Construction d'une application windows MVC à trois couches avec Spring, M2VC-win et VB.NET

serge.tahe@istia.univ-angers.fr, juin 2005

1 Introduction

Nous poursuivons ici les articles :

- [Construction d'une application web à trois couches avec Spring et VB.NET - Partie 1] disponible à l'url [http://tahe.developpez.com/dotnet/web3tier-part1/]. Nous le nommerons par la suite [article1].
- [Construction d'une application web à trois couches avec Spring et VB.NET. - Partie 2] disponible à l'url [http://tahe.developpez.com/dotnet/web3tier-part2/]. Nous le nommerons par la suite [article2].

Rappelons que ces deux articles présentaient une application simplifiée d'achats de produits sur le web et que celle-ci était un simple prétexte pour étudier un exemple d'architecture web à trois couches, couches intégrées et configurées avec la version .NET de **Spring**.

Nous commencerons par rappeler ce qui a été fait et notamment l'architecture à trois couches [web, domain, dao] utilisée. Puis nous remplaçerons celle-ci par l'architecture [win,domain, dao] suivante :

- [dao] : la couche implémentée par la version [sqlMap] de l'article 2
- [domain] : la couche implémentée dans l'article 1 avec une légère variante
- [win] : une couche implémentée par une interface windows. Nous supposons ici que l'application web initiale est devenue une application windows classique. Pour implémenter la couche [win] nous utilisons le moteur [M2VC-win].

Outils utilisés :

- **Visual Studio.net** pour le développement
- **Spring** pour l'intégration et la configuration des couches de l'application web - voir l'annexe de la partie 1 de l'article à l'url [http://tahe.developpez.com/dotnet/web3tier-part1/]
- **Ibatis SqlMap** pour la couche d'accès aux données du SGBD - voir l'annexe de la partie 2 de l'article à l'url [http://tahe.developpez.com/dotnet/web3tier-part2/]
- le moteur **M2VC-win** - voir [http://tahe.developpez.com/dotnet/m2vc-win]
- le composant **LameGrid** disponible à l'url http://kikos31.developpez.com/lamegrid/

Dans une échelle [débutant-intermédiaire-avancé], ce document est dans la partie [avancé]. Sa compréhension nécessite divers pré-requis. Certains d'entre-eux peuvent être acquis dans des documents que j'ai écrits. Dans ce cas, je les cite. Il est bien évident que ce n'est qu'une suggestion et que le lecteur peut utiliser ses documents favoris.

- [article1] - déjà cité plus haut
- [article2] - déjà cité plus haut
- [M2VC-win, un moteur MVC pour des applications windows .NET] disponible à l'url [http://tahe.developpez.com/dotnet/m2vc-win]
- langage VB.net : [http://tahe.developpez.com/dotnet/vbnet/]
- utilisation de l'aspect IoC de Spring : [http://tahe.developpez.com/dotnet/springioc]
- documentation Ibatis SqlMap : [http://prdownloads.sourceforge.net/ibatisnet/DevGuide.pdf?download]
- documentation Spring.net : [http://www.springframework.net/documentation.html]

2 L'application [webarticles] - Rappels

Nous présentons ici les éléments de l'application web simplifiée de commerce électronique étudiée dans [article1] et [article2]. Celle-ci permet à des clients du web :

- de consulter une liste d'articles provenant d'une base de données
- d'en mettre certains dans un panier électronique
- de valider celui-ci. Cette validation a pour seul effet de mettre à jour, dans la base de données, les stocks des articles achetés.

2.1 Les vues de l'application

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

- la vue "articles en vente

- la vue [INFOS] qui donne des informations supplémentaires sur un produit :

- la vue [PANIER] qui donne le contenu du panier du client

- la vue [PANIERVIDE] pour le cas où le panier du client est vide

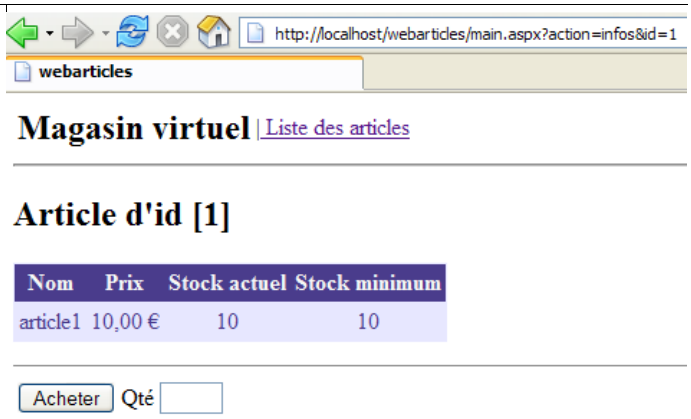
- la vue [ERREURS] qui signale toute erreur de l'application

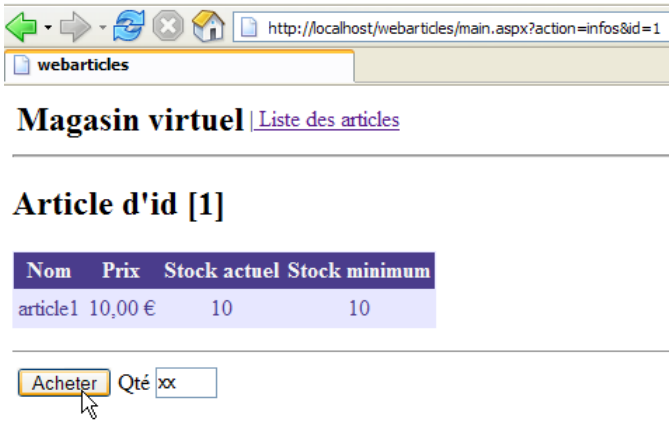
2.2 Fonctionnement de l'application [webarticles]

Nous présentons ci-dessous l'enchaînement des vues lors d'une utilisation typique de l'application :

A partir de la vue ci-dessus, nous utilisons les liens du menu pour faire des opérations. En voici quelques unes. La colonne de gauche représente la demande du client et la colonne de droite la réponse qui lui est faite.


http://localhost/webarticles/main.aspx
webarticles
Magasin virtuel | Voir le panier
Liste des articles
Nom Prix
article1 10,00 € Infos
article2 20,00 € Infos
article3 30,00 € Infos
article4 40,00 € Infos

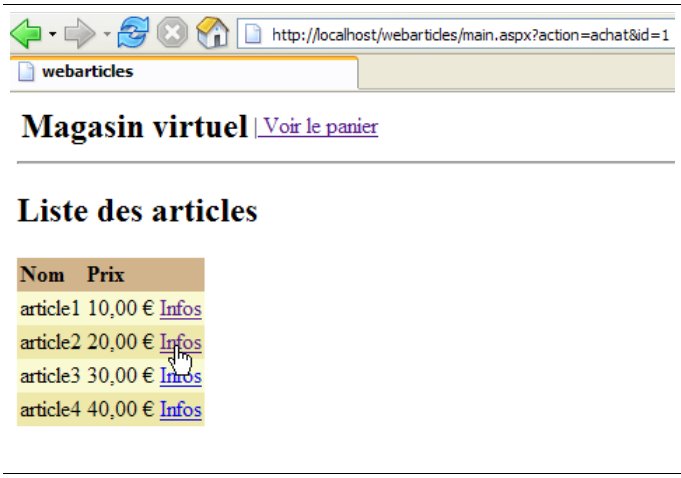

http://localhost/webarticles/main.aspx?action=infos&id=1
webarticles
Magasin virtuel | Liste des articles
Article d'id [1]
Nom Prix Stock actuel Stock minimum
article1 10,00 € 10 10
Acheter Qté


http://localhost/webarticles/main.aspx?action=infos&id=1
webarticles
Magasin virtuel | Liste des articles
Article d'id [1]
Nom Prix Stock actuel Stock minimum
article1 10,00 € 10 10
Acheter Qté xx


http://localhost/webarticles/main.aspx?action=achat&id=1
webarticles
Magasin virtuel | Liste des articles
Article d'id [1]
Nom Prix Stock actuel Stock minimum
article1 10,00 € 10 10
Acheter Qté xx Quantité incorrecte

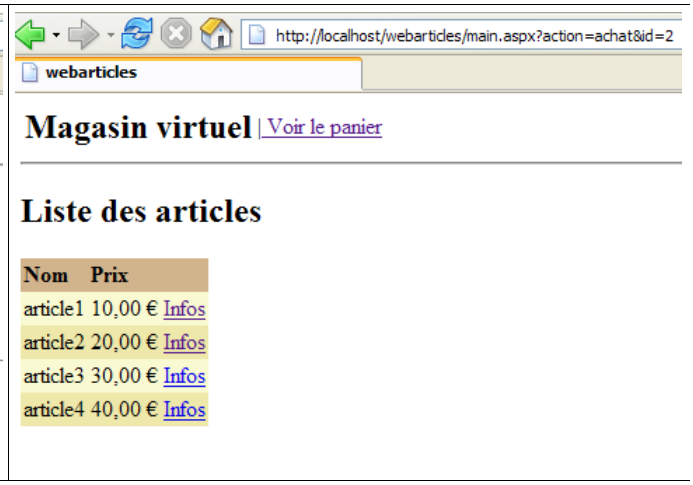

http://localhost/webarticles/main.aspx?action=achat&id=1
webarticles
Magasin virtuel | Liste des articles
Article d'id [1]
Nom Prix Stock actuel Stock minimum
article1 10,00 € 10 10
Acheter Qté 2 Quantité incorrecte

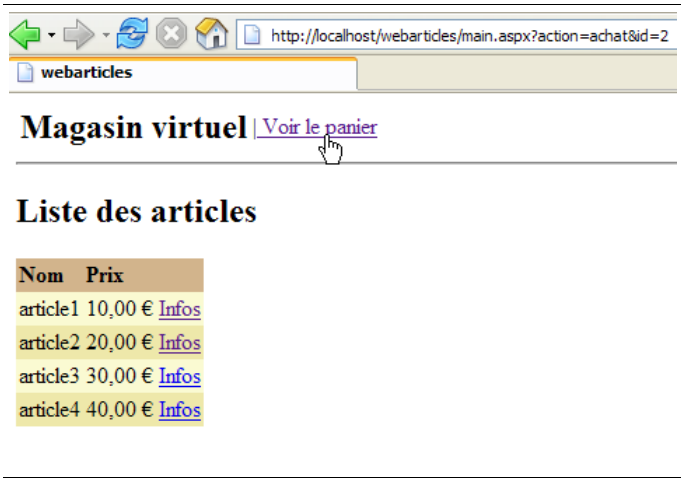

http://localhost/webarticles/main.aspx?action=achat&id=1
webarticles
Magasin virtuel | Voir le panier
Liste des articles
Nom Prix
article1 10,00 € Infos
article2 20,00 € Infos
article3 30,00 € Infos
article4 40,00 € Infos


http://localhost/webarticles/main.aspx?action=achat&id=1
webarticles
Magasin virtuel | Voir le panier
Liste des articles
Nom Prix
article1 10,00 € Infos
article2 20,00 € Infos
article3 30,00 € Infos
article4 40,00 € Infos


http://localhost/webarticles/main.aspx?action=infos&id=2
webarticles
Magasin virtuel | Liste des articles
Article d'id [2]
Nom Prix Stock actuel Stock minimum
article2 20,00 € 20 20
Acheter Qté

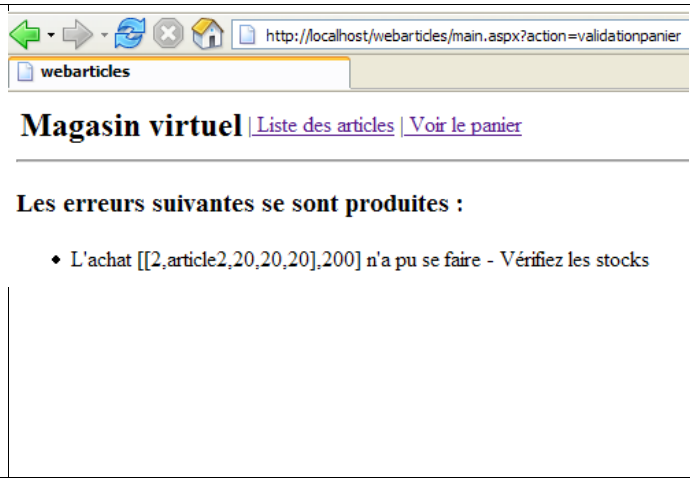

http://localhost/webarticles/main.aspx?action=infos&id=2
webarticles
Magasin virtuel | Liste des articles
Article d'id [2]
Nom Prix Stock actuel Stock minimum
article2 20,00 € 20 20
Acheter Qté 200


http://localhost/webarticles/main.aspx?action=achat&id=2
webarticles
Magasin virtuel | Voir le panier
Liste des articles
Nom Prix
article1 10,00 € Infos
article2 20,00 € Infos
article3 30,00 € Infos
article4 40,00 € Infos


http://localhost/webarticles/main.aspx?action=achat&id=2
webarticles
Magasin virtuel | Voir le panier
Liste des articles
Nom Prix
article1 10,00 € Infos
article2 20,00 € Infos
article3 30,00 € Infos
article4 40,00 € Infos

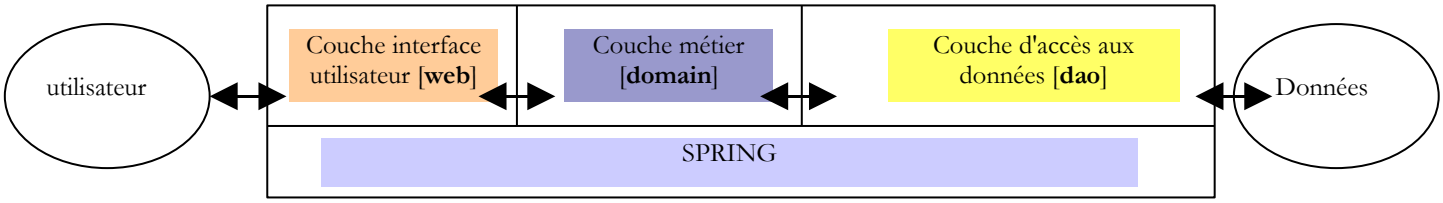

http://localhost/webarticles/main.aspx?action=panier
webarticles
Magasin virtuel | Liste des articles | Valider le panier
Contenu de votre panier
Article Qté Prix Total
article1 3 10 30,00 € Retirer
article2 200 20 4 000,00 € Retirer
Total de la commande : 4030 euros


http://localhost/webarticles/main.aspx?action=panier
webarticles
Magasin virtuel | Liste des articles | Valider le panier
Contenu de votre panier
Article Qté Prix Total
article1 3 10 30,00 € Retirer
article2 200 20 4 000,00 € Retirer
Total de la commande : 4030 euros


http://localhost/webarticles/main.aspx?action=validationpanier
webarticles
Magasin virtuel | Liste des articles | Voir le panier
Les erreurs suivantes se sont produites :
L'achat [[2,article2,20,20,20],200] n'a pu se faire - Vérifiez les stocks

2.3 Architecture générale de l'application

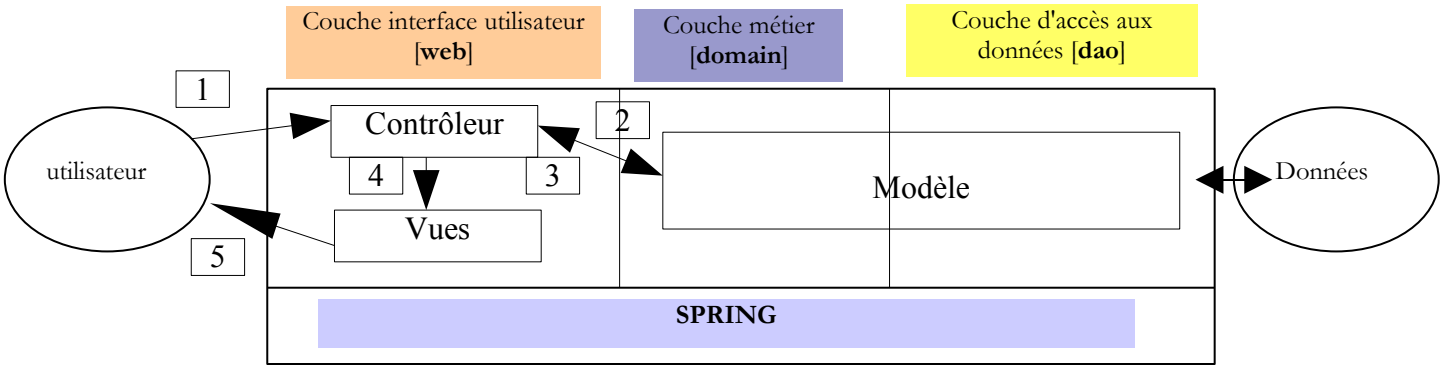
L'application web présente une architecture à trois couches :



- les trois couches ont été rendues indépendantes grâce à l'utilisation d'interfaces

- l'intégration des différentes couches a été réalisée avec **Spring**
- chaque couche fait l'objet d'espaces de noms séparés : **web** (couche UI), **domain** (couche métier) et **dao** (couche d'accès aux données).

