**(12) DEMANDE INTERNATIONALE PUBLIÉE EN VERTU DU TRAITÉ DE COOPÉRATION EN MATIÈRE DE BREVETS (PCT)**

**(19) Organisation Mondiale de la Propriété Intellectuelle**
Bureau international

**(43) Date de la publication internationale**
**5 janvier 2012 (05.01.2012)**

**PCT**

**(10) Numéro de publication internationale**
**WO 2012/000949 Al**

**(51) Classification internationale des brevets :**
*G06F 9/45* (2006.01)

**(21) Numéro de la demande internationale :**
PCT/EP201 1/060748

**(22) Date de dépôt international :**
27 juin 201 1 (27.06.201 1)

**(25) Langue de dépôt** : français

**(26) Langue de publication** : français

**(30) Données relatives à la priorité :**
1055261 29 juin 2010 (29.06.2010) FR

**(71) Déposant** *(pour tous les États désignés sauf US) :* **FLEXYCORE** [FR/FR]; 14E rue du Patis Tatelin, F-35700 Rennes (FR).

**(72) Inventeurs; et**
**(75) Inventeurs/Déposants** *(pour US seulement) :* **CABILLIC, Gilbert** [FR/FR]; 10 rue de Normandie, F-35530 Brece (FR). **LESOT, Jean-Philippe** [FR/FR]; Les Goupillères, F-35370 Argentre Du Plessis (FR).

**(74) Mandataire : LE SAUX, Gaël;** Technopole Atalante, 16B, rue de Jouanet, Bretagne, F-35703 Rennes Cedex 7 (FR).

**(81) États désignés** *(sauf indication contraire, pour tout titre de protection nationale disponible)* : AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PE, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

**(84) États désignés** *(sauf indication contraire, pour tout titre de protection régionale disponible)* : ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), eurasien (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), européen (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, гг, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

**Publiée :**

— *avec rapport de recherche internationale (Art. 21(3))*

**(54) Title** : SELECTIVE COMPILING METHOD, DEVICE, AND CORRESPONDING COMPUTER PROGRAM PRODUCT

**(54) Titre** : PROCÉDÉ DE COMPILATION SÉLECTIVE, DISPOSITIF ET PRODUIT PROGRAMME D'ORDINATEUR CORRESPONDANT

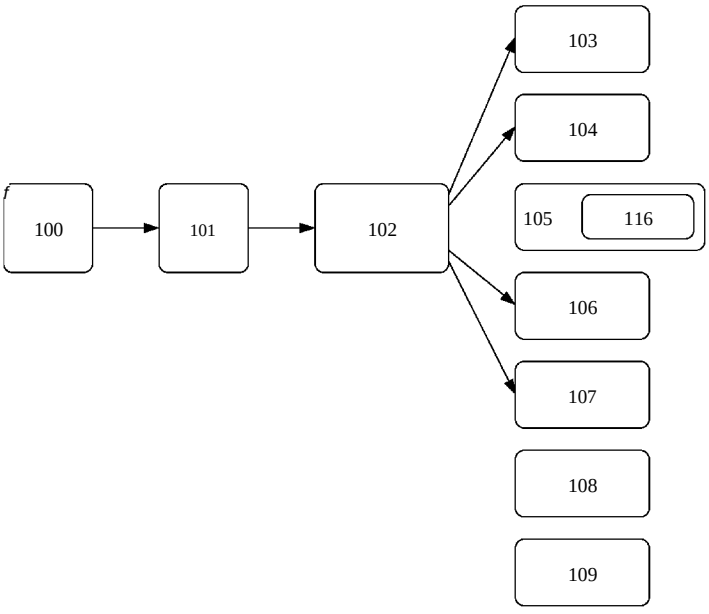


Figure 3

**(57) Abstract** : The invention relates to a method for compiling a software application to be executed on a virtual machine of a physical platfbrm, said software application originally being in the fbrm of a set of codes to be compiled. According to the invention, such a method includes a phase of selectively compiling said set of codes to be compiled, outputting a compiled application including a first application part which is executed by said virtual machine using commands from said virtual machine, and a second application part including binary commands which can be executed directly on said physical platform.

**(57) Abrégé** : L'invention concerne un procédé de compilation d'une application logicielle destinée à s'exécuter au sein d'une machine virtuelle d'une plateforme matérielle, ladite application logicielle se présentant originellement sous la forme d'un ensemble de codes à compiler. Selon l'invention, un tel procédé comprend une phase de compilation sélective dudit ensemble de codes à compiler délivrant une application compilée comprenant une première partie applicative exécutée par ladite machine virtuelle en utilisant des instructions de ladite machine virtuelle et une deuxième partie applicative comprenant des instructions binaires directement exécutable sur ladite plateforme matérielle.

WO 2012/000949 A1

**Procédé de compilation sélective, dispositif et produit programme d'ordinateur correspondant.**

## 1 DOMAINE DE L'INVENTION

La présente invention se rapporte au domaine de la compilation de programmes ou d'applications logicielles.

La présente invention se rapporte plus particulièrement à la compilation de programmes ou d'applications logicielles dans des environnements hétérogènes.

Un programme informatique, ou une application logicielle se compose en règle générale d'un ensemble d'instructions binaires ou intermédiaires dites exécutables. Ces instructions exécutables sont usuellement issues d'une compilation d'un programme écrit dans un langage source. Ainsi, pour pouvoir créer une application logicielle, un développeur rédige un programme source écrit dans un langage de programmation spécifique. Ce programme source est ensuite compilé afin de former un programme exécutable. Au cours de cette compilation, les instructions écrites en langage source sont transformées en programme binaire qui est lui-même exécuté sur une plateforme matérielle telle qu'un ordinateur, un terminal de communication.

Pour compiler un programme écrit dans un langage source pour une certaine cible (processeur, plateforme, …), il y a quatre méthodes générales :

- disposer d'un compilateur complet, c'est-à-dire comprenant une partie qui se charge de reconnaître le langage source, et une autre qui se charge de créer le code pour la cible choisie ;
- disposer d'un compilateur du langage source vers un code intermédiaire, et fournir une machine virtuelle de ce code intermédiaire sur la cible choisie. C'est le cas pour le langage Java™ par exemple ;
- fournir, à un compilateur qui existe déjà pour la cible choisie, un module complémentaire pour reconnaître le langage source ;
- disposer d'un compilateur qui transforme un programme écrit dans un langage source en un programme écrit dans un langage intermédiaire, et mettre en œuvre un compilateur existant du langage intermédiaire pour la

cible choisie.

## 2 SOLUTIONS DE L'ART ANTERIEUR

Considérant le cas où l'application est compilée du langage source vers un code intermédiaire, et que ce code intermédiaire est exécuté par une machine virtuelle sur la cible choisie, comme c'est le cas général pour le langage Java™, il se pose le problème de la performance d'exécution de l'application. La Figure 1 décrit le principe général d'une technique d'exécution de code à base de machine virtuelle. Le code source 100 est compilé vers un code intermédiaire 102 grâce au compilateur 101. Puis ce code est exécuté sur une machine virtuelle 103, elle-même s'exécutant sur le système d'exploitation 104, lui-même s'exécutant sur une plateforme matérielle 105. Dans la mesure où le code intermédiaire doit pouvoir s'exécuter sur n'importe quelle plateforme disposant d'une machine virtuelle adaptée au code intermédiaire, celui-ci n'est pas optimisé.

Dans un contexte d'optimisation de la performance du code intermédiaire, on distingue deux techniques d'optimisation générales dans l'état de l'art.

La première technique consiste à réaliser une compilation du code intermédiaire de façon dynamique. On la désigne par compilation dynamique. Pour se faire la machine virtuelle comprend un compilateur capable de compiler dynamiquement des portions de code intermédiaires en binaire, et comprend également un mécanisme pour exécuter dynamiquement le code binaire généré par ce compilateur à la place du code intermédiaire original. Ainsi l'exécution de l'application est constituée d'un mixte de code intermédiaire et de code binaire généré dynamiquement.

On distingue deux grands types de réalisations de la compilation dynamique : le « just in time » (JIT) et le « dynamic adaptive compiler » (DAC).

