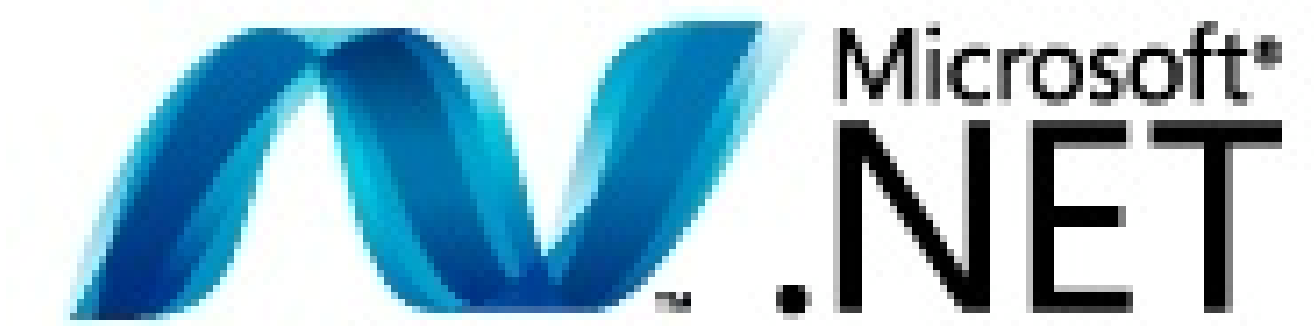


.NET : concepts

Romain Reuillon / Luc Touraille

Motivation

- .NET se présente comme une vision, celle de la prochaine génération d’Internet, fondée sur les standards ouverts, l’interopérabilité.
- Approche unifiée pour la conception d'applications Web et Windows.
- Réponse à la montée de Java EE.

Motivation

- Faciliter le développement d'applications riches (UI, BdD, identification, communication).
- Permettre l'intéropérabilité :
  - entre applications .NET (spécifications communes à tous les langages .NET)
  - entre applications hétérogènes (programmation distribuée : services web, remoting...)

.NET

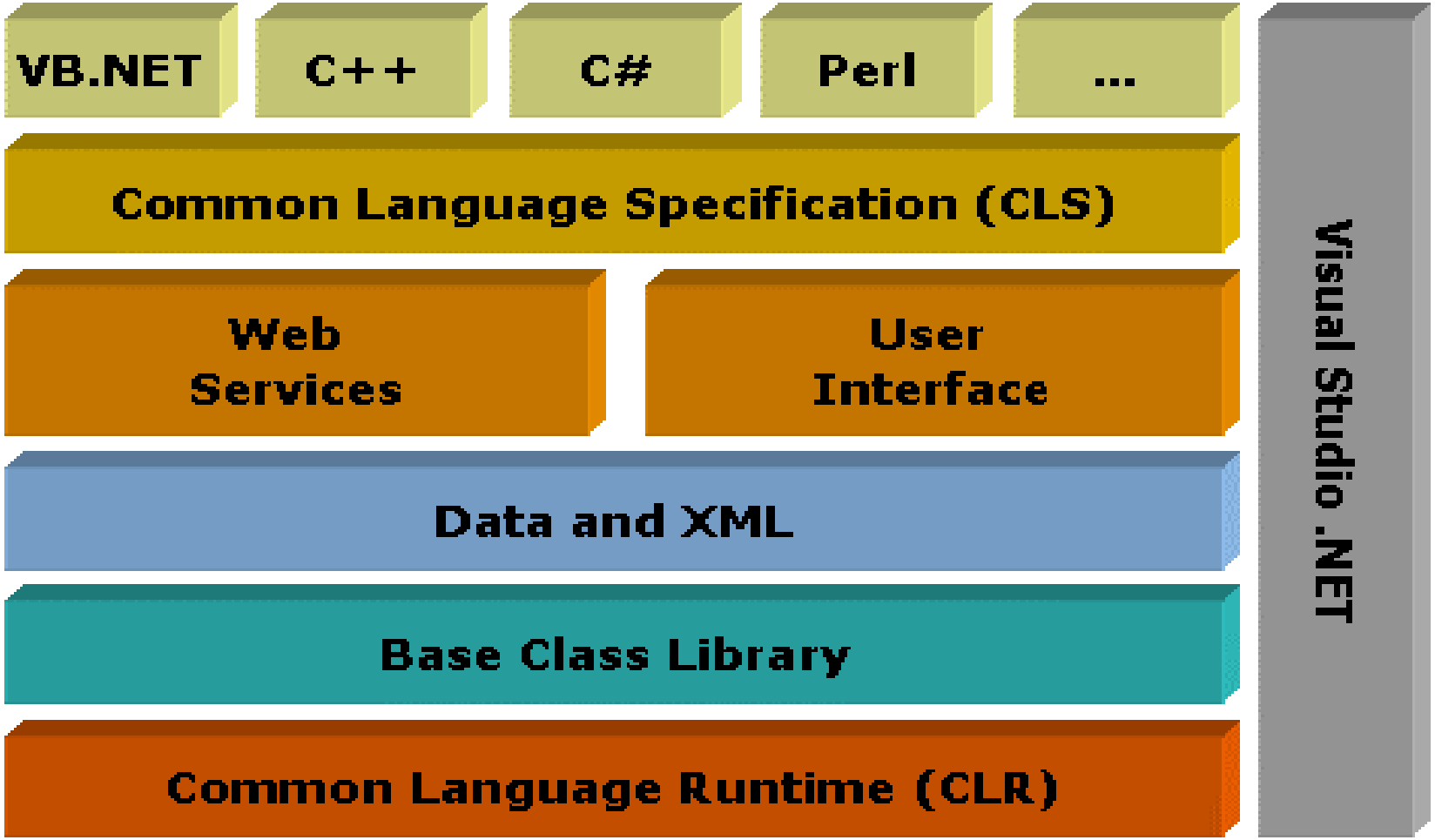
- Environnement appelé *.NET Framework* qui est distribué gratuitement ⇔ JRE
  - une machine virtuelle avec ramasse-miettes (*Common Language Runtime* - CLR)
  - Code intermédiaire (*Microsoft Intermediate Language* - MSIL ou IL)
  - une bibliothèque de classes riche accessibles par le CLR
  - Support de nombreux langages informatiques dans .NET : C++, C#, J#, Deplhi …
  - Formats de type et de description standards
  - Possibilités de communiquer avec l'extérieur grâce à SOAP et XML (WebServices)
- 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.

