C# et .NET
Version 2

Gérard Leblanc

ISBN : 2-212-11778-7

EYROLLES

Introduction à l'architecture .NET

Le concepteur et responsable du projet

Avant même d'expliquer dans les grandes lignes ce que sont l'architecture .NET (les Américains prononcent *dot net*) et le nouveau langage C# (prononcer « C sharp », même chez nous) de Microsoft, il faut parler de son concepteur et principal architecte chez Microsoft. Son nom, Anders Hejlsberg, ne vous dit sans doute rien. Et pourtant…

Anders Hejlsberg est né au Danemark en 1961. En 1983, il rencontre le Français Philippe Kahn, établi en Californie, et lui présente la première version d'un logiciel qu'il est en train d'écrire. Il s'agit d'un logiciel de développement de programmes, fondé sur le langage Pascal, d'une convivialité et d'une puissance inconnues à l'époque. Le résultat de cette rencontre est une success story qui a marqué le monde des outils de développement : celle de la société Borland et de son produit phare, Turbo Pascal. Dans sa version Turbo Pascal, le langage Pascal est en effet considérablement dopé par rapport à sa version d'origine, dont le succès était jusque-là limité aux milieux académiques. Avec ce produit, Anders Hejlsberg montrait déjà son souci de fournir des outils répondant aux besoins et attentes des développeurs.

Au début des années 1990, Anders Hejlsberg et Borland réitèrent le succès de Turbo Pascal avec Delphi, également fondé sur le langage Pascal, qui bouleverse cette fois la manière de développer des programmes Windows. À quelques années de distance (le début de l'ère DOS pour Turbo Pascal et le début de l'ère Windows pour Delphi), Anders Hejlsberg devait donc concevoir sous la bannière Borland des produits qui ont suscité admiration et respect chez les développeurs soucieux à la fois de convivialité et d'efficacité. Jusqu'alors, ces derniers étaient résignés à des outils peu puissants, peu performants ou alors totalement dépourvus de convivialité. Anders Hejlsberg prouvait que l'on pouvait allier puissance, efficacité, élégance et convivialité. Ses ajouts au langage Pascal devaient être massivement adoptés par les développeurs, au point d'en faire, dans la pratique, une norme de fait du langage.

Avec Turbo Pascal et Delphi, les développeurs trouvaient en Hejlsberg un pair, à l'écoute de leurs problèmes et soucieux d'apporter des solutions concrètes. Avec Delphi, les développeurs découvraient et pratiquaient le recours étendu aux composants, faisant de Delphi une véritable boîte à outils de composants logiciels. Certes, les théoriciens de la programmation orientée objet prônaient depuis quelque temps (mais sans proposer quoi que ce soit à réutiliser) la réutilisabilité et le développement à partir de briques logicielles. Anders Hejlsberg eut l'art de mettre ces idées en pratique, et cela sans faire de déclarations fracassantes (et même plutôt en toute discrétion).

Le nom d'Anders Hejlsberg restait peu connu en dehors de Borland et d'une couche périphérique. Les utilisateurs de Delphi pouvaient néanmoins le découvrir à condition de connaître la manière d'afficher l'œuf de Pâques (c'est-à-dire la commande cachée) de Delphi : `Aide` → `A propos`, maintenir la touche `ALT` enfoncée et taper, selon la version de Delphi : `AND` ou `TEAM` ou `DEVELOPERS` ou encore `VERSION`. Une photo d'un Anders hilare apparaissait même dans l'œuf de Pâques de l'une des versions de Delphi.

Figure 0-2

Figure 0-3

En octobre 1996, Microsoft, à la traîne pour ce genre d'outils, débauche Anders Hejlsberg avec des conditions presque dignes d'une vedette du sport. Dans la foulée, Microsoft débauche une trentaine d'autres développeurs de Borland, ce qui est énorme quand on sait que la conception et la réalisation de tels outils mobilisent rarement une foule de

développeurs, mais au contraire une poignée d'informaticiens compétents, efficaces et motivés.

Chez Microsoft, Anders conçoit d'abord WFC (*Windows Foundation Classes*), c'est-à-dire les classes Java pour interface Windows. Le but était de permettre aux programmeurs en Visual J++ (la version Microsoft du compilateur Java) de développer des applications professionnelles dignes de ce nom. En effet, à l'époque, n'étaient disponibles pour le développement Windows en Java que les classes AWT (*Abstract Window Toolkit*) de Sun, des classes qui ne pouvaient satisfaire que des développeurs vraiment peu exigeants (classes d'ailleurs aujourd'hui largement délaissées au profit de Swing).

