可视化的 SOZL 编辑器的设计与实现*)

The Design and Implantation of Visualized SOZL Editor

高晓雷　缪淮扣　陈怡海

(上海大学计算机工程与科学学院　上海200072)

Abstract In this paper, we briefly introduce the characteristics of structured-methods, object-oriented methods and formal methods and integration of these methods. It discusses the structure and object-oriented formal methodology and its supporting language: SOZL. Finally, we introduce the design and key techniques to implement the editor, which support the SOZL method.

Keywords Structured methods, Object-oriented methods, Formal methods, SOZL, Z, OLE, Class

1. 引言

从60年代"软件危机"出现以来，为了提高软件质量和软件开发的效率，人们提出了各种各样的软件开发方法。这些方法大致上可分为三类：结构化方法、面向对象方法和形式方法。在过去的三十多年中，人们在结构化方法和面向对象方法的研究及其应用上做了大量的工作。结构化方法[1]最为成熟，影响也最大。直到现在，仍有60-70%的系统是用结构化方法开发的。面向对象方法[2]近十年发展比较快，大有取代结构化方法的趋势。对于形式方法[3~5]，一直有人在研究，但由于种种原因，投入的人力和物力很不够，还未形成规模，应用也不广泛。最近几年这种情况有所改变，尤其是欧洲阿丽亚娜发射的失败[6]使形式方法的研究逐步受到越来越多的人们的重视。有识之士认识到形式方法是提高软件可靠性、促进程序设计的自动化和提高软件开发效率最有前途、最有希望的方法。

三种软件开发方法各有所长[7]，因此人们不断地探索各种方法的结合，以便扬长补短改进软件开发方法。如结构化与面向对象方法的结合[8]；结构化与形式化方法的结合[9,10]；面向对象与形式化方法的结合[11~13]；Shaoying Liu 博士提出了 SOFL 方法[14]，将结构化方法、面向对象方法和形式化方法进行结合。我们在对各种方法分析、比较的基础上提出了基于 Z 形式表示的结构化面向对象的形式方法 SOZL[15]。

Z 语言是目前使用比较广泛的软件规格说明语言，随着 Z 语言的流行，各种支撑该语言的工具不断出现，比较有影响的有英国 York 大学的 CADiZ，美国 Depaul 大学软件工程研究所的 Z Type Checker(ZTC)，加拿大渥太华的 Odyssey Research Associates 的 Z/EVES，以及 ProofPower，Zola，Formalister，Nipick，ZedBtool 等。这些工具各具特色，如表1所示。

为了支撑 SOZL 应用，我们设计了一个软件开发工具 SOZL Developer Studio，它不仅提供了对 SOZL 的支持，也支持对标准 Z 的开发。

SOZL 编辑器是 SOZL Developer Studio 集成环境中的编辑部分，它是一种基于窗口的、可视化的、所见即所得(WYSIWYG)的图形编辑环境，该编辑器提供一种用户易于输入的图形编辑环境，能同时处理 SOZL 语言及标准 Z 语言的编辑、排版、打印等。本文着重讨论 SOZL 编辑器的设计与实现。

表1　Z 工具的比较

工具名称	WYSIWYG	语法分析	验证
CADiZ	√	√	√
B-Tool	×	√	×
ZTC	×	√	×
Z/EVS	×	√	√
Proofpower	×	√	√
Zola	√	√	√
Formaliser	√	√	×
Nipick	°	°	×
ZedBtool	×	√	×

2　SOZL 简介

为了讨论的方便，我们简要介绍 SOZL，详细介绍请参见参考文[15]。"结构化方法＋面向对象方法＋Z"的软件开发方法将三种方法结合在一起，其基本思想是：在构造一个软件系统时，在前期抽象的分析阶段，使用基于分层数据流图的结构化方法，而在更详细、更具体的后期设计阶段，使用面向对象方法。在系统开发的整个过程，以一种比较实用的方式来应用形式方法。

