Conservatoire Nationale des Arts et Métiers

Centre d'enseignements de Grenoble

Année Universitaire: 2008-2009

J2EE vs NET

Cours :NFE107 Urbanisation & Architecture des Systèmes d'Information

Auditeurs : Leonardo AMODIO

INDEX

1 INTRODUCTION

Les technologies J2EE et NET font partie de la famille des langages OO (Orientés Objet) utilisés pour la création d'applications commerciales multi-niveaux.

1.1 LES LANGAGES ORIENTES OBJET

1.1.1 Historique

Le premier langage de programmation orienté objet a été Simula (1967), suivi dans les années 70 par Smalltalk et par diverses extensions du Lips. Dans les années 80 ont étais créés des extensions orientées objet pour des langages préexistants ; par exemple C++ et Objective C pour le C et Object Pascal pour le Pascal. Dans les années 90 il est devenu le paradigme dominant. Aujourd'hui les langages les plus utilisés parmi ces qui supportent que le paradigme OO soient Smalltalk et Eiffel. Des toutes façons les langages les plus utilisés en absolu sont ces qui supportent aussi le paradigme OO : C++, Java, NET, Phyton, Perl, PHP, etc.

Un langage de programmation, pour être définit « à objet » doit posséder certaines caractéristiques.

1.1.2 Caractéristiques d'un langage OO

Le but de tout langage de programmation consiste à fournir aux programmeurs des instruments d'abstraction pour exprimer, de façon le plus efficace possible, le fonctionnement d'un programme. Les langages OO (orientés objet) ont permit l'introduction de certains concepts d'abstraction plus puissants par rapport à ces qui étaient disponibles auparavant : c'est à dire les concepts de Classe et Objet.

Classe: Structure de données formées par un ensemble de variables et opérations (appelées méthodes) qui peuvent être exécutées sur ces données

Objet: Elément d'une classe qui est créé dans un programme pour être en suite utilisée à travers les méthodes que la classe définit. On dit qu'un objet est une instance d'une classe.

Les avantages de ce type d'abstraction sont plusieurs et s'expriment principalement à travers l'héritage, le capsulage et le masquage de l'implémentation.

Héritage: Mécanisme pour créer des classes à partir de celles déjà définies. La classe dérivée aura toutes les caractéristiques de la classe dont elle dérive plus éventuellement des autres ajoutés. En fin ça favoris la réutilisation du code.

Capsulage: Un objet peut baser son fonctionnement sur des variables internes (définies "privâtes") que dans aucun cas peuvent être modifiées par le programme qu'utilise l'objet sauf en utilisant les méthodes prédéfinis. Ce fonctionnement augmente la fiabilité des programmes par rapport aux erreurs de programmation.

Masquage de l'implémentation: Les objets d'une classe sont complètement et exclusivement définis par la définition de la classe. Il n'est pas nécessaire connaître la façon avec la quelle elles sont implémentées pour pouvoir les utiliser. Ca facilite la mise à jour de certaines caractéristiques d'une classe sans obliger à modifier qu'utilisent cette classe.

Aujourd'hui, la programmation par objet est vue davantage comme un paradigme, un ensemble d'instruments conceptuels, plutôt que comme une simple technique de programmation. Les langages Java et NET font partie de la famille de langages OO bien que plusieurs différences et au niveau technique et de mise en œuvre.

2 PRESENTATION DES TECHNOLOGIES J2EE ET NET

2.1 INTRODUCTION A J2EE

2.1.1 Définition

J2EE (aujourd'hui appelé JEE – Java Enterprise Edition) est une spécification pour le langage de programmation Java de Sun destinée aux applications d'entreprise. La première version des librairies J2EE a été mise a disposition des développeurs en 1999. J2EE offre une plate-forme de développement et déploiement en langage Java pour les applications distribuées à plusieurs niveaux. La version courante du langage est la 1.6 (ou Java 6).

