(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 102063315 A
(43)申请公布日 2011.05.18

(21)申请号 201010503801.1

(22)申请日 2006.06.02

(30)优先权数据
11/144411 2005.06.02 US

(62)分案原申请数据
200680028333.9 2006.06.02

(71)申请人 数学工程公司
地址 美国马萨诸塞州

(72)发明人 P·H·韦布 B·西蒙
C·G·尼兰德 J·米克 M·厄尔曼

(74)专利代理机构 北京泛华伟业知识产权代理有限公司 11280
代理人 王勇

(51)Int.Cl.
G06F 9/445(2006.01)
G06F 9/45(2006.01)

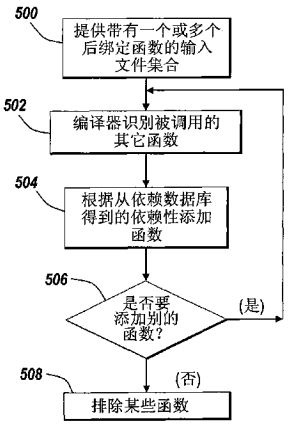
权利要求书 2 页 说明书 11 页 附图 5 页

(54)发明名称

从外部程序环境调用后绑定函数

(57)摘要

本发明旨在将后绑定程序语言自动地集成到外部程序环境之中。外部程序环境可以是先绑定环境、后绑定环境或它们的组合。与后绑定程序环境相关的是模块化执行引擎和接口机构，其中的接口机构能够以与外部环境兼容的格式表达后绑定语义。



CN 102063315 A

1. 一种将后绑定函数 (404a) 集成到支持先绑定函数 (416) 的语言中的方法，该方法包含以下步骤：

提供在支持先绑定函数 (416) 的语言中要被调用的后绑定函数 (404a)，该提供步骤包括：

检查后绑定程序 (402) 以标识由后绑定函数 (404a) 所需的一个或多个相关后绑定函数；以及

自动地生成所述后绑定函数 (404a)、一个或多个相关后绑定函数以及支持先绑定函数 (416) 的语言之间的接口 (408)。

2. 根据权利要求 1 所述的方法，其中所述接口 (408) 包括一个或多个先绑定函数或方法调用。

3. 根据权利要求 2 所述的方法，其中还包括以下步骤：

创建包括一个或多个先绑定函数 (408) 和后绑定函数 (404a) 以及一个或多个相关函数的包；

存储该包。

4. 根据权利要求 2 所述的方法，其中还包括以下步骤：

当后绑定函数 (404a) 的数据类型与所述一个或多个先绑定函数的数据类型不相兼容时，从所述一个或多个先绑定函数创建可与后绑定函数 (404a) 的数据类型相兼容的数据。

5. 根据权利要求 2 所述的方法，其中还包括以下步骤：

当所述一个或多个先绑定函数的数据类型与所述后绑定函数 (404a) 的数据类型不相兼容时，从后绑定函数 (404a) 创建可与所述一个或多个先绑定函数的数据类型相兼容的数据。

6. 根据权利要求 2 所述的方法，其中还包括以下步骤：

生成可适于在支持先绑定函数 (416) 的语言中调用的后绑定程序 (402) 的可调度形式。

7. 一种用于将后绑定函数 (404a) 集成到支持先绑定函数 (416) 的语言中的媒介，其包括：

提供在支持先绑定函数 (416) 的语言中要被调用的后绑定函数 (404a)，其特征在于下列步骤：

自动地生成所述后绑定函数 (404a) 和支持先绑定函数 (416) 的语言之间的接口 (408)，其中接口 (408) 包括一个或多个先绑定函数或方法调用；

创建包括一个或多个先绑定函数和后绑定函数 (404a) 以及该后绑定函数 (404a) 的一个或多个相关函数的包；

存储该包。

8. 根据权利要求 7 所述的媒介，还包括以下步骤：

生成可适于在支持先绑定函数 (416) 的语言中调用的后绑定函数 (404a) 的可调度形式。

9. 根据权利要求 7 所述的媒介，其中所述包包含与后绑定函数 (404a) 相关的所有可执行内容。

10. 根据权利要求 7 所述的媒介，其中所述包是加密的。

11. 根据权利要求 7 所述的媒介，还包括以下步骤：

使用支持先绑定语言的编译器对所述一个或多个先绑定函数进行编译。

12. 一种适于将后绑定函数 (404a) 集成到支持先绑定函数 (416) 的语言中的系统，该系统包括：

存储设备 (110)，其被配置用于存储：支持先绑定函数 (416) 的语言；所述后绑定函数 (404a)；以及

处理器 (102)，其被配置成：

提供在支持先绑定函数 (416) 的语言中要被调用的后绑定函数 (404a)；

检查后绑定程序 (402) 以标识由后绑定函数 (404a) 所需的一个或多个相关后绑定函数；以及

自动地生成所述后绑定函数 (404a)、一个或多个相关后绑定函数以及支持先绑定函数 (416) 的语言之间的接口 (408)。

13. 根据权利要求 12 所述的系统，其中所述接口 (408) 包括一个或多个先绑定函数或方法调用。

14. 根据权利要求 13 所述的系统，其中所述存储设备进一步被配置成：

存储所述一个或多个先绑定函数和后绑定函数 (404a) 以及一个或多个相关函数。

15. 根据权利要求 12 所述的系统，其中所述处理器进一步被配置成：

生成可适于在支持先绑定函数 (416) 的语言中调用的后绑定程序 (402) 的可调度形式。

16. 根据权利要求 12 所述的系统，其中所述处理器进一步被配置成：

使用支持先绑定语言的编译器对所述一个或多个先绑定函数进行编译。

从外部程序环境调用后绑定函数

[0001] 本申请为申请号为 200680028333.9、申请日为 2006 年 6 月 2 日、发明名称为“从外部程序环境调用后绑定函数”的申请的分案申请。

[0002] 相关申请

[0003] 本申请要求于 2005 年 6 月 2 日递交的、序列号为 11/144411 的美国专利申请的权益，该申请的内容包括在本申请中一并作为参考。

技术领域

[0004] 本发明涉及到应用以及软件组件，特别涉及从先绑定的应用中调用后绑定函数的方法与系统。

背景技术

[0005] 在应用中绑定函数调用需要用真实的机器地址取代对该函数的符号地址。因此，绑定操作确定哪个函数（即实现该功能的代码部分）在运行时执行。当绑定发生在编译阶段时，该绑定为“先绑定”。与此相对应地，当绑定发生在运行时，该绑定为“后绑定”。通常可将编程语言分类为先绑定的或后绑定的。

[0006] 函数调用绑定时间的选择（即是先绑定还是后绑定）是运行时的代码性能（先绑定在运行时更快）和程序员在编程时的便易性（后绑定程序更易于编写）之间的一种折衷。当前，程序员必须决定以下何种更为重要：程序员编写程序时所用的时间还是程序运行所用的时间。不幸的是，通常这是一种很难做出的选择。在现有的系统中还没有机制能有效地平衡这种折衷。

