Introduction à la programmation par composants

Solution .net

LAMURE Maxime / RAUGI Romain

Licence Informatique - Juin 2003

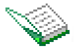


Table des matières

I- Introduction ……………………………………… P.3

II- Composants ……………………………………… P.4

1) *Composants client/IHM* ……………………………… *p.5*
2) *Composants serveur/métier* ……………………… *P.6*

III- Java/Corba ……………………………………… P.7

1) *Composants Java* ……………………………… *p.8*
2) *Services Java* ……………………………… *P.10*
3) *Corba* ……………………………… *P.11*
4) *Conclusion* ……………………………… *P.12*

IV- .net ……………………………………… P.13

1) *Philosophie .net* ……………………………… *p.14*
2) *En termes techniques* ……………………………… *p.15*
3) *Modèle de Composant* ……………………………… *p.21*
4) *Langage C#* ……………………………… *p.23*
5) *ASP.net* ……………………………… *p.30*
6) *Services Web* ……………………………… *p.31*
7) *ADO.net* ……………………………… *p.37*
8) *Conclusion* ……………………………… *p.42*

V- Conclusion ……………………………………… p.44

VI- Bibliographie ……………………………………… p.45

Introduction

La notion de réutilisation n'est pas nouvelle. Depuis le début de la programmation, on cherche à réutiliser ce qui a été déjà fait afin de le combiner, de l'intégrer. Les méthodes de réutilisation varient et se simplifient. En effet, au départ, le seul moyen de réutiliser du code était de le recopier dans notre programme. Puis sont apparues les premières bibliothèques de fonctions suivies des bibliothèques de classes. Mais depuis 95, l'émergence des composants logiciels permet une réutilisation plus simple et efficace grâce à ses interfaces et protocoles.

Nous allons, pour commencer, présenter ce qu'est un composant logiciel. Puis nous allons exposer le modèle de composant selon Java à travers les Beans et les EJBs, et l'infrastructure Corba. Enfin nous détaillerons la vision de Microsoft à ce sujet à travers .net.

Composants

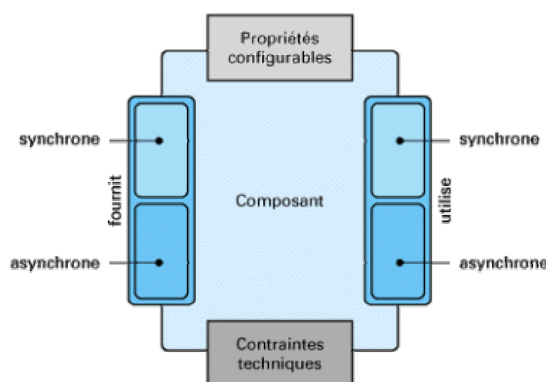
Le contenu d'un programme restera toujours le même. Néanmoins, sa phase de conception évolue. En effet, en 1972, un programme était fondé sur des données orchestrées par des algorithmes. Puis en 1990, Wegner introduisait l'Objet et la notion de message. Mais c'est en 1995, par Niertraz que les Composants et le terme de Connections apparaissaient. Mais qu'est ce qu'un composant ? La définition est assez floue et se rapproche beaucoup de celle de l'objet. En voici quelques unes :

- *« A software component is a static abstraction with plugs » Nierstraz et Tsichritzis(95).*
- *« A component is a piece of software small enough to create and maintain, big enough to deploy and support, and with standard interfaces for interoperability » Harris (95).*
- *« Software component are defined as prefabricated, pretested, self-contained, reusable software modules that perform specific functions » MetaGroup (94).*
- *« A software component is a unit of composition with contractually specified interfaces and explicit context dependencies. Software component can be deployed independently and is subject to composition by 3rd parties » Participant 1er Workshop Component Oriented Programming(97).*

En effet, il n'y a pas de consensus qui ait donné une définition commune aux composants. Selon les modèles (Corba, Java, Microsoft), les composants sont différents, ce qui rend difficile la tâche de les décrire.
A l'image des composants Java, il existe deux catégories de composants : les composants client/IHM (généraux) (Java Beans par exemple) et les composants serveur/métier (EJBs), dont le mode de fonctionnement diffère légèrement et qui répond à d'autres besoins.

I- Composants client/IHM

« Un composant est à l'objet ce que l'atome est à la molécule ». En effet, ce qui distingue un objet du composant, c'est d'une part sa granularité, mais aussi ses contraintes et ses fonctionnalités. On peut représenter un composant comme un type de boîte noire (la connaissance de son implémentation n'est pas nécessaire) constitué d'un ou plusieurs objets, où il suffit de connaître les services rendus et les quelques règles d'interconnexion. En effet, comme nous l'illustre la figure ci-dessous, un composant exporte les interfaces qu'il fournit et les interfaces qu'il requiert.



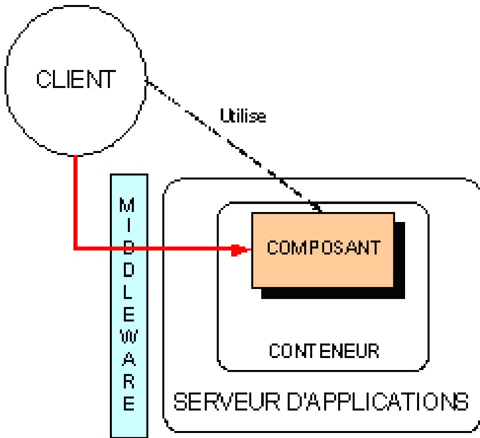
Ces interfaces sont aussi appelées *ports*. Un exemple de port est celui à la base de l'introspection. Celui-ci permet au composant de s'analyser lui-même (méthodes/propriétés via un mécanisme de réflexion) dans le but d'afficher une page des propriétés au sein de l'IDE sous lequel on développe, comme par exemple Visual Basic avec les contrôles ActiveX.
De plus, le composant est interconnectable avec d'autres composants d'origines diverses, configurable, auto descriptif (introspection), mais aussi, il est diffusable de manière unitaire et prêt à l'emploi. La plupart de ces fonctionnalités sont dues à ses interfaces. Dans le modèle CCM (Corba Component Model) de Corba par exemple, la connexion entre composants se fait grâce une interface réceptacle. De manière plus générale, on appelle cela des *connecteurs*. Pour le Java bean, les attributs du composant sont « créés » grâce à une interface donnant une spécification des noms des méthodes assesseurs et d'affectation, qui sont à la base de sa configuration.
En ce qui concerne la communication vis-à-vis des composants (application/composant ou composant/composant), on entend parler (comme l'illustre le schéma précédent) de communications synchrones, asynchrones voire de diffusions en continu (flots de données). La communication synchrone est en fait un simple appel de méthode comme il est possible de faire sur un objet, et le mode asynchrone reflète un mécanisme d'événements (abonnement/notification comme en Java). On appelle ce composant « composant IHM » car il est surtout dédié, à l'image des beans et des ActiveX, aux interfaces.
La programmation par composant est donc basée sur le fait d'intégrer, d'emboîter des composants entre eux pour former notre programme. A titre de comparaison, on pourrait prendre l'exemple des sociétés d'assembleur de Micro Informatique. Lors de la conception d'un PC, ils emboîtent des composants dont ils connaissent le principe de fonctionnement et la manière dont chacun d'eux peuvent communiquer avec son environnement.

II- Composants serveur/métier

Maintenant il existe un autre type de composant, appelé composant métier, qui est beaucoup plus spécifique. Il est aussi appelé composant serveur car il ne fonctionne que sur ce type de poste. En général on entend par composant ce composant-là, catégorie dans laquelle rentrent les EJBs de Java par exemple (et non les Java Beans qui peuvent s'exécuter chez le client). La présence d'un conteneur pour encapsuler le composant et d'un serveur d'applications les distingue des autres. C'est d'ailleurs eux qui répondent à d'autres besoins que les composants standards, pas les composants métier en eux-mêmes. Ces besoins expriment la volonté de ne pas se soucier des services non fonctionnels du composant, juste de sa partie métier.
Par exemple, quand on programme une application, il est nécessaire d'une part de se soucier de ce que fait l'application en elle-même, mais aussi d'autres caractéristiques plus proches du système et récurrentes à d'autres programmes, et qui ne sont pas incluses sous forme de services de manière standardisée. A l'échelle du composant, les services diffèrent mais l'objectif consiste en cela. Voici les services principaux masqués par les conteneurs :

- Nommage du composant.
- Gestion de la sécurité.
- Sûreté de fonctionnement (gestion de concurrence ou pannes par exemple).
- Persistance pour sauvegarder l'état d'un composant via une sérialisation.
- Prise en charge partielle des connecteurs inter composants.

En ce qui concerne le serveur d'applications, c'est sur lui que repose le conteneur. On parle aussi de « structures d'accueil ». Il joue le rôle d'interface entre le système et le conteneur et constitue un espace d'exécution pour le conteneur et ses composants. Voici un schéma illustrant ceci :



Un exemple de serveur d'applications est *Web Sphere* d'IBM, qui suit la spécification J2EE. Le Middleware qui est décrit ici est un terme générique pour parler de l'interface entre l'infrastructure réseau et la structure d'accueil. Cela peut être PHP, Java etc. Nous allons voir maintenant une représentation concrète de ce qu'est un composant dans les modèles Java.

Java/Corba

Popularisé par Internet, Java est devenu un langage à part entière ayant un succès majeur au sein des applications distribuées, et des applications réseau de manière plus générale. Dans cette partie sera traitée de la forme des composants au sein de l'environnement Java, de certains services et bibliothèques de J2SE et J2EE nécessaires à ce type de conception et enfin, d'une brève description de l'infrastructure Corba, qui concurrence les services Web qui seront détaillés plus loin.

I. Composants Java

a. *Java Beans*

Les Java Beans (ou beans) sont des composants Java, qui fonctionnent aussi bien chez un client que chez un serveur réseau et qui peuvent être de trois types : visuels (comme des boutons, des icônes etc.), non visuels (accès à une base de données) ou composites (feuilles de calculs, calendriers, etc.). Ils peuvent être utilisés dans n'importe quel programme Java, que ce soit une applet, une application, une servlet ou même une page JSP. Le bean est caractérisé par :

- Les services offerts (méthodes publiques par défaut)

- Ses propriétés (ou attributs) définis d'après ses méthodes publiques dont les signatures doivent respecter des conventions précises. Par exemple, l'existence des méthodes :

  ```
  Public T getP()
  Public setP(T val)
  ```

  Définira une propriété P de type T. Il peut y avoir plusieurs types de propriétés : des propriétés indexées (tableaux de valeurs de même type), et des propriétés liées (aux événements).

- Ses événements (dont le fonctionnement est similaire à ceux des bibliothèques AWT ou Swing, c'est-à-dire basés sur un abonnement et des notifications).

- La possibilité de l'éditer visuellement (via un environnement de développement de type *BeanBox* ou *JBuilder*).

- Une introspection pour visualiser ses caractéristiques depuis un environnement de développement sans avoir accès aux sources.

- Une instanciation en mémoire, qui détermine son contexte d'exécution (dans un programme final ou un environnement de développement) ou si c'est un bean visuel ou non.

- Une sérialisation pour sauvegarder son état courant.

- Sa forme de distribution (un package de type Jar).

- La possibilité de le connecter à d'autres beans, via un modèle de communication par événements.

