一种责任驱动的面向对象分析与设计方法*

田 忠 钱乐秋 赵文耘 钱家骅

(复旦大学计算机科学系,上海 200433)

摘要 本文结合数据流图编辑工具开发过程的描述,阐述了一个责任驱动的面向对象开发方法的基本思想、实施框架和记号系统.这一方法强调软件是对现实的模拟,软件开发应从问题理解入手,以对象在问题中所承担的责任来驱动软件的开发.这一过程包括反复循环的5个步骤:划分主题、标识对象类、建立联系、定义服务和定义属性.通过行为分析,获得问题的主题划分,从而引导标识与问题相关的对象类及其相互联系,并指导定义其属性和服务,由此获得未来系统的一个清晰的分层模型.

关键词 面向对象软件开发,责任驱动,主题,合作.

随着计算机应用的不断深入,软件需求量日益增大,软件复杂度和规模也越来越大,软件开发的效率及质量越来越难以保证,迫切需要采用更为先进的软件开发技术来指导软件开发.面向对象的软件开发为软件人员克服这些困难提供了有效的手段.

面向对象(OO)思想主要起源于面向对象程序设计和信息模型化两个方面,目前尚未形成主流的方法学,较著名的研究如OOA/OOD[1,2]、OMT[3]等.OOA/OOD概念简单,但其方法学缺少可操作性;OMT较为完整地体现了OO的思想,但概念复杂,文档规模庞大,且3个模型间的关系难以控制.我们在七五、八五及863项目开发积累的经验基础上,从OO基本原理出发,总结出一种责任驱动的面向对象软件开发方法.

1 面向对象的软件开发

1.1 面向对象软件开发的基本思想

OO开发的基本思想可以用如下等式来概括"面向对象=对象+分类+继承+基于消息传递的通讯"[1].OO范式中,系统的基本构件是对象,对象是封装了特定信息和功能的自主个体:它保存自己的状态,并提供一组用于维护该状态的服务,对象之间通过消息传递相互联系以完成系统的任务.对象类是对一组性质相同的对象的抽象.抽象和封装是OO软件

* 本文1994-05-12收到,1994-10-07定稿

本研究得到了国家八五攻关项目、863计划项目的支持.作者田忠,1968年生,博士研究生,主要研究领域为面向对象软件开发,基于知识的软件工程,CASE工具.钱乐秋,1942年生,教授,主要研究领域为软件工程,CASE工具,软件生产自动化.赵文耘,1964年生,讲师,主要研究领域为计算机辅助软件工程,面向对象开发技术,管理信息系统.钱家骅,享年64岁,生前曾支持多项"七五"、"八五"和"863"高技术项目,获得多项部、市委级科技进步奖.

本文通讯联系人:田忠,上海 200433,复旦大学计算机科学系

开发的基本原则:抽象原则即是忽略与当前目标无关的东西,以便充分地考察与当前目标有关的方面;封装原则即是将一目标的具体细节隐藏在完成该目标的对象内部,以使信息、服务局部化,使得系统将来的变化局部化.OO 方法要求开发者致力于理解问题,再由此导出问题的解,以便开发出较为稳定、易于修改的应用系统.

总之,OO 方法建立在对象及其属性/服务、类及其实例、整体与部份、一般与特殊等日常概念基础上,从理解问题入手,力图完整真实地反映用户需求,建立应用系统的用户模型,并最终演化成实现模型.

1.2　OO 开发实施框架

OO 范式下,软件开发所使用的术语来自应用领域,所使用的记号相当简单,这些都有助于软件开发者同用户的交流.用户反馈信息是系统模型不断演化的源泉,并促使最终的实现模型尽可能地贴近用户的要求.

与文献[1]类似,我们分 5 层来开发和描述问题:主题层、对象类层、联系层、服务层、属性层.不同的是,我们强调以主题、对象类在问题中所承担的责任为依据来构造和提炼这 5 层模型:通过对系统整体行为的分析获得对问题的主题划分;通过对象行为分析识别、定义对象类及其相互关系.这种责任驱动的开发方式通过主题、类、责任等的分级抽象,能有效地控制问题规模,摆脱数据驱动和功能驱动开发方式的片面性[4],更好地遵循封装、抽象原则.实施 OO 开发的基本步骤如图 1,本节将描述这种方法的基本思想,下节将结合实例描述具体实施的步骤和原则.

(需求分析是从领域的角度对用户陈述的提炼;设计是从机器实现角度对模型的进一步提炼.实际的开发过程必是一个反复循环的过程.)