2.1.2 Fonctionnement interne

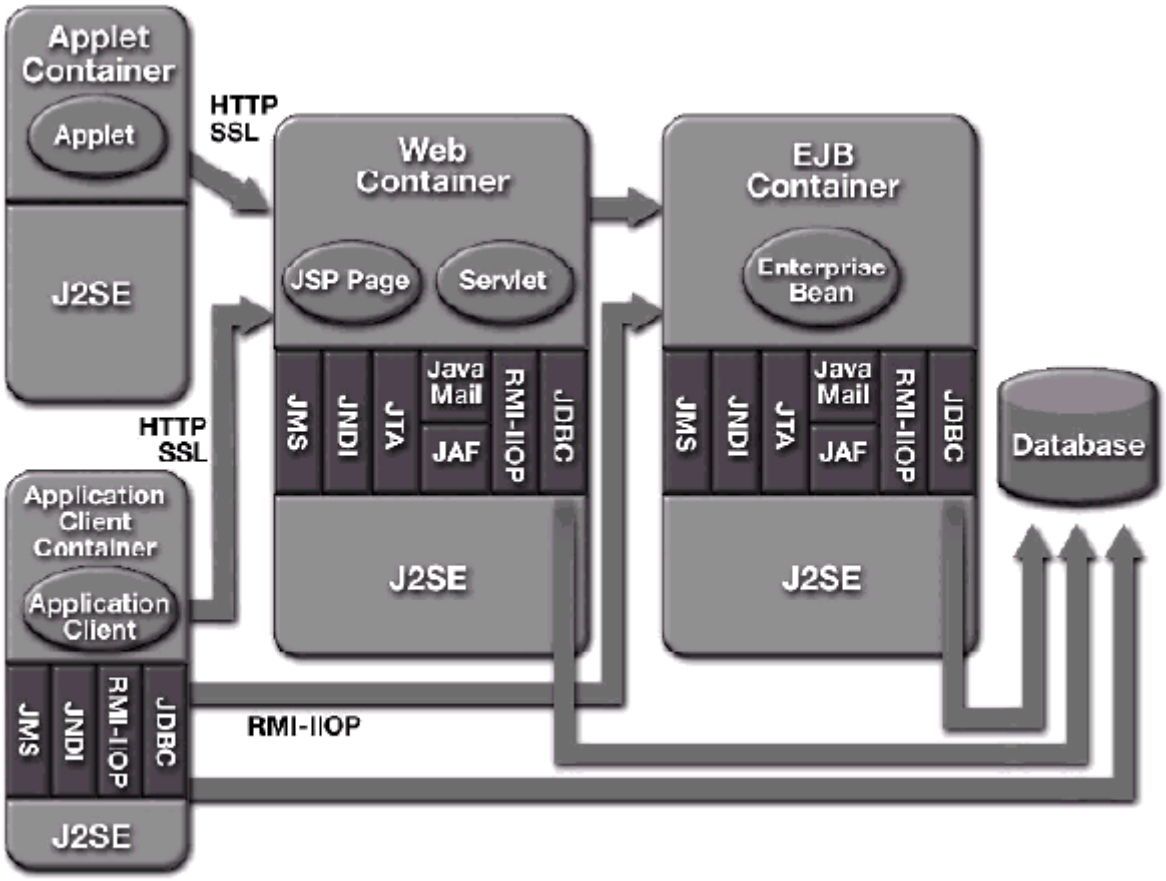
Le langage Java, sur lequel les librairies J2EE sont utilisées, met à disposition un compilateur et une machine virtuelle (JVM – Java Virtual Machine) qui se charge de créer un environnement standard pour le lancement de l'application sur tout type de système opérationnel. Le compilateur compile le code source est produit le bytecode, soit un code intermédiaire qui sera en suite lit par la machine virtuelle Java. Chaque système opérationnel majeur possède une JVM expressément codée.

2.1.3 Architecture

J2EE ajout nombreuses couches de niveau entreprise au-dessus de la plate-forme J2SE - Java Standard Edition. Chaque couche est conçue pour supporter une différente technologie de développement.

- **Technologie web application**: technologies liées à la production des interfaces web dynamiques, par exemple JSP (Java Servlet Pages) et servlet
- **Technologie enterprise application**: technologies plus directement liées à la logique de business : EJB (Enterprise Java Bean), JavaMail, JMS (Java Message Service), JTA (Java Transaction) ,etc.
- **Technologie web services**: technologies utiles au développement des applications adhérentes au paradigme SOA (Service Oriented Architecture) : web services, JAX-WS (java API for XML-based web services), JAX-RPC (Java API for XML-Based RPC)

- **Technologie management and security**: technologies liées à la gestion de la technologie enterprise afin de réaliser l'accès et l'échange d'infirmation entre machines et services distribués : JAAS (Java Authentication and Authorization Service), JCA (Java Connector Architecture)



voir glossaire

Pour expliquer l'utilisation de ces technologies on peut imaginer que les technologies enterprise sont utilisées pour gérer l'accès aux données (généralement un ou plus database), les technologies web application sont utilisées pour montrer les données aux utilisateurs génériques. Dans un contexte Business to Business, les technologies web service seront utilisées pour échanger les informations avec les partenaires commerciales et les technologies de gestion gèrent tous les processus informationnels assurant la sécurité des transactions.

2.1.4 Le serveur d'application

Les API J2EE ont était projetés pour travailler avec un particulier type de serveur appelé J2EE Application Server. Un serveur d'application est une couche software résident sur une machine serveur qui fournit les services que la technologie J2EE nécessite. Il y a plusieurs application serveurs. Parmi les produits commerciaux on rappel Bea WebLogic, IBM Websphere, Sun Application Serveur, Pramati, etc. Parmi les produits libres le plus connu est JBOSS. Les différences principales entre les différents serveurs d'applications sont relatives aux opérations de déploy, clustering, etc… Par contre toutes les fonctionnalités qui concernent strictement le fonctionnement des applications J2EE adhérent aux spécifications proposées par la Sun. Un serveur d'application s'installe et se lance comme un normale serveur web (de fait il met à disposition des services typiques d'un serveur web). En plus il possède un panneau d'administration accessible par un poste distant à travers lequel on peut définir les différents services. On peut considérer le serveur d'application comme une application Java standalone exécutée par une J2SE qui dessert les requêtes provenant des différents clients.

2.1.5 Outils de programmation

Plusieurs Environnements de développement intégrés (IDE) sont disponibles pour la plate-forme J2EE. Parmi les logiciels libres on rappel Eclipse de IBM, qui met à disposition, grâce au système des plug-in, un riche environnement de développement. Encore on retrouve NetBean de la Sun que, à différence d'Eclipse, a était créer exclusivement pour faciliter le développement d'applications J2EE.

