Introduction à la plateforme J2EE

Auteur : Oussama Essefi

Directeur technique

Expert Consulting

Oussama.essefi@expert-consulting.biz

1. Introduction

1.1. Pourquoi utiliser une plateforme ?

Une plateforme est une base générique qui fournit un ensemble de fonctionnalités utiles pour une majorité d'applications. Une plateforme se construit sur la base d'un ensemble de besoins génériques partagés entre plusieurs applications.

Il peut exister plusieurs types de plateformes. De la plus générique à la plus spécifique (optimisée pour un type de métier précis par exemple). Bon nombre de grandes entreprises ont déjà développé des plateformes tels que : IBM (WebSphere), SAP …

L'avantage principal de partir d'une plateforme est que l'équipe de développement n'a pas à s'acquitter de développer certaines tâches (connexion à la base de données par exemple, gestion d'objets …). Ce sont des tâches que l'on retrouve très souvent dans un grand nombre de projet et qui n'ont pas d'intérêt à être re-coder à chaque fois (perte de temps et d'argent). De plus, mieux vaut travailler sur une plateforme qui présente une forte stabilité (ça évite des débuggages inutiles !).

Un autre avantage est la facilité de prise en main des API de cette plateforme. En effet, celle-ci cache très souvent la complexité d'accès à telle ou telle ressource et permet donc un gain de temps énorme pour le développeur qui a donc plus de temps pour se préoccuper du fonctionnement réel de son application (pas de tâche ardue ou générique à développer).

1.2. Qu'est ce que J2EE ?

J2EE (Java 2 Enterprise Edition) est une norme proposée par la société Sun, portée par un consortium de sociétés internationales, visant à définir un standard de développement d'applications d'entreprises multi-niveaux, basées sur des composants.

On parle généralement de «plate-forme J2EE» pour désigner l'ensemble constitué des services (API) offerts et de l'infrastructure d'exécution. J2EE comprend notamment :

- Les spécifications du serveur d'application, c'est-à-dire de l'environnement d'exécution : J2EE définit finement les rôles et les interfaces pour les applications ainsi que l'environnement dans lequel elles seront exécutées. Ces recommandations permettent ainsi à des entreprises tierces de développer des serveurs d'application conformes aux spécifications ainsi définies, sans avoir à redévelopper les principaux services.

- Des services, au travers d'API, c'est-à-dire des extensions Java indépendantes permettant d'offrir en standard un certain nombre de fonctionnalités. Sun fournit une implémentation minimale de ces API appelée J2EE SDK (J2EE Software Development Kit).

Dans la mesure où J2EE s'appuie entièrement sur le Java, il bénéficie des avantages et inconvénients de ce langage, en particulier une bonne portabilité et une maintenabilité du code.

De plus, l'architecture J2EE repose sur des composants distincts, interchangeables et distribués, ce qui signifie notamment :

- Qu'il est simple d'étendre l'architecture.

- Qu'un système reposant sur J2EE peut posséder des mécanismes de haute disponibilité, afin de garantir une bonne qualité de service.

- Que la maintenabilité des applications est facilitée.

1.3 Les acteurs d'une application J2EE

La réalisation d'une application basée sur l'architecture J2EE fait appel à différents types de compétences que l'on trouve rarement chez une même personne car cela va de la conception jusqu'à la supervision de l'application en passant par le développement et le déploiement. Afin de pouvoir maîtriser ce processus J2EE adopte l'approche des partages des responsabilités.

Plus spécifiquement pour les EJB, cette approche définit plusieurs niveaux de responsabilité :

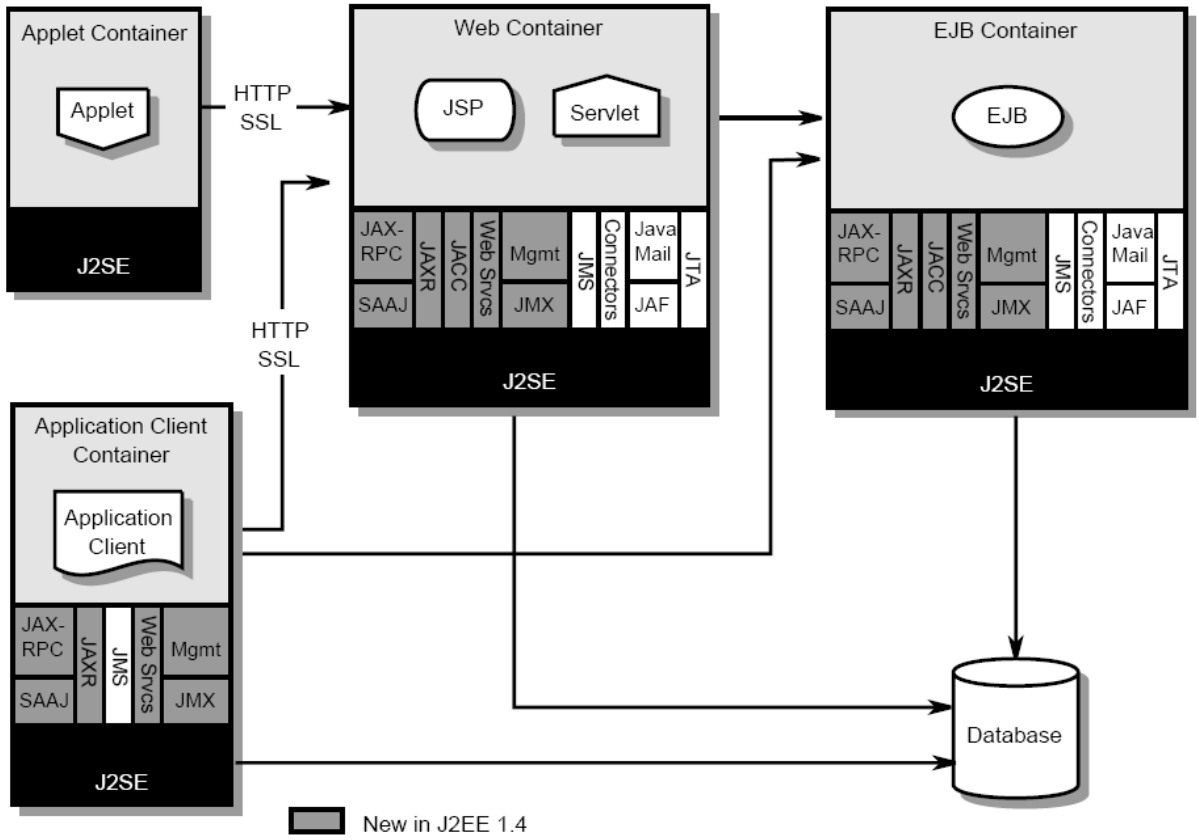
- Le fournisseur des EJB : c'est l'acteur qui fournit des composants métiers réutilisables soit par l'achat à un fournisseur de composants, soit par développement interne ;
- L'assembleur d'applications : l'acteur qui est capable de construire une application à partir d'EJB existants ;
- Le déployeur : l'acteur qui récupère l'application et s'occupe de son déploiement dans un serveur d'applications ;
- L'administrateur : l'acteur qui contrôle le fonctionnement du serveur d'application et assure la supervision des applications ;
- Le fournisseur de conteneur : l'éditeur qui commercialise un conteneur web ou un conteneur EJB ; cet éditeur commercialise souvent un serveur d'application incluant ces conteneurs ;
- Le fournisseur de serveur : c'est l'éditeur qui commercialise un serveur d'application (BEA, IBM etc...)

