(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 1641583 B
(45)授权公告日 2010.09.29

(21)申请号 200410102286.0

(22)申请日 2004.12.16

(30)优先权数据
60/530,129 2003.12.16 US
10/837,024 2004.05.01 US

(73)专利权人 微软公司
地址 美国华盛顿州

(72)发明人 B·克罗斯 J·格劳姆
L·麦克内尔 M·马克利 S·谢尔

(74)专利代理机构 上海专利商标事务所有限公司 31100
代理人 顾嘉运

(51)Int.Cl.
G06F 9/445(2006.01)

(56)对比文件
US 6381742 B2,2002.04.30, 说明书第6栏第36行-第7栏第60行、附图2A-2C.
全文.
WO 02/27444 A2,2002.04.04, 说明书第7页第4行-第7行, 附图4-5.
全文.
US 20030033480 A1,2003.02.13, 全文.
CN 1329309 A, 权利要求1.

审查员 鞠博

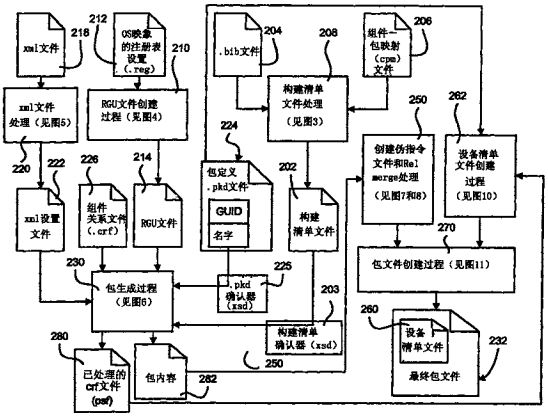
权利要求书 2 页 说明书 17 页 附图 13 页

(54)发明名称

自描述软件映象更新组件

(57)摘要

所描述的是一种系统和方法,其中,将包括更新的软件映象作为包来提供。该包是自描述的,由此方便了仅替换映象内的组件部分。一软件构建过程将操作系统特征(包括文件、元数据、配置信息等)映射到包,并且为定位目的处理可执行代码。最终的包包括描述包并传递依赖信息和关于设置优先级的信息(阴影数据)的设备清单文件。



CN 1641583 B

1. 在包括具有操作系统文件和其它关联的软件应用程序文件的软件映象的移动计算环境中，一种用于更新所述软件映象中的文件的一部分的方法，其特征在于，所述方法包括：

构建一包，所述包对应于一操作系统映象的一部分、并包含文件和设置，其中所述包的格式是自描述的；

通过采用所述操作系统映象的二进制映象构建器文件和一组件－包映射文件作为输入来生成指定所述包的文件内容的构建清单文件；

生成包括包依赖性以及包设置优先级数据的设备清单文件，其中所述设备清单文件存储在所述包内，其中所述设备清单文件进一步包括从所述构建清单文件导出的列表和属性信息，并且所述包设置优先级数据包括描述设置数据相对于其它设置数据的优先级的阴影信息；以及

将在所述构建清单文件以及所述设备清单文件内描述所述包的内容的信息与所述包相关联，使得一安装机制可确定如何将所述包安装到一设备。

2. 如权利要求 1 所述的方法，其特征在于，所述方法还包括向所述包添加所述设备清单文件。

3. 如权利要求 2 所述的方法，其特征在于，它还包括将包括描述所述包的至少部分内容对另一实体的依赖性信息的数据写入所述设备清单文件中。

4. 如权利要求 3 所述的方法，其特征在于，所述另一实体包括另一包，并且所述方法还包括从所述包提取所述设备清单文件，并从所述另一包提取另一设备清单文件。

5. 如权利要求 1 所述的方法，其特征在于，构建包包括确定哪些文件是可执行文件以及在所述可执行文件上执行一重定位过程。

6. 如权利要求 1 所述的方法，其特征在于，将描述所述包的内容的信息与所述包相关联包括：关联描述所述包的至少部分内容对另一实体的依赖性的依赖性信息。

7. 如权利要求 1 所述的方法，其特征在于，将描述所述包的内容的信息与所述包相关联包括：关联描述设置数据相对于其它设置数据的优先级的阴影信息。

8. 如权利要求 1 所述的方法，其特征在于，它还包括向所述包添加设置信息。

9. 如权利要求 1 所述的方法，其特征在于，构建包包括：将包映射到该包的定义现有文件的定义上、读取一指定该包的文件内容的构建清单文件、以及基于所述包定义和所述构建清单文件生成所述包。

10. 如权利要求 1 所述的方法，其特征在于，它还包括处理所述构建清单文件中标识的可执行代码，以使所述可执行代码能够在安装时重定位到一设备。

11. 如权利要求 1 所述的方法，其特征在于，构建所述包包括：读取一二进制映象构建器文件，其包含在构建时要包括在所述包内的文件的列表。

12. 如权利要求 11 所述的方法，其特征在于，构建所述包包括：创建一对应于所述二进制映象构建器文件的构建清单文件、以及读取所述构建清单文件。

13. 如权利要求 1 所述的方法，其特征在于，构建所述包包括：读取一设置文件，其包含当构建时要包括在所述包内的设置信息的列表。

14. 如权利要求 1 所述的方法，其特征在于，构建所述包包括：读取一组件映射文件，其将模块相关信息及文件映射到所述包。

15. 如权利要求 1 所述的方法，其特征在于，构建所述包包括 ：读取一定义阴影关系的组件关系文件。

16. 如权利要求 15 所述的方法，其特征在于，与所述包相关联的所述信息包括一设备清单文件，并且所述方法还包括 ：基于所述阴影关系将信息写入所述设备清单文件中、以及将所述设备清单文件添加到所述包。

17. 如权利要求 1 所述的方法，其特征在于，构建所述包包括读取一定义依赖关系的组件关系文件。

18. 如权利要求 17 所述的方法，其特征在于，与所述包相关联的所述信息包括一设备清单文件，并且所述方法还包括 ：基于所述依赖关系将信息写入所述设备清单文件中、以及将所述设备清单文件添加到所述包。

19. 如权利要求 1 所述的方法，其特征在于，构建所述包包括读取一定义存在着什么包的包定义文件。

20. 如权利要求 2 所述的方法，其特征在于，构建包包括使至少部分可执行代码文件与语言相关数据文件相分离，使得至少部分所述可执行代码不依赖于任一语言。

21. 在包括具有操作系统文件和其它关联的软件应用程序文件的软件映象的移动计算环境中，一种用于更新所述软件映象中的文件的一部分的方法，其特征在于，所述方法包括 ：

一包生成过程，该包生成过程构建包含文件和设置的包，其中所述包的格式是自描述的 ；以及

一构建清单文件生成过程，该构建清单文件生成过程通过采用操作系统映象的二进制映象构建器文件和一组件 - 包映射文件作为输入来生成指定所述包的文件内容的构建清单文件 ；

一设备清单文件生成过程，该设备清单文件生成过程生成包括包依赖性以及包设置优先级数据的设备清单文件，其中所述设备清单文件存储在所述包内，其中所述设备清单文件进一步包括从所述构建清单文件导出的列表和属性信息，并且所述包设置优先级数据包括描述设置数据相对于其它设置数据的优先级的阴影信息 ；

一包文件创建过程，该包文件创建过程通过将在所述构建清单文件以及所述设备清单文件内描述所述包的内容的信息与所述包相关联从所述包创建一包文件，所述信息包括组件关系信息。

22. 如权利要求 21 所述的方法，其特征在于，所述组件关系信息描述对另一实体的依赖性。

23. 如权利要求 22 所述的方法，其特征在于，所述另一实体是另一包。

24. 如权利要求 21 所述的方法，其特征在于，所述组件关系信息描述与另一实体的阴影关系。

25. 如权利要求 24 所述的方法，其特征在于，所述另一实体是另一包。

26. 如权利要求 21 所述的方法，其特征在于，所述包生成过程确定哪些文件是可执行文件，并在所述可执行文件上执行一重定位过程。

27. 如权利要求 21 所述的方法，其特征在于，所述包生成过程允许在至少某些可执行代码和数据之间存在区别，使得所述可执行代码的至少某些独立于任一语言。

自描述软件映象更新组件

[0001] 相关申请的参照

[0002] 本发明要求2003年12月16日提交的美国临时专利申请序列号60/530,129的优先权,该申请整体结合于此。

[0003] 本申请涉及与其同时提交的以下美国专利申请,这些申请整体结合于此:

[0004] 摘要号4271/307,649,“以故障保险方式向非易失存储应用自定义软件映象更新(Applying Custom Software Image Updates To Non-Volatile Storage in a FailsafeManner)”;

[0005] 摘要号4281/307,650,“确定对安装有效的最大依赖软件更新组(Determingthe Maximal Set of Dependent Software Updates Valid for Installation)”;

[0006] 摘要号4291/307,651,“确保软件更新仅在特定的设备或设备类上安装或运行(Ensuring that a Software Update may be Installed or Run only on a Specific Deviceor Class of Devices)”;以及

[0007] 摘要号4311/307,663,“以存储技术抽象方式在文件内创建文件系统(CreatingFile Systems Within a File In a Storage Technology-Abstracted Manner)”。

技术领域

[0008] 本发明一般涉及计算设备,尤其涉及更新计算设备的非易失存储。

背景技术

[0009] 诸如个人数字助理、当代移动电话和手持式及袖珍计算机等移动计算设备正在变为重要且流行的用户工具。一般而言,它们变得足够小,使得它们极度方便,而消耗较少的电池功率,且在同时变得能够运行更强大的应用程序。

[0010] 在制造这类设备的过程中,嵌入式操作系统映象通常被内建到每一设备的单块映象文件中,并储存在非易失存储中(如,NAND或NOR闪存、硬盘等等)。作为结果,更新这一设备有时是必需或期望的。

[0011] 然而,单块操作系统具有众多缺点,包括为安装更新,需要大量的资源(如,临时存储和带宽)来替换整个单块映象。同时,鉴于各种原因,安装操作系统的某些子集组件是一项困难的任务。需要一种方便操作系统映象的某一子集的更新的机制。

发明内容

[0012] 简言之,本发明针对一种提供安装和更新包的系统和方法,其中,每一包包括为安装目的被同样处理的一组文件的封装,并且其中,包的格式是自描述的,由此方便了映象的仅组件部分的替换。为此,该系统和方法将操作系统特征(包括文件、元数据、配置信息等)映射到包中,作为软件构建过程的一部分。

[0013] 在一个实现中,包逻辑处理特定文件和/或设置在相关的特征之间共享的情况,

其中，用户进而选择来映射到不同的包。给定若干可能的较高级包映射请求，该逻辑一般确保个别文件 / 设置被映射到正确的包。此外，包可任选地传送依赖信息，并由此提供（通过特征级依赖性规范）了一种包获取依赖信息的机制。逻辑分解了特征级以下的冲突和依赖性。

[0014] 在构建过程中，通过采用用于操作系统映象的二进制映象构造器文件和组件一包映射文件作为输入，创建一构建清单文件。该构建清单文件指定了用于特定包的文件内容。审阅这些文件内容，并且在插入到包之前处理任何可执行代码，以使可执行代码在安装时刻重新定位 / 修补到设备上。包生成过程基于构建清单和包定义文件中的信息创建设备清单。

[0015] 操作系统映象的注册表被分解，并基于一类似的算法将其分配到包，并且可类似地分解 XML 文件，并将其分配到特定的包。结果是对每一要构造的包有若干文件，可能包括包定义文件、组件映射文件、组件关系文件、构建清单文件、注册表文件和 XML 设置文件。从这些文件，包生成过程通过从包创建包集合来构造最终的包文件，包括将每一包映射到包定义、读取该包的构建清单文件及从该数据生成包。

[0016] 对于自描述的包，在包装过程中创建一设备清单文件，并将其储存在包本身之内。该设备清单文件在安装过程中使用。包依赖性和阴影（包设置优先级）数据也是包所附的数据的一部分，如通过将其写入设备清单文件中。

[0017] 当结合附图阅读以下详细描述时，可以清楚其它优点，附图中：

附图说明

[0018] 图 1 是一般表示可结合本发明的计算机系统的框图；

[0019] 图 2 所示是依照本发明的一个方面用于构造自描述更新包的各种组件的框图；

[0020] 图 3 所示是依照本发明的一个方面用于从二进制映象文件创建构建清单文件的逻辑的流程图；

[0021] 图 4 所示是依照本发明的一个方面用于从注册表设置创建注册表设置相关文件的逻辑的流程图；

[0022] 图 5 所示是依照本发明的一个方面用于处理将数据从其中写入包内的 XML 格式文件的逻辑的流程图；

[0023] 图 6 所示是依照本发明的一个方面用于生成包的逻辑的流程图；

[0024] 图 7A 和 7B 包括依照本发明的一个方面用于创建描述包的设备清单文件的逻辑的流程图；

[0025] 图 8 所示是依照本发明的一个方面由工具 (relmerge) 执行以插入可执行文件的重定位信息的逻辑的流程图；

[0026] 图 9 所示是依照本发明的一个方面描述包的设备清单文件的格式的框图；

[0027] 图 10A 和 10B 包括依照本发明的一个方面用于构建描述包的设备清单文件的流程图；以及

[0028] 图 11 所示是依照本发明的一个方面包的创建的流程图。

具体实施方式

[0029] 示例性操作环境

[0030] 图 1 示出 了一个这样的手持式计算设备 120 的功能组件，包括处理器 122、存储器 124、显示屏 126 和键盘 128（可以是物理或虚拟键盘，或表示两者）。可存在麦克风 129 以接收音频输入。存储器 124 一般包括易失存储器（如，RAM）和非易失存储器（如，ROM、PCMCIA 卡等等）。操作系统 130 驻留在存储器 124 中，并在处理器 122 上执行，如微软公司的Windows®操作系统或另一操作系统。

