

第14章. 面向对象建模

14.1. 概述

面向对象建模是面向对象方法学在需求分析中的应用，所以也称为面向对象分析。它采用面向对象方法学的世界观，将系统看作是一系列对象的集合。每个对象具有独立的职责，完成独立的任务，对象之间通过消息机制互相协作，共同实现系统的目标。

虽然面向对象编程出现的很早，在20世纪60年代。但是面向对象分析的方法却是直到20世纪90年代才初现端倪，它是在面向对象编程和面向对象设计得到大规模应用的情况下，吸收了传统软件方法学的很多成果之后才产生的。所以，虽然面向对象分析是现在需求分析的主流方法，但是读者在学习面向对象分析时要认识到它并不完全排斥传统方法中所采用的技术，它自身所使用的一些技术（例如用例模型、状态机模型）也并不是基于面向对象方法学的。在很多情况下，它需要和传统的技术互相配合才能取得更好的效果。

在产生之初，包括面向对象分析在内的面向对象方法出现了很多分支，其中主要的有Grady Booch的Booch方法[Booch1997]、Ivar Jacobson的OOSE方法[Jacobson1992]、James Rumbaugh的OMT方法[Rumbaugh1991]、Perter Coad与Edward Yourdon的Coad-Yourdon[Coad1990]方法、Sally Shlaer与Stephen J. Mellor的Shlaer-Mellor方法[Shlaer1988]等。这些不同的方法有着各自不同的技术、概念、表示法和开发过程，相互之间难以进行工作的协同。所以人们试图综合不同分支建立统一的面向对象方法，建立统一建模语言UML（Unified Modeling Language）。UML后来被OMG采纳作为面向对象建模的标准方法，得到了广泛的认同。

14.2. UML 与面向对象分析

14.2.1. UML 的需求分析模型

UML是以Booch方法、OOSE方法和OMT方法为基础，发挥三种方法的互补性（如图14-1所示），互相取长补短，并吸收了OCL（Object Constraint Language）等其他技术之后产生的。本书就以UML为面向对象建模的标准方法，介绍面向对象的需求分析过程。UML的表示法比较复杂，本书仅仅解释主要的概念和图示，更详细的内容请参考UML专门著作[Rumbaugh2004, Booch2005]。

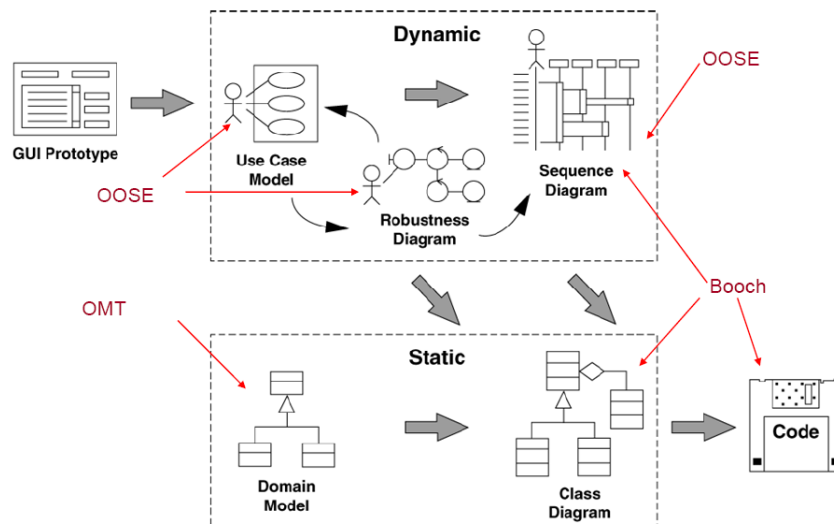


图14-1 Booch方法、OOSE、OMT与UML

虽然被称为统一建模语言，但UML其实是很多种技术的综合体。这些技术在一个统一的框架下，能够很好的实现相互之间的协同，共同构成完整的面向对象开发方法。在需求分析中涉及的UML技术有：

- 用例图（用例模型）；
- 类图（对象模型）；
- 交互图（顺序图/通信图，行为模型-对象协作）
- 状态图（行为模型-状态机）；
- 活动图（行为模型-业务过程）；
- 对象约束语言 OCL。

14.2.2. 基于 UML 的面向对象建模思路

如图14-2所示，面向对象分析方法在定义项目前景与范围时使用活动图分析业务过程，使用用例模型组织需求信息。面向对象分析与设计的最终目标则是建立完全的对象模型，并基于完全的对象模型完成向编码的平滑过渡，如图14-3所示意。

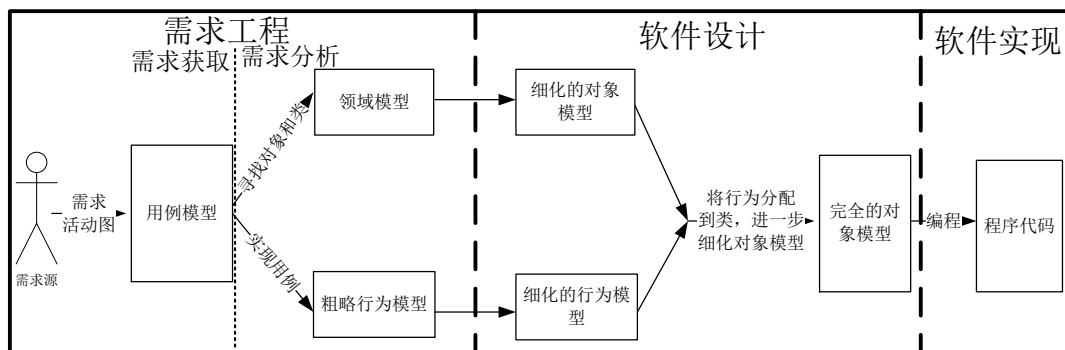


图14-2 面向对象方法的技术路线

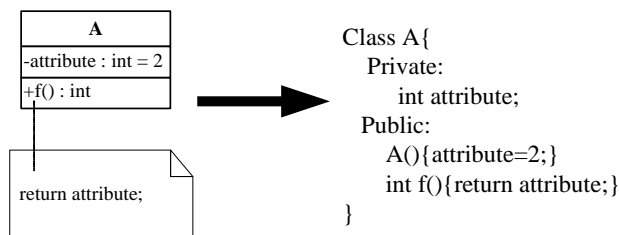


图14-3 完全类图向编码平滑过渡的简单情景示意

所以,面向对象分析与设计联合起来要做到的关键是实现从用例模型到完全对象模型的过渡,包括下列几个步骤(如图14-2所示):(1)从用例描述中识别出应用领域的对象和类,建立分析类图;(2)从用例描述中识别系统行为,建立分析的行为模型;(3)考虑设计要素,将分析类图转化为初始设计类图;(4)基于初始设计类图,将分析的行为模型转化为设计的行为模型;(5)综合考虑初始设计类图与设计行为模型,将系统行为分配给类,细化类的职责,建立完全的对象模型。

面向对象分析与设计的几个步骤可以划分为四个阶段:

①面向对象方法首先从需求的源头(主要是用户)进行需求的获取,描述业务流程(活动图),组织需求信息的用户描述,建立用例模型。

②在得到用例模型之后,面向对象方法一方面要从用例模型中寻找对象和类,建立领域模型,领域模型描述了业务工作中的概念类和类的重要属性;另一方面,面向对象方法依据用例模型建立行为模型,行为模型是用例模型的实现,体现了用例描述中的系统行为。这个阶段是需求的分析阶段,它的主要目的是理解用户的需求,所以它建立的对象模型和行为模型都是比较粗略的模型,还没有涉及与软件实现相关的技术细节。

③在得到用户需求的完整、准确理解之后,面向对象方法就开始考虑软件的实现机制,进行软件设计。设计阶段以软件的高质量实现为第一目的,所以设计阶段需要在领域模型和粗略的行为模型当中加入软件实现的细节信息,进行模型的细化,建立细化的对象模型和行为模型。最后结合细化后的对象模型和行为模型,为类进行系统行为的分配,完成最终的完全对象模型。

④在得到完全的对象模型之后,编程人员就可以选择一种面向对象的语言,完成程序的编写,使得软件变成实际的存在形式——程序代码。

整体来说,①、②阶段是需求分析的任务,③阶段是设计的任务,④是实现的任务。面向对象方法中分析和设计的分界线比较模糊,因为:(1)分析和设计都使用相同的模型,只是模型内包含的信息有所不同而已,分析侧重于需求的理解,设计侧重于实现的细节;(2)关于需求理解信息(What)和实现细节信息(How)的区分比较模糊[Siddiqi 1994];(3)模型“粗略”和“细化”仅仅是程度上的不同,无法定义一个非常准确的区分标准,所以无法准确界定分析应该何时终止对粗略模型的加工以及设计应该何时开始对模型进行细化。

因为前面章节以及介绍了活动图和用例模型,所以本章主要描述的面向对象分析工作有:

- 对象模型(类图):建立领域模型
- 交互图、状态图:建立行为模型

14.3. 对象模型

对象模型以对象和类的概念为基础,描述了系统中的对象和这些对象之间的关系。建立对象模型的过程被称为对象建模,它是面向对象建模的核心技术。

需要指出的是,对象模型(面向对象)的思想起源是很早的,可以追溯到20世纪60年代的Simula语言。早期的对象模型以对象、类及其之间的关系为基本元素进行系统模型的描述,这种早期的对象模型技术被称为基于对象(Object-Based)的技术。但是仅仅包含对象、类及其之间的关系还是远远不够的。在后期的发展当中,在对象、类及其之间关系的基础之上,对象模型又扩充了继承和多态等重要概念和方法,才最终形成了现在的面向对象(Object-Oriented)技术。所以,读者在学习当中需要认识到,除了对象、类及其之间关系之外,继承和多态也是面向对象方法的重要组成部分。

14.3.1. 对象

对象的概念是对象模型的基础。从理解现实世界的角度来看,它是对现实世界事物的抽象,也就是说对象代表了现实世界的事物。当然,在软件设计当中,也会出现一些在现实世界中没有对应事物的对象。不过在需求分析当中,对象都应该代表相应的现实世界事物,这也是面向对象分析和面向对象设计的一个不同点。

对象描述的是一种比较常见的存在,很多事物都可以被模型化为对象。常见的有:

- 和系统存在交互的外部实体,例如人、设备、其他的软件系统等;
- 问题域中存在的事物,例如报表、信息展示、信号等;
- 在系统的上下文环境中发生的事件,例如一次外部控制行为、一次资源变化等;
- 人们在与系统的交互之中所扮演的角色,例如系统管理人员、用户管理人员、普通用户等;
- 和应用相关的组织单位,例如分公司、部门、团队、小组等;
- 问题域中问题发生的地点,例如车间、办公室等;
- 事物组合的结构关系,例如部分与整体的关系等。

但是,对象并非普遍的存在,有一些事物是无法抽象为对象的。[Smith1988]给出的对象概念的定义为:对象是指在一个应用当中具有明确角色的独立可确认的实体。在这个定义当中,强调了一个事物可以被抽象为对象的两个条件:独立可确认;有明确的角色。

1. 独立可确认

独立可确认要求对象能够从周围的环境当中界定自己。如果一个事物无法在周围环境当中界定自己,那么这个事物就不是独立可确认的,也就是无法抽象为对象的。这里需要强调的是,一个事物是否可界定是相对于环境的,是相对于问题域的。例如,在一个图书管理系统当中,每一本图书都是可以界定的,但是每一本图书对借阅者的实际知识贡献是无法界定的,所以,可以利用一个抽象对象来描述图书,但是无法使用一个抽象的对象来描述图书给借阅者产生的实际价值。但是,如果存在另外一个专门分析借阅者知识结构的系统,就能够界定每一本图书对借阅者的实际知识贡献,就可以将其抽象为对象。在系统对事物的信息有需要时,这些无法界定的事物往往会以其他对象属性的方式出现。

界定的对象需要拥有一个标识符以唯一的标识自己。和实体关系模型中使用键来标识实体不同,对象模型使用对象的引用作为对象的标识。如果一个对象a需要和另一个对象b合作,那么a就必须要从外界环境当中将b识别出来,也即a需要知道b的引用。

2. 有明确角色

对象角色 (Role) 被认为是对象职责 (Responsibility) 的体现, 职责是指对象持有、维护特定知识并基于知识行使固定职能的能力, 因此, 要求对象有明确的角色就是要求对象在应用中维护一定的知识和行使固定的职能, 简单的说就是要拥有状态和行为。

状态是对象的特征描述, 包括对象的属性和属性的取值, 属性是描述对象时使用的特征选项。对象的行为通常是依赖于状态的, 如果状态发生了变化, 那么对象的行为往往也会随之变化。虽然在软件设计当中, 会出现没有状态而仅有方法的对象。但是在需求分析当中, 没有状态的事物是不能被抽象为对象的。例如, 一个纯粹的随机数字产生行为不能被抽象为对象, 而在一定状态 (例如取值范围或者历史的取值记录) 基础之上的随机数字产生行为就可以被抽象为对象。

行为是对象在其状态发生改变或者接收到外界消息时所采取的行动。对象的行为是基于其状态的, 而其状态又是历史行为的累积, 所以对象的多个行为之间往往具有相关性。如果对象的所有行为都互有相关性, 那么可以认为它拥有一个内聚的状态和行为, 此时称该对象具有单一职责, 是理想的抽象结果。现实世界当中只有状态没有行为的事物不能被抽象为对象, 例如蓝色、5M等原子属性及其取值。

现在, 再来重新认定对象角色的含义。一个对象维护其自身的状态需要对外公开一些方法, 行使其职能也要对外公开一些方法, 这些方法组合起来定义了该对象允许外界访问的方法, 或者说限定了外界可以期望的表现, 它们是对对象需要对外界履行的协议 (Protocol)。一个对象的整体协议可能会分为多个内聚的逻辑行为组, 例如, 一个学生对象的有些行为是在学习时发生的, 而另外一些可能是在购物时发生的, 这样, 学生对象的行为就可以分为两组。划分后的每一个逻辑行为组就描述了对对象的一个独立职责, 体现了对象的一个独立角色。如果一个对象拥有多个行为组, 就意味着该对象拥有多个不同的职责, 需要扮演多个不同的角色。例如, 上例的学生对象就需要同时扮演学生和顾客两个角色。每一个角色都是对象一个职责的体现, 所有的角色是对象所有职责的体现。所以, 理想的单一职责对象应该仅仅扮演一个角色。

综合上面的论述, 对象是对现实世界事物的抽象, 在应用中履行特定的职责。对象具有标识、状态和行为。

14.3.2. 对象之间的关系

系统中的对象不是孤立存在的, 它们需要互相协作完成任务。对象之间的这种互相协作的关系称为链接 (Link), 它描述了对对象之间的物理或业务联系。

链接通常是单向的, 当然也有双向的链接存在。如果一个对象a存在指向b的链接, 那就意味着a拥有对b的假设, 关于b的行为和行为效果的假设。也就是说, b需要满足a的某些行为期望。这样, 如果a需要这些行为, 它就可以按照假设向b发出请求并得到期望的结果。在相反的情况下, 如果a对b没有任何期望, 那么a就不会向b发出请求, 由a指向b的链接就是冗余的对象间关系。

由a指向b的链接除了包含假设和期望因素之外, 还意味着a能够在链接的指引下, 正确的找到并将消息发送给b。这种情况下, b是对a可见的, 或者说a拥有b的可见性 (Visibility)。如果一个对象a拥有了对象b的可见性, 那么a就可以按照b的协议发送消息, 请求b的服务。

a获取b的可见性或者说建立由a指向b的链接的途径有以下几种：

- b 是全局对象，它对系统内的所有其他对象都是可见的；
- b 是 a 的一部分；
- b 是被 a 创建的；
- b 的引用被作为消息的一部分传递给了 a。

14.3.3. 类

1. 类的概念

对象是一个存在于一定时间和空间中的具体实体，而人们认识和处理具体事物时总会有意识或无意识的对它们进行归类和抽象，类就是将对象进行归类和抽象的结果。

类是共享相同属性和行为的对象的集合，它为属于该类的所有对象提供统一的抽象描述和生成模板。抽象描述称为接口（Interface），定义了类所含对象对外的（其他类和对象）的统一协议。生成模板称为实现（Implementation），说明了类所含对象的生成机制和行为模式。

每个类都有能够唯一标识自己的名称，同时包含有属性和行为方法。类的UML的表示如图14-4（a）所示，图14-4（b）展示了一个类Student的UML图示。

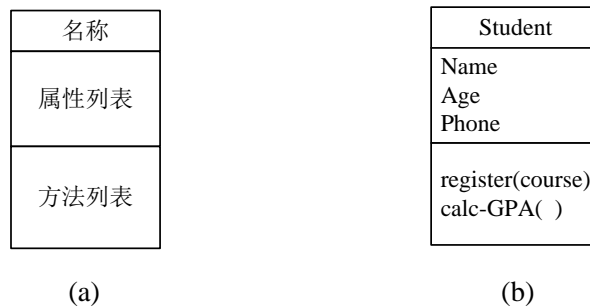


图14-4 类的UML图示

对象是类的实例，对象的UML图示如图14-5所示。因为对象的行为都是和类的方法声明保持一致的，所以在对象的图示当中没有方法列表。



图14-5 对象的UML图示

2. 类的产生——分类

“类”的概念体现了它的含义，它是众多对象分类后产生的类别。因此，分类方法的不同自然会导致所产生的结果类不同。目前常见的类产生方法有数据驱动（Data-Driven）和职责驱动（Responsibility-Driven）两种。

受到传统方法的影响，尤其是信息工程方法的影响（因为对象模型在某种程度上是在实体关系模型的基础上发展出来的），人们一度以数据（即对象的属性）作为对象分类的主要标准，将具有相同属性的对象归

为一类,这种类产生方法被称为数据驱动方法。[Booch1993]将这种类产生方法归因于哲学上传统的经典分类理论 (Classical Categorization Theory)。在经典分类法中,所有具有一个给定特性或共同特性集的实体组成一个类。因此,在考察一个对象的类别或者定义一个类别概念时,会首先考虑它是否具有某些指定特征。例如,这种理论就曾经将“人”定义为“无羽毛的双脚直立行走动物”。

经典分类理论明显是有缺陷的,人们在更多的时候会依据事物的相似性而不是完全的相同性来进行事物的分类,后一种分类方法在哲学上称为概念聚类 (Conceptual Clustering)。概念聚类使用概念描述而不是指定的特征来描述类别和事物。在进行事物分类时它会考虑概念之间的相似性,并将事物归入和其概念最为相似的类别。概念聚类在类产生方法中的应用就是职责驱动的方法。职责驱动方法要求结合对象的状态和行为来描述对象的职责,然后根据对象间职责的相似性进行聚集和分类,进而产生对象集的类。

