(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
29 décembre 2005 (29.12.2005)

PCT

(10) Numéro de publication internationale
WO 2005/124544 A2

(51) Classification internationale des brevets[7] : G06F 9/445

(21) Numéro de la demande internationale :
PCT/EP2005/052621

(22) Date de dépôt international : 7 juin 2005 (07.06.2005)

(25) Langue de dépôt : français

(26) Langue de publication : français

(30) Données relatives à la priorité :
04/06493 15 juin 2004 (15.06.2004) FR

(71) Déposant *(pour tous les États désignés sauf US)* : **GEMPLUS** [FR/FR]; Avenue du Pic de Bertagne, Parc d'activité de Gémenos, F-13420 GEMENOS (FR).

(72) Inventeurs; et

(75) Inventeurs/Déposants *(pour US seulement)* : **GRIMAUD, Gilles** [FR/FR]; 57 rue Faidherbe, F-59000 LILLE (FR). **VANDEWALLE, Jean-Jacques** [FR/FR]; 140 rue Raspail, F-59260 LILLE-HELLEMMES (FR).

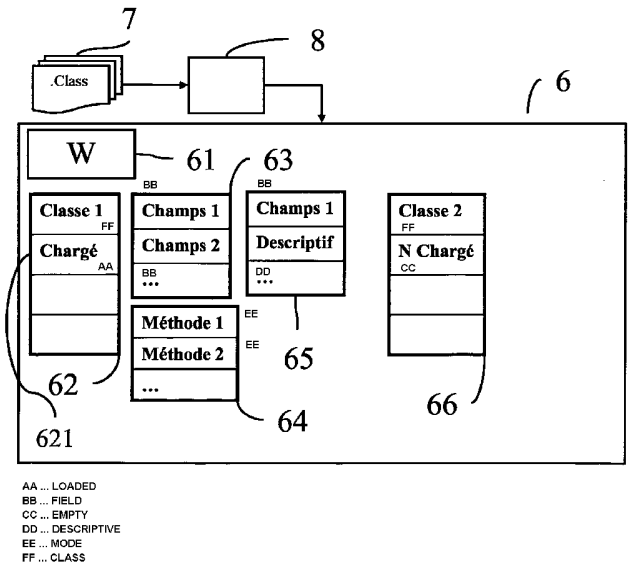
(81) États désignés *(sauf indication contraire, pour tout titre de protection nationale disponible)* : AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KP, KR, KZ, LC, LK,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

(84) États désignés *(sauf indication contraire, pour tout titre de protection régionale disponible)* : ARIPO (BW, GH,

[Suite sur la page suivante]

(54) Title: METHOD FOR LOADING SOFTWARE WITH AN INTERMEDIATE OBJECT ORIENTED LANGUAGE IN A PORTABLE DEVICE

(54) Titre : PROCEDE DE CHARGEMENT D'UN LOGICIEL EN LANGAGE INTERMEDIAIRE ORIENTE OBJET DANS UN APPAREIL PORTATIF



(57) **Abstract:** The invention relates to a method for loading a software comprising several modules (7) loadable in the non-volatile memory of a portable digital device (5), wherein the loading of at least one module consists in testing the existence of a data structure associated with said module in the non-volatile memory (6) and, when necessary, to create said data structure (62) indicating it as empty, in resolving links for totality of internal elements and, afterwards, in marking mark said structure as loaded (621), in testing the existence of an associated data structure for each other module referenced in the loadable module and, when necessary, create said structure and indicate it as empty. Said invention makes it possible to reduce the space occupied by the on-board software during loading of modules.

[Suite sur la page suivante]

WO 2005/124544 A2

GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), eurasien (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), européen (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Publiée :

— *sans rapport de recherche internationale, sera republiée dès réception de ce rapport*

En ce qui concerne les codes à deux lettres et autres abréviations, se référer aux "Notes explicatives relatives aux codes et abréviations" figurant au début de chaque numéro ordinaire de la Gazette du PCT.

(57) Abrégé : L'invention propose un procédé de chargement d'un logiciel comprenant plusieurs modules (7) à charger dans la mémoire non volatile d'un appareil numérique portatif (5), le chargement d'au moins un des modules comprend les étapes suivantes : tester l'existence d'une structure de données associée à ce module dans la mémoire non volatile (6) et créer le cas échéant cette structure de données (62) en la marquant comme non chargée ; résoudre les liens pour l'ensemble des éléments internes puis marquer cette structure de données comme chargée (621) ; pour chaque autre module référencé dans le module en cours de chargement, tester l'existence d'une structure de données associée et le cas échéant la créer et la marquer comme non chargée. L'invention réduit notamment l'espace occupé par le logiciel embarqué au fur et à mesure du chargement des modules.

