Département Informatique et Réseaux

Option Ingénierie Du Logiciel
Promotion 2001

Microsoft .NET :
Architecture et Services

Jean-François Bobier

Arnaud Fontaine et Sylvie Vignes

Juillet 2001

École Nationale Supérieure des Télécommunications - 46, rue Barrault - 75634 PARIS CEDEX 13

Microsoft .NET : architecture et services

Mémoire d'option IDL réalisé par **Jean-François Bobier**

Enseignants responsables : **Arnaud Fontaine** et **Sylvie Vignes**

Version du document : **1.01**

Résumé

Ce mémoire explore en profondeur le nouvel environnement de Microsoft nommé .NET. Les recherches effectuées s'appuient d'une part sur le .NET Framework SDK et Visual Studio.NET bêta 1, installés à cet effet sur une station Windows 2000 de l'ENST, et d'autre part sur les articles d'experts sur le Web et dans la presse écrite.

Dans une première partie, nous exposons la vision d'une nouvelle génération Internet et l'architecture que propose Microsoft pour la matérialiser. Des concepts centraux tels que le Common Language Runtime, le Common Language Specification, les assemblages, ASP.NET ou encore ADO.NET seront abordés.

Dans une deuxième partie, nous aborderons une comparaison entre .NET et J2EE, laquelle inclut le nouveau langage C# de Microsoft. Cette partie permettra aux développeurs Java de comprendre rapidement les innovations de .NET.

Dans une troisième partie, nous décrirons plus en détail les services Web, qui constituent une innovation centrale de .NET. Les services Web sont fondés sur SOAP, un protocole XML indépendant de toute architecture qui permet des appels distants de procédure simplement. Nous allons examiner SOAP, ses innovations et ses défauts, notamment concernant la sécurité. Ensuite, nous détaillerons l'implémentation de Microsoft et son intégration à .NET. Enfin, nous regarderons les alternatives libres disponibles, notamment Apache-SOAP, une implémentation libre pour Java.

Enfin, en conclusion, nous examinerons les profondes implications de la vision de Microsoft dans la vie de l'utilisateur, notamment les risques concernant les informations privées, le nouveau modèle de location du logiciel, un monopole de Microsoft dans les services Web avec HailStorm ou encore la dépendance forte envers une forte bande passante.

Abstract

This document explores the depths of the new framework devised by Microsoft called .NET. The research material come from direct manipulation of the .NET Framework SDK and Visual Studio.NET beta 1, that were made available at the university, and also numerous expert articles from the Web and the printed press.

First, the reader discovers the vision of a new Internet generation and the architecture that was accordingly designed. Items such as the Common Language Runtime, the Common Language Specification, assemblies, ASP.NET, ADO.NET will be looked at.

In a second part, an overview of .NET versus J2EE is attempted. A study of the new Microsoft language C# is included. This enables Java developers to quickly get in touch with .NET with much prior knowledge.

In a third part, the Web Services, a central innovation brought by .NET, are closely examined. They rely on SOAP which is an XML-based and architecture independent protocol designed to enable simple remote procedure calls. We'll first explore SOAP, especially what is new and which security concerns may arise. Then, Microsoft's nice implementation and it's integration to the framework is explored. Finally, we will look at open source alternatives, especially Apache-SOAP, a free Java implementation.

As a conclusion, we investigate the deep implications of Microsoft's vision in the user's everyday life, and concerns with privacy, the rental model, a possible new market monopoly with HailStorm and the reliance on high bandwidth.

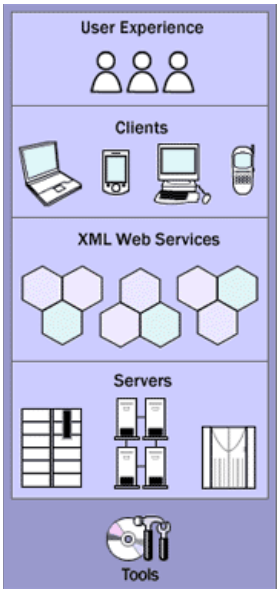
1 La plate-forme .NET

Depuis le 22 juin 2000, date de la première annonce de « l'initiative .NET » par Bill Gates, Internet foisonne d'articles à son sujet, et notamment MSDN (*Microsoft Developer Network*) à l'adresse msdn.microsoft.com/net. Il est rare d'obtenir tant de presse pour un produit dont la sortie, partielle au mieux dans Windows XP, n'est pas prévue avant le 25 octobre 2001[1] et le déploiement effectif quelques années plus tard selon le Gartner Group. Cette évangélisation n'est pas étrangère aux efforts colossaux déployés par Microsoft pour réussir une stratégie sur laquelle repose l'avenir de la compagnie.

.NET se présente comme une vision, celle de la prochaine génération d'Internet, fondée sur les standards ouverts et l'interopérabilité – domaines où la firme de Redmond n'a pas une réputation excellente[2]. Les cyniques diront que Microsoft s'inquiète de la monté en puissance irrésistible de Java sur les serveurs avec, à la clé, ni plus ni moins que la domination du Web entier. Suite à un litige avec Sun au sujet d'une extension apportée à Java, Microsoft s'était de lui-même exclu du mouvement Java, lequel a depuis cristallisé autour de lui de nombreux concurrents de poids dont IBM/Lotus et Oracle. L'annonce même de Bill Gates peut être considérée comme un « *vaporware* », un moyen de gagner du temps auprès des clients de Microsoft puisqu'à la rédaction de ce texte seule une première bêta de l'environnement et du nouveau Visual Studio est disponible.

Quoi que disent les opposants de Microsoft au sujet de .NET, la compagnie n'a pas l'habitude de perdre ses paris, aussi audacieux soient-ils que celui-là. Tout le personnel et les alliés déploient une grande énergie pour donner un corps à la vision de Bill Gates. Sans nul doute, .NET sera une plate-forme dont il faudra tenir compte dans un avenir proche.

1.1 Une nouvelle génération de services Internet



Dans la pagaille d'articles techniques, stratégiques ou commerciaux, rien que sur le site de MSDN, il est difficile de distinguer une réelle stratégie d'ensemble pour .NET. Est-ce un nouveau langage (C#) ? Est-ce un nouveau protocole de type DCOM ? Est-ce un ensemble d'outils destiné à faciliter le développement Web, comme avec Visual InterDev ?

La cohérence est difficile à discerner car Microsoft s'attaque simultanément à plusieurs fronts comme le montre la figure en face.

- L'expérience d'utilisation. .NET a pour ambition d'améliorer le confort de l'utilisateur.
- Les clients. .NET a pour vocation d'être omniprésent, ce qui implique qu'il ne se limite pas à Windows.
- Les services Web. Ce sont des applications mises en ligne sur

[1] Sortie officielle de Windows XP d'après 01 Informatique (http://www.01net.com/rdn?oid=147477&rub=1868)

[2] On se souviendra du procès antitrust où le juge Jackson a reconnu Microsoft coupable de pratiques monopolistiques, notamment en s'appuyant sur des standards propriétaires.

le Web grâce à un protocole ouvert fondé sur XML.

- Les serveurs. .NET s'appuie sur une offre logicielle Microsoft pour les serveurs comprenant Windows 2000, Windows XP, Internet Information Server, SQL Server 2000, Exchange…

- Les outils. Visual Studio.NET sera l'atelier unique de développement .NET. L'environnement lui-même est livré sous forme d'un kit de développement logiciel (SDK).

1.1.1 La vision : une nouvelle expérience d'utilisation

.NET se définit probablement mieux par ce qu'il compte offrir à son utilisateur, selon quatre principes :

- Le réseau Internet va se personnaliser en hébergeant pour chaque utilisateur les données de ses applications et ses préférences. Au lieu de posséder les logiciels comme c'est le cas aujourd'hui, ils seront loués et accédés en ligne.

- Le PC restera le médium favori de travail, mais l'utilisateur aura un accès permanent où qu'il soit à ses données et ses applications par le réseau, à partir de tout terminal.

- L'utilisateur pourra interagir avec ses applications d'une manière nouvelle, à l'aide notamment de la parole et de l'écriture manuscrite.

- Les frontières qui séparent les applications les unes des autres et de l'Internet vont disparaître. Au lieu d'interagir avec une application ou un seul site Web, l'utilisateur sera connecté à une « constellation » d'équipements et de services qui seront capables d'échanger et de combiner les objets et les données pour fournir les informations désirées.

Combinés, ces principes devraient offrir une expérience plus fluide de l'informatique aux utilisateurs. Avec .NET, il serait ainsi possible de faire interopérer bien plus étroitement tous les systèmes informatiques – grands serveurs, PC, terminaux mobiles…

La vision commence à prendre forme avec la nouvelle version de Office, connue sous le nom d'Office XP. Le site[3] propose déjà des composants .NET, nommés *eServices* qui permettent entre autres :

- Imprimer et poster le document à distance,

- Sauvegarder les documents sur le Web,

- Traduire les documents,

- …

[3] http://office.microsoft.com/default.aspx

Figure 1. Le portail eServices de Office XP

1.1.2 L'implémentation de .NET

Pour donner corps à la vision, Microsoft s'appuie sur un environnement appelé *.NET Framework* qui sera peut-être distribué gratuitement pour toutes les versions de Windows depuis Windows 95. Comparable en de nombreux points au *Java Runtime Environment* (JRE), l'environnement .NET comprend une machine virtuelle, un ramasse-miettes et une bibliothèque de classes riche. Connu pour sa force dans les outils de développement, Microsoft annonce le support de plus de 20 langages informatiques dans .NET. Cette prouesse est rendue possible d'une part par la définition de formats de type et de description standards, d'autre part par le code intermédiaire de la machine virtuelle.

L'environnement est conçu pour les serveurs, qui pourront mieux communiquer avec l'extérieur grâce à SOAP et XML, mais aussi à destination de nombreux terminaux. Les futures versions de Windows seront dotées de l'environnement par défaut et une version embarquée sera intégrée à des terminaux portables comme les téléphones ou les PocketPC. En outre, des tiers pourront communiquer avec les serveurs .NET grâce au protocole SOAP.

En s'appuyant sur le *.NET Framework*, les composants d'un logiciel comme Microsoft Word pourraient être localisés dans différents serveurs à travers le monde et manipulés par un grand nombre de terminaux. Cela aurait particulièrement un sens pour les terminaux mobiles à l'espace disque, la mémoire et au processeur trop faibles pour effectuer les traitements.

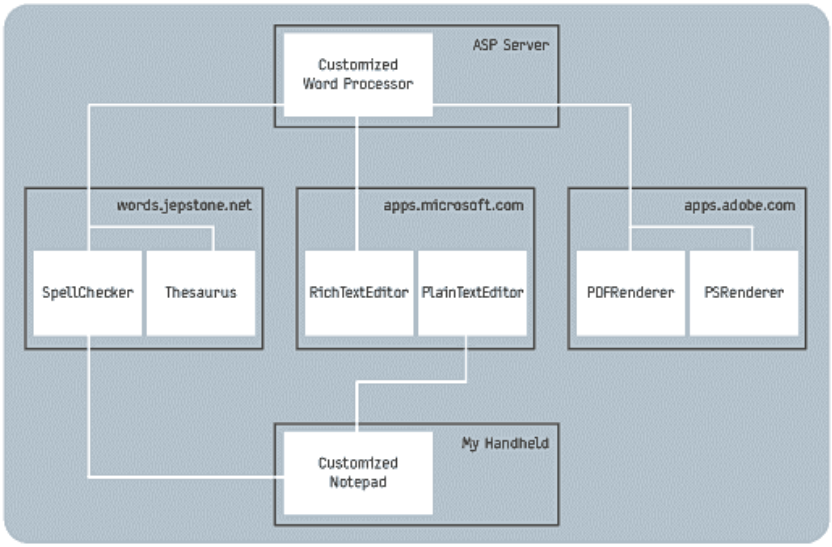


Figure 2. Un exemple fictif de distribution de composants (source : WebTechniques.com[4])

Cette ambition, selon le Gartner Group, pourrait mettre cinq avant de se concrétiser. L'avalanche de produits que Microsoft annonce s'orchestre autour de la vision et est axée autour de quatre standards Internet, dont deux auxquels la compagnie a beaucoup contribué :

- **HTTP**, pour le transport des données et la fourniture des applications sur l'Internet,
- **XML** (Extensible Markup Language), format d'échange des données neutre,
- **SOAP** (Simple Object Access Protocol), protocole qui permet de mettre en ligne des services très simplement,
- **UDDI** (Universal Description, Discovery and Integration), véritable répertoire ou pages jaunes des services SOAP.

1.2 L'architecture de .NET

A travers les différentes annonces de Microsoft depuis son lancement, les composants de .NET semblent s'organiser de la manière suivante :

- C#, un nouveau langage orienté objet destiné à faciliter la programmation dans .NET, notamment les composants, qui intègre des éléments de C, C++ et Java en apportant quelques innovations comme les méta-données.
- Un environnement d'exécution commun (*Common Language Runtime* - CLR) qui exécute un *bytecode* écrit dans langage intermédiaire (*Microsoft Intermediate Language* - MSIL ou IL). Du code et des objets écrits dans un langage quelconque peuvent être compilés en IL et exécutés par le CLR si un compilateur IL existe pour ce dernier.

[4] Article intitulé « Applying .Net to Web Services » - http://www.webtechniques.com/archives/2001/05/jepson

- Une grande bibliothèque de composants et d'objets de base accessibles par le CLR, qui fournissent les fondations pour écrire rapidement un programme (accès réseau, graphisme, accès aux bases de données).

- ASP.NET, une nouvelle version d'ASP (Active Server Pages) qui supporte une véritable compilation en IL, alors qu'ASP était interprété auparavant. On peut également écrire les pages ASP dans n'importe quel langage disposant d'un compilateur IL.

- Visual Studio.NET, une refonte de l'environnement Visual Studio et de Visual InterDev permettant aussi bien le développement d'applications et de composants classiques que Web.

- WinForms et WebForms, un ensemble de composants graphiques accessibles dans Visual Studio.NET.

- ADO.NET, une nouvelle génération de composants d'accès aux bases de données ADO qui utilise XML et SOAP pour l'échange de données.

- Un support des terminaux mobiles avec une version compacte de l'environnement .NET (limitée à Windows CE pour l'instant).

A l'aide de ces composants, .NET apporte trois grandes innovations aux utilisateurs de la plate-forme Windows :

- Les nombreux atouts de la plate-forme Java, à savoir la gestion automatique de la mémoire, la sécurité, une gestion unifiée des exceptions, une grande bibliothèque de classes, les vérifications de type lors des appels de fonctions.

- Un recentrage complet sur le Web. Pierre angulaire de la stratégie Internet de Microsoft, les Web Services constituent une nouvelle vision très pertinente des services distribués. Ils permettent en effet de mettre des objets en ligne très simplement, avec une API totalement neutre (architecture et système d'exploitation réellement indifférents) aussi facile à programmer que XML. Visual Studio.NET permet de concevoir des services ou des pages Web de la même manière que l'on concevrait une application classique.

- Le support d'un grand nombre de langages à l'aide du IL. Même si des compilateurs vers du *bytecode* JVM ont été réalisés pour Python (JPython), Cobol (PERCobol) ou Eiffel (Eiffel-To-Java VM), ces derniers ne permettent pas encore[5] d'accéder à l'ensemble des fonctionnalités de J2EE, cet environnement étant très fortement lié au langage. Le rouleau compresseur Java a ainsi marginalisé avec son succès croissant de nombreux langages qui, avant .NET, étaient menacés d'extinction.

On le voit, Microsoft joue la carte de l'ouverture avec .NET, stratégie jusqu'ici peu familière. L'environnement repose sur de nombreux standards ouverts, Microsoft a d'ailleurs directement contribué à l'élaboration de quelques-uns. De plus, des parties très importantes de

[5] Sun a réagit à la stratégie de Microsoft en proposant des compilateurs C# et Visual Basic vers JVM dans cet article du 5 avril de 2001 (http://www.01net.com/rdn?oid=144066&rub=1695)

l'environnement lui-même font actuellement l'objet d'une standardisation auprès de l'ECMA[6]. Cette ouverture offre la possibilité aux autres plates-formes de s'interfacer facilement.

Les composants s'articulent selon le schéma de la Figure 3.

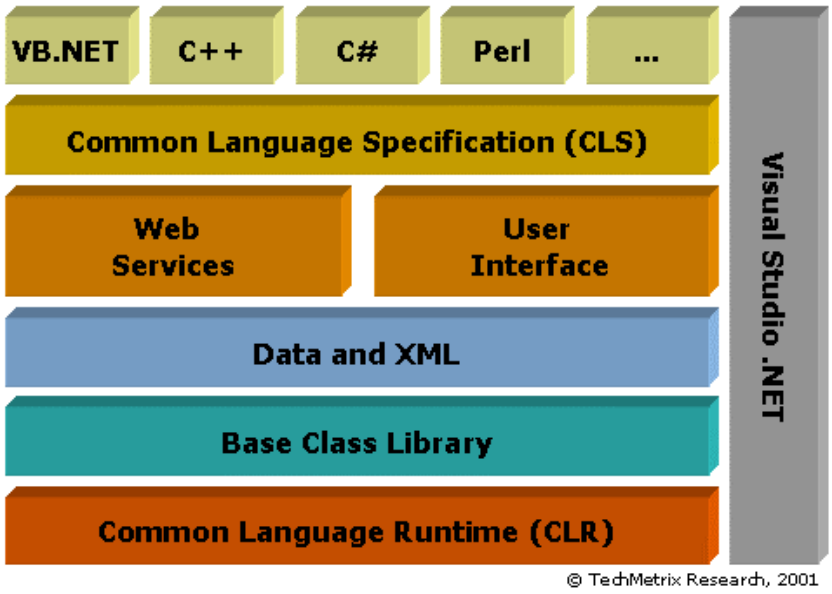


Figure 3. L'architecture fonctionnelle de .NET

1.2.1 L'environnement d'exécution commun

1.2.1.1 Common Language Runtime (CLR)

Microsoft, comme Sun, a choisi de se munir d'une machine virtuelle et d'un code intermédiaire ou *bytecode* nommé MSIL (*Microsoft Intermediate Language*). Si Java a pour son crédit vulgarisé le procédé, la première architecture de ce type avait vu le jour dans les années 70 avec le p-code de l'Université de Californie à San Diego. Les avantages sont considérables, mais il faut les payer au prix de performances amoindries, ce qui était plus dommageable en 1970 qu'aujourd'hui.

- Indépendance à l'architecture sous-jacente. Le p-code permet ainsi une grande portabilité, le fameux « *Write Once, Run Anywhere* » (WORA) de Java. Microsoft nomme cette capacité plus humblement « *execute on many platforms* » dans un document de MSDN[7], et pour cause, puisqu'il est annoncé que « Windows 95, Windows 98, Windows 98 SE, Windows Me, Windows NT® 4.0, Windows 2000 (tous service packs), Windows CE, et bientôt la version 64 bits de Windows 2000 » - nous pouvons presque résumer par Windows * - seront supportés par .NET. Microsoft semble plutôt tirer profit de l'indépendance au processeur - x86, ARM et bientôt

[6] European Computer Manufacturers Association (www.ecma.ch). Organisme célèbre pour avoir standardisé JavaScript (connu sous le nom de ECMAScript) et dont la licence est libre de tous droits.

[7] http://msdn.microsoft.com/msdnmag/issues/0900/Framework/print.asp

l'Itanium d'Intel – qu'une véritable indépendance à la plate-forme. L'article mentionne toutefois cette éventualité.

- Mécanisme de ramasse-miettes. Le code fonctionnant dans une machine virtuelle, cette dernière peut contrôler le cycle de vie des objets en libérant la mémoire occupée par un objet qui n'est plus référencé donc accessible par le programme.

- Sécurité du code. Puisque le code délègue à la machine virtuelle les accès véritables à la machine réelle, le programmeur ou l'utilisateur du code peuvent décider de donner des droits restreints à l'application. Ceci permet notamment de limiter les risques de débordement de buffer (*buffer overflow*) ou encore d'exécuter du code inconnu en toute confiance en restreignant l'accès au disque dur et au réseau du code (principe des *applets* Java).

Microsoft en tire d'autres avantages plus spécifiques à sa plate-forme et aux fonctionnalités du langage IL.

- Le support de nombreux langages. Si C# devrait s'arroger la part du lion des développements .NET, le langage intermédiaire a été écrit avec le respect des fonctionnalités de l'ensemble des langages connus, dont notamment un mécanisme commun de débuggage, la généricité, la surcharge d'opérateur, des gestions des exceptions ou la manipulation des pointeurs. Certains mécanismes ne sont pas présents dans le *bytecode* Java, qui est fait sur mesure pour le langage du même nom. Il reste évident que dans les deux cas, le seul langage vraiment reconnu est le p-code. Le support des autres langages peut être vu comme une facilité de syntaxe, mais on pourrait déjà dire cela des compilateurs actuels qui produisent de l'assembleur. Le tour de force de Microsoft est le concept de CLS (sous-ensemble commun aux langages, voir ci-dessous en page 18) qui permet à un objet Eiffel d'hériter d'un objet C# et inversement, en offrant notamment la possibilité de débugger l'objet de manière transparente.

- Prise en compte intégrée des versions (*run once, run always*). La notion d'assemblage (*assemblies*) équivalente aux paquetages de la machine virtuelle et du langage Java inclut un support des numéros de versions. Cela permettrait, pour emprunter un exemple à Linux, de mettre à jour la `glibc` sans casser les exécutables qui dépendaient de la précédente version, ce qui est appelé dans le monde Windows l'enfer du DLL (*DLL Hell*). Les composants .NET se « souviennent » de la version des composants partagés qu'ils utilisent, ce qui leur permet de fonctionner encore après une mise à jour. Ce comportement est configurable, il est toujours possible d'utiliser des versions plus récentes si elles sont compatibles.

- Sûreté de typage. Grand utilisateur de C++, Microsoft connaissait des difficultés liées au *casting* (transtypage) possible avec tout pointeur. On pouvait ainsi passer en paramètre des bibliothèques systèmes DLL (*Dynamic Linkable Library*) des pointeurs pointant vers des données incorrectement formatées voire une zone mémoire plus petite que celle attendue, créant des violations d'accès. A l'aide du *Common Type System* (CTS), le typage est sûr dans .NET, pour tous les langages l'utilisant rigoureusement, C# et Visual Basic.NET notamment. Microsoft a effectué un travail considérable pour unifier les types par valeur comme `int` et les objets, passés par référence à l'aide du concept de « *boxing* » et « *unboxing* ».

- Programmation simplifiée. Avec le temps et les nombreuses couches de sédimentation depuis MS-DOS, l'API Win32 est devenue, de l'aveu même de Microsoft, un véritable casse-tête pour les développeurs. La base de registre, les GUID de COM, les `Handle`, `Release` et autres `HRESULT` vont pouvoir devenir un mauvais souvenir du passé. De même, les messages d'erreurs étaient passés de manière inconsistante, tantôt par envoi d'exception, tantôt par code de retour à tester. Désormais, à la manière de Java, uniquement les exceptions sont utilisées pour gérer les erreurs.

1.2.1.2 Le langage intermédiaire MSIL

Microsoft a fait appel à de nombreux experts internationaux pour mettre au point un p-code très expressif et puissant. Le résultat est considéré comme une grande réussite et force est de constater que la rapidité d'exécution des programmes ne semble pas pâtir du surplus de travail pour le processeur.

Le langage IL étant particulièrement conçu pour les composants et commun aux différents langages cibles de .NET, il fournit des instructions que l'on ne trouverait normalement pas dans un code compilé. Des utilitaires nommés `ILAsm` (assembleur) et `ILDAsm` (désassembleur) permettent de travailler directement en langage IL. Nous nous intéresserons à `ILDAsm`, car il permet de découvrir le contenu d'un assemblage (fichier `.DLL` à ne pas confondre avec les anciens DLL Windows), comme ci-dessous où l'on examine le contenu de `mscorlib.dll`, qui contient des objets cœurs de .NET.

