计算机科学 1997Vol. 24№. 4

Z 和 VDM 规格说明的差异比较

A Comparison of the differences between Z and VDM Specification Languages

孙未未　白雪峰

（复旦大学计算机科学系　上海 200433）

摘　要　This paper attempts to provide an understanding of the interesting differences between well-known specification languages--Z and VDM.

关键词　Z,VDM,Differences

实现软件开发过程各阶段的自动化是软件工程的重要目标之一。软件自动化的前提是形式化，包括软件需求规格、软件设计规格和算法描述等的形式化。形式化软件规格说明不仅是对用户需求，也是对软件系统的严格定义，在软件开发中有着相当重要的作用。基于此，为发展我国民族软件产业，与国际标准相衔接，在我们的软件设计中，应坚持必要的规格说明，这样才能生产高质量、长生命期的优秀国产软件。现在已经有了比较流行的 Z 和 VDM 两种面向模型的规格说明语言，准确了解这两种语言的差别，对我们在具体工作中使用将有指导意义。因此，在参考了 I. J. Heyes 等的"Understanding the differences between VDM and Z"之后，形成了本文。希望我们的工作可以对我国软件设计规格化起一点积极作用。

一、起源

VDM 是由在维也纳的 IBM 实验室开发的。

表明，CASE 市场已有巨大增长，然而当前一代 CASE 工具却局限于浅层表达及浅层推理方法，因而 CASE 工具必须进化到或者替代以具有更深层表示、更成熟推理方法的工具，其可用技术来自 AI，形式化方法，程序设计语言理论及计算机科学的其它领域。

软件实践发展迅猛，覆盖了从软件系统的开发、技术的创新、工具的研制等各个方面。至今已形成众多的支持各种软件开发任务的 CASE 工具。然而据统计，在美国只有 2%的组织在软件开发过程中采用了 CASE 方法与工具，仅有 4%的软件厂家是 CASE 用户。这给软件研究软件人员提出一个新的问题。究其原因在于软件构造需要包容个体的知识、经验、技巧，形成一种可以共享知识的合作与协调开发过程，应用于软件开发任务，而现有 CASE 工具存在一些局限性，即没有强调开发群体（包括技术人员，用户，协调人员等）之间的合作式的开发，尤其是在需求分析阶段的计算机支持。MHSC 的研究从 CASE 工具的应用薄弱点分析出发，认为必须在软件需求分析阶段提供有力的支持，不仅可以促进开发人员与用户之间的沟通反馈，为软件的后端开发建立有效的基础，对软件质量进行保证，也为软件从需求定义层到实现层的自动转换提供一种新的途径，并为当前 CASE 工具的研究提供更加完善的支持和指导作用。

目前 CASE 工具仍没有发挥它的全部潜力，因为编码大部分仍是劳力密集型的手工过程。随着编码逐渐自动化，支持软件设计初始阶段的更多成熟工具将变得更为有用。以这些工具发展的快速原型系统将在人的最少指导下，由程序综合系统转换成具有产品级质量的软件。因而，CASE 表示将从软件设计结构和组织的部分表示迈进到形式定义。只要软件生产仍然还是一种劳动密集型活动，质量和生产率就不能期望有阶跃性的提高。软件产品必须通过自动化生产来确保其质量和生产率。CASE 研究将朝着工具、方法的集成，逐步形成一个跨越整个软件生命周期以及跨越多个应用领域的集成开发环境，R. J. Norman 提出的发展趋势引起了 CASE 研究人员的关注，将不断使 CASE 的研究达到应达的目标。（参考文献共 23 篇略）

孙未未　讲师，主要从事形式方法、数据库理论等方向研究。白雪峰　教授　博士生导师。

1961 年，维也纳技术大学的 Heinz Zemanek 教授决定把他的整个小组转向工业领域，他们发现不加入 IBM 就很难得到 IBM 对项目的充分支持，于是他们就加入了 IBM，成立了 IBM 实验室。从 1958 年起，这个小组开发了一些软件，如为 ALGOL 60 程序设计语言创建了一个早期编译器。还是在 60 年代前半期，IBM 雄心勃勃地想开发一个取代 FORTRAN 和 COBOL 的语言。这个语言开始被称为新型程序设计语言(后来由于英国国家物理实验室的反对，改名为现在大家都知道的 PL/1)，人们认为在这项庞大工作的描述中加入一些形式技术对这个语言将有些用处。

