带 OCL 约束条件的类图到 Object-Z 规格说明的转换*)

缪淮扣　陈怡海
(上海大学计算机工程与科学学院　上海 200072)

摘　要　如何提高软件的可靠性是目前软件研究领域的一个热点。将形式化方法和主流的软件开发方法相结合是一个可行的方法。本文研究 UML 语言和 Object-Z 语言相结合的方法，为主流的软件开发人员所使用的图形化规格说明技术与形式方法提供的精确的分析和验证技术架起了一座桥梁。本文定义如何将带 OCL 约束条件的类图转换到 Object-Z 规格说明的方法。这样不仅可以通过支持 Object-Z 语言的工具来对 UML 语言描述的系统性质进行验证和确认，而且能够帮助规格说明人员方便地构造 Object-Z 规格说明。

关键词　UML，类图，OCL 约束，Object-Z 规格说明

Transformation UML Class Diagrams with OCL Constraints into Object-Z Formal Specification

MIAO Huai-Kou　CHEN Yi-Hai
(School of Computer Engineering and Science, Shanghai University, Shanghai 200072)

Abstract　How to improve the software reliability is the hot research topic in the field of software engineering. Integrating formal methods and mainstream software development methods is a viable approach. The integration of UML and Object-Z provides a bridge between graphical specification techniques used by mainstream software engineers, and the precise analysis and verification techniques provided by formal methods. In this paper, we define a translation from UML class diagram with OCL constraints into Object-Z. By this way, we can use the tools supporting Object-Z to validate and verify the system properties described by UML class diagrams, and also facilitate the specifier to construct the Object-Z specification.

Keywords　UML, Class diagram, OCL constraints, Object-Z specification

1　引言

软件系统在人类社会中起着日益重要的作用，软件故障将会带来灾难性的后果。随着软件系统的复杂性的不断增加，传统的软件开发方在保证软件没有隐错方面变得力不从心。根据最新的美国商务部的调查报告[1]：软件中的隐错每年造成美国大约 595 亿美元的经济损失，或者是相当于百分之 0.6 的国内生产总值。如何构造高质量的软件系统是软件工程研究领域面临的一大难题。形式方法是一种基于数学的软件开发方法，形式方法能对系统应该满足的性质进行描述，它具有精确性，一致性和无歧义性等特点，并且能够对规格说明进行分析和验证，因此被认为是提高软件质量的有效方法[2]。然而形式方法并没有在工业界得到广泛的应用。形式方法没有得到广泛应用的一个重要原因是构造形式规格说明比较困难，用户抱怨困难主要在于难以发现有用的抽象[3]。另一方面，UML[4] 语言作为一种通用的可视化建模语言，在工业界和学术界得到了广泛的应用，并有大量的工具支持，如 Rational Rose, Argo UML, Rhapsody 等。UML 语言使用各种不同的直觉上容易理解的图形来对系统静态属性和动态属性进行描述。已有越来越多的软件开发人员应用 UML 语言来对所开发的软件系统进行建模。但是，UML 缺乏形式化的语义和有效的推理机制，因此并不能保证所描述的规格说明的正确性。在过去的几年中，将 UML 和形式方法相结合是国内外研究人员的一个研究热点[5]，目的是结合两种方法的优点，弥补一种方法单独使用的不足。目前结合 UML 和 Z 语言家族的研究工作[6~12] 中，主要是定义转换规则，将 UML 类图(没有 OCL 约束)转换为 Z 形式规格说明，大多没有考虑模型中的约束信息，因此在转换的过程中，仅考虑了类、类属性、类操作、关联以及泛化的转换，这只能产生一个形式规格说明的轮廓，为了产生完整的形式规格说明，有的方法[9] 要求规格说明书写人员使用 Latex 格式的 Z 语言对类的不变式以及操作的前置条件、后置条件等约束条件进行描述，这会对不熟悉 Z 语言的软件设计人员带来困难。为了描述 UML 语言中的约束条件，对象管理组织提供对象约束语言 OCL[13,14]，OCL 语言可以对上述约束条件进行描述。在结合 UML 和 Object-Z[18] 的研究过程中[12]，受文[15,16]等工作的启发，我们发现 OCL 语言和 Object-Z 语言具有很强的互补性[17]。本文给出了将 UML 类图模型及 OCL 表达式系统地转换到 Object-Z 规格说明的方法，本方法结合了 UML/OCL 模型直观性和 Object-Z 语言严密性的优点，可产生更为完全的 Object-Z 规格说明，并可以对 UML 类图模型和 OCL 表达式进行形式分析和验证。本文安排如下：第 2 节中介绍了 UML 类图到 Object-Z 规格说明的转换，在第 3 节中系统地给出了 OCL 表达式到 Object-Z 的转换规则，第 4 节中给出了

*)本文的研究工作得到国家自然科学基金(项目编号 60373072)、国家 973 项目(批准号 2002CB312001)和上海市教委第四期重点学科建设基金的资助。缪淮扣　教授，博士生导师，主要研究方向：软件工程、形式化方法等；陈怡海　博士研究生，主要研究方向：形式化方法。

一个实例研究，最后总结全文并提出了进一步的研究方向。

2 UML 类图到 Object-Z 规格说明的转换

2.1 UML 类图到 Object-Z 规格说明的转换规则

2.1.1 UML 类的形式化

类是构成类图的基础，因此首先对类的形式语义进行描述。类描述了具有相同特征、约束和语义的一组对象的集合。在语法上，一个类由类名、属性和操作组成，其中属性由属性名和属性类型组成，操作由操作名和操作参数组成，操作的参数又由参数名和参数类型组成。其中的属性和操作还可以存在某些约束。

首先给出类型的转换规则：假定 T 为一个 UML 类型。我们称 ***T*** 为 Object-Z 对 T 的形式化，Object-Z 对于 T 类型形式化定义如下：