L'application respecte une architecture MVC (Modèle - Vue - Contrôleur). Si nous reprenons le schéma en couches ci-dessus, l'architecture MVC s'y intègre de la façon suivante :



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

1. le client fait une demande au contrôleur. Ce contrôleur est ici une page .aspx à laquelle on fait jouer un rôle particulier. Elle voit passer toutes les demandes des clients. C'est la porte d'entrée de l'application. C'est le C de MVC.
2. le contrôleur traite cette demande. Pour ce faire, il peut avoir besoin de l'aide de la couche métier, ce qu'on appelle le modèle M dans la structure MVC.
3. le contrôleur reçoit une réponse de la couche métier. La demande du client a été traitée. Celle-ci peut appeler plusieurs réponses possibles. Un exemple classique est
   - une page d'erreurs si la demande n'a pu être traitée correctement
   - une page de confirmation sinon
4. le contrôleur choisit la réponse (= vue) à envoyer au client. Celle-ci est le plus souvent une page contenant des éléments dynamiques. Le contrôleur fournit ceux-ci à la vue.
5. la vue est envoyée au client. C'est le V de MVC.

2.4 Le modèle

Le modèle M du MVC est ici constitué des éléments suivants :

1. les classes métier
2. les classes d'accès aux données
3. la base de données

2.4.1 La base de données

La base de données ne contient qu'une table appelée ARTICLES générée avec les commandes SQL suivantes :

```
CREATE TABLE ARTICLES (
    ID            INTEGER NOT NULL,
    NOM           VARCHAR(20) NOT NULL,
    PRIX          NUMERIC(15,2) NOT NULL,
    STOCKACTUEL   INTEGER NOT NULL,
    STOCKMINIMUM  INTEGER NOT NULL
);
/* contraintes */
ALTER TABLE ARTICLES ADD CONSTRAINT CHK_ID check (ID>0);
ALTER TABLE ARTICLES ADD CONSTRAINT CHK_PRIX check (PRIX>=0);
ALTER TABLE ARTICLES ADD CONSTRAINT CHK_STOCKACTUEL check (STOCKACTUEL>=0);
ALTER TABLE ARTICLES ADD CONSTRAINT CHK_STOCKMINIMUM check (STOCKMINIMUM>=0);
ALTER TABLE ARTICLES ADD CONSTRAINT CHK_NOM check (NOM<>'');
ALTER TABLE ARTICLES ADD CONSTRAINT UNQ_NOM UNIQUE (NOM);
/* clé primaire */
ALTER TABLE ARTICLES ADD CONSTRAINT PK_ARTICLES PRIMARY KEY (ID);
```

`id`	clé primaire identifiant un article de façon unique
`nom`	nom de l'article
`prix`	son prix
`stockactuel`	son stock actuel
`stockminimum`	le stock au-dessous duquel une commande de réapprovisionnement doit être faite

2.4.2 Les espaces de noms du modèle

Le modèle M est fourni sous la forme de deux espaces de noms :

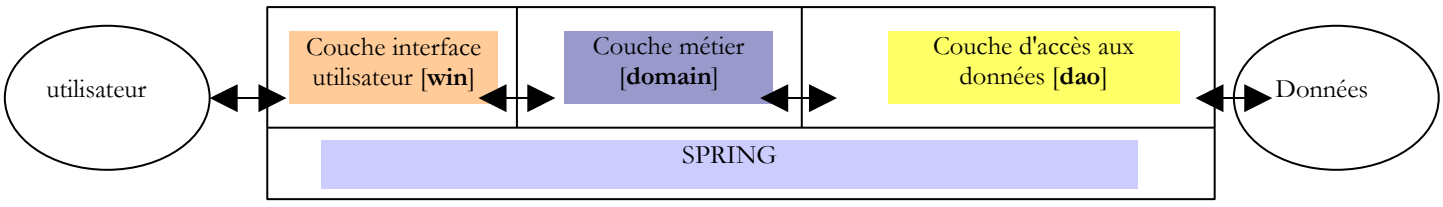
- **istia.st.articles.dao** : contient les classes d'accès aux données de la couche [dao]
- **istia.st.articles.domain** : contient les classes métier de la couche [domain]

Chacun de ces espaces de noms est contenu au sein d'un fichier " assembly " qui lui est propre :

assembly	*contenu*	*rôle*
webarticles-dao	- [IArticlesDao]: l'interface d'accès à la couche [dao] C'est la seule interface que voit la couche [domain]. Elle n'en voit pas d'autre. - [Article] : classe définissant un article - [ArticlesDaoArrayList] : classe d'implémentation de l'interface [IArticlesDao] avec une classe [ArrayList]	couche d'accès aux données - se trouve entièrement dans la couche [dao] de l'architecture 3-tier de l'application web
webarticles-domain	- [IArticlesDomain]: l'interface d'accès à la couche [domain]. C'est la seule interface que voit la couche web. Elle n'en voit pas d'autre. - [AchatsArticles] : une classe implémentant [IArticlesDomain] - [Achat] : classe représentant l'achat d'un client - [Panier] : classe représentant l'ensemble des achats d'un client	représente le modèle des achats sur le web - se trouve entièrement dans la couche [domain] de l'architecture 3-tier de l'application web

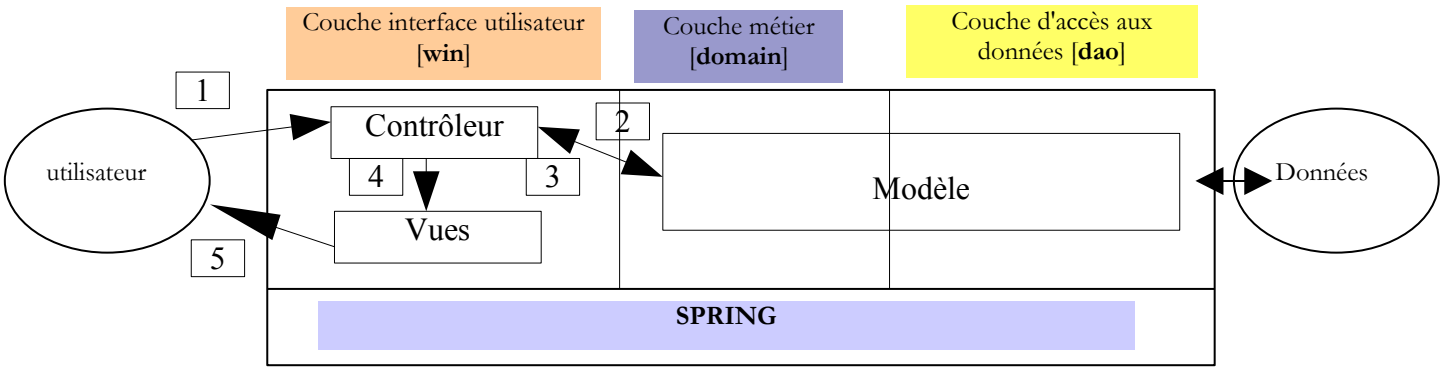
3 L'architecture de l'application windows [winarticles]

Nous allons construire une application windows qui reprendra l'architecture de l'application web précédente. On référençait cette dernière par [webarticles]. Nous référencerons la nouvelle par [winarticles]. Elle présentera l'architecture à trois couches suivante :



- les trois couches sont rendues indépendantes grâce à l'utilisation d'interfaces
- l'intégration des différentes couches est réalisée avec **Spring**
- chaque couche fait l'objet d'espaces de noms séparés : **win** (couche UI), **domain** (couche métier) et **dao** (couche d'accès aux données).

L'application respecte une architecture MVC (Modèle - Vue - Contrôleur). Si nous reprenons le schéma en couches ci-dessus, l'architecture MVC s'y intègre de la façon suivante :



Le fonctionnement de l'ensemble, décrit plus haut, peut être repris ici à l'identique. Les couches [**dao**] et [**domain**] ont déjà été construites dans les articles précédents. Ce seront pour nous des boîtes noires dont nous rappellerons cependant les caractéristiques principales. La couche [win] sera réalisée à l'aide du moteur [**M2VC-win**]. Cette couche est l'objet de cet article.

4 La couche [dao]

La couche [dao] choisie est celle implémentée par une classe utilisant l'outil [Ibatis SqlMap]. Le lecteur est invité à revoir éventuellement cette implémentation dans [article2], paragraphe 8.6. Rappelons-en quelques caractéristiques :

- [**IArticlesDao**] : l'interface d'accès à la couche [dao]
- [**ArticlesDaoSqlMap**] : la classe d'implémentation de cette interface
- [**Article**] : classe définissant un article

La classe définissant un article possède les propriétés publique suivantes :

id - Integer	identifiant de l'article
nom - String	nom de l'article
prix - Double	prix de l'article
stockactuel - Integer	stock actuel de l'article
stockminimum - Integer	si stockactuel<stockminimum alors il faut réapprovisionner

Cette classe offre par ailleurs :

1. un constructeur permettant de fixer les 5 informations d'un article : [id, nom, prix, stockactuel, stockminimum]
2. une vérification des données insérées dans l'article. En cas de données erronées, une exception est lancée.
3. une méthode **toString** qui permet d'obtenir la valeur d'un article sous forme de chaîne de caractères.

L'interface [**IArticlesDao**] est définie comme suit :

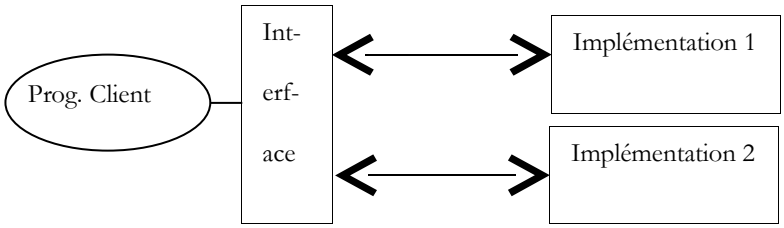
```
Imports System
Imports System.Collections

Namespace istia.st.articles.dao

  Public Interface IArticlesDao
    ' liste de tous les articles
    Function getAllArticles() As IList
    ' ajoute un article
    Function ajouteArticle(ByVal unArticle As Article) As Integer
    ' supprime un article
    Function supprimeArticle(ByVal idArticle As Integer) As Integer
    ' modifie un article
    Function modifieArticle(ByVal unArticle As Article) As Integer
    ' recherche un article
    Function getArticleById(ByVal idArticle As Integer) As Article
    ' supprime tous les articles
    Sub clearAllArticles()
    ' change le stock d'u article
    Function changerStockArticle(ByVal idArticle As Integer, ByVal mouvement As Integer) As Integer
  End Interface
End Namespace
```

Le rôle des différentes méthodes de l'interface est le suivant :

getAllArticles	rend tous les articles de la source de données
clearAllArticles	vide la source de données

`getArticleById`	rend l'objet [Article] identifié par son numéro
`ajouteArticle`	permet d'ajouter un article à la source de données
`modifieArticle`	permet de modifier un article de la source de données
`supprimerArticle`	permet de supprimer un article de la source de données
`changerStockArticle`	permet de modifier le stock d'un article de la source de données

L'interface met à disposition des programmes clients un certain nombre de méthodes définies uniquement par leurs signatures. Elle ne s'occupe pas de la façon dont ces méthodes seront réellement implémentées. Cela amène de la souplesse dans une application. Le programme client fait ses appels sur une interface et non pas sur une implémentation précise de celle-ci.



Le choix d'une implémentation précise se fait au moyen d'un fichier de configuration **Spring**. L'implémentation [**ArticlesDaoSqlMap**] choisie ici donne un accès transparent à toutes sortes de bases de données. Par "transparent", nous entendons le fait que changer de SGBD n'a aucune conséquence sur le code. La transparence est obtenue au moyen des fichiers de configuration [**articles.xml**, **properties.xml**, **providers.config**, **sqlmap.config**] :

- **articles.xml**

Ce fichier décrit les commandes SQL a émettre pour obtenir les données nécessaires à la couche [dao] :

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<sqlMap namespace="Articles" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="SqlMap.xsd">
  <!-- les resultMap -->
  <resultMaps>
    <resultMap id="article" class="istia.st.articles.dao.Article">
      <result property="id" column="ID" />
      <result property="nom" column="NOM" />
      <result property="prix" column="PRIX" />
      <result property="stockactuel" column="STOCKACTUEL" />
      <result property="stockminimum" column="STOCKMINIMUM" />
    </resultMap>
  </resultMaps>
  <!-- les requêtes SQL -->
  <statements>
    <!-- obtention de tous les articles -->
    <select id="getAllArticles" resultMap="article">
       select ID,NOM,PRIX,STOCKACTUEL,STOCKMINIMUM FROM ARTICLES
      </select>
    <!-- suppression de tous les articles-->
    <delete id="clearAllArticles" resultClass="int">
       delete from ARTICLES
      </delete>
    <!-- insertion d'un article -->
    <insert id="insertArticle" parameterClass="istia.st.articles.dao.Article">
       insert into ARTICLES (id, nom, prix,stockactuel, stockminimum) values
       ( #id# , #nom# , #prix# , #stockactuel# , #stockminimum# )
      </insert>
    <!-- suppression d'un article -->
    <delete id="deleteArticle" parameterClass="int" resultClass="int">
       delete FROM ARTICLES where ID= #value#
      </delete>
    <!-- modification d'un article -->
    <update id="modifyArticle" parameterClass="istia.st.articles.dao.Article" resultClass="int">
       update ARTICLES set NOM= #nom# ,PRIX= #prix# ,STOCKACTUEL= #stockactuel# ,STOCKMINIMUM=
#stockminimum# where ID= #id#
      </update>
    <!-- recherche d'un article précis -->
    <select id="getArticleById" resultMap="article" parameterClass="int">
       select ID, NOM, PRIX, STOCKACTUEL, STOCKMINIMUM FROM ARTICLES where ID= #value#
      </select>
    <!-- changement du stock d'un article -->
    <update id="changerStockArticle" parameterClass="Hashtable">
       update ARTICLES set STOCKACTUEL=(STOCKACTUEL + #mouvement#) where ID=#id# and ((STOCKACTUEL +
#mouvement#) >=0)
      </update>
  </statements>
</sqlMap>
```

- **sqlmap.config**

Ce fichier configure l'accès aux données :

```
<?xml version="1.0" encoding="utf-8" ?>
<sqlMapConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Schemas\SqlMapConfig.xsd">
  <properties resource="bin/properties.xml"/>
  <settings>
    <setting useStatementNamespaces="false" />
    <setting cacheModelsEnabled="false" />
  </settings>
  <!-- ==== source de données =========-->
  <database>
    <provider name="${provider}"/>
    <dataSource name="sqlmaparticles"  connectionString="${connectionString}"/>
    <transactionManager type="ADO/SWC" />
  </database>
  <sqlMaps>
    <sqlMap resource="bin/articles.xml" />
  </sqlMaps>
</sqlMapConfig>
```

La balise [**properties**] désigne le fichier de propriétés dans lequel seront trouvées les valeurs des clés de la forme **${clé}** du fichier courant. La balise [**provider**] indique la méthode d'accès aux données. Chaque méthode est associée à une bibliothèque de classes qui lui est propre. L'attribut [**connectionString**] de la balise [**dataSource**] fournit la chaîne identifiant la base de données à exploiter. Enfin la balise <**sqlMaps**> (au pluriel) sert à définir des fichiers de correspondances classes .NET <--> tables de SGBD. Chaque fichier de correspondances est défini par une balise <**sqlMap**> (au singulier). Ici, nous retrouvons le fichier [**articles.xml**] déjà présenté.

- **properties.xml**

C'est un fichier de propriétés associant des valeurs à des clés.