[0031] 一个或多个应用程序 132 被加载到存储器 124 中（或在 ROM 中原地执行），并在操作系统 130 上运行。应用程序的示例包括电子邮件程序、调度程序、PIM（个人信息管理）程序、文字处理程序、电子表格程序、因特网浏览器程序等等。手持式个人计算机 120 也可包括加载到存储器 124 中的通知管理器 134，它在处理器 122 上执行。通知管理器 134 处理如来自应用程序 132 的通知请求。同样，如下所述，手持式个人计算机 120 包括适用于将手持式个人计算机 120 连接到网络（包括作出电话呼叫）的网络软件 136（如，硬件驱动程序等）和网络组件 138（如，无线电和天线）。

[0032] 手持式个人计算机 120 具有电源 140，它被实现为一个或多个电池。电源 140 还可包括忽略内置电池或对其重新充电的外部电源，如 AC 适配器或加电对接托架。

[0033] 图 1 所示的示例性手持式个人计算机 120 被示出为具有三种类型的外部通知机制：一个或多个发光二极管 (LED)142 和音频生成器 144。这些设备可直接耦合至电源 140，使得当被激活时，即使手持式个人计算机处理器 122 或其它组件被关闭以保存电池能量时，它们也保留一段由通知机制指示的持续时间。LED 142 较佳地不限时间地持亮，直到用户采取行动。注意，音频生成器 144 的当代版本使用当今手持式个人计算机电池的太多能量，因此它被配置成当系统的剩余部分被关闭时，或者在激活后的一段确定持续时间之后被关闭。

[0034] 注意，尽管示出了基本手持式个人计算机，然而，为实现本发明的目的，实际上能够以可由程序使用的某一方式接收数据通信和处理数据的任何设备都是等效的。

[0035] 自描述软件映象更新组件

[0036] 本发明一般针对安装和 / 或更新储存在基于微软Windows® CE.NET 的便携式设备等小型移动计算设备上的软件，这些设备包括在其中将初始软件或软件更新写入诸如闪存等嵌入式设备的非易失存储器的那些设备。尽管如此，本发明提供了在总体上计算的益处，并由此可应用到其它计算设备和其它类型的存储，包括各种类型的存储器和 / 或其它类型的存储媒质，如硬盘驱动器。为简化目的，术语“闪存”在后文参考设备的可更新存储来使用，尽管可以理解，任一存储机制都是等效的。此外，术语“映象”一般包括初始软件安装映象以及对该映象的随后的软件更新的概念，即使仅更新该映象的一部分。

[0037] 作为背景，诸如Windows® CE 操作系统等当代操作系统是模块化的（组件化的）。然而，包含正确文件和设置的结果映象是单块操作系统映象。为此，在构建时，将特征变量映射到特定的文件和设置，以确定在结果单块操作系统映象中包含了什么内容。执行这一映射的能力利用两种类型的构建时配置文件：二进制映象构建器 (.bib) 和注册表 (.reg) 文件。.bib 文件包含要包括在结果映象中的文件的列表，.reg 文件包含要包括在映象中的注册表（设置）信息的列表。这些文件的内容按特征被组合成集合，并以可在构建时可任选地设置的条件变量包装。当在构建时设置条件特征变量时，.bib 和 .reg 文件的关联内

容被包括在该映象中，并且由此，系统用户能够在粒度特征级上选择结果映象应当包含什么内容。

[0038] 也向条件变量的选择应用较高级逻辑，使得特征级依赖性是自满足的(self-satisfying)。换言之，选择一个特征也将促使所选择的特征所依赖的其它特征被选择。由此，构建系统确保自相容单块操作系统映象从用户的一部分对特征的任一随机选择中得到。然而，如上所述，单块操作系统映象具有关于在所包括的映象中映象的子部分可被个别地更新的缺点。

[0039] 依照本发明的一个方面，映象包括从自包含的、安全实体构建的软件更新。基础更新基元被称为包，其中，一般而言，包是具有同一版本且可作为一个单元来更新的一组文件的封装。本发明提供了一种包格式，作为自描述的、在更新映象时具有显著的改进的格式，以及方便仅替换映象的组件部分的格式。

[0040] 映象从包构建，并包含可应用到存储的可执行代码和数据两者。注意，可执行代码在安装时被定制到嵌入式设备的虚拟地址空间环境；例如，根据基本的闪存技术，某些设备允许可执行代码直接从闪存中运行（原地运行，这意味着代码无法以压缩格式储存），而其它设备需要代码被复制（包括在必要时解压代码）到RAM中来运行。依照本发明的一个方面，映象更新技术使用包将操作系统映象分解成可单独更新的可更新组件，而保留任何跨组件的依赖性。

[0041] 依照本发明的一个方面，提供了一种将操作系统特征（包括文件、元数据、配置信息等等）映射到包作为软件构建过程的一部分的系统和方法。包可用于初始设备安装，也可用于更新。如后文所描述的，软件更新包可以是各种形式，例如，某些可仅包含对前一更新的改变（增量），而其它可包含完全替换其它文件的文件。一个其它类型的包可包含其它包。

[0042] 本发明描述的包概念是更新过程的组件部分。用于将操作系统特征（一般是映射到具体的文件、元数据、配置信息等的抽象概念）映射到包的过程为用户（如，映象的提供商）提供了易于使用的优点，包括作为必须标识操作系统映象的最低级组件的替代，用户能够参考为操作系统映象的特定方面描述关联的文件、元数据、配置信息等的完整组的较高级句柄。注意，如本发明所使用的，属于“特征”和“组件”通常可互换使用。通过参考特征句柄，用户能够获得通过抽象管理包装复杂性方面的优点。例如，与具体地映射可执行模块、动态链接库(DLL)、资源/数据文件、注册表信息等来浏览组件软件（如，Internet Explorer）并将每一部分个别地映射到包中不同，本发明令用户能够用单个“Internet Explorer”句柄参考这一关联的信息，由此在特征级将其映射到包。

[0043] 本发明也提供了处理特定文件/设置在相关的特征之间共享的情况的包装逻辑，其中，用户进而选择来映射到不同的包。给定若干可能的较高级包映射请求，各种算法（后文描述）中的逻辑一般确保个别文件/设置被映射到正确的包。在逻辑无法确定行动的正确过程的情况下，向用户提供消息以指示需要用户干预来解决的任何问题。

[0044] 此外，包可任选地传递依赖信息。例如，当一个包的内容依赖于另一包的内容时，在构建过程中捕捉这一关系，并将其编码到包内，用于稍后在安装过程中的分析。本发明也提供了一种包用于获取依赖信息的机制（通过特征级依赖性规范）。

[0045] 依照本发明的一个方面，特征选择可被映射到（一个或多个）包的数组，导致特定

的文件和设置被映射到适当的包。为正确完成这一过程，逻辑分解了特征级之下的冲突和依赖性。例如，如果两个特征逻辑上涉及同一具体文件，并且这两个特征被映射到不同的包，则逻辑确定该共享的文件要被放入哪一包中。

[0046] 在一个实现中，包文件在构建时由三个不同的文件定义：包定义文件 (pkd)、组件映射文件 (cpm) 和组件关系文件 (crf)。pkd 文件定义包内容的全局属性。pkd 文件是在构建过程中对照以下 XSD 来确认的 XML 文件：

[0047]

```
<? xml version =" 1.0" encoding =" utf-8" ? >
<xs:schema targetNamespace =" http://tempuri.org/Packages.xsd" elementFormDefault =" qualified"
    xmlns =" http://tempuri.org/Packages.xsd" xmlns:mstns =" http://tempuri.org/Packages.xsd"
xmlns:xs =" http://www.w3.org/2001/XMLSchema"
    xmlns:xsi =" http://www.w3.org/2001/XMLSchema-instance" >
    <xs:element name =" PackageCollection" >
        <xs:complexType>
            <xs:sequence>
                <xs:element name =" Package" minOccurs =" 1" maxOccurs =" unbounded" >
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name =" Name" type =" xs:string" minOccurs =" 1"
maxOccurs =" 1" ></xs:element>
                            <xs:element name =" Guid" type =" xs:string" minOccurs =" 1"
maxOccurs =" 1" ></xs:element>
                            <xs:element name" Version" type =" xs:string" minOccurs =" 1"
maxOccurs =" 1" ></xs:element>
                            <xs:element name =" LocBuddy" type =" xs:string" minOccurs =" 1"
maxOccurs =" 1" ></xs:element>
                            <xs:element name =" CertPaths" mlnOccurs =" 0" maxOccurs =" unbounded" >
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element name =" path" type =" xs:string" ></xs:element>
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

[0048] .pkd 文件的一个示例如下：

[0049]

```
<? xml version =" 1.0" encoding =" utf-8" ? >
<Package xmlns:xsd =" http://ww.w3.org/2001/XMLSchema"
xmlns:xsi =" http://www.w3.org/2001/XMLSchema-instance" >
    <Name>XSC1BD_OEM</Name>
    <Guid>0b3bc94d-e4c4-4509-930c-8131dde185c8</Guid>
    <Version>000000</Version>
    <LocBuddy>00000000-0000-0000-0000-000000000000</LocBuddy>
</Package>
```

[0050] 注意该定义文件中全局唯一 ID(GUID) 和包名字的存在。GUID 为包提供了可被引用以将包互相区分的唯一名字。包名字服务来提供当将操作系统特征 / 组件映射到包时所使用的句柄。

[0051] 在一个示例实现中,cpm 文件是包含组件 - 包映射信息的以逗号分隔值的文件。它是以下形式 :

[0052] < 组件变量 >,< 包名字 >

[0053] 或

[0054] < 组件文件名 >,< 包名字 >

[0055] 以下提供了某些组件 - 包映射信息示例 :

[0056]

```
nk.exe,XSC1BD_OEM
kd.dll,XSC1BD_OEM
CE_MODULES_FILESYS,XSC1BD_OEM
CE_MODULES_BINFS,XSC1BD_OEM
CE_MODULES_FSDMGR,XSC1BD_OEM
FILESYS_FSYSRAM,XSC1BD_OEM
FILESYS_FSREG,XSC1BD_OEM
fsdmgr.dll,XSC1BD_OEM
imgfs.dll,XSC1BD_OEM
CE_MODULES_STRATAD,XSC1BD_OEM
CE_MODULES_MSPART,XSC1BD_OEM
CE_MODULES_CEDDK,XSC1BD_OEM
```

[0057] 组件关系文件 (crf) 表示组件之间的依赖性和阴影关系 (阴影指设置的优先顺序)。组件关系文件是以下形式 :

[0058] < 组件变量 >DEPENDSON< 组件变量 >

[0059] 或

[0060] < 组件变量 >SHADOWS< 组件变量 >

[0061] 它的一个示例为 :

[0062] APPS_MODULES_OMADMCLIENT SHADOWS APPS_MODULES_MMS

[0063] 下表总结了本发明所描述的某些术语和文件类型的通用、非限制定义：

[0064] 术语

[0065]

术语	通用定义
模块	单个可执行文件（.EXE、.DLL 等）
设置	配置信息的集合，可包含注册表设置、文件系统初始化指令（.DAT 文件）、数据库初始化和供应 XML
组件	组成特征单元的模块、文件（非模块文件，包括模板，字体等）
	和设置的集合。组件通常与系统生成 MODULES_ 标签相关联。
包	被签署且被包装用于分发的组件的集合
清单（包、构建和设备）	描述包内容的文件。注意，在构建环境中，存在包含描述包中每一文件的名字的条目的的包清单文件（具有 .BIB 扩展名）；不同的清单文件 - 设备侧清单文件（具有 .DSM 扩展名）是描述关于包的信息的完整组的二进制文件（后文参考图 9 描述）。如参考图 3 所描述的，构建清单文件从包清单文件创建，并由生成设备清单文件（以及包的剩余部分）的包生成过程使用。
阴影排序工具	处理组件关系文件并生成包阴影文件的构建工具。

[0066] 文件类型

[0067]

扩展名	文件类型	一般描述
.bib	二进制映象构建器	包含应当包括在结果映象中的文件的列表
.reg	注册表文件	包含要包括在映象中的注册表（设置）信息的列表的文件
.pkd.xml	包定义文件	构建树中定义存在哪些包（包名字、GUID、版本、本地合伙者 (loc buddy) 的 XML 文件
.cpm.csv	组件 - 包映射文件	构建树中将 MODULES 标签和文件映射到包的 CSV 文件

扩展名	文件类型	一般描述
.crf.csv	组件关系文件	构建树中定义 MODULES 标签之间的关系（阴影和依赖性）的文本文件
bsm.xml	构建侧清单文件	.bib(二进制映象构建器）文件的 XML 文件转换
.psf	包阴影文件	构建环境中列出必须由命名的包作出阴影的包的中间文件（每一包中有一个,如＜包名＞.psf)
.dsm	设备侧清单文件	设备上描述包（包中的文件、名字、GUID、签名、阴影、依赖性、CRC、根证书等）的文件（每一包中一个）
.pkg.cab	规范包文件	包含包中所有文件的完整文件的包的完整版本
.pku.cab	更新包文件	方便将设备从单个包的特定版本更新到同一包的不同版本的文件。该文件可包含个别文件的二进
		制差或完整文件,它们的任何一个都将更新包的大小最小化。
.pks.cab	超级包文件	包含更新包和／或规范包的集合的文件。

[0068] 转向附图的图 2,作为整体包生成过程的一部分,在构建过程中,通过采用用于操作系统映象的二进制映象构建器文件 204(.bib 文件,也称为包清单文件）和组件－包映射文件 206 作为输入,创建构建清单文件 202。如上所述,构建清单文件 202 指定了用于特定包的文件内容。

[0069] 图 3 一般示出了一个示例构建清单文件创建过程 208,通过该过程,从二进制映象构建器文件 204 创建构建清单文件 202。如从图 3 中可见到的,一般而言,分析二进制映象构建器文件 204 中的每一行（步骤 302、306 和 308）;被分析为具有有效标签的 .bib 文件条目（在原地执行表中查找（步骤 310 和 312)）的行可如所需要地被压缩（步骤 318),并被写入构建清单文件 202 中（步骤 320)。如果行被忽略（即,当决定框 314 为真时),行从二进制映象构建器文件中移除（步骤 316)。

[0070] 结果的构建清单文件 202 是用以下 XSD 确认的 XML 文件：

[0071]

