第6卷第2期 软件学报 Vol. 6, No. 2
1995年2月 JOURNAL OF SOFTWARE Feb. 1995

基于细化的对象分析和设计方法*

柳诚飞 居德华

(华东理工大学计算机科学系,上海 200237)

摘要 面向对象的开发方法是一种很有前途的软件开发方法.它通过对应用系统的问题空间对象直接建模,然后将它们同态映射到解空间,使应用易理解、易维护.然而,对于大而复杂的应用系统,这种方法较难识别问题空间中的对象.本文在比较了功能化和面向对象的开发这两种方法后,提出了一种基于细化的对象分析和设计方法ROAD.

关键词 面向对象的开发,功能开发,细化,客户/服务员模型.

近年来,面向对象的开发(OOD)方法[1]在软件界引起了人们极大的关注.OOD方法是基于对象的概念对应用系统进行分析、设计和编程的软件开发方法,它不同于传统的功能化开发方法(FD).OOD方法根据对象的外部视域,即对象是如何出现的,而不是对象的内部视域,即对象究竟是什么,来识别问题空间中的对象,并用相应的软件对象来自然地模拟它们的行为,以便将问题空间同态映射到解空间.对象的行为(外部视域)是通过对象所遭受的动作和它需要其它对象应具有的动作来反映,对象的内部视域涉及到对象的具体设计.

Booch较早为Ada程序的开发提出了一个OOD方法[2].它根据应用问题形成一个正文形式的非形式化策略,并指出正文中的名词、动词和形容词,然后由名词导出对象,由形容词导出对象的属性,由动词导出对象的操作,最后建立对象的界面(操作)并实现这些操作.这种方法很难适应复杂的大型系统的分析,其对象的关系是一种平面结构.

Seidewitz在GOOD[3]中引入了子系统抽象来分解一个大系统,即将一个复合对象分解成较小的组成对象,因而系统的构成是分层次的,称之为复合层次.在最上层,为一个单一的系统对象.从系统层开始,每个对象细化为低层对象图中的组成对象.在最低层,为过程或数据存贮这些本原对象.GOOD最大的弊病是对象概念紊乱,对象有时退化为功能.

Bailin[4]在GOOD的基础上提出了分层的实体数据流图(EDFD)的OO分析方法,在EDFD中,包括了实体和功能,每个功能都依附于实体.

Wirf-Brook和Wilkerson[5]提出了职责驱动(responsibility-driven)的OOD方法,使用客户/服务员(client/server)模型强调对象之间的合同(contract)关系,即客户需要哪些服

* 本文1992-05-30收到,1993-01-09定稿

本文得到国家自然科学基金资助.作者**柳诚飞**,1961年生,副教授,主要研究领域为数据库,软件工程,分布式系统,面向对象的方法.**居德华**,1938年生,教授,主要研究领域为软件自动化技术,软件工具与环境,软件工程技术,办公室自动化.

本文通讯联系人:柳诚飞,上海 200237,华东理工大学计算机系

务，服务员提供哪些服务，对象的设计予以延迟.

Shlaer 和 Mellor[6]提出了一种组合型方法，它分别采用 ERD、DFD 和 STD 表示对象的结构、操作和生成周期. 该方法适合于实时系统的开发.

对于 OOD，面向对象的分析尤其重要，即如何去识别出对象及其对象之间的联系来反映应用的问题空间. 然而，至今这方面的工作尚显不足. 本文提出的方法力求解决这个问题.

1　OOD 和 FD 的比较

不少学者对 OOD 方法寄于厚望，希望它能取代传统的 FD 方法. 理由是 OOD 方法以对象构造现实世界应用的抽象模型的思考过程比 FD 方法以功能的思考过程自然. OOD 方法对问题空间的建模比 FD 方法更直接地映射到解空间，即 OOD 具有自然性和直观性. 通常 OOD 的这种映射是同态的，而 FD 方法从分析模型到设计模型的转换是异态的. 然而不幸的是，这种提法迄今只适合于小而简单的系统，尚未有大而复杂的实际例子证实这一点.

