第 6 卷第 12 期
1995 年 12 月

软 件 学 报
JOURNAL OF SOFTWARE

Vol. 6, No. 12
Dec. 1995

面向对象软件规格语言的设计*

全炳哲　金淳兆

（吉林大学计算机科学系，长春 130023）

摘要　本文提出一种面向对象软件的形式描述语言 JOOSL，用它可描述面向对象软件需求规格、概要设计和详细设计. 从描述方法角度看，需求规格和概要设计的描述在很大程度上相同，这就反映了 OO 开发模型中需求和设计之间的重叠. 在这些描述中用抽象方法描述数据和操作；详细设计中确定算法细节和数据的表示. JOOSL 认为对象是一种抽象的状态机，继承是行为特性的共享.

关键词　面向对象开发模型，面向对象设计，形式规格语言，自动程序设计，软件重用.

软件工程的重要目标之一是实现软件开发过程各阶段的自动化. 软件自动化的前提是形式化，包括软件需求规格的形式化、软件设计规格的形式化和算法描述的形式化. 随着对面向对象软件开发技术的推广，人们也开始研究了面向对象软件的形式规格描述问题. Z 语言是一个较著名的软件规格形式描述语言，其基础是集合论和类型化的一阶谓词逻辑. 用 Z 语言描述系统时，通常把系统看做是一个状态机，在这种意义下可用 Z 语言描述面向对象系统[1]. 但 Z 语言未提供描述面向对象系统的足够机制. Object－Z[2]是 Z 语言的一种扩充，其中引入了类及其继承的描述机制，其目标是用于描述面向对象系统. Z＋＋语言[3]是 Z 语言的另一种扩充，其中引入了 Class 的描述单位. 从 70 年代初开始，广泛研究了抽象数据类型的代数规格描述技术，OBJ3 语言[4]是一种基于 ADT 的代数规格描述方法的形式规格描述语言，从数据抽象角度看，它反映了面向对象的某些思想. COLD－K 语言[5]是另一种基于代数规格描述技术的面向对象设计形式描述语言，它提供了设计规格的描述机制和算法描述的机制，在设计规格的描述中采用一种扩充的代数规格描述方法和归纳定义方法. COLD－K 是一种软件设计的核心描述语言，其中缺乏一些较高级的描述机制. OBJ3 和 COLD－K 都没有提供继承机制. 面向对象软件的特点是它有机地结合了数据和操作，因此面向对象规格语言必须能够统一地描述数据和操作，这一要求对高层描述尤为重要.

1　JOOSL 的设计思想

1.1　面向对象软件的描述层次

* 本文 1994－06－27 收到，1994－09－23 定稿

本文研究得到“八五”攻关项目、国家“863”计划和国家自然科学基金的支持. 作者**全炳哲**，1961 年生，副教授，主要研究领域为规格语言，面向对象方法，程序自动化，软件工程. **金淳兆**，1937 年生，教授，主要研究领域为规格语言，面向对象方法，程序自动化，软件工程.

