ASP.NET MVC face à WebForms

Livre Blanc

Présentation, critères de choix

Jean-Philippe LECLERE
Clever Age

version 1.0
14/05/08

Sommaire

1 Introduction

Microsoft est sur le point intégrer un modèle MVC (Model-Vue-Controler) à son framework Web ASP.NET.

En effet, ASP.NET MVC RC1 est disponible depuis le 27 janvier 2009 et la sortie de la version définitive est prévue d'ici quelques semaines. Chose importante il s'agit d'un projet open source dont 5 versions beta ont fait l'objet de retour de la communauté.

ASP.NET est aujourd'hui une technologie mature, à la fois robuste et flexible qui permet de construire toutes sortes d'applications. Web Forms, la couche de présentation pour ASP.NET, est basée sur un modèle évènementiel qui a fait son succès. En incluant MVC dans ASP.NET, Microsoft ne souhaite pas remplacer Web Forms mais proposer une alternative pour répondre à l'ensemble des problématiques.

Il est vrai qu'il existe des adaptations de l'incontournable modèle MVC sur tous les langages et que des versions open source pour .NET ont vu le jour pour combler son absence. Microsoft se devait de réagir, c'est chose faite.

Mais quels sont les avantages et inconvénients de MVC.NET et Web Form et quels sont les critères qui permettent de choisir ?

2 Webforms

2.1 Principes

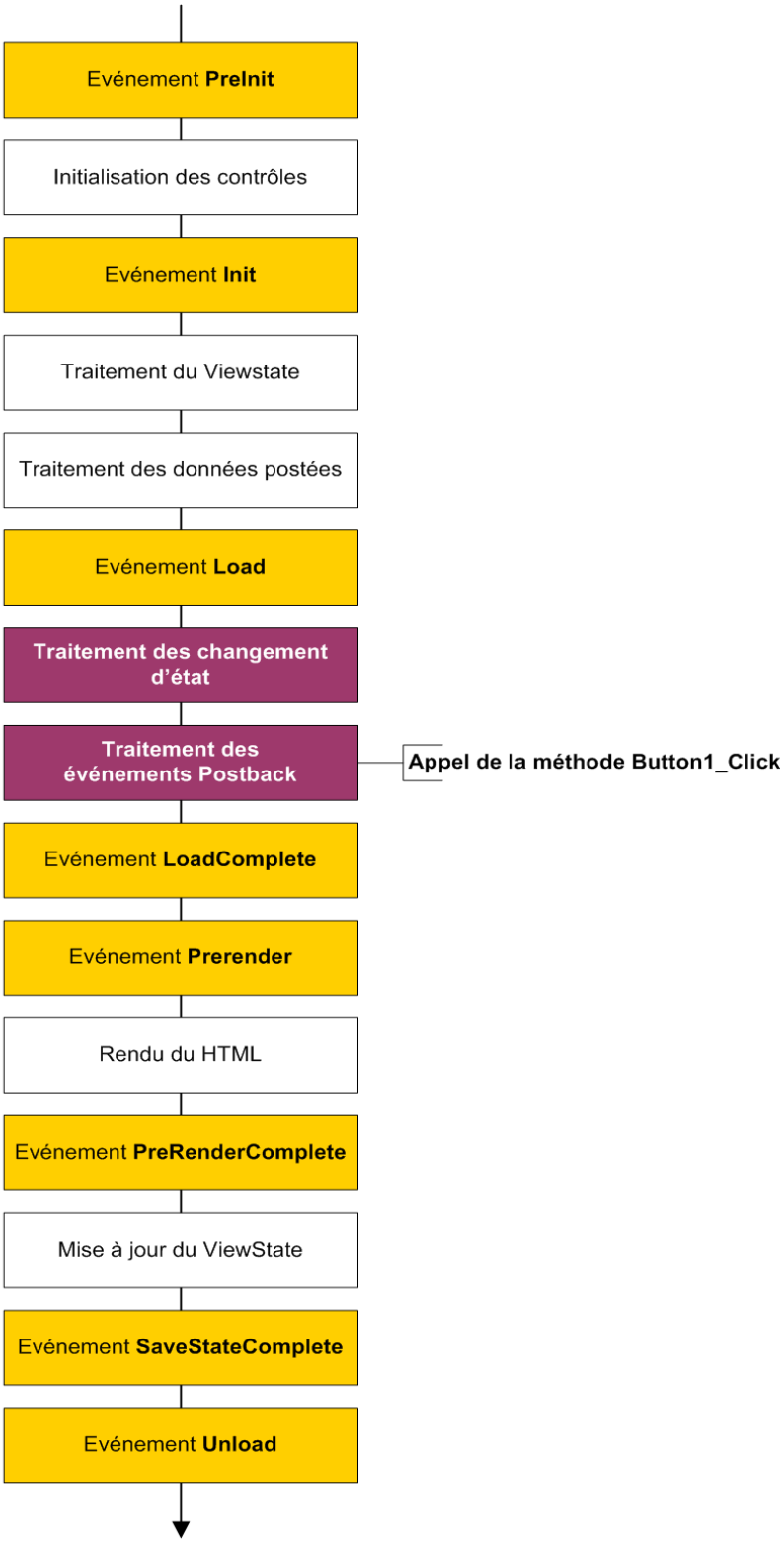
La couche présentation est composée des pages .aspx qui contiennent les éléments qui interagissent avec l'utilisateur appelés « contrôles ». Les pages sont associées chacune à un fichier « codebehind » qui contient le contrôleur de la page. Il s'agit d'une classe dérivant de la classe System.Web.UI.Page dont les méthodes prennent en charge les événements en provenance de la page. Le cycle de vie standard ASP.NET se charge quant a lui de produire le HTML pour le navigateur.

