Construction en VB.NET d'une application web MVC multi-couches formée d'un client riche et d'un service web

Les idées exprimées dans ce document ont pour origine un livre lu au cours de l'été 2004, un magnifique travail de **Rod Johnson : J2EE Development without EJB** aux éditions **Wrox**.

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

1 Introduction

Nous poursuivons ici les articles :

1. [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/]
2. [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/]
3. [M2VC-win, un moteur MVC pour des applications WinForms], disponible à l'url [http://tahe.developpez.com/dotnet/m2vc-win]
4. [Construction d'une application windows à trois couches avec Spring, M2VC-win et VB.NET], disponible à l'url [http://tahe.developpez.com/dotnet/win3tier]

Rappelons que les articles 1 et 2 présentent une application simplifiée d'achats de produits sur le web, celle-ci étant un simple prétexte pour étudier un exemple d'architecture web à trois couches, couches intégrées et configurées avec la version .NET de **Spring**. L'article 3 présente un moteur MVC (Modèle - Vue - Contrôleur) appelé [M2VC-win] qui permet de construire des applications à base de formulaire WinForms avec une architecture MVC analogue à celle des applications Struts/Java. L'article 4 reprend l'application des articles 1 et 2 et l'implémente avec le moteur M2VC-win.

Ce document termine cette série de cinq articles en :

- reprenant l'application web des articles 1 et 2
- lui donnant une structure à trois couches [ui,domain, dao], celles-ci étant maintenant sur deux machines distinctes :
  - les couches [domain, dao] sont sur une machine serveur et implémentent le modèle M du MVC. On offre aux machines clientes un accès à ce modèle M via un service web (WebService)
  - la couche [ui] qui implémente le contrôleur C et les vues V du MVC est placée sur une machine cliente sous la forme d'un client riche. Celui-ci est le client à base de WinForms décrit dans l'article 4. Il repose sur le moteur [M2VC-win].

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

- [dao] : la couche implémentée par la version [sqlMap] de l'article 2
- [domain] : la couche implémentée dans l'article 1 et légèrement modifiée dans l'article 4
- [ui] : l'application WinForms implémentée dans l'article 4

Outils utilisés :

- **Visual Studio.net** pour le développement
- le serveur web **Cassini** pour le déploiement et les tests - voir annexes article 1
- **Spring** pour l'intégration et la configuration des couches de l'application web - voir annexes article 1
- **Ibatis SqlMap** pour la couche d'accès aux données du SGBD - voir annexes article 2
- le moteur **M2VC-win** de l'article 3
- l'application **win3tier** de l'article 4

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. Outre les articles cités plus haut, on pourra lire :

- langage VB.net : [http://tahe.developpez.com/dotnet/vbnet/] et en particulier le chapitre sur les services web.
- 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 initiale - Rappels

Nous présentons ici les éléments de l'application web simplifiée de commerce électronique étudiée dans les articles 1 et 2. 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 "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 de l'application

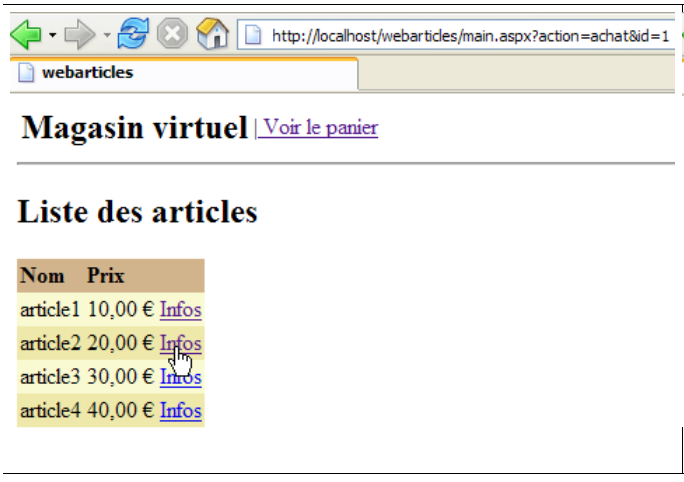
2.2 Fonctionnement de l'application web

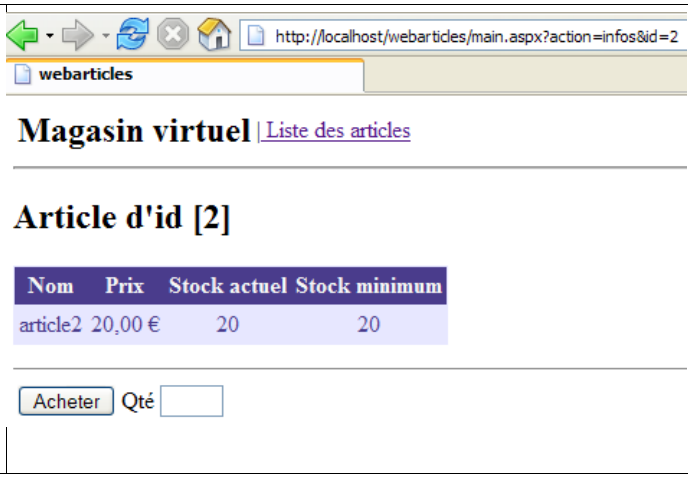
Nous présentons ci-dessous l'enchaînement des vues lors d'une utilisation typique de l'application :

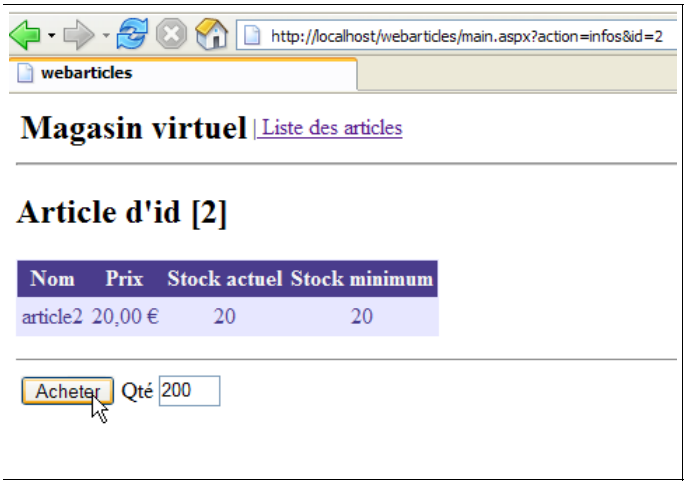
A partir de la vue ci-dessus, nous utilisons les liens du menu pour faire des opérations. En voici quelques unes. La colonne de gauche représente la demande du client et la colonne de droite la réponse qui lui est faite.


http://localhost/webarticles/main.aspx?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

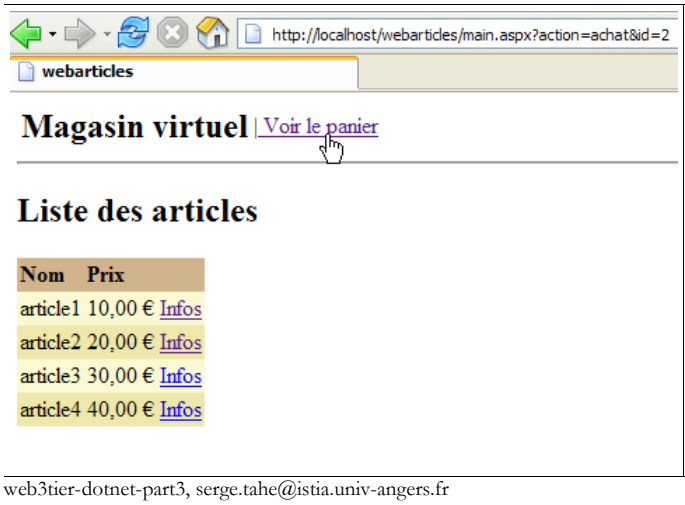

http://localhost/webarticles/main.aspx?action=achat&id=1
webarticles
Magasin virtuel | Voir le panier
Liste des articles
Nom Prix
article1 10,00 € Infos
article2 20,00 € Infos
article3 30,00 € Infos
article4 40,00 € Infos


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

Magasin virtuel | Liste des articles | Valider le panier

Contenu de votre panier

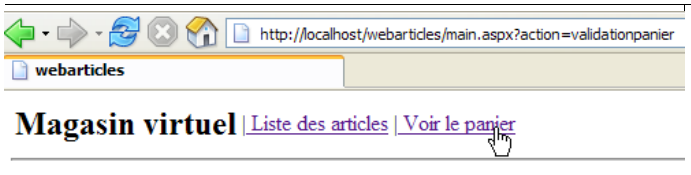
Article	Qté	Prix	Total	
article1	3	10	30,00 €	Retirer
article2	200	20	4 000,00 €	Retirer

Total de la commande : 4030 euros

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



Magasin virtuel | Liste des articles | Voir le panier

Les erreurs suivantes se sont produites :

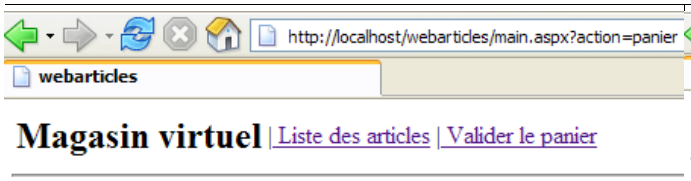
- L'achat [[2,article2,20,20,20],200] n'a pu se faire - Vérifiez les stocks

Magasin virtuel | Liste des articles | Valider le panier

Contenu de votre panier

Article	Qté	Prix	Total	
article2	200	20	4 000,00 €	Retirer

Total de la commande : 4000 euros

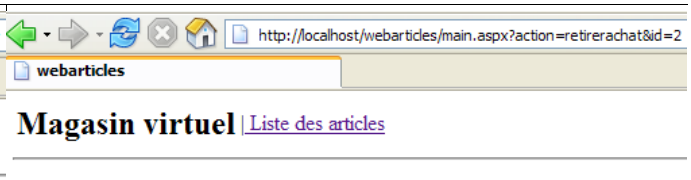


Magasin virtuel | Liste des articles | Valider le panier

Contenu de votre panier

Article	Qté	Prix	Total	
article2	200	20	4 000,00 €	Retirer

Total de la commande : 4000 euros



Magasin virtuel | Liste des articles

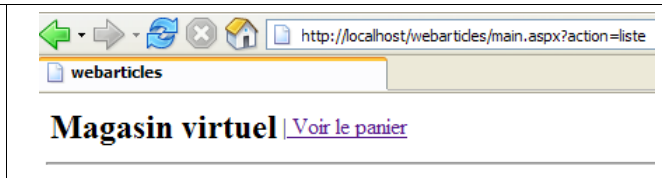
Contenu de votre panier

Votre panier est vide

Magasin virtuel | Liste des articles

Contenu de votre panier

Votre panier est vide



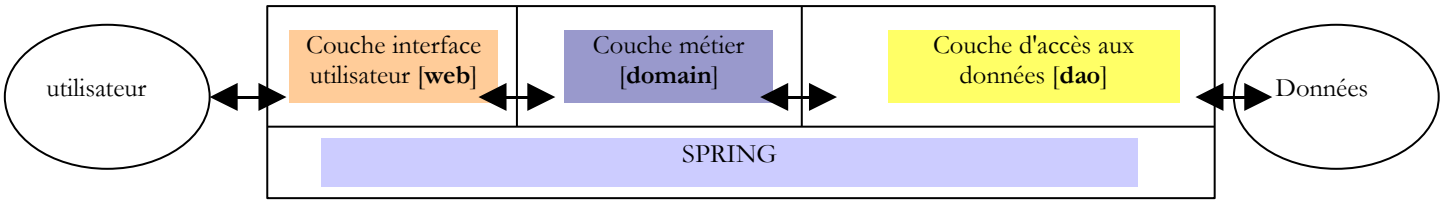
Magasin virtuel | Voir le panier

Liste des articles

Nom	Prix	
article1	10,00 €	Infos
article2	20,00 €	Infos
article3	30,00 €	Infos
article4	40,00 €	Infos

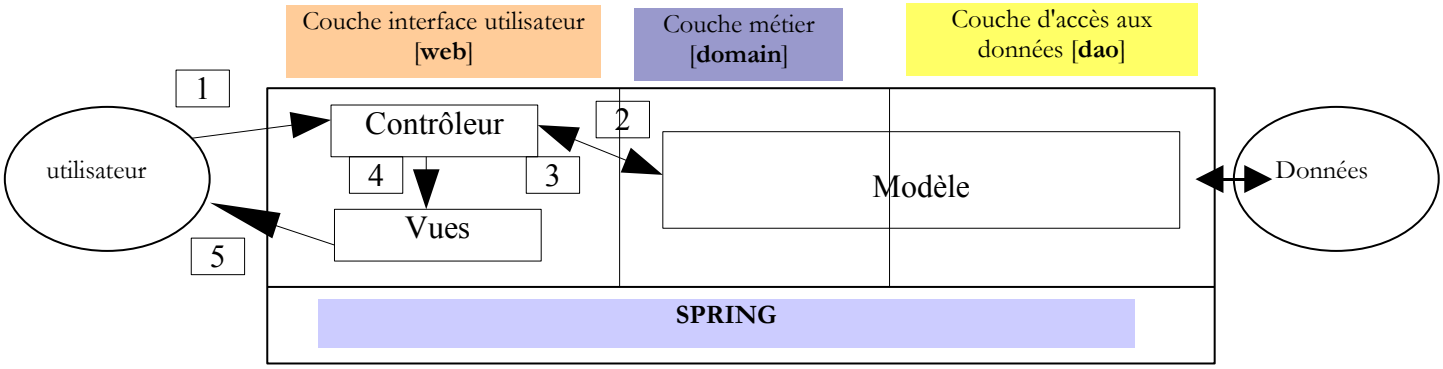
2.3 Architecture générale de l'application

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 :

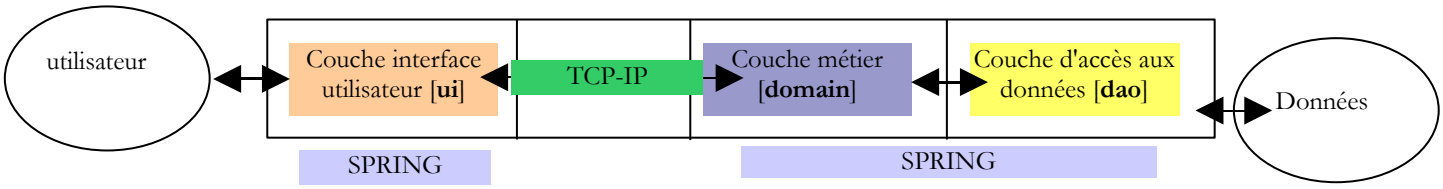
- **istia.st.articles.dao** : contient les classes d'accès aux données de la couche [dao]
- **istia.st.articles.domain** : contient les classes métier de la couche [domain]

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 - [ArticlesDaoSqlMap] : classe d'implémentation de l'interface [IArticlesDao] avec une classe utilisant la bibliothèque [Ibatis SqlMap]	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

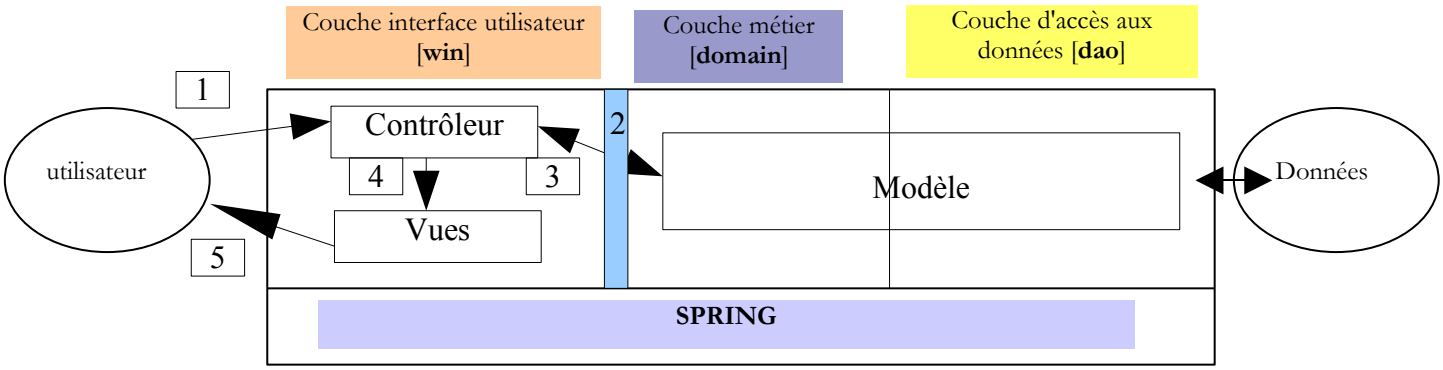
3 L'architecture de l'application web [webarticles-part3]

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 [webarticles-part3]. Elle présentera l'architecture suivante :



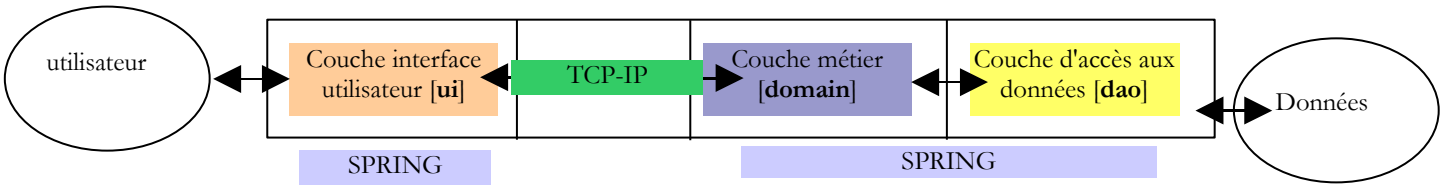
- la couche [ui] est la couche d'interface avec l'utilisateur. Elle implémente le C et le V du modèle MVC. Elle est implémentée par une application windows respectant elle-même l'architecture MVC. Elle s'appuie pour cea sur le moteur MVC [M2VC-win]
- le modèle M est implémenté par les couches [domain, dao].
- la couche [ui] n'est pas en contact direct avec le modèle M. Elle passe par le réseau pour dialoguer avec la couche [domain].
- la couche [domain] est accessible via un service web.