```
<?xml version="1.0" encoding="utf-8" ?>
<settings>
  <add key="provider" value="OleDb1.1" />
  <add
    key="connectionString"
    value="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=dbarticles.mdb;"/>
</settings>
```

Le fichier ci-dessus donne des valeurs aux deux attributs [provider, connectionString] du fichier [**providers.config**]. Ci-dessus, le fournisseur d'accès est [**OleDb1.1**]. Ce fournisseur permet d'accéder aux sources de données disposant d'un pilote OleDB. La chaîne de connexion désigne le fichier ACCESS [dbarticles.mdb] ayant la table [ARTICLES] suivante :

articles : Table

id	nom	prix	stockactuel	stockminimum
1	articles1	100	9	2
2	articles2	200	20	4

- **providers.config**

Ce fichier définit les classes d'accès aux données associées à chaque fournisseur d'accès :

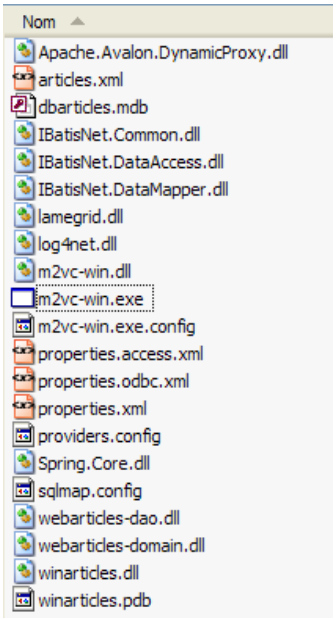
```
<?xml version="1.0" encoding="utf-8" ?>

<providers>
  <clear/>
  <provider
    name="Odbc1.1"
    enabled="true"
    assemblyName="System.Data, Version=1.0.5000.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
    connectionClass="System.Data.Odbc.OdbcConnection"
    commandClass="System.Data.Odbc.OdbcCommand"
    parameterClass="System.Data.Odbc.OdbcParameter"
    parameterDbTypeClass="System.Data.Odbc.OdbcType"
    parameterDbTypeProperty="OdbcType"
    dataAdapterClass="System.Data.Odbc.OdbcDataAdapter"
    commandBuilderClass="System.Data.Odbc.OdbcCommandBuilder"
    usePositionalParameters = "true"
    useParameterPrefixInSql = "false"
```

```
    useParameterPrefixInParameter = "false"
    parameterPrefix = "@"
  />
    <provider
    name="OleDb1.1"
    enabled="true"
    assemblyName="System.Data, Version=1.0.5000.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
    connectionClass="System.Data.OleDb.OleDbConnection"
    commandClass="System.Data.OleDb.OleDbCommand"
    parameterClass="System.Data.OleDb.OleDbParameter"
    parameterDbTypeClass="System.Data.OleDb.OleDbType"
    parameterDbTypeProperty="OleDbType"
    dataAdapterClass="System.Data.OleDb.OleDbDataAdapter"
    commandBuilderClass="System.Data.OleDb.OleDbCommandBuilder"
    usePositionalParameters = "true"
    useParameterPrefixInSql = "false"
    useParameterPrefixInParameter = "false"
    parameterPrefix = ""
  />
</providers>
```

Le fichier ci-dessus définit deux fournisseurs d'accès :

- [OleDb1.1] pour les sources OleDb
- [Odbc1.1] pour les sources Odbc

L'outil Ibatis SqlMap vient avec d'autres définitions de fournisseurs d'accès qui n'ont pas été intégrées au fichier ci-dessus. Au final, la couche [dao] va amener certains fichiers dans le dossier [bin] des exécutables de [winarticles] :



Les fichiers amenés par la couche [dao] dans le dossier des exécutables de [winarticles] seront les suivants :

- [Apache.Avalon.DynamicProxy.dll, articles.xml, IbatisNet.Common.dll, IbatisNet.DataAccess.dll, IbatisNet.DataMapper.dll, log4net.dll, properties.xml, providers.config, sqlmap.config] sont nécessaires à [Ibatis SqlMap].
- [log4net.dll, Spring.Core.dll] sont nécessaires à Spring.
- [webarticles-dao.dll] est le code de la couche d'accès à la base de données des articles.
- [dbarticles.mdb] est la base ACCESS que nous utiliserons pour nos tests.
- [properties.access.xml] est une version OleDB de [properties.xml], [properties.odbc.xml] une version ODBC. En copiant l'un de ces fichiers dans [properties.xml], le lecteur pourra utiliser la base [dbarticles.mdb] soit via un fournisseur d'accès OleDB, soit via un fournisseur d'accès Odbc1.1.

5 La couche [domain]

Le lecteur est invité à relire la définition de la couche [domain] dans [article1]. Nous en redonnons les grandes lignes.

5.1.1 Structure de la couche

La couche [domain] contient les éléments suivants :

- [**IArticlesDomain**]: l'interface d'accès à la couche [domain]
- [**Achat**] : classe définissant un achat
- [**Panier**] : classe définissant un panier d'achats
- [**AchatsArticles**] : classe d'implémentation de l'interface [IArticlesDomain]

5.1.2 L'interface [IArticlesDomain]

L'interface [IArticlesDomain] découple la couche [métier] de la couche [web]. Cette dernière accède à la couche [métier/domain] via cette interface sans se préoccuper de la classe qui l'implémente réellement. L'interface définit les actions suivantes pour l'accès à la couche métier :

```
Imports Article = istia.st.articles.dao.Article

Namespace istia.st.articles.domain
  Public Interface IArticlesDomain
    ' méthodes
    Sub acheter(ByVal panier As Panier)
    Function getAllArticles() As IList
    Function getArticleById(ByVal idArticle As Integer) As Article
    ReadOnly Property erreurs() As ArrayList
  End Interface
End Namespace
```

`Function getAllArticles() As IList`	rend la liste d'objets [Article] de la source de données associée
`Function getArticleById(ByVal idArticle As Integer) As Article`	rend l'objet [Article] identifié par [idArticle]
`acheter(ByVal panier As Panier)`	valide le panier du client en décrémentant les stocks des articles achetés de la quantité achetée - peut échouer si le stock est insuffisant
`ReadOnly Property erreurs() As ArrayList`	rend la liste des erreurs qui se sont produites - vide si pas d'erreurs

5.1.3 La classe [Achat]

La classe [Achat] représente un achat du client. Elle a les propriétés et méthodes suivantes :

`Public Property article() As article`	l'article acheté
`Public Property qte() As Integer`	la quantité achetée
`Public ReadOnly Property totalAchat() As Double`	le montant de l'achat
`Public Overrides Function ToString() As String`	chaîne d'identité de l'objet
`New(ByVal unArticle As article, ByVal qte As Integer)`	le constructeur

5.1.4 La classe [Panier]

La classe [Panier] représente l'ensemble des achats du client. Elle a les propriétés et méthodes suivantes :

`Public ReadOnly Property achats() As ArrayList`	la liste des achats du client - liste d'objets de type [Achat]
`Public ajouter(ByVal unAchat As Achat)`	ajoute un achat à la liste des achats
`Public enlever(ByVal idAchat As Integer)`	enlève l'achat de l'article idAchat
`Public ReadOnly Property totalPanier() As Double`	le montant total des achats du panier
`Oublic Function ToString() As String`	rend la chaîne d'identité du panier

5.1.5 La classe [AchatsArticles]

L'interface [IArticlesDomain] a été implémentée par la classe [AchatsArticles] suivante :

```
Imports istia.st.articles.dao

Namespace istia.st.articles.domain
  Public Class AchatsArticles
    Implements IArticlesDomain

    'champs privés
    Private _articlesDao As IArticlesDao
    Private _erreurs As ArrayList
```

```
    ' constructeur
    Public Sub New(ByVal articlesDao As IArticlesDao)
      _articlesDao = articlesDao
    End Sub

    ' liste des erreurs
    Public ReadOnly Property erreurs() As ArrayList Implements IArticlesDomain.erreurs
      Get
        Return _erreurs
      End Get
    End Property

    ' liste des articles
    Public Function getAllArticles() As IList Implements IArticlesDomain.getAllArticles
      ' liste de tous les articles
      Try
        Return _articlesDao.getAllArticles
      Catch ex As Exception
        _erreurs = New ArrayList
        _erreurs.Add("Erreur d'accès aux données : " + ex.Message)
      End Try
    End Function

    ' obtenir un article identifié par son n°
    Public Function getArticleById(ByVal idArticle As Integer) As Article Implements
IArticlesDomain.getArticleById
      ' un article particulier
      Try
        Return _articlesDao.getArticleById(idArticle)
      Catch ex As Exception
        _erreurs = New ArrayList
        _erreurs.Add("Erreur d'accès aux données : " + ex.Message)
      End Try
    End Function


    ' acheter un panier
    Public Sub acheter(ByVal panier As Panier) Implements IArticlesDomain.acheter
      ' achat d'un panier - les stocks des articles achetés doivent être décrémentés
      _erreurs = New ArrayList
      Dim achat As achat
      Dim achats As ArrayList = panier.achats
      For i As Integer = achats.Count - 1 To 0 Step -1
        ' décrémenter stock article i
        achat = CType(achats(i), achat)
        Try
          If _articlesDao.changerStockArticle(achat.article.id, -achat.qte) = 0 Then
            ' on n'a pas pu faire l'opération
            _erreurs.Add("L'achat " + achat.ToString + " n'a pu se faire - Vérifiez les stocks")
          Else
            ' l'opération s'est faite - on enlève l'achat du panier
            panier.enlever(achat.article.id)
          End If
        Catch ex As Exception
          _erreurs = New ArrayList
          _erreurs.Add("Erreur d'accès aux données : " + ex.Message)
        End Try
      Next
    End Sub
  End Class

End Namespace
```

Commentaires :

- cette classe implémente les quatre méthodes de l'interface [IArticlesDomain]. Elle a deux champs privés :

`_articlesDao As IArticlesDao`	l'objet d'accès aux données
`_erreurs As ArrayList`	la liste des erreurs éventuelles. Elle est accessible via la propriété publique [erreurs]

- pour construire une instance de la classe, il faut fournir l'objet permettant l'accès aux données :

```
Sub New(ByVal articlesDao As IArticlesDao)
```

- les méthodes [getAllArticles] et [getArticleById] s'appuient sur les méthodes de même nom de la couche [dao]

- la méthode [acheter] valide l'achat d'un panier. Cette validation consiste simplement à décrémenter les stocks des articles achetés. L'achat d'un article n'est possible que si son stock le permet. Si ce n'est pas le cas, l'achat est refusé : il reste dans le panier et une erreur est signalée dans la liste [erreurs]. Un achat validé est retiré du panier et le stock de l'article correspondant décrémenté de la quantité achetée.

Après usage, la gestion des erreurs faite par cette classe a paru maladroite. La classe intercepte en effet toutes les exceptions de la couche [dao] pour alimenter un [ArrayList] d'erreurs sans faire de disctinction entre erreurs d'accès à la base et erreurs dues au fait qu'un utilisateur a voulu acheter plus que les stocks ne le permettaient. Nous décidons de différencier ces deux types d'erreurs en laissant remonter les exceptions dues à des problèmes d'accès à la base. La classe d'implémentation de l'interface [IArticlesDomain] sera donc la suivante :

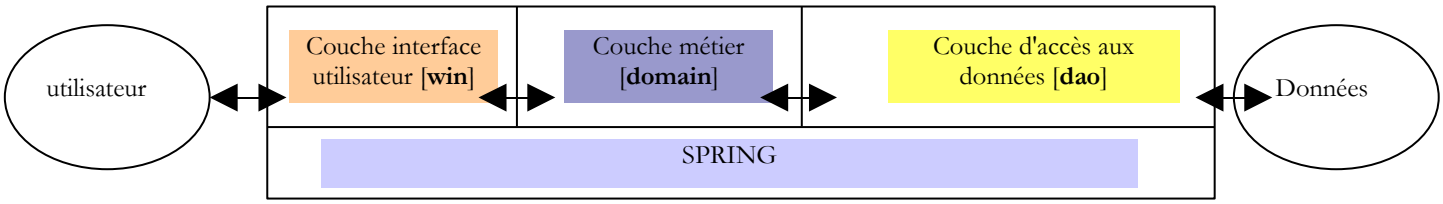
```
Imports istia.st.articles.dao

Namespace istia.st.articles.domain
  Public Class AchatsArticles
    Implements IArticlesDomain

    'champs privés
    Private _articlesDao As IArticlesDao
    Private _erreurs As New ArrayList

    ' constructeur
    Public Sub New(ByVal articlesDao As IArticlesDao)
      _articlesDao = articlesDao
    End Sub

    ' propriétés
    Public ReadOnly Property erreurs() As ArrayList Implements IArticlesDomain.erreurs
      Get
        Return _erreurs
      End Get
    End Property

    ' méthodes
    Public Function getAllArticles() As IList Implements IArticlesDomain.getAllArticles
      ' liste de tous les articles
      Return _articlesDao.getAllArticles
    End Function

    Public Function getArticleById(ByVal idArticle As Integer) As Article Implements
IArticlesDomain.getArticleById
      ' un article particulier
      Return _articlesDao.getArticleById(idArticle)
    End Function

    Public Sub acheter(ByVal panier As Panier) Implements IArticlesDomain.acheter
      ' achat d'un panier - les stocks des articles achetés doivent être décrémentés
      _erreurs.Clear()
      Dim achat As achat
      Dim achats As ArrayList = panier.achats
      For i As Integer = achats.Count - 1 To 0 Step -1
        ' décrémenter stock article i
        achat = CType(achats(i), achat)
        If _articlesDao.changerStockArticle(achat.article.id, -achat.qte) = 0 Then
          ' on n'a pas pu faire l'opération
          _erreurs.Add("L'achat " + achat.ToString + " n'a pu se faire - Vérifiez les stocks")
        Else
          ' l'opération s'est faite - on enlève l'achat du panier
          panier.enlever(achat.article.id)
        End If
      Next
    End Sub
  End Class

