计算机科学 1996 Vol. 23№. 1

图形化对象式需求定义语言 NDORL

张家重 王志坚 陶先平 吕 建 陈道蓄 朱 鸿
(南京大学计算机软件研究所 南京 210093)

摘 要 NDORL 语言属于一种半形式化需求定义语言，采用了面向对象方法，能够支持软件需求构模的图形化需求定义。其特点是：表达能力强、形象直观、易理解、易维护、便于复用。本文主要讨论 NDORL 语言。

关键词 软件需求定义，软件需求定义语言，面向对象，图形化。

一、引 言

1980 年代以来，我们对软件自动化进行了一系列的研究，先后设计了软件设计规约语言 GSPEC 和软件功能规约语言 FGSPEC，并研制了多个软件自动化系统。对从软件功能规约到软件设计规约，直到可执行代码全过程转换的自动化技术进行了系统的研究，并取得了一定进展。为了进一步研究软件自动化中从"非形式"软件需求定义到"形式"功能规约的自动转换技术，我们提出了一个对象式软件需求模型 NDHORM，设计了一种图形化对象式软件需求定义语言 NDORL，并以此为基础研制了对象式软件需求自动化系统 NDORAS。本文主要介绍 NDORL 语言。

图形化对象式需求定义语言 NDORL 是一种基于面向对象方法，能够支持软件需求构模的图形化需求定义语言，属于半形式化需求定义语言。它以问题领域中的对象及其相互关系为核心，刻画现实世界问题模型，从而得到能够反映与模拟问题结构的需求定义。

NDORL 主要具有如下特点：(1)采用面向对象方法，精确刻画对象式软件需求模型 NDHORM；(2)提供了行为制导的问题对象识别方法，以及由抽象到具体、逐步精化的层次构模成分；(3)具有较强的表达能力，并提供了机器支撑；(4)形象直观，易理解，易维护，便于复用。

二、语言的描述

2.1 语法的描述

语法描述采用扩充的 BNF 形式体系。它是用来描述图形化对象式需求定义语言 NDORL 的语法的语言，称为元语言。元语言由连接词、汉字列、以及元语言公式(或称为语法规则)组成。

连接词有：

::= 定义为

| 或

[ ] 表示被括内容是可选的

{ } 视被括内容为一个整体

{ }* 表示被括内容可选，也可重复有限多次

{ }+ 表示被括内容必选，且可重复有限多次

{ }" 表示被括内容的有限(无序)集合。

汉字列是对有关语法单位起限定作用的词句，其形式为"一串汉字"。

元语言公式，也称产生式，其形式为

$$S::=E$$

其中 S 为语法单位名，用〈汉字列〉表示。E 是语法表达式，形为

$$T_1|T_2|\cdots|T_n \qquad (n>=1)$$

其中 $T_i(i=1,2,\cdots,n)$ 为语法项。形式为

$$F_1F_2\cdots F_m \qquad (m>=1)$$

其中 $F_j(j=1,2,\cdots,m)$ 为语法因式。语法因式具有以下几种形式：

1)N 2)$C_1C_2\cdots C_t(t>=1)$ 其中 N 为语法单位名，如，〈对象〉。$C_i(i=1,2,\cdots,t)$ 为字符。语法项中两个语法单位名之间用空格隔开。

3)[E] 表示语法表达式，是空|E 的缩写。其中"空"指什么也不出现，E 则表示语法表达式。

4){E} 将语法表达式 E 看作一个整体。

5){E}* 表示语法表达式，是空|E|EE|EEE|…的缩写。其中"空"指什么也不出现。

6){E}+ 表示语法表达式，是 E|EE|EEE|…的缩写。

7)〈E〉* 表示语法表达式，是由E代表的所有语法项构成集合的编写。

上述语法规则事实上是局部语法规则，不能用这些局部语法规则刻画的语法称之为全程语法。其描述采用自然语言。

2.2 语义的描述

语义指需求定义语言的含义，其描述采用自然语言。

三、词法结构

3.1 字符集

字符集包括语言使用的全部字符。

〈字符〉::=〈字母〉|〈数字〉|〈特定字符〉|〈空格字符〉|〈汉字〉

〈字母〉::=a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z|A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