Les relations entre Anders Hejlsberg et la communauté « officielle » de Java devaient vite s'envenimer, car les classes WFC, bien que nettement plus professionnelles que ce qui était à l'époque disponible en provenance de Sun, étaient propres à Windows et incompatibles avec les autres systèmes, donc non conformes à la philosophie Java.

Dire que James Gosling, le concepteur de Java, n'apprécie guère Anders Hejlsberg relève de l'euphémisme. Lors de la conférence Java One de San Francisco en 1998, Gosling commente d'ailleurs WFC en ces termes, égratignant au passage Anders Hejlsberg : *something bizarre from Mr Method Pointers*, pour ne reprendre que la moins assassine de ses phrases, largement reprises par la presse spécialisée de l'époque.

Toutes ces querelles et attaques personnelles présentent d'autant moins d'intérêt que Sun et Microsoft roulent désormais sur des voies où le croisement est tout simplement évité : en juin 2000, Microsoft annonce, en même temps que la disparition de Visual J++ de sa gamme de produits, l'architecture .NET et le langage C# dont Anders Hejlsberg est le principal concepteur. Un an plus tard, Visual J++ fait néanmoins sa réapparition (sous le nom de Visual J#) sans toutefois attirer les projecteurs, quasiment dans l'indifférence.

Ce que .NET change

Pour la version 7 (2002) de l'outil de développement Visual Studio (version rebaptisée Visual Studio .NET), Microsoft a conçu un système qui rend le développement d'applications Windows et Web bien plus aisé. Une nouvelle architecture a été mise au point, des langages ont été modifiés et un nouveau langage créé, le C# qui devient le langage de référence et principal langage pour Microsoft. Le C++ est certes encore présent, mais rarissimes sont aujourd'hui les fragments et exemples de code en C++ dans la documentation de Microsoft, les articles ou ouvrages consacrés à .NET. Le fait que les applications Web doivent être impérativement écrites en C#, J# ou VB.NET est encore plus significatif. Des efforts considérables ont également été déployés pour faire de Visual Basic un langage de première catégorie. Visual Basic, rebaptisé VB.NET, devient un langage orienté objet, au même titre que le C# et se démarque ainsi nettement de la version 6 de Visual Basic (plus aucun nouveau développement pour ce produit et fin du support annoncée).

.NET constitue-t-il une révolution dans la manière de concevoir et d'utiliser les programmes ? La réponse est incontestablement affirmative pour la manière de concevoir

et d'écrire des programmes : les programmeurs C++ habitués au développement à la dure avec MFC découvriront, surtout s'ils passent au C# (un jeu d'enfant pour eux), la facilité de Delphi et du véritable développement à partir de briques logicielles. Les programmeurs en Visual Basic découvriront un environnement de développement entièrement orienté objet, comparable à l'orientation objet du C++ et du C#. Dans tous les cas, le développement d'applications Web et surtout de services Web s'avère bien plus facile. Écrire une application Web devient en effet presque aussi simple que l'écriture d'une application Windows.

L'autre révolution est l'importance accordée aux services Web. Visual Studio rend d'ailleurs leur implémentation d'une facilité déconcertante, ce qui favorise l'adoption de cette technologie par les développeurs. En gros, un service Web est une fonction qui s'exécute quelque part sur Internet, comme s'il s'agissait d'une fonction exécutée localement. Résultat : on simplifie le développement du programme en déportant des fonctionnalités sur des sites spécialisés fournissant tel ou tel service. De plus, ces services Web sont (grâce aux protocoles HTTP et SOAP (*Simple Object Access Protocol*), largement adoptés) indépendants du langage et même de la plate-forme.

.NET implique-t-il des changements quant à la manière d'utiliser les applications ? La réponse est, à la fois, oui et non. Visual Studio permet en effet d'écrire les applications Windows les plus traditionnelles, s'exécutant sur des machines dédiées et même non connectées à un réseau local ou à Internet. Mais Visual Studio permet aussi d'écrire, avec la même simplicité et les mêmes techniques, des applications pour appareils mobiles (appareils divers sous Windows CE, Pocket PC et smartphones), ainsi que des services Web. Cette mutation au profit des services Web implique une connexion performante et permanente au réseau Internet, ce qui n'est pas encore le cas aujourd'hui pour tous les utilisateurs. Mais nul doute que cela deviendra vite réalité, au même titre que la connexion au réseau électrique. Les serveurs Internet ne se contenteront plus de fournir des pages HTML statiques ou dynamiques : ils fourniront surtout des services Web aux applications. Ces serveurs Internet serviront de plus en plus, grâce à la technologie ClickOnce, à déployer et mettre à jour des applications Windows.

