(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
11 janvier 2001 (11.01.2001)

PCT

(10) Numéro de publication internationale
WO 01/02955 A1

(51) Classification internationale des brevets[7]: G06F 9/45

(21) Numéro de la demande internationale:
PCT/FR00/01815

(22) Date de dépôt international: 28 juin 2000 (28.06.2000)

(25) Langue de dépôt: français

(26) Langue de publication: français

(30) Données relatives à la priorité:
99/08460 1 juillet 1999 (01.07.1999) FR

(71) Déposants *(pour tous les États désignés sauf US)*: **BULL CP8** [FR/FR]; 68, route de Versailles, f-78430 Louveciennes (FR). **INSTITUT NATIONAL DE LA RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE** [FR/FR]; Domaine de Voluceau, Rocquencourt, F-78153 Le Chesnay Cedex (FR). **CENTRE NATIONAL DE LA RECHERCHE SCIENTIQUE (CNR S)** [FR/FR]; 3, rue Michel-Ange, F-75794 Paris Cedex 16 (FR).

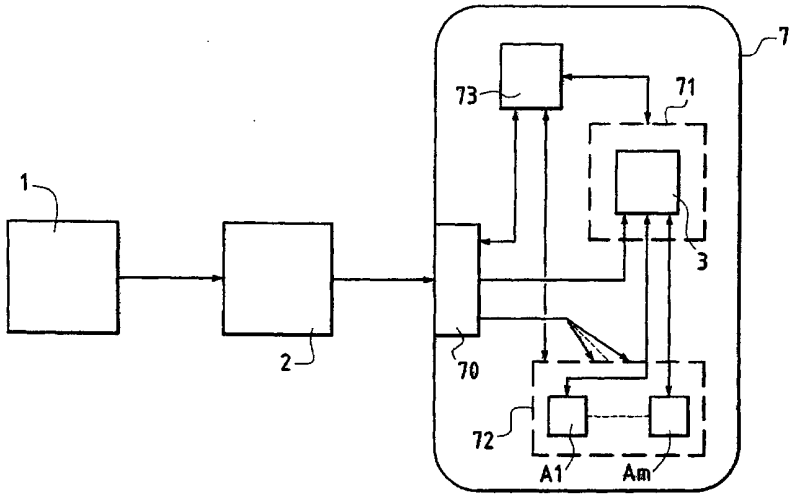
(72) Inventeurs; et

(75) Inventeurs/Déposants *(pour US seulement)*: **GOIRE, Christian** [FR/FR]; 8, allée du Mail, F-78340 Les Clayes-sous-Bois (FR). **JENSEN, Thomas** [FR/FR]; IRISA/INRIA, Campus Universitaire de Beaulieu, F-35042 Rennes Cedex (FR). **FRADET, Pascal** [FR/FR]; IRISA/INRIA, Campus Universitaire de Beaulieu, F-35042 Rennes Cedex (FR). **LE METAYER, Daniel** [FR/FR]; IRISA/INRIA, Campus Universitaire de Beaulieu, F-35042

[Suite sur la page suivante]

(54) Title: METHOD FOR VERIFYING CODE TRANSFORMERS FOR AN INCORPORATED SYSTEM, IN PARTICULAR IN A CHIP CARD

(54) Titre: PROCEDE DE VERIFICATION DE TRANSFORMATEURS DE CODES POUR UN SYSTEME EMBARQUE, NOTAMMENT SUR UNE CARTE A PUCE



(57) Abstract: The invention concerns a method for verifying a transformer of source code into a transformed code designed for an incorporated system (7). The method comprises at least steps which consist in: determining a single virtual machine factoring the behaviour of said two codes (1, 3); determining for each of said source (1) and transformed (3) codes a plurality of functions called auxiliary functions representing residual differences between said source (1) and transformed (3) codes; and a step which consists in verifying a property of correspondence between the auxiliary functions, the code transformer (2) verification resulting from the latter step. The invention is particularly applicable to chip cards (7).

[Suite sur la page suivante]

WO 01/02955 A1

Rennes Cedex (FR). **DENNEY, Ewen** [FR/FR]; IRISA/INRIA, Campus Universitaire de Beaulieu, F-35042 Rennes Cedex (FR).

(74) **Mandataire: ROGER PETIT, Georges**; Office Bletry, 94, rue Saint Lazare, F-75010 Paris (FR).

(81) **États désignés** *(national)*: BR, CA, CN, JP, US.

(84) **États désignés** *(régional)*: brevet européen (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Publiée:

— *Avec rapport de recherche internationale.*

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 concerne un procédé de vérification d'un transformateur de code source en un code transformé destiné à un système embarqué (7). Le procédé comprend au moins les étapes de détermination d'une machine virtuelle unique factorisant le comportement de ces deux codes (1, 3), la détermination, pour chacun desdits codes source (1) et transformé (3), d'une pluralité de fonctions dites auxiliaires représentant des différences résiduelles entre lesdits codes source (1) et transformé (3) et une étape consistant à vérifier une propriété de correspondance entre les fonctions auxiliaires, la vérification du transformateur de code (2) étant obtenue à partir de cette dernière étape. Application notamment aux cartes à puce (7).

PROCÉDÉ DE VÉRIFICATION DE TRANSFORMATEURS DE CODES POUR UN SYSTÈME EMBARQUÉ, NOTAMMENT SUR UNE CARTE À PUCE

L'invention concerne un procédé de vérification de transformateurs de codes pour un système embarqué.

L'invention concerne aussi l'application d'un tel procédé à un transformateur pour générer un code destiné à une carte à puce.

Dans le cadre de l'invention, le terme "système embarqué" doit être considéré dans son acception la plus générale. Il concerne notamment des systèmes destinés à une carte à puce, ce qui constitue l'application préférée de l'invention, mais également tout système destiné à un dispositif portable ou mobile comportant des moyens propres de traitement de données informatisés, que l'on appellera ci-après "ressources de traitement".

Les systèmes embarqués modernes sont munis de ressources de traitement des données permettent de remplir des fonctions de plus en plus complexes et de plus en plus nombreuses. Cependant, malgré la mise sur le marché de technologies et de composants de plus en plus performants, une caractéristique distinctive des systèmes embarqués, par rapport à des systèmes informatiques conventionnels (micro-ordinateur, station de travail, etc.), concerne les limitations qu'ils imposent en matière de ressources (taille mémoire et puissance des microprocesseurs notamment). Pour satisfaire ces contraintes, il est nécessaire de transformer le code destiné à être exécuté sur un système embarqué. Les transformations ont pour but de produire un code plus efficace et plus économe en ressources.