2.2 Modèle événementiel

En ASP.NET il y a de nombreuses étapes qui séparent l'action utilisateur de la nouvelle page affichée. Prenons l'exemple d'un bouton avec un événement « Click ». Nous aurons donc la méthode suivante dans le codebehind :

```
void Button1_Click(object sender, EventArgs e)
{
      // Exécute le code associé à l'action.
}
```

Avant que le code de la méthode Button1_Click soit exécuté, le runtime ASP.NET crée une instance de la classe Page qui associe le comportement attendu à l'URL demandée. La classe Page suit un cycle constitué des étapes du diagramme ci-dessous.



Cycle de vie d'une page ASP.NET (source msdn.microsoft.com)

Chacun des contrôles de la page possède son propre cycle qui prend part à celui de la page (par exemple, les événements *Init* et *Load* des contrôles sont déclenchés suite aux événements correspondant de la page).

Dans le diagramme il est question de *ViewState* et *PostBack*. Ce sont deux mécanismes qui permettent à ASP.NET de proposer un modèle événementiel à la sauce Windows Form

en contournant la nature sans état du Web. *ViewState* stocke l'ensemble des propriétés des contrôles de la page comme présentées au moment de l'affichage avant que l'utilisateur ne modifie quoi que ce soit. Le contenu du *ViewState* est transmis avec les données que la page s'envoie à elle-même lors de la validation. *PostBack* abstrait la communication client/serveur et génère notamment le code Javascript client pour les évènements souscrits.

2.3 Complexité du modèle

Pour pouvoir réaliser des interfaces complexes avec de multiples interactions utilisateur, il est crucial de maîtriser la survenue et l'ordre de l'ensemble des événements liés à la page et à ses contrôles.

L'abstraction procurée par les contrôles, le *data binding*, le *viewstate* et le cycle de la page complique souvent la tâche obligeant par exemple l'écriture d'un grand nombre de gestionnaires d'évènement pour le paramétrage des contrôles. Signalons également que la conception de contrôles customisés est une tâche qui nécessite la maîtrise du modèle événementiel et du designer de Visual Studio.

