VDM-TXT: Um Tradutor de Especificação em VDM

para Linguagem Natural

Marcos Vianna Villas

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Sueli Mendes, Ph.D.

(Presidente)

Prof. Sheila Regina Murgel Veloso, D.Sc.

Prof. José Lucas Mourão Rangel Netto, Ph.D.

RIO DE JANEIRO, RJ - BRASIL

JANEIRO DE 1991

VILLAS, MARCOS VIANNA

VDM-TXT: Um Tradutor de Especificação em VDM para Linguagem Natural [Rio de Janeiro] 1991

VIII, 96 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1991)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Computação I. COPPE/UFRJ II. Título

Curriculum

Marcos Vianna Villas, natural do Rio de Janeiro, nascido em 31 de Janeiro de 1962, formou-se na Graduação em Tecnologia de Processamento de Dados da PUC/RJ em 1982.

Trabalha desde 1982 no desenvolvimento de sistemas aplicativos e de software básico, onde aplicou o seu conhecimento sobre metodologias e técnicas estruturadas para desenvolvimento de sistemas, bancos de dados, sistemas operacionais e linguagens de programação. Já atuou como programador, analista de sistemas, consultor e gerente técnico.

Professor Agregado do Departamento de Informática da PUC/RJ, leciona a nível de graduação em Tecnologia de Processamento de Dados desde 1983 disciplinas como Linguagens e Técnicas de Programação, Estrutura de Dados, Inteligência Artificial e Banco de Dados, além de orientar trabalhos de final de curso.

É co-autor dos livros "VAX: Uma Introdução" (Ed. Campus, 1987), "Programação: conceitos, técnicas e linguagens" (Ed. Campus, 1988) e "Estruturas de Dados: especificação e implementação" (Ed. Campus, título provisório, em preparação).

Dedicatória

Esta tese é dedicada ao meu pai, Tomas Agustin Villas, falecido em 16 de Fevereiro de 1988, aos 66 anos.

Para um pai que teve que abandonar o seu pais (China) com poucos bens materiais e que sempre dizia "estuda, porque o que você aprende nunca ninguem vai poder te tirar", ver o seu filho trabalhando, lecionando, escrevendo livros e fazendo mestrado deve ter sido muito gratificante.

Pena ele não estar aqui para presenciar mais esta etapa realizada.

Agradecimentos

Gostaria de agradecer às seguintes pessoas:

- Professora Sueli, pela sua paciência e orientação;

- Professor Rangel, pelas aulas particulares de teoria de compilação, pelo seu bom humor e por me ajudar a "resolver conflitos" em reuniões marcadas invariavelmente nos domingos à noite, para serem realizadas na segunda-feira seguinte;

- Raul Martins, da IBM Brasil, por me fornecer material sobre VDM;

- Maria Emília X. Mendes, minha esposa e eterna namorada, pela sua compreensão e apoio (mesmo porque ela também está fazendo a sua tese de mestrado agora...).

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência (M.Sc.).

VDM-TXT: Um Tradutor de Especificação em VDM para Linguagem Natural

Marcos Vianna Villas

Janeiro de 1991

Orientadora: Sueli Bandeira Teixeira Mendes

Programa: Engenharia de Sistemas e Computação

Um dos principais problemas do desenvolvimento de sistemas é a existência de erros no software recém criado. Testes não resolvem este problema, pois nunca são completos e só servem para mostrar a existência de erros, e não a sua ausência.

O uso de métodos formais de especificação permite a verificação matemática da correção de um sistema, porém usuários finais não têm preparo para ler e validar estas especificações.

O VDM-TXT gera um texto em linguagem natural a partir de uma especificação formal, de tal forma que usuários possam validar o entendimento do analista de sistemas a respeito do problema a ser resolvido.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Master of Science (M.Sc.).

VDM-TXT: A Translator of a VDM Specification into Natural Language

Marcos Vianna Villas

January, 1991

Thesis Supervisor: Sueli Bandeira Teixeira Mendes

Department: Systems and Computation Engineering

One of the main problems of systems development is the existence of "bugs" in the newly created software. Tests doesn't solve this problem, since they are never complete and are only useful to show the existence of bugs, and not their abscence.

The use of formal specification methods make possible a mathematical verification of correctness of a system, but end users are not prepared to read and validate these specifications.

The VDM-TXT generates a natural language text, having as input a formal specification, enabling end users to validate what the system analyst understands about the problem being solved.

Índice

I	Introdução	01
II	Desenvolvimento de Software	02
II.1	Ciclo de Vida	02
II.2	Especificação: Informal, Semi-Formal e Formal	03
II.3	O Problema da Verificação da Especificação	05
III	Especificação Formal em VDM	07
III.1	Apresentação	07
III.2	A Linguagem do VDM	08
III.2.1	Descrição Geral	11
III.2.2	Descrição de Dados	17
III.2.3	Descrição de Expressões	22
III.2.4	Descrição de Operações	28
III.2.5	Descrição de Fórmulas	31

Índice

IV	O Tradutor VDM-TXT	35
IV.1	Compilação	35
IV.1.1	O Gerador R*S	36
IV.1.2	Análise Léxica	36
IV.1.3	Análise Sintática	37
IV.1.4	Análise Semântica	38
IV.2	Geração de Texto	42
IV.2.1	Ordenação da Árvore Semântica	43
IV.2.2	Textos Complementares para Identificadores	44
IV.2.3	Estrutura da Descrição	45
IV.2.4	Descrição do Estado	47
IV.2.5	Descrição dos Tipos	48
IV.2.6	Descrição das Operações	49
IV.2.7	Descrição do Invariante	51
IV.3	Arquitetura	52
V	Conclusões	54
Apêndice A:	Gramática do VDM	55
Apêndice B:	Tabela de Tokens	62
Apêndice C:	O Sistema Exemplo	65
Apêndice D:	Textos Complementares	74
Apêndice E:	O Texto Exemplo	77
Referências Bibliográficas		94

CAPITULO I: Introdução

Todos os conceitos relativos às áreas de Informática e Ciência da Computação têm uma base teórica formal, conhecida pelo meio acadêmico.

O que se pode observar na prática de uso da informática no dia a dia das empresas é que muitas tarefas envolvidas na concepção, análise e projeto de sistemas de computação são realizadas segundo "regras de ouro", adquiridas com a experiência e perpetuadas dos "mais velhos" para os "iniciantes".

Com a crescente demanda por qualidade de software, aliada ao aumento do uso de computadores em áreas onde um erro não pode ser tolerado e nem reparado (ex: controle de aparelhos médicos em um CTI), também o uso de informática nas empresas deve ter um salto qualitativo.

Uma das formas de se aumentar esta qualidade é voltar às origens matemáticas, tornando formal o processo de desenvolvimento de software.

Esta tese trata do software VDM-TXT, que tem por objetivo traduzir uma especificação formal em VDM para linguagem natural. Inicialmente são abordados alguns aspectos de desenvolvimento de software (II). Em seguida é apresentado o VDM, um tipo de especificação formal (III). Finalmente é descrito o VDM-TXT, em suas duas principais fases: compilação VDM e geração de texto (IV).

CAPITULO II: Desenvolvimento de Software

II.1 Ciclo de Vida

O software passa por várias fases ao longo da sua existência. Ele é concebido, projetado, implementado, utilizado, avaliado, modificado e (eventualmente) substituído por um outro software "mais novo".

As fase da "vida" de um software podem ou não fazer parte de um ciclo. Um dos primeiros modelos a surgir sugeria a existência de um grande ciclo (BOEHM, 1976), o que era razoável levando-se em conta os recursos disponíveis para desenvolvimento de software. Hoje em dia existem modelos que se utilizam de prototipação, refinando-se o protótipo sucessivamente até chegar a um sistema completo ou utilizando-se o protótipo apenas como forma de especificação. Todos estes modelos não são formais, pois são resultantes apenas de uma organização do processo de produção de software, em alguns casos com uso de recursos de aumento de produtividade.

A formalização do desenvolvimento de software é recente, tendo surgido inicialmente na "programação em pequena escala". A formalização se dá através da criação de modelos e de regras de transformação (cuja aplicação constroe o software a partir do modelo especificado). A aplicação destas regras garante a equivalência formal entre a especificação e o software construído.

II.2 Especificação: Informal, Semi-Formal e Formal

Todas as fases pelas quais um software passa são importantes, porém quando algum erro é cometido nas fases iniciais, maiores são as suas conseqüências e maior é o seu custo de correção.

A primeira fase, denominada especificação, diz respeito à definição do que deve ser desenvolvido, ou seja, qual o objetivo do software. Uma especificação pode ser realizada de três formas: informal, semi-formal e formal.

Em uma especificação informal geralmente usamos um texto em linguagem natural para descrever os objetivos do software, as entradas previstas e os resultados esperados a partir destas entradas. Este tipo de especificação era muito comum até meados da década de 70, pois exigia apenas o conhecimento de uma linguagem natural. O principal problema observado em sistemas desenvolvidos a partir deste tipo de especificação era o tradicional "não era bem isto o que eu queria...". Por não ter uma semântica rigorosa a linguagem natural leva a várias interpretações, todas corretas, dependendo do ponto de vista.

Nos anos de 1976 e 1977 surgiram os primeiros métodos para tornar mais formal o desenvolvimento de software: PSL/PSA (TEICHROEW e HERSLEY, 1977), SADT (ROSS, 1977) e Análise Estruturada (GANE e SARSON, 1979), entre outros. Estes métodos abordam outras fases além da especificação. Eles fazem uso de símbolos e palavras-chave padronizados, o que leva a um aumento na qualidade da especificação. Mesmo

assim muitos erros podem ser introduzidos e não detectados devido a informalidade da combinação destes símbolos, ou seja, embora a sintaxe seja rigorosa, a semântica não o é. No Brasil algumas empresas tentaram usar estes métodos com pouco sucesso, devido ao grande número de informações que precisavam ser mantidas manualmente (em fichas !). Com o surgimento das ferramentas CASE ("Computer-Aided Software Engineering") este procedimento manual foi automatizado, tornando mais suave o trabalho dos analistas e permitindo uma maior difusão destes métodos.

O uso de técnicas semi-formais é satisfatório em muitos ambientes, porém existem situações onde nenhum erro pode ser tolerado. Estas aplicações críticas são recentes e freqüentemente o ciclo "implementação-ajuste-avaliação" (muito utilizado em empresas) não pode ser realizado (ex: controle do projeto "Starwars" dos EUA). Nestas situações o formalismo contribui no sentido de eliminar a introdução de erros durante a construção do software. No modelo proposto por Lehman [1984] existem duas fases: abstração, onde a definição informal do software a ser desenvolvido é refinada até ser transformada em uma especificação formal, e reificação, na qual esta especificação sofre transformações de modo a ser construído o software. Erros não são introduzidos durante a reificação, pois as regras de transformação para esta fase garantem matematicamente que o software contruído corresponde logicamente à especificação formal inicial.

II.3 O Problema da Verificação da Especificação

As três formas de especificação apresentadas quando aplicadas geram uma descrição dos objetivos e requisitos do sistema a ser desenvolvido. Conforme mencionado no item anterior, o custo de um erro na especificação é muito alto, portanto esta precisa ser verificada.

Todo software é projetado para atender às necessidades de um ou mais usuários. Este usuário sabe (ou pelo menos deveria saber) descrever completamente todos os requisitos do software. Cabe ao analista de sistemas entender estes requisitos e expressar a sua interpretação através de uma especificação. Portanto o usuário é o responsável pela verificação do entendimento do analista acerca do problema (especificicação).

Para o usuário entender uma especificação informal é simples, uma vez que ela é um texto em linguagem natural.