Pour fixer les idées, et à titre d'exemple non limitatif de code, on considérera dans ce qui suit un programme écrit dans la machine virtuelle du langage "JAVA" (marque déposée par SUN MICROSYSTEMS) qui présente l'intérêt de pouvoir être exécuté dans de nombreux environnements. Les domaines d'application de ce langage se sont en effet multipliés notamment avec le développement important du réseau Internet. De nombreuses applications logicielles de petite taille, dites "applets", sont écrites dans ce langage et exécutables par un navigateur de type "WEB"

On se placera également dans le cadre de l'application préférée de l'invention, à savoir l'exécution d'un code de ce type par les ressources informatiques propres d'une carte à puce. Comme il a été rappelé ci-dessus, malgré des progrès technologiques importants, la capacité mémoire de la carte à puce, ainsi que la puissance du microprocesseur qui l'équipe restent relativement limités. Il est également important que le code soit résident dans la carte à puce, car les transmissions entre celle-ci et un terminal hôte, quel qu'il soit, s'effectuent à basse vitesse. Les standards actuels ne prévoient que des transmissions de type série. Le besoin se fait donc sentir de disposer d'un code que l'on pourrait qualifier "d'allégé", en tout cas optimisé pour cet usage. Pour ce faire, il a été proposé d'utiliser un langage dérivé de "JAVA", se présentant sous la forme d'une restriction de ce langage, à savoir le langage "JAVA CARD" " (marque déposée également par SUN MICROSYSTEMS).

Une complication supplémentaire vient du fait que les systèmes embarqués sont généralement utilisés dans des environnements qui requièrent les plus hautes garanties en matière, à la fois de fiabilité et de sécurité. On peut citer à titre d'exemple les nouvelles versions de cartes à puce sur lesquelles on souhaite installer des applications logicielles multiples qui doivent coopérer harmonieusement sans révéler d'informations confidentielles. En effet, *a priori*, ces multiples applications peuvent concerner des utilisateurs distincts. Malgré la coopération précitée, on doit conserver un cloisonnement rigoureux, pour que les informations concernant un utilisateur donné restent confidentielles, pour le moins ne puissent pas être mises à la disposition d'un utilisateur non habilité à les connaître (lecture) et/ou à les manipuler (écriture et opérations connexes : effacement, modification). En dehors de l'aspect "confidentialité", d'autres exigences sont à prendre en compte, notamment l'exigence dite "d'intégrité" : pertes de données, modifications non conformes, etc.

Si on considère le code "source", dans le sens de "code initial", tel que par exemple le "byte code" du langage "JAVA" précité, ce dernier présente toutes les garanties nécessaires et remplit les exigences précitées, le "byte code" étant un programme écrit dans la machine virtuelle du langage "JAVA".

En effet, de nombreux tests ont pu être effectués, ce pendant de longues périodes de temps.

Le code dit "transformé" est obtenu à partir du "code source" à l'aide d'un transformateur de code, généralement extérieur au système embarqué, mais qui peut également être résident dans celui-ci. Il est donc nécessaire de montrer l'équivalence entre le code source et le code transformé.

Cela peut s'effectuer en garantissant que les transformations effectuées sur le code ne changent en rien son comportement (d'un point de vue externe) et n'introduisent pas de failles de sécurité. En d'autres termes, le code initial (avant transformation) doit être d'un point de vue logique équivalent au code résultant (après transformation).

Il est particulièrement difficile de garantir cette propriété en général, car les transformations ont un effet global sur le code et sur les représentations des données qu'il manipule. De façon pratique, la complexité impliquée par cette opération ne permet pas une mise en œuvre dans des conditions économiques et/ou technologiques réalistes. En outre, on doit bien comprendre que de tels besoins ne sont apparus que très récemment, notamment en conjonction avec le développement des technologies précitées de cartes à puce multi-applications et/ou multi-utilisateurs.

L'invention vise à répondre à ces besoins, sans nécessiter des procédures extrêmement longues et coûteuses.

Le procédé selon l'invention permet de vérifier de manière systématique et modulaire la correction des transformations de codes.

Dans le cadre de l'invention, deux formalismes, bien connus en soi, seront utilisés de façon essentielle : les sémantiques opérationnelles et les relations logiques. Pour une description plus détaillée de ces formalismes, on se reportera avec profit, pour le premier, au livre de : H. R. Nielson, et F. Nielson, intitulé : "*Semantics with Applications: A formal Introduction*". Wiley, 1992, et, pour le second, au livre de J. Mitchell : *"Foundations for Programming Languages*", MIT Press, 1996.

Selon une caractéristique essentielle de l'invention, le procédé de vérification des transformateurs de codes consiste à spécifier le sens de deux

codes à l'aide d'une machine virtuelle commune paramétrée par des fonctions que l'on appellera "fonctions auxiliaires". Les différences entre les deux codes sont exprimées et regroupées dans les fonctions auxiliaires précitées. Chaque fonction auxiliaire possède deux versions : une version dans le code source et une version dans le code transformé. Les premiers modules étant identiques, puisque communs aux deux codes, il n'y a pas lieu de vérifier s'ils sont équivalents. Pour montrer l'équivalence des deux codes, il suffit donc de montrer que les fonctions dites auxiliaires, considérées deux à deux, sont équivalentes. Ces deux sous-ensembles peuvent être rendus beaucoup moins complexes que les deux ensembles représentés par les deux codes, source et transformé, considérés dans leur intégralité. Il s'ensuit que, selon le procédé de l'invention, la difficulté inhérente au processus de vérification est très fortement réduite, et de façon corrélative, le processus de vérification devient économiquement et technologiquement réalisable.

L'invention a donc pour objet un procédé de vérification d'un transformateur de code dit source en un code dit transformé destiné à un système embarqué, lesdits codes source et transformé étant associés à des machines virtuelles, caractérisé en ce qu'il comprend au moins les étapes suivantes :

- la détermination, pour chacun desdits codes source et transformé, d'un premier sous-ensemble commun, constituant une machine virtuelle unique factorisant le comportement de ces deux codes ;
- la détermination, pour chacun desdits codes source et transformé, d'un second sous-ensemble constitué d'une pluralité de fonctions dites auxiliaires, lesdites fonctions auxiliaires représentant des différences résiduelles entre lesdits codes source et transformé ;
- l'association par paire desdites fonctions auxiliaires, une première fonction auxiliaire de chaque paire appartenant audit second sous-ensemble associé audit code source et une seconde fonction auxiliaire de chaque paire appartenant audit second sous-ensemble associé audit code transformé ;

- la vérification d'une propriété de correspondance déterminée entre lesdites fonctions auxiliaires de toutes lesdites paires ; et
- la vérification que ladite transformation du code source en code transformé par ledit convertisseur respecte ladite propriété de correspondance déterminée.

L'invention a encore pour objet l'application d'un tel procédé à un transformateur pour la génération d'un code destiné à être enregistré dans une carte à puce.