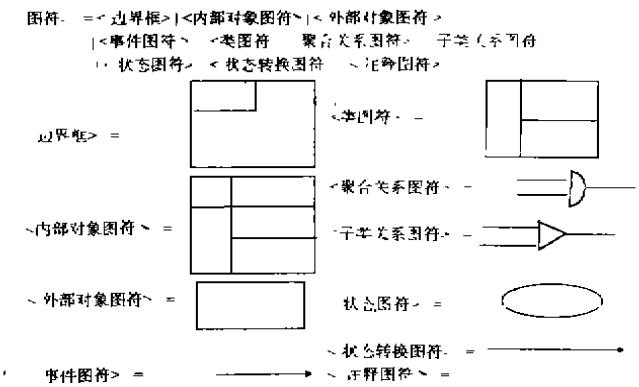
〈数字〉::=0|1|2|3|4|5|6|7|8|9

〈特定字符〉::=→|*|.|(|)|,|'|;|{|}|/|>|<|=|/|[|]|%|@|&|"|"|!|^|∧|∨

〈空格字符〉::=space

〈汉字〉::="汉语文字"

3.2 图符集



3.3 词法单位

词法单位是软件需求定义语言中具有确定含义的最小单位。

〈词法单位〉::=〈标识符〉|〈无正负号数〉|〈保留字〉|〈界限符〉|〈字符串〉

1)标识符

〈标识符〉::=〈字母〉|〈汉字〉|〈标识符〉〈字母〉|〈标识符〉〈汉字〉|〈标识符〉〈数字〉

2)无正负号数 用于构作常量，表达式等。

〈无正负号数〉::=〈无正负号整数〉|〈无正负号整数〉〈小数部分〉|〈无正负号整数〉〈指数部分〉

〈无正负号整数〉::=〈数字〉|〈无正负号整数〉〈数字〉

〈小数部分〉::=.〈无正负号整数〉

〈指数部分〉::=E+〈无正负号整数〉|E-〈无正负号整数〉

3)保留字

〈保留字〉::=〈标题字〉|〈操作字〉|〈其它字〉|〈注释〉

〈标题字〉::=**requirements**|系统名称|需求简介|非功能性需求|功能性需求|其它|应用范围|功能简介|常识|一般约束|假设和依赖关系|软件接口|硬件接口|通信接口|设计约束|安全性|易维护性|易移植性|其它约束|对象关系图|类关系图|状态图|对象/类字典|解释|类字典|对象字典|**end**

〈操作字〉::=**grt|eq|concat|isempty|head|tail|length|reverse|contain|connect|select|append|position|perm|delete|change|insert|number|take|remain|incorp|inter|union|differ|subset**

〈其它字〉::=**system|external|object|class|input|output|constraint|functionality|descrtption|empty|true|false|mod|div|attribute|behavior|pre|post|endclass|var|const|int|real|set|seq|enum|not|and|or|imply|iff|forall|exist|unique**

4)界限符

〈界限符〉::=;|.|,|'|(|)|[|]|%|,|"|"|{|}|=|<|>

5)字符串

〈字符串〉::="{〈字符〉}*"

6)注释

〈注释〉::={〈注释图符〉〈注释正文〉}|//〈字符串〉//

〈注释正文〉::=〈字符串〉

四、数据类型

本语言中，数据类型定义为三元组，其中一是值集，二是操作集，三是类型名，如下所提供的数据类型均给出这三部分。数据类型分为两类：基本数据类型和指定数据类型。

4.1 基本数据类型

〈基本数据类型〉::=〈简单类型〉|〈复合类型〉|〈类类型〉

〈简单类型〉::=〈整型〉|实型〉|〈布尔型〉|〈字符型〉|〈字符串型〉

〈复合类型〉::=〈序列型〉|〈集合型〉|〈枚举型〉|〈记录型〉

〈类类型〉::="表达类型的"〈类名〉

〈简单类型名〉::=int|real|bool|char|string

其中,类类型即将类也看作是类型,可以把变量定义为具有某个类所表达的类型。类类型的名是类名,值集是其描述的对象集,操作集是类中的行为集合,类的具体定义参见7.2及9.1,9.2。

4.1.1 布尔型

类型名:bool

值集:true,false

操作集:and,or,not

4.1.2 整型

类型名:int

值集:目标机所允许的所有整数,即

〈整数〉::=〈无正负号整数〉|-〈无正负号整数〉

操作集:>,<,>=,<=,<>,+,-,*,div,mod,-

4.1.3 实型

类型名:real

值集:目标机所允许的所有实数,即

〈实数〉::="整数以外的"〈数〉

〈数〉::=〈无正负号数〉|+〈无正负号数〉|-〈无正负号数〉

操作集:>,<,>=,<=,=,<>,+,-,*,div,-

4.1.4 字符型

类型名:char

值集:所有字符构成的集合。

操作集:grt(c1,c2):c1的序号大于c2的序号时为true,否则为false。eq(c1,c2):c1的序号等于c2的序号时为true,否则为false。