发明内容

[0007] 本发明提供了一种将后绑定程序函数集成到外部程序环境之中的方法和系统。该外部程序环境可以是先绑定环境，也可以是后绑定环境，或者两种环境的组合。

[0008] 根据本发明的第一个方面，提供了一种将后绑定应用集成到外部程序环境中的方法。提供后绑定程序和后绑定环境，随后加以检测以确定哪些后绑定调用存在于后绑定程序和环境中而应当使其为外部环境可用。然后生成一个接口，其中该接口允许将这些后绑定函数调用集成到该外部程序环境中。用于实现本发明的外部环境可以是后绑定环境、先绑定环境或者它们的组合。

[0009] 在本发明的另一个方面中，提供了一种用于将后绑定程序语言集成到外部环境中的系统，其中该系统包括一个外部环境，一个后绑定程序和程序环境。该外部环境可以是后绑定程序环境、也可以是先绑定程序环境、或者它们的某种组合。后绑定程序环境包括一个模块化执行引擎和一种接口机构，使得该接口机构允许用外部环境兼容的格式表达后绑定语义。

[0010] 在本发明的又一个方面中，提供了一种将可执行的后绑定内容打包到与外部程序环境一起使用的可重新发布的应用之中的方法。该外部环境可以是后绑定程序环境、或是

先绑定程序环境或者它们的某种组合。该方法包括以下步骤：首先检查后绑定内容，以确定后绑定内容中的哪些文件代表后绑定应用。然后提供后绑定模块化执行引擎，其中，模块化执行引擎仅包含那些支持后绑定应用所必需的函数。而且，模块化执行引擎被配置成仅执行那些支持后绑定应用所必需的函数。

[0011] 在本发明的又一个方面中，提供了一种包括客户端与服务器的网络，其中服务器端能够执行后绑定程序环境和外部程序环境。一些合适的外部环境包括（但不局限于）后绑定程序环境，先绑定程序环境或者它们的组合。客户端通过网络与服务器进行通信。所生成的接口允许后绑定程序和程序环境与外部程序环境之间的数据交换，以使得能够将后绑定应用集成到外部程序环境中。

[0012] 在本发明的一个可选择的方面中，描述了一种用于将后绑定应用集成到外部环境中的系统，其中分发服务器用于向客户端设备提供信息。此外，还提供了一种接口机构，其中该接口机构允许将后绑定函数调用集成到外部程序环境中。

附图说明

[0013] 以下，结合附图来详细说明本发明的实施例，其中：

[0014] 图 1 为对用于实现本发明的示意性实施例的电子设备进行说明的方框示意图；

[0015] 图 2 描述了一种适于实现本发明的示意性实施例的客户端－服务器环境，其中后绑定函数能被合并到外部编程环境中；

[0016] 图 3 描述了用于实现本发明的示意性实施例的组件；

[0017] 图 4 是对示意性实施例中的组件交互进行详细说明的图例；

[0018] 图 5 是说明用于实现确定对其它后绑定函数间的依赖性的步骤的流程图；

[0019] 图 6 是用于将后绑定函数调用集成到后绑定程序环境中所必需的组件的一个示意性实施例。

具体实施方式

[0020] 本发明的示意性实施例提供了一种机制，其对于存在于外部程序环境之中的应用调用在后绑定的语言中存在的函数。为了说明之目的，下面将参照将后绑定函数集成到先绑定外部程序环境进行描述。还存在一些可替代的外部环境，包括但不局限于那些包含先绑定语言、后绑定语言、或者它们的组合的环境。在本实施例中，将后绑定函数集成到先绑定环境之中允许程序员选择何时实现执行时间与编码便易性之间的折衷。因此，程序员可以采用先绑定函数调用来执行应用中那些关键部分，而对于难以编程的函数则采用后绑定函数调用。

[0021] 将后绑定函数的调用集成到先绑定编程环境之中，除了提供前述的折衷平衡外，还具有很多其它的优点。首先，这种集成为先绑定编程环境提供了一种利用后绑定的表达能力之优点的方式。其次，这种集成使得先绑定编程环境能够将一个先绑定函数调用与一个或多个后绑定函数调用的集合相绑定。再次，它为大量的遗留系统提供了包括进更多现代编程方法的方式，例如所发现的采用后绑定函数调用的方式。

[0022] 在替换实例中，将后绑定函数集成到后绑定程序环境中能提供大量的优点，而这些优点是在利用将后绑定函数集成到先绑定语言时并不容易被认识到的。例如，大多数后

绑定程序语言专门用于在某一特殊问题领域中操作。例如，LISP 后绑定语言特别适用于对字符串列表中的符号处理。与之相比，后绑定 PROLOG 编程语言则适用于原理的证明以及逻辑推理问题。与之相比，后绑定 MATLAB 编程语言适合于在数值线性代数领域中操作。上述的后绑定程序语言仅用于说明可用于本发明的可用的后绑定程序环境中的一些实例。这里所列出的例子并不是明确的，作为本领域的普通技术人员很容易理解，存在着大量的可适用于本发明的后绑定程序环境。

[0023] 当将后绑定函数的调用集成到后绑定环境中时，本发明允许用户构造一种程序，其将对多种后绑定语言的调用组合起来，以便采用最合适的语言来解决存在的计算问题。

[0024] 出于示例目的，本示意性的实施例将针对 MATLAB 的环境进行详细的说明。虽然本示意性实施例将会针对基于 MATLAB 的应用进行说明，但本发明可以被用来配置和发布在不同的编程和开发环境中所生成的应用，例如但不局限于由国家仪器公司 (National Instruments, Inc) 提供的 Labview，LISP，或 PROLOG。

[0025] MATLAB®是一种直观高性能的语言和技术计算环境。MATLAB®为数据分析、可视化以及程序的开发提供了数学和图形工具。MATLAB®以一个易用环境集成了计算与编程，在所述的易用环境中，问题与解决方案用人们熟悉的数学符号表达。MATLAB®是一种交互系统，该系统中的基本数据元素是无需定义维数的数组。这使得用户在解决许多技术性计算问题、特别是那些要用到矩阵和向量公式的问题时，编写程序所花费的时间仅是一种标量非交互语言（例如 C 和 FORTRAN 等）所花费时间的一小部分。MATLAB®提供了包括MATLAB®编译器在内的应用开发工具。用于 COM 的MATLAB®编制器 (Builder) 和用于 Excel 工作的MATLAB®编制器 (Builder) 与MATLAB®编译器协同工作，以自动生成在 COM 和 Excel 环境中执行的对象。这些应用开发工具使得用户能够创建并分配MATLAB®应用并与可能使用MATLAB®或其他环境的同事共享MATLAB®成果。