目前，面向对象的开发方法学还处于非形式化研究阶段，没有开发大型系统的经验. OOD 方法和 FD 方法的争论很热烈，它们的本质区别在于是以对象分析为主还是以功能分析为主. OOD 是否比 FD 分析方法自然、直接？它能否完全取代 FD？不少学者对此持怀疑态度，我们也有同感. 同 Loy[7]一样，我们认为 OOD 方法和 FD 方法应有机地结合起来，各取所长.

对象分析是一种构造方法，本质上是一种自底向上的组合. 程序设计语言多年来发展的经验告知，构造方法较难，这正是函数式程序设计语言等构造性语言推广的难处，其原因在于这种方法缺乏一条引导设计者思考问题的主线，从而使人感到无从入手. 对于 OOD 方法来说，当问题空间复杂时，很难寻找适合的对象. 相反，功能分析法则拥有这条主线，根据这主线对问题自顶向下进行分解，使得问题变成较易解决的若干子问题.

OOD 方法确实构造了问题空间到解空间的一种同态映射，这是 FD 方法所不具有的. 由于这种同态映射，也使得用 OOD 方法设计出的应用系统具有两个重要的特性：

易理解性：它自然地反映了应用.

易维护性：封装、信息隐蔽、类化、继承性等特性使应用易修改和复用.

上面的分析给我们一个启示，采用 OOD 方法设计出的应用系统具有很好的性质. 然而如何去识别错综复杂的问题空间中的对象及其对象之间的联系，可借助 FD 方法来引导. 因而本文的思想是，以功能细化和对象细化结合起来分析问题空间，然后将问题空间同态映射到解空间，以获取系统的易理解性和易维护性. OOD 和 FD 两者是有机地结合在一起，相互补充的，OOD 是目标，FD 是一种途径.

FD 和 OOD 的结合不是简单的结合，若不对 FD 方法加以改良和限制，会使 OOD 引入歧途. Yourdan[8]认为，SA 很难“有效地”产生 OO 设计和实现的需求，原因是 SA 模型转换到 OO 设计模型之间的跨度或障碍太大. 这一点值得借鉴.

作者认为，运用 FD 方法来识别问题空间的对象会产生一些“质量”问题. 原因是 FD 方法将数据和功能分离开. 例如，DFD，活动是输入到输出的变换，同样的输入总是产生同样的输出，而没有考虑到对象内在的持续性，它会产生副作用，由于对象内部状态的变化，同样的输入可能会产生不同的输出. 因而单纯使用现有的 FD 方法来识别问题空间的对象，很难保

证对象的质量,所以FD方法的使用应遵守OOA的聚集原理:每个操作都附属于某个对象,它不能与数据分离开. Demeter法则[9]也是这个道理.

2 ROAD方法

根据上述讨论,本文提出了一种基于细化的对象分析和设计的软件开发方法ROAD. ROAD吸取了现有OOD方法的一些经验.强调FD方法的使用应纳入OOD的轨道.其好处是既能借助FD方法对应用分析,又不使OOD引入歧途.通过对象和操作的细化过程来识别应用系统中的对象和联系,从而完成应用系统的分析.基于[1]的分类,ROAD不同于纯OOD和组合型方法,属于适应型(adaptive)方法.

下面介绍ROAD的设计思想,步骤和实例.

2.1 设计思想和原则

分析和设计采用OOD和FD的结合,其中OOD是目标,FD是一种途径.FD方法的使用以OOA的聚集原理为前提,即功能的实现要归结到对象的操作上.

- 采用分层细化:包括对象的细化和对象操作的细化,并且细化可以分层进行.
- 信息隐蔽:这包括分层抽象(高层隐蔽低层的细节)和对象封装(对象的实现部分不可见).
- 增量式细化:对象的分析随着细化过程而完善,不断地加入新的特性(包括属性和操作).
- 尽可能地复用系统已定义的对象,通过自底向上的方式使用这些积木块来构造应用系统.
- 对象的详细设计尽可能延迟.
- 在对象(类)的设计中,尽可能多地包含面向对象的方法的特点,使应用系统易于组织,理解,复用和维护.

2.2 具体步骤

Step 1. 初始化

在系统这个层次上,根据客户/服务员模型识别一些关键的问题空间对象.这里系统的用户可看作为客户对象.如果在这个层次上一开始难以识别任何对象,可将系统本身看成唯一的对象,它提供的服务可根据用户与它的客户/服务员关系来确定.