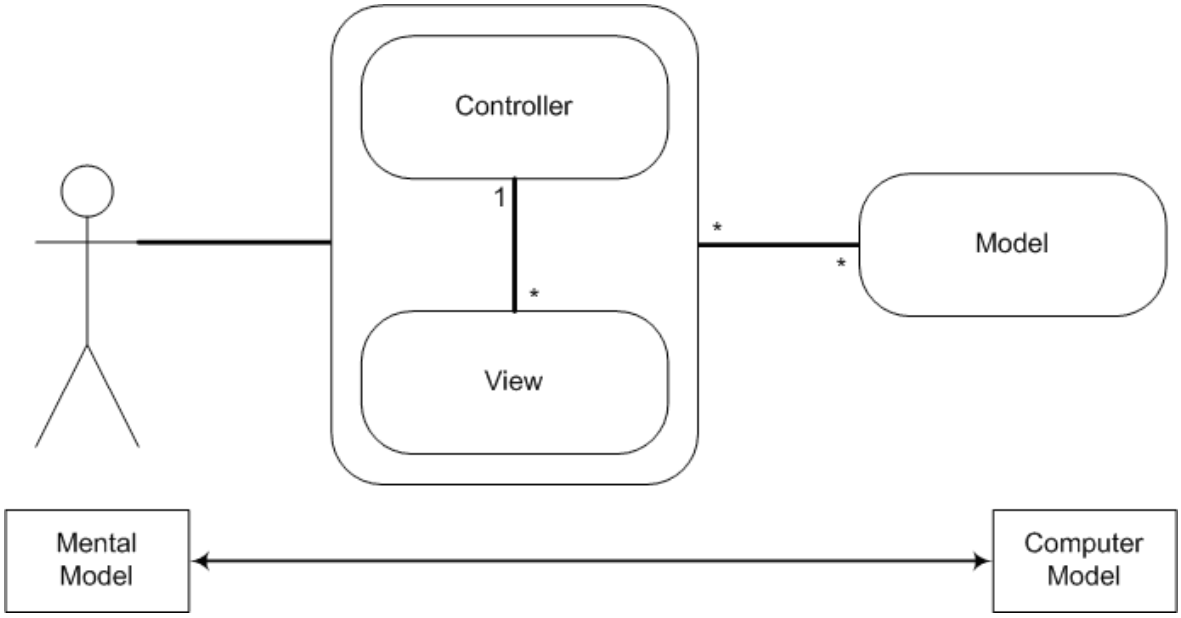
Le fait que l'approche Web Forms est centrée sur l'interface utilisateur génère aussi très vite de la complexité. Le problème est qu'il y a de l'intelligence dans la vue. En conséquence la mise en place de tests unitaires dans une application ASP.NET à base de Web Form n'est pas aisée. Il est possible de tester la couche métier, mais pas d'automatiser les appels. On peut mémoriser les requêtes http pour les rejouer ou utiliser des solutions simulant l'utilisation de l'interface mais dès qu'un contrôle dans la page sera modifié il faudra régénérer les tests pour la page !

3 ASP.NET MVC

3.1 Le pattern MVC

Origine

L'approche Modèle-Vue-Contrôleur a émergée en 1978 et a été décrite pour la 1ère fois en 1979 dans un projet Smalltalk par Trygve Reenskaug : « Le but principal est de combler l'écart entre la représentation humaine du modèle et le modèle digital dans l'ordinateur ».



Le modèle MVC tel que décrit par T. Reenskaug

Principes

En MVC, l'utilisateur interagit avec les vues et les actions sont capturée par les vues et envoyées au contrôleur, lequel décide de ce qu'il y a à faire et le met en œuvre en interaction avec le modèle. Le modèle est constitué essentiellement de la couche métier mais il a aussi en charge de notifier aux vues les changements qui peuvent nécessiter une mise à jour de l'interface utilisateur (UI).

