International Journal of Engineering Research and Development
e-ISSN: 2278-067X, p-ISSN: 2278-800X, www.ijerd.com
Volume 9, Issue 3 (December 2013), PP. 23-27

Etude du comportement des objets java dans le Tas

Laila Moussaid[1], Mostafa Hanoune[2]
[1,2]*Laboratoire TIM, Faculté des sciences Casablanca.*

Résumé:- L'allocation et la libération explicite des données dans la mémoire provoquent une quantité importante d'erreurs dans les langages de programmation .Or, avec L'apparition du java comme un langage Orienté objet qui possède la technique de la gestion de la mémoire d'une façon automatique simplifie les programmes et élimine une préoccupation du développeur qui ne se soucie ni de l'allocation d'objet ni de sa libération, contribuant de ce fait à s'abstraire facilement du matériel .pour ceci un grand nombre de technique ont été développées pour gérer la mémoire à objet automatiquement. Après une présentation de l'état de l'art dans ce domaine, on va pencher sur notre approche proposée pour gérer les objets java dans le Tas .on clôtura ce manuscrit par une simulation.

Mots-clés:- Système embarqué, ramasse-miettes, gestion de la mémoire, J2ME, Tas.

I. INTRODUCTION

L'utilisation de mémoire dynamique offre un confort de programmation considérable, mais comme nous l'avons vu, la libération manuelle des zones inutilisées est souvent trop difficile.

L'alternative consiste à confier entièrement la tâche de la gestion mémoire à l'environnement d'exécution. Le programmeur doit toujours réserver, explicitement ou non, la mémoire qu'il veut utiliser, mais il n'a plus à se préoccuper de la libérer, elle est récupérée automatiquement. Le ramasse miettes RM désigne l'ensemble des techniques permettant d'automatiser cette détection des zones mémoires inutilisées en vue de leur réutilisation [1,2]. Le concept (ainsi que le nom) de ramasse-miettes provient des premières implantations du langage Lisp, dans les années 60 [3].

L'objectif d'un ramasse miettes est de déterminer, parmi l'ensemble des objets présents en mémoire, lesquels sont encore vivants et lesquels sont devenus inaccessibles, de façon à recycler l'espace occupé par les objets morts. Ainsi, l'environnement d'exécution se réapproprie automatiquement la mémoire lorsque le programme ne l'utilise plus. La programmation avec un langage de bas niveau a beaucoup d'inconvénients ; il réduit La modularité, la portabilité et la réutilisation de l'application .Une approche modulaire basée sur les composants, pour les systèmes embarquées est donnée dans la référence [4]. Cette approche a été étendue dans la référence [5] pour les systèmes temps réels répartis. Une nouvelle approche pour les systèmes embarqués s'appuyant sur

le langage de programmation Orientée objet Java est donnée dans la référence [6].cet article se focalise sur la gestion automatique des objets java dans le tas.

II. LA GESTION DE LA MEMOIRE DYNAMIQUE

Pour permettre l'utilisation de Java dans le monde des systèmes embarqués et temps réel, le problème de la gestion mémoire reste rédhibitoire. On peut distinguer dans ce problème deux questions distinctes, même si elles ne sont pas complètement indépendantes: Celle de l'impact temporel du gestionnaire mémoire sur le programme et celle de la quantité de mémoire utilisée par le programme. La première est plutôt liée aux notions de système temps-réel embarqué, et la seconde à celles de système embarqué.

Les différentes approches présentées dans ce domaine attaquent chacune ce problème sous un angle différent, cependant aucune d'entre elles ne permet de résoudre de façon parfaitement le problème. Les travaux qui reposent sur l'analyse statique et la gestion en régions [7], [8] en générale et dans java en particulier [9], comme ceux basés sur la Spécification Temps-Réel pour Java (RTSJ), sont très délicats à mettre en œuvre surtout au niveau de développement. Il serait préférable pour le développeur dans ce cas de continuer à programmer selon ses habitudes. Encore ces approches basées sur la synthèse de régions souffrent de fuites de mémoire intempestives, qui sont détectées seulement lors de l'exécution du programme.

