



<http://www.labo-dotnet.com>

Framework .NET

SUPINFO DOT NET TRAINING COURSE

Auteur : Nicolescu Matthieu

Versi on 1.1 – 4 septembre 2003

Nombre de pages : 22



Ecole Supérieure d'Informatique de Paris
23. rue Château Landon 75010 – PARIS
www.supinfo.com

Table des matières

1. INTRODUCTION :	3
1.1. HISTORIQUE :	3
1.2. QU'EST CE QUE .NET ? :	3
2. PRESENTATION DU FRAMEWORK :	4
2.1. POURQUOI FRAMEWORK .NET ? :	4
2.2. ELEMENTS DU FRAMEWORK .NET:	5
2.3. LES BENEFICES DE .NET POUR LES ENTREPRISES :	5
2.4. NET ET JAVA:	6
2.5. REPERTOIRE D'INSTALLATION :	6
2.6. LES COUCHES DU FRAMEWORK :	7
3. COMMON LANGUAGE RUNTIME :	8
3.1. PRESENTATION DE LA CLR :	8
3.2. FONCTIONNEMENT DETAILLE DE LA CLR :	9
3.3. MSIL :	11
3.4. ASSEMBLIES :	12
3.5. GAC :	14
3.6. LES DOMAINES D'APPLICATION :	14
4. CLASSES DE BASE :	15
4.1. INTRODUCTION AUX CLASSES DE BASE:	15
4.2. NORME DU FRAMEWORK :	15
4.3. PRESENTATION DES PRINCIPAUX NAMESPACE :	16
4.3.1. <i>System</i> :	16
4.3.2. <i>System.IO</i> :	16
4.3.3. <i>System.Data</i> :	16
4.3.4. <i>System.Collections</i> :	16
4.3.5. <i>System.Net</i> :	16
4.3.6. <i>System.Reflection</i> :	16
4.3.7. <i>System.Xml</i> :	16
5. ASP.NET & WINDOWS FORM :	18
5.1. ASP.NET:	18
5.1.1. <i>Présentation de ASP.NET</i> :	18
5.1.2. <i>ASP.NET un langage compilé</i> :	18
5.2. WINDOWS FORM :	19
6. PORTABILITE DE .NET :	20
6.1. STANDARDISATION DE .NET:	20
6.2. CLS :	20
7. NET SERVER :	21
7.1. PRESENTATION .NET SERVER :	21
7.2. VERSIONS DE .NET SERVER :	21
7.3. MISE A JOUR DE .NET SERVER :	21

1. Introduction :

1.1.Historique :

L'informatique, durant ces vingt dernières années, a considérablement changé que ce soit du côté hardware que du côté software.

Nous vous proposons tout d'abord de faire un retour en arrière et de voir les grandes dates de l'ère Microsoft :

- 1981 : Sortie du Système d'exploitation MS-DOS et du langage BASIC.
- 1990 : Première version graphique de Windows et sortie du langage Visual BASIC.
- 1995 : Microsoft met un pied dans le monde de l'Internet en sortant un navigateur Web « Internet Explorer », qui avait à l'époque un grand concurrent : « Netscape ». Microsoft sort en parallèle son serveur Web IIS et propose aux développeurs une interface de développement appelée Visual Studio.
- 2000 : Début de l'ère .NET.

1.2.Qu'est ce que .NET ? :

Avant de rentrer dans les détails répondons tout d'abord à une question toute simple : Qu'est ce que .NET ?

Contrairement aux idées reçues .NET n'est pas un langage ou un logiciel : .NET est en fait la nouvelle stratégie de Microsoft. .NET se présente donc comme une vision de la prochaine génération d'applications qui repose sur des standards tels que XML, HTTP, SOAP, WSDL...

Pour donner corps à cette vision, Microsoft se repose sur quatre piliers :

- le .NET Framework
- les .NET Servers (futures versions Serveur de Microsoft).
- Visual Studio .NET
- .NET Enterprise servers (tous les logiciels serveurs comme Commerce Server, SQL Server, Content Management Server...)

Le .NET Framework est un environnement qui est distribuable gratuitement sur toutes les versions de Windows depuis Windows 95. Les .NET Servers sont la nouvelle génération des Serveurs Microsoft qui vont donc succéder aux Windows 2000 Servers.

Nous allons à présent étudier plus en détails le Framework : les nouveautés que le Framework peut apporter aux applications du futur, les différentes couches du Framework et ses utilitaires.

2. Présentation du Framework :

Nous allons dans ce chapitre vous présenter le Framework, voir ce qu'il peut apporter dans le monde de l'entreprise puis nous vous présenterons les différentes couches du Framework .NET.

2.1. Pourquoi Framework .NET ? :

En réalisant le Framework .NET, Microsoft voulait tout d'abord sortir de l'enfer des technologies COM¹ : en effet toutes les versions COM devaient supporter les anciennes versions ce qui était assez lourd à gérer. De plus la communication pour accéder aux objets COM se faisait toujours sur le même port d'écoute : cela ne posait pas de problème pour l'intranet de l'entreprise, mais lorsqu'une entreprise voulait utiliser un objet COM d'une autre entreprise, cela posait souvent des problèmes car la communication ne pouvait pas s'effectuer pour des raisons de sécurité (Firewall de l'entreprise).

Mais ce n'est pas pour autant que les technologies COM sont mortes : en effet il est tout à fait concevable de développer un objet COM puis ensuite de l'utiliser en .NET.

Pour résoudre ce dernier problème de sécurité avec les objets COM, le Framework propose les Services Web plus couramment appelés les WebServices. L'avantage des WebServices est que la communication entre le client et le serveur (ici le WebService) se fait via le protocole http, c'est à dire via le port 80 : il n'y a donc plus de problèmes de communication entre les entreprises. De plus les informations émises par le Webservice sont sous format XML donc peuvent être traitées par la quasi totalité des langages tels que le C, C++, Java, Perl, Python, PHP, Cobol... et bien sûr les langages .NET. Nous verrons plus tard que le fait de faire passer du XML via HTTP représente le protocole SOAP.

Ce qui fait donc la force de .NET c'est qu'il regroupe plusieurs technologies tels que COM, Applications Web (ASP.NET) , Applications Windows (Windows Form) et Applications Mobiles (Compact Framework)...

Pour ce qui est des applications Web, le changement est assez net entre l'ASP et l'ASP.NET, car le langage n'est plus interprété mais compilé. En ASP, le développeur n'avait pas le choix du langage alors qu'en ASP.NET il a le choix.

En effet la plateforme .NET est multi-langages ce qui est une grande nouveauté par rapport à ses concurrents direct.

Cela peut donc s'avérer très pratique lors du développement de grands projets : en effet, en utilisant la même technologie ici la technologie .NET, les développeurs pourront choisir leur propre langage sans avoir à se soucier si sa partie sera compatible avec celui de son collègue qui développe dans un autre langage.

