* représente ce tableau d'employés. Il est placé dans une variable de type *ArrayCollection*, un type représentant de façon générale une collection d'objets. Cette variable nommée *employes* est déclarée ligne 9. On se rappelle que cette variable est la source de données du combo des employés :

```
<mx:ComboBox id="cmbEmployes" dataProvider="{employes}" labelFunction="displayEmploye"/>
```

Pour chaque employé de sa source de données, le combo va appeler la méthode *displayEmploye* (attribut *labelFunction*) pour afficher l'employé. On voit lignes 32-34 que cette méthode affiche les nom et prénom de l'employé.

- ligne 37 : la méthode *loadEmployesFault* qui est exécutée si l'appel au service web de la ligne 18 se termine mal. *event.fault.message* est le message d'erreur renvoyé par le service web.
- ligne 39 : ce message d'erreur est placé dans la zone de message
- ligne 41 : la zone de message est affichée.

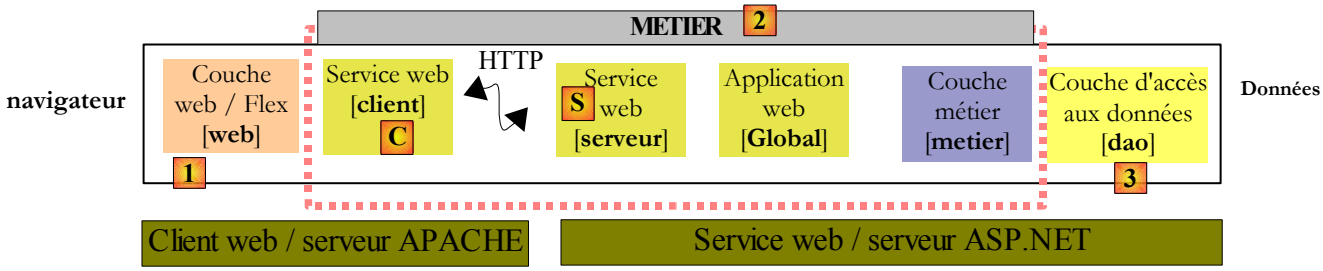
Lorsque l'application a été construite, son code exécutable se trouve dans le dossier [bin-debug] du projet Flex :



Ci-dessus,

- le fichier [main.html] représente le fichier Html qui sera demandé par le navigateur au serveur web pour avoir le client Flex
- le fichier [main.swf] est le binaire du client Flex qui sera encapsulé dans la page Html envoyée au navigateur puis exécuté par le plugin Flash Player de celui-ci.

Nous sommes prêts à exécuter le client Flex. Il nous faut auparavant mettre en place l'environnement d'exécution qui lui est nécessaire. Revenons à l'architecture client / serveur testée :



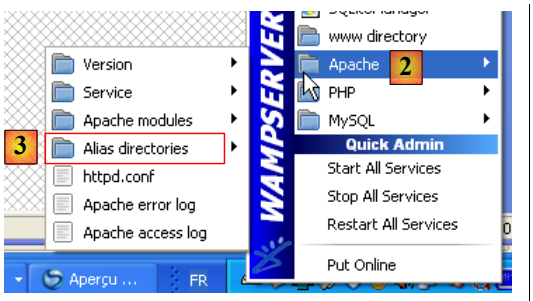
Côté serveur :

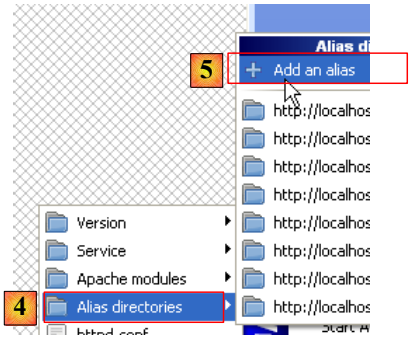
- lancer le service web Asp.net [S]

Côté client :

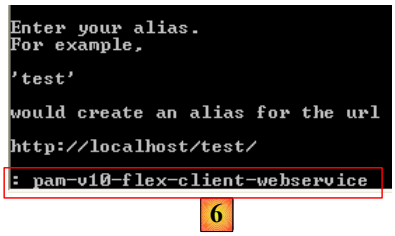
- lancer le serveur Apache à qui sera demandée l'application Flex.

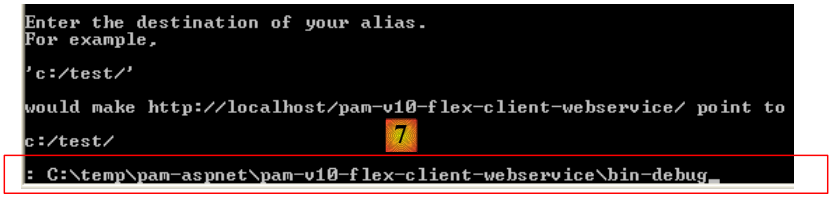
Ici nous utilisons l'outil *Wamp*. Avec cet outil, nous pouvons associer un alias au dossier [bin-debug] du projet Flex.





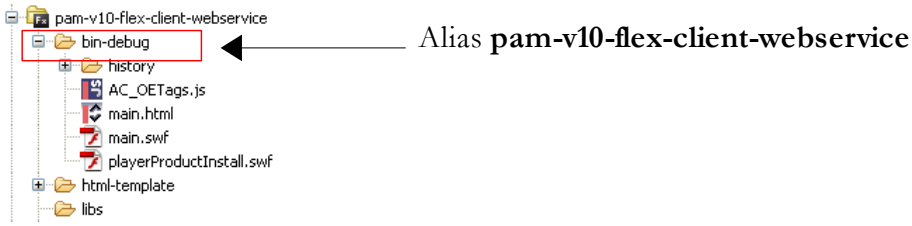
- l'icône de Wamp est en bas de l'écran [1]
- par un clic gauche sur l'icône *Wamp*, sélectionner l'option *Apache* [2] / *Alias Directories* [3, 4]
- sélectionner l'option [5] : *Add un alias*



