计算机科学1998 Vol. 25 №. 4

样本的分类研究*)

A Study on Classification of Patterns

杨卫东 蔡希尧 陈 平

(西安电子科技大学软件所 西安710071)

摘 要 A classification method of patterns is presented, patterns are organized by their different abstract levels and different problem domain, and combined with process and concrete problems in O-O method. It supports the development of software systems using patterns in O-O methods.

关键词 Object-Oriented methods, OMT method, Patterns

1 引 言

样本描述了软件开发中出现的具体问题及其解决方案,面向对象(O-O)的软件开发方法定义了软件开发的一般建模概念、设计原则和开发过程。通过在O-O方法中应用样本,可以在开发方法的指导下使用样本,从而能够更有效地开发高质量的、可重用的O-O系统。

在O-O方法中应用样本,关键问题之一就是对样本进行分类,合适的分类方案有利于理解、选择和应用样本。目前,不同的样本系统具有不同的分类方案,如:[6,5,7]等。其中,有些分类方法较为含糊,没有清楚地定义问题类别,有些虽然按问题领域进行分类,却没有与O-O方法结合起来。

要使开发人员在O-O方法的指导下有效地选择样本,就必须使样本的分类方案与O-O方法的开发过程和所定义的分析、设计问题结合起来。本文提出的分类方案具有两个标准:第一,关于样本不同抽象层次的分类,设计问题具有不同的抽象层次,同样,样本也具有不同的抽象层次。从目前各类样本系统描述的样本中可以看出,样本已经涵盖了软件开发过程的各个阶段:分析、设计和实施。第二,关于问题领域的分类,根据问题进行分类更能清楚地区分样本,也有利于加入其它样本系统中的样本,从而形成统一的样本分类方案[7],为开发人员提供更为丰富的样本,同时,这里所强调的问题应该是O-O方法本身所关心的分析、设计问题。

目前,出现了关于软件开发各个方面的样本,我们主要研究与软件结构、行为有关的样本(不包括组织样本)。本文讨论了样本的分类问题,并给出了简单的结论。

2 样本的分类

方案的第一个分类标准根据样本的不同抽象层次进行分类,并与开发过程结合起来,将样本分为分析样本、体系结构样本、设计样本和实施样本。

·分析样本:表示了某一问题领域的共同结构和行为,它可能仅与某一个领域相关,也可能跨越多个领域。分析样本主要强调的是对问题领域的理解和建模,而不是与计算机相关的概念,如:分布系统、内存管理等。

·体系结构样本:决定系统的高层组织,它提供了一套预先定义好的子系统,说明了它们的责任,如:组织子系统之间关系的规则和指导原则。体系结构样本可以在软件开发的概要设计阶段中使用,如:OMT方法的系统设计阶段。

·设计样本:在规模上小于体系结构样本,但也独立于特定的程序设计语言。它提供了精炼子系统、构件或它们之间关系的模板,解决了特定背景下一般的设计问题。设计样本可以应用于软件开发的详细设计阶段,如:OMT方法的对象设计阶段。

·实施样本:是最底层的样本,它同时强调了设计和实施方面的问题,描述了怎样用某种特定程序设计语言实现构件或构件之间关系的某些方面。它可以在软件开发的实施阶段使用。

第二个分类标准是根据问题领域进行分类。要在方法中应用样本,根据问题的分类应该与方法所定义的分析、设计问题一致。由于不同的方法定义了不同的分析、设计问题,在使用不同方法时,同样的样本系统的分类会有所不同,我们以OMT方法为例来说明这种分类方案。

为了以方法所定义的分析、设计问题来组织样本的问题类别,需要从方法中抽取它所定义的分析、设计问题。OMT开发过程分为分析、系统设计、对象设计和实施,分别对应于分析样本、体系结构样本、设计样本和实施样本。由于OMT的各个阶段所描

*)本文工作得到国防科技预研基金项目"面向对象软件工程(OOSE)的研究"的支持。

述的重点不同[1]，应该对分析样本、体系结构样本、设计样本和实施样本分别进行分类。

OMT 的分析阶段主要是建立分析模型以准确地描述应用问题，它描述的是系统是什么，而不是怎样开发系统，与具体的计算机实施概念（如：子系统、算法、数据结构、存储管理等）无关，它是对应用问题的理解和建模。下面是从 OMT 方法分析过程中抽取的具体问题：