D'autre part, une autre source d'imprévisibilité de java est le ramasse miettes. Les approches qui focalisent sur l'utilisation de la technique de Ramasse miettes proposent plusieurs scénario et algorithme .on peut citer les algorithmes du Ramasse miettes les plus utilisée dans le système embarqué : le marquage balayage et le comptage de références.Le premier nécessite au moins deux parcours de l'ensemble des objets ce qui rend le temps de l'allocation plus important et augmente la durée de pauses pendant l'exécution d'un programme, encore la technique marquage balayage n'empêche pas la fragmentation de la mémoire. Tandis que le ramasse miettes qui utilise l'algorithme comptage de référence présente une limitation importante pour les structures

cycliques, la chose qui impose alors d'associer ce type de RM avec un autre mécanisme, destiné à libérer la mémoire. En outre, cette méthode de récupération de la mémoire n'empêche pas la fragmentation du tas, autrement dit une allocation de mémoire peut échouer même si la quantité de mémoire libre est suffisante mais il est trop fragmentée.

III. CONTRIBUTION

Notre approche proposée dans les environnements des systèmes embarqués destinée à la gestion dynamique des objets java dans le tas [10].

En effet, Nous proposons le découpage de la mémoire en petit morceau nommé "page" de quelques kilo-octets. Nous affectons un compteur à Chaque page .Alors l'algorithme repose sur l'idée « les pages beaucoup utilisées dans un passé proche ont beaucoup de chance d'être à nouveau utilisées dans le futur proche ».

Autrement dit, une page peu utilisée dans un passé proche a peu de chance d'être utilisée dans un futur proche et par conséquent nous allons appliquer le RM par marquage-balayage avec la compactage dans les pages beaucoup utilisées dans un passé proche, et le RM par marquage-balayage sans défragmentation dans les pages moins utilisées dans le passé proche.

Le compteur de Chaque page est initialisé à 0 au début, à chaque utilisation de la page le bit de poids fort du compteur R prend la valeur 1 et dans la deuxième utilisation le compteur est décalé de 1 bit vers la droite.

Pour voir le comportement des objets java dans le cas d'implémentation notre approche proposée dans un environnement embarquée. Nous avons choisi la machine virtuelle java destinée au système embarqué pour les microéditions KVM comme un environnement de test.

IV. EXPERIENCES

Avant de décrire le comportement des objets java dans notre étude, nous avons décrire brièvement chacune des applications échantillons, ces applications ont l'avantage d'utiliser les concepts de la programmation : récursivité, utilisation de la mémoire dynamique, structures cycliques …etc., c'est pourquoi ils représentent bien le genre de programmes Java pour les microéditions :

Calculator : C'est une application de calculatrice qui prend en charge plus, la multiplication, la soustraction et des opérations de division. Les opérandes de ceux-ci sont entrés en utilisant un clavier virtuel.

Factorielle : factorielle d'un entier naturel n, notée $n!$, est le produit des nombres entiers strictement positifs inférieurs ou égaux à n. elle joue un rôle important en algèbre combinatoire parce qu'il y a n! Façons différentes de permuter n objets. Elle apparaît dans de nombreuses formules en mathématiques.

Fibonacci : elle est crée par le mathématicien italien Leonardo Fibonacci, c'est une application récursive, elle est utilisée dans tous les domaines soit la pédagogie, le cinéma, la géologie…

Ackermann : c'est un exemple simple d'application récursive, trouvée en 1926 par Wilhelm Ackermann. Elle est présentée sous la forme d'une fonction avec deux paramètres entiers naturels comme arguments et qui retourne un entier naturel comme valeur, noté en général A(n, m).

Padovan : C'est une application définie par une suite récurrente linéaire qui ressemble dans sa forme à la suite de Fibonacci, à une nuance près : la somme des termes de rang n et n+1 ne donne pas le terme de rang n + 2 mais celui de rang n + 3,La suite porte le nom de l'architecte Richard Padovan .Le terme général de la suite de Padovan est lié au trois racines du polynôme de degré trois $x^3 - x - 1$. Ce polynôme possède une racine réelle r1 et deux racines complexes conjuguées r2 et r3 , elle est toujours strictement croissante à partir du rang trois .

McCarthy : une fonction mathématique récursive qui prend un entier comme argument et retourne un entier. Elle est définie par l'informaticien John McCarthy comme un test pour la vérification formelle. Cette fonction McCarthy est définie comme M(n)=n-10 si n est supérieur strictement à 100 et elle est égale à M(M(n+11)) si n est inférieur à 100.