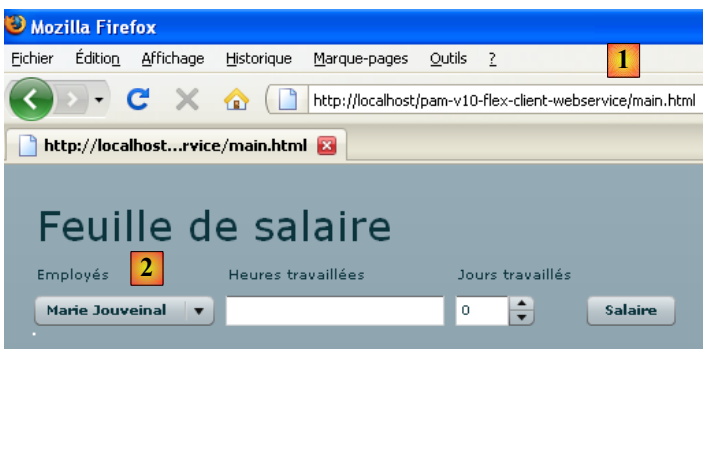


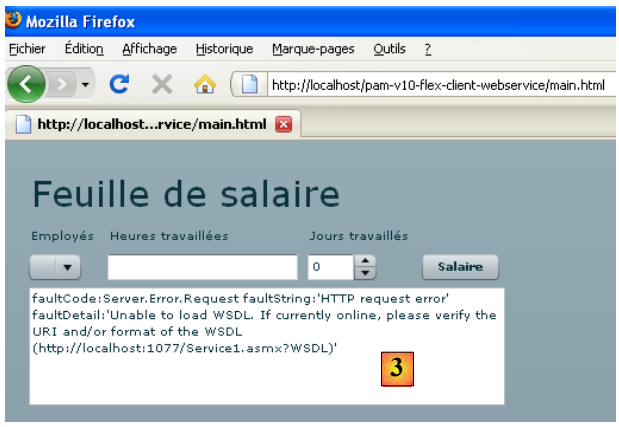
- en [6] donner un alias (un nom quelconque) à l'application web qui va être exécutée
- en [7] indiquer la racine de l'application web qui portera cet alias : c'est le dossier [bin-debug] du projet Flex que nous venons de construire.

Rappelons la structure du dossier [bin-debug] du projet Flex :



Le fichier [main.html] est le fichier Html de l'application Flex. Grâce à l'alias que nous venons de créer sur le dossier [bin-debug], ce fichier sera obtenu par l'url [http://localhost/pam-v10-flex-client-webservice/main.html]. Nous demandons celle-ci dans un navigateur disposant du plugin Flash Player version 9 ou supérieure :





- en [1], l'Url de l'application Flex
- en [2], le combo des employés lorsque tout va bien
- en [3], le résultat obtenu lorsque le service web est arrêté

Vous pourriez avoir la curiosité d'afficher le code source de la page Html reçue :

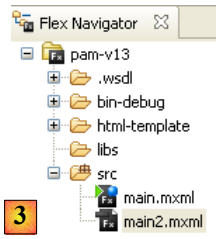
```
1.  <!-- saved from url=(0014)about:internet -->
2.  <html lang="en">
3.
4.  <!--
5.  Smart developers always View Source.
```

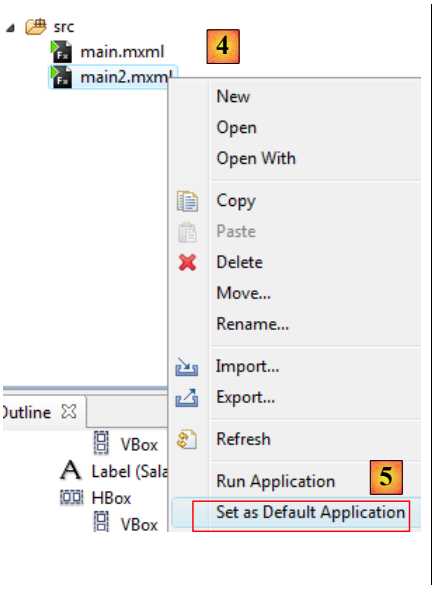
```
6.
7.  This application was built using Adobe Flex, an open source framework
8.  for building rich Internet applications that get delivered via the
9.  Flash Player or to desktops via Adobe AIR.
10.
11. Learn more about Flex at http://flex.org
12. // -->
13.
14. <head>
15. <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
16.
17. <!--  BEGIN Browser History required section -->
18. <link rel="stylesheet" type="text/css" href="history/history.css" />
19. <!--  END Browser History required section -->
20.
21. <title></title>
22. ....
23. </head>
24.
25. <body scroll="no">
26. ...
27. <noscript>
28.         <object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
29.                         id="main" width="100%" height="100%"
30.                         codebase="http://fpdownload.macromedia.com/get/flashplayer/current/swflash
    .cab">
31.                         <param name="movie" value="main.swf" />
32.                         <param name="quality" value="high" />
33.                         <param name="bgcolor" value="#869ca7" />
34.                         <param name="allowScriptAccess" value="sameDomain" />
35.                         <embed src="main.swf" quality="high" bgcolor="#869ca7"
36.                                width="100%" height="100%" name="main" align="middle"
37.                                play="true"
38.                                loop="false"
39.                                quality="high"
40.                                allowScriptAccess="sameDomain"
41.                                type="application/x-shockwave-flash"
42.                                pluginspage="http://www.adobe.com/go/getflashplayer">
43.                         </embed>
44.         </object>
45. </noscript>
46. </body>
47. </html>
```

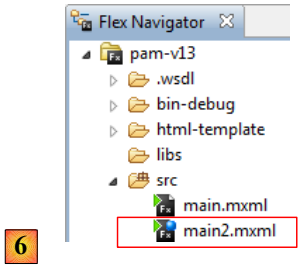
Le corps de la page commence ligne 25. Il ne contient pas du Html classique mais un objet (ligne 28) de type "application/x-shockwave-flash" (ligne 41). C'est le fichier [main.swf] (ligne 31) que l'on peut voir dans le dossier [bin-debug] du projet Flex. C'est un fichier de taille importante : 600 K environ pour ce simple exemple.

