[19] 中华人民共和国国家知识产权局

[51] Int. Cl[7]
H04Q 3/00

[12] 发明专利说明书

[21] ZL 专利号 97197811.5

[43] 授权公告日 2003 年 3 月 26 日

[11] 授权公告号 CN 1104143C

[22] 申请日 1997.7.16 [21] 申请号 97197811.5

[30] 优先权

[32] 1996. 7.19 [33] GB [31] 9615150.1

[86] 国际申请 PCT/EP97/03799 1997.7.16

[87] 国际公布 WO98/04091 英 1998.1.29

[85] 进入国家阶段日期 1999.3.10

[71] 专利权人 艾利森电话股份有限公司

地址 瑞典斯德哥尔摩

[72] 发明人 B·耶勒马 R·佩伦 L·克劳汉
F·阿本

[56] 参考文献

EP0505092A2 1992.09.23 H04Q3/545

WO9406252 1994.03.17 H04Q3/545

审查员 毕艳红

[74] 专利代理机构 中国专利代理(香港)有限公司

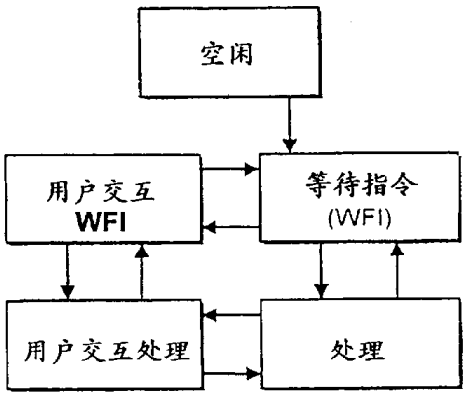
代理人 王 勇 李亚非

权利要求书 1 页 说明书 10 页 附图 2 页

[54] 发明名称 过程验证

[57] 摘要

一种验证过程的方法，其允许在单个事件的验证中涉及多个进程。 本方法包括定义用于描述实体的 FSM 之间的关系，选择满足特定标准的进程，并且在所有选择的进程中处理该事件。



ISSN1008-4274

1.在一个智能网络中验证一个事件的方法，该网络包括大量进程，该方法包括:

从所有进程中选择那些符合预定义标准的进程；并且

在所有被选择的进程中独立地处理该事件;

其特征在于，选择步骤包括定义表示进程的有限状态机之间的关系，这种定义包括定义进程间的容器接口。

2.根据权利要求1的方法，其中，预定义标准包括:

确定是否有有限状态机共享一个容器接口，并且

确定是否存在进程匹配当前事件的容器上下文。

3.根据权利要求1或2的方法，其中，该事件在第一个进程被接收，并且该方法包括在第一个进程中推迟处理该事件，直到选择步骤完成。

4.根据权利要求1所述的方法，其中，选择符合预定义标准的进程的步骤包括:

选择第一个进程集合，这些进程符合该标准集合;

在第一个进程集合中，预处理该事件；并且

根据预处理的结果选择第二个进程集合。

5.根据权利要求4的方法，其中，只有当第二个进程集合包括当前容器动作时，它们才被选择。

6.根据权利要求1所述的方法，其中，处理该事件的步骤包括将一个接收到的事件一直放在队列中，直到前一个被接收的事件已经被处理。

过程验证

技术领域

本发明是关于验证网络协议中的过程的一种方法。尤其，本发明涉及象智能网络应用协议（INAP）这样的网络协议，INAP 协议被用来支持功能集合 1（CS1），这个集合在欧洲电信标准 ETS 300 374 中进行了定义。

背景技术

在 INAP CS1 这样的协议中，存在大量的有效过程，或操作序列，它们可以用来实现特定的任务。然而，由于可能的过程数量很多，因此有效过程不是直接定义的，而是通过一组规则定义的。任何不违反这些规则的过程或操作序列都被认为是一个有效过程。