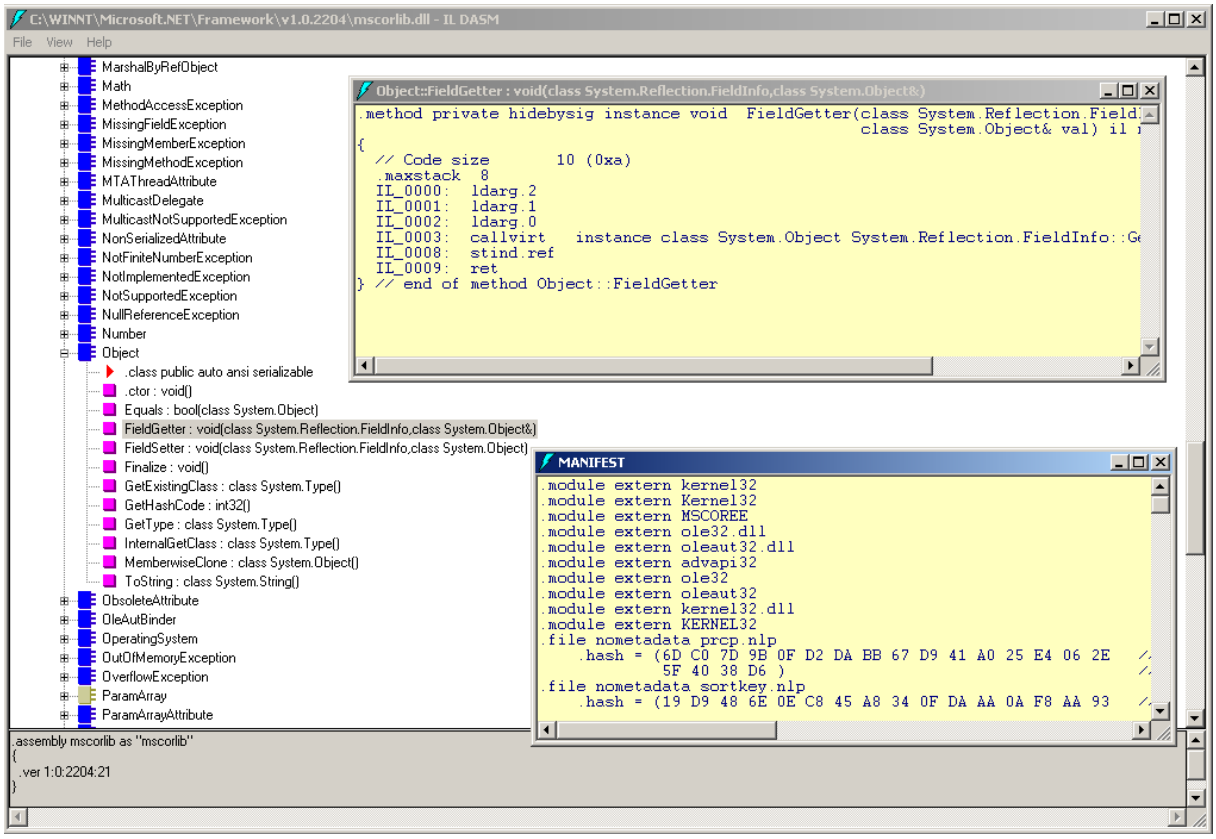


Figure 4. Le contenu de `mscorlib`

Les assemblages sont auto-descriptifs à l'aide des méta-données, ce qui permet au développeur d'utiliser directement `ILDAsm` s'il désire visualiser rapidement les classes d'un

assemblage. Nous pouvons également lire le code « source » IL qui est proche de l'assembleur tout en offrant des mécanismes plus évolués. Le code suivant le démontre avec un « Hello world » en IL :

```
.assembly hello {}
.assembly extern mscorlib {}
.method static public void main(  ) il managed {
       .entrypoint
       .maxstack 1
       ldstr    "Hello World"
       call     void [mscorlib]System.Console::WriteLine(class System.String)
       ret
}
```

Ce code se compile en exécutable `.EXE` à l'aide de `ILASM`, le pendant de `ILDASM`. Cet exécutable ne l'est que virtuellement, puisqu'il contient le p-code, indépendant du processeur et de la plate-forme, que nous avons évoqué en page 11. Les futures versions de Windows devraient être « IL-Ready », c'est-à-dire reconnaître une extension `.IL` ou `.ASM` et exécuter le p-code directement. A la différence de Java, ce p-code n'est jamais interprété mais est toujours compilé en code natif par un compilateur Just-In-Time (JIT). Selon Microsoft, .NET n'utilise ni machine virtuelle ni interpréteur mais un environnement d'exécution contrôlé (*managed runtime environment*). Les termes peuvent différer, mais les deux architectures gèrent l'exécution du code intermédiaire, inspectent la mémoire et surveillent les violations d'accès. Il semblerait néanmoins que la nuance soit justifiée par les mécanismes mis en œuvre par Microsoft pour optimiser les performances de son p-code. Trois compilateurs JIT sont ou seront fournis :

- Le compilateur JIT par défaut. Ce dernier effectue des compilations paresseuses en code natif à chaque fois qu'une portion de code est appelée par le programme. Les souches de code IL sont alors progressivement remplacées par du code natif, ce qui permet d'offrir des performances remarquables, comparables à du code natif avec un ramasse-miettes. Toujours pour améliorer les performances, le ramasse-miettes est également paresseux au sens où il attend que l'occupation mémoire justifie un nettoyage pour effectuer une collecte des objets inutiles. Tous ces principes se retrouvent dans la plate-forme Amiga actuellement commercialisée depuis deux ans, laquelle fonctionne aussi bien sous Linux que Windows et est reconnue pour ses remarquables performances. Au cours des tests effectués sur le SDK, les applications C# semblaient aussi rapides que leurs pendants classiques (compilation statique), à la différence de Java dont la dégradation de performances est toujours sensible malgré les améliorations apportées au JDK 1.3.

- Le compilateur EconoJIT. Ce compilateur est destiné aux terminaux mobiles dont la limitation principale est la mémoire vive. Il fonctionne comme le compilateur par défaut, à ceci près qu'il fourni un code moins optimisé et peut, si la mémoire devient trop faible, éliminer des portions de code qu'il a compilé (*code pitching*). Les portions sont recompilées à la demande, en utilisant la souche IL.

- Le compilateur OptJIT. Prévu dans une version future de l'environnement, ce compilateur laissera la possibilité à des tierces parties d'écrire un sur-ensemble de IL appelé OptIL (*Optimized IL*). OptIL est un IL avec des instructions supplémentaires pour indiquer à OptJIT comment optimiser certaines portions de code. L'intérêt du OptJIT est donc de fournir des optimisations liées à certaines tâches, comme les calculs mathématiques lourds mais les applications multimédia.

La compilation JIT est résumée dans la Figure 5.

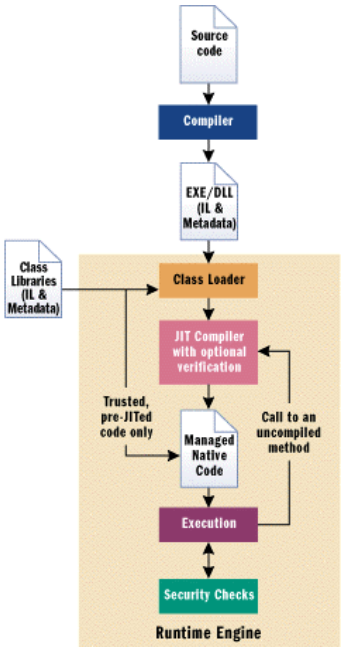


Figure 5. Le processus de compilation et d'exécution (source MSDN)

Précisons qu'un mécanisme nommé PreJIT permet de compiler entièrement le code IL en code natif, ce qui est notamment le cas pour la librairie standard du SDK. On le voit, l'optimisation des performances a été au centre des préoccupations de Microsoft.

1.2.1.3 Les assemblages (assemblies)

Un objectif majeur de .NET est la fourniture simplifiée de composants. Ce concept dépasse la simple notion d'objet pour offrir véritablement au programmeur une boite noire, pas nécessairement située ou exécutée sur la machine l'utilisant, contenant des outils réutilisables.

Même en boucle locale, les assemblages fournissent plus de services que leur équivalent standard Java, les *Java Archives* (JAR) – les *Entreprise JavaBeans* (EJB) étant par ailleurs restreints au monde des serveurs d'application, bien qu'ils se déploient sous forme de JAR. Certes, le premier rôle d'un assemblage, comme les JAR, est de rassembler tous les fichiers et classes nécessaires à la fourniture d'un ensemble cohérent de fonctionnalités dans un fichier `.DLL`. Il faut prendre garde, l'extension est trompeuse car ces fichiers n'ont plus rien de commun avec les anciens fichiers DLL Windows.

Une grande innovation est la notion de manifeste (*Manifest*) – certes, les JAR ont un fichier du même nom dans le sous-répertoire `meta-inf/`, mais son rôle n'est en rien comparable avec celui de Microsoft, bien que dans le cas des EJB1.1 des fichiers XML de déploiement peuvent le compléter. Le manifeste d'un assemblage définit ses propriétés comme son numéro de version, sa langue (*culture*), la liste des modules, fichiers et assemblages externes dont il dépend, numéro de version compris. La plupart des attributs, par exemple `[WebMethod]` (voir page 45), impactent directement le manifeste sans influencer le code IL généré car ils affectent le déploiement et la mise en œuvre de l'application plutôt que sa logique.

1.2.1.3.1 Déploiement des assemblages

Pour résoudre le problème de l'enfer du DLL (*DLL Hell*), les assemblages ont deux méthodes de déploiement et doivent suivre une politique stricte de version.

La première méthode consiste à effectuer un déploiement privé. De fait, Microsoft, après l'avoir imposé de longues années, admet que son système d'installation et de désinstallation de logiciels n'est pas bon. Les utilisateurs de Windows savent d'expérience qu'installer un logiciel sous Windows peut mettre à jour des DLL partagés susceptibles de casser certains logiciels installés ; la désinstallation laisse quant à elle souvent une trace indélébile dans le système (la base de registres, les répertoires, bibliothèques DLL laissées à l'abandon ou écrasées, etc.), ce qui le ralentit peu à peu jusqu'à le rendre inutilisable. Concrètement, Microsoft souhaite revenir au système simple « une application occupe un répertoire » et, par suite, imposer aux bibliothèques non partagées d'être situées dans le répertoire de l'application. Dans la terminologie .NET, il s'agit d'assemblages privés (*private assemblies*) dont la création et le déploiement sont voulus moins contraignants que de persister dans l'ancien paradigme.

Malgré ces bons sentiments, Microsoft ne pouvait pas oublier la raison d'être des bibliothèques partagées, à savoir l'économie de mémoire vive et d'espace disque – d'autant plus que dans un système réparti .NET, il est possible d'utiliser des bibliothèques partagées distantes. Toutefois, pour pallier l'enfer du DLL, .NET impose aux fournisseurs d'assemblages partagés (*shared assemblies*) de leur donner des numéros de version stricts et de signer leurs assemblages à l'aide d'une clé cryptographique. Les numéros de version permettent à une application de toujours fonctionner après mise à jour des bibliothèques, puisqu'elle connaît le numéro de version de la bibliothèque avec lequel elle a été testée. Les signatures permettent d'éviter qu'un tiers malveillant ne substitue une version détournée d'une bibliothèque, mais aussi de connaître l'auteur d'une bibliothèque partagée dont on n'aurait plus besoin.

1.2.1.3.2 Versions et cryptographie

Les clés utilisées pour signer les versions sont libres de droits, à la différence des certificats pour signer les applets Java, limités dans le temps et payables auprès de sociétés privées comme VeriSign. Pour générer une clé privée, un utilitaire nommé *strong name* (`sn.exe`) est fourni avec le SDK. Une compagnie peut décider de se doter d'une clé globale pour tous ses produits ou d'en générer autant qu'elle crée de composants distincts. Il est ainsi vivement conseillé de la conserver en lieu sûr pour éviter certains désagréments. L'usage est très simple :

```
sn.exe –k maCle.snk
```

Un attribut noté `[assembly:AssemblyKeyFileAttribute("maCle.snk")]` permet de signer un assemblage en C#.

Un numéro de version sous la forme majeur, mineur et *build* peut être donné à l'aide de l'attribut `[assembly:AssemblyVersionAttribute("majeur.mineur.build")]`. L'usage d'un astérisque (`*`) permet la génération automatique d'un numéro selon sa position. A la place de *build*, par exemple `1.0.*`, l'astérisque est remplacé par le nombre de jours écoulés depuis le premier janvier 2000 ; située après *build*, par exemple `1.0.0.*`, l'astérisque désigne la révision, c'est-à-dire le nombre de secondes écoulées depuis minuit.

Utilisés conjointement, ces attributs permettent la création des assemblages partagés (*shared assemblies*). Le système les enregistre à l'installation dans le cache global des assemblages (*Global Assembly Cache* – GAC) situé dans `C:\WINNT\Assembly`, en vérifiant, si une nouvelle version est ajoutée, que la signature cryptographique est bien compatible avec les versions précédentes.

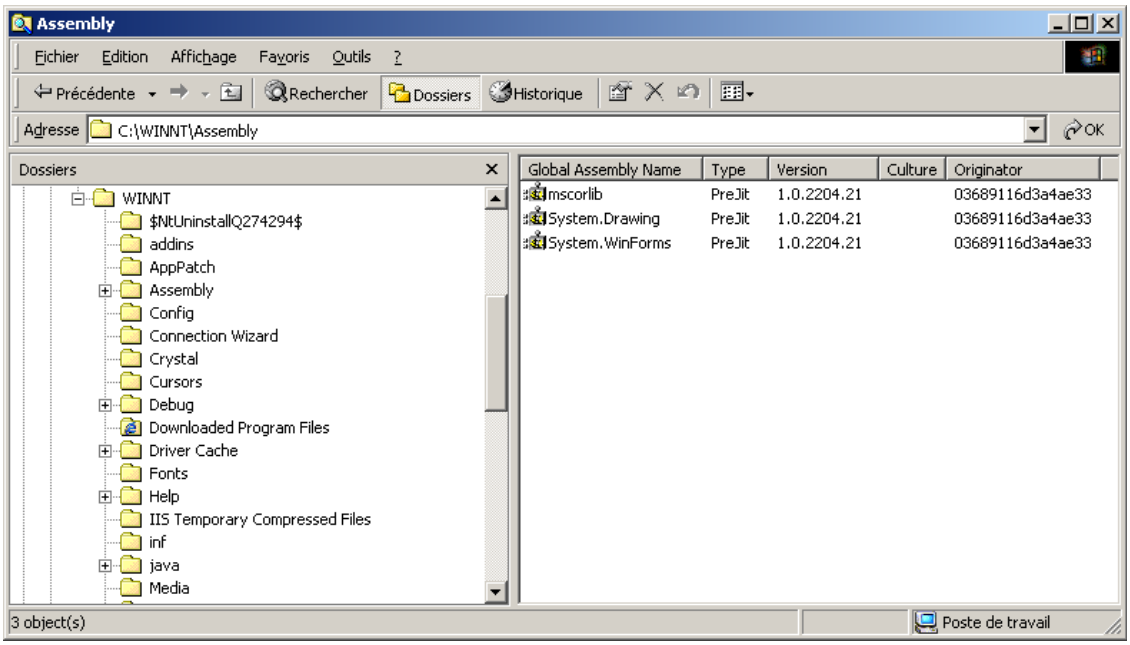


Figure 6. Global assembly cache (GAC)

Dans la figure ci-dessus, on peut constater que le GAC maintient, outre le numéro de version et le signataire (*Originator*), la langue de l'assemblage (*Culture*) et son type. Comme nous l'avions écrit, les assemblages partagés sont précompilés (PreJIT) pour optimiser les performances de l'environnement. A la différence de la version testée, la version bêta 2 devrait être localisée.

1.2.1.3.3 Architecture distribuée

Des utilitaires nommés `RegAsm.exe` (*Assembly Registration Tool*) et `TlbExp.exe` (*Assembly to Type Library Conversion*) permettent d'exposer un assemblage à COM+, rendant accessibles les programmes .NET aux applications Windows classiques.

Une autre manière de distribuer les assemblages est SOAP. Pour cela, il faut déployer l'assemblage dans le serveur IIS, en n'oubliant pas de préciser les méthodes exportées à l'aide de `[WebMethod]` (voir « Les Web Services et .NET » en page 39).

Enfin, Microsoft propose un remplacement de COM+ nommé « .NET remoting » qui permet la distribution de composants .NET, avec des mécanismes de ramasse-miettes, de références distantes, de sécurité…

1.2.1.4 Portabilité

Pour l'instant, la seule plate-forme .NET disponible est Windows dans ses nombreuses déclinaisons depuis Windows 95, Windows CE comprise. La question de savoir si un portage

vers UNIX verra le jour mérite d'être posée, et la réponse dépendra probablement du contexte d'ici à quelques années[8].

Les spécifications de .NET que Microsoft a soumis à l'ECMA sont modélisées en Figure 7.

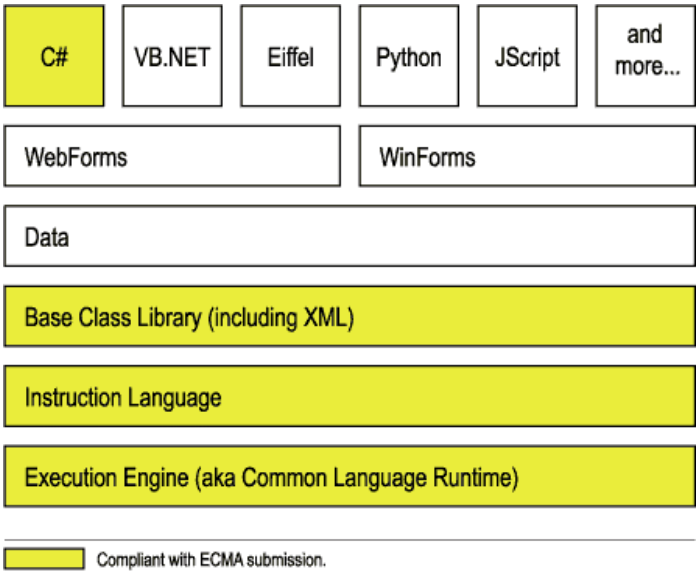


Figure 7. Les spécifications soumises à l'ECMA (source : WebTechniques.com[9])

Selon Brian Jepson, auteur de l'article dont est tiré cette figure, si Microsoft ne porte pas .NET lui-même sur d'autres architectures, la concurrence pourrait très bien le faire sur la base de ce qui a été soumis à l'ECMA.

1.2.2 Common Language Specification (CLS)

Le CLS est la solution mise en œuvre dans .NET pour permettre l'interopérabilité entre les langages. Microsoft emploie parfois le terme moins élogieux de *Common Language Subset* (sous-ensemble commun aux langages), ce qui nous semble plus approprié pour désigner le CLS.

Le premier obstacle, déjà rencontré en CORBA et en COM, est l'équivalence des types (*mapping*). Le CLS impose l'emploi d'un sous-ensemble du *Common Type System* (CTS) pour permettre un premier niveau d'interopérabilité – chaque langage devra, comme en CORBA, trouver ou construire un équivalent valable dans sa syntaxe et son système de type propres. En outre, il est bien évident que certains langages ont des spécificités propres, comme l'héritage multiple en C++ et en Eiffel, la surcharge d'opérateur en C++ et C# ou encore un système de typage automatique comme Objective CAML ; par ailleurs certains langages ne sont pas orientés objet, comme C. On le voit, ces caractéristiques sont souvent incompatibles, notamment dans le cadre de l'héritage des classes – une illustration est l'impossibilité d'utiliser l'héritage multiple dans un langage ne supportant que l'héritage

[8] Le CEO de Microsoft, Steve Ballmer, confirme une implémentation de .NET pour Linux dans l'article « Microsoft to unveil .Net software for Linux » (http://www.computerworld.com/cwi/story/0,1199,NAV47_STO58536,00.html).

[9] Article intitulé « Applying .Net to Web Services » - http://www.webtechniques.com/archives/2001/05/jepson

simple. Or, la gageure de Microsoft consiste à permettre à tous les langages une intégration poussée de toutes les classes, indépendamment du langage dans lequel elles sont écrites, et également d'utiliser la même librairie de base (BCL) de manière transparente.

La difficulté ne réside pas dans l'écriture d'un compilateur C++ vers IL supportant l'héritage multiple. Techniquement, il serait probablement plus facile à écrire en IL, qui a des constructions évoluées, qu'en assembleur, dont l'expressivité est très pauvre, mais le modèle d'héritage multiple choisi en IL n'aurait aucune signification pour les autres langages.

A la lumière de ces obstacles, on peut considérer le CLS comme le plus petit commun dénominateur entre les langages orientés objet. Ceci force l'emploi de mécanismes communs dans le système de types ou dans l'approche orientée objet. De fait, le CLS exclut les langages procéduraux comme C, mais également l'héritage multiple ou encore le typage automatique. Inversement, le CLS autorise parfois des fonctionnalités inexistantes dans certains langages que ces derniers doivent émuler.

Microsoft définit 41 règles nommées CLS Rules[10] édictant les propriétés que doivent respecter les langages CLS, selon une classification ternaire :

- Consommateurs (*CLS consumers*). Ces langages ont un grand degré de liberté puisqu'ils sont simplement tenus de savoir utiliser des objets ou composants du CLS.
- Amplificateurs (*CLS extenders*). Ces langages sont consommateurs, mais peuvent également exporter des classes et des assemblages à destination de tous les consommateurs.
- Environnements (*CLS frameworks*). Ces règles s'appliquent pour la conception d'environnements CLS. Ces environnements (bibliothèques) sont conçus pour être utilisées pour un grand nombre de langages, y compris les consommateurs et les amplificateurs.

Sans entrer dans les détails techniques, les 41 règles ont pour objectif d'assurer une interopérabilité satisfaisante mais minimale. Ceci implique notamment le support d'Unicode, la gestion des références, la résolution des conflits de nommage, les opérations interdites…

Prenons le cas d'Eiffel, que Bertrand Meyer a porté sur .NET sous le nom de Eiffel#[11] suite à un accord signé avec Bill Gates en 1999. Eiffel est un langage orienté objet qui a apporté en 1985 des fonctionnalités inédites comme l'héritage multiple, la programmation contractuelle, un système de typage unifié, la covariance[12], une grande bibliothèque de classes de base, la gestion des exceptions ou la généricité. De ces innovations, Eiffel# ne garde plus que le système de typage unifié (en fait celui de CTS), la bibliothèque (celle de .NET), la gestion des

[10] Ces règles sont édictées dans la documentation du Framework SDK à la rubrique « .NET Framework Developer Specifications » -> « Technical Overview of the Common Language Runtime » -> « Common Language Specification ».

[11] Prononcer « Eiffel sharp », spécifications du langage disponibles sur www.eiffel.com .

[12] Lorsque cela est possible, Eiffel autorise la redéfinition du type des arguments ou des valeurs de retour des méthodes ou attributs hérités. Ainsi, un attribut entier dans la classe mère peut devenir une fonction sans arguments rendant un résultat entier dans la classe fille.

exceptions (qui avait été déjà reprise d'Eiffel par C++ et Java), la généricité et la programmation contractuelle. De plus, Eiffel# comme Eiffel ne supporte pas la surcharge de méthode, c'est-à-dire que deux méthodes ne peuvent pas avoir le même nom même si leur signature (types des arguments) est différente alors que le CLS l'autorise (règle 6). Un outil nommé *Emitter* transforme les noms des méthodes d'une classe héritée du CLS pour la rendre compatible avec Eiffel#. Ainsi, les méthodes surchargées C# suivantes :

```
public static void WriteLine (String format, Object arg0);
public static void WriteLine (int value);
public static void WriteLine (String value);
```

sont traduites par l'*Emitter* en :

```
WriteLine_System_String_System_Object (format: STRING; arg0: ANY)
WriteLine_System_Int32 (value: INTEGER)
WriteLine_System_String (value: STRING)
```

De même, une équivalence des types Eiffel a été mise en place, ce qui a été plus facile car les types Eiffel de base ont chacun un pair natif CTS sans adaptation. Ainsi, comme on peut le constater, le type Eiffel `ANY` trouve son équivalent CTS en `System.Object`. Pour plus d'informations au sujet d'Eiffel#, nous recommandons la lecture de « Eiffel on the Web » sur MSDN[13].

Pour permettre une utilisation transparente de tous les langages, la CLS impose des limitations drastiques à chacun, y compris les langages créés directement par Microsoft comme C# et Visual Basic. Microsoft a d'ailleurs refondu le langage Visual Basic en Visual Basic.NET, un langage réellement orienté objet, à la différence de son prédécesseur. Ceci a un coût ; certains développeurs favorables à l'objet ont applaudi, les autres jugent que la complexité est devenue inacceptable, pour un langage qui se voulait extrêmement simple aux origines, et affublent la nouvelle création de Microsoft de « VB.NOT » ou « Visual Fred ». Enfin, comme nous l'avons évoqué, C++ en tant que tel n'est pas conforme au CLR, ce qui a amené Microsoft à proposer « C++ with managed extensions », une variante de C++ ne supportant pas l'héritage multiple et utilisant le ramasse-miettes avec le mot clé `_gc`.

