华章原创精品
HZ BOOKS

那些事儿

UML

邱郁惠 编著

你和UML不得不说的那些事儿

应该知道的基础知识 需要了解的核心秘密 亟待掌握的实战技能

机械工业出版社
China Machine Press

华章原创精品

那些事儿

UML

邱郁惠 编著

机械工业出版社
China Machine Press

本书通过引用UML规格书里的图例和定义详细介绍了UML的特色和发展历史，六类UML图，元模型的说明，并且以StarUML为示范，讲解UML工具如何落实UML概念，还介绍了活动图、控制节点、交互图、生命线、用例图等内容。

本书可作为高等院校计算机专业的教材和参考书，也可作为各类希望了解UML的人员的参考书。

图书在版编目（CIP）数据

UML那些事儿/邱郁惠编著. —北京：机械工业出版社，2008.12
（原创精品系列）

ISBN 978-7-111-25132-3

Ⅰ. U…　Ⅱ. 邱…　Ⅲ. 面向对象语言，UML－程序设计　Ⅳ. TP312-62

中国版本图书馆CIP数据核字（2008）第144656号

机械工业出版社（北京市西城区百万庄大街22号　邮政编码　100037）
责任编辑：王　玉
北京京北印刷有限公司印刷 · 新华书店北京发行所发行
2008年12月第1版第1次印刷
186mm × 240mm · 18印张
标准书号：ISBN 978-7-111-25132-3
定价：39.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换
本社购书热线：（010）68326294

前　言

本书写得很辛苦，不过也很过瘾，我几乎把UML的元模型（metamodel）都拆解、摸透了。我相信，本书对于想深入研究UML，或者是想开发或扩展UML工具的读者，相当有帮助。

整本书中，很多图的图题标有[ex]或[meta]，[ex]表示引用UML规格书里面的范例(example)，[meta]则表示这张图是元模型图。有些元模型图引用自UML规格书，有些是元模型图的局部或组合。每个主题第一次出现时，都标注定义。

引用UML规格书里的图例和定义主要是因为UML规格书中很多图例不容易懂，却有很多值得学习的地方，所以我想通过引用来解释这些经典范例。

第1章介绍了UML的特色和发展历史，是最精彩有趣的一章，期望给您一个有趣的阅读开端。如果您已经学过UML，可以跳过第2章，直接从第3章开始读起；否则，最好别遗漏第2章。因为从第3章开始介绍UML的每一个元素，所以很容易迷失在细节的枝蔓中。而第2章刚好给了六类UML图一个初步的认识，您可以从中知道每一张图的用途，以及常用的概念和图示。

一旦开始阅读第2章之后的章节，我希望您能够耐心地依照章节顺序往下阅读，特别是类图的章节中，有许多概念是后面章节的基础。跳着阅读可能会使有些说明不容易理解。如果依序那么你应该会有愈读愈轻松的感觉。所以，请务必耐心地读完讲述类图的第3~5章，那是本书的核心，也是最繁杂和难懂的部分。另外，请别匆忙跳过第9章的杂项，它占了10%的比例，特别是构造型（stereotype）的部分，要尽可能耐心读完才好。

书中有许多关于元模型的说明，这是与其他书籍最大的不同处。看懂了元模型也就看懂了UML规格书的核心，也可以通过UML图示真正掌握UML语法的根源。不过，元模型确实比范例难懂。如果理解有困难，请先行理解定义、范例及细节说明。

有时，本书会以StarUML为示范，让您看到UML工具如何落实UML概念。除了可以具体体会外，也能够增加阅读的趣味。StarUML是一套免费的UML工具，我推荐您安装它来试试，这有助于学习UML，或者将UML用到项目中。