.NET est-il significatif d'un changement d'attitude chez Microsoft ? Cette société a souvent été blâmée pour son manque de respect des normes mais aussi pour des incompatibilités qui portent gravement préjudice à ses concurrents. On comprend et réagit vite chez Microsoft : C# et le *run-time* (sous le nom de CLI, pour *Common Language Infrastructure*) font l'objet d'une normalisation Ecma et maintenant ISO. De plus, avec les services Web, l'accent est mis sur l'interopérabilité entre plates-formes. Enfin, mais indépendamment de Microsoft, une implémentation tout à fait remarquable (avec néanmoins des incompatibilités en ce qui concerne les programmes Windows) de .NET existe sur Linux avec le « projet mono » (voir `www.go-mono.com`) qui permet d'y exécuter des EXE .NET, sans même devoir recompiler le programme.

Les utilisateurs courent-ils un risque avec .NET ? Microsoft mise tout sur lui : plus aucun développement ni aucune communication (Web, articles, séminaires et grand-messes comme TechEd ou PDC (*Professional Developers Conference*) en dehors de .NET. Microsoft, dont on connaît le savoir-faire et les moyens, est condamné au succès.

L'architecture .NET

L'architecture .NET (nom choisi pour montrer l'importance accordée au réseau, amené à participer de plus en plus au fonctionnement des applications grâce aux services Web), technologie appelée à ses balbutiements NGWS (*Next Generation Web Services*), consiste en une couche Windows, en fait une collection de DLL librement distribuable et qui sera incorporée dans le noyau des prochaines versions de Windows (Windows Vista).

Cette couche contient un nombre impressionnant (plus de deux mille) de classes (tous les langages de .NET doivent être orientés objet), ainsi que tout un environnement d'exécution (un *run-time*, ou couche logicielle si vous préférez) pour les programmes s'exécutant sous contrôle de l'environnement .NET. On appelle cela le mode géré ou managé (*managed code*). La notion de *run-time* n'a rien de nouveau : les programmeurs en Visual Basic la connaissent depuis longtemps puisque même les programmes VB compilés en ont besoin. Les programmeurs Java connaissent aussi la notion de machine virtuelle. Néanmoins, même si le *run-time* .NET est, dans les faits, une machine virtuelle, Microsoft a toujours soigneusement évité d'employer ce terme, sans doute trop lié à Java et à Sun… Un *run-time* fournit des services aux programmes qui s'exécutent sous son contrôle. Dans le cas de l'architecture .NET, ces services font partie de ce que l'on appelle le CLR (*Common Language Run-time*) et assurent :

- le chargement (*load*) et l'exécution contrôlée des programmes ;
- l'isolation des programmes les uns par rapport aux autres ;
- les vérifications de type lors des appels de fonctions (avec refus de transtypages hasardeux) ;
- la conversion de code intermédiaire en code natif lors de l'exécution des programmes, opération appelée JIT (*Just In Time Compiler*) ;
- l'accès aux métadonnées (informations sur le code qui font partie du code même) ;
- les vérifications lors des accès à la mémoire (pas d'accès possible en dehors de la zone allouée au programme) ainsi qu'aux tableaux (pas d'accès en dehors de ses bornes) ;
- la gestion de la mémoire, avec ramasse-miettes automatique ;
- la gestion des exceptions ;
- la sécurité ;
- l'adaptation automatique des programmes aux caractéristiques nationales (langue, représentation des nombres et des symboles, etc.) ;
- la compatibilité avec les DLL et les modules COM actuels qui s'exécutent en code natif non contrôlé par .NET.