L'invention va maintenant être décrite de façon plus détaillée en se référant aux dessins annexés, parmi lesquels :

- la figure 1 illustre schématiquement le processus de transformation d'un code source en un code transformé final ;
- les figures 2A et 2B illustrent schématiquement une des caractéristiques essentielles du procédé selon l'invention ; et
- la figure 3 illustre schématiquement l'application du procédé selon l'invention à une carte à puce ;

On va maintenant décrire de façon détaillée le procédé de vérification de transformateurs de codes selon l'invention.

La figure 1 illustre schématiquement le processus de transformation d'un code 1, que l'on appellera "code source", dans le sens de code origine ou initial, en un code final 3, dit "code transformé", à l'aide d'un transformateur de code 2. Ce dernier organe peut être un moyen informatique ou une pièce de logiciel spécifique. De façon habituelle, le code transformé est destiné à être résident dans le système embarqué 4 (en trait plein). Le transformateur 2 peut également être résident ou téléchargé dans le système embarqué : référence 4' (en traits pointillés).

Après chargement ou enregistrement dans le système embarqué 4-4', le code transformé 3 permet l'exécution d'une ou plusieurs tâches en tant que de besoin, représentées sous la référence unique 5. On suppose que le système embarqué 4 dispose de ressources informatiques autonomes classiques (non représentées).

A priori, la transformation de code est effectuée une fois pour toutes par un transformateur donné 2 ou dans de rares occasions : modification de version du code d'origine ou code source 1, par exemple.

Il est donc nécessaire que l'on puisse établir la preuve formelle que le code transformé 3 est équivalent au code source 1. Ce processus permet de vérifier si le transformateur 2 fonctionne de façon correcte.

Cependant, comme il a été rappelé, si on considère, dans leur globalité, les deux ensembles formés par les codes, source et transformé, la théorie montre qu'une telle détermination n'est généralement pas possible de façon réaliste.

Une caractéristique essentielle du procédé selon l'invention va consister à trouver pour chacun des deux codes deux sous-ensembles, que l'on appellera premier et second sous-ensembles. Selon une caractéristique importante du procédé selon l'invention, les premiers sous-ensembles forment une machine virtuelle commune aux deux codes, source et transformé. De ce fait, il n'est donc pas nécessaire de vérifier l'équivalence des premiers sous-ensembles.

Les seconds sous-ensembles, constitués des fonctions auxiliaires, sont par contre distincts d'un code à l'autre. La détermination de l'équivalence des codes, source et transformé, se réduit alors à la détermination de l'équivalence de toutes les paires de fonctions auxiliaires des seconds sous-ensembles. Or, la complexité résiduelle des fonctions auxiliaires peut être très réduite. Il s'ensuit que la détermination d'équivalence précitée devient possible.

Les figures 2A et 2B illustrent très schématiquement le procédé selon l'invention.

Comme le montre plus particulièrement la figure 2A, les premiers sous-ensembles du code source 1 et du code transformé 3 forment une machine virtuelle commune 13. Les seconds sous-ensembles, 10 et 30, sont constitués chacun par une série de fonctions dites auxiliaires, dont l'équivalence devra être vérifiée. Ces fonctions auxiliaires, 10 et 30, paramètrent la machine virtuelle commune 13.

L'équivalence des deux codes, source 1 et transformé 2, se réduit donc à vérifier l'équivalence des fonctions auxiliaires, 10 et 30, prises deux à deux, comme il le sera montré ci-après en regard de la figure 2B.

Les étapes du procédé vont maintenant être décrites de façon plus détaillée.

Les codes source et transformé sont associés à des première et seconde machines dites virtuelles, respectivement.

La première étape consiste en la définition d'une machine virtuelle (ou sémantique opérationnelle) unique permettant de factoriser le comportement du code source et du code transformé. Les différences entre les deux codes apparaissent alors à travers des fonctions auxiliaires qui seront interprétées ou mises en oeuvre différemment dans les deux codes.

Une machine virtuelle peut se représenter par un ensemble de règles de la forme :

$$\frac{\text{prémisse 1} \quad . \quad . \quad . \quad \text{prémisse n}}{\text{état1[Instruction1]} \Rightarrow \text{état2}} \qquad (1).$$

Les prémisses sont, soit des conditions d'application d'une règle, c'est-à-dire des expressions booléennes, soit des affectations à des variables utilisées pour exprimer un changement d'état. Les prémisses font appel à des fonctions auxiliaires pour extraire des informations de l'état ou exprimer des conditions. Chaque règle indique comment l'état de la machine évolue lorsque les prémisses sont vérifiées et l'instruction "Instruction1" est rencontrée. On définit une ou plusieurs règles de cette forme pour chaque type d'instruction du code.

La seconde étape consiste en la définition de types ou structures de données utilisés dans les deux codes. On définit des types basiques comme, par exemple :

Basique ::= Nat | Bool | Nom... (2),

et des types construits comme, par exemple :

Environnement ::= Nom → Valeur

Instructions :: = Instruction1 I Instruction2 I ... (3),

La troisième étape consiste en l'interprétation de types, référencés θ, utilisés dans les machines virtuelles. Pour chaque type θ, on définit une interprétation pour le code source $[\![\theta]\!]_S$ et une interprétation pour le code transformé, $[\![\theta]\!]_T$, plus une relation R_θ entre les deux interprétations $[\![.]\!]_S$ et $[\![.]\!]_T$. Ces relations, appelées relations logiques, respectent la structure des types. Pour des types simples, elles doivent être définies explicitement ; pour des types structurés, elles sont déduites des types des composants de la structure.

Par exemple, pour les paires :

$$(a, b)\ R_{\theta 1 \times \theta 2}\ (a', b') \Leftrightarrow a\ R_{\theta 1}\ a' \wedge b\ R_{\theta 2}\ b' \quad (4),$$

relation dans laquelle $\theta 1$ et $\theta 2$ sont des types et a, b, a' et b' des éléments de type.

Il en est de même pour les fonctions :

$$f\ R_{\theta 1 \to \theta 2}\ f' \Leftrightarrow \forall a, a' .\ a\ R_{\theta 1}\ a' \Rightarrow f\ a\ R_{\theta 2}\ f'\ a' \quad (5).$$

Les relations logiques doivent être la relation "identité" pour les types observables, c'est à dire des types pour lesquels on veut montrer que les deux

codes rendent le même résultat. Il s'agit habituellement de types imprimables et/ou affichables sur un écran informatique. Cela peut être des types basiques, mais également des types structurés représentant, par exemple, une pile ou des variables d'un programme donné.

La quatrième étape consiste en l'interprétation des fonctions auxiliaires utilisées dans les machines virtuelles. Pour chaque fonction auxiliaire *f*, on donne sa définition pour le code source, notée $[\![f]\!]_S$, et sa définition pour le code transformé, notée $[\![f]\!]_T$.

