Especificação de um Programa de Matrículas Utilizando VDM

Leila Ribeiro, Daltro José Nunes

Pós-Graduação em Ciências de Computação
Universidade Federal do Rio Grande do Sul
CP 1501, 90001 Porto Alegre - RS - Brasil

Resumo

Este trabalho apresenta uma aplicação do método VDM (The Vienna Development Method): a especificação formal de uma parte do programa de matrículas da Universidade Federal do Rio Grande do Sul.

Abstract

This paper shows an application of VDM (The Vienna Development Method): the formal specification of a subset of the registration system of UFRGS.

1 Introdução

Este trabalho mostra uma aplicação do método VDM (Vienna Development Method) [BJO 78][JON 86][VDM 87] : a especificação formal de uma parte do programa de matrícula da Universidade Federal do Rio Grande do Sul.

A especificação está organizada em 3 partes:

1) Definição dos **Domínios Semânticos**;

2) Definição dos **Domínios Sintáticos** e

3) Construção da **Função de Elaboração**.

As funções auxiliares utilizadas na especificação não são mostradas aqui, mas podem ser encontradas em [RIB 88].

É importante salientar que a especificação formal aqui descrita foi feita com base no sistema de matrículas da UFRGS, e não na implementação existente na universidade.

2 Sistema de Matrículas da UFRGS

A figura 2.1 mostra como está organizado o sistema de matrículas da UFRGS.

Os cadastros existentes são:

A. Cursos: informações sobre todos os cursos da universidade;

B. Alunos: relação de todos os alunos da universidade com seus dados cadastrais;

C. Horários: informações sobre todos os horários da disciplinas que vão ser ministradas em um semestre;

D. Disciplinas: informações sobre as disciplinas existentes na universidade;

Existem também uma lista com os **requerimentos de matrícula** (E) e um conjunto de **alunos matriculados** (F) nas turmas das disciplinas.

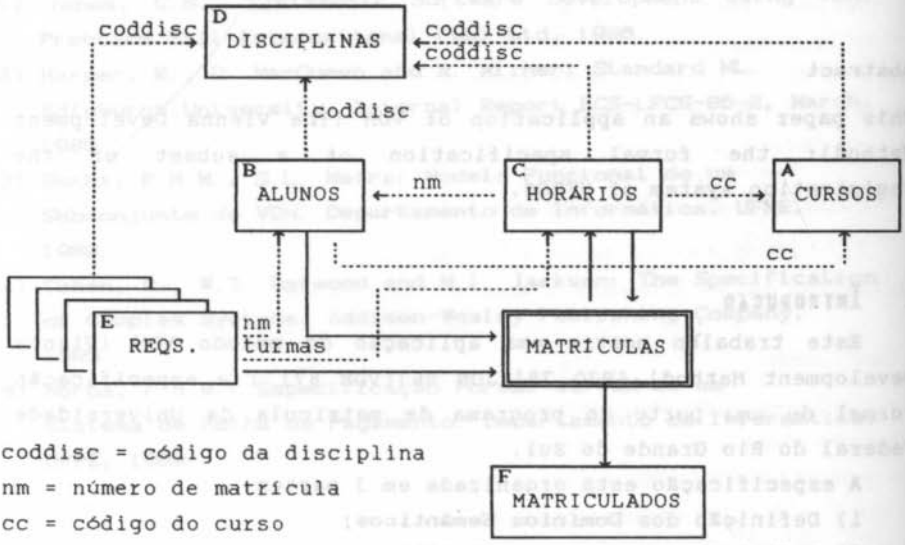

FIGURA 2.1 - Sistema de Matrículas da UFRGS

Obs.: Na figura 2.1, sobre o conjunto de alunos matriculados na turmas (F) devem ser feitas as mesmas consistências existentes sobre a lista de requerimentos (E).

O sistema funciona do seguinte modo:

(1) Busca-se o primeiro requerimento da lista de requerimentos;

(2) matricula-se o aluno em todas as disciplinas solicitadas, se houver vagas nas turmas[1];

(3) coloca-se o aluno no conjunto de matriculados;

(4) Se ainda houverem requerimentos na lista de requerimentos volta-se ao passo 1.

Consistências como:

- o aluno deve ter obtido aprovação nas disciplinas que são pré-requisito das quais ele deseja se matricular;

- não deve haver colisão de horários entre as turmas solicitadas no requerimento de matrícula;

- o aluno deve pertencer ao curso para o qual são oferecidas as vagas na turma, etc.

