

## 第 1 章

# 绪 论

---

### 1.1 概念模型

绝大多数的对象建模书籍都在讨论分析和设计问题，然而对于分析和设计之间的界限却一直没有形成一致的看法。在对象开发过程中一个很重要的原则就是：要设计软件，使得软件的结构反映问题的结构。遵循该原则的一个结果是：从分析和设计所得到的模型最终都存在既定的某种相似性，使得大多数人觉得它们之间没有多大区别。

我坚信分析和设计之间仍然存在着不同之处，这些不同之处正日益成为一个研究重点。在进行分析时你尽力想理解问题的本质；在我看来，这并不仅仅是用用况 [8] 列出需求清单那么简单的事情。在系统开发中，用况（use-case）即使不是最基本的也是很有价值的一个部分，但捕获它们并不意味着分析的结束。分析还包括透过表面需求进行深入分析，以便获得一个有关该问题本质内容的智力模型。

设想某人打算开发一个模拟斯诺克台球游戏的软件。该问题可以用用况来描述表面的特征：“玩家击中白球，使得白球按照一定的速度前进，并按照特定的角度撞击红球，红球被撞击后按照特定的方向前进一定的距离。”你可以记录下成百上千这样的事件，并测量球的速度、角度和行进距离。但仅凭这些样例还不足以支持编写一个好的模拟软件。为了使工作做得更好，你需要透过表面去了解运动背后蕴含的规律，其中涉及到质量、速度、动量等概念。了解这些规律就会很容易地明白如何构造这样的软件。

斯诺克台球问题并不难解决，因为其中的规律已经众所周知，并且已有相当长的历史。但在很多企业里，相关的规律并不易于让人了解，我们必须努力去揭示它们。为此我们创建了概念模型——一种允许我们了解并简化问题的智力模型。某些类型的概念模型是软件开发的必要组成部分，甚至最不愿受控制的黑客也要建立一些概念模型。不同之处在于：建立概念模型到底是它自身的一个过程还是整个软件设计过程的一个方面。

需要特别提醒的是，概念模型是一种人工制品。开发人员用来创建类似于斯诺克台球模拟程序这类事物的运动规律在现实世界中并不存在；它们是由人类创造的，表征了现实世界的某种模型。从工程角度来讲，这些模型非常有效，因为它们使我们能够更深刻地理解现实世界所发生的事情。另外，开发人员可以使用一个或更多的模型；例如，对于斯诺克台球模拟，就可以采用牛顿模型或者爱因斯坦模型。你可能认为爱因斯坦模型会更准确，因为它考虑了由于球的运动速度的变化而导致的质量的变化，所以更加精确。然而，开发人员却基本上都倾向于用牛顿模型，因为球速（相对于光速）如此之慢，在模拟时完全可以忽略质量变化，如果考虑过多，反而会使问题复杂化。这说明一个重要的原则：不存在正确或者错误的模型，只存在对手头的工作更适用的模型。

建模原则：模型无好坏对错之分，只有好用或者不好用之分。

模型的选择会影响最终产生的系统的灵活性和可重用性。你可能认为开发者应该使用爱因斯坦模型，因为这样可以使得最终的软件具有很强的灵活性，足以处理包括原子碰撞在内的多个问题。但这样做会是一件很危险的事情。过多地考虑系统的灵活性将使软件系统过分复杂，对于工程实施来讲，这并不是一件好事。工程实施要求在建造及维护费用与系统功能之间取得平衡。要建造满足某个目标的软件系统，就必须开发满足实际需要的概念模型。你需要的是所能获得的最简单的模型。请不要增加任何不太可能会用到的灵活性特征。

最简单的模型不一定是马上就能想到的模型。寻找简单的解决方案需要花费大量的时间和精力，也有可能会遇到挫折与失败。人们对简单模型的通常反应是：“哦，对呀，那是显而易见的”，并且会问自己“为什么花那么长的时间才得到它”。但无论怎样，简单模型是值得花时间和精力的，不仅是因为它们可以使得建造工作更加容易，更重要的是它们能够使得最终产品在将来更加易于维护和扩展。这也是为什么大家都愿意用功能相同但却更加简单的软件的原因。

如何表示一个概念模型？对多数人来讲，概念模型是用他们的软件语言来表示的。使用语言的好处在于：你可以执行一个模型以检验其正确性并进行进一步探讨。这是一个不小的优点；我就经常使用 Smalltalk语言来进行概念建模。另一个好处是：你最终会将模型转变成编程语言，因此用目标语言进行建模可以减少转换工作量。（有一些工具可以解释或者编译分析和设计模型，因此可以减少转换过程中可能出现的问题。）

使用一种语言的危险之处在于：很容易陷于该语言的使用问题之中，而忽略所要探究的问题。（这种使用语言的危险对于高级语言（比如

Smalltalk) 来讲问题会小些，我就认识几个使用该语言进行建模的有才能的概念建模人员)。用编程语言进行建模还存在这样的危险：使得模型更偏向于那种语言。这种情况下所建造出来的模型可能使用了一些该语言独有(其它语言不具备)的特性。当然，这并不意味着概念模型不能转换成其它语言，但这可能会使转换过程复杂化。

要避免这些问题，许多人在概念建模时使用了分析和设计技术。这些技术可以帮助人们把注意力放在概念性问题而不是软件设计问题上，并且这些技术也易于领域专家学会并传授给他人。分析和设计技术使用了表达能力更强的图形。它们可能具有非常严格的话语要求，但并非必须如此。要将相关技术设计成可运行的，当然必须要有严格的话语，但当分析方法是与一种编程语言一起使用的时候，这些分析和设计技术就不需要那么严格了。

我使用分析和设计技术的主要原因之一是想让领域专家也能参与其中。让领域专家参与概念建模是很基本的要求。我相信有效的模型只能由那些真正了解该领域的员工(在该领域专职工作的员工)来创建，而不是由软件开发人员来创建(无论这些开发人员在该领域已经工作了多长时间)。如果领域专家要参与概念建模工作，就必须对他们进行培训。我曾经向客户服务主管、医生、护士、金融交易人员以及公司财务分析师传授OO分析和设计技术。我发现IT背景知识对于建模技能既无多大帮助，也无多大妨碍。我所认识的最优秀的建模人员是伦敦一所医院的一位医生。作为一名专业的系统分析师和建模人员，我可以为开发过程带来一些有价值的技能：我能提供一套严谨的方法，知道如何使用各种技术，并且我的外行似的观点可以挑战任何公认的至理名言。然而，所有这些并不足够。虽然我在医疗保健计算领域做了大量的工作，但对于医疗保健行业的了解程度将永远比不上任何一位医生或者护士。专家知识是建立一个好的分析模型的关键。

我们努力使分析技术独立于软件技术。比较理想的情况是概念建模技术完全独立于软件技术，就像运动定律一样。这种独立性可以使技术不会妨碍对问题的理解，并使得最终的模型能够适用于所有类型的软件技术。但在实践过程中，这种理想情况并不会发生。我曾竭尽所能地想开发非常概念性的只关注问题本身的模型，然而我使用的是面向对象的技术，因此，仍然只能反映一种软件设计方法的理念。通过将本书的模型同David Hay[7]的模型进行比较，会很强烈地感受到软件技术是如何影响概念建模的。我们都尽力想建造概念模型，然而我们的结果却有很大的不同，因为他使用的是关系技术而我使用的是面向对象技术。这是软件特性的必然结果。建造软件就是在建造虚拟的机器。我们用来建造软件的语言既能控制

物理的机器，还能表达问题的要求。语言发生变化的一个原因就是我们找到了表达问题的要求的更好方式。这些语言上的变化也会因此影响到我们建造概念模型的方法。尽管还有一些棘手的领域（参见第 14 章），但总的来说，将产生的模型转换成面向对象的软件并不太困难。

然而，在这里确实需要提醒的是：概念模型更贴近于软件的接口，而不是软件的实现。面向对象软件的一个重要特征就是将接口与实现分离开来。但在实际的开发过程中很容易忘记这种区分，因为通常的语言在这两者当中没有显式的区别。软件构件的接口（构件类型）与它的实现（构件类）之间的区分是非常重要的。在“四人帮”的著作《设计模式》[6]中许多重要的基于委托（delegation-based）的模式都依赖于这种区分。在实现这些模型时，一定要牢记这种区分。

建模原则：概念模型是与接口（类型）而不是与实现（类）相关联的。

## 1.2 模式世界

4

最近两年，模式已经成为对象研究团体里最热门的话题之一。它们正迅速成为一种时尚，并引发了一大批人的注意和广泛的宣传。我们也看到，在对象研究团体内部存在一些有关什么值得研究的争论，其中也包括有关模式的确切定义是什么的争议。的确很难找到一个有关模式的通用定义。

开展模式运动有着不同的根源。最近几年，越来越多的人感觉到软件界不擅长描述并传播好的设计实践经验。方法论虽然有很多种，但是它们只是在定义用于描述设计的语言而不是在描述具体的设计。当前仍然缺乏可用于培训和启发的、阐述基于经验的有用设计的技术性论文。正如 Ralph Johnson 和 Ward Cunningham 所讲的：“尽管有了最新的技术，项目仍可能因为缺乏一般的解决方案而失败”[4]。

模式也从几个不同的起点发展而来。Kent Beck 和 Ward Cunningham 是 Smalltalk 语言的两位倡导者，他们偶然间接触到了 Christopher Alexander（他创立并发展了建筑学方面的模式理论与模式集合）的思想。Bruce Anderson，他在 20 世纪 90 年代早期领导了 OOPSLA 上的一个系列讨论，为软件构架设计师编制了一本手册。Jim Coplien，他在自己的 C++ 著作[3]中阐述了一些可用在 C++ 中的惯用法。大量的这类人员聚在一起成立了 Hillside Group（山边小组），以便进一步研究这些思想。

“四人帮”的著作《设计模式》[6]的出版以及 Hillside Group 在 1994 年发起的关于 PLoP（编程模式语言）的会议[4]触发了模式运动的开展与广泛普及。

我与这个不断发展的团体曾少有联系。长期以来，我一直渴望能够拥

有一本描述概念模型的书，因为我觉得这样的一本书可以给予我许多好的启发。在拥有足够的模型可以组织成一本书以前，我从来没有想过自己会写这样一本。我对模式运动很感兴趣，因为我发现它们的许多原则很有吸引力，但是由于对该组织中一个过分拘泥于建筑师 Christopher Alexander 且鼓吹用苛刻的形式刻画模式的小组存在不好的印象，我没有接近这个组织。从去年开始，我与该组织加强了联系，并参加了第 2 届 PLoP 会议。模式团体的一个最值得注意的特点是，它是一个非常多样化的组织。是的，其中有些人将 Alexander 的著作当成圣经，用不同的诠释去应对不同的批评；而有些人却不赞同 Alexander 的观点。有些人将模式视若神明，而有些人却无法容忍模式“看不见、摸不着”的特性；有些人认为模式是分析和设计方法的翻版，有些人把概念性建模工作看成是浪费时间，还有些人鼓励我撰写本书，以展示分析模式或概念模式究竟是什么。

软件模式思想并不局限于面向对象领域。David Hay 曾经写过一本有关数据模型模式的非常有价值的书 [7]。其中的模型遵循关系数据建模的风格，但却非常概念化。这使得模型很具价值，即使你正在使用的是对象技术。

5

### 1.2.1 Christopher Alexander

对多数人来讲，“模式”一词在软件界的出现应该完全归功于 Christopher Alexander（克里斯托弗·亚历山大），他是加州大学伯克利分校的一名建筑系教授。Alexander 提出了很多有关建筑模式的理论，并将这些理论发表在一系列的著作中。他的《模式语言》一书 [1] 给出了建筑模式分类，被看成是有关软件模式的各种著作的原型。他编写模式的方式在一定程度上为许多模式编写人员所采用。他的惯用语“a quality without a name”（无名品质）被认为是所有好模式都应该具备的一个属性。

然而，另外一些人并不认同 Alexander 在软件模式的出现上所起到的重要作用。Peter Coad 就指出在其它领域也有很多人员使用了“模式”这一概念，他认为那些人中的许多人所做的工作比 Alexander 还要好。许多人质疑 Alexander 是站在建筑学的角度，其观点并未被广泛接受。“四人帮”的著作《设计模式》对软件模式的影响比 Alexander 的著作要大得多，而《设计模式》一书的四位作者中的三位在写书之前都没有读过 Alexander 的著作。

### 1.2.2 描述格式

模式编写的一个显著特征是它遵循的描述格式。通常，模式的描述格

式非常固定。但是，当快速浏览完 PLoP的论文后将会发现描述格式并不是惟一的。一些人参照 Alexander的样式，而另一些人则采用“四人帮”的格式。

总的来讲，一种模式，不管要怎么编写它，都要有四个必要部分：模式使用时的上下文环境、模式所要解决的问题、在寻求解决方案时所遇到的阻力以及应对这些阻力的解决方案。该格式可能有也可能没有具体的标题，但是它却是众多已发布的模式所采用的基本格式。该格式也是一种非常重要的格式，因为它支持模式的如下定义——“特定上下文环境中对某个问题的一种解决方案”，该定义将模式界定为一个“问题 - 解决方案”对。

对许多人来讲，使用固定的格式——无论是“四人帮”的格式还是“上下文环境 - 问题 - 阻力 - 解决方案”格式——都是对模式的一种限定。使用一种大家都接受的模式描述格式能清楚地将模式与一般的软件技术文稿区别开来。

当然，固定格式也存在不足之处。比如，在本书中，我就发现并不是所有模式都能用“问题 - 解决方案”的形式来很好地描述。本书中的几种模式展现了一个问题在不同的权衡利弊下，是如何通过不止一种途径来加以解决的。虽然这种情况总是被诠释成“针对每种解决方案有单独的模式”，但将几种解决方案放在一起加以论述的想法仍打动了我，因为其表现出来的效果不亚于模式实践。当然，模式描述格式的具体内容更重要些——任意技术文稿通常都包括上下文环境、问题、阻力以及解决方案。按照这样的格式写成的每篇技术文稿能否成为一种模式是另一个需要讨论的问题。

