第19章
活　动　图

本章内容：

- 对一个工作流建模
- 对一个操作建模
- 正向工程和逆向工程

活动图是UML中用于对系统的动态方面建模的5种图中的一种图。一张活动图从本质上说是一个流程图，显示从活动到活动的控制流。【顺序图、协作图、状态图和用况图也可对系统的动态方面建模；在第18章中讨论顺序图和协作图；在第24章中讨论状态图；在第17章中讨论用况图；在第15章中讨论动作。】

活动图用于对一个系统的动态方面建模。多数情况下，它包括对计算过程中顺序的（也可能是并发的）步骤进行建模。当对象在控制流的不同点上从状态到状态移动时，用活动图也可对该对象的控制流进行建模。活动图可以单独用来可视化、详述、构造和文档化对象群体的动态特性，也可以用于对一个操作的控制流建模。交互图强调的是从对象到对象的控制流，而活动图强调的是从活动到活动的控制流。一个活动是一个状态机中进行的非原子的执行单元。活动最终导致一些动作，这些动作由可执行的原子计算组成，这些计算会导致系统状态的改变或一个值的返回。

活动图不仅对系统的动态特性建模是重要的，而且对于通过正向和逆向工程构造可执行的系统也很重要。

19.1　入门

考虑与建房子有关的工作流。首先，你选择一个地址。然后，委托一个建筑师对房子进行设计。当你确定计划后，开发商对房子进行投标竞价。一旦你同意了一种价格和设计计划，就可以开始建造房子。接下去获取执照、破土动工、挖地基、搭建框架等，直到建成房子。最后，你手里拿到房门钥匙和居住权证书，你就拥有了自己的房子。

尽管在整个建造过程中实际发生的琐碎事情还有许多，但是，它还是捕获住了工作流中的关键路径。在一个真实的工程中，多种事务之间存在着许多并行的活动。例如，电工活与管工和木工的活同时进行。工作流中还会有条件和分支。例如，你可能要根据土壤测试结果进行爆破、挖掘或注水。工作流中甚至还会有循环。例如，对一个建筑物的验收，可能发现不符合标准，结果是拆毁和重新施工。

在建筑工业领域，像甘特图（Gantt chart）和波特图（Pert chart）这样的技术被广泛用于可视化、详述、构造和文档化项目工程的工作流。

在对软件密集系统建模时，存在着类似的问题。如何最好地模拟系统的动态方面的工作流

和操作？回答是你有两个与使用甘特图和波特图类似的基本选择。【在第二部分和第三部分中讨论对系统的结构建模；在第18章中讨论交互图。】

一种选择是建立脚本的故事板，其中包括由某些感兴趣的对象以及它们之间传递的消息构成的交互。在UML中，有两种方法对这些故事板建模：强调消息的时间顺序（使用顺序图）或强调参加交互的对象间的结构关系（使用协作图）。交互图与甘特图类似，着眼于强调按时间完成某些动作的对象（资源）。

另一种选择是用活动图对这些动态方面建模，它首先关注于对象间发生的活动，如图 19-1所示。从这方面看，活动图与波特图类似。活动图本质上是流程图，它强调随着时间发生的活动。你可以把活动图看作翻新花样的交互图。交互图观察的是传送消息的对象；而活动图观察的是对象之间传送的操作。两者在语义上的区别是细微的，但看待世界的方式却是非常不同的。【在第15章中讨论动作。】