2.2 INTRODUCTION A NET

2.2.1 Définition

NET est un produit proposé par la société Microsoft, pour le développement d'applications d'entreprises multi-niveaux, basées sur des composants Microsoft. NET constitue ainsi la réponse de Microsoft à la plate-forme J2EE de Sun. Annoncé pour l'année 2000, la première version du langage a était livrée en 2002. La dernière version livrée est la 4.5 sortie en juin 2008.

2.2.2 Fonctionnement interne

Un moteur d'exécution, appelé CLR (Common Language Runtime) permet de compiler le code source de l'application en un langage intermédiaire, baptisé MSIL (Microsoft Intermediate Language) ou CIL (Common Intermadiate Language). Lors de la première exécution de l'application, le code MSIL est à son tour compilé à la volée en code spécifique au système grâce à un compilateur JIT (Just In Time). NET est de fait un framework qui contient différents langages de programmations intégrés entre eux. NET a était conçu pour les systèmes Windows et bientôt il sera disponible pour toutes les systèmes opérationnels.

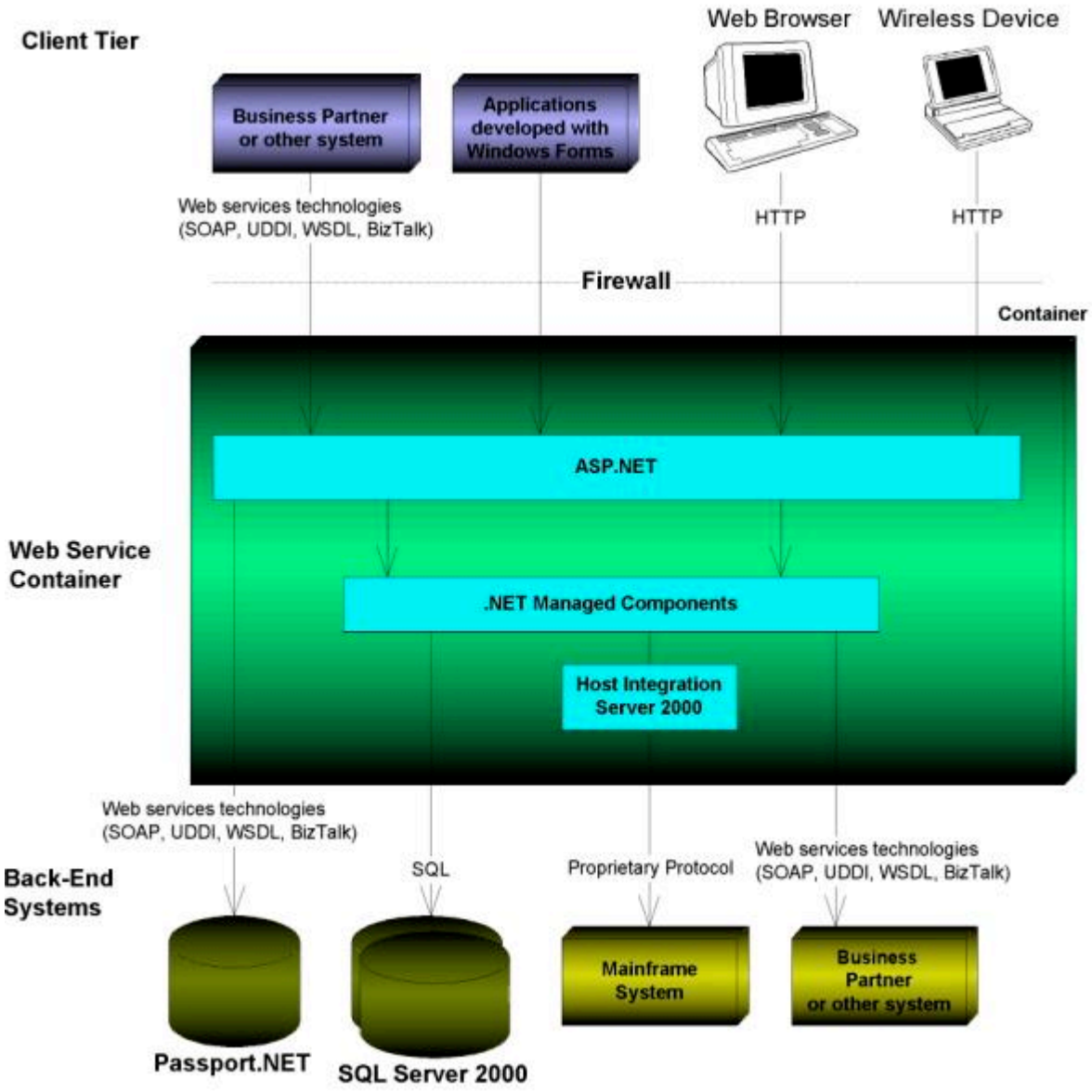
2.2.3 Architecture

La plate-forme NET a été élaborée en s'appuyant sur une communauté d'utilisateurs et a abouti à l'élaboration de spécifications. Ces spécifications ont été ratifiées par un organisme international de standardisation, l'ECMA (European Computer Manufacturers Association), ce qui en fait un standard. Ainsi l'effort de standardisation a permis l'émergence de plates-formes portées par des entreprises tierces et disponibles sous un grand nombre de systèmes d'exploitation.
On parle généralement de «Framework» pour désigner l'ensemble constitué des services (API) offerts et de l'infrastructure d'exécution. Le framework NET comprend notamment :

L'environnement d'exécution

- La machine virtuelle (CLR)
- Un environnement d'exécution d'applications et de services web, appelé ASP NET ;
- Un environnement d'exécution d'applications lourdes, appelé WinForms.
- Des services, sous forme d'un ensemble hiérarchisé de classes appelé Framework Class Library (FCL). La FCL est ainsi une librairie orientée objet, fournissant des fonctionnalités

pour les principaux besoins actuels des développeurs. Le SDK (Software Development Kit) fournit une implémentation de ces classes.