4.1.5 字符串型

类型名:string

值集:目标机所允许的所有字符串所构成的集合。

操作集:concat(x,y):将字符串x,y并置。

4.1.6 序列型

类型名:借助序列型说明所定义的序列型名,其中:

〈序列型说明〉::=〈序列型名〉=seq(〈项类型名〉)

〈序列型名〉::="类型"(标识符)

〈项类型名〉::=〈简单类型名〉|〈类类型〉|〈指定集合类型〉|〈枚举类型〉

值集:由有限多个序列值所构成的集合,每个序列值又是由有限多个相应项类型值集中的项构成的序列。序列可以为空,用empty或( )表示。

序列构造符:序列构造符用于指出一个序列值是由哪些项构成,并约定它们之间的次序关系,但它并不是常量。语法描述如下:

〈序列构造符〉::=["("〈项〉{,〈项〉}*")"]

〈项〉::=〈表达式〉

操作集:

(1)isempty(s):当序列s为empty时,其结果为true,否则为false。

head(s):当序列s非空时,取出s中的第一项,否则,其结果为空。

tail(s):截取序列s中第一项除外的部分,当s为empty时其结果为empty。

length(s):求序列s的长度,即s中项的个数。

reverse(s):将序列s倒序。

(2)contain(s,t):当t为序列s的项时,其结果为true,否则为false。

connect(s1,s2):将序列s1的各项接以序列s2的各项,构成一个新的序列。

append(t,s):将t添加到序列s中,成为新序列的首项。

position(t,s):求t在序列s中出现的最前位置,如t不在s中出现,其结果为0。

select(s,i):选出序列s的第i项。

perm(s1,s2):当序列s2是序列s1的置换时,其结果为true,否则为false。

delete(t,s):当元素t在序列s中出现时,删除其首次出现,否则为s不变。

(3)change(s,i,t):用t取代序列s中的第i项,当i大于序列s的长度时,s不变。

insert(s,i,t):在序列s的第i-1项后插入t,成为新序列的第i项,原序列的第i项成为第i+1项。

4.1.7 集合型

类型名:借助集合型说明所定义的集合型名。其中:

〈集合型说明〉::=〈集合型名〉=set(〈基类型名〉)

〈集合型名〉::="类型"〈标识符〉

〈基类型名〉::=〈简单类型名〉|〈类类型〉|〈指定集合类型〉|〈枚举类型〉

值集：相应基类型值集的所有子集构成的集合，即基类型值集的幂集，empty 或{}表示空集合。

集合构造符：集合构造符用于指出一个集合类型数据的值是由哪些元素所构成的，但它并不是常量。语法描述为：

〈集合构造符〉::=["{"〈元素〉{,〈元素〉}*"}"]

〈元素〉::=〈表达式〉[..〈表达式〉]

操作集：

(1)**isempty(s)**：当集合 s 为 empty 时，其结果为 true，否则为 false。

number(s)：求集合 s 中元素的个数。

take(s)：取出集合 s 中的任一元素，当 s 为空时，其结果为空。

remain(s)：去掉集合 s 中的元素 take(s)，当 s 为空时，其结果为空。

(2)**contain(s,e)**：当集合 s 中含有元素 e 时，其结果为 true，否则为 false。

incorp(s,e)：把元素 e 添加到集合 s 中去。

inter(s1,s2)：求集合 s1和集合 s2的交集。

union(s1,s2)：求集合 s1和集合 s2的并集。

differ(s1,s2)：求集合 s1和集合 s2的差集，即从集合 s1中去掉所有集合 s2中的元素。

subset(s1,s2)：当集合 s1是集合 s2的子集时，其结果为 true，否则为 false。

4.1.8 枚举型 枚举型是由有限多个枚举值(即枚举符号)构成的集合，并且用枚举型说明来引进。其语法描述为：

〈枚举型说明〉::=〈枚举型名〉=enum(〈枚举符号表〉)

〈枚举符号表〉::=〈枚举符号〉{,〈枚举符号〉}*

〈枚举型名〉::=〈标识符〉

〈枚举符号〉::=〈标识符〉

每个枚举型说明用来引进一个类型名，它表示一个枚举型，用括号括起来的枚举符号表由一个枚举符号或彼此间用逗号隔开的多个枚举符号组成，这些枚举符号代表该枚举型对象所有可取的值。

值集：所有枚举符号构成的集合。

操作集：=等于，<>不等于，

4.2 指定集合类型

在某些情况下，对于一实际的值集，人们能够识别或确定其值的范围，但不能或不需要给出其具体的内部结构，于是 NDORL 引入了指定集合类型，以表示上述情况。

类型名：借助指定集合类型说明定义的名字，指定集合类型说明的语法描述为：

〈指定集合类型说明〉::=〈指定类型〉

〈指定类型〉::="["〈类型名〉{,〈类型名〉}*"]"

