Optimisation des applications web dynamiques pour les besoins du commerce électronique : cas particulier de Java et MySQL

En hommage à Abu Abdullah Mohammad Ibn Musa al-Khawarizmi

Kamal AOUDA, Publishing

Sommaire

A propos de l'auteur de ce livre

5 années d'expérience en conseil acquises dans les secteurs publics et privés. Riche expérience régionale et internationale. Empathie culturelle. Double compétence en gestion et en technologies de l'information. Bilingue en français et en anglais. Actuellement à la recherche de nouvelles missions dans l'optimisation des processus d'affaires, le conseil en technologies de l'information ou le développement d'applications web (J2EE et .Net).

Je peux vous aider à accroître votre retour sur investissement, optimiser vos processus d'affaires et réduire le coût de développement et de maintenance de vos applications. Qu'il s'agisse de la continuité de vos plans d'affaires ou de la conformité de vos processus avec les benchmarks et la réglementation en vigueur, je suis capable d'implémenter des solutions efficaces et efficientes dans les entreprises de petite, moyenne et grande taille. Ma méthodologie d'intervention est conçue pour automatiser au maximum vos processus d'affaires, pour préserver l'intégrité de vos données et pour codifier les connaissances métiers indispensables à votre réussite. Pour plus de détails visiter les sites:

- http://www.kamalaouda.com/consultingsite pour une description detaillée de mes services.

- http://www.kamalaouda.com/ECtuning site d'accompagnement du livre.

Conditions d'utilisation

Toute demande spéciale doit être acheminée à l'adresse kamalaouda@kamalaouda.com. L'auteur comprend l'anglais, le français et l'arabe. Il s'engage à vous répondre dans les meilleurs délais si votre demande est recevable.

N'oubliez pas de visiter le site d'accompagnement de ce livre à l'adresse http://www.kamalaouda.com/ECtuning à partir du 25 mai 2005.

Bonne lecture et merci pour vos remarques et suggestions

1. Présentation

1.1 Problématique de l'essai

Selon une étude publiée par Zona Research[1], les entreprises américaines auraient subi en 2001 une perte[2] de 25 billions de dollars US à cause des problèmes de performance qui ont affecté leurs applications[3] de commerce électronique. Comme le montre le tableau 1, c'est le modèle d'affaires B2C qui serait le plus touché par ces pertes essentiellement dans les secteurs de la publication en ligne, de l'intermédiation financière, du tourisme et du voyage.

Tableau 1: Répartition par secteurs d'activité des pertes potentielles dues aux problèmes de performance affectant les applications de commerce électronique de type B2C[4]

POSSIBLE B2C SEGMENT IMPACTS OF LOAD SPEED LOSSES				
Calculated Losses from Unacceptable Load Speeds		$ 362,215,882 Per month		
Est. Business-to-Consumer Share of Total Web Market		28.3%		
Est. Business-to-Consumer Share of Load Speed Loss		$ 102,507,095 Per month		
Column A	Column B	Column C	Column D	Column E
Business to Consumer Market Segment	**Percentage of B2C Market**	**Theoretical Share of Load Speed Losses**	**Monthly Site Avg If 10 Major Sites In Each Segment**	**Annual Average if 10 Sites in Each Segment**
Securities Trading	32.5%	$ 33,314,806	$ 3,331,481	$ 39,977,767
Travel/Tourism	27.4%	$ 28,086,944	$ 2,808,694	$ 33,704,333
Book Publishing	11.4%	$ 11,685,809	$ 1,168,581	$ 14,022,971
Groceries	7.3%	$ 7,483,018	$ 748,302	$ 8,979,621
Personal Finances	3.7%	$ 3,792,763	$ 379,276	$ 4,551,315
Recorded Music	3.5%	$ 3,587,748	$ 358,775	$ 4,305,298
Box Office Receipts	2.8%	$ 2,870,199	$ 287,020	$ 3,444,238
Textiles/Apparel	2.6%	$ 2,665,184	$ 266,518	$ 3,198,221
Other	8.8%	$ 9,020,624	$ 902,062	$ 10,824,749

Le cas particulier des applications implémentées à l'aide des technologies Java mérite d'être soulevé. En effet, un rapport d'IDC Research estime que moins de 20% des applications[5] J2EE arrivent à satisfaire les standards de performance. Les retombées financières sont énormes puisque :

[1] Source: Zona Research, Need for speed 2, http://www.keynote.com/downloads/Zona_Need_For_Speed.pdf

[2] Cette perte est évaluée en termes de manque à gagner c'est à dire en calculant le montant des revenus qui auraient pu être acquis si ces problèmes avaient été évités.

[3] Source: Ethan Henry, Brad Micklea, Bridging the Java™ 2, Platform Enterprise Edition (J2EE™) Technology Performance Gaps, JavaOne 2004

[4] Source: Zona Research, Inc., The Economic Impacts of Unacceptable Web-Site Download Speeds, http://www.webperf.net/info/wp_downloadspeed.pdf, August 2000

[5] Dans le reste de ce document les termes site et application seront utilisés de façon interchangeable.

- Le marché des technologies J2EE est évalué actuellement à 2 billions de dollars[6].
- Plus de 85% des entreprises du Fortune 500 utiliseraient J2EE dans l'implémentation de leurs middle tiers (pour accroître l'interopérabilité des systèmes existants[7] mais aussi pour greffer de nouveaux services à leur noyau)[8].
- Le taux de pénétration de Java dans les entreprises serait le même que celui de Visual Basic [9](selon Gartner 80% des entreprises dans le monde utiliseraient ce langage).
- J2EE est devenu un standard de fait[10] (de facto standard) dans près de 75% des entreprises clientes de la Borland Software Corporation[11].

Contrairement à une croyance largement répandue les problèmes de performance sont beaucoup plus attribuables à des anomalies dans l'analyse, le design et la programmation des applications qu'à des insuffisances et/ou des dysfonctionnements au niveau des infrastructures matérielles[12] qui permettent de les mettre en réseau de les héberger et de les exécuter. Paradoxalement, les entreprises ont tendance à privilégier les solutions d'optimisation matérielles[13] dès qu'elles sont confrontées à des problèmes de performance. Un rapport[14] du Gartner Group confirme, à ce sujet, qu'elles font appel auxdites solutions pour améliorer les performances de 75% des applications J2EE nouvellement déployées. **Paradoxalement**:

- 40% des problèmes de performance sont dus à des erreurs commises par les opérateurs des applications (erreurs d'origine humaine).
- Seulement 20% des problèmes de performance sont dus à la défaillance du matériel et/ou l'inéquation de son environnement[15].

[6] Source: Segue Software, Inc., ACHIEVING HIGH PERFORMANCE J2EE APPLICATIONS**,** January 2003

[7] Legacy systems

[8] Source: Borland White Paper, Maximizing business value by optimizing J2EE™ performance, June 28, 2004

[9] Source: Thomas Malvehill, Key Challenges in developing and deploying J2EE applications, July 28, 2003

[10] Format, langage ou protocole qui est devenu un standard non pas en raison de son approbation par un organisme de standardisation mais en raison de sa grande popularité.

[11] Source: Borland presentation on Performance management for the J2EE platform, http://www.pcuf.fi/sytyke/kerhot/javasig/PerfMan_J2EE.pdf

[12] Indisponibilité du réseau, serveur avec une mémoire ou un processeur sous- dimensionnés par rapport à la charge de travail, panne du serveur, sinistres dus à des aléas climatiques…

[13] Cela n'est pas sans liens avec la baisse continue des prix du matériel informatique.

[14] Teresa Lanowitz, Tearing Down the Wall, Gartner, 2002

[15] Température excessive, rupture du courant électrique…

- Par contre **40% des problèmes de performance sont imputables à des anomalies persistantes dans le design et le code des applications.**

Si les deux premières causes préalablement citées découlent respectivement d'une formation insuffisante du personnel et d'un mauvais dimensionnement des infrastructures matérielles, la troisième cause trouve plutôt son origine dans **la non intégration du management des performances à l'ensemble des phases du cycle de développement.** En effet comme en témoignent plusieurs rapports sur la toile, les analystes, les architectes et les programmeurs des applications de commerce électronique ne tiennent pas toujours compte des benchmarks de performance dans les phases d'analyse, de design et de codage. Ils attendent que le code soit entièrement écrit pour tester sa capacité à performer dans un environnement où les contraintes d'utilisation[16] sont équivalentes à celles de l'environnement réel de production.

Le report du management des performances à la fin du cycle de développement amène les évaluateurs à constater d'importants dépassements par rapport aux échéanciers et aux budgets prévisionnels surtout lorsque la méthodologie de développement utilisée permet peu ou pas d'itérations entre les phases de test, d'analyse, de design et de codage.

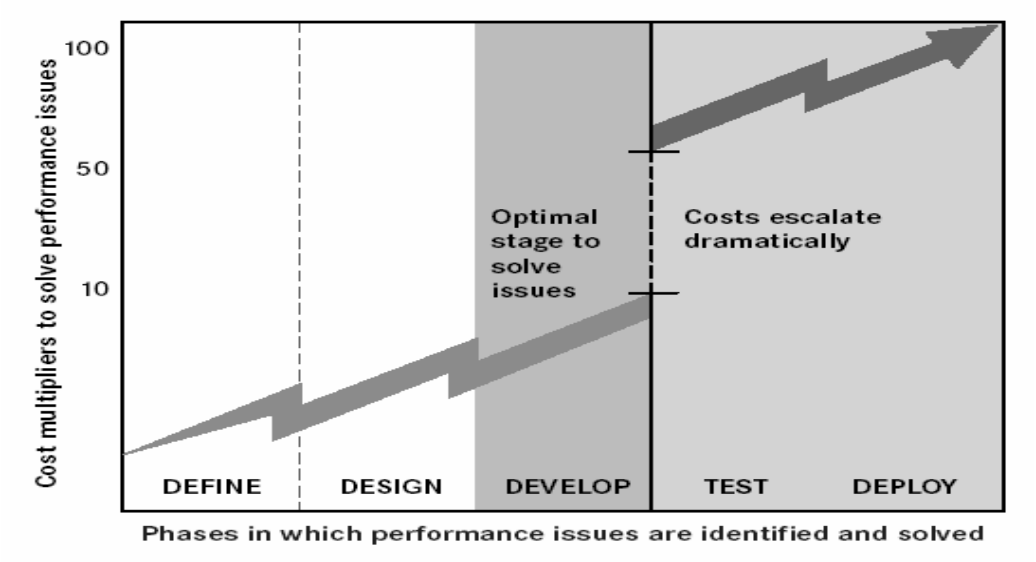
A en croire une présentation de la Borland Software Corporation[17] les ajustements apportés aux livrables desdites phases entraîneraient un retrait de 50% des applications déjà mises en service et des dépassements budgétaires évalués à 30% du coût total des projets[18]. L'importance de ces dépassements justifie la nécessité de concevoir les applications de commerce électronique dans une perspective d'optimisation, et ce, dès les premières phases du cycle de développement.

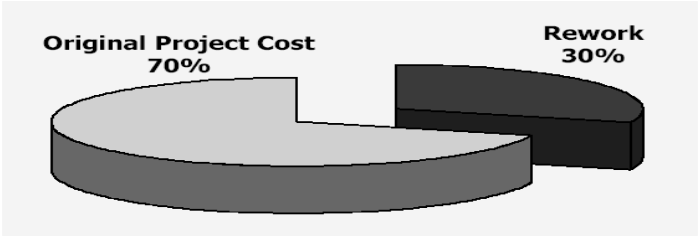
[16] Utilisation intensive des données et des traitements, accès concurrentiels…

[17] Jon Ha Building Quality into developement., Borland softwares,
http://www.java.no/web/moter/javazone03/presentations/JonHarrison/JH_QualityDevelopment_JavaZone.pdf

[18] Ces dépassements sont particulièrement importants dans les méthodologies dites séquentielles parce qu'elles contiennent peu ou pas d'itérations.

Figure 1: Dépassements budgétaires dus au report du management des performances à la fin du cycle de développement[19]





1.2 Objectif de l'essai

Cet essai qui traite du cas particulier des applications de commerce électronique implémentées à l'aide des technologies Java et MySQL, propose un référentiel pour l'intégration du management des performances aux trois premières phases du cycle de développement (analyse, design, codage).

Les technologies Java ont été choisies pour les raisons énumérées dans la section 1.1 et qui ont trait essentiellement au fort taux de pénétration du langage et au faible pourcentage des applications J2EE qui arrivent à satisfaire les standards de performance. Quant au SGBDR MySQL il a été choisi pour les raisons suivantes :

[19] Source : ACCELERATE YOUR PERFORMANCE, Borland softwares, http://www.borland.com/optimizeit/pdf/opt6_datasheet.pdf

- **Sa grande popularité :** d'après la société MySQL AB[20], ce SGBDR compte aujourd'hui plus de 5 millions d'installations actives.
- **Une utilisation accrue dans des applications web sensibles aux problèmes de performance :** MySQL est actuellement utilisé par des entreprises notoires dans des applications web où les données et les traitements sont utilisés de façon intensive (vente au détail, télécommunications, finances, santé…).

  A noter que dans le secteur du commerce électronique, MySQL a été rapidement adopté par des clients prestigieux comme Yahoo, Lycos Europe, Google et Powell's books[21].
- **L'intérêt grandissant de la communauté Java:** en effet dans un sondage réalisé par MySQL AB, 42.7% des développeurs souhaitent assister à une formation sur les développements combinant Java et MySQL (contre 24.3% pour les développeurs dotNet et 15% seulement pour les développeurs Perl)[22].
- **En perspective, une version 5.0 qui propose des fonctionnalités phares pour mieux adresser les problèmes de performance :** il s'agit principalement des procédures stockées dont l'absence a désavantagé les versions antérieures de MySQL par rapport à MS-SQL Server et Oracle. Une version bêta sortie en Janvier 2005 contient en plus des procédures stockées d'autres fonctionnalités centrées sur les besoins des entreprises (notamment les triggers, une réplication protégée contre les crashs et un cache pour les définitions de tables)[23].

Le référentiel proposé est compatible avec les méthodologies itératives qui adhèrent aux principes suivants:

- **Construction de l'application par assemblage de composants:** cette approche permet de détecter rapidement les anomalies de codage et/ou de design.
- **Utilisation d'UML comme notation pour la production des artefacts :** l'extensibilité de la notation permet de créer des profils pour associer les métriques de la performance aux différents diagrammes d'UML.

[20] Source : http://www.MySQL.com
[21] Une liste complète de ces entreprises est disponible à l'adresse : http://www.MySQL.com/customers/#Retail
[22] Source: http://dev.MySQL.com/tech-resources/quickpolls/
[23] Source: http://dev.MySQL.com/doc/MySQL/fr/todo-MySQL-5-1.html

- **Impliquer dans la gestion des performances les managers, les analystes, les architectes, les programmeurs, les testeurs, les administrateurs des bases de données et les administrateurs des serveurs web/application** : tous doivent être impliqués au même degré pour s'assurer que les besoins en performance sont convenablement pris en charge à travers tout le cycle de développement. Cette implication les amène à **considérer la performance comme un besoin d'affaires et non un réglage technique de dernière minute.**

1.3 Périmètre de l'essai

Comme son titre l'indique, cet essai ne traite que des problèmes de performance qui sont dus à des anomalies dans l'analyse, le design et le codage des applications de commerce électronique et des bases de données auxquelles elles sont adossées.

Sont exclues du périmètre du présent essai les anomalies attribuables :

- Au réseau qui connecte l'application à l'Internet.
- Au serveur web et au serveur d'application.
- A la version de la Java Virtual Machine utilisée.
- A tout matériel utilisé par l'application localement ou à distance.
- Aux scripts exécutés du côté du client (Vbscript, Javascript).

Notons par ailleurs que l'essai ne couvre que les phases d'analyse, de design et de codage. Les phases de test, de déploiement et de maintenance ont été sciemment exclues parce qu'il existe déjà un nombre conséquent de livres et d'articles qui traitent du management des performances dans ces 3 phases.

1.4 Méthodologie adoptée

Nous commencerons par définir la performance dans le contexte du commerce électronique ainsi que les métriques et les benchmarks qui permettent de l'évaluer aussi bien par les utilisateurs et les managers que par l'équipe de développement. Nous expliquerons par la suite comment le comportement d'un visiteur sur un site peut influencer la performance de l'application et les choix en matière de design et de codage.

Au niveau de la phase d'analyse nous procéderons à l'identification des besoins en performance à partir des cas d'utilisation qui rentrent typiquement dans la construction d'une application de commerce électronique. Nous effectuerons ensuite un profilage de ces besoins en tenant compte des spécificités du modèle d'affaires (B2B, B2C, C2C). Au terme de cette phase nous présenterons l'essentiel sur le profil UML for Schedulability Performance and Time et nous expliquerons comment il peut être utilisé pour améliorer la qualité de la communication au sein de l'équipe de développement.

Au niveau de la phase de design nous passerons en revue plusieurs anti-patterns et patterns de performance qui s'appliquent aussi bien aux composants de l'application qu'à l'architecture à laquelle ils sont intégrés.

Au niveau de la phase de codage nous analyserons les problèmes de performance liés à l'utilisation des objets qui rentrent habituellement dans la construction d'une application de commerce électronique. Nous proposerons au fur et à mesure des solutions pour les résoudre ou en limiter les conséquences. Les éléments couverts dans cette phase sont :

J2SE, J2EE:

- Les chaînes de caractère (classe String).
- Les collections (Vectors, Hashtables…).
- Les boucles.
- Le casting et le wrapping.
- La gestion des erreurs (bloc catch, try).
- Les fichiers .JAR.
- Les applets.
- Les Servlets et les Java Server Pages.
- Les Enterprise Java Beans.
- Les API des bases de données et les connecteurs (JDBC, connector J…).

MySQL :

- Le choix des moteurs de stockage et des types de données.
- Les techniques d'indexation.
- Les procédures stockées.
- L'optimisation des requêtes.

1.5 Résultats escomptés

Constituer un référentiel pour remédier aux problèmes de dépassement de temps et de coûts engendrés par le report des travaux d'optimisation à la fin du cycle de développement. Les apports de ce référentiel se situent à plusieurs niveaux :

- Proposer une démarche concrète pour intégrer le management des performances aux premières phases du cycle de développement.
- Définir les métriques et les benchmarks qui permettent d'évaluer la performance des applications de commerce électronique. Cette définition va au delà de l'analyse des indicateurs traditionnels comme le débit et le temps de réponse. Elle est centrée sur les cas d'utilisation, le modèle de comportement des utilisateurs et la veille concurrentielle.
- Proposer une démarche pour analyser et spécifier les besoins en performance au même titre que les besoins d'affaires.
- Recenser et classifier les principaux patterns et anti-patterns de performance et évaluer leur applicabilité aux applications de commerce électronique.
- Mettre en exergue les erreurs fréquemment commises dans la programmation des applications de commerce électronique et proposer des solutions pour les éviter ou amenuiser leur impact sur les performances.
- Faire le point sur les nouvelles fonctionnalités de la version 5.0 de MySQL et plus particulièrement celles qui contribuent à l'amélioration des performances.

2. La performance dans le contexte du commerce électronique

Dans la littérature les définitions de la performance abondent mais sont centrées pour la plupart sur 3 critères fondamentaux, à savoir :

- **Le temps :** selon ce critère est considéré(e) comme performant(e), tout(e) personne, organisation ou système qui réalise ses objectifs dans les délais impartis.
- **Le coût :** selon ce critère la performance se mesure par rapport à la capacité de réalisation des objectifs au moindre coût.
- **Le rendement :** selon ce critère la performance se mesure par rapport à la maximisation des extrants (outputs) associés à chaque ressource nouvellement consommée (qu'il s'agisse de temps et/ou d'argent).

Bien que claires ces définitions restent trop générales pour les besoins du présent essai. Elles doivent être affinées davantage pour mieux refléter les spécificités technologiques et fonctionnelles des applications de commerce électronique ainsi que les besoins des acteurs qui participent à leur développement. Cela nous amène à décliner les définitions supra en tenant compte des besoins des utilisateurs et des managers d'une part et ceux de l'équipe de développement d'autre part.

2.1 La performance telle que perçue par les utilisateurs et les managers[24]

2.1.1 Le temps de réponse

Du point de vue de l'utilisateur, le temps de réponse est la métrique la plus représentative de la performance d'une application Web[25]. En effet parce que quelques clics séparent le site d'une entreprise de celui des concurrents, il est très important que les requêtes soient traitées dans les plus brefs délais. Les temps de réponse acceptables pour l'utilisateur suscitent plusieurs controverses chez les experts en optimisation mais avant d'en parler, il est primordial d'analyser l'évolution des vitesses de connexion à l'Internet aussi bien au niveau des ménages qu'au niveau des entreprises.

[24] Les managers dont il est question sont supposés relever des fonctions métiers et non de la fonction chargée des technologies de l'information.
[25] Cette affirmation est démontrée par plusieurs études spécialisées dont les plus connues sont:

Zona Research, Need for speed 2, http://www.keynote.com/downloads/Zona_Need_For_Speed.pdf

Jakob Nielson, The Need for speed à l'adresse: http://www.useit.com/alertbox/9703a.html

Du côté des ménages, les connexions à 56 Kbps sont toujours majoritaires malgré la compétition acharnée à laquelle se livrent les FAI pour démocratiser les accès à haut débit en Amérique du nord. Une étude menée par Nielsen//NetRatings en 2004 conclue, à cet égard, que 51.39% des ménages américains se connectent toujours à l'Internet via un modem à 56 Kbps ou moins contre 48.61% pour les connexions à haute vitesse (voir figure 2).Du côté des entreprises, les connexions à haut débit représentaient 79.8% en 2004 contre 20.2% pour les connexions à 56 Kbps ou moins (voir figure 3).

Figure 2: Vitesse de connexion à l'Internet des ménages aux États-unis[26]

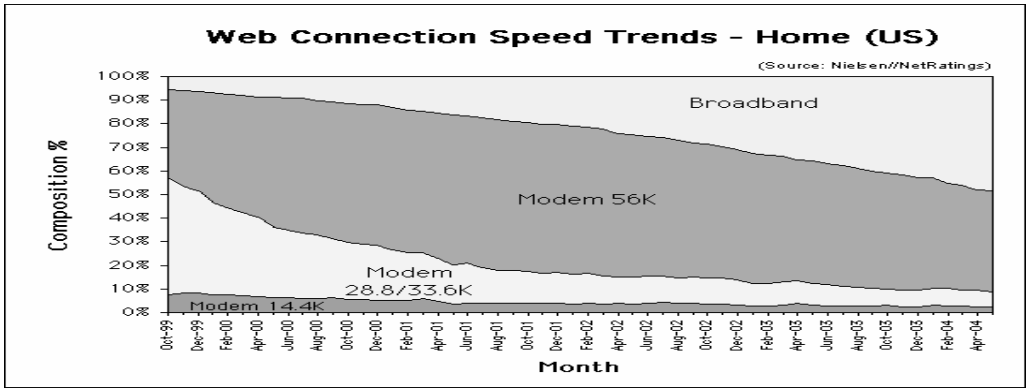
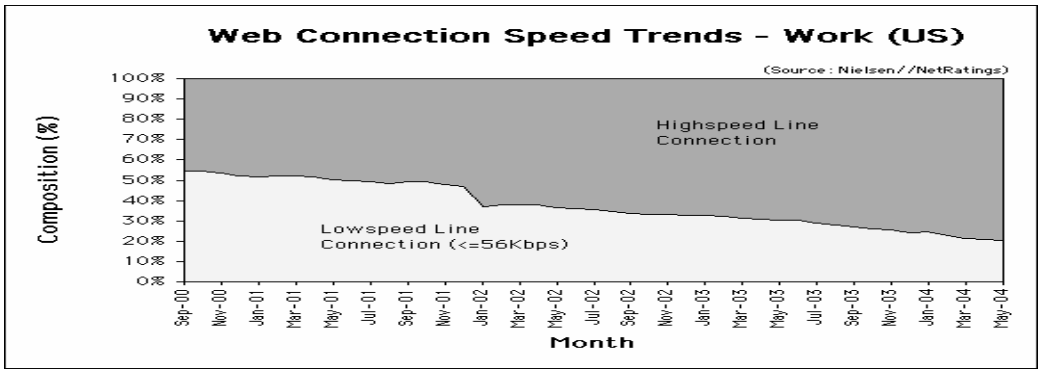


Figure 3: Vitesse de connexion à l'Internet des entreprises aux Etats-Unis[27]



Des experts en utilisabilité comme Jakob Nielsen[28] recommandent un affichage des résultats au bout de 6 à 10 secondes suivant l'envoi de la requête mais ces seuils doivent être ramenés au contexte d'utilisation parce la ténacité d'un internaute face à la latence d'un site de commerce électronique ne dépend pas seulement de sa vitesse de connexion mais également :

[26] Source: Nielsen//NetRatings, Web connection speed trends—U.S., http://www.websiteoptimization.com/bw/0406/
[27] Idem
[28] Pour plus de détails consulter l'article: The Need for Speed à l'adresse: http://www.useit.com/alertbox/9703a.html

- **De la rareté et/ou de la cherté du bien/service que l'utilisateur souhaite acheter** : ainsi si le bien/service est disponible à profusion à des prix équivalents ou moindres, l'utilisateur l'achètera sur le site d'une entreprise concurrente. Par contre si le bien/service est offert par peu de sites en quantités limitées ou fait temporairement l'objet d'une promotion, l'utilisateur sera certainement plus tenace vis-à-vis de la latence (à moins d'envisager une alternative brique et mortier). A titre d'exemple, une étude menée par Keynote montre que la ténacité des utilisateurs face à la latence est plus grande pour les sites d'intermédiation financière parce que leurs transactions sont très sensibles au temps (changement rapide des cours, risque de subir des pénalités si la transaction est reportée…).

Tableau 2: **La ténacité des utilisateurs serait plus grande sur les sites d'intermédiation financière**[29]

Keynote Brokerage Indices		
Brokerage Latencies (sec)	**56kbps**	**100Mbps**
Home Page	15.83	1.47
Login	8.35	0.75
Quote Entry	17.45	2.24
Quote Return	6.34	0.76
Order Confirmed	10.27	2.18
Logout	8.20	0.88
Total Latency	66.44	8.28

- **Du cas d'utilisation :** par exemple l'affichage de la page d'accueil demande moins de ténacité que la passation d'une commande ou le règlement d'un achat. Idem pour le remplissage d'un panier d'achat et l'inscription à une lettre de nouvelles…
- **Du modèle d'affaires :** une étude menée par Keynote[30] montre à ce sujet que les utilisateurs dans un modèle B2B sont moins tenace vis-à-vis de latence que les utilisateurs d'un modèle B2C (cela n'est pas sans liens avec la prédominance des connexions à haute vitesse chez les entreprises, Cf. figure 3).
- **Du coût de la connexion** : en ce sens qu'un utilisateur qui paye un forfait pour une connexion illimitée pourrait manifester plus de ténacité face à la latence qu'un utilisateur qui est facturé en fonction du temps de connexion.

Vu la multiplicité des facteurs qui influent sur la ténacité face à la latence, il est très difficile de fixer un seul temps de réponse pour tous les profils d'internautes et tous les cas d'utilisation. Idéalement les entreprises devraient mesurer sur les sites de leurs concurrents le temps de réponse associé à chaque cas d'utilisation.

[29] Source: Zona Research, Need for speed 2, http://www.keynote.com/downloads/Zona_Need_For_Speed.pdf
[30] Source: Keynote Systems Inc, E-COMMERCE RESPONSE TIME: A REFERENCE MODEL, http://www.avoka.com/resources/keynote/E-Commerce_Response_Time_CMG_2000__Chris_.pdf

Les résultats ainsi obtenus pourraient servir de benchmarks de performance à travers tout le cycle de développement. Des sociétés spécialisées effectuent régulièrement ce type de mesures[31] sur des sites à forte affluence et publient leurs résultats sous forme de benchmarks. Parmi ces sociétés c'est Keynote (Nasdaq "KEYN" The Internet Performance Authority)[32] qui jouie de la plus grande notoriété notamment en raison de la rigueur de sa démarche et de la fiabilité de ses résultats. Le reste de cette section passe en revue les benchmarks qui reflètent le mieux les spécificités des sites de commerce électronique. Il est important de noter que ces benchmarks ne sont pas conçus pour mesurer la performance d'une technologie donnée (J2EE, dotNet ou autre) mais uniquement le temps de réponse associé aux cas d'utilisation qui rentrent habituellement dans la composition d'une application de commerce électronique (rechercher un article dans un catalogue, ouvrir un compte, s'authentifier …).

2.1.1.1 Benchmark pour les transactions E-commerce (Keynote E-Commerce Index)

Ce benchmark mesure, sur des sites de vente au détail, les temps de réponse associés aux cas d'utilisation suivants :

- S'authentifier en renseignant un identifiant et un mot de passe (cela suppose que l'utilisateur possède déjà un compte sur le site en question).
- Rechercher un produit/service dans un catalogue.
- Ajouter ce produit/service au panier d'achat.
- Régler le dit produit/service.

Les sites sont sélectionnés en fonction de l'importance de leur part de marché. Leurs serveurs sont concentrés dans dix grandes métropoles américaines[33] et sont connectés à l'Internet par l'intermédiaire des plus grands FAI aux États-unis. Les mesures sont prises automatiquement par des agents (écrits en Java) qui simulent des transactions fantômes[34] par le biais du navigateur MIE sous Windows 2000. Elles sont effectuées quotidiennement entre 8 h et minuit via des connexions à haute vitesse (variant de T1 à T3). La médiane des mesures ainsi obtenues est publiée tous les mercredis en exclusivité sur le site d'E-commerce Times à l'adresse : http://ecommercetimes.com

[31] Pour plus de détails sur la méthodologie de mesure se rendre à l'adresse:
http://www.keynote.com/keynote_method/keynote_method_main_tpl.html
[32] http://www.keynote.com/about_us/about_us_tpl.html
[33] Boston, Chicago, Dallas, Detroit, Houston, Los Angeles, New York, Philadelphia, San Francisco et Washington, D.C
[34] Traduction littérale de ghost transactions. Il s'agit d'un ensemble de transactions fictives qui affectent des sites réels et qui simulent le comportement d'achat sur Internet.

Tableau 3: Résultats du Keynote E-Commerce Web Transaction Performance Index, semaine du 24 Janvier 2005[35]

RESPONSE TIME				"SUCCESS RATE"			
Rank	Site	Time (seconds)	Last Week	Rank	Site	Success Rate (percent)	Last Week
	Index	**15.18**	**15.47**		**Index**	**99.45**	**99.23**
1	Office Depot	10.14	1	1	Eddie Bauer	100.00	1
2	Eddie Bauer	10.66	2	2	JC Penny	99.81	2
3	Wal-Mart	11.64	3	2	Wal-Mart	99.69	4
4	JC Penny	13.33	4	4	Office Max	99.62	6
5	Amazon	13.49	5	5	Office Depot	99.53	7
6	Best Buy	17.38	6	6	Best Buy	99.52	2
7	Sears	19.68	7	7	Sears	99.36	5
8	Costco	23.13	8	8	Amazon	99.33	9
9	Office Max	25.00	9	9	Costco	98.21	8

2.1.1.2 Benchmark pour les transactions bancaires (Keynote E-Banking Web Transaction Performance Index)

Ce benchmark mesure les temps de réponse associés aux cas d'utilisation suivants :

- S'authentifier sur le site d'une banque en renseignant un identifiant et un mot de passe.
- Afficher le relevé de compte et en vérifier le solde.
- Quitter le site.

La méthodologie de mesure utilisée est identique à celle du Keynote E-Commerce Index. La médiane des résultats obtenus est publiée hebdomadairement sur le site de Digital Transactions[36].

Tableau 4: Résultats du Keynote E-Banking Web Transaction Performance Index[37]

Rank By Speed (seconds)
Week of 01-24-2005

Rank	Target	Response Time (sec)	Rank Last Week
	Index	**10.17**	
1	Etrade Bank	4.92	1
2	Washington Mutual	7.84	2
3	SunTrust	8.72	4
4	Bank of America	9.58	3
5	Wells Fargo	10.38	5
	Worst Average	**20.95**	

Rank By Success Rate (percentage)

Rank	Target	Success Rate (%)	Rank Last Week
	Index	**98.97**	
1	Bank of America	100.00	4
2	Etrade Bank	99.82	3
3	Wells Fargo	99.72	6
4	Chase	99.61	7
5	Washington Mutual	99.43	9
	Worst Average	**96.44**	

Wachovia was excluded from this week's index due to insufficient data points.

[35] Source: Keynote E-Commerce Transaction Performance Index, http://ecommercetimes.com/ectpi/#infofaq
[36] http://www.digitaltransactions.net/index.cfm?pageid=22
[37] Source: http://www.digitaltransactions.net/index.cfm?pageid=22.

2.1.1.3 Benchmark pour les connexions à 56 Kbps (Keynote Consumer 40 Internet Performance Index)

La particularité de ce benchmark est qu'il mesure le temps de réponse nécessaire pour l'affichage d'une page d'accueil par le biais d'une connexion à 56 Kbps[38] (établie via des modems V90 et transitant par les FAI suivants : AT&T, XO, Earthlink, MSN et AOL). Les mesures sont prises toutes les heures, 7 jours sur 7 entre 5 am et 9 pm. Elles ne tiennent pas compte du temps requis pour établir la connexion avec le point de présence (POP) du FAI.

Tableau 5: Benchmark pour la connexion aux pages d'accueil via des connexions à 56 Kbps[39]

Week Of
01.23.05 - 01.29.05

KC40 Index	30.61
Google [6]	4.98
FedEx [6]	6.86
Ask Jeeves [108]	10.16
Infospace [6]	12.29
Fidelity [161]	14.41
UPS [6]	16.48
Yahoo! [17]	17.16
Symantec [6]	18.49
Verizon [2]	20.67
Bank of America [3]	21.16
Anonymous	**61.65**

Week Of
01.16.05 - 01.22.05

KC40 Index	30.01
Google [5]	4.61
FedEx [5]	6.10
Ask Jeeves [107]	9.75
Infospace [5]	11.49
Fidelity [160]	13.76
UPS [5]	15.37
Yahoo! [16]	17.10
Symantec [5]	17.27
Bank of America [2]	19.35
Verizon	19.65
Anonymous	**81.50**

ABC, MSNBC and Wal-Mart will be removed from this week's index due to insufficient data points.

2.1.1.4 Benchmarks sectoriels

Toujours selon la même méthodologie[40], Keynote conçoit des benchmarks pour mesurer les temps de réponse spécifiques à des secteurs où le commerce électronique est omniprésent. Il s'agit notamment :

- Du **Keynote Airline Web Transaction Performance Index :** qui mesure le temps de réponse nécessaire pour s'authentifier et vérifier le solde courant sur une carte de points (flyers miles)[41].
- Du **Keynote Hotel Web Transaction Performance Index**: qui mesure le temps de réponse nécessaire pour réserver une chambre d'hôtel.
- Du **Keynote Travel Agency Web Transaction Performance Index**: qui mesure le temps de réponse nécessaire pour rechercher une offre packagée (chambre d'hôtel+billet d'avion) sur le site d'une agence de voyage.

[38] Pour les autres benchmarks la connexion se fait via des lignes T1 à T3.
[39] Source : http://www.keynote.com/solutions/performance_indices/consumer_index/consumer_40.html
[40] Pour plus de détails sur la dite méthodologie Voir 2.1.1.1 *Keynote E-Commerce Web Transaction Performance Index*
[41] Kilométrages gratuits offerts aux clients au fur et à mesure de leurs achats.

Tableau 6: Les benchmarks sectoriels de Keynote, semaine du 24 Janvier 2005[42]

The Keynote Airline Web Transaction Performance Index

Rank By Response Time

Week of 01-24-2005

Rank	Target	Response Time (sec)	Rank Last Week
1	Southwest	8.67	4
2	jetBlue	8.91	1
3	Delta	10.35	2
4	American	11.58	3
5	Continental	12.21	5
6	United	13.89	6
	Index	**14.94**	
7	Alaska	15.22	7
8	US Airways	15.26	8
9	Northwest	17.52	9
10	Air Canada	24.24	10
11	airTran	27.02	11
12	America West	36.80	12

Rank By Success Rate

Rank	Target	Success Rate (%)	Rank Last Week
1	Northwest	100.00	1
2	Delta	99.91	1
3	American	99.72	6
4	US Airways	99.54	7
5	airTran	99.35	10
6	jetBlue	99.34	3
7	United	99.16	4
8	Alaska	98.81	4
9	Continental	98.79	8
10	America West	98.51	12
	Index	**97.98**	
11	Air Canada	92.75	11
12	Southwest	89.81	9

The Keynote Hotel Web Transaction Performance Index

Rank By Response Time

Week of 01-24-2005

Rank	Target	Response Time (sec)	Rank Last Week
1	Marriott	11.25	1
2	Hyatt Regency	13.49	2
3	Westin	16.04	3
4	Hilton	16.20	4
5	Sheraton	16.37	5
	Index	**18.02**	
6	Intercontinental	18.98	6
7	Holiday Inn	24.99	8
8	Wyndham	29.41	9
9	Radisson	39.98	10

Rank By Success Rate

Rank	Target	Success Rate (%)	Rank Last Week
1	Holiday Inn	99.72	3
2	Hyatt Regency	99.63	1
2	Marriott	99.63	2
4	Radisson	99.06	9
5	Westin	99.05	5
6	Hilton	99.04	7
7	Intercontinental	98.62	6
	Index	**98.10**	
8	Sheraton	97.59	4
9	Wyndham	90.13	10

Crowne Plaza was excluded from this week's report due to insufficient data points.

The Keynote Travel Agency Web Transaction Performance Index

Rank By Response Time

Week of 01-24-2005

Rank	Target	Response Time (sec)	Rank Last Week
1	Hotels.com	20.17	1
2	Expedia	22.17	2
	Index	**32.54**	
3	Travelocity	32.59	3
4	Orbitz	41.80	4
5	Cheaptickets	43.44	5

Rank By Success Rate

Rank	Target	Success Rate (%)	Rank Last Week
1	Hotels.com	99.18	3
2	Expedia	98.31	2
3	Orbitz	96.52	4
	Index	**95.83**	
4	Cheaptickets	94.78	1
5	Travelocity	90.74	5

Priceline was excluded from this week's report due to insufficient data points.

2.1.2 Le débit

Fondamentalement cette métrique intéresse beaucoup plus les managers que les utilisateurs. Elle mesure dans un laps de temps donné[43] la charge de travail que l'application web est capable de traiter. Sur un site Internet plusieurs indicateurs sont traditionnellement utilisés pour mesurer la charge de travail. Les plus communs sont:

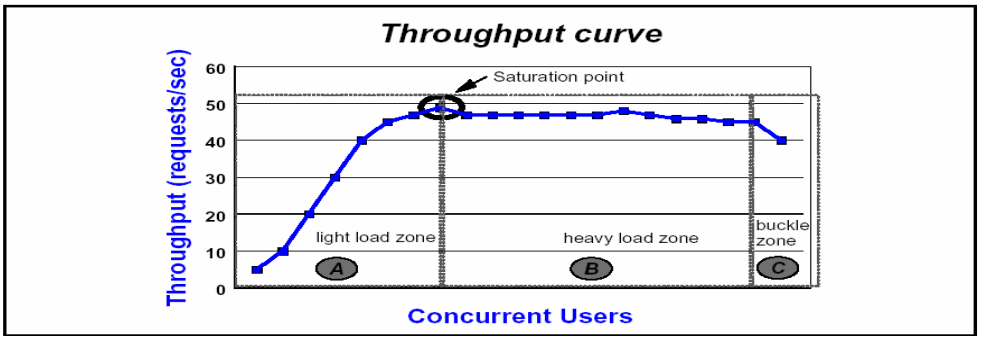
[42] Source: http://www.keynote.com/solutions/performance_indices/travel_hospitality/results-122704.html
[43] Une seconde en général

- Le nombre de requêtes ou de réponses http.
- La quantité de bits transférée entre le client et le serveur.
- Le nombre d'accès concurrentiels (utilisateurs accédant simultanément au site).
- Le nombre de pages consultées. Il est important de noter que ce nombre n'est pas toujours égal à celui des requêtes http. En effet parce qu'une page Web contient plusieurs composants[44] le fureteur doit générer de multiples requêtes avant de pouvoir l'afficher en totalité.
- Le nombre de transactions. Dans le domaine du Web le mot transaction n'a pas la même acception que dans le domaine des bases de données. Il désigne le couplet formé par une requête http et la réponse y afférente[45].

Typiquement, l'évolution du débit d'une application de commerce électronique suit 3 phases comme le montre la figure 4.

- **Une phase de croissance** : durant laquelle le débit augmente linéairement au fur et à mesure de l'augmentation de la charge de travail (Cf. light load zone).
- **Une phase plateau** : durant laquelle le débit de l'application demeure stable quelque soit la charge de travail traitée (Cf. heavy load zone).
- **Une phase de déclin :** pendant laquelle le débit se dégrade lorsque la charge de travail croît (Cf. buckle zone).

Figure 4: Courbe de débit d'un site de commerce électronique[46]



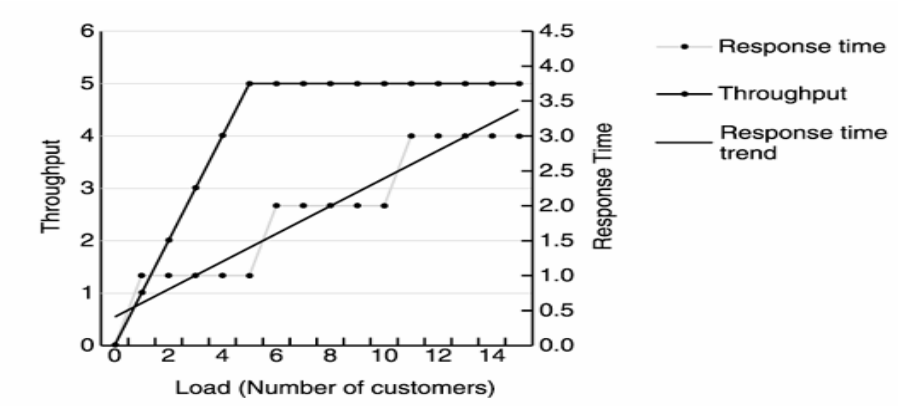
La relation entre le temps de réponse, la charge de travail et le débit pendant chacune de ces phases est expliquée ci-après :

[44] Images, sons, vidéos, autres pages incluses dans des frames…
[45] Source de la definition: Stacy Joines, Ruth Willenborg, Ken Hygh, Performance Analysis for Java™ Web Sites, Addison Wesley, September 10, 2002, ISBN: 0-201-84454-0, pages 464
[46] Source: Gennaro (Jerry) Cuomo, Srini Rangaswamy, IBM WebSphere Application Server 4.0 Performance Tuning Methodology

- **Pendant la phase de croissance :** le temps de réponse moyen reste relativement stable lorsque la charge de travail augmente.
- **Pendant la phase plateau :** le débit est constant mais le temps de réponse augmente de façon linéaire lorsque la charge de travail augmente.
- **Pendant la phase de déclin :** le temps de réponse croît de manière exponentielle lorsque la charge de travail augmente (la figure 5 illustre l'ensemble de ces relations).

La comparaison entre le fonctionnement des sites de commerce électronique et celui des magasins brique et mortier permet de mieux comprendre les relations mathématiques entre ces trois métriques au niveau des phases de croissance et de plateau.

Figure 5: Relation entre le débit, le temps de réponse et la charge de travail[47]

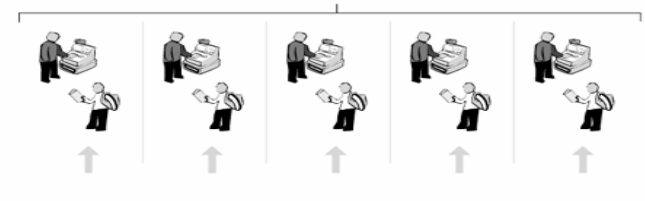


Supposons que dans une épicerie dotée de 5 caisses le règlement d'un article demande[48] :

- 1 minute si le nombre de clients est inférieur ou égal au nombre de caisses en service. Dans ce cas il n'y a pas de file d'attente. Le temps de réponse comprend uniquement le temps nécessaire pour le règlement[49] de l'achat et le débit de l'épicerie[50] augmente graduellement en fonction du nombre de clients qui se présentent à la caisse (de 0 à 5 clients par minute). Cette situation reflète la même dynamique que la phase de croissance d'un site de commerce électronique.

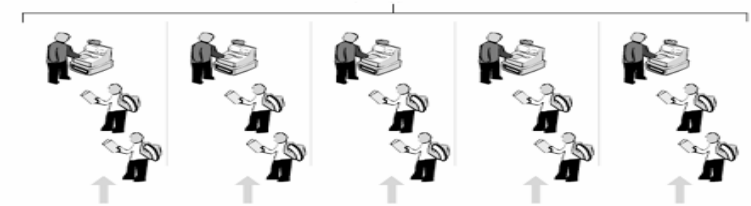
[47] Source: Stacy Joines, Ruth Willenborg, Ken Hygh, Performance Analysis for Java™ Web Sites, Addison Wesley, September 10, 2002, ISBN: 0-201-84454-0, pages 464
[48] Ces données sont fictives et reposent sur un modèle très simplifié.
[49] Lecture du code à barre, insertion de la carte bancaire dans le terminal de paiement, emballage…
[50] Nombre d'articles achetés par minute

Figure 6: Equivalent de la phase de croissance dans un magasin brique et mortier



- 2 minutes si le nombre de client est supérieur à 5 et inférieur ou égal à 10. Dans ce cas les 5 clients supplémentaires doivent attendre que les caissiers aient fini de servir leurs prédécesseurs. Le temps de réponse comprend donc le temps d'attente en plus du temps nécessaire pour le règlement de l'article. Il augmente de façon linéaire à chaque fois que cinq clients additionnels se présentent simultanément à la caisse. Le débit pour sa part reste constant du 6ème au 10 ème client (en effet même avec cette charge de travail l'épicerie sert toujours 5 clients par minute).

Figure 7 : Equivalent de la phase plateau dans un magasin brique et mortier



A l'instar du temps de réponse, une entreprise se doit de mesurer sur le site de ses concurrents le débit associé aux cas d'utilisation qui rentrent typiquement dans la construction d'une application de commerce électronique (Cf. tableau 7). Encore une fois pour qu'elles répondent aux préoccupations des managers ces mesures doivent faire abstraction de toute contrainte technologique (autrement dit elles doivent mesurer le débit relatif aux cas d'utilisation et non celui des technologies qui ont servi à implémenter le site de commerce électronique).

Tableau 7: Exemple de benchmarks[51] pour la mesure du débit (cas d'utilisation: accès à la page d'accueil d'un site de commerce électronique)[52]

Throughput (Bytes/Second)		
Category Leaders	**At-Home**	**At-Work**
Portals - Yahoo	1,991	28,318
eShopping - WebVan	923	8,005
Brokerages - Ameritrade	1,306	8,402
Travel - Yahoo! Travel	1,407	20,481
Overall Worst Site	526	11,200

[51] Source : Yahoo Wins Throughput Race, http://reviews-zdnet.com.com/AnchorDesk/4520-6033_16-4204488.html
[52] Ces résultats sont basés sur le Keynote Consumer 40 Index. Pour plus d'informations sur la méthodologie du benchmark voir section 2.1.1.3

2.2 La performance telle que perçue par l'équipe de développement

Les métriques définies dans la section 2.1 ne répondent pas entièrement aux besoins de l'équipe de développement qui, à la différence des utilisateurs et des managers, doit identifier quels éléments de l'application et du réseau sont à l'origine des problèmes de performance. Pour se prêter à leurs besoins, ces métriques doivent être segmentées et enrichies en tenant compte :

- Des composants qui rentrent dans la construction de l'application.
- Des cas d'utilisation et du modèle de comportement de l'utilisateur.