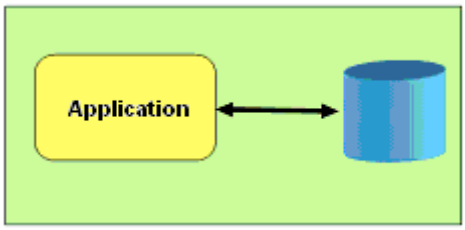
3 LES ARCHITECTURES DISTRIBUEES

Une application distribuée est une application (typiquement de grandes dimensions) pour la quelle les différents composants ont était distribués sur différents niveaux de l'environnement d'élaboration du réseau par rapport aux fonctions qu'elles offrent dans l'architecture globale. Ces applications sont complexes et sont constitués de nombreux programmes software qui sont exécutés sur différents supports hardware. Les « niveaux » (tiers) d'une application distribuée sont les niveaux entre sa propre application et les données qu'elle gère. Ces niveaux respectent les règles suivantes :

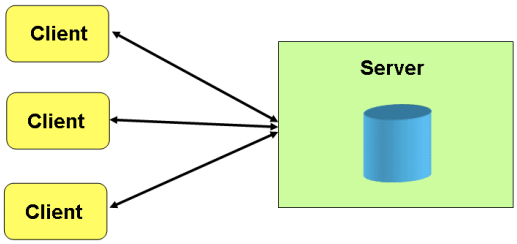
- Claque niveau ne communique qu'avec le niveau sous-jacent et dépend seulement de ce dernier
- Chaque niveau n'as pas connaissance d'aucun niveaux en dehors du niveau sous-jacent

3.1 TYPOLOGIES D'APPLICATIONS DISTRIBUEES

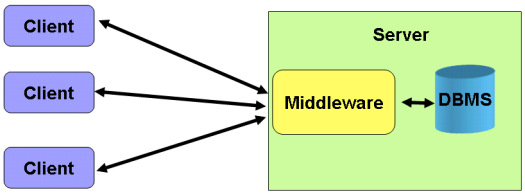
Un niveau: L'application elle-même gère ses données de façon directe. Ce type d'applications est très spécialisé et demande un accès aux données très rapide.



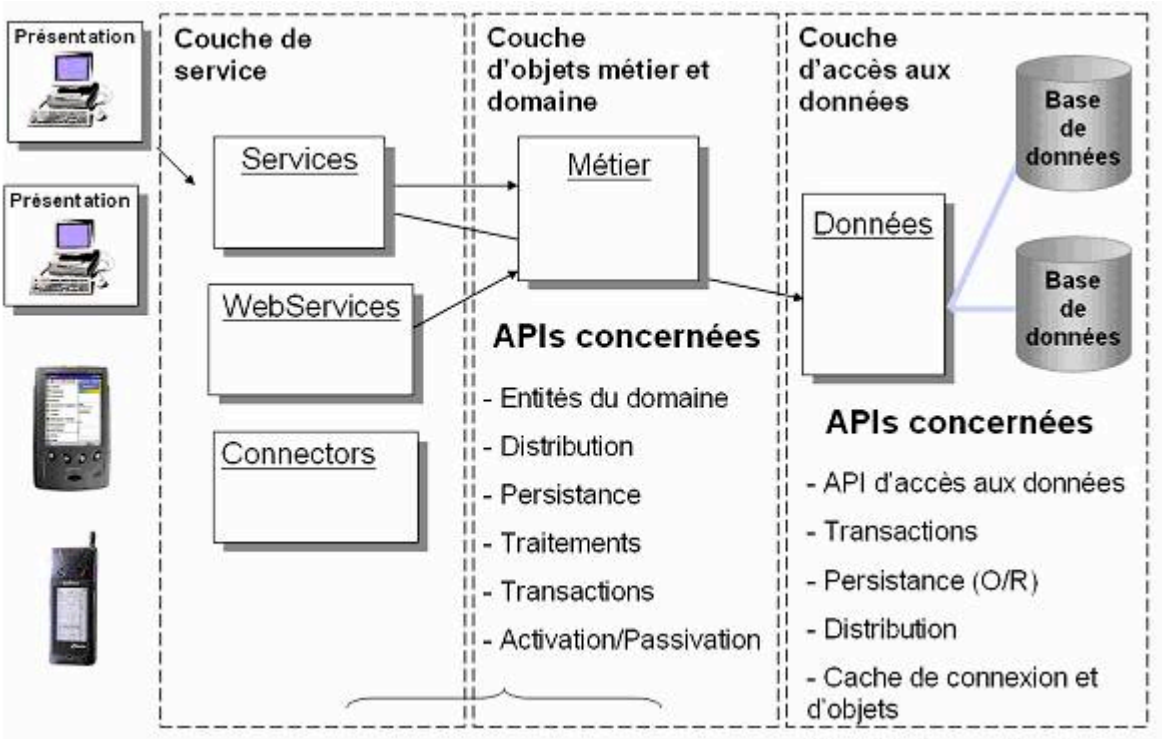
Deux niveaux: Un serveur de base de données s'occupe de gérer les données et de retourner les résultats. La communication envers la base de données peut être faite à travers des librairies particulières ou à travers un pilote chargé à run-time par l'application elle-même. Cette dernière possibilité implique que la base de données peut être changée sans modifier le programme.



