第9章 面向对象的分析阶段

如第1章所述，面向对象的范型是为解决结构化范型所存在的不足而引入的。许多利用结构化范型编制的小、中规模软件项目获得了极大的成功。然而，从80年代末开始，人们发现当把结构化范型应用于大型软件产品开发时，似乎很少取得成功。

9.1 面向对象范型与结构化范型的比较

可以从两个方面来考察每一个结构化的软件产品。一方面是仅考虑其中的数据，包括局部变量和全局变量，以及参数、动态数据结构、文件等。另一方面是仅仅考虑数据的操作，即过程和函数。使用这种把软件分成数据和操作的划分方法，则势必使所有结构化技术分属于这两个分组之一。面向操作的技术要考虑产品行为(操作)，数据是次要的，且仅在产品的行为被深入分析之后才予以考虑。相反，面向数据的技术则强调产品的数据，产品的行为仅仅被放在数据的框架内予以考察。

一个面向操作(行为)的结构化技术的例子是有穷状态机(8.6节)。这里强调的是行为；数据的重要性则要低得多。结构化系统分析(8.3节)是另外一个面向操作(行为)技术的例子。一个数据流图真实反映了数据和操作两方面。但是，数据流图的目的是为了展示产品将执行的所有操作行为，数据相对于操作来说是次要的。对于面向数据的结构化技术，在第11章中将给出这方面的一个详细情形，即Jackson系统的开发。在这一章的例子中，首先是确定数据结构，然后根据数据结构决定数据之上的行为结构。换句话说，行为结构遵从数据结构。另一个面向数据结构技术的例子是在8.5节中描述的实体关系模型。

面向数据和面向行为这两种方法的一个根本的缺陷是，把数据和操作分成一个硬币的两面。毕竟，离开了操作，数据便无法更改，而脱离了数据的操作则毫无意义。因此，我们需要对数据和操作予以同样的重视。面向对象的技术能做到这一点，所以它的出现是不足为奇的。毕竟，一个对象包含了数据和操作这两部分。回忆第6章便可知道，对象是抽象数据类型(更为确切地说是一个类)的一个实例，对象把数据和作用于这些数据的行为融合在一起。一个对象的数据有许多不同的名称，例如属性、状态变量、实例变量、字段或数据成员，而其操作被称为方法或成员函数。但是，不管数据和操作的叫法如何，两者都以平等的伙伴关系出现在对象里。同样，在所有的面向对象技术里，数据和操作被认为是同等重要的，而无优先次序之分。

在面向对象范型的技术里，认为数据和操作是同时考虑的这种说法是不恰当的。从4.1节逐步求精的资料中我们可以清楚地看到，有时候必须重点考虑数据，而有时候操作更为关键。然而，总的来说，在面向对象的范型的各阶段，数据和操作是同等重要的。

在第1章和第6章里，我们讲述了许多为什么面向对象的范型要比面向结构的范型优越的理由。隐藏在所有那些理由中的事实是，一个好的设计对象应具有较高的内聚性和较低的耦合性，应是一个包含对应物理实体的各个方面的模型。有关这种实现的所有细节均被隐藏在对象里，对象间仅有的通信是通过消息传递完成的。因此，对象本质上是具有完善的接口定义的一个独立的单元。所以它们便于安全地维护；且使产生回归错误的机会降低。此外，对象是可重用的，这种可重用性通过对属性的继承得以增强。现在转而讨论面向对象的开发，那些建造好了的基本软件模块比起结构化模块更能安全地组合成一个大型的软件产品。因为对象本质上是产品的

一个独立的部分，因此产品开发及对开发的管理更加容易，从而出错的可能性也更小。

面向对象范型的所有这些优越性给人们提出了这样一个问题：既然结构化范型比起面向对象范型来说是如此拙劣，为什么结构化范型却取得如此多的成功？ 对这一问题的解释是，结构化范型是人们在软件工程尚未普及时采用的；那时侯人们只是“写”软件。对管理人员来说，程序员最重要的任务就是编写代码行。软件的需求说明和产品的规格说明(系统分析)只是口头说说，设计几乎是从来不做的事情。边做边改模型(3.1节)是70年代技术的典型代表。因此，结构化范型的使用使大多数的软件开发者第一次有了一种系统的技术。由此，结构化技术给全世界范围内的软件产业带来了一场革命，因此，它的成功就显得不足为奇了。然而，随着软件规模的增大，结构化范型的不足之处便开始逐渐暴露出来，于是面向对象的范型作为一种更好的技术选择便应运而生。

反过来，这又引起了另外一个问题：我们怎么能知道面向对象的范型又一定优于其他现有的技术呢？目前还没有取得任何数据可以确定无疑地证明面向对象技术是最好的，而且，目前还难以想象如何获得这些数据。我们能得到的最好证据就是那些已经采用面向对象范型的组织的经验。虽然不是所有的报告均是有利的证据，但是，大多数(尽管不是绝大多数)报告都证明使用面向对象范型是一种明智的决策。

例如，IBM在三个完全不同的工程中运用了面向对象技术，并做了总结报告[Capper, Colgate, Hunter, and Jamse,1994]。几乎在每一方面，面向对象范型的性能均大大超过结构化范型。尤其是大幅度减少了软件中发现的错误数目，极大地减少了在开发和维护阶段修改需求(除了由一些不可预见的商业变化所引起的需求)，软件的适应性和完善维护性都显著地得以提高。同时，尽管它不像前面的四个改变那么显著，且在性能上没有什么较大的提高。但其可用性却得到改善。

最后一个问题是：有没有可能在未来会出现比面向对象范型更好的技术呢？甚至连最强有力的支持者也不会声称面向对象范型是所有软件工程问题的终级解答。而且，今天的软件工程正在寻求对象以外的技术，以取得下一步的重大突破。毕竟在人类为之奋斗的领域中很少有过去的技术比现今正在推出的技术更为先进的情况。今天的技术(像面向对象范型)很有可能被将来的技术所取代。重要的是就当今来说，面向对象技术看上去是最好的。

9.2 面向对象的分析

在面向对象的范型中，面向对象的分析(OOA)是一种半形式化的规格说明技术。在第8章曾经指出，结构化系统分析有许多不同的技术，其本质是相同的。同样，目前有40多种不同的技术用于面向对象的分析，而且新的技术还在不断的被提出。同样，所有的技术大部分相差无几。本章“进一步阅读”部分包括众多供参考的不同技术，并对不同的技术做了比较。

目前，最流行的技术是OMT[Rumbaugh et al., 1991]和Booch [Booch,1994] 开发的技术。这里介绍的技术是基于OMT的，但是，也带一定的概括性，以便能抓住大多数OOA技术的本质特性。

OOA由三个步骤组成：

1. 类建模

决定类及其属性，然后决定类间的相互关系。这一信息以流图的形式给出，它类似于一个实体关系图(8.5节)，称为类模型(有的作者也把这种图称为对象模型)。该步骤是纯面向数据的。

2. 动态建模

决定对每一个类或子类的操作。这一信息以类似于有穷状态机(8.6节)的流图形式给出，术语称为动态模型。该步骤是纯面向操作的。

3. 功能建模

决定产品如何计算各种结果(不考虑次序)。这一信息以类似于数据流图(8.3节)的形式给出，术语称为功能模型。该步骤大部分是面向操作的。

在实践中，以上三个步骤并非纯粹地按顺序执行，但总是从步骤1开始，因为整个开发过程都基于在这一步中所定义的类。而在后继的步骤里，规格说明组成员经常不得不修改类模型。反过来这些改变又引起了动态模型和功能模型的变化。换句话说，OOA的三个步骤是高效地并行发生的。于是我们就能理解为什么在面向对象的范型中，数据(步骤1)和操作(步骤2、3)是不分优先次序的。

一定不要把OOA视作前面章节中描述的三种结构说明技术的组合，即实体关系模型、有穷状态机和结构化系统分析的组合。相反，OOA是一种利用图表来说明三个建模步骤结果的建模技术。它不是通过发明一种全新的方法来表示这些信息，而是运用广泛使用的结构化技术中的概念。

观察这一问题的另一种方法是分析OOA的目标。正如所有其他的规格说明技术一样，OOA的目标也是明确最终要建立什么样的产品。目标产品的两个关键点是它的数据和操作。为了理解数据、操作以及数据与操作间的相互作用，OOA使用了三种不同的模型。在分析的过程中，所获得的有关产品的知识将用三种不同的方法表示，每一种方法反映目标产品的不同方面。当对系统模型形成一个更全面的理解时，流图便需要随之更新。在OOA的结束阶段，这三种观点合在一起反映了对产品的全面理解。如果仅仅采用其中一种建模技术，那么这样的理解将很难得到。