L'application dans son ensemble 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 page 7, peut être repris ici à l'identique. Il faut simplement se rappeler que le dialogue 2 entre les couches [ui] et [domain] se fait désormais au travers d'un réseau tcp-ip. Les couches [**dao**] et [**domain**] et [**ui**] ont déjà été construites dans les articles 1 à 4. Ce seront pour nous des boîtes noires dont nous rappelons maintenant les caractéristiques principales.

3.1 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 l'article 2, 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.

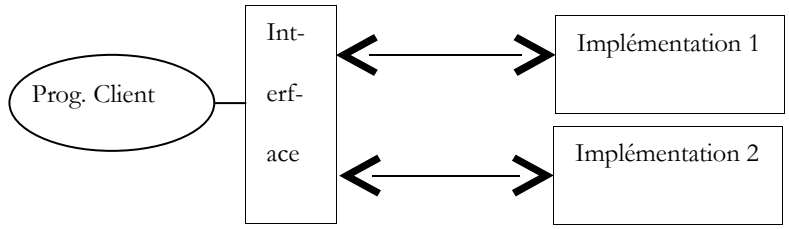
L'interface [**IArticlesDao**] est définie comme suit :

```
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 [p**roperties**] 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**

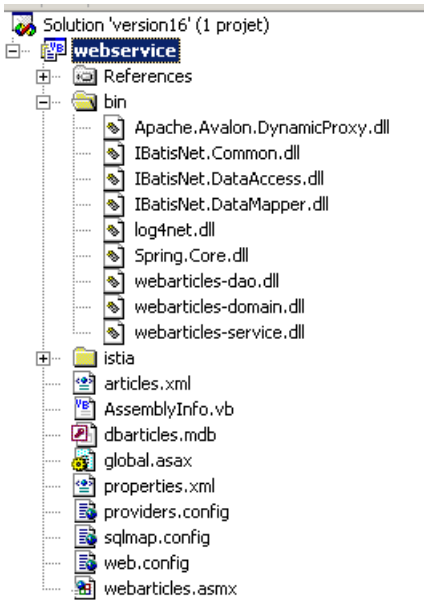
Ce fichier définit les classes d'accès aux données, classes associées à des fournisseurs d'accès :

```
<?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

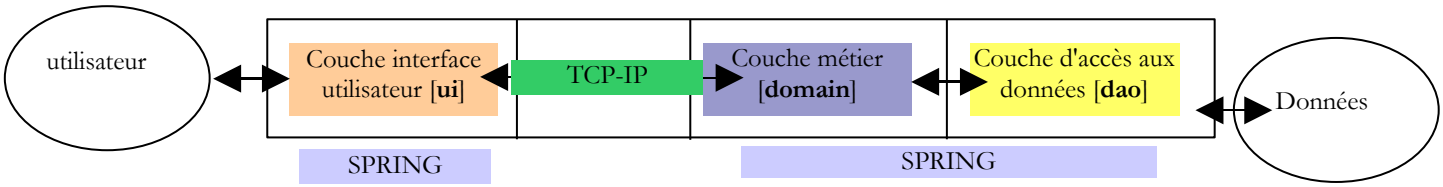
L'outil Ibatis SqlMap vient avec d'autres définitions de fournisseurs d'accès qui n'ont pas été intégrées au fichier ci-dessus. Au final, la couche [dao] va amener certains fichiers dans les dossiers du service web donnant accès aux couches [domain, dao] de l'application :



Les fichiers amenés par la couche [dao] 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.

3.2 La couche [domain]



Le lecteur est invité à relire la définition de la couche [domain] dans l'article 1 et la modification qui lui a été apportée dans l'article 4. Nous en redonnons les grandes lignes.

3.2.1 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]

3.2.2 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 lors de l'achat d'un panier - vide si pas d'erreurs

3.2.3 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

3.2.4 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

3.2.5 La classe [AchatsArticles]

L'interface [IArticlesDomain] est 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 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
```

Commentaires :

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

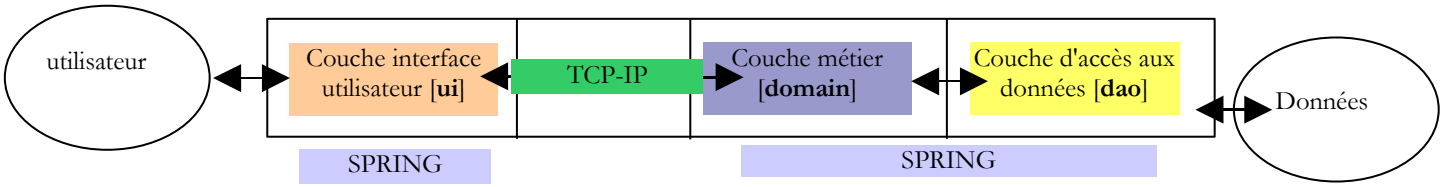
`_articlesDao As IArticlesDao` l'objet d'accès aux données
`_erreurs As ArrayList` la liste des erreurs éventuelles. Elle est accessible via la propriété publique [erreurs]

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

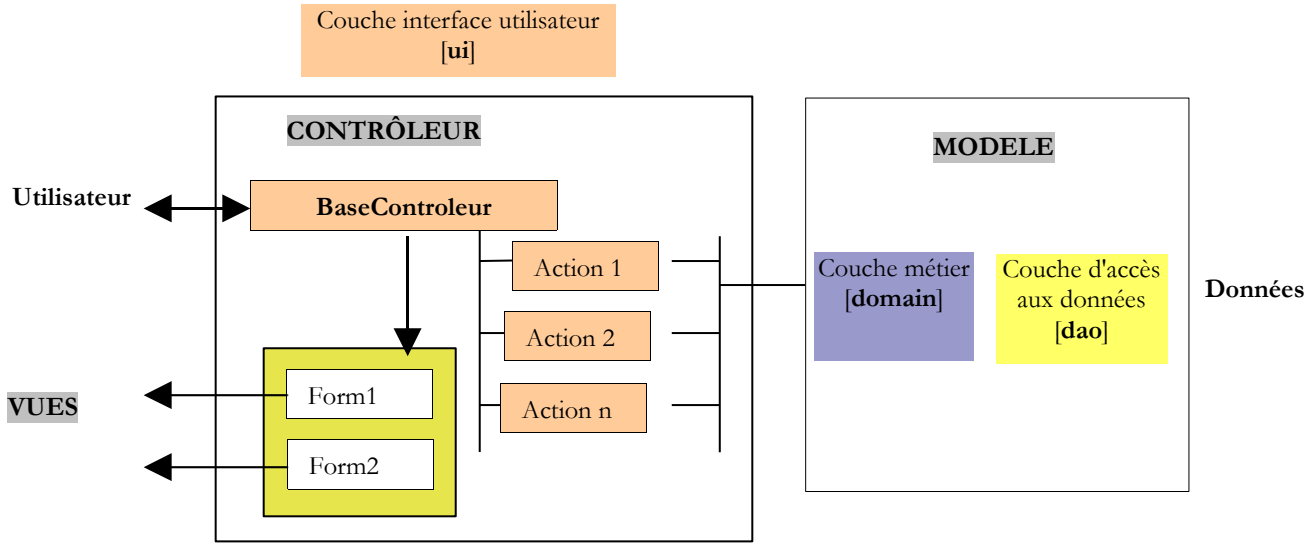
```
Sub New(ByVal articlesDao As IArticlesDao)
```

- 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.

3.3 La couche [ui]



Dans l'article 4, l'application WinForms écrite avait la structure suivante :



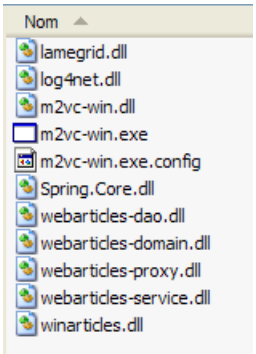
`M=modèle` les classes métier, les classes d'accès aux données et la base de données
`V=vues` les formulaires Windows

`C=contrôleur` le moteur [MVC2-win] de traitement des requêtes clientes, les objets [Action]

Rappelons les grands principes du fonctionnement de la couche [ui] écrite dans l'article 4:

1. le contrôleur [BaseControleur] est le coeur de l'application. Toutes les demandes du client transitent par lui. C'est une classe fournie par le moteur [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 à afficher selon les cas, 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 couche [ui] :



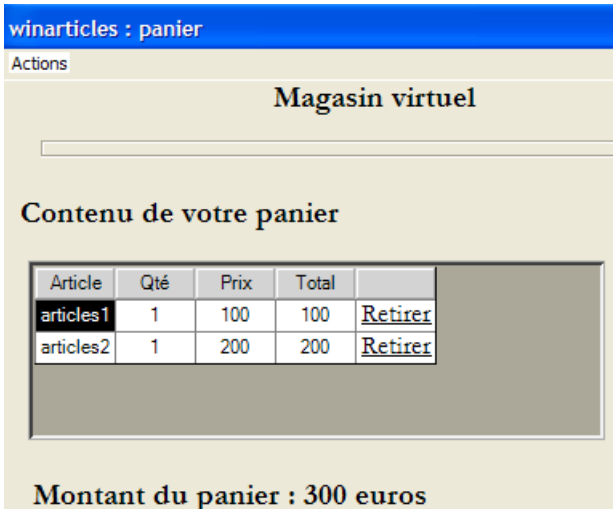
3.4 Les vues de l'application web [webarticles-part3]

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

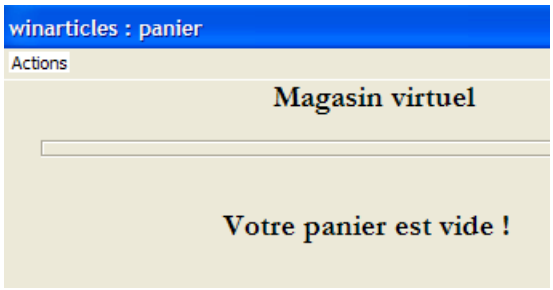
- la vue "LISTE" qui présente une liste des articles en vente

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

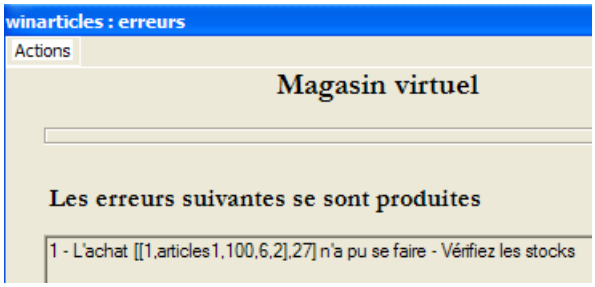
- la vue [PANIER] qui donne le contenu du panier du client



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



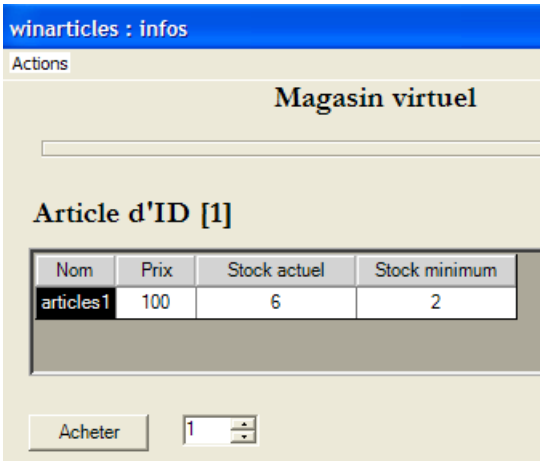
- la vue [ERREURS] qui signale toute erreur d'achat d'articles

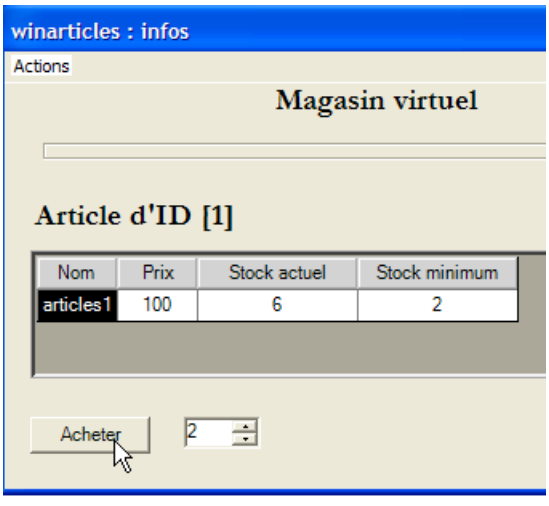


3.5 Fonctionnement de l'application web [webarticles-part3]

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.

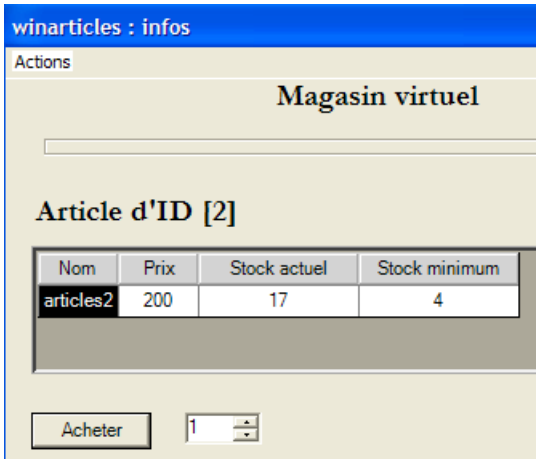


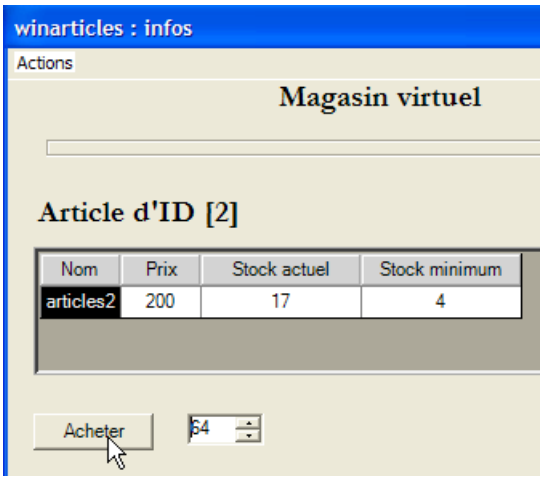



winarticles : panier
Actions
Liste
Valider le panier
Quitter
Magasin virtuel
Contenu de votre panier
Article
Qté
Prix
Total
articles1
2
100
200
Retirer

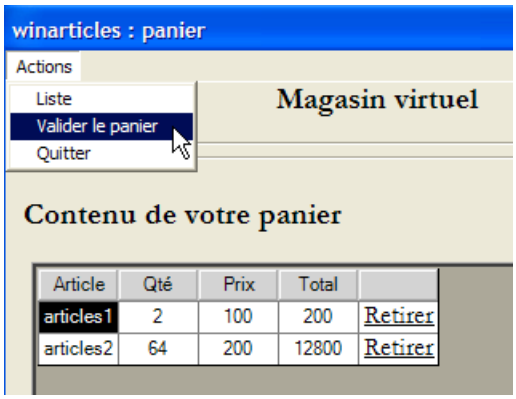

winarticles : liste
Actions
Magasin virtuel
Liste des articles
Nom
Prix
articles1
100
Info
articles2
200
Info

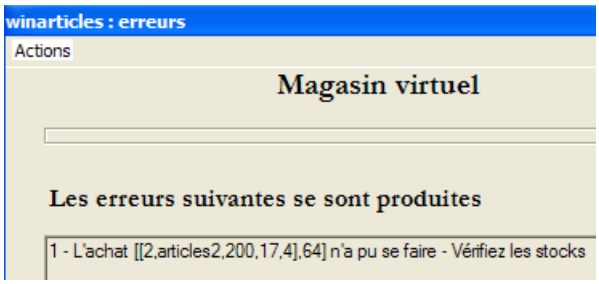

winarticles : liste
Actions
Magasin virtuel
Liste des articles
Nom
Prix
articles1
100
Info
articles2
200
Info

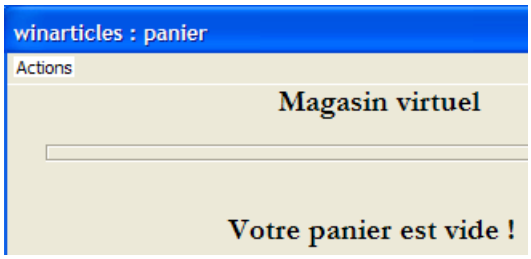

winarticles : infos
Actions
Magasin virtuel
Article d'ID [2]
Nom
Prix
Stock actuel
Stock minimum
articles2
200
17
4
Acheter
1

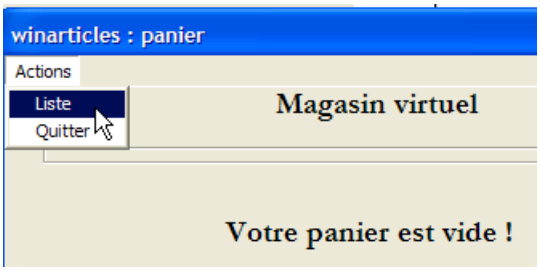

winarticles : infos
Actions
Magasin virtuel
Article d'ID [2]
Nom
Prix
Stock actuel
Stock minimum
articles2
200
17
4
Acheter
64


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



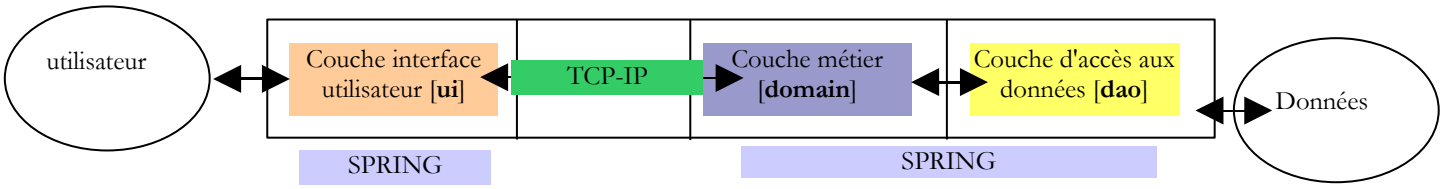




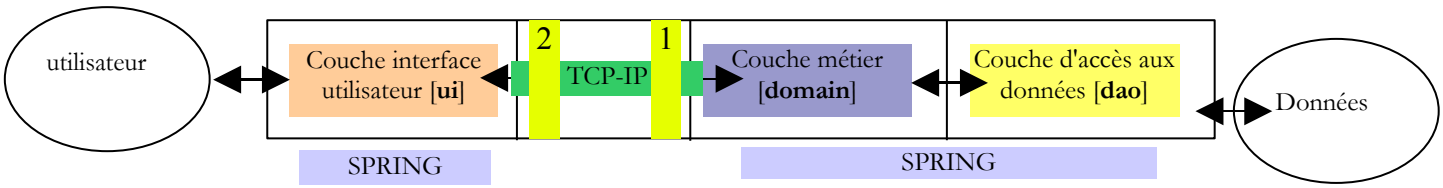


4 Les couches d'adaptation de l'application web [webarticles-part3]

Rappelons la structure de l'application [webarticles-part3] que nous voulons construire :



Nous voulons réutiliser autant que possible les couches [ui, domain, dao] développées dans les articles précédents. L'idéal serait qu'on utilise les DLL de ces couches sans toucher au code source. Il nous faut pour cela créer des couche d'adaptation entre le client et le serveur. L'architecture précédente devient alors la suivante :



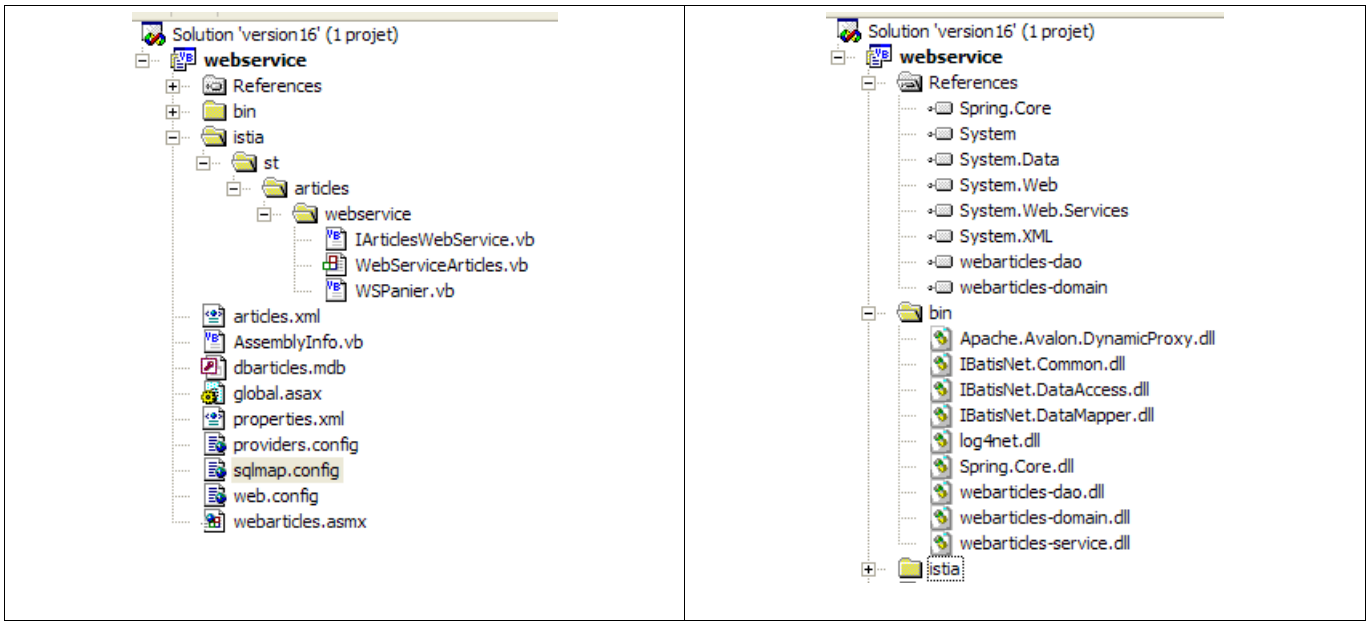
Nous ajoutons deux couches notées ci-dessus 1 et 2 :

- la couche 1 va exposer un service web d'accès à la couche [domain]. Nous l'appellerons couche [webservice].
- la couche 2 va implémenter les proxy d'accès aux services web de la couche 1. Elle le fait de façon transparente pour la couche [ui] qui ne saura pas que les données qu'elle reçoit et envoie vont sur le réseau. Nous l'appellerons couche [proxy].

5 La couche [webservice] de l'application web [webarticles-part3]

5.1 Architecture de la solution Visual Studio

L'architecture VS de la couche du service web est la suivante :

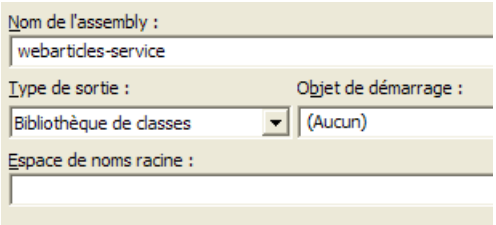


- dans [References], on trouve les deux DLL d'implémentation des couches [domain] et [dao] appelées [webarticles-domain.dll] et [webarticles-dao.dll]
- ces deux DLL sont dans le dossier [bin] du projet. Elles s'appuient elles-mêmes sur les DLL nécessaires aux outils [Ibatis SqlMap] et [Spring] qui elles-aussi sont placées dans le dossier [bin].
- toujours dans [bin], le fichier [webarticles-service.dll] est la DLL générée par le projet [webservice]. Elle contiendra le service web de [webarticles-part3].
- dans le dossier [istia/st/bin/articles/webservice], on trouve les codes source de la couche des services web. Les classes et interfaces de celle-ci sont placées dans l'espace de noms [istia.st.articles.webservice].

- directement dans le dossier du projet [webservice], on trouve les fichiers nécessaires à l'exécution du service web :

- web.config : le fichier de configuration de l'application web
- global.asax : la classe d'initialisation de l'application web
- dbarticles.mdb, articles.xml, sqlmap.config, properties.xml, providers.config : les fichiers nécessaires à la couche [dao] d'accès aux données. Celles-ci sont dans le fichier ACCESS [dbarticles.mdb].
- webarticles.asmx : le fichier du service web.

- le projet [webservice] est configuré pour générer la DLL [webarticles-service] :



Nous nous intéressons tout d'abord aux classes et interfaces de la couche [webservice].

5.2 L'interface IArticlesWebService

L'interface [IArticlesWebService] a pour but de définir les méthodes d'accès à la couche des services web. Son code est le suivant :

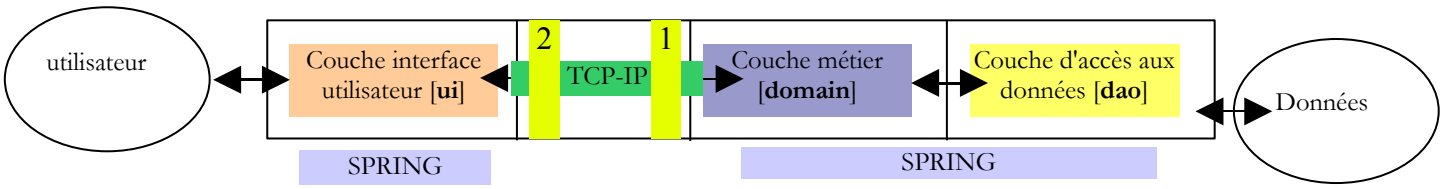
```
Imports istia.st.articles.dao
Imports istia.st.articles.domain

