



(12) 发明专利

(10) 授权公告号 CN 102087577 B

(45) 授权公告日 2015.05.20

(21) 申请号 201010576562.2

US 5046002 A, 1991.09.03, 说明书第2栏第

(22) 申请日 2010.12.07

1-33行.

CN 1835507 A, 2006.09.20, 全文.

(30) 优先权数据

12/632,253 2009.12.07 US

审查员 孙娟

(73) 专利权人 SAP 欧洲公司

地址 德国瓦尔多夫

(72) 发明人 约肯·格特勒 兰·格罗斯
亚哈利·舍曼 阿里尔·坦曼

(74) 专利代理机构 北京市柳沈律师事务所

11105

代理人 邵亚丽

(51) Int. Cl.

G06F 9/54(2006.01)

(56) 对比文件

US 5757925 A, 1998.05.26, 说明书第3栏第
28行 - 第6栏第30行.

US 5757925 A, 1998.05.26, 说明书第3栏第
28行 - 第6栏第30行.

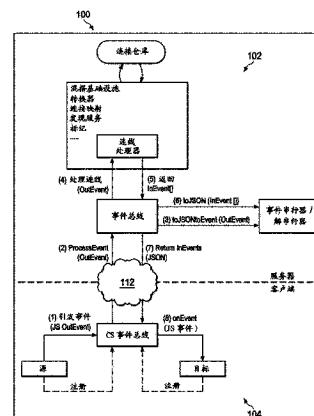
权利要求书2页 说明书8页 附图8页

(54) 发明名称

与位置无关地执行用户接口操作

(57) 摘要

本公开涉及系统、方法和软件，它们涉及与位置无关地执行用户接口操作。所述执行可以包括：通过客户端侧用户接口识别用户交互，该客户端侧用户接口与服务器侧计算机可通信地连接。然后动态地确定在客户端侧还是在服务器侧处理识别出的用户交互。如果用户交互将在客户端侧处理，则调用客户端侧动作处理程序。如果用户交互将在服务器侧处理，则请求服务器侧动作处理程序处理该交互。



1. 一种用于在混搭环境中使用连接应用的连线与位置无关地执行用户接口操作的计算机实现的方法,其中所述连线包括一个或多个输出元素、输入元素、一个或多个输入区以及一个或多个输出区,其中所述连线包含转换器,所述转换器根据具体实施的定义来翻译信息,由此使能应用之间的通信,该方法包括由一个或多个处理器执行的下列步骤:

导出所述连线的基于客户端侧的 JavaScript 表示;

导出所述连线的基于服务器侧的 JavaScript 表示;

其中所述基于客户端侧的 JavaScript 表示和所述基于服务器侧的 JavaScript 表示使用相同的元数据并且共享相同的定义;

通过客户端侧用户接口识别用户交互,该客户端侧用户接口与服务器侧计算机可通信地连接,其中所述用户交互触发从一个应用到另一个应用的导航;

动态地确定在客户端侧还是在服务器侧处理识别出的用户交互,其中所述动态地确定根据多个准则的全部来进行,所述准则包括处理能力、往返数量和操作环境、在触发所述导航之前确保应用之间的一致性的难度、以及所述应用仅在客户端侧、基于具有客户端前端的服务器侧、还是基于客户端侧的应用和基于服务器的应用的组合;

如果用户交互将在客户端侧处理,则调用客户端侧动作处理程序;以及

如果用户交互将在服务器侧处理,则请求服务器侧动作处理程序处理该交互。

2. 如权利要求 1 所述的计算机实现的方法,其中请求服务器侧动作处理程序处理交互包括:

确定用户交互包含同步动作还是异步动作;以及

基于所述确定请求同步服务器侧动作处理程序或异步服务器侧动作处理程序之一来处理该交互。

3. 如权利要求 1 所述的计算机实现的方法,其中,动态地确定在客户端侧还是在服务器侧处理识别出的用户交互包括:利用规则仓库来确定在客户端侧还是服务器侧处理识别出的用户交互。

4. 如权利要求 3 所述的计算机实现的方法,其中,所述规则仓库包括主键,其映射到具体的交互。

5. 如权利要求 3 所述的计算机实现的方法,其中,所述规则仓库包括存储各种不同交互动作的动态表达的规则。

6. 如权利要求 1 所述的计算机实现的方法,还包括:利用用户接口代码从服务器侧计算机接收客户端侧动作处理程序。

7. 一种在混搭环境中使用连接应用的连线通过知晓全景的应用间通信基础设施进行通信的计算机系统,其中所述连线包括一个或多个输出元素、输入元素、一个或多个输入区以及一个或多个输出区,其中所述连线包含转换器,所述转换器根据具体实施的定义来翻译信息,由此使能应用之间的通信,所述计算机系统包括:存储器,其存储客户端侧用户接口和客户端侧动作处理程序;以及

一个或多个处理器,其在执行时:

导出所述连线的基于客户端侧的 JavaScript 表示;

导出所述连线的基于服务器侧的 JavaScript 表示;

其中所述基于客户端侧的 JavaScript 表示和所述基于服务器侧的 JavaScript 表示使

用相同的元数据并且共享相同的定义；

通过客户端侧用户接口识别用户交互，该客户端侧用户接口与服务器侧计算机可通信地连接，其中所述用户交互触发从一个应用到另一个应用的导航；

动态地确定在客户端侧还是在服务器侧处理识别出的用户交互，其中所述动态地确定根据多个准则的全部来进行，所述准则包括处理能力、往返数量和操作环境、在触发所述导航之前确保应用之间的一致性的难度、以及所述应用仅在客户端侧、基于具有客户端前端的服务器侧、还是基于客户端侧的应用和基于服务器的应用的组合；

如果用户交互将在客户端侧处理，则调用客户端侧动作处理程序；以及

如果用户交互将在服务器侧处理，则请求服务器侧动作处理程序处理该交互。

8. 如权利要求 7 所述的计算机系统，其中所述一个或多个处理器请求服务器侧动作处理程序处理该交互包括该一个或多个处理器：

确定该用户交互包含同步动作还是异步动作；以及

基于所述确定请求同步服务器侧动作处理程序或异步服务器侧动作处理程序之一来处理该交互。

9. 如权利要求 7 所述的计算机系统，其中，所述存储器还存储规则仓库，并且其中所述一个或多个处理器利用该规则仓库来动态地确定在客户端侧还是在服务器侧处理识别出的用户交互。

10. 如权利要求 9 所述的计算机系统，其中，所述规则仓库包括主键，其映射到具体的交互。

11. 如权利要求 9 所述的计算机系统，其中，所述规则仓库包括存储各种不同交互动作的动态表达的规则。

12. 如权利要求 9 所述的计算机系统，所述一个或多个处理器还可操作用于利用用户接口代码从服务器侧计算机接收客户端侧动作处理程序。

与位置无关地执行用户接口操作

技术领域

[0001] 本公开涉及有关与位置无关地执行用户接口操作 (location-independent execution of user interface operation) 的系统、方法和软件。例如，本公开提供了一种框架 (framework)，其有助于在客户端侧或服务器侧的操作处理之间进行切换，这种切换往往是实时的。

背景技术

[0002] 图形用户接口 (GUI) 允许计算机用户通过显示接口控制计算机系统以及输入和修改计算机系统上的数据。一些应用在客户端服务器体系结构中运行的客户端程序中实现用户接口 (UI)，例如，网页浏览器。UI 一般包括可由用户操作以与应用交互的控件。控件可以包括按钮、菜单、下拉菜单、对话框、滚动条和使得用户能够查看应用数据、调用应用功能以及以其他方式与应用交互的任何其他控件。每个控件都具有相关联的控件状态，并且应用具有相关联的 UI 状态。控件状态和 UI 状态能够基于用户交互而改变。在 GUI 中导航一般包括移动诸如鼠标的指示设备到活动元素 (active element) 以及通过点击该元素来启动 (activate) 该元素。或者，来自键盘设备的各种键击（例如按下“tab”键）允许用户遍历 GUI 元素。