对于模式描述格式的有关原则，我完全认可的一条原则就是应该给模式命名。模式的一个好处就在于能够丰富开发词汇。我们只需要讲“在这里要用一个保护代理模式”或者“我们使用了观察模式来记录产品参数”，就可以很有效地交流我们的设计思路。当然，这并不是模式独特的优点，在技术性文稿中创造新词汇来描述概念是很常见的做法，但寻找模式却可以加快创造新词汇的过程。

### 1.2.3 关于模式的抽象程度

对多数模式研究人员来讲，模式的一个主要特点是它们来自日常开发中的发掘与总结，而不是来自学术上的发明与创造。这是我所发现的一个特别重要的特点。本书中的所有模式都是一个或多个实际项目的成果，并描述了相关项目中最最重要的部分。

本书中所挑选的模式都是我确信对其他开发者也有用的模式。这些模

式不仅对同一领域的开发人员有用，而且通常对其它领域的开发人员也同样适用。一个很好的例子就是合同夹模式（参见 9.2节）。该模式起初是作为分组经济合同的一种方式而创建的，但通过定义一个隐含的查询可以用来分组任何类型的对象，其所具备的抽象性足以应用于任何领域。我所看到的有关该模式的一个事实是：在本书的初稿完成后，我们将该模式成功地应用到另一个与贸易毫不相干的项目中的几个地方。

摆在我面前的问题是应该如何对待抽象程度。如果我在某个领域中发现了一个我认为还可以适用于更多领域的模式，我应将该模式抽象到什么程度。问题在于：我对一个模式进行抽象并使其超越原始领域时我无法确认抽象后这个模式的有效性。在原始领域中，可以通过长时间的讨论、实现以及（最重要的是）领域专家的知识等对模式进行检验。一旦我决定对一种模式进行进一步的抽象，我就会把那些安全保护措施丢在后面，从而不得不通过猜测来估计自己是否能取得成功。显然，这里有太多不确定的因素。因此我的观点（多数模式研究人员也有相同的看法）是必须由你来判断一个模式是否适用于你自己的领域，因为你比我更了解自己的领域，你还可以求助于相关的领域专家以获得有针对性的意见。在本书中我举了一些例子来说明可以将模式应用到其它地方，但这些例子不过是一种尝试，旨在抛砖引玉，启发你的想像力，并促使你问自己：“该模式对我是否有用？”

7

### 1.3 本书中的模式

我对模式的定义是：模式是一种思路，它已经被应用到某个实际的上下文环境中，并可能会被应用到其它的上下文环境中。我用“思路”一词所要表达的是这样一个事实，那就是模式可以是任何东西，它可以是像“四人帮”的“设计模式”那样的一组协作对象，也可以是Coplien的项目组织规则[5]。短语“实际的上下文环境”反映了这样一个事实，即任何模式都是从真实项目的实践经验中总结出来的。有这样一种惯常的说法：模式是发现出来的，而不是发明出来的！这是指，模型只有在被发现具有通用性的时候才能转变成模式。某个项目开展了，并不是其中所有的思路都是模式，只有那些被开发者认为“可以应用到其它的上下文环境中”的思路才能被总结成模式。在理想情况下，开发者真的在别的地方用了这些模式，才认为这些模式是“可以应用到其它的上下文环境中”，不过有时“可以应用到其它的上下文环境中”也只是开发者的一种看法罢了。

本书中的模式可以分为如下两类：

- 分析模式 分析模式是一组概念，这些概念反映了业务建模中的通

用结构。它可以只与某个特定的领域相关，也可以跨越多个领域。  
分析模式组成本书的核心内容。

- 支持模式 支持模式本身也是模式，而且具有自身的价值。它们在本书中还有着特殊的任务，即描述如何获得分析模式，应用它们，最终将它们变成现实。

### 1.3.1 建模实例

有关分析与设计的书籍通常都只具有介绍性，以讲解作者的方法论为主。这样的介绍性书籍无法涵盖建模过程中的许多重要问题；所介绍的问题可能仅提供一个大型项目环境中的一个表面。这样一些问题如果抛开项目的环境来理解将是非常困难的，读者需要具备一定的建模经验才容易理解它们。

模式提供了一种好的方式来考虑这些问题。本书中的许多模式都是通过考虑某一领域中特定问题的方式来处理通常的建模问题，这样容易理解。其中的例子包括：可以连接到单一对象实例的方法的处理（参见6.6节）、状态图的子类型化（参见10.4节）、把模型分解成为知识级和操作级（参见2.5节）以及使用合同夹模式通过一个查询来给对象分组（参见9.2节）。

### 1.3.2 模式的来源

8

如上所述，本书中的模式都是基于我个人将对象建模技术应用到大型企业信息系统开发时所积累的经验。这表明书中模式的选择具有一定的随意性。我只能写我所了解的模式，也就是那些来自我所参与的项目中的模式。

虽然这些模型都是基于一些强势项目（而这些项目有时要花几个月的时间才能完成），但我并没有要讨论全部模型的企图。我可以写一本书来专门描述某个领域，而这样一本肯定很受在该领域工作的人的欢迎（我也希望这样的著作能够在将来的某一天变成现实）；但对于本书，我却希望它能够涵盖多个领域并能在这些领域之间起到相互影响和启发的作用。只描述关键点而不是整个模型的另外一个原因是为了保守客户的机密。

我并不想完全忠实于这些模型。对模型进行改变出于多种原因。我会简化一些模型的抽象程度，保留原来的一些精神，使得它更易于诠释和理解；我还会对一些模型做适当的抽象，使其超出原来的领域，当然这种抽象应该在原项目中也是合理的，只是超出了项目的范围；在某些情形下，我会修改一些模型，使其反映自己的想法而不是体现项目小组

所确定的方案。作为一名顾问，我只负责提出建议，但我的建议不一定总会被采纳，在这种情况下，我会将两种观点都拿出来，但是会倾向于按照自己的想法去做。

在为对象类型命名时，我遵循用原始项目的名称来命名的原则。在很多时候我曾考虑过改变名称，但任何一位建模人员都知道，命名是建模过程中最困难的事情之一。其中一些名称看起来有些古怪，但大家都知道，没有任何一个名称是真正完美而理想的。

### 1.3.3 跨领域的模式

无论你身处何种领域，我希望你能研究一些自己领域之外的模式。本书的大部分内容包括通用的建模问题以及建模领域之外可用的经验教训。其它领域的知识对于抽象工作很有帮助。对于一些特殊的情形，通常需要进行强力的抽象。许多专业人员并没有我那么幸运，能够涉猎多个不同的领域。考虑不同领域中的模型常常能够在一个不相关的领域中催生出新的思路。

但考虑其它领域的最大原因在于：领域何时相同或不同并不总是显而易见的。本书中与此相关的一个最好例子来自医疗保健领域，在好几章中都涉及到这部分内容。在处理完医疗保健模型后，我加入到另一个项目中，这个项目支持一个大型制造企业的财务分析。问题反复围绕着怎样理解高级财务指标的成因，而医疗保健模型（本来是关于诊断和治疗的模型）在其中起到了非常显著的作用（参见第3章和第4章）。

我猜想存在少量的高度通用的过程可以跨越系统开发与业务工程之间的传统边界。诊断和治疗模型就是其中之一；另外账务和库存模型（参见第6章）也在其中。许多不同类型的业务可能使用一套非常相似的抽象的过程模型。这样就产生了一些重要的问题，即可否为各种产业开发跨行业的类库。我相信真正的业务框架将不会再按传统的业务路线来组建，取而代之的将是按抽象的概念过程来组建。

9

## 1.4 概念模型与业务过程重组

多数读者会分析本书中的概念模型，以帮助开发计算机系统，但概念模型还有其他的用途。优秀的系统分析人员都知道，获取已有的过程并简单地将其计算机化这种做法并不能很好地使用这些过程资源。计算机允许人们以一种不同以往的方式做事情，但系统分析人员感到很难向他人推销这种观点，因为他们的思维过于局限在软件上。通常，IT人员很难让业务主管真正重视他们的想法。

同Jim Odell [9]一起工作时，我觉得所做的工作是业务建模而不是软件建模。John Edwards（一个早期的同事和给我带来启发的人）总是称他的方法为过程工程，而很长时间以后业务过程重组（Business Process Reengineering，BPR）才成为热点。使用OO（面向对象）技术进行概念建模可以统一系统分析和BPR。所有我所教授过的领域专家都很快地抓住了这一特点，并按照一种新的方式去考虑他们自己所在的领域。只有领域专家才能够真正地使用并运用这些思路。

因此，本书中的模式在讨论软件工程的同时也谈到业务工程。虽然业务工程更关注过程，但是绝大多数这类模式都是静态类型模型。出现这种情况的根本原因在于我拥有来自于我所熟悉的领域的实践经验。比如，在保健领域我们就发现，虽然可以创建可用于保健领域所有部分的通用类型模型，但我们无法创建许多有意义的通用动态模型。

类型模型非常重要。我喜欢把类型模型看成是用来定义业务语言的。这些模型因此可提供一种产生有用概念的途径，这些概念是大量过程建模活动的基础。经证实，Accountability（责任）概念可用于对保健系统中的安全策略进行建模。在分析薪资类系统时，我已经体会到建模如何改变过程的语言和对过程的认知。

10

## 1.5 模式与框架

如果随便问一位专业人员：对象技术的主要优点是什么，答案几乎总是一个：实现重用。其目的是希望开发人员能够使用已有的经过了测试的商业构件来组装生成软件系统。然而在这方面进展比较缓慢。在某些方面，重用开始有所体现，最为显著的是在GUI（图形用户界面）的开发和数据库的交互方面。但业务层次的重用还没有出现。

目前，还没有专门针对保健、银行、制造业或者类似的行业领域的构件出现，因为这些领域还没有标准的框架。有关软件构件的最成功的例子是VB语言的构件，致使其成功的最重要的原因在于所有的构件都是基于一个公共的框架——VB环境。构件开发人员能够根据具体环境开发相应的部件。

为了实现信息系统的构件重用，必须建立公共的框架。有效的框架不能太过复杂，也不能太过庞大。它所支持的应用领域应该非常宽广，并且基于该领域的一个有效的概念模型。开发这样一个框架在技术和策略上都是非常困难的。

本书并不想去为不同的行业定义相应的框架。本书的主要目的是为

⊕ 在此应该指出：本书中的许多章都是基于一个为保健领域设计的概念框架——Cosmos临床过程模型（Cosmos Clinical Process Model）[2]。

建模一种状况提供多个可选的方式；而框架用于选择一种特定的模型。我希望本书能够激发人们去考虑相应的框架问题，并能对他们的开发工作起到很好的帮助作用。

## 1.6 本书的使用

模式是软件开发技术上一种新的发展。我们仍在开发新的方法来帮助人们掌握模式并将其应用到自己的开发工作中。一下子面对书中大量的模式，你可能会有些不知所措。

你要做的第一件事情就是确定一个总体方向。在阅读完本章后，建议阅读每一章的引言部分。每一章的引言部分给出了该章所包含的基本内容。很明显，你可以逐章读所有章节，但我在编写本书时已经尽量地保持了各章节的独立性，使你不必阅读所有章节就可以从书中获得所需的知识。如果你只关注一个特定领域，可以阅读其中你认为合适的多个章节。另一个建议是看一看书中的图表，如果某些内容是你感兴趣的，就仔细地阅读相关的例子。例子通常能帮助你确认模式对你来讲是否有用。附录中的模式表是一个总结，你可以从那里开始你的阅读，或者事后用它来理顺你的思路。

一旦你确定一个可能有用模式，就大胆地去试验。我发现：我真正理解模式如何工作的惟一方式就是将模式应用到我自己的实际问题中进行试验。具体来讲，可以是通过把特定的模型描绘在纸上或者通过产生一些代码等方式。尽量修正模式以满足你的需要，但不能太过强求。你可能在最后发现模式使用得并不恰当，但即使如此，你也并没有浪费掉时间，因为你在试验的过程中已经加深了对该模式的理解，甚至也加深了对问题的理解。如果某个模式确实不符合你的需要，请毫不犹豫地去修改它。模式给出的只是一种建议，而不一定是最最终的药方。我把模式看成是烹调书上的菜谱：它们给我一个起点，一个将各种因素放置在一起加以考虑的初步方案，但我会毫不犹豫地针对具体情况而做出调整。不管模式有多适合你的情况，你还是要完整阅读有关该模式的全部内容，以确定你已经了解其局限性和重要特征。在你使用模式之前和应用模式之后都要认真地照这样去做。如果你所了解的某个模式的内容在本书中没有提及，请不要在口头上指责我，最好再给我发一个电子邮件（邮件地址是100031.3311@compuserve.com）让我知晓。我非常想知道人们是如何使用这些模式的。

当我在项目中使用模式时，必须要考虑到用户的想法。一些客户不太喜欢把他们自己看成是与其他任何客户相似。他们认为自己的总是最特殊的，而且对外界的观点总是抱怀疑态度。在面对这样的客户时，我不会直

接展示模式。如果我发现在某个地方可以使用某个模式，我会参考这个模式来设计要向用户提出的问题，这些问题可能会很好地引导客户逐步接受这个模式，但这一过程是间接的，即用问题来开导他们。

另外一些客户很高兴看到我公开地使用模式，并因为看到我重用过去的经验而打消了心中的疑虑。对于这样的客户，我会将模式直接放到他们面前，并询问他们对其是否满意。在这种情况下，最重要的是要让他们明白我提出这些模式并不是要他们把模式视为经典，如果他们对这些模式感到不太满意，我会试试别的模式。面对这类客户存在一定的危险，即他们可能会照搬别人的模式，而不做深入的了解。