Em uma especificação semi-formal o usuário precisa conhecer os símbolos e palavras-chaves utilizados. A experiência mostra que, devido ao enfoque muito especializado das técnicas semi-formais (funcional, dados, temporal), nem sempre o usuário consegue entender completamente este tipo de especificação, porém em grande parte ela pode ser validada.

A notação fortemente matemática de uma especificação formal impede a sua verificação por usuários leigos (em matemática). A sintaxe e a semântica da lógica de primeira

ordem, utilizada tanto em especificações algébricas quanto em especificações baseadas em modelos, ambas formais, não é do uso cotidiano dos usuários. Paradoxalmente, embora as especificações formais tenham eliminado a introdução de erros na construção do software a partir da especificação, a dificuldade de entendimento desta pelo usuário pode comprometer todo um projeto.

No paradigma de programação automática proposto por Balzer em (BALZER, 1985), a primeira atividade a ser realizada é a "aquisição da especificação", onde é utilizada uma linguagem formal de especificação (Gist). Balzer acreditava que as especificações em Gist seriam de fácil leitura, porém na prática isto não ocorreu. Para resolver parcialmente este problema sua equipe criou um "parafraseador" (SWARTOUT, 1982), que traduz uma especificação em Gist para "linguagem natural" (inglês). Como vantagens da utilização deste recurso Balzer cita a apresentação da especificação com outro "ponto de vista", o que permitia encontrar erros na especificação.

O software VDM-TXT, tema desta tese, é um "parafraseador de Balzer" que tem por objetivo traduzir uma especificação formal em VDM, para linguagem natural (textos em português, tabelas etc), de tal forma que esta especificação possa ser verificada pelo usuário. Um uso secundário deste software, porém de grande utilidade, é que o próprio analista de sistemas pode verificar se a sua descrição formal corresponde ou não à sua interpretação do problema.

CAPITULO III: Especificação Formal em VDM

III.1 Apresentação

O VDM-TXT traduz especificações formais descritas segundo a linguagem proposta pelo VDM ("Vienna Definition Method"). O VDM é um método para desenvolvimento de sistemas grandes e complexos, desenvolvido na década de 70 pelo laboratório de pesquisa da IBM em Vienna. Ele é baseado em modelos e faz uso de semântica denotacional como base teórica.

O software é modelado através de dados (entradas, saídas e estados intermediários) e operações, que manipulam esses dados.

O uso de semântica denotacional em VDM teve originalmente por objetivo descrever a semântica de linguagens de programação. De fato, uma das primeiras utilizações de VDM foi no projeto de um compilador para ADA.

Existem outras formas de especificação formal, tais como especificação algébrica e especificação de sistemas concorrentes usando o modelo operacional (MENDES e DE AGUIAR, 1988) (COHEN et. al., 1986). A especificação formal baseada em modelos, da qual o VDM é o principal exemplo, foi a escolhida por atualmente ser a mais usada, tanto em ambientes comerciais quanto em ambientes acadêmicos, enquanto as outras formas ainda têm seu uso

restrito ao meio acadêmico.

Além destes fatores, o seu uso em ambiente comercial e a sua disseminação são fortemente estimulados pela IBM, o que leva a uma maior fonte de referências sobre VDM tais como livros, artigos e anais de encontros anuais, realizados principalmente na Europa.

Um exemplo completo de um sistema de biblioteca descrito em VDM encontra-se no apêndice C (O Sistema Exemplo).

III.2 A Linguagem do VDM

A linguagem para especificação formal proposta pelo VDM chama-se META-IV. Neste texto, "VDM" e "linguagem META-IV" possuem a mesma semântica.

A descrição será feita de forma narrativa, indicando os elementos da linguagem e seus objetivos. Seguindo cada descrição será apresentado um trecho de gramática em "estilo BNF", (com recursividade à esquerda) porém no formato necessário para seu uso no Gerador R*S (RANGEL NETTO, 1988), utilizado na implementação do VDM-TXT. A descrição completa da gramática utilizada encontra-se no apêndice A. A sintaxe utilizada na gramática encontra-se na monografia de Rangel Netto (RANGEL NETTO, 1988).

Os exemplos apresentados a seguir serão acumulativos, isto é, os exemplos são integrados e fazem referência entre si. Ao ser explicado o último elemento da linguagem será

apresentado um exemplo completo de uma descrição em VDM. Os exemplos serão apresentados mesmo que nem todos os elementos da linguagem exemplificados estejam descritos até o momento (neste caso eles são mencionados entre os sinais "-<" e ">-", que não são símbolos léxicos nem para o gerador R*S e nem para o VDM-TXT).

Será descrito como exemplo, e detalhado ao longo da apresentação da gramática do VDM, um trecho de um sistema de Folha de Pagamento, que mantém alguns dados sobre funcionários e departamentos de uma empresa e que tem uma operação de inclusão (cadastramento) de funcionários.

Exemplo (a):

```
SISTEMA FolhaDePagamento:


  TIPO Texto = CHAR-LIST
  TIPO MatrFunc = '5 letras seguidas e um digito'
  TIPO CodDepto = INT
  TIPO RegFunc =
    REGISTRO
      Nome: Texto;
      Salario: REAL;
      Depto: CodDepto;
    FIM
  TIPO RegDepto =
    REGISTRO
      Nome: Texto;
```

```
    Resp: MatrFunc;
    TotFunc: INT;
  FIM
TIPO FuncMap = MatrFunc <-> RegFunc
TIPO DeptoMap = CodDepto <-> RegDepto

ESTADO
  Funcionario   : FuncMap;
  Departamento  : DeptoMap;
  TotalSalarios: REAL;

OPERACAO IncluiFunc (MF       : MatrFunc;
                     Nome     : Texto;
                     Salario: REAL;
                     Depto    : CodDepto;)
  EXT
    Funcionario   : WR FuncMap;
    Departamento  : WR DeptoMap;
  PRE
    ((NAO (MF PERTENCE DOMINIO (Funcionario))) E
     (Salario > 0)                              E
     (Depto PERTENCE DOMINIO (Departamento)))
  POS
    ((Funcionario" = Funcionario +
     [MF <-> MK-Funcionario (Nome,Salario,Depto)]) E
     (SEJA tf = (TotFunc (Depto)) EM
      (tf" = tf + 1)))

INVARIANTE
```

```
(TODO d (
          (d PERTENCE DOMINIO (Departamento)) E
          (SEJA r = (Resp (d)) EM
            (r PERTENCE DOMINIO (Funcionario)))
        )
)

FIM
```

III.2.1 Descrição Geral

Um sistema é descrito em linhas gerais por 4 tipos de definição: estado, tipos, operações e invariante. Para o VDM-TXT estas definições podem ser realizadas em qualquer seqüência.

Gramática:

```
Sistema =
  'SISTEMA' Identificador ':' Definicoes 'FIM';

Definicoes =
  Definicoes Definicao
| ;

Definicao =
  DefEstado
| DefTipo
| DefOperacao
```

```
| DefInvariante :
```

Exemplo (b):

```
SISTEMA FolhaDePagamento:
  -<definicoes>-
FIM
```

No exemplo (b), o sistema "FolhaDePagamento" começa a ser definido.

O estado descreve os dados manipulados pelo sistema. Cada dado é especificado pelo seu nome e pelo seu tipo. Existe uma e apenas uma descrição de estado em um sistema.

Gramática:

```
DefEstado =
  'ESTADO' Campos ;
```

Exemplo (c):

```
SISTEMA FolhaDePagamento:
  ESTADO
    Funcionario   : FuncMap;
    Departamento  : DeptoMap;
  -<outras definicoes>-
FIM
```

No exemplo (c) são descritos os dados que serão manipulados: dados de funcionários e dados de departamento. Os tipos utilizados (FuncMap e DeptoMap) são definidos a seguir, no exemplo (d).

Um tipo é a associação de um identificador (o nome do tipo) a uma estrutura de tipos. Esta estrutura pode ser bem complexa, como veremos adiante, ou apenas fazer uso dos tipos primitivos da linguagem. A expressão indicada na gramática é do tipo "Identificador = Expressão". Podem existir várias descrições de tipo em um sistema. Para o VDM-TXT os tipos devem ser definidos antes de serem referenciados.

Gramática:

```
DefTipo =
  'TIPO' Expressao ;
```

Exemplo (d):

```
SISTEMA FolhaDePagamento:
  TIPO Texto = CHAR-LIST
  TIPO MatrFunc = '5 letras seguidas e um digito'
  TIPO CodDepto = {'Dep1','Dep3','Dep7','Dep2'}
  TIPO RegFunc =
    REGISTRO
      Nome: Texto;
      Salario: REAL;
      Depto: CodDepto;
    FIM
```

```
TIPO RegDepto =
  REGISTRO
    Nome: Texto;
    Resp: MatrFunc;
    TotFunc: INT;
  FIM
TIPO FuncMap = MatrFunc <-> RegFunc
TIPO DeptoMap = CodDepto <-> RegDepto
ESTADO
  Funcionario   : FuncMap;
  Departamento : DeptoMap;
 -<outras definicoes>-
FIM
```

No exemplo (d) são descritos todos os dados, e seus respectivos tipos, que serão manipulados pelo sistema exemplo FolhaDePagamento. Observe que as definições dos tipos utilizados pelo estado fazem referência a outros tipos, até que os tipos mais primitivos são definidos, ou através de tipos pré-definidos (vide Salario) ou via um comentário (vide MatrFunc). Ao final do item III.2.2, onde serão apresentados todos os tipos pré-definidos para o VDM-TXT, será detalhado o exemplo (d).

Após a definição dos dados que serão manipulados, são descritas as operações de manipulação destes dados. Podem existir várias descrições de operação em um sistema.

Gramática:

```
DefOperacao =
   'OPERACAO' Operacao ;
```

Exemplo (e): considerando o exemplo (d),

```
OPERACAO IncluiFunc (MF      : MatrFunc;
                     Nome    : Texto;
                     Salario: REAL;
                     Depto   : CodDepto;)
   EXT
     Funcionario   : WR FuncMap;
     Departamento : WR DeptoMap;
   PRE
     ((NAO (MF PERTENCE DOMINIO (Funcionario))) E
      (Salario > 0)                              E
      (Depto PERTENCE DOMINIO (Departamento)))
   POS
     ((Funcionario" = Funcionario +
     [MF <-> MK-Funcionario (Nome,Salario,Depto)]) E
      (SEJA tf = (TotFunc (Depto)) EM
       (tf" = tf + 1)))
```

No exemplo (e) é descrita a operação IncluiFunc do sistema exemplo FolhaDePagamento, considerando os dados, e seus respectivos tipos, definidos no exemplo (d). Os componentes da descrição de uma operação estão descritos no item III.2.4, onde o exemplo (e) é detalhado.

Por último é descrita uma fórmula lógica que representa propriedades do estado que não variam devido à aplicação das operações. Este invariante é utilizado na prova formal da correção das especificações das operações. Pode existir apenas uma especificação de invariante em um sistema.

Gramática:

```
DefInvariante =
  'INVARIANTE' Expressao
```

Exemplo (f): considerando o exemplo (d),

```
INVARIANTE
  (TODO d (
           (d PERTENCE DOMINIO (Departamento)) E
           (SEJA r = (Resp (d)) EM
             (r PERTENCE DOMINIO (Funcionario)))
          )
  )
```

No exemplo (f) é descrito invariante do sistema exemplo FolhaDePagamento, considerando os dados, e seus respectivos tipos, definidos no exemplo (d). A fórmula lógica que descreve o invariante está descrita no item III.2.5, onde o exemplo (f) e as pré e pós condições do exemplo (e) serão detalhadas.

