# 对象描述语言及其指称描述*

黄 涛 冯玉琳 倪 彬

(中国科学院软件研究所计算机科学开放实验室 北京 100080)

李 京

(中国科技大学计算机系 合肥 230027)

**摘要** 在面向对象的软件构造中,对象被视为软件系统的基本构件.本文提出对象规范描述语言ODL(object description language),并给出其主要结构的SOP指称描述.SOP演算的理论范集给出对象的形式描述.SOP理论范集反映了对象的封装性,在这样的逻辑框架下,对象的属性(结构)和动作(行为)得以统一.对象聚合提供了由已有对象描述构造复杂对象描述的机制;而继承则可以扩充给定的对象描述并保持原描述的特性.此外,本文还区分了继承和子类这2个相似而又不尽相同的概念.

**关键词** 对象,指称语义,演算,对象语义描述.

通常的基于抽象数据类型的建模与基于实体的建模,强调的是结构的聚合、抽象和继承.目前,面向对象的建模除考虑结构特性外,还讨论行为的聚合和继承.本文所给出的一种对象描述也是基于这一思想,将有关对象的静态结构动态行为都封装在一个对象描述中.

面向对象的程序设计强调的是对象的操作界面;面向对象的数据库强调的是对象的观察结构.这里,我们将两者有机地结合起来,即对象的静态结构可以通过对象的属性来观察;而对象的动态行为则由该对象所提供的动作来展示.不同对象的动作之间可以同步通信.作为一个动态演变的实体,对象具有其内部状态和固有的标识符,在动作作用下,内部状态不断变化.对象除了有静态结构约束外,其演变历史也受动态约束规定.我们可将数据看作是内部状态不随时间改变的对象,这样就可将数据和对象统一看待.

对象描述通常被视为具有相似结构和行为的对象实例的描述,在ADT意义下称为类型,在面向对象程序设计语言中称为类.然而对象还有一个重要特性,即每个对象都具有唯一区别于其它对象的对象标识,所以我们认为对象类包含了刻画该对象类的实例的结构和行为的实体、对象实例集合、对象标识集合及从对象标识集合到对象实例集合的映射.

---

* 本文研究得到国家"八五"攻关项目基金、国家863高科技项目基金和国家自然科学基金资助.作者**黄涛**,1965年生,博士,副研究员,主要研究领域为软件工程,程序设计方法学,对象语义理论.**冯玉琳**,1942年生,研究员,博士导师,中国科学院软件研究所所长,主要研究领域为软件工程,程序设计方法学,面向对象的理论和技术.**倪彬**,1969年生,博士生,主要研究领域为面向对象的理论和技术.**李京**,1966年生,副教授,主要研究领域为软件工程环境,程序设计方法学.

本文通讯联系人:黄涛,北京100080,中国科学院软件研究所

本文1995-08-18收到修改稿

对象是可以聚合的,一个对象可以由一些成分对象构成,聚合对象和成分对象间体现了part—of或引用关系.为了增加复用和体现类之间的概括和特化关系,类之间可以通过继承关系构成类层次结构(类格).我们所讨论的聚合和继承是在对象意义下的而不是ADT方法中类型意义下的,即聚合(继承)是包含结构和行为的聚合(继承).同文献[1]等同类工作相比,我们这里同时强调对象的静态结构和动态行为.

对象描述语言必须提供描述简单对象、对象继承、对象聚合的机制.本文将针对这些问题进行讨论,定义对象描述语言ODL(object description language)及其主要结构的SOP语义,即将ODL描述指称为对象语义描述[2](理论范集[2,3]),ODL的语法成分将被指称为对象语义描述的相应成分.其它结构或动作符,如含参类,也是对象描述语言所必须具备的,这里我们的主要目的是为对象描述语言ODL的基本结构提供SOP语义.