3 améliorations

- Les nombreux atouts de la plate-forme Java
  - 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 : 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.
- Le support d'un grand nombre de langages à l'aide du IL

On le voit, Microsoft joue la carte de l'ouverture avec .NET, stratégie jusqu'ici peu familière.

Architecture



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.
- 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.
- Vulgarisation avec java et sa JVM.
- Les avantages sont considérables, mais il faut les payer au prix de performances amoindries.

P-Code

- **Indépendance à l'architecture sous-jacente**. « *Write Once, Run Anywhere* » (WORA) de Java => « *execute on many platforms* » de Microsoft
- **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, autrement dit plus accessible par le programme.
- **Sécurité du code**. Puisque le code délègue à la machine virtuelle les accès véritables à la machine physique, le programmeur ou l'utilisateur du code peuvent décider de donner des droits restreints à l'application => d'exécuter du code inconnu en toute confiance en restreignant l'accès au disque dur et au réseau du code (principe des *applets* Java).

P-Code

- 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. Le seul langage vraiment reconnu est le p-code, ce qui permet à un objet Eiffel d'hériter d'un objet C# et inversement.
- **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. 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.

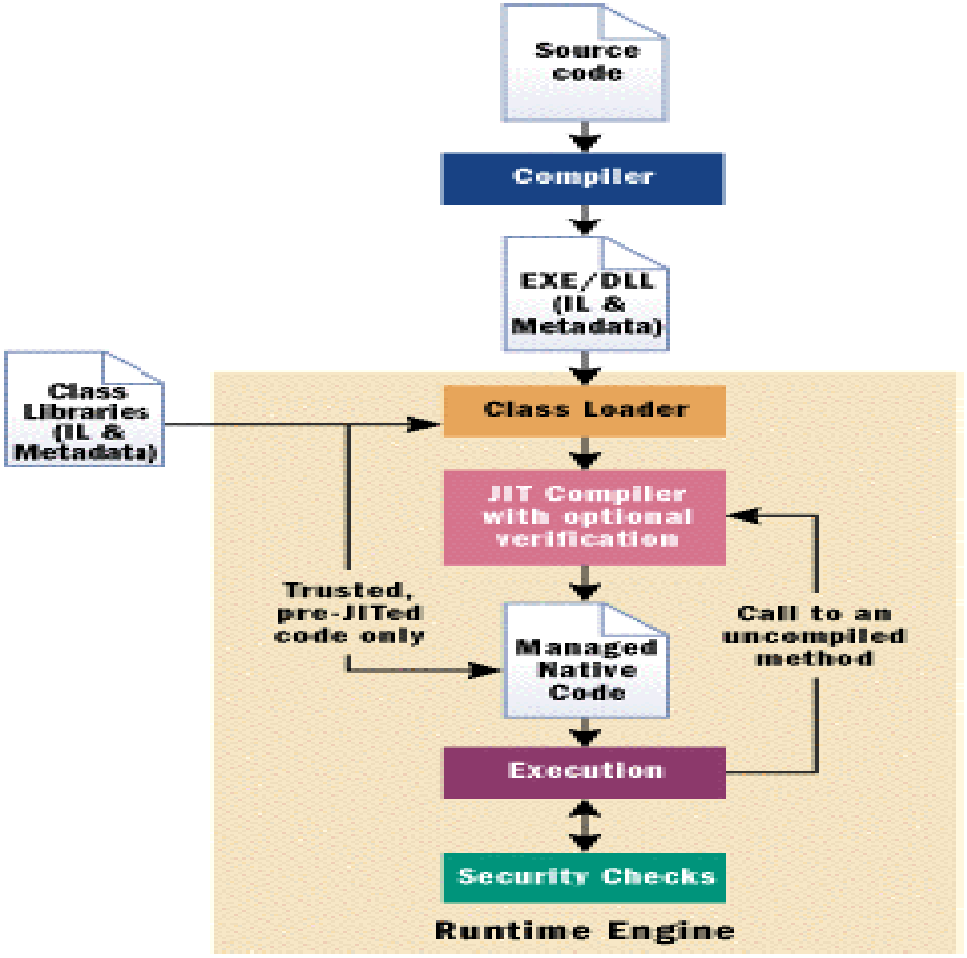
P-Code

- **Sûreté de typage**. Grand utilisateur de C++, Microsoft connaissait des difficultés liées au *casting* (transtypage) possible avec tout pointeur.
- A l'aide du ***Common Type System*** (CTS), le typage est sûr dans .NET, pour tous les langages l'utilisant rigoureusement.
- **Mécanisme unifié de passage des message d'erreur**. Désormais, à la manière de Java, uniquement les exceptions sont utilisées pour gérer les erreurs.

managed runtime environment

- 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).
- .NET n'utilise ni machine virtuelle ni interpréteur mais un environnement d'exécution contrôlé (*managed runtime environment*)
  - 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 => très bonnes perfomances.
  - 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 fournit 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.

managed runtime environment



Les assemblages

- Un objectif majeur de .NET est la fourniture simplifiée de composants => Offrir au programmeur une boite noire, pas nécessairement située ou exécutée sur la machine l'utilisant, contenant des outils réutilisables.
- Une grande innovation est la notion de **manifeste** (*Manifest*) Le manifeste d'un assemblage est un ensemble de méta-informations pour le package :
  - 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] impactent directement le manifeste sans influencer le code IL généré car ils affectent le déploiement et la mise en oeuvre de l'application plutôt que sa logique.