III.2.2 Descrição de Dados

Os objetos que são tratados pelo VDM são conjuntos, mapeamentos, listas, registros e funções.

Existem alguns conjuntos pré-definidos e que são os tipos mais primitivos (os outros são estruturas que usam conjuntos). Conjuntos podem ser aqueles primitivos, intervalos, especificados por fórmulas lógicas ou resultado da aplicação de determinadas operações (em conjuntos e em outros tipos).

Gramática:

```
 Conjunto =
   'UNIAO' '{' Expressoes '}'              # União Distribuída
 ! '{' Expressoes '}'                      # Conjunto Explícito
 ! '{' Identificador '|' Expressao '}'     # Conjunto Fórmula
 ! '{' Expressao '..' Expressao '}'        # Conjunto Intervalo
 ! '{' '}';                                # Conjunto Vazio
```

Exemplo (a):

(a.1): UNIAO {{1,2,3},{4,5,6}}

(a.2): {a, b, c, d, e}

(a.3): {x | x PERTENCE {1,2,3,4,5,6} E x < 5}

(a.4): {-7..18}

(a.5): {}

O exemplo (a.1) mostra como um conjunto pode ser definido pela união de outros conjuntos; (a.2) mostra a

definição explícita de um conjunto; (a.3) mostra como um conjunto pode ser definido por uma fórmula lógica; (a.4) mostra um conjunto representado apenas por seus limites inferior e superior (admitindo-se a existência de um conjunto-base onde existe uma ordenação entre seus elementos); (a.5) é a representação de um conjunto vazio.

Os conjuntos pré-definidos são: Nat (inteiros positivos a partir do 0, exclusive), Nat0 (inteiros positivos a partir do 0, inclusive), Int (inteiros), Real (reais), Bool (valor lógico: verdadeiro ou falso), Dat (data) e Char (caractere: letras, dígitos, sinais de pontuação, símbolos especiais).

Mapeamentos são funções. Eles podem ser explícitos, onde todas as associações entre elementos do domínio e da imagem (contradomínio) são representadas, ou implícitos, onde o domínio é finito.

Gramática:

```
 Mapeamento =
   '[' Expressoes ']'                    # Mapeamento Explícito
 ! '[' Expressao '|' Expressao ']'       # Mapeamento Implícito
 ! '[' ']';                              # Mapeamento Vazio
```

Exemplo (b):

```
 (b.1): [ 'a' -> 1, 'b' -> 2 , 'c' -> 1]
 (b.2): [ x <-> 2*x | x PERTENCE {-3..5} ]
 (b.3): []
```

O exemplo (b.1) mostra um mapeamento definido explicitamente do tipo "vários para um"; (b.2) mostra um mapeamento definido implicitamente (via uma fórmula lógica) do tipo "um para um" (mapeamento bijetivo); (b.3) é a representação de um mapeamento vazio.

Listas são seqüências de elementos. Uma lista sempre é explícita, com a indicação de todos os seus elementos. Em geral são utilizados os delimitadores "<" e ">" em listas, porém devido à implementação escolhida, novos delimitadores foram criados.

Gramática:

```
 Lista =
   '<<' Expressoes '>>'                    # Lista Explícita
 ! '<<' '>>' ;                             # Lista Vazia
```

Exemplo (c):

(c.1): <2,4,6,8,10>

(c.2): <>

O exemplo (c.1) mostra uma lista definida explicitamente; (c.2) é a representação de uma lista vazia. e uma lista vazia.

A criação de objetos que agrupam elementos de naturezas distintas é realizada através da utilização de registros. Na definição de um registro são indicados todos os seus componentes (campos).

```
Gramática:
 Registro =
  'REGISTRO' Campos 'FIM';
 Campos =
  Campos ';' Campo
! Campo;
 Campo =
  Identificadores ':' Expressao
!  ;

Exemplo (d):
 REGISTRO
  C1: {1,2};
  C2: Nat0
 FIM
```

O exemplo (d) mostra um registro com dois campos: C1, que pode possuir um dos valores descritos no conjunto definido explicitamente com elementos 1 e 2, e C2, que pode possuir apenas números naturais, inclusive o 0.

Todos os tipos de dados acima descritos possuem <Expressao> (ou <Expressoes>) como elemento. Na descrição de <Expressao> temos produções que levam a conjuntos, mapeamentos, listas e registros, o que faz com que todos os tipos de dados possam ser combinados de forma a representar estruturas complexas.

Considere o exemplo (d) do item III.2.1:

Exemplo (III.2.1.d):

```
SISTEMA FolhaDePagamento:
  TIPO Texto = CHAR-LIST
  TIPO MatrFunc = '5 letras seguidas e um digito'
  TIPO CodDepto = {'Dep1','Dep3','Dep7','Dep2'}
  TIPO RegFunc =
    REGISTRO
      Nome: Texto;
      Salario: REAL;
      Depto: CodDepto;
    FIM
  TIPO RegDepto =
    REGISTRO
      Nome: Texto;
      Resp: MatrFunc;
      TotFunc: INT;
    FIM
  TIPO FuncMap = MatrFunc <-> RegFunc
  TIPO DeptoMap = CodDepto <-> RegDepto
  ESTADO
    Funcionario  : FuncMap;
    Departamento : DeptoMap;
  -<outras definicoes>-
FIM
```

Vamos detalhar a descrição dos dados do sistema exemplo FolhaDePagamento. Os tipos utilizados para definir o estado do sistema são: Texto, uma lista de caracteres;

CodDepto, um conjunto-tipo definido explicitamente: RegFunc, um registro-tipo, com 3 campos: Nome, Salario e Depto (cada um com seu tipo); RegDepto, também um registro-tipo, com 3 campos: Nome, Resp e TotFunc; FuncMap, um mapeamento-tipo bijetivo entre a matrícula de um funcionário (MatrFunc) e seus dados (RegFunc); DeptoMap, também um mapeamento-tipo bijetivo, entre o código de um departamento (CodDepto) e seus dados (RegDepto).

Os dados que o sistema manipula são definidos pelo seu estado: Funcionario, que é um mapeamento do tipo FuncMap, e Departamento, que é um mapeamento do tipo DeptoMap.

III.2.3 Descrição de Expressões

Existem três tipos principais de expressão:

* Lógica, que tem operandos do tipo lógico (Bool), operadores lógicos e resultado lógico;

* Aritmética, que tem operandos numéricos, operadores aritméticos e resultado numérico;

* Relacional, que tem operandos do mesmo tipo, operadores de comparação e resultado lógico.

Além destes, existem expressões envolvendo conjuntos, mapeamentos e listas, com operadores próprios e diversos

tipos de resultado.

Existe um grupo especial de expressões que trata de fórmulas lógicas de primeira ordem. Este grupo será descrito no item III.2.5 (Descrição de Fórmulas).

De forma a implementar precedência entre operadores, a definição de expressão na gramática gerada é:

```
Expressoes =
  Expressoes ',' Expressao
! Expressao;

Expressao =
  Expressao   Oper_Bin_Pri_A Expressao_A
! Expressao_A;
Expressao_A =
  Expressao_A Oper_Bin_Pri_B Expressao_B
! Expressao_B;
Expressao_B =
  Oper_Un_A_Esquerda_Pri_A Expressao_C
! Expressao_C;
Expressao_C =
  Expressao_C Oper_Bin_Pri_C Expressao_D
! Expressao_D;
Expressao_D =
  Expressao_D Oper_Bin_Pri_D Expressao_E
! Expressao_E;
Expressao_E =
  Expressao_E Oper_Bin_Pri_E Expressao_F
```

```
! Expressao_F;
Expressao_F =
  Oper_Un_A_Esquerda_Pri_B Expressao_G
! Expressao_G;
Expressao_G =
  Expressao_G Oper_Un_A_Direita
! Expressao_H;
Expressao_H =
  Registro
! Conjunto
! Mapeamento
! Lista
! Formula
! Identificador
! Constante
! '(' Expressao ')';

Constante =
  'CT_Num'
! 'CT_Lit';

Identificador =
  'ID';
Identificadores =
  Identificadores ',' Identificador
! Identificador;
```

Os operadores das expressões do tipo lógico,

aritmético e relacional são os tradicionais. Foram introduzidos operadores que atuam sobre operandos do tipo conjunto, mapeamento e lista.

Alguns operadores estão "sobrecarregados", ou seja, a sua semântica depende do tipo dos operandos sobre os quais atuam (exs: "+", "-", "*", "/"). Os operadores "unários à direita" (Oper_Un_A_Direita) na verdade são construtores de conjuntos e listas.

A descrição completa da semântica dos operadores encontra-se em (MENDES e AGUIAR, 1988). Algumas simplificações e adaptações foram realizadas sobre a sintaxe proposta nesta referência:

- Conjuntos

  Alguns operadores, tanto binários quanto unários, foram representados por extenso, e não através dos símbolos convencionais: "UNIAO", "INTERSECAO", "PERTENCE", "CONTIDO" e "CONTEM"; a união distribuída e' indicada pelo operador "UNIAO"; subtipos numéricos e outros que indiquem um intervalo de um tipo primitivo são indicados pela construção "[X .. Y]", onde X e' o limite inferior e Y o limite superior, ambos inclusos; o conjunto vazio e' representado por "{}".

- Mapeamentos

  Os operadores "DOMINIO" e "IMAGEM" substituem

"dom" e "rng", respectivamente; o operador "escrever-sobre" é representado por "+".

- Listas

Os delimitadores de início e término de lista foram substituídos de "<" e ">" para "<<" e ">>", respectivamente; os operadores "INDICES" e "ELEMENTOS" substituem "inds" e "elems", respectivamente; o operador de concatenação de listas é representado por "+", e nao "^".

- Registros

A definição dos campos que compoem um registro é delimitada pelas "palavras-reservadas" "REGISTRO" e "FIM".

- Lógica de Predicados

Os símbolos que representam os quantificadores foram substituídos por "TODO", "EXISTE" e "UNICO"; o operador "SEJA .. EM" substitui o operador "let .. in".

```
Oper_Bin_Pri_A = /
  'E'                          # Lógico
! 'OU'                         # Lógico
! '=>'                         # Lógico
```

```
! '<=>';                          # Lógico
Oper_Bin_Pri_B =
  '<'                             # Relacional
! '='                             # Relacional
! '>'                             # Relacional
! '<='                            # Relacional
! '>='                            # Relacional
! '<>'                            # Relacional
! 'PERTENCE'                     # Conjunto
! 'CONTIDO'                      # Conjunto
! 'CONTEM';                      # Conjunto
Oper_Bin_Pri_C =
  '+'                             # Aritmético/Mapeamento/Lista
! '-'                             # Aritmético/Conjunto
! 'UNIAO';                       # Conjunto
Oper_Bin_Pri_D =
  '*'                             # Aritmético/Conjunto
! '/'                             # Aritmético/Mapeamento
! '\'                             # Mapeamento
! 'INTERSECAO';                  # Conjunto
Oper_Bin_Pri_E =
  '->'                            # Mapeamento
! '<->';                         # Mapeamento
Oper_Un_A_Esquerda_Pri_A =
  'NAO';                          # Lógico
Oper_Un_A_Esquerda_Pri_B =
  'DOMINIO'                       # Mapeamento
! 'IMAGEM'                       # Mapeamento
! 'INDICES'                      # Lista
```

```
! 'ELEMENTOS'                    # Lista
! '+'                            # Aritmético
! '-';                           # Aritmético
Oper_Un_A_Direita =
  '-SET'                         # Conjunto
! '-LIST' ;                      # Lista
```

I.II.2.4 Descrição de Operações

As operações em VDM são responsáveis pela manipulação dos dados. Elas possuem obrigatoriamente um nome, forma(s) de acesso aos dados, uma pré-condição e uma pós-condição. O seu formato genérico é:

```
Operacao =
  Identificador
  Parametros
  Tipo_Resultado
  Def_Externas
  Pre
  Pos ;
```

O acesso aos dados pode ser de três formas: parâmetros, campos externos ou tornando a operação uma função. O acesso para "entrada" (leitura) de dados pode

ser via parâmetros, campos externos ou ativação de uma função; o acesso para "saída" (gravação) pode ser via campos externos ou a atribuição de um valor ao retorno de uma função (Tipo_Resultado).

```
Parametros =
  '(' Campos ')'
!  ;

Tipo_Resultado =
   ':' Expressao
!  ;

Def_Externas =
  'EXT' Externas
!  ;
Externas =
  Externas ':' Externa
! Externa;
Externa =
  Identificadores ':' Modo Expressao
!° ;
Modo =
  'WR'
! 'RD' ;
```

A especificação formal do objetivo de uma operação é

expressa através de duas condições: a pré-condição e a pós-condição. Ambas são descritas em termos de fórmulas lógicas de primeira ordem.

