



[12] 发明专利说明书

专利号 ZL 00801310.1

[45] 授权公告日 2005 年 12 月 14 日

[11] 授权公告号 CN 1231838C

[22] 申请日 2000.6.28 [21] 申请号 00801310.1

[30] 优先权

[32] 1999.7.1 [33] FR [31] 99/08460

[86] 国际申请 PCT/FR2000/001815 2000.6.28

[87] 国际公布 WO2001/002955 法 2001.1.11

[85] 进入国家阶段日期 2001.3.1

[71] 专利权人 布尔 CP8 公司

地址 法国卢旺茨那斯

共同专利权人 国家信息及自动化研究院
国家科研中心

[72] 发明人 克里斯蒂安·高雷 托马斯·简森

帕斯卡·弗拉德特

丹尼尔·勒·梅塔耶 埃文·丹尼

审查员 张雪梅

[74] 专利代理机构 中国国际贸易促进委员会专利
商标事务所

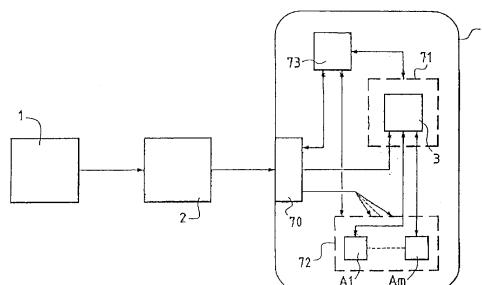
代理人 罗亚川

权利要求书 2 页 说明书 15 页 附图 2 页

[54] 发明名称 用于一种嵌入式系统的代码变换器
的检验方法

[57] 摘要

本发明涉及一种准备嵌入式系统(7)用的、从源代码到被变换代码的变换器的检验方法。该方法至少包含以下步骤：确定将这两个代码(1, 3)进行因式分解的唯一的虚拟机，对每个所述源代码(1)和被变换代码(3)，确定多个表示载所述源代码(1)和被变换代码(3)之间的剩余差值的所述辅助函数，以及一个在于检验这些辅助函数之间的对应特性的步骤，该代码变换器(2)的检验从最后步骤开始被获取。该应用尤其适合于芯片卡(7)。



1. 检验供嵌入式系统用的、从所谓源代码到被变换代码的代码变换器的方法，其中所述源代码和被变换代码与虚拟机相关联，该方法的特征在于它至少包含以下若干步骤：

- 对于所述源代码（1）和被变换代码（3）中的每一个，确定一个第一公用子集（13），该子集构成影响这二个代码（1，3）的行为的单个虚拟机；

- 对于每个所述源代码（1）和被变换代码（3），确定一个第二子集（10，30），该子集由多个被所述单个虚拟机使用的所谓辅助函数（10_i-30_i）构成，所述辅助函数（10_i-30_i）表示在所述源代码（1）和被变换代码（3）之间的剩余差；

- 将辅助函数成对地关联起来，每一个对中的第一辅助函数（10_i）属于与所述源代码（1）相关联的所述第二子集（10），且每一个对中的第二辅助函数（30_i）属于与所述被变换代码（3）相关联的所述第二子集（30）；

- 对所有所述对的辅助函数（10_i-30_i）之间的一个给定的对应特性进行检验；以及

- 验证由所述变换器（2）进行的从源代码（1）到被变换代码（3）的变换遵守所述给定的对应特性。

2. 根据权利要求 1 的方法，其特征在于，所述对应特性是一种逻辑关系，从而使每一个对所述辅助函数（10_i-30_i）在被执行时能产生受所述逻辑关系约束的结果，且该方法的特征还在于，对于任一辅助函数对，对于每个所述源代码和被变换代码的所谓的可观察的实体，这种关系是关系恒等式，使得当所述被变换代码（3）的所述变换和代码变换器（2）的检验被执行时，所述源代码（1）的功能性被保持。

3. 根据权利要求 1 的方法，其中所应用的代码变换器根据所述源代码（1），产生将被存储在芯片卡（7）的存储器设备（71）中的

被变换代码(3)。

4. 根据权利要求3的方法，其中所述被变换代码是在给定的一种计算机语言的虚拟机中所编写的程序，所述芯片卡(7)是存储多个以所述被变换代码(3)所编写的应用软件(A₁-A_n)的芯片卡。

5. 根据权利要求3或4的方法，其中所述源代码(1)是在“JAVA”语言的虚拟机中所编写的程序，而被变换代码(3)是在“JAVA CARD”语言的虚拟机中所编写的程序。

用于一种嵌入式系统
的代码变换器的检验方法

技术领域

本发明涉及用于一种嵌入式系统的代码变换器的检验方法。本发明还涉及在用于产生供芯片卡用的代码变换器中的这种方法的应用。

背景技术

就本发明而言，“嵌入式系统”术语应在它的最普通的词义中考虑。尤其涉及供一种芯片卡用的系统，该芯片卡是本发明的优先应用，但任何系统也供便携式的或移动的设备之用，该装置包含在下面叫做“处理资源”的信息化的数据处理的固有设备。

