计算机科学 1997 Vol. 24 №. 3

现有Z面向对象扩充语言的比较*)

袁晓东 许 皓 胡德强 李 勇 郑国梁
(南京大学计算机科学与技术系 南京 210093)

摘 要 This paper introduces briefly five influential object-oriented extensions of the specification notation Z: Object-Z, MooZ, ZEST, OOZE and Z++. We analysis and compare their strengths and weaknesses. Based on the weakness of these languages and current needs of software engineering, we point out the future research emphasis.

关键词 Z language, Object-oriented extension, Formal specification

1 引言

软件规格说明的形式化技术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。借助于一定的转换规则和数学证明手段产生应用系统有利于保证其正确性，并具有较高的自动化程度。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，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。目前，在国外已有多种Z的面向对象扩充语言[1]，并且这些语言仍在进一步的研究与开发之中，可见其重要性。那么，这些Z的面向对象扩充语言究竟优劣如何？是否已满足软件工程的需要？其进一步的发展方向又是什么？基于此，本文对五种较有影响的Z面向对象扩充语言进行了分析与比较，针对这些语言的欠缺之处指出进一步的研究方向。这五种语言分别是：Object-Z、MooZ、ZEST、OOZE、Z++。

下文中，对于每一种语言我们首先简要介绍其基本语法结构和各语言成分的含义，然后再对其主要优缺点加以分析，最后对这几种语言从对各种面向对象概念，软件工程概念的支持、形式化技术、使用角度、应用角度等诸多方面进行列表评价。

2 五种Z的面向对象扩充语言

2.1 Object-Z

Object-Z[2]是最初的Z面向对象扩充之一，将一个状态模式和基于此状态的多个操作模式封装起来

*)本文系国家九五攻关项目并受国家自然科学基金资助。

4 重用构件 构件分为实现环境提供的构件和本系统定义的构件。目前的构件重用基本上是重用前者中的构件，特别是界面构件。合理设计类层次能提高后者构件的可重用性。

5 控制对象 实现中，经常采用结构化程序设计和面向对象程序设计相结合的方法，控制对象在多数场合下可以用结构化控制流程来代替。

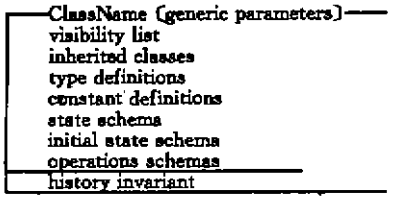
结束语 SOD是结构化思想和面向对象思想的结合，SOD的输入是按功能划分的结构化分析的结果，输出是适合于用OOP实现的对象模型。SOD指导思想从根本上说是面向对象的，属于OOD方法。

本文给出的SOD方法还过于粗糙，要成为实用的一种软件设计方法，还要考虑许多问题，在实际工作中对它不断完善，最后形成一个开发规范。

参考书目

[1] Object_ oriented Modeling and Design, James Rumbangh 等, Prentice-Hall, 1991
[2] Object_ oriented Software Engineering (A Use Case Driven Approach), Ivar Jacobson 等, Addison-wesley, 1994

共同组成类，并能支持各种面向对象机制的描述。Object-Z是目前应用最为广泛、最为成熟的Z面向对象扩充语言。Object-Z中类定义有如下的形式：



visibility list指出外部可见的属性列表，缺省为所有属性均可见。inherited classes指出所继承的父类。type definitions给出局部于该类的类型定义。constant definitions定义了一组常量属性及其关系，这些常量属性的值在每一对象创建时设置并且在对象生命期内不可改变。state schema是一无名模式，说明了一组状态变量及类不变式，状态变量可以说明为类类型，此时该状态变量为一指向对象的指针常量。initial state schemas定义了该类对象创建时可能有的一组属性(包括常量属性和状态变量)初始值。operations schemas定义了一组状态转换模式，即操作模式。history invariant用线性时序逻辑公式定义了一组对象行为约束条件。