在维也纳小组自己工作的基础上，也受 Cal Elgot，Peter Landin，John McCarthy 研究工作的影响，他们发展了 PL/1 运算上的语义定义，称为 ULD-3 (Universal Language Description)。ULD-3 风格的 PL/1 语言描述先后有三个版本，这是一大批文档。形式定义的工作较早揭示了许多语言问题，并且对语言的形式有不可替代的影响。

进入 60 年代末期，人们尝试使用 ULD-3 描述作为编译器设计的基础，发现存在许多问题。操作语义的过细机制特性在一定程度上造成了验证编译算法正确性的复杂性。但是在这个过程中也取得了一定的成绩，发表了一系列文章，描述程序设计语言概念可以对应不同的而且从描述中皆可证明的实现策略。也提出了一些可以简化从语义描述到开发编译器工作的建议，其中的一篇介绍了维也纳指称语义与人们在牛津使用的工具的差异。

在 1971 年至 1975 年间，维也纳小组实际上并没有把精力花费在形式描述上。Cliff Jones 在那时返回到 Hursley 实验室，并在功能语言描述等领域工作，他的一些工作后来也融入了 VDM. 特别是他发表的一篇 Earley 识别器发展的文章，第一次使用了数据具体化的方法。在 1972 年晚些时候，73、74 年维也纳小组有机会研究 PL/1 编译器，他们当然也想用程序设计语言形式描述的方法来构建编译器。PL/1 当时正在通过 ECMA/ANSI 标准，这就是 VDM 最初起源，VDM 的含义是指维也纳开发方法，在 1976 年 IBM 决定改变研究方向后，维也纳实验室把兴趣转移到其他领域，一些科学家便离去了。Dines Bjorner 到了哥本哈根大学然后又去了丹麦技术大学，Peter Lucas 到了美国的 IBM 研究所 Wolfgang Henhapl 到了德国，Cliff Jones 到了 IBM 在布鲁塞尔的欧洲系统研究所(ESRI)。

Cliff Jones 和 Dines Bjorner 继续着他们以前的工作，并写成了一些关于 VDM 语言特点的学术报告，LNCS61 就代表了其中的一部分工作。在 ESRI，Cliff Jones 以编译器研究为基础，推广了 VDM 特点，并写成了现在我们认为是第一本关于 VDM 的书。

Dines Bjorner 小组也使用 VDM 来进行语言描述。他和同事们负责 CHILL 程序设计语言的描述，也在进行 Ada 程序设计语言语义文件的整理。Cliff Jones1979 年至 1981 年在牛津学习。有趣的是，Jean-Raymond 那时也在牛津，他们交换了对描述技术演化的看法，后来这成了著名的 Z 规格语言。

VDM 非语言特性的工作后来由 STL 实验室继续下来，当然部分是由于商业上的推动力。BSI 开始了标准化的工作，但比较艰巨，因为标准委员会必须考虑来自不同用户，如关心 VDM 语言描述特性的用户和只关心 pre/post 条件、数据具体化、操作分解的用户。关于这方面的书和文章也有很多。VDM 的一些观点也影响了其他规格语言，如 RAISE、COLD-K、VVSL 等。下面来谈谈 Z 语言的起源。

尽管在 1979 年以前就已经存在 Z 表示，但在 Jean-Raymond Abrial 于 1979 至 1981 年在牛津大学工作时，仍有许多人参与了 Z 语言早期开发工作。Abrial 在贝尔法斯特讲学时，用与 Steve Schuman、Bertrand Meyer 合著的文章作讲义；受到 Tony Hoare 邀请后，他来到牛津大学，对程序设计研究小组(PRG)做了类似贝尔法斯特的报告，并产生了相当的影响和兴趣。报告中包含了集合论基础、谓词演算等。尽管当时模式表示还没有被很好地发展。

Abrial 几乎与 Cliff Jones 同时在牛津，他们应该有很多机会互相交换研究进展，但后来他们的发展却走了两条不同的路。

Z 是一种表示，并没有正式的方法。Z 的状态作为数学表示是经过慎重考虑的，这使它具有灵活性和开放性。

PRG 最初对 Z 的研究集中在工业应用方面。一个早期重要的应用是 CAVIAR，这是一个来自 STL 基于需求的工业公司的来访人员信息系统。早期还研究过 UNIX 的文件系统。PRG 与其他一些科学家进行合作，其中包括 Ian Hayes，后来他总结了这些项目并写出了有关 Z 的第一本专著[2]。另外一个用 Z 来定义 CICS 的重要例子是由 IBM 开发的交易处理系统。