3.类的产生——抽象

从分类之后,每个类别会有很多的对象实例,它们有相似性,也有差异性。从众多的对象当中归纳出共同的类描述的过程依据的是抽象 (Abstract) 原则。抽象是指在事物的众多特征当中只注意那些和目标密切相关的特征,同时忽略那些不相关的特征,进而找出事物的本质和共性。

抽象是人们在理解事物时常用的手段。在面对的事物过于繁杂或者需要寻找众多事物的共性时,人们就会运用抽象原则。通过以特定的目标和要求为导向,忽略不重要的特征。抽象可以使得人们摆脱具体事物的细节困扰,将面对的问题简单化。

忽略特征有两种方式,一种是在水平层次上忽略一些特征,多用于对复杂事物的简化。例如,一本书具有标题、作者、内容、写作时间、印刷时间、销售时间、销售商、购买者、价格、纸张质料等等非常多的特征,在开发图书管理系统时就可以将这些特性中的大部分忽略,仅仅保留标题、作者、内容等有限的重要特征。

忽略的第二种方式是在特征的垂直表达层次上忽略底层的细节,在一个更高的层次上分析事物,多用于归纳众多事物的共性。例如,对两个学生张三和李四,在底层细节上,他们的姓名属性分别取值为张三和李四,抽象的方法会忽略这些底层细节的差异,在它们都具有姓名的属性这个更高的层次上归纳出它们都有一个“姓名”属性的共性。

对象就是对现实世界事物的抽象结果,它表达了系统所需要的现实世界事物特征,抛弃了那些系统不需要的特征。类则是对象集的抽象结果,它忽略了具体某个对象在特定时间和空间的细节状态,从对象集的全局出发,在一个更高的逻辑层次上,描述了对对象集的共性。后面将要介绍的类的继承关系也是一种抽象,在类集合中寻找共性的抽象。

3.类的封装

需要特别指出的,类并不是简单的将属性和行为放置在一起,它还需要进行信息的隐藏 (Information Hiding), 又称封装 (Encapsulation)。封装是指尽可能隐藏构造单位内部的实现细节,只通过有限的外接口保持对外联系的一种软件构造策略。

类的封装要求类只对外公开其对象为履行职责所必需的协议 (Protocol), 其他的实现细节都要隐藏起来,包括不允许外界直接访问的属性和内部使用的局部方法。

因为类只通过封装后的协议与外界交互,所以类之间是平等的,不存在一个类控制其他类的现象

[Booch1994]。如果一个类需要使用其他类的功能，那么就按照对方的协议，向其发送消息请求，并从对方的响应消息中得到结果。

14.3.4. 类之间的关系

1. 关联

类之间的关系被称为关联（Association），它指出了类之间的某种语义联系。关联是类对其对象实例之间的无数潜在关系的描述，对象实例依据关联所带有的信息进行链接的建立和撤销。如果两个类之间没有关联，那么两个类的对象实例之间就不存在链接，就无法实现直接的协作。如果所有的类之间都没有关联，那么就只剩下一些不能一起工作的孤立的类。

对象模型是以实体关系模型为基础发展出来的（OMT是对ERD的面向对象改进，而UML的对象模型的主要思想又来自于OMT），所以对类之间关系的处理和对实体之间关系的处理存在很多的相似性（如图14-6所示）：（1）为关联赋予名称，表示关系的语义内涵；（2）将参与关联的类的数量表示为度数，用来度量关联的复杂度，常见的有一元关联、二元关联和N元关联；（3）使用最大基数和最小基数来明确一个类在关联中的参与情况；（4）如果一个关联同时具有了独立的状态和行为，那么这个关联就可以被看作是类，被称为关联类。

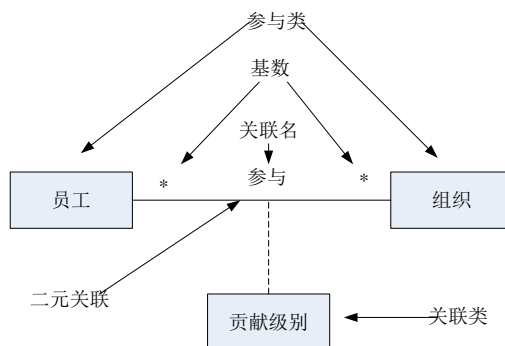


图14-6 关联的描述

除了上述的特性之外，对象模型还描述了关联的其他特性（如图14-7所示）：

（1）角色（Role）。在类参与关联关系时，依据类在关系中扮演的角色为关联端（Association End）赋予角色。关联端是类与关联的连接点，它定义了关联中类的参与行为。关联端的角色是可选的，可以标记出来，也可以不予指定。

（2）可见性（Visibility）。除了角色之外，在关联端上还可以标记可见性信息，用来说明某个关联端的类是否对关联中的其他类可见，也就是用来说明其他类的对象实例能否通过关联的链接实例访问关联端类的对象实例。可见性也是可选的，可以标记出来，也可以不标记出来，而且通常很少被标记出来。

（3）方向。对象实例之间的链接是单向的，但是类之间的关联通常是双向的。不过也可以将关联标记为单向的，可以为关联赋予方向。在分析阶段，通常不标记关联的方向。

（4）限定关联（Qualified Association）。在关联当中，一些类可以通过为关联端指定一些属性来将远端关联的类实例区分开来，这种关联被称为限定关联。

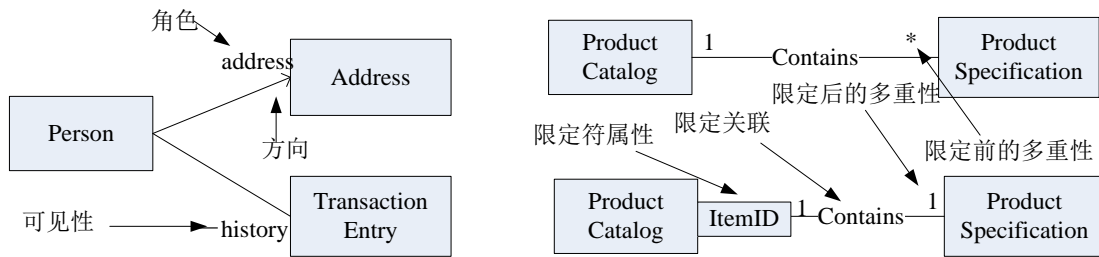


图14-7 关联的其他特性描述

2. 聚合与组合

类之间有一种特殊的关联被称为聚合（Aggregation），表示部分与整体之间的关系，如图14-8左边所示。如果整体除了包含部分之外，还对部分有完全的管理职责，即一旦一个部分属于某个整体，那么该部分就无法同时属于其他整体，也无法单独存在，则这种聚合关联被称为组合（Composition），如图14-8右边所示。

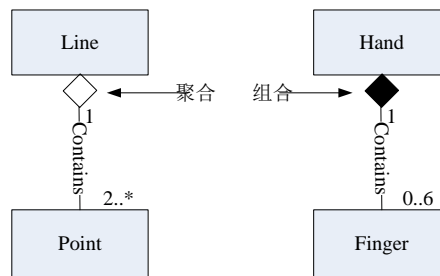


图14-8 聚合与组合示例

3. 继承

除了关联和聚合之外，类之间还有一种比较基本的关系，被称为继承（Inherit）。如果一个类A继承了类B，那么A就自然具有B的全部属性和服务，同时A也会拥有一些自己特有的属性和服务，这些特有部分是B所不具备的。其中，A被称为子类，B被称为父类（或者超类）。在继承关系当中，可以认为子类特化了父类，或者说父类是子类的泛化，所以继承关系又被称为泛化（Generalization）关系。

要更好、更准确的理解继承关系的含义，就不得不提到继承关系的起源。面向对象的继承概念来源于人工智能AI（Artificial Intelligence）领域。假设有下面两组条件Pc1和Pc2：

$Pc1 = F1(.) \wedge F2(.), Pc2 = F1(.) \wedge F2(.) \wedge F3(.)$ ，则如果Pc1能够满足规则Q，即 $Pc1 \rightarrow Q$ ，那么Pc2也就能够满足规则 $Pc2 \rightarrow Q$ 。Pc1和Pc2的关系就可以被认为是继承关系，Pc2继承了Pc1。从这里可以看出，如果一个子类继承了父类，那么子类就应该拥有父类所有的元素，同时额外追加自己特有的元素。除了子类要包含父类的元素之外，通过分析继承关系的起源，还可以发现继承关系另一层更加重要的语义含义，即子类要能够完成父类的工作，履行父类的职责。

对元素的继承是一种静态的结构关系，对职责的继承是一种动态的语义关系。实现静态继承关系的子类在结构上继承父类，被称为子类（subclass），静态继承关系被称为子类化（subclassing）关系。实现动态继承关系的子类在职责上继承父类，被称为子类型（subtype），动态继承关系被称为子类型化（subtyping）关系。同时实现静态和动态的继承是面向对象继承关系的理想情况。但是面向对象使用的是静态的继承实现技术，所以常常会出现仅仅实现静态继承的情况。这是一种不完整的继承关系，会带来很多的负面问题。

这里需要强调指出的是，不要仅仅为了实现结构上的复用效果而进行继承，要保证父类和子类之间职责上的继承。例如，在图14-9 a)中，B继承了A的结构，但是因为B更改了A的print协议，所以B没有在职务上继承A，此时B仅仅是对A的静态继承。静态继承可以实现代码的重用，但是因为B的print协议具有和A不同的语义，所以如果出现了按照A的语义应用B的print协议的情况，就会带来负面影响。在图14-9 b)中，A和B具有不同的结构，但是它们的职责是相似的，所以它们虽然没有子类关系，却可以是子类型关系。图14-9 c)中的A和B是对象模型中理想的继承关系，B同时实现了对A的静态继承和动态继承。

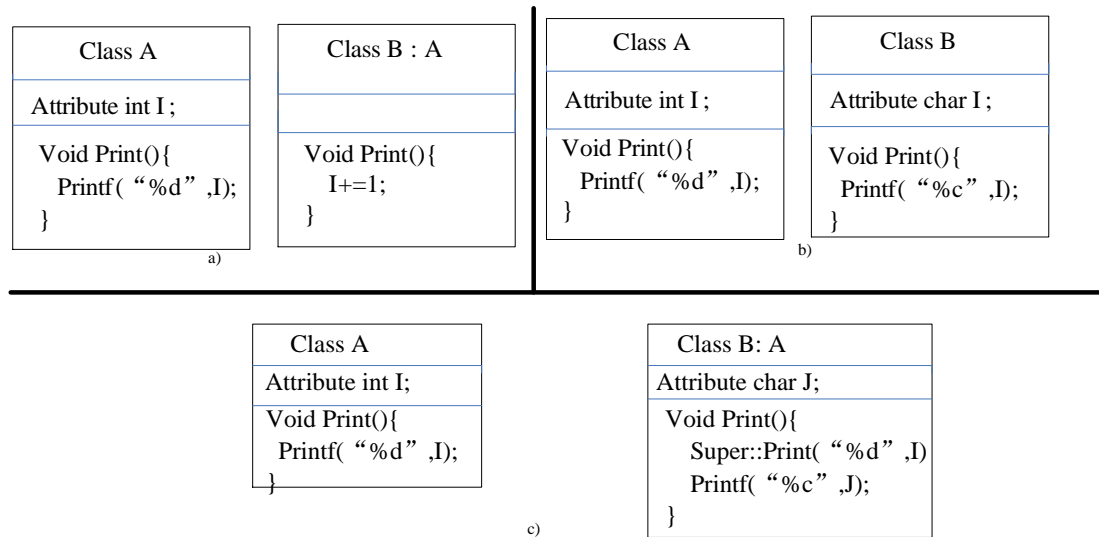


图14-9 静态继承和动态继承示例

继承关系的图示如图14-10所示，它被描述为带有三角形箭头的实线。

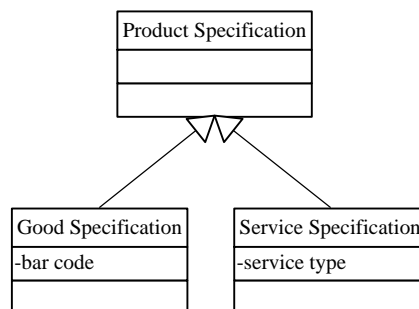


图14-10 继承关系图示

4 多态

多态 (Polymorphism) 是面向对象方法中对象模型的重要概念。多态有广义和狭义之分，对广义多态的理解需要分离对象的实现和它在一定情景下的表现形态。这里的实现是指对象具有的表现能力，它使得对象有能力表现出各种可能的行为。表现形态是指对象在某一特定情况下表现出来的行为。因为受到使用情景的限制和影响，所以对象表现出来的行为可能仅仅是其能力的一个部分。也就是说，表现形态只是实现的一种可能展示。

一个对象在不同情景下表现出不同形态，或者不同对象在相同情景中表现出相同形态的现象被称作广义

上的多态。

广义多态的第一种类型是重载与泛型，是同一对象在不同情景中表现出不同形态的现象的抽象。例如，同一块显卡，可以配合不同的主板配置，表现出不同的效果。面向对象方法对该种类型的多态有两种实现方式：一是依据参数或返回值的不同为协议定义不同的版本，每一个版本都可以表现出自己独特的行为。二是使用同一个通用的实现处理不同的数据类型，然后根据数据类型的不同表现出不同的行为。第一种实现方式被称为重载 (Overloading)。第二种实现方式被称为泛型 (Generality)，也被称为类定义的模版 (Template) 机制。

广义多态的第二种类型是狭义多态，是对多个对象在同一情景中表现出相同形态的现象的抽象。例如，不同的显卡，可以在同一套机器配置中实现相同的显示效果。这种多态也是狭义上的多态概念。面向对象方法使用继承作为狭义多态的实现方式。

14.3.5. 分析对象模型——领域模型

“领域” (Domain) 一词有四种常见的含义 [Schmid2000]：①业务范围；②问题的集合；③应用的集合；④有共同术语的知识范围。此处的“领域”一词是指软件系统所处的问题域和业务范围。也就是说，在进行系统分析时，开发人员关注的仅仅是实际的业务范围，分析阶段产生的对象模型是关注用户问题域的对象模型，它被称为领域模型 (Domain Model)，又被称为领域类图 (Domain Class Diagram)、概念类图 (Concept Class Diagram) 或分析类图 (Analysis Class Diagram)。

领域模型中的类大多是概念类 (Concept Class)，是一个能够代表现实世界事物的概念，来自于对问题域的观察。概念类也被称为领域对象 (Domain Object)。概念类之间存在指明语义联系的关联，这些关联通常不标记方向，也不标记关联端的可见性。概念类会显式的描述自己的一些重要属性，但不是全部详细属性，而且概念类的属性通常没有类型的约束。概念类不显式的标记类的行为，即概念类不包含明确的方法。

一个领域模型的示例如图14-11所示。领域模型和完整类图的比较如图14-12所示。

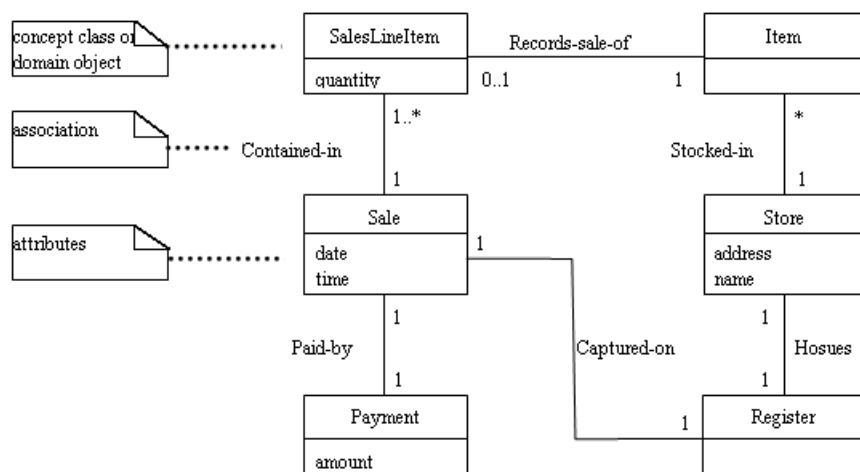


图14-11 领域模型示例

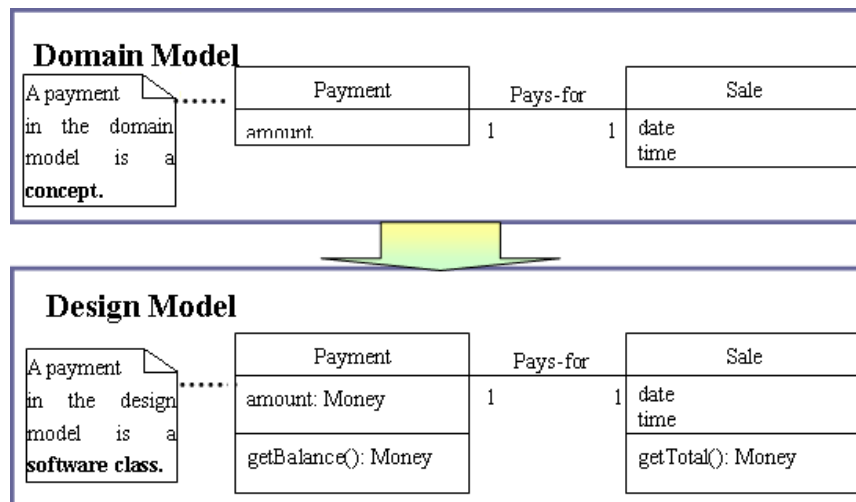


图14-12 领域模型和完整类图的比较示例

14.4. 建立领域模型

领域模型的建立过程如图14-13所示，包括四个步骤。



图14-13 领域模型建立过程

14.4.1. 识别候选对象与类

之所以使用“识别”一词而不是“建立”，是因为面向对象分析方法认为对象与类是本来就存在于应用领域之中的，分析人员的任务是找到已经存在的对象与类，而不是自由地建立需要的对象与类（像在结构化方法中建立过程那样）。

