软件需求定义语言 NDRDL

董丽君 费宗铭 朱 鸿 金凌紫
(南京大学计算机软件研究所 南京210093)

摘 要 NDRDL 语言是一种图形化的软件需求定义语言，用于书写软件需求定义。其特点是：形象直观、表达力强、实用性好、可靠性高。本文是该语言的试用文本。

关键词 软件需求定义，软件需求定义语言，图形化。

一、引 言

八十年代以来，南京大学计算机软件研究所对从软件功能规约到可执行代码全过程的自动化技术进行了较为系统的研究，并取得一定进展。先后设计了软件规约语言 GSPEC 和 FGSPEC，研制了 NDAUTO、NDTPS、NDADAS、NDIPS、NDSAIL 等软件自动化系统。然而，从非形式的软件需求定义到形式的功能规约的自动转换仍是软件自动化的一大难题。为此，我们致力于这方面的研究与开发工作，研制基于功能分解的软件需求支撑系统 NDRASS，旨在辅助分析人员实现从需求定义到功能规约的转换。同时，为了书写软件需求定义我们设计了软件需求定义语言 NDRDL。

NDRDL 以结构化数据/功能分析方法为基础，将待解的问题抽象成一系列需要具备的功能，通过表述各功能之间的数据流向、控制流程等关系，来刻画用户对系统的需求。NDRDL 还支持功能分解模型中自顶向下，逐层分解的结构化方法，提供了分层的数据流图、控制流图等设施。

NDRDL 语言具有如下特点：

1)形象直观 采用图形(即数据流图、控制流图、实体关系图)和正文相结合的方法描述软件需求，形象直观，易于理解。

2)表达力强 不仅提供了较强的功能需求的描述设施，而且还提供了描述非功能需求的设施，从而用户可以用 NDRDL 书写完整的软件需求定义。

3)实用性好 在功能需求描述设施的设计上，

失效时系统继续运行的问题。并在不同程度上解决故障情况下的负载均衡问题。相比之下，链式分布在容错和负载均衡方面最优。而数据镜象虽然容错性好，但负载均衡性差；交叉分布虽然负载均衡好，但容错性差。这里指的容错性是指任何两个结点同时失效时系统的可用性。在这种情况下，交叉分布不如数据镜象和链式分布。但是，任何收益都是需要付出代价的。链式分布的负载匀衡是动态调整来实现的，所以在 I/O 操作方面增加了软件的复杂性，需要有一定的算法来支持，但这一点不难做到。而交叉分布的负载均衡是静态的。

RAID 系统的数据校验方式的最大好处是不需要数据副本，减少了磁盘需求，同时还能提供并发 I/O 操作。但它只能对一个磁盘故障容错，并且对处理机故障无能为力。正是由于这一特点，它可以用在单机系统中，也可用在其他模式的多机系统中。

参考资料

[1]Haran Boral et al., Prototyping Bubba, A Highly Parallel Database System, IEEE trans. on Knowledge and data engineering, Vol. 2, No. 1, 1990

[2]David I. DeWitt et al., The Gamma Database Machine Project, same to [1]

[3]Tandem Computers inc., Introduction to Non-Stop SQL, Part no. 82317, March 1987

[4]J. Page, High Performance Database for Client/Server Systems, Applied Imformation Technology 13. Parallel Processing and Data Management, 1992

[5]S. Costicoglou et al., On Benchmarking the ORACLE Parallel Server on nCUBE2. Advances in Parallel & Distributed System. October 6, 1993

[6]杨利、周兴铭、郑若忠，并行数据库系统的体系结构，计算机科学，1994，Vol. 21, No. 4

[7]Patrick Valdnnieg. Parallel Database Systems: Open Problems and New Issues. Distributed and Parallel Databases, 1, 1993

以数据/功能分析方法这一较成熟的软件需求分析方法为基础，具有较好的实用性。

4)可靠性高　明确地定义了三个模型在刻画同一软件系统时所必须满足的一致性条件，有助于保证软件需求定义的一致性和完整性，也有助于保证所开发的软件具有较高的可靠性。

二、语言描述

2.1　语法的描述

2.1.1　局部语法的描述　采用扩充的 Backus-Naur 形式体系（简称 BNF），它由一组语法规则组成。语法规则由非终极符、连接词和终极符构成。形为：

S::=E

表示左边的 S 定义为右边的 E。其中 S 为非终极符，形为〈汉字列〉，表示语法单位；汉字列为一串汉字，是该语法单位的名。

E 是语法表达式，形为 $T_1|T_2|\cdots|T_n(n>=1)$，表示 E 可以是 T_1，T_2，或 $T_i(1<=i<=n)$；

$T_i(i=1,2,\cdots,n)$为语法项，形为 $F_1F_2\cdots F_m$　$(m>=1)$，表示 T_i 由 $F_1,F_2,\cdots,F_j(1<=j<=m)$组成，其中每两个语法单位名之间用空格隔开；

