面向对象软件的形式规格说明技术

全炳哲 金淳兆

（吉林大学计算机科学系 长春 130023）

摘 要 本文介绍四种面向对象形式规格语言。Object-Z 是 Z 语言的一种扩充，可用于面向对象软件需求规格的形式说明。为研究软件维护和逆向工程，提出了 Z^{++}，是 Z 的另一种扩充，其中引入了过程式描述机制。COLD-K 是基于代数规格说明技术的面向对象软件设计语言，是一种核心语言，可设计面向用户的形式规格语言。JOOSL 是基于 COLD-K 和 Z 语言的一种面向对象设计语言，可用于软件自动化的研究。

关键词 软件工程，面向对象软件，形式规格说明，形式规格语言。

一、引言

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。另一方面，自七十年化初提出面向对象概念以来，由于它能很好地支持软件开发过程的各阶段，已成为很重要的软件开发风范。因此，很有必要研究面向对象软件的形式规格描述方法和面向对象软件的自动化问题。Z 语言是一个较著名的软件形式规格描述语言，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[2,3,4]是 Z 语言的一种扩充，其中引入了类及其继承的描述机制，旨在描述面向对象系统。Z^{++} 语言[5,6,7]是 Z 语言的另一种扩充，支持面向对象软件的描述。OBJ3 语言[8]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，反映了面向对象的某些思想。COLD-K 语言[9,10,11]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，可用于描述概要设计和详细设计。JOOSL[12]是我们自行设计的面向对象形式规格语言，作为研究面向对象软件问题的基础。本文将分别介绍 Object-Z、Z^{++}、COLD-K 和 JOOSL 语言。

二、Object-Z

Object-Z 言的基础是 Z 语言，下面简要介绍 Z 言的结构，详细内容参见文[13]。Z 语言是基于集合论及一阶谓词逻辑的软件形式规格语言，Z 规格说明中大量使用了有关集合论和逻辑上的数学符号。规格说明的基本单位是 schema。一个系统由若干个 schema 组成，schema 可以用来描述类型、系统状态、操作等，schema 结构如下：

schema 名字
declaration（说明部分）
predication（谓词部分）

一个 schema 在语义上，将该 schema 名字与其内容等价起来，一个 schema 的名字可以出现在另一个与之相关的 schema 的说明或谓词部分，即它可被引用，在各 schema 之间，有可能产生一些关系，例如，设 S 与 T 分别是 schema 名字，则：

S∧T：为一 schema，其说明部分取自 S 与 T 的说明部分；其谓词部分为 S 与 T 的谓词部分的合取。

S∨T：为一 schema，其说明部分取自 S 与 T 的说明部分；其谓词部分为 S 与 T 的谓词部分的析取。

[student]

class
handin, nohandin: P student
handin ∩ nohandin = {}
(handin ∪ nohandin) ≤ max

图 1 班级的 schema

Class Name
visibility list
inherited classes
constants
variables
class invariant
initial state
operations

图 2 类的描述形式

*）本文得到国家自然科学基金和"863"计划项目的支持

假设有一班级，人数不超过 max，有交作业的和没交作业的，其 schema 如图 1 所示。在此 schema 中，class 为 schema 的名字，说明部分的 handin 与 nohandin 分别是交作业的和未交作业的同学的集合；谓词部分给出该 schema 的限制条件：前一条表示这两部分的学生的集合不相交；后一条表示班级的总人数不超过 max 这一限制。[studennt]表示假定 student 集合(或类型)已有定义。∩和∪分别表示集合的交和并运算，{}表示空集合。这个 schema 描述了班级状态。

在 Object-Z 语言中通过类的概念来封装对象的状态和相关的操作。语法上，用带有名字的框来表示，其形式如图 2 所示。其中，

Class Name：是欲定义的类名称。

visibility list：指明类中哪些成分是外部可见的，用 Z 语言的影射符号来说明。若类中没有这部分说明，则表明类中的所有成分都是外部可见的。

inherited calsses：指明被继承的所有祖先类的名字。

constants 和 variables：指类的属性。constants 是不变的量，即不可用类中操作改变其值，但当实例化该类时，这部分的值可以不同。常量和变量无论何时，都要满足类不变式(class invariant)和继承下来的不变式。

