J2EE vs .NET

Une comparaison à l'aide de l'application Cybercoach

Travail de Master

Alain Giller

Avril 2006

Supervisé par :

Prof. Dr. Jacques Pasquier-Rocha
et
Dr. Patrik Fuhrer
Groupe Génie Logiciel

UNIVERSITÉ DE FRIBOURG SUISSE
UNIVERSITÄT FREIBURG SCHWEIZ

Groupe Génie Logiciel
Département d'Informatique
Université de Fribourg (Suisse)

Software Engineering Group

“La persévérance, c’est ce qui rend l’impossible possible, le possible probable et le probable réalisé.”

- *Robert Half*

Résumé

Le but de ce travail de master est l'étude de la technologie .NET et la comparaison à son concurrent direct Java 2 Platform, Enterprise Edition (J2EE). La comparaison se fait à l'aide d'un exemple concret : le projet Cybercoach dont une application en J2EE a déjà été entièrement dévelopée et documentée dans le cadre d'un autre travail de master.
Il s'agit donc d'une étude de la solution existante (J2EE) et de l'implémentation d'une version aux mêmes fonctionnalités avec la technologie .NET et le langage C#. Afin de pouvoir bénéficier de l'indépendance de plateformes, Mono, l'implémentation libre de .NET, a été étudiée.
L'application développée respecte les principales qualités d'un logiciel, telles que, modularité, extensibilité, capacité à monter en charge, sécurité, etc. Pour atteindre cet objectif, différents concepts et techniques du génie logiciel ont été mis en pratique. Ce rapport décrit les différentes étapes nécessaires au développement d'une application avec .NET et les comparent (similarités, différences, avantages, désavantages) avec la plateforme J2EE.

Mots clés :
Java, C#, J2EE, .NET, Mono, ASP.NET, Struts, ADO.NET, EJB, architecture multi-tier, design patterns, Cybercoach

Table des matières

Liste des Figures

Liste des tableaux

Liste des codes source

1 Introduction

1.1 Motivation et Objectifs

Dans le domaine des applications d'entreprises, Java fut le premier à proposer une solution (J2EE). La platforme proposée est en fait un ensemble de technologies ou spécifications Java regroupées sous une seule bannière. Microsoft arriva dans un deuxième temps. Il proposa un *framework* (.NET) qui est une re-architecture de différentes technologies Microsoft. Dans la situation actuelle les deux technologies sont en concurrence directe dans plusieurs domaines, celui du *web* principalement. Laquelle est la meilleure, la plus sûre, performante ou complète ? Ce travail vise à répondre à ces questions.

D'un point de vu plus personnel, ce travail a également pour but de découvrir la technologie .NET. C'est pourquoi, les parties .NET sont parfois un peu plus développées que celles en J2EE.

1.2 Architectures multi-tiers

Les systèmes logiciels complexes sont séparés en plusieurs couches, ou tiers. Cette façon de faire offre beaucoup d'avantages. L'un d'eux est qu'un développeur peut ne s'occuper que d'une couche, sans trop se soucier du reste. Un modèle classique d'architecture multi-tiers est le 3-tiers. Les diffèrentes couches sont alors séparées de la manière suivante :

- **Présentation**
  La couche présentation s'occupe des interactions avec l'utilisateur. Dans la plupart des cas, il s'agit d'une interface client riche ou d'une interface *web*. La couche présentation relaie les requêtes de l'utilisateur à destination de la couche métier, et en retour présente

à l'utilisateur les informations renvoyées par les traitements de la couche métier. Il s'agit donc ici d'un assemblage de services métiers et applicatifs offerts par la couche inférieure.

- **Métier**
  La couche métier correspond à la partie fonctionnelle de l'application. C'est elle qui implémente la "logique", et qui décrit les opérations que l'application va appliquer sur les données en fonction des requêtes des utilisateurs, effectuées au travers de la couche présentation. Les différentes règles de gestion et de contrôle du système sont mises en oeuvre dans cette couche. La couche métier offre des services applicatifs et métiers à la couche présentation. Pour fournir ces services, elle s'appuie, le cas échéant, sur les données du système, accessibles au travers des services de la couche inférieure. En retour, elle renvoie à la couche présentation les résultats qu'elle a calculés.
- **Données**
  La dernière couche, est celle de l'accès aux données. Ces données sont destinées à durer dans le temps, de manière plus ou moins longue, voire définitive. Les données peuvent être stockées indifféremment dans de simples fichiers texte, des fichiers eXtensible Markup Language (XML), ou encore dans une base de données. Quel que soit le support de stockage choisi, l'accès aux données doit être le même. Cette abstraction améliore la maintenance du système.

1.3 Cadre du projet

1.3.1 Application Cybercoach

L'idée du Cybercoach vient d'un projet commun de l'Ecole d'ingénieurs et d'architectes de Fribourg et de l'entreprise Tecost. Il s'agit d'un système pour l'assistance de l'entraînement d'un sportif. Le système récolte des données (rythme cardiaque, dénivelé, etc.) provenant de différentes sources (montre "Polar", GPS, etc.) et les sauvegardes. Les fonctions de base de Cybercoach sont :

- offrir aux athlètes un outil pour :
  ▷ définir, actualiser leurs profils (âge, poids, rythme cardiaque au repos, etc.),
  ▷ enregistrer, visualiser, analyser leurs entraînements,
- récolter des données (pulsations cardiaque, coordonnées, etc.) pendant l'entraînement (montre Polar, etc.).

Le projet pourrait être étendu et offrir des fonctions plus avancées comme :

- offrir un service de suivi personnalisé (coachs professionnels, intelligence artificielle) ;
- catalogue de programmes d'entraînement ;
- conseils pour la nutrition (diététiciens, intelligence artificielle) ;
- e-Shop (articles de sports et nutrition) ;
- etc.

Pour des informations plus complètes et détaillées, voir le chapitre 1.2 de [Brü05].

- **Cybercoach J2EE**
  Le développement avec J2EE du projet Cybercoach était l'objet du travail de master de Dominic Brügger en 2005 [Brü05]. Dans le cadre de ce projet je n'ai pas eu à réimplémenter cette version J2EE mais uniquement en comprendre le fonctionnement

et l'architecture. Son application ainsi que la documentation sont toujours accessibles [2].

- **Cybercoach .NET**
  Ce projet s'est donc concentré sur l'implémentation de Cybercoach en .NET. Cette implémentation s'est faite avec les dernières technologies en vigueur, à savoir Visual Studio (VS) 2005 et la version 2.0 de .NET. Ceci est important à retenir lors de la comparaison, car la version .NET a une année d'avance sur son homologue en J2EE et bénéficie donc de technologies plus avancées. L'application ainsi que la documentation sont disponibles sur [3].

1.3.2 Connaissances techniques personelles préalables

Faire une comparaison est toujours une chose délicate. En effet, beaucoup d'aspects dépendent des préférences/besoins et connaissances propres de chaque personne. C'est pourquoi il est important de connaître mes niveaux de compétences préalables dans les différentes technologies comparées.

- **Java**
  C'est le langage que j'ai le plus utilisé en programmation. Depuis environ 6 ans, c'est le language avec lequel je travaille principalement.
- **C/C++**
  C'est un langage avec lequel j'ai également souvent travaillé. Je l'ai utilisé pendant environ 2 ans.
- **J2EE**
  C'est avec cette plateforme que j'ai travaillé dernièrement et surtout, développé un projet (MiniCybercoach) qui correspond à la version "*light*" de Cybercoach. Mes connaissances sont donc bonnes et surtout très fraîches. Ce projet a été fait dans le cadre de mes études.
- **.NET/C#**
  Mes connaissances étaient nulles avant de commencer ce projet. La première partie de ce projet a donc consisté à apprendre le fonctionnement de ce *framework* et surtout son Application Programming Interface (API). Puis, dans un deuxième temps, de développer une architecture propre à .NET (alors que dans le cas de J2EE, l'architecture était très proche du projet vu en classe).

1.4 Organisation

Ce rapport est organisé comme suit :

1. la première partie présente le projet Cybercoach codé avec l'une ou l'autre des deux plateformes ;
2. la deuxième partie consiste en une présentation générique des deux protagonistes, J2EE et .NET ;
3. et finalement une conclusion qui synthétise les principales différences et les leçons apprises.

Pour des raisons de clarté, les chapitres concernant chacune des technologies sont structurés de manière analogue.

1.5 Notations et Conventions

- Conventions de mise en page :
  - ▷ *italique* : utilisés pour les mots de langue anglaise ;
  - ▷ **gras** : utlisés comme en-tête de paragraphe ou pour mettre en évidence ;
  - ▷ `adresse` : Pour les adresses Internet ;
  - ▷ `Code` : est utilisé pour toutes références à un code source. C'est à dire un object, une classe, un mot clé, le nom d'une méthode, etc.
  - ▷ `Fichier.xml` : est utilisé pour tout les chemins ou noms de fichiers/répertoires.
- Ce rapport est divisé en chapitres, eux-mêmes subdivisés en sections. Si nécessaire, les sections sont divisées en sous-sections et ces dernières peuvent contenir plusieurs paragraphes.
- Les figures (Figures), tableaux (Tables) et codes source (Listings) sont numérotés à l'intérieur d'un chapitre. Par exemple une référence à la figure Figure j du chapitre i est notée *Figure i.j.*
- Les codes sources sont présentés de la manière suivante : voir Listing 1.1

```
SqlConnection conn = new SqlConnection(connString);
try{
    conn.Open();
}catch(Exception e){ //Exception has been raised
    Console.WriteLine("Error when contacting the database: " + e);
}finally{ //finally bloc
    conn.Close();
}
```

Listing 1.1: Exemple de présentation de code source

Première partie

Cybercoach

2
Cybercoach J2EE

Il est important de rappeler ici, que la version J2EE de Cybercoach a été développée par un autre étudiant, une année avant la version .NET [Brü05]. Pour les détails, le rapport de la version J2EE[1] est toujours disponible en ligne.

2.1 Structure de Cybercoach J2EE

L'architecture globale de Cybercoach J2EE (voir Figure 2.1) est divisée en deux conteneurs. Dans celui des *beans*, on trouve le tiers données (voir sous-section 2.1.3) avec les *entity-beans* qui offrent une vue objet de la base de données aux *session-beans* qui sont le tiers métier (voir sous-section 2.1.2). Le conteneur *web* (qui ne se trouve pas nécessairement sur la même machine que le conteneur des *beans*) contient le tiers présentation qui est fait avec le *framework* Struts de la fondation Apache [1], voir la sous-section 2.1.1.

Toute l'application se trouve dans un seul fichier Enterprise ARchive (EAR) qu'il suffit de déployer pour que l'application tourne.

2.1.1 Tiers présentation, Struts

Le tiers de présentation, se trouve dans une archive Web ARchive (WAR) à l'intérieur de l'archive EAR. Il est fait avec le *framework* Struts qui s'appuie sur la technologie des Servelts et des Java Server Pages (JSP). Pour des explications plus détaillées, voir le chapitre 5 de [Brü05].

Concrètement, pour l'interface *web* le programmeur a dû créer à la main :

- un fichier struts-config.xml avec toute la configuration des pages *web*, il s'agit du contrôleur du pattern Model View Controler (MVC) ;

[1] Le rapport est disponible uniquement en allemand.

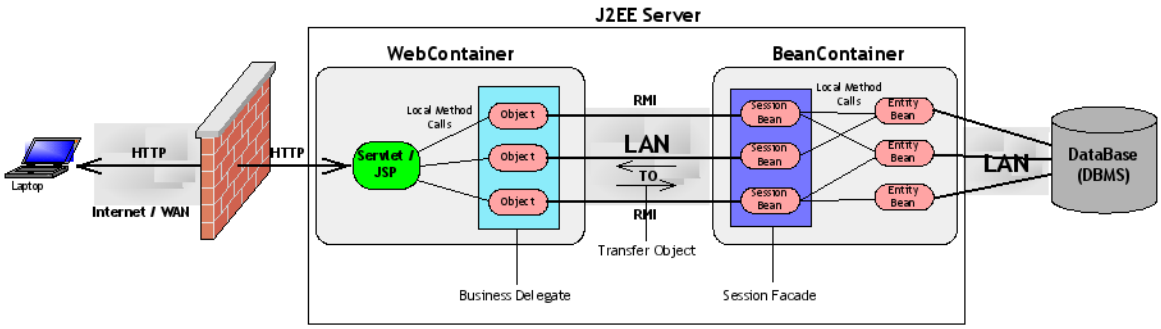

Figure 2.1: Architecture globale de Cybercoach J2EE

```
<tiles-definitions>
2    <definition name="base.layout" path="/layout.jsp">
         <put name="header" value="/header.jsp"/>
4        <put name="menu" value="/menu.jsp"/>
         <put name="content" value="${content}"/>
6        <put name="footer" value="/footer.jsp"/>
     </definition>
8    <definition name="welcome.page" extends="base.layout">
         <put name="content" value="/welcome.jsp"/>
10   </definition>
</tiles-definitions>
```

Listing 2.1: Extrait de `tiles-defs.xml`

- un fichier tiles-defs.xml Listing 2.1 pour créer une structure générale (*template*) des pages JSP. Dans l'exemple, le programmeur défini un base.layout (ligne 2) qui contient un entête, un menu, un contenu (qui doit encore être défini, on le voit grâce au champ value=${content}) et un pied de page (lignes 3 à 6). Ensuite, il doit définir le contenu pour chaque page. Ici on ne voit que la défintion du contenu de welcome.page (ligne 8) ;
- pour les validations des formulaires, un fichier validation.xml et un validator-rules.xml ;
- un fichier web.xml pour les droits d'accès, la définition des rôles, la définition du processus d'authentification, la description des *taglibs*, le nom du fichier de bienvenue, etc.
- le contenu de toutes les pages JSP. Un exemple de fichier JSP est disponible sous Listing 2.2. Cet exemple, défini le *layout* de l'application. On peut voir la déclaration de trois *taglibs* (lignes 1 à 3). Le *taglib* /tags/struts-tiles correspond au Listing 2.1. Il permet de faire correspondre les pages JSP aux *tags* des lignes 16 à 19 ;
- des fichiers texte de ressources pour les différentes langues ;
- des fichiers Java, plutôt lourds a implémenter, pour toutes les actions, *beans* et formulaires.

Les pages JSP contiennent des *taglibs* pour le contenu dynamique des pages comme, par exemple, la langue.

2.1.2 Tiers métier

Le tiers métier se trouve dans l'archive Java ARchive (JAR) à l'intérieur de l'archive EAR. Le tiers métier se divise concrètement en deux *session-beans*. Ce sont eux qui contiennent toute la logique métier et font le pont entre les *entiy-beans* (la persistance) et l'interface

```
<%@ taglib uri="/tags/struts-bean" prefix="bean" %>
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<%@ taglib uri="/tags/struts-tiles" prefix="tiles" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html:html xhtml="true">
    <head>
  <title><bean:message key="application.title"/></title>
  <link rel="stylesheet" href="styles/cybercoach.css" type="text/css" media="screen" />
  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
    </head>

    <body>
  <div id="container">
      <tiles:get name="header"/>
      <tiles:get name="menu"/>
      <tiles:get name="content"/>
      <tiles:get name="footer"/>
  </div>
    </body>
</html:html>
```

Listing 2.2: Fichier `layout.jsp` de Cybercoach

web. Dans les *sessions-beans*, en plus de la logique métier, on peut voir les champs XDoclet pour :

- transaction du type *required* ;
- les interfaces home et local ;
- toutes les références (les noms Java Naming and Directory Interface (JNDI)) aux entity-beans nécessaires ;
- les permissions d'accès pour les rôles de chaque méthode.

On constate donc que les transactions sont automatiques et que l'accès aux méthodes métier est autorisé uniquement aux personnes ayant le bon rôle (coach, par exemple). Les appels des méthodes métier se font depuis Struts par les fichiers action en java.

2.1.3 Tiers données, EJB

L'accès à la base de données se fait selon un modèle objet grâce aux *entity-beans* en mode Container Managed Persistence (CMP) (voir Chapitres 4.1.2 à 4.1.4 de [Brü05]). La base de données est MySQL 4.1 [7]. Les *entity-beans* modélisent la base de données. Il y a exactement un *entity-bean* par table dans la base de données. Il faut donc écrire les méthodes[2] set et get pour chaque champs des tables. Il faut en plus mettre des champs XDoclet pour les méthodes qui doivent apparaître sur l'interface de l'*entity-bean*. Pour chaque *entity-bean*, il faut encore ajouter les champs XDoclet pour le nom du *bean*, le type de persistance (CMP pour le Cybercoach), les permissions, les transactions, les *finders* et *query* en Enterprise Java Beans Query Language (EJBQL), etc.

[2]Il s'agit en fait de méthodes *abstract*, c'est-à-dire des méthodes avec des corps vides.

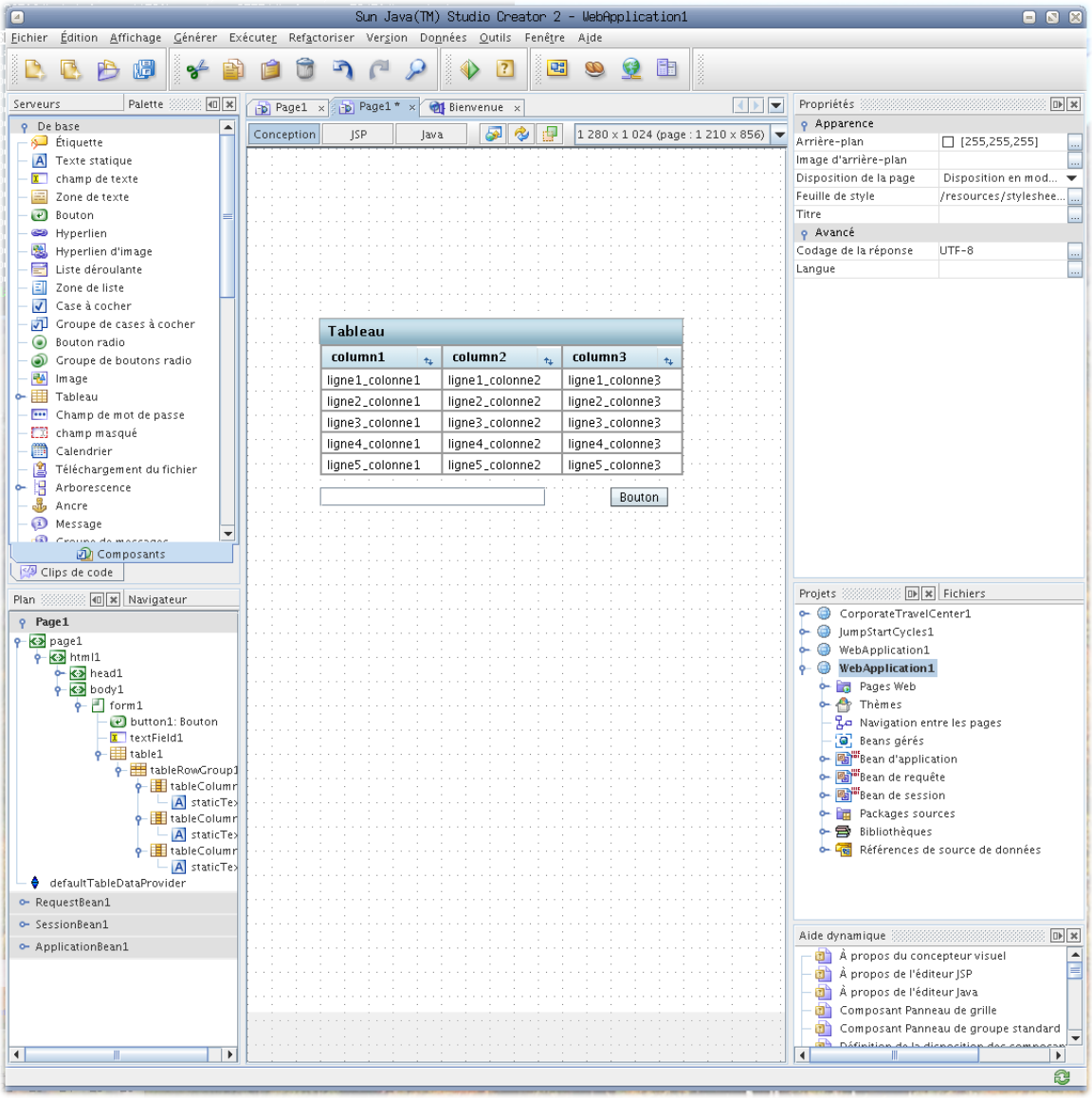

Figure 2.2: Sun Java Studio Creator

2.2 Autres possibilités, non exploitées dans Cybercoach

Avec les Java Server Faces (JSF), Java à rattrapé son retard dans le domaine des *webforms*. Les outils comme Sun Java Studio Creator [12], voir Figure 2.2, n'ont rien à envier à Visual Studio (VS) [23] de Microsoft pour la création de page *web*. De même, la nouvelle version de Netbeans [18], la version 5.0, facilite énormément la création d'applications d'entreprise en automatisant un maximum de tâches.

Du côté de la logique métier et de la persistance, la version 3 des Enterprise Java Beans (EJB) devrait fortement simplifier l'architecture et la programmation. Cette technologie remplace à la fois les *sessions* et les *entity-beans* dans leurs formes actuelles.

Une des grosses différence est le remplacement de l'héritage par l'injection sur le modèle

de la programmation orientée aspect (Aspect Oriented Programming (AOP)) pour les services transversaux tels que les transactions, la sécurité, la persistance, etc. Une autre différence importante est que le conteneur est appelé uniquement quand c'est nécessaire ou encore la suppression des XDoclet ou profit de noueaux *tag* Java.