La détermination de l'équivalence consiste à montrer que les définitions des fonctions auxiliaires respectent les relations logiques. Plus précisément, pour chaque fonction auxiliaire $f : \theta \rightarrow \theta'$, on montre

$$[\![f]\!]_S \; R_{\theta \rightarrow \theta'} \; [\![f]\!]_T \qquad (6).$$

Il s'ensuit que les deux machines virtuelles sont en relation, c'est-à-dire que :

$$[\![état]\!]_S \; R_{type\text{-}état} \; [\![état]\!]_T \qquad (7).$$

Comme les relations sont l'identité pour les types observables, les codes source et transformé sont observationellement identiques.

La dernière étape consiste à montrer qu'il existe un transformateur Γ (figure 1 : 2) qui satisfait les relations logiques. Cela peut être fait en vérifiant qu'un transformateur donné $\Gamma : S \rightarrow T$ satisfait la relation logique associée au type de son argument, avec S code source (figure 1 : 1) et T code transformé (figure 1 : 3). Pour ce faire il est nécessaire qu'il obéisse à la relation suivante :

$$\forall x \; [\![\theta]\!]_S \; . \; x \; R_\theta \; \Gamma(x) \qquad (8).$$

Il vient d'être montré que les relations logiques spécifient un ensemble de contraintes. On peut donc en extraire un transformateur 2 correct par

construction, en appliquant des techniques de raffinement ou d'extraction, en faisant appel à un des assistants de preuve appropriés..

Le procédé selon l'invention présente donc un avantage important car il permet une grande mécanisation du processus de vérification, et surtout permet de le conduire à bien, car cette vérification est menée sur des sous-ensembles moins complexes.

Si la transformation du code source 1 peut se décrire comme une succession de transformations plus simples, cette méthode peut s'appliquer pour montrer chaque transformation indépendamment. Il s'ensuit qu'elle présente un grand avantage de modularité.

La vérification ne doit être effectuée que sur les sous-ensembles de fonctions auxiliaires 10 et 30, comme illustré par la figure 2B, à l'aide d'un organe 6, matériel ou logiciel. On a supposé qu'il existe *n* fonctions auxiliaires, référencées 10_a, 10_b, ..., 10_i, ..., 10_{n-1}, 10_n et 20_a, 20_b, ..., 20_i, ...20_{n-1}, 20_n, respectivement. Si l'organe 6 est matériel, il comporte autant de circuits vérificateurs, 60_a, 60_b, ..., 60_i, ..., 60_{n-1}, 60_n (représentés arbitrairement sur la figure 2B par le symbole d'un comparateur), que de paires de fonctions auxiliaires à vérifier, par exemple le circuit vérificateur 60_i pour la paire de fonctions 10_i et 30_i. La ou les sortie(s) de cet organe 6, sous la référence unique 61, indique(nt) que la relation logique entre toutes les paires possibles de fonctions auxiliaires correspondantes des codes source 1 et transformé 3 est satisfaite. Cette série d'opérations est suffisante pour apporter la preuve formelle de l'équivalence des deux codes, dans leur globalité.

Il est à noter que le procédé selon l'invention est utilisable aussi bien, *a posteriori*, c'est-à-dire pour vérifier un transformateur existant, qu'*a priori,* comme une aide au développement d'un nouveau transformateur. Elle permet notamment, dans ce dernier cas, d'en déterminer les caractéristiques, pour qu'il fonctionne correctement, en d'autres termes pour que le code transformé qui sera généré par ce transformateur à partir du code source satisfasse l'exigence d'équivalence précitée.

On va maintenant se placer dans le cadre des cartes à puce. La figure 3 illustre schématiquement l'architecture d'une carte à puce, référencée

7. On n'a représenté sur cette figure que les éléments essentiels à la bonne compréhension du procédé selon l'invention.

La carte à puce 7 comprend notamment un organe d'entrée-sortie 70 permettant des communications avec le monde extérieur, un premier organe de mémoire 71, fixe ou programmable (de type "ROM", "PROM", "EPROM" ou "EEPROM"), et un organe de mémoire vive 72. La carte à puce 7 comprend enfin un microprocesseur ou un microcontrôleur 73 dialoguant par l'intermédiaire de bus avec les autres composants de la carte à puce 7.

L'architecture logicielle d'une telle carte à puce 7 obéit à la norme ISO 7816-3, se traduisant par une couche protocolaire allant des couches les plus basses associées aux organes d'entrée-sortie 70, jusqu'aux couches les plus hautes associées aux applications logicielles enregistrées dans la carte à puce 7. Ces normes prévoient que les transmissions s'effectuent en mode série.

Le code source 1, une fois transformé par le transformateur de code 2, est transmis à la carte à puce 7 pour y être enregistré, généralement dans l'organe de mémoire 71, fixe ou "semi-fixe", via l'organe d'entrée-sortie 70. La ou les applications logicielles traitées par la carte à puce 7 peuvent être enregistrées à demeure dans la carte à puce 7, c'est-à-dire dans l'organe de mémoire 71, ou de façon transitoire dans la mémoire vive 72. Dans ce dernier cas les applications sont téléchargées via l'organe d'entrée-sortie 70. Dans l'exemple décrit, il a été supposé que la carte à puce 7 est d'un type multi-applications, voire multi-utilisateurs. Il a donc été également supposé que la carte à puce 7 traite m applications logicielles, A_1 à A_m, écrites dans le langage transformé 3.

Un des langages couramment utilisés pour les cartes à puce est, comme il a été rappelé, le langage "Java Card". Il s'agit d'un langage dédié à la programmation des cartes à puce, langage qui constitue une restriction du langage "Java".

La carte 7 peut également stocker un convertisseur supplémentaire effectuant des conversions *in situ* au chargement sur des parties de codes.

Les étapes du procédé selon l'invention qui viennent d'être décrites dans un cadre général, vont être illustrées plus particulièrement dans ce cadre d'application préférée.

Comme il est connu, une implantation du langage "Java Card" fait appel à un convertisseur qui transforme des fichiers dits "de classes" en fichiers "CAP". Un fichier de classe est une unité de compilation et de représentation du code objet d'un programme "Java". Un fichier CAP regroupe toutes les classes d'un même "package Java Card" et ne comporte qu'un unique "constant pool". Un "package Java Card" est une construction "Java" pour regrouper des classes et créer des espaces de noms. Pour sa part, un "constant pool" est une table associée à chaque fichier de classe pour "Java" et à chaque fichier "CAP" pour "Java Card". Cette table regroupe des constantes (chaînes de caractères, entiers, ...). Elle est utilisée dans les machines virtuelles de "Java" et "Java Card". La transformation est non triviale et globale : elle remplace tous les noms (de packages, de classes, de champs, de méthodes) par des entités appelées "tokens", c'est-à-dire des nombres entiers de 7 ou 8 bits. Ces "tokens" servent d'index pour accéder à des tables. De plus, la transformation regroupe tous les fichiers de classes d'un même package en un fichier CAP (avec fusion des "constant pools" et réorganisation des tables de méthodes).