PROCEDE DE CHARGEMENT D'UN LOGICIEL EN LANGAGE INTERMEDIAIRE ORIENTE OBJET DANS UN APPAREIL PORTATIF

La présente invention porte sur les logiciels et environnements d'exécution embarqués dans un appareil portatif et en particulier sur les procédés de chargement de tels logiciels dans l'appareil portatif.

Des langages intermédiaires de programmation orientée objet ont été développés. Le but principal de ces langages est de rendre leurs logiciels indépendants du matériel sur lequel ils doivent être exécutés. Les programmeurs sont ainsi globalement dégagés des contraintes liées à des matériels spécifiques. La diffusion d'une même version de logiciel pour différents matériels est également possible. Les langages intermédiaires orientés objet tels que le pseudo-code Java (désigné par bytecode en langue anglaise) obtenu après compilation du langage source Java ont ainsi connu un essor conséquent.

Un logiciel Java pour un ordinateur de bureau est classiquement diffusé sous forme d'un ensemble de modules constitué de fichiers .class. Ces fichiers correspondent à une forme compilée du logiciel. Chaque fichier compilé correspond à une structure de données de type classe et comprend en tant que telle les informations de cette classe : à savoir, la description des éléments de la classe (ses constantes, ses champs, ses méthodes), la description des éléments utilisés par la classe et définis dans d'autres classes (des champs et des méthodes), le code des méthodes de la classe

sous forme d' instructions interprétables (pseudo-code) par l'interpréteur de la machine virtuelle Java. Dans un fichier .class les descriptions des éléments de la classe et des éléments utilisés par la classe sont regroupés dans une table de références (constant pool) associant notamment des symboles numériques (leur numéro d'entrée dans le constant pool) à des descriptions d'éléments. Ainsi dans les instructions interprétables (pseudo-code) les références aux éléments de la classe et aux éléments utilisés par la classe se retrouvent « référencés » sous la forme de leur numéro d'entrée dans la table de référence (constant pool). Un fichier .class présente par exemple la structure suivante :

```
ClassFile {
    Id_info ;           //Numéro de version
    Constant_pool ;     //Stocke toutes les références
        aux éléments de la classes et aux éléments
        utilisés par la classe
    Base_info ;         //Drapeaux d'accès, nom de
        classe et superclasse
    Interface_list ;    //Interfaces mises en œuvre par
        la classe
    Field_list ;        //Description des champs de la
        classe
    Method_list ;       //Méthodes de cette classe
        (incluant leur pseudo-code, c'est-à-dire leurs
        instructions interprétables par l'interpréteur
        de la machine virtuelle)
    Attribute_list ;    //Attributs (par exemple :
        informations de débuggage, etc…)
```

}

Les fichiers du programme compilé sont stockés sur une mémoire de masse de l'ordinateur, par exemple un disque dur. L'exécution du logiciel Java est réalisée par la machine virtuelle Java installée sur l'ordinateur. Cette machine virtuelle charge et transforme les informations du ClassFile 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.

Cette étape de chargement (et de transformation) de logiciels Java est réalisable sur la plupart des ordinateurs de bureau présentant des ressources importantes de mémoire vive et de processeur. Ces ressources leur permettent en effet de réaliser le chargement du logiciel en créant à la volée et en mémoire vive les structures de données internes à la machine virtuelle qui permettent ainsi l'exécution du logiciel en mémoire vive. Cette étape de chargement préalable à l'exécution implique notamment une étape d'édition de liens dans laquelle on remplace dans les structures de données créées les références à des éléments internes et externes à la classe par des adresses en mémoire des structures de données ainsi crées.

Cependant, certains systèmes devant embarquer des logiciels Java ont des capacités beaucoup plus limitées qui interdisent ce mode d'exécution. Ainsi, du fait des contraintes de miniaturisation, une carte à puce comprend typiquement entre 1 et 4 KiloOctets de mémoire

vive (utilisée comme mémoire de travail), entre 32 et 64 KiloOctets de mémoire non volatile réinscriptible (utilisée comme mémoire de masse et comme mémoire de travail) et environ 256 KiloOctets de mémoire morte (utilisée pour stocker le code de l'environnement d'exécution). Une telle carte à puce est donc inadaptée à ce mode de fonctionnement qui impliquerait de stocker en mémoire non volatile le fichier .class et de réaliser cette étape de chargement en mémoire volatile à chaque exécution. De plus il y aurait ainsi duplication des informations de la classes dans la carte : celles dans le .class en mémoire non volatile et celles sous forme de structures de données en mémoire volatile.

Une première solution consiste à mémoriser dans la mémoire non volatile de la carte à puce directement les structures de données définies par les spécifications de la machine virtuelle de la carte à puce. Ainsi la machine virtuelle dans la carte n'a plus à réaliser cette opération de chargement puisqu'elle reçoit directement les informations des classes dans son format de structures de données, donc prêts à être exécutés.