initial state：是一个 schema，用于描述所允许的初始实例，通常初始值不必由该 schema 唯一确定。

operations：是一个 schema，作用于类的属性以及可能的外部输入或输出变量上，通过这些输入、输出变量，类可以与环境进行联系。

对于类的语义模型，可以用事件历程(event history)来给出。一个事件由一个操作的名字和与它相对应的前置事例和后置事例(pre-and post-instance)组成；一个类的历程是一个事件序列。其中，第一个事件是初始事例；对于任意两个连续事件，前一个事件的后置事例与后一个事件的前置事例相同。这样一个类可以看作可能事件历程的集合，其形式语义参见[3]。下面给出用 Object-Z 描述的图形类的例子。图形包括圆和多边形，多边形包括平行四边形，平行四边形又包括菱形和矩形，而正方形分别是这两者的特例。这里我们仅给出图形(shape)和圆的形式描述，并假定已有全程类型 Vector(向量)、有关标准的向量函数和关系：

rotate：Vector × R → vector　　定义向量旋转，其中 R 是实数类型。

_+_：Vector × Vector → Vector 定义向量相加

由以上说明，可描述基本图形类，如图 3 所示。

由于 Shape 类的描述中没有外部可见部分的说明，所以它的所有部分外部都可见。另外，Shape 类没有祖先类，也没有常量，但有两个变量，一个是类型为 Vector 的 *position* 变量，表示相对于某一个座标原点的图形位置，另一个是类型为 R 的 perim 变量，表示图形的周长。类不变式是说图形的周长一定大于零。图形类提供 Move 和 Rotate 操作，用于移动和旋转图形。Δ 是一个变量列表，这些变量可能被该操作修改；若某变量不在 Δ 列表中，则本操作不修改此变量。在 Z 中，'?' 表示来自环境的一个输入量，因此 Move 操作需要来自当前环境的一个向量值 v，从实现角度看，可理解为它有一个参数。撇号算子是表示下一个状态下的值，即 position′ 表示执行 Move 或 Rotate 操作后的变量 *position* 的值。

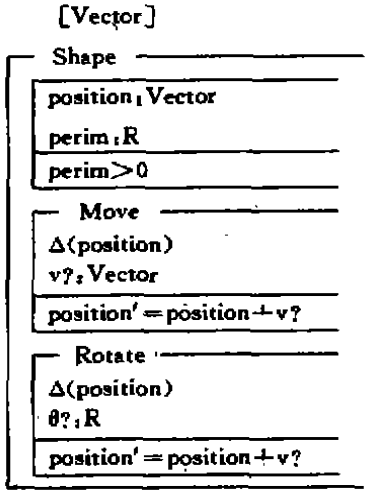

图 3 图形类的描述

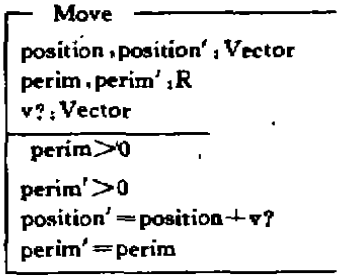

图 4　Move 操作的 Z 语言描述

当用 Z 语言描述 Move 操作时，其结构如图 4 所示。通过继承图形类可描述其它更具体的类，图 5

是圆类的 Object-Z 描述。圆类继承图形类，其中分别用 centre(中心)和 circum(圆周长)变量名重命名了 position 和 perim 变量。在圆类中引入了附加的变量 radius(半径)和类的不变式，此不变式规定了圆周长和半径之间的关系。Move 操作和 Rotate 操作原封不动地被继承。由于 radius 不在 Move 操作和 Rotate 操作的 Δ 列表上，所以这些操作不改变 radius 变量。我们可把圆类展开成与此等价的描述，以说明继承的涵义，展开结果如图 6 所示。