结构化方法在系统开发的抽象层次上的前期阶段比较适用，因为它为开发者提供了合适的抽象和功能分解的机制。而另一方面，当开发者对系统中数据和对象及其所规定的操作已经清楚地理解后，则在系统构造的后期阶段，对具体的软件行为采用面向对象方法比较合适。形式方法可用来提供软件规格说明、求精和系统各个层次上的验证。

SOZL 的软件开发方法与传统的方法的不同在于将需求分析与非过程设计结合起来，在过程设计阶段采用求精得到抽象程序，最后再由抽象程序进一步求精实现程序模块或采用手工编码。

开发一个软件系统，先构造分层的谓词数据流图(PDFD-Predicate Data Flow Digram)，谓词数据流图是一个有向图，由变量和处理组成。变量是有类型的，用来表示数据流。一个处理对应于 Z 的操作模式中的谓词部分，指明了操作的条件

*)本文受国家自然科学基金项目(60173030)资助。高晓雷　博士生，主要研究方向为软件工程、形式化方法。缪淮扣　博生生导师，主要研究方向为软件工程、形式化方法。陈怡海　博士生，主要研究方向为软件工程、形式化方法。

和操作后状态的变化，称为谓词处理。每一高层的谓词处理分解为一个谓词数据流图，经过分解的谓词数据流图给出了高层谓词处理的功能如何由低层的谓词处理来实现的细节。除了由谓词部分表示的功能规格说明以外，每一个最底层的谓词处理可由某种高级语言以面向对象的方式实现。为了方便起见，在 Z 规格说明语言与高级语言之间加进了中间代码语言；我们考虑的中间代码语言是基于 Dijkstra 的 guarded command 语言的扩充，由它再转换为 C++ 语言程序。选择 guarded command 语言为中间代码语言，可以避免直接从 SOZL 转换为 C++ 的困难，并易于向各种程序设计语言转换。

为了支持 SOZL 方法，我们对标准 Z 语言作如下扩充：一个 SOZL 规格说明可以相应于一个 PDFD 模型的分层序列[1,2]。SOZL 中有两种模块，一种叫 S-module（规格说明模块），对应于 PDFD 分层序列中的一个 PDFD。它可用作 PDFD 的文本界面，并且可以包括常量声明、类型声明、变量声明、谓词处理过程的功能规格说明。另一种模块叫做 T-module（实现模块）对应于 PDFD 分层序列中最底层谓词处理的实现。SOZL 系统结构定义为：

S::=S-module {;S-module|T-module}

模块 S-module 的语法直观表示如下所示：

```
S-module::=Module-name:PDFD-name
           [Module-name,PP-name]
           Module-body
```

一个 S-module 以模块名开始，冒号后紧跟着相关的 PDFD，这里我们用相关的 PDFD 名表示。这样一来，一个 S-module 和对应的 PDFD 之间的关系就建立了。S-module 代表一个高层模块中的某一谓词处理的分解的 PDFD，为了改善 SOZL 系统的可读性，可在 PDFD-name 后写上高层 S-module 名和一个定义在其中的谓词处理名 PP-name，而该谓词处理名 PP-name 表明该低层模块所对应的谓词数据流图 PDFD 是由该谓词处理分解而来。

模块体 Module-body 是由若干段落组成，段落同标准 Z 语言。

模式分为状态模式 StateSchema-Box、初始状态模式 InitSchema-Box 和操作模式 OperateSchema-Box，这里我们对操作模式作如下改进：

```
Operation-
Schema-Box ::=  SchemaName :PP-name
                /Deco-name
                Decl-Part
                Predicate

Deco-name ::= Module-name | T-module
T-module ::= main | function | class
```

一个操作模式由模式名、谓词处理名、模块名、声明部分和谓词部分组成，模块名 Module-name 或是 S-module 名或是 T-module ；谓词处理名 PP-Name 表示操作模式要定义的谓词处理，S-module 名表示该谓词处理分解后得到的 PDFD 所对应的模块（Deco-name）；T-module 表示最底层的谓词处理的分解，它代表名字与谓词处理名相同的实现模块。