2. J2EE en détail

2.1. Architecture du JDK J2EE

J2EE décrit l'architecture d'un standard pour les serveurs d'application. Nous allons étudier plus précisément les composants, les dépendances et les protocoles utilisés.

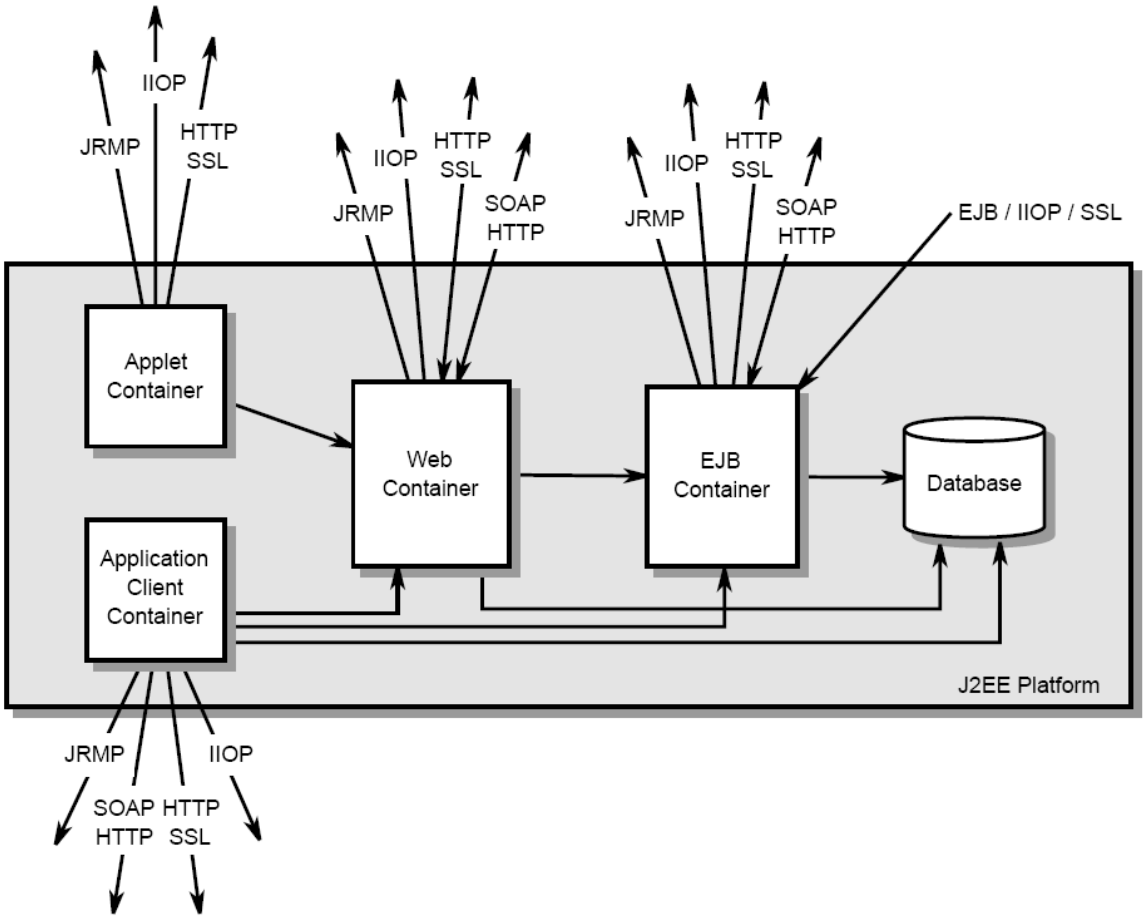
Schéma des relations entre composants et « tiers » dans l'architecture J2EE :



Les différents rectangles définissent les conteneurs (de l'environnement d'exécution J2EE) qui fournissent les services pour les différents composants (représenter par les rectangles dans les rectangles). L'ensemble des composants sera expliqué dans le chapitre suivant.

Les flèches représentent les types d'accès que le type d'application peut avoir avec les autres applications distantes. Par exemple, l'application client peut se connecter au « Web Container » par l'intermédiaire des JSP / Servlet, elle peut également se connecter à « EJB Container » …

Voici à présent le schéma de l'interopérabilité entre la plateforme J2EE et les autres programmes (les différents protocoles utilisés).



Ce schéma est très important dans le domaine de l'interopérabilité entre différentes applications. Il fournit les informations concernant les protocoles utilisés pour chacune des connexions distantes possibles. Par exemple, « Web Container » fournit des accès via HTTP / SSL ou SOAP ; « EJB Container » fournit des accès HTTP / IIOP (RMI : Internet Inter-Orb Protocol) / SSL …

Nous pouvons alors penser qu'un client en C#, par exemple, peut se connecter sur un EJB en mode HTTP (dans l'idéal) ou via le protocole IIOP (plus répandu).

2.2. Les différents outils de « bas niveau »

Nous venons de voir (très succinctement) l'architecture globale de la plateforme J2EE. Nous allons maintenant présenter les différents outils.

Il existe 3 grands types d'outils :

- Composants
- Services d'infrastructures
- Services de communications

2.2.1 Composants

On distingue, en général, 2 catégories de composants :

- Web
- Métiers

2.2.1.1 Web

Il s'agit de la partie présentation (interface de l'utilisateur et les traitements).

a. JSP

Les JSP (Java Server Page) sont les pages servant à générer l'ensemble du code HTML de l'interface utilisateur. On y intègre aussi bien du code HTML que des scriplet Java (code java) ou encore des balises personnalisées (tag-lib).

Cette technologie est donc dédiée à la génération de HTML et non au traitement de la requête de l'utilisateur. On l'appelle généralement : Vue.

b. Servlet

Une Servlet est une classe Java qui permet de traiter une requête venant d'un client. Cette technologie doit s'occuper de traiter les données envoyées par l'utilisateur et choisir la Vue à retourner à celui-ci.

On appelle cette partie : Contrôleur. En général, la classe Java ne doit quasiment pas générer de code HTML (sauf dans certains cas précis).

2.2.1.2 Métier - EJB (Entreprise JavaBean)

Il s'agit de composants spécifiques chargés des traitements des données propres à un secteur d'activité (on parle de logique métier ou de logique applicative) et de l'interfaçage avec les bases de données.

On parle de la partie : Modèle.