结合代数方法和时序方法,基于对象的重要特性—封装性,SOP演算[2]提供了形式描述和推理对象(结构和行为)以及对象特性的工具.所定义的对象语义描述对应于对象描述.也就是说,简单对象的语义由对象语义描述给出,至于继承和聚合等对象构造结构的语义可以通过对象语义描述范畴上的适当操作给出.因此,对象描述的语义可由如下的部分函数给出:

$$M: CLASS \rightarrow \mathbf{Th}$$

其中$CLASS$是对象描述的范畴,**Th**是对象语义描述的范畴.这里,对一个对象描述$C$,我们分别以$OSig[C]$和$Axioms[C]$表示其对象语义描述的对象标记部分和公理部分,即$M[C]=(Osig[C], Axioms[C])$;并以$Sort[C]$,$Func[C]$,$Attr[C]$和$Act[C]$表示相应对象标记的类型、函数符号、属性符号和动作符号部分,即$OSig[C]=(Sort[C], Func[C], Attr[C], Act[C])$.

值得一提的是对象描述中还包含数据部分,它给出对象的上下文信息,即对象所处的环境信息,这些信息是与对象状态无关的.对象属性的论域和余论域以及动作的参数需由这些信息给出.

ADT方法用初始语义[1,4]解释对象的属性.然而这样的解释还是过于宽松,没有反映出对象(属性)的特性.我们应在对象的可能的上下文环境中解释对象的属性,这个上下文环境是对象描述的代数,它可归约为数据描述(包含对象的标识类型)的代数.更准确些,对象描述引用一些数据描述,如布尔函数类型和整数类型;对象描述中增加属性描述,这些属性应被解释为数据类型上的函数,在不同的上下文环境,属性的解释不一样,这样对象的上下文环境构成一个S5 Kripke结构.

第1节将给出简单对象描述及其SOP语义;第2、3节分别介绍继承、聚合和精化结构的语法和语义描述;第4节讨论对象类以及子类描述,并区分子类和继承这2个概念;最后给出一些结论.

## 1 简单对象的描述

### 1.1 语法描述

首先,对象描述语言必须提供描述简单对象的手段.对象是唯一标识的动态实体,具有自己的内部状态;对象的内部状态可以通过属性来观察;状态的改变只能通过该对象所提供的动作来实现;在对象的动态演变过程中,对象保持其标识不变.一般地,一个简单对象类描

述如下:

**ClassSpec** C

| | |
|---|---|
| **Assume** *data* | 引用类子句 |
| **Observation** *O* | 观察子句 |
| **Operation** *E* | 操作子句 |
| **Ensure** ensure | 操作的语义规定 |
| **Local Static Constraint** *LSC* | 局部静态约束子句 |
| **Local Dynamic Constraint** *LDC* | 局部动态约束子句 |
| **Global Static Constraint** *GSC* | 全局静态约束子句 |

**End** C

前面几项一般 OO 语言均可提供,下面我们仅简要介绍 ODL 的特色.

(1)操作语义的说明　关于对象状态改变的描述有很多种方法.在静态框架下,通常描述为一些过程,这些过程可能解释为机器指令,语义约束隐含在这些过程中;在动态框架下,对象的活动通常描述为状态转换,其语义可以通过状态转换的前后条件描述;在时序框架下,对象的活动可通过一组时序断言描述也可由前后条件描述.ODL 用动作的前后条件来描述动作的语义.前条件给出了动作执行前应满足的条件;而后条件给出动作执行结束后应满足的条件.前条件可分为 2 种:即关于对象当前状态的静态条件和动作执行所应满足的时序条件.静态条件和后条件是用多类一阶谓词逻辑公式来表示的;时序条件是用时序公式表示的.

(2)约束　动作的前后条件并不能完全规范对象的行为.我们在文献[5]指出对象具有静态和动态 2 方面约束.对象类描述中的约束可分为局部和全局 2 种.从语义上讲,局部约束只涉及一个对象,它规范了对象的合法状态;全局约束涉及多个对象,它规定了合法对象.局部约束又可分为局部静态约束和局部动态约束.局部静态约束表示对象的静态属性之间的制约关系,规定了对象的合法状态;局部动态约束表示事件动作之间或对象合法状态之间的时序制约关系.至于全局约束,我们只考虑全局静态约束.显然静态约束可用多类一阶谓词逻辑公式来表示;动态约束可用时序公式来描述.