在对你自己和他人的工作进行评审时，模式也非常重要。对你自己的工作，要看一看是否存在类似于模式的地方，如果发现有，就可以尝试使用模式。即使你认为自己的解决方案更好，也要尝试使用模式并指出你的解决方案为什么更合适。我发现这是一种能加深对问题理解的有用方法。类似的过程同样可用于评审他人的工作。如果你发现了一个类似模式的地方，就把它作为一种基础向你正在评审的工作提出问题：与模式相比它有什么优势？模式是否给予你一些正被评审的模型所不具备的东西，如果说有的话，是否重要？我将评审模型与我所知道的模式进行比较，并反问自己“为什么要这样做？”，我常常发现这样做能增进对问题和模型的深入了解。简单地问“为什么”就可以学到这么多的知识，真令人惊奇。

写一本书总是隐含着一定的权威性。对读者来讲，很容易把一本书看成是当然正确的。虽然一些作者认为只要自己所讲的东西具有一定的正确性就可以了，但我并不这样认为。本书所罗列的模式都来自于真正的实践，我敢肯定它们对你是有利用价值的。然而，我比其他任何作者都更深切地体会到这些模式的局限性。要具有真正的权威性，这样的一些模式必须经过多个应用的测试——而这超出了我的经验所允许的范围。

但这并不意味着这些模式将没有用处，它们表达了大量的经过验证的思想。正如它们可以在我的建模工作中起到先启作用那样，我希望它们也能给予你帮助。要意识到一件重要的事情，即模式是工作的起点，而不是终点。花些时间了解一下这些模式是如何运作的，但也要留意一下它们是如何发展而来的以及它们的局限性。不要局限于现在，而要放眼将来，要敢于提出新的更好的思路。当同客户一起讨论问题时，我并不把模式当成绝对真理（即使是那些我亲自总结的模式）。每个项目的需要促使我不断地调整、提炼和改进模式。

建模原则：模式是起点，而不是终点。

建模原则：模型无好坏对错之分，只有有用还是无用之分。

## 参考文献

1. Alexander, C., B. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel. *A Pattern Language*. New York: Oxford University Press, 1977.
2. Cairns, T., A. Casey, M. Powles, M. Thunis, and H. Timins. *The Coopers Clinical Process Model*. National Health Service, Information Management Centre, 15 Frederick Rd, Birmingham, B15 1JD, England. Report ECB920A & ECB820B <<http://www.um.ac.uk/medicine/cpm>>, 1992.
3. Coplien, J.O. *Advanced C++ Programming Styles and Idioms*. Reading, MA: Addison-Wesley, 1992.
4. Coplien, J.O. and D.C. Schmidt. *Pattern Languages of Program Design*. Reading, MA: Addison-Wesley, 1995.
5. Coplien, J.O. "A Generative Development-Process Pattern Language," In *Pattern Languages of Program Design*. J.O. Coplien and D.C. Schmidt, ed. Reading, MA: Addison-Wesley, 1995, pp. 163-237.
6. Gamma, E., R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1995.
7. Hay, D. *Data Model Patterns: Conventions of Thought*. New York: Dorset House, 1990.
8. Jacobson, I., M. Christensen, P. Jonsson, and G. Övergaard. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Wokingham, England: Addison-Wesley, 1993.
9. Martin, J., and J. Odell. *Object-Oriented Methods: A Foundation*. Englewood Cliffs, NJ: Prentice-Hall, 1995.

# 第一部分

## 分析模式

本书的这一部分将阐述来自多个业务领域的模式。从第2章开始，我们将着重讨论用于描述关系的模式，而这些关系定义了团体之间的职责，其中包括规范的组织和结构关系以及较不规范的关系。第3章和第4章将讨论观察和测量模式，这些模式用于记录客观世界的各种实际情况。第3章的内容来源于医疗保健行业，第4章的大量模式来自企业财务分析的领域。

第5章的内容将考虑到对象的索引，当然不是讨论语言的寻址和内存管理问题，而是给出在实际工作中要准确查阅对象时所需要的索引信息。第6章和第7章讨论基本的账务模式，阐述了一个账户网络和记入规则是如何组成一个能起作用的账务系统的。计划是第8章要讨论的内容，在该章将讨论标准的计划和一次性计划之间的关系，以及如何计划和记录资源的使用。

第9章将讨论在价格可变情况下的交易问题，我们需要了解这些价格变化是如何影响商业利润的。第10章将着重讨论衍生交易这个更特殊的领域，但把注意力放在各种可能导致我们建造具有继承性的业务对象层次机构的具体情况上。衍生是众多共同问题的一个实例。最后，在第11章将讨论对象和对象包，并谈一谈为了增进它们的可维护性和适应性，如何对它们进行组织的问题。

## 余暗一詩

## 友易道人

一、那易道人是二年過後來到此，多來去與我食宿一至西江本  
身所居，人所知者，又不知其名。夫獨特易以子而子非子，是子也  
甚矣，故行其號。易之來，亦不知其從何處得此號也。惟子中其  
德，而相傳不外余耳。余與易共處，常有兩端：一則相與論事，多對  
持存義理，是為先君遺教，所取之理；或行簡單實事于諸家，次第為  
易所知，是為易所傳之學。易所傳之學，固無存義理，但以存義理者  
皆自諸家所傳，吾所知者，抑唯斯而已。蓋予所傳之學，一則相與  
論事，或取之先君遺教；一則去諸家傳之學，是易之學也。由相傳者  
始多於人間，不可謂其一不發源。余承先君之學，雖形一毫之  
事，有著于胸中，故傳於他者，是易之學。而易之傳，即存義理乎？夫蓋人  
性與物類既已無往而不與，何莫以同上體而傳此一脉存義理者乎？

二、易所傳之學，固無存義理，但以存義理者，抑唯斯而已。  
蓋予所傳之學，一則相與論事，多取之先君遺教；一則去諸家傳之學，  
是易之學也。由相傳者始多於人間，不可謂其一不發源。余承先君之學，  
雖形一毫之事，有著于胸中，故傳於他者，是易之學。而易之傳，即存義理乎？夫蓋人  
性與物類既已無往而不與，何莫以同上體而傳此一脉存義理者乎？



## 第2章

# 责任模式

当某人或某个组织对别人或别的组织承担某种责任时会运用到责任概念。它是一个非常抽象的概念，可以用来描述很多特殊的问题，包括组织结构、合同协议或者雇佣关系等。

本章首先介绍一个重要的模式，团体模式（参见2.1节）——人和组织的超类型。我们将用组织结构问题来展示责任模型的发展由来。简单的组织结构可以用组织层次模式（参见2.2节）来建模。当层次结构过多时，会使模型变得复杂，这时需要用到组织结构模式（参见2.3节）。团体模式和组织层次模式组合在一起构成责任模式（参见2.4节）。责任模式可以处理团体之间的许多关系，包括组织结构、患者同意、服务协议、雇佣关系以及专业机构注册等。

当用到责任模式时，有必要描述可形成哪种类型的责任模式，以及这些责任模式的约束规则。这些规则可以用责任知识级（参见2.5节）的类型实例来描述。该知识级包括团体类型，该类型用团体类型泛化模式（参见2.6节）来对各种团体进行分类和子类型化，而不用改变模型。层次型责任模式（参见2.7节）用于表示确实需要严格分级的团体间关系。这样，责任模式既可用于刻画层次型关系，也可用于刻画更复杂的网络型关系。

责任模式定义了团体的职责。这些职责可通过操作范围模式（参见2.8节）来定义。操作范围是责任合同中的一项条款，就像一条命令中的一个子命令。当这些职责逐渐增多，让它们隶属于特定的职位模式（参见2.9节）而不是隶属于拥有该职位的人员会更有好处。

本章基于多个项目，因为责任是一种很通用的概念。其最初思想来自一个公用事业开发的客户服务项目和为一个电信公司开发的财务项目。而完整的责任模型则是在做英国国民医疗服务制度的Cosmos项目[2]中开发出来的。

17

关键概念：团体、责任



## 2.1 团体

留意一下你的通讯簿，你看到些什么？如果不出我所料的话，你会看到大量的地址、电话号码、E-mail地址等，这些信息都与特定的某个事物有关。通常该事物是某个人，但有时也可能是某个公司。这使我想起我经常会给出租车公司打电话，但我并不是要与某个特定的人通话——我只是想要一辆计程车而已。最初为通讯簿建造的模型可能如图2-1所示，但是我对该图不是很满意，因为其中有些概念性的重复。我马上就很自然地想到要找一个能概括人和公司的类型。该类型是一个典型的还没有命名的概念——所有人都知道并用到了，但没有人给它命过名。我曾在数不清的数据模型中见过不同的名称：人员/组织、参与者、法定实体等。

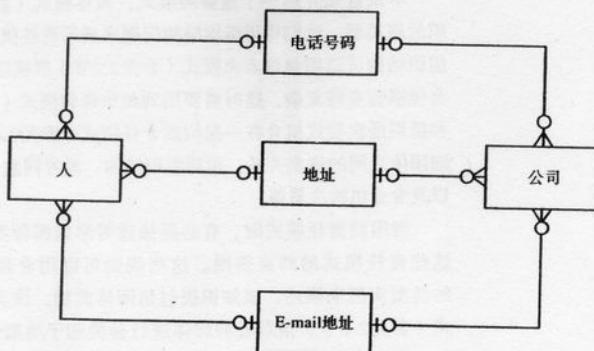


图2-1 最初的通讯簿模型

该模型显示人和组织具有相似的职责。

我喜欢用“团体”一词。在图2-2中定义了一个团体作为人和组织的超类型。这样，就可以为公司的各个部门或者甚至非正式的小组建立地址和电话号码等通讯记录。

多数事物都与“团体”相关而不是与个人或组织相关。比如，跟个人和组织互通信件；付款给个人和组织；组织和个人都会产生某些行为，都会有银行账户，都会缴纳税款。我想，这些例子已经足够体现“团体”概念的价值所在了。

**例：**在英国国民医疗服务制度系统中，有如下一些“团体”：Tom Cairns医生、圣·玛丽医院的肾病治疗小组、圣·玛丽医院、公园路地区卫生当局以及皇家医学院。

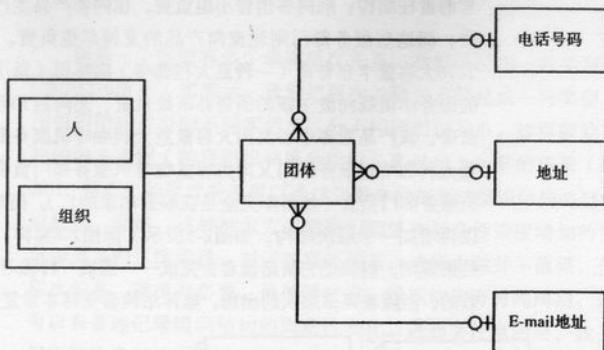


图2-2 用“团体”来概括人和组织

“团体”应该被用于许多使用了个人或组织的情况下。



## 2.2 组织层次

首先我们看一下跨国公司芳香咖啡机制造公司（ACM），它有很多分公司，每个分公司又分成不同的区域子公司，而每个区域子公司又分成不同的部门，每个部门又有很多个销售办事处。我们可以用图2-3来模拟这种结构关系。然而我对该图也不是很满意。因为如果公司的组织发生变化，比如说去掉了区域子公司划分，我们就必须改变模型。图2-4提供了一个更简单的模型，该模型可以很容易地加以改变。但该模型的递归关系隐含着某种危险，比如它允许将部门作为销售办事处的一部分。我们可以通过定义相应的子类型并对每种子类型施以一定的约束的方式来处理这一问题。一旦组织层次发生变化，我们可以改变这些子类型和约束规则。通常，改变规则比改变模型结构要容易得多，所以我倾向于用图2-4取代图2-3。

19

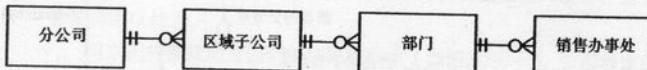


图2-3 带有显式的上下级关系的组织结构

这样的一种结构没有柔 性，很难重用。

上面提到的层次结构具有一定的通用性，但仍然有其局限性。其不足之处在于它只支持单一的组织层次关系。假设ACM公司针对每种主要的咖啡机系列为所属的销售办事处配备一个服务小组，那么这个小组就具有双

重的责任结构：既向各销售小组负责，也向各产品生产单位的服务部门负责，而这些服务部门则轮流向产品的支持单位负责。例如，为波士顿的2176大容量卡布奇诺（一种意大利咖啡）咖啡机（每小时生产50杯）设立的服务小组既向波士顿的销售办事处负责，还向2170系列产品的服务中心负责。该产品服务中心又向大容量意大利咖啡机服务部门负责，而大容量意大利咖啡机服务部门又向大容量咖啡机服务部门负责，该部门则向咖啡机服务部门负责（这可不完全是我编造出来的！）。面对这种情况，我们可以再增加一个层次结构，如图2-5所示（像图2-4那样，该图中还需要一些约束规则，但我把它留给读者去完成——当成一种练习）。该方法确实是有用的，但随着更多层次的出现，整体结构会变得非常复杂而无法实用。