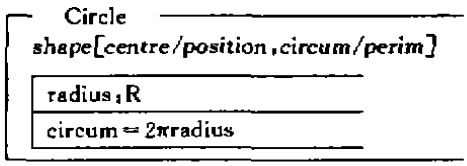

图 5 圆类的描述

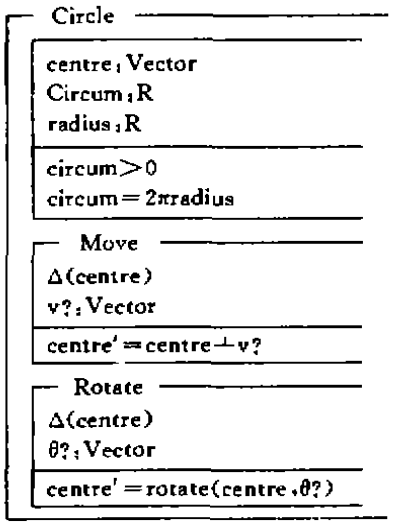

图 6 圆类的展开描述

三、Z++

Z++语言基本上采用 Z 语言的记法，但增加了类的描述机制。类规格说明主要由五个方面组成：(1)定义全称类型；(2)定义对象类；(3)定义系统状态，其中包括各操作中用到的变量及它们之间的不变式；(4)定义初始状态，是用状态变量上的约束(constraint)来规定；(5)定义操作，用状态变换描述操作。Z++语言中类的说明形式如下：

```
Object_Class::=
  CLASS Identifier TypeParameters
  [EXTENDS Imported]
  [TYPES Types]
  [OWNS Locals]
  [RETURNS Optypes]
  [OPERATIONS Optypes]
  [INVARIANT Predicate]
  [ACTIONS Acts]
  [CONSTRAINTS Constraints]
  [HISTORY History]
  END CLASS
```

TypeParameters::=["["Parlist"]"]

Parlist::=Identifier[,Parlist]|Identifier<<Identifier[,Parlist]

Imported::=Idlist

Types::=Type_Declarations

Locals::=Identifier:Type;Locals|Identifier:Type

Optypes::=Identifier:Idlist→Idlist;Optypes|Identifier:Idlist→Idlist

Acts::=[Expression &]Identifier Idlist⇒Code;Acts|[Expression &]Identifier Idlist⇒Code

Constraints::=Equation;Constraints|Equation

History::=$Formula_{LTL}$

TypeParameters 是类属类型(generic type)表，可为空。当类属类型 X 必须是某类 A 的子类时，用 A<<X 表示。EXTENDS 是继承的描述，用本子句可描述类之间的继承关系。Types 用于定义一些类型，定义对象中的局部变量，其中变量的定义方法采用 Z 中的定义方法，Locals 是对象属性的定义，即实例变量的定义。RETURNS 列表用于定义反映对象状态的操作类型，操作类型是指操作的定义域和值域的定义，值得注意的是这些操作无副作用，而 OPERATIONS 列表定义其它操作的类型。INVARIANT 是类的一种不变式，用谓词描述对象内部状态的性质或它们之间的关系，操作的执行前后对象内部状态必须满足此不变式。ACTIONS 列表用于定义操作的语义。例如，对于一个队列 q，READ x 操作可描述为：

READ x⇒q' = tail q ∧ x = head q

在 Acts 的定义中，要求把输入参数放在输出参数之前，Code 可以是 Z 谓词，也可以是 UNIFORM 语言[11]中的过程式代码。UNIFORM 是一个通用的中间语言，其特性来自 FORTRAN 语言和 COBOL 语言，但其程序结构更类似于 COBOL 程序。CONSTRAINTS 列表用于描述由类生成的对象初始状态。HISTORY 谓词规定所允许的对象执行顺序，其中采用了线性时态逻辑(Linear Temporal Logic)公式。

四、COLD-K

COLD(Common Object-oriented Language for Design)是一个形式化的软件设计语言，其目标是用于描述软件设计，它分为核心语言和面向用户的语

言。本文主要介绍核心语言，称为 COLD-K。COLD-K 的基本描述单位是类，可认为它是一个具有一些状态和一个初始状态的抽象机，通过过程(procedure)操作可改变其状态。

类的定义方式如图 7 所示。关键字 EXPORT 规定外部可见的部分，包括 sorts、谓词、函数和过程。signature(基调)中的每个名都应有相应的定义，有定义但没列在 EXPORT 表列上的成分称为隐藏的成分。类有一些状态成分(state components)，可用一组名来标识状态成分，称为状态基调(state aignature)，可分为如下三种：

(1)Sort 名：在每个状态下，sort 名表示一个 sort，是对象集合。

(2)Predicate 名：在每个状态下，Predicate 名表示一个谓词，是定义在给定状态下的通常的数学谓词。谓词对应于一般程序设计语言中的布尔型函数或布尔型变量，区别是，谓词是真正的数学谓词，没有返回值，对于给定参数它们要么真要么假，不存在无定义的情况。

