(19) RÉPUBLIQUE FRANÇAISE

INSTITUT NATIONAL
DE LA PROPRIÉTÉ INDUSTRIELLE

PARIS

(11) **N° de publication :** **2 879 319**

(à n'utiliser que pour les commandes de reproduction)

(21) **N° d'enregistrement national :** **04 13294**

(51) Int Cl[8] : **G 06 F 12/04** (2006.01), G 06 K 19/073

(12) DEMANDE DE BREVET D'INVENTION A1

(22) **Date de dépôt :** 14.12.04.

(30) **Priorité :**

(43) **Date de mise à la disposition du public de la demande :** 16.06.06 Bulletin 06/24.

(56) **Liste des documents cités dans le rapport de recherche préliminaire :** *Se reporter à la fin du présent fascicule*

(60) **Références à d'autres documents nationaux apparentés :**

(71) **Demandeur(s) :** *GEMPLUS Société anonyme* — FR.

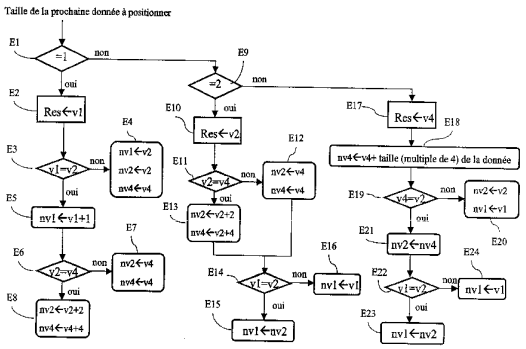
(72) **Inventeur(s) :** REQUET ANTOINE, LAGOSANTO LAURENT et VANDEWALLE JEAN JACQUES.

(73) **Titulaire(s) :**

(74) **Mandataire(s) :** NOVAGRAAF TECHNOLOGIES.

(54) PROCEDE DE POSITIONNEMENT DE DONNEES ELEMENTAIRES D'UNE STRUCTURE DE DONNEES DANS UNE MEMOIRE.

(57) L'invention concerne un procédé de positionnement de données élémentaires (b1, i1, s1) d'une structure de données dans une mémoire organisée en mots d'au moins quatre octets, comprenant, suivant la taille de la prochaine donnée à positionner, la détermination de la position (Res) à affecter en mémoire à ladite donnée élémentaire en fonction d'une description de ladite structure en mémoire, et la mise à jour de ladite description de ladite structure en mémoire, caractérisé en ce que ladite description est basée sur au moins trois indices (v1, v2, v4) indiquant respectivement la prochaine position en mémoire disponible pour placer une donnée de taille un octet, la prochaine position en mémoire disponible pour placer une donnée de taille deux octets et la prochaine position en mémoire disponible pour placer une donnée de taille au moins trois octets.



FR 2 879 319 - A1

PROCEDE DE POSITIONNEMENT DE DONNEES ELEMENTAIRES D'UNE STRUCTURE DE DONNEES DANS UNE MEMOIRE

La présente invention porte sur les logiciels et environnements d'exécution embarqués dans un appareil électronique portatif doté d'un microcontrôleur réunissant un processeur et des mémoires, tel qu'une carte à puce par exemple.

Plus particulièrement, l'invention concerne un procédé de positionnement de données élémentaires d'une structure de données dans une mémoire.

On utilise en effet, pour organiser le contenu d'une mémoire, des structures de données représentant une organisation des données élémentaires d'un code dans une structure logique particulière. Ces structures de données sont alors prévues pour être allouées en mémoire, c'est-à-dire qu'une zone mémoire est réservée pour stocker cette structure de données, lors du chargement et de l'installation du code en mémoire par l'environnement d'exécution.

Le code logiciel en lui-même ne contient pas d'information sur la façon de placer les données élémentaires de la structure en mémoire (ordre, disposition), mais fournit généralement des informations sur le type et la taille de ces données. Les données élémentaires sont typiquement codées soit sur 1 octet, soit sur 2 octets, soit sur des multiples 4 octets. Lors de l'affectation des données élémentaires d'une structure de données en mémoire, il est nécessaire de respecter les contraintes d'alignement pour la représentation des données en

mémoire, qui sont imposées par l'architecture matérielle du processeur. Ainsi, les données codées sur un octet peuvent être positionnées à n'importe quelle adresse mémoire, alors que les données codées sur deux octets ne peuvent être positionnées qu'à une adresse mémoire multiple de 2, et les données codées sur des multiples de quatre octets ne peuvent être positionnées qu'à une adresse mémoire multiple de 4.

Aujourd'hui, les architectures matérielles reposent principalement sur l'adressage de mémoire 32 bits, la taille des mots mémoire étant alors limitée à 4 octets. La description qui suit sera faite en référence à cette taille de mot mémoire bien que l'invention ne se limite pas à de telles structures de mémoire.

Plusieurs approches classiques peuvent alors être envisagées pour calculer le positionnement de données élémentaires d'une structure de données en mémoire. Les exemples ci-après sont basés sur une structure de données de type classe et vont ainsi décrire le chargement d'une classe par une machine virtuelle embarquée, qui constitue l'interface permettant de traduire en instructions exécutables par le processeur de l'appareil hôte, les langages intermédiaires orientés objet tels que le pseudo-code Java (désigné par bytecode en langue anglaise), obtenu après compilation du langage source Java.