MVC pour le Web

Model2 est une évolution de MVC pour le Web dans laquelle les vues sont exposées à l'utilisateur via le navigateur. Les actions sont capturées par le navigateur et transformées en requêtes http. Un contrôleur frontal au sein du serveur Web coordonne l'ensemble des requêtes envoyées à l'application.

3.2 Principes

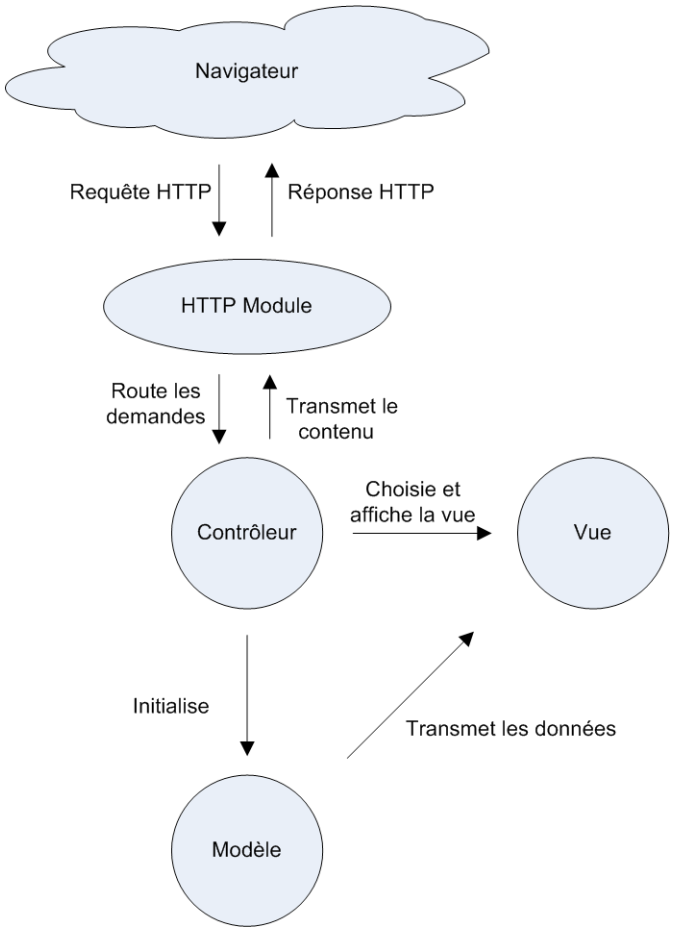


Diagramme de flux de données MVC

Le module de routage d'URL capture les requêtes et se charge de trouver le contrôleur et la méthode appropriés à l'action demandée. Lorsque le résultat de la méthode d'action est retourné, le contrôleur passe les données à la vue qui les met en forme. En bout de chaîne, la sortie de la vue est transmise au navigateur. Avec cette approche, il n'y a plus d'appel à la couche métier dans la vue.

3.3 Mise en œuvre

La mise en œuvre est classique du point de vue MVC. Une application MVC.NET contient logiquement 3 dossiers particuliers : Controllers, Models et Views.

Le paramétrage des routes s'effectue dans le fichier d'application global.asax via la méthode *MapRoute* qui fait l'association entre les URLs et les contrôleurs et leurs méthodes. Une route par défaut est proposée et permet de gérer les URLs de la forme monURL/<monControleur>/<monAction>/<monParametre>

La couche de routage

Concernant le mécanisme de routage, les actions sont mappées aux méthodes des contrôleurs par le framework. Ce mapping est flexible. Il est donc possible d'avoir des contrôleurs et des actions nommées différemment de ce que l'on trouvera dans l'URL. La seule contrainte imposée par le framework est d'ajouter le suffixe « Controller » au nom de la classe des contrôleurs par rapport à ce qui est déclaré dans la route et de placer les

vues associées à un contrôleur dans un sous-dossier du même nom.

```
public static void RegisterRoutes(RouteCollection routes)
{
   routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

   routes.MapRoute(
     "Default",                          // Route name
     "{controller}/{action}/{id}",       // URL with parameters
     new { controller = "Home", action = "Index", id = "" } // Parameter
defaults
   );
}
```