本文通讯联系人：全炳哲，长春 130023，吉林大学计算机科学系

软件开发要按一定的开发步骤进行,软件开发过程大致可分为软件需求分析、软件设计及实现.但不同的软件开发模型对各阶段的再分解及相应的活动有着不同的解释.在面向对象软件开发的喷泉模型[6](Fountain model)中把软件开发过程分为6个阶段,但着重强调不同阶段之间的重叠,认为面向对象软件开发过程是渐进的进化过程.基于喷泉模型,Henderson—Sellers等人提出了面向对象生命期的O—O—O方法[7].Hodge等人提出的面向对象开发方法[8]把开发阶段分为5个阶段,它也体现出分析和设计之间的重叠.从面向对象软件开发模型角度看,不需要(或不应该)严格区分不同的开发阶段,开发过程是一个螺旋形的渐进过程.从软件开发的形式描述和软件自动化角度看,应该用形式描述方法定义软件的需求规格和软件设计结果.本文认为可用如下3个不同的层次描述面向对象软件:(1)需求规格描述;(2)概要设计描述;(3)详细设计描述.在面向对象分析(OOA)中,首先,要准备软件需求文档;其次,要识别对象(这里所说的对象是一组实际对象的代表,对象的形式描述就是类),包括确定对象属性和服务;第三,要识别对象之间的关系,包括关联(Association)关系、聚合(Aggregation)关系和分类(Classification)关系.因此需求规格描述中应给出对象的属性描述、服务描述、引用关系和继承关系的描述.由于OOA的主要目标是严格确定用户需求,所以在对象的描述中主要考虑系统中的各对象和关系,而不考虑实现方法.OOA到面向对象概要设计(OOPD)是一个模型的扩充过程.OOA的各层模型化了"问题空间",而作为OOA各层扩充的OOPD,则模型化一个特定的"实现空间".这种扩充主要从增加属性和服务开始[9].在OOPD阶段中,首先,要选择实现服务的算法特性,这就常常需要引入附加的操作,这种操作是实现细节,需要对外隐蔽.因此,要确定类的移出操作和私有操作;其次,要确定软件的体系结构,即严格确定类、类的继承关系和类之间的引用关系.虽然OOA和OOPD的目标不同,但从描述形式上看,需求规格和概要设计的描述在很大程度上相同.这就反映了OO开发模型中需求和设计之间的重叠.对于我们设计的JOOSL中,用JOOADL描述如上两个层次,这些描述统称为规格描述.面向对象详细设计(OODD)阶段的目标是确定软件的具体实现策略,包括数据的具体表示方式和操作的实现方式.在详细设计中仍然采用OOPD中得出的软件体系结构,但OODD从具体实现角度进一步加细OOPD,主要表现在引入类的实例变量,并确定操作的算法细节.由于OODD仍属于设计,所以从形式描述角度看,应该采用抽象度较高、且独立于具体程序设计语言的算法描述语言.在JOOSL中,用JOODDL描述详细设计,这一描述称为实现描述.

1.2 对象的描述

面向对象软件的基本特点是:(1)系统由一些对象构成;(2)对象有自己的状态,并提供若干服务;(3)对象的描述构成类.对象可用状态机描述,通过操作可观察对象的当前状态,同时也可以改变对象的状态.

在面向对象的规格描述中,主要描述软件的需求和体系结构,而不具体确定对象中数据的具体结构和操作的实现细节.抽象地描述数据的一种较有效途径是用操作间接地描述它们.在JOOSDL中,对象所提供的操作可分为函数、谓词和过程等3种.函数用于观察对象的当前内部状态;谓词用于测试对象当前状态下的某种条件的成立与否;过程用于修改对象的当前状态.函数没有副作用.函数的另外一个特点是它们可以起类似于Smalltalk语言中的实例变量的作用,因此整型实例变量可定义为零元函数 f:—>integer,其函数值代表实

例变量的值.这种描述方法的根本出发点是完全从用户角度描述对象.用户关心的是如何了解对象状态,而不关心如何表示对象状态.实际实现中,对象状态可用一些变元记录,也可以用一段代码实现.谓词对应于布尔函数或布尔型实例变量.谓词的作用类似于函数,但它没有返回值,它们在某种状态下要么成立要么不成立.过程有两个方面的作用:一是用于描述完成某种功能的操作;二是可用于修改对象状态.对过程返回值可有可无.由于用函数或谓词描述对象状态,所以对象状态的改变就是指修改函数的返回值或改变谓词的成立与否.

在面向对象软件的实现层中,主要描述类的实现方式,包括实例变量的表示和操作的算法描述.JOOSL 语言中,规格层的描述是描述实现层的基础.实例变量对应于规格描述中的相应函数或谓词,其余操作成为对象所提供的操作.实现描述相当于软件的详细设计的描述,因此 JOODDL 是一种描述面向对象软件的形式化的 PDL.通常意义下的 PDL 中都提供描述算法结构的各种控制结构,包括顺序结构、分支结构和循环结构.另一方面,作为详细设计的描述语言,PDL 允许用受限的自然语言描述条件分支、循环条件和一些处理.JOODDL 语言的设计目标是:(1)它应是形式描述语言,即它应该具有精确的语法规则和明确的语义.(2)描述详细设计,应能够描述详细设计,但抽象度高于一般的高级程序设计语言.JOODDL 语言中算法描述的基本手段是语言所提供的控制结构,包括顺序结构、分支结构和循环结构;描述具有较高抽象度的算法的基本手段是:(1)可用一种受限的谓词公式描述分支结构的条件和循环结构中的循环控制条件;(2)提供处理过程的抽象描述语句.