Fort heureusement, les limitations deviennent obligatoires dès lors que l'on désire « exporter » une classe vers les autres langages. On peut donc utiliser les constructions internes aux langages dans des classes cachées, ce qui rétablit l'intérêt du support multilingue de .NET. En C# et Managed C++, l'attribut `[CLSCompliant(true)]` force le compilateur à vérifier que le code IL généré est compatible avec le CLS. On peut imposer la vérification à l'assemblage entier avec `[assembly:CLSCompliant(true)]` ou simplement se restreindre à un type ou objet.

Ainsi, la fonction C++ `public int SetValue(unsigned int value)` sera rejetée car le type `unsigned int` ne fait pas partie du CLS. De même, une fonction `public int SetValue(MyType value)` ne sera acceptée que si `MyType` possède un attribut `[CLSCompliant(true)]`. On voit que, par récursivité, tout type exposé comme compatible doit être composé de sous-types compatibles. La cohabitation des langages peut ainsi devenir un véritable casse-tête pour le programmeur, ce qui a peut être amené Microsoft à commettre

[13] http://msdn.microsoft.com/library/techart/pdc_eiffel.htm

un lapsus révélateur dans la documentation du SDK bêta 1 en écrivant dans un exemple de code l'attribut sous le nom `[CLSComplaintAttribute(false)]`[14].

1.2.3 Les briques de base

1.2.3.1 Base Class Library

Microsoft a bien compris l'intérêt de la grande bibliothèque unifiée de Java, car elle accélère les développements tout en les gardant compatibles entre eux. Ainsi, C++ en manquait cruellement, malgré les tentatives de Microsoft d'y remédier partiellement à l'aide des Microsoft Foundation Classes (MFC).

.NET fournit à tous les langages une grande bibliothèque de classes dont on peut voir un extrait ci-dessous.

Namespace	Description	Exemple de classes
`System`	Types de bases et la Console	`Object, Buffer, Byte, Char, Array, Int32, Exception, GC, String`
`System.Collections`	Collections d'objets	`ArrayList, BitArray, Dictionary, Hashtable, Queue, SortedList, Stack`
`System.Data`	Interaction avec les bases de données	`DataBinding, DataRelation, DataRow, DataSet, DataTable, DataSource`
`System.Globalization`	Types pour l'internationalisation (National Language Support - NLS), comparaison des chaînes de caractères et des calendriers	`Calendar, CultureInfo, JulianCalendar, NumberFormatInfo, NumberStyles, RegionInfo`
`System.IO`	Accès aux fichiers	`ByteStream, File, FileStream, MemoryStream,Path, StreamReader, StreamWriter`
`System.Net`	Accès au réseau	`WebRequest, WebResponse, TcpClient, TcpListener, UdpClient, Sockets`
`System.Reflection`	Inspection des méta-données	`Assembly, ConstructorInfo, FieldInfo, MemberInfo, MethodInfo, Module, ParameterInfo`
`System.Runtime.Remoting`	Types pour la gestion d'objets distants	`ChannelServices, RemotingServices, IMessage, ImessageSink`
`System.Security`	Types pour la gestion de la sécurité	`Permissions, Policy, Principal, Util, Cryptography`
`System.Web.UI.WebControls`	Composants Web	`AdRotator, BorderStyle, DataGrid, HyperLink, ListBox, Panel, RadioButton, Table`
`System.WinForms`	Composants graphiques (widgets) classiques	`Button, CheckBox, DataGrid, FileDialog, Form, ListBox, MainMenu, MonthCalendar, NewFontDialog, RichEdit, ToolBarTreeView`

[14] *To complain* signifie se plaindre en Anglais alors que *compliant* se traduit par conforme.

Innovation déterminante sur Java, l'espace de nommage `System.Xml` offre une variété impressionnante de composants et d'attributs spécialisés dans la gestion du XML.

1.2.3.2 Building-block services

Avec .NET, Microsoft souhaite vendre et louer des composants majeurs génériques pour la construction d'applications Web complexes. La compagnie inaugure la nouvelle génération d'Internet basée sur la vente et la location de petites parties d'applications, lesquelles pouvant être hébergées sur un serveur distant, comme Microsoft Passport, ou local, à l'aide d'un serveur Intranet Microsoft.

Les composants cœurs de ces « building block services » vont inclure :

- **Identification**. Au cœur de la stratégie de location de Microsoft fondée sur la facturation de l'abonné, Passport offre un service intégré d'authentification modulaire pouvant supporter les mots de passes jusqu'aux cartes à puce en passant par les empreintes digitales.

- **Notification et messagerie**. Ce service, qui utilisera probablement le portail MSN où il existe déjà, intègre la messagerie instantanée, le mél, fax, répondeur téléphonique et autres formes de notification et de messagerie.

- **Personnalisation**. Application directe d'un des quatre principes fondateurs, ce service permet à l'utilisateur de l'application de créer des règles et des préférences de gestion des messages et des données, lesquelles seront mémorisées par un serveur distant.

- **Entrepôt XML**. Ce service est un entrepôt de données (*data warehouse*) accessible par SOAP.

- **Calendrier**. C'est un agenda sécurisé et accessible par de nombreux terminaux mais aussi des services Web accrédités par l'utilisateur.

- **Répertoire et Recherche**. C'est un annuaire pour les services et les personnes destinées aux utilisateurs.

- **Livraison dynamique**. Ce service permet des mises à jour des services et d'applications transparentes pour les utilisateurs, quel que soit le terminal d'accès.

Microsoft a mis l'accent sur l'interopérabilité forte de ces services, fondée sur XML. On peut voir dans ces services une tentative de diversifier les revenus de Microsoft en donnant la part belle à la location d'applications ou de morceaux d'applications.

1.2.4 ASP.NET et les WebForms

Microsoft ASP.net est une nouvelle version d'ASP, un langage interprété produisant des pages HTML et WML très comparable à PHP en de nombreux points. La mise à jour introduit une vraie rupture dans le paradigme d'ASP avec l'ajout de deux nouveaux concepts comme la compilation des pages et les WebForms.

Comme les Java Server Pages (JSP), le code ASP.NET est toujours compilé (en IL puis code natif avec le JIT) et peut être écrit dans un langage de haut niveau. ASP.NET supporte tous

les langages du CLR, ce qui inclut C# et VB.NET. D'après une mesure effectuée par c2i[15], ASP.NET serait en bêta 1 déjà 44 fois rapide qu'ASP, grâce à la compilation et aux performances du compilateur JIT de Microsoft.

Les WebForms sont une couche d'abstraction ajoutée pour permettre une programmation composite d'interface homme-machine orientée Web. Des composants génériques tels les formulaires, tableaux, boutons et zones de textes peuvent être assemblés afin de générer les pages ASP.NET. S'ils sont utilisés avec soin (aucun ajout de code HTML en dur par exemple), ces composants nommés WebForms répondent à deux problématiques des développeurs Web :

- L'absence de véritable plate-forme HTML. Les différents navigateurs du marché, essentiellement Internet Explorer et Netscape Navigator dans leurs différentes versions, ne supportent pas HTML de la même manière. Même si l'on fait exception des balises propriétaires comme `<IFRAME>` de Internet Explorer, les balises de HTML 4.0 ne sont pas rendues de la même manière. Pire encore, les supports de JavaScript et de CSS[16] évoluent fortement au cours de versions. De fait, les développeurs Web exigeants doivent écrire des pages en fonction du type et de la version du navigateur afin d'offrir une expérience uniforme. Ceci s'effectue souvent à l'aide d'un JavaScript[17] inséré en entête de la page, mais peut être résolu en redirigeant les requêtes HTTP en fonction du `user agent` vers des URL (ex `/msie4/`) adaptées. Les WebForms peuvent apporter une solution plus simple pour le développeur grâce à la couche d'abstraction.

- *Pervasive computing*, c'est-à-dire l'explosion des moyens d'accès au Web. En effet, en remplaçant les différentes widgets de HTML, TinyHTML et WML voire VoiceXML, les WebForms permettent à ASP.NET de reconnaître à l'exécution le terminal et/ou du navigateur client afin d'utiliser un équivalent natif se rapprochant le plus possible de l'effet voulu. Cette couche d'abstraction constitue une approche différente du paradigme de l'Internet mobile actuel basé sur le transcodage XML+XSL, très coûteux en performances (Oracle Portal-To-Go et IBM WebSphere Transcoding Publisher).

1.2.5 Visual Studio.NET

Microsoft VisualStudio.net est l'outil de référence des développements .NET. Il a été fusionné avec Visual InterDev, l'outil de conception de pages HTML dynamiques. Si l'interface n'est pas révolutionnaire en soi, force est de constater que Visual Studio.NET tient les promesses de .NET envers les développeurs Web. Nous examinons de plus près ses innovations en leur faveur en page 39.

---

[15] http://www.c2i.fr/asp/code.asp?IDCode=152

[16] Cascading Style Sheets. Cette spécification du W3C permet un rendu plus uniforme entre les navigateurs tout en facilitant l'utilisation d'une charte graphique pour le site grâce à la notion de style. Malheureusement, le support est très partiel dans Netscape 4.x, dernière version réellement stable de Netscape.

[17] Par exemple, le script « `if (navigator.userAgent.indexOf("MSIE 4") != -1)` » vérifie si le navigateur est Internet Explorer 4.

Le développement d'applications classiques Windows a été radicalement simplifié, en prenant le meilleur de Delphi et de Java, ce qui n'est pas étonnant si l'on sait qu'Anders Heljsberg[18] a grandement influencé .NET.

UML apparaît dans un modeleur intégré à Visual Studio.NET. Pour l'instant, ses fonctionnalités nous ont semblées limitées ou peu abordables, mais sa présence indique la volonté de Microsoft d'intégrer le processus de conception dans l'environnement de développement.

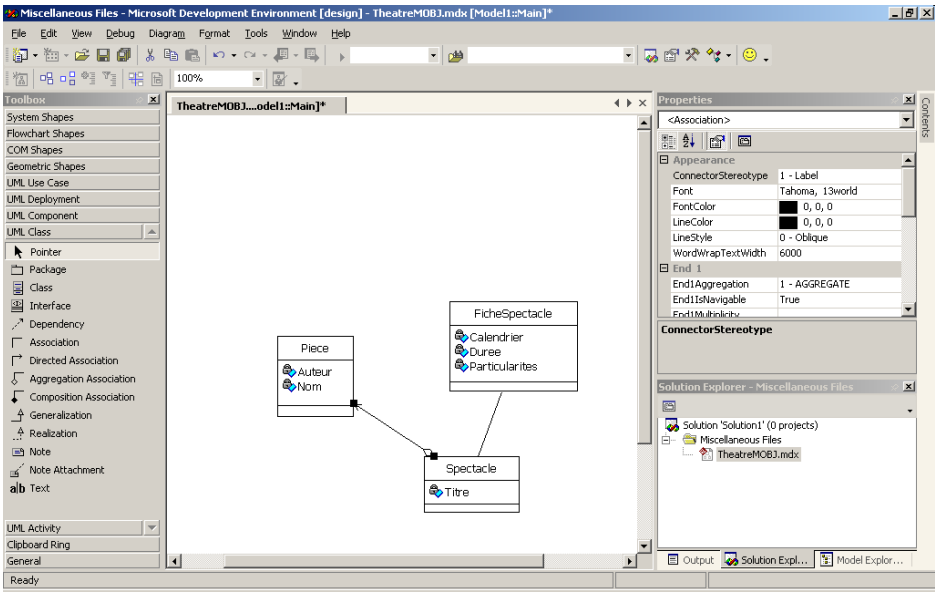


Figure 8. Le modèleur UML intégré de Visual Studio.NET

On le voit, cette nouvelle version de Visual Studio apporte réellement de nouvelles fonctionnalités intéressantes pour les développeurs Windows. Développer sous Windows n'a jamais été aussi facile, en particulier les applications Web que l'on peut écrire et déployer vraiment rapidement.

1.2.6 Les services Web et ADO.NET

1.2.6.1 Les services Web

Déterminants pour la réussite de .NET, les services Web sont une petite révolution dans l'informatique - petite, car rien n'est vraiment nouveau ni même original, mais une révolution dans la simplicité de son approche, ce qui est rafraîchissant dans un domaine réputé complexe.

Ils sont appelés à jouer un rôle important, et pas uniquement lié à la plate-forme de Microsoft puisque tous les acteurs du marché préparent un support complet (IBM WebSphere, SunONE…). Ceci explique pourquoi nous leur avons dédié un chapitre entier de ce mémoire à la page 34.

[18] Concepteur de Delphi pour Borland et de Visual J++ pour Microsoft, il est présenté en page 26 au sujet de C# qu'il a également conçu.

Un élément important de .NET appelé à être utilisé au sein des services Web est ADO.NET, un modèle orienté réseau pour les bases de données, qui permettra des prouesses comme la consultation et la mise à jour par HTTP et XML.

1.2.6.2 Les bases données en ligne : ADO.NET

Conformément à l'esprit de .NET, cette évolution d'ADO (ActiveX Data Objects) met XML au cœur des bases de données qu'elles soient relationnelles ou pas. L'innovation centrale d'ADO.NET est le `DataSet`, un modèle XML déconnecté des données. Un objet `DataSet` peut ainsi effectuer des requêtes sur la base et traduire les résultats en XML. Le grand intérêt réside dans le fait que les manipulations ultérieures sur le `DataSet` s'effectuent sans connexion à la base. L'API est très riche, elle permet notamment d'exprimer des contraintes, des relations entre tables, des insertions, des mises à jour… L'objet garde une trace de son état initial, ce qui permet d'effectuer ensuite des mises à jour « propres » de la base de données une fois les manipulations terminées. Le modèle de concurrence employé est dit « optimiste » au sens qu'il présuppose que peu de modifications concurrentes ont lieu.

L'objectif d'ADO.NET est à proprement parler de libérer les données en leur permettant d'être manipulées et échangées sur le Web avec une grande simplicité comme le montre la figure suivante.

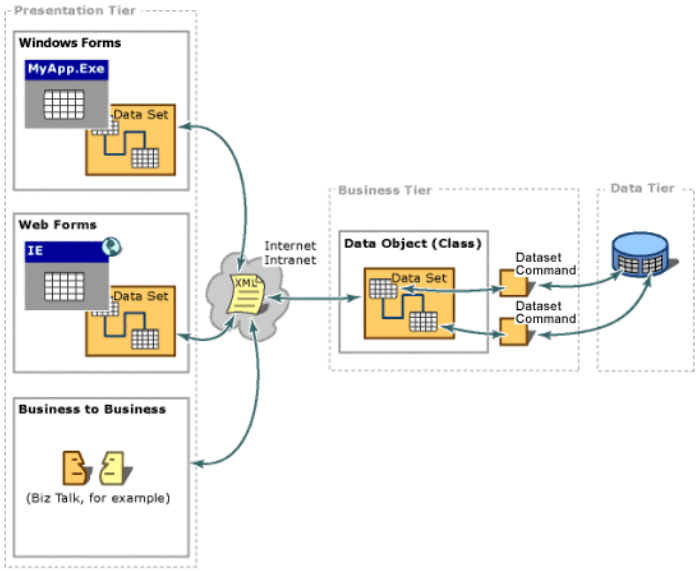


Figure 9. Les bases de données en ligne grâce à XML et HTTP (source MSDN)

2 Une comparaison entre .NET et J2EE

2.1 Le langage C# : un clone de Java ?

a été spécifiquement développé pour l'environnement .NET afin de le doter d'un langage facile à utiliser et très expressif. Il facilite ainsi la programmation par composants et a le potentiel d'attirer les développeurs vers .NET car ses fonctionnalités et sa syntaxe ont été voulues agréables et modernes. Malgré la disponibilité prochaine de vingt autres langages, il nous semble évident que le langage privilégié de développements de services sera C#, tant ses nombreuses innovations méritent réellement le détour et s'intègrent bien aux besoins de l'environnement.

Notons que le langage C#[19] et le langage intermédiaire commun MSIL généré par le compilateur vont probablement entrer dans le domaine public puisqu'ils ont été soumis à l'ECMA pour permettre une standardisation ouverte sans royalties. Nous pouvons ainsi espérer à terme disposer d'un compilateur et d'un environnement d'exécution C# libres pour UNIX, pourquoi pas *gcsc* (GNU C Sharp Compiler) ?

C# a été conçu par Anders Hejlsberg, le concepteur de Turbo Pascal puis de Delphi pour Borland et de Visual J++ pour Microsoft. Dans un entretien donné par la maison d'édition O'Reilly[20], Hejlsberg explique que, pour créer C#, il s'est inspiré principalement de C++, mais a regardé également tous les autres langages, dont Java. Certes, le discours officiel de Microsoft consiste à affirmer que C# est un descendant de C++, mais force est de constater que de nombreuses innovations introduites par Java ont été reprises, souvent en respectant la syntaxe d'origine.

2.1.1 Les points communs

- Le compilateur C#, comme Java, produit un p-code (*bytecode*) MSIL, Microsoft Intermediate Language, destiné à fonctionner dans une machine virtuelle (*managed execution environment*).
- La syntaxe élégante et économique est empruntée à C et C++.
- Présence d'un ramasse-miettes (*garbage collector*) et la possibilité de limiter les accès du code à certaines ressources dynamiquement couplés à l'élimination des pointeurs de C++ (toutefois, C# permet l'usage des pointeurs à plusieurs occasions).
- Grandes capacités d'introspection de classe (*reflection*).
- Pas de fichiers d'entête, tout le code est restreint à des paquetages ou assemblages, élimination des problèmes de dépendance circulaire dans la déclaration des classes.
- Tous les objets descendent de `Object` (`System.Object` en C#) et sont tous alloués sur le tas avec l'opérateur `new`.

[19] Prononcer « C Sharp ».

[20] Voir le document « *Deep Inside C#* » où Anders Hejlsberg répond aux questions de l'éditeur O'Reilly à l'adresse http://windows.oreilly.com/news/hejlsberg_0800.html

- L'héritage des classes est simple (C++ supporte un héritage multiple minimaliste), mais plusieurs interfaces peuvent être implémentées par une classe.
- Support natif des processus légers (*threads*) en posant un verrou sur les objets entrant dans du code synchronisé.
- Classes intérieures (*inner classes*).
- Les tableaux et les chaînes de caractères sont incluses dans le langage et font l'objet de vérifications automatiques de dépassement.
- L'opérateur « . » est toujours utilisé, plus d'opérateurs `->` ou `::` utilisés par C++.
- Le système d'exception est généralisé et les blocs `try` ont une clause `finally`.
- Toutes les variables sont initialisées avant usage et les accès aux tableaux (*arrays*) sont systématiquement testés pour éviter les débordements.
- Le mot clé `if` attend un paramètre booléen. En C et C++, ceci permettait parfois d'économiser astucieusement quelques cycles processeur mais causait parfois des erreurs, notamment avec du code du type `if (i=1)`.
- Les commentaires inclus dans le fichier source sont utilisés pour générer la documentation des classes (*JavaDoc* pour Java). Le compilateur C# génère un fichier XML qui peut être mis en forme à l'aide de feuilles de style XSL.

2.1.2 Les innovations de C#

- Les attributs. Ils permettent d'ajouter des méta-données aux composants, aux classes et à leurs membres. Ces informations renseignent le déploiement de la classe ou de l'assemblage de telle sorte qu'elles n'impactent pas nécessairement le code généré, mais son comportement à l'exécution ou encore des vérifications effectuées à la compilation. Cette innovation de Microsoft a pour but de faciliter la programmation par composants qui nécessitait avant C# des descripteurs séparés du code source, et va au-delà en répondant à d'autres besoins.
- Les événements. Chaque classe peut recevoir ou exporter des événements. Cette fonctionnalité ressemble à ce que le toolkit graphique Qt de TrollTech permet de faire en C++. Java, avec le toolkit Swing, offre une fonctionnalité similaire, mais bien moins intuitive. Les événements s'enregistrent très simplement à l'aide de l'opérateur `+=`.
- Système de type unifié. A la différence de Java, les types primitifs comme `int` et `float` héritent de `Object`, ce qui évite l'encapsulation parfois inévitable en Java par les classes comme `Int` et `Float`. Ces encapsulations sont gérées automatiquement par le mécanisme de « *boxing* » et « *unboxing* » en C#.
- Les fonctions déléguées (*delegates*). Elles sont un équivalent au typage sûr des références aux fonctions de C ou de C++. Cette fonctionnalité avait d'ailleurs été au cœur du désaccord entre Sun et Microsoft au sujet de Java.

- Les propriétés (*properties*). Elles contribuent à grandement simplifier la syntaxe en encapsulant les accesseurs (les `get` et `set` parfois désagréables de Java) dans la déclaration d'un membre.

- Les structures (*struct*). Héritées également de C++, elles sont similaires aux classes, sans héritage, et sont allouées sur la pile. Elles contiennent directement leurs valeurs (*value type*) à la différence des classes classiques dont on ne manipule en réalité que le pointeur vers une instance. L'usage des structures permet d'améliorer la performance du code produit.

- Les types énumérés (*enum*). Restaurés de C++, ils permettent d'associer des noms avec des valeurs numériques, ce qui évite les `static final int FIELD` de Java.

- La surcharge d'opérateur (*operator overloading*). Héritée de C++, elle simplifie également la syntaxe pour la manipulation des objets. Notons que Java l'avait repris, mais pour la classe `String` uniquement.

- La généricité (*templates*). C# reprend le principe des *templates* de C++. Pour rendre cela possible, le code MSIL n'a pas, comme Java, de fonction `AddInt` ou `AddFloat` mais une fonction `Add` générique (*type neutral*) qui est remplacée par du code natif approprié en fonction du contexte. Malheureusement, le support complet de la généricité n'est pas encore implémenté dans la bêta 1 du .NET SDK, mais est promis.

- Les méthodes virtuelles. Comme en C++, les méthodes ne sont pas virtuelles par défaut, c'est-à-dire qu'il faut déclarer une méthode `virtual` dans la classe mère et ajouter le mot clé `override` dans la méthode redéfinie pour pouvoir surcharger – c'est exactement le contraire en Java. Outre un gain faible en performance, ceci permet de voir clairement quelles méthodes peuvent et sont redéfinies.

- Les indexers. Les objets implémentant l'interface `IEnumerable` peuvent voir leur contenu parcouru par le mot clé `foreach`, ce qui simplifie également la syntaxe.

- Préprocesseur. Vécu comme un manque dans Java par de nombreux programmeurs C et C++, il est présent dans C#.

- Syntaxe de passage des paramètres riche. Les mots clés `in`, `out`, `ref` et `params` permettent un interfaçage aisé avec les autres langages.

- Les espaces de nommage (*namespaces*). Les paquetages (*packages*) de Java qui étaient très fortement liés à une structure sous forme de répertoire sont remplacés par une notion plus souple d'espace de nommage.

2.1.3 Le choc des philosophies

Si, bien sûr, C# a repris de nombreux éléments novateurs de Java à son compte, la volonté de Microsoft d'écrire plus qu'un simple clone amélioré de Java semble réellement sincère. Les philosophies et racines des deux langages sont différentes.

- Java hérite de la famille SmallTalk où la présence d'un interpréteur de code est indispensable. Les programmeurs de C/C++ et de Java savent que Java n'a réellement emprunté de C/C++ que la syntaxe claire et économique. A l'inverse de C++, Java se veut d'une simplicité et d'une rigueur extrêmes. La liberté du programmeur est

fortement restreinte précisément pour éviter les écueils de C++ en excluant toute manipulation considérée comme incompatible avec le paradigme objet. Ce paradigme est central dans Java, tel qu'il a été exprimé par James Gosling, « tout est objet ». Au nom de cette doctrine, la performance et la simplicité du code Java ont parfois été sacrifiées car l'approche objet ne soufre aucune exception, sauf peut-être les types primitifs tels que `int`.

- C# emprunte à Java, certes, mais hérite également de C++, ce dernier étant d'ailleurs le langage de développement interne de Microsoft pour l'ensemble de ses logiciels, Windows et Office notamment. A la différence de Java, C++ et C# sont pragmatiques au regard du paradigme orienté objet, leur objectif étant de fournir un langage le plus performant possible à l'écriture du code comme à son exécution pour des applications industrielles - a contrario, Java est très apprécié dans le monde universitaire pour ses qualités pédagogiques. Si le *bytecode* de la machine virtuelle Java a été conçu pour fonctionner dans un interpréteur, même s'il existe des compilateurs capables de le transformer en code natif[21], le langage intermédiaire de Microsoft (MSIL) a été conçu pour être compilé en code natif et c'est d'ailleurs son seul mode de fonctionnement possible. Cette compilation a lieu pendant l'exécution, un peu à la manière des compilateurs Just-In-Time (JIT) pour Java, mais elle est effectuée de manière astucieuse, de telle sorte qu'une procédure compilée l'est définitivement (sa référence dans l'environnement pointe vers le code compilé). La richesse et la puissance des fonctionnalités de C# peuvent faire penser à la maxime de Perl, « there's more than one way to do it »[22], tant la liberté de programmation, que Java restreint parfois excessivement, a été restituée tout en gardant ce qui avait assuré le succès de ce dernier. Cette liberté est d'autant plus garantie que le langage intermédiaire (IL) a été conçu pour supporter un grand nombre de langages avec efficacité[23]. Par exemple, C# n'a pas besoin d'un mécanisme équivalent à Java Native Interface[24] car le programmeur peut choisir de passer en mode « *unsafe* » pour exécuter un tel code, lequel est toujours écrit en C# dans le même fichier source. De même, C# autorise l'usage des pointeurs. Bien que leur manipulation soit très restreinte, il est par exemple possible d'obtenir un pointeur sur une fonction d'une classe instanciée via un délégué (*delegate*).