### 1.2　简单对象的对象语义描述

显然,对简单对象描述,其对象语义描述的对象标记包含引用类的标记(这里,引用类是通常 ADT 意义下的抽象数据类型,我们以通常代数规范的标记表示)、该简单对象描述中定义的属性符号(对应于观察子句中的观察)和动作符号(对应于观察子句中的观察和操作子局中的动作),即

$Sort[C]=Sort(data)$

$Func[C]=Func(data)\cup\{f:C\to r\mid \textbf{constant } f:r\in O\}$

$Attr[C]=\{f:r\mid f:r\in O\}$

$Act[C]=\{Get\_f:r\mid f:r\in O\}\cup\{f:s_1\times\cdots\times s_n\mid f(t_1\ s_1,\cdots,t_n\ s_n)\in E, t_i=\textbf{in},\textbf{out}\}$

$OSig[C]=(Sort[C],Func[C],Attr[C],Act[C])$

相应于该简单对象描述的对象语义描述的公理集包含数据描述的公理(这里我们用 *data* 的代数规范的公理表示)、对应于 **Ensure** 子句、全局静态约束子句、局部静态约束子句和局部动态约束子句的公理.即 $Axioms[C]=Axioms[data]\cup M_f[LSC]\cup M_f[LDC]\cup M_f$

$[GSC]\cup M_{ensure}$. 其中

$M_{ensure}=\{enable(f)\rightarrow M_f[tc], enable(f)\rightarrow M_f[pre_c], M_f[post_c]|$

$f$ **satisfy**

**temporal** $tc$

**pre** $pre_c$

**post** $post_c$

是 *ensure* 中的 **Ensure** 子句$\}$

语义函数 $M_f$ 给出公式的语义解释,即将该简单对象描述中的逻辑运算解释为 SOP 演算的逻辑运算符,将含@的解释为 SOP 演算中的动作项和公式. 如以下的 **Ensure** 子句,

$f(x,y)$ **satisfy**

**temporal** $\mathbf{X}^{-}(a>1)$

**pre** $b>10$ **or** $a=2$

**post** $@a=20$

相应的对象语义描述包含如下公理:

$enable(f(x,y))\rightarrow \mathbf{X}^{-}(a>=1)$

$enable(f(x,y))\rightarrow b>10\vee a=2$

$[f(x,y)]a=20$

这样,我们就将简单对象的描述指称为对象语义描述.

## 2 继 承

继承是面向对象描述的重要机制,它使得我们可以用一些新的对象属性和动作来扩充给定的对象描述,并保持对象的封装性.

### 2.1 语法描述

ODL 支持多重继承. 尽管多重继承的引入会增加系统复杂性,但我们认为多重继承是对客观世界的自然反映,在支持软件设计的 ODL 语言中应当有多重继承机制. 继承结构增加如下描述:

**Inherit** $C_1$ **Rename** $rename_{C_1}$

…

$C_n$ **Rename** $rename_{C_n}$

其中 $rename_{C_i}=a_{i_1}$ **as** $b_{i_1},\cdots, a_{i_{ki}}$ **as** $b_{i_{ki}}$,为换名子句以解决多继承引起的名字冲突.

### 2.2 语义描述

继承类继承了祖先类的所有内部结构,从而支持代码共享;此外继承类还需继承祖先类的行为,这要求继承类不仅继承父类的结构(属性和动作),而且继承祖先类动作的语义、静态约束和时序约束,使得把这些约束添加到子类定义的语义约束中之后,所得的约束集合仍保持一致. 也就是说,继承类的对象语义描述的对象标记包含引用的数据描述的标记、祖先类的对象标记(如果有 **Redefine** 子句则要适当换名)以及在继承类中定义的属性符号和动作符号;而对象语义描述的公理部分包含祖先类的对象语义描述的公理(如果有 **Redefine**

子句则要适当换名)以及新增加的公理.

仅有这些是不够的,还没有反映对象的封装性.不同于分型,继承需保护对象的封装性,即在继承的描述中不可以修改被继承者所定义的属性.如作为特殊的继承,子类对象的描述中不可以修改父类对象所定义的属性.也就是说我们所定义的继承是保"封装性"的,继承的部分将不受以后的构造的影响,这一点可以通过对象语义描述之间的态射来描述.