在以前的技术中，规则可以这样描述：为每个实体定义一个有限状态机（FSM），为两个实体间的边界定义接口。这样，FSM 就作为某个进程行为的模型而进行动作。FSM 由状态组成，状态之间可以互相连接。进程在任何时候都只能处于一个状态，不过可以作为一个事件的结果从一个状态转移到另一个相连的状态。在从一个状态到另一个状态的转换中，可以执行某些动作。这样的系统由 KAKUDA 等人在文章“用于特征交互及其进化的一种动态解决方法”有描述，该论文发表在通讯系统 III 的特征交互上，于第三次特征交互专题讨论会上提供（“A Dynamic Resolution Method for Feature Interactions and its Evolution”，Feature Interaction in Telecommunications Systems III，pages presented at the third Feature Interactions Work Shop）（FIW'9，KYOTO，JP，Oct. 11-13，1995）。

当使用 FSM 来验证网络协议中一个过程时，控制 FSM 的事件就是协议中定义的操作，或者是来自其他进程的事件，例如调用进程。

在象 INAP CS1 这样的网络协议中定义有效过程的规则，是单联结控制功能（Single association Control Function-SACF）规则和多联结控制功能（Multiple Association Control function-MACF）规则。SACF 规则用在是单一联结的地方，MACF 规则用在存在多个相关联结的地方，一个联结就是一个使用特定接口的信令信道，该接口允许

两个实体间进行通讯。

然而，仅仅采用一个 FSM 来描述所有 SACF 和 MACF 规则形式需要极其复杂的 FSM。结果是，很多过程，包括所有 MACF 过程，仍然用自然语言定义。没有机制来验证用于这类过程的规则，这类过程即那些不能用每个联结一个 FSM 来描述的过程。

发明内容

本发明是关于一种通过允许使用多个 FSM 来描述规则，从而验证一个过程的方法。特别地，多个各自运行一个特定 FSM 的进程被允许包含在一个单一事件的验证中。

事件的验证分为两个阶段。首先，通过估计相关过程是否满足特定标准来选择相关进程。在第二个处理阶段，所有已选的进程处理该事件。

这样做的优点在于，在任何被涉及的进程进行事件处理之前，所有这些进程都处于稳定的状态。

为了更好地理解这项发明，将参照附图以举例方式来说明。

附图说明

图 1 表示两个 FSM，它们共享一个容器接口。

图 2 表示一种情形，其中运行 A 的进程 Ax 包含运行 B 的两个进程，Bx 和 By。

图 3 表示一个 FSM，它包含了许多其他 FSM。

图 4 表示用于会话视图的 FSM。

图 5 表示用于调用视图的 FSM。

图 6 表示用于 Cp 或 Ap 的 FSM。

具体实施例

依照这项发明，过程的验证可以涉及多个进程，并且验证的第一阶段包括选择在验证中要涉及的进程。

进程选择所使用的标准，是基于 FSM 的特征和 FSM 之间的相互关系，这两者用来表示各个实体和接口。

首先，确定是否有 FSM 共享一个“容器接口”。图 1 表示的是共享一个容器接口的两个 FSM A 和 B。当两个 FSM 共享这种类型的接口时，就可以说一个包含于另一个，在后者的角度上说是后者包含前者。以图 1 为例，FSM A 包含 FSM B，FSM B 被 FSM A 包含。被第二个 FSM

包含的第一个 FSM，定义为也被任何包含第二个 FSM 的第三个 FSM 包含。也就是说，如果 A 被 B 包含，B 被 C 包含，那么定义 A 也被 C 包含。这可以说 A 被 C 间接包含。除非另外申明，否则这里使用的术语“包含”既指直接包含也指间接包含，这样也可以确定多于两个 FSM 间的关系。与其他类型的交互操作相比，通过容器接口上的联结而进行的进程间的交互操作是被不同地对待的。

如果两个 FSM 满足下列条件，那么其接口就可以归类为容器接口：

一个 FSM 仅仅被另一个 FSM 直接包含；

一个 FSM 不被它自己包含；

运行一个 FSM 的进程不与多于一个的特定进程有联结，这类特定进程运行的 FSM 直接包含第一个 FSM。

因此，回头看看图 1，运行 FSM B 的进程最多只能与一个运行 FSM A 的进程有一个联结。不过，运行 FSM A 的进程可以与多个运行 FSM B 的进程有联结（因此这些联结在运行 FSM A 的进程中相关）。A 或 B 也可以有其他（非容器类）接口。图 1 中，显示 FSM B 有一个另外的接口 I1。

