(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 106445487 A
(43)申请公布日 2017.02.22

(21)申请号 201610438015.5

(22)申请日 2016.06.17

(30)优先权数据
15305949.8 2015.06.19 EP

(71)申请人 国立民用航空学院
地址 法国图卢兹

(72)发明人 S·查蒂

(74)专利代理机构 永新专利商标代理有限公司 72002
代理人 刘瑜 王英

(51)Int.Cl.
G06F 9/44(2006.01)

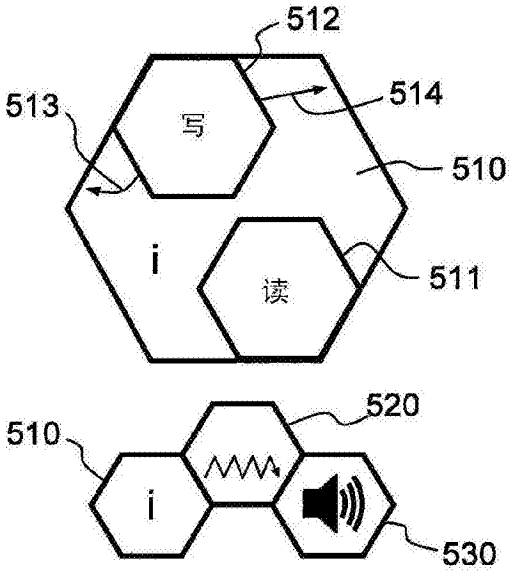
权利要求书2页 说明书27页 附图18页

(54)发明名称

用于控制交互式组件的处理单元、软件以及方法

(57)摘要

本发明涉及交互式软件的开发、控制和执行。本发明的交互式组件被配置为使能交互式数字系统与交互式数字系统的环境之间的定义的交互。该交互式组件包括第一子组件，其定义第二交互式组件与第三交互式组件之间的耦合。所述第一子组件被配置为，当由所述交互式数字系统执行时，生成以所述第二交互式组件的激活为条件的第三交互式组件的激活，所述激活使能所述定义的交互。



CN 106445487 A

1.一种交互式数字系统的交互式组件，所述交互式组件被配置为使能所述交互式数字系统与所述交互式数字系统的环境之间的定义的交互，所述交互式组件包括第一子组件，

其中：

-所述交互式组件和所述第一子组件中的一者定义第二交互式组件与第三交互式组件之间的耦合；

-所述交互式组件和所述第一子组件中的所述一者被配置为，当由所述交互式数字系统执行时，生成以所述第二交互式组件的激活为条件的所述第三交互式组件的激活，以所述第二交互式组件的激活为条件的所述第三交互式组件的所述激活使能所述定义的交互。

2.如权利要求1所述的交互式组件，其中，所述第二交互式组件是以下中的一者：所述交互式组件、所述交互式组件的另一子组件、所述交互式数字系统的应用和操作系统中的一者的子组件、以及在来自所述数字系统的所述环境的输入时激活的交互式组件。

3.如权利要求1所述的交互式组件，其中，所述第三交互式组件是以下中的一者：所述交互式组件的另一子组件、所述交互式数字系统的应用和操作系统中的一者的子组件、以及当激活时向所述数字系统的所述环境产生输出的交互式组件。

4.如权利要求1、3所述的交互式组件，其中，所述第一子组件是由所述数字系统上板载的处理器执行的。

5.如权利要求1至3中的一项所述的交互式组件，其中，被配置为生产所述定义的交互由如下的连续迭代引起：

-以下中的一者：将子组件添加到所述交互式组件、以及将所述交互式组件的子组件替换成另一交互式组件；

-检查所述交互式组件是否生产所述定义的交互。

6.如权利要求1至3中的一项所述的交互式组件，包括代表变量和算术运算中的一者的子组件。

7.如权利要求1至3中的一项所述的交互式组件，所述交互式组件是应用的一部分，其中，所述第二交互式组件和所述第三交互式组件中的一者是另一应用的一部分。

8.一种用于执行交互式数字系统的交互式组件的方法，所述交互式组件被配置为使能所述交互式数字系统与所述交互式数字系统的环境之间的定义的交互，所述交互式组件包括第一交互式组件，

其中：

-所述交互式组件和所述第一子组件中的一者定义第二交互式组件与第三交互式组件之间的耦合；

所述方法包括：

-当执行所述交互式组件和所述第一子组件中的所述一者时，生成以所述第二交互式组件的激活为条件的所述第三交互式组件的激活，以所述第二交互式组件的激活为条件的所述第三交互式组件的所述激活使能所述定义的交互。

9.一种存储在非暂时性计算机可读介质上的计算机程序，被配置为执行交互式数字系统的交互式组件，所述交互式组件被配置为使能所述交互式数字系统与所述交互式数字系统的环境之间的定义的交互，所述交互式组件包括第一交互式组件，

其中：

-所述交互式组件和所述第一子组件中的一者定义第二交互式组件与第三交互式组件之间的耦合；

-所述计算机程序包括代码指令，当执行所述交互式组件和所述第一子组件中的所述一者时，所述代码指令用于生成以所述第二交互式组件的激活为条件的所述第三交互式组件的激活，以所述第二交互式组件的激活为条件的所述第三交互式组件的所述激活使能所述定义的交互。

10.一种交互式数字系统的处理器，处理器被配置为执行所述交互式数字系统的交互式组件，所述交互式组件被配置为使能所述交互式数字系统与所述交互式数字系统的环境之间的定义的交互，所述交互式组件包括第一交互式组件，

其中：

-所述交互式组件和所述第一子组件中的一者定义第二交互式组件与第三交互式组件之间的耦合；

-所述处理器被配置为，当执行所述交互式组件和所述第一子组件中的所述一者时，生成以所述第二交互式组件的激活为条件的所述第三交互式组件的激活，以所述第二交互式组件的激活为条件的所述第三交互式组件的所述激活使能所述定义的交互。

11.一种用于生产交互式数字系统的交互式组件的方法，所述交互式组件被配置为使能所述交互式数字系统与所述交互式数字系统的环境之间的定义的交互，所述方法包括：

-将第一子组件添加到所述交互式组件中；

其中：

-所述交互式组件和所述第一子组件中的一者定义第二交互式组件与第三交互式组件之间的耦合；

-所述交互式组件和第一子组件中的所述一者被配置为，当由所述交互式数字系统执行时，生成以所述第二交互式组件的激活为条件的所述第三交互式组件的激活，以所述第二交互式组件的激活为条件的所述第三交互式组件的所述激活使能所述定义的交互。

用于控制交互式组件的处理单元、软件以及方法

技术领域

[0001] 本发明涉及交互式软件的开发和执行。更具体地，本发明涉及用于控制应用的交互式组件的行为的处理单元、软件和方法。

背景技术

[0002] 交互式软件是指除了执行计算之外还接受来自人类、物理环境或其它机器的输入的软件。几乎所有现代的软件都是交互式软件。例如，文本编辑器、视频游戏、互联网浏览器、智能手机应用以及飞机座舱显示器的软件都是交互式软件。计算机操作系统、连接的对象的固件以及网络服务器也是交互式软件。

[0003] 多年来，专业的编程技术已经用于运行于个人计算机上的图形交互软件，其中输入基本上利用鼠标和键盘来执行。现今，新的人类输入和输出技术、智能手机、平板设备、网络连接和连接的对象的普及已经广泛地增加了用于设计交互式软件的可能组合的数量。例如，在平板计算机上，输入现在能够利用触摸敏感表面、诸如空气指针的连接的对象以及诸如陀螺或加速度计之类的内部传感器来进行输入。更复杂的交互式应用能够由运行于计算机中、运行于触摸敏感桌面显示器的固件中、运行于互联网服务器中以及运行于世界范围内的多个传感器中的多个交互式软件组件构成。因此，有必要提供更广泛地涵盖交互式软件的编程技术。

[0004] 软件组件是能够独立地开发和汇编以产生软件产品的指令的集合。

[0005] 软件组件的互操作性是两个或更多个软件组件互连且一起正确地起作用的能力。当在不增加适配层的情况下存在将组件组合的句法正确的方式时以及当它们的语义直接兼容时，组件是可互操作的。互操作性是软件开发中主要关心的问题，因为其规定了软件组件如何能够跨多个应用复用和适配，以及在设计应用的过程中组件何时能够互换。互操作性也是创新的有利条件，因为其允许通过先前未使用的方式实现组件的连接。例如，当使得加速度计与图形能互操作且与触摸区域可互换时，借助所述平板设备的方位来驱动图形对象在平板设备的显示器上的位置成为可能。在程序执行期间也能够利用互操作性和互换性，产生编程者无需显式地且穷尽地描述的连接。例如，游戏可被编程为在会话期间随机地改变用户必须使用哪个输入设备来控制对象，或者将哪个变换法应用于该输入。

[0006] 软件体系结构是用于将软件组件组织在较小的可复用组件中的规则的集合。软件体系结构涉及到如何组织软件执行和数据二者，以及它们如何在软件组件的互操作性和互换性方面起作用。当复用所述组件来创建较大的组件时能够使用用于创建任意组件的相同的规则时，给定的软件体系结构是递归的。递归性有助于管理软件复杂度，因为编程者能够在将组件与其它组件汇编时选择忽略组件的内部复杂度。递归性还利于不同粒度级的软件组件之间的互操作性和互换性。例如，自动地计算图形组件的尺寸和位置的图形布局组件依赖于递归体系结构以便于操作于全部的图形组件之上，无论是像矩形那样简单，还是像整个对话框那样构成。如果相同的软件功能能够由具有不同级别的内部复杂度的输入所触发，例如简单的键盘按键、图形按钮以及对话框的输出，也需要递归体系结构。

[0007] 软件体系结构能够得到编程规约或者编程语言支持。例如，面向对象的语言有助于将软件组织在组件中，该组件既是数据单元，又是执行单元。根据在这些类型的软件中如何更自然地组织数据和执行，已经提出用于不同类型的软件的不同的体系结构。

[0008] 传统的编程语言，诸如C、C++、Lisp或Java，已经通过添加利于互操作性的特征而从集中于计算的编程语言获得。例如，函数式编程语言将函数定义为软件组件的最重要的类别，且甚至将数据变量视为不含变元的函数。该基于单个构造的递归体系结构利于在软件中创建可互操作组件，其中每个单个组件的作用是实现全局结果的计算的一部分。类似地，通过将计算和数据聚集在对象中，面向对象的语言利于在软件中组件的互操作性，其中每个单个的组件必须存储数据以便于贡献于全局计算。面向对象的语言还通过支持类继承而有利于互操作性和复用。对于更复杂的情形，已经提出了设计模式作为附加的方法以用于将通过函数调用或继承关系不完全地描述其关系的软件组件进行互连。

[0009] 交互式软件在影响软件体系结构的若干方式上不同于面向计算的软件。

[0010] 在执行方面，计算程序具有起始和结束，并且执行由朝向结束的步骤和循环构成。相反，交互式软件等待输入且取决于接收到的输入来触发反应行为或计算。

[0011] 在数据管理方面，交互式软件也不同。维持组件状态和数据值是交互式软件的主要关注点，而其经常被认为是计算软件的副作用。

[0012] 交互式软件还展现出如何组合软件组件的更广泛的方式。在面向计算的软件中，函数及其变元之间的关系已经被证明作为大多数情况下足够用的组合手段。可替代地，命令式编程语言提供了能够用于将计算程序中的编程指令互连的若干控制结构(顺序、循环和测试)。在交互式软件中，能够存在大量的附加情形。例如，图形组件能够集群在场景图形中，动画能够组织成并行地执行，图形对象能够与对话组件的各种状态相关联，指令能够被定义为当外部事件发生时执行，图形对象的视觉属性能够被定义为随着物理环境中测得的数据的值而连续地变化。

[0013] 传统的编程语言已经接收到支持交互式软件的执行的扩展。例如，等待函数支持通过外部输入进行执行控制，而线程支持动作的并行执行。通过这些扩展，它们理论上可以支持交互式软件的开发。然而，组件的可能的输入、状态和组合的增加显著地增加了给定应用的可能的执行的数量。如果该多个可能的执行是利用通常的控制结构来编程的，则软件复杂度增加:程序行为的任何修改需要多个组件的变化，从而限制了在初始设计阶段之后做出选择的能力。

[0014] 连同该软件复杂度的增加一起，软件组件的互操作性趋于降低，并且软件开发和验证变长，成本高，且易于出错。而且，变得难以在适当的抽象级分析软件属性，并且仅交互式软件的特定类能够经历在一些工业领域要求的软件核验过程。而且，变得难以设计利于软件开发的编程工具，这是因为不存在适当地捕获软件结构的视觉表征。

[0015] 已经提出了各种软件模式来降低利用传统编程语言开发的交互式软件的复杂度。每种模式解决了复杂度的一个原因。

[0016] 最常见的软件模式是回调函数及其变体，诸如控制反转(Inversion of Control)模式以及信号/槽(Signal/Slot)模式，它们旨在限制由于外部控制诱发的复杂度。在该模式中，编程者能够注册给定的函数，以使其当满足一些条件时被调用，诸如给定类型的外部输入的出现。在该模式的一些实现方式中，向回调函数传递包含了关于什么导致调用的信

息的名为“事件”的数据结构。

[0017] 已经提出了各种软件模式来通过根据软件组件的作用来组织软件组件以及定义它们如何能够组合来抑制软件复杂度。例如，通过模型查看控制器(Model-View-Controller)模式，应用组件由分别负责管理数据和计算、可视化数据、以及管理用户输入的三个子组件构成。呈现抽象控制(Presentation-Abstraction-Control)以及模型查看查看模型(Model-View-View Model)模式具有类似的结构。扩展的场景图形是另一类模式，从图形场景图形得到，其中能够添加各种类型的非图形软件组件作为图形的节点，以使得在其图形结构上对准应用的软件体系结构。

[0018] 已经提出了其它模式来组织交互式软件中的控制流，并且弥补由编程语言提供的控制结构的局限性。例如，Harel,D.Statecharts:A visual formalism for complex systems.Science of Computer Programming 8,3(1987年6月),pp.231-274公开了能够被组合以描述交互式系统的状态表(Statecharts)、层级状态机组件。Myers,A new model for handling input,ACM Transactions on Information Systems,第3期第8卷,1990年7月,pp 289-320公开了一种能够被适配以在各种类型的软件组件中编程交互的状态机组件。在特定事件发生时执行状态之间的迁移，并且软件的外观和行为取决于软件的所述状态。Myers等人,Garnet:Comprehensive Support for Graphical,Highly-Interactive User Interfaces.I15Computer,第23卷第11编号(1990年11月)公开了以数据流方式在组件之间传播值的单向约束系统。Dragicevic等人,Support for input adaptability in the ICon toolkit.Proc.ICMI'04,ACM Press(2004),pp.212-219公开了一种能够用于对输入管理编程的数据流系统。Nigay等人,A design space for multimodal interfaces:concurrent processing and data fusion,Proceedings of INTERCHI'93,ACM(1993),pp172-178公开了一种用于将来自多个输入的事件和状态组合的多模态融合模式。Calvary等人,COMET(s),A Software Architecture Style and an Interactors Toolkit for Plastic User Interfaces.Interactive systems.Design,Specification,and Verification,Lecture Notes in Computer Science,Vol.5136.Springer-Verlag(2008),pp 225-237公开了一种用于管理软件对计算平台和执行上下文的变化的适配的模式。

[0019] 然而，这些解决方案中的每个均仅解决复杂度的一个原因，并且在最具交互式的软件中它们需要组合以解决全部的原因。这构成了软件结构中的异质的源，因为这些模式不可互操作，并且利用它们所创建的组件既不可互操作，又不可互换。例如，数据流系统中的值变化不能直接地用作回调系统中的事件或状态机中的迁移(transition)。必须利用每种编程语言所提供的基本机制编写适配代码以组合它们，并且这引入了额外的异质。这在互操作性方面不能令人满意并且引入了新的复杂度，具有之前所描述的所有后果。

[0020] 已提出部分解决方案以使得这些软件模式可互操作。例如，Chatty.S,Extending a graphical toolkit for two-handled interaction.Proc.UIST'94ACM(1994),pp.195-204公开了一种用于将状态机和数据流组合的方法，其中当状态变化时，数据流的配置变化。Jacob等人,A Software Model and Specification Language for Non-{WIMP}User Interfaces,ACM Transactions on Computer-Human Interaction,6:1(1999),pp 1-46,公开了一种类似的方法。Appert等人,FlowStates:prototypage d′applications

interactives avec des flots de données et des machines à états,Proceedings of the IHM 2009会议,ACM出版,pp.119-128,公开了另一种组合状态机和数据流的方法,利用Java代码来执行适配。Elliott等人,Functional Reactive Animation,Proceedings of the International Conference on Functional Programming(1997),pp 263-273,公开了函数式反应编程,允许利用相同的句法以表达传统计算和数据流两者的功能语言的执行语义的改变。Chatty等人,Revisiting visual interface programming:Creating GUI tools for designers and programmers.Proc.UIST'04,ACM(2004),pp.267-276公开了一种用于以同质方式汇编图形和异质行为组件的扩展场景图形的应用。Prun等人,Towards Support for Verification of Adaptative Systems with djnn,The Seventh International Conference on Advanced Cognitive Technologies and Applications,pp 191-194公开了具有更大数量控制结构模式可用以及鼓励编程者利用树结构的益处的交互式组件定义的类似解决方案。但是,树中的组件仍基于不同的机制,并且在它们可能的组合方面受限制。例如,这些解决方案不提供用于将反应行为与计算组合、用于将数值变量与具有数值状态的状态机互换、或者用于通过添加条件到迁移来细化状态机的简单方法。传统编程语言的使用仍需要表达应用的部分,并且组件的执行模型仍在极大程度上取决于底层的传统编程语言的执行模型。

[0021] 上述解决方案中的任一个均不能确保能够利用单组同质和可互操作组件来创建任何软件应用。另外,这些解决方案中的大多数专用于图形交互软件,并且没有一个足够可扩展而引入新的交互模态和新的交互式样所要求的新的控制结构。全部均需要在程序中使用来自传统编程语言的提供缺少的控制结构、体系结构模式或者甚至是功能的指令,在复杂度、互操作性、复用、认证等方面具有之前所描述的后果。

[0022] 已经提出了专用语言来利用同质组件对交互式软件的类进行编程。例如,XUL,XAML和QML语言提出了用于在用户界面中汇编图形组件的递归体系结构。然而,它们不能容易地扩展到除了图形用户界面之外的其它使用,它们提供了有限范围的控制结构,并且它们所能产生的应用和交互是老套的。通过它们产生非WIMP(窗口、图标、菜单、指针)应用需要使用通用语言,并且它们不能用作用于交互式软件的通用解决方案。

[0023] 已经创建了同步数据流语言来支持诸如自动控制系统的交互式软件的创建。N.Halbwachs等人,The Synchronous Data Flow Programming Language LUSTRE.Proc.I15 1991Vol.79,No.9公开了一种同步数据流语言,LUSTRE。已经开发了对LUSTRE的扩展以实现用户界面。在LUSTRE中,输入用于控制数据流。另外,LUSTRE代码能够用于定义状态机。然而,LUSTRE中状态机与数据流之间的互操作性如之前所述的解决方案那样受限制。另外,一旦定义,用另一数据流取代一个数据流是非常难的。不支持新的控制结构的定义。

[0024] 本发明的目的是,通过定义确保软件组件的同质性、复杂度管理、互操作性和互换性、支持来自交互式软件的计算和全部常规控制结构、能扩展到全部的交互模态和交互式样并且支持交互式软件的新用途所要求的新控制结构的创建的用于交互式软件的递归体系结构,来克服现有技术的这些局限性。

发明内容

[0025] 为此目的，本发明公开了一种交互式数字系统的交互式组件，所述交互式组件被配置为使能在所述交互式数字系统与交互式数字系统的环境之间的定义的交互，所述交互式组件包括第一子组件，其中交互式组件和第一子组件中的一者定义了在第二交互式组件与第三交互式组件之间的耦合；所述交互式组件和第一子组件中的所述一者被配置为，当由交互式数字系统执行时，生成以第二交互式组件的激活为条件的第三交互式组件的激活，以第二交互式组件的激活为条件的第三交互式组件的所述激活使能所述定义的交互。

[0026] 有益地，所述第二交互式组件是以下中的一者：交互式组件、交互式组件的另一子组件、交互式数字系统的应用和操作系统中的一者的子组件、以及在来自数字系统的环境的输入时激活的交互式组件。

[0027] 有益地，第三交互式组件是以下中的一者：交互式组件的另一子组件、交互式数字系统的应用和操作系统中的一者的子组件、以及当被激活时对数字系统的环境产生输出的交互式组件。

[0028] 有益地，第一子组件由数字系统上装载的处理器来执行。

[0029] 有益地，交互式组件被配置为产生定义的交互，由于如下连续的迭代而实现：将子组件添加到交互式组件中以及将交互式组件的子组件替换成另一交互式组件中的一者；检查所述交互式组件是否产生定义的交互。