Pour rappel, un logiciel Java est classiquement diffusé sous forme d'un ensemble de modules constitué de fichiers .class. Ces fichiers correspondent donc à une forme compilée du logiciel. Chaque fichier compilé

correspond à une structure de données de type classe et comprend en tant que telle la description des éléments de la classe, notamment ses champs, qui constituent dans ce contexte les données élémentaires à placer en mémoire pour l'exécution du pseudo-code Java.

Le chargeur de la machine virtuelle est alors prévu pour charger et transformer les informations du fichier .Class en des structures de données en mémoire de travail, qui sont propres à la machine virtuelle et qui permettent à cette machine virtuelle d'exécuter le logiciel.

Pour illustrer le processus de positionnement des champs d'une classe lors de son chargement dans une structure de mémoire selon l'état de la technique, soit le morceau de pseudo-code suivant, décrivant une classe comportant trois champs de tailles différentes.

```
Class C {       //la classe est nommée 'C'
Byte b1 ;       //un champ nommé 'b1' de taille 1 octet
Int i1 ;        // un champ nommé 'i1' de taille 4 octets
Short s1 ;      // un champ nommé 's1' de taille 2 octets
}
```

La taille optimale des instances de la classe est égale à la somme de la taille de chacun de ses champs, soit 7 octets pour les instances de la classe 'C'.

La figure 1 illustre une zone 10 dans une structure de mémoire 32 bits, telle qu'elle se présente à l'issue d'un processus de stockage des champs du pseudo-code considéré, où chaque champ est placé à la

suite de l'autre, en tenant compte des contraintes d'alignement décrites plus haut. Ainsi, pour la classe 'C' considérée, le champ b1, codé sur 1 octet, est positionné à l'adresse mémoire 0, le champ i1, codé sur 4 octets, est positionné nécessairement à l'adresse mémoire 4 du fait des contraintes d'alignement du processeur, et le champ s1, codé sur 2 octets, est alors positionné à l'adresse mémoire 8.

Chaque champ de la classe occupe donc la même place en mémoire. On s'aperçoit que cette solution n'est pas du tout optimale en ce qui concerne la taille totale d'une instance de cette classe, qui dans cet exemple occupe 12 octets, alors que 5 octets restent inoccupés (octets apparaissant en blanc sur la figure), correspondant à autant de place perdue en mémoire.

Une solution permettant d'optimiser l'occupation de la mémoire lors du positionnement des champs d'une structure de données type classe a alors été développée et est présentée à la figure 2. Cette solution consiste à établir une carte de la mémoire 20, permettant de mémoriser pour chaque adresse de la mémoire 10, si elle est occupée ou non. Au départ, tous les bits de la carte dont initialisés à 0. Selon cette méthode, le premier champ b1, codé sur 1 octet, est positionné à l'adresse mémoire 0 et le bit correspondant de la carte mémoire pour cet octet est alors placé à 1, indiquant que cet octet est désormais occupé.

Pour positionner le champ suivant i1, codé sur 4 octets, on va parcourir la carte mémoire depuis son début jusqu'à trouver le prochain emplacement bien aligné (c'est-à-dire à une adresse multiple de 4), de

taille 4 octets, qui soit disponible. En conséquence, le champ i1 est positionné à l'adresse mémoire de départ 4 et la carte mémoire est mise à jour. Pour ce faire, les bits correspondant de la carte mémoire pour les octets des adresses mémoires 4 à 7 sont placés à 1, indiquant que ces octets sont désormais occupés.

Puis, pour positionner le champ suivant s1, codé sur 2 octets, on va à nouveau parcourir la carte mémoire depuis son début jusqu'à trouver le prochain emplacement bien aligné (c'est-à-dire à une adresse multiple de 2), de taille 2 octets, qui soit disponible. Cet emplacement correspond aux adresses mémoires 2 et 3. Le champ s1 est alors positionné à l'adresse mémoire 2 et la carte mémoire est mise à jour. Pour ce faire, les bits correspondant de la carte mémoire pour les octets des adresses mémoires 4 à 7 sont placés à 1, indiquant que ces octets sont désormais occupés. La carte mémoire permet ainsi de fournir lors du processus de positionnement des champs, une description de la structure de données en mémoire.

On obtient alors un positionnement optimal des champs de la classe dans la zone mémoire qui lui est affectée. En effet, dans cet exemple les instances de la classe 'C' auraient une taille de 8 octets, et on a uniquement un octet de perdu au niveau de l'occupation de la mémoire pour une instance de cette classe.

Cette solution présente néanmoins deux inconvénients majeurs. Tout d'abord, elle est consommatrice en mémoire. En effet, elle nécessite, lors du calcul du placement des champs, de stocker la carte mémoire, cette dernière occupant une quantité de

mémoire non négligeable, puisque proportionnelle à la taille de la structure de mémoire dont elle reproduit la cartographie. Cette méthode pourrait donc s'avérer inadaptée à une implémentation dans des environnements contraints en mémoire, tels que les cartes à puce par exemple, où une utilisation optimale de la ressource mémoire est une exigence forte.