Pour respecter ces caractéristiques, le bean doit obéir à des conventions lors de son développement. Ceci peut se faire de manière « manuelle » par le programmeur ou par le biais d'une interface, *java.beans.BeanInfo* (pour l'introspection). Cette interface est d'ailleurs surchargeable pour particulariser

l'introspection. Dans le cadre d'un bean visuel, celui-ci doit hériter de *java.awt.Component*. Il peut être complété avec d'autres classes, le tout devant être déployé sous forme de package Jar.

b. **EJBs** – Enterprise Java Beans

Les EJBs sont des Java Beans dont les fonctionnalités ont été étendues et ne fonctionnant que dans des environnements serveurs les prenant en compte. Ils sont disponibles dans la version *Entreprise* de Java. Alors que les Java Beans peuvent être utilisés à des fins très diverses, les EJBs sont dédiés à l'encapsulation de la logique métier d'une application et sont très spécialisés. Ils n'interviennent pas par exemple au niveau d'une interface utilisateur. En fait, ils s'apparentent plus à une servlet ou à une JSP qu'à un Java Bean.
Concrètement, un composant EJB est un ensemble de classes et d'un fichier XML permettant de le configurer. Les classes doivent respecter une certaine spécification (la spécification EJB), et le tout doit être dans un fichier unique pour former une entité (Jar). Ces composants, comme les beans, peuvent être accessibles par tous types de clients : des servlets, des JSP, des clients Java via Java RMI (*Remote Method Invocation*) ou via l'interface de Corba. Mais ce qui les caractérisent, ce sont la nécessité d'être dans un conteneur, appelé conteneur EJB. Il est nécessaire de distinguer deux catégories de ces composants :

- Les EJBs session

Ce type de bean sert un client à la fois et peut être vu comme une extension du client chez le serveur. Il peut par exemple servir de panier à un client dans le cadre d'un site commercial. Sa durée de vie doit être inférieure à celle du client.

- Les EJBs entité

Les beans entité servent à représenter selon un modèle objet des données d'une base de données. Ils peuvent être accédés par plusieurs clients simultanément et leur durée de vie est calquée sur celle des données qu'ils représentent. La relation intervenant entre la base et le bean, à la base du concept de persistance, peut être paramétrée par le développeur ou de manière automatique par le conteneur.

Le déploiement des EJBs consiste à produire un fichier Jar muni d'un fichier de configuration XML (dont la génération peut être automatisée) et à le mettre dans un répertoire reconnu par le conteneur. Celui-ci, dans l'environnement Java, n'est rien d'autre qu'un logiciel agissant comme une extension du serveur Web (au même titre qu'un moteur de servlets pour les JSP et les servlets). *JBoss* est un exemple de conteneur EJB Open Source.

II. Services Java

Ci-dessous sont présentés quelques services que peuvent offrir des plates-formes comme J2SE et J2EE pour aider à prendre en charge ce type de conception par composants, en dehors des bibliothèques pour les créer.

a. **RMI** – *Remote Invocation Method*

Java RMI appartient au monde des applications distribuées et permet, en ayant une vision très générale, d'accéder à des objets à distance. Le modèle de communication utilisé par les EJBs sur les réseaux étend Java RMI. RMI est disponible dans la version standard de Java.

b. ***J2EE Deployment API***

J2EE Deployment API est une méthode permettant de déployer une application ou un composant Java de manière standardisée.

c. **JNDI** - *Java Directory and Naming Interface*

JNDI intervient dans le milieu des services de nommages et des annuaires (qui sont des services de nommages munis d'une hiérarchie et de possibilité de recherches). Ces services font partie des éléments essentiels dans les environnements répartis et la programmation par composants. Ils peuvent être utilisés au sein des entreprises pour créer des bases d'informations sur les utilisateurs, le matériel, et … les composants. Ceci dans le but de retrouver et localiser un élément rapidement depuis son nom sur un réseau (comme par exemple le fait *DNS (Domain Name Service)*, qui est un annuaire, pour retrouver un site depuis son nom de domaine). En ce qui concerne JNDI, il s'agit d'une interface permettant d'utiliser ces services depuis une application Java, pour obtenir et stocker des objets et composants.

d. **JMS** - *Java Messaging Service*

JMS est un ensemble d'APIs qui s'utilise aussi avec les EJBs. Il s'agit, comme son nom l'indique, d'un service de messages, qui interviennent entre composants et applications de manière asynchrone.

III. Corba

Corba est un framework (ou plutôt des frameworks) assez vaste implémentant notamment son propre modèle de composants (CIF – *Component Implementation Framework*), ses conteneurs et une infrastructure de communication (CCM – Corba *Component Model*). Cette infrastructure est adaptée aussi à d'autres langages de programmation que le langage de Corba (CIDL – *Component Implementation Definition*). Plus concrètement, ce framework de Corba est un environnement client/serveur pour les applications réparties, ayant son propre langage et ses propres services. Son objectif est de masquer l'hétérogénéité des langages de programmation et des machines, via un langage « intermédiaire » d'interface (*IDL : Interface Definition Language*) permettant l'interopérabilité des langages, un bus spécifique pour l'acheminement des requêtes en IDL (*ORB : Object Request Broker*) et des services pour la programmation des applications réparties. Corba concurrence les services Web (qui seront traités au sein du chapitre sur .net), pour faire des applications distribuées hétérogènes (utilisant des composants reposant sur des plates-formes ou programmés dans des langages différents).

IV. Conclusion

Après avoir vu comment des composants peuvent être implémentés au sein du framework Java et être utilisés, on peut distinguer deux catégories : le composant dans le sens large du terme, à l'image du Java Bean, et le composant beaucoup plus technique, à l'image de l'EJB, qui ne s'exécute que dans un environnement particulier et via un conteneur. Java a été traité pour montrer ces deux types de composants et les services qui peuvent être offerts comme JNDI, RMI à la base de la communication vers EJBs dans des applications distribuées essentiellement Java, et plus généralement toutes les bibliothèques permettant de réaliser la COM (*Component Object Model*). Quant à Corba, il a été décrit brièvement pour montrer le rôle qu'il tient dans ces environnements par composants, c'est-à-dire à la fois celui de modèle de composants mais aussi celui d'infrastructure à la base de la communication inter composants hétérogènes, au même titre que les services Web.

On parle souvent de .net comme étant le concurrent de Java. Ce n'est pas exactement cela dans le sens où .net et Java ne sont pas de simples langages (C# pour .net) mais des frameworks apportant chacun leurs concepts dans le but de réaliser des applications distribuées sur les réseaux et des applications Internet (pour être très général). En cela, ce n'est pas « Java » que concurrence .net mais la spécification J2EE.
Pour situer .net dans l'optique des composants, on peut le placer au niveau des structures d'accueil (plate-forme d'exécution en plus d'intervenir en tant que plate-forme de développement).
Nous allons voir dans ce chapitre assez vaste, la philosophie de .net, pour décrire globalement ce qu'il apporte, puis ensuite une présentation du développement et de l'exécution des applications dans cet environnement, les modèles de composants COM et ActiveX et leurs évolutions dans .net, , le langage C# qui est le langage majeur de .net, ASP.net qui est à la base des applications Internet, les services Web, ce qu'ils sont et en quoi .net intervient, le framework ADO.net à la base de la persistance des composants et enfin en conclusion, quels intérêts y a t il dans .net pour Microsoft et pour les entreprises.

I- Philosophie .net

Voila bientôt 3 ans que l'on parle de .net, et pourtant peu de gens savent ce que c'est réellement. .net et plus particulièrement le framework .net est un environnement qui se veut le plus universel à tout point de vue.
En informatique, pour mener à bien un projet, il faut avoir des connaissances dans divers domaines et savoir manier différents outils spécifiques à un système, un langage … (debugger, OS). La philosophie de .net est tout contraire à cela. En effet le nouveau slogan DO MORE WITH LESS que prône Microsoft peut se prêter à cette idée (ce n'est pas le syllogisme de ce slogan).
Tout informaticien, qu'il soit analyste programmeur, technicien, chercheur (bien que le .net ne soit pas fait pour cela), chef de projet ou même administrateur réseau a des préférences au niveau du langage, de l'OS et même des outils, et ce choix lui est propre. En effet, des personnes vont préférer par exemple développer en Scheme car ils vont se sentir plus à l'aise, souvent pour des raisons de syntaxe (bien que les derniers langages se ressemblent syntaxiquement parlant), alors que d'autres préfèrent le C. Cette préférence se traduit souvent par le fait, qu'une personne qui va aimer Pascal va savoir faire des choses, par manipulation du langage, que d'autres n'auraient même pas pensées. On retrouve la même idée au niveau des OS, certaines personnes préfèrent un environnement graphique, convivial et accessible, alors que d'autres, qui y trouvent moins d'intérêt, préfèrent manipuler, utiliser la robustesse du système.
Et maintenant, imaginons un environnement de travail et d'exécution multi plateforme et multi langage. Notre pro du Pascal pourra, grâce à la richesse des APIs fournies, manipuler directement des bases de données sans passer par SQL, faire des Macros ou autre scripts. Tous ça avec le même langage. Et son application pourra tourner, sans aucune retouche, ni aucune recompilation, sur tous les systèmes présents comme Windows XX, Unix, MAC OS, mais aussi sur toutes les plates-formes présentes et à venir (PDA, Pocket PC, SmartPhone, montres, réfrigérateurs etc. Ce n'est rien de moins que l'objectif de .net.

II- En termes techniques

Avec le framework .net est apparu de nouveaux termes : MSIL, CLR, CLS …
Pour vous expliquer le plus simplement possible comment marche .net, on va voir comment se déroulent les différentes étapes de la programmation à l'exécution d'une application :

a. Le développement

Il faut bien entendu passer par là. Coder son programme dans le langage que l'on désire parmi les 24 langages actuellement supportés par le framework. Enfin, ce n'est pas exactement vrai. En fait, ce n'est pas 24 langages que reconnaît le framework, mais 24 syntaxes qui sont propres aux langages. Je m'explique. Si vous faites du C, généralement, pour utiliser la puissance de ce langage, vous allez utiliser les pointeurs, et donc faire des mallocs. Or malloc appartient à la bibliothèque stdlib qui, elle-même, appartient à la bibliothèque C, mais pas à l'API du framework. Donc il est interdit de faire des mallocs si vous voulez construire des applications .net. L'exemple le plus flagrant est Java, le J# reprend toute la syntaxe de java, mais on ne peut pas utiliser l'API de Sun. Mais pourquoi l'avoir renommé J# ? Certain langages ont subi des transformations pour pouvoir être utilisé. C'est le cas, par exemple, de Visual Basic qui, par les transformations apportées à son passage à .net, est devenu un nouveau langage.
Le nombre de langages (syntaxes) supporté augmente de plus en plus. On parle de S# pour Smalltalk, X# pour un langage basé sur le XML … Pourquoi refaire ces langages ? Et bien disons que chaque langage a sa spécificité et qui de ce fait peut le rendre très compliqué. Si certaines propriétés ne sont pas développées par le framework, alors on créé un nouveau langage, reprenant les mêmes principes, mais dépourvu de tout ce qui est incompatible .net. C'est le cas pour Cobol et Smalltalk. Voici les langages actuellement disponibles :

☐ Perl	☐ Managed C++
☐ Python	☐ Visual Basic
☐ Cobol	☐ C#
☐ Haskel	☐ J#
☐ ML	☐ SmallTalk
☐ Jscript	☐ Oberon
☐ ADA	☐ Scheme
☐ APL	☐ Mercury
☐ Eiffel	☐ Oz
☐ Pascal	☐ Objective Caml
☐ Fortan	☐ …

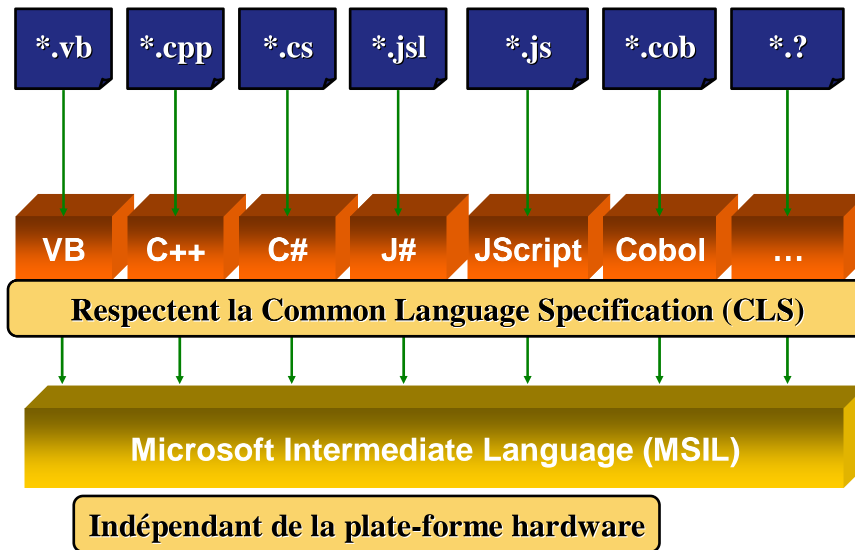
Bien que tous ces langages aient perdu le droit d'utiliser leurs bibliothèques, ils se voient enrichis d'une API énorme optimisée pour le framework.
Pour faire cohabiter tous ces langages, Microsoft à dû adopter un système de type commun à tous les langages de la plateforme, la CLS, qui formalise les conditions nécessaires à l'interopérabilité des langages.

