第5章　建立辅助模型

5.1 顺序图

类图对对象之间的消息（交互情况）表达不够详细;
详细说明对消息的表达虽然详细，但不够直观;

一、　概念与表示法

1、概述

顺序图（**Sequence Diagram**）是一种详细表示对象之间以及对象与参与者实例之间交互的图，它由一组协作的对象（或参与者实例）以及它们之间可发送的消息组成，它强调消息之间的顺序。

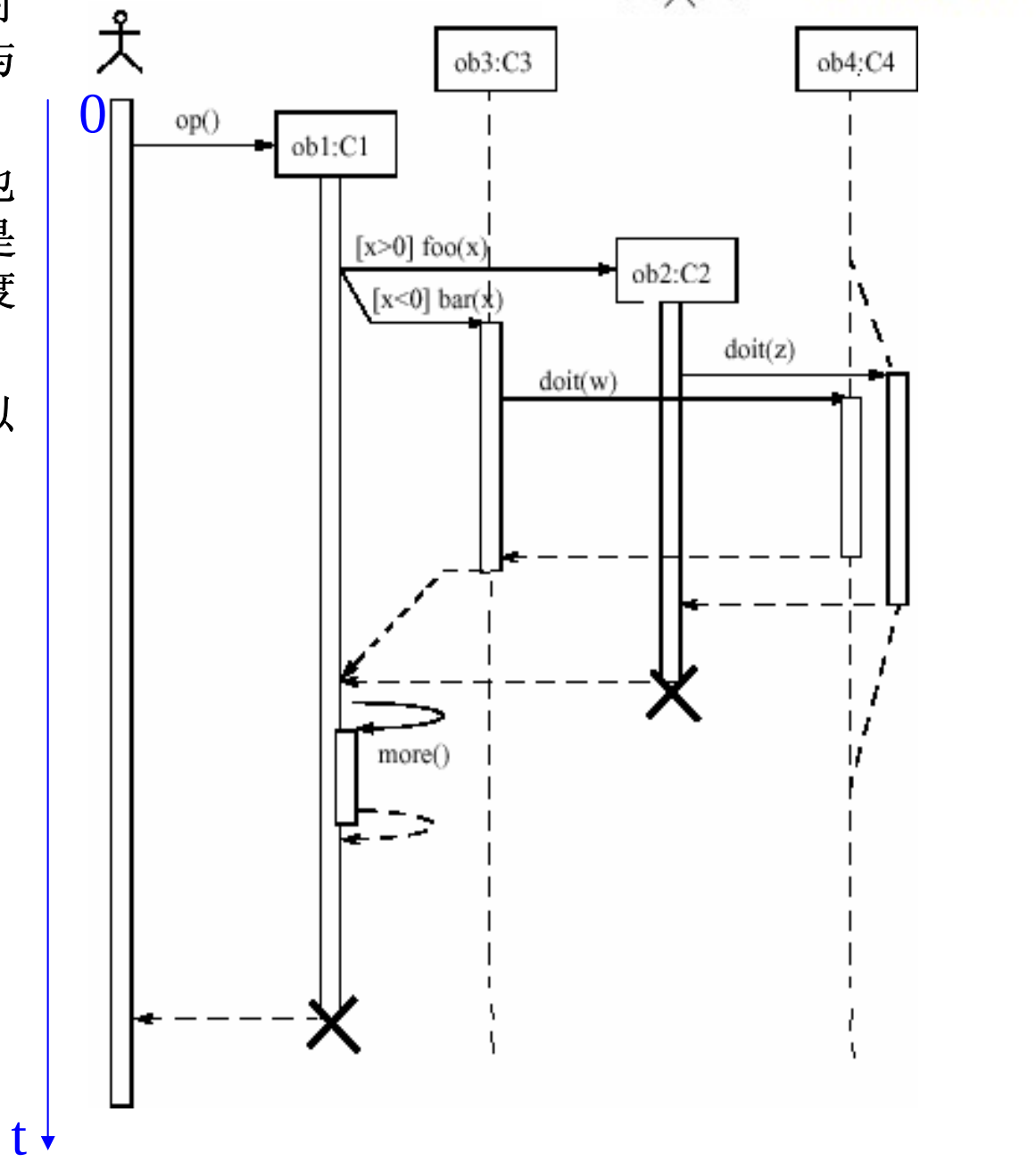
图中含有对象（参与者）、消息、生命线和执行规约组成。

[对象名]：类名

顺序图是二维的：垂直方向表示时间，水平方向表示不同的对象或参与者。

通常时间维由上到下（根据需要，也可以由下到上）。通常只有时间顺序是重要的，但在实时应用中时间轴是能度量的。

对象的水平顺序并不重要，顺序可以是任意的。



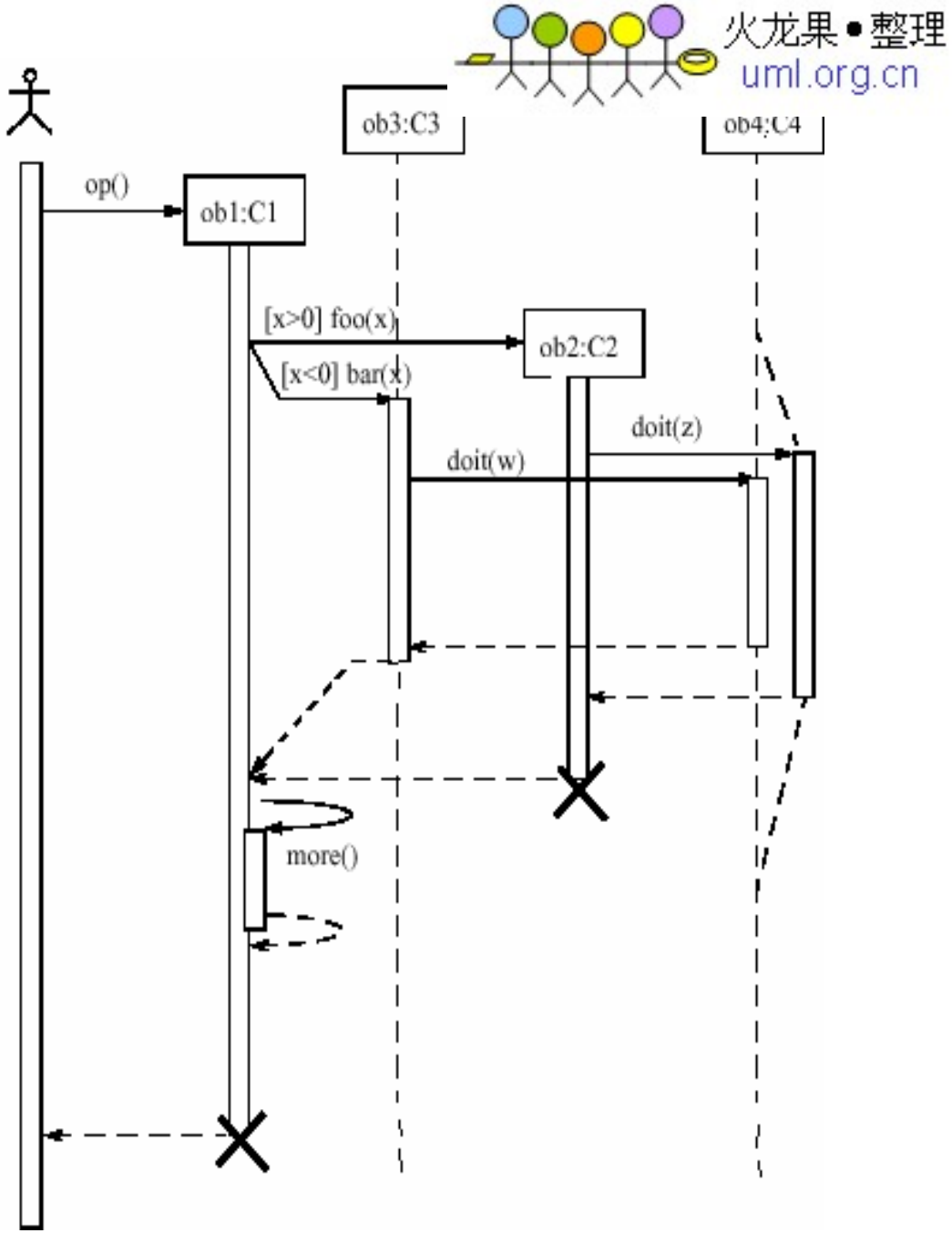
2、对象生命线

把对象表示成称之为“生命线”的垂直虚线。 生命线代表一个对象在特定时间内的存在。

在图的顶部（第一个箭头之上）放置在交互开始时就存在的对象，而在整个交互完成时仍然存在的对象的生命线，要延伸超出最后一个箭头。

如果一个对象在图中所规定的时间段被创建，那么就把创建对象的箭头的头部画在对象符号上。如果对象在图中被销毁，那么用一个大的“X”标记它的析构，该标记或者放在引起析构的箭头处，或者放在从被销毁的对象最终返回的箭头处（在自析构的情况下）。

生命线可以分裂成两条或更多条并发的生命线，以表示条件性。这样的每一个生命线对应于交互中的一个条件分支。生命线可以在某个后续点处合并。



3、执行规约

执行规约表示一个对象直接或者通过从属例程执行一个行为的时期。它既表示了行为执行的持续时间，也表示了调用者与被调用者之间的控制关系。

用一个窄长的矩形表示执行规约，矩形顶端和它的开始时刻对齐，末端和它的结束时刻对齐。

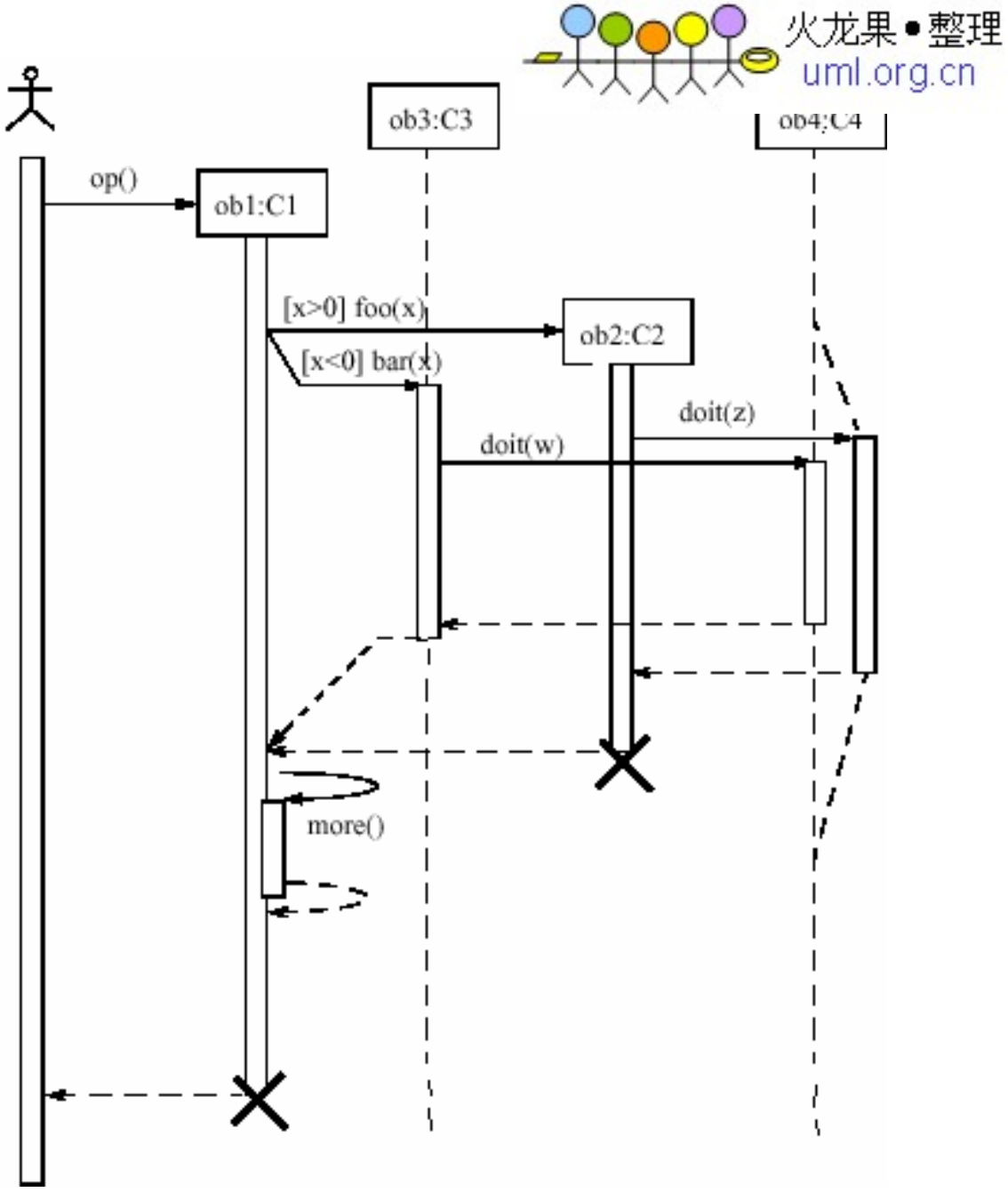
执行规约符号的顶端画在进入的箭头的尖端（开始该动作的那个箭头），底端画在返回的箭头的尾部。

当一个对象处于执行规约期时，该对象能够响应或发送消息，执行对象或活动。

当一个对象不处于执行规约期时，该对象不做什么事情，但它是存在的，等待新的消息执行规约它。

若调用一个对象的另一个操作，第二个执行规约符号画在第一个符号稍微靠右的位置。

递归？



4、消息

消息是对象之间的通讯的规格说明，这样的通讯用于传输将发生的活动所需要的信息——控制信息（如调用）和所使用的数据的规格说明。

一个消息会调用另一个对象的操作，调用本对象的操作，向另一个对象发送一个信号，创建或者撤消一个对象（可以自己销毁自己），还可能向调用者返回一个结果。

把消息表示为从一个对象生命线到另一个对象生命线的一个水平实线箭头，即从源对象指向目标对象，以触发目标对象中的特定操作。对于对象到自身的消息，箭头就从同一个对象符号开始和结束。

用消息（操作或信号）的名字及其参数值或者参数表达式标示箭头。

用如下种类的箭头表示不同种类的通讯：

同步消息 

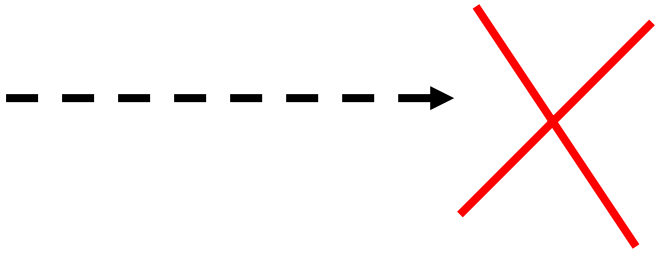
一般把它用于普通的过程调用。在外层控制恢复之前，要完成整个嵌套序列。通常把它用于普通的过程调用。

同步消息返回 

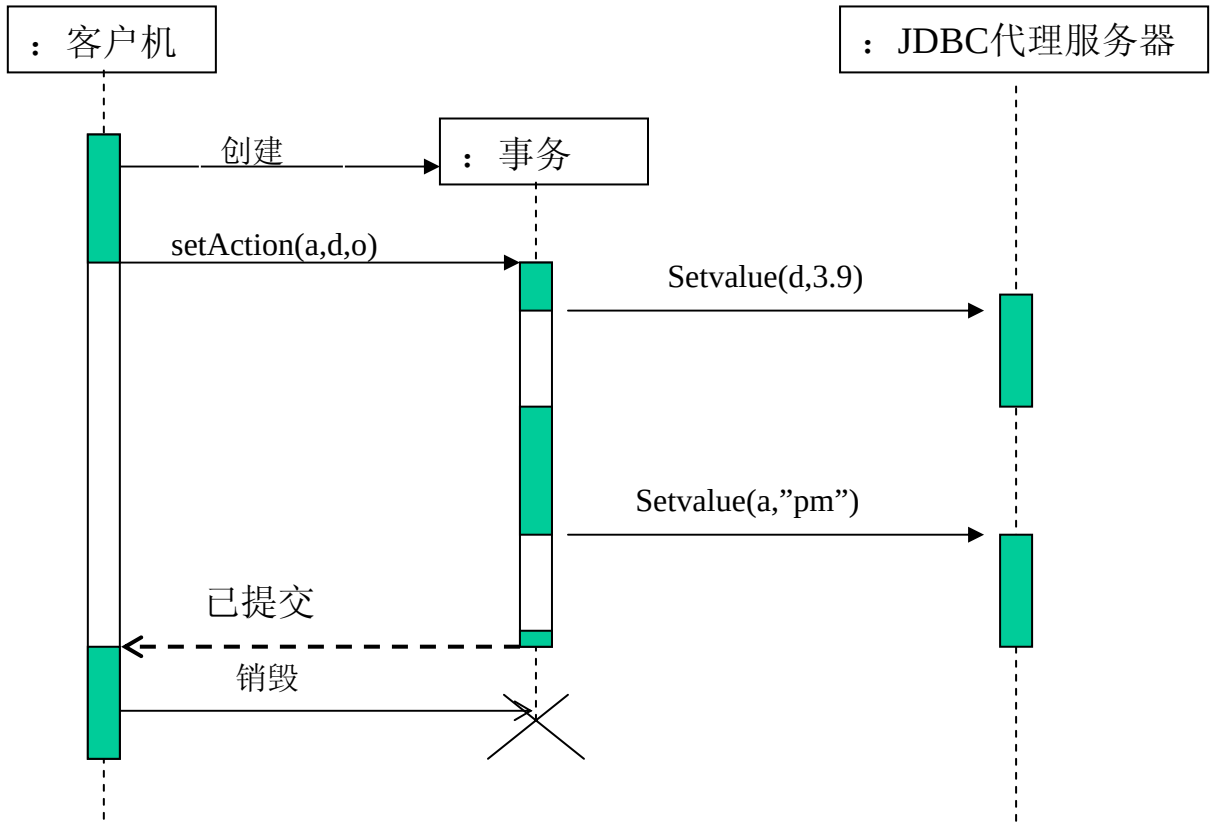
用它显式地表示从过程调用的返回。

在控制的过程流中，可以省略返回箭头（暗示执行规约结束），这是要假设每个调用在任何消息后都有一个配对的返回。

若需要标识返回值，则要显式地把它标示在返回的箭头上。



对象的创建与销毁



在过程性代码的情况下，一个执行规约表示在一个对象中一个过程是活动的，或者它的从属过程（可能在其它的对象中）是活动的持续时间。

换句话说，可以在一个特定的时间看到所有活动着的嵌套过程执行规约。

异步消息 

用它表示异步通讯，也即发送者发出消息后，立即继续执行中的下一步，不进行等待。

异步消息返回

若请求方发了一个异步消息，且接收方响应它后要返回信息，则使用另一个异步消息。

注意：消息与对消息的响应。

在多数情况下，收发消息的时间是可以忽略的。通常消息箭头都画成水平的。

这表示发送消息所需要的持续时间是“原子的”（也即，它与交互的粒度相比是短暂的，并且在传送消息的中间不能发生任何事情）。这在很多计算机中都被假设是正确的。

如果需要表示收发消息间的时间差，有三种方法：

（1）可以在图中使用约束，用于指示时间间隔。可以用消息名和经过规定的函数书写计时表达式，如下图的“b.receiveTime – a.sendTime<1 分钟”。