Trois niveaux: Une application à trois niveaux implique que parmi elle-même et la base de données il y ait une autre application qui augmente le niveau d'isolation entre les deux extrémités. L'application envoi ses requêtes à une autre application qui se charge d'interagir avec la base de données.



Multi niveaux: En général il peut y avoir plus que trois niveaux parmi l'application et la base de données. Ca permet d'avoir une mineure dépendance entre les composants et, par exemple, on pourra changer de base de données sans trop de problèmes. Une application multi-tiers permet de distribuer la charge de travail. Ca permet d'améliorer l'utilisation du réseau et éviter la saturation du réseau même et des composants.



- **La couche de présentation** contient les différents types de clients, léger (Web, ASP, JSP) ou lourd (Swing, WinForm)
- **La couche de service** contient les traitements (contrôleurs de Use Case UML) représentant les règles métier (créer un compte, rechercher un client, calculer un amortissement, ...)
- **La couche d'objets métier** est représentée par les objets du domaine, c'est à dire l'ensemble des entités persistantes de l'application (Facture, Bon de Commande, Client, ...)
- **La couche d'accès aux données** contient les usines d'objets métier, c'est à dire les classes chargées de créer des objets métier de manière totalement transparente, indépendamment de leur mode de stockage (SGBDR, Objet, Fichiers, Legacy, ...)

4 J2EE, NET ET LES APPLICATIONS MULTI-NIVEAUX

4.1 COUCHE DE PRESENTATION

La couche de présentation est la face visible de notre application. Les clients se divisent en deux catégories :

- **Client léger** : les navigateurs HTML car ils sont pré-installés sur la plupart des plate-formes avec le système d'exploitation.
- **Client lourd** : utilisant une interface graphique à base de formulaires riches et de contrôles graphiques complexes

4.1.1 J2EE

Pour le client léger la technologie J2EE possède les APIs standard Java Servlet et Java Server Pages. Le servlet container de l'application server permet le déploiement de ces composants web sur tout type de systèmes opérationnels. Des framework comme Struts sont mis à disposition pour améliorer l'architecture de cette couche.

Pour le client lourd les librairies Java mettent à dispositions les librairies Swing (propre à J2EE) et AWT. La portabilité d'AWT et de Swing provient de la conception interne des classes existantes. Toutes les plate-formes possèdent des systèmes d'affichage vidéo différents : XWindow Motif pour Unix, Win32/GDI pour Windows, Apple/MAC. Sun a donc mis en place la notion de Composant *Peer* simplifiant le portage vers une plate-forme cible. Cette partie est donc native (code C/C++) et utilise des bibliothèques liées à chaque plate-forme.
Quant à la conception du modèle objet Swing, elle est basée sur le concept de **conteneur/composant** et utilise intensivement le design pattern MVC (Modèle Vue Controlleur).

4.1.2 NET

Par rapport aux clients légers les **ASP.NET** représentent l'équivalent des JSP avec un modèle de développement totalement intégré basé sur les WebForms. Les **WebForms** permettent de développer une interface graphique Web de la même manière qu'une interface graphique VB, cela dans le but de séparer les traitements de la présentation. Ainsi, le formulaire représente la page Web et les traitements sont contenus dans une seconde page appelée **Code Behind**. Cette page peut être codée dans n'importe quel langage du Framework.NET et contient plusieurs méthodes, dont l'implémentation des évènements liés à cette page. La page HTML finale qui sera générée au client intègre la présentation et le Code Behind en ciblant différents navigateurs. D'ailleurs, vous n'avez pas ou très peu la maîtrise du code généré qui est laissé à la charge du Framework. Pour résumer, Microsoft adopte une stratégie d'intégration des outils et permet de simplifier le modèle de développement en proposant les WebForms au dessus des ASP.NET avec Visual Studio comme chef d'orchestre. Bien entendu, il est possible d'utiliser uniquement les ASP.NET sans les WebForms tout comme en Java il est possible d'utiliser les JSP sans Struts.

Pour les clients lourds Microsoft propose une API similaire à AWT/Swing : les WinForms (Windows Forms). Les WinForms sont une surcouche de GDI+ (Graphical Device Interface) dont la responsabilité est de créer nativement (code non managé) des contrôles graphiques du type fenêtres, boutons, ... D'ailleurs GDI+ est écrit en C++ et améliore considérablement le modèle (non) objet de GDI/Win32 pour cibler l'architecture .NET et le namespace System.Drawing qui l'intègre. Dans le monde Java, Java 2D est l'API se rapprochant le mieux de GDI+ à ceci près que la majeure partie du code de Java 2D est écrit en Java. A l'heure actuelle, l'architecture des WinForms cible essentiellement Windows et le portage des WinForms vers d'autres systèmes s'avère délicat. Les WinForms sont le pendant des Swing dans le monde Java et constituent la partie la plus délicate à porter du fait de leur forte intégration avec Windows. Quant à la conception du modèle objet WinForms, tout comme Swing, elle est basée sur le concept de **conteneur/composant** avec une utilisation moindre du design pattern MVC.

