Z 的面向对象扩充 COOZ 的设计*

袁晓东　郑国梁

（南京大学计算机科学与技术系　南京　210093）

摘要　为了使 Z 规格说明与面向对象开发方法相结合，本文在 Z 中扩充了对象类型和模块描述机制，使之成为面向对象的形式化规格说明语言 COOZ(complete object-oriented Z). 内容包括 COOZ 的设计思想、语法定义及说明、形式化语义、实例以及与相关研究工作的比较.

关键词　Z 语言，面向对象，形式化，规格说明，扩充.

中图法分类号　TP311

Z 是目前最为流行的一种形式化规格说明语言，其数学基础是集合论和一阶谓词逻辑，借助于模式来表达系统结构. 它提供了一种能独立于实现的、可推理的系统数学模型，具有精确、简洁、无二义性的优点，有利于保证程序的正确性，尤其适用于无法进行现场调试的高安全性系统的开发. 其不足之处是缺乏表达大型系统规格说明的必要机制. 面向对象方法是设计大型、复杂系统软件结构行之有效的方法，如果能在 Z 中引入面向对象描述机制，必将大大拓宽 Z 的应用领域，克服其现有的一些弱点，成为形式化技术和面向对象技术结合的一种途径. 我们在参考一些现有的 Z 面向对象扩充研究工作的基础上[1]，提出了全面支持面向对象思想和概念描述的 Z 面向对象扩充语言 COOZ(complete object-oriented Z)，从而将形式化方法应用于面向对象系统需求和设计的精确描述.

由于 Z 本身并不具备显式描述各种面向对象概念的机制[1]，我们在 COOZ 中引入了对象类型描述机制 Object 模式，在 Object 模式中对各种面向对象概念建立相应的描述方法，使得形式化规格说明具有较为清晰的结构. 为了进一步提高形式化规格说明的系统结构表达能力，提高清晰性和封装性，我们在对象类型之上又增加了一层模块描述机制. 在不同抽象层次上描述系统结构，便于我们对整个系统结构的理解和把握，又降低了单个模块和对象类型规格说明的复杂程度. 为了保证形式化规格说明含义的精确性，我们还给出了 COOZ 的形式化语义.

1　Object 模式

COOZ 扩充了 Z 中的类型概念，引入一种新的用以描述对象集的构造类型，称之为对象

---

* 本文研究得到国家自然科学基金和国家"九五"攻关项目基金资助. 作者袁晓东，1972 年生，博士生，主要研究领域为形式化方法，面向对象技术. 郑国梁，1937 年生，教授，博士导师，主要研究领域为软件工程.

本文通讯联系人：袁晓东，南京 210093，南京大学计算机科学与技术系

本文 1996-11-18 收到修改稿

类型. 对象类型的型构由对象的属性说明及各对象方法的型构所组成. 我们用 Object 模式来描述对象类型. 本节将具体介绍 Object 模式的语法定义及各成分的含义.

1.1 语法定义

我们采用 Z 的直观书写格式给出 Object 模式的语法定义,便于理解,其中各个成分再用 BNF 语法加以描述. 这里仅给出对 Z 扩充部分的定义,原有部分用... 代替,见文献[2].

Object-Type ::=

Object *Typename TypeParameters*
Supertype List
Local Definitions
Anonymous State Schema
Initialize Schema
Method Schemas

Real-Time and History Constraints

Typename ::= *IDENT*
TypeParameters ::= ["[" *Parlist* "]"]
Parlist ::= *IDENT* [<<*IDENT*][,*Parlist*]
Supertype List ::= { *Typename TypeParameters* ["[" *Renamelist* "]"] }
Renamelist ::= *IDENT* / *IDENT* [,*Renamelist*]
Local definitions ::= { *Typedef* | *Axdef* }
Typedef ::= *GIVENSETDEF*
　　| *FREETYPEDEF*
Axdef ::= *Schema-Text*
Schema-Text ::= *Declarations* ["|" *PRED*]
Declarations ::= *SDECL* {; *SDECL*}
Anonymous State Schema ::= "["*Schema-Text* "]"
Initialize Schema ::= **Init** ≙"[" *Declarations* {"|" *Post-condition* [**if** *Precondition*]}"]"
Precondition ::= *PRED*
Post-condition ::= *PRED*
Method Schemas ::= {[**internal**|**external**] [*] *Methodname* ≙
　　"["[△*Idlist*;] *Declarations* { "|" *Post-condition* [**if** *Precondition*] } "]"}
Methodname ::= *IDENT*
Real-Time and History Constraints ::= { *PRED* }
PRED ::= .....
　　| *ExtendPRED*
ExtendPRED ::= *BoolState*
　　|*Methodname*. **executing**
　　|*Methodname*. **waiting**
　　|⌈⌉["("*Interval*")"]
　　|⌈*PRED*⌉["("*Interval*")"]
　　|*PRED*⌢*PRED*["("*Interval*")"]
　　|**nonexecuting**["("*Interval*")"]
BoolState ::= *IDENT*
Interval ::="["*EXP*,*EXP*"]"
EXP ::= .....
　　|*REAL*
　　|*t*["("*Interval*")"]
　　|**exec** (*Methodname*) ["("*Interval*")"]
　　|$\int$*PRED* ["("*Interval*")"]
.....