目　录

前言

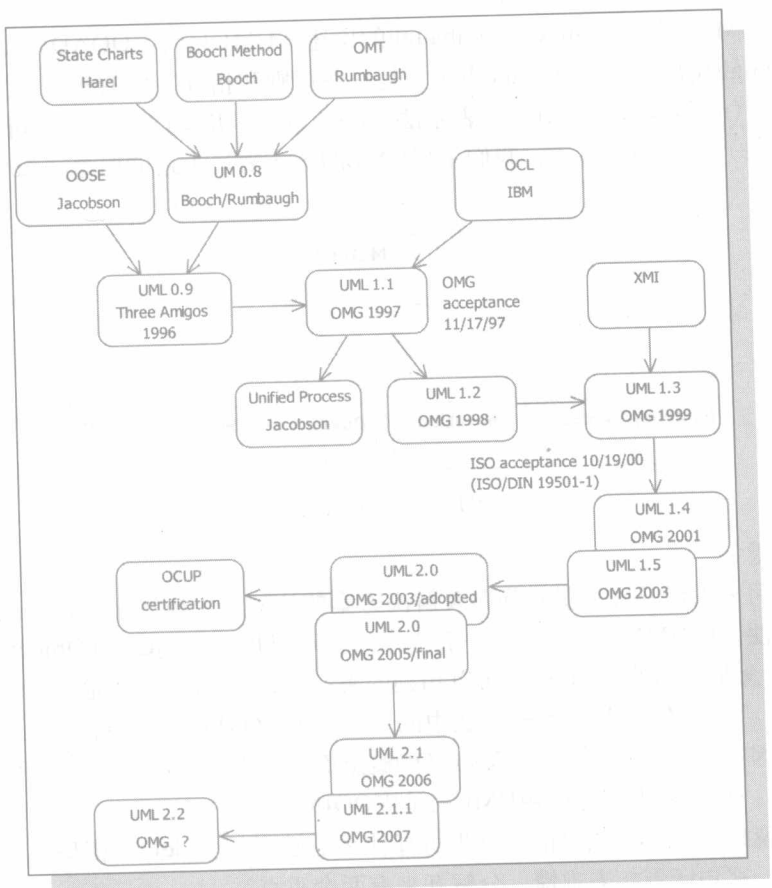
第1章　UML概述 ……1
1.1　语言与方法 ……2
1.2　图形语言 ……7
1.3　国际标准 ……10
1.4　面向对象 ……14
第2章　六类UML图 ……17
2.1　类图 ……18
2.1.1　类 ……18
2.1.2　可见性 ……19
2.1.3　关联 ……20
2.1.4　多重性 ……22
2.1.5　聚合与组合 ……22
2.1.6　泛化 ……23
2.1.7　依赖 ……24
2.1.8　接口 ……24
2.1.9　注释 ……25
2.2　对象图 ……26
2.3　包图 ……27
2.3.1　包 ……27
2.3.2　元素导入 ……27
2.3.3　包导入 ……28
2.3.4　包合并 ……28
2.4　活动图 ……29
2.4.1　动作与控制流 ……30
2.4.2　对象节点与对象流 ……31
2.4.3　活动参数节点 ……31
2.4.4　引脚 ……32
2.4.5　起点与终点 ……32
2.4.6　合并 ……33
2.4.7　判断 ……33
2.5　序列图 ……34
2.5.1　交互 ……34
2.5.2　生命线 ……35
2.5.3　执行发生 ……36
2.5.4　消息 ……36
2.5.5　终止 ……37
2.5.6　一般次序 ……37
2.5.7　状态不变式 ……38
2.6　用例图 ……39
2.6.1　用例与执行者 ……39
2.6.2　包含关系 ……40
2.6.3　扩展关系 ……40
2.6.4　扩展点 ……41
第3章　类图（一） ……43
3.1　根基 ……44
3.1.1　元素 ……44
3.1.2　关系 ……47
3.1.3　有向关系 ……49
3.1.4　注释 ……51
3.2　名称空间 ……54
3.2.1　具名元素 ……55
3.2.2　名称空间 ……56
3.2.3　可见性种类 ……58
3.2.4　装包元素 ……60
3.2.5　元素导入 ……63
3.2.6　包导入 ……66
3.3　多重性 ……70
3.3.1　多重性元素 ……70
3.3.2　类型与类型元素 ……72
3.4　表达式 ……73
3.4.1　值规格 ……74

3.4.2 表达式 ……74
3.4.3 不透明表达式 ……75
3.4.4 文字规格 ……76
3.4.5 实例值 ……76
3.5 约束 ……77
第4章 类图（二） ……81
4.1 实例 ……82
4.1.1 实例规格 ……82
4.1.2 槽 ……85
4.2 类元 ……87
4.2.1 定义 ……87
4.2.2 泛化 ……89
4.2.3 可重定义元素 ……92
4.3 特征 ……95
4.3.1 定义 ……96
4.3.2 结构特征 ……98
4.3.3 行为特征 ……99
4.3.4 参数 ……99
4.3.5 参数方向种类 ……100
4.4 操作 ……102
4.5 类 ……106
4.5.1 定义 ……107
4.5.2 性质 ……110
4.5.3 关联 ……112
4.5.4 聚合种类 ……119
第5章 类图（三） ……121
5.1 数据类型 ……122
5.1.1 定义 ……122
5.1.2 基本类型 ……123
5.1.3 枚举与枚举文字 ……124
5.2 包 ……125
5.2.1 定义 ……125
5.2.2 包合并 ……128
5.3 依赖 ……133
5.3.1 依赖与具名元素 ……133
5.3.2 使用关系 ……137
5.3.3 许可 ……139
5.3.4 抽象 ……140
5.3.5 实现 ……142
5.3.6 替代和类元 ……143
5.4 接口 ……145
5.4.1 实现和行为类元 ……145
5.4.2 接口定义 ……147
第6章 活动图 ……153
6.1 流程 ……154
6.1.1 活动节点与活动边 ……154
6.1.2 控制流 ……160
6.1.3 对象流 ……161
6.2 节点 ……162
6.2.1 活动 ……163
6.2.2 执行节点与动作 ……167
6.2.3 对象节点 ……168
6.2.4 活动参数节点 ……170
6.3 动作 ……171
6.3.1 引脚 ……172
6.3.2 输出引脚与输入引脚 ……173
6.3.3 值引脚 ……176
6.4 控制节点 ……177
6.4.1 定义 ……177
6.4.2 起始节点 ……178
6.4.3 终止节点与活动终点 ……179
6.4.4 合并节点 ……183
6.4.5 判断节点 ……184
第7章 交互图 ……187
7.1 交互 ……188
7.2 消息 ……190
7.2.1 消息与消息端 ……191
7.2.2 消息种类与消息性质 ……195
7.2.3 事件发生与执行发生 ……199
7.2.4 一般次序 ……201
7.3 生命线 ……203