Object-Z在Z中扩充了类的描述机制，将类的局部定义、状态模式、对象的初始化模式和操作模式封装在类模式中，支持类之间的继承，允许继承时对操作模式的重定义。其突出之处：一是用线性时序逻辑来描述对象的历史性约束，二是作为类型机制的类模式实例化时产生一指向对象的指针常量，即对象标识。其不足是没有模块机制。OOZ[3]对Object-Z作了一些改进，主要体现在对象方法调用的合理解释、形式化语义、规格说明的精化演算等方面，而主要的语言成分和语法结构未作改变，并且忽略了对象标识的概念。

2.2 MooZ

MooZ[4]在Z中扩充了支持面向对象设计和管理大型规格说明的机制，与Object-Z有一定关联，但它有更为简单的语义模型。一个MooZ规格说明由一组类定义组成，这些类与Eiffel语言中的类定义类似，既可作为类型，也可作为其他类的通用模板。MooZ中类定义的一般格式如下：

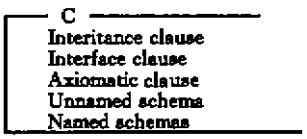
```
Class ⟨Class-name⟩
    givensets ⟨type-name-list⟩
    superclasses ⟨class-reference-list⟩
                 ⟨auxiliary-definitions⟩
    private ⟨definition-name-list⟩
      or
    public ⟨definition-name-list⟩
    constants ⟨axiomatic-description-list⟩
        ⟨auxiliary-definition⟩
    state ⟨anonymous-schema⟩or⟨constraint⟩
        ⟨auxiliary-definitions⟩
    initialstates ⟨schema⟩
              ⟨auxiliary-definitions⟩
    operations ⟨definition-list⟩
EndClass ⟨Class-name⟩.
```

givensets子句给出了给定集合名列表，它的作用是定义参数化类。superclasses子句定义父类列表。private和public子句定义了该类对象外部用户不可见和可见的属性，但所有属性在子类中都是可见的，子类中可改变父类中有关private和public子句的定义。constants子句定义了局限于该类的常量。state子句通过无名状态模式定义了类的状态变量和类不变式。initialstates子句定义了该类对象的初始状态。operations子句通过方法模式定义了类的方法集。

MooZ支持类和对象的封装结构、类属、多继承、对象接口、消息传递等面向对象机制，将Z中的预定义类型及其上的操作以类的形式给出，从而与面向对象思想更为一致。类的状态和操作被当作可被对象处理的消息，模式也可以成为消息，因而增加了消息处理的方式。但MooZ中的对象在语义上相当于一个记录，因而没有历史性约束机制，MooZ也不支持模块设施。

2.3 ZEST

ZEST(Z Extended with structuring)[5]是为满足网络管理和分布式系统的需要而开发的将Z的精确性和面向对象的结构性相结合的语言。其语法借鉴于Object-Z，语义则完全不同。ZEST在Z的类型系统中扩充了类类型(class types)，每个类类型是对一组实例对象的公共行为模板的说明，其语法形式由如下五部分组成：

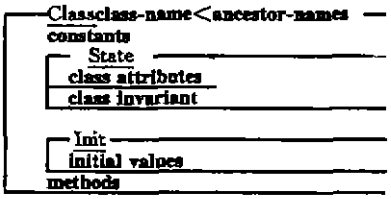


Interitance clause给出该类所继承的父类名称和继承的属性。Interface clause可定义多个接口，每个接口由接口名和外部可见属性组成。Axiomatic clause定义了一组常量属性及其关系，这些常量属性的值在每一对象创建时设置并且在对象生命期内不可改变。Unnamed schema是一无名模式，说明了一组状态变量以及它们之间、它们与常量属性之间的关系。Named schemas是一组命名模式，即操作模式，其中一般包含Init模式用以说明该类新创建实例对象的合法初始值。