允许执行越来越复杂和越来越多的数据处理资源供给现代嵌入式系统。然而，虽然越来越多的竞争强的技术和部件投放市场，但是和传统的信息系统（电子计算机，工作站等）比较，这些嵌入式系统的有特色的特征涉及在资源方面（尤其是微处理器的存储器容量和功率）被强制性的限制。为了满足这些限制，必须变换供在嵌入式系统上被执行之用的代码。变换的目的旨在产生一种更有效的和更经济的代码作为资源。

为了固定这些想法，作为非限定代码的例子，人们将考虑在该例子中带有在“JAVA”（SUN MICROSYSTEMS 的注册商标）语言的虚拟机中编写的程序，该语言具有能在很多环境中被执行的好处。该语言的应用范围实际上尤其随着因特网的大量发展而增多。许多小容量的叫做“小应用程序”（applet）的应用软件用该语言来描述并可以通过“WEB”型导航器被执行。

人们也将被安排在本发明的优先应用范围，即由一种芯片卡的固有信息资源执行该类型的代码。象以上被提醒的那样，虽然取得重大的技术进步，但芯片卡的存储器的容量以及所装备的微处理器的功率保持相对的限制。同等重要的是，该代码假定驻留在芯片卡中，因为在后者和

主机终端之间的传输不管是否以低速度被实行。实际标准只规定连续型的传输。因此人们感觉到需要配置一种人们可以看作是“（传输）被减轻的”、在任何情况下对于这种使用是最理想的代码。为做到这一点，建议使用由一种由“JAVA”派生出来的语言，它以该语言的紧缩形式出现，即“JAVA CARD”（同样是 SUN MICROSYSTEMS 注册商标）语言。

一种额外的复杂来自于这样的事实，即这些嵌入式系统通常在这些环境中被使用，该环境在材料方面，同时在可靠性和安全性方面要求得到最高的保证。作为例子人们可以引用新型芯片卡，在该卡上，人们希望安装多种应用软件，在不泄露秘密信息的情况下这些软件应该和谐地协作。事实上，一开始，这些应用软件可能涉及不同的用户。然而，人们将上述的协作应保持严格的隔开，以便使涉及用户的信息允许保持秘密，至少不能由一个未授权知道它们（读）和/或操作它们（写和相关操作：抹去，修改）的用户操作。除“保密性”方面之外，其他的考虑有所谓“完整性”的要求：数据丢失，不合适的修改等。

如果人们在“初始代码”的意义上考虑该“源”代码，例如，象上述“JAVA”语言的“字节代码”，那么，最后一个代码将表现出必要的所有保证并履行上述的要求，该“字节代码”是在“JAVA”语言的虚拟机中编写的程序。事实上，在长时间周期期间，大量的实验能够被执行。

所述“被变换的”代码从“源代码”开始，借助于通常是嵌入式系统外部的代码变换器获得，但是，该代码同样地可以驻留在后者之中。因此，需要表现出源代码和被变换的代码之间的等效。

在保证丝毫不改变它的特性（从外部观点看）和不导致安全的缺陷而对该代码执行变换的同时，该等效可以被执行。换句话说，该起始代码（变换前）应是等价于结果代码（变换后）的逻辑观察点。

通常难以保证这种特性，因为这些变换对于该代码和被操作的数据的表示法具有整体作用。实际上，这种操作所包含的复杂性不允许在注重现实的经济和/或技术的条件下实行。人们还应懂得这种需要只是最

近才产生，尤其与多应用和/或多用户的芯片卡的上述技术的发展相关。

发明内容

本发明就是以满足这些需要为目的，而不需要非常长的和昂贵的工序。

根据本发明的方法允许用系统和模块的方法检验代码变换的正确。

在本发明的范围，被充分认识的二个形式化方法本身将主要地用于：运算语义学和逻辑关系。为了详细地说明这些形式化方法，第一，人们将有益地参照 H.R.Nielson 和 F.Nielsonl 的书，标题为：“应用语义学：形式引论” Wiley,1992 以及第二， J.Mitchel 的书，标题为：“编程语言基础” MIT Press 1996。

根据本发明的一个主要特征，代码变换器的检验方法在于借助于用叫做“辅助函数”的函数作为参数的公用虚拟机规定二个代码的意义。二个代码的差别在上述辅助函数中被表达和重组。每个辅助函数具有二种形式：源代码形式和变换的代码形式。第一种模块是相同的，因为它为二个代码所共有，如果是相等，就没有要检验之处。为显示二个代码的等价，因此只需显示一对一对地被考虑的所述辅助函数是等价的就足够。这二个子集比起用考虑到它们的完整性的源代码和被变换的代码所表示的二个集合变得不复杂。由此可见，根据本发明的方法检验过程中固有的难度大大地简化，并且以相关的方式该检验过程在经济上和在技术上将变得可以实现。