2.2.1 Segmentation du temps de réponse en fonction des composants de l'application

Parce que tout est objet dans Java, l'approche la plus patente pour identifier les anomalies qui augmentent le temps de réponse consiste à :

1. Identifier pour chaque fonctionnalité du site les classes utilisées, leurs interactions mutuelles et leur ordonnancement dans le temps (les diagrammes de séquence d'UML sont particulièrement adaptés pour schématiser ce genre d'interactions).
2. Chronométrer le temps nécessaire pour l'instanciation des dites classes et l'appel de leurs interfaces. (**Les interfaces dont il est question ici sont les méthodes et les propriétés des classes. Pour éviter toute confusion, les classes d'interface seront dorénavant désignées par des majuscules**).
3. Repérer, sur cette base, les impasses et les goulots d'étranglement (c'est à dire les classes dont l'instanciation ou les interfaces provoquent un blocage ou exposent un temps de réponse anormalement long).
4. Analyser le code encapsulé dans lesdites classes pour relever les anomalies qui compromettent la performance.

2.2.1.1 Injection d'un compteur dans le code source (Custom Instrumentation)

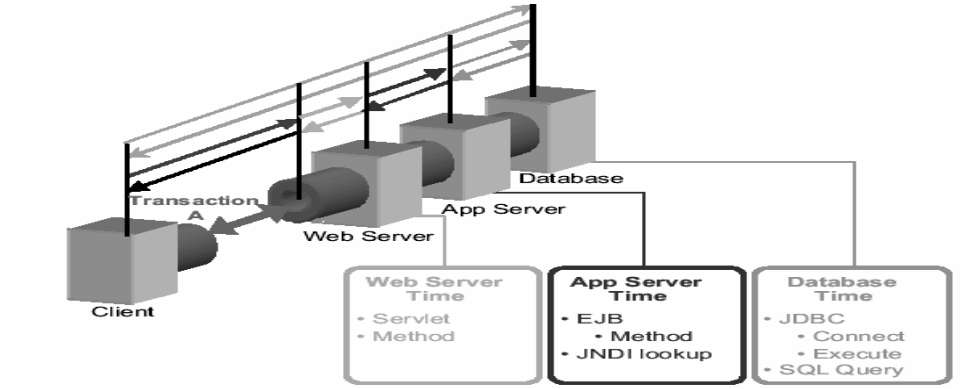
La technique expliquée dans cette section consiste en l'injection d'un compteur dans le code source des classes dont on souhaite mesurer la performance[53].

[53] Comme nous le verrons plus loin cette technique a plusieurs inconvénients.

Le compteur en question ne doit pas consommer plus de 1% du temps requis pour l'exécution des opérations auxquelles il est associé. Dans le cas contraire, le recours à cette technique doit être remis en cause surtout dans les applications fortement distribuées. Pour notre part, nous recommandons l'utilisation du compteur System.currentTimeMillis()parce que son exécution sur un PC standard[54] consomme moins de 0.5 milliseconde.

Cela dit, si une transaction web fait, en arrière plan, appel aux services d'une servlet, des EJB et de JDBC on peut segmenter le temps de réponse total de la manière suivante :

Figure 8: Exemple d'une segmentation en fonction des composants de l'application[55]



1. Mesurer le temps nécessaire pour acheminer la requête http du client vers le conteneur de la servlet.
2. Mesurer le temps requis pour instancier la servlet. Pour cela on peut placer un compteur System.currentTimeMillis() au début de la méthode init(ServletConfig config)[56] qui est automatiquement appelée par le conteneur lorsqu'il termine l'instanciation de la servlet. A noter, toutefois, que cette mesure ne doit se faire que pour la première requête entrante si l'INTERFACE[57] implémentée par la servlet n'est pas de type SingleThreadModel[58].

[54] Au moins PII, 128Mo, JVM *1.1.8* et plus, Windows 2000
[55] Source: Diagnosing J2EE performance problems throughout the application life cycle. MERCURY INTERACTIVE, www.mercuryinteractive.com
[56] Pour plus d'informations sur la signature de cette méthode se rendre à l'adresse : http://www.jakarta.apache.org/tomcat/tomcat-5.0-doc/servletapi/javax/servlet/http/HttpServlet.html
[57] INTERFACE en majuscules désigne les classes d'interface et non les méthodes et les propriétés.
[58] Le package javax.servlet comprend une INTERFACE SingleThreadModel qui permet de créer autant d'instances de la servlet qu'il y a de requêtes entrantes. En règle générale, il faut éviter l'implémentation de cette INTERFACE pour ne pas épuiser rapidement et vainement les ressources du serveur.

En effet de la deuxième jusqu'à la n^{ème} requête, il n'y aura pas de nouvelle instanciation car la première instance persiste en mémoire et crée à l'intérieur du processus qui lui est assignée, un thread indépendant pour chaque requête nouvellement reçue. Dans ce contexte, une nouvelle instanciation de la servlet n'aura lieu que si son code source est recompilé ou si le conteneur est redémarré.

3. Mesurer le temps de réponse nécessaire pour passer la requête http à la servlet ainsi créée. Cela peut se faire en plaçant un compteur au début de la méthode service() qui est invoquée automatiquement par le conteneur après la création de la servlet. A noter que contrairement aux méthodes init() et destroy() qui ne sont appelées qu'une seule fois dans le cycle de vie de la servlet, la méthode service() est invoquée autant de fois qu'il y a de requêtes entrantes ou de réponses y afférentes.
4. Associer le, cas échéant, un compteur aux méthodes des classes qui implémentent l'INTERFACE Filter[59] pour mesurer le temps qu'elles consomment avant d'envoyer/recevoir les requêtes/réponses à/de la servlet. Cette mesure ne sera, cependant, pertinente que si le code contient des instructions pour intercepter les requêtes/ réponses en amont /aval de la servlet (par exemple pour effectuer par le biais de la méthode doFilter() une authentification de l'utilisateur, une validation ou un encryptage des données).
5. On peut par ailleurs placer deux compteurs respectivement au début et à la fin de la méthode doFilter() pour calculer le temps de réponse total de la servlet aux multiples requêtes qu'elle reçoit[60] via les classes implémentant l'INTERFACE Filter (voir snippet 1).

Snippet 1: Exemple d'un compteur associé à la méthode doFilter() d'une classe qui implémente l'interface Filter[61]

```
public void doFilter(ServletRequest request,
ServletResponse response,
FilterChain chain) throws IOException,
ServletException {
long before = System.currentTimeMillis( ); // compteur
```

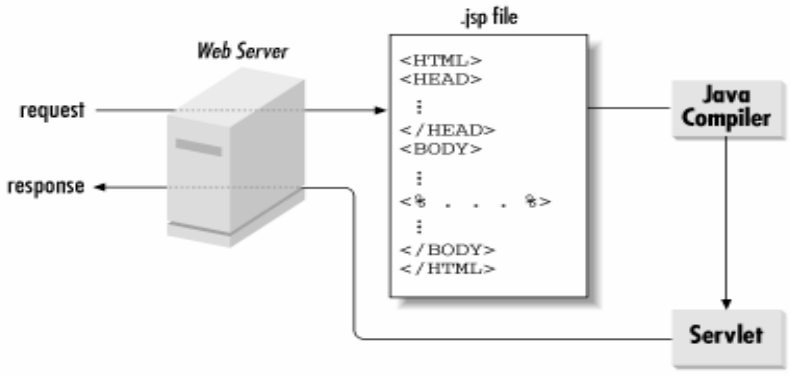
[59] public void init(FilterConfig filterConfig). public void doFilter(HttpServletRequest request, HttpServletResponse response, FilterChain chain). public void destroy()
[60] Nous rappelons à ce sujet que contrairement aux méthodes init() et destroy() qui ne sont appelées qu'une seule fois dans le cycle de vie du Filter la méthode doFilter() est invoquée autant de fois qu'il y a de requêtes entrantes ou de réponses y afférentes.
[61] Source: Jack Shirazi, Java Performance Tuning, 1st Edition September 2000 , ISBN: 0-596-00015-4, pages 496

```
chain.doFilter(request, response);
long after = System.currentTimeMillis( );
```

Remarque1 : Cette technique peut être bien sûr généralisée aux FilterChains ainsi qu'à toutes les méthodes de la servlet aussi bien en entrée qu'en sortie. Toutefois, il faut l'utiliser à bon escient pour ne pas consommer inutilement les ressources du serveur et par conséquent biaiser les résultats finaux en y incluant le temps d'exécution consommé par les compteurs.

Remarque 2 : Pour les JSP on peut quasiment effectuer les mêmes mesures que pour les servlets. En effet au moment de leur génération, le conteneur transforme toutes les JSP en servlets par le biais d'un mécanisme de translation avant de les placer sous un répertoire temporaire dont le chemin est affecté à l'attribut de contexte javax.servlet.context.tempdir (A quelques différences[62] près on peut donc associer les mêmes compteurs aux méthodes jspInit(),_jspService()…)

Figure 9: Translation des JSP en Servlets[63]

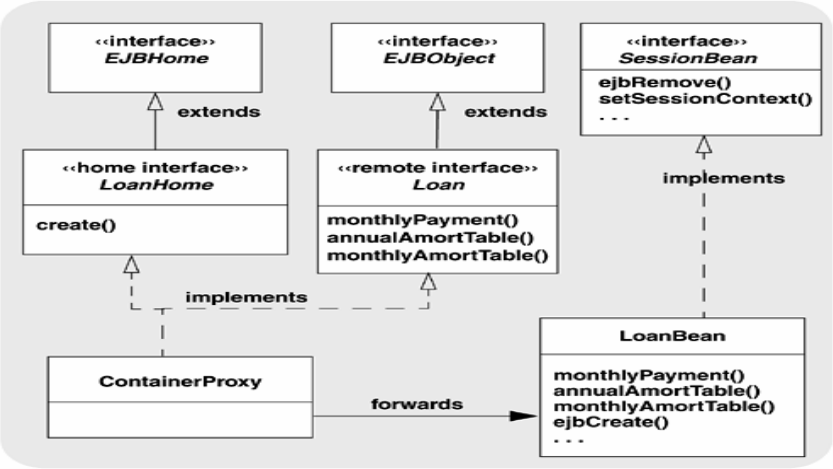


6. Mesurer le temps requis pour relayer la requête du conteneur servlet vers le conteneur EJB (par exemple si l'application utilise conjointement Tomcat comme conteneur de servlets et JBoss comme conteneur d'EJB).
7. Mesurer le temps requis pour la création de l'EJB[64] : le compteur qui sert à la mesure peut être placé à plusieurs endroits dépendamment du type d'EJB utilisé[65] et du caractère plus ou moins distribué de l'application[66].

[62] Différences qui ont trait à la syntaxe et aux noms des interfaces.
[63] Source : Jason Hunter , William Crawford, Java Servlet Programming, 2nd edition, O'Reilly, ISBN: 0-596-00040-5
[64] Un EJB est crée lorsqu'il acquiert le statut ready et que ses business methods deviennent invocables.
[65] Session, entity, message-driven beans
[66] Répartition des EJB sur une ou plusieurs JVM.

Par exemple si une application de commerce électronique utilise un stateless session bean pour éditer le tableau d'amortissement d'un prêt bancaire (Cf. figure 10), on peut associer un compteur à la méthode ejbCreate() de la classe LoanBean (cf. snippet 2). A noter que lorsqu'un client demande la création du stateless bean c'est la méthode create() du home interface qu'il invoque à charge pour le conteneur EJB de rediriger (forwards) cette invocation vers la méthode ejbCreate()[67]. La création du stateless bean n'est donc parachevée qu'après l'invocation de la méthode ejbCreate().

Figure 10: diagramme de classes d'un EJB permettant d'éditer un tableau d'amortissement[68]



Snippet 2: Exemple de compteur pour mesurer le temps nécessaire pour l'instanciation d'un stateless bean

```
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;
... // les invocations sont possibles à partir de plusieurs types de clients (servlet, EJB…)

try
{Context initial = new InitialContext();
  Object objref = initial.lookup("java:comp/env/ejb/MyLoan");
  LoanHome home =
       (LoanHome)PortableRemoteObject.narrow(objref,
```

[67] En fait create() est une méthode abstraite et ejbCreate() correspond à son implémentation dans la classe LoanBean
[68] Source: Gail Anderson, Paul Anderson, Enterprise JavaBeans Component Architecture: Designing and Coding Enterprise Applications, Prentice Hall PTR, 0-13-035571-2, pages 456

```
            LoanHome.class);
        long before = System.currentTimeMillis( ); // compteur
      Loan loanEJB = home.create();
        long after = System.currentTimeMillis( ); // compteur
        log.debug("create ran in: "+(after - before));
    }
    catch (Exception ex) {...}
```

Fondamentalement cette technique de mesure associe un compteur aux méthodes que le conteneur invoque pour gérer le cycle de vie des EJB (c'est à dire les méthodes de l'INTERFACE EJBHome[69]).

Elle peut être généralisée aux stateful session beans aussi bien pour les interfaces locales[70] (Local Home Interface) que pour les interfaces distantes[71] (Remote Home Interface). Nous émettons toutefois des réserves quant à son utilisation avec les entity beans parce que l'invocation de leur méthode ejbCreate() provoque non seulement la création d'un bean mais également l'insertion d'un nouvel enregistrement dans la base de données à laquelle il est adossé. Cela dit, pour les stateless session et les entity bean la mesure doit être effectuée autant de fois qu'il y a d'instances dans l'EJB pool du conteneur. En revanche pour les stateful session beans elle doit se faire autant de fois que l'EJB est invoqué par les clients[72] puisque le conteneur crée une nouvelle instance de l'EJB à chaque nouvelle invocation.

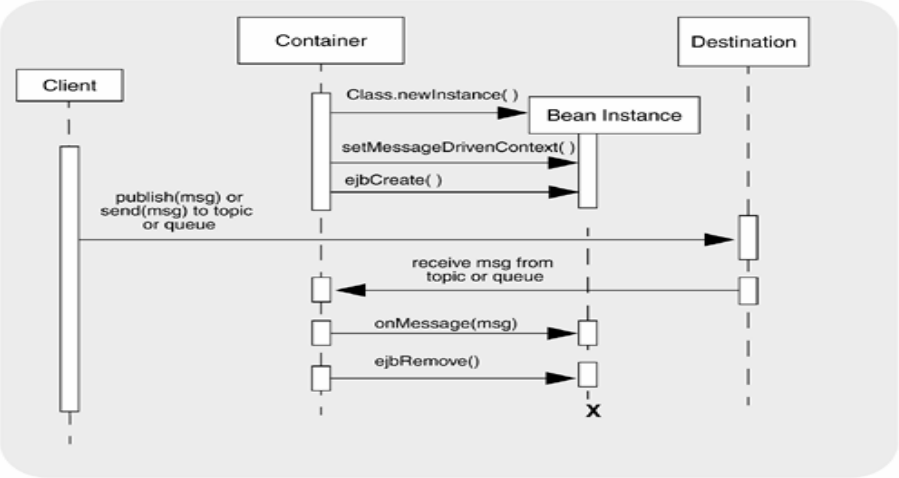
Il est important de noter que la technique présentée dans la snippet 2 ne peut pas être employée pour la mesure du temps consommé par l'instanciation d'un message driven bean. En effet, à la différence des session et entity beans, les messages driven beans n'exposent pas d'interfaces locales ou distantes à leurs clients. Rappelons que même s'ils implémentent la méthode ejbCreate() les messages driven beans ne peuvent communiquer avec leurs clients qu'à travers un serveur de messages asynchrones[73] (typiquement JMS[74]). Cette particularité nous amène à mesurer également:

[69] Le conteneur appelle ces méthodes pour gérer le cycle de vie des EJB.
[70] Invocation de la méthode create() à partir d'un client situé sur la même JVM.
[71] Invocation de la méthode create() à partir d'un client situé sur une JVM différente de celle où se trouve l'EJB.
[72] Par exemple une servlet, un autre EJB ou un serveur JMS
[73] Ces messages sont placés dans une file d'attente (Topic ou Queue) pour ne pas obliger le client à attendre la réponse du bean (Contrairement aux invocations qui bloquent le client tant que le bean ne lui a pas retourné une réponse).
[74] Java Messaging Service

- Mesurer le temps requis pour envoyer un message du client[75] vers le bean (par exemple si l'échange des messages se fait par l'entremise de JMS on peut calculer la différence entre un compteur associé à la méthode onMessage() du bean et un autre associé à la méthode publish() de la classe TopicPublisher ou encore la méthode send() de la classe QueueSender[76]).

- Et inversement, mesurer le temps requis pour envoyer un message du bean vers le client (pour cela il suffit d'inverser l'emplacement des compteurs préalablement cités).

Figure 11: Cycle de vie d'un MessageDrivenBean[77]



8. Mesurer le temps requis pour trouver l'emplacement des EJB via JNDI[78] (par exemple dans la snippet 2 il suffit de placer des compteurs avant et après la ligne Object objref = initial.lookup("java:comp/env/ejb/MyLoan");).

9. Mesurer, pour les session et les entity beans seulement, le temps requis pour l'invocation des méthodes de l'interface EJBObject communément appelées business methods dans le jargon des EJB (dans l'exemple de la figure 10 cela peut se faire en adaptant la snippet 2 aux méthodes monthlyPayment(), annualAmortTable(), monthlyAmortTable()).

[75] Servlet, EJB ou tout autre classe Java.
[76] Selon qu'il s'agisse d'un Topic ou d'un Queue.
[77] Source: Gail Anderson, Paul Anderson, Enterprise JavaBeans Component Architecture: Designing and Coding Enterprise Applications, Prentice Hall PTR, 0-13-035571-2, pages 456
[78] Nous rappelons à ce sujet qu'avant de pouvoir invoquer les méthodes de l'EJB le client doit trouver son emplacement sur le réseau. Cette responsabilité est confiée à l'API JNDI via un mécanisme de look up. Pour plus de détails voir snippet 2.

10. Mesurer le temps requis pour établir la connexion avec la base de données (en associant par exemple un compteur à la méthode getConnection() du DriverManager). Bien sûr cette mesure doit être effectuée pour toutes les connexions placées dans le connection pool.

11. Mesurer le temps consommé par l'exécution d'une requête, l'insertion, la modification ou la suppression d'un enregistrement (par exemple en associant un compteur respectivement aux méthodes executeQuery() et executeUpdate() de la classe Statement ou PreparedStatement).

12. Mesurer le temps nécessaire pour l'exécution d'une procédure stockée (en associant par exemple des compteurs aux méthodes execute() ou execcuteUpdate() de la classe CallableStatement).

13. Mesurer le temps nécessaire pour l'exécution ou l'annulation d'une transaction[79] (par exemple en associant des compteurs respectivement aux méthodes commit() et rollback() de l'objet Connection).

Jusqu'ici nous avons proposé une liste non limitative de mesures pour segmenter le temps de réponse total d'une application utilisant les servlets les EJB, et l'API JDBC. Dans une application plus complète qui intègre d'autres composants clés de J2EE, cette liste peut être étendue aux mesures suivantes :

- Le temps requis pour trouver un objet sur un annuaire[80] via une requête JNDI.
- Le temps requis pour invoquer un objet[81] distant par l'intermédiaire de RMI.
- Le temps passé par un message asynchrone dans un Topic ou un Queue JMS (Java Messaging Service) avant d'être délivré à l'objet destinataire.
- Le temps requis pour délivrer un courriel via JavaMail.
- Le temps requis pour l'authentification et l'autorisation d'un utilisateur par le biais du service JAAS (Java Authentification and Authorization service).
- Le temps requis pour invoquer un service web via JAX-RPC.
- Le temps requis pour l'exécution (commit) ou l'annulation (rollback) d'une transaction via JTS (Java Transaction Service).

[79] Transaction de base de données et non une transaction Web.
[80] LDAP par exemple
[81] Objet placé sur une aute JVM

La fréquence des mesures doit refléter la charge de travail que l'application sera amenée à traiter dans l'environnement de production. Les moyennes arithmétiques[82] des résultats obtenus doivent être compilées dans un tableau de bord de façon à ce que les développeurs puissent repérer rapidement les fragments de code qui sont responsables des impasses et des goulots d'étranglement. Bien évidemment toutes les actions qu'ils mènent pour corriger les anomalies relevées doivent se solder par des temps de réponse inférieurs ou égaux aux benchmarks présentés dans la section 2.1.

Nous proposons ci-après un modèle simplifié de tableau de bord pour les composants qui rentrent habituellement dans la construction d'une application de commerce électronique. Dans un projet de développement réel, ce modèle doit être ajusté en tenant compte du workflow, de l'architecture et des autres composants de l'application.

Tableau 8: Modèle de tableau de bord simplifié pour la segmentation du temps de réponse

Références de l'application	Cette case sert à indiquer le nom et les autres références permettant d'identifier l'application et les fonctionnalités mesurées.	
Cas d'utilisation	Cette case sert à indiquer le cas d'utilisation pour lequel on souhaite effectuer la mesure	
Temps total nécessaire pour la réalisation du cas d'utilisation	**Benchmark**	**Mesure réelle**
	Cette case indique le benchmark associé au cas d'utilisation en question (Cf. section 2.1)	Cette case indique le résultat final de la mesure effectuée pour le cas d'utilisation en question.
Interprétation des écarts	Cette case interprète les écarts relevés entre le benchmark et la mesure réelle	

Détail des mesures effectuées

Composants / Jointures/ API	**Mesures possibles**	**Interfaces associées aux compteurs**	**Résultat en ms**	**Cumul des résultats**	**Commentaires**
Jointure 1	Temps nécessaire pour acheminer la requête du client vers le conteneur	Fichiers log	-	-	-
JSP	Temps nécessaire pour la création et l'initialisation de la JSP.	_jspInit(),	-	-	-
	Temps nécessaire pour la génération de la JSP.	jspService()	-	-	-
FilterChain	Temps passé par la requête/ réponse http dans le FilterChain avant d'être envoyée à la servlet/client	doFilter()	-	-	-

[82] Somme des temps de réponse pour un composant sur le nombre de mesures prises pour le composant en question.

Suite du tableau de bord

Détail des mesures effectuées					
Composants / Jointures/ API	**Mesures possibles**	**Interfaces associées aux compteurs**	**Résultat en ms**	**Cumul des résultats**	**Commentaires**
Servlet	Temps nécessaire pour la création de la servlet	init()	-	-	-
	Temps nécessaire pour acheminer la requête/réponse http vers la servlet/client	service()	-	-	-
Jointure 2	Temps requis pour établir la communication entre le conteneur servlet et le conteneur EJB	Fichiers log	-	-	-
EJB	Temps requis pour créer, activer ou passiver un session ou entity bean	ejbActivate()[83], ejbPassivate()[84], ejbCreate()	-	-	-
	Temps requis pour envoyer un message asynchrone à un message driven bean	onMessage(), publish(), send()	-	-	-
	Temps requis pour trouver l'emplacement d'un EJB via JNDI	lookUp()	-	-	-
	Temps requis pour appeler une business method	Méthodes de EJBObject	-	-	-
JDBC	Temps requis pour établir une connexion avec la base de données	getConnection()	-	-	-
	Temps requis pour exécuter une requête simple ou pré-compilée	execute() ou executeUpdate() classes Statement ou PreparedStatement	-	-	-
	Temps requis pour exécuter une procédure stockée	execute() ou executeUpdate() classe CallableStatement	-	-	-
	Temps requis pour exécuter ou annuler une transaction	commit() et rollback()	-	-	-

La technique de mesure présentée dans cette section est facile à mettre en oeuvre lorsque les applications sont de petite taille et que leurs composants se trouvent sous la même JVM. Dans les applications de grande taille qui ont des composants intégrés à une architecture distribuée cette méthode présente les limites suivantes:

[83] Entity et stateful session beans seulement.
[84] Idem.

- Nécessité de recompiler, après l'insertion des compteurs, le code source de tous les composants dont on souhaite mesurer la performance. Or cela est difficile voir impossible à mettre en œuvre lorsque les composants sont pléthoriques ou lorsque le développeur n'a pas accès à leur code source[85]. Pour contourner ce problème, certains auteurs[86] proposent d'envelopper (wrapping) les composants dont on souhaite mesurer la performance à l'intérieur d'un nombre restreint d'objets proxy[87]. Cette technique permet, certes, de réduire le nombre de compteurs en les centralisant dans les objets proxy mais elle entraîne une consommation supplémentaire des ressources qui peut biaiser les résultats finaux.
- Les horloges des machines sur lesquelles les composants sont placés doivent être synchronisées.
- La technique ne peut pas être employée pour surveiller les performances de l'application en phase de production (parce que les messages affichés par les compteurs peuvent gêner/dérouter l'utilisateur).

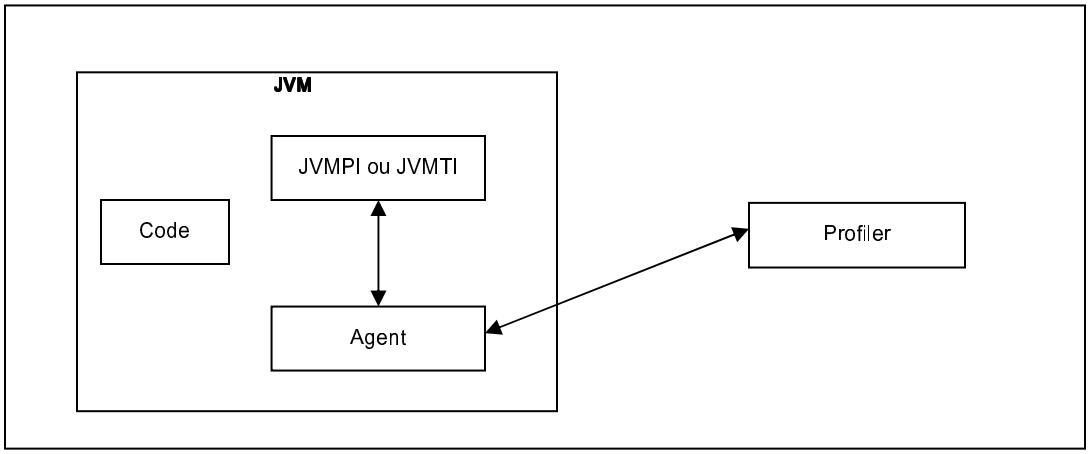
2.2.1.2 Utilisation d'un profiler

Les profilers sont des outils qui mesurent la performance des applications et assistent les développeurs dans l'identification des fragments de code pouvant la dégrader. Les profilers existant actuellement sur le marché relèvent d'une ou plusieurs des catégories suivantes :

La première catégorie fait appel aux fonctionnalités de monitoring déjà intégrées à la JVM. En fait, le développeur n'a pas besoin d'un profiler pour accéder aux dites fonctionnalités. Dépendamment de la version utilisée, il peut les manipuler directement par le biais des interfaces JVMPI[88] ou JVMTI[89]. L'apport des profilers de cette première catégorie se situe essentiellement au niveau de l'utilisabilité des interfaces (GUI) et de la lisibilité des résultats.

[85] Par exemple lorsque l'application utilise des composants commerciaux à source fermée (par opposition aux composants open source).
[86] A l'adresse : http://www.onjava.com/pub/a/onjava/2001/12/05/optimization.html, Jack Shirazi auteur du livre Java Performance Tuning propose une technique pour envelopper les principales classes de JDBC.
[87] Objets mandataires qui reçoivent les requêtes destinées à d'autres objets avant de la leur déléguer. Cette alternative permet de réduire le nombre de compteurs.
[88] Pour la JVM 1.4 et les versions antérieures
[89] Introduite dans la version 1.5 elle est plus robuste et consomme moins de ressources que JVMPI.

Figure 12: Interactions entre le Profiler et la JVM



Dans la figure 12, le rôle de l'agent est de gérer la communication entre la JVM et le Profiler notamment lorsque ce dernier est installé sur une machine distante (Rappelons que l'installation du Profiler sur une machine différente de celle de l'application permet de réduire la consommation des ressources (mémoire, CPU…) engendrée par le processus de mesure. Ceci étant un profiler ne doit pas consommer plus de 5% des ressources de la machine testée[90]).

La deuxième catégorie de profilers utilise des techniques de mesure qui ressemblent à bien des égards à celles présentées dans la section 2.2.1.2. Les mécanismes de fonctionnement sont néanmoins différents parce que les compteurs de cette deuxième catégorie sont :

- Générés automatiquement.
- Injectés directement dans le byte code de l'application.
- Basés sur des algorithmes plus élaborés.
- Optimisés pour réduire la consommation des ressources au strict minimum.
- Supprimés automatiquement du byte code lorsque les mesures sont terminées.

Comparée aux profilers qui utilisent les interfaces de la JVM, cette deuxième catégorie consomme moins de ressources parce que les messages et les événements générés massivement par la JVM sont désactivés.

[90] Source: Ali Syed and Jamiel Sheikh, Java Doctor, Manning Publications , Spring 2005, http://www.theserverside.com/articles/article.tss?l=JavaDoctorBookInReview

Notons par ailleurs que les profilers de la première catégorie ne fournissent que des mesures générales relatives à la consommation des ressources[91] alors que la deuxième catégorie permet d'associer des mesures beaucoup plus ciblées aux différentes classes et interfaces de l'application.

Les fonctionnalités de la **troisième catégorie** de profilers sont intégrées au serveur d'application. Elles sont généralement accessibles via des API qui se basent de plus en plus sur les Java Management Extensions (JMX). (Le tableau 9 fournit une liste non exhaustive de produits utilisant JMX).

JMX sera présenté rapidement dans la suite de cette section. Cette présentation sera limitée à son architecture globale et aux possibilités offertes en matière de surveillance des performances.

Tableau 9 : Liste non exhaustive de produits utilisant JMX[92]

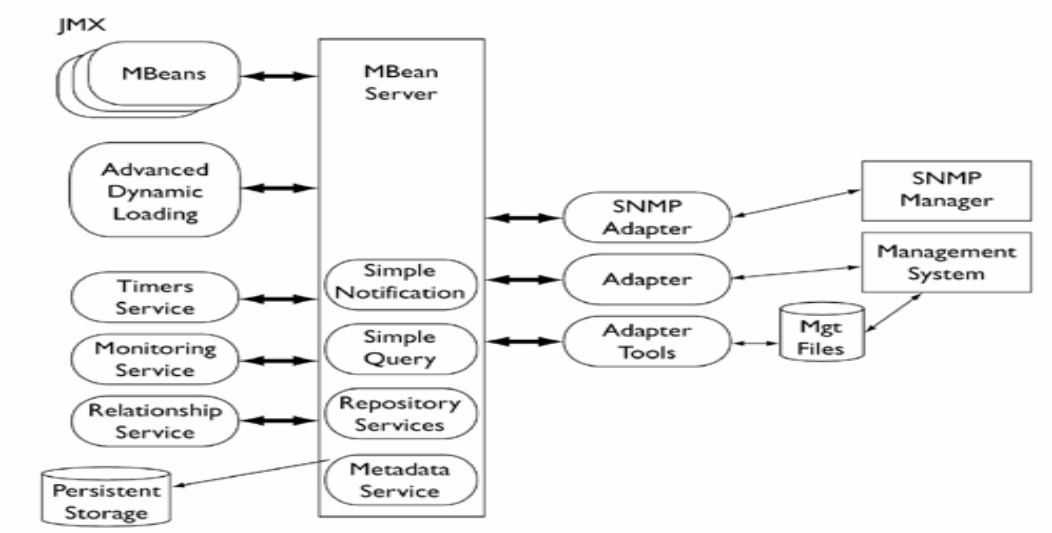
Product	Company	Description
WebLogic Application Server	BEA Systems	JMX is used to monitor J2EE services running in the server.
JBoss Application Server	JBoss	JMX is used for the application server architecture and to monitor services running in the server.
Bluestone Application Server	Hewlett-Packard	JMX is used to configure the application server.
OpenView	Hewlett-Packard	The OpenView monitoring suite can interface to JMX MBeans.
Adventnet Manager	Adventnet	This is a JMX-based monitoring solution.
JDMK (Java Dynamic Management Kit)	Sun Microsystems	This development kit is used to build JMX products.
Tivoli JMX	IBM	Tivoli is IBM's reference Implementation of JMX.
JOnAS Application Server	Bull	JMX is used to monitor J2EE services running in the server.

JMX est un framework qui permet d'administrer et de surveiller à distance des applications Java, mais également les composantes matérielles d'un réseau (PC, serveur, routeur,...). Comme le montre la figure 14, les MBeans (managed resources and management beans) se trouvent au cœur de ce framework. Ils correspondent concrètement à des Java Beans qui permettent à des objets écrits en Java[93] de disposer d'une interface contenant les informations et les leviers de contrôle nécessaires à leur surveillance et à leur administration. Les autres composantes clés du framework sont :

[91] Occupation de la mémoire, nombre de threads actifs, pourcentage du CPU consommé…
[92] Source : Benjamin G. Sullins and Mark B. Whipple , JMX in Action, Manning, ISBN 1930110561, October 2002
[93] Ces objets peuvent être aussi écrits dans un autre langage mais ils doivent être dotés d'une interface ad hoc qui permet leur interopérabilité avec les plateformes Java.

- Les agents qui fournissent des services[94] pour gérer les MBeans, mais également des adaptateurs de protocoles permettant d'y accéder localement ou à distance (par exemple depuis un browser HTML) .
- Un serveur MbeanServer qui effectue l'enregistrement des MBeans, leur référencement et l'agrégation de leurs interfaces et de leurs meta-données. Cela les rend visibles et exploitables sur le réseau soit par la logique spécifique des agents, soit par les différents connecteurs[95] et services qui leur sont associés.

Figure 13: Architecture des Java Management Extensions



Grâce au package javax.management.monitor, un MBean est capable de surveiller le temps consommé par tous les composants d'une application de commerce électronique (JSP, servlet, EJB). Notons par ailleurs que la traçabilité offerte par JMX est meilleure que celle de la deuxième catégorie de profilers parce que ses fonctionnalités de monitoring s'étendent non seulement aux composants de l'application mais également aux composantes matérielles du réseau. Un autre avantage de JMX a trait à la consommation des ressources qui est moindre comparativement à celle des profilers de la catégorie 2. Cela en fait une alternative efficace pour surveiller les performances d'une application en phase de production.

[94] Monitoring, timing, relation et class-loading
[95] En soi un agent ne dispose pas de capacités de communication. Il doit transiter par un connecteur pour obtenir des capacités de communication via SNMP, http ou tout autre protocole

A moins que le profiler ne combine toutes leurs fonctionnalités, la sélection de l'une ou l'autre des catégories précitées devra se faire en fonction :

- **De la taille de l'application et de son degré de distribution:** comme nous l'avons vu, la technique du custom instrumentation est difficile voir impossible à mettre en œuvre dans les grandes applications qui ont des composants répartis sur plusieurs JVM.
- **De la phase du projet :** lorsqu'on souhaite effectuer les mesures en phase de production il faut éviter d'utiliser le custom instrumentation et les profilers de la catégorie 1 et 2 parce qu'ils consomment beaucoup de ressources et génèrent des messages gênants pour l'utilisateur. Comme dit par ailleurs c'est les profilers de la catégorie 3 qu'il faut privilégier dans cette phase. Pour leur part, les profilers de la catégorie 1 et 2 sont typiquement utilisés dans les phases de codage et de test. Ils disposent pour la plupart de plugs-in qui permettent de les contrôler à partir des environnements de développement intégrés (IDE).
- **De l'objectif de la mesure:** les profilers de la catégorie 1 fournissent des métriques relatives à la consommation des ressources(CPU, mémoire, threads…) alors que les autres catégories de profilers permettent de surveiller de plus près le comportement de n'importe quel composant de l'application.
- **De l'organisation de l'entreprise :** ainsi lorsque le monitoring des performances est confié à plusieurs intervenants, il faut opter pour des profilers qui ont des fonctionnalités de travail collaboratif[96] (exemple : permettre à des consultants externes de suivre à distance le temps de réponse de l'application. Avertir le Département d'Assurance Qualité par courriel en cas de dégradation des performances. Assigner des tâches aux programmeurs. Editer des rapports d'activité…).

NB : une liste exhaustive des principaux profilers est disponible à l'adresse : http://www.JavaPerformanceTuning.com/resources.shtml

[96] La plupart des profilers de la catégorie 2 et 3 offrent ce genre de fonctionnalités.

2.2.2 Mesure de la charge de travail en fonction du comportement des visiteurs

Avant de commencer le design et le codage de l'application, les équipes de développement doivent évaluer avec le plus d'exactitude possible la charge de travail qu'elle sera amenée à traiter en phase de production. Cela ne peut se faire que si elles disposent d'un modèle de comportement qui reflète l'essentiel des interactions entre les visiteurs potentiels et l'application. Ce modèle doit être élaboré dès la phase d'analyse parce qu'il a un impact direct sur l'implémentation de l'application. Il nécessite une étroite collaboration entre l'équipe de développement et le Département Marketing qui doit mettre à sa disposition toutes les données commerciales, psychologiques et socio-économiques qui peuvent aider à mieux comprendre les interactions des visiteurs potentiels avec l'application.

Les recherches que nous avons effectuées nous ont permis de recenser un nombre restreint de références[97] qui proposent des méthodologies pour modéliser le comportement des visiteurs sur un site de commerce électronique. Ces références se basent pour la plupart sur une analyse rétrospective des fichiers de log au niveau du serveur web et/ou au niveau du serveur d'application. Cependant, elles ne tiennent pas, compte :

- Des interactions qui ont lieu sans recours aux ressources d'un serveur (par exemple les validations de données qui font uniquement appel à des scripts exécutés du coté du client ou les pages mises en cache dans un browser ou un proxy).

- Des interactions générées par les robots[98] dont le comportement doit faire l'objet d'un modèle à part (en effet pendant une session donnée ceux-ci génèrent beaucoup plus de requêtes que les utilisateurs humains)[99].

---

[97] D. A. Menascé and V. A. F. Almeida, "Challenges in Scaling E-Business Sites," Proc. 2000 Computer Measurement Group Conference, Orlando, FL, December 10-15, 2000.

D. A. Menascé and V. A. F. Almeida, Scaling for E-Business: technologies, models, performance, and capacity planning, Prentice Hall, 2000.
D. A. Menascé, V. Almeida, R. Fonseca, and M. A. Mendes, "A Methodology for Workload Characterization of E-commerce Sites," Proc. First ACM Conference on Electronic Commerce, Denver, CO, November 3-5, 1999.

[98] Crawlers, auction proxies…
[99] L'étude suivante propose une méthodologie pour analyser les comportements des robots sur les sites d'affaires électroniques. Virgilio Almeida, Daniel Menasce, Rudolf Riedi, Rodrigo Fonseca, Wagner Meira Jr., Flavia Peligrinelli, Analyzing Robot Behavior in E-Business Sites.

Pour les besoins du présent essai, nous avons choisi de décrire brièvement les résultats des travaux menés par A. Menascé et V. A. F. Almeida[100] dans une étude intitulée 'A Methodology for Workload Characterization of E-commerce Sites'. Cette étude qui fait autorité dans la littérature propose un modèle de comportement et des algorithmes pour évaluer la charge de travail qui en découle. A. Menascé et V. A. F. Almeida développent leur modèle de comportement en partant de l'exemple de deux catégories de visiteurs à savoir :

1. Les visiteurs qui utilisent le site pour rechercher des informations sur les produits/services de l'entreprise mais qui finissent très souvent par ne rien acheter.
2. Des clients qui en plus d'utiliser le site pour s'informer, finissent toujours par acheter lorsqu'ils trouvent des produits/services qui correspondent à leurs besoins.

Le comportement de ces deux catégories est représenté à l'aide d'un diagramme sur lequel on retrouve les fonctionnalités de base d'un site de commerce électronique ainsi que les transitions possibles entre ces différentes fonctionnalités. Les chiffres annotés sur les flèches, marquant la transition, correspondent à la probabilité pour qu'un visiteur passe d'une fonctionnalité à une autre.

Comme on peut le voir sur les deux graphiques la probabilité pour qu'un visiteur utilise la fonctionnalité 'achat' est égale à un 1 pour les visiteurs de la première catégorie alors qu'elle est pratiquement nulle pour les visiteurs relevant de la deuxième catégorie. La transition d'une fonctionnalité à une autre comporte un temps de réflexion (think time) durant lequel le visiteur n'a aucune interaction avec l'application. Pendant ce temps les ressources oisives du serveur peuvent être allouées au traitement d'une nouvelle requête ou d'une opération restée en suspens.

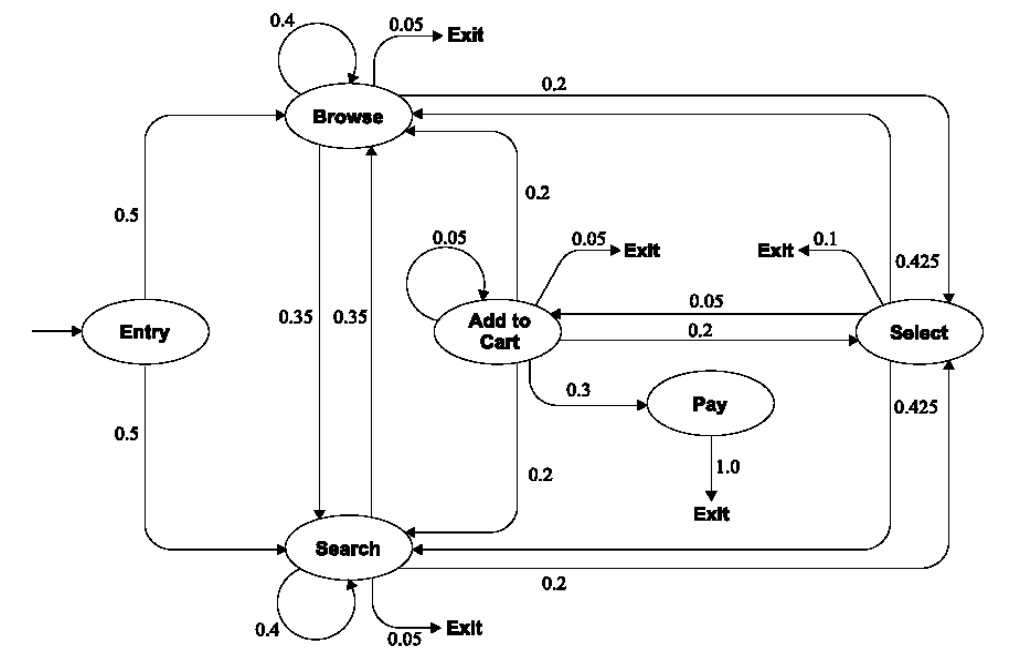
Mathématiquement ce modèle de comportement est représenté par un système (P,Z) dans lequel $P=[p_{i,j}]$ est une matrice n X n contenant les probabilités de transition entre les n fonctionnalités du site. Z pour sa part est une matrice qui contient les temps de réflexion moyens avant chaque transition.

[100] Source: D. A. Menascé, V. Almeida, R. Fonseca, and M. A. Mendes, "A Methodology for Workload Characterization of E-commerce Sites," Proc. First ACM Conference on Electronic Commerce, Denver, CO, November 3-5, 1999.

Pour connaître le nombre moyen de fois qu'une fonctionnalité sera utilisée par un visiteur (V_j), les auteurs suggèrent la résolution du système d'équations linéaires suivant :

$$V_1 = 1$$
$$V_j = \sum_{k=1}^{n} V_k \times p_{k,j} \qquad \text{for all } j = 2, \cdots, n$$

Figure 14: Modèle de comportement d'un visiteur relevant de la catégorie 1[101]



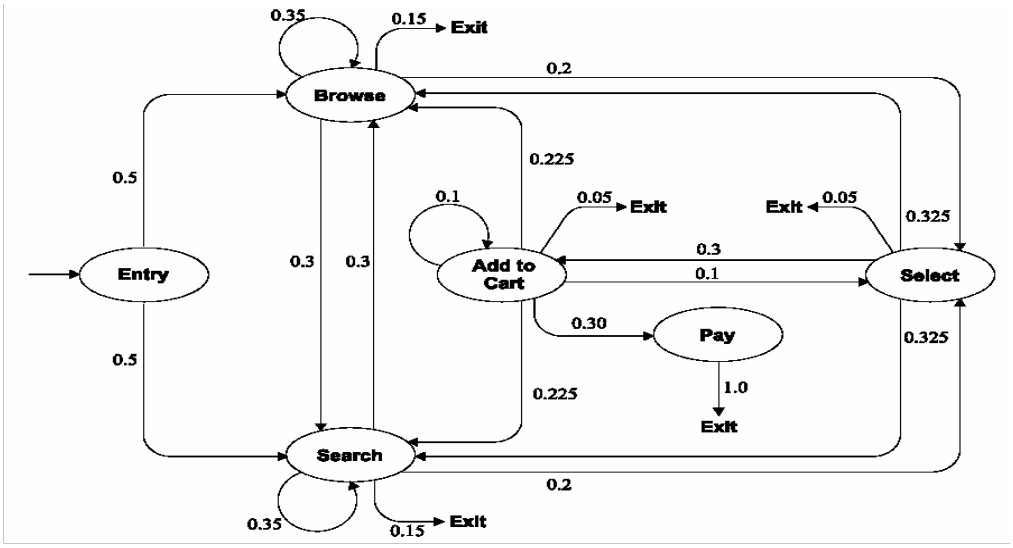
Les figures 14 et 15 ne reflètent pas l'ensemble des transitions possibles sur un site de commerce électronique mais leur adaptation peut s'avérer d'une grande utilité lorsqu'on veut prévoir l'impact du comportement des visiteurs sur la performance de l'application.

2.2.3 Mesure du débit et des coûts y afférents (benchmark TPC-W)

Des mesures traditionnelles comme le nombre de bits, de requêtes ou de transactions par seconde ne suffisent pas pour évaluer le débit d'une application de commerce électronique et les coûts qui lui sont associés.

[101] Source: D. A. Menascé, V. Almeida, R. Fonseca, and M. A. Mendes, "A Methodology for Workload Characterization of E-commerce Sites," *Proc. First ACM Conference on Electronic Commerce*, Denver, CO, November 3-5, 1999.

Figure 15: Modèle de comportement d'un visiteur relevant de la catégorie 2



A l'instar de la charge de travail, ces mesures doivent être caractérisées en tenant compte du comportement des visiteurs parce que des modèles d'interactions[102] différents engendrent des débits différents et n'affectent pas les performances de la même manière[103]. Les études qui traitent de cette caractérisation sont légions mais les méthodologies qu'elles utilisent ne sont pas standardisées et ne tiennent pas toujours compte des spécificités des applications de commerce électronique.

Dans une tentative de standardiser lesdites méthodologies et de les adapter aux particularités des applications de commerce électronique, le Transaction Processing Performance Council a crée le benchmark transactionnel TPC-W. Ce benchmark repose sur une spécification qui définit:

- Les fonctionnalités que doit offrir l'application.
- Sa taille.
- Ses interfaces.
- Les temps de réponse tolérables pour chaque fonctionnalité.
- Le Modèle Relationnel des Données (MRD).
- Les modèles de comportement à suivre pour interagir avec le site.

[102] Modèle d'interactions avec le site appelé aussi modèle de comportent dans le reste de ce document.
[103] Par exemple sur les sites de commerce électronique la demande pour les pages générées automatiquement est plus grande que sur les autres sites. Cela réduit les possibilités de mise en cache et par conséquent augmente la charge de travail que le serveur doit traiter.

La spécification ne précise ni l'architecture du site ni les technologies qui doivent être utilisées pour le construire. C'est aux chercheurs, aux éditeurs de logiciels et aux constructeurs qu'il appartient de choisir les technologies et l'architecture qui permettent d'atteindre le meilleur niveau de performance aussi bien en termes de débit qu'en termes de coût.

La charge de travail de TPC-W est générée dans un environnement contrôlé où des émulateurs simulent plusieurs modèles de comportement sur un site de commerce électronique. Cela permet de tester les éléments suivants:

- Gestion des sessions multiples.
- Mise en cache. Génération dynamique des pages à partir d'une base de données.
- Gestion des transactions. Respect de l'intégrité de la base de données (ACID).
- Connexions SSL.