14.4 La vue n° 2

Nous allons ajouter un nouveau conteneur de type *VBox* à la vue actuelle :

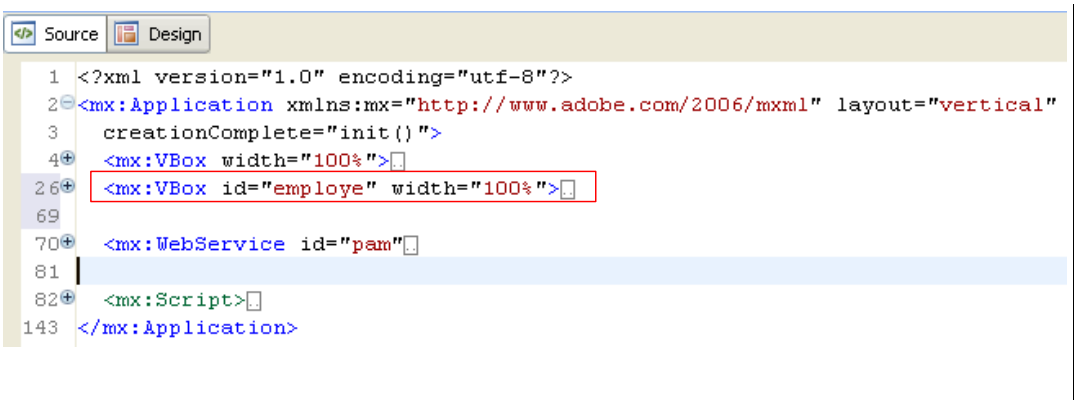






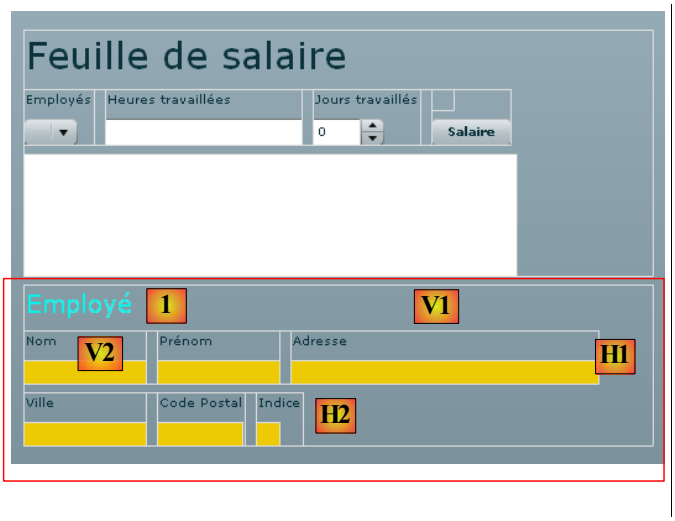
- en [4,5], nous faisons de [main2.mxml] la nouvelle application par défaut. C'est elle qui sera désormais compilée.
- en [6], l'application par défaut est désignée par un point bleu.

Le conteneur [1] affichera les informations concernant l'employé sélectionné dans le combo [2]. Nous dupliquons [main.xml] en [main2.xml] [3] pour construire la nouvelle vue. Nous travaillerons désormais avec [main2.xml].



La modification faite au projet précédent est l'ajout du conteneur de la ligne 26 ci-dessus qui contient le code MXML du conteneur [1] de la vue. Nous lui donnons l'identifiant *employe* afin de pouvoir le manipuler par du code. En effet ce conteneur devra pouvoir être caché / montré par la même technique qu'utilisée précédemment pour la zone de message.

Revenons au visuel de la vue :



Repérons les différents conteneurs des nouvelles informations affichées :

- *V1* : conteneur vertical de l'ensemble des composants : le libellé *Employé* [1] et les conteneurs horizontaux [H1] et [H2]
- *H1* : conteneur horizontal pour les informations *Nom*, *Prénom*, *Adresse*
- *V2* : conteneur vertical pour le libellé *Nom* et l'affichage du nom de l'employé.
- *H2* : conteneur horizontal pour les informations *Ville*, *Code Postal*, *Indice*

Le code complet du conteneur "employe" est le suivant :

```
1.  <mx:VBox id="employe" width="100%">
2.        <mx:Label text="Employé" fontSize="20" color="#09F3EB"/>
3.        <mx:HBox>
4.        <mx:VBox >
5.           <mx:Label text="Nom"/>
6.           <mx:VBox backgroundColor="#EECA05">
7.           <mx:Text id="lblNom" minWidth="100" minHeight="20" fontFamily="Verdana"
    textAlign="center"/>
8.           </mx:VBox>
9.        </mx:VBox>
10.       <mx:VBox >
11.          <mx:Label text="Prénom"/>
12.          <mx:VBox backgroundColor="#EECA05">
13.          <mx:Text id="lblPreNom" minWidth="100" minHeight="20" fontFamily="Verdana"
    textAlign="center"/>
14.          </mx:VBox>
15.       </mx:VBox>
16.       <mx:VBox >
17.          <mx:Label text="Adresse"/>
18.          <mx:VBox backgroundColor="#EECA05">
19.          <mx:Text id="lblAdresse" minWidth="250" minHeight="20" fontFamily="Verdana"
    textAlign="center"/>
20.          </mx:VBox>
21.       </mx:VBox>
22.       </mx:HBox>
23.       <mx:HBox>
24.       <mx:VBox >
25.          <mx:Label text="Ville"/>
26.          <mx:VBox backgroundColor="#EECA05">
27.          <mx:Text id="lblVille" minWidth="100" minHeight="20" fontFamily="Verdana"
    textAlign="center"/>
28.          </mx:VBox>
29.       </mx:VBox>
30.       <mx:VBox >
31.          <mx:Label text="Code Postal"/>
32.          <mx:VBox backgroundColor="#EECA05">
33.          <mx:Text id="lblCodePostal" minWidth="70" minHeight="20" fontFamily="Verdana"
    textAlign="center"/>
34.          </mx:VBox>
35.       </mx:VBox>
36.       <mx:VBox >
37.          <mx:Label text="Indice"/>
38.          <mx:VBox backgroundColor="#EECA05">
39.          <mx:Text id="lblIndice" minWidth="20" minHeight="20" fontFamily="Verdana"
    textAlign="center"/>
```

```
40.          </mx:VBox>
41.       </mx:VBox>
42.       </mx:HBox>
43.    </mx:VBox>
```