[0030] 有益地，交互式组件包括代表变量和算术运算中的一者的子组件。

[0031] 有益地，所述交互式组件是应用的一部分，并且第二交互式组件和第三交互式组件中的一者是另一应用的一部分。

[0032] 本发明还公开了一种执行交互式数字系统的交互式组件的方法，所述交互式组件被配置为使能在交互式数字系统与交互式数字系统的环境之间的定义的交互，所述交互式组件包括第一交互式组件，其中交互式组件和第一子组件中的一者定义第二交互式组件与第三交互式组件之间的耦合，所述方法包括：当执行所述交互式组件和第一子组件中的所述一者时，生成以第二交互式组件的激活为条件的第三交互式组件的激活，以第二交互式组件的激活为条件的第三交互式组件的所述激活使能所述定义的交互。

[0033] 本发明还公开了一种存储在非暂时性的计算机可读介质上的计算机程序，被配置为执行交互式数字系统的交互式组件，所述交互式组件被配置为使能交互式数字系统与交互式数字系统的环境之间的定义的交互，所述交互式组件包括第一子组件，其中交互式组件和第一子组件中的一者定义第二交互式组件与第三交互式组件之间的耦合，其中所述计算机程序包括代码指令，当执行所述交互式组件和第一子组件中的所述一者时，所述代码指令用于生成以第二交互式组件的激活为条件的第三交互式组件的激活，以第二交互式组件的激活为条件的第三交互式组件的所述激活使能所述定义的交互。

[0034] 本发明还公开了交互式数字系统的处理器，所述处理器被配置为执行所述交互式数字系统的交互式组件，所述交互式组件被配置为使能交互式数字系统与交互式数字系统的环境之间的定义的交互，所述交互式组件包括第一子组件，其中所述交互式组件和所述第一子组件中的一者定义第二交互式组件与第三交互式组件之间的耦合，并且所述处理器被配置为，当执行所述交互式组件和第一子组件中的所述一者时，生成以第二交互式组件的激活为条件的第三交互式组件的激活，以第二交互式组件的激活为条件的第三交互式组件的所述激活使能所述定义的交互。

[0035] 本发明还公开了用于产生交互式数字系统的交互式组件的方法，所述交互式组件被配置为使能交互式数字系统与交互式数字系统的环境之间的定义的交互，所述方法包括将第一子组件添加到所述交互式组件中，其中交互式组件和第一子组件中的一者定义了第二交互式组件与第三交互式组件之间的耦合，并且所述交互式组件和第一子组件中的所述一者被配置为，当由交互式数字系统执行时，生成以第二交互式组件的激活为条件的第三交互式组件的激活，以第二交互式组件的激活为条件的第三交互式组件的所述激活使能所述定义的交互。

[0036] 本发明使得交互式应用的开发简易。

[0037] 本发明容许在单一基元下的应用的操作和控制结构的定义。

[0038] 本发明容许开发其组件可互换的应用。

[0039] 本发明使得应用的输入和交互的管理简易。

附图说明

[0040] 根据下面对多个示例性实施例及其附图的描述，本发明将得到更好的理解，并且其各种特征和优点将是显然的，在附图中：

[0041] -图1显示出本发明的多个实施例中的耦合的三个示例，分别是在按键与蜂鸣器之间、定时器与蜂鸣器之间，以及定时器与LED之间；

[0042] -图2显示出在本发明的多个实施例中包括子组件树的组件；

[0043] -图3显示出在本发明的实施例中由称为绑定的控制结构组件创建的耦合的示例；

[0044] -图4显示出在本发明的实施例中鼠标与蜂鸣器之间的绑定的示例；

[0045] -图5显示出在本发明的实施例中属性作为其源的绑定；

[0046] -图6显示出在本发明的多个实施例中称为分配的控制结构组件的三个示例；

[0047] -图7显示出在本发明的多个实施例中称为连接器的控制结构组件的示例；

[0048] -图8显示出作为连接器组件的变体且对于其输入中的两个值变化的任何序列仅传播一个值变化的组件；

[0049] -图9显示出被构建作为定制组件的按钮组件的XML表示；

[0050] -图10显示出在本发明的多个实施例中用于执行交互式应用和组件的示例性的体系结构，其中组件由作为程序运行于计算机上的解译器执行；

[0051] -图11显示出解译器能够从其装载组件的多个位置；

[0052] -图12显示出根据本发明的用于使处理器执行交互式应用的第二体系结构；

[0053] -图13显示出根据本发明的用于使处理器执行交互式应用的第三体系结构；

[0054] -图14显示出在本发明的多个实施例中用于执行交互式应用和组件的示例性的体系结构，其中组件被编译成用于专门设计为用于执行交互式组件的操作系统的可执行形式；

[0055] -图15显示出在本发明的多个实施例中用于执行交互式应用和组件的示例性的体系结构，其中组件被编译成能够由处理单元直接执行的应用；

[0056] -图16显示出在本发明的多个实施例中用于执行交互式应用和组件的示例性的体系结构，其中组件被编译成处理单元的电路设计；

[0057] -图17显示出在本发明的多个实施例中显示由交互式组件构建的飞机的主飞行显

示器的应用；

[0058] -图18显示出在本发明的实施例中显示主飞行显示器中的警报的组件的XML表示；

[0059] -图19显示出在本发明的实施例中用于设定主飞行显示器中的目标高度的触摸显示器；

[0060] -图20显示出在本发明的实施例中连接用于设定主飞行显示器的目标高度的Phidget的示例。

具体实施方式

[0061] 在本说明书中，将通过与由解译器或编程架构执行的交互式组件相关的示例来描述本发明，所述解译器和编程架构是参考图10和图12来描述的。组件类型的命名是指它们在djnn架构中的名称。可在http://djnn.net/得到的djnn架构是申请人开发的用于设计交互式应用的编程架构，其中利用组件类型中定义的模板来创建组件。然而，本发明能应用于其它的平台和产品，并且技术人员能够容易地定义具有新功能或名称的组件类型以及保持在本发明的范围内的同时创建组件的其它方式。在本说明书中描述的组件类型和实现方式被提供仅作为非限制示例，绝不以任何方式限制如随附权利要求中限定的本发明的范围。

[0062] 根据本发明，能够将任何软件组件生产为交互式组件的集合，并且其执行被定义为所述交互式组件的受控激活。例如，不是限制，下面的实体能够被表示为交互式组件：常量值、存储器变量、数据结构、存储器分配指令、添加指令、指令序列、迭代循环、函数或过程、计算算法、图形对象、对话框、姿势识别系统、鼠标输入设备的软件代理、鼠标输入设备的按钮的软件代理、应用。

[0063] 在本说明书以及在附图中，将名称“组件”解释为“交互式组件”的同义词，除了另外陈述时否则对物理实体或软件环境的任何元素的引用将意指对其软件代理的引用，软件代理是其行为反映所述物理实体或环境的元素的行为的交互式组件。软件的环境是能访问软件的任何可测量现象，包括但不限于硬件设备、传感器测量环境以及其它程序。

[0064] 根据本发明的交互式应用的组件能够处于活跃状态或不活跃状态。一些组件可被激活和去激活。对组件的激活时所执行的操作取决于所述组件。例如，计算算法当被激活时执行其计算，图形对象当被激活时被显示出，而蜂鸣器当被激活时在有限时间内产生声音。

[0065] 一些组件在被激活时自动返回不活跃状态。例如，算术运算非常简短得活跃且立即返回到不活跃状态；函数或动画在其正在运行的同时是活跃的，而当其结束时返回到不活跃状态。其它组件在被激活时保持活跃直到它们被去激活。例如，图形对象在不活跃时不可见，当被激活时变得可见，并且保持活跃和可见直到其被去激活。一些组件总是活跃。例如，在本发明的多个实施例中，存储器槽总是活跃。类似地，一些物理实体的软件代理在它们存在时总是活跃，例如在本发明的多个实施例中当被插到计算机上时鼠标总是活跃。

[0066] 组件的激活能够由组件与其它组件或者与软件的环境的交互来控制。相互地，组件的激活能够控制其与软件的环境以及与其它组件的交互。根据本发明，这两个控制的相互的情形是由组件对之间的耦合以及由组件与软件环境之间的耦合来表示的。耦合代表了激活之间的因果关系。

[0067] 当在第一组件与第二组件之间存在耦合时，第一组件的激活触发第二组件的激活。第一组件称为耦合的源(source)，而第二组件称为耦合的作用(action)。例如，当键盘

的按键的软件代理与蜂鸣器耦合时，所述软件代理的每一激活触发蜂鸣器的激活。类似地，来自传统编程语言的函数调用能够根据本发明来再制，因为函数的激活由于调用者与所述函数之间的耦合而成为可能。

[0068] 当在软件的环境与组件之间存在耦合时，软件的环境中的一些条件的变化能够触发所述组件的激活。例如，当用户按下鼠标按钮时，鼠标按钮被激活，当用户或操作系统运行程序时，程序被激活，并且温度传感器被定义为使得当外部温度越过定义的阈值时所述温度传感器被激活。

[0069] 当在组件与软件的环境之间存在耦合时，所述组件的激活能够触发软件的环境的变化。例如，图形对象的激活使得所述图形对象出现在显示器上，“刺激”组件的激活在与动物的神经系统连接的电极中产生电刺激。

[0070] 图1显示出在本发明的多个实施例中的耦合的三个示例，分别是在按键与蜂鸣器之间，定时器与蜂鸣器之间，以及定时器与LED之间。组件101是按键的软件代理，组件104是蜂鸣器的软件代理，而组件105是LED的闪烁动画的软件代理。在100中，在按键101与蜂鸣器104之间定义耦合103。当键盘的物理按键被按下时，按键组件101被激活，并且通过耦合，其激活了蜂鸣器组件104，这进而激活了物理声音产生设备。

[0071] 耦合能够存在于任意两个组件之间。这施行了组件的互操作性和互换性。例如，在110中，在与蜂鸣器104的耦合中，按键101由“定时器”组件102替代。在该配置中，当定时器102中限定的延时到期时，蜂鸣器104被激活。在120中，在与按键101的耦合中，蜂鸣器104由闪烁动画105替代。在该配置中，当按键101被按下时，闪烁开始。这些示例证实了本发明中组件的互换性，并且甚至在本发明的简单的实施例中也能够产生效果的多样性。

[0072] 组件能够构造为子组件的集合。组件通过其本身的耦合和激活及其子组件的耦合和激活来与其环境进行交互。组件的环境包括软件的其它组件和软件的环境。例如，鼠标的软件代理通过其按钮和位置跟踪器的耦合激活来与其它组件进行交互，所述按钮和位置跟踪器是所述鼠标的子组件。类似地，存储器槽通过代表了其读和写能力的其子组件的耦合和激活来与其环境进行交互。类似地，图形对象通过代表了表示所述图形对象的位置的存储器槽的读和写能力的子组件的耦合和激活来与其环境进行交互。

[0073] 在本发明的多个实施例中，组件的子组件能够按层级组织在树中。图2显示出包括子组件树的组件。树200包括组件201以及子组件202、203、204、205、206。组件的子组件是树中组件的所有直接子组件，树由子组件的连续级别来定义。这图示在210中，其中显示出组件205和206是组件204的直接子组件。在210中使用的可视表示在本说明书中常用来表示组件的子组件。

[0074] 树结构有益地反映了编程者所具有的关于如何组织软件和硬件设备的自然感知。例如，图形按钮可以是对话框的子组件，鼠标按钮是鼠标的子组件，图形对象的位置是所述图形对象的子组件。

[0075] 树结构还提供了一种利用路径句法来命名和访问组件的子组件的自然的方式。在本发明的一些实施例中，组件的直接子组件能够可选地被赋予相对于所述组件的名称，并且组件能够可选地被赋予作为组件树的根的名称。在图2中，组件210被赋予作为树的根的名称“a”，而组件202、203和204分别被赋予作为组件201的子组件的名称“b”、“c”和“d”。这允许将组件202指定为一般上下文中的“/a/b”，或者在名称被理解为相对于组件201的上下

文中指定为“b”。类似地，组件206在相对于组件204的上下文中称为“f”，在相对于组件201的上下文中称为“d/f”，或者在一般上下文中称为“/a/d/f”。诸如“a/d/f”的名称称为“路径”，并且路径中的单个名称称为路径段。

[0076] 树结构还允许利用XML记法来定义和表示组件。在220中，组件201以XML记法来表示，利用了XML记法表示树的能力。在220中以及在本说明书中使用的全部XML表示中，表示组件的XML标签利用所述组件的类型的名称来书写，并且“id”属性用来显示树中每个组件的名称。例如，组件201是类型A，其名称是“a”，结果其表示为<A id＝”a”>。在组件的XML表示中，当所述组件的内部结构对于相同类型的全部组件总是相同时，可以省去组件的子组件。例如，如果类型D的组件总是包含分别命名为“e”和“f”的类型E和F的两个子组件，则在类型D的所述组件的XML表示中没有必要包含所述子组件。在230中是组件201的表示，其中未显现组件204的结构，而在220中显现所述结构。

[0077] 树结构还有益地容许将子组件替换成另一子组件，从而允许组件的进一步的互操作性和互换性。例如，如果子组件202是按钮，则其能够通过将子组件简单地替换成另一子组件而由另一按钮替代。类似地，如果子组件206是在按下按钮202时执行的函数或操作，则其能够由将执行另一函数或操作的另一子组件来替代。每个可能的输入、输出或操作能够表示为组件。该性质容许组件之间的完全的互换性。

[0078] 在本发明的一些实施例中，能够将环境的元素的软件代理组织为其根在任何应用的树之外的组件树。这允许应用之间的软件代理相互发生关系，并且代表了环境的实际设备或元件具有独立存在性的事实。

[0079] 在本发明的一些实施例中，当定义组件时控制对其子组件的访问从而隐藏所述组件的一些细节是可能的。使得组件的子组件在另一名称下可访问也是可能的，目的是使得所述组件的复用更容易。例如，“/mouse/move/x”子组件可能作为“/mouse/x”被访问。

[0080] 在本发明的一些实施例中，除了树结构以外的其它结构能够定义对树结构的替选或补充以组织组件的子组件。例如，组件中包含的所有子组件能够组织为子组件的关系数据库。这有益地允许对如何根据例如性能约束来存储和管理子组件进行优化。这还允许通过查询语言针对组件的子组件来查询组件，而不是基于路径句法针对其自身的子组件来递归地查询子组件。

[0081] 在本发明的多个实施例中，定义了一组基元组件。任何基元组件的激活代表了由例如底层硬件平台、软件执行平台或由所述平台提供的基元信号检测的组件的执行环境所定义的基元操作的执行。例如，存储器槽以及输入和输出设备是djnn架构中的基元组件，依赖于计算平台上可用的存储器读操作和写操作。如果所述实施例专门用于创建形式，则本发明的另一实施例可以将文本标注和文本输入字段定义为基元组件。

[0082] 在本发明的多个实施例中，能够定义非基元组件。不是基元组件的任意组件的激活等价于命名为START的所述组件的子组件的激活，并且所述组件的去激活等价于命名为STOP的其子组件的激活。该规约施行组件的体系结构的递归本质，同时允许供应商选择如何在给定执行平台上接入组件体系结构。然而，用于定义非基元组件的其它手段会出现且由技术人员定义，而无需创造性步骤。

[0083] 在本发明的多个实施例中，组件的激活可取决于其激活上下文。激活上下文是对其它组件的引用的集合。所述其它组件称为所述组件的激活上下文的元素。组件的子组件

可以是所述组件的激活上下文的元素。例如，当被激活时产生声音的蜂鸣器能够在其子组件中的一个中找到所述声音的频率。类似地，一些组件可以在它们作为其子组件的任何组件中或者在所述组件的子组件中找到其激活上下文的元素。例如，蜂鸣器组件的替选实现可依赖于组件的子组件，所述组件还包含给出声音频率的子组件。可替代地，当耦合激活了其作用组件时，所述耦合使得其源和所述源的激活上下文可作为所述作用组件的激活上下文的元素来使用。例如，蜂鸣器组件的另一实现可以被编程为在触发其激活的源中找到声音频率。可替代地，组件的激活上下文的元素能够由执行平台来提供。例如，执行平台能够保持当蜂鸣器组件被激活时全部蜂鸣器组件所使用的声音频率，并且其它类型的组件可以修改所述频率。类似地，执行平台能够在图形对象被激活时保持图形对象所使用的绘图颜色，并且颜色组件能够修改所述绘图颜色。

[0084] 给定执行平台，能够定义基元组件以使得允许用于所述执行平台的任何可构思计算机程序实现为所述组件的集合。可用的组件可以根据本发明的各个实施例和实现方式而变化，并且可属于下面的非正式的、非限制性的以及非排他性的类别：

[0085] -容器组件，其允许将组件组织在子组件的集合中；

[0086] -控制结构组件，其控制其它组件的激活；

[0087] -存储器组件，其容许计算机或设备的存储器的解译、读和写；

[0088] -输入和输出组件，其定义了应用的输入和输出；

[0089] -操作组件，其执行由底层计算平台定义的操作，例如算术运算；

[0090] -编程者定义的组件，其通过将来自任何类别的组件组合而获得，包括其它编程者定义的组件。

[0091] 本说明书提供了来自所有类别的示例性的组件，数量足以使得技术人员能够利用这些组件或者从这些组件中得出新的组件从而产生交互式软件的全部当前类型，以及重命名它们或者修改它们的行为。技术人员还能够根据执行平台的当前的或未来的能力来定义新的类别。然而，在本说明书中提供的示例证实了根据本发明的交互式组件涵盖应用的最广泛可能范围的能力。

[0092] 所有所述的组件是通过将与由计算平台及其外围设备所提供的操作或机制的软件代理的耦合进行组合来定义的。所述组件本身能够通过利用耦合来组合。用于汇编和控制交互式组件的唯一机制的该通用使用提供了全递归软件体系结构的所有益处：同质性、复杂度管理、互操作性和互换性。

[0093] 空白组件是用来根据需要汇编任何子组件的简单的容器组件。缺省地，空白组件不与其子组件耦合，且其激活没有效果。在本发明的一些实施例中，用另一缺省激活来改变该缺省激活是可能的。该变化能够通过将其激活具有期望的效果的子组件添加到空白组件以及通过规定所述子组件必须取代所述空白组件的缺省START子组件来执行。

[0094] 图3显示出在本发明的实施例中通过称为绑定的控制结构组件创建的耦合的示例。

[0095] 其激活能够创建在两个其它组件之间的耦合的组件被命名为控制结构。

[0096] 绑定组件是简单的控制结构。绑定被定义有对称为其源的第一组件的引用以及对称为其作用的第二组件的引用。当被激活时，绑定创建在源或源的子组件与作用或作用的子组件之间的耦合。当被去激活时，所述绑定破坏耦合。图3在300中显示出绑定301，其具有

按键101作为其源如引用箭头302所指示，且具有蜂鸣器104作为其作用如引用箭头303所标示。绑定301在被激活时创建了按键101与蜂鸣器104之间的耦合103，产生了如图1中的100所示的配置。

[0097] 表示310是表示300的更紧缩的表示，其中已省略箭头302和303。并置组件的规约在本说明书的其它图中用来表示诸如源和作用或者输入和输出的引用。

[0098] 图4显示出在本发明的实施例中鼠标与蜂鸣器之间的绑定的示例。

[0099] 在本发明的多个实施例中，当在源组件和作用组件之间创建了绑定时，耦合的源可以是源组件的子组件，而不是源组件本身，所述子组件由所述源组件选定。类似地，绑定的作用可以是作用组件的子组件，而不是作用组件本身，所述子组件由所述作用组件选定。图4显示出其源是鼠标组件410而其作用是蜂鸣器430的绑定420。鼠标410由子组件按钮411、位置跟踪器412和组件413构成，使得413是与按钮411的耦合414中的作用，也是与位置跟踪器412的耦合415中的作用。鼠标410被定义为使得当其用作绑定的源时，所述绑定的激活创建了其源是组件413的耦合。

[0100] 结果，由位置跟踪器412检测到的鼠标410的任何移动或者按钮411的任何使用触发蜂鸣器410的激活。根据绑定的源和作用来定义耦合的源和作用的这一间接机制有益地支持期望查看系统的行为缩减成查看其部分中的一个部分的行为的情形。在申请人的经验中，尤其有用的是总是活跃的组件，并且为此然而在激活的辅助概念上推理也是有用的。例如，连接的输入设备总是活跃作为组件，因为其总是起到传感器的作用，但当用户在其操纵方面推理时作为激活的辅助概念。然而，不通过该间接机制定义绑定组件而同时仍在本发明的范围内是可能的。

[0101] 尽管其简单，绑定组件，与存储器、操作、输入和输出组件相结合，能够表达交互式应用的全范围的行为。在本说明书中的大量其它的控制结构组件(虽然更加复杂且容许容易地定义应用的复杂行为)是基于绑定与非控制结构组件的组合的。