En tant que benchmark[104], TPC-W permet de comparer la performance et le ratio prix/performance des systèmes de commerce électronique (Systeme Under Test) composés d'un serveur machine, d'un serveur d'application, d'un serveur web, d'un gestionnaire de base de données et d'une application qui respecte les clauses de la spécification[105]. La performance est exprimée en WIPS@<facteur _scalabilité> (Web Interactions Per Second) et en $/WIPS@<facteur _scalabilité>. Pour bien comprendre en quoi consistent ces deux métriques, il est nécessaire de revenir avec plus de détails sur le contenu de la spécification.

L'application de commerce électronique telle que définie par la spécification est un site de vente de livres doté des fonctionnalités suivantes : recherche de produits, affichage des informations relatives aux produits, registration, panier d'achat, affichage des produits vedettes, administration des produits. Ces fonctionnalités sont réparties sur 14 pages dont 1 page statique (Customer Registration), 4 pages qui peuvent être mises en cache (New Product, Best Seller, Product Details, Search Results) et 4 pages accessibles via SSL (Buy request, buy confirmation, order inquiry, order display).

[104] L'acronyme TPC-W désigne à la fois le benchmark, la spécification et l'application.
[105] Certains auteurs affirment que TCP-W sert uniquement à comparer les performances des serveurs d'application mais la spécification indique clairement que le système testé (SUT ou Systeme Under Test) est composé d'un serveur machine, d'un serveur d'application, d'un serveur web, d'une base de données et d'une application. Le benchmark sert donc à tester la performance globale de tout le système et non la performance individuelle de ses différents composants.

La spécification définit pour chaque page le nombre d'images incluses, leur taille (qui varie de 5KB à 250KB) et le temps de réponse maximum (Cf. tableau 10)[106].

Tableau 10 : Pages du site TPC-W [107]

Name	Dynamic Generation	Number of Joints	Number of Images	Max Response Time (s)	Interaction Type
Admin Confirm	Yes	4	5	20	Order
Admin Request	Yes	2	6	3	Order
Best Seller	Yes	3	9	5	Browse
Buy Confirm	Yes	1	2	5	Order
Buy Request	Yes	1	3	3	Order
Customer Registration	No	N/A	4	3	Order
Home	Yes	1	9	3	Browse
New Product	Yes	2	9	5	Browse
Order Display	Yes	1	2	3	Order
Order Inquiry	No	N/A	3	3	Order
Product Detail	Yes	2	6	3	Browse
Search Request	No	N/A	9	3	Browse
Search Result	Yes	2	9	10	Browse
Shopping Cart	Yes	1	9	3	Order

Toutes les données de l'application sont conservées dans une base de données[108] construite conformément au modèle relationnel de la figure 16. Un facteur de scalabilité (appelé aussi facteur d'échelle) définit la taille du magasin virtuel. Il correspond au nombre de livres stockés dans la table d'inventaire ITEM. La spécification exige à ce que le cardinal de cette table soit choisi dans l'ensemble suivant : { 1,000; 10,000; 100,000; 1,000,000; 10,000,000}. Lors des tests de scalabilité[109], la cardinalité des tables doit varier comme indiqué dans le tableau 11.

La navigation à travers les différentes pages du site se fait selon les modèles de comportement suivants :

- **Shopping Mix :** il simule le comportement d'achat selon une distribution de Pareto. Dans ce modèle 80% de la charge de travail est générée par la recherche d'information (browsing). Les 20% restants sont générés par des achats.

[106] Si le système testé (SUT ou System Under Test) n'arrive pas à répondre aux requêtes qu'il reçoit au bout desdits temps de réponse, la charge de travail doit être réduite jusqu'à ce qu'il y arrive.
[107] Source: Mehdi Khouja, Farouk Kamoun, Experimenting With the TPC-W E-commerce Benchmark.
[108] Selon la spécification l'utilisation d'une base de données relationnelle n'est pas obligatoire.
[109] La spécification distingue entre deux types de scalabilité. La première qui est fixe affecte la taille du magasin virtuel (augmentation du nombre de livres dans la table ITEM). La deuxième qui est variable résulte de l'augmentation du nombre d'émulateurs (augmentation du nombre d'enregistrements dans les autres tables).

Tableau 11: Conditions de scalabilité de TPC-W

Table Name	Cardinality (in rows)	Typical Row Length (in bytes)	Typical Table Size (in bytes)
CUSTOMER	2880 * (number of EB)	760	2,188,888 k
COUNTRY	92	70	6.44 k
ADDRESS	2 * CUSTOMER	154	887,040 k
ORDERS	.9 * CUSTOMER	220	570,240 k
ORDER_LINE	3 * ORDERS	132	1,026,432 k
AUTHOR	.25 * ITEM	630	1,575 k
CC_XACTS	1 * ORDERS	80	207,360 k
ITEM	1k, 10k, 100k, 1M, 10M	860	8,600 k

- **Browsing mix:** il simule le comportement des visiteurs qui utilisent le site pour s'informer sur les produits et qui finissent souvent par ne rien acheter. Dans ce modèle 95% de la charge de travail est générée par la recherche d'informations. Les 5% restants sont générés par les achats.
- **Ordering mix :** dans ce modèle la charge générée par la recherche d'informations est égale à celle générée par les achats.

Le pourcentage des accès canalisés vers chacune des pages du site est donné par le tableau 12.

Comme le montre la figure 17, la charge de travail est générée artificiellement par des émulateurs qui imitent les 3 comportements décrits ci-dessus. Chaque émulateur établit une connexion indépendante avec le système testé via les protocoles http (couche d'application) et TCP/IP (couche de transport et couche Internet). Les activités des émulateurs sont interrompues entre deux requêtes consécutives pour simuler le temps de réflexion (think time). Ces interruptions se font selon une distribution exponentielle négative[110].

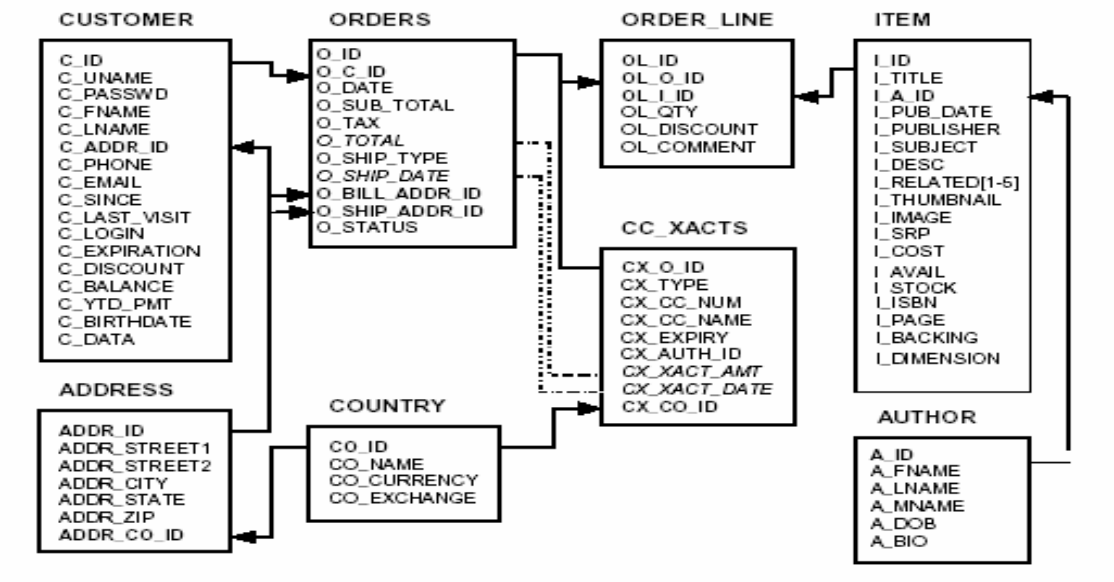
T= -ln(r) X U

r est un nombre aléatoire appartenant à l'intervalle]0, 1]

U est un délai compris entre 7 et 8 secondes bornes incluses.

[110] Source: Transaction Processing Performance Council (TPC), TPC-W BENCHMARK, Version 1.7, Oct 11, 2001

Figure 16: Modèle relationnel de données de TPC-W[111]



NB : Pour connaître les types et la description des champs, consulter la version officielle de la spécification TPC - W à l'adresse : http://www.tpc.org/tpcw/spec/tpcw_V1.8.pdf

Les requêtes envoyées par les émulateurs au site sont rattachées à des sessions dont la durée est également déterminée selon une distribution exponentielle négative. Les sessions commencent obligatoirement à la page d'accueil. Elles sont entamées selon 2 scénarios :

- Scénario 1 : les requêtes proviennent d'un utilisateur qui ne possède pas de compte (ces requêtes sont générées 20% du temps).
- Scénario 2 : les requêtes proviennent d'un utilisateur qui est déjà enregistré. Dans ce cas une authentification est requise (ces requêtes sont générées 80% du temps).

La durée moyenne de chaque session est de 15 minutes. Lorsque cette durée s'écoule, l'émulateur ferme les connexions TCP/IP et SSL puis régénère une nouvelle session.

Le débit est calculé à partir du nombre d'interactions[112] réussies qui ont lieu chaque seconde entre les émulateurs et le SUT (System Under Test). Une interaction réussie doit obligatoirement respecter les limites de temps définies dans le tableau 10.

[111] Source: Transaction Processing Performance Council (TPC), TPC BENCHMARKTM W, Version 1.7, Oct 11, 2001
[112] Requêtes et réponses.

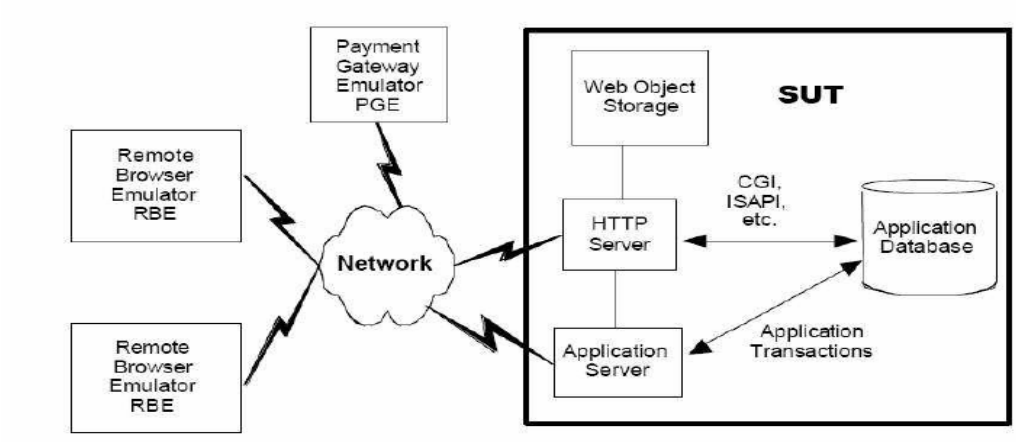
Comme l'exige la spécification le débit doit être caractérisé en tenant compte du modèle de comportement des visiteurs. Ainsi :

- WIPS désigne le débit pour le modèle de comportement shopping mix.
- WIPSo celui du modèle ordering mix.
- WIPSb celui du modèle browsing mix.

Tableau 12: Pourcentage des accès à chacune des pages de TPC-W[113]

Web Interaction	Browsing Mix(WIPSb)	Shopping Mix(WIPS)	Ordering Mix (WIPSo)
Browse	**95%**	**80%**	**50%**
Best Sellers	11.00%	5.00%	0.46%
Home	29.00%	16.00%	9.12%
New Products	11.00%	5.00%	0.46%
Product Detail	21.00%	17.00%	12.35%
Search Request	12.00%	20.00%	14.54%
Search Results	11.00%	17.00%	13.08%
Order	**5%**	**20%**	**50%**
Admin Confirm	0.09%	0.09%	0.11%
Admin Request	0.10%	0.10%	0.12%
Buy Confirm	0.69%	1.20%	10.18%
Buy Request	0.75%	2.60%	12.73%
Customer Registration	0.82%	3.00%	12.86%
Order Display	0.25%	0.66%	0.22%
Order Inquiry	0.30%	0.75%	0.25%
Shopping Cart	2.00%	11.60%	13.53%

Figure 17: Environnement TPC-W[114]



NB : Le PGE génère les mêmes requêtes qu'un système de paiement en ligne utilisant SSL version 3 ou TLS (RFC2246). Les RBE génèrent le reste des requêtes.

[113] Source: Transaction Processing Performance Council (TPC), TPC-W BENCHMARK, Version 1.7, Oct 11, 2001
[114] Source: Idem

Parce qu'on ne peut comparer que ce qui est comparable la spécification exige que les métriques énumérées ci-dessus soient couplées au facteur de scalabilité fixe qui comme nous l'avons vu reflète la taille du magasin virtuel[115]. Les métriques deviennent donc WIPS@<facteur _scalabilité>, WIPSo@<facteur _scalabilité>, et WIPSb@<facteur _scalabilité>.

Lorsqu'on augmente le nombre d'émulateurs pour générer plus de charge de travail la spécification exige que la métrique WIPS vérifie l'inégalité suivante :

(Nombre émulateurs)/14 < WIPS < (Nombre émulateurs)/7

Pour vérifier cette inégalité il faut respecter les conditions de scalabilité définies dans le tableau 11.

La spécification définit une autre métrique pour calculer le rapport prix/performance (prix/WIPS). Le prix en question comprend :

- Le prix des différentes composantes matérielles et logicielles du SUT (A noter que le prix des émulateurs n'est pas inclus).
- Le prix de stockage des données en ligne (Cela comprend le prix de stockage des données conservées dans la base initiale plus celui des données générées par les émulateurs sur une période de 180 jours à raison de 8 heures par jour. La clause 7.2.3.1 de la spécification exige par ailleurs à ce qu'il y ait une période de traitements ininterrompus d'au moins 14 jours).
- Le coût de tous les logiciels ayant servi à la production, à l'administration et à la maintenance de l'application (IDE, WYSIWYG…).

Cela dit, le prix total est calculé sur trois ans, au lieu des 5 ans habituels, probablement en raison du rajeunissement rapide des serveurs de commerce électronique comparativement aux autres serveurs.

Des grandes compagnies comme IBM, SUN ou Microsoft utilisent aujourd'hui TPC-W pour tester la performance de leurs produits[116] et vanter leurs mérites par rapport à la concurrence.

[115] Nombre de livres stockées dans la table ITEM choisi obligatoirement dans l'intervalle { 1,000; 10,000; 100,000; 1,000,000; 10,000,000}
[116] Serveur d'application, base de données, serveur http…

Ce benchmark leur permet, ainsi, de disposer d'un champ commun de bataille grâce auxquels les clients[117] peuvent relever les forces et les faiblesses de leurs produits.

Du côté des équipes de développement le benchmark TPC-W peut servir de template pour la mise en place d'un environnement de test permettant la mesure et la caractérisation du débit des applications de commerce électronique. A l'aide des outils de tests automatisés, l'activité générée par les émulateurs peut être adaptée à des applications qui comportent des fonctionnalités plus nombreuses et plus complexes. De plus les modèles de comportement et les conditions de scalabilité définis par la spécification peuvent s'avérer d'une grande utilité lorsqu'on souhaite évaluer les scénarios suivants :

- WIPSo : mettre beaucoup de pression sur la base de données en générant un nombre important de requêtes update et insert. Mesurer le temps de réponse supplémentaire engendré par les connexions SSL. Utiliser massivement les transactions tout en préservant l'intégrité de la base de données (propriétés ACID).
- WIPS et WIPSb : accéder à la base de données en lecture seule (read-only). Mise en cache des données et des images.

Comme dit par ailleurs, la spécification ne définit ni l'architecture de l'application de commerce électronique ni les technologies qui doivent être utilisées pour la construire[118]. Cela garantie sa portabilité et laisse aux entreprises le choix de l'implémenter à leur convenance. Puisque cet essai traite du cas particulier des applications de commerce électronique écrites en Java nous avons effectué des recherches pour répertorier les différentes implémentations Java de TPC-W. Les résultats de nos recherches sont compilés dans le tableau 11.

Comme tout autre benchmark TPC-W a des limites qui doivent être prises en considération au moment de la mesure du débit et des coûts y afférents :

[117] Les résultats des tests effectués par les compagnies qui sponsorisent le benchmark sont publiés à l'adresse : http://www.tpc.org/tpcw/results/tpcw_price_perf_results.asp . Les résultats des autres compagnies sont publiés sur leurs sites et/ou dans les réseaux de presse spécialisée.

[118] Les pseudo-codes inclus dans les annexes de la spécification ne sont donnés qu'à titre d'exemple.

- Les émulateurs génèrent une activité qui reflète uniquement les comportements des visiteurs dans un modèle d'affaire B2C (les modèles C2C et B2B ne sont pas pris en compte).
- TPC-W simule uniquement des comportements humains. Il fait abstraction de la charge de travail générée par les robots[119] qui ont un modèle de comportement spécifique.
- Le benchmark ne prévoit aucune interaction pour évaluer les performances des services web qui s'imposent de plus en plus dans le monde du commerce électronique.
- Les tests sont menés dans un environnement contrôlé qui ne reflète pas forcément les contraintes de connexion sur le web.

Tableau 13: Implémentations Java du benchmark TPC-W

Implémentation	Eléments de l'application	Serveur http/ Conteneur	Bases de données	Eléments manquants
Pharm , http://www.ece.wisc.edu/~pharm/tpcw.shtml	Servlet et JDBC	IBM WebSphere , Apache/JServ, Sun Java Web Server, et Jigsaw Web Servers	IBM DB2	PGE, SSL
Amit Manjhi http://www-2.cs.cmu.edu/~manjhi/tpcw.html	Servlet et JDBC	Tomcat	MySQL	PGE, SSL
Christian Plattner	Servlet et JDBC	Tomcat	Postgresql	PGE, SSL
Mehdi Khouja, Farouk Kamoun, Catalina M. Llad_o, Ramon Puigjaner cllado@uib.es, puxi@uib.es	JSP, Servlet et JDBC	Apache Tomcat	Oracle	-
Parallel and Distributed Systems Research Group (PDSG) http://www.cs.nyu.edu/~totok/professional/software/tpcw/tpcw.html	Servlet, EJB et JDBC	JBoss	MySQL	-

[119] Crawlers, shopping agents…

3. Gestion de la performance dans la phase d'analyse

Pendant la phase d'analyse les analystes d'affaires[120] ont tendance à polariser toute leur attention sur les spécifications fonctionnelles au détriment des spécifications non fonctionnelles comme les benchmarks et les métriques de performance. Leurs efforts sont concentrés uniquement sur la compréhension du domaine d'affaires, l'analyse des « besoins métier » et la formulation des cas d'utilisation qui en découlent. Parce que la performance est la conséquence directe des solutions retenues au niveau des phases de design et de codage, ils considèrent que sa gestion[121] devrait être l'apanage des architectes et des programmeurs. Cela les conduit bien souvent à ne pas spécifier les benchmarks et les métriques de performance dans les cahiers des charges. L'expérience de plusieurs projets de développement montre que cette démarche est à l'origine des inconséquences suivantes :

- La performance ne fait pas partie des principaux critères d'évaluation de la qualité de l'application. Elle est reléguée à un rang inférieur loin derrière les spécifications fonctionnelles.
- Le coût des ressources humaines, technologiques et financières allouées à la gestion des performances est faible pendant la phase de design et de codage mais il croit démesurément pendant les phases subséquentes comme le montre la figure 1.
- Malgré son importance, la performance ne fait pas partie des termes du contrat qui lie la maîtrise d'œuvre à la maîtrise d'ouvrage.
- Les équipes de test et de maintenance deviennent des boucs émissaires qui doivent endosser indûment les négligeances des architectes et des programmeurs et trouver à la halte des solutions palliatives pour livrer l'application dans les délais.

Il ressort de ces inconséquences que **les besoins en performance doivent être considérés comme des besoins d'affaires et non comme des réglages de dernière minute qui n'apparaissent pas sur la liste des priorités**. Leur intégration au processus de développement demande une contribution active et responsable de tous les acteurs qui participent au projet (managers, analystes, architectes, programmeurs, administrateurs de bases de données, administrateur du serveur web/application, équipe de maintenance…)

[120] Ces affirmations sont basées sur des constats personnels dans plusieurs grands projets de développement.
[121] Gestion des performances

Dans cette partie, nous proposons une démarche pour l'intégration du management des performances à la phase d'analyse. Cette démarche est centrée sur le modèle d'affaires duquel relève l'application de commerce électronique. Une fois mise en œuvre, elle permettra aux architectes et aux programmeurs de disposer d'un ensemble plus complet d'artefacts qui spécifient les besoins en performance au même titre que les besoins fonctionnels.

3.1 Besoins en performance et modèles d'affaires

3.1.1 Besoins en performance des sites B2C

A l'intérieur du modèle d'affaires B2C, nous ferons dorénavant la distinction entre deux types de sites :

- Les sites d'intermédiation financière.
- Les sites conventionnels de shopping (qui vendent au détail des produits/ services non financiers).

Les sites d'intermédiation financière sont visités à un rythme très intensif. Pendant les horaires d'ouverture des marchés les clients s'y connectent fréquemment pour suivre l'évolution des cours, effectuer des transactions, obtenir des nouvelles sectorielles et/ou télécharger des rapports d'activité. Le caractère volatile des données ainsi que la surveillance des organismes de réglementation exigent de ces sites un très grand taux de disponibilité et des temps de réponse hors du commun (Cf. section 2.1.1.2).

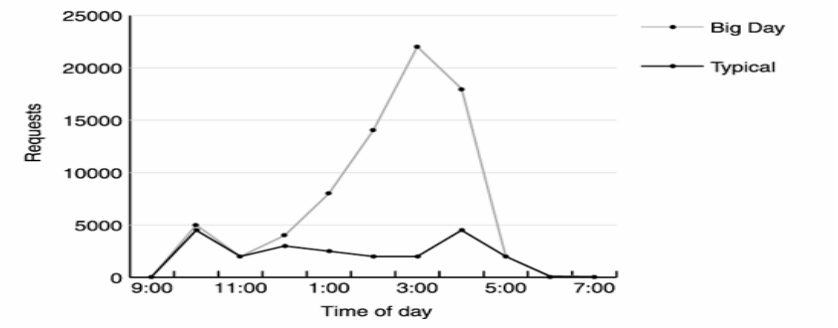
Les cas d'utilisation des sites d'intermédiation financière posent de véritables défis à l'équipe de développement parce que :

- Ils augmentent la demande pour les pages générées dynamiquement.
- Ils mettent beaucoup de pression sur la base de données.
- Ils font massivement appel aux services d'applications tierces (exemple : services web d'un système de compensation installé sur le main frame d'un tiers).

Les pages des sites d'intermédiation financière contiennent peu de graphismes[122]. Certains sont statiques (logo, image gif pour la navigation) mais la plupart sont générés à la volée sur la base des données personnalisées de chaque utilisateur (ce qui limite les possibilités de mise en cache).

[122] Ces affirmations sont basées sur les résultats des travaux de Stacy Joines, Ruth Willenborg, Ken Hygh publiés dans le livre Performance Analysis for Java™ Web Sites, Addison Wesley, September 10, 2002, ISBN: 0-201-84454-0

Figure 18: Pattern du trafic quotidien sur un site d'intermédiation financière[123]



Les cours des titres, pour leur part, doivent être rafraîchis en temps réel afin de refléter la dynamique de l'offre et de la demande sur les marchés. Ce rafraîchissement épuise non seulement la base de données mais également les applications tierces qui fournissent leurs services au site. Les bulletins d'informations et les rapports d'activités offrent, néanmoins, plus de possibilités de mise en cache parce que leurs données ne sont pas rapidement périssables (dépendamment de la périodicité des bulletins et des rapports celles-ci ne changent pas aussi fréquemment que les cours des titres).

Les besoins en sécurité affectent également les performances des sites d'intermédiation financière. Outre l'obligation légale de garder des traces sur toutes les transactions effectuées ceux-ci échangent des données confidentielles[124] via des protocoles sécurisés (SSL, SET…). Comme le montre les figures 19 et 20 le recours à ces protocoles augmente le temps de réponse et ralentit le débit notamment en raison des encryptages de données et des vérifications qui doivent être effectuées auprès des organismes de certification (exemple : vérification des données d'un certificat numérique auprès de Verisign[125]) .

Pour leur part, les sites de shopping présentent les mêmes caractéristiques que les sites d'intermédiation financière à la différence près que :

- Leurs pages retournent un nombre plus important d'images (sous forme de catalogues avec la possibilité d'agrandir ou de réduire la taille par effet de zoom).
- Les données qui servent à la génération des catalogues sont moins volatiles ce qui offre plus de possibilités de mise en cache.

[123] Source: Stacy Joines, Ruth Willenborg, Ken Hygh, Performance Analysis for Java™ Web Sites, Addison Wesley, September 10, 2002, ISBN: 0-201-84454-0, pages 464
[124] Numéro de carte de crédit. Numéro de compte. Identité des clients….
[125] Clé publique, date d'expiration…

- Le trafic, qui est plus prévisible, atteint son paroxysme pendant les périodes de fêtes et de liquidations (voir figure 21).
- L'activité des robots sur ces sites est beaucoup plus intensive.

Figure 19: Impact de l'utilisation du protocole TLS sur le débit d'une application d'e-commerce[126]

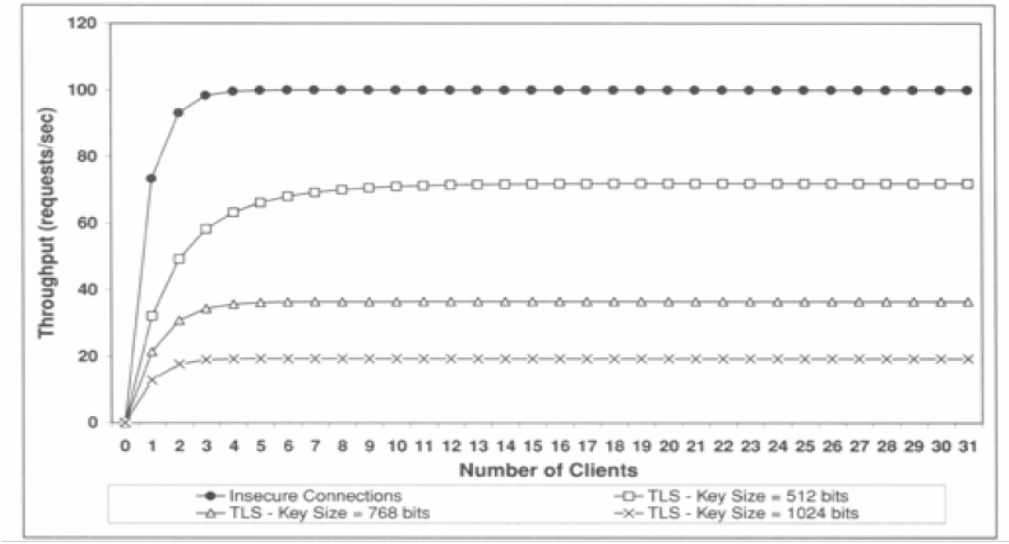
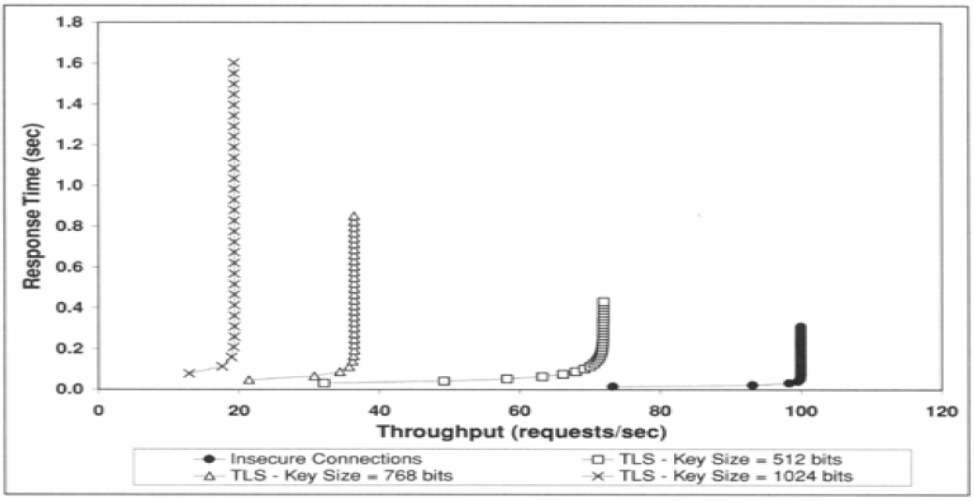


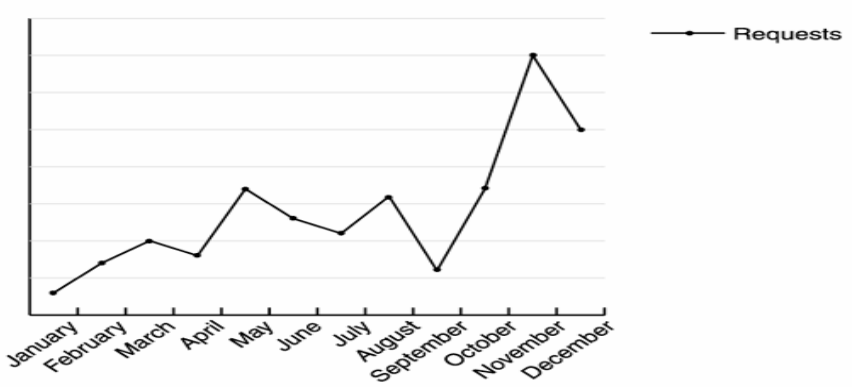
Figure 20: Variation du temps de réponse et du débit en fonction de la taille de la clé TLS[127]



Pour de plus amples informations sur les caractéristiques des sites B2C et leur impact sur les performances voir la section 2.2.3.

[126] Source: Daniel A. Menascé, Virgilio A. F. Almeida, Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning, Prentice Hall, 2000, ISBN 0-13-086328-9, pages 357
[127] Source: Idem

Figure 21: Pattern de trafic annuel sur un site de shopping[128]



3.1.2 Besoins en performance des sites B2B

Comparativement aux sites d'intermédiation financière, les sites B2B :

- Contiennent moins de graphismes et de publicité (leurs catalogues sont moins riches en images que ceux des sites d'intermédiation financière).
- Ont moins de visiteurs.
- Traitent un nombre moins important de transactions[129] (transactions de gros moins fréquentes que les transactions de détail effectuées sur les sites d'intermédiation financière).
- Echangent en masse de gros fichiers formatés en XML (en lieu et place des anciens protocoles spécialisés utilisés pour l'EDI[130]).
- Sont visités par des utilisateurs qui ont peu de ténacité face à la latence (voir section 2.1.1 pour plus de détails à ce sujet).

Même si ces sites ont un trafic sporadique, leurs cas d'utilisation déclenchent des traitements plus complexes et plus gourmands en ressources que les sites d'intermédiation financière[131]. A ce sujet, l'échange et le parsing de gros fichiers XML limitent les possibilités de mise en cache et réduisent la quantité de mémoire disponible pour l'application. Notons par ailleurs que la plupart de ces sites utilisent des serveurs de messages asynchrones[132] qui placent les données dans une file d'attente avant de les délivrer au destinataire final (exemple : IBM's WebSphere MQSeries).

[128] Source: Stacy Joines, Ruth Willenborg, Ken Hygh, Performance Analysis for Java™ Web Sites, Addison Wesley, September 10, 2002, ISBN: 0-201-84454-0, pages 464
[129] Transaction au sens économique, c'est à dire l'achat et la vente des biens et/ou services
[130] EDI FACT ou X.12
[131] Parsing de gros fihciers XML, connexion avec des applications tierces via des middletiers, traitements batch…
[132] Comme IBM's WebSphere MQSeries

3.1.3 Besoins en performance des sites C2C (online auctions)

Les sites de vente aux enchères de particulier à particulier ont des cas d'utilisation et des modèles de comportement qui affectent différemment les besoins en performance. Les spécificités de ces sites se résument comme suit :

- Une plus grande affluence parce qu'ils sont fréquentés aussi bien par des acheteurs que par des vendeurs.
- Le catalogue contient un plus grand nombre d'articles (ce qui limite les possibilités de mise en cache)
- Les produits doivent faire l'objet de plusieurs enchères avant d'être achetés (ce qui augmente considérablement le nombre de requêtes traitées).
- Une utilisation accrue des agents (bidding proxies). Une étude menée[133] par Daniel A. Menascé et Vasudeva Akula, montre à ce sujet que 55% des enchères sont placées par des agents dont l'activité est plus intensive que celle des utilisateurs humains.
- Les données relatives aux produits ne sont pas conservées aussi longtemps que dans les sites de shopping parce que chaque vente a une date de validité au delà de laquelle les enchères ne sont plus acceptées. Cela entraîne plusieurs mises à jour sur la table des produits.
- L'activité s'intensifie quand la période de validité de la vente aux enchères touche à sa fin. Une autre étude[134] menée par Daniel A. Menascé montre, en effet, que 30% des enchères sont faites à la fin du cycle de vie de chaque vente (fin qui représente 5% de la durée de validité de la vente).

Les besoins en performance des 3 catégories de sites que nous avons étudiées sont résumés dans le tableau 14.

[133] Daniel A. Menascé, Vasudeva Akula, Towards Workload Characterization of Auction Sites, In Proc. Sixth IEEE Workshop on Workload Characterization (WWC-6), Austin, TX, Oct. 27, 2003.
[134] Daniel A. Menascé, Vasudeva Akula, Improving the Performance of Online Auction Sites through Closing Time Rescheduling, September 27 - 30, 2004

Tableau 14: Besoins en performance en fonction du modèle d'affaires

Modèle d'affaires	Nombre de requêtes	Patterns du trafic	Volatilité des données	Possibilités de mise en cache	Images et graphiques
B2C intermédiation financière	Très élevé	Atteint son sommet en début de matinée et en fin d'après midi. Ce pattern reste toutefois très sensible aux aléas conjoncturels.	Elevée pour les cours des titres. Faible pour les bulletins d'informations et les rapports d'activité.	Limitée sauf pour les bulletins d'informations et les rapports d'activités	Peu nombreux
B2C Shopping	Très élevé	Atteint son apogée pendant les périodes de fêtes et de liquidations	Les données qui servent à la génération des catalogues sont peu volatiles	Plus de possibilités de mise en cache en raison de la faible volatilité des données	Très nombreux
B2B	Peu élevé mais échange de gros fichiers XML	Sporadiques (transactions de gros reparties sur des intervalles de temps distants)	Peu volatiles	Idem	Peu nombreux
C2C	Très élevé	Augmente démesurément lorsque la période de validité de la vente aux enchères touche à sa fin	Très volatile parce chaque vente a une période courte de validité	Limitée	Très nombreux

3.2 Spécification des besoins en performance dans le cahier des charges de l'application

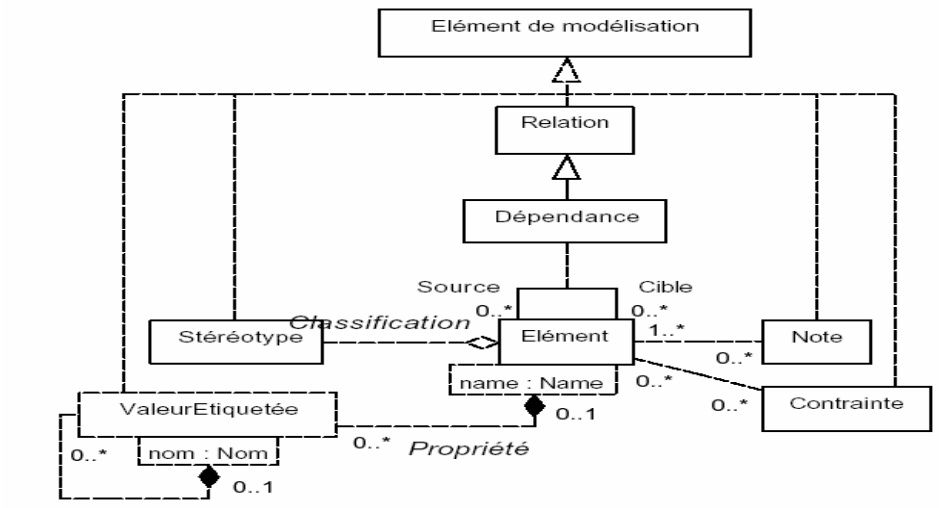
Avant 2001, les professionnels de l'informatique reprochaient à UML de ne pas avoir d'artefacts pour spécifier les besoins en performance au même titre que les besoins fonctionnels. Afin de combler cette lacune plusieurs groupes ont pris l'initiative d'en créer mais leurs efforts ont voué à l'échec parce que les résultats obtenus manquaient de standardisation. En validant le profil UML RTP (UML Profile for Schedulability, Performance, and Time) l'Object Management Group a apporté en 2001 une solution complète et standardisée pour la spécification des besoins en performance.

Dans cette section nous donnerons des exemples concrets pour montrer comment ce profil peut être utilisé pour spécifier les besoins en performance dans les cahiers des charges des applications de commerce électronique. Au préalable, nous expliquerons rapidement en quoi consistent les profils UML et quel est leur apport dans la spécialisation de la notation.

3.2.1 L'essentiel sur les profils UML

Comme le langage naturel, la notation UML est extensible à l'infini. Cette extensibilité est rendue possible grâce à ses propriétés récursives qui lui permettent de se décrire et de se représenter de façon autonome. Pour comprendre le concept de récursivité, il suffit de faire une analogie avec un dictionnaire. Les mots qui y figurent sont définis par d'autres mots de la même langue. Cette manière de définir les mots s'applique non seulement à ceux qui existent déjà dans le dictionnaire mais également à ceux qui vont l'intégrer dans le futur en tant que néologismes.

Figure 22: Eléments de base du méta-modèle d'UML[135]



Les profils sont pour la notation UML ce que les néologismes sont pour le langage naturel. Ils permettent de spécialiser et d'étendre la notation à de nouveaux domaines d'application sans remettre en cause la syntaxe et la sémantique définies dans le méta-modèle de base[136].

Dans le méta-modèle de base, les éléments qui servent à la création des profils sont les stéréotypes, les étiquettes et les contraintes. Ces éléments sont décrits ci-après.

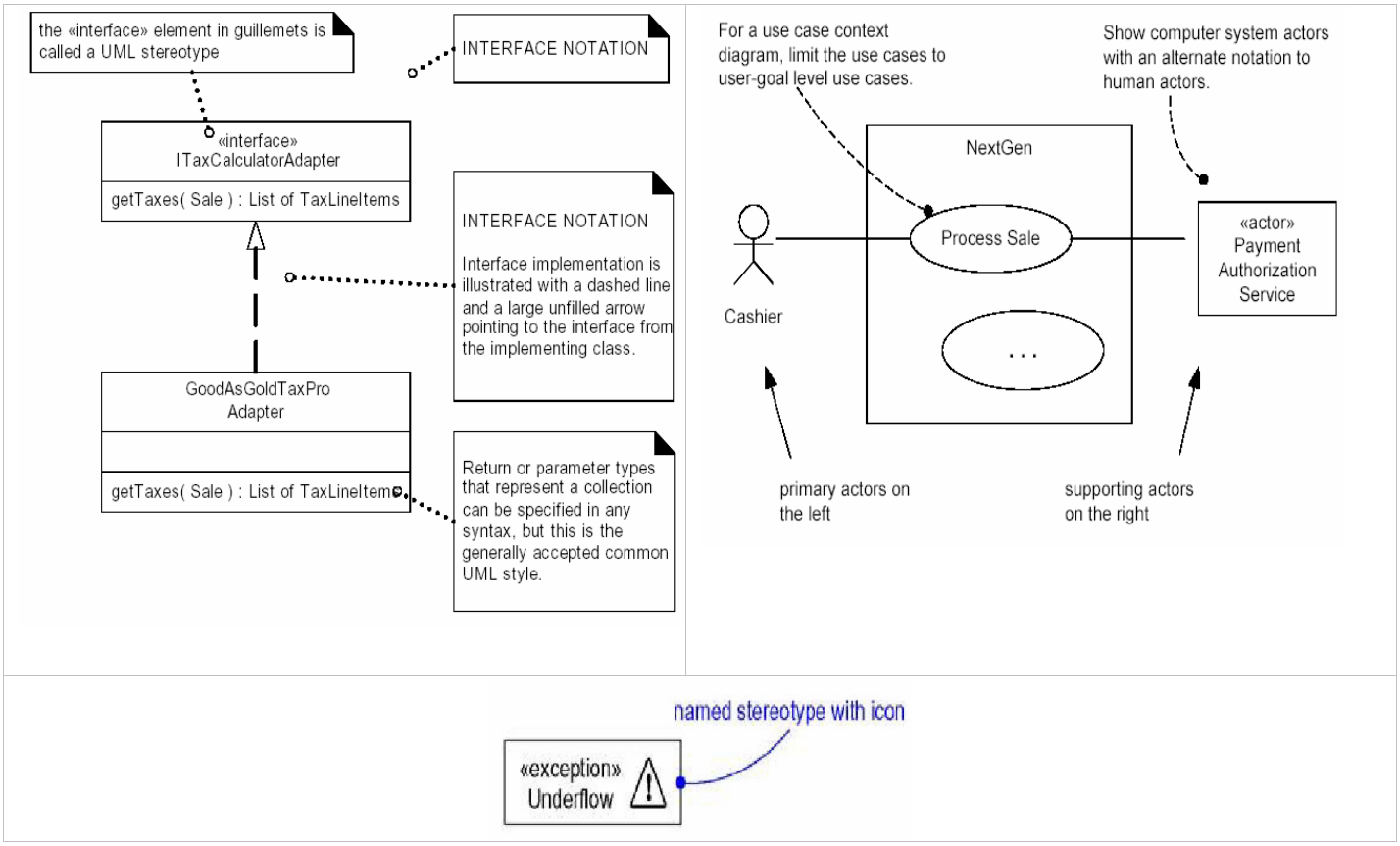
- **Les stéréotypes :** ils permettent de spécialiser la sémantique des éléments du méta-modèle de base. Par exemple pour représenter les exceptions du langage Java sur un diagramme de classe, on peut ajouter le stéréotype « exception » au symbole graphique qui représente les classes dans le méta-modèle[137].

[135] Pierre Alain Muller, modélisation objet avec UML , Rational Software Europe, avril 1997
[136] Il s'agit du méta-modèle original tel que défini par la spécification d'UML.
[137] Parce qu'une exception n'est rien d'autre qu'une classe

Pour marquer la différence entre un acteur humain et un acteur machine sur un diagramme de cas d'utilisation, on peut ajouter le stéréotype « actor » à ce même symbole. Pour représenter une classe d'INTERFACE et la distinguer d'une classe concrète, on peut faire la même chose avec le stéréotype « interface ». La figure 23 illustre ces différents exemples.

Avant de créer un stéréotype, il faut vérifier qu'il n'existe pas dans le méta-modèle de base un élément avec la même sémantique. Autrement on risque de démultiplier inutilement les synonymes et donc les confusions. La responsabilité de vérifier l'absence des redondances et la pertinence des nouveaux stéréotypes crées revient à des comités adhoc de l'Object Management Group. Lorsque ces derniers les valide, ils acquièrent dans leur champ d'application la même force de loi que les éléments du méta-modèle de base.

Figure 23: Exemples de stéréotypes spécialisant les éléments de base du méta-modèle d'UML[138]

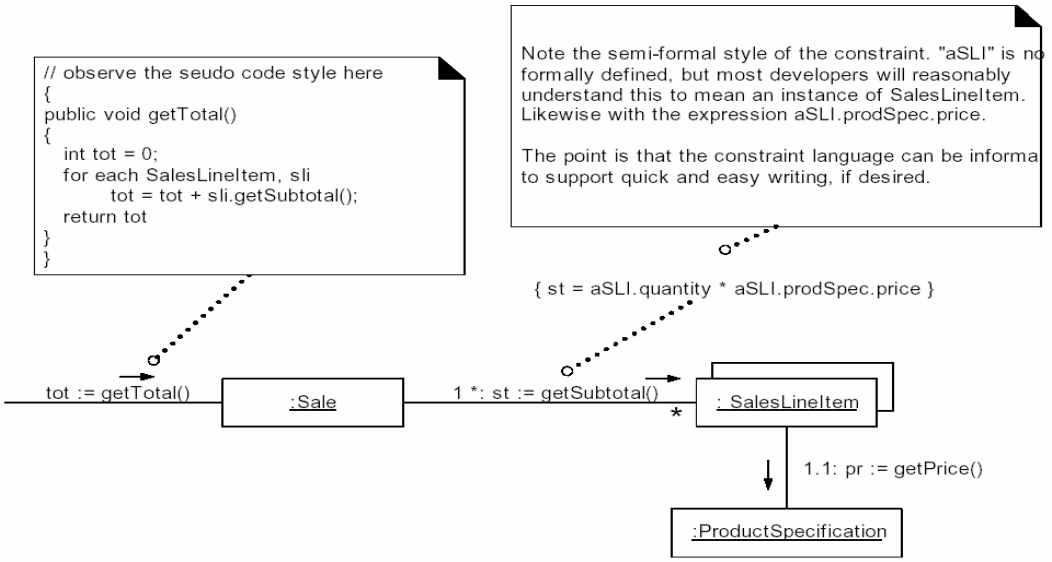


[138] Source: Craig Larman, An Introduction to Object-Oriented Analysis And Design And The Rup, 3 edition (October 20, 2004), Prentice Hall PTR, ISBN: 0131489062, 736 pages

- **Les contraintes** définissent les algorithmes et/ou les conditions que doivent vérifier les éléments de modélisation. La spécification UML n'impose pas de syntaxe particulière pour les formaliser.

  D'habitude elles sont exprimées en langage naturel, avec des expressions mathématiques ou avec l'Object Constraint Language. Par exemple dans la figure 24, la contrainte {st=aSLI.quantity * aSLI.prodSpec.price} définit l'algorithme et les conditions pour calculer le sous-total des ventes pour chaque ligne de produit.

Figure 24: Exemple d'une contrainte associée aux éléments de modélisation d'un diagramme UML[139]



- **Les étiquettes** correspondent à un couple (nom,valeur) qui décrit les propriétés d'un élément de modélisation. Elles étendent les attributs du méta-modèle et enrichissent la sémantique des éléments qu'elles qualifient[140]. Des exemples concrets sur l'utilisation des étiquettes seront donnés dans les prochaines sections. Pour le moment, il suffit de retenir que c'est, entre autres, par le biais des étiquettes que sont annotés les métriques de la performance sur les diagrammes d'UML.

  L'annexe 1 donne une liste des profils UML que nous avons recensés sur Internet. Il est important de noter que certains d'entre eux sont le fruit d'initiatives individuelles non encadrées par l'Object Management Group.

[139] Source: Craig Larman, An Introduction To Object-Oriented Analysis And Design And The Rup, 3 edition (October 20, 2004), Prentice Hall PTR, ISBN: 0131489062, 736 pages
[140] Basée sur la définition donnée par Pierre Alain Muller dans modélisation objet avec UML , Rational Software Europe, avril 1997

3.2.2 L'essentiel sur le UML Profile for Schedulability, Performance, and Time

Notre objectif dans cette section n'est pas d'étudier en détail le contenu de ce profil mais de présenter les éléments qui permettent d'annoter les benchmarks et les métriques de performance sur les diagrammes d'UML. Il est important de noter que ce profil n'est pas la seule alternative possible. Comme nous l'avons vu dans la section précédente, la notation UML est extensible à l'infini. Cette extensibilité confère aux équipes de développement le droit de créer leur propre profil par le biais des stéréotypes, des étiquettes et des contraintes. Pour notre part, nous recommandons d'utiliser des profils standardisés comme RTP[141] parce que :

- Ils sont basés sur les meilleurs patterns de l'industrie.
- Ils permettent aux équipes de développement de disposer d'un référentiel commun d'artefacts ce qui est une nécessité dans les grands projets de développement qui font intervenir des acteurs externes à l'entreprise.
- Ils sont supportés par plusieurs profilers et logiciels de modélisation (Rational softwares, Rhapsody, Objecteering Software…).