（2）若要在图中显式地表示时间差的数值，还可以通过构造标记来指明，如下图右下角所示。

（3）如果需要表示发送消息是需要时间的，还可把消息箭线向下倾斜，使箭线头部在尾部下方，表示消息需要一段时间到达。

消息分支

把分支画成从一个点出发的多个箭头，每个箭头由监护条件标示。依据监护条件是否互斥，这个结构可以表达条件或者并发。

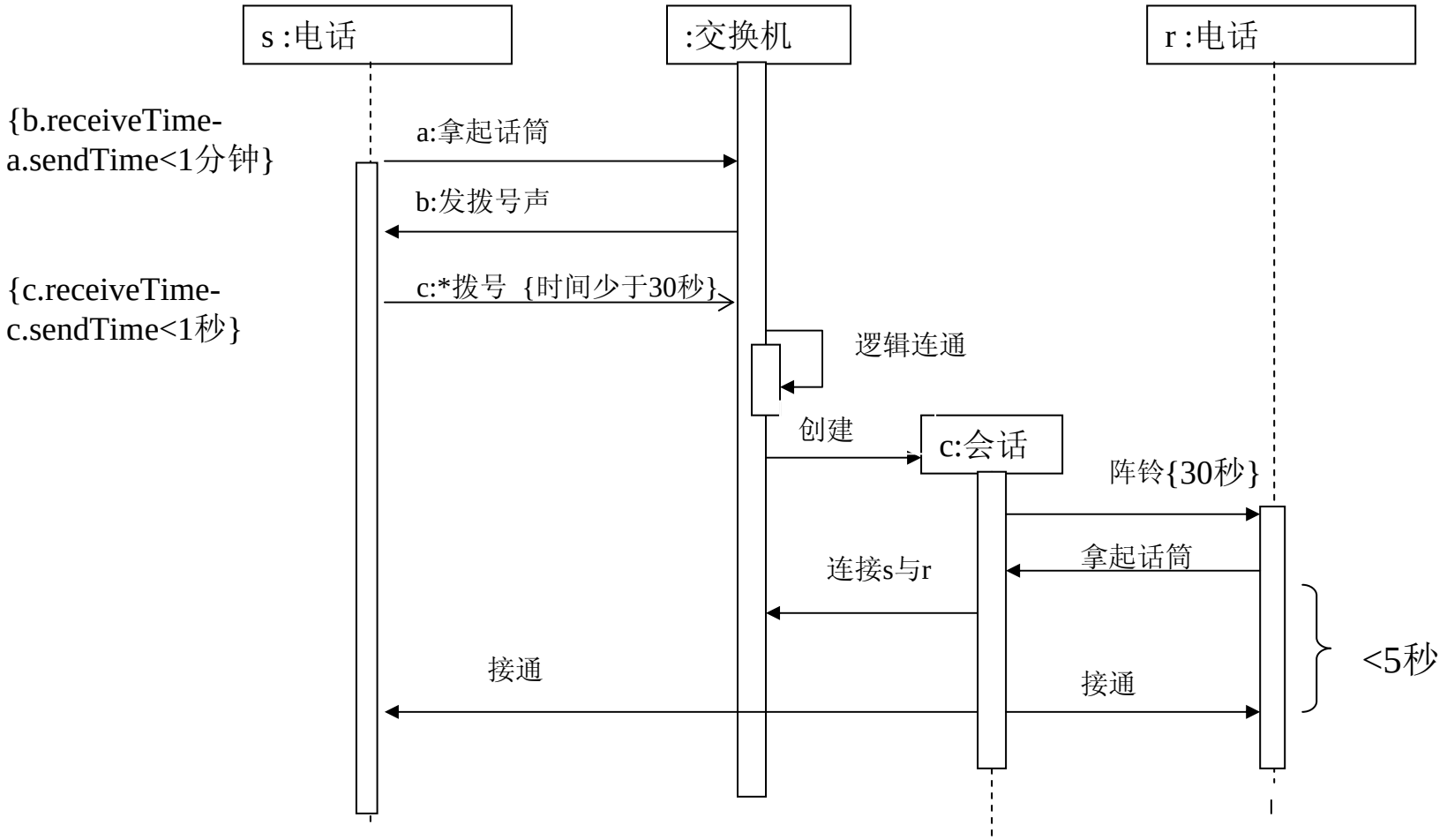
消息循环

标以持续的条件：　　*[条件]

方框围起来的区域为重复的。

可通过使用预定的接口或消息设施（如分布式系统中的中间件）来实现和管理各软件部件间的请求通信。

例题：打电话

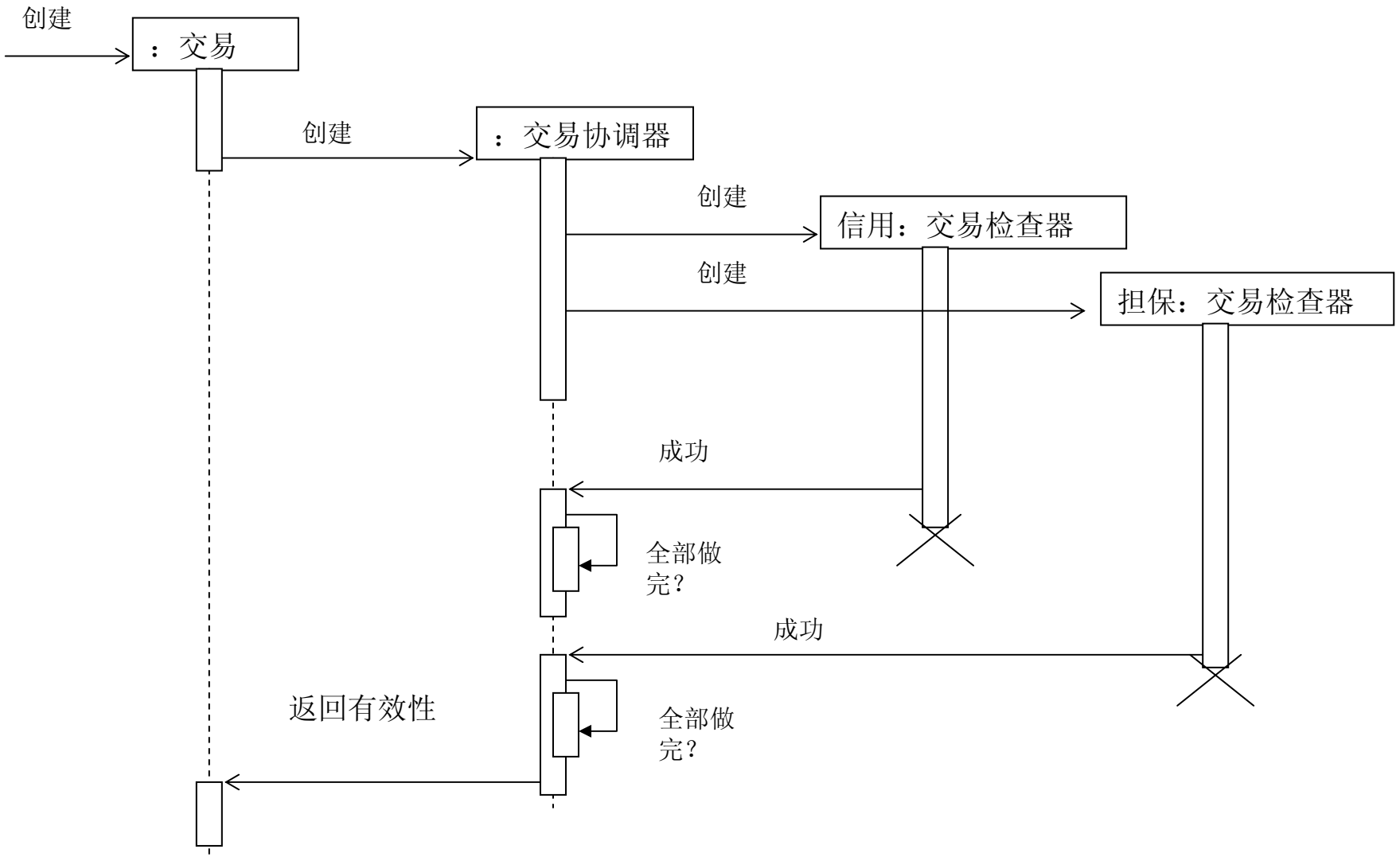


把电话机换为打电话的人如何？

问题：时间超过30秒的情况没说明

会话对象没有说明计费等情况

例题 银行系统的交易验证



5、信号

信号是对象之间的**异步通讯**的规格说明。

信号名 ‘(‘用逗号分隔的参数列表‘)’

从一个对象可以向另一个对象或对象的集合发送信号。例如消息广播。

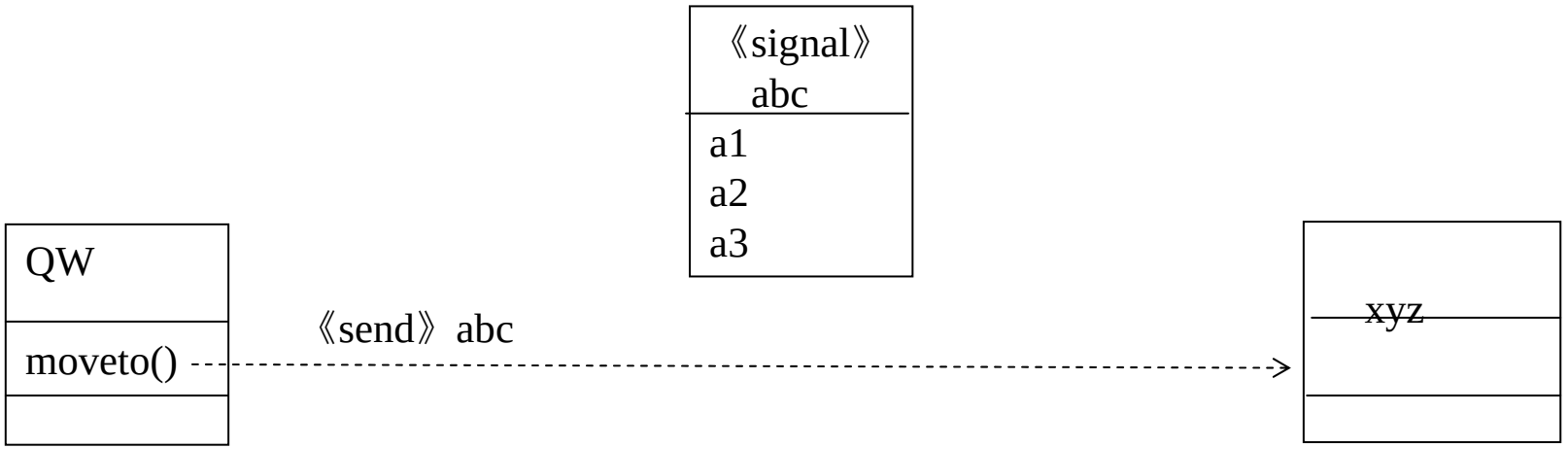
发送者在发送信号时，要实例化其参数。

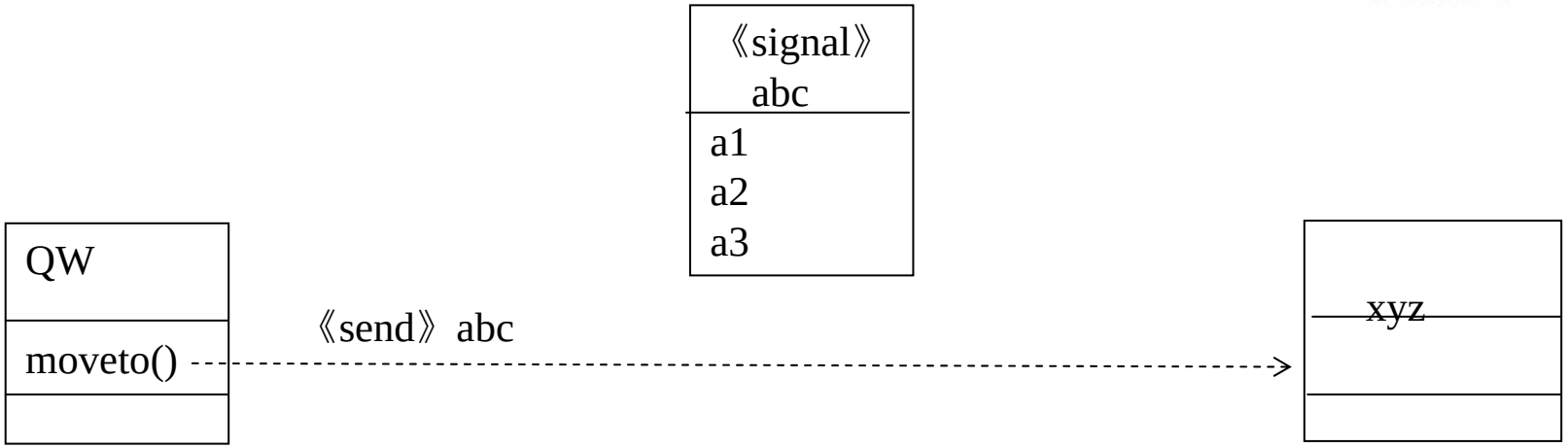
对于接收者来说，它收到的是一个事件。

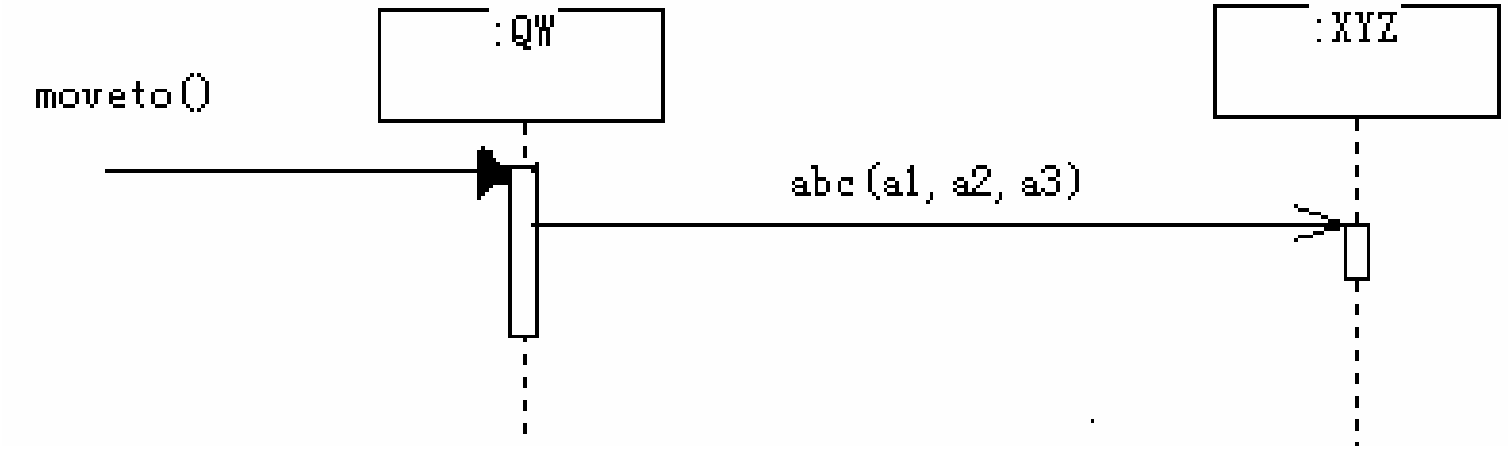
在类图中，在类符号上用关键字<<signal>>声明信号。把参数说明为属性。信号没有操作。