·选择对象、类：是建立分析模型的第一步，所选择的对象、类应该是应用领域内物理实体以及概念。避免与计算机实施相关的物理实体和概念（如：CPU、链表等）。

·标识类之间的联结：联结表明了两个或多个类之间的依赖性，标识类之间的联结包括标识作用名（Role Name）、联结的多重性（Multiplicity）等。

·标识对象和链的属性：属性是对象的特性，如：名字、重量、速度、颜色等。如果某个特性依赖于链的存在，则这个特性是链的属性。

·组织和简化对象类层次：组织和简化对象层次一般使用继承和聚合关系。继承关系可以从两个方向加入：从现存的类概括出超类（自底向上）或者将现存类进行精炼以得出子类（自顶向下）。

·确定并测试对象间查询的存取路径：通过对对象间存取路径的确定和测试，可以帮助找到所丢失的信息，如：属性、操作。

·标识事件：事件是发生在某一时刻来自对象外部的刺激。事件包括输入、判定、中断、转移等。

·标识对象之间的约束：约束是指对象之间的函数依赖关系。如：前置条件（Precondition）是输入值必须满足的约束，后置条件（Postcondition）是输出值必须成立的约束。

·在对象中加入操作：每一个对象类都包含一些基本的操作，如：属性值的读、取等，向对象发送的每一个事件都对应于对象的一个操作。

通过这样的分类方式，清楚地说明了 OMT 方法所定义的分析问题和分析样本之间的对应关系。例如我们将人作为对象，人有高度、重量、血糖含量，通常的建模方法如图1，但这种模型存在一些问题：如：人的高度为6，其语义较含糊，若将它限制在一种单位上，以后改变单位时较为困难，较为灵活的模型是[8]中的 Quantity 样本，如图2，它增加了一个称为量（Quantity）的对象，其中包括相应的运算和比较操作。Quantity 样本的使用更能清楚地表达应用语义，可以处理多种单位，使得模型更加灵活。更复杂的情况（如货币处理、面积换算等）可以使用 Conversion Ratio 样本、Compound Unit 样本等。[8]中的其它样本如：Organization Hierarchies、Organization Structure 是组织类层次结构的样本，Object Merge 样本用于认定对象。

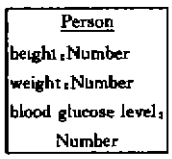

图1

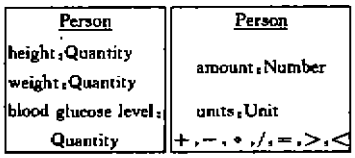

图2

OMT 的系统设计阶段决定了应用系统的高层组织结构，它将应用系统分为若干子系统，说明了各个子系统的责任以及它们之间的约束关系、合作方式等。体系结构样本是创造具有软件体系结构的模板，在系统设计阶段应用体系结构样本，可以在系统一级进行软件重用，有助于预测应用系统的一些非功能特性，如：可改变性、可扩充性、可重用性等。不同计算机领域的系统具有一定的共性，按计算机领域对体系结构样本分类比较合适。文[1]简要论述了建立几种系统的体系结构的原则：批处理、交互界面、动态模拟、实施系统等，根据计算机技术的发展，我们将体系结构样本分为以下几类：

·批处理系统：用于批处理系统典型的体系结构样本是 Pipeline，该样本将系统分解成独立的计算处理（过滤器）。数据流通过过滤器传递。重组过滤器可以建造一族相关的系统。

·仓库式系统：该类样本主要处理集的信息，并以多种方式操纵，通常需要持久性存储。如：Blackboard。

·分布式系统：这类样本描述了构件分布在不同的进程（Process）之上或在不同的子系统之中的体系结构，例如：OMG 的 CORBA 采用的中介（Broker）样本，客户-服务器样本，反应器（Reactor）样本。

·交互式系统：包括帮助建立需要进行人机交互的计算机系统的体系结构样本。建立这种系统体系结构的关键在于将特定应用的功能核心与其用户界面分离，使得用户界面发生变化时，不影响其功能核心或系统的数据模型，以增加系统的可重用性、可变化性。如：MVC、PAC[3]

·自适应系统：所包含的样本给系统提供了在功能需求发展、变化时，具有扩展和适应能力的基本结构。系统在一段时间以后，可能需要增加新的功能或对现存的服务进行修改，因此，该系统的体系结构必须支持新版本的操作系统、用户界面平台或第三方构件，并保证功能需求的发展和变化不影响其核心功能，否则，系统将难以维护。如：微内核样本（Microkernel）和反射样本（Reflection）[7]。微内核样本将系统的最小功能核心与扩展功能和用户定制部分分开，以适应将来系统需求的变化，现代操作系统（Mac、Window NT 等）就采用了这种结构。反射样本提供了动态改变系统结构和行为的机制，SOM 采用

