(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 102736919 A
(43)申请公布日 2012.10.17

(21)申请号 201210100410.4

(22)申请日 2006.05.16

(30)优先权数据
60/681,420 2005.05.16 US

(62)分案原申请数据
200680025983.8 2006.05.16

(71)申请人 夏寿民
地址 中国台湾台北市信义吴兴街284巷59弄12号7楼

(72)发明人 夏寿民

(74)专利代理机构 北京清亦华知识产权代理事务所(普通合伙) 11201
代理人 黄德海

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

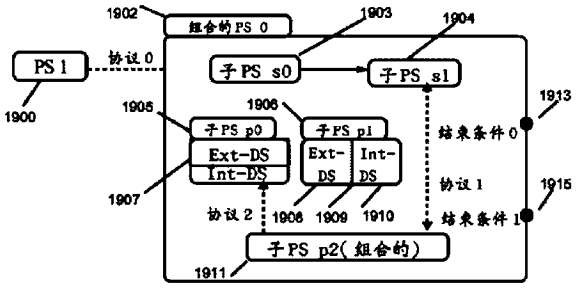
权利要求书 3 页 说明书 39 页 附图 23 页

(54)发明名称

描述及开发动态行为应用系统的方法与系统

(57)摘要

本发明提供了对动态应用系统行为进行建模的一套系统、过程和过程的运用方法以及开发这种动态行为模型的方法。本发明提供了一个动态系统(即Dynamic System,简称DS)以及相应的操作方法来对普遍的复杂而且动态的行为进行建模。为了对一个分布式环境中的应用系统进行建模,本发明采用了一个外部的DS(即External DS)来对这个系统的外部行为进行建模,而采用一个自主系统(即Proactive System)来对这个应用系统的反应式行为进行建模,以及一个内部DS(Internal DS)来对这个应用系统的内部主动式行为进行建模。本发明同时还提供了表示DS以及它的组件的图形化语言。最后本发明描述了一个系统化的方法,从外部行为推导出反应式行为或者反过来,从反应式行为推导出外部行为。



CN 102736919 A

1. 一种包括至少一个用于设计应用软件系统行为的动态系统类型的计算机系统，其特征在于，所述计算机系统包括：

一个设计工具，所述设计工具用于生成所述动态系统类型的设计文档以便代码生成器能根据所述动态系统类型的设计文档生成表示所述动态系统类型的二进制代码，其中所述二进制代码能够由执行平台装载到内存中而为所述组合动态系统类型创建组合动态系统实例。

2. 根据权利要求 1 所述的计算机系统，其特征在于，所述动态系统类型包括一个第一个简单动态系统类型或一个第一个组合动态系统类型。

3. 根据权利要求 2 所述的计算机系统，其特征在于，所述第一个组合动态系统类型包含多个阶段机类型，一个结束情况集或一个结束函数，其中所述一个结束情况集至少包含一个结束情况，所述结束情况包含一对结束条件和结束代码，其中结束条件描述在何种情况下所述第一个组合动态系统类型的实例会结束，结束代码用来标识对应的结束条件。

4. 根据权利要求 3 所述的计算机系统，其特征在于，所述一个结束函数会返回结束代码以标明在何种情况下所述第一个组合动态系统类型的实例会结束。

5. 根据权利要求 3 所述的计算机系统，其特征在于，每个所述阶段机类型包括一个或多个阶段类型，其中每个阶段类型包括一个子动态系统类型以及至少一个迁移，每个迁移有一个迁移条件和一个目标阶段，每个所述子动态系统类型包括一个第二个简单动态系统类型或一个第二个组合动态系统类型。

6. 根据权利要求 3 所述的计算机系统，其特征在于，至少一个所述至少一个的阶段机类型进一步包括：静态实例数，用于描述出该所述至少一个的阶段机类型的静态实例的数量。

7. 根据权利要求 3 所述的计算机系统，其特征是，至少一个所述多个阶段机类型仅具有一个所述至少一个的阶段类型。

8. 根据权利要求 2 所述的计算机系统，其特征在于，所述的简单系统类型进一步包括：

第一个所述子系统类型，用于描述出待发生的外部事件；以及

第二个所述子系统类型，用于描述出为响应发生的所述外部事件而执行的内部动作。

9. 根据权利要求 3 所述的计算机系统，其特征在于，至少一个所述至少一个的组合系统类型是以一种程序语言、一种数据描述语言、一种基于 XML 的语言和一种图形语言之一来描述。

10. 根据权利要求 5 所述的计算机系统，其特征在于，进一步包括：

至少一个组合系统实例，包括：

一个对所述第一个组合系统类型的引用；

一个结果用来记录所述至少一个组合系统实例结束时的情况；以及

至少一个阶段机；

其中所述至少一个阶段机包括：对所述至少一个的阶段机类型的一个引用、一个当前阶段和至少一个阶段；

其中所述至少一个阶段包括：对所述至少一个的阶段类型的一个引用和一个子系统实例；

其中所述子系统实例是所述第二个组合系统类型的实例或所述第二个简单系统类型

的实例中的至少一个；

其中所述第二个简单系统类型的实例包括：对所述第二个简单系统类型的一个引用和一个对执行所述第二个简单系统类型行为的成果进行记录的结果。

11. 根据权利要求 10 所述的计算机系统，其特征在于，所述计算机系统进一步包括所述执行平台。

12. 根据权利要求 5 所述的计算机系统，其特征在于，所述计算机系统进一步包括所述代码生成器。

13. 根据权利要求 9 所述的计算机系统，其特征在于，所述至少一种图形语言包括：

至少一个包含框，用于可视化地表示所述应用系统的一部分，至少一个所述至少一个包含框包括多个子包含框，

至少一个带有标识符的双向链接组件用于表示所述多个子包含框中的两个之间的一组交流，而所述标识符标明了所述的一组交流必需遵循的一个协议规范，和

多个结束点组件；

其中所述多个结束点组件中至少有一个组件具有一个用于描述出该包含框在何种条件下会结束的结束条件组件。

14. 根据权利要求 11 所述的计算机系统，其特征在于，所述结束条件组件可视化地以一段文本，一个逻辑表达式和一个数学公式之一来表示。

15. 根据权利要求 13 所述的计算机系统，其特征在于，所述多个结束点组件中至少一个结束点组件可视化地以一个符号或一个框子来表示。

16. 根据权利要求 13 所述的计算机系统，其特征在于，所述框子可视化地以一个多边形，一个圆形和一个椭圆形之一来表示。

17. 根据权利要求 11 所述的计算机系统，其特征在于，至少一个所述至少一个包含框进一步包括至少一个迁移箭头，用于连接所述多个子包含框中的一个子包含框和所述多个子包含框中的另外一个子包含框。

18. 根据权利要求 11 所述的计算机系统，其特征在于，至少一个所述至少一个子包含框包括多个子子包含框，以及至少有一个迁移箭头连接所述多个子子包含框中的一个子子包含框与所述多个子子包含框中的另一个子子包含框，其中所述多个子子包含框中至少有一个子子包含框不通过所述的迁移箭头与任何其它所述多个子子包含框连接。

19. 根据权利要求 11 所述的计算机系统，其特征在于，至少一个所述至少一个包含框进一步包括至少一个用于指向动态组件的生成箭头。

20. 一种包括至少一个用于设计应用软件系统行为的动态系统类型的计算机系统，其特征在于，所述计算机系统包括：

一个代码生成器，所述代码生成器用于根据由设计工具生成的动态系统类型的设计而生成表示所述动态系统类型的二进制代码，其中所述二进制代码能够由执行平台装载到内存中以被执行。

21. 一种包括至少一个用于设计应用软件系统行为的动态系统类型的计算机系统，其特征在于，所述计算机系统包括：

一个执行平台，所述执行平台将用于表示动态系统类型的二进制代码装载到内存中以生成所述动态系统类型的实例，其中所述二进制代码是由代码生成器根据由设计工具所生

成的动态系统类型的设计而生成的。

22. 根据权利要求 21 所述的计算机系统，其特征在于，所述动态系统类型包括一个第一个简单动态系统类型或一个第一个组合动态系统类型。

23. 根据权利要求 22 所述的计算机系统，其特征在于，所述第一个组合动态系统类型包含多个阶段机类型，一个结束情况集或一个结束函数，其中所述一个结束情况集至少包含一个结束情况，所述结束情况包含一对结束条件和结束代码，其中结束条件描述在何种情况下所述第一个组合动态系统类型的实例会结束，结束代码用来标识对应的结束条件。

24. 根据权利要求 23 所述的计算机系统，其特征在于，所述一个结束函数会返回结束代码以标明在何种情况下所述第一个组合动态系统类型的实例会结束。

25. 根据权利要求 23 所述的计算机系统，其特征在于，每个所述阶段机类型包括一个或多个阶段类型，其中每个阶段类型包括一个子动态系统类型以及至少一个迁移，每个迁移有一个迁移条件和一个目标阶段，每个所述子动态系统类型包括一个第二个简单动态系统类型或一个第二个组合动态系统类型。

26. 根据权利要求 23 所述的计算机系统，其特征在于，至少一个所述至少一个的阶段机类型进一步包括 ：静态实例数，用于描述出该所述至少一个的阶段机类型的静态实例的数量。

27. 根据权利要求 23 所述的计算机系统，其特征是，至少一个所述多个阶段机类型仅具有一个所述至少一个的阶段类型。

28. 根据权利要求 22 所述的计算机系统，其特征在于，所述的简单系统类型进一步包括 ：

第一个所述子系统类型，用于描述出待发生的外部事件 ；以及

第二个所述子系统类型，用于描述出为响应发生的所述外部事件而执行的内部动作。

29. 根据权利要求 23 所述的计算机系统，其特征在于，至少一个所述至少一个的组合系统类型是以一种程序语言、一种数据描述语言、一种基于 XML 的语言和一种图形语言之一来描述。

30. 根据权利要求 23 所述的计算机系统，其特征在于，进一步包括 ：

至少一个组合系统实例，包括 ：

一个对所述第一个组合系统类型的引用 ；

一个结果用来记录所述至少一个组合系统实例结束时的情况 ；以及

至少一个阶段机 ；

其中所述至少一个阶段机包括 ：对所述至少一个的阶段机类型的一个引用、一个当前阶段和至少一个阶段 ；

其中所述至少一个阶段包括 ：对所述至少一个的阶段类型的一个引用和一个子系统实例 ；

其中所述子系统实例是所述第二个组合系统类型的实例或所述第二个简单系统类型的实例中的至少一个 ；

其中所述第二个简单系统类型的实例包括 ：对所述第二个简单系统类型的一个引用和一个对执行所述第二个简单系统类型行为的成果进行记录的结果。

描述及开发动态行为应用系统的方法与系统

[0001] 本申请是申请号为 200680025983.8、发明名称为“描述及开发动态行为应用系统的方法与系统”的中国发明专利申请的分案申请。

[0002] 对相关应用的交叉引用

[0003] 统一建模语言 (UML) :总体结构 (Superstructure),版本 2.0,2005 年 8 月

[0004] 流程定义接口 -XML 流程定义语言 (XPDL),版本 1.0,2002 年 10 月 25 日

[0005] 业务流程建模语言 (BPMN),版本 1.0,2004 年 5 月 3 日

[0006] 面向 Web 服务的业务流程执行语言 (WS-BPEL),版本 1.1,2003 年 5 月 5 日

[0007] Web 服务编排描述语言 (WS-CDL),版本 1.0,2005 年 11 月 9 日

[0008] 描述反应式系统的方法和设备 (动态状态机 Dynamic State Machine),2003 年 4 月 30 日

[0009] 优先权申明

[0010] 本申请基于 35U.S.C.119(e) 要求,序列号为 60/681,420,申请日为 May 16,2005,名称为″ Method and apparatus for modeling dynamic system behavior″ 的美国临时专利申请的优先权,在此为了各种需要,而对以上全文作引用。

技术领域

[0011] 本发明与流程图,活动图,业务流程,流程建模,工作流,状态机,协议规程以及应用软件设计与实现有关。

背景技术

[0012] 按照传统的做法,可以有两种不同途径来对一个系统的行为进行建模。一种途径是关注系统的内部主动式行为,而另一种途径则关注系统对外部事件的反应式行为。这两种途径表现为 UML 中两种不同风格的图画 :关注主动式行为的“活动图 (Activity Diagram)”和关注反应式行为的“状态图 (Statechart)”。我们把前一种途径称为“主动式行为建模”而把后一种途径称为“反应式行为建模”。

[0013] 主动式行为建模语言包括流程图,业务流程建模符号 (BPMN),XML 流程定义语言 (XPDL),业务流程执行语言 (BPEL) 和 UML 活动图。BPMN 和 UML 活动图都在一个图画中使用节点来表示要执行的活动,而使用连接两个节点的链接来表示这些活动的顺序。在另一方面,XPDL 和 BPEL 用一个 XML 句法替代一个图形来定义这些活动节点和它们之间的链接。

[0014] 虽然这些主动式行为建模语言在描述一个系统的内部主动式行为时有用,但它们在描述对随时可以发生的外部事件做出的反应式行为时有所不足。一般的解决办法是定义一些特殊的事件检测活动,比如 BPMN 和 BPEL 中的“接收 (receive)”活动,来检测这些可能发生的外部事件。由于这些外部事件随时都可能发生,因此不得不在一幅图的各个地方插入各种事件检测活动。然而,和一个状态机不同的是,这些主动式行为的建模图画没有一个能把事件检测活动集中在一起的“状态”机制。结果导致这些事件检测活动散布在图画中的各个地方,从而使得在运行时刻要想知道某个时刻一个应用系统在等待哪些外部事件变

得非常困难。

[0015] 这种方法的主要缺点在于如果在图画中的关键点没有插入正确的事件检测活动，那么一些意外的事件会被漏掉，应用系统就有可能会因此而崩溃。为了克服这个问题，一个设计者会试图在整个图画中布满各种事件检测机制。而这样又会导致一个过度复杂的图画。

[0016] 这种方法的另一个不足在于这些事件检测活动是按照一种随意、而没有系统的方式布置在图中，这使得验证各方行为是否一致的工作变得非常困难。许多重要的系统设计方面的问题就是由这个不足所造成的。

[0017] 因为一个主动式行为的建模语言无法很好地支持对外部事件做出的反应式行为，所以也有针对反应式行为而设计的建模语言来解决这个问题。

[0018] 反应式行为的建模语言包括有限状态机 (FSM)，UML 的状态图以及动态状态机 (Dynamic State Machine)。一个纯粹的 FSM 状态图包括若干个状态及这些状态的向外的迁移 (Transition)，而且每一个迁移的外部事件都定义得非常清楚。当处于某个状态时，FSM 在等待所有可能的外部事件。当收到一个适当的外部事件时，会触发一个迁移，而且伴随着这个迁移会执行一个关联的动作。当一个迁移发生之后，当前状态就变成不活跃的，而另外一个状态会成为活跃的。然后这个 FSM 会等待与这个新活跃的状态相关联的另外一组事件。因此，在任何时候，一个 FSM 一般都处于某个状态在等待可能发生的各种外部事件。

[0019] 一个纯粹的 FSM 有几个缺点。比如，纯粹的 FSM 没有内存就是一个局限。因为没有内存，所以纯粹的 FSM 的用处就非常有限。因为它的状态没有内存，任何状态变化或者历史信息，比如一个简单的计数器，都必须通过定义额外的状态来保存这些变化。假如一个计数器有无限取值的可能，那么就会需要无限多个的状态。

[0020] 从一个纯粹 FSM 扩展出来的有限状态机 (EFSMs) 或者寄存自动机 (Register Automata) 为每一个状态都增加了内存或数据变量以存值，比如数字和计数器等，这克服了纯粹 FSM 的一个主要问题。因此 EFSM 被广泛地用于反应式系统建模。

[0021] 当 EFSM 被用来为现实世界的事件或行为进行建模时，人们经常会发现一个分层的状态或者嵌套的状态对减少状态的数量以及组织一组复杂的状态有帮助。比如，UML 的状态图 (Statechart) 及 STATEMATE 就是这样的例子。状态图 (Statechart) 这个术语在这里就用来专指一个分层的有限状态机。

[0022] 虽然一个反应式行为建模语言适合用来处理外部事件的，但它却忽略了如何描述系统内部的一些复杂的主动式行为。正如我们已经从一些主动式行为建模语言所了解到的，这些内部主动式行为可以是非常复杂的。因此需要能把反应式行为建模语言和主动式行为建模语言结合起来使用。

[0023] 另外一个现有的主动式行为建模语言和反应式行为建模语言共有的缺点是它们都只能描述一个单一系统的行为。它们无论用哪种方式都无法有效地描述一个父系统内的组件间的复杂的内部消息交换。而一个分布式系统恰恰就包含多个相互通信，彼此协调的子系统，所以缺乏对多方行为的支持对一个分布式系统的限制非常大。

[0024] 除了上面提及的主动式行为建模和反应式行为建模，还有一些其它语言用来描述两个交流系统之间的协议。Web 服务编排描述语言 (WS-CDL) 就是这样的一个例子。一种可能的应用系统开发过程是用 WS-CDL 来定义协议，然后用另一种语言来设计一个参与系统

的行为，比如 BPEL，XPDL 或者 BPMN。但是，由于这些语言间存在的差异，从一个协议映射到两个参与系统，而且要保证这两个系统彼此协调，是有很多问题的。

[0025] 因此需要一个能把主动式行为建模语言和反应式行为建模语言都统一起来的建模语言，并且它能够描述现代分布式计算环境中的多方参与的动态行为。本发明正是提供了这样一个建模语言的解决方法。

发明内容

[0026] 本发明描述设计及开发具有动态行为的应用系统的方法和系统。

[0027] 本发明是旨在改进我们用图画方式描述一个应用系统的动态行为的方法，以便用一个计算机软件或硬件来实现这个系统。

[0028] 本发明还旨在提供必需的方法及过程来构造设计工具，代码生成器以及运行平台来实现定义好的动态行为。

[0029] 为了实现这些目的，本发明提供了描述应用系统的动态行为的一套系统、过程和过程的运用方法以及开发这种动态行为模型的方法。

[0030] 这个发明用一个动态系统（即 Dynamic System，简称 DS）类型来描述一个应用系统的动态行为。一个 DS 类型可以是简单的或组合的。一个简单的 DS 类型有一个行为类型来描述这个应用系统的简单行为。一个组合的 DS 类型（模型 1 的）有若干个阶段机 (Phased Machine) 类型用来描述复杂的顺序和并行的行为以及一个结束情况集或一个结束函数。每个阶段机类型有一个或多个阶段 (Phase) 类型和一个静态实例数（也可以可省略的）。每个阶段类型有一个子 DS 类型以及零个或多个迁移。每个迁移有一个迁移条件和一个目标阶段。

[0031] 如果定义了一个结束函数，它会返回一个结束代码，该结束代码标识了一个所述组合系统类型实例结束时的条件。如果定义了一个结束情况集，每个结束情况包括一对结束条件和结束代码，该结束条件描述一个所述组合系统类型实例结束时的条件，而该结束代码则标识对应的结束条件；

[0032] 一个阶段机类型可以用多个阶段类型来描述组合 DS 中的顺序行为，而另一个阶段机类型可以只用一个阶段类型来描述所述组合 DS 中的并行行为。

[0033] 所述 DS 类型可以特例化为一个自主系统（即 Proactive System，简称 PS）类型来描述一个在分布式环境中与其他系统交流的系统的反应式行为。一个 PS 类型可以是简单的或者组合的。一个简单的 PS 类型是一个简单 DS 类型的特例，它的行为包括一个外部 DS（即 External DS，简称 Ext-DS）类型以及跟在这个 Ext-DS 类型后面的内部 DS（即 Internal DS，简称 Int-DS）。Ext-DS 类型描述了和另外一个系统通讯时会发生的外部事件。而 Int-DS 类型则描述为响应所发生的那些外部事件而执行的内部动作。一个组合 PS 类型是一个组合 DS 的特例。因此，它和一个组合 DS 类型是相同的，除了它包含的是子 PS 类型，而不是子 DS 类型。

[0034] 一个 Ext-DS 类型也是一个 DS 类型的特例。一个 Ext-DS 类型可以是简单的或组合的。一个简单的 Ext-DS 类型是一个简单 DS 类型的特例，它的行为是一个事件类型，可以用来描述通讯协议中的一个向内的外部事件类型。而一个组合 Ext-DS 类型是一个组合 DS 类型的特例。因此，它和一个组合 DS 类型是相同的，除了它包含的是子 Ext-DS 类型，而不

是子 DS 类型。

[0035] 一个 Int-DS 类型也是一个 DS 类型的特例。一个 Int-DS 类型可以是简单的或组合的。一个简单的 Int-DS 类型是一个简单 DS 类型的特例，它的行为是一个动作类型，可以用来描述待执行的内部动作。一个组合 Int-DS 类型是一个组合 DS 类型的特例。因此，它和一个组合 DS 类型是相同的，除了它包含的是子 Int-DS 类型，而不是子 DS 类型。

[0036] 在运行时，一个 DS 类型，包括 PS 类型，Ext-DS 类型和 Int-DS 类型，可以生成多个 DS 实例 (instance)。一个 DS 实例可以是简单的或组合的。一个简单的 DS 实例有对它的 DS 类型的一个引用 (Reference) 和对执行它的行为的成果进行记录的一个结果。一个组合 DS 的实例包含有对它的组合 DS 类型的一个引用，一个结果以及若干个阶段机。每一个阶段机都有对它的阶段机类型的一个引用，一个当前阶段以及若干个阶段。每个阶段有一个对它的阶段类型的引用和一个子 DS 类型。

[0037] 在一个 DS 实例生成以后，在运行时刻，这个 DS 实例的各个部分可以在活跃状态和非活跃状态之间切换，以反映现实世界中一个系统行为的演进。和一个 DS 实例类似，在运行时刻，一个 Ext-DS，Int-DS 和 PS 实例的各个部分也都可以在活跃状态和非活跃状态之间切换，以表现现实世界中一个所对应系统的行为的演进。

[0038] 为了描述当一个简单 PS 变成活跃状态后要立即执行的 Int-DS，可以在这个简单 PS 类型的 Ext-DS 中设定一个预定义的“开始”事件。而为了描述当一个简单 PS 变成非活跃状态之前一瞬间要执行的 Int-DS，可以在这个简单 PS 类型的 Ext-DS 中设定一个预定义的“结束”事件。

[0039] 使用 DS 模型的一个主要优点在于整个设计流程只使用了一个一致性的通用模型，只是在不同的设计步骤会用到这个通用模型的某个特殊模型来关注某个特定行为。

[0040] 在本发明中提供了几种设计方法，从而可以从协议规范 (protocol specification) 推导出一个与其他系统有交流的系统的行为。首先，遵循一个通用性的转换方法，从一个角色的每一个协议都推导出一个参与者 PS。然后，把这些参与者 PS 合并成一个角色 PS。这些设计方法可确保协议规范和通讯各方保持彼此一致。

[0041] 这些设计方法中用到的某些组件可以被扩展得更具通用性，以用于其他行为建模语言。一个尚待定义的 Ext-DS 被扩展成一个尚待定义的外部活动，用于描述要接收某些尚待定义的外部事件的活动。而一个尚待定义的 Int-DS 被扩展成一个尚待定义的内部活动，用于描述要执行某些尚待定义的内部动作的活动。

[0042] 一个通用性的转换方法是用来把一个组合活动从协议规范转换为一个参与者行为规范。这个方法包括把每一个描述参与者要接收事件的向内的简单活动，替换成一个反应式组合活动，这个反应式组合活动把这个向内的简单活动作为它的外部活动，而且用一个尚待定义的内部活动来描述为响应所发生的向内的简单活动而执行的活动；并且把每一个描述参与者要发送事件的向外的简单活动，替换成一个反应式组合活动，这个反应式组合活动有一个尚待定义的外部活动，用于描述待接收的事件，而且把这个向外的简单活动作为它的内部活动，以对发生的尚待定义的外部活动做出响应。

[0043] 本发明提供了几种实施例以便用类似的方法分别采用 UML 状态图、UML 活动图、BPMN 和 BPEL 语言来描述主动式行为和反应式行为。

[0044] 本发明还提供了一个方法来从一个参与者组合活动推导出一个描述协议的协议

组合活动。这个方法包括：复制一份描述一个参与者行为的参与者组合活动；并且把所述参与者组合活动中既不接收向内事件也不发送向外事件的各个简单活动改变为空 (Null) 活动。

[0045] 本发明中提供了一些 DS 类型的图形化表示法，包括 PS 类型，Ext-DS 类型和 Int-DS 类型。一个 DS 类型可以图形化地以一个包含框来表示。一个代表组合 DS 类型的包含框内可以有若干个子包含框，若干个双向的交流链接，若干个生成箭头，若干个迁移箭头以及若干个结束点。每一个这些子包含框代表了一个简单的或者组合的子 DS。一个组合子 DS 可以把它的子子包含框画在它的框内。一个至少带有一个迁移箭头的子包含框代表了一个顺序的子 DS，而一个不带任何迁移箭头的子包含框则代表了一个并行的子 DS。

[0046] 一个简单的子 PS 可以图形化地用一个带有两个子包含框的包含框来表示。其中之一的子包含框代表了描述待接收外部事件的 Ext-DS，而另一个子包含框则代表了用于描述为响应所发生的外部事件而执行的内部行动的 Int-DS。