[0026] 图 1 描述了适用本发明的描述性实施例的电子设备 100 的一个实施例。电子设备 100 代表许多不同的技术，例如个人电脑 (PC)，膝上型电脑，工作站，个人数字助理 (PDA)，因特网用具，蜂窝电话，以及类似的设备。在所示的实施例中，电子设备 100 包括中央处理单元 (CPU)102 以及显示设备 104。显示设备 104 使得电子设备 100 通过视觉显示器直接与用户进行通信。电子设备 100 还包括键盘 106 以及例如鼠标器的输入设备 108。其他未被描述的潜在输入设备包括但不局限于：指示笔，轨迹球，控制杆，触摸垫，触摸屏，以及其他类似的设备。电子设备 100 包括用于存储数据和指令的第一存储设备 110 和第二存储设备 112。存储设备 110 和 112 包括但不局限于如下技术：软盘驱动器，硬盘驱动器，磁带驱动器，光盘驱动器，只读存储器 (ROM)，随机存取存储器 (RAM)，以及其他类似的设备。如浏览器，JAVA 虚拟机之类的应用程序，以及其他实用程序和应用可以驻留在存储设备 110 和 112 之一或二者中。电子设备 100 还可以包括用于与一个或多个在图示的电子设备 100 之外的外部电子设备进行通信的网络接口 114。调制解调器是用来实现与外部的电子设备或网络建立连接的网络接口 114 的一种形式。CPU102 通过内部或外部方式连接到前述的一个或多个部件。除了前述的这些应用之外，可以在电子设备 100 上安装并运行例如MATLAB® 116 的建模应用。

[0027] 应当注意的是，电子设备 100 仅仅代表了实现本发明的一种结构。尽管如此，本领域普通技术人员将会认识到本发明并不局限于以上具体描述的设备 100。可以采用其他类型的实现方式，包括部分或整体基于嵌入代码的实现方式，在此则不需要用户输入或显示设备；而且，处理器能够与其他处理器或设备直接进行通信。

[0028] 图 2 是对用于实现本发明的分布式网络的示意性实施例。该系统包括与网络 200 相连的客户端设备 202。网络上还连接有服务器设备 204、接口机构 210 和存储设备 206。接口机构 210 能够将后绑定函数调用集成到先绑定程序环境之中。

[0029] 当在面向对象的环境中进行编程时，采用后绑定函数调用的优点之一在于，能够使得开发者在编写代码时无需就所要操作的对象的确切对象类型而作特殊的考虑。换句话说，后绑定函数调用可以支持多态。

[0030] 利用本发明的接口机构 210，当先绑定编译器在编译时遇到存在于后绑定环境中的后绑定函数时，，利用该接口机构 210，先绑定环境中的编译器允许在运行时执行存在于后绑定环境中的后绑定函数。接口机构 210 作为一组后绑定函数的代理，该后绑定函数组中的某一个（根据在运行时输入的类型）将在运行时被调用。

[0031] 此外，为了举例说明的目的，在本实施例中的接口机构是与网络 200 相连接的；但本领域的普通技术人员很容易理解，该接口机构 210 还能够连接到客户端设备 202、服务器 204 或者它们的组合。本领域的普通技术人员还可以认识到，本实施例中的网络只是一个示例性的实例，它的功能可以以任意的方式分到整个网络中。例如，多个客户端可以与一个服务器连接，一个客户端也可以连接到多个服务器上，或者上述情况的一些组合。

[0032] 图 3 描述了应用于本发明的示意性实施例中的组件。在图示实施例中所描述的组件用于将后绑定函数集成到先绑定环境 300 之中。在一个示例中，先绑定程序环境为由位于美国华盛顿州雷德蒙德市 (Redmond) 的微软公司所提供的 Visual Studio® 程序环境，后绑定函数为 MATLAB® 函数。选择 MATLAB® 和 Visual Studio® 只是出于举例说明的目的。对多种其它在先和后绑定语言的选择都可用于实现本发明。

[0033] 在图示的实例中，当后绑定函数为 MATLAB® 函数时，MATLAB® 编译器生成后绑定接口函数 304 和 306，后者是位于先绑定环境 300 中的先绑定函数。这些接口函数 304 和 306 的名称与功能完全在编译时得到确定。在运行时内，这些接口函数 304 和 306 调用后绑定函数。具体的，后绑定接口函数 304 通过一个模块化后绑定执行引擎 302（下文中将作详细说明）调用包含有一个或多个后绑定函数的集合 308。类似的，后绑定接口函数 306 通过此模块化后绑定执行引擎 302 调用包含有一个或多个后绑定函数的集合 310。从而，提供了一种使得先绑定环境 300 能够将函数的绑定延迟到所选定函数的运行时的机制。

[0034] 正如图 3 所示，后绑定接口函数 304 和 306 可能会引起多于一个后绑定函数要被执行（参见 308 和 310）。也可能存在这样的情况，一个后绑定接口函数与一个单独的后绑定函数相关联，也可能存在其他情况，多个后绑定函数与一个单独的后绑定接口函数相关联。此外，根据提供给接口函数的输入类型，每次调用会导致对所关联的集合中一个不同的后绑定函数的调用。

[0035] 在一个实施例中，接口机构可通过应用程序接口 (API) 实现，该接口具有可以从先绑定应用调用的函数。API 可被用于启动对在后绑定环境中的函数的调用。例如，在 C

程序环境下运行的 API 能够调用在MATLAB®后绑定环境中的函数。可选择地，API 可以是不需要其他辅助程序操作的独立可执行程序。从以先绑定 C 语言运行的应用程序的角度看，在后绑定MATLAB®应用中所调用的函数，表现为例如是一种传统的函数调用，其中函数及其参数是不言自明的，其输出以与任何现有函数相类似的方式传递。

[0036] 尽管将 API 作为接口机构的实例加以描述，但接口机构并不局限于采用 API，例如还可以是运行于先绑定或后绑定环境中的软件或硬件组件。接口机构能够以先绑定编程环境在运行时易于识别的格式，自动表达后绑定编程环境的语义，并且如前面已经提出的，可以采用适当形式来表达所述语义。

[0037] 模块化后绑定执行引擎 302 提供用于执行存在于后绑定环境中的函数的机制。由于这些函数是后绑定函数，它们在运行时被绑定在后绑定环境中。在后绑定环境为MATLAB®环境的情况下，模块化后绑定执行引擎可以是MATLAB®组件运行引擎 (runtime engine)。MATLAB®组件运行引擎是一种独立的共享库的集合，所述的共享库能够执行 M- 文件，MEX- 文件以及其它类型的MATLAB®可执行内容。采用MATLAB®组件运行引擎作为模块化执行引擎，当在后绑定环境中执行函数时，无需在一个新的用户可视窗口中打开MATLAB®组件运行引擎；而是由组件运行引擎以一种对用户不可见的方式在后台执行。

