第7卷第3期
1996年3月

软 件 学 报
JOURNAL OF SOFTWARE

Vol. 7, No. 3
Mar. 1996

面向对象的软件构造*

冯玉琳　黄 涛　李 京

（中国科学院软件研究所　北京　100080）　（中国科学技术大学计算机系　合肥　230026）

摘要　面向对象技术的发展改变了传统软件系统的结构和设计方法学.本文作者提出一种称为SCOP的面向对象的软件系统结构模式,即软件是对象模块的复合,而软件设计则是对象模块经过程控制进行复合的构造生成.本文从语义模型、描述语言、设计方法学和支持环境等几方面对SCOP进行扼要的介绍.

关键词　语义模型,面向对象,设计方法学,软件构造.

自有计算机以来,计算机软件生产自动化就一直是困扰人们的一个问题.随着计算机技术的进步,人们对软件生产自动化的要求不断提高.从汇编语言到高级语言,这是软件生产自动化第一次质的飞跃,实现了从高级源码到机器代码变换的自动化.促成这次飞跃的决定因素是编译理论和技术的完善.70年代结构化程序设计的研究,从程序结构和设计风格上提出了一些基本的原则和方法,奠定了程序设计方法学的基础,这一阶段的研究工作是卓有成效的.继之到70年代末80年代初,人们曾把软件生产自动化的希望寄托于程序综合和程序变换.事实已经证明,这种努力基本上是不能成功的,原因在于其研究目标囿于低一级的程序设计活动,做起来往往事倍功半,即使对于小规模程序设计,这种方法也难以奏效,更何况对于大规模软件的设计呢?

问题在于传统的软件结构和设计方法难以适应软件生产自动化的要求.传统软件以过程为中心进行功能组合,软件的扩充和复用能力都很差.虽然对于大规模软件系统的设计,提出有如模块封装、数据抽象等一般性原则,但仍缺乏相应的实现机制和环境支持.

从80年代开始的面向对象机制的研究,为软件生产自动化带来了新的生机.面向对象的软件,以数据为中心设计;与面向过程的传统软件相比较,面向对象软件本身的内部结构已发生质的转变,由此带来良好的可构造性、可扩充性和可复用性.面向对象研究的一大贡献在于改变了传统软件系统的结构,而正是这种结构,可使软件用构造的方法生成,这与软件生产自动化的要求是相适应的.[1]本文提出的用于软件构造的语义模型和规范就是在如

* 本研究得到国家"八五"攻关项目、国家863高科技项目基金资助.作者**冯玉琳**,1942年生,研究员,博士导师,主要研究领域为系统模型和语义理论,软件设计方法学,软件工程环境.**黄涛**,1965年生,副研究员,主要研究领域为面向对象软件构造,软件设计方法学,语义理论,软件工程环境.**李京**,1966年生,副研究员,主要研究领域为软件工程环境,程序设计方法学,系统模型.

本文通讯联系人:冯玉琳,北京100080,中国科学院软件研究所

本文1994－02－24收到(注:作者要求与国家863高科技项目成果论文一同发表,故推迟至今)

此背景之下产生的. SCOP 是一种基于对象的软件构造模式,来自 SoftwareConstruction =Objects+ProcessControl的缩写,其含义是:软件是对象模块的复合,而软件设计则是对象模块经进程控制复合的构造生成. 本文将从语义模型、描述语言、设计方法学和支持环境等几方面对这种新的设计风范进行简要的介绍.

1 语义模型

传统的数据模型(如 ER 或 ERA 模型)建立于实体、关系和属性之上,缺乏数据抽象的能力,也没有表示动态行为特征的机制. 而另一方面,关于系统动态特性的研究如 Petri 网、进程代数 CCS 等,却又是与系统的静态结构方面分开进行的. 这样,人们往往使用一种模型(如代数模型[2])表示系统的静态方面,而使用另外一种模型表示系统的动态方面,2 种模型难以很好的结合.

对象语义模型是两者的有机结合. 对象是数据封装的载体,对象提供一组操作供外界通过消息驱动观察对象属性或改变对象的内部状态. 对象的内部属性和对象之间的关联是对象的静态结构,为了能完整的反映对象的行为特征,对象的语义模型还应给出对象的动态变化历史.

对象一旦初始建立,就在对象操作的作用下不断更新变化. 作用在初始对象上的一串对象操作的序列,称为对象踪迹(trace). 对象踪迹决定对象演变的历史. 因此,从时序模型的观点出发,可以将对象定义为对象操作、对象属性和对象踪迹的集合.

定义 1. 对象 $ob=(X,A,\Lambda,\alpha)$,这里,X 是对象操作的集合,A 是对象属性的集合,Λ 是对象操作序列即踪迹的集合,α 是映射:$\alpha: X^* \to 2^{\{(a,b)\mid a\in A, d\in type(a)\}}$ 给出合法事件序列下的属性值集合.