Chess : Ce jeu d'échecs est une application de jeu dans lequel l'utilisateur joue contre l'ordinateur. Le joueur peut déplacer les pièces en sélectionnant et en les faisant glisser à l'aide d'un dispositif de type stylo. Une séquence de 24 se déplace prises par l'utilisateur avant de gagner un match a été utilisé dans le profil.

Nous testons les applications choisies dans le simulateur java Wireless Toolkit 2.5.2 destiné aux applications développées en j2me. Le simulateur intègre un profiler du processeur et de la mémoire qui nous aide à réaliser le profilage des applications échantillons et par la suite nous observerons l'optimisation de l'occupation mémoire dans le cas d'implémentation notre approche proposée. La comparaison entre le profilage des applications échantillons dans le cas de kvm et le profilage des applications échantillons dans le cas d'implantation de nouvelle approche nous permettra de bien saisir le comportement des objets java dans la mémoire Tas dans les deux cas.

Nous allons exécuter chaque application échantillon dans le simulateur et bien préciser les deux comportements des objets java dans le tas dans le kvm et le cas de l'implantation de l'approche proposée qu'on aille désigner par kv2m pour la distinction.

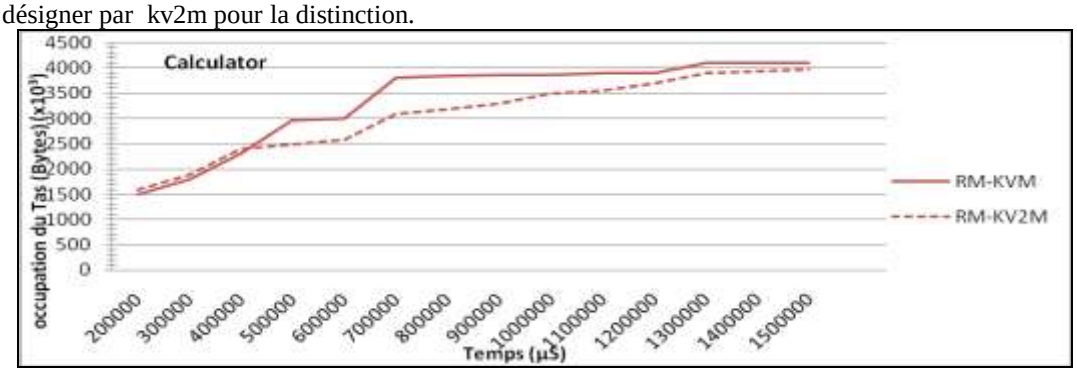

Figure7 : Profilage de la mémoire pour l'application Calculator

Dans la figure7, le comportement de L'application calculator se change au cours de temps dans les deux tests de kvm et kv2m .on remarque que la courbe dans le cas de kv2m est au dessous de celle de kvm si on néglige la petite phase de 200000µs-400000µs.Alors on conclut que la taille de l'occupation de la mémoire Tas se baisse avec l'implémentation de notre proposition kv2m.

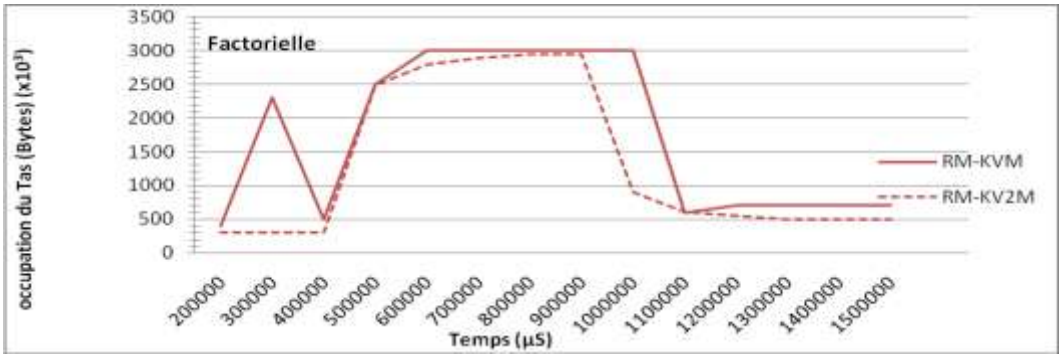

Figure8 : Profilage de la mémoire pour l'application Factorielle

Si on néglige le comportement de l'intervalle 400000µs-500000µs on observe que la courbe de l'application Factorielle, qui joue un rôle important en algèbre combinatoire, dans le cas kvm est toujours au dessus de celle de kv2m. Cette observation peut être traduire par la diminution de l'occupation de la mémoire Tas dans l'implémentation de kv2m .