Le langage "Java Card" est notamment destiné à être utilisé sur des cartes à puce bancaires. Il est donc impératif de vérifier la correction de la transformation d'un programme (ou "byte code") écrit dans la machine virtuelle du langage "Java" en un programme écrit dans la machine virtuelle du langage "Java Card", c'est-à-dire d'apporter la preuve de l'équivalence de ces deux programmes.

Cette preuve formelle va être apportée en exécutant les étapes du procédé selon l'invention.

La première étape consiste en la définition d'une sémantique opérationnelle.

On associe à chaque instruction du "byte code" une ou plusieurs règles sémantiques. Le "byte code" est un code assembleur portable. C'est le

code objet pour les machines virtuelles "Java" ou "Java Card" Par exemple, la règle sémantique associée à l'une des instructions de ce code, l'instruction "getfield" peut se décrire ainsi:

$$\frac{f\_ref := \text{constant\_pool}\ (c)(i) \quad \langle c\_ref, iv\rangle := h(\tau) \quad v := iv(f\_ref)}{\langle \text{getfield}\ i;\ bc,\ r :: ops,\ l,\ c,\ h\rangle \Rightarrow \langle bc,\ v :: ops,\ 1,\ c,\ h\rangle} \qquad (9).$$

Dans l'exemple, l'état se compose du code exécuté avec l'instruction courante en tête (getfield *i; bc),* d'une pile d'opérandes (*r :: ops)* , des variables locales (*l*), d'une référence à la classe courante *(c)* et du tas *(h)*. La règle spécifie les opérations effectuées lors de l'exécution de getfield *i* :

- La fonction auxiliaire "constant_pool" utilise l'index *i* pour obtenir la référence f_ref du champ (une signature ou un "token", selon qu'il s'agit du code source ou transformé) dans le "constant pool" approprié.
- La référence τ à l'objet dont le champ doit être lu est trouvée en sommet de pile. Cette référence permet de trouver dans le tas *(h(r))* la classe dynamique de l'objet c_ref (un nom qualifié ou une paire de tokens selon qu'il s'agit du code source ou transformé) et la liste des champs de l'objet *(iv)*.
- En utilisant la référence précédemment calculée et la liste des champs, le champ est lu *(v := iv(f_ref))*.
- L'instruction getfield change l'état en replaçant la référence à l'objet par la valeur du champ et l'exécution se poursuit avec la suite du code *(bc)*.

La deuxième étape consiste en la définition des types.

Dans le cas du langage "Java Card", on définit le type Word pour représenter l'unité de stockage :

$$\text{Word} = \text{Object\_ref} + \text{Null} + \text{Boolean} + \text{Byte} + \text{Short} \qquad (10),$$

Comme exemple de type construit, le type d'un constant pool est :

$$\text{Constant\_pool} = \text{CP\_index} \rightarrow \text{CP\_inf o} \qquad (11),$$

avec :

$$\text{CP\_info} = \text{Class\_ref} + \text{Method\_ref} + \text{Field\_ref} \qquad (12).$$

Dans l'exemple, un "constant pool" est vu comme une fonction prenant un index (le type CP_index est considéré comme basique) et rendant une entrée (ici une référence à une classe, une méthode ou un champ).

Le type du "byte code" est :

$$\text{Bytecode} = \text{Instruction} + \text{Bytecode; Bytecode}$$
$$\text{Instruction} = \text{getfieldCP\_index} + \text{Invokevirtual Cp\_index} + \ldots \qquad (13).$$

Le "byte code" est une séquence d'instructions. Le type Instruction énumère toutes les instructions utilisées dans le "byte code" de "Java Card".

La troisième étape consiste en l'interprétation des types

Dans le cas de "Java Card", l'interprétation pour le code source, sous la forme de fichiers de classe (qui utilise des noms), est notée $[\![.]\!]_{name}$ et l'interprétation pour le code transformé sous la forme de fichiers CAP (qui utilise des "tokens") est notée $[\![.]\!]_{tok}$.

A titre d'exemple, le type $[\![\text{CP\_index}]\!]_{name}$ est vérifié pour le code source :

$$[\![\text{CP\_index}]\!]_{name} = \text{Class\_name x Index} \qquad (14).$$

Dans le modèle à base de noms, un index de "constant pool" est constitué d'un nom de classe (pour indiquer le "constant pool" auquel on fait référence) et d'un index.

Le type $[\![\text{CP\_index}]\!]_{tok}$ est vérifié pour le code transformé :

$$[\![\text{CP\_index}]\!]_{tok} = \text{Package\_token x Index} \qquad (15).$$

Un index de "constant pool" est constitué d'un "token" de "package" (dans l'exemple décrit, il existe un "constant pool" unique par "package" ou fichier CAP) et d'un index.

La relation R_{CP}_index est définie comme une bijection telle que : (16)

$\langle c\_name,i\rangle$ R_{CP}_index $\langle p\_tok,\ i'\rangle \Rightarrow pack\_name(c\_name)\ R_{package\_ref}\ p\_tok$

Le nom du "package" de la classe contenant le "constant pool" auquel il est fait référence dans le modèle à base de noms doit être en relation avec le "token" du "package" contenant le "constant pool" auquel il est fait référence dans le modèle à base de "tokens". La seule contrainte sur les index *i* et *i'* est que R_{cp_index} doit être une bijection (les entrées des "constant pools" peuvent donc être regroupées et réordonnées).

La quatrième étape consiste en l'interprétation des fonctions auxiliaires

Par exemple, la version de la fonction auxiliaire "constant_pool" pour le modèle à base de noms est :

$$[\![\text{constant\_pool}]\!]_{name} = \text{cp\_name} \qquad (17),$$

avec :

$$\text{cp\_name c} = \text{let (..., cp, ...)} = \text{env\_name(pack\_name(c))(c)} \qquad (18).$$

in cp

La fonction pack_name prend un nom de classe et rend un nom de "package" et la fonction env_name prend un nom de package et un nom de classe et trouve dans la hiérarchie de classes la structure représentant le fichier de classe désigné. Le constant pool est extrait du fichier de classe.

