结合形式化的面向对象设计方法与支撑系统*

郑明春 段会川 高 波 张家重

(山东师范大学计算机科学系 济南 250014)
(南京大学计算机软件新技术国家重点实验室 南京 210093)
E-mail: compass@jn-public.sd.cninfo.net

摘要 为了有效地结合形式化和非形式化设计方法各自的优点,克服其不足之处,以尽可能保证软件设计的质量与可靠性,文章提出了一种将形式化方法与非形式化的面向对象设计方法HOOD(hierarchical object-oriented design)相结合的途径,并介绍了其机器支撑环境的设计与实现.该途径在对层次式面向对象设计方法HOOD进行必要扩充的基础上,有机地集成了Z语言等形式规约技术.支持这一途径的支撑环境提供了一套方便灵活的图形构筑工具、语法制导的形式语言与文本编辑工具,以及自动检查机制等.

关键词 面向对象,形式化方法,HOOD(hierarchical object-oriented design),软件设计,支撑系统.
中图法分类号 TP311

众所熟知,现有的面向对象分析或设计方法大都用图形化语言从宏观上刻画软件系统的模型结构,而在某些局部方面则使用自然语言或结构化英语,如OMT(object modeling technique),HOOD(hierarchical object-oriented design)[1,2]方法等,其主要特点是比较直观、易于理解,且便于表达与使用.但是,采用图形化语言所构筑的模型(或软件规约)中往往存在着潜在性错误,且难以发现,而这些错误将直接影响后续工程及软件的安全性与可靠性.究其原因,乃是图形化软件规约所具备的非形式特征,即存在模糊性或歧义性问题,正因如此,才会妨碍软件开发人员就待开发软件的各个细节达成共识,从而也就难以发现不完整性与不一致性,乃至错误的问题.

为解决上述问题,近年来,人们开始探索形式化方法与面向对象方法的结合问题,其目的在于有效地结合形式化和非形式化方法各自的优点,并力图克服其不足之处,以求获得较为严格、精确的软件规约,并尽可能地保证软件规约的质量与可靠性.如,H. C. Cheng等人开发的图形化软件开发环境VisualSpecs[3],它结合了OMT和形式规约语言Larch;R. B. Jackson等人提出的一种形式化的面向对象规约模型OSS(object-oriented system specification)[4],它在OSA(object-oriented system analysis)方法的基础上加以扩充,其规约语言基于一阶谓词逻辑和时态逻辑;还有Mike Moulding等人关于CORE(controled requirements expression)和VDM(vienna development method)结合的研究[5];R. D. Givoanni等人提出的集成HOOD和Z的方法等.一般地,形式化技术与面向对象方法的集成方式有两个层次[6]:(1) 借助形式规约语言来提高图形化语言的描述能力,而开发过程则主要还是采用非形式化方法;(2) 从图形化模型到形式规约语言的转换,而在后继的开发阶段便可以采用形式化方法.其中,第1个层次较适宜于"精确的"、但不是完全形式化的开发模式,如,Fusion方法和Syntropy方法[7,8]等;第2个层次则较适宜于需要极度集成的系统开发,如ROOA(rigorous object-oriented analysis)方法[9]等,而且,通过转换,原本对象模型中某些"不良定义"的概念可以得到精确的解释,有利于实现整个软件开发过

---

* 作者郑明春,1963年生,女,副教授,主要研究领域为软件自动化技术.段会川,1966年生,副教授,主要研究领域为计算机软件开发环境.高波,1970年生,讲师,主要研究领域为计算机软件开发环境.张家重,1965年生,博士,教授,主要研究领域为软件自动化技术.

本文通讯联系人:郑明春,济南 250014,山东师范大学计算机科学系

本文1997-12-05收到原稿,1998-05-12收到修改稿

程的自动化.

