返回总目录

目　　录

第 5 章 动 态 建 模

所有系统均可表示为两个方面：静态结构和动态行为。UML 提供图来描述系统的结构和行为。类图(class diagram)最适合于描述系统的静态结构：类、对象以及它们之间的关系。而状态、序列、协作和活动图则适合于描述系统的动态行为，即描述系统中的对象在执行期间不同的时间点是如何动态交互的。

类图将现实生活中的各种对象以及它们之间的关系抽象成模型。描述系统的静态结构能够说明系统包含些什么以及它们之间的关系，但它并不解释系统中的各个对象是如何协作来实现系统的功能。

系统中的对象需要相互通信：它们相互发送消息。例如，客户对象张三发送一个消息“买”给售货员对象李四。通常情况下，一个消息就是一个对象激活另一个对象中的操作调用。对象是如何进行通信以及通信的结果如何则是系统的动态行为，也就是说，对象通过通信来协作的方式以及系统中的对象在系统的生命期中改变状态的方式是系统的动态行为。一组对象为了实现一些功能而进行通信称之为交互，可以通过三类图来描述交互：序列图、协作图和活动图。

本章中描述的动态图有：

- 状态图：状态图描述对象在生命周期内处于哪些状态，每一种状态的行为以及什么样的事件引起对象状态发生改变；例如，一张发票可以是已付(状态 paid)和未付(状态 unpaid)。
- 序列图：描述对象如何相互交互和通信。序列图中的最要的是时间。通过序列图，可以看出为了完成某种功能一组对象如何发送和接收一序列消息。
- 协作图：协作图也是描述对象交互的，但侧重于空间的协作，意即明确地给出对象间的关系(链接)。
- 活动图：也是描述对象交互的，但侧重于工作的描述。当对象相互交互时，需要执行一些工作或活动。这些活动以及它们的出现顺序就是活动图所要描述的。

因为序列图、协作图和活动图都是用来描述对象交互的，所以在具体描述一个交互时就需要为此作出选择，具体如何选择起决于你需要着重描述交互的哪个方面：时间？空间？还是活动？

除了静态结构和动态行为外，还可以从功能的角度来描述系统。功能的角度主要是描述系统提供的功能。用例就是从功能的角度来描述系统：它描述角色如何使用系统。正如我们所讨论的，通常情况下在系统描述的早期阶段(如，需求分析阶段)通过用例来描述角色是如何使用系统的。用例模型描述的是角色如何使用系统而不是如何建立系统、类和交互实现系统中的用例。交互通过序列图、协作图或活动图来描述，因而在系统的功能视图和动态视图之间也存在着联系。在用例的实现中所使用的类在类图和状态图中描述。用例和它们的关系在第 3 章中已经作了描述。

5.1 对象之间的交互——消息

在面向对象的编程中，两个对象之间的交互表现为一个对象发送一个消息给另一个对象。在这里，有一点很重要，不能仅仅从字面上理解“消息”这个词，因为消息是在通信协议中发送的。通常情况下，当一个对象调用另一个对象中的操作时，消息是通过一个简单的操作调用来实现；当操作执行完成时，控制和执行结果返回给调用者。消息也可能是通过一些通信机制在网络上或一台计算机内部发送的真正的报文。在所有动态图(序列图、协作图、状态图、活动图)中，消息是作为对象间的一种通信方式来表示的。具体来说，消息是连接发送者和接收者的一根箭头线。箭头的类型表示消息的类型。

图 5-1 显示了 UML 中的消息类型。它们是：

- 简单消息：表示普通的控制流。它只是表示控制是如何从一个对象传给另一个对象，而没有描述通信的任何细节。这种消息类型主要用于通信细节未知或不需要考虑通信细节的场合。它也可以用于表示一个同步消息的返回；也就是说，箭头从处理消息的对象指向调用者表示控制返回给调用者。
- 同步消息：一个嵌套控制流，典型情况下表示一个操作调用。处理消息的操作在调用者恢复执行之前完成(包括任何在本次处理中发送的其它消息)。返回可以用一个简单消息来表示，或当消息被处理完毕隐含地表示。
- 异步消息：异步控制流中，没有直接的返回给调用者，发送者发送完消息后不需要等待消息处理完成而是继续执行。在实时系统中，当对象并行执行时，常采用这类消息。

可以将一个简单消息和一个同步消息合并成一个消息，原同步消息的箭头和简单消息的箭头分别放在合并后的消息的两端。这样的消息意味着操作调用一旦完成就立即返回。

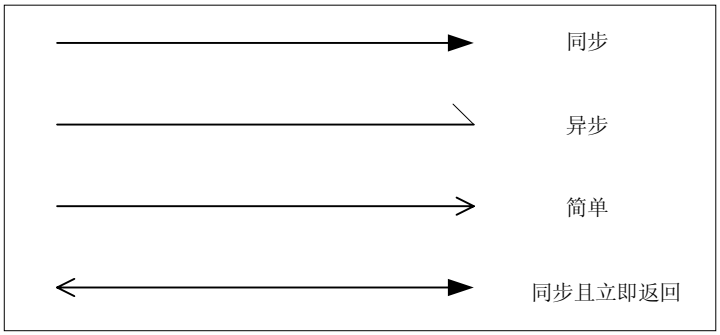

图 5-1 消息类型

5.2 状 态 图

状态图主要用来描述对象、子系统、系统的生命周期。通过状态图可以了解到一个对象所能到达的所有状态以及对象收到的事件(收到消息，超时，错误，条件满足)对对象状态的影响等。所有的类，只要它有可标记的状态和复杂的行为，都应该有一个状态图。

状态图指定对象的行为以及根据不同的当前状态行为之间的差别。同时，它还能说明事件是如何改变一个类的对象的状态。

5.2.1 状态和转移

所有对象均有状态：状态是对象操作的前一次活动的结果，通常情况下，状态由对象的属性值以及指向其它对象的链来决定的。类的状态由类中的指定属性来说明或对象的状态由对象中的通用属性的值来确定。下面举例说明对象的状态：

- 支票(对象)已付(状态)。
- 汽车(对象)停在那儿(状态)。
- 发动机(对象)正在运行(状态)。
- 吉姆(对象)正在卖货(状态)。
- 小王(对象)已婚(状态)。

当某些事情发生时对象的状态发生改变，我们称改变对象状态的事情为“事件”，例如，付了支票，开始启动汽车，或结婚。动态性表现在两个方面：交互和内部状态改变。交互描述对象的外部行为以及对象如何与其它对象交换信息(通过发送消息或链接和断链到其它对象)。内部状态改变描述对象是如何改变其状态的，例如，对象的内部属性值。状态图用来显示对象对事件的反应以及对象状态的改变。例如，当某人付了一张支票，则支票对象的状态从未付转移到已付。当支票对象被创建时，它的状态为未付(见图 5-2)。

状态图可以有一个起点和多个终点。起点(初始态)用一个黑圆点表示。终点(终态)用黑圆点外加一个圆表示(很像一只牛眼睛)。状态图中的状态用一个圆角四边形表示。状态之间为状态转换，用一条带箭头的线表示。引起状态转换的事件可以用状态转换线旁边的标签来表示，如图 5-3 所示。当事件发生时，状态转换开始(有时也称之为转换“点火”或转换被“触发”)。

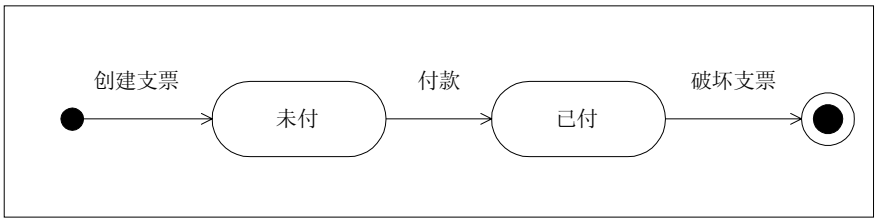

图 5-2 支票对象的状态图。黑圆点代表支票对象的起点(对象刚创建)。黑圆点外加一个圆代表对象的终点(对象被删除)。状态间的箭头表示状态转移和引起状态改变的事件

一个状态一般包含三个部分，如图 5-4 所示。第一部分为状态的名称，如空闲、已付、移动。第二部分为可选的状态变量的变量名和变量值。属性(变量)指的是状态图中类的属性。在某些情况下，临时变量也是很有用的，如计数器。第三部分为可选的活动表，列出有关的事件和活动。在活动表中，常常使用下面三种标准事件：进入，退出，做。“进入”事件用来指定进入一个状态的动作，例如，给属性赋值或发送一条消息。“退出”事件用来指定退出一个状态的动作。“做”事件用来指定在该状态下的动作，例如，发送一条消息，等待或计算。这些标准的事件一般不作它用。活动部分的语法如下：

事件名　参数表　'/'　动作表达式

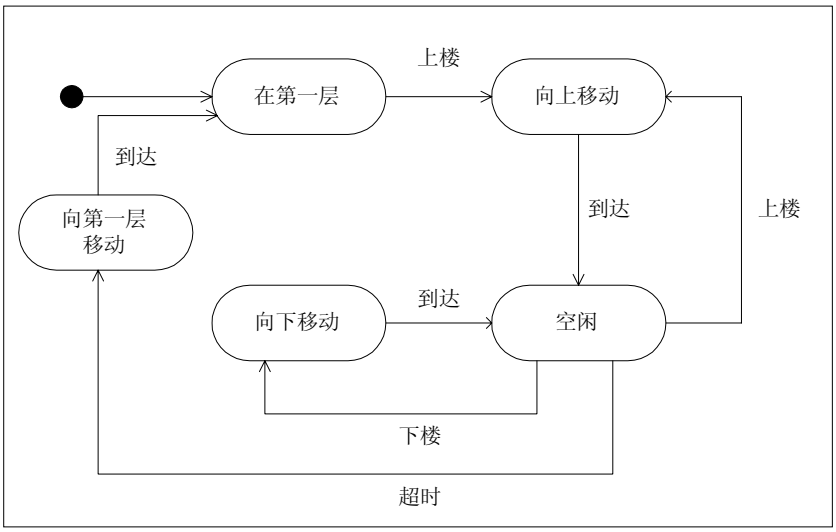

图 5-3　电梯的状态图。电梯从第一层开始启动。它能够上下移动。如果电梯在某一层上处于空闲状态，则当超时事件出现时，它就会返回到第一层。本状态图中没有终点(终态)

“事件名”可以是任何事件，包括上述三种标准事件。“动作表达式”用来指定应该做何种动作(如操作调用，增加属性值等等)。有时还需要为事件指定一些参数(但是上述三种标准事件没有任何参数，参考图 5-5)。

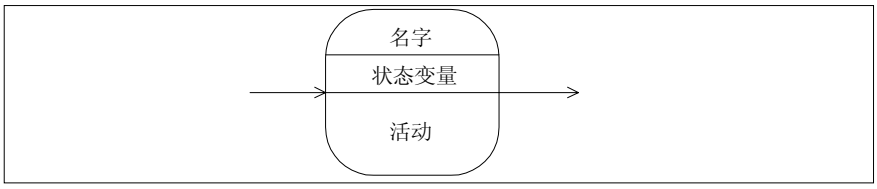

图 5-4　状态的三个组成部分：名称，状态变量，活动

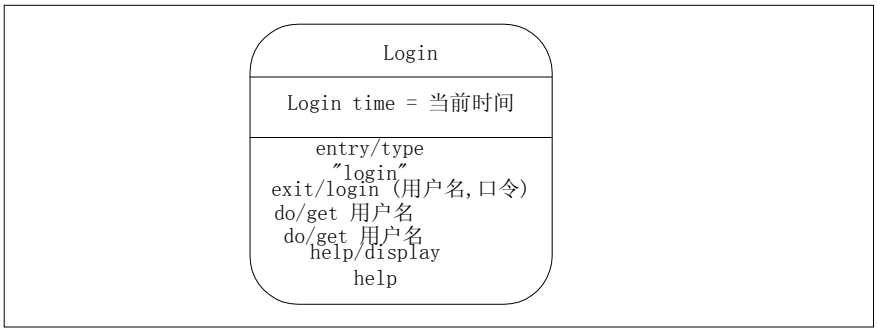

图 5-5 login 状态：状态名为"login"；状态变量为"login time"，变量值为当前时间；在活动表中，列出了该状态下可能发生的各种事件。"help/display help"是一个用户定义的事件