Alias		
Visual Basic.Net	Visual C#.Net	.NET Runtime type structure
Boolean	bool	System.Boolean
Byte	byte	System.Byte
Char	char	System.Char
Date		System.DateTime
Decimal	decimal	System.Decimal
Double	double	System.Double
Integer	int	System.Int32
Long	long	System.Int64
Object	object	System.Object
Short	short	System.Int16
Single	float	System.Float32
String	string	System.String

En fait, pour qu'un nouveau langage puisse être compatible avec le framework .net, il faut que ce dernier remplisse une sorte de cahier des charges reprenant les caractéristiques de cet environnement. La CLS est ce fameux cahier des charges.
Chaque langage possède ses propres mots clés, ses constantes. Imaginons que l'on se trouve dans le cas où vous développez une application A en VB .net, et qu'elle soit utilisée par un serveur S écrit en C#. Imaginons maintenant que dans le code de A se trouve une fonction dont la signature est utilisée par la syntaxe du C#. La compilation de S engendrerait une erreur. Pour y remédier, la CLS possède dans son jeu d'instructions un caractère d'échappement '@' qui permet de spécifier qu'on a développé l'instruction qui suit ce caractère.
Mais il est vrai, et ce TE le montre, que cette première partie va progressivement diminuer dans le développement de projet informatique (sans jamais réellement disparaître) par l'intégration de composants dans notre application.

b. *La compilation*

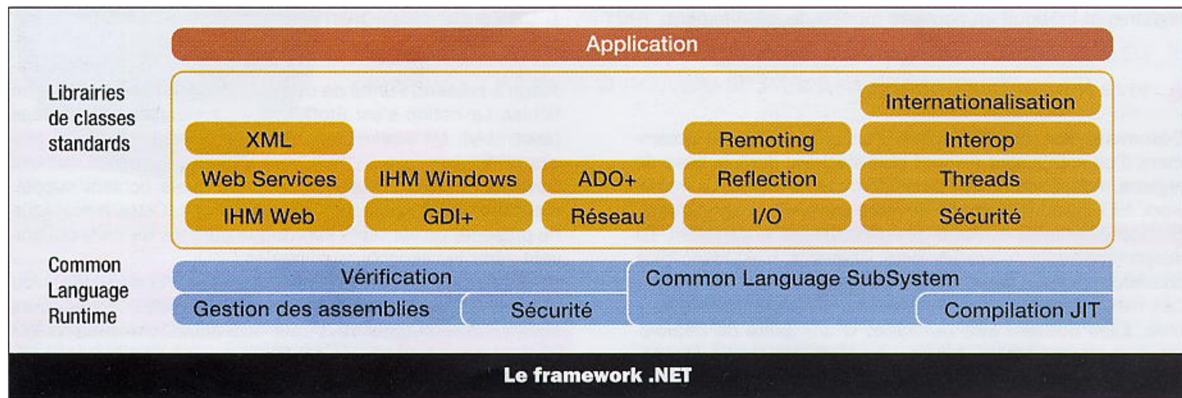
Une fois le programme écrit, on passe à la compilation. Cette étape fait un prétraitement de notre langage préféré en le transformant en CIL (aussi appelé MSIL) : Le CIL, ou Commun Intermediate Language, est en fait le seul langage qui pourra permettre l'exécution dans le framework. Ce langage est déposé à l'ECMA (http://www.ecma.ch) .C'est lui qui permet l'interopérabilité, au même titre que le java Byte Code, mais ici, on est pas contraint de développer en Java. On obtient alors un fichier avec comme extension .modules ce qui correspond au .class de Java.



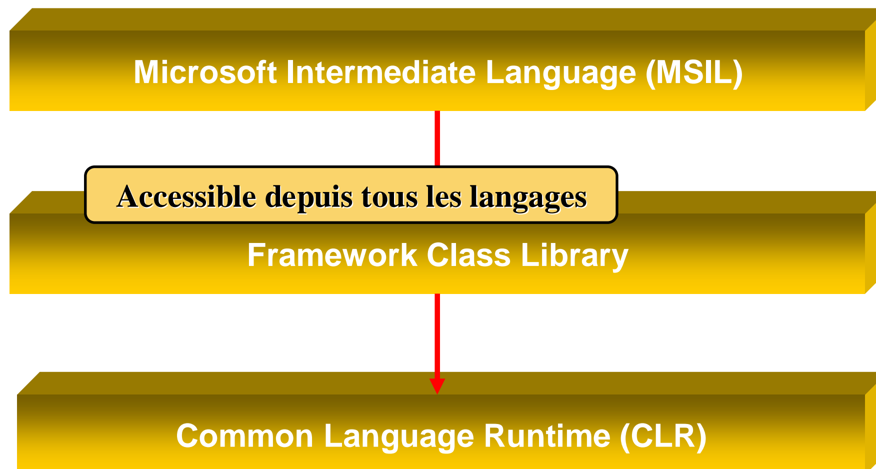
Le CIL nous permet de mieux comprendre de quelle façon, il est possible de relier entre eux des composants et des programmes écrits dans des langages différents et avoir un seul runtime.
La phase d'édition de lien consiste simplement à rassembler plusieurs fragments de code IL, avant d'arriver à l'exécutable final. Il est donc très facile d'ajouter des composants de langages différents. Une fois l'assemblage finit, on obtient un assembly (.dll) au même titre qu'on obtenait un .jar en Java.

c. *L'exécution*

Arrivé à ce niveau, il nous reste plus qu'à exécuter notre programme. Lors de la première exécution, la framework commence par rassembler tout le code IL, et toutes les fonctionnalités des classes unifiées qu'il utilise dans un seul code. C'est lors de cette phase que les notions de sécurités rentrent en compte. En effet, il est fort probable que le contenu des composants que vous utilisez vous soit inconnu, surtout si on utilise les services Web. On fait appel à des services externes, donc à partir de là, il faut une sécurité optimale. C'est la CLR (ou Common Language Runtime) qui se charge de cela. Cette dernière gère tous les aspects de sécurité et d'exécution. C'est grâce à sa robustesse, que votre programme pourra être exécuté en toute sécurité.



Juste après cette vérification, le compilateur JIT (Just In Time) va transformer le code IL en code natif de la plate-forme courante, afin de gagner en vitesse d'exécution. C'est pour cette raison que la première exécution d'un programme .net est toujours plus lente, mais les suivantes se font à pleine vitesse.
En fait le principe est simple, le programmeur développe son programme en communiquant avec la CLR qui elle communique avec les APIs système. C'est pour cette raison, qu'il faut un framework par système.



d. *L'enfer des DLL*

Tous les développeurs qui travaillaient sur Windows connaissent bien l'expression « l'enfer des DLL ». En fait cela représente bien l'atmosphère de maintenance dans ce système. Si vous voulez faire des mises à jours d'un programme, il fallait, jusqu'à maintenant, développer de nouvelles DLL qui remplaceraient les anciennes. Mais, si un programme utilisait les anciennes, un bug apparaissait. Avec .net, toute cette confusion est finie. En effet, on accompagne chaque assembly d'un fichier de configuration écrit en XML, à l'intérieur duquel on indique la

version de la DLL à utiliser. Vous l'aurez bien compris, chaque assembly contient un nom pour l'identifier, une clé de sécurité et un numéro de version.
Le choix du standard XML permet une facilité de lecture (texte), mais il est avantageux par le fait qu'il soit multi plateforme et qu'il passe par les firewalls (ce qui n'est pas le cas des fichier .ini).
L'avantage de ce système est qu'on peut regrouper toute nos assemblies dans un répertoire spécial, appeler GAC ($WINDIR\assemblys) qui permet de les rendre accessibles par tous. On peut conserver nos anciennes versions toute en ayant dans le même répertoire la mise à jour possédant le même nom, mais pas le même numéro de version. Les programmes utilisant nos assemblies n'auront qu'à indiquer le numéro de version nécessaire à leur bon fonctionnement dans le fichier de configuration correspondant.

e. *Mise en œuvre*

Voici un exemple qui nous montre l'interopérabilité de .net. Nous allons faire un programme qui va calculer la somme et le carré d'un nombre. Pour ce faire nous allons écrire une classe ComplexMath en C# qui va contenir une fonction Square calculant le carré de notre nombre, et une classe SimpleMath en Visual Basic .net qui va contenir la fonction Add et Sub calculant respectivement la somme et la différence de deux nombre entier.

complex.cs :

```
using System;
public class ComplexMath{
	public int Square(int a){
		return a*a;
	}
}
```

simple.vb

```
Imports System

Public class SimpleMath
	Function Add(a As Integer,b As Integer) As Integer
	return a+b
	End Function
End class
```

Maintenant nous allons écrire le serveur en C#. Ce dernier vas utiliser nos fonctions développées en C# et en Visual Basic .net. (Après que ces dernières aient été compilées).

```
using System;
	public class MyApp
	{
		static void Main()
		{
			simplemath simple = new simplemath();
			int sum=simple.Add(2,2);
			Console.writeln("2+2={0}",sum);  //{0} se referent au 
premir parametre apres la virgule ...
			int sq=simple.Square(2);
			Console.writeln("sq(2)={0}",sq);
		}
	}
```

Et pour finir nous allons écrire son fichier de configuration en XML.

```
<?xml version="1.0" encoding="utf-8" ?>
	<configuration>
		<runtime>
			<assemblyBinding xmlms="urn:schemas.microsoft.com:asm.v1">
			<probing privatePath="bin"/>
			</assemblyBinding>
		</runtime>
	</configuration>
```

Methodologies de compilation:

1: on compile nos fichiers sources avec l'option /target :module pour obtenir des modules (ni exécutable, ni dll → CIL)
csc /target :module complex.cs	pour le C#
	|---> création de complex.netmodule
vbc /target :module simple.vb	pour le VB.net
	|---> création de simple.netmodule

2 : On crée notre assembly que l'on nommera Math.dll grâce à l'option /out:math.dll
al /target :library /out :math.dll simple.netmodule complex.netmodule

3: On compile notre serveur avec notre assembly:
csc /target.exe /reference :math.dll mathdemo.cs

Notre exécutable mathdemo.exe est créé.