Step 2. 对象细化

在当前这个层次上,描述每个已识别的对象的行为特性和生存周期.其目的是细化对象的结构和功能,并识别出本层和下层的新对象.新对象可分成两类:下层的组成(嵌套)对象和本层的引用对象.另外,若识别出的对象是对象库中的现存对象,它们可以被复用.

这里推荐两个分析模型:JSD和STD,它们分别可用以功能型对象和实时型对象.

Step 3. 对象操作的细化

描述本层已识别出的对象的操作.

对于本原操作,其操作体可用规格说明语言实现,也可以直接用某种程序设计语言来实现.例如C++.

对于颗粒度比较大的操作,可以对它进行细化,以便:

• 根据嵌套和引用关系识别下层和本层的新对象;

• 将该操作的子功能用新对象所提供的操作(服务)来实现. 若本层有新识别出的对象,转 Step 2.

Step 4. 确定下一层对象之间的关系

采用客户/服务员模型确定下一层对象之间的通讯关系,这种关系也可能发生在上层对象与本层对象上. 可以使用对象图(object diagram)刻划这种关系.

Step 5. 重复 Step 2~Step 4 直到无新对象产生为止

此时,这一层次上的对象可能是现存的复用对象;或者,这些对象的操作是本原的.

Step 6. 类化和类格(class lattice)的构造

将具有相同结构特性和行为特性的对象抽象成类,并描述其结构特性和行为特性. 类中的每个实例(即对象)共享类的特性. 并将一些相似类的共性泛化成超类,构造出类格,在这个类格中,下层结点(子类)继承上层结点(超类)的属性和操作,并允许有它特有的属性和操作.

2.3　实例

为了说明如何使用 ROAD 方法来具体开发一个实际的应用系统,请看一个服务台系统 SDS(Service Desk System)的实例. SDS 系统是为某个计算机服务公司而设计的,该公司承接新老用户对计算机的安装、修理业务,并为老用户做计算机的定期维护.

Step 1. 在 SDS 系统层,若一开始未能识别出明确的对象,SDS 系统被看作单一的对象. 依据用户和 SDS 对象的需求关系,SDS 要提供下列操作:接受请求,处理请求,定期维护,报表结帐.

Step 2.1. 由于在系统层,只有 SDS 一个对象且属功能型的,可采用 JSD 方法描述 SDS 对象的结构和生存周期,如图 1 所示.

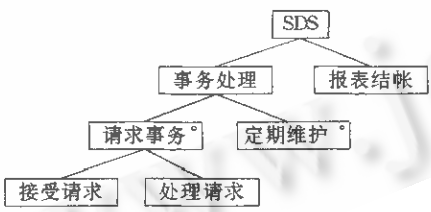

图1 SDS对象的结构和生存周期

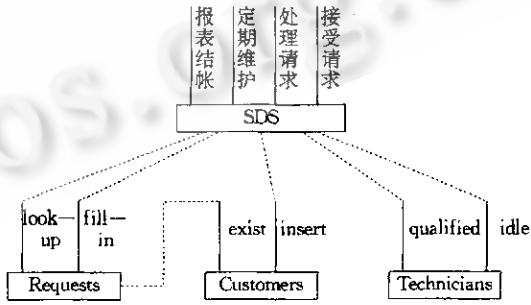

图2 SDS系统部分对象的客户/服务员关系

Step 3.1. 接着对 SDS 的每个操作进行细化. 在这里,我们看两个操作的细化.

对于接受请求操作,无非让用户填表登记,因而可将申请(Requests)看作新对象. 接受请求可调用对象 Requests 提供的填表操作 fill—in 即可.

对于处理请求操作,它的功能是重复查看 Requests,可调用 Requests 的新操作 look—up 完成,然后根据安装/维修信息决定是做安装处理还是维修处理. 下面看一下维修处理,新用户的处理与老用户不同,首先要备案成老用户,这又可通过在新识别的对象 Customers

中加入插入新操作 insert，然后调用之. 接着，将请求交给一个合适的技术员，这又涉及新的对象 Technicians，寻找“合适的”技术员可通过在 Technicians 中加上“空闲”和“胜任”判别新操作 idle 和 qualified，并调用它们来实现. 待技术员维修完成后，要对费用估计，产生维修表，这个过程又会识别出一些新的对象. 这里不再深入了.