7.3.1 定义 ……203
7.3.2 状态不变式 ……209
7.3.3 终止 ……211
第8章 用例图 ……213
8.1 用例与类元 ……214
8.2 关联与泛化 ……218
8.3 执行者 ……223
8.4 包含关系 ……224
8.5 扩展关系 ……228
8.6 扩展点 ……232
第9章 杂项 ……239
9.1 基本类型 ……240
9.2 UML图 ……240
9.2.1 图框 ……240
9.2.2 13类图 ……242
9.3 共同行为 ……244
9.3.1 行为 ……245
9.3.2 行为类元 ……250
9.3.3 活动 ……252
9.3.4 不透明表达式 ……253
9.4 构造型 ……254
9.4.1 辅助与焦点 ……257
9.4.2 元类 ……258
9.4.3 类型与实现类 ……259
9.4.4 工具 ……260
9.4.5 派生 ……261
9.4.6 精化 ……263
9.4.7 追踪 ……263
9.4.8 实例 ……264
9.4.9 创建 ……268
9.4.10 调用 ……269
9.4.11 发送 ……270
9.4.12 责任 ……271
9.4.13 创建 ……271
9.4.14 销毁 ……272
9.4.15 框架 ……273
9.4.16 模型库 ……274
9.4.17 实现 ……275
9.4.18 建造组件 ……276
9.4.19 脚本 ……277

第1章　UML概述

1.1　语言与方法
1.2　图形语言
1.3　国际标准
1.4　面向对象



什么是UML？如果用一句话来回答这个问题，即：UML是一种获取了国际双标准的图形语言，专门用来表达OOAD（Object-Oriented Analysis and Design）的生成。这句话包含了UML的四项特色：

1）UML是一种语言，不是一种方法。

2）UML是图形语言，不是编程语言，也不是自然语言。

3）UML通过了两项国际标准。

4）UML深具面向对象（Object-Oriented）色彩。

在接下来的各节中，我们会依次说明UML的四项特色。

1.1 语言与方法

UML不是OOAD开发方法，它只是OOAD表示法。我在网络上看到许多OOAD开发方法的课程，八成的授课时间都在教授UML，这是有问题的。OOAD开发方法少不了表示法，而UML是目前最流行的表示法，这意味着UML是OOAD开发方法的组成元素，但是不等同于OOAD开发方法。

如果套用UML之父——James Rumbaugh的论述，UML只约占OOAD开发方法中四分之一。James Rumbaugh指出开发方法（method）是一些规则和指导方针的集合，包含四项重要的元素：建模概念（modeling concept）、表示法（notation）、开发过程（development process）和经验法则（rule）。如图1-1所示，我们采用UML的图示表达上述的概念，该方法有四个重要的组成元素。

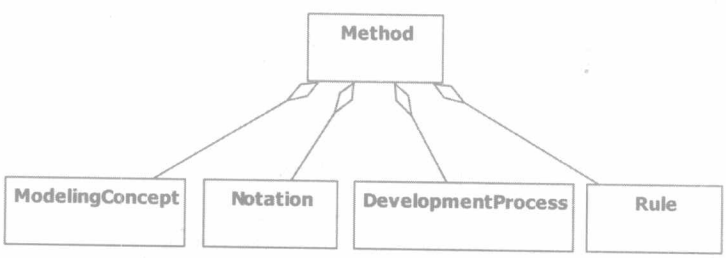


图1-1 方法的组成元素

1. 建模概念

建模概念用来表示开发人员对问题及解决方案的认知。建模概念应该与表示法分开，不局限于任何表示法。以UML为例，它是一套表示法，可以用来表达OO（Object-Oriented）概念，但是它与OO概念是分离的。OO概念可以用别的表示法，而UML也可能用来表达非OO的概念。

如图1-2所示，UML只是众多表示法中的一种，而OO只是一种建模概念。UML与OO有密切的关联的原因在于，UML就是为了表达OO概念而设计的表示法，但是这不代表UML只能用来表达OO概念，也不代表仅能使用UML来呈现OO概念。