包含关系的细节可以树的形式很容易地存储在一个系统中，以下称这棵树为 FSM 树。FSM 树的根是一个 FSM，该 FSM 不包含在其他任何 FSM 中。FSM 树的叶子是这样一些 FSM，它们分别都不包含任何其他的 FSM。

包含的概念也可以用于进程：如果由第一个进程运行的 FSM 包含在由第二个进程运行的 FSM 中，则前一个进程包含于后一个进程。另外，这两个进程必须相关连，直接或间接地通过一个进程相关连，这两个进程（直接或间接地）通过这一进程相连接的。例如，图 2 代表了一种情形，在该情形下运行 A 的进程 Ax 包含运行 B 的两个进程，Bx 和 By。

如同 FSM，这些包含关系的细节可以树的形式很容易地存储在一个系统中，以下称这棵树为进程树。当扫描进程树时，可以使用一个单独的数据结构，也可以使用进程间的连接。在某一进程树中的进程集合与该进程树，一起构成了选择进程的一个基础。

在定义进程的选择标准中所使用的第二个描述特征是“容器上下

文”，该“容器上下文”是一个预定义值的集合，用来标识一个需要验证的事件的各异的、可能存在的上下文。因此，使用相同联结的不同事件，可以拥有不同的上下文。例如，在 CS1 中，一些操作处于在该调用中涉及的某一方的上下文中。

进程也可以被分配一个容器上下文。在这种情况下，该容器上下文标识一个事件和一个特定进程的关系。

这样，当一个事件（“当前事件”）将要被验证时，识别匹配于当前事件的容器上下文的进程集合成为可能。

容器上下文的分配，是通过分配每个 FSM 一个容器上下文的集合实现的：Fx = { CC1，CC2，. . . ，CCn } 。如果一个容器上下文已经被分配给一个 FSM 所包含的或包含在该 FSM 中的 FSM，则这个容器上下文不能被分配给该 FSM。当容器上下文被分配给 FSM 时，系统将进行以上的检查。一个事件可以被分配一个单一的容器上下文。这个容器上下文也就被分配给当前事件，并且被称为“当前容器上下文”。考虑可能到达某一接口的事件集合，用于该事件集合的所有可能的容器上下文必须分配给一个 FSM，这个 FSM 是包含或者被包含于共享该界面的一个 FSM。

在进程树中的每个进程都从 Fx 中被分配了一个容器上下文，这样运行同一个 FSM 的不同进程，以及这棵进程树的局部，都被分配了不同的容器上下文。

例如，在一个协议中，存在 30 个可能的 leg（引线）标识，它们每个都可以用作为一个容器上下文。一些操作用于整个调用（例如释放调用（ReleaseCall）），并且它们将该调用作为容器上下文。可以具有一个 leg 标识作为参数的操作，被看作 30 个独立的事件（一个事件对应一个可能的 leg 标识）。这就要求对该操作进行分析，以将操作映射到 30 个可能事件中的一个（例如，该 leg 标识参数，或者缺省 leg 标识值，确定从操作到事件的映射）。

在定义进程的选择标准中所使用的第三个描述特征是“容器动作”，“容器动作”是一个特殊的动作集合，可以以一个 FSM 中的状态转换指定。这些动作可以通过其他进程影响当前事件的处理。例如，它们可以导致另一个进程不处理这个事件，尽管过去该进程被指定要进行这种处理。

每个系统将拥有大量已定义的容器动作。由于容器动作可能影响其他进程，当几个进程在执行相互影响的容器动作时，可能会产生冲突。系统必须清楚地指定这些容器动作间的相互作用。例如，拥有较高优先权的容器动作将排斥所有较低优先权的容器动作（针对当前事件）。在这里，定义这些相互作用所用的原理就不进一步描述了；假设：对于一个容器动作集合，系统可以识别那个将被允许的容器动作子集。这个子集以下就称之为当前“容器动作”。