Namespace istia.st.articles.webservice
  Public Interface IArticlesWebService
    ' méthodes utiles
    Function getAllArticles() As IList
    Function getArticleById(ByVal idArticle As Integer) As Article
    Function acheterPanier(ByVal panier As Panier) As WSPanier
    ' méthodes de tests
    Function status() As ArrayList
    Function testAchatPanier() As WSPanier
  End Interface
End Namespace
```

- la méthode [getAllArticles] fournira la liste des articles
- la méthode [getArticleById] fournira l'article identifié par son numéro idArticle
- la méthode [acheterPanier] est la méthode qui permettra d'acheter un panier
- les méthodes [status] et [testAchatPanier] sont des méthodes à finalité de tests. Elles seront explicitées plus loin.

Ces méthodes seront toutes exposées en tant que méthodes du service web.

5.3 Un service web sans état

Reprenons l'architecture de l'application [webarticles-part3] :



Le service web implémenté dans la couche 1 est un service sans état. Il ne conserve aucune information entre deux demandes successives d'un client. Il n'y a pas la notion de Session comme dans une application web classique. Nous le verrons grâce à la méthode [status] du service web. Cette méthode n'a été implémentée que pour montrer ce point.

Quelles conséquences a pour notre application cette absence d'état ? L'absence d'état a une importance lorsque pour exécuter une opération demandée par l'utilisateur, la couche [proxy] a besoin de faire plusieurs échanges avec la couche [webservice] et que ceux-ci ont besoin d'avoir une mémoire commune gérée par le serveur. L'absence d'état du service web interdit ce mode d'échanges.

Listons les différentes actions d'un utilisateur et voyons comment celles-ci peuvent être traitées :

- **liste des articles**

L'utilisateur demande la liste des articles.

La méthode

```
    Function getAllArticles() As IList
```

du service web va être interrogée par la couche 2 [proxy]. Elle va rendre une liste d'articles éventuellement vide. Elle peut également lancer une exception si l'accès aux données n'a pu se faire. Les méthodes des services web propagent correctement les exceptions via le réseau. Aussi la couche [proxy] pourra gérer l'éventuelle exception. Il n'y a là besoin que d'un échange.

- **demande d'un article particulier**

La méthode

```
    Function getArticleById(ByVal idArticle As Integer) As Article
```

du service web va être interrogée par la couche 2 [proxy]. Elle va rendre l'article demandé s'il existe ou une exception s'il n'existe pas ou si l'accès aux données s'est révélé impossible. Il n'y a là besoin que d'un échange.

- **gestion du panier**

Autour de la gestion du panier on trouve les actions suivantes :

1. voir le panier
2. ajouter un article dans le panier
3. retirer un article du panier
4. acheter le panier

Dans les applications [webarticles] classiques précédentes, le panier d'un utilisateur était mémorisé dans la session de celui-ci. Il était ainsi conservé au fil des échanges entre le client et le serveur. Cette solution n'est pas possible avec un service web sans état. Il nous faut trouver un autre moyen pour gérer le panier du client. La solution proposée est la suivante :

- le panier sera géré par la couche [ui]. C'est donc cette couche qui détiendra l'objet [Panier] de l'utilisateur.

- les opérations 1 (voir le panier), 2 (ajouter un article) et 3 (retirer un article) ne nécessitent pas l'interrogation du service web. Elles pourront être gérées directement par la couche [ui]
- l'opération 4 (acheter le panier) est la seule opération sur le panier nécessitant l'interrogation du service web. Rappelons comment la couche [ui] achetait le panier dans les versions précédentes de [webarticles]. Elle utilisait la méthode

```
Sub acheter(ByVal panier As Panier)
```

de l'interface [IArticlesDomain]. Cette méthode ne rend aucun résultat. Le contenu du panier est modifié. Après son achat, il est soit vide, soit il contient les articles qui n'ont pu être achetés pour cause de stocks insuffisants. Dans ce cas, la couche [ui] peut avoir la liste des erreurs en faisant appel à la méthode

```
ReadOnly Property erreurs() As ArrayList
```

de l'interface [IArticlesDomain]. On voit donc que pour exécuter l'achat du panier, la couche [ui] fait deux échanges avec la couche [domain] :

1. elle demande l'achat du panier
2. elle demande la liste des erreurs mémorisées par la couche [domain] à l'issue de l'échange précédent.

Avec un service web sans état tout ceci n'est plus possible :

- à l'issue de l'échange 1, le service web connaît les éventuelles erreurs qui se sont produites mais il n'est pas capable de les mémoriser pour les donner lors de l'échange suivant. Il faut qu'il les donne à l'issue de l'échange 1. La liste des erreurs doit donc être dans la réponse du service web.
- à l'issue de l'échange 1, la couche [ui] s'attend à avoir un panier vidé des articles achetés. Dans les applications [webarticles] précédentes, les couches [ui] et [domain] travaillaient sur le même panier car la couche [ui] transmettait à la couche [domain] une **référence** du panier. Ici ce n'est plus le cas. Les échanges client-serveur se font par **valeur** et non par **référence**. La couche [ui] ne peut transmettre à la couche [domain] qu'une **copie de la valeur du panier**. Cette valeur est transmise dans un format XML. Après l'échange 1, le service web sait ce qu'est devenu le contenu du panier. Il doit donc transmettre la nouvelle valeur de celui-ci dans sa réponse. Celle-ci sera également transmise dans un format XML.

Pour tenir compte des remarques précédentes, le service web offre la méthode suivante pour l'achat d'un panier :

```
Function acheterPanier(panier As WSPanier) As WSPanier
```

- le paramètre d'entrée est le panier à acheter. Il a été envoyé par le client via le réseau.
- la réponse est de type [WSPanier] défini dans la paragraphe suivant. La réponse contient le nouveau panier après achat ainsi que la liste des éventuelles erreurs d'achats.

5.4 La classe [WSPanier]

La classe [WSPanier] définit le type de réponse envoyée par le service web pour la méthode :

```
Function acheterPanier(ByVal panier As Panier) As WSPanier
```

Son code est le suivant :

```
1. Namespace istia.st.articles.webservice
2.
3.  Public Class WSPanier
4.    Inherits istia.st.articles.domain.Panier
5.
6.    ' liste des erreurs d'achat du panier
7.    Private _erreurs As ArrayList
8.    Public Property erreurs() As ArrayList
9.      Get
10.       Return _erreurs
11.     End Get
12.     Set(ByVal Value As ArrayList)
13.       _erreurs = Value
14.     End Set
15.   End Property
16.
17. End Class
18.
19.End Namespace
```

- ligne 1 - la classe est placée dans l'espace de noms [istia.st.articles.webservice] comme tous les autres éléments de la couche [webservice]
- lignes 3-4 : la classe **WSPanier** dérive de la classe **Panier**

- lignes 7-15 : un attribut [erreurs] est défini. Il contiendra la liste des éventuelles erreurs qui se sont produites lors de l'achat d'un panier.

5.5 La classe [WebServiceArticles] d'implémentation du service web

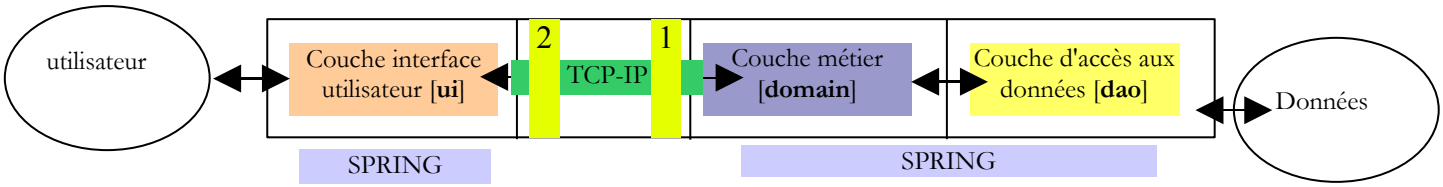
La classe [WebServiceArticles] est la classe qui va implémenter l'interface [IArticlesWebService] comme un service web. Son code est le suivant :