什么是建模概念呢？在此之前，要先解释什么是模型（model）。模型代表着系统某一个抽象层面的表达。这句话看起来难懂，分解开来有两个重点：

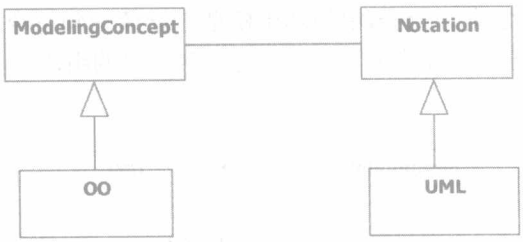


图1-2 建模概念与表示法

1）模型不等于系统，它只是系统的一个样貌。正如人与照片之间的关系，照片仅捕捉到了人的一个样貌，但是照片不等于真实的人。

2）一个模型呈现一个抽象层面。系统是复杂、难以理解的，针对一个系统，我们会构建出多个不同抽象层面的模型，试图通过这些不同角度的模型来趋近对系统的认知。继续以人为例，照片是一个模型，一种静态的抽象层面，若更清楚地认识这个人，可能需要录像，这是另一种抽象层面，因为录像记录了人的动态行为。

再以UML模型为例，一个分析层面的UML模型必须有较高的抽象程度，不能暴露实现细节。如果是设计层面的UML模型，就必须表达出足够的实现细节，以便程序设计师能够按照设计图编写代码。所以，从分析设计来看，不同阶段的UML模型会呈现出不同的抽象程度。

现在回过头来解释建模概念。建模概念是建构模型的基础概念，有了基础概念，才能建构出好的模型。再以人为例，采用照相技术可以捕捉到人静态的外貌，采用录像技术可以捕捉到人的外部行为。可以这么说，照片、影音都是一种模型，而照相技术、录像技术则是支撑它们的基础概念，懂了照相技术和录像技术，才能拍出好的照片、录下好的影音。

所以，有了OO作为基础的建模概念后，就可以构建出符合OO概念的模型。如果采用UML作为表达工具，便可以构建出符合OO概念的UML模型。

最后来理清模型与元模型（metamodel）的概念。我们可以针对一个系统建构模型，试想，可否把一个“开发方法”当成一个分析设计的对象，为它构建模型呢？答案当然是肯定的。这种特殊的模型称为“元模型”，以便与一般的模型区分开来，如图1-3所示。

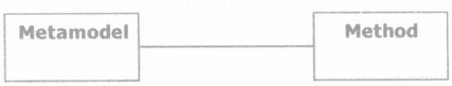


图1-3 元模型与开发方法

另一个问题是，如果可以用UML来呈现一般系统的模型，那么可否同样使用UML来呈现开发方法的元模型呢？答案当然也是肯定的。

元模型跟UML有什么关系呢？如果将元模型的概念进一步运用在UML本身，也可以将UML视为一个分析设计的对象，为它构建元模型，再使用UML作为自身元模型的表示法。从语言的特性来看，UML有自己的词汇和语法，词汇相当于UML的元素，而语法则通过UML元模型呈现。

对于UML而言，它有自己的元模型，用来表达重要的语法限制。在构建UML模型时，如

果遵守UML元模型，那么它是一个合法的UML模型，就像一句符合英文文法的句子一样。所以，如图1-4所示，UML有自己的元模型，而且是非常重要的组成元素之一，它规范了UML中的所有元素。

图1-4 元模型与UML

UML元模型相当复杂，我们选取了一小片段元模型图，如图1-5所示。这个元模型片段在规范用例（use case）之间可以有扩展关系（extend）以及其他一些琐碎的细节。

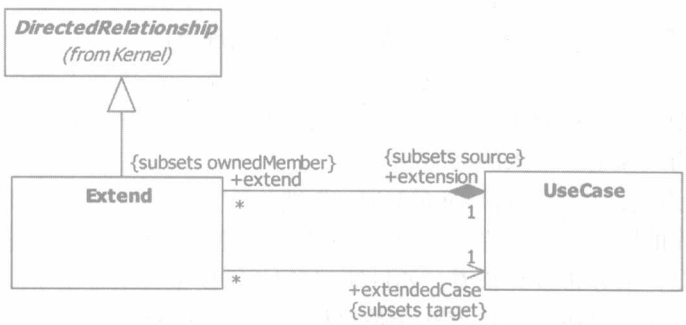


图1-5 扩展关系

简单来说，UML元模型规范了UML所有元素之间的关系，而且UML元模型本身就是一些复杂的UML类图（class diagram），再配合一些文字说明。

2. 表示法

虽然说建模概念与表示法彼此独立、互不影响，但是它们又彼此互补，两者结合一起才能够呈现出具体的模型（model），如图1-6所示。有了具体的模型，开发人员才能够沟通、表达、反思对系统的认知。

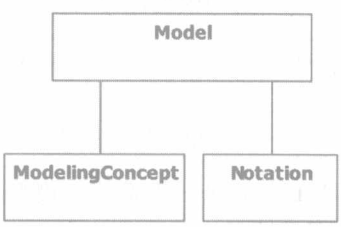


图1-6 模型

单一的建模概念无法发挥其大功效，集合数个相关的建模概念就可以完善地表达一个观点。因此，UML提出了13类图（diagram），每一类图集合了数个相关的建模概念，共同表达一个特定的观点。这13类图也是一般开发人员初学UML时，首先要认识的部分。13类UML图分为结构图（structure diagram）和行为图（behavior diagram），如图1-7所示。