从目前来看,有关形式化方法与对象技术的结合的研究多集中在需求级[10],而在设计与实现级则较少,而且现有工作尚有许多问题,如,有的工作缺乏良好的支撑环境,使形式化技术的使用受到一定限制,有的未涉及从"非形式"或"半形式"到"形式"的转换问题,难以直接适应上述两个层次的研究需要.为此,我们从软件结构设计入手,通过分析现有的相关工作,选择了层次化面向对象设计方法HOOD,结合形式规约技术对其模型与规约语言加以扩充,并研制了支持这一途径,即形式化与面向对象相结合的软件设计支撑环境OOFE(object-oriented formal environment),它由软件设计支撑子系统和设计规约转换子系统构成.前者支持用户借助扩充的HOOD模型和方法所进行的面向对象软件设计,对所构模型进行一致性与完整性检查,形成软件设计规约;后者可将设计规约自动转换成用Z语言[11]书写的形式规约.由此,本项目可以达到上述集成方式的两个层次,其中设计支撑子系统实现第1层,设计规约转换子系统实现第2层.本文仅讨论形式化方法与面向对象设计方法HOOD的结合途径,并介绍其机器支撑系统的设计与实现.

1 设计方法

1.1 HOOD方法及其改进

HOOD是欧洲航空局开发的层次式面向对象设计方法,在欧洲被广泛应用到大型复杂系统的设计中.它主要面向Ada语言,具有描述并行实时系统的能力.为了使HOOD具有更广泛的适应性,我们对HOOD进行了必要的扩充,使之具有更多的面向对象特征,提高了其复用能力.

HOOD设计过程可以用一棵设计过程树来刻画,其树根是欲构模的系统,设计过程与对象的分解相关,其高层对象对应于高级抽象,低层对象对应于低级抽象,树叶是无需继续分解的终极对象,树的分支由包含关系衍生得到.由此可以看出,HOOD较好地结合了结构化设计方法和面向对象方法,在很大程度上解决了许多面向对象方法对系统功能刻画不足的缺点.HOOD将对象分为两类,即被动对象和主动对象.主动对象可以接受外部的刺激,并根据其内部状态作出应.同时,HOOD还区分了3类对象之间的关系,即包含(include)关系,使用(use)关系和实现(implemented by)关系.

为了使所提出的对象技术和形式化方法相结合的途径具有广泛的适应性,所选择的对象模型应该具有更多的OO特征.为此,我们在HOOD方法中增加了继承机制,允许系统中的某个对象继承其他对象.为了保证继承的正确性,同时又不破坏HOOD的层次结构,使用继承时必须遵循以下规则:

- 继承关系不能形成环,并且它与使用、实现关系也不能形成环.
- 只允许单继承,而不允许多继承.
- 不允许连续继承.

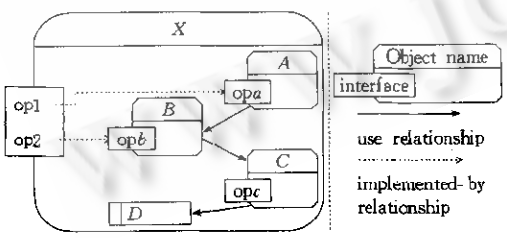

图1 HOOD对象模型

此外,根据HOOD之对象概念,每一个非终极对象是由其子对象构成的复合对象,也就是说,当对象A继承了对象B时,实际上是A继承了以B为根的子树.因此,我们规定,继承对象不可以继续分解,因为这种分解可以在继承之前进行.对于继承的数据类型、操作等,我们定义了重命名和重定义设施.HOOD对象模型如图1所示.其中,父对象X包含子对象A,B,C,对象A使用B,B使用C,X提供的操作op1由对象A的操作op$a$实现,操作op2由对象B的操作op$b$实现,对象C继承了外部对象D.

1.2 设计过程

软件设计分两个阶段进行:第1阶段,从已有的需求规约中识别对象,建立对象模型;第2阶段,对建立起来的对象模型进行精化,包括对对象接口、状态、操作等进行详细定义,形成完整的设计规约.整个软件设计过程遵循逐步精化的原则,对象的识别和描述可以分布在不同的抽象级上,模型的精化也可以逐步进行.从总体上来

看,在设计前期,用户主要使用非形式和半形式化方法进行对象构模,随着设计过程的深入,用户所使用的形式化成分逐渐增多,规约的形式化程度逐渐加强.因此,规约语言应集成非形式化(自然语言)、半形式化(图形和结构化英语)及形式化(形式规约语言)3 种成分.在设计过程中,主要用图形语言描述应用系统的宏观结构,用结构化英语和形式规约语言描述系统的微观设计,这样既有利于设计人员理解,又保持了规约的严格性,从而在一定程度上保证了其正确性.这个过程如图 2 所示.