识别候选对象与类的基本思路是分析应用领域的描述信息，从中发现可能的对象与类。发现对象和类的方法主要有三种：使用概念类分类列表、名词分析和行为分析。

1. 概念类分类列表方法

这种方法事先给出一个概念类的分类列表，然后由分析人员在需求信息当中寻找相应类别的候选对象，最后对候选对象进行确定和归纳，形成概念类。

一些常见的概念类分类方式如表14-1所示。

表14-1 常见的概念类分类

方式来源	Shlaer-Mellor[Shlaer1988]	Ross[Ross1987]	Coad-Yourdon[Coad1990]
分类列表	有形的事物 角色 事件 交互功能	人 地点 事物 组织：集合体 概念	结构 其他系统 设备 事件：需要被记录 角色

表14-1 常见的概念类分类

方式来源	Shlaer-Mellor[Shlaer1988]	Ross[Ross1987]	Coad-Yourdon[Coad1990]
		事件：需要被记录	地点 组织单位

例如，图14-14 a)是一个CD商店网上购物系统的购物用例的简单描述。根据概念分类列表，可以发现候选对象如图14-14 b)所示。

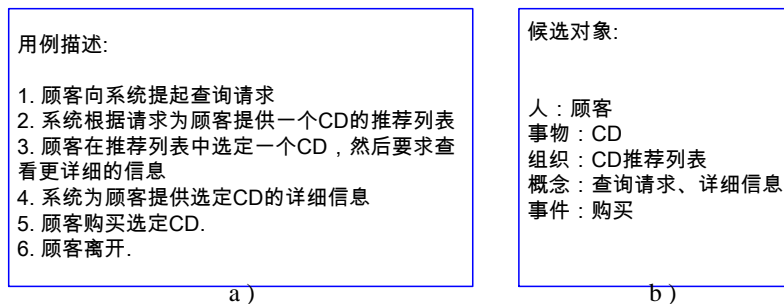


图14-14 利用概念类分类列表发现对象和类的示例

2. 名词分析

名词分析是一种运用语言分析的实用方法。名词分析从文本描述中识别出有关的名词和名词短语，然后将它们作为候选的对象，最后对候选对象进行确定和归纳，形成概念类。

例如，一个商店销售处理的用例简单描述如图14-15所示。对其进行语言分析，识别其中的名词和名词短语，并使用下划线进行标记。识别时，重复出现的名词和名词短语只需要标记一次即可。这些标记出来的名词和名词短语就成为候选对象。

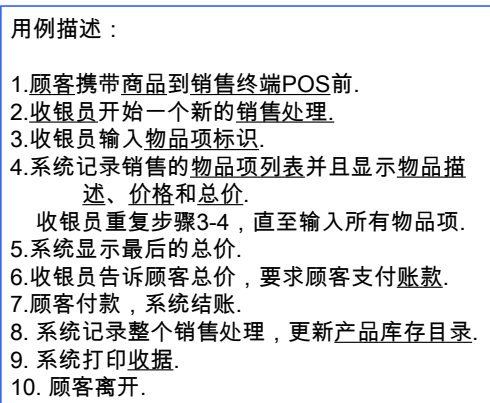


图14-15 利用名词分析发现对象和类的示例

3. 行为分析

和名词分析不同的是，行为分析是从需求描述中搜寻动词，识别出系统行为。然后找出系统行为的主动对象和被动对象作为候选对象。

一个利用行为分析发现对象和类的示例如表14-2所示。

表14-2 利用行为分析发现对象和类的示例

用例描述	行为	候选对象	
		主动对象	被动对象
1. 用户在第i层按下向上的楼层按钮	按下	用户	第i层向上楼层按钮
2. 第i层的向上按钮灯亮	亮	第i层的向上楼层按钮灯	
3. 电梯到达第i层；	到达	电梯	第i层
4. 第i层的向上楼层按钮灯灭	灭	第i层的向上楼层按钮灯	
5. 电梯门开启	开启	电梯门	
6. 计时器开始计时	计时	计时器	
7. 用户进入电梯	无		
8. 用户按下到j层的电梯按钮	按下	用户	到第j层的电梯按钮
9. 到第j层的电梯按钮灯亮	亮	到第j层的电梯按钮灯	
10. 计时时间到，电梯门关闭	到时	计时器	
	关闭	电梯门	
11. 电梯到达第j层	到达		
12. 到达j层的电梯按钮灯灭	灭	到第j层的电梯按钮灯	
13. 电梯门开启	开启	电梯门	
14. 计时器开始计时	计时	计时器	
15. 用户走出电梯门	无		
16. 计时时间到，电梯门关闭	到时	电梯门	
	关闭	计时器	

表14-5的第1列是电梯控制系统的一个用例的描述片断。

第2列是从用例描述中发现的行为。其中，用例的第7行和第15行所说明的行为（用户的进入和走出电梯）是与系统无关的行为，它既不会影响系统也不会受到系统的影响，所以不属于系统行为。

表的第3列（“候选对象”）是根据行为的主动对象和被动对象初步选择的候选对象。

4.方法比较

上述的三种方法各有优缺点和适用场景。

使用概念类分类列表和名词分析又被称为经典方法。之所以称这些方法为经典方法，是因为它们主要来源于经典分类理论，产生的概念类容易出现数据驱动缺陷。相比之下，行为分析就是以概念聚类为基础的职责驱动的方法，它适用于描述复杂协同系统和控制系统。

概念类类别比较依赖于分析人员的个人技能，名词分析和行为比较依赖于用例文本的写法。实践中，[Abbott1983]提出的名词分析方法因为容易使用和有效得到了较为广泛的应用，它能够帮助解决常见情况下的候选类识别工作，虽然它不够严谨，因为名词比较依赖于用例文本的写法，但总体效果受到的影响不大[Booch2007]。

三种方法的共同缺点是需要以明确的需求描述为前提,如果遇到复杂和有较多不确定性的系统,难以建立明确需求描述的情况下,这三种方法就会碰到困难,这时可以使用CRC方法(本章后面有描述)。CRC方法能够处理复杂情景,但应用起来较为困难,及其依赖于分析人员自身的技能和经验。一个不熟练的分析人员是很难应用好CRC方法的。

4.类的归纳

按照对象模型的概念定义,类是对象的归纳。那么在识别候选对象和类时,是否也需要先识别对象再归纳类呢?实际工作不是这样的,因为人们在观察和理解事物时很自然的就会带有分类的观点,所以在寻找对象时并不是搜寻孤立的对象,而是带着类型的观点搜寻和外界存在一定联系的对象类型,也就说寻找到的每个对象通常都能代表一个概念类。例如,在看待一个人时,人们不会仅仅停留在对他本人的观察上,而是会自然的将这个人跟很多社会角色(学生、顾客等)联系起来,赋予其一种类型。只有在少数的特殊情况下需要按照分类法对对象进行分类处理,以产生概念类。例如,在图书管理系统当中,如果确定了“学生”对象和“教师”对象,在二者没有被图书管理系统区别对待的情况下,这两个对象应该被合并为一个概念类“读者”。

需要说明的是,这并不意味着类的分类思想就是没用的,因为分析人员在下意识进行归类时,仍然要遵循职责驱动的分类思路。

14.4.2. 确定概念类

1.确定概念类的准则

选定了候选类之后,还需确定其是否应该作为一个概念类存在。判断的标准就是在应用背景(尤其是需求要求)下考察候选类的内涵,看其是否同时具有状态和行为特征:

- 如果候选类既维持一定的状态,又依据状态表现一定的行为,那么它就应该是一个独立存在的概念类。例如,假设在销售系统当中有两个候选类“商品”和“价格”。其中“商品”需要维护商品的描述信息,也即同时具有状态(描述信息)和行为(为查询提供描述信息),所以它应该是一个独立存在的概念类。“价格”拥有一个状态(即价格的取值),但是它没有行为,因为人们查询商品价格的时候是向“商品”发出请求而不是询问“价格”,所以它不是独立存在的概念类。

- 如果候选类只有状态没有行为,那么就要分析它的状态是否是系统需要的数据。如果系统需要它的状态数据,那么该候选类就应该作为其他类的属性出现在最终的领域模型当中。否则,该候选类应该被摒弃。上一段例子中的“价格”就属于只有状态没有行为的候选类,而且系统需要它的状态数据,所以它最终会被抽象为“商品”类的一个属性。

- 如果候选类只有行为没有状态,那么往往意味着需求信息的遗漏,因为行为的表现总是要以状态为基础的。此时就需要重新进行需求的获取。例如,假设在销售系统当中出现了一个候选类,它仅仅包含打印收据行为却不维护任何的状态,那么很容易就可以发现关于收据内容的信息在执行需求获取时被遗漏了。

- 既没有状态也没有行为的候选类很少会出现,即使出现也可以很容易地做出将其摒弃的决定。

- 在判断一个候选类是否应该成为独立类时一定要紧密结合类的含义。尤其是要避免实体关系建模思想所带来的误区。

■ 实体关系建模思想所带来的误区之一是属性的复杂度问题。在进行实体关系建模时，因为实体往往需要以二维关系表的形式表达出来，因此人们就会使用“二维”形式来限制实体的属性。例如，在实体出现组合属性或者多值属性时，这些属性就会超出二维的限制，以至于无法被实现为一个对应的关系表，就常常会被独立为单独的实体。其实这种二维的限制本来就不该出现在实体关系建模当中的，它应该是数据设计需要考虑的内容，而不是数据分析。在面向对象分析当中，这种限制就更不应该存在了。例如，在图书管理系统当中，在描述“图书”类时，需要“作者”的信息。假设“作者”的信息比较复杂，包括姓名、职业、联系方式等很多内容，而且这些信息仅仅是被用来描述图书。按照“二维”形式的实体关系建模思想，具有复杂结构的“作者”自然应该是独立的实体。但是如果将“作者”作为候选类进行考察，那么依据类的含义进行判断，“作者”具有系统需要的状态数据但是没有独立的行为，它就自然应该被抽象为“图书”类的属性，是“图书”类的一个复杂属性。

■ 实体关系建模思想所带来的误区之二是人们易于武断的将单值状态类抽象为其他类的属性。例如，在前面考察销售系统当中的“商品”和“价格”两个候选类的例子当中，很多人会因为“价格”仅仅包含一个属性（价格：Integer）而武断地将其作为“商品”的属性。但是在特定的情况下，假设商店实行的是依据外界环境变化的浮动价格（例如分时段、分顾客等级），那么“价格”就具有了独立的行为。再对“价格”进行考察时，它就应该是一个独立的存在类。除了“价格”这样带有数量性质的单值状态类之外，标识符也是常常被错误处理的单值状态类。例如，“商品”都有“条形码”作为标识符，但是在“条件码”具有独立行为时（例如合法性验证），“条形码”就应该被看作独立的类而非“商品”的属性。将上述情况下的“价格”和“条形码”建模为独立类的好处可以在[Fowler1996, Chapter3 & Chapter 5]中找到充足的证据。

2. 示例

例如，对图14-14中所描述的候选对象，按照上述准则进行判定：“详细信息”应该是“CD”的属性而非独立的对象，因为它只有系统需要的状态没有独立的行为。所以，对候选类进行处理后就可以得到如图14-16 c)所示的概念类。每一个确定后的对象都被归纳为一个独立的概念类。

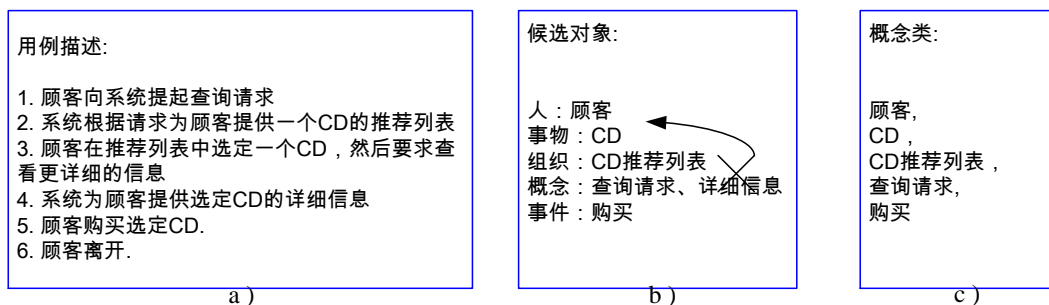


图14-16 从候选类中确定概念类示例一

再例如，对图14-15所描述的候选类，进行概念类确定可以发现：“物品项标识”、“价格”、“总价”和“收据”四个候选类不符合条件，需要被摒弃。当然，系统设定“物品项标识”和“价格”不存在合法性验证和浮动价格这样的特殊行为。而且，系统还设定它不需要保存和后续处理打印出来的收据，这样收据就是由系统产生但在系统之外存在的事物，属于既无状态又无行为的应摒弃对象。最后确定的概念类如图14-17 c)所示。

<p>用例描述：</p> <ol style="list-style-type: none"> 1. 顾客携带商品到销售终端POS前. 2. 收银员开始一个新的销售处理. 3. 收银员输入物品项标识. 4. 系统记录销售的物品项列表并且显示物品描述、价格和总价. <p>收银员重复步骤3-4，直至输入所有物品项.</p> <ol style="list-style-type: none"> 5. 系统显示最后的总价. 6. 收银员告诉顾客总价，要求顾客支付账款. 7. 顾客付款，系统结账. 8. 系统记录整个销售处理，更新产品库存目录. 9. 系统打印收据. 10. 顾客离开. 	<p>确定类：</p> <p>顾客，商品，POS，收银员，销售处理，物品项列表，物品描述，账款，产品目录</p> <p>摒弃类：</p> <p>物品项标识: 只有状态没有行为 价格: 只有状态没有行为 总价: 只有状态没有行为 收据: 既无状态也无行为</p>	<p>概念类：</p> <p>顾客，商品，POS，收银员，销售处理，物品项列表，物品描述，账款，产品目录</p>
a)	b)	c)

图14-17 从候选类中确定概念类示例二

再例如，分析表14-2中的候选类可以确定如图14-18所示概念类：

<p>保留对象：</p> <p>第i层的向上楼层按钮</p> <p>第i层的向上楼层按钮灯</p> <p>电梯</p> <p>电梯门</p> <p>计时器</p> <p>到第j层的电梯按钮</p> <p>到第j层的电梯按钮灯</p>	<p>最后的概念类：</p> <p>楼层按钮</p> <p>电梯按钮</p> <p>按钮灯</p> <p>电梯</p> <p>电梯门</p> <p>计时器</p>
<p>摒弃对象：</p> <p>用户：系统外对象，既没有状态也没有行为；</p> <p>第i层、第j层：只有状态没有行为</p>	

图14-18 从候选类中确定概念类示例三

- 候选类“用户”没有状态，所以在设定“按下”行为的归属时，候选对象“第i层的向上楼层按钮”和“到第j层的电梯按钮”被作为首选，这样候选类“用户”就既没有状态也没有行为。
- “第i层”与“第j层”两个候选类只有状态，没有行为，应该作为其他类的属性存在。
- 如图14-18所示的初步确定后的类还需要进行归纳。“第i层的向上楼层按钮”和“到第j层的电梯按钮”被归纳为概念类“楼层按钮”和“电梯按钮”。如果系统内同时存在多个电梯的话，“楼层按钮”和“电梯按钮”的行为会有所不同，所以可以保留这两个概念类。（当然，也可以再为“楼层按钮”和“电梯按钮”抽象出一个共同的超类“按钮”。）如果系统内只有一个可以调度的电梯，那么“楼层按钮”和“电

梯按钮”的行为会类似，就可以再将它们合并归纳为一个概念类“按钮”。电梯内所有按钮灯的行为都是类似的，所以被共同抽象为一个概念类“按钮灯”。

14.4.3. 建立类之间的关联

在得到孤立的概念类之后，要建立它们之间的关联，把它们联系起来。发现概念类之间的关联可以从两个方面着手：（1）分析问题域内的静态结构关系，发现概念类之间的整体部分关系和明显的语义联系；（2）分析概念类之间的协作，协作能够体现概念类之间明显以及不明显的语义联系。

建立类之间的关联时，要注意下列原则：

（1）保证类之间协作所必需的可见性。如果两个对象实例需要实现互相之间的协作，那么至少它们中的一个对象实例要持有另一个对象实例的链接，在保证可见性的情况下协作才能成为可能。因此，如果两个类存在协作，那么它们就应该具有能够保证可见性的关联。为保证类之间协作而建立的关联是必要关联，被称为“需要知道”（Need to Know）型关联，是对象模型必不可少的部分。

（2）适当使用问题域内的关联，增强领域模型的可理解性。有些类之间不需要互相协作，但是它们的对象实例在问题域内存在某些重要而且固定的关系。这些关系是问题域特性的必要部分，因此需要为这些关系建立关联以增强领域模型的可理解性。对问题域内关系的识别要适可而止，因为问题域内的关系是复杂和繁多的，因为它们建立太多的关联不仅不能有效的表示领域模型，反而会使得领域模型变得混乱。

（3）不要在关联的识别上花费太多的时间。识别概念类比识别关联更加重要。一方面是因为遗漏的概念类比较难以发现，而遗漏的关联则很容易在后续的处理阶段得到建立。另一方面是因为常常有些深层次的关联发现起来非常费时，但带来的好处不大。

（4）避免显示冗余和导出的关联。

发现关联后使用合适的动词短语为关联命名，描述每个关联端的角色和多重性。关联名称通常按照自上至下、自左至右的方式表达概念类之间的关系。在分析阶段，一般不会描述关联的方向和关联端的可见性。不过在非常必要的情况下（例如存在重要的约束或者某些类有特殊要求），也可以描述关联的方向和关联端的可见性。

例如，根据图14-19 a)的需求信息（其概念类如图14-16 c）），可以从协作和问题域两个方面为其发现关联如图14-19 b)所示，并建立图14-19 c)所示的领域模型。