- são feitas antes do requerimento entrar na lista de requerimentos do sistema (ou seja, a lista de requerimentos do sistema só possui requerimentos válidos).

Deste modo, a execução do sistema de matrículas terá os seguintes resultados:

- para cada turma, existirá um conjunto de alunos nela matriculados (este conjunto pode ser vazio);

- será criada uma lista de alunos matriculados que conterá os números de matrícula dos alunos e o conjunto de turmas nas quais eles se matricularam. Esta lista será usada para a confecção dos comprovantes de matrícula e também para a atualização dos históricos escolares dos alunos.

---

[1] Esta é a única consistência que só pode ser feita na hora da matrícula.

3 Especificação do Sistema de Matrículas da UFRGS

3.1 Domínios Semânticos

A) Cursos da UFRGS

```
1.0  CURSOS    =  CODCURSO ──m──→ (DIS-CUR × INFOCURSO)
1.1  DIS-CUR   =  CODDISC ──m──→ STATUS
1.2  STATUS    =  { ob, op }
1.3  CODCURSO  =  Nat0
```

Exemplo de um elemento da classe CURSOS:

```
El-CURSOS = [ 1200  ──→  ( [( CPD, 135 ) ──→ ob,
                           ( CPD, 149 ) ──→ op,
                           ( MAT, 103 ) ──→ ob,
                           ( ECO, 134 ) ──→ op ], infocurso1),
            1060  ──→  ( [( FIS, 181 ) ──→ ob,
                           ( MAT, 103 ) ──→ ob,
                           ( CPD, 101 ) ──→ op ],infocurso2)]
```

Invariantes:

```
1.5  type : inv-CURSOS : CURSOS ──→ BOOL
1.6  type : inv-CODCURSO : CODCURSO ──→ BOOL
```

```
1.7  inv-CURSOS (c)  ≜
1.8      ( ∀ cod  ∈  dom (c) )
1.9          inv-CODCURSO (cod)    ∧
1.10         let  (discs,info) = cursos(cod)  in
1.11             discs  ≠  []    ∧
1.12             ( ∀ cd  ∈  dom (discs) )
1.13                 inv-CODDISC (cd)    ∧
1.14                 is-STATUS (discs(cd)) ∧
1.15             is-INFOCURSO(info)
```

```
7  inv-CURSOS
8  Todos os códigos de cursos
9  estão corretos e
10 o conjunto de disciplinas pertencentes a esse curso
11 nao é vazio ( o curso possui disciplinas ) e
12 todas essas disciplinas
13 tem código válido e
14 status válido  e
15 as informações sobre o curso sao válidas.
```

```
1.16 inv-CODCURSO (cod)  Δ
1.17     0010 ≤ cod ≤ 9999
```

```
16 inv-CODCURSO
17 O código dos cursos está entre 0010 e 9999.
```

B) Alunos da UFRGS

```
2.0  ALUNOS      =  NMATRIC ─m→ DADOS-AL
2.1  DADOS-AL   ::  (NOME x CODCURSO x HISTORICO)
2.2  HISTORICO =  DADOSHIST-set
2.2  DADOSHIST ::  (CODDISC x CONCEITO x PERIODO)
2.3  CONCEITO   =  { a, b, c, d, e,...}
2.4  PERIODO   :: (SEMESTRE x ANO)
```

```
Exemplo de um elemento da classe Alunos:
El-ALUNOS = [ 1276850 → ( 'Leila', 1200,
                          {( (CPD,130),a,(86,2) ),
                             ( (MAT,166),b,(85,1) ),
                             ( (HUM,468),a,(88,1) ) }),
              2196840 → ( 'Paulo', 1030,
                          {( (FIS,181),a,(84,1) ),
                             ( (MAT,103),c,(84,2) ),
                             ( (CPD,101),a,(87,2) ) })]
```