ZEST除支持类和对象、封装、多继承、实例化等通常的面向对象概念之外，最显著的特点是一个类可以定义多个不同的对外接口，按照具体情况使用特定的接口名。ZEST另外两个特点是提供了常量属性的定义和子类可以有选择地继承父类的属性。其不足之处是不支持参数化类，也没有引入模块设施。

2.4 OOZE

OOZE(Object Oriented Z Environment)[6]是在Z上发展起来的面向对象广谱语言，不仅支持需求分析和规范，还支持可解释的和可编译的应用程序，整个语言的语义基础是有序代数，在原理上同OBJ3和FOOPS比较接近。OOZE中class的一般格式如下图所示。



上图中class-name<ancestor-names定义了类名及其继承的父类名列表。constants定义了为该类对象所共有的常量。class attributes定义了该类对象的状态变量，状态变量可为类类型。class invariant是类不变式。initial values给出了对象的初始状态。methods是对类中操作的定义。

OOZE除了class结构外，还包括module、Theory、View和Data等多种语言成分。模块定义(modula)用来处理大型的软件系统，整个系统由若干模块组成，每一模块又由功能相关的若干类所组成。OOZE中的模块允许带有参数，称为参数化模块，代之以实参则产生该参数化模块的一个实例。Theory是模块的一种特殊形式，用于定义类型参数的性质，它自身也可以带参数、使用和继承其他模块，通过Theory机制描述模块参数，OOZE提供了特别强有力的参数化程序设计。View给出了一组集合、方法、函数、常量等特征之间的映射关系，用来表示Theory的实参中的特征名与Theory中特征名的对应关系。Data是专门用于定义数据类型的模块，可以带参数或移入其他数据类型。

OOZE是一广谱语言，提供了抽象数据类型、多继承、复杂对象、重载和动态联接等面向对象机制，每个类可有多个实例对象，每一对象都有唯一的对象标识。OOZE中还提供了元类(meta-class)的概念，使得描述和推理更为简单，尤其是对类中一组对象的操作的描述。其不足之处是没有历史性约束描述机制，也没有将对象的内部操作与外部操作分开。

2.5 Z^{++}

Z^{++}[7,8]产生于Esprit I项目REDO中对于大型数据处理系统进行抽象表示的需要。Z^{++}的语法在形式上与Z及其他Z面向对象扩充都有很大的不同，类的定义不再使用模式框的格式，一方面是为了与一些比较好的面向对象语言如Eiffel等相接近，另一方面也便于在类的定义中加进若干子句而不受模式框的限制。Z^{++}中类的语法定义如下：

```
Object-Class::=CLASS Identifier TypeParameters
               [EXTENDS Imported]
               [TYPES Types][FUNCTIONS Axdefs]
               [OWNS Locals]
               [RETURNS Optypes]
               [OPERATIONS Optypes]
               [INVARIANT Predicate]
               [ACTIONS Acts]
               [HISTORY History]
               END CLASS
```

上述定义中，TypeParameters是类属的类型参数，EXTENDS列表是该类所继承的父类。Types和Axdefs是局部类型说明和公式定义。Locals是类的属性说明。OPERATIONS列表是对类中操作的输入参数类型和结果类型的说明。RETURNS列表定义了外部可见的有关对象内部状态的属性和函数的输出类型，它们不改变对象状态。INVARIANT给出了类不变式。ACTIONS列表给出各种对象操作前后置条件的定义。HISTORY谓词用时序逻辑公式或实时逻辑公式说明了允许的对象操作执行顺序。

Z^{++}以在工业化环境中应用为目的，尽可能兼顾通用性和可实现性，支持对系统需求、设计和规格说明的表达。Z^{++}除了支持一般的面向对象概念之外，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，通过自发内部操作处理实时系统的描述，较完善的历史性约束描述机制，便于软件自动化等，因而是一个相当先进的形式化面向对象语言。Z^{++}中还可以进一步扩充模块描述机制，另外较难于掌握，可读性稍差。