每个 $C_i$ **Rename** $rename_{c_i}$ 定义了一理论态射[2] $m_i$: $M[C_i]\rightarrow M[C]$ 满足:

(1) $m_i(a_{i_j})=b_{i_j}$, $1\leqslant j\leqslant k_j$

(2) $m_i(a)=a$, $a\in Func[C_i]\cup Attr[C_i]\cup Act[C_i]$ 且 $a$ 没在 $rename_{c_i}$ 中出现

(3)对任意 $a\in Attr[C_i]$, $b\in Act[C]-m_i(Act[C_i])$, 有 $[b]m_i(a)=m_i(a)$

这里,条件 3 保证相应理论态射是保封装性的.所以,继承结构的对象语义描述的对象标记为:

$$OSig[C]=Sig[data]+m_1(OSig[C_1])+\cdots+m_n(OSig[C_n])+(\varphi,\varphi,M_{attr},M_{act})$$

其中

$$M_{attr}=\{f:r\mid f:r\in O\};$$

$$M_{act}=\{Get\_f:r\mid f:r\in O\}\cup\{f:s_1\times\cdots\times s_n\mid f(t_1\ s_1,\cdots,t_n\ s_n)\in E, t_i=\textbf{in},\textbf{out}\}$$

继承结构的对象语义描述的公理集为:

$$Axioms[C]=Axioms[data]\cup m_1(Axioms[C_1])\cup\cdots\cup m_n(Axioms[C_n])$$
$$\cup M_f[LSC]\cup M_f[LDC]\cup M_f[GSC]\cup M_{ensure}\cup Enc$$

其中　$Enc=\{[b]m_i(a)=m_i(a)\mid a\in Attr[C_i], b\in Act[C]-m_i(Act[C_i]), 1\leqslant i\leqslant n\}$

这样,我们就将继承结构的描述指称为对象语义描述.

## 3 聚　合

面向对象的优点之一是可以对所研究的问题域(论域)的各个不同部分分别建模,然后根据它们之间的相互关系将它们组合在一起,这样的过程可以一直迭代下去.这样,我们可以对同一实体的不同部分分别建模,然后将它们组合在一起构成系统的一个新成分.

### 3.1 语法描述

聚合提供了由已有对象的描述构造复杂对象的描述的手段.为描述聚合,必须为构成聚合对象的各成分对象提供名字,此外还必须描述各成分对象之间以及成分对象和聚合对象之间的关系(结构和行为的关系).一般地,一个聚合对象类增加如下描述:

**Assume** $data, C_1, \cdots, C_m$

**Observation** $a_1:C_{c_1};\cdots;a_n:C_{c_n}$;

**Communication** $Com$

其中 $C_{c_j}\in\{C_1,\cdots,C_m\}$; $Com$ 中包含形如 $a_i.\ act_{i_k}\ sync\ a_j.\ act_{j_l}$ 的通信子句,这里 $1\leqslant i,j\leqslant n$, $act_{i_k}\in Act[C_{c_i}]$, $act_{j_l}\in Act[C_{c_j}]$, $sync\in\{$**sync with**, **trigger**$\}$.

在我们的对象描述中,聚合对象的成分对象是作为聚合对象的属性出现在聚合对象的观察子句描述中,也就是说,我们的对象描述中的属性可以是数据属性也可以是对象属性.考虑到对象的动态聚合,我们假设对象的属性值可以是 $void$,即相应的对象还没有生成.

聚合对象的结构可以是不变的,即聚合对象一经生成,其拓扑结构将不会改变,这一点

可以在局部静态约束子句中加以描述,即静态聚合可以通过关于对象标识和常数的断言来描述.然而某些系统(或对象)的拓扑结构是随时间而改变的.