End Namespace
```

La couche [domain] amène la DLL [webarticles-domain.dll] dans le dossier des exécutables de [winarticles].

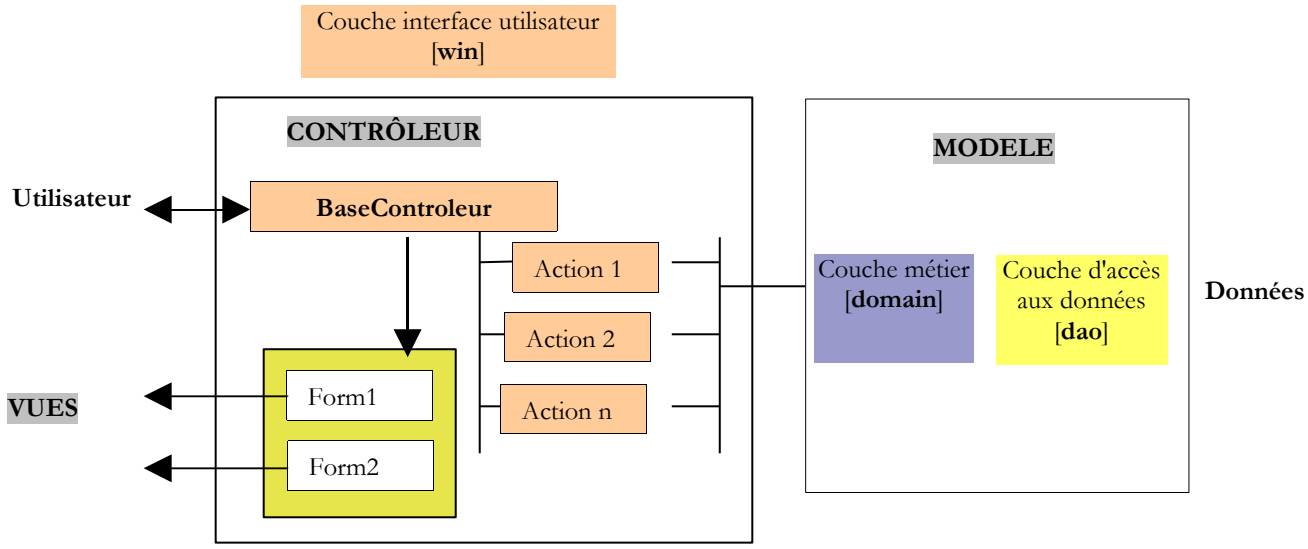
6 La couche [win]

6.1 Architecture générale

Revenons au schéma général de notre application [winarticles] :



La couche [win] est la couche d'interface avec l'utilisateur. Pour l'implémenter nous allons utiliser le moteur MVC [M2VC-win]. Celui-ci nous amène à implémenter la couche [win] de la façon suivante :

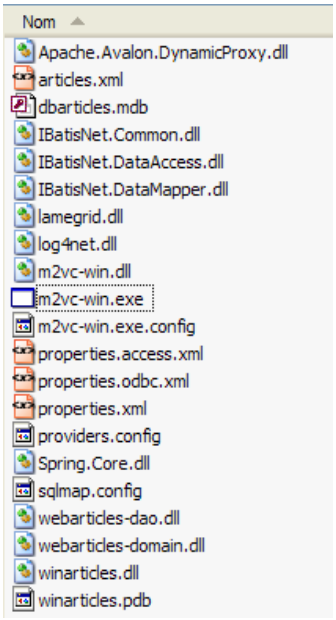


M=modèle	les classes métier, les classes d'accès aux données et la base de données
V=vues	les formulaires Windows
C=contrôleur	le contrôleur [BaseControleur] de traitement des requêtes clientes, les objets [Action]

Rappelons les grands principes de fonctionnement du moteur [M2VC-win] :

1. le contrôleur [BaseControleur] est le coeur de l'application. Toutes les demandes du client transitent par lui. C'est une classe fournie par [M2VC-win]. On peut dans certains cas être amené à la dériver. Pour les cas simples, ce n'est pas nécessaire.
2. [BaseControleur] prend les informations dont il a besoin dans un fichier appelé [**m2vc-win.exe.config**]. Il y trouve la liste des objets [Action] destinés à exécuter les demandes du client, la liste des vues de l'application, une liste d'objets [InfosAction] décrivant chaque action. [InfosAction] a les attributs suivants :
   - ✗ [vue] : désigne une vue [Form] à afficher si l'action ne consiste qu'à changer de vue.
   - ✗ [action] : désigne un objet [Action] à exécuter si l'action demandée nécessite l'exécution d'un code
   - ✗ [états] : un dictionnaire associant une vue à chacun des résultats possibles de l'objet [Action]. Le contrôleur affichera la vue associée au résultat renvoyé par l'action.
3. l'utilisateur a devant lui un formulaire windows. Celui-ci traite certains événements lui-même, ceux qui ne nécessitent pas la couche métier. Les autres sont délégués au contrôleur. On dit alors que la vue demande l'exécution d'une action au contrôleur. Le contrôleur reçoit cette demande sous la forme d'un nom d'action.
4. [BaseControleur] récupère alors l'instance [InfosAction] liée au nom de l'action qu'on lui demande d'exécuter. Pour cela, il a un dictionnaire associant le nom d'une action à une instance [InfosAction] rassemblant les informations nécessaires à cette action.
5. si l'attribut [vue] de [InfosAction] est non vide, alors la vue associée est affichée. On passe ensuite à l'étape 9.
6. si l'attribut [action] de [InfosAction] est non vide, alors l'action est exécutée. Celle-ci fait ce qu'elle a à faire puis rend au contrôleur une chaîne de caractères représentant le résultat auquel elle est parvenue.
7. le contrôleur utilise le dictionnaire [états] de [InfosAction] pour trouver la vue V à afficher. Il l'affiche.
8. ici une vue a été affichée. Le contrôleur s'est synchronisé avec et attend que la vue déclenche une nouvelle action. Celle-ci va être déclenchée par une action particulière de l'utilisateur sur la vue, qui à cette occasion va repasser la main au contrôleur en lui donnant le nom de l'action à exécuter.
9. le contrôleur reprend à l'étape 1

L'utilisation du moteur [M2VC-win] nécessite la présence des fichiers [m2vc-win.dll, m2vc-win.exe, m2vc-win.exe.config, Spring.Core.dll, log4net.dll] dans le dossier des exécutables de la solution :



6.2 Les vues de l'application

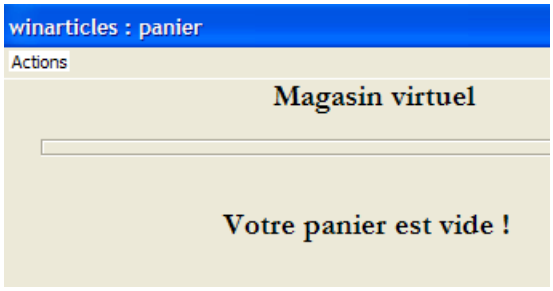
Les différentes vues présentées à l'utilisateur sont les suivantes :

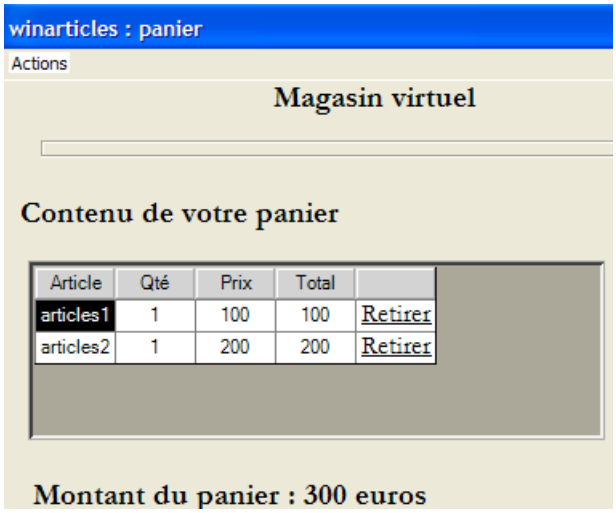
- la vue "LISTE" qui présente une liste des articles en vente

- la vue [INFOS] qui donne des informations supplémentaires sur un produit :

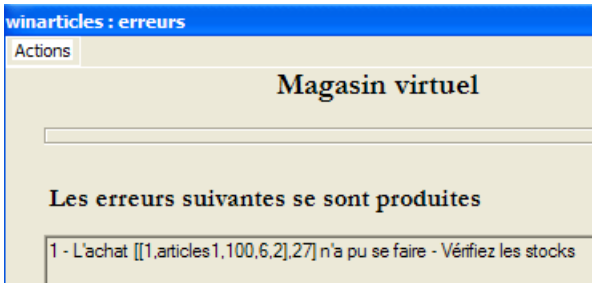
- la vue [PANIER] qui donne le contenu du panier du client

- la vue [PANIERVIDE] pour le cas où le panier du client est vide





- la vue [ERREURS] qui signale toute erreur d'achat d'articles



6.3 Fonctionnement de l'application [winarticles]

Nous présentons ci-dessous l'enchaînement des vues lors d'une utilisation typique de l'application :

A partir de la vue ci-dessus, nous utilisons les options du menu pour faire des opérations. En voici quelques unes. La colonne de gauche représente la demande du client et la colonne de droite la réponse qui lui est faite.


winarticles : liste
Actions
Magasin virtuel
Liste des articles
Nom
Prix
articles1
100
Info
articles2
200
Info


winarticles : infos
Actions
Magasin virtuel
Article d'ID [1]
Nom
Prix
Stock actuel
Stock minimum
articles1
100
6
2
Acheter
1

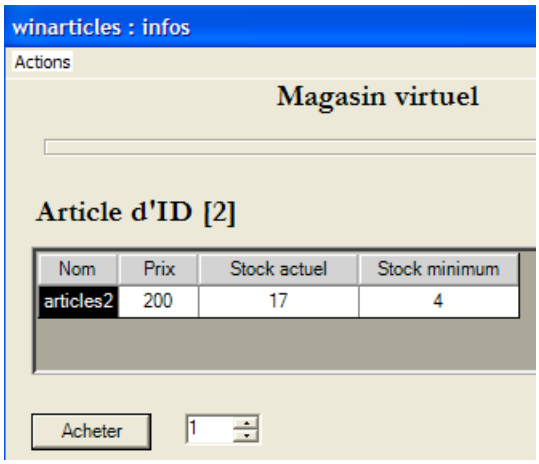

winarticles : infos
Actions
Magasin virtuel
Article d'ID [1]
Nom
Prix
Stock actuel
Stock minimum
articles1
100
6
2
Acheter
2

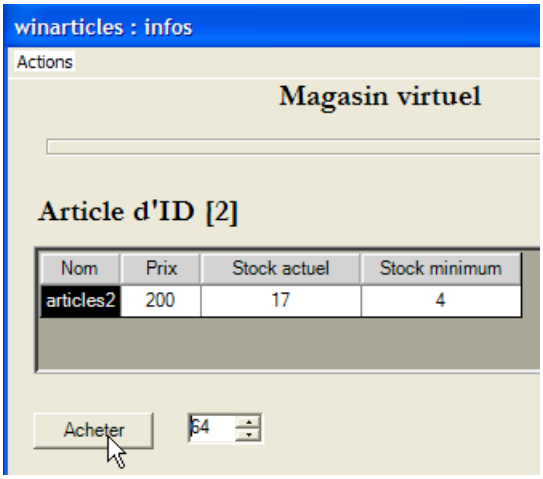

winarticles : panier
Actions
Magasin virtuel
Contenu de votre panier
Article
Qté
Prix
Total
articles1
2
100
200
Retirer
Montant du panier : 200 euros


winarticles : panier
Actions
Liste
Valider le panier
Quitter
Magasin virtuel
Contenu de votre panier
Article
Qté
Prix
Total
articles1
2
100
200
Retirer

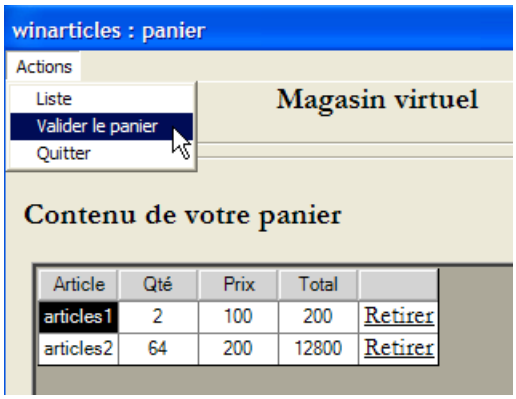

winarticles : liste
Actions
Magasin virtuel
Liste des articles
Nom
Prix
articles1
100
Info
articles2
200
Info

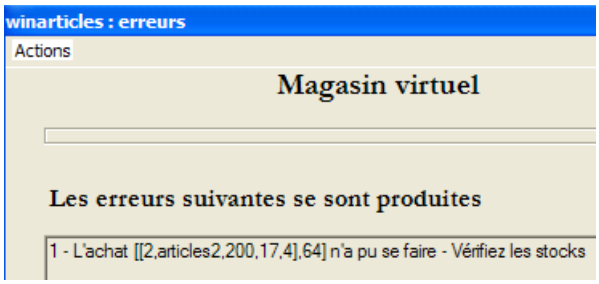

winarticles : liste
Actions
Magasin virtuel
Liste des articles
Nom
Prix
articles1
100
Info
articles2
200
Info


winarticles : infos
Actions
Magasin virtuel
Article d'ID [2]
Nom
Prix
Stock actuel
Stock minimum
articles2
200
17
4
Acheter
1

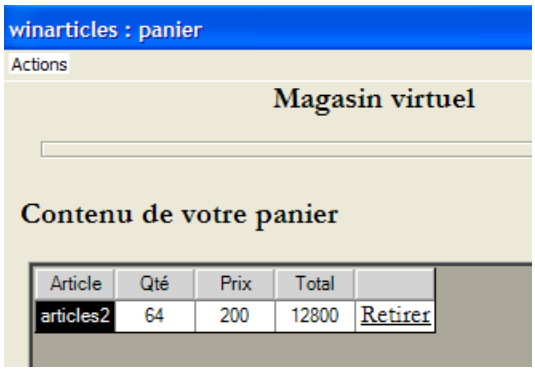

winarticles : infos
Actions
Magasin virtuel
Article d'ID [2]
Nom
Prix
Stock actuel
Stock minimum
articles2
200
17
4
Acheter
64

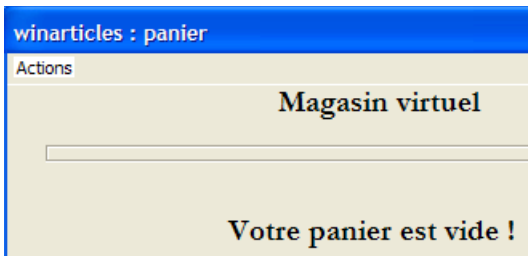

winarticles : panier
Actions
Magasin virtuel
Contenu de votre panier
Article
Qté
Prix
Total
articles1
2
100
200
Retirer
articles2
64
200
12800
Retirer

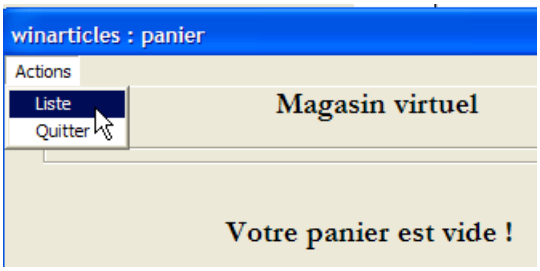

winarticles : panier
Actions
Liste
Valider le panier
Quitter
Magasin virtuel
Contenu de votre panier
Article
Qté
Prix
Total
articles1
2
100
200
Retirer
articles2
64
200
12800
Retirer


winarticles : erreurs
Actions
Magasin virtuel
Les erreurs suivantes se sont produites
1 - L'achat [[2,articles2,200,17,4],64] n'a pu se faire - Vérifiez les stocks


winarticles : erreurs
Actions
Liste
Voir le panier
Quitter
Magasin virtuel
Les erreurs suivantes se sont produites
1 - L'achat [[2,articles2,200,17,4],64] n'a pu se faire - Vérifiez les stocks

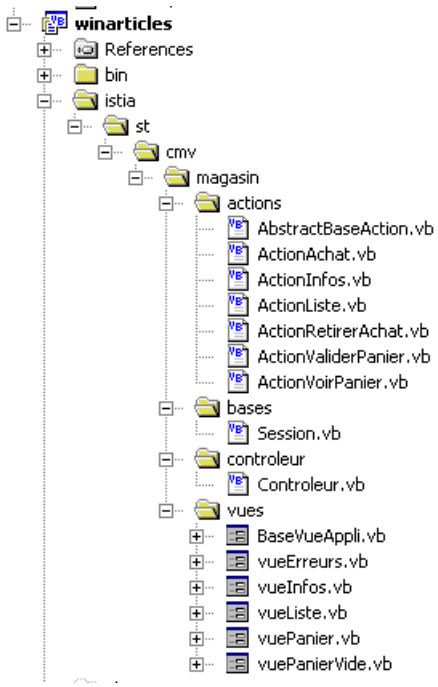

winarticles : panier
Actions
Magasin virtuel
Contenu de votre panier
Article
Qté
Prix
Total
articles2
64
200
12800
Retirer



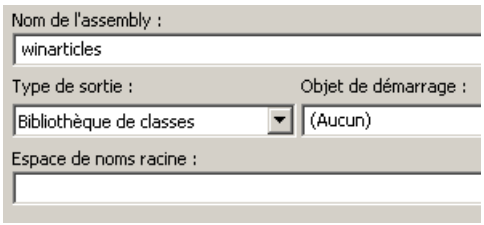


6.4 La structure de la solution Visual Studio

La structure de la solution Visual Studio de [winarticles] est la suivante :



Le projet [winarticles] est un projet de type [Bibliothèque de classes] configuré pour générer [winarticles.dll] :



Les classes et interfaces du projet sont placées dans l'espace de noms [istia.st.cmv.magasin] manuellement avec, à chaque fois, l'instruction :