Les classes .NET peuvent être utilisées par tous les langages prenant en charge l'architecture .NET. Ces classes sont regroupées dans des espaces de noms (*namespaces)* qui se

présentent en quelque sorte comme des répertoires de classes. Quelques espaces de noms et quelques classes :

Les classes de l'architecture .NET		
Espace de noms	**Description**	**Exemples de classes**

`System`	Accès aux types de base.
	Accès à la console.
		`Int32, Int64, Int16`
		`Byte, Char`
		`String`
		`Float, Double, Decimal`
		`Console`
		`Type`

`System.Collections`	Collections d'objets.	`ArrayList, Hashtable, Queue,` `Stack, SortedList`

`System.IO`	Accès aux fichiers.	`File, Directory,`
		`Stream, FileStream,`
		`BinaryReader, BinaryWriter`
		`TextReader, TextWriter`

`System.Data.Common`	Accès ADO.NET aux bases de données.	`DbConnection, DbCommand, DataSet`

`System.Net`	Accès au réseau.	`Sockets`
		`TcpClient, TcpListener`
		`UdpClient`

`System.Reflection`	Accès aux métadonnées.	`FieldInfo, MemberInfo,` `ParameterInfo`
`System.Security`	Contrôle de la sécurité.	`Permissions, Policy` `Cryptography`
`System.WinForms`	Composants orientés Windows.	`Form, Button, ListBox` `MainMenu, StatusBar, DataGrid`
`System.Web.UI.WebControls`	Composants orientés Windows.	`Button, ListBox, HyperLink` `DataGrid`

Il y a compatibilité absolue entre tous les langages de l'architecture .NET :

- une classe .NET peut être utilisée de manière identique (à la syntaxe du langage près) dans tout langage générant du code .NET ;
- une classe peut être créée dans un premier langage, servir de classe de base pour une classe dérivée implémentée dans un deuxième langage, et cette dernière classe enfin instanciée dans un troisième langage ;
- la manière de créer et d'utiliser les objets est identique (évidemment aux détails de langage près).

Les services de .NET créent les objets (tout est objet dans l'architecture .NET) et se chargent de libérer automatiquement de la mémoire les objets qui ne peuvent plus être utilisés : technique du ramasse-miettes (*garbage collection*). On retrouvait déjà ce procédé dans le

langage Smalltalk créé par le Parc (Palo Alto Research Center) de Xerox, à l'origine des interfaces graphiques, de la souris et de bien d'autres choses, notamment dans le domaine de l'orienté objet. Java a d'ailleurs largement emprunté de nombreux concepts à Smalltalk. Il n'y a pas de honte à reconnaître que l'architecture .NET et le langage C# en particulier, reprennent le meilleur du C++, de Delphi, de Java, de Visual Basic et de Smalltalk. C'est à ce dernier langage, pourtant le moins utilisé des cinq, que doit aller prioritairement la reconnaissance des développeurs.

Les composants .NET utilisent la technique de la clé privée, connue du seul développeur, et de la clé publique, à disposition des utilisateurs. Il faut une concordance du code, de la clé publique et de la clé privée pour pouvoir exécuter le code d'un composant. Toute modification du code d'un composant (opération effectuée malicieusement par les virus) rend ce composant inutilisable.

.NET utilise aussi une technique qui met fin au problème connu sous le nom de l'enfer des DLL (problème créé lors de l'installation d'un logiciel, par écrasement d'une DLL existante) en attribuant des numéros aux versions. Il est maintenant possible de garder plusieurs versions d'une même DLL, les programmes utilisant automatiquement la version de la DLL qui leur convient.

Les langages de l'architecture .NET

Tous les langages .NET doivent présenter des caractéristiques communes :

- mêmes types de données (tailles et représentation), ce que l'on appelle le CTS (*Common Type System*) définissant précisément ces caractéristiques ;
- même utilisation des classes, même manière de créer et de gérer les objets ;
- même code intermédiaire : MSIL généré (*Microsoft Intermediate Language*).

Les compilateurs créant des programmes pour .NET doivent générer un code intermédiaire, appelé MSIL. Il s'agit d'un code intermédiaire entre le code source (par exemple du code C# ou Visual Basic) et le code natif directement exécutable par le microprocesseur. Ce code intermédiaire est donc indépendant du code de bas niveau qu'est le langage machine, mais il est capable de manipuler ces constructions de haut niveau que sont les objets. C'est ce qui explique pourquoi .NET a pu être porté sur d'autres plates-formes comme Linux (voir le projet mono qui est disponible et librement téléchargeable).