1.3　继承的描述

继承是面向对象技术的重要特点之一.从严格意义上讲,不支持继承的"面向对象"语言应称为基于对象的语言.但目前对继承有多种不同的解释.对继承的解释大体可分为两大类:一是把继承看做是面向实现的概念,其目标是实现代码的共享.多数面向对象程序设计语言都把继承认为是面向实现的概念,可惜这种概念没有明确的语义;二是把继承看做是语义概念,其目标是共享行为特性.在 JOOSL 中,把继承看做是语义概念,其基础是 Barbara Liskov 的替换原则[10].类型层由子类型和父类型构成.子类型的行为特性是父类型的所有行为特性加上其它行为特性.若对类型 S 的每个对象 O_S,存在类型 T 的对象 O_T,使得对所有用类型 T 定义的程序 P,当用 O_S 替换 O_T 时,不改变程序 P 的行为特性,则 S 是 T 的子类型.类型的作用在于规定对象所满足的条件,即类型是关于对象的断言.对象 O 满足类型 T 的断言,则 O 具有类型 T.这就意味着一个对象可能具有多个不可比较的类型[11].我们的目标是使父类和子类间的关系反映父类型和子类型的关系.为使继承反映类型的层次关系,在 JOOSL 中规定:(1)子类中重载定义父类的操作时,不可改变操作的参数个数和顺序,且相应参数及返回值类型要相容,即要求重载操作的参数及返回值类型是被重载操作的相应参数或返回值类型的子类型;(2)重载操作和被重载的操作的语义要一致.但由于通常无法自动验证这种语义的一致性,需要人工验证语义的一致性;(3)子类中附加定义的过程不可修改父类中定义的函数和谓词(或实例变量).

2　规格描述

JOOADL 是一个强类型面向对象设计规格描述语言,其中整型、实型、字符型、布尔型和表作为基本型.类是规格描述的基本单位.类型就是基本型和类.类的描述形式如下.

〈类规格〉::=class〈类名〉

inherit〈类名序列〉;

import〈类名序列〉;

export〈操作的 signature 序列〉

spec

〈私有操作的 signature 序列〉

〈操作的定义序列〉

end

用函数、谓词和过程操作描述类规格. 函数的形式为:

function〈函数名〉$(t_1:T_1,t_2:T_2,\cdots,t_n:T_n):T$

$$\text{compute } t:T\ P(t,t_1,t_2,\cdots,t_n); \tag{1}$$

其中 P 为谓词. 函数值是满足谓词 P 的对象 t. 在函数的定义中,参数值是任意的,但省略了对参数的全程量词,即(1)式是如下定义的简写:

function〈函数名〉$(t_1:T_1,t_2:T_2,\cdots,t_n:T_n):T$

$$\forall\ t_1:T_1,\cdots,\forall\ t_n:T_n\exists\ !\ t:T\ P(t,t_1,\cdots,t_n); \tag{2}$$

其中$\exists\ !\ xQ(x)=_{df}\exists\ x(Q(x)\wedge\forall\ y(Q(y)\rightarrow I(x,y)))$即表示"论域中恰有一个个体". 为保证计算的确定性,$\exists\ !$是必须的. 但从逻辑角度看,(2)式可能不是一个定理,从规格描述角度看,它是一种错误的规格说明,应避免.

函数的第二种描述方法是归纳定义方法,其描述格式如下.

function〈函数名〉$(t_1:T_1,t_2:T_2,\cdots,t_n:T_n):T$

$IND\ for\ t_i:T_i(PB(t_1,t_2,\cdots,t_n),PI(t,t_1,t_2,\cdots,t_n))$;

它是施归纳于 ti,其中 PB 和 PI 是谓词,分别对应于基始和归纳. 这里 T_i 应是良序类型,包括非负整数、非正整数和表.