Cela dit, le profil RTP se compose de 6 sous-profils qui sont décrits sommairement dans le tableau 15.

Tableau 15: Présentation sommaire du profile UML RTP

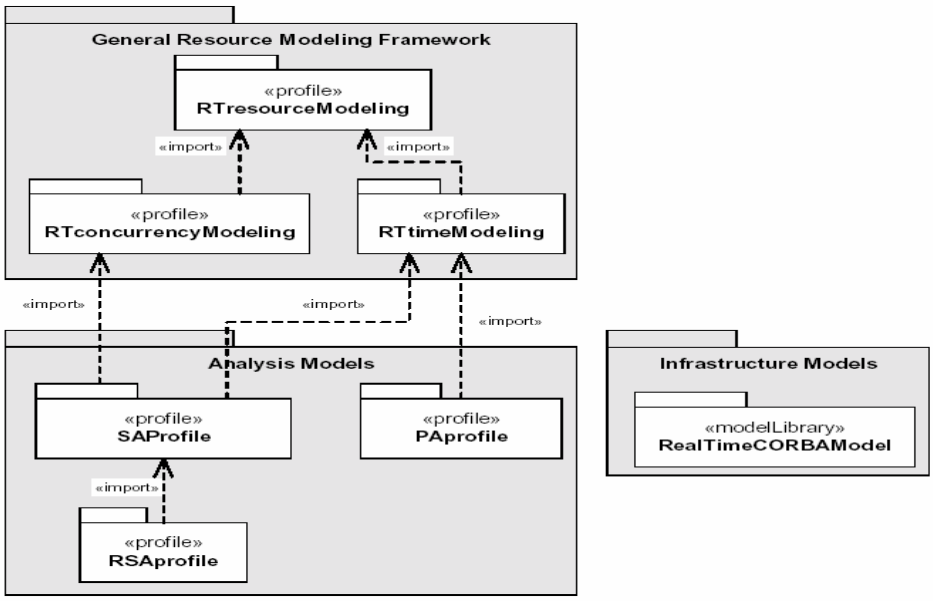
Sous-profil	Description
Ressources	Fournit des concepts, une terminologie et des éléments de modélisation pour associer aux diagrammes d'UML des informations quantitatives sur la qualité de service (QoS) de l'environnement physique et/ou logique de l'application (comme par exemple la capacité de traitement, la disponibilité…).
Temps	Fournit un ensemble de stéréotypes et d'étiquettes pour mieux représenter le concept de temps sur les diagrammes d'UML. Les éléments de modélisation fournis vont au delà de la simple sequentialité projetée par les diagrammes de séquence. Ils permettent d'annoter des informations beaucoup plus ciblées comme l'heure de début et de fin d'une activité, sa périodicité, le temps d'attente…

[141] Cet acronyme désigne le UML Profile for Schedulability, Performance, and Time

Suite du tableau

Sous-profil	Description
Concurrentialité (concurrency)	Fournit des stéréotypes, des étiquettes et des contraintes pour modéliser les aspects concurrentiels d'une application ou de son environnement matériel/logiciel (comme l'exécution des threads, les invocations synchrones et asynchrones, les queues…)
Planification (Schedulability)	Fournit des éléments de modélisation qui permettent de représenter des concepts de planification comme les priorités, les précédences, les événements déclencheurs (triggers), le partage et l'affectation des ressources…
Real-Time CORBA	Fournit un ensemble d'extensions pour adapter les éléments de modélisation du sous-profil planification aux applications CORBA.
Performance	Fournit des éléments de modélisation pour intégrer les mesures quantitatives de la performance aux diagrammes d'UML. Etant donné l'objectif du présent essai, **seuls certains de ces éléments seront étudiés dans la suite de cette section.**

Figure 25: Structure du profil RTP[142]



[142] Source: OMG, UML Profile for Schedulability, Performance, and Time Specification, September 2003 Version 1.0

Les éléments de modélisation relatifs aux performances sont décrits en détail dans l'annexe 2. Dans cet essai nous n'allons utiliser qu'un sous-ensemble de ces éléments pour ne pas ajouter aux diagrammes une couche inutile d'abstraction et de complexité. Lors du choix de notre anthologie nous avons accordé le prima à la simplicité, à la représentativité et à la lisibilité. Dans la figure 26 :

PerfmanceContext rassemble les différents scénarios (PScenario) qui permettent de spécifier les performances de l'application ou de son environnement[143]. Sur les diagrammes d'UML ces scénarios sont représentés à l'aide du stéréotype «PAcontext». Dans la version officielle du profil RTP aucune étiquette n'est associée à «PAcontext».

PStep représente à l'intérieur d'un scénario, une étape particulière dans l'exécution d'une action[144]. Son stéréotype est « PAstep ». Typiquement il est annoté sur les actions ou les messages des diagrammes de séquence mais cet usage n'est pas limitatif[145]. Plusieurs étiquettes permettent de qualifier les éléments de modélisation auxquels « PAstep » est associé. Les plus pertinents eu égard aux besoins de cet essai sont :

- PArespTime: temps de réponse du Step.
- PAdemand : temps total requis pour exécuter le Step.
- PArep: nombre de fois que le Step est répété dans un même scénario.
- PAdelay: délai d'attente entre l'exécution de deux ou plusieurs Step.

Workload permet de spécifier la charge de travail. Elle est représentée sur les diagrammes à l'aide des stéréotypes «PAopenLoad» et «PAclosedLoad». Selon le profil «PAopenLoad» correspond à un nombre variable de requêtes qui suit une distribution statistique donnée. «PAclosedLoad» en revanche désigne un nombre fixe d'utilisateurs qui interagissent avec l'application ou son environnement moyennant un temps de réflexion (think time) entre les différentes interactions. Les principales étiquettes associées à ces deux stéréotypes sont :

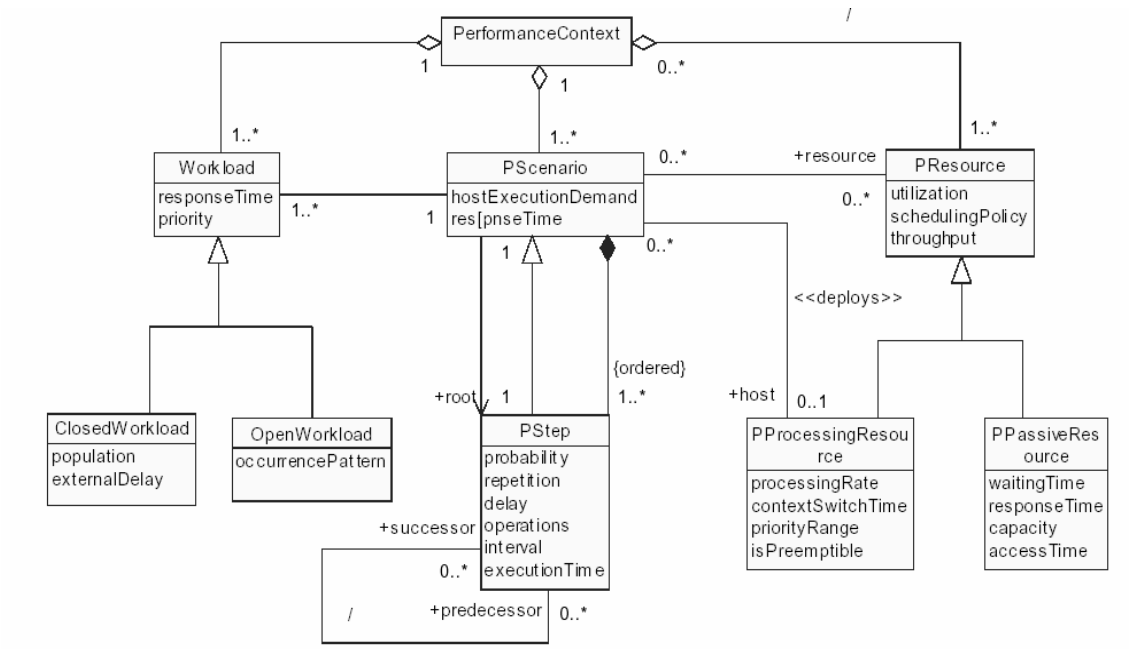
- PApriority: priorité de la charge de travail.
- PAoccurrence: distribution statistique du «PAopenLoad»
- PApopulation: taille du «PAclosedLoad» (soit le nombre d'utilisateurs).

[143] Par exemple il peut être utilisé pour décrire un scénario d'heures de pointe pendant lesquelles la charge de travail traitée par le processeur atteint son paroxysme.
[144] Par exemple l'initialisation d'une servlet
[145] Consulter l'annexe 2 pour connaître la liste des méta-classes associées à ce stéréotype

PResource désigne un dispositif sur lequel les scénarios sont exécutés (comme par exemple : un processeur, un serveur ou une unité de stockage). «PAhost» permet de la stéréotyper sur les diagrammes. Les principales étiquettes qui lui sont associées sont:

- PAutilization: correspond au nombre d'utilisateurs concurrentiels.
- PAthroughput: correspond au débit.

Figure 26: Sous-profil de performance tel que défini par UML RTP[146]



Nous n'irons pas plus loin dans la description des éléments du profil RTP parce que nous disposons désormais de tous les stéréotypes et étiquettes nécessaires pour spécifier les métriques de la partie 2 sur les diagrammes d'UML. Compte tenu de la sémantique qu'ils ajoutent au méta-modèle ces stéréotypes et étiquettes ne peuvent être annotés que sur les diagrammes qui reflètent implicitement ou explicitement les aspects suivants :

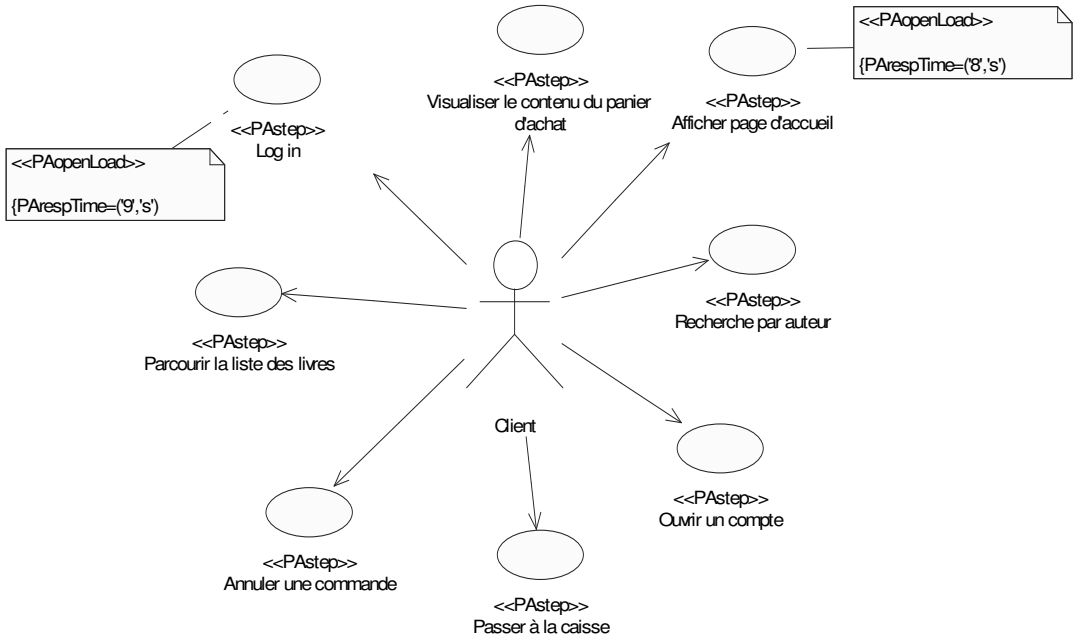
- **L'aspect temporel:** qu'on retrouve sur les diagrammes de séquence mais aussi sur les diagrammes d'état et d'activité.

[146] Source: OMG, UMLTM Profile for Schedulability, Performance, and Time Specification, September 2003 Version 1.0

- **L'aspect collaboratif :** qu'on retrouve explicitement sur les diagrammes de collaboration/ séquence et tacitement sur les diagrammes de cas d'utilisation.
- **L'aspect fonctionnel :** projeté par les cas d'utilisation.

Dans la suite de cette section nous donnerons deux exemples concrets sur l'utilisation du profil RTP pour spécifier les besoins en performance.

Figure 27: Exemple d'annotations ajoutées aux diagrammes de cas d'utilisation



La figure 27 illustre les principaux cas d'utilisation dans un site de vente de livre. Sur cette figure nous avons spécifié le temps de réponse au terme duquel la page d'accueil doit s'afficher. Nous avons associé le stéréotype « PAstep » au cas d'utilisation 'afficher page d'accueil' pour marquer une étape parmi d'autres dans l'utilisation du site. Le temps (PAresTime) est exprimé à l'aide du couple ('8','s') signifiant 8 secondes.

NB : Il est important de noter que les stéréotypes et les étiquettes du profil RTP peuvent designer aussi bien les performances souhaitées que les performances mesurées[147].

[147] C'est à dire les performances réelles.

Le diagramme d'activité de la figure 28 illustre les interactions et les collaborations nécessaires pour afficher les informations d'un produit à partir de deux bases de données différentes[148].

Sur ce diagramme on retrouve, entre autres, des indications sur le nombre d'accès concurrentiels ($NUsers), une estimation (est) du temps moyen (mean) requis ('req') pour compléter chaque Step, ainsi qu'une indication sur le temps de réflexion (PAexDelay ou PAdelay).

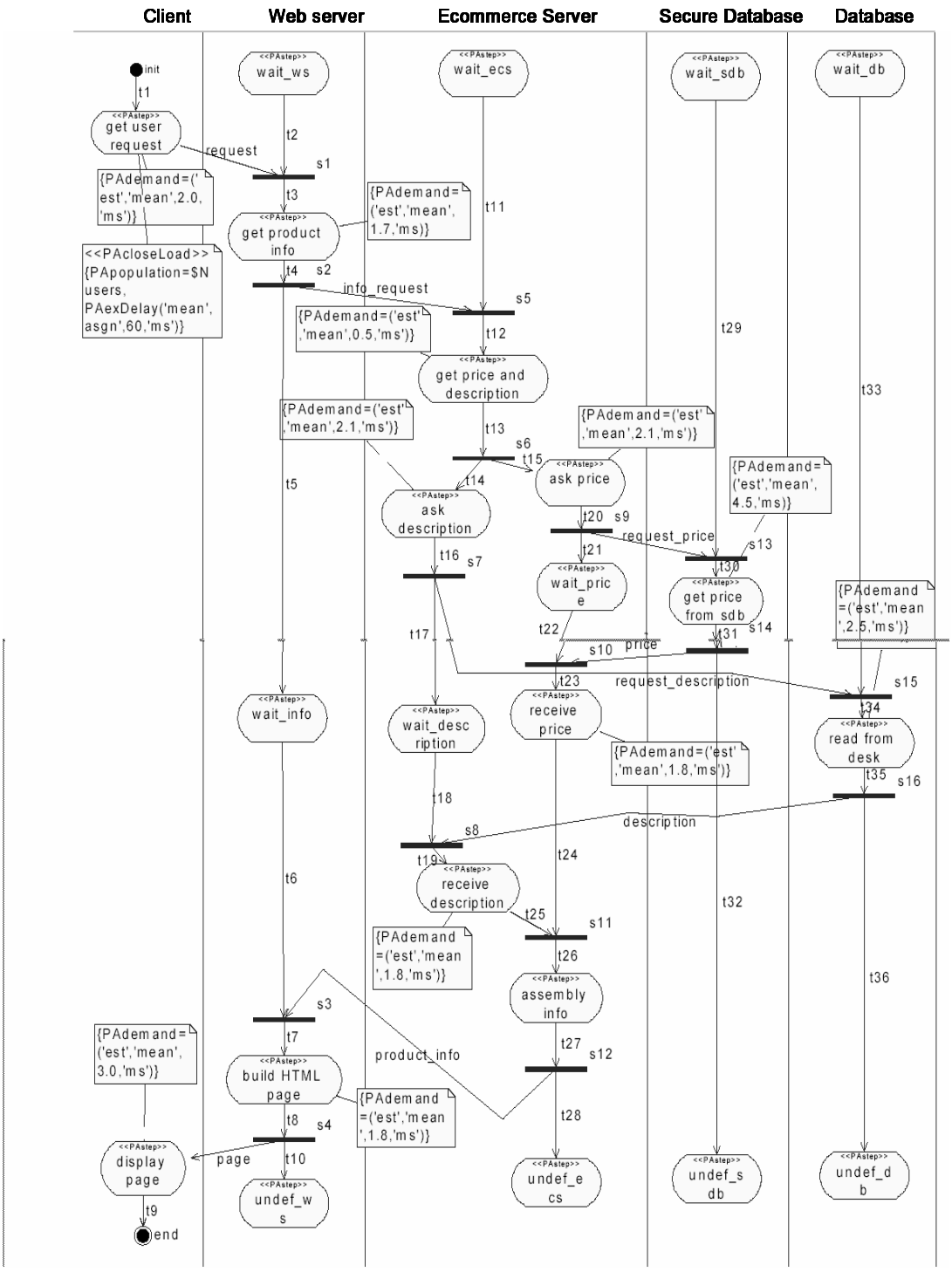
Pour se familiariser d'avantage avec les éléments du sous-profil RTP et la modélisation UML dans les applications de commerce électronique nous recommandons la lecture des deux livres suivants :

- Bruce Powel Douglass, Real Time UML: Advances in The UML for Real-Time Systems, Third Edition, Addison Wesley, February 20, 2004, ISBN: 0-321-16076-2, 752 pages

- Doug Rosenberg , Kendall Scott, Applying Use Case Driven Object Modeling with UML: An Annotated e-Commerce Example, Addison Wesley, First Edition June 14, 2001, ISBN: 0-201-73039-1, 176 pages

[148] Celle qui contient les prix est sécurisée mais celle qui contient les autres informations relatives au produit ne l'est pas.

Figure 28: Exemple d'annotations ajoutées aux diagrammes d'activité [149]



[149] Source: Early Evaluation of Software Performance based on the UML Performance Profile, Gordon Ping Gu and Dorina C. Petriu, Department of Systems and Computer Engineering, Carleton University, Ottawa

4. Gestion de la performance dans la phase de design

Dans cette partie il sera question d'évaluer l'impact du design sur les performances des applications de commerce électronique. Pour parvenir à cet objectif nous passerons en revue plusieurs anti-patterns qui reflètent les erreurs fréquemment commises dans le design des composants et des architectures auxquelles ils sont intégrés. Au fur et à mesure nous proposerons des patterns pour corriger ces erreurs ou en réduire les conséquences.

4.1 Impact de l'architecture de l'application sur les performances

Les architectures des applications web développées en Java relèvent communément de l'une des deux catégories suivantes :

- Architecture Model 1 appelée aussi JSP centric architecture.
- Architecture Model 2 qui est une adaptation de l'architecture MVC[150] aux applications web.

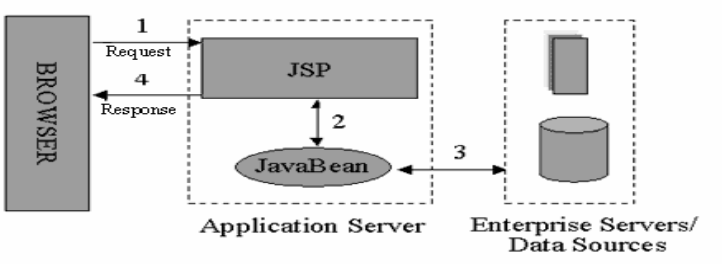
Dans le reste de cette section nous examinerons l'impact de ces deux architectures sur la performance des applications de commerce électronique.

4.1.1 Architecture Model 1

Dans cette architecture les JSP cumulent deux rôles :

- Celui du contrôleur chargé de diriger les requêtes entrantes vers un Java Bean qui encapsule toute la logique d'affaires nécessaire pour les traiter.
- Celui de la vue qui permet au client d'interagir avec l'application et d'afficher le résultat des traitements effectués par le Java Bean.

Figure 29: Architecture Model 1[151]



[150] Cette architecture a été initialement conçue par SmallTlak au début des années 80
[151] Source : Govind Seshadri, Understanding JavaServer Pages Model 2 architecture, http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html

Avant que cette architecture ne se répande, les développeurs avaient coutume de placer tout le code Java dans des servlets qui étaient directement appelées à partir d'un simple formulaire HTML. Les servlets couplaient à la fois la logique d'affaires et les instructions permettant de gérer les interactions avec la vue. Plus tard il s'est avéré que ce couplage rendait difficile la reutilisabilité du code, la maintenance des applications et la répartition des tâches au sein des équipes de développement. C'est alors qu'on a décidé de migrer vers l'architecture Model 1 en sectionnant le code initialement inclus dans les servlets en deux parties bien distinctes comme le montre la figure 29.

Si ce découplage rend plus facile la maintenance et la répartition des tâches, des interrogations subsistent quant à son impact sur les performances. En effet plus on ajoute des couches et des composants à l'application, plus on accroît la consommation des ressources nécessaires pour la faire fonctionner (mémoire, CPU..). Cet argument était d'ailleurs souvent cité par les architectes et les développeurs pour justifier leur réticence face à l'utilisation des patterns. Sa recevabilité reste néanmoins discutable. On peut leur donner tort ou raison dépendamment de la taille de l'application, des modifications qu'on compte lui apporter mais aussi des compétences de l'équipe de développement.

La concentration de tout le code dans les servlets reste une alternative simple lorsqu'on développe des applications de très petite taille, auxquelles on ne compte pas ajouter de nouvelles fonctionnalités et lorsque les développeurs sont polyvalents[152]. Dans un tel scénario pourquoi créer un Java Bean et une JSP si on a la possibilité d'insérer toute la logique de présentation et d'affaires dans une servlet. Outre la mémoire additionnelle qu'il faut allouer pour leur instanciation, cela entraînerait une augmentation du temps de réponse parce que le chemin emprunté par les requêtes devient plus long.

Dans les applications de taille moyenne développées par des équipes plus spécialisées, les arguments évoqués ne sont plus recevables. Il est vrai que l'ajout des composants et des couches augmentent le temps de réponse mais comme nous l'avons vu dans la partie 2, la performance n'est pas seulement une affaire de temps mais également une affaire de coûts.

[152] C'est-à-dire qu'ils sont capables de développer en Java, HTML, Javascript **et** tout autre technologie qui peut rentrer dans la construction de l'application

En d'autres termes, il vaudrait mieux sacrifier quelques millisecondes de temps de réponse si en contrepartie on peut économiser des sommes substantielles sur les coûts de maintenance et sur les coûts des erreurs d'origine humaine imputables à la mauvaise répartition des tâches au sein des équipes de développement. Bien évidemment ce sacrifice doit tenir compte des métriques et des benchmarks définis dans la partie 2.

4.1.2 Architecture Model 2

L'architecture Model 1 permet, certes, d'encapsuler la logique d'affaires dans les Java Beans mais elle n'évacue pas entièrement le code Java des JSP. Cela pose d'importants problèmes d'extensibilité et de maintenance dans les applications de grande taille développées par des équipes très spécialisées. Ces problèmes se résument comme suit :

- Les équipes de développement tombent rapidement dans le piège de la décomposition fonctionnelle en ce sens qu'elles créent une nouvelle JSP pour chaque nouvelle fonctionnalité de l'application. Elles se retrouvent ainsi avec un nombre exorbitant de JSP et plusieurs lignes de code mal factorisé.
- Les erreurs d'origine humaine se multiplient parce que les membres de l'équipe de développement touchent accidentellement à des fragments de code qui ne relèvent pas de leurs attributions (par exemple un designer qui altère accidentellement le code Java alors qu'il n'est censé travailler que sur la partie HTML, CSS et Javascript).

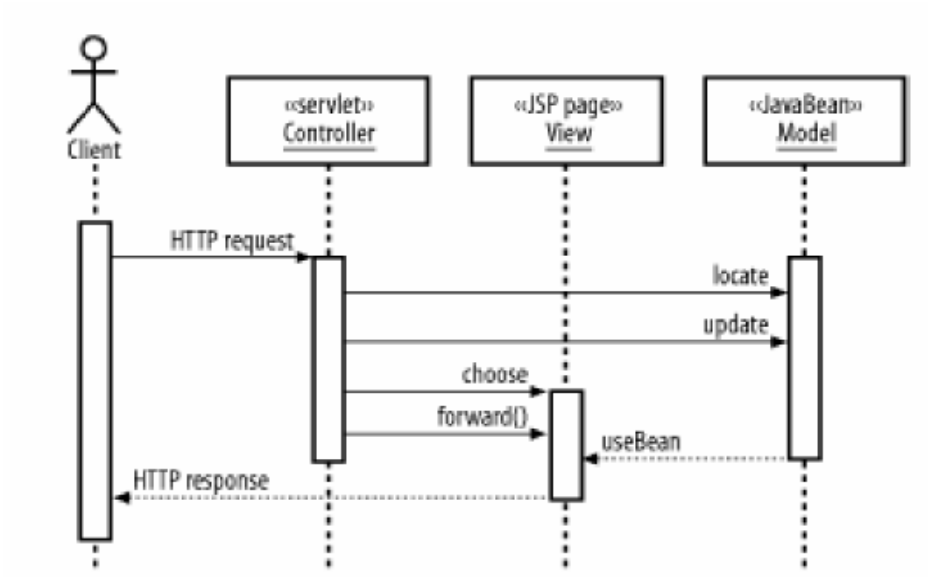
La résolution de ce problème exigeait une autre séparation entre la logique de présentation et la logique de coordination qui permettait aux JSP de diriger les requêtes vers les Java Beans. C'est ainsi qu'on a ravivé l'ancienne architecture MVC de SmallTalk que Sun a pour l'occasion baptisé Model 2 en raison de l'utilisation conjointe des JSP et des servlets dans la même application. En plus d'ajouter une servlet à l'architecture il a fallu créer des balises XML comme JSTL pour évacuer entièrement le code Java des JSP. Assurément, cela a permis aux designers de disposer d'une syntaxe qui leur est beaucoup plus familière mais a occasionné une augmentation des ressources consommées et du temps de réponse notamment en raison du parsing des balises XML et de l'utilisation de langages comme XSL ou XSLT pour les transformations de leur contenu et la gestion de leur présentation.

Soulignons par ailleurs que les requêtes n'ont plus qu'un seul point d'entrée vers l'application alors que dans l'architecture Model 1 chaque JSP constitue un point d'entrée en soi. D'emblée on pourrait présumer que ce design provoque une congestion des requêtes au niveau du contrôleur et par conséquent un retard dans leur acheminement vers le Model.

Il est important de souligner que le pattern MVC ne proscrit pas la création de plusieurs contrôleurs au sein de la même architecture. Pour étayer cette affirmation nous nous permettons de reproduire un extrait du livre J2EE Design Patterns[153].

" As an architecture, MVC is a good start and suitable for many applications. But sometimes more advanced processing is called for, so it's time to start filling in the holes in the MVC architecture, starting with the controller. Not only is the controller the first point of contact for requests, it is also the place where we have the most programming and design freedom... ***The MVC pattern does not specify how many controllers there should be****."*

Figure 30: Architecture Model 2[154]

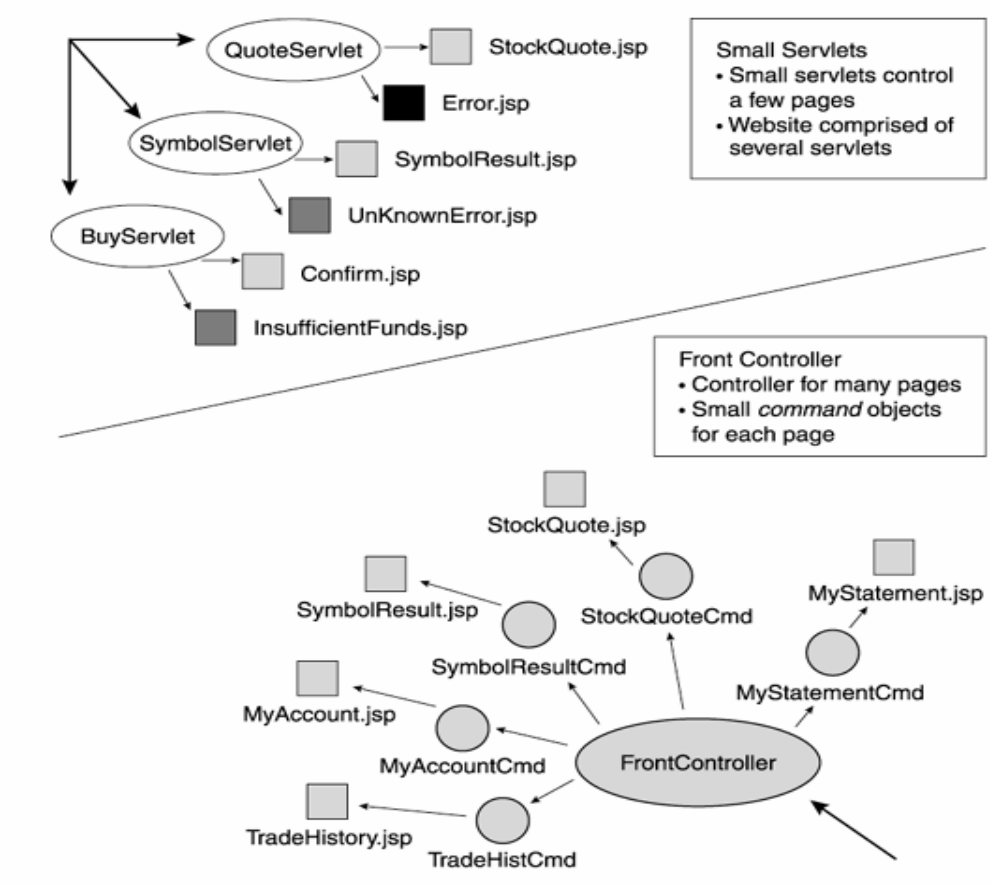


[153] William Crawford, Jonathan Kaplan, J2EE Design Patterns, O'Reilly, September 2003, 0-596-00427-3, pages 368
[154] Source: idem

Comme le montre la figure 31 la création de plusieurs contrôleurs consomme plus de ressources mais elle permet de répartir la charge de travail sur un nombre restreint de servlets. De cette façon il y a moins de congestions au niveau des contrôleurs et les requêtes arrivent plus rapidement à destination.

Cela dit, il n'est pas question dans cette section de remettre en cause la pertinence de l'architecture Model 2 pour une centaine de millisecondes de plus dans le temps de réponse. En effet la facilité de maintenance et l'extensibilité sont aussi importantes que la performance et doivent être prises en compte dans le design de l'application.

Figure 31: Architecture multiController versus architecture FrontController[155]



[155] Source: Small servlets versus front controller strategy. From IBM Software Group Services Performance Workshop presented by Stacy Joines, 2001, Hursley, U.K. © IBM Corp. 2001. Reprinted by permission of IBM Corp.

4.2 Impact du design des composants sur les performances

Dans le reste de cette section, nous prenons pour acquis que l'architecture MVC est un bon choix aussi bien pour la performance que pour la maintenance et l'extensibilité des applications de commerce électronique. C'est d'ailleurs sur cette base que nous présenterons les anti-patterns et patterns qui doivent régir le design des composants.

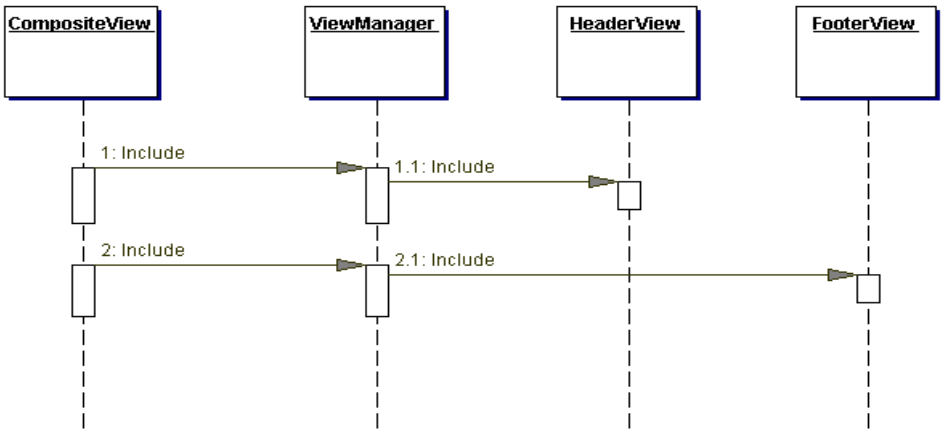
4.2.1 Anti-patterns et patterns de la couche de présentation

4.2.1.1 Les vues composites

La construction dynamique des pages web par assemblage de composants est devenue un standard de fait dans les applications de commerce électronique. Il suffit de parcourir les sites de grandes compagnies comme Ebay ou BestBuy pour constater que les pages sont de plus en plus façonnées à partir d'une collection de blocs qui puisent leurs données de plusieurs sources locales et/ou distantes statiques et/ou dynamiques (par exemple : un bloc qui récupère dynamiquement les cours des actions via un service web, un autre qui importe des données syndiquées (RSS feeds) à partir d'un fichier xml placé sur un site tiers ou encore un bloc qui contient le logo et les informations de copyright…).

Cette manière de construire les pages web facilite l'administration des sites et la reutilisabilité des blocs mais elle consomme beaucoup de ressources comme le montre le diagramme de séquence suivant.

Figure 32: Génération dynamique des pages web par assemblage de composants (pattern du composite view)[156]



[156] Sun Microsystems, Core J2EE Pattern Catalog,
http://java.sun.com/blueprints/corej2eepatterns/Patterns/CompositeView.html

Dans les applications Java les pages sont typiquement composées selon une ou plusieurs des techniques infra :

- Utilisation du mécanisme d'inclusion intégré aux JSP : un peu comme le font les frames d'HTML, cette technique permet d'incorporer à la volée une page à l'intérieur d'une autre. Nous verrons dans la partie 5 que son efficacité est subordonnée au type d'inclusion utilisée (include directive versus include action) et à la nature du contenu inclus (statique ou dynamique). En règle générale il faut éviter de l'utiliser lorsque le contenu incorporé ne peut pas être mis en cache.

- Utilisation des Java Bean : les Java Beans sont des composants sérialisables qui font massivement appel aux unités d'entrée/sortie (I/O) pour assurer la persistance de leurs propriétés. Vu que ces appels sont coûteux en mémoire, en processeur et en temps de réponse, il faut réduire leur utilisation au strict minimum. Il faut aussi raccourcir leur durée de vie et réduire leur portée (scope) pour éviter une occupation inutile de la mémoire lorsque la vue n'en a plus besoin. Enfin il faut essayer dans toute la mesure du possible de stocker les propriétés des Java Bean dans des bases de données parce qu'elles gèrent les mises à jour multiples (update) plus efficacement que le système de fichiers (unités d'entrées sorties I/O).

- Utilisation des librairies de tags personnalisés (custom tags) : lorsque le conteneur trouve un tag personnalisé dans le code d'une JSP, il procède au parsing du Tag Library Descriptor (fichier .tld) pour déterminer le chemin d'accès à la classe à laquelle le tag est adossé. Au passage il charge aussi le Tag Handler et appelle des méthodes qui permettent de gérer son cycle de vie. Ces opérations sont pénalisantes aussi bien pour la mémoire que pour le processeur. Si elles sont effectuées à une grande échelle elles peuvent dégrader brutalement la performance de l'application. Comme les Java Bean, il faut limiter l'utilisation des tags personnalisés au strict minimum. Il faut aussi créer un nombre restreint de Tag Handler dans le resource pool pour éviter au conteneur d'instancier le Tag Handler à chaque fois qu'il croise un tag personnalisé dans le code de la JSP.

  Cela dit l'utilisation de la mémoire cache du browser pour conserver des données peu ou pas volatiles est souvent d'un grand apport dans l'amélioration des performances des vues composites.

Par exemple, au lieu de télécharger à maintes reprises une applet ou le logo, il est possible de les stocker dans la mémoire cache du browser de façon à soulager le trafic sur le réseau et à réduire le temps de réponse perçu par le client. La principale difficulté avec cette technique a trait à la fréquence de mise à jour du cache. Normalement c'est au serveur d'avertir le client lorsque les données changent mais malheureusement les browsers ne supportent pas ce mode de communication. Le modèle pull du web veut que la communication soit initiée par les browsers. C'est donc à eux d'aller vérifier à la source que les données n'ont pas changé. Cette vérification doit être occasionnelle sinon l'utilisation de cette technique doit être remise en question.

NB : des librairies de tags spécialisés comme OSCache offrent une solution complète et simple pour cacher ou mettre sur disque les différents composants d'une JSP. Ces libraires seront étudiées avec plus de détails dans la partie 5.

4.2.1.2 La gestion des sessions au niveau de la couche de présentation

La personnalisation des services sur un site de commerce électronique passe nécessairement par la collecte de données précises sur les besoins des clients et le suivi de leurs différentes interactions avec le site. Intrinsèquement, le protocole http ne prend pas en charge ces deux tâches parce qu'il est conçu pour mettre fin à la connexion courante dès que le serveur web envoie une réponse aux différentes requêtes reçues. A vrai dire, le protocole http a une mémoire éphémère (stateless). Après la clôture de la connexion courante, il ne se souvient ni des clients qui ont déjà eu recours à ses services ni des données transportées par les différentes requêtes qu'ils ont envoyées.

Pour permettre à l'application Web de se souvenir des données des clients avec lesquels elle a eu de récentes interactions, les développeurs ont recours au mécanisme des sessions[157]. Dans Java celui-ci est souvent mis en œuvre par le biais de trois artifices :

- **Les champs HTML cachés :** cette technique consiste à insérer des indices (tokens) dans des champs cachés d'un formulaire HTML[158]. De cette manière l'application reconnaît le client à chaque fois que l'indice lui est transmis par une requête http.

[157] Dans la plupart des cas les informations stockées dans les sessions expirent lorsque le client ferme le navigateur, se déconnecte (log out), se dirige vers un autre site ou encore lorsque la durée de la session est révolue.
[158] Champs dont l'attribut TYPE est égal à HIDDEN.

- **Les cookies :** chaîne de caractères déposée dans un fichier sur le disque dur du client. A moins qu'elles ne soient bloquées, les données stockées dans cette chaîne sont transmises à l'application Web à chaque fois qu'une requête lui parvient du même client.
- **La réécriture d'URL :** cette technique est souvent utilisée lorsque le client bloque la réception des cookies. Elle consiste à ajouter un identifiant unique à tous les liens hypertextes qui se trouvent sur la vue. A chaque fois que le client clique sur ces liens une requête contenant l'identifiant unique est envoyée à l'application pour lui permettre de le reconnaître.

Les artifices décrits ci-dessus font appel à des données insérées dans la vue ou sur le disque dur du client. Ils décentralisent la gestion des sessions au niveau de la couche de présentation. Cette décentralisation peut s'avérer fatale pour la performance de l'application car :

- L'utilisation de la technique des champs cachés avec la méthode POST[159] ne contingente pas la quantité de données échangée entre le client et le serveur. A mesure que cette quantité croît le réseau s'engorge et le temps d'acheminement des requêtes vers le serveur augmente[160].
- La réécriture d'URL augmente le temps nécessaire pour générer la vue et réduit les possibilités de mise en cache.
- Les données véhiculées entre le client et le serveur ne sont pas réutilisées par les différents composants[161] de l'application. Elles doivent être re-collectées à chaque fois qu'un composant en a besoin. Il en résulte des redondances dans l'envoi /réception des données et une mauvaise factorisation du code au niveau des composants.

Même s'il s'appuie sur les cookies ou la réécriture d'URL, le mécanisme de gestion de session intégré au package javax.servlet.http.HttpSession permet de réduire de façon significative la quantité de données véhiculées entre le client et le serveur.

[159] La méthode GET en revanche limite cette quantité à 240 caractères
[160] A en croire un article paru sur precisejava.com la performance diminuerait de façon inversement proportionnelle à la quantité de donnée véhiculée. Pour plus de détails voir Ravi Kalidindi and Rohini Datla , Best Practices to improve performance in Servlets, http://www.precisejava.com/javaperf/j2ee/Servlets.htm#Servlets106
[161] Servlets, JSP …

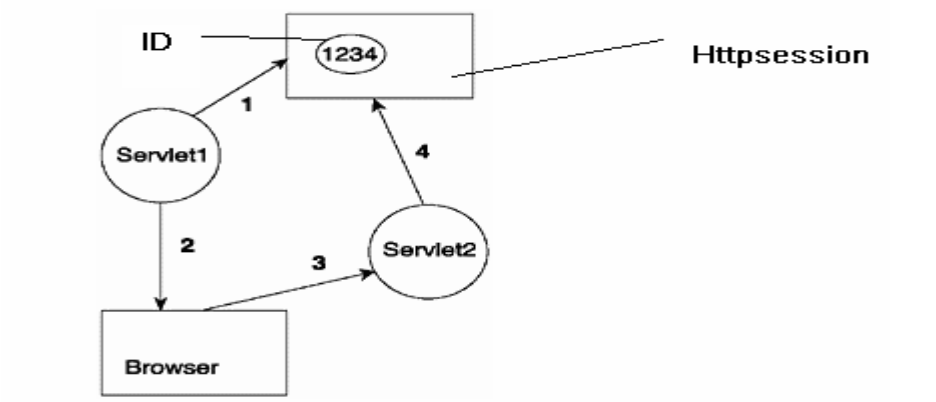
Ainsi lorsque le client envoie pour la première fois une requête à l'application, le conteneur servlet lui attribue un identifiant unique[162] qui permet de le reconnaître à chaque nouvelle interaction. Il crée ensuite un Hashtable[163] appelé HttpSession object où il stocke les données reçues du client. De cette façon ces données ne sont transportées qu'une seule fois vers l'application.

Après leur stockage dans l'objet HttpSession elles peuvent être lues ou modifiées par d'autres servlets et JSP. Il est important de souligner que le conteneur crée autant d'objets HttpSession qu'il y a de clients et que chaque client ne peut accéder qu'aux données de l'objet qui lui est dédié.

Les avantages de cette technique se situent à deux niveaux :

- Le réseau est soulagé parce que l'identifiant unique est la seule donnée transférée entre le client et le serveur.
- Les données sont lues et modifiées plus rapidement parce qu'elles sont stockées dans la mémoire vive du serveur.

Figure 33: Gestion des sessions par le biais du mécanisme intégré à l'objet HttpSession [164]



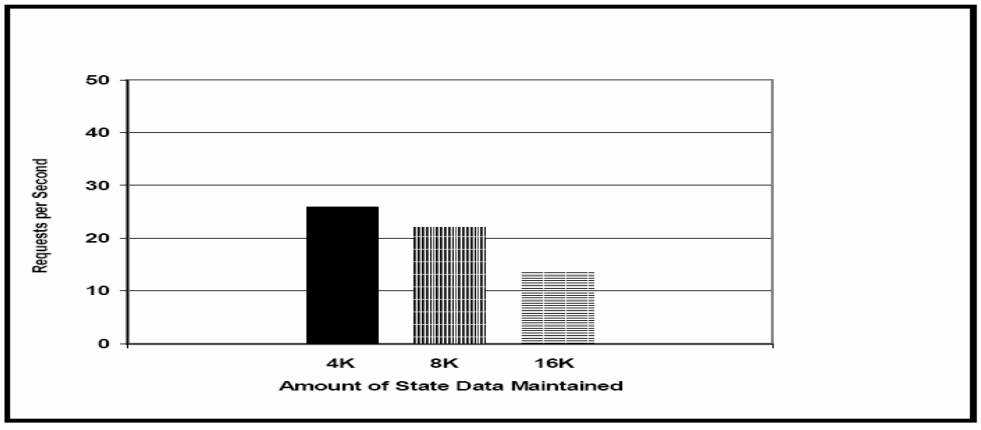
Notons cependant qu'au fur et à mesure du stockage de ces données les objets HttpSession réduisent la taille de la mémoire disponible pour l'application et affectent à la baisse le débit du conteneur comme le montre la figure 34.

[162] Cet identifiant est attribué par le biais d'un cookie ou de la réécriture d'URL
[163] Le conteneur crée un nouvel objet pour chaque session
[164] Source: inspiré d'un schéma du livre Budi Kurniawan, Java for the Web with Servlets, JSP, and EJB, A Developer's Guide to J2EE Solutions, New Riders Publishing, April 12, 2002, 0-7357-1195-X pages 976

La spécification J2EE n'offre à ce jour aucun moyen pour plafonner la taille de la mémoire occupée par les objets HttpSession mais les techniques décrites ci-dessous permettent d'en juguler la progression.

- Configurer le conteneur de façon à limiter le nombre d'objets HttpSession crées par l'application.
- Stocker seulement les clés[165] dans les objets HttpSession et conserver le reste des valeurs dans la base de données de l'application. Avec cette technique il faut, cependant, s'attendre à une augmentation du temps de réponse.
- Réduire la durée de vie des objets HttpSession afin que le conteneur puisse libérer de la mémoire lorsque le client arrête d'interagir avec le site. Cette réduction doit, néanmoins, tenir compte des cas d'utilisation pour ne pas perturber le fonctionnement global de l'application. Par exemple si on utilise les objets HttpSession pour stocker les articles d'un panier d'achat, il faut éviter de trop raccourcir leur durée de vie. Sinon le panier risque de se vider avant que le client n'ait le temps de passer à la caisse. A l'opposé les durées de vie courtes sont plus tolérées sur les sites d'intermédiation financière parce qu'elles contribuent au renforcement de la sécurité. Cela dit, la plupart des conteneurs attribuent par défaut une durée de validité de 30 minutes aux objets HttpSession mais cette durée peut être aussi modifiée à partir du code de l'application comme nous le verrons dans la partie 5.

Figure 34: Impact de la taille d'un objet HttpSession sur le débit du Websphere Application Server[166]



[165] Rappelons que les objets HttpSession sont des Hastables qui stockent les données dans un vecteur (clef, valeur).
[166] Source : Harvey W. Gunther, WebSphere Application Server, Development Best Practices for Performance and Scalability, IBM White Paper, September 7, 2000

- Intégrer à l'application une fonctionnalité qui permet au client de se déconnecter (log out) à la fin de chaque visite. L'objectif étant d'invalider l'objet HttpSession qui lui a été attribué dès la survenance de la déconnexion. Cette technique présente toutefois des limites décrites dans cet extrait du livre Performance Analysis for Java™ Web Sites[167].

  " While we recommend a logout feature, we don't really expect it to solve HTTP session management for your web site. Sadly, most users never touch the logout buttons on web sites they visit. Most users just move to the next web site without formally logging out of yours. So, while it provides some benefits, the logout function is only a part of an overall HTTP session management strategy. If your web application handles sensitive data (such as financial information) and may be accessed from a shared workstation or kiosk, consider a logout function to be a requirement. Logging out prevents subsequent users from obtaining a previous visitor's information via an existing HTTP session. Naturally, these applications generally support a very short HTTP session timeout interval and may force a logout after completing certain tasks."

4.2.1.3 La validation des données au niveau de la couche de présentation

Les développeurs doivent souvent valider les données renseignées dans les formulaires électroniques aussi bien au niveau du client qu'au niveau du serveur. Ces validations peuvent être complémentaires ou redondantes dépendamment de la confidentialité de l'algorithme de validation, des ressources nécessaires pour l'implémenter et de la configuration du browser utilisé par le client.