Le Just In Time a généralement une granularité de compilation à la méthode (il ne peut compiler que des méthodes entières). Il réalise la compilation en code binaire de la méthode à exécuter lors de la première exécution de la méthode. Le dynamic adaptive compiler (DAC) met en oeuvre une autre stratégie. II exécute l'application par la machine virtuelle et détermine dynamiquement les

parties de code pénalisant la performance de l'application. Ces parties sont appelées les points chauds de l'application. Chaque DAC a sa propre stratégie de détection de point chaud. Par exemple, les méthodes les plus souvent exécutées, ou les méthodes comprenant des boucles, ou des méthodes prenant le plus de temps à s'exécuter. Une fois cette identification faite, il compile les points chauds en code binaire. Ensuite, la machine virtuelle exécute le code binaire des points chauds.

On présente, en relation avec la figure 2, le principe général de la compilation et de l'exécution d'une application grâce à une machine virtuelle, en combinant un compilateur dynamique afin d'accélérer l'exécution de l'application.

Une application logicielle source (100) est rédigée par un développeur d'application. Ce programme source comprend des instructions de programme informatique écrites dans un langage de programmation source. Le programme source (100) subit une première phase de compilation par un compilateur (101) qui aboutie à sa transformation en un programme intermédiaire (102). Ce programme intermédiaire (102) est exécuté par une machine virtuelle (103) qui s'exécute au-dessus d'un système d'exploitation (104) en binaire, qui repose sur une plateforme matérielle (105). Cette machine virtuelle (103) est couplée à un compilateur dynamique (106) qui détermine automatiquement et compile de façon dynamique des portions de code intermédiaire en code binaire au sein d'un emplacement de mémoire temporaire (buffer) de code binaire (107). La machine virtuelle (103) comprend également des moyens d'exécution d'une version binaire d'un code intermédiaire, si il existe, afin d'accroître la performance de l'application.

L'utilisation d'un compilateur dynamique pose cependant deux problèmes. Tout d'abord un compilateur dynamique prend du temps pour s'exécuter en plus de l'application à la fois pour déterminer les points chauds, et lors de la compilation du code intermédiaire en code binaire. En effet, pour réaliser l'identification des points chauds, une instrumentation de l'interpréteur de la

machine virtuelle est généralement réalisée afin de surveiller le comportement de l'application. Ces instrumentations pénalisent de ce fait la performance de l'application. Concernant le compilateur, afin de limiter leur temps de compilation, les optimisations réalisées par ces compilateurs sont simples et ne peuvent pas être agressives. Ceci va limiter la performance du code binaire généré par rapport à un compilateur qui s'exécute en amont de l'exécution. Ensuite, ces solutions utilisent de la mémoire temporaire pour stocker les codes binaires générés. Dans la majorité des cas, ce code est stocké dans une zone de mémoire limitée, et le compilateur doit donc faire de façon continue des choix pour déterminer quels sont les bons codes binaires à garder dans ce buffer. Il en découle un problème de performance de ces compilateurs dynamiques dès lors qu'ils gèrent de nombreuses applications, ou que la taille de cette zone mémoire est petite.

Ainsi les compilateurs dynamiques, bien que générant du code binaire ne permettent pas d'atteindre des niveaux de performances importants. Par ailleurs, plus le code intermédiaire est important (comme c'est le cas sur une plateforme Android™ où les applications, services systèmes et les interfaces de programmation sont exécutées en code intermédiaire) plus la détection de point chaud est difficile à réaliser, et plus la gestion de buffer de code est sollicitée, ce qui limite d'autant la performance du système.

La deuxième technique d'optimisation de la performance du code intermédiaire consiste à rendre la machine virtuelle plus rapide sur la cible choisie. Cette optimisation peut se faire en optimisant la machine virtuelle pour qu'elle exploite le plus possible les capacités du processeur qui l'utilise. Elle peut également se faire au travers de support matériel spécifique additionnel. Elle peut également être réalisée par la conception d'algorithme spécifique permettant d'accroître la performance de la machine virtuelle sur la cible.

En dehors de ces deux solutions, il demeure la possibilité d'utiliser un compilateur hors ligne qui compile la totalité du code source ou le code intermédiaire en code binaire en amont de l'exécution de l'application. Cette

approche permet de ne pas pénaliser l'exécution de l'application par le temps pris par le compilateur dynamique, et permet également de réaliser des compilations très optimisées de l'application. Cette solution a l'avantage de permettre d'obtenir des performances importantes pour les applications. Cette solution impose cependant de compiler complètement une application et utilise le système d'exploitation pour contrôler l'exécution de l'application. Dans les environnements où la machine virtuelle est le cœur d'exécution du système, comme c'est le cas pour le système Android™, cela la rend inutilisable car cette solution n'est pas prévue pour s'intégrer à une machine virtuelle. Par ailleurs, la taille du code binaire généré étant plus important que la taille du code interprété, la taille de l'application devient très importante et implique l'utilisation de plus de mémoire.

## 3 RESUME DE L'INVENTION

L'invention ne comprend pas ces inconvénients de l'art antérieur. En effet, l'invention concerne un procédé de compilation d'une application logicielle destinée à s'exécuter au sein d'une machine virtuelle d'une plateforme matérielle, ladite application logicielle se présentant originellement sous la forme d'un ensemble de codes à compiler.

Selon l'invention un tel procédé comprend une phase de compilation sélective dudit ensemble de codes à compiler délivrant une application compilée comprenant une première partie applicative exécutée par ladite machine virtuelle en utilisant des instructions de ladite machine virtuelle et une deuxième partie applicative comprenant des instructions binaires directement exécutable sur ladite plateforme matérielle.

Ainsi l'invention permet d'optimiser la performance des applications en réalisant, à partir d'un même ensemble de codes une application constituée d'une partie applicative exécutée par ladite machine virtuelle et une partie applicative exécutée directement par la plateforme matérielle, sans utilisation de l'interpréteur de la machine virtuelle. L'ensemble de codes à compiler est constitué de fichiers qui sont écrits dans un unique langage.

Selon un mode de réalisation particulier, ledit ensemble de codes à compiler est un ensemble de codes intermédiaires et en ce que ledit procédé comprend une première phase de compilation d'un ensemble de codes sources et délivrant ledit ensemble de codes intermédiaires utilisés lors de ladite phase de compilation sélective.

Ainsi, l'invention permet d'optimiser directement un code intermédiaire, qui a déjà fait l'objet d'une première compilation, pour optimiser l'exécution de ce code intermédiaire.

Selon un mode de réalisation particulier, ledit ensemble de codes à compiler est un ensemble de codes sources écrits dans un langage de programmation.

Selon une caractéristique particulière ladite phase de compilation sélective comprend :

- une première étape de sélection, parmi ledit ensemble de code à compiler, d'un premier sous ensemble de code à compiler ;
- une étape de compilation dudit premier sous ensemble de code à compiler délivrant ladite première partie applicative.

Selon une caractéristique particulière, ladite phase de compilation sélective comprend en outre :

- une deuxième étape de sélection, parmi ledit ensemble de code à compiler, d'un deuxième sous ensemble de code à compiler complémentaire dudit premier sous ensemble préalablement sélectionné ;
- une étape de compilation dudit deuxième sous ensemble de code à compiler délivrant un troisième sous ensemble de code source à compiler ;
- une étape de compilation dudit troisième sous ensemble de code source à compiler délivrant ladite deuxième partie applicative.

Selon une caractéristique particulière, ledit procédé comprend en outre lors de ladite étape de compilation dudit deuxième sous ensemble de code source à compiler, une étape d'introduction d'au moins une structure d'interfaçage de ladite deuxième partie applicative avec ladite machine virtuelle.

Cette structure de données peut par exemple une interface de type JNI (« Java Native Interface »). Selon une autre approche, cette structure d'interfaçage peut être une structure particulière permettant de réaliser un lien avec une interface interne, plus performante, de la machine virtuelle.

Selon une caractéristique particulière, ledit ensemble de codes à compiler est un ensemble de « ByteCode » Java.