Common Language Specification (CLS)

- Le CLS est la solution mise en oeuvre dans .NET pour permettre **l'interopérabilité entre les langages**.
- Pb de 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
- CLS ⇔ **le plus grand dénominateur commun entre les langages orientés objet** => l'emploi de mécanismes communs dans le système de types ou dans l'approche orientée objet. 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** édictant les propriétés que doivent respecter les langages CLS. Les 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…

Classification des langages

Les langages CLS peuvent être de deux types :

- 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.

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. 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.

Les Services Web

- Les services Web permettent **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.
- Les services Web reposent sur le protocole **SOAP**, anciennement connu sous le nom de XML-RPC, pour interopérer de manière neutre.

Le protocole SOAP

- Spécification, sur laquelle Microsoft a travaillé avec IBM notamment. SOAP est maintenant validé par le W3C dans sa version 1.2.
- Décrit 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.

SOAP

- SOAP se différencie des autres protocoles par sa grande simplicité d’usage et sa neutralité.
- XML a été choisi comme format d’échange, ce qui permet d’utiliser la norme XML Schema 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.
- Les données transportées sont lisibles telles quelles par le développeur et leur manipulation par le client en est grandement facilitée.
- La norme de SOAP prévoie qu’il utilise le protocole HTTP pour transporter les données (d'autres protocoles sont disponibles, dont SMTP). SOAP sur HTTP n’est pas bloqué par les pare-feu (*firewalls*) des entreprises.
- La gestion des sessions et des transactions a été volontairement exclue de la norme.

Les composants du Framework 2.0

- ASP.NET → programmation web (sites web dynamiques, applis web, services web XML)
- ADO.NET → accès aux données (BdD)
- Windows Forms → interface graphique
- System.XML → manipulation de XML (avec support de XSLT et XPath)

Les composants du Framework 3.0

- Windows Workflow Foundation → approche processus
- Windows Communication Foundation → Service Oriented Architecture
- Windows Presentation Foundation → rendu d'interfaces graphiques
- Windows CardSpace → identification digitale

Les "nouveautés" (3.5, 4.0)

- Language INtegrated Query (unification de la façon d'interroger des données Objets/XML/SQL)
- Amélioration du support de la prog parallèle
- Quelques ajouts de fonctionnalités pour les langages .NET

Source

- Microsoft .NET : architecture et services par Jean François Bobier : http://www.developpez.biz/downloads/dotnet/MemoireDotNet.pdf
- Les nouveautés du Framework 2.0 : http://lgmorand.developpez.com/dotnet/framework2/