〈类型名〉::=〈标识符〉

对于指定类型，每个指定集合类型说明引进一个或多个类型名，表示一个或多个指定集合类型。如，[COMPANY]，[PERSON]分别定义了两个指定集合类型 COMPANY 和 PERSON，它们也可以采用复合定义的方式，等价于[COMPANY，PERSON]。

值集：指定的值集。

操作集：同集合类型的操作集。

五、常量、变量和表达式

5.1 常量

按照名和值是否一致的角度来区别，常量分为字面常量和非字面常量，如，2，3为字面常量，X 为非字面常量。字面常量包括无正负号数，字符，字符串，布尔值和枚举值等，它可在需求定义中直接使用。非字面常量是用常量名表示的常量，它通过常量说明或有关构造符来定义，以此确定非字面常量的类型和值。事实上，常量名也即常量的助记符，它由用户自由确定，用助记名表示变量比直接使用常量的值更为方便，也易于理解和维护。常量说明的语法描述为：

〈常量说明〉::=const〈常量定义表〉

〈常量定义表〉::=〈常量名〉=〈常量值〉{;〈常量名〉=〈常量值〉}*

〈常量名〉::=〈标识符〉

〈常量值〉::=〈常量表达式〉

〈常量表达式〉::=〈表达式〉

此处作为常量表达式的表达式指的是由字面常量、已定义的其它常量名所构成的表达式。表达式的定义见5.3。

5.2 变量

在需求定义中可以取不同值的量称为变量，每个变量必须联系一个变量名，其通常为标识符，并以该变量名代表其数据值。本语言中的变量是通过显式变量说明来定义的变量，即静态变量。变量说明用来对变量命名，并规定其取值范围。变量说明的语法

描述为：

〈变量说明〉::=var〈变量名〉{,〈变量名〉}*:〈类型名〉

〈变量名〉::=〈标识符〉

其中，一个变量名只能被说明一次，不能重复说明多次；当有多个变量说明时，相邻两个变量说明之间需要用分号";"隔开。

5.3 表达式

表达式是语言中最基本的组成部分，它表示一种求值规则，由三部分组成：运算对象、运算符、圆括号。运算符和圆括号指示表达式的求值规律。

一般来说，表达式是依参与计算的运算符和运算对象从左到右，按先乘除后加减等规律求值，以得到一个表达式的值。但括号能够改变求值顺序。

每个表达式有一个值和一个类型。表达式的类型由其中的运算决定，但未必和运算对象的类型相同。表达式的语法描述如下：

〈表达式〉::=〈简单表达式〉[〈关系运算符〉〈简单表达式〉]

〈简单表达式〉::=[〈单目运算符〉]〈项〉{〈加减法运算符〉〈项〉}*

〈项〉::=〈因式〉{乘除法运算符〉〈因式〉}

〈因式〉::=〈变量〉|〈无正负号常量〉|〈界限符〉|函数命名符〉|〈集合构造符〉|〈表达式〉|not〈因式〉

〈无正负号常量〉::=〈无正负号数〉|〈字符串〉|〈常量标识符〉

〈常量标识符〉::=〈标识符〉

〈函数命名符〉::=〈函数标识符〉[〈实参表〉]

〈函数标识符〉::=〈函数名〉

〈函数名〉::=〈操作字〉

〈实参表〉::=(〈实参〉{,〈实参〉}*)

〈实参〉::=〈表达式〉|〈变量〉

〈关系运算符〉::=＝|<>|<|<=|>|>=

〈加减法运算符〉::=＋|－|∨

〈单目运算符〉::=＋|－

〈乘除法运算符〉::=＊|/|mod|∧

六、对象关系图

6.1 **对象关系图**(Object-Relationship Diagrams，ORD)

对象关系图是对象关系模型的图形化表示，主要刻画问题领域中对象的动态行为及其相互关系。其中对象间的相互关系说明了对象之间的交互通信及消息传递情况，也就是，对象的发送行为和接收行为。为了便于问题分析，我们称对象之间消息的一次传递为一个事件。对于消息的发送对象而言，该事件为请求事件；对于消息的接收对象而言，该事件为接收事件。事实上，对象关系模型的形成过程是一个事件（或行为）制导的问题对象的识别过程，这也就为用户提供了一种比较自然的识别对象的方法。