为了使系统能够协调来自于不同进程的不同容器动作，在处理一个事件前需要两个步骤。

首先，假如存在可能处理这个事件的一个进程集合，这样的容器动作被收集起来，一旦这些进程处理该事件，这些进程就将执行这些容器动作。这就要求通过所涉及的进程对一个事件进行预处理：确定结果状态转换（因为这些状态确定了将要执行的动作），但是并不执行。

然后，基于搜集到的容器动作，以及系统中定义的它们之间的优先权，系统将创建当前容器动作。

当一个事件将要被一个进程集合处理时，它们的容器动作可能不得不进行协调，以避免相同的动作被执行两次。

如上所描述的，验证的第一个阶段包括应用基于上面描述的特征的标准，选择在本验证中将要涉及的进程。

即使存在这种进程，该进程最初通过一个联结接收到该事件，这个事件也不会立即被该进程处理。尽管接收进程总是被选作选择阶段的一个结果，也只有当它已被选中时才可以处理这个事件。

作为选择阶段的一部分，将确定是否存在一个当前容器上下文。如果没有当前容器上下文，则只有接收到这个事件的进程才被选择。如果存在当前容器上下文，则所有满足以下两个标准的进程都将被选择。首先，这个进程必须持有，或包含一个进程，后者持有或者包含于另一个持有这个容器上下文的进程中，该进程可以是、包含、或者被包含于接收该事件的进程。其次，这个进程除了执行当前容器动作之一外，绝不执行其他的容器动作。

选择阶段以检测接收该事件的进程开始。从这儿，这棵进程树被扫描到根和叶子。对每个被扫描到的进程，检查它是否与容器上下文

匹配。每个匹配的进程被加入该选择。这里使用了几种优化方法。首先，如果一个匹配的进程包含那个接收该事件的进程，则将不存在其他的匹配进程。这是由于如果一个 FSM 已经包含或者被包含于一个已经被分配了容器上下文的 FSM，则该容器上下文不能被分配给这个 FSM。其次，如果一个匹配进程已经被找到，则不存在其他运行相同的 FSM 的匹配进程。

这样将给出至少一个匹配进程，并且，对于每一个找到的匹配进程，进程树将又一次被扫描到根和叶子。现在，每个匹配的进程都被加入选择，如果之前它没有被选择的话。

然后，如前面讨论的那样，这个事件被所有选择了的进程预处理。这将产生当前容器动作，当前容器动作又在该选择中确定最终的进程集合。

为了使选择进程正确工作，在第一个事件被验证前，这棵完全的进程树必须看上去是可用的。这里的意思是，对于每个 FSM，在进程树中，对于分配给它的任何容器上下文都必须存在一个进程（即使它处于空闲状态）。由此引起的系统开销对任何系统都是不可接受的。下面的方法可以用来创建所需的进程。

如果，在进程选择中，到达了树端（该进程因此被称为：最终进程），但是由最终进程运行的 FSM（因此称为最终 FSM）不是相应 FSM 树的根或叶子，那么：

-当向根扫描时，如果一个直接包含了最终进程的 FSM 是，或者包含于一个 FSM，后者为当前事件指定了从空闲状态开始的、具有“无动作”之外的一个动作的一个状态转移，则将为直接包含了最终进程的该 FSM 创建一个进程。

-当向叶子扫描时，将为每个这样的 FSM 创建一个或多个进程，这类 FSM 直接包含于最终 FSM，并且是或包含一个 FSM，后者为当前事件指定了从空闲状态开始的、具有“无动作”之外的一个动作的一个状态转移。如果一个 FSM 的 Fx 将当前容器上下文作为一个成员，或者它的 Fx 为空，则将为这个 FSM 创建一个进程。否则，将为这个 FSM 的 Fx 中每个成员创建一个进程。然后，各个进程间建立一个联接，接着继续搜索。

系统必须知道，对于每个 FSM，如果这个 FSM 在空闲状态时接收

到事件时，对于哪些事件它需要指定一个不是“无动作”的动作。

当进程处于空闲状态，且不包含任何其他进程时，进程可以被移走，它在容器接口上的任何联结都可以被释放。当一个进程不是处于空闲状态，但不再具有任何联结，例如进程挂起时，就出现一种情形。系统可以检测这种情形，因为该进程在此时是根，也是唯一的叶子。在这种情况下，系统产生一个特殊事件，例如“孤儿化”（orphaned），此时该进程将进入空闲状态（如果没有，则被认为是一个错误）。