[0047] 借助本文的其他部分，包括附图及权利要求，可以体现本专利的其他特征和优势。

[0048] 下面会结合附图更进一步详细介绍本发明的特性和优势，以及各种实施例的结构和操作。

附图说明

[0049] 本发明上述的和 / 或附加的方面和优点从下面结合附图对实施例的描述中将变得明显和容易理解，其中，

[0050] 图 1 表示的是一个简化的框图，代表了一个分布式计算环境的例子；

[0051] 图 2 表示的是一个简化的框图，展示了一个可以用来实施本发明的计算机系统。

[0052] 图 3 表示的是一个包含子 DS 树图的 DS；

[0053] 图 4 表示的是一个简单 DS；

[0054] 图 5 表示的是一个组合 DS；

[0055] 图 6 表示的是一个结束情况，它包括一对结束条件和结束代码。

[0056] 图 7 表示的是一个 DS 常见的生命周期；

[0057] 图 8 表示的是协议，参与者 PS 和角色 PS；

[0058] 图 9 表示的是一个简单的 Ext-DS；

[0059] 图 10 表示的是一个简单的 PS；

[0060] 图 11 表示的是一个简单的 Int-DS；

[0061] 图 12 表示的是用图形语言表示的事件检查函数 (Matching procedure) 的伪代码；

[0062] 图 13 表示的是用图形语言表示的反应函数 (React procedure) 的伪代码；

[0063] 图 14 表示的是用图形语言表示的动作执行函数 (Act procedure) 的伪代码；

[0064] 图 15A 表示的是用没有任何迁移的并行阶段机来模拟一个并行单元；

[0065] 图 15B 表示的是用只有一个迁移的并行阶段机来模拟一个并行单元；

[0066] 图 16 表示的是一个模型 1 的组合 DS；

[0067] 图 17 表示的是一个用图形语言表示的 DS 例子；

[0068] 图 18 表示的是另一个用图形语言表示的 DS 例子；

[0069] 图 19 表示的是一个用图形语言表示的组合 PS 例子；

[0070] 图 20 表示的是一个组合 PS 的例子，它包含了一个子 PS，而这个子 PS 又包含一个组合的 Ext-DS 和一个组合的 Int-DS；

[0071] 图 21 表示的是一个包含了“开始”、“结束”预定义事件的组合 PS；

[0072] 图 22 表示的是一个没有包含“开始”预定义事件的 X-Server 参与者 PS；

[0073] 图 23 表示的是一个包含了“开始”预定义事件的 X-Server 参与者 PS；

[0074] 图 24 表示的是一个简单 DS 的类型 (Type) 属性 (properties) 和实例 (Instance) 属性；

[0075] 图 25 表示的是用 UML 类图 (class diagram) 来定义的一个简单 DS 的类型属性和实体属性

[0076] 图 26 表示的是一个简单 Ext-DS 的类型属性和实例属性；

[0077] 图 27 表示的是一个用 UML 类图来定义的简单 Ext-DS 的类型属性和实例属性；

[0078] 图 28 表示的是一个简单 PS 的类型属性和实例属性；

[0079] 图 29 表示的是一个用 UML 类图来定义的简单 PS 的类型属性和实例属性；

[0080] 图 30 表示的是一个简单 Int-DS 的类型属性和实例属性；

[0081] 图 31 表示的是一个用 UML 类图来定义的简单 Int-DS 的类型属性和实例属性；

[0082] 图 32A 表示一个带有一个结束情况集的模型 1 的组合 DS 的类型和实例；

[0083] 图 32B 表示一个带有一个结束函数的模型 1 的组合 DS 的类型和实例；

[0084] 图 32C 表示模型 1 的组合 PS 类型和实例。

[0085] 图 33 表示的是基于 DS 模型来设计，编译和运行一个应用软件的流程。

[0086] 图 34 表示的是基于 DS 模型来设计一个应用软件，然后在不同平台上编译和运行这个应用软件的流程；

[0087] 图 35 表示的是一个用 UML 类图来定义的模型 1 的组合 DS 的类型属性和实例属性；

[0088] 图 36 表示的是图 23 中参与者 PS”X-server”的组合 PS 类型；

[0089] 图 37 表示的是一个与客户端和服务端有交流的系统“X”；

[0090] 图 38 表示的是协议的例子“协议 0”和“协议 1”；

[0091] 图 39 表示的是从图 38 中“协议 0”推导出来的“X-server”的参与者 PS；

[0092] 图 40 表示的是从图 38 中“协议 1”推导出来的“X-client”的参与者 PS；

[0093] 图 41 表示的是协议“协议 2”；

[0094] 图 42 表示的是如何把参与者 PS 合并成一个角色 PS；

[0095] 图 43 表示的是如何加上内部的交流与行为；

[0096] 图 44 是使用一个组合协议“协议 2”的“X”的角色 PS；

[0097] 图 45 表示的是用 UML 活动图描述的“协议 0”和“协议 1”；

[0098] 图 46 表示的是用 UML 活动图表示的从图 45 中“协议 0”推导出来的“X-server”；

[0099] 图 47 表示的是用 UML 活动图表示的从图 45 中“协议 1”推导出来的“X-client”；

[0100] 图 48 表示的是用 UML 活动图的“分叉”和“汇合”节点来描述由“X-server”和“X-client”合并而成的角色 PS；

[0101] 图 49 表示的是用 BPMN 来描述的“协议 0”和“协议 1”；

[0102] 图 50 表示的是用 BPMN 来描述从“协议 0”推导得到的“X-server”；

[0103] 图 51 表示的是用 BPMN 来描述从“协议 1”推导得到的“X-client”；

[0104] 图 52 表示的是用 BPMN 的“分叉 (Fork)”和“汇合 (Join)”并行网关 (Parallel Gateway) 来描述由“X-server”和“X-client”合并得到的角色 PS；

[0105] 图 53 表示的是用 BPMN 的并行框图 (Parallel Box) 来描述由“X-server”和“X-client”合并得到的角色 PS；

[0106] 图 54 表示的是用状态图描述的“协议 0”和“协议 1”；

[0107] 图 55 表示的是用状态图描述的由“协议 0”推导出的“X-server”；

[0108] 图 56 表示的是用状态图描述的由“协议 1”推导出的“X-client”；

[0109] 图 57 表示的是用状态图描述的由“X-server”和“X-client”合并得到的角色 PS；

[0110] 图 58 表示的是由“X-server”参与者组合活动推导出的一个协议；

[0111] 图 59 表示的是如何遵循本发明所描述的设计流程来设计一个应用系统，然后在不同的运行平台上运行这个应用系统。

[0112] 图 60 表示的是一个组合活动，它包含有顺序的子组合活动和并行的子组合活动。其中有一个子组合活动是一个反应式组合活动；

[0113] 图 61 表示的是一个组合活动，它包含有顺序的子组合活动和并行的子组合活动以及一个交流链接；

[0114] 图 62 表示的是一个组合活动，它包含有子组合活动，一个交流链接以及两个结束点；

[0115] 图 63 表示的是一个组合活动，它包含有顺序的子组合活动和并行的子组合活动。外部活动是通过一个箭头和内部活动相连。

具体实施方式

[0116] 下面会通过一个或多个实施例来描述本发明。下面的描述中会提供很多细节来更加透彻地阐明本发明。但是显然对一个本领域的技术人员而言，即使没有这些描述，他也可以实施本发明。另一方面，本文没有对一些众所周知的特性进行详细描述以免使得本发明重点不清。

[0117] 现实世界中的许多系统都有复杂而动态的行为。一个行为之所以是复杂的，是因为它会涉及许多系统单元，而一个行为之所以是动态的，则是因为它会随着时间而演变。对一个复杂行为建模的基本指导原则是先对复杂行为进行分解然后再各个击破（建模）。本发明用动态系统 (DS) 和它的子系统（子 DS) 来对任何的复杂行为进行分解与建模。这些复杂行为可以在很广的的范围内适用，既可以是图 1 中所描述的实时的或者分布式的软、硬件系统，也可以是现实世界中像人和组织这样的实体。

[0118] 一个分布式的环境一般都是由多个通过网络互联的硬件系统组成。每个硬件系统如图 2 所示，包含有一个中央处理单元 (CPU)，存储器 (ROM，RAM 和磁盘）以及输入输出设备。

[0119] 这硬件系统是一种计算机，它可以是 PDA(Personal Data Assistant)、笔记本、桌面机，也可以是工作站、主机、实时控制设备、网络设备或者任何具有计算能力和输入输出能力的设备。

[0120] 在本发明中，类型 (Type) 或者类 (Class) 与实例 (Instance) 是有区别的。类型或类是一个设计阶段的概念，用来设定一个实体 (Entity) 的某些行为属性。而实例是运行时刻的概念，它的行为属性是由所属的类规定好的。除了特别申明为类的，本发明中所用的术语一般都指具体的实例。

[0121] 用 DS 模型描述动态行为

[0122] 为了把复杂的系统行为分解成更小的组件，DS 模型借助了这样一个常识，即我们所在的宇宙有两个基本的时间和空间维度这一事实。沿着时间维度一个复杂的系统可以分解为顺序的子 DS，而沿着空间维度一个复杂的系统分解为并行的子 DS。在时间和空间的任何一点上，一个 DS 可以被分解为多个并行子 DS 和多个顺序子 DS。而且每个子 DS 都可以沿着这两个维度继续分解为它自己的子 DS。

[0123] 一般而言，一个 DS 可以包含一组顺序的子系统和一组并行的子系统。因为每个子系统又可以包含它自己的子系统，因此一个 DS 一般可以分解成由子系统构成的树状图。在这棵树上处于中间层次的 DS 对它的子系统而言是父系统，而对它的父系统而言又成了子系统。因此根据不同的角度一个 DS 可以被称作 DS，父 DS 或者子 DS。如图 3 所示，DS0 是 DS1 和 DS2 的父系统。因此 DS1 是 DS0 的子系统，而同时它又是 DS3 和 DS4 的父系统。

[0124] 当描述互联世界里一个系统的行为时，需要同时考虑外部和内部准备就绪的情况。一般而言，只有当一个系统自己准备就绪，并且某个外部条件后被满足后才能执行一个特定的动作。本发明也会反映出系统间的这些相互依赖关系。

[0125] 一个 DS 可以是简单的或者组合的。一个简单的 DS（如图 4 所示）有一个行为和一个结果。这个行为就是用来描述 DS 的行为，而结果则可以被这个行为用来记录行为执行完毕后的最终成果。一个行为的例子可以是当特定条件满足时，一个 DS 就执行一个特定的动作。

[0126] 请注意：在图 4 中一个简单的 DS 是用一个在顶部有标签的框来表示的。而 DS 的组件则用画在大框内部的小框表示。本文档都会使用这个规范。

[0127] 一个组合的 PS（如图 5 所示）有如下组件：

[0128] ●零个或者多个阶段机 501，是用来表示这个组合 DS 内的顺序行为。

[0129] ●每个阶段机都有一个或者多个阶段 502，以及一个当前阶段 503。

[0130] ●当一个阶段机变成活跃状态时，它的多个阶段中的一个就会成为活跃状态。当阶段机处于活跃状态时，它的各个阶段会根据自己的迁移定义在活跃状态和非活跃状态间切换。一般情况下，当阶段机处于活跃状态时，它最多只有一个阶段能处于活跃状态。

[0131] ●一个正常的阶段有一个顺序的子 DS 504 及零个或多个迁移 (Transition)505。

[0132] ●这个顺序的子 DS 包含了这个阶段的行为。它可以是一个简单的 DS，或者一个组合的 DS。

[0133] ●每个迁移都有一个迁移条件 506 和一个目标阶段 507。

[0134] ●在当前阶段变为非活跃状态时，另一个阶段就会被选中并成为活跃状态。一个迁移会指明如果它的迁移条件得到满足，它的目标阶段就是在迁移结束后下一个成为活跃状态的阶段。这迁移条件可以使用顺序子 DS 结果中的信息或者它在运行时刻可以获得的任何其他信息来做判断。

[0135] ●迁移中指定的目标阶段必须是同一个阶段机中的阶段。换言之，迁移不能够跨

越阶段机的边界。请注意，一个阶段可以作为多个源阶段的目标阶段。另外，还允许循环迁移，包括指向当前阶段自己的迁移。这样就可以描述循环重复的行为。

[0136] ●因为任何时候都最多只能有一个阶段处于活跃状态，所以从同一个源阶段出来的各个迁移条件必须是彼此排他的 (mutually exclusive)。

[0137] ●当前阶段是在运行时刻指向阶段机中处于活跃状态的阶段的一个指针 (Pointer) 或者一个引用 (Reference)。

[0138] ●为了设置当前阶段，阶段机可以有两种特殊的阶段。

[0139] ●其中一种特殊阶段是初始阶段。一个阶段机最多可以有一个初始阶段。初始阶段的作用是在阶段机成为活跃状态时选择最先成为活跃状态的阶段。这个初始阶段有一个空的顺序子 DS 和一组初始迁移。

[0140] ●另外一种特殊的阶段是空 (Null) 阶段。当一个阶段机的当前阶段指向一个空阶段时，整个阶段机就变成非活跃状态。一个阶段机可以有一个默认的 (default) 空阶段以及零个或多个用户定义的空阶段。一个空阶段是不包含任何子 DS 和任何迁移的阶段，它只有一个和其他空阶段相区别的独特标识 (ID)。默认的空阶段所预定义的 ID 和任何用户定义的空阶段的 ID 都不会相同。空阶段的这个 ID 可以被结束情况或结束条件（在下文会有描述）用来判定一个组合 DS 是否应当结束。

[0141] ●零个或多个并行单元 508

[0142] ●每个并行单元包含一个并行的子 DS 509 来代表组合 DS 中的并行行为，一个静态实例数（可省略的）510 以及一个用来指明运行时刻这个并行子 DS 是否处于活跃状态的模态 (Mode)511。

[0143] ●这个静态实例数（可省略的）是一个数字，用来指明当生成包含它的组合 DS 时，会静态地生成多少个并行子 DS 的实例。如果省略了静态实例数，静态实例数的取值默认为 1。当生成包含它的组合 DS 时，总会静态地生成一个并行子 DS 的实例。除了这些静态生成的实例，在组合 DS 中也可以在运行时刻根据用户定义的行为而动态地生成这些并行子 DS 的实例。动态生成的被称为动态子 DS，而那些伴随父 DS 生成的被称为静态子 DS。在并行子 DS 实例的数目无法事先确定的情况下，动态子 DS 会很有用。

[0144] ●当一个并行子 DS 在运行时刻变成活跃状态时，它的模态就会被设置成“活跃的”。当它变成非活跃状态时，它的模态又会被设置成“非活跃的”。

[0145] ●当一个组合 DS 成为活跃状态时，它所有的并行子 DS 的实例都会成为活跃状态。

[0146] ●这些并行子 DS 每一个都可以是简单的或者是组合的。

[0147] ●一个包含零个或多个结束情况的结束情况集 512，其中每一个结束情况 600 用来指明在哪种情况下整个组合 DS 应该变成非活跃状态。每一个结束情况可以显式地定义为一对结束条件 601 和与之对应的结束代码 602（如图 6 所示）。这个结束条件可以用一个逻辑表达式，一个数学公式或者数学符号，甚至是一段文字描述来设定。同一个组合 DS 中的每一个结束代码都是不同的。结束代码通常是由设计工具生成或者由设计人员设定的一个数值。当某一个结束条件测试为“真”时，组合 DS 就会结束并且把对应的结束代码记录在组合 DS 的结果中。根据在结果中记录的结束代码我们就可以知道组合 DS 是在哪种情况下结束的。另外，结束情况集也可以定义为一个结束函数，这个函数既可以是一个面向函数程序语言的函数，也可以是一个面向对象程序语言的方法。这结束函数会在内部用到一组

隐式定义的结束条件来判断是否满足了某个结束情况，并返回一个结束代码以说明具体是哪个结束条件被满足了。

[0148] ●一个结束条件的实施例可以是当任何阶段机或者并行子 DS 结束时，这个组合 DS 就结束；

[0149] ●另一个结束条件的实施例是当某一个特定的并行子 DS 结束时，这个组合 DS 就结束；

[0150] ●如果在结束情况集中没有指明结束情况，默认的结束情况是当所有的阶段机和并行子 DS 成为不活跃状态时，包含它们的组合 DS 就变成不活跃状态，同时会把一个预定义的默认结束代码记录在结果中。这个默认的结束情况是对任何一个组合 DS 都有效的。

[0151] ●一个属于整个组合 DS 的结果 513。这个结果是用来记录一个结束函数返回的结束代码。或者，如果定义了一个结束情况集，则结果会记录那个判断为真的结束条件所对应的结束代码。

[0152] 请注意，如果允许某个组件有多个实例，这个组件就用带阴影的图画表示。比如，在一个组合 DS 中可以有若干个阶段机和并行单元。在一个阶段机中又可以一个或多个阶段。每个阶段还可以有多个迁移。因此如图 5 所示，每个阶段机，并行单元，阶段和迁移都用有阴影的图画表示。

[0153] 许多现实世界中的系统会随着时间的推移有不同的行为表现。为了表现这种随时间推移的行为变化，DS 的某些部分会随着时间推移在活跃状态和非活跃状态之间切换，但是在任何一个时间点上只有活跃的那部分是生效的。因此在 DS 内部控制子 DS 活跃与否就可以控制系统的行为变化了。

[0154] 一个 DS 的常见生命周期如图 7 所示：在一个 DS 生成以后，它会成为活跃状态或者非活跃状态。一个活跃状态的 DS 在它的行为完成或者结束以后会变成非活跃状态。当一个 DS（不管它是活跃的还是非活跃的）不能再次成为活跃状态时可以被销毁。在图 7 中，一个长方形代表一个阶段，这个阶段可以有一个标签（比如“已生成”）。一个箭头代表了从一个阶段向另一个阶段的迁移。一个在长方形边框上的空心小长方形代表一个初始阶段或一个空阶段，它表明这个 DS 不存在（要么是还没有生成，要么是已经被销毁）。一个阶段还可以包含子阶段。比如，阶段“已生成”就包含“活跃”和“非活跃”两个子阶段。

[0155] 在运行时刻可以通过装载相关代码和数据到内存或者持久存储设备来生成一个 DS 类所定义的 DS 实例。然而生成和销毁 DS 的具体过程对本发明来说不是很重要，因此，这个过程留给实施本发明的应用系统去决定。应用系统只需要确保一个 DS 实例生成以后它所有的静态组件都会生成，并且为进入活跃状态或非活跃状态做好了准备，一直到这个 DS 最后被销毁。一个 DS 变为活跃（或非活跃）状态的具体过程描述如下：

[0156] 在运行时刻一个简单的 DS（如图 4 所示）变为活跃（或非活跃）状态的过程如下：

[0157] ●当一个简单的 DS 变成活跃状态时，它的行为就会被执行；

[0158] ●当简单的 DS 完成它的行为以后，执行的成果被记录在结果中，随后这个简单的 DS 就变成非活跃状态。

[0159] 在运行时刻一个组合的 DS（如图 5 所示）变为活跃（或非活跃）状态的过程如下：

[0160] ●当一个组合的 DS 变成活跃状态时，它所有的阶段机和静态的并行单元都会变成活跃状态；

[0161] ●当一个并行单元变成活跃状态时，它的并行子 DS 会变成活跃状态，而且这个并行单元的模态会被设成“活跃的”；

[0162] ●当一个阶段机变成活跃状态时，它的阶段之一（并包括该阶段所包含的顺序子 DS）会变成活跃状态。

[0163] ●阶段机中最先成为活跃状态的阶段是根据以下方法加以选定的：如果阶段机中只有一个阶段，这个阶段会被选为当前阶段，并成为活跃状态。否则这些阶段之一必须是设定为初始阶段。在一个阶段机变成活跃状态时会对初始阶段的这些初始迁移的迁移条件进行判断。如果某一个迁移条件被满足了，对应的目标阶段会被选作为当前阶段，并成为活跃状态。

[0164] ●当一个顺序的或者并行的子 DS（它可以是简单的或者组合的）成为活跃状态时，前文针对简单或组合 DS 的行为会递归性性地重复发生在子 DS 身上。

[0165] ●当一个并行的子 DS 后来变成非活跃状态时，它的模态会被设置为“非活跃的”；

[0166] ●当一个顺序的子 DS 后来变成非活跃状态时，它的阶段随之也变成非活跃状态，同时这个阶段的迁移条件也会被评估。如果有一个迁移条件判断为“真”，相应的目标阶段（可以是空阶段）会变成活跃状态，而且当前阶段会指向这个目标阶段。如果没有迁移条件判断为“真”，会默认地把当前阶段指向默认的空阶段。

[0167] ●当一个阶段机的当前阶段指向一个空阶段时，整个阶段机会变为非活跃状态。

[0168] ●任何时候，当任何一个阶段机或并行单元变成非活跃状态时，它的组合 DS 的结束情况集将会被评估。如果显式地定义了一个包含结束条件和结束代码的集合（如图 6 所示），则任何时候当一个结束条件判断为“真”时，对应的结束代码就会被记录在结果中，然后组合 DS 会变成非活跃状态。如果定义了一个结束函数，它会测试每一个结束情况，如果有任何一个结束情况被满足，一个对应的结束代码就会被记录在结果中，然后这个组合 DS 就会变成非活跃状态。否则，这个组合 DS 会继续保持活跃状态。

[0169] ●当一个组合 DS 变成非活跃状态时，它所有的阶段机（包括它们处于活跃状态的顺序子 DS）以及并行单元（包括它们的并行子 DS）都会变成非活跃状态。

[0170] 重复性的行为可以用重复性的阶段来描述。一个重复性的阶段是一个阶段，它的某一个迁移会把它自己作为迁移的目标阶段。

[0171] 在本发明中支持递归性的 (recursive)DS。一个递归性的 DS 有一个或多个子 DS 具有和递归性 DS 相同的类型，或者与递归性 DS 的直接或间接父 DS 有相同的类型。有了递归性的 DS，一个相同的行为会在越来越小的范围内重复。一般情况下，当选择不再进一步生成新的递归性子 DS 时，递归就会被终止。

[0172] 用 DS 来进行行为建模的优点在于：

[0173] ● DS 的组合结构直接反映了我们所生活的世界是由时间维度和空间维度所构成的。

[0174] ●从时间维度和空间维度来观察东西对人类是一种很自然的事。它较于现有的其他行为建模语言所定义的节点和链接赋予了更重要的含义。因此 DS 模型更易于学习和理解。

[0175] ●通过控制各种子 DS 变成活跃状态和非活跃状态，一个 DS 的行为可以随着时间非常有序地演进；

[0176] ●由若干结束条件与结束代码构成的结束情况集为控制一个组合 DS 什么时候应该结束提供了一种强大而灵活的机制；

[0177] ●和许多采用分叉节点和汇合节点的现有技术不同的是，一个组合的 DS 呈现出来的块状结构组织为它的所有组件定义了一个清晰的生命周期。

[0178] ●因为每个子 DS 都可以是组合的，所以 DS 模型支持以递归的方式不断细化对系统行为的描述。

[0179] ●通过重复阶段和动态的并行子 DS 还可以描述重复的行为和动态的行为；特殊的 DS：Ext-DS，PS，Int-DS

[0180] 尽管 DS 是一个描述动态行为的强有力的工具，但为了充分描述一个分布式环境中与多个对端系统交流的应用系统的行为，我们还需要使用三种 DS 的特例。第一种是外部 DS(Ext-DS)，用于描述一个应用系统必须处理的外部活动，包括向内的消息和事件。第二种是内部 DS(Int-DS)，用于描述一个应用系统为了完成某些工作要执行的内部活动。第三种是把 Ext-DS 和 Int-DS 集成在一起的自主系统 (PS)，用于描述一个应用系统的反应式行为和自主行为。

[0181] 外部 DS(Ext-DS)

[0182] 一个 Ext-DS 是用来定义一个 PS 在执行内部动作之前所等待的外部事件。

[0183] Ext-DS 是 DS 的一个特例，它可以是简单的或者组合的。作为一个简单 DS 的特例，一个简单的 Ext-DS（如图 9 所示）也有一个行为和一个结果。它的行为就是一个事件类型。而事件类型是用来描述一个 PS 可能接收的信号、消息或者事件的类型。一个事件类型可以下列语言之一来描述：1) 一种程序语言，比如 C 和 JAVA；2) 一种数据标记语言，比如 ASN.1(Abstract Syntax Notation One)；3) 一种基于 XML 的语言，或者 4) 一种图形语言。

[0184] 在运行时刻一个简单的 Ext-DS 变为活跃（或非活跃）状态的过程如下：

[0185] ●当一个简单的 Ext-DS 变成活跃状态时，它的事件类型也变成活跃状态。

[0186] ●如果随后这个简单的 Ext-DS 收到了一个和它的事件类型相符的消息，那么这个消息就会被保存下来备以后使用，然后这个简单的 Ext-DS 就会变成非活跃状态。否则，如果这个消息和 Ext-DS 的事件类型不相符，它将被抛弃掉，而这个简单的 Ext-DS 会继续保持活跃状态。

[0187] 一个组合的 Ext-DS 和一个组合的 DS 相似（如图 5 所示），只是子 DS 换成了子 Ext-DS 而已。所有这些子 Ext-DS（不论是顺序的还是并行的）都可以是简单的或者组合的。

[0188] 因为一个组合的 Ext-DS 是组合 DS 的一个特例，前文关于一个组合的 DS 变成活跃（或非活跃）状态的过程同样适用于一个组合的 Ext-DS。只需将文字中的 DS 替换成 Ext-DS 就可以了。

[0189] 当一个简单 PS 在运行时刻收到一个向内的事件时，它会调用函数“检查事件 ()”去检查这个事件是否是它的 Ext-DS 中定义的类型相符。这个检查事件 () 函数返回下列值之一：

[0190] ●不匹配，这表示事件类型不相符

[0191] ●部分匹配，这表示事件类型相符，这个向内的事件已被保存起来供以后处理，但是 Ext-DS 尚未结束，因为 Ext-DS 还在等待更多的事件；或者

[0192] ●完全匹配，这表示事件类型相符，这个向内的事件已被保存起来供以后处理，而且 Ext-DS 已经结束。

[0193] 下面用伪代码提供了一个处于活跃状态的 Ext-DS 的运行时刻事件检查函数的范例。其中参数“Ext-DS”是用来描述待接收的向内的外部事件的外部 DS，而参数“事件”是待检查的向内的事件。

[0194] 请注意，方括号“[]”之间的内容是注释。

[0195] 检查事件 (Ext-DS，事件){

[0196] 如果 Ext-DS 是简单的，

[0197] 检查收到的事件的类型和 Ext-DS 的事件类型是否相同；

[0198] 如果是“否”，返回“不匹配”；

[0199] 否则 [事件类型是正确的]

[0200] 保存这个事件；

[0201] 返回“完全匹配”；

[0202] 否则 [Ext-DS 是组合的]，

[0203] [检查全部阶段机]

[0204] 对每一个“当前阶段”不是空的阶段机，执行以下步骤：

[0205] 为顺序的子 Ext-DS，递归地调用检查事件 () 函数；

[0206] 如果它返回“部分匹配”，即返回“部分匹配”

[0207] 否则，如果它返回“完全匹配”，

[0208] 逐个检查每一个迁移；

[0209] 如果任何一个迁移条件是“真”，

[0210] 把当前阶段指向迁移的目标阶段；；

[0211] 否则 [如果没有一个迁移条件为“真”]，

[0212] 把当前阶段的取值设为默认的空阶段；

[0213] 如果当前阶段不是空阶段，

[0214] 返回“部分匹配”；

[0215] 否则 [当前阶段是空阶段]，

[0216] 检查结束情况集；

[0217] 如果有任何结束情况得到满足，

[0218] 把结束代码记录在结果中；

[0219] 返回“完全匹配”；

[0220] 否则 [没有结束情况被满足]，

[0221] 返回“部分匹配”；

[0222] 否则 [返回“不匹配”]，去检查下一个阶段机；

[0223] [如果执行到这里，所有阶段机都是“不匹配”，去检查所有的并行单元]

[0224] 为每一个活跃状态的并行单元，执行以下步骤，

[0225] 对于并行子 Ext-DS 递归地调用检查事件 () 函数；

[0226] 如果它返回“部分匹配”,返回“部分匹配”;

[0227] 否则,如果它返回“完全匹配”,

[0228] 设置模状为“非活跃的”;

[0229] 检查结束情况集;

[0230] 如果有任何结束情况被满足,

[0231] 在结果中记录结束代码;

[0232] 返回“完全匹配”;

[0233] 否则[没有结束情况被满足],返回“部分匹配”;

[0234] 否则[返“不匹配”],去检查下一个并行单元;

[0235] [如果执行到这里,这个事件和所有顺序的Ext-DS和并行的子Ext-DS都不相符,]

[0236] 返回“不匹配”;

[0237] }

[0238] 为了清楚起见,这个事件检查函数也会用本发明后面将介绍的图形语言表示在图12中。

[0239] 内部DS(Int-DS)

[0240] 一旦一个简单的PS被一个与Ext-DS完全匹配的事件所触发,对应的Int-DS就会被执行。因此,一个Int-DS可以看作一个PS的行为中主动的那一部分行为,或者PS的主动式行为。

[0241] 一个Int-DS是DS的一个特例,它可以是简单的或者组合的。作为一个简单DS的特例,一个简单的Int-DS(如图11所示)也有一个行为和一个结果。这个行为就是一个动作。在本发明中的动作是指不能再进一步分解的最基本的行为单元。一个动作可以是读写一段信息,也可以是查找或匹配信息,还可以是读Ext-DS所保存的事件,或者是构建、发送事件,或者是生成、销毁DS实例,或者是调用由其它系统执行的复杂动作。一个动作可以用面向函数程序语言(比如C语言)的一个函数来实现,也可以用面向对象程序语言(比如JAVA语言)的一个方法来实现,或者用脚本语言(比如Linux的Shell语言)的一个脚本来实现。一个动作也可以用脚本语言、程序语言的一个代码片断来实现,或者是用字节代码(比如Java的字节代码),甚至包括汇编语言来实现。它的执行成果则可以记录在结果中。

[0242] 在运行时刻一个简单的Int-DS变成活跃(或非活跃)状态的过程描述如下:

[0243] ●当一个简单的Int-DS变成活跃状态时,它的动作也变成活跃状态,并且被执行;

[0244] ●在它的动作被执行完以后,这个简单的Int-DS随之变成非活跃状态。

[0245] 一个的组合Int-DS和一个组合的DS相似(如图5所示),只是把子DS换成子Int-DS而已。这些子Int-DS(不管是顺序的还是并行的)都可以是简单的或者是组合的。

[0246] 因为一个组合的Int-DS是一个组合DS的特例,前文关于一个组合的DS变成活跃(或非活跃)状态的过程同样适用于一个组合的Int-DS。只需将文字中的DS替换成Int-DS就可以了。