在类的描述模板中，要指定所能接收的信号。

通常用信号对异常情况建模。





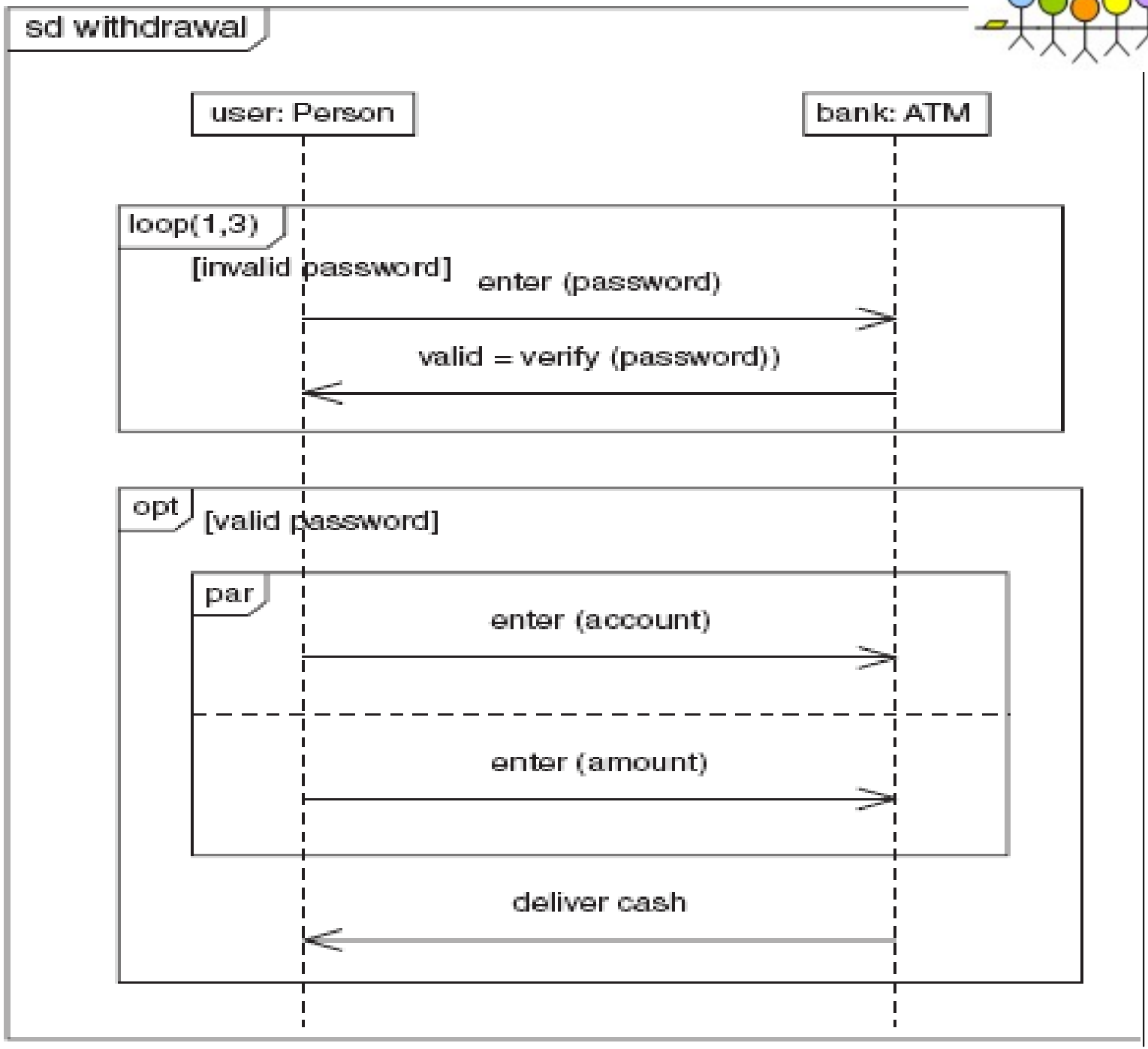


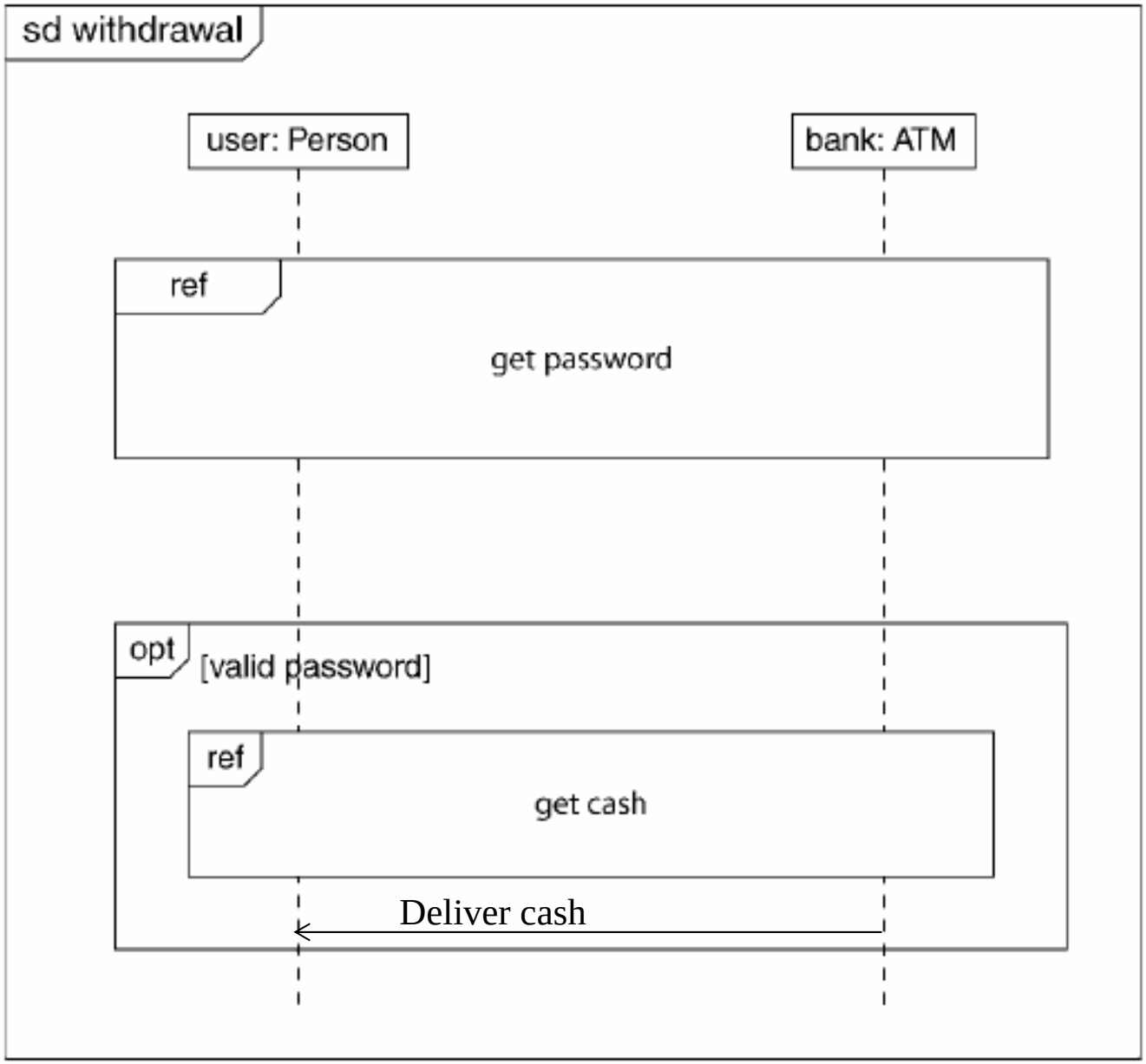
二、 顺序图中的结构化控制

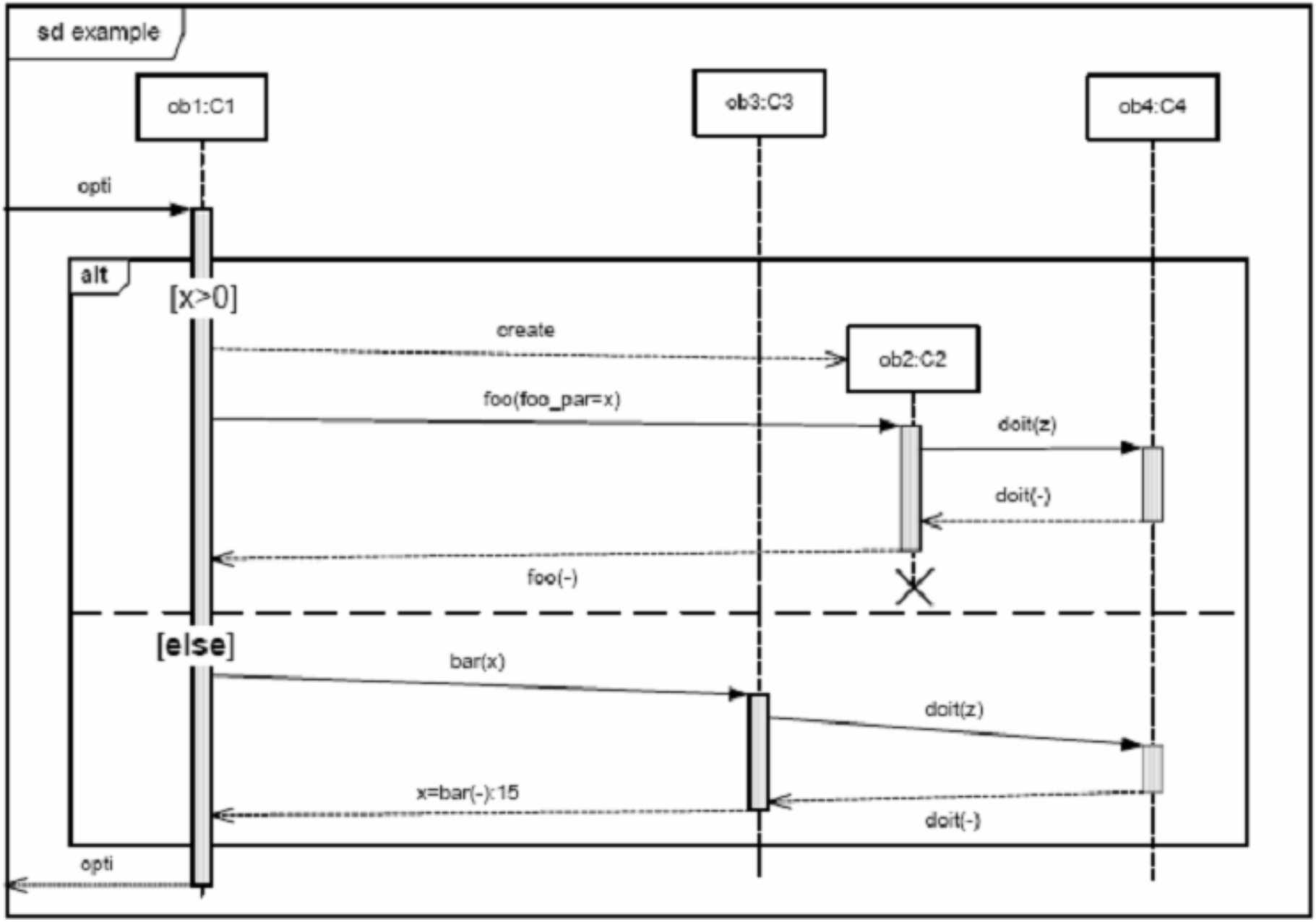
序列性的消息能很好地说明单一的线性的序列，但是我们通常需要展示条件和循环。有时候我们想要展示多个序列的并行执行。在顺序图中用结构化控制操作符能展示这种高层控制。

为了表示顺序图的边界，可以把顺序图用一个封闭的矩形包围起来，并在矩形的左上角放一个小五边形。在这个小五边形内先写上sd，再后面写出图的名字。

对每个子顺序图加上一个矩形区域作为外框，再在其左上角放一个小五边形，在这个小五边形内写上用来表明控制操作符的类型的文字。







可选执行 标签是opt。如果控制进入该操作符标识的交互区域，监护条件成立，那么执行该交互区域。监护条件是一个用方括号括起来的布尔表达式，它要出现在交互区域内部第一条生命线的顶端，在其中可以引用该对象的属性。

条件执行 标签为 alt。用水平虚线把交互区域分割成几个分区，每个分区表示一个条件分支并有一个监护条件。如果一个分区的监护条件为真，就执行这个分区，但最多只能执行一个分区。如果有多于一个监护条件为真，那么选择哪个分区是不确定的。若没有应对措施，在模型中要避免这种情况。如果所有的监护条件都不为真，那么控制流将跨过这个交互区域而继续执行。其中的一个分区可以用特殊的监护条件[else]，这意味着如果其他所有区域的监护条件都为假，就执行该分区。

并行执行 标签是 par。用水平虚线把交互区域分割为几个分区。每个分区表示一个并发计算。当控制进入交互区域时并发地执行所有的分区；在并行分区都执行完后，那么该并行操作符标识的交互区域也就执行完毕。每个分区内的消息是顺序执行的。需要指出的是，并发并不总是意味着物理上的同时执行。并发其实是说两个动作没有协作关系，而且可按任意次序发生。如果它们确实是独立的动作，那么它们还可以交叠。

循环（迭代）执行 标签是 loop。在交互区域内的顶端给出一个监护条件。只要在每次迭代之前监护条件成立，那么循环主体就会重复执行。一旦在交互区域顶部的监护条件为假，控制就会跳出该交互区域。

三、 建立顺序图

步骤：

■ 按照当前交互的意图，如系统的一次执行,或者一组对象（包括参与者实例，以下不再明确地提及参与者实例）之间的协作，详细地审阅有关材料（如有关的用况），设置交互的语境，其中包括可能需要的那些对象。

■ 通过识别对象在交互中扮演的角色，在顺序图的上部列出所选定的一组对象（应该给出其类名），并为每个对象设置生命线。通常把发起交互的对象放在左边。

■ 对于那些在交互期间要被创建和撤销的对象，在适当的时刻，用消息箭线显式地予以指明。

■ 决定消息将怎样或以什么样的序列在对象之间传递。

通过首先发出消息的对象，看它需要哪些对象为它提供操作，它向那些对象提供操作。追踪相关的对象，进一步做这种模拟，直到分析完与当前语境有关的全部对象。

如果一个对象的操作在某个执行点上应该向另一个对象发消息，则从这一点向后者画一条带箭头的直线，并在其上注明消息名。用适当的箭头线区别各种消息。

■在各对象下方的生命线上，按使用该对象操作的先后次序排列各个代表操作执行的棒形条（执行规约）。若出于某种目的要简化顺序图，可不画棒型条，或者针对一个对象只用一个棒型条代表其上的所有操作的执行。
■两个对象的操作执行如果属于同一个控制线程，则接收者操作的执行应在发送者发出消息之后开始，并在发送者结束之前结束。不同控制线程之间的消息有可能在接收者的某个操作的执行过程中到达。
■如果需要，也可以对对象所执行的操作的功能以及时间或空间约束进行描述。
■ 如果需要，可使用结构化控制。

用途

- ❑帮助分析员对照检查每个用况中描述的用户需求，是否已经落实到一些对象中去实现。提醒分析员去补充遗漏的对象类或操作。
- ❑通过对一个特定的对象群体的动态方面建模，深刻地理解对象之间的交互。
- ❑帮助分析员发现哪些对象是主动对象

课后作业

1、 使用信用卡可以在AMT机上进行取款，针对一次取款，建立类图、顺序图。注意ATM机是与银行连网的。

要求：

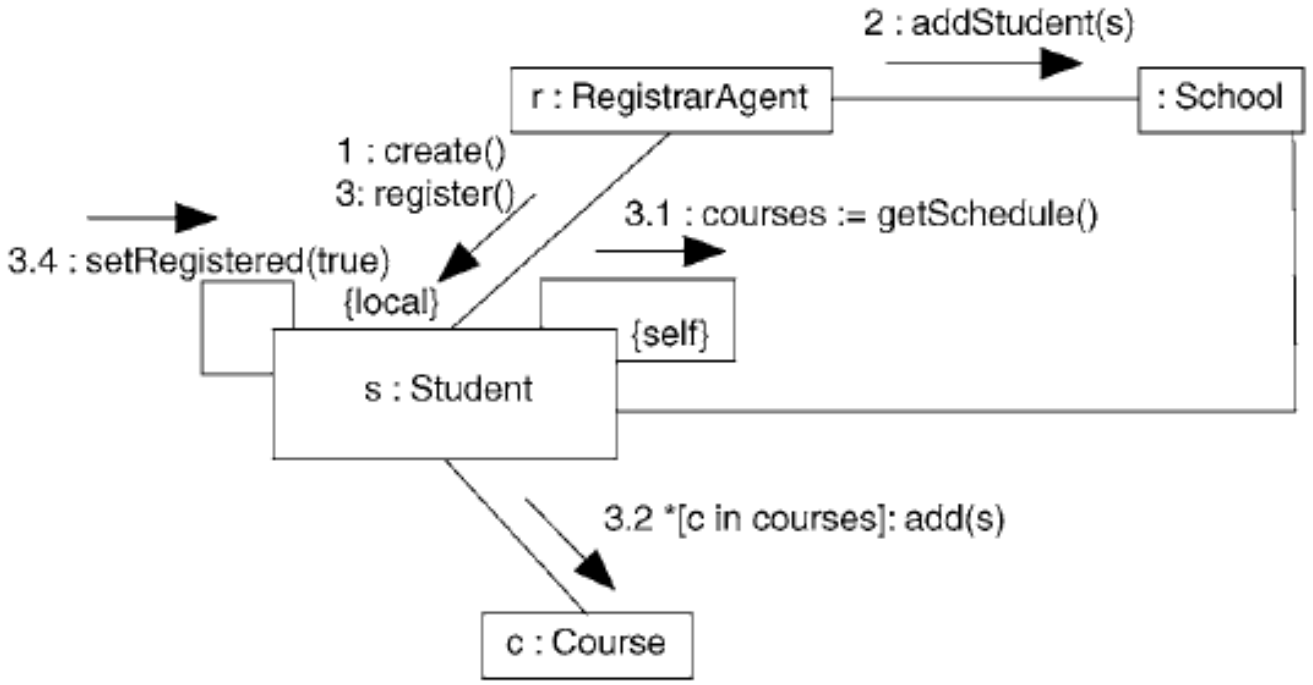
（1）绘制一个类图（不要过于复杂）

（2）绘制顺序图

2、几台计算机公用一台打印机，打印机由打印服务器管理，请建立顺序图。

5.2 通讯图

简言之，通讯图表示围绕着对象角色以及对象角色之间的链所组织的交互。



与顺序图不同:

1)通讯图表示扮演不同角色的对象之间的关系。

2)通讯图不表示作为单独维度的时间，所以交互的顺序和并发进程必须用顺序数决定。

顺序图表示执行消息的显式顺序，最好用于描述实时系统和复杂的场景。

一、 概念与表示法

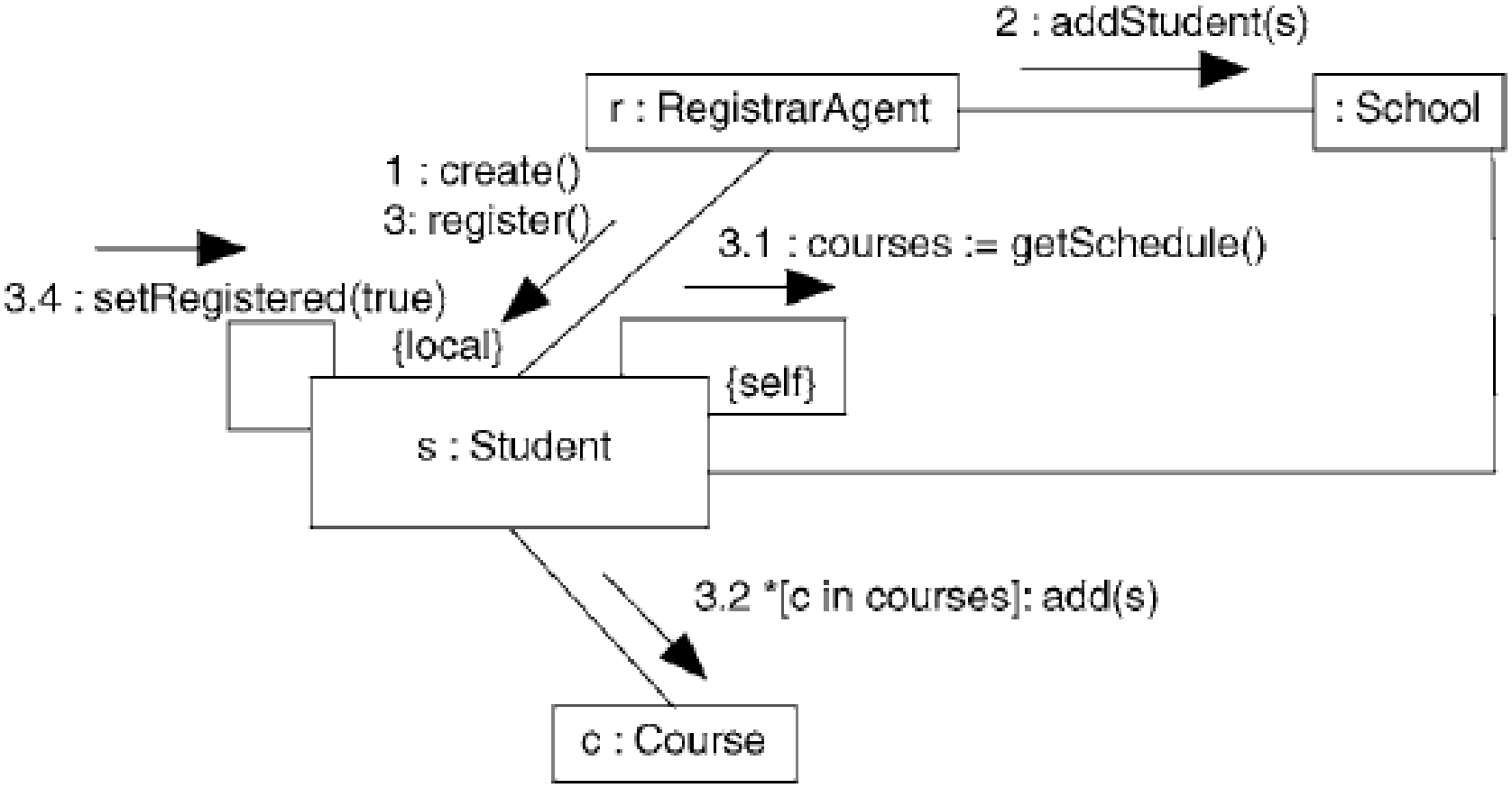
通信图是一种强调发送和接收消息的对象结构组织的交互图，展示围绕对象以及它们之间的连接器而组织的交互。

连接器是由关联实例化的链以及通过过程参数、局部变量或全局变量而产生的对象之间的临时连接。

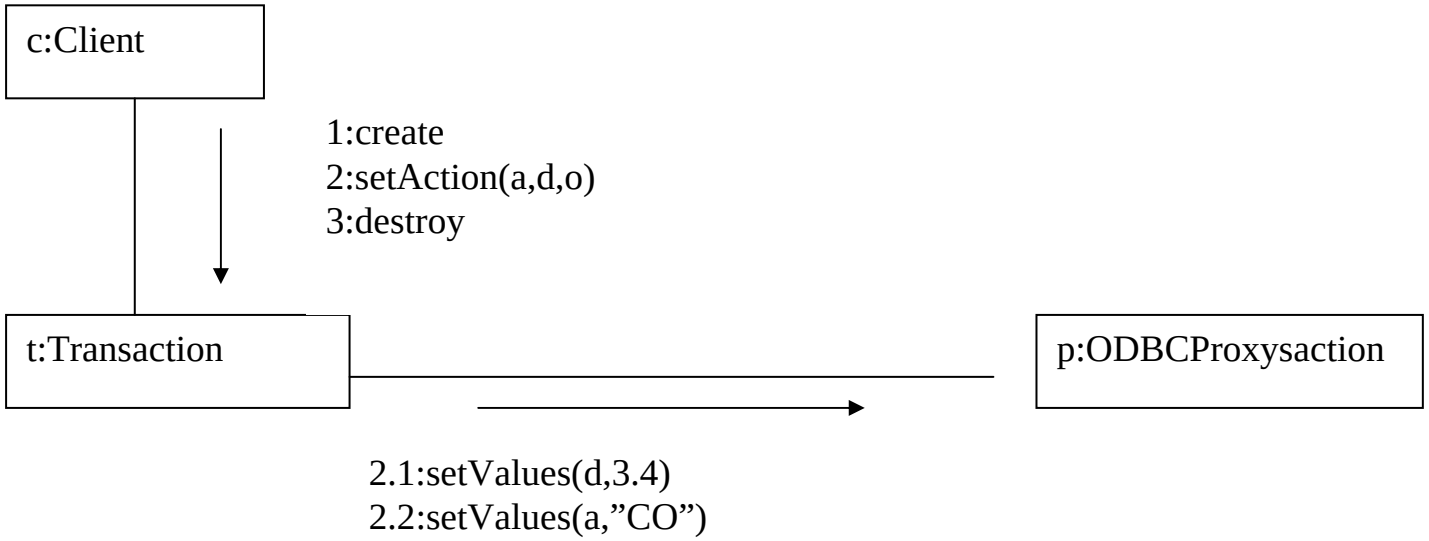
通讯图由对象（参与者）、连接器以及连接器上的消息构成。这些概念及表示法与前述中的都完全相同。