图1　OO开发方法的实施步骤示意图

应用系统的分析与设计大致包括需求分析、系统设计、概要设计和详细设计 4 个阶段.图 1 不涉及非功能性需求和系统设计的工作.目前对于非功能性需求只能作一般的定性描述,如项目进度、行业公约、环境限定、性能/安全等几方面的要求.系统设计是对系统整体实现策略进行规划,主要考虑的是分布、并发以及数据的存储/冗余控制等要求及其对模型的影响;概要设计是在系统设计确立的实现原则指导下,将用户模型转化到更贴近实现的系统模型上;详细设计把系统模型映射到选定的实现语言和环境上.这一过程的主要问题是如何更好地遵守抽象与封装原则,如何控制问题的规模.

1.2.1　主题(Subject)划分与复杂度控制

OO 模型有助于理解问题,但问题本身的复杂性并不会自动消失.我们仍需要一些辅助手段来控制我们所解决问题的规模.目前流行的 OO 开发方法,一般只提供到对象类一级的抽象手段,即使提供了其它机制,其定义也存在较大的问题.

文献[1]定义主题为“控制读者在同一时间所考虑模型规模的机制”,并将主题与结构对应起来,主题仅作为读者复审和概括问题空间的手段,其划分在确定了应用的对象类空间

和结构后进行.因此,主题有时庞大(复杂结构),有时琐碎(单个类),反映了这种 OOA/OOD 由底向上数据驱动的特征,对控制问题的复杂度作用有限.

OMT[3]引入了 module 和 sheet:前者反映问题的某一侧面;后者将 module 分割成若干块,以便每块用一页纸绘制.module 比文献[1]的主题概括能力有所增强,但与文献[1]类似,module 和 sheet 都只是模型制作后期的包装手段.

我们认为主题应成为模型开发者控制模型复杂度和项目进展的工具.主题是模型中一组紧密相关的对象类的合成体,反映了系统某一相对独立方面的活动.主题间可建立消息连接[1];不同主题的对象类间可建立聚合、合作关系,但不允许建立继承关系(否则基类的内部细节会通过继承关系向其它主题公开,破坏封装原则).合作和聚合能帮助澄清消息连接的具体含义;聚合可以通过责任委派(delegation)来模拟继承.这样的主题划分较为平衡,反映出应用的逻辑关系,是控制问题复杂度的有力手段.

1.2.2 责任驱动的软件开发过程

通常有 3 种驱动软件开发的方式:功能驱动、数据驱动和责任驱动.功能驱动以功能分解为核心,但忽视了数据结构的复杂性对系统的影响;数据驱动以数据结构设计为核心,但往往易将数据结构反映到系统各模块的界面上[5],背离 OO 的封装与抽象原则.责任驱动基于客户—服务器模型,每一对象都是不可缺少的自主个体,为系统提供某些不可替代的动作.软件开发通常有两种典型的过程:自顶向下逐步求精和由底向上逐步合成.目前典型的 OO 方法大多采用由底向上数据驱动的过程.

我们认为 OO 开发应是一种由未知到已知的过程:对于熟悉的领域,我们往往掌握了相当数量的可选对象类和较稳定的主题划分;对于较生疏的领域,则采取分而治之的办法,努力通过划分主题掌握问题的概貌,再各个击破(确定其中的对象类及相互联系).而此过程是责任驱动的:责任不仅为标识对象类提供了线索,而且为定义对象类提供了依据.责任还为划分主题提供了依据:每一主题应承担问题中一组紧密相关或相似的责任.因此,整个开发过程都应围绕责任分析进行.

1.3 记号系统(notation)

图 2 给出了我们所使用的图形记号.与文献[1]不同,我们利用主题将问题划分成逻辑片段,只有关系密切的主题才画在同一起,而不用一张平面图来表示整个模型.我们在模型表示中只记录对象类的名称,而其细节则记录在为每一对象类准备的字典卡上,因为将所有对象类的属性和服务同时记录在一张图上为模型的修改、阅读带来了不必要的困难,也使我们难以对属性、服务进一步说明.开发过程的每次反复,对象类内部的变化一般能局限于个别卡片,而模型整体相对稳定.

对象类是 OO 模型的基本成分,模型中一部份对象类(具体类)与问题空间的实体直接对应,而另一部分(抽象类)则是出于对问题的分析或出于对设计、实现、重用的考虑而对其它类的抽象.OO 模型还包括 4 种联系:继承关系刻划了一般与特殊的关系(导出类具有基类的全部特征);聚合关系刻划了整体与部分的关系(包括两种聚合:被聚集成份只有一种出现在聚合体中的相斥(exclusive)聚合、被聚集成份必须全部出现在聚合体中的相加(inclusive)聚合);合作关系表示对象类实例间的信息/服务交换;消息连接用于总结主题间的信息交换关系.此外,还需说明聚合、合作中各成分的参与性(重数).