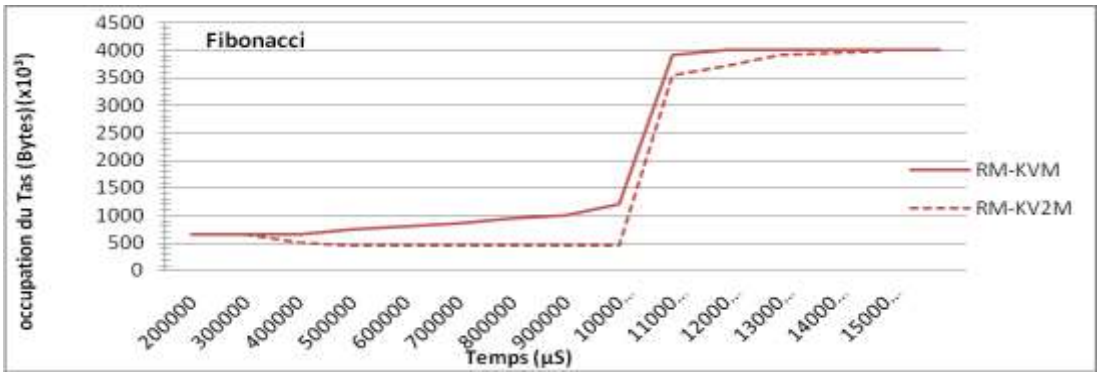

Figure9 : Profilage de la mémoire pour l'application Fibonacci

Au cours de l'exécution de l'application échantillon Fibonacci (figure9), dans l'intervalle temporelle 300000µs-1500000µs, la courbe de Fibonacci dans le cas kv2m est toujours au dessous de la courbe de kvm. Donc la taille de l'occupation de la mémoire Tas par l'application dans le cas ou on implémente notre nouveau RM dans kvm c'est-à-dire lorsqu'on exécute sous kv2m est inférieure à la taille de l'occupation de la mémoire Tas dans le cas de kvm.

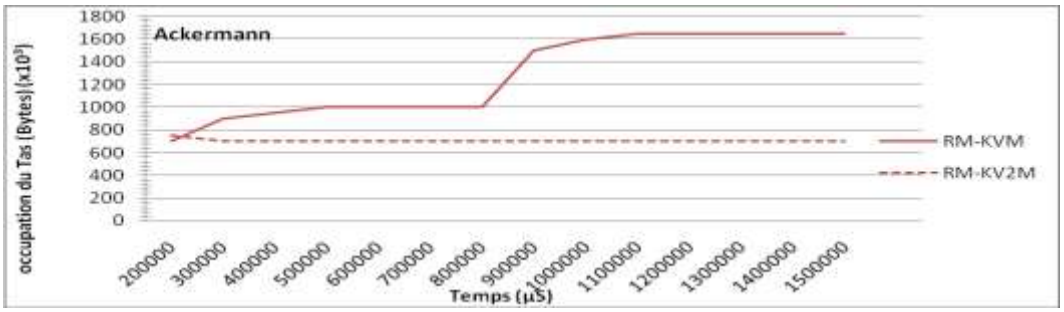

Figure10 : Profilage de la mémoire pour l'application Ackermann

Dans la figure10, le comportement de L'application Ackermann se change au cours de temps dans les deux tests de kvm et kv2m d'une façon plus claire et plus visible.

On remarque que la courbe dans le cas de kv2m est presque stable dans la valeur 760000Bytes, par contre la courbe dans le cas de kvm est s'augmente de 760000Bytes jusqu'à la valeur 1640000Bytes.

On peut dire que dans le cas de KVM l'application est trop consommable dans l'utilisation de la mémoire Tas. Alors on conclut que la taille de l'occupation de la mémoire Tas se baisse avec l'implémentation de notre proposition kv2m.