On peut pratiquement affirmer, que le seul point commun entre les EJB versions 2 et 3 est le nom[3] [32]. Pour plus d'informations sur la programmation orientée aspect avec Java, voir [Lad03].

[3]Malgrés toutes les différences, la compatibilité avec les EJB 2 est assurée.

3

Cybercoach .NET

3.1 Stucture de Cybercoach .NET

L'architecture globale de Cybercoach .NET (voir Figure 3.1) est plus simple que celle avec J2EE (voir Figure 2.1). Le tiers données, voir la sous-section 3.1.3, est fait avec ADO.NET qui propose un modèle ensembliste. Le tiers métier, voir la sous-section 3.1.2 est très similaire à celui de J2EE. Et pour finir, le tiers présentation, voir la sous-section 3.1.1, est fait avec ASP.NET.

La version 2.0 du *framework* .NET avec le langage C# a été utilisée pour le développement de Cybercoach et seule l'implémentation de Microsoft est capable de la faire fonctionner actuellement. En effet, Mono [57], ne supporte pas encore complètement la version 2.0. Toutefois il est prévu que ce soit le cas d'ici la fin 2006. Les technologies ASP.NET, pour la partie *web*, et ADO.NET, pour la partie d'accès aux données, sont toutes deux des technologies propriétaires de Microsoft.

3.1.1 Tiers présentation, ASP.NET

Le tiers de présentation est fait avec la technologie ASP.NET. L'interface *web* est principalement composée de pages ASP.NET (fichier `*.aspx`) avec un contrôleur associé (dans le sens du *pattern* MVC) qui prend la forme d'un fichier C# (fichier `*.cs`) et d'un fichier de configuration `Web.config`. Le programmeur a dû concrètement implémenter :

- Tous les fichiers ASP.NET (`*.aspx`) correspondant aux différentes pages du site Cybercoach. La création se fait de manière graphique et la Cascading Style Sheets (CSS) a été reprise du projet Cybercoach version J2EE. Les fichiers se trouvent dans deux répertoires. Ceux se trouvant à la racine sont les fichiers accessibles par tout le monde et

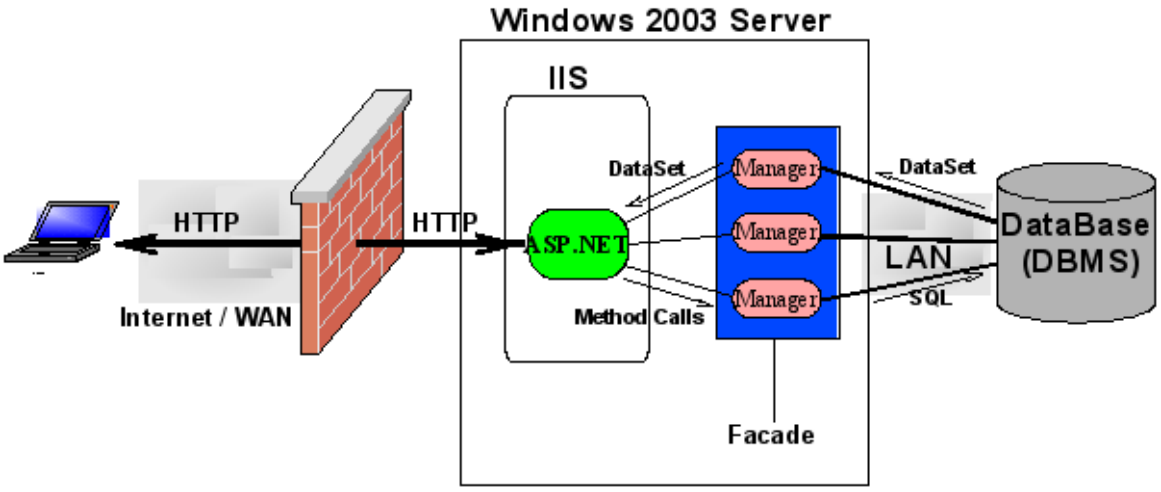

Figure 3.1: Architecture globale de Cybercoach .NET

```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <authorization>
      <deny users="?" />
    </authorization>
  </system.web>
</configuration>
```

Listing 3.1: Fichier `Web.config` pour la restriction aux utilisateurs authentifiés

dans l'autre répertoire, ils sont réservés aux utilisateurs authentifiés. Les droits d'accès sont spécifiés dans le fichier `Web.config` se trouvant dans chacun des deux répertoires.

- Tous les fichiers contrôleurs (`*.cs`) associés aux pages `*.aspx`. Ces derniers correspondent au code qui doit être exécuté côté serveur (par exemple, le code pour changer la langue d'affichage ou encore pour l'envoi des données d'un formulaire).
- Le fichier de configuration `Web.config`, voir Listing 6.1. Il contient toute la configuration du projet, à savoir :
  - ▷ les droits d'accès (pour les détails, voir la sous-section 6.5.2) ;
  - ▷ le *string* d'accès à la base de données (ligne 2 de Listing 6.1) ;
  - ▷ le profile[1] (lignes 16 à 28 de Listing 6.1) ;
  - ▷ etc.
- Un deuxième fichier `Web.config`, voir Listing 3.1, se trouve dans le dossier contenant les pages accessibles uniquement aux personnes authentifiées. Ce dernier est très simple, il interdit tout accès aux personnes non authentifiées (`<deny users="?" />`, le `?` veut dire "utilisateur non authentifié").
- Un fichier `Sitemap.xml`, voir Listing 7.4, contenant le menu de l'application pour les utlisateurs authentifiés (pour les autres, le menu est fait de manière statique, car il n'y a que deux pages différentes). Pour chaque menu, les rôles y ayant accès sont spécifiés.
- Un fichier `Global.asax` (voir Listing 3.2) pour définir les évènements, objets et variables au niveau de l'application (et pas seulement dans un *webform*). En l'occurence,

[1] Dans l'application Cybercoach, le profile est utilisé pour définir la langue. Les autres possibilités qu'offre normalement le profile sont (actuellement) réservées aux utilisateurs de SQL Server.

```
<%@ Application Language="C#" %>
<script runat="server">
    void Application_Start(object sender, EventArgs e){ /* Code that runs on application
        startup*/ }
    void Application_End(object sender, EventArgs e){ /*Code that runs on application shutdown
        */}
    void Application_Error(object sender, EventArgs e) { /* Code that runs when an unhandled
        error occurs*/}
    void Session_Start(object sender, EventArgs e){ /* Code that runs when a new session is
        started*/
        string language = this.Request.UserLanguages[0];
        Session["Language"] = language;
    }
    void Session_End(object sender, EventArgs e){// Code that runs when a session ends. }
</script>
```

Listing 3.2: Fichier `Global.asax`

```
...
  <data name="editProfile" xml:space="preserve">
    <value>Athletisches Profil &amp;auml;ndern</value>
  </data>
  <data name="editUser" xml:space="preserve">
    <value>Benutzerangaben &amp;auml;ndern</value>
  </data>
...
```

Listing 3.3: Extrait du fichier `Menu.de.resx`

il est utilisé pour changer la langue en début de session. On voit dans l'exemple, ligne 7, la récupération de la langue de l'utilisateur que l'on sauvegarde dans la variable globale Session à la ligne 8.

- Des fichiers de ressources en XML pour les langues.

Pour faire un site multilingue, il faut créer des fichiers (en XML) de ressources contenant à chaque fois le terme dans une langue, et le *tag* unique pour le remplacer dans les pages. La facon de faire est similaire à Struts. Il suffit d'ajouter des *tags* dans les pages `.aspx` de la forme :

```
<%$ Resources:Menu, editProfile %>
```

Dans ce cas, le *tag* sera remplacé par le terme correspondant à "editProfile" dans le fichier de ressources XML `App_GlobalResources/Menu.??.resx`, voir Listing 3.3. les deux " ? ?" correspondent à un code de langue tel que "fr" pour le français, "de" pour l'allemand ou encore vide pour la langue par défaut.

3.1.2 Tiers métier

Le tiers métier se trouve dans le répertoire `App_Code`. Contrairement à la version J2EE, il n'y a pas deux couches (*session* et *entity-beans*) mais qu'une seule. Il y a néanmoins deux fichiers qui corresponderaient aux *session-beans* (`ActivityManager.cs` et `PersonManager.cs`). Leur logique métier est d'ailleurs similaire à la version J2EE.

Avec les *entity-beans*, le programmeur a recréé une structure objet qui correspond à la base de données. Ici, c'est l'objet DataSet qui va remplir ce rôle. Pour l'affichage de données sur les pages ASP.NET, il suffit de donner directement l'objet DataSet au composant *web*

```
string _connectionString = ConfigurationManager.ConnectionStrings["MySqlCybercoachDB"].
    ConnectionString;
string query = "SELECT weight, height, validTo FROM profile WHERE profile.userId = '" +
    userID + "'";
  MySqlConnection conn = new MySqlConnection(_connectionString);
  MySqlDataAdapter da = new MySqlDataAdapter();
  da.SelectCommand = new MySqlCommand(query, conn);
  DataSet ds = new DataSet();

  try{
      conn.Open();
      da.Fill(ds);
  }catch (MySqlException e){
      System.Console.Out.WriteLine("DEBUG: Exception in DataBase.cs = " + e);// Handle
          exception.
  }finally{
      conn.Close();
  }
  return ds.Tables[0];
```

Listing 3.4: Accès à la base de données grâce aux DataSet

(*webform*) pour que les données soient affichées. Il n'y a donc aucun intermédiaire (tel que des *Transfert Object*) ce qui simplifie la structure et le codage.

3.1.3 Tiers données, ADO.NET

L'accès à la base de données se fait de manière codée à l'aide du langage Structured (English) Query Language (SQL) (ligne 2, Listing 3.4) et des DataSet (ligne 10, Listing 3.4) qui font partie de ADO.NET. Il s'agit d'un modèle ensembliste (dans la version J2EE il s'agit d'un modèle objet). Avec Visual Studio (VS), l'accès aux données est très orienté SQL Server [10]. Il peut également accèder à Object Linking and Embedding (OLE) Open DataBase Connectivity (ODBC) et Oracle [9]. Pour le projet Cybercoach la base de données choisie est MySQL 4.1 [7] (car gratuite, et multi-plateformes). Actuellement, le *provider* de MySQL ne permet pas directement de décrire le schéma de la base de données à l'aide de XML Schema Definition Language (XSD) sous VS. On est donc obligé de passer par ODBC ce qui limite les fonctionnalitées et les performances. Dans la version actuelle du Cybercoach, l'utilisation de DataSet fortement typés [43] (à l'aide d'un fichier XSD) n'est pas utlisé.

3.2 Remarques

L'utilisation de *webforms* permet l'automatisation de beaucoup d'aspects (comme la création automatique et transparente de Javasript pour le client, ou l'affichage direct d'informations d'une base de données). En revanche, la configuration des *webforms* n'est pas aussi intuitive que pour le HyperText Markup Language (HTML). Si on veut, par exemple, afficher une table avec des informations provenant d'une base de données. En HTML on code une table avec un *tag* dans chaque cellule où l'information provient de la base de données. L'apparence de la table se modifie dans ce cas très facilement (mais pas forcément rapidement). Avec les *webforms*, il faut chercher le paramètre permettant, par exemple : une couleur différente pour les titres, ou pour que 1 ligne sur 2 soit grisée.

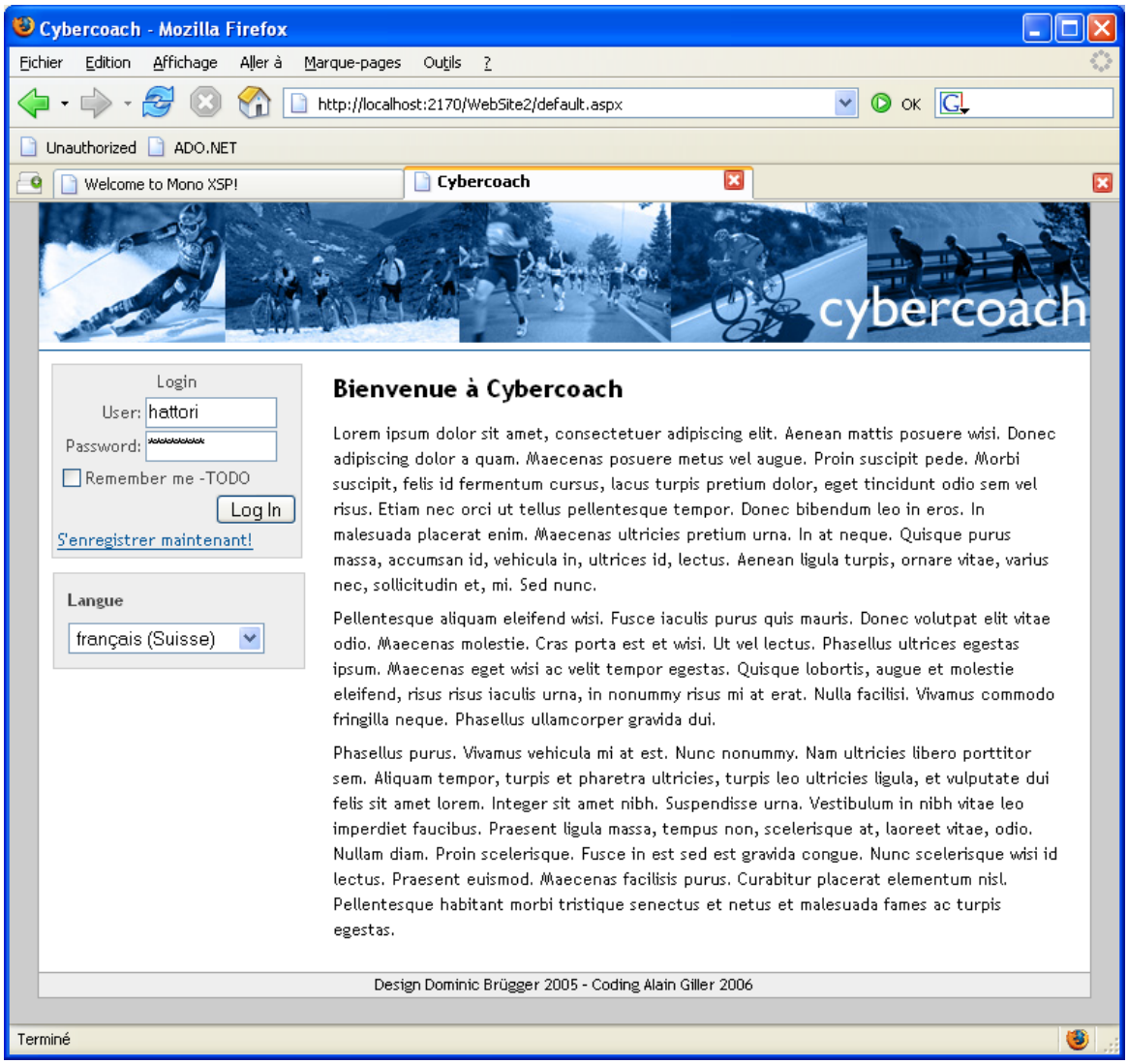

Figure 3.2: Page d'accueil de Cybercoach .NET

Les possibilités sont plus limitées (impossible[2], par exemple, d'afficher une ligne rose, une autre jaune et une troisième rouge et ainsi de suite).

3.3 Manuel d'utilisation

L'interface de Cybercoach .NET étant très proche de celle avec J2EE (voir Figure 3.2), le même manuel d'utilisation s'applique, voir Chapitre 6 [Brü05]. Toutefois, il y a quelques différences et certaines parties ne sont pas implémentées.

- Voici les parties implémentées :
  - ▷ l'utilisateur arrive non authentifié sur la page d'accueil, il peut choisir la langue (français, allemand, anglais) ;
  - ▷ il peut créer un nouveau compte ;
  - ▷ il peut s'authentifier et accéder à ses données personnelles ;

[2]En réalité, il existe des possiblités, mais cela devient relativement complexe.

  - ▷ il peut éditer ces informations personnelles ainsi que son profile athlétique ;
  - ▷ il peut afficher ses activités, ainsi que leurs détails.
- Les parties non implémentées :
  - ▷ il ne peut pas ajouter de nouvelles activitées / détails d'activitées ;
  - ▷ il ne peut pas modifier les droits des utilisateurs ;
  - ▷ les menus correspondant aux rôles (utilisateur, administrateur ou coach).
- Et enfin, les parties différentes :
  - ▷ le menu de la page d'accueil propose directement un module pour s'authentifier ;
  - ▷ le module d'authentification comprend directement un lien pour créer un nouveau compte ;
  - ▷ le module d'authentification comprend un case à cocher pour se souvenir de l'utilisateur (à la prochaine visite, l'utilisateur sera directement authentifié) ;
  - ▷ le choix de la langue se fait par une liste ;
  - ▷ l'apparence des menus est légèrement différente.

Deuxième partie

J2EE vs .NET

4

J2EE : Aspect global

Java 2 Platform, Enterprise Edition (J2EE) est une spécification pour le langage de programmation Java de Sun Microsystems plus particulièrement destinée aux applications d'entreprise. Dans ce but, toute implémentation de cette spécification contient un ensemble d'extensions au *framework* Java standard (Java 2, Standard Edition (J2SE)) afin de faciliter la création d'applications réparties. Voici une liste des API pouvant être contenues dans une implémentation J2EE [36], [35] :

- Servlets : Conteneur *Web* ;
- JSP : *Framework Web* (Java Server Pages (JSP)) ;
- JSF : *Framework Web*, extension des JSP ;
- EJB : Composants distribués transactionnels ;
- JNDI : API de connexion à des annuaires, notamment des annuaires Lightweight Directory Access Protocol (LDAP) ;
- JDBC : API de connexion à des bases de données ;
- JMS : API de communication asynchrone ;
- JCA : API de connexion, notamment à des Enterprise Resource Planning (ERP) ;
- JavaMail : API de gestion des mails ;
- JMX : Extension d'administration des applications ;
- JTA : API de gestion des transactions ;
- JAXP : API d'analyse XML ;
- JAXM : API de communication asynchrone par XML ;

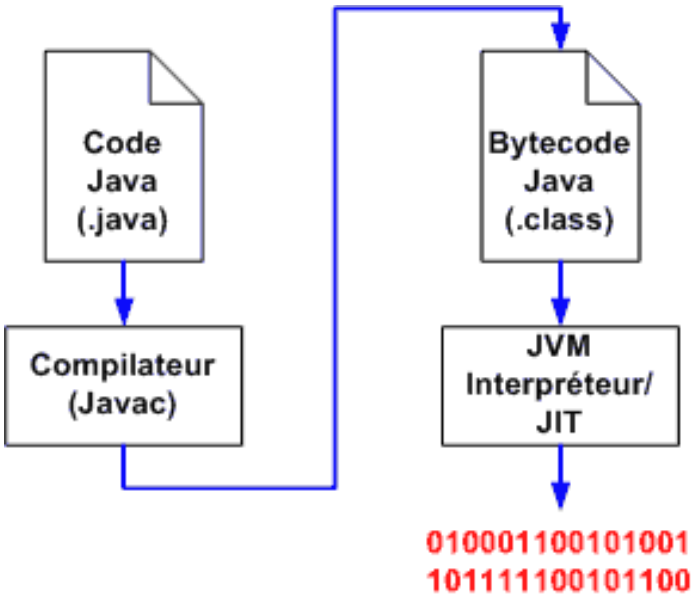

Figure 4.1: Vue d'ensemble de l'architecture Java

- JAX-RPC : API de communication synchrone par XML, par exemple, à l'aide du protocole Simple Object Access Protocol (SOAP) ;
- JAXB : API de sérialisation par XML ;
- JAXR : API de gestion des registres XML, permettant d'enregistrer des *Web Services* en Electronic Business using eXtensible Markup Language (ebXML) ;
- RMI : API de communication distante entre des objets Java ;
- Java IDL : API de communication entre objets Java et objets non-Java, via le protocole Common Object Request Broker Architecture (CORBA).

4.1 Architecture

Les applications Java sont compilées en *bytecode* (les fichiers `*.class`) destinés à être interprétés plus tard par la marchine virtuelle de Java (Java Virtual Machine (JVM)). Il ne s'agit donc pas de code machine, mais d'un code intermédiaire, voir Figure 4.1. Une application Java est, en règle générale, un ensemble de fichiers `.class` regroupés dans un paquetage (*package*). Ce paquetage prend la forme d'une archive compressée avec le standard *zip* renommée en JAR, ou WAR s'il s'agit d'une application *web*. Une application Java peut ainsi être déployée sur n'importe quelle plateforme compatible Java (c'est-à-dire avec un Java Runtime Environment (JRE) installé) sans être recompilée. On dit alors que les applications sont portables au niveau binaire. Pour une comparaison de l'architecture J2EE et .NET, voir [73].

4.1.1 Machine virtuelle

La machine virtuelle de Java (JVM) fait partie de JRE. L'environnement Java existe sur une très grande variété de machines, de systèmes d'exploitation[1] ou de plateformes, y compris sur la plateforme .NET[2], ce qui permet l'exécution de *bytecode* Java directement, sans recompilation.

4.1.2 JIT

Le *bytecode* Java était originellement destiné à l'interprétation par la machine virtuelle de Java (JVM). Depuis, des techniques de Just In Time (JIT) ont fait leur apparition, comme par exemple, les *HotSpot* de Sun Microsystems. Les *HotSpot* fonctionnent de la manière suivante : le système analyse continuellement les parties fréquemment ou périodiquement exécutées, pour cibler l'optimisation. Ce système permet d'avoir de bonnes performances avec les sections critiques sans pour autant trop pénaliser les autres sections. Les *HotSpot* sont largement reconnus pour leur améliorations de performances dans les JVM. En théorie, il est même possible que l'optimisation adaptative d'une JVM soit plus efficace qu'un code en C++ ou assembleur ! Mais en pratique, ce cas de figure est (hélas) peu fréquent.