Invariantes:

```
2.5  type : inv-NMATRIC : Nat0 → BOOL
2.6  type : inv-PERIODO : PERIODO → BOOL
2.7  type : inv-HISTORICO : HISTORICO → BOOL
2.8  type : inv-ALUNOS : ALUNOS → BOOL
```

```
2.9  inv-NMATRIC (nm)  Δ
2.10     0001000 ≤ nm ≤ 9999999
```

```
9  inv-NMATRIC
10 O número de matrícula deve estar entre 1000 e 9999999
```

```
2.11 inv-PERIODO ( mk-PERIODO ( sem, ano ) ) Δ
2.12     sem ∈ { 1, 2 } ∧
2.13     00 ≤ ano ≤ 99
```

```
11 inv-PERIODO
12 O semestre só pode ser 1 ou 2, ou seja, o primeiro
    ou o segundo  e
13 o ano deve estar entre 00 e 99
```

```
2.14 inv-HISTORICO (h)  ≜
2.15 ( ∀ ( mk-DADOHIST( cd, conc, per ) )  ∈  h )
2.16       inv-CODDISC (cd)  ∧
2.17       is-CONCEITO (conc)  ∧
2.18       inv-PERIODO (per)  ∧
2.19 let   mk-DADOHIST ( cd1, conc1, per1 ),
2.20       mk-DADOHIST ( cd2, conc2, per2 )  ∈  h  in
2.21       if  cd1 = cd2
2.22       then  per1 ≠ per2  ∧
2.23          if  aprovacao ( conc1 )
2.24          then  per1 = mais-recente (per1, per2)  ∧
2.25                ~ aprovacao ( conc2 )
2.26          else  if  aprovacao ( conc2 )
2.27                then  per2 = mais-recente (per1, per2)
```

```
14 inv-HISTORICO
15 Todos os registros pertencentes ao histórico (h)
16 tem códigos de disciplina válidos e
17 conceitos válidos e
18 períodos válidos e
19,20  tomando-se duas árvores pertencentes ao histórico
21 se eles forem referentes a mesma disciplina
22 então necessariamente possuem períodos diferentes(uma
   disc. nao pode ser cursada 2 vezes no mesmo semestre)
23 e se um dos conceitos é de aprovação,
24, 25, 26 então o período no qual esta disciplina foi
   cursada é mais recente que o da outra
```

```
2.28 inv-ALUNOS (al)  ≜
2.29     ( ∀ nm  ∈  dom (al) )
2.30          inv-NMATRIC (nm)  ∧
2.31          let  mk-DADOSAL ( nome, cc, h ) = al (nm) in
2.32               nome ≠ ' '  ∧
2.33               inv-CODCURSO (cc)  ∧
2.34               inv-HISTORICO (h)
```

```
28 inv-ALUNOS
29 Todos os números de matrícula dos alunos
30 são válidos e
31 os registros associados a cada aluno
```

32 possuem um nome válido (diferente de brancos) e
33 código de curso válido e
34 histórico válido.

Funções auxiliares :

2.35 type : mais-recente : PERIODO x PERIODO → PERIODO

2.36 type : aprovacao : CONCEITO → BOOL

C) Horários de um Semestre

```
3.0  HORARIOS  =  CODDISC →(m) TURMA-set
3.1  TURMA     ::  (NOMETURMA x HORARIO x VAGAS)
3.2  NOMETURMA =  { a, b, c, ce, ...}
3.3  HORARIO   =  ( DIA x HORA x DURACAO )-set
3.4  VAGAS     =  CODCURSO →(m) ( Nat0 x LISTAL )
3.5  LISTAL    =  NMATRIC-set
3.6  DIA       = { d ∈ Nat0 | 1 < d < 6 }
3.7  HORA      = { h ∈ Nat0 | 7 < h < 21 }
3.8  DURACAO   = { d ∈ Nat0 | 1 < d < 10 }
```

Exemplo de um elemento da classe HORARIOS:

```
El-HORARIOS = [ (CPD,137) → { ( u,
                                { (3,8,2),(5,8,2) },
                                [1200 → (30,{})]) },
              (MAT,166) → { ( a,
                                { (2,8,2),(4,16,2),(6,15,2)},
                                [1010 → (15,{}),
                                 1200 → (20,{})] ) } ]
```

Os pares (30,{}), (15,{}) e (20,{}) indicam o número de vagas disponíveis na turma para cada curso e o conjunto de alunos deste curso matriculados nesta turma. Neste exemplo foram utilizados apenas conjuntos vazios, ou seja, este é um exemplo do estado inicial desta classe para o sistema de matrículas.