Step 4. 1. 确定新对象 Requests，Customers，Technicians 以及 SDS 之间的联系. 这待 Step 3. 2 完成后一起给出.

Step 2. 2. 分析第二层对象 Requests，Customers，Technicians 的结构和生存周期.

Step 3. 2. 细化第二层对象 Requests，Customers，Technicians 的操作. 例如，Requests 的 fill－in 操作的实现涉及到填写用户信息(例如，姓名，地址，新/老用户等)；服务信息(例如，安装/维修，是否上门服务等等). 若要在 fill－in 操作中检查一下是否该用户是老用户，这又引出了 Customers 对象上检查老用户是否存在的新操作 exist. 因此，对象图上应反映 Requests 和 Customers 的客户/服务员关系，如图 2 所示.

Step 4. 2/Step 5. 假设第二层未产生新对象，分析结束.

Step 6. 可将对象 Requests，Customers 和 Technicians 分别构造成对象类 Request－Form，Customer－Form 和 Technician－Form. 可将 Customer－Form 和 Technician－Form 的共性抽象成超类 Person. 为实现方便，也可将 Request－Form，Customer－Form 和 Technician－Form 作为 Form 的子类，共享表格类的操作.

3 结束语

本文分析了一些典型的 OOD 方法，比较了 OOD 方法和 FD 方法，提出了一种基于细化的对象分析和设计方法 ROAD. ROAD 吸取了 OOD 和 FD 方法的长处，以 OOD 为目标，以 FD 为途径，系统的分析和设计是识别对象，确定对象之间的客户/服务员关系，细化对象和对象操作的分层细化过程，是增量式的不断完善的过程，这与人们分析问题的思路是一致的. 目前，一个基于 ROAD 方法的集成软件工程环境 ISEE/OOD 正在开发之中，ISEE/OOD提供了五个 SPEC 来描述一个应用系统，它们分别是对象规格说明 OSPEC，对象生存周期规格说明 LSPEC，对象操作规格说明 FSPEC，类格规格说明 HSPEC 和类规格说明 CSPEC. 这些 SPEC 对应五个图形编辑器，在协调程序的控制下构成 ISEE/OOD 的用户界面. 程序生成器分析这些 SPEC 信息并生成应用的代码. 环境中所有的信息由一个面向对象的 DBMS 来支持.

参考文献

1 Monarchi D, Puhr G. A research typology for object－oriented analysis and design. CACM, 1992, **33(9)**: 35－47.

2 Booch G. Object－oriented development. IEEE Trans. Software Eng., 1986, **12(2)**: 211－221.

3 Seidewitz E, Stark M. General object－oriented software development. Software Eng. Laboratory Series, SEL－86－002, NASA, Goddard Space Flight Center, Aug. 1986.

4 Bailin S C. An object－oriented requirements specification method. CACM, 1989, **32(5)**: 608－623.

5 Wirf－Brook R, Wilkerson B, Wiener L. Designing object－oriented software. N. J.: Prentice Hall, Englewood Cliffs, 1990.

6 Shlaer S, Mellor S. Object lifecycles: modelling the world in states. N. J.: Prentice Hall, Englewood Cliffs, 1992.

7 Loy P H. A comparison of object—oriented and structured development methods. ACM SIGSOFT Software Engineering Notes, 1990, 15(1): 44—48.
8 PANEL: structured analysis and object oriented analysis. In: ECOOP/OOPSLA'90 Proceedings, 1990. 135—138.
9 Lieberherr K, Riel A. Demeter: a CASE study of software growth through parameterized classes. J. of Object—Oriented Program, 1988, 1(3): 8—22.

REFINEMENT—BASED OBJECT ANALYSIS AND DESIGN

Liu Chengfei　Ju Dehua

(*Department of Computer Science, East—China University of Science and Technology, Shanghai* 200237)

Abstract　Object—oriented development is a promising method for software development. It models problem space objects directly, then isomorphically maps the problem space onto the corresponding solution space. However, it is difficult to find problem space objects for a pure OOD method. In this paper, after comparing object—oriented development with functional development, a method called ROAD, refinement—based object analysis and design, is proposed.

Key words　Object—oriented development, functional development, refinement, client/server model.