Actuellement le Framework .NET supporte une vingtaine de langages dont le C#, VB.NET, J#, COBOL, Eiffel#, PERL.NET...

De plus avec le Framework, Microsoft a voulu mettre fin aux nombreux problèmes causés par des dll : il était impossible d'avoir plusieurs versions de dll en même temps et cela posait donc de gros

¹ : COM (Component Object Model). Les objets COM fournissent des fondations pour le développement de logiciel. On peut accéder aux fonctionnalités d'un objet COM via ses interfaces.

problème de compatibilité pour les applications. En effet lorsque par exemple au sein d'une entreprise on changeait de version de dll, certaines applications ne marchaient tout simplement plus car elles n'étaient pas compatibles avec la nouvelle version de la dll.

En .NET lorsque vous créez une application Windows ou Web, une assemblée couramment appelée assembly est automatiquement créée. Une assembly est en fait le conteneur physique de classes de votre projet et donc ces classes peuvent être utilisées par plusieurs applications. Et il est bien sûr possible d'avoir plusieurs versions de la même assembly.

2.2.Eléments du Framework .NET:

Contrairement au API Windows, le Framework .NET est totalement objet : ce n'est plus en effet une simple liste de méthode comme les API Windows. De plus les classes du Framework sont ordonnées hiérarchiquement. En effet, toutes les classes concernant la manipulation des fichiers XML se trouveront dans un namespace ² « System.XML » ; les classes permettant de faire de la manipulation de données seront dans le namespace « System.Data »...Cela facilite donc grandement la tâche du développeur pour trouver les bonnes classes.

Le Framework intègre de base des classes pour la connexion aux bases de données via ADO.NET, OLEDB, ODBC, ODBC.NET... Ces classes permettent donc de se connecter à toutes les bases de données existantes sur le marché telles que SqlServer, Oracle, Access, Sybase... Il existe aussi actuellement des drivers optimisés pour SqlServer et Oracle.

Il existe deux versions téléchargeables du Framework qui sont totalement gratuite:

- « Framework Redistribuable » : Version qui contient les classes de bases ainsi que l'environnement d'exécution .NET.
- « Framework SDK » : Version identique à la version précédente à la différence près que la version SDK contient les compilateurs CSharp et VB.NET ainsi que des utilitaires. La version SDK va donc permettre de créer des applications .NET.

2.3.Les bénéfices de .NET pour les entreprises :

Les avantages de .NET pour les entreprises sont multiples et variés : tout d'abord, la productivité, c'est à dire le développement des applications, est plus rapide. Nous avons un gain de productivité en .NET car tout est objet en .NET, comme nous venons de le voir. Lorsque vous développez des composants, vous pouvez les réutiliser dans plusieurs applications ce qui évite de développer les mêmes méthodes à chaque fois.

Vous pouvez très bien avoir un composant « Employé » qui sera utilisé par toutes les applications de votre entreprise : ce composant permettra donc d'avoir des informations sur les employés et de modifier ces informations. Le composant « Employé » fait donc le lien entre la couche de données de l'entreprise (Bases de données) et la couche applicative.

Pour faciliter la tâche du développeur, Microsoft propose un environnement de développement appelé Visual Studio .NET. Avec Visual Studio le développement .NET devient beaucoup plus rapide et simple : en effet Visual Studio dispose d'un éditeur WYSIWYG (What You See Is What You Get) et il est entièrement RAD (Rapid Application Development). C'est à dire que lorsque vous voulez utiliser un composant il vous suffit de le faire glisser sur votre page ou votre form (Principe du Drag And Drop).

Visual .NET intègre aussi plusieurs outils et le développeur n'est plus obligé d'utiliser plusieurs logiciels pour développer ses applications. Par exemple, via le Solution Explorer de Visual .NET, le

² : Namespace = Espaces de noms permettant de regrouper plusieurs classes.

développeur peut accéder à sa base de données que ce soit un SqlServer, Oracle ou autre... Et comme nous l'avons vu auparavant, Visual .NET permet de développer des applications Web, Windows, WebService, applications mobiles.

Le Framework .NET est multi-langages, donc passer d'un langage à un autre ne pose pas de problème. Le plus dur lorsque un développeur change de langage n'est pas l'apprentissage de la syntaxe, mais l'utilisation des fonctionnalités propre au langage. Or en .NET, tous les langages utilisent les mêmes fonctionnalités, c'est à dire les mêmes classes de base donc le changement ne pose aucun problème au développeur. Par exemple l'apprentissage du VB.NET pour un développeur C# peut se faire en un jour !

Un autre point très important dans la stratégie .NET de Microsoft surtout pour les entreprises, c'est la sécurité. En effet, Microsoft notamment avec leurs serveurs a été beaucoup critiqué par les professionnels de la sécurité. Les .NET Servers qui sont les successeurs des serveur Windows 2000 changent complètement de stratégie : en effet, tout est fermé par défaut sur les .NET Servers. Par exemple, pour faire fonctionner votre Serveur Web, vous devrez vous même donner les droits nécessaire à l'utilisateur lançant le processus du Serveur Web. De plus la nouvelle version du Serveur Web IIS 6 est beaucoup plus fiable, plus robuste, plus sécurisée et surtout plus performante, car IIS 6 passe d'une architecture multi-threads à une architecture multi processus comme le serveur Web Apache qui est reconnu pour ses performances.

2.4.NET et Java:

Le Framework est assez similaire à l'environnement de Java, le Java Runtime Environnement (JRE). En effet comme la JRE, le Framework dispose d'une machine virtuelle appelé la CLR (Common Language Runtime) ainsi que des classes de base mises à disposition du développeur. Alors .NET est il un clone de Java ?

Non, .NET n'est pas un clone de Java. Tout d'abord la Plateforme .NET est la seule pour l'instant à être multi langages. Cela a été rendu possible par la définition de format de type et de description standard. Cette définition s'appelle la CLS (Common Language Specification).

De plus Java et .NET n'ont pas les mêmes objectifs. La stratégie .NET est axée sur les Webservices et sur les mobiles à partir de 2002 avec le Compact Framework, alors que Java s'adresse directement aux applications mobiles, tout en offrant un bon support des WebServices.

Pour ce qui concerne les performances entre .NET et Java, cela dépasse le cadre de notre sujet mais il faut savoir qu'une solution n'est pas forcément meilleure qu'une autre. Cela dépend principalement de l'environnement dans lequel on se trouve. Vous ne verrez donc aucun article sérieux disant que .NET est meilleur que Java ou que Java est meilleur que .NET.

2.5.Répertoire d'installation :

Nous allons voir au tout au long de cet essentiel qu'il existe plusieurs utilitaires. Certains sont propres au Framework SDK, d'autres à Visual Studio .NET.