在选择阶段完成，并且选择了一组进程后，每个被选择的进程都将独立于其他被选择的进程对该事件进行处理。此后，这个事件就可以被验证了。

由于事件可以到达不同的联结，并且只有在前一个事件被处理后，后续事件才可能被处理，因此必须排队。如果已经存在排队的事件，或者已经存在一个当前事件，则一个事件将被排队。事件排队等候接收到该事件的进程的根进程。依赖存在的不同优先权等级的数目，可以使用不同的队列。通过容器接口收到的事件拥有比通过其他类型接口收到的事件更高的优先权。这样，在从其他队列中提取事件前，用于这些事件的队列必须为空。这给予所有被涉及的进程一个机会，以在要验证的下一个事件接收到之前进行相互更新。

现在，本发明将通过对一个特定例子的应用，进行更详细的描述，这个例子取自用于 IN 的 Ericsson CS1+ INAP 协议，该协议是通过大量的 FSM 进行描述的。这些 FSM 中的一个是 SSF-FSM，如图 3 所示，它由多个 FSM 组成。

这样，会话视图 FSM（Session-view）包含调用视图 (Call view) FSM，后者又包含 Cp 和 Ap FSM。这些 Cp 和 Ap FSM 共享一个接口，但不是容器接口。

调用视图 FSM 与该调用以及 SCF（服务控制功能）有外部接口。用 SACF 规则和 MACF 规则进行校验的事件，将到达的是这些接口。

以下的容器动作将被识别（按优先权顺序）：

buffer Event（缓冲区事件）：该事件被缓冲（在这里对缓冲机制本身不进行描述）。不跟有出错处理程序。

reject Event（拒绝事件）：该事件被拒绝。跟有一个出错处理程序。

图 4 描绘了用于会话视图的 FSM，该 FSM 指定用于 INAP 的 MACF 规则。这里存在一个 MACF 规则：连接到几个 SCF 的联结可以同时存在，但是只能有一个联结控制该调用。

这里不存在将会话视图作为容器上下文的操作。由于会话视图包括所有其他的 FSM，因此会话视图总是涉及在每个事件的验证中。

只有在用于会话视图的 FSM 处于空闲或监视关系状态时，才允许一个与该 SCF 相连接的新的联结。因此，当这个用于会话视图的 FSM 处于控制关系状态时，InitialDP 操作（该操作启动一个与 SCF 的新的联结）被拒绝。这样，该操作被忽略。

图 5 描绘了用于调用视图的 FSM，该 FSM 指定用于与 SCF 的联结以及调用的 SACF 规则。每个与 SCF 的联结都存在一个调用视图进程。该 FSM 从一个单一 SCF 进程的角度描绘了一个调用的结构。

这里存在一些将调用视图作为容器上下文的操作，例如，释放调用（ReleaseCall）。

状态：相应于 BCSM 中的状态的设置（Set Up），挂起（Pending）以及活跃（Active）状态。当处于设置状态，SCF 仍然能够使用通常的 CCF 程序（CollectInformation 操作）从该用户收集数据。当处于挂起状态，上面的情况不再可能（CollectInformation 被拒绝）。当处于活跃状态，SCF 不能发送该调用（连接被拒绝）。该调用只能通过这个调用的一个确定的事件返回挂起状态。

当处于代理状态（Surrogate），启动该调用的一方撤离这个调用。此后，该 SCF 就不可能发送这个调用（来自于该调用的事件，此事件致使 FSM 的状态从活跃转向挂起，现在将引发一个到没有动作的同一状态的状态转移）。当处于监控关系状态（Monitoring Relation），SCF 不再能够控制该调用，但是可以监控在该调用中发生的事件（所有的控制操作都被拒绝）。应该注意的是，当进入监控关系状态时，FSM 产生一个事件，并且发送到会话视图（为了引发一个到监控关系状态的状态转移）。