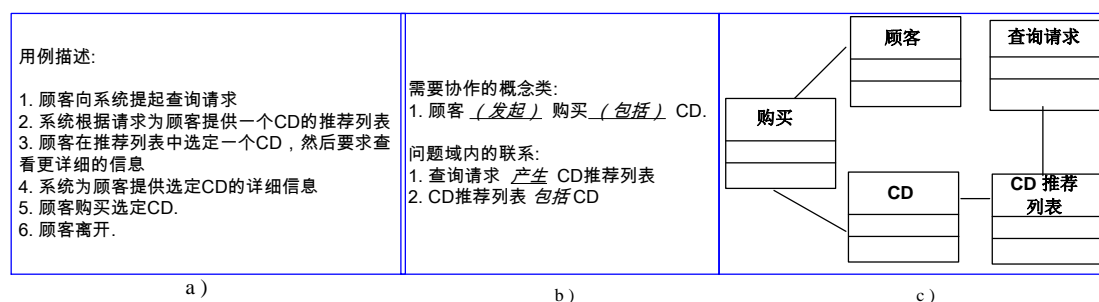


图14-19 关联建立示例

为图14-19 c)的领域模型关联添加了详细的描述之后，产生的领域模型如图14-20所示。

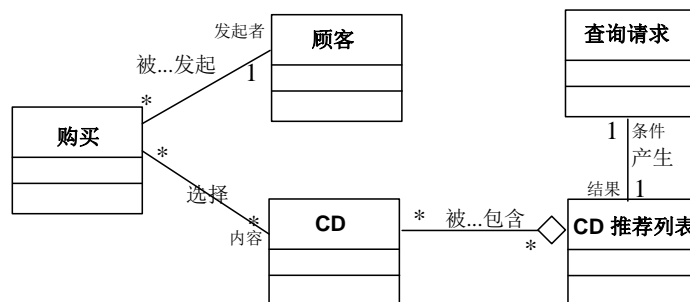


图14-20 建立关联后的领域模型示例

读者可以按照上述步骤自行行为图14-17和图14-18所描述的概念类建立关联。其参考结果分别如图14-21 a)和图14-21 b)所示。

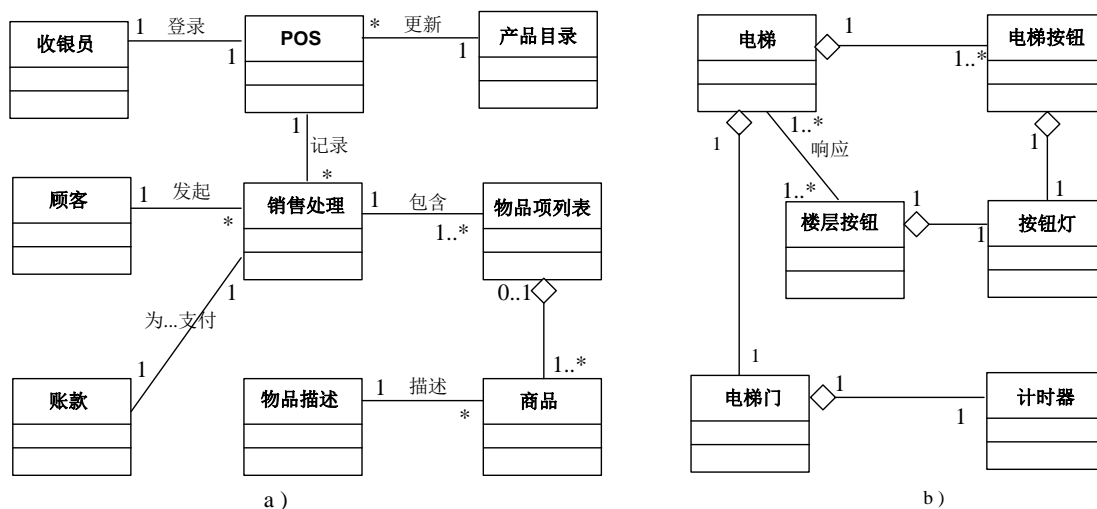


图14-21 应用的参考示例

14.4.4. 添加类的重要属性

建立领域模型的最后一个步骤是添加概念类的重要属性。这些属性往往是实现类协作时必要的信息，是协作的条件、输入、结果或者过程记录。在用例的描述当中可以发现关于属性的信息。在需求获取阶段获取的硬数据更是在添加类属性时非常重要的资料。

在分析阶段，建模的首要目标是理解现实，其次才是提高软件模型的质量。所以，在添加概念类的属性时，通常遵循用户的描述方式，不进行类型和约束的严格定义。

遇到复杂的属性时，按照提高质量的考虑，应该进行简化处理。不过如果是为了理解需要，也可以先不加处理，留待设计阶段再行考虑。

例如，可以根据前面用例描述中的有限信息，为图14-20的模型添加属性，建立如图14-22所示的领域模型。

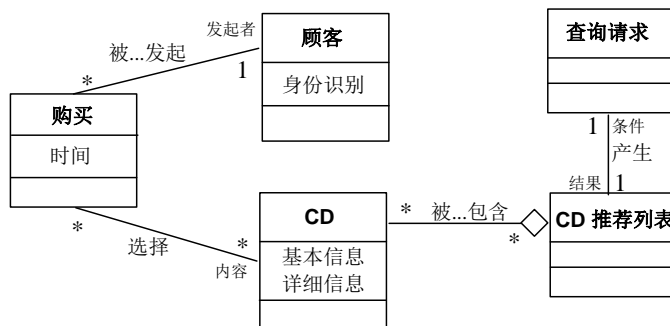


图14-22 添加了属性的领域模型示例

14.4.5. 领域模型的分析作用

建立领域模型时始终要记得需求分析的目的在于理解需求内容，发现其中的缺陷与不足，而不是简单地建立一个图形模型。

领域模型的主要作用是描述数据（参见第11章Zachman框架），所以建立领域模型的过程中最能发现的需求缺陷与不足也是在数据方面，表现为数据的定义、加工与使用。

例如，在图14-14、14-16、14-19、14-20的分析案例中，可以发现下列需求缺陷：

- 顾客是否需要注册？注册哪些信息？
- 查询请求包括哪些条件？
- CD推荐列表包含哪些信息？
- CD详细信息包含哪些信息？
- 购买行为操作哪些数据？
-

再例如，图14-15、14-17、14-21a的分析案例中，可以发现下列需求缺陷：

- 物品项标识需要标准定义吗？
- 总价的计算规则是怎样的？
- 结账操作的数据有哪些，如何操纵的？
- 产品库存目录是怎样定义的？
- 收据包含哪些信息，格式怎样？
-

分析中发现的需求缺陷有些可以通过分析其他用例来解决。例如分析“顾客注册”用例可以解决问题“顾客是否需要注册？注册哪些信息？”。一个系统中有很多用例，所有用例的领域模型合并起来，才构成完整的需求信息，所以单个用例的领域模型有缺陷是正常的，可以在其他用例的领域模型中得到解决。

有些分析中发现的需求缺陷的确是未获取过的信息或者分析中出现的错误，这时就需要根据发现的缺陷重新获取相关内容（参见第7章）或者修正错误的分析模型。

如果一个用例的领域模型分析没有发现缺陷，那么该用例的需求内容就是比较完备的，不再需要后续的获取过程。

14.5. 行为模型——交互图

14.5.1. 概述

对象需要相互协作才能完成任务。这种交互可以从两个角度进行描述，一个角度是以单个对象为中心，另一个角度则以互相协作的一组对象为中心。交互图就是以一组对象为中心的交互描述技术。

交互图用于描述在特定上下文环境中一组对象的交互行为，该上下文环境就是被实现用例的一个或多个场景。所以，交互图通常描述的是单个用例的场景。交互图中的每一个交互都描述了环境中的对象为了实现某个目标而执行的一系列消息交换。

UML的交互图又包括顺序图（Sequence Diagram）、通信图（Communication Diagram）、交互概述图（Interaction Overview Diagram）和时间图（Timing Diagram）。需求分析中常用的是顺序图和通信图。

本书将不介绍交互概述图和时间图的相关内容，感兴趣的读者请参考[UMLOMG2011]。

14.5.2. 顺序图

顺序图可以突出消息的时间顺序。一个简单的顺序图图示如图14-23所示。

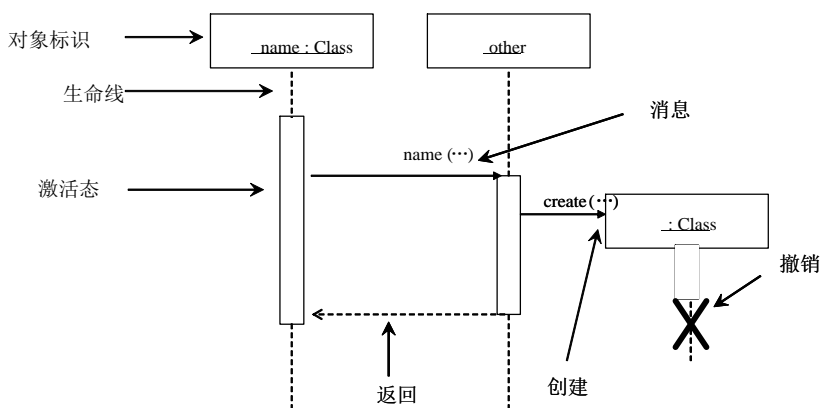


图14-23 一个简单的顺序图图示

顺序图将交互表示成一个二维图表。纵向是时间轴，时间沿竖线向下延伸。横轴表示了参与协作的对象标识。每个对象标识用一条带有头符号和竖线的垂直栏（生命线）表示。当对象存在但不工作时，标识用一条虚线表示。在对象处于工作状态时，生命线是一个双道线，表示对象处于激活状态。

顺序图显示了交互行为中的消息序列。消息用从一个对象的生命线到另一个对象的生命线的箭头来表示。箭头以时间顺序在图中从上到下排列。消息有同步、异步和返回消息之分，分别用不同的图形符号进行表示，如图14-24所示。消息箭头的标注语法为：[attribute=]name[(argument)]:[return-value]，其中 attribute 是生命线所代表对象的可选属性名称，用于保存返回值。同步消息通常需要有相应的返回消息，异步消息则不需要返回消息。

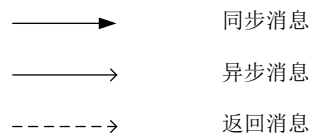


图14-24 顺序图中不同消息类型的图示

除了图14-23所示的简单图示之外，顺序图还使用了一些表达复杂情景的组合片段（Combined Fragment），其图示如图14-25所示。

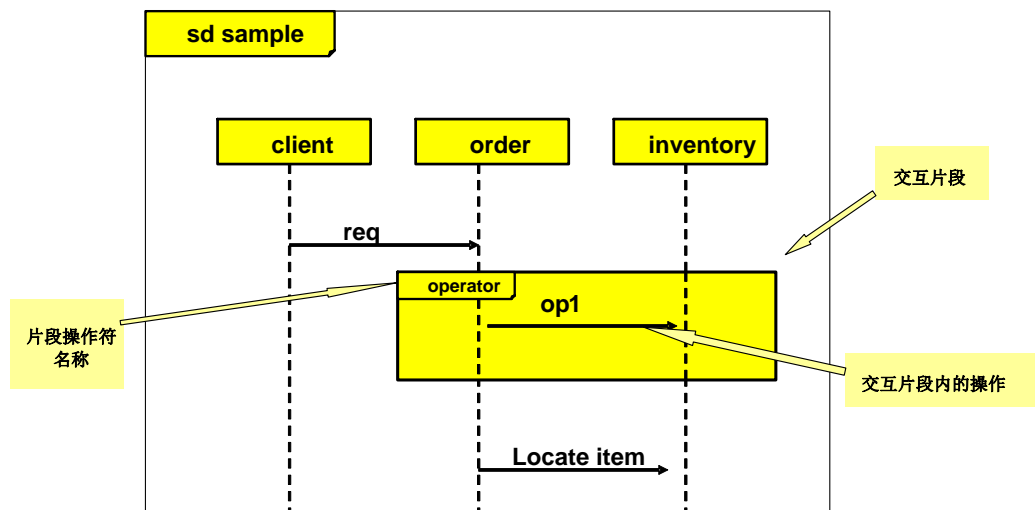


图14-25 组合片段示意图

顺序图的组合片段有：

- 选择（Alternatives）：操作符alt，如图14-26 a)所示，表示要从多个行为中根据监护条件选择一个交互行为执行。在用例中存在分支流程时，往往要使用多选一组合片段。
- 可选（Option）：操作符opt，如图14-26 b)所示。如果符合监护条件，那么片段内的交互行为就会得到执行，否则就不执行。
- 循环（Loop）：操作符loop，如图14-26 c)所示。只要监护条件为真，片段内的交互行为就会被反复执行。
- 中断（Break）：操作符break，如图14-26 d)所示。如果监护条件为真，片段内的交互行为就会被执行，并在执行后退出中断片段所在的顺序图，也就是说如果中断发生，那么顺序图中中断片段之后的交互行为就不会得到执行。
- 并行（Parallel）：操作符par，如图14-26 e)所示。并行片段内的不同交互行为可以不遵循顺序图的时间线顺序，可以并行、交织进行。
- 关键区域（Critical Region）：操作符critical，如图14-26 f)所示。区域内的交互行为是一个原子操作，一旦开始执行就必须执行完成，在执行过程中不能被打断，例如其另一个并发片段中发生了Break行为，关键区域内容的交互行为仍然要执行完成后才能退出其所在的顺序图。
- 强顺序（Strict Sequencing）：操作符strict，如图14-26 g)所示。强顺序内的交互行为必须顺序执行，例如图14-26 g)内的行为必须按照“search_google()→search_bing()→Search_yahoo()”的顺序执行。

- 弱顺序 (Weak Sequencing) : 操作符seq, 如图14-26 h) 所示。在弱顺序下: ①每个操作组内的交互行为要维持顺序关系; ②不同生命线上的不同操作组可以按照任何顺序执行; ③同一个生命线上的不同操作组要按照顺序执行。例如, 在图14-26 h) 中, Search_bing () 与 Search_yahoo () 必须先后执行, 但是 Search_goolge () 是 “Search_bing () → Search_yahoo () ” 是并发关系。

- 除了上述组合片段之外, 还有否定 (Negative)、断言 (Assertion)、忽略 (Ignore)、关注 (Consider) 三个组合片段类型, 它们更多地是用在软件设计中, 所以这里不作介绍, 需要的读者可以自行阅读 [UMLOMG2011]。

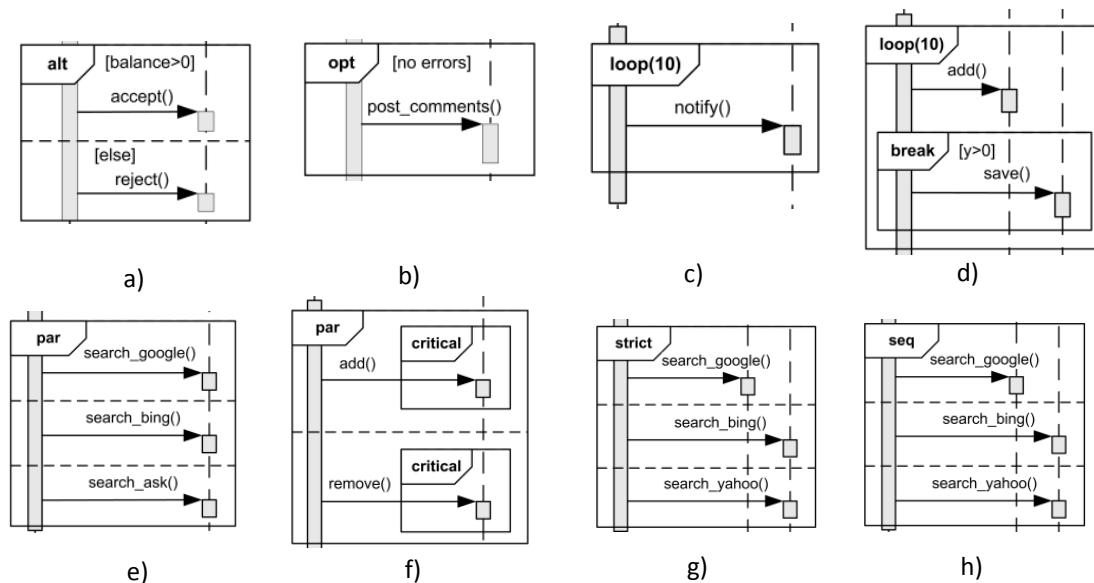


图14-26 顺序图组合片段示例

引用并不是一种组合片段, 但它使用了类似于组合片段的图示 (操作符ref), 使得一个顺序图中可以引用其他的顺序图, 从而实现将一个复杂顺序图分解为多个简单顺序图的目的, 如图14-27所示。

14.5.3. 通信图

通信图能够突出交互当中协作对象之间的关系, 在结构上它有些类似于概念类图。一个简单的通信图示例如图14-28所示。

对象标识及其连接说明了在上下文环境中可能出现的对象和对象间的链接情况。对象之间的消息用附加在连接上带标签的箭头来表示。

消息的标注为: [sequence-expression:]message, 其中message描述了消息的内容, 其格式和序列图中的消息标注相同, 即message为[attribute=]name[(argument)][:return-value]。

sequence-expression是对消息执行顺序的描述, 其格式为label [iteration-expression]。

label是整数或者名称。整数代表了消息的执行序号。例如, 消息3.1.4要发生在3.1.3之后。同时消息也能表达活动的嵌套层次, 例如消息3.1.4是在消息3.1的嵌套范围内发生的。名称代表了并发的控制线程。最后名称不同的消息在所处的嵌套层次上是并发的。例如, 3.1a和3.1b就是在3.1嵌套范围内的两个并发消息, 它们之间不分先后, 是完全平等的。