Le document US2003/0028686 décrit un tel procédé de chargement. Les fichiers .class sont ainsi transformés en un fichier .cap au format Java Card (marque déposée) sur une machine de chargement, ce fichier étant ensuite chargé dans la carte à puce soit lors de sa fabrication, soit ultérieurement au cours de son utilisation. Le fichier .Cap comprend notamment un

entête, une table de références, un descripteur de méthode et un descripteur d'éléments externes. La machine de chargement génère également un fichier appelé ExportFile, associé au fichier .Cap généré. Ce fichier ExportFile permet de déclarer les éléments publics dans le fichier .Cap, par exemple pour que d'autres applications séparées puissent utiliser ces éléments.

Cependant, l'utilisation de cette solution présente des inconvénients. Java Card présente un jeu d'instructions réduit par rapport à Java. Java Card ne supporte notamment pas le chargement dynamique de classes. L'édition dynamique de liens est limitée après le chargement du fichier .cap dans la carte à puce. Les fichiers .cap chargés ne peuvent pas être générés aisément à partir de fichiers .class faisant appel à des instructions non supportées par Java Card. De plus, l'étape de conversion complique la tâche des programmeurs puisqu'une conversion doit être prévue spécifiquement pour chaque type de carte à puce (en fonction des logiciels déjà chargés dans la carte et pour laquelle il est nécessaire d'avoir les fichiers ExportFile correspondants au moment de la conversion). Le chargement du logiciel dans la carte à puce doit également être effectué d'un bloc. Toute interruption de transmission impose la reprise du chargement depuis l'origine. Enfin, deux autres inconvénients à cette solution sont l'utilisation d'un format de fichier pour la distribution des classes qui est particulier à la carte (le .cap) et la nécessaire normalisation des

structures de données internes de la machine virtuelle qui sont décrits dans le nouveau fichier .cap ainsi défini.

Une seconde solution consiste à compiler les fichiers .class lors de leur installation dans un langage natif de la carte à puce. La mémoire de travail de la carte à puce n'est alors pas occupée par une machine virtuelle et le temps d'exécution est notablement réduit. Le processus de chargement et d'édition de liens est dès lors réalisé. Le code natif ainsi généré présente des inconvénients puisqu'il occupe une place mémoire très supérieure à celle du code interprété. De plus, la compilation est complexe et coûteuse puisqu'elle doit être spécifique à la carte à puce. En outre, la sécurité de la carte à puce est amoindrie car le logiciel compilé en langage natif peut fournir des informations sur le fonctionnement de la carte.

L'invention vise à résoudre un ou plusieurs de ces inconvénients. L'invention a ainsi pour objet un procédé de chargement d'un logiciel compilé en langage intermédiaire orienté objet et comprenant plusieurs modules à charger dans la mémoire non volatile d'un appareil numérique portatif muni d'une machine virtuelle d'exécution, ces modules devant être chargés sous forme exécutable par l'interpréteur de ladite machine virtuelle, le chargement d'au moins un des modules comprenant les étapes suivantes :

-tester l'existence d'une structure de données associée à ce module dans la mémoire non volatile ; et

-si cette structure de données n'existe pas, créer une structure de données associée à ce module dans la mémoire non volatile et marquer cette structure de données comme non chargée ;

-résoudre les liens pour l'ensemble des éléments internes à ce module dans cette structure de données puis marquer cette structure de données comme chargée ;

-pour chaque autre module référencé dans le module en cours de chargement, tester l'existence d'une structure de données associée à cet autre module ;

-si une structure de données associée à cet autre module n'existait pas, créer une structure de données associée à cet autre module dans la mémoire non volatile et marquer cette structure de données comme non chargée.

Selon une variante, l'étape de création de la structure de données associée au module à charger comprend la mémorisation dans cette structure de données d'un descriptif du module à charger contenu dans ce module à charger.

Selon encore une variante, l'étape de résolution des liens comprend :

-la vérification de l'existence d'une structure de données associée à chaque élément interne ;

-si cette structure de données n'existe pas, la création d'une structure de données associée à cet élément interne dans la mémoire non volatile ;

-pour chaque élément externe mentionné dans le module en cours de chargement :

-la vérification de l'existence d'une structure de données associée à cet élément externe;

-si cette structure de données n'existe pas, la création d'une structure de données associée à cet élément externe dans la mémoire non volatile.

Selon une autre variante, le chargement dudit module comprend en outre, après la résolution de liens, la marquage de ce module comme lié si tous les modules externes qu'il mentionne sont marqués comme chargés.

Un test d'existence d'une structure de données peut comprendre un test de conformité entre cette structure de données si elle existe et sa référence dans le module en cours de chargement et la mémorisation d'informations de vérification de conformité ; les informations de vérification de conformité des références sont supprimées lorsque le module est marqué comme lié.