Au moment d'exécuter un programme, ce code intermédiaire est pris en charge par .NET qui le fait exécuter, fonction après fonction, par un *JIT compiler*. .NET procède par compilation et non par interprétation, toutefois il s'agit d'une compilation (de code MSIL en code natif) en cours d'exécution de programme. Au moment de commencer l'exécution d'une fonction, mais lors du premier appel seulement, .NET appelle le JIT. Le JIT, qui connaît alors l'environnement d'exécution, et notamment le type de microprocesseur, compile le code intermédiaire de la fonction en code natif, en fonction du microprocesseur réellement utilisé (ce qui permet une optimisation). Du code natif est dès lors exécuté. Le processus recommence quand une autre fonction, non encore compilée,

est appelée. Très rapidement, on n'exécute plus que du code natif optimisé pour le microprocesseur de l'utilisateur. Il est aussi possible, avec l'utilitaire `ngen`, de générer des programmes précompilés (et, par conséquent, propres à un microprocesseur bien particulier).

Les compilateurs fournis par Microsoft avec Visual Studio sont :

- Visual Basic qui a été profondément modifié et qui est devenu entièrement orienté objet ;
- Visual C++ qui peut travailler dans deux modes :
  - dans le premier mode, compatible avec les versions précédentes, le code généré est le code natif du microprocesseur et les classes sont les classes MFC. Ce mode n'est donc pas compatible .NET et n'en utilise pas les possibilités, ni pour le développement ni pour l'exécution. Ce mode est là pour assurer la compatibilité avec l'existant ;
  - le second mode (*managed code*) vise l'architecture .NET et utilise les ressources du nouvel environnement de développement. Développer en C++ pour .NET reste néanmoins lourd, surtout quand on connaît la facilité offerte par C#, le nouveau langage introduit par Microsoft ;
- C#, qui devient de facto le langage de référence et qui fait l'objet de cet ouvrage ;
- enfin, J#, le langage Java de .NET dont on doute qu'il soit largement utilisé à voir le peu d'activité dans les forums consacrés à ce langage.

Les langages .NET ne se limitent cependant pas à ceux-là. Microsoft publie toute la documentation nécessaire pour permettre à d'autres fournisseurs de compilateurs de livrer des versions .NET de leur produit : Eiffel, Pascal, Perl, Cobol, Python, Oberon, Scheme et Smalltalk pour n'en citer que quelques-uns. Tous ces langages adaptés à .NET (sauf exception, ils ont été adaptés au niveau du code généré, pas de la syntaxe) utilisent les mêmes classes et les mêmes outils de développement et leur intégration dans Visual Studio est saisissante. Les développeurs de ces langages n'ont pas à créer les librairies et outils nécessaires (par exemple le debugger) pour une véritable utilisation professionnelle. Or, développer ces librairies et outils prend généralement beaucoup plus de temps et mobilise plus de ressources humaines que le développement du compilateur lui-même, ce qui constitue un frein à l'apparition de ces nouveaux langages (hors des circuits académiques évidemment).

Avec l'architecture .NET, Microsoft joue incontestablement la carte de l'ouverture aux langages, y compris ceux provenant de sources extérieures à la société puisque l'accent est mis sur la variété des langages et les possibilités d'inter-langage : le programmeur a le choix, à tout moment, du langage qu'il connaît le mieux ou du langage le plus approprié à certaines parties de l'application. On peut en effet très bien envisager une partie de l'application écrite dans un langage et une autre partie dans un langage différent. La pierre lancée dans le jardin de Sun, qui prône Java comme unique langage, n'échappe à personne, d'autant moins que Microsoft a fait du C# un langage normalisé par un comité indépendant de Microsoft (Ecma/ISO) alors que Java reste un langage propriétaire de Sun.

Tous ces langages doivent évidemment suivre les règles édictées pour être compatibles .NET. Ces règles forment le CLS (*Common Language Specification*). Signalons quand même qu'en générant un code susceptible d'être utilisé ou réutilisé (fichier EXE ou DLL, ce que l'on appelle un *assembly* dans le jargon .NET), le compilateur doit générer pour cette pièce de code (on parle d'ailleurs de composants auto-descriptibles, *self describing components*) :

- des informations sur le contenu de cette pièce de code (classes, propriétés et méthodes publiques) ainsi que sur les librairies utilisées par le code de l'*assembly* (nom des librairies nécessaires, numéro de version minimale, etc.) ;
- le code des fonctions des classes (code MSIL).

Tout cela donne à .NET un contrôle bien plus élaboré des programmes et notamment des librairies extérieures qui, à un moment ou à un autre, sont appelées par le programme.