(3)Function 名：在每个状态下，function 名表示一个函数，是给定状态下的 sort 到 sort 的部分函数。函数对应于一般程序设计语言中的函数子程序或某变量，区别是，COLD 中的函数没有副作用。

```
EXPORT
  signature
FROM
CLASS
  definition1
    ...
  definitionn
END
```

图 7 类定义

从类的使用者角度看，类不关心对象状态是用某种变量表示的，还是用一段代码表示的，所以在 COLD 的类中不引入状态变量，而是统一地用谓词或函数来表示状态。类的定义中，可采用显式定义方式和隐式方式，显式定义方式是指直接给出某成分的性质，这里采用扩充的一阶谓词和归纳定义方法，而隐式定义方式是指通过其它的定义规定某成分的性质，其中采用代数规格说明方法。下面是自然数的一种 COLD-K 描述。

```
EXPORT
  SORT Nat,
  FUNC zero:    →Nat,
  FUNC succ:Nat→Nat,
  FUNC pred:Nat→Nat,
  PRED less:Nat # Nat
FROM
CLASS
  SORT Nat,
  FUNC zero:    →Nat,
  FUNC succ:Nat→Nat,
  AXIOM
  {NAT1}zero!;
  {NAT2}FORALL n:Nat (succ(n)!)
  AXIOM FORALL m:Nat,n:Nat(
  {NAT3}NOT succ(n)=zero;
  {NAT4}succ(m)=succ(n)⇒m=n)
PRED is_gen:Nat
IND is_gen(zero);
  FORALL n:Nat(is_gen(n)
                ⇒is_gen * succ(n)))
AXIOM
{NAT5}FORALL n:Nat(is_gen(n))
FUNC pred:Nat→Nat
IND FORALL n:Nat
    (Pred)succ(n))=n)
PRED less:Nat # Nat
IND FORALL m:Nat,n:Nat
  (less(m,succ(m));
    less(m,n)⇒less(m,succ(n)))
PRED is_pred:Nat # Nat
IND FORALL n:Nat
  (is_pred(succ(u),n))
END
```

感叹号(!)是 COLD 语言定义的一个谓词，表示有定义；分号(;)是一种合取联结词，只是其优先级低于古典逻辑中的合取联结词；AXIOM 表示公理，不给证明地引入类成分的性质；IND 表示归纳定义。另外，is_gen 和 is_pred 是对外隐蔽的操作。

类的另一个成分是过程操作。只有通过它，才可以改变对象状态。所谓改变对象状态实际上意味着改变函数或谓词。过程只可修改不依赖于其它操作的谓词或函数，这种谓词和函数称为变量型的。为描述状态转换，COLD-K 提供：(1)两个特殊的公式，它们源于动态逻辑。一个是作用 always 算子的公式，[X]A，意味着求表达式 X 值后，所有状态均使 A 成立，另一个是作用 sometimes 算子的公式，<X>A，意味着求表达式 X 值后，将有一个状态使 A 成立；(2)三个语言结构。一个是说明结构，例如 LET x：T，它引入类型为 T 的一个对象名；另一个是约束结构，如 x：=Y，它表示 x 代表 Y 的值，一旦约束一个名后，不允许再对它进行其它的约束，因而约束可认为是一种缩写结构，而不是赋值语句；第三个是块结构，BEGIN X END，它用于规定对象名的作用范围，在断言或表达式 X 中说明的对象名的作用范围不超过此范围；(3)表示前一状态的算子 PREV。PREV A 表示断言 A 在前一状态下成立，PREV X 表示在前一状态下的表达式 X 的结果。利用这些机制和代数规格说明技术可描述过程操作。