- 如果 T 是一个已知基本类型的名字，如 unsigned integer，integer，real 和 Boolean，那么对应的 Object-Z 类型为N，***Z***，***R*** 和 ***B***；
- 如果 T 是一个已存在类的名字，则对应的 Object-Z 类型也为 T；
- 对于自定义的类型，则在 Object-Z 中定义为给定类型，在形式化时可以直接引用这些类型；
- 如果 UML 类型是一个数组类型 T[]，那么对应的 Object-Z 类型为 seq(T)；

在定义了类型转换规则以后，接着处理 UML 类中的属性。

- UML 类中的属性转换为 Object-Z 类中的属性，UML 属性的名称为对应的 Object-Z 属性名称；
- 假如 UML 类属性的可见性为 public，则在相应的 Object-Z 类模式的可见列表中进行声明；
- 对于 UML 类的属性的重数约束，则在相应 Object-Z 类模式的状态模式中用谓词进行约束；
- UML 属性的性质决定了相对应的 Object-Z 类属性的位置。假如属性是不可以被修改的，在 UML 中定义为{frozen}，则在 Object-Z 的类模式中声明，其他可以被修改的属性{changeable}则在 Object-Z 的状态模式中声明；
- 如果在 UML 中，提供了属性的初始值，根据属性的性质，将按如下的方式转换：假如属性的性质是{changeable}，则在对应的 Object-Z 的初始状态模式中进行声明；假如属性的性质是{frozen}，则在 Object-Z 类的公理描述中进行声明；
- 在 UML 中，属性的作用域既可以是类(class)，也可以是实例(instance)。当一个属性的作用域为它的类时，就意味着类的所有实例都必须共享同样的值，假如属性的作用域说明为实例时，这意味着每一个类的实例都可以拥有自己的属性值。UML 中的缺省定义是实例作用域。当属性拥有类作用域时，在相应的 Object-Z 类模式中，必须保证所有的类实例都拥有同样的值。因此该属性在 Object-Z 类模式的局部定义中定义为常量；
- 在 UML 中，在属性的名称前面加上斜线(/)表示该属性包含一个导出值，在 Object-Z 中声明为 Δ 说明引导的从属属性。

下列规则处理 UML 类中的操作：

- UML 类中的操作转换为 Object-Z 类中的操作模式，操作的名称转换为操作模式的名称；
- 如果操作具有参数，则对参数的转换规则可定义如下：每一个操作参数的类型将根据前面定义的类型转换规则进行转换，如果参数的方向为 in，则转换为 Object-Z 中的输入变量，假如参数的方向为 out，则转换为 Object-Z 中的输出变量，假如参数的方向为 inout，则在 Object-Z 中分别转换为输入变量和输出变量；
- 操作中的前置条件和后置条件在操作模式的谓词部分声明；
- 如果操作的可见性是 public，则也在 Object-Z 的可见列表中进行声明；

在对类型类属性和操作定义了转换规则后，下面定义整个 UML 类到 Object-Z 类的转换规则：

- UML 类将转换为 Object-Z 中的类模式，UML 类名将作为 Object-Z 类模式的名字；
- UML 模板类的参数则在对应 Object-Z 类模式的参数列表中列出；
- UML 类的所有直接父类的名称将在对应的 Object-Z 类模式的继承类中列出；
- 对于 UML 类中的属性和操作，则按前面定义的方法进行转换；

2.1.2 关系的形式化

在 UML 中，类之间的关系包括关联和继承和依赖，而关联又包括一般关联、聚合关联和复合关联。其中复合表达了对象之间的部分与整体关系。

2.1.3 继承关系的形式化

由于 Object-Z 直接支持类之间的继承关系，因此，UML 类的继承关系的形式化方法比较简单，在子类的 Object-Z 类模式中直接引用父类就表示了两者之间的继承关系。

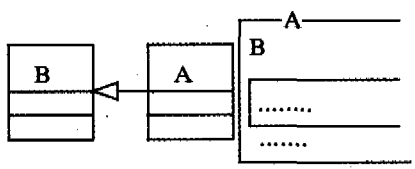

图 1 继承关系

在图 1 中，我们仅给出了一个单继承的例子。Object-Z 也能很好地支持多继承的概念。其结果等同于按任意顺序继承每一个单个的类。Object-Z 同样也支持操作的重定义，在继承类的可见列表中，我们可以重新定义同样名字的操作。

2.1.4 关联的形式化

关联表示一组具有共同结构特征、行为特征和语义的链接。每个关联可以附带一个名称来表示这个关联的含义。关联有两个端点，称作角色，表示它所连接的两个类各自在关联中的作用。角色可以具有多重性，就是说关联中的一个角色可以有多个对象来扮演，也称为度。关联包含一般关联、共享聚合关联和复合聚合关联三种。在大多数的研究方法中，如文[6]，将关联定义为相应 Object-Z 类的一个属性，缺点在于将关联定义为属性会增加类定义的复杂程度，从而不利于类定义的可修改性和重用性[24]。本文将关联视为一个自治的实体，用数学关系来形式地表示二元关联，并在对应 Object-Z 类的状态模式中进行定义，其优点在于可以方便增加关于关联的全局属性的描述。关联包括二元关联和多元关联，本文只讨论二元关联。

2.1.4.1 二元关联

二元关联的转换规则定义如下：

(1)类 A 和类 B 之间的可双向导航的二元关联 AssocAB 产生两个关系 f1 和 f2,分别定义在相对应的类的状态模式中,假如存在角色名,则关系 f1 和 f2 的名字分别用隶属于类 B 和类 A 上的关联的角色名来替换。如果是单向关联,则只产生一个关系。