在开发这个 Z 模式最有别于其他规格说明的特

点时，模式的操作也同时出现。Z模式表示开始被当作结构化大的规格时使用的一项技术，并被看成文本的宏语言来命名和复制大段文本。后来才被人们发现它也可以定义一般的规格组合，即模式的基本操作，进而定义成模式演算等。

在Z发展的初期，PRG以文件的形式描述了Z表示，并在当地散发。完整描述Z的书是A. Sufrin的[3]。后来，又出现了更全面的文章和书籍。

随着Z的工业用户增加，使之成为标准化的要求也在增加。这一工作是由PRG开始倡导的。到1989年，在商业和工业界资助下，Z方法和工作的标准就建立起来了。

Z标准化小组又为标准增加了新内容，不仅是以标准化的格式重写了相应文件，而且为语言增加了新内容[4,5]。新的标准在1992年12月Z用户大会上通过，现在正在使之符合ISO的标准。同时，Z的工业用户正在许多项目中使用Z，开发一些Z的工具，并且在考虑如何把Z和其他表示与方法相结合。有关这部分内容可在Z百科全书[6]中得到。

Z标准化委员会和VDM-SL标准化委员会经常互相交换文档，它们同属BSI标准委员会，都是其分支机构。

二、具体特点差异比较

在对Z与VDM起源、历史有所了解之后，我们来具体比较一下两者的差异，分以下几点：

1. 表示形式

一般看来，VDM规格中多用关键字，而Z规格中多用表格。VDM使用关键字来区分规格说明中不同部分的作用。例如，与一个抽象机相对应的模块有状态(state)和操作(operation)，状态有部件(component)和不变量(invariant)，操作有独立的前置(pre)和后置(post)条件。不同部件因不同的作用而不同，VDM通过关键字具体说明，这些都是VDM语言的一部分。在Z中并没有这样清楚的结构。典型的Z规格由许多定义组成，其中有一些是模式(schema)，即box，模式不仅定义状态，也定义操作。任何一个模式都是由自己明确定义的，尽管从表面上很难看出来。模式也是构建机器状态或操作的基础。这些模式在VDM中没有对应的类型。

为了与VDM一样可以具体化定义形式，还需要考虑具体的编程语言。例如，如果想生成一个Modula代码时，就会发现VDM模块定义与Z模式定义模块十分相似。我们用一个简单的关系数据库NDB[7,8]例子来具体说明。

整个NDB中，所有的状态都包含有关实体(entities)和关系(relations)的信息。下面定义基本集合(sets)。

Eid：数据库中每个实体都有一个唯一的标识，属于Eid集合。Eid是描述集合内部的一个标识，可以透明地理解。

Value：实体都有一个值，属于Value集合。

Estnm：每个实体可属于一个或多个实体集合，其名字属于Estnm集合。

Rnm：数据库是由带名字的一些关系组成，这些名字属于Rnm集合。

现在我们用基本集合构建新类型。NDB是一个二维的关系数据库，每一个元组(tuple)有两个元素：一个from值，一个to值。每个元组包含一对Eid，在VDM中元组类型可如下定义：

```
Tuple :: fv : Eid
         tv : Eid
```

关系就可以这样定义：Relation = Tuple-set

下面再定义关系的几种类型：

Maptp = OneOne(一对一) | OneMany(一对多) | ManyOne(多对一) | ManyMany(多对多)。

关系信息(Rinf)包含了一个关系的信息：除了Tuple集合，Maptp的元素限制了关系的形式，Rinf的tp、r域的一致性可以用如下不变量来说明。

```
Rinf :: tp : Maptp
        r : Relation
        inv(mk-Rinf(tp, r)) ≜
        tp = OneMany ⇒ ∀ t1, t2 ∈ r · t1.tv = t2.tv ⇒ t1.fv = t2.fv ∧
        tp = ManyOne ⇒ ∀ t1, t2 ∈ r · t1.fv = t2.fv ⇒ t1.tv = t2.tv ∧
        tp = OneOne ⇒ ∀ t1, t2 ∈ r · t1.fv = t2.fv ⇒ t1.tv = t2.tv
```

值得注意的是，如果关系rel.r与映射类型rel.tp一致，那么我们说rel∈Rinf。

在NDB中，关系不仅由名字定义，而且也由与它们相关的值的实体集定义(于是一个数据库有两种关系：拥有和在。人拥有汽车，人在房间里)。一个关系的key有三个内容：

```
Rkey :: nm : Rnm
        fs : Esetnm
        ts : Esetnm
```

