计算机科学2000Vol 27№. 2

基于 Object-Z 的 UML 对象模型的形式化

The Formalization of Object Model in UML Based on Object-Z

杨卫东 蔡希尧

（西安电子科技大学软件工程研究所 西安710071）

Abstract UML is the main visual Object-oriented modeling language currently, which is used widely and supported by most CASE tools. Comparing with traditional Object-oriented methods, UML describes its semantics and syntax more rigouly by using metamodel and Object Constrain Language. But some important concepts in UML are not specified clearly. This paper presents a formal specification for object model of UML, mainly includes the concepts of class, association, association class, aggregation, and inheritance, etc, so that the analyse, verification, refine, and consistent cheking can be applied to object model.

Keywords Object-oriented, Formal specification, UML, Object-Z, Object model, Schema

1 引言

UML 作为面向对象的可视化建模语言，已被对象管理集团(OMG)作为面向对象分析和设计的标准，获得了众多工具的支持。UML 提供了不同抽象层次的描述以支持面向对象的分析、设计和实施，它从不同的视图描述软件系统，减少了建模的复杂度，更为重要的是建立了基于元模型的体系结构，提供了较为灵活的扩充机制，使开发人员可以根据不同的领域需求定制 UML，也易于加入新的建模概念。

UML 与以前的主流软件开发方法(OMT、BOOCH 方法和 OOSE)相比，其中一个显著特点就是加强了建模语言语义的严格描述，主要使用了元模型、对象约束语言(OCL)和自然语言。元模型的使用严格地描述了 UML 建模技术的抽象语法，可用于语义描述的基础。OCL 适合于表达约束规则，但用于描述动态语义则不容易被理解，也缺乏语义的形式描述。因此，UML 虽然在建模概念的语法和语义的描述上进了一步，还是难以支持软件模型的分析、验证以及一致性检查等。

Object-Z 是基于 Z 语言的面向对象的扩充，与 OCL 不同，Object-Z 和 Z 具有严格的语义，提供了较强的证明、分析和验证的能力，具有众多支持工具。Object-Z 支持主要的面向对象的概念(类、继承、多态等)，用 Object-Z 作为 UML 模型的形式化描述语言，可以在软件开发的初期阶段分析模型、检查错误，提供了验证模型和开发过程中严格求精的基础，能够支持模型的一致性检查等。

对象模型是 UML 中最重要的模型，本文用 Object-Z 描述了 UML 对象模型的主要概念，如：类、关联、聚合、继承等。

2 UML 对象模型的形式化

UML 从四个视图描述软件系统，即使用实例(use case view)、逻辑(logical view)、物理(physical)、交付(deployment view)，逻辑视图主要描述了软件的逻辑

它们逐步整理到当前解答框架中。

参 考 文 献

1 Handley M, Crowcroft J. The World Wide Web. UCL Press, 1995

2 Dunteman J. Mosaic Pocket Companion. Coriolis Group Inc., 1995

3 Kuokka D, Harada L. Matchmaking for Information Agents. IJCAI-95

4 Bestavros A. Demand-based document dissemination to reduce traffic and balance load in distributed information system. In: Proc. of the 1995 7th IEEE Symposium on Parallel and Distributed Processing. IEEE, 1995

5 Newton P. Capitalizing on the World Wide Web's Knowledge Infrastructure. 12th Conf. Proc. ADBS-IDT, 1995

组织，建立了软件系统的对象模型，构成了整个体系结构的骨架，主要由类图和对象图表示，包括对类及其属性、操作的描述和类之间的关联的描述。

2.1 类的描述

类是具有共同特征和语义的对象集合的描述，UML 对象模型中每一个类由 Object-Z 中的类框架形式化地表示，属性和对象标识表示为状态变量，与类相关的不变量（在 UML 中通常由注释表示）由 Object-Z 中类框架中的谓词部分表示，类型必须定义为 Object-Z 中的基本类型或框架，值和变量的绑定表示了类的实例。公有的属性和操作在 Object-Z 中用可见性表 ｜ ( ) 表示，模板（或称参数类）描述了一族类，每一个类通过参数和值的绑定说明。

2.2 关联的描述

UML 中定义类之间的关系为关联，关联的实例称为链接，是相关联的对象元组。在二元关联中，它是一个二元组。为了更清楚地表示关联，可以使用角色名。角色表示了关联的一端和类在关联中所起的作用。从本质上讲，角色是一对象集合，由于参与关联的对象通常并不是类的整个对象集合，而是它的一个子集，因此角色的使用使关联的语义更加清晰（特别是对"稀疏"的关联），关联有时可能具有自己的特征，因此，需要将关联作为独立的实体来描述，UML 中称为关联类，在 Object-Z 中，关联是通过定义类的属性来表示的，为了表示关联类，对 Object-Z 加以补充，将关联统一描述如下：