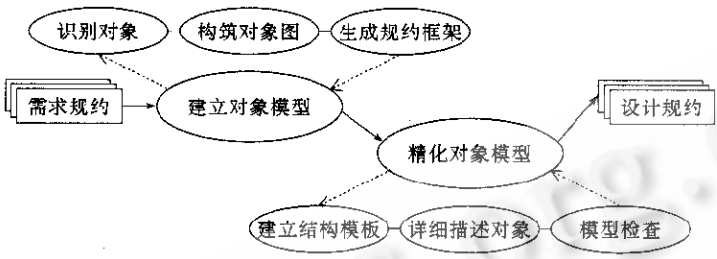

图2　设计过程

其中,建立对象模型阶段包括如下 3 个步骤:

(1) 识别对象.

(2) 构筑对象图.

(3) 生成规约框架.由 HOOD 图自动生成文本形式的规约结构,即对象描述框架(object description skeleton,简称 ODS),它用框架的方式反映对象的各部分及对象之间的联系.

精化对象模型阶段也包括 3 个步骤:

(1) 建立结构模板.在对象描述框架的基础上,建立对象结构模板,它是对对象详细定义的概要指导.

(2) 详细描述对象.用结构化英语或形式规约语言 Z 详细描述对象.

(3) 模型检查.对模型进行一致性和完整性检查.

上述两个阶段及其各个步骤,并非必须顺序执行,它们之间可以循环、交叉.

2　软件设计规约语言

HOOD 所提供的规约语言 ODL(object description language)支持用户在面向对象框架之下进行软件设计.由于 HOOD 基于 Ada 语言,因此,ODL 在很大程度上依赖于 Ada,在对操作进行详细描述时则直接使用 Ada 伪码,为使 ODL 具有更广泛的适应性,我们对 ODL 也进行了必要改进,主要工作包括:(1) 使 ODL 支持改进的对象模型的描述;(2) 非形式、半形式和形式 3 种方式并用;(3) 对 ODL 修改的部分进行严格的定义,包括给出详细的 BNF 形式定义和一致性、完整性检查规则;(4) 修改了设计规约的结构框架,使之更适合对象结构的描述.

2.1　对象描述框架

采用改进后的 ODL 所描述的对象框架 ODS 具有下面 4 个部分:对象名、接口、内部描述和对象行为描述.其形式概略地描述如下:

〈对象〉::=OBJECT〈对象名〉〈接口〉〈内部描述〉〈行为描述〉
　　ENDOBJECT
〈对象名〉::=〈标识符〉':'〈对象类型〉[〈继承对象〉]
〈接口〉::=PROVIDED〈提供的接口〉
　　REQUIRED〈需要的接口〉
〈内部描述〉::=INTERNAL
　　[〈类型定义〉]
　　[〈数据描述〉][CONSTRAINT〈约束条件〉]
　　[〈包含子对象〉][〈内部操作型构〉]

〈行为描述〉::=[〈对象控制结构〉]〈对象操作结构〉

〈对象控制结构〉::=OBCS [DESCRIPTION 〈语义解释〉]〈ObcsCode〉

〈对象操作结构〉::=OPCS〈操作名〉[〈受约束类型〉]

INPUT〈输入变量表〉 OUTPUT〈输出变量表〉

[DESCRIPTION〈语义解释〉]

[IMPLEMENTED-BY〈实现关系〉]

PRE〈前断言〉 POST〈后断言〉

其中,〈内部描述〉定义对象的属性,对于非终极对象,它仅包括反映包含关系的子对象列表和内部操作的型构,对于终极对象,它还包括数据类型的定义、数据的声明和约束等;〈OBCS〉只为"主动对象"所设,它刻画操作之间的同步关系;〈OPCS〉描述了所有内部的和所提供的操作.显然,不同类型的对象,其ODS有不同的部分.

2.2 语言成分

ODL语言有机地结合了非形式、半形式和形式3种表示方式,相应的语言成分分述如下:

(1) 非形式化成分.在ODS中以DESCRIPTION引导的用于对规约进行语义描述的部分,这些成分主要是为了便于用户理解.

(2) 半形式化成分.它包括图形和结构化英语两种形式.图形方式用于对软件结构进行概要描述,基本图元有两种对象图元、3种对象关系图元和被精化对象框图元;结构化英语一方面用作图形模型的等价描述,另一方面用于对软件的细节描述.

(3) 形式化成分.用于对软件进行详细定义.为了便于向Z语言的转换,ODL的形式化成分采用与Z语言相同的一阶谓词表达形式描述约束条件和前后置断言或直接用Z模式(schema)描述对象的行为.