Un tel procédé peut présenter les étapes suivantes :

-les données statiques du module marqué comme lié sont initialisées ;

-ce module est marqué comme prêt à l'emploi.

Un code mémorisé dans l'appareil numérique et destiné à l'initialisation des données statiques peut alors être effacé lorsque le module est marqué comme prêt à l'emploi.

Selon une variante, les modules à charger sont des fichiers .class compilés en Java. Les structures de données sont des structures de données manipulables par la machine virtuelle. L'appareil numérique est une

carte à puce embarquant une machine virtuelle du type Java ou DotNet.

On peut prévoir qu'un tel chargement de module est effectué pour l'ensemble des modules du logiciel.

Suite à l'indisponibilité d'un module, la disponibilité de ce module peut être testée pour que ce module soit automatiquement chargé par l'appareil portatif.

L'invention a aussi pour objet un chargeur d'un logiciel compilé en langage intermédiaire orienté objet comprenant plusieurs modules à charger dans un appareil numérique portatif, ce chargeur étant susceptible d'être stocké dans une mémoire non volatile de l'appareil, susceptible d'accéder aux modules mémorisés sur un appareil source et susceptible de mettre en œuvre un procédé mentionné ci-dessus.

L'invention porte encore sur un logiciel chargé dans un appareil portatif selon un tel procédé.

L'invention porte par ailleurs sur un appareil numérique portatif comprenant une mémoire non volatile mémorisant le chargeur mentionné. Selon une variante, sa mémoire non volatile mémorise le logiciel mentionné.

Selon encore une variante, cet appareil est une carte à puce.

D'autres particularités et avantages de l'invention apparaîtront clairement à la lecture de la description faite à titre d'exemple non limitatif et en regard des dessins annexés sur lesquels :

-la figure 1 illustre un exemple de système informatique mis en œuvre pour réaliser un chargement dans une carte à puce ;

-la figure 2 illustre des données stockées dans la mémoire non volatile de la carte à puce durant le chargement ;

-la figure 3 illustre l'évolution de l'état d'un marqueur d'une structure de données en fonction d'étape réalisées lors du chargement.

Un appareil électronique portatif désignera par la suite un appareil portatif tel qu'un ordinateur portable mais aussi tout appareil électronique doté d'un microcontrôleur réunissant un processeur et des mémoires.

L'invention propose de charger les modules du logiciel dans un appareil portatif en créant une structure de données pour des éléments mentionnés pour la première fois durant ce chargement. Ainsi, une structure de données destinée à être exécutée par la machine virtuelle de l'appareil portatif est créée directement pour les éléments externes au module en cours de chargement, sans en stocker une représentation temporaire. Les éléments inutiles à l'exécution sont effacés de la mémoire non volatile de l'appareil non portatif au fur et à mesure des étapes de chargement, édition de lien et optimisation réalisés par un chargeur de modules logiciels.

La figure 1 illustre un exemple de moyens mis en œuvre pour charger un logiciel compilé dans une carte à puce 5 qui constitue en l'occurrence l'appareil

portatif. Un ordinateur 1 est muni d'un lecteur de carte à puce 2, dans lequel est insérée la carte à puce 5. L'ordinateur 1 est par exemple connecté à un serveur 3 par l'intermédiaire d'un réseau informatique 4. Des modules d'un logiciel compilé sont laissés en téléchargement sur le serveur 3 (par exemple sous la forme d'un ensemble de fichiers .Class pour un logiciel compilé en Java). La carte à puce 5 est munie d'une machine virtuelle pour l'exécution des logiciels chargés.

Par la suite, on détaillera plus particulièrement l'invention dans le cas d'un logiciel compilé en Java, bien que l'invention s'applique également pour des logiciels compilés dans d'autres langages intermédiaires orientés objet, tels que .Net (marque déposée).

La figure 2 illustre les données mémorisées dans la mémoire 6 de la carte à puce 5 durant le chargement des modules 7 par un chargeur 8. Le chargeur 8 est une application qui est par exemple mémorisée dans la mémoire morte de la carte à puce 5. Le chargeur 8 accède à un premier module à charger nommé Classe1. Le chargeur 8 place en mémoire de travail 61 tout ou partie de Classe1.

Le chargeur teste au préalable si le fichier .class est disponible (le fichier .class d'une classe à charger dynamiquement peut par exemple être associé à une adresse URL). Le chargeur teste également l'existence d'une structure de données 61 associée à Classe1 dans la mémoire non volatile. En effet, selon

le procédé de chargement de l'invention, cette structure de données a pu être créée auparavant lors du chargement d'un autre module. Si la structure de données 62 associée à Classe1 n'existe pas, cette structure de données est créée dans la mémoire non volatile de la carte 5. La structure 62 comprend un marqueur 621 d'état de chargement. Le marqueur 621 est placé à l'état « non chargé » lors de la création de la structure de données. Différents états du marqueur ainsi que des étapes les séparant sont illustrés à la figure 3.