Selon une caractéristique particulière, ledit ensemble de codes à compiler est un ensemble de « ByteCode » Dalvik.

Selon une caractéristique particulière, ledit ensemble de codes à compiler est un ensemble de codes Java.

Selon un mode de réalisation particulier, ledit procédé est mis en œuvre préalablement à une exécution de ladite application logicielle au sein de ladite machine virtuelle.

Selon un autre aspect, l'invention concerne également un dispositif de compilation d'une application logicielle destinée à s'exécuter au sein d'une machine virtuelle d'une plateforme matérielle, ladite application logicielle se présentant originellement sous la forme d'un ensemble de codes à compiler. Selon l'invention un tel dispositif comprend des moyens de compilation sélectifs dudit ensemble de codes à compiler délivrant une application compilée comprenant une première partie applicative exécutée par ladite machine virtuelle en utilisant des instructions de ladite machine virtuelle et une deuxième partie applicative comprenant des instructions binaires directement exécutable sur ladite plateforme matérielle.

Selon un autre aspect, l'invention concerne également un programme d'ordinateur comprenant des instructions de code de programme pour la mise en œuvre du procédé de navigation tel que décrit précédemment, lorsque ce programme est exécuté par un processeur.

L'invention apporte une solution nouvelle et inventive à ces problèmes de l'art antérieur. En effet l'invention propose un procédé de compilation automatique et sélectif d'un code écrit dans un langage de programmation source

ou intermédiaire afin de générer un code binaire des parties sélectionnées pouvant être exécuté par une machine virtuelle.

Cette machine virtuelle pouvant de façon complémentaire exécuter un compilateur dynamique qui se chargera de compiler le code intermédiaire restant à s'exécuter.

Ainsi, l'invention repose sur une approche tout à fait nouvelle et inventive de la compilation d'applications logicielles pour les environnements fonctionnant à base de machine virtuelle. En étant partiel et sélectif, l'invention permet au développeur de choisir précisément ce qu'il veut compiler afin de garder le contrôle de l'utilisation de la compilation hors ligne. Elle permet également de combiner les approches de compilation hors ligne et l'approche de compilation dynamique, dans l'objectif de fournir une meilleure performance d'exécution

## 4 LISTE DES FIGURES

D'autres caractéristiques et avantages de l'invention apparaîtront plus clairement à la lecture de la description suivante d'un mode de réalisation préférentiel, donné à titre de simple exemple illustratif et non limitatif, et des dessins annexés, parmi lesquels :

- la figure 1, déjà commentée, présente le principe général de la compilation d'un code vers un code intermédiaire afin de l'exécuter dans une machine virtuelle ;
- La figure 2, déjà commentée, présente le principe général de la compilation dynamique d'une application logicielle qui s'exécute dans une machine virtuelle ;
- la figure 3 décrit le principe de l'invention ;
- la figure 4 est un schéma de principe d'un dispositif de compilation selon l'invention.

## 5 DESCRIPTION DETAILLEE DE L'INVENTION

### 5.1 Rappel du principe de l'invention

Le principe général de l'invention réside dans la combinaison d'une approche statique de la compilation, permettant de créer des applications

informatiques optimisées en réalisant une compilation (préalablement à l'exécution de l'application) tout en conservant une exécution basée sur l'implémentation de machines virtuelles. Comme cela a déjà été mentionné, une des problématiques des applications qui fonctionnent à l'aide de machines virtuelles est la performance. En effet, le principe à la base de l'utilisation de machines virtuelles est de permettre une portabilité du code intermédiaire (issu d'une ou plusieurs compilations de code source) sur toute plateforme matérielle disposant d'une machine virtuelle capable d'exécuter le code intermédiaire en question.

Or cette universalité du code intermédiaire implique de réaliser une interprétation du code intermédiaire à l'aide d'instructions comprises par le processeur, rendant peut performant l'exécution de ce code. Aussi, cela est généralement reconnu, le code binaire offre des performances nettement supérieures à un code intermédiaire. Cette performance provenant principalement du fait que le code binaire s'exécute directement par le processeur de la plateforme matérielle en utilisant l'ensemble des instructions du processeur et sans nécessité d'interprétation complémentaire.

L'invention se rapporte ainsi à une technique de compilation pour les environnements d'exécution à base de machine virtuelle qui fournit une meilleure performance d'exécution.

Le principe d'amélioration des performances repose sur le déplacement de l'exécution de certaines parties de code intermédiaire sous la forme de code binaire nettement plus performant, tout en restant sous le contrôle d'exécution de l'application par la machine virtuelle cible.

La compilation se base sur le code source (ou intermédiaire) de l'application. Lors de la compilation, le compilateur réalise une génération de code comprenant deux parties : un premier ensemble de code intermédiaire qui s'exécute en mettant en œuvre la machine virtuelle et un deuxième ensemble de code binaire qui s'exécute directement par le processeur de la plateforme matérielle.

La Figure 3 décrit le principe de l'invention.

L'invention permet au développeur de sélectionner (101) depuis un code source ou un code intermédiaire 100 les parties de codes à compiler en code binaire. Le compilateur 102 génère les codes (103, 104, 106 et 107) permettant d'exécuter les portions de codes choisies par le développeur en code binaire. Les codes générés (103, 104, 106 et 107) vont s'exécuter par la machine virtuelle 105, grâce à son support d'exécution de code binaire 116. La machine virtuelle s'exécute au-dessus du système d'exploitation 108, lui-même s'exécutant sur une plate-forme matérielle 109.

Ainsi, l'invention permet au développeur de choisir les morceaux de codes sources devant s'exécuter avec une grande performance, sans opération d'intégration manuelle additionnelle.

La sélection du code source (et/ou intermédiaire) à compiler en binaire :

- Est faite en fonction des choix du développeur. Il n'y a donc plus de détection de points chauds par la machine virtuelle pour ce code source sélectionné
- Cette sélection permet de travailler le compromis expansion mémoire généré et donc d'optimiser la taille mémoire nécessaire. Ceci est particulièrement intéressant en fonction des types de machines virtuelles utilisés, comme cela sera décrit par la suite.
- Permet d'accélérer des codes d'application, ou de services systèmes,
- Permet d'accélérer des libraires de la machine virtuelle.

Au niveau de l'intégration basée sur le support de l'exécution de code binaire de la machine virtuelle :

- L'intégration à la machine virtuelle est transparente. Il n'y a pas de modification de la machine virtuelle.
- La performance du code final dépend du support d'exécution de code binaire de la machine virtuelle.

- L'intégration est compatible avec une solution de compilation dynamique d'une machine virtuelle. Cette intégration permet de cumuler les performances.
- Le code binaire permet d'être chargé dynamiquement, ou embarqué selon possibilités de la machine virtuelle.
- Ce type d'intégration offre la possibilité de procéder par compilation successive pour arriver au résultat (optimiseur de code intermédiaire, compilateur binaire dédié).

L'invention permet également de disposer de différentes granularités de compilation : méthode, application, service, APIs, ou bout de code. Le fait de disposer de différentes granularités de compilation permet de cibler précisément les parties de code à accélérer selon les cas.

La méthode de compilation de l'invention est mise en œuvre en amont de l'exécution de l'application sur la plateforme. N'ayant pas de contraintes de ressources, le compilateur peut ainsi utiliser des optimisations agressives très poussées pour générer le code binaire ou le code intermédiaire qui permettront d'améliorer encore plus les performances de l'application.

Il est également possible d'utiliser un schéma de compilations successives afin de générer les codes intermédiaires et binaires amenées à être exécutés. Ainsi le compilateur générera des codes sources qui seront compilés par des compilateurs tiers pour générer ensuite les codes intermédiaires et les codes binaires. De ce fait, les codes générés bénéficieront du cumul des optimisations réalisées par ces compilateurs.

Selon un mode de réalisation particulier, afin de permettre son exécution par la machine virtuelle cible, un ensemble de codes spécifiques liant les deux ensembles (intermédiaire et binaire) est intégré aux deux ensembles de codes générés en utilisant une interface native de la machine virtuelle. L'utilisation d'une interface native impacte positivement le degré d'amélioration de performance obtenu (en fonction de l'interface utilisée).