在 COLD-K 中，除提供如上较高层次的描述方

法外，还提供了算法化的描述机制。这些都称为表达式，分为如下七种：(1)SKIP 表达式：在任何状态下都停止，且不改变当前状态，产生空对象序列。它有时记作()，可用于调用无参操作，如 zero()；(2)条件表达式 A?：它在某状态 S 下停止，当且仅当 A 在 S 下成立。此表达式不改变当前状态，且产生空对象序列；(3)复合表达式 X；Y：把状态 X 转换成 T，并产生对象 $x_1,\cdots x_n$，当且仅当存在状态 U，表达式 X 把状态 S 转换成 U，并产生对象 $x_1,\cdots,x_k$，同时表达式 Y 把状态 U 转换成 T，并产生对象 $x_{k+1},\cdots,x_n$。(4)选择表达式 X|Y：把状态 S 转换成 T，并产生对象 $x_1,\cdots,x_n$，当且仅当 X 把状态 S 转换成 T，并产生对象 $x_1,\cdots,x_n$，或 Y 把状态 S 转换成 T，并产生对象 $x_1,\cdots,x_n$；(5)循环表达式 X^*：把状态 S 转换成 T，当且仅当存在一个状态序列 $s_1,\cdots,s_n$，使得 $s_0=s,s_n=t$ 且 X 把状态 s_i 转换成 $s_{i+1}(i=0,\cdots,n-1)$。它产生空对象序列；(6)修改表达式 MOD $v_1,\cdots,v_n$ END：把状态 S 转换成 T，当且仅当从状态 S 只修改 $v_1,\cdots,v_n$ 可得到状态对象 T；(7)选取表达式 SOME$x_1:T_1,\cdots,x_n:T_n$.A：它在状态 S 下停止，当且仅当存在 S 中的对象 $x_1,\cdots,x_n$ 使 A 成立。若它停止，则不改变当前状态，但产生满足 A 的(任意)对象 $x_1,\cdots,x_n$。

在表达式的组合中规定了优先级，从低到高依次是(1)"|"；(2)";"(3)","；(4)"[]","<>"；(5)"PREV"；(6)"?","*"；(7)"!"；(8) SOME，FORALL，EXIST 等。通过这些低级表达式，可描述较高级的控制表达式，例如，可把 IF A THEN X ELSE Y 看成(A?;X|NOT A?;Y)的简写；也用(A?;X)*;NOTA? 表示当型控制结构 WHILE A DOX。

除如上的特点外，COLD-K 还提供参数化描述和实例化机制，但未提供继承概念，因而未能完整支持面向对象方法。为解决继承的描述，文[11]中讨论了在 COLD-K 中引入继承机制的方法和语义。

五、JOOSL

JOOSL (Jilin Object-Oriented Specification Language)是以 COLD-K 语言为基础，吸收 Z 语言的描述方法和 Rcos 语言[15]语法格式的一种面向对象软件设计描述语言。从面向对象软件开发模型[16][17][18]角度看，不需要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。但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和软件设计结果。JOOSL 认为可用三个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。其中，用 JOOADL 描述前两个层次，用 JOODDL 描述第三个层次。

5.1 JOOADL

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，包括确定对象属性和服务；第三，要识别对象之间的关系，包括引用关系和继承关系。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应主要考虑系统中各对象及其关系，而不考虑实现方法。面向对象概要设计(OOPD)是 OOA 的延续和扩充。OOPD 的主要目标是从实现角度进一步分析类及它们之间的关系。首先，从实现角度进一步分析系统所涉及到的类，从实现角度引入附加类；其次，分析类之间的公共属性，构造抽象类；第三，要选择操作的算法特性，这就常常需要引入附加操作，这种操作属实现细节，应该作为类内部的私有操作。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类似，这也就反映了 OO 开发模型中任务和任务之间的重叠。在 OOA 和 OOPD 阶段，均可用 JOOADL 描述软件。

为提高软件高层描述的抽象度，在 JOOADL 中用操作统一地描述类的数据特性及所提供的服务。这就为实现者提供更多的灵活性，以选择不同的实现方案。操作分为函数操作、谓词操作和过程操作。函数操作用于观察对象的当前内部状态。函数没有副作用，它的一种特点是可以起类的实例变量的作用，但它属于实现细节。实际实现中，对象状态可用一些变元记录，也可用一段代码实现。谓词操作用于测试对象当前状态下的某种条件的成立与否。它们的作用类似于函数，但它无返回值，它们在某种状态下要么成立要么不成立。具体实现中，谓词可对应于布尔型函数或布尔型实例变量。过程操作作用于修改对象的当前状态，也可用于描述完成某种功能的操作。由于用函数和谓词描述对象状态，对象状态的改变就指修改函数的返回值或改变谓词的成立与否。在过程的说明中，需要描述过程的修改权限，即指定过程所修改的函数和谓词。