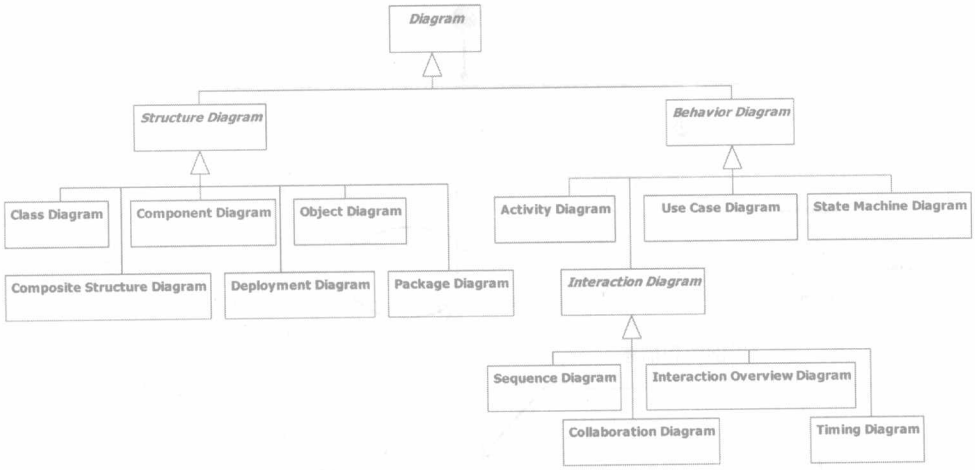


图1-7 UML图

结构图主要表达系统的静态结构（static structure），行为图则表达系统的动态行为（dynamic behavior）。

3. 开发过程

UML本身就是OO概念的表示法，也可以说它结合了建模概念与表示法。虽然将UML定位为表示法，但它其实与建模概念有相当密切的关系。可是和开发过程相比，我们可以十分笃定UML未涉及开发过程，同时UML也不局限于任何开发过程。换句话说，UML适合各种开发过程。

这也是UML不等同于OOAD开发方法的最重要的一项证据，OOAD开发方法必须涵盖一套开发过程。但是，我们翻遍了UML规格书，也找不到一丝一毫的开发过程。UML的标准中，确实剔除了开发过程。

但是，UML模型或者UML图是开发过程中的重要生成。也就是说，搭配UML的开发过程中，除了需要说明开发步骤外，还必须指出每个开发步骤将生成什么样的UML图。例如，我在项目中就经常使用如图1-8所示的简易开发过程，一开始并行构建用例图和类图，随后才构建序列图，接着按图编码。开发过程中的每一个步骤都生成特定的UML图。

4. 经验法则

各行各业中的“老鸟”都积累了许多经验法则，所以如果想迅速脱离“菜鸟”身份，最好偷学几招有用的经验法则。一个好的、成熟的开发方法可以积累许多有用的经验法则，不仅可以降低学习曲线，也可以提高生成的品质。

目前，最蓬勃发展的经验法则就是模式（pattern）。每一个模式用来解决一个特定的问题，所以它是一个很好的经验法则，多学有益。关于模式的书籍、文章很多，有设计模式、分析模式、架构模式、UML模式，也有用例模式，几乎开发过程中的各个阶段都有模式可学，都有经验法则。