```
1. Imports istia.st.articles.dao
2. Imports istia.st.articles.domain
3. Imports System.Web.Services
4. Imports Spring.Context
5. Imports System.Configuration
6.
7. Namespace istia.st.articles.webservice
8.  <System.Web.Services.WebService(Namespace:="istia.st.articles.webservice"), _
9.  System.Xml.Serialization.XmlIncludeAttribute(GetType(Achat))> _
10. Public Class WebServiceArticles
11.   Inherits System.Web.Services.WebService
12.   Implements IArticlesWebService
13.
14.   'champs privés
15.   Private articlesDomain As IArticlesDomain
16.
17.   ' constructeur
18.   Public Sub New()
19.     ' classe de base
20.     MyBase.New()
21.     ' on récupère l'objet IArticlesDomain d'accès à la couche métier
22.     articlesDomain = CType(Application.Item("articlesDomain"), IArticlesDomain)
23.   End Sub
24.
25.   ' liste de tous les articles
26.   <WebMethod()> _
27.   Public Function getAllArticles() As IList Implements IArticlesWebService.getAllArticles
28.     ' liste de tous les articles
29.     Return articlesDomain.getAllArticles
30.   End Function
31.
32.   ' recherche d'un article
33.   <WebMethod()> _
34.   Public Function getArticleById(ByVal idArticle As Integer) As Article Implements
    IArticlesWebService.getArticleById
35.     ' on récupère l'article
36.     Dim article As article = articlesDomain.getArticleById(idArticle)
37.     ' article trouvé ?
38.     If article Is Nothing Then
39.       Throw New Exception(String.Format("L'article d'id [{0}] n'existe pas", idArticle))
40.     End If
41.     ' on rend l'article
42.     Return article
43.   End Function
44.
45.   ' achat du panier
46.   <WebMethod()> _
47.   Public Function acheterPanier(ByVal panier As Panier) As WSPanier Implements
    IArticlesWebService.acheterpanier
48.     ' on achète le panier
49.     articlesDomain.acheter(panier)
50.     ' on prépare le résultat
51.     Dim wspanier As New wspanier
52.     For i As Integer = 0 To panier.achats.Count - 1
53.       wspanier.ajouter(CType(panier.achats(i), Achat))
54.     Next
55.     wspanier.erreurs = articlesDomain.erreurs
56.     ' on rend le résultat
57.     Return wspanier
58.   End Function
59.
60.   <WebMethod()> _
61.   Public Function testAchatPanier() As WSPanier Implements IArticlesWebService.testAchatPanier
62.     ' le panier à acheter
63.     Dim unPanier As New Panier
64.     ' on ajoute des achats fictifs
65.     unPanier.ajouter(New Achat(New Article(1, "article1", 100, 10, 10), 2))
66.     unPanier.ajouter(New Achat(New Article(2, "article2", 200, 20, 20), 200))
67.     ' on rend le résultat
68.     Return acheterPanier(unPanier)
69.   End Function
70.
71.   ' débogage
72.   <WebMethod()> _
73.   Public Function status() As ArrayList Implements IArticlesWebService.status
```

```
74.     Dim infos As New ArrayList
75.     Dim texte As String
76.     If articlesDomain Is Nothing Then
77.       texte = "articlesDomain=nothing"
78.     Else
79.       texte = "articlesDomain<>nothing"
80.     End If
81.     infos.Add(texte)
82.     If Session Is Nothing Then
83.       texte = "Session=nothing"
84.     Else
85.       texte = "Session<>nothing"
86.     End If
87.     infos.Add(texte)
88.     Return infos
89.   End Function
90.
91. End Class
92.
93.End Namespace
```

5.5.1 L'environnement de la classe

Rappelons tout d'abord dans quelle architecture vient s'insérer la classe [WebServiceArticles] :



La classe [WebServiceArticles] se trouve dans la couche 1 appelée couche [webservice]. Elle reçoit des demandes de la couche 2 appelée couche [proxy]. La couche 2-[proxy] ne voit de la couche 1-[webservice] que son interface [IArticlesWebService] décrite paragraphe 5.2, page 22. La couche 1-[webservice] ne voit de la couche [domain] que son interface [IArticlesDomain] décrite paragraphe 3.2.2, page 13.

Commentaires du code :

- lignes 1-2 : la couche [webservice] s'appuie sur les services de la couche [domain] et utilise des types définis dans la couche [dao]. On importe donc les espaces de noms de ces deux couches.
- ligne 3 : on importe l'espace de noms nécessaire à la construction d'un service web.
- ligne 4 : on importe l'espace de noms nécessaire à Spring.
- ligne 5 : on importe l'espace de noms nécessaire à l'exploitation du fichier de configuration [web.config].
- ligne 7 : la classe est placée dans l'espace de noms [istia.st.articles.webservice].
- lignes 8-9 : deux attributs indiquant que :
  - la classe qui suit est un service web (ligne 8). Elle est placée dans un espace de noms XML nommé [istia.st.articles.webservice]. On notera que ces espaces de noms servent à éviter les conflits de noms entre services web. Même si la notion est proche, elle n'est pas identique à celle des espaces de noms des classes. Ici, rien ne nous obligeait à utiliser un espace de noms XML identique à l'espace de noms des classes.
  - que ce service a besoin d'une sérialisation XML des types [Achat] - ligne 9. Un service web échange avec ses clients des objets sur un réseau. Ces objets sont échangés dans un format XML. Lorsque l'objet est écrit sur le réseau, on parle de sérialisation XML, lorsqu'il est lu de désérialisation XML. Dans notre application les objets qui vont aller sur le réseau sont les suivants :
    - ArrayList d'objets [Article] résultat de la méthode [getAllArticles]
    - [Article] résultat de la méthode [getArticleById]
    - un entier [idArticle] paramètre de [getArticleById]
    - un objet [Panier] paramètre de la méthode [acheterPanier]
    - un objet [WSPanier] résultat de la méthode [acheterPanier]

  Parfois la définition de l'objet n'est pas assez précise pour que la plate-forme [dotnet] sache quels objets vont être réellement transportés. Ainsi l'objet [Panier] a comme attribut un champ [achats], défini comme étant de type [ArrayList]. Les éléments de cette liste seront de type [Achat] mais cela le code ne le dit pas. Dans ce cas, nous devons indiquer explicitement que le service web a besoin de sérialiser - désérialiser le type [Achat]. Lorsqu'on ne le fait pas, on a un "plantage" à l'exécution avec un message d'erreur abscons pour un néophyte.
- lignes 10-12 : on dit que la classe [WebServiceArticles] dérive de la classe prédéfinie [WebService], ce qui en fait un service web. Par ailleurs, on indique qu'elle va implémenter les méthodes de l'interface [IArticlesWebService].

5.5.2 Le constructeur

Le constructeur est défini lignes 18-23. A chaque appel de l'une des méthodes du service web, l'objet [WebServiceArticles] est reconstruit. Il doit fournir un accès à la base des articles à des clients distants. Pour cela, il a besoin d'un objet implémentant l'interface [IArticlesDomain]. Celui-ci sera construit par le fichier [global.asax] de l'application qui le stockera ensuite dans l'objet [Application]. C'est là qu'ira le chercher le service [WebServiceArticles]. L'objet [IArticlesDomain] ne sera donc construit qu'une fois.

- ligne 20 : la classe parent est construite
- ligne 22 : l'objet [IArticlesDomain] est récupéré dans l'application

5.5.3 La méthode [getAllArticles]

C'est la méthode qui permet d'obtenir la liste de tous les articles.

- ligne 29 - on se contente de demander la liste des articles à la couche [domain]

5.5.4 La méthode [getArticleById]

C'est la méthode qui permet d'obtenir un article identifié par son numéro.

- ligne 36 - on demande à l'article à la couche [domain]
- lignes 38-40 : dans le cas où l'article demandé n'existe pas, la couche [domain] rend la référence [nothing]. Si ce cas est détecté, on lance une exception indiquant que l'article demandé n'existe pas.
- ligne 42 : si tout s'est bien passé, on rend l'article

5.5.5 La méthode [acheterPanier]

C'est la méthode qui permet d'acheter un panier. On rappelle que le seul effet de cet achat est de décrémenter dans la base des articles, le stock des articles achetés des quantités achetées. Par ailleurs, si les stocks sont insuffisants pour certains articles achetés, ceux-ci restent dans le panier alors que les autres en disparaissent. En cas d'insuffisance de certains stocks, la couche [domain] ne lance pas d'exception mais rend disponible la liste des erreurs qui se sont produites, à raison d'une erreur par article n'ayant pu être acheté.

- ligne 49 : le panier est acheté
- ligne 51 : on déclare l'objet de type [WSPanier] qui va contenir le résultat envoyé au client. Le type [WSPanier] est défini paragraphe 5.4, page 24. Il doit contenir le panier original débarrassé des articles achetés ainsi que la liste des erreurs d'achats, liste qui peut être vide. C'est tout cela qui est renvoyé au client.
- lignes 52-54 : les achats encore dans le panier original sont recopiés dans l'objet [WSPanier]
- ligne 55 : les erreurs d'achats sont placées dans l'objet [WSPanier]
- ligne 57 : on renvoie le résultat au client

5.5.6 La méthode [testAchatPanier]

C'est une méthode qui va nous permettre de tester l'achat d'un panier. Il est possible d'interroger directement les méthodes d'un service web à partir d'un navigateur. Si la méthode a des paramètres, l'interrogation directe n'est possible que si ceux-ci sont de type simple, nombre ou chaînes. La méthode [acheterPanier] qui attend un paramètre de type [Panier] ne pourra être interrogée en direct. Les méthodes [getAllArticles] et [getArticleById] elles pourront l'être, car la première n'attend aucun paramètre et la seconde attend un nombre entier.

- ligne 61 - pour tester l'achat d'un panier, on crée une méthode sans paramètres [testAchatPanier].
- lignes 63-66, on crée un panier fictif de deux articles. On s'arrangera pour que le deuxième article ait des stocks insuffisants pour la demande afin de voir le mécanisme des erreurs.
- ligne 68 - maintenant qu'on a un panier, on l'achète avec la méthode [acheterPanier] déjà décrite.

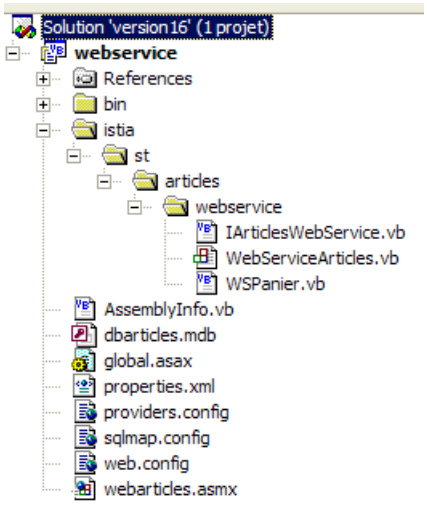
5.5.7 La méthode [status]

Cette méthode sans paramètres pourra être testée en direct avec un navigateur. Elle nous servira à vérifier deux points :

- lignes 76-81 : la valeur du champ privé [articlesDomain] qui est l'objet d'accès à la couche [domain]. On veut simplement vérifier que cet objet a bien reçu une valeur lors de la construction du service web.
- lignes 82-87 : la valeur de l'objet [Session]. On veut montrer que cet objet a pour valeur la référence [nothing] indiquant par là que le service web n'a pas de session.
- ligne 88 : ces deux informations sont mises dans un objet [ArrayList] et envoyées au client distant.

5.6 Configuration du service web

Revenons sur la structure générale de notre projet Visual Studio :



Nous venons de décrire les classes et interfaces du dossier [istia/st/articles/webservice]. Nous décrivons maintenant le reste des fichiers:

5.6.1 Le fichier [web.config]

Son contenu est le suivant :

```
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.     <object id="articlesDao" type="istia.st.articles.dao.ArticlesDaoSqlMap, webarticles-dao"/>
15.     <object id="articlesDomain" type="istia.st.articles.domain.AchatsArticles, webarticles-domain">
16.       <constructor-arg index="0">
17.         <ref object="articlesDao" />
18.       </constructor-arg>
19.     </object>
20.   </objects>
21. </spring>
22.</configuration>
```

Les lignes 9-21 sont destinées à [Spring]. Elles permettent de définir l'objet [articlesDomain] qui sera utilisé par la couche [webservice] pour accéder à la couche [domain] de l'application.

5.6.2 Le fichier [global.asax]

Ce fichier sert le plus souvent à deux choses :

1. instancier les objets qui sont partagés par tous les utilisateurs d'une application web et les mettre dans l'objet [Application]. Ceci est fait dans la méthode [Application_Start]. Cette méthode n'est excutée qu'une fois dans le cycle de vie de l'application web.
2. initialiser une session pour l'utilisateur si celui-ci en a besoin d'une. Ceci est fait dans la méthode [Session_Start]. Cette méthode est exécutée pour chaque nouveau client.

Un service web n'ayant pas la notion de session, seule la méthode [Application_Start] peut ici nous être utile. Lorsque l'une de ses méthodes est interrogée, le service web est intancié. Une fois la méthode exécutée, le service web disparaît pour être réinstancié à la

demande suivante. C'est pourquoi nous l'avons appelé un service sans état. Notre service web a besoin de l'objet [articlesDomain] défini dans le fichier [web.config]. Plutôt que de réinstancier un nouvel objet [articlesDomain] à chaque appel au service web, nous décidons de créer cet objet une unique fois dans [Application_Start].

Le code de [global.asax.vb] est le suivant :

```
1. Imports System
2. Imports System.Web
3. Imports System.Web.SessionState
4. Imports System.Configuration
5. Imports istia.st.articles.domain
6. Imports System.Collections
7. Imports Spring.Context
8.
9. Namespace istia.st.articles.webservice
10.
11. Public Class GlobalWebServiceArticles
12.   Inherits System.Web.HttpApplication
13.
14.   ' init application
15.   Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
16.     ' on crée un objet IArticlesDomain d'accès à la couche métier
17.     Dim contexte As IApplicationContext = CType(ConfigurationSettings.GetConfig("spring/context"),
   IApplicationContext)
18.     Dim articlesDomain As IArticlesDomain = CType(contexte.GetObject("articlesDomain"),
   IArticlesDomain)
19.     ' on mémorise l'objet dans l'application
20.     Application.Item("articlesDomain") = articlesDomain
21.   End Sub
22.
23. End Class
24.End Namespace
```

- la méthode [Application_Start] est définie lignes 15-21
- les lignes 17-18 demandent à Spring une référence sur l'objet " articlesDomain " défini dans le fichier [web.config] présenté précédemment. Spring instancie cet objet et en rend une référence.
- ligne 20 - cette référence est rendue disponible à tous les clients de l'application, associée à la clé "articlesDomain".

Nous rappelons ci-dessous comment le service web [WebServiceArticles] récupère cette référence lorsque lui-même est instancié :