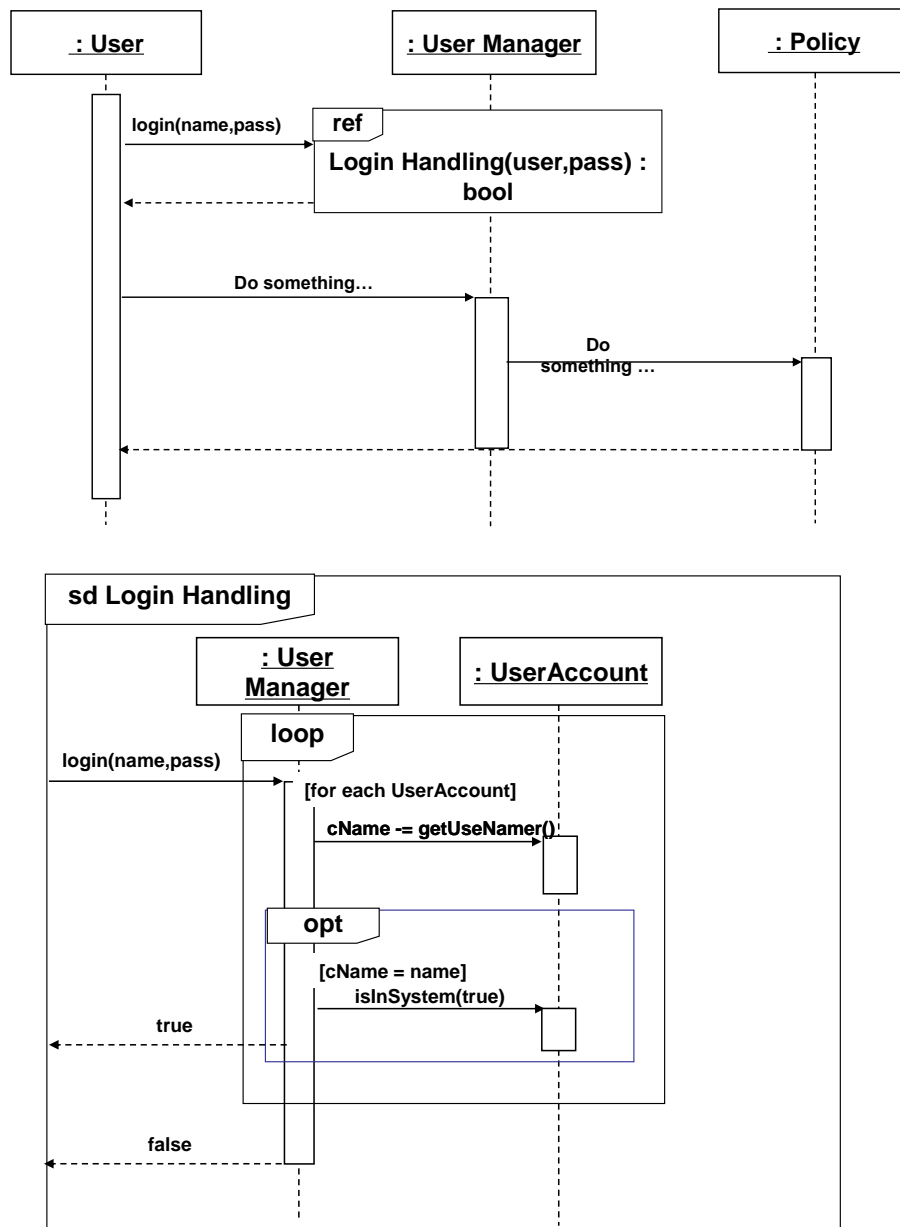


图14-27 顺序图引用示例

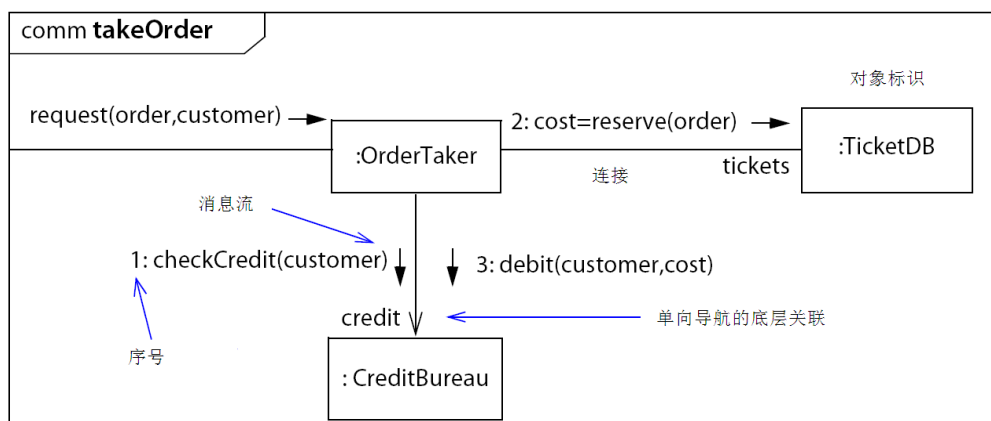


图14-28 简单的通信图示例，源自[Rumbaugh2004]

iteration-expression表示消息应该条件执行或者迭代执行，即进行0次或多次消息发送。其形式为：

*[iteration-clause]或者[condition-clause]。例如,要表示一个消息需要发送n次,则完整消息标签示例为:
 3.1 * $[i=1..n]$:update()。如果要表示一个消息需要满足条件才能执行,则完整的消息标签示例为:1b.4
 $[x<0]$:invert(x, color)。

14.5.4. 系统顺序图

顺序图和通信图从不同的侧重点对用例的典型场景进行了完全等价的实现。在实现当中,顺序图和通信图都需要和领域模型保持一致。一致性是指在交互图中出现的对象应该在领域模型中有相应的对象存在。

因为系统分析的过程是一个不断迭代和深入的过程,所以一致性的要求会使得用例的实现出现迟滞现象。因为对象的发现是无法在项目初始阶段就全部完成的,所以实现用例的详细交互图就不能确定参与协作的对象,进而无法形成实现用例的交互图。

一个更常见的做法是在分析阶段的开始开发系统顺序图,而不是直接实现在多个对象之间展开的比较详细的交互图。

系统顺序图将整个系统看作一个黑箱的对象,强调外部参与者和系统的交互行为,重点展示系统级事件。

一个系统顺序图的示例如图14-29所示。

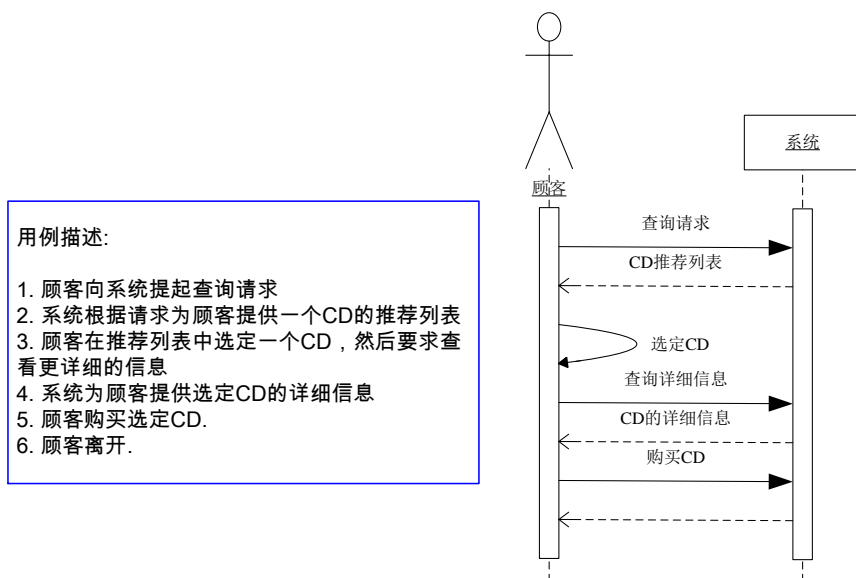


图14-29 系统顺序图示例, 左为用例的场景描述, 右为对应的系统顺序图

14.6. 建立交互图

通信图和顺序图是等价的图示, 创建的方法也类似, 下面就只展示顺序图的建立过程, 不再描述通信图的建立过程。

14.6.1. 建立典型场景的系统顺序图

最基础的交互图是用例典型场景(通常是主流程场景)的系统顺序图。面向对象需求分析中最基本的工作也是为复杂用例的典型场景建立系统顺序图。

为典型场景建立系统顺序图的一般步骤如下:

(1) 确定系统顺序图的上下文环境。系统顺序图是对用例描述中典型场景的实现，展示了场景当中发生的对象交互行为。也就是说，系统顺序图的交互是在一定的场景环境下发生的，离开这个上下文环境的限定，对交互行为的描述和理解都会出现一定的问题。因此，建立系统顺序图需要首先确定下文环境，限定描述范围。而且，上下文环境的前置条件和后置条件应该被分配给系统顺序图中的相应行为，这个工作会在为交互行为添加契约说明的时候得到实现。

(2) 找出参与交互的对象。在场景环境中寻找参与交互的对象，寻找的目标是系统、系统之外的对象和其他系统。

(3) 根据发现的对象建立交互图框架。将对象平行排列，并添加对象的生命线。

(4) 添加消息，描述交互行为。以消息的方式，将对象之间的交互行为描述出来，并建立行为之间的顺序，要注意维护对象生命线的激活状态。描述时仅仅需要考虑系统与外界的交互行为，忽略那些与系统无关的（外部对象之间的）或系统内部的交互行为。

例如，对前面所述的商品销售的用例描述，可以按照下述步骤建立系统顺序图：

(1) 确定上下文环境，以用例描述中的流程为场景环境。例子中的场景描述相对比较独立，没有对其他用例或场景的引用，因此建立系统顺序图的过程和结果也相对比较简单。

(2) 根据用例描述可以找到顾客、收银员和系统三个交互对象。仔细分析后可以发现顾客和系统之间没有直接的交互，因此可以剔除。最后发现的交互对象为收银员和系统。

(3) 按照用例描述中的流程顺序，逐步添加消息，并在进行详细信息描述后建立如图 14-30 所示的系统顺序图。

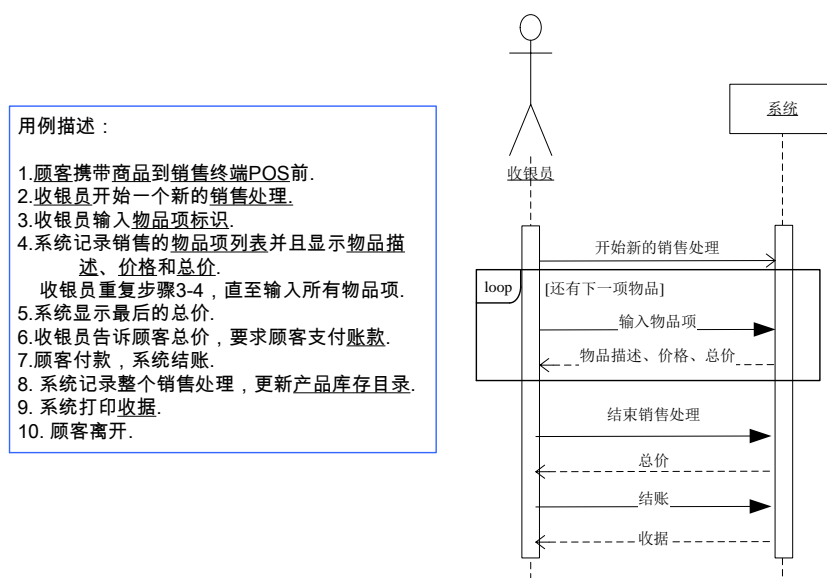


图 14-30 系统顺序图的建立示例

14.6.2. 建立用例（多场景）系统顺序图

组合片段使得顺序图能够处理复杂分支，使得顺序图能够同时描述多个场景。所以，在描述多场景用例的系统顺序图时，可以先为主流程场景建立基础的系统顺序图，然后根据分支场景与异常场景的分支点、异

常点，建立组合片段描述，从而在一个系统顺序图中描述多个场景。

例如，如图要在图14-30的系统顺序图中添加下列场景：

- 在销售开始时输入VIP会员的编号，分支点在开始一个销售处理之后，为可选场景；
- 删除一个已输入商品，分支点在输入物品项及其返回消息之间，为选择场景；
- 取消销售，分支点在开始一个销售处理之后到结账之前，为中断场景。

那么就可以为14-30的系统顺序图添加组合片段，建立如图14-31所示的系统顺序图。

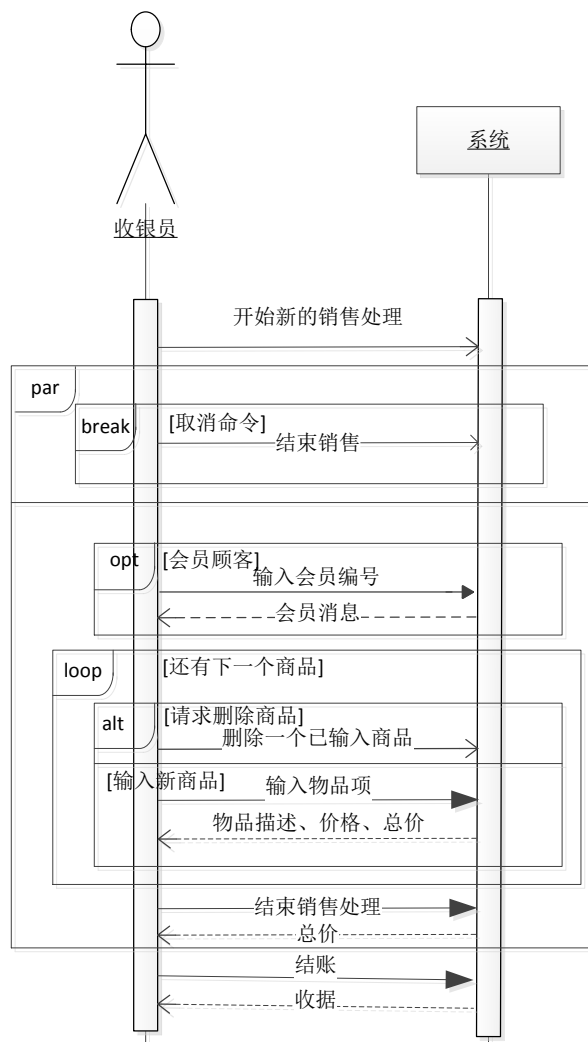


图14-31 多场景的系统顺序图示例

14.6.3. 建立详细顺序图

对简单项目进行需求分析时，建立系统顺序图描述就足够了。如果项目比较复杂，系统顺序图中的交互行为还嫌粒度太大，可以考虑适度使用详细顺序图。

建立详细顺序图的关键是正确识别参与交互的对象，这个可以借鉴领域模型的工作。一个用例的详细顺序图中参与对象应该与该用例的领域对象是一致的。

确定交互对象之后，将每一个外部交互转化为内部交互序列，就可以建立详细顺序图。详细顺序图仍然是分析模型，因为它没有添加设计因素，所以到了设计阶段还需要被设计师继续细化，添加界面、数据模

型等设计要素。

例如，对图14-30的系统顺序图，如果参考图14-21 a)所示的领域模型进行交互对象的确定，可以建立详细的顺序图如图14-32所示。

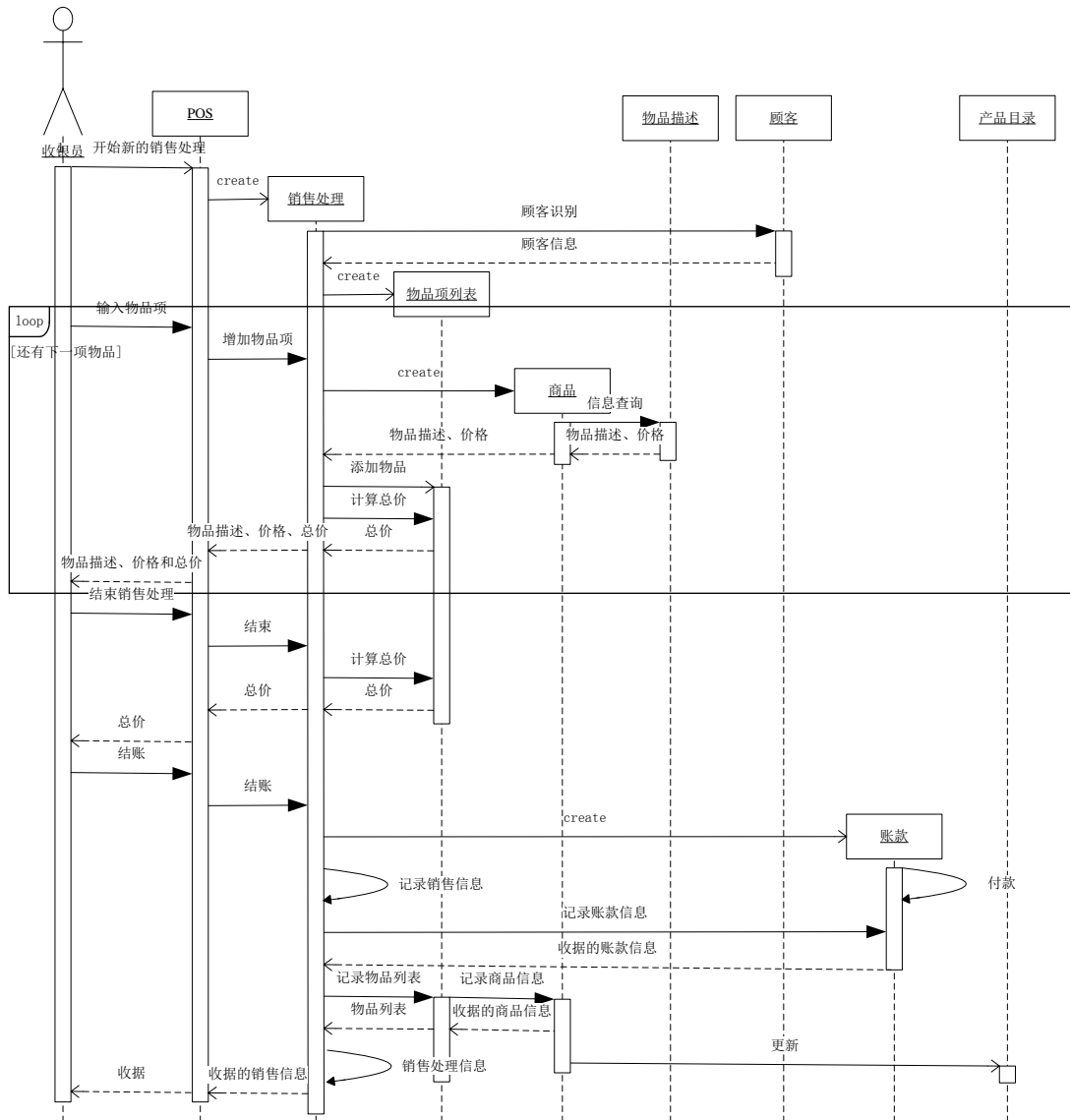


图14-32 详细的顺序图示例

14.6.4. 交互图的分析作用

建立交互图时，也可以发现其所描述需求内容中的缺陷，主要是从行为方面发现问题，例如行为缺失、行为的数据不清晰等等。

建立交互图时最常发现的问题是系统的交互行为缺失。如果外界发起了同步交互，但是系统没有给出响应（返回消息），那么就意味着响应行为缺失。如果在外界没有任何请求的情况下，系统主动给出了响应，那么就意味着请求行为的缺失。

组合片段和消息描述可能会发现行为数据的问题。如果一个交互消息的数据内容在领域模型中没有描述,就意味着其数据内容是缺失的。如果组合片段监护条件使用的数据内容在领域模型中没有描述,也意味着其数据内容的缺失。