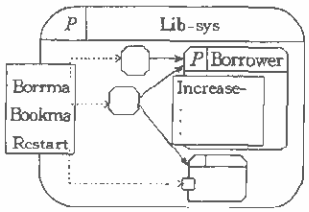

图3　图书馆对象

对于主动对象并行的描述,将其分为两个层次,一个是操作级,一个是对象级.操作级描述受约束的操作;对象级则主要刻画并行,我们借鉴了VDM^{++}描述并行的方法,用事件的历史函数来反映状态,用许可谓词描述对并行的约束,并在ObcsCode中将它们一并写成Z语言的形式.

例如,图3是一个简单的图书馆系统的根对象的图形描述.它提供了3个操作,包含4个子对象,其中有两个控制对象.在HOOD中,控制对象不能被进一步分解,且不能被其他对象使用,也没有提供的接口.对象Borrower的ODS形式如下:

```
OBJECT Borrower: PASSIVE
  DESCRIPTIONS...
  PROVIDED
    OPERATIONS  Increase-amount,Decrease-amount...
  DATATYPE BORROWER
  INTERNAL
    TYPE
      [BORROWER]
      RIGHT=(true,false)
    DATA
      readers:P BORROWER
      borrowers: BORROWER ⇸ ◁ limit:Z; amount:Z; right:RIGHT▷
      CONSTRAINT
        INIT borrowers=∅
        dom borrowers=readers
    OPCS  Increase-amount
```

```
    INPUT borr?:BORROWER; amnt?:Z
    DESCRIPTIONS...
      PRE   borr?∈readers
            ∃ bf∈borrowers•bf(borr?).amount+amnt?≤bf(borr?).limit
      POST  borrowers'=borrower+{borr?↦ (limit⇒bf(borr?).limit;
            amount⇒bf(borr?).amount+amnt?; right⇒true)}
  OPCS  Decrease-amount
    ⋮
ENDOBJECT
```

3 支撑系统

3.1 系统概述

设计支撑系统的功能部件包括一个图形编辑浏览器,一个语法制导的规约语言支撑设施以及规约文档管理部件、文档生成程序、用户界面管理和系统监控部件等,其结构如图4所示.系统所支持的主线操作是对象模型的构作,它利用超文本技术将非形式化文本、图形化成分和形式规约语言 Z 这3种不同成分结合起来,用 OLE (object linking embedding)技术实现图形编辑浏览程序和文本编辑程序的连接.系统在 P5,Windows 3.x 环境下,用 Visual BASIC 实现.它以窗口作为基本的界面,支持多窗口显示、菜单选择、超文本连结等功能,用户界面相当友善.

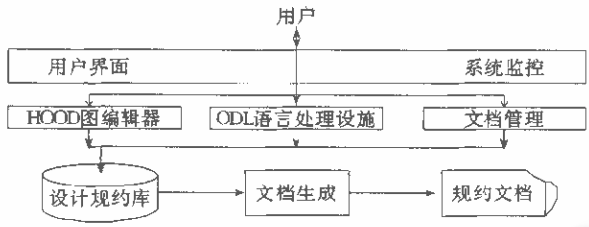

图4　系统结构

3.2 主要功能

(1) 图形编辑.系统提供基本图元和若干编辑功能按钮,图形的编辑以被精化的某个对象为基本单元.通过点中对象图元的关键区域可以完成对象分解(zoom in)与返回(zoom out).

(2) 图形浏览.图形浏览有两种方式:一种是通过窗口的切换、平铺和重叠,另一种是通过对象概貌浏览图.

(3) 模型检查.

(4) 生成规约框架.根据 HOOD 图由系统自动生成规约框架,它是建立规约基本形式制导模板的基础.

(5) 文本编辑.

(6) 显示与打印.

3.3 数据组织

以用户角度看,从 HOOD 图的构筑到规约文档的详细定义,是一个逐步精化、充实完善的过程,其用户界面是一个统一的整体,通过激活图元或文本框,可以连接到其他成分的描述或浏览.支撑系统利用超文本技术实现了这一过程.根据超文本数据的组织模型,其数据组织有3个层次,即用户层、逻辑层和物理存储层.在系统中,用户层是以 HOOD 图和语法制导的文本编辑窗口为主的用户界面,逻辑层呈双层网络结构,物理层采用关系数据库形式存储.VB(visual BASIC)的对象定义和事件驱动方式为超文本的实现提供了许多方便.