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

1)N　2)$C_1C_2\cdots C_t$　$(t>=1)$　3)X

N 为非终极符，如〈数据流图〉；$C_i(i=1,2,\cdots,t)$为终极符。终极符是 NDRDL 语言的组成部分；X 为连接词。

本文本中使用的连接词及其语义如下：

[　]　表示被括内容可以不出现或出现一次

{　}　表示被括内容可以不出现或出现有限多次

|　　或

A&B　表示 A 出现在 B 之内

A#B　表示 A 出现在 B 之上

2.1.2　全程语法的描述　语言中不能用 BNF 刻画的语法规则为全程语法，它们采用自然语言描述。

2.2　语义的描述

语义指出需求定义语言 NDRDL 的含义，其描述采用自然语言。

三、词法结构

在软件需求定义语言 NDRDL 中，词法单位是由字符和图符构成的。

3.1　字符集

字符集包括本语言使用的全部字符，分字母、数字、特定字符、汉字和空格字符五种。字母用来构成标识符和字符串；数字用来构成数、标识符和字符串；特定字符用作语言中的运算符或其它成分；空格字符用作词与词之间的分隔符号。

〈字符〉::=〈字母〉|〈数字〉|〈特定字符〉|〈汉字〉|〈空格字符〉

〈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〉::="|'|||%|$|#|;|'>|'>=|<|<=|<>|∀|∧|⇒|∀|∃|∃|!|;|;|'|\|{|}|[|]|.|=|-|+|*|/|**|(*|*)|.|∈|↑

〈汉字〉::="汉字字符"

〈空格字符〉::=space

3.2　图符集

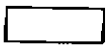
图符集包括本语言所使用的全部图符。

<图符> = <数据流图符> | <控制流图符> | <实体关系图符>

<数据流图符> = <数据存储图符> | <数据池图符> | <处理图符> | <数据流向图符>

<控制流图符> = <状态图符> | <控制流向图符>

<实体关系图符> = <实体图符> | <关系图符> | <属性图符> | <属性连线符>

<数据存储图符> = ——　<数据池图符> = 

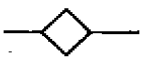
<处理图符> =  　<数据流向图符> =

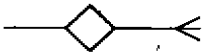
<状态图符> =  　<控制流向图符> =

<实体图符> =  　<属性图符> = 

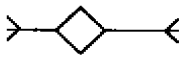
<属性连线符> = ——

<关系图符> = <一一对应关系图符> | <一多对应关系图符> | <多一对应关系图符> | <多多对应关系图符> <多元关系图符>

<一一对应关系图符> = 

<一多对应关系图符> = 

<多一对应关系图符> = 

<多多对应关系图符> = 

<多元关系图符> = 

3.3　词法单位

词法单位是软件需求定义中具有含义的最小单位。在 NDRDL 中，词法单位有如下七种：

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

3.3.1 标识符 用来标记数据类型、常量、变量、数据流、操作、控制流、状态、关系、属性等，也可作为常量的值。标识符由一串字母字符和数字字符组成，或由汉字字符组成，其第一个字符必须是字母或汉字。

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

3.3.2 标识图符 用来标记数据流图、控制流图和实体关系图中的元素。它由一个名和相应的图符组合而成。

〈标识图符〉::=〈名〉&〈图符〉|〈名〉#〈图符〉

〈名〉::=〈标识符〉

3.3.3 无正负号数 用于构作常量和表达式。

〈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〉

3.3.4 保留字 在语言中有其特定含义，不能由用户重新定义。保留字分为类型字、运算字、说明字和其它字四种。类型字用于类型定义，运算字用于刻画 NDRDL 中允许出现的运算，说明字是 NDRDL 需求定义中所涉及的关键字，其它字列出了需求定义中必须用到的其它一些保留字。保留字与其它词法单位之间用空格字符隔开。