在函数的描述中,可使用表示下一个状态的算子,用撇号(')表示. 例如,a'表示下一个状态下的变元 a 或函数 a 的值. 除此之外,还允许使用消息发送,但消息中只允许引用函数或谓词操作. 谓词的描述类似于函数的描述,但谓词不表示求值,只表示在当前状态下某种条件的成立与否. 函数和谓词的共同特征是它们表示对象的内部状态,但不修改对象的内部状态. 对象的状态可分为不依赖于其它状态的独立状态和依赖于其它状态的非独立状态. 相应地函数和谓词也可分为独立和非独立的函数或谓词. 函数和谓词的描述中,对于零元的独立函数和谓词,可不给出具体描述. 这种函数或谓词的结果是不经过任何处理的相应的当前状态.

过程是修改对象状态或完成某种功能的操作,它的描述形式如下.

procedure〈过程名〉$(t_1:T_1,t_2:T_2,\cdots,t_n:T_n)[:T]\ [MOD\ f_1,f_2,\cdots,f_m]$

$[\text{compute } t:T]\ P(t,t_1,t_2,\cdots,t_n)$;

其中 $MOD\ f_1,f_2,\cdots,f_m$ 是本过程的修改权限说明,表示本过程将修改 $f_1,f_2,\cdots,f_m$,是函数或谓词名. 修改权限说明中,只能列出独立函数或谓词. 过程的描述中,不仅可使用撇号算子和消息发送,而且可使用从外部获取变元值的算子(用问号(?)表示)和输出变元值的算子(用感叹号(!)表示). 例如,$a?$ 表示从外部获取 a 的值,$a!$ 表示把 a 的值输出到外部,这里要

求 a 的类型必须为整型、实型或字符型.?算子和!算子的直观意义分别是输入和输出操作.

例 1:函数 f 将一个大于 2 的偶数分解成两个素数之和,并返回其中一个素数.

function f(i:integer):integer

compute n:integer (((i>2) and (i/2=0)) imply (n>0 and n<i and self.prime(n) and self.prime($i-n$)));

predicate prime(n:integer)

(n>2) and forall k:integer ((k>=2 and k<=$n-1$) imply (n/k <> 0));

3 实现描述

JOODDL 是实现层的描述语言,其中基本型包括整型、实型、字符型、布尔型和数组型.实现层的描述单位仍然是类.类的描述主要由接口描述、实例变量描述和操作描述构成.类的描述形式如下.

〈类实现〉::=class〈类名〉
　　inherit〈类名序列〉;
　　import〈类名序列〉;
　　export〈操作的原型序列〉
　　implementation
　　var
　　〈实例变量说明〉
　　method
　　〈操作的实现描述〉
　　end

实例变量的类型可以是 JOODDL 的基本型或类.在实现描述中,操作分为函数和过程,有返回值的操作称为函数,没有返回值的操作叫做过程.操作的实现是用一些结构化的控制语句描述,JOODDL 提供 if then 语句、if then else 语句、for 语句、while 语句、repeat until 语句、消息发送语句、输入/输出语句、return 语句和 compute 语句.除 compute 语句外,其它控制语句的结构类似于 Pascal 语言中的相应控制语句,但 JOODDL 语言中的分支和循环语句的抽象度高于一般程序设计语言的控制语句.

分支条件和循环控制条件都可以是受限的谓词.compute 语句也利用受限谓词描述处理,这一计算语句的形式如下:

compute〈基本型变元或受限良序型变元序列〉:〈受限谓词〉

其含义是计算满足受限谓词的相应变元.

所谓受限谓词的含义是:首先,形式上和一阶谓词逻辑中的谓词公式相同;其次,作为详细设计的描述手段,必须提供完整的算法特性.对谓词公式的限定体现在如下 5 个方面:(1)公式中出现的变元必须已定义,变元可以是实例变量、局部变量或操作的形式参数.(2)受限谓词 P 中,量词所约束的变元的类型必须是离散量,且有上、下界的规定.(3)谓词公式应体现完整的计算特性,例如,允许 compute x:x=1,但不允许 compute x:x>1,因为后一种描述中,x 的值是不确定的.(4)谓词中允许出现带撇号(')的变元,撇号是表示下一个状态的

算子.不带撇号的变元值隐含指前一状态下所取的值.但是,不允许出现具有循环定义形式的谓词.本限制的目的是为了防止出现某种方程的直接描述,而解方程含有明显的算法特性.例如,不允许出现 $x'=y'+1$ *and* $y'=2x'+3$ 的形式.(5)等式公式的左边必须是带撇号的变元,而不允许是复杂表达式.