因此本发明的目的是提供检验供嵌入式系统用的、从所谓源代码到被变换代码的代码变换器的方法，其中所述源代码和被变换代码与虚拟机相关联，该方法的特征在于它至少包含以下若干步骤：

- 对于所述源代码和被变换代码中的每一个，确定一个第一公用子集，该子集构成影响这二个代码的行为的单个虚拟机；
- 对于每个所述源代码和被变换代码，确定一个第二子集，该子集由多个被所述单个虚拟机使用的所谓辅助函数构成，所述辅助函数表示在所述源代码和被变换代码之间的剩余差；
- 将辅助函数成对地关联起来，每一个对中的第一辅助函数属于与

所述源代码相关联的所述第二子集，且每一个对中的第二辅助函数属于与所述被变换代码相关联的所述第二子集一个；

- 对所有所述对的辅助函数之间的一个给定的对应特性进行检验；
以及

- 验证由所述变换器进行的从源代码到被变换代码的变换遵守所述给定的对应特性。

本发明的目的是还提供一种根据上述方法的代码变换器，根据源代码，代码变换器产生将被存储在芯片卡的存储器设备中的被变换代码。

附图说明

本发明现在将参照附图更详细地被说明，在这些附图中：

- 图 1 概略地说明从源代码到最后的被变换代码的变换过程；
- 图 2A 和 2B 概略地说明根据本发明的方法的主要特征；以及
- 图 3 概略地说明从根据本发明的方法到一种芯片卡的应用。

具体实施方式

现在我们将详细地说明根据本发明的代码变换器的检验方法。

图 1 概略地说明借助于代码变换器 2 从叫做在原点代码或起始代码的含义下的“源代码”的代码 1 到所述“被变换代码”的最后代码 3 的变换过程。最后的装置可以是一种专用的信息设备或软件。通常，该变换代码驻留在嵌入式系统 4 中（实线部分）。变换器 2 同样可以被驻留在或远程加载在嵌入式系统中：参考（标号）4'（虚线部分）。

在加载在或记载在嵌入式系统 4-4' 之后，根据需要，该变换代码 3 允许执行唯一参考标号 5 所表示的一个或多个任务。假定嵌入式系统 4 拥有传统的独立的信息资源（未示出）。

一开始，该代码变换通过给定的变换器 2 只被执行一次或在很少场合被执行：例如，原点代码或源代码 1 的形式的修改。

因此，必须使人们能确立该变换代码 3 等价于源代码 1 的形式验证。该过程允许检验该变换器 2 是否正确地运行。

但是，象所想到的那样，在它们的整体中，如果将考虑由源代码和被变换代码形成的二个集合，那么，理论表明这样的确定按现实的方式

通常是不可能的。

根据本发明的方法的一个主要特征将在于为二个代码中的每一个找到二个叫做第一和第二子集的二个子集。根据本发明的方法的一个重要特征，第一子集形成为二个代码即源代码和被变换代码所共有的虚拟机。因此，不必检验第一子集的等效。

由辅助函数组成的第二子集相反地在代码上一个不同于另一个。那么，源代码和变换代码的等效的确定被简化为确定所有成对的第二子集的辅助函数的等效。不过，辅助函数的剩余的复杂性可能大大地减小。于是，上述等效的确定将变为可能。

图 2A 和 2B 非常概略地说明根据本发明的方法。

尤其象图 2A 所指出的那样，源代码 1 和被变换代码 3 的第一子集形成共有的虚拟机 13。通过一系列所述辅助函数每个第二子集 10 和 30 被建立，辅助函数的等效应该被检验。这些辅助函数 10 和 30 将给共有的虚拟机 13 作参数。

因此象在下面参照图 2B 将被指出的那样，二个代码，即源代码 1 和被变换代码 3 的等效被简化为检验二个二个地被选取的辅助函数的等效。

这些源代码和被变换代码分别与所述虚拟的第一和第二机器相结合。

第一步骤包括确定允许对源代码和被变换代码的特性进行因式分解的唯一的虚拟机（或运算语义学）。那么，二个代码之间的差别体现在二个代码中的解释或不同地使用辅助函数。

一个虚拟机可以通过形式规则的集合被表现：

pr é misse (前提) 1

.

.