〈保留字〉::=〈类型字〉|〈运算字〉|〈说明字〉|〈其它字〉

〈类型字〉::=bool|int|real|string|seq|set|record|enum

〈运算字〉::=div|mod|abs|and|or|not|concat|isempty|head|tail|length|reverse|member|append|position|select|replace|insert|a.n|number|incorp|inter|union|differ|subset|succ|pred|ord

〈说明字〉::=Requirements|Introduction|General_description|Functional_requirements|Domain|System_category|Requirements_details|Nonfonctional_requirements|Function_list|Fonction|Graphical_description|Function_introduction|Description|Input|Output|Process_description|Data_name|Form|Description|Form|Process_name|Entity|Data_flow_diagram|Control_flow_diagrm|Entity_relationsbip_diagram|Data_dictionaries|Dictionaries|Process_dictionaries|Relation_dictionaries|Appendix|Index|Purpose|Scope|General_functions|Abbreviation|Bibliographies|User_characteristics|General_restriction|Security|Portability|Software_interface|Assumptions_and_dependences|Hardware_interface|Design_constraints|Maintainability|Correspondence_interface|Other_constraints|Management_information_system|Real_time_system|Decision_support_system|Data_processing_system|Expert_system|Plant_monitor_system|Constraints

〈其它字〉::=end|true|false|empty|abnormal

3.3.5 界限符 在其出现处含义自明。

〈界限符〉::= =|;|.|:|(|)|[|]|%|,|:|{|}

3.3.6 字符串 在 NDRDL 中，用户用自然语言书写的软件需求可以字符串的形式出现。字符串是由双引号括起来的一串字符。所谓一串字符就是一个或多个字符的并置。如果字符串中的某一字符本身就是双引号，该双引号必须重写两次。字符串的值就是由双引号括起来的那一串字符。

〈字符串〉::=""|"〈字符行〉"

〈字符行〉::=〈字符〉|〈字符行〉〈字符〉

3.3.7 注释 是由('和')括起来的一串字符。可以出现在 NDRDL 书写的需求定义中的任何位置，用于向软件开发人员提供信息，以助于理解，而对需求定义的含义无影响。

〈注释〉::=('字符串')

四、数据类型

数据类型是对具有相同取值范围和运算等性质的一组数据的抽象。数据类型包含三要素。即类型名、值集和运算集。NDRDL 需求定义中出现的任何常量、变量、函数及表达式有且仅有一种类型。

NDRDL 中提供了一组基本类型和一组从已有类型构造类型的类型构造符。基本类型有整型、布尔型、实型和字符串型四种。其类型名、值集和运算集由语言所定义，详见4.1；类型构造符包括枚举型构造符、序列型构造符、集合型构造符、记录型构造符。它们由用户指定类型的名、值集和运算集，详见4.2。NDRDL 中还提供了由用户通过自然语言指定数据类型的设施，详见4.3。

〈数据类型〉::=〈基本数据类型〉|〈构造数据类型〉|

〈用户指定的数据类型〉

〈基本数据类型〉::=〈整型〉|〈实型〉|〈布尔型〉|〈字符串型〉

〈构造数据类型〉::=〈序列型〉|〈集合型〉|〈枚举型〉|〈记录型〉

〈整型〉::=int

〈实型〉::=real

〈布尔型〉::=bool

〈字符串型〉::=string

〈序列型〉::=*seq*(〈数据类型〉)

〈集合型〉::=set(〈数据类型〉)

〈枚举例〉::=enum(〈枚举符号表〉)

〈枚举符号表〉::=〈枚举符号〉{,〈枚举符号〉}

〈枚举符号〉::=〈标识符〉

〈记录型〉::=record〈域表〉end

〈域表〉::=〈域〉{;〈域〉}

〈域〉::=〈域标识符〉:〈数据类型〉

〈域标识符〉::=〈标识符〉

4.1 基本数据类型

1)整型

类型名:int

值集:目标机所允许的所有整数的集合,即

〈整数〉::=〈无正负号整数〉|-〈无正负号整数〉

运算集:>(大于),<(小于),>=(大于等于),<=(小于等于),=(等于),<>(不等于),+(加),-(减),*(乘);abs 为求绝对值运算;div 为整除运算;mod 为模取运算。

2)实型

类型名:real

值集:目标机所允许的所有实数的集合

运算集:>,<,>=,<=,<>,=,+,-,*,/(除)