Le code se comprend de lui-même. Expliquons simplement le conteneur vertical affichant le nom de l'employé par exemple :

- lignes 4-9 : le conteneur vertical
- ligne 5 : le libellé *Nom*
- lignes 6-8 : un conteneur vertical qui affichera le nom de l'employé (ligne 7). Nous souhaitons donner une couleur de fond différente aux champs affichant des informations sur l'employé. Le composant *Text* n'offre pas cette possibilité (ou alors j'ai mal cherché). On peut fixer la couleur de fond d'un conteneur. C'est pourquoi il a été utilisé ici.
- ligne 7 : le composant *Text* qui affichera le nom de l'employé. On lui fixe une hauteur et une largeur minimales.

Nous allons utiliser le conteneur "*employe*" pour afficher les informations de l'employé que l'utilisateur sélectionne dans le combo des employés et cela indépendamment du bouton [Salaire] dont le rôle sera ultérieurement de calculer le salaire une fois que toutes les informations nécessaires auront été saisies.

Pour gérer le changement de sélection dans le combo "*employes*" son code MXML évolue comme suit :

```
<mx:ComboBox id="cmbEmployes" dataProvider="{employes}" labelFunction="displayEmploye"
change="displayInfosEmploye();"/>
```

L'événement **change** est diffusé par le combo lorsque l'utilisateur change sa sélection. Le gestionnaire de cet événement sera la méthode **displayInfosEmploye**.

Rappelons les méthodes exposées par le service web distant :

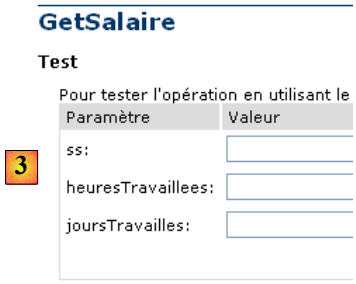
```
1.      // liste de toutes les identités des employés
2.      public Employe[] GetAllIdentitesEmployes();
3.      // ------- le calcul du salaire
4.  public FeuilleSalaire GetSalaire(string ss, double heuresTravaillees, int joursTravailles);
```

Nous voulons ici afficher les informations (nom, prénom, ...) de l'employé sélectionné dans le combo. Le service web n'expose pas de méthode pour les obtenir. Néanmoins, nous pouvons utiliser la méthode *GetSalaire* en passant le n° SS de l'employé sélectionné et 0 pour les heures et jours travaillés. Un calcul de salaire inutile sera fait mais la méthode *GetSalaire* nous retournera un objet de type *FeuilleSalaire* dans lequel nous trouverons les informations dont nous avons besoin.

La déclaration actuelle du service web est modifiée pour inclure la définition de la méthode *GetSalaire* :

```
1.  <mx:WebService id="pam"
2.        wsdl="http://localhost:1077/Service1.asmx?WSDL"
3.        fault="wsFault(event);"
4.        showBusyCursor="true">
5.        <mx:operation
6.           name="GetAllIdentitesEmployes"
7.           result="loadEmployesCompleted(event)"
8.           fault="loadEmployesFault(event);">
9.           <mx:request/>
10.       </mx:operation>
11.       <mx:operation name="GetSalaire"
12.          result="getSalaireCompleted(event)"
13.          fault="getSalaireFault(event);">
14.          <mx:request>
15.             <ss>{employes.getItemAt(cmbEmployes.selectedIndex).SS}</ss>
16.             <heuresTravaillees>{heuresTravaillees}</heuresTravaillees>
17.             <joursTravailles>{joursDeTravail}</joursTravailles>
18.          </mx:request>
19.       </mx:operation>
20. </mx:WebService>
```

- lignes 11-19 : la définition de la méthode *GetSalaire* du service web
- ligne 12 : définit la méthode à exécuter lorsque l'appel à la méthode *GetSalaire* réussit
- ligne 13 : définit la méthode à exécuter lorsque l'appel à la méthode *GetSalaire* échoue
- lignes 14-18 : la méthode *GetSalaire* attend trois paramètres. Ils sont définis à l'intérieur d'une balise <*mx:request*> sous la forme **<param1>valeur1</param1>**. L'identifiant **param1** ne peut être quelconque. Il faut utiliser les noms attendus par le service web :



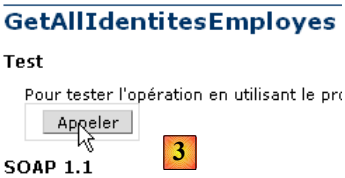
- en [1], la page du service web [http://localhost:1077/Service1.asmx]
- en [2], le lien vers la page de test de la méthode [GetSalaire]
- en [3], les paramètres attendus par la méthode. **Ce sont ces noms** qu'il faut utiliser comme balises enfants de la balise *<mx:request>*.