En conclusion, C# est séduisant dans sa promesse de concilier le meilleur de C++ et de Java en rendant au programmeur la liberté d'écrire un code plus optimisé et élégant que ne le permettrait Java dans certaines occasions. James Gosling, le concepteur de Java, pourra très probablement le détester au motif que le paradigme objet pur tel qu'il le conçoit est violé par C# ; il avait d'ailleurs déjà qualifié Anders Hejlsberg en 1998 du sobriquet de « Mr Method

[21] Des compilateurs tels que Jove, BulletTrain, JET peuvent générer du code natif, mais ont malgré tout recours à au moins une *thread* interpréteur et une *thread* ramasse-miette.

[22] « Il y a plus d'une manière de le faire » (Larry Wall, concepteur de Perl)

[23] En effet, une simple machine de Turing est théoriquement capable d'offrir les fonctionnalités des langages actuels, mais cela demanderait beaucoup de travail !

[24] JNI : outil permettant d'exécuter du code natif (C ou C++) dans la machine virtuelle. JNI est très utile notamment pour fournir des pilotes de périphériques à Java ou pour accélérer des sections critiques de code. A la différence du programme Java qui l'utilise, le code C utilisé n'est généralement pas portable.

Pointers ». En vérité, il nous semble au contraire que C# le respecte pleinement, son seul désavantage réside peut-être dans une plus grande difficulté d'apprentissage que Java car son expressivité peut être source de confusion. A cet égard, Java nous semble être une très bonne école avant de s'engager vers C#.

Notons enfin que, à la différence de Sun, Microsoft a accepté de standardiser C# ce qui le libère de son emprise directe – il restera en pratique lié aux bibliothèques de .NET - et permet aux concurrents d'écrire des compilateurs sans verser de royalties à Microsoft, comme c'est le cas actuellement notamment pour IBM ou Amiga avec leur Java Development Kit.

2.2 Les plates-formes

Si les langages illustrent certains partis pris opposés, les plates-formes sont encore bien plus différenciées. Chaque architecture répond globalement à une problématique de service identique, qui correspond aux besoins des clients et du marché, mais avec des approches différentes.

2.2.1 Comparaison détaillée

2.2.1.1 Composants clés

Microsoft .NET	J2EE	Commentaires
Langage C#	Langage Java	C# et Java ont des mécanismes équivalents (ramasse-miettes, héritage simple, typage sûr…) et héritent de la syntaxe de C++. Plus moderne, C# innove avec les attributs qui permettent de décrire le comportement (i.e. déploiement) du code dans le source. Avantage ou désavantage selon les écoles, C# offre une souplesse et une expressivité plus grande que Java, très puriste quant au paradigme orienté-objet.
Common Language Runtime (CLR)	Java Virtual Machine (JVM)	La JVM est un interpréteur de *bytecode*, le CLR compile le code IL systématiquement en code natif. Le CLR est conçu pour accepter un grand nombre de langages et permet grâce aux assemblages la création facile de composants partagés. Java ne peut pour l'instant le faire qu'à l'aide de CORBA, qui n'est pas autant intégré dans J2EE.
Base Class Library	Java 2 Platform API	Quoique très comparable dans les classes de base fournies, .NET se démarque de Java par son excellent support intégré de XML et de SOAP.
- Active Server Pages (ASP.NET) - WebForms	Java Server Pages (JSP)	ASP.NET et JSP sont toutes deux écrites en langage de haut niveau (Java pour JSP, les langages du CLS pour ASP.NET) et sont compilées (en *bytecode* JVM pour Java, en IL puis code natif pour ASP.NET). Les WebForms apportent une couche d'abstraction par rapport au langage cible de présentation (HTML, WML…) dont JSP ne dispose pas.
- Services Web - ADO.NET	- Enterprise JavaBeans - JDBC - (Apache-SOAP)	A l'aide de HTTP et XML, les services Web et ADO.NET permettent de mettre en ligne des services, de partager et mettre à jour des bases de données simplement et indépendamment de l'architecture cible. L'architecture J2EE est moins ouverte de par sa dépendance envers Java (RMI, messagerie Java). Un pont CORBA existe,

		mais n'est pas directement utilisable. Le modèle de données de JDBC est connecté à la différence d'ADO.NET. Apache-SOAP apporte les services Web à J2EE, mais n'est pas intégré à la plate-forme pour l'instant. La version 1.4 de J2EE prévoit un kit de développement pour les services Web[25].
WinForms	Java Swing	Bien que fortement orientés réseau, .NET et J2EE permettent le développement d'applications graphiques classiques. Swing et WinForms sont orientés-objets en respectant le paradigme Modèle-Vue-Contrôleur (MVC). Après longue maturation (échec de AWT), Swing fonctionne de manière presque similaire sur toutes les plates-formes cibles. WinForms est pour l'instant restreint à Windows uniquement.
Visual Studio.NET	- VisualAge for Java - Sun Forte for Java - …	Visual Studio.NET est simple d'emploi et très puissant, son intégration fine avec IIS permet des cycles de développements et de tests rapides. Les débutants peuvent écrire des services avec peu de connaissances initiales. Les outils de développement Java sont pléthore et dépendent de l'intégration avec un serveur d'application spécifique pour le déploiement (WebSphere pour VisualAge par exemple). Même dans ce cas, les EJB demeurent des entités complexes à manipuler.

2.2.1.2 Caractéristiques particulières

Caractéristique	Microsoft .NET	J2EE
Composants réutilisables et partagés, avec informations de déploiement	Assemblages, SOAP et DCOM - auto-description à l'aide des attributs.	Entreprise JavaBeans, RMI et CORBA – descripteurs des EJB séparés et souvent dépendants du serveur d'application.
Support des terminaux mobiles	.NET Mobile SDK - limité à Windows CE pour l'instant.	Java 2 Micro Edition (J2ME).
Développement multi-langage	Plus de 20 langages annoncés.	Langage Java uniquement - quelques tentatives de support de Python, Eiffel ou Cobol mais l'intégration est limitée.
Architectures supportées	Windows – le CLR et les soumissions à l'ECMA permettent un portage par des tiers.	Nombreuses (Windows, Solaris, AIX, Linux, Amiga…) – J2EE reste la propriété de Sun.
Disponibilité[26]	15% maintenant, 50% dans 6 mois et le reste dans 2 ou 3 ans.	90% maintenant, le reste dans 6 mois.

2.2.2 Synthèse

A première vue, Microsoft tente une nouvelle fois de s'imposer dans un domaine déjà défriché où il n'est pas installé en s'appuyant sur l'ubiquité de sa plate-forme Windows, mais cela reviendrait à négliger ce qui fait vraiment sa force sur la concurrence. Au-delà des nombreuses et réelles qualités techniques de son offre, Microsoft apporte son expertise en

[25] D'après cet article de 01net (http://www.01net.com/rdn?oid=150809&rub=1695).

[26] D'après Internet World: ".NET Framework: C# and the .NET Framework: A Better Java Platform?" du 15 mars 2001.

matière d'interface homme-machine et la formidable capacité d'intégration avec ses propres produits pour créer un environnement de développement Web puissant et réellement simple d'emploi. En tant que nouvel entrant, la compagnie de Redmond reprend l'initiative, ce qui est visible dans la presse, et met à profit l'expérience de Java à laquelle elle a participé avec Visual J++ et une Java Virtual Machine remarquable (celle d'Internet Explorer). Ce qui est apparu dans Java par sédimentation au fur et à mesure des évolutions, Microsoft l'intègre pleinement dans la plate-forme avec des constructions plus simples.

La plate-forme J2EE est beaucoup plus complexe à utiliser pour les développeurs, mais à ce prix elle prend en charge plus de mécanismes comme la gestion de la sécurité (au sens authentification et contrôles d'accès), la persistance et la gestion du cycle de vie des objets notamment. Sa robustesse, qui s'appuie sur Java, se vérifie avec de nombreux déploiements couronnés de succès dans le monde. L'approche par descripteur de déploiement XML de J2EE se justifie théoriquement par une séparation des rôles entre le développeur de Bean et son utilisateur, mais en pratique les attributs apportent une solution plus élégante dans la majorité des cas d'usage (nous ne parvenons pas à trouver de contre-exemple significatif), au point que nous accueillerions volontiers une évolution du langage Java dans ce sens. Les services Web par SOAP sont déjà implémentés en Java avec le support solide de IBM, de Sun et de sociétés comme CapeClear (pont SOAP-EJB), bien que l'intégration à J2EE ne soit pas aussi complète que dans .NET.

Commercialement, le matériel PC Intel x86 (et demain Intel Itanium 64 bits) et les logiciels Microsoft sont généralement beaucoup moins coûteux que leurs équivalents J2EE, les serveurs UNIX Sun ou IBM et les serveurs d'application d'entreprise[27]. La question des performances et de la disponibilité est plus délicate à débattre, les deux camps ayant chacun des arguments en leur faveur et à charge pour le concurrent. Les deux plate-formes sont scalables grâces aux équilibreurs de charge, mais J2EE sous UNIX peut s'appuyer sur du matériel nettement plus performant que l'architecture Intel x86, à savoir les serveurs haute-performance RISC/6000 (IBM) et SPARC (Sun Microsystems) notamment. Concernant la qualité des logiciels, Microsoft essaie de balayer les critiques sur les bugs depuis Windows 2000 et s'appuie sur le CLR pour augmenter la fiabilité de ses propres applications.

Sur le front des standards, Microsoft a fait preuve d'ouverture en embrassant XML, SOAP et en standardisant des parties importantes de .NET. Ce changement de stratégie s'explique bien sûr par la concurrence de Java, réputé pour son ouverture (API publiée, indépendante de la plate-forme, gratuité du JDK et des navigateurs pour les applets), mais surtout par le modèle de location des services – pour pouvoir louer efficacement ses services, Microsoft a besoin de disposer d'une assise aussi large que possible, c'est-à-dire de permettre à un grand nombre de postes clients d'accéder aux services Microsoft.

Le support de nombreux langages par le CLR est profitable à Microsoft puisqu'il attire la faveur des laissés pour compte par Java tout en permettant à Microsoft ou à ses partenaires de vendre plusieurs outils de développement appropriés, ce qui était déjà le cas avant .NET avec Visual Basic, Visual J++, Visual C++ et Visual FoxPro. En revanche, la réhabilitation des langages menacés s'effectue au sacrifice de la portabilité, puisque .NET n'existe que sous

[27] Certes des solutions extrêmement peu coûteuses combinent notamment Linux, Tomcat et JBoss sur plate-forme x86, mais de nombreuses entreprises se sentent plus sécurisées par des solutions commerciales, malgré leur coût parfois prohibitif.

Windows à moyen terme. De plus, ainsi que nous l'avons vu, les règles du CLS imposent certaines contraintes aux langages du CLR, qui peuvent y perdre leur âme.

En conclusion, Microsoft.NET se présente comme une plate-forme très attractive pour le développeur, à ceci près qu'elle n'existe pas et n'est pas encore tout à fait stabilisée[28] alors que J2EE est tangible et a déjà été déployée dans de nombreux projets. Si la version finale comble les manquements de la version testée comme promis, il est évident qu'elle sera un concurrent de poids pour J2EE, ne serait-ce à travers les entreprises utilisant déjà un réseau Microsoft. L'interopérabilité avec les autres implémentations de SOAP et Java notamment sera un enjeu majeur car, à la différence de Java, .NET n'existe que sous Windows.

[28] Le site 123ASPX répertorie un très grand nombre de changements entre la version bêta 1 et bêta 2 (sortie autour de la fin de la rédaction de ce mémoire) à l'adresse http://www.123aspx.com/b1to2changes/default.asp

3 Les Services Web

Indéniablement une innovation centrale de .NET, les services Web sont la brique de base de l'Internet de demain. Ils permettront d'agréger des informations provenant de nombreuses sources distinctes dans l'Internet, qu'elles soient des serveurs au sens fort du terme ou des pairs (*peer-to-peer*), sans se soucier de l'architecture matérielle ou logicielle de ces derniers. Un logiciel de messagerie instantanée nommé Jabber développe précisément ce concept, et il n'est que le précurseur.

Les services Web reposent sur le protocole SOAP, anciennement connu sous le nom de XML-RPC, pour interopérer de manière neutre. La spécification, sur laquelle Microsoft a travaillé avec IBM notamment, est en cours de validation par le W3C. Ainsi, SOAP est réellement ouvert et, quand les implémentations des différents acteurs seront stabilisées, il sera facile d'écrire des clients SOAP effectuant des appels sur des objets servis par .NET sur n'importe quelle architecture. Ce qui est plus surprenant, c'est qu'il existe déjà une implémentation libre en Java de SOAP connue sous le nom de Apache-SOAP dont la dernière version, si cela a une signification, est 2.1 à ce jour. Apache-SOAP permet d'écrire des clients SOAP en Java et, côté serveur, se conjugue très bien avec le serveur d'application libre Apache Tomcat, lequel offre toute l'infrastructure nécessaire pour servir des requêtes SOAP sur l'Internet. Cette implémentation est dans sa quasi-totalité l'œuvre d'IBM (issue du projet AlphaWorks SOAP4J) qui a mis à profit sa participation dans la conception du protocole pour en fournir une implémentation très complète. De même, un module `SOAP::Lite` existe pour le langage Perl.

3.1 Le protocole SOAP

SOAP est un protocole en cours d'adoption par le W3C[29] décrivant des appels de procédure à distance (*remote procedure call – RPC*). Officiellement, l'acronyme signifie *Simple Object Access Protocol* (protocole simple d'accès aux objets) mais la qualification de *Services Oriented Architecture Protocol* (protocole pour une architecture orientée-services) lui convient parfaitement.

La répartition des auteurs est très intéressante, en effet sur huit auteurs de la version 1.1, on peut compter quatre membres de Microsoft et deux d'IBM/Lotus (les deux derniers membres provenant de UserLand et DevelopMentor) ; de plus, IBM n'avait pas participé à la version 1.0. On le voit, Microsoft soutient activement SOAP, mais IBM a également compris l'intérêt du protocole.

Le seul poids de ces deux compagnies sur le monde informatique laisse à penser que le développement des services basés sur SOAP va devenir très important au cours des prochaines années.

3.1.1 Principes de fonctionnement

Pour résoudre les problèmes d'interopérabilité entre composants logiciels, de nombreux protocoles ont été conçus ces dernières années. Jusqu'à la création de SOAP, trois protocoles étaient plus particulièrement utilisés :

[29] La spécification de la version 1.1 est disponible à l'adresse http://www.w3.org/TR/SOAP .

- **RMI** (*Remote Method Invocation*). Ce protocole est très simple et très efficace, mais ne fonctionne que dans un environnement Java de bout en bout.

- **CORBA/IIOP** (*Common Object Request Broker Architecture*). Ce protocole, fruit du travail de l'OMG[30], permet de s'affranchir des architectures et des langages utilisés, objectif repris par .NET, à l'aide d'un langage neutre de définition d'interface IDL (*Interface Definition Language*) et d'un protocole commun de transport et de sérialisation des données. Malheureusement, l'OMG, en ciblant un public de professionnels de l'informatique très exigeant, a fourni une spécification très dense et une architecture au final très difficile à déployer hors quelques bons cas simples.

- **COM** (*Component Object Model*) et **DCOM** (*Distributed Component Object Model*). Ces protocoles ont été écrits par Microsoft pour faciliter la communication entre composants logiciels de Windows. Un portage UNIX a été réalisé par Software AG, mais en pratique ce protocole reste largement restreint au monde Windows et dans le contexte des Intranets.

On le voit, chaque protocole imposait une barrière à une véritable interopérabilité, qu'elle soit la plate-forme (Windows ou Java pour DCOM et RMI) ou tout simplement la complexité avec CORBA.

SOAP se différencie de ces protocoles par sa grande simplicité d'usage et sa neutralité. Les concepteurs de la norme se sont visiblement mis à la place des développeurs des services avec pour objectif de faciliter leur travail en supprimant les fonctionnalités les plus contraignantes de ses prédécesseurs.

XML a été choisi comme format d'échange, ce qui permet d'utiliser la norme XML Schema[31] pour la sérialisation des données. Les types autorisés sont limités à des cas volontairement simples (essentiellement chaînes de caractères, entiers, flottants et listes de ces derniers) mais suffisants pour reconstruire la plupart des objets. A la différence de ces derniers protocoles dont le format est binaire, les données transportées sont lisibles telles quelles par le développeur et leur manipulation par le client en est grandement facilitée.

La structure choisie pour SOAP correspond à celle d'une enveloppe dans laquelle le message, les données, sont encapsulés, selon la figure suivante.

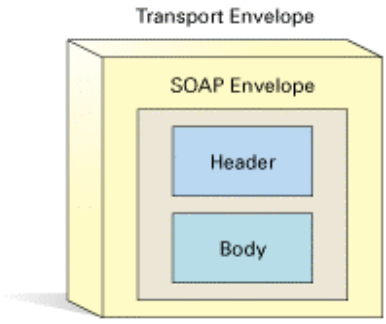


Figure 10. L'enveloppe SOAP

[30] Object Management Group, dont le site Web est à l'adresse http://www.omg.org .

[31] Disponible sur le site du W3C à l'adresse http://www.w3.org/TR/XMLSchema-2

Autre point fort, le XML se prête bien à l'intégration dans un serveur Web. Ainsi, la norme de SOAP prévoie qu'il utilise le protocole HTTP pour transporter les données, bien que d'autres protocoles soient disponibles, dont SMTP, implémenté par Apache-SOAP. Comme l'écrit avec enthousiasme Microsoft dans la présentation des Web Services, SOAP sur HTTP n'est pas bloqué par les pare-feu (*firewalls*) des entreprises, ce qui advenait souvent avec CORBA, RMI et DCOM, rendant impossible une communication au-delà de l'Intranet. En revanche, ceci pose un réel risque que nous examinons dans la section « SOAP et les firewalls » en page 38.

SOAP peut ainsi être servi efficacement par les serveurs d'application capables de gérer le XML, ce qui est le cas de IIS (Microsoft) et de J2EE.

Le client retire également des avantages de SOAP. Le pré-requis sont peu nombreux et très répandus, à savoir HTTP et XML ; les nombreuses implémentations de SOAP déjà disponibles tendent à le démontrer. Selon une classification consacrée, le protocole SOAP appartient à la famille des « glues dynamiques », c'est-à-dire que le déploiement (clients et serveurs) est léger et rapidement évolutif, à la différence de CORBA ou de RMI où l'installation de binaires et de descripteurs de déploiement lourds sont nécessaires pour interopérer.

Pour qu'il reste simple, les concepteurs du protocole ont dû accepter des concessions importantes. SOAP n'inclut notamment pas dans sa spécification la gestion du cycle de vie des objets et les pointeurs vers des instances distantes d'objets, comme CORBA, COM ou RMI. La gestion des sessions et des transactions a été volontairement exclue de la norme.

Enfin, comme le souligne Jim Farley, auteur d'ouvrages Java pour O'Reilly, l'actuelle spécification (1.1) ne spécifie pas d'encodage par défaut des messages. Certes, un encodage est défini, mais il n'est pas obligatoire pour être conforme à la spécification. N'importe quel encodage peut être spécifié dans l'attribut `encodingStyle` du message ou de certaines parties du message. Si l'encodage défini dans la spécification a de fortes chances de devenir un standard de fait, il existe une menace à l'interopérabilité qui rappelle les erreurs de jeunesse de CORBA lorsqu'il n'avait pas spécifié un protocole d'échange standard.

3.1.2 La sécurité et SOAP

Considérée comme une lacune dans le B2B, la spécification a volontairement écarté les considérations de sécurité, avec un accord récent du W3C sur cette question. Si la pression et la tentation d'inclure des mécanismes de sécurité ont été fortes, le protocole y aurait probablement perdu son intérêt, c'est-à-dire sa simplicité et sa facilité de mise en œuvre.

En vérité, les services n'implémentent pas de sécurité **par défaut**. Les critiques du protocole négligent le fait que SOAP ne définit qu'une enveloppe pour véhiculer les données et que leur représentation est extensible grâce à XML. De plus, d'après les prévisions de Microsoft, la majorité des services à venir n'auront pas ou peu besoin de sécurité.

Microsoft propose une vision centrée sur ses produits dans MSDN[32]. Nous nous attacherons à étudier la sécurité pour SOAP sur HTTP principalement, ce dernier étant le médium privilégié

[32] Disponible à l'adresse http://msdn.microsoft.com/vstudio/nextgen/technology/security.asp sous le titre «Web Services Security ».

de SOAP. Nous considérerons la sécurité sous ses trois aspects, à savoir la confidentialité, l'authentification et l'autorisation, puis enfin la relation entre SOAP et les pare-feu.

3.1.2.1 La confidentialité

SOAP servira sans aucun doute à transporter des données plus sensibles que les services HTTP actuels. Le cryptage devient alors nécessaire pour s'assurer qu'un tiers malveillant ne peut pas, en écoutant le réseau, intercepter les informations.

Or, avec un chiffrement autorisé jusqu'à 128 bits, le protocole HTTPS (HTTP over SSL) répond parfaitement à ces attentes. Précisons que puisque HTTP est encapsulé dans SSL, le cryptage est complètement transparent pour l'implémentation des services SOAP, quoiqu'il entraîne une dégradation des performances et de la scalabilité de l'infrastructure.

Enfin, pour les autres protocoles comme SOAP sur SMTP, les messages peuvent aussi être cryptés en utilisant un mécanisme natif du protocole, comme S/MIME pour le mail signé et crypté.

3.1.2.2 Authentification et autorisation

L'authentification est l'acte qui permet au système d'identifier un utilisateur, le plus souvent à l'aide d'un couple login et mot de passe ; elle se distingue de l'autorisation qui consiste à vérifier qu'un utilisateur authentifié a la permission d'effectuer une tâche. Alors que l'authentification est nécessaire avant d'effectuer toute action dans le système, l'autorisation dépend des politiques internes à l'application et n'entre donc pas sous la responsabilité du protocole de transport des données.

L'authentification pour SOAP peut être effectuée de deux manières. La première consiste à utiliser le mécanisme de HTTP, la deuxième étend le protocole en lui apportant une extension.

3.1.2.2.1 Authentification HTTP

Par construction, SOAP peut utiliser les mécanismes du protocole sous-jacent utilisé, HTTP dans notre cas.

En excluant les extensions apportés à Microsoft pour un client Internet Explorer et un serveur IIS, HTTP et la RFC 2617 plus précisément fournit deux méthodes d'authentification, « Basic » et « Digest ». Ces méthodes ne supportent que le mécanisme login / mot de passe classique et peuvent être inadaptées selon les cas. De plus, HTTPS peut également fournir une authentification fiable à l'aide de certificats.

- **Basic**. Utilisé pour une authentification non sécurisée ou semi sécurisée, car le login et le mot de passe sont envoyés en clair sur le réseau.

- **Basic sur SSL**. Fonctionne comme « Basic » à l'exception que les données sont cryptées par SSL. La sécurité fournie par cette méthode est très bonne, mais affecte les performances du serveur.

- **Digest**. Utilise un algorithme de hachage pour transmettre le login et le mot de passe qui offre également une protection contre les « replay attacks », c'est-à-dire le fait de rejouer l'authentification avec les mêmes données pour entrer dans le système sans connaître le mot de passe.

- **Certificats**. Cette méthode nécessite que les deux parties aient obtenu un certificat d'un tiers certificateur (payant), et connaît un développement plutôt lent. Toutefois, elle est reconnue comme très sûre.

Le choix est déterminé par les besoins de chaque service, il faut garder simplement à l'esprit que « Basic » offre une protection très faible et qu'il est donc à proscrire si l'enjeu du service est important.

Dans le cas d'un poste client et d'un serveur Windows, IIS peut fournir des mécanismes d'authentification basés sur l'identité dans le réseau Microsoft et donc transparents. De plus, le *building-block service Passport* (abordé en page 22) permet une authentification plus universelle, qui d'adapte au terminal en utilisant notamment Basic sur SSL et les certificats pour fonctionner. Passport fournit un jeton valide au consommateur de service qui peut alors être utilisé pour l'autorisation. N'oublions pas que ce service sera payant, probablement en fonction du temps (location) ou du trafic engendré.