3)布尔型

类型名:bool

值集:true,false

运算集:not(逻辑否定),and(逻辑合取),or(逻辑析取)

4)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。

运算集:concat(x,y):将字符串 x,y 并置。

4.2 数据类型构造符

用户可以在数据字典中定义数据类型,所定义的类型的名为数据字典中的数据名,值集和运算集由数据字典中相应数据形式项中的内容,根据如下类型构造符的定义确定。

1)序列型构造符 seq 用于构造序列型数据类型。

类型名:通过数据字典中的数据名反映。

值集:让 T 为一类型,则 seq(T)的值集是由有限多个 T 类型的值的序列所构成的集合,不含任何值的序列称为空序列,记作 empty。

运算集:

lsempty(s):当序列 s 为 empty 时,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,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。

ordered(s):当序列 s 中元素按从小到大的次序排列时,其结果为 true,否则为 false。

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

concat(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 不存 s 中出现,其结果为0。

select(s,i):选出序列 s 的第 i 项,若 s 不存在第 i 项,则结果为异常 abnormal。

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,否则仍为 s。

replace(s,i,t):用 t 取代序列 s 中的第 i 项,当 i 大于序列 s 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;若 i 大于 s 的长度,则 t 加在 s 之后。

2)集合型构造符 set 用于构造集合型数据类型。

类型名:通过数据字典中的数据名反映。

值集:让 T 为一类型,则 set(T)的值集为 T 的值集的所有有限子集构成的集合,即类型 T 值集的

幂集，empty 表示空集合。

运算集：

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。

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.

member(e,s)：当 e 是集合 s 的元素时，结果为 true，否则为 false。

3)枚举型构造符 enum 用于构造枚举型数据类型。

类型名：通过数据字典中的数据名反映。

值集：让 $T_1, T_2, \cdots, T_n$ 分别为枚举符号表所列的 n 个枚举符号，则 enum($T_1, T_2, \cdots, T_n$)的值集就是由 $T_1, T_2, \cdots, T_n$ 所组成的集合。

运算集：=等于，<小于，<=小于等于，<>不等于，>大于，>=大于等于

succ(T_i)：表示 T_{i+1}

pred(T_i)：表示 T_{i-1}

ord(T_i)：表示求 T_i 的序号

4)记录型构造符 record 用于构造记录型数据类型。

类型名：通过数据字典中的数据名反映。

值集：所有相应的记录值所构成的集合，这些记录值都是由多个域值组成的多元组。

运算集：

a.n：取记录 a 中以 n 为域名的域值。

4.3 用户指定数据类型

在软件需求定义中，有时需要指定一个数据类型的值集，这个值集是由特定的软件环境决定的，这样的数据类型不能(或不便于)用枚举等简单数据类型来描述，而用结构数据类型又不能准确地反映数据的取值范围，NDRDL 语言中，提供了用户指定的数据类型这一设施，来定义这样的数据类型。

用户可以在数据字典数据名项中给出一个类型名，而在相应的数据形式项中指定 given_type，在非形式描述项中用自然语言指定其值集和运算集。

如：在数据字典中可包含如下数据项：

城市名	表示在数据库 A 中所列出的所有城市的名	given_type

五、常量、变量和表达式

5.1 常量

按照名和值是否一致来区别，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、布尔值、字符、字符串和枚举值五种，它们是表达式或说明的组成部分，定义如下：

〈常量〉::=〈字面常量〉|〈非字面常量〉

〈字面常量〉::=〈无正负号数〉|〈布尔值〉|〈字符〉|〈字符串〉|〈枚举值〉

〈布尔值〉::=true|false

〈枚举值〉::=〈枚举符号〉

〈枚举符号〉::=〈标识符〉

非字面常量是用户在数据字典中其数据形式项中指定为 const〈常量值〉的数据名，其值为：〈常量值〉

〈常量值〉::=〈常量表达式〉

此处常量表达式是不含变量的表达式，即由字面常量、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

变量可以取不同的值，本语言中通过变量说明来对变量命名，并规定其取值范围。

〈变量说明〉::=var〈变量名〉{,〈变量名〉}:〈类型名〉

〈变量名〉::=〈标识符〉

5.3 表达式

表达式用于表示求值规则，其内部各个运算符的执行顺序，按照优先级高低依次进行。各类运算符按照优先级从高到低的顺序依次为：乘幂运算符，单目运算符，乘除运算符，加减运算符，关系运算符，逻辑运算符。