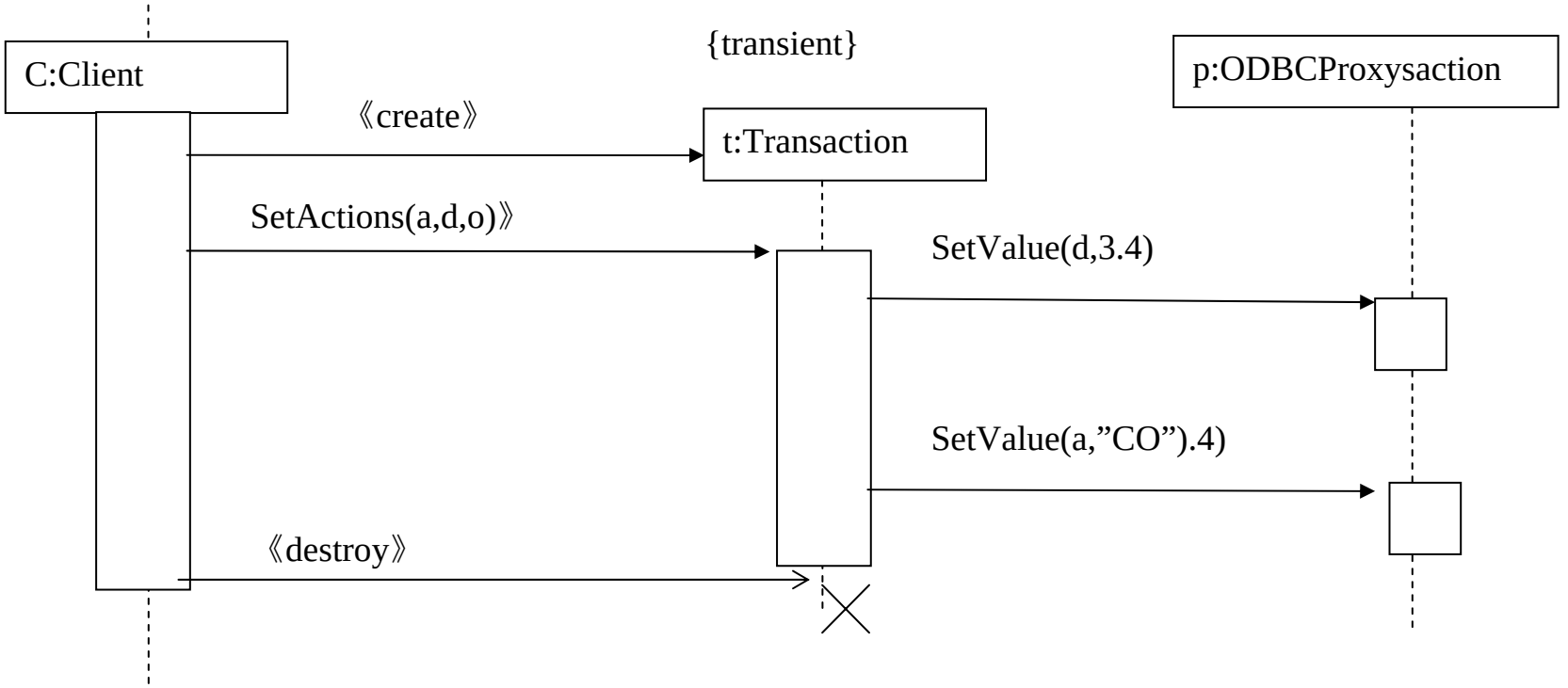
为表示一个消息的时间顺序，可以给消息加一个数字前缀（从1号消息开始），在控制流中，每个新的消息的顺序号单调增加（如2，3等等）。为了显示嵌套，可使用带小数点的号码（1表示第一个消息；1.1表示嵌套在消息1中的第一个消息；1.2表示嵌套在消息1中的第二个消息；等等）。

嵌套可为任意深度。要注意的是，沿同一个链，可以显示多个消息（可能发自不同的方向），并且每个消息都有唯一的一个顺序号。



顺序图和通讯图在语义上是等价的，它们可以从一种形式的





二、建立通讯图

应遵循如下策略建立通讯图：

■ 设置交互的语境，不管它是一个系统、子系统、类，还是用况的脚本。

■ 通过识别对象在交互中扮演的角色，设置交互的场所。将它们作为图的顶点放在通讯图中，较重要的对象放在图的中央，然后放置邻近的对象。

■ 若对象之间可能要传递消息，说明对象之间的连接器。

■ 从引起这个交互的消息开始，然后将随后的每个消息附到适当的连接器上，恰切地设置其顺序号。并用带小数点的编号来显示嵌套。

■如果需要展示消息的循环或分支，就是使用相应的表示法。

■ 如果需要说明时间或空间约束，则用时间标记修饰每个消息，并附上合适的时间和空间约束。

像顺序图一样，建议一个单独的通讯图只描述一个控制流。一般来说，可能要建立许多通讯图，其中一些是基本的，另一些描述的是可选择的路径或例外情况。

可以使用包来组织一组通讯图，并给每个图起一个合适的名称，以便与包中其它图的相区别。

5.3 活动图

活动图可用于对业务过程和操作的算法建模

一、 概念与表示法

活动图显示从活动到活动的流。下面详述有关的概念。

1、动作和活动

动作（action）是行为规约的基础单元，用以描述系统中的活动，它是原子的和即时的。动作是原子的，是指在与状态相关的抽象层次上，动作是不可间断的；动作时即时的，是指动作执行的时间是可忽略不计的。

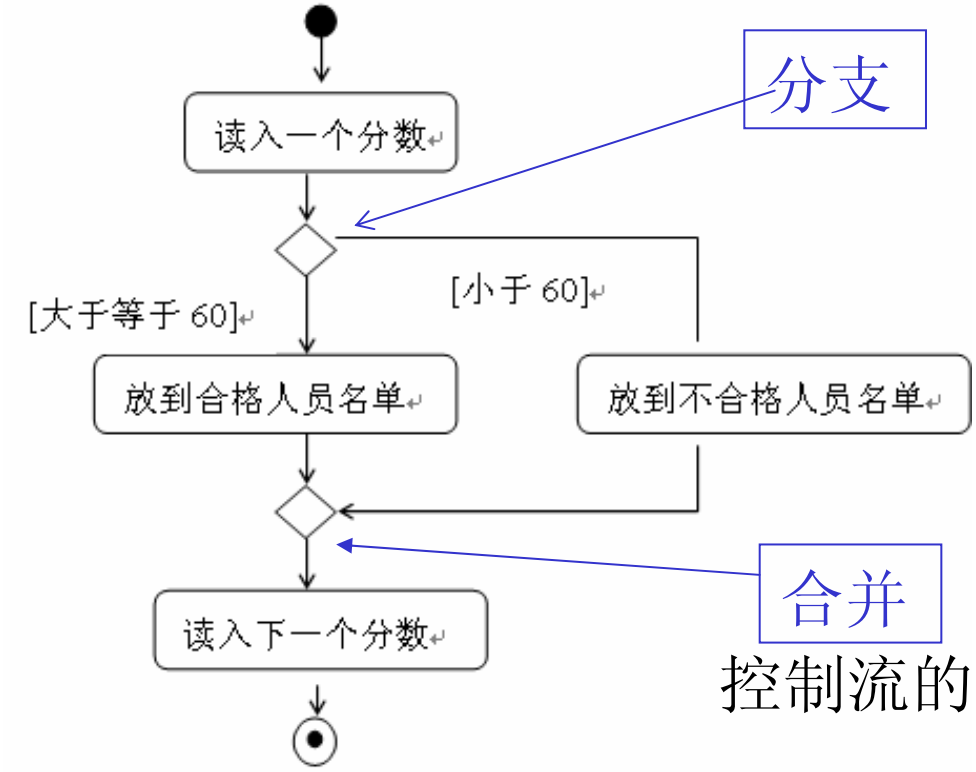
如调用另一个操作，发送一个信号，创建或撤销一个对象，或者某些纯计算（例如对一个表达式求值），都是一个动作。

活动（activity）是由一系列的动作构成的（也称为动作表达式），用于描述系统的一项行为，它由动作和其他活动组成。

在活动图中，动作和活动均具有图形表示法，且是一样的

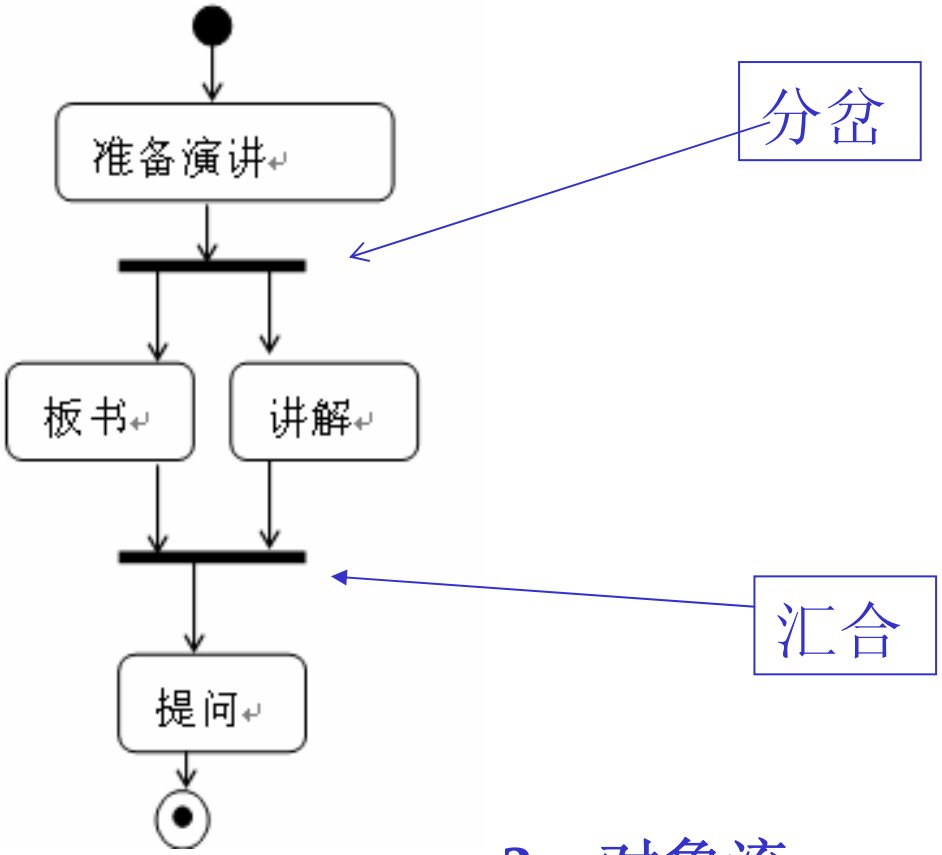
2、控制流

当动作或活动结束时，马上进入下一个动作或活动。一系列的动作和活动的执行构成了一个控制流。在图形上，用一个箭头表示从一个动作或活动到下一个动作或活动的转移



控制流的分支与合并

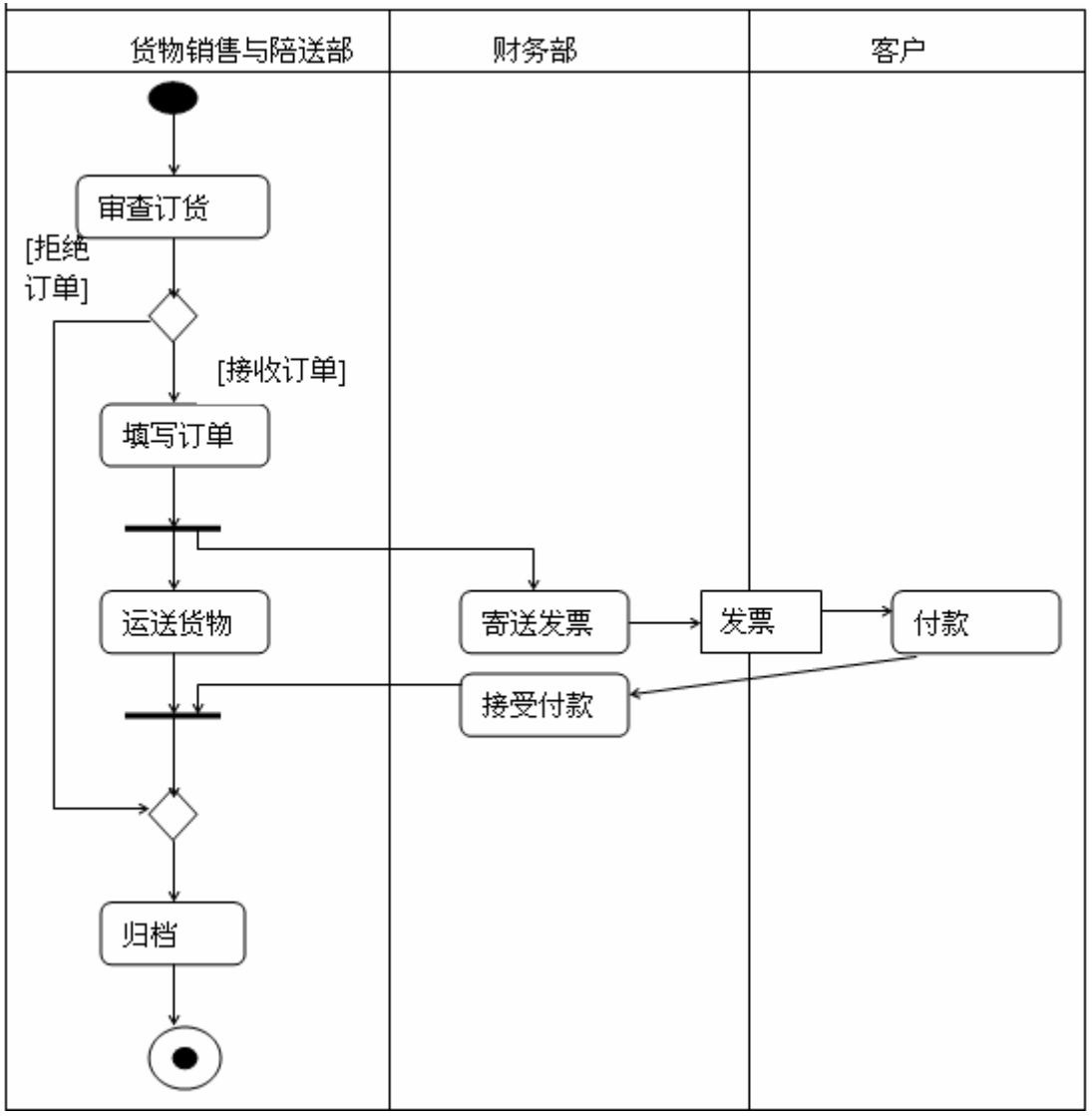
控制流也可以是并发的。用同步条表示并发控制流的分岔和汇合。



3、对象流

4、泳道

在对业务过程建模时，可以把活动或动作分成组，每组由特定的履行者来执行。履行者可为人员、组织或其他业务实体。把每个组分别称为一个泳道。



5、活动图

活动图是展示从动作或活动间的控制流和对象流的图，其中的结点描述动作或活动，边描述控制流或对象流。

一般用它对计算过程中的步骤建模，也可用它对步骤间的数值的流动建模。上述的计算过程可为业务过程，也可为操作的算法。注意，顺序图强调对象间的控制流，而活动图强调的是活动间的控制流。

对于活动图中一个活动结点，可用另一张活动图（子活动图）进行详述。

二、 建立活动图

对业务过程建模，要遵循如下的策略：

■设置业务过程的语境。即要考虑在特定的语境中要对哪些业务的履行者和业务实体建模。
■考虑为每个重要的业务的履行者建立一个泳道。
■建立初始状态和终止状态，并识别该业务过程的前置条件和后置条件。
■从初始状态开始，说明随着时间发生的动作或活动，并在活动图中表示它们。
■如果涉及到重要的对象，则把它们也加入到活动图中。如果有必要，可展示对象的属性值和状态。
■连接这些动作和活动结点的流。
■如果需要，使用分支和合并来描述条件路径和迭代，使用分岔和汇合来描述并发的动作或活动流。
■针对活动建立子活动图。

作为用况文字描述的补充

对一个操作建模时，应遵循如下策略：

■收集该操作所涉及的事物，包括操作的参数、可能的返回类型、它所属于的类以及某些邻近的类的特征。

■识别操作的前置条件和后置条件以及操作所属的类在操作执行期间必须保持的不变式。

■从该操作的初始状态开始，按照时间顺序设立活动或动作，并在活动图中将它们表示出来。

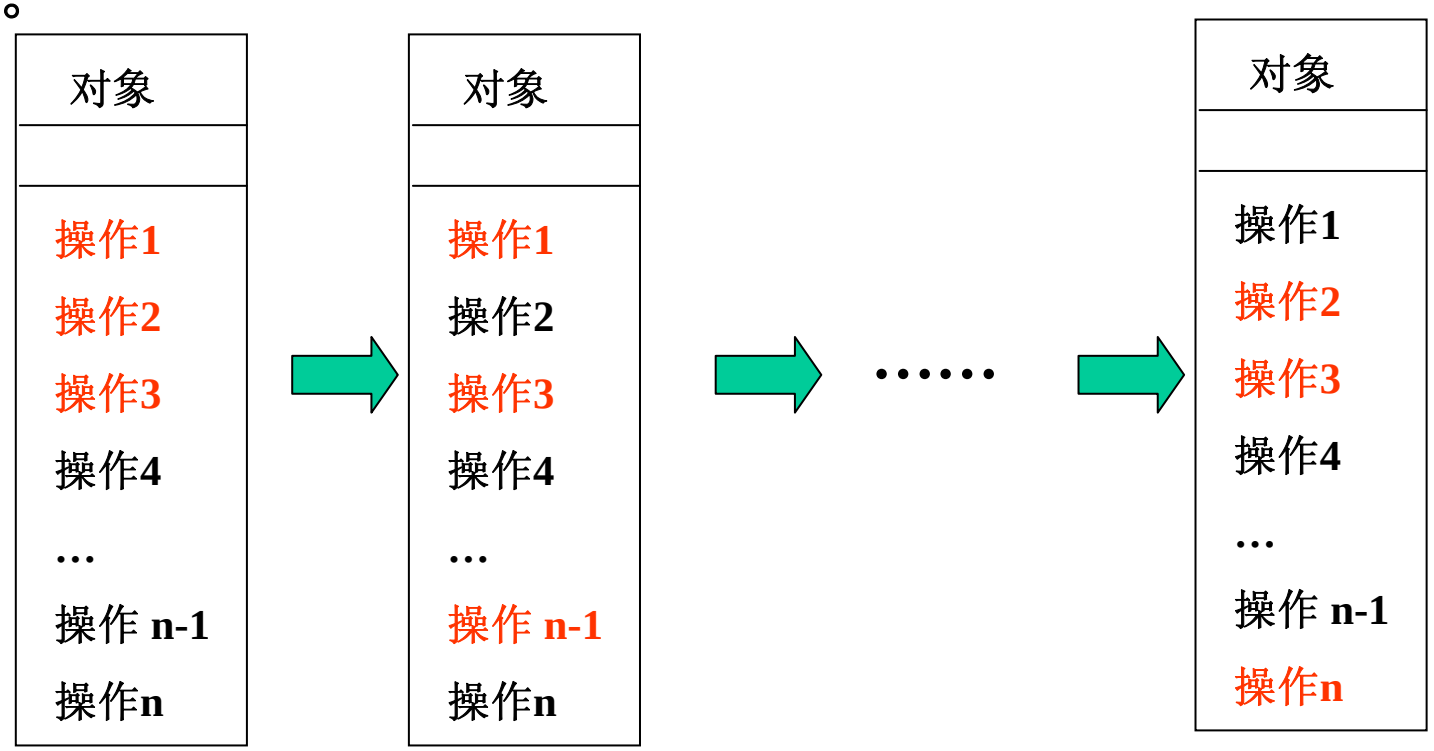
■如果需要，使用分支和合并来描述条件路径和迭代。

■仅当这个操作属于一个主动类时，才在必要时用分岔和汇合来描述并发的控制流。

在OOA阶段，仅用活动图对关键的复杂操作进行建模，用以展示关于算法的一些信息。除非想直接从模型生成代码，即使在OOD阶段并也不要求用活动图对每个操作的算法都建立模型。

5.4 状态机图

对在不同的状态下发挥不同作用的复杂对象，可建立状态机图。



状态1　　状态2　　状态m

可具有状态机图的事物：人员、系统、控制器、事务、设备等。

对事物所处的状态及其变迁建模需要考虑：

- 事物在其生命周期中经历了不同的状态；
- 在特定的时间，一个事物精确地位于一个状态，发挥特定的作用；
- 在现实世界存在着引起事物的状态发生变化的事件；
- 事物在其状态间按次序转化；
- 事物从一个状态到另一个状态的转化通常是即时的。
- 当事件发生时，事物可能需要采取一些动作。