.

pr é misse (前提) n

é tat (状态) 1[instruction (指令) 1] ⇒ é tat (状态) 2 (1)

这些前提或者是一个规则的应用的条件，也就是说是布尔表达式，或者是用于表达状态变化的被使用的变量的赋值。前提要求调用用于提取状态信息或表达这些条件的辅助函数。当前提被检验和指令“instruction1”被碰见时每个规则指出该机器状态怎样变化。对于代码指令的每个类型确定一个或多个这种形式的规则。

第二个步骤在于确定在二个代码中所使用的数据的结构和类型。确定这些基本类型，例如：

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

以及例如如下构成的结构：

Environnement(环境)::=Nom (名字) → Valeur (值)

Instruction ::= Instruction1|Instruction2|... (3)

第三个步骤在于提供在虚拟机中使用的被标识为 θ 的类型解释。对于每个 θ 类型确定一个源代码 $[\![\theta]\!]_S$ 的解释和一个被变换代码 $[\![\theta]\!]_T$ 的解释。还有一个遵守类型结构的在二个叫做逻辑关系的解释 $[\![.\].]_S$ 和 $[\![.\].]_T$ 之间的关系 R_θ 。对于简单的类型，它们应被明确地确定；对于结构类型，它们将从该结构的成分中被推断出来。

例如，对于这些对来说：

$(a, b) R_{\theta_1 \times \theta_2} (a', b') \Leftrightarrow a R_{\theta_1} a' \wedge b R_{\theta_2} b' (4)$

在该关系中 θ_1 和 θ_2 是类型，而 a, b, a' 和 b' 是类型的元素。

$$f R_{\theta_1 \rightarrow \theta_2} f' \Leftrightarrow \forall a, a'. a R_{\theta_1} a' \Rightarrow f a R_{\theta_2} f' a' \quad (5)$$

对于可观察的类型这些逻辑关系应是“同一性”关系，就是说对这些关系的类型能指出提供相同结果的二个代码。通常在信息屏幕上涉及可以打印和/或可以显示的类型。这可能是基本类型，但所构成的类型例如也表示一个给定的程序栈或程序变量。

第四个步骤在虚拟机中所使用的辅助函数的解释。对于每个辅助函数 f ，在于源代码给出它的定义，记为 $\llbracket f \rrbracket_s$ ，以及为被变换代码给出它的定义，记为 $\llbracket f \rrbracket_T$ 。

等效的确定在于指出遵守逻辑关系的辅助函数的定义。更明确地说，对于每个辅助函数 $f: \theta \rightarrow \theta'$ ，人们指出

$$\llbracket f \rrbracket_s R_{\theta \rightarrow \theta'} \llbracket f \rrbracket_T \quad (6).$$

由此可见，二个虚拟机是有关联的，就是说：

$$\llbracket \text{état} \rrbracket_s R_{\text{type-état}} \llbracket \text{état} \rrbracket_T \quad (7).$$

由于这些关系对于可观察的类型是同一性，因此，源代码和被变换代码被观察为是相同的。

最后的步骤在于指出存在一种满足逻辑关系的变换器 Γ 。这可通过检验使得给定的变换器 $\Gamma: S \rightarrow T$ 满足与源代码 S 和被变换代码 T 的它的自变量类型相关联的逻辑关系来做到。为做到这点，必须遵守下面的关系：

$$\forall x \llbracket \theta \rrbracket_s . x R_\theta \Gamma(x) \quad (8).$$

刚刚指出这些逻辑关系规定一种约束集合。因此应用精练技术或提

取技术、并使用合适的证明的辅助者之一，通过构造在该集合中提取一个正确的变换器 2。

因此，根据本发明的方法具有一个重要的优势，因为它允许检验过程的大量机械化，尤其允许它获得好处，因为这种检验在不太复杂的子集上进行。

如果该源代码 1 的变换能被描述为一系列的更简单的变换，那么该方法可以被应用以便独立地指出每个变换。由此可见，它具有很多模块化的优势。

象图 2B 所说明的那样，该检验借助于一种硬件或软件设备 6 只需对辅助函数 10 和 30 的子集执行。假设存在着 n 个辅助函数，分别标记为 $10_a, 10_b, \dots, 10_i, \dots, 10_n$ 和 $30_a, 30_b, \dots, 30_j, \dots, 30_{n-1}, 30_n$ 。如果该设备 6 是硬件，那么，它包含与要检验的辅助函数对同样多的检验电路 $60_a, 60_b, \dots, 60_i, \dots, 60_{n-1}, 60_n$ （用比较器的符号在图 2B 上任意地表示），例如，对于函数 10_i 和 30_i 的对的该检验电路 60_i 。该设备 6 的输出或者输出在唯一参考标记 61 下指出在对应的源代码 1 和被变换代码 3 的辅助函数的所有可能的对之间的逻辑关系被满足。这一系列操作在它们的整体上足以提供二个代码的等效的形式证据。

要注意，根据本发明的方法也可以经验地完全被利用，就是说一开始象一个新的变换器的研制助手用来检验现存的变换器。尤其在最后的情况下，它允许确定变换器的特性，以便正确地运行，换句话说，以便用该变换器从源代码产生的被变换代码满足上述等效要求。