Enregistrement des routes dans le global.asax

La couche contrôleur

Au niveau des contrôleurs, la seule contrainte est qu'ils doivent implémenter l'interface *IController* qui a une seule méthode *Execute*. La manière la plus simple de créer un contrôleur est donc de créer une classe qui hérite de la classe *Controller* mais comme MVC est extensible on aurait pu créer notre propre type de contrôleur et utiliser la classe *ControllerFactory* pour générer des instances de ce nouveau contrôleur.

```
public class ProductsController : Controller
{
   NorthwindDataContext db = new NorthwindDataContext();

   //
   // GET: /Products/Category/1
  public ActionResult Category(int id)
   {
        Category category = db.GetCategoryById(id);

        return View("List", category);
   }
```

Exemple de contrôleur et de méthode d'action

Les méthodes d'action retournent un objet de type *ActionResult* qui sert de conteneur. Il y a 3 types d'ActionResult :

- ContentResult permet de retourner des chaînes de caractères,
- FileResult des fichiers,
- ViewResult des vues

La couche modèle

Comme on peut le voir dans l'extrait de code précédent, le contrôleur s'adresse au modèle pour effectuer les opérations demandées sur les données (dans ce cas simplement retourner les données). La couche modèle rend donc un service à la couche contrôleur.

La couche vue

```
<%@ Page Language="C#"
         MasterPageFile="~/Views/Shared/Site.Master"
         Inherits="System.Web.Mvc.ViewPage<DB.Northwind.Category>" %>

 <ul>
<%
   foreach (var product in ViewData.Model.Products) {
%>
     <li>
       <%=product.ProductName %>

       <span class="editlink">
          (<%=Html.ActionLink("Edit", "Edit", new{id=product.ProductID})%>)
       </span>

     </li>
<%
   }
%>
 </ul>
```

Exemple de vue affichant une liste de produits

La transmission des données du contrôleur à la vue est réalisée par l'objet ViewData ou ViewData<T> qui permet de typer la vue. Le type choisi est transmis à la vue via le paramètre *Inherits* de la la balise *Page*.

La génération du HTML se fait à la manière « classique » en insérant le code entre balises, c'est à dire façon ASP, JSP, PHP...

Le framework inclut une série de helpers HTML qui permettent le rendu des éléments HTML standard. Ces helpers remplacent les contrôles WebForm de manière plus légère puis qu'il n'y a pas de gestion des évènements. Les éléments pris en charge sont les ancres (voir dans l'exemple ci-dessus la génération d'une ancre pointant sur une méthode d'action avec la méthode *ActionLink*), les listes, les tableaux, les formulaires et les éléments qui les composent.

Pour l'affichage des données en mode tableau il est toujours possibles d'utiliser les GridView, DataList, et autre Repeater, mais cela n'est pas conseillé car on ne dispose pas des viewstate et postback et que l'approche évènementielle n'est plus de mise. Ce qui pourrait être un manque est en passe d'être comblé par la communauté open source. Citons le projet MVC Contrib (http://www.codeplex.com/MVCContrib) qui propose entre autres choses un datagrid.

Concernant le moteur de template, précisons qu'il est tout à fait possible d'en utiliser un autre. Par exemple, NVelocity est le portage de Velocity pour Java.

3.4 Bilan

Après ce rapide tour des couches de cette RC1, nous soulignons la facilité avec laquelle il est possible d'étendre le framework MVC fourni par Microsoft. L'ensemble des composants a été conçu pour être remplacé ou customisé. WebForms peut aussi être étendu mais l'approche événementielle complexe à de quoi rebuter.