Il faut savoir tout d'abord que le Framework est installé dans le chemin suivant :

- c:\WinNt\Microsoft.NET

Ce chemin peut bien sûr varier : en effet si votre système est installé sur le d:\ le chemin sera alors : « d:\WinNt\Microsoft.NET ».

Tous les utilitaires du Framework que nous étudierons au cours de cet essentiel se trouvent dans le chemin suivant :

- `c:\WinNt\Microsoft.NET\Framework\numero_version_framework\`

Tous les utilitaires de Visual Studio .NET sont installés dans le chemin :

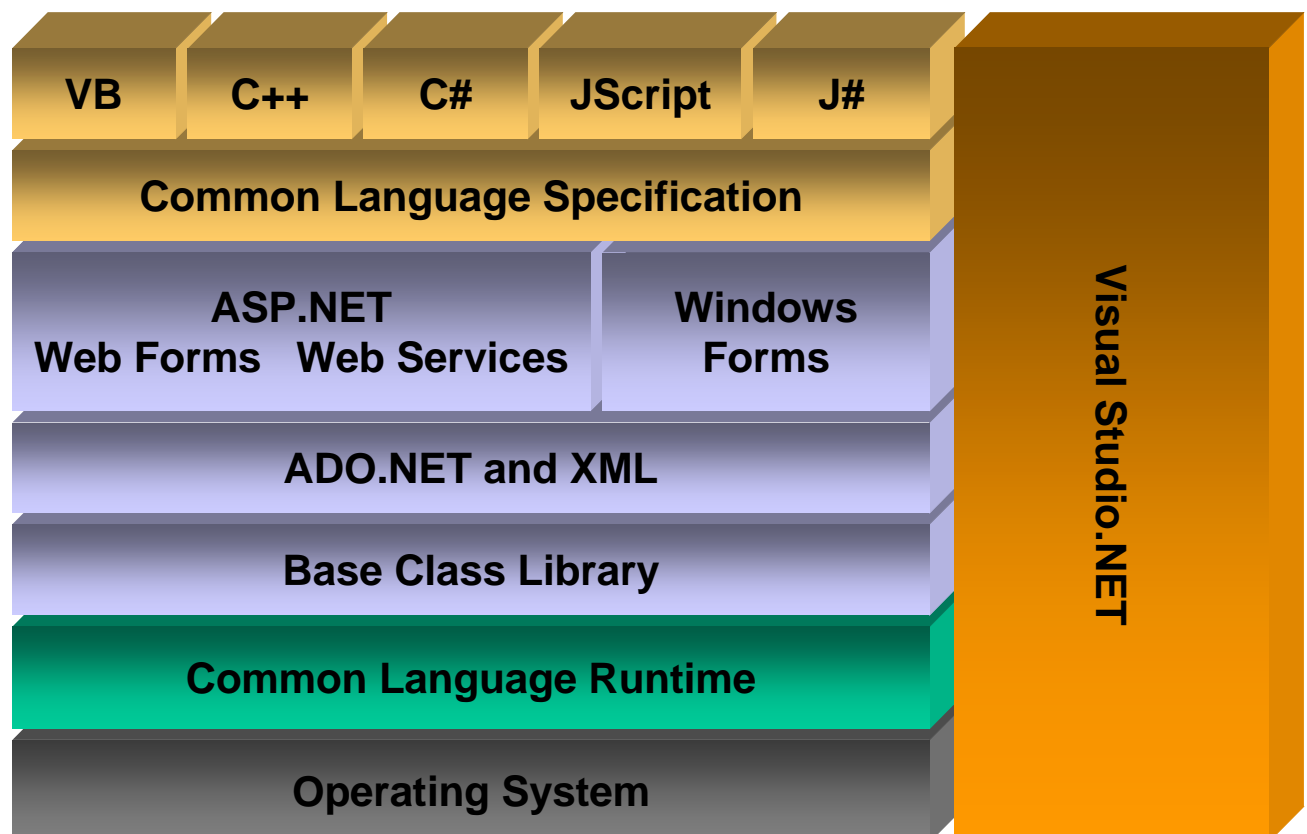
- `c:\Program Files\Microsoft Visual Studio .NET\FrameworkSDK\Bin`

Nous vous conseillons fortement d'entrer ces deux derniers chemins (à modifier selon les configuration de votre système) dans votre valeur d'environnement Windows « Path ».

Cela vous permettra d'avoir accès directement aux utilitaires du Framework et de Visual Studio .NET en ligne de console sans à avoir à taper le chemin complet de ces utilitaires.

2.6. Les couches du Framework :

Voici les différentes couches du Framework que nous allons étudier dans les chapitres suivants :



3. Common Language Runtime :

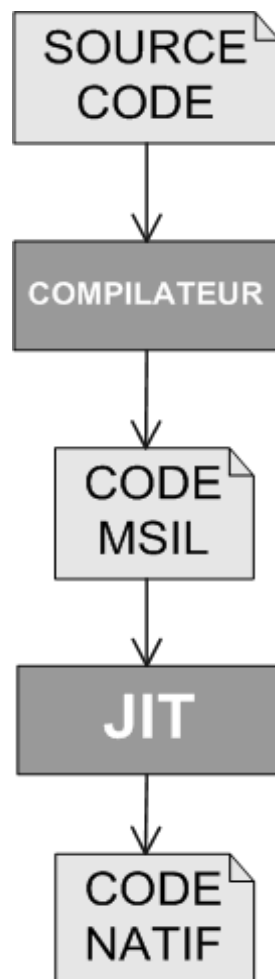
La CLR est un des piliers du Framework .NET : nous verrons dans ce chapitre quel est le rôle de la CLR et quel est son fonctionnement.

3.1.Présentation de la CLR :

Comme Sun avec Java, Microsoft avec .NET a choisi de se munir d'une machine virtuelle appelé la CLR (Common Language Runtime) et d'un code intermédiaire nommé MSIL (Microsoft Intermediate Language).

La CLR se place juste au dessus du Système d'exploitation et c'est la CLR qui va exécuter les applications .NET puis gérer la gestion de mémoire et la sécurité de l'application.

Dans des langages traditionnels tels que le C ou le C++, le code était compilé dans un code natif, c'est-à-dire un code machine qui était propre au processeur de la machine. La différence entre ces langages et les langages .NET est qu'en .NET les langages ne sont pas compilés en langage machine, mais en code intermédiaire comme le montre le schéma ci-dessous. Ce n'est qu'au moment de l'exécution de l'application que la CLR va interpréter le code intermédiaire (MSIL) en code machine via son compilateur JIT (Just In Time) : le code sera donc compilé à la volée.



Le schéma ci-dessus représente de manière très simplifiée le processus de compilation et d'exécution d'une application .NET.