En outre, dans ce même contexte, pour pouvoir assurer la notion d'héritage de classe, la cartographie de la mémoire doit être maintenue en mémoire. En effet, lorsque la machine virtuelle reçoit une classe fille d'une classe mère déjà traitée, elle doit tenir compte, pour le calcul du positionnement des champs de la classe fille, de la position des champs de la classe mère : ils existent aussi dans la classe fille et sont positionnés au même endroit.

Un autre inconvénient de cette solution concerne l'inefficacité en termes de temps d'exécution du calcul de positionnement des champs. En effet, cette méthode impose de parcourir la carte mémoire depuis son début autant de fois qu'il y a de champs à placer dans la structure mémoire. Ce processus peut devenir fortement pénalisant en présence d'un grand nombre de champs à placer, notamment dans les environnements embarquées, disposant généralement de faibles ressources de calcul.

L'invention vise à résoudre un ou plusieurs de ces inconvénients, en proposant un procédé permettant d'obtenir un positionnement optimal en mémoire de données élémentaires d'une structure de données, tout en étant efficace en termes de consommation de mémoire et de temps de calcul.

L'invention a ainsi pour objet un procédé de positionnement de données élémentaires d'une structure de données dans une mémoire organisée en mots d'au moins quatre octets, comprenant, suivant la taille de la prochaine donnée à positionner, la détermination de la position à affecter en mémoire à ladite donnée élémentaire en fonction d'une description de ladite structure en mémoire, et la mise à jour de ladite description de ladite structure en mémoire, caractérisé en ce que ladite description est basée sur au moins trois indices indiquant respectivement la prochaine position en mémoire disponible pour placer une donnée de taille un octet, la prochaine position en mémoire disponible pour placer une donnée de taille deux octets et la prochaine position en mémoire disponible pour placer une donnée de taille au moins trois octets.

Selon un mode de réalisation, la détermination de la position en mémoire de la donnée élémentaire à positionner en fonction de la description de la structure de données en mémoire, consiste à affecter à ladite position la valeur courante de l'indice correspondant à la taille de la donnée considérée.

Selon un mode de réalisation, la mise à jour de la description de la structure de données en mémoire consiste à affecter de nouvelles valeurs aux dits indices en fonction de leurs positions relatives, de sorte que lesdites valeurs mises à jour décrivent respectivement la nouvelle position en mémoire la plus petite disponible où une prochaine donnée élémentaire de taille un octet, deux octets, ou au moins trois octets pourra être positionnée.

Le procédé selon l'invention s'applique avantageusement au positionnement en mémoire des champs d'une classe d'un code logiciel compilé en langage intermédiaire orienté objet, la classe pouvant être une classe héritant d'une autre classe.

Le procédé selon l'invention s'applique aussi avantageusement au positionnement en mémoire dans une pile d'exécution, des variables d'un ensemble de variables locales d'une fonction.

L'invention concerne également un chargeur d'un code logiciel compilé en langage intermédiaire orienté objet destiné à être chargé dans un appareil électronique portatif, caractérisé en ce qu'il est susceptible d'être stocké dans une mémoire non volatile de l'appareil pour la mise en œuvre du procédé selon l'invention.

L'invention concerne encore un appareil électronique portatif comprenant une mémoire non volatile mémorisant le chargeur tel qu'il vient d'être décrit.

De préférence, cet appareil est une carte à puce.

D'autres caractéristiques et avantages de la présente invention apparaîtront plus clairement à la lecture de la description suivante donnée à titre d'exemple illustratif et non limitatif et faite en référence aux figures annexées dans lesquelles :

- la figure 1 est une représentation schématique d'une zone mémoire dans une structure mémoire 32 bits, dans laquelle ont été positionnés les champs d'une classe selon un procédé de l'art antérieur ;

- la figure 2 est une représentation schématique de ladite zone mémoire, dans laquelle ont été positionnés les champs d'une classe selon un procédé optimisée de l'art antérieur ;

- la figure 3 est un organigramme illustrant la mise en oeuvre du procédé selon la présente invention.

Selon l'invention, le procédé de positionnement de données élémentaires d'une structure de données en mémoire est basé sur une description particulière de la structure en mémoire. Cette description repose sur le calcul et la mise à jour de trois indices, v1, v2 et v4, en lieu et place du maintient en mémoire de la carte de la mémoire de la solution de l'art antérieur décrite plus haut.

Les indices v1, v2 et v4 sont prévus pour décrire respectivement la prochaine position ou adresse mémoire disponible (et donc bien alignée) où une donnée codée sur 1 octet pourra être placée, la prochaine adresse mémoire disponible où une donnée codée sur 2 octets pourra être placée et la prochaine adresse mémoire disponible où une donnée codée sur 3 octets, 4 octets ou plus (dans ce cas, on arrondira la taille de la donnée au multiple de 4 supérieur) pourra être placée. Du fait des contraintes d'alignement, des données codées sur 3, 4, 5 octets et plus ne peuvent être alignées que sur des multiples de 4 en adresse mémoire.

Les trois indices v1, v2 et v4 nécessaires à la mise en œuvre du procédé de positionnement selon l'invention, peuvent par exemple être codés chacun sur 2 ou 4 octets. Ils occupent alors au maximum 6 ou 12 octets en mémoire. L'utilisation de tels indices

présente donc l'avantage de ne consommer qu'une quantité fixe de mémoire, indépendante de la taille de la structure de mémoire à décrire.