操作的功能基本上用一阶谓词逻辑的公式描述，但在下述几个方面有别于经典逻辑。首先，增加

不可交换的合取联结词，称为弱合取，记为分号(;)。例如，A;B，其中A或B可以是引用过程操作的子公式，其形式是：<过程引用>[TRUE]，意味着过程引用结束后，整个子公式取"真"值。由于过程引用引起状态变化，所以弱合取右边的公式的值依赖于新状态，因而弱合取不可交换。当A和B都不是过程的引用时，A;B等价于A∧B。其次，操作的描述中，允许使用归纳方法。归纳定义方法的形式如下：

IND for t_i : T_i(PB(t_1,t_2,…,t_n),PI(t,t_1,t_2,…,t_n))；

它是施归纳于t_i，其中PB和PI是谓词，分别对应于基始和归纳。这里T_i是良序类型，包括非负整数、非正整数和表。第三，谓词公式中可使用表示下一个状态的算子，用撇号(')表示。第四，原子公式只有关系表达式和操作的引用。

5.2 JOODDL

JOODDL中程序的基本单位仍然是类，其描述形式类似于JOOADL中类的形式。JOODDL和JOOADL的区别在于JOODDL中可直接说明类的实例变量，且用算法化的形式描述操作。变量类型可以是基本型或类，其中基本型包括整型、实型、字符型、布尔型和数组型。操作分为函数和过程，有返回值的操作称为函数，无返回值的操作叫做过程。操作的算法是用一些结构化的控制语句描述，JOODDL提供if语句、for循环语句、while循环语句、repeat循环语句、消息发送语句、输入/输出语句、return语句和compute语句。控制语句中条件表达式可以是受限谓词。compute语句是赋值语句的一种拓广，可描述算法特性较明显的一系列处理动作，其形如下：

compute(变元序列)' :' (受限谓词)

所谓受限谓词的含义是：首先，形式上和一阶谓词逻辑中的公式相同；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五个方面：(1)公式中出现的变元必须已定义；(2)受限谓词中，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；(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，但不允许出现变元之间的循环定义；(5)等式公式的左边必须是作用撇号算子的变元。

六、结束语

本文讨论了几种面向对象形式规格语言。Object-Z语言在Z上扩充了类的描述机制，但类的描述中除描述结构有其特殊规定之外，基本采用了Z的描述方法，可用于面向对象软件的需求规格说明。Z++语言也在Z语言上扩充了类的描述机制，这部分扩充很大部分类似于Object-Z语言的相应的部分，但在操作的描述中引入了UNIFORM语言的一些描述机制，因而Z++可认为是一种广谱语言。COLD-K语言是基于抽象数据类型的代数规格说明技术的一种形式规格语言，从这一点上看它类似于OBJ3语言，当然它提供了许多其它描述机制，包括归纳定义机制、低级控制结构的描述机制等。由于COLD-K不仅可描述软件的高层设计，而且可描述算法级的设计，所以它也是一种广谱语言。另一方面，由于COLD-K是作为核心语言提出的，所以用户不便直接使用，也未提供继承机制。为研究面向对象软件自动化问题，我们设计了JOOSL，基于它我们已实现了一种软件自动化系统JDAUTO/0，它分为概要设计到详细设计的自动转换工具PDAUTO和详细设计到C++程序的自动转换工具DDAUTO。

参考文献

[1]A. Hall, Using Z as a specification calculus for object-oriented system. LNCS 428, pp. 290-318

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

[3]D. Duke et al., Towards a semantics for object-Z, In VDM'90, VDM and Z!, Springer, 1990

[4]R. Duke et al., Aspects of object-oriented formal specification, ASWEC'90, Sydney, May 1990

[5]K. Lano at al., A specification-based approach to maintenance, Software Maintenance: R & P, vol. 3, no. 4, 1991, pp, 193-213

[6]J. P. Bowen, et al., Formal specifications in software maintenance: From code to Z++ and back again, Info. and Soft. Tech., vol. 35, no. 11/12, Nov./Dec. 1993, pp. 679-690

[7]K. Lano, Z++, an object-oriented extension to Z, In J. E. Nicholls (ed.) Z user workshop, Oxford, 1990

[8]J. A. Gougen et al., Introducing OBJ3, Report SRI-CSL-89-9, SRI Intl. Compt. Sci. Lab.