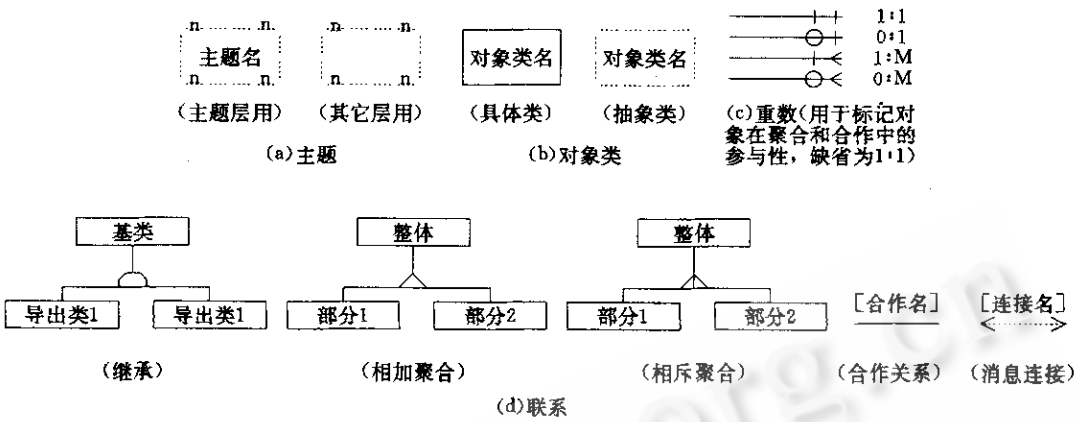


图2 OO模型记号体系

2 SAT的需求分析

SAT*包括分层数据流图(DFD)与小说明字典的编辑、一致性维护、需求文档生成以及联机求助等几个方面的工作.其中DFD编辑器用以编辑DFD图示,并确保DFD各成份语义完整、一致.本文以SAT的DFD编辑器为例进行讨论,并简称为SAT.

2.1 划分主题

主题层用以刻划系统的轮廓,将一复杂问题分割成若干易于控制的子问题.主题划分要求充分理解应用领域,不必考虑机器如何实现,应用领域的现实分工往往为主题划分提供了蓝本.通常由经验丰富的分析员根据其工作经验和常识,从用户的需求陈述中快速地识别出几个初始的主题.

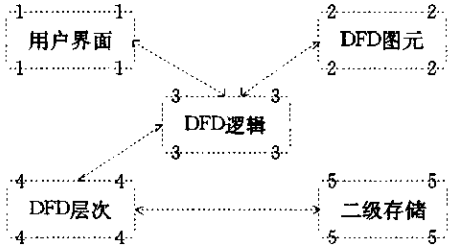
根据对SAT的问题描述以及关于结构化分析方法、CASE工具的知识,我们将SAT承担的责任分成5类(图3):与用户的交互(用户界面)、DFD元素的图示(DFD图元)、DFD实际含义(DFD逻辑)、DFD文档(DFD层次)、模型存储(二级存储).我们利用主题将DFD的含义与其图示、单个元素同整体模型区分开来;交互设备、外存等的变化局限于各自主题,不会对全局造成大的影响.这样,每一主题都有相对独立的明确任务,将引导我们寻找与问题有关的对象类及其相互关系.我们不必担心初次划分是否完全正确,OO模型的抽象性和封装性、责任驱动的循环步骤,使我们在开发过程中能以较小的代价尽早发现和纠正错误.重要的是一开始就努力通过划分主题来掌握问题的全貌.

2.2 标识对象类

寻找合适对象是OO开发中最艰巨的工作,目前多数OO方法都建议通过问题描述中的名词来标识对象,这显然是不够的,我们通过主题/责任为标识对象提供线索.主题为系统活动作了明确的划分,其定义为通过责任分析标识该主题的对象类提供了明确的指示.对象类的责任确定了它是否与该主题有关,而相应的合作者(即其它的信息/服务提供者)预示着新的类.DFD逻辑的责任是负责各DFD实体的创建和销毁,它们分别由各实体承担,因此

---

* 结构化分析工具SAT(Structured Analysis Tool)是"七五"、"八五"攻关项目"集成化软件工程支撑环境"青鸟Ⅰ型和青鸟Ⅱ型中支持软件需求分析阶段工作的软件工具.