状态转移的语法表示如下：

event-signature '{' guard-condition '}' '/' action-expression

'^' send-clause

其中，"event-signature"的语法表示如下：

事件名 '{' 参数 ',', … '}'

"send-clause"的语法表示如下：

destination-expression '.' destination-event-name '{' argument ',' …'}'

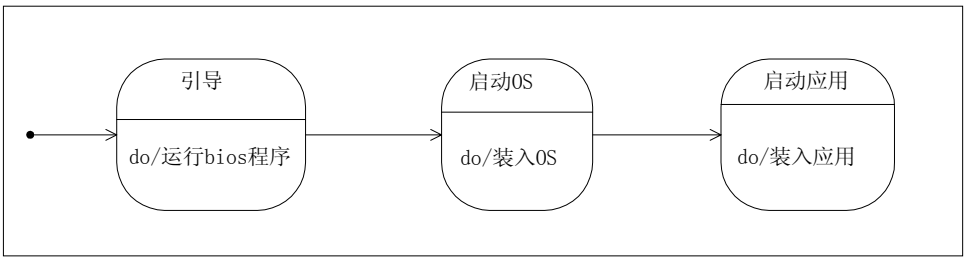

图 5-6 没有直接事件的状态转移。当每一个状态下的活动被实现时状态发生转移

其中，"destination-expression" 是一个由一个或多个对象组成的表达式。有关状态转移语法中的各部分的例子将在后面给出。

5.2.1.1 事件说明(Event-Signature)

从图 5-7 中可以看出，"event-signature"由事件名，参数，触发状态转移的事件，与事件有关的附加数据组成。参数由逗号隔开的参数表来表示，其语法表示如下：

参数名 ':' 类型表达式，参数名 ':' 类型表达式, …

其中，“参数名”是参数的名称，“类型表达式”为参数的类型。例如，类型可以是integer, Boolean, string。类型表达式可以省略，即不一定显示出来。

带有 event-signature 的状态转移的例子如下所示：

```
   draw ( f : Figure, c : Color)
   redraw ()
redraw
   print (invoice)
```

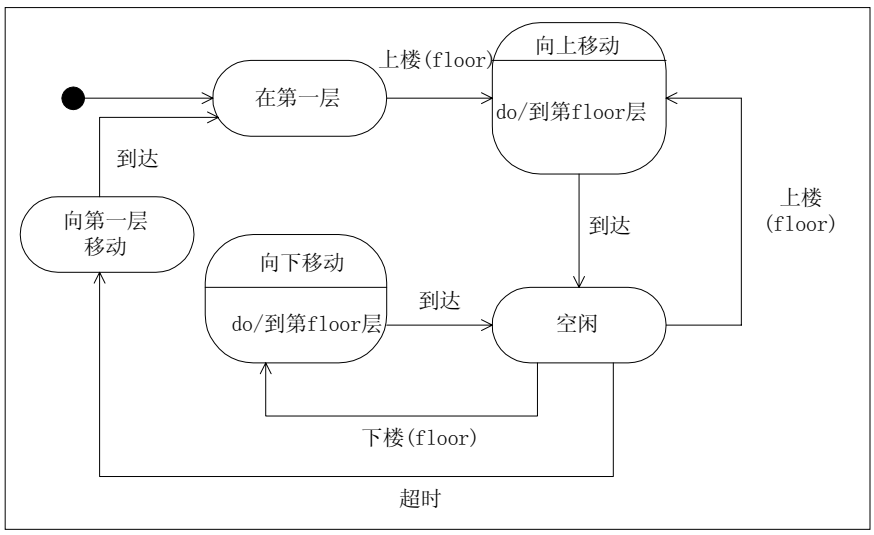

图 5-7 状态转移：状态 “在第一层”和状态“向上移动”之间的状态转移有一个参数"floor"(不考虑类型)，在“空闲”和“向上移动”之间以及“空闲”和“向下移动”之间的状态转移也有"floor"这个参数

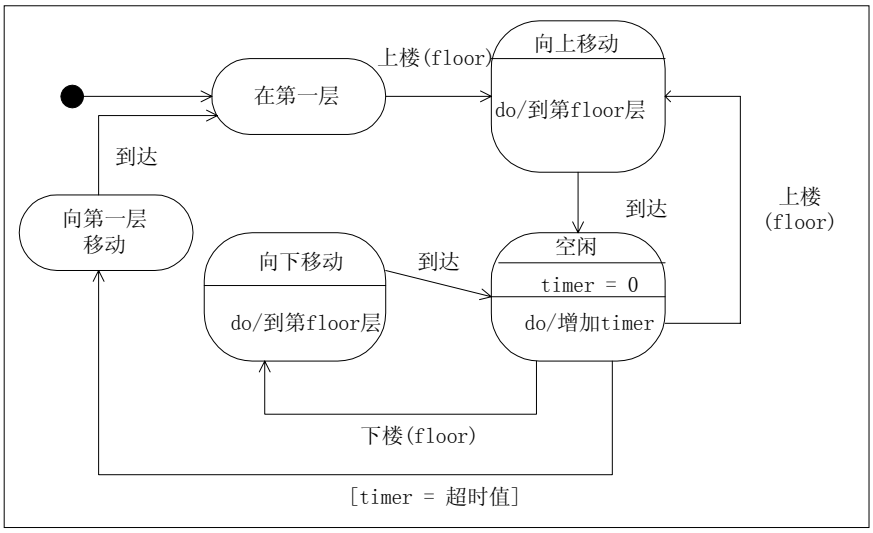

图 5-8 在“空闲”状态，将属性 timer 的值置 0，然后连续递增 timer 的值，直到“下楼”或“上楼”事件发生或守卫条件“timer = 超时值”成真

5.2.1.2 守卫条件(Guard-Condition)

守卫条件是状态转移中的一个布尔表达式。如果将守卫条件和事件说明放在一起使用的话，则当且仅当事件发生且布尔表达式成立时，状态转移才发生。如果状态转移只有守卫条件这一个条件，则只要守卫条件为真，状态转移就发生(如图 5.8 所示)。带守卫条件的状态转移的示例如下所示：

```
[  t = 15sec ]
[ number of invoices > n]
withdrawal (amount)  [ balance >= amount]
```

5.2.1.3 动作表达式(Action-Expression)

动作表达式是一个过程表达式，当状态转移开始时执行，如图 5.9 所示。它可以由对象(拥有所有状态的对象)的操作和属性组成，也可以由事件说明中的参数组成。在一个状态转移中，允许有多个动作表达式，但是多个动作表达式之间必须用斜杠(/)分隔开。动作表达式按指定顺序(从左至右)一个一个地执行。不允许有嵌套的动作表达式或递归的动作表达式。但是，只带一个动作表达式的状态转移是可能的。下面就是只有一个动作表达式的状态转移的例子(':= ' 表示赋值)：

```
increase () / n := n + 1 / m := m + 1
add (n) / sum := sum + n
/ flash
```

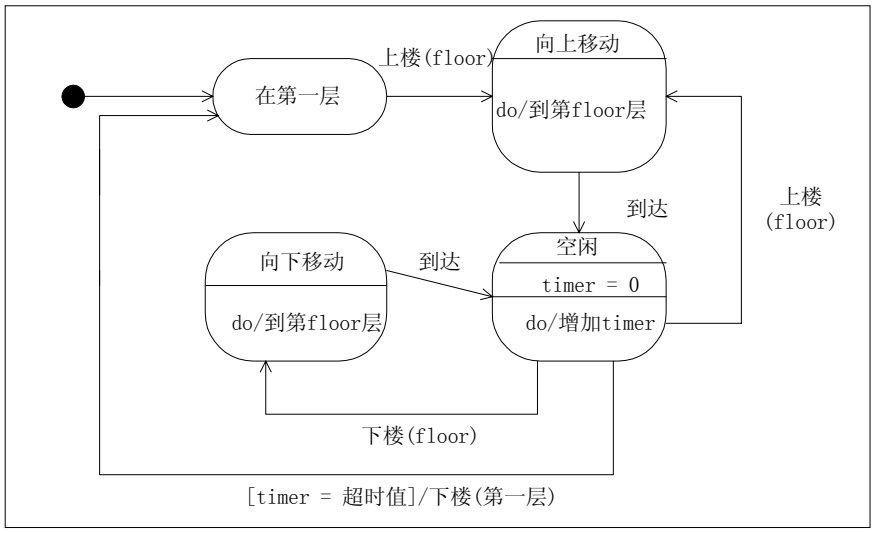

图 5-9 从“空闲”到“在第一层”之间的状态转移：有一个守卫条件和一个动作表达式。当定时器的属性值等于超时值时，动作“下楼(第一层)”被执行；然后状态由“空闲”转移到“在第一层”

5.2.1.4 发送子句

发送子句是动作的特例。它被用来在两个状态转移之间发送消息。发送子句由目的表达式和事件名组成。目的表达式由一个或多个对象组成。事件名是对目的对象(一组对象)有意义的事件的名称。目的对象可以是对象本身。

可以将:

```
[ timer = Time-out ] / go down (first floor)
```

转换成一个发送子句：

```
[ timer = Time-out] ^ self.go down (first floor)
```

再举几个带发送子句的状态转移的例子：

```
out_of_paper () ^ indicator.light()
left_mouse_btn_down(location) / color := pick_color(location) ^ pen.set(color)
```

一般来说，状态图应该比较容易理解些(对模型的要求也一样)，但是要做到这一点，有时是需要很高的技巧，如在描述复杂的内部动态性(对象的内部状态和所有的状态转移)的同时又要求定义一个易理解的模型时。在任何一种情况下，建模者必须在将所有的内部动态属性模型化和对模型进行简化使得人们比较容易理解之间作出选择。

5.2.2 事件

“事件”指的是发生的且引起某些动作执行的事情(参考图 5.10)。例如，当你按下 CD 机上的 Play 按钮时，CD 机开始播放(假定 CD 机的电源已开，已装入 CD 盘且 CD 机是好的)。在此例中，“按下 Play 按钮”就是事件，而事件引起的动作是“开始播放”。当事件和动作之间存在着某种必然的联系时，我们将这种关系称为“因果关系”。在软件工程中，我们常常需要模型化具有这种因果关系的系统。有些情况则不是因果关系，如“一个人在高速公路上高速行驶”和“警察让他停下”之间就不能算是因果关系，因为动