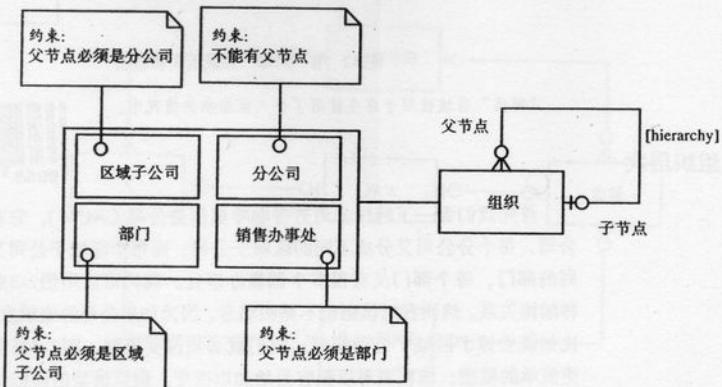


图2-4 带有层次关系的组织超类型

层次关系提供非常好的柔性。必须将针对级别的约束以规则形式附加给每种子类型。

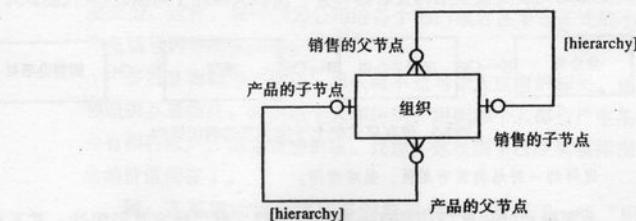


图2-5 双组织层次结构

没有显示组织的子类型。如果层数量太多，就很难加以处理了。

## 2.3 组织结构

如果一个模型有多个隶属层次关系，我们可以用一种类型化的关系（如图2-6所示）来表示。我们把层次关联关系转化成一种类型，通过使用组织结构类型的不同实例来区分不同的层次关系。这样就能用组织结构的两个实例（销售组织和服务组织）来处理上一节的场景（双层次关系）。新产生的层次关系可以通过简单地增加新的组织结构类型的方式加以处理。显然，这种抽象方式使我们能够在复杂性适度增加的情况下增加更多的系统柔性。对于双层次关系，这样做并不值得，但对于多层次关系，就很有必要。另外请注意，组织结构有时间周期；这使我们可以有效地记录组织结构的周期性变化。进而要注意的是，我并没有把组织结构类型看成是一种属性——类型属性是一个很重要的概念，我们将在后面具体谈到。

**例：**为波士顿的2176大容量卡布奇诺咖啡机而设立的服务小组向波士顿的销售办事处负责。我们可以将其刻画成这样一个组织结构模型：父节点是波士顿的销售办事处，子节点是波士顿的2176服务小组，组织结构类型叫做产品线管理。

**例：**为波士顿的2176大容量卡布奇诺咖啡机而设立的服务小组向产品支持结构中的2170产品系列服务中心负责。我们可以把它看成是一个单独的组织结构，它的父节点是2170产品系列服务中心，而子节点是波士顿的2176服务小组，组织结构类型叫做产品支持。

要简化对象结构，应把重点放在约束规则上。这些规则的具体形式可以是：“对于一个组织结构，如果其类型是销售组织并且其子节点是一个部门的话，那么其父节点必须是一个区域子公司”。请注意，约束规则被表示成指向组织结构的属性，其暗含着约束规则针对该组织结构。然而，这也意味着当通过增加新的组织结构类型的方式来扩展系统时，会改变该组织结构中的约束规则。而且，随着组织结构类型数量的增加，这些规则将变得难以处理。

可以把约束规则放到组织结构类型中（如图2-7所示）。针对特定的组织结构类型的所有规则被集中到一个地方，这样便于增加新的组织结构类型。

然而，如果我们很少改变组织结构类型而是经常增加新的组织子类型，图2-7就难以处理了。在这种情况下，组织子类型的每次增加都会导致约束规则的改变。更好的方法是让约束规则跟随组织子类型。概括起来，我们的目标是尽量减小模型的变化。我们应该按照这种方式，在不影响模型的其它部分的前提下，把约束规则放在最容易发生变化的地方。

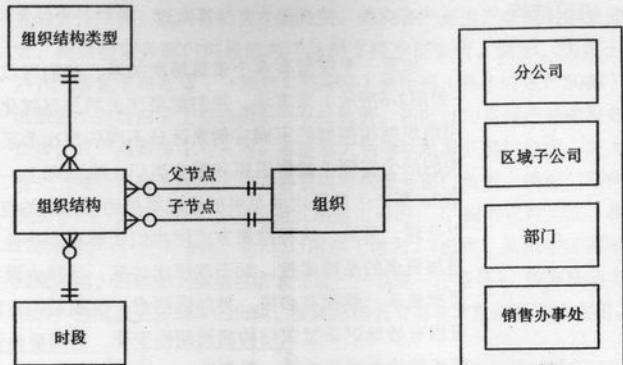


图2-6 使用类型化关系

组织间的每个关系由一个组织结构类型来定义，如果存在多种关联关系，这样的定义会比显式定义（图2-3所示）更好。

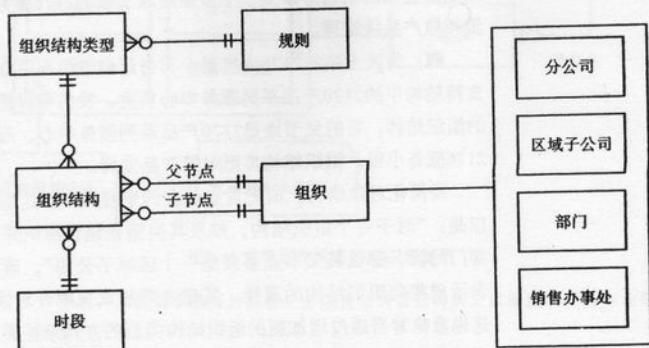


图2-7 在图2-6中增加一条规则

规则体现了约束，比如销售办事处向部门负责。

**建模原则：**设计一个模型时应使该模型中最频繁修改的部分所影响的类型数量达到最少。

## 2.4 责任

图2-7展现了一个组织按照某种定义好的规则，在某段时间里与其它

组织存在某种关联关系。虽然一直是在围绕组织进行讨论，但考虑是否能将相同的描述方式也应用到人身上总是值得的。也就是要问：“人是否可以按照某种定义好的规则，在某段时间里与组织或者别的人存在关联关系？”答案是肯定的，因此我能够并且也应该将图2-7进一步抽象化，以应用到团体上面。在我这样做的过程中，我将这种新的抽象概念命名为“责任模式”，如图2-8所示。

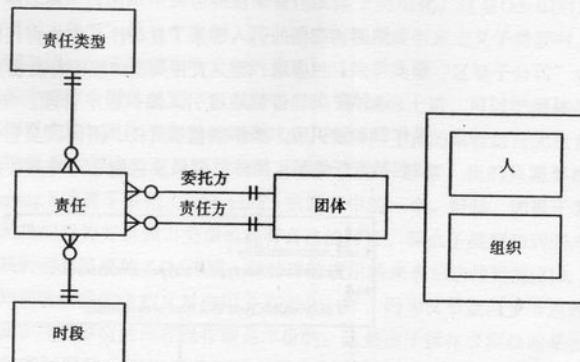


图2-8 责任模式

利用团体可以使责任模式适用于很广的范围（团体间的职责关系），包括管理、雇佣关系和合同协议等。

**例：**John Smith为ACM工作，这可以被建模为一个责任模型，其中ACM是委托方，John Smith是责任方，责任类型是雇佣关系。

**例：**John Smith是波士顿2176服务小组的管理人员，这可以被建模为一个管理者类型的责任模型，其中John Smith对波士顿2176服务小组负责。

**例：**Mark Thursz是皇家医学院的成员，这可以被建模为一个职业注册类型的责任模型，其中Mark Thursz对皇家医学院负责。

**例：**John Smith允许Mark Thursz做内窥镜检查，这可以被建模为一个患者许可类型的责任模型，其中Mark Thursz对John Smith负责。

**例：**圣·玛丽医院与公园路地区卫生当局签订有1996/1997年度内窥镜检查方面的执行合同，这可以被建模为一个内窥镜检查服务类型的责任模型，其中圣·玛丽医院对公园路地区卫生当局负责，该责任的有效期是从1996年1月1日到1997年12月31日。责任的子类型可以提供一些额外的信息，比如所包括的具体工作内容以及在合同期内的执行次数。

**建模原则：**只要为一个拥有超类型的类型定义了特征，就应该考虑将这些特征放在该超类型上是否有意义。

正如前面这些例子所显示的，从组织结构中抽象出责任模型为我们带来更广阔的空间，可以应付更多的情形，而模型的复杂度并没有增加。基本的模型与图2-7有着相同的结构，惟一的变化是将组织替换成团体。

## 2.5 责任知识级

然而，责任的引入带来了复杂性，因为责任的类型比组织结构的类型要多得多。相应地，定义责任类型的规则将变得更加复杂。

这种复杂性可以通过引入知识级来管理。知识级将模型分成两部分：操作级和知识级。操作级包括责任、团体以及它们之间的关联关系；知识级包括责任类型、团体类型以及它们之间的关联关系，如图2-9所示。

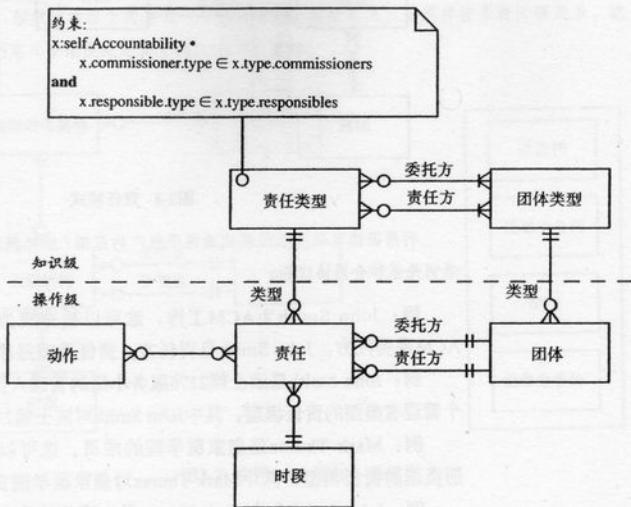


图2-9 责任的知识级和操作级

知识级对象定义了操作级对象的合法配置，根据相应的责任类型与团体类型的约束，只能在允许的团体之间创建责任。

在操作级，模型记录该领域每天所发生的事件；在知识级，模型记录

控制着结构的各种通用规则。知识级的实例支配着操作级的实例的具体配置。在该示例中，责任实例（实际团体之间的关联）受到责任类型与团体类型之间的关联的约束。

[24]

**例：**区域子公司又可细分成各个部门。这可以由一个区域子公司结构的责任类型来处理，其中委托方是区域子公司，责任方是部门。

**例：**患者许可可被定义成责任类型，其中委托方是患者，责任方是医生。

请注意如何用团体类型映射来替代团体子类型化。这是Odell在[3]所讲的“强力类型”的一个实例，其主要出现在用映射来定义子类型时。团体类型与团体的子类型有很紧密的关系，因为子类型“区域子公司”必须有它的类型作为团体类型“区域子公司”。从概念上讲，可以把团体类型的实例看成是与团体子类型相同的对象，虽然用主流的编程语言无法直接实现该概念。团体类型是团体的一种强力类型。通常，我们只需要映射(mapping)或者子类化(subtyping)这两者中的一个。但是，如果子类型具有特殊的行为并且强力类型也拥有自己的特征，那么子类型和到强力类型的映射都是需要的(Odell用一种特殊的表示法来表示这种情况[3])。

[25]

知识级和操作级相互对应但并不完全一样，因为父节点与子节点的映射在知识级是多值的而在操作级是单值的。这是由于操作级刻画的是参与责任的实际团体，而知识级刻画的是参与责任类型的可能的团体类型。这种用多值的知识映射来表示单值的操作映射的可能类型的方式是很常见的。

知识级和操作级是模型所具有的一个共同特征，虽然我们经常不显式指明两者之间的差异。我将两者明确地分开是因为我发现这样做有助于理清建模思路。本书中有大量关于操作级与知识级的例子，尤其是在第3章。

**建模原则：**将模型清晰地分解成操作级和知识级。

多数的数据建模人员用“元模型”一词来描述知识级，我对这样的用法不是很赞同。因为元模型也可用于建模技术，其中，元模型包括了诸如类型、关联、子类型以及操作等(例如Rational软件的统一方法[1]的元模型)。知识级则不同，它不描述用于操作级的各种图符。因此我只在刻画用于描述模型语言(图符的语法)的模型时才使用“元模型”一词<sup>②</sup>。

责任模型表达了一些相当原始的抽象概念。就像在爬山的过程中，在高原反应到来前，我们会停下来做适当的准备。虽然对象模型的结构非常简单，但在知识级的实例中却隐藏着大量的知识。仅仅做这些工作还不足

<sup>②</sup>当然，如果我定义一个图表，它展示责任类型和团体类型的实例，那么知识级可当成是该图表的元模型。这种图表对于刻画复杂的责任类型网络可能会有用。

以实现对象模型，必须对知识级加以初始化。初始化知识级是一个受限的因而也是简单的编程过程，目的是要有效地配置系统。虽然简单，但仍然是属于程序设计，因此应当考虑如何加以测试。

丰富的知识级也会影响系统间的通信。如果两个系统要进行通信，它们不仅必须共享对象模型，还必须拥有同样的知识对象（或者至少是如5.4节所述的知识级间的一些等价物）。接下来最终我们会遇到下面这个问题：如果责任类型的数量非常庞大，使用图2-9的结构更容易些还是扩展图2-5为每种责任类型配置关联关系更容易些？问题本身的复杂度是回避不了的，我们只能比较权衡类型结构和知识对象，判断哪个模型更简单。

需要注意的是，对关系加以类型化并不是可以应用在所有的团体关系上。比如，生物学上的父子关系就不能作为责任类型的一个实例，因为没有团体之间的责任关系，也不存在固有的责任期限，但是，法定的监护关系属于责任类型。

## 2.6 团体类型泛化

正如模型自身所体现的那样，其功能的强大不容置疑，但如果能够增加一些有用的变化，则可以使得模型具有更强的适应能力。这些变化可以用在任何使用了知识级/操作级划分方式的模型中。