为了刻画较大的软件需求定义的需要，提供了逐步精化的手段，从而可根据实际问题的需要不断将某些复杂对象进行逐层分解，得到一个层次式的对象关系图。

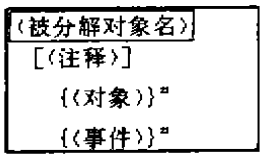
因此，对象关系图由对象、事件及相应的图符等组成。每一层对象关系图均用一个边界框界定，不同层次之间的联系与区分是借助一个称为"被分解对象"的对象名字来表达的。其语法描述如下：

〈对象关系图〉::=〈边界框〉〈被分解对象名〉[〈注释〉]{〈对象〉}{〈事件〉}*

〈被分解对象名〉::=〈标识符〉|system

〈对象分解关系图〉::=〈对象关系图〉

可由以下图形形式等价表示：



上述定义中的〈对象分解关系图〉部分并不直接出现在该对象关系图中，而是单独作为新的一层对象关系图存在，二者的联系是借助〈被分解对象名〉来实现的。

"被分解对象名"用于说明该层（子）对象关系图是由此对象分解而来。当为 system 时，表明该对象关系图是对系统的第一次分解，即最顶层对象关系图。对顶层对象关系图中的某对象进行分解，即得到该对象的下一层对象关系图，不继续分解过程，就能得到不同层次的对象关系图。

上述定义中，对于某对象，如果后接〈对象分解关系图〉项，则说明存在分解该对象所得的下层对象关系图。

6.2 对象

对于某一层对象关系图，其对象分为两类：内部对象和外部对象，内部对象是在分解上层某对象得

到该层对象图的过程中新生成的对象。外部对象或为目标软件(系统)的外部环境中的对象(如,系统用户等),或为上层对象关系图中与对应的被分解对象有事件关系的所有对象在该层对象关系图中的出现。它是为了更加清晰、完整地描述每层对象关系图中对象之间的行为关系而引入的。

对于某一对象,它在且仅在唯一一层对象关系图中为内部对象。如果在其它层对象关系图中出现,那么它仅能作为外部对象。

内部对象和外部对象在对象关系图中的图形表示是不同的。事实上,对象关系图的每次精化均是指对于某层对象关系图中内部对象的分解。外部对象是不能被分解的。

〈对象〉::=〈外部对象〉|〈内部对象〉

〈外部对象〉::=〈外部对象图符〉〈外部对象名〉

〈外部对象名〉::=〈标识符〉|external

可由以下图形形式等价表示:

〈外部对象名〉

其中,名为 external 的外部对象是指软件系统的外部环境中的对象,例如,系统用户等。

〈内部对象〉::=〈内部对象图符〉〈内部对象名〉[标志][所属类名][〈属性值约束表〉][〈行为表〉]

〈内部对象名〉::=〈标识符〉

〈标志〉::=→

〈所属类名〉::=〈标识符〉

〈属性值约束表〉::={〈属性名〉[="值"]}*

〈属性名〉::=〈标识符〉

〈行为表〉::=〈行为名〉{,〈行为名〉}*

〈行为名〉::=〈标识符〉

可由以下图形形式等价表示:

[标志]	[〈所属类名〉]
内部对象名	[〈属性值约束表〉]
	[〈行为表〉]

其中,当标志为"→"时,表示该内部对象已被分解,即存在其下一层的对象关系图。我们把不具有"→"标志的内部对象称为"基本对象";其它内部对象称为"扩展对象"。

如果没有〈所属类名〉部分,则认为该对象是类关系图中同名类的唯一实例。对象的"属性值约束"用于说明其类创建该对象时,相应属性的初始值。如果在此表中只有属性名,没有指定值,则表明属性的约束值在对象/类字典中刻画。

6.3 事件

对象之间所发生的事件反映了对象之间的交互通信关系。对于分解内部对象 A 所得的下层对象关系图 ORD(A)而言,ORD(A)中内部对象之间的事件称为内部事件;内部对象与外部对象之间的事件称为外部事件。外部事件亦即上层对象关系图中与 A 相关的所有事件。

〈外部事件〉::=〈事件图符〉〈消息序列〉:〈外部对象〉〈内部对象〉}|〈事件图符〉〈消息序列〉:〈内部对象〉〈外部对象〉}

〈内部事件〉::=〈事件图符〉〈消息序列〉{〈内部对象〉〈内部对象〉}

〈消息序列〉::=〈消息〉{,〈消息〉}*

〈消息〉::=〈行为名〉[(〈消息参数表〉)]

〈行为名〉::=〈标识符〉

〈消息参数表〉::=〈消息参数〉{,〈消息参数〉}*

〈消息参数〉::=〈常数〉|〈变量〉|〈表达式〉

两对象之间发生的一个事件,必须说明其间的消息传递情况,这是事件的基本内容。每一个消息可以有消息参数,也可以没有,这要根据对象所属类的相关行为定义中的形式参数情况而定,其详细定义参见8.1.2。