基于以上给出的对象定义,可以给出对象继承、对象复合等重要概念的语义解释.[3]

对象通常是作为一个类的成员而存在. 对象类表示这样的一簇对象,它们具有相同的一组对象操作和对象属性,且其对象踪迹满足相同的一组对象约束. 从对象类生成对象,需经过命名和初始.

定义 2. 对象类定义为五元组 $C=(I,A,X,C_{st},C_{dy})$

这里,I 是一个命名系统,为每一个对象实例分配一个系统唯一的标志名字;A 和 X 的意义同前;C_{st}是一组用一阶谓词逻辑公式表示的对象静态约束的集合,它规定了对象的合法状态;C_{dy}是一组用时序逻辑公式表示的对象动态约束的集合,它规定了对象踪迹的合法状态之间应该满足的时序制约关系.

2 描述语言

SCOP 是基于对象语义模型提出的一种用于软件自动化构造的结构框架和方法学. 其中,对象的结构和行为是用对象定义语言 ODL 描述的. 对象是对数据和操作进行封装的自主单元,对外提供界面规定的服务,是现实世界中实体的自然抽象.[4]但是,仅有对象作为软件模块分解和复用的机制是不够的,对象无法提供对整个系统设计和复杂演变过程的控制(即使可以,实际上也是表示不清楚或不方便的).[5]为此,SCOP 利用进程控制语言 PCL,模

拟人们选择和操作对象模块并将它们组装成软件系统的过程.

2.1 对象定义语言

对象是软件构造的基本单元.对象由数据及其所提供的服务组成.对象的内部数据对外界是不可见的,外界只有通过发送消息来获得对象的服务或改变对象的内部状态.对象理解是通过对象界面的语义描述来实现的.SCOP 遵循对象定义和对象实现相分离的原则,在 ODL 中的对象类规范分为两部分,即对象定义部分和对象实现部分:

⟨class－specification⟩::=⟨class－definition⟩⟨class－body⟩

对象类定义在抽象数据域上描述对象,与对象类的具体实现无关.对象是对象类的实例,对象的理解是在对象类定义的抽象层次上进行.SCOP 将对象作为处理的基本单元.通常意义下的代码模块及对象类本身也都是预先定义的系统类的对象.

对象类规范的形式框架如下:

ClassSpec	类名[(类参)]
Description	非形式描述
Category	范畴
Inherit	父类表
Assume	引用表
Observation	属性
Operation	操作
Ensure	操作的语义规定
Constraint	行为约束规定
End	

为了增加复用和体现类之间的概括和特化关系,类之间可以通过父类/子类的继承关系构成类层次结构(类格).继承类继承其祖先类的所有内部结构,包括属性和操作,从而支持代码复用;继承类还继承其祖先的行为特征,包括静态约束和时序约束.当然,在子类中可以添加属于自己的新的属性和操作,也可以细化或者重定义父类的操作内容,但要求这些细化或修改必须保证子类和父类语义的一致性,即

(Ⅰ)子类对象的行为要符合父类对象的行为约束.

(Ⅱ)对于子类中的重定义操作,其操作规范的前条件不强于父类中相应的前条件,后条件不弱于父类中相应操作的后条件.

ODL 中将对象类作为类型来处理,子类体现子类型的概念.因此子类对象可以出现在父类对象的任一可出现的位置上,作为父类对象使用.ODL 中的类型包括原始类(如 Integer),不含参数的类(如 Person)以及实例化的含参类(如 Stack(Integer)).类型 Y 与类型 X 是兼容的,当且仅当下列条件之一满足:

(1)X 和 Y 是相同的类型

(2)X 和 Y 不是原始类,Y 是 X 的子类

(3)X 和 Y 不是原始类,X 的形式如 $P(U_1,U_2,\cdots,U_n)$,而 Y 是类 $P(V_1,V_2,\cdots,V_n)$的子类,且对于 $i=1,2,\cdots,n$,类型 U_i 与类型 V_i 是兼容的.

同一个 ODL 对象类定义,允许有不同的对象类实现,选择不同的程序语言和算法给出

对象类的实现描述.为此,在对象类规范的实现部分,必须提供与实现相关的环境信息,包括实现基语言说明(如 Source:C),语言库说明(如 Library:Graphic.lib,Math.lib),编译器说明(如 Compiler:cc)等.常见的高级语言如 C,Pascal,Fortran,4GL 等,不具备访问对象的机制,因此,在使用它们作为对象实现语言时,要对它们作适当的自然扩充.