(主题间的消息连接反映了主题间的信息需求关系，是对不同主题的对象类间合作、聚合关系的总结)

图3 SAT的主题层

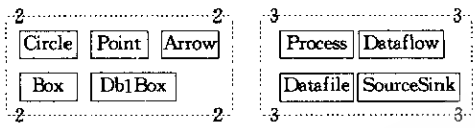


图4 SAT的对象类层(片断)

就有了图 4 中主题 3 的 4 个类. 而为完成这些责任，各 DFD 实体对应的图元又需提供各自图形的显示、删除服务，暗示出图 4 主题 2 的 5 个类. 寻找对象类的工作在完成了全部预想责任，不再有新的合作者出现时结束. 重要的是一切紧紧围绕主题的责任，找出承担这些责任的最小对象类空间. 在此过程中还有一些辅助工具(如合作—责任卡[5]，事件跟踪图[3])能帮助分析和分派责任. 合作显示出类之间、主题之间的耦合度：当主题耦合度过高时就要重新考虑主题的划分.

2.3 标识联系

目前常见的 OO 方法过多地强调继承. 我们认为继承只是抽象的一种形式，应强调以合作为基础标识对象间的关系，以合作→聚合→继承的顺序标识 3 种联系(不同主题的类之间的聚合/合作关系说明了这些主题间消息连接的确切含义). 提供信息交换的合作者为标识合作联系提供了依据；这种关系如果反映了组装、包含、容纳这样的整体/部分概念，则可提炼为聚合关系；如果多个类具有相同/相似的责任或联系则可考虑用继承来简化 OO 模型. 当主题间联系过于密切时也需要调整主题划分以降低耦合度.

图 5 给出了 SAT 的联系层一部分. Circle 上的动作总是由对 Process 的编辑动作传播而来，它俩的合作表现为聚合关系；类似的关系也存在于其它 DFD 实体及其图元之间. 对于 Process/SourceSink/Datafile 的动作往往也影响到相伴的 Dataflow，故前三者均与 Dataflow 存在伴随合作(AssocWith)，由此我们又需对前三者进行抽象，得到图 5 中的抽象类 ChargibleDFDEntity. 考虑到 DFD 实体共同的创建/销毁责任，我们在主题 3 中加入 DFDEntity作为对全体 DFD 实体的抽象.

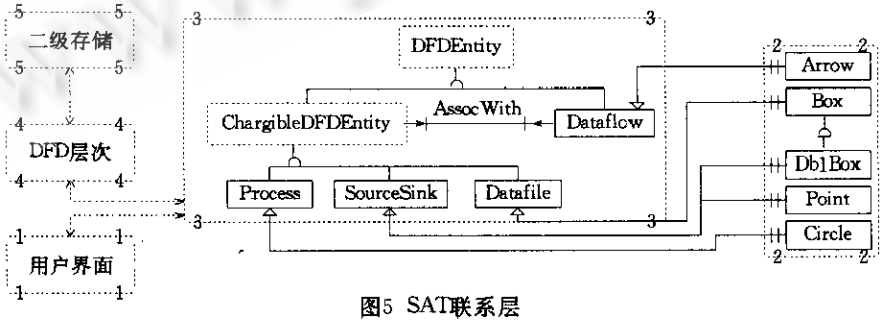


图5 SAT联系层

2.4 定义服务、定义属性

在建立了问题用户模型的基本框架之后，我们要进一步分析每一对象所承担的责任，落

实它要提供的服务,并据此确定它所需保存的信息,从而建立该对象类如图 6 的字典卡.服务和属性的定义往往交替进行,必要时还可使用一些传统方法(如 DFD,STD 等)对服务做进一步的分析和说明.圆(Circle)为完成责任"显示加工外形",要提供画圆/画标号/画名称(drawCircle/drawID/ drawLebel)三项服务,而为完成画圆(drawCircle)它必须存储其圆心(center)和半径(radius).为所有类建立字典卡后,我们就获得了 SAT 的初始用户模型.

对象类	Process
主题	DFD逻辑实体
服务	1. insert
	2. delete
	3. moveTo
	4. rename
	5. decompose
	6. isChosen
属性	1. name
	2. ID
	3. level

对象类	Circle
主题	DFD图元
服务	1. drawcircle
	2. undraw
	3. distanceOf
	4. drawLabel
	5. drawID
属性	1. center
	2. radius

图6 SAT对象类字典卡(部分)

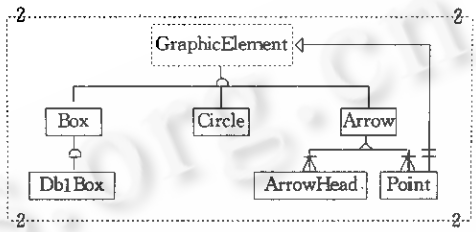


图7 SAT概要设计的联系层(DFD图元主题)

在此基础上经过评审、修改、确认,就可提交 SAT 的用户模型,进入设计阶段.