9.3　电梯问题：面向对象的分析

下面通过一个例子来描述OOA的步骤，即第8章开始时描述的电梯问题。为了便于参照在此重新提出该问题。

在一幢有m层的大厦中安装一套控制n部电梯的产品，我们所关心的是依照下列条件约束求解电梯在各楼层之间移动的逻辑关系：

C_1：每台电梯有m个按钮，每一按钮代表一个楼层。当按下一个按钮时该按钮指示灯亮，同时电梯驶向相应的楼层，当到达按钮按下的相应楼层时指示灯熄灭。

C_2：除了最低层和最高层之外，每一层楼都有两个按钮分别指示电梯上行和下行。这两个按钮按下时指示灯亮，当电梯到达此楼层时灯熄灭，并向所需要的方向移动。

C_3：当电梯无升降动作时，关门并停在当前楼层。

OOA的第一步是构造对象模型，更确切地说是类模型。

9.3.1　类模型

在这一步里，类和它的属性被抽象出来，并以一种实体关系图来加以描述。此时只定义了类的属性而没有定义方法；后者在面向对象的设计(OOD)阶段被嵌入类中。

整个面向对象范型的共同特点是各个步骤都很难实施，幸运的是，使用对象带来的好处使这种努力是值得的。在第一次类建模的初始阶段(即类和属性的抽象阶段)，难于得到正确的结果是不足为奇的。一种较好的方法是依照以下三个过程以产生候选类，并对其加以精化。

阶段1：精确的问题定义。

尽可能简洁地定义产品，最好以一个句子予以描述。在电梯问题中，符合这一点的一种可能的描述是：

在一个m层楼的大厦里，用每层楼的按钮和电梯内的按钮来控制n部电梯的移动。

阶段2：非形式化策略。

为提出一种解决以上问题的非形式化策略，必须确定问题的约束条件。在前一小节的结尾，对电梯问题提出了三种约束。非形式化策略最好能以一小段文本表达清楚。对电梯问题一种可能的非形式化策略表达是：

在一栋m层楼的大厦里，用电梯内的和每层楼的按钮来控制n部电梯的运动。当按下电梯按钮以请求在某一指定楼层停下时，按钮指示灯亮灯指示；到达指定楼层时，按钮指示灯灭。当电梯无升降操作时，关门，并停在当前楼层。

阶段3：策略的形式化。

定义非形式化策略中的名词(不包括那些问题之外的名词)并把这些名词作为候选类。现在，重新生成非形式化策略，这时名词用黑体来标识：

在一栋m层楼的**大厦**里，用**电梯**内的和每个**楼层**的**按钮**来控制n部**电梯**的**运动**。当按下**电梯按钮**以**请求**在某一指定**楼层**停下时，**按钮指示灯**亮；当**请求**获得满足时，**指示灯**熄灭。当**电梯**无升降操作时，关门，并停在当前**楼层**。

在以上这段里共有八个不同的名词，即按钮(Button)、电梯(Elevator)、楼层(floor)、运动(movement)、大厦(building)、指示灯(illumination)、请求(request)和门(door)。其中有三个名词：楼层、大厦和门是处于问题的边界之外的，因此可被忽略。其他三个的名词：运动、指示灯和请求是抽象名词，也就是说它们“标识的不是物理存在的思想和数量”[World Book Encyclopedia，1992]。一个有用的经验准则是，抽象名词很少最终对应于一个类，而经常作为类的属性。例如，指示灯可作为按钮的属性，其他抽象名词(例如请求)证明与问题描述无关。这样就只剩下了两个候选类，即电梯和按钮(Elevator 和 Button)。

该问题指定了两种类型的按钮，故按钮的两个子类将被定义，即电梯按钮和楼层按钮。最终的类模型如图9-1所示。图中有两个基本类，即电梯(其属性为开门)和按钮(其属性为指示灯)。按钮含有电梯按钮和楼层按钮两个子类，图中的三角形表示继承关系。

这不是一个良好的开端。在真正的电梯中，按钮并不直接与电梯通信；要想决定分派哪一部电梯来响应一个特别请求，必须有某种类型的电梯控制器。然而陈述问题时，并未提及控制器，故它未被选入类中。换句话说，寻找候选类的技术提供了一个起点，但不能依靠它做更多的事情。

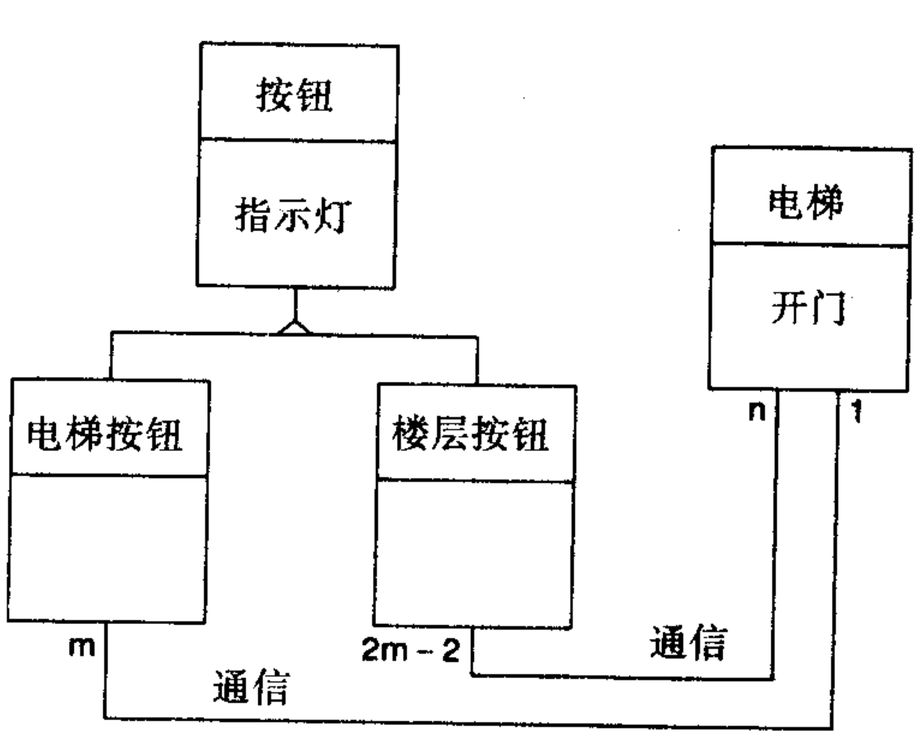

图9-1 类模型的第一次迭代

加入了电梯控制器后便形成了图9-2，这个图必然有更多的含义。此外，由于现在所有的关系均为一对多关系，因此，这可以使设计和实现更容易。此时似乎应该可以进入第2步，但必须牢记，任何时候均可返回到类建模步骤上去。