我们来分析一下Edwards医生，他是一个普通的分析模式实践者（简称GP）。使用图2-9所示的模型，我们可以把他看成是GP或者是医生，但不能两者都是。针对医生所定义的而又适用于GP的所有责任类型都必须重复描述。我们可以采用多种技术来缓解这个问题。其中一种方法就是允许团体类型拥有子类型或者超类型关系（如图2-10所示）。这样就引入了团体类型的泛化概念，其作用与类型的泛化相似。泛化使得针对责任类型的约束发生了变化，因此既需要考虑团体的类型（来自类型映射），又需要考虑团体的超类型（来自所有的类型映射）。

图2-10给出了团体类型的一个单继承关系。多继承可以通过允许超类型的映射是多值的方式来实现。此外，图2-10只支持单一分类，即如果Edwards既是一个GP，又是一个儿科医生的话，我们只能通过创建一个特殊的“GP/儿科医生”团体类型来刻画，其中GP和儿科医生都是超类型。多分类允许在团体类型的泛化结构之外，赋予团体多种团体类型，这可以通过允许映射到团体的类型可有多个取值的方式来实现。

有关知识级和操作级之间的内在关系的许多讨论与元模型建模中的对象和类型的关系是类似的。

[26]

[27]

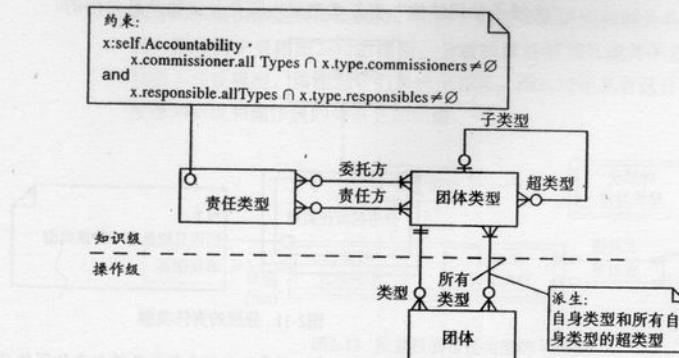


图2-10 允许团体类型拥有子类型和超类型

给团体类型增加泛化使得定义知识级更容易。

## 2.7 层次型责任

要实现责任模型所给出的柔性结构，还需要在一些责任类型的约束规则上多下功夫。比如，图2-3所示的组织结构定义了一个严格的层级序列：分公司被划分成若干区域子公司、区域子公司被划分成若干部门、部门被划分成若干销售办事处。定义一个区域子公司结构的责任类型是可能的，但是如何才能保持图2-3中的严格规则呢？

首要的问题是图2-3刻画了一个层次型的结构，而责任模型没有相应的规则来约束这样一种结构。这可以通过提供附加有约束规则的责任类型的子类型的方式来加以处理（如图2-11所示）。这些附加的约束规则和惯例性的约束规则一起共同作用于责任类型，从而保持操作级结构的层次特性。类似的责任类型子类型可用于实现一个有向无循环图结构。

利用图2-11，通过一系列的责任类型，我们可以刻画图2-3所示的情形。责任类型“区域子公司结构层1”对分公司负有区域性的责任，“区域子公司结构层2”对区域子公司负有部门性的责任，以此类推。这种方法确实可行，但是有点笨拙。另一种方式是使用一种如图2-12所示的分级的责任类型，采用这种方式就只有一种“区域子公司结构”责任类型，其中，各级分别映射到相应的团体类型——分公司、区域子公司、部门、销售办事处。这种模型使得可以很容易地增加新的级别的责任类型，并便于修改所需结构中的级别。分层的责任类型捕获团体的责任关系并组成一个层次型的模型，而分级的责任类型用于捕获其中具有固定次序的团队责任关系。

“区域子公司结构”责任类型可以既是分级的又是分层的。

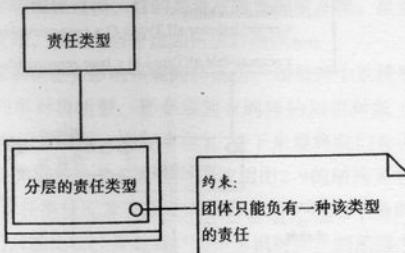


图2-11 分层的责任类型

新增的约束的意思是通过该类型的责任关系所连接起来的团体必须组成一个层次型的模型。

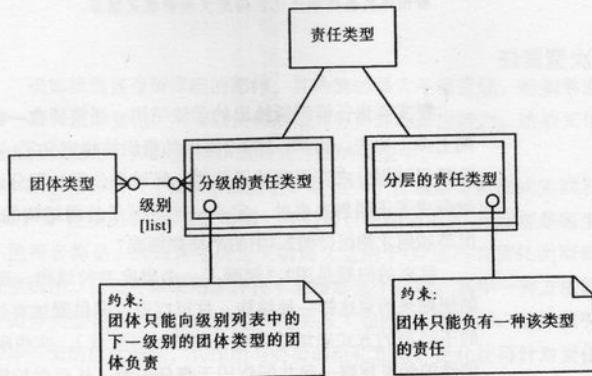


图2-12 分级的责任类型

分级责任支持固定的级别关系，例如销售办事处、部门和区域子公司之间的隶属关系。

遵循“契约式设计”[4]的原则，为责任类型所定义的约束规则与附加的约束规则一起共同作用于子类型。对于存在分级责任类型的情况，约束包括了对超类型的约束，并使得委托方和责任方的映射显得多余。这样的一种想法可用图2-13所示的模型来表达。需要特别强调的是，图2-12并非不正确。分级的责任类型是一种非常好的责任类型子类型，因为施加于责任类型的约束规则仍然会适用于分级的责任类型，而其中的责任方和委

托方的映射仍然继续得以保持，虽然它们需要从级别映射关系中派生而来的。我倾向于采用图2-12的模型。分级的责任类型可能并不总是必需的，只要不违反模型，添加起它们来也很容易。图2-12还具有这样的优点：它使得知识级和操作级的关系更加明确。

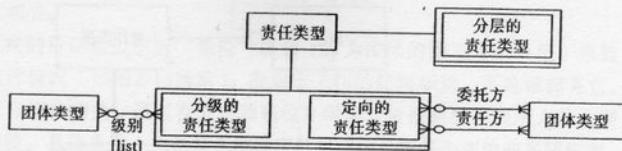


图2-13 反复权衡责任类型的子类型

一种组织责任类型层次的更好的方法。

## 2.8 操作范围

责任提供了一条描述团体之间如何相互联系的颇有价值的途径。责任的类型描述了它们所具有的关系的种类；通常还需要其它一些更详细的信息来具体阐明责任的内涵。举例来讲，1997年，一名医生被聘用为肝脏方面的外科医生，其任务是为伦敦东南部地区实施20例的肝脏移植手术；某医院的糖尿病治疗小组应红十字会的要求，为马萨诸塞州西部地区依赖胰岛素的糖尿病患者提供医疗服务。

诸如此类的详细说明称为责任的操作范围（如图2-14所示）。每个操作范围定义了责任团体所担当的某部分责任。很难用抽象的方式来列举操作范围的具体属性。因此，我们认为责任应当拥有大量的操作范围，每一操作范围是某种描述实际特征的子类型。

**例：**某位要在一年内负责伦敦东南部地区20例肝脏移植手术的外科医生，他参与一个雇佣责任模型，该责任有一个协议范围：数量是20；肝脏移植方面的协议；地点是伦敦东南部地区。

**例：**糖尿病治疗小组和红十字会参与一个责任模型，其临床治疗范围如下：其中“观察概念”对象是依赖胰岛素的糖尿病患者；地点是马萨诸塞州西部地区。

**例：**芳香咖啡机制造公司（ACM）与印度尼西亚咖啡出口公司（ICE）签订有每年3000吨爪哇咖啡和2000吨苏门答腊咖啡的合同。这可以用ACM与ICE之间的责任模型来描述，其责任期限是一年，资源类型有两种：3000吨/年的爪哇咖啡和2000吨/年的苏门答腊咖啡。

30

31

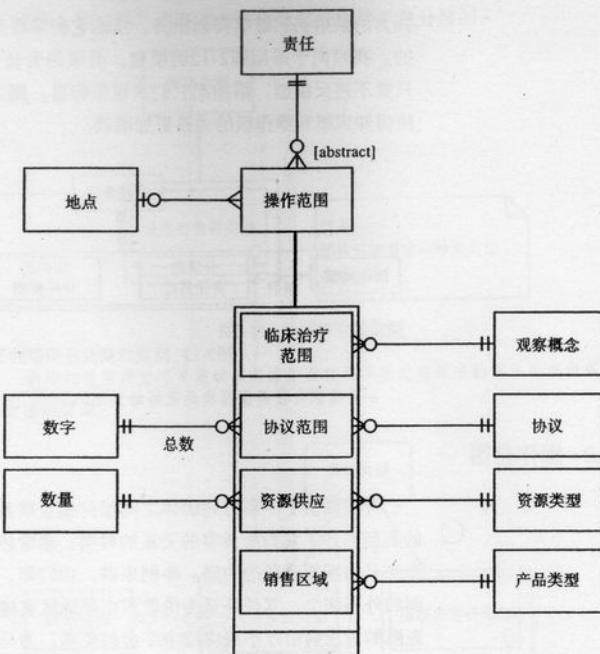


图2-14 操作范围

在创建责任模型时，操作范围规定了应承担的具体责任。这些操作范围可用于工作说明。

例：John Smith为ACM销售1100和2170系列的大容量咖啡机。他在新英格兰既销售1100系列又销售2170系列，在纽约仅销售2170系列。因此他与ACM之间存在雇佣类型的责任模型，其销售区域是：新英格兰，1100系列；新英格兰，2170系列；纽约，2170系列。

在针对特定的组织使用操作范围时，需要识别其所具有的操作范围类型及属性。要用很抽象的方式来概括操作范围是非常困难的，但地点好像是一个各种操作范围都具有的属性。操作范围的子类型如果很多的话，其间的继承关系可能会形成层次结构。在特别复杂的实例中，你可能会见到在知识级放置有操作范围类型<sup>①</sup>，用以说明哪些责任类型具有哪些操作范围类型。

① 在这种情况下，操作范围类型的实例必须与操作范围的各种子类型匹配。操作范围类型因此是操作范围的一个强力类型[3]。

## 2.9 职位

通常，对于人的操作范围——他们的职责（包括他们的诸多责任中的许多责任）——已经预先在工作说明中做了具体规定。当某人辞去工作时，替代者将继承所有的职责。也就是说，职责与工作相关，而不是与人相关。

我们可以通过引进“职位”概念（作为团体的第三个子类型）来处理这种情况（如图2-15所示）。隶属于工作的任何职责，无论谁拥有它，都只与职位相关。通过在人员和职位之间建立责任模型来表示人员占据该职位。具体来讲，就是某人在规定的时间内负责某个职位的各项职责。

例：Paul Smith是大容量产品开发小组的负责人，这可以通过设立大容量产品开发小组负责人这样一个职位的方式来描述，该职位和大容量产品开发小组（一个团体）之间有管理责任关系，而Paul Smith和该职位之间有另一个责任关系（聘用关系）。

例：某医院做器官移植手术的外科医生这一职位，其工作要求是每年完成50例肾脏和20例肝脏移植手术。该职位与医院之间存在责任关系，协议范围涉及50例肾脏移植手术和20例肝脏移植手术。

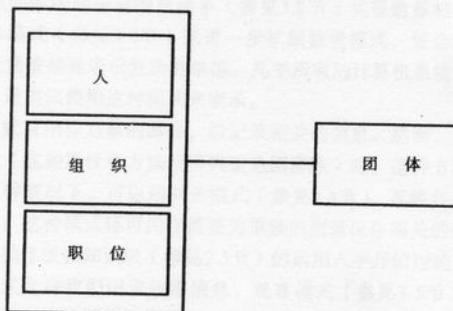


图2-15 职位

职位这个模式适用于当责任及其操作范围由职位来定义的情况，并且当职位的拥有者变化时，职位并不发生变化。人员和职位之间具有聘用关系。

不应该在所有时候都使用职位，因为它们引入了更多的间接联系，会增加操作级的复杂性。只有职位中体现了一些稳定的重要职责，而人员经常变换职位的情况下才使用职位概念。对于模型中所有的职责都与某个特定的人相关的情况，就没有必要再使用职位这一概念。

## 参考文献

1. Booch, G., and J. Rumbaugh. *Unified Method for Object-Oriented Development*. Rational Software Corporation, Version 0.8, 1995.
2. Cairns, T., A. Casey, M. Fowler, M. Thursz, and H. Timimi. *The Cosmos Clinical Process Model*. National Health Service, Information Management Centre, 15 Frederick Rd, Birmingham, B15 1JD, England. Report ECBS20A & ECBS20B, <http://www.sm.ic.ac.uk/medicine/cpm>, 1992.
3. Martin, J., and J. Odell. *Object-Oriented Methods: A Foundation*. Englewood Cliffs, NJ: Prentice-Hall, 1995.
4. Meyer, B. "Applying 'Design by Contract,'" *IEEE Computer*, 25, 10 (1992), pp. 40-51.