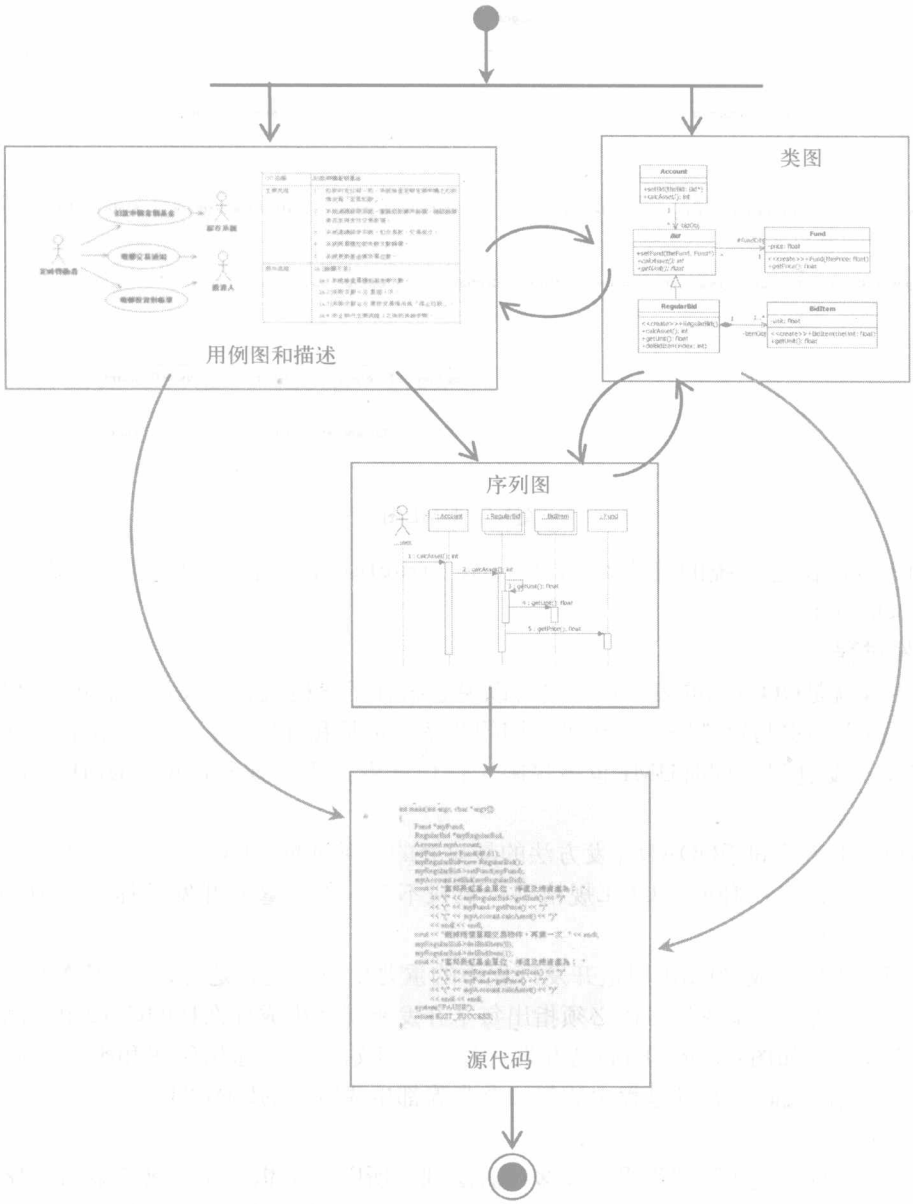


图1-8 简易的开发过程

我们来看一个用在分析阶段的模式——交易模式（transaction pattern）。在多数的企业中，交易（transaction）通常是一项很重要的企业概念。一旦交易发生，经常需要保存相关的交易数据。知名学者Peter Coad以此为核心概念，在<<Object Models: Strategies, Pattern, and Applications>>一书中，提出一个以交易为主的经验法则，记录了构成交易的人（participant）、事（transaction）、时（也记录在transaction中）、地（place）、物（item），称为交易模式，如图1-9所示。

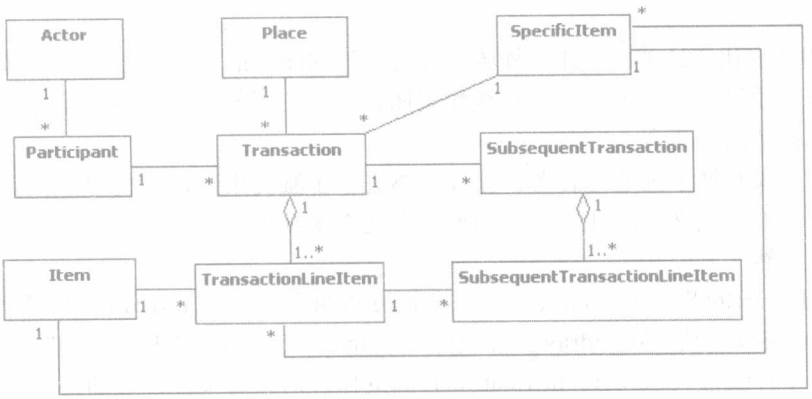


图1-9 交易模式

所以，遇到系统必须记录交易时，可以直接采用交易模式来构建类图。以基金系统为例，要记录定期定额申购基金的交易，便可以直接套用交易模式，立即获得如图1-10所示的类图。对应的细节如下：

- Participant-Transaction 投资人（investor）-定期申购（period purchase）
- Place-Transaction 银行（bank）-定期申购
- SpecificItem-Transaction 账户（account）-定期申购
- Transaction-TransactionLineItem 定期申购-单期交易（line item）
- Item-TransactionLineItem 基金（fund）-单期交易

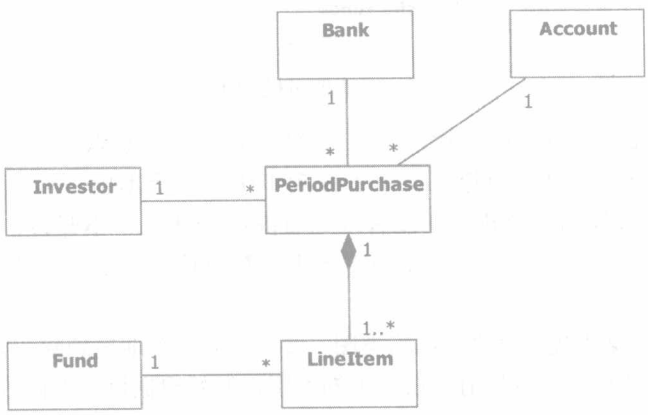


图1-10 套用交易模式

1.2 图形语言

在日常的自然语言之外，还有编程语言。倘若使用过数据流图（Data Flow Diagram, DFD）、结构图（structure chart）或者实体－关系图（Entity-Relationship Diagram, ERD），那其实已经认