我们前面的定义与说明可归结所有状态的三个部件：一个实体集映射(esm)，定义了任一实体属于哪一个合法的实体集合；一个映射(em)，包含了每一个定义实体的集合；一个第三映射(rm)，为每个Rkey存贮了相关的Rinf，不变量记录了部件间的一致性条件：

$$\text{Ndb} :: \text{esm} : \text{Esetnm} \xrightarrow{m} \text{Eid\_set}$$

$$\text{em} : \text{Eid} \xrightarrow{m} \text{Value}$$

$rm:Rkey \xrightarrow{m} Rinf$

$inv\ (mk\_Ndb(esm,em,rm)) \xrightarrow{\triangle} dom\ em = \bigcup rng\ esm \wedge \forall\ rk \in dom\ rm(rk.fs,rk.ts) \subseteq dom\ esm \wedge \forall\ mk\_Tuple(fs,ts) \in rm(rk) \cdot fs \in esm(fs) \wedge ts \in esm(ts)$

稍后我们来看看如何把这些定义构建成一个模块。下面我们来讨论用 Z 描述 NDB。显而易见的是所有细节都需考虑，但结构有所不同。规格是由一系列段(section)组成，每一个段都由带着操作的状态部件集合组成。用 Z 来定义规格的方法是用互不相关的部件进行构建。我们必须尽可能地区分系统的不同状态，这样才能使不同状态标识对应不同的操作。

NDB 规格可以分为三部分：①实体和类型或实体集合；②一个单一关系；③多对关系。

最后，我们把这些规格放在一起组成完整的规格。我们不采用通常的 Z 方法，而采用一种方便与 VDM 对比的方法。VDM 的基本集合是[Eid，Esetnm，Value]。前面已经提到过，Esetnm 与 Value 集可以认为是参数，Eid 集合是规格内部使用的。

任何一个实体必须属于一个或多个类型。下面的实体模式把状态的部件和与之相连的不变量组合在一起。

Entities

$esm:Esetnm \overset{m}{\rightarrow} [F\ Eid]$

$em:Eid \overset{m}{\rightarrow} Value$

$dom\ em = \bigcup ran\ esm$

显然我们可以看出 Z 与 VDM 的差别：用横栏(bar)把具体的语法分隔开。接下来我们定义 Esetnm 与 Eid 的二维关系 esm。

Tuple 是一对实体的标识，关系是实体之间的二维关系。

$Tuple == Eid \times Eid$

$Relation == Eid \leftrightarrow Eid$

这与 VDM 形式上是不同的。VDM 用 P(Eid×Eid)形式，真正区别是，在 Z 中关系预先定义，而且也定义了关系上的操作集。

我们再来看看 Z 中如何描述 Maptp，可以设想一个虚拟的标识：Maptp :: OneOne | OneMany | ManyOne | ManyMany。当一个关系建立之后，它的类型必然是这四种之一。产生一个特殊类型的关系后，关系上的操作受到类型的约束。

Rinf

$tp:Maptp$

$r:Relation$

$[tp = OneOne \Rightarrow r \in Eid \overset{m}{\rightarrow} Eid] \wedge$

$[tp = ManyOne \Rightarrow r \in Eid \overset{m}{\rightarrow} Eid] \wedge$

$[tp = OneMany \Rightarrow r^{\sim} \in Eid \overset{m}{\rightarrow} Eid]$

这里如果扩展一下定义就可以得到与 VDM 相同的限制。

再下来我们看看 Z 中如何表示 NDB 中的命名关系。如前所述，数据库中包含了一些带有从 Rnm 集中取出的名字的关系。一个关系是由它的名字和与之相关 from、to 实体集标识的。这造成一些关系有相同的 Rnm，但 from、to 实体集是不同的组合。

Rkey

$nm:Rnm$

$fs,ts:Esetnm$

与关系相关的实体必须属于由关系字(relation key)标识的实体集。

Ndb

Entities

$rm:Rkey$

$\forall\ rk:dom\ rm \cdot$

$\{rk.fs, rk.ts\} \subseteq dom\ esm$

$\forall\ t:rm\ rk \cdot$

$first\ t \in esm[rk.fs] \wedge second\ t \in esm[rk.ts]$

到此为止，Z 版的 NDB 状态部件就已经说明了，其中与 VDM 的差异也是明显的。

2. 初始状态

在 NDB 例子中，VDM 的初始状态是唯一的：$init(ndb) \triangleq ndb = mk\text{-}Ndb(\{\},\{\},\{\})$。Z 中的初始状态是由下列模式定义的：$Ndb\_Init \triangleq [Ndb \mid esm = \{\} \wedge em = \{\} rm = \{\}]$。