3.1.2.2.2 Authentification par extension

Le IBM Resarch Center de Tokyo a élaboré une extension légale de SOAP presque équivalente aux certificats SSL. L'extension utilise l'entête SOAP pour transmettre l'information d'authentification et pour garantir l'intégrité des données.

Microsoft, VeriSign et WebMethods ont de leur côté annoncé une spécification pour le XML sécurisé fondé sur les signatures digitales et le cryptage nommée XML Key Management Specification (XKMS). D'après VeriSign, XKMS est une spécification pour la gestion des clés publiques pour permettre de signer ou de crypter, ou autres applications des clés publiques. Il est prévu pour interopérer avec le standard de signature XML préparé par l'IETF (*Internet Engineering Task Force*) et le W3C.

Il est ainsi démontré que la sécurité peut être implémentée en utilisant l'extensibilité de SOAP et justifie, s'il était besoin de le faire, de garder le protocole aussi simple que possible.

3.1.2.3 SOAP et les firewalls

Microsoft présente le fait que le protocole SOAP sur HTTP ne soit pas refusé par les pare-feu comme une fonctionnalité, et il est vrai qu'ils constituaient un obstacle dans les déploiements hors Intranets de CORBA, COM ou RMI. Or, le rôle des pare-feu n'est pas de bloquer les protocoles utiles mais bien de protéger un réseau contre des attaques extérieures malveillantes. SOAP par HTTP (port 80) pose de réels problèmes de sécurité pour l'administrateur réseau comme pour le dirigeant d'entreprise, lequel n'aimerait pas entendre que son coûteux pare-feu ne peut pas le protéger contre des attaques RPC distantes par SOAP.

Le risque peut être prévenu de deux manières :

- Utiliser des mécanismes logiciels d'autorisation. En programmant les Web Services de telle sorte qu'un service sensible attend un identifiant valide de session ou s'assure que l'adresse IP de l'envoyeur n'est pas à proscrire, la sécurité peut être assurée au cas par cas en conservant une certaine souplesse. Les risques ne sont pas nuls, mais utilisée avec HTTPS, cette méthode offre une sécurité de très haut niveau, à condition que la partie logicielle soit correctement programmée.

- Configurer le pare-feu pour détecter les entêtes SOAP. Le protocole SOAP spécifie les entêtes des messages SOAP de telle sorte qu'un pare-feu peut être configuré pour les

bloquer, pourquoi pas en conjonction avec d'autres règles comme le blocage de certaines adresses IP ou de certains services SOAP destinés à l'Intranet uniquement.

3.2 Les Web Services et .NET

3.2.1 Vue d'ensemble de l'architecture

Microsoft a incontestablement fourni un effort remarquable pour que la conception des clients comme des serveurs SOAP soit extrêmement simple depuis Visual Studio.NET. Ce dernier intègre désormais le logiciel Visual InterDev dans une nouvelle interface, assez déroutante de prime abord mais l'impression disparaît à l'usage comme pour la plupart des outils Microsoft.

Les Web Services sont servis par le serveur IIS 5.0 de Windows 2000 et à ce titre bénéficie des fonctionnalités particulières de ce dernier.

- Le déploiement d'un service est exactement le même que pour les pages ASP à l'aide des extensions de publication FrontPage. Ces extensions définissent un protocole de mise à jour et de téléchargement de données entre un client habilité à modifier ou ajouter une page et le serveur destiné à l'héberger. Elles sont très répandues pour publier simplement des pages par Internet aujourd'hui, le serveur libre Apache implémentant également le protocole sous la forme d'un module. Visual Studio.NET prévoit la publication simultanée vers plusieurs miroirs et/ou une ferme de serveurs à l'aide des « *deployment projects* » qui contiennent les options de déploiement. Notons qu'il est également possible d'écrire un simple fichier `.asmx` contenant tout le code, dans n'importe quel langage supporté, et de le copier à l'emplacement désiré dans le répertoire `wwwroot` d'IIS.

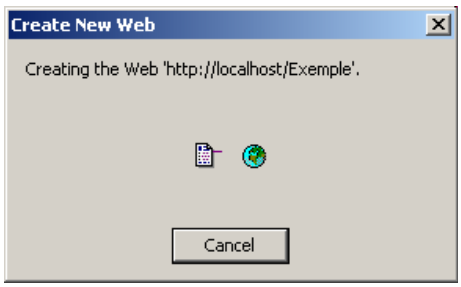


Figure 11. L'extension serveur FrontPage en action

- ASP.NET utilise les extensions ISAPI d'IIS pour donner à chaque service une URL distincte en `.asmx` à laquelle il attache des comportements.
    - Sur un GET HTTP simple, IIS sert une page de présentation simple du service. En un coup d'œil, le consommateur prend connaissance en langue humaine (Anglais uniquement pour l'instant) du contenu du service.
    - Sur un POST SOAP, IIS interprète la requête SOAP, l'exécute et sérialise le résultat vers le client.
    - Si le client demande la page avec le paramètre `?SDL`, la description SDL du service est renvoyée.

- Enfin, si une erreur survient pendant une requête, IIS renvoie, en mode de débuggage, une page HTML détaillant l'erreur avec un message en Anglais clair et bien sûr l'état de la pile.

Les images suivantes illustrent notre propos sur un exemple simple.

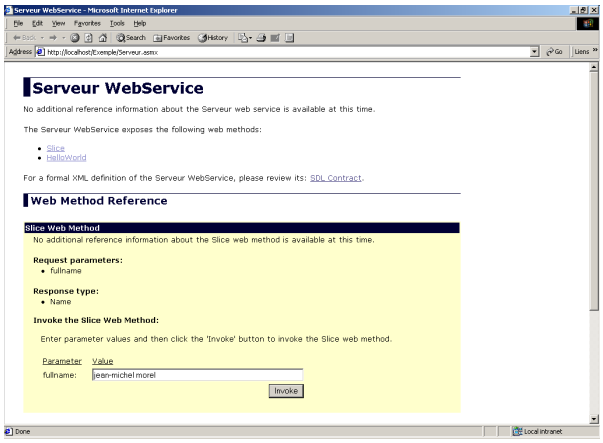


Figure 12. La page HTML d'auto description générée par ASP.NET

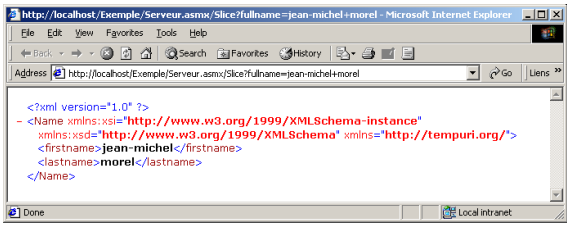


Figure 13. La réponse de ASP.NET à une requête effectuée « à la main »

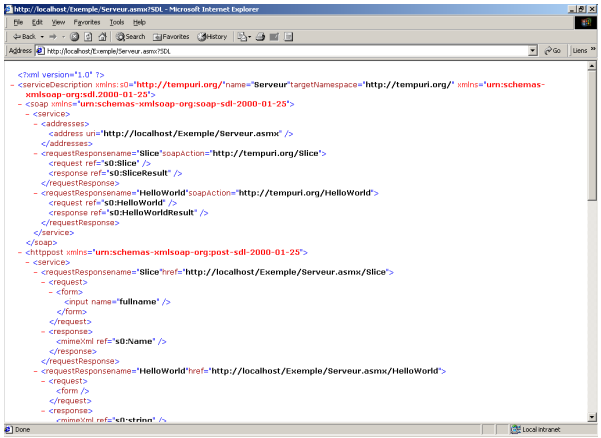


Figure 14. La description SDL (en ajoutant `?SDL` à l'URL)

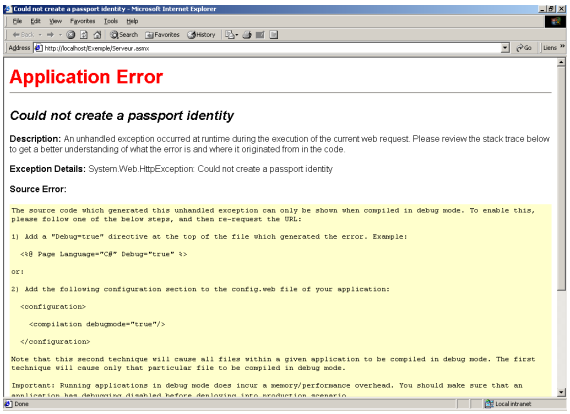


Figure 15. L'affichage d'une erreur par IIS

On le voit, la mise en œuvre est facilitée à l'extrême grâce à l'environnement Visual Studio.NET dont l'interaction étroite avec IIS et ASP.NET permet un développement rapide des services.

3.2.2 Le service

Nous allons implémenter un service simple en C# pour illustrer sa puissance. A cet effet, nous créons un nouveau projet C# « Web Service ».

Un *wizard*, très classique dans le monde Windows, nous demande quelques informations simples puis crée plusieurs fichiers dont un source `.asmx` décrivant le service. Nous recommandons l'affichage du « Solution Explorer » (onglet *View*).

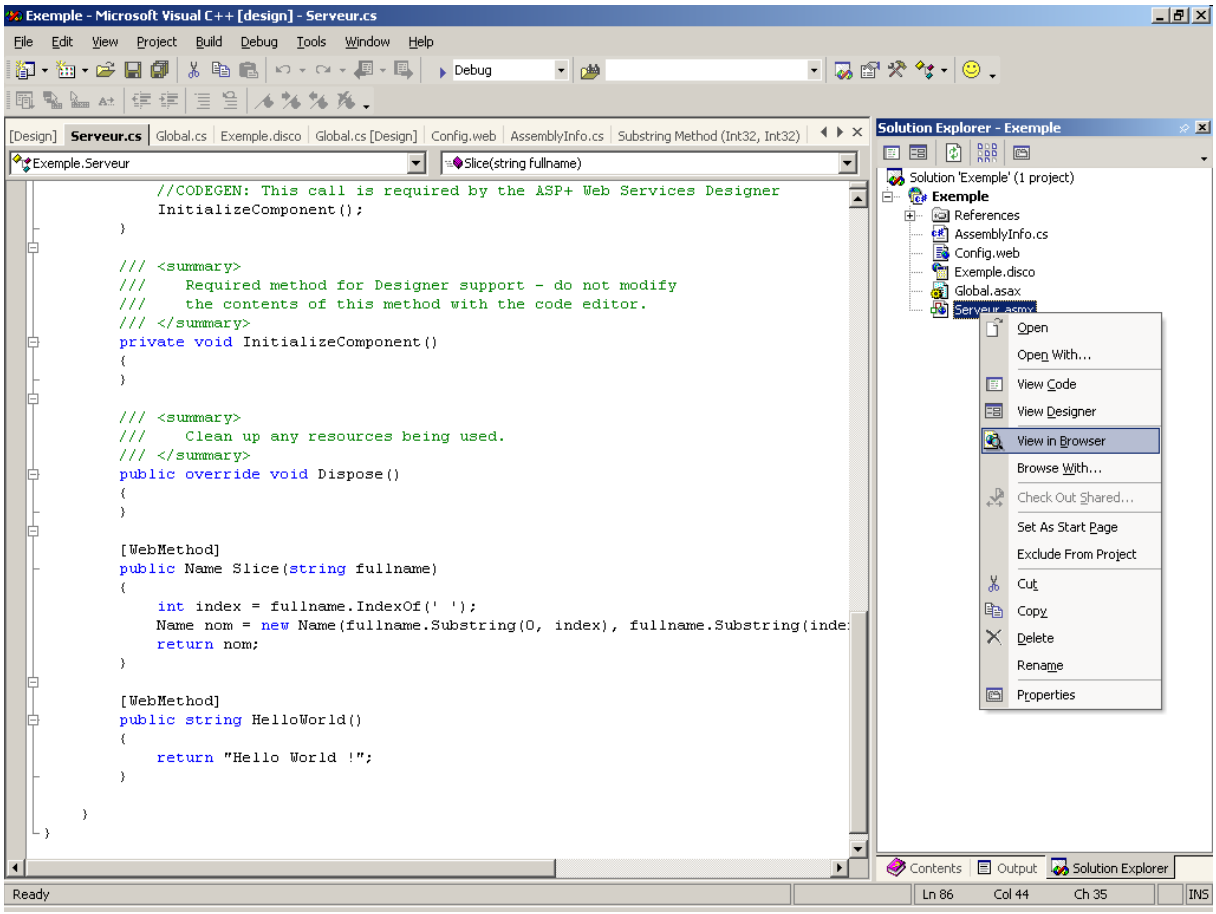


Figure 16. Implémenter un service Web avec Visual Studio.NET

Sur la capture d'écran ci-dessus, on peut voir la simplicité et la puissance de l'environnement. Il suffit d'ajouter le code du service dans le source `.cs`, de construire (*build*) le projet et de cliquer bouton droit sur le fichier `.asmx` dans le « Solution Explorer » et de sélectionner « View in Browser » pour afficher la page HTML d'auto description vue précédemment.

Nous avons simplement ajouté :

```
[WebMethod]
public Name Slice(string fullname)
{
    int index = fullname.IndexOf(' ');
    Name nom = new Name(fullname.Substring(0, index), fullname.Substring(index+1));
    return nom;
}

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

L'attribut `[WebMethod]` indique une méthode qui doit être exportée pour les classes filles de `System.Web.Services.WebService`. Une grande puissance de C# est l'auto description du comportement du service dans son source et son code. C# élimine les fichiers de description ou de déploiement additionnels auxquels Java nous avait habitués. Ces derniers peuvent être découplés du service qu'ils décrivent, ce qui devient ingérable lorsque la complexité augmente. Avec .NET et les attributs, le code sait dès la compilation comment se comporter

au déploiement, ce qui autorise les prouesses comme l'auto description dont nous avons parlé avec le SDL.

La classe `Name` a été définie dans le même fichier par simplicité. Nous la présentons car elle illustre la sérialisation des objets. La contrainte dans .NET est exactement la même qu'en Java avec les JavaBeans. En effet, pour être sérialisable, et donc utilisable dans une requête SOAP, la classe doit avoir les caractéristiques suivantes :

- Posséder un constructeur sans arguments.
- Toutes ses propriétés et membres publics doivent être également sérialisables.

Précisons qu'il s'agit, comme pour les JavaBeans, d'un contrat moral avec l'application parce que le compilateur ne peut pas vérifier qu'un objet sérialisé ainsi ne sera pas « cassé » lors de la récupération puisque certaines données privées peuvent manquer.

Nous avons écrit le code suivant pour `Name`, dont on remarquera l'ajout d'un constructeur vide sans arguments :

```
public class Name
{
   string _firstname;
   string _lastname;

   public string firstname
   {
        set
        {
             _firstname = value.Trim();
        }

        get
        {
             return _firstname;
        }
   }

   public string lastname
   {
        set
        {
             _lastname = value.Trim();
        }

        get
        {
             return _lastname;
        }
   }

   // Le vrai constructeur
   public Name(string first, string last)
   {
        _firstname = first;
        _lastname = last;
   }

   // Un constructeur vide pour permettre la sérialisation (// JavaBeans)
   public Name()
   {
   }
}
```

Enfin, la sécurité du service se définit très simplement selon les règles énoncées page 37 à l'aide de l'outil d'administration de IIS (dans « Panneau de configuration » puis « Outils d'administration » comme illustration en Figure 17).

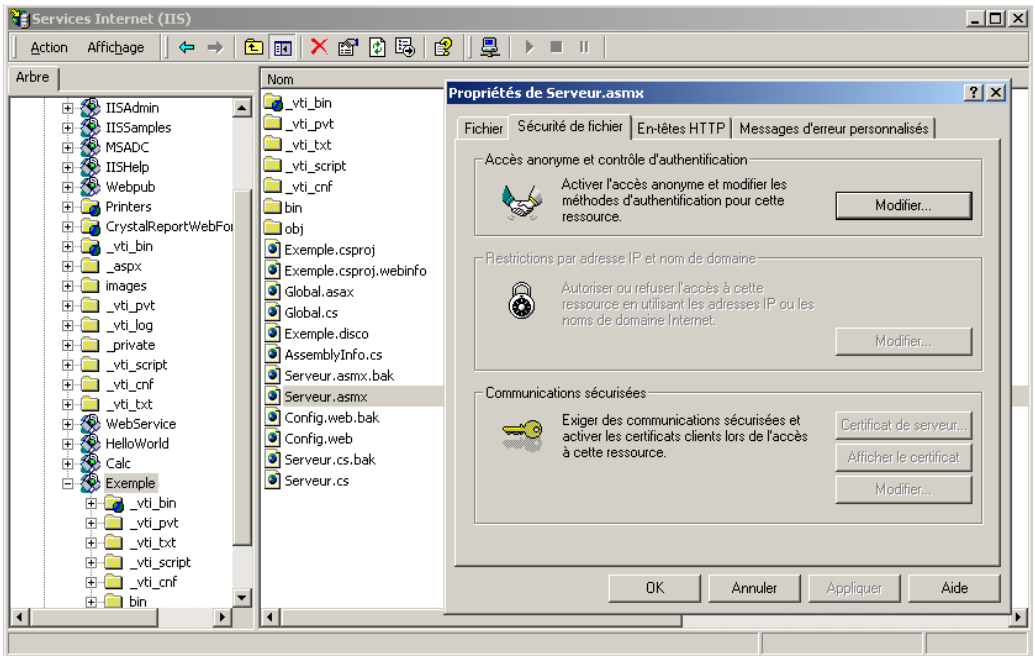


Figure 17. La sécurité sous Internet Information Server[33]

3.2.3 Le consommateur

Le fichier SDL permet de créer un client pour le Web Service. Nous n'avons pas trouvé d'outil, s'il existe, intégré dans Visual Studio.NET, mais il existe un utilitaire en ligne de commande livré avec le *.NET Framework SDK* nommé `WebServiceUtil`. Nous avons enregistré le fichier SDL de notre projet dans un répertoire de travail puis lancé la commande :

```
WebServiceUtil /command:proxy /language:CSharp /path:slice.sdl
```

Nous obtenons alors le fichier suivant, auquel nous avons simplement ajouté la classe Client pour effectuer un appel au service « HelloWorld ».

```
using System.Xml.Serialization;
using System.Web.Services.Protocols;
using System.Web.Services;
using System;

public class Serveur : System.Web.Services.Protocols.SoapClientProtocol {
    public Serveur() {
        this.Url = "http://localhost/Exemple/Serveur.asmx";
    }

    [System.Web.Services.Protocols.SoapMethodAttribute("http://tempuri.org/Slice")]
    public Name Slice( string fullname) {
        object[] results = this.Invoke("Slice", new object[] {fullname});
        return (Name)(results[0]);
    }
    public System.IAsyncResult BeginSlice(string fullname, System.AsyncCallback callback, obje
ct asyncState) {
        return this.BeginInvoke("Slice", new object[] {fullname}, callback, asyncState);
    }
    public Name EndSlice(System.IAsyncResult asyncResult) {
        object[] results = this.EndInvoke(asyncResult);
```

[33] Windows 2000 Server est requis pour dégriser certaines fonctionnalités (tests effectués sous Windows 2000 Professional).

```
        return (Name)(results[0]);
    }
    [System.Web.Services.Protocols.SoapMethodAttribute("http://tempuri.org/HelloWorld")]
    public string HelloWorld() {
        object[] results = this.Invoke("HelloWorld", new object[0]);
        return (string)(results[0]);
    }
    public System.IAsyncResult BeginHelloWorld(System.AsyncCallback callback, object asyncStat
e) {
        return this.BeginInvoke("HelloWorld", new object[0], callback, asyncState);
    }
    public string EndHelloWorld(System.IAsyncResult asyncResult) {
        object[] results = this.EndInvoke(asyncResult);
        return (string)(results[0]);
    }

}
[System.Xml.Serialization.XmlRootAttribute("result", Namespace="http://tempuri.org/", IsNullab
le=true)]
public class Name {
    [System.Xml.Serialization.XmlElementAttribute("firstname", IsNullable=true)]
    public string Firstname;
    [System.Xml.Serialization.XmlElementAttribute("lastname", IsNullable=true)]
    public string Lastname;

}

public class Client {
    public static void Main(string[] args) {
        Serveur exemple = new Serveur();

        Console.WriteLine(exemple.HelloWorld());
        Name test = exemple.Slice("Jean Dupont");
    }

}
```

Nous pouvons à l'aide du code mieux comprendre le mécanisme de sérialisation. Le fichier SDL fournit une description de `Name` qui se voit reconstitué ici, limité à ses attributs `firstname` et `lastname`. Une méta-donnée indique qu'il s'agit d'une classe obtenue à partir d'une description de sérialisation pour permettre de substituer la définition originale si elle est fournie.

`WebServiceUtil` permet également de générer un fichier SDL à partir d'un fichier source C# équivalent, mais l'usage est moins pratique que le paramètre `?SDL` dans l'URL.

Contrairement aux idées reçues, il est possible travailler sans Visual Studio.NET en compilant à l'aide d'une ligne de commande. L'intérêt de cela est l'apparition plus claire de l'équivalent .NET du `CLASSPATH` peu apprécié des utilisateurs de Java.

```
csc   /r:System.Web.Services.dll;System.XML.dll;System.Xml.Serialization.dll
Client.cs
```

Les fichiers DLL proviennent du *.NET Framework SDK* et peuvent être trouvés dans `C:\WINNT\Microsoft.NET\Framework`. Nous obtenons alors un fichier exécutable – à la différence de Java – qui affiche bien « Hello World ! » sur la console.

Regardons à l'aide de l'utilitaire de débuggage de Apache-SOAP présenté plus loin l'échange qui a lieu pour la ligne Name test = exemple.Slice("Jean Dupont"). Pour se faire, nous modifions l'URL du service en écrivant `localhost:8080` plutôt que `localhost` – les requêtes pourront alors être interceptées par l'utilitaire d'Apache qui les renverra sur le port 80 d'IIS.

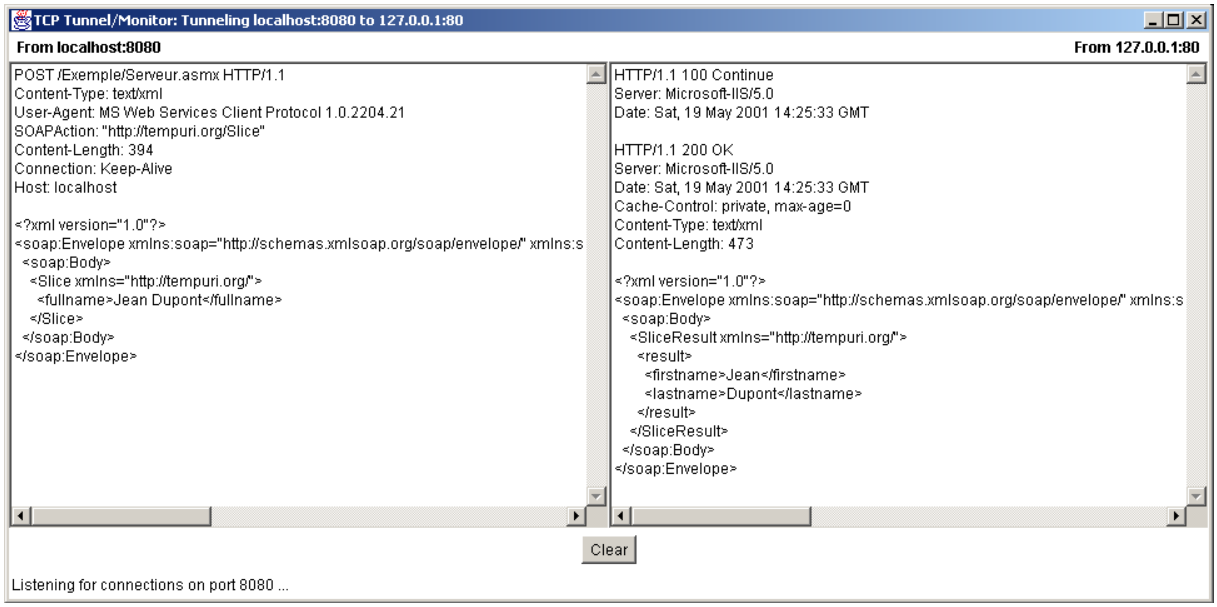


Figure 18. Un avantage de SOAP : les échanges sont compréhensibles par un être humain

3.2.4 Zoom sur l'API des Web Services .NET

Cette API est fournie par le SDK dans l'espace de nommage `System.Web.Services`. La présence dans le SDK est important, car il semblerait probable que Microsoft le distribue gratuitement, certes pour les seuls utilisateurs de Windows.

- `WebServiceAttribute`. C'est un attribut global pour une classe héritée de `WebService`. Il permet notamment de préciser l'espace de nommage du service, qui l'identifie de manière unique dans l'Internet. Par défaut, Microsoft utilise http://tempuri.org/ ce qui explique pourquoi il apparaît dans les échanges SOAP. Par exemple, l'attribut `[WebService Namespace="http://www.enst.fr/memoire_NET/")]` permet de spécifier un espace non ambigu pour des services développés pour ce mémoire.

- `WebMethodAttribute`. Cet attribut est associé aux méthodes qui doivent être exportés par le service. Il permet également de commenter et de préciser très finement le comportement de la méthode, dans l'esprit des attributs, c'est-à-dire éviter les fichiers de description en incluant les informations directement dans le source directement.

- `WebServicesConfiguration`, `WebServicesConfigurationSectionHandler`. Ils fournissent des informations à l'environnement d'exécution.

Les attributs liés à `[WebMethod]` méritent d'être décrits plus en détail.

- `Description`. Il permet de fournir un message décrivant le service pour l'utilisateur. Ce dernier sera visible dans le fichier SDL et la page de présentation du service.

- `EnableSession (boolean)`. Cet attribut permet de préciser à IIS de transmettre et de conserver l'état de la session pour cette méthode. La session est activée par défaut, mais la désactiver pour les méthodes ne le nécessitant pas permet d'améliorer les performances.

- `MessageName`. Cet attribut permet de distinguer deux méthodes ayant le même nom. Le fichier SDL ajoute le message comme une propriété dans la description XML du service, ce qui permet de distinguer les deux méthodes.

- `TransactionMode`. Puisque le protocole HTTP est sans état, s'il y a transaction, le service est toujours l'initiateur de celle-ci. Activer le mode transaction permet en revanche de travailler avec des objets COM au sein d'une transaction.

Les attributs s'écrivent de la manière suivante :

```
[WebMethod(MessageName="Slice", Description="Separe nom et prenom.")]
public Name Slice(string fullname)
```

Notons enfin l'existence du fichier `config.web`, généré automatiquement par Visual Studio.NET - si le service est décrit directement à l'aide d'un éditeur de texte, les valeurs par défaut sont utilisées.

Dans ce fichier XML, on trouve des paramètres supplémentaires, exclus du source ce que déroge à la règle des attributs. Outre des paramètres de débuggage, qui doit être désactivé en production, le fichier permet de configurer la sécurité, la journalisation, la gestion de la session et l'internationalisation.