Il faut savoir tout d'abord que chaque langage possède son propre compilateur : pour C# le compilateur se comme « csc », pour VB.NET c'est le « vbc »... Le compilateur ne va donc pas compiler le programme en code machine mais en code intermédiaire appelé le MSIL : c'est à dire que si vous compilez deux programmes, un en C# et le deuxième en VB.NET, vous obtiendrez le même code. C'est donc pour cette raison que tous les langages .NET ont les mêmes performances. Il n'y a donc pas un langage qui est plus performant qu'un autre.

Ce code intermédiaire donne aux langages .NET une grande portabilité, car le code intermédiaire n'est pas propre à la plateforme ou au processeur et ce n'est que lors de l'exécution de l'application que le code intermédiaire est compilé à la volée en code machine. Microsoft nomme ce principe « Execute on Many Platforms » alors que SUN l'appelle « Write One, Run Anywhere » (WORA).

Dans l'absolu on pourrait faire fonctionner les applications .NET sur n'importe quelle plateforme mais pour l'instant le Framework n'est implémenté que sur les versions de Windows à partir de Windows 98. De plus il existe des projets d'implémentation du Framework sur d'autres plateformes telles que Linux avec le projet « Mono ». Il est en effet possible à présent d'écrire un programme C# et de l'exécuter sur Linux. Mais actuellement le projet « Mono » est encore à ses débuts et n'a donc pas encore implémenté toutes les classes du Framework. Pour plus de détails sur le projet Mono, vous pouvez consulter leur site officiel :

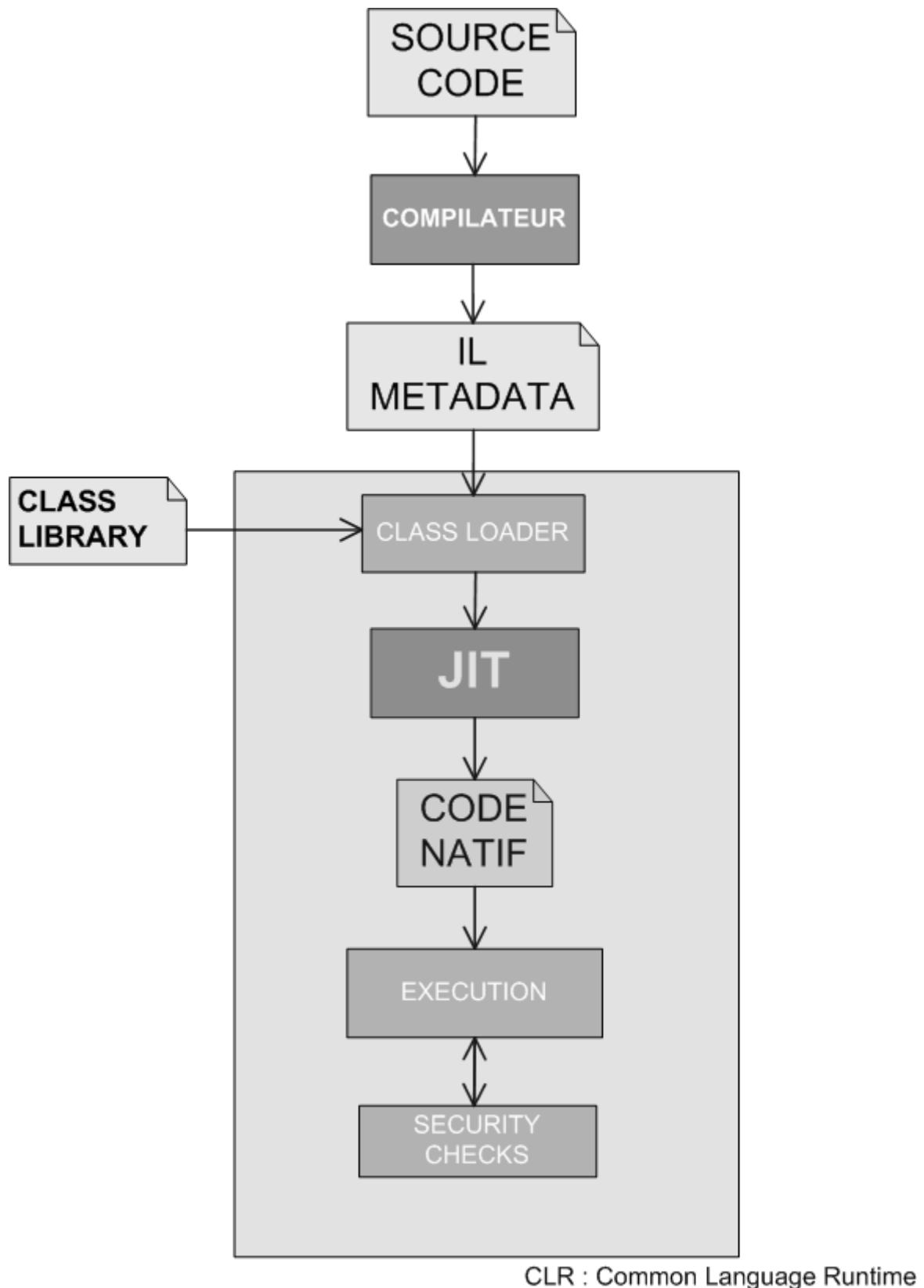
<http://www.go-mono.net>

3.2.Fonctionnement détaillé de la CLR :

Nous avons vu que la CLR permettait via son compilateur JIT de compiler à la volée du langage intermédiaire en code natif. Mais le travail de la CLR ne s'arrête pas là : en effet l'environnement .NET comprend aussi un mécanisme de « ramasse-miettes » et gère aussi la sécurité de l'application tout au long de son exécution.

Le mécanisme de ramasse-miettes de la CLR permet de contrôler le cycle de vie des objets en libérant la mémoire occupée par un objet qui n'est plus référencé dans le programme. Ce mécanisme se nomme le « Garbage Collector » et permet donc de libérer la mémoire qui n'est plus utilisée par votre application. Le « Garbage Collector » travaille tout seul et il est déconseillé de faire appel à lui au sein d'un programme pour libérer un objet ou de la mémoire.

La CLR gère aussi la sécurité des applications .NET et le développeur peut donc décider de donner des droits restreints à l'application. Par exemple on peut très bien interdire à l'application l'accès au disque dur de la machine.