[0038] 本发明并不仅仅局限于MATLAB®应用以及MATLAB®组件运行引擎，还可以用各种替代后绑定环境以及执行引擎来实现。

[0039] 模块后绑定模块化后绑定执行引擎 302 跟随先绑定环境 300 中的后绑定接口函数调用 304 或 306 的执行，模块后绑定模块化后绑定执行引擎 302 既可以保持运行，也可以终止。允许模块后绑定模块化后绑定执行引擎 302 保持运行，可以使得从先绑定环境 300 中调用的后续的函数在后绑定环境中执行得更快，并且不再需要启动模块后绑定模块化后绑定执行引擎 302。可选择地，在执行后绑定函数之后终止在先绑定环境 300 中受到控制的模块化后绑定执行引擎 302，可以减少对处理器的需要和资源的分配，并且在模块化执行引擎 300 终止时，与模块后绑定模块化后绑定执行引擎 302 相关联的资源得以释放。

[0040] 一个具体的实例有助于说明图示示意性实施例的操作。该实例是一个来自于后绑定程序环境（例如MATLAB®）之中的MTALAB®函数，该函数用于为任何的输入数字加 10。该函数被称为 plus 10 并定义如下：

[0041] function s = plus 10(x)

[0042] s = x+10;

[0043] disp([num2str(x)' +10 =' num2str(s)]);

[0044] 以下所描述的是由MATLAB®编译器从包含函数 plus 10 的 M 类型的文件 plus 10.m 自动生成的 C 函数库的源文件。下面所列举的第一个文件是头文件 libplus.h。该头文件之后跟随文件 libplus.c。函数 mlfPlus10 是一个接口函数的实例，该接口函数是存取由 plus 10 函数提供的MATLAB®功能的 C 程序调用。

[0045] 以下为头文件：

[0046] #ifndef__libplus_h

[0047] #define__libplus_h1

```
[0048]  #include " mclmcr.h"
[0049]  #ifdef__cplusplus
[0050]  extern " C" {
[0051]  #endif
[0052]  extern bool libplusInitializeWithHandlers(mclOutputHandlerFcn error_
handler,
[0053]            mclOutputHandlerFcn print_handler);
[0054]  extern bool libplusInitialize(void);
[0055]  extern void libplusTerminate(void);
[0056]  extern void mlxPlus10(int nlhs,mxArray  *plhs[],int nrhs,mxArray
*prhs[];
[0057]  extern void mlfPlus 10(int nargout,mxArray **s,mxArray *x);
[0058]  #ifdef__cplusplus
[0059]  }
[0060]  #endif
[0061]  #endif
```

[0062] 以下为 libplus.c 文件：

```
[0063]  #include <stdio.h>
[0064]  #include" mclmcr.h"
[0065]  #include " libplus.h"
[0066]  #ifdef cplusplus
[0067]  extern " C" {
[0068]  #endif
[0069]  extern const unsigned char__MCC COMPONENT_public_data[];
[0070]  extern const char*__MCC_COMPONENT_name_data;
[0071]  extern const char*__MCC_COMPONENT_root_data;
[0072]  extern const unsigned char__MCC_COMPONENT_session_data[];
[0073]  extern const char*__MCC_COMPONENT_matlabpath_data[];
[0074]  extern const int__MCC_COMPONENT_matlabpath_data_count;
[0075]  extern const char*__MCC_COMPONENT_mcr_runtime_options[];
[0076]  extern const int__MCC_COMPONENT_mcr_runtime_option_count;
[0077]  extern const char*__MCC_COMPONENT_mcr_application_options[];
[0078]  extern const int__MCC_COMPONENT_mcr_application_option_count;
[0079]  #ifdef__cplusplus
[0080]  }
[0081]  #endif
[0082]  static HMCRINSTANCE_mcr_inst = NULL;
[0083]  static int mclDefaultErrorHandler(const char *s)
[0084]  {
```

```
[0085]          return fprintf(stderr,s) ;
[0086]   }
[0087]   static int mclDefaultPrintHandler(const char *s)
[0088]   {
[0089]          return fprintf(stdout,s) ;
[0090]   }
[0091]   bool libpluslnitializeWithHandlers(
[0092]       mclOutputHandlerFcn error_handler,
[0093]       mclOutputHandlerFcn print_handler
[0094]   )
[0095]   {
[0096]     return(_mcr_inst == NULL ?
[0097]         mcllnitializeComponentlnstance(&_mcr_inst,
[0098]                          __MCC_COMPONENT_public_data,
[0099]                          __MCC_COMPONENT_name_data,
[0100]                          __MCC_COMPONENT_root_data,
[0101]                          __MCC_COMPONENT_session_data,
[0102]                          __MCC_COMPONENT_matlabpath_data,
[0103]                          __MCC_COMPONENT_matlabpath_data_count,
[0104]                          __MCC_COMPONENT_mcr_runtime_options,
[0105]                          __MCC_COMPONENT_mcr_runtime_option_count,
[0106]                          true,NoObjectType,LibTarget,
[0107]                           NULL,error_handler,print_handler)
[0108]    :true) ;
[0109]   }
[0110]   bool libpluslnitialize(void)
[0111]   {
[0112]          return libpluslnitializeWithHandlers(mclDefaultErrorHandler,
[0113]              mclDefaultPrintHandler) ;
[0114]   }
[0115]   void libplusTerminate(void)
[0116]   {
[0117]        if(_mcr_inst ! = NULL)
[0118]           mclTerminatelnstance(&_mcr_inst) ;
[0119]   }
[0120]   void mlxPlus10(int nlhs,mxArray*plhs[],int nrhs,mxArray*prhs[])
[0121]   {
[0122]        mclFeval(_mcr_inst, " plus10" ,nlhs,plhs,nrhs,prhs) ;
[0123]   }
```

[0124] void mlfPlus 10(int nargout,mxArray **s,mxArray *x)

[0125] {

[0126] mclMlfFeval(_mcr_inst, " plusi0" ,nargout,1,1,s,x) ;

[0127] }

[0128] 从一个先绑定编程环境 (如 C) 之中调用前述接口函数 mlxPlus10,在以下代码中描述 :

[0129] #include<stdio.h>

[0130] #include " matrix.h"

[0131] #include " libplus.h"

[0132] void usage(char *name)

[0133] {

[0134] printf(" Usage : % s integer\n" ,name) ;

[0135] printf(" \tAdds 10 to the given integer\n") ;

[0136] exit(-1) ;

[0137] }

[0138] int main(int ac,char *av[])

[0139] {

[0140] mxArray*input = mxCreateDoubleScalar(atoi(av[1])) ;

[0141] mxArray*output = NULL ;

[0142] double result ;

[0143] if(ac ! = 2)usage(av[0]) ;

[0144] mcllnitializeApplication(NULL,0) ;

[0145] libpluslnitialize() ;

[0146] mlxPlus 10(1,&output,1,&input) ;

[0147] libplusTerminate() ;

[0148] result = mxGetScalar(output) ;

[0149] printf(" % s+10 = % .0f\n" ,av[1],result) ;

[0150] mclTerminateApplication() ;

[0151] return 0 ;