Lorsque le chargeur 8 constate que la structure de données 62 est existante et à l'état « non chargé », le chargeur résout les liens pour l'ensemble des éléments internes pour lesquels des référence sont trouvées dans Classe1. Le chargeur élimine alors des paramètres inutiles à l'exécution du logiciel ou du module Classe1 (par exemple la table de numérotation de lignes, le fichier source, les informations de débuggage...). L'espace occupé par les données mémorisées dans la mémoire non volatile de la carte est ainsi significativement réduit.

On peut envisager de ne créer qu'une structure de données par module, et d'inclure l'ensemble des données des éléments internes de ce module dans sa structure de données associée. Avantageusement, on crée une structure de données dans la mémoire non volatile pour chaque élément référencé dans un module à charger. Les structures de données sont ainsi plus faciles et rapides à manipuler, notamment pour permettre

l'effacement d'un élément référencé devenu inutile suite à une édition de lien ou une optimisation.

La résolution des liens internes implique dans l'exemple la création des structures de données 63, 64 et 65 manipulables par la machine virtuelle. Ces structures de données sont soit créées durant le chargement antérieur d'un autre module (lors d'une résolution de liens externes détaillée ultérieurement), soit par le chargeur 8 ayant déterminé leur non existence durant cette résolution de liens internes. Les structures de données 63 et 64 sont des tables listant respectivement les champs internes et les méthodes dont les références apparaissent dans Classe1. La structure de données 65 correspond au descriptif d'un champ listé dans la structure de données 63. Ainsi, on peut associer une structure de données à chaque élément interne ou à chaque liste d'éléments internes.

Les structures de données sont complétées au fur et à mesure que des éléments les concernant sont effectivement chargés. Un tel chargement permet ainsi de ne mémoriser dans ces structures de données que les éléments utiles à l'exécution, en supprimant automatiquement les redondances des éléments référencés depuis plusieurs autres éléments.

Une fois cette résolution de liens effectuée, le marqueur 621 est placé à l'état « chargé ». Cette étape de résolution de liens est avantageusement utilisée pour réaliser un déboggage du module. Ainsi, cette étape d'édition de liens permettra aisément de

déterminer les liens internes qui peuvent se révéler défaillants.

Un certain nombre d'informations peuvent être supprimées suite au chargement, afin d'encombrer au minimum la mémoire non volatile de l'appareil. On constate notamment que les tables de références des fichiers .class contiennent de nombreuses entrées qui sont utilisées uniquement durant le processus de chargement des modules et non durant l'exécution. Ainsi des méta-informations utilisées durant l'édition de lien peuvent être supprimées. Certains descripteurs de méthodes peuvent ainsi être supprimés une fois que l'édition de liens est réalisée pour un fichier .class. Il est, par exemple, possible d'effacer la description textuelle des éléments internes privés au module une fois l'édition des liens internes réalisés : cette description servait uniquement à réaliser cette édition de liens et ne servira plus jamais.

En outre, il est possible d'effacer la description de certains éléments internes publics (destinés à être lus par d'autres modules) si on estime qu'aucun autre module ne pourra les référencer. C'est notamment le cas lorsqu'on interdit tout chargement ultérieur de modules afin de figer l'appareil portable. L'appareil peut notamment être figé en invalidant le chargeur 8 ou en effaçant le code du chargeur 8 si celui-ci était stocké en mémoire non volatile.

Suite à l'étape de résolution des liens internes, le chargeur 8 effectue une résolution des liens pour les éléments externes référencés dans le module Classe1

en cours de chargement. Ainsi, pour chaque module externe référencé dans Classe1, on teste l'existence d'une structure de données associée. Si tel n'est pas le cas, une structure de donnée associée est créée et son marqueur est placé à l'état « non chargé ». Dans l'exemple, Classe1 fait référence au module externe Classe2. Aucune structure de données n'existant pour Classe2 avant le chargement de Classe1, l'étape de résolution des liens externes pour Classe1 implique la création de la structure de données 66.

Si une structure de données doit être créée pour chaque élément d'un module, toute référence à un élément externe dans le module en cours de chargement sera résolue. De façon similaire, une structure de données est créée pour chaque élément externe si un test détermine que la structure de données associée n'existe pas encore.

Lors de la résolution de liens pour les éléments externes, le test d'existence comprend avantageusement un test de conformité de la structure de données trouvée avec sa référence dans le module en cours de chargement. Si ce test de conformité est négatif, un message d'échec de chargement est généré. Ce test de conformité fournit des moyens de déboggage supplémentaires.