一个谓词处理对应一个操作模式，谓词处理的输入输出数据流必须在对应操作模式的声明部分中定义。每一个谓词处理必须对应一个操作模式，每一个操作模式必须在一个 S-module 中定义。

一个模块 S-module 是 PDFD 中的谓词处理的文本定义的封装，而 PDFD 图表示谓词处理之间的关系。当说明一个系统时模块和 PDFD 是互为补充的。一个模块 T-module 用 C++ 中的主函数或函数或类来实现。

3. SOZL 编辑器的设计与实现

整个系统是围绕文件工作的，所以系统的建模也以文件为中心。我们采用面向对象方式建模，所建的对象模型如图1所示。一个 SOZL 文档 SozDoc 是由若干页 Page 组成且至少包含一页；一页由一个或多个块 Block 组成；块有四种类型，分别为数据流图 PDFD、模式 Schema、段 Paragraph 和文本行 TextLine，每种类型均由块派生出；一个段至少包含一个文本行；一个模式由一个或零个模式名 Name、一个或多个声明 Declaretion 和零个或多个谓词组成 Predicte，模式名、声明和谓词均由段派生而来。SOZL 文档与 TXT 文挡由属性类 Doc-ToTxt 所关联。为了节省篇幅，我们只介绍 SozDoc 类和 TextLine 类的具体实现。

·Text 类　产生用于语法分析的文本。语法分析的文本是纯文本文件，所以该类的实现可以采用 MFC 所提供的 CFile 类，在需要时声明一个 CFile 类的实例即可。

·SozToText 类　是类 SozDoc 和类 Text 的关联，我们利用 VC++ 自动生成的框架类 CSozDoc 作为类 SozToText 的实现，在类 CSozDoc 增加两个属性成员 soz、text 和一个成员函数 SozToText，该函数负责将 soz 文挡转换成易于语法分析的 text 文本。

·SozDoc 类　PageTotal 记录文档的总的页数，PageSize 用来记录文档中页的尺寸，Pages 用来保存指向页对象的指针；其主要操作有生成一个新页 SetNewPage()、在指定位置插入一页 InsertPage(int i,Page* InsertPage)；删除指定页 DeletePage(int i)；定位光标位置 Mouse(CDC* pDC,CPoint point)；显示文件 FileOut()；文件存储 Serialize(CArchive& ar)其具体实现叙述如下：

SozDoc 类使用继承 MFC 的 CObject 类创建，使用继承 CObject 类创建的类能够充分利用 MFC 所提供的结构化存贮方法。使用 CObArray 类实例可以存放任何指向继承 CObject 类的对象指针，从而简化编程，减少工作量。由于 Page 类、Block 类、Pragraph 类和 Schema 类的实现与类 SozDoc 类基本上相似，此处不再赘述。

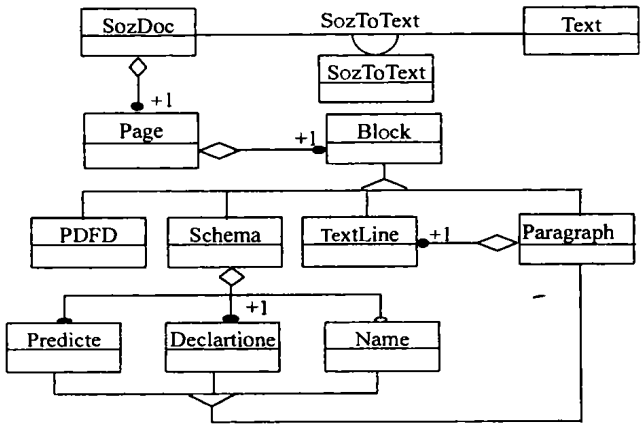


图1　SOZL 对象模型

·Page 类　BlockTotal 页中包含的块数；Blocks 用于存放指向 Block 对象的指针，由于类 PDFD、类 Schema、类 Paragraph 和类 TextLine 都是由类 Block 派生而来，因此，Blocks

也可用于存放指向上述四种类实例的指针。

·Block 类　块类型 BlockType 有三种，它们是 PDFD 型、Schema 型和 Paragraph 型；BlockSize 是块的大小，用于限定块中内容的显示区域。