Par exemple si l'objectif de l'algorithme de validation est de vérifier qu'un numéro de carte de crédit respecte la structure Master Card ou VISA, il serait à première vue aberrant de l'implémenter au niveau du serveur. Cet algorithme est en effet public. De plus des appels fréquents à un serveur distant peuvent l'épuiser inutilement et augmenter le temps de réponse perçu par le client. En revanche si l'objectif de l'algorithme est de vérifier que le numéro de la carte de crédit existe dans la base de données de la banque est qu'il n'a pas été opposé par le détenteur de la carte, son implémentation au niveau du serveur devient incontournable.

[167] Stacy Joines,Ruth Willenborg, Ken Hygh, Performance Analysis for Java™ Web Sites, Addison Wesley, September 10, 2002, 0-201-84454-0, pages 464

Un autre problème posé par le caractère fortement distribué du web est celui des scripts clients sur lesquels le développeur n'a aucune forme de contrôle. Ainsi si le client désactive l'interprétation du Javascript ou du Vbscript sur son browser le développeur est obligé d'implémenter un deuxième mécanisme de validation au niveau du serveur.

Pour éviter les va et vient inutiles entre le browser et l'application, le développeur peut toujours insérer dans son code des instructions pour vérifier si Javascript ou Vbscript sont activés sur le poste du client[168]. Dans l'affirmative, une double validation au niveau du serveur ne ferait qu'accroître le temps de réponse et la consommation des ressources.

Habituellement, les scripts et les messages de validation sont générés dynamiquement à partir d'un fichier de ressources partagées comme celui utilisé par Struts (Resource Bundle) ou codés en dur manuellement dans la vue. La première approche facilite la maintenance du code mais elle a des retombées non négligeables sur les performances parce qu'elle fait appel à des opérations complexes comme :

- Le parsing des fichiers de ressources[169] pour charger les algorithmes et les paramètres de validation dans des objets Java.
- L'interprétation des formules de l'Expression Language. Notons cependant que ces formules peuvent parfois se révéler plus performantes que les algorithmes de validation basés sur les boucles et les switchs (surtout lorsque les validations déclenchent des itérations dans les éléments d'une chaîne de caractères).

Pour sa part, la deuxième approche consomme moins de ressources parce que le serveur traite les scripts client inclus dans les JSP comme du code statique. Contrairement à la première approche le script de validation n'est pas construit de toutes pièces à partir du fichier des ressources. Il est renvoyé au browser aussitôt qu'il est détecté par le serveur web.

NB : Il est important de noter que la mixture de plusieurs langages de scripts dans une même page est une cause de ralentissement de l'interprétation au niveau du browser.
On constate encore une fois que les besoins en performance et les besoins en maintenance ne vont pas toujours de pair.

[168] Cette information est incluse par le browser dans l'entête du paquet http envoyé au serveur.
[169] Formatés souvent en XML

Face à ce dilemme il faut essayer de trouver une solution de compromis en gardant à l'esprit que la performance n'est pas seulement une affaire de temps de réponse mais aussi une affaire de coûts.

4.2.2 Anti-patterns et patterns de la couche de contrôle

Dans l'architecture Model 2 le contrôleur joue un rôle charnière entre la couche de présentation et l'ensemble des composants qui encapsulent la logique d'affaires de l'application. C'est un passage obligé par lequel doivent transiter toutes les requêtes avant d'être traitées. S'il est mal conçu il peut devenir une véritable source de congestion. Par contre si son design respecte les patterns et les meilleures pratiques de l'industrie, il peut contribuer efficacement à l'amélioration des performances de l'application comme nous le verrons dans le reste de cette section.

4.2.2.1 Quel objet utiliser pour implémenter le contrôleur ?

Utiliser une servlet et des filtres pour implémenter le contrôleur est un bon choix technologique parce que le conteneur ne crée qu'une seule instance de cette servlet pour gérer toutes les requêtes entrantes vers l'application. Celles-ci sont, en fait, traitées par des threads crées à l'intérieur du processus attribué à la servlet. Il en résulte des économies substantielles sur les ressources et des performances supérieures à celles des technologies CGI.

Les JSP pour leur part ne sont que des abstractions des servlets. Comme nous l'avons expliqué dans la partie 2 le conteneur les convertit, en arrière plan, en servlets via un mécanisme de translation. Cela les rend moins performantes que les servlets et donc moins qualifiées pour occuper le rôle de contrôleur. Même conclusion pour les EJB qui sont très gourmands en ressources. De plus ils ne peuvent pas être appelés à partir d'un browser via le protocole http (en fait l'invocation des EJB se fait via des protocoles synchrones comme RMI-IIOP ou encore des services de messagerie asynchrone comme JMS).

4.2.2.2 Stratégies pour la gestion du cache

Parce qu'il est récipiendaire de toutes les requêtes entrantes vers l'application, le contrôleur est un très bon emplacement pour implémenter la gestion du cache.

Sa position intermédiaire dans l'architecture lui permet de connaître les objets fréquemment utilisés par les clients et le caractère plus ou moins volatile de leurs états.

Le rôle d'un bon contrôleur ne se limite pas à l'identification des objets pouvant être mis en cache. Il lui revient également de mettre en œuvre une stratégie efficace pour rafraîchir, alimenter et libérer la mémoire cache de manière à :

- Eviter les fuites de mémoire[170] (memory leaks)
- Limiter la génération dynamique des pages.
- Limiter les accès à la base de données.

Les stratégies conventionnelles de rafraîchissement du cache reposent sur l'échange de messages synchrones entre le contrôleur et la source des données mises en cache. Dans ce modèle le contrôleur envoie périodiquement des messages à la source pour vérifier que les objets mis en cache n'ont pas changé d'état.

Vu que les messages envoyés sont synchrones le contrôleur reste en instance jusqu'à ce que la source lui réponde. A moins de ne lui attribuer un processus dédié cette opération peut bloquer le contrôleur et condamner le fonctionnement total de l'application. L'opération épuise aussi la source qui doit sacrifier une partie de ses ressources[171] pour répondre au contrôleur même si les objets mis en cache n'ont pas changé d'état.

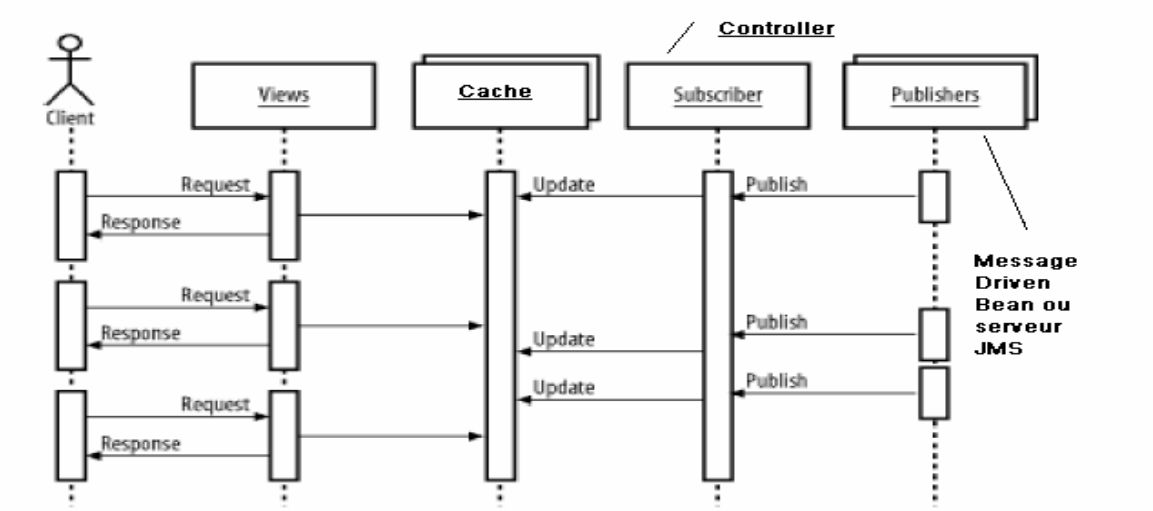
Pour de meilleures performances, c'est la source qui doit avertir la destination lorsque l'état des objets change. L'inversion du sens de la communication soulage, en effet, le réseau et diminue les risques de blocage au niveau du contrôleur. Dans la section 4.2.1.1 nous n'avons pas pu implémenter ce mode de communication inversé au niveau la couche de présentation parce qu'il n'est pas supporté par les browsers. Cependant, au niveau de la couche de contrôle il y a techniquement plusieurs possibilités.

L'avènement de l'API JMS et des messages driven beans a solidifié le rôle joué par les messages asynchrones dans la gestion du cache. Grâce à ces messages le contrôleur est exempté d'une partie de sa charge de travail et jouie d'un meilleur taux de disponibilité.

[170] Contrairement à une croyance largement répandue le garbage collector ne prévient pas toujours les fuites de mémoire.
[171] Mémoire, processeur, temps de réponse …

Un peu comme dans un serveur de nouvelles le contrôleur s'inscrit auprès d'un message driven bean ou d'un serveur de messages JMS qui doivent le notifier à chaque fois que l'état des objets mis en cache change (subscribe/publish). Dès réception de cette notification le contrôleur met à jour les données conservées dans le cache de façon à refléter les changements d'états récemment survenus (cf. figure 35).

Figure 35: Rôle joué par les messages asynchrones dans la gestion du cache[172]

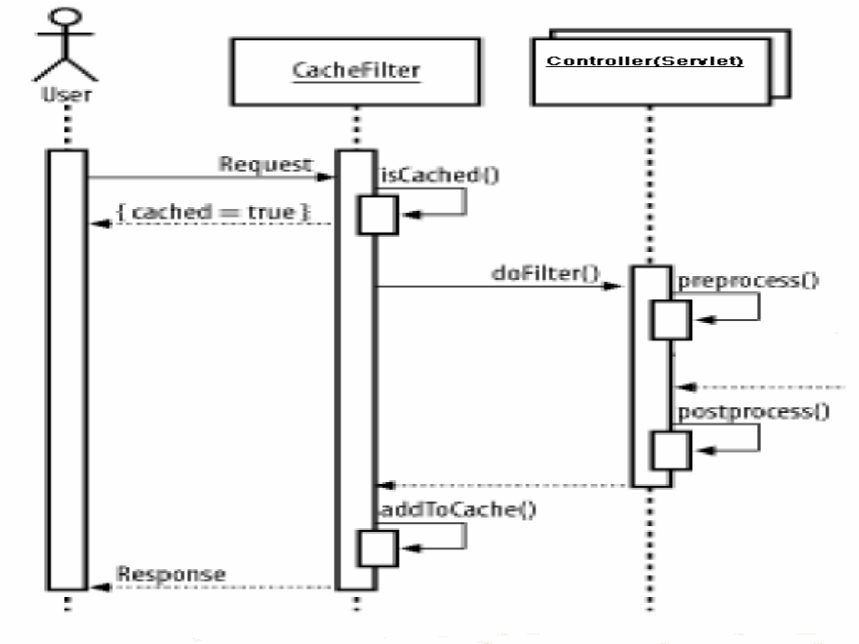


Dans les applications J2EE la stratégie de gestion de cache est souvent codée à l'intérieur des filtres (servlet filters) qui comme nous l'avons vu dans la partie 2 interceptent toutes les requêtes /réponses en amont/aval du contrôleur/ de la vue. En amont les requêtes portant sur des données déjà mises en cache sont interceptées par le(s) filtre(s) qui les empêche(nt) d'arriver jusqu'au contrôleur (servlet). De cette façon ces requêtes reçoivent des réponses plus rapidement. Les requêtes destinées aux données non placées dans le cache sont relayées vers le contrôleur mais en aval les réponses y afférentes sont à leur tour mises en cache comme le montre la figure 36.

☞ NB : le cas échéant, le filtre qui gère le cache doit être placé après celui qui gère la sécurité pour empêcher des personnes non autorisées d'accéder à des données confidentielles.

[172] Adaptation d'un digramme de séquence publié dans J2EE Design Patterns, William Crawford, Jonathan Kaplan, O'Reilly, September 2003, ISBN: 0-596-00427-3, pages 368

Figure 36: Gestion de cache au moyen d'un Servlet Filter[173]



Les objets sont mis en cache dans la mémoire vive du serveur, sur son disque dur ou encore dans la mémoire cache du browser. Chacun de ces récipients présente les avantages et les inconvénients énumérés ci-dessous :

Mémoire vive du serveur : ce récipient est rapide d'accès mais il ne permet de stocker qu'une quantité limitée de données. Pour que cette mémoire soit gérée rationnellement il faut utiliser des algorithmes qui éliminent (flush) rapidement les données superflues et qui mettent en cache celles ayant une très forte probabilité d'être demandées par les prochaines requêtes[174]. Parmi ces algorithmes on peut citer :

- **Least Frequently Used (LFU):** comme son nom l'indique cet algorithme élimine du cache les données non récemment demandées. Dans les programmes il est mis en œuvre par le biais d'une liste qui place au premier rang toute donnée nouvellement demandée. Au fur et à mesure de l'arrivée des requêtes, les données les moins demandées sont surclassées jusqu'à ce qu'elles soient définitivement éliminées de la liste.

[173] Adaptation d'un digramme de séquence publié dans J2EE Design Patterns, William Crawford, Jonathan Kaplan, O'Reilly, September 2003, ISBN: 0-596-00427-3, pages 368
[174] Par exemple les données relatives aux produits vedettes.

- **First In, First Out (FIFO):** très rarement utilisé cet algorithme élimine du cache l'élément le plus ancien de la liste (premier entré, premier sorti).
- **Last in, First Out (LIFO) :** la dernière donnée ajoutée au cache est éliminée en premier.

  NB : cette liste n'est pas limitative. Pour connaître les autres algorithmes nous recommandons la lecture des deux livres suivants :

  - Duane Wessels, Web Caching, O'Reilly, First Edition June 2001, ISBN: 1-56592-536-X, pages 318
  - Michael Rabinovich and Oliver Spatscheck, Web Caching and Replication, Addison Wesley, December 21, 2001, 0-201-61570-3, pages 400

Disque dur du serveur: ce récipient permet de stocker plus de données que la mémoire vive mais cela se fait au détriment du temps de réponse.

Mémoire cache du browser : comparé aux autres ce récipient offre une bonne capacité de stockage et le meilleur temps de réponse mais il présente aussi des inconvénients que nous avons déjà décrits à la fin de la section 4.2.1.1. Nous verrons dans la partie 5 comment modifier l'entête des paquets http pour ordonner au browser de mettre en cache une partie ou la totalité d'une page Web.

Remarque 1 : pour connaître les objets qui peuvent être mis en cache dans une application de commerce électronique, il faut se rendre aux sections 2.2.3 et 3.1.

Remarque 2: les données peuvent être également mises en cache dans d'autres récipients comme les conteneurs, les routeurs ou encore les serveurs proxy. Ces récipients n'ont pas été traités parce qu'ils dépassent le cadre de cet essai (cf. section 1.3).

4.2.2.3 Compression des paquets http au niveau des filtres

La compression est une technique communément utilisée pour réduire la taille des données envoyées au client. Elle permet de les transporter plus rapidement sur le réseau mais elle engendre un temps de réponse supplémentaire attribuable aux algorithmes de compression/décompression.

Les paquets http sont de simples fichiers ASCII avec plusieurs balises redondantes qui facilitent énormément leur compression[175]. Par exemple sur le site de PopularMechanics (PopularMechanics.com) l'utilisation de GZIP a permis de réduire de 84.3% la taille originale de la page d'accueil.

Tableau 16: Compression de la taille de la page d'accueil du site http://www.popularmechanics.com/

	PopularMechanics.com (HTML)	GZIP -9	Percentage Savings
Original	138,548	21,743	84.3%

Avant de compresser les données le filtre doit analyser l'entête des paquets http reçus pour vérifier si le format et l'algorithme de compression utilisés sont supportés par le browser.

```
GET /products/firefox/start/ HTTP/1.1
Host: www.mozilla.org
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7)
Gecko/20040614 Firefox/0.9
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/pla
in;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5

Accept-Encoding: gzip,deflate

Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7

Content-Encoding: gzip

Keep-Alive: 300
Connection: keep-alive
```

Dans l'entête ci-dessus le champ Content-Encoding indique que le browser (Mozilla) supporte le format GZIP. En fait depuis 1998 la plupart des browsers ont été dotés des fonctionnalités requises pour supporter la compression des paquets http avec des algorithmes du domaine public comme GZIP[176]. Les recherches que nous avons effectuées nous ont cependant révélé certaines incompatibilités entre GZIP et Microsoft Internet Explorer. La base des connaissances (Knowledge Base) de Microsoft contient de plus amples détails à ce sujet. http://support.microsoft.com/kb/321722

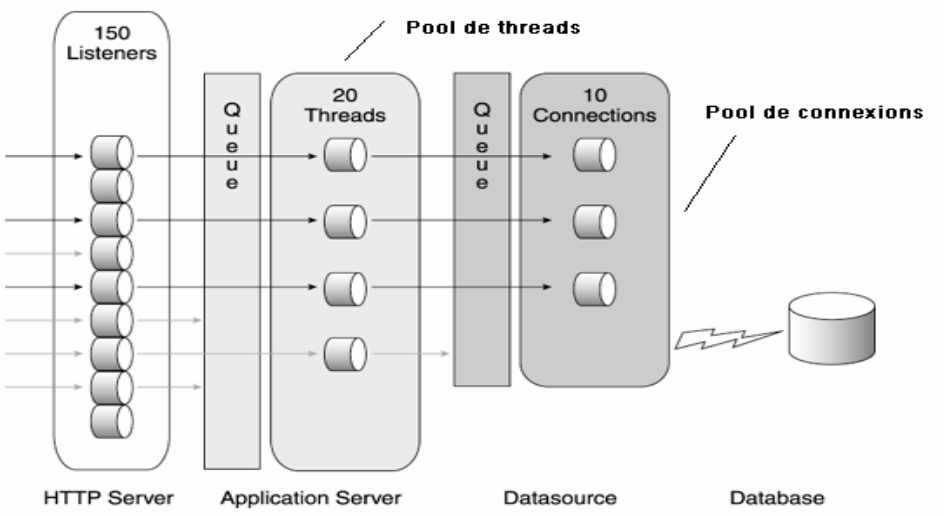
[175] Source: Speed Up Your Site: Web Site Optimization, New Riders Publishing, January 17, 2003, ISBN: 0-7357-1324-3
[176] GZIP par exemple

4.2.2.4 Pooling des ressources

Un pool est un ensemble de ressources réutilisables. Il permet de partager des objets déjà crées entre plusieurs requêtes de manière à éviter les coûts liés à de nouvelles instanciations. Dans les applications J2EE on retrouve les pools à plusieurs niveaux :

Au niveau des servlets : une servlet associe un thread à chaque requête nouvellement reçue mais cela ne veut pas dire qu'elle crée autant de threads qu'il y a de requêtes entrantes. Pour ne pas épuiser les ressources du serveur, le conteneur crée un nombre limité de threads et les place dans un pool. Parce que le nombre des requêtes entrantes est largement supérieur à celui des threads placés dans le pool, il doit mettre les requêtes excédentaires dans une file d'attente jusqu'à ce qu'un nouveau thread soit libéré. A l'issue du traitement d'une requête le thread qui lui a été assigné n'est pas détruit. Il est gardé dans le pool est réinitialisé pour qu'une nouvelle requête puisse s'en servir.

Figure 37: Pool de threads maintenu par un serveur d'application[177]

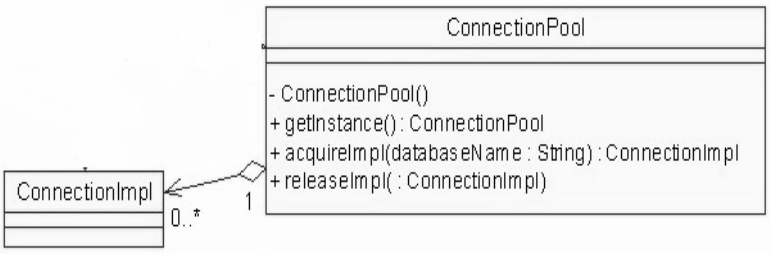


Au niveau des bases de données : la connexion à une base de données est une opération très coûteuse en mémoire et en processeur. A l'instar des threads un développeur ne peut pas se permettre de créer autant de connexions qu'il y a de requêtes entrantes. De la même manière, il ne peut pas partager une seule connexion entre toutes les requêtes au risque de condamner les traitements parallèles et les accès concurrentiels à la base de données.

[177] Source : Adaptation d'une figure publiée dans: Threading/queuing "funnel." From IBM Software Group Services Performance Workshop presented by Stacy Joines, 2001, Hursley, U.K. © IBM Corp. 2001

Comme le montre le diagramme de classe de la figure 38 le ConnectionPool regroupe un nombre limité de connexions déjà créées (ConnectionImpl). Lorsqu'une nouvelle requête arrive le ConnectionPool appelle la méthode acquireImpl() qui lui retourne une connexion prête à l'emploi. Une fois le traitement de la requête terminé le ConnectionPool appelle la méthode releaseImpl() pour libérer la connexion et permettre à de nouvelles requêtes de la réutiliser.

Figure 38: Diagramme de classe du ConnectionPool[178]

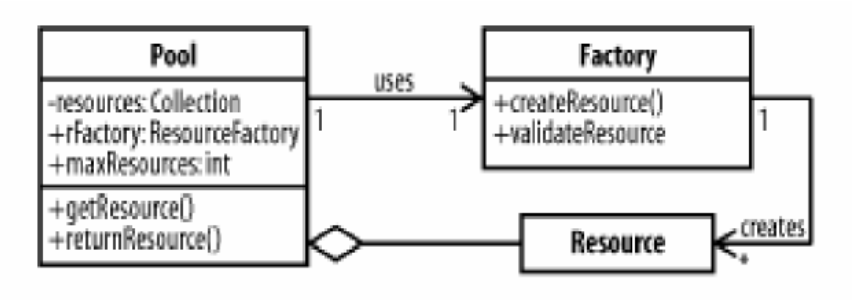


Au niveau de la couche métier : à la différence des stateful session beans les stateless session beans ne conservent pas de données personnalisées dans les objets de session. Cette propriété facilite énormément leur conservation dans des pools. Lorsque le conteneur d'EJB reçoit une requête, il n'a pas à se soucier des données personnalisées du client qui l'a envoyée. Parce qu'elles sont identiques, il peut lui retourner n'importe quelle instance de l'EJB pool. Ce raisonnement s'applique par transitivité aux entity beans et aux message driven beans mais pas aux stateful session beans parce qu'ils conservent pendant la session les données personnalisées de chaque client. En fait, sans réinitialisation, une réutilisation des stateful session beans peut affecter la confidentialité et l'intégrité des données (par exemple si un client réutilise le panier d'achat rempli par autre client).

Cela dit il n'y a pas de restrictions quant aux types d'objets qu'on peut placer dans un pool. La reutilisabilité est pratiquement la seule condition qu'ils doivent vérifier. Comme le montre la figure 39 un resource pool peut être par abstraction généralisé à tous les types d'objets via une classe intermédiaire (Factory) qui permet la factorisation des méthodes et des propriétés du pool.

[178] Adaptation d'un diagramme de classe publié à l'adresse: http://www.developer.com/java/other/article.php/626291 , Wiebe de Jong, Implement a JDBC Connection Pool via the Object Pool Pattern

Figure 39: Factorisation des ressources d'un pool[179]



La méthode createResource() de la classe Factory permet de créer une nouvelle instance de l'objet qu'on souhaite placer dans le pool. Avant de pouvoir réutiliser cette instance le pool doit appeler la méthode validateResource( ) pour la réinitialiser.

4.2.3 Anti-patterns et patterns de la couche d'affaires

4.2.3.1 A propos de la performance des EJB

Dans Java tout est objet mais tout n'est pas EJB. Tel est le jeu de mots que nous avons choisi pour alimenter le débat sempiternel sur l'utilisation des Enterprise Java Beans dans les applications J2EE. L'engouement pour ces composants est tel que certains architectes n'hésitent pas à les implémenter dans leurs applications quelques soient les contraintes et le contexte d'utilisation. Un rapport du Gartner Group estime d'ailleurs que les entreprises ont mondialement dépensé un peu plus d'un billion de dollars US pour intégrer les EJB à leurs applications.

A dire vrai, les EJB sont des composants robustes qui facilitent le déploiement des applications dans les environnements fortement distribués. Ils sont également un bon choix technologique lorsqu'on souhaite construire des middletiers ou des passerelles vers les systèmes hérités[180] (vieilles applications écrites en Cobol et exécutées sur un gros système). Le revers de la médaille est que les EJB sont des composants très complexes qui nécessitent une main d'œuvre très qualifiée (et donc très chère).

La couche de complexité technique ajoutée par les EJB à l'architecture MVC n'est pas sans effet sur les performances de l'application. Comme nous l'avons vu tout au long de cet essai les invocations multiples de leurs méthodes via le protocole RMI-IIOP sont extrêmement pénalisantes pour le temps de réponse.

[179] Source: William Crawford, Jonathan Kaplan, J2EE Design Patterns, O'Reilly, September 2003, ISBN: 0-596-00427-3, 368
[180] Traduction littérale de legacy systems

De plus des opérations comme l'instanciation, l'activation ou la passivation des beans sont gourmandes en processeur et en mémoire. Notons par ailleurs que les EJB ne peuvent pas être exécutés sur un conteneur de servlets comme Tomcat. Ils ont besoin d'un conteneur spécial comme celui d'IBM (Websphere) ou de BEA Systems (Weblogic). La politique de tarification de ces sociétés est très agressive parce qu'ils vendent leurs produits dans un seul bloc monolithique avec aucune possibilité de découplage entre la couche qui gère les JSP/Servlets et celle qui gère les EJB. A moins d'envisager l'utilisation d'un conteneur gratuit[181] comme JBOSS il faut donc évaluer les pour et les contre des EJB aussi bien en termes de complexité technique qu'en termes de coûts (rappelons le encore une fois la performance n'est pas seulement une affaire de débit et de temps de réponse mais également une affaire de coûts).

En pratique on peut développer avec les JSP, les servlets et les Java Beans tout ce qu'on peut développer avec les EJB. Soulignons à ce sujet que le framework Struts ne contient pas d'EJB et cela n'affecte ni sa popularité ni son applicabilité à plusieurs types de sites. Dans bien des cas les EJB peuvent s'avérer surdimensionnés par rapport aux besoins d'affaires. Ils sont certainement un bon choix technologique pour l'implémentation des middletiers dans des environnements fortement distribués mais ils peuvent devenir de véritables goulots d'étranglement dans les applications simples et centralisées (exemple un site de commerce électronique d'une PME dans lequel il n'y a aucune interaction avec un système d'héritage[182]).

Cet extrait du livre Mastering Enterprise Java Beans nous paraît très pertinent. Il résume les arguments qui plaident en faveur de l'intégration des EJB aux applications ainsi que les situations dans lesquelles il faut éviter leur utilisation.

[181] Très souvent les licences sont gratuites mais il y a plusieurs coûts cachés liés à la formation, à la maintenance à l'ajout de modules supplémentaires…
[182] Legacy system

Deciding whether EJB is appropriate

Now that we've blown away the FUD (Fear, Uncertainty, and Doubt), here are the real reasons to use EJB over Java classes:

Your system is built faster and more reliably. EJB components benefit from declarative middleware, such as instance pooling, transactions, security, container-managed persistence, container-managed relationships, and data caching. If you used regular Java classes, you'd need to write this middleware yourself over time. Eventually you might find that you have your own middleware framework. That 'framework' is a fancy word for building your own home-grown application server. The framework needs to be tested, debugged, features need to be added. This is a non-trivial task indeed. Can you honestly state that your staff is capable of building a better application server than the market leaders who specialize in middleware?

It is easier to hire new employees. If you build your own custom framework using Java classes, then newhires need to be trained on this framework. If your framework is complex, then you can no longer look for "EJB" on a resume when hiring a developer and expect them to be productive on your system.

You benefit from the best practices the world is building around EJB. You can figure out how to optimize your system by reading articles on the Internet, or picking up a book on EJB best practices, such as Floyd Marinescu's "EJB Design Patterns" book. This global knowledge base is not at your disposal with a proprietary Java class framework.

You can have different user interfaces. You can reuse the same EJB component layer for a thick client as well as a Servlet/JSP client. With Java classes, you cannot achieve this because Java classes are not remotely accessible. If you wrapped those Java classes in RMI objects, you'd need to invent your own load-balancing, instance pooling, and fail-over.

You can work with industry-standard tools to rapidly develop your system. While in the short run you may think that Java classes are going to make it faster to develop than writing all those files that comprise an EJB component, in reality there are many tools around that help streamline the EJB development process. There are command-line tools that generate the files you need, there are IDEs that help you build EJB components, and there are UML editors that help you generate EJB components from UML diagrams. See for more.

You can separate your web tier and application server. If you require your business logic to be protected by a firewall, then you can deploy the web server and application server on separate machines and stick a firewall in the middle.

And here are the real reasons not to use EJB:

You can't deal with the limitations of EJB. Examples include threading, static variables, and native code. Most companies can deal with this, because there are good reasons why the restrictions exist. But if (for example) you need to have a multi-threaded engine, and you can't deal with the EJB paradigm of load-balancing across single-threaded instances, then EJB is not a good fit for you. EJB is a square peg--don't try to stick it into a round hole.

You have existing skillsets or investments in a working technology. For example, if your developers are proficient in CORBA, then great--why not stick with it? As an anecdote, The Middleware Company consulted with one of our clients who wrote a CORBA application that assisted with mapping the human genome. It worked well with CORBA, and they had no major complaints, and so we recommended they stick with CORBA and avoid the EJB hype.

Your application is a big GUI to a database. If you are just a big GUI to a database--heavy on data logic but no business logic--you could achieve a deployment easily using JSPs with tag libraries connecting to a database via JDBC.

Your application is simple. If you are prototyping, building a simple system, or developing a one-off application that will not evolve over time, EJB may be overkill.

Source: Ed Roman, Scott Ambler, Tyler Jewell, Ed Roman, Tyler Jewell, Floyd Marinescu, Mastering Enterprise JavaBeans (2nd Edition), Wiley, ISBN: 0471417114, pages 672

4.2.3.2 Gestion des sessions au niveau de la couche d'affaires

Le conteneur d'EJB ne gère pas les stateful session beans de la même manière que les stateless session beans. Les stateful beans sont alloués au client tant et aussi longtemps que sa session est active. Le maintien des données de la session demande plus de ressources que les stateless beans et limite les possibilités de réutilisation à travers les pools (cf. section 4.2.2.4).

Parce que les stateless session beans sont des composants robustes, certains développeurs ont pris l'habitude de les utiliser pour gérer les données des sessions en lieu et place des objets HttpSession que nous avons étudiés dans la section 4.2.1.2. Pour notre part nous désapprouvons cette technique parce qu'elle re-localise la gestion des sessions au niveau de la couche d'affaires. Cela augmente non seulement le trafic des données sur le réseau mais également la latence et la consommation des ressources (passage par RMI, activation, passivation des états…).

Comme nous l'avons vu dans la section 4.2.1.2, la gestion des sessions au niveau du contrôleur permet d'optimiser les performances de l'application. De plus les objets HttpSession consomment moins de ressources que les EJB. Pour cette raison il ne faut utiliser les stateful beans que si le client accède à l'application via une interface non webifiée (par exemple si l'administrateur de la table des produits accède aux menus d'administration via une interface SWING et un protocole autre que http).

4.2.3.3 Les apports des communications asynchrones

Le rôle que peuvent jouer les messages asynchrones dans l'amélioration des performances est souvent sous estimé. En effet sur les sites de commerce électronique il existe plusieurs cas d'utilisation qui ne requièrent pas une réponse immédiate de la part de l'application. Par exemple lorsqu'un client passe à la caisse et valide sa commande l'application n'est pas supposée lui donner un feed-back instantané. Plutôt que d'utiliser des invocations synchrones qui sont bloquantes pour le client et consommatrice de ressources, le développeur peut transporter les données de la commande via des messages asynchrones et les placer dans une file d'attente (topic ou queue) en attendant que le département chargé des livraisons les valide. De cette façon l'injection de la commande dans la base de données est différée et le nombre d'insert, d'update et de connexions ouvertes est réduit grâce au regroupement des commandes.

Même si la commande n'a pas encore été traitée on peut en retour envoyer au client un accusé de réception par le biais d'une page web ou d'un courriel.

Les messages driven beans sont particulièrement adaptés pour mettre en œuvre ce genre de workflow. Nous recommandons vivement leur utilisation dans les applications B2B parce que comme nous l'avons vu dans la partie 3 la plupart de leurs cas d'utilisation ne sont pas interactifs.

4.2.3.4 Transformation des invocations distantes en appels locaux

L'invocation des méthodes distantes d'un EJB (remote methods) est un processus complexe qui fait appel à des opérations pénalisantes pour le temps de réponse et le débit de l'application. Cette invocation se fait typiquement via le protocole RMI-IIOP. Pour bien comprendre ses retombées sur la performance il est important d'expliquer son mode de fonctionnement.

RMI (Remote Method Invocation) est une API qui permet à un client de manipuler des objets distants c'est-à-dire des objets placés sur une JVM différente de celle sur laquelle il se trouve. Parce que l'API cache au client toute la complexité technique nécessaire pour invoquer les objets distants, leur manipulation se fait comme s'ils se trouvaient sur la même JVM.

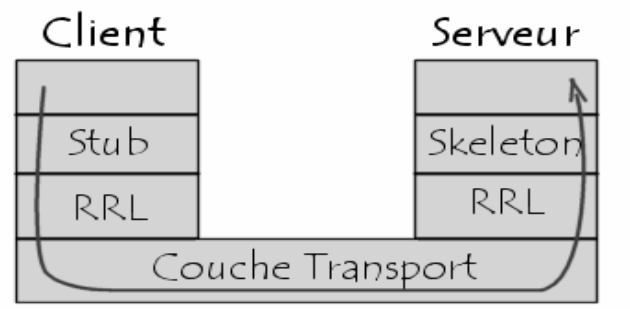
Initialement les connexions et les transferts de données dans RMI étaient effectués via TCP/IP grâce à un protocole propriétaire (JRMP, Java Remote Method Protocol). A partir de la version 1.3 de Java 2 ce protocole a été remplacé par l'Internet Inter-Orb Protocol(IIOP) pour permettre aux EJB de communiquer avec les applications CORBA.

Concrètement lorsqu'un client invoque la méthode distante d'un EJB il fait en premier lieu appel à un objet proxy appelé stub. Ce dernier clone localement toutes les méthodes de l'objet distant. Il est capable via des lookup JNDI de trouver le chemin vers l'objet distant. Comme le montre la figure 40 la transmission de données se fait par le truchement d'une architecture en couches inspirée du modèle OSI.

- La couche de référence (RRL, Remote Reference Layer) fournit au stub les informations et les services nécessaires pour localiser l'EJB distant.
- Le skeleton se charge de déléguer les invocations reçues du stub vers les EJB distants.

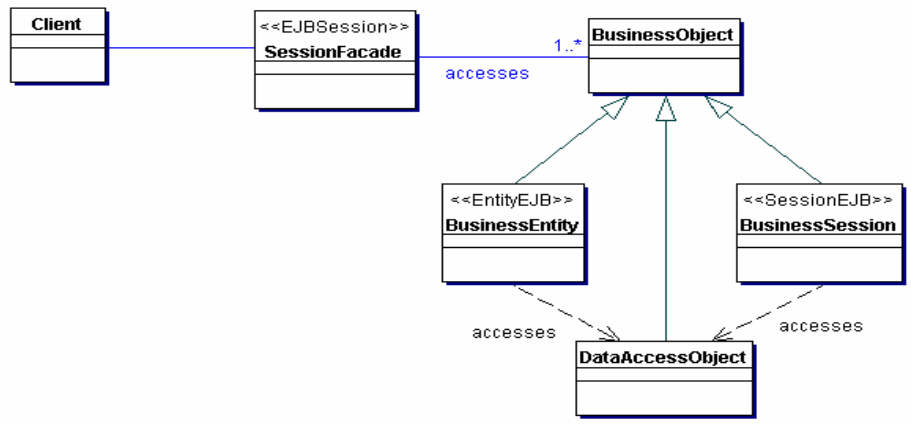
- La gestion des connexions et le transport des données se fait via le protocole TCP (Les packages java.net.Socket et java.net.SocketServer assurent implicitement cette fonction).

Figure 40: Modèle en couches RMI-IIOP[183]



Fort heureusement les EJB disposent de méthodes locales en plus de leurs méthodes distantes. Cette dualité permet de mettre en œuvre le pattern « session façade » grâce auquel les invocations distantes sont transformées en appels locaux. Ainsi au lieu d'invoquer directement les méthodes distantes de chaque EJB le client s'adresse à un objet intermédiaire placé **sous la même JVM**. Ce dernier se charge ensuite d'appeler les méthodes locales de l'EJB pour lui déléguer la requête reçue du client. L'objet intermédiaire est mis en ouvre à l'aide d'un stateless session bean. Il permet de réduire le nombre d'invocations EJB/EJB et clients/EJB (cf. figure 41).

Figure 41: diagramme de classe du session façade[184]



[183] Source: introduction à RMI, http://www.commentcamarche.net/rmi/rmiintro.php3
[184] Source : Core J2EE Patterns - Session Façade,
http://java.sun.com/blueprints/corej2eepatterns/Patterns/SessionFacade.html

Le session façade n'élimine pas les invocations RMI-IIOP. Il permet seulement de réduire leur nombre. Cela nous amène encore une fois à vanter les mérites des servlets en tant que composante charnière de l'architecture MVC. En effet si le développeur arrive à les substituer au session bean qui implémente la façade les invocations RMI-IIOP seront complètement éliminées du workflow.

4.2.3.5 BMP vs CMP

Les entity beans sont des EJB persistants qui sauvegardent leurs états dans la base de données à laquelle ils sont adossés. La logique qui permet de gérer cette persistance peut être placée au niveau du bean (Bean Managed Persistence ou BMP) et/ou au niveau du conteneur (Container Managed Persitence).

Lorsqu'elle est convenablement paramétrée, la persistance gérée par le conteneur est de loin plus performante que celle gérée au niveau du bean. En effet avec la BMP il faut exécuter deux instructions SQL pour charger les propriétés[185] d'une instance[186] d'entity bean (une première pour localiser l'instance via la clé primaire –finder method- et une deuxième pour charger ses propriétés –ejbLoad method-). Avec une collection de n instances d'entity beans, le nombre d'instructions à exécuter est de n+1. Par contre avec la CMP ces n+1 instructions individuelles sont remplacées par une seule instruction globale placée dans un fichier de description xml factorisé pour l'ensemble des beans.

4.2.3.6 Entity façade

La création d'autant d'entity beans qu'il y a de tables dans une base de données est une pratique courante qui est cautionnée par plusieurs outils de mapping entre les modèles objets et les modèles relationnels de données. Cette pratique est pénalisante pour les ressources parce qu'il faut associer une instance du bean à chaque ligne de la table, stocker les champs (colonnes) dans les propriétés de cette instance et gérer les relations entre les tables comme des relations entre les beans.

Le pattern Entity Façade permet de résoudre ce problème en compilant les données issues de plusieurs tables dans un nombre restreint d'entity beans composites.

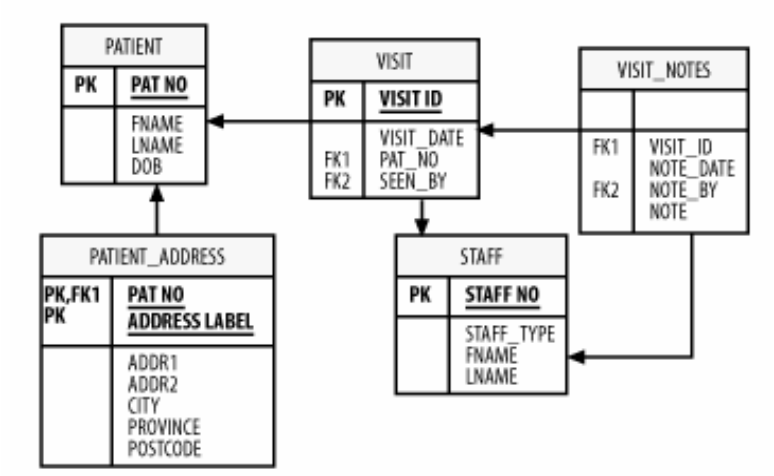
[185] Ces propriétés correspondent généralement aux colonnes d'une table
[186] Une instance correspond généralement à une ligne dans une table

Outre la diminution de la quantité de mémoire consommée, le pattern réduit le trafic sur le réseau, limite le nombre de messages échangés entre les différents beans et diminue le nombre d'appels à la base de données.

Pour étayer les affirmations énumérées ci-dessus nous nous permettons de reproduire un exemple donné dans le livre J2EE design patterns. La figure 42 représente le modèle relationnel d'une clinique médicale. Dans ce modèle normalisé un patient peut avoir une ou plusieurs adresses. De même une à plusieurs notes peuvent être associées aux différentes visites qu'il effectue. Nous supposons que le client effectue en moyenne 4 visites par an, que 10 notes sont en moyenne annotées à son dossier et que la clinique reçoit en moyenne 10.000 clients par an.

Si on crée autant d'EJB qu'il y a de tables dans le modèle relationnel, il faudra créer 40 instances de l'entity bean pour récupérer les notes associées au dossier de chaque client à partir de la base de données (soit 400.000 instances pour gérer tout le portefeuille de clients). Ces instanciations pléthoriques influent non seulement sur la mémoire consommée mais également sur le trafic et le temps nécessaire pour générer les vues.

Figure 42: Modèle relationnel de données d'une clinique médicale[187]



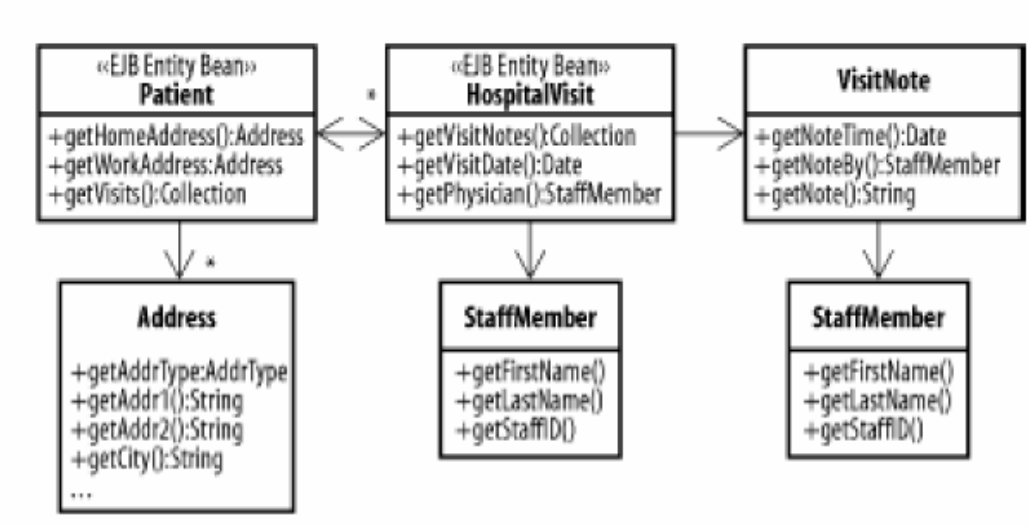
Pour remédier à ces problèmes le pattern Entity Facade propose de ne convertir en entity bean que les tables parent PATIENT et VISIT.

[187] Source: William Crawford, Jonathan Kaplan, J2EE Design Patterns, O'Reilly, September 2003, ISBN: 0-596-00427-3, 368

Les autres tables sont versées dans des collections et/ou de simples objets Java au moment du chargement du HospitalVisit et du Patient Beans (cf. figure 43).

Ce design rend difficile la gestion de la persistance au niveau du conteneur (CMP) mais il permet des économies substantielles sur la mémoire, le temps de réponse et le trafic.

Figure 43: Pattern Entity Facade[188]



[188] Source: Idem

5. Gestion de la performance dans la phase de codage

Dans cette partie il sera question :

- De relever les erreurs fréquemment commises lors du codage des objets qui rentrent habituellement dans la construction d'une application de commerce électronique (se rendre à la section 1.4 pour connaître la liste de ces objets).
- De proposer des solutions pour les éviter ou amenuiser leur impact.
- De proposer des solutions pour optimiser les performances de MySql.
- De faire le point sur les fonctionnalités phares de la version 5 de MySql et plus particulièrement celles qui contribuent à l'amélioration des performances.

5.1 Optimisation des principaux composants J2SE

5.1.1 Les chaînes de caractères

5.1.1.1 Concaténation des objets String

Un objet String est immuable (final). Cela signifie que lorsqu'on lui attribue une chaîne de caractère il n'est plus possible de le modifier. Cette caractéristique a d'importantes conséquences sur la performance des opérations de concaténation. En effet lorsqu'on demande à Java de réaliser une opération aussi simple que String p = a + b ; (a et b étant des Strings), le compilateur effectue en arrière plan une opération plus complexe qui est: String p = (new StringBuffer()).append(a).append(b).toString();

Dans cette opération masquée on peut constater que le compilateur a procédé à la création de deux objets supplémentaires en plus des objets p, a et b. Ces objets sont StringBuffer et l'objet retourné par la méthode toString(). Après l'opération le handle p ne pointe plus vers l'objet initial mais vers le nouvel objet retourné par cette méthode.

Les objets StringBuffer pour leur part sont modifiables. En d'autres termes, on peut leur ajouter ou leur soustraire des caractères sans la création de nouveaux objets. Pour effectuer une opération comme String p = a + b ils sont plus performants que les objets String parce qu'ils occupent moins de mémoire et génèrent moins de charge de travail pour le garbage collector (qui doit intervenir afin d'éliminer de la mémoire les objets non pointés[189]).

[189] C'est à dire ceux qui ne sont plus référencés par des handles.

Pour bien comprendre l'impact des opérations de concaténation sur les performances nous proposons la comparaison des deux programmes suivants :

Tableau 17: Concaténation String vs concaténation StringBuffer

Concaténation avec String	Concaténation avec StringBuffer

String s = new String();
long start = System.currentTimeMillis(); //
<+++ Start timing
for (int i = 0; i < 10000; i++) {
s += "a";
}
long stop = System.currentTimeMillis(); //
<+++ Stop timing
	StringBuffer s = new StringBuffer();
	long start = System.currentTimeMillis(); //
	<+++ Start timing
	
	for (int i = 0; i < 10000; i++) {
	s.append("a");
	}
	long stop = System.currentTimeMillis(); //
	<+++ Stop timing

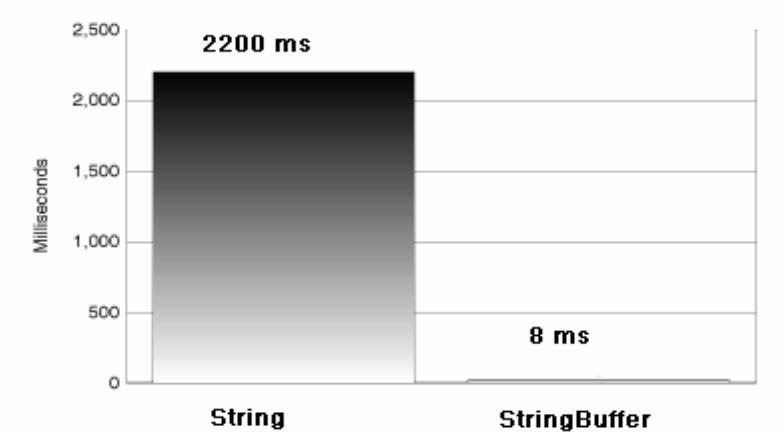
Comme on peut le voir sur la figure 44, la première technique augmente démesurément le temps de réponse parce que le compilateur doit créer à chaque fois de nouveaux objets pour effectuer la concaténation. En fait en arrière plan l'équivalent de l'opération s += "a"; est s = (new StringBuffer()).append(s).append("a").toString();

La deuxième technique pour sa part ne fait que placer de façon itérative le caractère 'a' dans le buffer interne du StringBuffer. Aucun nouvel objet n'est crée tant que le buffer n'est pas entièrement rempli. De plus le handle s pointe toujours vers le même objet (de ce fait il y aura moins de charge de travail pour le garbage collector).