14.7. 行为模型——状态图

14.7.1. 状态图的发展历程

状态图是以状态机理论为基础建立的对系统行为的描述手段。状态机是以“状态”概念为基础解释系统行为的一种技术,它在对系统行为的描述上得到了大量的应用。

最简单的状态机是有限状态机FSM (Finite State Machine), 它从理论上阐释了如何使用状态机模型来表示系统的行为。在各种方法的FSM应用当中,产生了状态转移图STD (State Transition Diagram)、Yourdon状态图示、SDL(Specification and Description Language)状态图示和状态转移矩阵STM(State Transition Matrix) 等多种表示法。

在大量的应用当中,人们发现了FSM的很多不足,为此, David Harel对FSM进行了发展,补充的很多模型元素,建立了状态图SC (State Chart)。Harel的方法被OMT采纳,并最终融入了UML,成为UML中的状态图SD (State Diagram)。

为了更好的理解UML中状态图的理论和应用,下面将根据它的发展历程来介绍它的重要概念。而且基于状态机的建模技术是实践和应用(尤其是控制系统和实时系统的应用)中非常重要的技术手段,所以按照发展历程来介绍状态图,这还可以帮助读者更好的理解其他相关的状态机知识。

14.7.2. 有限状态机 FSM

1.理论

状态机理论认为,系统总是处于一定的状态之中。而且,在某一时刻,系统只能处于一种状态之中。系统在任何一个状态中都是稳定的,如果没有外部事件触发,系统会一直持续维持该状态。如果发生有效的触发事件,系统将会响应事件,从一种状态转移到唯一的另一种状态。

依据上述的状态机理论,如果能够罗列出系统所有可能的状态,并发现所有有效的外部事件,那么就能够从状态转移的角度完整的表达系统的所有行为。这就是有限状态机的基本思想,它用来描述那些状态和事件数目有限的系统的行为。

FSM可以被看作是一个5元组: $FSM = (Q, \Sigma, \delta, q_0, F)$, 其中

- Q 是系统所有可能的状态集合。每个状态都记录了系统曾经发生过的行为,是过去的系统行为的累积结果。状态包含的信息要能够决定系统可能的下一步去向。
- Σ 是系统所有可能面对的触发事件。事件通常由一个刺激 (stimulus) 因素引起,并要求系统做出一定的响应 (response)。可能的事件包括三种类型: 外部事件 (external event)、内部事件 (internal event) 和时序事件 (time-based event)。外部事件是由系统外部的刺激因素引起的,例如用户的输入。内部事件的刺激因素是系统内部的数据状态满足了事先预定义的条件,例如某一数据达到了最大或最小边

界。时序事件是由系统时钟触发的事件，例如时间到了某个指定时刻或者累积时间达到了限值。

- δ 是状态转移函数。它决定了在某个状态下，发生触发事件时，系统将转移到哪一个后续状态。 $\delta: Q \times \Sigma \rightarrow Q$

- $q_0 \in Q$ ，是系统的初始状态，也就是系统一开始时所处的状态。

- $F \subseteq Q$ ，是系统可能的结束状态的集合。在持续的行为当中，一旦系统转移到了某个结束状态，系统就将结束自己的行为，否则会继续执行自己的行为。

FSM的5元组为FSM建立了完善的数学基础。所以FSM是在行为描述的诸多技术手段当中少数的形式化技术之一。这样，FSM就在易于理解和易于使用之余还具备了严谨性，这也正是FSM以及它的后续技术得到广泛应用的重要原因。

2. 图示

为了更好的利用FSM描述系统的行为，人们为它定义了图形表示法，将FSM用直观的图形符号描述出来。例如，图14-33就是一个简单的FSM图示，它说明了一个电灯系统（灯泡和开关）的系统行为。

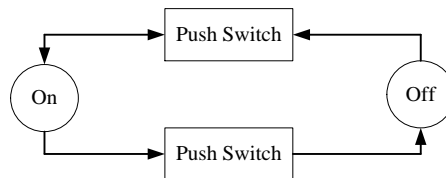


图14-33 FSM图示示例

状态转移图STD是最为常见的FSM表示法，如图14-34所示。其中，不确定性是指状态的转移情况不确定，需要依据一些其他的数据才能做出决策，即常说的分支选择。图14-33使用的就是STD的图形表示法。

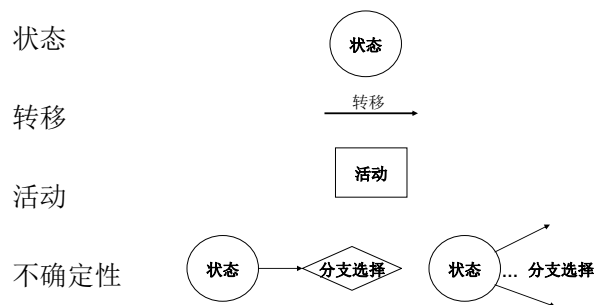


图14-34 STD表示法图示

Yourdon方法和SDL也提出了自己的FSM图示法，分别如图14-35 a) 和图14-36 a) 所示。按照它们的表示法，电灯系统的行为描述分别如图14-35 b) 和图14-36 b) 所示。

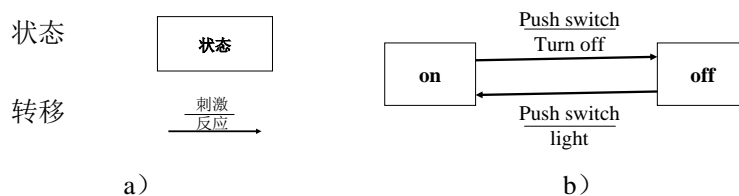


图14-35 FSM的Yourdon表示法图示

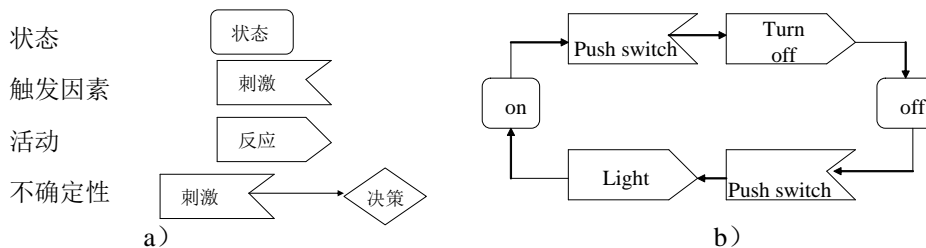


图14-36 FSM的SDL表示法图示

除了上面的几种图形表示之外，状态转移矩阵STM也常常被用来进行FSM的描述。常见的ATM取款机的STM描述如表14-3所示。STM的行表示了系统的状态列表。STM的列代表了可能的触发事件。如果第*i*行第*j*列的单元格内容为*k*，则表示在状态*i*下如果发生事件*j*则系统从状态*i*转向状态*k*。矩阵最后一列的action是特殊列，用来说明进入某个状态后要执行的活动。和表14-3等价的STD如图14-37所示。

表14-3 ATM取款机的STM描述

	卡插入	密码正确	密码错误	请求结束	输入数额	动作
1 等待插卡	2					显示 “请插卡 ”
2 等待输入密码		4	3	1		显示 “请输入密码 ”
3 等待二次输入密码		4	1	1		显示 “重试 ”
4 等待输入数额				1	1	显示 “输入数额 ”

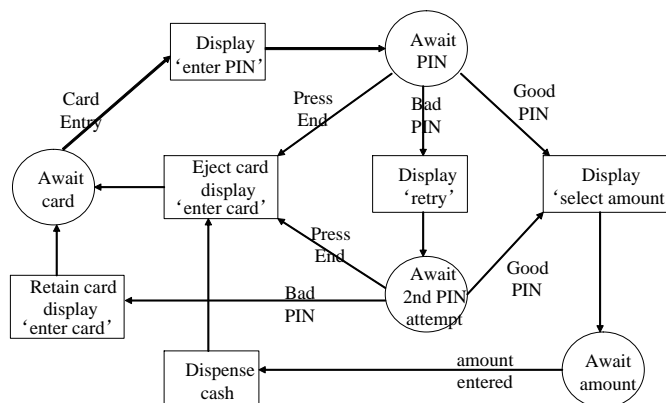


图14-37 ATM取款机的STD描述，源自[Bary2002]

14.7.3. David Harel 的发展

FSM有自己的优点，得到了广泛的应用。但是在实践当中人们也逐渐发现了FSM的局限性，尤其是它所固有的“平坦”（flat）性。任何一对状态之间都可以通过一个转移来连接，因此它不能适用于状态数量较多的系统，而这偏偏是复杂系统都可能会出现的情况。

为了解决FSM的很多局限性，David Harel对FSM进行了扩展，建立了状态图SC（State Chart）表示法。SC的FSM基本元素表示如图14-38所示。

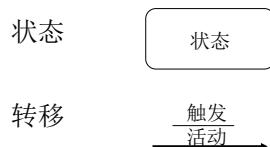


图14-38 SC的FSM图示

在FSM的基本元素之上，SC进行了6个方面的扩展。

1. 设定触发条件

状态转移的触发是有条件触发，只有在满足条件的情况下，触发才是有效的。其图示如图14-39 a)。

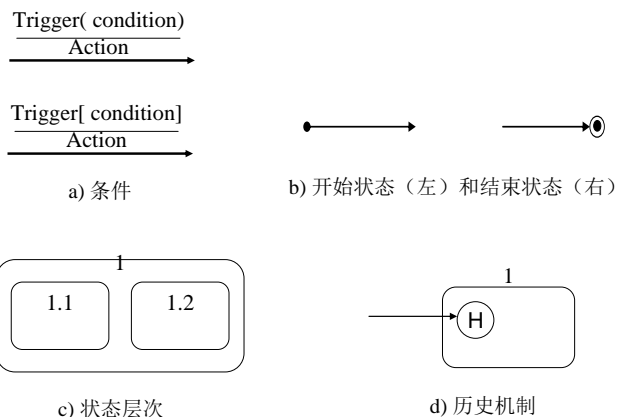


图14-39 SC对FSM的扩展

2. 引入起始状态和结束状态

起始状态表示了FSM的初始状态，结束状态表示了FSM的结束状态集。按照FSM的定义，一个系统只能有一个初始状态，但可以有多个结束状态。初始状态和结束状态的图示如图14-39 b)。

3. 构造状态层次

在系统中引入了超状态（Super-State）的概念。超状态可以包含其他状态。这样，通过将一些联系紧密的系统状态包装成超状态，就可以对系统行为进行层次式的描述，减少了FSM“平坦”性所带来的图示复杂性。状态层次的图示如图14-39 c)。

3. 使用历史机制

为了更好的构造状态层次，使用了历史机制。当转移返回到超状态中时，通常要返回到该超状态中最近经历过的子状态。其图示如图14-39 d)。

一个结合了状态层次和和历史状态的SC示意图如图14-40所示。

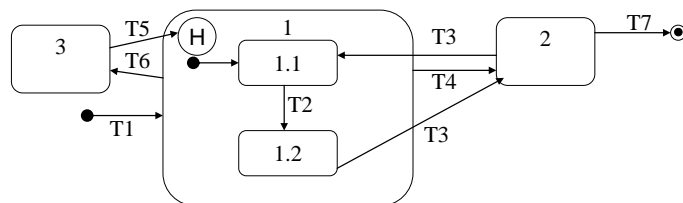


图14-40 SC示意图

该图可解释为：

- 开始触发事件 (T1) 导致进入状态 1 和子状态 1.1 ；
- 在状态 1 和子状态 1.1 时，如果发生触发事件 T2，则进入状态 1 和子状态 1.2 ；
- 在状态 1 和子状态 1.2 时，如果发生触发事件 T3，则进入状态 2 ；
- 在状态 1 (即子状态 1.1 或者 1.2) 时，如果发生触发事件 T4，则进入状态 2 ；
- 在状态 1 (即子状态 1.1 或者 1.2) 时，如果发生触发事件 T6，则进入状态 3 ；
- 在状态 2 时，如果发生触发事件 T3，则进入状态 1 和子状态 1.1 ；
- 在状态 2 时，如果发生触发事件 T7，则状态机结束运行 ；
- 在状态 3 时，如果发生触发事件 T5，则进入状态 1 和历史子状态，历史子状态是上一次离开状态 1 时的子状态 (1.1 或者 1.2) 。

5.完善内部触发和时序触发

虽然FSM在理论上将触发事件分成外部事件、内部事件和时序事件三种类型，但FSM的图示法通常认为事件来自系统外部。为了完善这一点，SC使用条件来表示内部事件，并通过在状态上添加时序限制来表示时序事件。图14-41为SC的内部事件和时序事件表示的一个示例，其中条件“(PIN Invalid)”和“(PIN Valid)”表示了两个内部事件，加在状态 2 上的限制“<10 second”表示了一个时序事件。

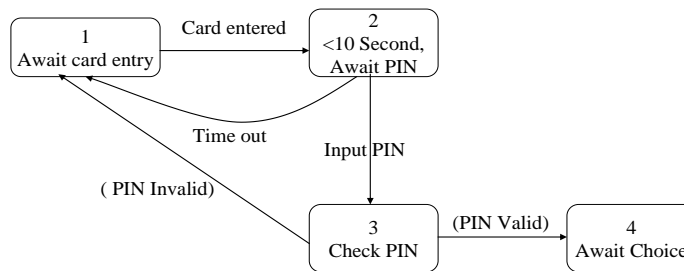


图14-41 SC的内部事件和时序事件示例，源自[Bary2002]

6.支持并发

FSM认为在任意时刻，系统都处于唯一的一种状态。这样，FSM就拒绝了系统的并发性，因为在并发行为下，系统的状态是不确定的。为了支持并发，SC建立了并发的图示机制。SC把并发表示在一个超状态中，并用虚线隔开。不同并发之间的交互局限于触发器的传递和相互之间的条件测试。图14-42展示了一个并发的SC示例。

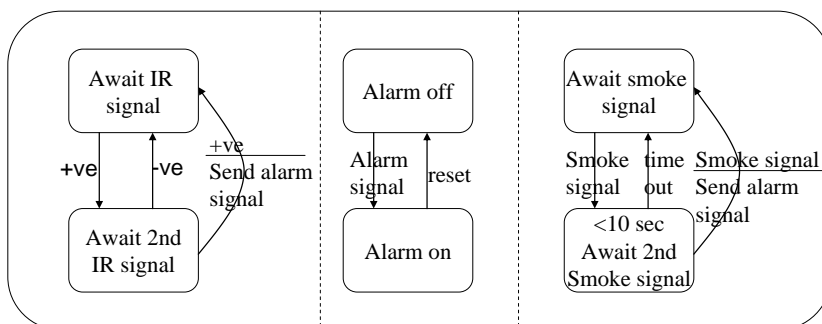


图14-42 SC的并发示例，源自[Bary2002]



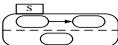




图14-42描述了一个建筑内的火警系统。系统有三个并发的部分：红外探测器IR（左边）、烟雾探测器（右边）和警报器（中间）。红外探测器信号可分为正负两种情景，两个连续的正信号将触发警报。烟

雾探测器信号只取正值，如果在10秒内收到两个信号，那么也会触发警报。警报器收到触发信号后就开始发出警报。复位信号将重置警报器。

14.7.4. UML 的状态图

UML的状态图是在SC的基础上进一步发展而成的，它的表示法如表14-4所示。

表14-4 UML的状态图表示法，源自[Rumbaugh2004]

元素	类型	说明	表示法
事件	调用事件	接收一个对象显示的同步的调用请求	op(a:T)
	变化事件	对布尔表达式值的修改	when(exp)
	信号事件	接收对象间显示的、命名的、异步的通信	sname(a:T)
	时间事件	一个绝对时间的到达或者相对时间段的逝去	after(time)
转换	入口转换	进入某一状态时执行的入口活动	entry/activity
	出口转换	离开某一状态时执行的出口活动	exit/activity
	外部转换	对事件做出的响应引起状态变化或自身转换，同时引发一个特定的效果。如果离开或进入状态也可能引起状态的入口转换或出口转换	e(a:T)[guard]/activity
	内部转换	对事件做出的响应并引发执行一个特定的效果，但是并不引起状态变化或入口转换、出口转换的执行	e(a:T)[guard]/activity
状态	简单状态	没有子结构的状态	
	初始状态	起点状态	
	结束状态	表明活动已完成	
	正交（并发）状态	被分成两个或多个区域的状态，当复合状态被激活时，每个区域中的一个直接子状态被并发激活	
	非正交（顺序）状态	包含一个或多个直接子状态，当复合状态被激活时，只有一个子状态会被激活	
	终止	终止状态机所描述的对象运行	
状态	选择	在运行至完成的转换中起动态分支的作用	
	历史状态	它被激活时会将复合状态还原成之前被激活时的状态	
	结合	将转换的片断串联成一个单一的“运行至完成”的转换	
	子机状态	引用其他状态机的连接，被引用的状态机可以在概念上替换该子机状态	
	入口点	标识出一个内部状态作为目标	
	出口点	标识出一个内部状态作为源	