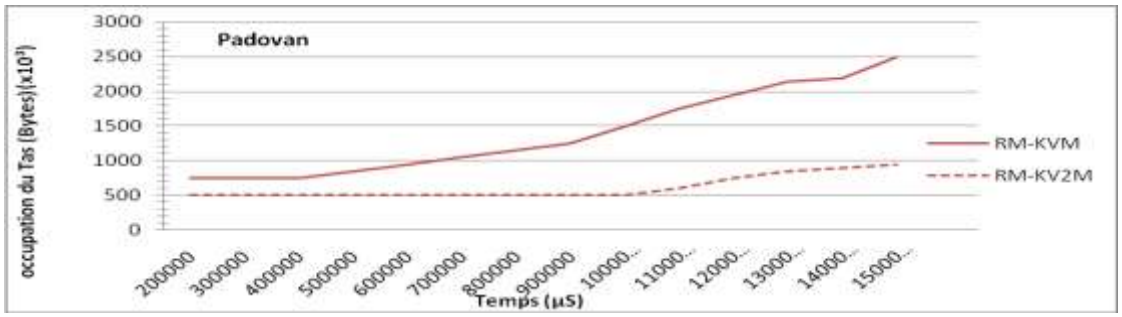

Figure11 : Profilage de la mémoire pour l'application Padovan

Au cours de l'exécution de l'application échantillon Padovan (figure11), les deux courbes de l'application Padovan dans les deux cas kvm et kv2m est toujours s'éloignent entre eux d'une façon multiple dans l'intervalle temporelle 200000µs-1500000µs.

Donc la taille de l'occupation de la mémoire Tas par l'application dans le cas de kvm multiple la taille de l'occupation de la mémoire Tas dans le cas ou on implémente notre nouveau RM dans kvm, c'est-à-dire lorsqu'on exécute sous kv2m, de point temporelle à l'autre. Dans cette figure 11, on observe toujours que l'application Padovan , dans le cas de kvm, est trop consommable par rapport au cas de kv2m .

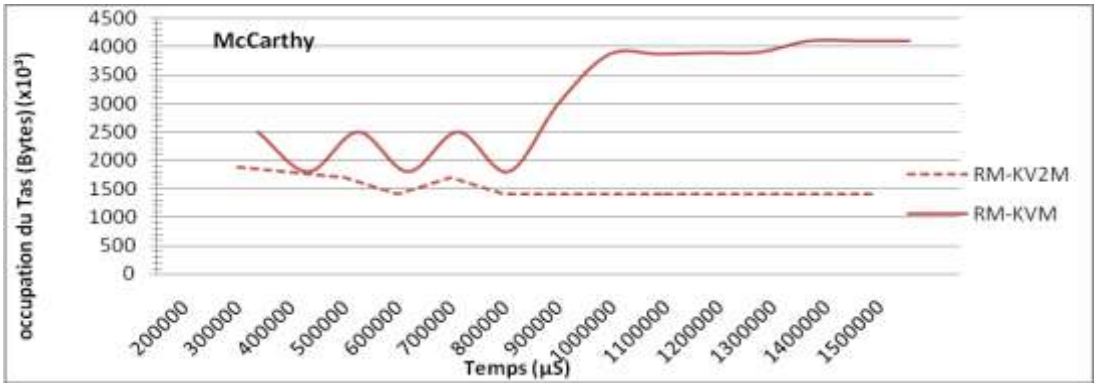

Figure12 : Profilage de la mémoire pour l'application McCarthy

La figure 12 illustre le comportement de l'application échantillon McCarthy .Au cours de l'exécution de cette application, sa courbe dans le cas kvm est au dessus de la courbe dans le cas de kv2m .ainsi, à partir du point temporelle 800000 µs, la taille des objets dans la mémoire Tas dans le cas de kv2m est presque fixe à 1500000bytes tandis que la taille de l'occupation de la mémoire Tas par l'application dans le cas de kvm varie

entre 2000000bytes et 4500000bytes . On peut conclure donc que la consommation du Tas est assez importante dans le cas de kvm que dans le cas de kv2m.