事物的一般生命周期形式

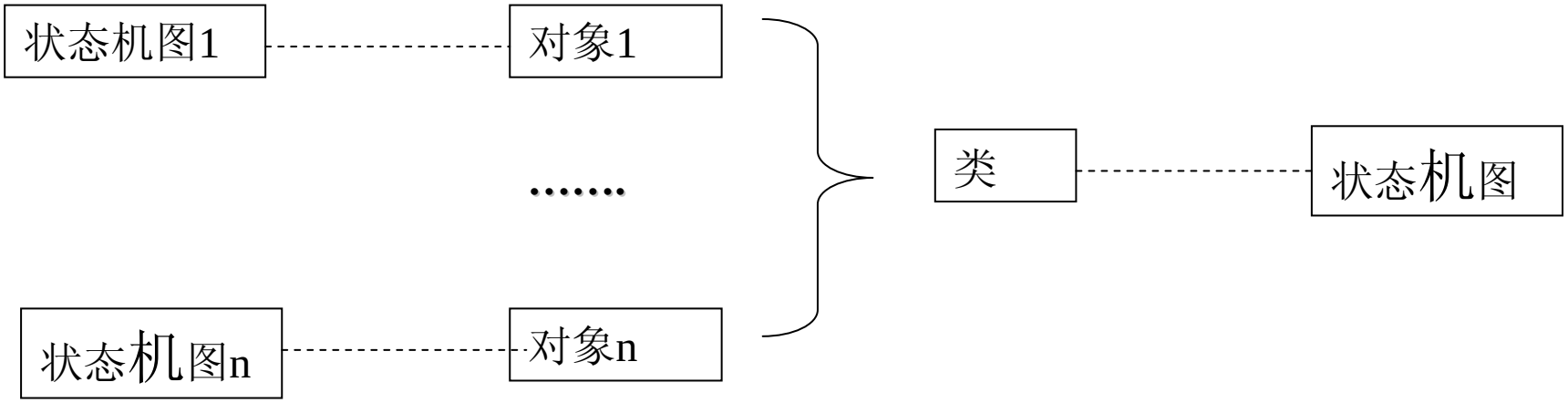
目前主要有两种对生命周期进行建模的形式：

■周期性生命周期

如：飞机、微波炉

■出生-死亡生命周期

若一个对象具有明确的生命期阶段（状态），且需要通过状态分析对其复杂性进行深刻地认识，就可为它建立一个状态机图。



在实际系统中，无法建立全系统的状态机图。

什么是对象的状态？

《对象技术词典》的另一种定义：

对象或者类的整体行为（例如响应消息）的某些规则所能适应的（对象或类的）状况、情况、条件、形式或生存周期阶段。

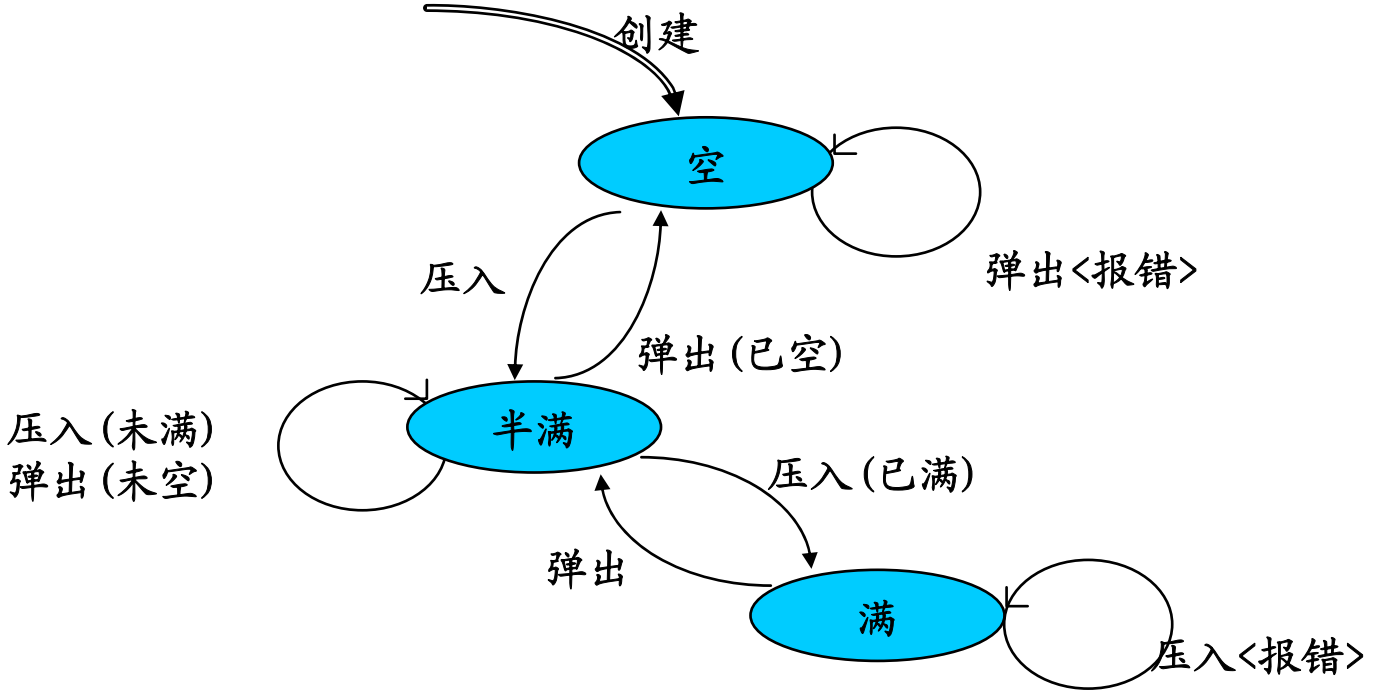
——按某种规定，按对象的属性值划分等价类

例1：一个容量为1000的栈，需要区分几种状态？

服务 \ 状态	空	半满	满
压入	可执行	可执行	不可执行
弹出	不可执行	可执行	可执行

在此例中,每一种状态是一组使对象呈现共同行为规则的属性值组合。

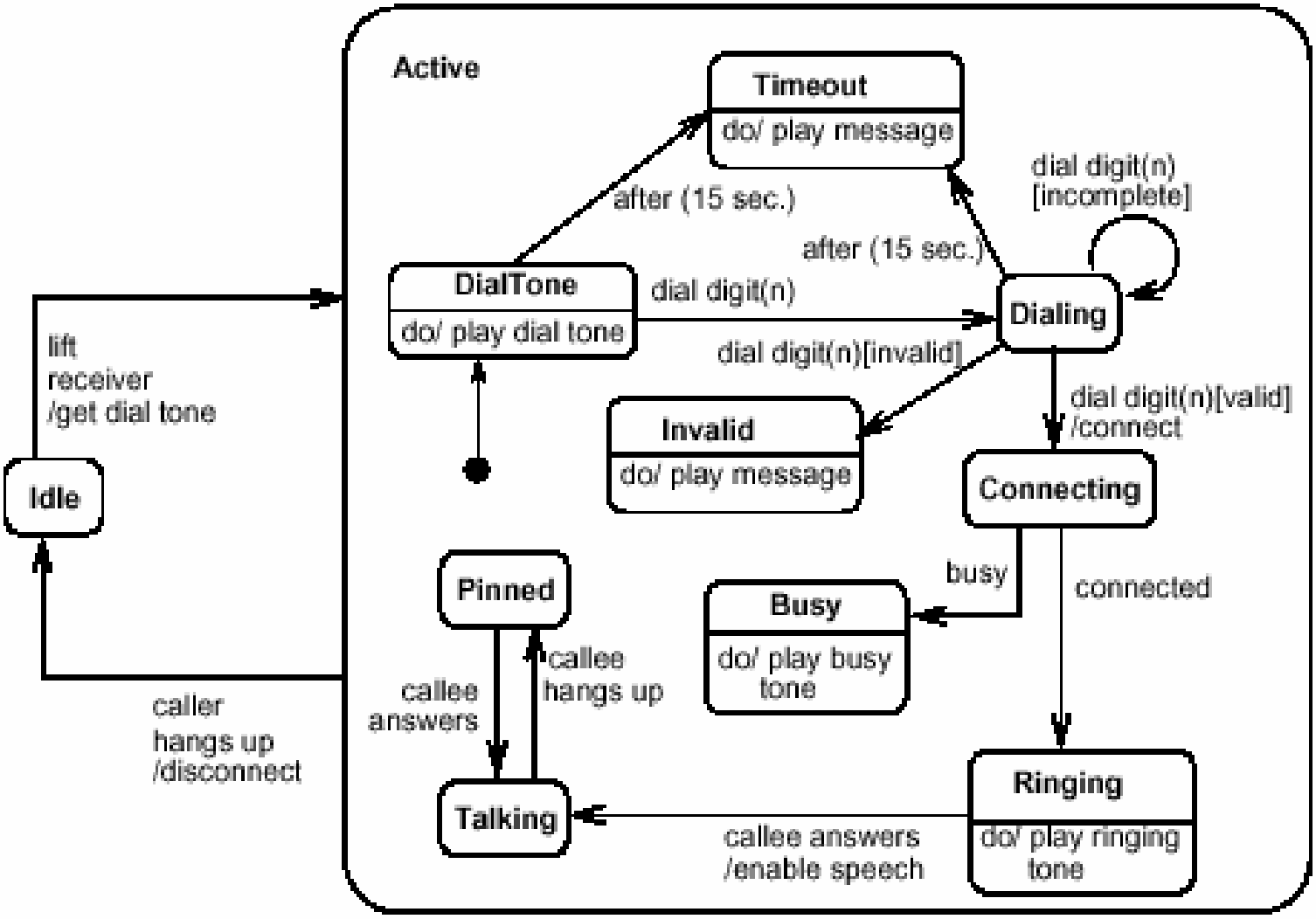
状态转换示意图



例2：为“设备”对象设立一个属性，名为“状态”
属性值：关闭、待命、运行、故障等。

在这里，“状态”是一个专门设置的属性，它的值反映了实际事物的状态。

状态机图示例


Active
Timeout
do/ play message
after (15 sec.)
dial digit(n)
[incomplete]
after (15 sec.)
DialTone
do/ play dial tone
dial digit(n)
Dialing
dial digit(n)[invalid]
lift
receiver
/get dial tone
Invalid
do/ play message
dial digit(n)[valid]
/connect
Idle
Connecting
Pinned
busy
Busy
do/ play busy
tone
connected
callee
hangs up
callee
answers
caller
hangs up
/disconnect
Ringing
Talking
callee answers
/enable speech
do/ play ringing
tone

一、 概念与表示法

1、事件

从一般意义上讲，事件是指在时间和空间上可以定位并具有实际意义、值得注意的所发生的事情。

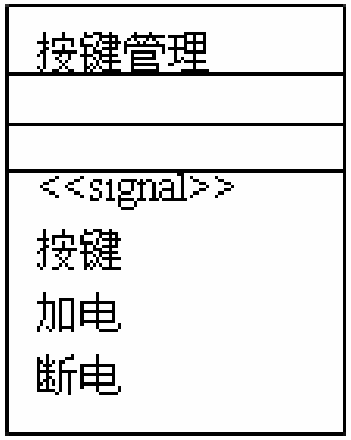
在OO中，事件是对一个可观察的事情的规格说明，这种事情的发生可以引发状态的转换。

事件可以分为多种:

1) 信号事件 一个对象对一个信号实例（在不引起混淆的情况下，以下简称信号）的接收，导致一个信号事件。

在一个类的符号中加一个附件的信号栏，列出其能接收的信号。

信号可以作为状态机中的状态转换上的动作被发送，或者作为交互中的一个消息被发送。



2) 调用事件

对操作的调用的接收（这样的操作由接收事件的对象实现）

调用事件一般来说是同步的

3）时间事件

在指定事件（经常是当前状态的入口）后，经过了一定的时间或到了指定日期/时间，导致一个时间事件。

时间经历事件用后跟有计算时间量的表达式的关键词“after”表示，比如“after （5 秒)”或者”after （从状态A退出后经历了10秒)”。如果没指明时间起始点，那么从进入当前状态开始计时。

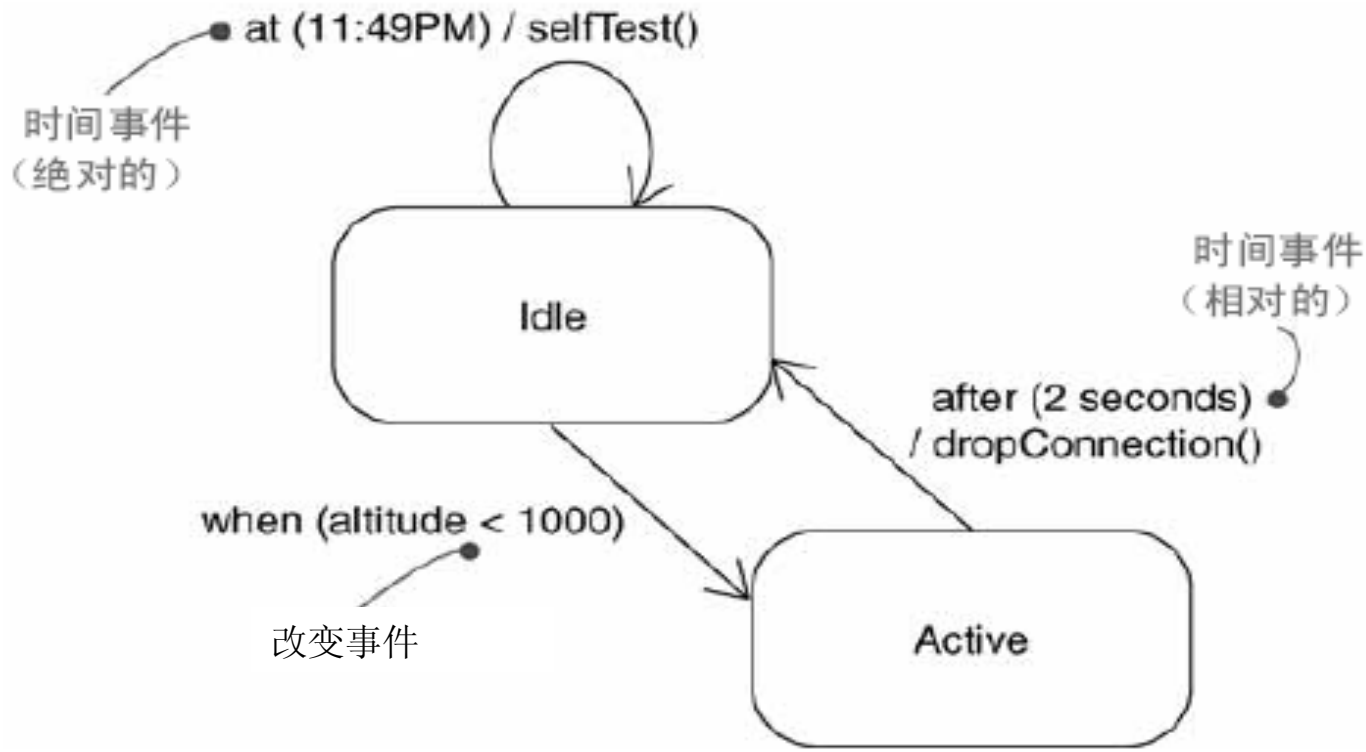
使用关键字at 来指出在某个绝对时间点上发生的时间事件。例如，时间事件at(1 Jan 2005, 12:00 UT)指出该事件发生在格林尼治时间2005 年1 月1 日的中午12点。

4) 改变事件（条件变为真事件）

用布尔表达式描述的指派条件变为真，就导致了一个改变事件。无论表达式的值何时由假变成真，事件都发生。与改变事件关联的布尔表达式的值变成真时事件发生一次，即使之后布尔表达式的值变为假，产生的事件仍将保持，直到它被处理为止。

用后有跟布尔表达式的关键词“when”表示变为真的条件，例如

when (altitude < 1000)　可以把其看作是连续测试条件，直到它为真。



事件的格式：

事件名　　‘(‘用逗号分隔的参数列表‘)’

参数的格式如下：

参数名　　‘:’ 类型表达式

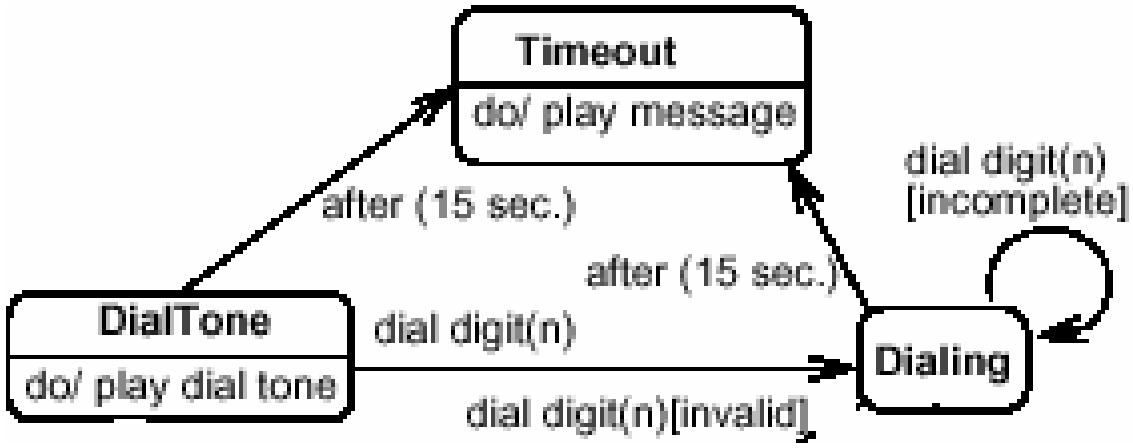
参数值由监护条件和动作表达式使用.

事件出现的位置：状态内，转换上。

通常事件后面还跟有一个监护条件（布尔表达式）， 当事件出现要触发转换时,对它求值：

——如果表达式取值为真，则触发转换；

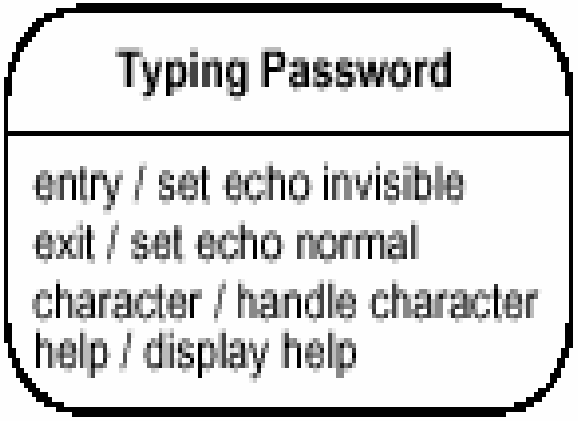
——如果为假，则不触发转换，而且如果没有其他的转换被此事件所触发，则该事件丢失。



2、状态

状态是一个对象的生命期的一个阶段，在该阶段中该对象要满足一些特定的条件、执行特定的活动或等待某个（些）事件。

把状态表示成四角均为圆角的矩形，并把状态的名称放在其中。



a) 名称分栏

在该分栏中放置状态名。没有名称的状态是匿名的，但同一张图中的匿名状态是各不相同的。

b) 内部转换分栏

用该分栏给出对象在这个状态中所执行的内部动作或活动的列表。各表项的表示法的基本格式为：

事件名[‘（’用逗号分隔的参数表‘）’][监护条件]‘/’ 动作表达式

用户可以自己对对事件进行命名，只是entry、exit和do这三个保留字外，因为UML已经为它们规定了特定含义

•内部转换

内部转换分栏中的事件的发生不会导致状态的改变，也就是说，即使发生了事件且执行了相应的动作，对象仍然处于原来的状态，故把这样的事件触发的转换称为内部转换。

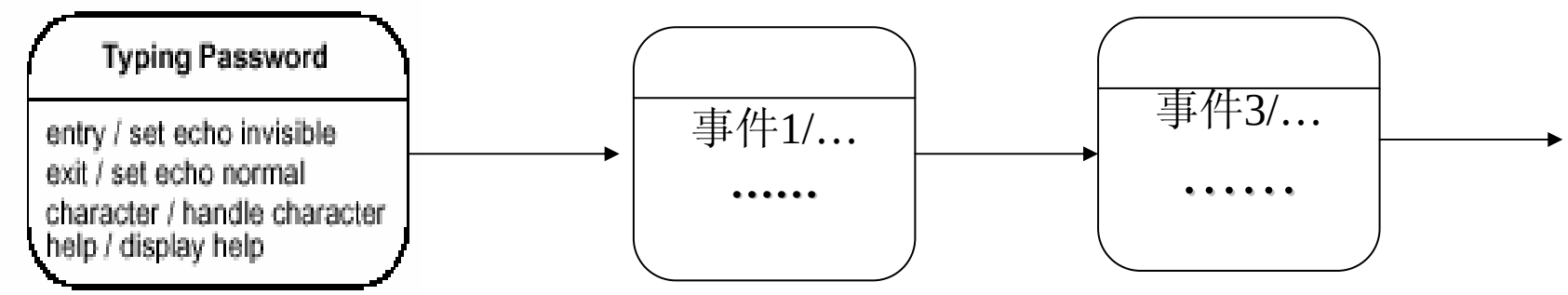
•伪状态

初始状态是状态机图的缺省开始状态，终止状态是状态机图执行已经毕后的结束状态。

•延迟事件

延迟事件是指在当前状态下暂不处理，但将推迟到该对象的另一个状态下排队处理的事件。也就是说，在某些建模情况下，可能想要识别某些事件，但延迟对它们的响应，直到以后才执行。

用特殊的动作defer表明一个事件被延迟：　　事件/defer



事件1，事件2，事件3/defer　　事件2，事件3/defer　　事件2，事件4/defer

•**动作表达式**是由一些动作组成的动作序列，见下节。

3、动作

动作是在状态内或在状态转化时所执行的操作，是原子的和即时的。

动作可为：

- ■设置或修改本对象的属性操作；
- ■向一个对象发送信号；
- ■调用另一个对象的一个可见性为公共操作；
- ■创建或撤消对象；
- ■返回一个值或值集；
- ……