·Pragraph 类　一个段落 Paragraph 是由若干文本行 Textline 组成，LineTotal 记录文本行数；Textlines 用于存放指向 TextLine 对象的指针。当一行中字符串的宽度大于块的宽度时，应进行行分裂，以便显示时不会超出指定区域；当一行中字符串的宽度小于块的宽度并且该行后仍有行存在时，应进行行合并，以便能显示正确、完整的段落。

·Name 类、Declare 类和 Predict 类　它们都是由类 Paragraph 派生而来，可完全继承父类的属性和方法，它们和父类 Paragraph 的唯一区别是矩形的起始点不同，区别如图2所示。因此，实现时只须要在各类的析构函数中重构矩形的起始点，其他均继承父类。

模式由模式名、声明部分、谓词部分和模式框组成，如图2所示。模式名是指向 Name 类的实例指针，声明部分由若干指向 Declare 类的实例指针组成，谓词部分由若干指向 Predicte 类的实例指针组成，它们分别存放在指针数组 SchemaName、SchemaDeclaretions 和 SchemaPredictes 中；模式框用八个坐标点表示，八个坐标点记录在整型数组中 SchemaPoint[8]。

·PDFD 类　使用 Windows 和 MFC 提供的 OLE 技术来实现。只要用户拥有任何一个能够作为服务器使用的绘图程序，就可以绘出所要的任何图形，用户也可在不知道我们编辑器的源码情况下开发自己的绘图程序，只要该程序支持 OLE 技术。

·TextLine 类　是构造整个 SOZL 文档的基础，是类 Schema 和类 Paragraph 的成员，而类 Paragraph 又是类 Name、类 Declare 和类 Predict 的父类，因此，TextLine 类是整个类结构的基础和关键。

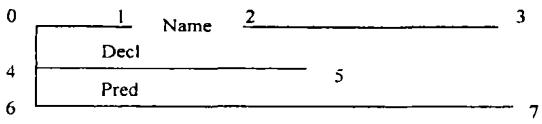


图2　模式表示

每一文本行用 MFC 提供的类 CString 的实例 m_TextLine 来保存，m_TextLine 中存放的字符数受显示区域的限制，如图3所示，显示区域用 MFC 提供的矩形类 CRect 的实例 BlockSize 来表示，字符串 m_TextLine 中字符的宽度之和应小于或等于矩形 BlockSize 的宽度；Windows 环境下是图形方式显示字符，字符的宽度以像素点数表示，由于字符的宽度不尽相同，字符所占像素点数也就不同；如字符 W 与字符 L 的宽度显然不同，字符 W 所占像素点数大于字符 L 所占像素点数；因此，字符串 m_TextLine 中字符数无法事先确定的；为了使字符串 m_TextLine 的宽度不超过矩形 BlockSize 的宽度，我们利用 Windows 提供的计算字符串宽度的 API 函数计算 m_TextLine 的宽度，以便决定 m_TextLine 中允许的字符数个数。

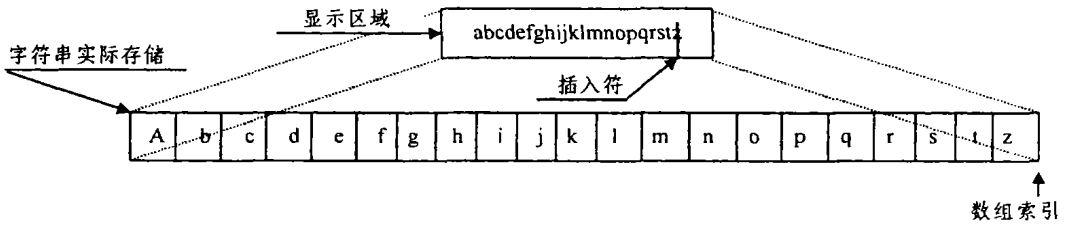
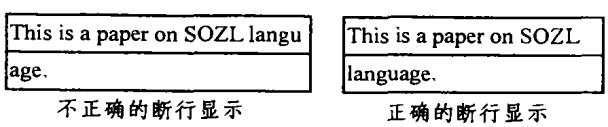