Pour les *HotSpot* dans le JRE de Sun, il existe 2 versions. L'une est appelée client et l'autre serveur. La version client est optimisée pour un chargement rapide et compile uniquement les classes et méthodes essentielles. La version serveur, charge les applications plus lentement, mais fait plus d'efforts pour produire une compilation JIT très optimisée, pour de meilleures performances [33].

4.1.3 Java Runtime Environnement (JRE)

Le Java Runtime Environment (JRE) comprend la Java Virtual Machine (JVM) ainsi que toutes les bibliothèques nécessaires au fonctionnement des applications Java[3]. Voici quelques exemples de ces bibliothèques :

- la bibliothèque Java principale qui inclue, entre autres :
  - ▷ les bibliothèques de collections (les listes, arbres, ensembles, etc.) ;
  - ▷ les bibliothèques pour la gestion du XML ;
  - ▷ la sécurité ;
  - ▷ les bibliothèques d'internationalisation et de localisation ;
- les bibliothèques d'intégration (pour la communication avec des systèmes externes) :
  - ▷ API pour l'accès aux bases de données (Java DataBase Connectivity (JDBC)) ;
  - ▷ JNDI ;
  - ▷ Remote Method Invocation (RMI) et CORBA pour les applications distribuées ;
- les bibliothèques pour l'interface utilisateur (Graphical user interface (GUI)) :
  - ▷ Swing pour les environnements fenêtrés natifs ;

[1] Liste non exhaustive des systèmes disposant d'une implémentation de Java : Windows 3.1/95/98/NT4/2000/ME/XP/CE, MacOS, Solaris, Linux, AIX, OS/2, IRIX, UnixWare, HP/UX, Digital Unix, AmigaOS, BeOS, OpenVMS, FreeBSD, SunOS, RiscOS, etc.

[2] Le projet IKVM est une implémentation de la JVM en .NET [54].

[3] Toutefois, le JRE seul ne permet pas la compilation d'une application Java.

  - ▷ Abstract Windowing Toolkit (AWT), version plus légère que Swing, mais dont la GUI n'est pas native ;
  - ▷ API pour la capture audio, le traitement, et le *playback*.
- etc.

4.2 Implémentations existantes

Il existe plusieurs implémentations de Java (Sun [40], IBM [34], Blackdown [31], Lejos [39], etc. [38]). Pour J2EE qui est un ensemble de spécifications basées sur le langage Java, il existe également plusieurs implémentations. Les applications J2EE tournent dans ce qu'on appelle un serveur d'applications, voir la section 4.3 pour quelques exemples d'implémentations existantes.

4.3 Serveur d'application, tiers métier

Il est avant tout indispensable de définir clairement ce qu'est un serveur d'application. En effet, une confusion règne dans les esprits quant à la notion de serveur d'application. Cette confusion a été introduite en grande partie par les éditeurs de serveurs d'application J2EE afin de s'approprier, à tort, ce marché. La notion de serveur d'application a en effet été mélangée avec celle de serveur d'objet qui n'a absolument rien à voir. Le serveur d'application est l'environnement d'exécution des applications côté serveur. Il prend en charge l'ensemble des fonctionnalités qui permettent à N clients d'utiliser une même application (gestion de la session utilisateur, *pooling*, montées en charge et reprise sur incident) [73], [69].

Le serveur d'application est le coeur de J2EE. C'est lui qui va fournir le conteneur pour toute la logique métiers et les accès à la base de données. Il peut s'occuper aussi du conteneur *web* ou laisser cette tâche à un serveur *web*, tel Tomcat de Apache [1].

Il existe un très grand nombre d'implémentation de serveur d'application pour J2EE qui sont autant d'implémentations différentes des spécifications J2EE. Elles peuvent être gratuites, commercialles, *open-sources*, etc. Quelques exemples : Sun Java System Application Server [29], Websphere (IBM) [30], JBoss [28], Geronimo (Apache) [1], GlassFish (Sun) [27], BEA WebLogic [26], etc. Pour une liste plus complète, voir [37].

4.4 Outils de développement, IDE

De même que pour les serveurs d'application, il existe une multitude d'Integrated Development Environment (IDE). Une liste (non exhaustive), avec leurs principales caractéristiques, peut se trouver sous [16]. Voici quelques exemples :

- **Eclipse**
  C'est actuellement l'IDE le plus connu et probablement le plus utilisé, il est *open-source*, gratuit, et entièrement programmé en Java [13].
- **Websphere**
  Il s'agit d'Eclipse avec en plus des outils commerciaux développés par IBM [30].

- **Netbeans**
  Développé par Sun Microsystem, il est, tout comme Eclipse, entièrement programmé en Java, *open-source* et gratuit. Bien que légèrement moins populaire, il s'agit d'une excellente alternative à Eclipse qui est en avance sur ce dernier dans certains domaines, notamment la gestion des EJB [18].
- **Sun Java Studio Creator**
  Basé sur Netbeans, il ajoute la création What You See Is What You Get (WYSIWYG) d'applications *web* grâce aux JSF. L'interface est d'ailleurs très similaire à celle de Visual Studio (VS) de Microsoft, voir Figure 2.2 Il est également disponible gratuitement [12].
- **Borland JBuilder**
  IDE très complet de Borland, il existe une version gratuite. A noter, que son cousin Borland C# builder pour .NET peut également faire du J2EE [11].
- **IntelliJ IDEA**
  Un excellent IDE commercial [14].
- **JDeveloper**
  Développé par Oracle, il est diponible gratuitement [15].
- **Bloc-notes ou *notepad***
  Bien que fortement déconseillé, il reste tout à fait possible de développer une application J2EE grâce à un simple éditeur de texte [25].

4.5 Sécurité

4.5.1 Sécurité du code

Java est un langage destiné à fonctionner dans une machine virtuelle. Il bénéficie donc de toutes les protections que cette dernière peut procurer, comme par exemple, la gestion de mémoire (par ex. le ramasse-miettes *(garbage collector)*) qui rend caduques toutes les formes d'attaques de dépassement de capacité (*buffer overflow*). Pour les applications distantes, telles que les *applets*, les applications tournent dans une *sandbox* et n'ont des droits que très limités.

4.5.2 Gestion des droits d'accès

La gestion des droits d'accès se fait de manière déclarative et, tout comme .NET, peut définir des rôles pour donner des droits d'accès aux ressources du tiers métier. Dans le modèle de sécurité des EJB, on ne spécifie que les autorisations d'un utilisateur (c'est-à-dire les droits d'accès d'une ressource protégée). L'authentification proprement dite n'est pas spécifiée dans le modèle. Elle peut, par exemple, prendre la forme d'un *login* sur une page *web* (exemple de restrictions avec Struts : Listing 4.1). Dans cet exemple on voit que les pages /showActivity.do (ligne 3) et /showActivities.do (ligne 7) sont accessibles uniquement aux rôles User, Coach et Administrator (lignes 12 à 14).

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>Show activity page</web-resource-name>
        <url-pattern>/showActivity.do</url-pattern>
    </web-resource-collection>
    <web-resource-collection>
        <web-resource-name>Show activities page</web-resource-name>
        <url-pattern>/showActivities.do</url-pattern>
    </web-resource-collection>

    <auth-constraint>
        <role-name>User</role-name>
        <role-name>Coach</role-name>
        <role-name>Administrator</role-name>
    </auth-constraint>
</security-constraint>
```

Listing 4.1: Exemple avec Struts pour la restriction de ressources à des rôles

4.5.3 Transactions

Le J2EE supporte les transactions déclaratives[4] ou les transactions codées[5] au travers des EJB. L'avantage des transactions manuelles sont leurs performances, mais c'est là leur unique avantage. En effet, les transactions automatiques permettent de programmer plus rapidement, de générer un code plus lisible, d'éviter des bogues, et surtout de gérer le cas des transactions distribuées. C'est pourquoi, les développeurs utilisent le plus souvent les transactions automatiques. Concrètement, il suffit au programmeur de définir un champ XDoclet @ejb.transaction dans toutes les classes des *beans* où une transaction est nécessaire.

4.6 Aspect "politique"

Toutes les facettes de J2EE font partie de spécifications publiques. Les évolutions de Java et de J2EE (qui n'est rien d'autre qu'un ensemble de technologies Java) se font au travers des Java Specification Requests (JSR) qui sont publiques une fois finalisées et approuvées. Ces Java Specification Requests (JSR) sont rédigés par Sun Microsystems et la Java Community Process (JCP) dont tout le monde peut librement faire partie (les entitées commerciales ont une taxe annuelle). L'avantage de ce système est que les changements ne sont, en règle générale, pas aussi radicaux qu'ils peuvent l'être sur la plateforme de Microsoft, par exemple[6]. Par contre, la politique de Sun vise à conserver le contrôle sur l'évolution de Java et garde un droit de véto sur les décisions prises. Et, par exemple, a toujours refusé que Java devienne complétement *open-source*. Toutefois, Sun à mis certains paquetage (Java3D, par exemple) ou certaines applications (Netbeans, par exemple) en *open-source*.

En plus de fonctionner sur une multitude de plateformes et systèmes d'exploitation différents, il existe beaucoup d'entreprises différentes qui fournissent une implémentation du standard J2EE. Il est donc possible de changer à tout moment de fournisseur J2EE pour

[4]Ces transactions sont également appelées implicites, automatiques ou passives.
[5]Ces transactions sont également appelées explicites, manuelles ou actives.
[6]Les spécifications pour les EJB 3.0 en sont un mauvais exemple, le changement pour l'accès aux données est important, voir radical en comparaison avec la version 2.

une application (en pratique, il y a parfois quelques problèmes de compatibilité, mais la plus grosse partie du code est totalement compatible).

5

J2EE : Aspect pratique

5.1 Caractéristiques

Les principales caractéristiques de J2EE[1] sont :

- Un seul langage : Java ;
- Une grande variété de plateformes et de systèmes d'exploitation sont capables de faire fonctionner une application J2EE ;
- Il existe un grand nombre d'IDE différents pour le développement ;
- Il existe beaucoup d'implémentations de différentes entreprises ;
- Toutes les spécifications sont publiques.

[1] Bien évidemment, le J2EE a beaucoup d'autres caractéristiques importantes. Il s'agit ici des caractéristiques qui le différencie de son concurent .NET.

5.2 Prise en main

Dans cette section, il est surtout question de l'IDE Eclipse 3 [13], du *framework* Struts [1] (voir la section 5.3) et des EJB 2 synchronisés automatiquement avec la base de données (CMP, voir la section 5.5). Pour plus de détails sur l'implémentation avec ces outils et technologies, voir [Brü05].

5.2.1 Global

Le développement d'une application avec Struts se fait directement à la main. Il n'est pas question ici d'outils WYSIWYG. Le mappage des *entity-beans* à la base de données se fait également à la main. Il faut également créer le fichier *build* pour Ant [1] à la main. L'architecture est en plus relativement complexe et, encore une fois, à développer à la main.

5.2.2 Développement de l'interface *Web*

Le *framework* Struts est lourd à mettre en place. Il y a une multitude de fichiers de configurations différents, tous en XML. Une bonne introductions à Struts est donné dans [Cav04].

5.2.3 Développement de l'accès aux données

Le concept des *entity-beans* est simple à comprendre. La mise en place est par contre un peu laborieuse : il y a tout d'abord beaucoup de code à écrire (tous les `set` et `get` correspondants aux champs des tables de la base de données) ; ensuite, il faut chercher les bons *tags* XDoclet[2] et recréer toutes les relations des tables. Par contre, une fois cette étape passée, il n'y a pratiquement pas de requêtes SQL à implémenter (en dehors de quelques simples requêtes SQL du type "`SELECT`"), ce qui simplifie grandement la programation. En effet, le progammeur dispose d'une vue "objet" de la base de données. La mise en place de nouvelles fonctionnalitées (*use case*) se fait sans écriture de nouvelles requêtes SQL comme c'est le cas avec un modèle ensembliste (ADO.NET, par exemple).

5.2.4 Déploiement

Le déploiement d'application est simple. L'interface *web* se trouve dans une archive WAR alors que la partie métier se situe dans une archive JAR. Les deux archives sont ensuite empactées dans une archive EAR qui est l'archive à déployer. On a donc un seul fichier qu'il suffit de copier dans le répertoire de déploiement. La configuration de l'application se trouve dans différents fichiers XML pour décrire les droits d'accès, la page par défaut, etc.

[2]Les *tags* XDoclet sont en plus partiellement dépendant du serveur d'application utilisé.

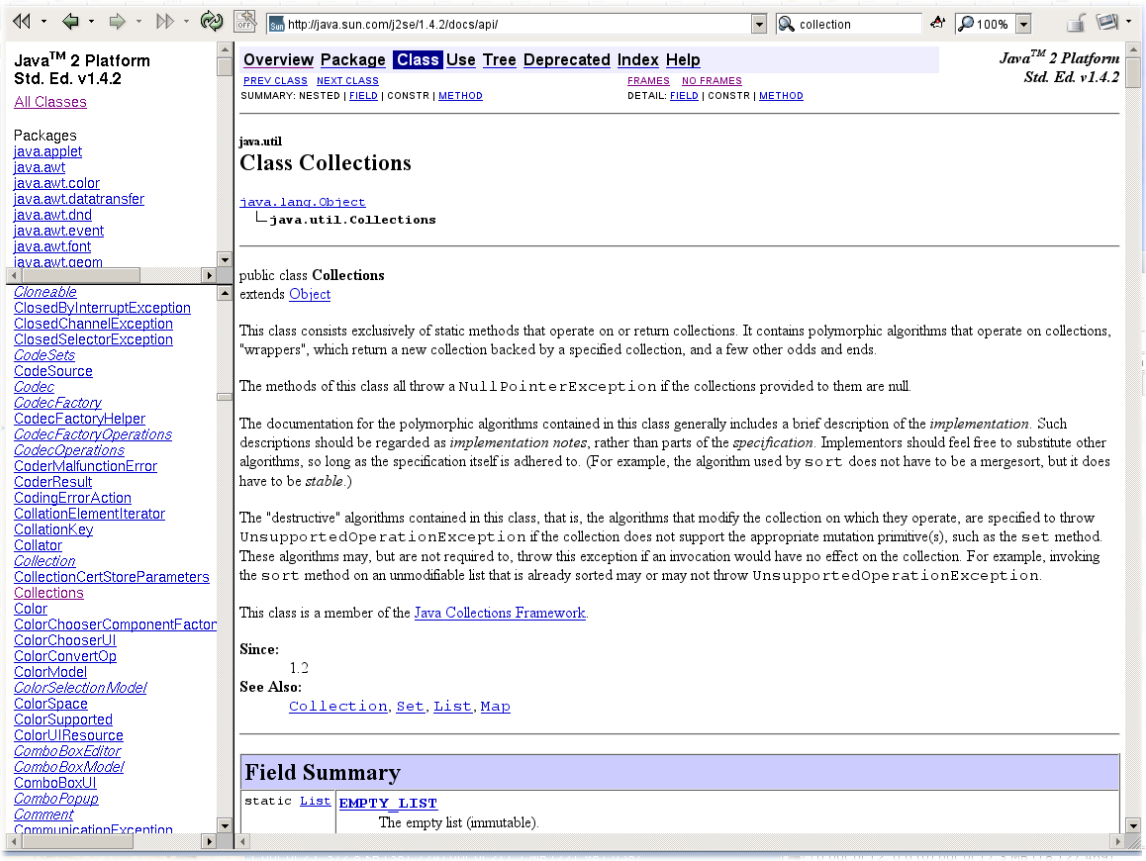

Figure 5.1: Documentation Java de la classe Collection

5.2.5 Déverminage

Le déverminage de la partie métier se fait grâce au fichier de *logging*. Ce dernier informe de toutes les étapes et signale toutes les erreurs. Pour la partie *web*, l'erreur apparaît, en règle générale, directement dans le navigateur en plus des fichiers de *logging*. Pour chaque modification, il faut impérativement recompiler et redéployer l'application. Pour les fichiers JSP (couche *web*), Eclipse ne possède aucun éditeur pour eux et il n'y a aucune vérification, que se soit au niveau du code ou au niveau du HTML. Il semble toutefois qu'il existe maintenant des *plugins* pour Eclipse 3 améliorant cette situation.

5.2.6 Langage

Le langage Java n'évolue que lentement (voir la section 4.6), ce qui permet de s'assurer que les programmeurs ne sont pas trop perdus à chaque nouvelle version. Plutôt que d'inventer ou créer des nouvelles structures de langage, Java utilise ce qui existe déjà. Un exemple, la syntaxe des *beans* (Listing 5.1). Java est extrêmement répendu et il est, en règle générale, assez facile de trouver des exemples sur le *web*. La documentation est également disponible sur Internet et facile à utiliser, voir Figure 5.1.

```
public class Person{
    String name;
    public String getName(){
        return this.name;
    }
    public void setName(String name)
        this.name = name;
    }
}
```

Listing 5.1: Exemple de *bean* en Java

5.3 Tiers présentation, Struts

Le développement d'applications à l'aide du *framework* Struts de la fondation Apache [1] se fait à la main. Les pages elles-mêmes peuvent être faites de manière graphique, mais pour la configuration, le développeur, devra les faire lui-même[3]. En plus des différentes pages *web* (les fichiers JSP) il y a "beaucoup" (en comparaison à ASP.NET) d'autres fichiers (configuration, validations de formulaires). Une grosse différence entre Struts et ASP.NET est la technologie des *webforms*[4]. Pour plus de détails, il existe une comparaison faite par Microsoft [71] (comparaison assez objective, mais bien entendu, pour la conclusion il faut garder en tête que c'est Microsoft). Il s'agit ici que de quelques exemples pour illustrer les différences et similaritées des deux approches (ASP.NET et Struts)

5.3.1 *Layout*

Pour l'interface graphique, une solution propre est de faire un fichier de *layout* pour définir la structure générale des pages. De cette manière, il n'y a que le contenu à adapter pour chaque page. Cette solution n'est pas imposée, c'est au programmeur d'y penser. La mise en place est plutôt facile, elle se fait à l'aide de deux fichiers (voir Listing 2.2 et Listing 2.1 sous-section 2.1.1).

5.3.2 Menu

Pour le menu, il existe également plusieurs possiblités. Une solution propre consiste à dédier un fichier JSP pour lui. Il suffit ensuite de mettre le menu en ajoutant au besoin des *taglibs*. Par exemple, le menu n'est pas le même pour un utilisateur authentifié, que pour un administrateur, il est donc nécessaire d'ajouter une logique, ce qui peut-être fait avec des *taglibs* du genre <req :isUserInRole role="User">. Il en va de même pour la langue.

5.3.3 Validation des formulaires

Pour la validation côté serveur des formulaires, Struts [1] propose l'utilisation de fichiers XML. C'est-à-dire que le champ et sa vérification ne se trouvent pas dans le même fichier (ni dans le même répertoire en pratique). Il y a tout d'abord le fichier `validator-rules.`

[3]En théorie, il est tout à fait possible de faire des outils graphiques dans le genre de Sun Java Studio Creator [12] ou VS [23] pour Struts. Il existe d'ailleurs du développement dans cette direction, notamment avec un *plugin* pour Eclipse [17].

[4]Ce "retard technologique" est maintenant comblé grâce au JSF.