动作是原子的，是指在与状态相关的抽象层次上，动作是不可间断的；动作时即时的，是指动作执行的时间是可忽略不计的。

在转换中、在状态的入口、在一个对象处于一个状态的整个期间或在状态的出口，都是执行动作的时机。

UML的三个保留字：

1） entry/进入动作表达式

entry这个标号标识由相应的动作表达式规定的动作，在进入状态时首先执行该动作。它不能有参数或监护条件。

2）exit/退出动作表达式

exit这个标号标识由相应的动作表达式规定的动作，在退出状态时最后执行该动作。它不能有参数或监护条件。

3）do/活动

活动是在对象处于一个状态中的整个阶段执行的一个动作或动作的集合。　　如：op1(a)；op2(b)；op3(c)

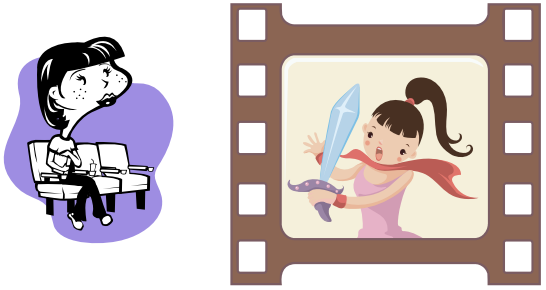
活动不是原子的，在执行中可以被事件打断。

动作是从不中断的，但动作序列是会中断的。也即，在每两个动作之间（由分号分开），由于事件的出现，导致一个离开此状态的转换。

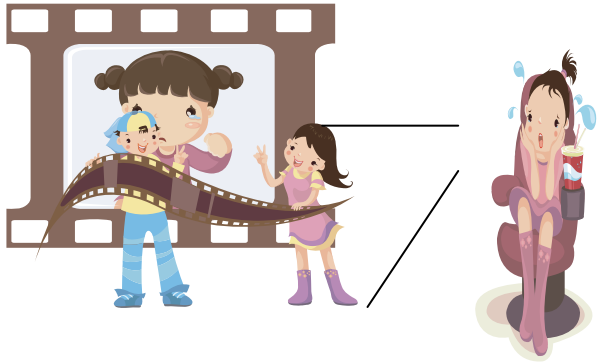
do这个标号标识正在进行的活动（“do 活动”）。活动[illegible]表达式，它在执行中可以被事件中断。

do活动在状态的入口动作执行后开始执行，并且它与其他的动作或活动是并发的。

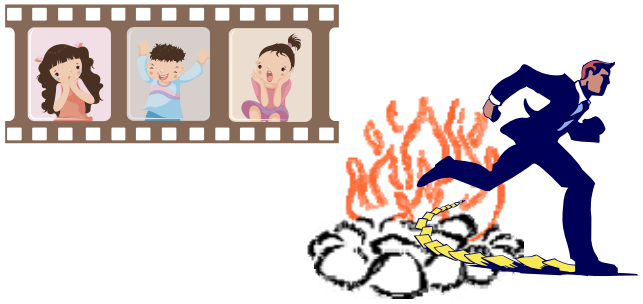
——只要被建模的对象是在当前的状态中，就执行这个活动，直到对象离开该状态为止。

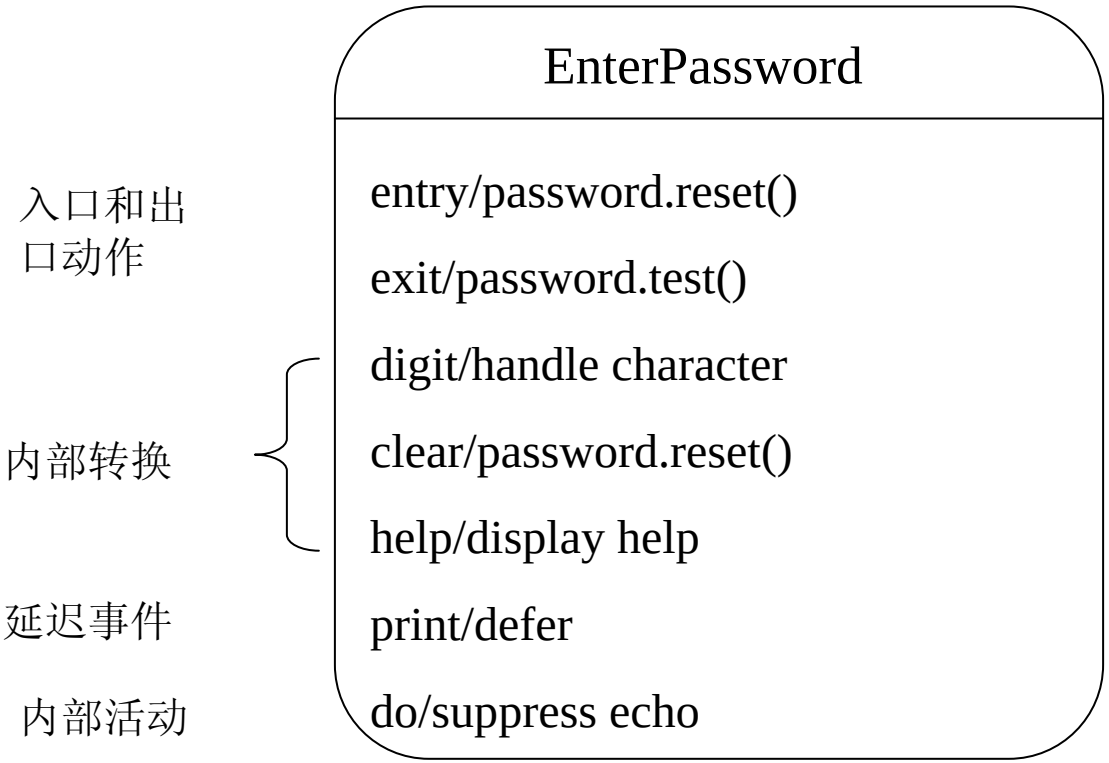


——若do活动执行完毕后对象仍处于当前状态，这时会导致一个完成事件，如果存在一条外出的完成转移，若满足监护条件就退出当前状态。



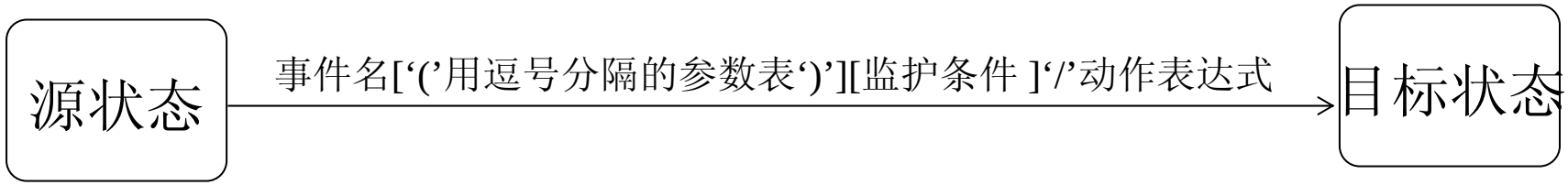
——如果在活动没未完成之前，由于外出转换的激发(满足监护条件)而导致了状态的退出，就中断活动。





4、状态转换

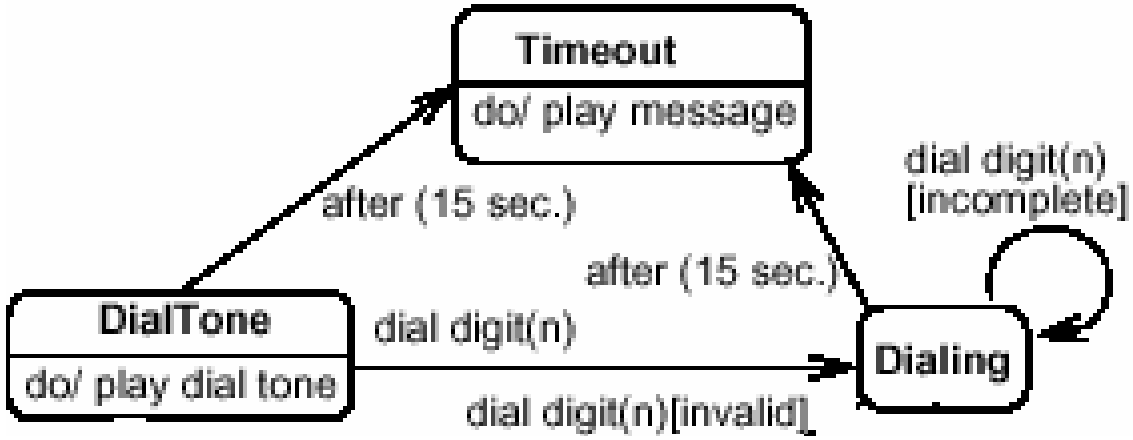
状态转换是两个状态之间的一种关系，表示当一个特定事件出现时，如果满足一定的条件，对象就从第一个状态（源状态）进入第二个状态（目标状态），并执行一定的动作。转换本身也是原子的。