2.2.2 Services d'infrastructures

2.2.2.1 JDBC - Java Database Connectivity

C'est une API d'accès aux bases de données.

2.2.2.2 JNDI

C'une API d'accès aux services de nommage et aux annuaires d'entreprises tels que DNS, NIS, LDAP, etc.

2.2.2.3 JTA / JTS - Java Transaction Api / Java Transaction Services

C'est un API définissant des interfaces standard avec un gestionnaire de transactions.

2.2.2.4 JCA (J2EE Connector Architecture)

C'est une API de connexion au système d'information de l'entreprise, notamment aux systèmes dits «Legacy» tels que les ERP.

2.2.2.5 JMX (Java Management eXtension)

Cette API fournit des extensions permettant de développer des applications web de supervision d'applications.

2.2.3 Services de communications

2.2.3.1 JAAS (Java Authentification and Authorization Service)

C'est une API de gestion de l'authentification et des droits d'accès.

2.2.3.2 RMI (Remote Method Invocation)

C'est une API permettant la communication synchrone entre objets.

2.2.3.3 Web services

Les Web services permettent de « partager » un ensemble de méthodes qui pourront être appelées à distance. Cette technologie utilise XML, ce qui permet d'être utilisée par n'importe quel langage et n'importe quelle plateforme.

2.2.3.4 JMS (Java Message Service)

Cette API fournit des fonctionnalités de communication asynchrone (appelées MOM pour Middleware Object Message) entre applications.

2.2.3.5 JavaMail

C'est une API permettant l'envoi de courrier électronique.

2.3 Implémentation de J2EE : les serveurs d'application

Il est avant tout indispensable de définir clairement ce qu'est un serveur d'application. En effet, une confusion règne dans les esprits quant à la notion de serveur d'application. Cette confusion a été introduite en grande partie par les éditeurs de serveurs d'application J2EE (Java2 Entreprise Edition) afin de s'approprier ce marché. La notion de serveur d'application a en effet été mélangée avec celle de serveur d'objet qui n'a absolument rien à voir.

2.3.1 Qu'est ce qu'un serveur d'application ?

Le serveur d'application est l'environnement d'exécution des applications côté serveur. Il prend en charge l'ensemble des fonctionnalités qui permettent à N clients d'utiliser une même application :

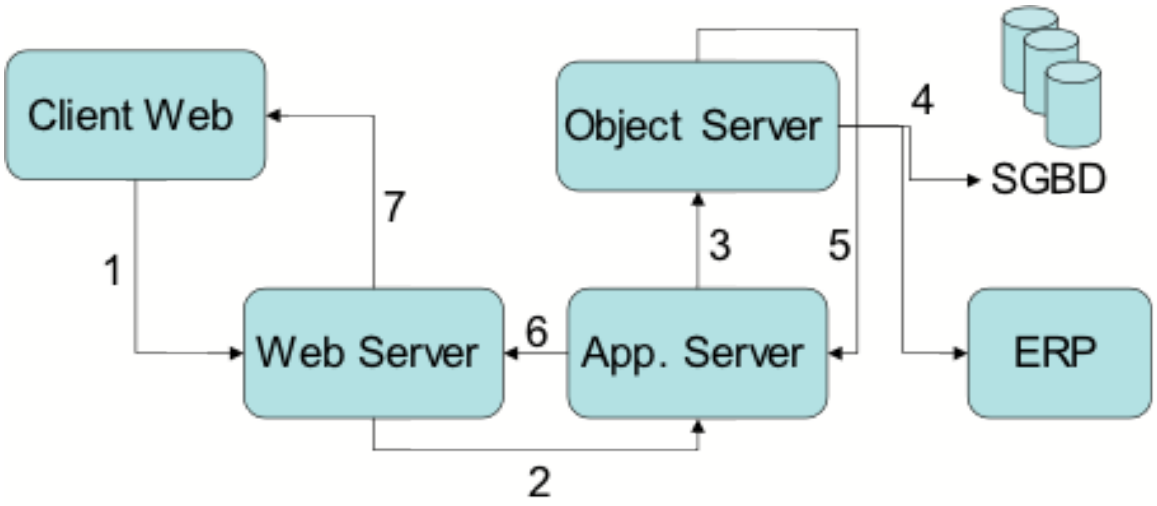
- Gestion de la session utilisateur : N clients utilisant une même instance d'application sur le serveur, il est nécessaire que le serveur d'application puisse conserver des contextes propres à chaque utilisateur (par exemple, un panier de commandes). La plupart des serveurs d'application génèrent un identifiant unique pour chaque nouveau client et transmettent cet identifiant lors de chaque échange HTTP par URL longs, variables cachées ou cookies.
- Gestion des montées en charge et reprise sur incident : Afin de gérer toujours plus d'utilisateurs, le serveur d'application doit pouvoir se déployer sur plusieurs machines et éventuellement offrir des possibilités de reprise sur incident (même si dans la grande majorité des cas, on se contente d'une gestion des montées en charge au niveau réseau - boîtier de répartition, DNS round-robin, reverse proxy ...).
- Ouverture sur de multiples sources de données : C'est le serveur d'application qui rend accessible les données des applications du système d'information. Il doit donc pouvoir accéder à de nombreuses sources de données. On s'attend également à ce qu'il fournisse des mécanismes performants comme le pooling de connexion base de données.
- ...

Le serveur d'application est donc indispensable si l'on souhaite éviter de re-développer l'ensemble de ces fonctionnalités (cas des GGI). Les moteurs JSP/Servlets, Microsoft ASP, Cold Fusion, PHP ... sont à ce titre des serveurs d'application (même si il sont intégrés au ServeurWeb PHP/ASP).

2.3.2 Qu'est ce qu'un serveur d'objet ?

Pour aborder la notion de serveur d'objets, il faut comprendre qu'il existe deux méthodes pour accéder aux données et aux traitements.

- La première consiste à accéder directement aux sources de données. Cette méthode de programmation n'empêche en aucun cas de structurer ses développements.
- La deuxième méthode consiste à s'appuyer sur des objets métier (client, fournisseur ...) afin de masquer la complexité d'accès aux données. Un objet AssuréSocial possédera par exemple une méthode débit() et une méthode crédit () qui à chaque appel iront modifier les données dans une ERP (Entreprise Resource Planning), un système de CRM (Customer Relation Ship Managment) ou une base de données.



1. Requête du client
2. Le serveur web passe la requête au serveur d'application
3. Le serveur d'application traite la requête par des appels au serveur d'objets
4. Le serveur d'objet traite les données avec les bases de données (en tout genre)
5. Le serveur d'objet retourne les objets au serveur d'application
6. Le serveur d'application renvoie le résultat au serveur web
7. Le serveur web fait suivre le résultat au client

Pour gérer ces objets, un environnement d'exploitation est nécessaire : le serveur d'objets. Ce serveur d'objets va devoir fournir des services tout à fait différents de ceux des serveurs d'application :

- Gestion de la persistance des objets,
- Gestion des transactions objets métier

- Gestion des montées en charge : ici les mécanismes n'ont rien à voir avec ceux mis en oeuvre pour un serveur d'application. Il faut pouvoir assurer la transparence de localisation, l'instanciation, ... des objets métier ...