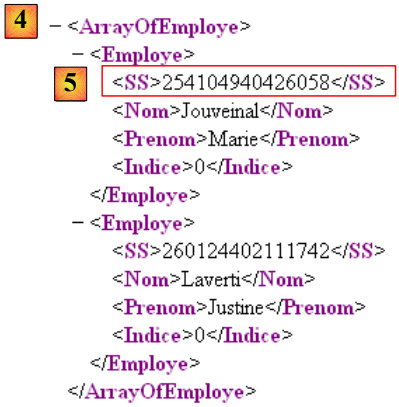
Revenons à la déclaration du service web :

```
1.       <mx:operation name="GetSalaire"
2.          result="getSalaireCompleted(event)"
3.          fault="getSalaireFault(event);">
4.          <mx:request>
5.             <ss>{employes.getItemAt(cmbEmployes.selectedIndex).SS}</ss>
6.             <heuresTravaillees>{heuresTravaillees}</heuresTravaillees>
7.             <joursTravailles>{joursDeTravail}</joursTravailles>
8.          </mx:request>
9.       </mx:operation>
```

- ligne 5 : le paramètre **ss**. On se rappelle qu'au démarrage de l'application Flex, le tableau de tous les employés a été stocké dans une variable **employes** de type *ArrayCollection*.
  - *employes.getItemAt(i)* : est l'employé n° i du tableau
  - *employes.getItemAt(i).**SS*** : est le n° de sécurité sociale de cet employé.
  - *cmbEmployes.selectedIndex* : est le n° de l'élément sélectionné dans le combo des employés *cmbemployes*.

Ci-dessus, comment sait-on que **SS** est le n° de sécurité sociale d'un employé ? Pour cela, il faut revenir à la réponse envoyée par la méthode *GetAllIdentitesEmployes* :





- en [1], page du service web [Service.asmx]
- en [2], le lien vers la page de test de la méthode [GetAllIdentitesEmployes]
- en [3], le test est fait. Aucun paramètre n'est attendu.
- en [4] : la réponse XML contient un tableau d'employés. C'est ce tableau qui a été mémorisé dans la variable *employes*. On voit en [5], que **SS** est bien la balise utilisée pour contenir le n° de sécurité sociale.

Terminons l'étude du service web :

```
1.       <mx:operation name="GetSalaire"
2.          result="getSalaireCompleted(event)"
3.          fault="getSalaireFault(event);">
4.          <mx:request>
5.             <ss>{employes.getItemAt(cmbEmployes.selectedIndex).SS}</ss>
6.             <heuresTravaillees>{heuresTravaillees}</heuresTravaillees>
7.             <joursTravailles>{joursDeTravail}</joursTravailles>
8.          </mx:request>
9.  </mx:operation>
```

- ligne 6 : le nombre d'heures travaillées sera fourni par une variable **heuresTravaillees**
- ligne 6 : le nombre de jours travaillés sera fourni par une variable **joursDeTravail**

Ces variables doivent être déclarées dans la balise *<mx::Script>* avec l'attribut **[Bindable]** qui leur permet d'être référencées par des composants MXML (lignes 7-10 ci-dessous).

```
1.     <mx:Script>
2.        <![CDATA[
3.  ...
4.           // données
5.           [Bindable]
6.           private var employes : ArrayCollection;
7.           [Bindable]
8.           private var heuresTravaillees:Number;
9.           [Bindable]
10.          private var joursDeTravail:int;
11. ...
12. </mx:Script>
```

Le code de gestion des événements de la vue évolue comme suit :

```
1.  <mx:Script>
2.        <![CDATA[
3.           import mx.rpc.events.FaultEvent;
4.           import mx.collections.ArrayCollection;
5.           import mx.rpc.events.ResultEvent;
6.
7.           // données
8.           [Bindable]
9.           private var employes : ArrayCollection;
10.          [Bindable]
11.          private var heuresTravaillees:Number;
12.          [Bindable]
13.          private var joursDeTravail:int;
14.
15.          private function init():void{
16.             // on note les hauteur / largeur de # blocs
17.             employeHeight=employe.height;
18.             employeWidth=employe.width;
19.             // on cache certains éléments
20.             hideEmploye();
21. ...
22.          }
23.
24.          private function displayInfosEmploye():void{
25.             // formulaire
26.             hideEmploye();
27.             // on calcule un salaire fictif
28.             heuresTravaillees=0;
29.             joursDeTravail=0;
30.             pam.GetSalaire.send();
31.          }
32.
33.          private function getSalaireCompleted(event:ResultEvent):void{
34.    ...
35.          }
36.
37.          private function getSalaireFault(event:FaultEvent):void{
38.    ...
39.          }
40.
41.       // vues partielles ------------------------------------------------
42.       private var employeHeight:int;
43.       private var employeWidth:int;
44.
```

```
45.      private function hideEmploye():void{
46.         employe.height=0;
47.         employe.width=0;
48.      }
49.
50.      private function showEmploye():void{
51.         employe.height=employeHeight;
52.         employe.width=employeWidth;
53.      }
54.      ]]>
55.   </mx:Script>
```

- ligne 15 : la méthode **init** exécutée au démarrage de l'application Flex mémorise les hauteur / largeur du conteneur vertical *employe*, ceci afin de pouvoir le restaurer (lignes 50-53) après l'avoir caché (lignes 45-48).
- ligne 24 : la méthode **displayInfosEmploye** est la méthode exécutée lorsque l'utilisateur change sa sélection dans le combo des employés.
- ligne 26 : le conteneur *employe* est caché s'il était visible
- ligne 30 : la méthode *GetSalaire* du service web est appelée de façon asynchrone. On sait qu'elle attend trois paramètres :

```
1.            <ss>{employes.getItemAt(cmbEmployes.selectedIndex).SS}</ss>
2.            <heuresTravaillees>{heuresTravaillees}</heuresTravaillees>
3.            <joursTravailles>{joursDeTravail}</joursTravailles>
```

  - ligne 1 : le paramètre **ss** sera le n° SS de l'employé sélectionné dans le combo des employés
  - ligne 2 : la méthode *displayInfosEmploye* affecte la valeur 0 à la variable **heuresTravaillees** (ligne 28)
  - ligne 3 : la méthode *displayInfosEmploye* affecte la valeur 0 à la variable **joursDeTravail** (ligne 29)