外部事件与内部事件的形式均可由以下图形形式等价表示:

〈消息序列〉 ——→

二者的不同在于事件所连接的两个对象的类型,前者要求两个对象,一个是外部对象,一个是内部对象;后者要求两个对象皆为内部对象。

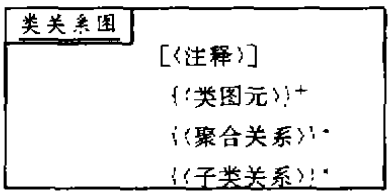
七、类关系图

7.1 类关系图(Class-Relationship Diagrams, CRD)

类关系图是类关系模型的图形化表示。类关系模型是刻画对象所属的类,类的组成(属性与行为)及类间关系的模型。类间的关系包括聚合关系和子类关系两种。目标系统中,每一个对象均应属于某一个类。一个系统只有一个类关系图。

〈类关系图〉::=〈边界框〉类关系图[〈注释〉]{〈类图元〉}+{〈聚合关系〉}*{〈子类关系〉}*

可由以下图形形式等价表示：



7.2 类

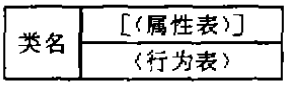
类是一组具有共同属性和行为的对象的抽象，可以作为刻画对象的模板。相应地，对象看作是类的实例。在类关系图中，我们用类图元表示上述类。描述基本对象的类称为基本类；描述扩展对象的类称为扩展类。

〈类图元〉::=〈类图符〉〈类名〉[〈属性表〉][〈行为表〉]

〈类名〉::=〈标识符〉

〈属性表〉::={〈属性名〉}*

可由以下图形形式等价表示：



其中，类名是必需的，若无〈属性表〉和〈行为表〉两部分，则表明该类是扩展类。若有〈属性表〉和〈行为表〉，则需在对象/类字典中具体描述其内容。

7.3 类间关系

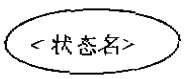
类间关系包括聚合关系和子类关系。聚合(aggregation)关系也称整体-部分(whole-part)关系，指整体类的对象是由部分类对象组成。它对应于对象关系图中的对象分解，即层次分解关系。

子类关系也称类的层次关系(hierachy，is-a)，指一个或一些类是另一个类的子类，后者称为父类。父类具有所有子类中共同的属性与成分，子类至少具有父类中的全部属性与行为，并可能增加新的成分。二者的语法描述分别为：

(1)聚合关系

〈聚合关系〉::=〈〈类〉}*〈聚合关系图符〉〈类〉

可由以下图形形式等价表示：



其中，聚合关系所连接的类(左/右)必须是不同的，且左边的类应至少两个。

(2)子类关系

〈子类关系〉::=〈类〉〈子类关系图符〉〈类〉

可由以下图形形式等价表示：

其中，子类关系所连接的左、右类必须是不同的，且左边的类至少应有一个，右边的类有且仅有一个。称左边的诸类为子类，右边的类为父类。

八、状态转换图

8.1 状态转换图(State-Transition Diagrams，STD)

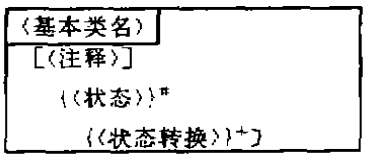
状态转换图是状态模型的图形化表示。状态模型是刻画一组相似对象在其生存期内属性的变化情况，以及行为控制关系的模型。

系统中只有基本类才具有且必须有其相应的一个状态转换图。状态转换图包括状态、状态转换，以及引起这些转换的原因和应该采取的相应动作。语法描述如下：

〈状态转换图〉::=〈边界框〉〈基本类名〉[〈注释〉]{〈状态〉}*{〈状态转换〉}+

〈基本类名〉::=〈标识符〉

可由以下图形形式等价表示：



8.2 状态

状态是指对象的所有属性在某时刻的取值情况。在状态转换图中，状态用一个名字标志，不同状态的名字不能相同。其语法描述如下：

〈状态〉::=〈状态图符〉〈状态名〉

〈状态名〉::=〈标识符〉

可由以下图形形式等价表示：

8.3 状态转换

状态转换刻画引起一状态到另一状态转换的原因，以及应该采取的相应动作。转换原因有事件触发和条件触发两类。采取的动作实际是对对象行为的激发。其语法描述如下：

〈状态转换〉::=〈状态转换图符〉[〈触发条件〉]〈动

作〉

〈触发条件〉::=〈事件触发〉〈条件触发〉

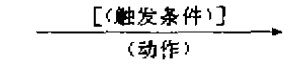
〈事件触发〉::=〈消息序列〉[/〈事件结果〉]