[0003] 用户接口（或网页）应用框架的开发往往涉及关于操作或用户交互（例如用户点击链接或按钮以触发操作）应当在客户端侧（可能使用在浏览器内运行的 Java Script 框架）处理还是在服务器侧（可能使用在网页应用服务器上运行的 Java 框架）处理的争论。通过在客户端服务器体系结构中运行的客户端程序中绘制应用（例如，网页浏览器）的 UI 状态，可以生成应用的可视表示 (visual representation)。

发明内容

[0004] 本公开的一个方面涉及与位置无关地执行用户接口操作或交互。例如，一种用于与位置无关地执行用户接口操作的计算机程序产品包括存储计算机可读指令的有形存储介质。在一个实例中，所述指令可操作用于识别通过客户端侧用户接口的用户交互，所述客户端侧用户接口与服务器侧计算可通信地耦合。然后，所述指令动态地确定在客户端侧还是在服务器侧处理识别出的用户交互。在一些情况下，至少部分地根据多个标准来进行所述动态确定，所述多个标准包括处理能力、往返数量以及操作环境。如果用户交互将在客户端侧处理，则所存储的指令调用客户端侧动作处理程序。否则，如果用户交互将在服务器侧处理，则指令请求服务器侧动作处理程序处理该交互。

[0005] 不同的实施例可以实现或利用以下特征中的一个或多个。例如，所存储的指令可以要求服务器侧动作处理程序处理交互包括：确定该用户交互包含同步动作还是异步动作，并基于所述确定请求同步服务器侧动作处理程序或异步服务器侧动作处理程序之一处理该交互。在另一个例子中，所存储的指令可以利用规则仓库或其他类似的逻辑来确定在客户端侧还是服务器侧处理识别出的用户交互。这个示例性的规则仓库可以包括主键，该

主键例如直接地映射到具体的交互,或者该规则仓库可以包括存储不同交互动作的动态表达的规则。在再一个示例中,所存储的指令还可操作用于利用用户接口从服务器侧计算机接收客户端侧动作处理程序。在用户交互包含多个事件周期的实例中,动态确定在客户端侧还是服务器侧处理识别出的用户交互可以针对每个事件周期发生。

[0006] 尽管被一般地描述为处理和转换相应数据的计算机实现的软件,但是一些或所有方面可以是计算机实现的方法或者进一步包括在相应的系统或用于执行所描述功能的其他设备中。本公开的这些和其他方面和实施例的细节将在附图和以下描述中阐述。本公开的其他特征、对象和优点将从说明书、附图和权利要求中变得清晰。

附图说明

[0007] 图 1 示出了根据本公开的一种实现方式的、提供与位置无关地执行用户接口操作或交互的示例系统;

[0008] 图 2 是图 1 中描述的连线 (wire) 的示例配置。

[0009] 图 3 是图 2 中描述的连线的示例性实例。

[0010] 图 4 是使用图 1 中描述的系统在运行时的示例性连线事件周期。

[0011] 图 5 示出了图 1 中描述的系统的各种使用情况。

[0012] 图 6A-6C 示出了根据本公开的一种实现方式的、图 5 中的使用情况的各种示例流程。

[0013] 图 7 示出了根据本公开的一种实现方式的示例性连线运行时序列。

具体实施方式

[0014] 本公开内容的一个方面针对提供与位置无关地执行用户接口操作或交互的系统。更具体来说,该系统提供了用于以逐个情况的方式 (a case by casemanner) 解决“客户端侧”对“服务器侧”这一问题的框架。换句话说,在企业解决方案中可能需要控制请求的源,特别是对于外部内容更是如此。例如,服务提供商可能限制来自客户机器或托管环境 (hosting environment) 的对服务的访问。再例如,解决方案可以实现安全性,从而使外部网页内容可以从服务器侧提取,但不能直接通过客户端浏览器提取 (这可以帮助控制数据大小,利用黑名单过滤掉可疑网站等等)。对于大多数场景,系统可以实现服务器侧方式。然而在特定运行时场景中,系统能够实现这样的机制:如果在客户端侧执行用户交互对于该场景而言是合理的或者满足特定准则,则就在客户端侧执行用户交互。这些准则可以包括确定这是否改善系统性能或总体操作,这是否节省往返 (roundtrip),或者这是否不会过于复杂以至于不能处理,等等。在一些情况下,对于使用哪个模型来执行交互的决定可以对于开发者透明地通过框架来选择 (即,开发者不必决定是使用客户端侧方式还是服务器侧方式)。取决于运行的场景 / 应用,框架动态地决定实施那种方式。这也取决于正在运行所述场景 / 应用的环境 (例如,如果应用正在门户中运行则结果可能不同)。简言之,系统不强制实施事先的 (或一般的) 定义,而是提供实施逐个情况地进行决定的能力以及针对特定情况或场景的最优化的解决方案。这样,在特定配置中,这种基础设施可以不限于客户端侧方式或服务器侧方式,可以不限于特定系统全景 (landscape),并且可以不限于任何具体的 UI 技术。

[0015] 一个例子属于往返最优化领域。在该示例中，用户希望触发到另一个应用的导航。取决于场景，这可以经由直接的客户端侧调用来完成（例如在简单场景中，其中应用能够作为相对独立的应用来执行操作），或者可以经由服务器往返来完成（例如在较复杂场景中，其中，框架帮助确保在触发导航之前运行在同一页面上的其他应用之间的一致性）。所述框架并非仅仅提供多种可能性中的一种，而是提供一个触发导航的 API，同时在总体上由框架决定在哪里执行该导航（即在客户端侧执行还是在服务器侧执行）。

[0016] 另一个例子属于混搭 (mashup)（即，若干小的“应用”或“芯片”的复合 (composition)，这些“应用”或“芯片”经由“连线” (wire) 连接以进行简单的通信）领域，所述混搭包括来自服务器侧或客户端侧的组件 (component) 或其他技术。为了构筑混搭，一般以基于事件的通信基础设施为基础来完成“连线” (wiring)。可以存在纯粹的基于客户端侧的混搭环境，也存在基于服务器的环境。所描述的框架不是以静态方式定义处理交互的位置，而是能够根据不同的约束来执行操作。这些示例性的约束可以包括：作为混搭场景和当前连线的一部分的应用是哪种类型？例如，该应用可以包括仅仅是客户端侧的应用、仅仅是基于服务器的应用（具有客户端前端）、或者组合 (combination)。为了帮助避免不期望的往返，（例如如果用户点击控件，该控件改变某个其它客户端侧应用的某个状态，则）该框架可以在客户端侧执行用户交互。或者，该框架可以确定例如，i) 用户交互涉及过于复杂的操作；ii) 难以确保一致的屏幕，因此请求操作主要在服务器侧完成；或者 iii) 混搭场景具有混合运行时技术（客户端侧和服务器侧），从而需要组合的事件周期。

[0017] 转到图示的实施例，图 1 示出了示例系统 100，其实现了知晓全景的应用间通信基础设施。系统 100 可以是跨越一个或多个网络的分布式客户端 / 服务器系统。在这样的实现方式中，可以以使用任何标准的或私有的加密算法的加密格式来通信或存储数据。但是系统 100 可以处于专用企业环境（在局域网或子网上）中，或处于任何其他适当的环境中，而不会偏离本公开的范围。系统 100 可以包括服务器 102、一个或多个客户端 104 以及一个或多个网络 112，或者系统 100 可以可通信地与它们连接。