```
  'champs privés
  Private articlesDomain As IArticlesDomain

  ' constructeur
  Public Sub New()
    ' classe de base
    MyBase.New()
    ' on récupère l'objet IArticlesDomain d'accès à la couche métier
    articlesDomain = CType(Application.Item("articlesDomain"), IArticlesDomain)
  End Sub
```

5.6.3 les fichiers de configuration de [Ibatis-SqlMap]

- **sqlmap.config** : a été décrit paragraphe 3.1, page 10
- **providers.config** : a été décrit paragraphe 3.1, page 12
- **properties.xml** : a été décrit paragraphe 3.1, page 12
- **articles.xml** : a été décrit paragraphe 3.1, 10

5.6.4 la base de données [dbarticles.mdb]

Son contenu est le suivant :

articles : Table

id	nom	prix	stockactuel	stockminimum
1	articles1	100	32	2
2	articles2	200	45	4

5.6.5 le fichier [webarticles.asmx]

Son contenu est le suivant :

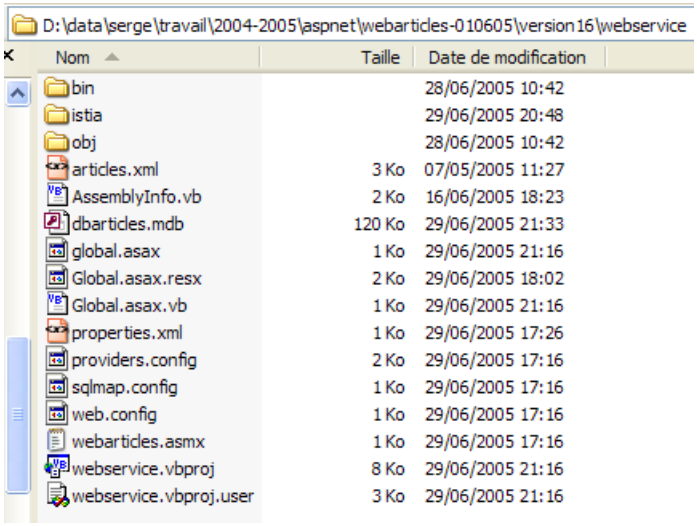
```
<%@ WebService Class="istia.st.articles.webservice.WebServiceArticles,webarticles-service" %>
```

Il n'y a donc qu'une ligne qui désigne la classe à instancier [istia.st.articles.webservice.WebServiceArticles] pour assurer le service web et la DLL dans laquelle elle se trouve (webarticles-service). Celle-ci sera recherchée dans le dossier [bin] de l'application web ainsi que tous les autres exécutables.

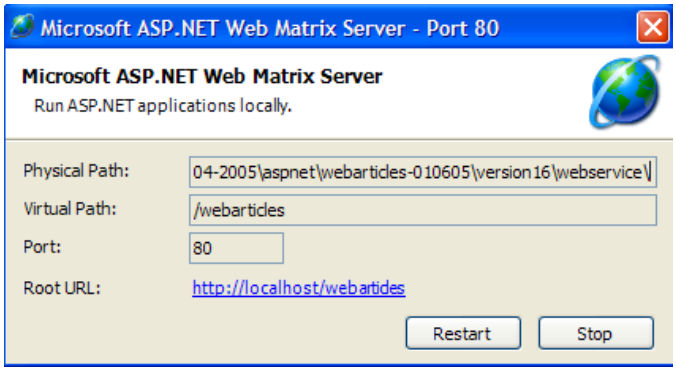
5.7 Tests de la couche [webservice]

5.7.1 Configuration du serveur web Cassini

Nous avons présenté à plusieurs reprises et sous différents angles, le dossier de notre application web. En voici un autre :



Le serveur web Cassini est configuré pour atteindre ce dossier avec l'alias /webarticles :



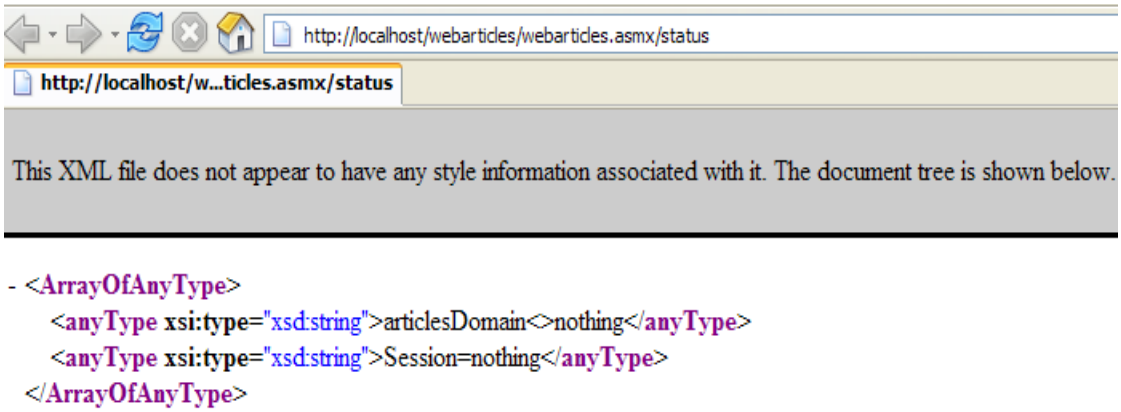
Cela signifie que le service web [webarticles.asmx] sera accessible via l'url [http://localhost/webarticles/webarticles.asmx].

5.7.2 Les tests

Nous sommes prêts pour les tests. Nous lançons Cassini configuré comme ci-dessus et demandons l'url [http://localhost/webarticles/webarticles.asmx] avec un navigateur :

Les cinq méthodes de notre service web nous sont présentées sous forme de liens. Chaque lien mène à une page de test de la méthode associée. Commençons par suivre le lien [status] :

La méthode [status] est une méthode sans paramètres. Le bouton [Appeler] fait un appel direct à la méthode [status] du service web et affiche le résultat qui lui est renvoyé. Cela donne le résultat suivant :



On notera plusieurs choses :

- l'url [http://localhost/webarticles/webarticles.asmx/status] utilisée par le navigateur. Cela signifie que la méthode [status] peut être interrogée à la main de cette façon.
- la méthode [status] a été décrite paragraphe 5.5.7, page 27. Elle rend un [ArrayList] avec deux informations sous forme de chaînes de caractères :
  - la valeur de l'objet [articlesDomain] que le service web va utiliser pour accéder aux données. On voit qu'ici la référence a bien été initialisée. On ne peut rien dire d'autre mais c'est encourageant.
  - la valeur de l'objet [Session]. Sa référence est nulle montrant que le service web n'a pas la notion de session.

Revenons à l'écran initial :

pour suivre maintenant le lien [getAllArticles]. On obtient la vue suivante :

La méthode [getAllArticles] a été décrite paragraphe 5.5.3, page 27. Elle n'attend aucun paramètre et renvoie une liste d'articles. L'utilisation du bouton [Appeler] donne les résultats suivants :


```
- <ArrayOfAnyType>
  - <anyType xsi:type="Article">
      <id>1</id>
      <nom>articles1</nom>
      <prix>100</prix>
      <stockactuel>32</stockactuel>
      <stockminimum>2</stockminimum>
    </anyType>
  - <anyType xsi:type="Article">
      <id>2</id>
      <nom>articles2</nom>
      <prix>200</prix>
      <stockactuel>45</stockactuel>
      <stockminimum>4</stockminimum>
    </anyType>
</ArrayOfAnyType>
```

Nous récupérons bien le contenu de la base [dbarticles.mdb] présenté paragraphe 5.6.4, page 29. Maintenant à partir de la vue

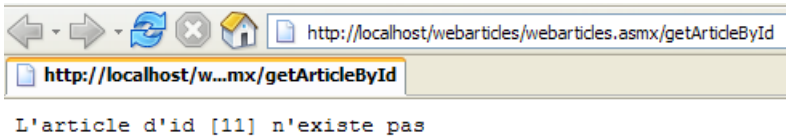
nous suivons le lien [getArticleById] :

La méthode [getArticleById] a été décrite paragraphe 5.5.4, page 27. Elle accepte un entier comme paramètre : le numéro de l'article demandé. Elle renvoie l'article demandé ou une exception si celui-ci n'existe pas. Nous prenons la configuration ci-dessus et utilisons [Appeler] pour obtenir le résultat suivant :


```
- <Article>
    <id>1</id>
    <nom>articles1</nom>
    <prix>100</prix>
    <stockactuel>32</stockactuel>
    <stockminimum>2</stockminimum>
  </Article>
```

Nous avons bien récupéré l'article n° 1. Maintenant essayons avec un numéro inexistant :

Nous obtenons le résultat suivant :



En-dehors du fait qu'on a eu une réponse qui a du sens, on peut se demander d'où elle provient. Si on regarde le code de [getArticleById], on a la réponse :

```
    ' recherche d'un article
    <WebMethod()> _
    Public Function getArticleById(ByVal idArticle As Integer) As Article Implements
IArticlesWebService.getArticleById
      ' on récupère l'article
      Dim article As article = articlesDomain.getArticleById(idArticle)
      ' article trouvé ?
      If article Is Nothing Then
        Throw New Exception(String.Format("L'article d'id [{0}] n'existe pas", idArticle))
      End If
      ' on rend l'article
      Return article
    End Function
```

Le message affiché dans la vue est le message d'erreur de l'exception lancée par la méthode [getArticleById]. Ainsi on s'aperçoit que les exceptions sont gérées proprement. La vue renvoyée au client dans ce cas, consiste à afficher le message de l'exception.

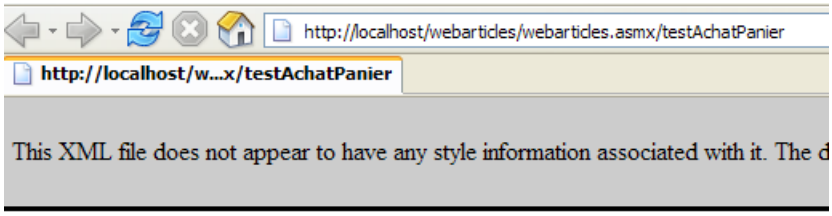
Revenons au menu initial

pour suivre le lien [testAchatPanier] :

La méthode [testAchatPanier] a été décrite au paragraphe 5.5.6, page 27. Elle n'accepte aucun paramètre :

```
    ' test achat panier
    <WebMethod()> _
    Public Function testAchatPanier() As WSPanier Implements IArticlesWebService.testAchatPanier
      ' le panier à acheter
      Dim unPanier As New Panier
      ' on ajoute des achats fictifs
      unPanier.ajouter(New Achat(New Article(1, "article1", 100, 10, 10), 2))
      unPanier.ajouter(New Achat(New Article(2, "article2", 200, 20, 20), 200))
      ' on rend le résultat
      Return acheterPanier(unPanier)
    End Function
```

Elle achète un panier composé de deux articles n° 1 et de 200 articles n° 2. Le stock de l'article 1 devrait passer à 30 (32-2). Celui de l'article 2 ne devrait pas bouger car on en demande plus qu'il n'y en a en stock. L'article n° 2 devrait donc rester dans le panier et une erreur être signalée. Ces deux informations (panier, erreurs) doivent être dans la réponse de type [WSPanier] renvoyée par la méthode. Essayons. Le résultat de l'appel est le suivant :

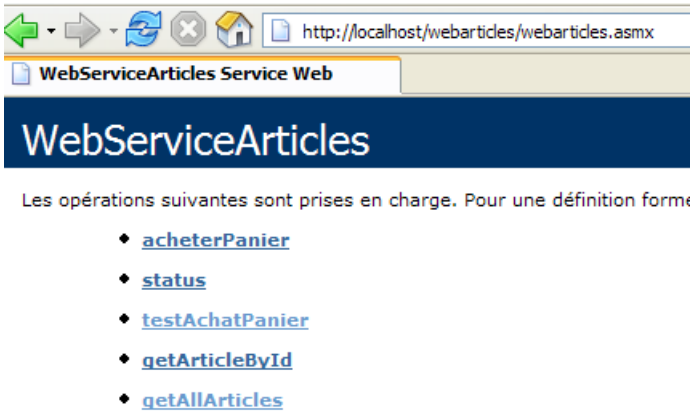



```
- <WSPanier>
  - <achats>
    - <anyType xsi:type="Achat">
      - <article>
          <id>2</id>
          <nom>article2</nom>
          <prix>200</prix>
          <stockactuel>20</stockactuel>
          <stockminimum>20</stockminimum>
        </article>
        <qte>200</qte>
      </anyType>
    </achats>
    <totalPanier>40000</totalPanier>
  - <erreurs>
    - <anyType xsi:type="xsd:string">
        L'achat [[2,article2,200,20,20],200] n'a pu se faire - Vérifiez les stocks
      </anyType>
    </erreurs>
  </WSPanier>
```

Ce qui est affiché ci-dessus est le résultat renvoyé par le service web. On y voit deux choses :

- l'article n° 2 est resté dans le panier car ses stocks étaient insuffisants
- le message d'erreur associé est bien présent lui aussi

Il nous reste à vérifier si les stocks des articles 1 et 2 sont bien 30 et 45 comme attendus. Retour au menu



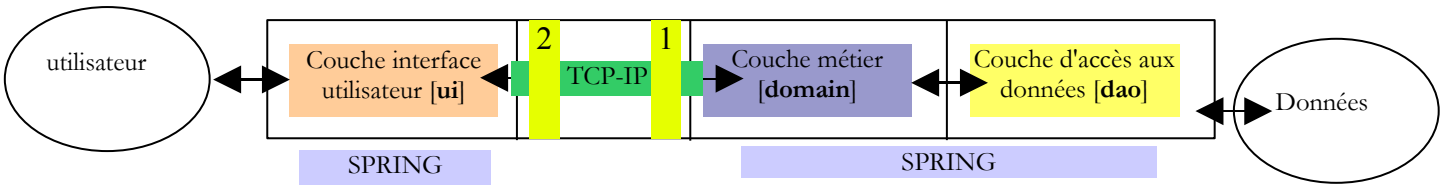
pour suivre le lien [getAllArticles] :


```
- <ArrayOfAnyType>
  - <anyType xsi:type="Article">
      <id>1</id>
      <nom>articles1</nom>
      <prix>100</prix>
      <stockactuel>30</stockactuel>
      <stockminimum>2</stockminimum>
    </anyType>
  - <anyType xsi:type="Article">
      <id>2</id>
      <nom>articles2</nom>
      <prix>200</prix>
      <stockactuel>45</stockactuel>
      <stockminimum>4</stockminimum>
    </anyType>
</ArrayOfAnyType>
```

Les stocks sont bien ceux qui étaient attendus.

5.8 Conclusion

Revenons sur l'architecture de notre application web [webarticles-part3] :



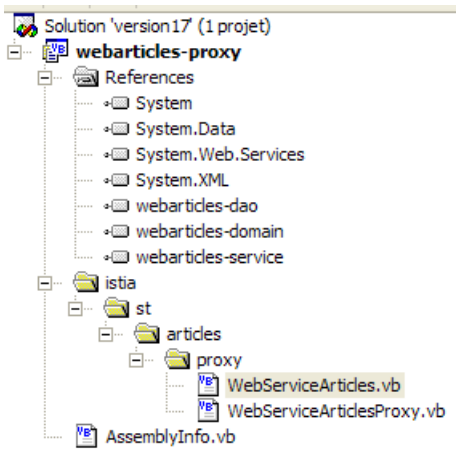
- la couche 1 expose le service web d'accès à la couche [domain]. Elle vient d'être écrite et testée. On notera que notre architecture en couches originelle nous a permis de développer la couche [1-webservice] sans toucher au code des couches [domain] et [dao]. Nous n'avons fait qu'utiliser les DLL encapsulant ces deux couches.

- la couche 2 va implémenter le proxy d'accès au service web de la couche 1. Elle le fait de façon transparente pour la couche [ui] qui ne saura pas que les données qu'elle reçoit et envoie vont sur le réseau. Nous l'avons appelée la couche [proxy] et c'est maintenant à elle que nous nous intéressons.

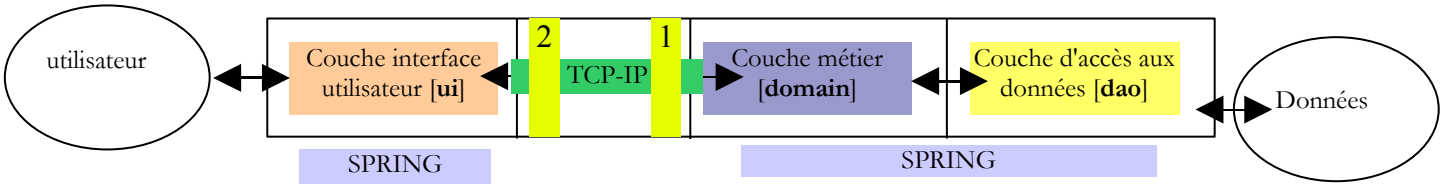
6 La couche [proxy] de l'application web [webarticles-part3]

6.1 Architecture de la solution Visual Studio

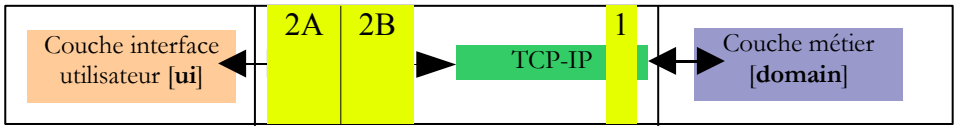
L'architecture VS de la couche [proxy] est la suivante :



Revenons sur l'architecture générale de l'application web :



- le projet VS [webarticles-proxy] va constituer la couche [2-proxy] ci-dessus.
- les références utilisées par le projet [webarticles-proxy] incluent les DLL des couches [1-webservice], [domain] et [dao]. Ci-dessus, on voit que la couche [2-proxy] dialogue avec la seule couche [1-webservice]. On peut se demander alors, pourquoi on a besoin des DLL des couches [domain] et [dao]. Elles représentent des couches normalement inconnues à la couche [2-proxy]. Cependant la couche [2-proxy] utilise des types de données définies dans ces deux couches. C'est le cas notamment des types [IArticlesDomain], [Article], [Panier]. Si donc la couche [2-proxy] n'utilise pas les services des couches [domain] et [dao], elle utilise certains de ses types. Cette réflexion suggère que l'architecture ci-dessus pourrait être améliorée en ne gardant dans les couches verticales [domain] et [dao] que des services et en ajoutant une couche horizontale d'objets partagés par toutes les couches verticales de services. La couche [2-proxy] n'aurait eu besoin alors que de deux DLL :
  - celle de la couche des objets partagés par les couches verticales
  - celle de la couche [1-webservice]
- le projet VS [webarticles-proxy] est constitué de deux classes appelées [WevServiceArticles] et [WebServiceArticlesProxy] placées toutes deux dans l'espace de noms [istia.st.articles.proxy]. Elles s'intègrent dans le schéma ci-dessus de la façon suivante :

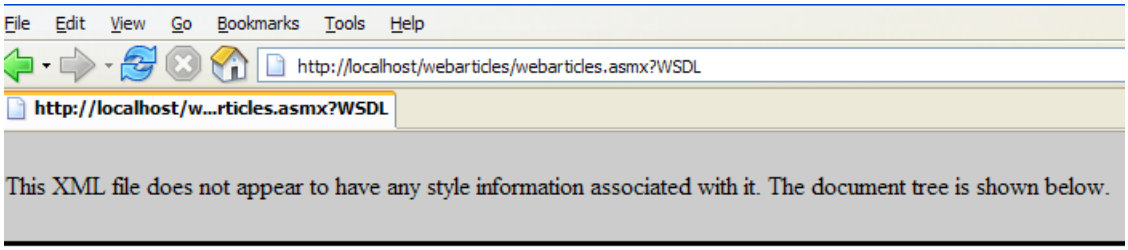


  - la couche 2B qui s'interface avec la couche [1-webservice] est implémentée par la classe [WebServiceArticles]
  - la couche 2A qui s'interface avec la couche [ui] est implémentée par la classe [WebServiceArticlesProxy]

6.2 La classe [WebServiceArticles]

Cette classe a pour rôle d'être cliente du service web que nous avons construit précédemment. L'environnement [dotnet] offre des outils pour créer le squelette d'un client d'un service web existant. On désigne souvent ce client par le terme de [proxy], aussi utiliserons-nous ce terme.

Comment générer la classe proxy d'un service web ? Un service web est toujours accompagné d'un fichier de description au format XML. Si l'URI de notre service web est [http://localhost/webarticles/webarticles.asmx], son fichier de description est disponible à l'URL [http://localhost/webarticles/webarticles.asmx***?wsdl]*** comme le montre la copie d'écran suivante :