```
Namespace istia.st.cmv.magasin
```

6.5 La session

Le moteur [M2VC-win] ne donne aucune aide pour les échanges d'informations entre vues et actions. C'est au développeur d'organiser ceux-ci. Nous créons ici un unique objet qui contiendra toutes les informations à partager entre les vues et les actions. Il n'y a pas de problème de synchronisation. Les objets du contrôleur ne sont jamais amenés à accéder à cet objet partagé de façon simultanée. Nous appelons cet objet [Session] par similitude avec l'objet [Session] des applications web dans lequel on stocke tout ce qu'on veut garder au fil des échanges client-serveur. Tous les objets d'une application web ont accès à cet objet [Session] comme ce sera le cas ici.

Notre classe [Session] est la suivante :

```
1. Namespace istia.st.cmv.magasin
2.
3.  Public Class Session
4.    ' contient les données partagées par les actions et les vues
5.
6.    ' le service d'accès à la couche métier
7.    Private _articlesDomain As istia.st.articles.domain.IArticlesDomain
8.    ' le panier des achats
9.    Private _panier As istia.st.articles.domain.Panier
10.   ' la liste des articles
11.   Private _articles As IList
12.   ' un article particulier
13.   Private _article As istia.st.articles.dao.Article
14.   ' un identifiant d'article
15.   Private _idArticle As Integer
16.   ' une liste d'erreurs
17.   Private _erreurs As ArrayList
18.   ' une quantité achetée
19.   Private _qte As Integer
20.   ' les différentes options de menu possibles
21.   Private _etatMenuListe As Boolean
22.   Private _etatMenuValiderPanier As Boolean
23.   Private _etatMenuVoirPanier As Boolean
24.   Private _etatMenuQuitter As Boolean
25.   ' un message à afficher
26.   Private _message As String
27.
28.   ' propriétés publiques associées aux champs privés
29.   Public Property articlesDomain() As istia.st.articles.domain.IArticlesDomain
30.     Get
31.       Return _articlesDomain
32.     End Get
33.     Set(ByVal Value As istia.st.articles.domain.IArticlesDomain)
34.       _articlesDomain = Value
35.     End Set
36.   End Property
37.....
38.
39. End Class
40.End Namespace
```

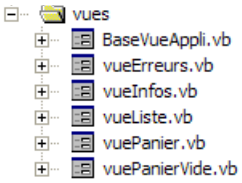
Les éléments de la classe sont tous déclarés privés et on déclare des propriétés publiques pour y accéder. Nous ne présentons ci-dessus que la première propriété publique. Les autres sont analogues et omises.

Les éléments de l'objet [Session] sont les suivants :

`articlesDomain - ligne 7`	le service d'accès à la couche métier. Sert uniquement aux objets [Action] pas aux vues.
`panier - ligne 9`	le panier des achats. Utilisé par la vue [Panier] et les actions [ActionAchat, ActionRetirerAchat, ActionVoirPanier, ActionValiderPanier]
`articles - ligne 11`	la liste des articles en vente. Utilisé par la vue [VueListe] et l'action [ActionListe]
`article - ligne 13`	un article particulier. Utilisé par la vue [VueInfos] et l'action [ActionInfos]
`idArticle - ligne 15`	l'identifiant d'un article particulier. Utilisé par les vues [VueInfos] et les actions [ActionAchat, ActionRetirerAchat]
`erreurs - ligne 17`	une liste d'erreurs. Utilisé par la vue [VueErreurs] et par toutes les actions susceptibles de rencontrer des erreurs [VueListe, VueInfos, VueValiderPanier]
`qte - ligne 19`	la quantité achetée d'un article particulier. Utilisé par la vue [VueInfos] et l'action [ActionAchat]
`les options de menu - lignes 21-24`	fixe l'état des options du menu de la vue [BaseVueAppli]. Cette vue, classe de base de toutes les autres vues, a un menu à quatre options [Liste, Voir le panier, Valider le panier, Quitter]. Les attributs [etatMenuListe, etatMenuVoirPanier, etatMenuValiderPanier, etatMenuQuitter] de l'objet [Session] permettent de fixer l'éta visible ou non de ces quatre options. Uitilisé par le contrôleur [Controleur] et la vue [BaseVueAppli].
`message - ligne 26`	un message affiché dans la vue [VueListe] lorsqu'une validation de panier a réussi. Utilisé par la vue [VueListe] et l'action [ActionValiderPanier]

6.6 Les vues

Elles sont implémentées par les classes suivantes :



6.6.1 La vue [BaseVueAppli]

La vue [BaseVueAppli] est une classe de base pour les cinq autres vues [VueListe, VueInfos, VuePanier, VuePanierVide, VueErreurs]. On y a mis le comportement commun des cinq vues :

```
1. Imports System.Windows.Forms
2. Imports System.Drawing
3. Imports istia.st.cmv
4. Namespace istia.st.cmv.magasin
5.
6.  Public Class BaseVueAppli
7.    Inherits BaseVue
8.
9. #Region " Code généré par le Concepteur Windows Form "
10....
11.   Friend WithEvents GroupBox1 As System.Windows.Forms.GroupBox
12.   Friend WithEvents MainMenu1 As System.Windows.Forms.MainMenu
13.   Friend WithEvents MenuActions As System.Windows.Forms.MenuItem
14.   Friend WithEvents MenuListe As System.Windows.Forms.MenuItem
15.   Friend WithEvents MenuVoirPanier As System.Windows.Forms.MenuItem
16.   Friend WithEvents MenuValiderPanier As System.Windows.Forms.MenuItem
17.   Friend WithEvents MenuQuitter As System.Windows.Forms.MenuItem
18.   Friend WithEvents LabelTitre As System.Windows.Forms.Label
19.
20. ...
21.
22.#End Region
23.
24.   ' données d'instance
25.   Private _session As Session
26.
27.   ' propriété publique associée
28.   Public Property session() As Session
```

```
29.     Get
30.       Return _session
31.     End Get
32.     Set(ByVal Value As Session)
33.       _session = Value
34.     End Set
35.   End Property
36.
37.   ' l'affichage du formulaire
38.   Public Overrides Sub affiche()
39.     ' le menu
40.     MenuListe.Visible = session.etatMenuListe
41.     MenuVoirPanier.Visible = session.etatMenuVoirPanier
42.     MenuValiderPanier.Visible = session.etatMenuValiderPanier
43.     MenuQuitter.Visible = session.etatMenuQuitter
44.     ' la position
45.     Me.StartPosition = FormStartPosition.CenterScreen
46.     ' le titre
47.     LabelTitre.Text = "Magasin virtuel"
48.     ' on dégèle le formulaire
49.     Me.Enabled = True
50.     ' on affiche la classe parent
51.     MyBase.affiche()
52.     ' on active le formulaire
53.     Me.Activate()
54.   End Sub
55.
56.   ' exécution asynchrone
57.   Protected Sub exécuteAction(ByVal action As String)
58.     ' fait exécuter [action] par le contrôleur
59.     ' le titre
60.     LabelTitre.Text = "Magasin virtuel : patientez..."
61.     ' on gèle le formulaire
62.     Me.Enabled = False
63.     ' on passe la main à la classe de base
64.     MyBase.exécute(action)
65.   End Sub
66.
67.   ' les évts du menu
68.   Private Sub MenuListe_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
    MenuListe.Click
69.     ' on exécute l'action [liste]
70.     exécuteAction("liste")
71.   End Sub
72.
73.   Private Sub MenuVoirPanier_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
    MenuVoirPanier.Click
74.     ' on exécute l'action [voirpanier]
75.     exécuteAction("voirpanier")
76.   End Sub
77.
78.   Private Sub MenuValiderPanier_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
    Handles MenuValiderPanier.Click
79.     ' on exécute l'action [validerpanier]
80.     exécuteAction("validerpanier")
81.   End Sub
82.
83.   Private Sub MenuQuitter_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
    MenuQuitter.Click
84.     ' on exécute l'action [quitter]
85.     exécuteAction("quitter")
86.   End Sub
87. End Class
88.End Namespace
```

- la classe [BaseVueAppli] dérive de [BaseVue] une classe de base pour les vues définie dans le moteur [M2VC-win] - lignes 6-7
- l'objet [Session] défini précédemment devra être injecté dans tous les actions [IAction] et les vues [IVue]. La classe [BaseVueAppli] définit donc une propriété publique appelée [session] pour intégrer cet objet (lignes 25-35).
- la méthode [affiche] (lignes 38-54) définit un mode d'affichage standard pour toutes les vues de l'application. Elle affiche les options du menu du formulaire de base d'après les indications mises par le contrôleur dans la session (lignes 40-43). Elle " dégèle " le formulaire affiché (ligne 49). En effet, nous allons voir bientôt qu'à l'exécution d'une action asynchrone par le contrôleur, le formulaire se met en attente du résultat de l'action en se " gelant " (ligne 62). La méthode [affiche] termine l'affichage du formulaire par l'affichage de la classe de base (ligne 51). C'est obligatoire.
- la méthode [affiche] se termine en activant le formulaire (ligne 53). Les tests ont montré en effet que, sans cette opération, les formulaires étaient correctement affichés mais parfois sans avoir le focus. La ligne 53 met donc systématiquement le focus sur le formulaire qui vient d'être affiché.
- la méthode [exécuteAction] a pour but de donner un comportement standard à l'exécution des actions asynchrones des différentes vues :
  - le titre est modifié pour indiquer une attente (ligne 60)
  - la vue est gelée - ligne 62

- l'exécution de l'action est déléguée à la classe de base [BaseVue] (ligne 64).
- la vue retrouvera un état actif lorsqu'elle sera réaffichée (ligne 49)

- la vue [BaseVueAppli] a un unique composant dont elle gère les événements. C'est le menu, lignes 12-17. Les gestionnaires d'événements sont :
  - lignes 68-71 : on y gère le clic sur l'option [Liste] en faisant exécuter l'action asynchrone " liste "
  - lignes 73-76 : on y gère le clic sur l'option [VoirPanier] en faisant exécuter l'action asynchrone "voirpanier"
  - lignes 78-81 : on y gère le clic sur l'option [ValiderPanier] en faisant exécuter l'action asynchrone "validerpanier"
  - lignes 83-86 : on y gère le clic sur l'option [Quitter] en faisant exécuter l'action asynchrone "quitter"

6.6.2 La vue [VueListe]

Rappelons l'aspect visuel ce cette vue :

La liste des articles est affichée dans un composant [LameGrid]. Ce composant est disponible à l'URL suivante : [http://kikos31.developpez.com/lamegrid/]. Il présente l'avantage d'être plus simple d'utilisation que le composant [DataGrid] de Visual Studio.

Le code de la classe [VueListe] est le suivant :

```
1. Imports System.Windows.Forms
2. Imports istia.st.articles.dao
3. Imports System.Drawing
4.
5. Namespace istia.st.cmv.magasin
6.  Public Class Vue_Liste
7.    Inherits BaseVueAppli
8.
9. #Region " Code généré par le Concepteur Windows Form "
10....
11.   Friend WithEvents Label2 As System.Windows.Forms.Label
12.   Friend WithEvents GridArticles As LameGrid.Grid
13.   Friend WithEvents LabelMessage As System.Windows.Forms.Label
14....
15.#End Region
16.
17.   ' données globales
18.   Dim police As New Font(New FontFamily("Garamond"), 12, FontStyle.Underline)
19.
20.   ' init formulaire
21.   Public Overrides Sub affiche()
22.     ' nettoyage grille des articles
23.     For i As Integer = GridArticles.RowsCount - 1 To 0 Step -1
24.       GridArticles.RemoveRow(i)
25.     Next
26.     ' remplit la gridarticles avec les éléments de la liste [articles]
27.     Dim articles As IList = session.articles
28.     ' ligne d'entête
29.     GridArticles.AddRow()
30.     GridArticles(0, 0).Text = "Nom"
31.     GridArticles(0, 1).Text = "Prix"
32.     ' affichage de la liste d'articles
33.     Dim unArticle As Article
34.     For i As Integer = 0 To articles.Count - 1
35.       GridArticles.AddRow()
36.       unArticle = CType(articles(i), Article)
37.       GridArticles(i + 1, 0).Text = unArticle.nom
38.       GridArticles(i + 1, 1).Text = unArticle.prix.ToString
```