III- Modèle de Composant

Le framework .net dispose de son propre modèle de composant. Nous allons voir dans cette partie l'architecture COM/ActiveX, qui fonctionne toujours dans le framework .net et les composants .net en eux-mêmes avec leurs différences.

a. Architecture COM/ActiveX

COM est le modèle de composant propre à Microsoft depuis 1990 (avec O.L.E – *Object Linking and Embedding*, qui permet d'insérer des composants dans des applications ou des documents et de les lier), et qui est à l'origine des contrôles ActiveX (en 1996), plus adaptés à Internet. Comme tout modèle de composants, COM permet de masquer leur langage de programmation et de les traiter de manière indépendante. Ceci se fait via une ou des interfaces, qui sont utilisables grâce à une implémentation spécifique au sein du composant. Voici quelques unes de ces interfaces :

IUnknown (obligatoire)	• QueryInterface() • AddRef() • Release()
IDimensions	• SetWidth() • SetHeight() • getWidth() • getHeight()
IGraphics	• Draw() • Move() • Rotate() • Scale()

IUnknown est obligatoire et permet de consulter dynamiquement les interfaces du composant et gérer le « reference counting » (gestion de sa durée de vie et de la mémoire).
Un composant COM nécessite ce qui est appelé une librairie de type (générée à partir d'une description IDL (comme Corba, sauf que le langage est différent) depuis un fichier ODL (*Object Description Language*)), qui permet de le décrire complètement, avec notamment ses interfaces. Cette librairie peut être incluse comme ressource au sein d'un programme ou d'une DLL. Chaque composant COM est identifié par un identifiant nommé CLSID (*CLaSs IDentifier*) qui est stocké dans la base de registres de Windows. Il permet d'associer le composant à l'exécutable qui l'implémente (.exe ou .dll).
Chaque objet COM est créé depuis un serveur COM, qui est déployé sous forme d'exécutable ou de DLL, et qui regarde les entrées CLSID de la base de registres pour savoir lesquels il peut instancier en mémoire. Ce serveur permet en outre de les distribuer sur des réseaux (via une évolution de COM : DCOM).
En ce qui concerne les contrôles ActiveX, ils incarnent l'évolution des anciens contrôles VBX, qui étaient limités à *Visual Basic* et aux environnements Windows 16 bits, et OCX, liés à O.L.E. ActiveX ne remplace pas COM mais utilise son architecture et ajoute quelques spécifications et contraintes. Cette technologie

permet, comme pour les Java Beans, de manipuler les composants via des méthodes, événements et propriétés et d'avoir accès à des pages de propriétés.
En définitive, COM et ActiveX incarnent les composants propres à l'environnement Windows, non portables car utilisant des mécanismes propres à celui-ci (base de registres, code natif) et, peut être, le futur passé du composant. Etant donné l'imposant existant basé sur cette technologie, le framework .net assure encore son support pour plusieurs années, grâce à des mécanismes automatisés permettant de voir ce type de composant comme une classe du framework. Des contraintes sont à respecter, des pertes de performances impliquées mais l'interopérabilité est assurée.

b. Composant .Net : Assemblies

Le composant dans le framework .net est un Assembly (ou Assemblage), qui est un package contenant un manifeste (fichier de description), des classes codées en MSIL (.dll et .exe) et d'autres ressources comme une documentation. Il en existe de plusieurs types, privés donc visibles que par une application et partagés, qui nécessitent un enregistrement dans un cache appelé GAC – Global Assembly Cache.
Voici ce qui distingue fondamentalement les assemblies :

- Ils n'ont pas besoin d'un fichier de description, ils s'auto décrivent grâce à un « manifeste d'assemblage » basé sur des méta données (cf. chapitre sur C#), identiques à tous les langages .net et accessibles grâce à un mécanisme de réflexion (analyse des attributs, méthodes et constructeurs d'une classe chargée).
- Ils n'ont pas besoin d'interface, ils communiquent directement.
- Ils n'ont pas besoin d'être enregistré dans la base de registres, grâce au manifeste d'assemblage.
- Ils peuvent cohabiter dans plusieurs versions différentes, grâce au GAC et à un contrôle de version effectué par la CLR via le manifeste.
- Ils ne gèrent pas eux-mêmes complètement la mémoire qu'ils allouent, elle est gérée principalement par le GC (ramasse-miettes) du framework.

Les assemblies s'apparentent plus aux Java Beans de Sun qu'aux EJBs, du moins par défaut. En effet, ils peuvent s'exécuter dans un conteneur via une application .net : MTS (Microsoft Transaction Server), qui définit à la fois le modèle de programmation des composants et leur sert d'environnement d'exécution. Dans ce cas là, l'assembly s'apparente à un EJB Session. En définitive, là où en Java il y a deux types de composants (bean et EJB), dans .net il y en a un seul (assembly). Le framework se base encore, à l'heure actuelle, sur un serveur d'applications et l'infrastructure évoluée de COM+ (COM+ 2.0).

c. Conclusion

Ce chapitre montre que les composants ne sont pas nés avec des plates-formes comme J2EE ou .net mais bien avant, COM l'illustre. Mais la tendance actuelle du modèle de programmation par composants tend vers l'indépendance vis-à-vis de l'environnement (pour la portabilité), la gestion des versions et la simplification (dans le déploiement par exemple).

IV- Langage C#

Avec la sortie de framework .net est apparu un nouveau langage : le C#. La conception de ce langage s'est inspirée de ce qu'il y a de mieux actuellement. En d'autres termes, le C# avait pour objectif de reprendre la rapidité du développement de Java, sa simplicité, sa conception objet et sa sécurité avec la puissance, le contrôle et la rapidité du C++. Par le manque d'expérience de ce langage, on ne peut affirmer cela, mais tout porte à croire qu'il va tenir sa promesse. Pour preuve, le framework, que tout les spécialistes décrivent comme une prouesse technologique, est développé en C#. On peut dire que le C# est au framework ce que le C est à Unix. Le C#, tout comme son confrère MSIL, est déposé à l'ECMA et est à la norme ISO.
Avec sa conception orientée objet, C# permet le développement de composants allant des objets de haut niveau aux applications de niveau système. De même, grâce à quelques constructions simples, C# permet de convertir des composants en services Web qui pourront êtres appelés sur Internet à partir de n'importe quel langage s'exécutant sur n'importe quel système d'exploitation. Et surtout le C#, par le fait qu'il se soit inspiré du Java et du C/C++, pourra permettre aux développeurs d'être rapidement productifs.

a. *Les similitudes avec le langage Java*

- C# possède une superclasse, mère de tous les objets : *System.Object* comme Java avec *java.lang.Object*, très semblable excepté les méthodes *wait()* et *notify()* et les autres méthodes liées à la synchronisation qui ne sont pas disponibles dans cette classe en C#.

- Au niveau de la syntaxe, quasiment tous les mots-clefs Java ont un équivalent en C#. Voici un tableau provenant du site *dotnetguru* les présentant :

mot-clé C#	mot-clé Java	mot-clé C#	mot-clé Java	mot-clé C#	mot-clé Java	mot-clé C#	mot-clé Java
abstract	abstract	explicit	N/A	object	N/A	this	this
as	N/A	extern	native	operator	N/A	throw	throw
base	super	finally	finally	out	N/A	true	true
bool	boolean	fixed	N/A	override	N/A	try	try
break	break	float	float	params	N/A	typeof	N/A
byte	N/A	for	for	private	private	uint	N/A
case	case	foreach	N/A	protected	N/A	ulong	N/A
catch	catch	get	N/A	public	public	unchecked	N/A
char	char	goto	goto[1]	readonly	N/A	unsafe	N/A
checked	N/A	if	if	ref	N/A	ushort	N/A
class	class	implicit	N/A	return	return	using	import
const	const[1]	in	N/A	sbyte	byte	value	N/A
continue	continue	int	int	sealed	final	virtual	N/A
decimal	N/A	interface	interface	set	N/A	void	void
default	default	internal	protected	short	short	while	while
delegate	N/A	is	instanceof	sizeof	N/A	:	extends
do	do	lock	synchronized	stackalloc	N/A	:	implements
double	double	long	long	static	static	N/A	strictfp
else	else	namespace	package	string	N/A	N/A	throws
enum	N/A	new	new	struct	N/A	N/A	transient
event	N/A	null	null	switch	switch	N/A	volatile

- Même gestion des tableaux à plusieurs dimensions.
- Pas de méthodes globales (en dehors de classes) comme en C++, pour les deux langages.
- Même gestion des interfaces et pas d'héritage multiple.
- Chaînes de caractères non modifiables (modifiables via un *StringBuffer* en Java et un *StringBuilder* en C#).
- Possibilité de rendre des classes non extensibles (*final* en Java, *sealed* en C#), dans le but de ne pas redéfinir leurs méthodes.
- Gestion des exceptions très proches, excepté le fait qu'en C#, il n'y a pas de mot clef *throws* et pas de contrôle sur les exceptions à l'exécution.
- Gestion des variables et constructeurs (blocs) statiques identiques.
- Le point d'entrée est *main* en Java et *Main* en C#.
- La syntaxe de l'héritage diffère. Exemple :

  - Java : class A extends B implements C
  - C# : class A:B, IC

  Les interfaces en C# par convention sont nommées Inom_interface.

- *instanceof* en Java devient *is* en C#.

- *package* en Java devient *namespace* en C#. L'arborescence du package représentée par nom1.nom2... en Java ne signifie rien en C#. Dans ce dernier, il est d'ailleurs possible de définir plusieurs namespaces au sein d'un même fichier.

- C# possède des destructeurs comme en C++, alors que Java possède des *finaliseurs*. C# possède en outre des *finaliseurs*, mais ils sont plus délicats à manipuler.

- *Synchronized* en Java devient *Lock* en C# pour la synchronisation des blocs. En ce qui concerne les méthodes, en C# il est nécessaire de faire appel à une méta donnée ou à un mot clef *Interlocked*.

- Accessibilité à une classe : Voici les changements dans la syntaxe :

C#	Java
private	private
public	public
internal	protected
protected	N/A
internal protected	N/A

internal protected permet d'étendre la visibilité aux classes dérivées. La visibilité par défaut d'un champ ou d'une méthode est *private* en C# et *protected* en Java.

- La réflexion est disponible dans les deux langages. Elle consiste à découvrir et invoquer à l'exécution les méthodes et attributs d'une classe. La différence entre les deux langages va sur la portée, sur l'Assembly en C# et sur la classe en Java.

- *final* en Java permet de déclarer des constantes à la compilation et à l'exécution. *const* en C# est utilisé pour la compilation et *readonly* pour l'exécution.

- Chaque type primitif en Java a un équivalent en C#, excepté quelques différences sur le sens de *byte* et sur la possibilité qu'à C# d'avoir des types non signés.

- A la différence de Java, C# n'offre pas la possibilité de déclarer des tableaux comme en C.

- En ce qui concerne le chaînage et l'appel au constructeur père, il n'y a quasiment pas de différences. *super* en Java devient *base* en C#.

b. Les Differences

- C# ne possède pas de classes internes non statiques comme en Java.
- La gestion des threads est différente en Java. *Object* n'a pas de méthodes associées à ce type d'utilisation (*wait()* et *notify()* n'existent pas).
- Surcharge de quelques opérateurs en C# (pas tous ceux de C++).
- Les *Switch* C# autorisent le test sur chaînes de caractères et interdisent le « Fall-through » qui exécute plusieurs « case » si il n'y a pas de *break*.
- Les collections Java sont plus complètes (pas de bibliothèques pour les ensembles en C# par exemple).
- C# possède un *goto*.
- La gestion des fichiers Entrée/Sortie est plus simple en C#.
- La sérialisation en C# se fait via des méta données alors qu'en Java elle se fait via une interface. De plus, la sérialisation C# peut être faite en binaire ou en XML.
- C# comme Java permet de générer via des commentaires spécifiques une documentation. La syntaxe est différente, C# utilise XML.
- Il n'est pas obligatoire de donner au fichier C# le nom de la classe codée. Cela implique la possibilité en C# de déclarer plusieurs classes publiques dans un même fichier.
- *import* en Java devient en *using* en C#. L'import de bibliothèques en C# ne se fait pas via un CLASSPATH mais via une option du compilateur « /r ».
- Le mécanisme de gestion d'événements en C# se fait via des délégués (pointeurs sur fonctions).
- En C# il est impossible de déclarer des constantes dans des interfaces.
- Meilleure interopérabilité de C#. Il est par exemple possible d'importer des méthodes de DLL WIN32 plus aisément qu'en Java, c'est-à-dire sans passer par ce qu'on appelle un *wrapper* de code.

c. L'évolution du C# vis-à-vis de Java

La philosophie de ce langage est d'être le plus simple possible à écrire. C'est pour cela qu'il se rapproche beaucoup de Java. Néanmoins, ils ont enrichi leur langage de syntaxes et de termes pour prendre le dessus.

- Possibilité pour le développeur C# de contrôler la libération d'objets via l'interface *IDisposable*.

- Les délégués C# qui permettent d'implémenter des mécanismes de callback, qui, concrètement, sont des pointeurs sur fonctions.

- Les types Enumération (*enum*) existent en C#.

```
public enum Season
{
        Spring,Summer,Autumn,Winter
}

public struct Point
{
        public Point(int x, int y)
        {
                ...
        }
}
```

Seulement, tous les langages ne supportent pas les énumérations dans leur syntaxe. Pour conserver l'interopérabilité, l'énumération est traduite lors de son passage dans un assemblage (CIL).On obtiendrait par exemple.

```
public seales class Season
{
        ...
        public const int Spring =0;
        public const int Summer =1;
        public const int Autumn =2;
        public const int Winter =3;
}
```

- ValueTypes : il est possible en C# de déclarer des objets qui seront stockés directement dans la pile et qui ne seront pas libérés par le ramasse-miettes. Pour cela, au lieu d'utiliser le mot-clef *class*, il faut utiliser le mot-clef *struct*.