1.2 类型参数

COOZ 中的 Object 模式可以带有类型参数成为参数化类型. 类型参数由类型名后的 TypeParameters 给出,可同时有多个类型参数,每个参数都可以通过任选项<<*IDENT* 指明其实际参数必须是类型 *IDENT* 的子类型.

1.3 子类型

Object 模式中 *Supertype List* 指出其父类型列表，每个父类型由类型名、实际类型参数和可选的改名列表组成. 改名列表可对父类型中的状态变量名和方法名进行改名. 子类型的规格说明中自动引入其所有父类型的规格说明，并通过模式合取操作将同名的模式加以合并. 通过子类型关系可实现多态性和动态定连. COOZ 中指向父类型对象的指针也能指向其子类型对象，因而在其所对应的实现语言中，方法调用的实现代码通过动态定连来确定.

1.4 局部定义

Local definitions 指作用范围局限于该 Object 模式的定义，包含类型定义和公式定义. 局部定义的目的是给出仅在该 Object 模式中需要使用的类型和常量、关系、函数等公式的定义，从而避免在高抽象层次中增加不必要的细节描述. 如果局部定义与全局定义同名，则在局部定义的作用范围内隐藏了全局定义，以局部定义为准. 局部定义的对象常量对不同的对象可有不同的值，但在对象生命期内其值不能被改变.

1.5 状态空间

每个 Object 模式中都有一个 *Anonymous State Schema*（无名状态模式）描述对象的属性说明及类不变式，亦即描述了该类对象的状态空间. 用无名模式来描述对象状态空间的目的有 3 个：①信息隐蔽. 由于没有名称，在该 Object 模式外即无法引用该模式，因而无法直接访问对象的状态量. ②便于将状态模式同方法模式加以区分. 所有的方法模式都是有名字的，无名的模式即为状态模式. ③在继承时，依照同名模式进行合并的原则，无名状态模式可以自动地加以合并.

1.6 实例化

由对象类型定义产生相应的实例对象称为实例化. 对象类型定义中 **Init** 模式即是初始化新创建的实例对象时自动调用的. **Init** 模式规定对象的初始状态值. 每个对象在创建时就同时产生在对象生命期内永不改变的具有唯一性的对象标识. 如实例化时遇到说明为对象类型的状态变量，则必须启动一新的对象实例化过程，该状态变量的值为新创建对象标识.

1.7 方 法

Object 模式中可以有任意多个方法模式. 一类对象可以接受的消息以及消息的定义必须通过该对象类型的方法模式来说明. 无名状态模式将被自动引入该对象类型的方法模式中. 方法模式的 ΔIdlist 中所包含的状态变量可以被该方法所修改，否则其值保持不变. 说明为 **internal** 的方法只能在对象内部使用，说明为 **external** 的方法可被其他对象调用，如缺省则为 **external**. 我们将方法模式分为主动模式和被动模式，模式名前加“*”表示主动模式，否则表示被动模式. 主动模式表示当前置条件满足时方法自动执行，而不需要被调用；被动模式则在被调用时才执行，它定义了当假设成立时方法的执行结果，当假设不成立时方法执行的结果是不确定的. 为了使方法的语义描述更加清晰，便于求精，我们将 Z 模式中混合在一起的假设和结果分开书写，用 **if** 显式地指明假设部分，并且一个方法模式中可有多对假设与结果，它们之间用单线隔开.

为了规格说明书写时的方便，作为一种替代形式，我们允许在 Object 模式中通过 ::*methodname* 只定义方法名，而方法模式可以在外面定义. 为避免混淆，在外面定义的对象方法模式名必须写成 *typename*::*methodname* 的形式.