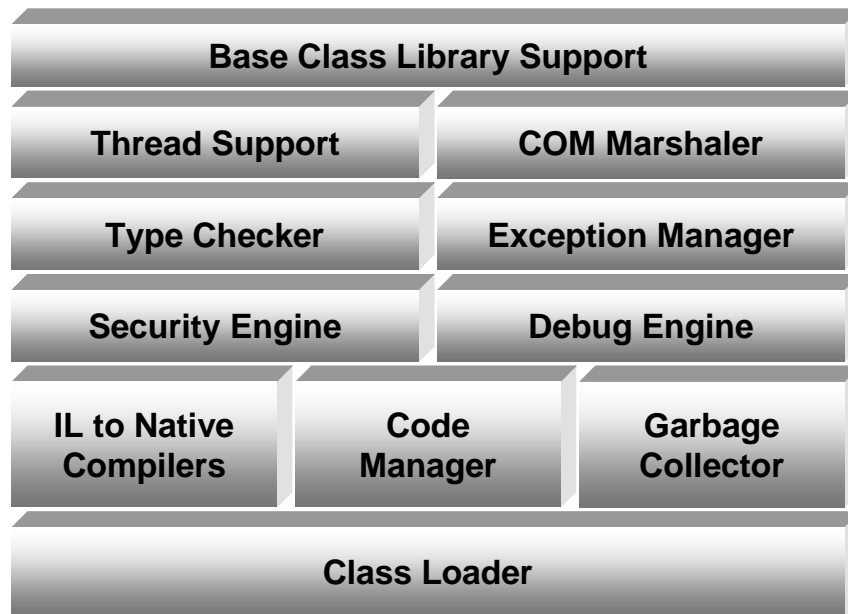
Comme vous pouvez le voir, il y a quelques petites modifications par rapport à notre premier schéma. Tout d'abord, les compilateurs du Framework que ce soit le compilateur C#, VB.NET... ne compile pas l'application en un simple fichier contenant du code MSIL mais en une assembly que nous allons

étudier dans le prochain chapitre. En fait une assembly contient du langage intermédiaire, des Metadata ainsi qu'un Manifest.

Lors de l'exécution de l'application, la CLR prend le relais en chargeant le type et les classes nécessaires puis fait appel au compilateur Just In Time (JIT) pour compiler le code MSIL en code natif. Durant l'exécution de votre application, la CLR gère la sécurité de votre application et décide si elle aura les droits nécessaires pour exécuter les routines voulues.

Nous verrons dans un autre essentiel comment gérer la sécurité du Framework.

Etudions à présent plus en détail les différents éléments qui sont appelés lors de l'exécution de votre application .NET à travers ce nouveau schéma :



Schema: source Microsoft

Comme vous pouvez le voir, le code MSIL de votre application est tout d'abord compilé en code natif. Mais le travail de la CLR ne s'arrête pas là !

En effet tout au long de l'exécution de l'application, le « Garbage Collector » permet de libérer l'espace mémoire des objets qui ne seront plus référencés.

Le « Garbage Collector » libère de l'espace mémoire que lorsque le processeur le permettra. En effet, le Garbage Collector attend que les ressources processeurs soient de niveau assez bas pour pouvoir libérer cet espace. Il est possible de forcer le Garbage Collector à libérer de l'espace mémoire, mais ce n'est pas conseillé car cela pourrait dégrader les performances de votre application au lieu de les améliorer.

Comme nous l'avons vu un peu plus haut, la CLR gère aussi la sécurité de votre application pour vérifier qu'aucune violation sur le système n'a été effectuée.

Nous verrons dans d'autres essentiels les systèmes d'Exception pour gérer les erreurs de vos applications, le Thread Support...

3.3.MSIL :

Vous devez savoir à présent que le MSIL est le langage intermédiaire de Microsoft : Microsoft Intermediate Language. Tous les compilateurs des langages .NET génèrent du code MSIL qui est ensuite interprété par la CLR.

Pour lire ou générer du code MSIL, vous avez besoin de deux utilitaires : ildasm et ilasm.

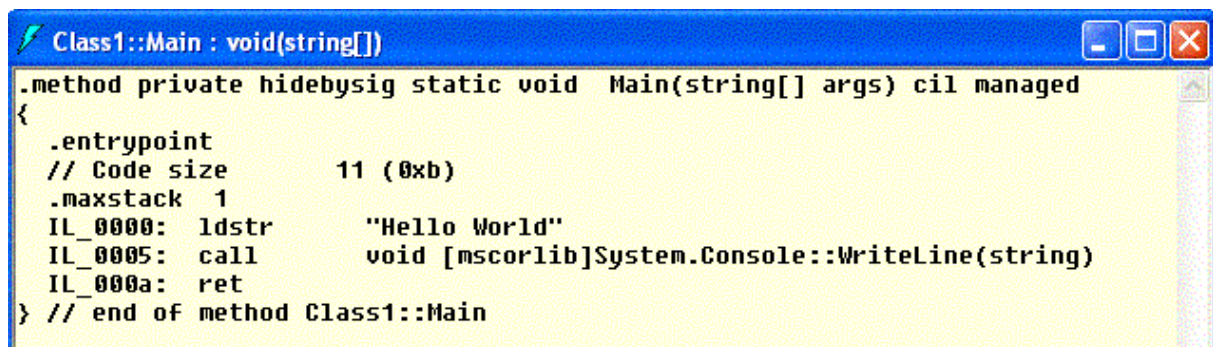
Ildasm est en fait un désassembleur qui vous permet de lire du code MSIL à partir d'une assembly (fichier .dll qu'il ne faut surtout pas confondre avec les anciens DLL Windows). Ildasm se trouve dans le répertoire des utilitaires de Visual Studio .NET.

Ilasm est donc l'assembleur et il se trouve dans le répertoire des utilitaires du Framework.

Nous allons nous intéresser au désassembleur ildasm, qui va donc nous permettre de voir notre premier code en MSIL. Dans notre exemple, nous allons ouvrir l'assembly d'une application console qui se contente d'afficher « Hello World ».

Pour cet exemple, nous ne vous demandons pas de savoir comment créer une application car cela sera vu dans le prochain essentiel.

Voici le code MSIL de notre application « Hello World » :



```
Class1::Main : void(string[])
.method private hidebysig static void Main(string[] args) cil managed
{
    .entrypoint
    // Code size          11 (0xb)
    .maxstack 1
    IL_0000: ldstr        "Hello World"
    IL_0005: call         void [mscorlib]System.Console::WriteLine(string)
    IL_000a: ret
} // end of method Class1::Main
```

Comme vous pouvez le constater, le code MSIL est compréhensible par rapport à l'assembleur x86. Nous pouvons très bien voir que le code MSIL appelle la méthode « WriteLine » de la classe Console qui permet en fait d'afficher notre « Hello World ».

Pour protéger vos application d'un possible désassemblage (c'est à dire qu'une personne puisse retrouver votre code source à partir du fichier MSIL), vous pouvez utiliser des Obfuscators qui permettent de modifier le code intermédiaire.