L'organigramme de la figure 3 illustre la mise en œuvre du procédé de l'invention basé sur l'utilisation des trois indices v1, v2 et v4.

Soient des données élémentaires d'une structure de données de tailles 1, 2 et multiples de 4 octets à positionner en mémoire. Les trois indices v1, v2 et v4 sont initialisés à zéro au début du processus et on donne en entrée la taille de la prochaine donnée à positionner en mémoire.

Plus précisément, on initialise les indices à zéro si on est en adressage relativement au début de la structure de données, sinon les indices sont initialisés avec l'adresse de début de structure dans la mémoire. L'initialisation des indices n'intervient que la première fois où l'on a à positionner une donnée élémentaire dans la structure. Par la suite, les indices conservent leurs valeurs courantes au fur et à mesure de l'évolution de la structure de données en mémoire.

En E1, on regarde si la taille de la donnée est égale à 1. Dans cette hypothèse, en E2, la position (nommée « Res ») qui est allouée à cette donnée en mémoire est donnée par la valeur courante de l'indice v1. La position des données en mémoire est en effet déterminée par la valeur courante de l'indice correspondant à la taille de la donnée considérée. Les valeurs des trois indices doivent alors être mises à jour selon le processus suivant.

En E3, on regarde si v1 est égal à v2. Si ce n'est pas le cas, on passe en E4 où la nouvelle valeur de l'indice v1 mise à jour, nommée nv1, prend la valeur courante v2, la nouvelle valeur de l'indice v2 mise à jour, nommée nv2, reste égale à la valeur courante v2, et la nouvelle valeur de l'indice v4 mise à jour, nommée nv4, reste égale à la valeur courante v4.

Par contre, dans le cas où l'égalité testée en E3 est vérifiée, on passe en E5, où la valeur de l'indice v1 est mise à jour. Cette nouvelle valeur nv1 de l'indice v1 mise à jour prend dans ce cas la valeur v1+1. Pour la mise à jour des valeurs des autres indices v2 et v4, on regarde alors en E6 si v2 est égal à v4. Si ce n'est pas le cas, on passe en E7, où la nouvelle valeur nv2 de l'indice v2 mise à jour prend la valeur courante v4, et la nouvelle valeur nv4 de l'indice v4 mise à jour, reste égale à la valeur courante v4. Par contre, dans le cas où l'égalité testée en E6 est vérifiée, on passe en E8, où dans ce cas, la nouvelle valeur nv4 de l'indice v4 mise à jour prend la valeur v4+4, et la nouvelle valeur nv2 de l'indice v2 mise à jour prend la valeur v2+2.

Les trois indices sont donc mis à jour de sorte qu'ils correspondent chacun à la nouvelle adresse mémoire la plus petite disponible où une prochaine donnée respectivement de taille 1, 2, ou multiple de 4 octets pourra être positionnée. Cette mise à jour des indices est avantageusement déduite d'une simple analyse de leurs positions relatives.

De façon similaire à ce qui vient d'être décrit pour une donnée de taille 1, si, en E9, la prochaine

donnée à placer est de taille 2 octets, la position Res qui est allouée en E10 à cette donnée en mémoire est donnée, selon le principe de l'invention, par la valeur courante de l'indice v2. La position allouée à une donnée de taille 2 en mémoire est en effet déterminée par la valeur courante de l'indice correspondant à la taille de la donnée considérée, à savoir dans ce cas, v2.

Une fois le positionnement de la donnée considérée déterminée, les valeurs des trois indices v1, v2 et v4 doivent alors être mises à jour.

Pour ce faire, on regarde en E11 si v2 est égal à v4. Si ce n'est pas le cas, on passe en E12, où la nouvelle valeur nv2 de l'indice v2 mise à jour prend la valeur courante v4, et la nouvelle valeur nv4 de l'indice v4 mise à jour, reste égale à la valeur courante v4. Par contre, dans le cas où l'égalité testée en E11 est vérifiée, on passe en E13, où dans ce cas, la nouvelle valeur nv4 de l'indice v4 mise à jour prend la valeur v2+4, et la nouvelle valeur nv2 de l'indice v2 mise à jour prend la valeur v2+2.

Pour ce qui est de la mise à jour de l'indice v1, on passe en E14, où on vérifie l'égalité v1 = v2. Si c'est le cas, en E15, la nouvelle valeur nv1 de l'indice v1 mise à jour prend la valeur nv2 de l'indice v2 qui vient d'être mis à jour, sinon, en E16, la nouvelle valeur nv1 de l'indice v1 mise à jour garde la valeur courante v1.

Enfin, si la prochaine donnée à placer a une taille de 3 octets ou plus, le placement Res qui est allouée en E17 à cette donnée en mémoire est

donnée, selon le principe de l'invention, par la valeur courante de l'indice v4. Il convient de noter que si la taille de la donnée considérée est supérieure à 4 octets, on arrondira sa taille pour les calculs de positionnement et de mise à jour des indices à la valeur multiple de 4 immédiatement supérieure ou égale (ainsi, une donnée codée sur 3 octets verra sa taille arrondie à 4 octets, une donnée codée sur 4 octets verra sa taille demeurée à 4 octets, une donnée codée sur 5 ou 6 ou 7 ou 8 octets verra sa taille arrondie à 8 octets, etc.).