```
<? xml version =" 1.0" encoding =" utf-8" ? >
<xs:schema targetNamespace =" http://tempuri.org/ManifestCollection.xsd"
elementFormDefault =" qualified"
    xmlns =" http://tempuri.org/ManifestCollection.xsd"
xmlns:mstns =" http://tempuri.org/ManifestCollection.xsd"
    xmlns:xs =" http://www.w3.org/2001/XMLSchema" >
    <xs:element name =" ManifestCollection" >
        <xs:complexType>
            <xs:sequence>
                <xs:element name =" BibFileEntry" minOccurs =" 1"
maxOccurs =" unbounded" >
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name =" DeviceName" type =" xs:string"
minOccurs =" 1" maxOccurs =" 1" />
                            <xs:element name =" ReleaseName" type =" xs:string"
minOccurs =" 1" maxOccurs =" 1" />
                            <xs:element name =" Region" type =" xs:string"
minOccurs =" 1" maxOccurs =" 1" />
                            <xs:element name =" Section" type =" xs:string"
minOccurs =" 1" maxOccurs =" 1" />
                             <xs:element name =" Attribs" type =" xs:string"
minOccurs =" 0" maxOccurs =" 1" />
```

```
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

[0072] 类似于创建构建清单文件 202 的方式，分解操作系统映象的注册表（分解成有时候被称为 RGU 文件的文件）（过程 400，步骤 402），并基于类似的算法将其分配到包。这一过程 210（图 2）的一个示例在图 4 中描述，其中，与有效标签（步骤 414）相关联的注册表设置 212（图 2），在适当时包括其注册表键名（曲在步骤 406、408、410 和 412 中确定的），被写入包 RGU 文件中（步骤 422）。注意，只要所处理的包改变，注册表键名就被写入该包的 RGU 文件中（步骤 418 和 420）。如果注册表键名被忽略（即，当决定框 416 为真时），通过不写该名而从注册表中移除注册表键名（步骤 424）。

[0073] 此外，可分解 XML 文件 218（图 2，如包含其它设置），并将其分配到特定的包（例如，在步骤 502、504 以及 506 中）。用于完成这一过程的一个示例过程 220 在图 5 中描述，其中，子节点的有效标签（在步骤 508 评估）被分配到包的节点（步骤 514）。如果 XML 文件被忽略（即，当决定框 510 为真时），通过不写该 XML 文件而被移除（步骤 516）。

[0074] 包创建过程的这一中间步骤的结果是对要构造的每一包有多个文件，在图 2 中示出，包括：

[0075]

包定义文件 (pkd)224； 组件映射文件 (cpm)206； 组件关系文件 (crf)226 可任选； 构建清单文件 202 可任选； 注册表文件 (rgu) 可任选；以及 XML 设置文件 222 可任选

[0076] 从这些文件，(包括如通过 XSD 203 和 XSD 225 的确认）包生成过程 230 一般依照图 6 所示的流程图构造最终的包文件 232。如图 6 中一般所表示的，在某些检查和确认（步骤 600-610）之后，从包创建包集合（步骤 612)，包括将在包循环 624 内的每一包 616 映射到包定义（其中包集合不具有 0 的长度（即，当决定框 614 为否时））、读取该包的构建清单文件以及从该数据生成包（步骤 618-622)。

[0077] 将构建清单文件转化成最终包文件列表形式并用 relmerge 工具 250（如后文参考图 8 所描述的将重定位信息插入到可执行文件中的工具）处理每一可执行文件的过程一般在图 7A 和 7B 中描述。如图 7A 和 7B 中通过步骤 700-714 一般描述的，创建一伪指令文件，并查找和分析构建清单文件，(假定不出现错误)。如果分析成功（步骤 716)，则在步骤 720 创建设备清单文件，并且过程继续到图 7B 的步骤 726，这确保了设备清单对象被正确地创建。如果在步骤 716 分析没有成功，返回分析错误 718。

[0078] 通过步骤 732 和 756，处理构建清单文件中列出的每一文件 730。为此，在步骤 734，找到该文件，并在步骤 738 确定它是否可执行。如果没有对象被创建，返回错误消息 728。如果文件没有被找到，返回错误消息 736。如果不是，则照原样将该文件复制到一临时构建目录（步骤 742)，否则，该文件需要由令可执行代码能够在安装时重新定位 / 修补到设备上（如果尚未被处理，如通过步骤 740 所测试的）的工具 250（图 2，如名为 relmerge.exe) 来处理。如果文件需要 relmerge 处理，如后文参考图 8 所描述的，则在步骤 744 调用该工具，并且如果它成功地执行（在步骤 746 被确定)，则将文件名添加到设备清单（步骤 750)。如果文件没有成功地执行，返回错误 748。如果在决定框 752 确定文件名没有被添加成功，返回错误 754。

[0079] 由此，如上文参考图 7A 和 7B 所描述的，在插入到包中以允许可执行代码在安装时能够重新定位 / 修补到设备上之前，审阅构建清单文件中列出的包的文件内容，并处理任何可执行代码。为将构建清单文件转化成最终包文件列表形式，用 relmerge 工具 250 处理每一可执行文件，该工具压缩已在文件中的重定位信息（步骤 816)，并可任选地（如果提供了 .REL 文件）提供允许操作系统将代码和数据段分隔成不连续的存储器区域的更详细的重定位信息。

[0080] relmerge 工具操作一般在图 8 的流程图中描述。如步骤 802 所表示的，作出输入文件的副本，因为输入文件将通过运行该工具而被修改。由此，relmerge 工具 250 作出该输入文件的副本，并从该副本工作，当程序退出时，该副本被删除。

[0081] 在步骤 804，移除签名，因为否则签名将导致处理的稍后部分（空格计算检查，后文描述）失败。注意，由于工具 250 将输出完全不同的文件，因此签名在任何情况下都与输

出文件无关。注意,在可移植的可执行文件中,(.EXE 和 .DLL 文件的文件格式),签名储存在文件中任何分段之外的文件末端,其中,每一分段对应于一单元,数据与该单元一起由加载器加载到存储器中。每一分段具有一头部,被称为 032 头部,或用 IMAGE_SECTION_HEADER 结构来标识。如由步骤 806 一般表示的,为方便处理该文件,工具 250 将 PE 文件的 PE 文件头部及其分段解析成合适的内部数据结构。IMAGE_SECTION_HEADER 结构表示映象分段头部格式(关于 IMAGE_SECTION_HEADER 的另外细节可在 msdn.microsoft.com 上找到):

[0082]

```
typedef struct_IMAGE_SECTION_HEADER{
    BYTE Name[IMAGE_SIZEOF_SHORT_NAME];
    union{
        DWORD PhysicalAddress1
        DWORD VirtualSize;
     }Misc;
     DWORD VirtualAddress;
     DWORD SizeOfRawData;
     DWORD PointerToRawData;
     DWORD PointerToRelocations;
     DWORD PointerToLinenumbers1
     WORD NumberOfRelocations;
     WORD NumberOfLinenumbers;
     DWORD Characteristics;
}IMAGE_SECTION_HEADER,
*PIMAGE_SECTION_HEADER;
```

[0083] 由于该文件将用新头部重新布局,并移除填充,因此执行空格计算检查(步骤 808)来核实在文件中没有未包括在文件的分段中的信息。注意,存在数据储存在分段之外的 .EXE 或 .DLL 文件中的某些情况,包括文件签名和 Codeveiw 调试目录条目,(由工具 250 处理)。存在将数据储存在分段之外的其它应用程序,如将要提取的数据储存在 .EXE 本身之后的自提取可执行文件。压缩的数据不储存在分段中,否则加载器将试图将压缩的数据加载的存储器中,这并不是自提取可执行文件应当操作的。该工具不支持这些实例。

[0084] 空格计算由 CSpaceAccounting 类的实例来实现,它维护 SpackBlock(空格块)结构的数组,其每一个都占有文件中的数据块。为实现空格计算,文件中可被占有的区域作为个别的块被添加到空格计算中。这对文件头部(包括 E32 和 032 头部)以及文件中的每一分段完成。为容纳 CodeView 调试条目,它们的每一个也作为单独的块添加。然后按照其在文件内的偏移对这些块排序。在新的顺序中相邻的块(如,块 2 在紧靠块 1 的最后一个字节之后的字节开始)被合并。在过程的最后,如果文件中的所有空格可被占用,则该列表应当包含一个且仅一个块,它在偏移 0 处开始,并且其长度等于整个文件的长度。如果该条件为真,则测试通过。如果不是,则报告错误,并且该工具退出。

[0085] 如果空间计算检查通过,则工具 250 在指定输入文件的同一目录中搜索 .REL 文

件，如由步骤 810 所表示的。如果存在，则它通过步骤 812 来处理，否则，通过步骤 814 来处理 PE 文件的重定位分段。更具体地，重定位分析工具可分析两种不同类型的重定位信息，即 .REL 文件中的重定位信息，它包含目标分段信息，以及 PE 文件的 .reloc 分段中的重定位信息，它不包含目标分段信息。为此，重定位分析具有对应于分析 .REL 文件或分析 PE 文件的内部重定位的两个进入点；在当前的实现中，调用一个或另一个是合法的，但不能同时调用两个。

[0086] 为储存重定位，维护 CRelocData 类的二维数组，其中，该数组的两个维度是所述重定位的源分段和目标分段。重定位的源分段是重定位所定位的分段，其中，重定位是当文件被修补以在特定的地址上加载时更新文件中特定数据片段的指令；源分段标识了数据片段所出现的分段。源分段通过查找重定位的相对虚拟地址并将其与文件中的分段的相对虚拟地址范围相比较来推算。

[0087] 重定位的目标分段标识包含重定位所指向的数据片段的分段。检查技术并不总能起作用，因为已知优化器以重定位看似指向其它分段的这样一种方式来优化代码，如，它优化出加和减，并将其放入引用而非代码中。为此，无法从数据中推算目标分段。两个重定位格式 (.REL 或 .reloc) 之间的差异是重定位的目标分段是否已知。.REF 文件明确地标识了目标分段，而 .reloc 分段未标识。

[0088] 结果，对于仅具有 .reloc 分段中的重定位信息的文件，需要将整个文件一起重定位。由于没有目标分段信息，无法将两个分段分离，并按不同的量来重定位它们。由此，该工具需要跟踪每一重定位的源分段和目标分段，并跟踪目标分段是否有效。

[0089] 为达到这一目的，该过程维护 CRelocData 类的二维数组，并且每一 CRelocData 类构建其自己的重定位数据流。该数组在两个维度上都是固定的大小，这意味着该工具只能处理具有某些最大 (如，16) 分段的 PE 文件。采用这一限制，持久保持重定位的数据格式对源分段和目标分段的每一个储存 8 位，在数据格式内为 256 个分段留下了可能性。两个其它的函数 (CalculateRelocSize(计算重定位大小) 和 WriteRelocationsToFile(将重定位写到文件)) 然后通过对具有至少一个重定位的源分段和目标分段的每一个组合写出块头部来组合这些个别的流。