[0018] 服务器 102 包括电子计算设备，其可操作用于接收、发送、处理和存储与系统 100 相关联的数据。总体来说，图 1 仅仅提供了可与本公开一起使用的计算机的一个例子。一般来说，每个计算机本意是包括任何适当的处理设备。例如，尽管图 1 示出了一个服务器，其可与本公开一起使用，但系统 100 也可以使用服务器之外的计算机以及服务器池来实现。事实上，服务器可以是任何计算机或处理设备，例如刀片服务器、通用个人计算机 (PC)、Macintosh、工作站、基于 Unix 的计算机、PDA、提供离线功能（设备中的小型足迹 (footprint) 服务器）的智能电话、以及其他任何适当的设备。换句话说，本公开考虑到了通用计算机之外的计算机以及没有传统操作系统的计算机。服务器可以被适配为执行任何操作系统，包括 Linux、UNIX、WindowsServer 或其他任何适当的操作系统。根据一个实施例，服务器还可以包括网页服务器和 / 或邮件服务器，或者可通信地与它们连接。

[0019] 服务器可以包括本地存储器。存储器可以是任何有形的计算机可读存储器，其可以包括采用易失性或非易失性存储器形式的任何存储器或数据库模块，包括但不限于磁介质、光介质、随机存取存储器 (RAM)、只读存储器 (ROM)、可移动介质、或其它任何适当的本地或远程存储器组件。除其他项目外，存储器可以包括用于与位置无关地执行用户接口操作或交互以及多个应用的框架，后面将对其进行更详细的描述。此外，存储器可以包括一

个或多个操作环境。存储器还可以包括其他类型的数据，例如环境和 / 或应用描述数据、一个或多个应用的应用数据、以及涉及 VPN 应用或服务的数据、防火墙策略、安全或访问日志、打印或其他报告文件、超文本标记语言 (HyperText Markup Language, HTML) 文件或模板、相关或不相关的软件应用或子系统，等等。因此，存储器也可以被看作是数据仓库 (repository)，例如来自一个或多个应用的本地数据仓库。事实上，与位置无关的框架可以驻留在服务器侧和客户端侧（客户端侧逻辑和分派）两者之上。

[0020] 更具体来说，框架可以是能够使用或有助于使用本身也是软件的知晓全景的应用间通信基础设施的任何应用、程序、模块、处理或其它软件。不管具体实现方式如何，“软件”可以包括软件、固件、连线或编程的硬件或包括在有形计算机可读介质上的以上各项的任何组合。事实上，形式转换模块 130 可以以任何适当的计算机语言编写或描述，所述计算机语言包括 C、C++、Java、Visual Basic、编译语言、Perl、任何适当版本的 4GL 等等。例如，形式转换模块 130 可以是复合应用 (composite application)，所述复合应用中的部分可以被实现为 Enterprise Java Beans (EJBs)，或者设计时 (design-time) 组件可以具有将运行时实现生成到不同平台中的能力，所述不同平台例如 J2EE (Java 2 平台企业版)、ABAP (Advanced Business Application Programming, 高级商业应用编程) 对象或微软的 .NET。

[0021] 根据一些实现方式，通信可以采用一个或多个可扩展标记语言 (eXtensible Markup Language, XML) 文档文件或部分或结构化查询语言 (“SQL”) 语句或脚本编辑 (scripting) 语言或脚本语言或扩展语言的脚本的形式。例如，通信结构 145 可以被格式化、存储或定义为文本文件、虚拟存取方法 (Virtual Storage Access Method, VSAM) 文件、平面文件 (flat file)、Btrieve 文件、逗号分隔值 (comma-separated-value, CSV) 文件、内部变量或者一个或多个库中的各种数据结构。除了 XML，其他的示例脚本编辑语言可以包括 JavaScript、层叠样式表单 (Cascading Style Sheets, CSS)、HTML、异步 JavaScript 和 XML (asynchronous JavaScript and XML, AJAX) 等等。

[0022] 服务器也可以包括处理器。处理器执行诸如前述软件的指令，并操控数据以执行服务器的操作，并且可以是例如中央处理单元 (CPU)、刀片 (blade)、专用集成电路 (ASIC) 或现场可编程门阵列 (FPGA)。将会理解到，可以根据具体需要使用多个处理器，并且提到处理器时旨在根据应用需要包括多个处理器。

[0023] 网络 112 便利计算机服务器和任何其他本地或远程计算机，如客户端 104，之间的无线或有线通信。所述网络可以是企业或安全网络的全部或一部分。在另一个例子中，网络可以是跨越有线或无线链路的、仅仅位于服务器和客户端之间的 VPN。这样的示例性无线链路可以经由 802.11a、802.11b、802.11g、802.20、WiMax 等等。尽管被图示为单个或连续网络，但所述网络可以在逻辑上被划分为各种不同的子网或虚拟网络，这不会偏离本公开的范围，只要网络的至少一部分可以便利服务器和至少一个客户端之间的通信即可。例如，服务器可以通过一个子网可通信地连接到仓库，同时通过另一个子网可通信地连接到特定的客户端。换言之，所述网络包括可操作用来便利系统 100 中的各种计算组件之间通信的任何内部或外部网络、网络、子网络或它们的组合。网络可以在网络地址之间传送例如网际协议 (IP) 分组、帧中继帧、异步传输模式 (ATM) 信元、语音、视频、数据和其他适当的信息。所述网络可以包括一个或多个局域网 (LAN)、无线接入网 (radio access network, RAN)、城

域网 (MAN)、广域网 (WAN)、被称为互联网的全球计算机网络的全部或一部分、和 / 或位于一个或多个地点的其他任何通信系统。在特定实施例中，网络可以是用户可经由特定的本地或远程客户端 104 访问的安全网络。

[0024] 客户端 104 可以是可操作用来使用任何通信链路与服务器或网络连接或通信的任何计算设备。在高层，每个客户端包括或执行至少 GUI，并且每个客户端包括可操作用来接收、发送、处理和存储与系统 100 相关联的任何适当数据的电子计算设备。此外，客户端包括有形存储器和一个或多个处理器。将会理解到，可以存在可通信地连接到服务器的任何数量的客户端。再有，可以适当地可交换地使用“客户端”、“本地机”和“用户”而不会偏离本公开的范围。另外，为了便于说明，每个客户端都被描述为被一个用户使用。但是本公开考虑到了许多用户可以使用一个计算机，或者一个用户可以使用多个计算机。本公开所使用的客户端旨在包括个人计算机、触摸屏终端、工作站、网络计算机、kiosk、无线数据端口、智能电话、个人数字助理 (PDA)、这些或其他设备中的一个或多个处理器、或任何其他适当的处理设备。例如，客户端可以是可操作用来与外部或非安全网络无线连接的 PDA。在另一个例子中，客户端可以包括膝上型计算机，其包括诸如键区、触摸屏、鼠标或能够接受信息的输入设备，以及输出设备，该输出设备传送与服务器或客户端的操作相关联的信息，包括数字数据、可视信息或 GUI。输入设备和输出设备都可以包括固定或可移动存储介质，诸如磁性计算机盘、CD-ROM 或其他适当的介质，用以通过显示，即 GUI 的客户端部分或应用接口，接收来自客户端的用户的输入并向客户端的用户提供输出。

[0025] GUI 可以包括图形用户接口，其可操作用以允许客户端的用户与出于各种目的（如查看应用或其他事务数据）与系统 100 的至少一部分交互。例如，GUI 可以呈现（往往是经由页面或容器）有关在一个或多个应用或应用实例之间通信的业务信息的各种视图。一般来说，GUI 向特定用户提供对于系统 100 所提供的或在系统 100 内通信的数据的有效且用户友好的表示。GUI 可以包括多个可定制图文框或视图，所述可定制图文框或视图具有交互区、下拉菜单和可由用户操作的按钮。GUI 也可以呈现多个入口 (portal) 或应用程序控制面板 (dashboard)。例如，GUI 可以显示允许用户输入和定义用于一个或多个模块的搜索参数的安全网页。应理解，术语“图形用户接口”可用作单数或复数，以描述一个或多个图形用户接口以及特定图形用户接口的每个显示。实际上，不脱离本公开的范围，在适当的情况下，提到 GUI 可能是指应用的前端或组件，以及通过客户端 104 可访问的特定接口。因此，GUI 考虑到了任何在系统 100 中处理信息并有效地向用户呈现结果的图形用户接口，如通用网页浏览器或触摸屏。服务器可以通过网页浏览器（如 Microsoft Internet Explorer 或 Netscape Navigator）从客户端接受数据，并利用网络向浏览器返回适当的 HTML 或 XML 响应。