了它的一些原则。

体系结构样本的使用并没有建立起完整的体系结构，只是软件系统的结构构架，需要进一步细化，如：任务的分配、构件及构件间关系的加细。另外，以上的几种类别并没有穷尽所有的样本，有时需要定义新的类别以增加新的样本。

设计样本应用于 OMT 的对象设计阶段，对象设计是对分析、设计模型的完善，加入了实施细节。根据对象设计阶段所强调的设计问题，我们将设计样本分类为：

·子系统接口设计：设计子系统接口旨在通过定义较为高层的接口，对子系统的客户屏蔽其内部结构，减少子系统间的依赖关系。文[1]并没有明确指出子系统接口的设计问题，但是在[2]和[3]中，Rumbaugh 定义了子系统对象，它提供了子系统的所有服务。Facade 样本中的 facade 对象扮演了子系统接口的角色，它提供了子系统所实施服务的高层接口，将从客户端接收的请求转送到子系统内部，并给客户端返回结果。

·对象方法设计：对象中的每一个操作都必须用合适的方法实现，OMT 的对象方法设计包括：考虑合适的数据结构，必要时定义新的内部类及操作，给相应的类分配操作的责任，需要对计算的复杂性、方法是否易于实施和理解、灵活性、可重用性进行权衡。例如，Strategy 样本将一族算法分别封装起来，提供了给一个类配置不同行为的方式。Master Slave 样本支持容错、并行计算，定义了 Master 对象和 Slave 对象，Master 对象将任务分为若干子任务，指派到几个独立的 Slave 对象，并返回最终结果。

·对象之间关系的设计：包括设计联结及对象之间的合作。典型的用于设计联结的样本是 Observer，Observer 定义了对象间的一对多的依赖关系，当一个对象状态发生变化时，所有依赖于它的对象被通知并自动更新。Mediator 样本改善了对象之间的联结及它们之间的合作，当对象之间的关系过于复杂时，可以引入 Mediator 样本，它通过 Mediator 对象来集中控制并协调对象之间的交互关系，减少了对象之间的依赖性，将多对多的联结变为一对多的联结，易于理解、实施、重用对象。

·对象接口设计：OMT 指出了对象接口设计的问题，主要从信息隐藏和优化的角度考虑。Adaptor 是用来设计对象接口的样本，主要解决的是从现有构件实施对象时其接口的兼容性问题。

·聚合关系设计：文[1]定义了三种聚合关系：固定聚合、可变聚合和递归聚合，没有指出如何设计聚合关系，[3]建议使用复合对象进行设计，它包含了一套对象实例和链接。Composite 和 Whole-Part 是用来设计聚合关系的样本，与 OMT 方法不同的是，客户可以将整体与其组成部分等同看待，更适合于设计递归的聚合关系。

·优化存取：据 OMT 方法的原则，在分析模型中不应该包含冗余类、冗余联结、导出属性等对应用领域冗余的信息，在对象设计阶段，应当增加这些信息以优化存取。Proxy[7]是用于优化存取的样本，它使构件的客户通过代理而不是直接存取构件的服务，方便了存取并防止无权限的存取。

·调整类继承关系：在对象设计阶段需要调整类和操作以提高共享的程度，包括重新安排类和操作、从一组类中抽象出共同的行为，当继承关系的语义无效时使用指派来共享行为。例如，要对某一个对象而不是整个类增加额外的操作时，若用继承关系，需要加入子类，增加了系统的复杂性，Decorator 样本引入了 Decorate 对象并使用指派解决了这样的问题，提供了较为灵活、高效的方式。

这样的分类建立了设计样本与 OMT 方法对象设计阶段的设计问题的对应关系。有时，有的样本同时考虑了几个设计问题，有的样本结构非常相似，但用来解决不同的设计问题，例如，Proxy[7]和 Decorator 是结构相似的样本，但 Proxy 样本用于优化存取，而 Decorator 用于调整类继承关系。

OMT 实施阶段可以采用实施样本，它是将对象设计阶段的模型转换为特定的程序设计语言、数据库或硬件实施。实施样本和 OMT 的实施阶段都与特定的程序设计语言密切相关，因此，实施样本在不同的语言中具有不同的分类与风格，文[10]描述了有关 C++语言的样本及其分类，文[9]描述了有关 SMALLTALK 语言的样本及其分类。由于不同的程序设计语言的特征不同，常常解决同样问题的实施样本在不同的语言中风格完全不同，有时甚至在一种语言中有用而在另一种语言中却没有意义，例如，在 C++中使用 Reference Counting 样本来管理动态资源分配，而 SMALLTALK 语言则提供了自动废物回收机制，该样本在 SMALLTALK 语言中没有意义。在实施阶段应用实施样本可以加速软件开发和维护，易于程序设计人员之间的交流，有助于提高代码质量。