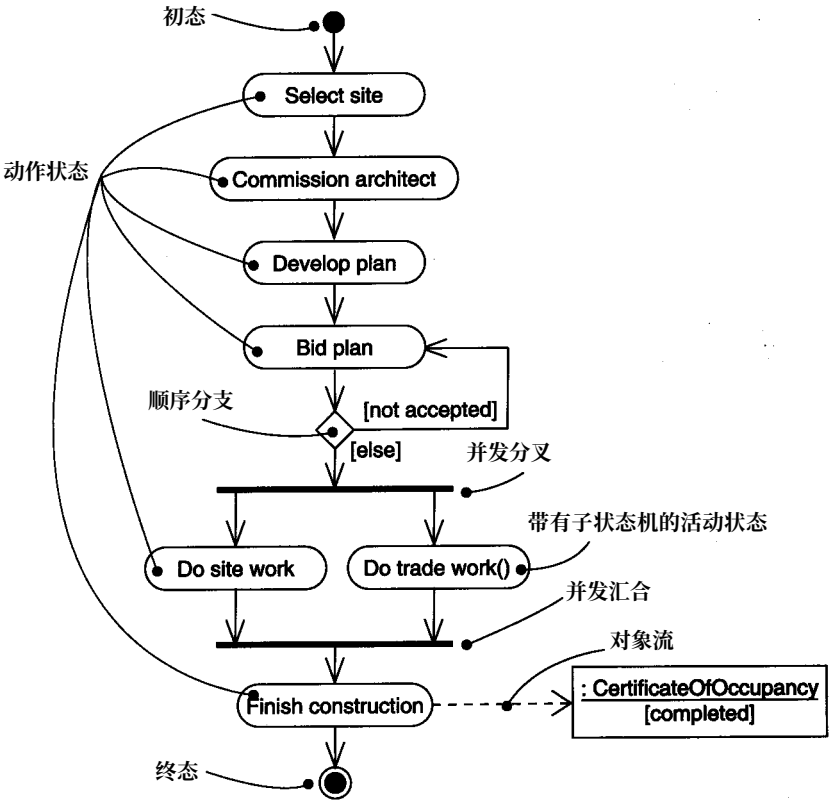

图19-1 活动图

19.2 术语和概念

活动图（*activity diagram*）显示从活动到活动的流。一个活动是一个状态机中进行的非原子

的执行单元。活动最终导致一些动作，这些动作由可执行的原子计算组成，这些计算会导致系统状态的改变或一个值的返回。动作包括调用另一个操作，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。在图形上，活动图是顶点和弧的集合。

1. 公共特性

活动图只是一种特殊的图，它具有与所有其他图一样的公共特征——即一个名称以及投影到一个模型上的图形内容。活动图与所有其他图不同的是它的特殊内容。【在第7章中讨论图的一般属性。】

2. 内容

活动图一般包括：

- 活动状态和动作状态
- 转换
- 对象

【在第21章中讨论状态、转换和状态机；在第13章中讨论对象。】

注释 活动图基本上是一个活动图形中发现的元素的一个投影，它是状态机的一种特殊情况。其中全部或大多数状态是活动状态，并且全部或大多数转换是通过源状态中活动的完成来触发的。因为活动图是一种状态机，所以它可以适应状态机的全部特性。这就意味着活动图可以包括简单的和组合的状态、分支、分叉和汇合。

像所有其他图一样，活动图可以包括注解和约束。

3. 动作状态和活动状态

用活动图建模的控制流中，会发生一些事情。你可能要对一个设置属性值或返回一些值的表达式赋值。你也可能要调用对象上的操作，发送一个信号给对象，甚至创建或撤销对象。这些可执行的原子计算被称作动作状态，因为它们是该系统的状态，每个原子计算都代表一个动作的执行。如图19-2所示，用一个菱形（一个顶部和底部为水平线，两边凸出的符号）来代表一个动作状态。在这个图符内部可以写任意表达式。【在第4章和第9章中讨论属性和操作；在第20章中讨论信号；在第15章中讨论对象的创建和撤销；在第21章中讨论状态和状态机。】

注释 UML并不规定这些表达式所使用的语言。较抽象地，可以仅使用结构化的文字描述；较具体地，可以使用特定的编程语言的句法和语义。

动作状态不能被分解。而且，动作状态是原子的，也就是说事件可以发生，但动作状态的工作不能被中断。最后，动作状态的工作所占用的执行时间一般被看作是可忽略的。

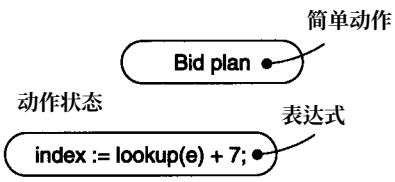

图19-2 动作状态

注释 当然，在现实世界中，每个计算要占用一定数量的时间和空间。尤其是对于那些注重于实时的系统，对这些特性建模就更加重要。【在第23章中讨论对时间和空间建模。】

相比之下，活动状态能够进一步被分解，它们的活动由其他的活动图表示。而且，活动状态不是原子的，也就是说它们可以被中断，并且，一般来说，还要考虑到它需要花费一段时间来完成。可以把一个动作状态看作一个活动状态的特例。一个动作状态是不能被进一步分解的活动状态。类似地，可以把一个活动状态看作一个组合，它的控制流由其他的活动状态和动作状态组成。放大一个活动状态的细节，你就会发现另一个活动图。如图 19-3所示，在活动状态和动作状态之间没有表示法的差别，只是活动状态可以有附加的部分，如进入和退出动作（分别指进入和离开该状态的动作）和子状态机说明。【在第21章中讨论状态机、状态的部分（包括进入和退出动作）和子状态机。】