Ainsi, la nouvelle valeur nv4 est mise à jour en E18 et prend la valeur courante v4 à laquelle on ajoute la taille de la donnée considérée, arrondie à la valeur multiple de 4 immédiatement supérieure ou égale.

Pour ce qui est de la mise à jour des indices v1 et v2, on passe en E19, où on vérifie l'égalité v4 = v2. Si les deux indices v4 et v2 ont des valeurs différentes, la mise à jour est effectuée de la façon suivante en E20 où la nouvelle valeur nv2 mise à jour demeure inchangée et garde la valeur courante v2, et la nouvelle valeur nv1 mise à jour demeure inchangée et garde la valeur courante v1. Par contre, si l'égalité en E19 est vérifiée, la nouvelle valeur nv2 de l'indice v2 est mise à jour en E21 et prend la valeur nv4 qui vient d'être mis à jour. Puis, la mise à jour de l'indice v1 consiste à vérifier en E22 l'égalité v1=v2. Si c'est le cas, en E23, la nouvelle valeur nv1 de l'indice v1 mise à jour prend la nouvelle valeur nv2 de l'indice v2, sinon, en E24, la nouvelle valeur nv1 de l'indice v1 mise à jour garde la valeur courante v1.

Ainsi, le procédé de l'invention a simplement besoin en entrée de la taille de la prochaine donnée élémentaire de la structure de données à placer dans la mémoire et de la description de cette structure en mémoire, qui est fournie par les trois indices v1, v2 et v4, indiquant respectivement la prochaine adresse mémoire disponible et bien alignée pour placer une donnée de taille 1, la prochaine adresse mémoire disponible et bien alignée pour placer une donnée de taille 2 et la prochaine adresse mémoire disponible et bien alignée pour placer une donnée de taille multiple de 4. On obtient alors en sortie la position en mémoire (Res) qui doit être allouée à la donnée à placer et la description mise à jour de la structure en mémoire, qui est déduite d'une simple analyse des positions relatives des indices.

Le procédé de positionnement d'une donnée en mémoire basé sur les trois indices v1, v2 et v4, qui vient d'être décrit est plus particulièrement prévu pour répondre aux contraintes d'alignement inhérentes aux structures de mémoires de type 32 bits, où la taille des mots mémoire est donc de 4 octets. Toutefois, il pourrait être envisagé, sans pour autant sortir du cadre de l'invention, d'utiliser plus d'indices si les contraintes d'alignement matériel étaient différentes. Cela pourra être le cas par exemple avec une mémoire organisée en mots de 8 octets (64 bits) où les règles d'alignement imposent que les données de 1 octet ne peuvent être positionnées qu'à des adresses multiples de 1, les données de 2 octets ne peuvent être positionnées qu'à des adresses multiples

de 2, les données de 3 et 4 octets ne peuvent être positionnées qu'à des adresses multiples de 4 et les données de taille autre ne peuvent être positionnées qu'à des adresses multiples de 8. Dans cet exemple, il faudrait alors prévoir 4 indices pour pouvoir décrire complètement la structure en mémoire et modifier en conséquence les règles de mise à jour des indices dans l'algorithme décrit en figure 3.

De manière générale, le procédé de positionnement de données en mémoire selon l'invention est prévu pour s'appliquer à tous les environnements d'exécution où il faut organiser la mémoire et stocker des données élémentaires de tailles variables dans la mémoire de façon optimisée en termes d'occupation de la mémoire, en tenant compte des contraintes d'alignement matériel.

Ainsi, le procédé selon l'invention s'applique au calcul du positionnement des champs d'une structure de données en général et, par exemple, au calcul du positionnement des champs d'une classe d'un code logiciel compilé en langage intermédiaire orienté objet, lors du chargement de la classe en mémoire de travail par un chargeur d'une machine virtuelle embarquée. Le procédé de l'invention est de plus particulièrement avantageux dans le cas où la classe à charger est une classe qui hérite d'une autre classe. En effet, lorsque la machine virtuelle reçoit une classe fille d'une classe mère déjà traitée, elle doit tenir compte, pour le calcul du positionnement des champs de la classe fille, de la position des champs de la classe mère dont la classe fille hérite. A cet effet, les trois indices v1, v2 et v4 déjà utilisés

lors du calcul du positionnement des champs de la classe mère sont ajoutés à l'information de la classe, de sorte qu'ils peuvent être réutilisés pour calculer les positions des champs de la classe fille lors de son chargement, qui sont alors ajoutés aux champs de la classe mère déjà positionnés.

Le procédé de l'invention s'applique ainsi typiquement à la plate-forme Java Card (marque déposée) ou .Net (marque déposée).

Egalement, une autre application du procédé selon l'invention concerne le positionnement d'un ensemble de variables locales d'une fonction dans une pile d'exécution, cet ensemble de variables locales formant une structure de données. En effet, de la même manière que chaque champ se voit affecter une position dans une instance de classe en mémoire, les variables locales d'une fonction stockées dans la pile doivent se voir affecter une position par rapport à la position de la pile à l'entrée de la fonction.