Il est important de noter que le compilateur n'applique pas la logique présentée ci-dessus à tous les types d'opérations. En effet si on lui demande de réaliser le programme infra il ne créera pas d'objets supplémentaires comme il l'a fait pour String p = a + b ; ou s += "a";

```
for (int i = 0; i < 100000; i++) {

String s = "Hello" + "Steely" + "Dan";

}
```

Figure 44: Comparaison de la concaténation String vs concaténation StringBuffer[190]



Pour le compilateur cette boucle est équivalente à la boucle :

```
__temp = " Hello Steely Dan ";

for (int i = 0; i < 100000; i++) {
String s = _temp;
}
```

De même les trois opérations du tableau suivant sont équivalentes en termes de performance.

Tableau 18: Trois opérations qui ont le même impact sur les performances[191]

Opération 1 équivaut à	Opération 2 équivaut à	Opération 3
String s = "Hello" + "Steely" + "Dan";	String s = "Hello Steely Dan";	private static final String hello = "Hello "; private static final String steely = " Steely "; private static final String dan = " Dan "; String s = hello + steely + dan;

En revanche les deux opérations suivantes n'affectent pas les performances de la même manière.

[190] Source: adaptation d'une figure publiée dans : Doy Bulka, Java™ Performance and Scalability Volume 1: Server-Side Programming Techniques, Addison Wesley, June 05, 2000, 0-201-70429-3, pages 320
[191] Source : inspiré d'un algorithme paru dans le même livre

Tableau 19: Deux concaténations qui ont le même résultats mais pas les mêmes impacts sur les performances

Opération 3 plus performante que	Opération 4

for (int i = 0; i < 100000; i++) {
String s = "Hello " + "Steely " + "Dan";
}
	for (int i = 0; i < 100000; i++) {
	String s = "Hello ";
	s += "Steely ";
	s += "Dan";
	}

Résultat : moins d'une milliseconde	Résultat : 1370 milliseconds

Il ressort de la comparaison de ces différentes formules que la performance des opérations de concaténation est tributaire de :

- La variabilité du résultat en termes de taille et de contenu (en effet dans l'opération 3 le contenu et la taille de String s est toujours le même quelque soit la valeur de i. En revanche dans l'opération 4 String s change de taille et de contenu au fur et à mesure de la progression de la boucle).
- Le moment où la concaténation se produit : dans une opération comme String s = "Hello " + "Steely " + "Dan"; la concaténation est résolue[192] en mode compilation parce que les chaînes concaténées sont fixes. Le temps de réponse est par conséquent inférieur à celui des opérations portant sur des chaînes à contenu variable traitées en mode exécution.

5.1.1.2 Comparaison des chaînes de caractères

Pour comprendre l'impact des méthodes de la famille equals() sur la performance nous allons procéder à la comparaison des 3 classes suivantes :

Clase 1 :

```
public class Comparaison1 {          // Comparer deux chaînes identiques
public static void main(String args[]) {
String a = "HelloWorld";
String b = "HelloWorld";
long debut = System.currentTimeMillis();  // compteur
for (int i = 0; i < 10000000; i++) {
```

[192] Cela signifie que le compilateur élimine tout simplement les signes +

```
a.equals(b);
}
long fin = System.currentTimeMillis();
System.out.println( " time = " + (fin - debut) );
}
```

Classe 2 :

```
public class Comparaison2 {          // Comparer deux chaînes différentes
public static void main(String args[]) {
String a = "HelloWorld";
String b = "HelLoWorld";

long debut = System.currentTimeMillis();  // compteur
for (int i = 0; i < 10000000; i++) {
a.equals(b);
}
long fin = System.currentTimeMillis();
System.out.println( " time = " + (fin - debut) );
}
```

Classe 3 :

```
public class Comparaison3 {          // Comparer deux chaînes différentes
public static void main(String args[]) {
String a = "HelloWorld";
String b = "3HelloWorld";

long debut = System.currentTimeMillis();  // compteur
for (int i = 0; i < 10000000; i++) {
a.equals(b);}
long fin = System.currentTimeMillis();
System.out.println( " time = " + (fin - debut) );}
```

Parmi ces trois classes c'est Comparaison1 qui a mis le moins de temps à s'exécuter (environ 110 millisecondes), suivie de Comparaison3 (environ 600 millisecondes) suivie de Comparaison2 (environ 1800 millisecondes).

C'est donc les chaînes identiques qui donnent la meilleure performance avec la méthode equals(). Avec la méthode equalsIgnoreCase() c'est l'inverse qui se produit : plus il y a de différences entre les chaînes comparées et moins le programme met du temps à s'exécuter.

5.1.1.3 StringTokenizer

StringTokenizer est typiquement utilisé pour le parsing des chaînes de caractères. S'il est utilisé à outrance, il peut devenir une véritable source de congestion. Cette section propose un algorithme de substitution qui fait appel aux méthodes indexOf() et substring() de la classe String.

Classe 1 : parsing d'une chaîne de caractères avec le StringTokenizer

```
private static void jdkTokenizer(String s) {
String sub = null;
StringTokenizer st = new StringTokenizer(s,",");

try {
while ( (sub = (String) st.nextToken()) != null) {
// Normally, you would compute something useful here...
// ...but not now as we are trying to isolate the ...
// ...performance cost of a StringTokenizer}
}
catch (NoSuchElementException e ) {}
}
```

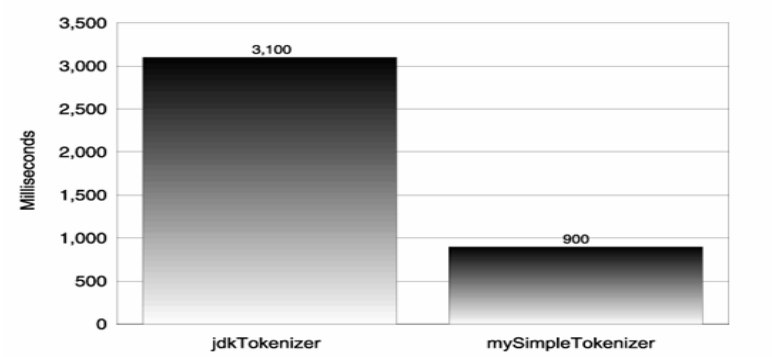
Classe 2 : parsing utilisant les méthodes indexOf() et substring() de la classe String

```
private static void mySimpleTokenizer(String s) {
String sub = null;
int i = 0;
int j = s.indexOf(",");
while ( j >= 0) {
```

```
sub = s.substring(i,j);
// Normally, you would compute something here
i = j+1;              // skip over comma we already found
j = s.indexOf(",",i);   // find next comma
}
sub = s.substring(i);       // Don't forget the last substring}
```

Les résultats de ces deux classes sont donnés par la figure 45.

Figure 45: Performance du StringTokenizer comparée à celle d'un Tokenizer qui utilise les méthodes indexOf() et substring()de la classe String[193]



5.1.1.4 Tri des chaînes internationalisées

Le tri des chaînes de caractères internationalisées est une opération complexe qui nécessite l'adaptation des algorithmes de tri à l'alphabet et aux localités utilisés. La classe java.text.Collator offre les méthodes et les propriétés requises pour effectuer ce genre de tri mais son niveau de performance laisse à désirer. La classe java.text.CollationKey est une alternative plus efficace qui effectue les tris deux fois plus rapidement.

Les deux méthodes suivantes permettent de comparer la performance d'un quicksort implémenté respectivement à l'aide de java.text.Collator et java.text.CollationKey.

Quicksort implémenté avec java.text.Collator[194]

```
public runsort( ) {
quicksort(stringArray,0,stringArray.length-1,
```

[193] Source: Doy Bulka, Java™ Performance and Scalability Volume 1: Server-Side Programming Techniques, Addison Wesley, June 05, 2000, 0-201-70429-3, pages 320
[194] Source: Jack Shirazi, Java Performance Tuning, 1st Edition September 2000 , ISBN: 0-596-00015-4, pages 496

```
Collator.getInstance( ));}
public static void quicksort(String[ ] arr, int lo, int hi,
java.text.Collator c)
{... int mid = ( lo + hi ) / 2;
String middle = arr[ mid ]; //String data type ...
//uses Collator.compare(String, String)
if( c.compare(arr[ lo ], middle) > 0 )
...}
```

Quicksort implémenté avec java.text. java.text.CollationKey [195]

```
public runsort( ) {
quicksort(stringArray,0,stringArray.length-1,
Collator.getInstance( ));}
public static void quicksort(String[ ] arr, int lo, int hi,
Collator c)

{//convert to an array of CollationKeys
CollationKey keys[ ] = new CollationKey[arr.length];
for (int i = arr.length-1; i >= 0; i--)
keys[i] = c.getCollationKey(arr[i]);
//Run the sort on the collation keys
quicksort_collationKey(keys, 0, arr.length-1);
//and unwrap so that we get our Strings in sorted order
for (int i = arr.length-1; i >= 0; i--)
arr[i] = keys[i].getSourceString( );}
public static void quicksort_collationKey(CollationKey[ ] arr,
int lo, int hi)

{...
int mid = ( lo + hi ) / 2;
CollationKey middle = arr[ mid ]; //CollationKey data type
...//uses CollationKey.compareTo(CollationKey)
if( arr[ lo ].compareTo(middle) > 0 )...}
```

[195] Source: Idem

5.1.2 Création/ réutilisation/ importation d'objets

5.1.2.1 Création prématurée d'objets

Dans les langages structurés comme C les programmeurs ont pris l'habitude de déclarer toutes les variables au début du programme. Dans Java cette pratique est pénalisante pour les performances. Par exemple dans la méthode f, l'objet x est crée à l'extérieur du périmètre dans lequel il sera utilisé[196]. Entre le moment où l'objet x est crée et le moment où il est effectivement utilisé, le constructeur de la classe Date est appelé vainement et la mémoire est occupée inutilement.

Instanciation prématurée

```
void f() {  int i;
 Date x = new Date();   // x est crée prématurément
   if (...) { // x ne sera utilisé que dans cette partie du programme } ...}
```

La solution à ce problème est très simple : **il ne faut créer l'objet que lorsqu'on veut l'utiliser (lazy instanciation).**

Lazy instanciation

```
void f() {
   int i;
   ...
   if (...) {  Date x = new Date();
                    // x est construit à l'intérieur du périmètre où il est utilisé  ..... }...}
```

5.1.2.2 Réutilisation des objets

En règle générale il est toujours souhaitable de garder un objet en mémoire si on compte le réutiliser dans la suite d'un programme. La raison est que la réinitialisation d'un objet déjà crée consomme moins de temps et de ressources que la création d'un nouvel objet. L'exemple suivant montre que la création d'un nouveau vecteur est une opération moins performante que le recyclage d'un vecteur déjà existant.

[196] Nous rappelons que ce périmètre est délimité par les symboles {}

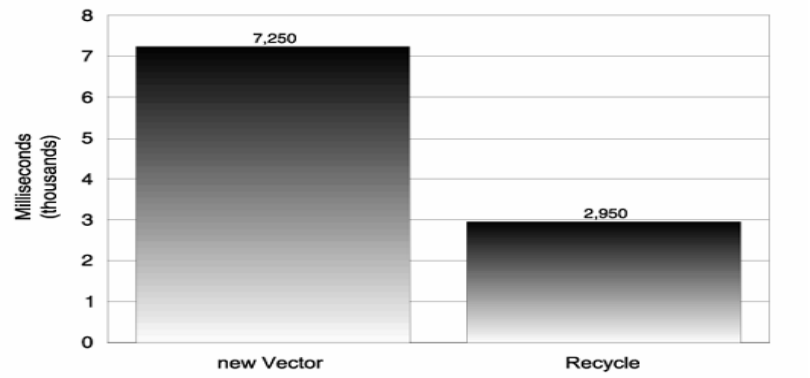
Création d'un nouveau vecteur [197]

```
String s = "HelloWorld";
String p = null;

long start = System.currentTimeMillis();  // <+++ Start timing
for (int i = 0; i < n; i++) {
   Vector v = new Vector();
   v.addElement(s);
   v.addElement(s);
   v.addElement(s);
   p = (String) v.elementAt(0);}
long stop = System.currentTimeMillis();  // <+++ Stop timing
```

Recyclage d'un vecteur existant

```
Vector v = new Vector();
long start = System.currentTimeMillis();  // <+++ Start timing
for (int i = 0; i < n; i++) {
   v.clear();                    // Reset the Vector to an initial state
   v.addElement(s);
   v.addElement(s);
   v.addElement(s);
   p = (String) v.elementAt(0);}

   long stop = System.currentTimeMillis();  // <+++ Stop timing
```

Figure 46: Création d'un nouveau vecteur vs recyclage d'un vecteur déjà existant[198]



[197] Source: Doy Bulka, Java™ Performance and Scalability Volume 1: Server-Side Programming Techniques, Addison Wesley, June 05, 2000, 0-201-70429-3, pages 320
[198] Source: Idem

5.1.2.3 Importation des classes

Contrairement à une croyance largement répandue, l'importation des classes n'a aucun impact sur les performances. Le fait d'importer une seule classe (par exemple import java.awt.Button;) ou toutes les classes d'un package (par exemple import java.awt.*;) n'augmente ni la taille de la mémoire consommée ni celle du programme. Les programmeurs peuvent donc utiliser autant d'instructions import qu'ils le souhaitent.

5.1.3 Boucles for

5.1.3.1 Quel type d'indice utiliser avec les boucles for ?

De prime abord, certains programmeurs peuvent penser que le choix du type d'indice doit dépendre de la valeur maximale qu'il prend à la fin de la boucle (par exemple le type byte pour les indices dont la valeur maximale ne dépasse pas 127 ou le type short pour les indices limités à 32 767). En fait les boucles for fonctionnent beaucoup plus rapidement avec des indices de type int qu'avec tout autre type. La raison est que Java convertit les byte, short et char en int avant d'effectuer des calculs, puis procède à la conversion inverse une fois le calcul terminé. Le programme suivant permet de vérifier nos affirmations :

```
class TestByte {
public static void main(String[ ] args) { int x = 0;
long t = System.currentTimeMillis();
for (byte i = 0; i < 1342;i++) {
for (byte j = 0; j < 1342;j++) {
for (byte k = 0; k < 1342;k++) {
for (byte l = 0; l < 1342;l++) {
x = 300; } } } }
```

Dans cette imbrication, un indice de type int permet d'obtenir un gain en performance d'environ 30%.

5.1.3.2 Appel de méthodes dans les paramètres d'une boucle for

En règle générale, il faut éviter d'utiliser un appel de méthode dans les paramètres d'une boucle for. Par exemple dans une boucle qui contient autant d'itérations qu'il y a d'éléments dans une collection, il faut préférer la syntaxe 2 à la syntaxe 1.

Syntaxe 1 :

```
for(int i = 0; i < collection.length( ); i++)
```

Syntaxe 2:

```
int maximum = v.length( )
for(int i = 0; i < maximum; i++)
```

5.1.3.3 System.arraycopy( ) vs boucle for

Lorsqu'on souhaite copier les éléments d'un tableau dans n'importe quel récipient, il est toujours recommandé d'utiliser la méthode System.arraycopy() au lieu d'un algorithme basé sur une boucle for (notons néanmoins que lorsque la JVM compile à la volée (JIT) les écarts de performance entre une boucle for et System.arraycopy() sont insignifiants).

```
public class loopTest1{[199]
        public static void main(String s[]){
                long start,end;
            int[] a=new int[2500000];
            int[] b= new int[2500000];
            for(int i=0;i<a.length;i++){
                    a[i]=i;
            }
            start=System.currentTimeMillis();
                for(int j=0;j<a.length;j++){
            b[j]=a[j];}
            end=System.currentTimeMillis();
            System.out.println(end-start + " milli seconds for loop copy ");
                int[] c= new int[2500000];
            start=System.currentTimeMillis();
            System.arraycopy(a,0,c,0,c.length);
            end=System.currentTimeMillis();
            System.out.println(end-start + " milli seconds for System.arraycopy() ");
        }}
```

[199] Source: http://www.precisejava.com/javaperf/j2se/Loops.htm

Résultats

- 110 millisecondes pour la boucle.
- 50 millisecondes pour System.arraycopy()

5.1.3.4 Générer une exception pour mettre fin à une boucle for

Si une boucle doit se terminer lorsqu'une exception est générée, il ne faut pas fixer de limite maximale dans les paramètres de la boucle pour éviter au programme d'effectuer inutilement des comparaisons. Dans ce genre de situations, il faut préférer la syntaxe 2 à la syntaxe 1.

Syntaxe 1 :

```
for (int boucle = 0; boucle < limitemaximale; boucle++)
```

Syntaxe 2 :

```
try
{for (int boucle = 0; ; boucle++) //une exception est déclenchée ici pour mettre fin à la boucle }
catch(Exception e) { }
```

5.1.4 Collections Java 2

5.1.4.1 Comparatif des performances

Pour pouvoir comparer les performances des collections Java 2, il faudrait les départager en leur faisant subir le même test. Afin de mettre en œuvre ce test nous nous sommes basés sur un benchmark publié dans le livre Java(TM) Platform Performance: Strategies and Tactics[200]. Ce benchmark compare pour les différents types de collections, les temps nécessaires pour:

- Ajouter de nouveaux éléments à leur contenu.
- Supprimer des éléments de ce contenu.
- Accéder de façon aléatoire aux données stockées dans les collections.
- Parcourir les différents éléments d'une collection.

[200] Steve Wilson, Jeff Kesselman, Java(TM) Platform Performance:Strategies and Tactics, Addison-Wesley Professional; 1st edition (May 31, 2000), ISBN: 0201709694, pages 230

La méthode 1 déclenche tous les tests effectués par le benchmark. La méthode 2 contient la logique nécessaire pour chronométrer les opérations effectuées par le benchmark. Les méthodes 3, 4 et 5 implémentent ces opérations (ajout, suppression, parcours des éléments, random access).

Méthode 1 : Déclenchement du test

```
public static void main(String[] args) { // cette méthode n'est pas limitative
Collection arrayList = new ArrayList(); // On peut lui ajouter les autres types de collections
test(arrayList);
Collection linkedList = new LinkedList();
test(linkedList);
Collection vector = new Vector();
test(vector);
Collection treeSet = new TreeSet();
test(treeSet);
Collection hashSet = new HashSet();
test(hashSet);
}
```

Méthode 2 : Chronométrage des opérations

```
public static void test(Collection c) {
System.out.println("Testing "+c.getClass());
Stopwatch timer = new Stopwatch().start();
add(c);
timer.stop();
System.out.println("add: "+
timer.getElapsedTime());
timer.reset().start();
iterate(c);
timer.stop();
System.out.println("iter: "+
timer.getElapsedTime());
if (c instanceof List) {
List l = (List)c;
timer.reset().start();
randomAccess(l);
timer.stop();
```

```
System.out.println("random: "+
timer.getElapsedTime());
}
timer.reset().start();
remove(c);
timer.stop();}
```

Méthode 3: Ajout d'éléments aux collections

```
public static void add(Collection c) {
for (int i = 0; i < NUM_ITEMS; i++) {
c.add(objects[i]); }}
```

Méthode 4: Parcours des éléments de la collection

```
public static void iterate(Collection c) {
for (int i = 0; i < 100; i++) {
Iterator iter = c.iterator();
while (iter.hasNext()) {
Object o = iter.next();}}}
```

Méthode 5: Suppression des éléments de la collection

```
public static void remove(Collection c) {
Iterator iter = c.iterator();
while (iter.hasNext()) {
Object o = iter.next();
iter.remove();}}
```

Méthode 6: Accès aléatoire

```
public static void randomAccess(List c) {
for (int i = 0; i < NUM_ITEMS; i++) {
Object o = c.get(random());}
}
```

Les résultats obtenus sont donnés par le tableau 20.

Tableau 20: Comparatif des performances des principales collections

Classe	Ajout	Parcours	Accès aléatoire	Suppression
ArrayList	1 ms	700 ms	1 ms	3000 ms
LinkedList	60 ms	1200 ms	3002 ms	1 ms
Vector	1 ms	880 ms	1 ms	2589 ms
TreeSet	330 ms	1410 ms	Ne s'applique pas	68 ms
HashSet	110 ms	1430 ms	Ne s'applique pas	55 ms

En généralisant ce test aux autres collections et en calculant la moyenne des temps de réponse obtenus nous sommes arrivés aux conclusions suivantes :

Classement des classes implémentant l'interface **Set** de la plus performante à la moins performante.

1. HashSet
2. LinkedHashSet
3. TreeSet

Classement des classes implémentant l'interface **Map** de la plus performante à la moins performante.

1. IdentityHashMap
2. HashMap
3. Hashtable
4. LinkedHashMap
5. TreeMap

Classement des classes implémentant l'interface **List** de la plus performante à la moins performante.

1. ArrayList
2. Vector
3. Stack (meme performances que Vector)
4. LinkedList

5.1.4.2 Impact de l'emplacement des éléments d'une collection sur la performance

Les collections exposent plusieurs méthodes pour ajouter des éléments à leur contenu. Malgré leur diversité ces méthodes peuvent être scindées en deux grandes catégories :

- Les méthodes qui ajoutent les éléments à la fin de la collection.
- Les méthodes qui ajoutent les éléments dans un emplacement spécifié par le paramètre index.

Les méthodes de la deuxième catégorie sont moins performantes que celles de la première catégorie parce qu'avant de pouvoir insérer les nouveaux éléments elles doivent libérer de l'espace en déplaçant le contenu de la collection.

L'exemple du tableau 21 permet de comparer les performances de ces deux catégories de méthodes pour la classe Vector.

NB: ces résultats sont aussi valables pour les méthodes de suppression et les autres types de collections.

Tableau 21: Impact des méthodes addElement() et insertElementAt() sur les performances (classe Vector)

addElement()	insertElementAt()

public void add1000(Vector v, String s) {
for (int i = 0; i < 1000; i++) {
v.addElement(s);}
}
	public void add1000AtFront(Vector v, String s)
	{for (int i = 0; i < 1000; i++) {
	v.insertElementAt(s,0);}
	}

Temps consommé : 750 millisecondes	Temps consommé : Résultat: 5400

5.1.4.3 Capacité d'une collection

Parce que l'expansion de la taille d'une collection déjà créée est une opération pénalisante pour les performances, il faut dès le départ prévoir le nombre d'éléments qui y seront stockés et le passer en paramètre au constructeur pour éviter que cette expansion ait lieu (exemple : Vector v = new Vector(100);).

5.1.5 Gestion des exceptions

5.1.5.1 Impact des blocs try/ catch qui ne génèrent pas d'exceptions

Lorsque aucune exception n'est générée, les blocs try/catch n'affectent pas les performances de façon significative. Pour corroborer cette affirmation nous nous permettons de reproduire les résultats des tests effectués par Jack Shirazi dans la première édition du livre Java Performance Tuning.

Ce test compare sur plusieurs JVM, les performances d'un bloc catch try placé à l'intérieur d'une boucle avec celui d'un bloc catch/try qui enveloppe cette même boucle.

```
public class TryCatchTimeTest[201]
{ public static void main(String[ ] args) {
int REPEAT = (args.length = = 0) ? 10000000 :
Integer.parseInt(args[0]);
Object[ ] xyz = {new Integer(3), new Integer(10101), new
Integer(67)}; boolean res;
long time = System.currentTimeMillis( );
res = try_catch_in_loop(REPEAT, xyz);
System.out.println("try catch in loop took " + (System.currentTimeMillis( ) - time));
time = System.currentTimeMillis( );
res = try_catch_not_in_loop(REPEAT, xyz);
System.out.println("try catch not in loop took " +
(System.currentTimeMillis( ) - time));
//Repeat the two tests several more times in this method
//for consistency checking}
public static boolean try_catch_not_in_loop(int repeat,
Object[ ] o){ Integer i[ ] = new Integer[3];
try { for (int j = repeat; j > 0; j--)
{ i[0] = (Integer) o[(j+1)%2];
i[1] = (Integer) o[j%2];
i[2] = (Integer) o[(j+2)%2]; }
return false; } catch (Exception e) {return true;} }
public static boolean try_catch_in_loop(int repeat,Object[ ] o)
{Integer i[ ] = new Integer[3];
for (int j = repeat; j > 0; j--)
```

[201] Source: Jack Shirazi, Java Performance Tuning, 1st Edition September 2000 , ISBN: 0-596-00015-4, pages 496

```
{try {
i[0] = (Integer) o[(j+1)%2];
i[1] = (Integer) o[j%2];
i[2] = (Integer) o[(j+2)%2];
}catch (Exception e) {return true;} }
return false;} }
```

Les résultats obtenus sont donnés par le tableau 22.

Tableau 22: Impact des blocs catch/try qui ne génèrent pas d'exception[202]

Extra cost of the looped try-catch test relative to the nonlooped try-catch test							
VM	**1.1.8**	**1.2.2**	**1.3.1**	**1.3.1-server**	**1.4.0**	**1.4.0-server**	**1.4.0 -xInt**
Increase in time	~5%	~10%	None	None	None	None	~2%

5.1.5.2 Impact des blocs try/ catch qui génèrent des exceptions

Contrairement aux blocs catch/try étudiés dans la section précédente, les blocs catch/try qui génèrent des exceptions sont pénalisants pour le temps de réponse parce qu'ils doivent afficher le message d'erreur crée dans le traceur de la pile.

Si on répète le test effectué dans la section 5.1.5.1 et qu'on l'adapte pour qu'il génère des exceptions on obtient des temps de réponse supérieurs 6 à 10 fois à ceux de la section précédente. Pour éviter ces pénalités, les programmeurs doivent réduire au strict minimum le nombre de blocs catch/try inclus dans leur code (en règle générale il faut préférer les blocs factorisés[203] aux blocs individuels[204]).

5.1.6 Casting

Les opérations de sur-casting et de sous-casting ont un impact direct sur les performances des applications. La gravité de ces impacts varie :

- Selon que le casting se fasse en mode compilation ou en mode exécution.
- Selon que le casting porte sur des objets ou des primitives, des interfaces ou des classes concrètes.

[202] Source: idem.
[203] Blocs partagés par plusieurs méthodes.
[204] Autant de blocs catch/try qu'il y a de méthodes.

En règle générale:

- Les castings qui se font en mode exécution sont moins performants que les castings en mode compilation.
- Le casting des primitives est plus performant que celui des objets.
- Le casting des interfaces et moins performant que celui des classes concrètes.

Un autre facteur qui peut aggraver l'impact du casting sur les performances est la profondeur des relations d'héritage qui existent entre les classes. En effet plus l'arborescence des classes est profonde et plus le casting met du temps à se réaliser. Compte tenu de ces impacts, les programmeurs doivent être précis dans le choix des types d'objets et doivent éviter les castings s'ils connaissent à l'avance le type des objets traités par leur application (par exemple au lieu de créer un List pour stocker un ensemble de chaînes de caractères String ils peuvent réaliser des gains importants en utilisant dès le départ des objets StringList).

5.1.7 Passage des paramètres, propriétés statiques, propriétés d'instances

Quelque soit leur nombre et leur type le passage des paramètres a des conséquences anodines sur les performances.

A ce sujet, les paramètres et les propriétés locales[205] sont lus et écrits beaucoup plus rapidement que les propriétés statiques et les propriétés d'instances (la raison et que les paramètres et les propriétés locales sont stockés dans une zone plus rapide d'accès au niveau de la pile). Notons enfin que les propriétés d'instance occupent beaucoup plus de mémoire que les propriétés statiques.

5.1.8 Fichiers JAR

Il est toujours préférable de regrouper les classes d'une application dans des fichiers JAR avant de la déployer. Outre la réduction du trafic sur le réseau, la compression permet une distribution plus facile de l'application. Cela dit, il ne faut pas non plus placer un nombre trop élevé de classes dans un même fichier JAR au risque d'accroître le temps nécessaire pour leur décompression.

[205] C'est à dire les propriétés temporaires qui sont créées à l'intérieur des méthodes.

En vue de profiter pleinement des avantages des fichiers JAR, il faudrait que les classes soient compressées dans le même ordre de leur chargement par l'application (ordre qui peut être connu en exécutant l'application avec l'option –verbose)

5.1.9 Caching des applets

Le temps de téléchargement est l'un des principaux problèmes de performance rencontrés avec les applets. Outre les techniques de compression présentées dans la section précédente, la mise en cache des applets au niveau du client contribue à la résolution de ce problème.

Cette mise en cache peut être gérée automatiquement par les Sun's Java Plug-in ou codée par le programmeur dans l'applet deployer (cf. tags infra). Pour notre part, nous recommandons cette deuxième alternative parce qu'elle permet d'avoir plus de contrôle sur le processus de gestion de cache au niveau du navigateur (notons qu'avec la première alternative, le navigateur peut supprimer les applets du cache pour faire place à des fichiers volumineux).

Tags à ajouter à la page HTML pour contrôler la gestion du cache :

```
<OBJECT ....>
        <PARAM NAME="archive" VALUE="...">
        ....
        <PARAM NAME="cache_option" VALUE="...">
        <PARAM NAME="cache_archive" VALUE="...">
          <PARAM NAME="cache_version" VALUE="...">
</OBJECT>
```

L'attribut cache_option peut prendre l'une des trois valeurs suivantes:

- No : l'applet n'est jamais mise en cache. Elle est toujours téléchargée à partir du serveur.
- Browser: l'applet est exécutée à partir du cache du browser (option par défaut).
- Plugin: l'applet est exécutée à partir du cache du plug-in.

L'attribut cache_archive pour sa part contient la liste des fichiers JAR dans le cas où l'applet est compressée. Exemple d'utilisation:

<PARAM NAME="cache_archive" VALUE="a.jar,b.jar,c.jar">

La mise à jour de l'applet placée dans le cache a lieu lorsque :

- La valeur "Last-Modified" du cache_archive placé sur le serveur est plus récente que celle du cache_archive placé dans le cache.
- La valeur "Content-Length" du cache_archive placé sur le serveur est plus récente que celle du cache_archive placé dans le cache.

Il est important de noter que Last-Modified et Content-Length ne retournent pas toujours des résultats fiables lorsque la communication entre le client et le serveur se fait via HTTPS (utilisation du protocole SSL). Pour éviter ce problème il faut utiliser l'attribut cache_version qui associe des versions au cache_archive pour mieux gérer leur mise à jour.

5.2 Optimisation des principaux composants J2EE

5.2.1 Optimisation des servlets et des JSP

5.2.1.1 Se méfier des objets PrintWriter

Une des erreurs fréquemment commises dans le codage des servlets consiste à utiliser la méthode println() de l'objet PrintWriter pour générer le contenu HTML destiné au browser (cf. classe Writer1). Avec une console[206], la méthode println() crée un séparateur pour afficher les résultats sur une nouvelle ligne. Avec les pages HTML ce séparateur est tout simplement ignoré parce que le browser n'interprète que les balises HTML.

Classe Writer1

```
public class Writer1 extends HttpServlet {public void doGet (HttpServletRequest req,
HttpServletResponse res) throws ServletException, IOException {
res.setContentType("text/html");
   PrintWriter out = res.getWriter();
     out.println("<html>"); out.println("<head><title>Hello World</title></head>");
```

[206] Ecran MS-DOS

```
out.println("<body>"); for (int i = 0 ; i < 10; i++) { out.println("<h1>Hello World 1</h1>");}
out.println("</body></html>");}}
```

En utilisant de façon interchangeable les méthodes println() et print() on obtient les mêmes résultats mais pas le même niveau de performance. En fait le traitement d'une méthode println(String s) équivaut en termes de performance au traitement de la méthode suivante :

```
public void println(String s) { print(s); println();}
```

La méthode println() fait appel en arrière appel à la méthode print() qui à son tour doit réaliser des conversions vers le format spécifié par la méthode setContentType() de la classe HttpServletResponse. Ces conversions sont très coûteuses en mémoire et en processeur pour des formats comme l'unicode. Pour cette raison il faut essayer dans toute la mesure du possible de substituer des objets comme ServletOutputStream à l'objet PrintWriter (un exemple sur ce genre de substitutions est donné dans la section 5.2.1.4).

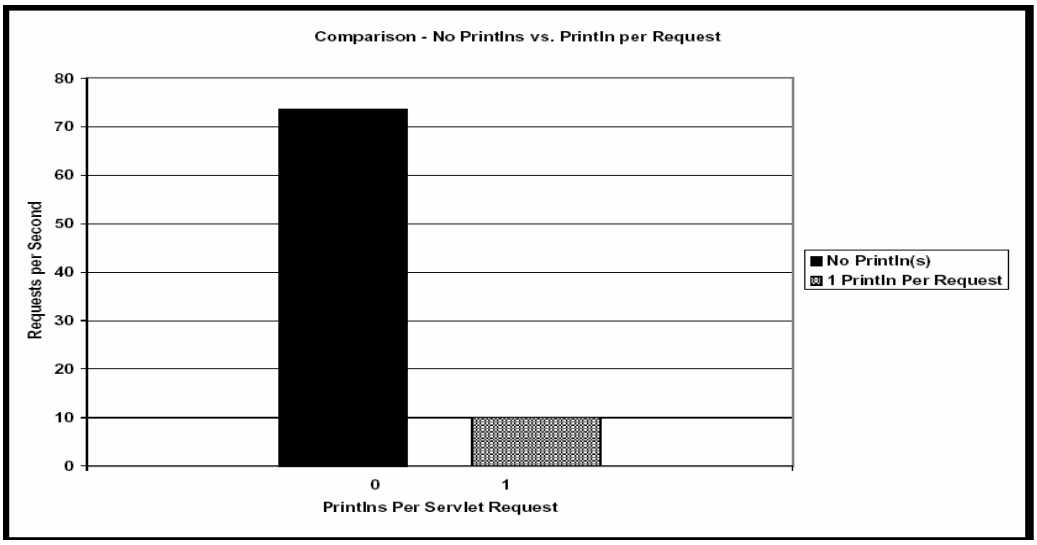
Cela dit les servlets ne sont pas optimisées pour implémenter la couche de présentation. Comme nous l'avons vu dans la partie 4, elles sont plus efficientes lorsqu'elles sont utilisées exclusivement dans l'implémentation du contrôleur. A chaque fois que cela est possible, il faut évacuer des servlets[207] tout le code qui contribue directement à la génération de la vue. Autrement on risque de créer des couplages qui peuvent s'avérer fatal pour le débit de l'application comme le montre la figure 47.

5.2.1.2 Eviter l'implémentation de l'interface SingleThreadModel

Comme nous l'avons déjà expliqué dans les parties 2 et 4, le conteneur crée pour les classes qui implémentent l'interface SingleThreadModel autant d'instances de la servlet qu'il y a de requêtes entrantes vers l'application. Cela affecte non seulement la quantité de mémoire disponible pour l'application mais également les traitements parallèles.
En règle générale il faut éviter d'implémenter cette interface pour ne pas épuiser inutilement les ressources du serveur. Il faudrait plutôt opter pour la classe HttpServlet parce qu'une seule instance de cette classe traite toutes les requêtes provenant du client (en fait le conteneur attribue un seul processus à la servlet mais crée à l'intérieur de ce processus autant de threads qu'il y a de requêtes).

[207] Typiquement System.out.print() ou println()

Figure 47: Impact de la méthode println() sur le débit d'une servlet[208]



5.2.1.3 Implémenter un filtre de cache

Comme nous l'avons vu dans la partie 4, les Filters interceptent en amont toutes les requêtes entrantes vers l'application et en aval toutes les réponses sortant de celle-ci. Cette position intermédiaire les qualifie pour jouer un rôle clé dans la gestion du cache.
Dans cette partie nous présenterons un exemple qui montre comment implémenter la gestion du cache au niveau d'un Filter (Cf. classe CacheFilter). Cet exemple a été initialement publié dans le livre: Servlets and JavaServer Pages™: The J2EE™ Technology Web Tier.

Classe CacheFilter[209]

```
package com.jspbook;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.Calendar;
public class CacheFilter implements Filter {ServletContext sc;
  FilterConfig fc;
```

[208] Source: Harvey W. Gunther, WebSphere Application Server Development Best Practices for Performance and Scalability, IBM White Papers, September 7, 2000
[209] Source: By Jayson Falkner, Kevin Jones, Servlets and JavaServer Pages™: The J2EE™ Technology Web Tier, Addison Wesley, September 19, 2003, ISBN: 0-321-13649-7, pages 784

```
  long cacheTimeout = Long.MAX_VALUE;
public void doFilter(ServletRequest req,
                     ServletResponse res,
                     FilterChain chain)
    throws IOException, ServletException {
   HttpServletRequest request =
      (HttpServletRequest) req;
   HttpServletResponse response =
      (HttpServletResponse) res;
   // check if was a resource that shouldn't be cached.
   String r = sc.getRealPath("");
   String path =
      fc.getInitParameter(request.getRequestURI());
   if (path!= null && path.equals("nocache")) {
    chain.doFilter(request, response);
    return;}
   path = r+path;
   String id = request.getRequestURI() +
      request.getQueryString();
   File tempDir = (File)sc.getAttribute(
    "javax.servlet.context.tempdir");
   // get possible cache
   String temp = tempDir.getAbsolutePath();
   File file = new File(temp+id);
   // get current resource
   if (path == null) {  path = sc.getRealPath(request.getRequestURI()); }
   File current = new File(path);
   try { long now =
      Calendar.getInstance().getTimeInMillis();
    //set timestamp check
    if (!file.exists() || (file.exists() &&
        current.lastModified() > file.lastModified()) ||
        cacheTimeout < now - file.lastModified()) {
      String name = file.getAbsolutePath();
      name =
        name.substring(0,name.lastIndexOf("/"));
      new File(name).mkdirs();
```

```
        ByteArrayOutputStream baos =
            new ByteArrayOutputStream();
        CacheResponseWrapper wrappedResponse =
          new CacheResponseWrapper(response, baos);
        chain.doFilter(req, wrappedResponse);
        FileOutputStream fos = new FileOutputStream(file);
        fos.write(baos.toByteArray());
        fos.flush();
        fos.close();}} catch (ServletException e) {
      if (!file.exists()) {
        throw new ServletException(e);}}
    catch (IOException e) {
      if (!file.exists()) {
        throw e;}}
    FileInputStream fis = new FileInputStream(file);
    String mt = sc.getMimeType(request.getRequestURI());
    response.setContentType(mt);
    ServletOutputStream sos = res.getOutputStream();
    for (int i = fis.read(); i!= -1; i = fis.read()) {
      sos.write((byte)i);}}
  public void init(FilterConfig filterConfig) {
    this.fc = filterConfig;
    String ct =
        fc.getInitParameter("cacheTimeout");
    if (ct != null) {
      cacheTimeout = 60*1000*Long.parseLong(ct);}
    this.sc = filterConfig.getServletContext();}
  public void destroy() {this.sc = null; this.fc = null;}}
```

Le filtre implémenté par la classe CacheFilter peut s'appliquer à n'importe quelle servlet ou JSP de l'application. A partir de l'URI de la requête et des paramètres qu'elle transporte, il construit un identifiant unique pour référencer les JSP/servlets dans le cache :

```
String id = request.getRequestURI()+request.getQueryString();
```

Le premier paramètre transporté par la requête permet d'indiquer si la JSP/servlet demandée peut être mise en cache. Par défaut toutes les JSP/servlets peuvent être mises en cache sauf si ce paramètre est égal à 'nocache'. Si tel est le cas, le filtre interrompt son activité et passe la main à la prochaine ressource de l'application (servlet, autre filtre…)

```
// check if was a resource that shouldn't be cached.
String path = fc.getInitParameter(request.getRequestURI());
if (path!= null && path.equals("nocache")) {chain.doFilter(request, response);
return;}
```

Si la JSP/servlet demandée par la requête peut être mise en cache, le Filtre vérifie si elle a déjà été mise en cache. Si tel est le cas le Filtre vérifie si la date de la dernière modification de la JSP/servlet est postérieure à celle du cache. Si c'est le cas le filtre met en cache la nouvelle version de la JSP /servlet sinon il répond au client à partir du cache.

Dans le reste des cas, le filtre passe la main aux autres ressources de l'application puis intercepte la réponse pour la mettre en cache dans le répertoire temporaire javax.servlet.context.tempdir avant de l'envoyer au client. Le diagramme de séquence de la figure 36 (partie 4) illustre ces différents cas de figure.

La technique présentée jusqu'ici met en cache les données au niveau du serveur. Une autre technique qui permet de partager le fardeau avec le client consiste à altérer l'entête Cache-Control des paquets http pour ordonner au navigateur de conserver les données dans son propre cache (voir classe ResponseHeaderFilter).

class ResponseHeaderFilter

```
package com.jspbook;[210]
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
```

[210]Source: Jayson Falkner, Another Java Servlet Filter Most Web Applications Should Have, Client-Side Cache Control, http://www.onjava.com/pub/a/onjava/2004/03/03/filters.html

```
public class ResponseHeaderFilter implements Filter {
  FilterConfig fc;
  public void doFilter(ServletRequest req,
                       ServletResponse res,
                       FilterChain chain)
                       throws IOException,
                            ServletException {
    HttpServletResponse response =
     (HttpServletResponse) res;
    // set the provided HTTP response parameters
    for (Enumeration e=fc.getInitParameterNames();
        e.hasMoreElements();) {
      String headerName = (String)e.nextElement();
      response.addHeader(headerName,
              fc.getInitParameter(headerName)); }
    // pass the request/response on
    chain.doFilter(req, response);}
  public void init(FilterConfig filterConfig) {
    this.fc = filterConfig;}
  public void destroy() { this.fc = null;}}
```

Le filtre présenté ci-dessus permet d'altérer n'importe quel entête du paquet http. Pour altérer celui chargé de la gestion du cache il faut créer un mapping entre le paramètre Cache-Control, le headerName et les données qu'on souhaite mettre en cache. Ce mapping doit être crée au niveau du fichier de déploiement web.xml (deployement descriptor).

```
<filter>
  <filter-name>
   ResponseHeaderFilter</filter-name>
  <filter-class>
   com.jspbook.ResponseHeaderFilter</filter-class>
  <init-param>
    <param-name>
     Cache-Control</param-name>
  </init-param>
```

```
</filter>
<filter-mapping>
  <filter-name>
   ResponseHeaderFilter</filter-name>
  <url-pattern>/logo.png</url-pattern> // cet exemple met le logo dans le cache du client.
</filter-mapping>
```

5.2.1.4 Gérer le cache via la méthode init() de la classe HttpServlet

La méthode init() de la classe HttpServlet est également un bon emplacement pour implémenter la gestion du cache. Contrairement à la méthode service() qui est appelée à chaque fois que la servlet reçoit une requête ou renvoie une réponse, init() n'est appelée qu'une seule fois dans le cycle de vie de la servlet. Cela permet de réduire considérablement le nombre de va et vient (round trip) entre le client et la servlet lorsque les réponses retournées ne changent pas fréquemment (si les réponses ne changent jamais elles doivent être normalement stockées dans une page HTML statique).

- L'exemple 1 régénère la réponse à chaque fois que la servlet reçoit une nouvelle requête.
- L'exemple 2 génère la réponse une seule fois dans le cycle de vie de la servlet puis la place dans le cache. Notons par ailleurs que l'objet PrintWriter est remplacé par un StringBuffer pour améliorer les performances (pour plus de détails à ce sujet se rendre à la section 5.2.1.1).

Exemple 1 [211]:

```
public void service(HttpServletRequest req, HttpServletRespnse res) throws
ServletException,IOException {
res.setContentType("text/html");
Printwriter out = req.getWriter();
out.print("<html>");
out.print("<head><title>Hello world</title></head>");
out.print("<body>");
// send the dynamic data here
```

[211] Source: Ravi Kalidindi and Rohini, http://www.precisejava.com/javaperf/j2ee/Servlets.htm#Servlets102 , Best Practices to improve performance in Servlets

```
out.print(“</body>”);
out.print("</html>”);}
```

Exemple 2 :

```
public class servlet extends HttpServlet {
        byte[] header; byte[] navbar; byte[] footer; byte[] otherStaticData;
public void init(ServletConfig config) throws ServletException{
//create all the static data here
//StringBuffer sb = new StringBuffer(); // better to initialize the StringBuffer with some size to
//improve performance
sb.append("<html>”);
sb.append("<head><title>Hello world</title></head>”);
sb.append(“<body>”); header = sb.toString().getBytes();
// do same for navbar if its data is static
// do same for footer if its data is static}
public void service(HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOException { res.setContentType("text/html");
ServletOutputStream out = res.getOutputStream();
out.write(header); out.write(navbar);
// write dynamic data here
out.write(footer);}
```

5.2.1.5 Contrôler la progression des objets HttpSession

Comme nous l’avons vu dans la partie 4 (section 4.2.1.2) la gestion des sessions via les objets qui implémentent l’interface HttpSession offre de meilleures performances comparativement aux techniques basées sur les champs HTML cachés[212], sur la réécriture d’URL ou encore sur les cookies. Cependant si la taille de ces objets n’est pas contrôlée, ils peuvent réduire la quantité de mémoire disponible pour l’application et affecter son débit à la baisse (par exemple sur le serveur d’application WebSphere le nombre de requêtes traitées par seconde passe de 28 r/s à 15 r/s lorsque la taille des objets de session passe de 4 Ko à 16 Ko[213]).

[212] Les objets HttpSession utilisent également les cookies et la réécriture d’URL mais ils offrent un meilleur niveau de performance parce que les états (states) de la session sont stockés dans la mémoire RAM du serveur. Se rendre à la section 4.2.1.2 pour plus de détails à ce sujet.
[213] Source: Harvey W. Gunther, WebSphere Application Server Development Best Practices for Performance and Scalability, IBM White Papers, September 7, 2000

La spécification J2EE n'offre aucun moyen pour fixer la taille de la mémoire occupée par les objets HttpSession mais leur progression peut être jugulée par le biais des techniques suivantes :

- Configurer le conteneur de façon à limiter le nombre d'objets HttpSession crées par l'application.
- Réduire la durée de vie des objets HttpSession afin que le conteneur puisse libérer de la mémoire lorsque le client arrête d'interagir avec le site (cette réduction se fait via la méthode getMaxInactiveInterval()). Il faut néanmoins tenir compte des cas d'utilisation pour ne pas perturber le fonctionnement de l'application (pour plus de détails à ce sujet se rendre à la section 4.2.1.2)
- Invalider les variables de session par le biais de la méthode invalidate() (par exemple lorsque le site contient un bouton qui permet au client de se déconnecter : bouton de type log out).

Cela dit, si l'application n'utilise pas les objets HttpSession il faut penser à les désactiver au niveau des JSP à l'aide de la directive <%@ page session="false"%> pour ne pas épuiser inutilement les ressources du serveur.

5.2.1.6 Désactiver le chargement automatique des servlets/JSP

Les conteneurs sont dotés d'une fonctionnalité qui permet de recharger dynamiquement une servlet/JSP à l'expiration d'un certain délai ou lorsque son byte code change. En phase de développement ce chargement évite au programmeur de redémarrer le conteneur pour tenir compte des changements survenus mais en phase de production les performances sont pénalisées. Il faut donc penser à désactiver le chargement automatique des JSP/servlets avant de mettre l'application en production (par exemple sous Tomcat cela se fait en affectant la valeur 'false' aux balises de contexte contenues dans le fichier server.xml).

Example :

```
<Context path="/examples" docBase="examples" debug="0"

        reloadable="false">
</Context>
```

5.2.1.7 Compression des réponses

Comme nous l'avons expliqué dans la partie 4, la position occupée par les servlets et les filtres dans l'architecture MVC leur permet de jouer un rôle clé dans la compression des données. Avant de procéder à la compression, la servlet et/ou le filtre doivent savoir si le navigateur supporte le format de compression utilisé. Cette information peut être puisée à partir de l'entête des paquets http reçus (dans l'exemple suivant, la méthode acceptsGZIP() vérifie si le browser supporte le format de compression GZIP).