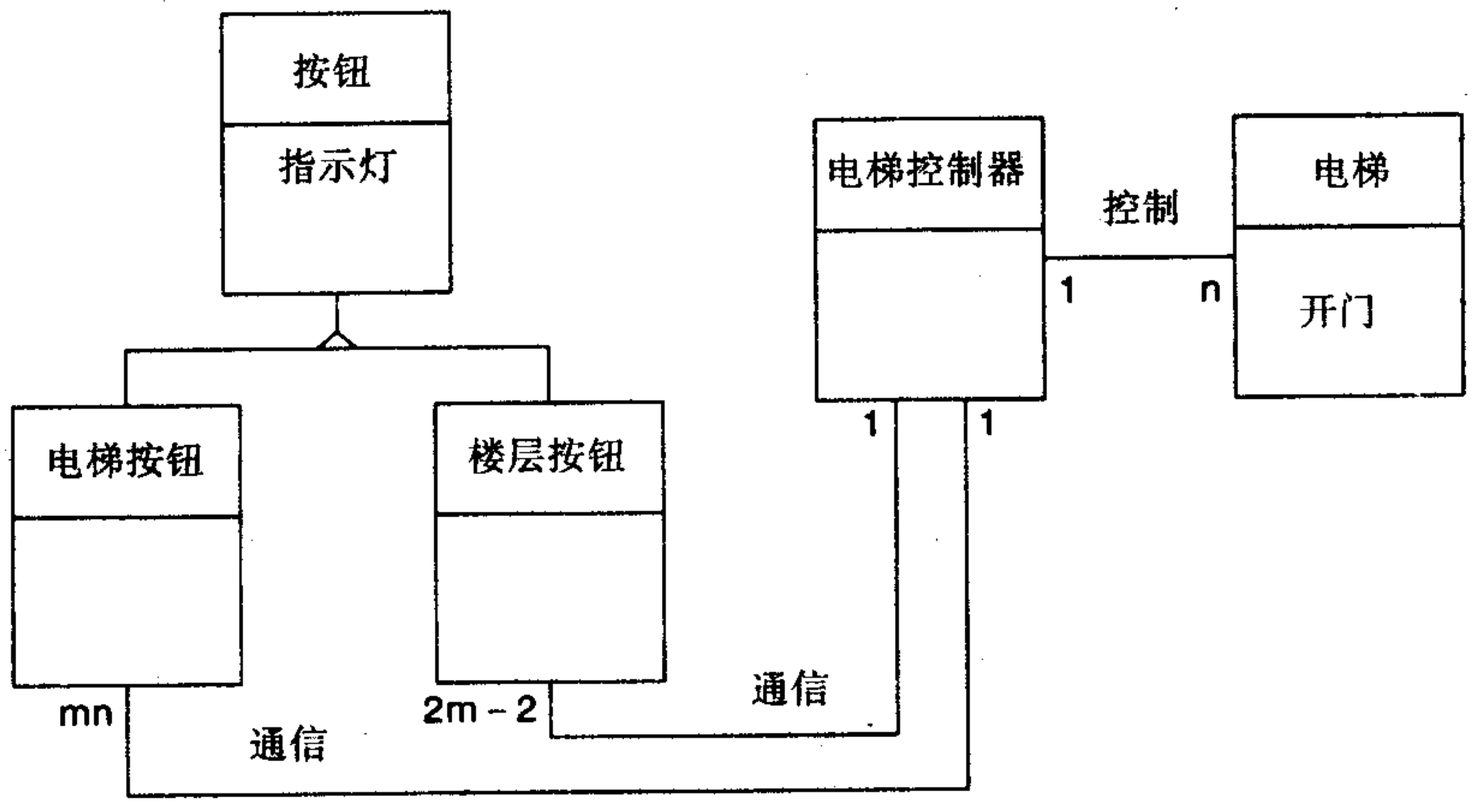

图9-2　类模型的第二次迭代

9.3.2　动态建模

这一步骤的目的是决定每一类需做的操作或对每一类所做的操作。达到这一目的较好的一种方法是列出用户和系统之间相互作用的典型情形(参见7.1节)，包括正常的和异常的情况。表9-1给出了一个正常的情形描述，而表9-2给出了一个非正常的情形描述。

表9-1　正常情形的说明

用户A在3楼按上行按钮呼叫电梯。用户A希望到7楼去
上行按钮指示灯亮
一部电梯到达3楼。电梯内有个用户B已按下了到9楼的按钮
上行按钮指示灯熄
电梯开门
用户A进入电梯
用户A按下电梯内到7楼的按钮
7楼按钮指示灯亮
电梯关门
电梯到达7楼
7楼按钮指示灯熄灭
电梯开门让用户A走出电梯
用户A出电梯
电梯在等待时间到后关门
电梯载着用户B继续前行到达9楼

表9-2　非正常情形的说明

用户A在3楼按上行按钮呼叫电梯，但用户A希望到1楼
上行按钮指示灯亮
一部电梯到达3楼，电梯内用户B已按下了到9楼的按钮
上行按钮指示灯灭
电梯开门
用户A进入电梯

（续）

用户A按下电梯内到1楼的按钮
电梯1楼按钮指示灯亮
电梯在等待超时后关门
电梯运行到达9楼
电梯内9楼按钮指示灯熄灭
电梯开门让用户B走出电梯
用户B出电梯
电梯在等待超时后关门
电梯载着用户A驶向1楼

应充分研究各种的情形，以使规格说明组能对系统模型化的行为有一个详细透彻的理解。然后，构造一个类似于有穷状态模型的用来描述目标产品的动态模型。首先考虑电梯控制器(Elevator Controller)类。为了简化起见，这里仅仅考虑一部电梯。电梯控制器的动态模型如图9-3所示。这里的概念在某种程度上类似于8.6节的有穷状态机(FSM)，但却有一点显著的不同。第8章里的FSM是作为一个形式化技术的示例。状态转换图(STD)本身并不完全代表要建立的产品，而是由前面给出的表达式(8.2节)的一系列转换规则构成，即：

当前状态 + 事件 + 谓词=>下一状态

这个表达式是通过以一套数学规则的形式来表达该模型而得到的。

反之，模型的表达在某种程度上就不那么正规了。有穷状态机中关于状态、事件和谓词这三个方面的清晰描述消失了。例如，在图9-3中，当进入“处理下一个请求”状态时，必须执行三个操作。其中之一是，如果灯亮，则关掉楼层按钮。也就是说，如果电梯的打算向方向d移动，而一部正向该方向驶来的电梯的按钮是开着的，则该按钮要关闭。这一动作包括状态、事件和全局谓词。当前的OOA版本是半形式化的技术，所以动态模型内在的非形式化并不是一个问题。但当面向对象范型成熟时，很可能更为形式化的版本将开发出来，那时对应的动态模型也许更接近有穷状态机。

同时观察图9-3所示的动态模型和从图8-12到图8-14所示的STD模型，并考虑说明中的不同部分。例如，考虑表9-1中所示的情形说明的第一部分。首先，用户A在3楼按上行按钮。如果楼层按钮指示灯不亮，则根据图8-13和图9-3所示最左边的路径，按钮要求灯亮。在动态模型中下一状态为“电梯控制器循环”。

下一步，电梯接近3楼。在图8-14中电梯进入状态S(U,3)，即电梯停在3楼,并即将往上行驶(因为我们简单地假设仅仅有一部电梯，所以图8-14中的变量e在此忽略)。从图8-13可知，当电梯到达时楼层的按钮熄灭，现在(图8-14)电梯关门，并开始向4楼驶去。

回到图9-3所示的动态模型，考虑当电梯接近3楼时发生什么？由于电梯处于运行中，故下一个状态即将进入“判定是否有停止请求”。电梯检测请求，由于用户A已请求电梯在3楼停，故电梯停在3楼(状态“停在楼层”)，开门。由于无人按下3楼的电梯按钮，故该按钮无须关闭。该请示被更新，以便反映电梯已停止在3楼。“电梯控制器循环”继续。由于电梯已停止且有被挂起的请求，故后继状态为“处理下一个请求”。当进入这一后继状态时，楼层按钮关闭。电梯关门，并驶向4楼方向。相应流图的有关方面清晰地反映了这一情形。读者也可考虑其他可能的情形。

其他类的动态模型则相对简单一些，留作练习(问题9.1)。一旦动态模型作为一个整体已经完成，则可根据在动态建模步骤中获得的信息重新考虑类模型(图9-2)。如果一切看上去依然令人满意，现在就可以开始第3步：功能建模。