图3　字符的存储与显示

插入符 Caret 是一个共享资源，我们将其定义为全局变量；插入符的作用是提示用户插入的字符在屏幕上的显示位置；如果字符是等宽的，则每插入一个字符，插入符 Caret 的 x 坐标增加一个常量即可。然而，大部分 Windows 的字符都是不等宽的，无法使用上述方法定位插入符 Caret，为此我们设计了一个定位插入符 Caret 的函数 StringWiden(CDC* pDC, CString st)，其基本思想是利用插入符 Caret 与数组索引 CurrentIndex 之间的关系：插入符 Caret 的 x 坐标等于数组索引 CurrentIndex 左边字符串的宽度加上页左边距(显示起始点)。

Caret .x=StringWiden(pDC, st.Left(CurrentIndex))
+LeftMargin

英文以词作为语义单位，如果需要换行易产生词的不完整错误，如图4所示：



不正确的断行显示　　正确的断行显示

图4　断行示意图

为了解决上述错误，我们设计一个断行函数 FormatTextLine；其算法如下所示：

```
如果(StringWiden(pDC,m_TextLine)> BlockSize.right)
{
  Temp = m_TextLine 中左起最后一个标点符号以后的字符;
  m_TextLine = m_TextLine 去掉 Temp ;
  生成一个新行;
  拷贝 Temp 到新行中;
  重定位插入符 Caret;
}
```

4　SOZL 编辑器关键技术的实现

·特殊字符的实现

由于 SOZL 语言中有许多特殊符号如→⇒⇔∀ 等，给用户输入造成很多困难。尽管 windows 系统中自带一些特除符号，但有些特除符号是没有的，且其输入方法也是笨拙的；由于我们的编辑器不仅要处理文本，还要对文本进行语法分析、类型检查、求精等操作，而这些操作又需要知道符号的具体编码，这在 windows 系统中显然是无法办到的。如何实现对特殊符号的输入输出有二种方案可选。一种是使用位图，将符号

画出来，一种是使用 TrueType 字体。由于位图与设备相关性较大，很难达到所见即所得的效果，因此我们选择了字符编码方式。然而英文字符的编码是8位编码，无法满足一些特殊符号的编码，所以我们选择了汉字编码方式。首先，我们利用中文 windows 95中的造字程序创建所需符号并编码，比如⇒，编码为 AAC1，则在中文输入区位码时，键入 AAC1即可得到的输入；由于区位码的输入要记住每一个字符的编码，因此，该输入法使用起来并不方便，如何解决特殊符号的输入呢，我们采用了工具箱式的输入法。只要在工具箱中用鼠标选取想要输入的字符即可完成输入。其实现是利用 MFC 中提供的工具条，工具条只能显示一行，因为 SOZL 中特殊字符很多，一行中无法全部显示出来，因此我们对 MFC 中的工具条进行修改，将工具条改成工具箱，使其能多行显示。

·上标与下标的实现

上标与下标的实现是利用图形显示的特点，通过改变字体的逻辑大小，重新定位字符显示的位置而产生的特殊显示效果。上标在数组中以\a 开始，以\a 结束；下标以\b 开始，以\b 结束。例如 h^2对应 h\a2\a 和 k_n 对应 k\bn\b，其存储与显示关系如图5所示。

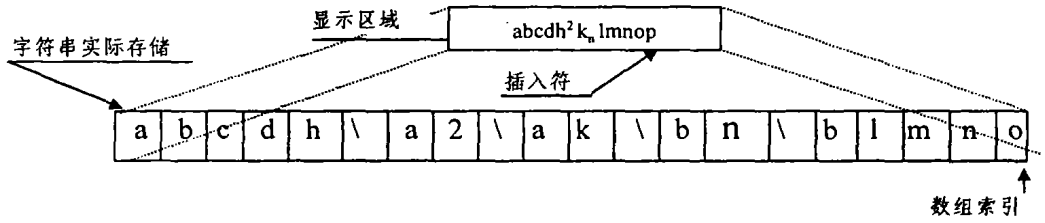


图5　上标与下标的处理

·汉字的处理

如上所述，在处理特殊字符时使用了双字节编码，而双字节编码也正是汉字编码，所以处理好汉字也就是处理好特殊字符；另外作为可交流文本汉字也是必不可少的，尤其是和国内的用户打交道时，汉字文本更精确。因此，汉字的处理也是非常重要的一环。