La méthode *GetSalaireCompleted* est exécutée si la méthode *GetSalaire* du service web se termine bien :

```
1.  private function getSalaireCompleted(event:ResultEvent):void{
2.            // on cache le msg d'erreur
3.            hideMsg();
4.            // on reçoit une feuille de salaire
5.            var feuilleSalaire:Object=event.result;
6.            // affichage
7.            lblNom.text=feuilleSalaire.Employe.Nom;
8.            lblPreNom.text=feuilleSalaire.Employe.Prenom;
9.            lblAdresse.text=feuilleSalaire.Employe.Adresse;
10.           lblVille.text=feuilleSalaire.Employe.Ville;
11.           lblCodePostal.text=feuilleSalaire.Employe.CodePostal;
12.           lblIndice.text=feuilleSalaire.Employe.Indice;
13.           showEmploye();
14.        }
```

- ligne 3 : on cache la zone de message au cas où elle serait affichée.
- ligne 5 : on récupère la feuille de salaire renvoyée par la méthode *GetSalaire*

Pour savoir ce que renvoie exactement la méthode *GetSalaire*, nous revenons sur la page du service web :

- en [1], la page du service web [Service.asmx]
- en [2], le lien qui mène à la page de test de la méthode [GetSalaire]
- en [3], on fournit des paramètres
- en [4], le résultat XML obtenu.

Revenons à la méthode *getSalaireCompleted* :

```
1.  private function getSalaireCompleted(event:ResultEvent):void{
2.              // on cache le msg d'erreur
3.              hideMsg();
4.              // on reçoit une feuille de salaire
5.              var feuilleSalaire:Object=event.result;
6.              // affichage
7.              lblNom.text=feuilleSalaire.Employe.Nom;
8.              lblPreNom.text=feuilleSalaire.Employe.Prenom;
9.              lblAdresse.text=feuilleSalaire.Employe.Adresse;
10.             lblVille.text=feuilleSalaire.Employe.Ville;
11.             lblCodePostal.text=feuilleSalaire.Employe.CodePostal;
12.             lblIndice.text=feuilleSalaire.Employe.Indemnites.Indice;
13.             showEmploye();
14. }
```

- ligne 5 : *feuilleSalaire=event.result* représente le flux XML [4] renvoyé par la méthode *GetSalaire*. D'après ce flux, on voit que :
  - *feuilleSalaire.Employe* est le flux XML d'un employé
  - *feuilleSalaire.Employe.Nom* est le nom de cet employé
  - ...
- lignes 7-12 : le flux XML *feuilleSalaire* est exploité pour remplir les différents champs du conteneur **employe**.
- ligne 13 : le conteneur **employe** est affiché.

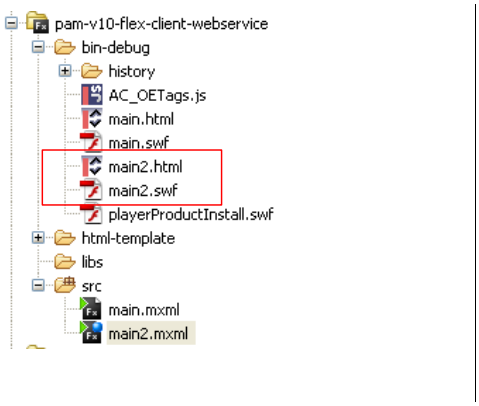
La méthode *getSalaireFault* est exécutée si la méthode *GetSalaire* du service web se termine mal :

```
1.          private function getSalaireFault(event:FaultEvent):void{
2.              // affichage msg d'erreur
3.              msg.text=event.fault.message;
4.              // formulaire
5.              showMsg();
```

```
6.          }
```

- ligne 3 : le message d'erreur *event.fault.message* est mis dans la zone de message
- ligne 5 : la zone de message est affichée

Ici s'arrêtent les modifications nécessaires à cette nouvelle version. Lorsqu'on la sauvegarde et si elle est syntaxiquement correcte, la version exécutable est générée dans le dossier [bin-debug] du projet :



Ci-dessus, [main2.html] est la page HTML qui embarque le binaire de l'application Flex [main2.swf] qui sera exécutée par Flash Player.

Nous pouvons tester cette nouvelle version :

- le service web Asp.net doit être lancé
- le serveur Apache doit être lancé pour le client Flex

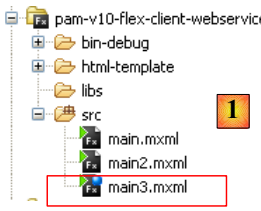
Si on suppose que l'alias [pam-v10-flex-client-webservice] utilisé dans la précédente version existe toujours, on demande au serveur Apache l'url [http://localhost/pam-v10-flex-client-webservice/main2.html] dans un navigateur :

- en [1], l'Url demandée
- en [2], le combo des employés
- en [3], on change la sélection dans le combo pour provoquer l'événement *change*
- en [4], le résultat obtenu : la fiche de *Justine Laverti*.

14.5 La vue n° 3

La vue n° 3 amène le contrôle de validité du formulaire. Ici, seul le champ de saisie "*txtHeuresTravaillees*" est vérifié. Tant que le formulaire est incorrect, le bouton "*btnSalaire*" restera inhibé.