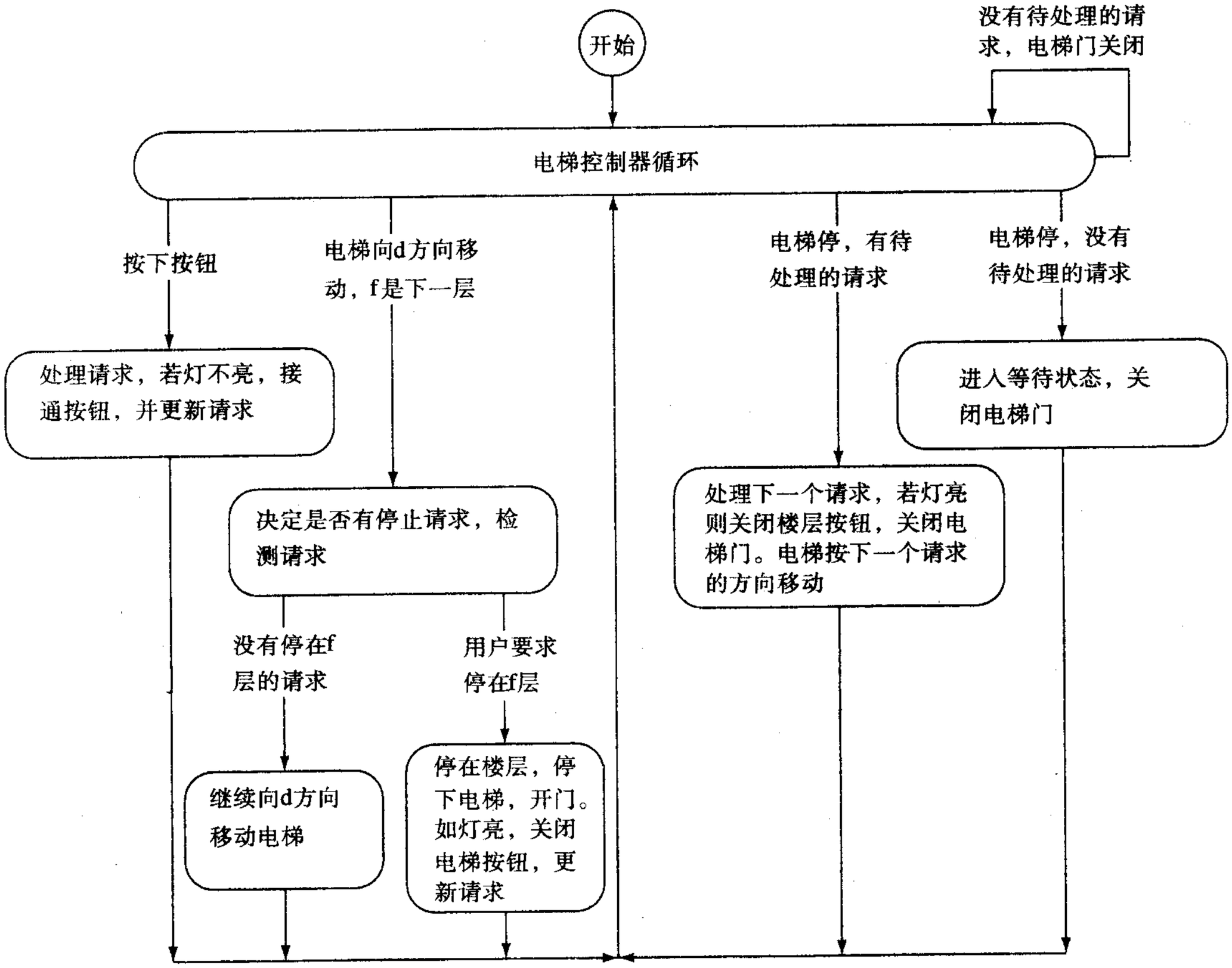

图9-3　电梯控制器类的动态模型

9.3.3　功能建模

OOA的第三个步骤是决定产品在不考虑动作次序的情况下，如何进行各种不同的动作操作。这种信息是以类似于数据流图(8.3节)的形式体现出来的，称为功能模型，因为它表示在产品范围内的功能相关性。

关于电梯问题的功能模型如图9-4所示。它按需要做出了各种动作，反映数据是如何在产品中流动的，但它没有反映动作的次序。

时常会遇到的问题是如何确定功能模型中的一个条目是源数据或目的数据(例如用户)，还是数据存储(例如电梯或电梯控制器)。确定这一点的关键是状态的概念。类属性有时也称为状态变量，其理由是在面向对象的实现中，产品的状态是由各种不同对象的属性值决定的。动态模型有许多与有穷状态机相同的特性，相应地，状态的概念在面向对象例程中扮演着重要角色，也就不足为奇了。这一概念能有助于决定一个条目应被当作源数据和目的数据模型还是数据存储模型。若一个条目拥有一个在实现执行期间将被改变的状态，则它应当作数据存储模型。从图9-4中可清楚地看到，电梯控制器、请求、电梯和门均具有状态动态地变化的特点。然而，用户只按按钮，故用户的状态却是不变的。

结构化范型中的DFD与功能模型之间的差别和数据存储有关。尤其在结构化范型中，数据存储将几乎肯定当作文件来保存。然而，如前一段所述，一个类的状态变量也是数据存储。因

此，功能模型包含两类数据存储，即类的存储和不属于类的数据存储。

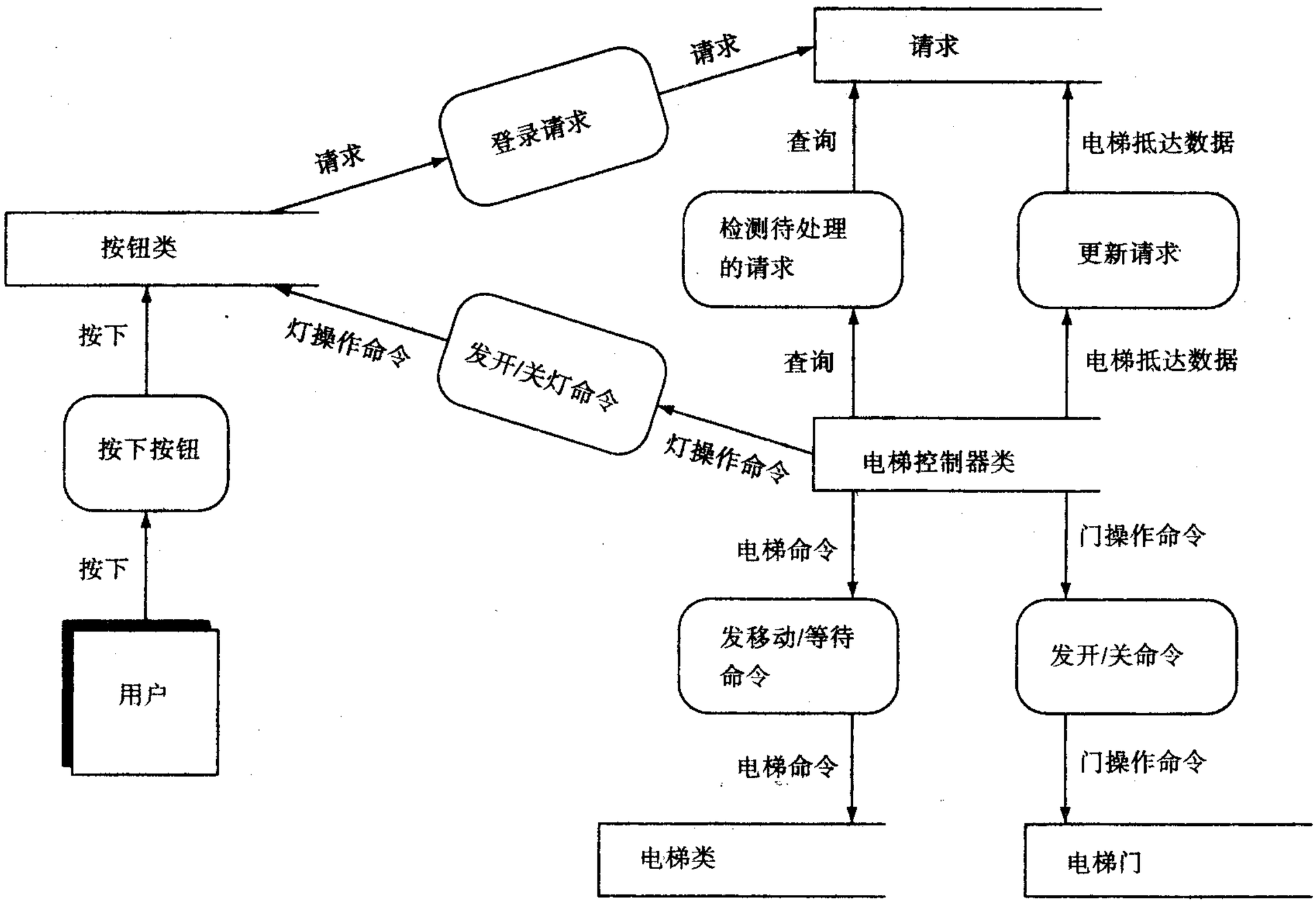

图9-4 功能模型的第一次迭代

从表示法上看，功能模型中的数据存储均以开口的矩形表示，如图8-1所示。在面向对象的分析阶段，可以更进一步对类进行抽象，现有类可能要被修改甚至抛弃。因此，功能模型中数据存储的性质可能要改变。

现在最好是根据在功能建模中获得的信息重新考虑一下类模型(图9-2)和动态模型(图9-3)。参照图9-4，“电梯门”应被标识为一个电梯门类，而不是把属性“开门”附到电梯类上。理由是面向对象的范型允许把状态隐藏在对象内，从而避免被非法修改。一旦将“电梯门”作为一个对象，则打开或关闭电梯门的唯一办法是向对象“电梯门”发送一条消息。而如果在错误的时间内开或关电梯门，会引起严重的意外事件，这一点请看“9-I 你想知道的进一步的信息”。因此，对某些类型的产品来说，第6章列出的关于对象的优点中，应加上考虑安全性这一条。

显然，出于和“电梯门”的状态在无授权情况下决不允许改变一样的原因，存储在图9-4的“请求”中的数据也必须隐藏起来。也就是说，做了这两项修改后，即加入电梯门类和请求类后，就形成了类模型的第三次求精，如图9-5所示。该图加入了两个新类。电梯应用类将被方法main所请求。同样在详细设计期间，几乎肯定需要加上各种实用例程，这些例程将被指定到电梯实用程序类中(Java作为纯面向对象的语言，没有过程和函数，因此，所有的数据和操作必须作为类的字段)。