[0026] 在客户端侧，可以存在客户端动作分派器 (client action dispatcher, CAD) 或事件总线或框架和应用逻辑。在特定配置中，这是由用户（例如通过按下按钮、选择表行等）触发的用户交互的单个进入点。CAD 收到该动作并使用规则引擎对其进行分析。规则引擎可以访问规则仓库。规则仓库包含应如何处理“动作”的规则。在最简单的情况下，规则是简单的映射（例如“按钮点击 → 服务器调用”），但你也可以想象规则是用来确定应当如何处理动作的动态表达 (dynamic expression)。然后，CAD 可以将动作分派给所需要的动作处理程序。更具体来说，尽管连线信息通常存储在服务器中，但运行时处理器 (控制器) 也可

以被实现在客户端侧和 / 或服务器侧，并且可以负责在客户端侧事件周期和服务器侧事件周期之间进行缓和。例如，图 4 示出了服务器侧实现方式。返回图 1，所示的配置描述了补充的客户端侧运行时组件，其帮助使解决方案成为环境不可知的 (environment agnostic)（意思是，能够在事件周期中处理客户端侧 UI 组件和服务器侧 UI 组件两者）。

[0027] 客户端侧调用 (client side call) 直接由客户端调用动作处理器 (client callaction handler) (或其他某个相关模块) 来处理。对于服务器侧调用，CAD (或其他某个相关模块) 可以区别同步调用和异步调用，这样可以向适当的同步和异步处理器发送请求。而且，在服务器侧可以存在运行应用的特定应用容器。从而对于每个通道，能够经由服务器侧动作处理器 (在上面的例子中，一个用于同步调用，一个用于异步调用) 从客户端侧调用应用。

[0028] 在操作混搭配置的一个例子中，如图 1 中的例子所示，通过客户端侧事件总线识别来自源的 RaiseEvent (引发事件) 调用。CAD 触发对服务器侧事件总线的 ProcessEvent (处理事件) 调用，服务器侧事件总线经由 fromJSONtoEvent 调用事件串行器 / 解串行器 (Events Serializer/Deserializer)。服务器侧事件总线经由 ProcessWires 联系混搭基础设施 (可能是连线处理器)，然后 ProcessWires 调用 ReturnInEvent (返回输入事件)。然后，该事件经由 toJSON 被事件串行器 / 解串行器处理，并经由 ReturnInEvents 返回到客户端侧事件。然后，CA 经由 onEvent 将事件发送给目标。

[0029] 图 2 是图 1 中描述的连线的示例配置 200。连线定义对象的结构包含转换器 (transformer)，转换器表示源和目标芯片 (chip) 定义端口之间的实际连线转换数据。换言之，转换器帮助实现基本不兼容组件相互通信并交换信息的通信。转换器可以根据在特定对象类中实现的定义来翻译信息。为了容易处理，简单的转换可以在客户端侧完成，但具有业务逻辑 (服务调用) 的较复杂转换可以在服务器侧完成。这样，混合的解决方案能够实现这种混合的数据流。在这个例子中，连线定义 200 还包括一个或多个输出元素、输入元素、一个或多个输入区以及一个或多个输出区。在一些情形下，可以针对运行时、运行时构建 (runtime authoring) 和管理场景优化连线持续性。在运行时进程中，源的输出事件 (out event) 被转换成目标的输入事件 (in event)。连线数据被传输并能够与容器 (工作空间 / 页面) 相关联，以使得其生命周期能够利用容器的生命周期来管理。将会理解，这是对连线的抽象说明，从而存在导出的基于客户端侧的连线表示 (基于 JavaScript) 以及导出的基于服务器侧的连线表示 (基于 java)。这两种表示同时存在并且共享相同的定义。这种配置帮助实现基于相同的元数据运行客户端侧事件场景和服务器侧事件场景，同时根据环境激活相应的转换器和事件处理器。

[0030] 连线定义可以存储在全局仓库中 (或其他持续性) 并且表示在系统上已经创建的连线。在一些情况中，这些定义可以被不同系统中的各种连线实例访问。当添加新的实例时，可以通过建议机制 (suggestion mechanism) 搜索连线定义。

[0031] 图 3 是图 2 中描述的连线的示例实例 300。该实例 300 将源芯片实例的输出端口 (out-port) 和目标芯片实例的输入端口 (in-port) 映射成特定的连线定义。在一些配置中，实例与特定容器 (页面 / 工作空间) 相关联，并且其帮助提供在运行时中对相应连线定义的快速访问。如上面提到的，连线实例的生命周期往往与容器相关。例如，当页面被删除 / 复制 / 传输时，一般以相同方式处理该页面的连线实例。为了帮助完成这一点，一种实现

方式可以将连线示例作为属性中的 XML 表示本地存储在该页面之下。这种实现方式的例子可以如下：

```
[0032]  <wiring>
[0033]    <target id = ““port = ““wireDefinitionId = ““>
[0034]      <src id = ““port = ““/>
[0035]      <src id = ““port = ““/>
[0036]    </target>
[0037]  </wiring>
```

[0038] 在这个例子中，连线映射在运行时将被解析为存储器中的专用数据结构，该专用数据结构将提供对相关连线数据的快速访问。当然，这种示例实现方式仅仅是用于举例说明的目的，并且可以根据情况适当地使用任何语言、格式或其他实现方式细节。

[0039] 图 5 示出了图 1 中描述的系统的各种使用情况。在高层，该图示绘出了三种使用情况：运行时、运行时构建和管理，每种情况都将在图 6 中更具体地描述。具体来说，图 6A 示出了示例的组件和数据流 600，其用于实现连线的运行时处理；图 6B 示出了示例的组件和数据流 630，其用于实现连线的运行时构建；图 6C 示出了示例的组件和数据流 660，其用于实现连线管理。

[0040] 图 7 示出了根据本公开的一个实现方式，例如数据流 600，的示例连线运行时序列 700。为了清楚演示，以下的描述一般在图 1 示出的环境的上下文中描述方法 700。但是将会理解到，方法 700 可以可替换地由任何其他适合的系统、环境、或适当的系统和环境的组合来执行。

[0041] 具体来说，运行时序列 700 示出了在 702 WiringRuntimeExtension（连线运行时扩展）610 通过 ProcessWires（处理连线）方法调用 WiringProcessor（连线处理器）620。然后在 704，WiringProcessor 调用 iLocalWireInstanceManager（本地连线实例管理器）630，这可能通过 init() 方法进行调用。在 706，通过 createWiringMapForPage()（为页面创建连线映射）调用环境或工作空间访问模块 640，在 708，createWiringMapForPage() 生成并返回 WiringMap（连线映射）。在步骤 710，WiringProcessor（连线处理器）通过联系 iLocalWireInstanceManager 模块确定连线实例 300，连线实例 300 在步骤 712 被返回。可以将这些实例作为任何适当的通信或数据结构，例如数组、对象或列表，来返回。然后在 714 和 716 WiringProcessor 联系环境或工作空间访问模块以确定每个具体的连线实例是否与这个具体动作相关。之后，WiringProcessor 在步骤 718 转换参数并在 720 返回输入端口（inport）事件。