4 结束语

本文提出了一种支持形式化方法与对象技术结合的软件设计方法,介绍了其支撑系统的设计与实现.与此

相关的工作有 R. D. Giovanni 等人提出的一种集成 HOOD 和 Z 的方法，这种方法主要给 HOOD 对象增加了形式语义信息，未涉及向形式设计规约的转换，且无机器系统的支持. 本文所述工作吸取了 R. D. Giovanni 等人所提出方法的许多好思想，并借鉴了 VDM^{++}描述主动对象的一些方法，提出了将 Z 语言嵌入到设计规约中以增强其非形式化文本语言的描述能力，而且，这种具有部分形式化形式的设计规约可以自动地转换到形式规约. 这项研究是对 OO 技术与形式化方法结合的一个实践，其结果也可以用于其他 OOD(object-oriented design)方法. 这项工作的特点在于，该方法既保持了图形化构模易于使用和理解的优点，又增加了其设计规约的严格性，并在非形式化和形式化之间建立了沟通的桥梁. 从第2.2节给出的例子可以看出，根对象 Lib-sys 代表图书馆系统，对象模型形象地反映了系统层次分解的情况和功能，用户只需在语法制导的框架中填入部分形式化描述，系统就可以自动转换到相应的 Z 语言形式. 但是，这种途径可能会给设计者带来的困难是，部分形式化成分的嵌入依然要求设计者具有一定的数学基础. 今后，我们将着手对 HOOD 设计规约进行细化以形成软件的详细设计规约，并进一步生成可执行代码，以期对面向对象软件自动化工作有一个更深入的探索.

参考文献

1 Rumbaugh J *et al*. Object-oriented Modeling and Design. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1991

2 HOOD Working Group. European space agency. HOOD Reference Manual, WME/89-173/JB, Issue 3.0, 1989

3 Bourdeau R H, Cheng H C. A formal semantics for object model diagrams. IEEE Transactions on Software Engineering, 1995,21(10):799～821

4 Jackson R B *et al*. Developing formal object-oriented requirements specifications: a model, tool and technique. Information Systems, 1995,20(4):273～289

5 Moulding M, Smith L. Combining formal specification and CORE: an experimental investigation. Software Engineering Journal, 1995,10(2):31～42

6 Goldsack S J, Kent S J H *et al*. Formal Methods and Object Technology. Berlin: Springer-Verlag, 1996

7 Coleman D *et al*. Object-oriented Development, the FUSION Method. Englewood Cliffs, NJ: Prentice Hall Object-oriented Series, 1994

8 Cook C, Daniels J. Designing Object Systems: Object-oriented Modelling with Syntropy. Englewood Cliffs, NJ: Prentice Hall, Inc., 1994

9 Morera A M D, Clarck R G. Rigorous object-oriented analysis. Technical Report, CSM-109, University of Stiring, Scotland, 1993

10 张家重.对象式需求分析及其自动化技术的研究[博士学位论文].南京大学,1997
(Zhang Jia-zhong. A study of object-oriented requirements analysis and its automation [Ph. D. Thesis]. Nanjing University, 1997)

11 Spivey J M. The Z Notation: a Reference Manual, 2nd Edition. Englewood Cliffs, NJ: Prentice Hall, Inc., 1989

A Design Method and Its System Supporting the Combination of Formal Methods and Object Technologies

ZHENG Ming-chun DUAN Hui-chuan GAO Bo ZHANG Jia-zhong

(*Department of Computer Science Shandong Normal University Ji'nan* 250014)

(*State Key Laboratory for Novel Software Technology Nanjing University Nanjing* 210093)

Abstract In order to pick up the complementary benefits of formal and informal design methods, and overcome their shortcomings to ensure the quality and reliability of software design, an approach to the combination of formal methods and informal object-oriented design methods—— HOOD is proposed in this paper. The design and implementation of a mechanical supported environment for the approach is also introduced. Based on the necessary improvement on HOOD, a hierarchical object-oriented design method, the approach integrates the Z notation and other formal specification techniques. The supported environment provides the users with the convenient and flexible tools for constructing graphics, syntax-directed editors of formal notations and text, and facilities of automatic verification.

Key words Object-orientation, formal method, HOOD (hierarchical object-oriented design), software design, support system.