```
package ejb.monApplication;
import javax.ejb.* ;
import java.rmi.* ;
public class MonComposant implements  SessionBean{
  public void ReserveHotel(int numeroChambre) {
            // Met à jour la base de données Hotel
  }
  //Methodes obligatoires dans tous les sessions-beans
  public void ejbActivate() {} ;
  public void ejbPassivate() {} ;
  public void ejbRemove() {} ;
  public void setSessionContext(SessionContext ctx) {} ;
}
```

Listing 5.2: Exemple de *session-bean*

`xml` fourni par Struts. Il contient les règles de validation, par exemple, pour un numéro de carte de crédit ou un nombre entier. Un deuxième fichier, `validation.xml`, contient lui le nom des champs à vérifier ainsi que son type de "valideur". S'il s'agit d'un mask il faut encore écrire l'expression régulière qui lui est associée. Pour la validation côté client, le programmeur doit se débrouiller seul.

5.3.4 Configuration

La configuration est la partie la plus délicate de Struts. Elle se fait à l'aide de deux fichiers, `web.xml` et `struts-config.xml`. Le premier s'occupe des contraintes de sécurités, des rôles, de la page d'accueil, etc. Le deuxième prend en charge toutes les actions. Il contient donc toutes les pages JSP avec les paramètres et actions (pour plus de détails, voir Chapitre 5 de [Brü05]).

5.4 Tiers métier

Le tiers métier se fait à l'aide des *session-beans*. Ce sont des objets en mémoire, non persistants. Il en existe deux types :

1. **Stateful Session Beans**
   Certaines conversations se déroulent sous forme de requêtes succesives. D'une requête HyperText Transfert Protocol (HTTP) à l'autre, il faut un moyen de conserver un état. Par exemple, pour des opérations bancaires, si le client effectue plusieurs opérations, on ne va pas à chaque fois lui redemander son numéro de compte. C'est la tâche de ces *session-beans*. Leur durée de vie est la durée de la session. C'est-à-dire le temps de connexion du client. Une fois qu'il se déconnecte, le/les *session-beans* peuvent être détruits par le conteneur.
2. **Stateless Session Beans**
   Certaines conversations peuvent se résumer à un appel de méthode, sans besoin de connaître l'état courant du *bean*, par exemple, un simple calcul ou une validation d'un numéro de carte. Le client passe toutes les données nécessaires au traitement lors de l'appel de méthode. Le conteneur est responsable de la création et de la destruction du *bean*. Il peut le détruire juste après un appel de méthode, ou le garder en mémoire pendant un certain temps pour réutilisation. Une instance de

stateless session bean n'est pas propre à un client donné, elle peut être partagée entre chaque appel de méthode.

Le conteneur se charge de toute la gestion des *session-beans*. Il se charge de les créer et de les détruire ainsi que de les passiver et de les activer. Il se charge aussi du *pooling*. Un exemple de *session-bean* est disponible sous Listing 5.2. On constate qu'en plus des méthodes désirées (dans l'exemple, il n'y en a qu'une), le programmeur doit encore écrire quatres méthodes supplémentaire. Heureusement des outils tel que XDoclet proposent la création automatique de ces squelettes de classes via des assistants.

5.5 Tiers données, EJB

Pour la persistance des données il existe plusieurs possiblitiés. Les Java Data Object (JDO) qui sont une API standard pour une abstraction de la persistance, les accès directs à des fichiers, JDBC ou les EJB au travers des *entity-beans* synchronisés avec la base de données, soit automatiquement (de manière déclarative, CMP), soit de manière programmative (Bean Managed Persistence (BMP)). Les *entity-beans* proposent d'accéder à la base de données de manière transparente en accédant simplement à des objets. Du point de vue du programmeur, il suffit d'appeler les méthodes *get* et *set* des *entity-beans* et d'écrire les requêtes SQL "SELECT" en EJBQL. C'est le conteneur qui se charge ensuite de faire le lien *entity-beans* - base de données. Les accès à la base de données se font de manière déclarative. Le programmeur n'a pas besoin de connaissances approfondies du SQL.

5.6 Utilisation des *design patterns*

L'utilisation des *design patterns* est fortement conseillée pour tous les avantages qu'ils procurent. Toutefois, certains *patterns* sont très verbeux et donc pénibles à mettre en place[5], et n'est, en règle générale, pas "dirigé" comme on le verra dans .NET[6].

5.7 Conversion d'un ancien (*legacy*) système

Java offre relativement peu de possibilités pour porter un ancien système vers J2EE. Il peut utiliser le protocole de transmission standard CORBA pour interagir avec d'anciens systèmes mais pour convertir le système en J2EE 100%, il faudra impérativement réécrire le code. Il existe parfois des outils de conversion, mais ils ne remplacerons jamais un développeur. Ils sont là pour faciliter la conversion tout au plus.

[5]Typiquement le *pattern Transfert object* nécessite beaucoup de code, bien qu'aucune logique métier.
[6]Typiquement les dataset utilisés en temps que *Transfert object*.

6
.NET : Aspect global

.NET est le nom donné à la plateforme de développement logiciel principale de Microsoft. C'est un regroupement de l'architecture interne de langages, de composants et d'outils [52]. La technologie .NET comporte trois grandes parties :

- un ensemble extensible de langages dont C#, VB.NET, J#, Delphi#, etc. Ces langages doivent respecter la spécification Common Language Specification (CLS) du Common Language Infrastructure (CLI) ;
- un ensemble de classes de base utilisées par les applications. C'est ce que l'on appelle le *framework* .NET ;
- une couche logicielle nommée CLI qui est responsable de l'exécution des applications .NET.

Le *framework* .NET propose une API objet de très haut niveau qui permet de créer des applications complexes plus rapidement. Le .NET respecte un bon nombre de standards d'organisations comme le World Wide Web Consortium (W3C)), l'Institute of Electrical and Electronics Engineers (IEEE), l'Organization for the Advancement of Structured Information Standards (OASIS), l'Internet Engineering Task Force (IETF) et la Web Services Inspection (WSI). Les spécifications du CLI et du langage C# sont publiques et

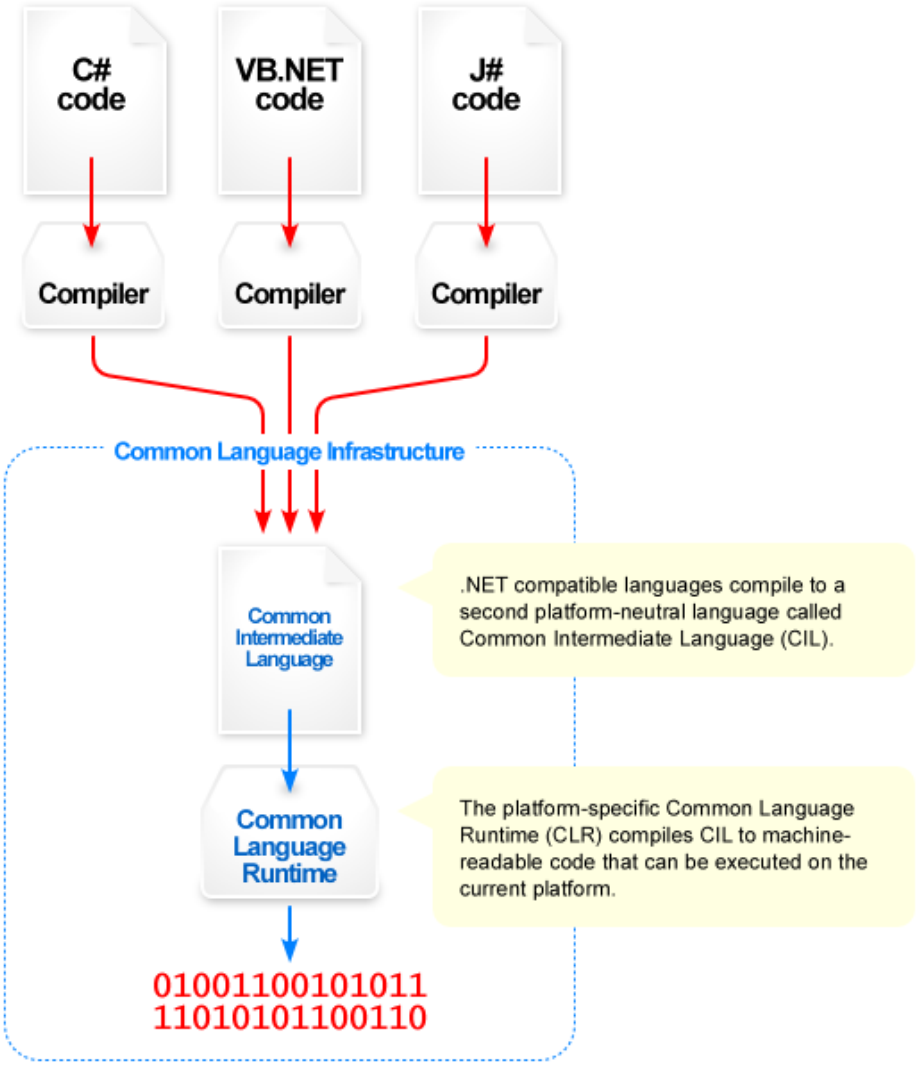

Figure 6.1: Vue d'ensemble du CLI (tirée de wikipedia.org)

ont été déposées chez un organisme indépendant, l'European Computer Manufacturers Association (ECMA)[1]. Elles ont également été déposées à l'IEEE[2].

6.1 Architecture

Les applications .NET, quel que soit le langage haut niveau utilisé, sont compilées dans un langage intermédiaire (Intermediate Language (IL)) appelé Common Intermediate Language (CIL)[3]. On parlera alors de code managé (l'IL) et non managé (le code machine). Le but est de transformer le code IL en code machine lors de l'exécution lorsque le système d'exploitation et le processeur sont connus. Une application .NET peut ainsi être déployée sur n'importe quelle plateforme compatible .NET sans être recompilée. On dit alors que les applications sont portables au niveau binaire.

[1]Les spécifications sont diponibles sous les numéros ECMA-334 (C#) et ECMA-335 (CLI) [64].
[2]Les spécifications sont diponibles sous les numéros ISO/IEC 23270 (C#) et ISO/IEC 23271 (CLI) [67].
[3]L'implémentation de Microsoft du CIL est le Microsoft Intermediate Language (MSIL).

Spécifications .NET	Implémentations MS	(Equivalent Java)
CLI	CLR	JRE
VES[a]	CLR	JVM
CIL[a]	MSIL	Bytecode Java
BCL[a]	CLR	JRE (sans JVM)
–	ADO.NET[b]	JDBC
–	Winforms[b]	Swing, AWT
–	ASP.NET[b]	JSF, JSP, Servlets
–	COM+[c]	JTA
–	MSMQ[c]	JMS
–	ADSI[c]	JNDI
–	Serviced Component (COM+)[c]	EJB Session Beans

[a]Cette spécification fait partie des spécifications du CLI.

[b]Cette implémentation fait parti intégrante du *framework* .NET, mais il s'agit d'une technologie propriétaire Microsoft, protégée par des brevets, dont les spécifications ne font pas parties du CLI et ne sont pas ouvertes.

[c]Cette technologie ne fait pas du tout partie du *framework* .NET mais du Système d'exploitation Windows Server 2000/2003.

Table 6.1: Les correspondances entre spécifications .NET et implémentations Microsoft

Quand on parle de l'architecture de .NET, on est vite perdu à cause du nombre impressionant d'abréviations. D'autant plus, que beaucoup sont très similaires dans leurs fonctionnalitées (IL, CIL MSIL ou encore, CLI et Common Language Runtime (CLR)) ou dans leur orthographe (CIL et CLI). Une grosse partie des confusions vient du fait qu'on mélange parfois les spécifications (CIL, CLI) et les implémentations de Microsoft correspondantes (MSIL, CLR). Par abus de langage, on parle souvent du "cas Microsoft" comme d'une généralité. Voir le tableau Table 6.1 pour plus de clarté[4] [70].

6.1.1 Machine Virtuelle (CLI / CLR)

Le Common Language Infrastructure (CLI) est **la spécification** de .NET (déposée sous la spécification ECMA 335) voir figure Figure 6.1. Dans le monde de Java, le CLI correspond en gros à JRE. Son but est de proposer un langage pour le développement d'applications avec, entre-autres, la gestion d'exceptions, un ramasse-miettes, la sécurité et l'interopérabilité. Il est composé de 5 parties principales :

- **Common Type System (CTS)**
  Cette partie s'occupe de la spécification des différents types possibles tels que, par exemple : les nombres entiers sur 8, 16 ou 32 bits, nombres à virgules flottantes, etc.
- **Common Language Specification (CLS)**
  Cette partie s'occupe des spécifications du langage lui-même, comme par exemple : pas d'héritage multiple, gestion automatique de la mémoire (ramasse-miettes), etc.

[4]Les correspondances Java sont approximatives et sont là pour faciliter la compréhension.

- **Common Intermediate Language (CIL)**
  Cette partie spécifie la forme que doit avoir le langage intermédiaire (*bytecode* dans le monde Java).
- **Just In Time (JIT)**
  Il s'agit d'un compilateur "juste à temps", voir la section 6.1.2.
- **Virtual Execution System (VES)**
  Le Virtual Execution System (VES) offre un environnement pour l'exécution du code managé, c'est-à-dire le support de l'ensemble des instructions du CIL. Il se charge donc du chargement et de l'exécution des programmes compatibles CLI.

Le Common Language Runtime (CLR) est le nom de **l'implémentation** de Microsoft du CLI. La machine virtuelle .NET de Microsoft (CLR) peut être comparée à la machine virtuelle Java, bien que cette dernière n'ait été conçue que pour le seul langage Java. Pour passer le code IL en code machine, le CLR va utiliser la technique de compilation JIT. A la différence de Java, le code IL n'est jamais interprété mais toujours compilé **avant** l'exécution. Le CLR compile des portions de code (méthodes) juste avant de les exécuter. Cette technique donne de meilleurs résultats en termes de performances que l'interprétation séquentielle d'instructions, voir la section 6.1.2.

Le CLR est capable d'appliquer bon nombre d'optimisations et peut générer du code optimisé pour un certain type de processeur, ou encore placer les variables les plus fréquemment utilisées directement dans les registres du processeur, ce qui accélère leurs accès. La version 2.0 de .NET améliore nettement les performances de l'outil NGen qui, lancé à la fin de l'installation, permet de compiler entièrement une application avec le compilateur JIT. Dans certains cas, la compilation JIT peut affecter les performances. Cet outil peut donc devenir intéressant bien que son utilisation désactive certaines optimisations appliquées en temps réel.

Le CLR gère également le chargement des applications et leur exécution. Les applications sont placées dans une *sandbox* qui les isole totalement des autres applications et du système. Ce point est développé plus en détails dans le chapitre dédié à la sécurité (voir la sous-section 6.5.1). La mémoire sous .NET est entièrement gérée par le CLR qui utilise un ramasse-miettes pour libérer automatiquement la mémoire inutilisée.

6.1.2 JIT

Le JIT se charge concrètement de compiler le code CIL en code machine, puis de l'exécuter. Il n'est pas une composante obligatoire de .NET, mais le CLR fonctionne de cette manière. D'autres, comme Portable.NET [50] utilise une approche différente (dans leur cas, une conversion du langage intermédiaire suivie d'une interprétation).

Le "*just in time*" a son importance : il ne s'agit pas ici de recompiler la totalité du programme en langage machine, ni d'en interpréter chaque ligne une à une. Pour obtenir de meilleures performances, c'est une troisième solution qui a été choisie : compiler le corps d'une méthode juste avant son appel. La compilation se fait donc juste à temps pour que l'exécution de la méthode en langage machine puisse avoir lieu. Le JITer compile donc le code en fonction des besoins [52].

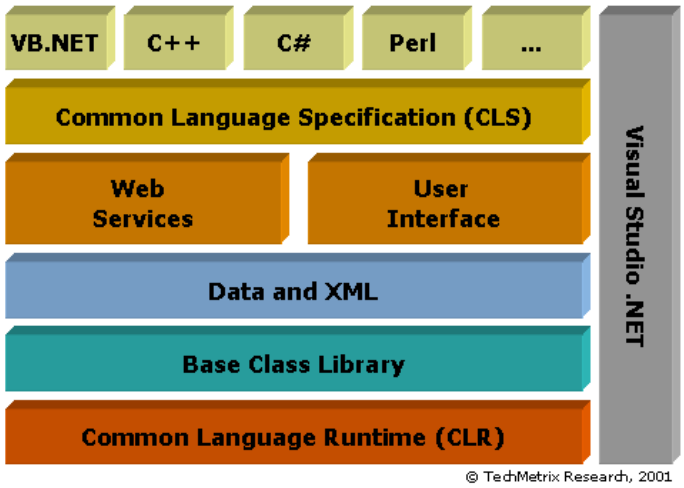

Figure 6.2: Structure du *framework* .NET (tirée de pcinpact.com)

6.1.3 Base Class Library (BCL)

Il y a trois couches entre le CLR et le CLS, voir la figure Figure 6.2, chacune comportant des classes apportant des fonctionnalités de plus en plus évoluées. Seule la couche Base Class Library (BCL) fait partie des spécifications de l'ECMA. Les deux autres n'en font pas partie. Ce sont des composantes propriétaires de Microsoft pour la version commerciale de .NET. Les implémentations par des entreprises tierces (par exemple, Mono [57], DotGNU [50]), en plus des difficultés techniques, peuvent même soulever des problèmes légaux [52], [61].

- **Première couche, BCL**
  La BCL, est une bibliothèque de types disponible pour tous les langages qui utilisent le Framework .NET. Elle comporte des fonctions telles que la lecture et l'écriture de fichiers, l'interactions avec les bases de données, la manipulation de chaînes de caractères (*string*) (qui, en .NET ne sont pas des objets comme en Java), de collections (liste), de graphiques, de documents XML, etc.
- **Deuxième couche, ADO.NET, XML**
  La deuxième couche est composée de deux bibliothèques de classes d'accès aux données. Tout d'abord, ADO.NET, permettant l'accès sous format XML aux interfaces de bases de données SQL Server [10], ODBC, OLE DB et Oracle [9], et bien sûr aux fichiers XML. La deuxième bibliothèque de classes regroupe les *XML Classes* qui permettent de manipuler les données XML. On y trouve, par exemple, les classes eXtensible Stylesheet Language Transformations (XSLT) permettant la transformation d'un document XML vers n'importe quel autre type de document.
- **Troisième couche, ASP.NET, *winforms***
  Et enfin, la troisième couche, est utilisée pour la création de *Web Services*, de pages *Web*, et d'applications Windows. Les deux premiers forment ASP.NET (les interfaces *web*) et le dernier forment les *winforms* (les interfaces clientes riches).

6.1.4 Annotations (méta-données)

Tous les fichiers CIL (les fichiers `.exe` ou `.dll`) s'auto-décrivent au travers d'annotations .NET[5]. Le CLR s'assure que la méthode correcte est appelée. Les annotations sont généralement générées par le compilateur, mais le développeur peut créer ses propres attributs [61].

6.1.5 Gestion des bibliothèques partagées (*assemblies*)

L'unité de base d'une application .NET est appelée un assemblage ou une assemblée *(assembly)*. Il s'agit d'une bibliothèque de codes compilés en code IL, c'est-à-dire d'un ensemble de codes, de ressources et d'annotations. Un assemblage est toujours accompagné par un manifeste (*assembly manifest*) décrivant son contenu : nom, version, types de données exposées, autres assemblages utilisés, instructions de sécurité. Un assemblage est composé de un ou plusieurs modules qui contiennent le code.

Sous Windows on les trouve sous deux types : les `.EXE` et les `.DLL`. On sait que des bibliothèques de code standard peuvent être partagées par plusieurs applications, ce qui peut induire un problème récurrent, connu en général sous le nom poétique "d'enfer des DLL" : certaines mises à jour de biliothèques partagées peuvent empêcher d'autres applications de fonctionner suite à des changements de code. Cela a été plus ou moins réduit par les modules d'installation récents mais on en trouve encore la présence.

L'environnement .NET propose une solution intéressante : il est capable de charger des assemblages de différentes versions. L'application utilisera la version pour laquelle elle a été prévue. Pour différencier les assemblages, .NET utilise la méthode des noms forts (sorte d'identificateur unique). Un nom fort est constitué de quatre entités : le nom de l'assemblage, sa version, sa culture (qui est optionnelle), et sa clé publique (qui est également optionnelle)[6]. Le répertoire Global Assembly Cache (GAC) permet de stocker les assemblages partagés grâce au nom fort qui les identifie de façon unique [49], [52], [61].

6.1.6 Remarques

L'utilisation du XML est omniprésente et permet une meilleure interopérabilité des applications. Par exemple, la méthode de sauvegarde de la configuration passe par l'utilisation d'un fichier XML et rend obsolète la base de registres. Alors que sur Windows XP, le *framework* .NET repose sur l'API Win32 (typiquement les *winforms*), il en est totalement indépendant sous Vista (la prochaine version majeur de Windows dont la sortie est prévue début 2007, aux dernières nouvelles), ce qui devrait améliorer les performances. Sous Windows Vista, .NET devrait être au coeur même du système d'exploitation [58].

[5] Les annotations comprennent entre autres : le nom de version, la langue, le nom du produit, le copyright, le nom de la compagnie, la signature numérique, etc.

[6] La clé publique est utilisée si l'assemblage est fait pour être partagé. Elle est obligatoire pour l'utilisatation de COM+ (transactions).

6.2 Implémentations existantes

Pour le moment, le *framework* .NET est principalement implémenté pour les plateformes Windows mais cela est voué à changer dans le futur, et l'on peut déjà trouver le projet Mono [57] sur Linux (implémentation open source du *framework*). Il existe également le projet Rotor [60] de Microsoft (en shared source) et enfin, on peut encore citer le projet DotGNU Portable.NET.

- **Microsoft .NET**
  Microsoft étant l'auteur de la technologie .NET, leur implémentation est (en tout cas pour l'instant) la référence. C'est-à-dire que les autres implémentations tierces doivent s'assurer que leur code intermédiaire IL généré est bien compatible avec celui de Microsoft et que le MSIL est compatible avec leur implémenation du CIL. On peut télécharger (gratuitement) leur *framework* pour Windows à cette adresse [53][7].
- **Shared Source CLI ou Rotor**
  Rotor ou Shared Source Common Language Infrastructure (SSCLI) est le code source d'une implementation fonctionnelle des spécifications de l'ECMA [64] pour le CLI et le langage C# faite par Microsoft (MS). En plus des standards de l'ECMA il offre un compilateur .NET C# ainsi que .NET JScript, un dévermineur, des outils, une Platform Adaption Layer (PAL), des suites complètes de tests, etc. Par contre, certaines parties à valeur ajoutée de la version commerciale, comme ASP.NET ou ADO.NET n'en font pas partie. Le principal but de SSCLI est pour l'étude (académique) et la recherche. Il fonctionne sous MS Windows XP, FreeBSD, Mac OS X 10.x et de manière non-officielle, sous GNU/Linux [60]. La licence est une licence de partage de sources, non-commerciale.
- **.NET Compact Framework**
  Il s'agit de l'implémentation commerciale de Microsoft pour les appareils portables (PDA, téléphones mobiles).
- **Mono**
  Mono [57] propose une implémentation multi-plateforme du *framework* .NET selon les spécifications de l'ECMA. Il implémente en plus les ajouts de Microsoft pour sa version commerciale qui ne font pas partie des spécifications de l'ECMA, entre autres : ADO.NET, ASP.NET, les *winforms* (System.Windows.Forms), les classes XML (System.XML), etc.
  Le code (par exemple, C# ou VB.NET) s'écrit et se compile avec Mono de la même manière que sous .NET. Cependant toutes les biliothèques n'ont pas été entièrement portées sous Mono. Les *winforms*, par exemple, n'ont pas été complétement portées car elles sont très fortement liées à l'architecture de Windows, elles ne font pas partie des spécifications et il y a des problèmes légaux. A la place, Mono propose d'utiliser les bibliothèques GTK#2.0 pour l'interface graphique (mais les *winforms* restent diponibles).
  La programmation est indépendante du système d'exploitation / machine (Il est actuellement supporté par GNU/Linux, Windows, FreeBSD, Solaris et Mac OS X). Mono est également en retard dans les spécifications. Actuellement la version supportée est la version 1.1 alors que Microsoft a déjà sorti la 2.0. Le support de cette dernière est

[7] Contrairement à Java, il n'y pas deux versions (JRE et Java Development Kit (JDK)), mais qu'une seule. Le *framework* permet donc l'exécution de fichiers CLI ainsi que la compilation de fichiers sources (C#, VB.NET, etc.).

prévu pour la fin de l'année 2006. Le nombre de langages supportés, n'est pas aussi important que sous la plateforme de Microsoft mais leur nombre s'agrandit[8].
Pour l'ASP.NET le serveur *Web* peut être, soit celui de Microsoft Internet Information Services (IIS), soit celui d'Apache [1] avec le module Mono, soit le mini serveur *Web* (XSP) fourni avec Mono, soit encore le mini serveur (gratuit) de Microsoft (Cassini) [48].
Et enfin, Monodevelop, qui est en fort développement, est l'IDE naturel pour Mono et il est le seul à proposer nativement le GTK#. On peut également travailler avec Eclipse et le plugin C# ou encore avec VS puis Mono pour la compilation et l'exécution.

- **DotGNU Portable.NET**
  DotGNU Portable.NET [50] est une implémentation du CLI (appelé "ilrun", l'équivalent du CLR de Microsoft) y compris tout le nécessaire pour compiler et exécuter des applications C et C# qui utilisent la BCL, le XML et les *winforms* (Systems.Windows.Forms). Le but de cette implémentation, tout comme Rotor, est de respecter à 100% le CLS. Ce qui ne comprend donc pas les bibliothèques supplémentaires fournies par Microsoft pour la version commerciale (par exemple, ADO.NET et ASP.NET) qui elles, n'en font pas partie.

6.3 Serveur d'application, tiers métier

Le serveur d'application de .NET est directement le système d'exploitation Windows Server 2003[9]. Ce système d'exploitation s'accompagne du *framework* .NET et des *middlewares* sous-jacents (IIS[10], COM+[11] et MSMQ Microsoft Message Queuing (MSMQ)[12]). L'ensemble, constitue une base technologique dont le rôle est analogue à celui d'un serveur d'applications [70], [46], [73] et [74].

Pour les autres implémentations, par exemple, Mono, il n'y a pas à proprement parler de serveur d'application. La partie *web* tourne sur un serveur *web* tel que celui d'Apache, et les parties métier et accès aux données sont tout simplement exécutées avec Mono.

6.4 Outils de développement, IDE

L'outil de développement de .NET par excellence est Visual Studio (VS) de Microsoft. Toutefois, il n'est pas sans concurence, et d'autre IDE ont fait leur apparition, notamment Borland C# Builder qui n'a rien à envier à son concurent de chez Microsoft. Voici une liste (non-exhaustive) des IDE pour le développement avec .NET.

- **Visual Studio (VS) de Microsoft**
  Cet IDE très complet [23], voir Figure 6.3, s'occupe du développement d'application *web*

[8] Mono supporte actuellement une dixaine de languages de programmation différents [55].
[9] Windows Server 2000 peut également prendre en charge le .NET et donc faire office de serveur d'application.
[10] IIS est le serveur *web* (propriétaire) de Microsoft. Il prend en charge les applications ASP.NET.
[11] COM+ est une technologie (propriétaire) de Microsoft datant d'avant .NET. Il subsiste toutefois en tant que *middleware* assurant la gestion des transactions, l'activation des objets, et le partage de ressources (connexions à la base de données, par exemple).
[12] MSMQ est une technologie de communication (propriétaire) de Microsoft qui permet à des applications sur différents serveurs de communiquer. Il peut aussi le faire nativement en XML ou SOAP.

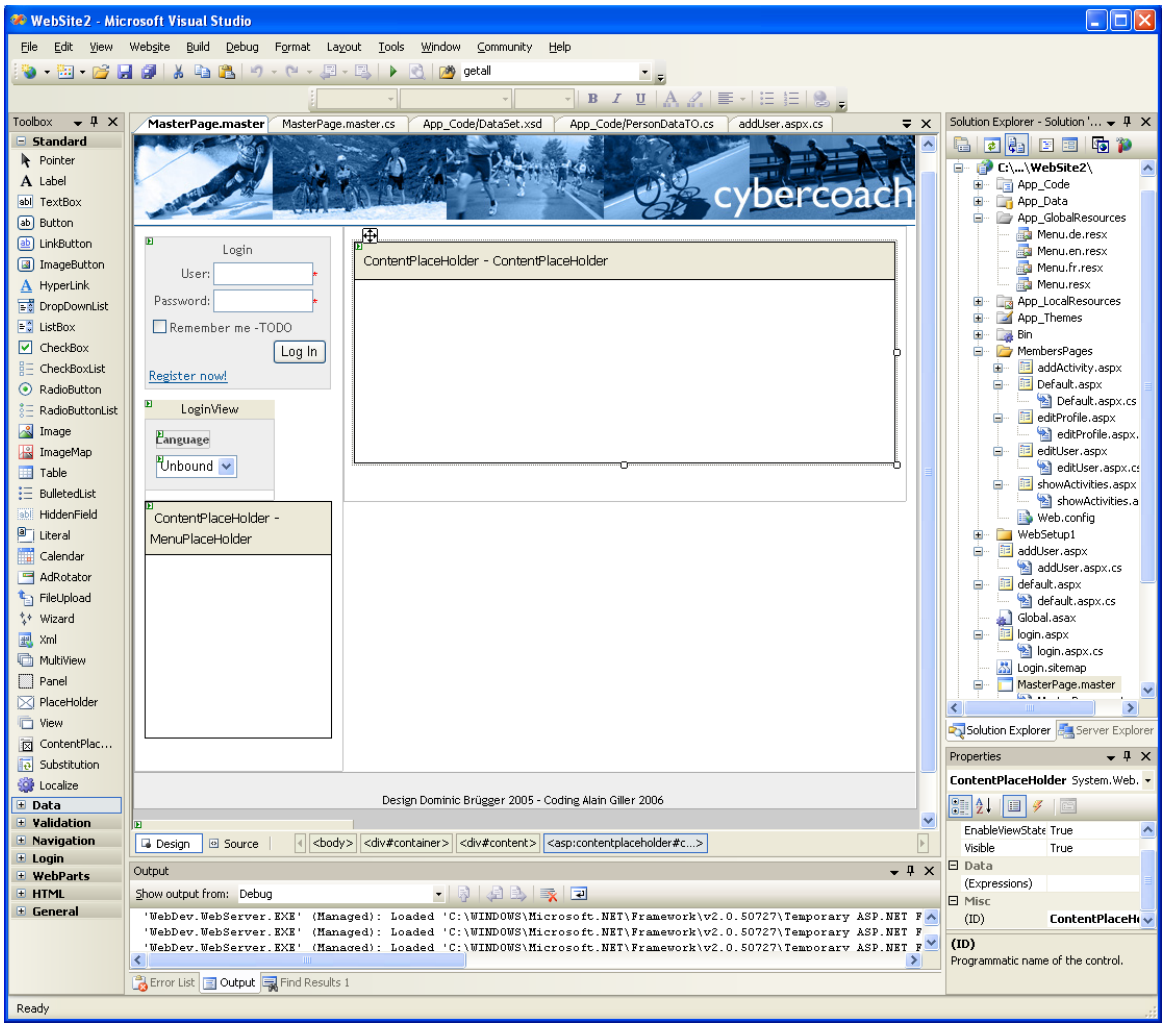

Figure 6.3: Fichier MasterPage.master en mode *design* avec Visual Studio 2005

ou Windows, du déverminage (*debuggage*) inter-langage, du déploiement d'applications *web*, de la création des fichiers de ressources, de la gestion des bases de données, de l'édition de fichier XML, etc. Il n'existe pas de versions gratuites en dehors des versions beta, pour les étudiants et collaborateurs des universités et hautes écoles ou des versions d'essai pour 60 jours.

- **Borland C# Builder**
  Cet IDE [19] est très semblable à VS du point de vue des fonctionnalités. L'interface est également suffisament proche de VS pour ne pas dérouter un habitué de ce dernier. Pour les usagers des produits de Borland, ils trouveront tout de suite leurs marques et outils (par exemple, Together pour faire le design de l'application en Unified Modeling Language (UML)). Et, contrairement à ce que le nom pourrait laisser croire, il supporte de manière intégrée également d'autres langages (C, C++, C#, Delphi, Java). Il intègre également de manière native la technologie J2EE et CORBA. Un autre avantage de Borland est qu'il supporte nativement plusieurs bases de données (VS est très orienté SQL server) comme, entre autres : Borland InterBase [6], dBase [5], ADO, MySQL [7], SQL Server [10], Access [4], etc. Il existe une version moins puissante mais gratuite pour les développements non commerciaux. Une comparaison des deux IDE (VS, Borland

C# Builder) est disponible sous [76].

- **Visual Studio Express**
  Il s'agit d'une version allégée de son grand frère Visual Studio (VS) également produit par Microsft [24]. Il est dédié au développement avec les *winforms* (ou pour la console). Il est gratuit jusqu'en novembre 2006. Il existe des versions pour le C#, J#, VS.NET et C++, chaque version ne support qu'un seul langage.
- **SharpDevelop**
  Un IDE gratuit et open-source pour Windows, surtout pour le développement de *winforms*. Le développement de *winforms* se fait graphiquement à la manière de VS. Il supporte le C# et le VB.NET [22].
- **Monodevelop**
  Monodevelop [21] se base sur SharpDevelop et, tout comme lui, il est *open-source* et gratuit. L'avantage par rapport à SharpDevelop [22], est qu'il est multi-plateforme. Il supporte actuellement le C#, VB.NET, Boo [47] et Java (grâce au projet IKVM [54]).
- **Visual Web Developer (remplace Web Matrix)**
  Visual Web Developer [24] est un produit de Microsoft pour le développement ASP.NET. Il est très léger (<3MB), gratuit et supporte C# et VB.NET comme langage et SQL Server [10] et Access [4] comme base de données. Il est désigné pour les non-professionels.
- **Eclipse avec le plugin C#**
  Cette solution permet de travailler avec le langage C# sous Eclipse. On peut compiler, mais évidement la plupart des outils disponible sous VS ou Borland ne sont pas disponibles. Même le déverminage est très limité [20].
- **Bloc-notes ou *notepad***
  Bien que fortement déconseillé, il reste tout à fait possible de développer une application .NET grâce à un simple éditeur de texte [25].

6.5 Sécurité

6.5.1 Sécurité du code

Il y a deux mécanismes pour assurer la sécurité du code. Le premier est le Code Access Security (CAS) qui permet d'assigner des permissions de code à une application .NET. Ceci est possible car les applications .NET tournent dans une machine virtuelle (CLR) et sont totalement isolées. Le deuxième mécanisme, est une vérification et validation du code IL avant l'exécution. Quand un assemblage est chargé, le CLR effectue plusieurs tests, notamment une validation et une vérification. Pour la validation, il contrôle que l'assemblage contient des annotations et un CIL valides et que les tables internes soient valides. La vérification n'est pas aussi exacte, il va contrôler si le code fait quelque chose qui est considéré comme non sûr (*unsafe*) (par exemple, l'accès par pointeurs avec le langage C pour .NET). Le code non sûr ne sera exécuté que si l'assemblage a explicitement la permission de ne pas effectuer de verification (*skip verification*), ce qui signifie généralement que le code est installé sur la machine locale.

Il existe environ une trentaine de permissions pour contrôler ce à quoi peuvent accéder les assemblages (fichier .exe ou .dll .NET). Le CAS permet aussi de rajouter ses propres permissions, parmi lesquelles on trouve l'accès à la base de registre, aux fichiers, aux

Domain Name System (DNS), au réseau ou au *web*. Le grand nombre de permissions fournit une granularité fine sur la sécurité. Ces permissions s'accordent par des polices de sécurité à partir de preuves fournies au CLR par l'assemblage. Il existe huit types de preuves :

- le répertoire dans lequel est stocké l'assemblage ;
- si le répertoire est stocké dans le répertoire GAC contenant les assemblages partagés par plusieurs applications ;
- si l'assemblage provient d'un site *web* particulier ;
- si l'assemblage a été fourni à partir d'une adresse HTTP ;
- la zone d'où provient l'assemblage : l'Internet, un site de confiance ou un site sensible, intranet local, disque dur, etc.
- si l'assemblage a été signé par un certificat "Authenticode". Cela détermine ainsi l'éditeur de l'application ;
- si l'assemblage possède un nom fort (voir 6.1.5) ;
- si l'assemblage fournit une valeur de hachage (hash). Cette valeur est un "résumé" de tout le code de l'assemblage qui permet de fournir une identité unique de l'assemblage. Si le code est modifié, ne serait-ce que légèrement, le hash calculé sera totalement différent.

Si une application .NET provient de la zone Internet, elle n'aura pas accès à la base de registre ou au disque dur. Le système lui fournira par contre une permission de stockage isolé et le CLR un répertoire associé à l'application pour stocker des données. A noter qu'on pourra attribuer également un quota pour l'espace utilisé.

Le compilateur de Visual Studio (VS) 2005 permet d'analyser le code de l'application et de déterminer les permissions minimales nécessaires pour l'application. Quand un assemblage est signé avec un nom fort, le CLR vérifie son identité avant chaque chargement. Si l'application a été infectée par un virus ou un *malware*, elle ne se lancera pas, car son identité aura été modifiée. L'isolation dans une machine virtuelle permet également d'éviter les attaques qui exploitent les failles de dépassement de tampon (*buffer overflow*) [59].

6.5.2 Gestion des droits d'accès

Plusieurs mécanismes de sécurités interviennent pour les droits d'accès à des ressources ou pages HTML. La gestion des droits d'accès se spécifient dans le fichier Web.config. Il peut y avoir plusieurs fichiers Web.config, un par répertoire. Dans ce fichier, on spécifie la façon de se *logger*, et les droits ainsi que les rôles (utilisateur authentifié, non authentifié, administrateur, etc.), voir Listing 6.1 lignes 6 à 14 pour les rôles, 32 à 35 pour la façon de s'authentifier.

La logique métier se trouve dans le répertoire App_Code[13] qui n'est pas accessible depuis le *web*. Enfin, pour afficher des informations provenant de la base de données, on utilise des *webforms*, tel des gridview qui vérifient automatiquement si l'utilisateur à le droit d'accès aux informations (par exemple, s'il est authentifié).

[13] Ce nom est imposé, mais peut néanmoins être modifié.