1.8 约　束

为便于描述实时系统和并发系统的规格说明,在 COOZ 中可描述对象的实时约束和历史约束. 现有的 Z 面向对象扩充往往采用时序逻辑描述对象的历史约束,引入离散的时间概念来描述系统的实时性能. 由于时段演算比时序逻辑有更强的表达能力,且支持连续时间的概念,因此我们采用时段演算来表示对象的实时约束和历史约束. 详细内容将另文介绍.

2 模　块

COOZ 中的模块是一种结构描述机制. 对于复杂系统的规格说明,可将若干联系紧密、相互协作完成一定功能的对象类型封装为一个模块. 由模块统一规定外部接口,即可被其他模块调用的对象方法集. 该方法集包含于模块内部各个对象类型的方法集的并集中. 虽然模块与对象类型都是一种封装与抽象的机制,但模块的主要目的是为规格说明提供一层次描述手段,便于开发者从高抽象层次上了解系统结构以及系统设计、实现时的分工协作. 在模块内部只是各个对象类型的简单组合,不象对象类型那样有自己的状态模式,所以模块不是类型机制. 下面给出模块的语法定义,上文已经有的定义这里不再重复.

Module::=
Module *Modulename TypeParameters*
Importing Modules
Local Definitions
Contained Object-Schemas and Interfaces
Object-Type

Modulename::=*IDENT*

Importing Modules::={*Modulename TypeParameters*}

Contained Object-Schemas and Interfaces::={**Object** *Typename TypeParameters*[; *Methodlist*]}

Methodlist::=*Methodname*{,*Methodname*}

Object-Types::={*Object-Type*}

TypeParameters 表示模块的类型参数,一般来说将模块中各个对象类型所需要的类型参数合并起来即得到模块的类型参数. *Importing Modules* 表示被引入的模块,如在本模块中需要使用其他模块的对象和接口操作,则必须引入该模块. *Local definitions* 表示局部于本模块的类型定义和公式定义,常是提取本模块中几个对象类型所共有的局部定义所得. *Contained Object-Schemas and Interfaces* 表示本模块所包含对象类型的名称、实际类型参数以及该类对象可供其他模块调用的方法,而在本模块内部所有说明为 **external** 的方法都可被调用. *Object-Types* 为各个对象类型的规格说明,为书写方便起见,也可以在模块外面给出.

3 COOZ 的形式化语义

文献[3]给出了基于 Zermelo-Fraenkel 集合理论的形式化语义,我们将其扩充为 COOZ 的形式化语义. 由于篇幅的限制,这里只重点给出主要扩充部分——对象类型的形式化语义,Z 的形式化语义部分不再重复. 对象类型描述了一组对象的集合,所以我们首先给出对象的语义. 对象不同于其他类型的数据值,而是一个有生命期的动态变化的实体,因而对象的语义由对象标识、状态序列和执行的方法序列 3 部分组成. 给定一对象标识、状态序列和方法序列,我们用下面定义的 object 函数构造一对象:

$$object : NAME(\mathbb{N} \times (IDENT \pfun \mathbb{N}) \times (\mathbb{N} \times (\mathrm{WORD} \times (IDENT \pfun \mathbb{N}))) \pfun \mathbb{N}$$

$$\forall n_1, n_2 : NAME; s_1, s_2 : \mathbf{seq}(IDENT \pfun W); m_1, m_2 : \mathbf{seq}(\mathrm{WORD} \times (IDENT \pfun W)) \bullet$$
$$(n_1, s_1, m_1) \in \mathbf{dom}\ object \land (n_2, s_2, m_2) \in \mathbf{dom}\ object \Rightarrow$$
$$object(n_1, s_1, m_1) = object(n_2, s_2, m_2) \Leftrightarrow n_1 = n_2 \land s_1 = s_2 \land m_1 = m_2$$

由对象类型的定义应能产生相应类型的对象集，因此用于构造对象类型的基本成分应包括产生对象状态序列的状态量定义以及产生方法序列的一组方法定义. 下面我们对 Z 中类型系统进行扩充，由一组对象状态量的类型说明和一组方法的类型说明来构造对象类型. 缩写形式 Σ 和 Π 表示从状态变量名到类型的函数和从方法名到 Σ 的函数：

$$\Sigma == IDENT \pfun TYPE$$
$$\Pi == WORD \pfun \Sigma$$
$$TYPE ::= given \ll TNAME \gg$$
$$\mid powerT \ll TYPE \gg$$
$$\mid tupleT \ll \mathbf{seq}\ TYPE \gg$$
$$\mid schemaT \ll IDENT \pfun TYPE \gg$$
$$\mid objectT \ll \Sigma \times \Pi \gg$$