## 第3章

# 观察和测量模式

---

现实世界中，大多数的计算机系统都记录有关现实世界各种对象的信息。这些信息以记录、属性、对象或者其它表现形式存在于计算机系统之中。典型的方式是将信息片断记录成某个对象的一个属性。比如，将我的体重为185磅的信息记录成“人”这种类型的一个属性。本章将讨论该方法为什么会失效，并将提出更精确的方法。

首先，我们将讨论数量模式（参见3.1节）——由数字和相关的单位组合而成的一种类型。通过将数字和单位组合起来，我们能够更准确地对客观世界进行建模。将单位建模成为对象，再将数量模式与之关联，我们就可以描述如何按照一定的转换率（参见3.2节）实现数量的换算。可以用复合单位模式（参见3.3节）来进一步扩展数量模式，复合单位模式可以用组成元素清楚地表示复杂的单位。几乎所有的计算机系统都涉及到数量；币值总是应该使用这种模式来表示。

数量模式可用作对象的属性，以记录相关的信息。然而，当类型存在大量的属性（这些属性和方法使得类型急剧膨胀）时，这种方法就会失去效用。在这种情况下，可以用测量模式（参见3.4节），其将各种测量视为相应的对象；这种模式还可用在需要为单独的测量保存相关的信息时。本章我们将从操作级和知识级（参见2.5节）的运用入手开始讨论。

测量模式允许我们记录数量信息。观察模式（参见3.5节）为了处理数量信息，进一步扩展了该模式；进而可以在知识级中进行观察概念的子类型化（参见3.6节）。为一个观察记录相关的观察方案模式（参见3.7节）通常是很基本的要求，这样做可以使临床医生更好地解释观察的结果，并更好地确定观察的准确度和灵敏度。

为了扩展观察模式需要大量的小模式。观察所发生的时间和被记录的时间之间的差异可以用双时间记录模式（参见3.8节）来刻画。通常，保存已经确认为错误的观察记录非常重要，这就需要用到被否决的观察模式（参见3.9节）。观察结果最让人头疼的是处理其可靠性，因为在一般情况下所记录的主要内容都是有关对象的假设。临床观察、假设与推理（参见

3.10节)的子类型化是处理该问题的一种途径。

有关观察模式的大多数陈述是用一个诊断过程来实现的。我们基于其它的观察来推断某个观察。关联观察模式(参见3.11节)可以用于记录作为证据的观察以及用于诊断的各种知识。

上述的各种模式可加以组织并用于刻画观察概念。要理解它们是如何发挥作用的,就很有必要看一看观察过程模式(参见3.12节),可以采用基于事件的技术来对其进行建模。

很少有专业像医药行业那样对测量和观察有如此复杂的要求。本章中的模型来自一个医疗保健系统(英国国民医疗服务制度的Cosmos项目)的建模工作。在该项目中,在一起工作的是一个由医生、护士和分析员组成的联合小组,他们所面对的是一个非常难处理的领域。在本书里,我们并没有把Cosmos模型完整地包括进来,如果你对其感兴趣,请参阅参考文献[1]中所述的完整模型。在这里所谈到的一些模式还能应用到其它领域:第4章将讨论这些模式是如何应用到公司财务分析中的。

**关键概念:**数量、单位、测量、观察、观察概念、现象类型、关联函数、被否决的观察、假设

### 3.1 数量

在目前的计算机系统中,记录测量信息的最简单和最常用的方法是将数字记录到为特定的测量而设计的域中(如图3-1所示)。该方法的问题之一是仅仅用一个数字来表示人的身高是很不够的。如果说我的身高是6或者体重是185,那究竟表示什么意思呢?要搞清楚这些数字的真实含义,需要使用单位。方法之一就是将单位引入到关系名称中,比如体重(磅)。单位可以澄清数字的具体含义,但这样一种表示方法显得十分笨拙,另外的问题就是记录人员必须为信息使用正确的单位。如果某人告诉我他的体重是80公斤,那我将如何记录?事实上对于某些行业,尤其是医药行业,要求非常精确地记录所测量的结果——不能多,也不能少;如果记录的是经过转换的结果,即使转换方法是很确定的,也显得不够忠实。

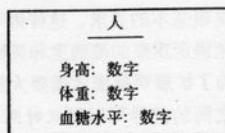


图3-1 数字类型的属性

这种方法没有特定的单位。

在这个上下文中，数量是一个很有用的概念。图3-2展示一个由数字和单位组成的对象类型（比如，6英尺或者180磅）。数量还包括适当的算术运算和相应的操作。比如，其中的一个操作能够实现数量相加功能，但是在相加之前会检查单位是否一致（34英寸就不能跟68公斤直接相加）。数量是用户界面可以解释和显示的“完整值”[2]（一个简单的打印操作可以显示数字和单位）。这样，数量就可以像整型或者日期类型那样，广泛地应用到对属性的定义中。

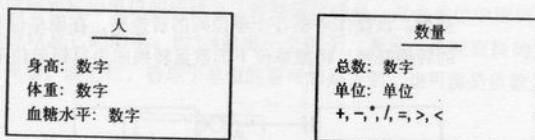


图3-2 用数量将测量定义为属性

数量可以应用到需要用单位的地方。

**例：**我们可以将数字185和单位磅组合成为数量来表示体重为185磅。币值总是应该表示为数量，用币制作作为单位（在本书中我使用“金钱”一词）。使用数量可使你很容易地处理多币制的情况，从而摆脱单一币制的局限性（只有在我自己所使用的财务程序中我才使用单一币制）。金钱对象还可以控制数量的表达方式。在财务系统中，如果使用了浮点数来表示币值，常常会产生四舍五入的问题；采用数量可以用定点数来定义数量属性。

[37]

**例：**80美元可以用由数字80和单位美元所组合成的数量来表示。

数量的使用是面向对象分析的一个重要特征。许多建模方法都明确区分属性和关联。在模型中，关联是连接两个类型，而属性则是根据某个属性类型来包含某个值。问题是：什么时候用属性而什么时候又用关联？通常，在绝大多数的编程环境中，属性类型都是典型的固有类型（比如整型、实型、字符串型、日期型等）。属性与关联的这种区别对于像数量这样的类型不一定适用。某些建模人员认为数量应该作为关联来使用（因为它不是一种固有类型），而另外一些建模人员认为数量应当作为属性来看待（因为它是一个自包含的且被广泛加以采用的类型）。在概念性建模中并不太在意你究竟采用了什么方法，最重要的是你是否寻找并使用了形如数量这样的类型。因此，我并不太关心属性与关联之间的区别，也不想陷入有关的争论中。（我强调这一点是因为我发现在我所见到的绝大多数模型中，很少用到像数量这样的类型。）

**建模原则：**当多个属性与可能会在几个类型中使用的行相关时，就把这些属性组合成新的基础类型。

### 3.2 转换率

我们可以很好地利用模型中明确表示的各种单位。这些单位的用处之一是使我们可以实现从一个单位到另一个单位的数量转换。如图3-3所示，我们可以在单位之间使用“转换率”对象，并赋予数量对象一个操作：convertTo (Unit)，该操作能够将给定单位的数量转换成为新的单位下的数量。该操作考察各个单位间的转换率，在原单位和目标单位间寻找合适的转换路径，将源单位下的数量转换成为目标单位下的数量。



图3-3 给单位增加转换率

例：通过定义一个1:12（英尺比英寸）的转换率，我们可以将英尺转换为英寸。

例：转换率具有复合传递性，比如，英寸与毫米之间的转换率是1:25.4，那么通过将这个转换率与上一个转换率复合，可以将英尺转换成毫米。

转换率可用在很多地方，但并不是万能的。比如，摄氏温度与华氏温度之间的转换就不是简单的乘法关系，而需要通过更复杂的运算来实现。在这种情况下，需要单独的实例方法（参见6.6节）。

如果是多个单位之间存在转换关系，我们可以使用多维单位转换阵列。比如，力的单位转换阵列[MLT<sup>-2</sup>]。对于非国际单位制单位，可以使用梯形单位转换阵列。利用多维单位转换阵列和梯形单位转换阵列，可以实现单位之间的自动转换，虽然建立这样的转换阵列需要花点功夫。

要注意：日和月之间的转换关系是不确定的，因为每月的天数是不确定的。

如果存在候选的转换途径，可以在测试用例中加以利用——可以用各种途径去验证转换过程。

对于币值，其单位是特定的货币，转换率并不固定。我们可以通过给转换率指定可用的时间范围来解决这一问题。

当进行单位转换时，既可以使用在这里所阐述的转换率，也可以使用

9.4节所介绍的场景。如果转换关系频繁变化，而且转换时需要了解一系列的转换，就可以考虑采用场景。否则，采用简单的转换率是再好不过的。

### 3.3 复合单位

单位可以是原子单位，也可以是复合单位。复合单位由原子单位组合而成，如“平方英尺”或者“米/秒”。特定的操作可以将针对原子单位的转换率应用到复合单位的转换上。需要记住的是，复合单位中用到了哪些原子单位以及它们的幂。图3-4给出一个例子，展示一个较直接的复合单位转换模型。请记住，各原子单位的幂可能是正数，也可能是负数。

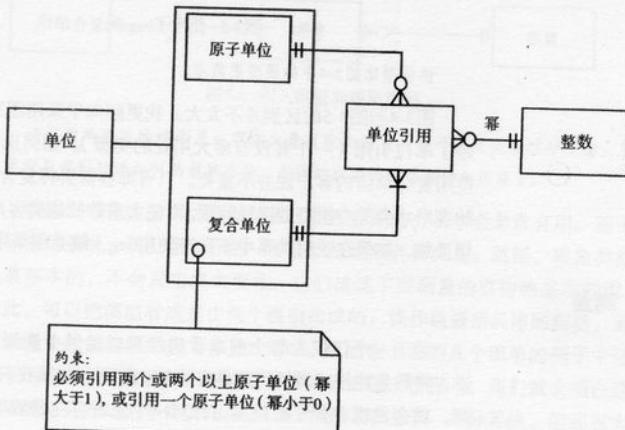


图3-4 复合单位

该模型可用于描述加速度和类似的现象。

例：150平方码的一个区域面积可以表示成一个数量，其数值是150，单位是“9平方英尺”的复合单位。<sup>Θ</sup>

该模型的一个变种是使用一种叫“bags”（袋）的映射描述方式。和我们通常所讲的集合不同，“bags”允许我们在一个映射中多次使用同一个对象（如图3-5所示）。“bags”尤其适用于当拥有某种具有单一数字属性的关系时。

<sup>Θ</sup> 平方码和平方英尺之间的换算关系为：1平方码=9平方英尺，所以150平方码=150×9平方英尺。——编辑注

例：由重力所引起的加速度可以表达成一个数量，其数值是9.81，单位是由“米”做分子、“二次方秒”做分母的复合单位。

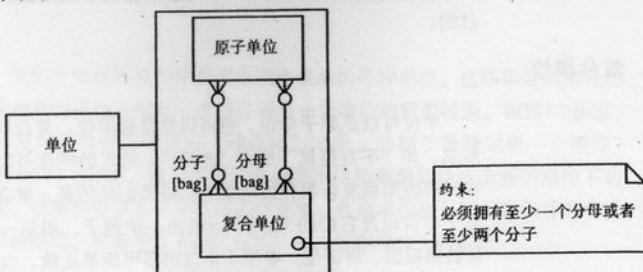


图3-5 使用了bags的复合单位

该模型比图3-4中的模型更简洁。

40

图3-4和图3-5的区别并不太大。我更倾向于采用图3-5的模型，因为它回避了单位引用（一个并没有多大用处的类型）。如何选择这些模型对大多数使用复合单位的客户说并不重要。只有那些需要将复合单位分解成原子单位的客户才会关心模型的选择问题，而绝大多数的这类客户都只需要书面表述。很显然，如果在映射关系中不允许使用bag，就必须采用图3-4的模型。

### 3.4 测量

对于一个仅仅为每个前来看病的患者提供少量测量结果的医疗部门来讲，将数量建模成属性可能是有效的。然而，如果我们考虑到医疗的方方面面，就会发现在整个医院里，针对一个患者存在着成千上万种可能的测量。为每种测量定义一个相应的属性，意味着一个患者对象可以拥有成千上万的操作——其接口的复杂性是难以想像的！解决方法之一是考虑将所有可以被测量的不同事物（身高、体重、血糖水平……）都作为测量对象，并引入一种叫“现象类型”的对象类型（如图3-6所示）。这样，每个患者就可能拥有多种测量，每种测量是一个带有数量属性的现象类型；而该患者只需要拥有一个专门针对所有测量的属性。测量处理的复杂性就被转移到了对成千上万的测量和现象类型的实例的查询上。进而，我们可以给测量增加别的属性，以描述其它的一些信息，比如谁来做、什么时候做、在哪里做等。

例：John Smith身高6英尺，该信息可以通过一个测量来表达，其中人员是John Smith，现象类型是身高，数量是6英尺。

例：John Smith的肺部呼气峰流量是每分钟180升（肺部呼吸空气的

[41]

数量和速度), 该信息可以用一个测量来表示, 其中人员是John Smith, 现象类型是肺部呼气峰流量, 数量是每分钟180升。

例: 一个混凝土样本能够承受每平方英寸4000磅的压力, 用一个测量来表示时, 人员就变成了混凝土, 而现象类型是所能承受的压力(压强), 数量是每平方英寸4000磅。

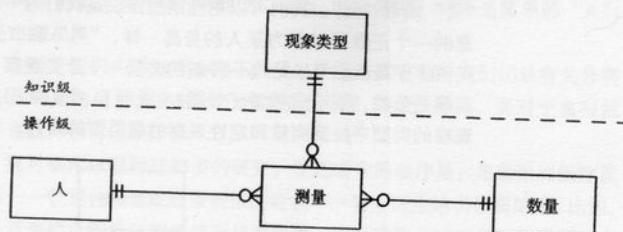


图3-6 引入测量和现象类型

该模型所适用的情形是: 存在大量可能的测量会使得人员类型过分复杂。现象类型是我们知道如何测量的事物, 具体的内容位于模型的知识级中。

该模型中存在简单的划分关系, 其在后续的分析中会非常有用。基于该模型的系统, 其日常工作的一部分就是创建各种测量。然而, 现象类型是最基本的, 不会发生多大变化, 它们描述了要测量的事物的基本知识。因此, 可以把模型看成是由两个级别构成的: 操作级囊括具体的测量, 而知识级中包含现象类型(参见2.5节)。虽然在上面的几个简单的例子中还看不出该划分的重要性, 但随着建模活动的进一步深入, 我们就会明白这两种二级划分方式是非常实用的。(虽然图3-6给出了一条分界线, 但还有大量的信息没有描述, 我们将把它们留到后面的图表中去描述; 此外, 我还有一个习惯, 就是喜欢将知识级的内容放在图的上半部分。)