·关联的两端称为关联端，它是关联连接的类的对象集合的子集合。设 A 和 B 为关联 Assoc 关联的两个类，用 AssociationEndA 和 AssociationEndB 表示关联 Assoc 的两端，则

$$\begin{aligned}&\text{AssociationEndA}:\mathbb{P}A\\&\text{AssociationEndB}:\mathbb{P}B\\&\text{Assoc}\triangleq\text{AssociationEndA}\leftrightarrow\\&\qquad\text{AssociationEndB}\end{aligned}$$

角色可以作为关联的一端在关联中描述，若 RoleA 和 RoleB 表示关联 Assoc 两端的角色，关联可以定义如下：

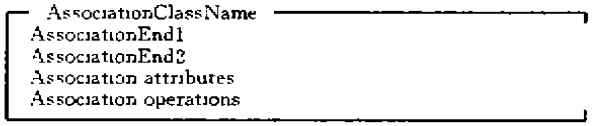
$$\text{Assoc}\triangleq\text{RoleA}\leftrightarrow\text{RoleB}$$

关联的属性和操作通常不属于任何一端（特别是在多对多的情况下），因此，描述如下：

设 Attri 为关连的某一属性，op 为关联的某一操作：

$$\begin{aligned}&\text{AssocAttribute}\triangleq(\text{AssociationEndA}\times\\&\qquad\text{AssociationEndB})\rightarrow\text{Attri}\\&\text{AssocOperation}\triangleq(\text{AssociationEndA}\times\\&\qquad\text{AssociationEndB})\rightarrow\text{op}\end{aligned}$$

·关联的描述形式如下：



其中，关联的两端分别由 AssociationEnd1 和 AssociationEnd2 表示，其属性由状态框架表示，操作由操作框架表示。

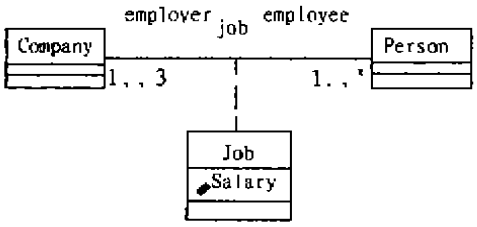
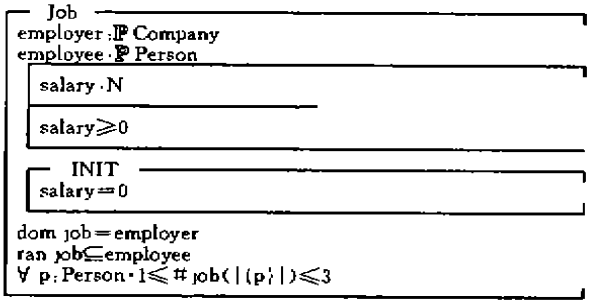


图1 关联类的 UML 表示

在这里只描述了二元关联，当然三元及三元以上的关联可以用类似的方法表示。通常三元关联可以转换为二元关联，（这里不再加以阐述）。

图1是用 UML 描述的一个例子，Job 是关联类，employer 和 employee 表示角色名，用 Object-Z 描述如下：



2.3 聚合关系的表示

聚合关系表示了部分和整体的关系，在 UML 中，将聚合关系视为一种特殊的关联，且只能是二元关联。聚合关系分为两种：共享的聚合关系（a shared aggregate）和合成聚合关系（a composite aggregate）。合成聚合关系是强"拥有"的聚合关系，其组成部分的实例不能共享，部分与整体具有相同的生命周期，整体对部分的一些动态语义是可传递的。共享聚合关系说明了较弱的"拥有"关系，没有对象实例的共享性和部分与整体实例的生存周期的依赖关系的约束，两种聚合关系都具有可传递性和非对称性，合成聚合关系构成了一棵严格的树。

UML 比 OMT、BOOCH 方法和 OOSE 采用了更严格的描述，用 OCL 描述其约束如下：

关联的两端只能有一端是聚合：

self. allConnections→select(aggregation()≠none)→size<=1

在合成聚合关系中对象实例不能被多个聚合实例共享：

self. aggregation=#composite implies self. multiplicity. max<=1

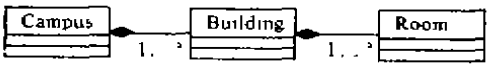
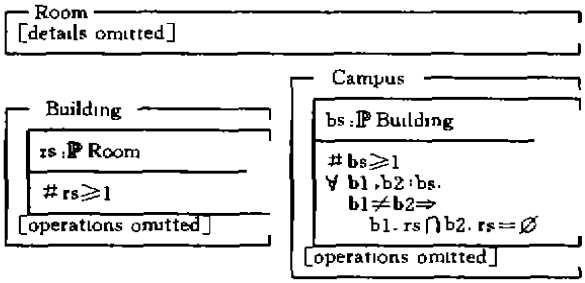