Selon un mode de réalisation particulier, afin de maximiser la performance de l'application le compilateur utilise l'une des interfaces natives disponibles par la machine virtuelle. Ainsi, cette interface native pourra être soit l'interface native standard de la machine virtuelle permettant de faire le lien avec des librairies natives (par exemple une interface JNI), soit une des interfaces internes plus performantes. Ainsi, il n'est pas nécessaire de prévoir un ensemble de codes spécifiques pour lier l'ensemble de codes intermédiaires et l'ensemble de codes binaires.

Selon une caractéristique particulière, la dépendance entre la partie applicative binaire et la machine virtuelle réside dans l'interface native de cette dernière. Parfois cette interface native n'a pas besoin d'être modifiée, parfois, elle doit être modifiée afin d'obtenir l'amélioration de performance désiré.

Dans ce mode de réalisation, comme le procédé de compilation s'appuie sur l'une des interfaces natives de la machine virtuelle, il permet de conserver le système objet de la machine virtuelle cible, sans avoir à le modifier.

Ainsi, par exemple, il est prévu, lors de la compilation permettant d'obtenir l'ensemble de codes binaires, d'introduire au sein de celui-ci un ensemble d'instructions permettant l'accès au système objet de la machine virtuelle. Cet ensemble d'instructions, appelé ensemble d'instructions de référencement, permet, lors de l'exécution de l'application, de créer une structure d'interfaçage de la partie applicative binaire (qui correspond à l'ensemble de codes binaires) avec la machine virtuelle. Plus particulièrement, cette structure d'interfaçage est dans un mode de réalisation particulier une ou plusieurs variables globales de la partie applicative binaire. Une variable globale comprend un pointeur qui pointe vers une adresse (en mémoire) d'accès à des objets ou des structures de données gérés par la machine virtuelle pour l'application. Ainsi, dans ce mode de réalisation il n'y a qu'un seul ensemble d'objets ou structures de données, commun à la fois à l'ensemble de codes intermédiaires et l'ensemble de codes binaires.

Un autre exemple consiste lors de la compilation permettant d'obtenir l'ensemble de codes binaires, d'introduire au sein de celui-ci un autre ensemble d'instructions de référencement permettant, lors de l'exécution de l'application, de créer une structure d'interfaçage vers un tableau de référencement des objets utilisés par la partie applicative binaire. Ce tableau de référencement des objets fait partie des données qui sont gérées par la partie applicative. En effet, lors de l'exécution de l'application, les objets étant partagés entre la machine virtuelle et la partie applicative binaire, il est nécessaire que cette dernière connaisse les objets qui sont utilisés par la partie applicative binaire pour éviter que le « garbage collector » de la machine virtuelle ne supprime des objets utilisés par la partie applicative binaire. Il est donc nécessaire que la machine virtuelle puisse aller prendre connaissance des objets utilisés par la partie applicative binaire au sein de l'espace mémoire alloué à la partie applicative. L'ajout, lors de la génération des ensembles de code, des structures de référencement, permet d'autoriser la machine virtuelle à s'assurer qu'elle ne supprime pas des objets utilisés par la partie binaire.

Dans un autre mode de réalisation l'utilisation d'un second système objet permettant d'exécuter le code binaire sans avoir à utiliser l'interface native de la machine virtuelle est également possible afin d'améliorer toujours plus les performances.

A l'extrême, selon un mode de réalisation particulier, la quasi totalité d'une application peut être compilée en code binaire en utilisant un système objet propre. Dans ce mode de réalisation particulier l'ensemble de codes intermédiaires généré est minimal et permet à la machine virtuelle de réaliser le minimum d'opération requise pour exécuter l'application telle que lancer, terminer ou suspendre l'exécution de l'application sur le terminal.

Cependant, quel que soit le mode de réalisation, le moteur de base de l'exécution de l'application reste la machine virtuelle.

La base du procédé consistant à compiler un ensemble de code source ou intermédiaire en deux ensembles de codes (intermédiaire et binaires) est

automatique. L'ensemble du procédé peut être complètement automatique, ou comporter des phases manuelles complémentaires.

Généralement le code binaire occupe une place plus importante en mémoire que du code intermédiaire. En effet, celui-ci est prévu pour tenir compte des contraintes de taille mémoire du terminal. Ainsi lors de l'utilisation de l'invention, lorsqu'un code intermédiaire est compilé en code binaire, il y a un accroissement de la taille du code de l'application. Afin de contrôler à la fois le compromis entre amélioration de performance et taille du code binaire généré, le procédé permet, dans au moins un mode de réalisation, de sélectionner les parties de codes à compiler. Ainsi le développeur de l'application maîtrise l'expansion de code au vu des contraintes mémoire du terminal cible.

Dans un mode de réalisation alternatif, le compilateur peut également décider lui-même de la découpe des codes à effectuer en utilisant une optimisation particulière qui permet de réaliser un compromis expansion mémoire/performance de façon automatique.

Le procédé est compatible avec une machine virtuelle cible interprétée. Cette machine virtuelle peu également disposer d'un compilateur dynamique qui se chargera d'améliorer la performance du code intermédiaire. Ainsi, les améliorations de performances se cumulent (initiale avec le procédé de l'invention et dynamique avec le compilateur dynamique).

Le procédé n'est pas restreint aux applications et peut être utilisé pour du code source ou intermédiaire d'une application, service ou librairie amené à être exécuté par la machine virtuelle.

Selon un mode de réalisation particulier, le procédé peut être utilisé sur tous les ensembles de codes d'une application, ou une portion de code particulier, sans limite de granularité (une classe, une méthode, ou un morceau de code d'une méthode).

Par la suite on décrit une mise en œuvre de l'invention adaptée à la plateforme Androïd™. Il est évident que l'invention n'est nullement limitée à

cette plateforme particulière mais peut également être mise en œuvre dans le cadre d'autres plateformes.

5.2 Description d'un mode de réalisation

On présente dans ce mode de réalisation, la mise en œuvre de l'invention pour un environnement d'exécution Android™.

L'environnement d'exécution Android™ de Google™ est basé sur une distribution du système d'exploitation Linux™ auquel sont ajoutées des librairies, une machine virtuelle qui se nomme Dalvik™ et du code intermédiaire. Ce code intermédiaire comprend des applications, des services système et des librairies embarqués sur la plateforme. Les codes sources des applications, services et librairies sont écrits en JAVA™, puis compilés en code intermédiaire Dalvik™. La particularité d'Android™ est qu'il est adapté au fonctionnement de terminaux légers disposant, au regard des ordinateurs personnels actuels, de faibles capacités d'exécution. La machine virtuelle Dalvik™ a la particularité d'être basée sur une architecture à « registres » (de l'anglais « register-based virtual machines ») à la différence des autres machines virtuelle qui sont dite « à pile » (de l'anglais « stack based virtual machines »).

En général, les machines virtuelles à base de « pile » doivent utiliser les instructions pour charger des données sur la pile et de manipuler les données, et, par conséquent, exigent plus d'instructions que les machines à « registre » pour mettre en œuvre le code de même niveau, mais les instructions dans une machine à « registre » doivent encoder les registres source et destination et, par conséquent, ont tendance à être plus volumineuses.

La chaîne de compilation d'Androïd™ pour le langage java réalise la compilation du code java en code intermédiaire en utilisant deux compilateurs. Le premier compile le code source java en code intermédiaire Java (ByteCode Java). Le second compile le code intermédiaire java (ByteCode Java) en code intermédiaire Dalvik (fichier .dex). On a donc une chaîne de deux compilations successives pour aboutir à un code exécutable *« .dex »* sur une plateforme Androïd™ à partir d'un code source écrit en java.