识图形语言了。

UML采用图形化表达方式，许多领域中都有专门的图形语言。例如，建筑领域的建筑蓝图、木工领域的家具设计图、水电领域的管线配置图，以非工程的音乐领域的乐谱，就连日常生活中的交通标志也是图形语言。

图形语言不仅有助于沟通与思考，而且能够进一步激发出新颖的构想，我自己认为这是图形语言最大的好处。当然，除此之外，图形语言还有多项优点，如图形可以隐藏细节，凸显重点，令人一目了然。

以图1-11所示的用例图为例，一眼就能够得知顾客（Customer）使用自动柜员机（ATMSystem）获取提款（Withdraw）和转账（Transfer Funds）的服务，同时也得知这两项服务中都包含了卡片验证（Card Identification）的动作。对顾客来说，自动柜员机所提供的服务就是重点，通过这张用例图可以一目了然。

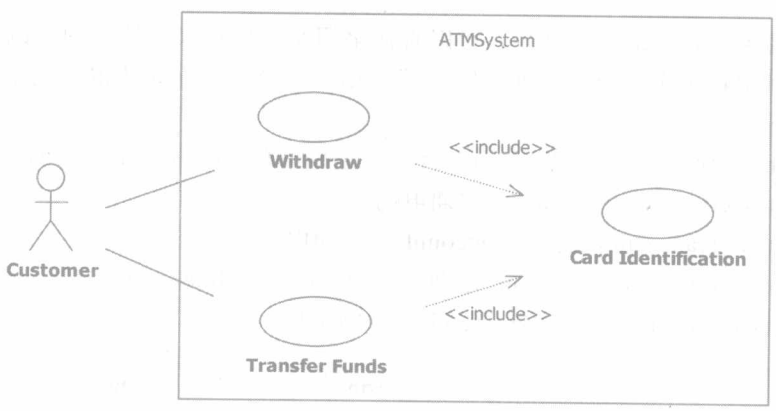


图1-11 自动柜员机的用例图

另外，图形还可以具体化脑海中的构思，方便保存、沟通及思考。我在辅导项目时，曾经有过这样的经验——5~6人开会讨论，从早上开到下午，还得不出一个结论，而且整个会议总是各自表述。后来，我提出一个建议：没有事先准备好UML图，不开会；没有讨论出一个有共识的UML图，不结束会议。这样，通过UML图来保存讨论的想法和结果、引导会议的进行及沟通、表述自己的构思和接纳别人的意见。

例如，会议开始，某人提出了图1-10的类图，大家七嘴八舌地提到除了定期定额申购基金外，单笔申购基金怎么处理呢？针对投射在白板上的类图，开始根据讨论修修改改，如图1-12所示。

除了增加了一个单笔申购（purchase）的类外，同时改变了其他类与单笔申购之间的关系，最后修改成图1-13后，会议告一段落。一张具体的图可以抛砖引玉，激发大家的想法，并且达成共识。

此外，图形还可以快速提升我们对陌生领域的认知，易于修正及扩展整个知识体系。这一点对经常接触各式不同领域的开发人员特别有帮助。每个人都有自己对知识独特的组织方式，无论讲述者或者写作者，想要迅速理解并吸收这些未曾涉及的陌生知识，最有效的方式就是去