- Boxing : C# permet de traiter des objets comme des types de valeur de manière implicite, sans passer par des classes « intermédiaires ». Un exemple connu en Java est l'encapsulation d'un *int* par un *Integer* pour pouvoir le passer en paramètre en tant qu'*Object*. En C#, c'est automatique.

- C# possède un identificateur de type *as*, permettant d'effectuer une assignation de valeur et une conversion de type en même temps.

- L'instruction *foreach* de C# permet aisément de parcourir des collections.

- Mécanisme de propriétés en C# masquant l'implémentation d'assesseurs.

- Une des différences les plus fondamentales entre C# et Java concerne les attributs ou méta données, gérés en natif par la CLR .net. Dans C#, ils permettent par exemple de sérialiser ou encore de créer des services Web (cf. chapitre associé) aisément. Ils se présentent sous la forme : [attribut] et peuvent être personnalisés.

- C# dispose d'indexeurs qu'il est possible de surcharger (via l'opérateur []).

- C# possède un pré processeur comme en C et permet donc d'utiliser des directives de pré processeur comme *#define* ou *#undef* et de faire de la compilation conditionnelle via *#if* et *#elif*. En outre, *#error* et *#warning* et permettent de générer des messages à la compilation.

- C# possède des aliases similaire aux *typedef* de C.

- Génération de code à l'exécution, qui n'est pas proposée en standard avec Java.

- Possible utilisation de pointeurs dans des blocs *unsafe* (non protégés). Permet de faire de la programmation de bas niveau avec C#.

- C# utilise le passage de paramètres par référence contrairement à Java, via un mot clef *ref*. Dans le cas où une méthode se charge de créer l'objet, un mot clef *out* doit être utilisé (un peu à la manière d'ADA).

- Liste variable de paramètres en C# via le mot clef *params*.

- C# gère de manière différente les caractères spéciaux dans une chaîne de caractères avec l'aide du symbole @ avant la chaîne elle-même. Cela évite d'utiliser les caractères d'échappement de type backslash.

- Il est possible en C# de détecter des débordements dans le cas de conversions de types. Ce mécanisme étant assez lourd, il est possible de le désactiver via des mots clefs *checked* et *unchecked* en début de bloc.

- C# propose l'implémentation explicite d'interfaces dans le cas où des méthodes seraient de signatures identiques à plusieurs d'entre elles, via un casting.

- Possibilité de charger des classes à distance en C# (comme Java) avec l'API .NET Remoting.

De plus, le C# utilise des librairies simplifiées, ce qui permet un meilleur lisibilité du code et une plus grande facilité d'écriture. Pour illustrer cela, voici deux exemples qui lisent un fichier en paramètre :

En Java :

```
import java.io.*;

public final class SourceFile{
   public static void read(String filename) throws IOException
   {
      FileReader file =new FileReader(filename);
      BufferedReader reader = new BufferedReader(file);
      try
      {
         readFrom(reader);
      }
      finally
      {
         reader.close();
      }
   }
}
```

En C# :

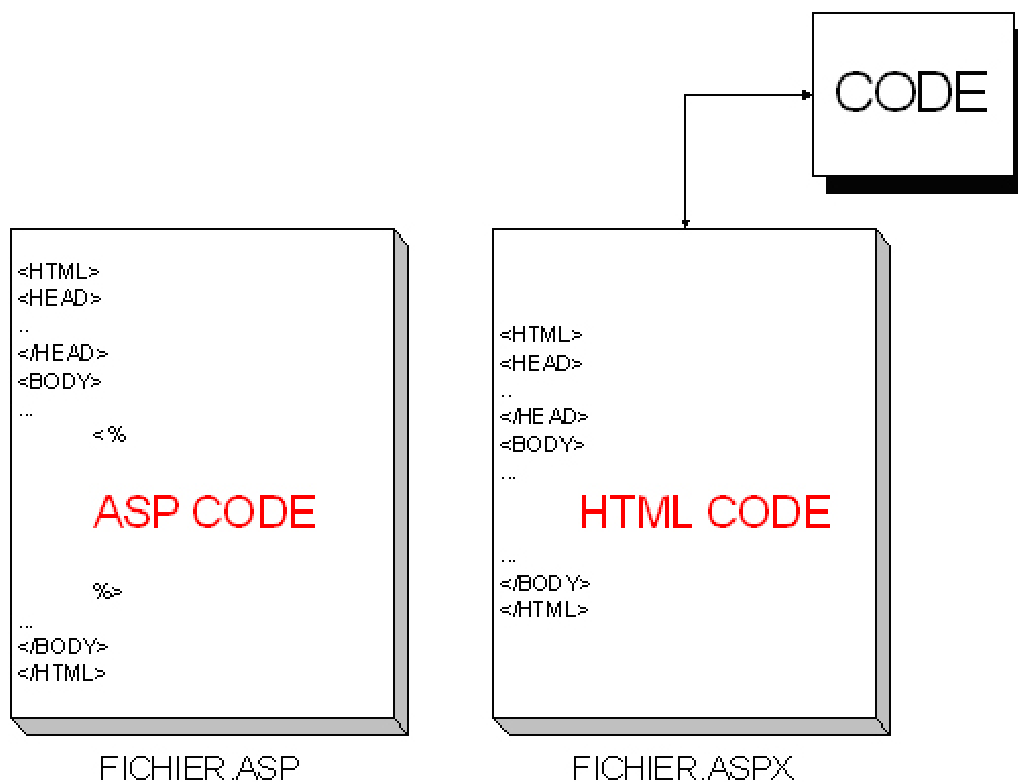
```
using System.IO;

public sealed class SourceFile
{
	public static void read(String filename)
	{
		StreamReader file = new StreamReader(filename);
		try
		{
			readFrom(reader);
		}
		finally
		{
			reader.Close();
		}
	}
}
```

Le résultat est le même, le code Java n'est pas bien plus compliqué, mais il est vrai que l'on fait la même chose avec moins de ligne en C#.

V- ASP.net

ASP.net permet de faire des pages Web dynamiques. On ne peut pas lui prêter le terme de langage, car ça n'en est pas un, contrairement à son concurrent direct PHP. En effet ASP.net se sert des langages reconnus par le framework, qu'il intègre dans les tags d'une page HTML.
L'évolutions par rapport à son prédécesseur, c'est qu'au lieu d'avoir un seul fichier .asp contenant toutes les données, on a deux fichiers : un fichier .aspx contenant les tags HTML et un fichier .aspx.cs (pour le C#) contenant le code. Ce dernier devant être compilé pour l'exécution.



L'intérêt majeur de ce type de développement réside dans le fait que le développeur et les designers peuvent travailler indépendamment. ASP.net est avant tout une technologie permettant de créer des applications (Web Forms) et des services Web sans aucune nécessité de manipuler XML/SOAP. L'un des grands intérêts de développer en ASP.net, c'est sa facilité à intégrer des composants, appelés ici Starter Kits. Ces derniers sont très complets. Ils intègrent une interface qui permet de les modifier et de les configurer selon nos désirs. De plus, ces starters Kit sont gratuits, libres, et nous permettent de nous en servir à des fins commerciales, ce qui n'était pas possible avec d'autres technologies.

VI- Web Services

Les services Web sont une nouvelle technologie de l'informatique qui s'appuie sur un vieux rêve visant à faire « communiquer » des applications hétérogènes et mettre à disposition des fonctionnalités pour des logiciels sur un réseau, et qui est en phase de devenir réalité. Les services Web sont un des principaux apports du framework .net. Dans cette partie seront détaillés ce qu'ils sont, les architectures et protocoles utilisés et, enfin, le déploiement d'un service .net, au travers d'un exemple.

a. *Presentation*

- *Emergence des services Web*

Faire communiquer des programmes indépendamment de leur langage de programmation et des plates-formes sur lesquels ils reposent, tel est l'objectif des services Web, de manière générale. Concrètement, les services Web sont une manière d'invoquer (via une URL) une méthode d'un composant à distance, dans le but de disposer d'une certaine fonctionnalité sans avoir à se soucier de son implémentation. Ils ne sont pas spécifiques à .net, bien que ce soit Microsoft qui en soit à l'origine et qui, en partenariat avec IBM, l'ai fait arriver à maturité. Beaucoup d'entreprises aussi prestigieuses que Borland et même Sun avec Java (JAXP, JAX-RPC etc.) proposent des environnements de développement. Des langages de programmation existant ont même été adaptés (Perl par exemple), ainsi que des serveurs Web aussi généralisés qu'Apache. Une des raisons est qu'ils s'appuient sur des protocoles standard ouverts tels que HTTP, SOAP et XML, contrairement par exemple à Corba. A terme, les services Web permettront de déployer des applications distribuées au sein de réseaux intra entreprises voire même, à plus long terme, au sein de réseaux inter entreprises.

- *Mise en œuvre*

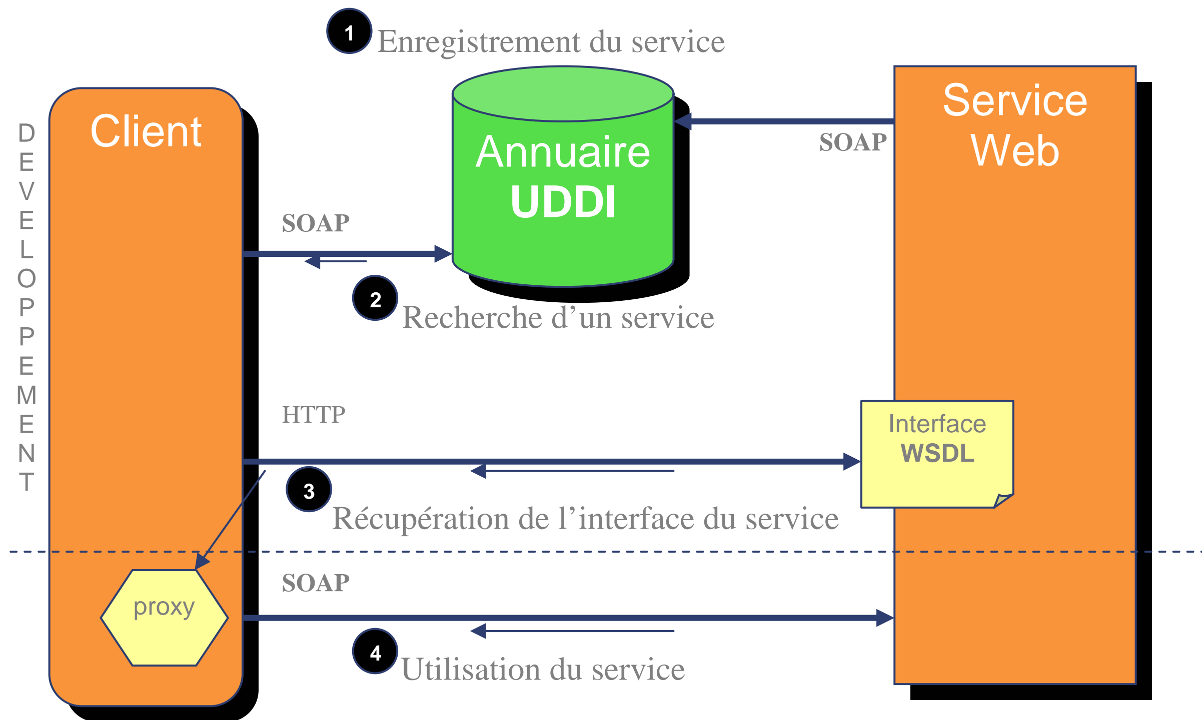
Pour déployer un service Web, on peut distinguer trois phases ou trois « chantiers » :

- Publication du service :
  Pour « mettre » le service sous une forme standardisée (interface avec WSDL, cf. « Architecture »).

- Accès à distance grâce à SOAP :
  Pour le rendre accessible à distance sur un réseau, il est nécessaire de « modéliser » les messages qui vont intervenir entre le client du service et le serveur. Plus concrètement, il s'agit de déployer une API basée sur SOAP et WSDL.

- Référencement dans un annuaire :
  Pour permettre à d'éventuels intéressés d'avoir accès au service.