Invariantes:

```
3.9  type : inv-VAGAS : VAGAS → BOOL
3.10 type : inv-HORARIO : HORARIO → BOOL
3.11 type : inv-HORARIOS : HORARIOS → BOOL
```

```
3.12 inv-VAGAS ( v ) ≜
3.13    ( ∀ cc ∈ dom (v) )
3.14        inv-CODCURSO (cc)  ∧
3.15        let  ( nv, la ) = vagas (cc) in
3.16              nv  ∈  Nat0  ∧
3.17              ( ( ∀ nm  ∈  la  )
3.18                    inv-NMATRIC (nm) )  ∧
3.19              if  nv = 0 then  la ≠ {}
```

12 inv-VAGAS
13 Todos os códigos de cursos
14 são válidos e
15 as árvores associadas a esses códigos
16 possuem um número de vagas pertencente aos naturais e
17,18 um conjunto de números de matrícula válidos, e
19 se o número de vagas for igual a 0 então o conj. de alunos matriculados nessa turma é diferente de vazio (o número de vagas oferecidas em uma turma deve ser maior que 0).

```
3.20 inv-HORARIO (h)  ≜
3.21     ( ∀ ( dia, hora, dur )  ∈  h
3.22         is-DIA (dia)  ∧
3.23         is-HORA (hora)  ∧
3.24         is-DURACAO (dur)  ∧
3.25         if  dia = 6  then  hora ≤  11  ∧
3.26         if  hora = 20 then  dur ≤ 3  ∧
3.27         if  hora = 21 then  dur ≤ 2  ∧
3.28     ( let (d1, h1, dur1), (d2, h2, dur2) ∈ h  in
3.29           if  d1 = d2  then  ( ( h1 + dur1) ≤ h2 ) ∨
3.30                              ( ( h2 + dur2) ≤ h1 ) )
```

20 inv-HORARIO
21 Todos os registros pertencentes a um horario (h)
22 possuem dia,
23 hora e
24 duração válidos e
25 se o dia for Sábado entao a aula deve ser de manhã e
26 se a hora for 20 então a duração deve ser no máx. 3 e
27 se a hora for 21 então a duração deve ser no máx. 2 e
28 dois registros pertencentes a h
29,30 nao podem ter colisão de horário.

```
3.31 inv-HORARIOS (h) ≜
3.32 (∀ cd ∈ dom (h))
3.33     inv-CODDISC (cd)  ∧
3.34     ((∀ mk-TURMA (nome,horario,vagas) ∈ horarios(cd))
3.35         is-NOMETURMA (nome)  ∧
3.36         inv-HORARIO (horario)  ∧
3.37         inv-VAGAS (vagas) ) ∧
3.38     ( let mk-TURMA  (nome1,,),
3.39           mk-TURMA  (nome2,,)  ∈  horarios (cd)  in
3.40           nome1  ≠  nome2 )
```

```
31 inv-HORARIOS
32 Todos os códigos de disciplinas
33 são válidos e
34 todos os registros associados a esses códigos
35 tem nome de turma,
36 horário e
37 vagas válidos e
38,39 duas turmas de uma mesma disciplina
40 não podem ter nomes iguais.
```

D) Disciplinas da UFRGS

```
4.0 DISCIPLINAS =  DISCIPLINA-set
4.1 DISCIPLINA  ::(CODDISC x NOMEDISC x CR x CH x PRE-REQ)
4.2 CODDISC     :: (DEPTO x NUM)
4.3 DEPTO       =  { cpd, arq, eng, med, hum,...}
4.4 CR, CH, NUM = Nat0
4.5 PRE-REQ     = CODDISC-set
```

```
Exemplo de um elemento da classe DISCIPLINAS:
El-DISCIPLINAS = { ( (cpd,137),Sist.Distribuidos,4,4,
                     {(cpd,131),(cpd,129)} ),
                   ( (cpd,101),Computacao Bas.,4,4,{}),
                   ( (mat,103),Calculo II,6,6,
                     {(mat,166),(mat,155)} ) }
```

Invariantes :

```
4.6  type : inv-CODDISC  :  CODDISC → BOOL
4.7  type : inv-PREREQ   :  CODDISC x PRE-REQ x
4.8                         DISCIPLINAS→  BOOL
```

```
4.9  type : inv-DISCIPLINA  : DISCIPLINA ⟶ BOOL
4.10 type : inv-DISCIPLINAS : DISCIPLINAS ⟶ BOOL
```

```
4.11 inv-CODDISC ( mk-CODDISC ( depto, num ) ≜
4.12     is-DEPTO (depto)  ∧
4.13     001 ≤ num ≤ 999
```