修改了类模型之后，现在必须重新检查动态模型和功能模型，看是否需要进一步求精。如前所述，必须改变功能模型，将“请求”和“电梯门”标识为类；这项改变如图9-6所示。仔细审查便会发现，动态模型依然是适用的。然而在面向对象设计阶段，可能还需要回到面向对象分析阶段，修订其中一个或多个模型。

9-I　你想知道的进一步的信息

大约在几年前，我在一幢高楼的第10层焦急地等待着电梯。电梯门打开了，我迈步往里走，但里面却没有电梯。当我即将跨入电梯门时，眼前的一片漆黑提醒了我，我本能地意识到出了什么故障。

如果那部电梯的控制系统是用面向对象范型开发的话，电梯门在第10 层莫名其妙地打开的情况也许能避免。

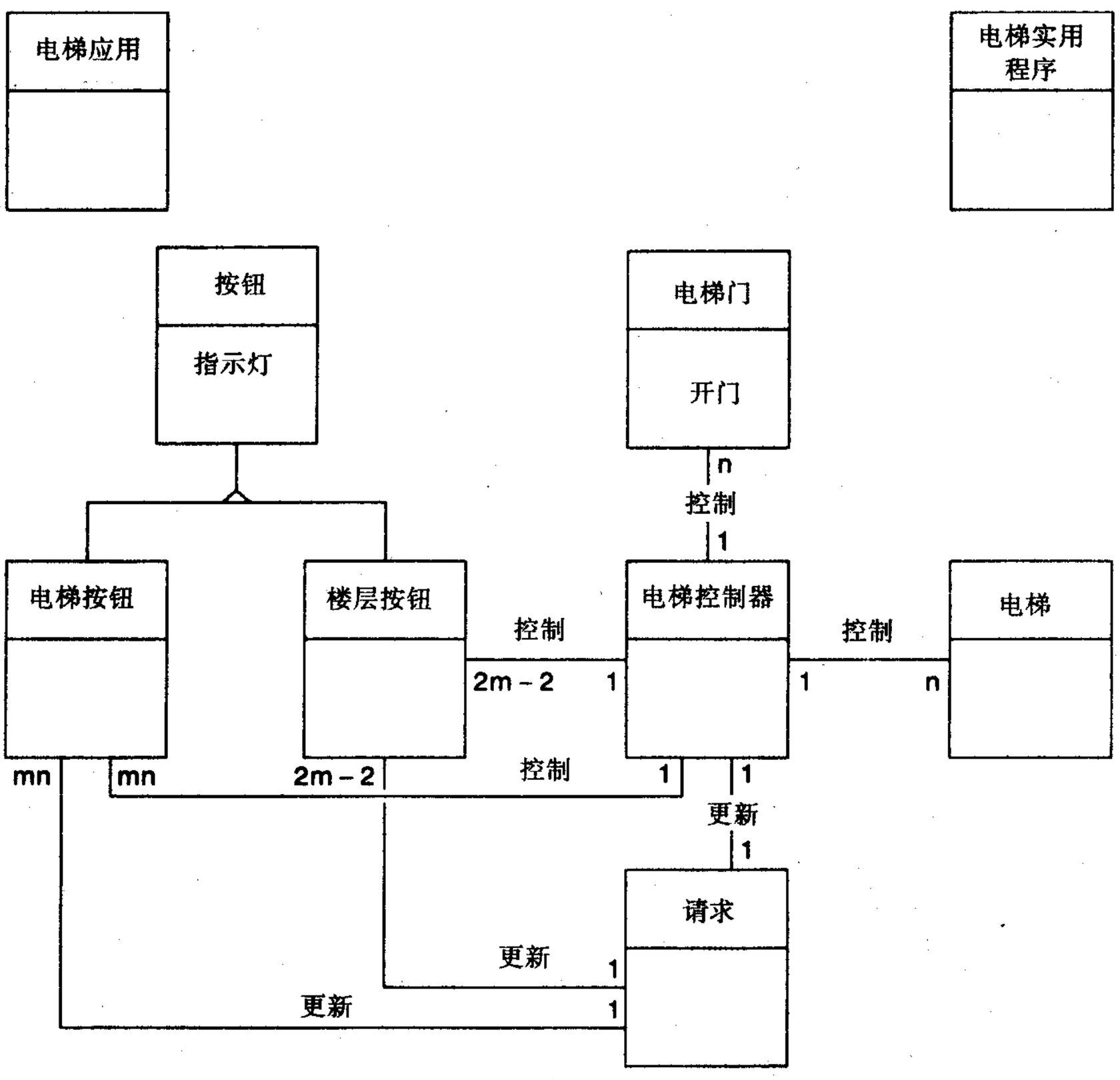

图9-5　类模型的第三次迭代

9.4　面向对象的生命周期模型

使用面向对象范型的经验表明，软件过程各阶段之间的迭代或各阶段的某部分之间的迭代，在面向对象的范型中似乎比在结构化范型中更常见。已经有人提出了明确地反映了逐步求精需求的面向对象生命周期模型。图9-7所示的喷泉模型[Henderson-Sellersand Edwards,1990]便是这类模型的一种。图中，代表不同阶段的圆圈相互重叠，这明确反映了两个活动之间的交迭。在一个阶段内的箭头代表在该阶段内的迭代(求精)。较小的圆圈表示维护，象征着当采用了面向对象的范型后维护期缩短了。

除了喷泉模型外，其他面向对象生命周期的模型也已提出，包括递归／并行(recursive/parallel)生命周期模型[Berard,1993]，循回完形设计(roundtrip gestalt design)模型

[Booch,1994]。所有这些模型均为迭加式,融入了某种形式的并行机制(活动的交迭),支持增进开发(3.4节)。这类生命周期模型存在的危险是,它们可能被曲解成出于需要而试图突出某一方面优点,因而导致软件开发的完全无序,开发组成员几乎可以在各阶段之间随意地移动。他们可以首先设计一件产品,然后分析另一件产品,然后实现另外一件既未分析过又没有设计过的产品。“9-II 你想知道的进一步的信息”中对这种不好的方法做了更为详细的说明。解决这一问题较好的一种办法是把一个线性处理过程(如3.3节中的快速原理模型或图9-7的中心垂直线)作为总目标。但同时应记住,面向对象范型的本身要求必须经常不断地对模型进行迭代和求精。