作“警察让他停下”不一定发生，因而在这两者之间不存在着必然的联系。

“事件”的准确定义可以参考"Cambridge Dictionary of Philosophy" 。

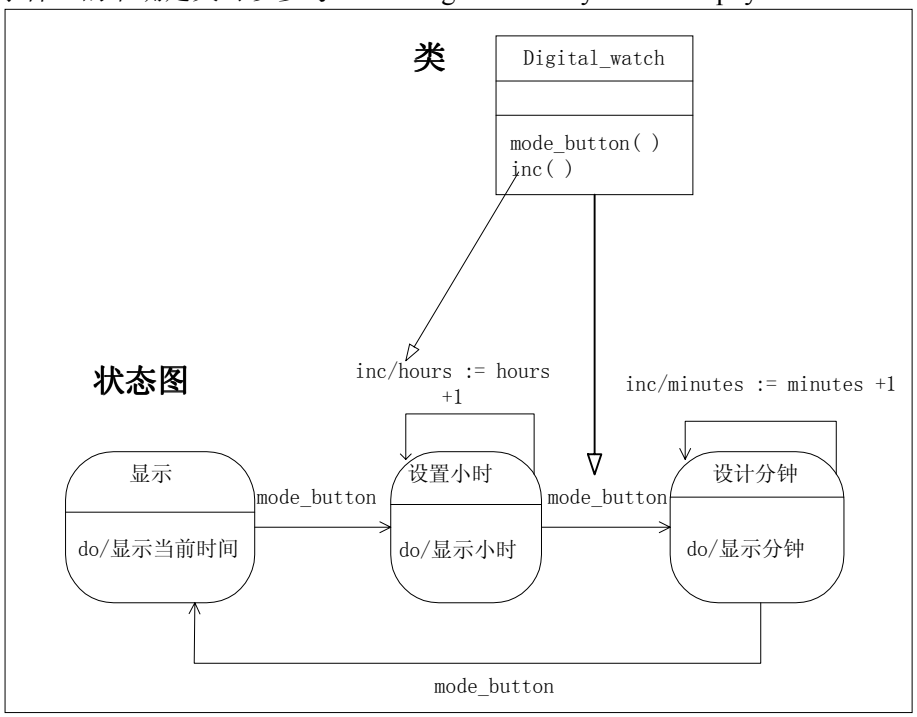

图 5-10 类 Digital_watch 及其状态图。本图说明状态图中的事件与类中操作的关系。该对象有三个状态：正常显示状态，两个设置时钟的状态(分别为小时和分钟)

UML 中有四类事件：

- 条件成真：即状态转移上的守卫条件。
- 收到另一个对象中的信号：信号本身也是一个对象。在状态转移中，表示为事件说明。这类事件也称作消息。
- 收到另一个对象(或对象本身)的操作调用：在状态转移中，表示为事件说明。这类事件也称为消息。
- 经过指定时间间隔：通常情况下，时间是从另一个指定事件(通常是当前状态的入口)或一给定时间段之后开始计算的。在状态转移中，表示为时间表达式。

需要注意的是，错误也可以是事件，对建模也许有用。UML 并不对错误事件提供直接的支持，但是可以将错误事件定义成版类(stereotype)，例如：

```
<<error>>  out_of_memory
```

了解事件的一些基本语义是很重要的。首先，事件是触发状态转移的触发器，并且一个事件只能被处理一次。如果一个事件可能触发多个状态转移，则只能有一个状态转移发生(具体触发哪一个是未定义的)。如果事件发生了，且状态转移的守卫条件为假，则事件被忽略(也不保存事件，如果守卫条件在后来成真则触发状态转移)。

类可以接收和发送消息，也就是说，可以调用操作或发送、接收信号。状态转移中的事件说明可以用来描述消息的发送和接收。当操作被调用时，操作开始执行并产生结果当一个信号被发送时，接收者接收该信号对象并使用它。信号是普通的类，但仅用来发送信

号；它们表示系统中的对象间的发送单元。信号类可以是版类，类型为<<signal>>，通过该版类来限制对象的语义，使得它只能用作信号。也可以建立信号的层次关系来支持多态，使得如果给一个状态转移的事件说明指定了一个信号时，所有的子信号也能被同一个事件说明收到(参考图 5-11)。

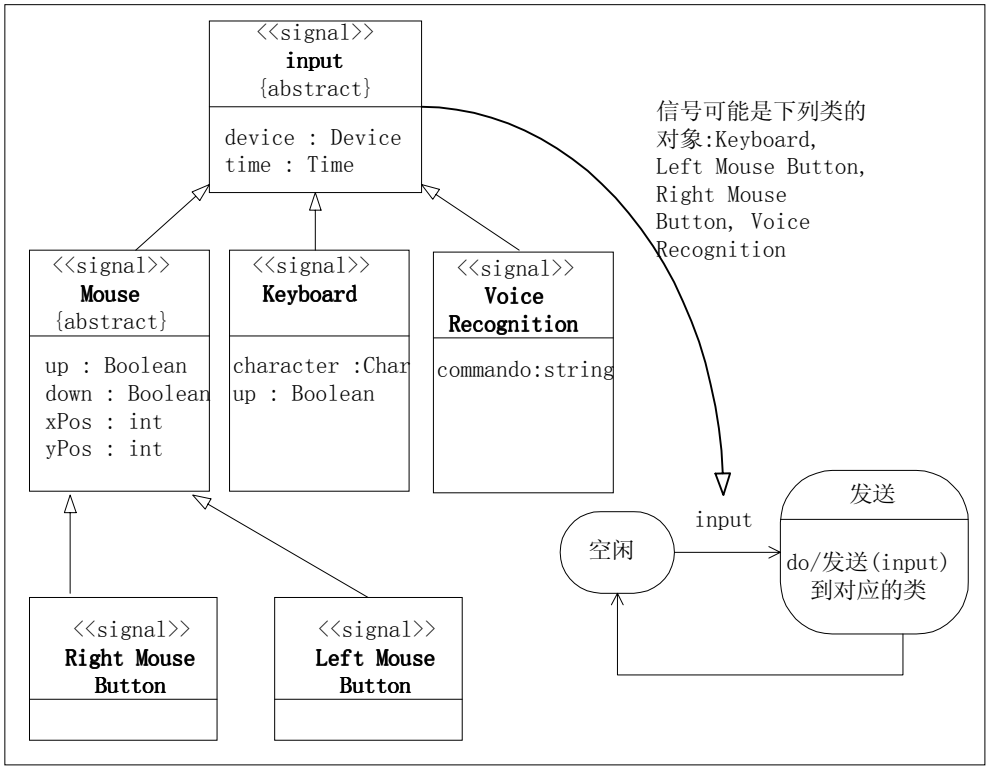

图 5-11 带有抽象超类的信号类的层次结构。右边的状态图接收输入信号(包括送给 Input 类的子信号)。只能发送具体的信号(因为抽象信号类没有任何实例)

5.2.3 Java 实现

在某些情况下，状态图中有一些冗余信息，这主要取决于类中的操作是否有指定的算法。换言之，类的行为可以通过操作的算法来指定或直接在状态图中说明或者两者同时使用。当用面向对象的编程语言来实现状态图时，要么直接在算法中实现状态图(用 case 语句等)，要么用独立的机制来实现，如有限状态自动机(finite state machines)或功能表(function table)。有关这方面的细节超出了本书的范围，在此不再描述。但是，图 5-12 描述了有关在类的操作中实现状态图的方法。

UML 并不就如何在编程语言中实现信号事件提供任何建议。然而，实现信号是很简单的：只需用包含属性和公共操作的类来实现即可。接收信号的类必须有一个对应的接收操作，且将信号对象作为操作的参数。

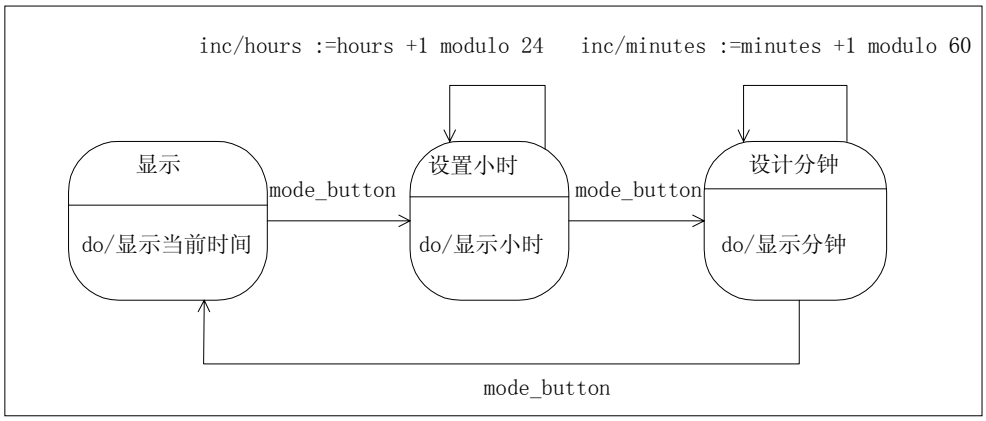

图 5-12 数字表的状态图

带有状态图的类的 Java 实现类似于下面的代码：

```
public class State
{
   public final int Display = 1;
   public final int Set_hours =2;
   public final int Set_minutes =3;
   public int value;
}

public class Watch
{
  private State state = new State();
  private DigitalDisplay LCD = new DigitalDisplay();

  public Watch()
       {
          state.value = State.Display;
          LCD.display_time();
       }

  public void mode_button()
      {
         switch (state.value)
          {
            case State.Display :
  LCD.display_time();
  state.value = State.Set_hours;
  break;
 case State.Set_Hours:
  LCD.display_hours();
  state.value = State.Set_minutes;
```

```
        break;
         case State.Set_minutes :
        LCD.display_time();
        state.value = State.Display;
        break;
      }
           }

        public void inc()
{
       switch (state.value)
        {
          case State.Display:
           break;
          case State.Set_hours:
           LCD.inc_hours();
          break;
          case State.Set_minutes:
LCD.inc_minutes();
   break;
              }
         }
   }
```

5.3 状态图之间发送消息

状态图可以给其它的状态图发送消息。状态图间的消息发送可以通过动作(即，在发送子句中指定接收者)或在状态图间的虚线箭头来表示。如果使用虚线箭头来表示，则必须将状态图中的所有对象组合在一起(即使用类矩形符号)。矩形符号也可以用来描述子系统或系统(一种宏类)。可以采用两种不同的技术来画出表示状态图间的消息发送的虚线箭头。

第一种方法是从源对象的状态转移画虚线箭头至目标对象的边缘(这种方法实际上是发送子句中的文本表示方法的另一种表示)。然后，在目标对象中画一条状态转移线，表示收到指定的消息。

第二种方法是从源对象画虚线箭头到目标对象，表示在执行过程中的某一时刻，源对象发送消息给目标对象。不管怎样，目标对象必须有对应状态转移说明来接收消息(参考图 5-13)。

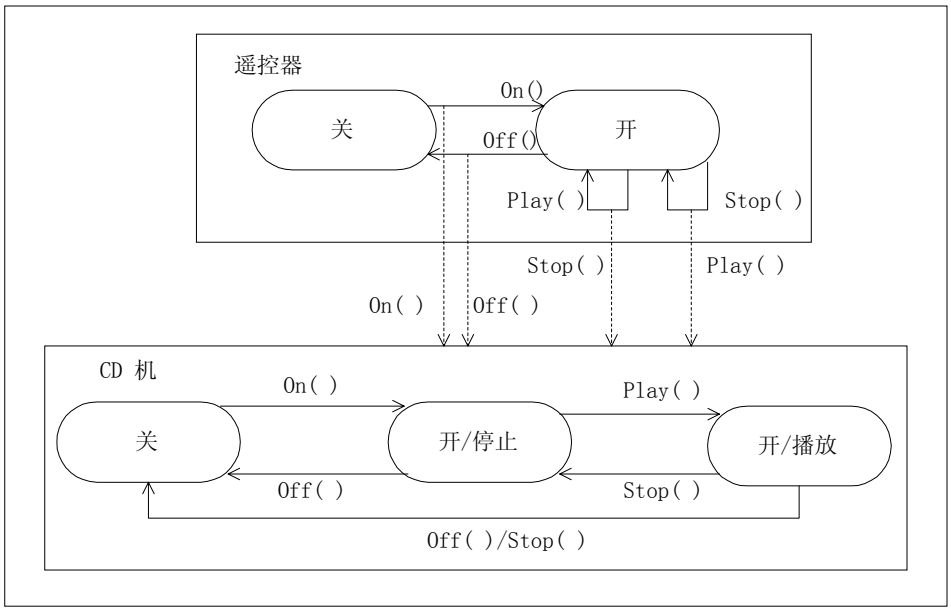

图 5-13 遥控器(Remote Control)发送消息给 CD 机

5.3.1 子状态

状态可能有嵌套的子状态，且子状态可以在另一个状态图。子状态又可分为两种：与子状态(and-substate)，或子状态(or-substate)。与子状态指的是一个状态可以有子状态，但是一次只能有一个子状态，如图 5-14 所示。例如，一辆车可以处于运行态，它的运行态可以有两个子状态：前进和后退，它们是或子状态，因为它们不能同时为真。嵌套的子状态可以显示在另一个状态图中，方法是在初始状态图中扩展运行状态。