Bref, on le voit, on a à faire à des techniques très différentes. Les principaux serveurs d'objets à ce jour sont les serveurs EJB (Enterprise Java Beans), Corba. Ils ne sont nécessaires à ces développements que si l'on souhaite utiliser pleinement la logique d'objets métier.

Il est donc important de ne pas mélanger ces notions afin d'éviter de se faire « prendre » comme 80% des acheteurs de serveurs J2EE (incluant serveur d'application et serveur d'objets) qui n'utilisent que le moteur de JSP/Servlets dont les coûts sont beaucoup plus limités que l'ensemble J2EE (incluant le serveur d'objets EJB).

Sur le terrain, on rencontre beaucoup plus de développements sur des serveurs d'application seuls que d'applications utilisant des serveurs d'objets. En fait, le marché des serveurs d'application s'est fortement structuré depuis une ou deux années. De plusieurs dizaines de technologies il y a peu, seules trois technologies émergent aujourd'hui : l'offre Java, l'offre Microsoft et l'offre PHP. Hormis cas particulier, nous recommandons de ne pas sortir de ces trois choix.

Les points clés d'une architecture sont les capacités transactionnelles du serveur d'application à délivrer des pages et à intégrer une montée en charge… L'ergonomie au sens large est un autre point clé. Les choix de design doivent prendre en compte les contraintes du Web (taille des images, ...).

Le marché offre trois familles de solutions de développement pour les serveurs d'application.

- Les solutions de scripting peuvent être simples et productives mais plutôt orientées vers les sites jetables, de type évènementiel. Un site en ASP, PHP 3 ou ColdFusion peut être développé très rapidement ; par contre, sa maintenance est compliquée voire quasi-impossible.
- Les solutions orientées objets techniques permettent de factoriser le code sans rentrer dans la complexité des objets métier. Il est important d'imposer des règles de développement précises à ses équipes et prestataires. Les développements JSP/Servlets/JavaBeans, PHP4/5, ASP/DCOM (et ASP.Net/DCOM) permettent de tels développements.
- Les solutions orientées métier sont plus complexes et plus coûteuses à mettre en oeuvre. Elles nécessitent la mise en place de serveur d'objets. On retrouve principalement sur ce marché les serveurs d'EJB libres et propriétaires.

Pour ces trois familles de solutions, des produits Open Source existent et sont de plus en plus adoptés dans les administrations et entreprises (TomCat, JBoss, JonAS…).

3. J2EE en pratique

3.1 J2EE contre ses concurrents

Chaque plateforme de développement a ses avantages et ses inconvénients. Le premier avantage de cette plateforme est qu'elle a été adoptée par les plus grands groupes dans le monde entier. Nous parlons, bien entendu d'IBM, Oracle, BEA …

Celle-ci est également stable et fiable, en effet, elle existe depuis 1998 et évolue constamment. Même si c'est Sun Microsystems qui organise les spécifications de cette plateforme, elle a depuis le début su écouter les retours de développeurs. Sun a mis en place un système de spécifications pour lesquelles les développeurs peuvent indiquer leurs besoins, solutions ou mécontentement.

De plus, la plateforme s'appuyant sur Java, permet d'avoir des applications totalement indépendantes de la plateforme système utilisée (aussi bien Windows que Linux ou Mac OS).

Son « concurrent » principal est la plateforme .Net (développé par Microsoft). Cette plateforme, bien que plus facile à prendre en main, manque de maturité et même si elle séduit certaines entreprises, elle est plutôt utilisée pour les projets beaucoup moins importants, complexes que ceux utilisant J2EE.

.Net n'a bien sûr rien à envier à J2EE et réciproquement. De plus, l'interopérabilité étant de plus en plus exploitée, J2EE et .Net peuvent communiquer ensemble de façon transparente.

3.2 Composants standards contre framework

Nous vous avons présenté le standard J2EE avec l'ensemble de ses composants. Cependant il est parfois lourd d'utiliser un composant conçu pour de grande architecture alors que notre application est restreinte.

L'ensemble de la communauté OpenSource (principalement) s'est occupé (et s'occupe) de lancer sur le marché des frameworks servant à simplifier l'utilisation de telle ou telle technologie.

Souvent plus limité que le standard, les framework sont plus simple d'utilisation et plus performant dans certains cas.

3.2.1 EJB vs Hibernate & Spring

Les EJB peuvent parfois être la bête noire des développeurs J2EE. En effet, ils sont pas évident à mettre en place et sont souvent lourds à l'utilisation. Ils s'intègrent le plus souvent dans les projets de grandes envergures.

Pour les projes plus courants et plus petits, les framework Hibernate et Spring peuvent très largement remplacer ces EJB.

3.2.1.1 Présentation

Les EJB gèrent l'accès et le traitement des données (persistantes ou non). Pour faire de même avec l'utilisation de framework externe, il faut en utiliser deux en général.

- Hibernate : c'est un framework qui permet de « mapper » une base de données relationnelle en objets (POJO : Plain Old Java Object). Il permet donc d'abstraire totalement l'accès à la base de données et propose donc un accès totalement orienté objet aux données.

- Spring : c'est un framework qui permet de « remplacer » la lourdeur des serveurs d'application lourds. En effet, on parle de « conteneur léger ». Il prend en charge la création et la mise en relation d'objets par l'intermédiaire d'un fichier de configuration décrivant l'ensemble de ces relations. L'un des avantages principal est qu'il n'impose pas d'hériter ou d'implémenter une quelconque interface ou classe contrairement aux EJB.

3.2.1.2 Utilisation

L'utilisation de ces deux frameworks dans une même application est recommandée, en effet Hibernate vous permet d'accéder aux données alors que Spring vous servira de « grosse fabrique automatisée » !

De plus ces deux framework s'intègrent très facilement et ne nécessite qu'un moteur de servlet. Contrairement aux EJB qui nécessitent un serveur d'application les gérants (beaucoup plus lourd !).

3.3 Servlet & JSP vs Struts

Avec l'arrivée des JSP après celle des Servlets, les développeurs ont pu commencer à séparer de façon remarquable la couche présentation de la couche application / traitement. Cependant la maintenance du code et la lourdeur d'utilisation des servlets ont montré leurs limites…

Le modèle MVC que chaque développeur pensait de son côté était à chaque fois trop limité et peu évolutif.

L'arrivée de Struts à permis d'avoir une base solide répondant à un modèle MVC bien cadré.

3.3.1 Présentation

Struts est un framework qui permet de construire des applications web. Il se base sur la technologie Servlet / JSP en ajoutant la prise en charge du Modèle MVC2 (Modèle Vue Contrôleur). Il fournit la charpente d'une application web et évite aux développeurs d'avoir à gérer de façon fastidieuse la séparation Modèle Vue Contrôleur.