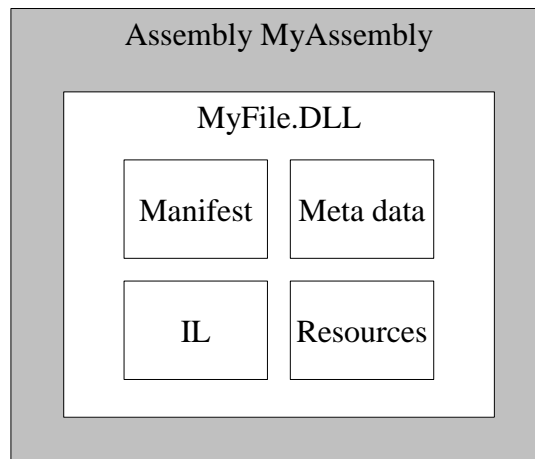
3.4.Assemblies :

Qu'est ce qu'une assembly ? Comme nous l'avons vu auparavant, une assembly est le conteneur physique des classes qui seront utilisées par votre ou vos applications. L'assembly est automatiquement générée à chaque fois que vous créez une DLL ou un exécutable.

Une assembly est en fait une collection de plusieurs fichiers : Metadata, Manifest, IL et ressources.

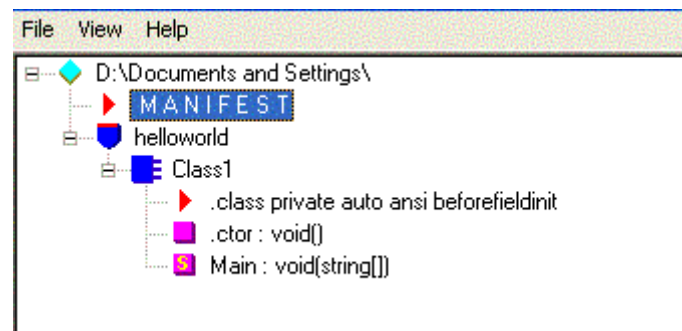
Le Manifest définit toutes les exigences de contrôle de version, l'auteur de l'assembly, les autorisations, et les dépendances avec les autres assemblies (et pour chaque dépendance il y a le numéro de version de l'assembly, car il peut avoir plusieurs versions de la même assembly, contrairement aux DLL Windows).

Les métas données de l'assembly permettent d'interroger l'assembly sur ses types de données, méthodes... Une assembly contient enfin le code MSIL de votre application qui sera ensuite compilé en code natif par la CLR lors de l'exécution de l'application.



Nous allons à présent reprendre notre exemple d'application « HelloWorld » en utilisant le désassembleur ildasm pour pouvoir voir le Manifest.

Lorsque nous ouvrons l'assembly de notre application, la fenêtre suivante s'ouvre :



Toute cette arborescence constitue notre assembly. Vous pouvez voir qu'à chaque nœud, il y a une icône. Voici la légende de ces icônes :

Namespace:	
Class:	
Interface:	
Value Class:	
Enum:	
Method:	
Static method:	
Field:	
Static field:	
Event:	
Property:	
Manifest or a class info item	

Si vous n'avez jamais développé avec un langage objet que ce soit avec le C++, Java, C#..., vous ne comprendrez pas tous les termes de la légende. Vous en connaîtrez la signification lors du prochain essentiel qui traitera de la Programmation Orienté Object ainsi de la programmation avec C#.

Pour voir le Manifest de l'application « HelloWorld », il vous suffit de double-cliquer sur « Manifest » : vous verrez alors un fichier assez technique que nous ne détaillerons pas dans cet

essentiel. Ce fichier définit toutes les exigences de contrôle de version et les dépendances de l'assembly.

3.5.GAC :

GAC veut dire Global Assembly Cache. La GAC va en fait contenir toutes les Assemblies du Framework. Nous verrons dans les prochains essentiels qu'il est aussi possible de déployer votre Assembly dans la GAC, ce qui est préférable pour le fonctionnement de vos applications. En effet, la CLR recherche les Assemblies tout d'abord dans la GAC.

Les assemblies sont stockées dans le répertoire suivant :

- c:\WinNt\Assembly\

Une fois que vous êtes dans ce répertoire, vous allez avoir une fenêtre de ce type :

Global Assembly Name	Type	Version	Culture	Public Key Token
System.Data.resources		1.0.2411.0	fr	b77a5c561934e089
System.Design	Native Images	1.0.3300.0		b03f5f7f11d50a3a
System.Design		1.0.3300.0		b03f5f7f11d50a3a
System.Design.resources		1.0.2411.0	fr	b03f5f7f11d50a3a
System.DirectoryServices		1.0.3300.0		b03f5f7f11d50a3a
System.DirectoryServices.resources		1.0.2411.0	fr	b03f5f7f11d50a3a
System.Drawing	Native Images	1.0.3300.0		b03f5f7f11d50a3a
System.Drawing		1.0.3300.0		b03f5f7f11d50a3a

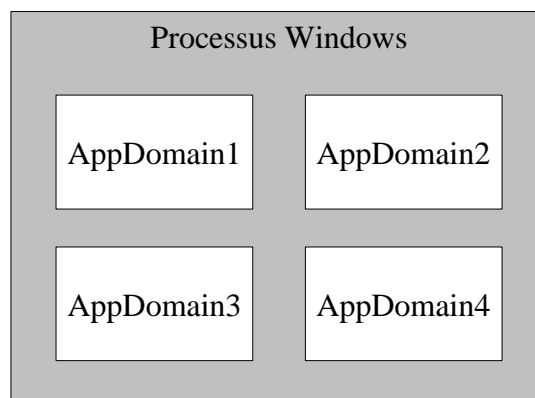
Vous pouvez donc lister le nom de toutes vos assemblies et leur numéro de version, leur type, culture et leur « Public Key Token » (pour le cryptage des assemblies).

Il existe aussi un utilitaire en ligne de commande qui permet de lister toutes les assemblies qui sont contenues dans la GAC : cet utilitaire se nomme « gacutil ».

3.6.Les domaines d'application :

Sous Windows, les applications fonctionnent dans un processus (c'est-à-dire des ressources et des threads). Cette architecture pose un problème car il faut beaucoup de ressources pour la création et gestion des processus.

Sous .NET, il y a la notion de domaine d'application. Un domaine d'application est identique à un processus mais plusieurs domaines d'applications peuvent s'exécuter dans un processus ce qui a pour conséquence un gain de performance et de ressources non négligeable.



4. Classes de Base :

Nous allons voir à présent la couche du Framework nommé « Class Base Library » : le Framework fournit en effet plusieurs classes de base qui vont permettre aux développeurs d'avoir des outils pour réaliser leurs applications.

4.1.Introduction aux classes de base:

Comme nous l'avons vu dans les chapitres précédents, le Framework met à disposition des développeurs une boîte à outils qui est constituée de classes qui sont accessibles par tous les langages .NET. En effet, seule la syntaxe change entre les différents langages .NET, mais les classes restent les mêmes. Il est donc très facile de passer d'un langage .NET à un autre (par exemple du C# au VB.NET).