4.2 COUCHE DE SERVICE

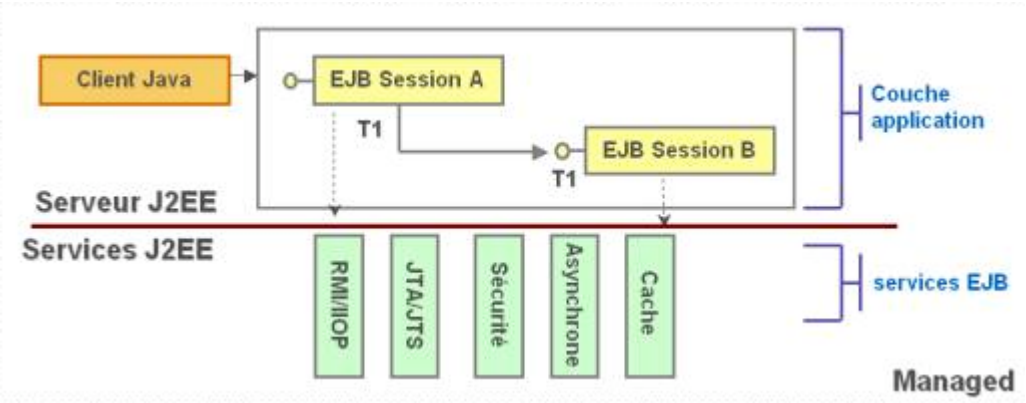
La couche de service est la partie de notre architecture contenant les traitements applicatifs. Vous avez plusieurs alternatives au développement de services dans cette couche. Soit vous prenez l'entière responsabilité de l'implémentation des composants sans l'utilisation des services du Framework (vos besoins sont simples), soit vous faites appel au Framework pour :

- La gestion des transactions automatiques
- Le cache d'objets (Pooling)
- La montée en charge et le multi-threading
- Les composants asynchrones, ...

La responsabilité du conteneur est de vous fournir tous ces services en vous proposant un canevas dans lequel vous implémenterez vos composants.

4.2.1 J2EE

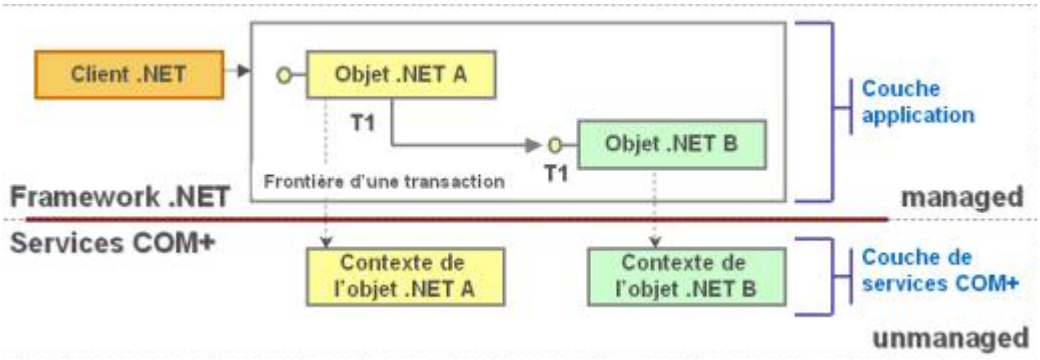
Le choix de l'implémentation de cette couche réside dans deux approches : utiliser les EJB ou ne pas les utiliser. Les composants concernés par cette couche sont les EJB Session. Sun spécifie un certain nombre d'éléments dans J2EE allant de la gestion des transactions en passant par la sécurité à la synchronisation des tâches. Encore une fois, Java se distingue par l'ouverture de ces APIs avec comme maître mot : la portabilité du serveur.



4.2.2 NET

Tout comme J2EE, .NET propose le même ensemble de services sous l'appellation de ServicedComponent. Le conteneur utilisé dans le Framework est COM+. D'ailleurs, cela n'est pas sans poser de problème, COM+ fonctionne dans un environnement non managé avec une gestion de type différente de celle de .NET (Common Type System). Tout appel à une méthode d'un

ServicedComponent utilisant des services spécifiques (Transactions, sécurité, ...) se traduit par une redirection vers les couches basses COM+ entraînant la création d'un Proxy (Wrapper). Ce dernier joue le rôle d'interface entre le nouveau monde et le vieux monde. Cet aspect qui pourrait sembler être un inconvénient n'en est pas vraiment un. En effet, le code d'un client utilisant un ServicedComponent n'est en aucun cas lié aux interfaces COM+ tout comme le composant lui même qui utilise l'API .NET : ServicedComponent. Tout cela se passe dans les tuyauteries internes du Framework comme décrit dans le schéma suivant. A plus long terme, lorsque Microsoft aura entièrement porté COM+ en COM+.NET 2020 seules ces couches seront impactées, pas le composant et encore moins le client.