[0152] }

[0153] 前述的代码是用 C 编程语言编写的,但本领域的普通技术人员应当理解可选用各种其它先绑定环境实现本发明。

[0154] 正如通过前述调用后绑定函数的方法所证明的,该处理被流水化,并同时提供了用最小的用户交互来实现将后绑定函数集成到先绑定环境中的这样一种合适的机制。对原始后绑定函数 (plus10.m) 的调用与对先绑定环境中的其它普通函数的调用看起来是类似的。此外,先绑定环境与往常一样继续处理先绑定函数,因而提供了更快的函数执行速度。

[0155] 图 4 对在示意性实施例中组件的具体交互过程进行了详细说明。首先,用户采用后绑定环境 400 开发一个后绑定应用或组件。在图 4 中,这一后绑定应用或组件表示为用户编写的后绑定函数套件 (function suite)402,其包括后绑定函数 404A 和 404B。为了

说明的目的，假设后绑定环境 400 是MATLAB®，而后绑定函数是MATLAB®函数。然后，用户调用MATLAB®编译器 406 以生成其应用或组件的可调度形式。MATLAB®编译器 406 检查输入文件以决定将要生成的用以调用后绑定函数 404A 和 404B 的先绑定接口文件的适当形式。进一步的，对先绑定环境的选择也会影响所生成的先绑定接口的形式。MATLAB®编译器检查输入文件以确定在后绑定环境 400 中对其它后绑定函数的依赖性。MATLAB®编译器然后将所有必需的后绑定函数打包到组件技术文件 (CTF) 档案 410 之中，并生成合适的先绑定接口函数 408。CTF 档案被用来包含与某一组件相关的所有可执行内容（即，M- 文件或 Mex- 文件）。CTF 档案也包含应用程序所必需的所有数据文件，以生成对程序执行所需的所有信息的快照。CTF 档案的内容是经过加密的。

[0156] 图 5 是用来描述确定对其它后绑定函数的依赖性所执行步骤的流程图。在一个实施例中，图 5 中所描述的步骤可用于计算函数调用树的传递闭包。这一过程起始于用户提供一个输入文件集合，该集合中的每个输入文件都包含一个或多个后绑定函数（步骤 500）。MATLAB®编译器检查步骤 500 所提供的输入文件集合中的每个后绑定函数，以确定哪些其它函数被该后绑定函数所调用（步骤 502）。编译器查看在编译过程中所生成的函数调用树。在步骤 502 中所识别的每个函数可能包含有被明确记载在依赖数据库中的依赖关系。例如，该数据库可以指出某一特定函数为了正确运行需要的一个或多个其它函数。依赖性分析器将这些函数添加到必要函数清单中（步骤 504）。步骤 502 和 504 重复执行直到没有其它的函数被添加到清单中（步骤 506）为止。在步骤 508 中，某些函数或函数组被明确排除在清单之外。对函数的排除行为会由于诸多原因而被激发。例如，函数排除行为可基于预先确定的商业原因而发生，其中，特定文件或函数由于安全考虑或知识产权保护而不会伴随最终部署的文件而分发。此外，函数可能会因为防止对已知的非必要函数的无意包含而被加以排除。这些函数偶尔会因为一些依赖确定机制的内在不严密性而不经意间被包含到必要函数清单中（步骤 504）。最后，“后绑定模块化执行引擎”典型地包含一个与模块化后绑定执行引擎相关的后绑定函数的固定的基本集合。准确地说，这些函数的基本集合在安装执行引擎的任何地方都是可用的，因此，在已部署的应用中并不需要包含这些函数。采用当前的排除机制，最终部署的应用在大小上会显著降低，并更容易被用户掌握。用户可以在步骤 504 和 508 提供输入，在上述步骤中用户明确地声明依赖性并明确地声明排除。此外，可由用户实现对必要函数列表的全部或局部控制。

[0157] 依赖性分析器的输出是包含有该应用的后绑定函数的最终清单。这些函数由编译器结合用户定义的数据（如输入数据和分路选择）一起使用，以确定应当生成哪些接口函数以及所生成的函数的形式。

[0158] 所生成的先绑定接口函数 408（图 4）与可选的用户编写的先绑定语言代码 416 以及数据类型创建 API 418 结合在一起，以生成先绑定应用或组件 414。

[0159] 在运行时最终用户应用或组件 414 调用数据类型创建 API 418 以创建数据，所创建的数据与包含在外部环境中的数据是后绑定环境兼容的。在本实施例中，所述的外部环境是一种先绑定编程环境。数据类型创建 API418 然后向最终用户应用或组件 414 标识这一新创建的数据。这一数据标识可以采用本领域普通技术人员所知的“句柄 (handle)”或“代理”来实现。

[0160] 最终用户应用程序或组件 414 然后调用先绑定接口函数 408，该函数依次调用模

块化后绑定执行引擎 420。然后，从外部环境数据所创建的兼容数据、以及先绑定接口函数 408 的名称被传递到模块化执行引擎 420，使得模块化执行引擎能够将该数据和函数名称结合起来（即后绑定），以使得从包含在 CTF 档案 410 中的集合中选择合适的函数。模块化后绑定执行引擎 420 的结果然后被送回到最终用户应用或组件 414。

[0161] 本发明的示意性实施例提供了使得后绑定函数能够从先绑定环境中被调用的数据类型。这些数据类型允许先绑定语言来表示用后绑定语言表示的数据类型。这些数据类型可以成为提供给用户的接口库中的一部分。

[0162] 与编译器 406 的每次调用相关的还包括加密 / 解密模式。在一个实施例中，在调用编译器时，生成唯一的编密 / 解密密钥。该密钥可以采用公钥加密的方法进行保护，如美国政府所采用的高级加密标准 (AES)。本领域的普通技术人员很容易理解到，现有的其它各种加密 / 解密方法可用于本发明。在运行时，该唯一的密钥被解密并用于对 CTF 档案中所需要的文件进行解密。在一个实施例中，对文件的解密仅仅发生在该文件被需要的时候，从而防止将已解密信息存储在磁盘上。

[0163] 诸如上述的加密 / 解密模式的使用，为由那些创建应用所开发的代码提供了保护。由于在 CTF 档案中所使用的文件是加密的，因此这些文件在被共享和分配的同时也能对包含在其中的内容提供保护。例如，一个包含有公司商业秘密的文件可以以加密的方式被分发，使得这一交易秘密不会披露给用户。此外，由于在应用中所使用的文件被加密，因此终端客户无法轻易地对文件进行修改从而影响程序的执行。进一步的，由于文件的加密和解密是基于编译器所生成的唯一密钥，因此用于一个应用中的文件无法被用到另一个应用之中。最后，利用当前加密 / 解密机制可实现对盗版软件的追踪，因为能够分析每个生成的应用程序而确定该应用是由哪个编译器许可证密钥所生成的。因此，一旦知道编译器许可证密钥，就有可能在许可证清单中将该许可证密钥与被许可人清单进行对照参考，从而确定盗版软件的来源。