REVENDICATIONS

1. Procédé de positionnement de données élémentaires (b1, i1, s1) d'une structure de données dans une mémoire organisée en mots d'au moins quatre octets, comprenant, suivant la taille de la prochaine donnée à positionner, la détermination de la position (Res) à affecter en mémoire à ladite donnée élémentaire en fonction d'une description de ladite structure en mémoire, et la mise à jour de ladite description de ladite structure en mémoire, caractérisé en ce que ladite description est basée sur au moins trois indices (v1, v2, v4) indiquant respectivement la prochaine position en mémoire disponible pour placer une donnée de taille un octet, la prochaine position en mémoire disponible pour placer une donnée de taille deux octets et la prochaine position en mémoire disponible pour placer une donnée de taille au moins trois octets.

2. Procédé selon la revendication 1, caractérisé en ce que la détermination de la position en mémoire de la donnée élémentaire à placer en fonction de la description de la structure de données en mémoire, consiste à affecter à ladite position (Res) la valeur courante de l'indice (v1, v2, v4) correspondant à la taille de la donnée considérée.

3. Procédé selon la revendication 1 ou 2, caractérisé en ce que la mise à jour de la description de la structure de données en mémoire consiste à

affecter de nouvelles valeurs aux dits indices (v1, v2, v4) en fonction de leurs positions relatives, de sorte que lesdites valeurs mises à jour (nv1, nv2, nv4) décrivent respectivement la nouvelle position en mémoire la plus petite disponible où une prochaine donnée élémentaire de taille un octet, deux octets, ou au moins trois octets pourra être positionnée.

4. Procédé selon l'une quelconque des revendications précédentes, caractérisé en ce qu'il consiste à positionner en mémoire des champs d'une classe d'un code logiciel compilé en langage intermédiaire orienté objet.

5. Procédé selon la revendication 4, caractérisé en ce la classe est une classe héritant d'une autre classe.

6. Procédé selon l'une quelconque des revendications 1 à 3, caractérisé en ce qu'il consiste à positionner en mémoire dans une pile d'exécution, des variables d'un ensemble de variables locales d'une fonction.

7. Chargeur d'un code logiciel compilé en langage intermédiaire orienté objet destiné à être chargé dans un appareil électronique portatif, caractérisé en ce qu'il est susceptible d'être stocké dans une mémoire non volatile de l'appareil pour la mise en œuvre du procédé selon l'une quelconque des revendications 1 à 5.