Méthode acceptsGZIP()

```
public boolean acceptsGZIP(HttpServletRequest request) {
//Get any "Accept-Encoding" headers
String header;
Enumeration e = ((HttpServletRequest)request).getHeaders(
"Accept-Encoding");
while (e.hasMoreElements( ))
{String header = (String)e.nextElement( );
//And check that GZIP is supported
if ( (header != null) &&
(header.toUpperCase( ).indexOf("GZIP") > -1) )
return true;}
return false;}
```

La deuxième étape est d'informer le browser que la réponse envoyée est compressée (dans l'exemple suivant, la méthode setGZIPContent() informe le browser que la réponse est compressée au format GZIP).

Méthode setGZIPContent()

```
public void setGZIPContent(HttpServletResponse response) {
response.setHeader("Content-Encoding", "gzip");
}
```

Enfin la dernière étape consiste à compresser la réponse avec un utilitaire ou les classes du package java.util.zip qui supportent le format GZIP (dans l'exemple suivant cela se fait via la classe GZIPOutputStream).

```
if (acceptsGZIP(request))
{ setGZIPContent(response);
out = new GZIPOutputStream(response.getOutputStream( ));}
else{ out = response.getOutputStream( );}
```

Cela dit le programmeur ne doit recourir à la compression que si la taille de la réponse le justifie. En fait, il doit comparer le temps économisé lors du transport de la réponse vers le client avec le temps consommé par le processus de compression/ décompression pour savoir si la compression est une solution viable.

5.2.1.8 Include directive vs include action

La directive include et l'action include n'affectent pas les performances des JSP de la même manière :

- La directive include procède à l'inclusion de la page associée à son attribut file lorsque la JSP inclusive est compilée : <%@ include file="file .html or .jsp" %>. Que la page incluse soit statique ou dynamique son contenu est conservé en cache à l'issue de la compilation. Il n'est mis à jour dans la JSP inclusive que lorsque celle-ci est de nouveau recompilée.
- L'action include pour sa part procède à l'inclusion de la page associée à son attribut page lorsque la JSP inclusive est exécutée : <jsp:include page="file.jsp" flush="true"/>. Cette exécution consomme plus de mémoire et de processeur parce que la JSP inclusive est mise à jour à chaque fois qu'une nouvelle requête lui est envoyée.

5.2.1.9 Minimiser la portée des Java Beans

L'attribut scope de l'action usebean permet de définir la portée des Java Beans utilisés par une JSP. Dépendamment de la valeur assignée à cet attribut, l'accessibilité au bean peut être restreinte :

- A la page courante après la ligne de code contenant l'action usebean (<jsp:useBean id="objectName" scope="page" />).
- Aux pages référencées par les actions jsp:forward et jsp:include action (<jsp:useBean id="objectName" scope="request" />).

- A la session courante. Dans ce cas le bean est placé dans un objet de session. Il est accessible à toutes les pages visitées par le client pendant la durée de la session (<jsp:useBean id="objectName" scope="session" />).

- A toutes les pages de l'application (<jsp:useBean id="objectName" scope="application" />).

Lorsque la portée du bean est limitée à la page courante, le conteneur le détruit automatiquement aussitôt que le focus est déplacé vers une autre page. Par contre lorsque la portée du bean est étendue au niveau de la session ou de l'application, le conteneur est obligé de le garder en mémoire pour qu'il survive aux redirections et aux inclusions. Ce maintien en mémoire des beans peut être fatal pour le débit de l'application comme nous l'avons vu dans la section 4.2.1.2.

5.2.1.10 Redirects Versus Forwards

Les redirections effectuées par le biais de la méthode sendRedirect() de l'interface HttpServletResponse sont plus lentes que celles effectuées par l'action <jsp:forward ...> parce que le browser doit générer une nouvelle requête vers la page cible. Avec l'action <jsp:forward ...> le conteneur fait un simple appel interne vers la page cible. Cette deuxième alternative est, somme toute, moins pénalisante pour les performances que la première.

5.2.1.11 Précompilation des servlets/JSP

La compilation des servlets/JSP peut augmenter anormalement le temps de réponse perçu par le premier visiteur du site. Pour cette raison il faut précompiler toutes les classes de l'application avant de la déployer dans l'environnement de production (cela peut se faire manuellement via un environnement de développement intégré ou automatiquement par le conteneur servlet/JSP ou encore le serveur d'application).

5.2.1.12 Tags OSCache

Comme nous l'avons vu dans la partie 4, l'assemblage des vues composites est une opération très pénalisante pour les performances surtout sur les sites à forte affluence.

Les filtres que nous avons présentés dans les sections précédentes sont particulièrement adaptés pour mettre en cache l'intégralité d'une page. Toutefois lorsqu'il s'agit de mettre en cache les différents blocs des vues composites ces filtres perdent leur efficacité.

La librairie OSCache d'OpenSymphony apporte une solution simple et élégante à ce problème. Elle fournit un ensemble préconçu de tags qui libèrent les développeurs d'une majeure partie des tâches liées à l'utilisation des filtres. Cette librairie Open Source peut être utilisée pour cacher aussi bien des fragments de JSP que des JSP entières.

Le package OSCache est disponible gratuitement sur le site d'OpenSymphony (www.opensymphony.org). Pour l'installer sous Tomcat, il suffit de copier l'archive oscache.jar dans le répertoire WEB-INF/lib de l'application et de placer les fichiers oscache.tld et oscache.properties dans le répertoire WEB-INF/classes. Une fois ces fichiers copiés il faut modifier le deployement descriptor (web.xml) de la manière suivante :

```
<taglib>
  <taglib-uri>oscache</taglib-uri>
  <taglib-location>classes/oscache.tld</taglib-location>
</taglib>
```

Pour cacher un fragment particulier dans une page, il faut l'entourer avec le tag <cache :cache …> comme le montre l'exemple suivant :

```
<%@ page import="java.util.*" %>
<%@ taglib uri="oscache" prefix="cache" %>
<html>
<head><title>OsCache demonstration</title></head>
<body>
Non cached portion : <%= new Date() %><p>
<cache:cache time="10">
Cached portion: <%= new Date() %> <! (Refreshed every 10 seconds)>
</cache:cache> </body> </html>
```

Pour tester le fonctionnement du tag il faut charger la page à plusieurs reprises (bouton refresh du navigateur). La date non cachée s'incrémente alors que la date cachée reste fixe. Cependant s'il s'écoule un délai de plus de 10 secondes entre chaque rafraîchissement la date cachée est automatiquement mise à jour.

Il est également possible de cacher le contenu dans le cadre d'une session utilisateur:

```
<cache:cache time="10" scope="session">
…
</cache:cache>
```

OSCache est aussi capable d'associer des identifiants uniques aux tags de façon à pouvoir les référencer dans d'autres JSP (voir exemple).

Example1.jsp

```
<%@ page import="java.util.*" %>
<%@ taglib uri="oscache" prefix="cache" %>
<html>
<head><title>Example 1</title></head>
<body>
Non cached portion: <%= new Date() %><p>
<cache:cache key="associatedkey">
Cached portion: <%= new Date() %> <p>
</cache:cache>

<a href="Example2.jsp">Rafraîchir</a>
</body>
</html>
```

Example2.jsp

```
<%@ taglib uri="oscache" prefix="cache" %>
<html>
<head><title>Example2.jsp</title></head>
<body>
```

```
Flushing the cache...<p>
<cache:flush key="associatedkey" scope="application"/>
<a href="Example1.jsp">Retour</a>
</body>
</html>
```

La JSP Example1.jsp cache une portion du code en lui associant la clé " **associatedkey**". La JSP Example2.jsp utilise cette clef pour vider le cache. Sur un site de commerce électronique, la clé peut être calculée dynamiquement pour cacher les informations et les images des produits.

OScahe permet également de limiter le nombre d'objets placés dans le cache. Par défaut, ce nombre est de 1000 dans la version 1.7.5 mais une fois atteint, la mémoire est réutilisée selon l'algorithme LRU (Least Recently Used) que nous avons déjà présenté dans la section 4.2.2.2. Pour augmenter la taille du cache, il suffit d'éditer le fichier oscache.properties et d'ajuster la valeur du paramètre cache.capacity. Il est d'autre part possible de configurer OSCache pour cacher les fragments de page sur disque. Pour cela il suffit de modifier le paramètre cache.path en lui donnant comme valeur le chemin du répertoire disque où on souhaite conserver les données.

Cela dit, la librairie OSCache a dépassé le stade d'expérimentation. Aujourd'hui elle est utilisée par des sites aussi prestigieux que theserverside.com et bénéficie du support d'une communauté de programmeurs de plus en plus grandissante.

5.2.2 Optimisation des Enterprise Java Beans

5.2.2.1 Appels distants, EJB 1.1 vs EJB2.0

A la différence des EJB2.0, les EJB implémentés selon la spécification 1.1 ne supportent que les appels distants. En d'autres termes, ils n'ont pas de méthodes locales comme c'est le cas des EJB 2.0. Nous avons vu dans la partie 4 que les invocations distantes via le protocole RMI-IIOP pénalisent aussi bien le débit que le temps de réponse. Pour cette raison il faut utiliser la spécification 2.0 au lieu de la spécification 1.1 lorsqu'on a le choix.

A noter toutefois que certains conteneurs d'EJB comme JBOSS ou IBM WebSphere permettent de convertir les invocations distantes en appels locaux lorsque le client et l'EJB se trouvent sous la même JVM. Cette option est d'habitude activable dans les fichiers de déploiement (il faut consulter la documentation du conteneur pour plus d'informations à ce sujet).

5.2.2.2 Mettre en cache les objets retournés par les lookup JNDI

A chaque fois qu'un client invoque un EJB, le conteneur exécute une requête JNDI pour le localiser et lui affecte ensuite un pointeur pour gérer son cycle de vie via les méthodes de l'interface EJBHome. Ces opérations sont redondantes, épuisantes pour le conteneur et pénalisantes pour le temps de réponse. Il est possible de les éviter si on met en cache les objets EJBHome retournés après la première invocation effectuée par le client. L'exemple suivant montre comment mettre en cache les objets EJBHome dans un Hashmap.

```
import javax.ejb.*[214];
import javax.rmi.*;
import java.util.*;
import javax.naming.*;
public class EJBHomeCache {
        // cache home references in Hashtable
        private static Hashtable homes = new Hashtable();
        Context ctx;
   public EJBHomeCache() throws NamingExeption { }
   public static synchronized EJBHome getHome(Class homeClass) throws NamingExeption {
        EJBHome home = (EJBHome) this.homes.get(homeClass);
        if(home == null) {
           ctx = getInitialContext();
           home = (EJBHome)
PortableRemoteObject.narrow(ctx.lookup(homeClass.getName()),homeClass);
           this.homes.put(homeClass,home); }
        return home;}
   private Context getInitialContext() throws NamingException {
```

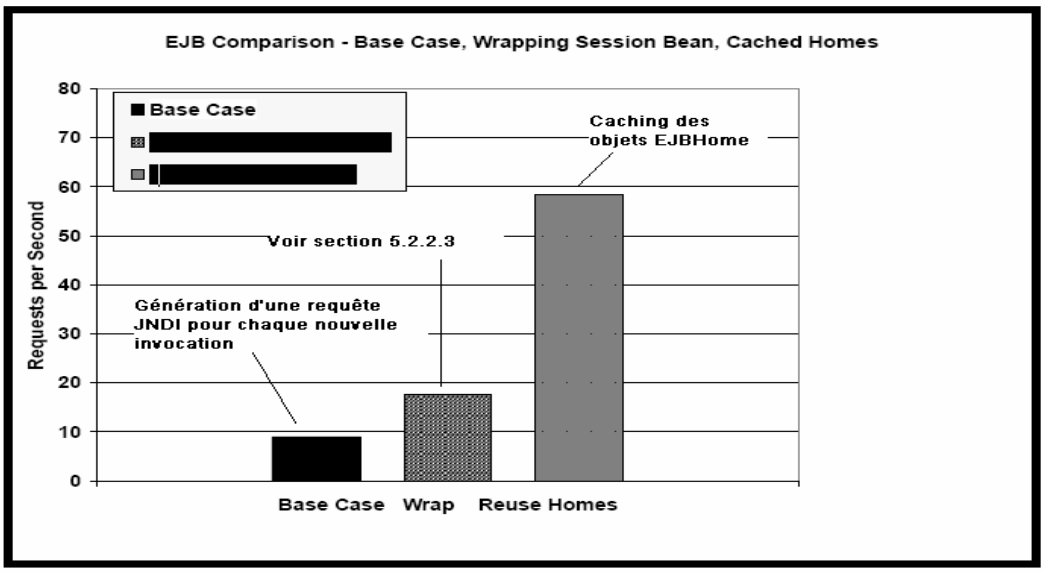
[214] Source: Ravi Kalidindi and Rohini Datla, Best practices to improve performance using Patterns in J2EE,

```
    Properties props= new Properties();
    props.put (Context.PROVIDER_URL,"urlname");
    props.put(Context.INITIAL_CONTEXT_FACTORY,"nameservice");
    return new InitialContext(props);}
} //end class
```

Pour déclencher la mise en cache après la première invocation, le client n'a qu'à appeler le bean en respectant la syntaxe suivante : NomduBean handleverslebean = (NomduBean)EJBHomeCache.getHome(NomduBean.Class);

La figure 48 montre les gains en performance que cette technique permet de réaliser sur le serveur IBM Websphere.

Figure 48: Comparatif des performances, caching des objets EJBHome vs génération d'une requête JNDI pour chaque invocation



5.2.2.3 Impact du niveau d'isolation des transactions sur les performances

Les transactions sont souvent à l'origine d'incidents qui compromettent les propriétés ACID[215] d'une base de données. Parmi ces incidents nous citons :

[215] atomicity, consistency, isolation, and durability

Dirty read : se produit lorsqu'une transaction T1 modifie le contenu d'une base de données et qu'une transaction T2 lit ce contenu avant que T1 n'effectue le commit. Après la lecture du contenu par T2, T1 effectue un rollback et le client servi initialement par T2 se retrouve avec des informations qui ne reflètent pas fidèlement le contenu de la base de données (cf. figure 49). L'exemple suivant permet de mieux comprendre cette dynamique.

1. Un client A qui a une autorisation de $1000 sur sa carte de crédit effectue un virement de $ 100 vers le compte d'un client B qui a un solde de $ 500.
2. La transaction T1 débite le compte du client A de $100 et crédite celui du client B du même montant sans effectuer un commit (le solde du client A devient $900 et celui de B devient $600).
3. Quelques millisecondes après, la transaction T2[216] est déclenchée pour calculer les intérêts sur les $100 pris par le client A.
4. En raison d'un problème technique T1 effectue un rollback pour annuler l'opération. A la suite du rollback les soldes des deux clients deviennent comme ils étaient avant le début des transactions mais le client A est toujours redevable des intérêts sur les $100.

Unrepeatable read: se produit lorsque T1 lit un contenu que T2 modifie avant que T1 n'effectue le commit (cf. figure 50).

Phantom read: se produit lorsqu'une transaction reçoit pendant son cycle de vie des réponses différentes à une même requête. Par exemple dans la figure 51 :

1. PRODUCT = A001 et COMPANY_ID = 10, avant le début des transactions.
2. T1 exécute la requête : SELECT PRODUCT WHERE COMPANY_ID = 10.
3. T2 insère un nouvel enregistrement puis effectue le commit: INSERT PRODUCT=A002 WHERE COMPANY_ID= 10.
4. T1 exécute encore une fois la requête de l'étape 2. Elle obtient des résultats différents alors que la requête n'a pas changé.

Pour prévenir ces incidents ou les laisser se produire, le gestionnaire de bases de données met en œuvre des mécanismes de verrouillage appelés niveaux d'isolation.

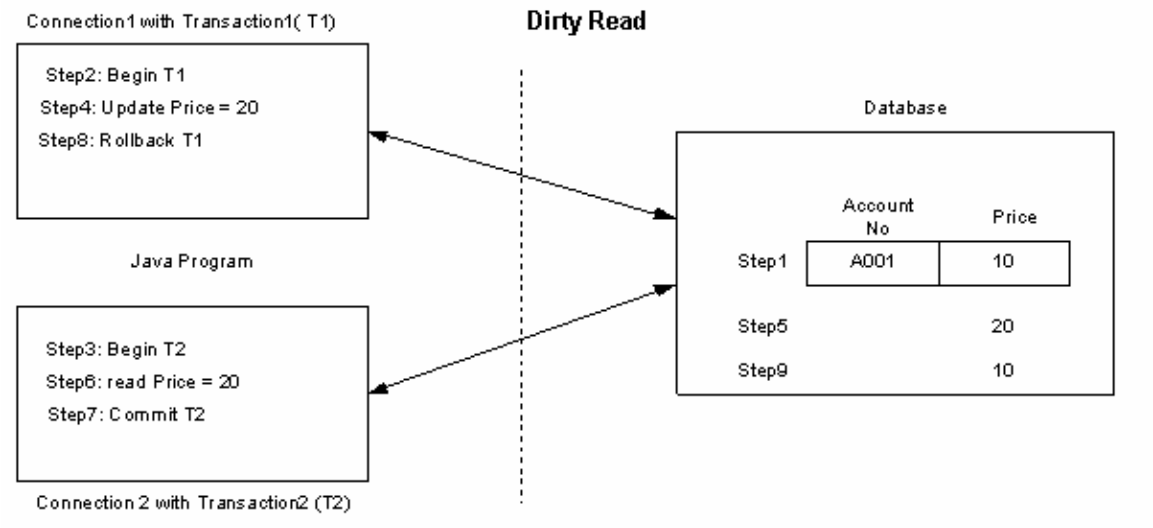
[216] Cet exemple est hypothétique. D'habitude les chaînes d'intérêts sont déclenchées à la fin du mois et le client bénéficie d'une période de grâce qui peut dépasser les 25 jours.

Ces mécanismes régissent aussi bien les opérations de lecture que les opérations d'écriture. Dépendamment de la granularité choisie et du SGBDR utilisé, ils peuvent s'appliquer :

- A une ou plusieurs lignes (enregistrements).
- A une ou plusieurs colonnes (champs ou attributs).
- Aux tables.
- A toute la base de données.

Il y a lieu de noter que le temps de réponse est d'autant plus grand que la granularité est élevée.

Figure 49: Dirty Read[217]



Dans une application qui utilise les entity beans pour interagir avec la base de données, les transactions sont traitées selon le mode déclaratif. Cela signifie que la gestion des transactions est confiée au conteneur d'EJB[218] et que les niveaux d'isolation sont définis au niveau des fichiers de déploiement et non par l'intermédiaire de l'API JTA[219] .

Le tableau 23 définit les niveaux d'isolation applicables aux transactions EJB.

[217] Source: Ravi Kalidindi and Rohini Datla, Best Practices to improve performance in EJB, http://www.precisejava.com/javaperf/j2ee/EJB.htm
[218] Se rendre à la section 4.2.3.5 pour un comparatif des performances de la persistance gérée au niveau du bean (BMP) avec celles de la persistance gérée au niveau du conteneur.
[219] Les sessions beans supportent aussi bien le mode déclaratif que l'API JTA (Java Transactions).

Figure 50: Unrepeatable read

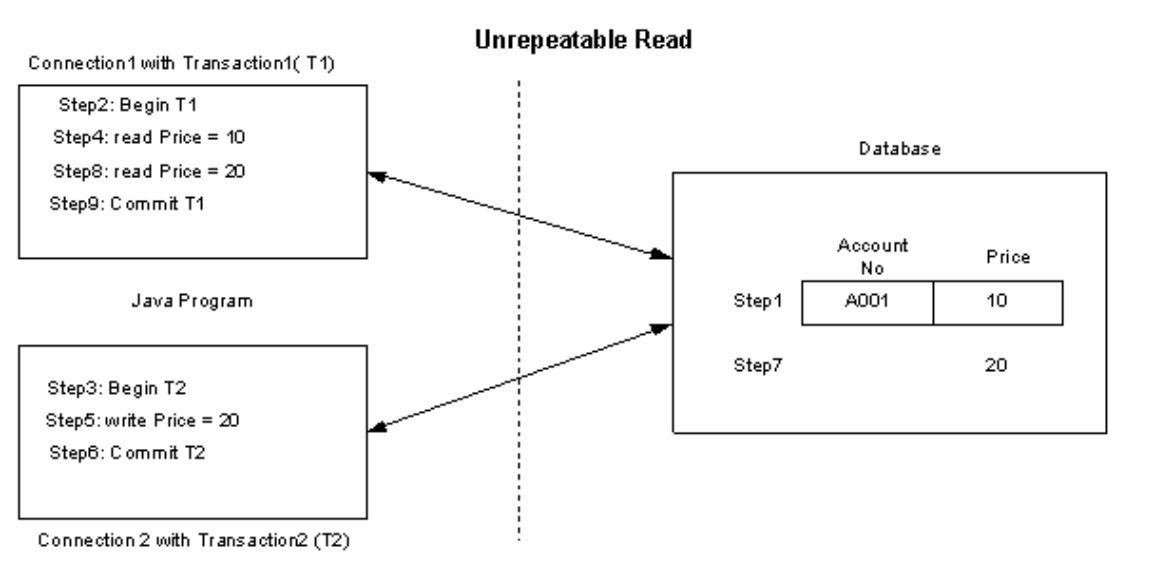


Figure 51: Phamtom reads

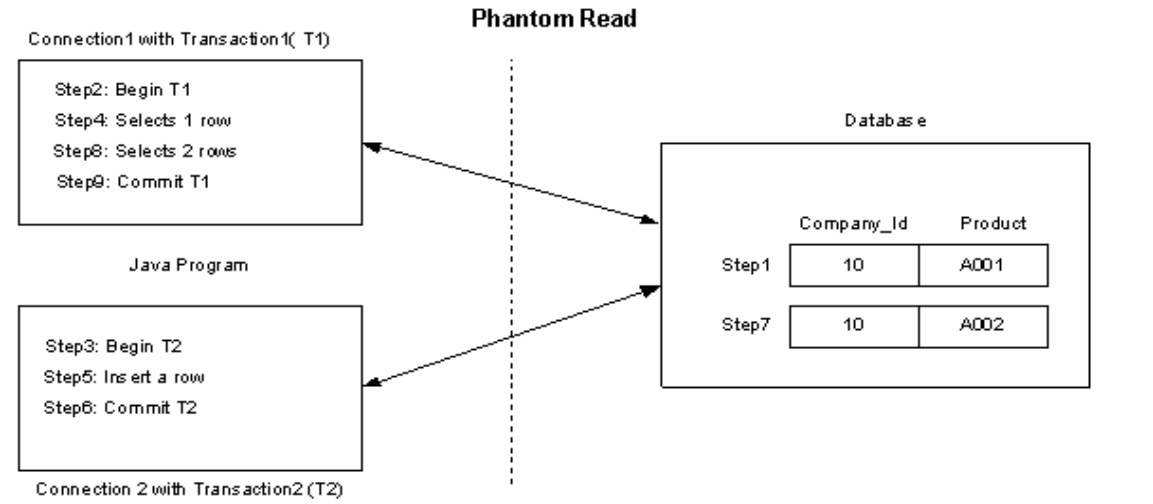


Tableau 23: Niveaux d'isolation des transactions EJB

Niveau d'isolation	Incidents		
	Dirty reads	Non Repeatable reads	Phantom reads
TRANSACTION_READ_UNCOMMITED	Permis	Permis	Permis
TRANSACTION_READ_COMMITED	Non permis	Permis	Permis
TRANSACTION_REPEATABLE_READ	Non permis	Non permis	Permis
TRANSACTION_SERIALIZABLE	Non permis	Non permis	Non permis

Il est important de noter que ce sont les niveaux d'isolation les plus permissifs qui ont le moins d'impact sur les performances (cf. tableau 24).

Tableau 24: Impact des niveaux d'isolation sur les performances

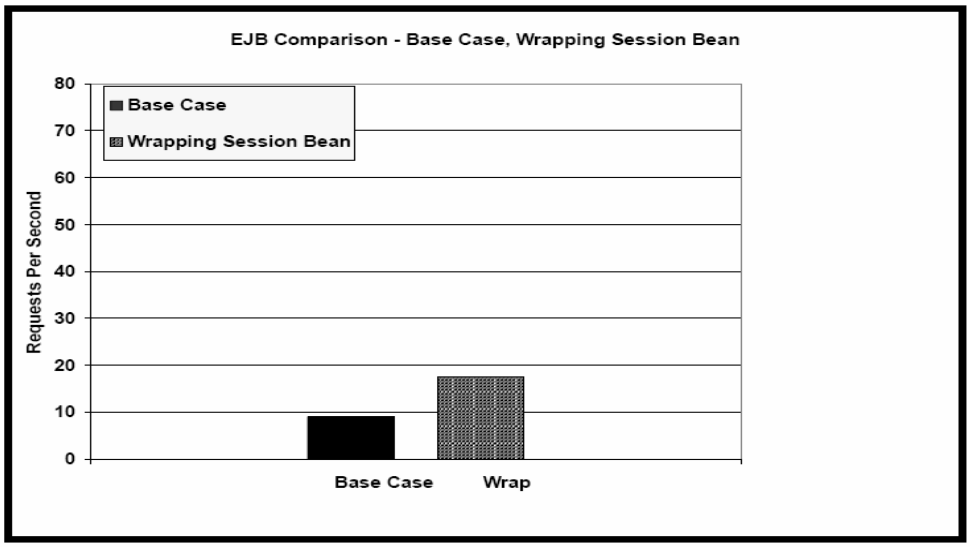
Niveau d'isolation	Performance
TRANSACTION_READ_UNCOMMITED	Très rapide
TRANSACTION_READ_COMMITED	Rapide
TRANSACTION_REPEATABLE_READ	Moyen
TRANSACTION_SERIALIZABLE	Lent

5.2.2.4 Accéder aux entity beans par l'entremise d'un session bean

Comme nous l'avons vu dans la partie 4 les entity beans sont des composants très pénalisants pour les performances d'une part parce qu'ils sont invocables via RMI-IIOP et d'autre part parce qu'ils synchronisent leurs états avec celui de la base de données sous-jacente. Notons par ailleurs que lorsque les entity beans sont invoqués directement chacune de leurs méthodes est traitée comme une transaction.

Le fait de forcer les requêtes à transiter par un session bean avant d'être déléguées (via un appel local) à l'entity bean permet d'améliorer les performances de l'application comme le montre la figure 52.

Figure 52: Comparatif des performances, accès direct à un entity bean vs accès via un session bean



5.2.3 Optimisation de JDBC

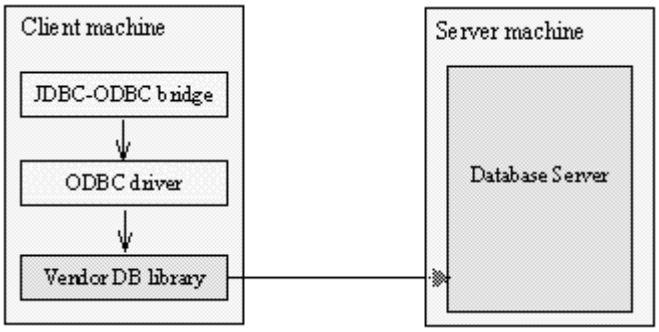
5.2.3.1 Choix du driver

Les drivers qui permettent à l'API JDBC d'interagir avec une base de données relèvent de l'une des quatre catégories suivantes:

- JDBC-ODBC Bridge
- Native-API/partly Java driver
- Net-protocol/all-Java driver
- Native-protocol/all-Java driver

Les drivers relevant de la **première catégorie** transitent par une passerelle ODBC avant d'acheminer les appels de l'API JDBC vers le gestionnaire de base de données. Parce qu'ODBC est écrit en C, ils doivent effectuer des conversions qui sont très coûteuses pour les performances. Notons par ailleurs que ces drivers doivent nécessairement être installés sur le poste du client pour qu'il puisse interagir avec la base de données (or sur le réseau Internet les développeurs n'ont aucun contrôle sur les drivers et les plug-in installés sur les postes des clients).

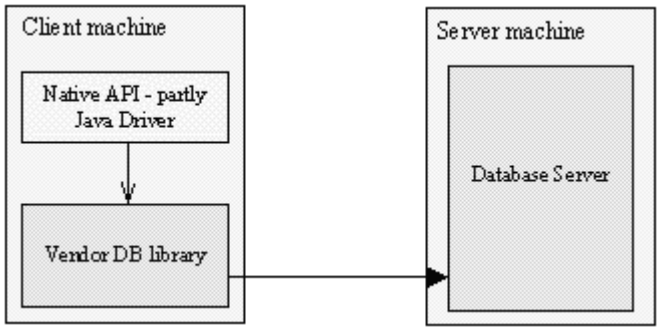
Figure 53: JDBC-ODBC bridge[220]



Les drivers de la **deuxième catégorie** convertissent les appels de l'API JDBC via une interface native (JNI) qui est supportée par le gestionnaire de base de données. Ils sont plus performants comparativement aux drivers de la première catégorie mais pour qu'ils fonctionnent correctement une partie de leur byte code doit être installée sur le poste du client (d'où le nom partly Java driver).

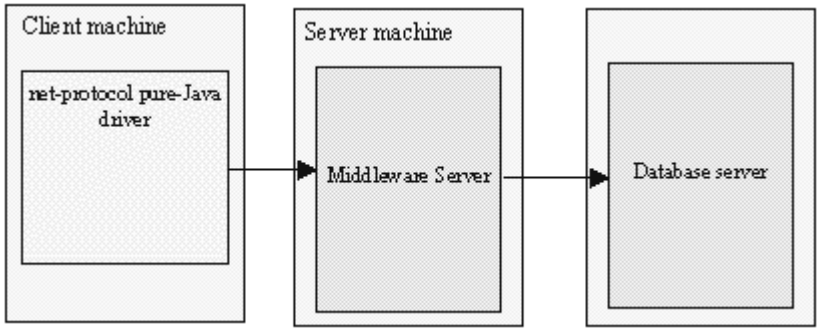
[220] Source: Nitin Nanda, Learn how to deploy, use, and benchmark JDBC driver types 1, 2, 3, and 4, http://www.javaworld.com/javaworld/jw-07-2000/jw-0707-jdbc.html

Figure 54: Native-API/ partly Java driver



Les drivers de la **troisième catégorie** sont entièrement écrits en Java et s'intègrent dans une architecture à trois tiers. Ils envoient les appels de l'API JDBC à un middleware via un protocole indépendant du gestionnaire de base de données. A réception de ces appels le middleware les convertit dans un protocole spécifique au gestionnaire de base de données. Les drivers de la catégorie 3 offrent un meilleur niveau de performance que les drivers de la catégorie 2 parce qu'ils supportent des techniques d'optimisation comme la gestion du cache et le load balancing[221]. A noter qu'avec cette catégorie, le client n'a pas besoin d'installer le driver pour pouvoir interagir avec la base de données.

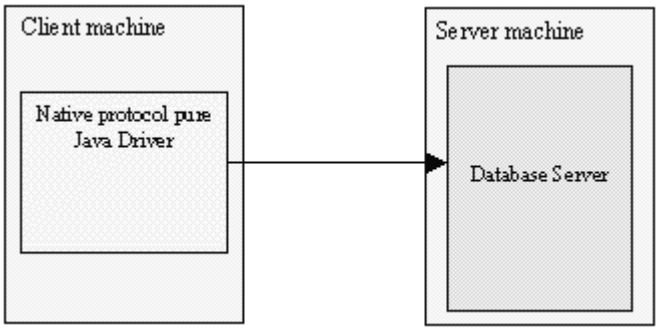
Figure 55: Net-protocol/all-Java driver



A l'instar des drivers de la catégorie 3, les **drivers de la catégorie 4** sont entièrement écrits en Java. Leur particularité c'est qu'ils reconvertissent les appels de l'API JDBC directement dans le protocole spécifique de la base de données. **Pour interagir avec MySQL via JDBC, c'est cette catégorie de drivers qui offre le meilleur niveau de performance.**

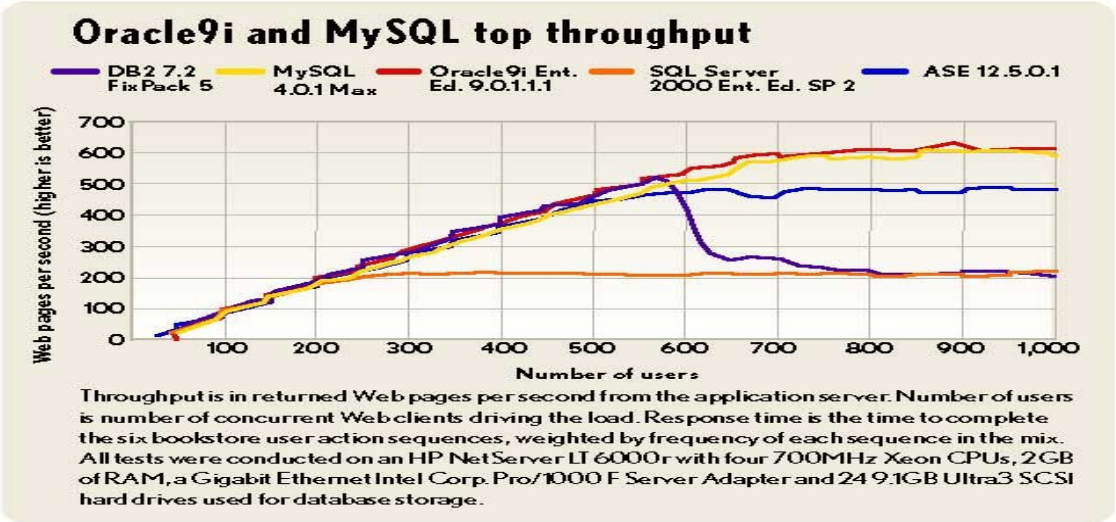
[221] Technique consistant à partager et équilibrer une charge sur plusieurs serveurs.

Figure 56: Native-protocol/all-Java driver

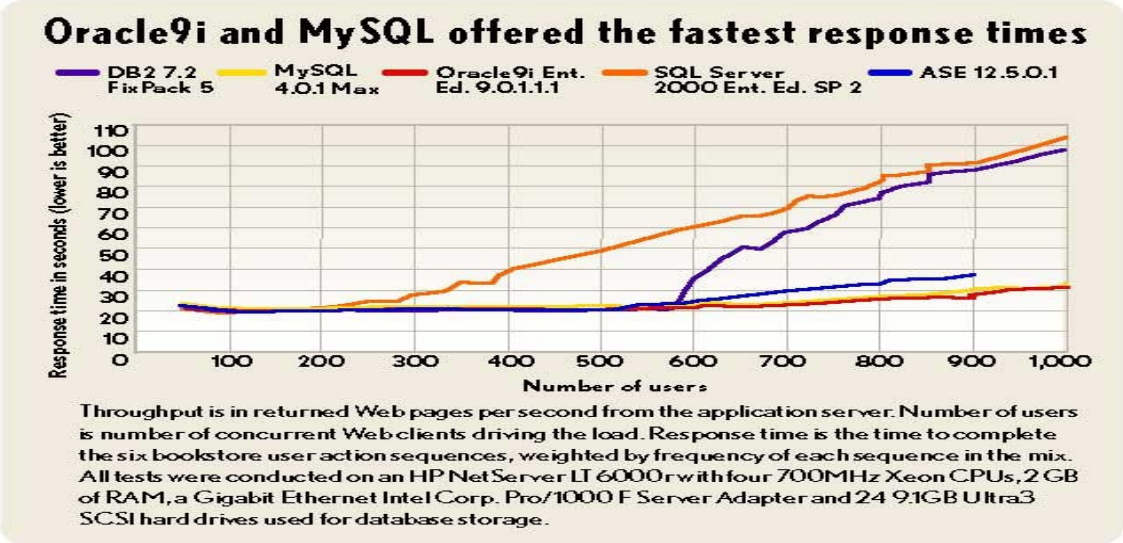


Cela dit, dans les tests de performance effectués par Ziff Davis Media Inc, c'est avec le driver Connector J[222] que MySql s'est qualifié au premier rang en ce qui concerne le débit et le temps de réponse (cf. figure 57). Notons par ailleurs que les versions récentes de ce driver sont 50 à 100 fois plus rapides que les versions initiales. De plus elles supportent le mode streaming qui permet aux requêtes de renvoyer un nombre très important d'enregistrements sans consommer excessivement la mémoire tampon (d'après le site www.mysql.com le driver est capable de renvoyer jusqu'à 2 gigabits d'enregistrements).

Figure 57: Database benchmark/ Ziff Davis Media Inc[223]



[222] Nous rappelons que Connector J fait partie des drivers de la catégorie 4.
[223] Source: Eweek, Database benchmarking, http://www.eweek.com



5.2.3.2 Connection pooling

Comme nous l'avons vu dans la partie 4, un développeur ne peut pas se permettre de créer autant de connexions qu'il y a de requêtes entrantes. De la même manière, il ne peut pas partager une seule connexion entre toutes les requêtes au risque de condamner les traitements parallèles et les accès concurrentiels à la base de données.

Un connection pool contribue à l'amélioration des performances en partageant un nombre restreint de connexions déjà créées entre les différentes requêtes reçues par l'application.

Dans la suite de cette section nous donnerons un exemple qui montre comment implémenter les connection pools dans une application qui utilise Tomcat comme conteneur, Mysql comme gestionnaire de base de données et Connector J comme driver.

Dans Tomcat les connection pools sont traditionnellement implémentés via les classes DBCP de Jakarta Commons[224]. Cependant il est aussi possible d'utiliser n'importe quel autre pool qui implémente l'interface javax.sql.DataSource. Les étapes pour implémenter les connection pools dans Tomcat sont décrites ci-après[225] :

[224] Site du projet: http://jakarta.apache.org/commons/
[225] Le code source pour tester cet exemple est disponible à l'adresse suivante :
http://christophej.developpez.com/fichiers/pooltomcat/TutoPool.zip

- Le driver Connector J et les fichiers JAR des packages Jakarta-Commons DBCP 1.0, Jakarta-Commons Collections 2.0 et Jakarta-Commons Pool 1.0 doivent être installés dans le répertoire $CATALINA_HOME/common/lib.
- Configuration du fichier de déploiement web.xml pour spécifier le nom JNDI du DataSource (dans notre exemple, la servlet s'appelle 'TutoPool' elle fait partie d'un package nommé 'tutorial' et la base de données sous-jacentes s'appelle 'base').

Fichier de déploiement web.xml[226]

```
<?xml version="1.0" encoding="UTF-8"?>[227]
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
    <display-name>TutoPool</display-name>
    <servlet>
        <servlet-name>TutoPool</servlet-name>
        <servlet-class>tutorial.TutoPool</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>TutoPool</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
    <resource-ref>
        <description>
            Référence à la ressource BDD pour le pool
        </description>
        <res-ref-name>jdbc/TutoPool</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
    </resource-ref>
</web-app>
```

[226] Basé sur un tutoriel de Christophe Jollivet, publié à l'adresse http://christophej.developpez.com/tutoriel/j2ee/pooltomcat/
[227] Cette section doit être placée après la section servlet-mapping.

- Configurer le fichier server.xml (dans ce fichier username et password correspondent aux paramètres d'authentification d'un utilisateur qui a le droit d'accéder à la base de données).

Resources factory/ server.xml[228]

```
<Context path="/TutoPool"
        reloadable="true"
        docBase="\TutoPool" >
        <Resource
                name="jdbc/TutoPool"
                auth="Container"
                type="javax.sql.DataSource"/>

        <ResourceParams name="jdbc/TutoPool">
                <parameter>
                        <name>username</name>
                        <value>user</value>
                </parameter>
                <parameter>
                        <name>password</name>
                        <value>password</value>
                </parameter>
                <parameter>
                        <name>driverClassName</name>
                        <value>org.gjt.mm.mysql.Driver</value>
                </parameter>
                <parameter>
                        <name>url</name>
                        <value>jdbc:mysql://localhost:3306/base</value>
                </parameter>
        </ResourceParams>
</Context>
```

[228] Cette section doit être placée avec les autres déclarations de contexte entre les balises <Host> du fichier server.xml

Code de la servlet TutoPool

```
package tutorial;
import java.io.*;
import java.sql.*;
import javax.naming.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.sql.*;
public class TutoPool extends HttpServlet {
        private DataSource ds; //la source de données
        protected void doGet(
                HttpServletRequest request,
                HttpServletResponse response)
                throws ServletException, IOException {
                response.setContentType("text/html");
                PrintWriter out = response.getWriter();
                out.println("<html><head></head>");
                out.println("<body>");
                Connection con=null;
                Statement s=null;
                ResultSet rs=null;
                try { //récupération de la Connection depuis le DataSource
                        con = ds.getConnection();
                        s = con.createStatement();
                        rs = s.executeQuery("SELECT * FROM table1");
    // table1 est une table qui est supposée exister dans la base de données 'base'
                        while (rs.next()) {
                                out.println(rs.getString(1) + "      ");
                                        out.println(rs.getString(2) + "<br/>");} } catch
            (SQLException e) {response.sendError(500, "Exception sur l'accès à la BDD " +
            e);} finally { if (rs != null)
                        { try { rs.close();
                                } catch (SQLException e) {}
                                rs = null; }
                        if (s != null) { try { s.close();
                                } catch (SQLException e) {}
```

```
                    s = null;}
                if (con != null) {
                    try { con.close();} catch (SQLException e) {}
                    con = null;}}
        out.println("</body>");
        out.println("</html>");
        out.close();}
    public void init() throws ServletException {
        try {//récupération de la source de donnée
            Context initCtx = new InitialContext();
            ds = (DataSource) initCtx.lookup("java:comp/env/jdbc/TutoPool");
        } catch (Exception e) {
            throw new UnavailableException(e.getMessage());}
        }}
```

Pour délimiter la taille du pool, il suffit de définir les paramètres suivants dans le fichier server.xml :

- **maxActive :** le nombre maximum de connexions qui peuvent être placées dans le pool. Valeur par défaut = 8. Une valeur égale à 0 indique qu'il n' y a pas de limite au nombre de connexions stockées dans le pool (il faut éviter d'attribuer cette valeur à maxActive parce qu'elle pénalise les performances beaucoup plus qu'une application qui n'utilise pas du tout le connection pooling).
- **maxIdle :** le nombre maximum de connexions inactives qui peuvent être tolérées par le pool. Sa valeur par défaut est 8. Il n'y a pas de limite lorsque la valeur=0.
- **maxWait:** le temps d'attente maximum en millisecondes pour récupérer une connexion du pool. Passé ce délai, une exception est générée. La valeur par défaut (-1) correspond à un temps infini.

Dans la servlet TuroPool, la fermeture de la connexion avec la méthode close() ne provoque pas sa destruction parce que la méthode s'applique à une connexion logique et non à la connexion physique. L'appel de la méthode close() permet, plutôt, au pool de récupérer la connexion pour qu'elle puisse être réaffectée à une nouvelle requête. Si le programmeur oublie de fermer la connexion celle-ci ne revient pas dans le pool et l'application risque de se bloquer faute de connexions disponibles.

Le conteneur Tomcat peut être configuré pour replacer automatiquement les connexions dans le pool au cas ou le programmeur aurait oublié de les fermer. Cette configuration se fait également dans le fichier server.xml par le biais des paramètres suivants :

- **removeAbandoned :** si sa valeur est égale à true la connexion est considérée comme abandonnée (et donc éligible à la récupération) si elle est inactive pendant un délai qui est supérieur à celui du paramètre removeAbandonedTimeout.
- **removeAbandonedTimeout :** définit en secondes le temps avant qu'une connexion inactive ne soit considérée comme abandonnée (sa valeur par défaut est 300).
- **logAbandoned :** lorsque sa valeur est égale à true, Tomcat ajoute une trace de la pile de l'application qui a abandonné la connexion.

Le fichier suivant montre comment ajuster les paramètres énumérés ci-dessus dans le fichier server.xml.

```
<parameter>[229]
                <name>maxActive</name>
                <value>8</value></parameter>
        <parameter> <name>maxIdle</name> <value>8</value>
        </parameter>
        <parameter>
                <name>maxWait</name>
                <value>10000</value>
        </parameter>
        <parameter> <name>validationQuery</name>
                <value>select 1</value>
        </parameter>
        <parameter>
                <name>removeAbandoned</name>
                <value>true</value>
        </parameter>
        <parameter>
                <name>removeAbandonedTimeout</name>
                <value>20</value>
```

[229] Basé sur un tutorial de Christophe Jollivet, publié à l'adresse http://christophej.developpez.com/tutoriel/j2ee/pooltomcat/

```
    </parameter>
    <parameter>
            <name>logAbandoned</name>
            <value>true</value>
</parameter>
```

5.2.3.3 Procédures stockées

Les procédures stockées contiennent du code SQL (pré)compilé. Elles permettent d'exécuter des requêtes complexes ou des instructions SQL régulièrement utilisées. Contrairement aux requêtes simples, les procédures stockées résident dans la base de données. Pour les déclencher on a pas besoin de transporter le code SQL du client vers le serveur. Parce qu'elles sont conservées dans la base de données sous une forme exécutable, il suffit de les appeler un peu comme on le ferait avec une fonction ou une procédure dans un langage structuré.

A l'instar des méthodes d'objets, les procédures stockées sont capables de recevoir des paramètres d'entrée et de retourner des valeurs en sortie. Cela permet de réduire considérablement le trafic entre le client et le serveur en limitant les données échangées aux appels des procédures et à leurs paramètres.

Notons par ailleurs que l'exécution d'une procédure stockée est plus rapide que celle d'une requête normale parce qu'elle profite de la puissance du serveur qui est généralement mieux nanti que le poste client. Un autre avantage des procédures stockées a trait à la facilité de mise à jour et de déploiement. En effet parce que le code SQL est centralisé les modifications apportées à la procédure stockée sont automatiquement répercutées sur tous les clients qui l'appellent.

Pour appeler une procédure stockée via JDBC, il faut obtenir une référence vers un objet de type CallableStatement par l'intermédiaire de la méthode prepareCall de l'interface Connection. Cette méthode reçoit un paramètre de type String qui correspond au nom de la procédure stockée précédé du mot clé call.

Nous présenterons dans le reste de cette section un exemple simpliste pour montrer comment appeler une procédure stockée à partir de JDBC. Nous reviendrons toutefois sur les procédures stockées avec plus de détails dans la section 5.3

Sous MySql 5.0 les instructions suivantes permettent de créer une procédure stockée appelée SHOW_SUPPLIERS(). Ces instructions sont conformes à la norme SQL:2003 qui est aussi utilisée dans DB2.

```
create procedure SHOW_SUPPLIERS ()
SELECT SUPLLIERS.NAME_SU, CAFE.NAME_CAFE
FROM SUPPLIERS, CAFE
WHERE SUPPLIERS.SU_ID = CAFE.SU_ID
order by NAME_SU
```

La procédure SHOW_SUPPLIERS() sera compilée et stockée dans la base de données comme un objet qui peut être appelé de façon similaire à l'appel d'une méthode. Pour appeler SHOW_SUPLLIERS() à partir de JDBC il faut créer un objet CallableStatement. L'objet CallableStatement contient l'appel d'une procédure et non la procédure elle-même. Dans le code ci-dessous, les instructions entre accolade appellent la procédure stockée. Quand le driver rencontre "{call SHOW_SUPPLIERS}", il traduit ces instructions dans le code SQL natif utilisé par la base de données pour appeler la procédure stockée nommée SHOW_SUPPLIERS(). La méthode executeQuery de l'objet cs renvoie le résultat de la procédure stockée dans un ResultSet.

```
CallableStatement cs = conn.prepareCall("{call SHOW_SUPPLIERS}");
ResultSet rs = cs.executeQuery();
```

5.2.3.4 Prepared Statement

Lorsqu'une requête de type Statement est exécutée le gestionnaire de base de données déclenche en arrière plan les opérations suivantes :

- Parsing des requêtes pour vérifier qu'elles respectent la syntaxe.
- Trouver le meilleur algorithme pour effectuer la recherche.
- Trouver l'emplacement des index.
- Trouver l'emplacement des tables.
- Trouver les enregistrements qui répondent aux critères de la requête.