表达式中每个运算符的各运算分量的类型必须与该运算符的定义相一致；函数调用中的实在参数的类型必须与相应的形式参数的类型相一致；由量词构成的表达式中相应的简单表达式必须是谓词，即布尔类型的表达式，量词中引入的变量的作用域限于该表达式。

〈表达式〉::=〈简单表达式〉|〈单目运算符〉〈简单表达式〉|〈简单表达式〉〈双目运算符〉〈简单表达式〉|〈量词〉〈变量〉:〈类型〉〈简单表达式〉

〈简单表达式〉::=〈表达式〉|〈常量〉|〈变量〉|〈函数调用〉

〈函数调用〉::=〈函数名〉(〈实参部分〉)

〈实参部分〉::=[〈实在参数〉{,〈实在参数〉}]

〈实在参数〉::=〈表达式〉

〈函数名〉::=〈标识符〉

〈量词〉::=∀|∃|∃!

〈单目运算符〉::=¬|→,

〈双目运算符〉::=〈逻辑运算符〉|〈关系运算符〉|〈加减运算符〉|〈乘除运算符〉|〈乘幂运算符〉

〈关系运算符〉::= =|<>|<|<=|>|>=|∈|∉|⊃|⊇|⊂|⊆|⊄

〈加减运算符〉::=+|−

〈乘除运算符〉::=*|/|mod|div

〈乘幂运算符〉::=**

〈逻辑运算符〉::=∧|∨|⇔|⇒

∀为全称量词,∃为存在量词,∃!为唯一存在量词,〈函数名〉或为语言中数据类型的运算名,或为操作字典中定义的操作名。

六、数据流图

数据流图(Data_flow_diagram)通过刻画数据在系统中的处理和变换过程来描述软件需求。

数据流图中的结点分三类,即数据池、数据存储和处理。数据池是指不受系统控制在系统以外的事物或人,表达了该系统数据的外部来源或去处,例如银行储户。数据存储是对数据储存的逻辑描述,例如职工档案。数据池和数据存储又称为数据终结结点,表示数据流图的结束结点。处理表达了对数据的变换功能。图中每一个结点都标以唯一的名。不同类的结点用不同的图符表示。如:

数据池 名 表示数据的产生者与接收者

数据存储 名 表示数据的存储库

处理 (名) 表示一个处理

结点之间通过有向边(即数据流向)彼此相连。

数据流向表示数据在系统中流动的方向,如:A $\xrightarrow{a}$ B,表示数据a从结点A流向结点B。每一个数据流向都应该是标以所流过的数据的名。

〈数据流图〉::=(〈数据池〉|〈数据存储〉)〈数据转换表〉〈数据终结〉|〈数据流图〉;〈数据通路〉

〈数据转换表〉::={〈数据转换〉}1,n

〈数据转换〉::=〈数据流〉〈处理〉〈数据流向〉|〈数据转换〉〈数据结点〉〈数据流〉

〈数据结点〉::=〈数据池〉|〈处理〉|〈数据存储〉

〈数据终结〉::=〈数据池〉|〈数据存储〉

〈数据通路〉::=〈数据结点〉〈数据转换表〉

〈数据流向〉::=〈数据流向名〉#〈数据流向图符〉

〈数据池〉::=〈数据池名〉&〈数据池图符〉

〈数据存储〉::=〈数据存储名〉&〈数据存储图符〉

〈处理〉::=〈处理名〉&〈处理图符〉

〈数据流向名〉::=〈标识符〉

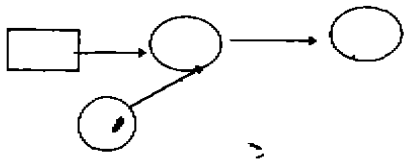
〈数据池名〉::=〈标识符〉

〈数据存储名〉::=〈标识符〉

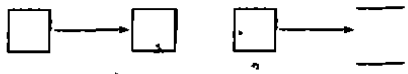
〈处理名〉::=〈标识符〉

数据流图中需注意以下几点:

1)在一个NDRDL数据流图中只有数据终结结点可以没有数据流出,所以数据流图要么以数据终结结点结束,要么以前面的数据流图中的结点结束。例如下图为不合法图:



2)数据终结结点之间,不允许有直接相连的数据流向。例如下图均不合法:



七、控制流图

控制流图(Control_flow_diagram)刻画系统的状态转换及操作合法性控制。

控制流图的结点只有一类,即状态结点, 。每一个结点都标以唯一的名。结点之间由控制流向连接。每一个控制流向也都标以唯一的名,如A $\xrightarrow{a}$ B,表示从状态A到状态B之间有一条名为a的控制流向。

〈控制流图〉::=〈状态转换通路〉|〈控制流图〉;〈状态转换通路〉

〈状态转换通路〉::=〈状态〉〈状态转换〉|〈状态转换通路〉〈状态转换〉

〈状态转换〉::=〈控制流向〉〈状态〉

〈控制流向〉::=〈控制流向名〉#〈控制流向图符〉

〈状态〉::=〈状态名〉&〈状态图符〉

〈控制流向名〉::=〈标识符〉

〈状态名〉::=〈标识符〉

NDRDL 控制流图中需要注意以下几点：1)控制流图只能有一个开始结点；2)从开始结点可以到达图中任一结点；3)允许有多个结束结点。

八、实体关系图

实体关系图(Entity_relationship_diagram)是对现实世界的对象实体及其间的关系的概念性表示。实体关系图中有两类结点，即实体和属性。实体结点之间通过四类关系进行连接。这四类关系分别是一一对应、一多对应、多一对应和多多对应。一个实体可以有若干个属性，属性刻画实体的特性。附加以一个关系字典，对各实体间的关系作进一步具体刻画。

〈实体关系图〉::=〈实体表〉[;〈关系表〉]
〈实体表〉::=〈实体〉|〈实体表〉;〈实体〉
〈关系表〉::=〈关系〉|〈关系表〉;〈关系〉
〈实体〉::=〈实体名〉&〈实体图符〉{〈属性连线符〉〈属性〉}0,n
〈属性〉::=〈属性名〉&〈属性图符〉
〈关系〉::=〈关系首部〉,〈实体名表〉
〈实体名表〉::=〈实体名〉|〈实体名表〉,〈实体名〉
〈关系首部〉::=〈关系名〉&〈关系图符〉
〈实体名〉::=〈标识符〉
〈属性名〉::=〈标识符〉
〈关系名〉::=〈标识符〉

九、字典

9.1 数据字典(Data_dictionaries)

数据字典以关键字 Data_dictionaries 开头，用于储存系统中出现的所有数据的相关信息。其数据条目的形式为：

Data_name Description Form Constraints

数据名(Data_name)用来标识数据；描述(Description)以非形式化的语言说明该数据项的相关信息，以便于用户查阅；数据形式(Form)给出了该数据的类型、分量名及形式，用以刻画它的类型与结构；约束条件(Constraints)是一个〈表达式〉，它用于刻画对该数据所取值的约束。

〈数据字典〉::=〈数据条目〉{;〈数据条目〉}end
〈数据条目〉::=Data_name〈数据名〉Description〈描述〉Form〈数据形式〉[Constraints〈表达式〉]
〈数据名〉::=〈标识符〉
〈描述〉::=@〈字符串〉
〈数据形式〉::=〈数据类型〉{,〈表达式〉}

9.2 操作字典(Process_dictionaries)

操作字典以关键字 Process_dictionaries 开头，用以描述系统中出现的所有的操作。其操作字典中条目的形式为：

Process_name Input Output Description Formal_description

其中操作名(Process_name)用以标识操作，输入(Input)与输出(Output)给出该操作的输入数据及输出数据，描述(Description)用自然语言说明该操作的作用；形式描述(Formal_description)给出该操作的形式定义。

〈操作字典〉::=〈操作条目〉{;〈操作条目〉}0,n end
〈操作条目〉::=Process_name〈操作名〉Input〈输入〉Output〈输出〉Description〈描述〉[Formal_description〈形式描述〉]
〈操作名〉::=〈标识符〉
〈输入〉::=〈变量说明列〉
〈输出〉::=〈变量说明列〉
〈描述〉::=@〈字符串〉
〈形式描述〉::=〈表达式〉
〈变量说明列〉::=〈变量说明〉{,〈变量说明〉}

9.3 关系字典(Relation_dictionaries)

关系字典以关键字 Relation_dictionaries 开头，刻画系统中实体之间的关系，是关系条目的集合。关系条目的形式为：

Relation_name Entities Description Formal_description

关系名(Relation_name)标识关系；实体(Entities)给出了该关系所涉及的实体的名；关系描述(Description)用自然语言描述实体间的关系；形式描述(Formal_description)先给出该关系的形式定义。