[0247] 当一个简单PS在运行时刻收到一个向内的事件时,它会调用检查事件函数去检查这个事件是否是它的Ext-DS中所定义的类型。如果返回值是“完全匹配”,这个简单PS会调用动作执行函数“动作()”去执行它的Int-DS中所定义的内部活动。“动作()”函数

返回下列值之一：

[0248] ●错误，这表示发生了一个错误；或者

[0249] ●正确，这表示内部活动已经被成功执行

[0250] 下面用伪代码描述了一个处于活跃状态的 Int-DS 的动作执行函数的范例，其中参数”Int-DS“是定义要执行的内部活动的内部 DS：

[0251] 动作 (Int-DS){

[0252] 如果 Int-DS 是简单的，

[0253] 执行 Int-DS 的动作来对保存的事件进行处理；

[0254] 如果检测到错误，返回“错误”；

[0255] 否则，返回“正确”；

[0256] 否则 [Int-DS 是组合的]，

[0257] [检查全部阶段机]

[0258] 为每一个当前阶段不是空阶段的阶段机，执行以下步骤：

[0259] 为顺序的子 Int-DS 递归地调用动作 () 函数；

[0260] 如果子 Int-DS 返回“错误”，返回“错误”；

[0261] 否则，逐个检查每一个迁移；

[0262] 如果有任何一个迁移条件是“真”，

[0263] 把当前阶段指向迁移的目标阶段；

[0264] 否则，把当前阶段指向默认的空阶段；

[0265] 如果当前阶段不是空阶段，对新的“当前阶段”重复这个过程；

[0266] 否则，

[0267] 检查结束情况集；

[0268] 如果任何结束情况被满足，

[0269] 把结束代码记录到结果中；

[0270] 返回“正确”；

[0271] 否则，去检查下一个阶段机；

[0272] [检查所有的并行单元]

[0273] 为每一个活跃状态的并行单元执行以下步骤，

[0274] 对并行子 Int-DS 递归地调用动作 () 函数；

[0275] 如果并行单元返回“错误”，则返回“错误”；

[0276] 否则，

[0277] 设置模态为“非活跃的”；

[0278] 检查结束情况集；

[0279] 如果有任何结束情况被满足，

[0280] 把结束代码记录到结果中；

[0281] 返回“正确”；

[0282] 否则，去检查下一个并行单元；

[0283] }

[0284] 为了清楚起见，这个动作函数也会用本发明在后面将介绍的图形语言表示在图 14

中。

[0285] 自主系统 (PS)

[0286] 一个在外部交流或外部事件触发下能做出反应式行为的应用系统可以用一个自主系统（即 Proactive System，简称 PS）来定义。

[0287] 一个 PS 是一个 DS 的特例，它可以是简单的或者组合的。作为一个简单的 DS 的特例，一个简单的 PS 也有一个行为和一个结果。这个行为包括一个外部 DS(Ext-DS) 以及后面跟着的一个内部 DS(Int-DS)。Ext-DS 描述了要发生的外部事件，而 Int-DS 描述了为响应所发生的 Ext-DS 中定义的外部事件而执行的内部活动。Ext-DS 和 Int-DS 都可以是简单的或组合的。结果则可以被 Int-DS 用来记录动作执行的成果。

[0288] 在运行时刻一个简单的 PS 变成活跃（或非活跃）状态的过程描述如下：

[0289] ●当一个简单的 PS 变成活跃状态时，它的 Ext-DS 也变成活跃状态；

[0290] ●

[0291] 这个简单 PS 的 Ext-DS 随后接收到一个完全匹配的事件（如前文所述），这个简单 PS 的 Int-DS 就变成活跃状态，并且会被执行；

[0292] ●当它的 Int-DS 被执行完以后，这个简单的 PS 也随之变成非活跃状态；

[0293] 一个组合的 PS 和一个组合的 DS 相似（如图 5 所示），只是子 DS 被换成子 PS 而已。这些子 PS（不管是顺序的还是并行的）都可以是简单的或者组合的。

[0294] 因为一个组合的 PS 是一个组合 DS 的特例，前文关于一个组合 DS 变成活跃（或非活跃）状态的过程同样适用于一个组合的 PS。只需将文字中的 DS 替换成 PS 就可以了。

[0295] 当一个 PS 在运行时收到一个向内的事件时，它会调用反应 () 函数去处理这个事件。反应函数返回下列值之一：

[0296] ●未做任何事情，这表示没有执行任何内部活动；

[0297] ●未作部分完成，这表示内部活动的一部分被完成了，但是 PS 尚未结束；或者

[0298] ●完成，这表示内部活动被全部完成，而且 PS 已经结束。

[0299] 下面用伪代码描述了一个处于活跃状态的 PS 在收到事件后的反应函数的范例，其中参数“PS”是做出反应的 PS，参数“事件”是待处理的事件：

[0300] 反应 (PS，事件){

[0301] 如果 PS 是简单的，

[0302] 用事件作为参数，为 PS 的 Ext-DS 来调用检查事件 () 函数；

[0303] 如果返回结果是“不匹配”，返回“未做任何事情”；

[0304] 否则如果返回结果是“部分匹配”，返回“部分完成”；

[0305] 否则 [返回结果是“完全匹配”]，

[0306] 调用动作 () 函数，执行 Int-DS；

[0307] 返回“完成”；

[0308] 否则 [PS 是组合的]，

[0309] [检查全部阶段机]

[0310] 为每一个当前阶段不是空阶段的阶段机执行以下步骤：

[0311] 对于顺序的子 PS 递归地调用反应 () 函数；

[0312] 如果顺序的子 PS 返回“部分完成“，返回“部分完成”；

[0313] 否则如果它返回“完成”,检查每一个迁移 ;

[0314] 如果有任何一个迁移条件为“真”,

[0315] 把当前阶段指向迁移的目标阶段 ;

[0316] 否则 [没有迁移条件为真]

[0317] 把当前阶段设置为默认的空阶段 ;

[0318] 如果当前阶段不是空阶段,返回“部分完成”;

[0319] 否则 [当前阶段是空阶段],

[0320] 检查结束情况集 ;

[0321] 如果有任何结束情况被满足,

[0322] 把结束代码记录在结果中 ;

[0323] 返回“完成”;

[0324] 否则 [没有结束情况被满足],

[0325] 返回“部分完成”;

[0326] 否则 [返回“未做任何事”],去检查下一个阶段机 ;

[0327] [如果执行到这里,所有的阶段机都是返回“未作任何事情”,去检查所有的并行单元]

[0328] 为每一个活跃状态的并行单元执行以下步骤 :

[0329] 对于每一个并行子 PS 递归地调用反应 () 函数

[0330] 如果并行子 PS 返回“部分完成”,则返回“部分完成”;

[0331] 否则如果它返回“完成”,

[0332] 设置模态为“非活跃的”;

[0333] 检查结束情况集 ;

[0334] 如果有任何结束情况被满足,

[0335] 把结束代码记录在结果中 ;

[0336] 返回“完成”;

[0337] 否则 [没有结束情况被满足],返回“部分完成”;

[0338] 否则 [返回”未作任何事情”],去检查下一个并行单元 .

[0339] [如果执行到这里,未做任何事情] 返回“未做任何事情”;

[0340] }

[0341] 为了清楚起见,上述反应函数也会用本发明后面将介绍的图形语言表示在图 13 中。

[0342] 在不同的步骤中使用 Ext-DS, PS 和 Int-DS 这些特殊的 DS,比用一个通用性的 DS 有很多额外的好处 :

[0343] ● Ext-DS,PS 和 Int-DS 提供了多种层面的抽象方法,使得定义分布式环境中的复杂系统行为更加方便,因为它们中的每一个都以一种特殊的方式帮助设计者专注在系统行为的某一个方面。

[0344] ● PS 是对基于状态机的传统技术 (比如状态图技术) 来做扩展和改进的。而 Int-DS 是对基于流程图的传统技术 (比如 BPMN 技术和 UML2.0 活动图技术) 来做扩展和改进的。

[0345] ●因为 Ext-DS，PS 和 Int-DS 中的每一个都是同一个 DS 模型的特例，所以在设计的每个层面上都有很强的一致性，使得本发明更加容易学而且容易懂。

[0346] ●正如在本发明后面会展示的，因为这种一致性，可以系统化地、机械地从一个用 DS 所定义的通讯协议来推导出来一个用 PS 所定义的反应式行为。因此本发明为解决如何保证通讯协议以及参与各方彼此一致的难题提供了一种解决方案。

[0347] DS，Ext-DS，PS 和 Int-DS 的简化模型

[0348] 到目前为止，我们已经描述了 DS 的基本模型以及它的一些特例，比如 Ext-DS，PS 和 Int-DS。在本发明中它们都被称为 DS 模型。这个基本的 DS 模型（以后我们称之为“模型 0”）可以通过用并行阶段机来模拟并行单元进而简化。这个简化后的模型我们将称之为“模型 1”；

[0349] 一个模型 0 的组合 DS 中的一个并行单元可以用一个只有一个阶段及一个空迁移的阶段机来模拟。这个用来模拟并行单元的阶段机，我们在本发明中称之为“并行阶段机”。一个并行阶段机的阶段只有一个子 DS。它的空迁移可以由下面几种方法之一来实现：

[0350] ●不定义任何迁移，如图 15A 所示；

[0351] ●只有一个标识为空的迁移；

[0352] ●只有一个迁移，而它的迁移条件设置为真，它的目标阶段指向一个空阶段 1502，如图 15B 所示。

[0353] 这个并行阶段机还增加了一个组件“静态实例数”（可省略的）1604。和并行单元的静态实例数类似，这个阶段机的静态实例数用来指明当包含阶段机的 DS 生成时，要生成多少个这种阶段机的静态实例。如果没有设定静态实例数，静态实例数的取值默认为 1，因此当包含它的组合 DS 生成时，总会生成一个阶段机的实例。

[0354] 一个并行阶段机的子 DS 的行为是和该阶段机所模拟的并行单元的并行子 DS 的行为相同。

[0355] 在本发明中，用术语“顺序的阶段机”来表示一个没有受上述并行阶段机限制的正常的阶段机。一个顺序阶段机的静态实例数通常是被设置为“1”，因此当包含它的组合 DS 生成时，会生成一个顺序阶段机的实例。

[0356] 在用了并行阶段机来模拟组合 DS 中的并行单元后，一个模型 1 的组合 DS 就只需要有若干个阶段机 1601，一个结束情况集 1602 以及一个结果 1603。其中的这些阶段机有些是顺序的，有些则是并行的（如图 16 所示）。

[0357] 因为每一个并行单元都是用一个阶段机来模拟，所以一个组合的 DS 变成活跃（或非活跃）状态的过程对模型 0 和模型 1 都是一样的，只是模型 1 的组合 DS 不再有任何并行单元，所以和并行单元有关的那部份代码不会再被执行。

[0358] 同样地，一个组合的 Ext-DS，组合的 PS 或组合的 Int-DS 也是一个模型 1 的组合 DS 的特例。因为每一个并行单元都被替换成一个并行阶段机了，这些模型 1 的 DS 组件变成活跃（或非活跃）状态的过程和模型 0 的相同，只是和并行单元有关的那部分代码不会再执行。类似的，前文描述的一个活跃的 Ext-DS 的事件检查函数，一个活跃的 Int-DS 的动作函数以及一个活跃的 PS 的反应函数对模型 1 还是同样适用。

[0359] 图形化表示法

[0360] 本发明的一种实施例是用人可以读懂的一种图形语言来表示 PS，Ext-DS，Int-DS

或者任何其他由 DS 导出的组件（我们将把这些都称为“DS 组件”）。

[0361] 作为一个这种图形语言的例子，一个 DS 或子 DS 可以用一个包含框，或者带标签的包含框表示。这种包含框可以是各种形状的，比如多边形，圆形或者椭圆形等。一个包含若干子 DS（可以是顺序的或者并行的）的父 DS 可以用一个大框（代表父 DS）里包含若干个小框（代表子 DS）来表示。如果一个 DS 可以有多个实例，这个框就会带有阴影。另外，一个初始阶段或空阶段可以用组合 DS 框边上一个小框或者一个符号来表示。

[0362] 图 17 描述了用一种图形语言表示的组合 DS 的一个例子：组合 DS “DS0”1700 有三个顺序的子 DS（它们是子 DS “s0”1701，“s1”1702，和“s2”1703），3 个并行的简单子 DS（它们是子 DS “p0”1706，“p3”1709 和“p5”1708）以及两个并行的组合子 DS（它们是子 DS “p1”1705 和“p2”1708）。请注意，子 DS “p2”1708 和“p3”1709 带有阴影，这表示它们在运行时刻可以有多个实例。

[0363] 在本例中，一个简单的 DS 用一个内部带有名字或者一个简单的行为描述的长方框来表示。图 17 中的子 DS “p0”1706 和“p5”1704 就是两个简单 DS 的例子。

[0364] 一个迁移是用一根链接或者箭头（我们称之为迁移箭头）来表示，箭头指向的是迁移的目标阶段，而迁移条件则显示在靠近箭头旁作为迁移的一个标签。图 17 中有一个从子 DS “s2”1703 指向子 DS “s0”1701（目标阶段）的用“迁移条件 5”作为标签的迁移箭头。每一个具有一个或多个迁移箭头（向内或向外的）的子 DS 是一个顺序的子 DS，而没有迁移箭头的子 DS 则是一个并行的子 DS。

[0365] 一个生成箭头可以用来指向一个动态生成的动态组件。尽管一个生成箭头在图 17 中是用一个粗的虚线箭头 1707 作为图形语言的范例，它也可以用其他种类的链接表示，只要这种链接能够和迁移箭头区别开。正如图 17 所示，子 DS“p0”1706 可以动态地生成多个子 DS “p2”1708 的实例，而且子 DS “p2”1708 可以动态地生成子 DS “p3”1709 的多个实例。一个带有向内的生成箭头的并行子 DS，比如图 17 中的子 DS”p2”1708 和“p3”1709，是一个动态的并行子 DS，它的并行单元中的静态实例数会被设置成 0。而不带有向内的生成箭头的并行子 DS，比如图 17 中的子 DS “p0”1706 和“p1”1705，则是静态的并行子 DS，相应的静态实例数会被设置成 1，表示当”DS 0”1700 生成时，只会生成一个子 DS “p0”1706 和“p1”1705 的实例。

[0366] 请注意，在图 17 中的 DS 0 1700 没有显式地定义任何结束情况。正如前面所讲，默认的结束情况是当 DS0 1700 的所有子 DS 结束时，DS0 才结束。

[0367] 图 18 表示的是如何用可视化的方式显式地表示初始阶段，空阶段，结束条件和结束点。在图 18 中，初始阶段 1801 用左边框上的一个小框代表，它有两个迁移，其中一个指向包含子 DS “s0”1802 的阶段，另一个指向包含子 DS “s1”1803 的阶段。子 DS “s0”1802 有一个迁移指向子 DS “s2”1804，这个迁移的条件永远为“真”。子 DS “s2”1804 有 3 个迁移，其中一个标签为“迁移条件 2”的迁移指向空阶段 1806，标签为“迁移条件 3 “的迁移指向它自己，而标签为”迁移条件 4 “的迁移则指回到包含子 DS “s0”1802 的阶段。DS0 还包含一个并行的子 DS “p1”1805 和两个结束点。

[0368] 每个结束点代表了一个结束情况来指明在哪种情况下一个组合 DS 会结束。一个结束情况有一对结束条件和结束代码。一个结束点可以用组合 DS 的包含框的边框内或者边框上的一个符号或者小框来表示。作为一个例子，在图 18 中每一个结束点用边框上的一

个小的黑框 1807 表示，并且用相关的结束条件作为小黑框的标签。

[0369] 一个结束条件可以用一段文字，一个逻辑表达式或者一个数学公式来描述在哪种情况下一个组合 DS 会结束。结束条件“子 DS P1 已经结束”就是一个文字条件的例子，表示当子 DS P1 结束后，组合 DS “DS 0”1800 就会结束。结束条件“□”是一个逻辑表达式的例子，表示当所有组件都结束后，“DS0”1800 才会结束。结束条件“年龄＜ 10”是一个数学公式的例子，表示当“年龄”取值小于 10 的时候，“DS0”1800 就会结束。

[0370] 为了简化行为模型和对应的图形语言，下面定义了一些默认情况的范例 ：

[0371] • 一个永远为真的迁移条件可以被省略。

[0372] • 如果一个阶段只有一个和初始阶段相连的向内的迁移，而且它的迁移条件永远为“真”，那么这个迁移可以被完全省略。

[0373] • 如果一个阶段只有一个和空阶段相连的向外迁移，而且它的迁移条件永远为“真”，那么这个迁移也可以完全省略。

[0374] 图 19 描述了一个用图形语言表示的组合 PS 例子。在这个范例图画中，组合的”PS0”1902 包含两个顺序的子 PS（即子 PS “s0”1903 和子 PS “s1”1904）、三个并行的子 DS（即子 PS “p0”1905，子 PS “p1”1906 和子 PS “p2”1911）和两个结束条件分别为“结束条件 0”和“结束条件 1”的结束点 1913 和 1915。

[0375] 一个简单的 PS 1906 用一个包含它的 Ext-DS 组件 1908 和 Int-DS 组件 1910 的框表示。为了区分一个简单 PS 的 Ext-DS 和 Int-DS，在这两个组件之间画了一根分隔线（一根垂直线 1909 或者一根水平线 1907）。子 PS “p0”1905 和”p1”1906 就是两个这样的例子。

[0376] 两个 PS 之间的交流可以用一个连接它们的双向链接来表示（我们称之为交流链接）。图 19 表示的是组合“PS 0”1902 和“PS1”1900 有交流，子 PS “s1”1904 和子 PS “p2”1911 有交流，子 PS “p0”1905 和子 PS “p1 “1911 有交流。在这个例子中，一个交流链接是用虚线表示的（图 19 中的“协议 0”就是这样的一个例子）。然而，它也可以用其他种的链接来表示，只要它在视觉上能够和一个迁移箭头和生成箭头区分开来。这些交流链接还可以加上一个或两个箭头（可省略的）来指明交流的方向。一个只发送一个方向的事件的简单交流可以用只带一个箭头的虚线表示（图 19 中的“协议 2”就是这样的一个例子）。一条两端都有箭头的交流链接表示有双向的交流（图 19 中的“协议 1”就是这样的一个例子）。一个交流链接还可以有一个标签或标识符（可省略的）来指明这个交流所必须遵循的协议规范。图 19 中的“协议 0”，“协议 1”和“协议 2”就是这样的标签。

[0377] 在本发明中，一个组合 DS 中的任何子 DS 都可以是简单的或组合的。甚至一个简单子 PS 中的 Ext-DS 或 Int-DS 也都可以是简单的或者组合的。因此一个包含框可以有多个子包含框，而且这些子包含框中的任何一个又都可以包含多个子子包含框。这些子子包含框中有些是顺序的，因此，它们之间连着迁移箭头。而有些子子包含框是并行的，因此没有与任何其他子子包含框相连。图 20 表示了一个组合 PS 的例子“PS0”。组合 PS“PS0”2000 包含一个子 PS“s0”2001，子 PS“s0”又有一个组合的 Ext-DS 2002 和一个组合的 Int-DS2006。组合的 Ext-DS 2002 还包含了两个顺序的 Ext-DS（即“Ext-DS s0”2003 和“Ext-DS s1”2004）及一个并行的 Ext-DS“Ext-DS p0”2005。组合的 Int-DS 2006 包含两个顺序的 Int-DS2007 和 2008，以及一个并行的 Int-DS 2009。

[0378] 用图形语言表示 DS,Ext-DS,PS 和 Int-DS 有以下好处 :(为了简捷起见,这里用术语“系统”代表 DS,Ext-DS,PS 或 Int-DS)

[0379] ●能够很清晰地同时展现一个组合系统内的顺序组件和并行组件。

[0380] ●包含框可以清晰地表明当一个父系统变成非活跃状态或者被销毁时,它所有的组件都会变成非活跃状态或者被销毁。

[0381] ●结束点可以清楚地表明一个组合子系统的内部操作会如何影响它的父系统改变当前阶段的方式。

[0382] ●一个组合系统里只有关键的单元才被画出来。而阶段机,阶段和并行单元都没有画在图中,因为它们在图中对人们理解系统的行为不是很重要。这样做会极大地简化了表明一个系统行为的图形。

[0383] ● Ext-DS, PS 和 Int-DS 图形的一致性会让人们更容易学习和理解一个复杂行为的描述。

[0384] 请注意,上文的描述只给出了一种描述 DS, Ext-DS, PS, Int-DS 及其组件的方法。然而根据在这里披露的技术和给出的指导,一个本领域的技术人员可以很容易明白,只要保持以下关键点不变,可以用各种图形表示方法来表示与本发明一致的机制 :

[0385] ●一个组合的 DS 用一个包含框表示,包含框里包含了它的组件,比如顺序的子 DS,并行的子 DS,初始阶段,空阶段,结束点,迁移箭头,生成箭头以及交流链接等。

[0386] ●组合 DS 的包含框里的每一种组件必须要能够在视觉上彼此区分 ;

[0387] ●一个简单 PS 的包含框里包含一个 Ext-DS 组件和一个 Int-DS 组件,并且它们能在视觉上彼此区分 ;

[0388] 预定义的事件类型

[0389] 有些系统需要在 PS 一变成活跃状态时就执行某一个 Int-DS 的动作。本发明支持一个预定义的“开始”事件,每当一个 PS 刚刚变成活跃状态时,这个事件就会被分发给这个 PS。图 21 中的一个组合 PS 范例 2100 包含一个用“开始”事件作为其 Ext-DS 2102 的简单的子 PS2101。当组合 PS2100 变成活跃状态时,它所有的并行子 PS 也会变成活跃状态。因此,与“开始”事件对应的 Int-DS 会在组合 PS 2100 变成活跃状态后立即被执行。

[0390] 有些系统需要在 PS 变成非活跃状态前一瞬间执行某一个 Int-DS 的动作。本发明支持一个预定义的“结束”事件,每当一个 PS 即将变成非活跃状态时,这个事件就会被分发给这个 PS。图 21 中的一个组合 PS 范例 2100 包含一个把“结束”事件作为其 Ext-DS 2105 的简单的子 PS 2104。当组合 PS 2100 变成非活跃状态时,它所有的并行子 PS 也会变成非活跃状态。因此,与“结束”事件对应的 Int-DS 会在这个组合 PS 2100 变成非活跃状态前一瞬间立即被执行。

[0391] 预定义的“开始”和“结束”事件的好处在于 :

[0392] ● UML 状态图需要提供额外的特殊机制才能实现一个状态的开始动作和结束动作,而有了“开始”和“结束”事件类型,用同一个简单的 PS 机制就可以用来实现同样的功能。因此用户在本发明中不需要学习额外的特殊机制。

[0393] ●在有些时候需要把某个特定的 Int-DS 与一个 Ext-DS 分离开来。比如,在根据协议推导参与者 PS 时 (这一部分本发明后面会有进一步描述),参与者 PS 最好能保持和协议同一样的组合结构。以图 39 中的 X-server 的参与者 PS 为例,它应该具有和图 38 所示

协议“协议 0”一样的组合结构。然而，如果尚未定义的 Ext-DS “??”3905 并没需要接收一个额外的向内的事件，Int-DS “答复 0”3906 就只得和 Ext-DS “请求 0”3902 绑定在一起，这样就破坏了同样的组合结构（如图 22 所示）。但是如果为尚未定义的 Ext-DS “??”加上了一个“开始”事件，则同样的组合结构可以得到保持（如图 23 所示）。

[0394] 实现中的问题

[0395] 一些 DS，Ext-DS，PS 和 Int-DS 的属性会在运行时刻频繁改变，然而有些静态的设计阶段的属性则在整个生命周期都不会变化。另外，同一种类型在运行时也可以有多个实例。因此把这些类或类型结构中的设计时属性与实例结构中的运行时属性分离开是有意义。

[0396] 在本专利的一种实施例中，一个简单 DS 的实例结构 2402 包含有一个指向它的类型结构 2400 的指针或引用 2403 以及一个结果组件 2404（如图 24 所示）。一个简单 DS 的类型结构 2400 包含一个行为类型的组件 2401。如图 24 所示，可以有多个简单 DS 的实例共享同一个简单 DS 的类型。图 25 用 UML 类图来表示同一个简单 DS 的类型结构和实例结构。

[0397] 在本专利的一种实施例中，一个简单 Ext-DS 的实例结构包含一个指向它的类型结构的指针或引用以及一个结果组件（如图 26 所示）。一个简单 Ext-DS 的类型结构包含一个事件类型的组件。如图 26 所示，可以有多个简单 Ext-DS 的实例共享同一个简单 Ext-DS 的类型。图 27 用 UML 类图来表示同一个简单 Ext-DS 的类型结构和实例结构。

[0398] 在本专利的一种实施例中，一个简单 PS 的实例结构包含一个指向它的类型结构的指针或引用以及一个结果组件（如图 28 所示）。一个简单 PS 的类型结构包含一个 Ext-DS 类型组件（它可以是简单的或者组合的）和一个 Int-DS 类型组件（它可以是简单的或者组合的）。如图 28 所示，可以有多个简单 PS 的实例共享同一个简单 PS 的类型。图 29 用 UML 类图来表示同一个简单 PS 的类型结构和实例结构。

[0399] 在本专利的一个实施例中，一个简单 Int-DS 的实例结构包含一个指向它的类型结构的指针或引用和一个结果组件（如图 30 所示）。一个简单 Int-DS 的类型结构包含一个动作类型组件，它可以是一段代码，一个函数（对 C 语言这样的面向函数程序语言而言）或者一个方法（对 JAVA 这样的面向对象程序语言而言），甚至可以是一个脚本（对 Linux Shell 这样的脚本语言而言）。如图 30 所示，可以有多个简单 Int-DS 的实例共享同一个简单 Int-DS 的类型。图 31 用 UML 类图来表示同一个简单 Int-DS 的类型结构和实例结构。

[0400] 组合 DS 类型和实例

[0401] 如图 32A 所示，模型 1 的组合 DS 的类型 3200 包含有若干个阶段机类型 3201，以及一个如图 32A 所示的结束情况集 3202 或者一个如图 32B 所示的结束函数 3218。如果定义了一个结束条件集，那么它至少包含一个结束情况，而且每一个结束情况都包含一对结束条件和结束代码，其中结束条件描述在何种情况下一个组合 DS 类型的实例会结束，而结束代码则用来标识对应的结束条件。如果定义了一个结束函数，这结束函数会返回一结束代码以标明在何种情况下一个组合 DS 类型的实例会结束。每个阶段机类型有一个静态实例数 3203（可省略的）以及一个或多个阶段类型 3204。每个阶段类型有一个子 DS 类型 3205，以及零个或多个迁移 3206。每个迁移有一个迁移条件 3207 和一个目标阶段 3208。每个子 DS 类型可以是简单 DS 类型或者组合 DS 类型。

[0402] 如果在一个阶段机类型中定义了静态实例数，它定义了这个阶段机类型的静态实

例个数。如果没有定义静态实例数，这个阶段机类型则默认仅有一个静态实例。

[0403] 请注意，每一个阶段机类型都可以是顺序的或者并行的阶段机。如前文所述，一个并行的阶段机类型只有一个阶段类型，如图 15A 和图 15B 所示 .