[0090] 重定位编码在 CRelocData 类中实现。该类采用重定位地址流 (作为对 CRelocData::AddReloc 方法的个别调用)，并创建表示对这些重定位进行编码所需的命令的字节流。该字节流可在稍后被检索。为实现这一过程，该类有效地留下一个命令，总是以“Single(单个)”命令开始，表示单个重定位。然后，当新的重定位到达时 (到 AddReloc 方法)，分析它们来看是否可使用前一命令和新的位置形成模式。如果前一命令是“Single”命令，则如果新的重定位地址是 DWORD 对齐的，仅可通过将其从“Single”命令转换成“Pattern(模式)”命令来扩展前一命令，并且前一命令处于模式命令的最大跳过范围内 (3 个 DWORD)。如果前一命令是模式命令，则该模式已具有建立的形式，并且可推算出该模式中的下一元素。如果新地址恰巧与该模式中的下一元素相匹配，则将该模式扩展一个周期。否则，开始一个新的 Single 命令。

[0091] 转向图 8 的整体流程，由于文件的分段在文件的头部之后线性地出现，因此需要重新布局整个文件以容纳新的头部格式并移除分段间的所有填充。为此，如由步骤 818 所示的，移除旧的 .RELOC 分段 (如果存在的话)，然后在末端添加新的 .CRELOC 分段，如果有

任何重定位需要添加。使用步骤818完成的文件布局，最后在步骤820创建输出文件，将每一分段的头部以及数据写入输出文件。

[0092] 在步骤822，relmerge.exe现在进行测试来看目标输出文件是否为nk.exe。如果输出文件是nk.exe，则relmerge工具250处理两个文件的内容，以写出pTOC信息和RomExt信息（步骤824），（后文描述）。relmerge工具250考虑输出文件，因为nk.exe通过基于调试设置复制不同的文件来创建。所处理的第一个文件是config.bsm.xml。该文件由MakePkg.exe在映象更新处理过程中产生。它包含名字的文本表示以及对系统内核中的FIXUPVARS的期望值。分析该文件的内容，并将其储存用于稍后的使用。第二个文件是输入文件的映射文件。该文件由ProcessFixupVars处理。它采用源文件路径、复制它并用.map替换.exe，并试图打开映射文件（映射文件是包含关于PE文件中的函数和变量的物理和虚拟地址的大量信息的文本文件）。如果成功地打开了映射文件，则分析第一行以检索映射文件的时间戳。然后将时间戳与PI文件中的时间戳进行比较。如果它们不同，则生成警告，并且映射文件处理结束。如果时间戳匹配，则读取该文件的每一行，并使用正则表达式串来查找每一FIXUPVAR。如果找到匹配，则从映射文件中取出地址信息，并将其用于将新变量值（来自config.bsm.xml）在正确的位置写入源文件中。在同一时刻，ProcessFixupVars也查找pTOC（其中，TOC表示目录）和RomExt变量用于稍后的处理。当找到pTOC时，创建新的.creloc头部，将bSrcSection设为从映射文件检索的分段（从基于1调整到基于0），将bDstSection设为254，并将长度设为4。将该头部写入目标文件中，其后为4个字节的RVA+基础地址信息。对于RomExt，头部是相同的，除bSrcSection被设为253之外。

[0093] 使用pTOC/RomExt分析操作，因为操作系统内核需要关于ROM中的文件的信息，它通过pTOC变量来提供。该变量需要由DiskImage工具和运行在设备上的更新应用程序来更新。关于该变量的信息只能通过在系统构建的编译和链接阶段创建的.MAP文件来检索。分析该文件以检索这一信息。

[0094] 某些运行时工具需要访问内核数据结构内声明的变量。该变量被命名为RomExt。该变量需要由DiskImage工具和运行在设备上的更新应用程序来更新。关于该变量的信息只能通过在系统构建的编译和链接阶段创建的.MAP文件来检索，并且分析该文件以检索这一信息。

[0095] 依照本发明的一个方面，对于自描述的包，在包装过程中创建设备清单文件260（图2，在图9中更具体地示出），并将其储存在包本身中。该设备清单文件260在安装过程中使用。包含在该设备清单文件中的格式和信息在图9中示出，并也用以下结构定义来描述：

[0096]

```
typedef struct_DeviceManifestHeader
{
    const DWORD dwStructSize;  //用于指定版本的该结构的大小(以字节表示)
    const DWORD dwPackageVersion;  // 该包的版本
    const DWORD dwPrevPkgVersion;  // 该包所更新的包的版本，(0) 代表规范
    const DWORD dwPackageFlags;    // 包专用标识符
```

```
    const DWORD dwProcessorID;      //什么处理器(匹配winnt.h中的定义)
    const DWORD dwOSVersion;       //构建到操作系统的哪一版本
    const DWORD dwPlatformID;      //目标平台是什么
    const DWORD dwNameLength;      //以字节表示的文件名长度
    const DWORD dwNameOffset;      //对包的友好名的偏移
    const DWORD dwDependentCount;      //依赖GUID列表中有多少条目
    const DWORD dwDependentOffset;     //从文件的前端开始有多少字节是依赖
                                       //GUID结构
    const DWORD dwShadowCount;       //阴影GUID列表中有多少条目
    const DWORD dwShadowOffset;      //从文件的前端开始有多少字节是附有
                                        //阴影包的GUID的数组
    const DWORD dwFileCount;        //该清单中列出了多少文件
    const DWORD dwFileListOffset; //从文件的前端开始到第一个文件条目
                                     //有多少字节
    const DWORD cbCERTData;        //数字证书数据的字节数
    const DWORD dwCERTDataOffset;        //从文件的前端开始到证书数据
                                          //有多少字节
    const GUID guidPackage;              //该包的GUID
}DeviceManifestHeader*PDeviceManifestHeader;
typedef struct_DependentEntry{
    const DWORD size;
    const DWORD version;
    const GUID guid;
}DependentEntry,*PDependentEntry;
typedef struct_FileEntry{
     const DWORD dwNameLength;
     const DWORD dwFlags;
     const DWORD dwOffset;
```

```
    const DWORD dwBase;    //该文件最初与其链接的基地址
    const DWORD dwFileSize;  //整个文件的大小,对更新包不准确。
}FILEENTRY,*PFILEENTRY;
```

[0097] 在一个实现中,如上所述,设备清单文件的内容从pkd和经处理的crf文件(一旦被处理,称为psf文件280)信息,以及构建清单文件(如,对于文件列表和属性信息)导出。crf文件描述了组件级依赖性,并被处理成描述包级依赖性的形式。

[0098] 此外,组件设置(配置信息)可彼此作出阴影(换言之,在系统上存在两个相关的设置,而只有一个会胜出的情况下,存在一种优先顺序)。组件级的依赖性和阴影信息通过阴影排序工具被转化成包级关系。

[0099] 阴影排序工具依照组件关系文件226(图2)对为其它包作出阴影的每一包生成阴影文件(包名.psf)。包的.psf文件列出(每行一个)附有阴影的包的GUID。阴影排序工具的输入是合并的组件-包映射文件(.cpm)、合并的组件关系文件(.crf)和合并的包定义文件(.pkd)。输出文件是文本文件,包括以下形式的格式化的文本行:

[0100] <附有阴影的包的GUID>,<附有阴影的包名>,<规则>(其中,规则=SHADOWS或DEPENDSON)。

[0101] 一个示例行如下:

[0102] 273cd4bf-d4ef-4771-b2ce-6fe2fa2b2666,SMARTFON,SHADOWS

[0103] 根据这一信息,包生成器创建设备清单文件(如在步骤1000-1008中描述的),如图2中通过设备清单文件创建过程262所示的,并在图10A和10B中一般描述。为此,将.psf文件中的每一GUID添加到设备清单文件,如通过图10A的步骤1010一般表示的。注意,在一个实现中,这包括确定该GUID是依赖性GUID还是阴影GUID,以及在设备清单对象上调用适当的添加函数。当添加了GUID之后(决定框1012),设备清单对象将设备清单文件写到临时目录中,如通过10B的步骤1030一般表示的。如GUID循环1020表示的那样过程重复。如果没有GUID被添加,返回错误消息1014。如果文件没有被写(如在决定框1032中指示的),返回错误消息1034。

[0104] 如图11所描述的,使用诸如在步骤1102-1110中描述的CAB文件API(CABAPI)等包文件创建过程创建包文件232(参见包文件创建过程270),包括伪指令文件中描述的已处理文件-rgu文件214、xml文件218和设备清单文件260,以及其它包内容282,。CABAPI预期提供对机柜(Cabinet)文件的内容的访问,机柜文件作为映象更新过程的一部分用作映象更新中所涉及的文件的传输机制。

[0105] 为创建包,PackageDefinition(包定义)类负责管理包的整体创建。作为创建过程的一部分,PackageDefinition对象在由“_FLATRELEASEDIR”环境变量指定的目录下创建一新的子目录。该目录名是包名,并向其添加串“_PACKAGE_FILES”,如,给定名为“LANG”的包,将创建名为“LANG_PACKAGE_FILES”的目录。当在对象上调用方法SetDirectoryBase(设置目录基础)时创建。当PackageDefinition对象创建包时,所得的包文件的名字是具有“.PKG”扩展名的包的友好名。包文件服从微软对CAB版本1.3的CAB文件规范。

[0106] PackageDefinition类提供了以下公用方法:

[0107] ● PackageDefinition(XipPackage pkg)-基于XipPackage对象创建包的构造函数。

[0108] ● SetDirectoryBase(string path)-在指定的目录下创建新的子目录。

[0109] ● Validate()-确定包是否具有两个需要的字段,即名字和GUID。

[0110] ● ReadManifest()-促使与该包关联的BuildManifest分析适当的构建清单文件。

[0111] ● MakePackage()-为该包定义创建实际的包文件。

[0112] 此外,当用各种文件工作以包括在包内时,应当注意,可在单独的文件中排列可执行代码和数据。其一个显著的优点是方便多语言系统,其中,特征的语言专用部分被放入对语言专用的单独的包中。这创建了一种系统,其中,包含系统的可执行代码的包与包含系统

的语言专用组件的包相互分离。结果，对特征的可执行代码的补丁可被应用到任一设备，而与安装在该设备上的语言的任一组合无关。

[0113] 更具体地，通过构造，当构建特征时，可执行代码（和语言不相关数据）被分离成一组文件，语言相关数据（和可能的代码）被分离成另一组文件。这些文件被标记为特征的一部分，但是语言相关数据文件被进一步标记为语言相关的。该系统然后将这些文件移至单独的包中（如，由 pkd 文件中的 LocBuddy 标签描述）。

[0114] 作为示例，基于电话的特征可以在语言不相关的库（如，tpcutil.dll）中。对电话特征的语言相关资源被构建到另一资源 dll 中，如名为 tapres.dll，它对每一语言被进一步本地化，例如，它可变成 tapres.dll.0409.mui（对于美国英语）、tapres.dll.0407.mui（对于德语）等等。这些文件被标记为电话特征的一部分，但是语言专用文件用合适的文件名被进一步标记为语言专用的。例如，文件名可通过将合适的语言标签代入名字中基于位置的变量中来构造，如由 tapres.dll.%LOCID%.mui 所表示的。然后，对它所支持的每一语言处理该文件，并且生成多个 LANGPHONE(locbudy) 包，例如，LANGPHONE_0409（对于美国英语）、LANGPHONE_0407（对于德语）等等。结果，系统稍后可更新 LANGPHONE 区域（通过不同的 LANGPHONE_xxxx 包），如修补 tpcutil.dll 中的错误，而与上安装了哪一语言无关。

[0115] 另外，系统的灵活性使得在需要的时候允许语言专用可执行代码。例如，不同的语言和场所具有用于捕捉该语言中的文本的不同的输入法编辑器 (IME)。这些 IME 需要对每一语言特殊的代码，并由此被置于 LANG 区域之一之内。

[0116] 最后，注意，可执行柜 (cabinet) 核实。柜核实模块从最终的包文件 232（柜文件）中提取设备清单文件 260，并对照设备清单文件 260 的内容核实柜文件。

[0117] 如从以上详细描述可以见到的，提供了方便更新操作系统映象的某一子集的各种机制。提供了一种自描述包文件，包括令映象的更新明了且正确的依赖性、阴影和其它特征。

[0118] 尽管本发明易受各种修改和替换构造的影响，然而在附图中示出了某些说明的实施例并在上文详细描述了它们。然而应当理解，这并非将本发明限于所揭示的具体形式，而是相反，本发明覆盖落入本发明的精神和范围之内的所有修改、替换构造和等效技术方案。