11 inv-CODDISC
12 o departamento é válido e
13 o número da disciplina está entre 1 e 999.

```
4.14 inv-PREREQ (cd, discs, conjdiscs) ≜
4.15     cd  ∉  ( discs ∪
4.16                 prereqind (discs, conjdiscs) ({}))
```

14 inv-PREREQ
15 a disciplina não pertence ao conjunto formado pelos seus pré-requisitos diretos e
16 os seus pré-requisitos indiretos (uma disciplina não pode ser pré-requisito dela mesma).

```
4.17 inv-DISCIPLINA (mk-DISCIPLINA (cd,nome,cr,ch,pr)) ≜
4.18     inv-CODDISC (cd)  ∧
4.19     nome ≠ ' '  ∧
4.20     ch ≥ 1
```

17 inv-DISCIPLINA
18 O código da disciplina é válido e
19 ela possui um nome (seu nome é diferente e brancos) e
20 sua carga horária é maior ou igual a um (a carga horária é semanal e não semestral).

```
4.21 inv-DISCIPLINAS ( discs ) ≜
4.22     ( ∀ mk-DISCIPLINA ( cd1, n1, _, _, _),
4.23         mk-DISCIPLINA ( cd2, n2, _, _, _)  ∈  discs)
4.24         cd1  ≠  cd2  ∧
4.25         n1   ≠  n2  ∧
4.26     ( ∀ mk-DISCIPLINA ( cd, _, _, _, pr ) ∈ discs
4.27         inv-PRE-REQ(cd, pr, discs)
```

21 inv-DISCIPLINAS
22,23 Duas disciplinas pertencentes ao conjunto de disc.
24 têm códigos diferentes e
25 nomes diferentes e
26 todas as disciplinas pertencentes ao conjunto
27 não são pré-requisito delas mesmas.

Funções auxiliares :

4.29 type : prereqind : PRE-REQ x DISCIPLINAS → CODDISC-set → CODDISC-set

4.30 type : pr : CODDISC x DISCIPLINAS → CODDISC-set

E) Requerimentos de Matrícula

```
5.0  REQLIST = REQUERIMENTO*
5.1  REQUERIMENTO :: ( NMATRIC x DIS-TUR )
5.2  DIS-TUR      = CODDISC → NOMETURMA
```

Exemplo de um elemento pertencente a classe REQLIST :

```
El-REQLIST = < ( 1276850  [ (cpd,137) → u,
                            (cpd,145) → a,
                            (mat,166) → c ] ),
               ( 1654879  [ (mat,166) → b ] ) >
```

Invariantes :

```
5.3  type : inv-REQLIST : REQLIST → BOOL
5.4  type : inv-REQUERIMENTO : REQUERIMENTO → BOOL
```

```
5.5  inv-REQLIST (reql) ≜
5.6  let conjnm = {nm | mk-REQUERIMENTO (nm,_) ∈ reql} in
5.7       card (conjnm) = len (reql)
```

5 inv-REQLIST
6,7 um aluno nao pode ter 2 requerimentos de matrícula

```
5.8  inv-REQUERIMENTO (nmatric, distur) ≜
5.9     inv-NMATRIC(nmatric) ∧
5.10     distur ≠ [] ∧
5.11     ( ∀ cd ∈ dom (distur) )
5.12          inv-CODDISC (cd) ∧
5.13     ( ∀ nometurma ∈ rng (distur) )
5.14          is-NOMETURMA ( nometurma )
```

8 inv-REQUERIMENTO
9 O número de matrícula do requerimento é válido e
10 o aluno quer se matricular em alguma disciplina e
11 todos os códigos de disciplinas do requerimento
12 são válidos e
13 todas as horas do turno do requerimento
14 são válidos.

F) Sistema de Matrículas

```
6.0  SMATR ::  CURSOS x DISCIPLINAS x ALUNOS x
6.1            HORÁRIOS x REQLIST x MATRICULADOS
6.2  MATRICULADOS = REQUERIMENTO-set
```

Um elem. da classe SMATR é uma árvore que tem como folhas:
- o conjunto de cursos da universidade;
- o conjunto de disciplinas da universidade;
- o conjunto de alunos da universidade;
- o conjunto de horários das turmas para um determinado período letivo;
- a lista de requerimentos de matrícula dos alunos;
- a lista dos alunos matriculados em um periodo na universidade.