结论 本文提出的分类方法由两个分类标准组成，简单、易于理解和使用，主要以[7]、[4]、[8]为例进行讨论，也适合其它一些样本，通过与 O-O 方法中的具体问题及开发过程结合，使开发人员能够在 O-O 方法的指导下有效地选择样本。

参考文献

[1] J. Rumbaugh et al., Object Oriented Modeling and Design, Prentice Hall, 1991

[2] James Rumbaugh. Get Started, Using Use Cases to Capture Requirements, JOOP, Sept. 1994

计算机科学1998 Vol. 25 №. 4

引入元表示的可视化复合重用构件模型研究*)

On Composite Visual Reusable Part Model with Meta Representation Introduced

刘西洋　桑大勇　蔡希尧　陈　平

(西安电子科技大学软件工程研究所　西安710071)

摘　要　Considering the limitations of the meta representation in CORBA, OLE/COM and Java Beans, a composite visual resuable part model with meta representation is proposed based on the development of a reusable chinese reports part under IBM VisualAge for Smalltalk. The essential technologies related to its implementation and its implementation strategies in various object-oriented programming languages and systems are also disscussed.

关键词　Meta representation, Composite visual reusable part model, Multi-coordination system collaboration, Multi-Design pattern collaboration

1　引言

可视化程序设计以面向对象范型为基础，以可重用软件构件为基本构造单元，正越来越多地被用于关键(Misson critical)应用软件系统的设计与实现，而不是仅仅局限于人机界面设计。研究实践表明：上述关键应用软件系统的功能和非功能进化，迫切要求软件技术支持运行时的递增式有序软件更新[1]。目前工业界广泛基于的一阶对象技术(以C++为代表)显然难以满足上述要求。与此同时，以对象反射(Reflection)、元数据以及元对象协议为代表的经典动态对象技术已经提供了现实的对象系统自适应机制，"动态对象技术被专门设计成构造频繁演化的软件系统，使得软件系统演化的代价正比于其中必要的变化部分的规模，而不是正比于整个软件系统的规模"[1]，因而关键应用软件系统的开发迫切需要引入经典动态对象技术。在可视化程序设计的前提条件下，上述动态对象技术的引入将直接落实到面向可视化程序设计的可重用构件模型——可视化重用构件模型之上，因为可视化重用构件模型是可视化程序设计得以实施的基础和前提。还应看到，经典动态对象技术的精髓在于对象系统的高阶抽象，即将对象系统划分为关于目标应用领域的对象层系以及关于对象系统自身的元对象层系，对象层系中反映不同应用语义的不同对象，在元对象层系中被统一地加以表示和操纵，其中关于对象以及对象系统自身结构和行为的运行时表示即元表示，是元对象层系得以建立的基础，因而在可视化重用构件模型中引入经典动态对象技术的一个首要前提是：引入关于可重用构件自身的元表示。

进一步分析可以看出，可视化重用构件模型本身就需要引入元表示，否则可视化程序设计环境将无法一致，也无法有效地获取其所操纵的可重用构件的接口定义，从而难以在可重用构件之间建立可视化的语义关联，可视化程序设计因此将难以实施。事实上，中立于体系结构、操作系统、程序设计语言以及软件开发环境的统一元表示，是任何工业化的

*)本文得到九五军事预研课题"软件重用及可重用部件库"(15.1.4.2)的支持。刘西洋　在职博士生。桑大勇　博士生。蔡希尧　教授、博士导师。陈　平　教授、博士导师，主要研究方向为面向对象的软件工程。

[3] James Rumbaugh, Object-Oriented Modeling Technology, Tutorial, OOPSLA'94

[4] Erich Gamma et al., Design Patterns: Elements of Reusable Object-Oriented Software, 1994

[5] W. Zimmer, Relationships Between Design Patterns, Proc. of PLoP'94

[6] R. Eisenhauer et al., Architecture-Handbook for Software Architecture, Internal report, 1994

[7] Frank Buschmann, et al., Pattern-Oriented Software Architecture, 1996

[8] Analysis Patterns: Reused Object Models, Addison Wesley Longman, Inc., 1996

[9] K. Beck, Smalltalk Best Practice Patterns, Volume 1: Coding, Prentice Hall

[10] J. O. Coplien, Advanced C++-Programming Styles and Idioms, Addson-Wesley, Reading, MA, 1992