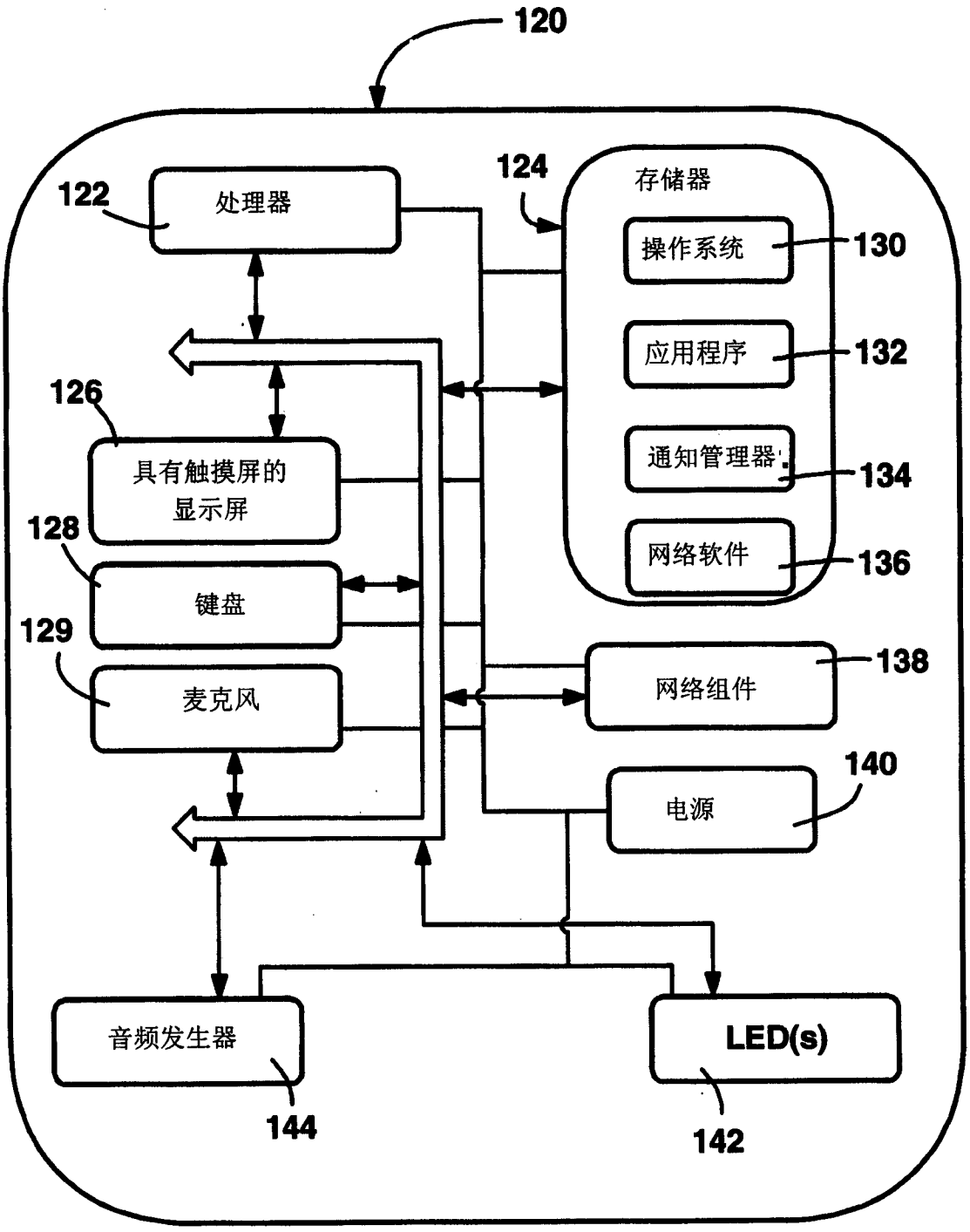

120
122
处理器
124
存储器
操作系统
130
应用程序
132
126
具有触摸屏的
显示屏
通知管理器
134
128
键盘
网络软件
136
129
麦克风
网络组件
138
140
电源
音频发生器
LED(s)
144
142

图 1

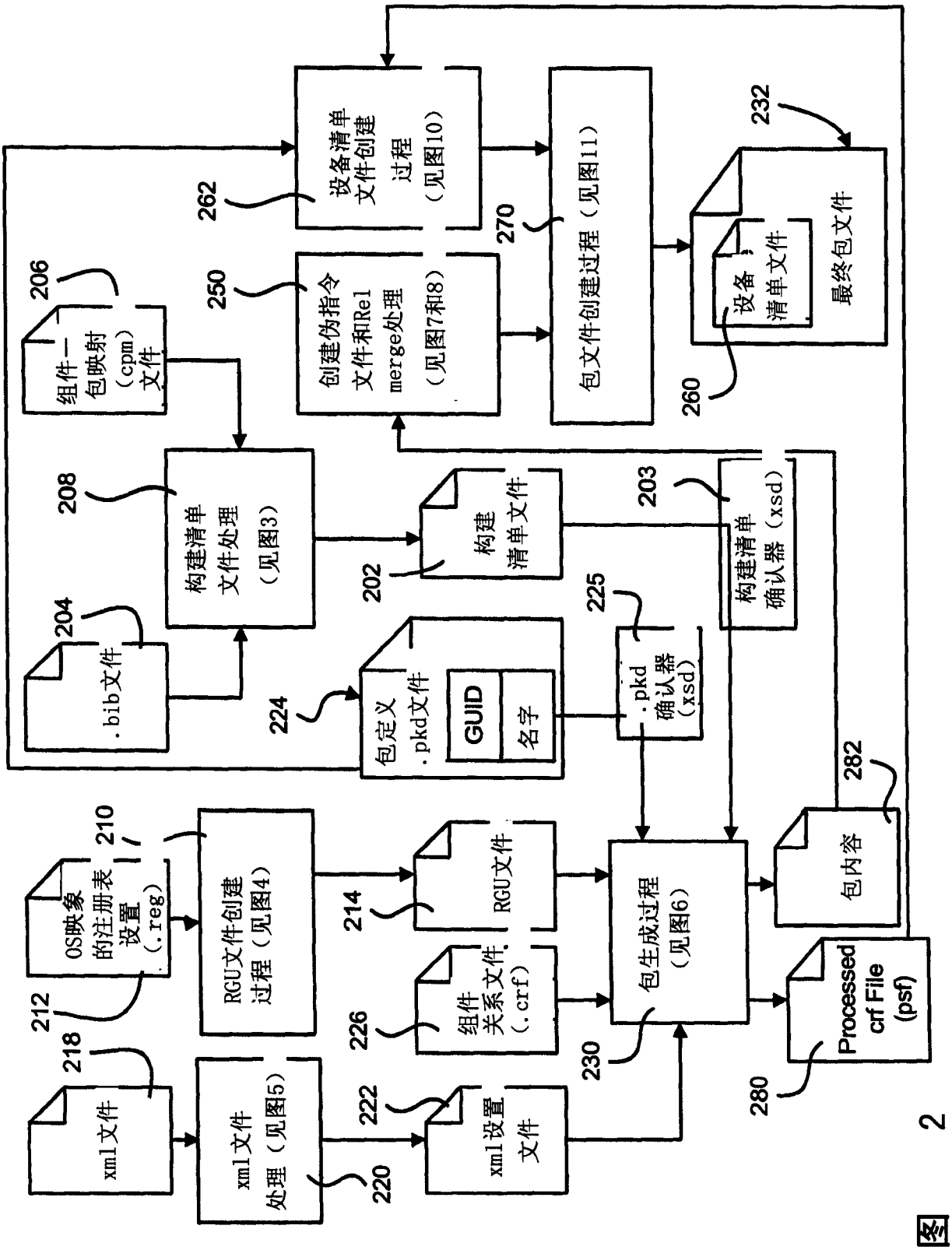

206
组件—包映射（cpm）文件
250
创建伪指令文件和Rel merge处理（见图7和8）
262
设备清单文件创建过程（见图10）
270
包文件创建过程（见图11）
232
设备清单文件
最终包文件
260
208
构建清单文件处理（见图3）
202
构建清单文件
203
构建清单确认器（xsd）
204
.bib文件
224
包定义.pkd文件
GUID
名字
225
.pkd确认器（xsd）
210
OS映象的注册表设置（.reg）
212
RGU文件创建过程（见图4）
214
RGU文件
282
包内容
218
xml文件
220
xml文件处理（见图5）
222
xml设置文件
226
组件关系文件（.crf）
230
包生成过程（见图6）
280
Processed crf File (psf)

图 2

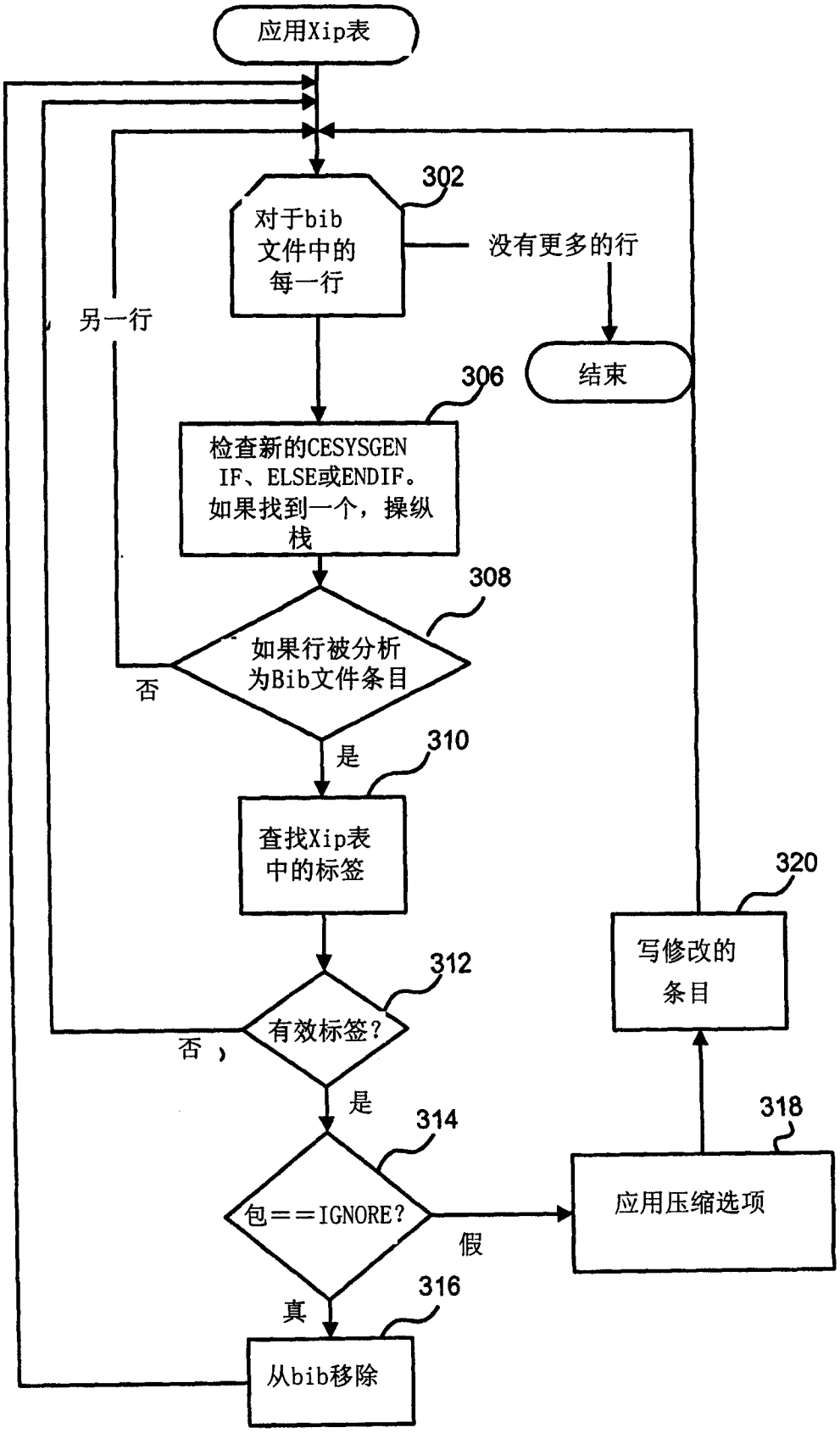

应用Xip表
302
对于bib
文件中的
每一行
没有更多的行
另一行
结束
306
检查新的CESYSGEN
IF、ELSE或ENDIF。
如果找到一个，操纵
栈
308
如果行被分析
为Bib文件条目
否
是
310
查找Xip表
中的标签
320
写修改的
条目
312
有效标签?
否
是
314
318
包==IGNORE?
应用压缩选项
假
真
316
从bib移除