例 2:下面是例 1 的详细描述.

```
function f(i:integer):integer
var
  n:integer;
begin
  if ((i>2) and (i/2=0)) then
    compute n(2:i):(self.prime(n) and self.prime(i-n));
    return n;
  else
    return 0;
end
function prime(n:integer):boolean
begin
  if ((n>=2) and (forall k:integer 2<=k<=n-1 (n/k <> 0))) then
    return true;
  else
    return false;
end
```

4 结束语

JOOSL 是以 COLD-K 为基础,吸收 Z 语言的描述方法和 Recos 语言[12]语法格式的一种面向对象软件设计描述语言.但 JOOSL 与这些规格描述语言有较大的差别.JOOSL 有如下几个特点:(1)支持面向对象技术的所有特性,包括类、消息发送、多继承、多态和动态结合;(2)可描述面向对象软件的需求规格、概要设计规格和详细设计规格;(3)在规格描述中,用抽象操作描述对象状态;(4)以面向对象的 PDL 级语言(JOODDL)支持实现层的描述.

JOOSL 的重要目标之一是研究面向对象程序自动化技术.基于 JOOSL 我们已开发了概要设计到详细设计的转换系统和详细设计到 C++程序的转换系统.概要设计到详细设计的转换是规格到算法的转换,其中需要部分的人工干预;详细设计到 C++的转换中实现了自动转换,其中分支条件、循环控制条件和 compute 语句的转换是不同抽象层之间的转换.本文是一种探索性的工作,今后需要进一步提高 JOOSL 的描述能力,继续研究规格到算法的转换技术.

参考文献

1 Anthony Hall. Using Z as a specification calculus for object-oriented system. LNCS 428, 290-318.

2 David Carrington *et al*. Object-Z: an object-oriented extension to Z. In: Vuong S ed. Formal Description Techniques(FORTE'89), North Holland, 1990.

3 Lano K. A specification-based approach to maintenance. Software Maintenance: Research and Practice, 1991, **13** (**4**):193-213.

4 Gougen J A, Winkler T. Introducing OBJ3. Report SRI-CSL-88-9, SRI International Computer Science Laboratory.

5 Jonkers H B M. An introduction to COLD－K. Algebraic Methods: Theory, Tools and Applications, Berlin: Springer－Verlag, 1990. 139－200.

6 Henderson Sellers B, Edwards J M. The object－oriented system life cycle. CACM, 1990, **33(9)**: 142－159.

7 Henderson－Sellers B, Edwards J M. The O－O－O methodology for the object－oriented life cycle. ACM SIGSOFT Soft. Eng. Notes, 1993, **18(4)**: 54－60.

8 Hodge L R, Mock M T. A proposed object－oriented development methodology. Software Engineering Journal, March 1992. 119－129.

9 Coad P, Yourdon E. Object－oriented analysis. Yourdon Press, 1990. 邵维忠等译,北京大学出版社,1992.

10 Liskov B. Data abstraction and hierarchy. (addendum to) OOPSLA'87, 1987.

11 Palsberg J, Schwartzbach M I. Three discussions on object－oriented typing. OOPS Messenger 1992, **13(1)**: 31－38.

12 全炳哲,余江,金淳兆. 可重用构件及其描述语言. 软件学报,1994, **5(1)**: 42－46.

AN OBJECT－ORIENTED SOFTWARE SPECIFICATION LANGUAGE

Quan Bingzhe　Jin Chunzhao

(*Department of Computer Science, Jilin University, Changchun* 130023)

Abstract　JOOSL is an object－oriented formal specification language, which can be used to describe requirement specification, preliminary design and detailed design of an object－oriented software. From the view point of specification method, requirement specification is about the same as preliminary design specification, and it characterizes the overlap between requirement analysis and design. In these specifications the data and operations should be described abstractly. The detailed design is concerned with specifying algorithmic details and concrete data representations. In JOOSL, an object is considered as an abstract state machine and inheritance is defined as sharing of behavior.

Key words　Object－oriented development model, object－oriented design, formal specification language, automatic programming, software reuse.