(2)关系的定义域和值域由每个关联端的度决定。

(3)关联的类型在模式的谓词部分描述。

(4)根据每个关联端的度数不同,关联还可以表示为如表 1 所示的各种函数。

表 1 在 Object-Z 中表示关联

函数名称	A 度数	B 度数
关系(↔)	*	*
部分函数(⇸)	*	0..1
全函数(→)	*	1
部分入射函数(⤔)	0..1	0..1
全入射函数(↣)	0..1	1
部分满射函数(⤀)	1.. *	0..1
全满射函数(↠)	1.. *	1
部分双射函数(⤖)	1	0..1
全双射函数(⤖)	1	1

2.1.4.2 共享聚合和复合聚合关联

在 UML 中另外两种可能的关联形式为共享聚合(shared aggregation)和复合聚合(composition aggregation)关联,两者都可以用来表示整体和部分的关系。共享聚合的精确语义随着应用领域和建模人的不同而不同,部分实例的创建的次序和方式是没有定义的[4,p36]。因此本文将共享聚合作为一般关联来处理。而对于复合组合而言,部分对象只能属于一个整体,而且整体和部分同时存在,同时消亡。我们使用 Object-Z 中的©符号表示对象之间的复合聚合关系。

2.1.4.3 依赖关系

依赖关系是一种二元关系,依赖关系表示单个或者一组模型元素需要其他模型元素用于它们的规格说明或者是实现[4,p60],这意味着客户方元素的完全的语义在语义上或者是在结构上依赖于供应方元素的定义。因此在 UML 中,依赖关系的语义是可变的并且是依赖于上下文的,故本文不考虑依赖关系到 Object-Z 规格说明的转换。

2.1.5 关联类的形式化

当两个类之间的关联存在一些信息无法用关联表达时,往往用到关联类。关联类兼有关联和类的性质,因此关联类的形式化需要按两种方法进行。一方面,要按照类图的形式化规则将关联类转化为一个 Object-Z 类模式;另一方面,还要按关联转换为关联类中的属性。在转换时,将关联类的名字后加上 Cls 作为它的类模式名,加上 Ass 作为其关联模式名,这样做的目的是防止模式的重名。比如一个关联类的名字是 AssociationClassOne,则其形式化后的类模式名为 AssociationClassOneCls,关联模式名为 AssociationClassOneAss。

2.1.6 系统类的形式化

系统类的形式化是指将整个 UML 类图形式化为一个 Object-Z 系统类模式。前面我们已经将类图中的各种元素进行了形式化,在系统类模式中,我们可以直接引用这些类模式。系统类的形式化规则如下:

(1)UML 类图的名称作为系统类模式的名称;

(2)当类作为类图的一个组成部分时,代表了某一时刻该类的对象的一个集合,根据这个语义,UML 类图中的各个类对应的 Object-Z 类在系统类的状态模式中实例化为集合。

2.1.7 讨论

已经有多个研究小组提出如何把 UML 类图模型形式化和转换为 Z 规格说明[8,9],Object-Z[6,7]规格说明,TCOZ[11]规格说明,COOZ 语言[10]规格说明。这些方法对于 UML 类和关系的转换方法相互类似并且也是本文工作的基础。我们的方法和它们的不同在于本文的目的是为了使用 UML 类图来方便用户构造形式规格说明,而并非是对 UML 类图模型的完全的形式化,并且本文讨论了方法和属性的范围。我们只考虑那些有自然的 Object-Z 表示的 UML 模型元素,因此我们只考虑二元关联,不考虑抽象类,不区分共享聚合和关联,也不考虑关联的泛化。

3 OCL 表达式到 Object-Z 的转换

上节讨论了 UML 类图模型到 Object-Z 规格说明的形式转换规则,应用这些转换规则,只能产生一个规格说明的轮廓,而对模型中的约束信息无法处理。OCL 语言可以用来表达单纯用图形难以表示的 UM 模型的约束信息,OCL 语法非常简单,其核心是表达式。OCL 表达式可以用于不同的场合,例如用于定义类的不变式、操作的前置条件和后置条件。OCL 是有类型的,因此一个 OCL 表达式也是有类型的。OCL 类型可以分为两个部分:预定义类型和用户自定义类型。而预定义类型包括基本类型和集合类型。基本类型包括整型,实型,布尔类型和字符类型,而集合类型则包括 Set(T)(集),Sequence(T)(序列),Bag(T)(包)和 OrderedSet(T)(有序集)。我们给出 OCL 预定义的标准类型以及操作到 Object-Z 的转换规则。

3.1 预定义的基本类型

转换规则 1(整数) 在 Object-Z 中的预定义类型Z 可以对应于 OCL 整数类型。表 2 给出了所有的关于整数的 OCL 操作与 Object-Z 的对应关系。

表 2 整数上的操作转换

OCL	Object-Z	OCL	Object-Z
a:Integer	a∈Z	a−b	a−b
a=b	a=b	a * b	a * b
a<>b	a≠b	a/b	a/b
a<b	a<b	a. mod(b)	a mod b
a>b	a>b	a. div(b)	a div b
a<=b	a<=b	a. abs()	max(−a,a)
a>=b	a>=b	a. max(b)	max(a,b)
a+b	a+b	a. min(b)	min(a,b)

转换规则 2(布尔类型) 和 Z 语言相比,在 Object-Z 中新增布尔类型 B,它对应于 OCL 语言中的布尔类型(表 3)。

表 3 布尔类型上的操作转换

OCL	Object-Z	OCL	Object-Z
a:Boolean	a∈B	a xor b	¬a=b
a=b	a=b	not a	¬a
a and b	a∧b	a<>b	a≠b
a or b	a∨b	a implies b	a⇒b

转换规则 3(字符串型) 由于在 Object-Z 中没有和字符串类型对应的类型,在 Object-Z 中,可先定义给定类型

[char]，然后可用 seq char 来模拟字符串类型。OCL 在 String 上的操作(除了 toUpper 和 toLower 两个操作以外)可以对应于 seq char 的 Object-Z 表达式(见表 4)。