8. Appareil électronique portatif comprenant une mémoire non volatile mémorisant le chargeur de la revendication 7.

9. Appareil selon la revendication 8, caractérisé en ce que cet appareil est une carte à puce.

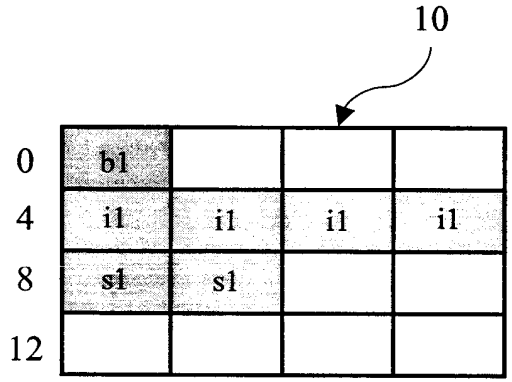


Fig. 1

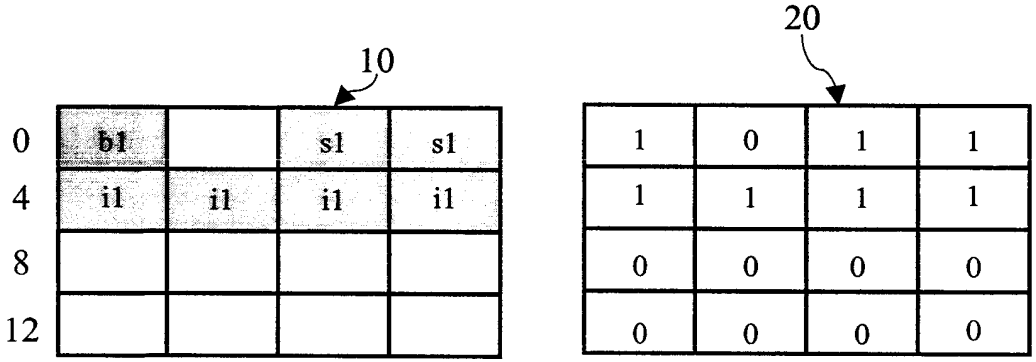


Fig. 2

2879319

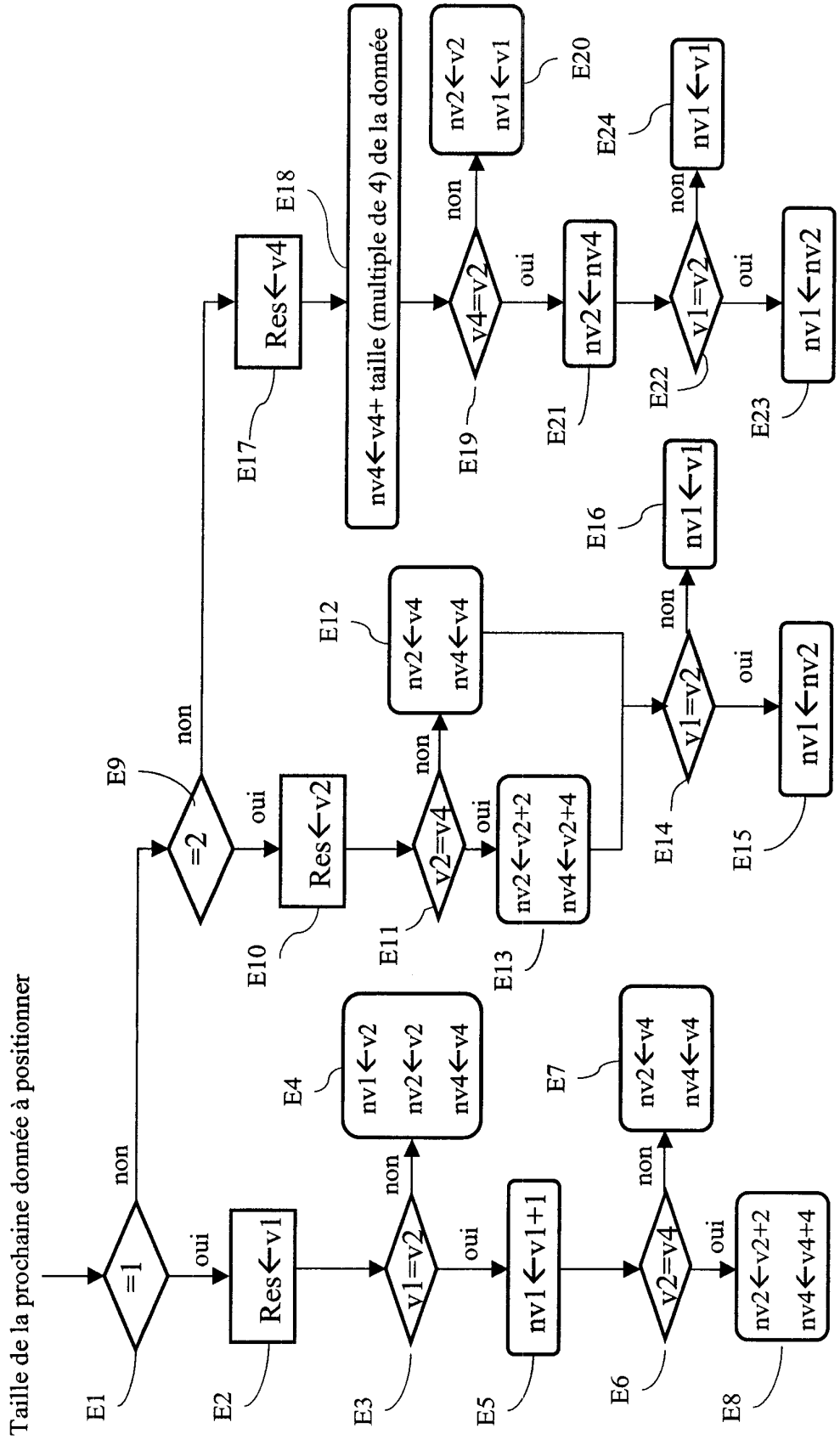

Taille de la prochaine donnée à positionner
E1
=1
non
oui
E2
Res←v1
E3
v1=v2
non
oui
E4
nv1←v2
nv2←v2
nv4←v4
E5
nv1←v1+1
E6
v2=v4
non
oui
E7
nv2←v4
nv4←v4
E8
nv2←v2+2
nv4←v4+4
E9
=2
non
oui
E10
Res←v2
E11
v2=v4
non
oui
E12
nv2←v4
nv4←v4
E13
nv2←v2+2
nv4←v4+4
E14
v1=v2
non
oui
E15
nv1←nv2
E16
nv1←v1
E17
Res←v4
E18
nv4←v4+ taille (multiple de 4) de la donnée
E19
v4=v2
non
oui
E20
nv2←v2
nv1←v1
E21
nv2←nv4
E22
v1=v2
non
oui
E23
nv1←nv2
E24
nv1←v1

Fig. 3

INSTITUT NATIONAL DE LA PROPRIETE INDUSTRIELLE

RAPPORT DE RECHERCHE PRÉLIMINAIRE

établi sur la base des dernières revendications déposées avant le commencement de la recherche

N° d'enregistrement national

FA 658896
FR 0413294

DOCUMENTS CONSIDÉRÉS COMME PERTINENTS		Revendication(s) concernée(s)	Classement attribué à l'invention par l'INPI
Catégorie	Citation du document avec indication, en cas de besoin, des parties pertinentes		
A	US 5 623 654 A (PETERMAN ET AL) 22 avril 1997 (1997-04-22) * colonne 4, ligne 20-29; figure 6 * * colonne 5, ligne 62 - colonne 6, ligne 31; figure 9 *	1,2,7,8	G06K19/073 G06F12/04
A	WO 01/16759 A (CRYPTEC SYSTEMS, INC) 8 mars 2001 (2001-03-08) * abrégé * * page 4, ligne 9,10 * * page 8, ligne 2-10 * * page 9, ligne 11-17; figure 3 *	1-3,7-9	
A	WILSON P R ET AL: "Dynamic Storage Allocation: A Survey and Critical Review" MEMORY MANAGEMENT. INTERNATIONAL WORKSHOP IWMM. PROCEEDINGS, 1995, pages 1-78, XP002262845 * page 30, colonne de gauche, ligne 16-27 * * page 30, colonne de gauche, ligne 35 - page 32, colonne de gauche, ligne 6 * * page 36, colonne de gauche, ligne 3 - page 38, colonne de gauche, ligne 11 * * page 38, colonne de gauche, ligne 36 - page 40, colonne de gauche, ligne 46 * * page 41, colonne de droite, ligne 35 - page 42, colonne de droite, ligne 30 * -/--	1-3,7,8	DOMAINES TECHNIQUES RECHERCHÉS (Int.CL.7) G06F