现在让人们回到芯片卡的范围。图 3 概要地说明参考标号为 7 的芯片卡的结构。在完全理解根据本发明的方法后，人们在该图上只示出主要部件。

尤其是芯片卡 7 包含一个允许和外部世界通信的输入-输出设备 70，一个固定的或可编程的（“ROM”，“PROM”，“EPROM”或“EEPROM”类型）的存储器的第一设备 71 以及一个随机存取存储器 72。该芯片卡 7 最后包含一个通过总线和该芯片卡 7 的其它部件通信的微处理器或微控制器 73。

一种这样的芯片卡 7 的软件结构符合 ISO7816-3 标准，它通过翻译成从与输入-输出设备 70 结合的最低层直到与被记载在该芯片卡 7 中的应用软件结合的最高层的协议被表示。该标准规定这些传输使用串行方式被执行。

一旦通过代码变换器 2 被变换的源代码 1 被发送到芯片卡 7，以便经由输入-输出设备 70 被写入通常到是固定的或“半固定”的存储器设备 71 中。用芯片卡 7 处理的应用软件可以永久地写入芯片卡 7 中，就是说，写入在存储器设备 71 中，或暂时写入到随机存取存储器 72 中。在后者的情况下，这些应用软件经由输入-输出设备 70 被远程加载。在所描述的例子中，假设该芯片卡 7 是多应用、甚至是多用户类型的卡。因此，同样假设芯片卡 7 处理应用软件 m，即在被变换的语言 3 中所描述的 A₁ 至 A_m。

象被提及的，通常对于芯片卡 7 所使用的语言是“Java Card”语言。它涉及专门用于芯片卡的程序设计的语言，该语言受“Java”语言的限制。

该芯片卡 7 同样可以储存在对部分代码加载时在原位执行变换的附加变换器。

刚才在一般范围被描述的根据本发明的方法的步骤在优先应用范围将更详细地说明。

象人们已经知道的那样，“Java Card”语言的插入要求一个将所谓的“分类”文件变成“CAP”文件的变换器（变换程序）。分类文件是“Java”程序的目标代码的编译和表示的单元。一个 CAP 文件对所有相同的“Java 卡程序包”的分类分组并且只包含一种唯一的“常数库”。一个“Java 卡程序包”是一种用于与分类分组和建立名称空间的“Java”结构。“常数库”是与每个“Java”分类文件和每个“Java 卡”相关联的“CAP”文件的表。该表包含常数（一系列符号，整数，...）。它在“Java”和“Java 卡”的虚拟机中使用。该变换是非平凡的和全部的：它是用叫做“令牌（token）”的实体代替所有的名称（程序包的，分类的，字段的，方法的名称），就是说 7

或 8 位整数。“令牌”作为访问表的下标。此外，该变换对所有在 CAP 文件中的相同的程序包的分类文件进行分组（包含“常数库”的合并和方法表的重组）。

该“Java card”尤其准备在银行芯片卡中被使用。因此，迫切需要检验从用“Java”语言的虚拟机编写的程序（或“字节代码”）到用“Java 代码”的虚拟机编写的程序的变换的精确性，就是说，产生二个程序的等效证据。

该形式证明将在执行根据本发明的步骤的同时被产生。

第一步骤包括定义运算语义学。

人们将一个或多个语义学规则和每个“字节代码”指令关联起来。该“字节代码”是可移植的汇编语言代码。它是用于“Java”或“Java card”虚拟机的目标代码。例如，该语义学规则和该代码指令之一关联，“取得字段 (getfield)”指令也可以描述为：

```
f_ref := constant_pool (c)(i)
(c_ref, iv) := h(r)
v := iv(f_ref)
```

$\langle \text{getfield } i; bc, r :: \text{ops}, l, c, h \rangle \Rightarrow \langle bc, v :: \text{ops}, 1, c, h \rangle$ (9).

在该例子中，该状态包括在开始部分的执行的 (`getfield i;bc`) 具有现有指令的代码，一组操作数 (`r::ops`)，局部变量 (`l`)，一个分类流调用 (`c`) 和堆栈 (`h`)。在执行取得字段 `i` 时该规则规定操作的执行：

- 辅助函数“`constant_pool`”使用下标 `i` 以便在合适的“常数库”中获取字段调用 `f_ref`（一种签名或“令牌”，根据是否是原代码或被

变换代码).

- 属于目标的调用 τ , 它的应被读出的字段在栈顶被找到。该调用允许在堆栈 ($h(r)$) 中找到目标动态分类 c_ref (根据涉及原代码和被变换代码的一个合格的名称或令牌对) 和目标字段表 (iv).