Dans la plateforme Android™, pour améliorer les performances, la mise en œuvre de l'invention est la suivante :

- le code source java initial CSj, est compilé à l'aide du compilateur java de la plateforme Android™ afin de produire un premier ensemble de code intermédiaire (ByteCode Java) Clj, qui est, dans ce mode de réalisation, le code à compiler.
- puis, le code intermédiaire Clj est compilé de façon sélective pour obtenir deux ensembles de code distincts : un premier ensemble de code source C/C++ CScl et un deuxième ensemble de code intermédiaire Clj2.
- plus précisément, le deuxième ensemble de code intermédiaire CIj2 est compilé en code intermédiaire Dalvik™ à l'aide du compilateur disponible dans la chaîne de compilation Androïd™ afin de produire l'application *« .dex »* exécutable sur la plateforme par l'intermédiaire de la machine virtuelle Dalvik™.
- le premier ensemble de code source CScl, comprenant du code C/C++ est compilé à l'aide du compilateur C/C++ disponible dans la chaîne de compilation Android™ afin d'en générer du code binaire qui se présente sous la forme de librairies dynamiques Linux™, afin de produire la deuxième partie de l'application.

La machine virtuelle Dalvik™ comprend plusieurs interfaces natives, dont une interface JNI. JNI (de l'anglais « Java Native Interface » pour « interface Java Native ») est un framework (« cadre de développement ») qui permet à du Byte Code Java s'exécutant à l'intérieur d'une machine virtuelle java d'appeler (et d'être appelé) par des applications ou librairies dites natives s'exécutant en code binaire.

Cette interface JNI est largement utilisée dans les environnements Java. Aussi, afin de lier l'exécution des deux types de codes générés (d'une part le code Dalvik™ et d'autre part les librairies dynamiques Linux™, le compilateur de l'invention ajoute des codes spécifiques JNI permettant d'exécuter les deux parties.

Ainsi, l'invention permet de compiler des parties Java en code natif, tout en gardant des parties de code intermédiaire, le tout en liant l'exécution des deux parties de l'application à l'aide de l'interface JNI de Dalvik™.

Une autre implémentation de l'invention peut également consister à utiliser l'interface native interne de la machine virtuelle Dalvik™ pour éviter l'utilisation de l'interface JNI, qui peut elle-même poser quelques problèmes de performances.

Bien entendu, comme cela a déjà été exposé, la machine virtuelle Dalvik™ reste maître de l'exécution et son modèle d'exécution objet reste conservé et utilisé pour exécuter l'application. Ce modèle d'exécution objet est en charge de la gestion des champs, des méthodes, de la mémoire associée.

D'un point de vue exécution, l'invention permet donc de bénéficier d'un code binaire très performant qui s'exécute au sein de la machine virtuelle.

Un autre avantage est de pouvoir tirer parti des propriétés de l'architecture mémoire de la plateforme matérielle. Chaque processeur de la plateforme matérielle possède une architecture mémoire associée. Celle-ci peut être composée de plusieurs niveaux de cache mémoire, de mémoire spécifiques (à accès direct, ROM, ...), de mémoire additionnelle vive ou flash, de différentes fréquences de cadencement.

Outre sa performance de base pour exécuter une instruction binaire, la performance du processeur va être fonction de la localisation des instructions et des données d'une application dans sa hiérarchie mémoire. Généralement le processeur permet d'avoir deux canaux d'accès mémoire en parallèle. Un pour les instructions, l'autre pour les données. Ces deux canaux sont par ailleurs optimisés et calibrés en fonction de la distribution des accès instructions et données.

Concernant les machines virtuelles, le code intermédiaire est considéré être une donnée. Aussi, tout les accès mémoires passent par le même canal, ce qui pénalise le fonctionnement de l'architecture mémoire et impacte la performance d'exécution.

Les compilateurs dynamiques génèrent du code binaire pouvant être chargé par le canal d'instruction. Aussi, ils permettent d'utiliser les deux canaux

d'accès mémoire en parallèle. Néanmoins ce code binaire est dans une mémoire vive, empêchant l'utilisation de mémoire morte, résidente à proximité du processeur.

Dans une implémentation particulière de l'invention, puisque le compilateur peut s'exécuter en amont de l'intégration du terminal, le code binaire peut être placé dans une mémoire morte proche du processeur. Ceci permet :

- de profiter de la performance des mémoires mortes afin de rendre encore plus performant l'application ;
- de profiter des performances énergétiques des mémoires mortes permettant d'accroître l'autonomie de la plateforme ;
- de limiter l'utilisation de la mémoire vive en n'y insérant pas de données ou code additionnels.
- De profiter de l'utilisation des différents canaux d'accès mémoire instruction et donnée.

Au final, la performance globale du système est améliorée.

5.3 Autres caractéristiques optionnelles et avantages

Dans au moins un mode de réalisation, l'invention se présente sous la forme d'un dispositif apte à mettre en œuvre le procédé de compilation tel que décrit précédemment. Un tel dispositif est schématiquement décrit en relation avec la figure 4.

Il comprend une mémoire 41, et une unité de traitement 42 qui est équipée d'un microprocesseur piloté par un programme d'ordinateur (ou application). L'unité de traitement 42 reçoit en entrée, via un module d'interface d'entrée I, un ensemble de codes à compiler, que le microprocesseur traite, selon les instructions du programme précité, pour générer une application composée de deux parties. Cette application en deux parties est délivrée par l'intermédiaire de l'interface T.

Pour réaliser cette application en deux parties, le dispositif comprend des moyens de compilation sélectifs de l'ensemble de codes à compiler délivrant une application compilée comprenant une première partie applicative exécutée par ladite machine virtuelle en utilisant des instructions de ladite machine virtuelle et

une deuxième partie applicative comprenant des instructions binaires directement exécutable sur ladite plateforme matérielle.

Les moyens de compilation sélectifs comprennent des moyens de sélection des sous-ensembles de codes à compiler. Ces moyens de sélection sont soit mis en œuvre de manière manuelle soit mis en œuvre automatiquement. Une mise en œuvre automatique de la sélection peut être réalisée par un dispositif de préanalyse de code 43 visant à identifier les portions de code pour lesquelles une optimisation et/ou une exécution binaire est préférable.

Les moyens de compilation sélectifs comprennent en outre au moins deux dispositifs de compilation qui permettent d'une part de générer la première partie de l'application, qui est exécutée par la machine virtuelle et la deuxième partie de l'application, qui est exécutée directement par le système d'exploitation de la plateforme matérielle. Ces deux dispositifs de compilation peuvent être complétés par d'autres dispositifs, comme d'autres compilateurs, des éditeurs de liens ou des optimisateurs.

# REVENDICATIONS

**1.** Procédé de compilation d'une application logicielle destinée à s'exécuter au sein d'une machine virtuelle d'une plateforme matérielle, ladite application logicielle se présentant originellement sous la forme d'un ensemble de codes à compiler caractérisé en ce qu'il comprend une phase de compilation sélective dudit ensemble de codes à compiler délivrant une application compilée comprenant une première partie applicative exécutée par ladite machine virtuelle en utilisant des instructions de ladite machine virtuelle et une deuxième partie applicative comprenant des instructions binaires directement exécutable sur ladite plateforme matérielle.

**2.** Procédé selon la revendication 1, caractérisé en ce que ledit ensemble de codes à compiler est un ensemble de codes intermédiaires et en ce que ledit procédé comprend une première phase de compilation d'un ensemble de codes sources et délivrant ledit ensemble de codes intermédiaires utilisés lors de ladite phase de compilation sélective.

**3.** Procédé selon la revendication 1, caractérisé en ce que ledit ensemble de codes à compiler est un ensemble de codes sources écrits dans un langage de programmation.

**4.** Procédé selon la revendication 1, caractérisé en ce que ladite phase de compilation sélective comprend :

- une première étape de sélection, parmi ledit ensemble de code à compiler, d'un premier sous ensemble de code intermédiaire ;
- une étape de compilation dudit premier sous ensemble de code à compiler délivrant ladite première partie applicative.

**5.** Procédé selon la revendication 4, caractérisé en ce que ladite phase de

compilation sélective comprend en outre :

- une deuxième étape de sélection, parmi ledit ensemble de code à compiler, d'un deuxième sous ensemble de code à compiler complémentaire dudit premier sous ensemble préalablement sélectionné ;
- une étape de compilation dudit deuxième sous ensemble de code à compiler délivrant un troisième sous ensemble de code source à compiler ;
- une étape de compilation dudit troisième sous ensemble de code source à compiler délivrant ladite deuxième partie applicative.