Voici un schéma provenant d'un diaporama d'un séminaire Microsoft sur les services Web :

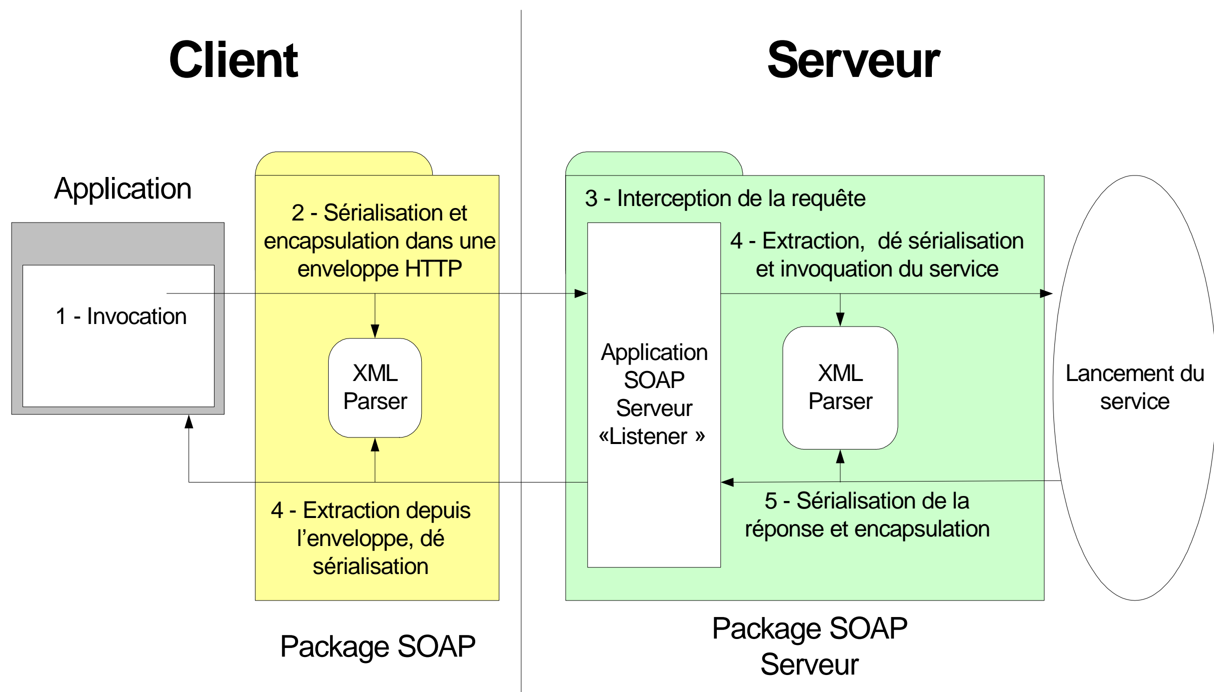


La première phase décrite ci-dessus est l'enregistrement du service (une fois développé) dans un annuaire. Ici il s'agit d'UDDI (*Universal Description, Discovery and Intégration*), lancé par Microsoft et IBM, qui est un annuaire mondial référençant les connexions aux services. La deuxième phase montre un client recherchant ce service dans l'annuaire. La troisième montre le client récupérant l'interface WSDL de manière à l'utiliser (phase 4) une fois l'application cliente développée.

b. *Architecture*

- *Soap*

SOAP signifie *Simple Access Object Protocol* et caractérise le format de communication des services Web. C'est le protocole d'échange des messages, qui utilise lui-même le vocabulaire XML. Ce dernier est un standard, indépendant de toute plate-forme et de tout langage, ce qui le rend universel et permet de faire communiquer des applications à priori incompatibles. Par exemple, un client sur une plate-forme Microsoft peut communiquer avec des composants sur un serveur Unix. Qui plus est, SOAP encapsule les requêtes et les réponses sur le réseau et peut utiliser n'importe quel protocole de transport. En général, HTTP est utilisé car il bénéficie de beaucoup d'avantages (il passe par exemple sans problème au travers des firewalls, ce qui évite d'avoir à les configurer ou d'ouvrir des ports, et ainsi des brèches). Voici un schéma présentant une communication basée sur SOAP :



Dans le cas présent, le client formule des requêtes SOAP à destination du serveur, qui sont sérialisées en XML et enveloppées selon le protocole HTTP (à l'aide d'un package SOAP) et ensuite envoyées pour être interceptées par un « listener » (souvent une Servlet). Celui-ci va décoder le message SOAP avec l'aide d'un parser XML et invoquer une méthode du service, qui contient la logique de requête de la base de données, ceci grâce aussi à un package SOAP. La réponse sera elle-même encapsulée dans un message à destination du client. En ce qui concerne le format des messages SOAP, voici un exemple (global) de requête et de réponse reposant sur l'invocation d'une méthode :

Soit l'objet Personne :

```
Personne {
        String name ;
        int phone ;
}
```

Soit la méthode getPersonneFromId dont la signature est :

```
Personne getPersonneFromId ( int id ) ;
```

- *Requête :*

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope ... ENVELOPPE, ENTETE et règles d'encodage ...>
      <SOAP-ENV:Body>
            <ns1:getPersonneFromId
             xmlns:ns1="urn:MonServiceSOAP">
                  <param1 xsi:type="xsd:int">PARAMETRE1</param1>
            </ns1: getPersonneFromId>
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- *Réponse :*

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope ... ENVELOPPE, ENTETE et règles d'encodage ...>
    <SOAP-ENV:Body>
        <ns1:getEmployeeDetailsResponse
         xmlns:ns1="urn:MonServiceSOAP" règles d'encodage des types>
            <return xsi:type="ns1:Personne">
              <name xsi:type="xsd:string">name_value</name>
              <phone xsi:type="xsd:int">phone_value</phone>
            </return>
        </ns1:getEmployeeDetailsResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Comme on peut le constater, le contenu du message SOAP est lisible, ce qui facilite le déboguage pour les développeurs.

- *WSDL*

WSDL signifie *Web Services Description Language*. Il s'agit d'une description des services offerts par le composant applicatif, hébergée chez le serveur et codée aussi en XML. Cette description nécessite d'être connue lors du développement de l'application cliente pour savoir comment utiliser les fonctionnalités du service. Plus concrètement, WSDL définit de manière indépendante du langage de programmation dont est issu le service, une description des opérations et des messages qui peuvent être transmis vers et depuis celui-ci. Il décrit quatre types de données :

- Les fonctions publiques disponibles
- Les types de données pour les requêtes et les réponses
- Le protocole de transport utilisé
- Son adresse

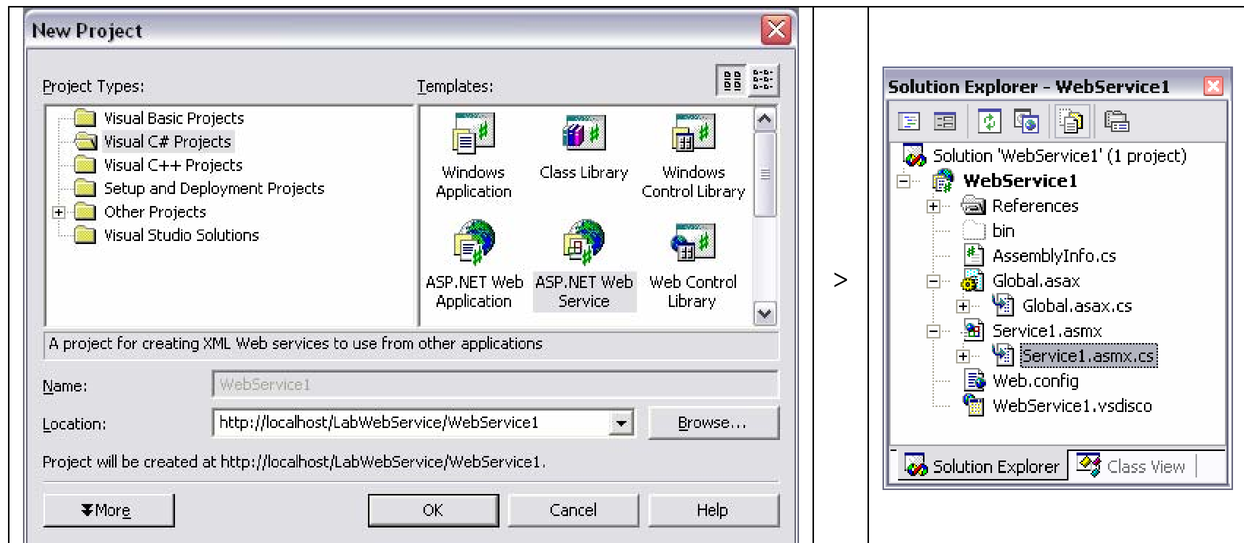
On peut aussi intégrer une documentation.
Le référencement du service consiste à donner l'URL de cette description, pour que les éventuels intéressés puissent la télécharger. Mais cela ne s'arrête pas là. En effet, ce concept implique une automatisation dans l'utilisation des services et donne naissance à des IDE (comme *Visual Studio .NET* ou *Web Matrix*) capables de les intégrer sans avoir à écrire une seule ligne de code.

c. Déploiement d'un service Web .net

Cette partie est un exemple de génération et d'utilisation d'un service Web avec un IDE, ici *Visual Studio .NET*. Dans un premier temps, il va être traité de la création d'un service basique en C#, de type « HelloWorld ». Autrement dit le client invoque une méthode HelloWorld() du service qui lui retourne une chaîne « HelloWorld ».

- Création du service

Sous *Visual Studio .NET*, on crée un projet C# (pour l'implémentation du service proprement dit) utilisant ASP.net comme « listener » pour intercepter les requêtes client et envoyer les réponses.



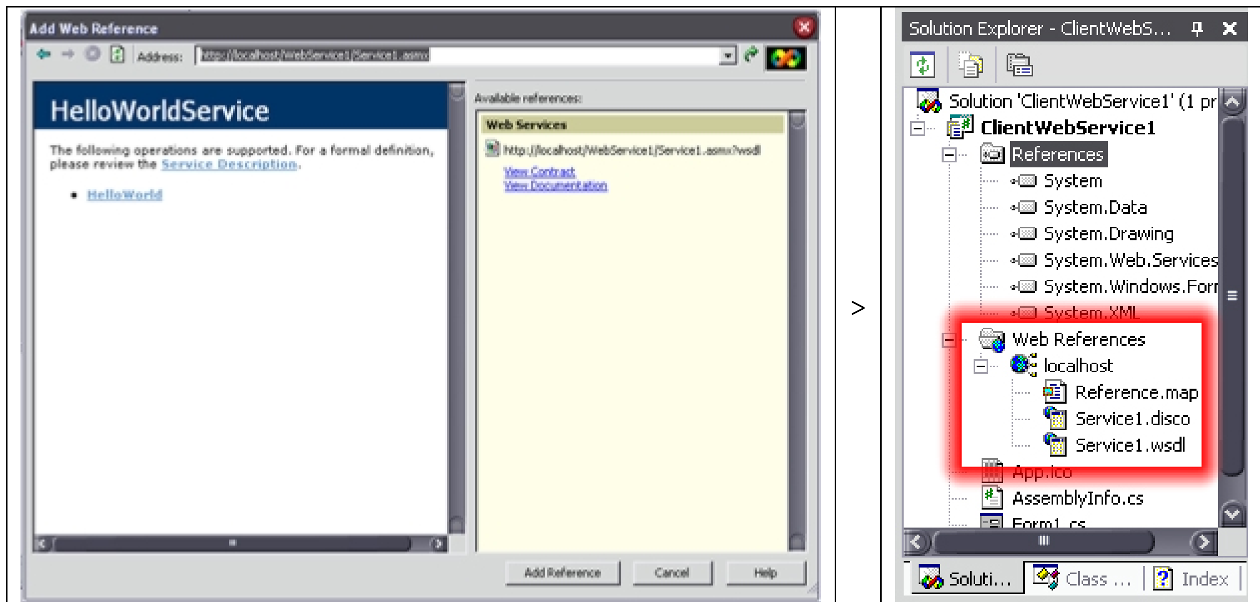
L'IDE génère quasiment tous les fichiers nécessaires à la publication du service Web, reste au programmeur à modifier l'implémentation du fichier source de celui-ci (en C#). Dans le cas d'une méthode *HelloWorld* :

```
[WebMethod]
public string HelloWorld()
{
        return "Hello World";
}
```

La méta donnée *[WebMethod]* permet d'indiquer que cette méthode est accessible par Internet. Après suivent la phase de compilation, qui va générer (sous Windows donc) un fichier DLL pour le service et le test, créé automatiquement par le framework .net, qui est en fait une page Web pour afficher des informations et l'invoquer. Les informations présentent éventuellement des erreurs commises mais aussi les messages qui seront échangés via les protocoles SOAP, HTTP GET et HTTP Post. Le test consiste à invoquer le service via le protocole HTTP GET, ce qui retourne comme résultat la chaîne « HelloWorld » dans le format XML. En outre, cette page donne accès à la description du service en WSDL. Il est possible de le compléter et le modifier via des attributs spécifiques au sein de l'implémentation du service (une description par exemple). L'exemple ici est très basique mais il montre bien à quel point créer un service Web peut être simplifiée par un IDE.