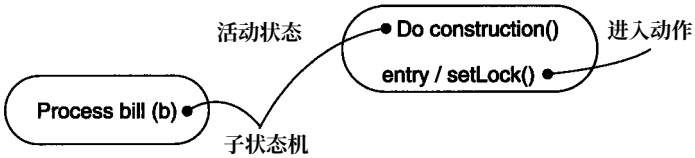

图19-3 活动状态

注释 动作状态和活动状态仅仅是状态机中状态的特殊种类。当你进入一个动作或活动状态时，你是简单地执行该动作或活动；当你结束一个动作或活动状态时，控制权就传递给下一个动作或活动。所以，活动状态有点像速记。一个活动状态在语义上等同于在适当的地方展开它的活动图（并且如此传递下去），直到图中仅看到动作为止。然而，活动状态是很重要的，因为它们帮助你将复杂计算分解成一些部分，就像使用操作来组织和复用表达式一样。

4. 转换

当一个状态的动作或活动结束时，控制流会马上传递给下一个动作或活动状态。你可以用转换来说明这个流，显示从一个动作或活动状态到下一个动作或活动状态的路径。如图 19-4所示，UML中用一条简单的有向直线表示一个转换。【在第21章中讨论转换。】

注释 从语义上讲，这些转换被称作无触发转换或完成转换，这是因为一旦源状态的工作被完成，控制会立即传送。一旦一个给定的源状态的动作完成，就执行该状态的退出动作（如果有）。接着，控制就不加延迟地跟着转换传送到下一个动作或活动状态。你执行那个状态的进入动作（如果有），那就是执行目标状态的动作或活动，一旦此状态的工作完成，就又跟着进行下一个转换的传送。这个控制流会不停地继续下去（在无穷的活动情况下）或者直到遇到一个停止状态才停止。【无触发转换可以有监护条件，表明该转换只有当条件满足时才触发；在第21章中讨论监护条件。】

事实上，一个控制流都会从某个地方开始和到某个地方结束（除非它是一个只有开始没有结束的无穷的流）。所以，如图所示，要说明初始状态（一个实心球）和停止状态（一个圆圈内

有一个实心球)。

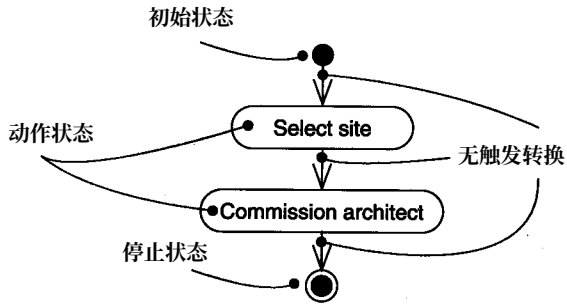

图19-4 无触发转换

5. 分支

简单地、顺序的转换是常见的，但并不是对一个控制流建模所需的唯一的一种途径。像在一个流程图中，可以包含一个分支，它描述了基于某个布尔表达式的可选择的路径。如图 19-5所示，用一个钻石形来表示分支。一个分支可以有一个进入转换和两个或多个离去转换。在每个离去转换上放置一个布尔表达式，它只在进入这个分支时被判断一次。在所有这些离去转换中，其监护条件不应该重叠（否则，控制流将是模糊的），但是它们应该覆盖所有的可能性（否则，控制流可能会冻结）。【分支在表示法、方便性和语义上与通过监护进行多种转换是等价的，这在第 21章中讨论。】

为了方便，可以使用关键字 `else`来标记一个离去转换，它表示如果其他的监护表达式都不为真时所执行的路径。

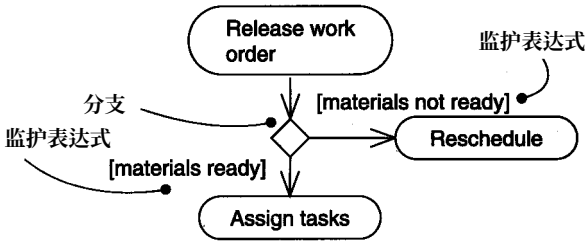

图19-5 分支

为了获得迭代的效果，可以用一个动作状态设置迭代器的值，用另一个动作状态增加该迭代器的值，并用一个分支来判断该迭代是否结束。【在交互图中可能会有分支和迭代，这在第 18章中讨论。】