**建模原则:** 操作级中的对象会经常发生变化。它们的配置由很少发生变化的知识级来约束。

**建模原则:** 如果某个类型拥有多种相似的关联, 可以为这些关联对象定义一个新的类型, 并建立一个知识级类型来区分它们。

我们可以有选择性地为现象类型增加测量单位, 并用数字来取代测量中的数量。我自己更倾向于在测量中使用数量, 这样就能够更方便地支持“一个现象类型, 多种单位”的情形。也可以为现象类型建立一个单位集合, 用来检查所输入的测量单位的正确性, 并为用户提供一个可选择的单位列表。

### 3.5 观察

针对患者除了需要大量定量的描述外，还需要一些定性的描述，比如性别、血型以及是否患过糖尿病等。我们无法使用属性来定义这些描述，因为这样的描述实在太多了，于是就需要用到一种类似于测量的结构。

[42]

设想一下我们是如何记录一个人的性别，性别存在两种可能的值：“男”或者“女”。我们可以把性别想像成是我们的一个测量目标，就像任意的一个正数可以作为某人的身高一样，“男”和“女”是可能的测量结果值。于是我们可以给出一种新的类型——分类观察，它类似于测量，但其值是分类，而不是数量（如图3-7所示）。我们还引入另外一种新的称为观察的类型来扮演测量和定性观察的超类型的角色。

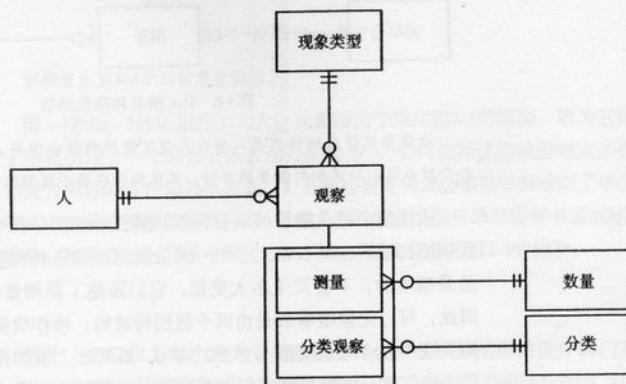


图3-7 观察和分类观察

该模型支持定性测量，例如血型A。

通过图3-7，可以认为性别是现象类型的实例，“男”和“女”是分类的实例。要刻画一个性别为男性的人，我们可以创建一个观察对象，其分类是男性，现象类型是性别。

我们现在不得不考虑一下如何才能保证特定分类只能用于特定的现象类型。高、中等、矮可能是身高这种现象类型的可选分类值，而A、B、A/B、O可能是血型这种现象类型的可选分类值。这可以通过给出分类与现象类型之间的映射关系的方式来实现。这里就出现了一个有趣的问题，即从分类到现象类型的映射集合的基数。我们可能会问，血型A是否会对应不只一个现象类型？一种答案是“当然如此”：比如在Childs-Pugh等级

测定中，我们用A（合理的）、B（中等的）、C（差的）来表示肝功的正常性。但是这又引出另外一个问题，A的具体含义是什么？如果仅仅从字面上来理解“A”的含义，其结果是该映射是多值的，而且分类与现象类型无关。只有当现象类型产生自某个定性观察时，分类的含义才是清楚明确的。可采取的办法是使分类与现象类型之间的映射为单值，此时分类是在现象类型的上下文环境中加以定义的；也就是说，并不是简单的“A”，而是血型A。

43

这对于我们来讲究竟有何不同呢？单值方式有利于我们记录有关分类的一些有用的信息，比如对于肝功检查结果，A比B要好，而对于血型就不存在这样的优劣顺序。

我对临床过程做过初步的研究，发现通常的顺序是：患者走到医疗设备前——收集各种有关患者病情的症状——临床医生给出诊断结论。比如，某患者来到后就首先抱怨感到异常口渴，而且体重也减轻了，还尿频（多尿症），这些症状引导医生给出该患者患了糖尿病的诊断结论。对于该次诊断，还有两件事情非常重要。第一，仅仅简单地说明患者是糖尿病患者是不够的，医生还必须清楚地记录下得出这样的诊断结论的各种症状；第二，医生不能无中生有地做出这样的推演，不能利用很随意的症状进行很随意的推演，医生必须基于临床知识加以诊断。

设想将该诊断过程放到迄今为止我们所拥有的模型中。患者痛苦于体重的减轻，我们可以通过如下方式来捕获该信息：存在这样的现象类型——“体重的变化”，其可能的取值是“增加”、“减轻”或者“保持稳定”。同样地，存在“糖尿病”这样的现象类型，其可能的取值是“有”或者“没有”。显然，我们可以通过给观察对象设置合适的递归关系的方式，来刻画观察之间的这种关联关系（如图3-8所示）。这样，我们就能够刻画糖尿病及其症状之间的关联关系。我们还需要记录诊断性的知识，如体重减轻与糖尿病之间的关联关系。使用图3-7所示的模型，我们很难刻画这样的关联关系。只有在创建了观察对象之后，体重减轻这样的现象类型与减轻分类之间的关联关系才建立起来。我们需要通过某种方式来说明体重减轻这样的事实，其确确实实存在着，并不会因为观察是否建立而消亡，这样的事实应该放在知识级中。具体来讲，可以通过建立从分类到现象类型之间的单值映射的方式来实现。（在3.11节将做进一步论述。）

44

很明显，这种使分类与现象类型之间形成单值映射是一种强制性的行为。其将分类信息转移到知识级中，并重命名为现象（如图3-9所示）。现象为现象类型定义了可能的取值。

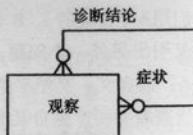


图3-8 递归关系刻画了症状与诊断结论之间的关系

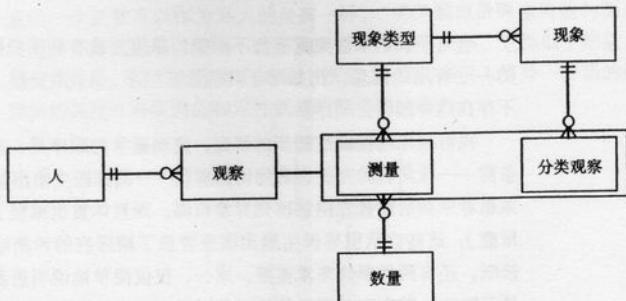


图3-9 知识级中的现象（原来叫分类）

将定性的描述（例如血型A）放到知识级中使得可以按照规则来使用它们。

例：“某人的血型为A”这样的事实可以由一个人的分类观察来简要说明，其中现象是“血型A”。“血型A”这一现象与血型的现象类型相关联。

例：我们可以建造一个汽车的分类观察模型来刻画汽车油量的不足。其中，现象类型是油量的多少，可能的现象（取值）是“过量”、“刚好”、“不足”，而观察将汽车和油量不足联系起来。

图3-9中的模型可以很好地刻画一种现象类型具有多个取值的分类观察。但很多观察可能只有两种取值——“有”与“没有”，而不是一个很大的取值范围。糖尿病就是一个很好的例子：糖尿病要么是处于“有”状态，要么是处于“没有”状态。清楚地记录“没有患糖尿病的迹象”固然重要，但如果能记录下“体重没有减轻”的事实又何尝不是明智之举。（如果某个前来看病的患者表现出糖尿病的症状，但体重没有减轻，就不能简单地判断他患有糖尿病。此时并不用强调体重究竟是增加了或者保持着稳定，只需要说明没有减少就可以了。）事实上，我们可以对任何一种现象处于“没有”状态加以刻画，尤其是对于带排除性的假定性诊断。因而，图3-10所示的模型允许任何分类观察可以具有“有”和“没有”的取

值。增加观察概念作为现象的超类型。这样就可以将糖尿病看成是一个观察概念，而与现象类型无关。

例：我们可以通过将John Smith的一个“有”状态的观察与糖尿病的观察概念关联起来的方式来记录John Smith患有糖尿病这一事实。

例：我们可以用一个观察（其中将观察中的人换成隧道）和一个混凝土碎裂的观察概念来表示隧道中发生的混凝土碎裂现象我们还需要为观察追加一个特征，来说明在隧道中混凝土碎裂所发生的具体位置。（在医学上，对于某些观察概念，还需要通过解剖来定位）。

### 3.6 观察概念的子类型化

图3-10引入一种超类型关系，允许观察概念的泛化的存在。这在医疗领域非常普遍而且非常实用，因为这样的话，就可以创建具有任意通用度的观察对象。如果某个观察是在子类型处于“有”状态下创建的，那么可以认为其所有的超类型都应该处于“有”状态。相反，如果某个观察是在某个子类型处于“没有”状态下创建的，那么就表示超类型的状态是不确定的。观察处于“没有”状态表示所有的子类型也处于“没有”状态。因而，“有”状态沿超类型层次向上传播，而“没有”状态则向下传播。

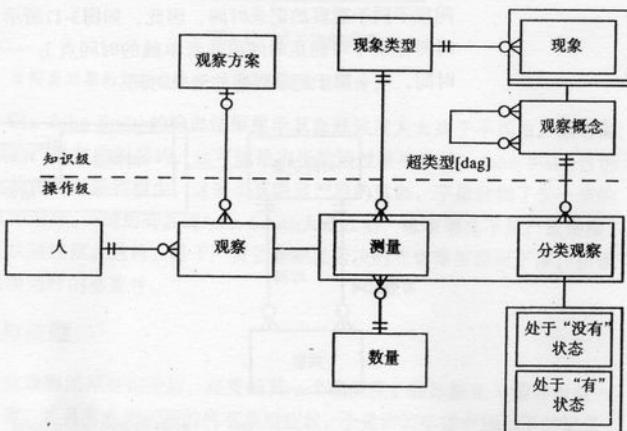


图3-10 观察概念的两种状态（“有”与“没有”）

发现一个现象的“没有”状态和发现一个“有”状态一样，都是有价值的。

例：糖尿病具有两种子类型：类型I和类型II。如果John Smith患有类型I的糖尿病，则可以说John Smith患有糖尿病。

例：我们常说血型A是多形态的，因为它可以分为A1和A2两种子类型。其它的血型不是多态的。

### 3.7 观察方案

观察方案是刻画观察模型的一个非常重要的概念（处于知识级）——其给出进行观察的具体方法。我们可以采取将温度计放在口中、腋下或者直肠里等方式来测量一个人的体温。通常，各种方式所得到的最后结果是同一类型的测量，然而，至关重要的是要记录究竟采取的是什么方式。某种奇特的观察结果往往可以通过对所采用的具体技术加以分析来做出解释。因此，在医疗保健领域中，普遍使用的方法就是记录下什么测试被用于记录这些观察。

[46]

观察方案的用处之一在于它能够用来确定一个测量的精确度和灵敏度。有关精确度与灵敏度这样的一类信息可以跟随测量本身一起加以描述，但通常是基于用于特定观察的观察方案来加以阐述，将这些信息保存在观察方案中可以使获取它们时更加容易。

### 3.8 双时间记录

通常，观察都会有一定的适用期限，一旦过期，就没有什么用处。时间段不同于观察的记录时间，因此，如图3-11所示，每种观察都需要两个时间记录（可能是时间段或者单独的时间点）：一个用于记录观察的有效时间，一个用于记录观察的记录时间。

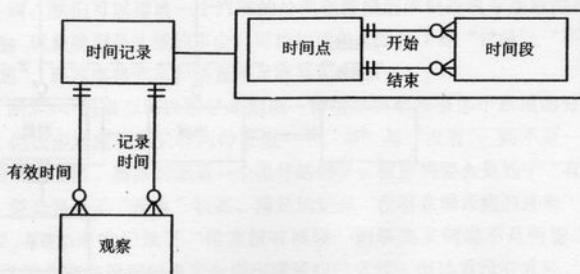


图3-11 针对观察的双时间记录

允许一个时间记录可以是时间段，也可以是单独的时间点。绝大部分事件的发生时间（即有效时间）和记录时间是不一样的。

例：在1997年5月1日的一次会诊中，John Smith告诉他的医生，他在六个月前曾胸口疼痛，并持续了一周左右的时间。医生就将其记录为一个观察：出现胸口疼痛（观察概念）症状。发生时间（有效时间）是开始于1996年11月1日、结束于1996年11月8日的一个时间段。记录时间是1997年5月1日（时间点）。（请注意：某些记录大致时间点的方法在此处会非常有价值。）

47

### 3.9 被否决的观察

在进行观察的时候，不可避免地会犯一些错误。但是，在医疗记录中，却不能简单地将它们抹掉。治疗过程可能曾经是基于这些错误的方案而进行的，通常会涉及到法律责任问题。要处理这样的情况，在发现观察结果过去和现在都是错误的时候，可以将其归为被否决的观察，如图3-12所示。（要注意这种观察与另一种过去是正确的但现在不再正确的观察（如一条已治愈的断臂）之间的不同。一条已治愈的断臂现在不会是被否决的，但它的有效时间被给定一个结束时期。）被否决的观察必须与否决掉它们的观察相关联。

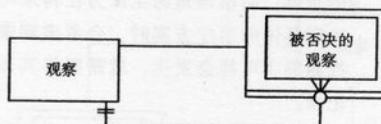


图3-12 被否决的观察

当需要完整的审查线索时，不能将观察清除掉。

例：John Smith的验血结果显示其血球浓度大大高于平均值。这可能是因为严重贫血引起的，也可能是由于饮酒过度导致的。John Smith告诉医生他喝酒时喝得很少，这表明其患有严重的贫血，于是就做了进一步的测试和治疗。6周后却发现John Smith大量饮酒，这表明并不是严重贫血，而是饮酒过度。这样，基于严重贫血的被否决的观察需要保留下来，以说明后续治疗的必要性。