Les classes du Framework sont organisées selon une structure hiérarchisée. En effet, les classes sont regroupées selon leur utilité dans des namespaces. Cela facilite grandement le travail du développeur, car il peut trouver très facilement les classes dont il a besoin pour réaliser son application.

Par exemple, toutes les classes qui permettent de manipuler des fichiers XML, valider un fichier XML... se trouvent dans le namespace «System.Xml». Les namespaces vont permettre ainsi de regrouper des classes suivant leur utilité et de faciliter ainsi la recherche de la bonne classe.

4.2.Norme du Framework :

Une norme définit comment, au sein d'une application, vont se déclarer les différents types (string, int, bool...), comment vont se nommer les méthodes, les classes... L'application d'une norme au sein d'un grand projet est indispensable, car généralement dans de grands projets, il n'y a pas qu'un seul développeur. Il faut donc que le code soit lisible et compréhensible par tous les développeurs du projet.

Si vous parcourez les classes du Framework, vous pouvez remarquer qu'une norme a été respectée pour nommer les classes. Les classes du Framework respectent la norme suivante :

- La première lettre d'une classe est en majuscule, les autres en minuscules exemple : la classe « Console » du namespace «System».
- Si le nom d'une classe comporte plusieurs mots, chaque mot doit commencer par une majuscule, le reste en minuscules (exemple : la classe « DateTime » du namespace « System »).
- Si le nom d'une classe ne contient que deux lettres, alors ces deux lettres pourront être en majuscule (exemple : « System.IO »).

Un cours va être consacré à la méthodologie et ce cours détaillera notamment la norme .NET. Pour avoir plus de détails sur la norme, nous vous conseillons donc de consulter ce cours et d'aller aussi sur la page MSDN consacrée à la norme :

<http://msdn.microsoft.com/library/en-us/cpgenref/html/cpconnetframeworkdesignguidelines.asp>.

Si vous n'avez pas de norme définie pour votre application, il est conseillé d'utiliser celle du Framework pour vous habituer à vous contraindre à utiliser une norme.

<http://www.labo-dotnet.com>

Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

4.3.Présentation des principaux namespaces :

4.3.1. System :

Le namespace « System » regroupe tout d'abord tous les types de bases : String, Int, Bool... C'est donc pour cela qu'il faut toujours inclure dans vos applications le namespace « System » à moins de vouloir taper le chemin complet de vos types (par exemple « System.String »).

Le namespace « System » contient aussi une classe qui est très importante pour les applications console : c'est la classe « Console ». A partir de cette classe, vous avez toutes les méthodes qui vous permettent d'afficher du texte, de prendre une valeur entrée par un utilisateur...

4.3.2. System.IO :

Le namespace « System.IO » regroupe toutes les classes qui vous permettent de lire des fichiers, écrire et modifier un fichier...

Par exemple, la classe « TextReader » du namespace « System.IO » vous permettra de lire un fichier texte et afficher son contenu.

4.3.3. System.Data :

Le namespace « System.Data » regroupe toutes les classes et types qui vous permettent d'accéder à une base de données.

4.3.4. System.Collections :

Le namespace « System.Collections » regroupe toutes les classes qui vous permettent de manipuler des collections d'objet. Une collection d'objet permet de stocker dans un ensemble logique des objets. A l'instar des tableaux, les collections peuvent être redimensionnées. Vous pourrez gérer vos piles FIFO avec la classe « Queue » et vos piles LIFO avec la classe « Stack ».

4.3.5. System.Net :

Le namespace « System.Net » regroupe toutes les classes utiles à l'accès au réseau : socket, requête http...

4.3.6. System.Reflection :

Le namespace « System.Reflection » regroupe toutes les classes utiles à l'interrogation des métas données. La réflexion permet de connaître toutes les méthodes d'une classe, ainsi que ses paramètres, constructeurs... L'utilitaire « Wincv » qui permet d'effectuer des recherches sur les classes du Framework et d'afficher les informations sur ces classes utilise la réflexion.

4.3.7. System.Xml :

Le namespace « System.Xml » regroupe toutes les classes qui vous permettent de manipuler des fichiers XML et de les modifier.

En effet, vous aurez à disposition plusieurs classes qui vous permettront de parcourir les différents nœuds d'un fichier XML, de lire et de modifier les éléments et ses attributs.

5. ASP.NET & Windows Form :

5.1.ASP.NET :

5.1.1. Présentation de ASP.NET :

ASP.NET permet la création d'application Web au sein de .NET et non pas de pages Web comme avec ASP. En .NET tout est objet, donc en ASP.NET une page est un objet, un bouton est un objet... : ces objets s'appellent des contrôles Web.

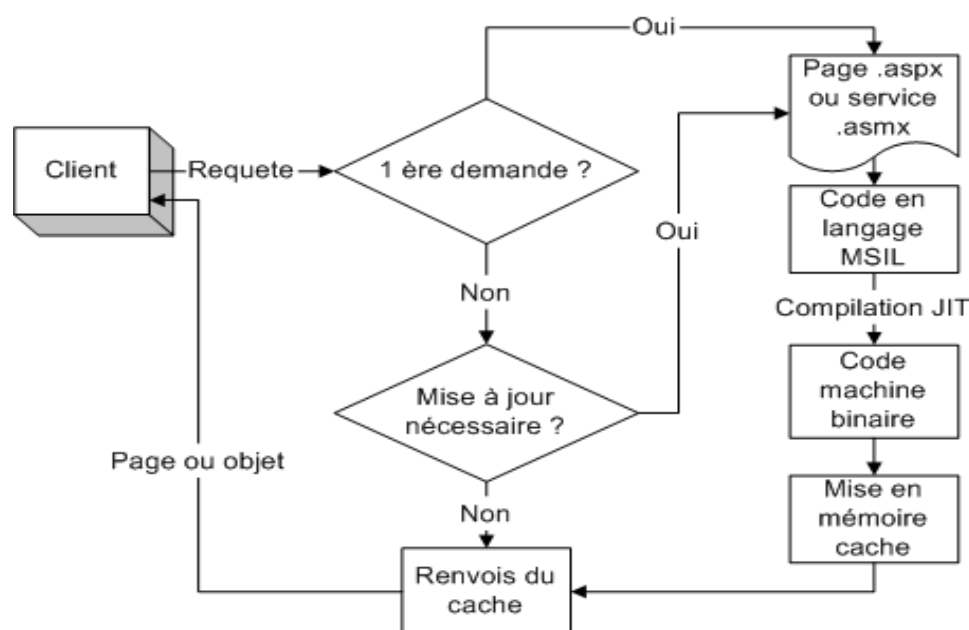
Lorsqu'un développeur voudra insérer un bouton sur son application Web, il devra insérer le contrôle button. C'est ensuite ASP.NET qui se chargera de créer le code HTML correspondant au contrôle.