Pour le modèle à base de "tokens", la version de la fonction auxiliaire $[\![\text{constant\_pool}]\!]_{tok}$ est :

$$[\![\text{constant\_pool}]\!]_{tok} = \text{cp\_tok} \quad (19),$$

avec :

$$\text{cp\_tok c} = \text{let } (..., \text{cp}, ...) = \text{env\_tok(p)} \quad (20).$$

in cp

Le "constant pool" est trouvé dans l'environnent (c'est-à-dire les fichiers CAP) à l'aide de la fonction env_tok et du token de package.

La cinquième étape consiste à prouver que les fonctions auxiliaires respectent des relations logiques.

Si on se reporte de nouveau à l'exemple de la fonction d'accès au "constant pool", il est nécessaire de déterminer que :

$$[\![\text{constant\_pool}]\!]_{name} \; R_{\text{cp\_index} \rightarrow \text{CP\_info}} \; [\![\text{constant\_pool}]\!]_{tok} \quad (21).$$

La relation $R_{\text{cp\_index} \rightarrow \text{CP\_info}}$ est complètement définie en fonction des relations $R_{\text{CP\_index}}$ et $R_{\text{CP\_info}}$. En se servant de cette définition, on montre qu'il suffit de vérifier que :

$\forall(c\_name, i)(p\_tok, i')$ tels que $\langle c\_name, i\rangle \; R_{CP\_index} \; \langle p\_tok, i'\rangle$

$\text{cp}(i) \; R_{\text{CP\_info}} \; \text{cp}'(i')$ (22),

avec :

(..., cp, ...) = env_name(pack_name(c_name))(c_name)

(..., cp', ...) = env_tok(p_tok) (23).

La preuve se fonde sur la définition de R_{CP_info} et la propriété rappelée ci-dessus : (24)

$(c\_name, i)\ R_{cp\_index}\ \langle p\_tok, i' \rangle \Rightarrow pack\_name(c\_name)\ R_{package\_ref}\ p\_tok$

La sixième et dernière étape du procédé consiste à déterminer un transformateur tel que la transformation du code et des données par ce convertisseur respecte des relations logiques déterminées. Par exemple, les références à des "packages" sont, soit des noms, soit des "tokens" suivant le modèle. La relation logique associée $R_{package\_ref}$ est définie simplement comme une bijection entre les noms de "package" et les "tokens" de "package". Il suffit de vérifier que la fonction du convertisseur réalisant la transformation des noms de package en "tokens" est effectivement une bijection.

A la lecture de ce qui précède, on constate aisément que l'invention atteint bien les buts qu'elle s'est fixés.

Il doit être clair cependant que l'invention n'est pas limitée aux seuls exemples de réalisations explicitement décrits, notamment en relation avec les figures 2 et 3.

Enfin, bien que le procédé ait été décrit de façon détaillée dans le cas de la transformation d'un programme de la machine virtuelle du langage "Java" en un programme de la machine virtuelle du langage "Java Card", particulièrement intéressant pour les applications de type carte à puce ou similaire, l'invention n'est en aucun cas limité à cette application particulière.

L'invention peut trouver application à chaque fois que le dispositif impliqué ne dispose que de ressources informatiques relativement limitées, notamment en ce qui concerne la capacité mémoire (vive ou fixe) et/ou la

puissance de calcul du processeur utilisé. On peut citer à titre d'exemple des livres électroniques, par exemple du type dit "e-book", destinés à télécharger et stocker des données en provenance de sites Internet, des calculateurs de poche, par exemple du type dit "organiser", certains téléphones mobiles pouvant être connectés au réseau Internet, etc. Dans tous ces cas, il est nécessaire de disposer d'un langage optimisé pour utiliser au mieux les ressources informatiques intégrées.

REVENDICATIONS

1. Procédé de vérification d'un transformateur de code dit source en un code dit transformé destiné à un système embarqué, lesdits codes source et transformé étant associés à des machines virtuelles, caractérisé en ce qu'il comprend au moins les étapes suivantes :

   - la détermination, pour chacun desdits codes source (1) et transformé (3), d'un premier sous-ensemble (13) commun constituant une machine virtuelle unique factorisant le comportement de ces deux codes (1, 3) ;
   - la détermination, pour chacun desdits codes source (1) et transformé (3), d'un second sous-ensemble (10, 30) constitué d'une pluralité de fonctions dites auxiliaires (10_i - 30_i) utilisées par ladite machine virtuelle unique, lesdites fonctions auxiliaires (10_i - 30_i) représentant des différences résiduelles entre lesdits codes source (1) et transformé (3) ;
   - l'association par paire desdites fonctions auxiliaires, une première fonction auxiliaire (10_i) de chaque paire appartenant audit second sous-ensemble (10) associé audit code source (1) et une seconde fonction auxiliaire (30_i) de chaque paire appartenant audit second sous-ensemble (30) associé audit code transformé (3) ;
   - la vérification (6) d'une propriété de correspondance déterminée entre lesdites fonctions auxiliaires de toutes lesdites paires (10_i - 30_i) ; et
   - la vérification que ladite transformation du code source (1) en code transformé (3) par ledit convertisseur (2) respecte ladite propriété de correspondance déterminée.

2. Procédé selon la revendication 1, caractérisé en ce que ladite propriété de correspondance est une relation logique, de manière à ce que lesdites fonctions auxiliaires de chacune desdites paires (10_i - 30_i), lors de leur exécution, génèrent des résultats liés par ladite relation logique, et en ce que cette relation est la relation identité pour des entités dites observables

de chacun desdits codes, source et transformé, pour toute paire de fonctions auxiliaires, de manière à ce que les fonctionnalités dudit code source (1) soient préservées lors de ladite transformation dans ledit code transformé (3), et ladite vérification de transformateur de code (2) réalisée.

3. Application du procédé de vérification selon l'une des revendications 1 ou 2 à un transformateur de code (2) générant, à partir dudit code source (1) un code transformé (3) destiné à être enregistré dans des moyens de mémoire (71) d'une carte à puce (7).

4. Application selon la revendication 3, caractérisé en ce que, ledit code transformé étant un programme écrit dans la machine virtuelle d'un langage informatique déterminé, ladite carte à puce (7) est une carte à puce enregistrant une pluralité d'applications logicielles (A_1 à A_n) écrites dans ce code transformé (3).

5. Application selon les revendications 3 ou 4, caractérisé en ce que ledit code source (1) est un programme écrit dans la machine virtuelle du langage "JAVA" (marque déposée) et ledit code transformé (3) est un programme écrit dans la machine virtuelle du langage "JAVA CARD" (marque déposée).