Les avantages de MVC par rapport à WebForms:

- MVC donne la main sur le protocole HTTP et sur le HTML produit.
- Les pages HTML sont beaucoup moins lourdes.
- Les URLs sont testables.
- Les développements concurrents sont facilités

D'autres avantages sont inhérents au modèle MVC. La possibilité d'adopter une méthode de développement centrée sur les tests unitaires. Ceux-ci sont naturellement mis en place en automatisant les appels aux différents contrôleurs. De plus, il est possible d'effectuer ces tests sans avoir à exécuter les contrôleurs dans le process ASP.NET ce qui permet d'aller très vite. Les tests unitaires sont intégrés dans l'environnement de développement et plusieurs frameworks de test sont déjà disponibles.

La séparation claire des couche facilite l'intégration avec des librairies clientes Javascript telles que jQuery, JSON, ExtJs ou des librairies de type Flash, Silverlight. D'ailleurs, il est déjà possible de trouver des articles en la matière sur internet. Citons le très bon exemple d'application à page unique de Nicolas Moyère (disponible ici) à base de framework ASP.NET MVC et jQuery.

4 Choisir entre ASP.NET MVC et WebForms

WebForms et MVC sont 2 couches de présentation ayant chacune ses avantages et inconvénients.

Si WebForms permet de produire rapidement des interfaces de gestion de formulaire grâce à aux composants serveur manipulables en glisser-déposer, son modèle événementiel piloté par la vue complique la conception d'application avec une architecture en couches puisqu'il est souvent est obligatoire de mettre de l'intelligence dans la vue.

Au contraire, MVC agit comme un cadre en assurant la séparation des rôles des couches de l'application. La contrepartie est peut-être une vitesse de développement inférieure, bien que cela dépende du type d'interface à produire. Par exemple, le développement de pages de backoffice comportant de nombreuses interactions peut être laborieux en WebForms. A tout le moins, il nécessite de maitriser le cycle de vie ASP.NET décrit plus haut. Dans ce cas précis une vue MVC couplée à des requêtes Ajax permet de répondre à toutes les exigences.

Au final, le choix du modèle pour un projet s'opérera en fonction de multiples critères :

- la nature et la taille du projet,
- le type d'architecture applicative
- le niveau d'expérience et le background des développeurs.

Les projets avec un fort besoin d'intégrabilité (par exemple le projet de blog engine Oxite sur codeplex.com) ou avec des contraintes d'accessibilité nécessitant le rendu du site à travers plusieurs vues bénéficieront de la séparation des couches du modèle MVC.

Pour des projets de taille importante bâtis sur une architecture orientée service ou dont la méthodologie est centrée sur les test (TDD), l'architecture MVC est idéale.

En ce qui concerne l'équipe de développement, pour des développeurs junior MVC présente l'avantage de séparer de manière claire les couches. En conséquence, le code produit sera naturellement mieux architecturé et l'application facilement maintenable.

Pour des développeurs expérimentés, ayant une connaissance du modèle MVC mais ne connaissant pas .NET (univers Java, Ruby ou Symphony sous PHP), il sera plus rentable de développer à base de modèle MVC. Non seulement, le temps d'apprentissage des composants WebForms n'est pas négligeable, mais il sera plus intéressant qu'ils mettent à profit leur expérience MVC et montent en compétence sur les autres couches du framework .NET.

5 Bibliographie

- Introducing the ASP.NET MVC http://www.coderjournal.com/2008/12/introducing-aspnet-mvc-part-1-model-view-controller
- Building Web Apps without Web Forms http://msdn.microsoft.com/en-us/magazine/cc337884.aspx
- How does ASP.NET MVC work http://blogs.charteris.com/blogs/gopalk/archive/2009/01/20/how-does-asp-net-mvc-work.aspx
- Utilisation de jQuery avec ASP.NET MVC http://techheadbrothers.com/Articles.aspx/utilisation-jquery-aspnet-mvc-page-1