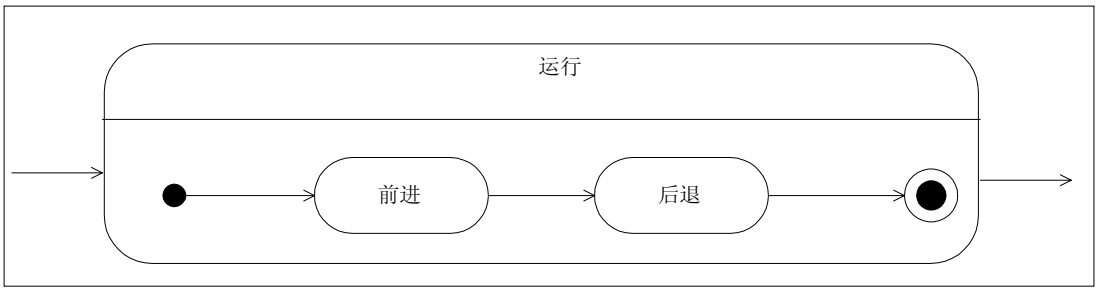

图 5-14 或子状态

另一方面，运行态可能有多个并行的子状态(与子状态)：前进和低速，前进和高速，后退和低速，后退和高速。当一个状态有与子状态且它们中的几个可以同时为真时，表示一个状态既有与子状态也有或子状态，如图 5-15 所示。与子状态也称作并行状态，可以用来抽象并行线程的状态。有关细节将在第 8 章讨论。

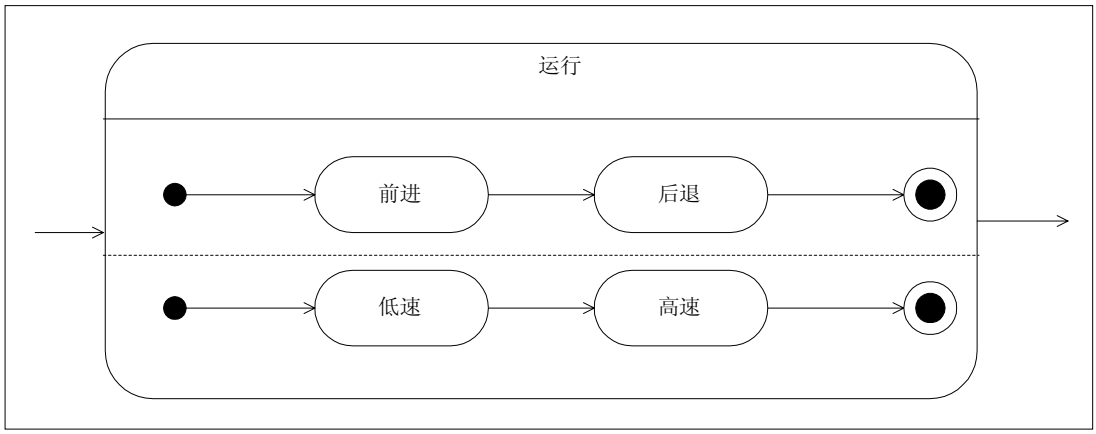

图5-15　与子状态(也有或子状态)

5.3.2　历史指示器

历史指示器被用来存储内部状态，例如，当对象处于某一状态，经过一段时间后可能会返回到该状态，则可以用历史指示器来保存该状态。可以将历史指示器应用到状态区。如果到历史指示器的状态转移被激活，则对象恢复到在该区域内的原来的状态。历史指示器用空心圆中放一个‘H’来表示。可以有多个指向历史指示器的状态转移，但没有从历史指示器开始的状态转移，参考图 5-16。

5.4 序 列 图

序列图描述对象是如何交互的，并且将重点放在消息序列上，也就是说，描述消息是如何在对象间发送和接收的。序列图有两个坐标轴：纵坐标轴显示时间，横坐标轴显示对象。序列图也显示特殊情况下的对象交互：在系统执行期间的某一时间点发生在对象间的特殊交互(如，当用到一个特殊功能时)。

在序列图的横坐标轴上是与序列有关的对象。每一个对象的表示方法是：矩形框中写有对象和/或类名，且名字下面有下划线。同时，有一条纵向的虚线表示对象在序列中的执行情况(即，发送和接收的消息，对象的活动)，这条虚线称为对象的“生命线”。对象间的通信用对象的生命线之间的水平的消息线来表示。消息线的箭头说明消息的类型，如同步，异步，或简单(本章前面定义的)。浏览序列图的方法是：从上到下查看对象间交换的消息。

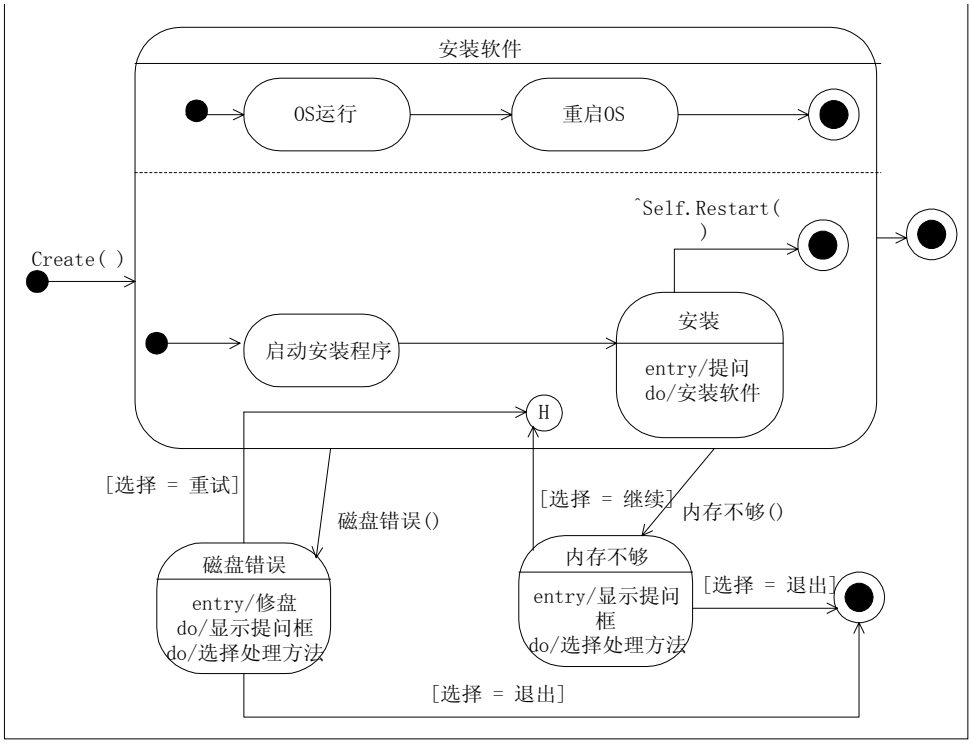

图 5-16 软件安装程序。历史指示器被用来处理错误，如“内存溢出”，“磁盘错误”等。当错误状态被处理完后，用历史指示器返回到错误之前的状态

5.4.1 一般和实例格式

有两种使用序列图的方式：一般格式和实例格式。实例格式详细描述一次可能的交互。实例格式没有任何条件、分枝或循环，它仅仅显示选定的情节的交互。而一般格式则描述所有的情节，因此，包括了分枝、条件和循环。例如，用一般格式来描述序列图中的情节“打开一个帐户”时，所有有关的细节都将体现出来：操作是否成功，是否不允许客户打开帐户，是否立即将钱从帐户中取出来，等等。同样的情况，如果用实例格式来描述，则描述的内容就不一样。它仅仅选择一个特定情节来描述，例如，一个图可能只显示成功地打开帐户。如果要显示所有的情节，则需要很多个实例序列图。

一条消息是一次对象间的通信，通信所传递的信息是期望某种动作发生。通常情况下，接收到一条消息被认为是一个事件。消息可以是信号，操作调用或其它类似的东西(如，C++中的 RPC(Remote Procedure Calls)或 Java 中的 RMI(Remote Method Invocation))。当对象收到一条消息时，活动就开始，称之为“激活(activation)”。激活显示控制的焦点，对象及时地在某一点执行。一个被激活的对象要么执行自己的代码或等待另一个对象返回结果。“激活”用对象的生命线上的窄的矩形来表示。生命线表示一个对象在一个特定时间内的存在，它是一条从上到下的虚线。消息用对象的生命线间的箭头线(同步，异步，简单)表示。每一条消息可以有一个说明，内容包括名称和参数，例如：

```
print (file : File)
```

消息也可以有序列号，虽然序列号不常用(因为消息序列是在图中直接表示的)。返回

(从同步消息，如操作调用)也被显示成一个箭头(用简单箭头)。但是，有一点要注意，返回不一定显示出来(参考图 5-17)。

消息也可以有条件。只有条件为真时才可以发送和接收消息。条件被用来抽象分枝或决定是否发送一条消息。如果条件被用来描述分枝，则会有好几个互斥的消息箭头，换言之，一次只能发送一条消息，如图 5-18 所示。如果用条件来抽象分枝，则各分枝并不互斥，可以并行发送消息。一个对象可以给自己发送消息，在这种情况下，消息符号是从对象符号指向对象本身，如图 5-19 所示。