文献[3]中函数 $Type$ 和 $names$ 分别表示由一组给定类型可构造产生的类型集合以及用于构造某一类型的给定类型集合. COOZ 中将其扩充如下，原有部分用……代替：

$$Type : \mathbb{P}\ NAME \to \mathbb{P}\ TYPE$$
$$names : TYPE \to \mathbb{P}\ NAME$$

……

$$names(objectT(\sigma, \pi)) = \bigcup names(\{\bigcup \{M : \mathbf{ran}\ \pi \bullet \mathbf{ran}\ M\} \cup \mathbf{ran}\ \sigma\})$$

高阶函数 $Carrier$ 表示某一类型在给定映射 $gset$ 下的所有可能取值的集合，$gset$ 表示将给定类型名映射到对应值集的函数. 则对象类型的 $Carrier$ 函数可描述为该对象类型的所有对象的集合. 我们将 $Carrier$ 函数的定义扩充如下，原有部分用……代替.

$$Carrier : (NAME \to \mathbb{N}) \to (TYPE \to \mathbb{N})$$

……

$$Carrier\ gset\ (objectT(\sigma, \pi)) = object(\{Carrier\ gset\ (tupleT\ (IDENT \times LEVEL)) \times$$
$$\mathbf{seq}\ Carrier\ gset(schemaT\ \sigma) \times$$
$$\mathbf{seq}\ \lambda m : \mathbf{dom}\pi \bullet Carrier\ gset(schemaT\pi\ m)\})$$

COOZ 中对象类型是通过 object 模式定义的，根据 object 模式的各组成部分，对象类型的型构由给定类型名、对象常量定义、状态模式的型构和方法的型构组成，其中方法的型构是从方法名到方法模式的映射：

CSIG

$$given : \mathbb{F}\ NAME$$
$$constants : IDENT \pfun TYPE$$
$$ssig : SIG$$
$$msig : WORD \pfun SIG$$

$$ssig.given = given$$
$$\forall S : \mathbf{ran}\ msig \bullet ssig\ subsig\ S \Rightarrow S.given = given$$

对象类型的含义可用如下的 OBJECT 模式表示，它除包含对象类型的型构、对象常量、状态模式和各方法模式的含义外，由于对象状态序列和执行方法序列并非合法状态和所有方法各自的简单组合，而是彼此紧密相关的，所以，还应包括满足对象类型约束条件的相关的状态、方法序列的集合，在下面的 OBJECT 模式用 *states_and_methods* 表示.

OBJECT

$csig : CSIG$
$cmod : \mathbb{P}(IDENT \mapsto \mathbb{P})$
$svar : VARIETY$
$mvar : WORD \mapsto VARIETY$
$states\_and\_methods : \mathbb{P}((\textbf{seq}\ STRUCT) \times \textbf{seq}(WORD \times VARIETY))$

$csig.ssig = svar.sig$
$csig.msig = mvar ; \lambda VARIETY \bullet sig$
$\forall f : cmod \bullet \textbf{dom}\ f = \textbf{dom}\ csig.constants$
$\quad \forall c : \textbf{dom}\ f \bullet f(c) \in Carrier\ gset\ (csig.constant(c))$
$\forall s : states\_and\_methods \bullet (\#\ \text{first}\ s = \#\ \text{second}\ s \vee \#\ \text{first}\ s = \#\ \text{second}\ s + 1)$
$\quad s \in (\textbf{seq}\ svar.models) \times \textbf{seq}(\lambda m : csig.msig \bullet mvar(m))$

4 实　例

由于篇幅限制,下面实例中我们仅给出一个图形系统的部分规格说明. 图形系统一般包括点、线、圆等多种图形元素,我们抽象出各种基本图形元素的父类型 GraphElement 来描述它们所共有的属性和方法. GraphElement 的状态由图形特征点(如直线的两个端点)坐标序列和图形线宽组成,其方法包括平移变换、比例变换、旋转、恢复上一次状态等.