```
39.      With GridArticles(i + 1, 2)
40.        .Text = "Info"
41.        .Font = police
42.      End With
43.    Next
44.    ' affichage message
45.    LabelMessage.Text = session.message
46.    ' affichage parent
47.    MyBase.affiche()
48.  End Sub
49.
50.  ' demande d'informations sur un article
51.  Private Sub GridArticles_Click(ByVal Row As Integer, ByVal Col As Integer, ByVal Header As
   LameGrid.HeaderInfo) Handles GridArticles.CellClick
52.    ' on ne s'intéresse qu'à la colonne 2 [Infos] et pas à la ligne 0
53.    If Col <> 2 or Row = 0 Then Exit Sub
54.    ' on note l'ID de l'article cliqué
55.    session.idArticle = CType(session.articles(Row - 1), Article).id
56.    ' on fait exécuter l'action
57.    MyBase.exécuteAction("infos")
58.  End Sub
59.
60. End Class
61.End Namespace
```

- la classe [VueListe] dérive de [BaseVueAppli] la classe de base pour toutes les vues de l'application - lignes 6-7
- les composants de la vue sont définis lignes 11-13. On ne gère que les événements de la grille [GridArticles]
- la méthode [affiche] (lignes 21-48) gère l'affichage des composants propres au formulaire [VueListe]. Elle commence par nettoyer le composant [GridArticles] en supprimant toutes les lignes que celui-ci pouvait avoir (lignes 23-25). Ceci fait elle régénère [GridArticles] (lignes 29-43) avec la liste d'articles qu'elle trouve dans [session.articles] (ligne 27). On rappelle que [session] est un attribut de la classe de base [BaseVueAppli]. La méthode [affiche] affiche ensuite l'éventuel message qu'on lui a passé dans la session (ligne 45) puis se termine en demandant à sa classe de base de s'afficher (ligne 47).
- le formulaire ne gère qu'un événement : le clic sur la colonne [Infos] de la grille [GridArticles]. Celui-ci est géré lignes 51-58. On note l'identifiant de l'article cliqué et on le met dans la session (ligne 55). Puis, on demande l'exécution de l'action asynchrone " infos " (ligne 57). Le contrôleur va alors prendre la main.

6.6.3 La vue [VueInfos]

Rappelons l'aspect visuel ce cette vue :

L'article est affiché dans un composant [LameGrid].

Le code de la classe [VueInfos] est le suivant :

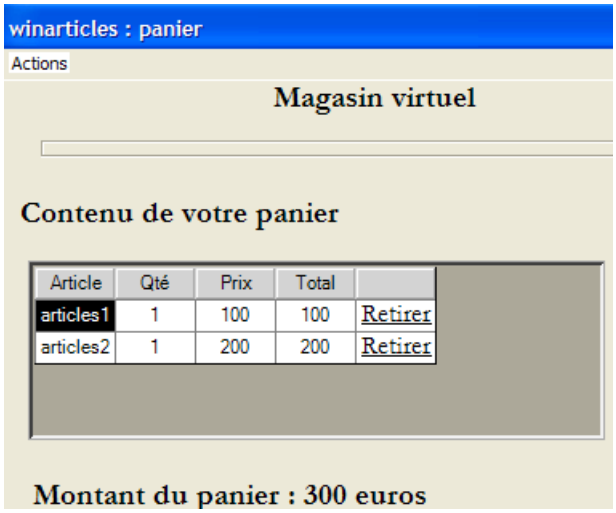
```
1. Imports istia.st.articles.dao
2. Imports System.Windows.Forms
3.
4. Namespace istia.st.cmv.magasin
5.  Public Class Vue_Infos
6.    Inherits BaseVueAppli
7.
8. #Region " Code généré par le Concepteur Windows Form "
9. ...
10.   Friend WithEvents lblID As System.Windows.Forms.Label
11.   Friend WithEvents GridInfos As LameGrid.Grid
12.   Friend WithEvents btnAcheter As System.Windows.Forms.Button
13.   Friend WithEvents NumericUpDownQte As System.Windows.Forms.NumericUpDown
14....
```

```
15.#End Region
16.
17.   ' affichage formulaire
18.   Public Overrides Sub affiche()
19.     ' on nettoie le [LameGrid]
20.     For i As Integer = GridInfos.RowsCount - 1 To 0 Step -1
21.       GridInfos.RemoveRow(i)
22.     Next
23.     ' init formulaire avec les informations d'un article donné
24.     Dim unArticle As Article = session.article
25.     ' le label
26.     lblID.Text = String.Format("Article d'ID [{0}]", unArticle.id)
27.     ' l'incrémenteur
28.     NumericUpDownQte.Value = 1
29.     ' la grille - ligne d'entête
30.     GridInfos.AddRow()
31.     GridInfos(0, 0).Text = "Nom"
32.     GridInfos(0, 1).Text = "Prix"
33.     GridInfos(0, 2).Text = "Stock actuel"
34.     GridInfos(0, 3).Text = "Stock minimum"
35.     GridInfos.Col(0).Width = 50
36.     GridInfos.Col(1).Width = 50
37.     GridInfos.Col(2).Width = 100
38.     GridInfos.Col(3).Width = 100
39.     ' affichage de l'article
40.     GridInfos.AddRow()
41.     GridInfos(1, 0).Text = unArticle.nom
42.     GridInfos(1, 1).Text = unArticle.prix.ToString
43.     GridInfos(1, 2).Text = unArticle.stockactuel.ToString
44.     GridInfos(1, 3).Text = unArticle.stockminimum.ToString
45.     ' affichage parent
46.     MyBase.affiche()
47.   End Sub
48.
49.   Private Sub btnAcheter_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
   btnAcheter.Click
50.     ' achat de l'article affiché
51.     ' on note la qté achetée
52.     session.qte = CType(NumericUpDownQte.Value, Integer)
53.     ' on fait exécuter l'action
54.     MyBase.exécuteAction("achat")
55.   End Sub
56.
57. End Class
58.End Namespace
```

- la classe [VueInfos] dérive de [BaseVueAppli] la classe de base pour toutes les vues de l'application - lignes 5-6
- les composants de la vue sont définis lignes 10-13. On ne gère que les événements du bouton [btnAcheter].
- la méthode [affiche] (lignes 18-46) gère l'affichage des composants propres au formulaire [VueInfos]. Elle commence par nettoyer le formulaire en supprimant toutes les lignes que pouvait avoir le composant [GridInfos] (lignes 20-22). Ceci fait elle
    - récupère dans la session l'article à afficher - ligne 24
    - affiche l'identifiant de cet article - ligne 26
    - remet à 1 l'incrémenteur des quantités achetées - ligne 28
    - régénère [GridInfos] (lignes 29-44)
    - affiche la classe de base (ligne 46).
- le formulaire ne gère qu'un événement : le clic sur le bouton [Acheter]. Celui-ci est géré lignes 49-55. On note la quantité achetée et on met celle-ci dans la session (ligne 52). Puis, on demande l'exécution de l'action asynchrone "achat" (ligne 54). Le contrôleur va alors prendre la main.

6.6.4 La vue [VuePanier]

Rappelons l'aspect visuel ce cette vue :



Les achats sont affichés dans un composant [LameGrid].

Le code de la classe [VuePanier] est le suivant :

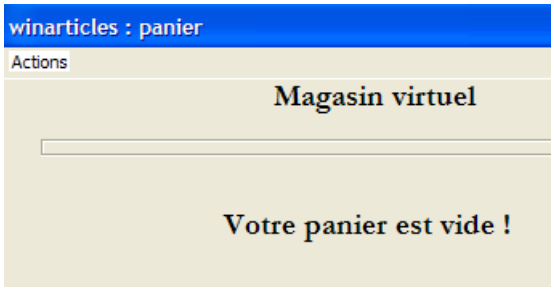
```
1. Imports System.Windows.Forms
2. Imports istia.st.articles.dao
3. Imports istia.st.articles.domain
4. Imports System.Drawing
5.
6. Namespace istia.st.cmv.magasin
7.  Public Class Vue_Panier
8.    Inherits BaseVueAppli
9.
10.#Region " Code généré par le Concepteur Windows Form "
11.
12....
13.   Friend WithEvents Label2 As System.Windows.Forms.Label
14.   Friend WithEvents GridAchats As LameGrid.Grid
15.   Friend WithEvents LabelMontantPanier As System.Windows.Forms.Label
16....
17.#End Region
18.
19.   ' données globales
20.   Dim police As New Font(New FontFamily("Garamond"), 12, FontStyle.Underline)
21.
22.   ' init formulaire
23.   Public Overrides Sub affiche()
24.     ' nettoyage [LameGrid]
25.     For i As Integer = GridAchats.RowsCount - 1 To 0 Step -1
26.       GridAchats.RemoveRow(i)
27.     Next
28.     '
29.     ' init formulaire avec les informations du panier
30.     Dim monPanier As Panier = session.panier
31.     ' ligne d'entête
32.     GridAchats.AddRow()
33.     GridAchats(0, 0).Text = "Article"
34.     GridAchats(0, 1).Text = "Qté"
35.     GridAchats(0, 2).Text = "Prix"
36.     GridAchats(0, 3).Text = "Total"
37.     GridAchats.Col(0).Width = 50
38.     GridAchats.Col(1).Width = 50
39.     GridAchats.Col(2).Width = 50
40.     GridAchats.Col(3).Width = 50
41.     ' affichage du contenu du panier
42.     Dim unAchat As Achat
43.     For i As Integer = 0 To monPanier.achats.Count - 1
44.       ' affichage achat n° i
45.       unAchat = CType(monPanier.achats(i), Achat)
46.       GridAchats.AddRow()
47.       GridAchats(i + 1, 0).Text = unAchat.article.nom
48.       GridAchats(i + 1, 1).Text = unAchat.qte.ToString
49.       GridAchats(i + 1, 2).Text = unAchat.article.prix.ToString
50.       GridAchats(i + 1, 3).Text = (unAchat.qte * unAchat.article.prix).ToString
51.       With GridAchats(i + 1, 4)
52.         .Text = "Retirer"
53.         .Font = police
54.       End With
55.     Next
56.     ' affichage montant à payer
57.     LabelMontantPanier.Text = String.Format("Montant du panier : {0} euros", monPanier.totalPanier)
58.     ' affichage parent
```

```
59.     MyBase.affiche()
60.   End Sub
61.
62.   ' retirer un achat
63.   Private Sub GridAchats_Click(ByVal Row As Integer, ByVal Col As Integer, ByVal Header As
   LameGrid.HeaderInfo) Handles GridAchats.CellClick
64.     ' on ne s'intéresse qu'à la colonne 4 [Retirer] et pas à la ligne 0
65.     If Col <> 4 or Row = 0 Then Exit Sub
66.     ' on note l'ID de l'article cliqué
67.     session.idArticle = CType(session.panier.achats(Row - 1), Achat).article.id
68.     ' on exécute l'action
69.     MyBase.exécuteAction("retirerachat")
70.   End Sub
71.
72. End Class
73.End Namespace
```

- la classe [VuePanier] dérive de [BaseVueAppli] la classe de base pour toutes les vues de l'application - lignes 7-8
- les composants de la vue sont définis lignes 13-15. On ne gère que les événements du composant [GridAchats].
- la méthode [affiche] (lignes 23-60) gère l'affichage des composants propres au formulaire [VuePanier]. Elle commence par nettoyer le formulaire en supprimant toutes les lignes que pouvait avoir le composant [GridAchats] (lignes 25-27). Ceci fait elle
    - récupère dans la session le panier à afficher - ligne 30
    - régénère [GridAchats] avec le contenu du panier - lignes 32-55
    - affiche le montant à payer - ligne 57
    - affiche la classe de base - ligne 59
- le formulaire ne gère qu'un événement : le clic sur la colonne [Retirer] du composant [GridAchats]. Celui-ci est géré lignes 63-70. On note l'identifiant de l'article retiré et on le met dans la session (ligne 67). Puis, on demande l'exécution de l'action asynchrone "retirerachat" (ligne 69). Le contrôleur va alors prendre la main.

6.6.5 La vue [VuePanierVide]

Rappelons l'aspect visuel ce cette vue :



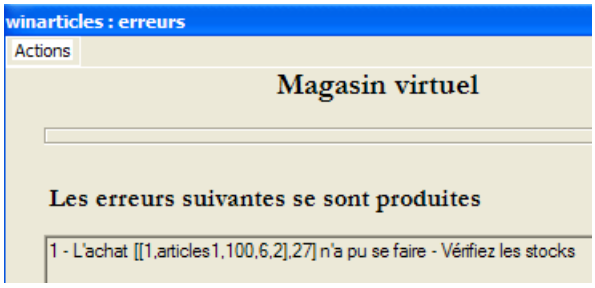
Le code de la classe [VuePanierVide] est le suivant :

```
1. Imports System.Windows.Forms
2.
3. Namespace istia.st.cmv.magasin
4.  Public Class Vue_PanierVide
5.    Inherits BaseVueAppli
6.
7. #Region " Code généré par le Concepteur Windows Form "
8. ...
9.     Me.Label2 = New System.Windows.Forms.Label
10....
11.#End Region
12.
13.   ' constructeur
14.   Public Overrides Sub affiche()
15.     ' affichage parent
16.     MyBase.affiche()
17.   End Sub
18.
19. End Class
20.End Namespace
```

- la classe [VuePanierVide] dérive de [BaseVueAppli] la classe de base pour toutes les vues de l'application - lignes 4-5
- l'unique composant de la vue est défini ligne 9.
- la méthode [affiche] (lignes 14-17) se contente de demander l'affichage de la classe parent [BaseVueArticle] pour hériter du menu.

6.6.6 La vue [VueErreurs]

Rappelons l'aspect visuel ce cette vue :



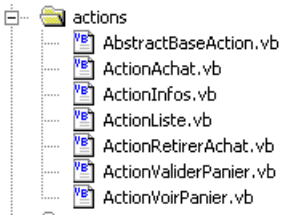
Le code de la classe [VueErreurs] est le suivant :

```
1. Imports System.Windows.Forms
2.
3. Namespace istia.st.cmv.magasin
4.
5.  Public Class Vue_Erreurs
6.    Inherits BaseVueAppli
7.
8. #Region " Code généré par le Concepteur Windows Form "
9. ....
10.   Friend WithEvents TextBoxErreurs As System.Windows.Forms.TextBox
11.   Friend WithEvents Label2 As System.Windows.Forms.Label
12....
13.#End Region
14.
15.   ' affichage formulaire
16.   Public Overrides Sub affiche()
17.     ' affichage erreurs
18.     Dim erreurs As ArrayList = session.erreurs
19.     TextBoxErreurs.Text = ""
20.     For i As Integer = 0 To erreurs.Count - 1
21.       TextBoxErreurs.Text += (i + 1).ToString + " - " + erreurs(i).ToString + vbCrLf + vbCrLf
22.     Next
23.     ' affichage parent
24.     MyBase.affiche()
25.   End Sub
26.
27. End Class
28.End Namespace
```

- la classe [VueErreurs] dérive de [BaseVueAppli] la classe de base pour toutes les vues de l'application - lignes 5-6
- les composants de la vue sont définis lignes 10-11. Il n'y a aucun événement à gérer.
- la méthode [affiche] (lignes 16-25) gère l'affichage des composants propres au formulaire [VueErreurs]. Elle
  - récupère dans la session la liste des erreurs à afficher - ligne 18
  - les affiche - lignes 19-22
  - affiche la classe de base - ligne 24

6.7 Les actions

Elles sont implémentées par les classes suivantes :



6.7.1 L'action [AbstractBaseAction]

Le code de [AbstractBaseAction] est le suivant :

```
1. Imports istia.st.articles.domain
2.
3. Namespace istia.st.cmv.magasin
4.  Public MustInherit Class AbstractBaseAction
5.    Implements IAction
6.
7.    ' la session des objets communs
8.    Private _session As session
9.
10.   Public Property session() As Session
11.     Get
12.       Return _session
13.     End Get
14.     Set(ByVal Value As session)
15.       _session = Value
16.     End Set
17.   End Property
18.
19.   ' la méthode execute sera implémentée dans les classes dérivées
20.   Public MustOverride Function execute() As String Implements IAction.execute
21. End Class
22.End Namespace
```

- la classe implémente l'interface [IAction] (ligne 5) une interface du moteur [M2VC-win]. C'est obligatoire.
- elle doit donc implémenter la méthode [execute] de cette interface. C'est fait ligne 20. On ne sait pas quoi exécuter. Seules les classes dérivées le sauront. La méthode [execute] est donc marquée abstraite (MustOverride) ce qui entraîne que la classe est elle-même abstraite (attribut MustInherit, ligne 4). On rappelle qu'une classe abstaite est une classe qu'on doit obligatoirement dériver pour en avoir des instances.
- nous avons dit que la communication [actions-vues] se faisait via un unique objet [Session] partagé par tous les objets [actions-vues]. L'objet [Session] sera injecté dans [AbstractBaseAction] grâce à l'attribut public [session] (lignes 8-17).

6.7.2 L'action [ActionListe]

Cette action se produit à deux moments :
- lorsque l'application démarre
- lorsque l'utilisateur clique sur l'option [Liste] d'une vue

Elle a pour rôle de mettre dans la session la liste des articles en vente.

Le code de [ActionListe] est le suivant :