```
<configuration>
    <connectionStrings>
      <add name="MySqlCybercoachDB" connectionString="Server=localhost;Database=cybercoachdb;
          Uid=cybercoach;Pwd=cybercoach1234;"/>
    </connectionStrings>
    <system.web>
    <roleManager defaultProvider="MySqlRoleProvider" enabled="true" cacheRolesInCookie="true"
        cookieName=".ASPROLES"
        ... >
      <providers>
        <clear />
        <add
            .../>
      </providers>
    </roleManager>
    <globalization culture="auto:fr-CH" uiCulture="auto:fr-CH"/>
    <profile defaultProvider="MySqlProvider">
      <providers>
        <add
          name="MySqlProvider"
          type="System.Web.Profile.SqlProfileProvider"
          connectionStringName="MySqlCybercoachDB" />
      </providers>

      <properties>
        <add name="PreferredCulture" type="System.String"  allowAnonymous="true"/>
        <add name="ZipCode" allowAnonymous="true" />
      </properties>
    </profile>

    <anonymousIdentification enabled="true" cookieName=".ASPXANONYMOUS" cookieTimeout="43200"
        cookiePath="/" cookieRequireSSL="false" cookieSlidingExpiration="true"
        cookieProtection="All" cookieless="UseDeviceProfile"/>

    <authentication mode="Forms" >
      <forms loginUrl="default.aspx"
        name=".ASPXFORMSAUTH" />
    </authentication>
```

Listing 6.1: Extrait d'un fichier `Web.config`

```
[Transaction(TransactionOption.Required)]
public class TellerBean : System.EnterpriseServices.ServicedComponent{ ... }
```

Listing 6.2: Exemple d'une transaction implicite avec .NET