3.3.2 Utilisation

L'utilisation de ce framework est quasiment omniprésente dans le développement d'application web. Cependant Struts n'est pas la seule technologie aidant au développement de la couche application / présentation web. En effet, on peut également retrouver JSF (Java Server Faces) ou Cocoon …

3.4 Serveurs d'application : utilisations et limites

Les serveurs d'application se sont développés depuis la création de J2EE. On peut distinguer principalement 2 grandes catégories de serveurs :

- Open Source : évolue grâce à la communauté
- Propriétaire : évolue selon l'éditeur

Chaque catégorie a ses avantages et ses inconvénients. Nous allons décrire les serveurs les plus connus afin d'avoir une vision globale des solutions disponibles.

3.4.1 Open Source

3.4.1.1 Tomcat : Apache

Tomcat est un conteneur de servlet qui implémente la référence officielle pour les Servlet Java et les JSP. Ce serveur est très répondu pour les applications web.

Source : http://jakarta.apache.org/tomcat/index.html

Technologies implémentées :

- JSP
- Servlet
- JDBC
- JNDI

3.4.1.2 Jonas : ObjectWeb

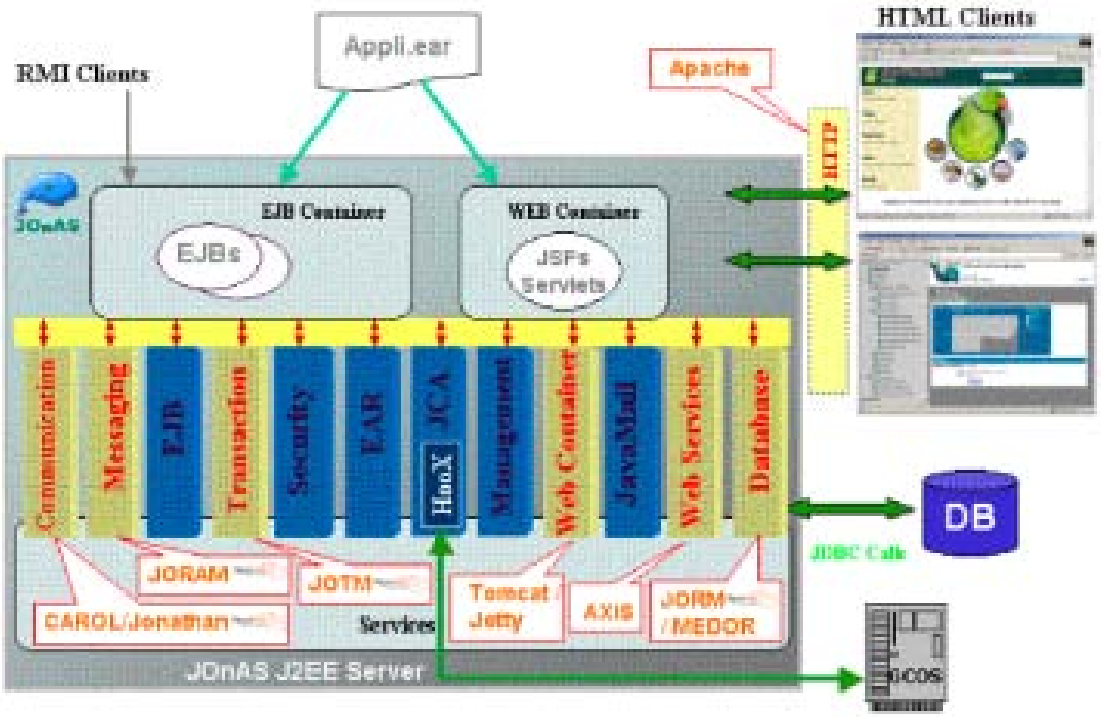
Jonas est un serveur d'application implémentant la référence officielle pour les EJB. Il intègre un lien avec Tomcat afin d'intégrer les fonctionnalités pour les applications web.

Source : http://jonas.objectweb.org/

Technologies implémentées :

- JSP
- Servlet
- EJB
- JCA
- JDBC
- JTA
- JMS
- JMX
- JNDI
- JAAS
- JavaMail

Architecture :



3.4.1.3 JBoss : JBoss

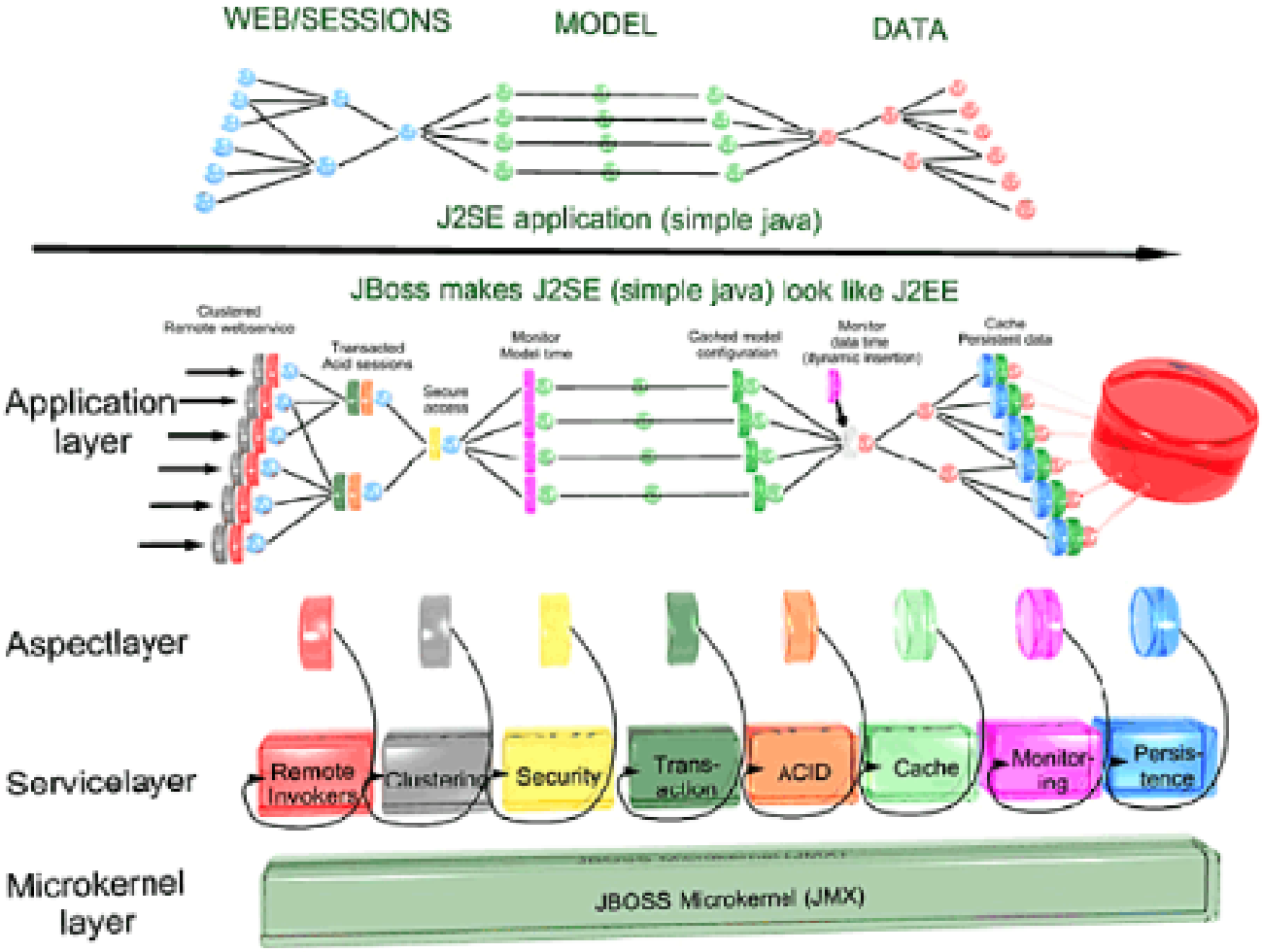
JBoss accumule les mêmes fonctionnalités que Jonas. Cependant son architecture est assez différente et repose principalement sur un « BUS ».