[0404] 一个模型 1 的组合 DS 的实例结构 3209 有一个对它的 DS 类型结构的引用 3210，若干个阶段机 3211，以及一个结果 3212。每个阶段机有一个对它的阶段机类型结构的引用 3213，若干个阶段 3214 以及一个当前阶段 3215。每个阶段有一个对它的阶段类型结构的引用 3216 和一个子 DS 实例结构 3217。这个子 DS 实例可以是简单的或者组合的。如果它是简单的，如图 24 中 2402 所示，那么它包含一个对它的简单 DS 类型 2400 的引用 2403，以及一个结果 2404 用于记录执行它的简单 DS 类型的行为 2401 的成果。如果这个子 DS 实例是组合的，则它的结构和图中 3209 一样。

[0405] 请注意，图 32A，32B 及 32C 中实例结构的阴影表示可以有多个组合 DS 的实例来共享同一个组合 DS 的类型。

[0406] 图 35 是用 UML 类图表示的同一个组合 DS 的类型结构和实例结构。

[0407] 作为一个模型 1 的组合 DS 的一个特例，一个模型 1 的组合 Ext-DS 也可以类似地分为类型结构和实例结构两部分。在用“Ext-DS”替代每一个“DS”以后，图 32A 或 32B 中所示的组合 DS 的类型结构、实例结构即变成一个模型 1 的组合 Ext-DS 的类型结构、实例结构。

[0408] 作为一个模型 1 的组合 DS 的特例，一个模型 1 的组合 PS 也可以类似地分为类型结构和实例结构两部分。因为一个模型 1 的组合 PS 的类型结构和组合 DS 的类型结构完全相同，在用“PS”替代每一个“DS”以后，图 32A 或 32B 中所示的即变成一个模型 1 的组合 PS 的类型。请注意，一个阶段类型中的子 PS 类型可以是简单的或者组合的。图 32C 表示一个包含一个简单 PS 类型 3220 的组合 PS 类型 3219。这个简单的 PS 类型包括一个 Ext-DS 类型用于描述待发生的外部事件和一个 Int-DS 类型用于描述为响应所发生的所述外部事件而执行的内部动作。

[0409] 在用“PS”替代每一个“DS”以后，图 32A 或 32B 所示的组合 DS 的实例即变成一个模型 1 的组合 PS 的实例。请注意，一个阶段中的子 PS 可以是简单的或者组合的。如图 32C 中表示，一个组合 PS 实例 3221 包含了一个简单 PS 实例 3222。这个简单的 PS 实例包括了一个 Ext-DS 实例和一个 Int-DS 实例。

[0410] 图 36 表示了如何用一个包含了一个默认结束情况集和一个阶段机类型“0”3601 的组合 PS 的类型结构 3600 来定义图 23 中的“X-server”参与者 PS 类型。阶段机类型“0”有一个静态实例数，和三个阶段类型。它们是“初始”3615，“0”3602 和“1”3605。它的静态实例数取值为 1，表明当这个组合 PS 生成时，会生成一个这种类型阶段机的实例。阶段类型“初始”3615 是一个带有迁移 3616 的初始阶段。迁移 3616 的迁移条件 3617 永远为真，它的目标阶段是类型“0”的阶段 3618。请注意，图 36 中没有画出空的顺序子 DS。阶段类型“0”3602 有一个迁移 3603 和一个子 PS 类型 3604。而子 PS 类型 3604 以“请求 0”作为它的 Ext-DS 类型，以“??”作为它的 Int-DS 类型。阶段类型“1”3605 把一个空阶段作为它的目标阶段 3606，并且在它的子 Ext-DS 类型 3607 中定义了一个预定义的“开始”事件类型。

[0411] 图 36 还表示了一个“X-server”参与者 PS 类型的实例 3608。这个实例有一个结

果，一个指向它的类型 3600 的引用 3609，和一个阶段机“0”3610。阶段机“0”3610 包含有一个指向它的阶段机类型“0”3601 的引用，一个当前阶段以及两个阶段，即阶段“0”3611 和阶段“1”3613。阶段“0”有一个指向它的阶段类型“0”3602 的引用以及一个子 PS 3612。子 PS 3612 包含了一个指向它的 PS 类型 3604 的引用及一个结果。

[0412] 同样地，作为一个模型 1 的组合 DS 的特例，一个模型 1 的组合 Int-DS 也可以类似地分为类型结构和实例结构两部分。在用“Int-DS”替代每一个“DS”以后，图 32A 或图 32B 所示的组合 DS 的类型结构、实例结构即变成一个模型 1 的组合 Int-DS 的类型结构、实例结构。

[0413] 尽管把类型信息从实例信息中抽象出来了，但只要在运行时刻能够访问到这些类型信息和实例信息，在前文所描述的那些执行函数都还是可以照用的，包括一个处于活跃状态的 Ext-DS 的事件检查函数，一个处于活跃状态的 PS 的反应函数，以及一个处于活跃状态的 Int-DS 的动作执行函数。

[0414] 尽管把实例结构与类型结构清晰地分割开有它的好处，但具体实现时也可以选择把二者联合在一起或者按某种方式进行混合。本发明并不会限制具体的实现方法。

[0415] 在本发明前面描述的那些执行函数可能需要在执行过程中保存一些临时的中间数据。在本发明的一种实施例中，除了这些定义在实例结构中的数据以外，一个 DS 组件的实例结构中还可以加上一些本地使用的数据。

[0416] 本发明包括了对一个应用软件的整个开发生命周期都有用处的方法和组件。在本发明的一种实施例中，可以用一个设计工具来设计一个基于 DS 组件的应用软件，然后用代码生成器来生成准备在运行平台上执行的机器可读的二进制代码。如图 33 所示，设计工具的产出是 DS 类型定义，而这个 DS 类型定义可以传送到代码生成器或者保存在存储设备中供代码生成器以后访问。代码生成器的产出是代表 DS 类型的一组二进制代码，这些二进制代码可以被一个运行平台装载到内存中去执行。这个运行平台也可以提供一些为实现基本动作或已存在的 Int-DS 的所需要的二进制代码。

[0417] 这个设计工具可以支持图形用户界面 (GUI) 以便设计者用前面描述的图形语言来定义 DS 组件。在设计流程结束后，设计工具会生成一组用户定义的 DS 类型或者 DS 类的详细说明，比如 PS 类型，Ext-DS 类型和 Int-DS 类型。在设计阶段，设计工具只会涉及类型信息而不需要涉及实例结构。

[0418] 一个代码生成器读入这些保存在存储设备上的 DS 类型详细说明，然后生成代表 DS 类型的二进制代码。这个代码生成器的输出可以保存在存储设备上，也可以直接传送给运行平台去执行。

[0419] 在执行过程中，运行平台把这些代表 DS 类型的二进制代码装载到内存中，生成本发明中描述的实例结构并引用它们的类型结构。这些 DS 组件的实例是根据 DS 类型详细说明中的信息来生成并进行初始化的。这些 DS 组件也能够根据本发明所描述的过程动态地变成活跃状态和非活跃状态以完成它们各自的功能。

[0420] 在本发明的一种实施例中，一个运行平台可以实现上面所述的动作执行函数以便处理每一个用 Int-DS 类型来定义的组合 Int-DS，而在另一种实施例中，每一个组合的 Int-DS 类型也可以被编译成一个底层、而更高效的执行代码，然后再在一个运行平台上运行。一种底层的代码可以是能够被虚拟机直接执行的字节码格式，比如可以被 JAVA 虚拟机

执行的 JAVA 字节码。还有另一种底层的代码可以是一种程序语言（比如 C/JAVA）的目标代码。在这种情况下，可以先为每个组合 Int-DS 类型生成相应的程序语言源代码，然后再编译成目标代码。

[0421] 本发明中描述的类型结构和实例结构是抽象的结构，也就是说它们并不依赖于某一种具体的实现。当实现在一个工具里，这些抽象的结构需要映射到工具所实现的各种具体结构上去。比如，这些 DS 组件可以按照一种具体的外部格式保存在存储设备中，而按照另一种具体的内部格式装载到内存中进行处理。这种具体的外部格式可以是下面任何一种：1）一种程序语言，比如 C 或者 JAVA；2）一种数据描述语言，比如 ASN.1(Abstract Syntax Notation One)；3）一种基于 XML 的语言；或者 4）一个可视化设计工具可以支持的任何表示方法。具体的内部格式则可以是一种数据单元的形式，而这种数据单元可以用数据描述语言，或者用基于 XML 的语言，或者用脚本语言，或者用像 C 或 JAVA 这样的程序语言，以及任何程序语言的二进制代码或机器码来表示。

[0422] 实现本发明所描述的抽象结构时，并没有必要一字不变地照搬抽象结构的定义。比如阶段机类型或并行单元类型中的“静态实例数”就可以采用一个标识的形式来指明当一个阶段机的父组合 DS 生成时，是否会静态地生成这个阶段机的静态实例。这个标识甚至可以用另一个变量的特殊字位来实现。另外一个例子是一个 DS 实例结构中的结果，它可以被实现为一个整数，或者一个包含各种结果信息的复杂的数据结构。简而言之，作为本发明的一种实施例的具体格式不需要完全参照本文所描述的抽象结构。

[0423] 尽管本发明用 PS，Ext-DS 和 Int-DS 来描述一个与其他系统有交流的系统的行为，但并不需要同时都用到 Ext-DS，PS 和 Int-DS，它们中的任何一个都可以被替换成使用现有的或者未来的行为描述技术，脚本语言或程序语言所定义的组件。

[0424] 在本发明的另外一种实施例中，只在设计阶段和代码生成阶段用到 DS 组件。如图 34 所示，一个应用系统可以基于 DS 组件来设计，然而一个代码生成器可以将输出的应用系统的 DS 类型用来为另一个不支持 DS 组件的运行平台生成代码。这个代码生成器只要遵循某种合适的方式把本发明描述的 DS 类型结构映射为运行平台所支持的另一种机制就行了。

[0425] 设计方法

[0426] 要设计一个分布式的、彼此有交流的多个应用系统是一个有挑战性的任务。如果我们分别地来设计两个交流的应用系统，要确保它们的行为完全协调是比较困难的。一个更好的办法是，先定义这些交流系统间的通讯协议，然后用一种系统化的、机械的方法把这些协议规范转换成每一方的行为规范，从而避免了设计过程中产生不一致的可能性。

[0427] 在本发明的一种实施例中，一个应用系统可以扮演多个角色，而且每个角色可能需要同时和多方进行交流。为了清楚起见，本发明用术语“角色 PS”来特指这个应用系统所扮演的角色的行为。对每一个协议而言都会有两个交流参与方。“参与者 PS”这个术语指的是一个交流参与者的那部分行为。因此一个系统中一般有多个“角色 PS”，而每个角色 PS 又会有多个参与者 PS，其中每个参与者 PS 对应了该角色所支持的一个协议。图 8 所示例子中“系统 0”800 和“系统 1”801，“系统 2”802 及“系统 3”803 之间有交流。“系统 0”扮演两个角色，即“角色 PS0”804 和“角色 PS1”805。其中“角色 PS0”会与“系统 1”和“系统 2”交流，而“角色 PS1”会与“系统 3”交流。

[0428] 为了说明这个理想的设计过程，我们使用了一个应用系统的例子“X”。图 37 表明系统“X”有一个角色 PS 同时与一个客户端和一个服务器端有交流。“X”内部有一个参与者 PS “X-server”通过协议“协议 0”和客户端进行交流，而另一个参与者 PS “X-client”通过协议“协议 1”和服务器端进行交流。遵循这个设计过程，我们首先定义协议“协议 0”和“协议 1”，然后我们把“协议 0”转换成参与者 PS “X-server”，而把“协议 1”转换成参与者 PS “X-client”。最后，我们把“X-server”和“X-client”合并成一个角色 PS “X”.。

[0429] 本发明最提倡的设计过程描述如下：

[0430] ●第一步，为一个角色可能与之交流的每一个对端系统都定义一个协议；这个协议是用一个 DS 类型来定义，它包含了该角色和一个对端系统间两个方向上可能发生的所有外部事件。

[0431] ●第二步，遵循一个转换方法，把第一步得到的每一个协议转换成该角色的一个参与者 PS；

[0432] ●第三步，遵循一个合并方法，把第二步得到的同一个角色的所有参与者 PS 合并成一个角色 PS；

[0433] ●最后一步，通过增加一些内部的交流和内部的主动式行为，让这个角色的行为规范变得完整。每一个这些内部交流都用一个 DS 类型来定义，而每一个这些内部主动式行为都用一个 Int-DS 类型来定义。

[0434] ●当设计有变化时，可以重复以上这几个步骤。

[0435] 在本发明中，一个协议是用一个 DS 类型来定义，它只描述了一个参与者要接收的向内事件和要发送的向外事件（其他与收发事件无关的内部活动都可以被忽略）。这个协议 DS 类型也可以被视为一个处于通讯双方中间，能接收双向事件的一个实体的行为。

[0436] 图 38 描述了用 DS 类型定义的协议例子“协议 0”和“协议 1”。“协议 0”规定客户端首先向 X-server 发送一个“请求 0”的事件，然后“X-server”回送一个“答复 0”事件。“协议 1”规定“X-client”首先向服务器发送一个“请求 1”事件，然后服务器回送一个“答复 1”事件。请注意，“→”和“←”符号是用来使得在图中更容易看出事件流的方向。

[0437] 把一个描述协议的 DS 类型转换为一个描述参与者的反应式行为的 PS 类型的转换方法说明如下：

[0438] ●对协议 DS 类型中描述每一个参与者要接收事件的向内的简单子 DS 类型，用一个简单的子 PS 类型来替代这个简单子 DS 类型，这个简单的子 PS 类型把这个向内的简单子 DS 类型作为它的 Ext-DS 类型，并包括一个尚待定义的 Int-DS 类型；

[0439] ●对协议 DS 类型中描述每一个参与者要发送事件的向外的简单子 DS 类型，用一个简单的子 PS 类型来替代这个简单子 DS 类型，这个简单的子 PS 类型包括一个尚待定义的 Ext-DS 类型，并把这个向外的简单子 DS 类型作为它的 Int-DS 类型；

[0440] 基于上面描述的转换方法，可以按照下面的步骤得到 X-server 的参与者 PS 3900（如图 39 所示）：

[0441] ●因为“请求 0”对“X-server”而言是一个向内的简单子 DS 类型，所以它被替换成一个简单的子 PS 类型 3901，这个简单的子 PS 类型 3901 把“请求 0”作为它的 Ext-DS 类型 3902，并包含一个尚待定义的 Int-DS 类型 3903（在图 39 中用“??”表示）。

[0442] ●因为“答复 0”对“X-server”而言是一个向外的简单子 DS 类型，所以它被替换

成了一个简单的子 PS 类型 3904，这个简单的子 PS 类型 3904 包括一个尚待定义的 Ext-DS 类型 3905（在图 39 中用“??”表示），并把这个向外的简单子 DS 类型“答复 0”作为它的 Int-DS 类型 3906。

[0443] 基于上面描述的转换方法，可以按照下面的步骤得到 X-client 的参与者 PS 4000（如图 40 所示）：

[0444] ●因为“请求 1”对“X-client”而言是一个向外的简单子 DS 类型，所以它被替换成了一个简单的子 PS 类型 4001，这个简单的子 PS 类型 4001 包括一个尚待定义的 Ext-DS 类型 4002，并把这个向外的简单子 DS 类型“请求 1”作为它的 Int-DS 类型 4003。

[0445] ●因为“答复 1”对“X-client”而言是一个向内的简单子 DS 类型，所以它被替换成一个简单的子 PS 类型 4004，这个简单的子 PS 类型 4004 把“答复 1”作为它的 Ext-DS 类型 4005，并包含一个尚待定义的 Int-DS 类型 4006。

[0446] 在本专利的一种实施例中，也可以不用图 38、39 和 40 所示的 DS 类型的图形语言而直接在 DS 类型的数据结构的基础上实现上述转换方法。

[0447] 把所得到的属于同一个角色的所有参与者 PS 并成一个角色 PS 的合并方法是先定义一个父 PS，然后把这些参与者 PS 都作为它的并行子 PS。图 42 表示出如何将 X-server 和 X-client 的参与者 PS 合并成一个角色 PS “X”。

[0448] 在本专利的一种实施例中，也可以不用图 42 所示的 PS 类型的图形语言而直接在参与者 PS 类型的数据结构的基础上实现这个合并方法。

[0449] 这样得到的角色 PS 一般是不完整的，因为会有很多尚待定义的 Ext-DS 和 Int-DS。下一步就是用内部的交流和行为来替换这些尚待定义的 Ext-DS 和 Int-DS。

[0450] 我们假定“X”系统的行为是：在收到“请求 0”以后，X-server 会完成部分的工作，然后通过发送“请求 1”给服务器，让服务器完成余下的工作。我们还假定在收到来自服务器的“答复 1”后，“x-client”会把服务器的工作结果通知”x-server”，然后由“x-server”把“答复 0”送回给客户端。

[0451] 为了达成这些内部交流的需要，X-server 可以发送一个事件“aa”给 X-client 以请求服务器完成余下的工作。然后 X-client 可以发送一个事件“bb”给 X-server 以通知服务器的工作结果。在替换了这些尚待定义的 Ext-DS 和 Int-DS 后，新的角色 PS 如图 43 所示。

[0452] 这个例子用了两个非常简单的内部协议（每个协议只有一个事件），比如“aa”和“bb”，来完成这项工作。本发明还支持在两个内部子 PS 间用一个组合的 DS 来定义复杂的协议。事实上，可以用同一个例子来举例说明这一点。图 44 描述的是同一个例子，只是内部协议用一个组合的 DS “协议 2”（如图 41 所示）来定义。

[0453] 使用本发明描述的各种设计方法的好处在于：

[0454] ●通过机械地由一个外部行为推导出内部的反应式行为，本发明可以确保协议和通讯各方是一致的。

[0455] ●因为 Ext-DS，Int-DS 和 PS 都是 DS 的特例，并且共享同样的组合结构，所以很容易识别哪些是在机械式推导出的反应式行为之上增加的内部行为。

[0456] ●通过从外部行为机械地推导出内部反应式行为，可以极大地减少描述复杂行为的工作量。

[0457] 上述的这些设计方法可以扩展到其他基于节点和链接的建模语言（比如UML活动图，BPMN，XPDL，WS-CDL 和 BPEL 或者这些技术的结合）。相对应于本发明中的一个组合 DS 的等同物，在 UML 活动图中称为一个活动 (activity)，在 BPMN，XPDL 和 BPEL 中称为一个流程 (process)，在 WS-CDL 称为一个流程编排 (choreography)。为了简化描述，我们将这些组合 DS 的等同物都称作组合活动。相对应于本发明中的一个简单 DS 的等同物，在 UML 活动图中称为一个动作 (action)，在 BPMN 和 XPDL 中称为一个任务 (task)，在 BPEL 和 WS-CDL 中称为一个活动 (activity)。它们都将被称作简单活动。另外，我们用术语“活动”来泛称一个简单的或者组合的活动。

[0458] 一个简单的 PS 可以映射到这些建模语言中的一个反应式的组合活动，这个组合活动包含一个外部活动，用于描述待接收的外部事件，以及一个内部活动，用于描述为响应所发生的这个外部活动而执行的内部动作。因为一个组合的 PS 和一个组合的 DS 有相同的结构，一个组合的 PS 可以被直接映射到一个组合活动。

[0459] 这些设计方法中用到的其他一些组件也被扩展得更具通用性，因此它们也可以用于其他行为建模语言。一个尚待定义的 Ext-DS 被扩展成一个尚待定义的外部活动，用于描述要接收某些尚待定义的外部事件的活动。而一个尚待定义的 Int-DS 被扩展成一个尚待定义的内部活动，用于描述要执行某些尚待定义的内部动作的活动。

[0460] 使用了以上这些通用性的构件，以上这些设计方法可以扩展成通用性的方法使用其他行为语言来做设计，它的的过程可以描述如下：

[0461] ●第一步是用 UML 活动图，BPMN，XPDL，WS-CDL 或者 BPEL 的组合活动来描述一个角色所支持的每一个协议。

[0462] ●第二步是使用通用性的转换方法，将每一个描述协议的组合活动转换成一个参与者的组合活动。

[0463] ●第三步是把那些属于同一个角色的参与者组合活动合并成一个用 UML 活动图，BPMN，XPDL，WS-CDL 或者 BPEL 表示的角色组合活动。

[0464] ●第四步是把上一步得到的角色组合活动中尚待定义的外部活动和尚待定义的内部活动替换成内部的交流和行为，从而使这个角色的规范变得完整。

[0465] 一个协议可以用 UML 活动图，BPMN，XPDL，WS-CDL 或者 BPEL 描述成一个组合活动，它包含了一系列只描述交流双方之间的双向的外部事件（其他与收发事件无关的活动可以被忽略）的简单活动。因为这些建模语言的每一种都有用于接收外部事件的简单活动，所以一个协议可以用一些接收事件的简单活动来描述。换言之，这个协议组合活动可以看作是被用来描述一个处于通信双方中间，能接收两个方向事件的一个实体的行为。根据这个方法，一个协议可以用 UML 活动图的组合活动来定义，并且用“AcceptEventAction”来接收双向的外部事件。比如，图 38 中用 DS 类型定义的协议“协议 0”和“协议 1”可以用 UML 活动图表示在图 45 中。

[0466] 转换成一个参与者组合活动

[0467] 可以用这个通用性的转换方法把一个组合活动从一个协议规范转换成一个参与者的行为规范。这个方法包括：

[0468] ●对所述组合活动中每一个描述参与者待接收事件的向内的简单活动，把这个向内的简单活动替换成一个反应式的组合活动，而这个反应式的组合活动把所述向内的简单

活动作为它的外部活动，这个外部活动后面跟着一个尚待定义的内部活动，以描述为响应所发生的向内的简单活动而执行的活动。

[0469] ●对所述组合活动中每一个描述参与者待发送事件的向外的简单活动，把这个向外的简单活动替换成一个反应式组合活动，这个反应式组合活动包含一个尚待定义的外部活动用于描述要接收的事件，然后跟着这个尚待定义的外部活动后面，用这个向外的简单活动作为它的内部活动，以响应所发生的这个尚待定义的外部活动。

[0470] 图 46 是用 UML 活动图表示的由图 45 中“协议 0”4500 转换得到的“X-server”的参与者组合活动 4600。简单活动“请求 0”4501 是一个向内的待接收事件，因此，它被替换成一个反应式的组合活动 4601，而这个反应式的组合活动包含一个外部活动用以接收向内的“请求 0”事件 4602，以及跟在这个外部活动后面的尚待定义的内部活动 4603 用以响应“请求 0”事件。这个内部活动可以用 UML 活动图中的一个动作 (action) 或一个活动 (Activity) 来定义。“协议 0”4500 中的简单活动“答复 0”4502 是一个向外的待发送事件，因此，它被替换成一个反应式的组合活动 4604，而这个反应式的组合活动包含一个尚待定义的外部活动 4605 用以接收尚待定义的向内事件，以及跟在这个外部活动后面发送向外的“答复 0”事件 4606 的内部活动。这些外部活动和内部活动中的每一个都可以用一个动作或一个 UML 活动图中的活动来定义。

[0471] 图 47 是用 UML 活动图表示的由图 45 中“协议 1”4503 转换得到的“X-client”的参与者组合活动 4700。简单活动“请求 1”4504 是一个向外的待发送事件，因此，它被替换成一个反应式的组合活动 4701，而这个反应式的组合活动包含一个尚待定义的外部活动 4702 用以接收尚待定义的向内事件，以及跟在这个外部活动后面发送向外的“请求 1”事件 4703 的内部活动用以响应尚待定义的向内事件“??”。这些外部活动和内部活动中的每一个都可以用 UML 活动图中的一个动作或一个活动来定义。“协议 1”4503 中的简单活动“答复 1”4505 是一个向内的待接收事件，因此，它被替换成一个反应式的组合活动 4704，而这个反应式的组合活动包含一个外部活动 4705 用以接收向内的“答复 1”事件，以及跟在这个外部活动后面的尚待定义的内部活动 4706。这个外部活动可以用 UML 活动图中的一个动作或一个活动来定义。

[0472] 当使用基于节点和链接的建模语言时，可用一个通用性的合并方法把以上得到的属于同一个角色的所有参与者组合活动合并成一个描述单一角色行为的组合活动，具体步骤描述如下：

[0473] ●增加一个父组合活动，然后把所有这些得到的参与者组合活动作为它的并行子活动。这可以通过增加一个分叉节点和一个汇合节点来实现，而这个分叉节点会有一个链接指向每一个参与者组合活动，这个汇合节点则是每一个参与者组合活动的目标节点。

[0474] 图 48 用 UML 活动图表示如何把得到的参与者组合活动“X-server”和“X-client”合并成“X”角色组合活动。请注意，分叉节点 4800 和汇合节点 4801 用 UML 活动图中的竖线表示。

[0475] 当把这个通用性的设计过程用于 BPMN 建模语言时，具体过程描述如下：

[0476] ●用 BPMN 定义协议组合活动（如图 49 所示），它只包含接收双向外部事件的事件接收任务。

[0477] ●作如下修改把它转换成一个参与者组合活动（如图 50 和图 51 所示）：

[0478] ●为每一个从参与者角度看是接收向内事件的“接收 (Receive)”任务 4901，把“接收”任务替换成一个反应式流程 5001，这个反应式流程把该“接收”任务 5003 作为它的外部活动，而后面跟着一个尚待定义的内部活动 5004（是 BPMN 的一个子流程 (Sub-process) 或任务 (Task)）作为响应所发生的向内事件时执行的内部活动。

[0479] ●为组合活动中每一个从交流对端系统看来是接收事件，而从参与者角度看来是发送事件的“接收”任务 4902，把“接收”任务替换成一个反应式流程 5002，这个反应式流程包含一个尚待定义的外部活动 5005（是 BPMN 的一个子流程或任务），用来描述待接收的尚待定义的外部事件，后面跟着一个发送这个事件的任务作为响应所发生的尚待定义的外部活动时执行的内部活动 5006。

[0480] ●为了把这些参与者组合活动合并成一个角色组合活动，我们定义一个父流程，它用一对并行的“与分流”网关 (AND-Split Gateway)5200 和“与汇合”网关 (AND-join Gateway)5201（如图 52 所示）或者一个并行框 (Parallel box)5300（如图 53 所示）把这些参与者组合活动作为它的并行子流程整合在一起。与一个组合 PS 的并行子 PS 类似，当父流程开始运行时，这些子流程就会开始并行运行。

[0481] 随后可以通过用内部交流和行为来替换尚待定义的外部活动和尚待定义的内部活动让所得到的流程定义变得完整。

[0482] 当把这个通用性的设计过程用于 BPEL 建模语言时，具体的过程描述如下：

[0483] ●用 BPEL 定义一个协议组合活动，它只包含用 BPEL “接收 (receive)”或“拾取 (pick)”活动来定义的接收双向外部事件的事件接收任务。

[0484] ●如果用了“拾取 (pick)”活动来定义一个协议，那么它的 <onMessage> 结构应该把这个事件的名称描述在它的“operation”中，而且把它的活动定义为“Empty”。举个例子，“协议 0”的 BPEL 伪代码应该是下面这样的：

[0485]

```
<sequence ...>
  <pick ...>
    <onMessage ... operation="Request0" ...>
    ...
    empty
    </onMessage>
  </pick>
  <pick ...>
    <onMessage ... operation="Reply0" ...>
    ...
    empty
    </onMessage>
  </pick>
</sequence>
```

[0486] ●如果用了“接收 (receive)”活动来定义一个协议，应该把这个事件的名称描述在它的“operation”中。举个例子，“协议 1”的 BPEL 伪代码应该是下面这样的：

[0487]

```
<sequence ...>
    <receive ... operation="Request1" ...>
    ...
```

[0488]

```
    </receive>
    <receive ... operation="Reply1" ...>
    ...
    </receive>
</sequence>
```

[0489] ●请注意，尽管在本例中用到了“sequence”来定义顺序的活动，BPEL 中还可以用“flow”和“link”来定义顺序的活动。

[0490] ●作以下修改来得到一个参与者组合活动：为协议组合活动中的每一个“接收 (receive)”或“拾取 (pick)” 活动执行下面的步骤：

[0491] ●如果这个活动描述的是参与者要接收的向内的事件，把它替换成一个反应式组合活动，这反应式组合活动把这个向内的活动作为它的外部活动，后面跟着一个尚待定义的内部活动 (BPEL 中的一个子活动) 作为响应所发生的向内活动时执行的内部活动。

[0492] ●对“拾取 (pick)”活动，在 <onMessage> 结构中的事件应该由“empty”变成一个尚待定义的内部活动“InternalActivity ??”，并且