注释 UML并不规定这些表达式所使用的语言。较抽象地，可以仅使用结构化的文字描述；较具体地，可以使用一种特定的编程语言的语法和语义。

6. 分叉和汇合

简单的和具有分支的顺序转换是活动图中最常见的路径。然而，尤其当你对业务过程的工作流建模时，可能会遇到并发流。在 UML中，用一个同步棒来说明这些并行控制流的分叉和汇

合。一个同步棒是一条粗的水平或垂直的线。【每个并发控制流存在于一个独立的主动对象的语境中，通常被建模为一个进程或线程，这在第22章中讨论；在第26章中讨论节点。】

例如，在模拟人讲话和作手势的听觉模拟装置中，考虑它的并发控制流。如图19-6所示，一个分叉表示把一个单独的控制流分成两个或更多的控制流。一个分叉可以有一个进入转换和两个或更多的离去转换，每一个转换表示一个独立的控制流。在这个分叉之下，每一个路径相关的活动将并行地继续。从概念上说，这些流中的每一个流的活动都是真实的并发，尽管在一个运行系统中，这些流既可以是真实的并发（在系统被装配在多个节点上的情况下），也可以是顺序但又交替的（在系统只装配在一个节点上的情况下），因此只给出真实并发的图示。

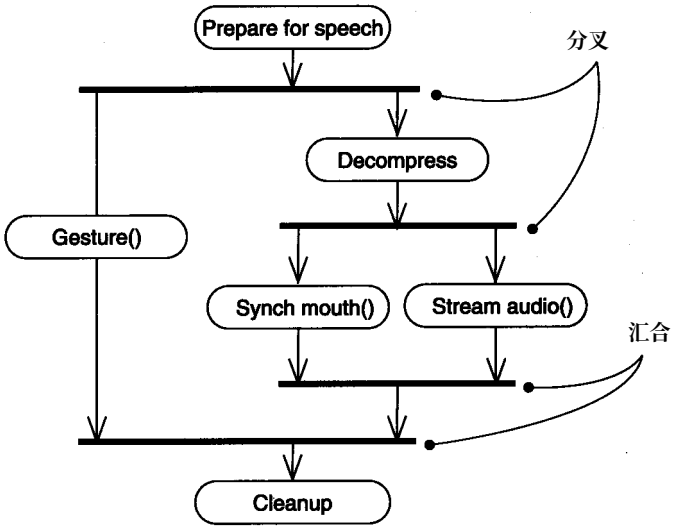

图19-6 分叉与汇合

像图中还显示的那样，一个汇合表示两个或更多的并发控制流的同步发生。一个汇合可以有两个或多个进入转换和一个离去转换。在该汇合的上面，与每一个路径相关联的活动并行地执行。在汇合处，并发的流取得同步，这意味着每个流都等待着，直到所有进入流都到达这个汇合处，然后，在这个汇合的下面，只有一个控制流从这一点继续执行。

注释 汇合和分叉应该是平衡的，即离开一个分叉的流的数目应该和进入它的对应的汇合流的数目相匹配。位于并行控制流中的活动也可能通过发送信号而相互通信。这种风格的通信顺序进程被称作协同例程。多数情况下，用主动对象来对这种通信方式建模。也可以对这些信号的发送和响应建模，这些信号存在于与每个通信活动状态相关的子状态机里。例如，假设活动Stream audio需要告诉活动Synch mouth何时出现重要的停顿和语调。在Stream audio的状态机中，你将看到发送给状态机Synch mouch的信号。类似地，在Synch mouch的状态机中，你将看到被这些相同信号触发的转换，Synch mouch状态机将对这些信号做出响应。【在第22章中讨论主动对象；在第20章中讨论信号。】

7. 泳道

泳道将一个活动图中的活动状态分组，每一组表示负责那些活动的业务组织；尤其当你对业务过程的工作流建模时，泳道是很有用的。在 UML中，每个组被称为一个泳道，因为从视觉上，每组用一条垂直的实线把它与邻居分开，如图 19-7所示。一个泳道说明一个活动轨迹。