图2

在 Object-Z 中，共享的聚合关系可以用类引用表示，合成的聚合关系图2用 UML 描述了类 Campus、Building 和 Room 之间的合成聚合关系，称为包含关系[15]。图2是用 UML 表示的聚合关系，用 Object-Z 描述如下：



其中，类 Campus 的不变量说明了不同的 Building 实例不能包含同一个 Room 实例。合成的聚合关系在 Object-Z 中可以用符号Ⓒ直接表示，即：rs:ℙ Room Ⓒ，bs:ℙ Building Ⓒ

在 UML 和 Object-Z 中，并没有清楚地说明部分与整体的生存周期的依赖关系，而生存周期的依赖关系和共享性是聚合关联的两个最基本的特性，现在对这两个特性进一步加以阐述。聚合关联 agg 可以表示为：

agg：Whole↔Part

我们用 Containers 和 Components 分别表示现存的整体和部分的实例，它们是有限的实例集合：

Containers:ℙ Whole，Components:ℙ Part

下面的描述只考虑现存在的对象实例之间的链接，即：

∀ ct:ℙ Whole，cp:ℙ Part·

cp∈ct. agg⇒cp∈Components∧ct∈Containers

·部分对整体的生存周期的依赖关系，表明了离开了整体，部分是否可以独立存在。

cp∈ct. agg≡cp∈Components (1)

如果(1)式成立，则去掉 cp 和 ct 之间的链接 agg，意味着删除 cp(部分依附于整体)。

cp∈Components≡ct∈Containers (2)

如果(2)式成立，表示了 cp 的生存周期包含在 ct 的生存周期之内。

·整体对部分的生存周期的依赖关系。表明了整体的组成部分是否可以被删除。

cp∈ct. agg≡ct∈Containers (3)

如果(3)式成立，则 ct. agg 的值在其生存周期内不能改变。

ct∈Containers⇒cp∈Components (4)

如果(4)式成立，则整体的生存周期不能超过部分的生存周期，即当整体的某一个部分被删除时，整体也将不存在。

·共享性。说明了整体的实例是否可以共享同一部分实例。

∃ ct1，ct2:Containers|ct1≠ct2·

cp∈ct1. agg∩ct2. agg (5)

如果(5)式成立，表示多个整体实例可以共享同一个部分实例。

当然，生存周期的依赖关系还意味着封装性(是否 cp 仅可通过 ct 访问)和动态语义的可传递性等其它特性。如整体和部分的实例的生存周期相互依存，则 ct 实例的创建和删除操作必须传递到 ct. agg 的元素，如果再加上不可共享性，则拷贝操作也应被传递。

2.4 继承关系的描述

在 Object-Z 中，继承是一个递增的说明机制，据此新类可以由一个或几个现存的类导出。继承时，合并子类和父类的类型、常量定义和历史不变量，合并状态和初始化框架，同名的操作框架合并(名称冲突可以用重命名的方式解决)被继承的操作框架可以被重新定义(redef)或被删除(remove)，对属性可以重新命名。

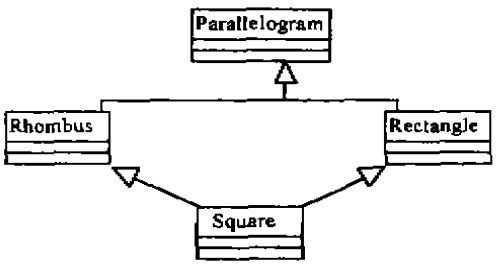
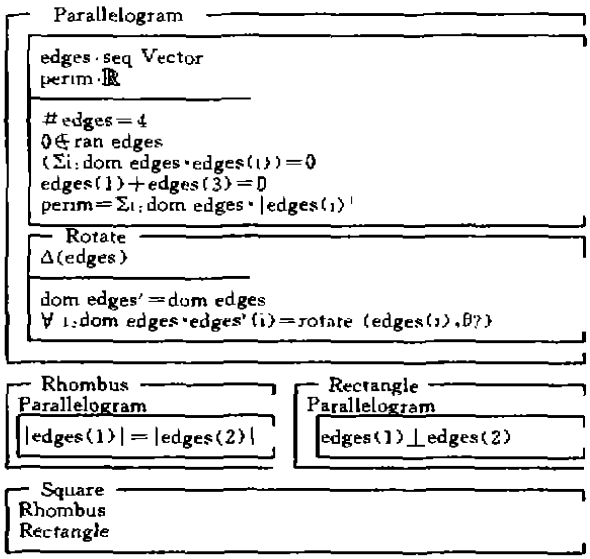