```
bool IsConsistent = false;
using (System.Transactions.TransactionScope transac = new System.Transactions.TransactionScope
    ()){
      SqlConnection conn = newSqlConnection(CONNECTION_STRING );
      SqlCommand cmd = newSqlCommand("DELETE Profile", conn);
      conn.Open();
          try{
              cmd.ExecuteNonQuery();
              IsConsistent = true;
          }catch (SqlException ex) {
              IsConsistent = false;
              //You can specify additional error handling here
          }finally{
              conn.Close();
          }
          //Since this was set to false originally it will only commit if it worked.
          transac.Consistent = IsConsitent;
}
```

Listing 6.3: Exemple d'une transaction explicite avec .NET

6.5.3 Transactions

Tout comme pour J2EE, il existe 2 types de transactions. Les déclaratives et les codées. Les avantages et inconvénients sont les mêmes que pour le J2EE (voir la sous section 4.5.3).

Pour les transactions automatiques, l'approche consiste à utiliser le moteur transactionnel COM+, accompagné de Microsoft Distributed Transaction Coordinator (MSDTC). Le principe est d'encapsuler son code dans un composant COM+ afin que ce dernier, en collaboration avec MSDTC, assure un comportement transactionnel de façon entièrement transparente. Il ne reste au développeur plus qu'à définir la transaction en mettant un attribut[14] sur une page ASP.NET, sur une méthode d'un *web service* ou sur une classe (voir Listing 6.2). Une fois que l'objet est marqué pour participer à la transaction, il va être automatiquement exécuté à l'intérieur de la transaction.

Pour les transactions manuelles, le développeur doit démarrer la transaction, ajouter les autres transactions, effectuer ou annuler la transaction et enfin la terminer (voir Listing 6.3).

6.6 Aspect "politique"

Les spécifications de l'ECMA ne couvrent que le langage C# et le CLI. Certaines composantes de .NET comme ASP.NET, les *winforms*, ADO.NET, les classes XML n'en font absolument pas partie. Il n'y a donc ni spécifications, ni code sources à disposition et elles sont même protégées par des brevets. Autrement dis, pour la plupart des applications en .NET, l'aspect "standards ouverts" n'est qu'un leurre. Et, bien qu'il existe des

[14] Pour pouvoir utiliser les transactions grâce à COM+, l'assemblage doit absolument comporter un nom fort, c'est-à-dire être signé, voir la sous section 6.1.5.

implémentations d'entreprises tierces (par exemple, Mono, DotGNU), celles-ci peuvent soulever des problèmes légaux. Il y a d'ailleurs beaucoup de discussions pour savoir si Microsoft peut détruire le projet Mono avec ses brevets. On peut affirmer, qu'à l'heure actuelle, les développements d'application avec .NET poussent à une dépendance totale envers les produits de Microsoft. Le système d'exploitation (Windows, Windows Server pour IIS), la base de données (SQL Server [10]), le serveur *web* (IIS) et le *framework* sont des produits de Microsoft.

Bien que Microsoft aie déposé les spécifications du .NET, rien ne garantit qu'à l'avenir, une nouvelle version du *framework* soit toujours publique. Ou encore, que Microsoft ne suive pas ses propres spécifications pour son implémentation comme il l'a déjà eu fait par le passé avec, par exemple, le format de texte Rich Text Format (RTF). De plus, Microsoft est connu pour faire parfois des changements assez radicaux sans trop se soucier des utilisateurs.

7

.NET : Aspect pratique

7.1 Caractéristiques

Une caratéristique importante de .NET est la volonté de Microsoft d'uniformiser tous ses produits. Les outils de développements ou la documentation sont similaire qu'on développe une page *web* ou une application Windows. Pour l'interface elle-même, le développement est très similaire (notion de *webforms* pour les page *web* et *winforms* pour les applications Windows ou client riche). Quand la vague .NET est arrivée, plusieurs produits se sont vus appeler ".NET" et ceci même s'il n'y avait pas de rapport avec le *framework* (par exemple, .NET passport, il était même question d'appeler Windows Server 2003, Windows .NET).

Une autre caractéristique est que si on développe une application grâce au *framework* de Microsoft en utilisant l'ASP.NET, les *winforms*, ADO.NET ou encore la gestion du XML, il faudra impérativement utiliser une machine Windows pour le développement et

pour le fonctionnement. L'application ne pourra pas être portée vers d'autres plateformes. Les alternatives (Mono) n'offrent, en tout cas pour l'instant, pas de réelle solution viable commercialement (voir la section 6.6).

Et bien sûr, une caractéristique fondamentale de .NET, est la possibilité d'utiliser une multitude de langages différents, et ceci même dans un seul projet. Ce qui donne des avantages très importants en cas de mise à jour d'un ancien système (voir la section 7.7).

7.2 Prise en main

Pour cette section, il est principalement question de l'IDE de Microsoft, Visual Studio (VS) 2005, ASP.NET et ADO.NET.

7.2.1 Global

L'interface de VS [23] est une belle réussite. La prise en main est plutôt aisée. La première "difficulté" quand on ne connaît pas du tout .NET, est de choisir le bon projet. En effet, il existe une multitude de choix possibles, et celui de création de page *web*, ne se trouve pas avec les autres. On se fait toujours "guider" ou diriger par l'outil. Par exemple, la partie métier "arrive toute seule", tout le code qui se trouve dans le répertoire App_Code est invisible depuis le *web* et correspond à la logique métier.

7.2.2 Développement de l'interface *Web*

La création de pages *web* est intuitive et se fait à l'aide de simple glisser-déposer avec la souris. Bien évidemment, il faut quand même ensuite aller dans le code pour mettre en place la logique de la page[1]. Le choix des *webforms* est, en règle générale, une chose aisée, bien que certaines nécessites un apprentissage. La mise en place des contrôleurs de pages (dans le sens du *pattern* MVC)[2] se fait également de manière intuitive grâce au fichier code (`.cs`) lié à chaque fichier *web* (`.aspx`). VS fait également de la "complétion" automatique pour le XML en plus des différents langages qu'il supporte, ce qui est un gain de temps appréciable.

7.2.3 Développement de l'accès aux données

La difficulté de l'accès aux données dépend passablement de la base de données utilisées. S'il s'agit de la base de données de Microsoft (SQL Server [10]), beaucoup de choses se font très facilement, voir automatiquement (par exemple, la gestion des rôles et des profiles, la création d'un DataSet fortement typé, etc.). Par contre, avec l'utilisation de MySQL [7], les choses se compliquent. Nativement, la gestion des rôles n'est pas possible (il faut créer soi-même une interface), la gestion des profiles est très limité (impossible de spécifier la langue, par ex.) et la création d'un fichier XSD n'est pas possible[3]. Sinon, l'utilisation des

[1] Il est également possible de tout faire graphiquement, à l'aide des propriétés des *webforms*, mais cela revient au même que de le faire dans le code.
[2] Il s'agit concrètement de la partie dynamiques des pages.
[3] L'utilisation de DataSet fortements typés est tout de même possible en utilisant ODBC à la place du connecteur natif, voir la section 7.5.

DataSet est très efficace et pratique. La récupération de données vers un DataSet se fait très simplement. Par contre, l'insertion de données demande l'écriture de requêtes SQL pouvant être relativement complexes[4]. La mise en place de nouvelles fonctionnalitées (*use case*) se fait presque obligatoirement en écrivant de nouvelles requêtes SQL. Le programmeur doit donc bien maîtriser le SQL et le modèle relationnel. Pour plus de détails, voir la section 7.5.

7.2.4 Déploiement

En cas de test, la procédure est extrêmement simple, il suffit de cliquer sur "*run*" et l'application va être vérifiée, compilée et déployée. Sinon, en cas de déploiement sur un serveur *web* IIS, il faut lui indiquer le chemin de l'application. Si en plus, l'application utilise COM+, il faut y enregistrer les parties qui utilise COM+, ce qui n'est pas forcement évident. Pour Mono, si on utilise le serveur *web* Apache [1], il suffit de configurer ce dernier pour fonctionner avec .NET puis de lui indiquer le chemin d'accès à l'application.

7.2.5 Déverminage

Le déverminage s'effectue de manière assez similaire à J2EE. Au moment de lancer une application (déploiement), il y une vérification du code qui s'effectue. En cas d'erreurs, le programmeur en est averti de la même manière que pour Java, à ceci près, que les erreurs sur les pages `.aspx` sont également décelées. Il peut s'agir d'erreurs dans les *webforms*, dans les *scripts*, ou même d'HTML[5]. Une fois l'application déployée, il peut encore survenir des erreurs. Suivant le mode de déploiement (mode normal ou mode déverminage) l'application va générer une exception et s'arrêter ou montrer la ligne qui pose problème, pour que le programmeur puisse modifier le code en conséquence[6], et continuer de tester sans redéploiement. Les erreurs apparaissent directement dans la console de VS. Il est également possible de faire du déverminage pas à pas.

7.2.6 Langage (C#)

Le C# est très ressemblant au Java, et les habitués de ce dernier s'y retrouvent tout de suite. Les différences sont parfois subtiles, comme par exemple[7] :

- les *beans* (voir Listing 7.1 et Listing 7.2) ;
- le nom du fichier et le nom de la classe ne sont pas liés (contrairement à Java) ;
- il n'y pas de notion de package mais de namespace (et contrairement à Java il ne représente pas une arborecence physique mais logique) ;
- l'héritage se fait se code avec " :", exemple : public class ServiceFacade : ServicedComponent ;
- pour faire un *override*, on doit explicitement le signaler à l'aide du mot clé override.

Toutefois, le plus difficile **n'est pas d'apprendre le langage mais l'API**. Bien qu'elle présente également des similitudes avec celle de Java, elle est quand même très différente.

[4] En réalité, on peut utiliser la commande SqlCommandBuilder pour s'affranchir des requêtes SQL, voir 7.5.

[5] VS [23] propose différentes versions du HTML ou du XHTML que le code doit respecter.

[6] Les modifications en *runtime* sont limitées, il n'est, par exemple, pas possible d'ajouter des nouveaux fichiers.

[7] Pour une comparaison plus complète du C# du point de vue d'un programmeur Java, voir [72].

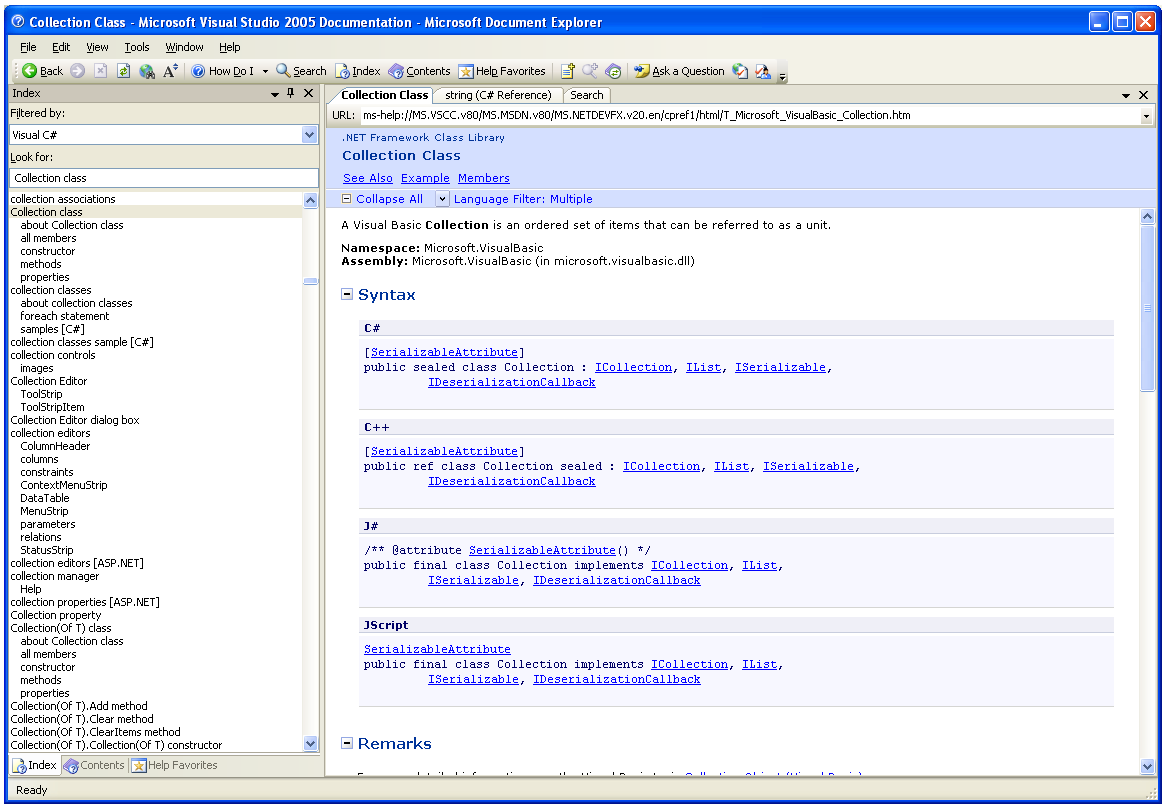

Figure 7.1: Documentation .NET de la classe Collection

```
public class Person{
    string name;
    public string Name{
        get{ return this.name;}
        set{ this.name = value;}
    }
}
```

Listing 7.1: Exemple d'un *bean* en C#

De même, pour la documentation, voir Figure 7.1, ça demande un temps d'adaptation de passer de la Javadoc à la documentation .NET. Une difficulté supplémentaire de la documentation pour .NET, est la multitude de langages. En effet, il n'est pas toujours évident de trouver l'exemple ou la documentation dans le bon langage. De manière générale, les langages les mieux documentés sont le C# et VB.NET. Et pour finir, il n'est pas rare que les exemples comportent des erreurs (heureusement, en règle générale, facile à détecter) ou alors que la page recherchée, n'existe pas.

7.3 Tiers présentation, ASP.NET

Le développement d'applications *web* en ASP.NET à l'aide de VS se fait graphiquement, à l'aide de glisser-déposer. Il n'y que relativement peu de fichiers différents. En règle générale, il n'y a qu'un fichier de configuration (`Web.config`) et un fichier `.aspx` avec à chaque fois un fichier de code (par exemple, C# ou VB.NET) associé. Pour une com-

```
Person person = new Person();
person.Name = "Alain";
string localName = person.Name;
```

Listing 7.2: Exemple d'appel d'un *bean* en C#

```
<%@ Master Language="C#" AutoEventWireup="true" CodeFile="MasterPage.master.cs"
    Inherits="MasterPage" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <link href="App_Themes/ThemeJ2EE/cybercoach.css" rel="stylesheet" type="text/css"/>
  <title>Cybercoach</title>
</head>
<body>
<form id="form1" runat="server">
  <div id="container">
    <div id="header"><img src="App_Themes/ThemeJ2EE/images/header.jpg" height="100"
    width="750" alt="Cybercoach" /></div>
    <div id="menu"><asp:ContentPlaceHolder ID="MenuPlaceHolder" runat="server" /></div>
    <div id="content"><asp:ContentPlaceHolder ID="ContentPlaceHolder" runat="server" /></div>
    <div id="footer"><p>Design Dominic Br&uuml;gger 2005 – Coding Alain Giller 2006</p></div>
  </div>
  <asp:SiteMapDataSource runat="server" ID="mainSitemap" />
</form>
</body>
</html>
```

Listing 7.3: Fichier `MasterPage.master` de Cybercoach

paraison plus complète de l'ASP.NET et Struts, voir [71] (attention à la conclusion, la comparaison vient de Microsoft qui a forcément un parti-pris). Mise à part que l'ASP.NET se base sur la technologie des *webforms*, le code généré est assez ressemblant aux pages JSP.

7.3.1 *Layout*

La structure générale d'un site Internet se fait grâce au fichier `MasterPage.master`. Il suffit de créer la page comme on le souhaite, et mettre des *webforms* ContentPlaceHolder là où le contenu doit changer dynamiquement, par exemple, le contenu et/ou les menus, voir Listing 7.3[8] et la capture d'écran correspondante Figure 6.3. Dans l'exemple, on peut voir qu'il y a deux contenus dynamique aux lignes 15 et 16. Le *webform* à la ligne 19 est pour la gestion du menu avec fichier XML, voir Listing 7.4 il n'apparaît pas dans la page HTML.

7.3.2 Menu

Pour la gestion des menus, .NET propose à nouveau une élégante solution grâce à l'utilisation d'un fichier XML dédié (`Web.sitemap`), voir Listing 7.4. L'avantage de cette structure, outre que l'architecture du site est centralisée, est la gestion de la hiérarchie (la gestion de l'arborescence du site) et des droits d'accès grâce notamment aux rôles[9] (dans

[8] Le lecteur intéressé peut comparer ce code source avec son équivalent Struts Listing 2.2.
[9] Les rôles sont définis dans le fichier `Web.config`, voir la sous-section 7.3.4.

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" enableLocalization="true
    ">
  <siteMapNode roles="*">
    <siteMapNode title="$resources:Menu,account" roles="user">
      <siteMapNode url="~/MembersPages/editUser.aspx" title="$resources:Menu,editUser"
          description="" />
      <siteMapNode url="~/MembersPages/editProfile.aspx" title="$resources:Menu,editProfile"
          description="" />
      <siteMapNode url="~/MembersPages/deleteUser.aspx" title="$resources:Menu,deleteUser"
          description="" />
    </siteMapNode>
  </siteMapNode>
</siteMap>
```

Listing 7.4: Extrait du fichier `Web.sitemap` de Cybercoach

l'exemple, il s'agit du *tag* roles="user" à la ligne 4 qui permet l'accès du menu uniquement au rôle user ou le *tag* roles="*" à la ligne 3 qui permet l'accès du menu à tout le monde). Pour chaque page, il spécifie qui en a le droit d'accès (rôles), le nom du fichier et le nom à afficher ainsi qu'éventuellement une description. Ensuite, VS [23] se charge tout seul de générer les Javascripts adéquats (d'après les paramètres spécifiés par le *webform* se chargeant de l'affichage du menu).

7.3.3 Validation des formulaires

La validation des formulaires se fait grâce à des *webforms* spécifiques. Il y a une vérification côté client, grâce à Javascript, et une autre côté serveur. Concrètement, le programmeur doit mettre les *webforms* là où il veut que les erreurs apparaissent. Typiquement :

- Un message d'erreur près du champ erroné sitôt après la saisie (côté client, grâce à Javascript), voir Listing 7.5, ligne 5, le *webform* asp :RequiredFieldValidator ;
- Au début du formulaire, un champ qui récapitule toutes les erreurs présentes sur la page (côté server), voir Listing 7.5, ligne 1, le *webform* asp :ValidationSummary.

Dans la capture d'écran Figure 7.2 on peut voir en haut de la page, en rouge, un résumé des erreurs présentes. Ce résumé apparaît sitôt que l'utilisateur envoie le formulaire (la vérification est faite sur le serveur). On peut également voir une petite astérisque rouge près du champ "e-mail". Elle apparaît sitôt après la saisie (la vérification est faite grâce à Javascript côté client).

Concrètement, le *webform* asp :ValidationSummary à la ligne 1 va parcourir tous les *webforms* de validation (dans l'exemple il n'y en a qu'un seul, asp :RequiredFieldValidator à la ligne 5), regarder s'ils sont valide et, le cas échéant, récupérer la valeur qui se trouve dans le *tag* ErrorMessage. Il existe plusieurs validations possibles : on peut exiger que le champ aie été modifié, qu'il satisfasse un masque (expression régulière) ou un type (entier, date, etc.), qu'il soit identique à un autre champ (pour les mots de passe, par exemple), etc. Le *webform* se charge tout seul de générer le code Javascript, le programmeur n'a pas à s'en soucier.

```
<asp:ValidationSummary ID="ValidationSummary1" runat="server" CssClass="errorbox"
    HeaderText="<%$Resources: errors_header%>" />
<asp:Label ID="Label1" runat="server" Text="<%$ Resources:firstName %>" AssociatedControlID="
    firstName" />
<asp:TextBox ID="firstName" runat="server"/>
<asp:RequiredFieldValidator ID="firstNameChk" runat="server"
    ErrorMessage="<%$ Resources:firstName %>"
    ControlToValidate="firstName" Text="*"/>
```

Listing 7.5: Extrait d'un formulaire avec la vérification côté serveur et client

```
using System.EnterpriseServices;
namespace MonApplication{
  [ObjectPooling(MinPoolSize = 3,MaxPoolSize = 10,CreationTimeout = 20)]
  public class MonComposant : ServicedComponent {
    [AutoComplete]
    public void ReserveHotel(int numeroChambre){
      // Met à jour la base de données Hotel
    }
    //méthodes pour le pooling
    public override void  Activate()
      {   // Initialise les ressources ;}
    public override void  Deactivate()
      {   //  Libère des ressources  }
    public override bool CanBePooled()
      {    return true ; }
  }
}
```

Listing 7.6: Exemple de Serviced Component

7.3.4 Configuration

Toute la configuration tient dans un seul fichier[10] (`Web.config`, voir Listing 6.1), il contient, entre autre : les rôles, le type d'authentification, la connexions à la base de données, l'état des sessions (*cookies*), etc.

7.4 Tiers métier

Le tiers métier se fait grâce a Serviced Component (COM+) (System.EnterpriseServices). L'exemple Listing 7.6, à comparer avec celui des *session-beans* Listing 5.2, montre qu'il hérite de ServicedComponent. De plus, on constate que le code contient :

- la méthode désirée (ReserveHotel(int numeroChambre) ligne 6) ;
- une annotation [AutoComplete] ligne 5 pour les transactions ;
- une annotation (facultative) ObjectPooling ligne 3 pour les spécification du *pooling* ;
- des méthodes (facultatives) pour le *pooling* à partir de la ligne 10.

Pour plus d'information, voir [74].

7.5 Tiers données, ADO.NET

Pour la persistance des données il existe plusieurs possibilités, dont les principales sont les accès directs à des fichiers ou ADO.NET. Ce dernier permet, l'accès sous format XML, de manière unifiée aux données (base de données, fichiers XML, etc.). Deux objets permettent de manipuler des données extraites de la base de données : DataSet (mode déconnecté) et DataReader (mode connecté). Le DataReader, parcourt les enregistrements en lecture seule, séquentiellement, et par défilement vers l'avant.

Le DataSet (voir Figure 7.4), quant à lui, est un objet qui réside en mémoire et qui correspond à une copie locale des données d'une base. Une fois en mémoire, les informations du DataSet (provenant des requêtes SQL "SELECT") peuvent être manipulées sans avoir recours à une connexion au serveur. Il contient les tables d'une base mais aussi les relations entre ces différentes tables et les contraintes appliquées aux données (comme le type ou l'attibut UNIQUE, par exemple). Dans .NET, XML est utilisé comme le standard de description et de persistance des données et à ce titre l'objet DataSet est représenté sous cette forme. Les données d'un DataSet sont écrites en XML et le schéma est écrit en XSD[11].

Comme l'objet DataSet est déconnecté de toute source de données, il peut être utilisé pour la transmission de données entre différents tiers d'une application *Web*, mais aussi à d'autres systèmes, grâce à sa représentation XML.

Pour accèder aux données, ADO.NET a besoin d'un *provider*. Microsoft, avec VS 2005 [23], en propose quatre, à savoir : ODBC, OLE DB, SQL Server [10] et Oracle [9] (voir Figure 7.3). D'autres entreprises ont développé leur propre provider pour .NET (c'est le cas de MySQL [7], par exemple). Grâce à ces *providers*, on peut copier toute ou partie de la base de données dans un DataSet, puis en afficher le contenu, ou le modifier et mettre à jour la base de donnée [45]. VS propose des outils pour la conception des fichiers XSD mais ils fonctionnent, apparament, uniquement avec les *providers* fournis par Microsoft. Pour les autres (par exemple, MySQL [7]), cela reste néanmoins possible en utilisant ODBC (si la base de données propose une interface ODBC) à la place du connecteur natif. Mais dans ce cas, les performances et les fonctionnalitées sont bien évidemment bridées.

Avec les *entity-beans* dans J2EE les requêtes sont déclaratives (en EJBQL), et le conteneur se charge tout seul de générer les requêtes SQL des accès (en règles général, il n'y a que les requêtes SQL "SELECT" à écrire). Il s'agit d'une vision objet de la base de données. Avec ADO.NET, Avant d'envoyer la mise à jour à la base de données, on doit configurer les objets InsertCommand, UpdateCommand et DeleteCommand[12] pour harmoniser les modifications avec la base de données. Pour des scénarios limités, on peut utiliser l'objet SqlCommandBuilder pour les générer automatiquement. Il est également possible de définir dans les DataSet des procédures stockées. ADO.NET offre une vision ensembliste au programmeur. Ce dernier se doit de bien connaître le SQL contrairement au monde J2EE. Exemple : Listing 7.7. Les 5 premières lignes se chargent de récupérer les données de la base de données et de les sauvegarders dans un DataSet. Ensuite, à la ligne 8 on

[10] En fait, dans beaucoup d'applications, on ajoute encore un fichier (en C#) spécifique pour les notations (voir la sous section 6.1.4).

[11] Il n'est pas obligatoire de faire un schéma pour accèder aux données, mais cela permet d'avoir un DataSet fortement typé.

[12] C'est-à-dire que le programmeur doit écrire les requêtes SQL correspondantes.

```
string strRequete = "SELECT * FROM Categories ORDER BY CategoryID";
// Chargement de la liste des catégories dans oDataSet
SqlDataAdapter oSqlDA = new SqlDataAdapter(strRequete, oConnection);
DataSet oDataSet = new DataSet("Categories");
oSqlDA.Fill(oDataSet, "Categories");

// Definition de l'objet InserCommand (Il existe DeleteCommand, UpdateCommand, SelectCommand)
oSqlDA.InsertCommand = New SqlCommand("INSERT INTO Categories(CategoryName, Description)
    Values(@CategoryName,@Description)", oConnection);
oSqlDA.InsertCommand.Parameters.Add("@CategoryName", SqlDbType.NVarChar, 15, "CategoryName");
oSqlDA.InsertCommand.Parameters.Add("@Description", SqlDbType.NText, 16, "Description");

// Creation d'un objet "Ligne" avec des données
DataRow oDataRow;
oDataRow = oDataSet.Tables("Categories").NewRow();
oDataRow("CategoryName") = "Wine";
oDataRow("Description") = "French Wine";

// Ajout de l'objet "Ligne" au DataSet
oDataSet.Tables("Categories").Rows.Add(oDataRow);

// Mise à jour de la source de données à partir du DataSet
oSqlDA.Update(oDataSet, "Categories");
```

Listing 7.7: Exemple de mise à jour d'une base de données avec des procédures stockées

définit la commande InsertCommand[13] à l'aide la commande SQL INSERT. Les 2 lignes suivantes sont là pour spécifier le type des champs de la base de données. De la ligne 13 à 16 on consruit une ligne (*row*) avec les valeurs "Wine" et "French Wine"[14]. Cette ligne (*row*) se trouve ajoutée au Dataset. Sitôt que le DataSet est modifié, on peut mettre à jour la base de données en invoquant la méthode Update définie précédement (ligne 8) [44], [43].