```
<!--
     SECURITY
    This section sets the security policies of the application. Possible modes are
    "Windows", "Cookie", "Passport" and "None"
-->
 <security>
   <authentication mode="None" />
 </security>
 <!--
     APPLICATION-LEVEL TRACE LOGGING
    Application-level tracing enables trace log output for every page within an
    application. Set trace enabled="true" to enable application trace logging.  If
    pageoutput="true", the trace information will be displayed at the bottom of each page.
    Otherwise, you can view the application trace log by browsing the "trace.axd" page
    from your web application root.
 -->
 <trace enabled="false" requestlimit="0" pageoutput="false" tracemode="SortByTime" />
 <!--
     SESSION STATE SETTINGS
    By default ASP+ uses cookies to identify which requests belong to a particular
    session. If cookies are not available, a session can be tracked by adding a session
    identifier to the URL. To disable cookies, set sessionstate cookieless="true".
 -->
 <sessionstate inproc="true" usesqlserver="false" cookieless="false" timeout="20"
    server="localhost" port="42424" />
 <!--
     GLOBALIZATION
    This section sets the gloablization settings of the application.
 -->
 <globalization requestencoding="utf-8" responseencoding="utf-8" />
```

Comme nous l'avons exposé en page 22, Microsoft souhaite louer la gestion de la sécurité des services Web à l'aide de Passport, ce qui explique sa présence dans la liste des paramètres possibles de sécurité. Le paramètre « Windows » indique une sécurité Basic, Digest ou le module intégré d'authentification Windows de IIS comme décrits en page 37.

3.2.5 Le proxy ROPE (Soap Toolkit)

Ce *proxy* a un intérêt pour mettre en ligne des objets COM existants puisqu'il agit en intermédiaire (*proxy* signifie délégué) qui transforme les requêtes SOAP entrantes en requêtes COM et traduit la réponse COM reçue en réponse SOAP. La description du service se fait soit à partir d'un fichier SDL existant soit en choisissant des méthodes COM à exposer.

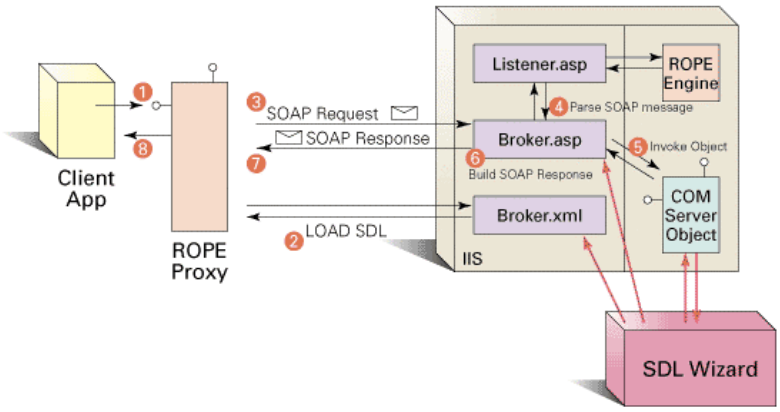


Figure 19. L'architecture ROPE pour exposer les objets COM (source xmlmag.com[34])

3.2.6 Notre avis

La rapidité d'écriture et de déploiement d'un Web Service n'a pas d'égal avec Visual Studio.NET. L'environnement a su faire ses preuves pour les applications classiques et offre un véritable plus au développeur Web grâce à sa puissance et, ce qui peut également être considéré comme un désavantage, sa forte intégration avec les autres produits Microsoft.

Malheureusement la bêta 1 commence à dater, et son implémentation de SOAP est incompatible avec les autres produits, plus rapides à la mise à jour, notamment Apache-SOAP et `Soap::Lite`. L'ajout conseillé sur un article de l'attribut `[SoapService(Style=SoapServiceStyle.RPC)]` n'a malheureusement pas amélioré la compatibilité. La version bêta 2, prévue pour juin 2001, devrait corriger cela avec l'intégration de SOAP 1.1 et de WSDL, déjà disponibles dans le SOAP Toolkit 2.0.

3.3 Les Web Services et Java

Une pléthore d'outils pour Java commence à apparaître sur Internet. Ceux-ci sont souvent commerciaux comme le serveur CapeClear[35] J2EE offrant un pont Entreprise JavaBeans vers Web Services ou l'initiative SunONE/iPlanet[36], mais des implémentations libres sont également très avancées. En raison de sa qualité et de sa pérennité, nous avons retenu l'implémentation libre d'Apache et les outils qui lui sont associés.

- **Apache-SOAP** (version testée 2.1). Livré sous la forme d'un JAR nommé `soap.jar`, Apache-SOAP offre une implémentation Open Source libre de droits, c'est-à-dire sous licence Apache, du protocole SOAP 1.1. Cette implémentation, pour la partie serveur, nécessite un serveur d'application Java comme WebSphere, Apache Tomcat, BEA WebLogic ou encore JRun. Apache-SOAP fournit par ailleurs un débuggeur SOAP agissant comme un tunnel HTTP, comme nous le verrons plus loin.

- **Apache Tomcat** (version testée 3.2). Issu du projet Apache Jakarta pour lequel Sun a fourni son implémentation des JSPs, ce serveur a acquis une solide réputation en

[34] http://www.xmlmag.com/upload/free/features/xml/2000/04fal00/kb0004/kb0004.asp

[35] http://www.capeclear.com

[36] http://www.sun.com/software/sunone/portfolio/aws.html

termes de stabilité et de performances. Nous l'avons choisi car, comme Apache-SOAP, il est libre de droits et ses sources sont disponibles.

- **WSDL Toolkit** (version testée 1.1). Fourni par IBM et disponible sous une licence limitée à 90 jours d'AlphaWorks, cette utilitaire permet de générer le squelette du serveur et du client d'un *Web Service* à l'aide de sa description WSDL.

- **Web Services Toolkit** (version testée 2.2). Egalement fourni par IBM sous licence AlphaWorks, il contient notamment Apache-SOAP et le WSDL Toolkit. Il facilite l'installation de SOAP sur Tomcat et WebSphere à l'aide d'une interface graphique et offre le support UDDI à SOAP. De plus, WSTK fournit un outil permettant de servir un composant Microsoft COM dans l'architecture SOAP-Java.

- **Linux** (Linux-Mandrake 8.0, noyau 2.4.3). Il s'agit de la version GPL (licence libre) de la distribution Linux-Mandrake, avec la dernière série de noyaux 2.4.x.

- **Java Development Kit** (version testée 1.3 de IBM). Sous Linux, le JDK d'IBM est recommandé pour ses performances exemplaires.

Nous avons également mené les tests sous architecture Windows pour tester la compatibilité avec .NET en boucle locale. Comme nous pouvions l'espérer, le fonctionnement est rigoureusement le même sur les deux plates-formes grâce à Java. On pourra déplorer la licence AlphaWorks qui empêche l'utilisation de WSTK et WSDL en production, mais nous verrons qu'il est parfaitement possible de s'en affranchir. Nous ne pouvons que souhaiter qu'IBM incorpore à terme au moins le WSDL Toolkit et l'interface de création d'un service à Apache-SOAP.

Nous ne nous intéresserons pas à l'intégration UDDI, qui sera discutée plus loin, car elle n'existe pas encore dans .NET. Tous les tests sont effectués en boucle locale, mais peuvent fonctionner en environnement distribué à condition de bien préciser au client l'adresse du serveur – rôle de UDDI le cas échéant.

3.3.1 Vue d'ensemble de l'architecture

L'installation de Tomcat[37] et de Apache-SOAP pose essentiellement des problèmes liés au `CLASSPATH` de Java. Nous recommandons de mettre tous les JAR nécessaires à Apache-SOAP[38] dans un même répertoire. La documentation fournie avec Apache-SOAP explique la marche à suivre pour interfacer Tomcat et SOAP.

[37] Disponible gratuitement sur le site http://jakarta.apache.org . Il se présente sous la forme d'un fichier ZIP d'environ 3 Mo qui contient la version Linux et Windows.

[38] *xerces.jar*, *activation.jar*, *mail.jar* et *soap.jar* provenant de http://xml.apache.org/xerces-j http://java.sun.com/products/beans/glasgow/jaf.html http://java.sun.com/products/javamail/ et http://xml.apache.org/soap/

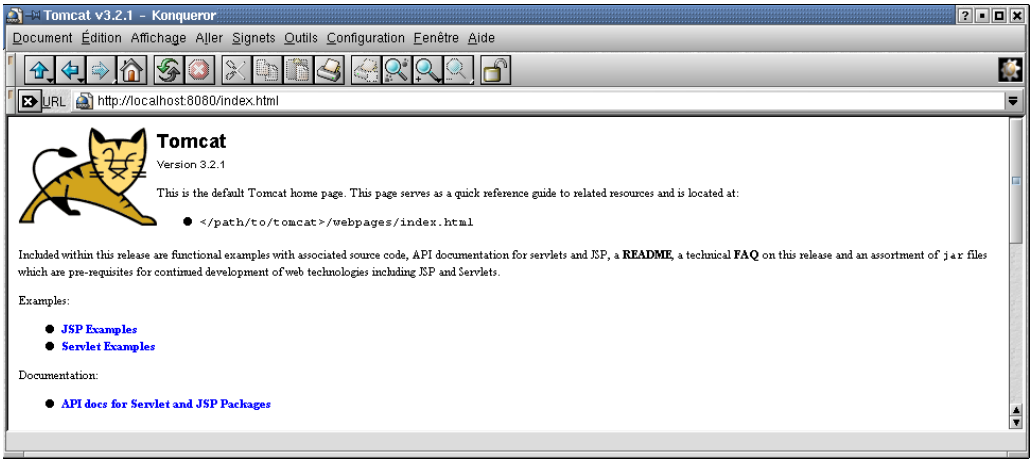


Figure 20. Le serveur d'application Apache Tomcat 3.2.1

Si l'installation s'est bien déroulée, une page d'accueil est disponible à l'adresse http://127.0.0.1/soap/admin/ .

Figure 21. La page d'accueil de Apache-SOAP

Comme on pourra le constater, le rôle d'IBM dans Apache-SOAP se voit aussi dans la documentation que dans les titres des pages.

Si le `CLASSPATH` contient le répertoire d'installation de Apache-SOAP, il est possible d'utiliser directement les exemples fournis. Le déploiement d'un service peut s'effectuer soit à partir de la page d'administration, soit à l'aide d'un descripteur XML de déploiement et d'une application Java console.

Cliquer sur « *Deploy* » ouvre la page suivante :

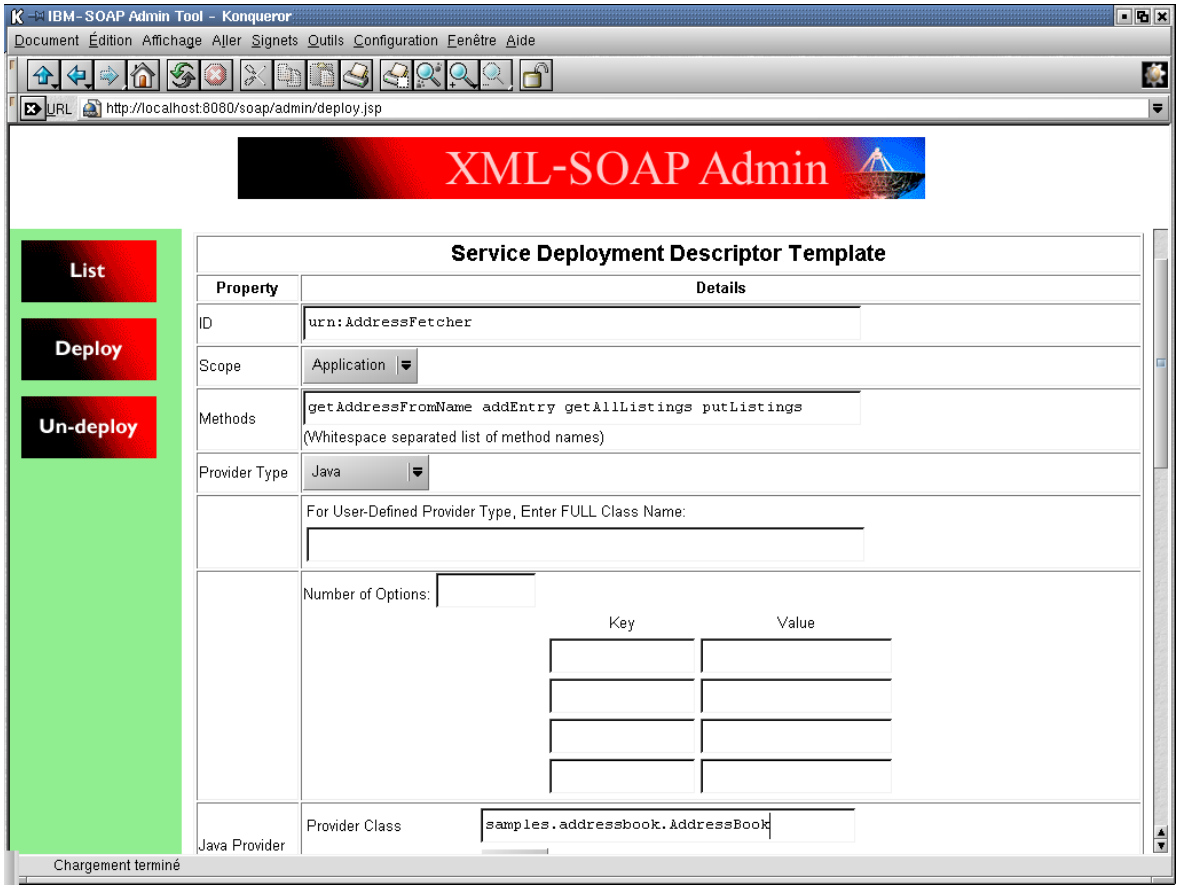


Figure 22. Interface de déploiement d'Apache-SOAP

Nous le remplissons avec les paramètres attendus par l'exemple `samples.addressbook` fourni avec Apache-SOAP.

Il est possible de s'affranchir de l'interface graphique en utilisant le descripteur de déploiement XML fourni dans le même répertoire dont voici le contenu :

```
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment" id="urn:AddressFetcher">
    <isd:provider type="java" scope="Application" methods="getAddressFromName addEntry
      getAllListings putListings">
      <isd:java class="samples.addressbook.AddressBook" static="false" />
   </isd:provider>
   <isd:faultListener>org.apache.soap.server.DOMFaultListener</isd:faultListener>
   <isd:mappings>
      <isd:map encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns:x="urn:xml-
         soap-address-demo" qname="x:address" javaType="samples.addressbook.Address"
         java2XMLClassName="org.apache.soap.encoding.soapenc.BeanSerializer"
         xml2JavaClassName="org.apache.soap.encoding.soapenc.BeanSerializer" />
      <isd:map encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns:x="urn:xml-
         soap-address-demo" qname="x:phone" javaType="samples.addressbook.PhoneNumber"
         java2XMLClassName="org.apache.soap.encoding.soapenc.BeanSerializer"
         xml2JavaClassName="org.apache.soap.encoding.soapenc.BeanSerializer" />
   </isd:mappings>
</isd:service>
```

La ligne de commande à utiliser est la suivante :

```
$ java org.apache.soap.server.ServiceManagerClient http://localhost:8080/soap/servl
et/rpcrouter deploy DeploymentDescriptor.xml
```

A ce stade, nous pouvons constater que Apache-SOAP utilise un système de nommage et d'appel différent de ASP.NET. Une servlet nommée `rpcrouter` se charge de diriger les requêtes et leurs réponses vers les bons destinataires. Elle maintient une base de données des

services SOAP disponibles et de leurs méthodes. Cette base est persistante, même si un arrêt de Tomcat survient. Cette servlet reprend l'idée du *portmapper* RPC d'UNIX en le restreignant aux services SOAP. Les services ont un espace de nommage indiqué par `urn:` à la manière des URL classiques.

Par curiosité, nous pouvons voir sur l'outil débuggeur quelles enveloppes SOAP ont été échangées pour effectuer le déploiement.

Le serveur Tomcat écoute par défaut sur le port 8080, nous avons placé le débuggeur en écoute sur le port classique HTTP (80). Ceci permet de ne plus mentionner le numéro de port dans nos requêtes. L'adresse virtuelle du routeur RPC dans ce contexte devient plus simplement `http://localhost/soap/servlet/rpcrouter`.

La commande pour rediriger le port 8080 vers 80 est la suivante :

```
# java org.apache.soap.util.net.TcpTunnelGui 80 127.0.0.1 8080
```

Notons que sous Linux, il faut avoir les droits d'administration pour effectuer cette manipulation, ou choisir un port non réservé (supérieur à 1024) à la place du port 80. Après lancement du déploiement par la méthode ligne de commande en utilisant l'adresse virtuelle du routeur RPC, les enveloppes s'affichent dans la Figure 23.

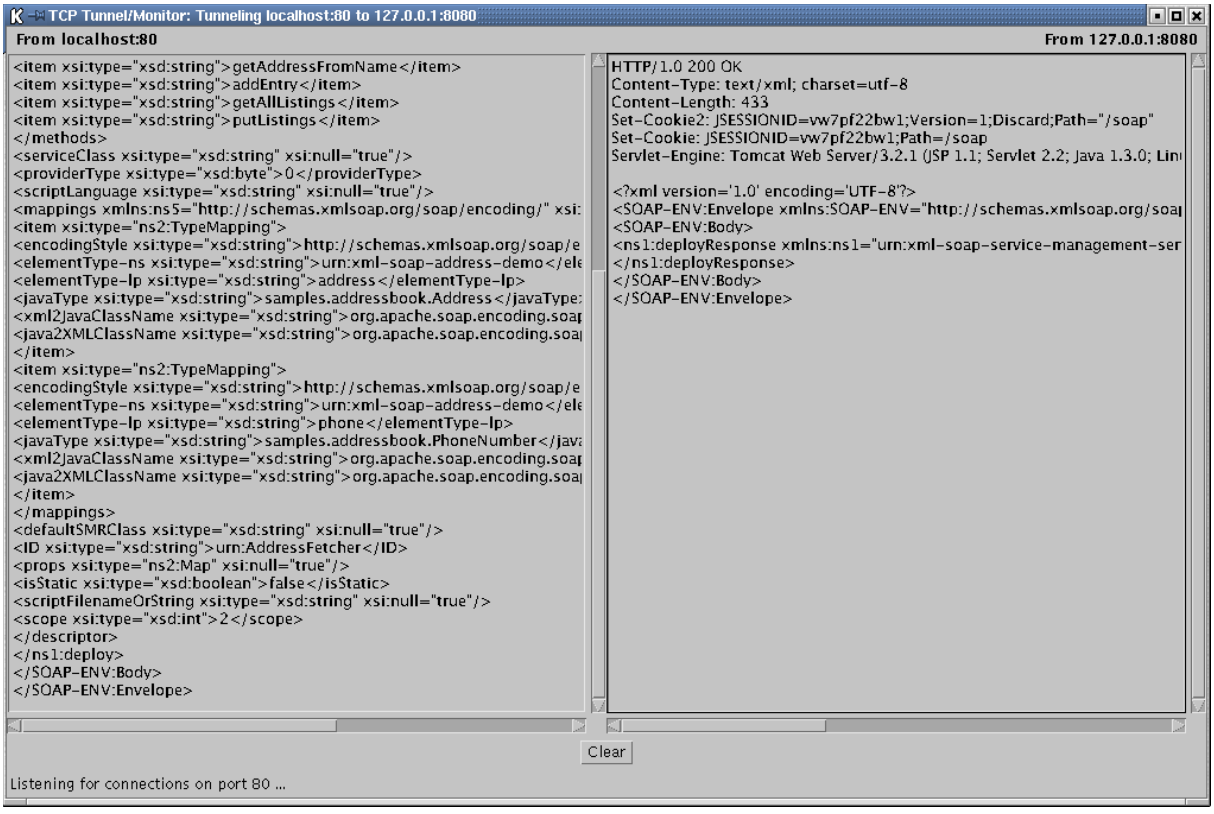


Figure 23. Les enveloppes SOAP de déploiement d'Apache-SOAP

Après déploiement, nous pouvons nous assurer que le service est bien enregistré dans la base à l'aide du bouton « *List* » de la page d'administration.

Figure 24. Liste des services déployés par Apache-SOAP

Cette liste s'obtient également par ligne de commande :

```
$ java org.apache.soap.server.ServiceManagerClient http://localhost:8080
/soap/servlet/rpcrouter list
Deployed Services:
        urn:AddressFetcher
```

Les descripteurs du service peuvent s'obtenir par ligne de commande ou en cliquant sur un service nous donne les détails de ce dernier, un peu à la manière du SDL de .NET.