- 使用以前被计算的调用和字段表, 该字段被读出 ($v:=iv(f_ref)$).
- 用该字段值替代目标调用的同时该指令取得字段将改变状态, 并且该执行随着代码(bc)的其余部分继续进行。

第二个步骤包括类型的定义。

在 “Java card” 语言情况下, 定义字类型以便表示存储单位:

Word=Object_ref+Null+Boolean+Byte+Short (10),

作为被构成的例子, 一个常数库的类型是:

constant_pool=CP_index → CP_info (11)

以及

CP_info=Class_ref+Method_ref+Field_ref (12)

在该例子中, 一个“常数库”被看作一个取下标 (类型 CP_index 被认为是基本的) 和提供一个输入端 (这里, 是一个分类, 方法或字段的调用) 的函数。

“字节”代码类型是:

Bytecode=Instruction+Bytecode;Bytecode

Instruction=getfieldCP_index+Invokevirtual Cp_index+... (13)

该“字节代码”是一个指令序列。指令类型列举了在“Java card”的“字节代码”中被使用的所有指令。

第三个步骤包括类型解释。

在“Java card”的情况下，用于源代码的解释以分类文件的形式（使用名称）记为 $[[.]]_{name}$ ，而用于被变换代码的解释以 CAP 文件的形式（使用“令牌”）记为 $[[.]]_{tok}$ 。

作为例子，对于源代码类型 $[[CP_index]]_{name}$ 被检验：

$$[[CP_index]]_{name} = Cass_name \times Index \quad (14)$$

在基于名称的模型中，“常数库”的下标由分类名称（以便指出要调用的“常数库”）和下标组成。

对于被变换代码类型 $[[CP_index]]_{tok}$ 被检验：

$$[[CP_index]]_{tok} = Package_token \times index \quad (15)$$

一个“常数库”的下标由一个“程序包”的“令牌”（在所述例子中，每个“程序包”或 CAP 文件只有一个“常数库”）和一个下标组成。

关系 R_{cp_index} 被定义为一个双映射，如：

$$\begin{aligned} & \langle c_name, i \rangle \quad R_{cp_index} \quad (p_tok, i') \quad \Rightarrow \quad pack_name(c_name) \\ & R_{pacakage_ref} \quad p_tok \end{aligned} \quad (16)$$

该分类的“程序包”的名称包含“常数库”，该常数库在基于名称的模块中被参考，该“程序包”的名称应和“程序包”的“令牌”有联系，该“程序包”的“令牌”包含“常数库”，在基于“令牌”的模型中参考该“常数库”。对下标 i 和 i' 的唯一限制是 R_{cp_index} 应是一个双映射（因此，“常数库”的输入端可以被再分组和重新整理）。

第四个步骤包括辅助函数的解释。

例如，用于基于名称模块的辅助函数“constant_pool”的形式是：

$$[[\text{constant_pool}]]_{\text{name}} = \text{cp_name} \quad (17)$$

以及

$$\text{cp_name } c = \text{let } (\dots, \text{cp}, \dots) = \text{env_name}(\text{pack_name}(c))(c) \quad (18)$$

in cp

函数 pack_name 选取一个分类名称和产生一个“程序包”名称，而函数 env_name 选取一个程序包名称和一个分类名称，并在分类等级中找到表示所指定的分类文件的结构。该常数库从分类文件提取。

对于基于“令牌”的模型，辅助函数 $[[\text{constant_pool}]]_{\text{tok}}$ 的形式是：

$$[[\text{constant_pool}]]_{\text{tok}} = \text{cp_tok} \quad (19)$$

以及：

$$\text{cp_tok } c = (\dots, \text{cp}, \dots) = \text{env_tok}(p) \quad (20)$$

in cp

“常数库”借助于函数 env_tok 和程序包的令牌在环境(就是 CAP 文件)中被找到。

第五个步骤在于证明辅助函数遵守逻辑关系。

如果参照按照访问“常数库”的函数的例子，那么必须确定：

$$[[\text{constant_pool}]]_{\text{name}} \ R_{\text{cp_index}} \rightarrow \text{cp_info} \ [[\text{constant_pool}]]_{\text{tok}} \quad (21)$$

关系 $R_{cp_index} \rightarrow cp_info$ 完全由关系函数 R_{cp_index} 和 R_{cp_info} 定义。因此使用该定义后，人们指出只需检验以下等式就足够了：

$$\forall (c_name, i)(p_tok, i') \text{ tels que } (c_name, i) R_{cp_index} (p_tok, i') \\ cp(i) R_{CP_Info} cp'(i') \quad (22),$$

以及：

$$(\dots, cp, \dots) = env_name(pack_name(c_name))(c_name) \\ (\dots, cp', \dots) = env_tok(p_tok) \quad (23)$$

该证明基于 R_{cp_info} 的定义中，而特性如上被提及：

$$(c_name, i) R_{cp_index} (p_tok, i') \Rightarrow pack_name(c_name) R_{pa_package_ref} p_t \\ ok \quad (24)$$