图 3

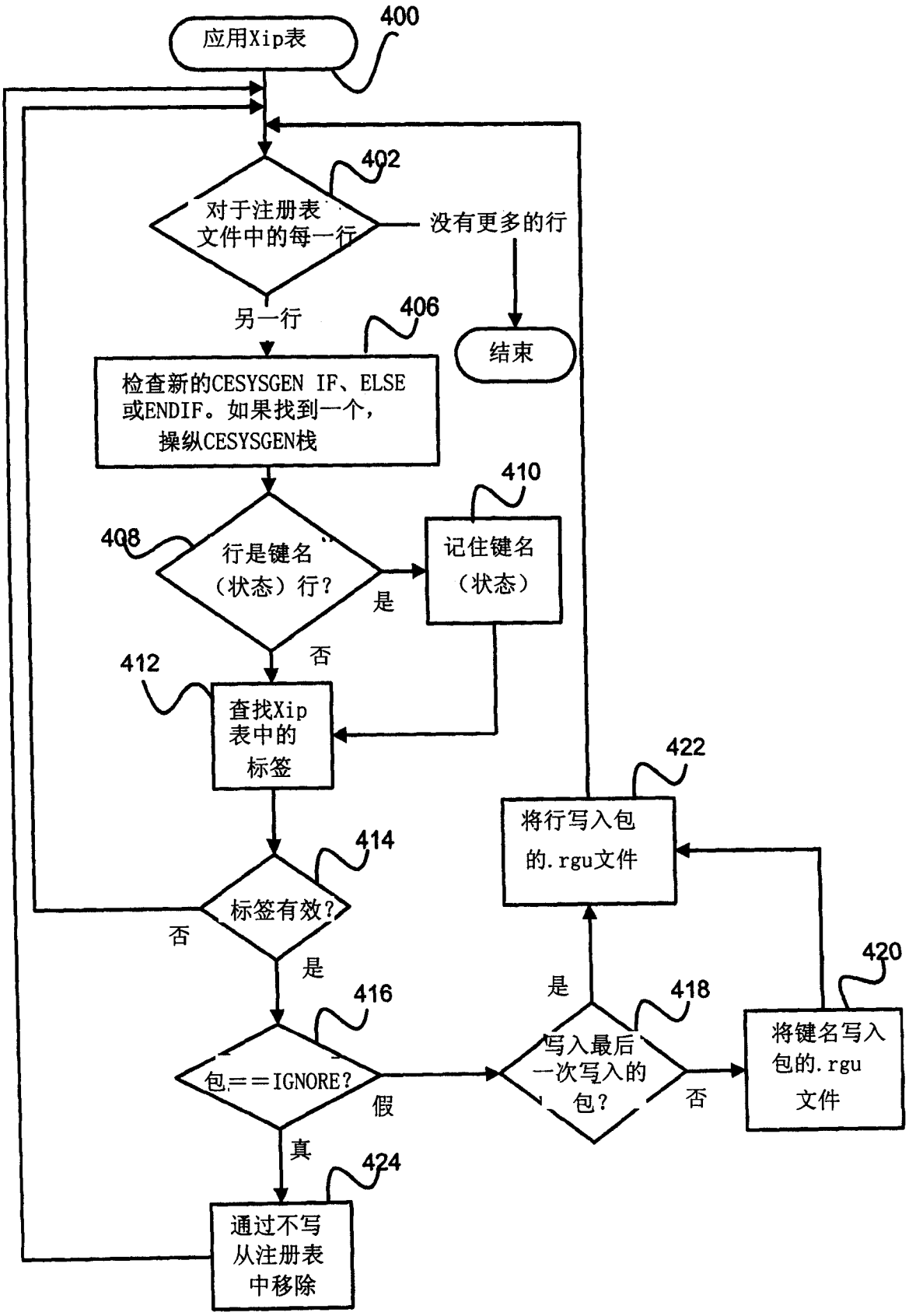

400
应用Xip表
402
对于注册表文件中的每一行
没有更多的行
另一行
406
结束
检查新的CESYSGEN IF、ELSE或ENDIF。如果找到一个，操纵CESYSGEN栈
410
408
行是键名（状态）行？
是
记住键名（状态）
否
412
查找Xip表中的标签
422
将行写入包的.rgu文件
414
标签有效？
否
是
420
是
416
418
包==IGNORE？
写入最后一次写入的包？
假
否
将键名写入包的.rgu文件
真
424
通过不写从注册表中移除

图 4

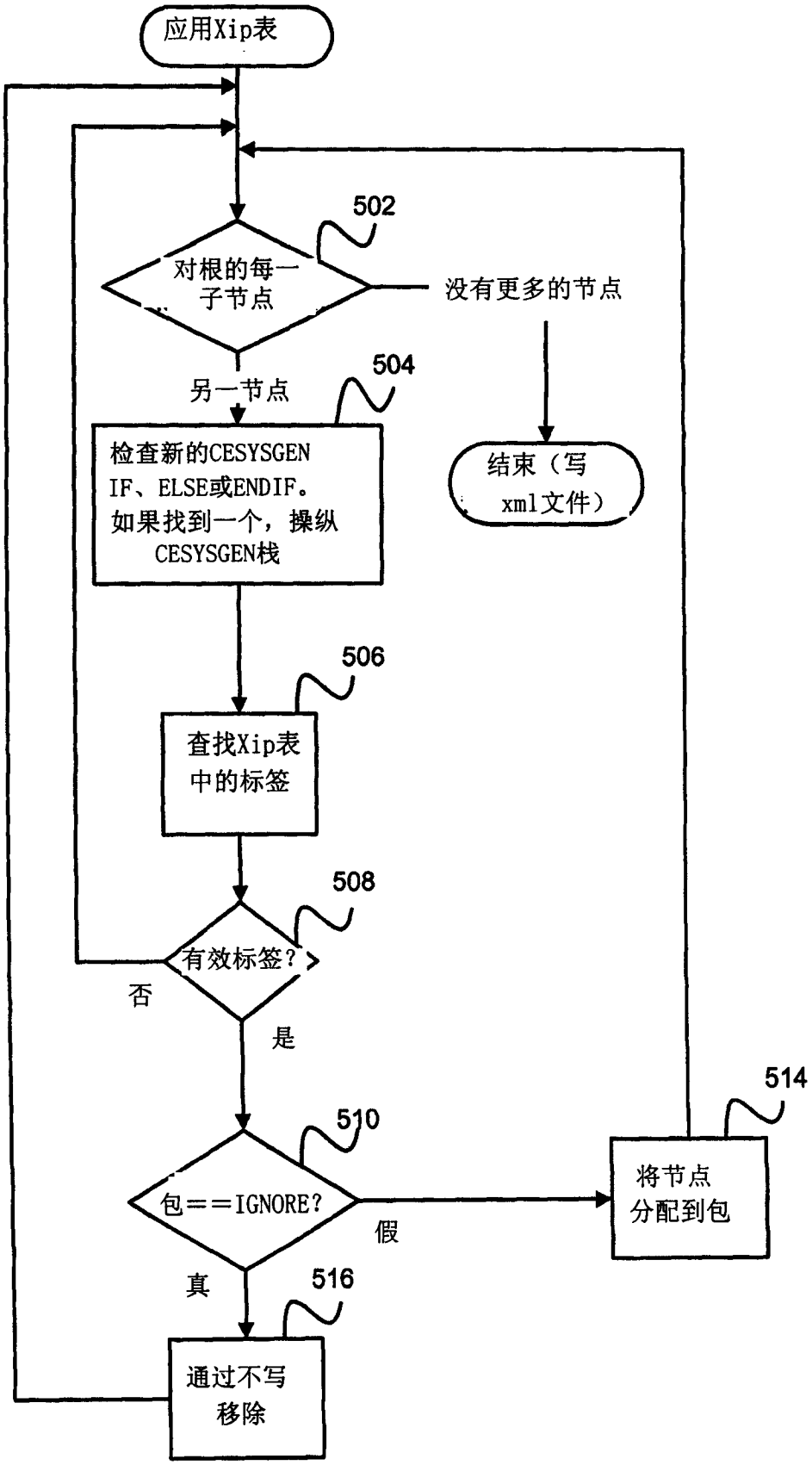

应用Xip表
502
对根的每一子节点
没有更多的节点
另一节点
504
检查新的CESYSGEN IF、ELSE或ENDIF。如果找到一个，操纵CESYSGEN栈
结束（写xml文件）
506
查找Xip表中的标签
508
有效标签?
否
是
514
510
包==IGNORE?
假
将节点分配到包
真
516
通过不写移除

图 5

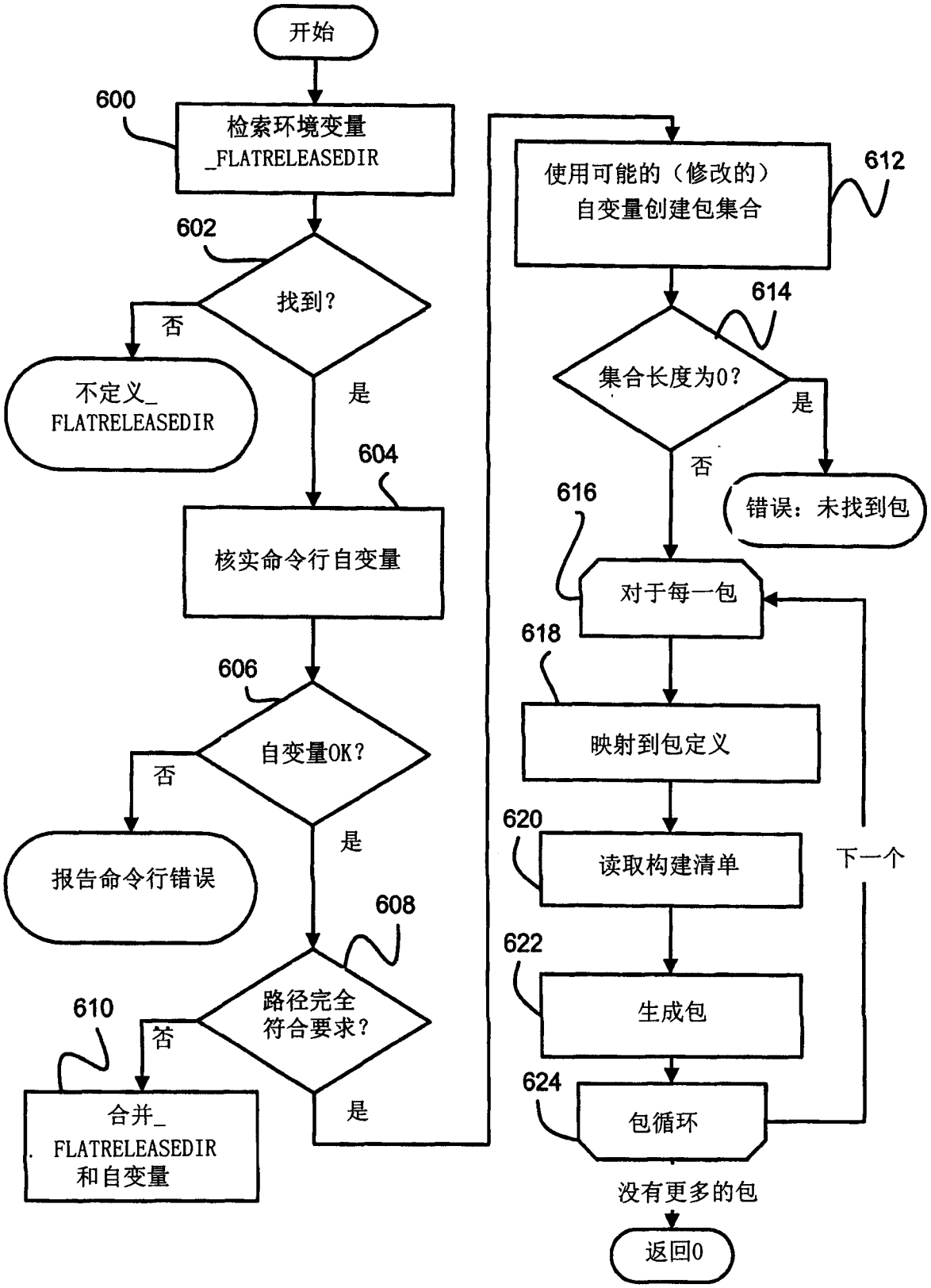

开始
600
检索环境变量
_FLATRELEASEDIR
602
找到?
否
不定义_
FLATRELEASEDIR
是
604
核实命令行自变量
606
自变量OK?
否
报告命令行错误
是
608
路径完全
符合要求?
610
否
合并_
FLATRELEASEDIR
和自变量
是
612
使用可能的（修改的）
自变量创建包集合
614
集合长度为0?
是
错误：未找到包
否
616
对于每一包
618
映射到包定义
620
读取构建清单
622
生成包
624
包循环
下一个
没有更多的包
返回0