[0102] 图5显示出在本发明的实施例中以性质作为其源的绑定。

[0103] 性质(property)是代表了存储器中的区域以及用于将信息存储在所述存储器中和取回存储器中的信息的编码格式的组件。例如，在本发明的多个实施例中，存在布尔性质，整数性质，浮点性质和文本性质。在本发明的多个实施例中，性质总是活跃的。在本发明的多个实施例中，性质具有仅直接用于分配组件的子组件读(READ)和写(WRITE)，并且子组件WRITE被定义当所述性质被用作绑定的源或者类似控制结构时所耦合的子组件。在这些实施例中，创建性质与作用之间的绑定确保了当性质被写入时作用被激活。因此，取决于所述存储器的值的任意值能够通过每次存储器被写入时的耦合来激活和更新。在图5所显示的示例性的实施例中，命名为“I”的性质510具有子组件读(READ)511以及子组件写(WRITE)512。箭头513表示子组件WRITE 512使用命名为其源的组件的引用以及其接受所述源作为其激活上下文的元素的事实。子组件WRITE 512的源是对从其中读取在性质510中写入到存储器的值的组件的引用。箭头514表示当性质510用作绑定的源时子组件WRITE 512被定义为所耦合的子组件的事实。蜂鸣器530每当被激活时就发声。绑定520具有性质510作为其源，蜂鸣器530作为其作用。当活跃时，绑定520创建其源是子组件WRITE且其作用是蜂鸣器组件530的耦合。结果，每当性质510被写入时，蜂鸣器530被激活时。

[0104] 空白组件能够用于汇编性质组件从而创建更复杂的性质组件。例如，几何点性质

能够通过汇编两个数值性质来创建，并且几何矩形性质能够通过汇编两个几何点来创建。组合的性质的值被定义为单独的性质的值的组合。

[0105] 图6显示出在本发明的多个实施例中称为分配的控制结构组件的三个示例。

[0106] 分配(assignment)组件是操作组件，定义有称为其源的第一组件的引用和称为其目的地的第二组件的引用。当分配被激活时，如果两个组件兼容，分配将第一组件的值复制到第二组件。配置600包括以名称为“j”的性质602作为其源而以名称为“i”的性质510作为其目的地的分配601。当分配601被激活时，将性质602的值复制到性质510。

[0107] 配置610证实了，在本发明的一些实施例中如何能够将分配601生产为创建其本身与性质510的WRITE子组件512之间的耦合611的组件，并且确保如箭头513所示WRITE子组件512的源引用与其自身的源引用相同。

[0108] 配置620显示出其中通过按下按钮来触发性质的复制的示例。绑定621具有按键101作为其源而分配601作为其作用。因此，在按键101按下时，分配601被激活，并且将源性质602的值复制到目的地性质510中。

[0109] 图7显示出在本发明的多个实施例中称为连接器的控制结构组件的示例。

[0110] 连接器是一种等价于绑定和分配的混合的控制结构。连接器定义有称为其输入的第一组件的引用和称为其输出的第二组件的引用。当其被激活时，连接器将其输入的修改与其输出耦合。配置700显示以性质602作为其输入且性质510作为其输出的连接器701。根据该配置700，每当性质602接收到新值，性质510接收性质602的值。连接器组件提供对数据流体系结构模式的支持，同时仍能与任何组件互操作且能与其它控制结构互换，例如，绑定组件，这是因为其是利用相同类型的构成部分来构造的。

[0111] 组件的配置710等价于配置700，其中连接器701已经替换成绑定711和分配712。箭头713代表了分配712的源引用，所述源引用指向性质602。

[0112] 配置720显示出在本发明的实施例中通过在空白组件721内汇编绑定子组件711和分配子组件712所创建的连接器。箭头713显示出，分配组件712的源引用被定义为与连接器组件的输入引用相同。类似于配置710的行为，被输入引用所指代的组件的任何修改激活分配712，分配712将所述组件的值复制到输出引用所指代的组件。

[0113] 在本发明的多个实施例中，算术运算和逻辑运算通过算术与逻辑组件来执行。算术组件与逻辑组件是能够利用与分配组件相同的原理定义的操作组件。代表了例如逻辑否的一元操作或符号改变操作的组件可具有对称为其操作数的数值或逻辑性质的引用以及对称为其结果的另一数值或逻辑性质的引用。代表例如逻辑OR或相加的二元操作的组件能够定义有称为其左操作数和右操作数的两个数值或逻辑性质的引用以及称为其结果的另一数值或逻辑性质的引用。当被激活时，对操作数的值执行运算，且将得到的值写入结果。算术与逻辑运算包括经典的算术运算、逻辑运算、数值比较和文本比较。相同的原理能够扩展到在本发明的实施例中技术人员能够通过参数和结果定义的任何数学运算或函数。

[0114] 其它算术和逻辑组件能够被定义成代表修改它们的操作数之一的运算。例如，增量组件被定义有对数值性质的引用，并且其激活将所述数值性质的值增加。操作数变化组件的列表能够容易地由传统编程语言中可用的那些推导出。

[0115] 控制结构组件还能够被定义成控制多个组件被激活的次序。下面的段落描述了一些更复杂的控制结构组件，它们证实了本发明的交互式组件如何能够用于定义复杂行为

的。

[0116] 例如，串行化组件能够定义有对第一组件和第二组件的引用，并且确保仅当第一组件返回不活跃状态时第二组件才被激活。串行化组件等价于第一组件的STOP子组件与第二组件之间的绑定。

[0117] 在本发明的多个实施例中，序列组件定义了多个组件逐一执行的次序。序列组件是一种能够被定义为具有其中规定了次序的多个子组件的容器组件的控制结构组件。例如，如果所述序列的激活触发了第一子组件的激活，并且通过对于所述序列的每个子组件创建所述子组件的STOP子组件与所述序列的下一子组件之间的耦合，则序列组件能够被创建。因此，当序列的每个组件在其执行结束时切换到不活跃状态时，其STOP子组件被激活，这激活了下一子组件，等等，直到序列末尾。

[0118] 在本发明的多个实施例中，循环组件是一种定义有称为其条件的布尔性质和称为其主体的组件的引用的控制结构组件。当循环组件被激活时，其重复地激活其主体且等待所述主体的去激活，只要所述循环组件的条件的值为真。

[0119] 在本发明的多个实施例中，并行组件是一种能够被定义为其中对子组件不规定次序的容器组件的控制结构组件，所述并行组件的激活触发全部其子组件的激活，并且不对子组件的所述激活的次序做出假设。

[0120] 在本发明的多个实施例中，同步组件能够定义有对称为其源的多个组件的引用、对称为其作用的组件的引用、以及对称为其初始原因的组件的引用。所述同步组件的激活被定义为使得由所述初始原因的相同的激活所触发的任何所述源的激活的任何非空数将触发一个且仅一个所述作用的激活。同步组件提供了用于控制复杂环境中软件的执行例如用于对数据流编程的有用的工具。

[0121] 在本发明的多个实施例中，能够扩展绑定组件以使得支持不位于相同程序中的组件的耦合，从而控制程序之间的分布式交互。

[0122] 先前通过控制结构组件定义的数学运算组件的组合能够用于定义以更适合于创建交互式软件的方式来代表数学运算的组件，利用了用于通过绑定和分配创建连接器组件的原理。

[0123] 在本发明的多个实施例中，通过将相加组件和称为左操作数、右操作数和结果的三个性质汇编，通过将两个操作数与所述相加组件耦合，以及通过使所述相加组件的左操作数、右操作数和结果引用分别指向所述左操作数性质、右操作数性质和结果性质，能够定义二元加法器组件。因此，左性质和右性质中的一个的任何修改激活所述左性质或右性质。左性质和右性质中的所述一个与相加之间的耦合激活了该相加，并且左性质和右性质求和为结果性质。

[0124] 在本发明的多个实施例中，计数器组件能够定义有数值性质和增量组件。能够为所有的数值、逻辑运算定义类似的组件。

[0125] 在本发明的一些实施例中，加法器组件中的操作数与相加之间的两个耦合还可以替换成耦合和存储器槽的更复杂配置。例如，相加的操作数可由等价于同步组件的组件来耦合，从而确保甚至当两个操作数由于相同的原因而改变时相加仅激活一次。

[0126] 计数器组件、二元加法器组件和类似的计算组件能够与连接器相结合使用来创建数据流，其中输入的任何变化产生新的计算。该数据流能够用于例如实现如下文说明的用

于图形对象的动画轨迹。

[0127] 能够定义输出组件，输出组件是与应用的物理或软件环境耦合或者其子组件与所述环境耦合的组件。所述输出组件或所述输出组件的子组件的激活触发了所述环境的变化，例如更新设备的显示器。

[0128] 图形对象组件能够被定义为表示图形显示器的2D内容的输出组件。图形对象组件通过计算机及其外围设备提供的一系列软件和硬件机制来与实际的物理显示器耦合。耦合的该集合称为图形对象组件的渲染引擎。

[0129] 图形形状能够被定义为当它们处于它们的活跃状态时在图形显示器上产生形状的图形组件。图形形状被定义为包含控制它们的效果的性质。例如，图形矩形包含了一个X、一个Y、一个WIDTH以及一个HEIGHT数值性质。渲染引擎确保了所述图形矩形的激活状态的变化或者所述性质的值的变化触发物理图形显示器的内容的变化。

[0130] 图形组是可以包含其它图形组件的图形组件。图形形状自然地用树结构来表达。例如，群组组件可使得表示按钮及其标记所需的全部的矩形和文本形状组件作为子组件。在本发明的多个实施例中，图形组或者任何其它施行次序的容器组件中的图形形状的次序确定了它们在图形显示器上的叠置次序。

[0131] 在本发明的多个实施例中，图形式样是控制例如图形形状的颜色、宽度或纹理的图形组件。图形式样被定义为包含控制它们效果的性质。例如，颜色组件包含了一个R、一个G和一个B数值性质，并且渲染引擎确保了所述颜色的激活状态的任何变化或者所述性质的值的任何变化触发物理显示器的内容的变化。

[0132] 在本发明的多个实施例中，受图形式样或图形式样的变化影响的组件是根据所述图形式样相对于图形组或施行次序的容器组件中的其它图形组件的位置来确定的。例如，能够定义，图形式样影响按照在所述图形式样处开始且在相同类型的任何图形式样处结束的次序间隔的所有的图形形状。

[0133] 在本发明的多个实施例中，图形变换是控制施加到图形形状的几何变换的图形组件。例如，尺度变换组件改变了受它影响的形状组件的尺寸。类似于图形式样，图形变换具有控制它们效果的性质，所述图形变换的激活状态的变化或者所述性质的值的变化触发物理显示器的内容的变化，并且受图形变换影响的形状列表由施行次序的容器组件中的图形对象的次序来定义。

[0134] 在本发明的一些实施例中，其它组件能够被定义为与物理显示器交互。能够根据文献中已知的任何绘图基元集合来定义所述其它组件。例如，图形对象组件能够被定义为形状、式样和变换的组合。可替代地，能够定义3D图形对象。可替代地，能够定义绘图操作组件，使得所述绘图操作组件中的一个的激活对图形显示器执行绘图操作，并且维持显示器上的图形仅能够通过所述绘图操作组件的反复启动来达到。

[0135] 能够为任何输出交互模态定义其它输出组件，例如用于发声的组件，用于控制灯的组件，以及用于控制任何电子装置的组件。而且，能够为计算机借以在其环境中操作的任何机制或者软件应用能够借以在其软件环境中操作的其它任何机制定义输出组件，例如停止运行于同一计算机上的其它应用的能力。

[0136] 输入组件是这样的组件：应用的物理或软件环境与该组件耦合，或者所述环境与所述组件的子组件耦合，使得所述环境的变化触发所述输入组件的激活或者所述输入组件

的子组件的激活。

[0137] 可以为任何输入设备定义输入组件。例如，键盘组件能够被定义为按键子组件的集合，每个按键代表了物理键盘的物理按键。对物理按键的物理作用与按键的子组件的激活耦合。类似地，鼠标输入设备能够表示为由用于每个按钮的子组件构成的鼠标组件，以及在物理鼠标的位置的每次变化时被激活的位置跟踪器子组件。在给定的计算机上可用何种输入组件取决于所述计算机的物理配置。

[0138] 输入组件还能够被定义成表示由连接到计算机的传感器检测到的对象或现象。例如，时钟组件是使用计算机的函数来检测时间的经过且在所确定的持续期间的每个时间间隔结束时被激活的组件。而且，输入组件能够被定义成表示测量计算机电池的能级的传感器、测量环境光的传感器、以及检测房间中存在人的传感器。

[0139] 通过传感器对新对象的检测以及对象的检测的丢失能够分别表示为代表所述对象的组件与集合组件的相加以及表示为从所述集合组件去除所述组件，集合组件将下下文描述。例如，如果传感器测量到多个手指在触摸敏感表面上移动，则手指的每次接触能够表示为代表所述触摸敏感表面的集合的子组件，所述接触的位置表示为所述子组件的位置性质。

[0140] 在各种交互式应用中，来自用户或环境的输入是根据应用的内容和状态来解译的，从而产生上下文丰富的输入。例如，当诸如鼠标、指示笔或触摸屏的指针设备与图形显示器相结合使用时，应用算法以计算哪个图形对象在所述指针设备的光标之下。所得到的对(位置、图形对象)能够视为上下文丰富的输入。这能够以多种方式来表示。例如，在本发明的一些实施例中，专用的输入组件被定义以表示所述上下文丰富的输入。例如，输入组件能够被定义有表示指针设备和图形显示器的组合的位置性质和引用性质。在本发明的一些实施例中，这由新的子组件与现有组件的相加来表示。例如，图形形状能够接收名称为“按下”、“释放”和“移动”的三个新的子组件，当指针设备的“按下”、“释放”和“移动”子组件分别被激活同时指针设备的位置处于图形显示器上该形状所占据的空间之内时所述三个新的子组件被激活。类似地，“进入”和“离开”子组件能够添加到图形形状中以表示指针位置何时进入或离开该形状所占据的空间的情况。

[0141] 输入/输出组件是被定义为既是输入组件又是输出组件的组件，例如对于接受输入和输出两者的外围设备。例如，带力反馈的操纵杆是输入组件和输出组件。表示计算机借以与其它计算机通信的通信设备的组件也能够定义输入/输出组件。输入/输出组件能够容易地在同一组件中利用创建输入组件的技术和创建输出组件的技术来创建。

[0142] 上述示例证实了根据本发明的交互式组件定义给定平台上可用的输入和输出的能力。所提供的概念能够容易地扩展到新的硬件、输入和输出外围设备或平台。

[0143] 通过创建输入组件与输出组件之间的耦合，能够获得交互式软件。所述耦合可通过将组件汇编来得到。所述组件的本质确定了交互式软件的行为。例如，绑定能够被创建为输入组件作为它们的源而输出组件作为它们的作用。通过将绑定、性质和操作汇编，然后将它们与输入组件和输出组件耦合，获得更复杂的行为。可替代地，能够创建诸如计数器和加法器之类的连接器和运算器的链从而产生输入组件与输出组件之间的数据流。例如，创建并激活其输入是指针的位置而其输出是矩形的位置的连接器确保矩形的位置将由鼠标的位置来驱动。作为另一示例，创建时钟与计数器之间的绑定，然后与所述计数器连接且计算

点在轨迹上的位置的连接器和数值运算器的集合，然后将得到的输出与图形对象的位置连接，得到在轨迹上成动画形式的图形对象。

[0144] 能够创建另外的控制结构来利于交互式软件的开发中的另外的模式。例如，列表组件能够定义被为其激活触发其子组件按它们规定次序激活且其去激活触发其子组件按相反次序的去激活的容器组件。与序列组件相比，列表组件不引入它们的子组件的任何串行化：列表组件的若干子组件能够同时活跃。

[0145] 因为列表组件是汇编交互式组件的非常自然的方式，所以定制组件类型由本发明的一些实施例定义为能够用于创建新组件类型的列表组件类型的变体。

[0146] 编程者能够使用定制组件来创建新的组件类型。为了创建新的组件类型，编程者能够创建定制组件，将子组件添加到其中，并且规定名称，在该名称下得到的组件必须被称为用于所述新组件类型的模板。另外，编程者能够通过规定新的子组件来取代所述定制组件的START子组件来重新定义定制组件的缺省激活行为。在该说明书中给出的XML示例中，标签“<component>”是指定制组件。该能力为编程者提供了开发和再使用为他们的需要而定制的组件的极大的灵活性。定制组件能够用于例如创建完整的应用，或者构成应用的部分的相当大的交互式组件、或者用于专用目的的新的可复用控制结构。

[0147] 集合组件是其激活触发了其子组件不按规定次序激活的容器组件。在本发明的一些实施例中，激活次序被定义成遵循伪随机法则。将子组件加入集合或者从集合中去除产生了分别是包含了对新添加的子组件的引用的名称为“ADDED”的引用性质以及包含了对新去除的子组件的引用的名称为“REMOVED”的引用性质的激活。

[0148] 本发明的一些实施例定义了一些组件类型中的“COPY”子组件。COPY子组件具有名称为“new”的引用性质。组件的“COPY”子组件的激活触发所述组件的副本的创建以及所述“new”性质到新创建的组件的引用的变更。

[0149] 能够为每个类型的容器组件定义迭代器组件类型。迭代器组件被定义有对称为其源的容器组件的引用，并且具有称为其触发器的空白子组件以及称为其输出的引用性质。当迭代器活跃时，每当其触发器被激活，其输出性质接收到对所述迭代器的源的子组件的新的引用(如果可能的话)。如果由于引用均已经被使用而没有新的引用可用，则迭代器在触发器被激活时被去激活。迭代器组件的变体能够被定义以使得从不自发地去激活，反而当触发器已经被激活并且没有新的引用被写入输出时，则子组件到所述迭代器的源的任何添加触发在所述迭代器的输出中写入对所述子组件的引用。

[0150] 状态机(或FSM)组件是管理状态变化的控制结构。FSM包含了称为状态的子组件、称为迁移的子组件、以及包含了状态的名称的命名为“状态”的性质。状态是定制组件。FSM的“状态”性质的值定义了所述FSM的哪个状态是活跃的。当所述FSM被激活时，对应于“状态”性质的初始值的状态被激活。迁移是定义有称为其源的组件的引用以及称为其目标的状态的名称的状态的子组件。迁移行为等价于绑定和分配的组合：当其源被激活时，其将“状态”性质的值修改成其目标的名称；这称为迁移的击发。当迁移被击发时，在本发明的一些实施例中迁移被定义为使用与在绑定中使用时的性质相同的间接机制，以使得当迁移被用作绑定的源时，是它的击发与所述绑定的作用耦合。因此，能够以各种方式使用状态机。子组件能够被添加到它们的状态中，以使得所述子组件仅在一种状态下活跃。可替代地，它们的状态能够用作绑定中的源。可替代地，迁移能够用作绑定中的源，以使得当所述迁移被

击发时作用被触发。可替代地，FSM的“状态”性质能够用作连接器的输入。

[0151] 图8和图9显示出FSM及其使用的示例。Petri Net和Statecharts组件能够被定义为类似于FSM组件的另外的控制结构，具有用于定义状态和迁移的不同的语义。

[0152] 交换机(switch)是具有状态而没有迁移的状态机的变体。交换机的状态也命名为其分支。如同FSM，交换机的状态性质控制其分支中的哪个是活跃的。交换机能够用于例如对操作结果施加测试，或者与FSM相结合。

[0153] 由于控制结构依赖于激活与和耦合的相同的基本机制，所以编程者能够轻易地将它们互换。例如，指针的位置与蜂鸣器之间的绑定确保了指针的任何移动将触发蜂鸣器。用指针的位置的X坐标与蜂鸣器的频率之间的连接器取代绑定反而确保了所述指针的移动控制所述蜂鸣器的频率。多个控制结构还能够同时用在相同的组件上。例如，配置能够被限定有指针的位置与蜂鸣器之间的绑定，以及所述指针的X坐标与所述蜂鸣器的频率之间的连接器。在该配置中，指针的移动将触发声音且同时控制该声音的频率。因此，根据本发明的交互式组件容许组件的最大的互操作性并且使得交互式软件的功能的修改极易执行。

[0154] 由于控制结构依赖于激活和耦合的相同的基本机制，还可能的是将程序缩减成耦合和基元组件的集合，从而对所述程序的行为进行分析。相互地，可能的是将它们汇编从而产生具有更复杂行为的控制结构以及允许编程者忽略所述控制结构的内部细节。

[0155] 而且，技术人员能够通过将现有的控制结构、操作和性质汇编在定制组件中来创建新的可复用控制结构。作为纯理论性的示例，图8显示出作为连接器组件的变体且对于其输入中的两个值变化的任何序列仅传播一个值变化的组件801。在800中，给出了XML表示，其中801看起来由性质802、两个连接器803和807以及状态机804构成，状态机804具有两个状态805和806以及两个迁移808和809。连接器807是状态806的子组件，并且因此它仅当状态机804处于状态806时才活跃。在820中是状态机804的另一表示，其两个状态805和806表示为圆圈821和822，其两个迁移808和809分别表示为箭头823和824。连接器807被显示为状态822-806的子组件825。而且，组件801的XML表示包含两个“<alias>”标签810和811以表示组件801分别具有与连接器803的输入引用等同的称为其输入的引用以及与连接器807的输出引用等同的称为其输出的引用。当连接器803将值变化传播到性质802时，状态机804的迁移808和809中的一个被击发，触发状态机804中的状态变化，并因此触发连接器807的激活或去激活。性质802的值变化由连接器807传播到其输出，并且因此传播到组件801的输出，仅当连接器807活跃时，其用于两个变化中的一个变化。得到的新的控制结构能够用于根据需要替代任何连接器。例如，其中性质831通过连接器832连接到性质833的830中所示的配置能够被修改以产生其中组件801取代连接器832的配置840。