48

### 3.10 临床观察、假设与推理

当观察结果被记录后，还要给其一个确信度。临床医生可能会面对一个患者，其具有患糖尿病的所有典型症状。于是该医生就会做如下的记录：我认为该患者可能患有糖尿病。但是直到相关的检验结果出来后她才能给出确切的诊断结果；况且，对于多数疾病，即使做过相应的化验，也不一定能提供百分之百的确定性。记录这类信息的方法之一是给观察设置可能

性系数，但这种方法太模糊，而且看起来也不是很自然。另一种方法是采取两分类法：即分为“临床观察”和“假设”两类（如图3-13所示）。两种分类有着很细微的差别：临床观察是临床医生已经获得的证据，可能会作为治疗的基本依据；而假设会导致做更进一步的检查或者测试。

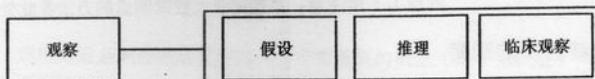


图3-13 临床观察、假设与推理

**例：**当患者呈现口渴、体重减轻和多尿等临床观察症状时，表明其患有糖尿病。然而，基于这些症状，临床医生只是做出患有糖尿病的假设，并安排做空腹情况下的血糖水平测量。根据化验结果才能对假设加以确认（也可能加以否决）。

观察的两种子类型，临床观察与假设，都表征了患者的当前状态的观察。而推理是医生认为在将来可能会发生的观察。通常，医生在决定具体的治疗方案时，会考虑到未来可能发生的情况。如果这种推测确实可能会发生，就需要将其当成是一个附加的临床观察来加以记录。

**例：**如果某人患了风湿热，紧接着又患风湿性关节炎，就有患心内膜炎的可能。这种可能会被记录为“预计会患心内膜炎”这种推理，而相应的治疗方案就会基于该推理来加以制定。

观察的确定性问题是Cosmos项目中的一个热点问题。项目组和评审人员在其上所做的更改和所花的时间比模型中的其它任何部分都要多。最后的模型从临床医生的角度给出了最自然的表达方式。设置可能性的方式对于一个科幻爱好者来讲是有意义的，但对于临床医生来讲显然不是这样（他们事先可能会问这样的问题：“我应该怎样解释可能性0.7与0.8之间的不同之处？”）。虽然在应该选择使用哪种分类时会存在一些困难，但是，最终只有由富有经验的临床医生组成的小组才有资格按照一种本能的方式，为项目组在该问题上做出有用的结论；小组中的专业分析人员只能在其中提出一些常规的结论。

49

### 3.11 关联观察

在这里，我们将寻求一种途径来记录隐藏在诊断后面的证据链。基本思路是允许将观察相互连接起来（某患者口渴预示其患有糖尿病），对观

察概念亦是如此（口渴预示着患糖尿病）。从而，我们会看到知识级和操作级相互映射（如图3-14所示）。这些映射通过关联（展示知识级的概念是如何应用到操作级中的）连接起来。这样的话，不仅观察与观察概念之间会产生连接关系，在证据和结论之间也会产生连接关系。因而，当我们说患者口渴预示患者患有糖尿病时，我们使用了，而且应该明确地记录我们确实使用了口渴与糖尿病之间的常规性的连接关系。图3-14显示我们是如何构造相应的类型来既支持观察和观察概念，又支持操作级（关联观察）和知识级（关联函数）中的连接关系。

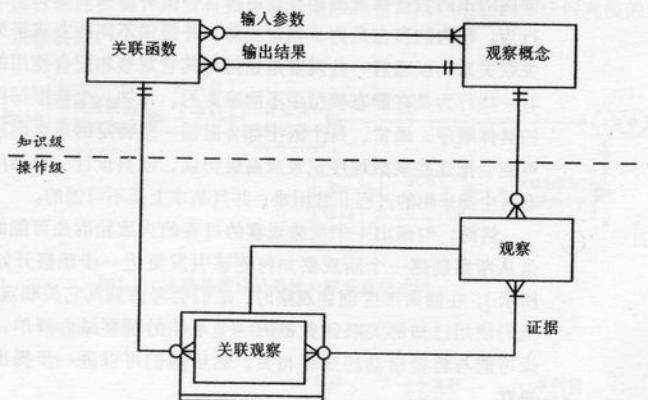


图3-14 观察之间的连接

针对一个患者的实际的证据链被记录在操作级中，而知识级描述可能的连接关系。 [50]

**例：**临床医生观察到某个患者呈现如下症状：体重减轻、口渴、尿频。他基于这些证据，构造了一个有关糖尿病的关联观察（并做出相应的假设）。该关联观察与一个关联函数相连，而该关联函数的输入参数是关联概念——体重减轻、口渴和尿频，输出结果是糖尿病。

**例：**如果我的汽车无法启动，而且车灯也不亮，这两条观察结果就可以作为电池没有电（关联观察）的证据。汽车无法启动、车灯不亮以及电池没有电是所有由一个关联函数连接的观察概念。

请注意，知识级与操作级中的元素之间的关联关系并不是完全对等的。比如，关联观察是观察的子类型，但关联函数就不是观察概念的子类型。在操作级，将关联观察当成是观察的子类型是一件很自然的事情，因为确实需要一个特定的观察来支持各种证据。而在知识级，则记录了

各种带有参数与结论的规则。一个观察概念可能作为多个关联函数的输出结果，但一个特定的观察只能有一个关联集合作为证据。

### 3.12 观察过程

本章的前面一些小节主要是在阐述观察模型的一些静态元素：什么是观察和测量，以及如何按照一种通用的途径去刻画它们，以支持对临床医生实际工作的分析。在建模过程中，我们发现一个很有意思的问题，即我们可以构想出一个非常通用的静态模型，但是在具体操作时，根据部门的不同会有很大的差异。当然，一个静态模型中确实隐含着大量的行为。行为的存在是为了创建观察，并提供不同的方式来浏览观察间的关联关系，以理解一些观察是如何与其它观察相配合使用的。然而，也有一些行为是在静态模型中不能隐含的，比如一个典型部门创建的观察的具体顺序。通常，每个医生都会遵循一些特定的观察顺序，而部门也可能会把这些观察顺序记录成高级协议，但要设计一个通用的而且是所有医生都采用的过程非常困难，并且基本上是不可能的。

然而，勾画出一个实施观察的过程的大致轮廓是可能的。我的做法是从考察创建一个新观察如何能够引发更进一步观察开始（如图3-15所示）。在临床医生创建观察时，他们会考虑到其它关联观察的可能性。他们使用已知的关联函数提出一个可能的观察概念清单，这些观察概念可能与将要启动的观察相关。然后他们可以进一步提出所需的其它观察。

51

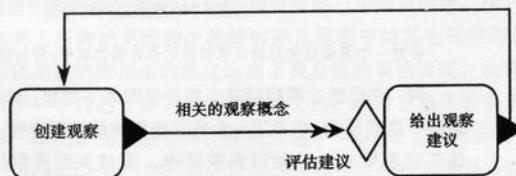


图3-15 创建一个观察引发更进一步的观察

更进一步的观察由知识级给出。

在图3-15中，将并发触发器规则标记为“相关的观察概念”。在事件图中使用触发器规则有两个目的：一是用于指示因果关系，在考虑业务过程时就是如此；但如果要考虑的因素更多，就会涉及到其另外一个目的。任何操作都具有输入和输出，连接两个操作的触发器必须能够描述如何从

触发（源）操作获取输出，并传递给被触发的操作作为输入。在多数情况下，这是微不足道的，尤其是当它们是同一对象时（如图3-15所示，从“给出观察建议”到“创建观察”之间的触发器）。然而，在寻找“相关的观察概念”过程中，可能就会变得非常复杂。

当触发器规则过于复杂时，我们可以用别的事件图来表示它们（如图3-16和图3-17所示）。首先是寻找所有的关联函数，其输入参数包括初始观察的观察概念；然后对每个关联函数加以评估；对于通过评估的关联函数，就将其输出添加到结果集合中。既然这些事件图描述了一个触发器规则查询，则所有操作都只是附属物而已，不能改变任何对象的可观察状态。

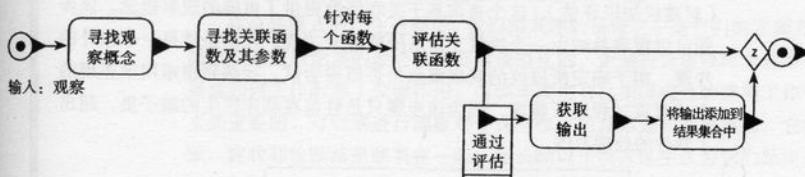


图3-16 本事件图描绘寻找关联观察的查询

本图基于图3-15的并发触发器或图3-18中的操作。

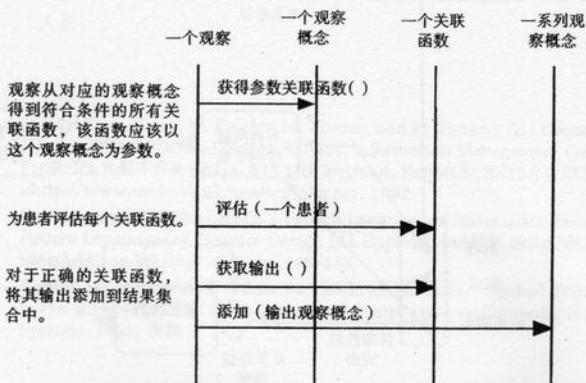


图3-17 寻找观察中所隐含的各种可能的观察概念的交互图

该交互图支持图3-16。

如果触发器规则查询非常复杂，也可以如图3-18所示的那样将该查询表示成一个操作。总之，两种方法都可行。

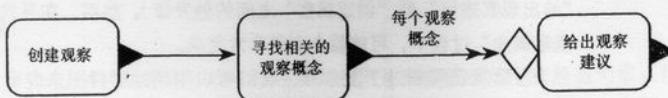


图3-18 将该查询清晰地表示成一个操作

该图等价于图3-15，也就是说：既可以用操作来刻画查询，也可以把它们看成是触发器的一部分。总之，要力求简洁紧凑。

即使在查询之后，在给出新的观察建议之前，还会增加一个控制条件（对建议加以评估）。这个查询基于关联函数提供了可能的观察概念。该步骤可以很容易地由一个决策支持系统软件来实现。控制条件是一个额外的步骤，用于确定所提议的观察概念是否值得尝试。我感到很难用很正规的方式对该过程进行建模，因为该步骤只是驻留在临床医生的脑子里，超出了软件的处理范围。

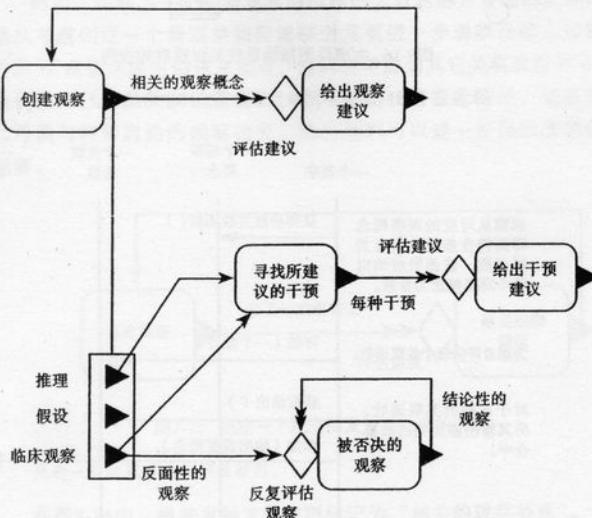


图3-19 观察的处理过程的事件图

该图用类似的用于干预和否决的触发器扩展了图3-15。

图3-19包含额外的触发器，它们是在推理和临床观察活动中产生的。针对给出干预建议的触发器的工作方式与针对前述病例的方式类似。我们所给出的干预建议，是由临床医生事先评估过的。这是为了进一步强调这样一个事实：虽然任何观察都可以导出进一步的观察，但只有临床观察和推理（不是假设）才导出干预。（干预是一种活动，用于预防在患者身上可能发生的病变与危险。）该触发器查询按照与知识级类似的方式工作，但包含启动函数（我们将在8.7节对其做简要介绍）。 [53]

图3-19中的最后一个触发器表明：通过临床观察可以发现其它观察存在的矛盾，从而否决掉它们。同样地，在这里需要用到关联函数，但这一次我们主要是要找出矛盾之处。一旦某个观察（可能是一个假设）被否决掉，则由其支持的进一步的观察也必须被重新考虑。

在这些模式的产生过程中，我们感兴趣的事情之一是它们的发现方式。虽然在这里所论述的最终结果都是规范化的，但是行为建模对于理解这些概念如何工作起到了关键作用。由临床医生自己来完成这些建模工作也是至关重要的。对观察进行抽象对于这些模式的总结是非常重要的：它将迹象、症状和诊断结果联系在一起，而长期以来临床医生认为它们是完全不相关的。只有通过建模，临床医生才能发现这些抽象的概念。如果是由软件工程师提出类似的抽象概念，我担心临床医生会质疑其正确性。产生这样的质疑是有根据的，因为软件工程师一般都不会具有很深厚的医疗卫生知识。最好的概念模型是由领域专家建造的，他们通常是最好的概念建模人员。 [54]

## 参考文献

1. Cairns, T., A. Casey, M. Fowler, M. Thursz, and H. Timimi. *The Cosmos Clinical Process Model*. National Health Service, Information Management Centre, 15 Frederick Rd, Birmingham, B15 1JD, England. Report ECBS20A & ECBS20B <<http://www.sm.ic.ac.uk/medicine/cpm>>, 1992.
2. Cunningham, W. "The CHECKS Pattern Language of Information Integrity," In *Pattern Languages of Program Design*. J.O. Coplien, and D.C. Schmidt, ed. Reading, MA: Addison-Wesley, 1995, pp. 145-155.
3. Thursz, M., M. Fowler, T. Cairns, M. Thick, and G. Gold. "Clinical Systems Design," In *Proceedings of IEEE 6th International Symposium on Computer Based Medical Systems*, 1993.