图 6

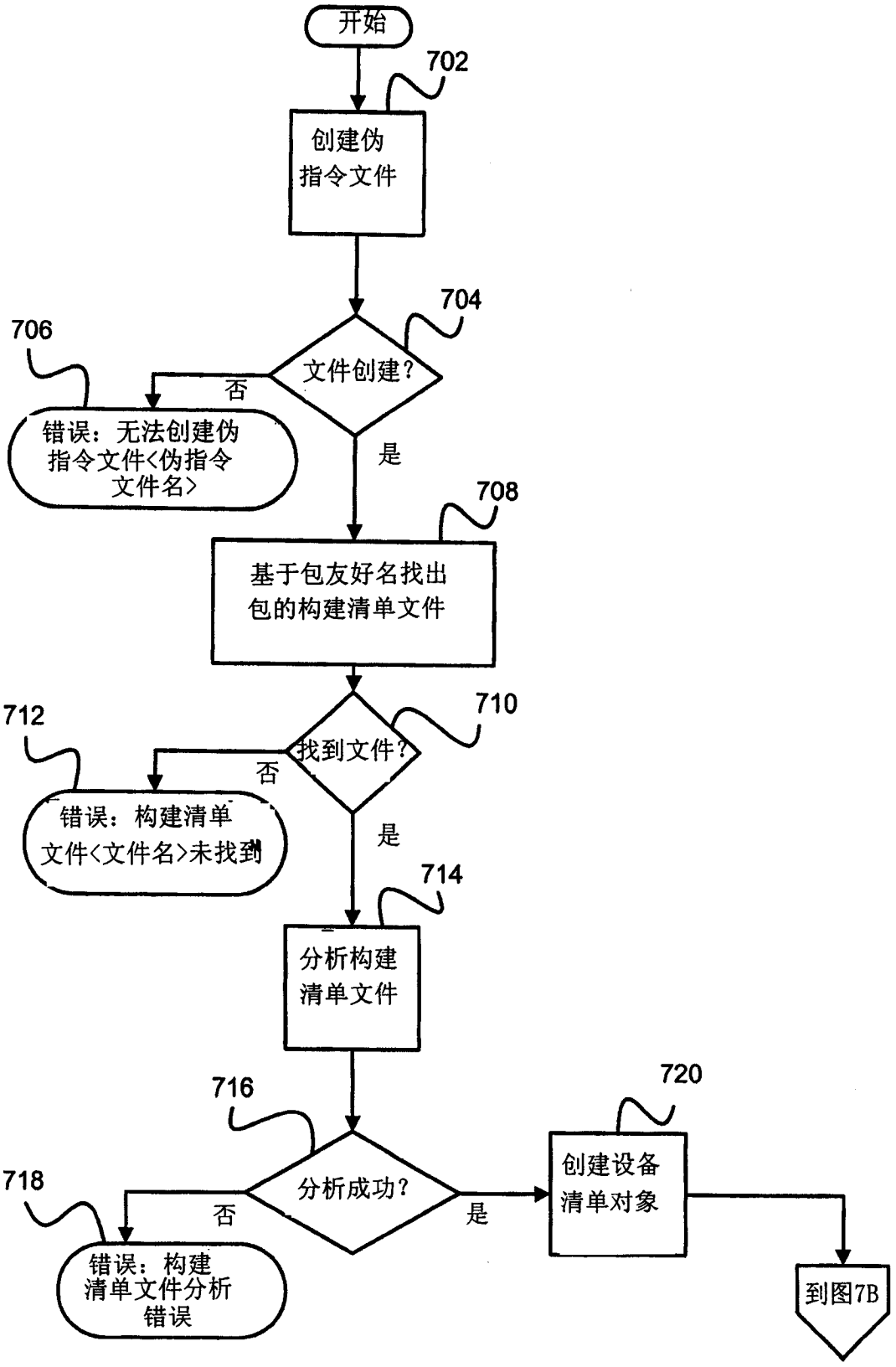

开始
702
创建伪指令文件
704
文件创建?
否
706
错误：无法创建伪指令文件<伪指令文件名>
是
708
基于包友好名找出包的构建清单文件
710
找到文件?
否
712
错误：构建清单文件<文件名>未找到
是
714
分析构建清单文件
716
分析成功?
否
718
错误：构建清单文件分析错误
是
720
创建设备清单对象
到图7B

图 7A

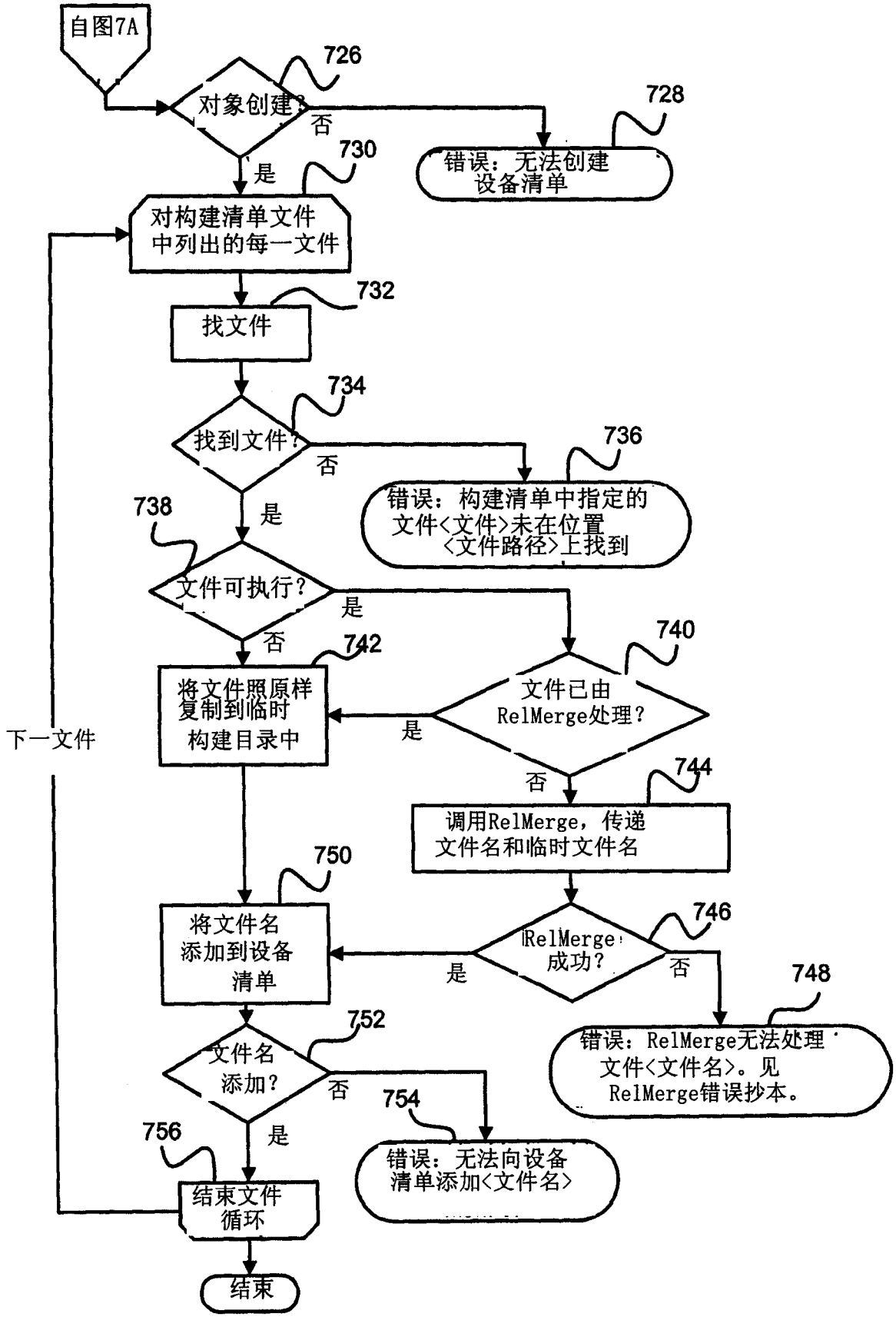

自图7A
726
对象创建?
否
728
错误：无法创建
设备清单
是
730
对构建清单文件
中列出的每一文件
732
找文件
734
找到文件?
否
736
错误：构建清单中指定的
文件<文件>未在位置
<文件路径>上找到
是
738
文件可执行?
是
否
740
文件已由
RelMerge处理?
是
742
将文件照原样
复制到临时
构建目录中
否
744
调用RelMerge，传递
文件名和临时文件名
下一文件
750
将文件名
添加到设备
清单
746
RelMerge
成功?
是
否
748
错误：RelMerge无法处理
文件<文件名>。见
RelMerge错误抄本。
752
文件名
添加?
否
754
错误：无法向设备
清单添加<文件名>
是
756
结束文件
循环
结束

图 7B

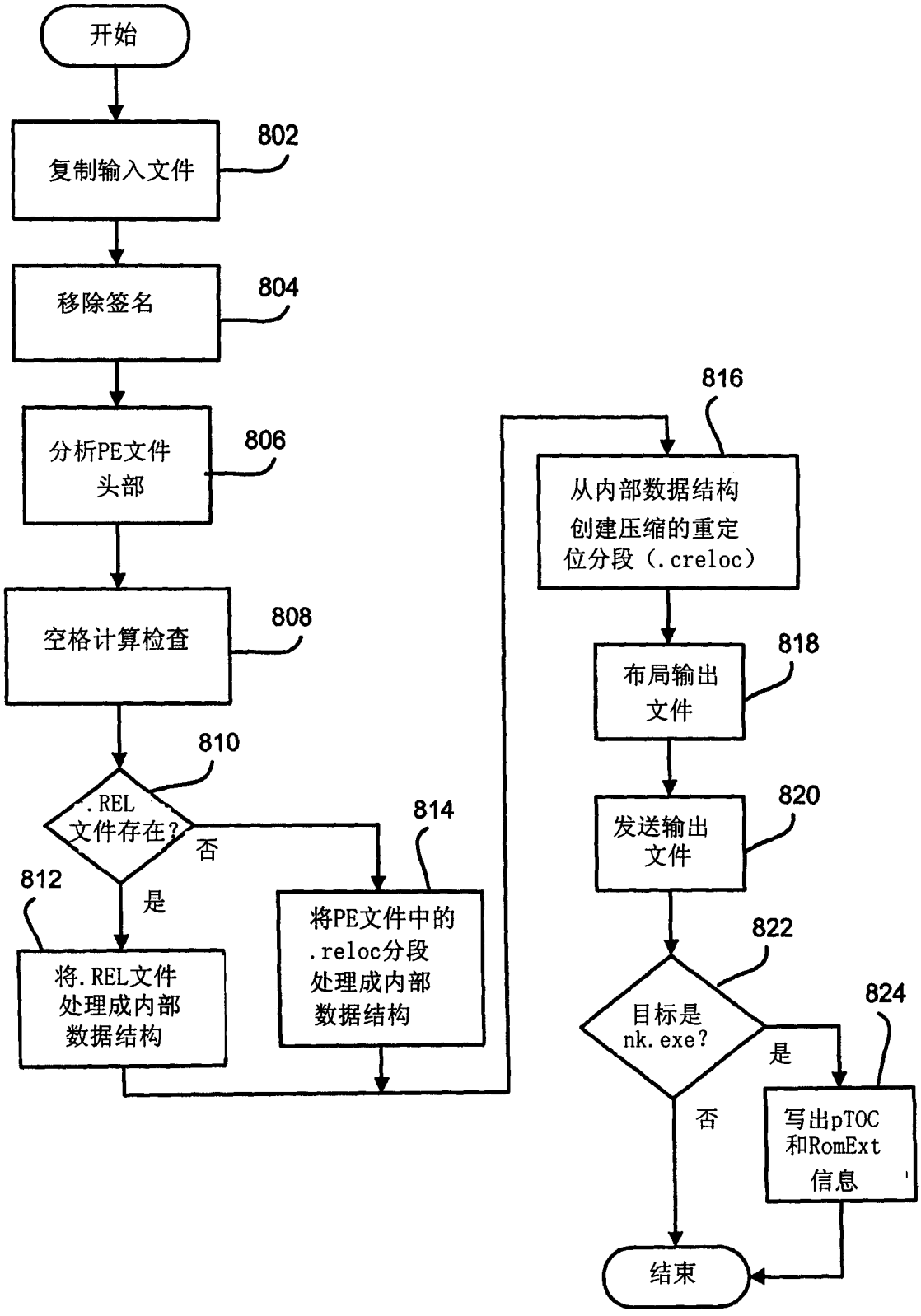

开始
复制输入文件
802
移除签名
804
分析PE文件头部
806
空格计算检查
808
.REL文件存在?
810
否
是
812
814
将.REL文件处理成内部数据结构
将PE文件中的.reloc分段处理成内部数据结构
816
从内部数据结构创建压缩的重定位分段（.creloc）
布局输出文件
818
发送输出文件
820
目标是nk.exe?
822
是
否
824
写出pTOC和RomExt信息
结束