[0156] 而且，技术人员能够创建用于产生计算或者用于对输入应用模式识别的可复用组件。例如，通过与指针的位置连接、包含控制结构和计算、以及还包含表示姿势的不同类的空白子组件的组件来执行姿势分类。所述组件中的控制结构和计算的作用是判定在指针位置的何种序列后必须激活哪个所述类的姿势。当该组件连接到指针位置时，该姿势分类组件能够用作新的输入组件。

[0157] 图9显示出构建为定制组件的按钮组件的XML表示。

[0158] 技术人员还能够创建实现交互序列的可复用复杂组件。例如，图9显示出构建为包含了构成其帧(frame)的矩形902、具有三个状态904、905和906以及五个迁移907至911的

FSM 903、具有三个分支913至915的交换机912、FSM 903与交换机912的状态之间的连接器916、作为按钮的标记(label)的文本组件917、名称为“pressed”的空白组件918以及绑定919的定制组件的按钮组件901的XML表示900。FSM 903的三种状态904至906命名为“空闲(idle)”、“按下(pressed)”和“出(out)”。交换机912的三个分支913至915具有相同的名称“idle”、“pressed”和“out”。所述分支包含了三个名称为“白(white)”、“黑(black)”和“灰(grey)”的矩形,它们是按钮901的三种可能的背景。在910中显示出按钮901的三种可能的方面。FSM 903的迁移和状态显示在FSM 903的图形表示920中。迁移的源是矩形902的输入子组件,例如“frame/press”,其对应于所述矩形上的指针设备的操作。所述操作触发了所述子组件的激活,这触发了对应迁移的击发。当迁移改变了FSM 903的当前状态时,所述状态的值被传播到交换机912的状态。交换机912的状态变化将当前背景矩形去激活且激活了新的背景矩形。因此,按钮背景的颜色根据指针的操作而发生视觉上的变化。另外,绑定919确保了每当迁移908被击发时“pressed”空白子组件被激活。因此,当通过指针对所述按钮执行了正确的交互序列时,与按钮901的“pressed”子组件耦合的任意组件被激活。然后,通过应用相同的方法来创建对话框,能够复制和复用得到的按钮901。申请人的经验是,相同的方法能够用于在给定任何任意组件行为规范的情况下产生与所述规范相符的可复用组件或应用。

[0159] 在本发明的一些实施例中,扩展机制允许编程者利用传统的编程语言来创建新的基元组件类型。例如,回调组件是参考传统编程语言的函数或过程定义的组件,并且其激活调用所述函数或过程。利用回调组件有助于创建用于新的输出模态的组件类型。通过将性质汇编且将它们与触发计算机或计算机外围设备的适当的函数的回调子组件耦合,能够创建新的输出组件。类似地,激活函数或过程是能够从传统编程语言中使用的触发组件的激活的函数或过程。使用激活函数允许编程者创建用于新的输入模态的组件类型。通过汇编子组件并且通过当检测到环境变化时被调用的激活函数控制所述子组件的激活来创建新的输入组件。

[0160] 还能够使用回调组件和激活函数以便于复用现有的软件而不终止以受益于本发明的优点。

[0161] 在本发明的一些实施例中,组件类型能够汇编在模块中,以使得当模块添加到计算机中时,所述模型中包含的组件类型能够用于软件应用中。通过本发明的实施例的供应商或者通过希望与其它用户共享他们工作的所述示例的用户,能够创建模块。

[0162] 上述示例证实了交互式组件定义用于定义交互式应用所必要的全部元件的能力。更具体地,它们能够定义操作、输入、输出和交互,同时保持能互操作且能互换,并且容许开发者和供应商开发用于当前和未来需要的新组件。

[0163] 本发明的示例实施例

[0164] 图10至16显示了在本发明的多个实施例中用于执行交互式应用和组件的示例性的体系结构。

[0165] 图10显示出在本发明的多个实施例中用于执行交互式应用和组件的示例性的体系结构,其中组件通过作为程序运行于计算机上的解译器来执行。

[0166] 体系结构1000包括计算机1010,计算机1010包括处理器1011、易失性存储器1012和非易失性存储器1013。

[0167] 处理器1011可以是可配置为执行可执行代码的处理器。其可以为例如中央处理单元(CPU),其可以从冯诺依曼(Von Neumann)体系结构、哈佛(Harvard)体系结构或修改的哈佛(Modified Harvard)体系结构中得到。

[0168] 易失性存储器1012主要用于装载待执行的计算机程序以及所述计算机程序的执行的上下文。其可以为例如随机存取存储器(RAM)、动态随机存取存储器(DRAM)、晶闸管随机存取存储器(T-RAM)或零电容器(Z-RAM®)存储器。

[0169] 非易失性存储器1012主要用于永久地存储待由计算机运行的应用。其可以为例如只读存储器(ROM)、磁带、硬盘驱动器、光盘驱动器、非易失性随机存取存储器(NVRAM)、闪速存储器、磁带、硬盘驱动器、如压缩盘(CD)、数字多功能盘(DVD)或Blu-Ray®盘的光盘、可编程金属化单元(PMC)、硅氧化物氮氧化物硅(SONOS)存储器、电阻随机存取存储器(RRAM)、域壁存储器(DWM)或浮动结栅随机存取存储器(FJG RAM)。

[0170] 在本发明的多个实施例中,交互式组件组织在诸如树100的树中并且由作为程序1020运行于计算机1010上的解译器来执行。在该体系结构中,组件1030和解译器程序1020装载到易失性存储器1012中。处理器1011执行解译器程序1020,这转而引起处理器1011执行组件1030。在图10中以及后面的图中使用的“IC”首字母缩略词代表“交互式计算”且一般地用于设计与交互式计算有关的任何概念,例如交互式计算平台或交互式计算供应商。

[0171] 在本发明的多个实施例中,解译器程序是当装载到易失性存储器1012中时能够在处理器1011中执行的程序1022。程序1022可以例如通过对例如由开发团队1023用C语言编写的源代码1021进行编译来获得。组件1030从存储在非易失性存储器1013上的可执行形式的组件1031、存储在非易失性存储器1013上的XML形式的组件1032或它们的组合而装载到易失性存储器1012中。因此,交互式应用由可执行形式的一组组件1031、XML形式的组件1032或它们的组合来定义。可执行形式的组件1031例如通过对例如C、C++、Perl或Java的源代码形式的组件1033进行编译来获得。开发团队1034因此能够通过创作XML形式1032、源代码1033或它们的组合的组件来开发应用。该实施例是有益的。事实上,以XML形式创作组件容许非常快速开发依赖于预定义子组件和操作的组件。同时,以可执行源代码创作组件容许开发实行利用目标平台上的预定义组件不可用的操作的组件。在本发明的多个实施例中,解译器是网络浏览器。组件可以例如是网络浏览器的脚本或附加。

[0172] 在本发明的多个实施例中,解译器1020将组件1030管理作为存储器中的数据结构,并且管理对组件的引用作为到所述数据结构的指针。在这些实施例中的一些实施例中,解译器能够使用至少两种表示存储器中的组件的方法。

[0173] 第一方法应用于将要对其激活唯一可能的交互的组件。为了易于理解,这些组件将被称为“本地组件”。第一方法包括通过指向用于创建解译器的相同的编程语言的函数的指针来表示组件。解译器1020通过调用表示它的函数并且通过将激活上下文通过传递给函数的变元进行传递来激活本地组件。

[0174] 第二方法包括在解译器1020的存储器中维持包含执行组件的行为所需的信息的数据结构。为了易于理解,这些组件将被称为“正常组件”。该数据结构包括至少指向名称为“组件类”的数据结构的指针、指向名称为“名称解析过程”的函数的指针、以及指向名称为“订阅列表”的订阅者的链接列表的指针。

[0175] 更具体地,在本发明的多个实施例中,组件类包含了表,在该表中存储有指向实现

为函数且为与同一类关联的全部组件共用的预定义子组件的指针的集合。每个公知的共用预定义子组件被分配了确定何种单元包含了与具有所述公知的预定义子组件的组件关联的全部组件类的表中的对应函数的数字。例如，对于由能够通过其它组件激活的解译器所提供的全部的组件类型，组件类具有存储在其表的第一单元中的函数，对应于START子组件。存在于一些组件类中的公知的共用预定义子组件的其它示例是STOP、COPY、ADD和REMOVE。当组件的START、STOP、COPY、ADD和REMOVE子组件需要被激活时，解译器从所述组件的组件类的表中取回对应的函数指针并且调用它。

[0176] 解译器通过取回其START子组件且通过相同的激活上下文激活它来激活正常组件。在正常组件的开发者已经对其START本地子组件编程而使其产生对其父组件所期望的行为的假设下，正常组件的激活的该递归序列最终实现了产生期望行为的本地START子组件的激活。

[0177] 由解译器所提供的基本构成组件，以及尤其是列表组件，具有被编程以确保它们适当的子组件在所述构成组件被激活时被激活的START子组件。

[0178] 如果它们想要解译器显现出更多关于所述组件的内部结构的细节，解译器的开发者能够容易地选择以将组件的内部表示从本地改成正常。

[0179] 所述名称解析过程用于从它们的名称取回所述正常组件的子组件。在一些组件中，所述名称解析过程能够被实现为与所述组件关联的数据结构中所维护的关联表中的查找。在其它组件中，所述名称解析过程能够被实现为关系数据库中的查询。在其它组件中，能够使用其它解析方法。例如，使用别名机制能够使得一些子组件在其它名称下可访问，从而掩盖组件的某些内部复杂度。

[0180] 除了为所有组件共用的部分之外，与正常组件关联的数据结构可以包含解译器必须维持从而确保组件的正确行为的任何数据。

[0181] 绑定组件包含了对其源组件和作用组件的引用。当被激活时，它们的START子组件将指向作用组件的指针添加到源组件的订阅列表。其它控制结构使用相同的技术来创建耦合。

[0182] 解译器确保，当第一组件被激活时，在其订阅列表中列出的所有组件被激活，接收到包含所述第一组件的激活上下文的激活上下文。

[0183] 在本发明的一些实施例中，组件的名称解析过程能够被扩展有用于对存在于另一程序中的组件进行命名的机制，所述程序可能在另一计算机上执行。在这些实施例中，与存在于另一程序中的组件的耦合不直接地通过将START子组件添加到订阅列表的方式来实现。反而，存储在订阅列表中的指针是指向包含将激活路由到另一程序所需的信息的数据结构的指针，利用进程间通信机制作为用于传递所述信息的传输层。当包含作用组件的程序接收到所述信息时，其利用其自身的激活机制来激活作用组件。

[0184] 解译器提供多个性质组件。每个对应于传统编程语言的基本数据类型：布尔值，整数，浮点数，文本串。附加的性质组件存储对另一组件的引用。全部的性质组件在它们被创建时是活跃的，并且没有START子组件。当性质的SET子组件被激活时，解译器确保所述性质的订阅列表中列出的全部组件都被激活，因此允许性质直接地用作绑定和其它控制结构中的源。

[0185] 在解译器的替选实施例中，全部的正常组件具有两个名称解析过程，用于解析分

别用于源上下文和作用上下文中的名称。在多数组件中，两个过程是相同的，但是在一些组件中，一个过程或另一过程可以被编程以实现别名使用机制，以使得子组件在替选名称下呈现，包括充当组件本身。例如，在性质中，用于源上下文的名称解析过程返回指向所述性质的SET子组件的指针，确保了当所述性质用作源时，其SET子组件是实际的源。

[0186] 能够通过如URI标准中定义的通用资源标识符引用来引用解译器中的组件。当解译器解析URI引用时，URI的根代表了应用的根组件，并且通过利用当前指针的名称解析过程将URI中的段连续地解析为指向组件的指针。

[0187] 如图11所示，各种方法能够用于创建解译器中的组件。通过示例，组件能够从解译器已知的组件复制，通过程序之间的通信方式从其它解译器接收，从永久存储器中或者从远程网络位置的各种格式装载，利用解译器所定义的应用编程接口(API)从现有的组件类型创建，或者通过解译器自动创建以代表软件的环境的元素。

[0188] 解译器1020从非易失性存储器1013装载组件。组件能够以各种格式存储在非易失性存储器中。通过示例，组件可以存储在解译器1020利用被提供有计算机的操作系统的动态库装载器装载的二进制库1031中。组件还可以采用解译器能够利用嵌入在所述解译器中的解析程序来装载的诸如XML 1032和Json的平台独立格式来存储。

[0189] 解译器1020还具有利用诸如HTTP和FTP的协议从远程网络连接装载交互式组件的能力。任何能够用于将组件存储到永久存储器中以及取回永久存储器中的组件的格式能够用于从所述远程位置取回组件。

[0190] 解译器1020还具有使用操作系统的动态装载器装载包含以传统编程语言编写的编译过程的二进制库的能力。所述过程使用解译器定义的组件创建API，以使得在解译器执行所述过程时，所述过程中规定的组件被创建。

[0191] 解译器1020还具有查询操作系统以得到计算机上可用的硬件设备以及使用可用的传感器来检测执行上下文及其行为的元素的能力。所述硬件设备的示例包括用户输入设备、环境传感器、网络接口、显示表面和物理效应器。执行上下文的所述元素的示例包括其它运行于计算机上的程序、传感器检测到的物理对象或人、能源以及网络业务流。对于每个所述硬件设备以及执行上下文的每个所述元素，解译器具有创建其结构和行为代表了所述硬件设备或执行上下文的所述元素的结构和行为的组件的能力。

[0192] 图11显示出能够由解译器从其中装载组件的多个位置。

[0193] 解译器1111是组件的解译器，例如体系结构1000的解译器1020。在其执行期间，解译器装载到为其执行指派的存储器区域1110中。待执行的组件1112装载到存储器中。

[0194] 组件1112能够例如从非易失性存储器1130装载。作为存储在非易失性存储器1130上的应用的部分的组件1131情况如此。

[0195] 组件1112还可以是应用的执行环境1120的组件1121。例如由计算机的硬件或OS暴露的组件情况如此。例如，如果计算机接受来自鼠标的输入，则“鼠标”组件能够被其操作系统暴露、以及通过应用检测和使用。

[0196] 组件1112还能够通过使用交互式组件的程序之间的交换来取回。例如存储在解译器1141的存储器1140中且由所述解译器1141执行的组件1142能够复制到存储器1110中且由解译器1111来执行。该副本可以在运行于相同机器上的两个解译器之间执行，或通过远程传输在两个不同的机器上的解译器之间执行。

[0197] 假设读者熟悉本发明的概念，现在将简要描述用于本发明的可能的实施例的其它可能的体系结构。

[0198] 图12显示出在本发明的多个实施例中用于执行交互式应用和组件的示例性的体系结构。

[0199] 图12显示出本发明的实施例，其中应用程序是通过实现解译器和附加模块的链接库以及使用解译器API来创建组件的程序来创建的。在http://djnn.net/可得到的djnn编程架构是由申请人开发的，用于设计交互式应用。djnn架构遵从体系结构1200，且技术人员能够容易地使用它来创建体系结构1000。

[0200] 类似于体系结构1000，体系结构1200包括计算机1010，计算机1010包括处理器1011、易失性存储器1012和非易失性存储器1013；易失性存储器1012中的组件1030从编译形式的组件1031以及从XML形式的组件1032装载到永久存储器1013中。

[0201] 不是通过解译器1020来执行，组件与通过装载动态库1222所获得的架构1220链接。这些库能够例如通过编译源代码1221来获得。该实施例有益地允许更容易地添加或更新动态库。例如，当将新外围设备添加到计算机上时，可以下载包含了用于所述外围设备的组件的动态库1222且保存在存储器1013中。

[0202] 图13显示出在本发明的多个实施例中用于执行交互式应用和组件的示例性的体系结构，其中组件被编译成可执行形式。

[0203] 类似于体系结构1000，体系结构1300包括计算机1010，计算机1010包括处理器1011以及易失性存储器1012。同时，开发团队1034通过分别创作XML形式和源代码形式的组件1032，1033来开发交互式应用。

[0204] 在体系结构1300中，交互式组件1032，1033由编译器1320变换成程序1310，程序1310能够直接由计算机1010及其操作系统来执行，并且使得处理器1011执行组件。在该实施例中，用平台独立的格式如XML和Json或者用编程者用来创建和汇编组件的专用编程语言来描述组件。编译器解析输入格式以提取交互式组件，并且将它们直接转换成机器语言或者目标计算机或者转换成之后能够转换成机器语言的传统的编程语言。

[0205] 图14显示出在本发明的多个实施例中用于执行交互式应用和组件的示例性的体系结构，其中组件被编译成可执行形式，用于专门设计用于执行交互式组件的操作系统。

[0206] 计算机1010的操作系统1420被编程以执行交互式组件，以平台独立的格式或者特定二进制代码。组件1431以如上所述的平台独立格式1032存储在非易失性存储器中，或者通过特定编译器从特定源代码的组件1432编译成特定于所述操作系统的可执行格式，并且操作系统将它们从存储器中装载到组件1430中以便执行它们。在这些实施例中，计算机的所有的硬件设备被表示为能够与编程者创建的组件耦合的交互式组件。在本发明的多个实施例中，操作系统是通过编译源代码1421获得的可执行程序1422。

[0207] 图15显示出在本发明的多个实施例中用于执行交互式应用和组件的示例性的体系结构，其中组件被编译成能够直接由处理单元执行的应用。

[0208] 体系结构1500包括计算设备1510，该计算设备具有为执行交互式组件而创建的专用处理器1511。在该实施例中，组件1032，1531能够以诸如XML和Json的平台独立格式或者以编程者用来创建和汇编组件的专用编程语言来描述。编译器1540将这些格式转换成二进制格式的组件1530，当这些组件闪存为存储器1512中的组件1520时，处理器1511能够直接

执行这些组件。

[0209] 图16显示出在本发明的多个实施例中用于执行交互式应用和组件的示例性的体系结构，其中组件被编译成处理单元的电路设计。

[0210] 体系结构1600包括可编程用于直接执行代表了组件的二进制代码的处理单元1610。在该实施例中，交互式组件及其子组件由电路编译器1640转换成其行为是所述交互式组件的行为的处理单元1610的专用电路1620。例如，能够将可编程灯开关编程为交互式组件的集合，然后转换成足够小以便嵌入壁式开关中的专用电路1620。

[0211] 给出上述的示例被作为本发明的实施例的说明。这些示例绝不以任何方式限制由随附的权利要求所限定的本发明的范围。

[0212] 本发明的应用的示例

[0213] 本发明能够用于使用各种产生过程来产生各种类型的交互式系统。

[0214] 作为示例，使用在图12中显示的本发明的实施例，可以创建类似于传统的用户界面工具箱中提供的可复用图形交互式组件，包括但不限于按钮、对话框、复选框、菜单和列表。所述组件的状态能够由状态机组件来表示，由所述状态机的迁移或者具有计算机的指针输入设备或它们的子组件作为它们的源的图形交互式组件中的其它控制结构来表示。值之间的相依性可以由连接器来表示。

[0215] 利用可供用来存储和装载交互式组件的全部机制，所述组件能够在完全利用本发明编程的应用中复用。可替代地，所述组件能够在用传统语言编写的程序中复用，利用由图12中显示的库所提供的API来创建组件并且将组件与应用的其余部分连接。

[0216] 可替代地，上述的图形交互式组件能够按如下方式利用本发明来创建：它们的图形子组件和它们的行为子组件在子组件树中明显地分开并且能够独立地装载，最初通过图形交互式组件提供的版本用作缺省设置。这支持其中图形和行为或者图形交互式组件的任何其它有意义的部分利用诸如图形设计软件的专用软件工具来独立地产生的各种工程过程。这包括但不限于：用户的定制；在应用开发期间软件设计与图形设计之间的同时工程；从第三方源免费地或者在合同执行中获取图形或行为子组件；利用用于基于触摸的指针设备以及用于鼠标或等同指针设备的不同行为，在部署时适配行为或图形以适应执行平台的能力，例如，显示器的尺寸或指针设备的类型。