一个UML的状态图的示例如图14-43所示，它描述了一个自动售票系统的行为。

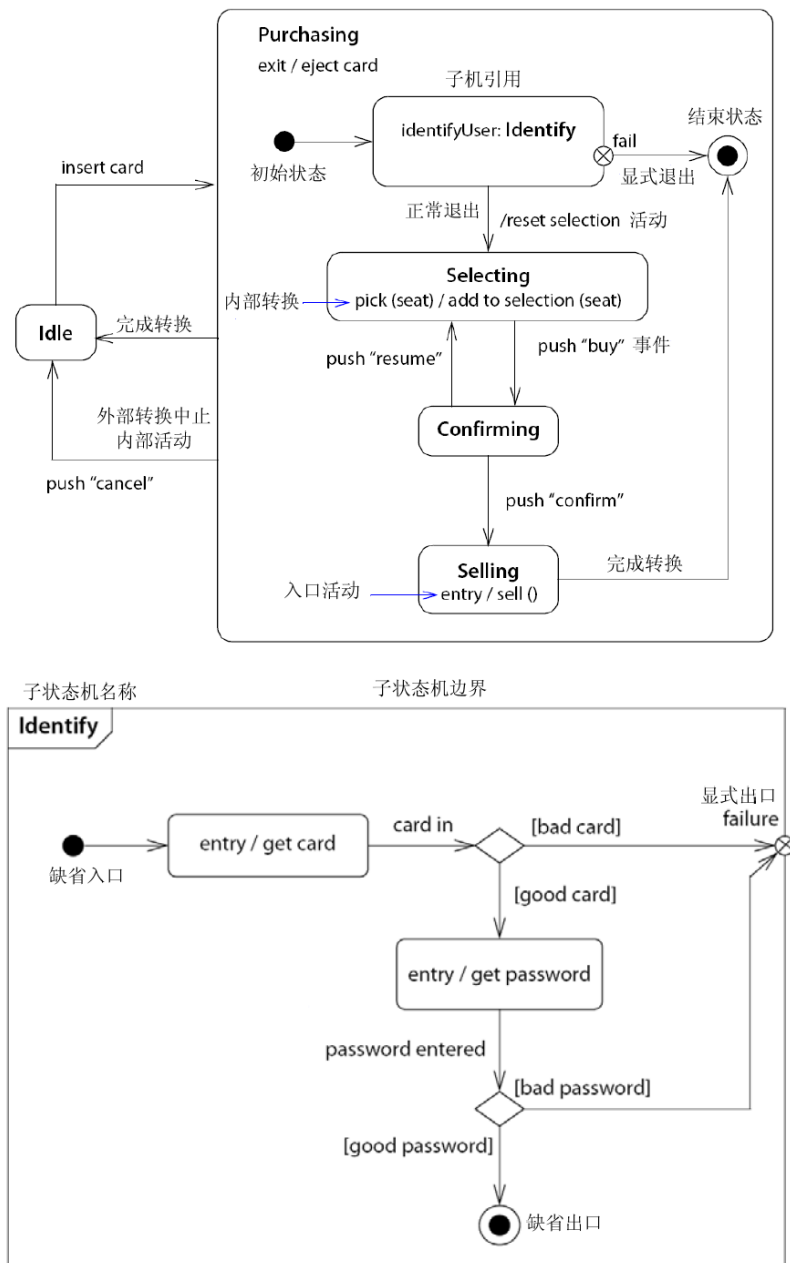


图14-43 UML状态图示例，修改自[Rumbaugh2004]

UML的状态图较为复杂，限于篇幅这里不再做详细的介绍。

UML的状态图是以状态机的方式描述系统的行为。依据对“系统”范围的不同定义，它可以描述不同的方面：如果以整个系统为“系统”，那么它描述的就是整个系统的行为；如果以一个或者几个用例为“系统”，那么它描述的就是一个或者几个用例所包含的行为；如果以一个对象为“系统”，那么它描述的就是一个对象的行为。

在UML的规范里面，认为状态图的主要用法是描述对象的行为。系统中往往会涉及一些重要而且复杂的对象，它们的行为会出现在很多的用例当中，这时就可以使用状态图，从众多用例当中抽取这些对象的行为进行集中的描述。此外，在面对一些特别复杂的用例（比如拥有很多复杂的场景）时，状态图也是一个比交互图更为好用的描述手段。

14.8. 建立状态图

14.8.1. 基于状态转移矩阵建立状态图

建立状态图的步骤如下：

(1) 确定上下文环境。状态图是立足于状态快照进行行为描述的，因此建立状态图时首先要搞清楚状态的主体，确定状态的上下文环境。常见的状态主体有：类、用例、多个用例和整个系统。

(2) 识别状态。状态主体会表现出一些稳定的状态，它们需要被识别出来，并且标记出其中的初始状态和结束状态集。在有些情况下，可能会不存在确定的初始状态和结束状态。

(3) 建立状态转换。根据需求所描述的系统行为，建立各个稳定状态之间可能存在的转换。

(4) 补充详细信息，完善状态图。添加转换的触发事件、转换行为和监护条件等详细信息。在有些情况下也可能需要建立状态图的层次结构或者进行其他更加复杂的工作。

例如，针对上面商品销售的示例，可以按照下面的步骤建立POS类的状态图：

(1) 明确状态图的主体：类POS。

(2) 识别POS可能存在的稳定状态：

- 授权状态：POS机已经准备就绪，但还未参加到工作中来的状态；
- 空闲状态：POS可以参与工作的执行，但并没有工作正在进行的状况；
- 销售开始状态：开始一个新销售事务，系统开始执行一个销售任务的状态；
- 商品信息显示状态：刚刚输入了一个物品项，显示该物品描述信息的状态；
- 错误提示状态：输入信息错误的状态；
- 列表显示状态：以列表方式显示所有已输入物品项信息的状态；
- 销售结束状态：收银员指令 POS 结束正在处理的已有的销售，打印收据。

其中，授权状态为系统的初始状态。

(3) 建立状态转换。可能的状态转换如表14-5所示，其中如果第*i*行第*j*列的元素被标记为Y，则表示第*i*行的状态可以转换为第*j*列的状态。

表14-5 建立状态转换示例

	授权	空闲	销售开始	商品信息显示	错误提示	列表显示	销售结束
授权	Y	Y					
空闲	Y		Y	Y			Y
销售开始				Y			
商品信息显示					Y	Y	
错误提示		Y					
列表显示		Y					
销售结束		Y					

(4) 在已识别状态和转换的基础上, 添加详细的信息说明, 建立如图14-44所示的状态图。

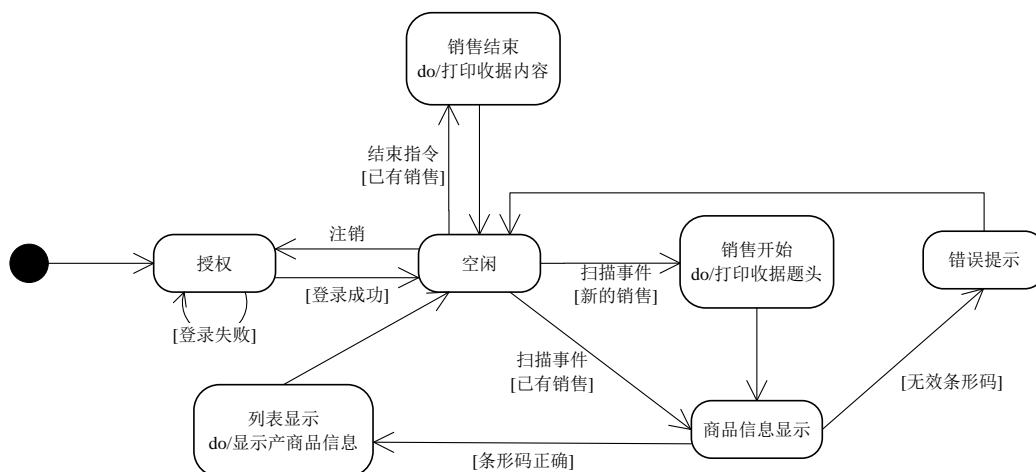


图14-44 状态图建立示例

14.8.2. 状态图的分析作用

分析状态图可以帮助发现需求内容的行为缺陷。如果状态图中发现有无法进入的状态或者无法跳出的状态, 就意味着相应行为的缺失。如果一个状态在发生触发时转移路线不确定, 就意味着监护条件数据缺失或者行为需要细化。如果应该建立的状态转移在需求内容中没有体现, 就需要修正需求内容。如果应该存在的状态在需求内容中没有体现, 也需要修正需求内容。

14.9. 对象约束语言 OCL

14.9.1. 概述

在对象模型、用例模型和行为模型当中, UML使用图形语言来进行系统数据和行为的描述。和自然语言相比, 这些图形语言是建立在一定的模型基础之上的, 每个符号都有着特定的语义和语法规则, 它们可以更加精确进行信息的描述。

但是, 底层模型的语法和语义限制往往又使得它们无法表达足够丰富的内容, 描述系统的各个方面。例如, 一个飞机Airplane和航班Flight之间关系的类图描述如图14-45所示。Airplane有两种类型: 客机passenger和货机cargo。同样, 航班也有客机航班和货机航班。按照常理, 如果一次航班是客机航班, 那么飞机也应该是客机。如果一次航班是货机航班, 那么飞机就应该是货机。但是上述的信息却无法在类图当中得到充分的表达。



图14-45 信息表达不充分的类图示例

对上述问题, 可以通过在类图中增加自然语言的描述注解进行解决。但是因为自然语言具有模糊性, 所以引入自然语言的做法会降低UML的精确性。也有在UML中引入动作规约语言ASL (Action Specification

Language) 以解决上述问题的做法, ASL是一种接近于编程语言的非常详细的动作描述语言。但是ASL过于具体, 而且它和UML模型的配合使用并不容易。最终, OMG在UML的1.1版本中采纳了由IBM工程师建立的对象约束语义OCL (Object Constraint Language)。OCL是一种介于自然语言和ASL之间的一种约束描述语言, 它比自然语言更加精确和规范, 同时又不像ASL那样烦琐。现在, OCL已经成了UML不可缺少的部分。

OCL并不是UML中单独的一个模型, 而是被应用在其他模型当中, 丰富其他模型的语义。OCL是一种规约语言, 它以表达式的方式定义对其他模型元素的约束。这些表达式会根据模型元素的实际状态而表现出“真”、“假”, 并以保持表达式为“真”作为对其他模型元素状态变化的约束和限制。但它们不会修改任何其他模型元素的表述, 也就是说, OCL是一种无副作用的规约语言。

OCL不是一种编程语言。OCL的首要定位是建模语言, 因此它在保证一定表达能力的前提下, 注重于语言的简洁性和抽象性。所以, 它无法被用来描述程序的控制逻辑和工作流程, 它的表达式定义也无法在程序中得到直接的执行。

14.9.2. OCL 的构成

OCL主要由类型、表达式和保留关键字三部分组成。

OCL是一种基于类型的语言, 有着严格的类型定义, 这可以保证它进行形式化描述的能力。OCL所包含的类型既包括原始数据类型 (例如Real、Boolean、String和Integer)、集合数据类型 (例如Collection、Set、Bag和Sequence), 又包括一些专门针对UML模型的类型。关于OCL具体类型及其详细信息请参见[OCLOMG2012, Chapter 11]。

在数据类型定义的基础之上, OCL又严格的定义了它的表达式建立规则。OCL将表达式统一表示为: 操作符+操作数。例如, 在表达式“ $1+1=2$ ”中, “+”和“=”是操作符, “1”、“1”和“2”是操作数。在复杂的嵌套计算当中, 操作数本身可能会是另一个子表达式。例如, 在表达式“ $f()+p()=t$ ”中, “+”和“=”是操作符, “f()”、“p()”和“t”是操作数, 而同时“f()”和“p()”又是子表达式。

依据“操作符+操作数”的思想, OCL定义了可用的操作符、操作数 (操作数最后会被规约为它的子表达式或者数据类型[OCLOMG2012, Chapter 10]) 和操作符与操作数的匹配规则 (参见[OCLOMG2012, Chapter 8]), 这保证了OCL表达式的有效性。

类型的定义和表达式的规则定义构成了OCL的主体, 辅之以必要的保留关键字, 就构成了整个OCL的体系。OCL的保留关键字如表14-6所示。

表14-6 OCL的保留关键字

context	声明OCL表达式的上下文环境
and, or, not, xor	逻辑运算
if ... then...else ...endif	条件判断
package ... endpackage	声明OCL表达式的package环境
inv, pre, post	声明不变量、前置条件和后置条件
let ... in ...	为后续的单个表达式声明变量
def, attr, oper	为后续一定范围内的多个表达式声明属性或者操作

14.9.3. OCL 的应用

OCL在UML模型中有很多的用途，其中最常见的是用来定义UML模型元素的四类约束：不变量（Invariant）、前置条件（Precondition）、后置条件（Postcondition）和监护条件（Guard）。

1. 不变量

不变量是可以对UML类元施加的约束。类元需要保持它的表达式取值在任何时候都为“真”，或至少在指定的时间范围内或者指定的条件下始终为“真”。它可以用在UML的多种类元之上，其中最常见的是用来约束类的属性或者类的方法。

例如，对如14-46 a) 所示类图中的航班Flight类，如果要限制Flight的时间小于4（小时），那么可以进行OCL描述如下：

```
context Flight inv: --context和inv为保留关键字，前者指明约束的环境，后置声明了一个不变量
duration < 4
```

除了上面的表达式描述之外，也可以通过在UML图中直接添加带有<<invariant>>标签的特殊注释来进行说明。在图示中会省略环境和不变量的声明，如图14-46 b)所示，它和上面的表达式完全等价。

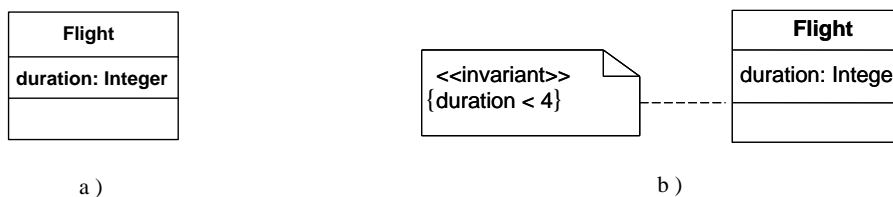


图14-46 不变量说明示例

再例如，对图14-45表达不充分的问题，可以添加OCL的约束表达式如下：

```
context Flight
inv: type = #cargo implies Airplane.type = #cargo
inv: type = #passenger implies Airplane.type = #passenger
```

其中，上面表达式中的“Airplane”是在Flight的上下文环境下依据关联进行的元素导航。导航时使用另一关联端（Airplane）的角色名，在角色缺失的时候就使用另一个关联类的类名作为默认角色名。

2. 前置条件和后置条件

前置条件和后置条件是可以对类元的操作施加的约束。前置条件要求类元在执行操作之前必须保证前置条件的表达式为真。后置条件要求类元在操作执行完成之后必须保证后置条件的表达式为真。

例如，图14-47 a) 所示的类图描述了银行的交易处理工作。如果现在需要添加一个约束为：如果一个顾客的交易总次数超过了30次，那么就在顾客执行新交易时赠与10点的赠品。那么就可以建立如下的表达式：

```
context Customer::getGift() : Integer
pre CustStartTransact:
    bankaccount.costs->size() > 30
post TenDollarsGift:
```

$gift = gift@pre + 10$ and $result = gift$

当然，除了使用表达式之外，也可以直接在类图上面添加带有特定标签的图示，如图14-47 b)所示。

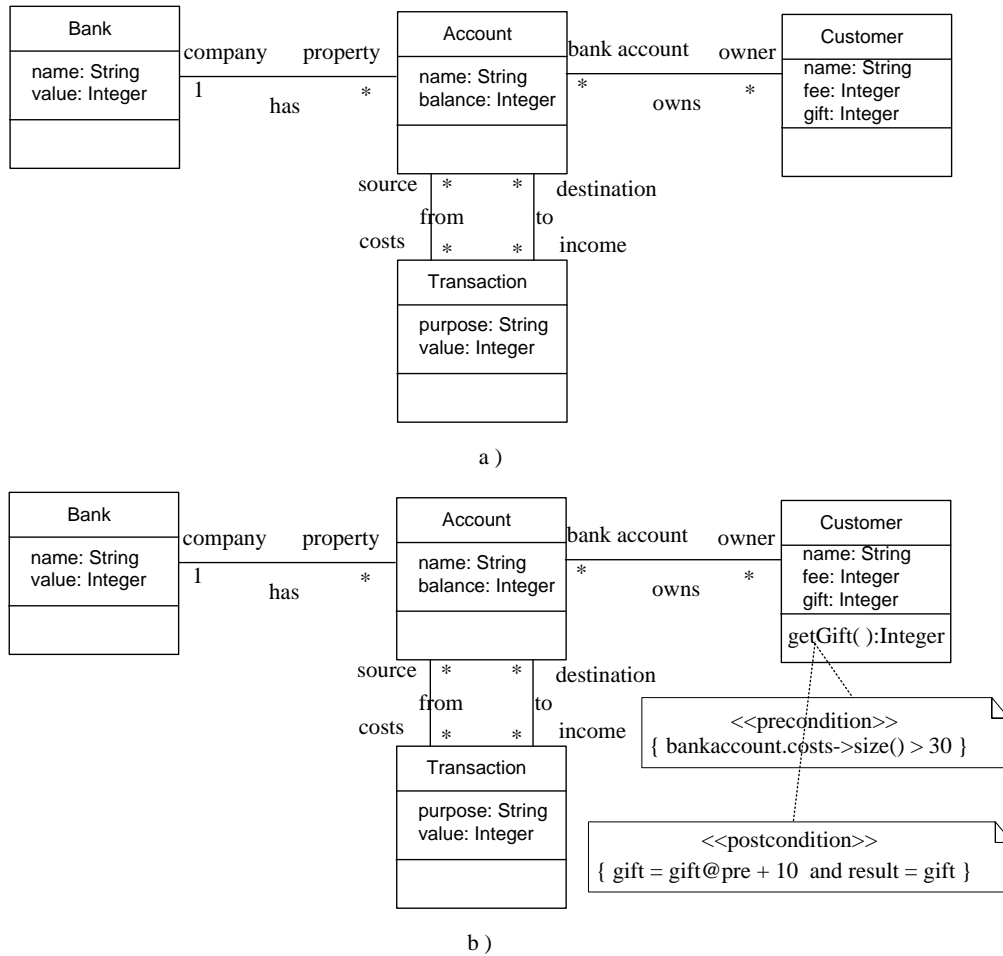


图14-47 前置条件和后置条件声明示例

3. 监护条件

监护条件是对状态机模型中状态转移施加的约束。在状态机到达转移点时，监护条件的表达式需要根据实际状态进行评估，并只有在表达式实际取值为“真”的情况下才进行转移。

监护条件约束被直接表示在状态图之上，放在转移语句之后，用“[]”进行界定，如图14-48所示。

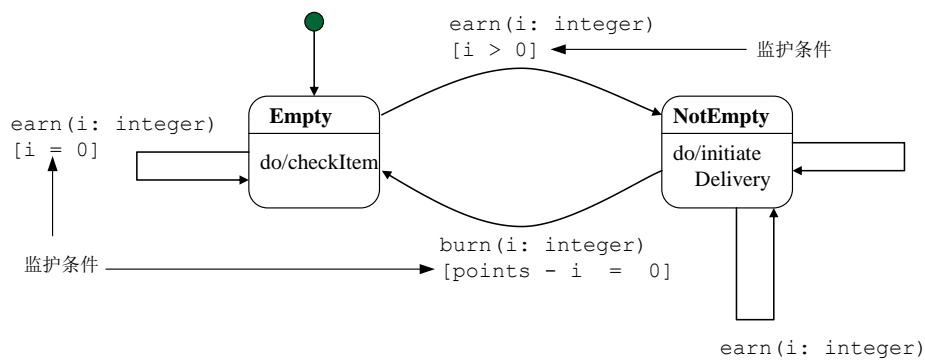


图14-48 监护条件表示示例

14.10. 使用 OCL 建立契约说明

在面向对象模型诸多的约束描述当中,对系统行为所施加的三种约束被认为在面向对象建模当中有着非常重要的作用。这三种约束就是不变量、前置条件和后置条件,它们用来明确和限定系统行为的语义。这三种对系统行为的约束被称为操作契约。明确定义系统的操作契约是面向对象建模其中的一个重要工作,也是契约化设计 (Design By Contract) 思想的主要着重点。