[0164] 图 6 是一个将后绑定函数调用集成到后绑定程序环境 600 中所需的组件的示意性实施例。在一个实例中，后绑定程序环境为 LISP 程序环境，而后绑定函数为 MATLAB® 函数。上述对后绑定函数和语言的选择仅仅作为描述的目的，因为本领域的普通技术人员很容易理解，可选用各种替代的现存的后绑定函数及语言。

[0165] 在此示意性实例中，当后绑定函数为 MATLAB® 函数时，MATLAB® 编译器生成后绑定接口函数 604 和 606，这些函数是存在于后绑定环境 600 中的后绑定函数。在运行时，这些接口函数 604 和 606 被应用于利用模块化后绑定执行引擎 602 调用后绑定函数。正如图 6 中所进一步描述的，后绑定接口函数 604 和 606 可能会引发执行多于一个的后绑定函数（参见 608 和 610）。可能存在这样一些情况，其中一个后绑定接口函数与一个单独的后绑定函数相关联，而且也可能存在其它情况，即多个后绑定函数与一个单独的后绑定接口函数相关联。此外，根据提供给接口函数的输入类型，每一调用操作可导致对相关集合中不同的后绑定函数的调用。

[0166] 在一个实施例中，所述的接口机构可以是应用程序接口 (API)，该 API 具有可以从后绑定应用调用的函数。可选择地，该 API 可以是不需要辅助程序即可执行的独立可执行程序。此外，接口函数可以是软件或硬件组件。

[0167] 模块化后绑定执行引擎 602 提供在运行时执行函数的机制，其中所述函数存在于

后绑定环境中。在一个MATLAB®环境中，模块化后绑定执行引擎可以是MATLAB®组件运行引擎，该引擎能够执行 M- 文件、MEX- 文件以及其它MATLAB®可执行内容。本发明并不仅仅局限于MATLAB®应用以及MATLAB®组件运行引擎，也可以用很多替代的后绑定环境和执行引擎来实现。

[0168] 模块化后绑定执行引擎 602 跟随后绑定环境 600 中的后绑定接口函数调用 604 或 606 的执行，模块化后绑定执行引擎 602 既可以保持运行以使得后续函数在后绑定环境中的执行随时发生，但也可以终止。

[0169] 本发明已通过举例的方式加以说明，本领域的普通技术人员应当理解，在不脱离本发明技术方案的精神的前提下，可以对所描述的实施例进行修改和变动。上述实施例的各个方面与特征可以合并使用。所描述的实施例仅用于示意而不应被理解为限制性的。本发明的范围应当由附属的权利要求进行限定，而不是由前述的说明，对本发明的技术方案进行修改或者等同替换，均应涵盖在本发明的权利要求范围当中。

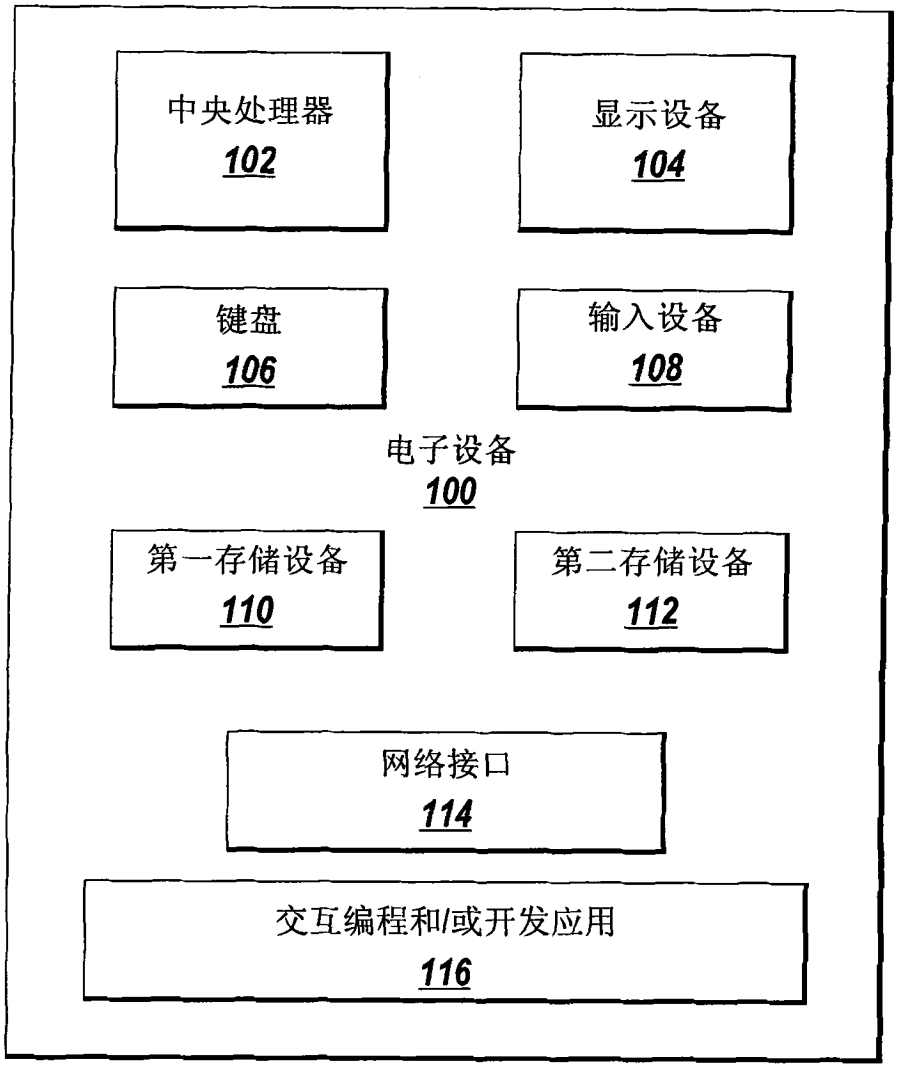

中央处理器
102
显示设备
104
键盘
106
输入设备
108
电子设备
100
第一存储设备
110
第二存储设备
112
网络接口
114
交互编程和/或开发应用
116