La charge de travail engendrée par ces opérations peut être réduite grâce aux requêtes précompilées appelées Prepared Statements dans le jargon des bases de données. Lorsque ces requêtes sont exécutées pour la première fois elles sont compilées à l'instar des requêtes Statement. Cependant, de la deuxième à la nème fois le gestionnaire de base de données ne les recompile pas. Il se contente de rechercher directement les enregistrements qui répondent à leurs critères. En fait les Prepared Statements sont un dispositif qui permet de « préparer » une requête une fois pour toute, puis de l'exécuter plusieurs fois avec des paramètres différents. Pour qu'elles contribuent à l'amélioration des performances, les deux conditions suivantes doivent être réunies :

- La structure de la requête reste la même pendant l'exécution de l'application.
- Seuls les paramètres de la requête changent (dans l'exemple ci-dessous les points d'interrogations correspondent aux paramètres de la requête).

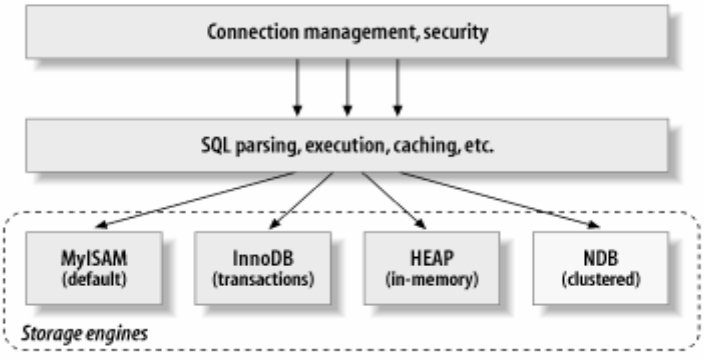
```
PreparedStatement updateSales = con.prepareStatement(
    "UPDATE COFFEES SET SALES = ? WHERE COF_NAME LIKE ?");
```

5.3 Optimisation de Mysql

5.3.1 Performance des moteurs de stockage

Dans cette section nous présenterons rapidement l'architecture de MySQL en polarisant l'attention sur les moteurs de stockage et leur impact sur les performances.

Figure 58: Architecture du serveur MySQL[230]



[230] Source: Jeremy Zawodny, Derek J. Balling, High Performance MySQL, O'Reilly, April 2004, ISBN: 0-596-00306-4

- Dans la figure 58, la couche supérieure offre des services relatifs à la gestion des connexions, de l'authentification et de la sécurité.
- La couche intermédiaire contient les fonctionnalités communes du SGBDR (logique permettant le parsing des instructions SQL, procédures stockées, cache, utilitaires d'optimisation…).
- La troisième couche contient les moteurs de stockage qui permettent de sauvegarder les données et de les rendre persistantes. C'est cette couche qui nous intéresse dans cette section.
- L'interface entre la deuxième et la troisième couche est une API indépendante du moteur de stockage utilisé. Elle se compose d'une vingtaine d'instructions de bas niveau qui permettent d'agir directement sur le contenu des moteurs de stockage (exemple récupérer d'une table l'enregistrement qui a une clé primaire donnée, commencer une transaction…).

Les moteurs de stockage ne communiquent pas les un avec les autres et ne traitent pas les instructions SQL. Ils se contentent de répondre aux appels qu'ils reçoivent de la couche intermédiaire. Comme le montre la figure 58, MySQL contient plusieurs moteurs de stockage qui peuvent être transactionnels ou non transactionnels.

- MyISAM et le moteur par défaut de MySQL. Il s'agit d'un moteur non transactionnel qui a été initialement conçu avec l'hypothèse que les applications font majoritairement appel à des opérations de lecture.
- Le moteur HEAP (appelé aussi MEMORY) propose des tables stockées en mémoire. Le moteur MERGE, qui n'apparaît pas sur la figure 58, a été ajouté dans la version 3.23.25. Il permet de fusionner des tables MyISAM identiques dans une seule table. A l'instar de MyISAM ces deux moteurs sont également non transactionnels.
- Le moteur InnoDB gère des tables transactionnelles. Il a été introduit dans la version 3.23.
- Le moteur NDB clustered gère les tables réparties sur plusieurs serveurs. Il a été introduit dans la version 4.1.2.

- Les tables BDB, qui n'apparaissent pas sur la figure 58, sont stockées sur le disque en deux fichiers. Le premier, qui a l'extension .frm, stocke la définition de la table. Le deuxième a l'extension .db et contient les données et les index.

Lors de la création d'une table, il faut indiquer à MySQL quel moteur de stockage utiliser. Cela se fait par le biais des instructions ENGINE ou TYPE comme le montre l'exemple suivant:

```
CREATE TABLE t (i INT) ENGINE = INNODB;
CREATE TABLE t (i INT) TYPE = MEMORY;
```

Lorsque le moteur de stockage n'est pas précisé ou n'est pas activé, c'est le type par défaut qui est utilisé (MyISAM). Pour changer le type d'une table déjà créée, il faut utiliser la commande ALTER TABLE, comme le montre l'exemple suivant :

```
ALTER TABLE t ENGINE = MYISAM;
ALTER TABLE t TYPE = BDB;
```

Il est important de rappeler que les tables non transactionnelles sont plus rapides à traiter que les tables transactionnelles. De plus elles utilisent moins d'espace disque et moins de mémoire pour les mises à jour. Notons par ailleurs que :

- Les tables MyISAM sont optimisées pour supporter des *mixes* composés de 90% d'opérations de lecture. Elles donnent de bonnes performances tant que le ratio read to write reste en deçà de ce pourcentage[231].
- Les tables MERGE utilisent massivement les pointeurs de fichiers. Par exemple une table MERGE qui couvre 10 autres tables et qui est utilisée par 10 personnes consomme 10*10 + 10 pointeurs de fichiers (10 fichiers de données et 10 utilisateurs avec 10 fichiers d'index).
- Les opérations de lecture sont très lentes avec les tables de type MERGE (pour lire une clé donnée le gestionnaire MERGE doit parcourir tous les fichiers d'index des tables sous-jacentes pour vérifier lequel est le plus proche de la clé recherchée).
- Les tables HEAP sont conservées en mémoire. Elles sont très rapides mais elles ne persistent pas en cas de plantage ou de redémarrage du serveur.

[231] Source: Jeremy Zawodny, Derek J. Balling, High Performance MySQL, O'Reilly, April 2004, ISBN: 0-596-00306-4

- Avec un index hash sur une table HEAP et un haut degré de duplication les suppressions sont plus lentes. Le facteur de ralentissement est proportionnel au degré de duplication (ou inversement proportionnel à la cardinalité). Notons cependant que depuis la version 4.1, MySQL supporte les indexes BTREE qui permettent de remédier à cette lenteur. Nous reviendrons avec plus de détails sur les index dans la section 5.3.3.

5.3.2 Verrouillage, concurrentialité et performance

Dans la section 5.2.2.3 nous avons analysé l'impact des niveaux d'isolation sur les performances des entity beans. Pour que cette analyse soit complète, il est nécessaire de l'étendre aux mécanismes de verrouillage utilisés par la base de données à laquelle ils sont adossés.

Les mécanismes de verrouillage de MySQL différent selon le type de table utilisé.

Dans les tables de type MyISAM et HEAP, seul le verrouillage de niveau table est supporté. Cela pourrait être pénalisant pour les traitements parallèles et la concurrentialité des applications qui font majoritairement appel à des opérations d'écriture. Or cela n'est pas le cas de la plupart des applications de commerce électronique qui comme nous l'avons vu dans la partie 3 font de nombreuses sélections, peu d'effacements, des modifications et des insertions basées sur des clés. Les tables MyISAM sont donc bien optimisées pour ce genre d'application.

Les tables de type InnoDB supportent le verrouillage de niveau ligne. Elles offrent un meilleur degré de concurrentialité mais elles consomment plus de mémoire que les verrous de table. Notons par ailleurs que les verrous de niveau ligne:

- Sont plus lents que les verrous de table surtout lorsqu'ils sont appliqués à une grande portion de la table.
- Donnent de piètres résultats avec les requêtes Group By ou tout autre requête qui scanne l'intégralité d'une table.
- Ont un coût d'administration qui est supérieur à celui des verrous de table.

Cela dit, les verrous de table sont plus performants que les verrous de ligne dans les cas suivants :

- Les opérations de lecture.
- Les opérations de modification portant sur des clés précises (comme celles de l'exemple ci-dessous).

  UPDATE table_name SET column=value WHERE unique_key#
  DELETE FROM table_name WHERE unique_key=#

- De nombreux scans / GROUP BY sur toute la table, sans aucune écriture[232].

Les autres niveaux de verrouillage possibles dans MySQL sont :

- Les verrous de niveau page : disponibles uniquement avec les tables de type BDB leur performance est tributaire de la taille de la page (ensemble constitué par deux ou plusieurs lignes). Avec ces verrous la concurentialité est d'autant plus faible que la taille des pages est grande.
- Le versionage : un thread qui écrit et de nombreux autres qui lisent. Dans ce cas Les verrous s'appliquent aux opérations d'écriture et non aux opérations de lecture.

Le tableau 59 résume la relation entre la performance, les mécanismes de verrouillage, les moteurs de stockage et la concurentialité.

Figure 59: Modèle de verrouillage et de concurrentialité dans MySQL

Verrouillage	Concurentialité	Charge de travail	Moteur de stockage
Table	La plus faible	Faible	MyISAM, Heap, Merge
Ligne / page	Moyenne	Moyenne	InnoDB /BDB
Versionage	La plus élevée	Elevée	InnoDB

5.3.3 Les indexes

Les analogies qui existent entre un annuaire téléphonique et une base de données permettent de mieux comprendre ce que sont les indexes.

[232] Source: Méthodes de verrouillage MySQL, http://www.sourcekeg.co.uk/www.mysql.com/doc/mysql/fr/internal-locking.html

Supposons que l'on souhaite rechercher le numéro de téléphone d'une personne dans une table qui contient près de 30 millions d'abonnés. Si on exécute une requête comme SELECT * FROM phonebook WHERE last_name = 'person', le gestionnaire de base de données sera obligé de parcourir tous les enregistrements de la table avant de retourner le résultat. Cette technique est bien sûr inefficace. Elle dégrade les performances à mesure que le nombre d'enregistrements croît (problème classique du O(n)).

D'habitude lorsqu'on souhaite effectuer une recherche dans un annuaire papier, on se positionne directement sur le groupe de pages qui commencent avec la même lettre que celle du nom/prénom de la personne recherchée. Ce positionnement est facilité par le classement alphabétique des pages. Il permet de réduire considérablement le nombre de pages à compulser.

Les index des bases de données fonctionnent de la même manière. Par exemple si on souhaite effectuer les recherches plus rapidement sur la base du prénom on peut appliquer un index à la colonne correspondante en utilisant l'instruction ALTER :

ALTER TABLE phonebook ADD INDEX (last_name)

Après l'exécution de cette instruction, MySQL crée une liste ordonnée de tous les prénoms de la table phonebook. Avec les tables MyISAM, les indexes sont stockés dans des fichiers à part pour rendre les recherches plus rapides (en fait si la table contient 1000 lignes, la recherche sera 100 fois plus rapide qu'une lecture séquentielle[233]).

Il est important de noter que la création d'indexes consomme beaucoup d'espace mémoire dans la base de données. De plus leur mise à jour à chaque modification de la table[234] peut augmenter le temps nécessaire pour réaliser des opérations d'écriture. De ce fait, il faut que la création d'indexes soit justifiée et que les colonnes auxquelles ils sont appliqués soient judicieusement choisies.

Outre la célérité des recherches, les indexes permettent également d'optimiser la performance des opérations suivantes :

- Lire des lignes dans d'autres tables lorsqu'il y a des jointures.

[233] Source: http://www.manuelphp.com/mysql/mysql-indexes.php
[234] INSERT , UPDATE , REPLACE , or DELETE

- Trouver les valeurs MAX() et MIN() dans une colonne.
- Trier ou grouper des lignes dans une table (exemple ORDER BY key_part_1,key_part_2).

Dans MySQL il existe plusieurs types d'indexes. Les plus utiles sont :

- **Les indexes partiels :** ils permettent de réduire la quantité de mémoire consommée par la création des indexes. Ainsi au lieu d'appliquer l'index à tous les caractères d'un nom/prénom il est possible de l'associer à certains caractères seulement. Dans l'exemple suivant, seuls les 4 premiers octets du prénom sont utilisés pour la création de l'index.

  ALTER TABLE phonebook ADD INDEX (last_name(4)).

  Après l'application de cet index une requête comme SELECT * FROM phonebook WHERE last_name = 'Jackie', retournera des résultats comme Jacky, Jackline, Jackie, Jackass. Par conséquent il faudra exécuter plus d'une requête pour retourner des résultats précis (Il s'agit donc de trouver une solution de compromis entre le nombre de requêtes à exécuter, le temps de réponse et la quantité de mémoire consommée par les index).

- **Les indexes multi-colonnes :** MySQL permet d'appliquer les indexes à plusieurs colonnes. Exemple :

  ALTER TABLE phonebook ADD INDEX (last_name, first_name)

  Ce genre d'index permet d'améliorer considérablement la vitesse d'exécution des requêtes qui portent sur toutes les colonnes indexées (dans notre exemple ces indexes donnent de bonnes performances avec une requête comme SELECT * FROM phonebook WHERE last_name = 'Jhon' AND first_name = 'Pizarelli').

  Bien sûr il est possible de combiner les indexes partiels avec les indexes multi-colonnes pour réduire la quantité de mémoire consommée :

  ALTER TABLE phonebook ADD INDEX (last_name(4), first_name(4))

- **Les indexes FULLTEXT :** ils sont utilisés pour les recherches en texte intégral uniquement avec les tables MyISAM. L'indexation se fait sur toute la largeur des colonnes de type CHAR, VARCHAR et TEXT. En règle générale, il faut toujours charger les données dans une table avant de leur appliquer les indexes FULLTEXT. La raison est que le chargement et l'écriture dans une table déjà indexée en FULLTEXT peuvent être anormalement longs.

5.3.4 Concordances entre les types de données

Les types de données de Java et de MySQL ne portent pas les mêmes noms. Cela cause souvent des erreurs dans le choix des types de données aussi bien au niveau du langage de programmation qu'au niveau du SGBDR. Si les types choisis ne sont pas concordants, ils peuvent devenir une véritable source de dégradation des performances.

Dans l'annexe 3, nous avons reproduit des tableaux de correspondance qui permettront aux équipes de développement de faire face à la perplexité à laquelle elles sont souvent confrontées lorsqu'il s'agit de trouver des équivalences entre les types utilisés par MySQL et ceux utilisés par Java et l'API JDBC.

5.3.5 Les procédures stockées

Les procédures stockées sont de nouvelles fonctionnalités introduites dans la version 5.0 de MySQL. Au moment où nous avons rédigé cet essai, cette version était encore en phase de test. Les procédures stockées fournissent un gain de performances parce que moins d'informations sont échangées entre le serveur et le client. Le revers de la médaille est que la charge de travail du serveur augmente parce qu'il centralise et exécute tout le code SQL. Les situations dans lesquels les procédures stockées sont utiles sont :

- Des applications clientes écrites dans différents langages et tournant sur différentes plateformes (celles-ci peuvent utiliser les procédures stockées comme point d'interaction).
- Des applications où les opérations et les données traitées par les procédures stockées sont confidentielles (par exemple sur le site d'une banque les procédures stockées permettent de s'assurer que les utilisateurs n'ont aucun accès direct aux tables).

Dans MySQL 5.0 les procédures stockées nécessitent la création d'un type de table spécial appelé proc. Cette table est créée automatiquement durant l'installation de MySQL 5.0 mais il est possible de convertir et mettre à jour des tables déjà existantes si on souhaite migrer d'une version antérieure (la procédure de mise à jour est décrite en détail à l'adresse : http://dev.mysql.com/doc/mysql/fr/upgrading-grant-tables.html).

Comme nous l'avons déjà vu dans la section 5.2.3.3, les procédures stockées sont créées avec la commande CREATE PROCEDURE. Leur appel se fait à l'aide de la commande CALL.

Syntaxe

CREATE PROCEDURE sp_name ([parameter[,...]])
[characteristic ...] routine_body

La liste de paramètres entre parenthèses est obligatoire. S'il n'y a pas de paramètres, une liste vide doit être quand même utilisée (voir exemple).

```
create procedure SHOW_SUPPLIERS ()
SELECT SUPLLIERS.NAME_SU, CAFE.NAME_CAFE
FROM SUPPLIERS, CAFE
WHERE SUPPLIERS.SU_ID = CAFE.SU_ID
order by NAME_SU
```

Pour renommer une procédure stockée ou changer ses caractéristiques, il faut utiliser la commande ALTER PROCEDURE.

Syntaxe :

ALTER PROCEDURE sp_name [characteristic ...]

characteristic:
NAME newname
| SQL SECURITY {DEFINER | INVOKER}
| COMMENT string

Pour effacer une procédure stockée, il faut utiliser la commande drop suivie de son nom.

Syntaxe :

DROP PROCEDURE [IF EXISTS] sp_name

La procédure à suivre pour appeler une procédure stockée à partir de JDBC est décrite dans la section 5.2.3.3

Pour plus de détails sur les procédures stockées consulter la documentation disponible à l'adresse : http://dev.mysql.com/doc/mysql/fr/stored-procedure-syntax.html .

5.3.6 Optimisation des requêtes

5.3.6.1 Rendre les requêtes SELECT … WHERE plus rapides

L'utilisation des indexes reste de loin la meilleure technique pour optimiser les performances des requêtes de type SELECT…. WHERE. Notons par ailleurs que MySQL dispose de plusieurs utilitaires qui permettent de suivre de plus près l'exécution de ces requêtes et de comprendre les traitements effectués en arrière plan. Parmi ces utilitaires nous pouvons citer :

- **EXPLAIN :** à l'aide d'EXPLAIN, il est possible d'identifier les indexes à ajouter pour accélérer les commandes SELECT (pour plus de détails au sujet d'EXPLAIN visiter le site de MySQL à l'adresse : http://dev.mysql.com/doc/mysql/fr/explain.html)

- **ANALYZE TABLE :** cet utilitaire analyse et stocke la clé de distribution des tables MyISAM et BDB. MySQL utilise les clés de distribution pour décider dans quel ordre les tables doivent être rassemblées lors des jointures qui ne s'effectuent pas sur une constante (pour plus de détails sur ANALYSE TABLE, visiter le site de MySQL à l'adresse : http://dev.mysql.com/doc/mysql/fr/analyze-table.html).

5.3.6.2 Performance des requêtes SELECT et ordre des jointures

Lorsqu'une requête porte sur plusieurs tables, l'ordre dans lequel ces tables sont parcourues est un facteur qui affecte directement les performances. Par exemple dans la requête infra, il faut parcourir la table customer avant la table order et la table region (le but final étant de réduire au strict minimum le nombre d'enregistrements parcourus).

Pour MySQL la détermination de l'ordre dans lequel les tables doivent être scannées n'est pas aussi simple parce qu'il doit, au préalable, identifier toutes les combinaisons possibles. Cette identification est très pénalisante pour les ressources et consomme plus de temps que l'exécution de la requête elle-même. En fait l'identification des combinaisons possibles peut prendre jusqu'à 29 fois plus de temps qu'il faut pour exécuter la requête[235].

```
SELECT customer.name, order.date_placed, region.name
FROM customer, order, region
WHERE order.customer_id = customer.id
AND customer.region_id = region.id
AND customer.name = 'Rajaz Camel'
```

Cela dit, pour réduire le délai d'exécution des requêtes portant sur plusieurs tables, on peut procéder de la manière suivante:

- Utiliser l'utilitaire EXPLAIN pour déterminer l'ordre dans lequel MySQL scanne les tables.
- Si cet ordre n'est pas efficient, le forcer à scanner les tables dans un ordre précis en utilisant l'instruction STRAIGHT_JOIN (par exemple si on exécute la requête SELECT * FROM table1 STRAIGHT_JOIN table2 WHERE ... on force MySQL à scanner la table1 avant la table 2).

5.3.6.3 Query cache

Le Query cache[236] sauvegarde le texte d'une requête avec le résultat qui a été renvoyé au client. Si une requête identique est appelée par la suite, MySQL retournera le résultat à partir du cache au lieu d'exécuter la requête à nouveau. Il est important de noter que le cache de requêtes ne retourne pas des données expirées parce qu'il est systématiquement effacé à chaque fois que les données mises en cache sont modifiées.

Comme nous l'avons vu dans la partie 4 le cache est extrêmement utile lorsque le contenu des tables change peu et qu'il y a une série de requêtes identiques à exécuter.

[235] Source: Jeremy Zawodny, Derek J. Balling, High Performance MySQL, O'Reilly, April 2004, ISBN: 0-596-00306-4
[236] Le Query cache n'est disponible que dans les versions 4.0.1 et plus

Les gains en performance peuvent atteindre 238% pour effectuer des recherches sur une colonne[237]. Pour que MySQL aille chercher les résultats d'une requête dans le cache, il faudrait que le texte de celle-ci soit parfaitement identique à celui d'une requête déjà traitée. Aucune différence, aussi mineure soit elle, n'est tolérée.

Exemple : pour MySQL les deux requêtes suivantes ne sont pas identiques lorsqu'il s'agit de chercher la réponse dans le cache.

```
SELECT * FROM tbl_name
Select * from tbl_name
```

Avec les requêtes SELECT le développeur peut préciser s'il souhaite que le résultat soit mis en cache. Ainsi :

- Une requête comme SELECT SQL_CACHE id, name FROM customer met en cache le résultat.
- Par contre la requête SELECT SQL_NO_CACHE id, name FROM customer ne met pas le résultat en cache.

NB: pour activer le Query Cache il faut configurer le fichier my.cnf en attribuant la valeur 1 à la propriété query_cache_type.

[237] Se rendre à l'adresse: http://dev.mysql.com/doc/mysql/fr/query-cache.html pour plus de détails sur le benchmark qui a permis d'obtenir ce résultat.

6. Conclusion

Dans cet essai nous avons présenté une démarche pour l'intégration du management des performances au cycle de développement des applications de commerce électronique. Notre attention a été polarisée sur les phases dans lesquelles le management des performances est souvent négligé (notamment les phases d'analyse, de design et de codage).

La mise en œuvre de cette démarche dans un projet réel nécessite une très forte implication de la part des acteurs qui participent directement ou indirectement à l'analyse, à la conception et à la programmation de l'application. Sans cette implication il va sans dire que les besoins en performance seront relégués à un rang inférieur sur la liste des priorités (très loin derrière les besoins d'affaires). D'ores et déjà les besoins en performance doivent être considérés comme des besoins d'affaires et non comme un réglage technique de dernière minute.

Bien que cette démarche fasse état des meilleures pratiques de l'industrie, nous souhaiterions la valider dans un projet réel et l'ajuster de manière itérative au fur et à mesure de l'amélioration de notre connaissance du domaine. Nous restons, par conséquent, ouverts à toute personne ou organisation qui souhaiterait l'appliquer dans un projet réel. Dépendamment des besoins notre contribution pourrait couvrir des attributions comme la formation, la gestion du changement ou encore le développement d'un prototype pour les différents modèles d'affaires étudiés (B2C, B2B, C2C).

Un site d'accompagnement sera crée au début du mois de mai 2005 pour faciliter l'accès à ce document et recueillir les précieuses remarques et suggestions de la communauté Internet. Nous essayerons aussi de le traduire dans la langue de Shakespeare de façon à élargir son audience.

Adresse du site d'accompagnement : http://www.kamalaouda.com/ECtuning

Nous espérons que cet essai contribuera à l'enrichissement du patrimoine documentaire déjà existant et suscitera de longues réflexions sur l'importance du management des performances dans le contexte du commerce électronique.

7. Revue de la littérature pertinente

Titre	Type	Référence	Objet
Building Scalable and High-Performance Java(TM) Web Applications Using J2EE(TM)	Livre	Greg Barish, Addison Wesley, December 27, 2001, ISBN: 0-201-72956-3, pages 416	Ce livre : • Met en évidence les problèmes communément liés au développement d'applications Web performantes. • Décrit les technologies J2EE du point de vue de leur scalabilité et de leur performance. • Propose plusieurs techniques d'optimisation spécifiques aux technologies J2EE.
Database Tuning: Principles, Experiments, and Troubleshooting Techniques	Livre	Dennis Shasha and Philippe Bonnet, Morgan Kaufmann Publishers, ISBN: 1558607536, pages 415	Ce livre qui est dédié à l'optimisation des bases de données contient un chapitre entièrement consacré aux applications de commerce électronique. Il contient également une étude de cas basée sur une mission d'optimisation réelle réalisée par l'auteur pour le compte de Wall Street.
MySQL and Java Developer's Guide	Livre	Mark Matthews Jim Cole Joseph D. Gradecki, Wiley Publishing, 2003, ISBN 0-471-26923-9, pages 433	Le chapitre 14 traite des problèmes de performance liés : • Au Connector J. • Aux options du serveur. • Au RAID • Aux tables et aux requêtes. • Aux travaux batch. • Au connection pooling. • A la gestion des transactions.
Multithreaded Programming with JAVA™ Technology	Livre	Bil Lewis, Daniel J. Berg, Prentice Hall PTR, December 01, 1999, ISBN: 0-13-017007-0, pages 461	Ce livre contient plusieurs exemples qui montrent comment implémenter les techniques de programmation multi-tâches dans les applications web dynamiques et ce afin d'exploiter de façon optimale les performances des machines sur lesquelles elles sont exécutées.
MySQL™ and JSP™ Web Applications: Data-Driven Programming Using Tomcat and MySQL	Livre	James Turner, March 27, 2002, 0-672-32309-5, pages 400	Ce livre contient un chapitre dédié à l'optimisation des applications web bâties à l'aide des technologies Java et Mysql et utilisant Tomcat comme conteneur.
Mysql Bible	Livre	Steve Suehring, Wiley Publishing, ISBN 0764549324	La partie 4 de l'ouvrage traite des sujets d'optimisation avancés comme les réplications et la gestion des DNS.
SQL Performance Tuning	Livre	Peter Gulutzan, Trudy Pelzer, Addison Wesley, September 10, 2002, ISBN : 0-201-79169-2, pages 528	Il s'agit d'un recueil des meilleures pratiques en matière d'optimisation des commandes et des requêtes SQL. Le chapitre 13 propose plusieurs solutions pour accroître les performances des applications utilisant l'API JDBC.
Database Administration: The Complete Guide to Practices and Procedures	Livre	Craig S. Mullins, Addison Wesley, June 14, 2002, ISBN: 0-201-74129-6, pages 592	Un très bon manuel sur le métier de DBA et les meilleurs benchmarks en matière d'analyse, de design et d'implémentation des bases de données relationnelles.
Storage and Querying of E-Commerce Data	Article	Rakesh Agrawal Amit Somani Yirong Xu, 27th VLDB Conference, IBM Almaden Research Center	Cet article propose des solutions architecturales pour optimiser le stockage et la récupération des données dans les applications de commerce électronique.
Web Performance Tuning	Livre	Patrick Killelea, O'Reilly, October 1998, ISBN: 1-56592-379-0, pages 374	Fournit des outils pour mesurer la performance des applications web, ainsi que des patterns pour l'améliorer.

Suite

Titre	Type	Référence	Objet
Zona Research, Need for speed 2	Rapport	http://www.keynote.com/downloads/Zona_Need_For_Speed.pdf	Etude qui évalue l'impact des problèmes de performance sur le profit d'un panel représentatif d'entreprises américaines.
Zona Research, Inc., The Economic Impacts of Unacceptable Web-Site Download Speeds	Rapport	http://www.webperf.net/info/wp_downloadspeed.pdf,	Version antérieure de la même étude.
Ethan Henry, Brad Micklea, Bridging the Java™ 2, Platform Enterprise Edition (J2EE™) Technology Performance Gaps	Diapositives	JavaOne 2004	Diapositives d'une présentation de JavaOne sur les performances des applications J2EE.
Segue Software, Inc, ACHIEVING HIGH PERFORMANCE J2EE APPLICATIONS, January 2003	Article	http://productfinder.comomagazine.com/cxocm/search/viewabstract/64559/index.jsp	Décrit l'apport des logiciels Borland et Segue dans le domaine de l'évaluation et du management des performances.
Borland White Paper, Maximizing business value by optimizing J2EE™ performance	Article	http://itpapers.zdnet.com/abstract.aspx?scname=J2EE&docid=92523	Idem
Borland, ACCELERATE YOUR PERFORMANCE	Article	http://www.borland.com/optimizeit/pdf/opt6_datasheet.pdf	Idem
Thomas Malvehill, Key Challenges in developing and deploying J2EE applications, July 28, 2003	Article	http://www.veritas.com/van/articles/3533.jsp	Décrit les défis auxquels les entreprises doivent faire face lorsqu'il s'agit de développer des applications J2EE performantes.
Borland presentation on Performance management for the J2EE platform,	Diapositives	http://www.pcuf.fi/sytyke/kerhot/javasig/PerfMan_J2EE.pdf	Diapositives d'une présentation de Borland sur les performances des applications J2EE.
Teresa Lanowitz, Tearing Down the Wall	Rapport	Gartner, 2002	Rapport qui recense les problèmes de performance rencontrés par la plupart des entreprises qui ont utilisé J2EE dans l'implémentation de leurs applications.
Jon Ha Building Quality into developement., Borland softwares	Diapositives	http://www.java.no/web/moter/javazone03/presentations/JonHarrison/JH_QualityDevelopment_JavaZone.pdf	Diapositives d'une présentation de Borland sur la qualité logicielle en général et les performances en particulier.
Jakob Nielson, The Need for speed	Article	http://www.useit.com/alertbox/9703a.html	Texte de référence sur l'utilisabilité des interfaces web.
Keynote Systems Inc, E-COMMERCE RESPONSE TIME: A REFERENCE MODEL	Article	http://www.avoka.com/resources/keynote/E-Commerce_Response_Time_CMG_2000_Chris_.pdf	Un modèle pour la décomposition et le calcul du temps de réponse sur les sites de commerce électronique.
Stacy Joines, Ruth Willenborg, Ken Hygh, Performance Analysis for Java™ Web Sites	Livre	Addison Wesley, September 10, 2002, ISBN: 0-201-84454-0, pages 464	Livre de référence sur l'analyse des performances des applications web écrites en Java.

Suite

Titre	Type	Référence	Objet
Diagnosing J2EE performance problems throughout the application life cycle	Article	MERCURY INTERACTIVE, www.mercuryinteractive.com	Décrit l'apport des logiciels de MERCURY INTERACTIVE dans l'identification des impasses et des goulots d'étranglement.
Jack Shirazi, Java Performance Tuning	Livre	1st Edition September 2000 , ISBN: 0-596-00015-4	Livre de référence sur l'optimisation du code Java toutes applications confondues. Il explique comment évaluer les performances d'un programme et présente une série d'outils et de techniques pour les améliorer.
Jason Hunter , William Crawford, Java Servlet Programming	Livre	2nd edition, O'Reilly, ISBN: 0-596-00040-5	Ouvrage complet sur la programmation des servlets.
Gail Anderson, Paul Anderson, Enterprise JavaBeans Component Architecture: Designing and Coding Enterprise Applications	Livre	Prentice Hall PTR, 0-13-035571-2	Ouvrage dédié au design et à la programmation des Enterprise Java Beans.
Ali Syed and Jamiel Sheikh, Java Doctor	Livre	Manning Publications , Spring 2005,	Livre très récent sur les bugs fréquemment rencontrés dans les applications Java
Benjamin G. Sullins and Mark B. Whipple , JMX in Action	Livre	Manning, ISBN 1930110561, October 2002	Livre complet sur les Java Management Extensions.
Daniel Menasce, Rudolf Riedi, Rodrigo Fonseca, Wagner Meira Jr., Flavia Peligrinelli, Analyzing Robot Behavior in E-Business Sites.	Article	http://portal.acm.org/citation.cfm?id=384268.378838	L'article propose une méthodologie pour analyser les comportements des robots sur les sites d'affaires électroniques.
D. A. Menascé, V. Almeida, R. Fonseca, and M. A. Mendes, "A Methodology for Workload Characterization of E-commerce Sites,"	Article	http://portal.acm.org/citation.cfm?id=337024	Article qui traite de la caractérisation de la charge de travail sur les sites de commerce électronique.
Mehdi Khouja, Farouk Kamoun, Experimenting With the TPC-W E-commerce Benchmark	Article	http://clei2004.spc.org.pe/es/html/pdfs/149.pdf	Article qui propose une implémentation Java du benchmark TPC-W
Transaction Processing Performance Council (TPC), TPC-W BENCHMARK	Spécification	Version 1.7, Oct 11, 2001 http://www.tpc.org/tpcw/spec/tpcw_V1.8.pdf	Texte intégral de la spécification TPC-W
Daniel A. Menascé, Virgilio A. F. Almeida, Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning	Livre	Prentice Hall, 2000, ISBN 0-13-086328-9,	Un livre qui fait autorité dans le domaine de l'évaluation des performances et de la scalabilité des applications d'affaires électroniques
Daniel A. Menascé, Vasudeva Akula, Towards Workload Characterization of Auction Sites	Article	In Proc. Sixth IEEE Workshop on Workload Characterization (WWC-6), Austin, TX, Oct. 27, 2003.	Cet article propose une méthodologie pour caractériser la charge de travail des sites C2C
Daniel A. Menascé, Vasudeva Akula, Improving the Performance of Online Auction Sites through Closing Time Rescheduling, 27 - 30, 2004	Article	http://csdl.computer.org/comp/proceedings/qest/2004/2185/00/21850186abs.htm	Cet article propose des techniques pour optimiser les performances des sites C2C

Suite

Titre	Type	Référence	Objet
Pierre Alain Muller, modélisation objet avec UML , Rational Software Europe, avril 1997	Livre	Eyrolles, 15/03/2000, ISBN : 2-212-09122-2	Livre complet sur la notation UML.
Craig Larman, An Introduction To Object-Oriented Analysis And Design And The Rup	Livre	3 edition (October 20, 2004), Prentice Hall PTR, ISBN: 0131489062	Livre complet sur la notation UML et le Rational Unified Process.
OMG, UML Profile for Schedulability, Performance, and Time Specification	Spécification	http://www.omg.org/docs/formal/03-09-01.pdf	Texte intégral du profil RTP
Bruce Powel Douglass, Real Time UML: Advances in The UML for Real-Time Systems	Livre	Third Edition, Addison Wesley, February 20, 2004, ISBN: 0-321-16076-2, 752 pages	Livre consacré à la modélisation des systèmes temps réel avec la notation UML
Doug Rosenberg , Kendall Scott, Applying Use Case Driven Object Modeling with UML: An Annotated e-Commerce Example	Livre	Addison Wesley, First Edition June 14, 2001, ISBN: 0-201-73039-1	Livre dédié à la modélisation des applications de commerce électroniques avec la notation UML
Early Evaluation of Software Performance based on the UML Performance Profile, Gordon Ping Gu and Dorina C. Petriu, Department of Systems and Computer Engineering, Carleton University, Ottawa	Article	http://portal.acm.org/citation.cfm?id=961335	Cet article donne des exemples sur l'utilisation du profil RTP pour la modélisation d'un site de commerce électronique
Govind Seshadri, Understanding JavaServer Pages Model 2 architecture	Article	http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html	Article sur l'architecture Model 2
William Crawford, Jonathan Kaplan, J2EE Design Patterns	Livre	O'Reilly, September 2003, 0-596-00427-3	Livre de référence sur les patterns J2EE
Sun Microsystems, Core J2EE Pattern Catalog, composite view	Catalogue	http://java.sun.com/blueprints/corej2eepatterns/Patterns/CompositeView.html	Présente le pattern composite view ainsi que ses implications sur les performances
Budi Kurniawan, Java for the Web with Servlets, JSP, and EJB, A Developer's Guide to J2EE Solutions	Livre	New Riders Publishing, April 12, 2002, 0-7357-1195-X	Livre détaillé sur les fondamentaux de la plateforme J2EE
Harvey W. Gunther, WebSphere Application Server, Development Best Practices for Performance and Scalability	Rapport	http://www.ibm.com/software/webservers/appserv/ws_bestpractices.pdf	Astuces pour l'optimisation des performances du WebSphere Application Server
Duane Wessels, Web Caching	Livre	O'Reilly, First Edition June 2001, ISBN: 1-56592-536-X	Livre consacré aux stratégies de cache sur le web
Michael Rabinovich and Oliver Spatscheck, Web Caching and Replication	Livre	Addison Wesley, December 21, 2001, 0-201-61570-3	Idem
Speed Up Your Site: Web Site Optimization	Livre	New Riders Publishing, January 17, 2003, ISBN 0-7357-1324-3	Ouvrage qui couvre plusieurs techniques pour l'optimisation des sites web

Suite

Titre	Type	Référence	Objet
Ed Roman, Scott Ambler, Tyler Jewell, Ed Roman, Tyler Jewell, Floyd Marinescu, Mastering Enterprise JavaBeans	Livre	2nd Edition,Wiley, ISBN: 0471417114, pages 672	Livre de référence sur les EJB.
introduction à RMI	Article	http://www.commentcamarche.net/rmi/rmiintro.php3	Brève introduction sur RMI
Core J2EE Patterns - Session Façade	Catalogue	http://java.sun.com/blueprints/corej2eepatterns/Patterns/SessionFacade.html	Présente le pattern session façade ainsi que ses implications sur les performances
Java™ Performance and Scalability Volume 1: Server-Side Programming Techniques	Livre	Dov Bulka, Addison Wesley, June 05, 2000, ISBN : 0-201-70429-3, pages 320	Présente en 48 leçons les techniques permettant d'améliorer la performance du code Java exécuté du côté du serveur. Ces leçons couvrent : La gestion de la mémoire. Le cashing. La programmation multitâche (multithreading).
Steve Wilson, Jeff Kesselman, Java(TM) Platform Performance:Strategies and Tactics	Livre	Addison-Wesley Professional; 1st edition (May 31, 2000), ISBN: 0201709694, pages 230	Excellent ouvrage sur l'optimisation des performances des applications écrites en Java
Jayson Falkner, Kevin Jones, Servlets and JavaServer Pages™: The J2EE™ Technology Web Tier	Livre	Addison Wesley, September 19, 2003, ISBN: 0-321-13649-7, pages 784	Livre de référence sur la programmation des servlets et des JSP
Jayson Falkner, Another Java Servlet Filter Most Web Applications Should Have, Client-Side Cache Control	Article	http://www.onjava.com/pub/a/onjava/2004/03/03/filters.html	Cet article explique comment modifier l'entête d'un paquet http pour ordonner au navigateur de mettre les données en cache
Ravi Kalidindi and Rohini Datla, Best Practices to improve performance in EJB	Article	http://www.precisejava.com/javaperf/j2ee/EJB.htm	Ensemble de conseils pour l'optimisation des EJB
Nitin Nanda, Learn how to deploy, use, and benchmark JDBC driver types 1, 2, 3, and 4	Article	http://www.javaworld.com/javaworld/jw-07-2000/jw-0707-jdbc.html	Article sur la performance des drivers JDBC
Eweek, Database benchmarking	Article	http://www.eweek.com	Comparatif des performances des principaux gestionnaires de base de données
Gestion d'un pool de connexions SGBD par Tomcat	Tutoriel	http://christophej.developpez.com/tutoriel/j2ee/pooltomcat/	Tutoriel qui explique comment gérer les connection pools sous Tomcat
Jeremy Zawodny, Derek J. Balling, High Performance MySQL,	Livre	O'Reilly, April 2004, ISBN: 0-596-00306-4	Livre de référence sur l'optimisation de MySQL

8. Annexes

Annexe 1: Liste exhaustive des profils UML

UML profiles
A Profile for Integrating Function Blocks into the Unified Modeling Language
A UML Profile for Agent-Oriented Modeling
A UML Profile for Aspect Oriented Modeling
A UML Profile for External AOR Models
A UML Profile for Modeling Workflow and Business Processes
A UML Profile for Real-Time Constraints with the OCL
A UML profile to support requirements engineering with KAOS
Building a UML Profile for MultiTEL
From UML to BPEL
Mapping Object to Data Models with the UML
Modelling QoS: Towards a UML Profile
Raven MDA-Profile
RCR: A UML Profile for Reverse Engineering, Program Comprehension, and Reengineering
Representing XML Schema in UML -- An UML Profile for XML Schema
Requirements for UML Profiles
Requirements for UML Profiles Presentations to the ADTF
The UML Profile for Framework Architectures
Towards a UML Profile for Model-Based Assessment
Towards a UML Profile for Model-Based Risk Assessment
Towards a UML Profile for Service-oriented Architectures
Towards a UML Profile for Software Architecture Descriptions
UML Data Modeling Profile
UML Profile and Interchange Models for EAI
UML profile for archetypes and archetype patterns
UML Profile for CORBA
UML Profile for CORBA Components
UML Profile for EJB
UML Profile for Enterprise Distributed Object Computing
UML Profile for Framework Architectures
UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms
UML Profile for MOF
UML Profile for Schedulability, Performance, and Time Specification
UML Profile for SPEM (from *Software Process Engineering Metamodel Specification*)
UML Profile for the Deployment and Configuration Language
UML Profile for Web Modeling (from *Building Web Applications with UML* by JIM CONALLEN)
UML Testing Profile
UML Testing Profile RFP
UML Textual Notation RFP

Annexe 2: Stéréotypes et étiquettes du sous-profil performance[238]

Stéréotypes du sous-profil performance

Stereotype	Applies to (metaclasses)[239]	Tags	Description
«PAclosedLoad»	Action, ActionExecution, ActionState, SubactivityState, Stimulus, Message, Method, Operation, Reception	PArespTime, PApriority, PApopulation, PAextDelay	A closed workload
«PAcontext»	Collaboration, Collaboration InstanceSet, ActivityGraph		A performance analysis context
«PAhost»	Classifier, Node, ClassifierRole, Instance, Partition	PAutilization, PAschdPolicy, PArate, PActxtSwT, PAprioRange, PApreemptible, PAthroughput	An execution engine that hosts the scenario
«PAopenLoad»	Action, ActionExecution, ActionState, SubactivityState, Stimulus, Message, Method, Operation, Reception	PArespTime [0..*,] PApriority, PAoccurrence	An open workload
«PAresource»	Classifier, Node, ClassifierRole, Instance, Partition	PAutilization, PAschdPolicy, PAschdParam, PAcapacity, PAaxTime, PArespTime, PAwaitTime, PAthroughput	A passive resource
«PAstep»	Messasge, ActionState, Stimulus, SubactivityState	PAdemand, PArespTime, PAprob, PArep, PAdelay, PAextOp, PAinterval	A step in a scenario

Etiquettes associées au stéréotype «PAclosedLoad»

Tag	Type	Multiplicity	Description
PArespTime	PAperfValue	[0..*]	Time required for completion of the scenario from the start of the scenario
PApriority	Integer	[0..1]	Priority of workload
PApopulation	Integer	[0..1]	The size of the workload (i.e., the number of system users)
PAextDelay	PAperfValue	[0..1]	The delay between the completion of one response and the start of the next for each member of the population of system users

Etiquettes associées au stéréotype «PAhost»

Tag	Type	Multiplicity	Description
PAutilization	Real	[0..*]	The mean number of concurrent users
PSschdPolicy	Scheduling Enumeration	[0..1]	Policy by which host schedules workloads
PArate	Real	[0..1]	Processing rate of the scenario execution
PActxtSwT	PAperfValue	[0..1]	Context switch time
PAprioRange	Integer range	[0..1]	Valid range for priorities of workloads
PApreemptable	Boolean	[0..1]	Whether scheduled workloads may be preempted
PAthroughput	Real	[0..1]	Number of workloads completed per unit time

[238] Source Bruce Powel Douglass, Real Time UML: Advances in The UML for Real-Time Systems, Third Edition, Addison Wesley, February 20, 2004, ISBN: 0-321-16076-2, 752 pages

[239] Ces classes sont définies dans le sous-profil des ressources. Pour plus de détails, consulter la version officielle du profil sur le site de l'Object Management Group

Etiquettes associées au stéréotype «PAopenLoad»

Tag	Type	Multiplicity	Description
PArespTime	PAperfValue	[0..*]	Time required for completion of the scenario from the start of the scenario
PApriority	Integer	[0..1]	The priority of workload
PAoccurrence	RTarrivalPattern	[0..1]	The arrival pattern of the workload

Etiquettes associées au stéréotype «PAresource»

Tag	Type	Multiplicity	Description
PAutilization	Real	[0..*]	The mean number of concurrent users
PAschdPolicy	Scheduling Enumeration	[0..1]	Policy by which host schedules workloads
PAcapacity	Integer	[0..1]	Number of workloads that can be handled simultaneously, i.e. the maximum number of concurrent users of the system
PAaxTime	PAperfValue	[0..1]	Resource access time, that is, the time required to acquire or release the resource
PArespTime	PAperfValue	[0..1]	Time required for the resource to complete its response
PAwaitTime	PAperfValue	[0..1]	The time between when a resource access is requested and when it is granted
PAthroughput	Real	[0..1]	Number of workloads that can be completed per unit time

Etiquettes associées au stéréotype «PAstep»

Tag	Type	Multiplicity	Description
PAdemand	PAperfValue	[0..*]	The total demand of the step on its processing resource
PArespTime	PAperfValue	[0..*]	The total time to execute the step, including the time to access and release any resources
PAprob	Real	[0..1]	In stochastic models, the probability that this step will be executed (as opposed to alternative steps)
PArep	Integer	[0..1]	The number of times the step is repeated
PAdelay	PAperfValue	[0..*]	Delay between step execution
PAextOp	PAextOpValue	[0..*]	The set of operations of resources used in the execution of the step
PAinterval	PAperfValue	[0..*]	The time interval between successive executions of a step when the step is repeated within a scenario

Annexe 3: Tables de correspondance entre les types de données de MySQL vs Java vs JDBC[240]

Character Column Types

MYSQL TYPE	JDBC TYPE	JAVA TYPE
CHAR	CHAR	String
VARCHAR	VARCHAR	String
TINYTEXT	LONGVARCHAR	String
TEXT	LONGVARCHAR	String
MEDIUMTEXT	LONGVARCHAR	String
LONGTEXT	LONGVARCHAR	String
TINYBLOB	LONGVARBINARY	byte[]
BLOB	LONGVARBINARY	byte[]
MEDIUMBLOB	LONGVARBINARY	byte[]
LONGBLOB	LONGVARBINARY	byte[]
SET	VARCHAR	String
ENUM	VARCHAR	String

Date and Time Column Types

MYSQL TYPE	JDBC TYPE	JAVA TYPE
DATE	DATE	java.sql.Date
TIME	TIME	java.sql.Time
DATETIME	TIMESTAMP	java.sql.Timestamp
YEAR	DATE	java.sql.Date
TIMESTAMP	TIMESTAMP	java.sql.Timestamp

[240] Source: Mark Matthew, Jim Cole, Joseph D. Gradecki, MySQL and Java Developer's Guide, Wiley Publishing,ISBN 0-471-26923-9

Numeric Column Types (continues)

MYSQL TYPE	JDBC TYPE	JAVA TYPE
TINYINT	TINYINT	byte
SMALLINT	SMALLINT	short
MEDIUMINT	INTEGER	int

Numeric Column Types (continued)

MYSQL TYPE	JDBC TYPE	JAVA TYPE
INT	INTEGER	int
BIGINT	BIGINT	long
FLOAT	REAL	float
DOUBLE	DOUBLE	double
DECIMAL	DECIMAL	java.math.BigDecimal

Liste des figures

Liste des tableaux