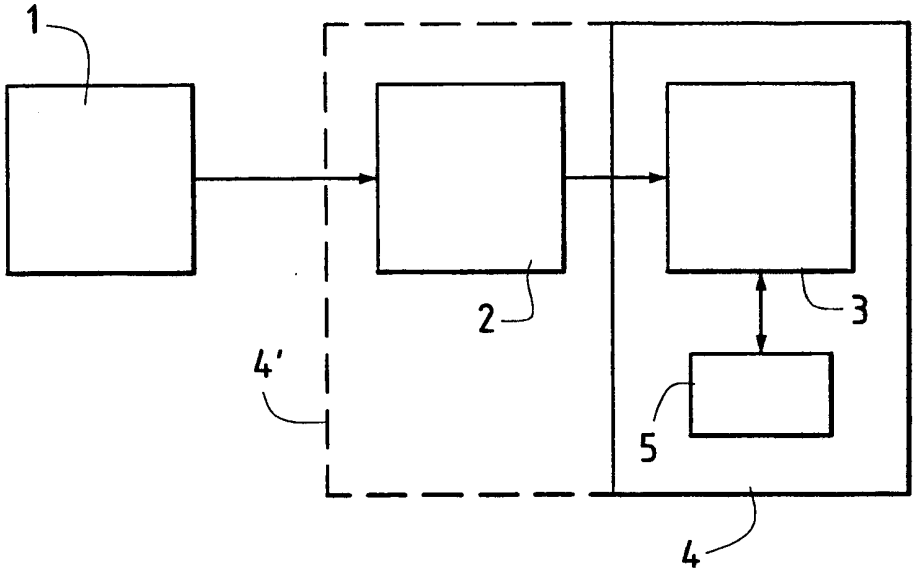


FIG.1

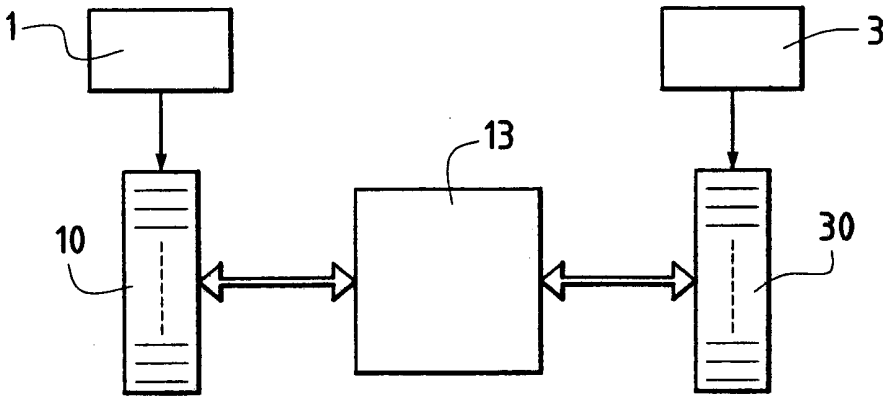


FIG.2A

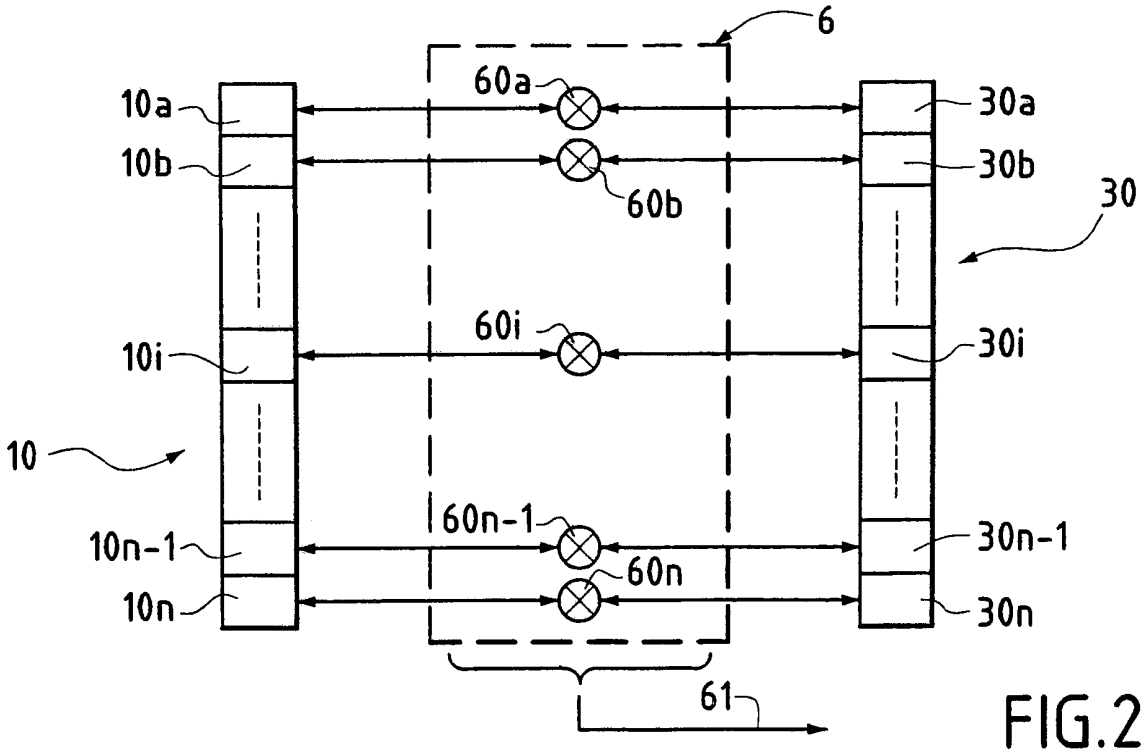


FIG.2B

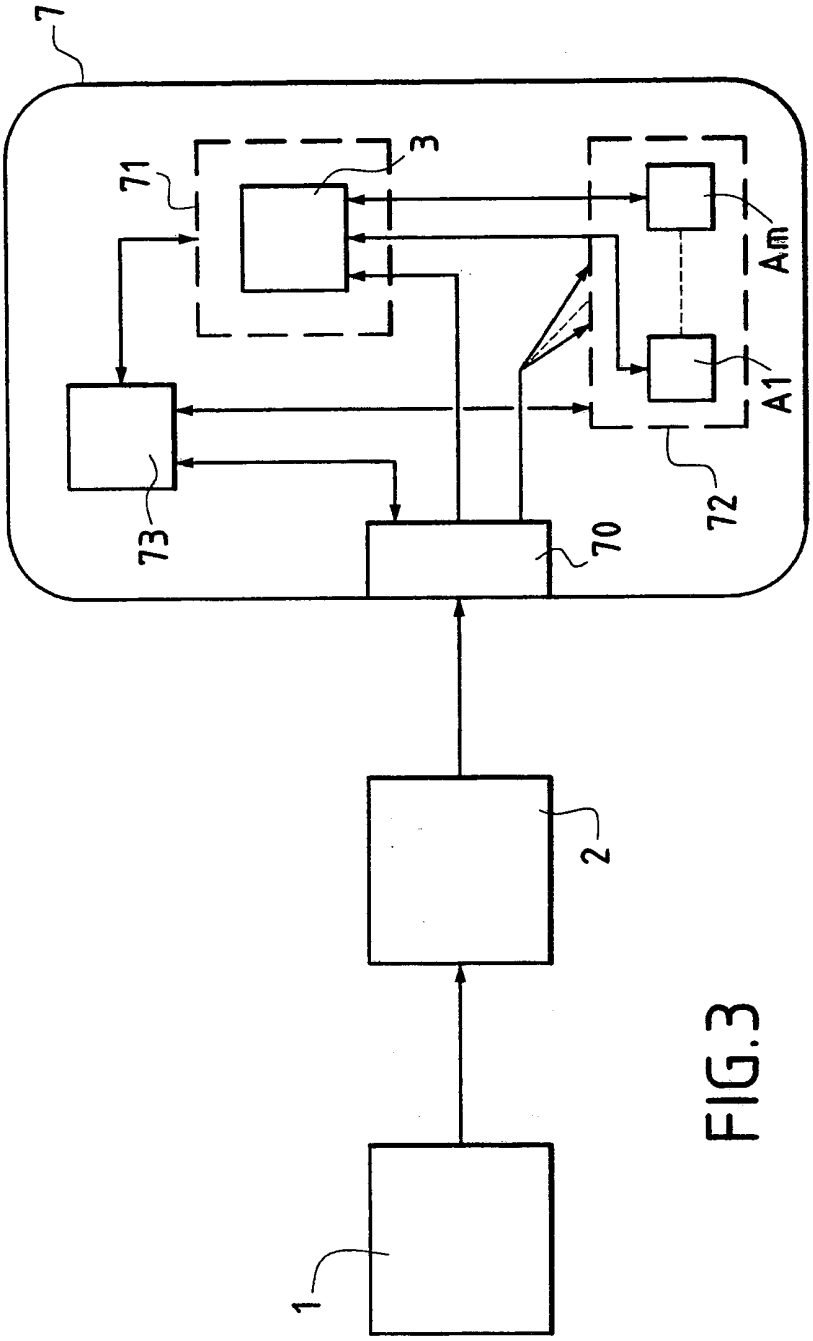


FIG.3

INTERNATIONAL SEARCH REPORT

Internal Application No
PCT/FR 00/01815

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F9/45

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F G06N

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, INSPEC, PAJ

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	KAUFMANN M ET AL: "AN INDUSTRIAL STRENGTH THEOREM PROVER FOR A LOGIC BASED ON COMMON LISP" IEEE TRANSACTIONS ON SOFTWARE ENGINEERING,US,IEEE INC. NEW YORK, vol. 23, no. 4, 1 April 1997 (1997-04-01), pages 203-213, XP000720014 ISSN: 0098-5589 the whole document ---	1,2
A	FR 2 757 970 A (GEMPLUS CARD INT) 3 July 1998 (1998-07-03) claims 1,6,7 --- -/--	1,4

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general 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" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

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

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with 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 documents, 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
1 September 2000	08/09/2000

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Kingma, Y

1

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Internal Application No
PCT/FR 00/01815

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
A	ALBERDA M I ET AL: "Using formal methods to cultivate trust in smart card operating systems" FUTURE GENERATIONS COMPUTER SYSTEMS,NL,ELSEVIER SCIENCE PUBLISHERS. AMSTERDAM, vol. 13, no. 1, 1 July 1997 (1997-07-01), pages 39-54, XP004081708 ISSN: 0167-739X page 53, left-hand column, line 2 - line 33 abstract	1-4

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

Interna al Application No
PCT/FR 00/01815

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
FR 2757970 A	03-07-1998	AU 5769598 A	31-07-1998
		CN 1247609 A	15-03-2000
		EP 1012710 A	28-06-2000
		WO 9829803 A	09-07-1998

RAPPORT DE RECHERCHE INTERNATIONALE

Dema. Internationale No
PCT/FR 00/01815

A. CLASSEMENT DE L'OBJET DE LA DEMANDE
CIB 7 G06F9/45

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)
CIB 7 G06F G06N