Ainsi, le chargement d'un module permet de créer au plus tôt des structures de données pour d'autres modules à charger, dans la mémoire non volatile de l'appareil portatif. De plus, les marqueurs permettent de reprendre un chargement au niveau où ce chargement aurait pu être interrompu. Un tel chargement permet de

charger seulement certains modules, avant que des modules associés ne soient disponibles. L'invention permet donc de faire évoluer dynamiquement les structures de données des modules au fur et à mesure de leur chargement.

Lorsque les liens de l'ensemble des références externes du module à charger ont été résolues, le marqueur de sa structure de données est placé à l'état « lié ».

Ensuite, les données statiques de ce module lié sont initialisées. Le marqueur de sa structure de données est alors placé dans l'état «prêt ». Ce module est dès lors dans la mémoire non volatile de l'appareil portatif et prêt à l'emploi. Un code mémorisé dans l'appareil portatif et destiné à l'initialisation des données statiques est effacé lorsque la structure de données est à l'état « prêt » afin de réduire l'espace occupé en mémoire. Lorsque le marqueur de la structure de données est placé à l'état « prêt », les éléments du module placés en mémoire de travail peuvent être effacés pour libérer de la place dans l'appareil numérique.

L'homme du métier saura définir une séquence de chargement appropriée des modules successifs d'un logiciel compilé. Le chargeur 8 peut notamment mémoriser les modules non chargés pour lesquels des structures de données ont été créées, puis réaliser successivement leur chargement. Le chargeur 8 peut

également ne charger que les modules qui lui sont soumis.

Suite à une indisponibilité d'un module à charger, le chargeur peut également effectuer des tests de disponibilité répétés et charger automatiquement ce module dès que possible.

Du fait de l'espace libéré en mémoire non volatile, une machine virtuelle Java standard capable de charger des ficiers .class peut être mémorisée dans la carte à puce. Un développeur pourra alors diffuser un même logiciel pour les appareils portatifs et les ordinateurs de bureau. Le développeur disposera également des éléments de bibliothèque de son choix. Contrairement à la programmation en JavaCard, le développeur n'a pas à récupérer au préalable des fichiers du type Exportfile.

REVENDICATIONS

1. Procédé de chargement d'un logiciel compilé en langage intermédiaire orienté objet et comprenant plusieurs modules (7) à charger dans la mémoire non volatile d'un appareil numérique portatif (5) muni d'une machine virtuelle d'exécution, ces modules (7) devant être chargés sous forme exécutable par l'interpréteur de ladite machine virtuelle, caractérisé en ce que le chargement d'au moins un des modules comprend les étapes suivantes :

-tester l'existence d'une structure de données associée à ce module dans la mémoire non volatile (6) ; et

-si cette structure de données n'existe pas, créer une structure de données (62) associée à ce module dans la mémoire non volatile et marquer cette structure de données comme non chargée ;

-résoudre les liens pour l'ensemble des éléments internes à ce module dans cette structure de données puis marquer cette structure de données comme chargée (621) ;

-pour chaque autre module référencé dans le module en cours de chargement, tester l'existence d'une structure de données associée à cet autre module ;

-si une structure de données associée à cet autre module n'existait pas, créer une structure de données (66) associée à cet autre module dans la

mémoire non volatile et marquer cette structure de données comme non chargée.

2. Procédé selon la revendication 1, caractérisé en ce que l'étape de création de la structure de données associée au module à charger comprend la mémorisation dans cette structure de données d'un descriptif du module à charger contenu dans ce module à charger.

3. Procédé selon la revendication 1 ou 2, caractérisé en ce que :

-l'étape de résolution des liens comprend :

-la vérification de l'existence d'une structure de données associée à chaque élément interne ;

-si cette structure de données n'existe pas, la création d'une structure de données (64, 65) associée à cet élément interne dans la mémoire non volatile ;

-pour chaque élément externe mentionné dans le module en cours de chargement :

-la vérification de l'existence d'une structure de données associée à cet élément externe;

-si cette structure de données n'existe pas, la création d'une structure de données associée à cet élément externe dans la mémoire non volatile.

4. Procédé selon l'une quelconque des revendications précédentes, caractérisé en ce que le chargement

dudit module comprend en outre, après la résolution de liens, la marquage de ce module comme lié si tous les modules externes qu'il mentionne sont marqués comme chargés.

5. Procédé selon la revendication 4, caractérisé en ce que :
-un test d'existence d'une structure de données comprend un test de conformité entre cette structure de données si elle existe et sa référence dans le module en cours de chargement et la mémorisation d'informations de vérification de conformité ;
-les informations de vérification de conformité des références sont supprimées lorsque le module est marqué comme lié.