图 8

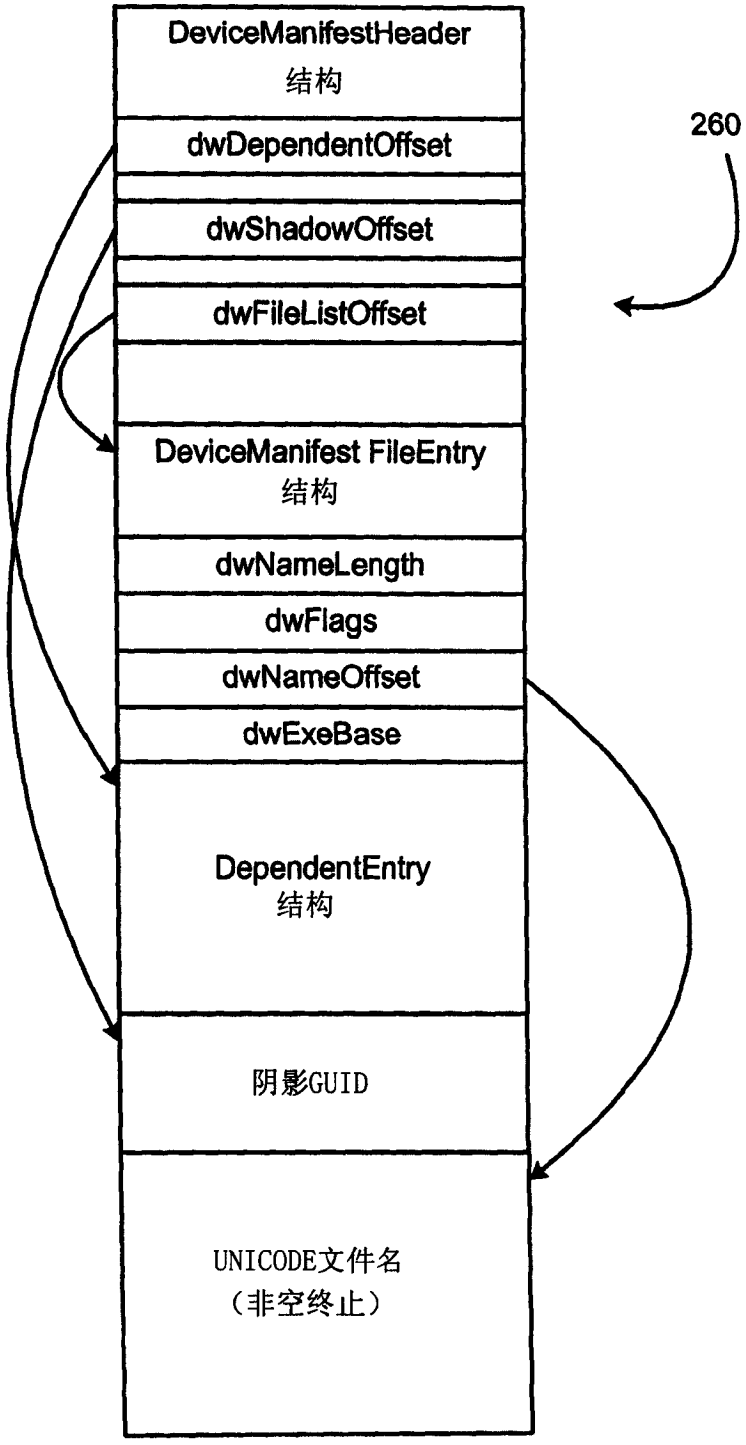

DeviceManifestHeader
结构
260
dwDependentOffset
dwShadowOffset
dwFileListOffset
DeviceManifest FileEntry
结构
dwNameLength
dwFlags
dwNameOffset
dwExeBase
DependentEntry
结构
阴影GUID
UNICODE文件名
（非空终止）

图 9

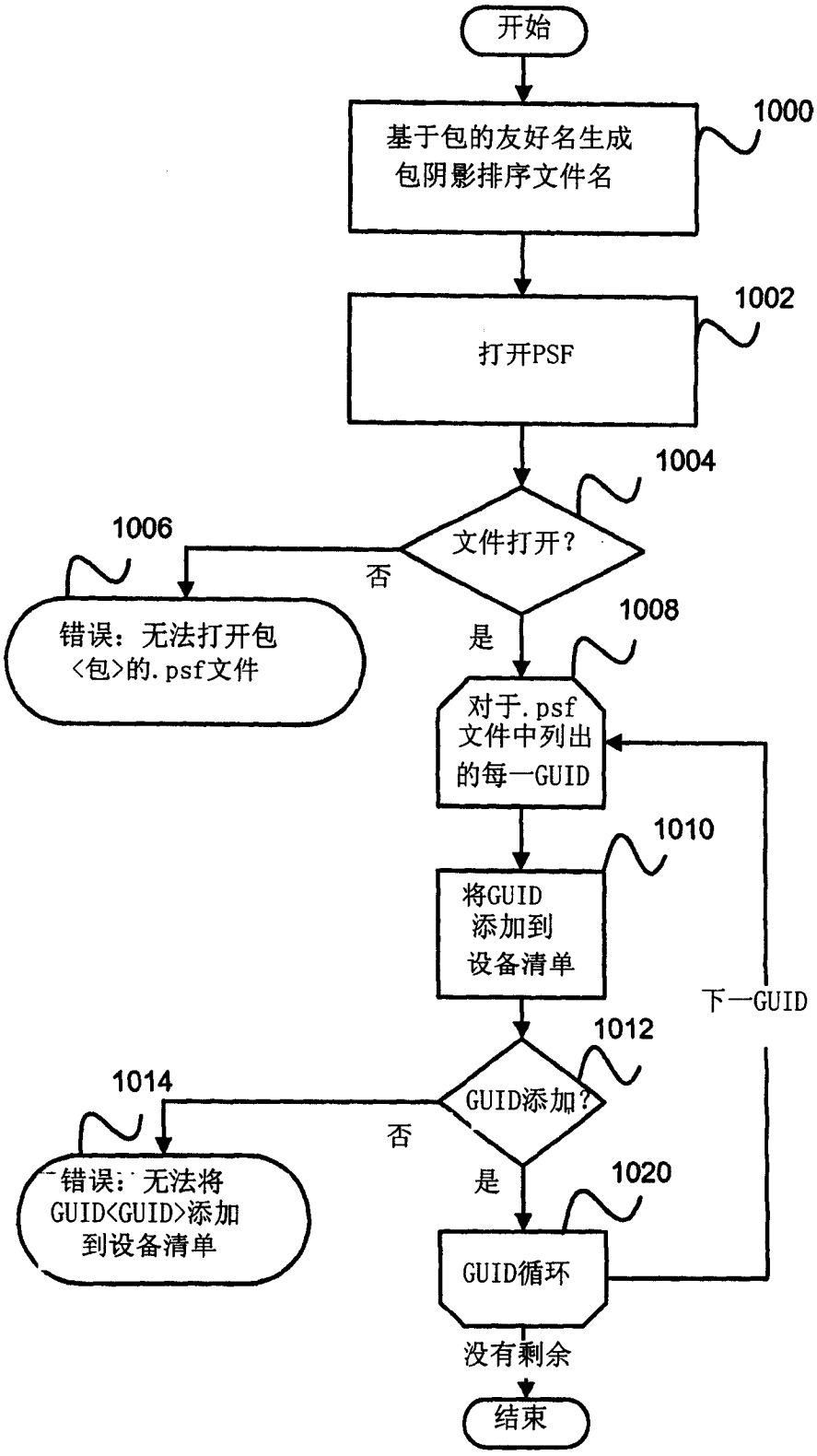

开始
基于包的友好名生成包阴影排序文件名
1000
打开PSF
1002
文件打开？
1004
否
1006
错误：无法打开包<包>的.psf文件
是
1008
对于.psf文件中列出的每一GUID
将GUID添加到设备清单
1010
下一GUID
GUID添加？
1012
否
1014
错误：无法将GUID<GUID>添加到设备清单
是
1020
GUID循环
没有剩余
结束

图 10A

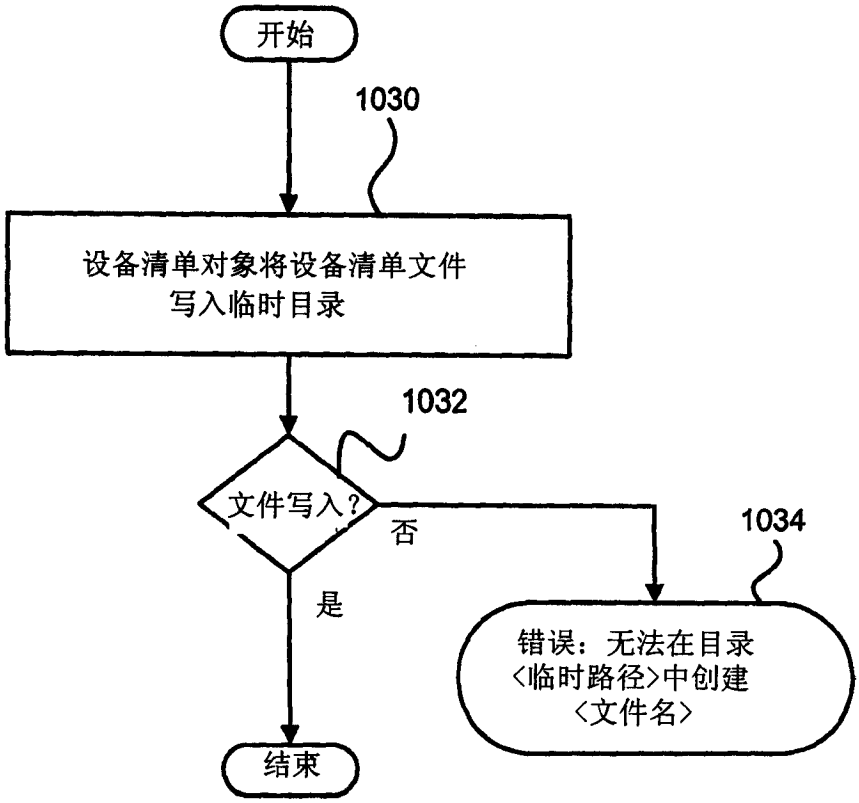

开始
1030
设备清单对象将设备清单文件
写入临时目录
1032
文件写入?
否
1034
是
错误：无法在目录
<临时路径>中创建
<文件名>
结束

图 10B

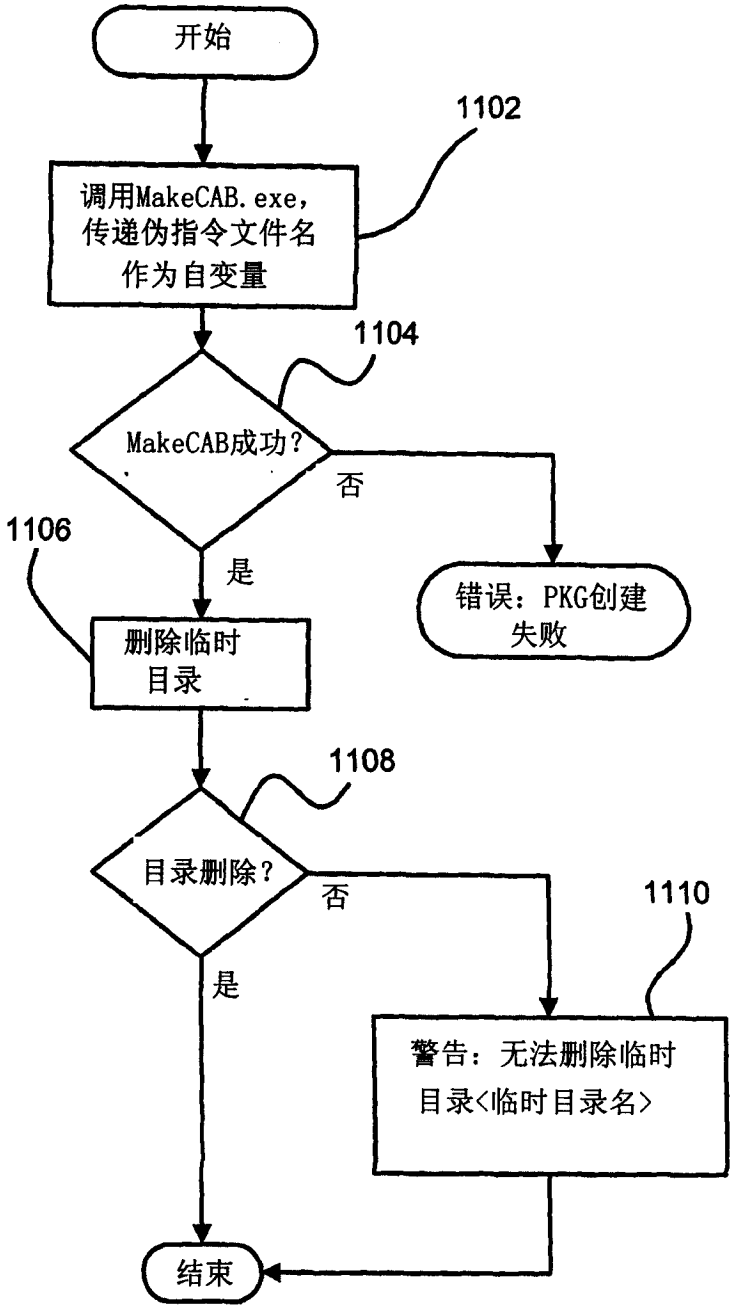

开始
1102
调用MakeCAB.exe，
传递伪指令文件名
作为自变量
1104
MakeCAB成功？
否
是
1106
错误：PKG创建
失败
删除临时
目录
1108
目录删除？
否
1110
是
警告：无法删除临时
目录<临时目录名>
结束

图 11