〈关系字典〉::=〈关系条目〉{;〈关系条目〉}end
〈关系条目〉::=Relation_name〈关系名〉Entities〈实体名〉{,〈实体名〉}Description〈关系描述〉[Formal_description〈形式描述〉]
〈关系描述〉::=@〈字符串〉
〈实体名〉::=〈标识符〉

9.4 语言的一致性约束

用 NDRDL 语言书写的软件需求定义必须满足如下一致性约束条件：

1)数据流图中所用的数据必须在实体关系图中定义。

2)控制流图中的操作必须是数据流图中出现的操作。

3)所用的术语都必须在数据字典、操作字典和关系字典中定义。

十、软件需求定义

软件需求定义是对软件需求的完整陈述。NDRDL 软件需求定义以关键字 Requirements 开头，然后是需求定义名；主体部分包括介绍、一般性描述、需求定义体、附录和索引五个部分。

〈软件需求定义〉::＝Requirements〈需求定义名〉；Introduction〈介绍〉；Generai_description〈一般描述〉；Requirements_details〈需求定义体〉；Appendix〈附录〉；Index〈索引〉

介绍部分是对其本身的自我介绍，包括需求定义的目的、产品的适用范围、应用领域、系统的类型及系统中所用的缩写语等。其语法描述为：

〈介绍〉::＝Purpose〈目的〉；Scope〈范围〉；Domain〈应用领域〉；System_catalogue〈系统类型〉；Abbreviation〈缩写语的定义〉[；Bibliographies〈参考文件〉]

系统类型部分建议用户使用如下关键字刻画系统类型：Management_information_system（＊管理信息系统＊）、Real_time_system（＊实时系统＊）、Expert_system（＊专家系统＊）、Decision_support_system（＊决策支持系统＊）、Data_processing_system（＊数据处理系统＊）、Plant_monitor_system（＊设备监管系统＊）。

系统的应用领域部分建议用户使用不多于四个字的关键字来刻画系统的应用领域，当一个关键字不能很好地刻画应用领域时，可用多个关键字。

一般描述部分提供该软件的所有功能，并对用户的背景及受训练的程度作简要性的介绍。其语法描述为：

〈一般描述〉::＝General_functions〈总功能〉；User_characteristics〈用户特性〉[；General_restriction〈一般约束〉[；Assumptions_and_dependences〈假设和依赖关系〉]

需求定义体是对软件需求的描述。软件需求包括功能需求和非功能需求两个方面。

〈需求定义体〉::＝Functional_requirements〈功能需求〉；Nonfunctional_requirements〈非功能需求〉

功能需求包括功能表、图形描述和字典，它是整个软件需求的核心。

〈功能需求〉::＝Function_list〈功能表〉；Graphical_description〈图形描述〉；Dictionaries〈字典〉

功能表列出了各个功能，每个功能的输入、输出及处理描述；图形描述给出相应的数据流图、控制流图、实体关系图和字典。

〈功能表〉::＝{Function〈功能名〉〈功能〉}1，n

〈功能〉::＝Function_introduction〈功能介绍〉；Input〈输入描述〉；Outpnt〈输出描述〉；Process_description〈处理描述〉

〈图形描述〉::＝Data_flow_diagram〈数据流图〉；Control_flow_diagram〈控制流图〉；Entity_relationship_diagram〈实体关系图〉

〈字典〉::＝Data_dictionaries〈数据字典〉；Process_dictionaries〈操作字典〉；Relation_dictionaries〈关系字典〉

非功能需求包括功能限制、设计限制，环境描述，数据与通信规程和项目管理等。

功能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，如硬件接口；

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；

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、模块测试与集成过程等。这部分的语法描述如下：

〈非功能需求〉::＝Software_interface〈软件接口〉[；Hardware_interface〈硬件接口〉][；Correspondence_interface〈通信接口〉][；Design_constraints〈设计限制〉][；Security〈安全性〉][；Maintainability〈易维护性〉][；Portability〈易移植性〉][；Other_constraints〈其它约束〉]