表 4 字符类型上的操作转换

OCL	Object-Z	OCL	Object-Z
a:String	a∈ seq char	a. concat(b)	a⌢b
a=b	a=b	a. subString (lower. upper)	a1{x:ℕ | lower <<x<<upper}
A<>b	a<>b	a. toUpper	未定义
a. size	#a	a. toLower	未定义

转换规则 4(实型) 在 Object-Z 中虽然没有对应于实数的预定义类型，但是有多篇文章[21,22]详细地讨论了如何在 Z 语言中表示实数。CadiZ[23]中定义了如何在 Z 语言中表示有理数和实数及其对应的操作，我们用相同的方法，在 Object-Z 中扩充定义实数类型ℝ和相对应的操作，用ℝ来对应于 OCL 中的实型。

转换规则 5(枚举类型) OCL 提供了 enum{value1,…. valuen}的句法来定义枚举类型，在 OCL 表达式中，可以通过前缀符号#来引用变量的值。OCL 中的枚举类型可以对应于 Object-Z 中相应的枚举类型机制，见表 5。

表 5 枚举类型的操作转换

OCL	Object-Z
a#=b#	a=b
a#<>b#	a≠b

3.2 集(Collection)类型

集表示成组的对象或元素，OCL 提供了一个抽象类型 Collection 和四种可以在表达式中使用的具体集类型：集合、序列(Sequence)、包(Bag)和有序集合(Ordered Set)，每一种类型都支持表示在一组对象上约束的各种预定义操作。而 Collection 类型是其他四种类型的抽象父类型，用于定义对所有集合类型的共同操作。下面给出集合、序列、包和有序集合的转换规则。

转换规则 6(集类型) 假定 T 为一个 OCL 类型。我们称 T 为 Object-Z 对 T 的形式化，Object-Z 对于 T 的集类型的形式化定义如下：

• Set(T)，集合是一个数学概念，它表示所有 T 的可能的子集，在 Object-Z 中可以用ℙ(T)来表示；

• Bag(T)，包可以包含重复元素的对象组，其中的元素可以出现多次，在 Object-Z 可以用[T]来表示；

• Sequence(T)，序列也是一个集合，其中的元素是有次序的并且一个元素可以出现一次以上，在 Object-Z 可以用 seq(T)来表示；

• OrderedSet(T)，一个有序集中的元素是依从小到大的顺序出现的，定义在集合 T 上的有序集合 totord[T] 的形式定义可以描述为下图所示的通用式定义[19]。

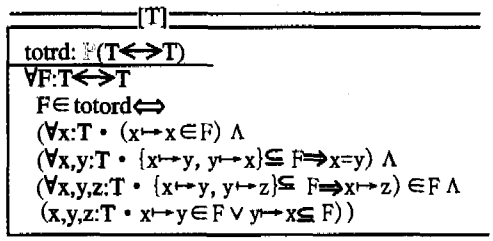

图 2 有序的通用式定义

表 6、表 7 和表 8 分别给出了定义在集合，序列和包类型上的 OCL 操作和相应的 Object-Z 表达式。其中：

• T 为一个 OCL 类型，在类型 T 上定义集类型，其中 tt:T；ss,ss2:Set(T)；bb,bb2:Bag(T)；se,se2:Sequence(T)；

• ss,ss2,bb,bb2,se,se2 和 tt 分别是 Object-Z 对应于 ss,ss2,bb,bb2,se,se2 和 tt 的形式化表示；

• 对于 sum 操作，T 必须为整数类型。

3.1.3 迭代操作的转换

转换规则 7(迭代操作) OCL 提供了对集合类型上元素的迭代操作，可以用一个已有集合的每一个元素对一个迭代表示进行求值。基于计算的结果，元素可能包含在结果集合里。迭代操作有 any(expr)，collect(expr)，collectNested(expr)，exists(expr)，forAll(expr)，isUnique(expr)，iterate(…)，one(expr)，reject(expr)，select(expr) 和 sortBy(expr)。

对于集合元素上的迭代操作的转换定义如表 9。

表 6 集类型的操作转换

OCL	Object-Z	OCL	Object-Z
ss. Bag(T)	ss⊆T	ss->size	# ss
ss=ss2	ss=ss2	ss->count(tt)	#(ss⌒{tt})
ss<>ss2	ss≠ss2	ss->includes(tt)	tt∈ss
ss->union(ss2)	ss⌣ss2	ss->includesAll(ss2)	ss2⊆ss
ss->union(bb)	ss×{1}⊎ bb	ss->includesAll(bb)	dom(bb)⊆ss
ss->intersection(ss2)	ss⌒ss2	ss->includesAll(se)	ran(se)⊆ss
ss->intersection(bb)	ss⌒dom(bb)	ss->excludes(tt)	¬(tt∈ss)
ss-ss2	ss\ss2	ss->excludesAll(ss2)	ss⌒ss2=∅
ss->symmetricDifference(ss2)	(ss⌣ss2)\(ss⌒ss2)	ss->excludesAll(bb)	ss⌒dom(bb)=∅
ss->including(tt)	ss⌣{tt}	ss->excludesAll(se)	ss⌒ran(se)=∅
ss->excluding(tt)	ss\{tt}	ss->isempty	ss=∅
ss->asBag	ss×{1}	ss->notEmpty	¬(ss=∅)
ss->assequence()	seq ss	ss->sum	Σ(xx). (xx∈ss | xx)