**6.** Procédé selon la revendication 5, caractérisé en ce qu'il comprend en outre, lors de ladite étape de compilation dudit deuxième sous ensemble de code source à compiler, une étape d'introduction d'au moins une structure d'interfaçage de ladite deuxième partie applicative avec ladite machine virtuelle.

**7.** Procédé selon la revendication 1, caractérisé en ce que ledit ensemble de codes à compiler est un ensemble de « ByteCode » Java.

**8.** Procédé selon la revendication 1, caractérisé en ce que ledit ensemble de codes à compiler est un ensemble de « ByteCode » Dalvik.

**9.** Procédé selon la revendication 1, caractérisé en ce que ledit ensemble de codes à compiler est un ensemble de codes Java.

**10.** Procédé selon la revendication 1, caractérisé en ce qu'il est mis en œuvre préalablement à une exécution de ladite application logicielle au sein de ladite machine virtuelle.

**11.** Dispositif de compilation d'une application logicielle destinée à s'exécuter au sein d'une machine virtuelle d'une plateforme matérielle, ladite

application logicielle se présentant originellement sous la forme d'un ensemble de codes à compiler caractérisé en ce qu'il comprend des moyens de compilation sélectifs dudit ensemble de codes à compiler délivrant une application compilée comprenant une première partie applicative exécutée par ladite machine virtuelle en utilisant des instructions de ladite machine virtuelle et une deuxième partie applicative comprenant des instructions binaires directement exécutable sur ladite plateforme matérielle.

**12.** Produit programme d'ordinateur téléchargeable depuis un réseau de communication et/ou stocké sur un support lisible par ordinateur et/ou exécutable par un microprocesseur, caractérisé en ce qu'il comprend des instructions de code de programme pour l'exécution du procédé de compilation selon l'une au moins des revendications 1 à 10, lorsqu'il est exécuté sur un ordinateur.