Des projets gravitent autour de ce serveur tels que des plug-in pour Eclipse, des modules pour l'AOP (Aspect Oriented Programming), Hibernate …

JBoss est l'un des serveurs d'application les plus populaires dans l'Open Source (avec Jonas). Il est également de plus en plus utilisé en milieu professionnel.

Source : http://www.jboss.org/products/jbossas

Architecture :



Propriétaire

Les serveurs d'application propriétaires se démarquent grâce à des outils facilitant le développement et/ou la configuration des applications au sein du serveur d'application.

En contrepartie, ils font payer les licences d'utilisation de leur serveur.

Voici un ensemble de serveurs les plus utilisés :

- WebSphere : IBM
- WebLogic : BEA
- WebObject : Apple
- Oracle Application Server : Oracle

3.5 IDE (Integrated Development Environment)

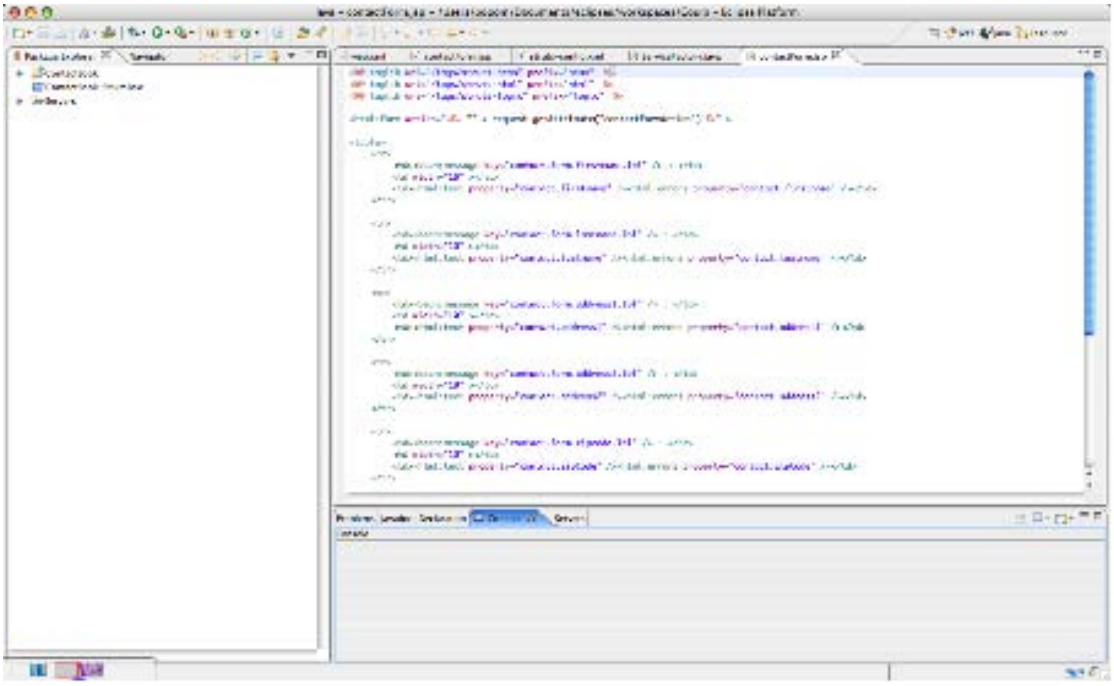
Pour développer des applications complexes, il faut impérativement un IDE (Integrated Environnement Development).

De même qu'avec les serveurs d'application, il existe les IDE Open Source et ceux qui sont propriétaires.

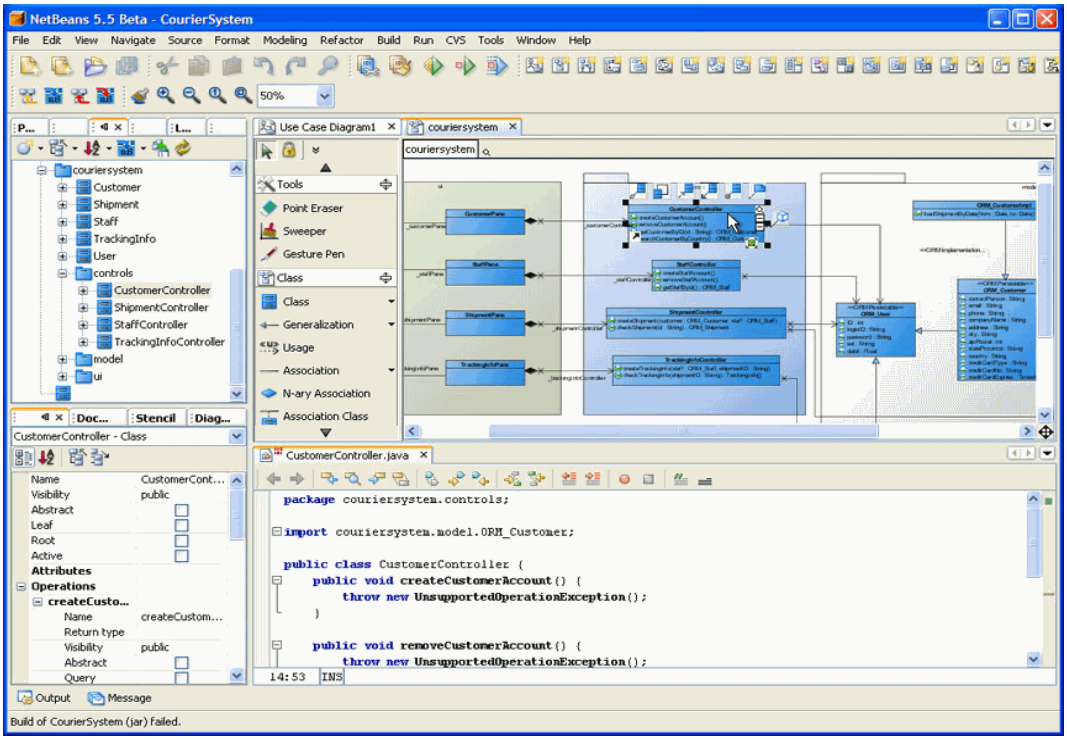
Voici une présentation de quelques IDE les plus connus.