[0042] 尽管本公开使用多个数据和处理流程以及相应的描述说明了与各种公开方法和技术相关的示例技术，但是系统 100 考虑到了使用或实现任何适当的技术来执行这些或其他任务。将会理解，这些技术仅仅是出于示例的目的，并且所描述的技术或类似的技术可以在任何适当的时间执行，包括并行执行、单独执行或组合执行。此外，这些流程中的许多步骤可以同时发生和 / 或以不同于所示出以及所描述的顺序的顺序发生。此外，系统 100 可以使用具有附加步骤、更少步骤和 / 或不同步骤的处理和方法，只要这些处理和方法适当即可。简言之，尽管本公开从特定实现方式以及一般关联方法的角度进行了描述，但这些实现方式和方法的替代方式或置换方式对于本领域技术人员而言将是明显的。因此，其他的

实现方式也在权利要求书的范围之内。

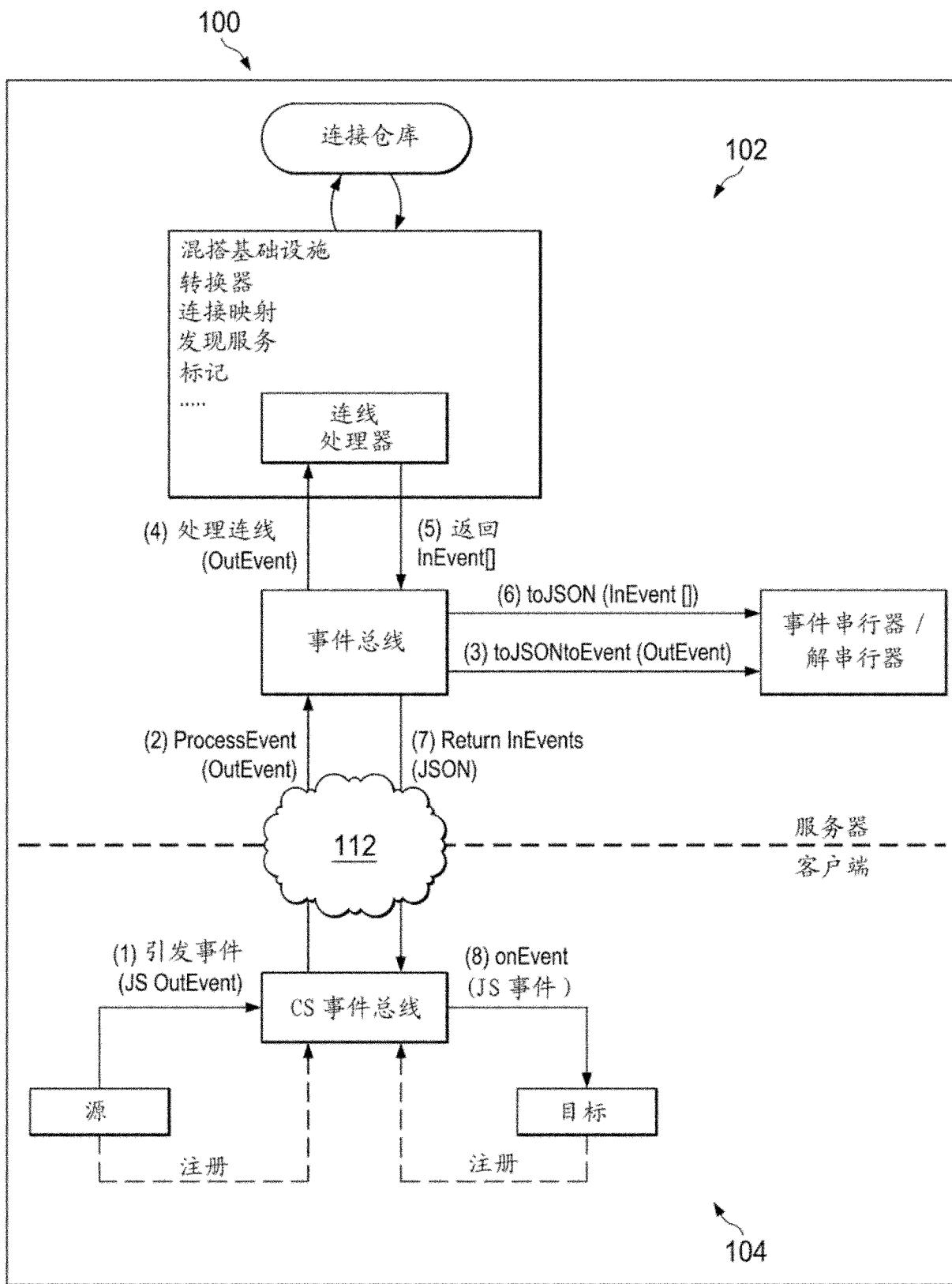