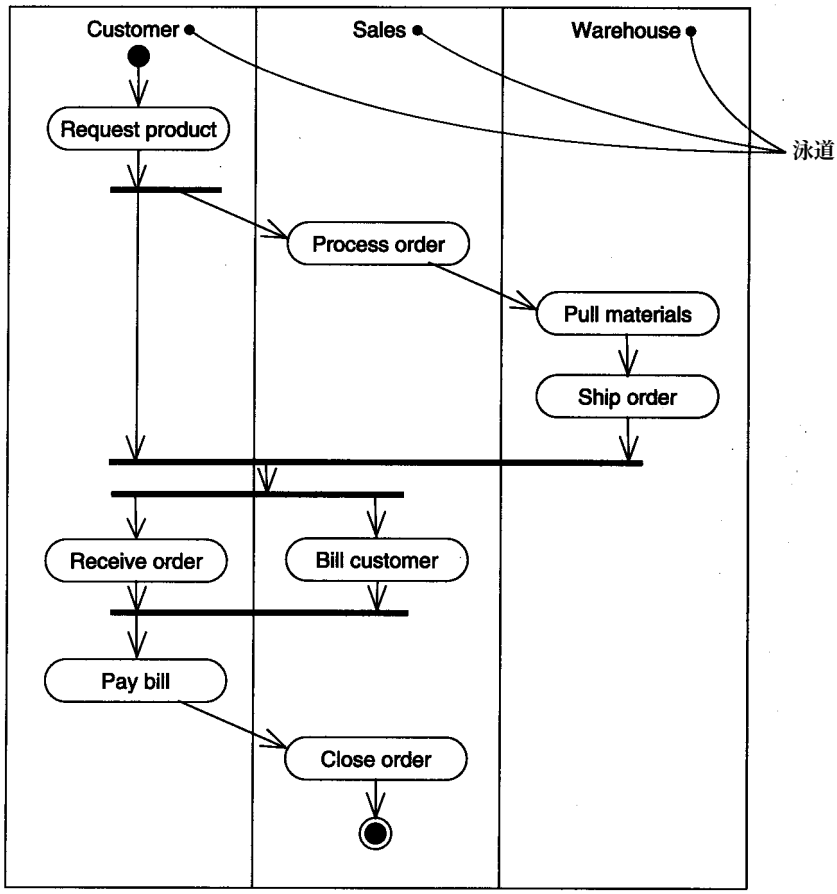

图19-7 泳道

每个泳道在图中都有一个唯一的名称。泳道除了可能代表现实世界的某些实体之外，实际上没有很深的语义。每个泳道代表一个活动图的全部活动中部分活动的高层职责，并且每个泳道最终可能由一个或多个类实施。在一个被划分为泳道的活动图中，每个活动都明确地属于一个泳道，而转换可以跨越泳道。【泳道是包的一种；在第12章中讨论包；在第4章和第9章中讨论类；在第22章中讨论进程和线程。】

注释 在泳道和并发的控制流之间有一个松散的连接。从概念上讲，每个泳道的活动一般——但不总是——被看作是与其邻近泳道的活动分开的。这非常有意义，因为在现实世界中，一般映射到这些泳道上的业务组织是独立的和并发的。

8. 对象流

对象可以被包含在与一个活动图相关的控制流中。例如，在上图处理订购的工作流中，问题空间的词汇中也将包括像Order（订购）和Bill（付帐）这样的类。这两个类的实例将由一定的活动生成（例如Process order将创建一个Order对象）；其他的活动可能修改这些对象（例如Ship Order将把Order对象的状态变为filled）。【在第13章中讨论对象；在第4章中讨论对系统的词汇建模。】

如图19-8所示，描述一个活动图中所涉及的事物，可以通过把这些对象放置在活动图中，并用一个依赖将它们连接到对它们进行创建、撤销或修改的活动转换上。这种对依赖关系和对象的应用被称作是一个对象流，因为它描述了一个对象在一个控制流中的参与情况。【在第5章和第10章中讨论依赖关系。】

活动图除了可以显示一个对象的流，也可以用来显示对象的角色、状态和属性值是如何改变的。就像图19-8中所示，在该对象名下面的方括号中命名它的状态。类似地，还可以在该对象名的下面加一个分隔栏表示对象的属性值。【在第13章中讨论对象的属性值和状态；在第4章和第9章中讨论属性。】