聚合对象将保有其各成分对象的特性,成分对象的属性只能通过成分对象所提供的动作加以修改;成分对象之间以及成分对象和聚合对象之间的相互作用必须通过相应的同步通信来描述,这一点可以在通信子句和局部动态约束子句中描述.通信子句描述了各成分对象之间的共享关系(通过动作的共享,两对象共享另一对象则意味着共享该对象的所有动作);而在局部动态约束子句则对各成分对象的复合行为作进一步的限制,序化其行为.

### 3.2 语义描述

我们知道,构造复杂对象的形式基础是余极限.[6]直观上讲,给定一个图表,余极限操作给出最小的包含同样信息的对象.也就是说,聚合对象中包含了其所有成分对象的信息以及将这些成分对象有机地组合在一起所必需的信息.这里我们考虑一简单的情形——外推,来讨论对象的聚合.事实上,余极限可以通过实施一系列外推而得.

给定3个对象语义描述(理论范集)$T$,$T_1$和$T_2$以及理论态射$m_i:T\to T_i$,$i=1,2$,则我们可得到一个对象语义描述(理论范集)$T'$以及理论态射$n_i:T_i\to T'$,$i=1,2$,满足$n_1\circ m_1=n_2\circ m_2$.

$T$的对象标记由$T_1$和$T_2$的对象标记而来.通过理论态射$m_1$和$m_2$,$T_1$和$T_2$的对象标记的部分成分在$T$中一致化,即$T_1$和$T_2$的对象标记中需共享的属性,反映为$T'$的对象标记中的同一属性;$T_1$和$T_2$中需同步的动作反映为$T'$中的同一动作.$T'$的对象标记中还包含$T_1$和$T_2$的非共享属性和非同步动作的不相交的并.$T'$的公理包含$T_1$和$T_2$的公理(经过适当的符号变换)和保封装性公理.

采用通常面向对象方法中的"点表示",假设我们分别给$T$,$T_1$和$T_2$以名字$name$,$name_1$和$name_2$,则$T'$的对象标记中的符号为:

(1)若$f\in T$,则$name.f\in T'$,否则

(2)若$f\in T_i$,则$name_i.f\in T'$,$i=1,2$

这里我们使用了不严格的$\in$.

因此,成分对象的对象语义描述为$a_i.M[C_{c_i}]$,$1\leqslant i\leqslant n$,即在$M[C_{c_i}]$的每个属性符号和动作符号及公式前附加前缀$a_i.$,聚合对象描述中的引用类子句、观察子句和通信子句定义了各成分对象的对象语义描述到聚合对象对象语义描述的理论态射$m_i:a_i.M[C_{c_i}]\to M[C]$,$1\leqslant i\leqslant n$满足:

(1)$m_i(a)=a$,$a\in Func[C_{c_i}]$

(2)$m_i(a_i.b)=a_i.b$,$b\in Attr[C_{c_i}]$

(3)对$b\in Act[C_{c_i}]$

(1)若$a_i.b$ $sync$ $a_j.c\in Com$或$a_j.c$ $sync$ $a_i.b\in Com$且$a_i\neq void$,$a_j\neq void$,则$m_i(a_i.b)=C_acti\ j$,否则

(2)$m_i(a_i.b)=a_i.b$

(4)对任意$a\in a_i.\ Attr[C_{c_i}]$,$b\notin m_i(a_i.\ Act[C_{c_i}])$,有$[b]m_i(a)=_{m_i}(a)$

这里,条件4保证相应理论态射是保封装性的.另外,我们对问题作了一些简化,即通信是1—1的,同时没考虑动作的参数.对于$n$—$n$的通信,情况较复杂,这里我们就不再讨论,

其主要思想是将所有要发生同步的动作取同一名字.