```
$ java org.apache.soap.server.ServiceManagerClient http://localhost:8080
/soap/servlet/rpcrouter query urn:AddressFetcher
```

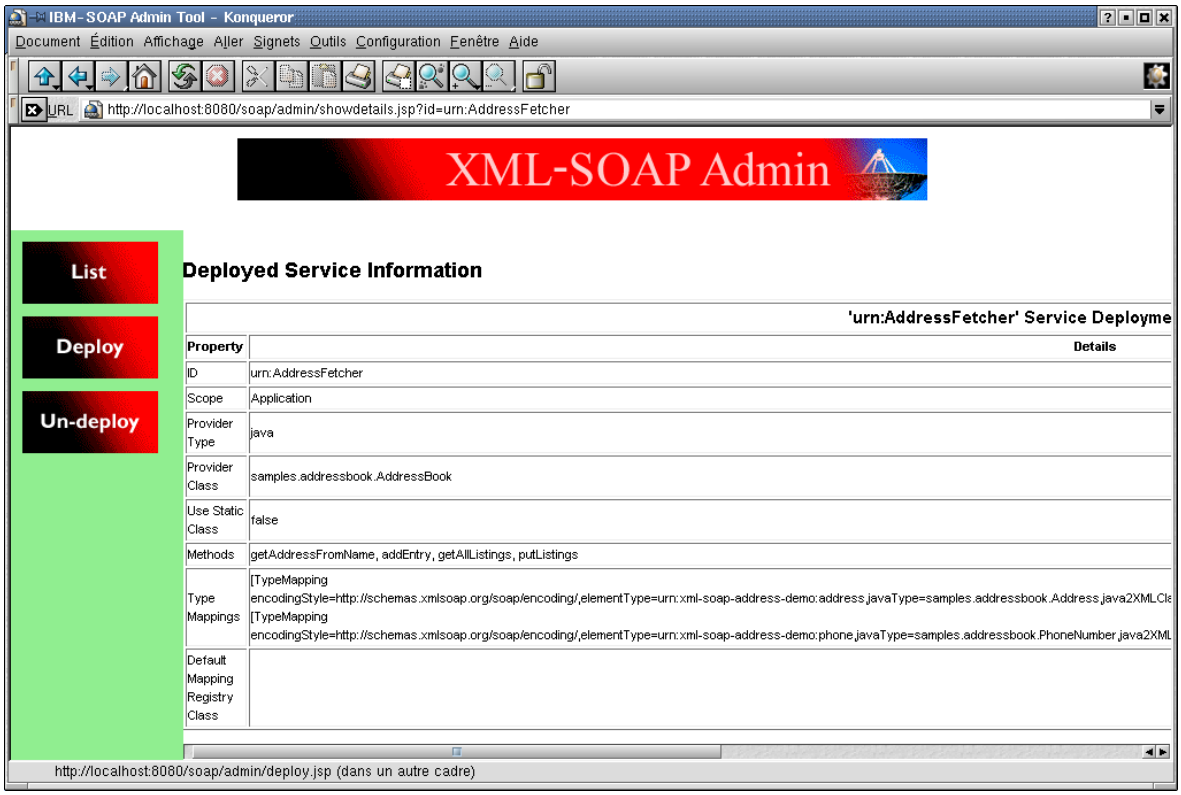


Figure 25. Informations de déploiement d'un service Web sous Apache-SOAP

Nous voyons que les détails du service reprennent dans leur quasi-totalité les paramètres de déploiement contenus dans le fichier XML.

- Les services SOAP sont dans cette architecture des Java Beans contactés par le routeur RPC. Ainsi, le paramètre *scope* (portée) correspond au tag `<jsp:useBean>` des JSPs dans `rpcrouter`. Rappelons que ce paramètre permet de gérer finement la gestion et la durée des instances des JavaBeans SOAP.
    - **page / request :** une nouvelle instance est créée à chaque appel à `rpcrouter` (lequel n'effectue pas de *forwarding*). Cette méthode n'est pas recommandée pour des raisons de performances.
    - **session :** une instance est créée pour la durée complète de la session.
    - **application :** toute page au sein de l'application peut accéder à l'objet. Des invocations successives au service appartenant à différentes sessions vont notamment partager la même instance de l'objet. Ce paramètre a un impact important sur la sécurité, alors que *page* et *request* assurent l'isolation des appels successifs. La portée *application* implique au contraire que les JavaBeans SOAP sont partagés par les différents utilisateurs du serveur.
- Les correspondances de type (*type mappings*) qui permettent au routeur RPC de sérialiser/désérialiser de Java vers XML et réciproquement. Les JavaBeans contenant les méthodes SOAP ne sont donc pas conscients de l'enveloppe SOAP, entièrement gérée par le routeur RPC.

Il est bien sûr possible d'enlever un service SOAP du référentiel à l'aide de *Undeploy* ou encore de la ligne de commande.

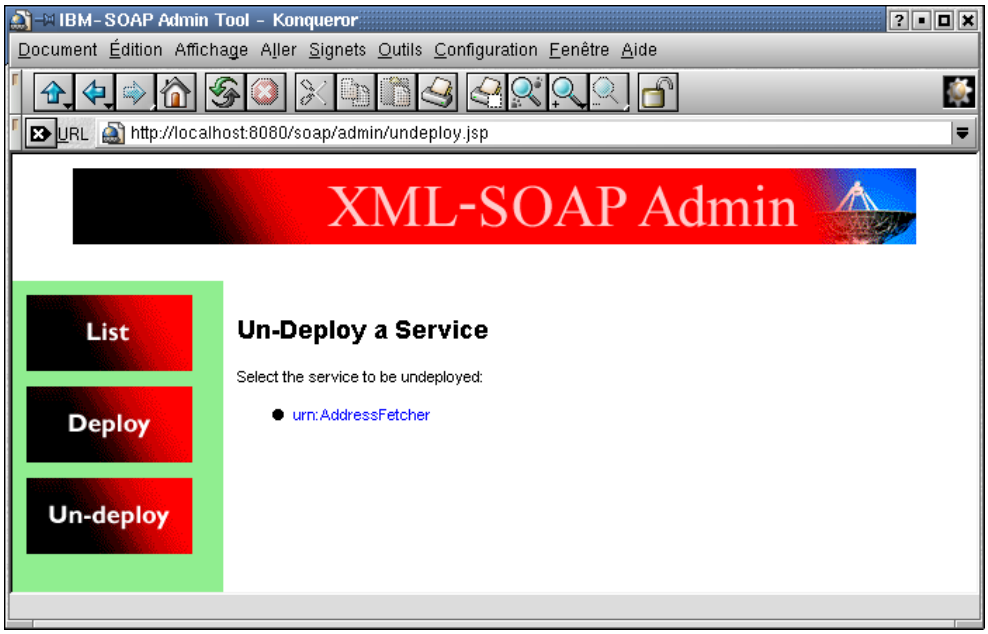


Figure 26. Désactiver un service avec Apache-SOAP

Les exemples SOAP fournissent également des clients Java. Nous choisissons `GetAddress` car ce dernier attend un paramètre.

```
$ java samples.addressbook.GetAddress http://localhost:8080/soap/servlet/
rpcrouter "John B. Good"

123 Main Street
Anytown, NY 12345
(123) 456-7890
```

La Figure 27 montre le résultat d'une redirection sur l'outil de visualisation.

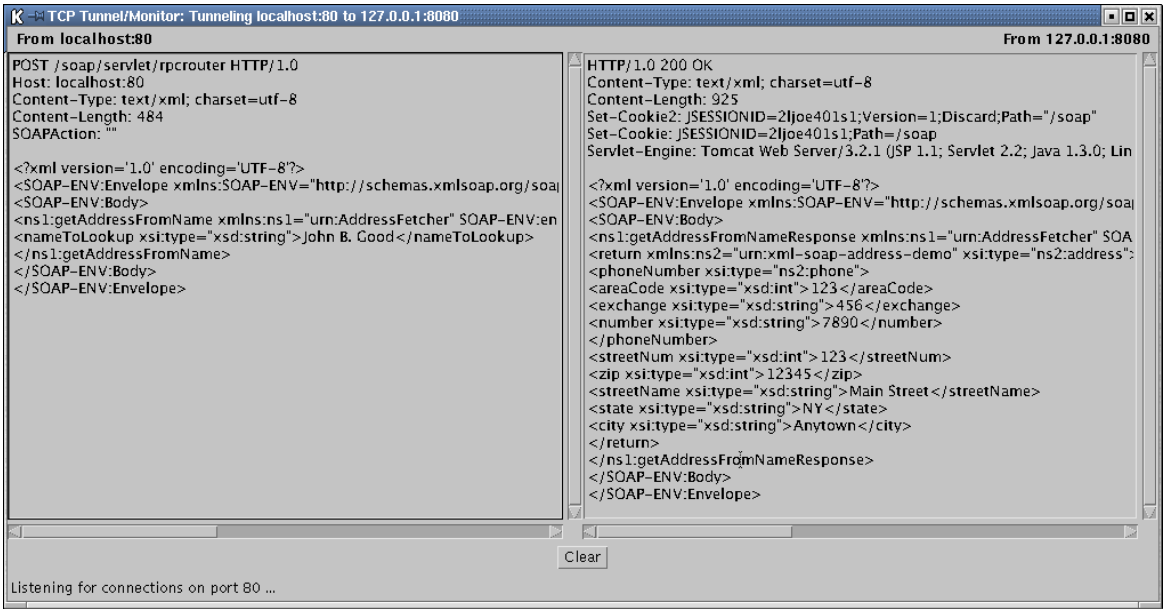


Figure 27. L'échange SOAP entre le client et le service

Maintenant que nous avons étudié l'architecture proposée par IBM, nous allons examiner son API pour regarder comment la mettre à profit. Nous utiliserons la terminologie du WSTK pour désigner les acteurs.

3.3.2 Le fournisseur de service (*Service Provider*)

Nous allons construire dans la suite un exemple `RenvoieChaine` qui n'implémentera qu'une méthode, `getMe`, prenant en paramètre une chaîne de caractères et la renverra au client.

Nous avons vu précédemment que le service peut se présenter sous la forme d'un JavaBean grâce à la simplicité de SOAP. Rappelons qu'un JavaBean est un objet Java qui expose ses propriétés avec des accesseurs `get` et `set`, utilisés pour l'introspection, et qui fournit un constructeur vide, afin de reconstruire l'objet. Voici le code très simple que nous souhaitons mettre en ligne :

```
public class RenvoieChaineService {
  // Méthode à exporter
  public String getMe(String message) {
    return "J'ai reçu : "+message;
  }
}
```

Il nous faut préciser le déploiement de ce code dans le fichier `DeploymentDescriptor.xml`. Nous proposons le descripteur suivant, en choisissant `urn:RenvoieChaine` comme identifiant de notre service :

```
  <isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
     id="urn:RenvoieChaine">
    <isd:provider type="java" scope="Application" methods="getMe">
      <isd:java class="RenvoieChaineService" />
    </isd:provider>
```

```
    <isd:faultListener>org.apache.soap.server.DOMFaultListener</isd:faultListener>
  </isd:service>
```

Nous prendrons garde à mettre le fichier `.class` dans un répertoire du `CLASSPATH`. Notons que si plusieurs méthodes étaient choisies, il suffit de les séparer par un espacement dans `methods`. Le service peut alors être déployé.

On le voit, rien n'est plus facile que de mettre le service SOAP en ligne. C'est donc bien le client qui est difficile à écrire.

3.3.3 Le consommateur (*Service Requestor*)

La difficulté principale réside dans la définition de l'IDL (*Interface Definition Language*) du service au format WSDL. Les exemples fournis montrent qu'il est tout à fait possible de s'en passer dans le cadre de Java et d'une correspondance simple, mais en toute rigueur il est nécessaire d'avoir un descripteur neutre du service dans une architecture réelle. Apache-SOAP ne fournit aucun outil générant le SDL à la demande comme le fait ASP.NET en ajoutant `?SDL` au bout de l'URL du service. Nous reviendrons dessus ultérieurement lorsque nous aborderons les outils fournis par IBM.

Nous n'avons pas besoin d'IDL puisque nous avons programmé le serveur et que les types utilisés sont simples. Le code du client génère le XML de la requête. Sa complexité apparente réside dans la liberté de codage que SOAP offre. Tout est réglé par défaut dans notre exemple.

```
import java.io.*;
import java.net.*;
import java.util.*;
import org.apache.soap.util.xml.*;
import org.apache.soap.*;
import org.apache.soap.rpc.*;

public class Client {
  static XMLParserLiaison xpl = new XercesParserLiaison ();

  public static void main (String[] args) throws Exception {
    if (args.length != 2
        && (args.length != 3 || !args[0].startsWith ("-"))) {
      System.err.println ("Usage: java " + Client.class.getName () +
                          " [-encodingStyleURI] SOAP-router-URL message");
      System.exit (1);
    }

    // Lecture des arguments.
    int offset = 3 - args.length;
    String encodingStyleURI = args.length == 3
                              ? args[0].substring(1)
                              : Constants.NS_URI_SOAP_ENC;
    URL url = new URL (args[1 - offset]);
    String message = args[2 - offset];

    // Construction de l'appel.
    Call call = new Call ();
    call.setTargetObjectURI ("urn:RenvoieChaine");
    call.setMethodName ("getMe");
    call.setEncodingStyleURI(encodingStyleURI);
    Vector params = new Vector ();
    params.addElement (new Parameter("message", String.class, message, null));
    call.setParams (params);

    // Appel : l'actionURI est vide car rpcrouter n'en a pas besoin
    // Toutefois, le SOAP Toolkit de Microsoft attendrait
    // "http://tempuri.org/RenvoieChaine"
```

```
    Response resp = call.invoke (/* router URL */ url, /* actionURI */ "" );

    // Vérification du résultat.
    if (resp.generatedFault ()) {
      Fault fault = resp.getFault ();
      System.out.println ("Erreur d'appel : ");
      System.out.println ("  Code d'erreur   = " + fault.getFaultCode ());
      System.out.println ("  Message d'erreur = " + fault.getFaultString ());
    } else {
      // Récupération de la réponse.
      Parameter result = resp.getReturnValue ();
      System.out.println (result.getValue ());
    }
  }
}
```

Ce client a été écrit pour admettre un paramètre optionnel d'encodage. On voit que ce code est réutilisable, seule les sections « construction de l'appel » et « récupération de la réponse » changent selon l'appel.

3.3.4 Fonctionnalités de l'environnement

3.3.4.1 Apache-SOAP

Apache-SOAP fournit une implémentation presque complète de SOAP 1.1 et immédiatement opérationnelle.

- Il est possible d'utiliser trois encodages : SOAP 1.1, XML pur et XMI, mais nous ne retiendrons que SOAP puisque Microsoft ne supporte que ce dernier.
- L'application cliente et le routeur RPC implémentent la sérialisation des types primitifs, des chaînes de caractères, des JavaBeans (via l'introspection), des vecteurs, des énumérations et des matrices unidimensionnelles de ces types.
- Trois moyens de transport sont proposés : HTTP, HTTPS et SMTP. Il est en effet possible d'effectuer des appels par e-mail (d'où l'usage de `mail.jar` du `CLASSPATH`). La classe `GetQuoteSMTP` des exemples illustre cette fonctionnalité. La présence de HTTPS nous rassure sur la gestion de la sécurité à partir de Apache-SOAP.
- Il est également possible d'écrire les services dans un langage de script, dont JavaScript, à l'aide du Bean Scripting Framework (BSF), également développé par IBM puis donné à la communauté.

Pour illustrer ce dernier point, examinons l'exemple `Calculator` que l'on trouvera dans le répertoire `samples`. On n'y trouve que le code du client et le descripteur de déploiement. En effet, le code du serveur est directement inclus dans ce dernier.

```
<?xml version="1.0" ?>
<isd:service   xmlns:isd="http://xml.apache.org/xml-soap/deployment"   id="urn:xml-soap-demo-
   calculator">
   <isd:provider type="script" scope="Application" methods="plus minus times divide">
      <isd:script language="javascript">function plus (x, y) { return x + y; } function minus
         (x, y) { return x - y; } function times (x, y) { return x * y; } function divide (x,
         y) { return x / y; }</isd:script>
   </isd:provider>
   <isd:faultListener>org.apache.soap.server.DOMFaultListener</isd:faultListener>
</isd:service>
```

La classe `org.apache.soap.server.InvokeBSF` de Apache-SOAP se charge de l'encapsulation des appels à BSF. Les JAR `bsf.jar` et `bsfengines.jar` doivent être inclus dans le `CLASSPATH` pour permettre l'utilisation des scripts dans Apache-SOAP.

3.3.4.2 Web Services Toolkit (WSDL Toolkit inclus)

IBM. Web Services Toolkit

Outre quelques bonnes surprises comme l'intégration COM, WSTK fournit surtout le support WSDL qui manque cruellement dans Apache-SOAP. Il est essentiel de disposer d'une définition réellement neutre des services pour l'interopérabilité.

Dans WSTK proprement dit, une application graphique nommée serviceWizard produit automatiquement à partir d'un objet Java Bean ou un EJB la description WSDL associée aux méthodes choisies. L'intérêt essentiel réside dans la gestion des équivalences (*mappings*) de types.

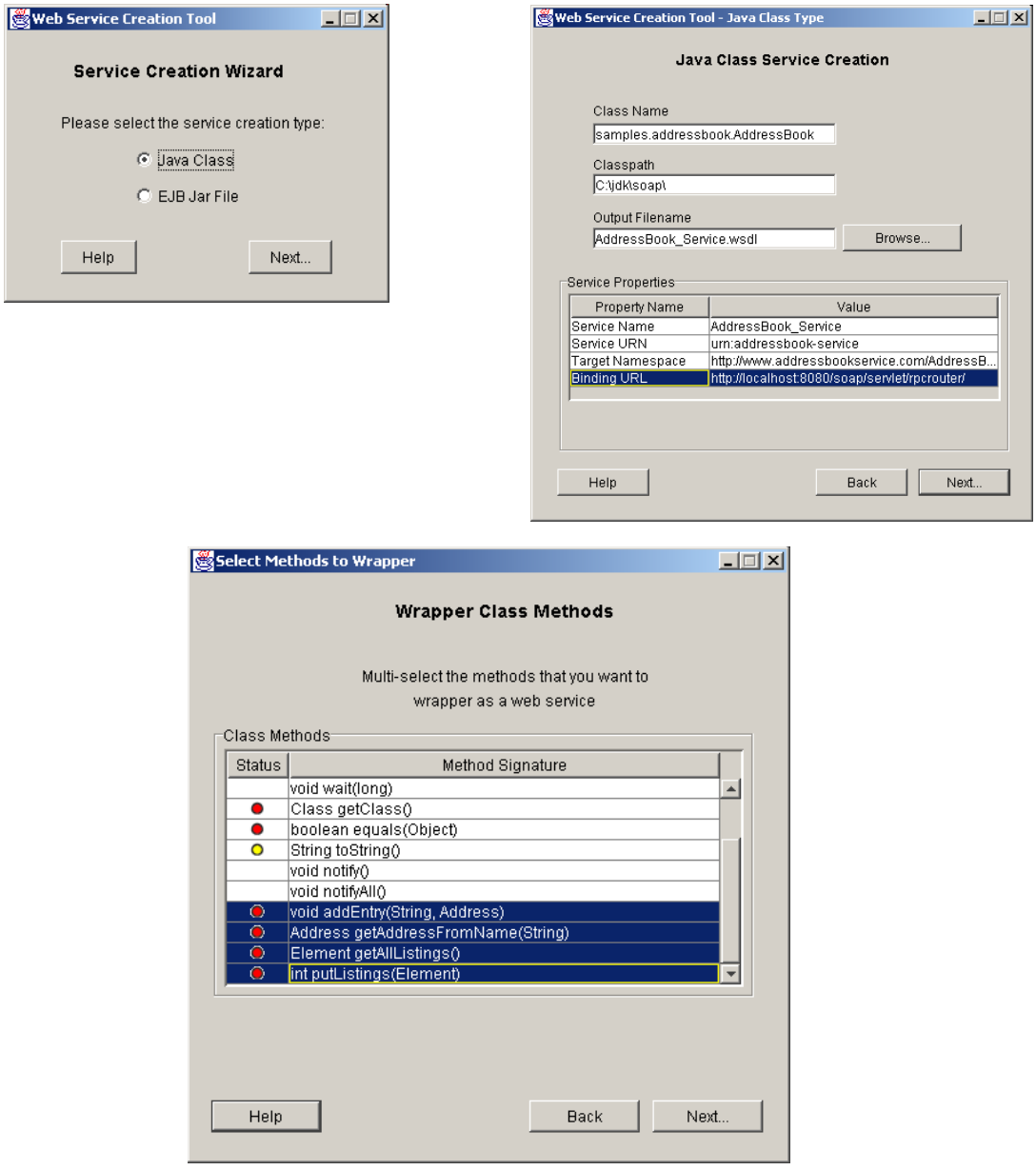


Figure 28. WSTK transforme les classes Java en services Web

Comme on le voit, l'interface graphique demande les quelques paramètres pertinents de l'application, dont l'adresse du `rpcrouter`, son `urn` et les méthodes exposées par le Web Service (un statut rouge indique que la sérialisation est « complexe » – ici, les classes `Address` et `Element` n'ont pas de *mapping* par défaut dans SOAP alors que c'est le cas pour `long` et `void`).

L'outil génère deux fichiers WSDL (implémentation et interface). Dans le fichier `impl`, Nous retrouvons notamment les descriptions complètes des types `AddressBook` et `Element` qui sont bien ce que nous attendions.

```
- <types>
  - <xsd:schema targetNamespace="http://www.addressbookservice.com/AddressBook"
      xmlns="http://www.w3.org/1999/XMLSchema/">
      <xsd:complexType name="Element" />
    - <xsd:complexType name="PhoneNumber">
        <xsd:element name="areaCode" type="xsd:int" />
        <xsd:element name="exchange" type="xsd:string" />
        <xsd:element name="number" type="xsd:string" />
      </xsd:complexType>
    - <xsd:complexType name="Address">
        <xsd:element name="streetNum" type="xsd:int" />
        <xsd:element name="streetName" type="xsd:string" />
        <xsd:element name="city" type="xsd:string" />
        <xsd:element name="state" type="xsd:string" />
        <xsd:element name="zip" type="xsd:int" />
        <xsd:element name="phoneNumber" type="tns:PhoneNumber" />
      </xsd:complexType>
    </xsd:schema>
</types>
```

Dans le WSDL Toolkit, nous trouvons un utilitaire ligne de commande qui génère les souches d'un client et d'un serveur Java à partir de la définition WSDL du service. Sa syntaxe est simple, et il autorise la création, au choix, de souches EJB ou Java Bean. Il faut également noter que l'outil ne supporte pour l'instant que les types simples de XML Schema.

L'architecture du WSTK ajoute un niveau d'abstraction sur celle de Apache-SOAP en proposant de distinguer le service de son fournisseur (*service provider*) et de l'entité qui fournit son interface (*service interface provider*). De plus, IBM introduit la notion de répertoire de service (*service registry*) que nous aborderons plus loin avec UDDI. Ce type de séparation a des connotations J2EE qui ne sont pas fortuites.

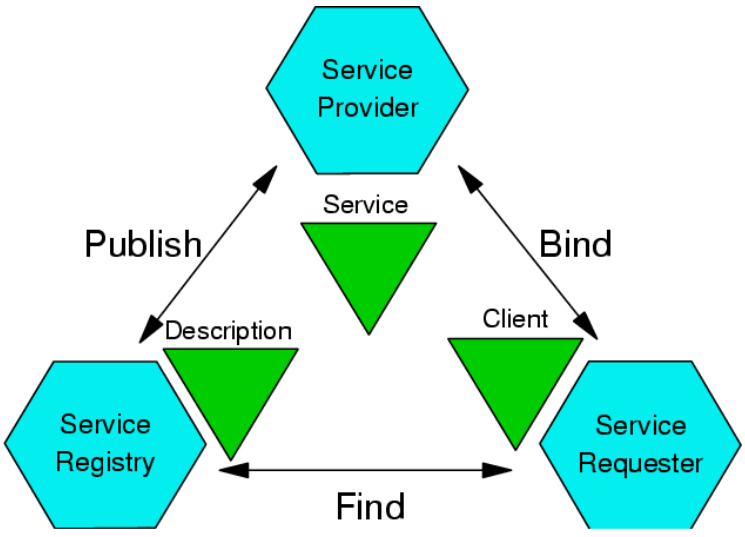


Figure 29. Le modèle IBM des services Web

Cette architecture est considérée comme plus évolutive que celle Microsoft .NET en ce qu'elle offre les avantages suivants :

- Un tiers normalisateur peut définir un service pour une tâche, activité, secteur ou application donnés par son interface. Chacun peut alors proposer le service avec une implémentation différente tout en restant compatible.
- L'emplacement physique du service, son lieu d'hébergement, n'a pas d'intérêt pour le consommateur du service. UDDI propose un service DNS pour les services.

Le modèle actuel de Microsoft fait la confusion entre le service, son fournisseur et son interface, mais est bien plus aisé à mettre en œuvre puisqu'un service est essentiellement égal à une URL et que chacun peut directement interroger le service pour obtenir son interface. De plus, Microsoft participe à l'élaboration de UDDI et supportera donc ce protocole dans la version finale de .NET.

Notons enfin qu'IBM va bientôt commercialiser un environnement graphique complet de développement nommé Web Services Development Environment (WSDE) ; l'environnement sera intégré dans WebSphere 4.0 qui sera disponible à la fin du mois de juin 2001. D'après 01net[39], le support par IBM des services Web comportera :

- WebSphere Application Server 4.0 : Invocation de services Web via SOAP ; interfaçage avec un annuaire UDDI ; proxy WSDL.
- WebSphere Studio Technology Preview for Web services : Permettra notamment de transformer un composant existant en service Web.
- WebSphere Business Integrator : Intégration B to B. Transport de SOAP sur MQSeries ; création de WSFL (Web Services Flow Language), un langage de description du workflow d'intégration issu d'un échange collaboratif.
- DB2 7.2 : Interfaçage à un annuaire UDDI. Un service Web accédera à des données DB2 via SOAP.
- Tivoli Web Services Manager : Administration des performances des services Web.
- Lotus Web Services Enablement Kit : Ce kit permettre d'exposer les fonctions de collaboration des outils Lotus selon le langage WSDL et de les accéder via SOAP.

3.3.4.3 La sérialisation des objets Java

Apache-SOAP fournit une classe `org.apache.soap.encoding.soapenc.BeanSerializer` qui implémente les interfaces `Serializer` et `Deserializer`. Cette classe permet de créer une représentation XML d'un JavaBean et réciproquement de transformer une représentation XML valide en une instance du JavaBean.

Rappelons qu'un JavaBean est une classe disposant d'un constructeur vide et d'accesseurs `get` et `set` publics. Ces accesseurs doivent permettre de reconstruire l'état interne de l'objet, mais cela ne peut pas être imposé par le compilateur, c'est donc un contrat moral.

[39] http://www.01net.com/rdn?oid=150401

Pour le serveur, aucun effort particulier n'est nécessaire. Le client, en revanche, doit créer une instance du `BeanSerializer` pour reprendre le code. Notons que dans le cadre de Apache-SOAP, le client doit disposer au préalable de l'interface ou du code de la classe visée. Microsoft .NET peut créer une classe équivalente, réduite aux propriétés.

Le WSDL Toolkit de IBM permet d'obtenir les classes Java à partir du fichier WSDL du `serviceWizard`.