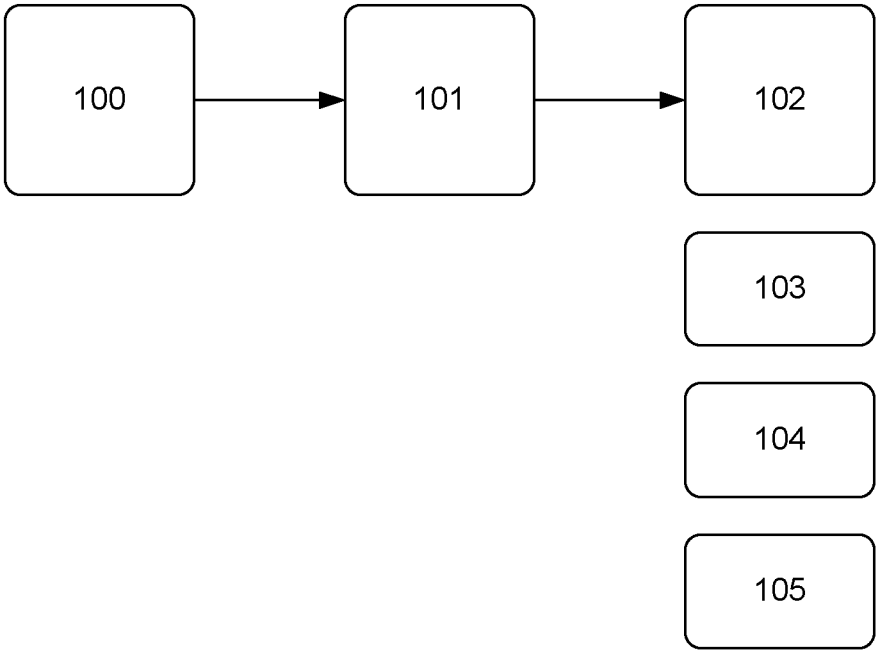


Figure 1

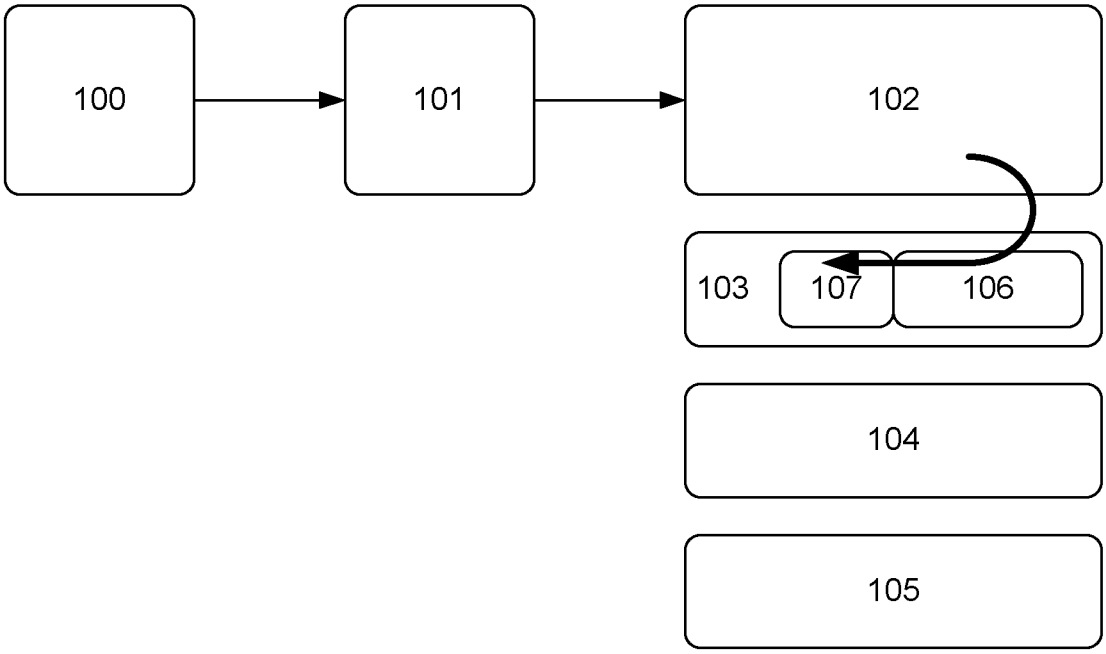


Figure 2

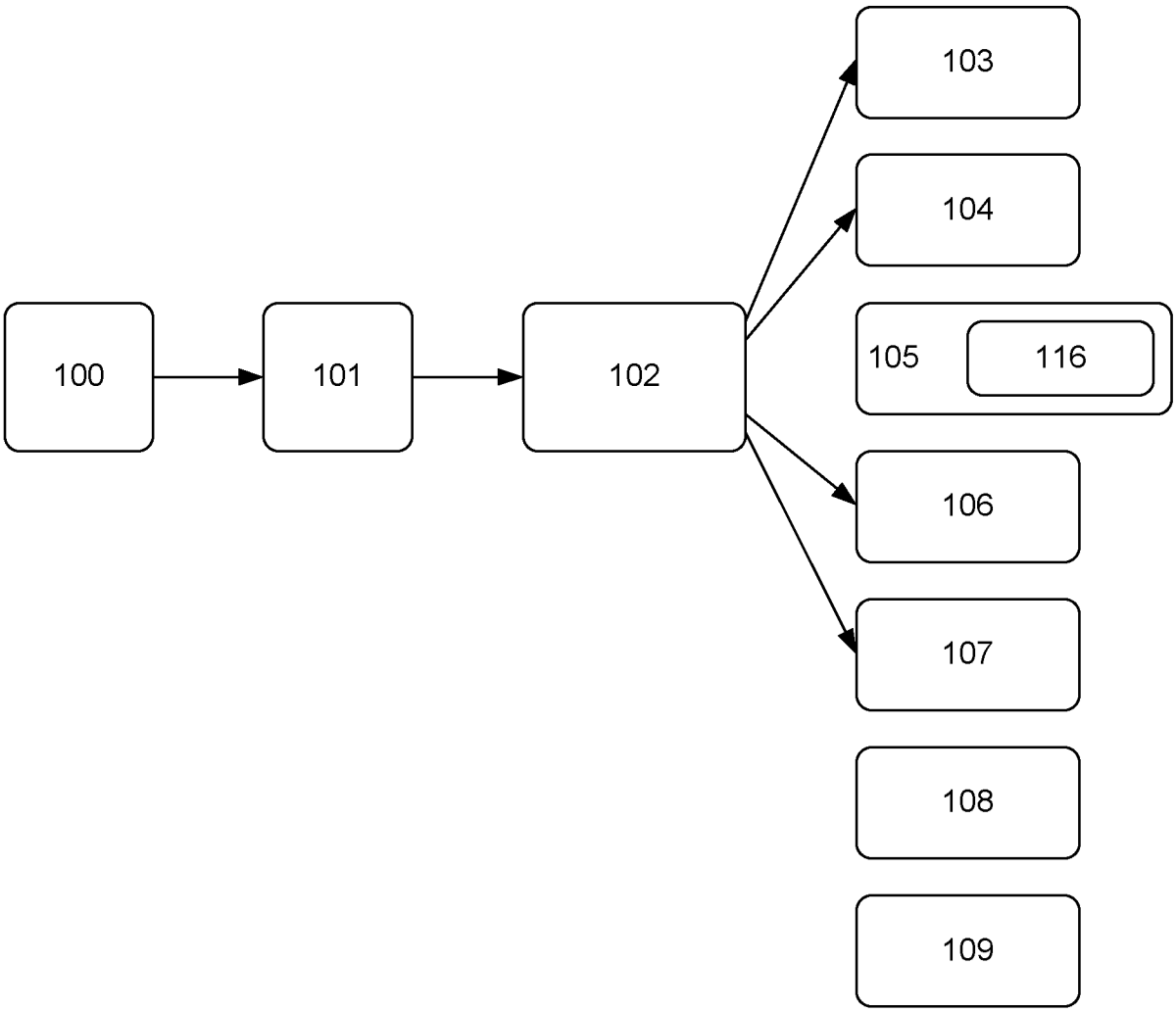


Figure 3

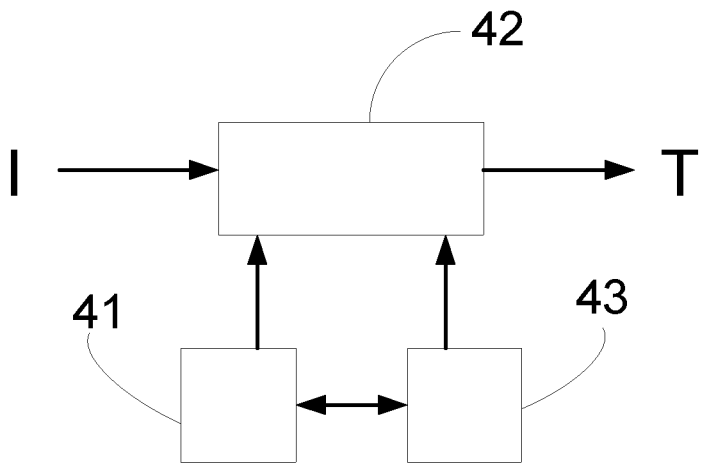


Figure 4

# INTERNATIONAL SEARCH REPORT

International application No
PCT/EP2011/060748

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F9/45
ADD.

Aooording to International Patent Classification (IPC) orto both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification **System** followed **by** classification **symbols)**
G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal , WPI Data, IBM-TDB, INSPEC, COMPENDEX

C. DOCUMENTS CONSIDERED TO BE RELEVANT

| Category* | Citation of document, with indication, where appropriate, of the relevant passages | Relevant to claim No. |
|---|---|---|
| X | US 2004/243989 Al (OWENS HOWARD DEWEY [US] ET AL OWENS HOWARD DEWEY [US] ET AL)<br>2 December 2004 (2004-12-02)<br>abstract; figures 2,4,5 ,6<br>page 1, paragraph 1<br>page 1, paragraph 13 - page 2, paragraph 16<br>page 2, paragraph 18 - page 3, paragraph 28<br>page 3, paragraph 30 - page 4, paragraph 32; claims 1-5 ,7-12 , 14-16, 18-22<br>-----<br>-/-- | 1-12 |

[X] Further documents are listed in the continuation of Box C. [X] See patent family annex.

* Spécial catégories of cited documents :

"A" document defining the gênerai state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" documentwhich maythrow doubts on priority claim(s) or which is cited to establish the publication date of another citation or other spécial reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published priorto the international filing date but laterthan the priority date claimed

"T" later document published after the international filing date or priority date and not in conflictwith the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such docu¬ments, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

| Date of the actual completion of the international search | Date of mailing of the international search report |
|---|---|
| 5 August 2011 | 16/08/2011 |

| Name and mailing address of the ISA/ | Authorized officer |
|---|---|
| European Patent Office, P.B. 5818 Patentlaan 2<br>NL - 2280 HV Rijswijk<br>Tel. (+31-70) 340-2040,<br>Fax: (+31-70) 340-3016 | Lelait, Sylvain |

3

Form PCT/ISA/210 (second sheet) (April 2005)

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

| Category* | Citation of document, with indication, where appropriate, of the relevant passages | Relevant to claim No. |
|---|---|---|
| X | US 2002/046298 Al (BAK LARS [US] ET AL) 18 Apri l 2002 (2002-04-18)<br>abstract; f i gures 4-6<br>page 1, paragraph 10 - page 2, paragraph 13<br>page 3, paragraph 51 - page 5, paragraph 80<br>page 7, paragraph 13 - paragraph 14;<br>cl aims 1,2,8-11 ,13, 19-23 ,29-32 ,36-38,43-45 ,49-51 , 57-59 ,65 | 1-12 |
| X | DE VERDI ERE V C ET AL: "Speedup prédi cti on for sél ecti ve compi l ati on of embedded Java programs" ,<br>EMBEDDED SOFTWARE. SECOND INTERNATIONAL CONFERENCE, EMS0FT 2002 . PR0CEEDINGS (LECTURE NOTES IN COMPUTER SCI ENCE VOL. 2491) SPRINGER-VERLAG BERLIN , GERMANY, 2002 , pages 227-239 , XP002618859 ,<br>ISBN : 3-540-44307-X<br>abstract<br>page 227 , paragraph 2 - page 228, paragraph 4<br>page 227 , l ast paragraph - page 229 , paragraph 5<br>page 231 , paragraph 4<br>page 232 , l ast paragraph - page 233 , paragraph 3<br>page 233 , l ast paragraph - page 234, paragraph 1<br>page 235 , paragraph 2<br>page 237 , paragraph 1 | 1-12 |
| X | US 6 289 506 Bl (KW0NG ALICE [US] ET AL) 11 September 2001 (2001-09-11)<br>abstract; f i gures 4,5 ,7-10<br>col umn 2, l i ne 35 - col umn 2, l i ne 44<br>col umn 3, l i ne 45 - col umn 4, l i ne 37<br>col umn 5, l i ne 40 - col umn 6, l i ne 6<br>col umn 8, l i ne 20 - col umn 10, l i ne 47;<br>cl aims 1,2,5-12 ,14-19 ,21-27 ,29-32 | 1-12 |
| A | W0 03/032155 A2 (SUN MICROSYSTEMS INC [US] ) 17 Apri l 2003 (2003-04-17)<br>abstract<br>page 1, l i ne 10 - l i ne 11<br>page 3, l i ne 15 - page 6, l i ne 5<br>page 7, l i ne 9 - page 9, l i ne 27<br>page 10, l i ne 20 - page 11, l i ne 14<br>page 12, l i ne 4 - page 12, l i ne 23<br>page 13, l i ne 4 - page 13, l i ne 6<br>page 13, l i ne 25 - page 17, l i ne 2<br>page 18, l i ne 15 - page 20, l i ne 14;<br>cl aims 1, 16,30,31 ,34,49 ,50,59 ; f i gures 1,3,4A | 1-12 |