- Intégration du service dans une application cliente

La première étape dans l'utilisation d'un service est sa localisation. Dans *Visual Studio .NET*, on ajoute une « référence » Web à partir de l'URL du service :



Cette référence permet d'avoir accès au fichier WSDL mais aussi, et surtout, à la classe *HelloWorldService* (ici située localement) depuis le projet, et la manipuler. Voici l'extrait de code C# permettant d'invoquer la méthode *HelloWorld* à distance, « chaine » récupérant le résultat :

```
//Déclaration de la variable sur l'objet
localhost.HelloWorldService MonService;
//Intanciation de la variable
MonService = new localhost.HelloWorldService();
//Appel du Service Web
string chaine = MonService.HelloWorld();
```

Cet exemple a été réalisé sous *Visual Studio .NET* mais il existe aussi des outils gratuits permettant aussi aisément de faire cela, par exemple *WebMatrix*. Il a été réalisé localement en moins de dix minutes par l'auteur de ses lignes, sachant qu'il n'en avait jamais fait.

d. *Conclusion*

Les services Web constituent un des fers de lance du framework .net et sont basés sur deux concepts d'avenir, le développement par composants et le Web. Reposant sur des protocoles ouverts et standards (SOAP, HTTP, XML et WSDL en voie de standardisation par le consortium W3C), soutenu par un grand nombre de sociétés via des Toolkits (pour la package SOAP par exemple) ou des IDEs, ils pourraient faire évoluer considérablement le domaine des applications réparties voire même l'usage des réseaux par les entreprises …

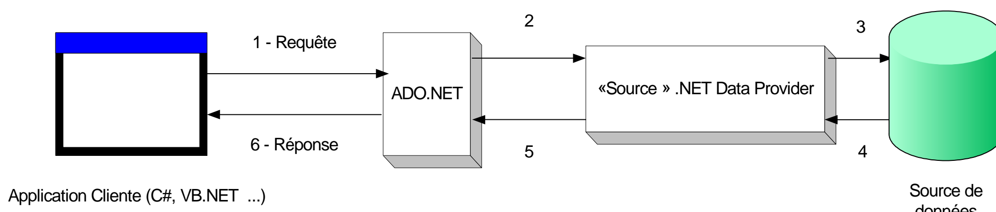
VII- ADO.net

ADO.net constitue un des frameworks les plus importants de .net. Il intervient au niveau de l'accès aux données (principalement aux bases de données) et est à la base de la persistance des composants dans .net. Nous allons voir dans cette section quel rôle il y joue, son historique car il découle de beaucoup de technologies élaborées bien avant .net et enfin de ses fonctionnalités, de la connexion à une base, à la manière de traiter des résultats.

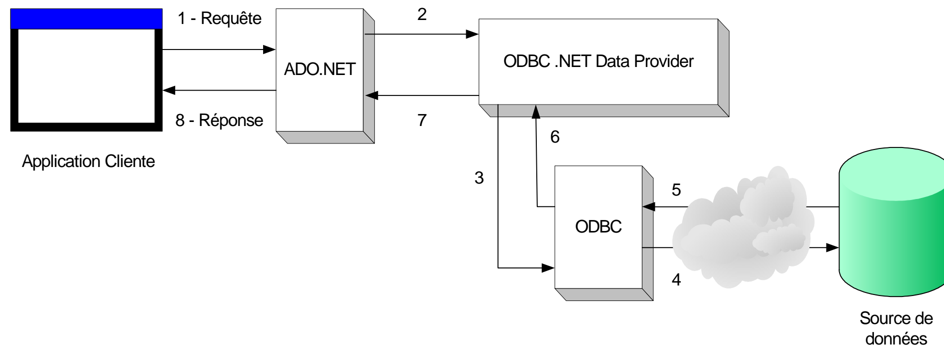
a. *Principe général*

ADO.net, comme son concurrent Java JDBC est une interface générique d'accès aux bases de données, mais aussi à d'autres structures tels que les feuilles de calcul d'Excel etc. Son rôle principal est de permettre au programmeur d'une application désirant y accéder de faire abstraction du système dans lequel elle fonctionne. Concrètement, cela signifie que quelque soit, par exemple, une base de donnée (*Access*, *MySQL*, *SQL Server*, etc.), l'implémentation reste (globalement) la même. ADO ne joue que le rôle d'interface. Il offre des méthodes de plus haut niveau au programmeur pour communiquer, mais il lui faut un « driver » de la source de données pour qu'il puisse s'y connecter et dialoguer. Ce driver s'appelle un « .NET Data Provider » et est spécifique de manière générale à la source (sauf s'il s'agit d'un driver « pont » vers un autre protocole d'accès, par exemple ODBC ou OLE DB, qui sont eux-mêmes génériques). Dans le framework .net, trois y sont inclus par défaut. Il s'agit de *SQL Server .NET Data Provider*, qui, comme son nom l'indique, permet de dialoguer avec le SGBD *Ms SQL Server*, de *OLE DB .NET Data Provider* qui se base sur le protocole OLE DB pour communiquer avec un grand nombre de bases, et de *ODBC .NET Data Provider* qui permet cela aussi mais via ODBC. Voici un schéma résumant tout ceci :

Cas n°1 : Accès direct



Cas n°2 : Exemple avec ODBC



Le cas 1 présente un accès « direct » à la source, et le cas 2 le passage par un pont

b. *Historique*

ADO.net n'est pas réellement nouveau car il est issu de plusieurs technologies, au sein de l'environnement Microsoft, dont l' « ancêtre » ODBC (*Open DataBase Connectivity*). Ce dernier permet de se connecter à des bases de données relationnelles avec pour même objectif de masquer son système de gestion au programmeur, et ce, à l'aide de drivers ODBC. Mais ODBC est lent, n'est pas portable (utilise du code natif via des DLL) et est difficilement utilisable avec Visual Basic. Ceci a donné naissance à RDO (*Remote Data Object*), qui est en fait une surcouche d'ODBC pour permettre à *Visual Basic* de se connecter à une base de données relationnelles. Mais il se limite à *Visual Basic*.
Ensuite est apparu DAO (*Data Access Object*) pour les développeurs en environnement C++, qui permet de se connecter aux bases *Access* ou à un pilote ODBC pour les autres, via un pont.
Plus proche d'ADO.net, OLE DB (*Object Linking and Embedding - DataBase*) est une uniformisation des accès, et supporte n'importe quel langage et source de données. Il ne se base plus sur les APIs citées précédemment car tout a été ré implémenté, ce qui implique des performances meilleures. OLE DB fonctionne avec des « providers OLE-DB » pour accéder aux sources. Dans le pire des cas, si il n'y en a pas pour une, il est possible d'utiliser un pont OLE DB/ODBC. Mais OLE DB est une API d'assez bas niveau.
C'est donc d'un besoin de simplification et d'abstraction qu'est apparu ADO (*ActiveX Data Object*), basé essentiellement sur des communications en mode connecté (connexion à la base, envoi des requêtes, traitements et ensuite déconnexion). ADO.net (précédemment appelé ADO+) se base lui surtout sur le mode déconnecté (les traitements se font après la déconnexion). Ceci sera détaillé ci-après.

c. *Fonctionnalités*

- *Objets centraux*

Nous nous limiterons ici aux bases de données relationnelles pour parler des fonctionnalités d'ADO.net et présenter des exemples. ADO.net repose sur deux objets centraux :

- Connection :
  Permet de définir à quelle source on souhaite s'adresser et d'ouvrir ou fermer une connexion.

- Command :
  Cet objet permet, comme son nom l'indique, d'envoyer des commandes vers la source. Cela peut être des requêtes SQL ou des invocations de procédures stockées dans la base. Un type lui est associé, pour déterminer quel type de requête va être envoyé et quel type de réponse va être retourné :

  - ExecuteReader : renvoie un *DataReader (cf. partie suivante)* (dans le cas d'une requête SQL de type SELECT)
  - ExecuteNonQuery : dans le cas d'une requête SQL de mise à jour de la base (INSERT, UPDATE, etc.).
  - ExecuteScalar : Retourne une valeur unique (SELECT Count(*) etc.)

Ces objets sont à la base de la connexion et des transactions avec la source de données. Pour résumer et illustrer ceci, voici un exemple de code VB.NET utilisant ces objets pour se connecter à une base de données (en mode connecté), provenant de diaporamas d'un séminaire Microsoft présentant ADO.net :

```
Imports System.Data.SqlClient
' Chaine définissant les paramètres nécessaires à la connexion à la base
' (serveur, base, login, password)
Public const CNXSTR = "server=(local);database=pubs;user ID=sa;pwd="

' Fonction se connectant, executant une requête et renvoyant le résultat
Public Shared function GetReader(requete as string) as SqlDataReader

      ' Déclarations d'objets
      Dim cnx As SqlConnection
      Dim cmd As SqlCommand
      Dim dr As SqlDataReader
      ' Constructeur de l'objet Connection
      cnx = New SqlConnection(CNXSTR);
      ' Etablissement de la connexion
      cnx.open()
      ' Définition de l'objet Command ç partir de la requête en paramètre
      cmd = New SqlCommand(requete, cnx)
      ' Execution de la requête
      dr = cmd.ExecuteReader(CommandBehavior.CloseConnection)
      ' Retour d'un DataReader (résultat en mode connecté)
      Return dr

End function
```

• *Modes connecté / déconnecté*

Les aspects les plus mis en avant d' ADO.net sont ses modes « connecté » et « déconnecté ». Il s'agit d'un concept très important qui permet de choisir si la déconnexion de la base doit avoir lieu après les traitements que l'on a à effectuer (auquel cas le résultat de la requête renvoie une sorte de « pointeur » sur la base permettant d'y accéder directement), ou avant, ce qui implique une sauvegarde d'une partie de la base dans un objet. A chaque mode correspond un objet : le DataReader et le DataSet. Selon le type d'application que l'on a à développer, l'un sera préférable à l'autre.

▪ DataReader

Un DataReader est le résultat d'une requête en mode connecté. Il permet de parcourir en lecture seule et séquentiellement les résultats renvoyés directement depuis la base de données. Les données sont transmises au fur et à mesure, ce qui suppose de maintenir la connexion ouverte.

▪ DataSet

Beaucoup des efforts portés sur ADO.net vont vers le mode déconnecté et l'objet DataSet. Celui-ci contient une vue de la base de données, tout en restant déconnecté. Le client n'a donc qu'à rapatrier cet objet, fermer la connexion avec le serveur et y travailler de son côté (voire le modifier et répercuter plus tard les modifications sur la base). Le DataSet contient des tables, des colonnes, des lignes, des vues, des contraintes et des relations. Son principal atout est qu'il est fortement typé, c'est-à-dire que sa classe est automatiquement générée en fonction de son contenu. Ainsi, ses propriétés seront calquées sur ses tables, ce qui a pour conséquence un code beaucoup plus lisible pour le développeur. Voici un exemple de code C# du parcours d'un DataSet :

```
// Création de l'objet Command
SqlCommand cmd = new SqlCommand();
cmd.CommandText = "SELECT Nom, Prenom, Age FROM TablePerso";
// Association de la connexion (cnx supposée initialisée)
cmd.Connection = cnx;
// Un DataAdapter permet d'exécuter des requêtes sur la base
// et de garnir un DataSet avec le résultat de ces requêtes
SqlDataAdapter da = new SqlDataAdapter();
da.SelectCommand = cmd;
// DataSet typé : toutes les tables et colonnes sont
// disponibles par la complétion automatique lors de l'édition !
TablePersoDataSet tpds = new TablePersoDataSet();
// On garnit le DataSet
da.Fill(tpds);
// On ferme la connexion
cnx.close();
// On parcourt et affiche
foreach (TablePersoDataSet.TablePersoRow row in
tpds.TablePersoDataSet.Rows) {
      Console.WriteLine("Age : " + row.Age);
}
```

Par ailleurs, le DataSet est sérialisable en XML, ce qui permet d'étendre ses possibilités au sein des applications distribuées. Il peut être aussi modifié, dans le but de voir ses modifications répercutées sur la base. Ceci se fait via un DataAdapter. Plus généralement, c'est ce dernier qui fait communiquer le DataSet et la base de données, dans un sens et dans l'autre, modélisés par deux méthodes :

- Fill, pour « remplir » le DataSet à partir de la base.
- Update, pour modifier la base à partir du DataSet.