[9]H. B. M 'onkers, An introduction to COLD-K, Algebraic methods: Theory, tools and applications, Berlin: Springer-Verlag, 1990, pp. 139-200

[10]W. E. Baats et al., A formal specification of INGRES, Same to [9]

(下转第71页)

性分别是关系 $R_1, R_2, \cdots, R_n$ 的主码，而另一些（可空）为 R 特有的主属性，且 $A_1, A_2, \cdots, A_n$ 所对应的主属性联系方式为 n（此时不满足第 8 步的条件），则定义 R 中 $A_1, A_2, \cdots, A_n$ 的值域分别为 $R_1, R_2, \cdots, R_n$。亦即定义 $R_1, R_2, \cdots, R_n$ 为 R 的组成对象；

10. 重复第 7 至第 9 步，直至所有关系处理完毕，结束。

该转换过程主要分两阶段进行。第一阶段为第 1 至第 5 步，生成面向对象模式中的类型和类属联系，第二阶段为第 6 至第 10 步，生成类型间的分属联系。

需要说明一下标识符。面向对象模型支持系统定义的标识符，而关系模型只支持主码概念。如果进行数据库系统的转换，并没有什么问题，但在以面向对象模型作全局模型的联邦数据库系统中，只进行模式转换，数据库实例是不存放在全局模式中的，此时需要考虑标识符问题。关键是需要在面向对象模型中的标识符和关系模型的码值之间建立一种一一对应联系，如可定义面向对象模型中的标识符为类型信息加上相应关系中元组的主码值。

对关系视图我们亦未加以讨论，由于面向对象模型可以方便地定义视图机制[3]，故亦不难转换。

5. 一个例子

假设有一关系数据库模式包含以下关系：

教职工{职工号，主码/姓名/年龄}，
学生{学号，主码/姓名/年龄}，
优秀学生{学号，主码/简介}，
在职学生{职工号，主码 1/学号，主码 2/姓名/年龄}，
课程{课程号，主码/课程名/任课教师}，
学生成绩{学号，主属性/课程号，主属性/成绩}；

采用上一节介绍的转换过程进行转换后，得到如图 2 所示的面向对象模式。其中实践表示类属联系，虚线表示分属联系。另外需指出的是'学生'与'优秀学生'之间的类属关系需操作人员指定。

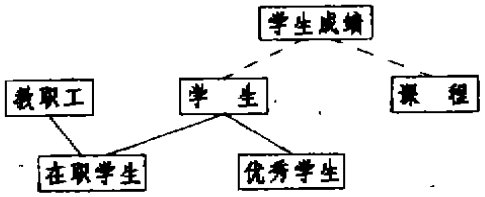

图 2 转换后所得的面向对象模式

由于本文只限于讨论模式转换，故未涉及实例转换的有关内容。

参考文献

[1]F. Salter, et al., Suitability of Data Models as Canonical Models for Federated Databases, SIGMOD RECORD, Vol. 20, No. 4, Dec., 1991

[2]B. Salzberg, Third Normal Form Made Easy, SIGMOD RECORD, Vol. 15, No. 4, Dec., 1986

[3]吴胜利，对象的视图与外模式，计算机研究与发展，1994 年第 8 期

[4]A. Motro, Superviews: Virtual Integration of Multiple Database, Transaction of Software Engineering, Vol. 13, No. 7, July 1987

（上接第 42 页）

[11]H. B. M. Jonkers, Inheritance in COLD, Algebraic methods I, Theory, tools and applications, Berlin: Springer-Verlag, 1991, pp. 277-302

[12]金娟哲，金淳兆，面向对象软件规格语言的设计，软件学报，待发表。

[13]Ian Hayes, Specfication case studies, Prentice-Hall Intl. (UK)Ltd, 1987

[14]C. Stanley-Smith, et al., UNIFORM, a language geared to system description and transformation, University of Limerick, Ireland, 1990

[15]金娟哲、余江、金淳兆，可重用构件及其描述语言，软件学报，vol. 5 no. 1, 1994, pp. 42-46

[16]B. Henderson-Sellers et al., The Object-Oriented System Life Cycle, CACM, 1990, 33(9): 142-159

[17]B. Henderson-Sellers et al., The O-O-O methodology for the object-oriented life cycle, ACM SIGSOFT Soft. Eng. Notes, 1993, 18(4): 54-60

[18]L. R. Hodge et al., A proposed object-oriented development methodology, Soft. Eng. J., March 1992 pp. 119-129