```
1. Imports istia.st.articles.domain
2.
3. Namespace istia.st.cmv.magasin
4.  Public Class ActionListe
5.    Inherits AbstractBaseAction
6.
7.    ' liste des articles
8.    Public Overrides Function execute() As String
9.      ' gestion des erreurs
10.     Session.erreurs = New ArrayList
11.     ' on demande la liste des articles
12.     Dim articles As IList
13.     Try
14.       Session.articles = Session.articlesDomain.getAllArticles
15.     Catch ex As Exception
16.       ' on note l'erreur
17.       Session.erreurs.Add(ex.ToString)
18.     End Try
19.     ' état application
20.     If Session.erreurs.Count <> 0 Then
21.       ' problème
22.       Return "échec"
23.     Else
24.       ' c'est bon
25.       Return "succès"
26.     End If
27.   End Function
28. End Class
29.End Namespace
```

- lignes 4-5 : la classe [ActionListe] dérive de la classe [AbstractBaseAction]

- lignes 8-27 : la classe [ActionListe] implémente la méthode [execute] que n'avait pas implémentée sa classe de base[AbstractBaseAction]. Qu'y fait on ?
- ligne 10, on prépare une liste d'erreurs vide
- ligne 14, on demande la liste des articles à la couche métier. Le service d'accès à cette couche est trouvé dans la session. Si on échoue, le message d'erreur de l'exception est mémorisé dans la liste des erreurs (ligne 17)
- si tout s'est bien passé (ligne 20), on retourne au contrôleur la chaîne " succès " (ligne 25), sinon la chaîne " échec " (ligne 22)
- la méthode [execute] a terminé son travail

6.7.3 L'action [ActionInfos]

Cette action se produit sur un clic dans la colonne [Info] de la vue [VueInfos] :

Elle a pour fonction de mettre dans la session l'article sélectionné par l'utilisateur.

Le code de [ActionInfos] est le suivant :

```
1. Imports istia.st.articles.dao
2.
3. Namespace istia.st.cmv.magasin
4.  Public Class ActionInfos
5.    Inherits AbstractBaseAction
6.
7.    Public Overrides Function execute() As String
8.      ' gestion des erreurs
9.      Session.erreurs = New ArrayList
10.     ' on demande l'article de clé id
11.     Dim unArticle As Article
12.     Try
13.       Session.article = Session.articlesDomain.getArticleById(Session.idArticle)
14.     Catch ex As Exception
15.       ' pb d'accès aux données
16.       Session.erreurs.Add("Problème d'accès aux données (" + ex.ToString + ")")
17.     End Try
18.     ' des erreurs ?
19.     If Session.erreurs.Count <> 0 Then
20.       Return "échec"
21.     End If
22.     ' a-t-on récupéré un article ?
23.     If Session.article Is Nothing Then
24.       ' l'article n'existe pas
25.       Session.erreurs.Add(String.Format("L'article d'id={0} n'existe pas", Session.idArticle))
26.       Return "échec"
27.     End If
28.     Return "succès"
29.   End Function
30. End Class
31.End Namespace
```

- lignes 4-5 : la classe dérive de la classe [AbstractBaseAction]
- lignes 8-29 : la classe implémente la méthode [execute] que n'avait pas implémentée la classe de base[AbstractBaseAction]. Qu'y fait on ?
- ligne 9, on prépare une liste d'erreurs vide
- ligne 13, on demande un article à la couche métier. Le service d'accès à cette couche est trouvé dans la session ainsi que l'identifiant de l'article recherché. Si on échoue, le message d'erreur de l'exception est mémorisé dans la liste des erreurs (ligne 16)
- si on a échoué sur une exception, on retourne au contrôleur la chaîne " échec " (ligne 20)
- si on n'a pas eu d'exception mais qu'on n'a pas eu l'article demandé, l'erreur est mémorisée dans la liste des erreurs (ligne 25) et on retourne au contrôleur la chaîne " échec " (ligne 26)
- si tout s'est bien passé, on retourne au contrôleur la chaîne "succès" - ligne 28

- la méthode [execute] a terminé son travail

6.7.4 L'action [ActionAchat]

Cette action se produit lors d'un clic sur le bouton [Acheter] de la vue [VueInfos] :

Elle a pour but d'ajouter au panier qui se trouve dans la session, l'article acheté.

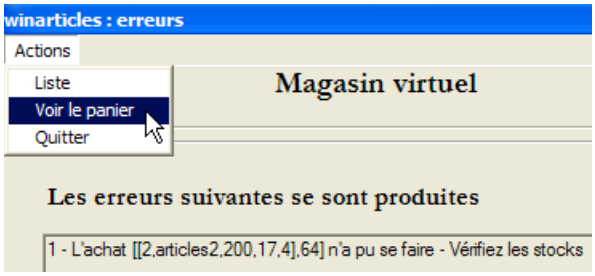
Le code de [ActionAchat] est le suivant :

```
1. Imports istia.st.articles.dao
2. Imports istia.st.articles.domain
3.
4. Namespace istia.st.cmv.magasin
5.  Public Class ActionAchat
6.    Inherits AbstractBaseAction
7.
8.    Public Overrides Function execute() As String
9.      ' on met l'achat dans le panier du client
10.     Session.panier.ajouter(New Achat(Session.article, Session.qte))
11.     Return "succès"
12.   End Function
13. End Class
14.End Namespace
```

- lignes 5-6 : la classe dérive de la classe [AbstractBaseAction]
- lignes 8-29 : la classe implémente la méthode [execute] que n'avait pas implémentée la classe de base[AbstractBaseAction]. Qu'y fait on ?
- ligne 9, on ajoute au panier (session.panier) un nouvel achat spécifiant l'article acheté (session.article) et la quantité achetée (session.qte). Tous ces éléments sont trouvés dans la session.
- ligne 11 - on retourne au contrôleur la chaîne "succès"
- la méthode [execute] a terminé son travail

6.7.5 L'action [ActionVoirPanier]

Cette action se produit lorsque l'utilisateur clique sur l'option [Voir le panier] du menu, ou lorsqu'il fait un achat ou lorsqu'il retire un achat de son panier.



Elle a pour rôle de dire si le panier est vide ou non afin qu'on sache quelle vue afficher.

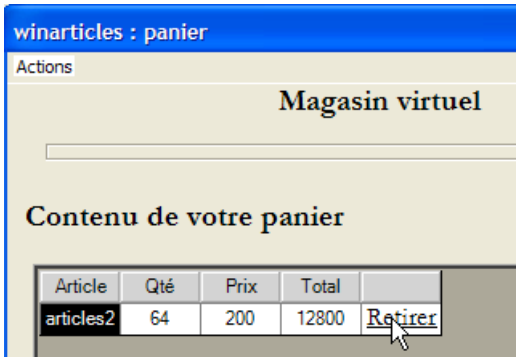
Le code de [ActionVoirPanier] est le suivant :

```
1. Namespace istia.st.cmv.magasin
2.  Public Class ActionVoirPanier
3.    Inherits AbstractBaseAction
4.
5.    Public Overrides Function execute() As String
6.      ' état
7.      If Session.panier.achats.Count = 0 Then
8.        Return "paniervide"
9.      Else
10.       Return "panier"
11.     End If
12.   End Function
13. End Class
14.End Namespace
```

- lignes 2-3 : la classe dérive de la classe [AbstractBaseAction]
- lignes 5-12 : la classe implémente la méthode [execute] que n'avait pas implémentée la classe de base[AbstractBaseAction]. Qu'y fait on ?
- ligne 7, on regarde si le panier est vide. Si oui, on retourne au contrôleur la chaîne "panierVide" (ligne 8) sinon la chaîne " panier " (ligne 10). Donc cette action se contente d'indiquer l'état du panier.
- la méthode [execute] a terminé son travail

6.7.6 L'action [ActionRetirerAchat]

Cette action se produit lorsqu'on clique sur la colonne [Retirer] d'un article de la vue [VuePanier] :



Elle a pour rôle d'enlever du panier de la session l'article ainsi désigné.

Le code de [ActionRetirerAchat] est le suivant :

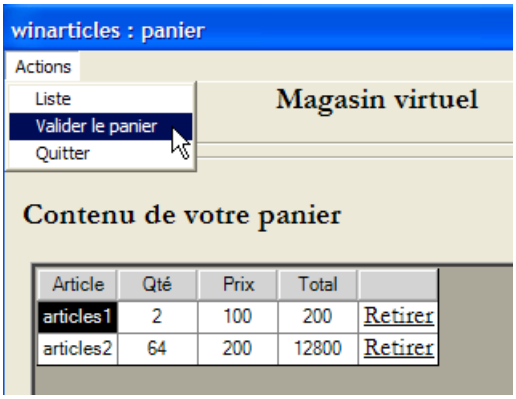
```
1. Namespace istia.st.cmv.magasin
2.
3.  Public Class ActionRetirerAchat
4.    Inherits AbstractBaseAction
5.
6.    ' retirer un achat du panier
7.    Public Overrides Function execute() As String
8.      ' on l'enlève du panier
9.      Session.panier.enlever(Session.idArticle)
10.     ' état
11.     If Session.panier.achats.Count = 0 Then
12.       Return "paniervide"
13.     Else
14.       Return "panier"
15.     End If
16.   End Function
17. End Class
18.
19.End Namespace
```

- lignes 3-4 : la classe dérive de la classe [AbstractBaseAction]
- lignes 7-17 : la classe implémente la méthode [execute] que n'avait pas implémentée la classe de base[AbstractBaseAction]. Qu'y fait on ?
- ligne 9, on enlève du panier (session.panier), l'achat d'identifiant (session.idArticle). Les informations nécessaires à l'action sont trouvées dans la session.

- ligne 11, on regarde si, après cette opération, le panier est vide. Si oui, on retourne au contrôleur la chaîne "paniervide" (ligne 12) sinon la chaîne " panier " (ligne 14).
- la méthode [execute] a terminé son travail

6.7.7 L'action [ActionValiderPanier]

Cette action se produit lorsque l'utilisateur clique sur l'option [Valider le panier] du menu :



Elle a pour but de décrémenter dans la base de données les stocks des articles achetés.

Le code de [ActionValiderPanier] est le suivant :

```
1. Namespace istia.st.cmv.magasin
2.  Public Class ActionValiderPanier
3.    Inherits AbstractBaseAction
4.
5.    ' validation du panier
6.    Public Overrides Function execute() As String
7.      ' au départ, pas d'erreurs
8.      Session.erreurs = New ArrayList
9.      ' on tente de valider le panier
10.     Try
11.       Session.articlesDomain.acheter(Session.panier)
12.       ' on note les éventuelles erreurs d'achats
13.       Session.erreurs = Session.articlesDomain.erreurs
14.     Catch ex As Exception
15.       ' on note l'erreur
16.       Session.erreurs.Add(String.Format("Erreur lors de la validation du panier [{0}]", ex.Message))
17.     End Try
18.     ' état application
19.     If Session.erreurs.Count <> 0 Then
20.       ' problème
21.       Return "échec"
22.     Else
23.       ' c'est bon
24.       Return "succès"
25.     End If
26.   End Function
27. End Class
28.
29.End Namespace
```

- lignes 2-3 : la classe dérive de la classe [AbstractBaseAction]
- lignes 6-26 : la classe implémente la méthode [execute] que n'avait pas implémentée la classe de base[AbstractBaseAction]. Qu'y fait on ?
- ligne 8, on prépare une liste d'erreurs vide
- ligne 11, on demande la validation du panier. Le service d'accès à cette couche est trouvé dans la session ainsi que le panier.
- si on échoue pour cause d'exception, le message d'erreur de l'exception est mémorisé dans la liste des erreurs (ligne 16)
- si on échoue pour cause de stocks insuffisants, la liste des erreurs rendues par la couche métier est mémorisée dans la session.
- si on a échoué (ligne 19), on retourne au contrôleur la chaîne " échec " (ligne 21), sinon " succès " (ligne 24)
- la méthode [execute] a terminé son travail

6.7.8 Conclusion

On pourra s'étonner de la grande simplicité de nos divers objets [Action]. Cette simplicité découle directement du découpage de l'application en trois couches [win, domain, dao]. La couche [win] d'interface avec l'utilisateur ne s'occupe que du dialogue avec lcelui-ci. Elle délègue tout le reste du travail au modèle de l'application, modèle implémenté ici par les couches [domain, dao].

6.8 Le contrôleur [Controleur]

Le moteur [M2VC-win] vient avec un contrôleur [BaseControleur] qui peut être dérivé. On le fait lorsqu'on veut implémenter la méthode [BaseControleur.initVue] qui par défaut ne fait rien :

```
1.    ' préparation d'une vue
2.    Protected Overridable Sub initVue(ByVal actionName As String, ByVal état As String, ByVal nomVue As
   String)
3.      ' à faire dans les classes dérivées
4.    End Sub
```

Cette méthode attend trois paramètres :

`actionName`	le nom de l'action asynchrone qui vient d'être exécutée
`état`	la chaîne de caractères que cette action a rendue au contrôleur
`nomVue`	le nom de la vue que s'apprête à afficher le contrôleur

L'idée derrière [initVue] est qu'une action n'a pas à savoir quelle vue va être affichée. Elle rend simplement une chaîne de caractères pour dire comment les choses se sont passées pour elle. Elle a mis dans la session des informations qui probablement seront utilisées par la vue que le contrôleur s'apprête à afficher. Le contrôleur lui sait quelle vue afficher mais il ne connaît pas les informations dont elle a besoin. On peut imaginer que, dans certains cas, l'action n'a pas mis dans la session toutes les informations dont a besoin la vue. Seul le développeur de l'application peut alors compléter celles-ci. Il le fera en dérivant [BaseControleur] et en redéfinissant la méthode [initVue].

Dans notre application, toutes les vues ont le même menu de base dont on n'affiche que certains éléments selon la vue. On ne voit pas bien pourquoi une action devrait s'occuper de ce menu. Elle pourrait le faire, puisque elle est écrite par un développeur qui lui sait quelle vue va être affichée. On peut ne pas trouver cela très " propre ". Pour l'exemple, on va donc dériver [BaseControleur] pour pouvoir gérer le menu des vues dans la méthode [initVue]. Le code du nouveau contrôleur est le suivant :

```
1. Namespace istia.st.cmv.magasin
2.  Public Class Controleur
3.    Inherits BaseControleur
4.
5.    ' la session de l'application
6.    Private _session As session
7.
8.    Public Property session() As session
9.      Get
10.       Return _session
11.     End Get
12.     Set(ByVal Value As session)
13.       _session = Value
14.     End Set
15.   End Property
16.
17.   ' init vues
18.   Protected Overrides Sub initVue(ByVal action As String, ByVal état As String, ByVal vue As String)
19.     ' on fixe les options de menu de la vue à afficher
20.     Select Case action
21.       Case "voirpanier"
22.         Select Case état
23.           Case "panier"
24.             enable(True, False, True, True)
25.           Case "paniervide"
26.             enable(True, False, False, True)
27.         End Select
28.       Case "achat"
29.         enable(True, False, True, True)
30.       Case "infos"
31.         Select Case état
32.           Case "succès"
33.             enable(True, False, False, True)
34.             session.message=""
35.           Case "échec"
36.             enable(True, False, False, True)
37.         End Select
38.       Case "liste"
39.         Select Case état
40.           Case "succès"
41.             enable(False, True, False, True)
42.           Case "échec"
43.             enable(True, False, False, True)
44.         End Select
```