7.6 Utilisation des *design patterns*

Tout comme avec J2EE, il est conseillé d'utiliser les *design pattern* pour faire une architecture qui respecte les principales qualités d'un logiciel. Le Gang of Four (GoF) a d'ailleurs publié un livre électronique sur le sujet, voir [EG05]. L'ASP.NET est basé, comme Struts [1], sur le *pattern* MVC. L'utilisation de ce *pattern* se fait de manière naturelle et intuitive, VS [23] nous dirige sans que l'on s'en rende compte. De même, le pattern *Transfert object* qui nécessite en Java beaucoup de code, s'utilise naturellement en ASP.NET et ADO.NET avec les DataSet.

7.7 Conversion d'un ancien (*legacy*) système

Le développement d'une application obéit souvent a des contraintes financières. La réécriture de gros projets et la formation des développeurs sur de nouveaux langages peuvent devenir lourds en termes de coûts pour les entreprises. Les applications .NET peuvent utiliser du code "legacy" qui est donc non managé (et donc non sûr !), en utilisant les bibliothèques System.InterOpServices pour créer des liaisons vers C#. Mono supporte également cette fonctionnalité (qui fait partie des spécifications du CLI). C'est d'ailleurs la

[13] Le SQL se trouvant à la ligne 8 peut aussi se mettre directement dans un fichier XSD comme procédure stockée.

[14] Les valeurs doivent respecter les types spécifier à la ligne 8 et aux 2 lignes suivantes.

méthode utilisée pour ajouter des fonctionnalités qui ne sont pas dans les spécifications, tel que Gtk# (concurent des *winforms*).

La conversion avec le langage Java se fait comme suit : en plus du J# qui ne nécessiterait que relativement peu de modification du code source Java, il est même possible d'utiliser directement le code compilé d'une application Java sur la plateforme .NET grâce au projet IKVM [54] (Le projet IKVM est une implémentation de la JVM en .NET).

Un dernier exemple de conversion est l'utilisation du C++/CLI (managé). La plateforme .NET 2.0 introduit le langage C++/CLI vers lequel tous les anciens programmes codés en C++ peuvent être compilés, bien que Microsoft aie ajouté des fonctionnalités internes à .NET, par exemple, pour la gestion mémoire. Il n'est en revanche pas possible de compiler un code C++/CLI en utilisant un ancien compilateur. A noter qu'il existait une version managée du C++ (*Managed C++*) dans la première version de l'environnement .NET. Les assemblages produits sont dits mixtes car ils contiennent à la fois des sections de code en IL et en code machine (x86). Les développeurs C++ pourront ainsi plus facilement migrer le vieux code et accéder aux nouvelles fonctionnalités apportées par .NET [58].

La multitude des langages de programmation de .NET est, sans aucun doute, une très grande force de cette plateforme[15].

[15] Actuellement, on dénombre plus d'une quarantaine de languages de programmation différents compatible avec la plateforme .NET [56].

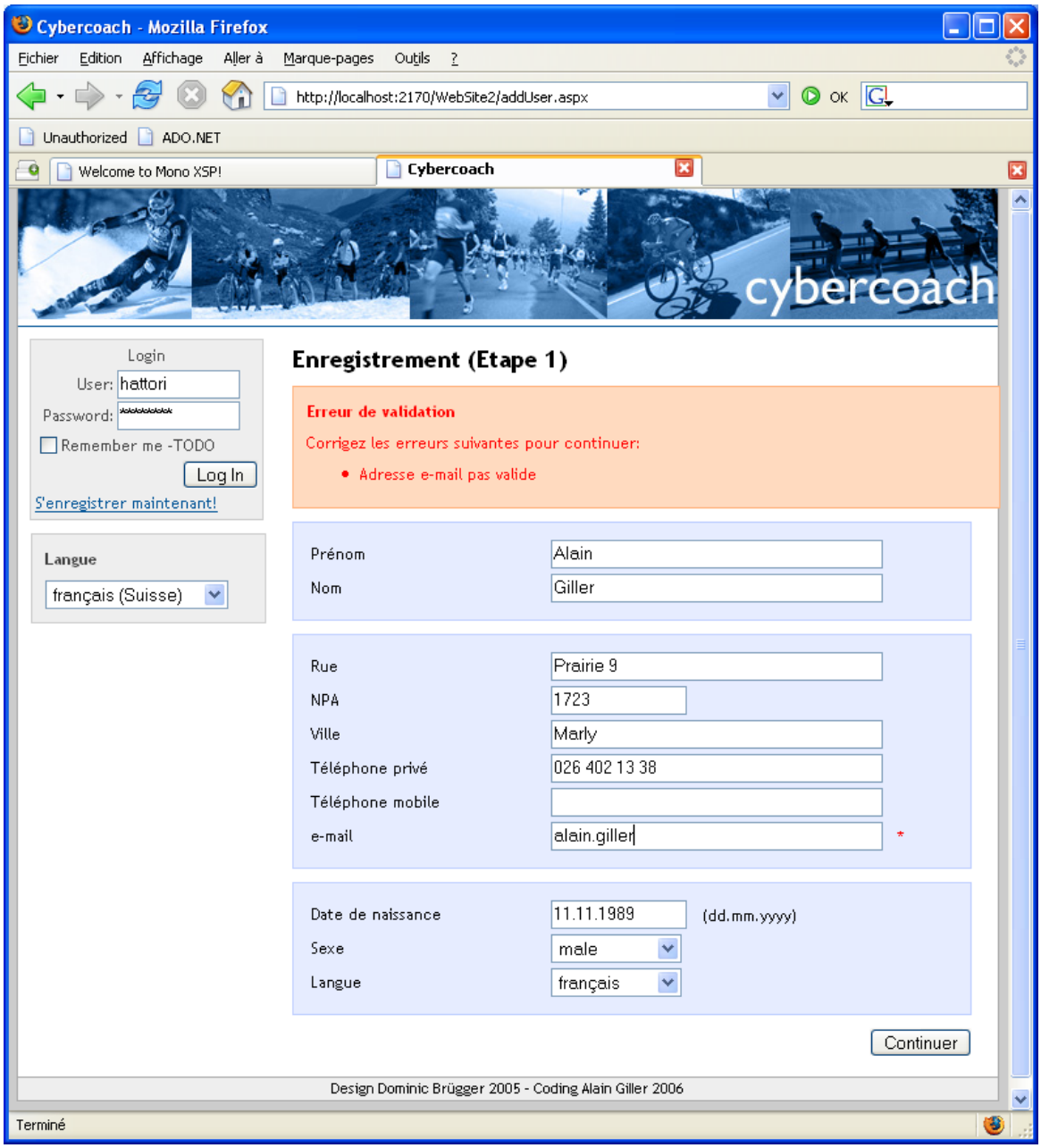

Figure 7.2: Validation d'un formulaire

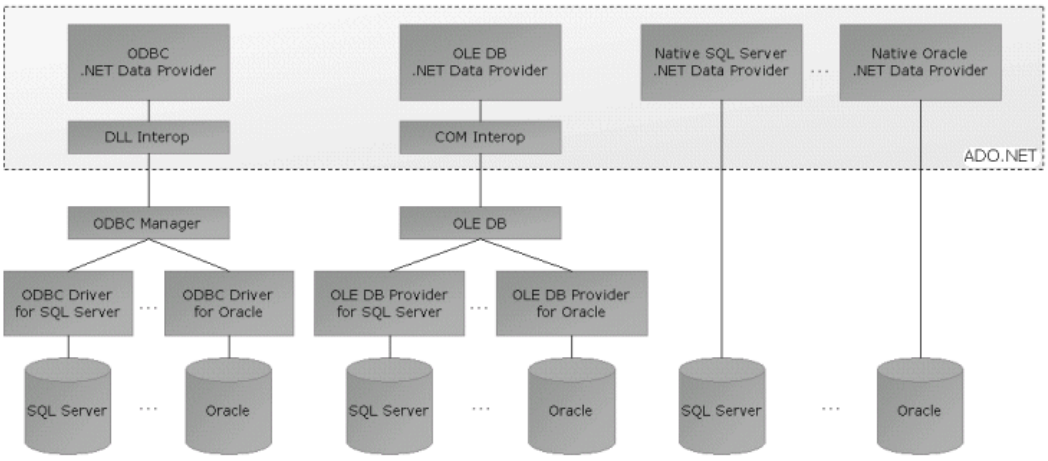

Figure 7.3: Schéma d'accès aux bases de données avec ADO.NET (tirée de developpeur.journaldunet.com)

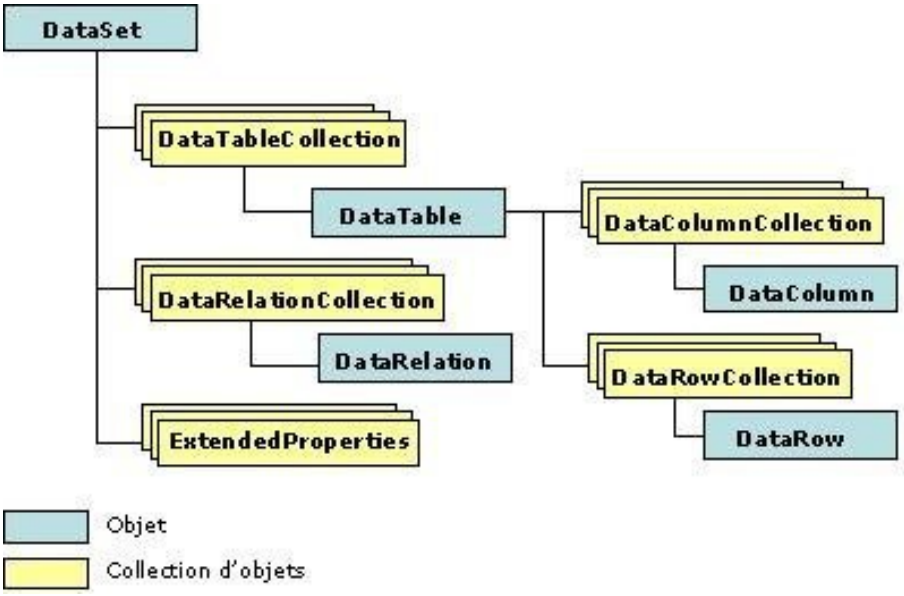

Figure 7.4: Structure d'un DataSet (tirée de dotnet.developpez.com)

8 Conclusion

Tout d'abord, les objectifs ont été atteints : j'ai appris le *framework* .NET [52] et j'ai pu faire une comparaison des deux technologies. Par contre, l'implémentation du Cybercoach en .NET [3] n'est pas aussi aboutie que son cousin J2EE [2] car ce travail a demandé beaucoup plus de travail de recherche que prévu :

- fonctionnement détaillé de .NET et J2EE ;
- compréhension de l'API de .NET et du C# ;
- installation et configuration de Mono [57] (bien que pas utlisé finalement[1]) ;
- compréhension de l'implémentation du Cybercoach J2EE [2] ;
- installation et brève utilisation d'autres IDE tel que Sun Java Studio Creator [12], Netbeans 5 [18], Borland C#Builder [19], etc.
- utlisation de MySQL [7] et non pas de SQL Server [10] pour .NET ;
- maîtrise et familiarisation des outils ;
- recherche d'une bonne architecture pour le Cybercoach .NET [3] ;
- etc.

La comparaison à l'aide du projet Cybercoach est délicate. Tout d'abord, la version J2EE [2] est plus vieille et ne bénéficie donc pas des dernières technologies tels les JSF. Ensuite, elle était déjà implémentée alors que la version .NET [3] à dû être développée depuis zéro.

Pour conclure récapitulons brièvement les différences et similitudes en se concentrant sur les trois aspects suivant :

- Architecture trois tiers ;
- Performance ;
- Politique.

[1] Au début du projet, Mono a été sérieusement étudié car le développement aurait dû se faire avec ce dernier. Hélas, la version présente alors ne permettait pas d'utiliser ASP.NET ni ADO.NET dans la version 2.0 du *framework*.

8.1 Architecture trois tiers

8.1.1 Tiers présentation

On peut affirmer que l'ASP.NET est bien plus accessible et rapide à comprendre que Struts. Il est clairement en avance. En plus, d'une manière générale, Microsoft développe toujours pour ses produits des outils très "accessibles", très graphiques. Avec l'arrivée des JSF, Java a comblé son retard technologique et, avec les derniers outils de développement tels que Netbeans 5 [18] ou Sun Java Studio Creator [12], etc., a amélioré l'accessibilité de sa plateforme. Il est clair que la venue d'un concurent sérieux au J2EE a dynamisé son développement.

Pour le déverminage, VS [23] est plus complet que Eclipse 3.0 [13] pour toute l'interface *web* avec Struts [1] et les JSP. Mais, même remarque que précédement, les nouveaux outils, ou *plugin* ([17], par exemple) comblent ce retard.

8.1.2 Tiers métier

Pour le tiers métier, les deux plateformes offrent les mêmes fonctionnalités (*pooling*, transactions, objets distribués, etc.) et s'implémentent de façon assez similaire. Avec .NET le tiers métier se base sur la technologie COM+ qui fait partie du système d'exploitation Windows Server. Ce qui implique quelques opérations à faire en dehors du .NET, comme par exemple, enregistrer l'assemblage dans le serveur d'application, en l'occurrence COM+ Component Server de Windows Server. Les *session-beans*, quand à eux, sont managés par le conteneur du serveur d'application J2EE.

8.1.3 Tiers données

Pour l'accès à la base de données, ADO.NET avec ses DataSet est un bon challenger des *entity-beans*. Les DataSet offrent une architecture très intéressante non seulement pour l'accès proprement dit, mais également pour le transfert de couche en couche, chose que les *entity-beans* ne proposent pas du tout. Toutefois, il ne supporte actuellement que peu de bases de données de façon native. Par défaut, VS [23] n'est pas capable de créer la description pour une base MySQL [7] sans passer par ODBC (ce qui limite les performances et les fonctionnalitées).

ADO.NET offre une vision ensembliste de la base de données ce qui oblige le programmeur à bien maitriser le SQL. Au contraire les *entity-beans* offrent une vision orientée objet de la base de données, ce qui permet de s'affranchir de l'écriture des requétes SQL.

Dans ADO.NET, la description de la base de données se fait dans un seul fichier XSD. Pour les *entity-beans*, cela se fait à l'aide de *tag* XDoclet pour chaque fichier correspondant aux tables de la base de données. Avec l'arrivée de EJB 3.0, le modèle de programmation est heureusement simplifié et allégé, ce qui devrait largement simplifier l'accès aux bases de données.

8.2 Performances

Comparer le .NET de Microsoft à J2EE du point de vue performance pure n'est pas très fair-play. En effet, le .NET de Microsoft, bien que basé sur un langage intermédiaire (IL), est dédié uniquement à la plateforme Windows alors que Java fonctionne sur une multitude de machines. De plus, il existe plusieurs implémentations différentes de J2EE (Sun [29], IBM [30], JBoss [28], etc), et également de .NET (DotGNU [50], Rotor [60], Mono [57]). Il existe un test de performances disponible sur Internet, voir [75]. Il faut bien évidemment relativiser les résultats (surtout qu'ils ont été faits dans un laboratoire de Redmond prêté par Microsoft...), mais on peut tout de même constater que la plateforme de Microsoft offre en règle générale des performances légèrement supérieures[2]. Il ne faut pas oublier que la comparaison date un peu, et que, par exemple, les JSF n'étaient pas encore disponibles.

8.3 Politique

Le problème majeur de .NET est l'ouverture uniquement partielle des spécifications et donc sa dépendance (en tout cas actuelle) totale envers Windows. En effet, les spécifications de l'ECMA ne couvrent pas tout le domaine .NET (par exemple ASP.NET, ADO.NET, les *winforms*, les bibliothèques pour le traitement du XML sont des technologies propriétaires de Microsoft) alors que dans le J2EE, les spécifications couvrent également l'accès aux bases de données (JDBC, EJB, JDO, etc.), la partie présentation (JSP, Servlets, JSF, etc.) ou le traitement des fichiers XML (Java API for XML Processing (JAXP), etc.). Par contre, l'accès aux spécifications est plus facile pour la plateforme de Microsoft. En effet il suffit de se rendre sur le site de l'ECMA [64] pour télécharger les spécifications. Pour Java, il faut obligatoirement faire partie de la Java Community Process (JCP), ce qui nécessiste une inscription (gratuite pour les privés) et l'acceptation des conditions (notamment de non divulgation).

Les implémentations tierces de .NET tel que Mono [57] essayent de recouvrir le même domaine que Microsoft (entre autre par *reverse engineering*) par souci de compatibilité, mais s'expose à d'éventuelles violations de brevets de la part de ce dernier. De plus rien n'empêche Microsoft, dans une future version, de modifier profondément l'API ou même de ne plus tenir compte des spécifications pour son implémentation.

Microsoft, oblige l'utilisation d'IIS avec l'ASP.NET. Ce serveur *web* est connu pour avoir souffert de beaucoup de vulnérabilités. Une application tournant sur IIS peut-être considérée comme une "invitation" aux *crackers*.

8.4 Conclusion

Pour un développeur Java il n'est pas très sensé de changer de plateforme, car il connait déjà bien l'API, le langage Java et ses outils. L'exception pourrait être la conversion d'un ancien (*legacy*) système (ou bien sûr, une demande du client).

[2]Ces résultats sont bien sûr valables sur une machine identique (x86 car Windows ne supporte que cette architecture) alors que J2EE peut tout à fait tourner sur des clusters. Et il est clair qu'à ce moment J2EE peut être largement plus performant.

Pour un développeur .NET, la raison pour passer à J2EE, en dehors de la demande explicite d'un client, peut être une raison politique (dépendance total à une seule entreprise, migration future impossible), ou d'interopérabilité (fonctionnement sur des serveurs Solaris par exemple).

Enfin, pour un développeur ne connaissant aucune des deux plateformes, il pourrait être tenté par la facilité d'accès du .NET et de tous les langages qu'il propose. Pour de petites et moyennes applications, il s'agit d'une très bonne alternative (mais attention, contrairement au J2EE, il n'existe pas de licences gratuites). Par contre, pour des grosses applications, destinées à durer longtemps, les contraintes de .NET sont importantes et peuvent limiter l'utilisation de l'application (par exemple, si quelques années après le développement on désire avoir autre chose que des serveurs Windows).

- **Prise en main**
  Egalité. Avantage historique pour .NET mais actuellement les deux plateformes se valent. C'est surtout une question de préférences, besoins et connaissances propres de chaque personne.
- **Accès aux données**
  Egalité. .NET offre une vision ensembliste alors que J2EE offre une vision orientée objet. D'un point de vue théorique pur, l'approche de J2EE est meilleur. Cependant, dans la pratique, ce modèle peut poser des problèmes, de performance notamment.
- **Performance**
  Egalité. Sur une plateforme identique (Windows) .NET est légèrement plus performante. Par contre J2EE permet l'utilisation d'autres plateformes que x86 et Windows, par exemple des clusters. Si la plateforme n'est pas imposée J2EE, grâce à sa portabilité, peut offrire des performances supérieures voir largement supérieures.
- **Conversion d'un ancien (*legacy*) système**
  Avantage .NET. Pour convertir un ancien système vers J2EE ou .NET, l'avantage est clairement pour la plateforme de Microsoft. Bien que la conversion ne se fera pas sans mal, la majorité du code pourra être réutilisée telle quelle, principalement grâce aux différents langages supportés.
- **Liberté**
  Avantage J2EE. Avec .NET, le fait de développer une application liée irrémédiablement à un seul système (fermé et propriétaire) est une solution très contraignante qui peut hypothèquer l'avenir d'un projet.

Finalement, peut-être que la réponse se trouve avec http://googlefight.com/ et des mots clés comme ASP.NET et Struts, ou Microsoft .NET et J2EE ou Java, mais la pertinence du résultat est très discutable, d'autant plus que Java ou .NET ont d'autres significations ailleurs qu'en informatique...