3.5.1 Open Source

3.5.1.1 Eclipse: IBM

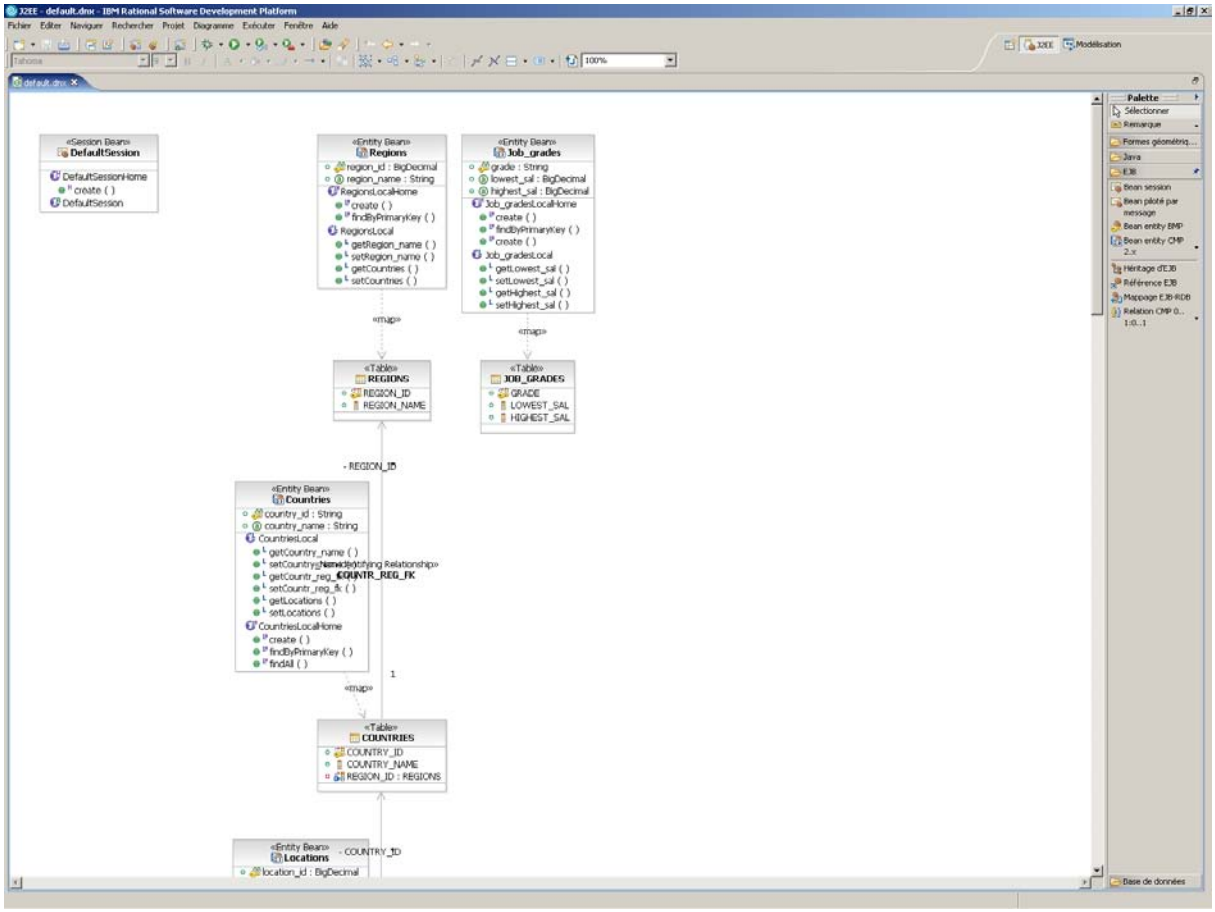


3.5.1.2 NetBeans : SUN (Oracle)

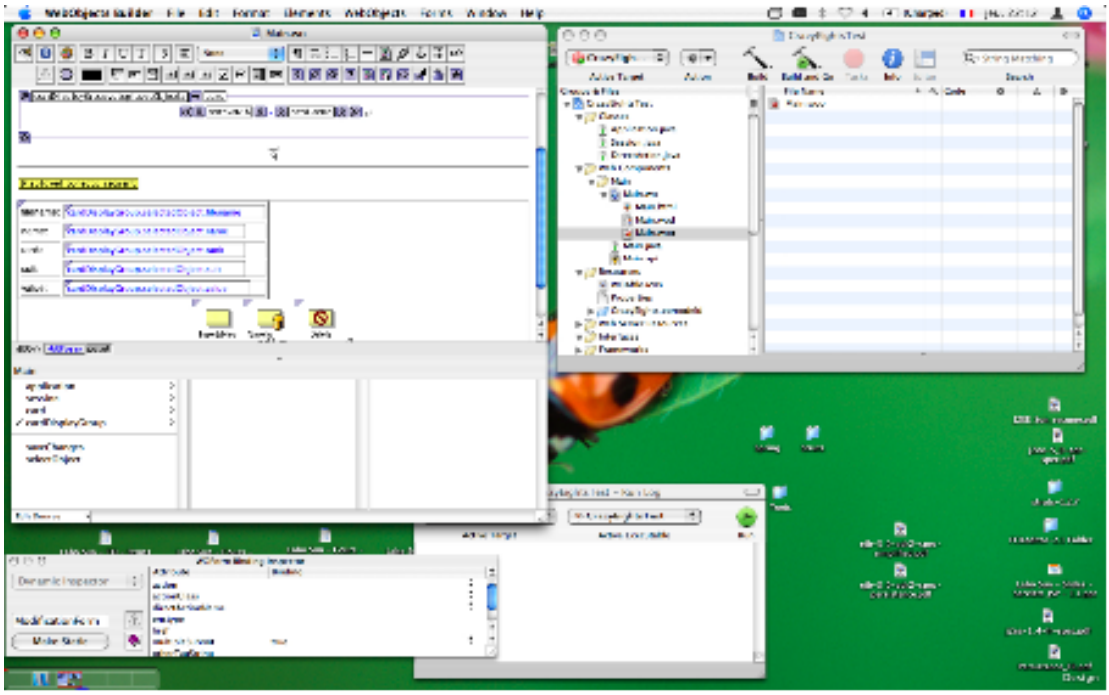


3.5.2 Propriétaire

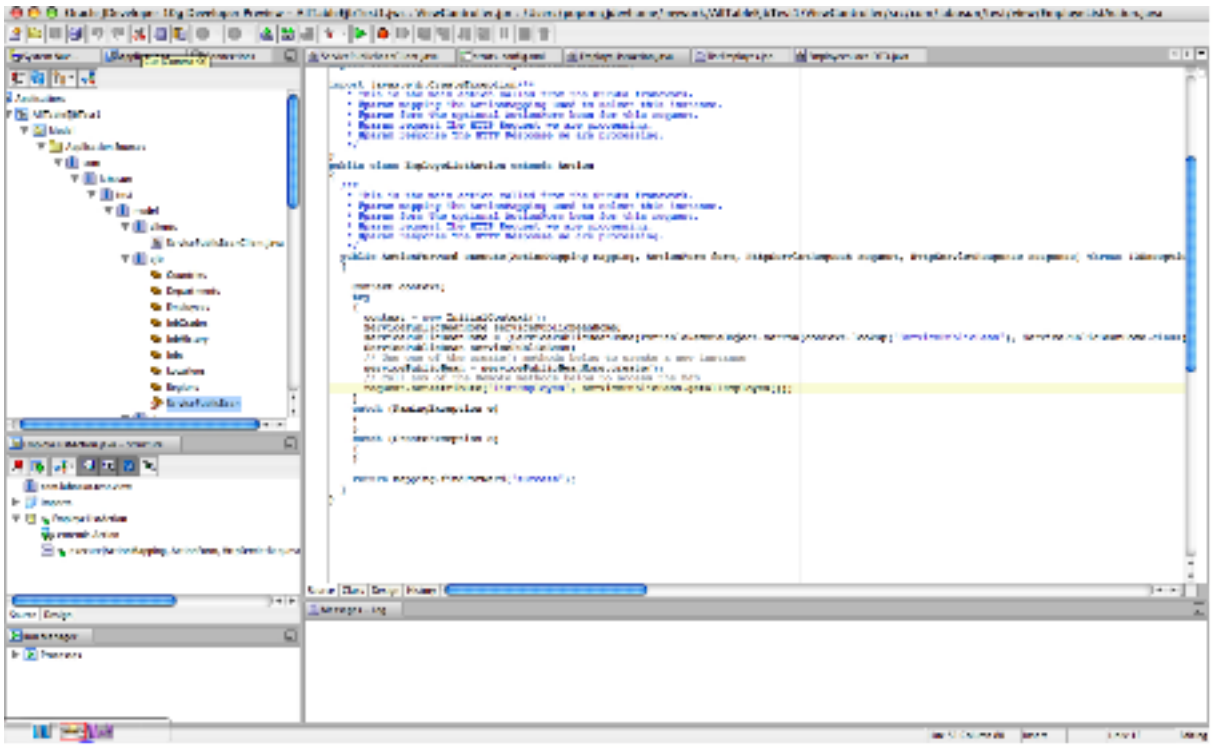
3.5.2.1 Rational Architect (avec WebSphere) : IBM



3.5.2.2 XCode & WebObject : Apple



3.5.2.3 JDev: Oracle

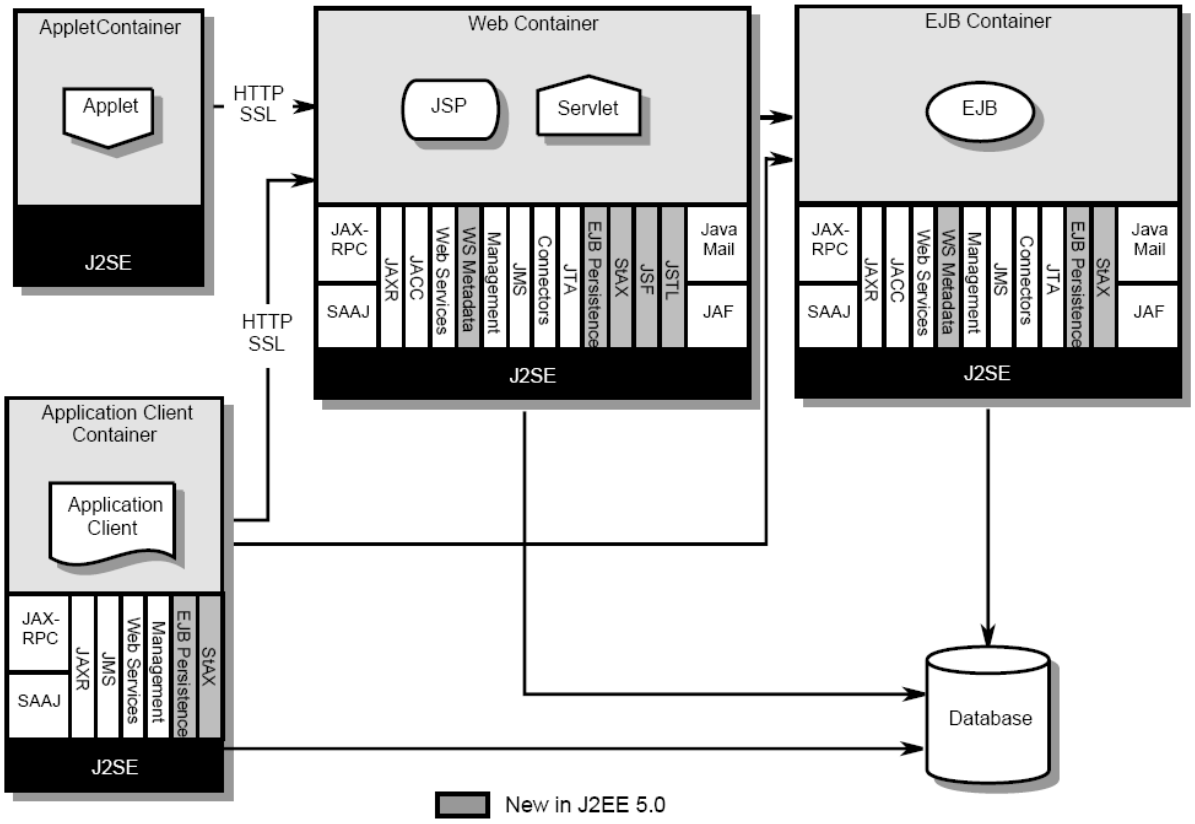


Le future de J2EE

J2EE est une plateforme en constante évolution. Nous sommes actuellement à la version 1.4 mais la version 1.5 est en cours de développement.

Cette version corrigera des spécifications parfois trop complexes. Elle ajoutera également de nouveaux composants à la plateforme afin de faciliter toujours plus le développement d'application d'entreprise.

Voici la nouvelle architecture prévue pour cette future version :



Le JDK 1.5 intègre de nouvelles fonctionnalités, c'est donc pour cela que la version J2EE 5 les utilisera afin d'avoir une plateforme bien plus abordable que la précédente et beaucoup plus optimisée.