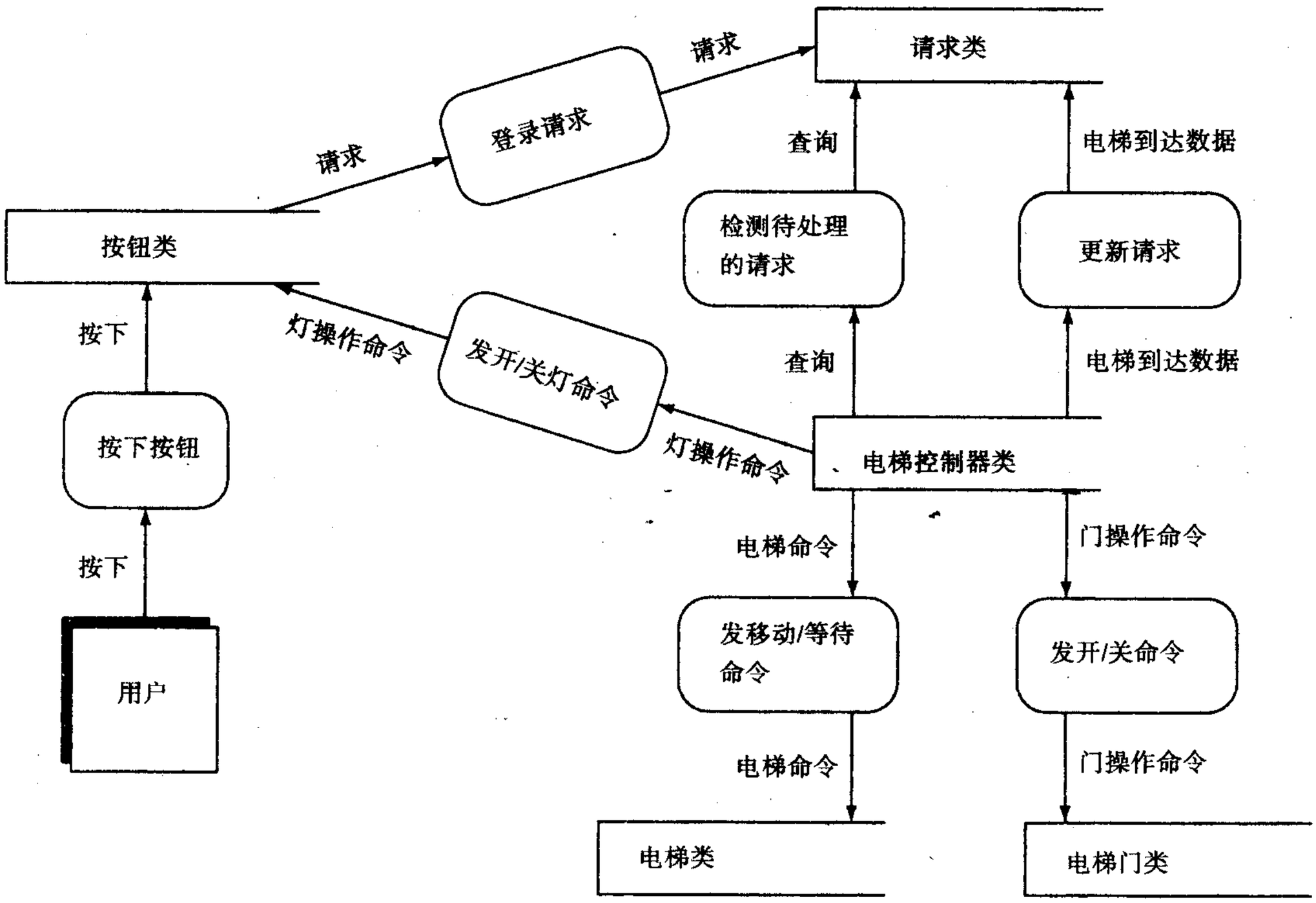

图9-6 功能模型的第二次迭代

9-II 你想知道的进一步的信息

当开发组成员偶而需要从一种任务转到另一任务时,有时把这称为CABTAB(编码一点,就测试一点——code a bit, Test a bit)。这个缩写词最初用于褒义,意指成功的交迭模型,例如那些已被用于与面向对象范型连接的模型。然而正如hacker这一单词如今已在它最初的意义里加入了贬义一样,如今CABTAB也被用作贬义,用来形容这种软件开发方法的任意性。

有人可能会说,以上问题只不过是由于面向对象范型才出现不久。这种论点认为,随着软件专业人员在OOA和OOD方面经验的积累和整体规则的成熟,重复地检查和修订的需求将会减少。为证明这种论点是荒谬的,让我们来考察结构化范型的各种生命周期模型(第3章)。首先出现的是带有明确反馈环的瀑布模型(3.2节)。接下来的主要是研究快速原型模型(3.3节),该模型的主要目标之一便是减少迭代的需求。然而,其后的螺旋模型(3.5节)明确地反映了软件开

发和维护的一种重复迭代的方法。此外，回溯(backtracking)是Coad-Yourdon面向对象的设计技术中一个本质方面[Coad and Yourdon,1991a,1991b]，这一点已在[Honiden,kotaka and Kishimoto,1993]中得到了证明；也许从其他的OOA技术中也会获得类似的结论。换句话说，迭代似乎一般是软件生产的一种内在属性，特别是在面向对象的范型中更是如此。

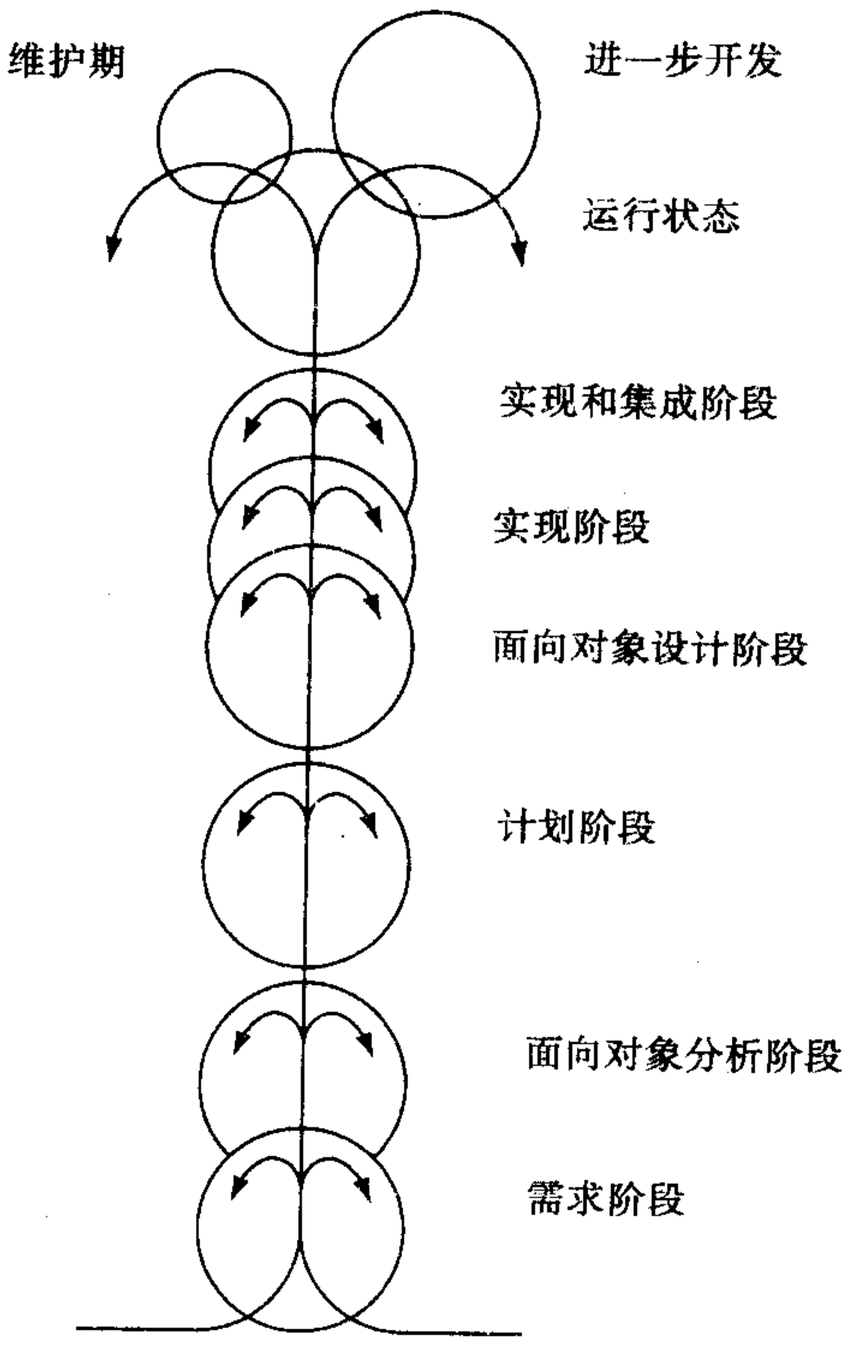

图9-7　喷泉模型

9.5　面向对象分析阶段中的CASE工具

在知道了图解表示法在面向对象的分析阶段中所起的作用之后，对人们开发了一系列CASE工具包来支持面向对象分析这一点也就不会感到奇怪了。从基本形式上说，这样的工具本质上 是绘图工具，可使建模步骤的每一步执行起来更容易。更为重要的是，修改一幅用绘图工具绘成的图表比修改一幅手工绘的图表容易得多。因此，这类CASE工具更多地支持面向对象分析的绘图方面。

另一方面，某些CAES工具不是只支持面向对象的分析，而且还支持面向对象生命周期中其它很多部分。这类工具的一些例子包括：浏览图片的软件、OMTool(它支持OMT[Rumbaugh et al.,1991])以及Rose(它支持Booch技术[Booch,1994])。

9.6　MSG实例研究：面向对象的分析

OOA由3个步骤组成，即类建模、动态建模和功能建模。这些步骤通常是交互进行的，第1步是类建模。这一步的目的是抽取类，找出它们的属性，并确定它们的相互关系。抽取类的工作通常有3个阶段。阶段1是尽可能简洁地定义产品，最好是用一句话。对于MSG，一种可能的定义方法是：

确定每周有多少钱可用于抵押。

第2阶段是拟定非形式化策略，最好用一段文字表示，一种可能的表述是：

每周要打印出关于有多少钱可用于抵押的报告。此外，投资和抵押的报表必须按需要随时打印。

第3阶段是从上一段中抽取出名词，如报告、钱、抵押、列表和投资。名词报告和列表是解答的一部分，而钱是一个抽象名词。余下的两个名词是候选类，即投资和抵押，它们均为金融工具，并且这两种类型的记录有许多共同操作，如插入一个新的记录、删除一个记录及显示一个记录。因而，定义一个超类——金融工具，应该是合理的。这个超类有投资和抵押两个子类。通过7.14节的需求分析可得出这些类的一个最小属性集。还需要另外的两个类，即main方法调用的MSG应用类和为实用程序设置的MSG实用程序类(MSG Utilities)；在设计期间，几乎每一件产品都会产生对实用程序的需求。同样，全局常量将被声明为MSG应用类的 Public Static final变量，图9-8给出了类模型的结果。