[0493] ●对“接收 (receive)”活动，它后面应该跟着一个在“sequence”里面尚待定义的内部活动“InternalActivity ??”。

[0494] ●如果这个事件是向外的，则定义一个反应式组合活动，这反应式组合活动包含一个尚待定义的外部事件用来描述要接收的事件，后面跟着一个发送这个向外事件的活动，作为响应所发生的尚待定义的外部事件时执行的内部活动。

[0495] ●如果用了“拾取 (pick)”活动，一个尚待定义的外部活动的 <onMessage> 结构应该等待一个尚待定义的向内事件“??”，而它的内部活动则应该包含这个向外的事件。

[0496] ●如果用了“接收 (receive)”活动，尚待定义的外部活动应该等待一个尚待定义的事件，而且后面跟着一个在“sequence”里面发送这个向外事件的内部活动。

[0497] ●比如，使用″拾取 (pick)″活动的 X-server 参与者组合活动，用 BPEL 伪代码应该如下所示：

[0498]

```
<sequence ...>
  <pick ...>
    <onMessage ... operation="Request0" ...>
      ...
```

[0499]

```
        InternalActivity??
    </onMessage>
  </pick>
  <pick ...>
    <onMessage ... operation="??" ...>
      ...
      <invoke ... operation="Reply0" ...
      </invoke>
    </onMessage>
  </pick>
</sequence>
```

[0500] ●又比如，使用″接收 (receive)″活动的 X-client 参与者组合活动，用 BPEL 伪代码应该如下所示：

[0501]

```
<sequence ...>
    <receive ... operation="??" ...>
        ...
    </receive>
    <invoke ... operation="Request1" ...
    </invoke>
    <receive ... operation="Reply1" ...>
        ...
    </receive>
    InternalActivity??
</sequence>
```

[0502] ●为了定义一个角色组合活动，我们增加一个父流程，通过用“flow”结构把这些参与者组合活动作为这个父流程的并行子流程进行合并。

[0503] ●举个例子，这个合并后的角色组合活动用 BPEL 伪代码可以表示如下：

[0504]

```
<flow>
    <sequence ...>
        <pick ...>
            <onMessage ... operation="Request0" ...>
                ...
```

[0505]

```
                    InternalActivity??
                </onMessage>
            </pick>
            <pick ...>
                <onMessage ... operation="??" ...>
                    ...
                    <invoke ... operation="Reply0" ...
                    </invoke>
                </onMessage>
            </pick>
        </sequence>
        <sequence ...>
            <receive ... operation="??" ...>
                ...
            </receive>
            <invoke ... operation="Request1" ...
            </invoke>
            <receive ... operation="Reply1" ...>
                ...
            </receive>
            InternalActivity??
        </sequence>
    </flow>
```

[0506] 随后可以通过用内部的交流和行为来替换那些尚待定义的外部活动和尚待定义的内部活动让得到的流程定义变得完整。

[0507] 请注意，上述的实施例中实用了“operation ＝”?? ””来表示尚待定义的外部活动，以及“内部活动?? ”来表示尚待定义的内部活动。每一个尚待定义的外部活动和尚待定义的内部活动可以在其他实施例中用其他方式来表示，只要它们能够与正常的活动区分开。

[0508] 尽管上面所述的通用性设计方法的实施例是基于 BPEL 来描述一个协议组合活动和一个参与者组合活动，但是类似的方法可以被本领域的技术人员用于任何基于 XML 并且支持“顺序 (sequence)”活动的行为描述语言，比如 XPDL 和 WS-CDL。一般而言，上述的一个反应式组合活动可以定义成一个顺序活动，而这个顺序活动包含一个等待向内事件的外部活动以及一个为响应所发生的这个向内事件而执行的内部活动。下面的 XML 表示一个用

顺序活动来定义的反应式组合活动。

[0509]

<sequence>

ExternalActivity

InternalActivity

</sequence>

[0510] 对于一个用 WS-CDL 定义的反应式组合活动，它的外部活动可以是用一个 WS-CDL 的交流活动 (Interaction activity) 来描述要等待的向内事件，而它的内部活动可以是一个用来描述一些待执行动作的活动。尽管 WS-CDL 目前还不支持内部活动，但可以想像得到，WS-CDL 是可以被扩展来支持内部的活动的。本发明描述的这些通用性的设计方法可以被用于这样一个扩展了的行为描述语言。

[0511] 生成一个参与者状态机

[0512] 当把这个通用性的设计过程用于一个状态机建模语言（比如状态图），具体的过程描述如下：

[0513] ●把每一个协议定义为一个状态机 (State machine)（如图 54 中的“协议 0”5400 和“协议 1”5403)，这个状态机只把向内或向外的事件定义为它的状态迁移的事件或者触发器 (Trigger)，比如“协议 0”中的“请求 0”5401 和“答复 0”5402，以及“协议 1”中的“请求 1”5404 和“答复 1”5405，但没有为这个状态迁移定义任何动作或迁移效果 (Effect)。一个协议状态机可以被视为是用来定义一个处于通讯双方之间，能接收两个方向的事件的一个实体的行为。每一个像这样的事件都可被定义为它所对应的状态迁移中的一个触发器。

[0514] ●按如下步骤修改每一个协议状态机来转换得到一个参与者状态机（如图 55 和图 56 所示）：

[0515] ●对这个协议状态机中的每一个迁移，如果它的触发器中包含一个向内的事件（从参与者角度来看），则在这个迁移上增加一个尚待定义的迁移效果（如图 55 中的“??”5501 和图 56 中的“??”5603 所示）。

[0516] ●对这个协议状态机中的每一个迁移，如果它的触发器中包含一个向外的事件（从参与者角度来看），则在这个迁移上增加一个迁移效果来发送这个向外事件。然后把这个迁移的触发器改成等待一个尚待定义的向内事件（如图 56 中的“??”所示）。

[0517] 为了生成一个角色状态机，一个父状态可以被定义来把所有这些参与者状态机合并到一个单一的角色状态机。这个父状态把所有这些得到的参与者状态机都当作它的正交区域 (orthogonal regions)。如图 57 所示，参与者状态机“X-server”5701 和“X-client”5702 变成合并在父状态“X”5700 内的两个正交区域。请注意，状态机中的正交区域类似于一个组合 PS 中的并行子 PS。

[0518] 随后可以通过用内部的交流和行为替换那些尚待定义的触发器和迁移效果以使得到的角色状态机定义变得完整。

[0519] 尽管图 54，55，56 和 57 是用状态图 (Statechart) 图画来表示的，它们也可以被替换成其他的状态机 (state machine) 图画而不会失去本发明的精髓。

[0520] 有时候，一个设计者需要定义一个需要和一个已存在应用系统进行交流的新的应用系统的行为。因为不太可能改变已存在应用系统的行为，所以需要根据已存在应用系统

的行为来推导出已存在应用系统所支持的协议。然后应用上述转换方法来生成新应用系统的参与者组合活动。这样得到的参与者组合活动就可以和已存在应用系统正确地进行交流。

[0521] 生成一个协议组合活动

[0522] 下面说明如何从一个描述参与者行为的参与者组合活动来生成一个描述两个参与者之间的协议的协议组合活动的方法：

[0523] ●把所述参与者组合活动复制为一个协议组合活动；

[0524] ●把所述协议组合活动中既不接收向内事件也不发送向外事件的各个简单活动改变为空活动，但是保持原有向外的链接不变。

[0525] 下面用 UML 活动图作为行为建模语言来举例说明如何应用这个方法。图 46 是用 UML 活动图表示的“X-server”参与者组合活动。遵循上述的协议生成方法可以推导出一个协议组合活动。

[0526] ●把图 46 中的“X-server”参与者组合活动 4600 复制一份为图 58 中协议组合活动 5800；

[0527] ●因为简单活动“??”4603 既不接收向内事件也不发送向外事件，所以把它变成一个空活动 5803；

[0528] ●因为简单活动”??”4605 既不接收向内事件也不发送向外事件，所以把它变成一个空活动 5805；

[0529] 请注意，除了不包含任何要执行的实际动作以外，这个空活动和一个普通节点是一样的。如果用 BPEL 来描述一个组合活动，则可以用一个“empty”活动来表示空活动。如果用 WS-CDL 来描述一个组合活动，则可以用一个“noAction”来表示一个空活动。如果一个组合活动是用不支持空活动的行为建模语言来表示的，比如 BPMN，XPDL 和 UML 活动图，则需要为这个行为建模语言增加一个特殊的空活动。

[0530] 以上所生成的协议组合活动可能包含了许多空活动。这些空活动可以根据以下规则加以清除：

[0531] ●对每一个空活动，可以将它的目标活动和源活动直接相连，从而把空活动删除；

[0532] ●如果一对分叉、汇合节点之间没有链接，或者只剩下一个链接，那么这对分叉汇合节点可以删除。

[0533] ●如果一个组合活动只剩下一个子活动，那么这个组合活动可以用这个剩下的子活动来替代。

[0534] 在清除完这些空活动后，图 58 中的协议就变成了图 45 中的“协议 0”4500。

[0535] 实现上文这些通用性设计方法的一个设计工具或者一个运行平台需要用一种具体的形式来表现所述的通用性组件，比如尚待定义的外部活动和尚待定义的内部活动，协议组合活动和参与者组合活动。一个具体的形式可以是下面的任何一种：

[0536] 1) 一种程序语言，比如 C 和 Java ;2) 一种数据描述语言，比如 ASN.1(Abstract Syntax Notation One) ;3) 一种基于 XML 的语言，比如 XPDL，BPEL 和 WS-CDL ;或 4) 一种图形语言，比如 BPMN，UML 活动图，或该设计工具可以支持的任何专有表示方法。

[0537] 在这些设计方法的一种实施例中，可以用同一种具体形式来表示协议组合活动和

参与者组合活动。而在另一种实施例中，可以用一种具体形式来表示协议组合活动，而采取另一种具体形式来表示参与者组合活动。

[0538] 结论

[0539] 本发明包括了对一个应用软件的整个开发生命周期都有用处的方法和组件。

[0540] 在本发明的一种实施例中，本发明所描述的 DS 组件和方法可以用于设计阶段，代码生成阶段以及执行阶段。而在另一种实施例中，这些 DS 组件和方法会用在设计阶段和代码生成阶段。还有一种实施例中，这些 DS 组件和方法仅被设计工具使用。

[0541] 还有一种实施例中可以把本发明所描述的通用性的方法和图形语言与 UML 状态图，UML 活动图，BPMN，XPDL 和 BPEL 等建模语言结合起来用在设计阶段。如图 59 所示，一个设计工具可以通过使用这些通用性的方法和指导来描述一个应用系统的内部行为和外部行为，从而支持一个应用系统设计流程。本发明描述的全部或部分图形语言可以被一个设计工具采用以方便设计流程。然而这样设计出来的行为是可以用另外一种机制来描述的，比如 UML 状态图，UML 活动图，BPMN，XPDL 和 BPEL。然后它可以经过一个代码生成器来生成二进制代码，再被运行平台装载执行。

[0542] 本发明描述的图形语言是很灵活而且模块化的，一个设计工具不必支持全部的组件。因此一个设计工具可以只选用本发明所描述的组件的一部分来用于各种目的。在下面三幅图中表示了一些这样的实施例。

[0543] 图 60 中的包含框 6000 代表了一个应用系统，而该应用系统的动态行为是用一个包含了 3 个子包含框 6001，6002 和 6003 的组合活动来描述的。其中两个子包含框 6001，6002 代表顺序的子活动，而子包含框 6003 代表一个并行子活动。有一个迁移箭头从子包含框 6001 指向子包含框 6002，而并行子活动 6003 则没有任何迁移箭头。那两个顺序子活动中的一个是一个反应式组合活动 6001，6001 包含一个事件子子包含框 6004 作为它的外部活动以描述待发生的外部事件，以及一个动作子子包含框 6005 作为它的内部活动以描述响应所发生的外部事件时执行的内部活动。一个双向的交流链接 6006 代表了顺序子活动 6001 和并行子活动 6003 之间的一组交流。这个交流协议用标识符“协议 0”来表示。附注 :这个交流协议的 DS 在此图中并没有显示出来。

[0544] 图 61 中的一个包含框 6100 代表一个应用系统，而该应用系统的行为是用一个包含了三个子包含框 6101，6102 和 6105 的组合活动来描述，每一个子包含框代表了一个子活动。一个双向的交流链接 6106 代表了子包含框 6101 和 6105 之间的一组交流。这个交流链接用一个标识符“协议 0”来表明这个交流集所要遵循的协议规范。组合活动 6100 有两个结束点，其中的一个结束点 6103 有一个没有显示出的默认的结束条件。另一个结束点 6104 有一个结束条件“迁移条件 1”。

[0545] 图 62 中的一个包含框 6200 代表一个应用系统，而该应用系统的行为是用一个组合活动 6200 来描述的，它有两个顺序的子活动 6201 和 6202，一个并行的子活动 6205，以及一个在顺序子活动 6201 和并行子活动 6205 之间的交流链接 6206。有一个迁移箭头 6208 从源子包含框 6201 指向目标子包含框 6202。还有一个生成箭头 6207 指向一个在运行时可以被顺序子活动 6202 生成的动态组件 6205。一个子包含框 6201 则代表了一个反应式组合活动，这个反应式组合活动包含一个事件子子包含框 6209 用来描述待发生的外部事件和一个动作子子包含框 6210 用来描述为响应发生的所述外部事件而执行的内部活动。

[0546] 在本发明的另外一种实施例中，可以采用有向链接，而不用图 60 中的分割线来区别一个反应式组合活动中的外部活动和内部活动。图 63 表示了和图 60 同样的组合活动例子，只是在反应式组合活动 6300 内部多了一个连接外部活动 6301 和内部活动 6302 的有向链接 6303。

[0547] 除了使用可视化的设计工具，也可以在本发明的基础上用其他工具来生成用户定义的 DS 类型。比如可以使用计算机辅助的软件工程工具 (CASE)。一个 CASE 工具可以在整个开发方法学中综合了本发明的功能以支持生成和编辑用户定义的 DS 组件。

[0548] 一个 CASE 工具也可以为用户定义的 DS 类型提供分析和验证功能，而另一个 CASE 工具更可以通过提供一个 DS 类型的分类和目录以及支持对 DS 类型的查询和匹配来帮助重复使用用户已定义的 DS 类型。

[0549] 其他的 CASE 工具可以对基于 DS 组件的应用程序进行模拟，甚至可以提供自动测试功能，而测试用例则可以根据 DS 类型的定义自动生成。

[0550] 在本发明的另一种实施例中可以用一个集成开发环境 (IDE) 来支持基于本发明的软件开发，包括软件设计，编译和执行。特别是，执行时刻可以通过带有软件库函数的运行平台来支持 DS 组件。

[0551] 类似于一个运行平台，本专利还有一种实施例是在一个软件的虚拟机里或者一个完全基于硬件和固件的计算设备上支持 DS 组件和它的执行函数。

[0552] 另一方面，并不是所有的 DS 组件都需要被自动化或者通过软件来执行。本发明的另外一种实施例是一个设计者通过一个设计工具用 DS 的组件或组合活动来定义人的行为，业务流程或者是行动计划。这些 DS 组件或组合活动中的一部分可以是要人去完成的工作。

[0553] 尽管上面的描述都带有某些特性，但这不能被看作是对本发明范围的限制，而只能看作是本发明的一些实施例。根据本文所披露的信息和所指导的内容，一个本领域内的技术人员应该可以意识到在本发明的范围内还可以有很多其他实施例和进一步的衍生例。

[0554] 本发明的应用将是非常广泛的，因为大多数解决现实世界问题的系统都可以看作为一个反应式的系统。比如，在一个 3 层架构中，数据库管理系统可以看作一个对数据库管理系统的上一层服务器的输入做出反应的系统。而这些服务器又可以看作对这些服务器的上一层输入做出反应的系统。表示层 (presentation tier) 则可以看作对人类的输入做出反应的系统。换言之，几乎所有的计算系统都可以用本发明描述的 DS，PS，Ext-DS 和 Int-DS 来建模，并获益于它们所具有的先进特性。

[0555] 本发明还可以用来为高度复杂的通讯协议进行建模。尤其是利用一个组合 DS 的特性，一个协议不仅可以支持单个会话，还可以支持多个并发的、嵌套的会话。

[0556] 本发明还可以用来为人机界面建模。一个人可以同时处理很多件事情。为了和人类的这种“多进程”能力相称，一个基于图形用户界面的应用可以用一个组合的 DS 来定义它与人的界面，而用一个组合的 PS 来定义它的反应式行为。

[0557] 因为本发明所具备的高度完善的建模能力，它甚至可以用来对人类行为或者组织行为进行建模。比如，一个组织的整个业务流程可以用一个组合的 PS 来建模，这个组合的 PS 包含有多个层次的子 PS 来实现部门，团队和雇员的行为。计划流程，决策流程以及解决问题的流程也可以用 DS，PS，Ext-DS 和 Int-DS 来建模。

[0558] 一个通用的业务运行环境，比如一个业务流程管理平台，一个工作流引擎或者一个通用的业务运营框架，也可以通过承载各种业务流程和活动的 DS，PS，Ext-DS 和 Int-DS 来实现。

[0559] 有些现实世界的东西，它们具有多个状态，比如机器，顾客记录，订单项，也可以用 DS 或者 DS 的扩展来建模。

[0560] 用本发明的方法来建模并设计各种应用系统，比如上文描述的那些系统，可以定义出 DS，PS，Ext-DS 和 Int-DS 的规范。这些规范的一部分也可以通过其他途径来得到。不管是采用哪种途径，所得到的这些应用系统的 DS，PS，Ext-DS 和 Int-DS 规范可以由一台实施本发明的设备或机器，比如一台计算机来执行。

[0561] 一般而言，本发明可能以方法的形式实施或者以执行这些方法的设备的形式来实施。本发明也可以通过有形媒体（比如磁盘，光盘，磁盘驱动器或者任何其它机器可读的存储介质）上的程序代码来实施。当这些程序代码被一个设备，比如计算机，装载并执行时，这个设备就成了一个实施本发明的设备。本发明也可以通过在某些传输介质（比如电缆，光纤或者电磁辐射）来传输程序代码而得到实施。当这些程序代码被一个设备或机器，比如计算机，装载并执行时，这个机器就成了一个实施本发明的设备。

[0562] 因此，可以看到本发明所描述的 DS 组件以及相关的方法和过程是定义系统的行为和通讯协议的非常强大的方法和技术。特别是它对现有技术，比如 BPMN，BPEL，XPDL，WS-CDL 或 UML 在以下几个主要方面都进行了扩展并加以改良。

[0563] 第一个方面的改良在于 DS 模型反映了我们所在宇宙中的时间维度和空间维度，从而在现有技术所描述的一堆节点和链接的基础上增加了重要的含义和结构。第二个方面的改良是通过在图画中使用包含顺序组件与并行组件的包含框，省去了各种分叉节点和汇合节点以及链接。第三个方面的改良是使用结果与结束情况为组合 DS 的结束提供了非常强大而且灵活的控制机制。第四个方面的改良是使用 DS 的特例，比如 Ext-DS，PS 和 Int-DS 来以一种一致的方式分别描述系统的外部行为，系统的反应式行为和系统的内部行为。第五个方面的改良是可以描述多个系统间的复杂的交流行为。第六方面的改良是提供了系统化的方法从协议导出角色 PS，从而确保了相关规范的一致性。

[0564] 应该可以理解的是，在本文中描述的例子和实施例都仅仅是为了说明本发明，对本领域的技术人员来说还会出现许多修正，但这些修正都应包含在本专利申请以及权利要求的范围以内。进一步可以理解的是，本领域的技术人员可以在不偏离权利要求中所表达的本发明的原则和范围的条件下，出于阐释本发明的本质的目的，对一些细节，材料或者编排做出修改。在此不管出于何种目的而引用的出版物，专利或专利申请都会被看作对相关内容的完整引用。

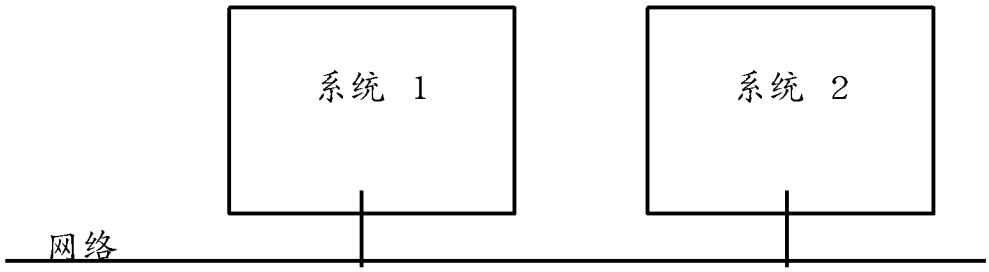

系统 1
系统 2
网络

图 1

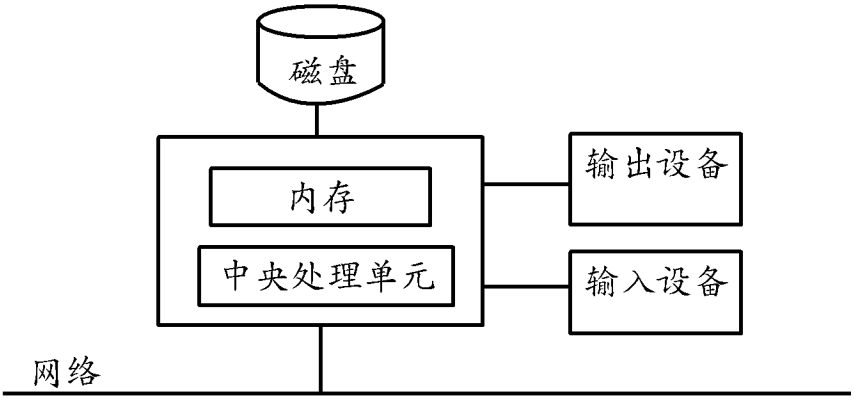

磁盘
内存
中央处理单元
输出设备
输入设备
网络

图 2

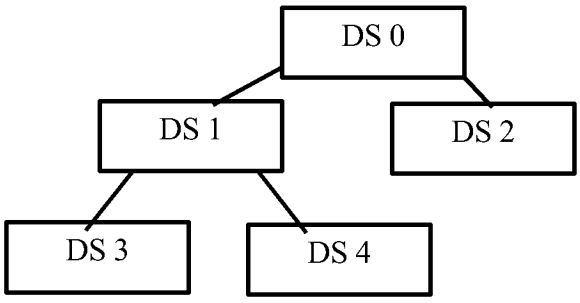

DS 0
DS 1
DS 2
DS 3
DS 4

图 3

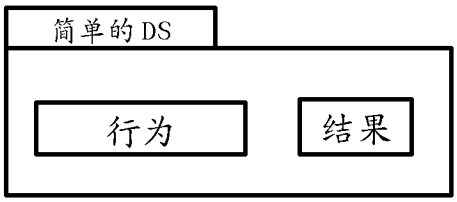

简单的 DS
行为
结果

图 4

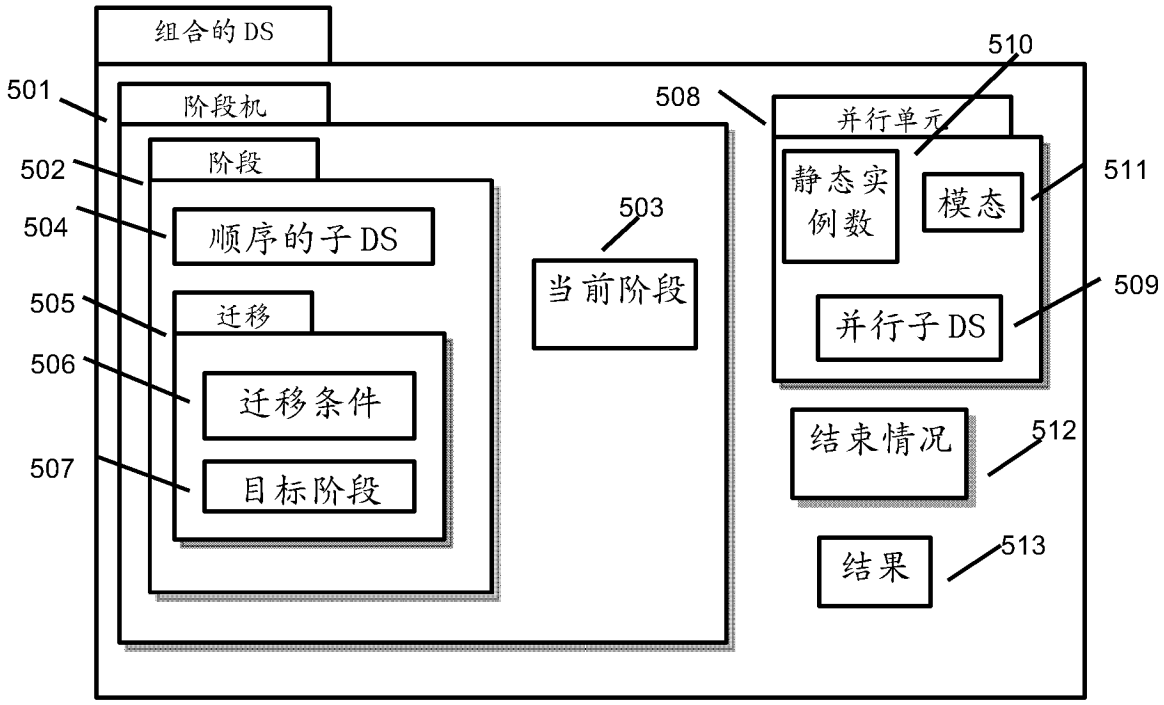

组合的 DS
501
阶段机
502
阶段
504
顺序的子 DS
505
迁移
506
迁移条件
507
目标阶段
503
当前阶段
508
并行单元
510
静态实例数
模态
511
509
并行子 DS
结束情况
512
结果
513

图 5

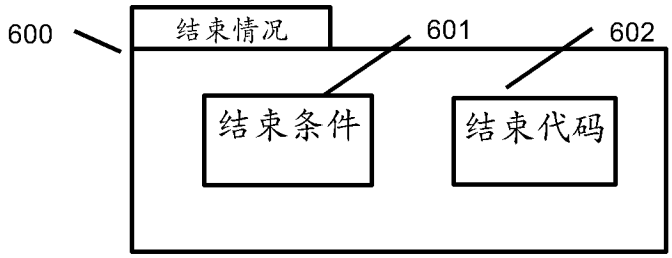

600
结束情况
601
602
结束条件
结束代码

图 6

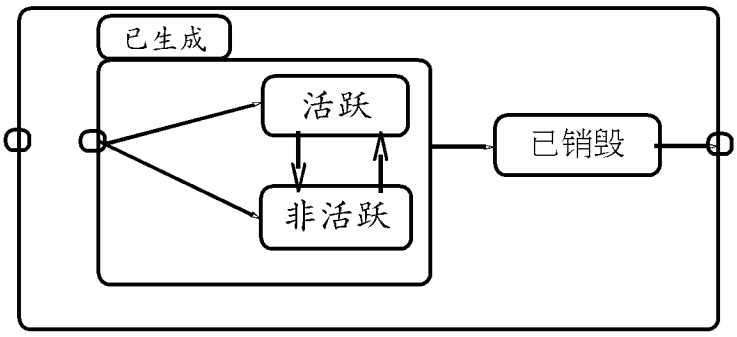

已生成
活跃
非活跃
已销毁

图 7

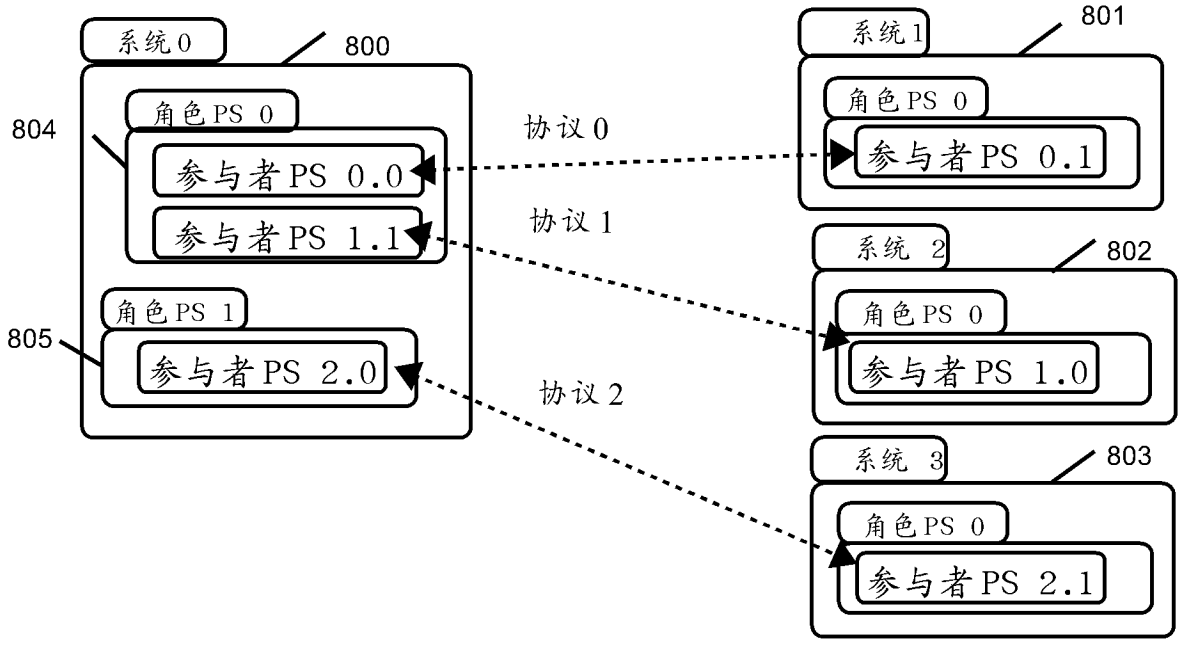

系统 0
800
角色 PS 0
804
参与者 PS 0.0
参与者 PS 1.1
角色 PS 1
805
参与者 PS 2.0
协议 0
协议 1
协议 2
系统 1
801
角色 PS 0
参与者 PS 0.1
系统 2
802
角色 PS 0
参与者 PS 1.0
系统 3
803
角色 PS 0
参与者 PS 2.1