[0217] 还可能的是，使用上述过程示例的任意示例来创建专用于数据管理和计算的交互式组件，以便于利用上述工程过程中的一个将它们复用于交互式应用中。由于本发明包括了传统编程语言可用的一般数据管理、计算和控制结构基元，所以技术人员能够实现全部的计算和数据表示。尽管如此，所提议的用于这样做的方法不同于传统语言所使用的方法，因为在传统计算中函数的调用是唯一执行的原因，而在交互式软件中数据的任何变化能够触发结果的重估计。所提议的方法因此包含了：在建立这些值、中间结果和将可用的结果之间的相依性关系之前首先确定哪些值易于彼此独立地变化，然后创建表示这些值和结果的性质组件，然后创建计算操作器、连接器和表示期望的相依性和计算的其他控制结构组件。与得到的计算或数据组件交互将通常通过与已经被定义且表示计算的输入和输出的性质子组件进行交互而成为可能。

[0218] 还可能的是，使用上述过程示例中的任意示例来创建专用于动态数据显示的图形交互式组件，包括但不限于计量器、标度盘和直方图。所述组件能够连接到任意数据源，利

用图12的API以将所述源变成能够与任何其它可用的交互式组件相结合使用的交互式组件。例如，给定提供值流的网络服务，技术人员能够容易地使用传统编程语言如C或Java以及图12的API来创建表示所述值流且每当接收到值就被激活的组件。

[0219] 还可能的是，使用上述过程示例中的任意示例来创建不进入通常取名为WIMP(窗口图标菜单指针)的软件组件的范畴且有时分类为后WIMP的交互式图形组件或者其可复用子组件。发明人的经验是，全部的后WIMP交互能够通过本发明的交互式组件的适当结合来产生，利用如下方法：所述方法以确定全部相关的可察觉子组件以及与所创建的组件交互的全部相关外部组件而开始，然后确定所述子组件与所述外部组件之间的全部的耦合和因果关系，以及确定所创建的组件的全部重要状态，然后选择表示所述状态和耦合的适当的控制结构、以及组织子组件的适当的结构。

[0220] 还可能的是使用上述过程示例中的任意示例来创建多模态交互式组件或其可复用子组件。多模态交互式组件是与多种类型的人类输入组件进行交互的交互式组件，例如对姿势和语言的组合做出反应的组件，或者对鼠标和键盘的组合做出反应的组件，或者对眼睛注视检测和平板计算机倾斜的组合做出反应的组件。一些多模态交互式组件还能够对人类输入和来自执行上下文的输入的组合做出反应，例如CPU温度和驱动用户分配给任务的CPU电力量的指针移动的组合。另一示例是外部光传感器和驱动图形对象的轨迹的指针移动的组合，所述轨迹在不利的灯光条件下通过附加的动画移动来夸大。创建这些多模态交互式组件能够利用与上文针对后WIMP交互所描述的相同的方法来执行，具有附加的步骤创建表示尚不可用的输入设备或输入处理计算的交互式组件。例如，上述的d jnn架构提供了对于标准输入设备的扩展覆盖，但是如果定制输入设备利用诸如其GPIO或SPI接口的计算机能力来创建，则技术人员能够容易地使用C或C++编程语言以及图12的API来创建表示所述定制输入设备的新的交互式组件。

[0221] 还可能的是使用上述过程示例中的任意示例来创建自适配交互式组件或者其可复用子组件。自适配是软件对其执行环境的变化做出回应以及改变其自身结构及其自身行为的能力。例如，当灯光条件变化，软件能够将一个文本输入组件替代成另一文本输入组件。能够使用上述用于后WIMP交互以及多模态交互的相同的方法，其中所研究的耦合不仅在外部组件和可觉察子组件之间，而且在外部组件与控制结构之间，所述控制结构用于确定在给定的时间哪个子组件是活跃的。例如，开关组件能够连接到灯传感器以控制作为所述开关组件的子组件的两个文本输入组件中的哪个子组件是活跃的。

[0222] 还可能的是使用上述过程示例中的任意示例来创建用于实现连接对象或者与连接对象交互的交互式组件。与连接对象交互类似于与输入和输出设备交互。创建用于此的交互式组件能够利用相同的方法来完成，具有附加的步骤创建交互式组件以表示每个连接的对象。创建用于交互式对象的软件类似于在计算机上创建交互式应用，并且包括创建与输入和输出设备交互的交互式组件。连接对象与位于远程计算机上的应用的通信能够利用本发明所描述的分布式交互模式来实现，或者通过将事件转换成诸如REST的选定传统协议来实现。在图15和图16所显示的体系结构尤其适合于实现连接对象。

[0223] 还可能的是使用上述过程示例中的任意示例来创建其子组件同时位于多个计算平台上从而支持分布式交互的交互式组件。分布式交互是这样的交互式样：其中一个或多个用户同时与多个计算设备交互从而实现任务。例如，隧道是一种部分显示在一个计算机

显示器上而部分显示在另一计算机显示器上的组件，且当图形对象被推入隧道一侧时，其出现在隧道的另一侧，触发由所述图形对象表示的数据从一台计算机传递到另一台计算机。

[0224] 还可能的是依赖于图10和图12所显示的体系结构来创建用于图10中显示的解译器或者图12中显示的编程架构的附加模块。模块可以包括支持与输入或输出设备进行交互的交互式组件的集合，例如能够用于在声卡上创建声音序列的组件的集合，或者表示专业网络服务的组件的集合。可替代地，模块可以包括由预定义子组件构成且支持给定交互式样的交互式组件的集合。例如，模块可以包含可复用WIMP微件的集合、数据显示的集合、或者姿势识别组件的集合。

[0225] 还可能的是依赖于图10至16所显示的体系结构来创建新的编程语言。这包括用于交互式软件的通用编程语言以及专门用于某些类型的交互式软件的编程语言、交互式软件的一些使用、或者交互式软件的用户的某些类。这还包括纯文本语言以及图形记号。

[0226] 该专业化视觉或文本编程语言的示例是用于将操作系统的行为参数化的脚本语言。所有操作系统提供若干用于编写当计算机启动或者计算机中发生某些事件时启动的所谓“脚本程序”的脚本语言。这样的脚本程序用于例如启动服务，修改输入设备的配置，发送消息。脚本程序还能够由技术娴熟的用户编写以创建对于其来说通用编程语言将过于复杂的小的便携程序。当前的脚本语言的局限性是缺少对大多数语言中的交互的支持，以及在其它语言中受限的支持。编写交互式脚本例如将可用于需要与用户进行超越了文本输入和输出的交互的小程序，用于为有专门需要的人创建输入事件的定制变换。

[0227] 该专业化的视觉或文本编程语言的另一示例是终端用户用来对它们的交互式对象和应用的行为进行编程的语言。终端用户可以使用所述语言来创建事件与动作之间的简单的绑定，并且还基于他们将创建或复用的交互式组件来创建更复杂的程序。取决于应用，用所述语言编写的程序可存储在终端用户所拥有的设备中或者存储在能够通过网络连接访问的服务器中。

[0228] 对于上述应用情况的示例使用本发明的益处在于，除了表示给定硬件设备且用传统语言编程的组件之外，在适当的格式转换后，相同的组件能够从一种应用情况到另一种应用情况复用。这些组件可以由借助不同工具创建的子组件汇编，共享或出售，根据可用的验证标准来验证和认证，以及在上述任意示例中复用。这显示出本发明如何基于交互式组件的交换来支持经济性，类似于电子组件所存在的交互式组件的交换。

[0229] 基于本发明的协作过程的示例是用于设计和开发用于关键应用的交互式软件的一套工具。

[0230] 基于本发明的协作过程的另一示例是存在于用户选择访问的网络服务器上且当被激活时在激活它们之前将它们的子组件整体地或者部分地转移到用户的计算设备上的交互式应用的创建。

[0231] 基于本发明的协作过程的另一示例是通过电子邮件、即时消息传递系统或社交网络所传递的交互式组件的交换。尽管当前的软件通常支持图像、文档的交换，本发明将使得技术人员更易于创建其中能够交换交互式组件的应用。例如，终端用户可以创建并发送接收者必须通过姿势打开而显现其内容的模拟信封或者带有必须打开来访问其内容的4数字锁的盒子。接收终端用户还能够将接收组件添加到正在相同的计算平台上运行的应用中从

而扩展其能力。例如，一个用户能够发送与位于他选择显示的位置的远程照相机连接的可缩放观看端口，并且接收用户将接收可缩放观看端口作为活跃组件以使她能够查看位置，并且她能够将接收到的观看端口添加到她计算机上的照片观看应用中，以使得位置总是在显示器的角处可见。

[0232] 基于本发明的协作过程的另一示例是由终端用户通过将他们创建或取回且他们能够与朋友共享的视觉交互的可能是3D的组件汇编所创建的交互式空间的创建。比如社交网络中像用户创建定制3D空间的Second Life或用户创建定制的“墙壁”的Facebook，用户可以创建交互式3D或2D空间来显示虚拟翻板上他们的照片或谜语，或者在他们所创建的定制可视化中他们喜爱的数据，并且邀请其它用户来与他们私有空间的部分进行交互。

[0233] 图17显示出在本发明的多个实施例中由交互式组件构建的显示飞机的主飞行显示器的应用。

[0234] 主飞行显示器(PFD)1700是通过将本发明的实施例中的交互式组件进行汇编来渲染的。其能够用于例如飞行模拟的应用。

[0235] PFD 1700包括有水平线、天空和陆地的背景、航向显示器、空气速度指示器、高度指示器1701、姿态指示器1704、警报地形警告和拉起警告1703。

[0236] 在PFD执行期间，当嵌入有PFD 1700的飞机的高度或者针对其模拟PFD的飞机的高度在参数阈值以上时，警报地形警告和拉起警告1703必须显现为红色消息“Alter Terrain”和“Pull-up”。

[0237] 图18显示出在本发明的实施例中定义显示警告的组件的XML。

[0238] 组件“警告”包括两个警告1801和1802。高度阈值以及高度被定义，并且以任意值初始化。在应用执行期间，这些值由环境更新，例如通过将组件中高度的值设定成飞机的预测或测量高度。

[0239] 在修改其左右子组件中的之一时激活比较器，并且如果右子组件具有大于左子组件的值，则返回值“真”。

[0240] 连接器将高度的值连接到比较器的左子节点。因此，在每次修改高度时，激活比较器。

[0241] 连接器将阈值的值连接到比较器的右子节点。因此，在每次修改阈值时，比较器被激活，并且每当高度在阈值以上时返回正值。

[0242] 地形警告组件和拉起警告组件定义了警告的布局，主要是消息的颜色、字体、大小和位置，这对应于PFD 1700中的红色消息。

[0243] 警告组件1801和1802封装在开关内。因此，仅当处于开关为真(true)的状态时，这些警告组件才是可见的。开关初始化为假(false)值。

[0244] 连接器将比较器的结果连接到开关的状态。因此，在阈值或高度每次变化时，如果高度在阈值以下，则开关的状态被设定为真，而如果在阈值以上，则开关的状态设定为假，并且当且仅当高度在阈值以下时，警告可见。

[0245] 图19显示出在本发明的实施例中用于设定模拟飞机的目标高度的触觉显示器。

[0246] 触觉显示器1901附接到两个迁移1904和1905，用于按下和释放显示器，选定的高度通过连接器1902和1903复制作为飞机的目标高度。

[0247] 图20显示出用于模拟目的设定飞机高度的phidget的示例。

[0248] 滚动件2002通过从phidget 2002和高度2001的连接器来设置高度2001。如果高度超过阈值，则两个警告被激活。滚动件2002的位置的水平修改产生高度2003的垂直修改

[0249] 这些示例证实了根据本发明的交互式组件创建复杂应用和交互的能力。它们还证实了根据本发明的交互式组件根据来自用户或环境的输入而修改应用的行为的能力。

[0250] 给出上述的应用的示例作为本发明的应用的示例说明。这些示例绝不以任何方式限制由随附的权利要求书所限定的本发明的范围。

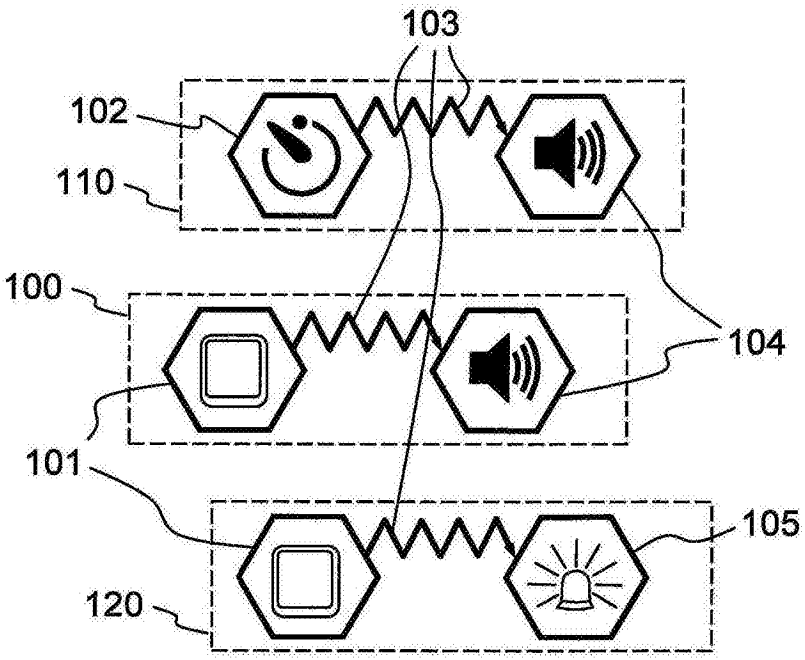

103
102
110
100
104
101
105
120

图1

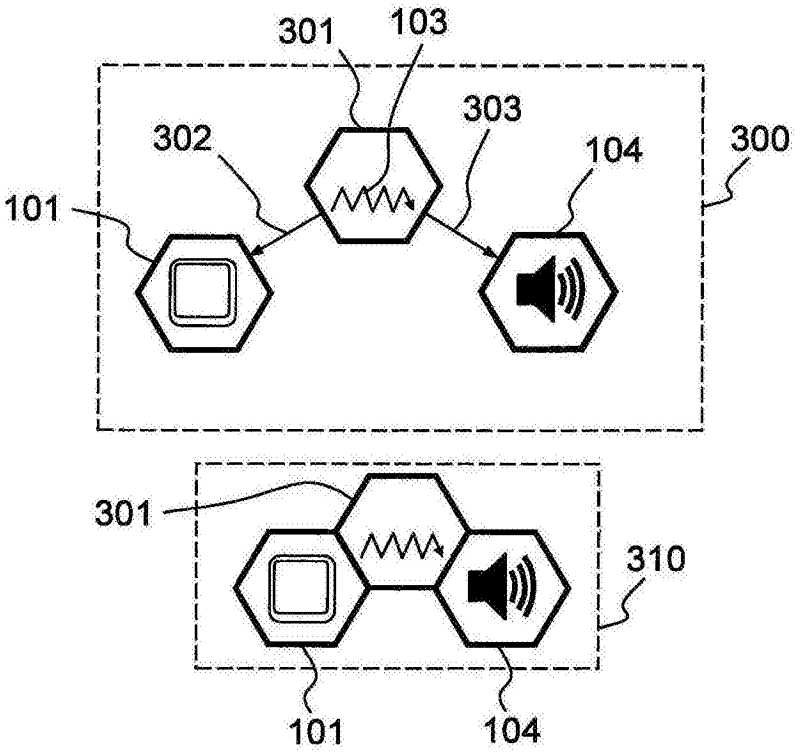

301
103
303
302
104
300
101
301
310
101
104

图3

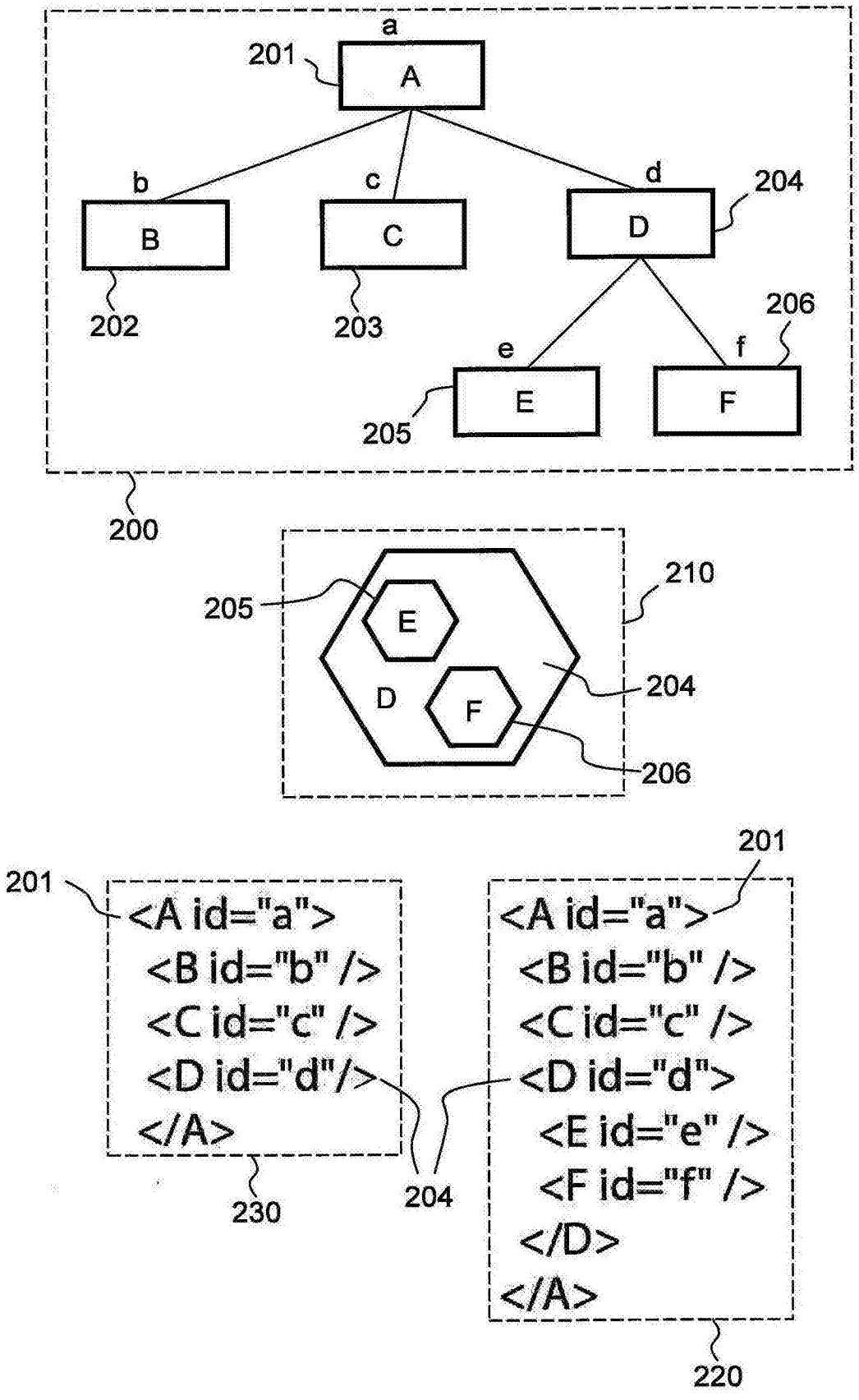

a
201
A
b
B
c
C
d
D
204
202
203
206
e
E
f
F
205
200
205
E
210
D
F
204
206
201
<A id="a">
<B id="b" />
<C id="c" />
<D id="d"/>
</A>
230
204
201
<A id="a">
<B id="b" />
<C id="c" />
<D id="d">
<E id="e" />
<F id="f" />
</D>
</A>
220