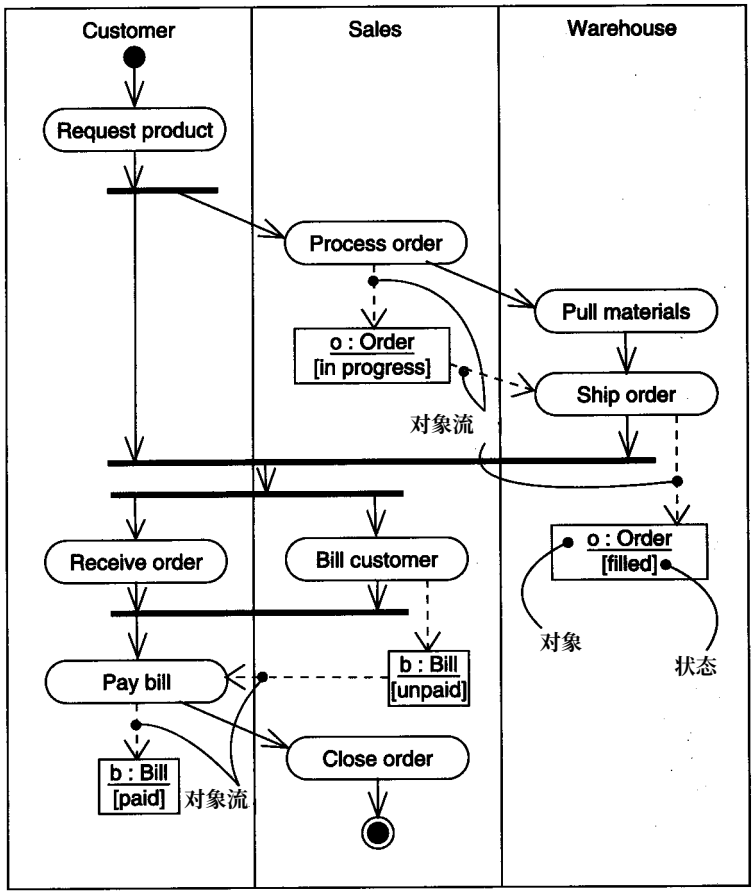

图19-8 对象流

9. 一般应用

活动图用于对系统的动态方面建模。这些动态方面可涉及一个系统体系结构的任意视图中任何类型抽象的活动，包括类（含主动类）、接口、构件和节点。【在第2章中讨论体系结构的5个视图；在第4章和第9章中讨论类；在第22章中讨论主动类；在第11章中讨论接口；在第25章中讨论构件；在第26章中讨论节点。】

当你使用活动图对系统的某些动态方面建模时，事实上你可以在任意建模元素的语境中这样做。你可以在整个系统、一个子系统、一个操作或一个类的语境中使用活动图。还可以把活动图附在用况（对脚本建模）和协作（对对象群体的动态方面建模）上。【在第31章中讨论系统和子系统；在第4章和第9章中讨论操作；在第16章中讨论用况和参与者。】

当你对一个系统的动态方面建模时，通常有两种使用活动图的方式。

1）对工作流建模

此时，你关注于与系统进行协作的参与者所观察到的活动。工作流常常位于软件系统的边缘，用于可视化、详述、构造和文档化开发系统所涉及的业务过程。在活动图的这种用法中，对对象流的建模是特别重要的。

2）对操作建模

此时，你把活动图作为流程图使用，对一个计算的细节部分建模。在活动图的这种用法中，对分支、分叉和汇合状态的建模是特别重要的。用于这种方式的活动图语境包括该操作的参数和它的局部对象。

19.3 普通建模技术

19.3.1 对工作流建模

没有任何一个软件系统是孤立存在的，一个系统总是存在于某些语境中，并且这些语境总是包含与该系统进行交互的参与者。尤其对紧要使命的企业软件，你将发现工作在较高层次的业务过程语境中的自动化系统。这些业务过程是一种工作流，因为它们代表工作的流程以及贯穿于业务之中的对象。例如，在一个零售业务系统中，有某些自动系统（例如与市场和仓库系统交互的销售点系统），还有人员系统（工作在各零售台、远程销售、市场、购买和货运部门的人员）。通过使用活动图，可以对业务过程中各种自动系统和人员系统的协作建模。【对第17章中讨论对系统的语境建模。】

对工作流建模，要遵循如下的策略：

- 为工作流建立一个焦点。除非很小的系统，否则不可能在一张图中显示所有感兴趣的工作流。
- 选择对全部工作流中的一部分有高层职责的业务对象。这些业务对象可以是系统的词汇中的真实事物，也可能较为抽象。无论哪种情况，为每个重要的业务对象建立一个泳道。【在第4章中讨论对系统的词汇建模；在第9章中讨论前置和后置条件。】