La distribution est assurée par **Remoting** pour .NET et **RMI sur IIOP** pour Java. Les transactions sont gérées en Java à l'aide des API **JTA/JTS** et l'ensemble des composants s'exécutent dans un seul et même environnement managé, ce qui n'est pas le cas de .NET comme nous l'avons vu précédemment avec COM+.

4.3 COUCHE D'OBJETS METIER

Les objets métiers sont représentés par l'ensemble des objets persistants du domaine de l'application. Une facture, un bon de commande ou tout autre objet nécessitant d'être stocké en base. Cette couche assure l'indépendance totale entre le client et le type de stockage utilisé (SGBDR, SGBDO, fichiers XML, ...). Le client doit posséder uniquement une vue sur un objet avec l'ensemble de ces attributs et non un éventuel curseur (ResultSet et RecordSet) pointant vers une ligne d'une quelconque base. Cette couche est en étroite collaboration avec la couche de persistance qui assure la création des objets métier de manière totalement transparente.

4.3.1 J2EE

Ce type d'objet est représenté par les **EJB Entity** dans la spécification J2EE. Dans ce cas, vos entités métiers sont couplées aux interfaces du Framework EJB et cela peut être gênant si la spécification EJB évolue dans le futur, d'autant plus que ces objets sont destinés à vivre très longtemps contrairement aux objets de services qui dépendent de l'évolution des règles métiers. Une autre approche consiste à développer des objets métiers simples n'héritant d'aucune interface ou classe du Framework en utilisant des outils de persistance fonctionnant par **introspection**. C'est à dire que le mapping entre vos objets et vos tables est réalisé via des fichiers de configuration externes à votre application. C'est le cas de la spécification JDO qui cible ce genre de besoin avec des produits comme Castor, JUDO, KODO etc.... Ainsi, vous ne couplez pas vos objets du domaine à votre Framework de persistance.

4.3.2 NET

Les objets métier persistants dans .NET sont à l'heure actuelle des objets C# ou VB simples sans aucune caractéristique particulière.

4.4 COUCHE D'ACCES AUX DONNEES

Cette couche est responsable de la création, destruction et chargement des objets métier de manière totalement transparente. La différence entre une application 2-tiers et 3-tiers se fait à ce niveau. Son rôle est de masquer entièrement l'opération de création et de manipulation de tables (SGBDR) ou autre types de stockages spécifiques (Fichiers à plats, ...). En règle général, vous avez deux alternatives : Implémenter cette couche vous même par l'intermédiaire de classes jouant le rôle d'usines à objets métier (pattern Factory), ou laisser le soin à un Framework tiers de réaliser cette tâche (produits du marché). Tout dépend de la nature de vos besoins et de la complexité des données manipulées pour mettre en oeuvre le mapping objet/relationnel.
Cette partie, souvent délaissée dans la mise en place d'un projet est le tendon d'Achille d'une architecture. La conception de la couche de persistance doit se faire avec soin et en tenant compte de multiples facteurs tels que la montée en charge, la complexité de mise en oeuvre mais aussi l'impact d'évolutions futures de la base. Voyons ce que proposent J2EE et .NET dans ce domaine.

4.4.1 J2EE

Dans le monde Java, l'API JDBC (Java Database Connectivity) est en charge de la communication entre un client et un SGBDR. Contrairement à .NET qui propose des API oledb ciblant aussi bien

des bases relationnelles qu'un annuaire distribué (Exchange), JDBC s'adresse uniquement aux bases de données SQL.

Si vous prenez le choix de gérer vous-même la persistance des données, il vous suffira d'implémenter des classes utilisant JDBC. Dans le cas où vos besoins sont plus complexes, il faudra faire l'acquisition d'un produit (gratuit ou payant) réalisant cette tâche à votre place. Il en existe un certain nombre s'interfaçant avec les spécifications EJB (EJB Entity). Le but étant de fournir un composant standard qui utilise les services des outils de mapping objet/relationnel de manière totalement transparente.

Il existe deux types d'EJB : BMP (Bean Managed Persistance) et CMP (Container Managed Persistance). Dans les deux cas, l'API utilisée pour communiquer avec les bases de données relationnelles est JDBC. Cette API est spécifiée par Sun, intégrée à J2EE et dispose de plusieurs implémentations à travers de multiples drivers JDBC disponibles sur le marché (Oracle, SQL Server, DB2, ...). JDBC permet aussi de convertir le résultat de requêtes SQL en XML (JDBC 3.0).

4.4.2 NET

Microsoft propose ADO.NET pour l'accès aux données. ADO.NET fonctionne de manière similaire à JDBC avec quelques variantes. Historiquement, ADO souffre d'un passé quelque peu lourd à porter. C'est pourquoi, vous trouverez deux types de provider :

Managed (utilisant les services de la CLR)

Unmanaged (pour supporter les anciennes versions)

5 CONCLUSIONS

Le Common Language Runtime (CLR), le Common Intermediate Language (CIL) et le .NET sont similaires à la Java Virtual Machine, au bytecode et au langage Java. Les deux utilisent un bytecode intermédiaire mais ce de NET est projeté pour être compilé au moment de l'exécution.

NET est pour l'instant compatible avec les plates-formes Windows. Java est disponible pour tout type de plate-formes. Le projet Mono essai de ramener NET sur des autres systèmes opérationnels.