```
45.      Case "retirerachat"
46.        Select Case état
47.          Case "panier"
48.            enable(True, False, True, True)
49.          Case "paniervide"
50.            enable(True, False, False, True)
51.        End Select
52.      Case "validerpanier"
53.        Select Case état
54.          Case "échec"
55.            enable(True, True, False, True)
56.          Case "succès"
57.            enable(False, True, False, True)
58.            session.message = "Validation réussie !"
59.        End Select
60.    End Select
61.  End Sub
62.
63.  Private Sub enable(ByVal etatMenuListe As Boolean, ByVal etatMenuVoirPanier As Boolean, ByVal
   etatMenuValiderPanier As Boolean, ByVal etatMenuQuitter As Boolean)
64.    ' on fixe l'état des options du menu
65.    With session
66.      .etatMenuListe = etatMenuListe
67.      .etatMenuVoirPanier = etatMenuVoirPanier
68.      .etatMenuValiderPanier = etatMenuValiderPanier
69.      .etatMenuQuitter = etatMenuQuitter
70.    End With
71.  End Sub
72. End Class
73.
74.End Namespace
```

- la classe [Controleur] dérive de [BaseControleur], lignes 2-3
- on injectera dans le contrôleur l'objet [Session] déjà partagé entre les actions et les vues - lignes 6-15
- lignes 18-61, on redéfinit la méthode [initVue] de [BaseControleur]. Rappelons que cette méthode est exécutée après qu'une action asynchrone se soit terminée et avant que la vue associée au résultat de celle-ci ne soit affichée.
- la méthode traite chaque action et pour celle-ci chacun de ses résultats possibles pour fixer les options du menu de la vue qui va être affichée. L'état visible ou non des quatre options du menu [MenuListe, MenuVoirPanier, MenuValiderPanier, MenuQuitter] est fixé par la méthode privée [enable] (lignes 63-71). L'état des quatre options est placé dans la session afin que la vue puisse l'y trouver. C'est l'une des raisons qui nécessitaient la présence de l'objet [Session] dans le contrôleur.
- lignes 56-58, on traite le cas du succès de la validation du panier. On sait que l'action [ActionValiderPanier] rend la chaîne " succès " si elle a réussi la validation. On peut estimer que c'est bien suffisant et que c'est au développeur de dire dans [initVue] qu'il veut voir le message " Validation réussie " sur la vue [VueListe] qui va être affichée (ligne 57).

La méthode [initVue] peut rendre les actions indépendantes des vues et c'est une bonne chose.

6.9 Le fichier de configuration

Il nous reste à écrire le fichier de configuration [m2vc-win.exe.config]. Cela nécessite de bien comprendre les règles de [Spring Ioc]. Le fichier de configurations de notre application [winarticles] est riche :

```
1. <?xml version="1.0" encoding="iso-8859-1" ?>
2. <configuration>
3.  <configSections>
4.    <sectionGroup name="spring">
5.      <section name="context" type="Spring.Context.Support.ContextHandler, Spring.Core" />
6.      <section name="objects" type="Spring.Context.Support.DefaultSectionHandler, Spring.Core" />
7.    </sectionGroup>
8.  </configSections>
9.  <spring>
10.   <context type="Spring.Context.Support.XmlApplicationContext, Spring.Core">
11.     <resource uri="config://spring/objects" />
12.   </context>
13.   <objects>
14.     <!-- la synchro -->
15.     <object id="synchro" type="System.Threading.AutoResetEvent,Mscorlib">
16.       <constructor-arg index="0">
17.         <value>true</value>
18.       </constructor-arg>
19.     </object>
20.     <!-- objets métier -->
21.     <object id="articlesDao" type="istia.st.articles.dao.ArticlesDaoSqlMap, webarticles-dao" />
22.     <object id="articlesDomain" type="istia.st.articles.domain.AchatsArticles, webarticles-domain">
23.       <constructor-arg index="0">
24.         <ref object="articlesDao" />
25.       </constructor-arg>
26.     </object>
27.     <object id="panier" type="istia.st.articles.domain.Panier, webarticles-domain" />
28.     <!-- les vues -->
```

```
29.     <object id="vueErreurs" type="istia.st.cmv.magasin.Vue_Erreurs, winarticles">
30.       <property name="nom">
31.         <value>erreurs</value>
32.       </property>
33.       <property name="session">
34.         <ref object="session" />
35.       </property>
36.       <property name="synchro">
37.         <ref object="synchro" />
38.       </property>
39.     </object>
40.     <object id="vueListe" type="istia.st.cmv.magasin.Vue_Liste, winarticles">
41.       <property name="nom">
42.         <value>liste</value>
43.       </property>
44.       <property name="session">
45.         <ref object="session" />
46.       </property>
47.       <property name="synchro">
48.         <ref object="synchro" />
49.       </property>
50.     </object>
51.     <object id="vueInfos" type="istia.st.cmv.magasin.Vue_Infos, winarticles">
52.       <property name="nom">
53.         <value>infos</value>
54.       </property>
55.       <property name="session">
56.         <ref object="session" />
57.       </property>
58.       <property name="synchro">
59.         <ref object="synchro" />
60.       </property>
61.     </object>
62.     <object id="vuePanier" type="istia.st.cmv.magasin.Vue_Panier, winarticles">
63.       <property name="nom">
64.         <value>panier</value>
65.       </property>
66.       <property name="session">
67.         <ref object="session" />
68.       </property>
69.       <property name="synchro">
70.         <ref object="synchro" />
71.       </property>
72.     </object>
73.     <object id="vuePanierVide" type="istia.st.cmv.magasin.Vue_PanierVide, winarticles">
74.       <property name="nom">
75.         <value>paniervide</value>
76.       </property>
77.       <property name="session">
78.         <ref object="session" />
79.       </property>
80.       <property name="synchro">
81.         <ref object="synchro" />
82.       </property>
83.     </object>
84.     <!-- la session -->
85.     <object id="session" type="istia.st.cmv.magasin.Session, winarticles">
86.       <property name="articlesDomain">
87.         <ref object="articlesDomain" />
88.       </property>
89.       <property name="panier">
90.         <ref object="panier" />
91.       </property>
92.     </object>
93.     <!-- les actions -->
94.     <object id="actionListe" type="istia.st.cmv.magasin.ActionListe, winarticles">
95.       <property name="session">
96.         <ref object="session" />
97.       </property>
98.     </object>
99.     <object id="actionInfos" type="istia.st.cmv.magasin.ActionInfos, winarticles">
100.      <property name="session">
101.        <ref object="session" />
102.      </property>
103.    </object>
104.    <object id="actionAchat" type="istia.st.cmv.magasin.ActionAchat, winarticles">
105.      <property name="session">
106.        <ref object="session" />
107.      </property>
108.    </object>
109.    <object id="actionVoirPanier" type="istia.st.cmv.magasin.ActionVoirPanier, winarticles">
110.      <property name="session">
111.        <ref object="session" />
112.      </property>
113.    </object>
114.    <object id="actionRetirerAchat" type="istia.st.cmv.magasin.ActionRetirerAchat, winarticles">
115.      <property name="session">
```

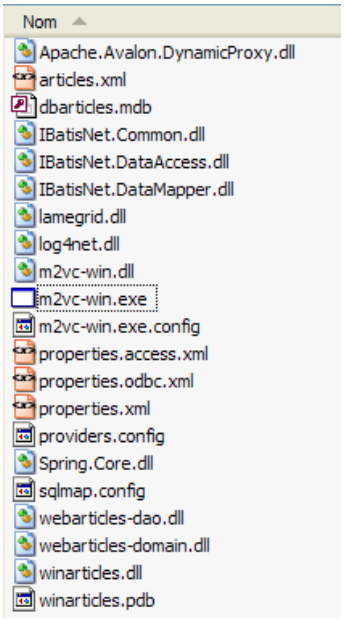
```
116.        <ref object="session" />
117.      </property>
118.    </object>
119.    <object id="actionValiderPanier" type="istia.st.cmv.magasin.ActionValiderPanier, winarticles">
120.      <property name="session">
121.        <ref object="session" />
122.      </property>
123.    </object>
124.    <!-- la configuration des actions -->
125.    <object id="infosActionListe" type="istia.st.cmv.InfosAction, m2vc-win">
126.      <property name="action">
127.        <ref object="actionListe" />
128.      </property>
129.      <property name="états">
130.        <dictionary>
131.          <entry key="succès">
132.            <ref object="vueListe" />
133.          </entry>
134.          <entry key="échec">
135.            <ref object="vueErreurs" />
136.          </entry>
137.        </dictionary>
138.      </property>
139.    </object>
140.    <object id="infosActionInfos" type="istia.st.cmv.InfosAction, m2vc-win">
141.      <property name="action">
142.        <ref object="actionInfos" />
143.      </property>
144.      <property name="états">
145.        <dictionary>
146.          <entry key="succès">
147.            <ref object="vueInfos" />
148.          </entry>
149.          <entry key="échec">
150.            <ref object="vueErreurs" />
151.          </entry>
152.        </dictionary>
153.      </property>
154.    </object>
155.    <object id="infosActionAchat" type="istia.st.cmv.InfosAction, m2vc-win">
156.      <property name="action">
157.        <ref object="actionAchat" />
158.      </property>
159.      <property name="états">
160.        <dictionary>
161.          <entry key="succès">
162.            <ref object="vuePanier" />
163.          </entry>
164.        </dictionary>
165.      </property>
166.    </object>
167.    <object id="infosActionVoirPanier" type="istia.st.cmv.InfosAction, m2vc-win">
168.      <property name="action">
169.        <ref object="actionVoirPanier" />
170.      </property>
171.      <property name="états">
172.        <dictionary>
173.          <entry key="paniervide">
174.            <ref object="vuePanierVide" />
175.          </entry>
176.          <entry key="panier">
177.            <ref object="vuePanier" />
178.          </entry>
179.        </dictionary>
180.      </property>
181.    </object>
182.    <object id="infosActionRetirerAchat" type="istia.st.cmv.InfosAction, m2vc-win">
183.      <property name="action">
184.        <ref object="actionRetirerAchat" />
185.      </property>
186.      <property name="états">
187.        <dictionary>
188.          <entry key="paniervide">
189.            <ref object="vuePanierVide" />
190.          </entry>
191.          <entry key="panier">
192.            <ref object="vuePanier" />
193.          </entry>
194.        </dictionary>
195.      </property>
196.    </object>
197.    <object id="infosActionValiderPanier" type="istia.st.cmv.InfosAction, m2vc-win">
198.      <property name="action">
199.        <ref object="actionValiderPanier" />
200.      </property>
201.      <property name="états">
202.        <dictionary>
```

```
203.          <entry key="succès">
204.            <ref object="vueListe" />
205.          </entry>
206.          <entry key="échec">
207.            <ref object="vueErreurs" />
208.          </entry>
209.        </dictionary>
210.      </property>
211.    </object>
212.    <!-- le contrôleur -->
213.    <object id="controleur" type="istia.st.cmv.magasin.Controleur, winarticles">
214.      <property name="synchro">
215.        <ref object="synchro" />
216.      </property>
217.      <property name="session">
218.        <ref object="session" />
219.      </property>
220.      <property name="firstActionName">
221.        <value>liste</value>
222.      </property>
223.      <property name="lastActionName">
224.        <value>quitter</value>
225.      </property>
226.      <property name="actions">
227.        <dictionary>
228.          <entry key="liste">
229.            <ref object="infosActionListe" />
230.          </entry>
231.          <entry key="infos">
232.            <ref object="infosActionInfos" />
233.          </entry>
234.          <entry key="achat">
235.            <ref object="infosActionAchat" />
236.          </entry>
237.          <entry key="voirpanier">
238.            <ref object="infosActionVoirPanier" />
239.          </entry>
240.          <entry key="retirerachat">
241.            <ref object="infosActionRetirerAchat" />
242.          </entry>
243.          <entry key="validerpanier">
244.            <ref object="infosActionValiderPanier" />
245.          </entry>
246.        </dictionary>
247.      </property>
248.    </object>
249.  </objects>
250. </spring>
251.</configuration>
```

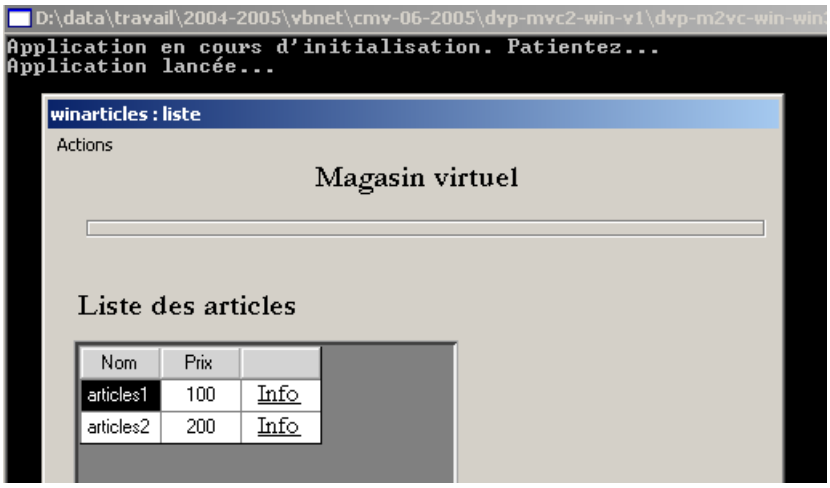
- l'objet [synchro] nécessaire à la synchronisation du contrôleur et des vues est défini lignes 15-19
- le service d'accès à la couche [dao] est défini ligne 21
- le service d'accès à la couche [domain] est défini lignes 22-26
- l'objet [Panier], objet injecté dans la session est défini ligne 27
- l'objet [Session] partagé par les actions et les vues est défini lignes 85-92. On lui injecte le panier de la ligne 27 et le service d'accès à la couche métier des lignes 22-26. Il sera injecté à son tour dans le contrôleur et chaque vue et chaque action.
- les cinq vues [VueErreurs, VueListe, VueInfos, VuePanier, VuePanierVide,] sont définies lignes 29-83. On y définit leurs attributs [nom, synchro, session].
- les six actions [ActionListe, ActionInfos, ActionAchat, ActionVoirPanier, ActionRetirerAchar, ActionValiderPanier] sont définies lignes 94-123. On injecte dans chacune d'elles l'objet [Session] défini lignes 85-92.
- les informations sur les différentes actions du contrôleur sont données lignes 125-211.
- l'objet [infosActionListe] sera associé à l'action "liste" qui demande la liste de tous les articles. Les lignes 125-139 disent les choses suivantes :
  - lignes 126-128 : que le contrôleur doit commencer par exécuter l'action [actionListe] définie lignes 94-98
  - lignes 129-137 : que sur le résultat "succès" de l'action [actionListe], la vue [VueListe] doit être affichée et que sur le résultat "échec", la vue [VueErreurs] doit être affichée.
- nous laissons le lecteur comprendre les autres objets [InfosAction] à la lumière de l'explication précédente
- le contrôleur est défini lignes 213-248. Les attributs [synchro, session, firstActionName, lastActionName] sont définis lignes 214-225, la liste des six actions contrôlées, lignes 226-247. Ainsi on lit que la première action à exécuter au démarrage du contrôleur est l'action " liste " qui vise à afficher la liste des articles et que la dernière action sera l'action " quitter ". Dans [BaseVueAppli], cette action a été associée à l'option de menu [Quitter].

6.10 Les tests

La génération du projet [winarticles] donne naissance à la DLL [winarticles.dll]. Au final, on a un dossier [bin] assez riche :



Un double-clic sur l'exécutable [m2vc-win.exe] lance l'application :



Nous invitons le lecteur à faire des tests. L'application utilise la base ACCESS [dbarticles.mdb] placé avec les exécutables dans le dossier [bin] du projet. Il n'y a donc pas à changer la configuration de test.

7 Conclusion

Nous avons transposé une application web, de complexité moyenne, mais néanmoins non triviale dans le monde windows .NET. Nous avons respecté l'architecture originelle de l'application web :

- architecture à trois couches
- indépendance des couches
- configuration des couches avec Spring
- fonctionnement MVC

Dans l'application web originelle, la couche [présentation] avait été codée sans outil. Ici, la couche [présentation] a utilisé le moteur MVC [M2VC-win], ce qui donne à l'ensemble une architecture à la " Struts "...

Table des matières