图2

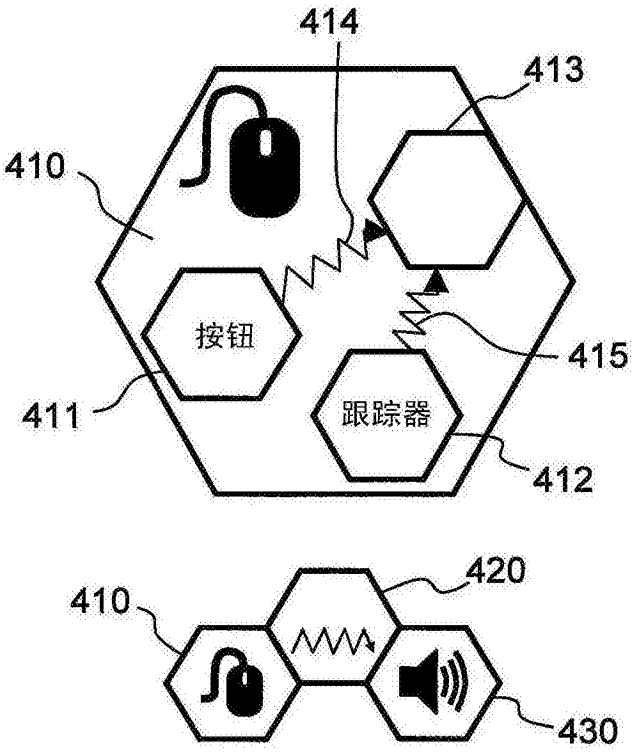

414
413
410
按钮
415
411
跟踪器
412
410
420
430

图4

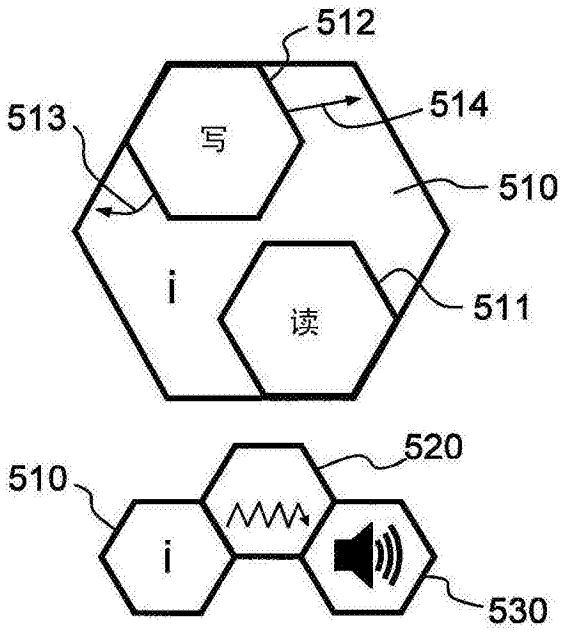

512
513
写
514
510
i
读
511
510
520
i
530

图5

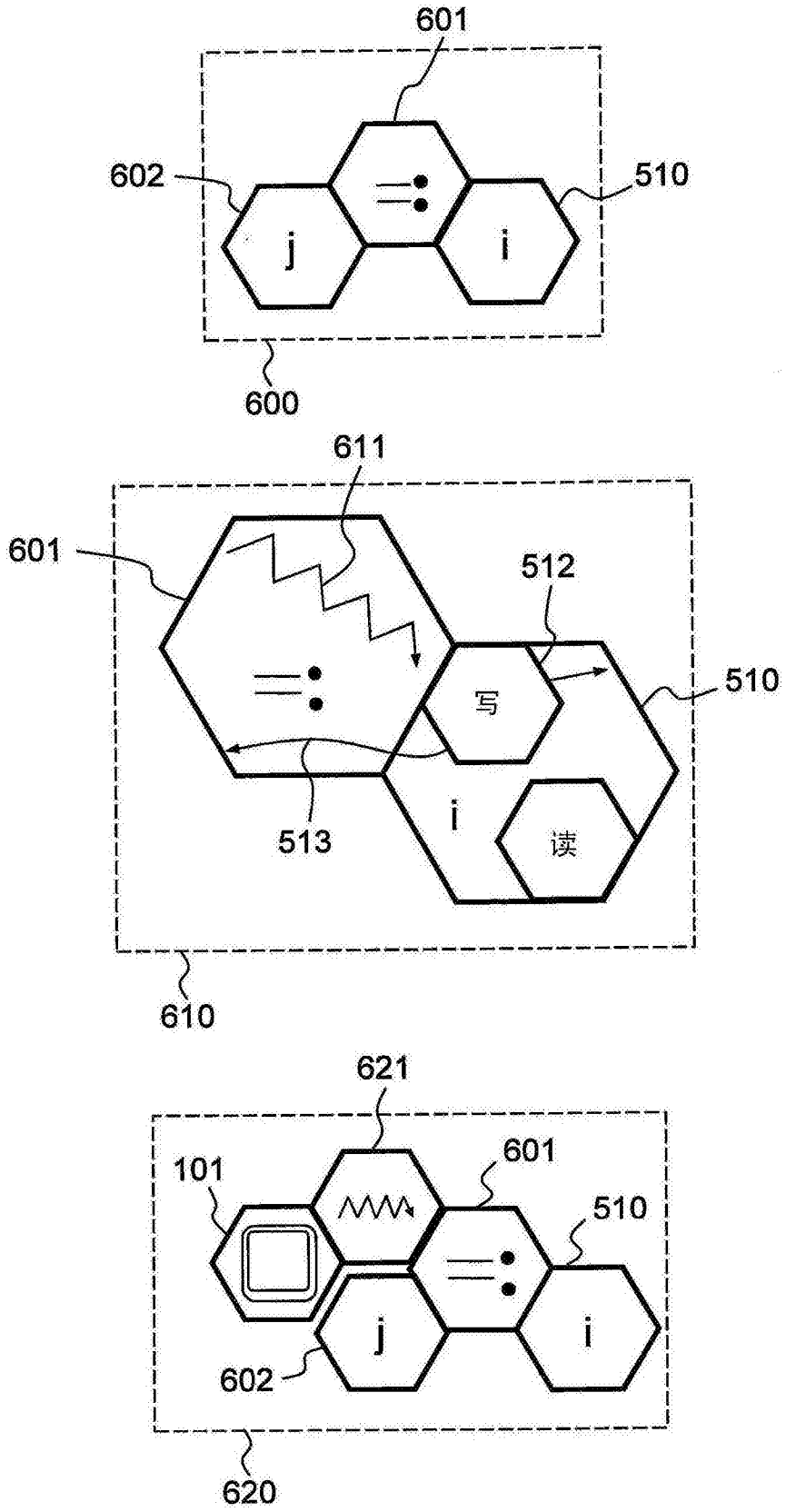

601
602
510
j
i
600
611
601
512
写
510
513
i
读
610
621
101
601
510
j
i
602
620

图6

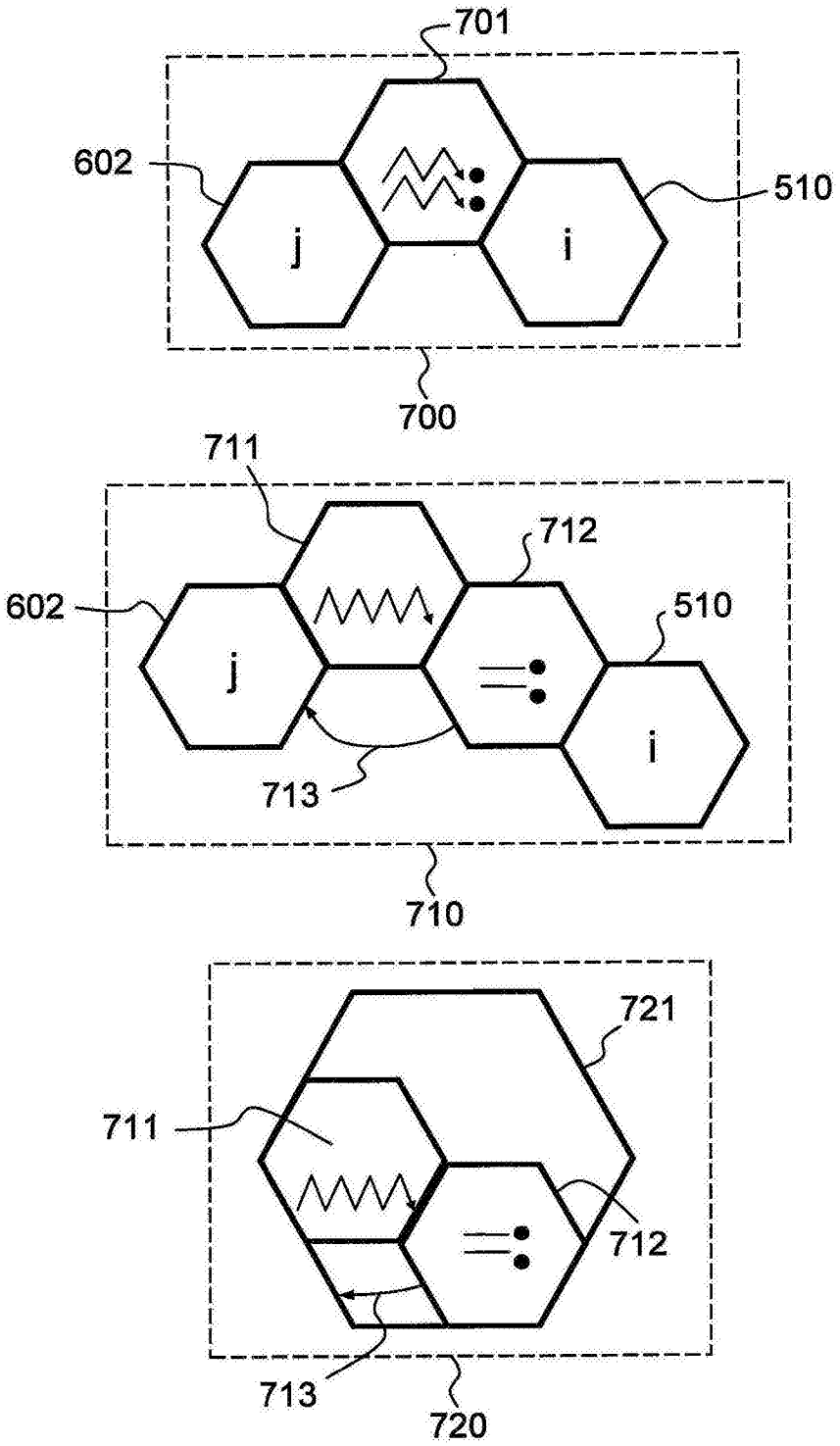
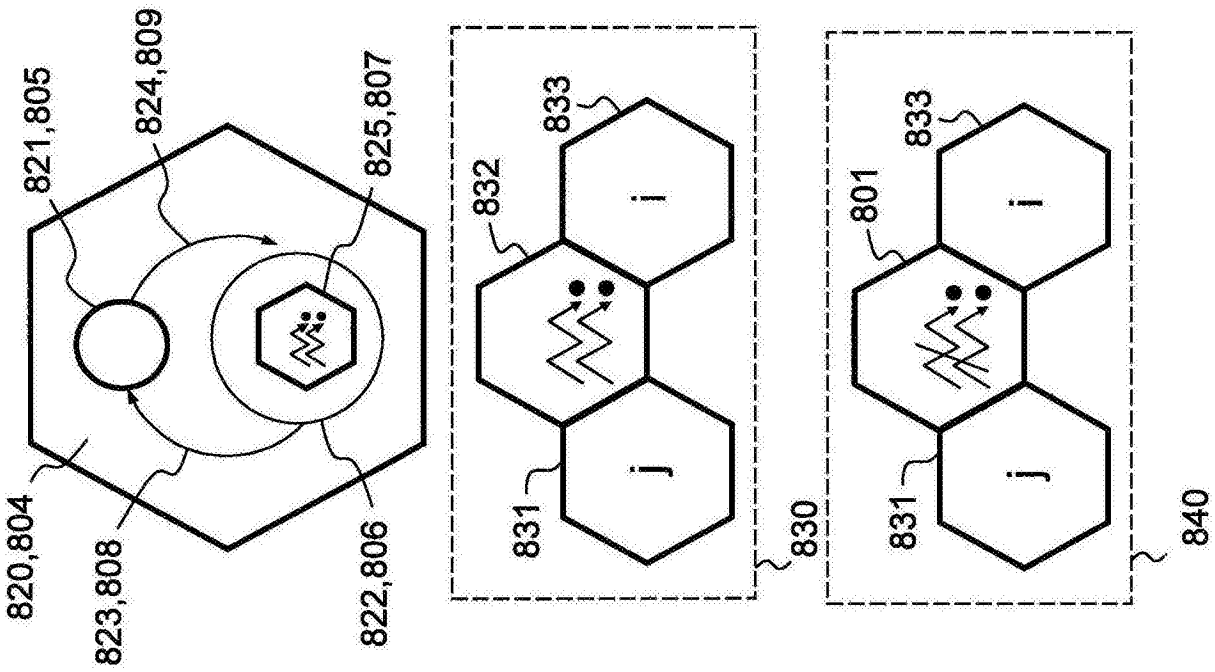


图7


821,805
824,809
825,807
820,804
823,808
822,806
832
833
831
830
801
840

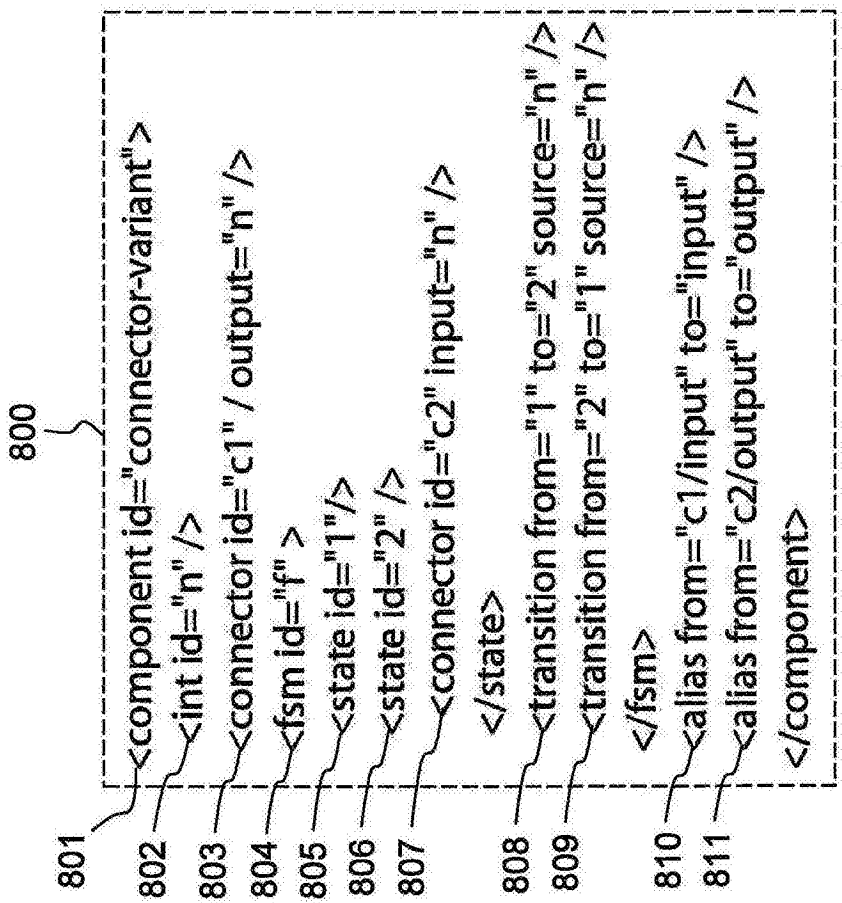

800
801 <component id="connector-variant">
802 <int id="n" />
803 <connector id="c1" / output="n" />
804 <fsm id="f" >
805 <state id="1"/>
806 <state id="2" />
807 <connector id="c2" input="n" />
</state>
808 <transition from="1" to="2" source="n" />
809 <transition from="2" to="1" source="n" />
</fsm>
810 <alias from="c1/input" to="input" />
811 <alias from="c2/output" to="output" />
</component>

图8

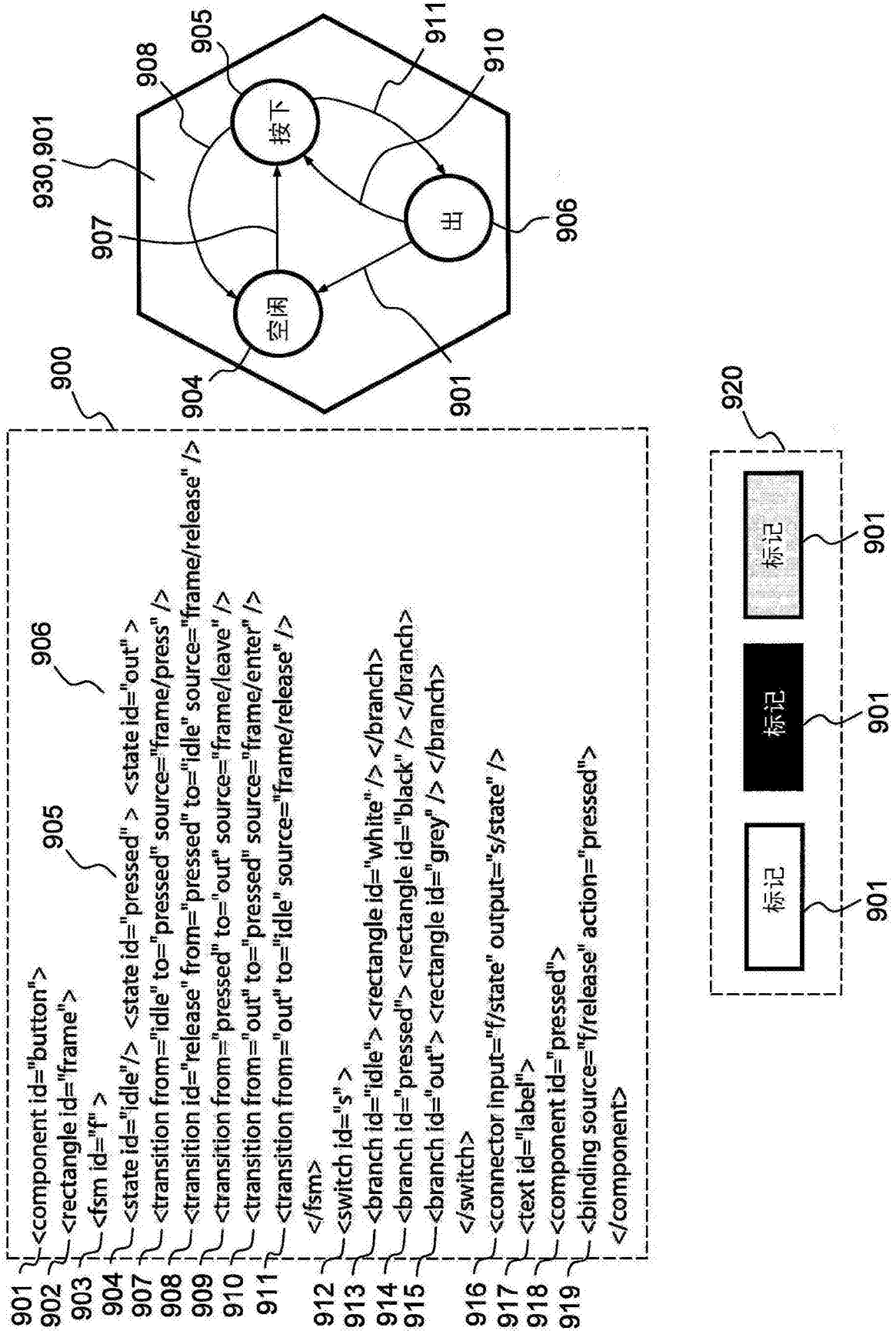

空闲
按下
出
标记
标记
标记
901
902
903
904
907
908
909
910
911
912
913
914
915
916
917
918
919
<component id="button">
<rectangle id="frame">
<fsm id="f" >
<state id="idle"/> <state id="pressed" > <state id="out" >
<transition from="idle" to="pressed" source="frame/press" />
<transition id="release" from="pressed" to="idle" source="frame/release" />
<transition from="pressed" to="out" source="frame/leave" />
<transition from="out" to="pressed" source="frame/enter" />
<transition from="out" to="idle" source="frame/release" />
</fsm>
<switch id="s" >
<branch id="idle"> <rectangle id="white" /> </branch>
<branch id="pressed"> <rectangle id="black" /> </branch>
<branch id="out"> <rectangle id="grey" /> </branch>
</switch>
<connector input="f/state" output="s/state" />
<text id="label">
<component id="pressed">
<binding source="f/release" action="pressed">
</component>