3 总结

上文中我们对Z的五种面向对象扩充Object-Z、MooZ、ZEST、OOZE、Z^{++}的语法形式和主要优缺点分别进行了简要介绍。作为总结，表1中从对面向对象概念、软件工程概念的支持、形式化技术、使用角度、应用角度等五个方面对上述五种扩充版本逐项进行比较与评价。

表 1　几种 Z 面向对象扩充的评价与比较

	比较内容	Object-Z	MooZ	ZEST	OOZE	Z++
面向对象概念	封装	支持	支持	支持	支持	支持
	抽象数据类型	不支持	不支持	不支持	支持	不支持
	实例化	支持	支持	支持	支持	支持
	继承	支持	支持	支持	支持	支持
	多继承	支持	支持	支持	支持	支持
	多态性	支持	不支持	支持	支持	受限制
	消息传递	支持	强支持	支持	支持	支持
	复杂对象	支持	支持	支持	支持	支持
	对象标识	指针常量	不支持	常量属性	支持	不支持
	类属	支持	支持	不支持	支持	支持
软件工程角度	模块机制	不支持	正在考虑	不支持	支持	正在考虑
	快速原型	支持	支持	不支持	支持	支持
	并发性	LTL	不支持	不支持	不支持	LTL 或 RTL
形式化技术	新增数学基础	时序逻辑	无	无	有序代数	时序逻辑
	形式化语义	不够完善	正在考虑	部分	较完善	不够完善
	易推理性	好	较好	较好	较好	好
	易精化性	好	较好	较好	较好	一般
使用角度	书写风格	与 Z 类似	Z 和 Eiffel 结合	与 Z 类似	Z 和 OBJ3 结合	与 Eiffel 类似
	掌握难易程度	较易	较易	较易	较难	较难
	工具支持	编辑、语法及类型检查工具	基于超文本的结构化原型开发管理环境 ForMooZ	编辑、语法及类型检查工具包 Zylva	语法及类型检查工具、快速原型工具、定理证明工具	快速原型工具、逆向工程工具
应用情况	主要应用领域	被广泛应用	大型系统	分布式系统	大型系统	实时系统
	主要应用项目	PREMO、mobile phone system	Unix filing System	ISO/TTU PROST-Object	OBJ3 系统	IBM COBOL'74 程序分析工具 REDO

从上表中可以看出，现有的 Z 面向对象扩充语言基本上做到了在形式化规格说明中支持各种面向对象概念的描述，并有一定的工具支持和实际应用。尽管几种 Z 面向对象扩充各有优点，但没有一种能对面向对象方法以及与软件工程相关联的各个方面提供全方位的支持[9]，从目前的实际需要来看，现有的 Z 面向对象扩充有几方面急需改进：①现有语言只能对离散时间进行描述处理，而不是实时系统中的连续时间；②有关语言的形式化语义的研究；③对并发性及其精化、推导的支持；④现有的工具多是供研究用的原型系统，缺乏促进语言推广的商业化工具。

基于这一情形，我们正在进行 COOZ(Complete Object-Oriented Z)[10]的设计、开发工作，其宗旨是实用、简明、全面，在尽可能吸取现有 Z 扩充语言优点的基础上加以改进，以弥补其不足。COOZ 支持封装、继承、类属、多态性和动态定连、对象接口、实例化、对象标识、对象约束、消息传递等所有广泛应用的面向对象概念，为大型软件的规格说明引入模块设施，在提高语法表达能力的同时，给出语言的形式化语义，为研制完善的推理系统和自动化工具打下基础。COOZ 中还设计了简单易用并具备较强表达能力的历史性约束机制和时间描述机制，目前正在进行开发工具和环境的研制。

（参考文献共 10 篇略）