```
- <wsdl:definitions targetNamespace="istia.st.articles.webservice">
  - <wsdl:types>
    - <s:schema elementFormDefault="qualified" targetNamespace="istia.st.articles.webservice">
      - <s:element name="testAchatPanier">
          <s:complexType/>
        </s:element>
```

Ci-dessus n'est donnée qu'une vue partielle du document de description du service web. Sa version complète pourra être trouvée en annexe page 51. Le lecteur découvrira que le document XML ci-dessus décrit précisément toutes les fonctions du service web avec pour chacune d'elles, le type et le nombre de paramètres ainsi que le type du résultat. On appelle ce fichier, le fichier WSDL du service parce qu'il utilise le langage **WSDL** (*Web Services Description Language*). A partir de ce fichier, une classe proxy peut être générée à l'aide de l'outil console *wsdl* :

```
1. dos>wsdl /language=vb http://localhost/webarticles/webarticles.asmx
2. Microsoft (R) Web Services Description Language Utility
3. [Microsoft (R) .NET Framework, Version 1.1.4322.573]
4. Copyright (C) Microsoft Corporation 1998-2002. All rights reserved.
5.
6. Écriture du fichier 'D:\temp\05-06-30\vs\WebServiceArticles.vb'.
7.
8. dos>dir
9. 30/06/2005  09:57             7 969 WebServiceArticles.vb
```

- ligne 1 : on génère le code source de la classe proxy. On précise le langage désiré (/language=vb) et l'url du service web dont on veut un proxy (http://localhost/webarticles/webarticles.asmx). L'outil [wsdl] va alors demander la description du service web à l'url [http://localhost/webarticles/webarticles.asmx?wsdl] et à partir de celle-ci générer la classe proxy.
- ligne 6 : parce que le service web est ici implémenté par une classe s'appelant [WebServiceArticles], l'outil [wsdl] génère un proxy portant ce même nom.
- lignes 8-9 : on vérifie la présence du proxy

Regardons le code du proxy généré :

```
1. '------------------------------------------------------------------------------
2. ' <autogenerated>
3. '     This code was generated by a tool.
4. '     Runtime Version: 1.1.4322.2032
5. '
6. '     Changes to this file may cause incorrect behavior and will be lost if
7. '     the code is regenerated.
8. ' </autogenerated>
9. '------------------------------------------------------------------------------
10.
11.Option Strict Off
12.Option Explicit On
13.
14.Imports System
15.Imports System.ComponentModel
16.Imports System.Diagnostics
17.Imports System.Web.Services
18.Imports System.Web.Services.Protocols
19.Imports System.Xml.Serialization
20.
21.'
22.'Ce code source a été automatiquement généré par wsdl, Version=1.1.4322.2032.
23.'
24.
25.'<remarks/>
26.<System.Diagnostics.DebuggerStepThroughAttribute(),  _
27. System.ComponentModel.DesignerCategoryAttribute("code"),  _
```

```
28. System.Web.Services.WebServiceBindingAttribute(Name:="WebServiceArticlesSoap",
   [Namespace]:="istia.st.articles.webservice"),  _
29. System.Xml.Serialization.XmlIncludeAttribute(GetType(Achat)),  _
30. System.Xml.Serialization.XmlIncludeAttribute(GetType(System.Object()))>  _
31.Public Class WebServiceArticles
32.    Inherits System.Web.Services.Protocols.SoapHttpClientProtocol
33.
34.    '<remarks/>
35.    Public Sub New()
36.        MyBase.New
37.        Me.Url = "http://localhost/webarticles/webarticles.asmx"
38.    End Sub
39.
40.    '<remarks/>
41.    <System.Web.Services.Protocols.SoapDocumentMethodAttribute("istia.st.articles.webservice/testAchatP
   anier", RequestNamespace:="istia.st.articles.webservice",
   ResponseNamespace:="istia.st.articles.webservice",
   Use:=System.Web.Services.Description.SoapBindingUse.Literal,
   ParameterStyle:=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)>  _
42.    Public Function testAchatPanier() As WSPanier
43.        Dim results() As Object = Me.Invoke("testAchatPanier", New Object(-1) {})
44.        Return CType(results(0),WSPanier)
45.    End Function
46.
47.    '<remarks/>
48.    Public Function BegintestAchatPanier(ByVal callback As System.AsyncCallback, ByVal asyncState As
   Object) As System.IAsyncResult
49.        Return Me.BeginInvoke("testAchatPanier", New Object(-1) {}, callback, asyncState)
50.    End Function
51.
52.    '<remarks/>
53.    Public Function EndtestAchatPanier(ByVal asyncResult As System.IAsyncResult) As WSPanier
54.        Dim results() As Object = Me.EndInvoke(asyncResult)
55.        Return CType(results(0),WSPanier)
56.    End Function
57.
58.    '<remarks/>
59.    <System.Web.Services.Protocols.SoapDocumentMethodAttribute("istia.st.articles.webservice/status",
   RequestNamespace:="istia.st.articles.webservice", ResponseNamespace:="istia.st.articles.webservice",
   Use:=System.Web.Services.Description.SoapBindingUse.Literal,
   ParameterStyle:=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)>  _
60.    Public Function status() As Object()
61.        Dim results() As Object = Me.Invoke("status", New Object(-1) {})
62.        Return CType(results(0),Object())
63.    End Function
64.
65.    '<remarks/>
66.    Public Function Beginstatus(ByVal callback As System.AsyncCallback, ByVal asyncState As Object) As
   System.IAsyncResult
67.        Return Me.BeginInvoke("status", New Object(-1) {}, callback, asyncState)
68.    End Function
69.
70.    '<remarks/>
71.    Public Function Endstatus(ByVal asyncResult As System.IAsyncResult) As Object()
72.        Dim results() As Object = Me.EndInvoke(asyncResult)
73.        Return CType(results(0),Object())
74.    End Function
75.
76.    '<remarks/>
77.    <System.Web.Services.Protocols.SoapDocumentMethodAttribute("istia.st.articles.webservice/acheterPan
   ier", RequestNamespace:="istia.st.articles.webservice",
   ResponseNamespace:="istia.st.articles.webservice",
   Use:=System.Web.Services.Description.SoapBindingUse.Literal,
   ParameterStyle:=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)>  _
78.    Public Function acheterPanier(ByVal panier As Panier) As WSPanier
79.        Dim results() As Object = Me.Invoke("acheterPanier", New Object() {panier})
80.        Return CType(results(0),WSPanier)
81.    End Function
82.
83.    '<remarks/>
84.    Public Function BeginacheterPanier(ByVal panier As Panier, ByVal callback As System.AsyncCallback,
   ByVal asyncState As Object) As System.IAsyncResult
85.        Return Me.BeginInvoke("acheterPanier", New Object() {panier}, callback, asyncState)
86.    End Function
87.
88.    '<remarks/>
89.    Public Function EndacheterPanier(ByVal asyncResult As System.IAsyncResult) As WSPanier
90.        Dim results() As Object = Me.EndInvoke(asyncResult)
91.        Return CType(results(0),WSPanier)
92.    End Function
93.
94.    '<remarks/>
95.    <System.Web.Services.Protocols.SoapDocumentMethodAttribute("istia.st.articles.webservice/getArticle
   ById", RequestNamespace:="istia.st.articles.webservice",
   ResponseNamespace:="istia.st.articles.webservice",
   Use:=System.Web.Services.Description.SoapBindingUse.Literal,
   ParameterStyle:=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)>  _
```

```
96.    Public Function getArticleById(ByVal idArticle As Integer) As Article
97.        Dim results() As Object = Me.Invoke("getArticleById", New Object() {idArticle})
98.        Return CType(results(0),Article)
99.    End Function
100.
101.    '<remarks/>
102.    Public Function BegingetArticleById(ByVal idArticle As Integer, ByVal callback As
   System.AsyncCallback, ByVal asyncState As Object) As System.IAsyncResult
103.        Return Me.BeginInvoke("getArticleById", New Object() {idArticle}, callback, asyncState)
104.    End Function
105.
106.    '<remarks/>
107.    Public Function EndgetArticleById(ByVal asyncResult As System.IAsyncResult) As Article
108.        Dim results() As Object = Me.EndInvoke(asyncResult)
109.        Return CType(results(0),Article)
110.    End Function
111.
112.    '<remarks/>
113.    <System.Web.Services.Protocols.SoapDocumentMethodAttribute("istia.st.articles.webservice/getAllArt
   icles", RequestNamespace:="istia.st.articles.webservice",
   ResponseNamespace:="istia.st.articles.webservice",
   Use:=System.Web.Services.Description.SoapBindingUse.Literal,
   ParameterStyle:=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)>  _
114.    Public Function getAllArticles() As Object()
115.        Dim results() As Object = Me.Invoke("getAllArticles", New Object(-1) {})
116.        Return CType(results(0),Object())
117.    End Function
118.
119.    '<remarks/>
120.    Public Function BegingetAllArticles(ByVal callback As System.AsyncCallback, ByVal asyncState As
   Object) As System.IAsyncResult
121.        Return Me.BeginInvoke("getAllArticles", New Object(-1) {}, callback, asyncState)
122.    End Function
123.
124.    '<remarks/>
125.    Public Function EndgetAllArticles(ByVal asyncResult As System.IAsyncResult) As Object()
126.        Dim results() As Object = Me.EndInvoke(asyncResult)
127.        Return CType(results(0),Object())
128.    End Function
129.End Class
130.
131.'<remarks/>
132.<System.Xml.Serialization.XmlTypeAttribute([Namespace]:="istia.st.articles.webservice")>  _
133.Public Class WSPanier
134.    Inherits Panier
135.
136.    '<remarks/>
137.    Public erreurs() As Object
138.End Class
139.
140.'<remarks/>
141.<System.Xml.Serialization.XmlTypeAttribute([Namespace]:="istia.st.articles.webservice")>  _
142.Public Class Article
143.
144.    '<remarks/>
145.    Public id As Integer
146.
147.    '<remarks/>
148.    Public nom As String
149.
150.    '<remarks/>
151.    Public prix As Double
152.
153.    '<remarks/>
154.    Public stockactuel As Integer
155.
156.    '<remarks/>
157.    Public stockminimum As Integer
158.End Class
159.
160.'<remarks/>
161.<System.Xml.Serialization.XmlTypeAttribute([Namespace]:="istia.st.articles.webservice")>  _
162.Public Class Achat
163.
164.    '<remarks/>
165.    Public article As Article
166.
167.    '<remarks/>
168.    Public qte As Integer
169.End Class
170.
171.'<remarks/>
172.<System.Xml.Serialization.XmlTypeAttribute([Namespace]:="istia.st.articles.webservice"),  _
173. System.Xml.Serialization.XmlIncludeAttribute(GetType(WSPanier))>  _
174.Public Class Panier
175.
176.    '<remarks/>
```

```
177.    Public achats() As Object
178.
179.    '<remarks/>
180.    Public totalPanier As Double
181.End Class
```

Il n'est pas indispensable de comprendre l'intégralité de ce code. Nous allons nous attarder sur les points les plus importants avant de modifier le code pour satisfaire nos besoins.

- ligne 31 : la classe proxy porte le nom de la classe du service web associé
- ligne 32 : elle dérive de la classe [System.Web.Services.Protocols.SoapHttpClientProtocol]. C'est cette classe qui gère les échanges tcp-ip avec le service web. Nous n'aurons pas à nous en occuper.
- lignes 35-38 : nous trouvons le constructeur de la classe. On voit que l'adresse du service web y est codée en dur. Nous serons amenés à modifier ce point.
- lignes 77-92 : nous trouvons les méthodes proxy de la méthode [acheterPanier], une méthode synchrone (lignes 78-81), deux méthodes asynchrones (lignes 84-86, 89-92). Seule la méthode synchrone sera utilisée dans notre application.
- lignes 78-81 : le code du proxy de la méthode [acheterPanier] est simple.
  - ligne 78 : la méthode proxy [acheterPanier] a la même signature que la méthode de même nom du service web
  - ligne 79 : la méthode [acheterPanier] du service web est appelée. On voit qu'elle utilise la méthode [Invoke] de la classe de base [System.Web.Services.Protocols.SoapHttpClientProtocol]. Cette méthode a deux paramètres :
    - le nom de la méthode du service web interrogée
    - un tableau d'objets contenant les paramètres nécessaires à la méthode. Ici il n'y a qu'un paramètre, le panier à acheter.
  - ligne 79 : la méthode [Invoke] rend ses résultats sous la forme d'un tableau d'objets
  - ligne 80 : le 1er élément du tableau des résultats contient le résultat de type [WSPanier] renvoyé par la méthode [acheterPanier] du service web.

- Toutes les méthodes du service web ont donné naissance à une méthode proxy analogue à celle qui vient d'être décrite pour la méthode [acheterPanier]. Nous ne nous attarderons pas à les détailler.
- lignes 131 - 180 : les objets manipulés par le service web sont décrits. On retrouve les objets [WSPanier], [Article], [Achat], [Panier].

Nous allons apporter plusieurs modifications à ce code généré :

- la classe va être placée dans l'espace de noms [istia.st.articles.proxy]
- les méthodes proxy des méthodes de test [status] et [testAchatPanier] vont être supprimées
- les définitions des classes vont être supprimées. En effet, si on les garde, elles entrent en conflit avec celles trouvées dans les DLL [articles-domain] et [articles-dao] référencées par le projet.

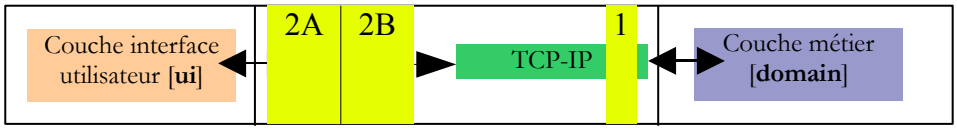
Le code épuré devient alors le suivant :

```
1. Imports System
2. Imports System.ComponentModel
3. Imports System.Diagnostics
4. Imports System.Web.Services
5. Imports System.Web.Services.Protocols
6. Imports System.Xml.Serialization
7. Imports istia.st.articles.dao
8. Imports istia.st.articles.domain
9. Imports istia.st.articles.webservice
10.
11.' proxy d'accès au service web d'achats d'articles
12.Namespace istia.st.articles.proxy
13. <System.Diagnostics.DebuggerStepThroughAttribute(), _
14.  System.ComponentModel.DesignerCategoryAttribute("code"), _
15.  System.Web.Services.WebServiceBindingAttribute(Name:="WebServiceArticlesSoap",
   [Namespace]:="istia.st.articles.webservice"), _
16.  System.Xml.Serialization.XmlIncludeAttribute(GetType(Achat)), _
17.  System.Xml.Serialization.XmlIncludeAttribute(GetType(System.Object()))> _
18. Public Class WebServiceArticles
19.   Inherits System.Web.Services.Protocols.SoapHttpClientProtocol
20.
21.   ' constructeur
22.   Public Sub New(ByVal urlWebServiceArticles As String)
23.     MyBase.New()
24.     Me.Url = urlWebServiceArticles
25.   End Sub
26.
27.   ' achat du panier
28.