在 TextLine 类中我们使用 MFC 类 CString 作为类 TextLine 中具体存贮字符的对象。CString 类具有显示汉字及大小写字符的功能，使得输出显示相对容易一些。但是，对于光标的移动，字符的删除等都带来了问题。因为一个汉字需 CString 中两个存贮单元，因此如果只删除一个字符，就易产生众所周知的"半个汉字"问题，造成屏幕显示混乱。因此，要将汉字和一般字符分别对待。当字符为普通英文字符号按照一个存贮单元处理，如为汉字，则按二个单元处理。判断存贮单元中存贮的是汉字还是英文字符成了问题的关键。由于英文字符号按照8位编码，其中最高位总为0，而汉字则16位编码，即二个8位编码代表一个汉字，每个8位的最高位总为1，以示与英文字符的区别，因此我们判别存贮单元中的最高位来决定是汉字还是英文字符。具体实现我们用存贮单元中的值与128按位与，如相与结果为128表示最高位为1，即存贮单元中是汉字。

·关于注释

尽管 SOZL 语言本身没有注释功能，我们的编辑器却能提供注释的功能。我们之所以在 SOZL 语法中没有给出注释，主要是由于注释影响文本的美观，因为我们总要使用一些关键字来表示注释的开始和结束。在编辑器中我们使用 OLE 技术，可在想要注释的任何地方加上注释，文本既美观又不影响语法分析、求精、测试和转换。

小结　本文较详细介绍了 SOZL 编辑器的建模、设计和实现，对一些关键技术如特殊符号的实现、汉字的处理、如何显示上标、下标以及利用 OLE 技术实现数据流图和注释作了详尽的剖析。由于篇幅所限，有些技术不能完全展开讨论，但愿我们的方法能起到抛砖引玉的效果，促使 Z 语言编辑器的不断发展。

参 考 文 献

1　Yourdon E. Modern Structured Analysis. Prentice Hall，1989

2　王博，晓龙. 面向对象的建模设计技术与方法. 学苑出版社，1993

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

4　Spivey J M. The Z Notation：a Reference Manual，second edition，London，Prentice Hall，1990

5　Diller A Z. An Introduction to Formal Methods，London：John Wiley & Sons，1990

6　李未. 欧洲程序设计方法研究的三个动向. 计算机科学，97. 1

7　缪淮扣，高晓雷. 结构化方法、面向对象方法和形式化方法的比较及结合. 计算机工程与科学，1999，21(4)

8　陈涵生，徐智晨，沈怡. 面向对象开发技术及应用. 上海科技文献出版社，1995. 3

9　Woodcock j c p . Struturing Specifications in Z. Software Engineering Journal，4(1)

10　Liu Shaoying . A Formal Requirements Specification Method Based on Data Flow Analysis. The Journal of Systems and Software，1993，21：141～149

11　Lano K Z++. An Object-Orientated Extension To Z，Z User Workshop，Oxford 1990，Workshops in Computing，Springer-Verlag，Dec. 1990. 151～172

12　袁晓东. al，Z 的面向对象扩充 COOZ 的设计. 软件学报，97. 9

13　李刚，朱关铭，童兆贞. 形式方法与面向对象方法的结合探讨. 计算机工程，1998(1)

14　Liu Shaoying. Structured Methodology+Object-Oriented Methodology+Formal Methods：Methodology of SOFL

15　Gao Xiaolei，Miao Huaikou，Chen Yihai. SOZL language：A new software development methodology. In：16 IFIP World Computer Congress，Beijing，China，Aug. 2000. 21～25