Un programme pour .NET démarre comme un programme traditionnel mais le contrôle passe immédiatement au *run-time* .NET qui fait exécuter le programme sous sa haute surveillance (compilation du code MSIL, fonction après fonction, par le JIT, appels aux services .NET, vérifications, etc.).

Tout programme écrit pour .NET a accès à l'API Windows (l'ensemble des fonctions de base de Windows) cependant, comme des classes .NET encapsulent ces fonctions, il est exceptionnel (mais possible) de faire appel à ces fonctions de base de Windows. Les programmes .NET peuvent utiliser des composants COM. Sous Visual Studio, ces composants COM s'utilisent aussi aisément que dans l'environnement VB6. Enfin, des composants .NET peuvent être transformés en composants COM et utilisés comme tels dans VB6 mais aussi dans tous les outils de développement qui reconnaissent la technologie COM.

Le langage C#

Le langage star de la nouvelle version de Visual Studio et de l'architecture .NET est C#, un langage dérivé du C++. Il reprend certaines caractéristiques des langages apparus ces dernières années et en particulier de Java (qui reprenait déjà à son compte des concepts introduits par Smalltalk quinze ans plus tôt). C# peut être utilisé pour créer, avec une facilité incomparable, des applications Windows et Web. C# devient le langage de prédilection d'ASP.NET qui permet de créer des pages Web dynamiques avec programmation côté serveur.

C# s'inscrit parfaitement dans la lignée C → C++ → C# :

- le langage C++ a ajouté les techniques de programmation orientée objet au langage C (mais la réutilisabilité promise par C++ ne l'a jamais été qu'au niveau source) ;
- le langage C# ajoute au C++ les techniques de construction de programmes sur base de composants prêts à l'emploi avec propriétés et événements, rendant ainsi le

développement de programmes nettement plus aisé. La notion de briques logicielles aisément réutilisables devient réalité.

Passons rapidement en revue les caractéristiques de C# (par rapport au C++) :

- orientation objet prononcée : tout doit être incorporé dans des classes ;
- types précisément conformes à l'architecture .NET et vérifications de type plus élaborées ;
- Unicode pour le code des caractères : les programmeurs C++ qui connaissent la lourdeur de l'interfaçage avec des modules Unicode (difficile aujourd'hui d'imaginer autre chose) apprécieront ;
- libération automatique des objets (*garbage collection*) ;
- les pointeurs ne disparaissent pas mais ne sont plus réservés qu'à des cas bien particuliers d'optimisation (voir par exemple la fin du chapitre 13 consacrée au traitement d'images, l'utilisation des pointeurs permet d'améliorer les performances de manière pour le moins spectaculaire) ;
- remplacement des pointeurs (sur tableaux, sur objets, etc.) par des références qui offrent des possibilités semblables mais avec bien plus de sûreté mais sans perte de performance ;
- disparition du passage d'argument par adresse au profit du passage par référence ;
- manipulation des tableaux de manière fort différente et avec plus de sécurité, ce qui est particulièrement heureux ;
- passage de tableaux en arguments ainsi que renvoi de tableau nettement simplifié ;
- nouvelle manière d'écrire des boucles avec l'instruction `foreach` qui permet de balayer aisément tableaux et collections ;
- introduction des propriétés, des indexeurs et des attributs ;
- disparition de l'héritage multiple mais possibilité pour une classe d'implémenter plusieurs interfaces.

Tout cela est étudié dans les premiers chapitres de l'ouvrage.

Créer des applications Windows et Web

Écrire une application Windows ou Web en C# avec le nouvel environnement Visual Studio devient incomparablement plus simple qu'avec MFC puisque Visual Studio utilise le modèle de composants ainsi que le développement interactif qu'ont pu apprécier les développeurs sous Delphi. Il faut même s'attendre à ce que ce développement s'avère de plus en plus simple au fur et à mesure que des composants deviennent disponibles sur le marché. Comme ces composants .NET sont susceptibles d'être utilisés dans tous les langages de l'architecture .NET, leur marché est considérable. Nul doute que l'offre sera dès lors tout aussi considérable.

Avec Visual Studio, les applications Web se développent comme les applications Windows, en utilisant le modèle ASP.NET, par insertion de code (C#, J# ou VB.NET) dans des pages HTML. Le code C# ou VB.NET est compilé sur le serveur au moment de traiter la page (mais il est maintenant possible de précompiler le code) et du code HTML pur est envoyé au client. N'importe quel navigateur peut dès lors être utilisé sur la machine du client.