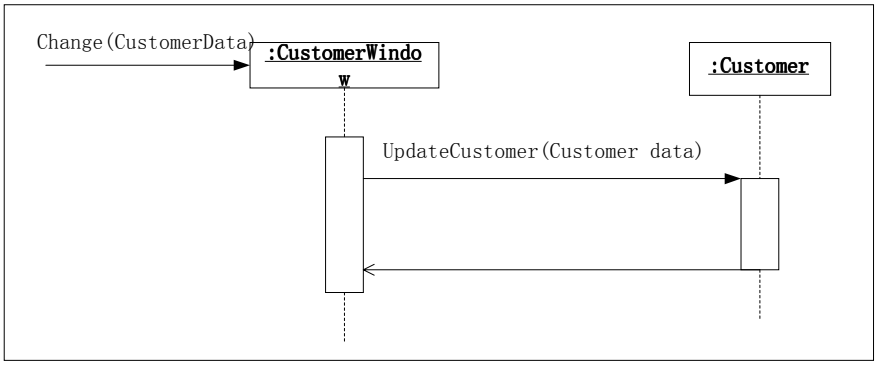

图 5-17　描述一个特定情节的序列图，从 Change 消息开始。图中显示了 UpdateCustomer 消息的返回结果

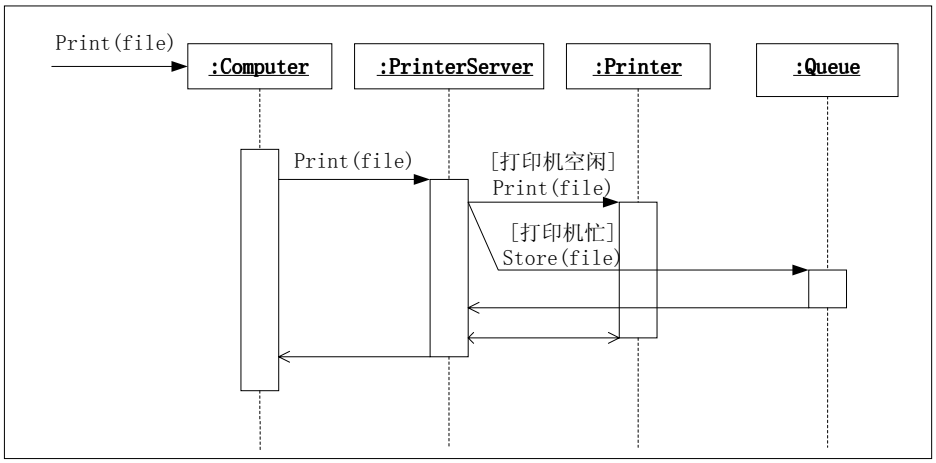

图 5-18　从 PrintServer 到 Printer 的消息带有条件，说明在序列图中如何描述选择。要么发送 Print 消息到 Printer，要么发送消息 Store 给 Queue

5.4.2　并发对象

在某些系统中，对象并发执行，每一个对象有一条自己的控制线程。如果系统使用这样的并发对象，则通过激活、异步消息和活动对象来表示。有关这个问题的细节将在第 8 章中作进一步讨论。

5.4.3　定义迭代和约束的标签

在对象图的左边和右边可以有标签和注释。标签可以是任何类型，如定时标记，激活

过程中的动作描述、约束，等等，如图 5-20 所示。通常情况下，循环就是用边缘注释来描述的，如图 5-21 所示。也可以用标签来描述定时约束。例如，限制两条消息之间的间隔，等待一条消息到达的时间(状态转移时间)。

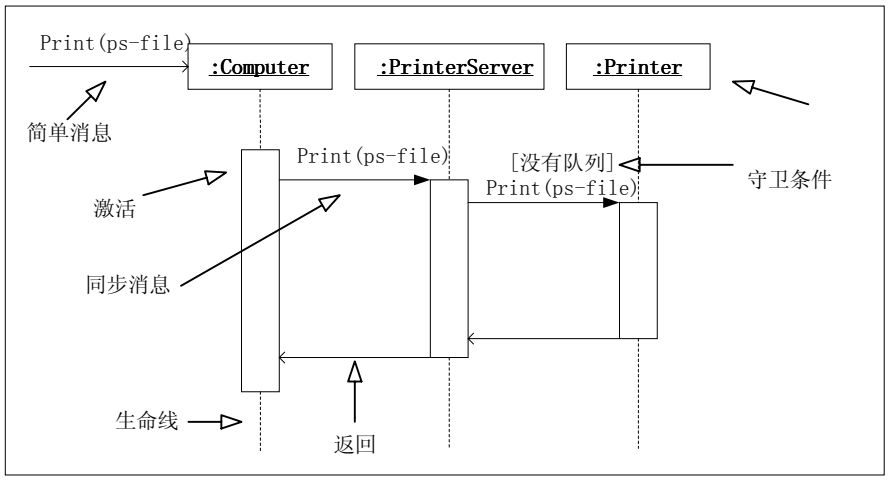

图 5-19　序列图中的概念

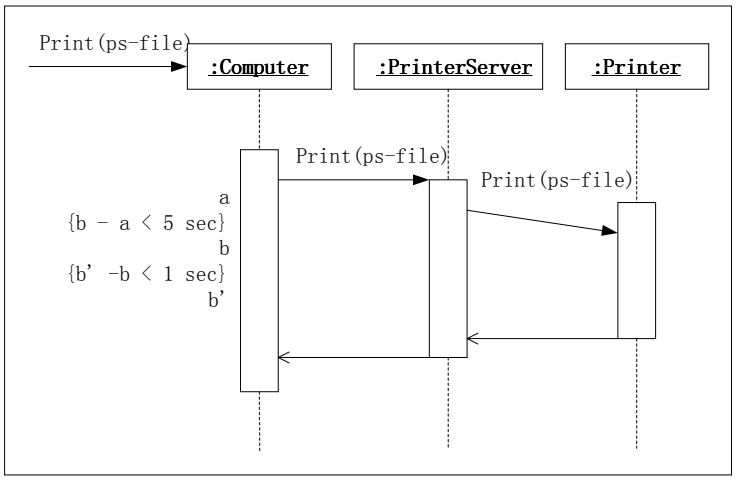

图 5-20　用来指定时间约束的标签。a 与 b 之间的时间间隔不能大于 5 秒。PrintServer 发送的消息 Print 必须在 1 秒内收到。下斜的箭头表示发送和接收消息之间的时间是足够的。通常情况下，将约束用大括号括起来

5.4.4　创建和破坏对象

在序列图中可以描述如何创建和破坏对象，并把它作为描述情节的一部分。一个对象可能通过一条消息来创建另一个对象。被创建的对象的对象符号放在创建它的地方(在纵坐标时间轴上)。创建和破坏对象的消息一般是同步消息。当一个对象被破坏后，用一个大 X 来标记。它进一步说明，对象的生命线只需划到对象被破坏时为止。

5.4.5 递归

在很多算法中，递归是一种常用的技术。当一个操作调用它自己时就是递归，如图 5-23 所示。在图 5-23 中，递归为激活它自己。当一个操作调用它本身时，消息总是同步的，因而可以用序列图中同样的方法来标记它。用一条简单消息来表示返回。

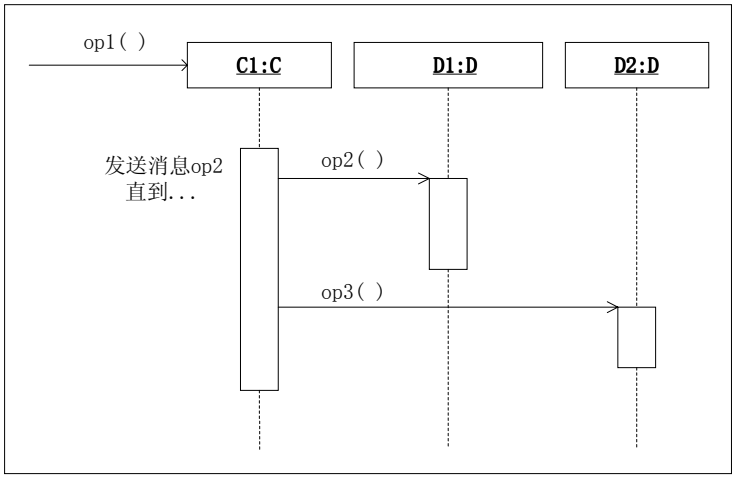

图 5-21 在边缘处表示的迭代

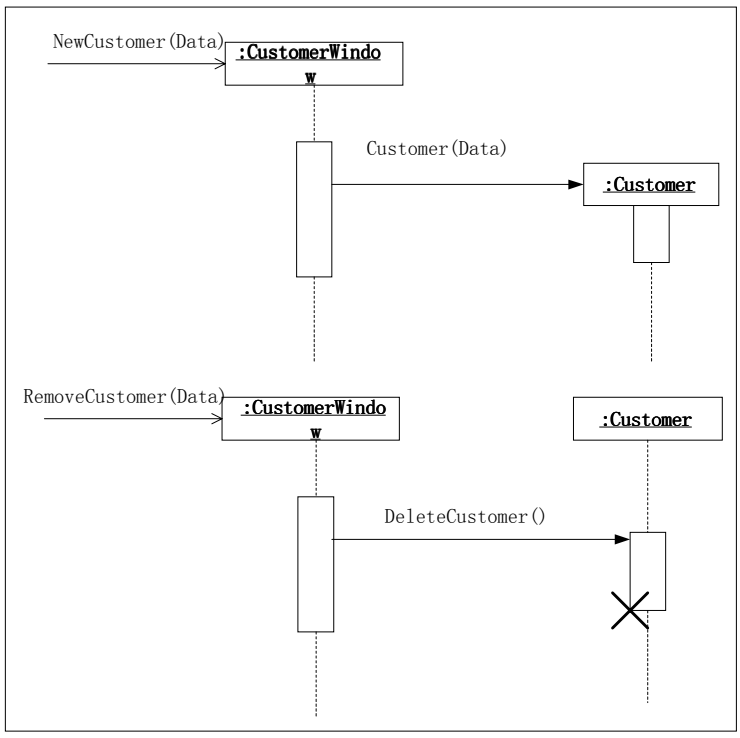

图 5-22 Customer 对象创建一个 Customer 类的新的对象(通常由 Customer 类的构造器来处理，新的对象与类有同样的名字)。DeleteCustomer 操作破坏 customer 对象。可以直接显示创建和破坏操作的返回，但它不在这些图中

5.5 协 作 图

协作图(Collaboration Diagram)主要描述协作对象间的交互和链接(一条链接是一个关联的实例化)。序列图和协作图都描述交互，但是序列图强调的是时间，而协作图强调的是空间。链接显示真正的对象以及对象间是如何联系在一起的。可以只显示对象的内部结构(构成对象的对象显示在对象的内部)。同序列图一样，协作图也可以说明操作的执行，用例的执行或系统中的一次简单的交互情节。

正如前面所说，并发和异步消息不在此讨论。它们将在第 8 章中描述。

协作图显示对象、对象间的链接以及链接对象间如何发送消息。用同类一样的符号来表示对象，但是对象的名字下面有下划线(对象符号)。链接用线条来表示(有点象关联，但没有重数)。在一条链接上，可以给消息加一个消息标签用来定义消息的序列号。定义标签需要一种特别的语法，将在后来给出。协作图从初始化整个交互或协作的消息开始，例如，一个操作调用(如图 5-24 所示)。

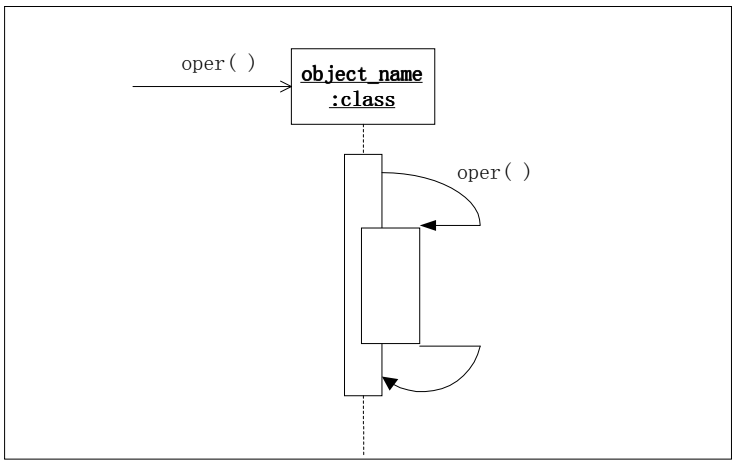

图 5-23 oper()操作调用其本身。操作中必须有一个停止递归的条件。返回被直接显示

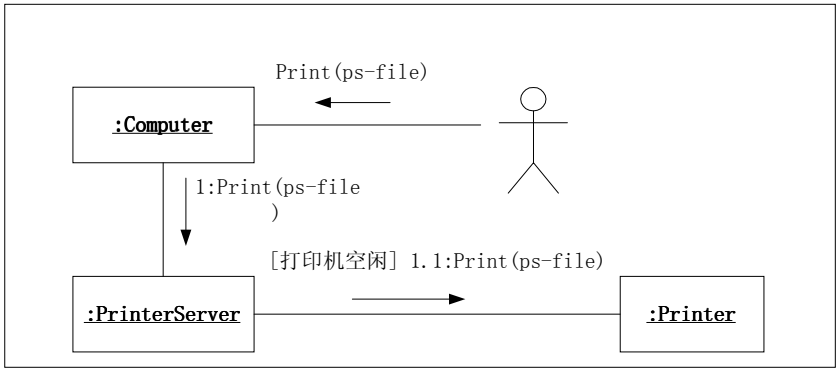

图 5-24 actor 发送 Print 消息给 Computer。Computer 发送一个 Print 消息给 PrintServer。如果打印机空闲 PrintServer 发送 Print 消息给 Printer