3. 操作定义

这里我们不采用更结构化的定义方式，而是忽略一些细节，看看对操作的处理。NDB 最简单的操作是在实体集中增加一个新实体集名，与新名相连的实体集初始值为空。VDM 是如下定义的：

$Addes(es,Esetnm)$

$ext\ wr\ esm:Esetnm \overset{m}{\rightarrow} Eid\_set$

$pre\ es \notin dom\ esm$

$post\ esm = \overleftarrow{esm} \dagger \{es \mapsto \{\}\}$

在 Z 中具体定义如下：

ADDES0

$\Delta Entities$

$es?:Esetnm$

$es? \notin dom\ esm \wedge$

$esm' = esm \cup \{es \mapsto \{\}\} \wedge$

$em' = em$

说明：$\Delta Entities$ 引入了 before 和 after 状态，$\Delta Entities \triangleq Entities \wedge Entities'$。

这里我们看到一个语法上的差异：在 VDM 中，用钩链变量表示 before 状态，用非钩链变量表示 after 状态；而在 Z 中，使用未定义变量表示 before 状态，用事先定义好的变量表示 after 状态。Z 中输入变量名以?结尾，输出变量名以!结尾。除了上面语法

上的差异，VDM 还使用了一个外部语句和一个不同于其他的前置条件。在 Z 中是不存在与外部语句相对应的语句的，即使变量值一直不变，谓词也必须为所有的变量定义最终值。

上面例子中 VDM 与 Z 的差别不很明显，下面我们看另一个操作：删掉一个实体集。

VDM 表示如下：

DELES(es:Esetnm)
ext wr esm:Esetnm $\xrightarrow{m}$ Eid-set
  rd rm:Rkey $\xrightarrow{m}$ Rinf
pre es∈dom esm∧esm(es)={}∧
  ∀ rk∈dom rm·es≠rk.fs∧es≠rk.ts
post esm=(es)⩤esm

Z 表示如下：

DELES0
ΔEntities
es?:Esetnm
es?∈dom esm∧esm(es)={}∧
esm'=(es?)⩤esm∧
rm'=rm

这里 DELES0只定义了实体状态，因此还不能谈论状态部件 rm，为了与 VDM 操作完全等价，需要扩展 DELES0至 NDB 全部的状态，我们通过定义 ΞRM 来完成，ΞRM 可引入 Ndb 的全部状态，同时保持 rm 不变，然后与 DELES0相结合，具体如下：

ΞRM≙[ΔNdb | rm'=rm]
DELES≙DELES0∧ΞRM
对 DELES，前置条件是

pre DELES
Ndb
es?:Esetnm
∃ Ndb'●
  es?∈dom esm∧esm(es)={}∧
  esm'=(es?)⩤esm∧
  rm'=rm

谓词可扩展成：

pre DELES
Ndb
es?:Esetnm
∃ Entities';rm';Rkey→Rinf●
  (∀ rk:dom rm'●
    {rk.fs,rk.ts}⊆dom esm'∧
    (∀ t:(rm'rk).r●
      first t∈esm'(rk.fs)∧
        second t∈esm'(rk.ts)))∧
  es?∈dom esm∧esm(es)={}∧
  esm'=(es?)⩤esm∧
  rm'=rm

也可简化成：

pre DELES
Ndb
es?:Esetnm
es?∈dom esm∧esm(es)={}∧
(∀ rk:dom rm·es?≠rk.fs∧es?≠rk.ts)

4. 一致性检查

在 VDM 中可满足条件限制(satisfiability proof abligation)是标识符主要的一致性检查手段，Z 中的前置条件也有类似的作用，经过计算的前置条件满足标识符的要求。

这是 VDM 与 Z 的一个明显区别：从技术角度看，在开发步骤一开始，就要求使用前置条件；从实际角度看，我们在读非正式的工业规格说明时，规格对所要达到功能的描述是清楚、准确的，只是经常遗漏假设条件。比较好的做法是提示开发者要注意前置条件。对 Z 描述操作的规格来说，只有执行最后一个操作后，前置条件才有意义。VDM 和 Z 在做一致性检查这个功能上是相同的，只是各有不同的实现方法罢了。