Invariantes:

```
6.3  type : inv-MATRICULADOS : MATRICULADOS → BOOL
6.4  type : inv-SMATR : SMATR → BOOL
```

```
6.5 inv-MATRICULADOS (m) ≜
6.6    (∀ req ∈ m)
6.7       inv-REQUERIMENTO(req)
```

5 inv-MATRICULADOS
6,7 Todos os requerimentos de matrícula são válidos.

```
6.8  pre : inv-CURSOS (cursos)  ∧
6.9        inv-DISCIPLINAS (discs)  ∧
6.10       inv-ALUNOS (alunos)  ∧
6.11       inv-HORARIOS (hors)  ∧
6.12       (∀ req ∈ reqs)
6.13          inv-REQUERIMENTO (reqs)

6.14 inv-SMATR(mk-SMATR(cursos,discs,alunos,hors,reqs,m)≜
6.15 (∀ (discur,_)  ∈  rng (cursos) )
6.16     (∀ cd  ∈  dom (discur) )
6.17        (∃ mk-DISCIPLINA (c,_,_,_,_) ∈ discs|c=cd) ∧
6.18 (∀  mk-DADOSAL (_, cc, hist)  ∈  rng (alunos) )
6.19     cc  ∈  dom (cursos)  ∧
```

```
6.20      (∀  mk-DADOHIST (cd, _, _)  ∈  hist )
6.21          (∃ mk-DISCIPLINA (c,_,_,_,_) ∈ discs|c=cd) ∧
6.22 (∀ cd  ∈ dom (hors) )
6.23      (∃ mk-DISCIPLINA (c,_,_,ch,_)  ∈  discs|c=cd) ∧
6.24      (∀  mk-TURMA (_,horario,vagas) ∈ hors(cd) )
6.25       •  dom (vagas)  ⊆  dom (cursos)  ∧
6.26          (∀  ( _, listal)  ∈  rng (vagas) )
6.27             listal  ⊆  dom (alunos)  ∧
6.28          (∀ cc ∈ dom (vagas) )
6.29            let  (_, listal) = vagas(cc) in
6.30                    (∀  nm ∈ listal )
6.31                          codcurso(nm, alunos) = cc    ∧
6.32          somadur(horario) = ch  ∧
6.33 (∀  mk-REQUERIMENTO (nm, distur)  ∈  elems (reqs) )
6.34      nm ∈ dom (alunos)  ∧
6.35      (∀  cd  ∈ dom (distur) )
6.36          cd ∈ {c|mk-DISCIPLINA (c,_,_,_,_) ∈ discs} ∧
6.37          let  t = { nome | mk-TURMA (nome,_,_) ∈
6.38                        hors(cd) }  in
6.39                 distur(cd) ∈ t  ∧
6.40      let  cd1, cd2  ∈  dom (distur)  in
6.41           let mk-TURMA( distur(cd1),
6.42               mk-HORARIO(d1,h1,dur1),_) ∈ hors(cd1) in
6.43           let mk-TURMA( distur(cd2),
6.44               mk-HORARIO(d2,h2,dur2),_) ∈ hors(cd2) in
6.45               if  d1=d2  then  ( (h1+dur1) ≤ h2 ) ∨
6.46                                ( (h2+dur2) ≤ h1 ) ∧
6.47      (∀  cd ∈ dom (distur) )
6.48          let mk-DADOS-AL (_,cc,hist) = alunos(nm) in
6.49          let mk-TURMA (distur(cd),_,v) ∈ hors (cd) in
6.50              cc ∈ dom v       ∧
6.51              let mk-DISCIPLINA(cd,_,_,_,pr) ∈ discs in
6.52                  pr ∈ {cdh | mk-DADOSHIST (cdh,con,_)
6.53                           ∈ hist ∧ aprovacao(con) }
```

```
8  Pré-condições : Os cursos,
9                  disciplinas,
10                 alunos,
11                 horários  devem ser válidos e
12                 todos os requerimentos de matrícula
13                 devem ser válidos.
14 inv-SMATR
15,16,17 Todas as disciplinas que são contidas pelos
   cursos estão no conjunto de disc. da universidade
18 e todos os alunos
19 pertencem a cursos que existem na universidade e
20,21 tem no seu histórico apenas disciplinas que existem
22,23 e todas as disciplinas para as quais existem turmas
   pertencem ao conjunto de disciplinas da universidade e
24,25 todos os cursos para os quais existem vagas nas
   turmas pertencem ao conjunto de cursos e
26,27 as listas de alunos de cada turma possuem alunos
   pertencentes a universidade e
28,29,30,31 o curso dos alunos matriculados em uma turma
   é mesmo para o qual sao oferecidas as vagas e
32 a soma das durações das aulas de cada disciplina é
   igual a carga horária da disciplina e
33,34 todos os requerimentos de matrícula sao feitos por
   alunos pertencentes a universidade e
35.36 as disciplinas solicitadas nos requerimentos existem
37,38,39  e  as  turmas  solicitadas  existem  para  as
   disciplinas solicitadas  e
40,41,42,43,44,45,46 nao existe colisão de horários entre
   as turmas solicitadas em um requerimento de matrícula e
47,48,49,50 foram oferecidas vagas na turma solicitada
   para o curso do aluno e
51,52,53  o  aluno  já  cursou  os  pré-requisitos  da
   disciplina solicitada.
```