为系统行为定义操作契约是非常重要的。但是复杂的系统具有很多的系统行为,如果要为每一个系统行为都定义操作契约的话,将会是一个非常繁重的工作。所以,在实际工作当中,人们并不会为所有的系统行为都定义操作契约,而是有选择的为其中的一部分系统行为定义操作契约。

为了保证操作契约定义的有效性,在进行系统行为的选择时通常要综合考虑以下两个因素:一、行为的复杂度,和简单的行为相比,复杂的行为会涉及到更多的状态变化,也就需要更多的语义限定;二、行为的清晰度,和逻辑清晰的行为相比,因果关系比较微妙的行为更需要进行语义上的明确。

作为UML的一个部分,有着严格语法和语义的OCL是进行操作契约描述的理想形式。但是在需求分析阶段缺乏完备的模型基础,即作为OCL基础的其他UML图还停留在一个粗略的框架层次上,所以在需求分析阶段会结合使用OCL和自然语言来进行契约的说明。在实践当中,人们通常会以OCL的格式来组织自然语言的描述,其模版如图14-49所示。

操作 (Operation): 操作的名称以及参数说明 引用 (Reference, 可选): 发生此操作的用例 不变量 (Invariant): 不变量描述 前置条件 (Precondition): 前置条件描述 后置条件 (Postcondition): 后置条件描述
--

图14-49 操作契约说明的模版

在进行操作契约的说明时,要从下面几个角度来进行约束的发现工作:

- 不变量: 系统行为中所涉及的敏感状态, 这些状态的改变往往会产生广泛的连锁反应
 - 不可改变的属性
 - 不可改变的关联关系
- 前置条件: 行为发生和顺利完成所需要的系统的状态条件
 - 合法的参数
 - 有效的状态
 - ◆ 对象的存在状态
 - ◆ 对象的属性取值
 - ◆ 有效的关联关系
- 后置条件: 行为顺利完成之后引起的系统状态改变
 - 有效状态的改变
 - ◆ 对象的存在状态
 - ◆ 对象的属性取值

◆ 关联关系的改变

例如对图14-30中的系统行为“输入物品项”，可以进行操作契约的描述如下所述：

操作:

输入物品项enterItem(ItemID, quantity)

引用:

用例：销售处理

前置条件:

有一个销售si正在进行

有一个物品项列表sli存在

si和sli建立了关联

quantity介于[Minimum...Maximum]

ItemID可以和某个物品描述实例spi建立联系

后置条件:

创建了一个商品实例spi

spi和sli建立了关联

spi.quantity被置为参数quantity

spi.itemID被置为参数itemID

14.11. 基于 CRC 卡面向对象分析方法

前面所描述的面向对象建模方法都是适用于简单情况下的建模方法。在需求信息比较明确时，它们能够发挥很好的作用。但是在复杂情况下，需求的获取和分析是交织前进的，也就是说在进行分析与建模时并没有非常明确和固定的需求信息可资利用。这样，前面描述的建模方法就很难起到好的效果。

基于CRC卡的职责驱动方法就是一种[Beck1989]提出用来处理复杂情况的面向对象建模方法，它和需求的获取活动互相促进，齐头并进。当然，基于CRC卡的职责驱动方法在实际开发中的应用要比前面所述的面向对象建模方法复杂的多，需要更多的实践经验和技巧。所以，在面对简单的应用时，本书还是推荐使用前面的简单方法。

基于CRC卡的职责驱动方法在应用的细节上包含有很多复杂的经验总结和启发式规则，所以对它感兴趣的读者可以参考[Wirfs-Brock1990, 2003]。本书仅介绍它的基本思想和大概框架。

14.11.1. CRC 卡

CRC是Candidates、Responsibilities和Collaborators三者的缩写。基于CRC可以建立一种索引卡片，被称为CRC卡。CRC卡是由Kent Beck和Ward Cunningham于1988年发明的设计方法，用于描述早期的设计思想。后来，因为使用起来方便有效，所以CRC卡在面向对象开发当中得到了大量的应用。

CRC卡如图14-54所示。每个卡片代表了一个被发现的候选对象。卡的背面是关于候选对象的非正式描述。卡的正面记录了对象的职责（所维护的状态和可以执行的行为）和协作者。

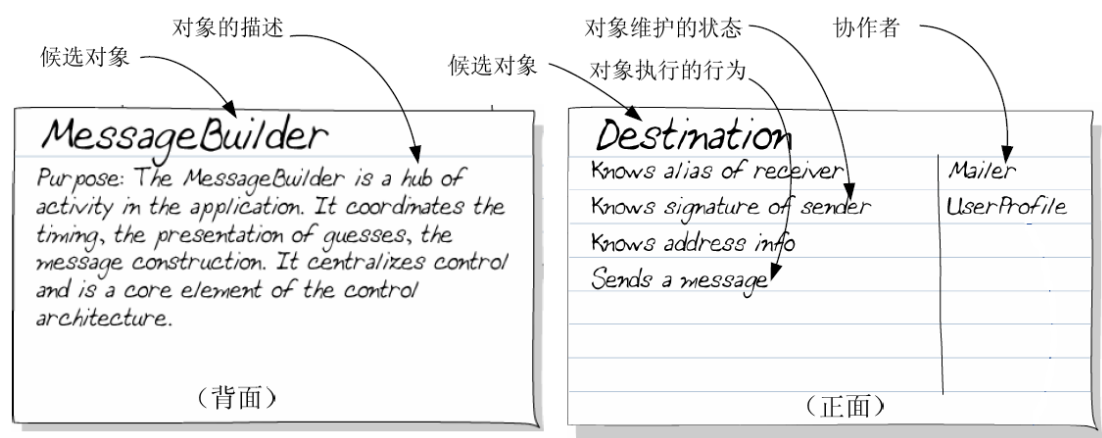


图14-50 CRC卡示例

图14-50仅仅是CRC卡的一个示例图。它在实际应用中的形式可能是多种多样的，卡片、纸张、黑板等等都可以作为CRC卡的介质载体。

CRC卡简洁方便，可以随时被移动、修改或者丢弃，所以它特别适合于在复杂的系统当中进行对象的发现和设计思想的挖掘，即进行复杂情况下的面向对象分析与设计。

14.11.2. 基于 CRC 卡的职责驱动方法

通过对CRC卡的建立、描述、修改和完善等行为，可以为复杂的系统最终建立有效的对象模型。这种面向对象的建模方法被称为基于CRC卡的职责驱动方法，它被广泛的用于面向对象的分析与设计。

在需求分析阶段，基于CRC卡的职责驱动方法的主要工作框架如图14-51所示。下面将分别简要介绍其工作框架中的具体步骤。

1. 确定主题

面对复杂的系统应用，基于CRC卡的职责驱动方法首先需要分析系统的背景信息，找出系统中重要的主题 (Topic)。每个主题意味着系统工作的一个重要部分，整个系统可以按照主题被划分成有机的不同部分。

业务需求和为满足业务需求而规划的系统特性是用来确定主题的最好信息。通常可以围绕业务需求组织主题，或者将每一个系统特性都限定为一个主题。

每个主题内部的系统功能是紧密相关的，而不同主题之间的功能相关性就会大大降低。因此，主题的确是利用模块化的方式在工作的初始阶段就降低问题处理的。

2. 识别候选对象

一旦确定了系统的主题，就可以围绕主题进行候选对象的识别。识别候选对象时可以从下列几个方面的内容着手：

- 问题域中重要的对象和结构；
- 系统的控制和协调行为；
- 需要被传递的重要信息流；
- 关联硬件和其他的关联系统。

识别出候选对象之后，为它们分别建立CRC卡，并给以合适的命名。

3. 描述对象特性

对识别出的每个候选对象，需要在CRC卡的背面进行描述。

可以依据下面几个特性来进行对象的描述：

- 在系统中的地位；
- 功能行为；
- 和其他对象的关系；
- 公共责任；
- 抽象级别。
- 复杂度。

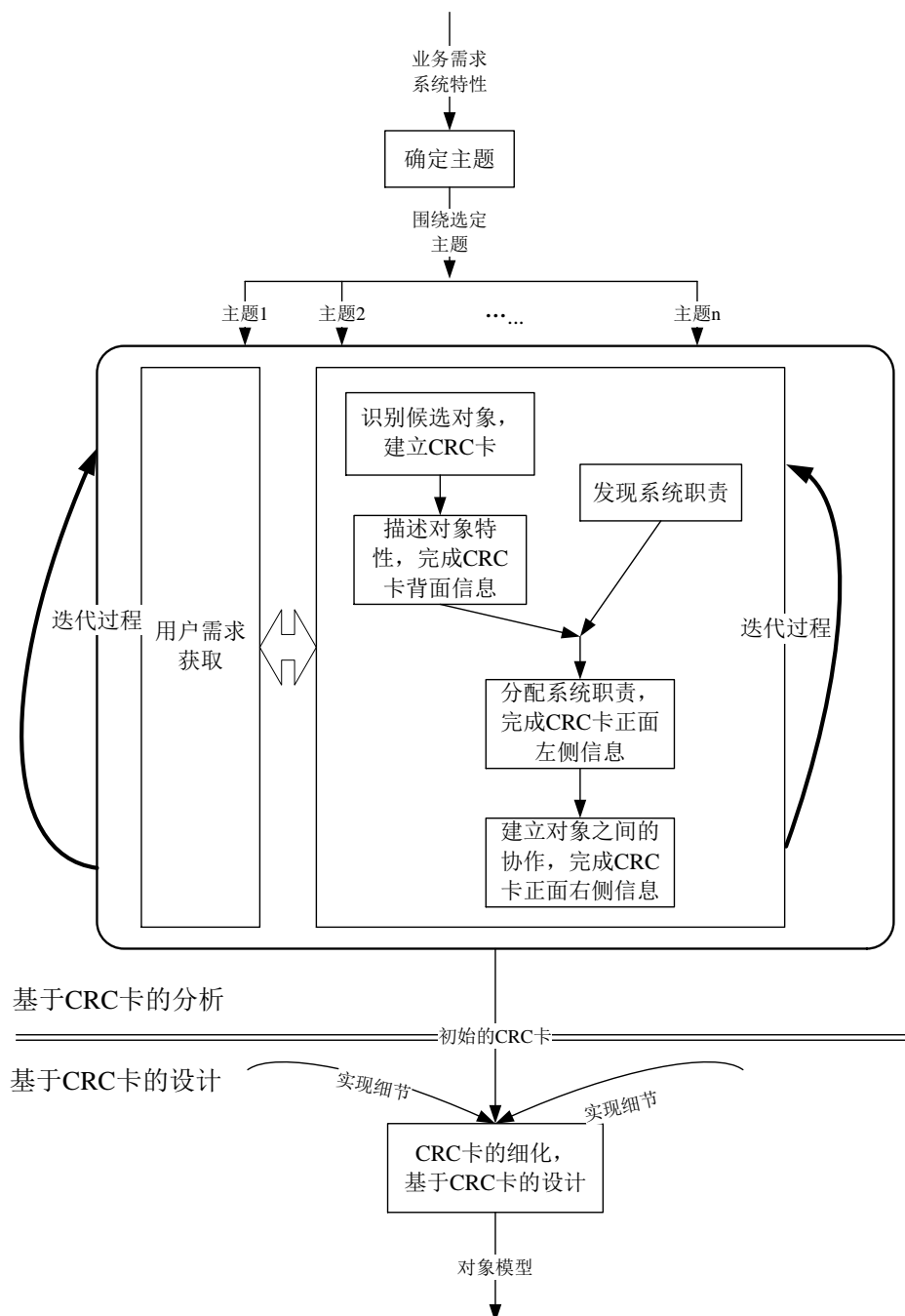


图14-51 基于CRC卡的职责驱动方法的工作框架

4.发现系统职责

职责分为状态的维护和行为的履行两个部分，所以对职责的发现要注重两个方面的内容：一、系统所需要和维护的信息；二、系统的功能和行为。

5. 分配系统职责

发现的系统职责需要被分配给已发现的候选对象，完成CRC卡正面左侧的信息描述。

职责分配时要注意下列原则：

（1）集中信息与行为。系统维持信息的目的是为了发挥信息的作用，即表现一定的行为。行为的表现需要状态信息的支持才有可能。因此，在分配一个状态维护职责时，要向能够利用该状态信息表现出一定行为的对象靠拢；在分配一个行为职责时，要向能够为其提供状态信息支持的对象靠拢。

（2）维持对象的角色。在对象的识别当中，已经为对象构思了某个特定的角色。角色是对象职责的体现，因此在为对象分配职责时，如果分配了过多和特定角色无关的职责，那么可能会冲淡或者改变事先预想的对象角色。如果在分析的过程中的确发生了事实和预计不太一致的情况，通过主动调整候选对象或对象角色（而不是被动改变对象角色）的方式来加以解决。

（3）保持对象责任的相关性。在给一个对象分配职责时，要保证新的职责应该和该对象原有的职责具有相关性，并且都符合该对象所扮演的角色。

（4）保持职责的合适粒度。在分配之前，要将职责控制在一个合理的粒度上。过于复杂的职责，要进行分割。过于简单的职责要进行合并。尤其需要注意的是那些和系统控制行为相关的职责，它们的初始体现往往会比较复杂。

（5）保持对象的粒度。不要让单个对象承担过多的责任。对象在履行责任时可以独立地完成，也可以部分或完全地请求其他对象的帮助。在分配责任时要在这两种方式中取得一个折中，以避免单个对象的职责过于复杂或者过于简单。对于无法对外委托责任的复杂对象，必要的时候考虑将其划分为多个小对象，这些小对象互相协作，共完成原有对象的工作。

（6）不要重复责任。不要将一个责任重复的分配给多个对象，这可能会导致后续阶段中细节设计出现混乱。

（7）必要的时候调整候选对象。在一些职责的分配出现困难时，往往意味着需要进行候选对象的调整。

6.建立对象之间的协作

在完成职责的分配之后，就可以着手进行对象之间协作的建立工作，完成CRC卡正面右侧的信息描述。

可以从下面几个角度出发建立对象之间的协作：

（1）角色。角色表明对象在系统和具体工作中的位置。在系统和工作中相邻或者相关的角色之间通常会进行协作，也就是说对象所扮演的角色隐含着特定的协作关系。

（2）任务。识别任务中的参与对象，分析任务中责任的分解、衔接和分配，这可以帮助发现对象之间的协作。

（3）职责。分析一个对象的职责（尤其是对象所维护的信息）也可以帮助发现对象之间的协作，这些协作体现了对象之间的帮助关系。

引用文献

- [Abbott1983] R. Abbott, Program Design by Informal English Descriptions. Communications of the ACM vol. 26(11), November 1983.
- [Beck1989] K. Beck, W. Cunningham, A Laboratory For Teaching Object-Oriented Thinking, OOPSLA'89, 1989.
- [Booch1993] Booch, G., Object-Oriented Analysis and Design with Applications (2nd Edition), Addison-Wesley Professional, 1993.
- [Booch1994] G. Booch, Coming of Age in an Object-Oriented World, IEEE Software, Vol. 11, 1994 , p. 33-41.
- [Booch1997] G. Booch, Object-Oriented Analysis and Design with Applications, 1st edition, Addison-Wesley, 1997.
- [Booch2005] Booch, G., Rumbaugh, J., Jacobson, I., The Unified Modeling Language User Guide, Addison Wesley Professional, May 19, 2005.
- [Booch2007] G. Booch, R. A. Maksimchuk, M. W. Engle, B. J. Yound, Ph.D., J. Conallen, K. A. Houston, Object-Oriented Analysis and Design with Applications, 3rd edition, Pearson Education, 2007.
- [Bray2002] Bray, I. K., An Introduction to Requirements Engineering (1st edition), Addison Wesley, 2002.
- [Coad1990] Coad, P. and Yourdon, E., Object-Oriented Analysis, Englewood Cliffs, NJ: Prentice-Hall, p. 62, 1990.
- [Cockburn2001] Cockburn, A., Writing Effective Use Cases, Addison-Wesley, 2001.
- [Fowler1996] Fowler, M., Analysis Patterns: Reusable Object Models, Addison-Wesley Professional, 1996.
- [Jacobson1992] Jacobson, I., Christerson, M., Jonsson, P., Oevergaard, G., Object Oriented Software Engineering: a Use Case Driven Approach, Addison-Wesley Publishing Company, 1992.
- [Larman2002] Larman, C., Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process, second edition, Prentice-Hall, 2002.
- [UMLOMG2011] OMG, UML Superstructure Specification, 2011.
- [OCLOMG2012] OMG, UML OCL Specification, 2012.
- [Rumbaugh1991] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, Object-Oriented Modeling and Design, Prentice Hall, 1991.
- [Rumbaugh2004] Rumbaugh, J., Jacobson, I., Booch, G., The Unified Modeling Language Reference Manual (2nd Edition), Addison-Wesley Professional, 2004.
- [Schmid2000] Schmid, K., Scoping software product lines. In Patrick Donohoe, editor, Software

Product Lines, Experience and Research Directions, pages 513–532. Kluwer Academic Publisher, 2000.

[Shlaer1988] Shlaer, S. and Mellor, S., Object-Oriented Systems Analysis, Modeling the World in Data, Englewood Cliffs, NJ: Yourdon Press, p. 15, 1988.

[Ross1987] Ross, R., Entity Modeling: Techniques and Application, Boston, MA: Database Research Group, p. 9, 1987.

[Siddiqi 1994] Siddiqi, J., Challenging Universal Truths of Requirements Engineering, IEEE Software, 1994.

[Smith1988] Smith, M. and Tockey, S., An Integrated Approach to Software Requirements Definition Using Objects, Seattle, WA: Boeing Commercial Airplane Support Division, p. 132, 1988.

[Wirfs-Brock1990] Wirfs-Brock, R., Wilkerson, B., and Wiener, L., Designing Object-Oriented Software Prentice-Hall, 1990.

[Wirfs-Brock2003] Wirfs-Brock, R. and McKean, A., Object Design: Roles, Responsibilities and Collaborations, Addison-Wesley, 2003.