6. Procédé selon la revendication 4 ou 5, caractérisé en ce qu'il comprend en outre les étapes suivantes :
-les données statiques du module marqué comme lié sont initialisées ;
-ce module est marqué comme prêt à l'emploi.

7. Procédé selon la revendication 6, caractérisé en ce qu'un code mémorisé dans l'appareil numérique et destiné à l'initialisation des données statiques est effacé lorsque le module est marqué comme prêt à l'emploi.

8. Procédé selon l'une quelconque des revendications précédentes, caractérisé en ce que les modules à charger sont des fichiers .class compilés en Java.

9. Procédé selon la revendication 8, caractérisé en ce que les structures de données sont des structures de données manipulables par la machine virtuelle.

10. Procédé selon la revendication 8 ou 9, caractérisé en ce que l'appareil numérique (5) est une carte à puce embarquant une machine virtuelle du type Java ou DotNet.

11. Procédé selon l'une quelconque des revendications précédentes, caractérisé en ce que ledit chargement de module est effectué pour l'ensemble des modules du logiciel.

12. Procédé selon l'une quelconque des revendications précédentes, caractérisé en ce que, suite à l'indisponibilité d'un module, la disponibilité de ce module est testée et ce module est automatiquement chargé par l'appareil portatif.

13. Chargeur (8) d'un logiciel compilé en langage intermédiaire orienté objet comprenant plusieurs modules à charger dans un appareil numérique portatif, caractérisé en ce qu'il est susceptible d'être stocké dans une mémoire non volatile de l'appareil, susceptible d'accéder aux modules mémorisés sur un appareil source et susceptible de mettre en œuvre le procédé selon l'une quelconque des revendications précédentes.

14. Logiciel chargé dans un appareil portatif par le procédé selon l'une quelconque des revendications 1 à 12.

15. Appareil numérique portatif comprenant une mémoire non volatile mémorisant le chargeur selon la revendication 13.

16. Appareil selon la revendication 15, caractérisé en ce que la mémoire non volatile mémorise un logiciel selon la revendication 14.