表 7 序列的操作转换

OCL	Object-Z	OCL	Object-Z
se, Sequence(T)	se∈ seq(T)	se->including(tt)	se⌢<tt>
se=se2	se=se2	se->excluding(tt)	squash(se⩤{tt})
se<>se2	se≠se2	se->size	#se
se->union(se2)	se⌢se2	se->count(tt)	#(ss⌢{tt})
se->append(tt)	se⌢<tt>	se->includes(tt)	tt∈ ran(se)
se->prepend(tt)	<tt>⌢se	se->includesall(se2)	ran(se2)⊆ran(se)
se->subSequence(i,j)	((1..j)◁(se)・((1..i+1)⩤se))	se->excludes(tt)	¬(tt∈ ran(se))
se->at(i)	se i	se->excludesAll(tt)	ran(se2)⊆ran(se)
se->first	head se	se->excludesAll(se2)	ran(se)∩ran(se2)=∅
se->last	last se	se->excludesAll(ss)	ss∩ran(se)=∅
se->asSet	ran(se)	se->isempty	se=<>
se->asBag	item se	se->notEmpty	¬(ss=<>)
		se->sum	∑(xx).(xx∈dom(se)|se(xx))

表 8 包类型的操作转换

OCL	Object-Z	OCL	Object-Z
bb:Bag(T)	bb in T	bb->count(tt)	count bb tt
bb=bb2	(bb⊑bb2)∧(bb2⊑bb)	bb->includes(tt)	tt in bb
bb<>bb2	¬(bb=bb2)	bb->includesAll(bb2)	dom(bb2)⊆dom(bb)
bb->union(bb2)	bb⊎ bb2	bb->includesAll(ss)	ss⊆dom(bb)
bb->union(ss)	cf. bb->union(ss->asBag)	bb->includesAll(se)	ran(se)⊆dom(bb)
bb->intersection(bb2)	bb⊍ bb2	bb->excludes(tt)	¬(tt in bb)
bb->intersection(ss)	dom(bb)∩ss	bb->excludesAll(bb2)	dom(bb2)∩dom(bb)=∅
bb->including(tt)	bb⊎⦇tt⦈	bb->excludesAll(ss)	dom(bb)∩ss=∅
bb->excluding(tt)	bb⩤tt	bb->excludesAll(se)	dom(bb)∩ran(se)=∅
bb->asSequnce	未定义	bb->isempty	bb=⦇⦈
bb->asset	dom(bb)	bb->notEmpty	¬(bb=⦇⦈)
bb->size	count(bb)	bb->sum	∑(xx)・(xx∈dom(bb)|xx×bb(xx))

表 9 迭代操作的转换

OCL	Object-Z
ss->any(exprtt)	∃tt:ss∧exprtt
bb->any(exprtt)	∃tt:ss・tt∈dom(bb)∧exprtt
se->any(exprtt)	∃tt:ss・tt∈ran(se)∧exprtt
ss->select(tt|boolexprtt)	{tt|tt∈ss∧boolexprtt}
bb->select(tt|boolexprtt)	{tt,nn|tt∈dom(bb)∧boolexprtt∧nn∈N∧nn=bb(tt)}
ss->reject(tt|boolexprtt)	{tt|tt∈ss∧¬boolexprtt}
bb->reject(tt|boolexprtt)	{tt,nn|tt∈dom(bb)∧¬boolexprtt∧nn∈N∧nn=bb(tt)}
ss->collect(tt|exprtt)	{tt:ss|exprtt}
bb->collect(tt|exprtt)	{tt:dom(bb)|exprtt}
se->collect(tt|exprtt)	squash(λ(ii)・(ii∈(dom(se)|exprtt(se(ii)))
ss->forAll(tt|boolexprtt)	∀tt:ss・<boolexprtt>
bb->forAll(tt|bollexprtt)	∀tt:dom(bb)・<boolexprtt>
se->forAll(tt|boolexprtt)	∀tt:ran(se)・<boolexprtt>
ss->exists(tt|boolexprtt)	∃tt:ss・<boolexprtt>
bb->exists(tt|bollexprtt)	∃tt:dom(bb)・<boolexprtt>
se->exists(tt|boolexprtt)	∃tt:ran(se)・<boolexprtt>
ss->isUnique(exprtt)	μtt|exprtt
bb->isUnique(exprtt)	μtt:dom(bb)|exprtt
se->isUnique(exptrr)	μtt:ran(se)|exprtt
ss->one(exprtt)	#{tt:ss|exprtt}=1
se->one(exprtt)	#{tt:dom(bb)|exprtt}=1
bb->one(exprtt)	#{tt:ran(se)|exprtt}=1

3.1.4 OCL 表达式到 Object-Z 表达式的转换

本节将定义 OCL 表达式转换到 Object-Z 表达式。

转换规则 8(导航表达式) 导航表达式允许一个 UML 模型中类的各种属性,操作,角色和关联在约束的上下文中被引用。导航表达式的形式为 source. property,其中 property 可以是一个属性或者是一个关联端。在 UML 到 Object-Z 的形式转换过程中,UML 中类的属性被转换到 Object-Z 中的状态模式,因此关于属性的 OCL 表达式可转换到 Object-Z 状态模式的谓词。而由于关联被形式地表示为 Object-Z 中的关系,因此我们用关系运算来抽取出所需要的元素。

转换规则 9(不变式转换规则) 在 OCL 语言中,表示不变式的句法为 context〈class name〉inv:〈Boolean OCL expression〉。对于类的不变式,如果不变式仅涉及类的属性,则将其转换到对应 Object-Z 类的状态模式。

转换规则 10(属性和关联端初始值的转换规则) 在 OCL 语言中,可以对属性和关联端的初始值进行设置,对于属性和关联端初始值设置的 OCL 表达式,可以转换为 Object-Z 规格说明中的初始状态模式中的谓词。

转换规则 11(前置条件和后置条件转换规则) 在 OCL 中前置条件表示在操作执行前模型必须保持的状态,而后置条件表示操作完成以后模型的状态。对于任何一个 UML 类图中的操作,可以使用 OCL 语言来表达操作的前置条件和后置条件。对 UML 类图操作的前置条件和后置条件进行说明的 OCL 约束条件转换到相应的 Object-Z 操作模式的谓词部分。