图 1

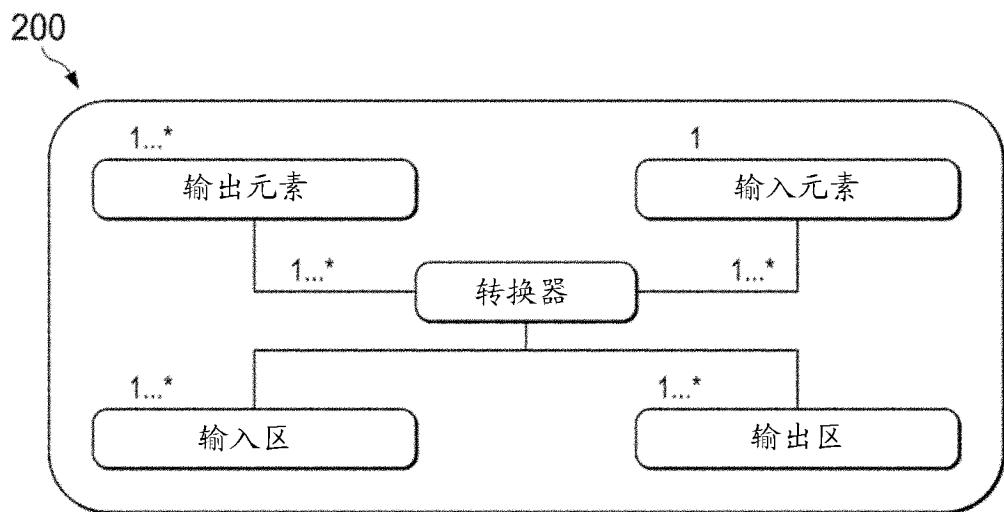


图 2

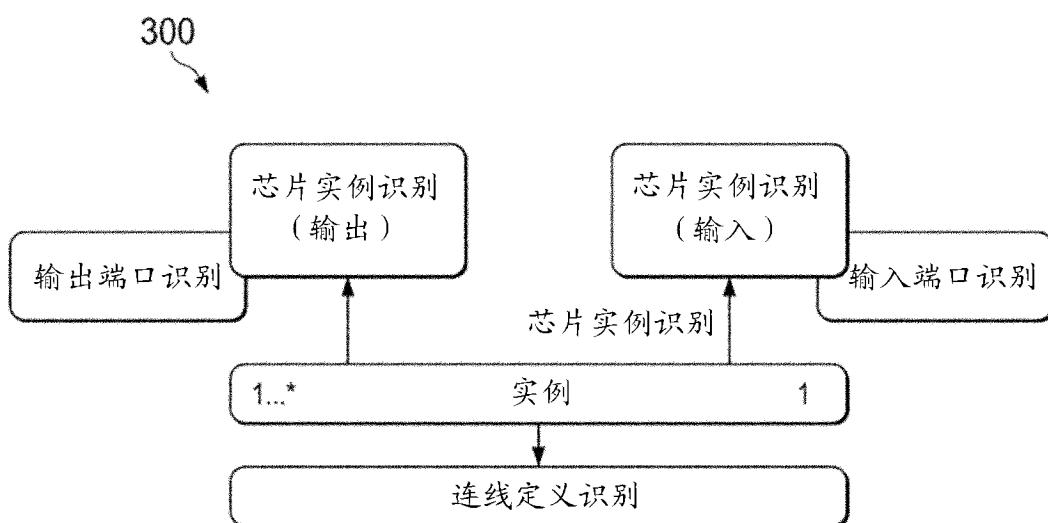


图 3

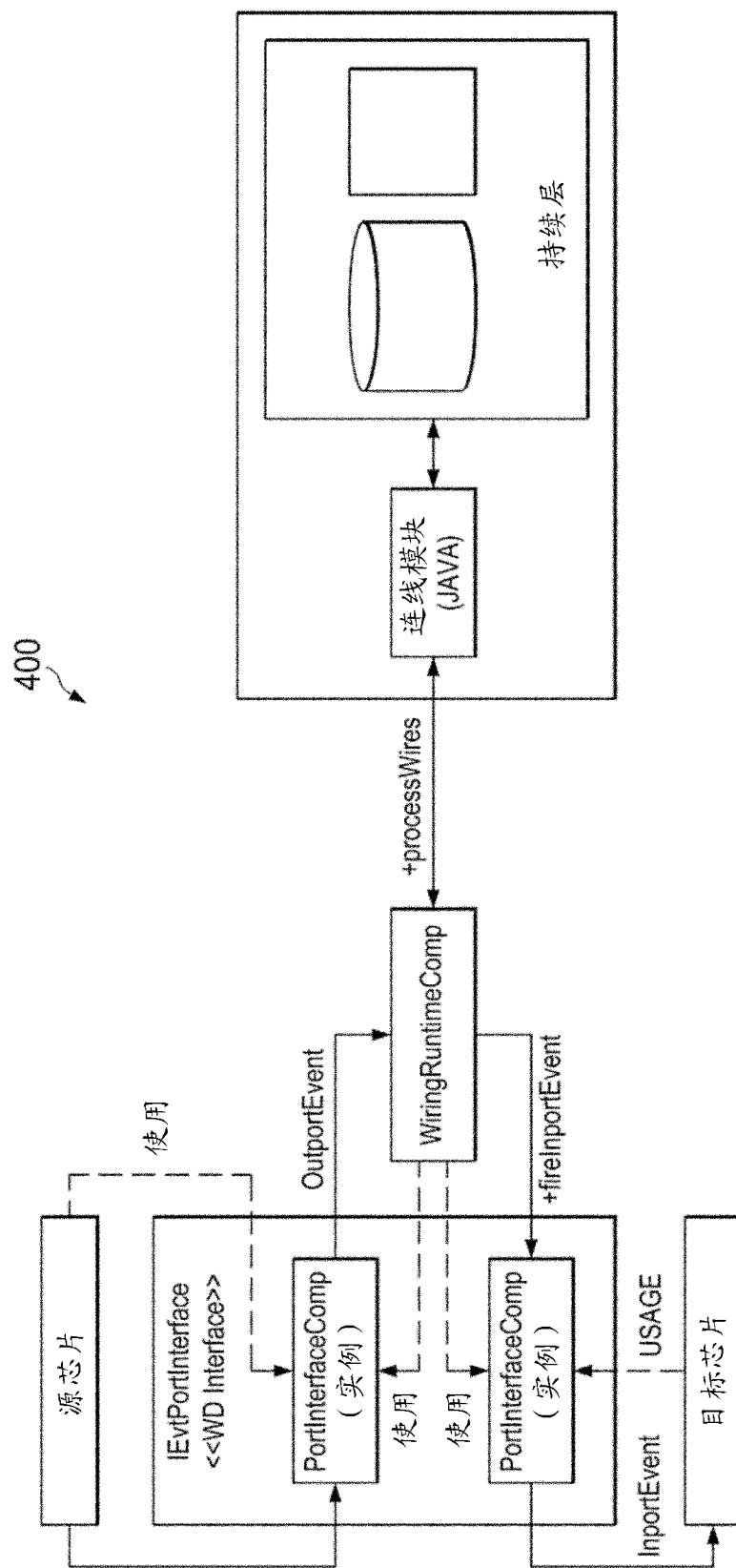


图 4

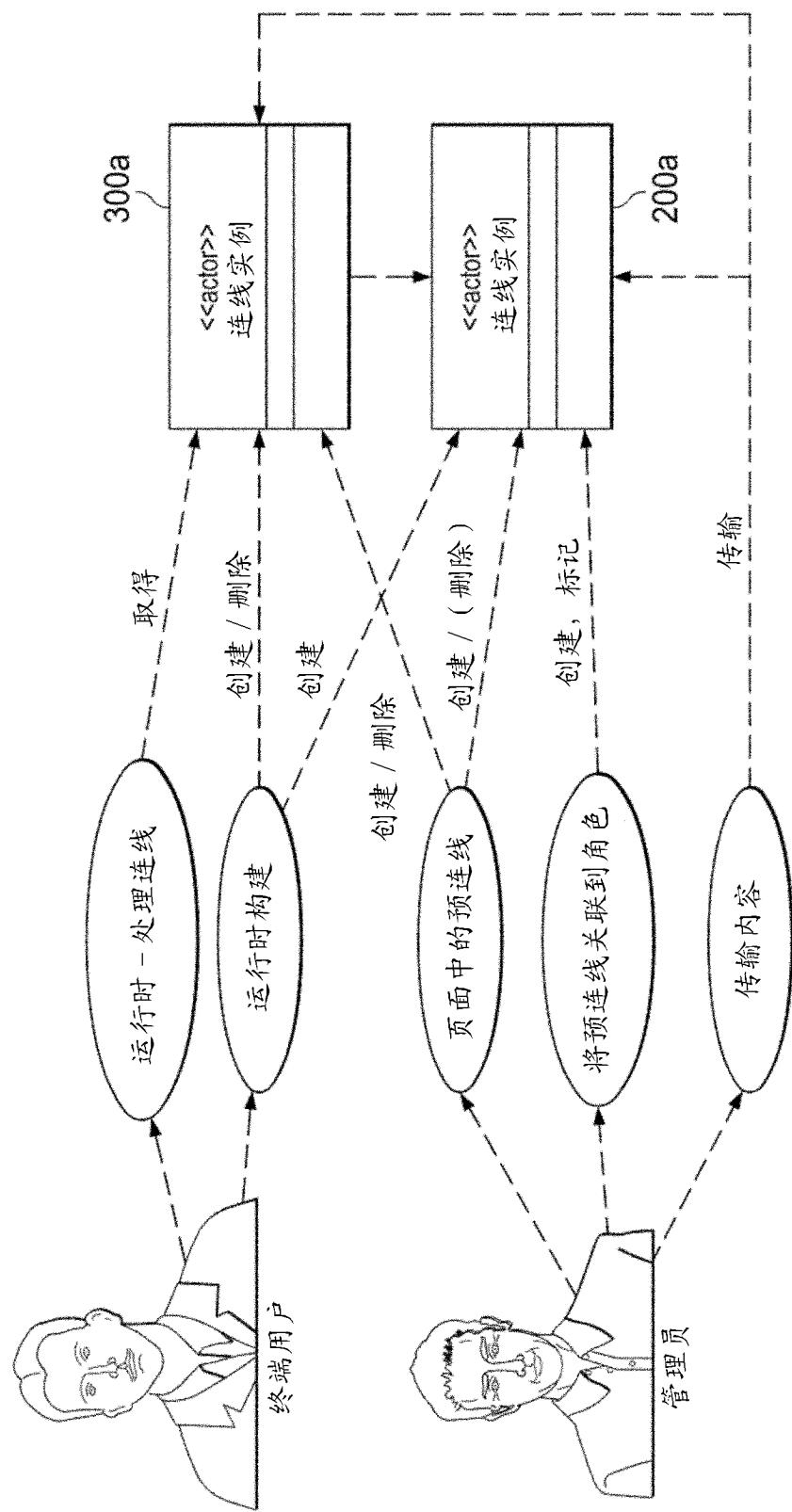


图 5