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 réalisable, termes de recherche utilisés)
EPO-Internal, INSPEC, PAJ

C. DOCUMENTS CONSIDERES COMME PERTINENTS

Catégorie °	Identification des documents cités, avec, le cas échéant, l'indication des passages pertinents	no. des revendications visées
A	KAUFMANN M ET AL: "AN INDUSTRIAL STRENGTH THEOREM PROVER FOR A LOGIC BASED ON COMMON LISP" IEEE TRANSACTIONS ON SOFTWARE ENGINEERING,US,IEEE INC. NEW YORK, vol. 23, no. 4, 1 avril 1997 (1997-04-01), pages 203-213, XP000720014 ISSN: 0098-5589 le document en entier ---	1,2
A	FR 2 757 970 A (GEMPLUS CARD INT) 3 juillet 1998 (1998-07-03) revendications 1,6,7 ---	1,4

-/--

[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'appartenant 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'inven tion revendiquée ne peut être considérée comme nouvelle ou comme impliquant une activité inventive par rapport au document considéré isolément

"Y" document particulièrement pertinent; l'inven tion 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
1 septembre 2000	08/09/2000

Nom et adresse postale de l'administration chargée de la recherche internationale	Fonctionnaire autorisé
Office Européen des Brevets, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31–70) 340–2040, Tx. 31 651 epo nl, Fax: (+31–70) 340–3016	Kingma, Y

1

RAPPORT DE RECHERCHE INTERNATIONALE

Dema. Internationale No
PCT/FR 00/01815

C.(suite) DOCUMENTS CONSIDERES COMME PERTINENTS

Catégorie °	Identification des documents cités, avec, le cas échéant, l'indicationdes passages pertinents	no. des revendications visées
A	ALBERDA M I ET AL: "Using formal methods to cultivate trust in smart card operating systems" FUTURE GENERATIONS COMPUTER SYSTEMS,NL,ELSEVIER SCIENCE PUBLISHERS. AMSTERDAM, vol. 13, no. 1, 1 juillet 1997 (1997-07-01), pages 39-54, XP004081708 ISSN: 0167-739X page 53, colonne de gauche, ligne 2 - ligne 33 abrégé	1-4

1

RAPPORT DE RECHERCHE INTERNATIONALE

Renseignements relatifs aux membres de familles de brevets

Demande internationale No
PCT/FR 00/01815

Document brevet cité au rapport de recherche	Date de publication	Membre(s) de la famille de brevet(s)	Date de publication
FR 2757970 A	03-07-1998	AU 5769598 A	31-07-1998
		CN 1247609 A	15-03-2000
		EP 1012710 A	28-06-2000
		WO 9829803 A	09-07-1998