图3

图3说明了一个几何图形的类层次结构(用 UML 的类图描述),先引入全局类型 Vector 以及 Vector 上的函数和关系:

$|\_|$: Vector → Real; rotate: Vector × ℝ → Vector; _⊥_: Vector↔Vector; _+_: Vector × Vector → Vector

对任何向量 v,|v|表示向量的量(长度),对任何向量 v 和实数 θ,rotate(v,θ)是将 v 逆时针旋转 θ 度所得到的向量。对于任意两个向量 v 和 w,v⊥w 当且仅当向量 v 和 w 垂直,v+w 表示 v 和 w 相加得到的向量,0∈Vector 表明零向量。下面是用 Object-Z 对图3中类图的描述:



Parallelogram 类具有属性 perim 和 edges,操作 Rotate 其类的不变量表示 edge 的数量是4并且第一个 edge 和第三个 edge 的和为零(即方向相反大小相等)。类 Rhombus 和类 Rectangle 都继承类 Parallelogram 类 Rhombus 的第一条 edge 和第二条 edge 的大小相等,类 Rectangle 的第一和第二条 edge 垂直。类 Square 同时继承了类 Rhombus 和类 Rectangle,即一个 Square 既是 Rhombus 又是 Rectangle。

结论 主要以图和自然语言作为描述手段的非形式化的面向对象开发方法(OMT 等),虽然易于理解和使用,但难以精确描述 O-O 建模概念和软件模型,尤其是在开发关键应用的系统时显得不足。UML 以其基于元模型的四层体系结构和 OCL 文本语言作为补充描述,在这方面前进了一步,但这些还难以支持软件模型的分析、验证、求精、推理等。因此,UML 模型的形式化很有意义。

UML 作为当前主流的面向对象的建模语言,包含丰富的建模概念,同时,软件工程领域的研究、大规模软件开发以及软件重用的迫切需要,对软件开发方法提出了更高的需求,如对软件体系结构、设计样本、构件、构架等支持,由于 UML 基于元模型的四层体系结构和灵活的扩充机制,使得这些具有较高抽象层次的概念与软件开发方法的集成更为自然。

从另一方面讲,对 UML 的形式化提出了更高的要求,如对高于类的抽象层次的实体的描述(如 Package 等),因此,还有许多需要进一步研究的工作。

参 考 文 献

1 Booch G, et al. UML: A Unified Modeling Language. 1997. Available at: http://www.rational.com/uml
2 Nicholls J. Z Notation. Ver 1.2, 1995
3 Duke R, et al. The Object-Z Specification Language, Version 1, April 1991
4 Boehm B. Software Engineering Economics: An Empirical Investigation, CACM, 1984, 21(Jan.): 42~52
5 Bourdeau R H, et al. A Formal Semantics for Object Model Diagrams. IEEE Trans. on Software Engineering, 1995, 21(10)
6 Wing J M. A Specifier Introduction to Formal Methods. IEEE Computer, 1990, 23(Sept.): 8~24
7 Object Constraint Language, Ver 1.1 Available at, http://www.rational.com/uml
8 Harry A. Formal Methods Fact File VDM and Z, John Wiley & Sons, 1994
9 Lano K. Formal Object-oriented Development, Springer
10 Woodcock J, Loomes M. Software Engineering Mathematics. Pitman, 1988
11 Duke R, Rose G, Smith G. Object-Z: a Specification Language Advocated for the Description of Standards, 1994
12 Duke D, Duke R. Towards a Semantics for Object-Z. VDM 0: VDM and Z!, volume 428 of Lect. Notes in Comput. Springer-Verlag, 1990
13 Spivery J M. Understanding Z: A specification language and its formal semantics. Volume 3 of Cambridge Tracts in Theoretical Comput. Sci. Cambridge University Press, UK, 1988
14 Dong J, Duke R. Class Union and Polymorphism. TOOLS, 12, 1993
15 Dong J, Duke R. The Geometry of Object Containment: [Technical Report 94-17]. 1994
16 Cook S, Daniel J. Designing Object Systems: Object-Oriented Modeling with Syntropy. Prentice Hall, Englewood Cliffs, NJ, September 1994
17 Coleman D, et al. Object-Oriented Development: The Fusion Method. Prentice Hall, Englewood Cliffs, NJ, Object-Oriented Series edition, 1994
18 Evans A S, et al. The UML as a formal semantics for UML. In UML 8-Beyond the Notation, LNCS. Springer, 1998