- *Pool de connexions*

Les pools de connexion permettent de gérer les accès à des bases de données quand elles sont très sollicitées, de manière à éviter un crash au niveau du serveur. En effet, établir une connexion est un processus très coûteux. Le pool est un ensemble de connexions ouvertes en permanence. Son rôle est d'en fournir une au programme qui en fait la demande, mais en les gérant de manière à les « recycler », pour éviter d'en établir des nouvelles. Avec ADO.net, ceci se fait de manière transparente, sans que le développeur n'ait à s'en soucier.

d. Conclusion

ADO.net est l'interface d'accès aux données en environnement .net, au même titre que JDBC dans J2EE. Il apporte beaucoup de simplification dans l'implémentation des applications désirant se connecter à une source de données. Ce framework n'est d'ailleurs pas qu'une interface mais aussi des bibliothèques offrant des services permettant aux développeurs de mieux gérer les transactions entre clients et serveurs. Ces services reposent principalement sur deux objets, le DataReader en mode connecté et le DataSet en mode déconnecté. Ce dernier est d'ailleurs à la base d'un autre framework .net en cours de développement (à la date où le rapport est écrit) sur la sauvegarde d'objets persistants dans des bases relationnelles, les ObjectSpaces.

VIII- Conclusion

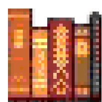
a. *But de Microsoft*

Depuis la sortie du framework .net, les habitudes de Microsoft ont bien changé. En effet, on entend parler d'open source, d'outils gratuits, de communauté, de normalisation, bref tout le contraire de ce à quoi nous avait habitué Microsoft.
On peut dire que Microsoft joue gros. A l'heure où les OS libres se développent, le géant de Redmond se lance dans une nouvelle technologie. Au niveau des chiffres, .net lui a coûté 3 Milliards de dollars, lui a pris 3 ans de développement, et 80 % des activités de la firme sont développées dans cet environnement. Mais pourquoi ?
Microsoft trouve ici un moyen de prendre une place prépondérante sur l'Internet, chose que la compagnie cherchait à faire de longue date sans avoir jusque là trouver le moyen d'y parvenir. De plus, cette technologie vient contrer son principal concurrent (SUN).
On peut regarder vers le futur et s'imaginer l'intérêt de Microsoft dans cette quête. Il y a fort à parier que Microsoft, au travers de ses nombreuses prises de position dans les sociétés d'édition de contenu multimédia, prendra à son compte la fourniture sous la forme de location ou d'abonnement de nombreux services Web. Cette idée vient se conforter quand on voit les autres éditeurs développer les mêmes services que Microsoft. En effet, IBM et plus récemment Oracle ont annoncé leurs offres permettant de créer des services Web. A terme, il est possible que l'intérêt de Microsoft sera de louer ses services de la même manière qu'il vend ses licences aujourd'hui. Cette stratégie est bien connue, on prend du vieux, on l'améliore et on le sort dans un bel emballage. En effet, Microsoft ne se cache pas d'avoir puisé de façon tout à fait pragmatique ses inspirations dans les technologies existantes, et en particulier Java.
Mais le risque est énorme. Certes Microsoft est à l'heure actuelle le leader mondial dans l'édition de logiciel, mais au niveau des services Web, tout le monde part du même niveau. Le fait de s'être investi autant peut à la fois conforter sa position de leader comme déclencher le début du déclin, ce qui fut le cas pour IBM qui n'a jamais réussi à reconquérir la place qu'il occupait à l'époque de la suprématie des ordinateurs centraux.
Qu'est ce que cela nous apporte et quel est notre intérêt :
Internet ne cesse de se développer. Le futur que l'on nous propose sera riche en communication. L'information sera partout. Aussi bien dans nos appareil ménagers que dans notre voiture, en passant par nos habits, et même nos animaux de compagnie (SONY). Le but premier de .net est de fournir aux développeur les moyens de créer des applications inter opérable utilisant des « Web Services » depuis tout type de terminal : PC, Pocket PC, téléphone, Tablet PC, Smart Display …

b. *Interêt pour les entreprises*

Tous les produits Microsoft vont l'intégrer de façon plus ou moins visible, depuis les serveurs de données (SQL Server) jusqu'aux environnements de développements (Visual Studio .NET) en passant par les SE (Longhorn, Server 2003), la technologie .net. Cela ne se limite d'ailleurs pas seulement à Microsoft, Borland par exemple, via la nouvelle mouture de Delphi va aussi l'intégrer.
L'une des forces de .net est la distribution, la communication entre systèmes distants grâce à XML. Un réel apport de .net est un gain de performances, sur une ancienne application, surtout si cette dernière a un besoin de passage à une architecture distribuée, comme la mise en place d'un intranet. Le coût de cette migration est minime. En effet, le langage C# par exemple est libre et doublement normalisé, il existe des environnements de développement gratuits (CSharpDevelop), et le fait que .net soit multi langage/plateforme implique une formation assez rapide (2 à 8 semaines). La seule difficulté pour l'entreprise est rencontrée lors de la migration de ses applications existantes.
Aujourd'hui, de nombreuses entreprises ont intégré .net. Ils y ont gagné en productivité (30% en moyenne) mais aussi en terme de sécurité. En effet, Microsoft a mis tout en œuvre pour que son environnement soit le plus sécurisé, ce qui semble primordial, vu le domaine d'application visé (Web Services, intranet …)
D'autre part, on peut dire que cet investissement est sans grand risque pour les entreprises. Aujourd'hui, de nombreuses entreprises possèdent une informatique vieillissante, basée sur des systèmes propriétaires et utilise des programmes dépourvus de maintenance, souvent du au fait que les entreprises n'existent plus. De plus les développeurs actuels ne sont plus formés à travailler sur ce type de machine. La conséquence première est que l'entreprise doit soit rechercher et engager des informaticiens qui, conscients de ce phénomène, deviennent plus exigeants, soit de changer leur système informatique, ce qui n'est pas toujours prévu dans le budget. .net permettrait d'éviter cela. La puissance économique de Microsoft assure, du fait qu'il mette tout en œuvre dans cette technologie, que dans dix ans cela existera encore. Au niveau du développement, .net donne l'opportunité d'objectif nouveau à moindre coût (Web Services, programmation par composants) avec une grande facilité d'intégration.
Conclusion : On peut imaginer que dans le futur, on montera notre programme à la carte. On demandera tel ou tel service, on les emboîtera et notre programme fonctionnera.

Nous avons vu à travers cette étude ce qu'étaient les composants et en quoi des frameworks tels que Java, Corba et enfin .net, sur lequel nous nous sommes focalisé, intervenaient.
La programmation par composants ne date pas d'hier. Pourtant elle était jusqu'alors peu utilisée. Avec l'importance grandissante des réseaux dans notre société et les implications que cela a au niveau logiciel, les nouvelles plates-formes de développement que sont .net ou J2EE permettent à cette forme de programmation d'émerger (ce modèle est en adéquation complète avec les architectures réseaux) et d'apporter au programmeur simplicité et gain (fiabilité, temps de développement etc.). Qui plus est, les services Web, fruits des efforts de Microsoft et d'IBM, garantissent l'interopérabilité des composants, ce qui leur promet encore un bel avenir.
En ce qui concerne .net et plus précisément C# qui en est son fer de lance, nous avons pu constater à travers les différents listings que ce langage a repris effectivement beaucoup d'éléments de Java, mais lui a apporté beaucoup de compléments qui font que C# lui est (actuellement) supérieur. Mais Sun (à l'heure où cette étude est faite), s'apprête à sortir une nouvelle version de Java qui pourrait changer la donne... jusqu'à ce que Microsoft ne ressorte à son tour une nouvelle version. Il est tout à fait envisageable que la confrontation Java/C# se résume à une partie de Ping Pong, l'un prenant le dessus sur l'autre temporairement.
Parallèlement aux langages, on peut constater que .net regroupe tous ce qui existe déjà, et il a pu être enrichi de nouvelles fonctionnalités grâce notamment, aux erreurs de ses concurrents mais aussi grâce à quelques ingéniosités. Néanmoins, beaucoup de services proposés initialement n'ont pas ou ont du mal à voir le jour. Ce démarrage en douceur, du sûrement à la crainte des entreprises, permettra peut être aux concurrents de rattraper leur retard. Au final, le seul gagnant dans l'histoire est le développeur qui verra le langage, et plus généralement la plate-forme qu'il a choisie évoluer plus vite ...



Bibliographie

LIFL http://www.lifl.fr/
- Java Beans
- Nouveaux défis des systèmes complexes

INSA-ROUEN http://asi.insa-rouen.fr/
- EJBs

LIFC http://lifc.univ-fcomte.fr/
- Conteneurs ouverts
- Service de Médiation pour les applications réparties à base de composants

Adele Team http://www-adele.imag.fr/
- Conteneurs ouverts

ONERA http://www.cert.fr/
- Les modèles à composants

LORIA http://www.loria.fr/
- Modèles par composant

INRIA http://www-sop.inria.fr/
- Architectures COM et ActiveX
- Programmation orientée composants

Site Perso http://pm95.free.fr/CrsArchiCom.htm
- Architecture COM

SELFHTML http://fr.selfhtml.org/
- Technologies ActiveX

ENST http://www.infres.enst.fr/
- Modèle de composants CORBA

Site Perso http://membres.lycos.fr/carinejammes/
- Présentation générale de CORBA

Soapuser http://www.soapuser.com/
- Architecture SOAP/XML

ESSI http://www.essi.fr/
- SOAP

Journal du NET http://solutions.journaldunet.com
- Comparaison J2EE/.NET

- Services et Composants J2EE
- WSDL
- Services Web
- Chat sur .NET

DevReference.net http://www.devreference.net/
- Architecture orientée Services Web

GUILDE http://www.guilde.asso.fr/
- L.D.A.P/Annuaires

Ashita http://www.ashita-studio.com/
- Introduction à J2EE

OSSIR http://www.ossir.org/
- Programmation sécurisée Java

INRIA-ALPES http://www.inrialpes.fr/
- Introduction à Java

DotNetGuru http://www.dotnetguru.org/
- ADO.NET vs. JDBC
- ObjectSpaces
- C# vs. Java
- Java Data Objects (JDO) vs. ObjectSpaces
- Déploiement d'un composant J2EE/.NET
- Les Assemblies

Ecole des Mines http://www.emn.fr/
- .NET dans le contexte des MDA

Builder.fr http://www.builder.fr/
- Article EJBs Entity

Psengineering http://www.psengineering.com/
- ADO.NET
- Services Web

CSharpHelp http://www.csharphelp.com/
- Programmation par composants dans .NET

DotNet-Fr http://www.dotnet-fr.org/
- Introduction à .NET
- Comparaison .NET/J2EE

Site Perso http://vmaviel.free.fr/
- Framework .NET

Developpez.com http://www.developpez.com/
- FAQ : Assemblies
- Architectures multi tiers

Bibliographie

EPFL http://sic.epfl.ch/

- J2EE vs. NET
- Interfaces utilisateurs J2EE

Microsoft http://www.microsoft.com/france

- Informations techniques .net
- Newsgroup MSDN

PWC http://www.programmationworld.com

- Cours .net/Java

Alain Vizzini http://alain.vizzini.free.fr

- Cours sur le C#