3

# INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/EP2011/060748

| Patent document oited in searoh report | Publication date | Patent family member(s) | Publication date |
|---|---|---|---|
| US 2004243989 A1 | 02-12-2004 | AU 2002363920 A1 | 25-05-2004 |
| | | WO 2004040445 A1 | 13-05-2004 |
| US 2002046298 A1 | 18-04-2002 | US 2002184399 A1 | 05-12-2002 |
| | | US 6591416 B1 | 08-07-2003 |
| US 6289506 B1 | 11-09-2001 | NONE | |
| WO 03032155 A2 | 17-04-2003 | AU 2002340087 A1 | 22-04-2003 |
| | | DE 60208710 T2 | 14-09-2006 |
| | | EP 1451682 A2 | 01-09-2004 |
| | | US 2003070161 A1 | 10-04-2003 |

# RAPPORT DE RECHERCHE INTERNATIONALE

Demande internationale n°
PCT/EP2011/060748

A. CLASSEMENT DE L'OBJET DE LA DEMANDE
INV. G06F9/45
ADD.

Selon la classification internationale des brevets (CIB) ou à la fois selon la classification nationale et la CIB

B. DOMAINES SUR LESQUELS LA RECHERCHE A PORTE

Documentation minimale consultée (système de classification suivi des symboles de classement)
G06F

Documentation consultée autre que la documentation minimale dans la mesure où ces documents relèvent des domaines sur lesquels a porté la recherche

Base de données électronique consultée au cours de la recherche internationale (nom de la base de données, et si cela est réalisable, termes de recherche utilisés)
EPO-Internal , WPI Data, IBM-TDB, INSPEC, COMPENDEX

C. DOCUMENTS CONSIDERES COMME PERTINENTS

| Catégorie* | Identification des documents cités, avec, le cas échéant, l'indication des passages pertinents | no. des revendications visées |
|---|---|---|
| X | US 2004/243989 Al (OWENS HOWARD DEWEY [US] ET AL OWENS HOWARD DEWEY [US] ET AL)<br>2 décembre 2004 (2004-12-02)<br>abrégé; figures 2,4,5 ,6<br>page 1, alinéa 1<br>page 1, alinéa 13 - page 2, alinéa 16<br>page 2, alinéa 18 - page 3, alinéa 28<br>page 3, alinéa 30 - page 4, alinéa 32;<br>revendi cati ons 1-5 ,7-12 , 14-16, 18-22<br>-----<br>-/- · | 1-12 |

[X] Voir la suite du cadre C pour la fin de la liste des documents

[X] Les documents de familles de brevets sont indiqués en annexe

* Catégories spéciales de documents cités:

"A" document définissant l'état général de la technique, non considéré comme particulièrement pertinent

"E" document antérieur, mais publié à la date de dépôt international ou après cette date

"L" document pouvant jeter un doute sur une revendication de priorité ou cité pour déterminer la date de publication d'une autre citation ou pour une raison spéciale (telle qu'indiquée)

"O" document se référant à une divulgation orale, à un usage, à une exposition ou tous autres moyens

"P" document publié avant la date de dépôt international, mais postérieurement à la date de priorité revendiquée

"T" document ultérieur publié après la date de dépôt international ou la date de priorité et n'appartenenant pas à l'état de la technique pertinent, mais cité pour comprendre le principe ou la théorie constituant la base de l'invention

"X" document particulièrement pertinent; l'invention revendiquée ne peut être considérée comme nouvelle ou comme impliquant une activité inventive par rapport au document considéré isolément

"Y" document particulièrement pertinent; l'invention revendiquée ne peut être considérée comme impliquant une activité inventive lorsque le document est associé à un ou plusieurs autres documents de même nature, cette combinaison étant évidente pour une personne du métier

"&" document qui fait partie de la même famille de brevets

| Date à laquelle la recherche internationale a été effectivement achevée | Date d'expédition du présent rapport de recherche internationale |
|---|---|
| 5 août 2011 | 16/08/2011 |

| Nom et adresse postale de l'administration chargée de la recherche internationale | Fonctionnaire autorisé |
|---|---|
| Office Européen des Brevets, P.B. 5818 Patentlaan 2<br>NL - 2280 HV Rijswijk<br>Tel. (+31-70) 340-2040,<br>Fax: (+31-70) 340-3016 | Lelait, Sylvain |

3

Formulaire PCT/ISA/210 (deuxième feuille) (avril 2005)

C(suite). DOCUMENTS CONSIDERES COMME PERTINENTS

| Catégorie* | Identification des documents cités, avec, le cas échéant, l'indication des passages pertinents | no. des revendications visées |
|---|---|---|
| X | US 2002/046298 Al (BAK LARS [US] ET AL) 18 avril 2002 (2002-04-18)<br>abrégé; figures 4-6<br>page 1, alinéa 10 - page 2, alinéa 13<br>page 3, alinéa 51 - page 5, alinéa 80<br>page 7, alinéa 13 - alinéa 14;<br>revendications 1,2,8-11 ,13, 19-23 ,29-32 ,36-38,43-45 ,49-51 , 57-59 ,65 | 1-12 |
| X | DE VERDIERE V C ET AL: "Speedup prédiction for sélective compilation of embedded Java programs" , EMBEDDED SOFTWARE. SECOND INTERNATIONAL CONFERENCE, EMS0FT 2002 . PR0CEEDINGS (LECTURE NOTES IN COMPUTER SCIENCE VOL. 2491) SPRINGER-VERLAG BERLIN , GERMANY, 2002 , pages 227-239 , XP002618859 , ISBN : 3-540-44307-X<br>abrégé<br>page 227 , alinéa 2 - page 228, alinéa 4<br>page 227 , dernier alinéa - page 229 , alinéa 5<br>page 231 , alinéa 4<br>page 232 , dernier alinéa - page 233 , alinéa 3<br>page 233 , dernier alinéa - page 234, alinéa 1<br>page 235 , alinéa 2<br>page 237 , alinéa 1 | 1-12 |
| X | US 6 289 506 Bl (KW0NG ALICE [US] ET AL) 11 septembre 2001 (2001-09-11)<br>abrégé; figures 4,5 ,7-10<br>colonne 2, ligne 35 - colonne 2, ligne 44<br>colonne 3, ligne 45 - colonne 4, ligne 37<br>colonne 5, ligne 40 - colonne 6, ligne 6<br>colonne 8, ligne 20 - colonne 10, ligne 47; revendications 1,2,5-12 ,14-19 ,21-27 ,29-32 | 1-12 |
| A | W0 03/032155 A2 (SUN MICROSYSTEMS INC [US] ) 17 avril 2003 (2003-04-17)<br>abrégé<br>page 1, ligne 10 - ligne 11<br>page 3, ligne 15 - page 6, ligne 5<br>page 7, ligne 9 - page 9, ligne 27<br>page 10, ligne 20 - page 11, ligne 14<br>page 12, ligne 4 - page 12, ligne 23<br>page 13, ligne 4 - page 13, ligne 6<br>page 13, ligne 25 - page 17, ligne 2<br>page 18, ligne 15 - page 20, ligne 14;<br>revendications 1,16,30,31 ,34,49 ,50,59 ;<br>figures 1,3,4A | 1-12 |

3

# RAPPORT DE RECHERCHE INTERNATIONALE

**Renseignements relatifs aux membres de familles de brevets**

Demande internationale n°
PCT/EP2011/060748

| Document brevet cité au rapport de recherche | Date de publication | Membre(s) de la famille de brevet(s) | Date de publication |
|---|---|---|---|
| US 2004243989 A1 | 02-12-2004 | AU 2002363920 A1 | 25-05-2004 |
| | | WO 2004040445 A1 | 13-05-2004 |
| US 2002046298 A1 | 18-04-2002 | US 2002184399 A1 | 05-12-2002 |
| | | US 6591416 B1 | 08-07-2003 |
| US 6289506 B1 | 11-09-2001 | AUCUN | |
| WO 03032155 A2 | 17-04-2003 | AU 2002340087 A1 | 22-04-2003 |
| | | DE 60208710 T2 | 14-09-2006 |
| | | EP 1451682 A2 | 01-09-2004 |
| | | US 2003070161 A1 | 10-04-2003 |