对象类实现部分的形式框架如下:

ClassBody	类名[(类参)]
Version	版本信息
Execution	环境信息
Implementation	对象类的语言实现
Transformation	抽象数据到具体数据的映射

以下以栈类为例,给出其规范的定义和实现2部分.

```
ClassSpec Stack(ITEM)
  Assume Array(ITEM)
  Observation
    Integer NumOfElement;
    Array(ITEM) Vals;
    ITEM Top;
    Boolean IsEmpty;
    Boolean IsFull;
  Operation
    Init(Integer);
    Push(ITEM);
    Pop;
  Ensure
    for st;Stack;t;ITEM;len;Integer;b;Boolean;
      st.Init(len) Satisfies
        Pre len>0;
        Post st'.Vals.Upper=len and st'.Vals.Lower=1 and st'.NumOfElement=0;
      st.Push(t) satisfies
        Pre st.NumOfElement<st.Vals.Length;
        Post st'.NumOfElement=st.NumOfElement+1 and st.Vals.(Entry st'.NumOfElement)=t;
      st.Pop Satisfies
        Pre st.NumOfElement>0;
        Post st'.NumOfELement=st.NumOFElement-1;
      st.Top return t Satisfies
        Pre st.NumOfElement>0;
        Post t=st.Vals.(Entry st.NumOfElement) and st'.NumOfElement=st.NumOfElement;
      st.IsEmpty return b Satisfies
        Post if (st.NumOfElement>0)
             then b=false
             else b=true;
      st.IsFull return b Satisfies
        Post if(st.NumOfElement=st.Vals.Length)
             then b=true
             else b=false;
  Constraint
    NumOfElement>=0 and
    NumOfElement<=Vals.Length;
```

```
    End Stack(ITEM)
ClassBody Stack(ITEM)
  Version "Stack 1.0";
  Execution
    Source:C
    Compiler:CC,V4.0;
  Implementation
    Typedef struct{
      int Number;
      Object Array(ITEM) Values;
      }Stack;
    Void Init(int length)
    {
      self.Number=0;
      self.Values=Array(ITEM).Create(1,length);
      return;
    }
    Void Pop()
    {
      self.Number--;
      return;
    }
    Object ITEM Top()
    {
      return(self.Values.Entry(self.Number));
    }
    Void Push(Object ITEM t)
    {
      self.Values.Store(++self.Number,t);
      return;
    }
    int IsEmpty()
    {
      return(self.Number);
    }
    int IsFull()
    {
      return(self.Number==self.Values.Length);
    }
  Transformation
    NumOfElement⇒Number;
    Vals⇒Values;
  End Stack(ITEM)
```

2.2　进程控制语言

对象是数据和操作的封装载体,组装在一起才能构成完整的系统.软件开发过程也是软件[6],SCOP 规范用进程控制语言 PCL 来描述对象的构造组装过程.由于面向对象的软件构造从本质上说是过程式的.因此 PCL 语言采用过程式风范比较合适.PCL 的外部语法采用传统的过程式程序设计语言的控制结构,包括顺序、选取和重复3种基本形式;PCL 的内部语法是以原语形式描述的活动(Activity).每个活动表示相对独立的完成一定功能的任

务.例如,

·Selecting	从对象库查询和选取对象
·Defining	定义 ODL 新对象
·Caching	置对象于内存缓冲
·Deleting	从内存缓冲中删除对象
·Showing	显示内存缓冲存储对象
·Storing	永久存储对象
·Waiting	暂停等待交互会话
·Main	定义主程序
·Testing	对象测试
·Verifying	对象验证
·Linking	对象连接
·Embedding	对象嵌入
·Compiling	对象联编

PCL 语言还允许基本活动的并发执行.于是,软件系统的构造组装过程就可以被模拟为用 PCL 语言表示的一系列可以顺序或并发执行的活动.解释执行 PCL 程序的结果输出是目标软件系统的代码.如此构造的目标软件代码可在 SCOP 环境中运行或脱离 SCOP 环境在相同的操作系统平台上运行,取决于 PCL 程序规定采用的对象联编方式是对象连接还是对象嵌入.

PCL 语言是用程序的形式控制软件的构造组装过程.SCOP 环境还提供菜单界面供用户交互式的实施 PCL 控制.不管采用何种形式,控制的活动都是以对象为基础的.这就使软件设计的层次提高,开发人员不再需要关注底层对象的细节,并要求尽可能支持对象模块的复用,大大提高编程效率.

3 支持 SCOP 软件构造的方法学和环境

面向对象软件的特点和组织结构与面向过程的软件有很大不同,传统的适合于过程式软件的"瀑布"型软件生命周期模型显然不能适合于面向对象软件.[7,8]

面向对象的软件设计方法学要求以对象为中心组织整个设计活动.对象作为可唯一标识的实体,封装了一组属性及其相关操作,提供了一种结构化系统设计的抽象手段.SCOP 软件系统的结构是基于对象构造的,系统开发的各个阶段(分析、设计和实现)都是统一的按照相互作用的对象来组织的,各个阶段的过渡比较平滑,不会有象"瀑布"型的结构化分析和设计中从数据流图到模块图那样大的跳跃.