事件可能有参数，这样的参数可由转换中的监护条件和动作使用，也可由与源状态和目标状态相关的退出和进入动作分别使用。

触发到自身的转换，要先退出当前状态，再进入该状态，这样要执行退出动作和进入动作。

触发内部转换，不需要退出当前状态。

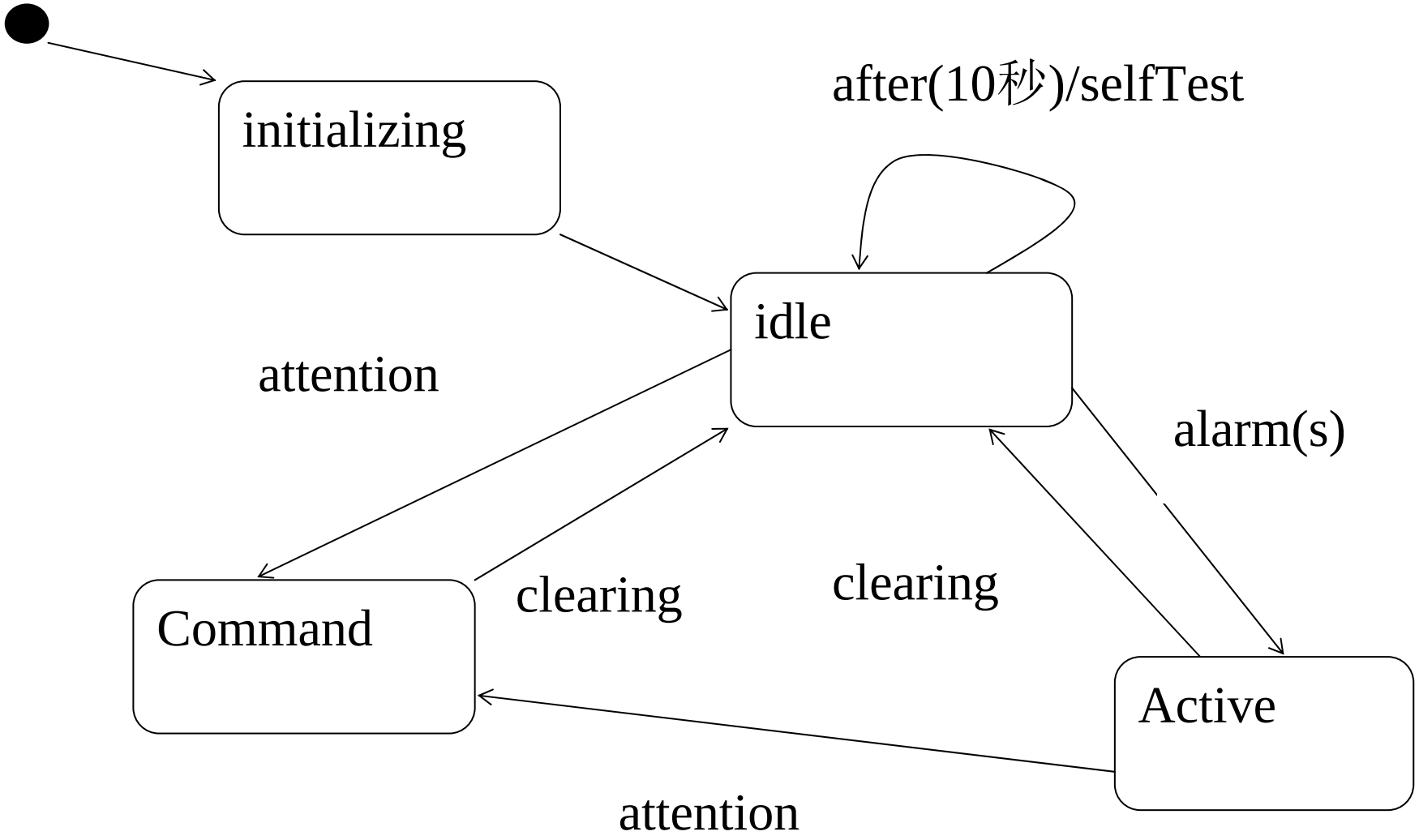


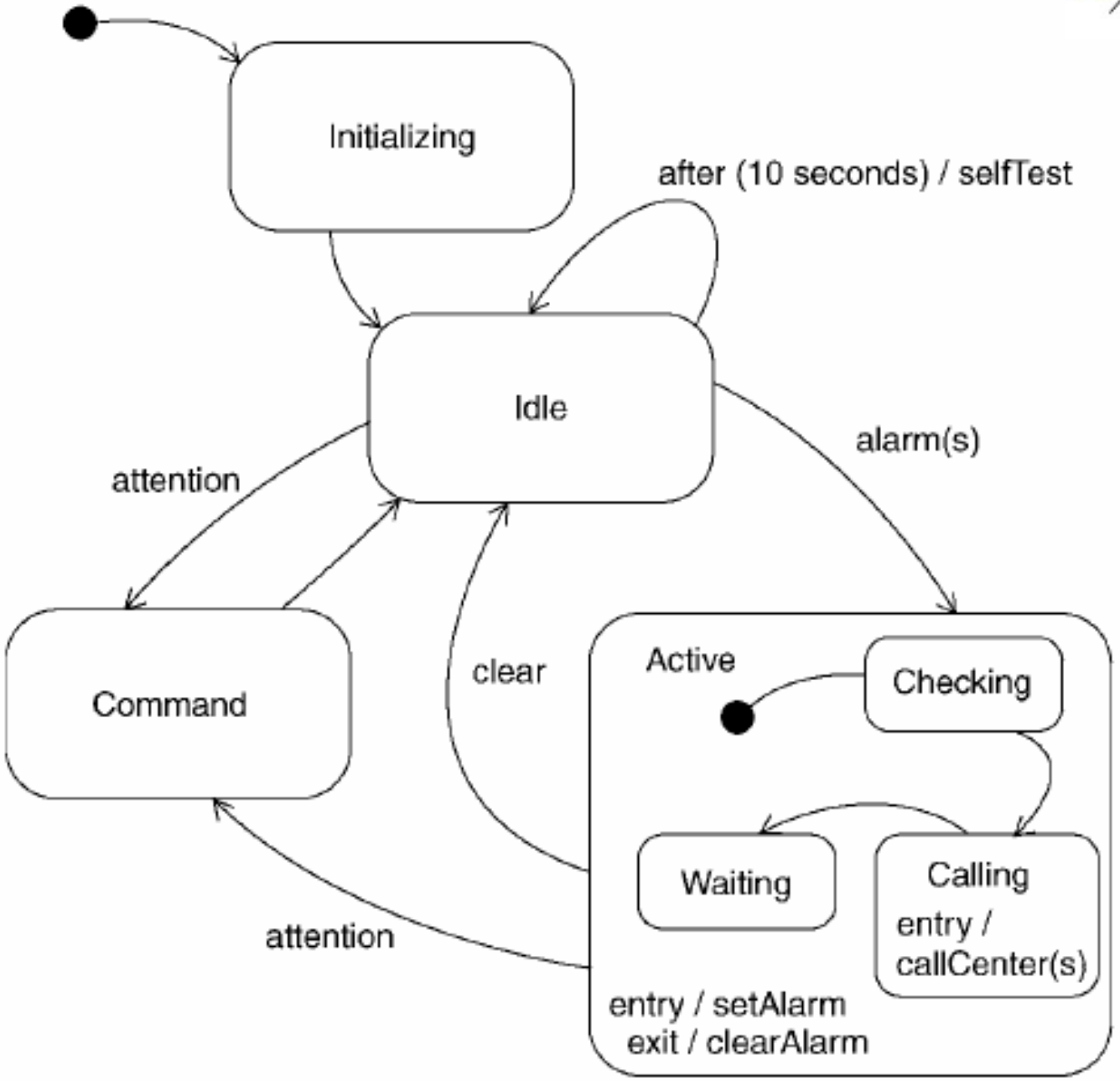
5、状态机图

一个**状态机图**描述一个对象在其生命期内响应事件所经历的状态序列，以及对这些事件所做出的反应。

通常用状态机图描述类的行为，也可以用它描述其它模型实体（如用况、参与者、子系统）的行为。

例题1 下图描述了负责监视某些传感器的一个控制器的状态机。



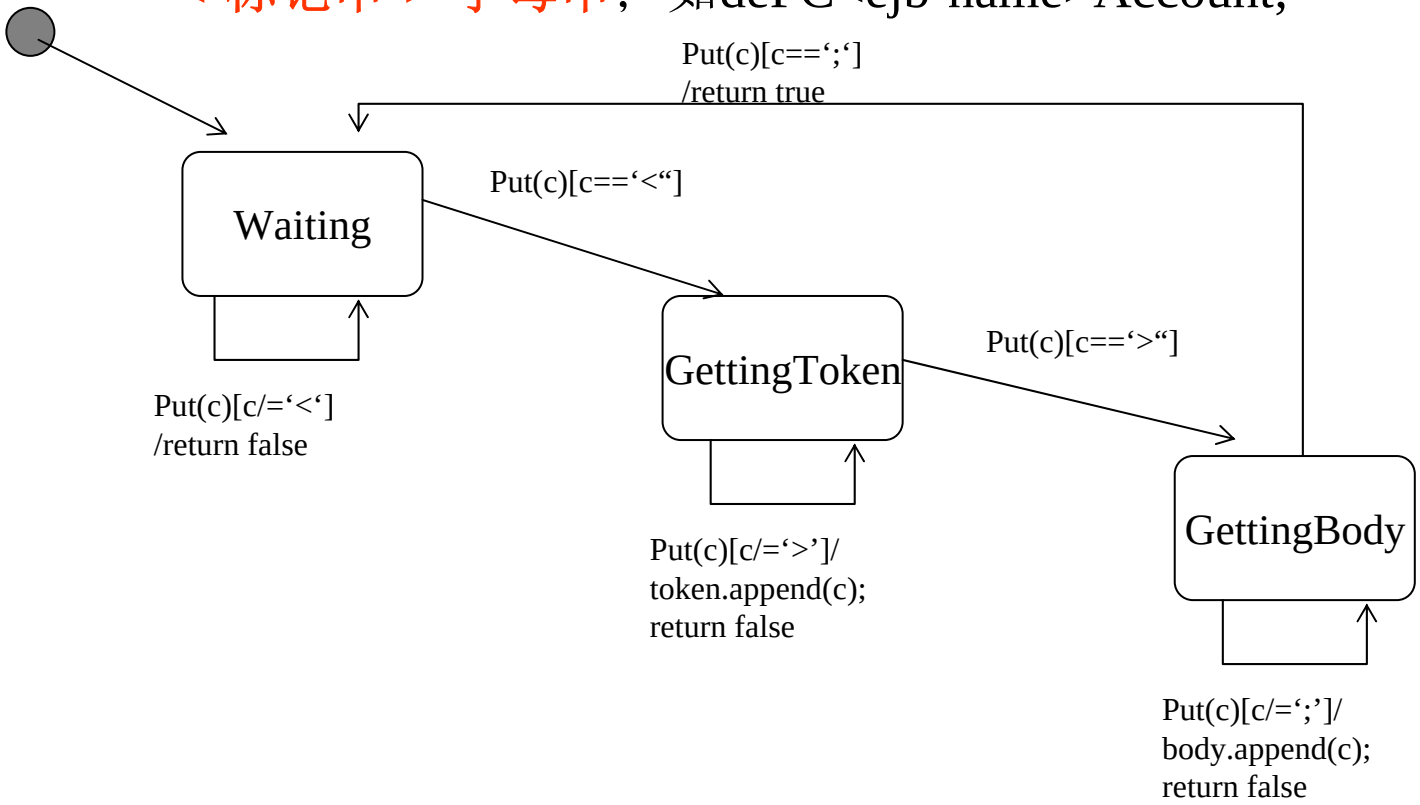


如果事件有参数，可以把参数用在动作表达式中。

例题2

绘制一个状态机图，它能分析如下格式的字符流：

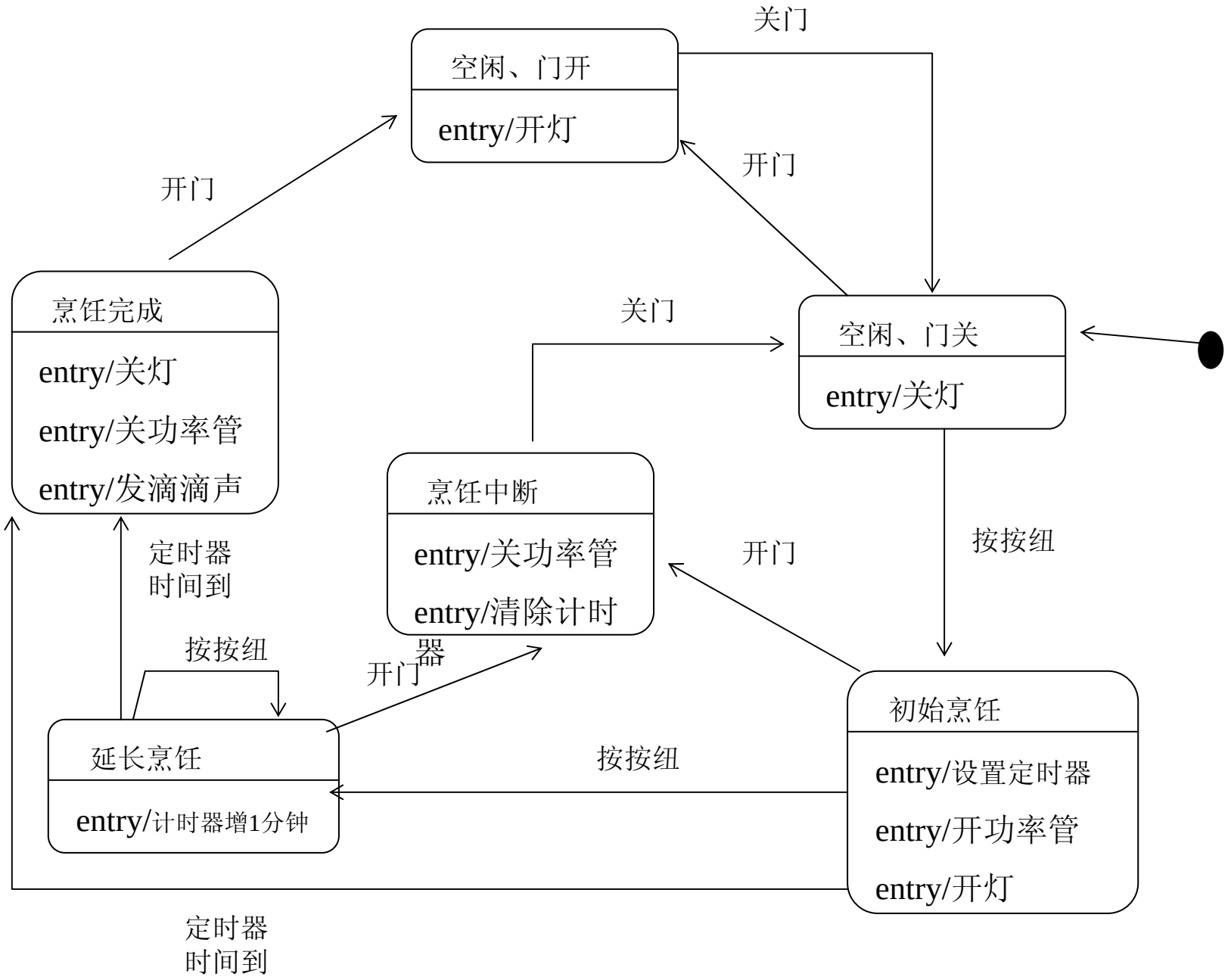
‘<’标记串‘>’字母串； 如deFC<ejb-name>Account;



如果监护条件不同，相同的事件名可以出现多次，当该事件发生时，根据监护条件决定触发那个转换。

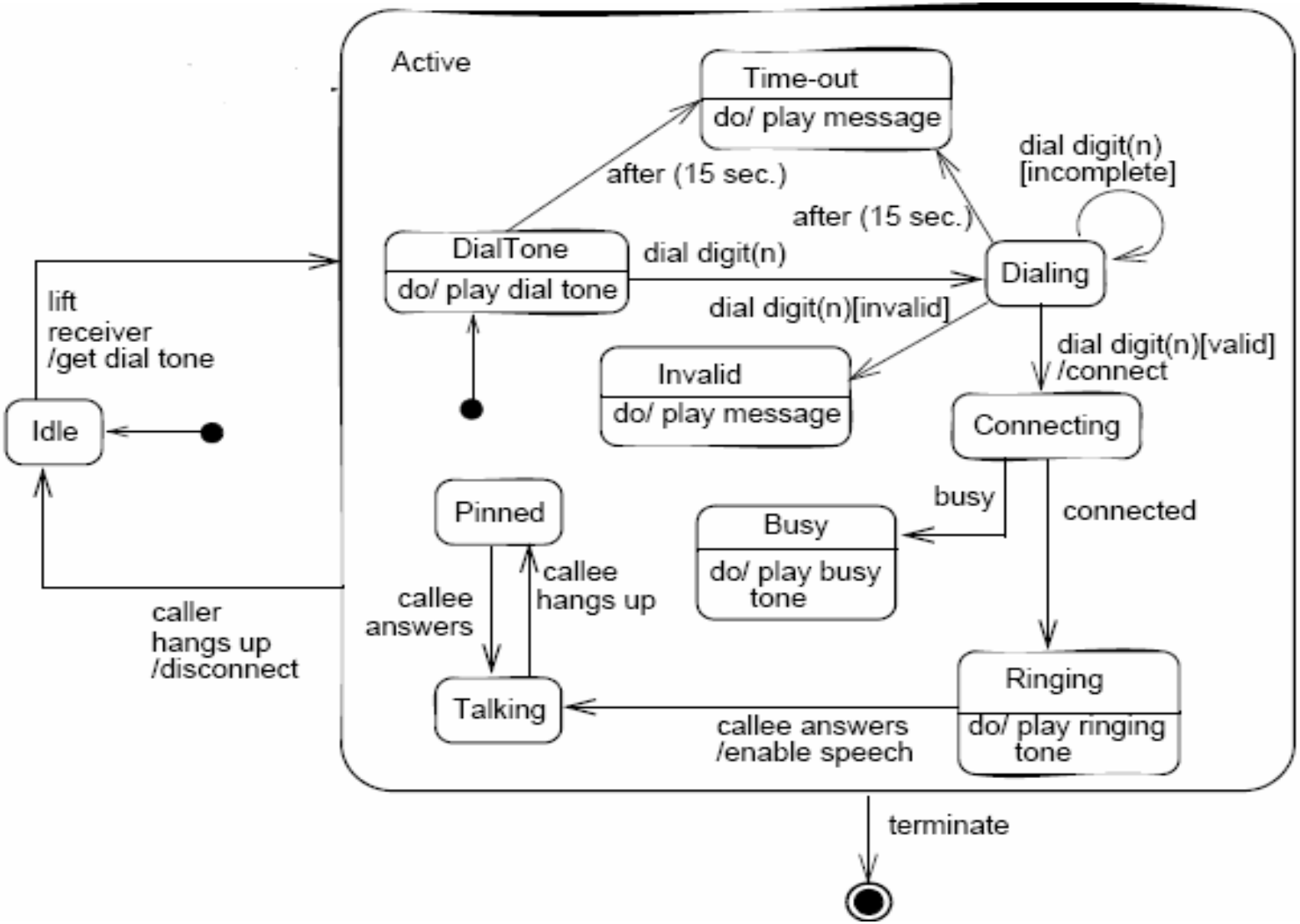
根据触发事件的参数和拥有这个状态机的对象的属性和链来写转换。

例题3 为简易微波炉（只有一个按钮）建模



6、组合状态

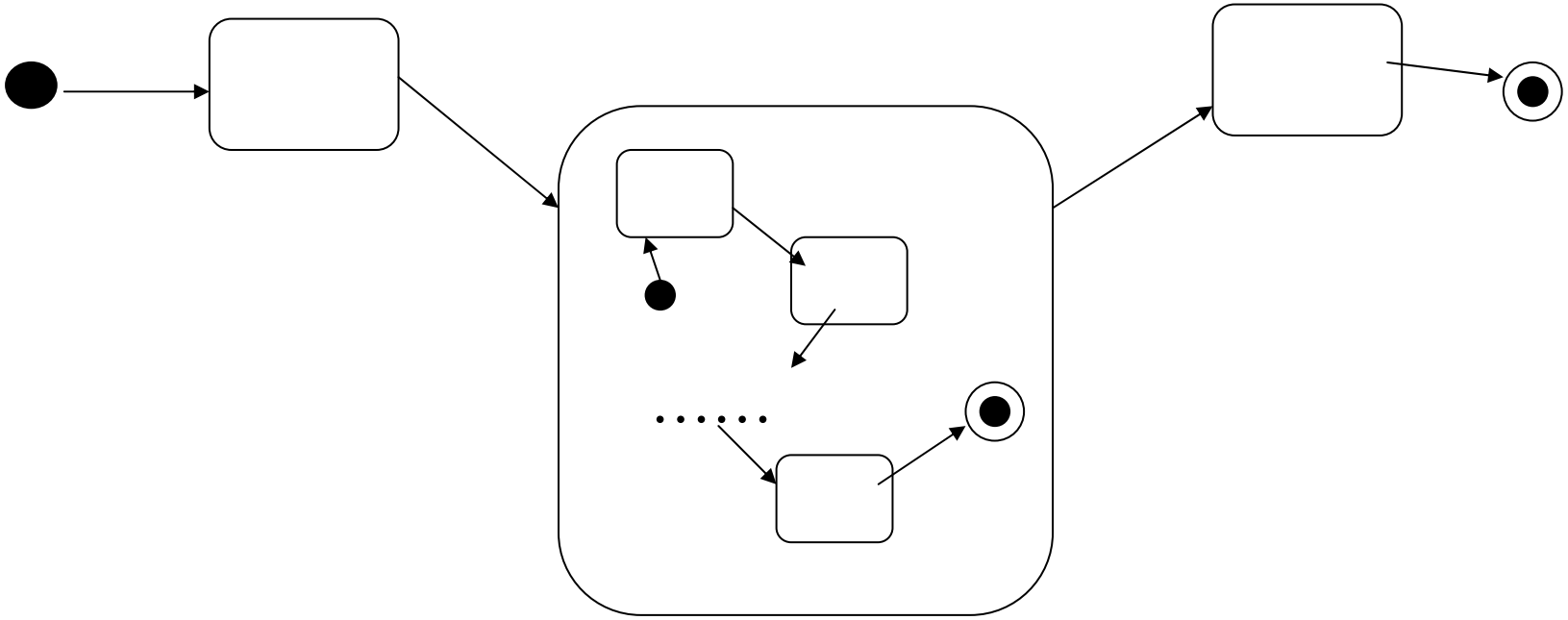
上面讲述的状态机图中的状态都是简单状态。例如，下图中的状态Active是一个组合状态。其中DialTone和Timeout等状态均为Active的子状态。



（1）语义

组合状态是由两个或多个子状态构成的状态，其中的子状态是顺序的或并发的，而且子状态还可以是组合状态。

新创建的对象，从最外层的初始伪状态开始，执行其最外层的缺省转换。若对象转换到了最外层的终结状态，则对象的生命期终止。

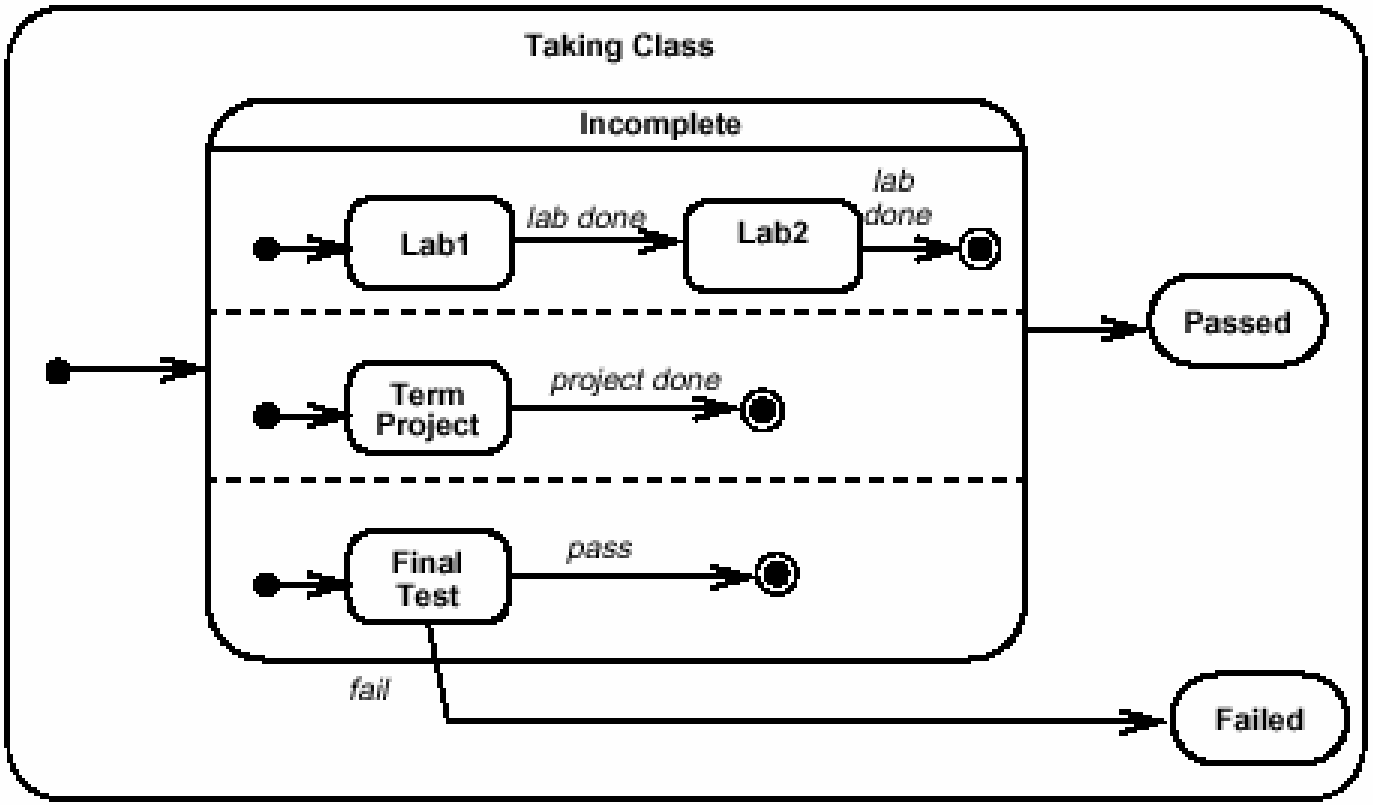


一个状态内的各区域可以有初始伪状态和终止状态。

到封闭状态的转换表示到其初始伪状态的转换。

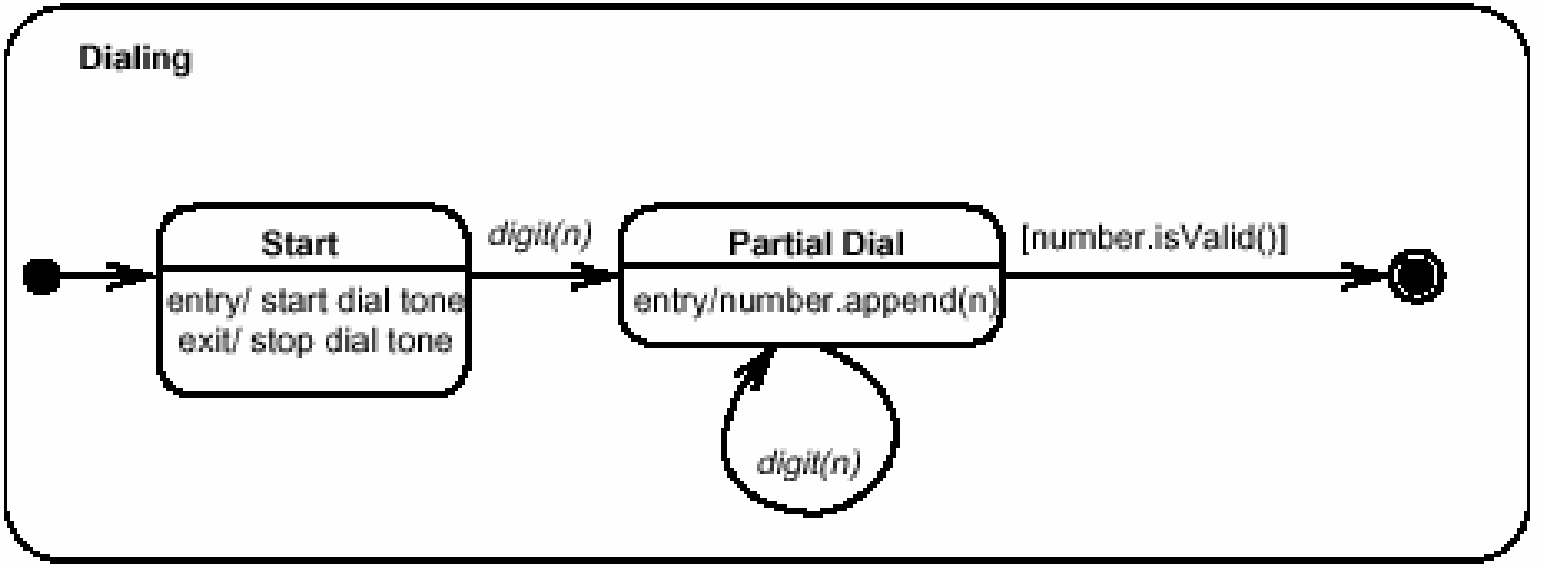
到最终状态的转换表示封闭区域中的活动的完成。

在所有并发区域中的活动的完成，表示经由封闭状态的活动的完成，并触发封闭状态上的完成事件。



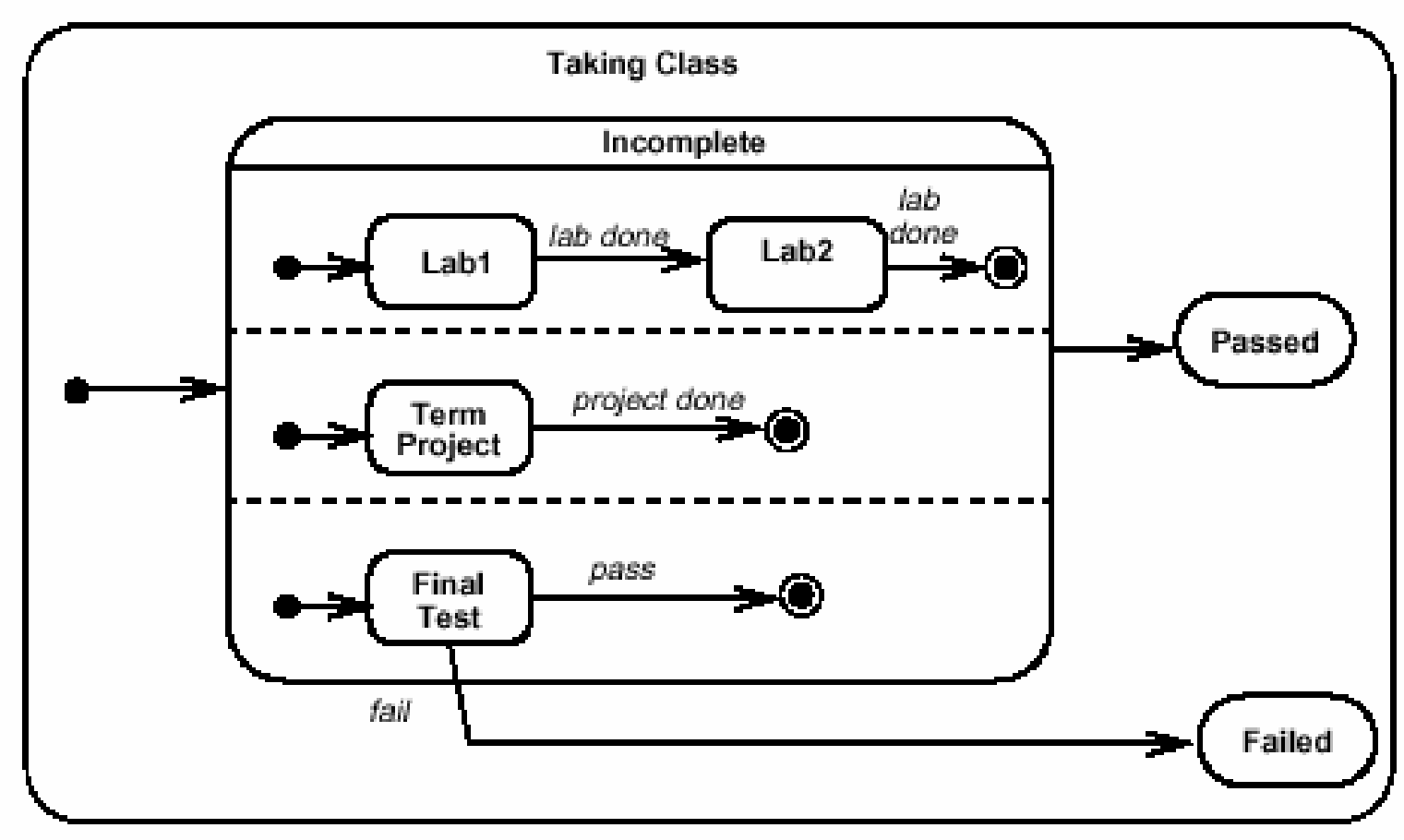
（2）表示法

把组合状态展开是为了表示它的内部状态机结构。除了（可选的）名称和内部转换分栏外，状态可以包含容纳嵌套图的附加分栏。通过在图形区域里显示嵌套状态机图，把状态展开，表示其不相交（顺序）的子状态。