转换规则 12(self 关键字的转换规则) 在 OCL 表达式

中,"self"关键字的类型是表达式所属的相关联的类,或者是封装或具有此操作或监护条件的类。这在表达式中可以省略,因此在转换到 Object-Z 表达式时,将其省略。

转换规则 13(allInstances 算子的转换规则) OCL 中的 allInstances 算子可以应用到一个 UML 类,它返回在当前状态下该类的所有实例的集合。在 Object-Z 中,Class. allInstance 转换为 ∀c:class。

4 实例研究

本节通过实例来演示本文定义的形式转换规则,图 5 给出了一个简化后的公司系统的 UML 类图模型例子。类 Person 具有以下属性:address(地址),age(年龄),ismarried(婚否),sex(性别),isUnemployed(失业),name(姓名)和 ssn(社会保障卡号码),类 Clerk 有 income 操作。Company 类具有的属性是:budget(预算),name(姓名),numberofemployee(雇员数目),类 Company 可以进行的操作有 dismiss(解雇)和 employ(雇佣)操作。关联 Worksfor 表示一个职员可以为 0 到多家公司工作,而一个公司可以有 0 到多个职员。

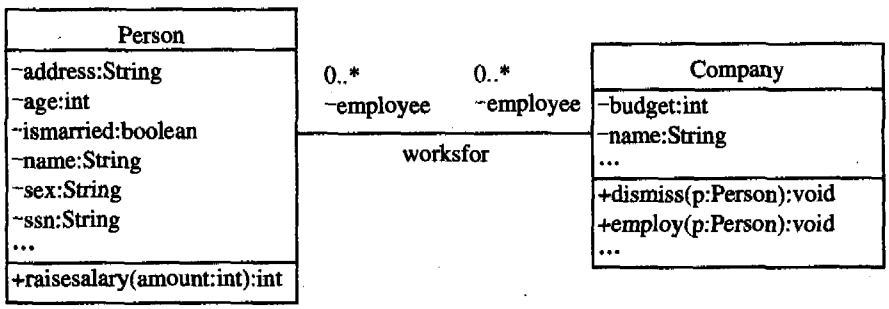

图 3 公司系统的类图

4.1 增添 OCL 约束条件

图 3 中的类图并不能完整地表示公司系统的所有相关的信息,因此只能产生一个形式规格说明的轮廓。为了使图 3 中给出的公司系统的类图模型更加完整,我们增添可以用 OCL 表达式说明而不能单纯使用类图表达的约束条件的信息,实现此模型的系统必须满足下列约束条件:

首先考虑如何对类的不变式进行说明,使用 OCL 可以对人的年龄属性加上如下的约束:

约束条件 1. (a)人的年龄必须大于 18 岁并且小于 60 岁。(b) 每个人的社会保障卡号码是唯一的

```
context Person
inv 1a:self. age)=18 and self. age<=60
inv 1b:Person. allInstance->forall p1,p2:Person|p1<>p2 implies p1. ssn<> p2. ssn
```

同样,对于类 Person 的 income 操作,

```
context Person::raisesalary(amount:int):int
    pre: amount>0
    post: self. salary=self. salary@pre+amount
```

对于公司类图,可以加上下述的约束条件:

(a). 一个公司最多有 100 个雇员。(b) 公司中年龄大于 18 岁的雇员的集合非空

```
context Company
        inv 2a:self. employee->size()<=100
        inv 2b: self. employee. select(p: Person|p. age)18)->notEmpty
```

对于公司类图,可以用 OCL 表达式对其属性和关联角色的初始值进行说明,下面的 OCL 表达式表示对于属性 numberofemployee 为 0,并且对于关联端 employee,它是一个空集。

```
context Company::numberofemployee:Integer
init:0
context Company::employee:Set(Employee)
init:Set{}
```

对于类 Company 的 employ 和 dismiss 操作,可以分别加上下述约束条件:

对于 employ 操作,其前置条件为公司没有已经雇佣该人,其后置条件为员工集合中增加此人。

对于 dismiss 操作,其前置条件为该人为公司员工,其后置条件为员工的集合中删除此人。

```
context Company::employ(Person p):void
    pre:not(employee->includes(p)
    post:numberofEmployee=numberofEmployee@pre+1
         employee=employee@pre->including (p)
context Company::dismiss(Person p):void
    pre: employee->include(p)
    post:numberofEmployee=numberofEmployee@pre-1
         employee=employee@press->excluding(p)
```

为了演示的目的,考虑类 Company 的第二个 OCL 不变式表达式 self. employee. select(p:Person|p. age>18)-> notEmpty 是如何转换到 Object-Z 表达式的。令 T(expr)表示 Object-Z 对 OCL 表达式 expr 的形式化。将 OCL 不变式 self. employee. select(p:Person|p. age>18)->notEmpty 转换到 Object-Z 表达式的处理过程如下:

T(self. employee. select(p:Person|p. age>18)->notEmpty)
===>T(self. employee->select(p|p. age>18)=∅(参见表 6 集类型的操作转换)
===>{p(employee(self)|p. age>18}=∅(参见表 9 迭代操作转换规则和规则 12)

4.2 UML 类图模型及 OCL 约束的转换

本节定义了公司—雇员类图模型上的各种 OCL 约束条件,在此应用前面定义的 OCL 到 Object-Z 的转换规则,图 3 中的类 Person 和类 Company 可以分别形式转换为图 4、图 5 所示的 Object-Z 规格说明。在图 6 中定义了公司系统的系统类模式。