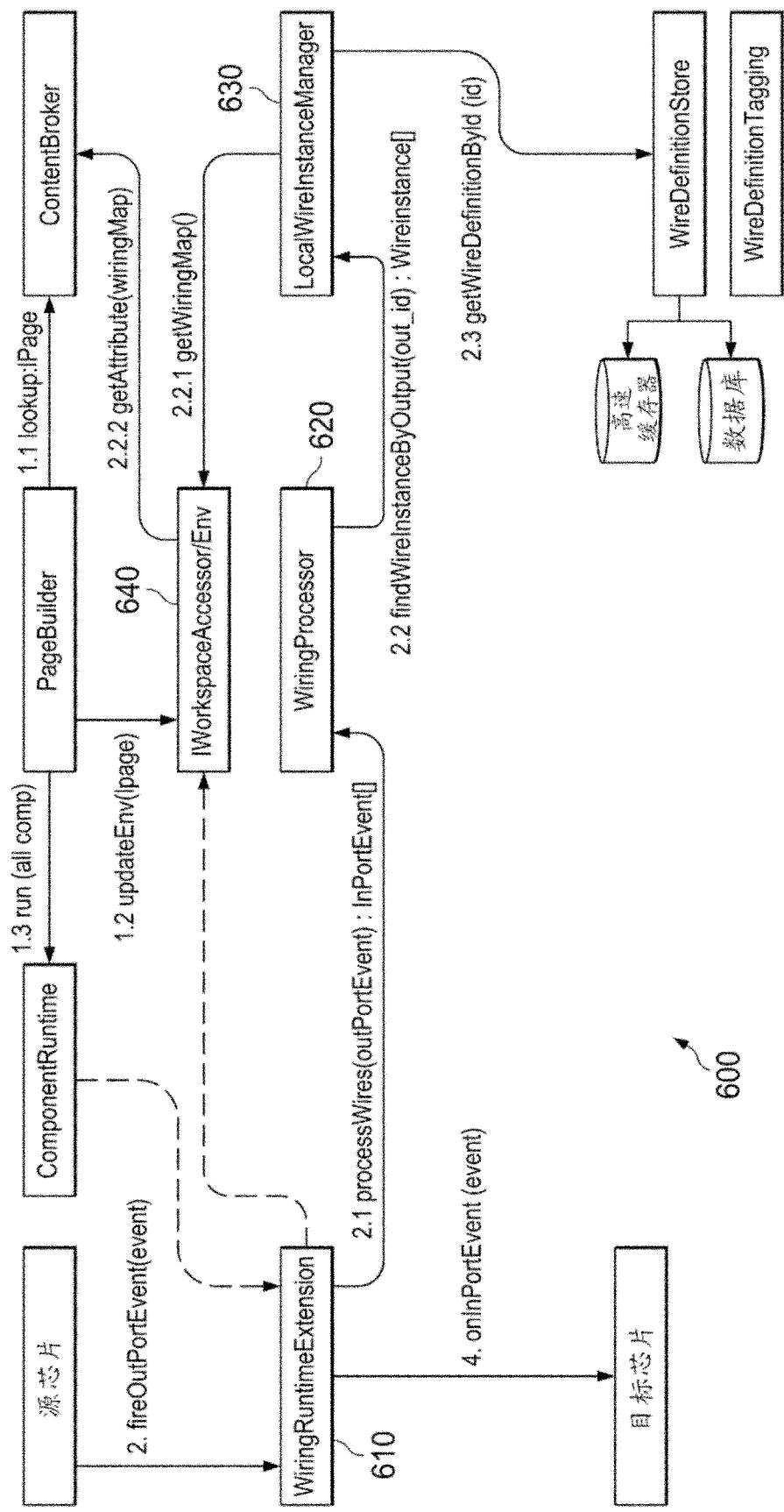


图 6A

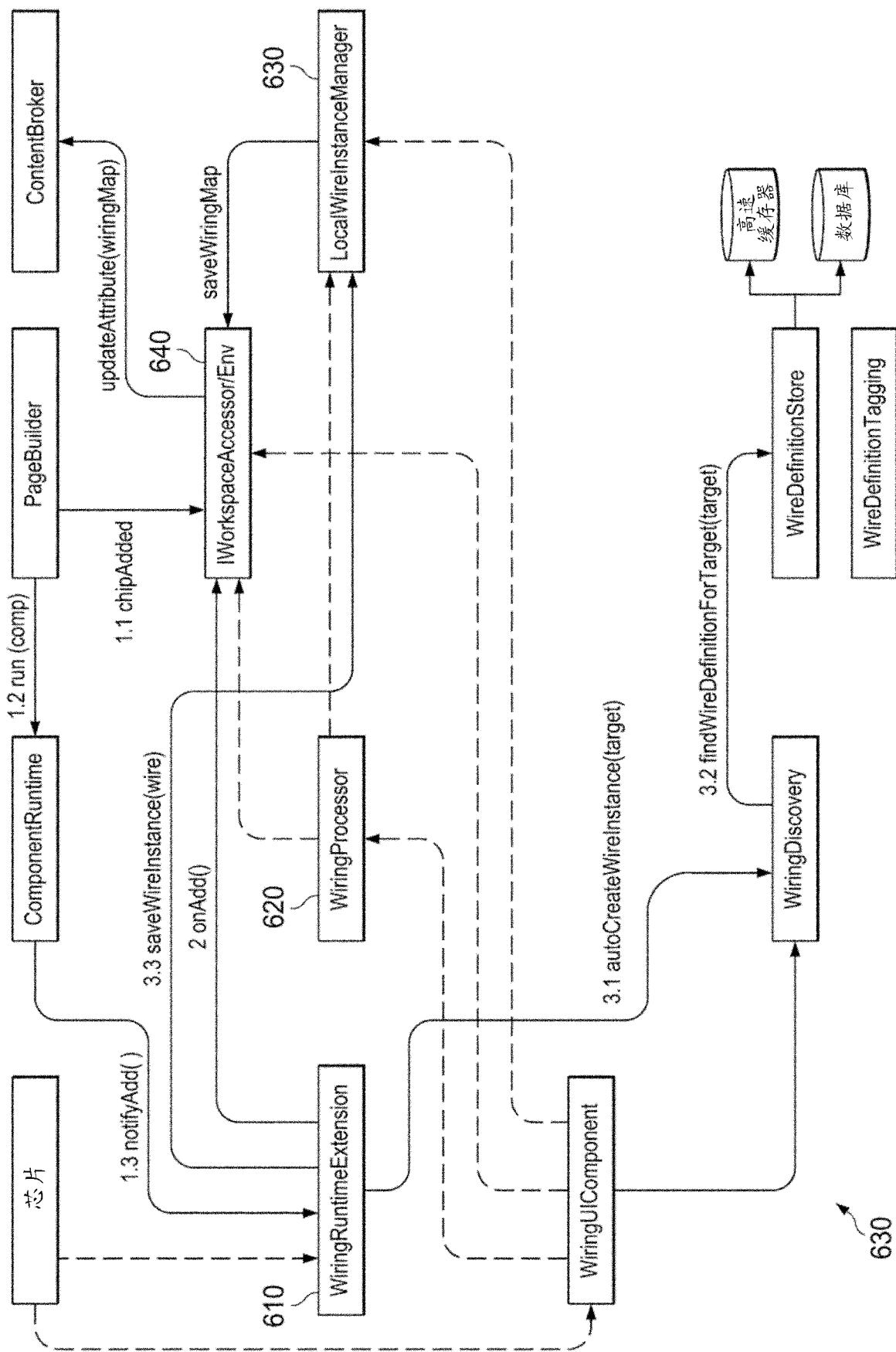


图 6B

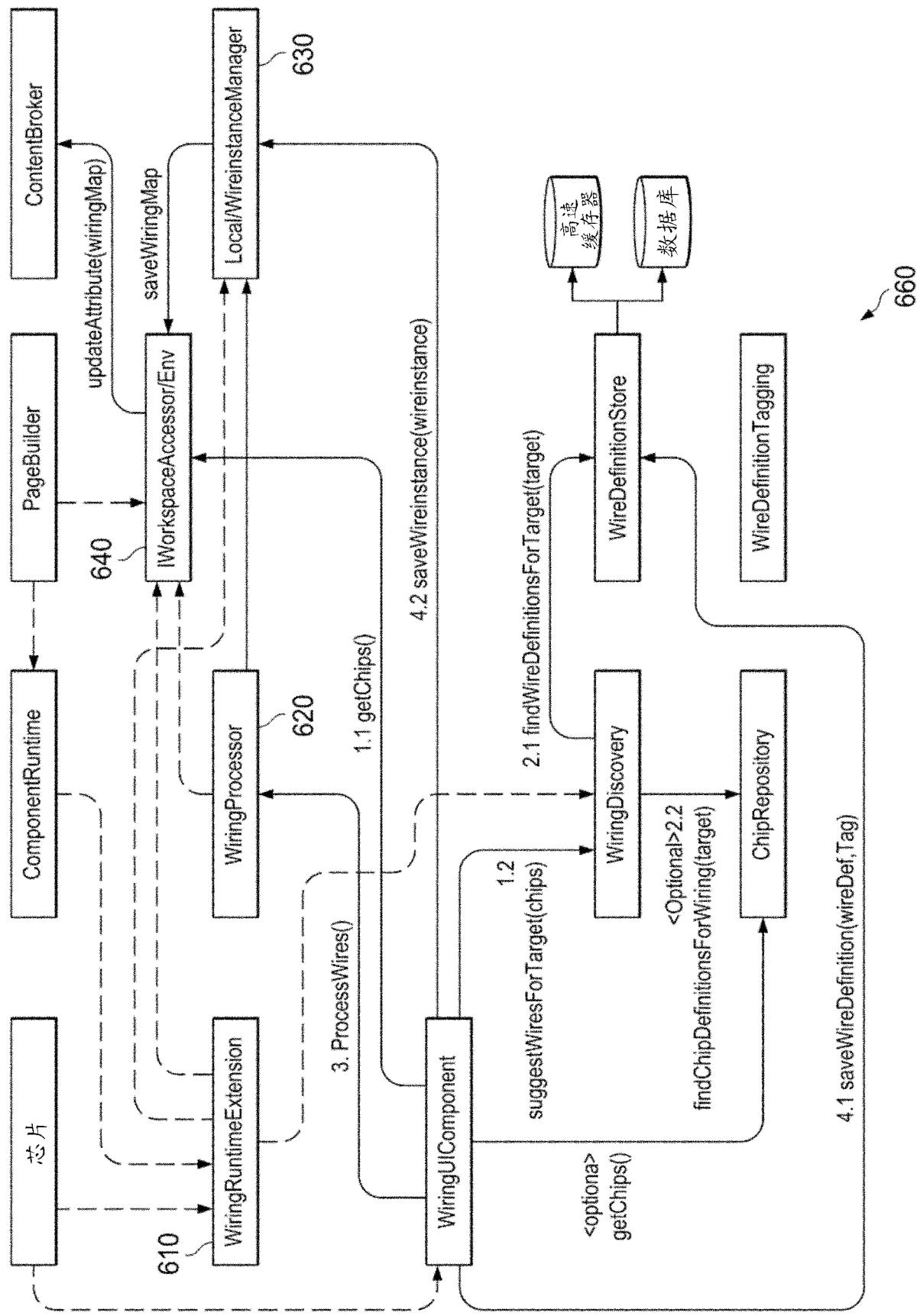


图 6C

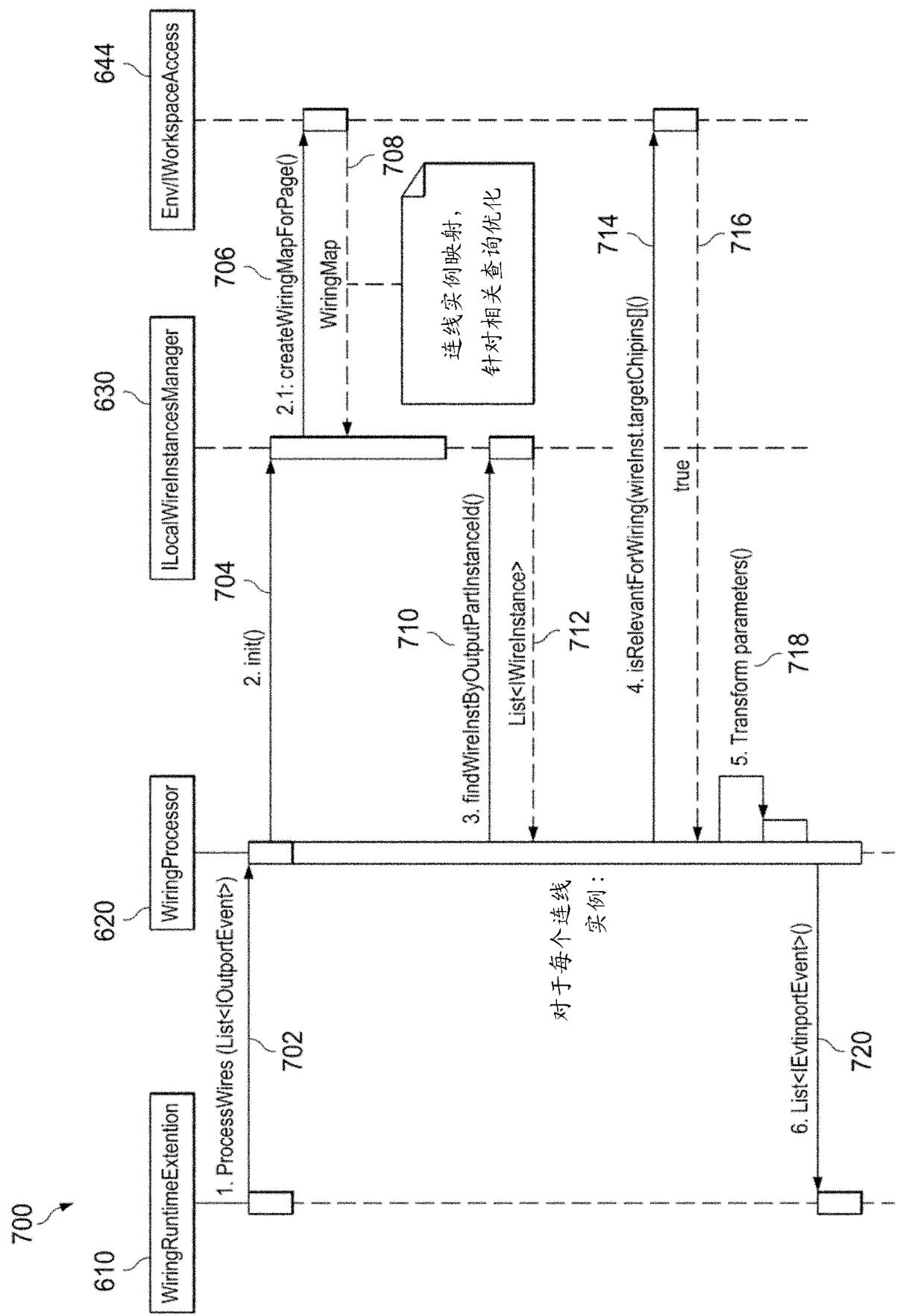


图 7