图 6 描绘了用于 Cp 或 Ap 的 FSM。Cp 或 Ap 拥有同样的状态。在一个调用视图中，只能有一个 Cp 视图，但是可以有几个 Ap 视图。对于那些涉及了 Cp 或 Ap 的操作，Cp 或 Ap 指定用于 SCF 的联结的 SACF 规则和该调用。

该 FSM 描述了 SSF 和 SCF 间的同步过程，以及用于与调用方交互的过程（而不是使用通常的 CCF 过程）。

这里不存在将 Cp 或 Ap 作为容器上下文的操作。视图中哪个视图被涉及，是由包含在 Cp 或 Ap 中的 FSM，leg 决定的。

如图 3 所示，在 Cp 和 Ap 之间存在一个接口，这个接口被用来将 leg 从一个视图移到另一个视图（实际上，这意味着用于该 leg 的被包括的进程被释放了，并且一个新的进程在另一边启动了）。

在该 FSM 的 WFI 状态，Cp 或 Ap 进程与 SCF 同步，尽管如果用户挂起（来自 CLSM 进程的事件进入），该同步可能被打断。应该注意的是，不同的 Ap 进程因此不能一起同步，并且也不能与 Cp 进程同步；该同步不用于整个调用。

当用户交互正在进行（FSM 处于用户交互 WFI，或用户交互处理状态），除了切断前向连接，所有的操作都将被 Cp 缓冲。当 FSM 转换回到 WFI 或处理，缓冲区被释放。Ap 只缓冲释放调用（Release Call），并且只在本地缓冲，也就是不影响其他 FSM。因此，Ap 不因为这而使用一个容器动作。

伴随处于状态处理中的 FSM，SSF 和 SCF 并行地运行，也就是它们不同步。

用于 Cp 和 Ap 中的 leg 的 FSM 是相同的，包括所有的状态转移，动作等。这些 FSM 只有空闲和活跃状态。这些 FSM 用作为操作的一个容器上下文。由于在 Cp 和 Ap 中的 leg 是相同的，并且由于每个只有一个状态（除了空闲），一个 leg 进程能够从 Ap “移动” 到 Cp，反之也可以从 Cp 转移到 Ap，这是通过将当前存在的进程从活跃状态转移到空闲状态，并且将一个新进程（在目标视图中）从空闲状态转移到活跃状态而实现的。

这样，leg 的定位将决定是否 Cp 视图或 Ap 视图将被涉及。“leg” 的概念描述了一种网络资源，该资源被分配给一个调用里的一个单一方。在 Ap 中只能有一个 leg，但在 Cp 中可以有好几个。