A pré-condição indica que a operação será "executada" (ativada) somente se determinadas características estiverem presentes nos dados aos quais a operação tem acesso para leitura (campos externos, parâmetros, funções).

Uma vez que a operação possa ser executada, o efeito da sua ativação será indicado através de características que passarão a estar presentes nos dados aos quais a operação tenha acesso para gravação (campos externos, função), que são indicados pela pós-condição.

```
Pre =
  'PRE' Expressao ;
Pos =
  'POS' Expressao ;
```

Considere a operação descrita no exemplo (e) do item III.2.1:

```
Exemplo (a):
  OPERACAO IncluiFunc (MF      : MatrFunc;
                       Nome    : Texto;
                       Salario: REAL;
                       Depto   : CodDepto;)
```

```
EXT
  Funcionario   : WR FuncMap;
  Departamento  : WR DeptoMap;
PRE
  ((NAO (MF PERTENCE DOMINIO (Funcionario))) E
   (Salario > 0)                              E
   (Depto PERTENCE DOMINIO (Departamento)))
POS
  ((Funcionario" = Funcionario +
  [MF <-> MK-Funcionario (Nome,Salario,Depto)]) E
   (SEJA tf = (TotFunc (Depto)) EM
    (tf" = tf + 1)))
```

No exemplo (a) é descrita a operação IncluiFunc do sistema exemplo FolhaDePagamento, considerando os dados e seus respectivos tipos definidos no exemplo (III.2.1.d). Esta operação não é uma função e possui como parâmetros MF, Nome, Salario e Depto, de tipos MatrFunc, Texto, REAL e CodDepto, respectivamente. Os campos externos aos quais ela tem acesso para leitura e gravação são Funcionario e Departamento. O detalhamento da pré e da pós-condição será realizado no item III.2.5, onde a estrutura de fórmulas lógicas é detalhada.

III.2.5 Descrição de Fórmulas

As fórmulas lógicas em VDM são baseadas em lógica de

predicados de primeira ordem, com algumas extensões: uma diz respeito à substituição de um termo em uma fórmula por outra fórmula (SEJA) e a outra permite a manipulação de registros, no que diz respeito à sua construção e ao acesso aos seus campos.

```
Formula =
  'SE' '(' Expressao ')'
  'ENTAO' '(' Expressao ')'
  'SENAO' '(' Expressao ')'

! Quantificador Identificador '(' Expressao ')'

! 'SEJA' Identificador
  Oper_Bin_Pri_B '(' Expressao ')'
  'EM' '(' Expressao ')'

! Identificador '(' Expressoes ')'

! 'MK-' Identificador '(' Expressoes ')'

! 'VERDADE'
! 'FALSO';

Quantificador =
  'TODO'
! 'EXISTE'
! 'UNICO' ;
```

Considere a pré-condição do exemplo (a):

Exemplo (b):

```
PRE
   ((NAO (MF PERTENCE DOMINIO (Funcionario))) E
    (Salario > 0)                              E
    (Depto PERTENCE DOMINIO (Departamento)))
```

No exemplo (b), a pré-condição da operação incluiFunc só será verdadeira se a matrícula do funcionário ainda não estiver cadastrada (ou seja, não pertencer ao domínio do mapeamento Funcionario), se o salário informado para o funcionário for maior que 0 e se o departamento ao qual o funcionário será alocado já estiver cadastrado (ou seja, pertencer ao domínio do mapeamento Departamento).

Considere a pós-condição do exemplo (a):

Exemplo (c):

```
POS
   ((Funcionario" = Funcionario +
        [MF <-> MK-Funcionario (Nome,Salario,Depto)]
E
    (SEJA tf = (TotFunc (Depto)) EM
     (tf" = tf + 1)))
```

No exemplo (c), a pós-condição da operação IncluiFunc só será verdadeira assim que ao mapeamento Funcionario for acrescentada a associação entre a matrícula do funcionário e seus dados (um registro constituído de Nome, Salario e Depto) e ao total de funcionários do departamento onde o funcionário será alocado for acrescida uma unidade (observe que nesta expressão o total de funcionários foi abreviado pelo termo "tf".

Considere o invariante descrito no exemplo (e) do item III.2.1:

Exemplo (d):

```
INVARIANTE
  (TODO d (
           (d PERTENCE DOMINIO (Departamento)) E
           (SEJA r = (Resp (d)) EM
             (r PERTENCE DOMINIO (Funcionario)))
          )
  )
```

No exemplo (d), o invariante do sistema FolhaDePagamento só será verdadeiro se, para todo departamento cadastrado (indicado na fórmula pelo seu código d), o seu responsável também estiver cadastrado como funcionário (ou seja, r pertencer ao domínio do mapeamento Funcionario, onde r é o responsável do departamento d).

CAPITULO IV: O Tradutor VDM-TXT

O software VDM-TXT tem por objetivo traduzir uma especificação formal em VDM para linguagem natural. Ele recebe como entrada um arquivo fonte com a descrição em VDM de um sistema, segundo a sintaxe descrita no item III.2, e gera um arquivo de mesmo nome com a extensão DOC, contendo um texto em linguagem natural. Caso ocorram erros de compilação, o arquivo com extensão DOC não será gerado e o erro será apresentado no vídeo.

Existem duas fases principais no VDM-TXT: a compilação da descrição VDM e a geração de texto. A compilação (sem erros) produz uma árvore semântica, que é percorrida e tratada pela fase de geração de texto.

O exemplo de um texto pelo VDM-TXT gerado a partir da descrição do sistema exemplo (apêndice C) encontra-se no apêndice E (O Texto Exemplo).

IV.1 Compilação

Toda compilação possui 3 grandes fases de análise: léxica, sintática e semântica. Para efetuar as análises léxica e semântica foram desenvolvidas "units" específicas para VDM; para efetuar a análise sintática foi utilizado o Sistema de Geração de Analisadores Sintáticos R*S Simples (RANGEL NETTO, 1988).

IV.1.1 O Gerador R*S

Durante o projeto do compilador existiam duas opções de desenvolvimento: preparar um analisador sintático específico para o VDM ("hard-wired") ou utilizar um gerador de analisador sintático. Como a ênfase da tese é na geração de texto, e não no desempenho das técnicas de compilação envolvidas, e a implementação da compilação deveria ser a mais rápida possível, optou-se por utilizar um gerador.

O gerador R*S (RANGEL NETTO, 1988) prepara, a partir da gramática a ser analisada sintaticamente, várias tabelas que indicam a seqüência de "tokens" permitida. Estas tabelas são utilizadas por uma "unit" padrão (VDM_SIN) que controla todo o processo de compilação. As tabelas geradas, que estão no arquivo VDM.TAB, foram transformadas na "unit" VDM_TAB.

Uma das dificuldades no uso deste gerador é que, como ele possui um "look ahead" de apenas um "token", muitas verificações foram transferidas para o nível semântico, de modo a eliminar as ambigüidades da gramática.

IV.1.2 Análise Léxica

A "unit" responsável pela análise léxica chama-se VDM_LEX, e o seu objetivo é classificar as cadeias de sinais, caracteres e dígitos do arquivo fonte VDM em "tokens".

Ela utiliza a "unit" VDM_TKST, que tem por objetivo carregar a tabela de "tokens" VDM_TKST.TAB, permitir a busca (rápida) de uma cadeia de caracteres para verificar se ela é uma palavra reservada (TKS_Scan_PalRes) e retornar dados de um "token" dada a sua posição nesta tabela (TKS_Scan_Token). Neste arquivo de "tokens" cada um possui dados como o "token" em si, o seu nome e se ele é uma palavra reservada ou não.

Este analisador considera todos os caracteres entre sinais de exclamação como comentários. Ele é ativado pelo analisador sintático e retorna para este o token (Simb) e a cadeia de caracteres a ele associado (NomeSimb), caso o token seja um identificador (T_ID), uma constante alfanumérica (T_CT_Lit) ou uma constante numérica (T_CT_Num).

O par Simb/NomeSimb representa o símbolo mais recente lido no arquivo fonte; o par Token/NomeToken representa o símbolo anterior a este.

O conteúdo do arquivo de "tokens" VDM_TKST.TAB encontra-se no apêndice B.

IV.1.3 Análise Sintática

A análise sintática é o "coração" da compilação e fisicamente corresponde à "unit" VDM_SIN. O seu objetivo é validar a seqüência de "tokens" de um arquivo fonte VDM.

Ela aciona rotinas das "units" VDM_LEX e VDM_SEM

(análise semântica) e utiliza as tabelas definidas na "unit" VDM_TAB.

A análise corresponde a acionar o analisador léxico e verificar se o "token" deve ser empilhado ou reduzido, via tabelas do VDM_TAB. A redução indica que o analisador sintático conseguiu identificar uma expressão da linguagem gerada pela gramática VDM (neste caso o analisador semântico é ativado) e o empilhamento indica que o "token" é válido na seqüência porém não caracteriza uma expressão. Caso não ocorra nem o empilhamento nem a redução ocorre um erro sintático.

IV.1.4 Análise Semântica

A análise semântica tem por objetivo verificar se uma seqüência válida de "tokens" (segundo a visão do analisador sintático) faz sentido ou não. Implementada na "unit" VDM_SEM, ela cria uma árvore semântica contendo toda a estrutura lógica da descrição de um sistema em VDM. A análise semântica realiza uma série de verificações que foram transferidas do nível sintático para o nível semântico, conforme mencionado no item IV.1.1.

A partir do número que identifica uma produção (Prod) informado pela "unit" VDM_SIN e do penúltimo símbolo identificado pelo VDM_LEX (Token/NomeToken), o analisador semântico cria um nó na árvore semântica. Um nó pode ter de 0 a 6 "filhos", dependendo do seu tipo.

Se o "token" for um identificador ou uma constante, a cadeia de caracteres correspondente (NomeToken) é procurada em uma tabela de símbolos, via "hashing" com tratamento de colisão por uma tabela de "overflow"; se ela não for encontrada é criado um nó na tabela de símbolos, onde a cadeia de caracteres é incluída na sua versão original e também com todos os caracteres em "caixa alta" (para facilitar buscas posteriores).

Um nó da árvore semântica recém criado é colocado em uma pilha semântica, de tal forma que os próximos nós possam tê-lo como "filho". Ao final da análise sintática deve existir apenas um nó na pilha semântica, correspondente à raiz da árvore semântica.