5. 异常处理

前置条件是用户必不可少的警告信息；如果初始条件满足前置条件，那么由后置条件定义的行为就可以得到保证。在形式研究中，前置条件可看做是一个限制的条件，只有条件得到满足，操作才会执行。这使得开发者可能忽略某些条件。VDM 中采用例外法。例外条件必须由开发者自己检测和处理。在 VDM 中一个操作规格由一正常的前置条件/后置条件对和一些异常前置条件/后置条件对组成。在 Z 中正常、异常处理皆由同一个模式标识。尽管 VDM 和 Z 看起来在描述异常时有所不同，但在具体语法之下的语义思想还是一致的。

我们来看一下在 Z 中试图加入一个已存在的实体的规格说明。

ESInUse
ΞEntities
es?:Esetnm
es?∈dom esm

这里 ΞEntities 引入了 before 和 after 状态，并由下式具体给出：ΞEntities ≙ [ΔEntities | θ Entities'=θ Entities]。以错误变量为参数的操作是 ADDESX≙ADDES0 ∨ ESInUse。在 VDM 中可通过增加一行 err ESInUse es∈dom esm 来解决。

6. 递归定义

递归定义的方式两者也不同，需要说明的是，在 Z 规格中经常回避使用递归结构，而用一个平铺表示来代替。以一个简单的二叉树为例具体说明。在 Z 中如下定义：

T::=nil | bin node≪T×N×T≫

这里引入了一个新类型 T，是由常量 nil 和一个功能构造来表示的，即给出一个左子树、一个自然数和一个右子树。

在 VDM 中，表示如下：

基于分布式对象的软件构件

Software Component Based on Distributed Object

陈 翀 麦中凡

(北京航空航天大学计算机科学与工程系 北京100083)

摘 要 With the development of network technology, software integration is transforming from single machine and centralized system to network and distributed application. It must ask for software component that is a bases of software integration to meet some new demands: supporting distributed application, open and flexibility and so on. In this paper, firstly, we provide an introduction to software component. Then, we discuss the inevitability of software component supporting distributed application and the advantage of adopting distributed object technology. After analyzing two main distributed object standards, we survey some important research problems demanding prompt solution and consider a few of development directions.

关键词 Software component, Software integration, Distributed object

自从60年代提出软件复用思想以来，就得到了 软件行业的普遍重视，它包括设计的复用(播种复用

---

binnode::l:T

v:N

r:T

T=[binnode]

这个例子有一点语法上的差别：在Z中，T是一个新类型；而在VDM中binnode是一个新类型，T则不是。附带一句，Spivey[9]和Z Base Standard都不是成熟的递归结构，因此可以说，在Z中定义递归不是一件自然的事。

结束语 我们对Z与VDM的差异有了一定的了解，但差别还不止上面提到的这些。有些人认为VDM的工具较多，与对象语言(如PASCAL、C、ADA)等较接近，因此更适合开发一个大的规格；还有人认为Z表述清晰、规范，而且是建立在集合的基础上，因此Z便于使用等。这里牵涉个人的感受、实际应用范围等，我们就不详细讨论了。因为历史原因，使用Z的工业用户比较多一些也是不争的事实。我们希望有更多的人投入到规格说明语言的研究和应用中去，希望我们的工作可以为我国软件设计规格做一点贡献。

参考文献

[1] C. B. Jones, Software Development: A Rigorous Approach, Prestice Hall Intl., 1980

[2] Ian Hayes, editor, Specification Case Studies, Prentice Hall Intl., second edition, 1993

[3] B. A. Sufrin, Notes for a Z handbook, Oxford University, 1988

[4] P. H. B. Gardiner 等, A simpler semantics for Z, Z User Workshop, Springer-Verlag, 1991

[5] J. C. P. Woodcock 等, W: A logic for Z, Z user Workshop, Springer-Verlag, 1992

[6] J. P. Bowen, Select Z bibliography, 同[5]

[7] A. Welsh, The specification, design and implementation of NDB, Master's thesis, University of Manchester, 1982

[8] A. Walshe, NDB, Prentice Hall Intl., 1990

[9] J. M. Spivey, The Z Notation: A Reference Manual, Prentice Hall Intl., second edition, 1992

[10] 施小英等，一种面向对象的形式化规范说明技术--VDM++，上海交通大学学报，30(6)1996

陈 翀 硕士研究生，主要研究领域：软件工程。 麦中凡 教授，主要研究领域：软件工程，程序设计语言，面向对象方法学，数据库等。