Pour l'ajout de cette fonctionnalité, nous dupliquons [main2.mxml] dans [main3.mxml] :



Désormais nous travaillerons avec [main3.mxml] dont on fera l'application par défaut (voir ce concept page 216). Tout d'abord, nous ajoutons un attribut au composant "*txtHeuresTravaillees*" :

```
<mx:TextInput id="txtHeuresTravaillees" change="validateForm(event)"/>
```

A chaque fois que le contenu du champ de saisie "*txtHeuresTravaillees*" change, la méthode *validateForm* est appelée. C'est une méthode locale écrite par le développeur. Dans celle-ci, nous pourrions vérifier que le contenu du champ de saisie "*txtHeuresTravaillees*" est bien un nombre entier positif. Nous allons procéder autrement en utilisant un composant de validation :

```
1.      <mx:NumberValidator id="heuresTravailleesValidator" source="{txtHeuresTravaillees}"
    property="text"
2.      precision="2" allowNegative="false"
3.      invalidCharError="Caractères invalides"
4.      precisionError="Deux chiffres au plus après la virgule"
5.      negativeError="Le nombre d'heures doit être positif ou nul"
6.      invalidFormatCharsError="Format invalide"
7.      required="true"
8.  requiredFieldError="Donnée requise"/>
```

- ligne 1 : le composant <*mx:NumberValidator*> permet de vérifier qu'un autre composant contient un nombre entier ou réel.
- ligne 1 : l'attribut *id* donne un identifiant au composant.
- ligne 1 : *source* est l'id du composant vérifié par le composant *NumberValidator*. Ici, c'est le champ de saisie "*txtHeuresTravaillees*" qui est vérifié.
- ligne 1 : *property* est le nom de la propriété du composant *source* qui contient la valeur à vérifier. Au final, c'est la valeur *source.property* qui est vérifiée, ici *txtHeuresTravaillees.text*.
- ligne 2 : *precision* fixe le nombre maximal de décimales autorisées. *precision=0* revient à vérifier que le nombre saisi est entier.
- ligne 2 : *allowNegative* indique si les nombres négatifs sont autorisés ou non
- ligne 7 : *required* indique si la saisie est obligatoire ou non.

Lorsqu'une condition de validation n'est pas vérifiée, un message d'erreur est affiché dans une bulle près du composant erroné. Par défaut, ces messages sont en anglais. Il est possible de définir soi-même ces messages :

invalidCharError : le message d'erreur lorsque le texte contient un caractère qu'on ne peut rencontrer dans un nombre
precisionError : le message d'erreur lorsque le nombre de décimales est incorrect vis à vis de l'attribut *precision*
negativeError : le message d'erreur lorsque le nombre est négatif alors qu'on a l'attribut *allowNegative*="false"
requiredFieldError : le message d'erreur lorsqu'il n'y a pas eu de saisie alors qu'on a l'attribut *requiredField*="true"
invalidFormatCharsError : le message d'erreur lorsque le texte a des caractères ou un format invalide ?

Revenons au composant "*txtHeuresTravaillees*" :

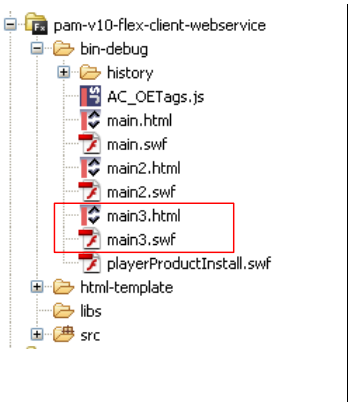
```
<mx:TextInput id="txtHeuresTravaillees" change="validateForm(event)"/>
```

La méthode *validateForm* pourrait être la suivante dans la balise <*mx::Script*> :

```
1.        private function validateForm(event:Event):void
2.        {
3.            // on valide les heures travaillées
4.            var evt:ValidationResultEvent = heuresTravailleesValidator.validate();
5.            // validation réussie ?
6.            btnSalaire.enabled=evt.type==ValidationResultEvent.VALID;
7.    }
```

- ligne 4 : le validateur "*heuresTravailleesValidator*" est exécuté. Il rend un résultat de type *ValidationResultEvent*.
- ligne 6 : *evt.type* est de type *String* et indique le type de l'événement. *evt.type* a deux valeurs possibles pour le type *ValidationResultEvent,* "invalid" ou "valid" représentées par les constantes *ValidationResultEvent.INVALID* et *ValidationResultEvent.VALID*. Si ligne 4, la validation a été réussie, *evt.type* doit avoir pour valeur *ValidationResultEvent.VALID*. Dans ce cas, le bouton *btnSalaire* est activé sinon il est inhibé.

Ceci est suffisant pour tester la validité des heures travaillées.

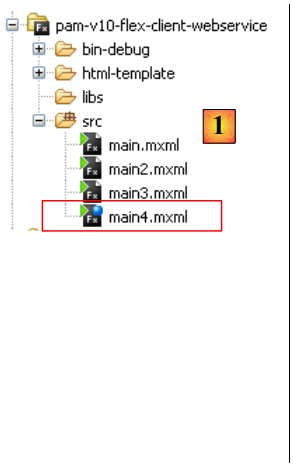


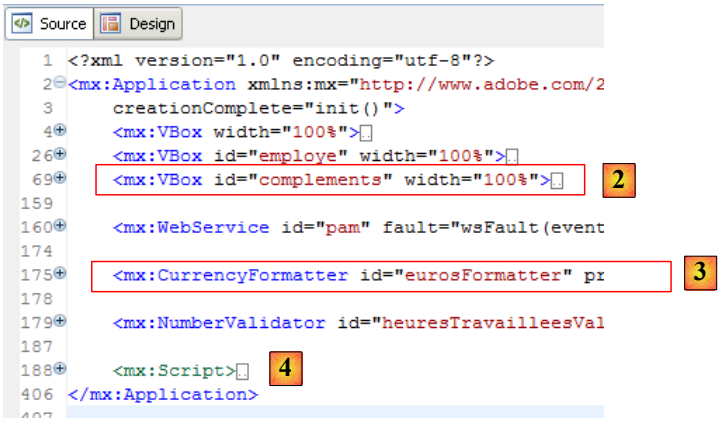
Ci-dessus, la compilation du projet a produit les fichiers [main3.html] et [main3.swf]. Nous demandons l'Url [http://localhost/pam-v10-flex-client-webservice/main3.html] dans un navigateur et nous vérifions divers cas d'erreur :

- un champ erroné a une bordure rouge [1, 2, 3], un champ correct une bordure bleue [4].
- en [4], on notera que le bouton [Salaire] est actif parce que le nombre d'heures travaillées est correct.