Côté accès aux bases de données, toute la couche ADO (*ActiveX Data Objects*, utilisée en Visual Basic 6) a été entièrement repensée et réécrite pour devenir ADO.NET. Malgré des ressemblances, ADO.NET se démarque nettement d'ADO et le A (pour ActiveX) d'ADO.NET perdra à terme toute signification. XML prend un rôle fondamental dans le monde ADO.NET en général et .NET en particulier.

Tout cela sera abordé avec de nombreux exemples pratiques dans les différents chapitres de cet ouvrage.

Pour résumer

L'architecture .NET pourrait être résumée à l'aide de ce schéma :

<table>
<tr><td>C#</td><td>C++</td><td>VB</td><td>.....</td></tr>
<tr><td colspan="4">COMMON LANGUAGE SPECIFICATION</td></tr>
<tr><td colspan="2">Formulaires Windows</td><td colspan="2">ASP.NET
Formulaires Web
Services Web</td></tr>
<tr><td colspan="2">ADO.NET</td><td colspan="2">XML</td></tr>
<tr><td colspan="4">Librairie des classes de base</td></tr>
<tr><td colspan="4">COMMON LANGUAGE RUN-TIME</td></tr>
<tr><td colspan="2">Windows de base</td><td colspan="2">.....</td></tr>
</table>

Commentons ce schéma en commençant par la couche supérieure, la plus proche des développeurs.

Les développeurs utilisent un ou plusieurs des langages compatibles .NET. Ceux-ci proviennent de Microsoft ou de sources extérieures. À ce jour, on dénombre plus de vingt compilateurs de langages différents disponibles ou prêts à l'être. Le développeur utilise donc le langage qu'il maîtrise le mieux ou qui est le plus approprié à la tâche qu'il entreprend. Rien n'empêche que les différentes parties d'un projet soient écrites dans des langages différents. Une classe de base peut être écrite dans un premier langage, sa classe dérivée dans un deuxième et cette classe dérivée utilisée dans un programme écrit dans un troisième langage.

Tous ces langages doivent avoir une orientation objet et respecter des règles établies par .NET en ce qui concerne les types utilisés, le contenu et le format des fichiers générés par les compilateurs. Ces règles forment le CLS (*Common Language Specification*). Elles sont normalisées au niveau mondial, parfaitement documentées et toute modification doit

être approuvée par le comité de normalisation, Microsoft n'étant que l'un des membres de ce comité.

Tous ces compilateurs s'intègrent parfaitement dans l'outil de développement de Microsoft qui reste le même quel que soit le langage utilisé. Tous ces langages utilisent les mêmes outils et les mêmes classes de base de l'architecture .NET. Pour un programmeur, passer d'un langage à l'autre devient donc nettement plus simple que par le passé puisqu'il garde tous ses acquis et toutes ses compétences.

Les programmes écrits dans ces différents langages offrent, globalement, le même niveau de performance.

Bien que tous ces langages soient égaux, ils gardent néanmoins leurs spécificités, leurs points forts et leurs limites : certains langages (c'est le cas de C#) offrent des possibilités, par exemple accès aux pointeurs, redéfinition d'opérateurs, etc., que d'autres ignorent.

Ces langages permettent d'écrire soit des programmes pour le mode console, des applications pour interface graphique Windows soit des applications pour le Web, côté serveur (le Web pouvant être limité à un réseau local). Pour ces dernières, on utilise les outils d'ASP.NET. Les outils de création de programmes Windows et ceux de création de programmes ASP.NET, quoique différents, sont particulièrement proches dans leurs possibilités et leur utilisation. Tous deux utilisent les mêmes langages (ASP.NET étant néanmoins limité à C#, J# et VB.NET) et les mêmes classes de base de l'architecture .NET.

L'outil de développement permet aussi, avec une facilité déconcertante, de créer et d'utiliser des services Web. Cette technique permet d'appeler des fonctions (au sens de la programmation) qui s'exécutent sur une autre machine du réseau local ou d'Internet. Cette technologie des services Web repose sur des protocoles standard que sont SOAP, XML et HTTP. Ces services Web peuvent être mis à disposition de n'importe quel client, quelle que soit sa machine ou le système d'exploitation qu'il utilise. Ainsi, le service Web, lorsqu'il est utilisé à partir d'un programme .NET, peut provenir de n'importe quelle machine sous contrôle de n'importe quel système d'exploitation.