```
public class Address{

          //instance variables
          public int streetNum_Elem;
          public java.lang.String streetName_Elem;
          public java.lang.String city_Elem;
          public java.lang.String state_Elem;
          public int zip_Elem;
          public PhoneNumber phoneNumber_Elem;

          //constructors
          public Address () { }

          public Address (int streetNum_Elem, java.lang.String streetName_Elem,
java.lang.String city_Elem, java.lang.String state_Elem, int zip_Elem, PhoneNumber
phoneNumber_Elem) {
                    this.streetNum_Elem = streetNum_Elem;
                    this.streetName_Elem = streetName_Elem;
                    this.city_Elem = city_Elem;
                    this.state_Elem = state_Elem;
                    this.zip_Elem = zip_Elem;
                    this.phoneNumber_Elem = phoneNumber_Elem;
          }
}
```

La classe `PhoneNumber` est également crée.

```
public class PhoneNumber{

          //instance variables
          public int areaCode_Elem;
          public java.lang.String exchange_Elem;
          public java.lang.String number_Elem;

          //constructors
          public PhoneNumber () { }

          public PhoneNumber (int areaCode_Elem, java.lang.String exchange_Elem,
java.lang.String number_Elem) {
                    this.areaCode_Elem = areaCode_Elem;
                    this.exchange_Elem = exchange_Elem;
                    this.number_Elem   = number_Elem;
          }

}
```

Les variables de `Address` ne correspondent étrangement pas exactement aux propriétés de la classe initiale puisque WSDL ajoute `_Elem` à chaque nom. Pour comparer, nous fournissons ci-dessous le squelette de la classe `Address` originale.

```
public class Address
{

  public Address()
  public Address(int streetNum, String streetName, String city, String state,
                 int zip, PhoneNumber phoneNumber)

  public void setStreetNum(int streetNum)
  public int getStreetNum()

  public void setStreetName(String streetName)
  public String getStreetName()

  public void setCity(String city)
```

```
  public String getCity()

  public void setState(String state)
  public String getState()

  public void setZip(int zip)
  public int getZip()

  public void setPhoneNumber(PhoneNumber phoneNumber)
  public PhoneNumber getPhoneNumber()

}
```

3.3.5 Notre avis

Apache-SOAP est un très beau cadeau d'IBM pour la communauté. L'architecture suit un modèle éprouvé dans le monde Java et l'implémentation est hors de tout soupçon quant à la qualité. De fait, Apache-SOAP a même une petite longueur d'avance sur son concurrent plus médiatique pour l'adhérence aux standards. Les utilisateurs de Java pourront continuer à s'interfacer avec le monde extérieur en gardant leurs infrastructures et leurs outils, ce qui était le but avoué d'IBM dont le soutien de J2EE dans la gamme WebSphere est important.

Le point noir du tableau est l'absence des outils WSDL décrits précédemment dans la distribution Apache. Les intentions d'IBM à leur sujet ne sont pas claires, il est en effet tout à fait possible qu'ils soient inclus à Apache-SOAP à terme. Heureusement, d'autres membres de la communauté implémentent avec des licences GPL et LGPL[40] des outils comparables à WSDL Toolkit, le plus avancé en Java étant Idoox[41].

Nous regrettons principalement trois fonctionnalités de ASP.NET :

- La possibilité d'accéder à un service directement par son adresse. Les développements sont accélérés et les services prennent une forme plus concrète. Dans l'architecture Apache-SOAP, tous les services ont la même URL, celle du routeur RPC, et sont reconnus uniquement par des noms virtuels URN.

- La possibilité d'interroger directement le service pour obtenir son descripteur SDL. Avec Apache-SOAP, on peut au mieux l'implémenter « à la main » puisque le support d'une telle fonctionnalité n'est pas prévu. Les protocoles DISCO et UDDI vont bientôt résoudre partiellement cette question, à condition que la communauté et/ou IBM en fournissent une implémentation libre.

- La simplicité de déploiement et de test des applications ASP.NET à partir de l'environnement Visual Studio.NET. Certes des scripts en ligne de commande peuvent faciliter la publication des Web Services Java, mais force est de constater que ASP.NET n'a pas besoin de descripteur de déploiement complexe.

Pour déroger à certaines habitudes, nous constatons qu'il est ainsi plus facile d'écrire un client Java pour accéder à un serveur Microsoft que l'inverse.

---

[40] Licences Open Source favorisant l'échange et le partage du code source décrites sur le site du projet GNU www.gnu.org

[41] http://www.zvon.org/index.php?nav_id=35 ou www.idoox.com . La licence et le code source sont difficiles à trouver sur les sites, aussi nous préférons émettre certaines réserves quant à la gratuité du projet.

3.4 Interopérabilité

Contrairement aux idées reçues, la solution libre Apache-SOAP 2.1 implémente la majeure partie de la spécification SOAP 1.1, alors que la version de Visual Studio.NET dont nous disposons (la plus récente) n'implémente que SOAP 1.0. Le SOAP Toolkit 2.0 pour Visual Studio 6.0 implémentant SOAP 1.1 est depuis peu disponible au téléchargement à l'adresse msdn.microsoft.com/soap, mais malheureusement ne fonctionne pas avec Visual Studio.NET. Il semble probable en revanche que Microsoft va l'inclure dans la bêta 2.

Ceci a pour conséquence d'empêcher l'interopérabilité entre ces deux logiciels.

3.4.1 Web Service Definition Language (WSDL)

Microsoft.NET va supporter dès la version bêta 2 un langage de description des services standardisé nommé WSDL (Web Service Description Language), comparable au SDL de la bêta 1. Pour ce standard, Microsoft s'est entendu avec IBM et Ariba, ce qui permet de penser qu'il va s'imposer dans le monde des services Web.

Son rôle est exactement identique à celui des interfaces IDL de CORBA, c'est-à-dire fournir une description complète et neutre des services afin de permettre de créer un client ou un serveur dans le langage et l'architecture choisie. Comme un compilateur IDL, `WebServiceUtil` et WSDL Toolkit génèrent les souches des clients et des serveurs, ce qui simplifie le développement pour le programmeur. Nous ne pouvons qu'espérer que des versions libres seront disponibles, car ces outils deviennent rapidement indispensables.

3.4.2 Universal Discovery, Description, and Integration (UDDI)

Véritable annuaire des services Web, UDDI est un projet qui comptait 36 membres en septembre 2000 contre plus de 260 aujourd'hui. Bien que les spécifications prévoient une série d'au moins trois versions[42], Microsoft, IBM et Hewlett Packard proposent déjà un annuaire sur leurs sites comme le montre la Figure 30.

L'objectif visé par UDDI est de favoriser le commerce B2B (Business-to-business, c'est-à-dire inter-entreprise) en proposant trois annuaires dont le rôle est de décrire (*Description*), de découvrir (*Discovery*) les services Web et d'y accéder (*Integration*) :

- Les « pages blanches » (*businessEntity*) décrivent l'entreprise par sa raison sociale (nom, adresse, téléphone).
- Les « pages jaunes » (*businessService*) donnent sa classification, son activité notamment.
- Les « pages vertes » (*bindingTemplate*) fournissent la description des services de l'entreprise – sa page d'accueil, d'emplois mais surtout ses services Web.

[42] La version 2.0 est prévue pour juillet 2001, la version 3.0, qui devrait être compatible avec eSpeak de HP, pour la mi-2002.

L'annuaire sera proposé par plusieurs entreprises sur leurs propres sites, mais sera régulièrement synchronisé pour garantir son unicité. Seule l'interface pourra donc éventuellement différer.

On le voit, UDDI devrait permettre à terme aux entreprises d'échanger des services plus facilement. D'après Marc Gardette, le responsable marketing développeurs de Microsoft France, « UDDI est à comparer à un serveur DNS : il s'agit d'un service de base, qui doit donc rester gratuit ». Ce dernier facteur pourrait bien lui assurer un succès immédiat dès le déploiement des services Web.

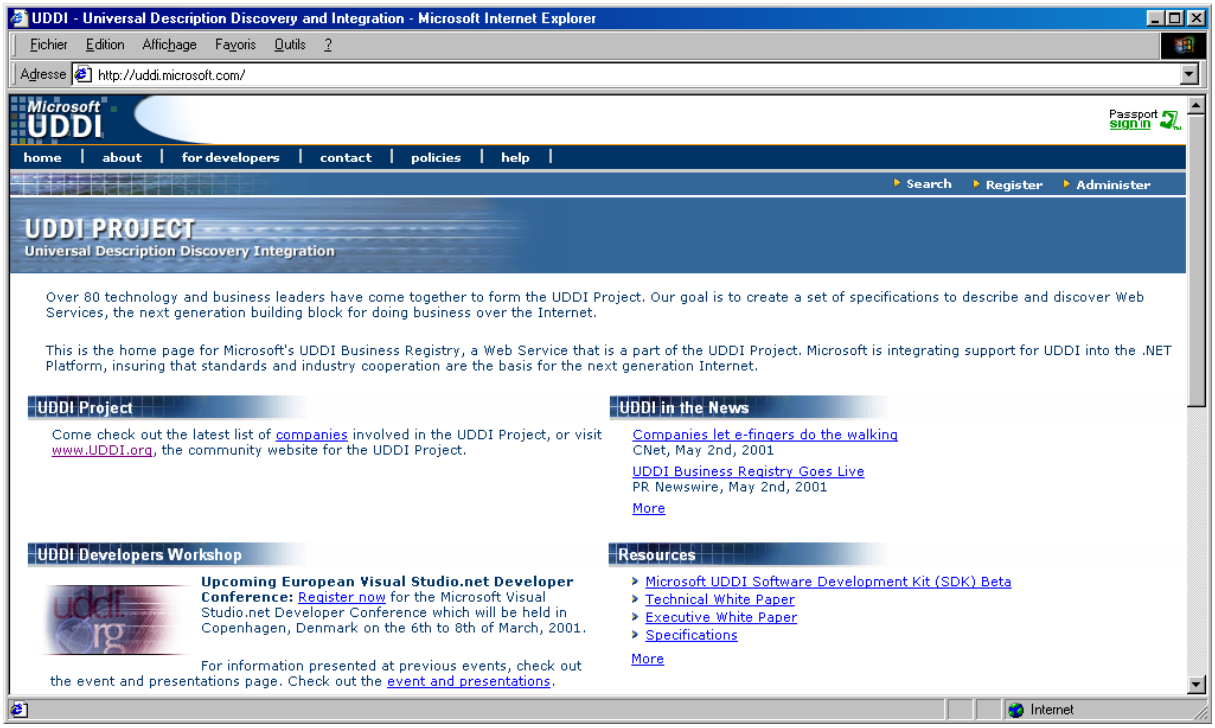


Figure 30. L'annuaire UDDI de Microsoft

4 Conclusion

4.1 Une critique de la « vision »

La tentation est grande de voir dans .NET un moyen pour Microsoft, comme pour l'industrie informatique, d'avoir de nouvelles architectures, de nouveaux logiciels, de nouveaux composants à vendre et très prochainement à louer.

Avec les Web Services, la location est un objectif central pour l'industrie du logiciel. Microsoft est en train de créer Personal.NET, un service de location de services. Comme l'indique un communiqué officiel, la branche Personal Services Group, responsable de Personal.NET, « sera chargée du développement d'un service haut de gamme par abonnement destiné aux particuliers, incluant les services MSN actuels ainsi qu'une offre s'appuyant, entre autres, sur des applications telles l'Encyclopédie Encarta, le gestionnaire de finances personnelles Money, ou le logiciel de traitement d'images Picturelt! ».

En effet, certains aspects de la vision de Microsoft exposée en page 6 nous paraissent discutables. Si l'initiative .NET se développe comme le souhaite Microsoft, les questions de propriété, de liberté et de vie privée reviendront au goût du jour. Aujourd'hui, l'utilisateur de Word possède physiquement ses documents sur son disque dur ou une disquette, mais qu'en sera t-il demain ? Où seront-ils, à qui seront-ils vraiment et à quels risques de piratage il s'expose ? Est-ce que son ordinateur sera inutilisable sans connexion au réseau ? La communauté du logiciel libre s'inquiète à juste titre de l'enfermement des données personnelles dans les formats propriétaires comme `.doc` ; le risque est que demain il faille payer pour en faire usage et qu'elles soient confinées dans un serveur distant, restreintes à un service donné.

Ces questions méritent d'être posées car nous ne devons pas céder au déterminisme technologique, qui pose en principe que la technologie induit nos besoins, et non l'inverse.

4.1.1 Le contre-exemple (fictif) de Office.NET

Après Office 95, 97 puis 2000, il semblait difficile d'annoncer de nouvelles fonctionnalités réellement novatrices ou nécessaires dans Office XP et bientôt Office.NET. Nous sommes peu convaincus de l'intérêt de louer une application telle que Word et encore moins de devoir être connectés en permanence pour que tout fonctionne normalement. A quoi vont servir les giga hertz des processeurs x86 actuels si le PC devient un Minitel amélioré ? Que se passe t-il si le réseau est engorgé ou que des serveurs essentiels tombent en panne ?

Dans leurs discours, Microsoft et Sun, avec *StarPortal* notamment, mettent en avant plusieurs arguments :

- L'élimination des mises à jour des postes clients. L'intérêt de mettre à jour Office aujourd'hui est déjà très faible, d'autant que le logiciel est censé fonctionner sans erreurs et fournit d'origine bien trop de fonctionnalités ou de *clipart* pour l'utilisateur même exigeant.

- Réduction des coûts de déploiement et de maintenance. Certes, il faut du personnel pour installer Office et parfois pour résoudre ses pannes, ce qui a un coût dans toute entreprise aujourd'hui, mais il restera nécessaire de conserver des techniciens pour

gérer les postes clients, le réseau et également régler les erreurs commises sur le serveur par les employés.

- Possibilité de payer en fonction de ses besoins. La location promet d'importantes économies aux petits consommateurs, mais son modèle semble plutôt favorable finalement aux éditeurs[43], qui en tirent un abonnement, qu'aux utilisateurs, lesquels ne possèdent plus leur outil de travail.

Certes il faut relativiser, il est en effet fort probable que seuls certains services facultatifs nécessiteront une location et une connexion permanente, mais il n'en reste pas moins que la tendance de fond de l'industrie penche en faveur de la location et que les services Web sont un moyen de la mettre en œuvre.

4.1.2 HailStorm : un portail de services hégémonique ?[44]

HailStorm est, pour Microsoft, la première incarnation de sa stratégie .NET. HailStorm est une plate-forme destinée à offrir des services aux particuliers. Comme dans sa stratégie Windows, Microsoft souhaite posséder l'infrastructure en fournissant la « plomberie » (*plumbing*) nécessaire, à charge pour des tiers de déployer leurs services à l'aide des outils Microsoft. Ainsi que nous l'avions déjà annoncé, le service Web Passport jouera un rôle déterminant dans le dispositif.

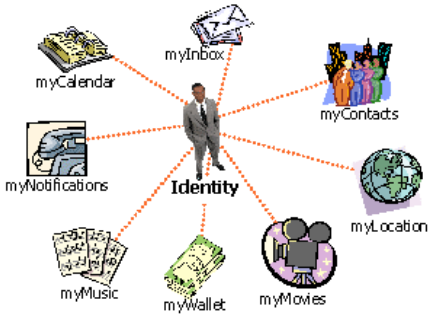


Figure 31. L'authentification par Passport est centrale (illustration Microsoft)

Comme l'illustre la Figure 31, Passport permettra aux services affiliés à HailStorm de connaître l'identité de leurs visiteurs. Mieux encore, le système centralisé mémorisera les numéros de carte de crédit, les adresses de livraison, les préférences, etc. Un système de micro-paiement où Microsoft jouerait un rôle comparable à celui de France Télécom avec le Minitel pourrait très bien apparaître, puisque la compagnie de Redmond compte prochainement facturer des abonnements aux possesseurs de Windows ou de Office. Ceci faciliterait grandement le commerce électronique, mais à l'intérieur de la plate-forme HailStorm. Si l'objectif de Microsoft est atteint et que HailStorm devient important, les sites de e-commerce, mais aussi les sites gratuits qui désirent facturer certains services pour survivre, n'auront très probablement pas d'autre alternative que d'être affiliés pour survivre, créant un cercle vertueux pour Microsoft, vicieux pour ses concurrents.

[43] Voir l'article de 01net « Microsoft, loueur au prix fort », http://www.01net.com/rdn?oid=148934&rub=1696

[44] Réflexion inspirée de l'article de Clay Shirky sur OpenP2P (site de l'éditeur O'Reilly) à l'adresse http://www.openp2p.com/pub/a/p2p/2001/05/30/hailstorm.html

Microsoft s'appuie sur son portail MSN, dont Hotmail, pour déployer rapidement HailStorm. La plate-forme est prévue pour être ouverte, au sens où elle communiquera uniquement par SOAP et où des serveurs « externes » pourront facilement s'interfacer. L'objectif de Bill Gates est de fournir du contenu pour tous les terminaux envisageables, afin d'atteindre une part de marché des transactions sur Internet importante et faire de Microsoft l'acteur dominant des services Web.

4.1.3 La croissance du haut débit est présupposée

Rien n'indique que l'Internet haut débit va réellement se développer étant donnés les goulets d'étranglements qu'il engendrera avec sa croissance. Ainsi, le volume d'*upload* vers les serveurs est actuellement limité par les opérateurs câblés avec un profil d'utilisation actuel bien moindre que ce que Microsoft espère demain. Les opérateurs pourront-ils suivre en maintenant le montant de la facture raisonnable ?

La mise en ligne d'applications pour mobiles posera également le problème de la double facturation. D'une part, il faut payer Microsoft ou un partenaire pour le service et d'autre part, il faut payer l'opérateur de téléphone mobile pour la connexion, qui devrait être facturée au volume de données pour le GPRS et l'UMTS, et non plus à la durée comme aujourd'hui. L'avalanche des factures à la fin du mois, déjà conséquente à la fin du XX^{ème} siècle, pourraient bien décourager de nombreux utilisateurs et entreprises à l'aube d'un troisième millénaire friand de location.

4.2 Un succès prévu d'avance

En cohérence avec une stratégie pratiquée avec succès depuis des années, Microsoft tente de s'imposer dans un domaine où il avait laissé l'initiative aux concurrents, Java notamment. L'extraordinaire déploiement technique, commercial et stratégique de ressources que Bill Gates a consenti pour réaliser sa vision commence à porter ses fruits, puisque .NET est probablement le *vaporware* le plus attendu depuis un an. La presse spécialisée relaie la plupart des déclarations des relations publiques de la firme de Redmond et des développements s'appuyant sur la bêta 1 sont déjà en cours.

.NET répond en effet à certaines attentes importantes du marché que la concurrence n'avait pas tout à fait comblé :

- Les services Web. Microsoft innove réellement avec la première plate-forme intégrant SOAP et des outils XML de première classe.

- Un environnement de développement Web puissant mais simple d'utilisation. Nous avons déjà évoqué tout le bien que nous pensons de Visual Studio.NET, qui devrait intéresser de nombreuses PME notamment.

- Des outils intégrés pour Windows adaptés à nouvelles exigences des développements Web. Java supporte Windows, mais rend inutilisables le serveur IIS, l'Active Directory, COM+ ou encore Windows DNA, de telle sorte que Windows est souvent utilisé comme plate-forme de développement Java, le déploiement étant ensuite effectué sur UNIX ou Linux, ce qui était d'ailleurs un objectif stratégique de Sun. Les entreprises utilisant exclusivement Microsoft, souvent pour des raisons de simplicité et de coût, avaient besoin d'outils à jour pour affronter les problématiques de demain.

- Le support de nombreux langages de programmation. Le geste fait par Microsoft en faveur des autres langages pourrait lui attirer la sympathie des développeurs marginalisés par Java. Il faut toutefois relativiser le support multilingue en raison du poids du CLS d'une part, et de la complexité de maintenir un projet écrit en plusieurs langages d'autre part.

- Le goût de la nouveauté. Il ne faut pas négliger le « cool effect », qui met en valeur les derniers produits disponibles, surtout quand ils se font tellement attendre et qu'ils demeurent largement mystérieux.

De plus, .NET s'appuie sur des qualités techniques indiscutables. Apparemment, Microsoft semble décidé à ne le livrer que lorsqu'il sera suffisamment mature, ce qui tranche avec des pratiques malheureusement courantes dans l'industrie informatique en général. La version bêta 1, contre toute attente, a fonctionné sans bugs visibles puisque seul le processus d'installation a posé quelques problèmes.

De .NET, nous retiendrons trois innovations majeures qui nous semblent mériter les louanges. Nous souhaitons de tout cœur pouvoir les retrouver sur d'autres plates-formes, voire dans une implémentation libre sur la base des soumissions à l'ECMA :

- Visual Studio.NET. L'environnement simplifie réellement les développements Web en permettant de les considérer comme des applications classiques, avec toute la puissance et l'expressivité que l'on peut attendre.

- Les services Web. Leur programmation et leur déploiement à l'aide des attributs et des assemblages sont intuitifs et s'intègrent à la plate-forme facilement.

- Le langage C#. Reprenant le meilleur de Java et de C++, C# est très expressif, souple et élégant. Bien qu'il ne soit pas au goût de certains puristes, il nous a séduit.

Bibliographie et liens utiles

- « **C# et .NET** » par Gérard Leblanc, Eyrolles (ISBN : 2-212-09278-4). Livre en français par l'auteur d'un excellent ouvrage sur *C++ Builder* de Borland. Des chapitres importants sont gratuitement téléchargeables à l'adresse http://www.dotnet-fr.org/sections.php3?op=viewarticle&artid=9

- Le site de Microsoft (http://www.microsoft.com/net/) propose des présentations généralistes de .NET exposant la vision et les principes. Une FAQ très intéressante est notamment disponible.

- Beaucoup de ressources développement .NET sur MSDN (Microsoft Developer Network) à l'adresse msdn.microsoft.com/net et www.gotdotnet.com (site géré par Microsoft)

- Le portail eServices de Office XP : http://office.microsoft.com/default.aspx

- Article de synthèse intitulé « Applying .Net to Web Services » : http://www.webtechniques.com/archives/2001/05/jepson

- Spécifications complètes soumises à l'ECMA (C#, CLI, base class library) http://msdn.microsoft.com/net/ecma/site.zip

- Portail francais sur .NET réunissant de nombreux articles à http://www.dotnet-fr.org

- Article technique sur Eiffel# : msdn.microsoft.com/library/techart/pdc_eiffel.htm

- Interview « Deep Inside C# » de Anders Hejlsberg à l'adresse http://windows.oreilly.com/news/hejlsberg_0800.html

- La liste exhaustive des changements entre la version bêta 1 et la bêta 2 sur le site 123ASPX : http://www.123aspx.com/b1to2changes/default.asp

- La spécification de la SOAP 1.1 à l'adresse http://www.w3.org/TR/SOAP

- La sécurité des services Web selon Microsoft http://msdn.microsoft.com/vstudio/nextgen/technology/security.asp

- Les Web Services dans SunONE : www.sun.com/software/sunone/portfolio/aws.html

- Les ressources XML pour Apache http://xml.apache.org

- Le site AlphaWorks d'IBM consacré aux services Web (un PDF de présentation est inclus à WSDL) http://www.alphaworks.ibm.com/tech/webservicestoolkit

- L'article de Clay Shirky sur HailStorm http://www.openp2p.com/pub/a/p2p/2001/05/30/hailstorm.html

Table des illustrations