5.5.1 消息流

协作图中的消息标签(放在消息上的标签)用下面的语法规则来书写：

前缀 守卫条件 序列表达式 返回值 := 说明

其中，前缀(predecessor)用下面的语法来描述：

序列号 ',' ... '/'

前缀是一个用来同步线程或路径(path)的表达式，意思是在发送当前消息之前指定序列号的消息被处理(也就是说，必须连续执行)。序列号之前用逗号隔开。

守卫条件用下面的语法来描述：

'[' 条件短句 ']'

条件短句通常用伪代码或真正的程序语言来表示。UML 并不规定其语法。

序列表达式的语法规则定义如下：

[integer | name] [recurrence] ':'

其中，integer 为指定消息顺序的序列号。消息 1 总是消息序列的开始消息，消息 1.1 是消息 1 的处理过程中的第一条嵌套的消息，消息 1.2 是消息 1 的处理过程中的第二条嵌套的消息。一个消息序列的例子如：消息 1, 消息 1.1, 消息 1.2, 1.2.1, 1.2.2, 1.3, 等等。这样的序列号不仅能够表示消息的顺序，而且还能表示消息的嵌套关系(当消息是异步消息时，消息为嵌套的操作调用及返回)。name 表示并发控制线程，有关细节在第 8 章中讨论。例如，1.2a 和 1.2b 为同时发送的并发消息。序列表达式用冒号表示。

recurrence 表示一个条件或迭代的执行。有两种选择：

'*' '[' 循环子句 ']'

'[' 循环子句 ']'

循环子句(iteration-clause)用来指定一个循环(重复执行)，循环子句是循环的条件，如[i := 1...n]。例如，对于一个包括循环的消息标签应该表示成：

```
1.1  *[x = 1..10] : doSomething()
```

而条件子句一般用来表示分枝，而不是用作守卫条件。[x < 0]和[x => 0]是两个可以用来分枝的条件子句，这两个条件只能有一个为真，因而只有一个分枝被执行(发送与分枝有关的消息)。条件子句和循环子句都可以用伪代码或真正的编程语言来表示。

应该给返回值指定一个消息说明。消息说明由消息名和参数表组成。返回值表示一个操作调用(消息)的结果。举一个返回值的例子，如消息标签 1.4.5: x := calc (n) (参考图 5-25)。下面是一个消息标签例子：

```
1: display ()
[mode = display] 1.2.3.7: redraw()
2 * [n := 1..z] : prim := nextPrim (prim)
3.1 [x<0] : foo()
3.2 [x=>0] : bar()
1.1a, 1.1b/1.2: continue()
```

5.5.2 链接

一条链接是两个对象间的连接。链接上的任何对象的角色名均作为链接的端点，与链

接的量词在一起。量词和角色均在对象类的类图中说明。有一些版类可能会同链接上的角色(链接角色)放在一起：global, local, parameter, self, vote, broadcast。

- Global：Global 是加在链接角色上的约束，说明与对象对应的实例是可见的，因为它是在全局范围内(可以通过系统范围内的全局名来访问对象)。
- Local：Local 也是加在链接角色上的约束，说明与对象对应的实例是可见的，因为它是操作中的一个局部变量。
- Parameter：Parameter 也是加在链接角色上的约束，说明与对象对应的实例是可见的，因为它是操作中的一个参数。
- Self：Self 是加在链接角色上的约束，说明一个对象可以给自己发送消息。
- Vote：Vote 是加在消息上的约束，限制一组返回值。Vote 限制说明返回值必须在返回的值中通过多数投票才能选出。
- Broadcast：Broadcast 是加在一组消息上的约束，说明这组消息不按一定按顺序激活。

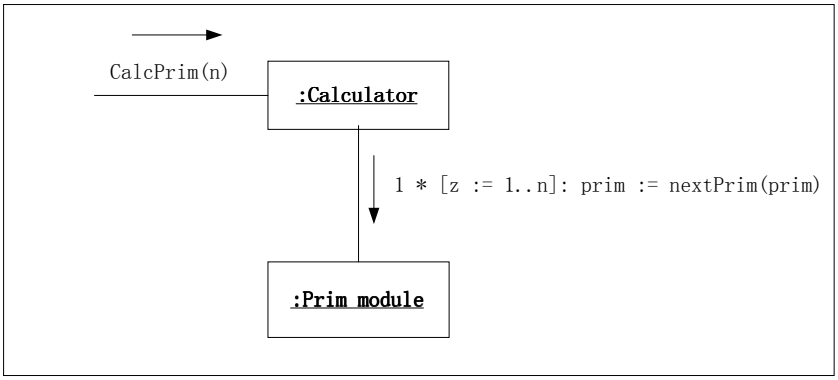

图 5-25　协作图中的循环和返回值

5.5.3　对象的生命周期

在给一个协作指派一个{new}期间，对象被创建，如图 5-26 所示。在给一个协作指定约束{destroyed}期间，对象被破坏。如果给协作指定{transient}，则对象被创建和破坏，相当于{ new } {destroyed} (参考图 5-27)。

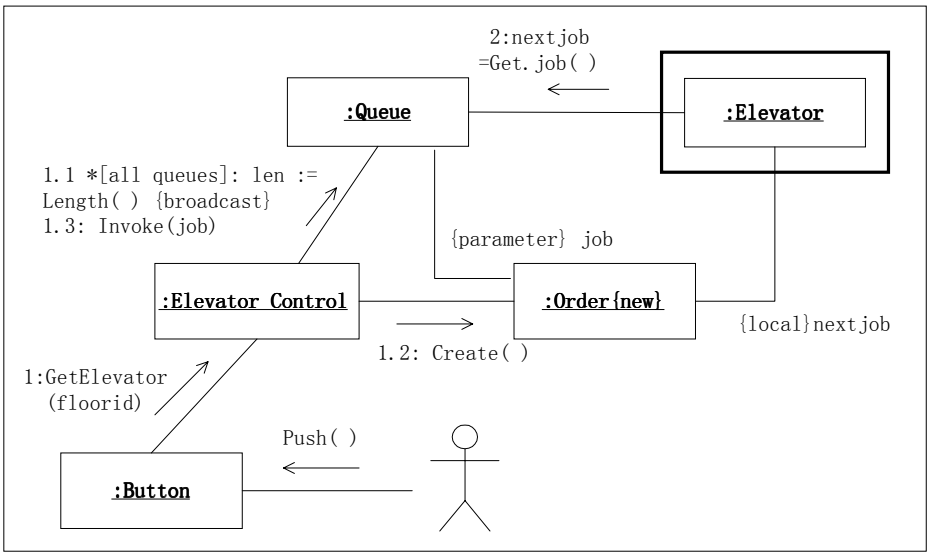

图 5-26 有关一个角色按下一个按钮让电梯到达他想去的楼层的协作图。电梯控制对象检查所有电梯的工作队列的长度并选择一个最短的。然后，它创建一个作业命令对象并将它放入队列激活它。电梯对象并发运行并从它的队列中选取一个作业。电梯是一个活动对象，意思是它与它的控制线程并发执行。活动对象将在第 8 章中讨论

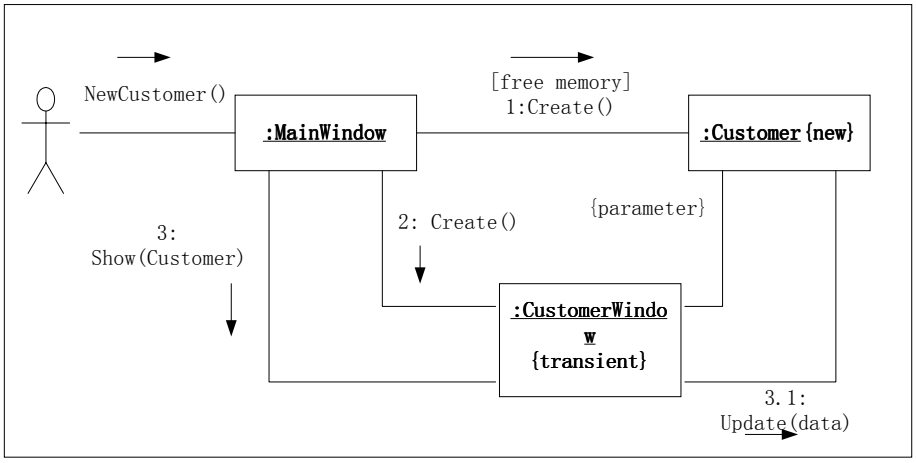

图 5-27 对象 MainWindow 收到消息 NewCustomer，创建 Customer 对象。创建 CustomerWindow，然后将 customer 对象传递给 CustomerWindow，从而实现客户数据的更新

5.5.4 使用协作图

可以用协作图来表示相当复杂的对象间的交互。尽管学习消息的编号机制需要花些时间，但是一旦学会，却相当容易使用。协作图和序列图间的主要区别在于协作图显示真正的对象及其链接(正在协作的“对象网络”)，在许多情况下，这有利于理解对象的交互。而时间序列在序列图更容易看出来，从上至下看即可。当要决定选用哪一种图时，一般的原则是当对象及其链接有利于理解交互时选择协作图，当只需了解序列时选择序列图。

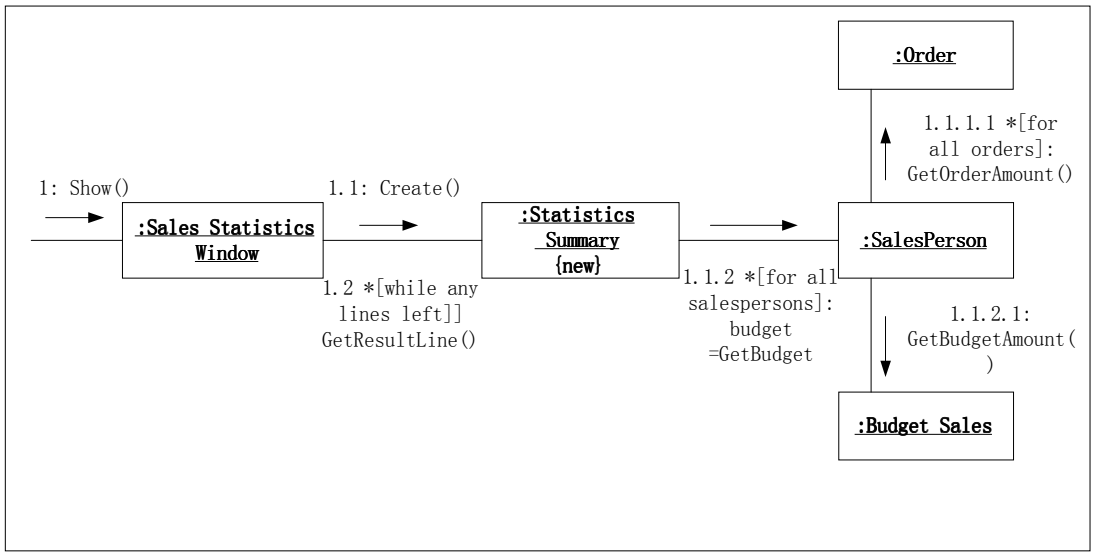

图 5-28 统计销售结果的协作图

图 5-28 显示了一个更复杂的协作图。一个 Sales Statistics window(消息 1)，创建一个 Statistics Summary 对象(1.1)，该对象收集统计信息显示在窗口中。当 Statistics Summary 对象被创建后，它不断地循环得到推销员的订单(1.1.1)和预算(1.1.2)。第一个 Salesperson 对象得到它的所有订单(1.1.1.1)，将它们加在一起，从 BudgetSales 对象得到预算。当 Statistics Summary 对象收集完所有推销员的数据后，它被创建完成(消息 1.1 返回)。然后，Sales Statistics window 对象从 Statistics Summary 对象得到结果行(结果行是指描述一个推销员的结果的格式化字符串)并将每一行显示在窗口中。当读完所有的结果行，Sales Statistics window 的显示操作返回，协作完成。