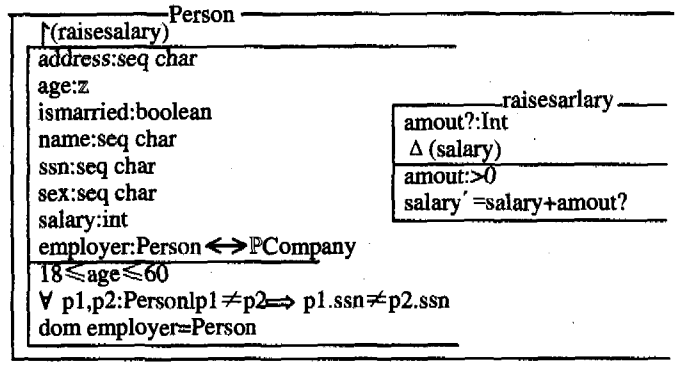

图 4　Person 类的 Object-Z 类模式

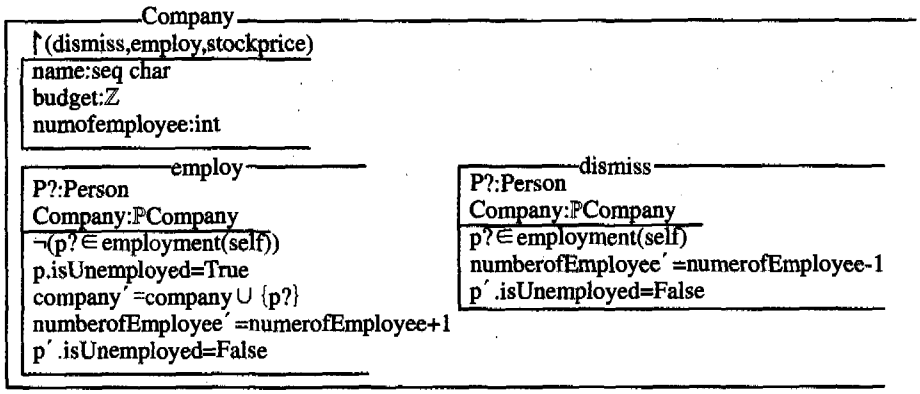

图 5　Company 类的 Object-Z 类模式

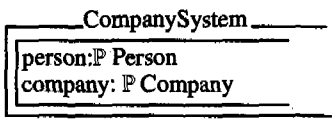

图 6　系统类模式

5　工具支持

当一个带有 OCL 约束条件的 UML 类图被转换成了一个 Object-Z 规格说明，我们就可以对其进行初始状态存在性证明[25]、相关的操作的前置条件简化[26]以及各种性质的证明等形式化的验证；也可以采用动画模拟的技术进行软件确认[27]。

为了使本文提出的 UML/OCL 模型形式验证与确认过程自动化，我们已设计实现了一系列的工具来支持我们的方法。转换和验证过程由下列的步骤组成(图 7)。

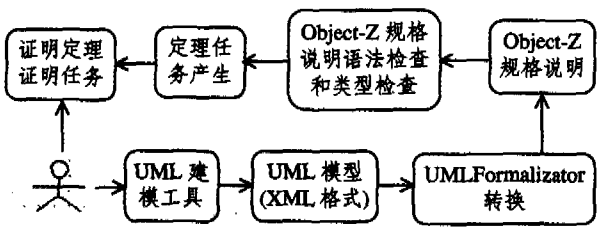

图 7　UML 模型到 Object-Z 规格说明的自动转换

(1)从 UML 模型到 XML 格式的模型信息的生成。作为第一步，用户使用 UML 建模工具生成 UML 模型，利用建模工具的 XMI 输出功能，生成 XMI 文件。

(2)从 UML 模型到 Object-Z 规格说明的转换。UML-Formalizer[20]从 XMI 文件中获取 UML 模型信息，根据转换规则，生成 Latex 格式的 Object-Z 形式规格说明。

(3)Object-Z 规格说明的语法分析和类型检查，对 Object-Z 规格说明进行语法检查和类型检查。

(4)定理证明任务的自动生成，以 Latex 格式的 Object-Z 规格说明为输入，经过转换后产生证明责任，最终保存在".tex"文件中，从定理证明器 Z/EVES 中导入进行证明。

小结　UML 是一种图形化的面向对象的标准建模语言。UML 类图和 OCL 表达式可以对系统的静态属性进行描述。尽管 UML 语言容易理解和使用，但是 UML 语言缺乏精确的形式语义，这给 UML 模型的形式验证和确认带来了困难。在创建 UML 模型的时候，现有的 UML 建模工具能做一些有限的检查，例如命名冲突检查，避免循环泛化关系，实体类型检查等等。但是我们不能检查一个用 UML 图描述的系统是否具有某种性质进行证明。为了克服这个缺点，将 UML 语言和某种形式方法相结合是一个可行的方法。对于 UML 类图模型和 OCL 约束进行形式分析检查的其他主要工作包括 *Martin Gogolla*[28]、*Jackson*[29]、赖明志[30]、I. Porres[31]等人的研究工作。在文[28]中，作者提出用 USE 工具对 UML 类图模型和 OCL 约束进行确认验证的方法，在此方法中，用 OCL 语言来表达类的静态属性，然后转换为 USE 语言。对这些性质的验证是通过验证在给定的时间，系统的一个特定的实例不违反 OCL 性质。此方法考虑了类，属性，关联，操作，角色名称等概念。在文[29]中，作者提出一种对 UML 模型自动验证的方法。作者考虑了状态图的验证方法。作者为 UML 状态图提供一个操作语义，接着此语义被转换为 PROMELA 语言，然后使用 SPIN 模型检查工具来验证规格说明的一致性。在文[30]，作者提出一种嵌入式实时