所以,聚合结构的对象语义描述的对象标记为:

$$OSig[C]=Sig[data]+m_1(a_1.\ OSig[C_{c_1}])+\cdots+m_n(a_n.\ OSig[C_{c_n}])$$

聚合结构的对象语义描述的公理集为:

$$Axioms[C]=Axioms[data]\cup m_1(a_1.\ Axioms[C_{c_1}])\cup\cdots\cup m_n(a_n.\ Axioms[C_{c_n}])$$
$$\cup M_f[LSC]\cup M_f[LDC]\cup M_f[GSC]\cup M_{ensure}\cup Enc$$

其中 $Enc=\{[b]m_i(a)=m_i(a)\mid a\in a_i.\ Attr[C_{c_i}],b\notin m_i(a_i.\ Act[C_{c_i}]),1\leqslant i\leqslant n\}$

由于我们的语义模型仅允许静态对象聚合,因此对于动态对象聚合,我们必须做一些技术上的处理:对每种可能的聚合,相应聚合对象的实体为所有成分对象的实体的复合,从语义上讲,相应聚合对象的对象语义描述为所有成分对象的对象语义描述经适当换名后的复合(余极限);在聚合对象的生命期中,若复合改变,则删除"老的"聚合对象,而以对应于当前复合的"新的"聚合对象代替之(保持对象标识不变).

我们看到,聚合仅仅是将一些对象通过适当的界面组合在一起,它并不增加更多的状态信息.聚合对象的状态是其成分对象状态的组合;而继承则是在给定对象描述中增加新的状态信息(增加属性,对行为加以限制等).因此,聚合对象的实现只需要其成分对象的实现再加上各成分对象动作之间的同步,而继承所描述的对象的实现则需附加的实现细节.继承和聚合可以组合使用.不难发现,多重继承可以由单继承得到.复杂对象的描述可以通过首先聚合我们所需要继承的信息,然后通过继承结构加以描述,即复杂对象的复合可以分为 2 步:成分的聚合和细节的增加.

### 3.3　对象精化描述

在设计时,人们通常不可能一下子分析清楚所有的细节.面向对象方法不仅提供了由对象构造系统(水平结构化)的方法,还提供了由已有对象实现新对象(垂直结构化,新对象称为抽象对象,而用来实现抽象对象的对象称为具体对象)的方法.这样,在设计的每个抽象层次,我们都可以认为(对象)系统是由一系列对象聚合而成.在系统设计的低层,我们可以进一步精化这些对象,即用一些更简单的对象实现之.这样的过程可以反复迭代,直到所有的对象都可以直接由机器实现.

抽象对象可以认为是相应具体对象的聚合,并对其行为进一步限制.然而,有一点需要注意,作为抽象对象的成分的具体对象是不可以共享的,它的存在依赖于抽象对象,并响应抽象对象的动作.在我们的对象描述中,将其描述为抽象对象的私有对象属性.抽象对象的属性和动作将由相应具体对象的属性和动作来实现.这里,我们将抽象对象的属性定义为具体对象的属性的导出属性,并在局部静态约束子句中描述它们之间的依赖关系;而抽象对象的动作则定义为具体对象动作所组成的事务,并在事务子句中描述.关于事务描述语言及其语义,这里我们不再讨论.

一般地,抽象对象可增加如下描述:

**Assume** $data,C_1,\cdots,C_m$

**Private Attrivute** $a_1:C_{c_1};\cdots;a_n:C_{c_n};$

**Internal Action** $E$

**Communication** *Com*

**Transaction** *trans*

有关精化的语义指称与聚合的情形类似.但是在抽象对象的对象语义描述的公理集部分需加入有关动作封装性以及动作作用次序的公理.这里我们以一简单情形为例,即考虑如下的事务子句: $act$ **implemented as** $act_1;act_2;\cdots;act_n$

其中 $act$ 是抽象对象的动作,$act_i$ 是具体对象的动作.对这样的事务子句需增加如下的公理:

$$act_i \rightarrow act, 1 \leqslant i \leqslant n$$

$$act \rightarrow \mathbf{F}\ act_1, act_1 \rightarrow \mathbf{F}\ act_2, \cdots, act_{n-1} \rightarrow \mathbf{F}\ act_n$$

## 4 对象类

在前几节,我们主要讨论了对象的模块特性.我们将对象类描述指称为对象语义描述,即我们定义了部分函数 $M:CLASS \rightarrow \mathbf{Th}$.这里,我们将讨论对象的类型特征.