17. Appareil selon la revendication 15 ou 16, caractérisé en ce que cet appareil est une carte à puce.

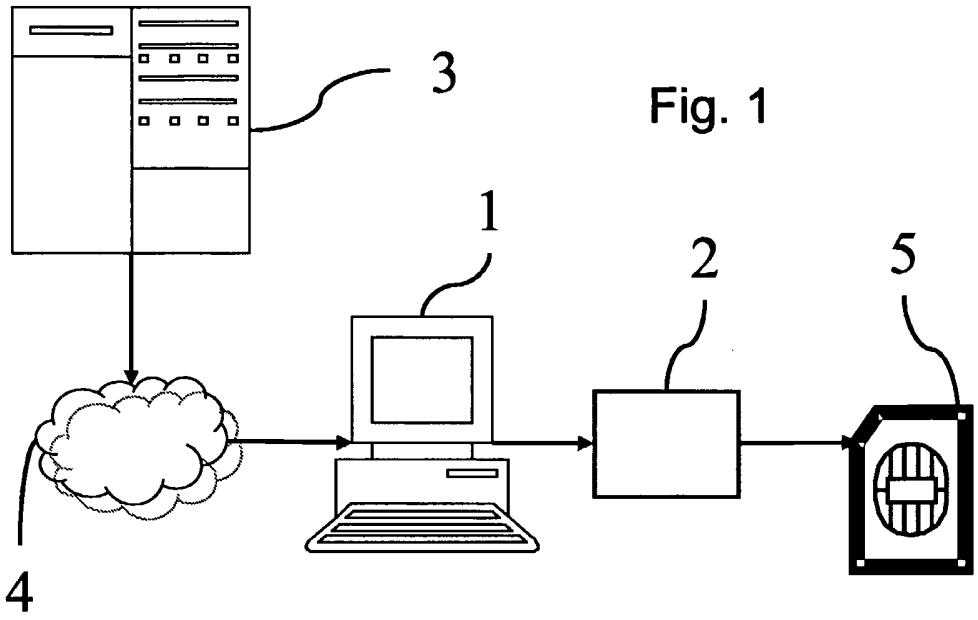

3
1
2
5
4

Fig. 1

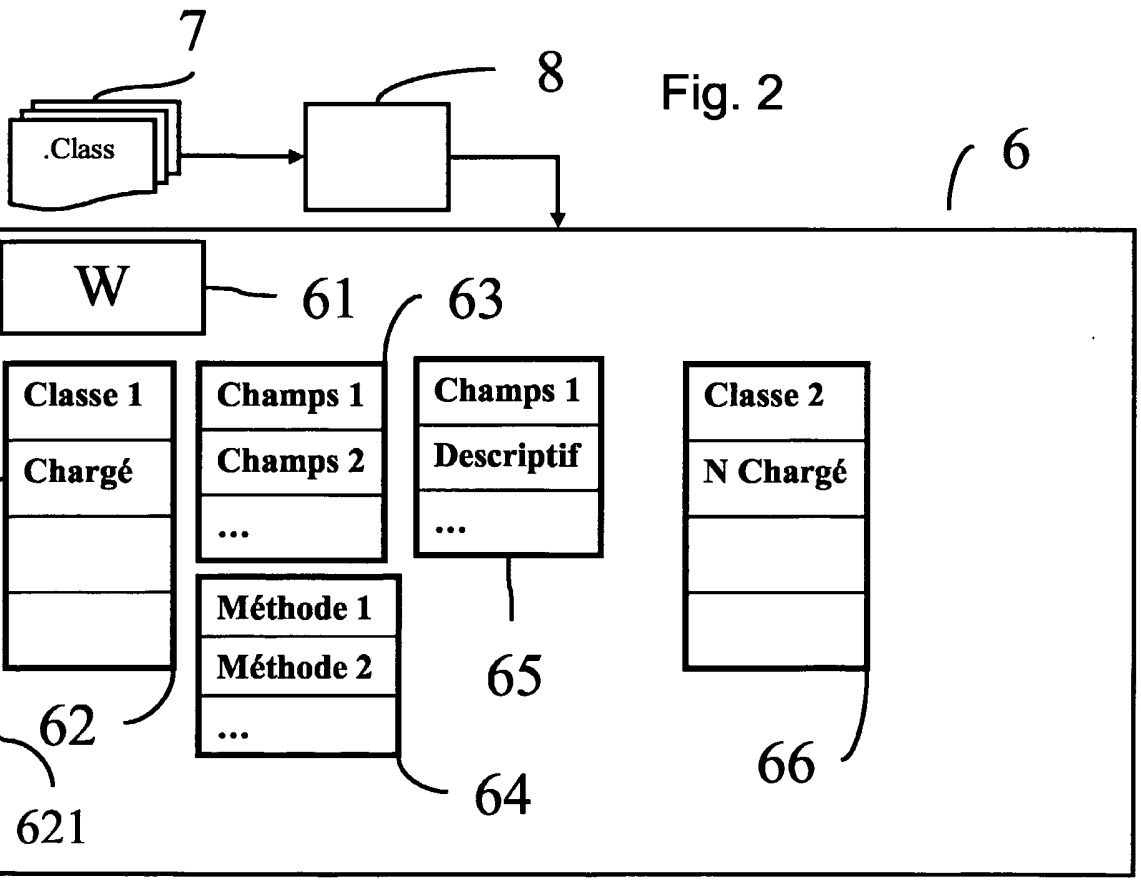
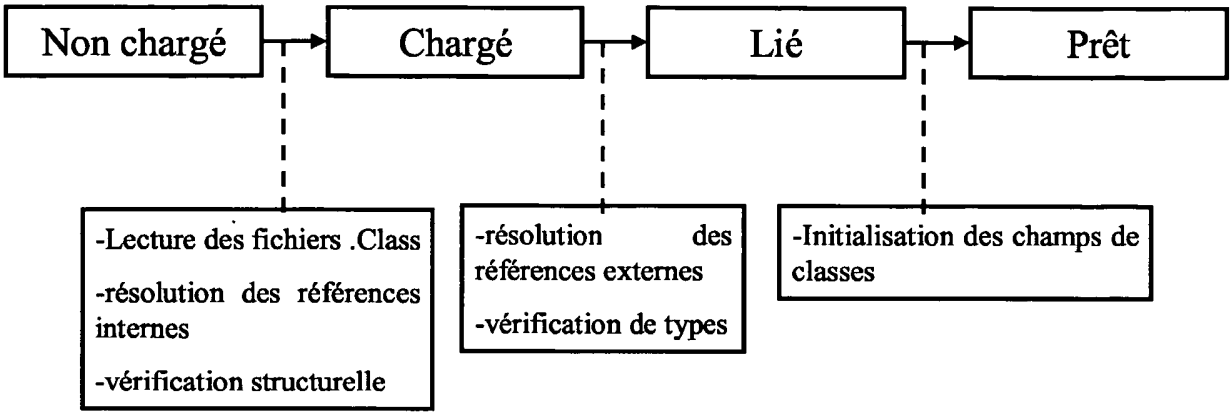

7
8
.Class
6
W
61
63
Classe 1
Chargé
Champs 1
Champs 2
...
Champs 1
Descriptif
...
Classe 2
N Chargé
Méthode 1
Méthode 2
...
62
65
66
64
621

Fig. 2

Fig. 3


Non chargé
Chargé
Lié
Prêt
-Lecture des fichiers .Class
-résolution des références internes
-vérification structurelle
-résolution des références externes
-vérification de types
-Initialisation des champs de classes