图 8

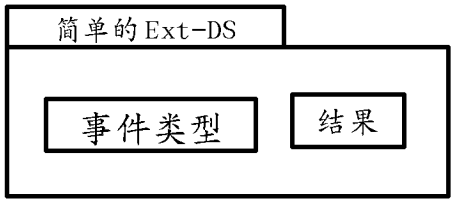

简单的 Ext-DS
事件类型
结果

图 9

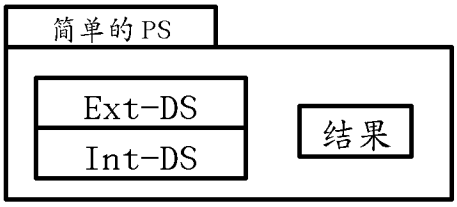

简单的 PS
Ext-DS
Int-DS
结果

图 10

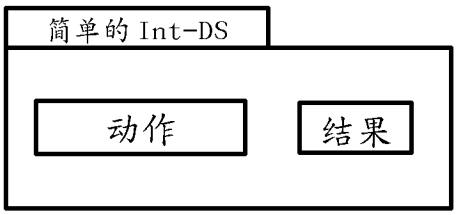

简单的 Int-DS
动作
结果

图 11

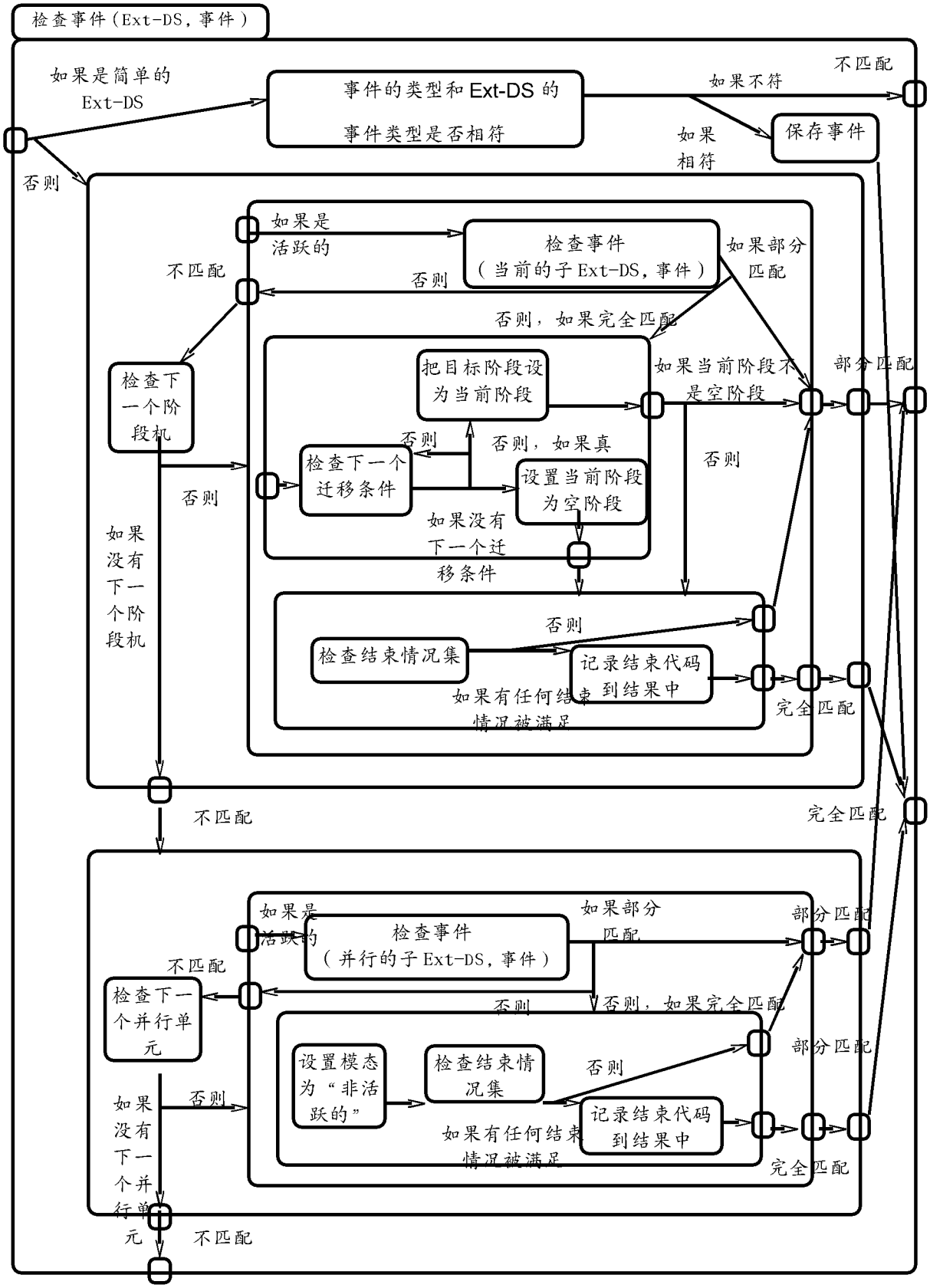

检查事件（Ext-DS，事件）
如果是简单的 Ext-DS
事件的类型和 Ext-DS 的事件类型是否相符
如果不符
不匹配
如果相符
保存事件
否则
如果是活跃的
检查事件（当前的子 Ext-DS，事件）
如果部分匹配
不匹配
否则
否则，如果完全匹配
检查下一个阶段机
把目标阶段设为当前阶段
如果当前阶段不是空阶段
部分匹配
否则
否则，如果真
检查下一个迁移条件
设置当前阶段为空阶段
否则
如果没有下一个迁移条件
否则
如果没有下一个阶段机
检查结束情况集
记录结束代码到结果中
如果有任何结束情况被满足
完全匹配
不匹配
完全匹配
如果是活跃的
检查事件（并行的子 Ext-DS，事件）
如果部分匹配
部分匹配
不匹配
检查下一个并行单元
否则
否则，如果完全匹配
设置模态为“非活跃的”
检查结束情况集
否则
部分匹配
如果没有下一个并行单元
否则
记录结束代码到结果中
如果有任何结束情况被满足
完全匹配
不匹配

图 12

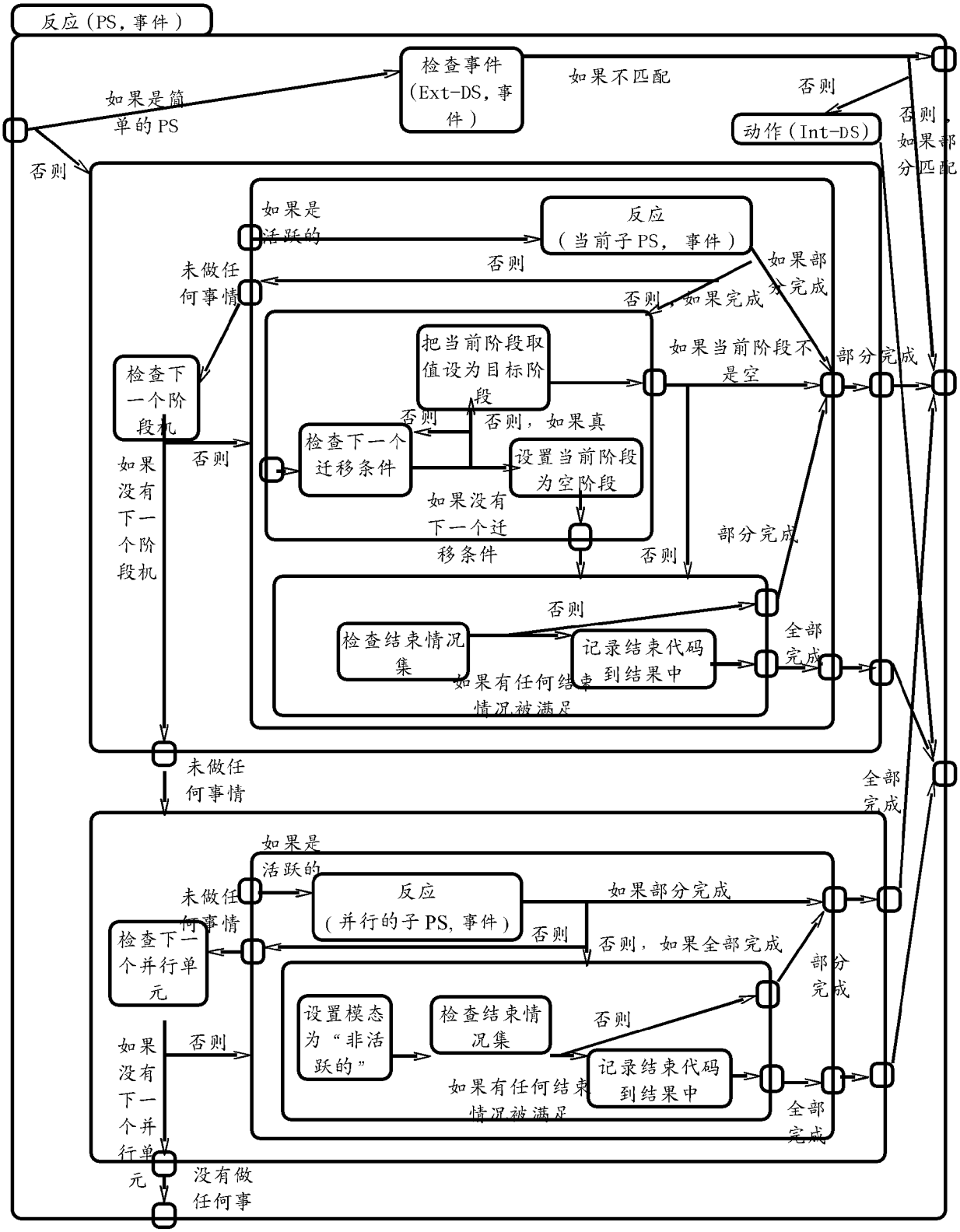

反应(PS,事件)
检查事件(Ext-DS,事件)
如果是简单的PS
否则
如果不匹配
否则
动作(Int-DS)
否则，如果部分匹配
如果是活跃的
反应(当前子PS，事件)
否则
如果部分完成
未做任何事情
否则，如果完成
把当前阶段取值设为目标阶段
如果当前阶段不是空
部分完成
检查下一个阶段机
否则
否则，如果真
检查下一个迁移条件
设置当前阶段为空阶段
如果没有下一个阶段机
否则
如果没有下一个迁移条件
部分完成
否则
检查结束情况集
记录结束代码到结果中
全部完成
如果有任何结束情况被满足
未做任何事情
全部完成
如果是活跃的
反应(并行的子PS,事件)
如果部分完成
未做任何事情
否则
否则，如果全部完成
检查下一个并行单元
部分完成
设置模态为“非活跃的”
检查结束情况集
否则
如果没有下一个并行单元
否则
记录结束代码到结果中
如果有任何结束情况被满足
全部完成
没有做任何事

图 13

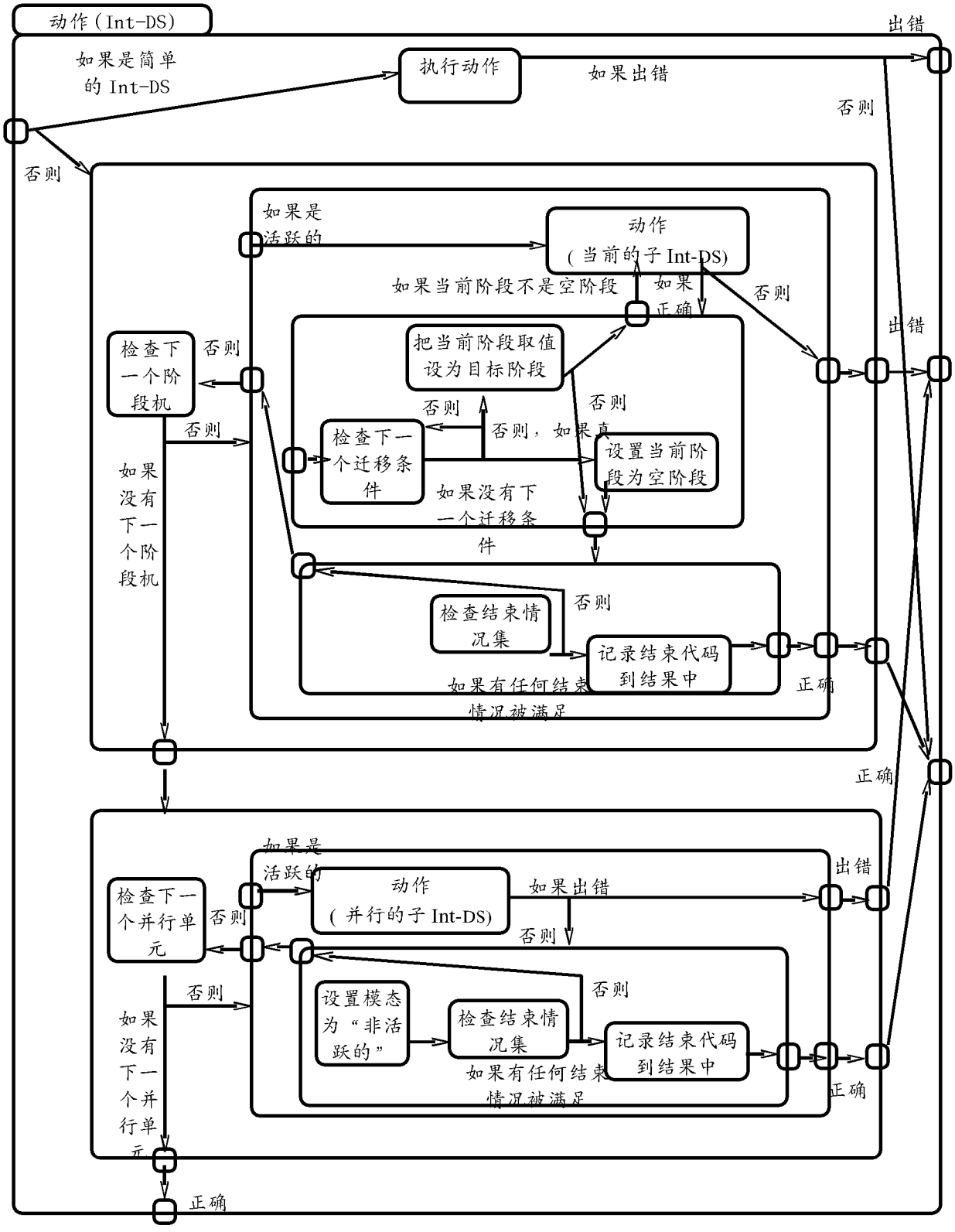

动作(Int-DS)
如果是简单的 Int-DS
执行动作
如果出错
出错
否则
否则
如果是活跃的
动作(当前的子 Int-DS)
如果当前阶段不是空阶段
如果正确
否则
出错
检查下一个阶段机
否则
把当前阶段取值设为目标阶段
否则
否则
否则，如果真
检查下一个迁移条件
设置当前阶段为空阶段
如果没有下一个阶段机
否则
如果没有下一个迁移条件
检查结束情况集
否则
记录结束代码到结果中
如果有任何结束情况被满足
正确
正确
如果是活跃的
动作(并行的子 Int-DS)
出错
如果出错
检查下一个并行单元
否则
否则
否则
设置模态为“非活跃的”
检查结束情况集
否则
记录结束代码到结果中
如果有任何结束情况被满足
正确
如果没有下一个并行单元
正确

图 14

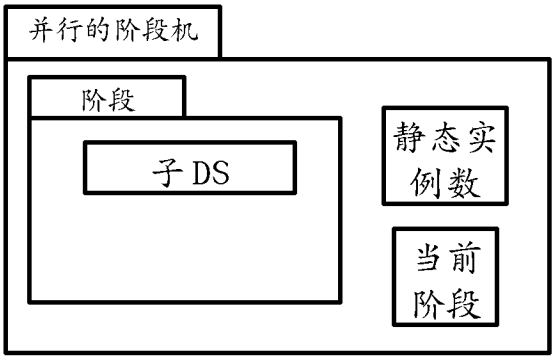


图 15A

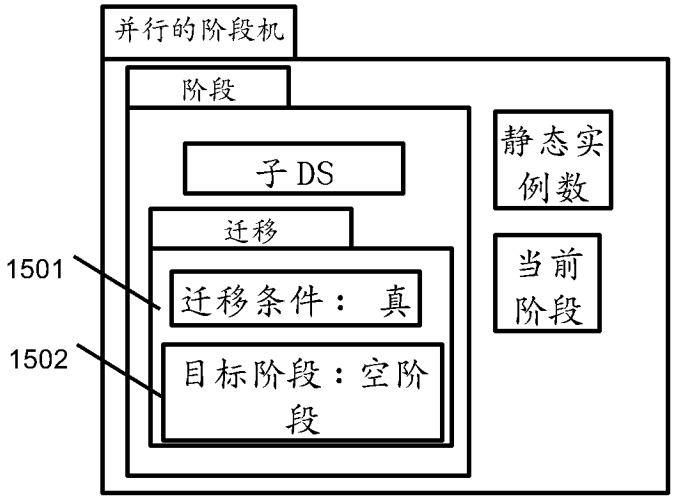


图 15B

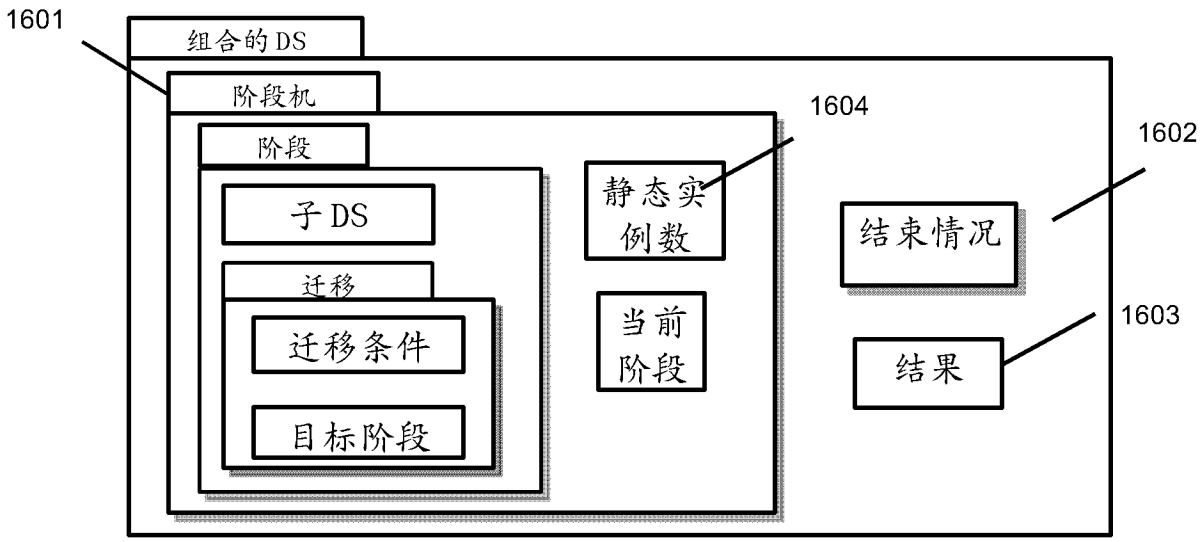


图 16

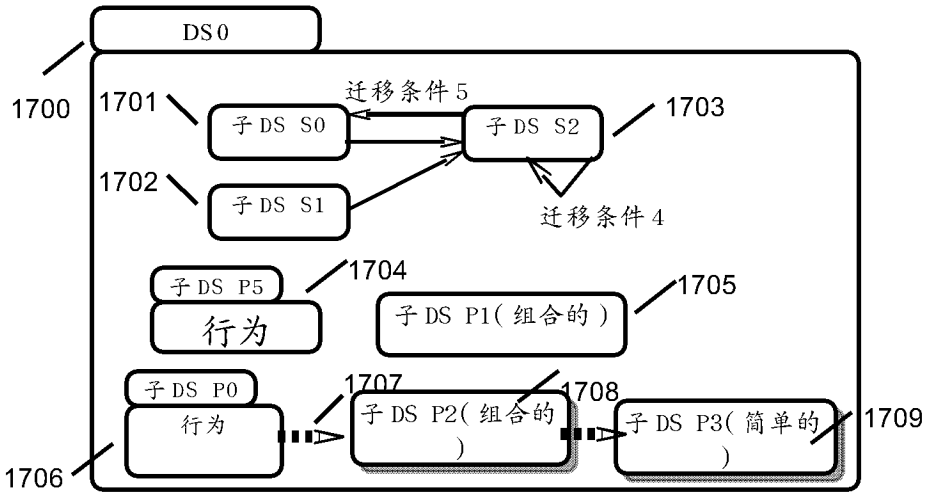

DS 0
1700
1701
子 DS S0
迁移条件 5
子 DS S2
1703
1702
子 DS S1
迁移条件 4
子 DS P5
1704
行为
子 DS P1(组合的)
1705
子 DS P0
行为
1707
子 DS P2(组合的)
1708
子 DS P3(简单的)
1709
1706

图 17

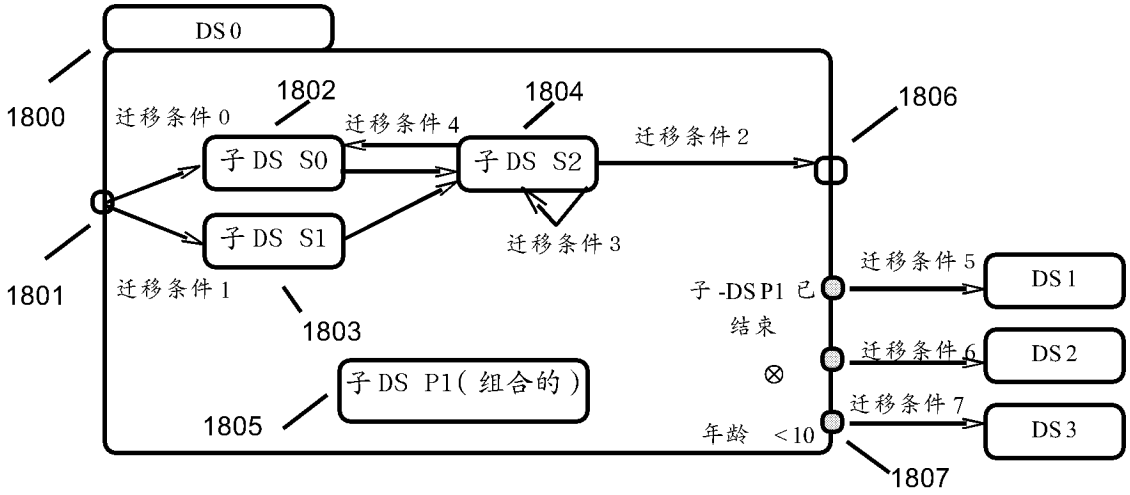

DS 0
1800
迁移条件 0
1802
迁移条件 4
1804
迁移条件 2
1806
子 DS S0
子 DS S2
子 DS S1
迁移条件 3
1801
迁移条件 1
1803
迁移条件 5
DS 1
子 -DS P1 已
结束
迁移条件 6
DS 2
⊗
迁移条件 7
DS 3
子 DS P1(组合的)
1805
年龄 <10
1807

图 18

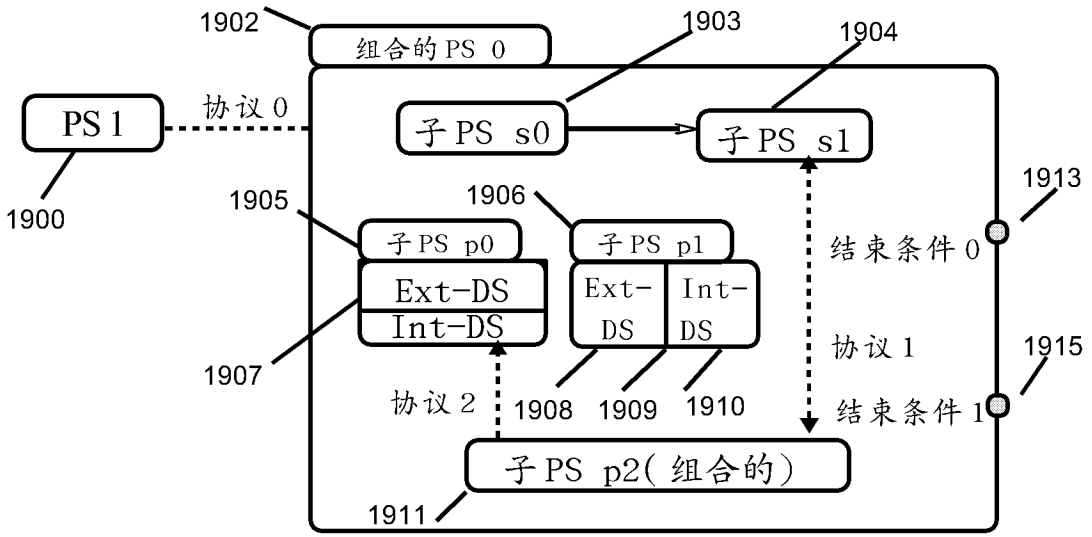

1902
组合的 PS 0
1903
1904
PS 1
协议 0
子 PS s0
子 PS s1
1900
1905
1906
1913
子 PS p0
子 PS p1
结束条件 0
Ext-DS
Int-DS
Ext-DS
Int-DS
协议 1
1915
1907
协议 2
1908
1909
1910
结束条件 1
子 PS p2(组合的)
1911