图 1

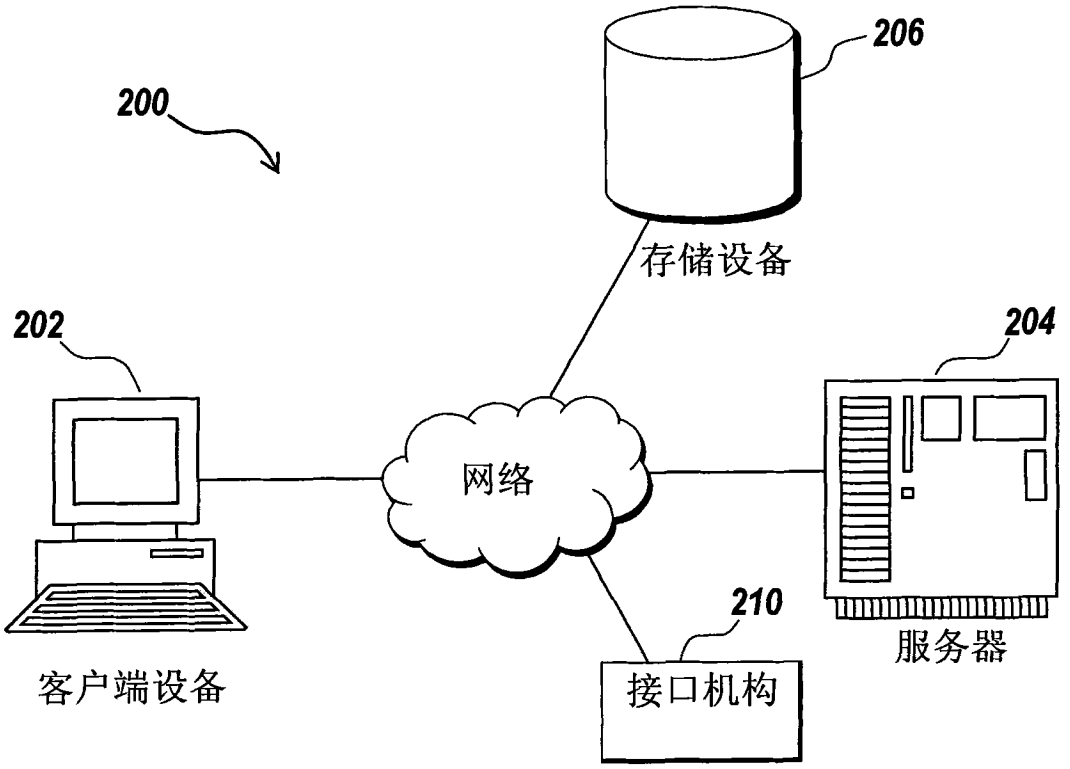

200
206
存储设备
202
204
网络
客户端设备
服务器
210
接口机构

图 2

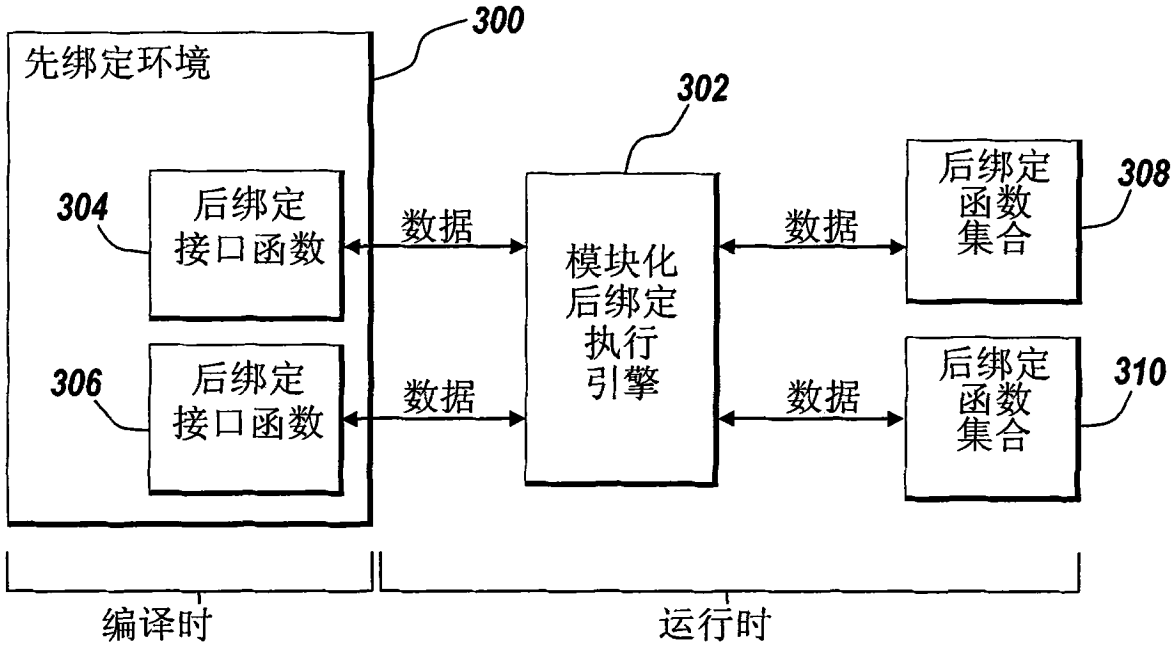

300
先绑定环境
302
304
后绑定
接口函数
数据
模块化
后绑定
执行
引擎
数据
后绑定
函数
集合
308
306
后绑定
接口函数
数据
数据
后绑定
函数
集合
310
编译时
运行时