14.6 La vue n° 4

La vue n° 4 termine le formulaire de calcul du salaire. Pour cela, nous dupliquons [main3.xml] dans [main4.xml] et nous travaillons désormais avec *main4* dont on fait l'application par défaut (cf page 216).

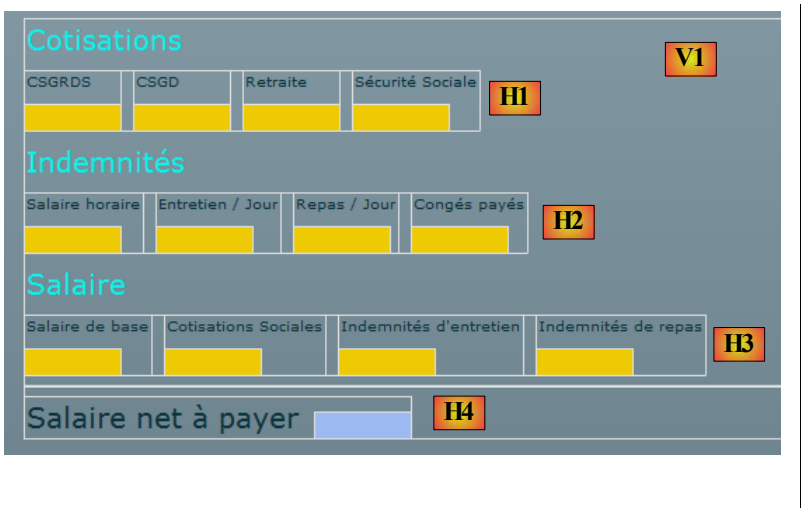




Les changements apportés dans [main4.xml] [1] sont les suivants :

- un nouveau conteneur vertical est ajouté à la vue [2] afin d'afficher les éléments du salaire de l'employé
- un composant permettant de formater les valeurs monétaires est ajouté [3]
- l'affichage des éléments du salaire est pris en charge par le gestionnaire associé à l'événement "clic" du bouton "*btnSalaire*".

La vue évolue comme suit :



Le nouveau conteneur suit le principe du précédent. C'est un conteneur vertical *VBox* [V1] contenant quatre conteneurs horizontaux *HBox* [Hi]. Les conteneurs horizontaux H1 à H3 sont formés de conteneurs verticaux contenant deux libellés dont le deuxième est lui-même dans un conteneur vertical pour disposer d'une couleur de fond.

Question 1 : écrire le conteneur du salaire. Il sera appelé par la suite *complements*.

Question 2 : écrire les méthodes permettant de cacher / montrer le conteneur *complements*. On s'inspirera de ce qui a été fait précédemment pour le conteneur *employe*.

On associe un gestionnaire à l'événement "click" du bouton "btnSalaire" :

```
<mx:Button id="btnSalaire" label="Salaire" click="calculerSalaire()"/>
```

La méthode *calculerSalaire* est la suivante :

```
1.          private function calculerSalaire():void{
2.             // préparation formulaire
3.             affichageSalaire=true;
```

```
4.              msg.text="";
5.              // paramètres du calcul du salaire
6.              heuresTravaillees=Number(txtHeuresTravaillees.text);
7.              joursDeTravail=int(joursTravailles.value);
8.              // le salaire est demandé au service web
9.              pam.GetSalaire.send();
10. }
```

- ligne 3 : le booléen *affichageSalaire* sert à indiquer s'il faut afficher ou non le conteneur *complements* qui affiche les éléments du salaire. La méthode *getSalaireCompleted* est exécutée sur deux événements :
  - le changement d'employé dans le combo des employés pour afficher ses informations sans le salaire. On mettra alors *affichageSalaire*=false.
  - le calcul du salaire
- ligne 6 : le texte du champ de saisie *txtHeuresTravaillees* est transformé en nombre réel.
- ligne 7 : la valeur de l'incrémenteur *joursTravailles* est transformé en nombre entier.
- ligne 9 : appel de la méthode distante *GetSalaire*. On rappelle que cette méthode attend trois paramètres dont les paramètres *heuresTravaillees* et *joursDeTravail* initialisés lignes 6 et 7. On rappelle également que si l'appel asynchrone de la méthode *GetSalaire* :
  - réussit, la méthode *getSalaireCompleted* sera appelée
  - échoue, la méthode *getSalaireFault* sera appelée

Question 3 : compléter la méthode actuelle *getSalaireCompleted* pour qu'elle affiche le salaire de l'employé si le bouton *btnSalaire* a été cliqué.

Actuellement, l'affichage des éléments du salaire se fait sans le signe des euros. On peut inclure celui-ci dans le code ou bien utiliser un formateur. C'est ce qui est proposé maintenant. Le formateur sera le suivant :

```
1.    <mx:CurrencyFormatter id="eurosFormatter" precision="2"
2.          currencySymbol="€" useNegativeSign="true"
3.  alignSymbol="right"/>
```

- ligne 1 : **id** est l'identifiant du formateur, **precision** le nombre de décimales à garder.
- ligne 2 : **currencySymbol** est le symbole monétaire à utiliser. **useNegativeSign** indique s'il faut utiliser ou non le signe - pour les valeurs négatives.
- ligne 3 : **alignSymbol** indique où placer le signe monétaire par rapport au nombre.

Ce formateur s'utilise dans le code du script de la façon suivante :

```
            lblSH.text=eurosFormatter.format(feuilleSalaire.Indemnites.BaseHeure);
```

- *eurosFormatter* est l'id du formateur à utiliser
- *format* est la méthode à appeler pour formater un nombre. Elle rend une chaîne de caractères.
- *feuilleSalaire.Indemnites.BaseHeure* est ici le nombre à formater.
- *lblSH* est le nom d'un composant de type Text.

Question 4 : modifier la méthode *getSalaireCompleted* pour qu'elle utilise le formateur monétaire.

Table des matières