顺序子状态示意图

用虚线划分图形区域，每个区域都是一个并发的子状态。每个区[illegible]名称，但必须包含一张具有不相交状态的状态机图。用实线把整个状态的名称分栏和内部转换分栏与并发的子状态相分离。



并发子状态示意图

从带有并发组合状态内的任一状态离开的转移导致离开所有的其他并发区域。这种转移通常表示有错误发生或例外发生，从而迫使所有并发计算都被中断。

5．3．2 建立状态机图

对对象的状态变迁建模，应遵循如下策略：

■ 设置状态机的语境

即要考虑在特定的语境中哪些对象与该对象交互，包括这个对象的类的所有父类和通过依赖或关联到达的所有类。这些邻居是动作的侯选目标或在监护条件中包含的侯选项。

■ 建立初始状态和终止状态。

■ 选定对象中的一组有意义的对对象状态有影响的属性, 结合有关的事件和动作, 对象可能在其中存在各段时间的条件，以决定该对象所在的稳定状态。

■ 在对象的整个生命期中，决定稳定状态的有意义的偏序。从初态开始到终态，列出这个对象可能处于的顶层状态

■ 决定这个对象可能响应的事件。可在对象的接口处发现这些事件，并给出一个唯一的名字。这些事件可能触发从一个合法状态到另一个合法状态的转换。

■ 用被适当的事件触发的转换将这些状态连接起来，接着向这些转换中添加事件、监护条件或动作。对于内部转换也是如此。

■ 识别各状态的进入或退出的动作

■ 如果需要，从这个对象的高层状态开始，然后考虑各自的可能子状态，用子状态进行扩充。

检查

- 检查该对象的接口所期望的所有事件，是否都被状态机所处理。
- 检查在状态机中提到的所有动作是否被闭合对象的关系和操作所支持。
- 通过状态机，跟踪检查事件的顺序和它们的响应，尤其要努力地寻找那些未达到的状态和导致状态机不能走通的状态。
- 在重新安排状态机后，按所期望的顺序再一次检查，以确保你没有改变该对象的语义。

作业：

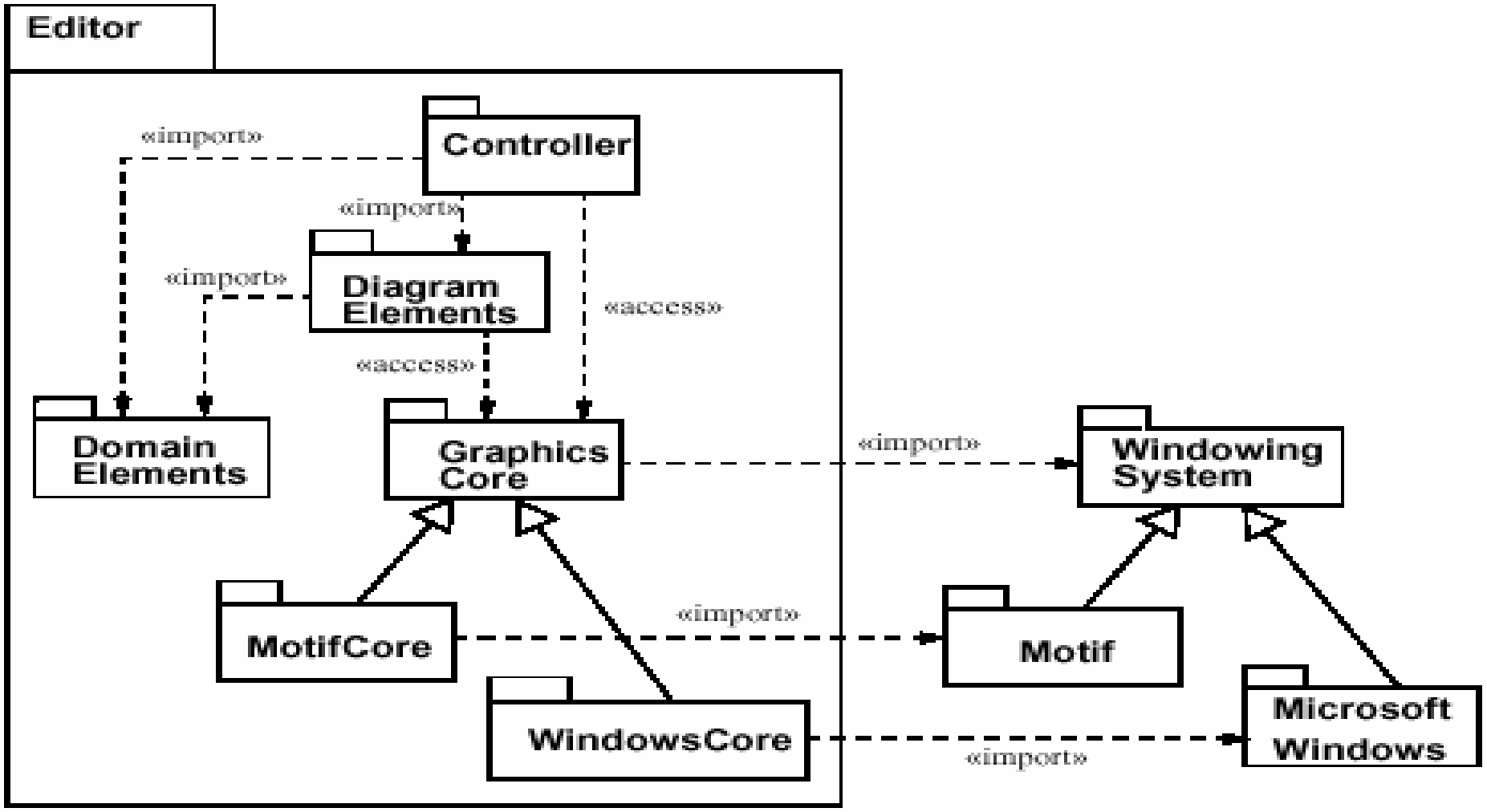
1. 为简易的电子表建立状态机图。

2.在图书馆中，购入的书在半个月内为新书，以后为旧书。书无论新旧，都可以向外借阅。针对上述要求建立状态机图。

3、针对简易电梯，建立状态机图。

5.5 包图

对一个较为复杂的系统建模，要使用大量的模型元素，这时就必要把这些元素分组进行组织。这样把在语义上接近且倾向于一起变化的模型组织在一起，不但控制模型的复杂度，有助于理解，而且也有助于按组控制元素的可见性。



7.3.1 概念与表示法

包是对模型元素分组的机制。

使用包的最常见目的是把建模元素组织成为组，作为一个集合进行命名和处理。

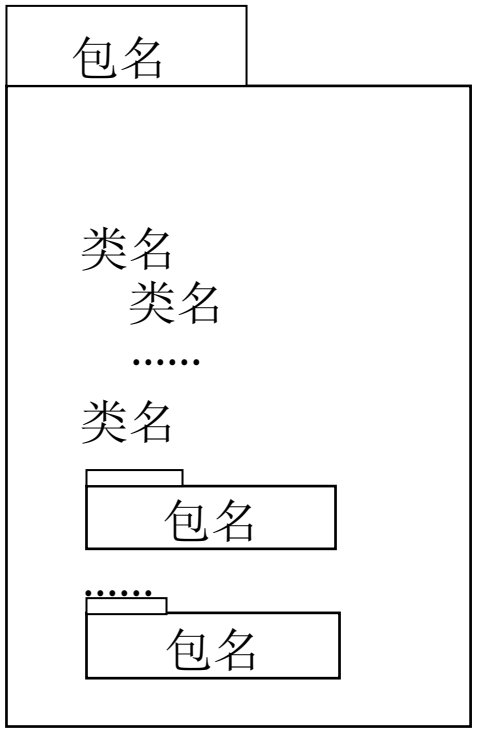
包可以拥有类、接口、构件、节点、用况和图，甚至可以是其它包。拥有是一种组成关系，这意味着被拥有的元素被声明在包中。如果包被撤消了，元素也要被撤消。——包是一个命名空间，一个元素只能被一个包所拥有。

设计良好的包，把在语义上接近并倾向于一起变化的元素组织在一起。因此结构良好的包是松耦合、高内聚的，而且对其内容的访问具有严密的控制。

包的层次性

因为包中还可以有包，这样包之间可以有一个层次，且在组织结构上是一棵严格的树。

在实际使用中，最好要避免过深地嵌套包，一般两、三层即可。对过多的嵌套，要用“引入依赖”来组织包。



对包中元素的命名

一个包形成了一个命名空间，这意味着在一个包的语境中同一种元素的名字必须是唯一的。

例如，同一个包不能拥有两个名为**Queue**的类，但在**P1**包中可以有一个名为**Queue**的类，而在**P2**包中又有另一个（不同的）名为**Queue**的类。实际上，类**P1::Queue**和类**P2::Queue**是不同的类，这可以由各自的路径名区别开来。

如果一个包位于另一个包中，外层的包可作为里层包的前缀。例如，在包Vision中有一个名为Camera的类，而包Vision又在包Sensor中。类Camera的全名为Sensor::Vision::camera。

在一个包中不同种类的元素可以有相同的名字。这样，在同一个包中，对一个类命名为Timer，对一个构件也可以命名为Timer。为了不造成混乱，最好对一个包中所有元素也都唯一地命名。

如果包的内容没有被显示在大矩形中，那么可以把该包的名字放在大矩形中。如果包的内容被显示在大矩形中，那么可以把该包的名字放在左上角的小矩形中。

包中元素的可见性

一个包中的元素在包外的可见性，通过在元素名字前加上一个可见性符号来指示。模型元素的可见性可为+(公共的)、-(私有的)、 #(受保护的) 或~（包范围的），它们的含义为：

＋：标有“+”号的模型元素对所有的引入包以及它们的后代是可见的。包的各公共部分一同构成包的接口。

－：标有“—”号的模型元素只对包内的元素是可见的。

＃：标有“#”号的模型元素只对那些与包含这些元素的包有泛化关系的子包是可见的。

~：标有“#”号的模型元素只对在同一包内声明的其他元素是可见的。

包间的关系

包之间不但可以具有拥有关系，包之间也可以具有引入关系、访问关系或泛化关系。

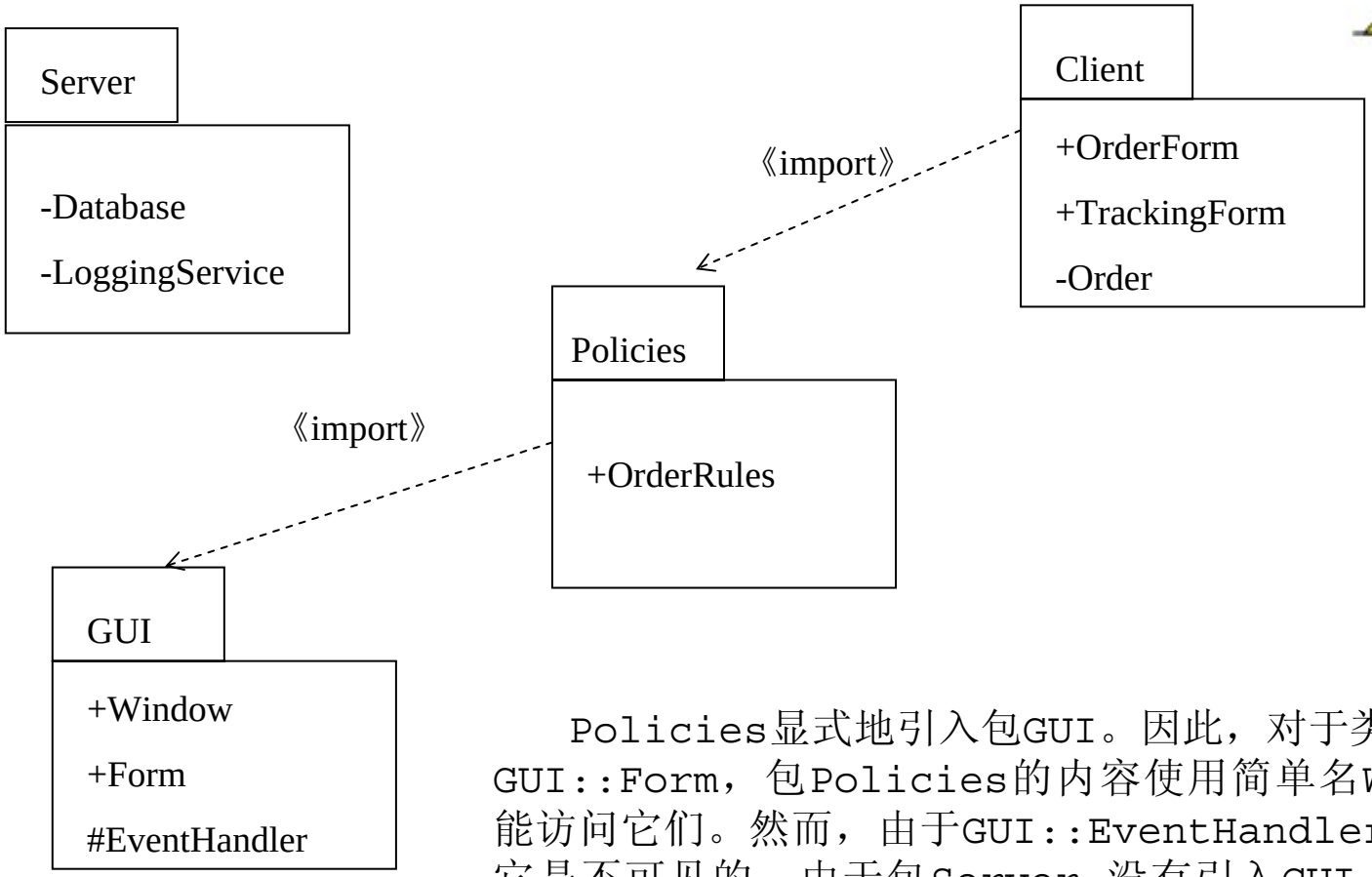
包间的引入依赖是两个包之间的一种许可依赖关系，一个包中的可见性为公有的模型元素，可以在指定的包（包括嵌套在该包中的子包）中被引用，相当于把提供者包的内容附加到客户包的公共命名空间中，而不必对名称进行限制。

把引入依赖绘制成带有箭头的虚线，其上标有串**<<import>>**。

包间的访问依赖是两个包之间的一种许可依赖关系，一个包中的可见性为公有的模型元素，可以在指定的包（包括嵌套在该包中的子包）中被引用，相当于把提供者包的内容附加到客户包的私有命名空间中，而不必对名称进行限制。

把访问依赖绘制成带有箭头的虚线，其上标有串**<<access>>**。

包间的泛化关系与类间的泛化很类似，即特殊包可以继承一般包中的可见性为公共的或受保护的元素，而且在特殊包中还可以有自己的元素，自己的元素可以覆盖继承来的元素。



Policies显式地引入包GUI。因此，对于类GUI::Window和类GUI::Form，包Policies的内容使用简单名Window 和Form 就能访问它们。然而，由于GUI::EventHandler 是受保护的，因此它是不可见的。由于包Server 没有引入GUI，Server 中的内容必须用限定名才能访问GUI的公共内容，例如，GUI::Window。类似地，由于Server中的内容是私有的，GUI的内容无权访问Server中的任何内容，即使用限定名也不能访问它们。

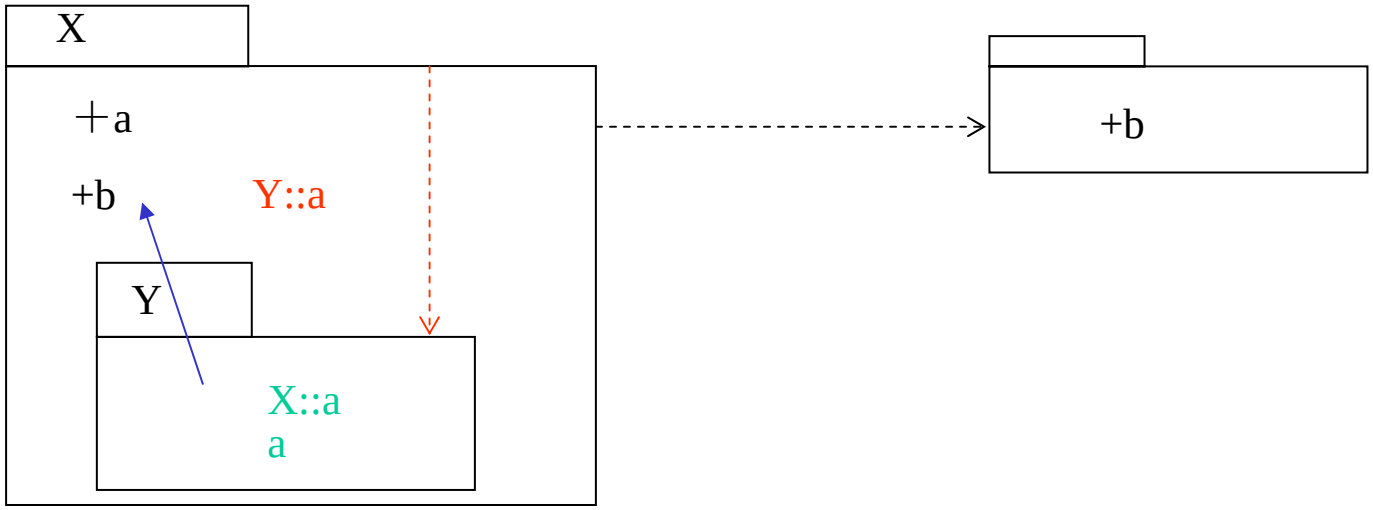
引入和访问依赖是传递的。在本例中，Client引入Policies，Policies引入GUI，所以Client就传递地引入了GUI。因此，Client的内容可以访问Policies的引出，同样可以访问GUI的引出。如果Policies访问GUI，而不是引入它，Client则不能把GUI中的元素添加到自己的命名空间，但是仍然能通过限定名（如GUI::Window）引用它们。

嵌套的包之间也存在着可见性问题：

1）里层的包中的元素能够访问其外层包中定义的可见性为公共的元素，也能访问其外层包通过访问或引入依赖而得来的元素。

2）一个包要访问它的内部包的元素，就有与内部包有引入、访问关系或使用限定名。

3）里层包中的元素的名字会掩盖外层包中的同名元素的名字，在这种情况需要用限定名引用外层包中的同名元素。



7.3.2 如何划分与组织包

识别低层包

每个具有泛化关系或聚合关系的元素位于一个包

关联密集的类划分到一个包

独立的类暂时作为一个包

合并或组织包

如果低层包数量过多，则把它们合并，或用高层包组织它们。

若低层包之间在概念上接近或具有较强的相关性，从作用上属于某项大的功能，在图上有较强的耦合性，或在分布上处于同一台处理机，则考虑把它们合并，或用高层包组织它们。

建议每个包有7±2个内层成分。——在一张A4纸上表达不清楚时就划分包（Martin Fowler）

组织包的层次

层次不宜太多

包的划分不是唯一的，有一定的随意性

标识包中的模型元素的可见性

对每一个包，确定哪些元素在包外是可以访问的，把它们标记为公共的。把所有其它的元素标记为受保护的或私有的。

建立包间的关系

根据需要，在包之间建立引入依赖、访问依赖或泛化关系。