〈条件触发〉::=〈条件表达式〉

〈事件结果〉::=〈字符串〉|〈条件表达式〉

〈动作〉::=〈行为名〉

可由以下图形形式等价表示：



九、对象/类字典

对象/类字典(Dictionary of Object & Classes, DOC)用以详细刻画对象及类有关信息，分为类字典和对象字典两部分。类字典用于具体刻画需求定义的每个基本类的属性和行为；对象字典用于刻画需求定义诸基本对象的创建约束。

〈对象/类字典〉::=类字典〈类字典〉对象字典〈对象字典〉

9.1 类字典

类字典中所描述的每个类皆允许有其常量说明和类型定义部分。如果所说明的类不是system，那么常量说明和类型定义只在当前类中有效，否则，将被认为是全局的，即对所有类均有效。类字典的语法描述如下：

〈类字典〉::={〈类〉}*

〈类〉::=**class**〈类名〉[〈常量说明〉][〈类型定义〉]〈非形式化概括描述〉[〈详细描述〉]**endclass**

〈类型定义〉::=〈序列型说明〉|〈集合型说明〉|〈枚举型说明〉|〈指定集合类型说明〉|〈类型别名定义〉

〈类型别名定义〉::=〈类型别名〉=〈类型名〉

〈类型别名〉::=〈标识符〉

〈非形式化概括描述〉::=〈注释〉

〈详细描述〉::={〈属性描述〉}*{〈行为说明〉}*

其中，非形式化概括描述和详细描述未必二者皆备，前者便于用户理解需求定义，后者便于进行需求分析和设计。

9.1.1 属性描述

〈属性描述〉::=**attribute**〈属性名〉:〈类型说明〉[〈约束条件〉]

〈属性名〉::=〈标识符〉

〈类型说明〉::=〈类型名〉

〈约束条件〉::=**constraint**〈字符串〉|〈谓词公式〉

其中，〈类型名〉允许是类名，或者是以一个类为基类型的复合类型名，表明一个类的属性可以说明为另一个类，或为以其它类为基类型的复合类型。约束条件用于对被描述属性的约束和限制，如，年龄>=18岁。谓词公式的语法描述参见9.1.2。

9.1.2 行为说明

〈行为说明〉::=**behavior**〈行为名〉"{"**input**[〈输入参数表〉]**output**[〈输出参数表〉]**functionality**〈语义解释〉|〈断言式〉[**constraint**〈限制条件〉]"}"

〈输入参数表〉::=〈参数组〉{;〈参数组〉}*

〈参数组〉::=〈常量参数组〉|〈变量参数组〉

〈常量参数组〉::=〈常量〉{,〈常量〉}*

〈变量参数组〉::=〈变量名〉{,〈变量名〉}*:〈类型名〉

〈变量名〉::=〈标识符〉

〈输出参数表〉::=〈变量参数组〉

〈语义解释〉::=**description**〈字符串〉

〈断言式〉::=**pre**〈前断言〉,**post**〈后断言〉

〈前断言〉::=〈谓词公式〉

〈后断言〉::=〈谓词公式〉

〈限制条件〉::=〈字符串〉|〈条件表达式〉

〈谓词公式〉::="取布尔值的"〈表达式〉|**not**(〈谓词公式〉)|**and**(〈谓词公式〉)|**or**(〈谓词公式〉)|**imply**(〈谓词公式〉,〈谓词公式〉)|**iff**(〈谓词公式〉,〈谓词公式〉)|**forall**(〈约束变量说明表〉)(〈谓词公式〉)|**exist**(〈约束变量说明表〉)(〈谓词公式〉)|**nnique**(〈约束变量说明表〉)(〈谓词公式〉)

〈约束变量说明表〉::=〈约束变量说明〉{,〈约束变量说明〉}*

〈约束变量说明〉::=〈变量标识符〉{,〈变量标识符〉}*:〈类型名〉

〈变量标识符〉::=〈标识符〉

其中，**not**，**and**，**or**，**imply**，**iff** 分别对应于一阶谓词演算中的连结词：$\neg$，$\wedge$，$\vee$，$\Rightarrow$，$\Leftrightarrow$。**forall**，**exist**，**unique** 分别对应于一阶谓词演算中的量词：$\forall$，$\exists$，$\exists!$。

类中的行为可能导致其某些属性值的改变，为了显式地体现这种改变，用原属性名后加一个符号"'"，如 att'，表示行为运行后的属性。上述定义中，前断言中自由变量只能是相应输入参数表中变量的