- 识别该工作流初始状态的前置条件和该工作流停止状态的后置条件。这对于帮助你对工作流的边界建模是重要的。
- 从该工作流的初态状态开始，说明随着时间发生的动作和活动，并在活动图中把它们表示成活动状态或动作状态。
- 将复杂的动作或多次出现的动作集合归并到一个活动状态，并对每个这样的活动状态提供一个可将它展开的单独的活动图。
- 找出连接这些活动和动作状态的转换。首先从工作流的顺序流开始，然后考虑分支，接着再考虑分叉和汇合。
- 如果工作流中涉及重要的对象，则也把它们加入到活动图中。如果对表达对象流的意图是必要的，则显示其变化的值和状态。

例如，图19-9显示了一个零售系统的活动图，它说明的是当一个顾客从邮件订单中返回一个项目时的工作流。工作从顾客对象Customer的动作Request return开始，然后这个流通过电话销售对象Telesales的Get return number，回到Customer的Ship item，然后到仓库对象Warehouse（先到Receiveitem，后到Restockitem），最后，以结帐对象Accounting的Credit accoun结束。如图所示，一个主要的对象（i，Item的一个实例）也在过程中流动，并且从returned状态变化到available状态。

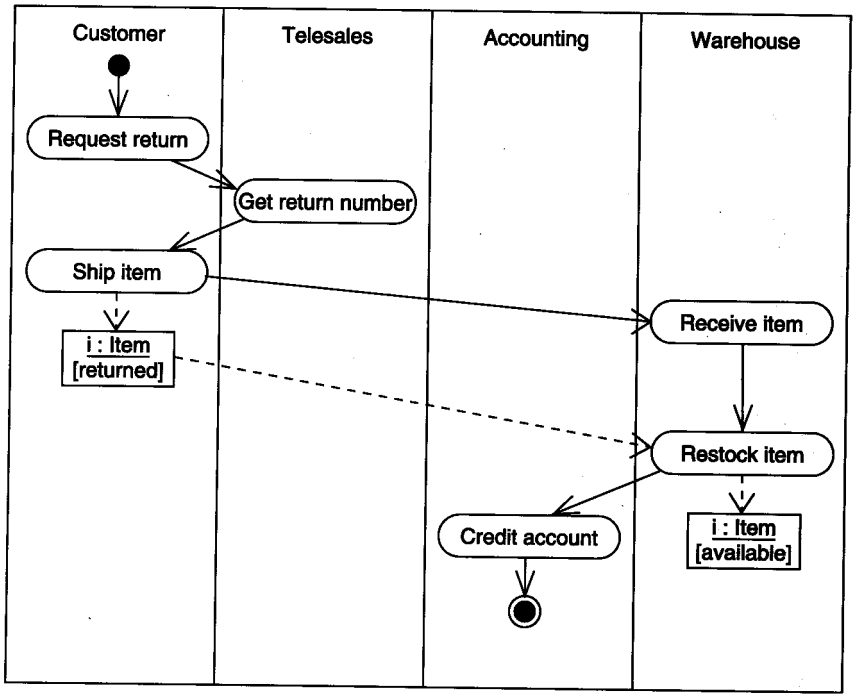

图19-9 对一个工作流建模

注释 工作流常常是业务过程，但也不总是。例如你也可以用活动图说明软件开发过程，如用于配置管理的过程。而且，你也可以用活动图描述非软件系统，如贯穿一个康复系

统的病人的流。

在这个例子中没有分支、分叉和汇合，你将在更复杂的工作流中遇到这些特征。

19.3.2 对操作建模

为了可视化、详述、构造和文档化元素的行为，可以将一个活动图附加到任意建模元素之上。你可以将活动图附加到类、接口、构件、节点、用况和协作上。其中，最常见的是向一个操作附加一个活动图。【在第4章和第9章中讨论类和操作；在第11章中讨论接口；在第25章中讨论构件；在第26章中讨论节点；在第16章中讨论用况；在第27章中讨论协作；在第9章中讨论前置条件和后置条件以及不变式。】

采用这种方式，活动图只是一个操作的动作的流程图。活动图的主要优点是图中所有的元素在语义上都与一个丰富的基本模型相连。例如，一个动作状态所引用的任何其他操作或信号都被目标对象的类进行类型检查。

对操作建模，要遵循如下的策略：

- 收集这个操作所涉及的抽象。包括操作的参数（含它的返回类型，如果有）、所属类的属性以及某些邻近的类。
- 识别该操作的初始状态的前置条件和停止状态的后置条件。也要识别在操作执行过程中必须保持的所属类的不变式。
- 从该操作的初始状态开始，说明随着时间发生的活动和动作，并在活动图中将它们表示为活动状态和动作状态。
- 如果需要，使用分支来说明条件路径和迭代。
- 仅当这个操作属于一个主动类时，才在必要时用分叉和汇合来说明并行的控制流。【在第22章中讨论主动类。】