系统的严格的建模方法，作者研究了类图转换到 PVS 规范语言的方法，然后使用 PVS 验证工具来对规格说明进行验证。在文[31]中，作者提出了将 UML 转换为 B 语言，然后通过 B 工具对 UML 模型进行形式验证的方法。Object-Z 规格说明语言是一种得到广泛应用的形式规格说明语言，并且有较为丰富的工具支持。本文系统提出了一种 UML 类图模型到 Object-Z 规格说明的转换方法并且按照最新的 OCL2.0 标准系统地给出了如何将 OCL 表达式转换到 Object-Z 表达式的转换规则，这样能产生更为完整的 Object-Z 形式规格说明，并提出了如何使用 Object-Z 语言和工具来对 UML 模型进行验证和分析方法，最后通过实例进行了演示，我们在已实现的原型工具 UMLFormalzier 上扩展实现了 OCL 约束到 Object-Z 规格说明的自动转换。本工作的研究目的是在软件开发过程中结合 UML 和形式化方法，通过将 UML/OCL 模型转换到 Object-Z 规格说明，我们不仅可以通过 UML 类图来方便地构造 Object-Z 形式规格说明，并且可以通过用本课题组开发的系列 Object-Z 规格说明支持工具来对形式规格说明进行分析和验证。

参 考 文 献

1 US Department of Commerce's National Institute of Standards and Technology. http://www.nist.gov/public affairs/releases/n02-10.htm

2 Whittaker J A, Voas J M. 50 years of software: key principles for quality. In: IT Pro, November|December 2002. 28～34

3 Snook C, Harrison R. Practitioners Views on the Use of Formal Methods: An Industrial Survey by Structured Interview. Information and Software Technology, 2001,43(4):275～283

4 OMG. UML 2.0 Superstructure Specification, October 8, 2004, Document-ptc/04-10-02, http://www.omg.org

5 Bruel J M. Integrating Formal and Informational Specification. Why? How? 2nd International Workshop on Industrial-strength Formal Techniques(WIPT'98), Oct. 1998. Boca Raton, FL, USA

6 Kim S, Carrington D. An Integrated Framework with UML and Object-Z for Developing a Precise and Understandable Specification: The Light Control Case Study. In: Proc of the 7th Asia-Pacific Software Engineering Conf, Dec 2000. 240～248

7 Kim S, Carrington D. Formalizing the UML clabb diagrams using Object-Z. Proceedings UML99 Conference, Lecture Notes in Computer Science 1723, 1999

8 韦银星，张申生，曹健. UML 类图的形式化与分析. 计算机工程与应用，2002(10)

9 Dupuy S, Ledru Y, Chabre-Peccoud M. An overview f RoZ - a tool for integrating UML and Z specifications. 12th Conference on Advanced information Systems Engineering (CAiSE'2000), 2000

10 黄春荣，李宣东，郑国梁. UML 模型到 COOZ 归约的形式化转换. 计算机工程与应用，2003(20)

11 许维新. 基于 TCOZ 的 UML 视图的形式化模型及验证:[硕士论文]. 上海：华东理工大学，2004

12 Chen Yihai, Miao Huaikou. Integrating Object-Oriented Methods and Formal Methods For Requirement Engineering, Journal of Harbin Institute of Technology(New Series), 2004,11(3)

13 Warmer J, Kleppe A. The Object Constraint Language-Precise Modeling with UML. Addison-Wesley, 1999

14 OCL Center. http://www.klabbe.nl/ocl/index.html

15 Hung Ledang, Sououeres J. Derivation Schemes from OCL Expressions to B: [Technical Report]. A02-R-042. May 2002, LORIA

16 Marcano R, Levy N. Using B formal specifications for analysis and verification of UML/OCL models. Workshop on Consistency Problems in UML-based Software Development. 5th International Conference on the Unified Modeling Language, Dresden; Germany, 2002

17 陈怡海，缪淮扣. OCL 与 Object-Z 作为 UML 约束语言的分析比较. 计算机科学，2004(12)

18 Smith G. The Object-Z Specification Language. American Kluwer Academic Publishers, 2000

19 缪淮扣，李刚，朱关铭. 软件工程语言—Z. 上海科技文献出版社，1999

20 陈怡海，缪淮扣，高如海. 基于 XMI 格式的 UML 模型的交换. 见：第十届全国计算机大会，2003 年 11 月，北京

21 Valentine S H. Putting Numbers into the Mathematical Toolkit. Proceedings of the Seventh Z User Meeting, Springer 1993

22 Oliveria W R, Barros R S M. The real numbers in Z. In: Duke D J, Evans A S. eds. 2nd BCS-FACS Northern Formal Methods Workshop. Electronic Workshops in Computing. Springer-Verlag, 1997

23 CADiZ 系统. http://www-users.cs.york.ac.uk/～ian/cadiz/cadizkits.html

24 Tanzer C. Remarks on object-oriented modeling of associations. JOOP, 1995,7(9):43～46

25 Miao Huaikou, McDermid J A, Toyn I. Proving the existence of sate in Z specifications. Chinese Journal of Advanced Software Research, 1995,2(3):326～337

26 缪淮扣. Z 前置条件的简化. 软件学报，1997,8(9)

27 朱江，陈怡海，缪淮扣. Object-Z 规格说明的结构模拟动画技术. 上海大学学报，2005

28 Gogolla M, Richters M, Bohling J. Tool support for validating UML and OCL models through automatic snapshot generation. Proceedings of the 2003 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on Enablement Through Technology, 2003

29 LAI Mingzhi, YOU Jinyuan. UML statechart based rigorous modeling of real time system. Journal of Harbin Institute of Technology (New Series), 2005,12(1)

30 Porres I. Modeling and Analyzing Software Behavior in UML: [PhD thesis]. Turku, Finlang: Turku Center for Computer Science, Nov. 2001

31 Lano K, Clack D, Androutsopoulos K. UML to B: Formal Verification of Object-Oriented Models. In: IFM 2004, LNCS 2999, 2004. 187～206