子集，后断言中自由变量只能是相应输入参数表中变量、输出参数表中变量、属性或标有'的属性。

对于行为的说明，允许没有输入或输出，也可以没有限制条件。functionality 部分是必需的，它用于刻画行为的具体功能含义。如果是面向用户则可采用自然语言进行其语义解释，否则，若为精确表达及便于到形式功能规约的转换，则采用形式化的断言式来定义其前、后置断言，语义解释和断言式未必二者皆备。

9.2 对象字典

由于任意一个对象都属于某个类，即对象是类的一个实例，那么，对象被创建时，作为一个实例，其某个或某些属性即被初始化为某(些)特定值。对象创建约束就是对这种性质进行刻画。对于一个类只有一个对象实例的情况，可以不作特别的约束。

〈对象字典〉::=｛〈对象创建约束条目〉｝*

〈对象创建约束条目〉::=object〈对象名〉'〈属性约束表〉

〈属性约束表〉::=〈属性约束〉｛,〈属性约束〉｝*

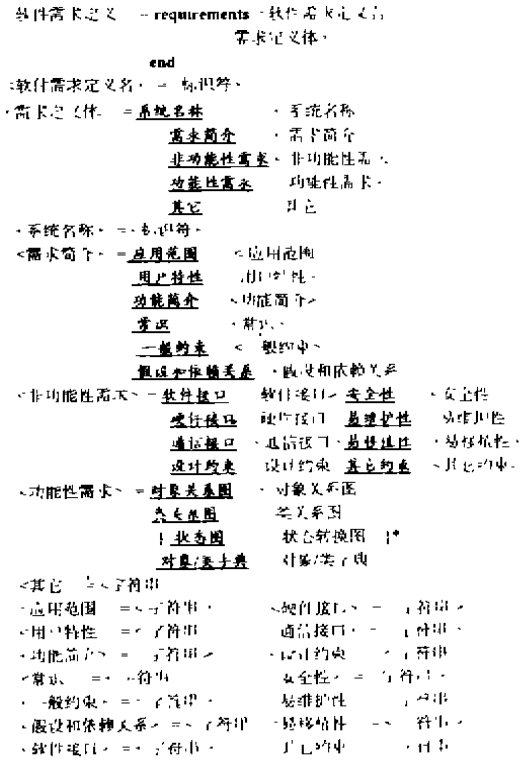
〈属性约束〉::=〈属性名〉=〈常量〉

十、软件需求定义

软件需求定义是软件需求的完整陈述。软件需求包括功能需求和非功能需求两个方面。功能需求从用户的角度明确了该软件系统必须具有的功能行为，其中包括对系统的操作过程和对操作模式的控制过程等的模式。功能需求不仅要解释每项功能要"做什么"，而且还要指明这些功能间的联系及相互的依赖关系(控制和数据)，但不涉及"怎样做"的描述。它是整个软件需求的核心所在。

非功能需求在功能需求的基础上对软件需求作进一步的刻画，包括功能限制、设计限制、环境描述、数据和通信规程和项目管理等。设计限制刻画软件系统的性能(如易维护性、易移植性)、响应时间、安全性标准和质量指标等；设计限制主要包括系统的开发平台等；环境描述主要包括系统所属环境的各个方面及其应用领域(如软件、硬件接口等)；数据与通信规程主要刻画系统各部分之间，以及系统与外部环境之间的数据流(如通信接口等)；项目管理涉及系统开发与系统交付等方面的需求，主要包括文档标准、模块测试与集成等。

其语法描述如下：



软件需求定义应该具有一个名称、需求定义体中应该具有需求简介，功能性需求以及必要的非功能性需求部分。其中，对象关系图、类关系图、对象状态图分别反映了问题的不同侧面。对象关系图反映与模拟问题领域对象的动态行为关系，类关系图说明问题领域对象所属的类及其间的静态关系；对应于基本类、状态转换图刻画一组对象在其生存期内属性值的变化情况，以及行为控制关系。对象/类字典是功能性需求部分必需的，它用以详细说明需求定义的有关信息。

致谢：NDORL 语言是由南京大学计算机软件研究所徐家福教授主持设计的，是课题组集体智慧的结晶。本文的写作自始至终得到徐先生的耐心指导与帮助，并受益于课题组的费宗铭博士、金凌紫博士、伊波博士及博士生董丽君等同志，作者谨此致谢。

参考文献

[1]徐家福著，系统程序设计语言，科学出版社，1983

[2]徐家福等著，对象式程序设计语言，南京大学出版社，1992