面向对象分析的第2步是动态建模。为开发动态模型，应首先拟定情景说明。在MSG模型中，可能的情景与可能具有的操作功能相对应，这些功能是添加、修改或删除一个金融工具(投资或抵押对象)、更新操作费用、列金融工具表、确定资金的可用性。这些操作反映在图9-9的动态模型中。

面向对象分析的第3步也是最后一步为功能建模。图9-10所示的功能建模除了6个数据存储外，非常类似于图8-26的数据流图。6个数据存储中有两个已被定义为类，即投资类和抵押类。对于其他4个数据存储，即经费数据、利润数据、贷款数据、收入数据，它们是不是应该作为类呢？在这一步骤中没有理由产生更多的类(请看问题9.11)。因此，最好不要返回第1步去修改图9-8的类模型，而宁可等到面向对象的设计阶段去决定是否实际需要附加的类。

客户通常将不认为这三个模型是恰当的规格说明文档。因此，必须起草一份客户更为习惯的文档，因为这种文档与附录D所提供的材料很相似，为节省篇幅，在此予以省略。

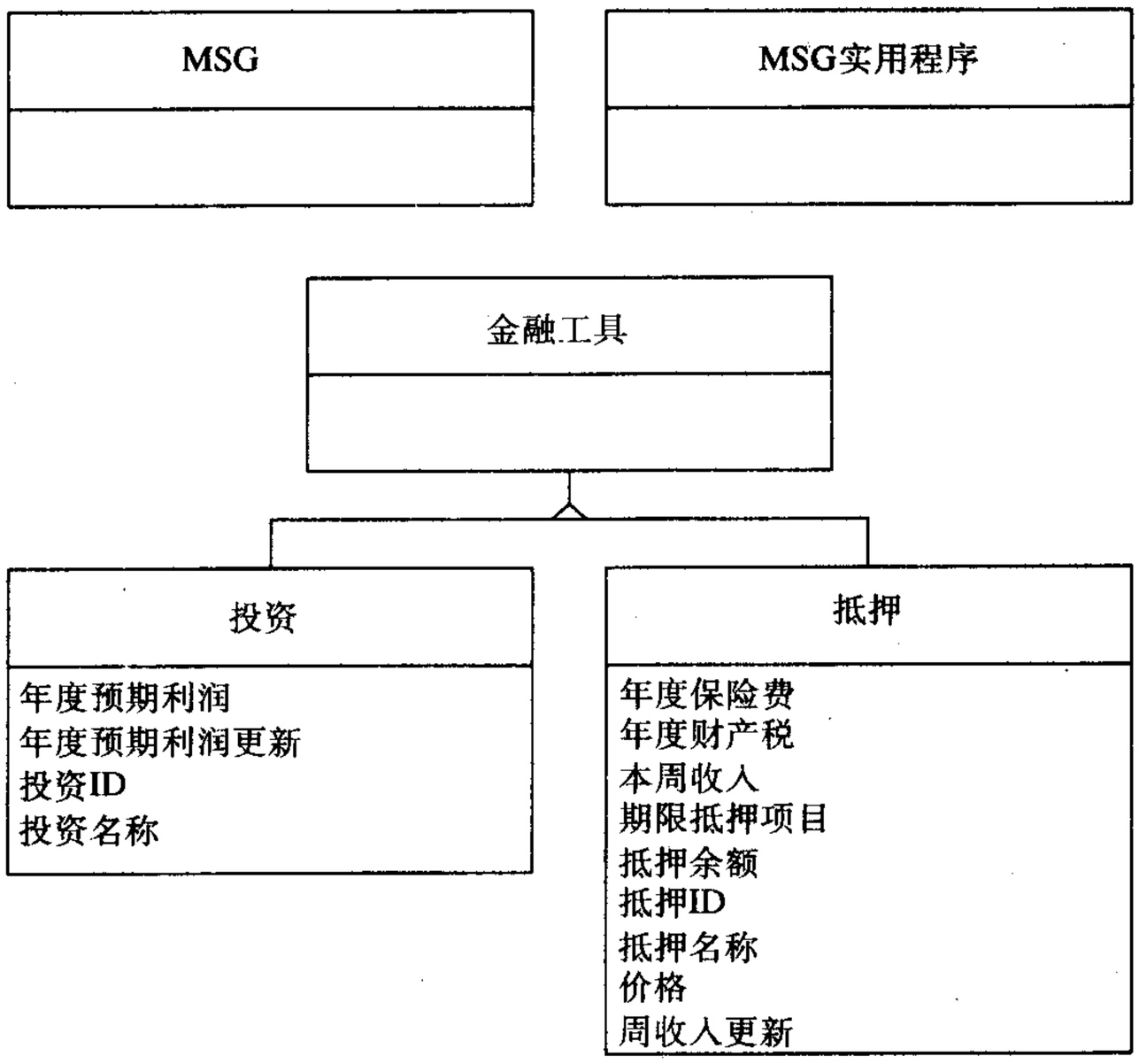

图9-8 MSG产品的类模型

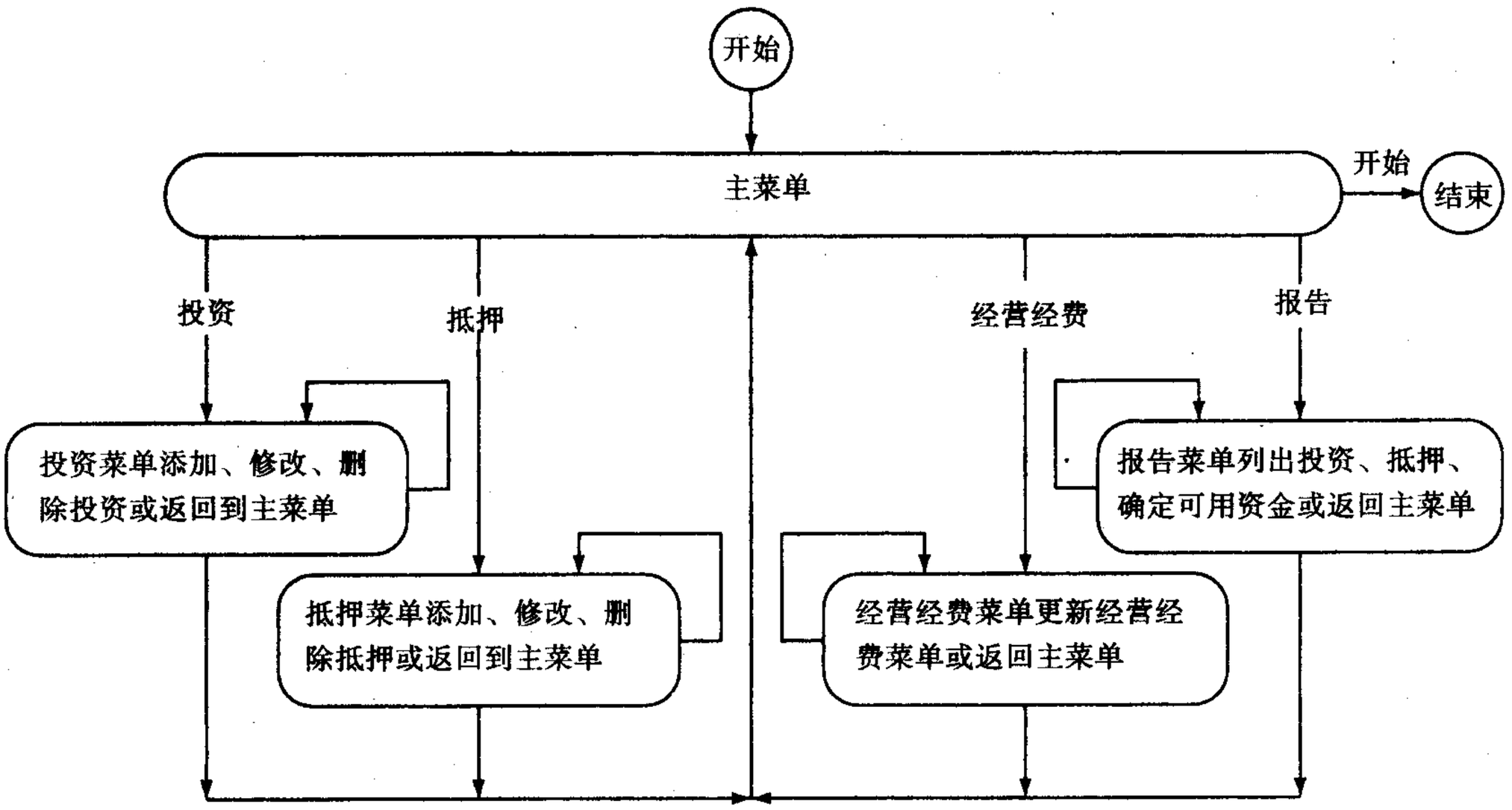

图9-9　MSG产品金融工具类的动态模型

本章回顾

9.1节比较了面向对象范型和结构化范型。然后在9.2节描述了面向对象的分析，并在9.3节以对一个事例的研究作为分析演示。接着，9.4节对面向对象的生命周期的各种模型做了比较。9.5节讨论了面向对象分析中CASE工具的运用。本章以9.6节中MSG事例的研究结束。