例如，在类Line的语境中图19-10显示了一个描述操作intersection的算法的活动图，它的特征标记包含一个参数（l，类Line的in参数）和一个返回值（属于类Point）。类Line有两个属性：slope（线段斜率）和delta（线段相对原点的偏移）。

这个操作的算法是简单的，如下面活动图所示。首先，检测当前线段的斜率slope是否和参数l的slope相同。如果相同，线段不交叉，并返回一点Point（0，0）。否则，操作首先计算交叉点的x值，然后计算y值；x和y对于操作都是局部对象。最后，返回一个点Point（x，y）。【如果一个操作涉及对象之间的交互，你也可以用协作对该操作的实现建模，这在第27章中讨论。】

注释 用活动图对一个操作建立流程图，这几乎是将UML作为一种可视化编程语言来使用。你可以对每个操作都建立流程图，但实际中你并不想这样做。用一个特定的编程语言书写一个操作的内容通常更为直接。当操作的行为很复杂时，你会想到用活动图来表达，因为仅仅关注于代码将很难理解。流程图将向你展示关于算法的一些信息，而这些信息如果只看代码，是看不到的。

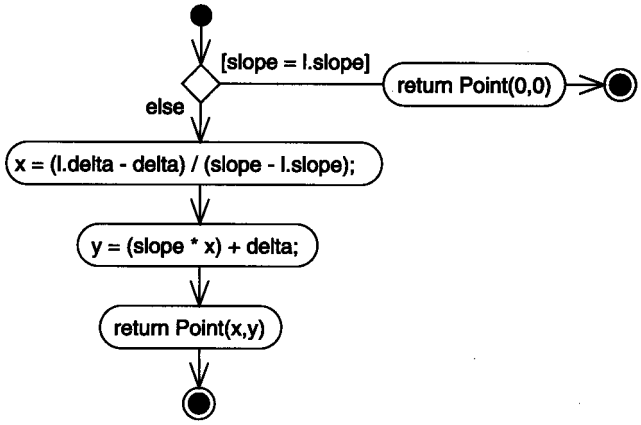

图19-10 对一个操作建模

19.3.3 正向工程和逆向工程

对于活动图可进行正向工程（从一个模型产生代码），尤其当图的语境是一个操作时就更有这种可能。例如，对于前一个活动图，一个正向工程工具可以产生以下关于 interaction操作的C++代码。

```
Point Line::intersection (l : Line) {
  if (slope == l.slope) return Point(0,0);
  int x = (l.delta - delta) / (slope - l.slope);
  int y = (slope * x) + delta;
  return Point(x, y);
}
```

这里有一点技巧，是有关两个局部变量的说明。一个不太复杂的工具可能会首先声明两个变量，然后对它们赋值。

对活动图也可以进行逆向工程（从代码产生一个模型），尤其当代码的语境是一个操作体时就更有这种可能。特别地，前一张图能够从类 Line的实现中产生出来。

比从代码到一个模型的逆向工程更有趣的是针对一个实施系统执行的模型模拟。例如，对于给定的上面的图，一个工具能够模拟图中的动作状态，就像在运行的系统中那样被调度。更好的是，在一个调试器的控制下使用这个工具，你能够控制执行的速度，并能在感兴趣的点上设置断点来停止动作，以检测单个对象的属性值。

19.4 提示和技巧

在UML中创建活动图时，要记住活动图只是一个系统的动态特性在同一模型上的投影。没有任何一个单独的活动图能捕捉关于系统动态特性的所有事情。相反，你将使用许多活动图对一个工作流或一个操作的动态方面建模。

一个结构良好的活动图，应满足如下的要求：

- 关注于与系统动态特性的一个方面的交流。
- 只包含那些对于理解这个方面必不可少的元素。
- 提供与它的抽象层次相一致的细节；只加入那些对于理解问题必不可少的修饰。
- 不应该过分简化和抽象信息，以致使读者误解重要的语义。

当绘制活动图时，要遵循如下的策略：

- 给出一个能反映其目的名称。
- 首先对主要的流建模。其次标出分支、并发和对象流，它们也可能被放在几张分开的图中。
- 摆放它的元素以尽量减少线的交叉。
- 使用注解和颜色作为可视化提示，以突出图形中重要的特征。