NET offre des avantages de prestations des applications en exécution et des coûts et temps mineurs de développement par rapport à J2EE

Les deux environnements disposent de mécanismes très similaires pour gérer chaque couche de l'application.

En plus de la simplicité, l´approche pragmatique de Microsoft a deux autres avantages : "les solutions étant plus packagées, elles nécessitent moins de développements, moins de ressources humaines, etc.", explique Jean-Christophe Cimetière, directeur technique adjoint du Groupe SQLI. "En d´autres termes, la phase d´acquisition des compétences est plus courte, la mise en oeuvre est plus rapide et tout cela coûte forcément moins cher que l´assemblage de briques de J2EE. Le retour sur investissement est donc plus rapide sur le court terme. Avec le temps, les choses s´équilibrent et c´est même plus délicat sur le long terme, car il n´est pas impossible que la phase initiale de J2EE, certes plus fastidieuse, s´avère plus avantageuse sur le long terme".

En outre, chaque environnement de développement présente ses particularités. Un développeur Visual Basic passera sans difficulté à .Net. Mais plus difficilement à Java, l´inverse étant également vrai. "Traditionnellement, les PME ont une culture Microsoft. Elles sont moins dépaysées par les outils de développement de .Net", estime Tanguy Crusson, responsable offre Web Services de la SSII Devoteam. "De plus, chez Microsoft, tout est packagé : à un problème correspond une solution alors que sur J2EE, il existe souvent différentes solutions, parfois sous forme de briques qu´il faut assembler. L´approche .Net est donc plus simple et adaptée à la PME. Mais attention : la plate-forme manque de maturité, Microsoft ne couvre pas tous les besoins d´une PME et faire le choix .Net, c´est accepter de dépendre de Microsoft là où avec J2EE on a le choix entre différents éditeurs et mêmes des solutions gratuites ou quasiment gratuites en Open Source".L´évolution de .Net reste en effet très liée à Microsoft malgré la standardisation de la plate-forme qui permet à n´importe quel

éditeur de "porter" .Net sur Linux, Solaris ou n´importe quel autre système d´exploitation. Des projets sont en cours, mais J2EE dispose d´une longueur d´avance incontestable avec de nombreux serveurs d´applications qui implémentent ses standards et des solutions en Open Source pour sa plate-forme. De plus, la possibilité de développer dans n´importe quel langage pour .Net, contrairement à J2EE qui impose Java, est selon Tanguy Crusson un argument avant tout marketing : ".Net accepte certes plus de 25 langages de développement là où J2EE n´accepte que Java. Mais je vois mal un développeur Cobol développer pour .Net. Ce n´est pas impossible, mais la culture, les architectures, les applications... tout est différent avec .Net ou J2EE. Lui permettre de conserver sa syntaxe pour développer pour un environnement dont il ne comprend pas les concepts, c´est certes lui faciliter la tâche et faire en sorte qu´il ne soit pas mis au placard par son entreprise sous prétexte que son langage ne sert plus à rien. Mais jusqu´à quel point est-ce réaliste ?".

6 BIBLIOGRAPHIE

http://fr.wikipedia.org/wiki/Orient%C3%A9_objet

http://java.html.it/guide/leggi/136/guida-j2ee/

http://it.wikipedia.org/wiki/Programmazione_orientata_agli_oggetti

http://it.wikipedia.org/wiki/Microsoft_.NET

http://fr.wikipedia.org/wiki/J2EE

http://ditwww.epfl.ch/SIC/SA/publications/FI02/fi-6-2/6-2-page8.html

http://www.journaldunet.com/developpeur/tutoriel/jav/050725-java-fonctionnement-jvm.shtml

http://www.juddsolutions.com/images/J2EEArchitecture.jpg

http://publib.boulder.ibm.com/infocenter/wbihelp/v6rxmx/topic/com.ibm.wics_developer.doc/doc/access_dev_j2ee/resou000.gif

http://www.dotnetguru.org/articles/architecturedotnet.htm

http://www.dotnetguru.org/articles/architecturedotnet.htm

http://www.indexel.net/bin/doc/2165

7 GLOSSAIRE

Applet container : A conteneur qui inclut un support pour la programmation des applets

Application Client container : A conteneur qui supporte les applications client

Web container : conteneur pour JSP et servlet

EJB container : conteneur pour les EJBs

Applet : client lourd Java

JSP : Page dynamique en Java

Servlet : objet Java qui génère une réponse par rapport à la requête

EJB : Enterprise Java Bean (composant côté serveur pour les applications distribuées)

J2SE : Java 2 Standard Edition (librairies standard Java)

JMS : Java Message Service API pour l'envoi de messages entre clients

JAAS : Java Authentication and Autorisation Service : framework de securité de codage

JAXP : Java API for XML Parsing: validation et parsing des documents XML

JDBC:Java Database Connectivity: méthodes pour les requêtes

JTA: Java Transaction API : permet aux applications web de gérer les anomalies à travers les transactions

JavaMail : interface pour envoyer et recevoir mails

JAF : JavaBeans Activation Framework: permet aux developpeurs de determiner le type d'u donnée, decouvrit les fonctionnalitées, etc.

IIOP (Internet Inter-ORB Protocol) protocôle qui permet la communication entre programmes écrit en differents langages