图9

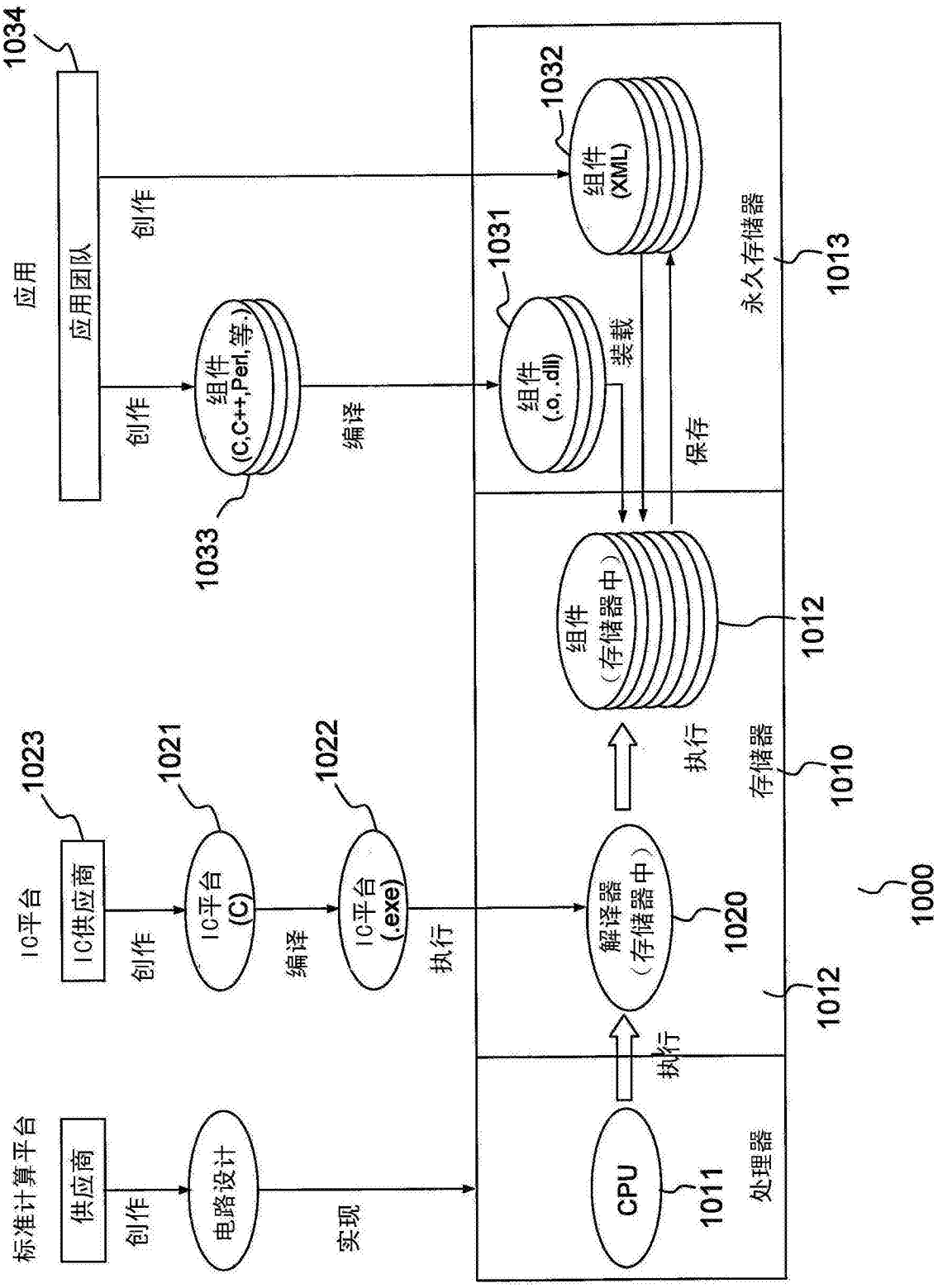

1034
应用
应用团队
创作
创作
组件
(C,C++,Perl,等.)
1033
创作
编译
1031
组件
(.o, .dll)
装载
保存
1032
组件
(XML)
永久存储器
1013
组件
（存储器中）
1012
执行
存储器
1010
1023
IC平台
IC供应商
创作
1021
IC平台
(C)
编译
1022
IC平台
(.exe)
执行
解译器
（存储器中）
1020
执行
1012
1000
标准计算平台
供应商
创作
电路设计
实现
CPU
1011
处理器

图10

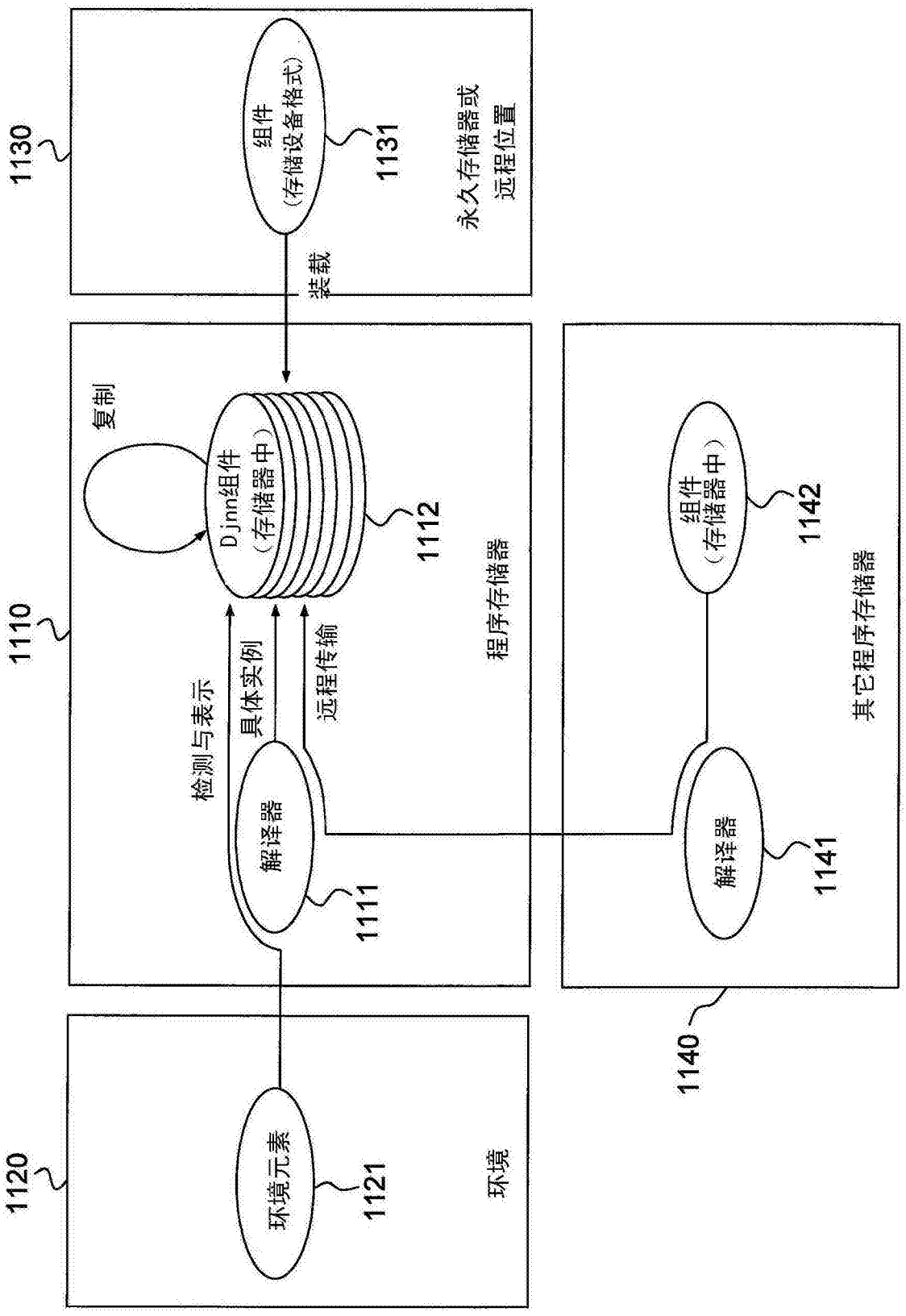

1130
组件
（存储设备格式）
1131
永久存储器或
远程位置
装载
复制
Djnn组件
（存储器中）
1112
1110
检测与表示
具体实例
远程传输
程序存储器
解译器
1111
组件
（存储器中）
1142
解译器
1141
其它程序存储器
1140
1120
环境元素
1121
环境

图11

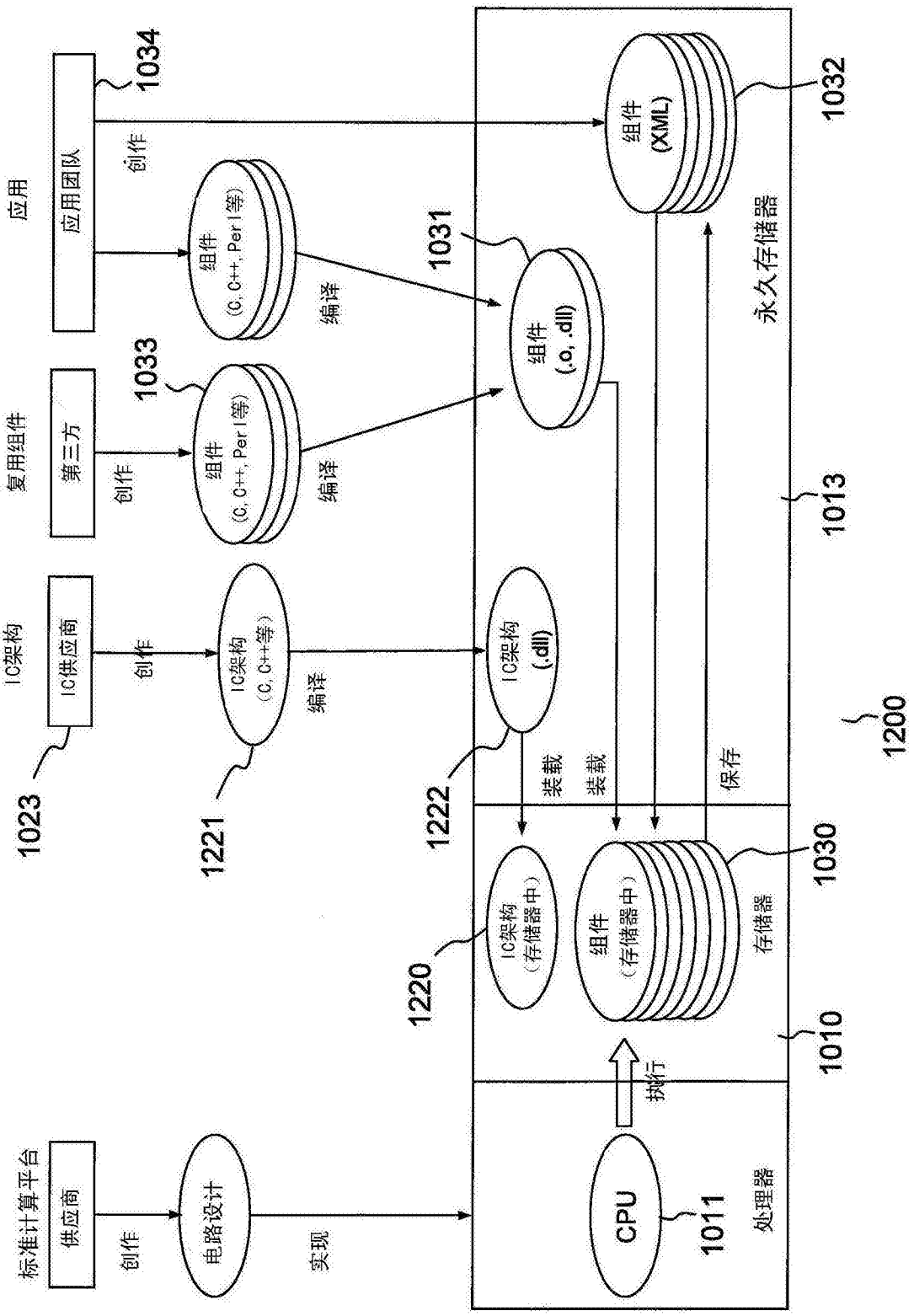

应用
应用团队
1034
创作
组件
(C, C++, Per l等)
编译
1031
组件
(XML)
1032
复用组件
第三方
1033
创作
组件
(C, C++, Per l等)
编译
组件
(.o, .dll)
永久存储器
1013
IC架构
IC供应商
创作
IC架构
(C, C++等)
编译
IC架构
(.dll)
1023
1221
1222
装载
装载
保存
1200
IC架构
(存储器中)
组件
(存储器中)
存储器
1030
1220
1010
执行
标准计算平台
供应商
创作
电路设计
实现
CPU
1011
处理器

图12

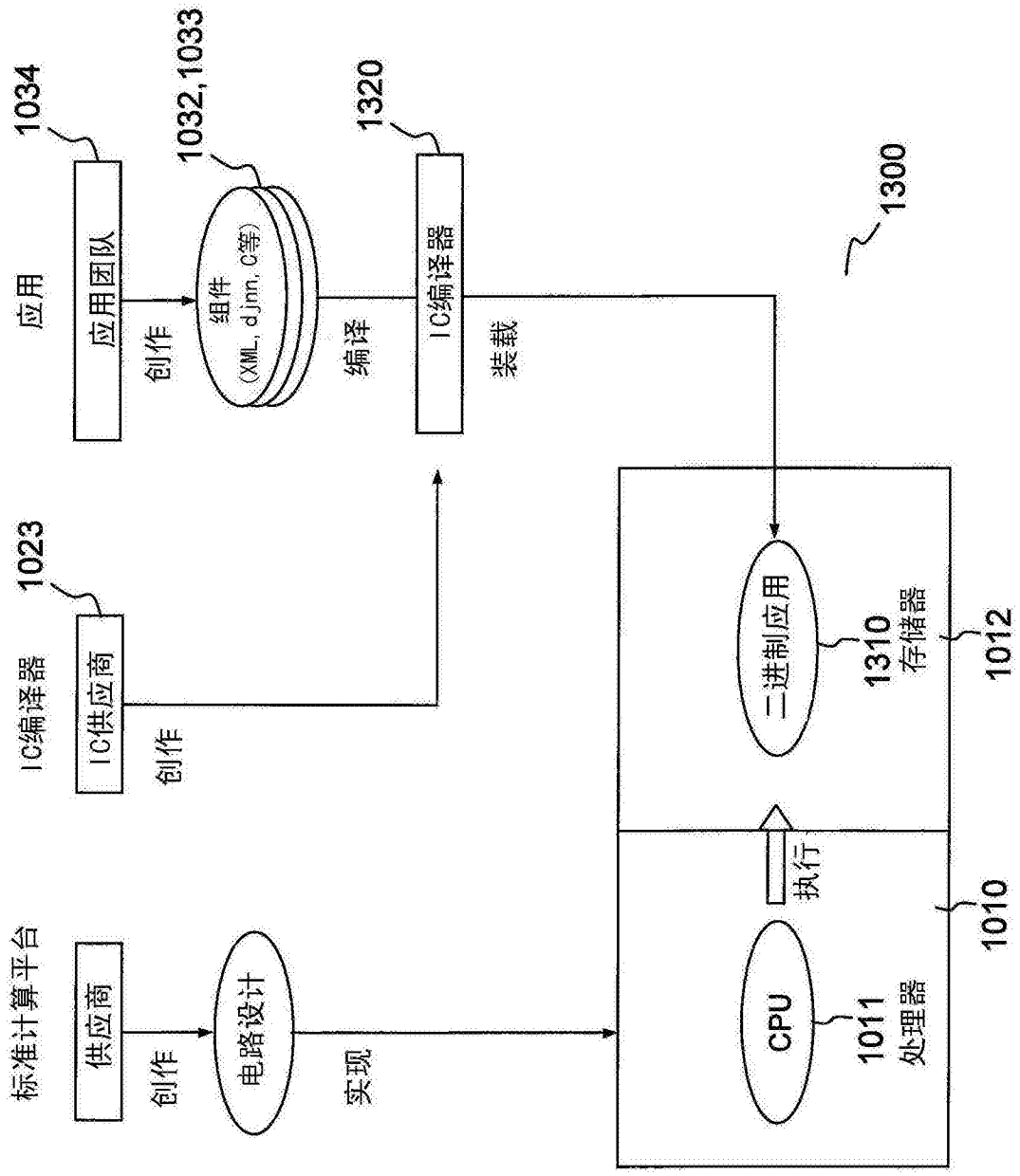

1034
应用
应用团队
创作
1032,1033
组件
(XML, d jnn, C等)
编译
1320
IC编译器
装载
1300
1023
IC编译器
IC供应商
创作
二进制应用
1310
存储器
1012
执行
1010
标准计算平台
供应商
创作
电路设计
实现
CPU
1011
处理器

图13

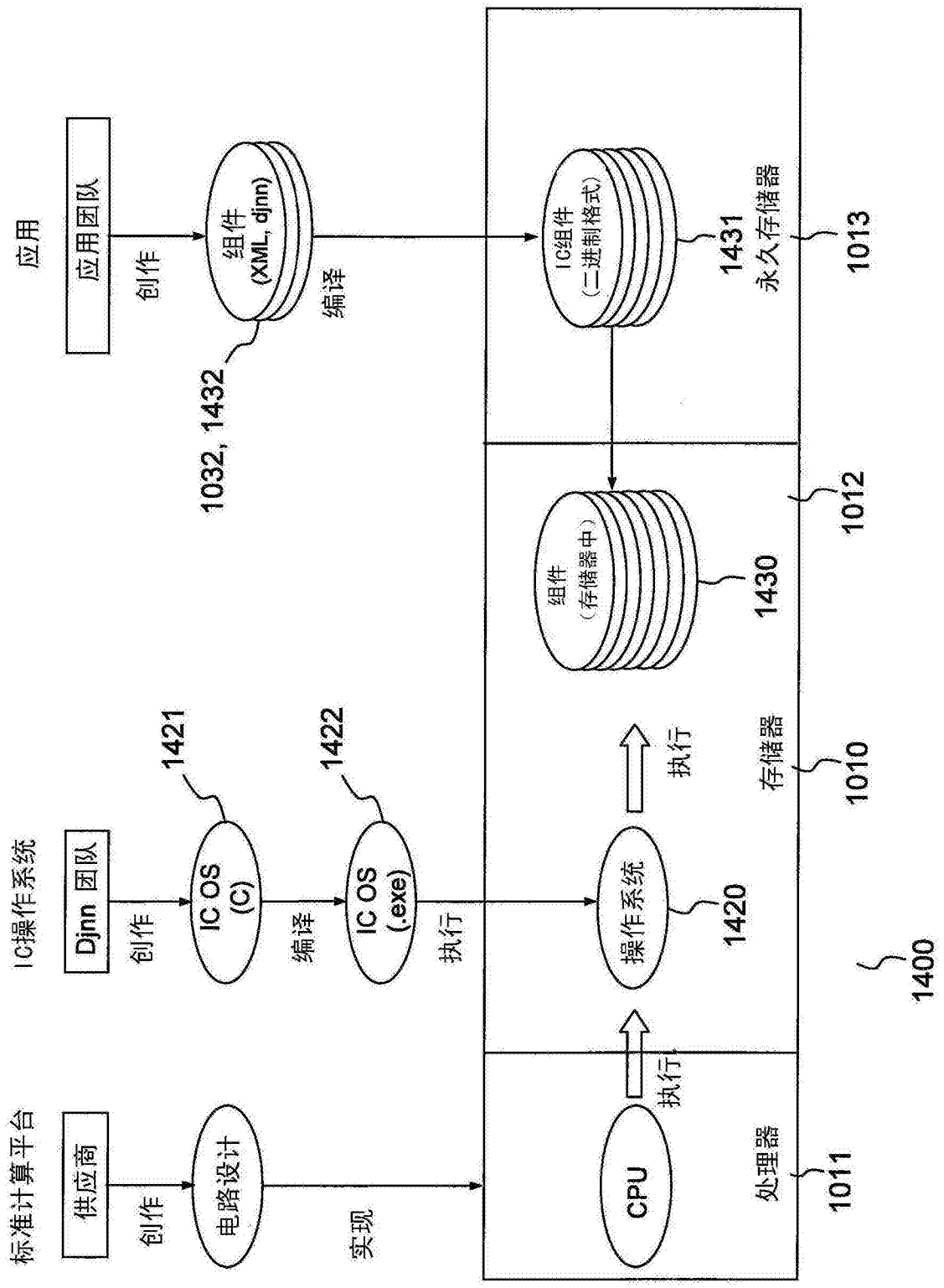

应用
应用团队
创作
组件
(XML, djnn)
编译
1032, 1432
IC组件
(二进制格式)
1431
永久存储器
1013
组件
(存储器中)
1430
1012
IC操作系统
Djnn 团队
创作
IC OS
(C)
1421
编译
IC OS
(.exe)
1422
执行
操作系统
1420
执行
存储器
1010
标准计算平台
供应商
创作
电路设计
实现
CPU
执行
处理器
1011
1400