O nó da árvore semântica possui o seguinte formato:

```
NoArvSem=
   record
      cod:cods;
      ult:boolean;
      prim,
      prox,
      tipo:pArvSem;
      case cods of
         identificador:
            (pnom,
             nival:integer);
   end;
```

Cod é o código do tipo do nó; ult indica se este é ou não o último nó de uma lista encadeada de nós "irmãos"; Prim aponta para o primeiro filho deste nó; Prox aponta para o próximo irmão ou, caso ele seja o último irmão, para o seu pai; caso o nó represente uma expressão, variável, constante ou uma função, Tipo aponta para o nome do tipo, se existir (ou para sua estrutura, caso contrário). Caso o nó seja de um identificador, Pnom aponta para a cadeia de caracteres correspondente e nival pode guardar o seu nível, o valor inteiro de um int ou Scalar, o valor char de um Car.

O nô da tabela de símbolos possui o seguinte formato:

```
noTS=
   record
      niv    : byte;
      prim   : pArvSem;
      prox   : pTS;
      postxt: longint;
      linhas: integer;
   end;
```

Os tipos primitivos de dados ('Registro!','Mapeamento!','Conjunto!', 'Lista!' e 'Numero!'), bem como os conjuntos pré-definidos para o VDM (Nat, NatO, Int, Real, Bool, Dat e Char) são incluídos na tabela de símbolos como constantes durante a inicialização da "unit" VDM_SEM.

IV.2 Geração de Texto

Nesta fase é gerado um texto em linguagem natural que tem por objetivo apresentar, da forma mais adequada possível, uma descrição VDM para um usuário.

Muitas das decisões acerca da forma do texto a ser gerado foram baseadas nas seguintes premissas:

- o usuário não tem conhecimento de VDM;

- o usuário deve ser capaz de, uma vez lido o texto gerado pelo VDM-TXT, decidir se o sistema em questão está ou não completamente descrito;

- o texto deve ser organizado e possuir referências que auxiliem o usuário na sua leitura, uma vez que o usuário não deverá ter muito contato com textos gerados pelo VDM-TXT (dificilmente ele é usuário de muitos sistemas);

- o analista deve ser capaz de introduzir dinamicamente observações no texto gerado de forma a elucidar determinados conceitos.

Tendo em mente estas premissas, o texto é gerado a partir da árvore semântica criada pela fase de compilação. A "unit" que possui as rotinas responsáveis por esta

geração é a VDM_TXT.

IV.2.1 Ordenação da Árvore Semântica

Antes da geração de texto propriamente dita, é feita uma ordenação entre as definições do sistema.

Conforme apresentado no item III.2.1, a descrição geral de um sistema possui 4 tipos de definição: Estado, Tipos, Operações e Invariante. O analista pode fazer definições em qualquer ponto do texto e em qualquer seqüência. Embora um analista com bom senso não vá fazer definições em uma seqüência aleatória, achamos por bem tornar esta seqüência objetiva do ponto de vista do usuário (do ponto de vista do analista, a sua seqüência de definições sempre será a mais sensata ...).

O critério de ordenação adotado é:

- Da primeira definição para a última: Estado, Tipos, Operações e Invariante;

- As definições de tipos são ordenadas ascendentemente segundo a seqüência lexicográfica dos identificadores de tipo;

- As definições de operações são ordenadas ascendentemente segundo a seqüência lexicográfica dos nomes das operações.

Desta forma a referência a qualquer um dos elementos da descrição é facilitada, permitindo ao usuário buscar estes elementos de forma semelhante ao acesso de palavras em um dicionário.

Como todas as definições são filhas imediatas da raiz da árvore semântica (nó Sistema), a implementação da ordenação é trivial.

IV.2.2 Textos Complementares para Identificadores

Identificadores de uma forma geral podem ter a eles associado um texto. Este recurso permite ao analista elucidar conceitos ou acrescentar observações ao texto gerado pelo VDM-TXT, de forma dinâmica.

O procedimento utilizado para implementar este mecanismo é:

1. O analista define que identificadores devem ter que textos, criando via qualquer editor de textos um arquivo seqüencial comum VDM_TXT.TXT, com o seguinte formato:

```
.Identificador 1
Texto...
   .
   .
   .
Texto...
```

.Identificador N

2. O analista executa o programa VDM_BIB, que le o arquivo VDM_TXT.TXT e cria um arquivo seqüencial, porém de acesso direto, chamado VDM_TXT.BIB;

3. O VDM-TXT, assim que entra na fase de geração de texto, atualiza a tabela de símbolos, criada pela análise semântica, incluindo os identificadores descritos no arquivo VDM_TXT.BIB e prepara um ponteiro e um contador de linhas para o texto em si;

4. No momento de apresentar textos, o VDM_TXT utiliza alguns identificadores pré-definidos que têm a eles associados textos que descrevem o VDM em si e quando alguns tipos de informação são descritos.

A apresentação dos identificadores pré-definidos e dos tipos de informação aos quais um texto pode ser associado será descrita a seguir.

IV.2.3 Estrutura da Descrição

A estrutura da descrição em linguagem natural gerada pelo VDM-TXT é:

1. Introdução ao VDM

Texto de introdução ao VDM, que tem por objetivo apresentar o método ao usuário. O VDM-TXT utiliza o identificador "IntroducaoAoVDM" para apresentar o texto desta introdução.

2. Definição de Sistema

Texto que apresenta as principais partes da descrição de um sistema. O VDM-TXT utiliza o identificador "DefinicaoDeSistema" para apresentar este texto.

3. Descrição do sistema X

Texto informal que descreve os objetivos do sistema que está sendo apresentado (X). O VDM-TXT utiliza o identificador X (nome do sistema) para apresentar este texto.

4. Definição de Estado

Texto que descreve o que é o estado de um sistema, via identificador "DefinicaoDeEstado", e apresentação do estado do sistema X.

5. Definição de Tipo

Texto que descreve o que são os tipos, via identificador "DefinicaoDeTipo", e apresentação dos tipos pré-definidos e dos tipos definidos para o sistema X.

6. Definição de Operação

Texto que descreve o que são as operações, via identificador "DefinicaoDeOperacao", e apresentação das operações definidas para o sistema X.

7. Definição de Invariante

Texto que descreve o que é o invariante de um sistema, via identificador "DefinicaoDeInvariante", e apresentação do invariante do sistema X.

IV.2.4 Descrição do Estado

A apresentação da descrição em linguagem natural de um estado está baseada na rotina "MostraEstrutura", da "unit" VDM_TXT.

Em última análise, a descrição sempre será de campos. A MostraEstrutura apresenta o nome do campo, o seu tipo e o texto complementar associado ao nome do campo. Em alguns casos esta rotina é chamada recursivamente, permitindo a descrição de campos com tipos compostos.

O texto gerado para descrever um estado, a partir do exemplo (c) do item III.2.1 e':

- Funcionario :FuncMap
- Departamento :DeptoMap
- TotalSalarios :Real

IV.2.5 Descrição dos Tipos

Análoga à descrição de estado, também utiliza a rotina MostraEstrutura.

Convém colocar no texto do identificador "DefinicaoDeTipo", que é apresentado antes desta descrição, a relação dos tipos pré-definidos, de forma que o usuário tenha uma relação completa dos tipos disponíveis.

O texto gerado para descrever os tipos (exceto o texto associado a "DefinicaoDeTipos"), a partir do exemplo (d) do item III.2.1 e':

```
Tipos definidos para este sistema:
   .CodDepto e' Int
   .DeptoMap e' composto , formado pelo mapeamento
      CodDepto <-> RegDepto
```

.FuncMap e' composto , formado pelo mapeamento

MatrFunc <-> RegFunc

.MatrFunc e' 5 letras seguidas e um digito

.RegDepto e' composto

- Nome :Texto
- Resp :MatrFunc
- TotFunc :Int

.RegFunc e' composto

- Nome :Texto
- Salario :Real
- Depto :CodDepto

.Texto e' composto de Char

IV.2.6 Descrição das Operações

A apresentação de uma operação possui duas partes: cabeçalho e fórmulas.

Na apresentação do cabeçalho são descritos: os parâmetros e seus tipos; se a operação é uma função ou não (e neste caso o tipo da informação retornada); se é feito uso de variáveis externas do estado e o seu modo de utilização (leitura/gravação); a apresentação do texto complementar associado ao nome da operação.

A apresentação da pré-condição e da pós-condição está baseada na rotina "MostraFormula", da "unit" VDM_TXT. Esta rotina descreve uma fórmula de lógica de primeira ordem através da descrição dos seus termos e operadores

primitivos. A fórmula é decomposta hierarquicamente em sub-fórmulas numeradas, e assim recursivamente, até chegar aos termos e operadores primitivos.

O texto gerado para descrever a operação IncluiFunc, a partir do exemplo (e) do item III.2.1 e':

IncluiFunc

A operacao IncluiFunc possui os seguintes parametros e seus respectivos tipos: MF (tipo MatrFunc), Nome (tipo Texto), Salario (tipo Real) e Depto (tipo CodDepto).

O uso do estado e' feito atraves da gravacao do campo Funcionario (tipo FuncMap) e gravacao do campo Departamento (tipo DeptoMap).

A pre-condicao desta operacao tem o resultado definido por F1.

Em F1, o resultado e' verdadeiro se (F2 e F3) e F4).

Em F2, nao "MF pertence ao dominio de Funcionario".

Em F3, "Salario e' maior que 0".

Em F4, "Depto pertence ao dominio de Departamento".

A pos-condicao desta operacao tem o resultado definido por F5.

Em F5, o resultado e' verdadeiro se (F6 e F7). Em F6, "o novo conteudo de Funcionario e' igual a " Funcionario mais mapeamento definido pela associacao "MF <->

registro de Funcionario, composto de Nome, Salario e Depto""".

Na formula F7, "o novo conteudo de tf e' igual a "tf mais 1"", onde tf e' igual a TotFunc de Depto.

IV.2.7 Descrição do Invariante

A náloga à descrição de fórmulas na descrição das operações, também utiliza a rotina MostraFórmula, neste caso para apresentar a sua única fórmula.

O texto gerado para descrever o invariante do sistema FolhaDePagamento, a partir do exemplo (f) do item III.2.1 e':

O invariante desta operacao tem o resultado definido por F1.

A formula F1 e' verdadeira se para todo d, o resultado e' verdadeiro se (F2 e F3).

Em F2, "d pertence ao dominio de Departamento".

Na formula F3, "r pertence ao dominio de Funcionario", onde r e' igual a Resp de d.

IV.3 Arquitetura

O VDM-TXT foi codificado em TURBO Pascal 5.0, usando "units":

- VDM_LEX

  Rotinas de análise léxica; usa VDM_TKST.

- VDM_SIN

  Rotinas de análise sintática; usa VDM_LEX, VDM_TAB, VDM_TKST e VDM_SEM.

- VDM_SEM

  Rotinas de análise semântica; usa VDM_LEX;

- VDM_TAB

  Possui as tabelas utilizadas pelo VDM_SIN, que são criadas pelo gerador R*S.

- VDM_TKST

  Rotinas para carregar e buscar a tabela de tokens VDM_TKST.TAB. Os tokens definidos como tipos do Pascal estão no arquivo VDM.TKS, que é gerado pelo programa VDM_TKS a partir da listagem criada pelo gerador R*S.

- VDM_TXT

  Rotinas de preparação e geração de texto; usa VDM_LEX e VDM_SEM.

Estas "units" são ativadas principalmente pelo programa principal de tradução: VDM. Existe apenas um programa auxiliar (VDM_BIB), que tem por objetivo criar a partir do arquivo texto VDM_TXT.TXT o arquivo de acesso direto VDM_TXT.BIB, que é utilizado pela "unit" VDM_TXT.