图 19

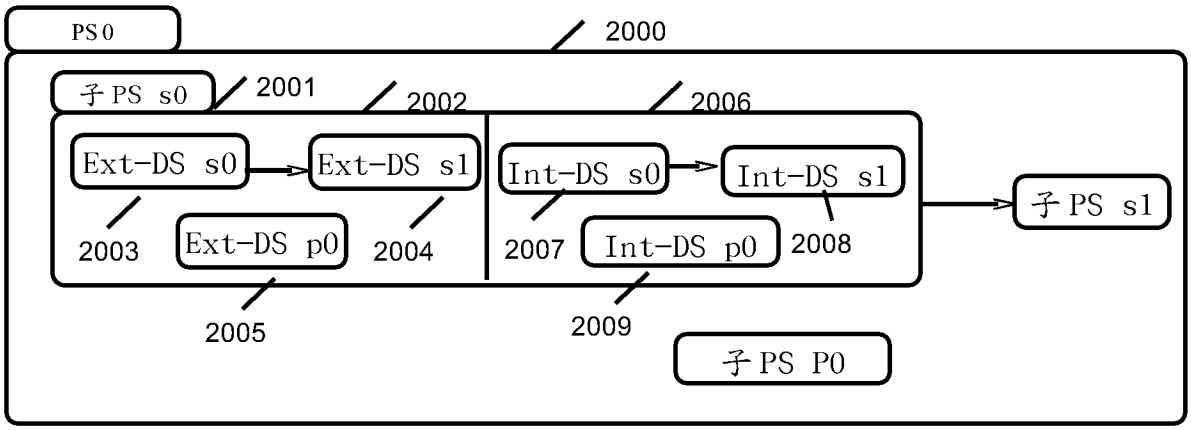

PS 0
2000
子 PS s0
2001
2002
2006
Ext-DS s0
Ext-DS s1
Int-DS s0
Int-DS s1
子 PS s1
2003
Ext-DS p0
2004
2007
Int-DS p0
2008
2005
2009
子 PS P0

图 20

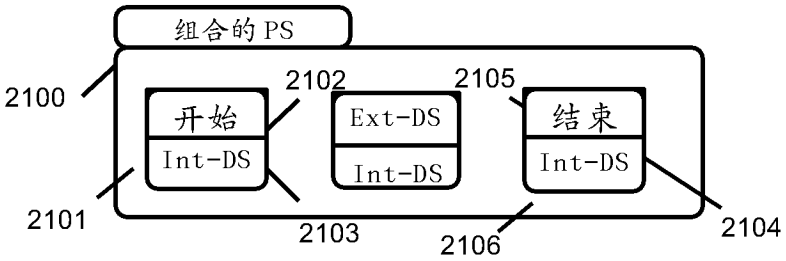

组合的 PS
2100
2102
2105
开始
Int-DS
Ext-DS
Int-DS
结束
Int-DS
2101
2103
2106
2104

图 21

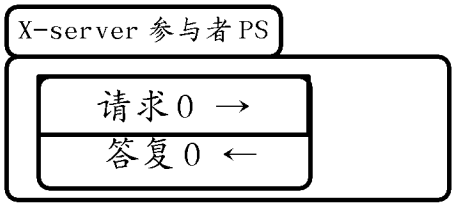


图 22

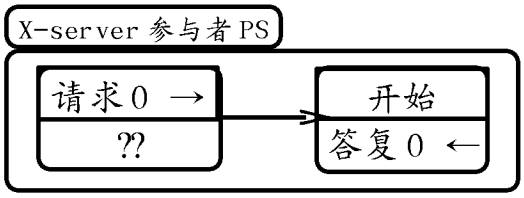


图 23

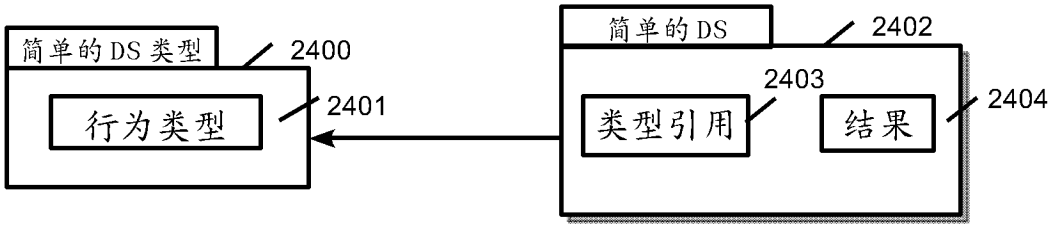


图 24

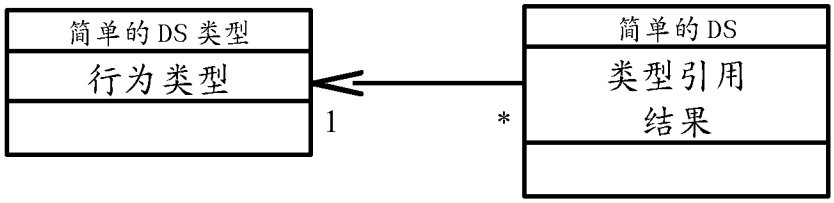


图 25

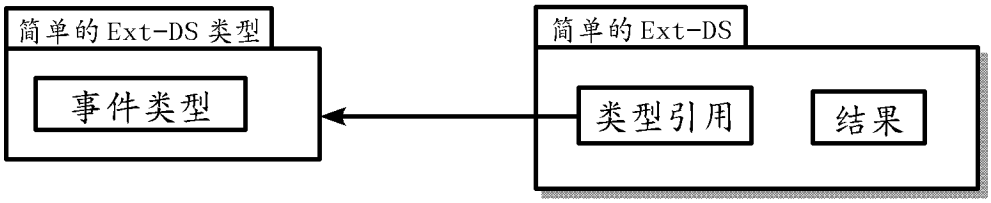


图 26

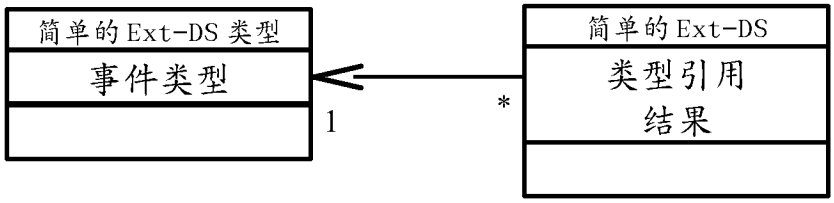


图 27

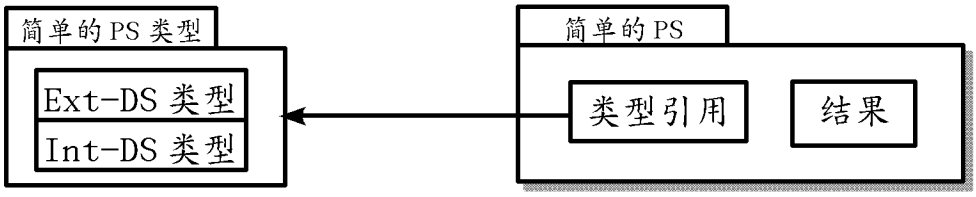


图 28

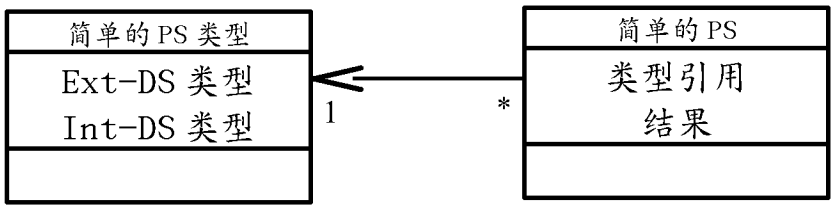


图 29

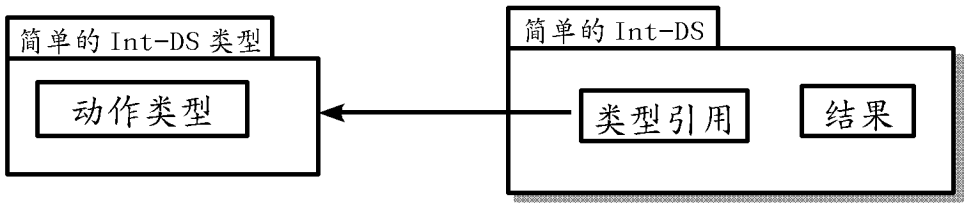


图 30

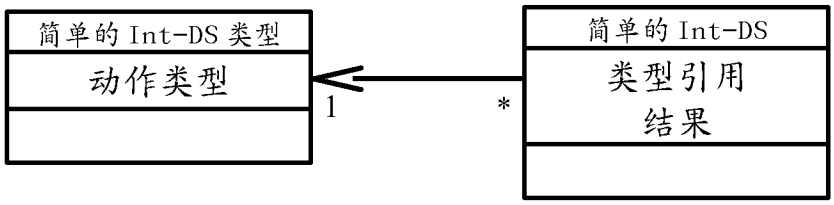


图 31

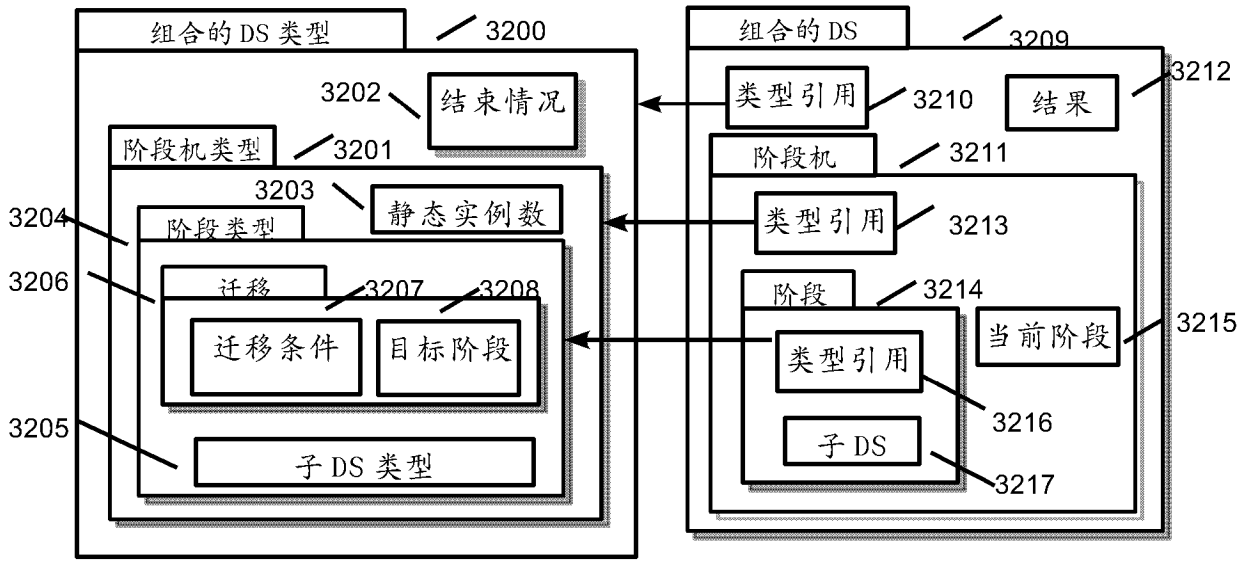

组合的DS类型
3200
3202
结束情况
阶段机类型
3201
3203
静态实例数
3204
阶段类型
3206
迁移
3207
3208
迁移条件
目标阶段
3205
子DS类型
组合的DS
3209
类型引用
3210
结果
3212
阶段机
3211
类型引用
3213
阶段
3214
当前阶段
3215
类型引用
3216
子DS
3217

图 32A

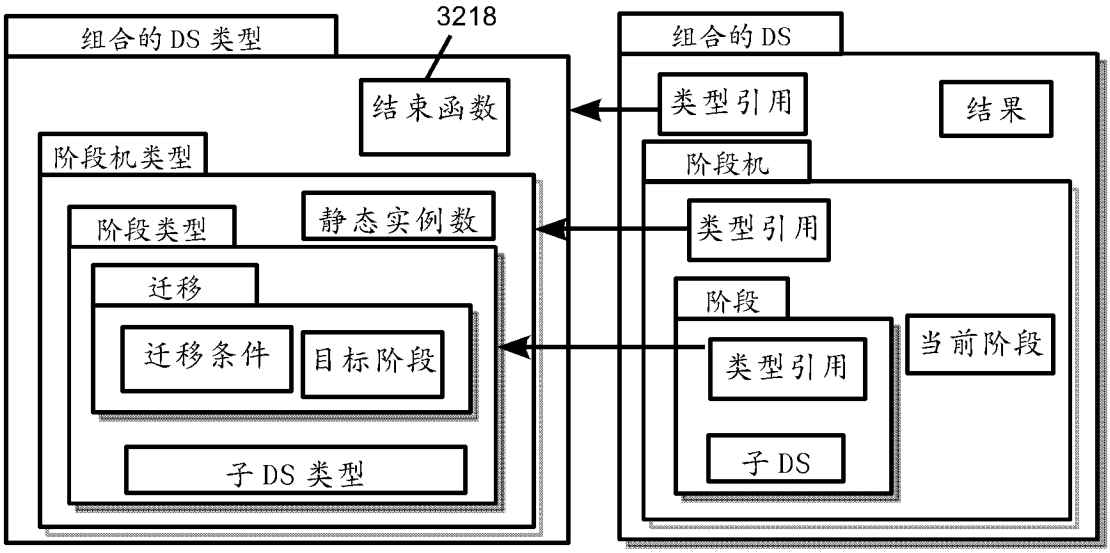

组合的DS类型
3218
结束函数
阶段机类型
阶段类型
静态实例数
迁移
迁移条件
目标阶段
子DS类型
组合的DS
类型引用
结果
阶段机
类型引用
阶段
当前阶段
类型引用
子DS

图 32B

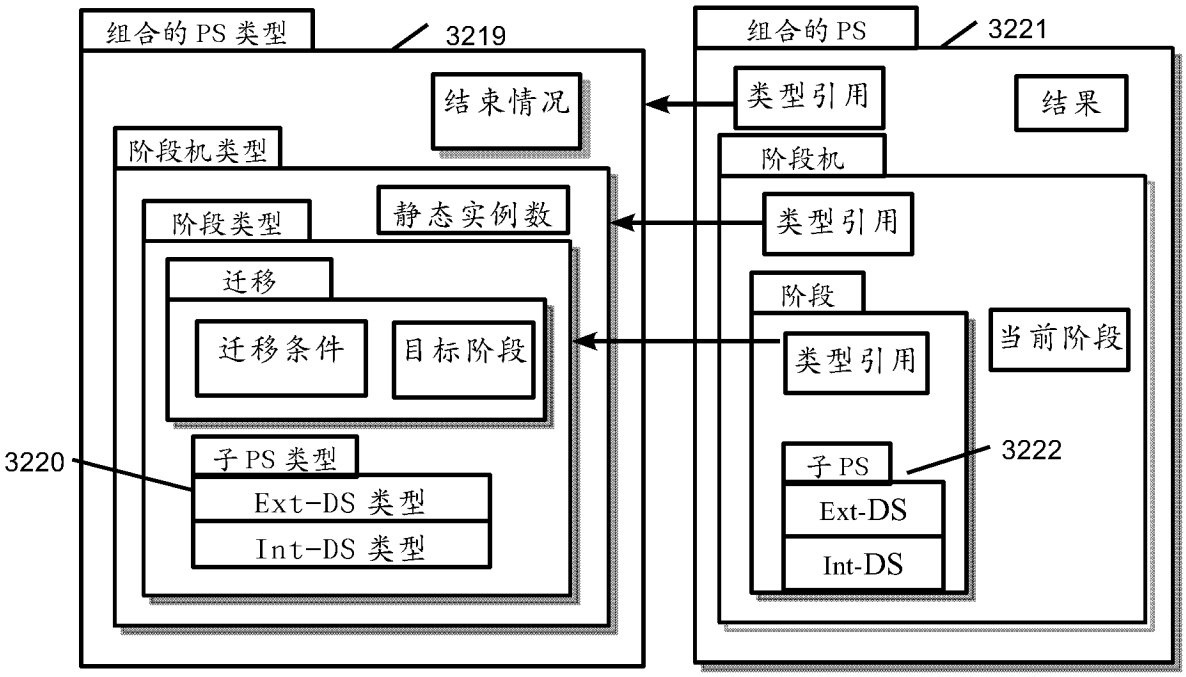

组合的 PS 类型
3219
结束情况
阶段机类型
阶段类型
静态实例数
迁移
迁移条件
目标阶段
3220
子 PS 类型
Ext-DS 类型
Int-DS 类型
组合的 PS
3221
类型引用
结果
阶段机
类型引用
阶段
当前阶段
类型引用
子 PS
3222
Ext-DS
Int-DS

图 32C

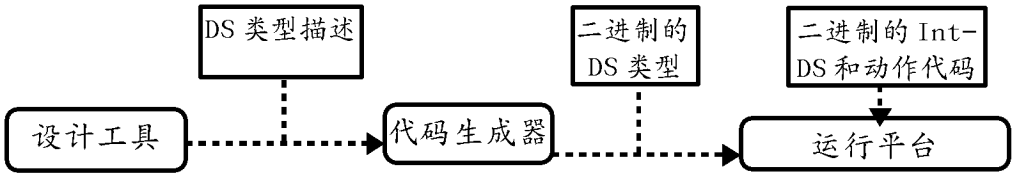

DS 类型描述
二进制的 DS 类型
二进制的 Int-DS 和动作代码
设计工具
代码生成器
运行平台

图 33

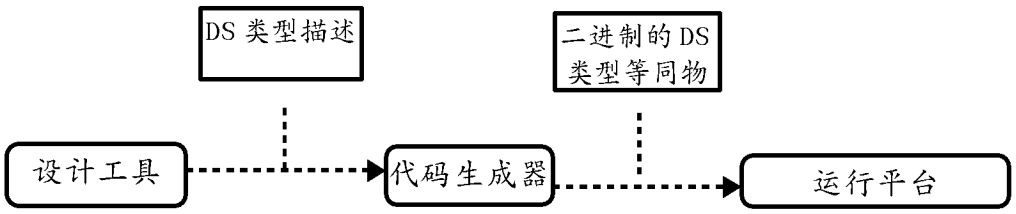

DS 类型描述
二进制的 DS 类型等同物
设计工具
代码生成器
运行平台

图 34

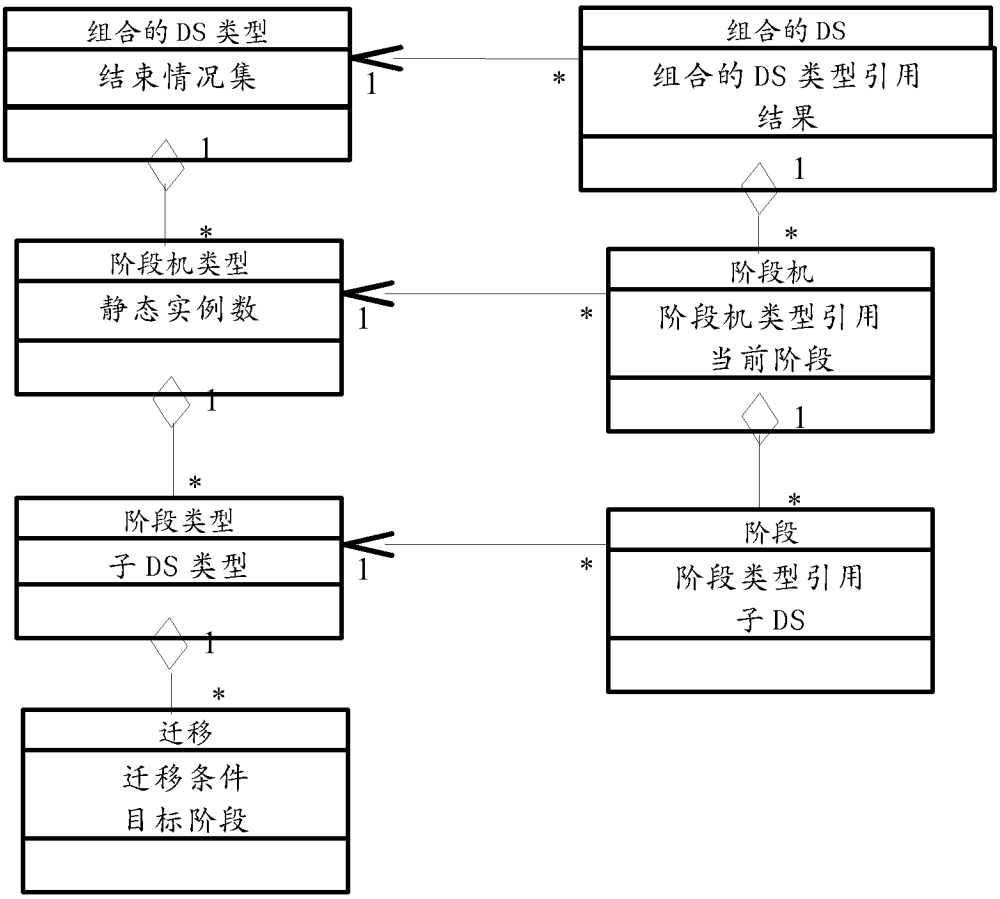

组合的DS类型
结束情况集
组合的DS
组合的DS类型引用
结果
阶段机类型
静态实例数
阶段机
阶段机类型引用
当前阶段
阶段类型
子DS类型
阶段
阶段类型引用
子DS
迁移
迁移条件
目标阶段
1
*

图 35

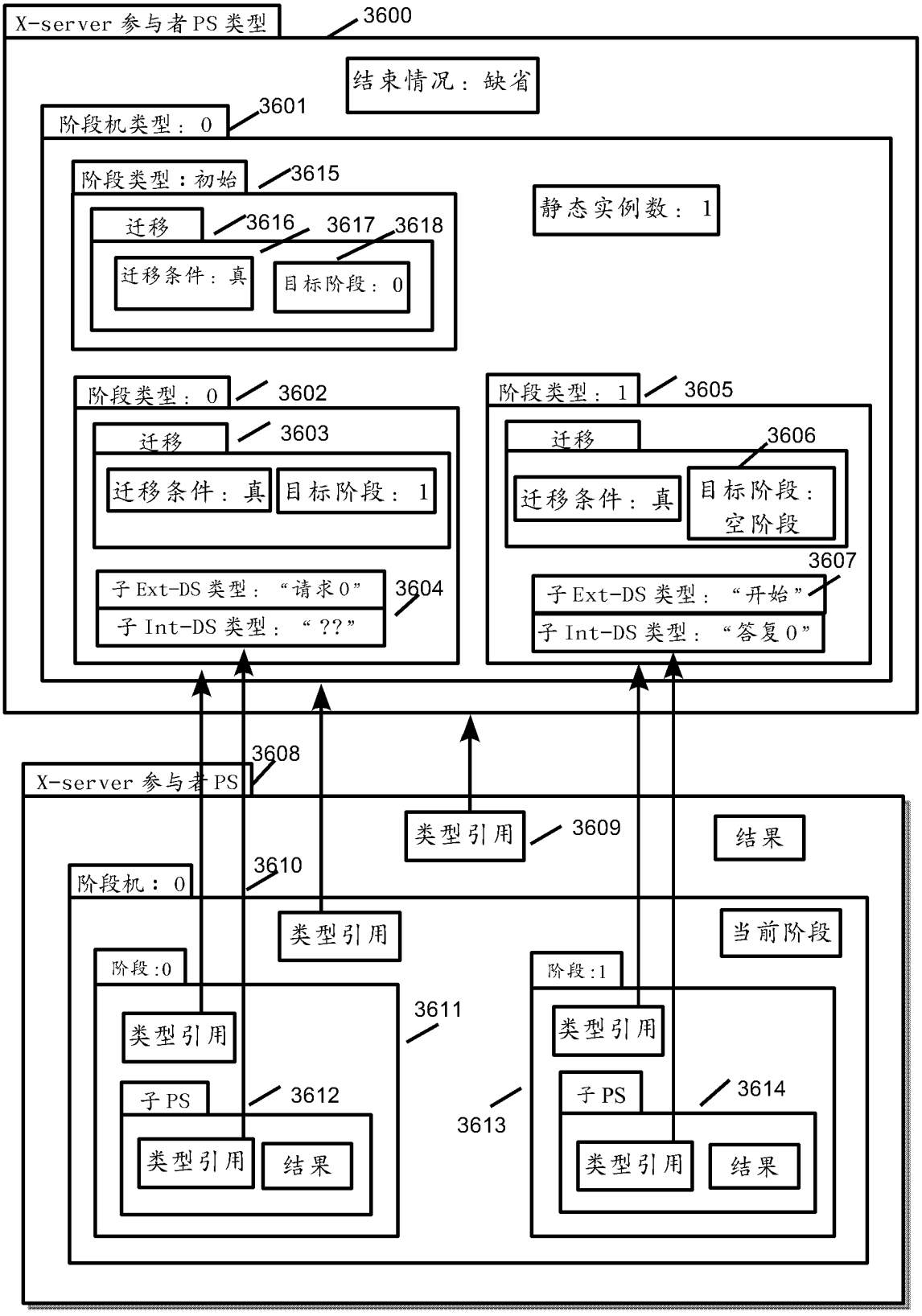

X-server 参与者PS 类型
3600
结束情况：缺省
阶段机类型：0
3601
阶段类型：初始
3615
迁移
3616
3617
3618
迁移条件：真
目标阶段：0
静态实例数：1
阶段类型：0
3602
迁移
3603
迁移条件：真
目标阶段：1
子 Ext-DS 类型：“请求0”
子 Int-DS 类型：“??”
3604
阶段类型：1
3605
迁移
3606
迁移条件：真
目标阶段：空阶段
3607
子 Ext-DS 类型：“开始”
子 Int-DS 类型：“答复0”
3608
X-server 参与者PS
类型引用
3609
结果
阶段机：0
3610
类型引用
当前阶段
阶段:0
3611
类型引用
子PS
3612
类型引用
结果
3613
阶段:1
类型引用
子PS
3614
类型引用
结果