图14

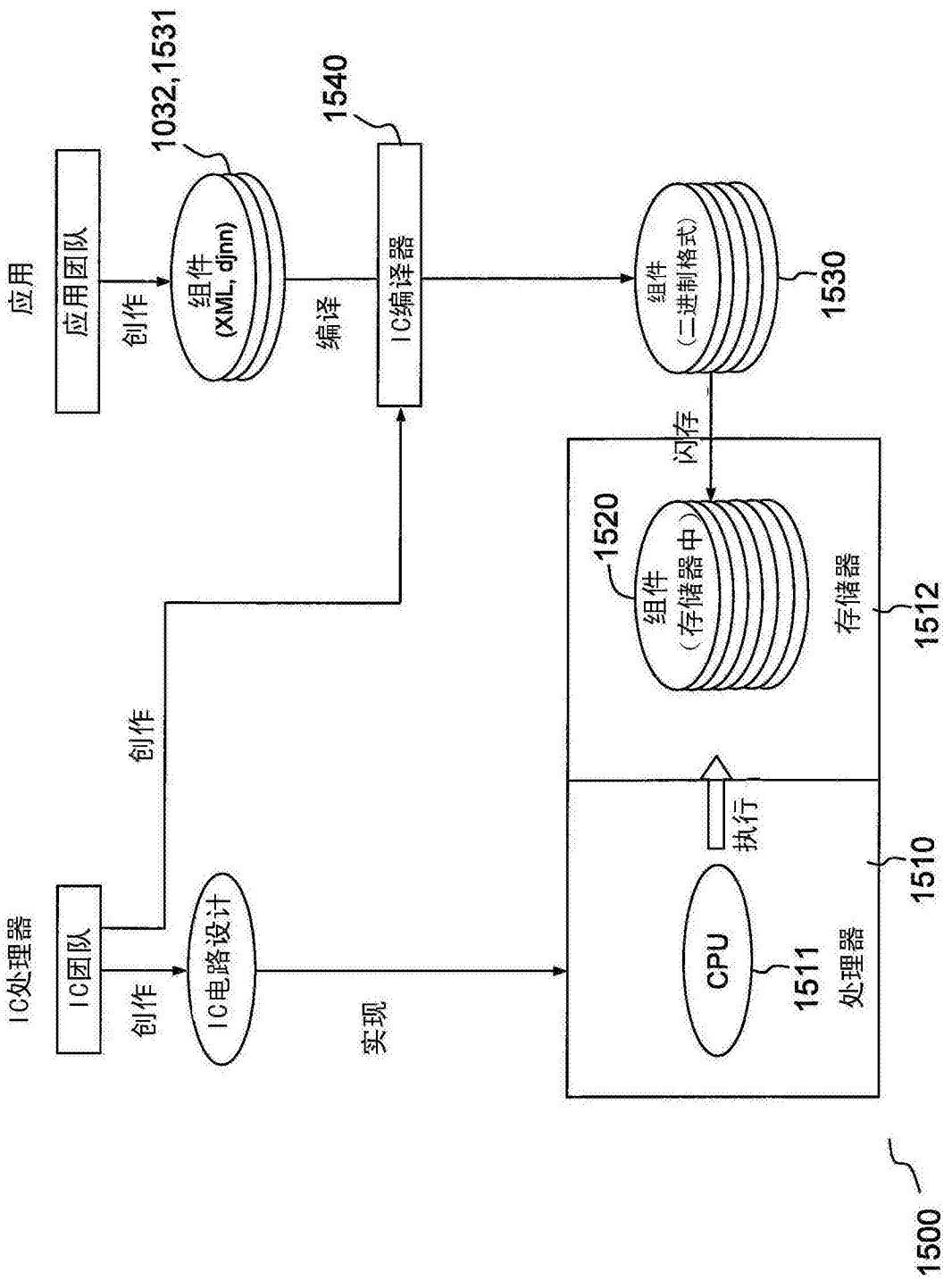

应用
应用团队
创作
1032,1531
组件
(XML, djnn)
编译
1540
IC编译器
组件
(二进制格式)
1530
闪存
1520
组件
(存储器中)
存储器
1512
创作
IC处理器
IC团队
创作
IC电路设计
实现
执行
CPU
1511
处理器
1510
1500

图15

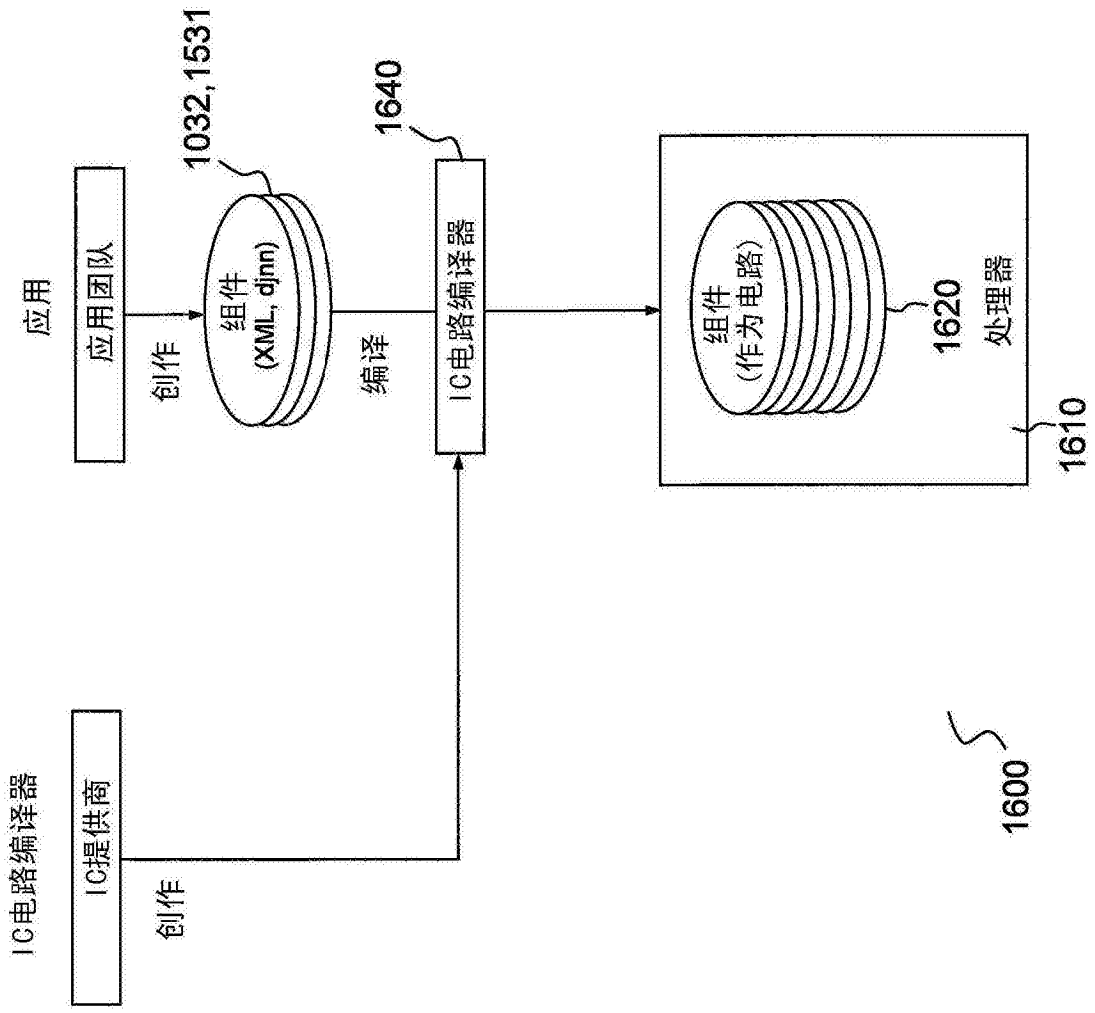

应用
应用团队
创作
1032,1531
组件
(XML, dinn)
编译
1640
IC电路编译器
组件
(作为电路)
1620
处理器
1610
1600
IC电路编译器
IC提供商
创作

图16

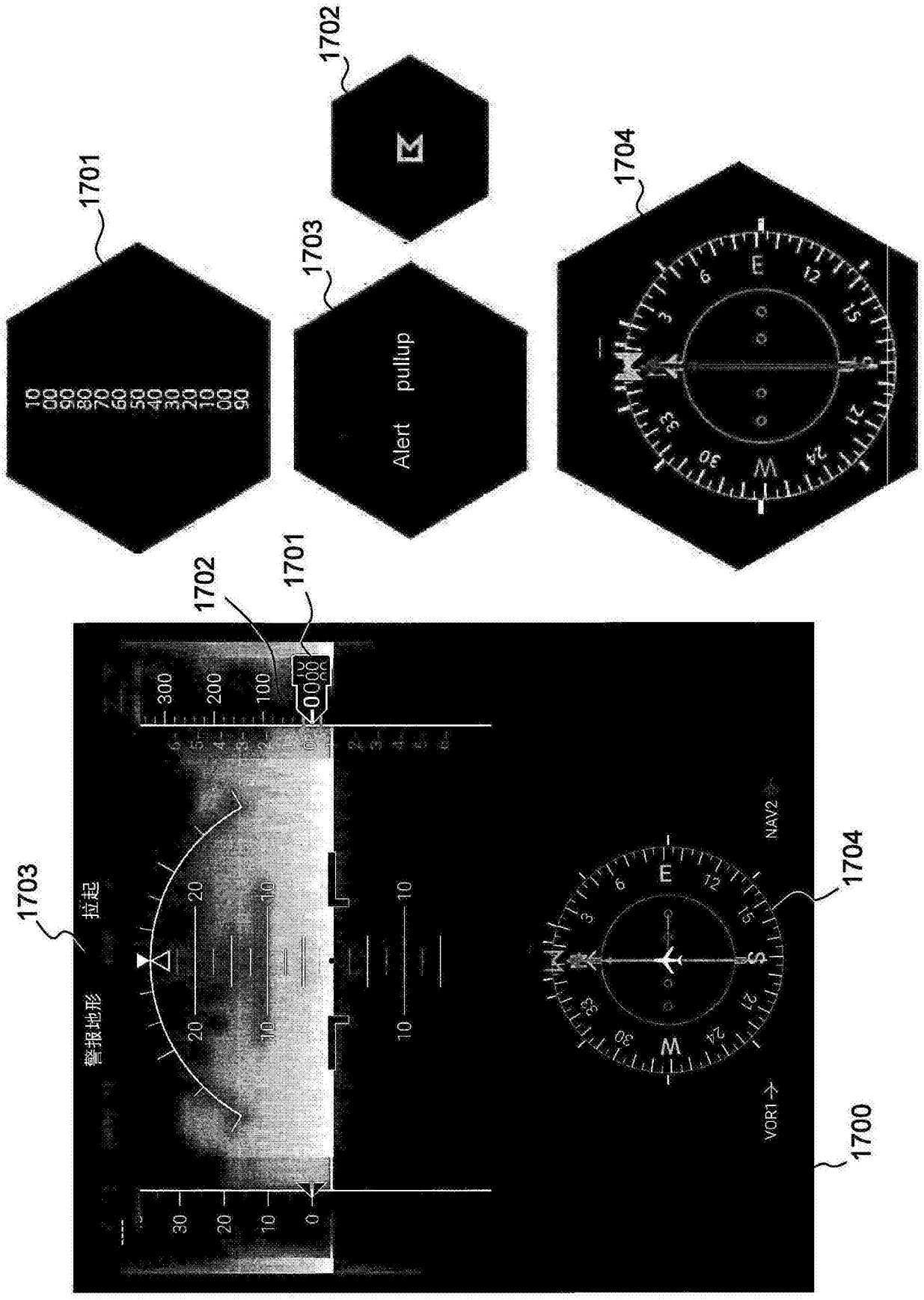


图17

```
<component id="alarms" >
    <double id="threshold" value="0.000000" />
    <double id="altitude" value="1.000000" />
    <base:ascending-comparator id="ascending" left="1.000000" right="0.000000" />
    <connector id="altToLeft" in="altitude" out="ascending/left" />
    <connector id="thresToRight" in="threshold" out="ascending/right" />
    <base:switch id="switch" initial="false" >
        <component id="false" />
        <component id="true" >
            <graphics:group id="alarmsSVG" >
                <component id="AlarmTerrain" >
                    <graphics:font-weight id="font-weight" weight="75" />
                    <graphics:font-size id="font-size" unit="px" size="18.000000" />
                    <graphics:font-family id="font-family" family="Roboto" />
                    <graphics:fill-color id="fill" r="189.000000" g="55.000000" b="56.000000" />
                    <graphics:fill-opacity id="fill-opacity" a="1.000000" />
                    <graphics:font-size id="font-size" size="18.179200" />
                    <graphics:no-outline id="no-stroke" />
                    <graphics:text id="AlarmTerrain" x="234.926730" y="16.032656"
                        dx="0.000000" dy="0.000000" encoding="UTF-8" text="Alert terrain" />
                </component>
                <component id="AlertPullUP" >
                    <graphics:font-size id="font-size" size="18.179200" />
                    <graphics:font-weight id="font-weight" weight="75" />
                    <graphics:font-size id="font-size" unit="px" size="18.000000" />
                    <graphics:font-family id="font-family" family="Roboto" />
                    <graphics:fill-color id="fill" r="189.000000" g="55.000000" b="56.000000" />
                    <graphics:fill-opacity id="fill-opacity" a="1.000000" />
                    <graphics:no-outline id="no-stroke" />
                    <graphics:text id="AlertPullUP" x="397.706850" y="15.502353"
                        dx="0.000000" dy="0.000000" encoding="UTF-8" text="Pullup" />
                </component>
            </graphics:group>
        </component>
    </base:switch>
    <connector id="fsmTosw" in="ascending/result" out="switch/state" />
</component>
```

图18

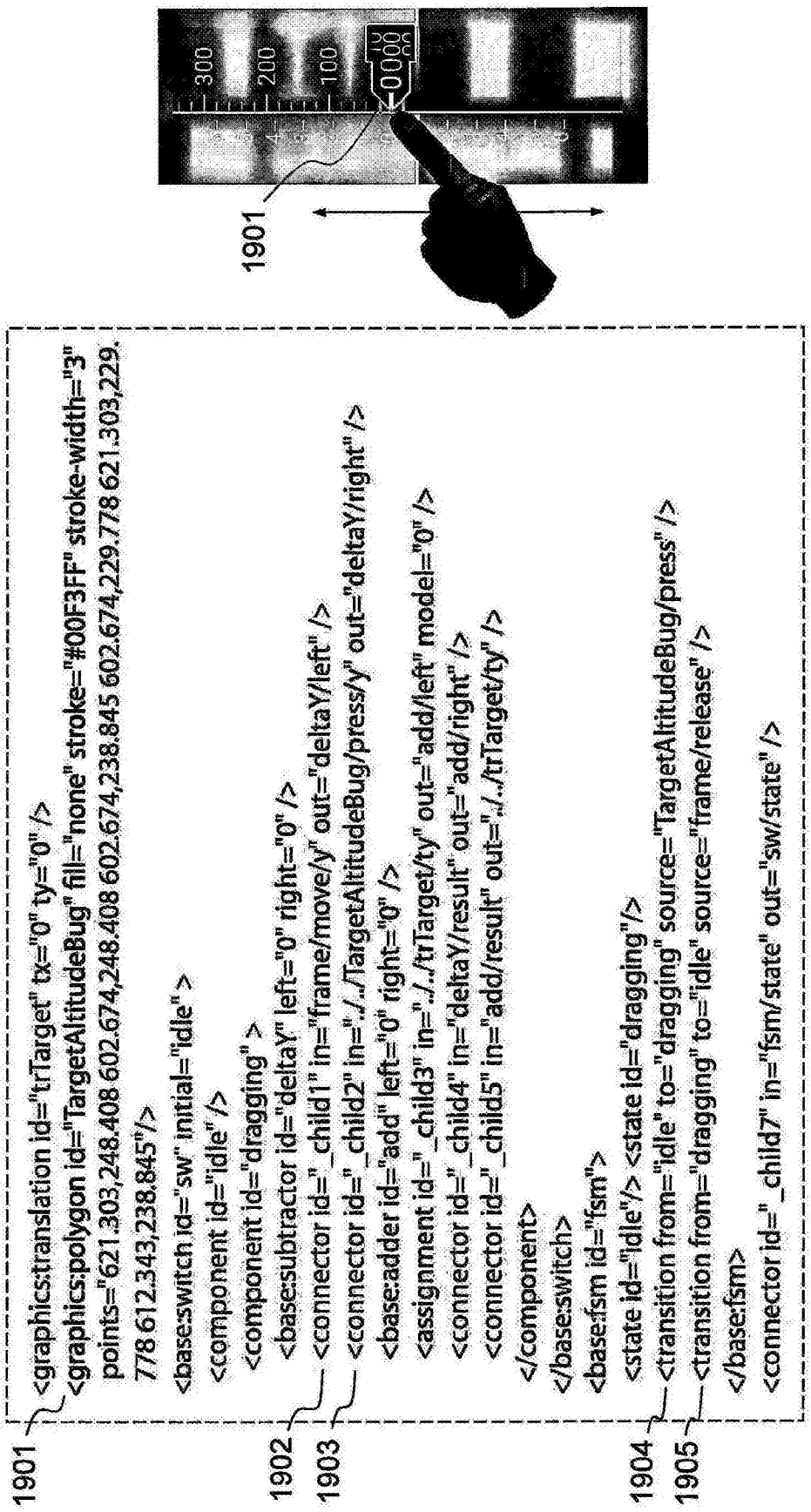

1901
<graphics:translation id="trTarget" tx="0" ty="0" />
<graphics:polygon id="TargetAltitudeBug" fill="none" stroke="#00F3FF" stroke-width="3"
points="621.303,248.408 602.674,248.408 602.674,238.845 602.674,229.778 621.303,229.
778 612.343,238.845"/>
<base:switch id="sw" initial="idle" >
<component id="idle" />
<component id="dragging" >
<base:subtractor id="deltaY" left="0" right="0" />
<connector id="_child1" in="frame/move/y" out="deltaY/left" />
<connector id="_child2" in="../../TargetAltitudeBug/press/y" out="deltaY/right" />
<base:adder id="add" left="0" right="0" />
<assignment id="_child3" in="../../trTarget/ty" out="add/left" model="0" />
<connector id="_child4" in="deltaY/result" out="add/right" />
<connector id="_child5" in="add/result" out="../../trTarget/ty" />
</component>
</base:switch>
<base:fsm id="fsm">
<state id="idle"/> <state id="dragging"/>
<transition from="idle" to="dragging" source="TargetAltitudeBug/press" />
<transition from="dragging" to="idle" source="frame/release" />
</base:fsm>
<connector id="_child7" in="fsm/state" out="sw/state" />
1902
1903
1904
1905

图19

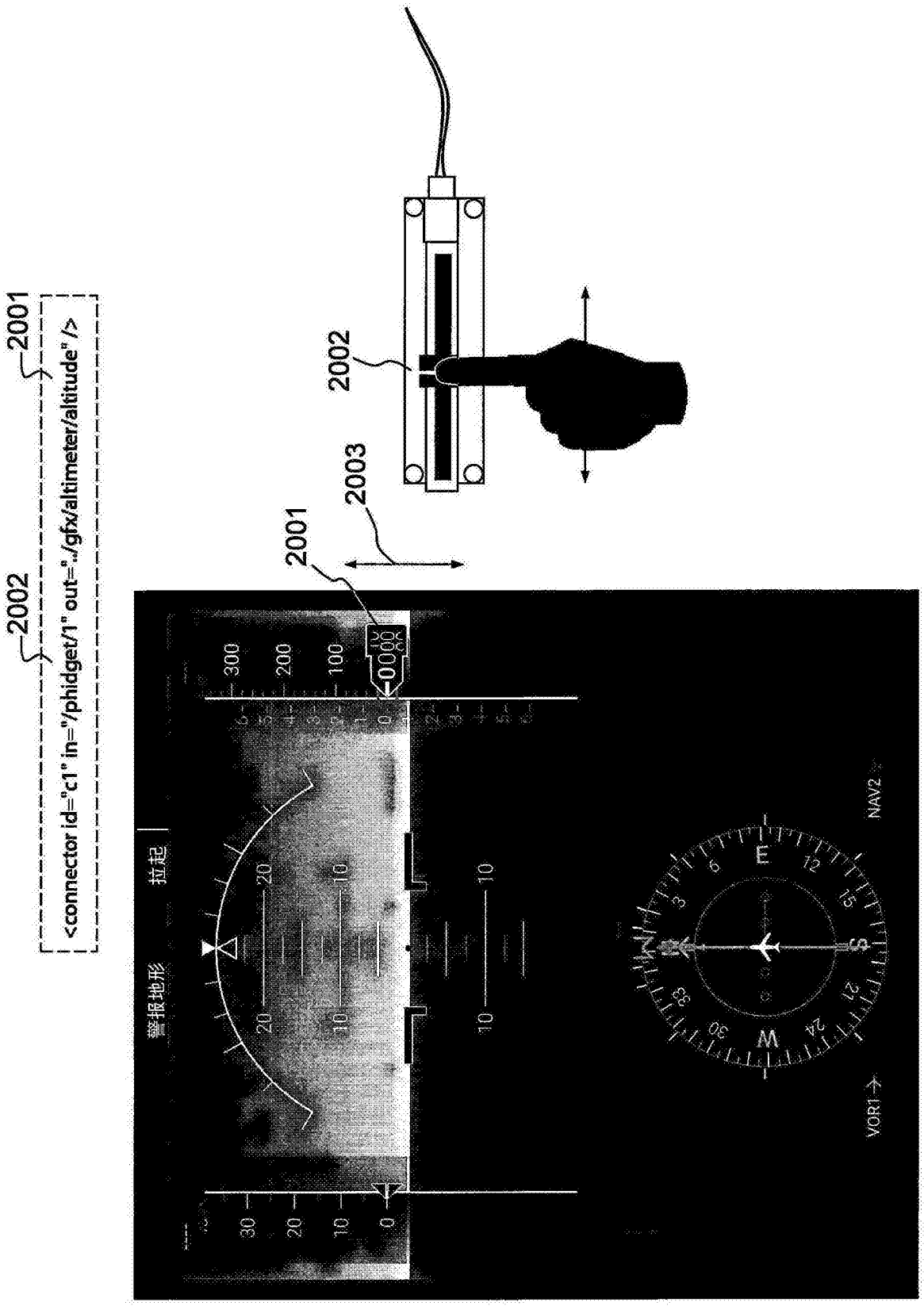

2001
<connector id="c1" in="/phidget/1" out="../gfx/altimeter/altitude" />
2002
2002
2003
2001
警报地形
拉起
NAV2
VOR1

图20