A
Acronymes

ACID Atomicity, Consistency, Isolation, and Durability

ADO ActiveX Data Objects

ADSI Active Directory Service Interfaces

AOP Aspect Oriented Programming

API Application Programming Interface

ASCII American Standard Code for Information Interchange

ASP Activ Server Pages

AWT Abstract Windowing Toolkit

BCL Base Class Library

BMP Bean Managed Persistence

CAS Code Access Security

CIL Common Intermediate Language

CLI Common Language Infrastructure

CLR Common Language Runtime

CLS Common Language Specification

CMP Container Managed Persistence

COM Component Object Model

CORBA Common Object Request Broker Architecture

CSS Cascading Style Sheets

CTS Common Type System

CVS Concurrent Versions System

DB Data Base

DLL Dynamically Linked Library

DNS Domain Name System

EAR Enterprise ARchive

ebXML Electronic Business using eXtensible Markup Language

ECMA European Computer Manufacturers Association

EJB Enterprise Java Beans

EJBQL Enterprise Java Beans Query Language

ERP Enterprise Resource Planning

FTP File Transfer Protocol

GAC Global Assembly Cache

GNU GNU is Not Unix

GoF Gang of Four

GUI Graphical user interface

HTML HyperText Markup Language

HTTP HyperText Transfert Protocol

IBM International Business Machines

IDE Integrated Development Environment

IDL Interface Description Language

IEEE Institute of Electrical and Electronics Engineers

IETF Internet Engineering Task Force

IIS Internet Information Services

IL Intermediate Language

ISO International Organization for Standardization

J2EE Java 2 Platform, Enterprise Edition

J2SE Java 2, Standard Edition

JAR Java ARchive

JAX-RPC Java API for XML-based RPC

JAXB Java Architecture for XML Binding

JAXR Java API for XML Registries

JAXM Java API for XML Messaging

JAXP Java API for XML Processing

JCA Java connector architecture

JCP Java Community Process

JDBC Java DataBase Connectivity

JDK Java Development Kit

JDO Java Data Object

JIT Just In Time

JMS Java messaging service

JMX Java Management Extensions

JNDI Java Naming and Directory Interface

JRE Java Runtime Environment

JSF Java Server Faces

JSP Java Server Pages

JSR Java Specification Requests

JTA Java Transaction API

JVM Java Virtual Machine

LDAP Lightweight Directory Access Protocol

MS Microsoft

MSDN Microsoft Developer Network

MSDTC Microsoft Distributed Transaction Coordinator

MSIL Microsoft Intermediate Language

MSMQ MSMQ Microsoft Message Queuing

MVC Model View Controler

OASIS Organization for the Advancement of Structured Information Standards

ODBC Open DataBase Connectivity

OLE Object Linking and Embedding

OOP Object Oriented Programming

OS Operating System

PAL Platform Adaption Layer

PDF Portable Document Format

POJO Plain Old Java Objects

RMI Remote Method Invocation

RPC Remote procedure call

SOAP Simple Object Access Protocol

SQL Structured (English) Query Language

SSCLI Shared Source Common Language Infrastructure

RTF Rich Text Format

SUN Stanford University Network

SVN SubVersioN

TO Transfert Object

UML Unified Modeling Language

VB Visual Basic

VES Virtual Execution System

VS Visual Studio

W3C World Wide Web Consortium

WAR Web ARchive

WSI Web Services Inspection

WYSIWYG What You See Is What You Get

XML eXtensible Markup Language

XSD XML Schema Definition Language

XSLT eXtensible Stylesheet Language Transformations

B
Logiciels utilisés

B.1 Systèmes d'exploitation

- **GNU/Linux**
  - ▷ **Gentoo** - Il s'agit de la distribution princalement utilisée durant tout le projet.
  - ▷ **Ubuntu** - L'utilisation a été plus marginale
- **Windows**
  - ▷ **Windows XP Professional** - Il s'agit du système d'exploitation principalement utilisé pour le développement de l'application .NET. Il tournait dans une machine virtuelle Vmware sous GNU/Linux.
  - ▷ **Windows Server 2003 Web Edition** - La version finale du Cybercoach .NET tourne sur Windows Server 2003 Web Edition (l'image Vmware est diponible sur le DVD, voir chapitre F).

B.2 Outils de dévelopement, IDE

- **Microsoft Visual Studio 2005**
  Il s'agit de l'IDE principalement utilisé pour le développement du Cybercoach .NET [23].
- **Monodevelop 0.9**.
  Principalement utilisé en début de projet [21].
- **Eclipse 3.0**
  Il s'agit de l'IDE utilisé pour l'étude de Cybercoach J2EE [13].
- **Sun Java Studio Creator, Netbeans 5, etc.**
  Ces outils ont été étudiés afin de découvrir leurs fonctionnitées. Ils n'ont pas été utilisés pour du développement dans le cadre du Cybercoach [12], [18].

B.3 Serveur *web* et d'applications

- **Visual Studio 2005**
  Visual Studio comprend un mini serveur *web* pour le déverminage. Il s'agit concrètement du serveur *web* le plus utilisé tout au long du projet.
- **IIS, Windows Server 2003**
  Le serveur *web* sur lequel tourne la version final de Cybercoach .NET.

- **JBoss 4.0.3**
  Il s'agit du serveur d'application pour la version J2EE du Cybercoach [28].
- **XSP, Apache**
  Il s'agit des serveurs *web* pour Mono 1.1.13 [57], [1].

B.4 Base de données

Pour la base de données, il s'agit de MySQL 4.1 [7] pour les deux versions du Cybercoach (J2EE et .NET) et pour les deux systèmes d'exploitation utilisés (Windows XP et GNU/-Linux). La gestion des données s'est fait en partie à la main, directement à l'aide de la console, voir section C.1. Mais la plus grosse partie s'est faite à l'aide d'outils graphique :

- **SQLyog**
  GUI pour Windows pour la gestion des tables et données de MySQL. Il existe une version gratuite. Elle permet d'afficher les tables et les données, de modifier les tables, d'exécuter des requêtes SQL, de sauvegarder des données, etc. [8].
- **MySQL Administrator**
  GUI pour Windows (il existe également des versions pour Linux et Mac OS X) *open-sources* pour l'administration de MySQL [7].
- **gmyclient**
  GUI pour GNU/Linux *open-sources* pour la gestion des tables et données de MySQL.

B.5 Outils de gestion de versions

Pour la gestion de versions, Subversions, le successeur de Concurrent Versions System (CVS), a été choisi. Et comme GUI :

- **TortoiseSVN**
  Interface graphique pour Windows très puissante s'intègrant directement à l'explorateur [68] ;
- **SmartSVN**
  Application multi-plateformes programmée entièrement en Java, avec une interface très agréable et efficace [62] ;
- **Subclipse**
  Plugin pour l'utilisation de Subversion avec Eclipse [13].

B.6 Documentation

La documentation est écrite en format LaTeX, avec Kile (GNU/Linux) comme éditeur. Le *template* LaTeX, pour la documentation, est disponible sur le site Internet du groupe Software engineering [63] (*Documentation Guidelines*).

C Installation

C.1 MySQL

Les détails pour l'installation de MySQL 4.1 sous Mac OS X sont disponibles dans Annexe G (page 86) de [Brü05]. Pour GNU/Linux, la procédure est identique et pour Windows elle se fait très facilement avec un installateur graphique (remarque : Les tables dans Cybercoach sont de type InnoDB). Les fichiers d'installations se trouvent sur le site de MySQL [7]. Ensuite, il faut créer la base de données Cybercoachdb. La procédure est la même pour toutes les plateforme (voir Annexe G de [Brü05]) :

1. Se logger comme root :
   ```
   mysql -u root -p
   ```
2. Créer la base de données cybercoachdb
   ```
   mysql> CREATE DATABASE cybercoachdb;
   ```
3. Spécifier les droits et le mot de passe pour la base de données cybercoachdb
   ```
   mysql> GRANT ALL PRIVILEGES ON cybercoachdb.* TO 'cybercoach'@'localhost' IDENTIFIED BY
       'cybercoach1234';
   mysql> GRANT ALL PRIVILEGES ON cybercoachdb.* TO 'cybercoach'@'%' IDENTIFIED BY '
       cybercoach1234';
   ```

Et pour finir, la création des tables et le remplissage par des données de test (pour toutes les plateformes) :

```
mysql -u cybercoach -h localhost -p cybercoachdb < sample_data.sql
```

Il faut encore installer le connecteur "Connector/Net". Il est écrit en C# et compatible .NET 2.0 ainsi que Mono. Ensuite, il faut l'ajouter `MySql.Data.dll` soit au GAC soit directement dans le projet dans le répertoire `Bin`.

Il est encore possible d'installer MySQL Administrator (disponible depuis le site de MySQL [7]) et SQLyog [8].

C.2 Visual Studio 2005

L'installation de Visual Studio 2005 et MSDN ne devrait pas poser de problème. Personnellement, je n'ai jamais réussi à installer SQL Server Express, mais cela ne pose pas de problème pour Cybercoach. En plus de l'environnement de développement, VS contient également le *framework* .NET.

C.3 *Framework* .NET

Pour installer le *framework* .NET, il y plusieurs solutions. La solution la plus simple est de passer par Windows Update. Sinon, Windows Server 2003 contient déjà la version 1.1 et une mise à jour suffit. Le *framework* en version 2.0 est également inclu dans VS 2005.

C.4 JBoss

JBoss [28] est l'environnement utilisé pour le Cybercoach J2EE. Son installation se fait graphiquement (sous Linux, il est préférable de l'exécuter avec les droits "root") :

```
java -jar jboss-4.0.3-installer.jar
```

Pour plus de détail voir Annexe G de [Brü05].

C.5 Déploiement du Cybercoach .NET sur IIS

Il faut démarrer IIS, et de lui spécifier le répertoire de l'application. On doit ensuite spécifier l'utilisatation de la version 2.0 de ASP.NET. Et pour finir, il faut encore enregistrer dans COM+ les composants qui utilisent System.EnterpriseServices.ServicedComponent. Cette dernière manipulation étant mal documentée et relativement pénible à mettre en place, la version fonctionnant sur IIS, contrairement au code sources disponible sur le DVD ou sur Internet, n'utilise pas les transactions.

Visual Studio offre également la possibilité de déployer une application sur IIS en local ou via File Transfer Protocol (FTP).

D

Licence de la documentation

Copyright (c) 2006 Alain Giller.

Permission est accordée de copier, distribuer et/ou modifier ce document selon les termes de la Licence de Documentation Libre GNU is Not Unix (GNU) (GNU Free Documentation License), version 1.2 ou toute version ultérieure publiée par la Free Software Foundation ; sans Sections Invariables ; sans Textes de Première de Couverture, et sans Textes de Quatrième de Couverture.

La Licence de Documentation Libre GNU (GNU Free Documentation License) est disponible sur [65] et une traduction française “non-officielle” est mise à disposition sur [66].

E

Page Internet du projet

La page *web* officielle de ce projet est :
http://diuf.unifr.ch/softeng/student-projects/completed/giller/. Sur cette page vous trouverez :

- le résumé du projet ;
- cette documentation au format Portable Document Format (PDF) ;
- le code source de l'implémentation Cybercoach .NET ;
- un lien vers l'application Cybercoach .NET.

Une capture d'écran de la page d'accueil du Cybercoach .NET Figure E.1.

Figure E.1: Page d'accueil de Cybercoach .NET

F

DVD

Sur le DVD Figure F.1 du projet, se trouvent :

- les codes source du Cybercoach .NET et J2EE ;
- le script de création des tables pour MySQL avec des données de tests ;
- cette présente documentation sous format PDF ainsi que son code source LaTeX ;
- le livre électronique [EG05] avec tous les exemples et diagrammes ;
- l'image Vmware d'un Windows Server 2003 Web Edition avec Cybercoach .NET installé et déployé sur IIS ainsi que Visual Studio 2005 installé.

Le contenu du DVD, à l'exception de [EG05] et de l'image Vmare peut également être téléchargé du site Internet officiel du projet (voir E).

Figure F.1: DVD du projet

Bibliographie

[Brü05] Dominic Brügger. Konzeption und Implementation einer multi-tier Anwendung mit J2EE. Technical report, Departement für Informatik Universität Fribourg (DIUF), 2005. [Obtenu le 1 mars 2006 de http://diuf.unifr.ch/softeng/student-projects/completed/bruegger/download/report.pdf].

[Cav04] Chuck Cavaness. *Programming Jakarta Struts*. O'REILLY Media, 2004.

[DA03] John Crupi et Dan Malks Deepak Alur. *Core J2EE Patterns : Best Practices and Design Strategies*. 2nd. Prentice Hall, 2003.

[EG05] Ralph Johnson John Vlissides Erich Gamma, Richard Helm. *.NET Design Pattern Framwork*. data and object factory [online], 2005. [Obtenu le 1 mars 2006 de http://www.dofactory.com/].

[Lad03] Ramnivas Laddad. *AspectJ in Action : Practical Aspect-Oriented Programming*. Manning, 2003.

Sites Web

[1] Site Internet de la fondation Apache. http://www.apache.org/ (dernière consultation le 23 mars 2006).

[2] Site Internet de Cybercoach version J2EE. http://diuf.unifr.ch/softeng/student-projects/completed/bruegger/index.html (dernière consultation le 23 mars 2006).

[3] Site Internet de Cybercoach version .NET. http://diuf.unifr.ch/softeng/student-projects/05-06/j2eevsdotnet.html (dernière consultation le 15 avril 2006).

[4] Site Internet de Access. http://office.microsoft.com/access/ (dernière consultation le 23 mars 2006).

[5] Site Internet de dBase. http://www.dbase.com (dernière consultation le 23 mars 2006).

[6] Site Internet de Borland InterBase. http://www.borland.com/fr/products/interbase/index.html (dernière consultation le 25 mars 2006).

[7] Site Internet de MySQL. http://www.mysql.com (dernière consultation le 23 mars 2006).

[8] Site Internet de SQLyog, outil pour MySQL. http://www.webyog.com/ (dernière consultation le 23 mars 2006).

[9] Site Internet de Oracle. http://www.oracle.com (dernière consultation le 23 mars 2006).

[10] Site Internet de SQL Server. http://www.microsoft.com/sql (dernière consultation le 23 mars 2006).

[11] Site Internet de Borland JBuilder. http://www.borland.com/jbuilder/ (dernière consultation le 25 mars 2006).

[12] Site Internet de Sun Java Studio Creator. http://developers.sun.com/prodtech/javatools/jscreator/ (dernière consultation le 25 mars 2006).

[13] Site Internet de Eclipse. http://www.eclipse.org (dernière consultation le 25 mars 2006).

[14] Site Internet de IntelliJ IDEA. http://www.jetbrains.com/idea/ (dernière consultation le 25 mars 2006).

[15] Site Internet de JDeveloper. http://www.oracle.com/technology/products/jdev/index.html (dernière consultation le 25 mars 2006).

[16] Liste d'IDE pour Java. http://java.developpez.com/outils/edi/ (dernière consultation le 10 mars 2006).

[17] Site Internet de Myeclipse. http://www.myeclipseide.com/ (dernière consultation le 25 mars 2006).

[18] Site Internet de Netbeans. http://www.netbeans.org (dernière consultation le 25 mars 2006).

[19] Site Internet de C#Builder. http://www.csharpbuilder.info (dernière consultation le 26 mars 2006).

[20] Site Internet de Improve technologies, C# Plugin for Eclipse. http://www.improve-technologies.com/alpha/esharp/ (dernière consultation le 26 mars 2006).

[21] Site Internet de Monodevelop. http://www.monodevelop.com/Main_Page (dernière consultation le 26 mars 2006).

[22] Site Internet de SharpDevelop. http://www.icsharpcode.net/opensource/sd/ (dernière consultation le 26 mars 2006).

[23] Site Internet de Visual Studio. http://msdn.microsoft.com/vstudio/ (dernière consultation le 26 mars 2006).

[24] Site Internet de Visual Studio Express. http://msdn.microsoft.com/vstudio/express (dernière consultation le 26 mars 2006).

[25] Site Internet de notepad. http://notepad.org/ (dernière consultation le 28 mars 2006).

[26] Site Internet de BEA Weblogic. http://www.bea.com (dernière consultation le 28 mars 2006).

[27] Site Internet de GlassFish. https://glassfish.dev.java.net/ (dernière consultation le 28 mars 2006).

[28] Site Internet de JBoss. http://www.jboss.org/ (dernière consultation le 28 mars 2006).

[29] Site Internet de Sun Java System Application Server. http://www.sun.com/software/products/appsrvr/index.xml (dernière consultation le 28 mars 2006).

[30] Site Internet de Websphere. http://www.ibm.com/websphere (dernière consultation le 28 mars 2006).

[31] Site Internet Java de Blackdown. http://www.blackdown.org/ (dernière consultation le 28 mars 2006).

[32] EJB 3.0 - Entity beans. http://hibernate.org/~gavin/ebejb3.pdf (dernière consultation le 22 mars 2006).

[33] Java - Technologie des HotSpot. http://java.sun.com/products/hotspot/ (dernière consultation le 23 mars 2006).

[34] Site Internet Java de IBM. http://www-128.ibm.com/developerworks/java (dernière consultation le 28 mars 2006).

[35] Site Internet J2EE. http://java.sun.com/javaee/index.jsp (dernière consultation le 23 mars 2006).

[36] Wikipédia - J2EE. http://fr.wikipedia.org/wiki/J2EE (dernière consultation le 1 mars 2006).

[37] Wikipédia - Java EE. http://en.wikipedia.org/wiki/Java_EE (dernière consultation le 24 mars 2006).

[38] Wikipédia - JVM. http://en.wikipedia.org/wiki/Java_virtual_machine (dernière consultation le 23 mars 2006).

[39] Site Internet Java de Lejos. http://lejos.sourceforge.net/ (dernière consultation le 28 mars 2006).

[40] Site Internet Java de Sun. http://java.sun.com (dernière consultation le 23 mars 2006).

[41] Answers - ADO.NET. http://www.answers.com/topic/ado-net (dernière consultation le 16 mars 2006).

[42] ADO.NET - DataSet. http://dotnet.developpez.com/tutoriels/ado2/csharp/ (dernière consultation le 17 mars 2006).

[43] ADO.NET - DataSet typés. http://www.supinfo-projects.com/fr/2005/datatype/ (dernière consultation le 22 mars 2006).

[44] ADO.NET - Didacticiel, démarrages rapides. http://fr.gotdotnet.com/quickstart/howto/doc/adoplus/UpdateDataFromDB.aspx (dernière consultation le 22 mars 2006).

[45] ADO.NET - Présentation. http://developpeur.journaldunet.com/tutoriel/asp/011019asp_adonet.shtml (dernière consultation le 17 mars 2006).

[46] Windows 2003 Application Server technologies. http://www.microsoft.com/windowsserver2003/appserver/features.mspx (dernière consultation le 6 mars 2006).

[47] Site Internet de Boo. http://boo.codehaus.org (dernière consultation le 26 mars 2006).

[48] Site Internet de Cassini. http://www.asp.net/Projects/Cassini/Download/ (dernière consultation le 25 mars 2006).

[49] MSDN - Concepte pour la gestion de version des DLL. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/dplywithnet.asp (dernière consultation le 26 mars 2006).

[50] Site Internet de DotGNU. http://www.dotgnu.org/ (dernière consultation le 23 mars 2006).

[51] FAQ - ASP.NET. http://www.syncfusion.com/FAQ/aspnet/WEB_c2c.aspx (dernière consultation le 4 avril 2006).

[52] Framework .NET. http://msdn.microsoft.com/academic/techdown/techprod/netframework/default.aspx (dernière consultation le 6 mars 2006).

[53] Site Internet du framework .NET. http://msdn.microsoft.com/netframework/downloads/default.aspx (dernière consultation le 25 mars 2006).

[54] Site Internet de IKVM. http://www.ikvm.net/ (dernière consultation le 23 mars 2006).

[55] Liste des languages de programmation supportée par Mono. http://boo.codehaus.org (dernière consultation le 26 mars 2006).

[56] Liste des languages de programmation supportée par .NET. http://www.dotnetpowered.com/languages.aspx (dernière consultation le 09 avril 2006).

[57] Site Internet de Mono. http://www.mono-project.com/Main_Page (dernière consultation le 23 mars 2006).

[58] Test de Windows Vista, partie .NET. http://www.pcinpact.com/articles/a/172/0.htm?pge=7 (dernière consultation le 27 mars 2006).

[59] Test de Windows Vista, partie sécurité. http://www.pcinpact.com/articles/a/174/0.htm?pge=2 (dernière consultation le 27 mars 2006).

[60] Site Internet de Rotor. http://www.microsoft.com/resources/sharedsource/Licensing/CSharp_JScript_CLI.mspx (dernière consultation le 1 mars 2006).

[61] Wikipédia - .NET. http://fr.wikipedia.org/wiki/Microsoft_.NET (dernière consultation le 1 mars 2006).

[62] Site Internet de SmartSVN. http://www.smartcvs.com/smartsvn/ (dernière consultation le 23 mars 2006).

[63] Site Internet du groupe de recherche Software Engineering. http://diuf.unifr.ch/softeng/ (dernière consultation le 22 mars 2006).

[64] Site Internet de l'ECMA. http://www.ecma-international.org/ (dernière consultation le 23 mars 2006).

[65] Free Documentation Licence (GNU FDL). http://www.gnu.org/licenses/fdl.txt (dernière consultation le 23 mars 2006).

[66] Licence de Documentation Libre (GNU FDL). http://www.idealx.org/dossier/oss/gfdl.fr.html (dernière consultation le 23 mars 2006).

[67] Site Internet de l'ISO. http://www.iso.org/ (dernière consultation le 25 mars 2006).

[68] Site Internet de TortoiseSVN. http://tortoisesvn.tigris.org/ (dernière consultation le 23 mars 2006).

[69] Serveur d'application .NET vs J2EE. http://www.adae.gouv.fr/IMG/rtf/serveurs_applications.rtf (dernière consultation le 28 mars 2006).

[70] .NET pour contrer J2EE. http://www.01net.com/article/202500.html (dernière consultation le 24 mars 2006).

[71] Comparatif ASP.NET - Struts. http://msdn.microsoft.com/asp.net/default.aspx?pull=/library/en-us/dnaspp/html/aspnet-aspnet-j2ee-struts.asp (dernière consultation le 7 mars 2006).

[72] C# du point de vu d'un développeur Java. http://www.25hoursaday.com/CsharpVsJava.html (dernière consultation le 8 mars 2006).

[73] Architecture .NET vs J2EE. http://www.dotnetguru.org/articles/architecturedotnet.htm (dernière consultation le 24 mars 2006).

[74] Couche de service, pooling .NET vs J2EE. http://www.dotnetguru.org/articles/CoucheService.htm (dernière consultation le 24 mars 2006).

[75] Comparaison de performances entre J2EE et .NET. http://www.dotnetguru.org/articles/Comparatifs/benchJ2EEDotNET/reload/benchmarkreload.htm (dernière consultation le 23 mars 2006).

[76] Comparatif C# Builder et VS. http://www.dotnetguru.org/articles/CSharpbuilder/csharpbuildervsdotnet.htm (dernière consultation le 7 mars 2006).

Index

S

T

W

X