图 36

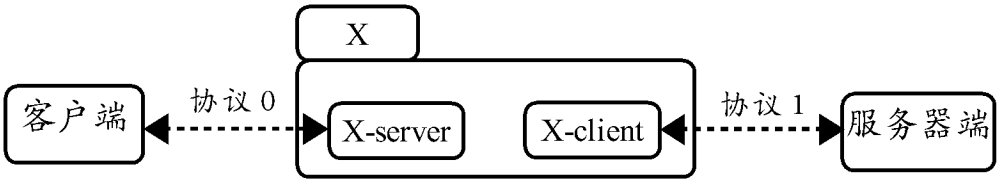


图 37

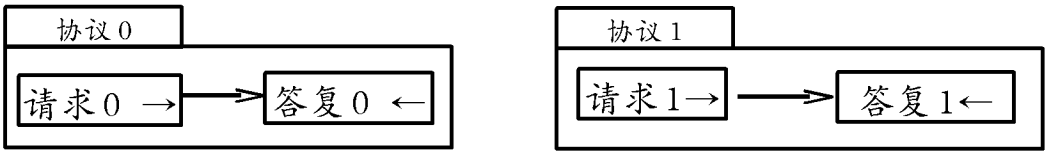


图 38

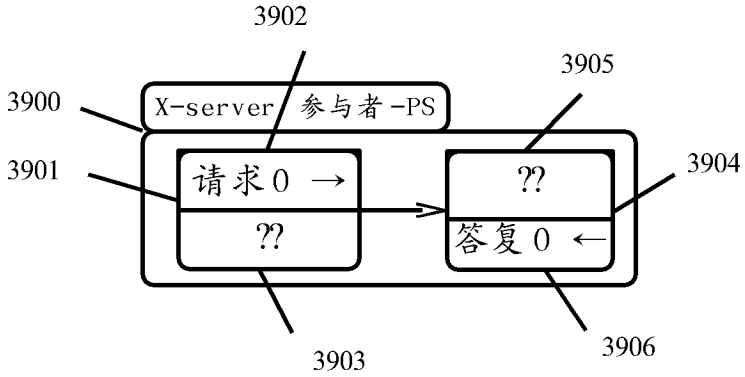


图 39

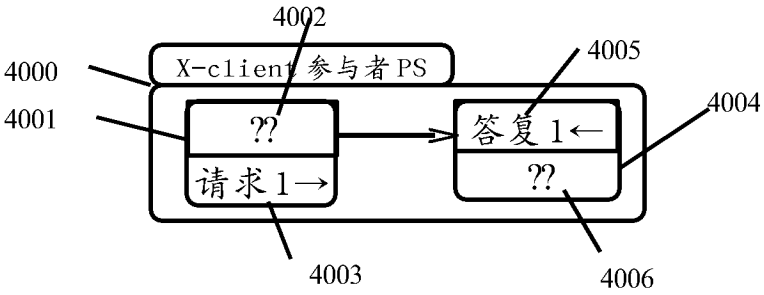


图 40

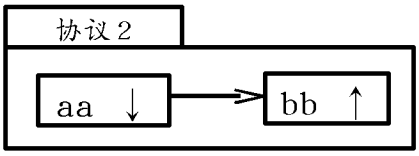


图 41

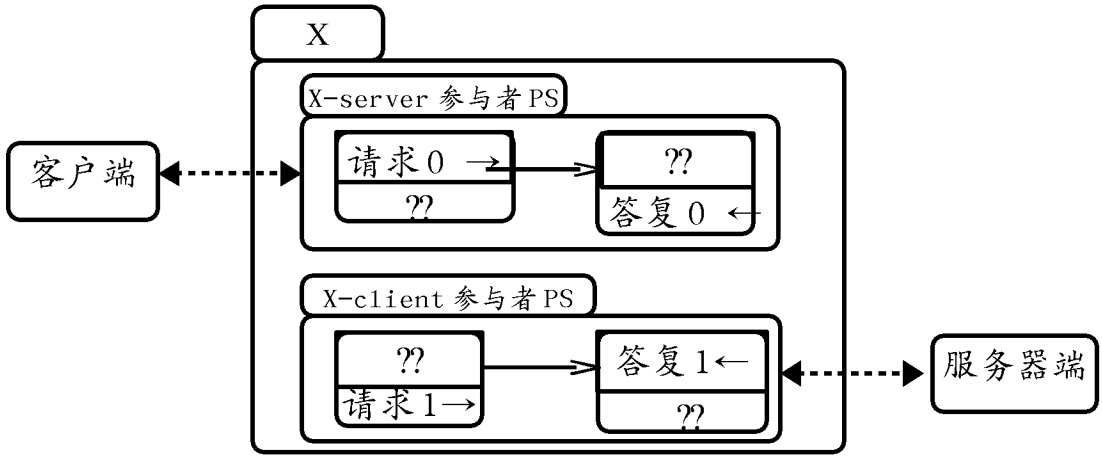

X
X-server 参与者 PS
客户端
请求 0 →
??
??
答复 0 ←
X-client 参与者 PS
??
请求 1→
答复 1←
??
服务器端

图 42

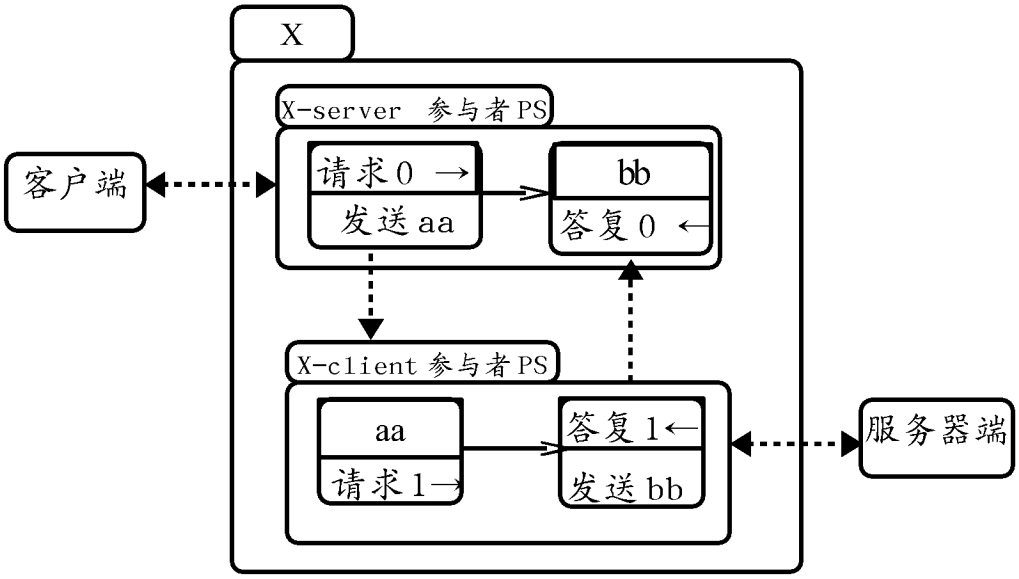

X
X-server 参与者 PS
客户端
请求 0 →
发送 aa
bb
答复 0 ←
X-client 参与者 PS
aa
请求 1→
答复 1←
发送 bb
服务器端

图 43

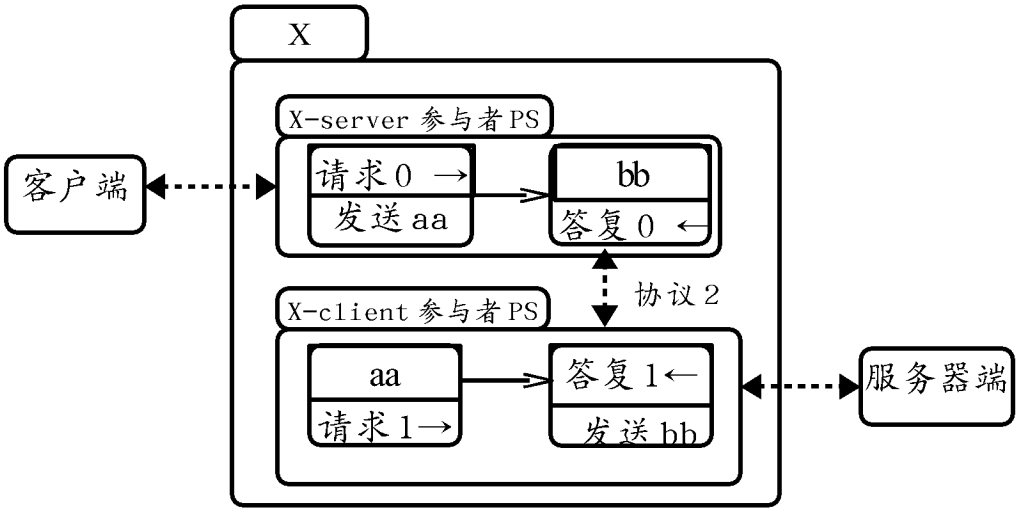

X
X-server 参与者 PS
客户端
请求 0 →
发送 aa
bb
答复 0 ←
协议 2
X-client 参与者 PS
aa
请求 1→
答复 1←
发送 bb
服务器端

图 44

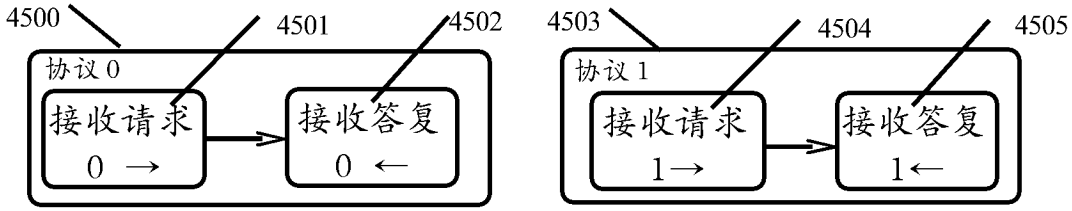

4500
4501
4502
协议 0
接收请求
0 →
接收答复
0 ←
4503
4504
4505
协议 1
接收请求
1→
接收答复
1←

图 45

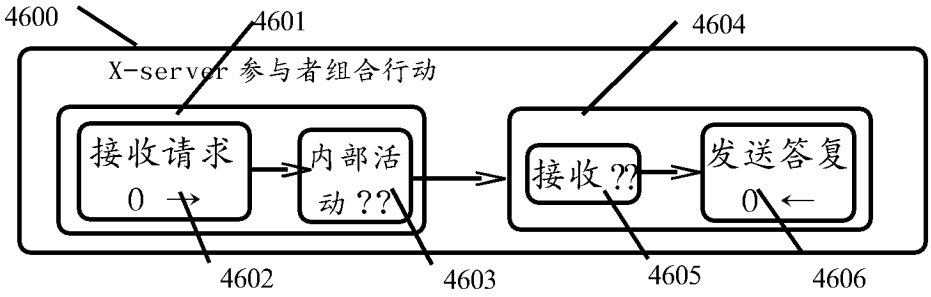

4600
4601
4604
X-server 参与者组合行动
接收请求
0 →
内部活动 ??
接收 ??
发送答复
0 ←
4602
4603
4605
4606

图 46

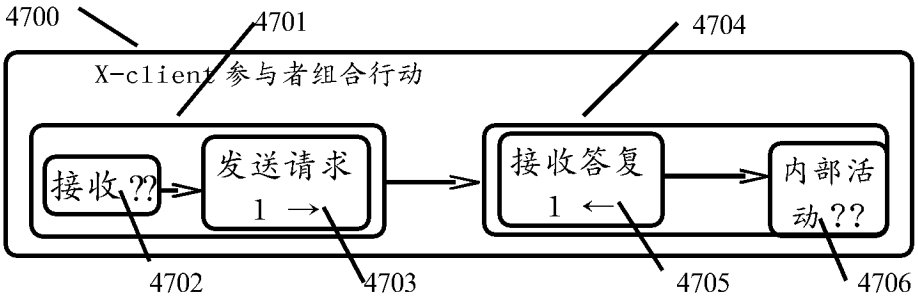

4700
4701
4704
X-client 参与者组合行动
接收 ??
发送请求 1 →
接收答复 1 ←
内部活动 ??
4702
4703
4705
4706

图 47

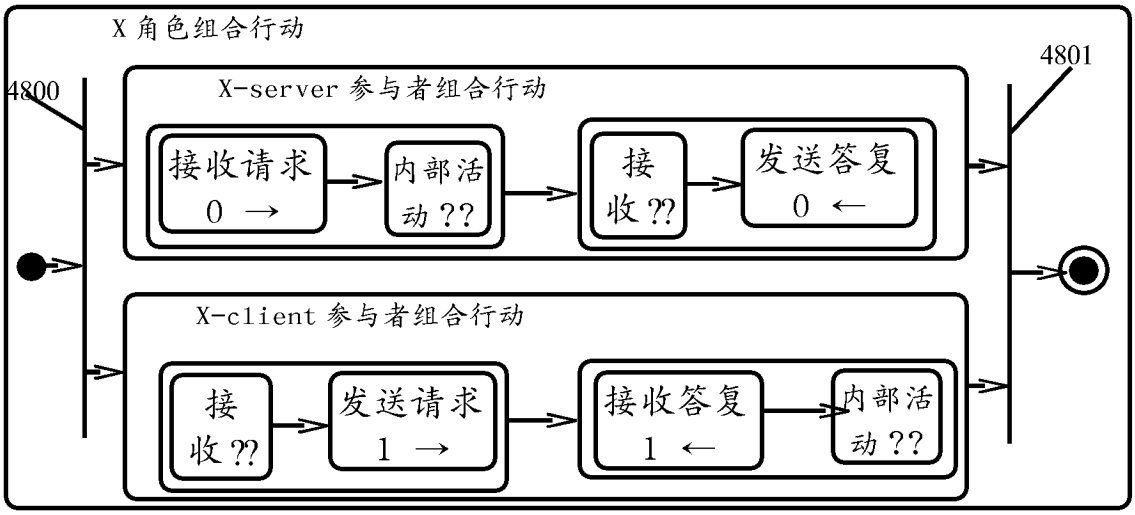

X 角色组合行动
4800
4801
X-server 参与者组合行动
接收请求 0 →
内部活动 ??
接收 ??
发送答复 0 ←
X-client 参与者组合行动
接收 ??
发送请求 1 →
接收答复 1 ←
内部活动 ??

图 48

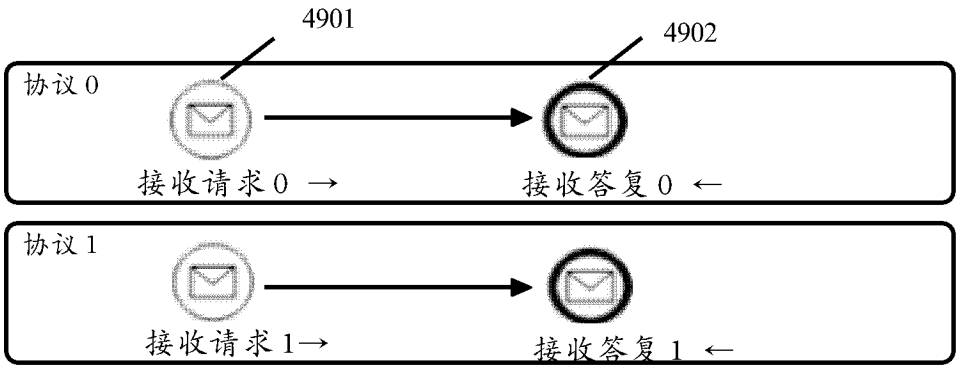

4901
4902
协议 0
接收请求 0 →
接收答复 0 ←
协议 1
接收请求 1→
接收答复 1 ←

图 49

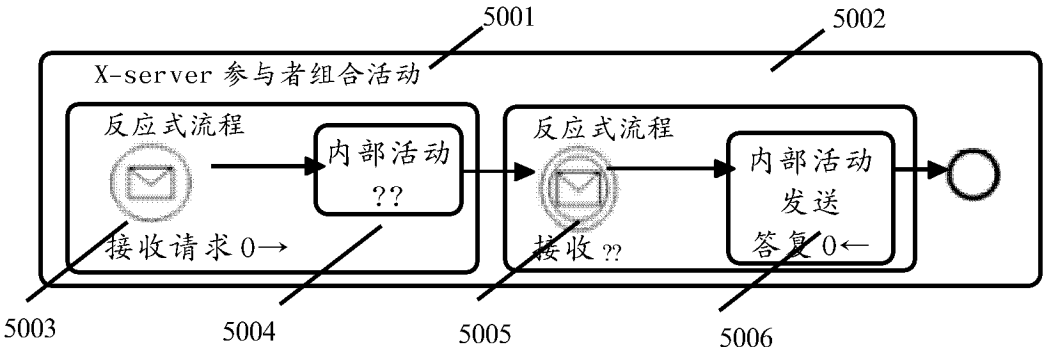

5001
5002
X-server 参与者组合活动
反应式流程
内部活动
??
反应式流程
内部活动
发送
答复 0←
接收请求 0→
接收 ??
5003
5004
5005
5006

图 50

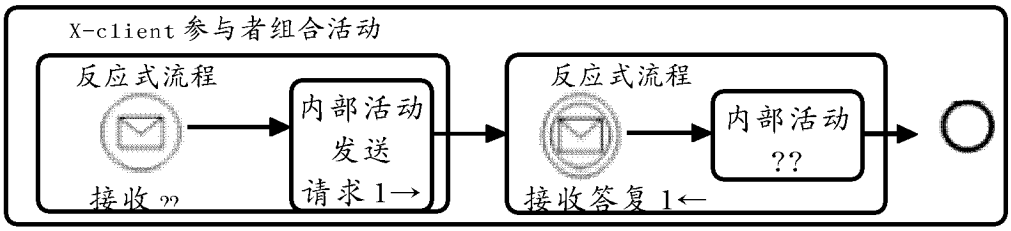

X-client 参与者组合活动
反应式流程
内部活动
发送
请求 1→
反应式流程
内部活动
??
接收 ??
接收答复 1←

图 51

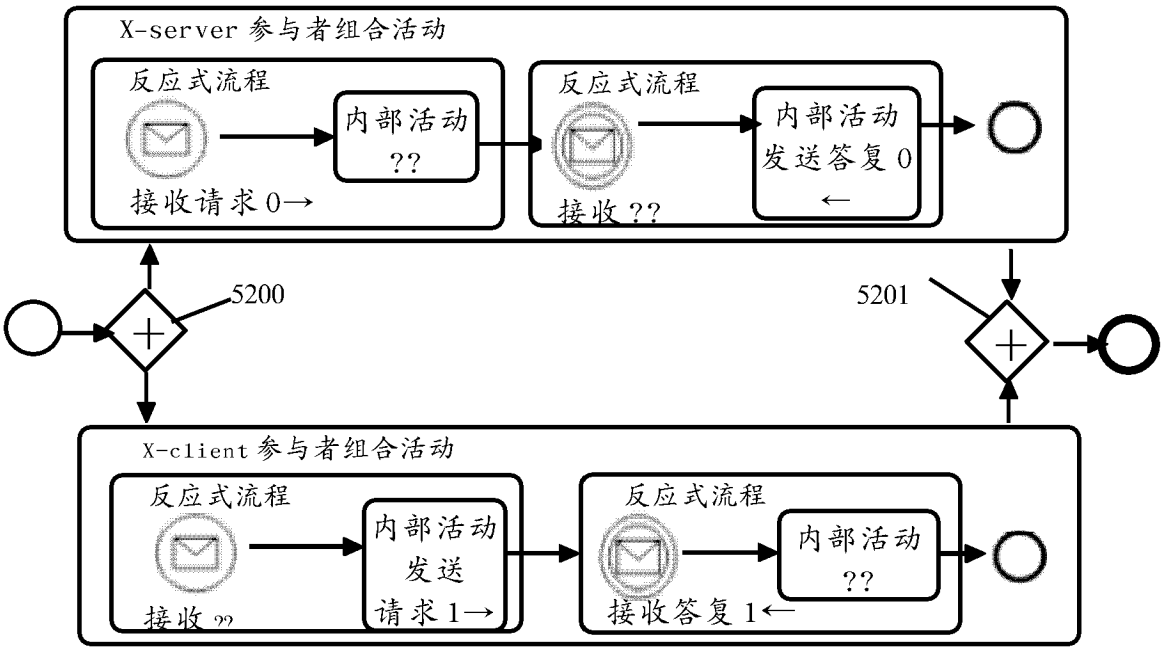

X-server 参与者组合活动
反应式流程
内部活动
??
反应式流程
内部活动
发送答复 0
←
接收请求 0→
接收 ??
5200
5201
X-client 参与者组合活动
反应式流程
内部活动
发送
请求 1→
反应式流程
内部活动
??
接收 ??
接收答复 1←

图 52

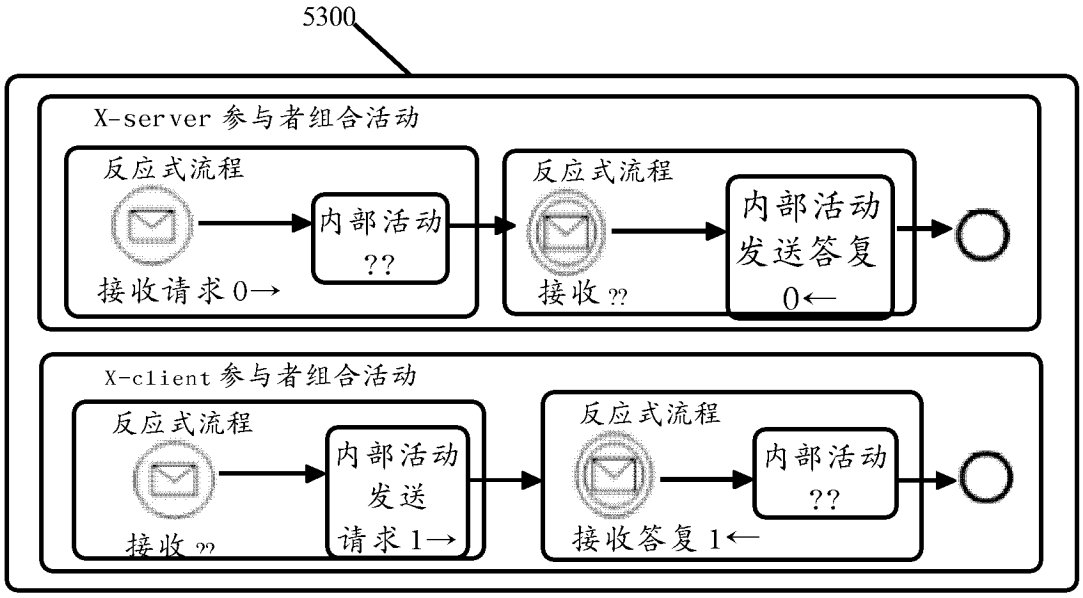


图 53

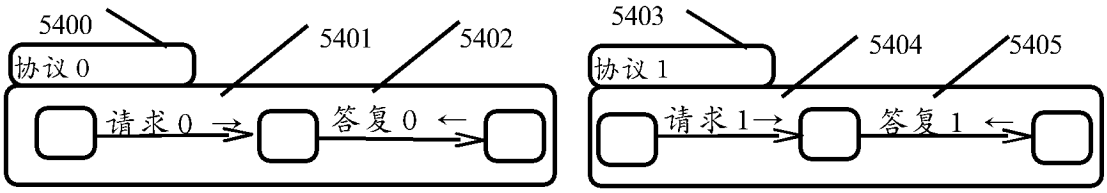


图 54

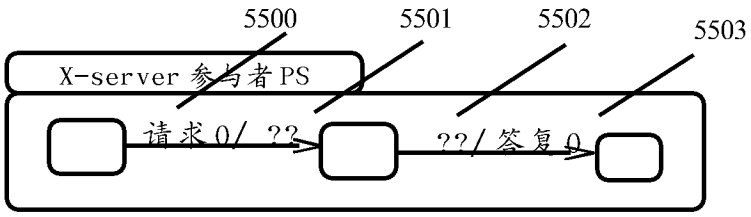


图 55

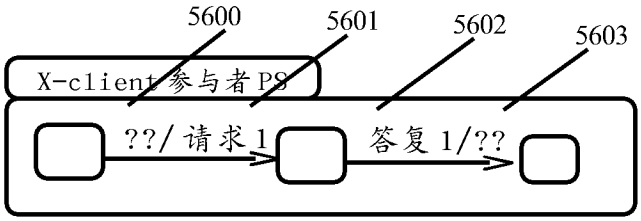


图 56

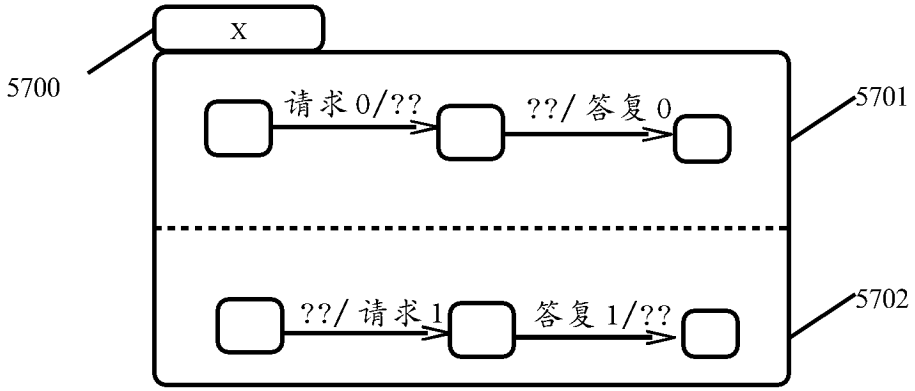


图 57

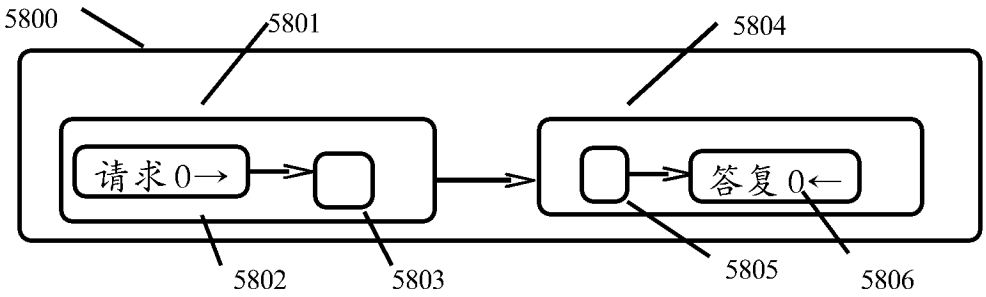


图 58

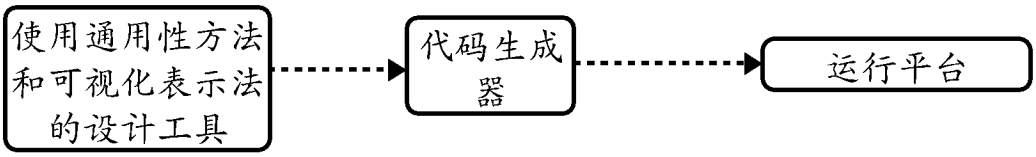


图 59

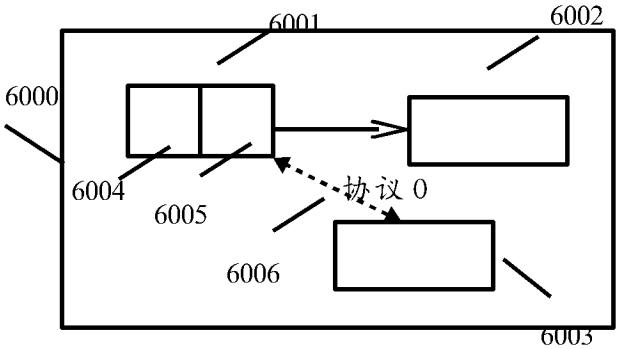


图 60

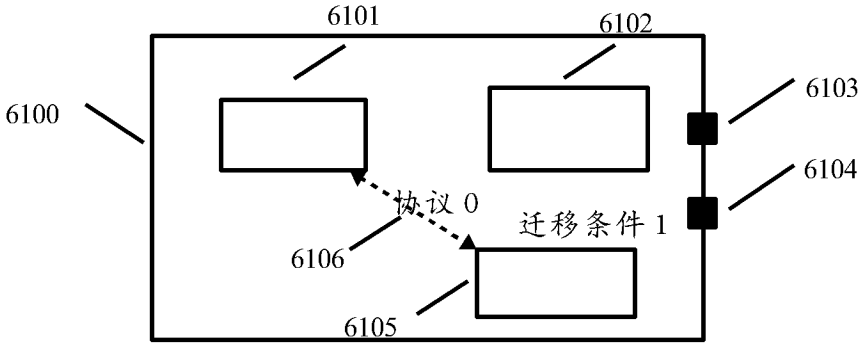

6101
6102
6100
6103
6104
协议 0
迁移条件 1
6106
6105

图 61

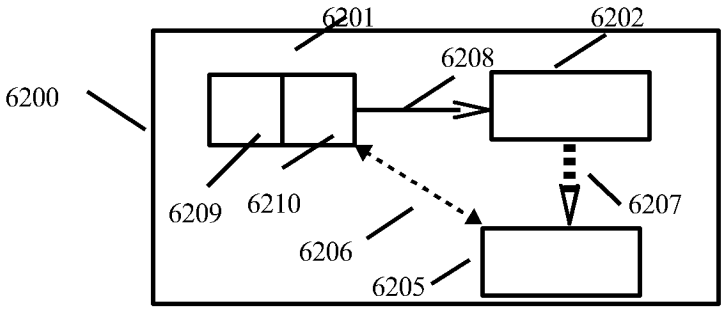

6201
6202
6208
6200
6209
6210
6207
6206
6205

图 62

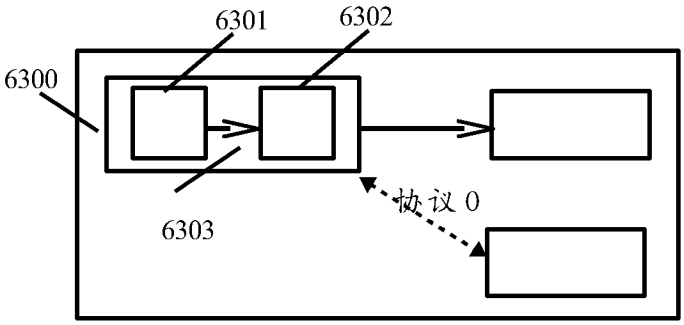

6301
6302
6300
6303
协议 0

图 63