Object *GraphElement*

$COORDINATE == \mathbb{Z} \times \mathbb{Z}$

$points, lastpoints : \textbf{seq}\ COORDINATE$
$linewidth : \mathbb{N}$

$linewidth \geqslant 1$

Init

$width? : \mathbb{N}$

$linewidth = width?$
$\quad \textbf{if}\ width? \geqslant 1$

$\because move$
$\because size$
$\because restore$
……

$restore.\ \textbf{executing}\ ([b, e]) \Rightarrow \neg \textbf{nonexecuting}([0, b])$

GraphElement::*move*

$\Delta points, lastpoints$
$x?, y? : \mathbb{Z}$

$lastpoints' = points$
$\#\ points' = \#\ points$
$\forall i : \mathbb{Z} \bullet i \leqslant \#\ points \wedge i > 0 \Rightarrow points'(i) = (first(points(i)) + x?, second(points(i)) + y?)$

GraphElement::*restore*

$\Delta points$

$points' = lastpoints$

……

5 相关工作的比较

Object-Z[4] 是最初的、也是较为成熟的 Z 面向对象扩充. 它在 Z 中扩充了类的描述机制,并通过时序逻辑描述对象的历史约束. 目前已给出语言的形式化语义,但没有提供模块

机制. Z++[5]在书写风格上不同于Z,而接近于Eiffle,其突出之处在于将方法的前置条件、后置条件分开描述.显式指明方法调用的参数,通过自发内部方法描述实时系统,可选择实时逻辑描述对象的历史性约束. Z++中也没有模块描述机制,另外,它较难于掌握. 其他的Z面向对象扩充如Object-Z的改进版ooZ[6]以及OOZE(object-oriented Z environment)[7], ZEST(Z extended with structuring)[8]等都各具特色,也存在一些不足.

6 总结和进一步工作

本文介绍了我们设计的Z面向对象扩充语言COOZ的语法定义和语义描述. COOZ尽可能吸取现有Z扩充方案的优点,除广泛支持各种面向对象概念外,还为大型软件的规格说明引入模块设施,用支持连续时间的时段演算描述实时系统和并发系统的规格说明,提出了主动模式的概念,并在Z的一些细节方面作了改进. 由于其数学基础较高,形式化方法的应用尤其需要工具的支持. 目前我们进一步的工作主要是对语言的进一步完善和工具的研制:规格说明编辑、显示、查询工具,语法、语义检查工具,精化工具,联机帮助工具等.

致谢 在COOZ的设计和本文的写作过程中,李宣东博士、陈家骏副教授、胡德强、许皓、李勇同学提供了很多参考意见,在此深表感谢!

参考文献

1 Stepney S, Barden R, Cooper D. Object orientation in Z. Springer-Verlag, 1992.
2 Nicholls J. Z Notation version 1.2. BSI Panel IST/5/-/19/2(Z Notation), ISO Panel JTC1/SC22/WG19(Rapporteur Group for Z), 1995.
3 Spivey J M. Understanding Z: a specification language and its formal semantics. London: Cambridge University Press, 1988.
4 Duke R, King P, Rose G A *et al*. The object-Z specification language. In: Korson T, Vaishnavi V, Meyer B eds. Technology of Object-Oriented Languages and Systems: TOOLS 5. Prentice-Hall, 1991. 465~483.
5 Lano K. Z++, an object-oriented extension to Z. In: Nicholls J E ed. Z User Workshop, 5th Z User Meeting, Oxford, Springer-Verlag, 1991. 151~172.
6 Duke D J. Object-oriented formal specification. Dissertation for Doctor Degree, University of Queensland, 1992.
7 Alencar A J, Goguen J A. OOZE: an object oriented Z environment. In: America P ed. ECOOP'91, European Conference on Object-Oriented Programming, Springer-Verlag, 1991. 180~199.
8 Zadeh H B, Stepney S. ZEST — Z Extended with structuring: a user's guide. BT. 7004. 0. 20. 13, British Telecommunications Plc, Logic UK Limited, 1996.

COOZ: COMPLETE OBJECT-ORIENTED EXTENSION TO Z

YUAN Xiaodong ZHENG Guoliang

(*Department of Computer Science and Technology Nanjing University Nanjing* 210093)

Abstract To combine Z notation with object-oriented techniques, this paper adds object type and module mechanism to Z and makes it become object-oriented formal specification language COOZ. The paper includes the design ideas of COOZ, syntax and explanations, formal semantics, a small case and comparison with the related work.

Key words Z language, object-oriented, formal, specification, extension.
Class number TP311