掉枝节、利用图形凸显重点，以自己的思考方式重新组织这些知识，因此可以立即发现认知上的模糊或不解之处。

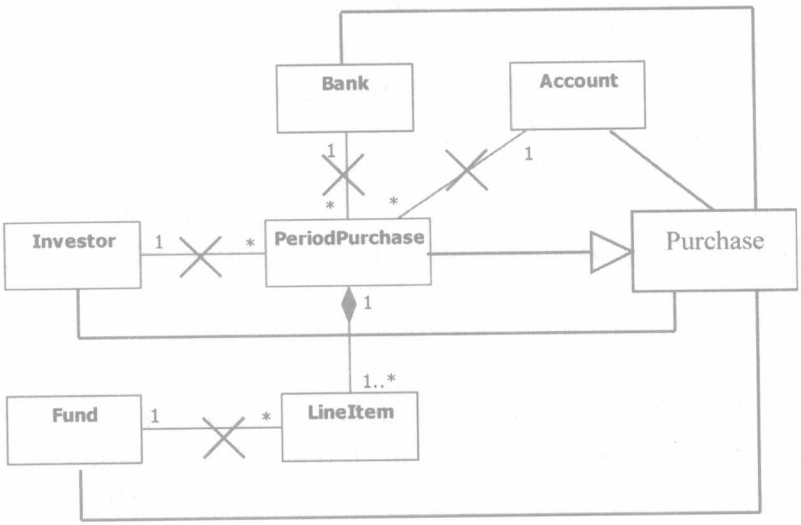


图1-12 增加一个类

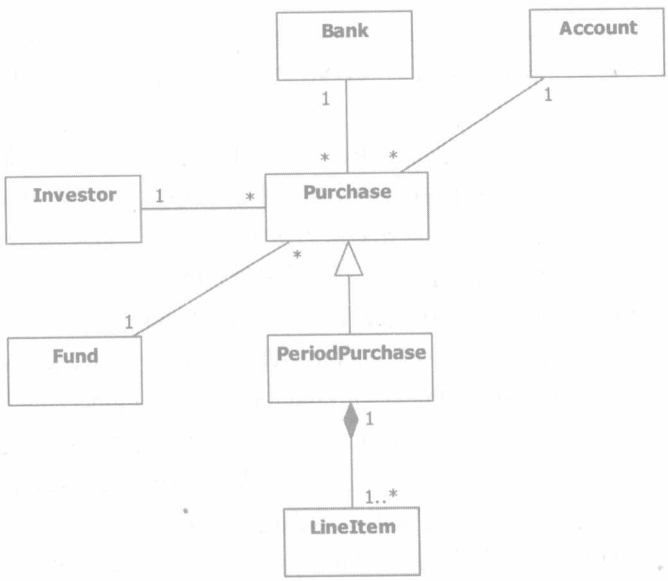


图1-13 新的类图

这也是我有过的经验。我曾经参与一个项目，开发一个功能丰富的产品。使用者和领域专家轮流上阵，让我听了整整一周的课，后来发现让我不至于头昏脑涨、昏昏欲睡的原因是，我在笔记本上画了一些UML图，一边记录整理，一边理解吸收。

图形还有一项很重要的特色是：可以凸显关系。散文式或条例式的文字说明可以将单一概念进行细腻的描述，当然也会详细描述概念与概念之间的关系。但是流于细节，难以察觉重要的关系，也难以将两个概念构成一个整体，激发出崭新的创意。

原创性的创新比较困难，也比较少见。更多的创新来自于整合现有的概念，看到别人所没有看到的新意。所以，单理解某一个概念是不够的，还需要通过图形凸显与其他概念之间的关系，从中创造出一个不同于其他的整合体。

总而言之，我确实享受到图形语言带来的下列好处。

- 图形不仅有助于沟通与思考，而且能够进一步激发出崭新的构想。
- 图形可以保存讨论的想法和结果，引导会议的进行及沟通，并且表述自己的构思和接纳别人的意见。
- 图形可以隐藏细节，凸显重点，令人一目了然。
- 图形可以具体化脑海中的构思，方便于保存、沟通及思考。
- 一张具体的图可以抛砖引玉，激发大家的想法，并且达成共识。
- 图形可以快速提升我们对陌生领域的认知，并且易于扩展整个知识体系。
- 图形可以凸显两个概念之间的关系，借此将不同的概念构成一个整体，激发出崭新的创意。

1.3 国际标准

我从<<UML 2 Certification Guide>>一书中摘录了一张UML的发展简图，并且做了一些修改，最后形成图1-14。从图1-14中，可以看到UML在1997年成为OMG标准，并于2000年成为ISO标准。

UML之父有三位：Grady Booch、James Rumbaugh和Ivar Jacobson，国外称他们为UML三友（Three Amgios）。最初，Booch和Rumbaugh两人提出OOAD方法——Booch Method和OMT（Object Modeling Technique），并且提出UM 0.8版。注意，那时还不是UML，而是UM（Unified Method）。两位OOAD大师试图提出一个统一的OOAD方法。

后来，Ivar Jacobson加入团队，UML三友正式一起工作，随后提出UML 0.9，并确定将开发过程剔除，提出统一的标准语言，而不是标准的OOAD方法。Ivar Jacobson是用例技术的创始人，他提出的OOAD方法叫OOSE（Object-Oriented Software Engineering）。

UML 0.9加入了IBM的OCL（Object Constraint Language）技术之后，提出UML 1.1版，并获得OMG的采纳。不同于UML主体的图示符号，OCL是由简单的文本数字符号组成，用来表达变量、约束或条件等比图示更小，不适合用图示表达的元素。

如今，UML已经是OMG最成熟且知名的标准了。您可以看到图1-15的OMG首页（http://www.omg.org），UML的标志高悬中央，也可以从OMG的首页链接到UML的首页（http://www.uml.org），如图1-16所示。

IBM一直对UML十分关注，在2002年12月花了21亿美元并购UML三友所在的公司——Rational。Rational公司当年的UML工具ROSE是最著名的UML工具。不过，IBM Rational会定期在Rational网站（http://www.rational.com）发表许多新技术，如图1-17所示。