5.6 活 动 图

活动图(activity diagram)显示动作及其结果。活动图着重描述操作(方法)实现中所完成的工作以及用例实例或对象中的活动。活动图是状态图的一个变种，与状态图的目的有一些小的差别，活动图的主要目的是描述动作(执行的工作和活动)及对象状态改变的结果。当状态中的动作被执行(不象正常的状态图，它不需指定任何事件)时，活动图中的状态(称为动作状态)直接转移到下一个阶段。活动图和状态图的另一个区别是活动图中的动作可以放在泳道中。泳道聚合一组活动，并指定负责人和所属组织。活动图是另一种描述交互的方式，描述采取何种动作，做什么(对象状态改变)，何时发生(动作序列)，以及在何处发生(泳道)。

活动图可以用作下述目的：

- 描述一个操作执行过程中(操作实现的实例化)所完成的工作(动作)。这是活动图最常见的用途。
- 描述对象内部的工作。

- 显示如何执行一组相关的动作，以及这些动作如何影响它们周围的对象。
- 显示用例的实例是如何执行动作以及如何改变对象状态。
- 说明一次商务活动中的工人(角色)、工作流、组织和对象是如何工作的。

5.6.1 动作和转移

执行动作就会产生结果。可以用一组相关动作来描述操作的实现，然后将这些动作转换成代码行。前面已提到过，活动图描述动作及动作之间的关系，有一个起点和一个终点。起点用黑圆点来表示，终点用黑圆点外加一个小圆来表示。活动图中的动作用一个圆角四边形表示(使用状态图中同样的记法，参考图 5-29)。

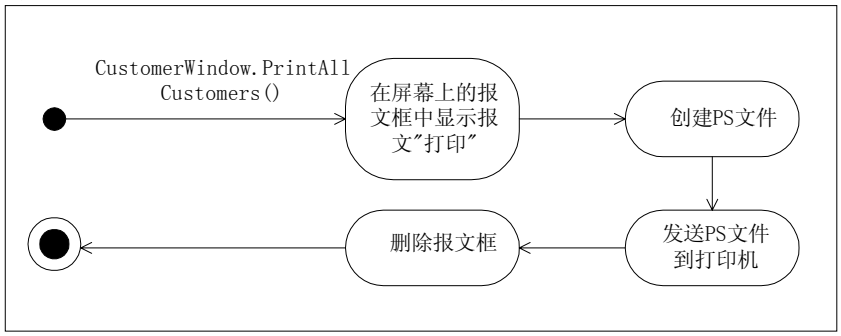

图 5-29 当调用 PrintAllCustomer 操作(在 CustomerWindow 类中)时，动作开始。第一个动作是在屏幕上显示一个消息框；第二个动作是创建一个 PS(postscript)文件；第三个动作是发送创建的文件给打印机；第四个动作是从屏幕上删除消息框。转移是自动进行的，只要源状态中的动作被执行，转移就开始

在动作内，一个文本串被用来说明动作或采取的动作。除了事件外，动作之间的转移的描述方法与状态图中所用的方法一致。事件可能只与从起点到第一个动作之间的转移联系在一起。动作之间的转移用箭头来表示，箭头上可能还带有守卫条件，发送短句和动作表达式。通常情况下，箭头上不带任何东西，表示只要动作状态中的所有活动一完成转移就开始。

用守卫条件来约束转移，如前所述，只有守卫条件为真时转移才可以开始。守卫条件的语法规则与状态图中的守卫条件的语法规则一致。用守卫条件来作决策，例如，[yes]和[no]。用菱形符号来表示决策点，如图 5-31 所示。一个决策符号可以有一个或多个进入转移，两个或更多的带有守卫条件的发出转移。正常情况下，出转移中的一个总是真。

可以将一个转移分解成两个或更多的转移，从而导致并发的动作。动作并发执行，虽然也可以串行地执行动作。重要的是所有的并行转移在合并之前(如果它们曾合并)必须被执行。一条粗黑线表示将转移分解成多个分枝，表示真正分解成并发动作。同样用粗黑线来表示分枝的合并(图 5-32)。

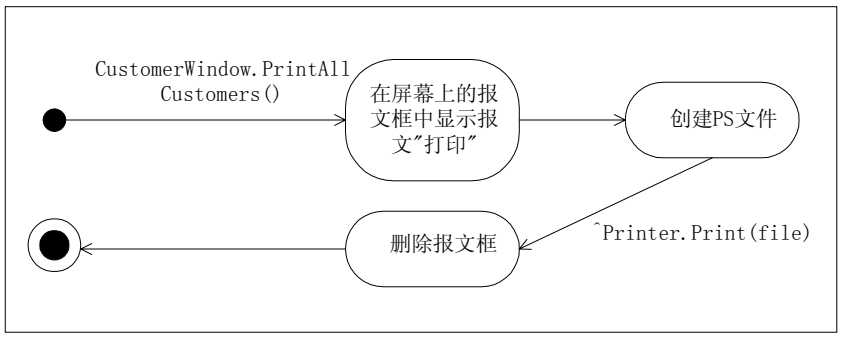

图 5-30 在第二个动作和第三个动作之间的转移带有一个发送短句，用来表示发送一个 Print(file)消息给 Printer 对象

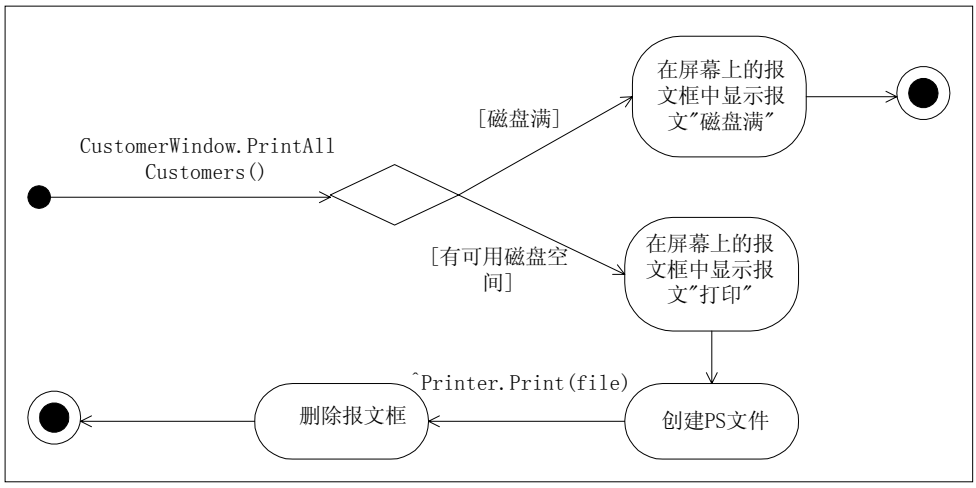

图 5-31 如果磁盘满，弹出“磁盘满”报文框。否则，弹出“打印”报文框

5.6.2 泳道

如前所述，泳道被用来组合活动。通常情况下，根据活动的功能来组合。具体来说，泳道有以下几个目的：例如，直接显示动作在哪一个对象中执行，或显示执行的是一项组织工作的哪一部分。泳道用纵向矩形来表示。属于一个泳道的所有活动均放在其矩形符号内。泳道的名字放在其矩形符号的顶部(图 5-33)。

5.6.3 对象

对象可以在活动图中显示。对象可以作为动作的输入或输出，或简单地表示指定动作对对象的影响。对象用对象矩形符号来表示，在矩形的内部有对象名或类名。当一个对象是一个动作的输入时，用一个从对象指向动作的虚线箭头来表示；当对象是一个动作的输出时，用一个从动作指向对象的虚线箭头来表示。当表示一个动作对一个对象有影响时，只需用一条对象与动作间的虚线来表示。作为一个可选项，可以将对象的状态用中括号括起来放在类名的下面，如[planned], [bought], [filled], 等等，参考图 5-34。

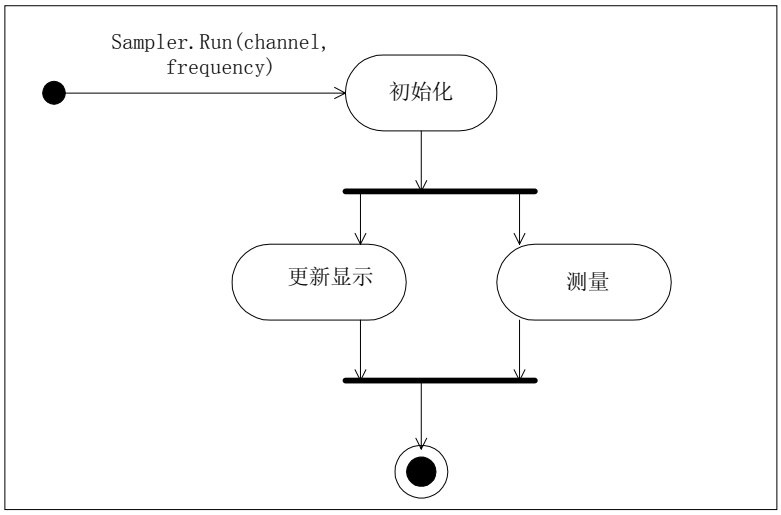

图 5-32 当 Run 操作被调用时，发生的第一个动作是初始化。然后是并发执行的两个动作：更新显示和测量

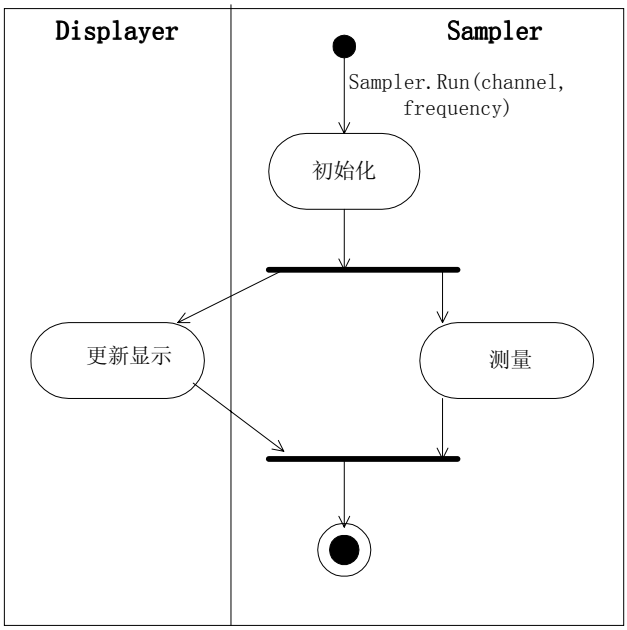

图 5-33 动作更新显示在 Displayer 内执行。动作初始化和测量在 Sampler 内执行

5.6.4 信号

可以在活动图中发送和接收信号(在本章的前面对信号已有描述)。有两个与信号有关的符号，一个表示发送信号，另一个表示接收信号。发送符号对应于与转移联系在一起的发送短句。同发送短句一样，发送和接收符号均同转移联系在一起。但是，从图形角度看，转移又分为两种：发送信号的转移和接收信号的转移。