En ASP, le code et l'interface (HTML) n'étaient pas séparés et se trouvaient donc sur le même fichier : cela pose beaucoup de problème car par exemple le designer d'un site ne connaît pas forcément le langage VB Script. En ASP.NET, l'interface et le code de l'application Web sont séparés ce qui facilite le travail du designer et du développeur.

L'une des grandes forces de ASP.NET est sa portabilité face aux différents clients : en effet, nous avons vu que ASP.NET se chargeait de créer le code HTML correspondant aux différents contrôles (Bouton, Label...). Mais le code HTML généré va dépendre du navigateur clients et donc les applications Web sous .NET pourront être lues sous la plupart des navigateurs Web que ce soit un Internet Explorer, Netscape, Mozilla, Konqueror...

5.1.2. ASP.NET un langage compilé :

Contrairement à ASP qui était un langage interprété, ASP.NET est un langage compilé comme le montre le schéma suivant :



Comme vous pouvez le voir sur ce schéma, le code n'est compilé qu'une seule fois lors de la première demande. Dans le cas contraire, les applications Web ne seraient pas du tout performantes s'il fallait compiler la page à chaque demande du client !

Le processus ASP.NET va vérifier si une mise à jour est nécessaire, c'est-à-dire si il y a eu une modification du code, et compilera donc la page si nécessaire.

5.2.Windows Form :

Windows Form est une infrastructure qui permet de construire des applications clientes Windows .NET, qui utilise la CLR. Les applications Windows .NET ne peuvent donc être déployées que sous un environnement où le Framework a été installé. Les applications Windows Form peuvent être créées dans n'importe quel langage .NET.

Les Windows Forms présentent une multitude d'avantages :

- Windows Form associe la simplicité de programmation (drag and drop de composant) et la puissance de la CLR.
- Windows Form tire profit des fonctionnalités de sécurité de la CLR pour avoir au final une application Windows sécurisée.
- Windows Form prend complètement en charge la connexion rapide et aisée à des services Web XML.
- Windows Form prend totalement en charge les contrôles ActiveX. Vous pouvez aisément héberger des contrôles ActiveX dans une application Windows Form.

6. Portabilité de .NET :

6.1. Standardisation de .NET:

Nous avons vu dans les chapitres précédents que la plateforme .NET était portable car le Framework .NET est en effet disponible sur toutes les versions de Windows à partir de Windows 98 mais aussi sur Linux avec le projet Mono.

Cela a été rendu possible car Microsoft a déposé les spécifications de la plateforme .NET à l'ECMA qui est un organisme de standardisation. Une partie des spécifications de la CLR et des classes de base du Framework ainsi que le langage C# sont donc disponibles sur le site de la ECMA ou sur le site de Microsoft :

<http://msdn.microsoft.com/net/ecma>

Ce n'est pas Microsoft qui va porter le Framework sur d'autres systèmes d'exploitation tels que Linux mais des organismes indépendants voire une communauté de développeurs comme pour le projet Mono qui est l'implémentation libre du Framework sous Linux.

Les spécifications soumises à la ECMA vont donc permettre aux développeurs de développer leur propre Framework voire leur propre compilateur C# sur leur architecture préférée.

6.2. CLS :

CLS signifie Common Language Specification. La CLS va être en fait la spécification commune à tous les langages .NET qui va permettre la compatibilité entre tous les langages .NET. La totalité des langages .NET respectent cette spécification : par exemple le C++ Managé, c'est à dire le C++ .NET qui utilise les classes du Framework et qui est managé par la CLR, a dû être épuré de certains concepts tel que l'héritage multiple car les spécifications de la CLS ne supportent pas l'héritage multiple.

La CLS est donc le plus petit dénominateur commun entre tous les langages .NET et de par ses spécifications, la CLS exclu tous langages procéduraux, c'est-à-dire non objet, tel que le C.

7. .NET Server :

7.1.Présentation .NET Server :

Windows .NET Server est la nouvelle génération de Serveurs Microsoft qui va succéder aux Windows 2000 Server.

Ce qui change radicalement avec .NET Server comparé aux anciennes versions des serveurs Windows est la stratégie de sécurité. En effet, Microsoft s'est fait beaucoup critiquer pour la sécurité de ses serveurs, notamment à cause de ses failles de sécurité.

Microsoft a répondu à ce défaut en fermant tout les services sur .NET Server : par défaut dorénavant, tout sera protégé. Avant, c'était exactement l'inverse : par défaut, tout était ouvert, et il fallait que les développeurs et administrateurs réseaux puissent protéger tous les accès. Maintenant, comme tout sera protégé par défaut, les applications seront beaucoup plus sécurisées : il faut autoriser les applications à avoir accès à certaines ressources.

7.2.Versions de .NET Server :

Il va y avoir 4 versions de .NET Server :

- Windows .NET Standard Server : pratiquement identique à Windows 2000 Server (2 processeurs et 4 Go de RAM). Elle est destinée à gérer les services élémentaires dans des environnements restreints.
- Windows .NET Enterprise Server : cette version succède à Windows 2000 Advanced Server (8 processeurs, 32 Go de RAM, et permet de relier 4 nœuds en clustering). Ce système d'exploitation est destiné aux environnements importants, et est prévu pour gérer des services d'applications exigeants, comme SQL Server ou Exchange.
- Windows .NET Datacenter Server : cette version succède à Windows 2000 Datacenter Server (32 processeurs, 8 processeurs minimum, 64 Go de RAM, 8 nœuds en clustering, système natif d'équilibrage de charge).
- Windows .NET Web Server : cette version est nouvelle. Il s'agit d'une version simplifiée, conçue pour faire tourner une seule application : IIS 6 (2 processeurs, 2 Go de RAM).

7.3.Mise à jour de .NET Server :

Il y a en effet plusieurs mises à jour qui ont été faites dans les .NET Servers :

- IIS 6, qui est une nouvelle version, sera installée par défaut, mais sans aucune option. Cela oblige l'administrateur à ajouter des fonctionnalités seulement si elles sont nécessaires (comme les ASP, les scripts d'exemple, ... qui pouvaient représenter des failles de sécurité et qui étaient installées par défaut).
- Active Directory : beaucoup de mises à jour sur Active Directory Services, comme la possibilité de renommer des domaines, la réplication des groupes, ... Par contre, ces modifications rendent les contrôleurs de domaine .NET incompatibles avec les contrôleurs Windows 2000.
- Interface Luna : nouvelle interface graphique, comme dans Windows XP. Par contre, ce n'est pas forcément intéressant car cela consomme des ressources inutiles sur un serveur !

- Activation du produit : la nouvelle politique de Microsoft est utilisée, et il faudra activer la version par Internet ou par téléphone.