Toutes ces applications, qu'elles soient de type console, Windows ou Web, manipulent vraisemblablement des données provenant de bases de données. .NET a développé ADO.NET qui a donné accès, de manière quasi transparente, aux différentes bases de données commercialisées.

Toutes ces applications, quel que soit leur type et quel que soit leur langage d'implémentation, ont accès aux mêmes classes de base de l'architecture .NET. Microsoft fournit un nombre impressionnant de classes de base (d'une qualité remarquable d'ailleurs) mais des classes et des composants provenant de sources extérieures peuvent être ajoutés.

Tous les compilateurs de langages génèrent un code intermédiaire appelé IL. Celui-ci est indépendant du microprocesseur. Au moment d'exécuter une fonction du programme, le code IL du programme est compilé fonction par fonction, mais cette compilation n'est effectuée qu'une seule fois, lors de la première exécution de la fonction. Le code IL est compilé en code natif optimisé pour le microprocesseur de la machine.

Les programmes s'exécutent alors sous contrôle strict du *run-time* .NET, avec bien plus de vérifications qu'auparavant. Ce mode de fonctionnement s'appelle le « mode managé ».

Les composants COM (ActiveX) ainsi que le code en DLL d'avant .NET peuvent néanmoins encore être exécutés dans un programme .NET.

Le *run-time* .NET (qu'on appelle aussi le *framework*) doit avoir été installé pour pouvoir exécuter des programmes .NET. Il est librement téléchargeable et généralement préinstallé sur les ordinateurs vendus aujourd'hui.

Le CLR et le C# font l'objet d'une normalisation internationale, ce qui devrait permettre de l'implémenter sous d'autres systèmes d'exploitation. Des implémentations de .NET existent d'ailleurs sous Unix/Linux, ce qui explique les points de suspension à la dernière ligne du schéma.

Le CLR agit comme couche logicielle au-dessus du système d'exploitation et en exploite donc les possibilités. À ce titre, le CLR agit comme machine virtuelle, bien que Microsoft rejette ce terme, certes parce qu'il est galvaudé, au point de signifier peu de choses mais aussi parce qu'il est très associé à la machine virtuelle Java…

C# et .NET en version 2

En novembre 2005, Microsoft a livré la version 2 de .NET et de C#, la nouvelle version de Visual Studio s'appelant Visual Studio 2005.

Cette nouvelle version constitue une version majeure, avec d'importantes améliorations à tous niveaux. Pour n'en reprendre que quelques-unes, parmi les principales :

- en C# : les génériques, le type nullable, les classes partielles, les méthodes anonymes ;
- pour la création de programmes : le refactoring, les extraits de code (*code snippets*) et améliorations dans le débogueur ;
- pour les applications Windows : plusieurs nouveaux composants, notamment pour les barres d'outils et le menu ;
- pour ASP.NET : des améliorations partout (notamment les pages maîtres, la sécurité et les composants orientés données) ;
- pour les bases de données : programmation générique, indépendante du SGBD ;
- pour les mobiles : prise en charge du .NET Compact Framework et nouveaux émulateurs ;
- pour le déploiement : la technologie ClickOnce qui bouleverse la manière de déployer des applications Windows.

Visual Studio est décliné en plusieurs versions, qui présentent des fonctionnalités très semblables mais à des niveaux différents d'utilisation :

- Visual C# Express s'adresse aux hobbyistes, curieux et étudiants : prix dérisoire (et même en téléchargement gratuit pour une durée limitée) pour un apprentissage de la

programmation en mode console et Windows (mais pas des applications pour mobiles) ;

- Visual Web Developer, qui fait également partie de la gamme des produits Express mais pour le développement d'applications Web ;
- SQL Server Express : base de données d'entrée de gamme, semblable dans son utilisation à la version complète et destinée au public des versions Express ;
- Visual Studio 2005 Standard pour un développement déjà plus professionnel en C#, VB, J# ou C++, tant pour des applications en mode console, Windows ou Web que pour des applications pour mobiles ;
- Visual Studio 2005 Professionnel pour développeurs professionnels travaillant seuls ou en équipe restreinte ;
- Visual Studio 2005 Team System pour grosses équipes très structurées fondant leurs développements sur des outils de modélisation.

Cet ouvrage ne couvre pas les fonctionnalités propres à Visual Studio 2005 Team System.