3 SAT 的设计

OO 范式中,分析与设计使用相同的术语、记号,不同的是人们的视图:分析是在问题领域中研究系统的用户模型;设计则是研究如何用机器来实现这一模型.概要设计的主要任务是在系统设计所确立的实现策略指导下将用户模型演化成系统模型.

概要设计的实施步骤如图 1②,其输入是用户模型,输出是系统模型,这一过程的步骤、原则与分析时相似,只是看问题的出发点由问题本身转向机器.SAT 的概要设计中,主题层不变;对象类层需要引入新的类(图 7):有向弧(Arrow)用有向折线段实现,而有向折线段是用一个箭头(ArrowHead)和一个拐点(Point)序列表示;属性层中,出于作图和距离计算的算法效率的考虑,要求每一 DFD 图元都定义"中心点"和"覆盖半径".这引起 SAT 联系层发生变化,工作流程要返回到"标识联系".注意这里的往返不是将前面的工作推倒重来,而是对前面工作的增量式修改以使模型更接近实现.

在此后的详细设计中,要根据所选择的实现环境、运行环境、实现语言以及内外存的要求对在概要设计中得到的系统模型进一步修改.经历图 1③的反复过程后,系统模型就演化为实现模型.由于 SAT 的实现、运行环境为 UNIX/Motif,实现语言是 C++2.1,要求对 SAT 系统模型进行一系列修改:增加属性来模拟聚合;引入基于模板(template)的类属链表来模拟 1:M 关系;增加一组图元类来封装 Motif 的伪对象(widget/gadget)等等.详细设计结束时我们得到能与 C++语言直接对应的 SAT 实现模型.

4 结　论

在采用本文所述 OO 方法开发 SAT 的过程中,我们发现:主题层相对稳定;对象类层的变化主要是对原有类的抽象、细化;对象类内部的变化一般局限于其字典卡.与目前典型的

OO 方法相比，这一方法强调理解问题，以责任分析为基础，围绕对象责任的识别和指派，通过一个责任驱动的 5 步反复过程，将用户需求陈述提炼成用户模型，并将此模型逐步演化成系统模型和实现模型. 这一方法力图在开发过程中为开发者控制复杂度、寻找合适的对象类及其相互联系、定义必要的服务和属性提供帮助. 通过采用 OO 方法，我们缩短了 SAT 的开发时间，提高了 SAT 的健壮性和可维护性. 采用该方法，我们在 SAT 基础上开发"八五"攻关项目"结构化设计工具"时对象类重用超过 50%. 这一方法是建立在我们 OO 开发实践和综合多种现有 OO 技术的基础上，适用于一般系统的开发，具有较大的实用意义.

参考文献

1 Coad P, Yourdon E. Object oriented analysis. Yourdon Press, Prentice Hall, 1990.

2 Coad P, Yourdon E. Object oriented design. Yourdon Press, Prentice Hall, 1991.

3 Rumbaugh J, Blaha M, Oremerlani W *et al*. Object—oriented modelling and design. Prentice Hall, 1991.

4 Beck K, Cunningham W. A laboratory for teaching object—oriented thinking. ACM OOPSLA'89 Proceedings, 1989. 1—7.

5 Wirfs—Brock R, Wilkerson B. Object—oriented design: a responsibility—driven approach. ACM OOPSLA'89 Proceedings, 1989. 71—75.

A RESPONSIBILITY—DRIVEN OBJECT—ORIENTED ANALYSIS AND DESIGN METHOD

Tian Zhong Qian Leqiu Zhao Wenyun Qian Jiahua

(*Department of Computer Science, Fudan University, Shanghai* 200433)

Abstract This paper presents the basic idea, notation and development process of an object—oriented analysis and design method through the development of a data flow diagram editor. This method takes a responsibility—driven modelling approach to software development, emphasizing that software simulates the real world. It starts with problem—understanding to build a user model of the problem, then this model is evolved into an implementation model through an iterating five—stage—process during each phase of the software development process. The five stages include: dividing subjects, identifying classes, establishing relations, defining services and defining attributes. Based on behavior analysis, the user problem is first divided into a set of subjects, each of which hosts a number of classes with distinguished responsibilities and collaborators. The responsibilities then further help define the services and attributes of each class and their inter—relations. All these put together give software developers a clear layered model of the proposed system.

Key words Object—oriented software development, responsibility—driven approach, subject, collaboration.