Date d'achèvement de la recherche	Examinateur
1 août 2005	de Junca, I

CATÉGORIE DES DOCUMENTS CITÉS

X : particulièrement pertinent à lui seul
Y : particulièrement pertinent en combinaison avec un autre document de la même catégorie
A : arrière-plan technologique
O : divulgation non-écrite
P : document intercalaire

T : théorie ou principe à la base de l'invention
E : document de brevet bénéficiant d'une date antérieure à la date de dépôt et qui n'a été publié qu'à cette date de dépôt ou qu'à une date postérieure.
D : cité dans la demande
L : cité pour d'autres raisons

& : membre de la même famille, document correspondant

3

EPO FORM 1503 12.99 (P04C14)

INSTITUT NATIONAL DE LA PROPRIETE INDUSTRIELLE

RAPPORT DE RECHERCHE PRÉLIMINAIRE

établi sur la base des dernières revendications déposées avant le commencement de la recherche

N° d'enregistrement national

FA 658896
FR 0413294

DOCUMENTS CONSIDÉRÉS COMME PERTINENTS		Revendication(s) concernée(s)	Classement attribué à l'invention par l'INPI
Catégorie	Citation du document avec indication, en cas de besoin, des parties pertinentes		
A	LINDBLAD J.: "Handling memory fragmentation" EDN MAGAZINE, [Online] mai 2004 (2004-05), page 47-48,50-51, XP002338069 Extrait de l'Internet: URL:http://a330.g.akamai.net/7/330/2540/20040514141109/www.edn.com/contents/images/417498.pdf> [extrait le 2005-07-26] * page 47, colonne de droite, ligne 4-14; figure 1 * * page 48, colonne du milieu, ligne 9 - page 48, colonne de droite, ligne 7 * * page 50, colonne de gauche, ligne 48 - page 50, colonne du milieu, ligne 35 *	1-3,7,8	
A	DATABASE INSPEC [Online] THE INSTITUTION OF ELECTRICAL ENGINEERS, STEVENAGE, GB; novembre 2002 (2002-11), SHUF Y ET AL: "Creating and preserving locality of Java applications at allocation and garbage collection times" XP002338073 Database accession no. 7624199 * abrégé * & 17TH INTERNATIONAL CONFERENCE ON OBJECT-ORIENTED PROGRAMMING, SYSTEMS, LANGUAGES AND APPLICATIONS (OOPSLA 2002) 4-8 NOV. 2002 SEATTLE, WA, USA, vol. 37, no. 11, 4 novembre 2002 (2002-11-04), pages 13-25, XP002338073 SIGPLAN Notices ACM USA ISSN: 0362-1340 * page 13, colonne de droite, ligne 32-36,38-41 * * page 14, colonne de gauche, ligne 15 - page 14, colonne de droite, ligne 14 * * page 16, colonne de droite, ligne 20-24,33-46 *	4,5	DOMAINES TECHNIQUES RECHERCHÉS (Int.CL.7)

Date d'achèvement de la recherche	Examinateur
1 août 2005	de Junca, I

CATÉGORIE DES DOCUMENTS CITÉS

X : particulièrement pertinent à lui seul
Y : particulièrement pertinent en combinaison avec un autre document de la même catégorie
A : arrière-plan technologique
O : divulgation non-écrite
P : document intercalaire

T : théorie ou principe à la base de l'invention
E : document de brevet bénéficiant d'une date antérieure à la date de dépôt et qui n'a été publié qu'à cette date de dépôt ou qu'à une date postérieure.
D : cité dans la demande
L : cité pour d'autres raisons

& : membre de la même famille, document correspondant

3

EPO FORM 1503 12.99 (P04C14)

ANNEXE AU RAPPORT DE RECHERCHE PRÉLIMINAIRE RELATIF A LA DEMANDE DE BREVET FRANÇAIS NO. FR 0413294 FA 658896

La présente annexe indique les membres de la famille de brevets relatifs aux documents brevets cités dans le rapport de recherche préliminaire visé ci-dessus.
Les dits membres sont contenus au fichier informatique de l'Office européen des brevets à la date du 01-08-2005
Les renseignements fournis sont donnés à titre indicatif et n'engagent pas la responsabilité de l'Office européen des brevets, ni de l'Administration française

Document brevet cité au rapport de recherche	Date de publication	Membre(s) de la famille de brevet(s)	Date de publication
US 5623654 A	22-04-1997	AUCUN	
WO 0116759 A	08-03-2001	US 6480935 B1	12-11-2002
		WO 0116759 A1	08-03-2001

EPO FORM P0465