发送和接收符号可以同消息的发送对象或接收对象联系在一起。具体表示方法是从发送或接收符号画一条虚线箭头到对象。如果是发送符号，则箭头指向对象；如果是接收符

号，则箭头指向接收符号。对象显示是可选的。发送符号是凸出五边形，而接收符号为凹入五边形，如图 5-35 所示。

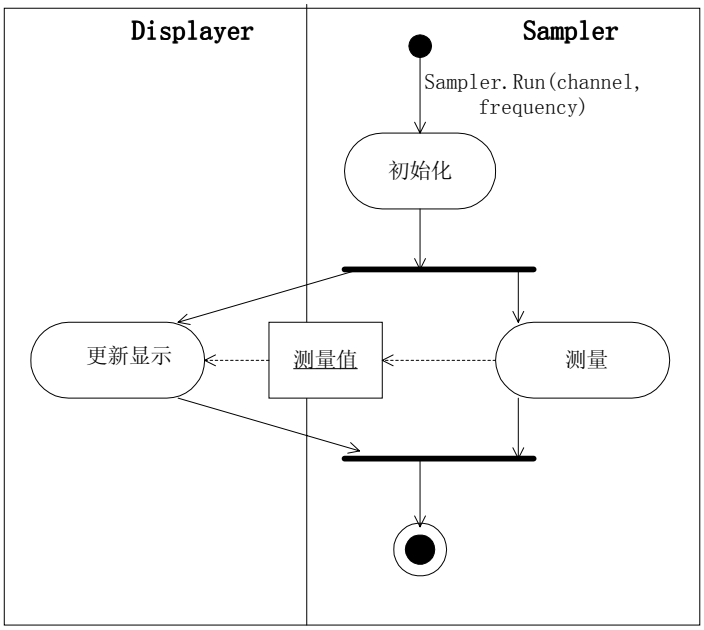

图 5-34 Measuring 动作给 Updating displayer 动作提供测量值(测量值是一个对象)

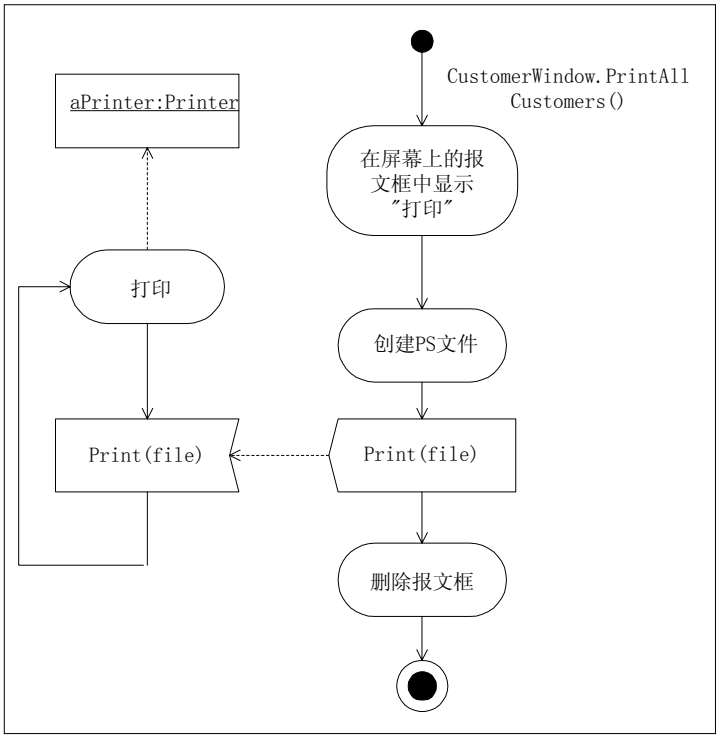

图 5-35 在创建 postscript 文件与删除消息框之间，Print 信号被发送。信号带有参数 file，代表 Printer 对象收到并打印的文件

5.6.5 运用活动图进行商业建模

根据 Nilsson(1991)所述，在商业建模时，下列方面是模型要着重描述的：资源、规则、目的和动作(工作流)。有两类资源：物质和信息。工人(商业活动中的角色)就是资源的例子，他们是物理对象。其它的物理对象可能是生产的、消费的或处理的物品。信息对象通常是被信息系统处理的对象。信息对象携带与商务有关的信息。商务规则约束资源的使用，包括物品和信息。例如，一条规则规定物品在搬运过程中不能损坏。另一条规则可能规定战略信息必须保密。这些资源的使用是真正的工作，称为工作流。

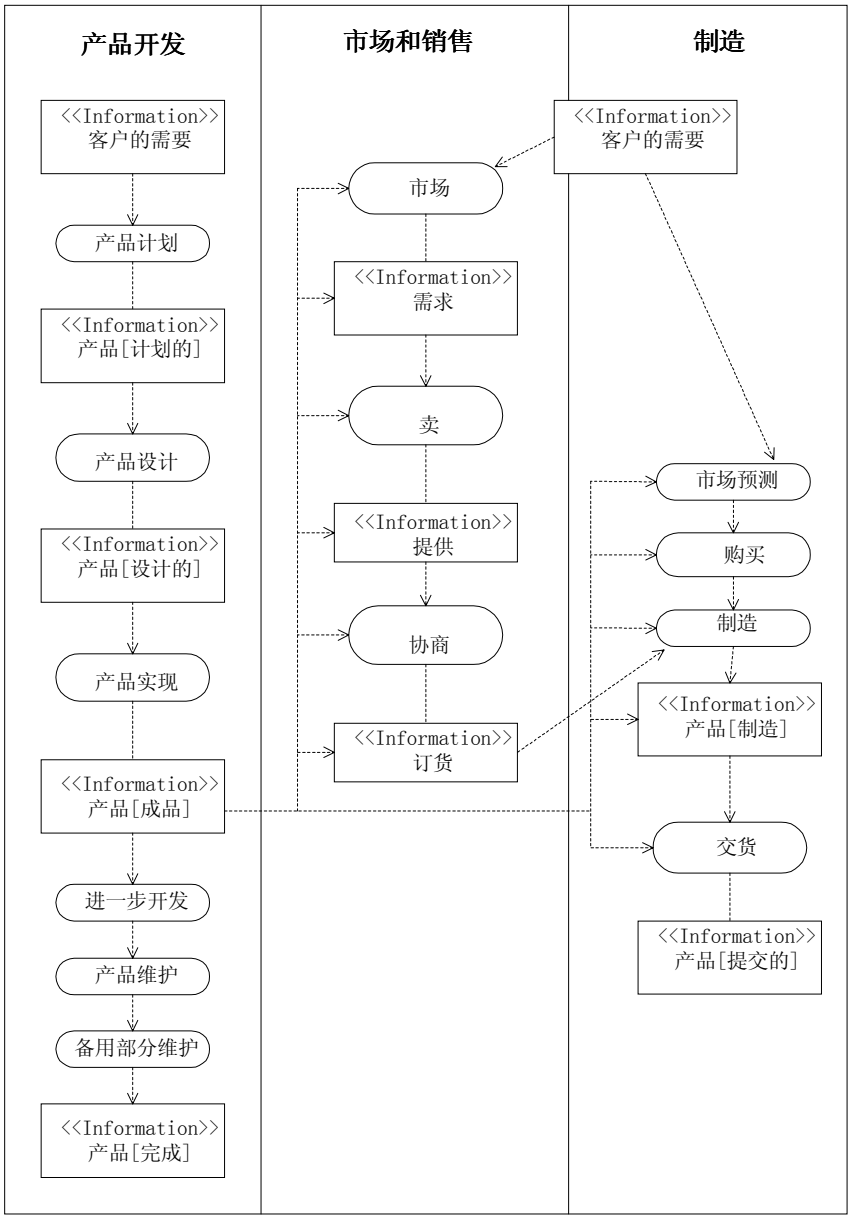

图 5-36 用活动图描述的制造业的商业模式

商务的目的推进工作流，在工作流中，根据指定的规则使用资源。在商业建模中，有一点非常重要，即将物理对象和信息对象分开。物理对象是哪些存在于现实世界中的对象(或我们对世界的理解)，例如，汽车、客户和合同。信息对象承载与工作或物理对象有关的信息。因而在一个信息系统中的客户对象不是一个真正的客户，它们包含的是有关真正客户的信息。版类可以用来将活动图中的物理对象和信息对象区别开。例如，版类《information》可能是用来表示对象是一个信息对象。同样，版类《Physical》可能是用来表示对象代表真正的物理对象。当对象被处理，产生或消费时，对象的状态发生改变，可以用活动图来描述这些改变(参考图 5-36)。

所有的商务都有组织，有时可能对模型有兴趣。在活动图中，泳道被用来表示组织。工人被作为资源，还可能被当作对象，虽然它们常被当作角色。角色是商务活动中的系统(人也可以作为系统)。通常情况下，工人是被雇用的、完成工作的人。角色推进工作流，被信息系统或其它的系统所支持。可以用 UML 中的曲棍球手图标来表示或用版类《actor》来表示，如图 5-36 所示。

用活动图进行商业建模可以用用例或其它的抽象系统需求的技术来补充。注意，采用这种建模途径，对象可以标识且被分成信息和物理对象。被标识的信息对象可能成为分析和建立支持模型化的商业系统的基础。使用活动图与实现工作流技术，如 IDEF0(标准的可视化建模语言)，是相似的。因此，动作可以用这些技术来描述。

可用很多方法来描述动作。一种可行的描述动作的途径如下所示：

- 定义：动作的形式化或非形式化描述。
- 目的：描述动作的目的。
- 特点：典型的可重复的或以前的翻版(one-time shot)。
- 测量方法：当需要测量动作且技术可行时，描述测量的方法。
- 角色：要求哪一个角色来执行动作。
- 信息技术：需要何种类型的信息系统的支持。
- 规则、政策、策略：所有的约束动作性能的文档、策略或政策均应提到。

活动图主要用来描述如何完成工作以及做什么工作。可以用活动图来描述操作、类或用例，但是它们只能显示工作流。可以用活动图来进行商业建模，在模型中，工作、工人、组织、对象被显示。

5.7 小　结

所有的系统均有静态结构和动态行为。结构可以用静态模型元素来描述，如类、关系、节点和构件。行为描述结构内的元素如何交互。通常情况下，这些交互是确定的且可以建立模型。抽象系统的动态行为也称为动态建模，UML 支持动态建模。在 UML 中有四类图，每一类用于不同的目的：状态、序列、协作和活动。

状态图被用来描述类(也可以用于子系统或整个系统)中的行为和内部状态。它着眼于描述随着时间的改变，对象如何改变其状态。状态的改变起决于出现的事件，状态中执行的行为和动作，状态转移等。事件可能是条件成真，接收一个信号或一个操作调用或经过

指定时间。

序列图主要用来描述在指定情节中一组对象是如何交互的。它着眼于消息序列，也就是说，在对象间如何发送和接收消息。序列图有两个坐标轴：纵坐标轴显示时间，横坐标轴显示有关的对象。序列图中最基本的东西是时间。

协作图主要用来描述对象在空间中的交互，即除了动态交互，它也直接描述对象是如何链接在一起的。在协作图中没有时间轴，因而将消息按序编号。