该方法的第六和最后的步骤在于确定变换器，使得由该变换器确定的代码和数据变换遵守给定的逻辑关系。例如，“程序包”的调用是依赖模型的名称或“令牌”。被关联的 $R_{package_ref}$ 的逻辑关系只是被定义作为在“程序包”名称和“程序包令牌”之间的一个双映射。只需检验实现从程序包名称到“令牌”变换的变换器的函数实际上就是双映射就足够了。

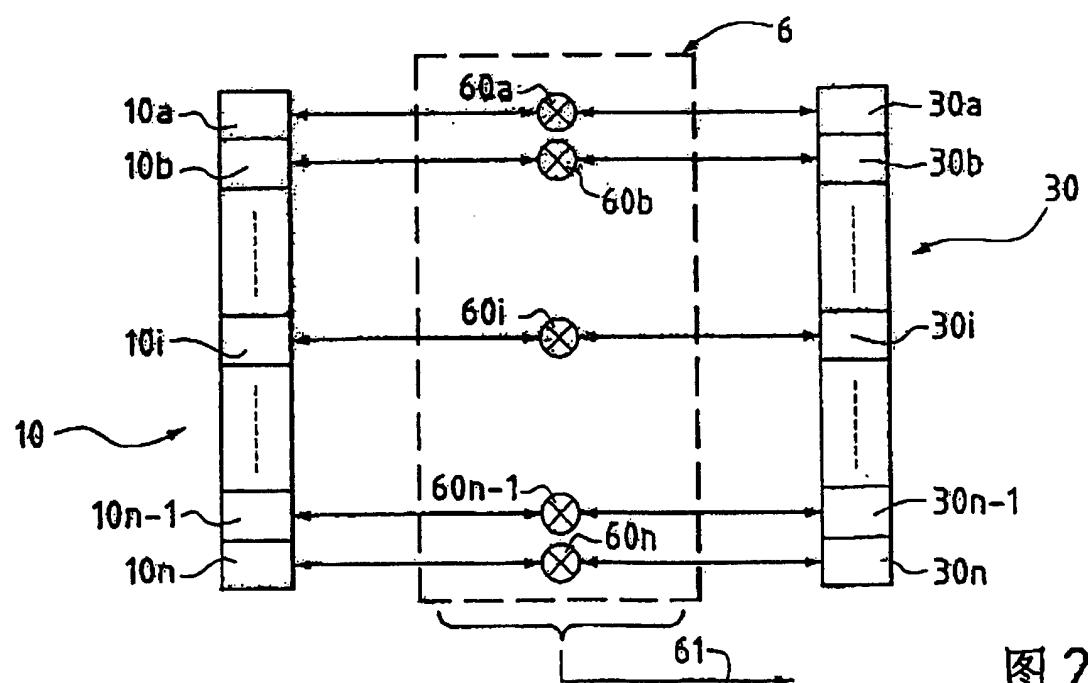
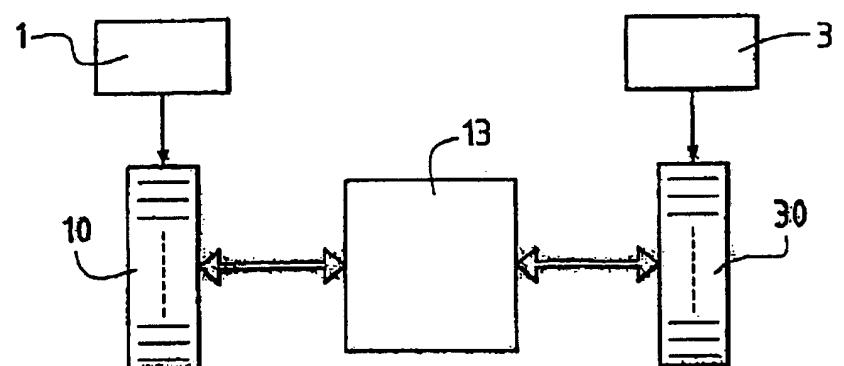
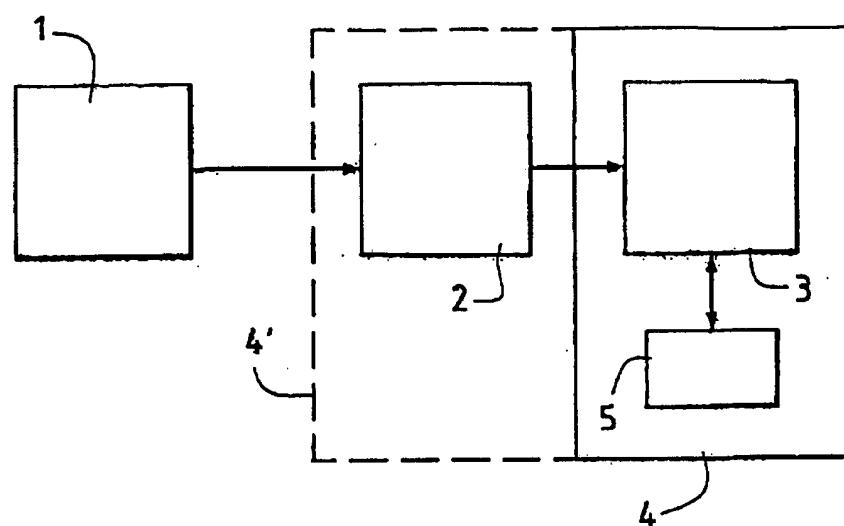
在以前的阅读中，人们容易地看到本发明完全达到它规定的目标。