CAPITULO V: Conclusões

A capacidade do VDM-TXT em gerar um texto em linguagem natural a partir de uma especificação em VDM e de outros textos descritivos auxiliares torna viável a utilização de VDM em ambientes "não-acadêmicos", pelo menos no que diz respeito à validação da especificação pelo usuário final.

O analista de sistemas que atualmente especifica sistemas com técnicas semi-formais deve ser preparado para utilizar VDM, e neste caso o VDM-TXT também pode ser útil, mostrando a este analista o que ele mesmo definiu, porém em linguagem natural.

Muita pesquisa tem sido realizada em VDM, principalmente na Europa. Em 1988, quando esta tese foi iniciada, dentre os diversos relatórios e artigos publicados sobre VDM (BJORNER, 1987a) (BJORNER, 1987b) não existia nenhum tradutor de VDM para linguagem natural, e sim experiências com outros formalismos para gerar automaticamente protótipos a partir de especificações, também visando a validação da especificação.

O VDM-TXT é apenas um "degrau", importante por ser um dos primeiros, no sentido da difusão da utilização do VDM como linguagem para especificação formal.

APÊNDICE A: Gramática do VDM

```
Sistema =
  'SISTEMA' Identificador ':' Definicoes 'FIM';
Definicoes =
  Definicoes Definicao
!   ;
Definicao =
  DefEstado
! DefTipo
! DefOperacao
! DefInvariante ;
DefEstado =
  'ESTADO' Campos ;
DefTipo =
  'TIPO' Expressao ; #ID '=' Exp
DefOperacao =
  'OPERACAO' Operacao ;
DefInvariante =
  'INVARIANTE' Expressao ;

#-----------------------------------------------------------#

Campos =
  Campos ':' Campo
! Campo;
Campo =
  Identificadores ':' Expressao
```

```
!  ;

Expressao =
  Expressao   Oper_Bin_Pri_A Expressao_A
! Expressao_A;
Expressao_A =
  Expressao_A Oper_Bin_Pri_B Expressao_B
! Expressao_B;
Expressao_B =
  Oper_Un_A_Esquerda_Pri_A Expressao_C
! Expressao_C;
Expressao_C =
  Expressao_C Oper_Bin_Pri_C Expressao_D
! Expressao_D;
Expressao_D =
  Expressao_D Oper_Bin_Pri_D Expressao_E
! Expressao_E;
Expressao_E =
  Expressao_E Oper_Bin_Pri_E Expressao_F
! Expressao_F;
Expressao_F =
  Oper_Un_A_Esquerda_Pri_B Expressao_G
! Expressao_G;
Expressao_G =
  Expressao_G Oper_Un_A_Direita
! Expressao_H;
Expressao_H =
  Registro
! Conjunto
```

```
! Mapeamento
! Lista
! Formula
! Identificador
! Constante
! '(' Expressao ')';

#-------------------------------------------------------------#

Oper_Bin_Pri_A =
   'E'                          # Logico
! 'OU'                         # Logico
! '=>'                         # Logico
! '<=>';                       # Logico
Oper_Bin_Pri_B =
   '<'                          # Relacional
! '='                          # Relacional
! '>'                          # Relacional
! '<='                         # Relacional
! '>='                         # Relacional
! '<>'                         # Relacional
! 'PERTENCE'                   # Conjunto
! 'CONTIDO'                    # Conjunto
! 'CONTEM';                    # Conjunto
Oper_Bin_Pri_C =
   '+'                          # Aritmetico/Mapeamento/Lista
! '-'                          # Aritmetico/Conjunto
! 'UNIAO';                     # Conjunto
Oper_Bin_Pri_D =
```

```
  '*'                          # Aritmetico/Conjunto
! '/'                          # Aritmetico/Mapeamento
! '\'                          # Mapeamento
! 'INTERSECAO';                # Conjunto
Oper_Bin_Pri_E =
  '->'                         # Mapeamento
! '<->';                       # Mapeamento
Oper_Un_A_Esquerda_Pri_A =
  'NAO';                       # Logico
Oper_Un_A_Esquerda_Pri_B =
  'DOMINIO'                    # Mapeamento
! 'IMAGEM'                     # Mapeamento
! 'INDICES'                    # Lista
! 'ELEMENTOS'                  # Lista
! '+'                          # Aritmetico
! '-';                         # Aritmetico
Oper_Un_A_Direita =
  '-SET'                       # Conjunto
! '-LIST' ;                    # Lista

#-------------------------------------------------------------#

Registro =
  'REGISTRO' Campos 'FIM';
Conjunto =
  'UNIAO' '{' Expressoes '}'
! '{' Expressoes '}'
! '{' Identificador '|' Expressao '}'
! '{' Expressao '..' Expressao '}'
```

```
! '{' '}';
Mapeamento =
  '[' Expressoes ']'
! '[' Expressao '|' Expressao ']'
! '[' ']';
Lista =
  '<<' Expressoes '>>'
! '<<' '>>' ;

#-------------------------------------------------------------#

Formula =
  'SE' '(' Expressao ')'
  'ENTAO' '(' Expressao ')'
  'SENAO' '(' Expressao ')'
! Quantificador Identificador '(' Expressao ')'
! 'SEJA' Identificador Oper_Bin_Pri_B '(' Expressao ')'
  'EM' '(' Expressao ')'
! Identificador '(' Expressoes ')'
! 'MK-' Identificador '(' Expressoes ')'
! 'VERDADE'
! 'FALSO';
Quantificador =
  'TODO'
! 'EXISTE'
! 'UNICO' ;

#-------------------------------------------------------------#
```

```
Operacao =
  Identificador
  Parametros
  Tipo_Resultado
  Def_Externas
  Pre
  Pos ;
Parametros =
  '(' Campos ')'
!  ;
Tipo_Resultado =
   ':' Expressao
!  ;
Def_Externas =
  'EXT' Externas
!  ;
Externas =
  Externas ';' Externa
! Externa;
Externa =
  Identificadores ':' Modo Expressao
!  ;
Modo =
  'WR'
! 'RD' ;
Pre =
  'PRE' Expressao ;
Pos =
  'POS' Expressao ;
```

```
#-------------------------------------------------------------#

Constante =
  'CT_Num'
! 'CT_Lit';
Identificador =
  'ID';
Identificadores =
  Identificadores ',' Identificador
! Identificador;
Expressoes =
  Expressoes ',' Expressao
! Expressao;
```

APÊNDICE B: Tabelas de Tokens

'$'	EOF	N
'SISTEMA'		S
':'	2PT	N
'FIM'		S
'ESTADO'		S
'TIPO'		S
'OPERACAO'		S
'INVARIANTE'		S
';'	PTVIRG	N
'('	ABREPAR	N
')'	FECHAPAR	N
'E'		S
'OU'		S
'=>'	IMPLICA2	N
'<=>'	EQUIVALE2	N
'<'	LT	N
'='	EQ	N
'>'	GT	N
'<='	LE	N
'>='	GE	N
'<>'	NE	N
'PERTENCE'		S
'CONTIDO'		S
'CONTEM'		S
'+'	MAIS	N
'-'	MENOS	N

'UNIAO'		S
'*'	VEZES	N
'/'	BARRA	N
'\'	BARRAINV	N
'INTERSECAO'		S
'->'	IMPLICA1	N
'<->'	EQUIVALE1	N
'NAO'		S
'DOMINIO'		S
'IMAGEM'		S
'INDICES'		S
'ELEMENTOS'		S
'-SET'	CONSTRSET	N
'-LIST'	CONSTRLIST	N
'REGISTRO'		S
'{'	ABRECHA	N
'}'	FECHACHA	N
'|'	VERTICAL	N
'..'	ATE	N
'['	ABRECOL	N
']'	FECHACOL	N
'<<'	ABRELISTA	N
'>>'	FECHALISTA	N
'SE'		S
'ENTAO'		S
'SENAO'		S
'SEJA'		S
'EM'		S
'MK-'	CONSTRMAKE	N

'VERDADE'		S
'FALSO'		S
'TODO'		S
'EXISTE'		S
'UNICO'		S
'EXT'		S
'WR'		S
'RD'		S
'PRE'		S
'POS'		S
'CT_Num'		N
'CT_Lit'		N
'ID'		N
','	VIRG	N