SCOP 软件开发方法采用"喷泉"软件生命周期,将软件开发过程划分为概念模型分析、系统设计、对象设计、对象实现和系统组装等5个阶段.

概念模型分析的主要目标是建立系统模型,具体包括系统分解和对象识别.系统模型中的对象是现实世界中的客观实体对象的抽象,结构清晰,易于理解且易于描述规范.虽然面向对象的软件开发是以对象为核心的,但我们并不要求系统模型分析时对子系统的分析也必须按对象观念进行.子系统模块涉及到较多的对象以及它们之间的相互作用,SCOP 方法

学将子系统模块区别于对象，可在 PCL 语言中给出定义，用过程模型来描述对象之间的相互作用和复合构造.

在分析阶段面向问题域建立对象模型和过程模型，进入系统设计阶段，就从问题域转入程序域，给出模型对象和过程的规范描述.

SCOP 方法学强调软件模块的复用和软件构造合成，因此，在对象设计和实现时，并不要求所有的对象均要从头开始设计，而是要充分复用以前的工作. 在支持 SCOP 软件开发的环境中检索对象库，若是对象库中已有的，则可再复用；否则，重新用 ODL 语言定义新的对象. 对象复用的途径有3种形式：

(Ⅰ)实例化含参类，用具体的类参去替换含参类的类参，从而得到满足自己要求的对象类，例如：设库中已有栈类 STACK(T)，T 是类参，用整形替换类参 T，就得到整数栈

INT_STACK=STACK(INTEGER)

(Ⅱ)通过继承，对库中已有的类添加或重定义新的属性、操作或约束来设计满足自己要求的对象类.

(Ⅲ)对于库中能够完全满足自己要求的对象类，直接纳入自己的应用.

一个支持 SCOP 软件构造的环境原型在 SUN 工作站和486微机上实现，UNIX 操作系统支持，包括概念模型分析子系统，ODL 类分析子系统，PCL 控制子系统，巡航理解子系统，对象管理服务子系统，性能约束推理子系统等部分. 各部分之间的关系和 SCOP 软件构造流程示意如图1所示.

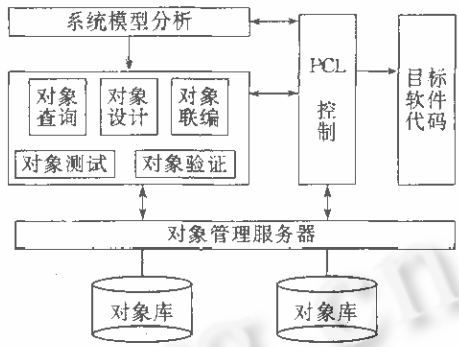

图1

支持 SCOP 环境的关键技术包括软件复用和组装技术，对象分布存储技术及软件规范的形式化技术等. 根据 SCOP 方法学原则，已开发2个示例性应用系统. 实践表明，这种基于对象的软件构造模式及 SCOP 环境对于软件设计质量和效率的提高，其优越性是非常明显的.

参考文献

1 Meyer B. Object oriented software construction. Prentice Hall, Englewood Cliffs, N. J., 1988.

2 Breu R. Algebraic specification techniques in object oriented programming environments. LNCS 562, Springer Verlag, 1991.

3 冯玉琳，李京，黄涛. 对象语义理论和行为约束推理. 计算机学报，1993，**16**(**11**)：823～828.

4 Saake G. Descriptive specification of database object behavior. Data and Knowledge Engineering, 1991, (**6**): 47～73.

5 Aksit M, Bergmans L. Obstacles in object oriented software development. Proc. of OOPSLA'92, 1992. 341～358.

6 Osterweil L. Software processes are software too, Proc. of 9th ICSE, 1987. 2～13.

7 Henderson-Sellers B, Edwards J M. The object oriented systems life cycle. CACM, 1990, **33**(**9**): 142～169.

8 Neighbors J M. The evolution from software components to domain analysis. Intl. J. of Software Engineering and Knowledge Engineering, 1992, **2**(**3**): 325～354.

OBJECT ORIENTED SOFTWARE CONSTRUCTION

Feng Yulin Huang Tao

(*Institute of Software The Chinese Academy of Sciences Beijing* 100080)

Li Jing

(*Department of Computer Science University of Science and Technology of China Hefei* 230026)

Abstract The development of object oriented techniques causes a great changes to traditional method of application software design. The authors of this paper propose a new development model for object oriented software construction, in which software is a composition of object modules, and software design is a construction of objects through process controls. This paper contributes to a brief description of the approaches, including semantic model, specification, design methodology and support environment etc.

Key words Semantic model, object oriented, design methodology, software construction.