   <System.Web.Services.Protocols.SoapDocumentMethodAttribute("istia.st.articles.webservice/acheterPanier"
   , RequestNamespace:="istia.st.articles.webservice", ResponseNamespace:="istia.st.articles.webservice",
   Use:=System.Web.Services.Description.SoapBindingUse.Literal,
   ParameterStyle:=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)> _
29.   Public Function acheterPanier(ByVal panier As Panier) As WSPanier
```

```
30.    Dim results() As Object = Me.Invoke("acheterPanier", New Object() {panier})
31.    Return CType(results(0), WSPanier)
32.  End Function
33.
34.  ' achat du panier
35.  Public Function BeginacheterPanier(ByVal panier As Panier, ByVal callback As System.AsyncCallback,
   ByVal asyncState As Object) As System.IAsyncResult
36.    Return Me.BeginInvoke("acheterPanier", New Object() {panier}, callback, asyncState)
37.  End Function
38.
39.  ' achat du panier
40.  Public Function EndacheterPanier(ByVal asyncResult As System.IAsyncResult) As WSPanier
41.    Dim results() As Object = Me.EndInvoke(asyncResult)
42.    Return CType(results(0), WSPanier)
43.  End Function
44.
45.  ' obtenir un article particulier
46.
   <System.Web.Services.Protocols.SoapDocumentMethodAttribute("istia.st.articles.webservice/getArticleById
   ", RequestNamespace:="istia.st.articles.webservice", ResponseNamespace:="istia.st.articles.webservice",
   Use:=System.Web.Services.Description.SoapBindingUse.Literal,
   ParameterStyle:=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)> _
47.  Public Function getArticleById(ByVal idArticle As Integer) As Article
48.    Dim results() As Object = Me.Invoke("getArticleById", New Object() {idArticle})
49.    Return CType(results(0), Article)
50.  End Function
51.
52.  ' obtenir un article particulier
53.  Public Function BegingetArticleById(ByVal idArticle As Integer, ByVal callback As
   System.AsyncCallback, ByVal asyncState As Object) As System.IAsyncResult
54.    Return Me.BeginInvoke("getArticleById", New Object() {idArticle}, callback, asyncState)
55.  End Function
56.
57.  ' obtenir un article particulier
58.  Public Function EndgetArticleById(ByVal asyncResult As System.IAsyncResult) As Article
59.    Dim results() As Object = Me.EndInvoke(asyncResult)
60.    Return CType(results(0), Article)
61.  End Function
62.
63.  ' obtenir tous les articles
64.
   <System.Web.Services.Protocols.SoapDocumentMethodAttribute("istia.st.articles.webservice/getAllArticles
   ", RequestNamespace:="istia.st.articles.webservice", ResponseNamespace:="istia.st.articles.webservice",
   Use:=System.Web.Services.Description.SoapBindingUse.Literal,
   ParameterStyle:=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)> _
65.  Public Function getAllArticles() As Object()
66.    Dim results() As Object = Me.Invoke("getAllArticles", New Object(-1) {})
67.    Return CType(results(0), Object())
68.  End Function
69.
70.  ' obtenir tous les articles
71.  Public Function BegingetAllArticles(ByVal callback As System.AsyncCallback, ByVal asyncState As
   Object) As System.IAsyncResult
72.    Return Me.BeginInvoke("getAllArticles", New Object(-1) {}, callback, asyncState)
73.  End Function
74.
75.  ' obtenir tous les articles
76.  Public Function EndgetAllArticles(ByVal asyncResult As System.IAsyncResult) As Object()
77.    Dim results() As Object = Me.EndInvoke(asyncResult)
78.    Return CType(results(0), Object())
79.  End Function
80. End Class
81.
82.End Namespace
```

- lignes 7-9 : les espaces de noms nécessaires aux objets [Article], [Panier], [Achat], [WSPanier] sont importés
- ligne 12 : la classe est placée dans l'espace de noms [istia.st.articles.proxy]
- lignes 22-25 : le constructeur reçoit désormais en paramètre l'url du service web cible
- les méthodes inutiles ont été supprimées ainsi que la définition des objets manipulés par le service web. Le reste n'a pas changé.

Revenons au schéma de notre couche [2-proxy] :



La couche [2B] implémentée par la classe [WebServiceArticles] a été écrite. Elle va nous donner un accès transparent au service web distant de la couche [1-webservice]. Il nous reste à écrire la couche [2A] qui va faire tampon entre la couche [ui] et la couche [2B]. Celle-ci va être implémentée par la classe [WebServiceArticlesProxy].

6.3 La classe [WebServiceArticlesProxy]

Pour comprendre le rôle de la couche [2A-WebServiceArticlesProxy], il faut se placer du point de vue de la couche [ui]. Celle-ci décrite dans l'article 4 de cette série d'articles s'attend à dialoguer avec une interface [IArticlesDomain]. Il faut donc que la couche [2A-WebServiceArticlesProxy] implémente cette interface. Le code de la classe est le suivant :

```
1. Imports istia.st.articles.domain
2. Imports istia.st.articles.dao
3. Imports istia.st.articles.webservice
4.
5. Namespace istia.st.articles.proxy
6.  Public Class WebServiceArticlesProxy
7.    Implements istia.st.articles.domain.IArticlesDomain
8.
9.    ' le proxy d'accès au service web
10.   Private _webServiceProxy As WebServiceArticles
11.   Public Property webServiceProxy() As WebServiceArticles
12.     Get
13.       Return _webServiceProxy
14.     End Get
15.     Set(ByVal Value As WebServiceArticles)
16.       _webServiceProxy = Value
17.     End Set
18.   End Property
19.
20.   ' les erreurs
21.   Private _erreurs As ArrayList
22.   Public ReadOnly Property erreurs() As ArrayList Implements IArticlesDomain.erreurs
23.     Get
24.       Return _erreurs
25.     End Get
26.   End Property
27.
28.   ' constructeur
29.   Public Sub New()
30.   End Sub
31.
32.   ' achat d'un panier
33.   Public Sub acheter(ByVal panier1 As Panier) Implements IArticlesDomain.acheter
34.     ' on envoie panier1 au serveur - on reÃ§oit en retour panier2
35.     Dim wsPanier2 As WSPanier = webServiceProxy.acheterPanier(panier1)
36.     ' on note les erreurs
37.     _erreurs = wsPanier2.erreurs
38.     ' on met les achats restants dans le panier original panier2 -> panier1
39.     For i As Integer = panier1.achats.Count - 1 To 0 Step -1
40.       panier1.enlever(CType(panier1.achats(i), Achat).article.id)
41.     Next
42.     For i As Integer = 0 To wsPanier2.achats.Count - 1
43.       panier1.ajouter(CType(wsPanier2.achats(i), Achat))
44.     Next
45.   End Sub
46.
47.   ' liste des articles
48.   Public Function getAllArticles() As IList Implements IArticlesDomain.getAllArticles
49.     Dim articles As New ArrayList
50.     Dim webServiceArticles As Object() = webServiceProxy.getAllArticles
51.     For i As Integer = 0 To webServiceArticles.Length - 1
52.       articles.Add(webServiceArticles(i))
53.     Next
54.     Return articles
```

```
55.   End Function
56.
57.   ' un article particulier
58.   Public Function getArticleById(ByVal idArticle As Integer) As Article Implements
   IArticlesDomain.getArticleById
59.     Return webServiceProxy.getArticleById(idArticle)
60.   End Function
61.
62. End Class
63.
64.End Namespace
```

- ligne 7 : la classe implémente bien l'interface [IArticlesDomain]
- lignes 22-26 : implémentation de la propriété [erreurs] de l'interface
- lignes 33-45 : implémentation de la méthode [acheter] de l'interface
- lignes 48-55 : implémentation de la méthode [getAllArticles] de l'interface
- lignes 58-60 : implémentation de la méthode [getArticleById] de l'interface
- lignes 9-18 : la classe [WebServiceArticlesProxy] va avoir besoin d'avoir accès au service web de la couche [1-webservice]. Elle va pour cela utiliser la classe proxy [WebServiceArticles] que nous venons de construire. La référence de cette classe proxy sera injectée par [Spring] dans l'attribut [webServiceProxy] de la classe. Le lecteur fera attention aux noms des classes qui prêtent malheureusement à confusion. Des noms plus judicieux auraient pu être choisis. Par ailleurs, il aurait été préférable que l'attribut [webServiceProxy] ait le type d'une interface plutôt que celui d'une classe. Nous laissons la classe en l'état mais elle pourrait être améliorée.
- lignes 58-60 : la méthode [getArticleById] est définie. Elle se contente d'appeler la méthode proxy [getArticleById] qui a la même signature.
- lignes 48-55 : la méthode [getAllArticles] est définie. Elle utilise la méthode proxy [getAllArticles] de même nom mais qui a une signature différente :

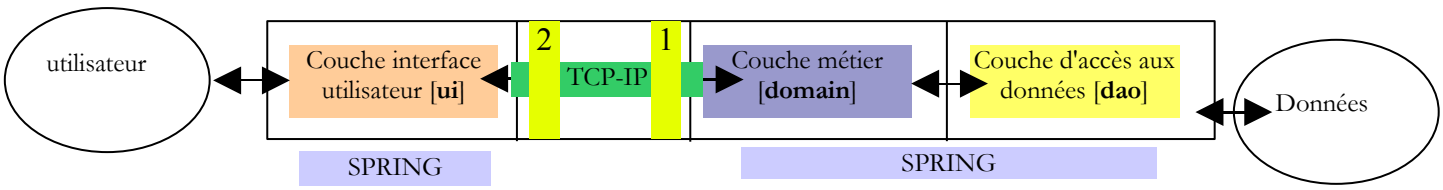
```
    Public Function getAllArticles() As Object()
```

  Au lieu de rendre un ArrayList d'objets de type [Article], elle rend un tableau d'objets [Article]. On transfère donc ce tableau dans un objet [ArrayList], lignes 51-53.
- lignes 33-45 : la méthode [acheter] est définie. Son rôle est
  - d'acheter le panier qu'on lui passe en paramètre
  - de vider le panier des articles achetés
  - de mémoriser les éventuelles erreurs d'achats, erreurs rendues disponibles par l'interface via la propriété [erreurs].
  - ligne 35 : on demande au proxy d'acheter le panier. Le résultat rendu par le proxy est un objet [WSPanier] contenant le nouveau panier ainsi que les éventuelles erreurs d'achats.
  - ligne 37 : les erreurs sont mémorisées dans un attribut privé de la classe
  - lignes 39-41 : le panier initial [panier1] est vidé de ses achats
  - lignes 42-44 : pour être rempli avec les achats restés dans le panier renvoyé par le proxy
- lignes 20-26 : la propriété [erreurs] est implémentée.

7 Tests de l'application web [webarticles-part3]

7.1 L'environnement des tests

Rappelons de nouveau l'ensemble de l'architecture de l'application [webarticles-part3] :



- nous avons une a**pplication client-serveur sur un réseau tcp-ip**
- les couches [ui] et [2-proxy] sont sur **une machine cliente C**
- les couches [1-webservice], [domain] et [dao] sont sur **une machine serveur S**

Les tests nécessitent donc :

- de lancer le service web sur **la machine serveur S**
- de lancer un client sur la **machine cliente C** et de faire des requêtes au service web

Dans les tests qui suivent, les machines S et C seront les mêmes afin de faciliter les tests.

7.1.1 Le service web

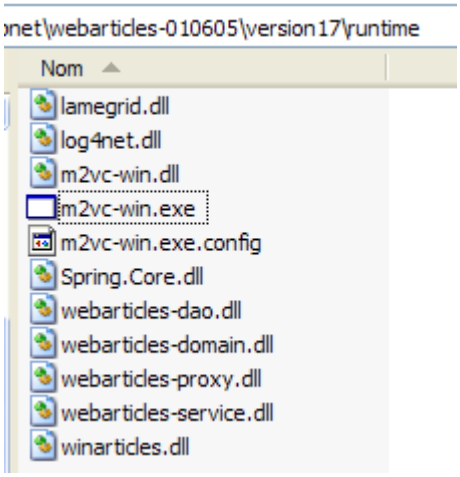
La configuration de tests utilisée est celle décrite plus haut au paragraphe 5.7, page 30.

7.1.2 Le client riche

La couche [ui] de l'architecture ci-dessus va être implémentée par l'application [winarticles] décrite dans l'article 4 de la série et décrite plus haut au paragraphe 3.3, page 15. C'est une application de type WinForms. On appelle ce type de client, un **client riche**, parce que les possibilités offertes par son interface visuelle sont supérieures à celles que peut offrir celle d'un navigateur. On a en effet une application windows classique avec toute la palette d'événements que sait gérer ce type d'applications. On pourrait ainsi gérer des déplacements de souris, ce qu'on ne sait pas faire avec une interface de type navigateur.

Rappelons que la particularité de l'application [winarticles] est d'avoir été construite autour d'un modèle MVC strict grâce au moteur MVC [M2VC-win].

Les exécutables du client sont rassemblées dans le dossier suivant :



- [**m2vc-win.exe**] : lanceur du moteur générique [M2VC-win] - utilise le fichier de configuration [**m2vc-win.exe.config**] pour configurer le client.
- [**m2vc-win.dll**] : le moteur générique MVC [**M2VC-win**] utilisé pour construire le modèle MVC du client [winarticles]
- [**Spring.core.dll, log4net.dll**] : le moteur [**M2VC-win**] s'appuie sur l'outil [**Spring**]
- [**webarticles-dao.dll**] : la couche [dao] de l'application
- [**webarticles-domain.dll**] : la couche [domain] de l'application
- [**webarticles-service.dll**] : la couche [webservice] de l'application
- [**webarticles-proxy.dll**] : la couche [proxy] de l'application
- [**winarticles.dll**] : la couche [ui] de l'application. Exécutable du client riche WinForms de notre service web.
- [**lamegrid.dll**] : un composant de type [DataGrid] utilisé par les vues du client [winarticles.dll]

Le client est configuré à l'aide du fichier [m2vc-win.exe.config]. Ce fichier a été longuement décrit dans l'article 4. Nous ne referons pas cette description ici car elle est assez complexe. Nous allons simplement nous attarder sur les modification à lui apporter.

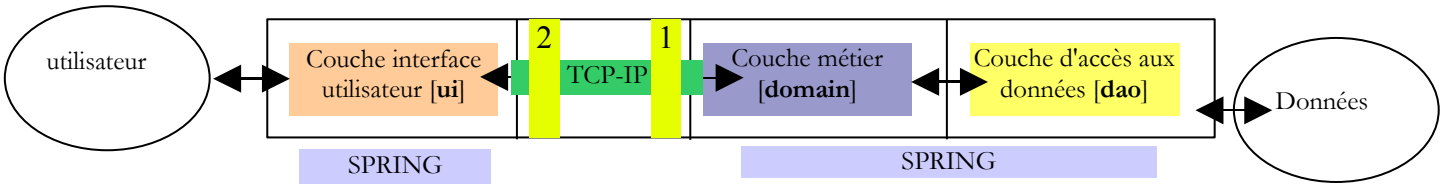
Le fichier [m2vc-win.exe.config] utilisé dans l'article 4 était le suivant :

```
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....
16.     <!-- objets métier -->
```

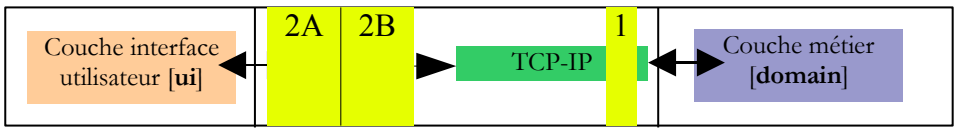
```
17.     <object id="articlesDao" type="istia.st.articles.dao.ArticlesDaoSqlMap, webarticles-dao" />
18.     <object id="articlesDomain" type="istia.st.articles.domain.AchatsArticles, webarticles-domain">
19.       <constructor-arg index="0">
20.         <ref object="articlesDao" />
21.       </constructor-arg>
22.     </object>
23.     <object id="panier" type="istia.st.articles.domain.Panier, webarticles-domain" />
24.     <!-- les vues -->
25....
26.     <!-- la session -->
27....
28.     <!-- les actions -->
29....
30.     <!-- la configuration des actions -->
31....
32.     <!-- le contrôleur -->
33....
34.   </objects>
35. </spring>
36.</configuration>
```

Sur les 250 lignes du fichier de configuration, seules les lignes 12 à 22 sont amenées à changer. Nous l'avons déjà dit, la couche [ui] de notre application, implémentée ici par [winarticles.dll] s'attend à dialoguer avec une interface [IArticlesDomain]. Dans le fichier de configuration ci-dessus, celle-ci est implémentée par l'objet [articlesDomain] de la ligne 18. Dans le contexte de l'article 4, cette objet dialoguait directement avec la couche [dao] de l'application. La ligne 17 instancie un objet [articlesDao] implémentant l'interface de la couche [dao] et les lignes 18-22 instancient un objet implémentant la couche [domain] en passant à celui-ci la référence de l'objet [articlesDao] afin que la couche [domain] puisse travailler directement avec la couche [dao].

Dans notre nouvelle architecture, la couche [ui] dialogue avec une interface [IArticlesDomain] qui n'est plus en contact avec la couche [dao] :



La couche 2 est elle-même divisée en deux couches :



La couche [ui] dialogue avec une interface [IArticlesDomain] implémentée par la classe [WebServiceArticlesProxy] de la couche [2A]. Cette classe dialogue avec la classe [WebServiceArticles] implémentée dans la couche [2B]. L'objet [articlesDomain] implémentant l'interface [IArticlesDomain] doit donc être construit avec un paramètre de type [WebServiceArticles] alors que dans l'exemple précédent, ce paramètre était de type [IArticlesDao], l'interface de la couche [dao].

Les lignes 16-23 précédentes deviennent les suivantes :

```
1.     <!-- objets metier -->
2.       <object id="proxy" type="istia.st.articles.proxy.WebServiceArticles,webarticles-proxy">
3.         <constructor-arg index="0">
4.           <value>http://localhost/webarticles/webarticles.asmx</value>
5.         </constructor-arg>
6.       </object>
7.     <object id="articlesDomain" type="istia.st.articles.proxy.WebServiceArticlesProxy, webarticles-
   proxy">
8.       <property name="webServiceProxy">
9.         <ref object="proxy" />
10.      </property>
11.    </object>
12.    <object id="panier" type="istia.st.articles.domain.Panier, webarticles-domain" />
```

- lignes 2-6 : on instancie un objet de type [WebServiceArticles]. Celui a le constructeur suivant :