图 3

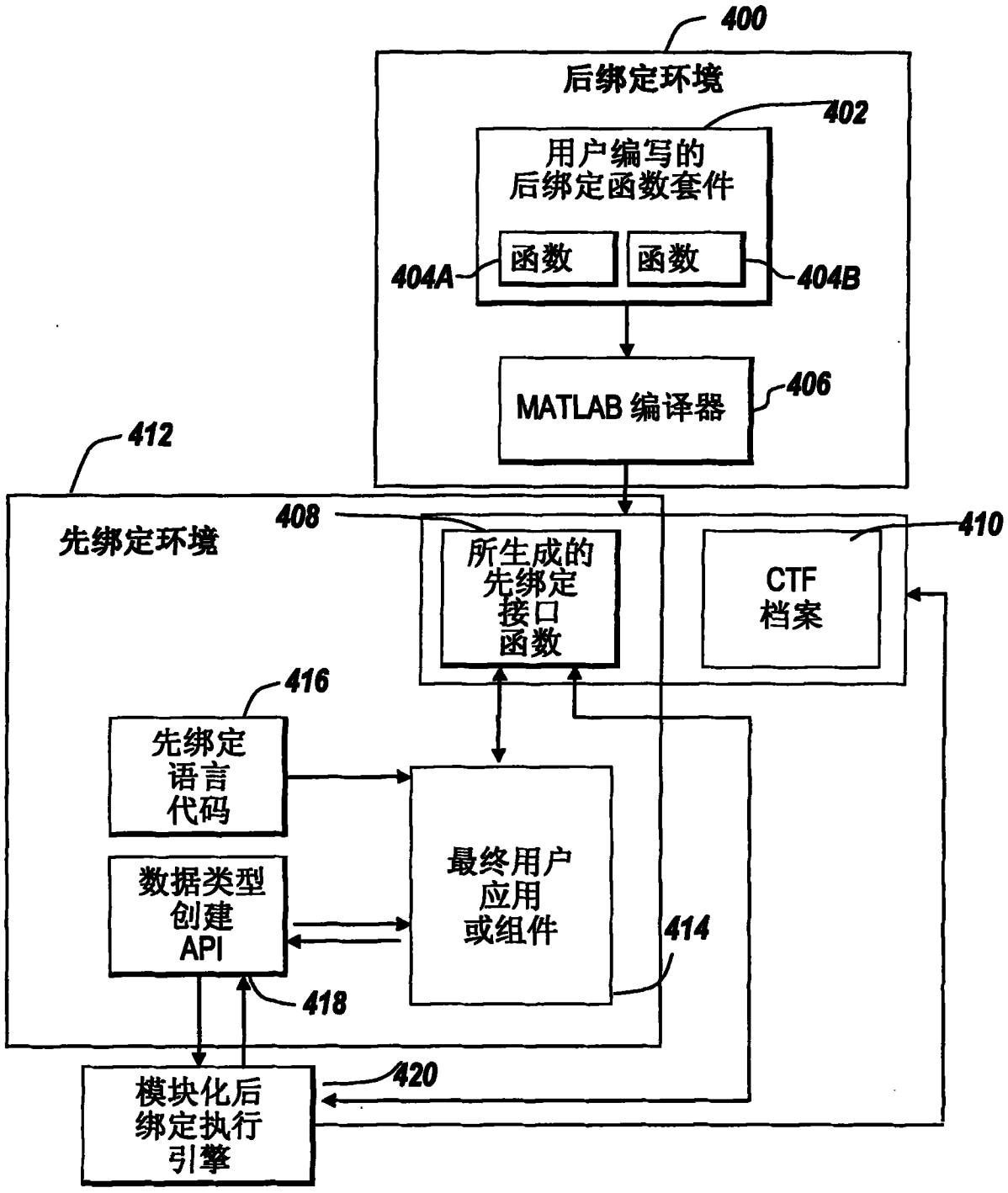

400
后绑定环境
402
用户编写的
后绑定函数套件
函数
函数
404A
404B
MATLAB 编译器
406
412
先绑定环境
408
所生成的
先绑定
接口
函数
410
CTF
档案
416
先绑定
语言
代码
数据类型
创建
API
最终用户
应用
或组件
414
418
420
模块化后
绑定执行
引擎

图 4

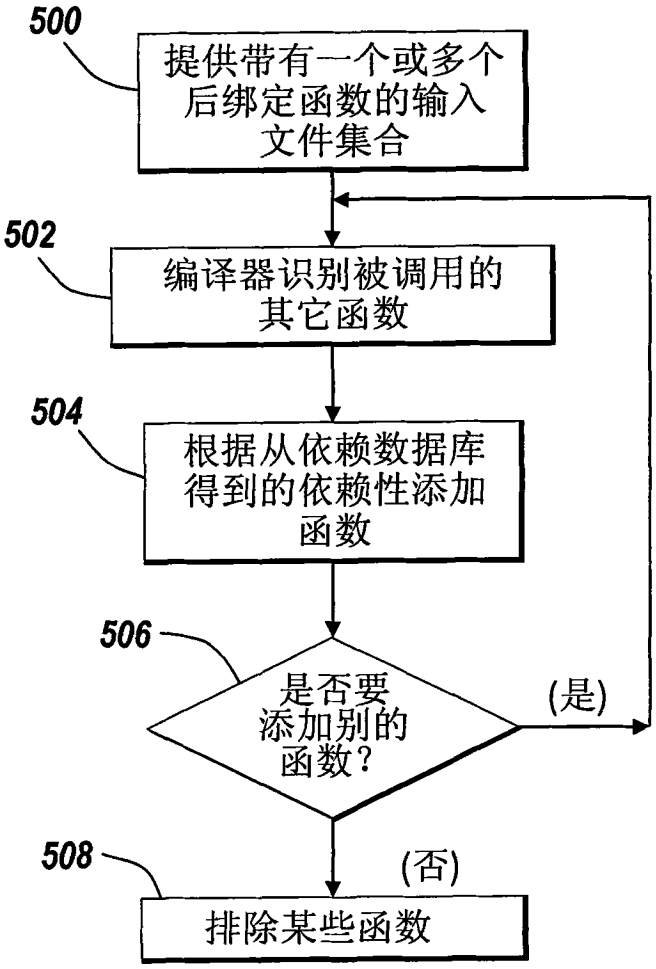

500
提供带有一个或多个后绑定函数的输入文件集合
502
编译器识别被调用的其它函数
504
根据从依赖数据库得到的依赖性添加函数
506
是否要添加别的函数?
(是)
(否)
508
排除某些函数

图 5

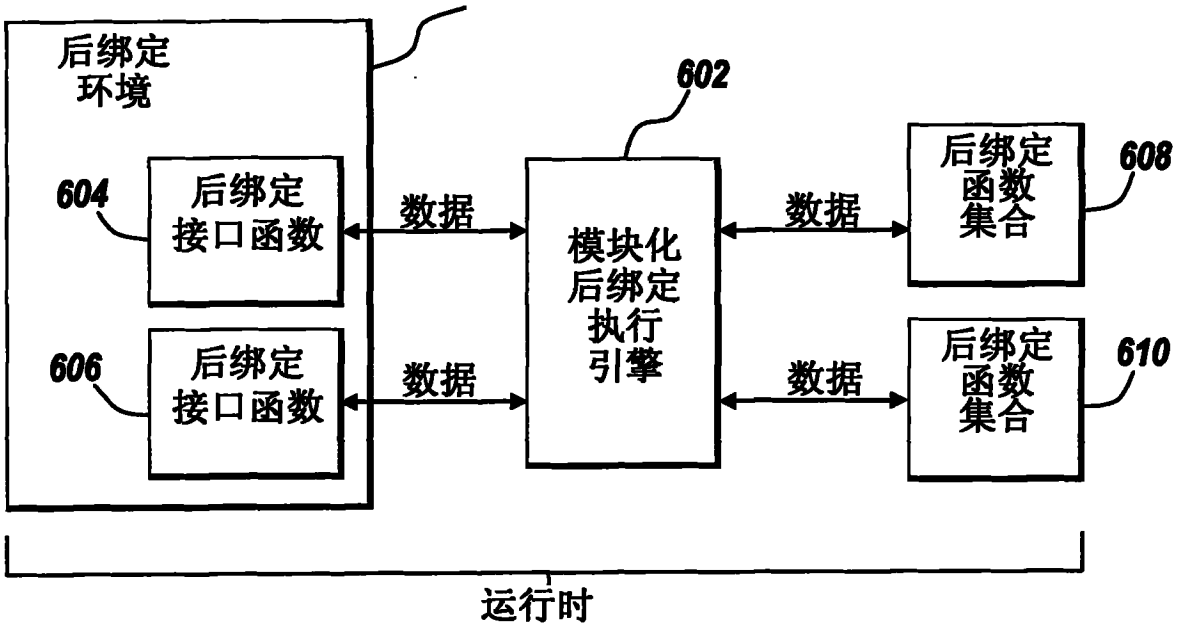

后绑定
环境
604
后绑定
接口函数
606
后绑定
接口函数
数据
数据
602
模块化
后绑定
执行
引擎
数据
数据
后绑定
函数
集合
608
后绑定
函数
集合
610
运行时

图 6