### 4.1 对象标识

在面向对象程序设计中,对象通常是通过对象类来描述,也就是说,对象类在某种程度上也可看成一类型,它描述了一组具有相似结构和行为的对象.然而与数据不同,对象具有区别于其它对象的唯一标识,若将这样的标识看成数据,则对象类描述不仅给出该类对象所共有的结构和行为,还"定义"了相应的对象标识的类型,如

**ClassSpec** *PERSON*

**Identifier Sorts** *person*

**functions** $p$: *person*; *new*: *person* $\rightarrow$ *person*

**End** *PERSON*

若将对象标识类型 *person* 看成一般的数据类型,并使用初始代数语义,则 *person* 类型元素为: $p, new(p), new(new(p)), \cdots$

然而上述的对象类描述只阐述了一件事,即存在无穷个不同的对象实例,这些对象实例的对象标识是 *person* 类型的元素.

为简明起见,我们在对象描述中略去对象标识类型的描述,而以相应对象类的名字代替之.也就是说,上面的语义函数同时还描述了对象标识类型,即我们有部分函数:

$$\mathbf{IdSort}:\mathbf{Th} \rightarrow \mathbf{Sort}$$

若 $sp = M[C]$,则 $\mathbf{IdSort}(sp) = C$,否则没定义.

### 4.2 子类和抽象

子类是一种抽象(通常又称为特化),它使得 2 个对象类的实例之间有一种继承关系.*ODL* 中,类不仅可以作为模块使用,而且还定义了对象的类型,相应地,*ODL* 的子类关系也包括 2 方面的内容:即子类继承了父类的所有内部结构和行为,从而支持代码共享;且子类的对象可以出现在父类对象的任一可出现的位置上.一个子类可以有多个父类,子类自动继承父类的全部属性和动作,同时还可以添加属于自己的新的属性和动作.当然子类也可以细化父类的属性和动作,多态性、动作重载和动态联编是面向对象的重要特征.但是我们要求这些细化或修改必须要保证子类和父类语义的一致性.如果考虑到子类关系,则必须对上面

定义的聚合对象的语义作一些修改,即以对象属性 $a_i$ 所指的对象创建时所使用的类 $IC_{c_i}$(即 $a_i$ 所指的对象是作为 $IC_{c_i}$ 的对象实例而创建的)代替指称描述中的 $C_{c_i}$,当然 $IC_{c_i}$ 必须是 $C_{c_i}$ 的子类.

抽象(通常又称为概括 Generalization)与特化类似,只不过是其反面,即通过特化我们可定义对象类的不同子类而抽象则为一些对象类定义共同的父类.

### 4.3 子类和继承

面向对象方法中,子类和继承是 2 个相似但又不尽相同的概念.要理解这两者的差异,需牢记对象类兼有模块和类型的特征.

继承和子类都是求精机制.继承的主要目的是支持复用,因而主要是在对象类的模块意义下.当然,我们描述对象,通常是将它作为某一类的实例来描述的.因此,如果我们在一个类 $C$ 的描述中显式指出它继承了另一类 $C'$ 的结构和行为(当然可以换名或作一些限制),我们是指类 $C$ 的实例继承类 $C'$ 相应实例的结构和行为,还是在模块意义下的,不过通常人们还是称对象类 $C$ 继承对象类 $C'$.更确切点,从语义上讲,$C$(对象)的模型均是 $C'$(相应对象)的模型(当然要作一些换名或作一些限制).而子类除了刻划子类对象实例继承父类相应对象实例的结构和行为外,还有更多的含意,即子类对象实例可以代替父类相应对象实例,在所有父类对象实例可以出现的场合子类相应对象实例也可以出现.因此,虽然子类是一种继承,但它有更多的含意是在类型意义下的,它刻画了所有子类对象实例同时也是其父类的对象实例.

许多面向对象方法将子类与继承等同看待,我们认为这样做限制了继承作为复用手段的能力.如桌子类对象具有属性 *name*,*birthdate* 和动作 *move*;计算机类对象也有这些属性和动作,当然还有其它属性如 *OS* 和动作如 *run* 等.显然它们并不构成父子关系,但没有理由说计算机类对象不可以复用桌子类对象的代码.

我们还可以从语义上区分这 2 个概念.