其余的语法单位定义如下：

<功能介绍> ＝@<字符串>
<功能名> ＝<标识符>
<输入描述> ＝@<字符串>
<输出描述> ＝@<字符串>
<处理描述> ＝@<字符串>
<目的> ＝@<字符串>
<范围> ＝@<字符串>
<应用领域> ＝@<字符串>
<系统类型> ＝@<字符串>
<缩写语的定义> ＝@<字符串>
<参考文件> ＝@<字符串>
<总功能> ＝@<字符串>
<用户特性> ＝@<字符串>
<一般约束> ＝@<字符串>
<假设和依赖关系> ＝@<字符串>
<软件接口> ＝@<字符串>
<硬件接口> ＝@<字符串>
<通信接口> ＝@<字符串>
<设计约束> ＝@<字符串>
<安全性> ＝@<字符串>
<易维护性> ＝@<字符串>
<易移植性> ＝@<字符串>
<其它约束> ＝@<字符串>
<需求定义名> ＝<标识符>

软件需求定义是用户与软件开发人员之间的契约的基础，主要面向用户，采用基于现实世界的描述

模型，以便于用户理解。

NDRDL 书写的软件需求定义具有如下结构：

```
Requirements 需求定义名
Introduction(*介绍*)
    Purpose(*目的*)
    Scope(*范围*)
    Domain(*应用领域*)
    System_catalogue(*系统类型*);
    Abbreviation(*缩写语的定义*)
    [;Bibliographies(*参考文件*)]
General_description(*一般描述*)
    General_functions(*总功能*);
    User_characteristics(*用户特性*)
    [;General_restriction(*一般约束*)]
    [;Assumptions_and_dependences(*假设和依
    赖关系*)]
Requirements_details(*需求定义体*)
  Functional_requirements(*功能需求*)
    Function_list(*功能表*)
      function 功能名(*功能1*)
        Function_introduction(*功能介绍*)
        Input(*输入描述*)
        Output(*输出描述*)
        Process_description(*处理描述*)
      function 功能名(*功能2*)
        Function_Introduction(*功能介绍*)
        Iuput(*输入描述*)
        Output(*输出描述*)
        Pocess_description(*处理描述*)
      Graphical_description(*图形描述*)
        Data_flow_diagram(*数据流图*)
        Control_flow_diagram(*控制流图*)
        Entity_relationship_diagram(*实体关系
        图*)
      Dictionaries(*字典*)
        Data_dictionary(*数据字典*)
        Process_dictionary(*操作字典*)
        Relation_dictionary(*关系字典*)
      Nonfunctional_requirements(*非功能需求
      *)
        Software_interface(*软件接口*)
        [;Hardware_interface(*硬件接口*)]
        [;Correspondence_interface(*通信接口
        *)]
        [;Design_constraints(*设计限制*)]
        [;Security(*安全性*)]
        [;Maintainability(*易维护性*)]
        [;Portability(*易移植性*)]
        [;Other_constraints(*其它约束*)]
Appendix(*附录*)
Index(*索引*)
```

致谢　软件需求定义语言 NDRDL 是由南京大学计算机软件研究所徐家福教授主持设计的。本文的成文工作自始至终得到徐先生的指导与帮助，并受益于课题组的吕建博士、尹波博士、陈道蓄副教授、王志坚博士、博士生张家重和陶先平硕士等同志。作者谨此致谢。

参考文献

[1]徐家福，系统程序设计语言，科学出版社

[2]徐家福，陈道蓄，吕建，王志坚，软件自动化，清华大学出版社，广西科学技术出版社

[3]徐家福，徐家福文集，南京大学出版社

[4]Jin Lingzi, Zhu Hong, Xu Jiafu, Proceedings of ICNGCS, 1989, 4

[5]Davis, A. M., Software Requirements, 2nd Edition, Prentice-Hall, 1993

[6]Rich, C., et al., Towards a requirements apprentice, Fourth International Workshop on Software Specification and Design, 1987

[7]Williams, M. H. et al., Conversion of unstructured flow diagrams to structured form, The Computer Journal, vol. 21, no. 2, 1978

[8]Cantone, G. et al., Well-formed conversion of unstructured one-in/one-out schemes for complexity measurement and program maintenance, The Computer Journal, vol. 29, no. 4, 1986

[9]Fraser, M. D. et al., Informal and formal requirements specification languages: bridging the gap, IEEE Trans. on Software Engineering, vol. 17, no. 5, 1991

[10]John A. McDermid (ed). Software Engineer's Reference Book, Butterworth-Heinemann, 1991