APÊNDICE C: O Sistema Exemplo

```
!-----------------------------------------------------------!
! Sistema de Acesso a Textos e Autores em uma Biblioteca !
!-----------------------------------------------------------!

 SISTEMA Biblioteca:

   TIPO T_Texto = CHAR-LIST
   TIPO T_Idioma={Ingles,Portugues,Frances,Espanhol,Alemao}
   TIPO T_Nivel = {Iniciante,Medio,Avancado}
   TIPO T_Especie = {Livro,Artigo}
   TIPO T_Armazenamento = 'volume fisico, meio magnetico'
   TIPO T_Pagina =
     REGISTRO
       PaginaInicial,
       PaginaFinal   : INT;
     FIM
   TIPO T_Data =
     REGISTRO
       Dia,
       Mes,
       Ano: INT;
     FIM
   TIPO T_Palavras = CHAR-LIST
   TIPO T_Emprestimo =
     REGISTRO
       Quem:   T_Texto;
```

```
    Quando: T_Data;
  FIM

TIPO IdeTexto = T_Texto
TIPO IdeAutor = T_Texto
TIPO IdePalavra = T_Texto

TIPO RegTexto =
  REGISTRO
    Titulo         : T_Texto;
    AutoresT       : IdeAutor-SET;
    Idioma         : T_Idioma;
    Nivel          : T_Nivel;
    Especie        : T_Especie;
    Armazenamento  : T_Armazenamento;
    Editora        : T_Texto;
    Pagina         : T_Pagina;
    DataEdicao     : T_Data;
    Comentario     : T_Texto;
    PalavrasT      : IdePalavra-SET;
    Emprestimo     : T_Emprestimo;
  FIM
TIPO RegAutor =
  REGISTRO
    Nome      : T_Texto;
    TotLivros: INT;
  FIM
TIPO RegPalavra =
  REGISTRO
```

```
    TotLivros: INT;
  FIM

TIPO TextoMap    = IdeTexto <-> RegTexto
TIPO AutorMap    = IdeAutor <-> RegAutor
TIPO PalavraMap = IdePalavra -> RegPalavra

|-----------------------------------------------------------|

ESTADO
  Textos   : TextoMap;
  Autores  : AutorMap;
  Palavras: PalavraMap;

|-----------------------------------------------------------|

OPERACAO CriaAutor (Autor : IdeAutor;
                    RAutor: RegAutor)
EXT
  Autores: WR AutorMap;

PRE
  (NAO (Autor PERTENCE DOMINIO (Autores)))

POS
  (Autores" = Autores + [Autor <-> RAutor])

|-----------------------------------------------------------|

OPERACAO RemoveAutor (Autor: IdeAutor)

EXT
```

```
    Autores: WR AutorMap;

  PRE
    ((Autor PERTENCE DOMINIO (Autores)) E
     (TotLivros (Autores (Autor)) = 0))

  POS
    (SEJA RAutor = (Autores (Autor)) EM
      (Autores" = Autores - [Autor <-> RAutor]))
```

```
|-------------------------------------------------------------|

  OPERACAO CriaPalavra (Palavra: IdePalavra)

  EXT
    Palavras: WR PalavraMap;

  PRE
    (NAO (Palavra PERTENCE DOMINIO (Palavras)))

  POS
    (Palavras" = Palavras + [Palavra -> MK-Palavras (1)])
```

```
|-------------------------------------------------------------|

  OPERACAO RemovePalavra (Palavra: IdePalavra)

  EXT
    Palavras: WR PalavraMap;

  PRE
```

```
    ((Palavra PERTENCE DOMINIO (Palavras)) E
     (TotLivros (Palavras (Palavra)) = 0))

  POS
    (SEJA RPalavra = (Palavras (Palavra)) EM
      (Palavras" = Palavras - [Palavra -> RPalavra]))

!-------------------------------------------------------------!

  OPERACAO AssociaAutorATexto (Autor: IdeAutor;
                               Texto: IdeTexto)
  EXT
    Autores: WR AutorMap;
    Textos : WR TextoMap;

  PRE
    ((Autor PERTENCE DOMINIO (Autores)) E
     (Texto PERTENCE DOMINIO (TEXTOS)))

  POS
    ((SEJA TL = (TotLivros (Autores (Autor))) EM
      (TL" = TL + 1)) E
     (SEJA AU = (AutoresT (Textos (Texto))) EM
      (AU" = AU + [Autor])))

!-------------------------------------------------------------!

  OPERACAO DesassociaAutorDeTexto (Autor: IdeAutor;
                                   Texto: IdeTexto)
  EXT
    Autores: WR AutorMap;
```

```
  Textos : WR TextoMap;

PRE
  ((Autor PERTENCE DOMINIO (Autores)) E
   (Texto PERTENCE DOMINIO (Textos)))

POS
  ((SEJA TL = (TotLivros (Autores (Autor))) EM
    (TL" = TL - 1)) E
  (RemoveAutor (Autor)) E
   (SEJA AU = (AutoresT (Textos (Texto))) EM
    (AU" = AU - {Autor})))
```

```
OPERACAO AssociaPalavraATexto (Palavra: IdePalavra;
                               Texto  : IdeTexto)
EXT
  Palavras: WR PalavraMap;
  Textos  : WR TextoMap;

PRE
  ((Palavra PERTENCE DOMINIO (Palavras)) E
   (Texto PERTENCE DOMINIO (Textos)))

POS
  ((SEJA TL = (TotLivros (Palavras (Palavra))) EM
    (TL" = TL + 1)) E
   (SEJA PC = (PalavrasT (Textos (Texto))) EM
    (PC" = PC + {Palavra})))
```

```
!-------------------------------------------------------------!
  OPERACAO DesassociaPalavraDeTexto (Palavra: IdePalavra;
                                     Texto  : IdeTexto)
  EXT
    Palavras: WR PalavraMap;
    Textos  : WR TextoMap;

  PRE
    ((Palavra PERTENCE DOMINIO (Palavras)) E
     (Texto PERTENCE DOMINIO (Textos)))

  POS
    ((SEJA TL = (TotLivros (Palavras (Palavra))) EM
      (TL" = TL - 1)) E
     (RemovePalavra (Palavra)) E
     (SEJA PC = (PalavrasT (Textos (Texto))) EM
      (PC" = PC - {Palavra})))

!-------------------------------------------------------------!
  OPERACAO CriaTexto (Texto : IdeTexto;
                      RTexto: RegTexto)
  EXT
    Textos: WR TextoMap;

  PRE
    (NAO (Texto PERTENCE DOMINIO (TEXTOS)))
  POS
```

```
  ((Textos" = Textos + [Texto <-> RTexto]) E
   (TODO a
    (SE (a PERTENCE AutoresT (RTexto))
     ENTAO (AssociaAutorATexto (a,Texto))
     SENAO (VERDADE))) E
   (TODO p
    (SE (p PERTENCE PalavrasT (RTexto))
     ENTAO (AssociaPalavraATexto (a,Texto))
     SENAO (VERDADE))))
```

```
!-------------------------------------------------------------!
```

```
OPERACAO RemoveTexto (Texto: !deTexto)
EXT
  Textos: WR TextoMap;

PRE
  (Texto PERTENCE DOMINIO (TEXTOS))
POS
  (SEJA RTexto = (Textos (Texto)) EM
   ((Textos" = Textos - [Texto <-> RTexto]) E
    (TODO a
     (SE (a PERTENCE AutoresT (RTexto))
      ENTAO (DesassociaAutorDeTexto (a,Texto))
      SENAO (VERDADE))) E
    (TODO p
     (SE (p PERTENCE PalavrasT (RTexto))
      ENTAO (DesassociaPalavraDeTexto (a,Texto))
      SENAO (VERDADE)))))
```

```
INVARIANTE
  (TODO t
   (SE (t PERTENCE DOMINIO Textos)
    ENTAO (SEJA RTexto = (Autores (t)) EM
     ((TODO a
       (SE (a PERTENCE AutoresT (RTexto))
        ENTAO (a PERTENCE DOMINIO (Autores))
        SENAO (VERDADE))) E
      (TODO p
       (SE (p PERTENCE PalavrasT (RTexto))
        ENTAO (p PERTENCE DOMINIO (Palavras))
        SENAO (VERDADE)))))
    SENAO (VERDADE)))

FIM
```

APÊNDICE D: Textos Complementares

.IntroducaoAoVDM

O VDM e' um metodo formal para definicao de sistemas. Ele foi desenvolvido no laboratorio da IBM em Vienna, e o seu nome e' a sigla de "Vienna Definition Method".

O modelo utilizado possui uma abordagem denotacional, onde existe a definicao de um estado e das operacoes e funcoes que modificam este estado.

.DefinicaoDeSistema

Um sistema descrito em VDM possui 3 partes:

1. O ESTADO, que identifica que informacoes sao manipuladas;
2. As OPERACOES, que mostra como o ESTADO e' manipulado;
3. O INVARIANTE, que identifica propriedades que o ESTADO possui e que nao sao alteradas pela ativacao das OPERACOES.

.Biblioteca

O sistema de biblioteca tem por objetivo permitir a recuperacao de informacoes sobre textos, autores e palavras-chave.

.DefinicaoDeEstado

Um estado representa quais as informacoes tratadas

pelo sistema.

.DefinicaoDeTipo

Um tipo representa um dominio de valores.

Tipos pre-definidos:

.Nat

Numeros naturais

.Nat0

Numeros naturais, exceto o zero

.Int

Numeros inteiros

.Real

Numeros reais

.Bool

Valores logicos: falso ou verdadeiro

.Dat

Datas validas

.Char

Sequencias de caracteres

.DefinicaoDeOperacao

As operacoes sao responsaveis pela manipulacao dos dados tratados pelo sistema, definidos pelo seu estado.

.DefinicaoDeInvariante

O invariante e' uma formula que representa restricoes de integridade do estado. Ela e' uma formula valida,

no contexto do sistema que esta' sendo definido.

.Pagina

Informacoes sobre o inicio e o final do texto. O seu tamanho pode ser calculado pela diferenca entre a Pagina Inicial e a Pagina Final, acrescido de uma unidade.

! Fim do Arquivo

APÊNDICE E: O Texto Exemplo

1. Introducao ao VDM

O VDM e' um metodo formal para definicao de sistemas. Ele foi desenvolvido no laboratorio da IBM em Vienna, e o seu nome e' a sigla de "Vienna Definition Method".

O modelo utilizado possui uma abordagem denotacional, onde existe a definicao de um estado e das operacoes e funcoes que modificam este estado.

2. Definicao de Sistema

Um sistema descrito em VDM possui 3 partes:

1. O ESTADO, que identifica que informacoes sao manipuladas;
2. As OPERACOES, que mostra como o ESTADO e' manipulado;
3. O INVARIANTE, que identifica propriedades que o ESTADO possui e que nao sao alteradas pela ativacao das OPERACOES.

3. Descricao do sistema Biblioteca:

O sistema de biblioteca tem por objetivo permitir a recuperacao de informacoes sobre textos, autores e palavras-chave.

4. Definicao de Estado

Um estado representa quais as informacoes tratadas pelo sistema.

- Textos :TextoMap
- Autores :AutorMap
- Palavras :PalavraMap

5. Definicao de Tipo

Um tipo representa um dominio de valores.

Tipos pre-definidos:

.Nat

Numeros naturais

.Nat0

Numeros naturais, exceto o zero

.Int

Numeros inteiros

.Real

Numeros reais

.Bool

Valores logicos: falso ou verdadeiro

.Dat

Datas validas

.Char

Sequencias de caracteres

Tipos definidos para este sistema:

.AutorMap e' composto , formado pelo mapeamento

IdeAutor <-> RegAutor

.IdeAutor e' T_Texto

.IdePalavra e' T_Texto

.IdeTexto e' T_Texto

.PalavraMap e' composto , formado pelo mapeamento

IdePalavra -> RegPalavra

.RegAutor e' composto

- Nome :T_Texto
- TotLivros :Int

.RegPalavra e' composto

- TotLivros :Int

.RegTexto e' composto

- Titulo :T_Texto
- AutoresT e' conjunto de IdeAutor
- Idioma :T_Idioma
- Nivel :T_Nivel
- Especie :T_Especie
- Armazenamento :T_Armazenamento
- Editora :T_Texto
- Pagina :T_Pagina

  Informacoes sobre o inicio e o final do texto. O seu tamanho pode ser calculado pela diferenca entre a Pagina Inicial e a Pagina Final, acrescido de uma unidade.
- DataEdicao :T_Data
- Comentario :T_Texto

- PalavrasT e' conjunto de IdePalavra
- Emprestimo :T_Emprestimo

.TextoMap e' composto , formado pelo mapeamento
IdeTexto <-> RegTexto

.T_Armazenamento e' volume fisico, meio magnetico

.T_Data e' composto
- Dia,
- Mes,
- Ano,
:Int

.T_Emprestimo e' composto
- Quem :T_Texto
- Quando :T_Data

.T_Especie e' composto , formado por: Livro Artigo

.T_Idioma e' composto , formado por: Ingles
Portugues Frances Espanhol Alemao

.T_Nivel e' composto , formado por: Iniciante Medio
Avancado

.T_Pagina e' composto
- PaginaInicial,
- PaginaFinal,
:Int

.T_Palavras e' composto de Char

.T_Texto e' composto de Char

6. Definicao de Operacao

As operacoes sao responsaveis pela manipulacao dos dados tratados pelo sistema, definidos pelo seu

estado.

Qualquer duvida quanto ao significado de algum tipo consulte o item "Definicao de Tipos".

AssociaAutorATexto

A operacao AssociaAutorATexto possui os seguintes parametros e seus respectivos tipos: Autor (tipo ideAutor) e Texto (tipo ideTexto).

O uso do estado e' feito atraves da gravacao do campo Autores (tipo AutorMap) e gravacao do campo Textos (tipo TextoMap).

A pre-condicao desta operacao tem o resultado definido por F1.

Em F1, o resultado e' verdadeiro se (F2 e F3).

Em F2, "Autor pertence ao dominio de Autores".

Em F3, "Texto pertence ao dominio de Textos".

A pos-condicao desta operacao tem o resultado definido por F4.

Em F4, o resultado e' verdadeiro se (F5 e F6).

Na formula F5, "o novo conteudo de TL e' igual a "TL mais 1"", onde TL e' igual a TotLivros de Autores de Autor.

Na formula F6, "o novo conteudo de AU e' igual a "AU mais conjunto definido pelo elemento Autor"", onde AU e' igual a AutoresT de Textos de Texto.

AssociaPalavraATexto

A operacao AssociaPalavraATexto possui os seguintes parametros e seus respectivos tipos: Palavra (tipo IdePalavra) e Texto (tipo IdeTexto).

O uso do estado e' feito atraves da gravacao do campo Palavras (tipo PalavraMap) e gravacao do campo Textos (tipo TextoMap).