```
    ' constructeur
    Public Sub New(ByVal urlWebServiceArticles As String)
      MyBase.New()
      Me.Url = urlWebServiceArticles
    End Sub
```

Ce constructeur nécessite un paramètre qui est l'url du service web cible. Ici ce sera [http://localhost/webarticles/webarticles.asmx], ligne 4.

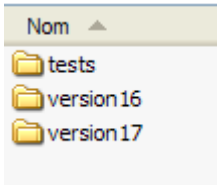
- lignes 7-11 : on instancie un objet [istia.st.articles.proxy.WebServiceArticlesProxy] implémentant l'interface [IArticlesDomain]. Cette classe a l'attribut suivant :

```
' le proxy d'accès au service web
Private _webServiceProxy As WebServiceArticles
Public Property webServiceProxy() As WebServiceArticles
  Get
    Return _webServiceProxy
  End Get
  Set(ByVal Value As WebServiceArticles)
    _webServiceProxy = Value
  End Set
End Property
```

Cet attribut de type [WebServiceArticles] est initialisé ligne 9 par l'objet "proxy" défini lignes 2-6.

7.2 Les tests de l'application web [webarticles-part3]

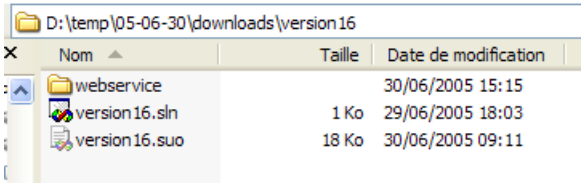
Nous sommes prêts pour les tests. Le lecteur est invité à les reproduire à partir des sources disponibles à l'url [ftp://ftp-developpez.com/tahe/fichiers-archive/vs-web3tier-dotnet-part3.zip]. Une fois le fichier précédent décompressé, on obtient le dossier suivant :



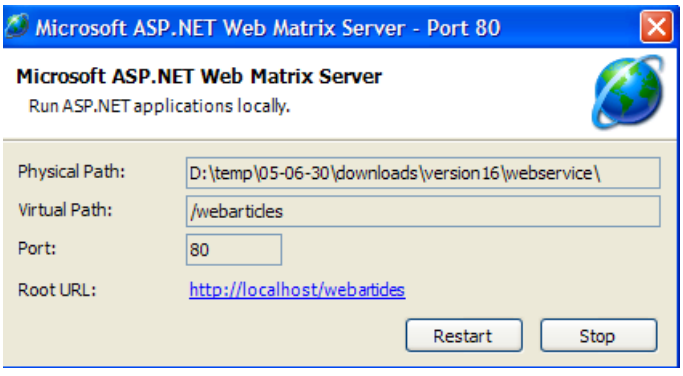
- **version16** : la solution Visual Studio de la couche [webservice]
- **version17** : la solution Visual Studio de la couche [proxy]
- **tests** : le dossier des exécutables du client

7.2.1 Configuration du serveur Cassini

Le service web doit être déclaré à votre serveur web. Nous utilisons ici le serveur gratuit Cassini. Le service web est dans le dossier [version16/webservice] :



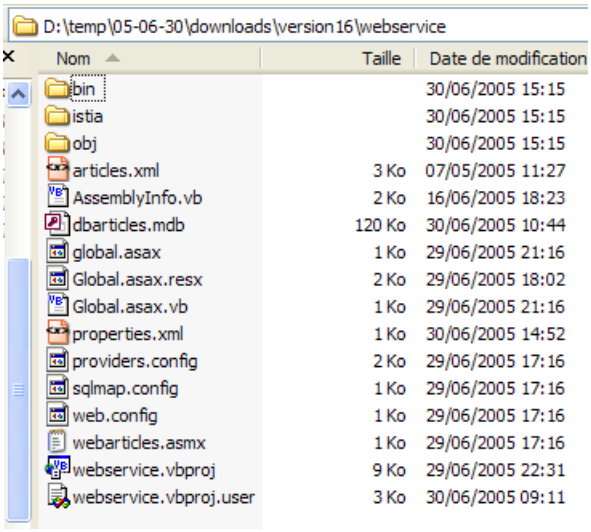
Le serveur Cassini sera donc configuré comme suit :



Le lecteur adaptera cette configuration à son propre environnement.

7.2.2 Configuration de la source de données

Rappelons la structure du dossier du service web :



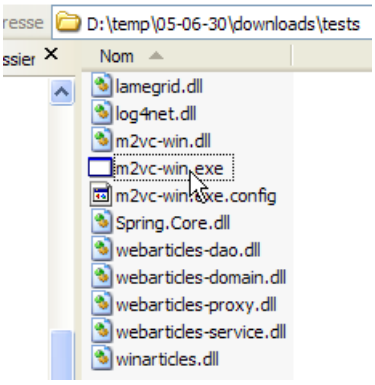
Le fichier [properties.xml] de ce dossier doit contenir la chaîne de connexion à la source de données des articles. Pour faciliter les tests, ce sera ici une base ACCESS. L'article 2 de la série a montré que la couche [dao] implémentée par [Ibatis SqlMap] savait travailler avec une multitude de sources de données sans modification de code. Nous utiliserons le fichier [properties.xml] suivant :

```
1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <settings>
3.   <add key="provider" value="OleDb1.1" />
4.   <add
5.      key="connectionString"
6.      value="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=D:\temp\05-06-30\downloads\version16\webservice\dbarticles.mdb;"/>
7.  </settings>
```

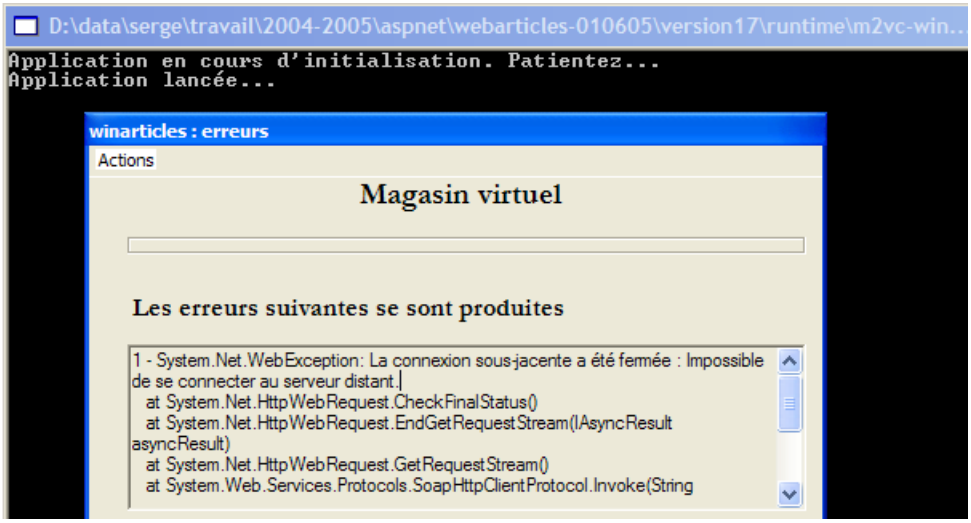
Le lecteur adaptera cette configuration à son propre environnement.

7.2.3 Test 1

Nous lançons le client sans lancer le serveur. Rappelons que pour lancer le client, il suffit de double-cliquer sur l'exécutable [m2vc-win.exe] du dossier des exécutables du client.



Nous obtenons le résultat suivant :



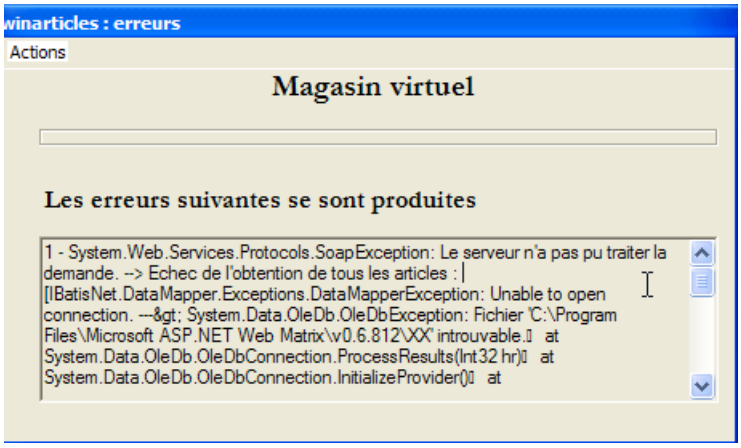
Ce premier résultat est encourageant. Il montre que notre client gère correctement les exceptions.

7.2.4 Test 2

Nous lançons le serveur Web Cassini comme il a été indiqué plus haut au paragraphe 7.2.1. Puis, nous éditons le fichier [properties.xml] décrit au paragraphe 7.2.2. Ligne 6, nous mettons un nom de fichier inexistant :

```
1. <?xml version="1.0" encoding="utf-8" ?>
2. <settings>
3.  <add key="provider" value="OleDb1.1" />
4.  <add
5.    key="connectionString"
6.    value="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=XX;"/>
7. </settings>
```

Arrêtons puis relançons le service web. Lançons le client en double-cliquant sur l'exécutable [m2vc-win.exe] du dossier des exécutables du client. Nous obtenons le résultat suivant :



De nouveau, l'exception attendue est correctement gérée. On voit ci-dessus qu'elle est différente de la précédente. C'est ici la couche [dao], plus exactement son implémentation [Ibatis SqlMap] qui signale l'inexistence du fichier XX. Au passage, on peut voir où celui-ci a été cherché. Pour qu'il soit trouvé, il faut mettre son chemin absolu dans le fichier [properties.xml]. C'est ce que nous faisons maintenant pour un troisième test.

7.2.5 Test 3

Le lecteur est invité à placer dans le fichier [properties.xml] du dossier du service web, le chemin absolu du fichier ACCESS à gérer. Arrêtons puis relançons le service web. Lançons le client en double-cliquant sur l'exécutable [m2vc-win.exe] du dossier des exécutables du client. Nous obtenons le résultat suivant :

Nous avons bien obtenu la liste des articles. Le lecteur est invité à découvrir le fonctionnement de l'application en reproduisant la séquence d'actions décrite page 17, paragraphe 3.5.

8 Conclusion

Cet article met fin à une série d'articles sur l'architecture des applications web multi-couches où nous avons découvert comment :

- structurer une application en couches, que celle-ci soit dans le domaine du web ou qu'elle soit une application windows "standalone" classique,
- rendre ces couches interchangeables grâce à l'utilisation d'interfaces,
- configurer l'ensemble des couches de l'application grâce à Spring,
- créer un client riche pour un service web,
- donner à l'application une architecture MVC stricte, que l'application soit dans le domaine du web ou qu'elle soit une application windows "standalone" classique.

Par ailleurs, pour arriver à nos fins, nous avons été amenés à créer un moteur MVC générique appelé [M2VC-win] utilisable par toute application windows .NET classique.

Enfin rappelons que le socle sur lequel tout repose est **Spring** et son concept d'IoC (**Inversion of Control**) appelé également DI (**Dependency Injection**). Sans Spring, tout aurait été plus compliqué. Le lecteur est ardemment invité à s'intéresser à cet outil.

---

9 Annexes

9.1 Le fichier de description du service web [WebServiceArticles]

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:tns="istia.st.articles.webservice"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
targetNamespace="istia.st.articles.webservice" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified" targetNamespace="istia.st.articles.webservice">
      <s:element name="testAchatPanier">
        <s:complexType />
      </s:element>
      <s:element name="testAchatPanierResponse">
        <s:complexType>
          <s:sequence>

            <s:element minOccurs="0" maxOccurs="1" name="testAchatPanierResult" type="tns:WSPanier" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:complexType name="WSPanier">
        <s:complexContent mixed="false">
          <s:extension base="tns:Panier">
            <s:sequence>
              <s:element minOccurs="0" maxOccurs="1" name="erreurs" type="tns:ArrayOfAnyType" />

            </s:sequence>
          </s:extension>
        </s:complexContent>
      </s:complexType>
      <s:complexType name="Panier">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="achats" type="tns:ArrayOfAnyType" />
          <s:element minOccurs="1" maxOccurs="1" name="totalPanier" type="s:double" />
        </s:sequence>

      </s:complexType>
      <s:complexType name="ArrayOfAnyType">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="unbounded" name="anyType" nillable="true" />
        </s:sequence>
      </s:complexType>
      <s:complexType name="Achat">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="article" type="tns:Article" />

          <s:element minOccurs="1" maxOccurs="1" name="qte" type="s:int" />
        </s:sequence>
      </s:complexType>
      <s:complexType name="Article">
        <s:sequence>
          <s:element minOccurs="1" maxOccurs="1" name="id" type="s:int" />
          <s:element minOccurs="0" maxOccurs="1" name="nom" type="s:string" />
          <s:element minOccurs="1" maxOccurs="1" name="prix" type="s:double" />
          <s:element minOccurs="1" maxOccurs="1" name="stockactuel" type="s:int" />

          <s:element minOccurs="1" maxOccurs="1" name="stockminimum" type="s:int" />
        </s:sequence>
      </s:complexType>
      <s:element name="status">
        <s:complexType />
      </s:element>
      <s:element name="statusResponse">
        <s:complexType>
          <s:sequence>

            <s:element minOccurs="0" maxOccurs="1" name="statusResult" type="tns:ArrayOfAnyType" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="acheterPanier">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="panier" type="tns:Panier" />
          </s:sequence>

        </s:complexType>
      </s:element>
      <s:element name="acheterPanierResponse">
```

```
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="acheterPanierResult" type="tns:WSPanier" />
          </s:sequence>
        </s:complexType>
      </s:element>

      <s:element name="getArticleById">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="idArticle" type="s:int" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="getArticleByIdResponse">
        <s:complexType>

          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="getArticleByIdResult" type="tns:Article" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="getAllArticles">
        <s:complexType />
      </s:element>
      <s:element name="getAllArticlesResponse">

        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="getAllArticlesResult" type="tns:ArrayOfAnyType"
/>
          </s:sequence>
        </s:complexType>
      </s:element>
    </s:schema>
  </wsdl:types>
  <wsdl:message name="testAchatPanierSoapIn">

    <wsdl:part name="parameters" element="tns:testAchatPanier" />
  </wsdl:message>
  <wsdl:message name="testAchatPanierSoapOut">
    <wsdl:part name="parameters" element="tns:testAchatPanierResponse" />
  </wsdl:message>
  <wsdl:message name="statusSoapIn">
    <wsdl:part name="parameters" element="tns:status" />
  </wsdl:message>
  <wsdl:message name="statusSoapOut">

    <wsdl:part name="parameters" element="tns:statusResponse" />
  </wsdl:message>
  <wsdl:message name="acheterPanierSoapIn">
    <wsdl:part name="parameters" element="tns:acheterPanier" />
  </wsdl:message>
  <wsdl:message name="acheterPanierSoapOut">
    <wsdl:part name="parameters" element="tns:acheterPanierResponse" />
  </wsdl:message>
  <wsdl:message name="getArticleByIdSoapIn">

    <wsdl:part name="parameters" element="tns:getArticleById" />
  </wsdl:message>
  <wsdl:message name="getArticleByIdSoapOut">
    <wsdl:part name="parameters" element="tns:getArticleByIdResponse" />
  </wsdl:message>
  <wsdl:message name="getAllArticlesSoapIn">
    <wsdl:part name="parameters" element="tns:getAllArticles" />
  </wsdl:message>
  <wsdl:message name="getAllArticlesSoapOut">

    <wsdl:part name="parameters" element="tns:getAllArticlesResponse" />
  </wsdl:message>
  <wsdl:portType name="WebServiceArticlesSoap">
    <wsdl:operation name="testAchatPanier">
      <wsdl:input message="tns:testAchatPanierSoapIn" />
      <wsdl:output message="tns:testAchatPanierSoapOut" />
    </wsdl:operation>
    <wsdl:operation name="status">
      <wsdl:input message="tns:statusSoapIn" />

      <wsdl:output message="tns:statusSoapOut" />
    </wsdl:operation>
    <wsdl:operation name="acheterPanier">
      <wsdl:input message="tns:acheterPanierSoapIn" />
      <wsdl:output message="tns:acheterPanierSoapOut" />
    </wsdl:operation>
    <wsdl:operation name="getArticleById">
      <wsdl:input message="tns:getArticleByIdSoapIn" />
      <wsdl:output message="tns:getArticleByIdSoapOut" />
```

```
    </wsdl:operation>
    <wsdl:operation name="getAllArticles">
      <wsdl:input message="tns:getAllArticlesSoapIn" />
      <wsdl:output message="tns:getAllArticlesSoapOut" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="WebServiceArticlesSoap" type="tns:WebServiceArticlesSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
    <wsdl:operation name="testAchatPanier">

      <soap:operation soapAction="istia.st.articles.webservice/testAchatPanier" style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="status">

      <soap:operation soapAction="istia.st.articles.webservice/status" style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="acheterPanier">

      <soap:operation soapAction="istia.st.articles.webservice/acheterPanier" style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="getArticleById">

      <soap:operation soapAction="istia.st.articles.webservice/getArticleById" style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="getAllArticles">

      <soap:operation soapAction="istia.st.articles.webservice/getAllArticles" style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>

  <wsdl:service name="WebServiceArticles">
    <documentation xmlns="http://schemas.xmlsoap.org/wsdl/" />
    <wsdl:port name="WebServiceArticlesSoap" binding="tns:WebServiceArticlesSoap">
      <soap:address location="http://localhost/webarticles/webarticles.asmx" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Table des matières