进一步阅读

描述不同版本面向对象分析的各种书籍有[Coad and Yourdon,1991a]、[Rumbaugh et al,1991]、[Jacobson Christerson,Jonsson and Overgaard,1992]、[Martin and Odell,1992]、[Shlaer and Mellor,1992]及[Booch,1994]。正如本章中所述，这些技术(以及其他在这里未列出的技术)基本上是类似的。除了这种类型的面向对象技术外，Fusion技术[Coleman et al.，1991]是第二代OOA技术，也就是说Fusion技术是一种将数种第一代技术组合(或熔合)而成的技术，包括OMT[Rumbaugh et al.，1991]和Objectory[Jacobson,Christerson,Jonsson,and Overgaard,1992]。

1992年9月出版的《Communicalion of the ACM》中有许多论述面向对象的文章。其中发表了许多比较各种面向对象分析技术的文章，包括[de Champeaux and Faure,1992]及[Monarchiand Puhr,1992]。也有将面向对象分析技术和结构分析技术进行比较的文章[Fichman andKemerer,1992]。[Capper,Colgate,Hunter,and James,1994]描述了三个使用面向对象范型的工程项目，并解释面向对象范型的方法在那些方面优于结构化方法。状态图(8.6节)已被扩展应用到了面向对象的范型[Coleman,Hayes and Bear,1992]。

面向对象的生命周期模型在[Henderson-Sellers and Edwards,1990]中，以及在本节的第一段中列出的面向对象分析的书籍中做了描述。

问题

9.1 为图9-5中的其他类开发动态模型，完成本章中电梯问题的实例研究。

9.2 设计一台自动取款机(ATM)。用户往ATM槽里插入一张卡，输入4位个人标识号(PIN)。如PIN不正确，就退出卡片。例如正确，则用户可在四个不同的帐户上进行下列操作：

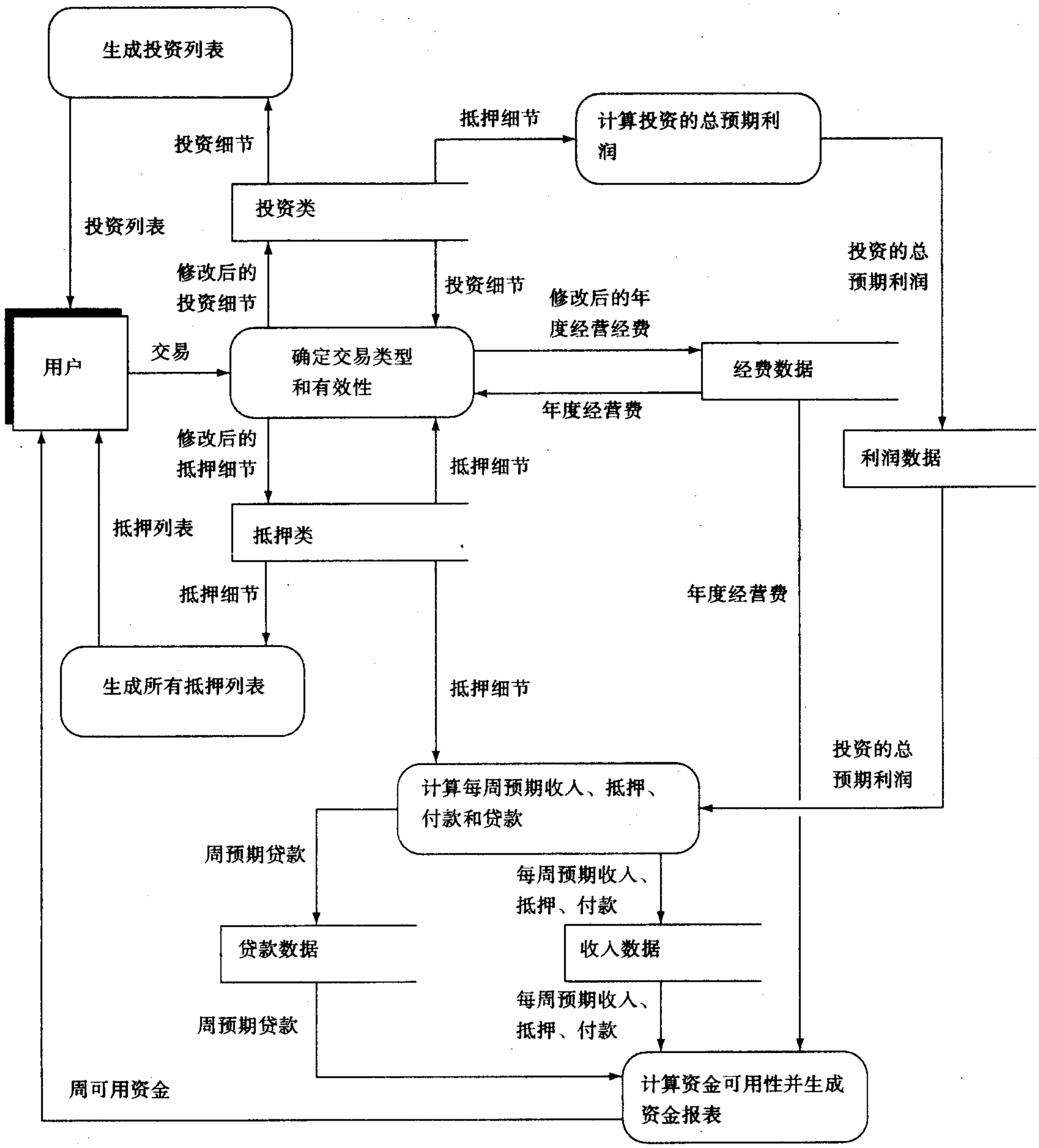

图9-10 MSG产品的功能模型

1) 存入任意数目的款项，ATM打印出一张标有存款日期、存款金额和银行帐户的收据。

2) 以20美元为一单位取出200美元(帐户不能被透支)。除了取出的钱外，还应给用户一张收据，其上应有日期、取款金额、银行帐户和取款后的帐户余额。

3) 屏幕显示用户操作后的帐户余额。

4) 在两个帐户之间转移资金，且被转移帐户的资金不能被透支。并给用户一张收据，其上应有日期、转移金额和两个帐户的帐号。

5) 退出，把插卡返回给用户。

用面向对象分析的方法来详述对ATM进行控制的软件，不需要考虑如读卡机、打印机、现款分配机等的硬件组成细节，只简单地假定当ATM向它们发出指令时，它们能正确执行。

9.3 面向对象分析采用的数据流图和第8章中描述的一样，为什么不能把第8章中的有穷状态机形式(无变化地)用于面向对象的分析?

9.4 用面向对象分析的方法详细描述问题6.18中的图书馆图书流通系统。

9.5 在面向对象的分析过程中，能在不对工程造成有害影响的情况下引入类的最迟点是什么时候?

9.6 在面向对象的范型中能引入类的最早点是什么时候?

9.7 动态模型能否以形式化的方法而不是本章描述的有穷状态机的方法给出？ 并解释你的答案。

9.8 瀑布和喷泉有什么共同点？本书介绍的瀑布模型和喷泉模型有什么共同点？它们的区别又是什么?

9.9 你认为为什么在面向对象分析阶段能确定类属性而不是类方法?

9.10 (学期项目) 用面向对象的分析方法详述附录A中的Osbert Oglesby产品。

9.11 (实例研究)在MSG的面向对象的分析里加入经费、贷款、利润和收入类，这是一种改进还是造成了不必要的复杂性?

9.12 (实例研究)当面向对象的分析从功能建模步骤开始时，将会发生什么？从图8-26的数据流图开始，完成面向对象的分析过程。

9.13(实例研究)对照和比较9.6节中面向对象的分析和附录D中的结构化系统分析。

9.14 (软件工程文献阅读)导师将给你们分发[Honiken,Kotaka and Kishimoto,1993]的拷贝。你赞同本书9.4节结尾部分所说的“从其他的OOA技术中很可能获得相似的结果”这一结论吗?