但是，应当清楚的是，本发明不受尤其与图 2 和 3 有关的明确地被描述的唯一实施例的限定。

最后，该方法在从“Java”语言的虚拟机的程序到“Java card”语言的虚拟机的程序的变换的情况下详细地被描述，尤其涉及芯片类型或类似的应用，但是，本发明在任何情况下不受这种特殊应用的限制。

每当本发明能够找到应用时，所包含的设备只配置相对有限的信

息资源，尤其在涉及存储器（随机存取的或固定的）的容量和/或所使用的处理器的计算功率方面。人们可以列举准备用于从所述例如“organiser(组织)”类型的袖珍计算机远程加载和储存来自 Internet 网络现场的数据的例如“e-book”类型的电子图书，以及某些可以连结到 Internet 网络的移动电话等。在所有这些情况下，都必须配置最佳语言以便最好地使用被集成的信息资源。



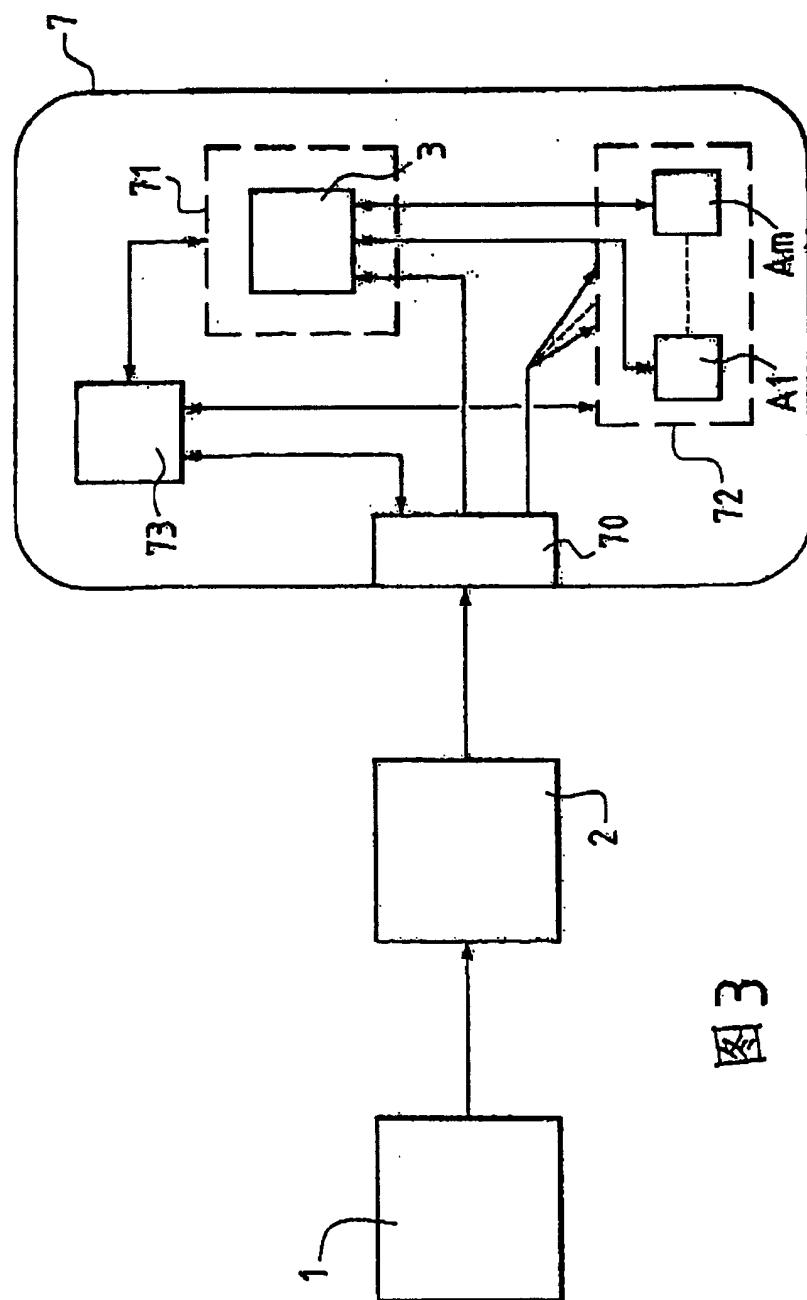


图3