$sp, sp' \in \mathbf{Th}$,$sp$ 继承 $sp'$ 当且仅当存在一理论态射

$$\sigma: OSig(sp') \to OSig(sp) \text{ 且 } Mod(sp)|_\sigma \subseteq \mathbf{Mod}(sp')$$

$sp, sp' \in \mathbf{Th}$,$sp$ 是 $sp'$ 的子类当且仅当存在一理论态射

$$\sigma: OSig(sp') \to OSig(sp) \text{ 且 } \mathbf{Mod}(sp)|_\sigma \subseteq \mathbf{Mod}(sp'), \mathbf{IdSort}(sp) \subseteq \mathbf{IdSort}(sp').$$

也可以将这里的理论态射定义为包含态射,即 $OSig(sp') \subseteq OSig(sp)$

继承主要是模块的求精,所以继承关系是基于对象类所允许的模型的子集关系(包含适当的换名).由于上面的理论态射不需要是满射,所继承类的对象标记可以比其祖先类的对象标记丰富.子类关系主要是子类的对象标识集和父类的对象标识集的子集关系.由于子类描述通常是在父类描述之上增加一些属性、动作等细化,所以子类关系也是对象标记和所允许的模型的子集关系.

## 5 结束语

作为软件构造的基本单元,对象由数据及其所提供的服务组成.对象的内部数据或状态是用一组属性来表示的,但是它对外界是不可见的,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.对象的理解也是通过对象对外的语法接口和语义描述来完成的,使用对象无需了解对象的内部细节.对象的封装性与传统的模块设计原则一样,具有很强的信息隐藏特性.

本文介绍了对象规范描述语言ODL,ODL用于描述对象的基本特性,它包含对象的语法界面的描述和对象语义特征的描述.对象语义特征的描述包含单个操作(观察和动作)的操作语义描述、对象静态结构约束、对象动态行为约束、同步通信关系约束以及事务约束等.操作语义描述采用公理化的方法,包含动作的时序前条件、状态前条件和动作的后条件等3部分.对象静态结构约束包含规定对象合法状态的局部静态约束和规定合法对象的全局静态约束.对象的同步通信关系约束是对对象聚合而言的,规定了成分对象的复合行为.对象的事务约束是对对象精化而言的.

同时,我们给出ODL的主要结构的SOP语义的指称描述,包含简单对象的指称描述、对象继承结构的指称描述、对象聚合结构的指称描述、对象精化的指称描述等.简单对象的语义由对象语义描述给出,继承和聚合等对象构造结构的语义可以通过对象语义描述范畴上的适当操作给出.并从语义上区分继承和子类(继承)这2个相似而又不尽相同的概念.

## 参考文献


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

2 黄涛.对象语义理论的研究[博士论文].中国科技大学,1994.

3 Goguen J A, Burstall R M. Introducing insitutions. In: Clarke E, Kozen D eds., Proc. Logic of Programming, LNCS164, 1984.

4 Ehrig H, Mahr B. Fundamentals of algebraic specifications 1: equations and initial semantics. Springer—Verlag, 1985.

5 黄涛,冯玉琳,李京.对象形式语义模型.软件学报,1995,6(增刊):207～212.

6 Michael Barr, Charles Wells. Category theory for computing science. Prentice Hall, 1990.


# OBJECT DESCRIPTION LANGUAGE AND ITS DENOTATIONAL SEMANTICS


Huang Tao Feng Yulin Ni Bin

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

Li Jing

(*Department of Computer Science University of Science and Technology of China Hefei* 230027)



**Abstract** Objects are considered as basic components of software system in object oriented software construction. This paper presents an ODL (object description language) and the denotational semantics of its main constructs. The formal counterparts of objects are given by theory presentation in SOP calculus. Attributes (structure) and operations (behavior) are integrated in coherent logical units around which the notion of encapsulation is formalized. Aggregation, inheritance etc. are formalized as specification constructs which allow people to assemble large specification from existed ones/ extend a given specification. Further, the notions of inheritance and subclass are clarified.



**Key words** Object, denotational semantics, calculus, object semantic description.