基于此，以确定这样的标准是可能的，这些标准被用于选择那些在验证期间处理事件的进程。

应该注意的是，容器接口的概念通常被用来对于外部触发器描述一个系统，或系统的一部分的行为（也就是来自于其他系统或同一系

统的其他部分）。

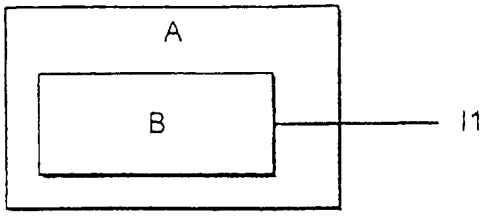


图 1

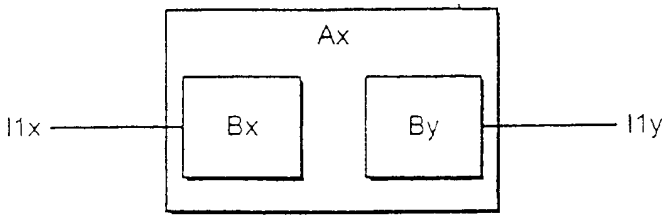


图 2

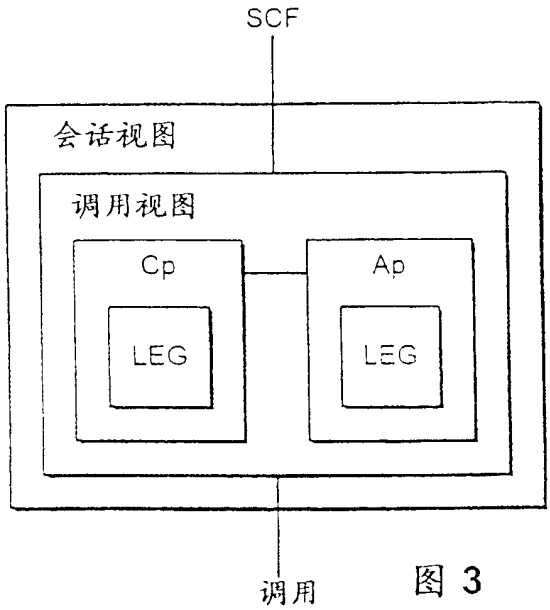


图 3

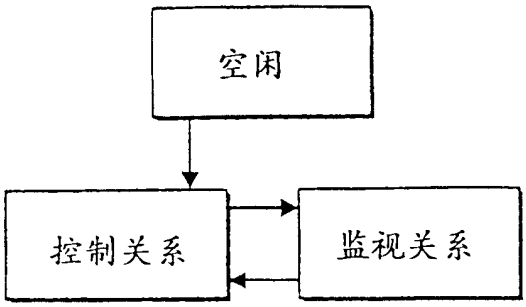


图 4

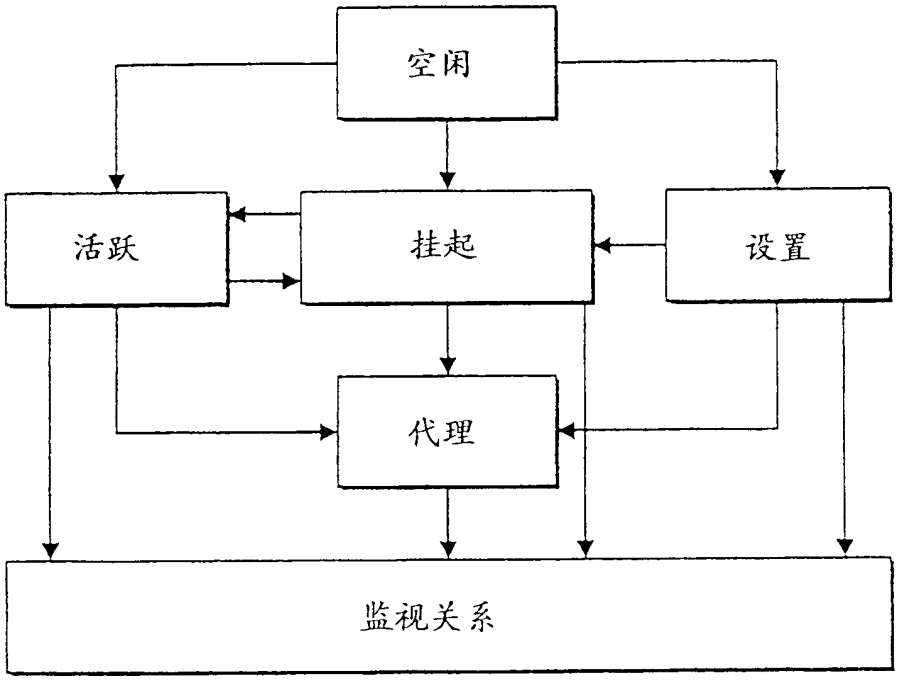


图 5

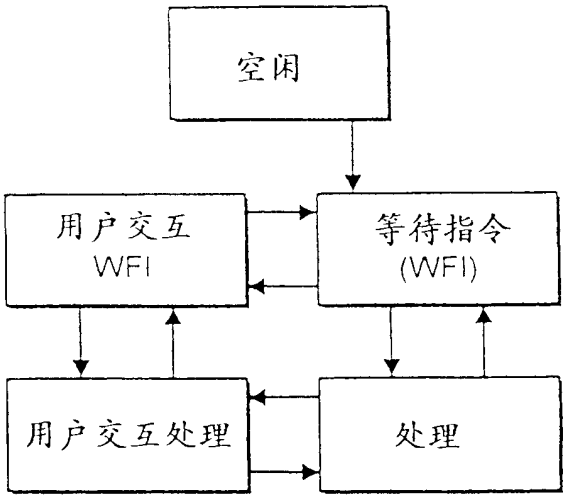


图 6