A pre-condicao desta operacao tem o resultado definido por F1.

Em F1, o resultado e' verdadeiro se (F2 e F3).

Em F2, "Palavra pertence ao dominio de Palavras".

Em F3, "Texto pertence ao dominio de Textos".

A pos-condicao desta operacao tem o resultado definido por F4.

Em F4, o resultado e' verdadeiro se (F5 e F6).

Na formula F5, "o novo conteudo de TL e' igual a "TL mais 1"", onde TL e' igual a TotLivros de Palavras de Palavra.

Na formula F6, "o novo conteudo de PC e' igual a "PC mais conjunto definido pelo elemento Palavra"", onde PC e' igual a PalavrasT de Textos de Texto.

CriaAutor

A operacao CriaAutor possui os seguintes parametros e seus respectivos tipos: Autor (tipo IdeAutor) e RAutor (tipo RegAutor).

O uso do estado e' feito atraves da gravacao do campo Autores de tipo AutorMap.

A pre-condicao desta operacao tem o resultado definido por F1.

Em F1, nao "Autor pertence ao dominio de Autores".

A pos-condicao desta operacao tem o resultado

definido por F2.

Em F2, "o novo conteudo de Autores e' igual a " Autores mais mapeamento definido pela associacao " Autor <-> RAutor""".

CriaPalavra

A operacao CriaPalavra possui apenas o parametro Palavra de tipo IdePalavra.

O uso do estado e' feito atraves da gravacao do campo Palavras de tipo PalavraMap.

A pre-condicao desta operacao tem o resultado definido por F1.

Em F1, nao "Palavra pertence ao dominio de Palavras"

.

A pos-condicao desta operacao tem o resultado definido por F2.

Em F2, "o novo conteudo de Palavras e' igual a " Palavras mais mapeamento definido pela associacao " Palavra -> registro de Palavras, composto de 1""".

CriaTexto

A operacao CriaTexto possui os seguintes parametros e seus respectivos tipos: Texto (tipo IdeTexto) e RTexto (tipo RegTexto).

O uso do estado e' feito atraves da gravacao do campo Textos de tipo TextoMap.

A pre-condicao desta operacao tem o resultado definido por F1.

Em F1, nao "Texto pertence ao dominio de Textos".

A pos-condicao desta operacao tem o resultado definido por F2.

Em F2, o resultado e' verdadeiro se (F3 e F4) e F5).

Em F3, "o novo conteudo de Textos e' igual a "Textos mais mapeamento definido pela associacao "Texto <-> RTexto""".

A formula F4 e' verdadeira se para todo a, se "a pertence ao AutoresT de RTexto", entao o resultado e' o resultado de F6; caso contrario o resultado e' sempre verdadeiro.

Em F6 o resultado e' o de AssociaAutorATexto de a e

Texto.

A formula F5 e' verdadeira se para todo p, se "p pertence ao PalavrasT de RTexto", entao o resultado e' o resultado de F7; caso contrario o resultado e' sempre verdadeiro.

Em F7 o resultado e' o de AssociaPalavraATexto de a e Texto.

DesassociaAutorDeTexto

A operacao DesassociaAutorDeTexto possui os seguintes parametros e seus respectivos tipos: Autor (tipo IdeAutor) e Texto (tipo IdeTexto).

O uso do estado e' feito atraves da gravacao do campo Autores (tipo AutorMap) e gravacao do campo Textos (tipo TextoMap).

A pre-condicao desta operacao tem o resultado definido por F1.

Em F1, o resultado e' verdadeiro se (F2 e F3).

Em F2, "Autor pertence ao dominio de Autores".

Em F3, "Texto pertence ao dominio de Textos".

A pos-condicao desta operacao tem o resultado definido por F4.

Em F4, o resultado e' verdadeiro se (F5 e F6) e F7).

Na formula F5, "o novo conteudo de TL e' igual a "TL menos 1"", onde TL e' igual a TotLivros de Autores de Autor.

Em F6, RemoveAutor de Autor.

Na formula F7, "o novo conteudo de AU e' igual a "AU menos conjunto definido pelo elemento Autor"", onde AU e' igual a AutoresT de Textos de Texto.

DesassociaPalavraDeTexto

A operacao DesassociaPalavraDeTexto possui os seguintes parametros e seus respectivos tipos: Palavra (tipo IdePalavra) e Texto (tipo IdeTexto).

O uso do estado e' feito atraves da gravacao do campo Palavras (tipo PalavraMap) e gravacao do campo Textos (tipo TextoMap).

A pre-condicao desta operacao tem o resultado definido por F1.

Em F1, o resultado e' verdadeiro se (F2 e F3).

Em F2, "Palavra pertence ao dominio de Palavras".

Em F3, "Texto pertence ao dominio de Textos".

A pos-condicao desta operacao tem o resultado definido por F4.

Em F4, o resultado e' verdadeiro se (F5 e F6) e F7).

Na formula F5, "o novo conteudo de TL e' igual a "TL menos 1"", onde TL e' igual a TotLivros de Palavras de Palavra.

Em F6, RemovePalavra de Palavra.

Na formula F7, "o novo conteudo de PC e' igual a "PC menos conjunto definido pelo elemento Palavra"", onde PC e' igual a PalavrasT de Textos de Texto.

RemoveAutor

A operacao RemoveAutor possui apenas o parametro Autor de tipo IdeAutor.

O uso do estado e' feito atraves da gravacao do

campo Autores de tipo AutorMap.

A pre-condicao desta operacao tem o resultado definido por F1.

Em F1, o resultado e' verdadeiro se (F2 e F3).

Em F2, "Autor pertence ao dominio de Autores".

Em F3, "TotLivros de Autores de Autor e' igual a 0".

A pos-condicao desta operacao tem o resultado definido por F4.

Na formula F4, "o novo conteudo de Autores e' igual a "Autores menos mapeamento definido pela associacao "Autor <-> RAutor""", onde RAutor e' igual a Autores de Autor.

RemovePalavra

A operacao RemovePalavra possui apenas o parametro Palavra de tipo IdePalavra.

O uso do estado e' feito atraves da gravacao do campo Palavras de tipo PalavraMap.

A pre-condicao desta operacao tem o resultado

definido por F1.

Em F1, o resultado e' verdadeiro se (F2 e F3).

Em F2, "Palavra pertence ao dominio de Palavras".

Em F3, "TotLivros de Palavras de Palavra e' igual a 0".

A pos-condicao desta operacao tem o resultado definido por F4.

Na formula F4, "o novo conteudo de Palavras e' igual a "Palavras menos mapeamento definido pela associacao "Palavra -> RPalavra"", onde RPalavra e' igual a Palavras de Palavra.

RemoveTexto

A operacao RemoveTexto possui apenas o parametro Texto de tipo IdeTexto.

O uso do estado e' feito atraves da gravacao do campo Textos de tipo TextoMap.

A pre-condicao desta operacao tem o resultado definido por F1.

Em F1, "Texto pertence ao dominio de Textos".

A pos-condicao desta operacao tem o resultado definido por F2.

Na formula F2, o resultado e' verdadeiro se (F3 e F4) e F5).

Em F3, "o novo conteudo de Textos e' igual a "Textos menos mapeamento definido pela associacao "Texto <-> RTexto""".

A formula F4 e' verdadeira se para todo a, se "a pertence ao AutoresT de RTexto", entao o resultado e' o resultado de F6; caso contrario o resultado e' sempre verdadeiro.

Em F6 o resultado e' o de DesassociaAutorDeTexto de a e Texto.

A formula F5 e' verdadeira se para todo p, se "p pertence ao PalavrasT de RTexto", entao o resultado e' o resultado de F7; caso contrario o resultado e' sempre verdadeiro.

Em F7 o resultado e' o de DesassociaPalavraDeTexto de a e Texto.

Nas formulas acima considere que RTexto e' igual a

Textos de Texto.

7. Definicao de Invariante

O invariante e' uma formula que representa restricoes de integridade do estado. Ela e' uma formula valida, no contexto do sistema que esta' sendo definido.

O invariante desta operacao tem o resultado definido por F1.

A formula F1 e' verdadeira se para todo t, se "t pertence ao dominio de Textos", entao o resultado e' o resultado de F2; caso contrario o resultado e' sempre verdadeiro.

Na formula F2, o resultado e' verdadeiro se (F3 e F4).

A formula F3 e' verdadeira se para todo a, se "a pertence ao AutoresT de RTexto", entao o resultado e' o resultado de F5; caso contrario o resultado e' sempre verdadeiro.

F5 e' verdadeira se "a pertence ao dominio de Autores" (caso contrario o seu resultado e' falso).

A formula F4 e' verdadeira se para todo p, se "p pertence ao PalavrasT de RTexto", entao o resultado e'

o resultado de F6; caso contrario o resultado e'
sempre verdadeiro.

F6 e' verdadeira se "p pertence ao dominio de Palavras"
(caso contrario o seu resultado e' falso).

Nas formulas acima considere que RTexto e' igual a
Autores de t.

Referências Bibliográficas:

AHO, A. V. e ULLMAN J. D. (1977), "Principles of Compiler Design", Addison-Wesley.

BALZER, R. (1985), "A 15 Year Perspective on Automatic Programming", IEEE Transactions on Software Engineering, Vol. SE-11, Num. 11, Novembro.

BJORNER, D. (1986a), "Project Graphs and Meta-Programs: towards a theory of software development",Department of Computer Science,Technical University of Denmark.

BJORNER, D. (1986b), "Software Development Graphs: a unifying concept for software development?", Department of Computer Science, Technical University of Denmark.

BJORNER, D. e JONES, C. B. (1982), "Formal Specification and Software Developent",Prentice-Hall International (UK) Ltd.

BJORNER, D., JONES, C. B., MAC AN AIRCHINNIGH, M. e NEUHOLD, E.J. (1987a), "VDM '87 - A Formal Method at Work", Lecture Notes in Computer Science, No. 252, Springer-Verlag.

BJORNER,D. e RASMUSSEN, A. (1987b), "An Annotated VDM Bibliography", Department of Computer Science, Technical University of Denmark.

BOEHM, B.W. (1976), "Software Engineering", IEEE Trans. on Computer Science, Dezembro.

COHEN,B., HARWOOD, W.T. e JACKSON, M.I. (1986), "The Specification of Complex Systems", Addison-Wesley Publishing Company, Great Britain.

GANE, C. e SARSON, T. (1979), "Structured System Analysis: Tools and Techniques", Prentice-Hall, EngleWood Cliffs, NJ.

JONES, C. B. (1986), "Systematic Software Development Using VDM",(Prentice-Hall International series in computer science), Prentice-Hall International (UK) Ltd.

MENDES, S. B. T. e DE AGUIAR, T. C. (1988), "Métodos para Especificação de Sistemas", III Escola Brasileiro-Argentina de Informática, Curitiba.

RANGEL NETTO, J. L. M. (1988), "Manual de operação do sistema de geração de analisadores sintáticos R*S simples", Monografias em Ciência da Computação No. 7/88, Departamento de Informática, PUC/RJ.

ROSS, D.T. (1977), "Structured Analysis (SA): A Language for Communicating Ideas", IEEE Transactions on Software Engineering, Vol. SE-3, No. 1, Janeiro.

SWARTOUT, W. (1982), "Gist English Generator", Proceedings of the National Conference on Artificial Intelligence (AAAI-82).

TEICHROEW, D. e HERSLEY, E. A. (1977), "PSL/PSA: A Computer Aided Technique For Structured Documentation and Analysis of Information Processing Systems", IEEE Transactions of Software Engineering, Vol. SE-3, Janeiro.