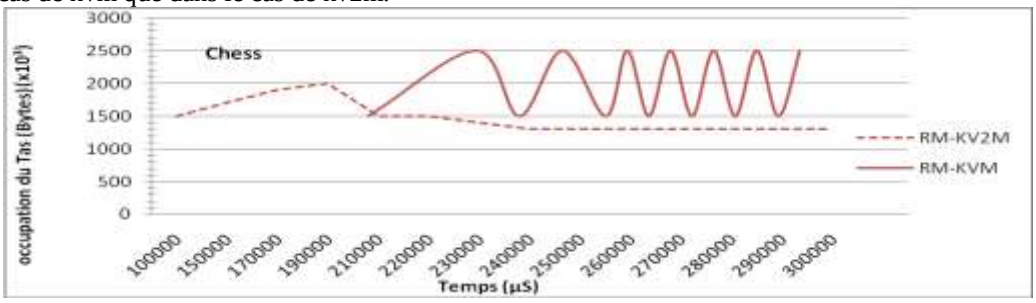

Figure13 : Profilage de la mémoire pour l'application Chess

Au cours de l'exécution de l'application échantillon Chess (figure13), sa courbe, dans le cas kvm, prend l'allure sinusoïdal entre les deux valeurs 1500000 µs et 2500000 µs, alors que dans le cas de kv2m la courbe varie entre 1500000 bytes et 2000000 bytes dans l'intervalle temporelle 100000 µs - 210000 µs. Puis à partir de 210000 µs la consommation de la mémoire Tas se baisse dans le cas de kv2m jusqu'à la valeur de 1300000bytes au court de l'intervalle temporelle 240000 µs-300000 µs.

Donc la taille de l'occupation de la mémoire Tas par l'application dans le cas de kvm est assez grande que la taille de l'occupation de la mémoire Tas dans le cas ou on implémente notre nouveau RM dans kvm, c'est-à-dire lorsqu'on exécute sous kv2m

V. CONCLUSION ET PERSPECTIVE

le test de sept applications échantillons dans le simulateur java wireless toolkit nous permet d'observer le comportement des objets java dans le Tas dans les deux cas de kvm et l'implémentation de notre approche proposée dans la machine virtuelle java destinée aux microéditions kv2m .effectivement, nous avons remarqués que l'occupation de la mémoire Tas dans le cas de kvm pour presque les sept applications échantillons est assez importante par rapport à l'occupation du Tas dans le cas d'implémentation de notre approche kv2m .Les sept applications sont beaucoup consommables de la mémoire dans le cas de kvm que dans le cas kv2m.

Notre travail ne s'arrête pas dans cette simulation mais nous continuerons à travailler sur l'implémentation de notre approche proposée dans le système embarquée d'un robot contrôlé à distance via un téléphone portable qui aura l'objective du prochain article.

REFERENCES

[1]. Richard E. Jones. Garbage Collection: Algorithms for Automatic Dynamic Memory Management. Wiley, 1996. [p. 35, 39,40,48]

[2]. Paul R. Wilson. Uniprocessor garbage collection techniques. In Proceedings of the 1992 International Workshop on Memory Management (IWMM'92), pages 1–42.Springer, 1992. [p. 39, 42]

[3]. John McCarthy. Recursive functions of symbolic expressions and their computation by machine. Communications of the ACM, 3(4):184–195, 1960. [p. 39]

[4]. S.REN and G.A.Agha.AModular Approch for programming Distributed Real-Time Systems.Journal Of Parallel and Distributed Computing,36(1),January 1996.

[5]. S.REN and G.A.Agha.AModular Approch for programming Embedded systems.In Proc.of Lectures on Embedded Systems,LNCS Vol.1494.springer verlag.http://www-osl.cs.uiuc.edu,1996.

[6]. J.SYoung,J.MacDonald,M.Shilman,A.Tabbara,P.Hilfinger,and A.R.Newton . Desing and Specification of Embedded Systems In Java Using Successive Formal Refinement.Technical Report,Departement Of Electrical Engendering and Computer Sciences University Of California,Berkeley,1998.

[7]. Mads Tofte and Jean-Pierre Talpin. Implementation of the typed call-by-value_-calculus using a stack of regions. In Proceedings of the 21st ACM Symposium on Principles of Programming Languages (POPL'94), pages 188–201. ACM Press, 1994. [p. 49, 70]

[8]. Mads Tofte and Jean-Pierre Talpin. Region-based memory management. Information and Computation, 132(2):109–176, 1997. [p. 49, 70]

[9]. Sigmund Cherem and Radu Rugina. Region analysis and transformation for Java programs. In Proceedings of the 4th International Symposium on Memory Management (ISMM'04), pages 85–96. ACM Press, 2004. [p. 10, 50, 54, 70, 76, 79, 88, 109,117, 123, 136]

[10]. Laila Moussaid, Mostafa Hanoune (2011, Mai) "Gestion du TAS: Application sur J2MEInternational Journal of Mathematical Archive, Page: 716-719.