3.2 Domínios Sintáticos

```
0  OPER :: INCLUSÃO | ATUALIZAÇÃO | CONSULTA | MATRICULA
1  MATRICULA :: SMATR
```

```
3  type is-wf-OPER : OPER  →  BOOL
4  is-wf-OPER (op) Δ
5     cases op:
6        ( mk-MATRICULA (s) → is-SMATR(s),
7          mk-INSERÇÃO (...) → ...
8          .... )
```

6 O sistema de matrícula inicial da operação matrícula é um sistema consistente.

3.3 Função de Elaboração

A seguir a operação matrícula está especificada implicitamente.

```
0  type  elab-MATRICULA : MATRICULA → SMATR → SMATR
1  pre-MATRICULA (mk-SMATR (_,_ ,horários,_,matrics)) Δ
2       matrics = {} ∧
3      (∀ coddisc ∈ dom (horários))
4         (∀ mk-TURMA (_,_,vagas) ∈ horários(coddisc))
5            (∀ codcurso ∈ dom (vagas))
6               let (numvag, listal) = vagas(codcurso) in
7                  numvag ≠ 0  ∧
8                  listal = {}
9  pos-MATRICULA (mk-SMATR (c,d,a,h,r,l),
                  mk-SMATR (c,d,a,h',r',l')) Δ
10     r' = <> ∧
11    (∀ cd ∈ dom (h'))
12       (∀ mk-TURMA (t,_,vagas') ∈ h'(cd))
13          (∀ codc ∈ dom (vagas'))
14             let (nvag',listal') = vagas'(codc) in
15             nvag'+card(listal')=totalvag(h,cd,t,codc) ∧
16             (∀ nmatric ∈ listal')
17                (∃! mk-MATRICULADOS(nmatric,distur) ∈ l')
18                   distur (cd) = t  ∧
19             (~ (∃ mk-MATRICULADOS (nmatric,distur) ∈ l')
                  (nmatric ∉ listal') ∧
                  (distur (cd) = t) ) )
```

1 Pré-condições para a matrícula:
2 Não existem alunos no conjunto de alunos matriculados e
3-7 o conjunto de vagas oferecidas para cada turma é diferente de zero e
8 não existe nenhum aluno nas listas de alunos das turmas.

9 Pós-condição da matrícula:
10 A lista de requerimentos de matrícula é vazia e
11,12,13,14,15 o número de alunos matriculados em uma turma mais o número de vagas disponíveis é igual ao número de vagas que foram oferecidas na turma e
16,17,18 todos os alunos que estão em uma turma tem um comprovante de matrícula e a matrícula nesta turma está registrada no comprovante e
19 não existem comprovantes de matrícula nesta turma nos quais os alunos não estejam na lista de alunos da turma.

Função auxiliar:

20 type totalvag : HORARIOS x CODDISC x NOMETURMA x CODCURSO → Nat0

4 Conclusão

A partir dessa especificação, podem ser feitos refinamentos sucessivos (incluindo em cada etapa mais detalhes de implementação) até se chegar em um programa em alguma linguagem de programação. Cada nova especificação que é feita deve ser equivalente à anterior (deve-se provar que elas são equivalentes). A utilização de VDM no desenvolvimento de software está explicada em detalhes em [RIB 88].

Deste modo, o uso de VDM para especificar sistemas tem 2 grandes vantagens sobre o uso de métodos informais:

- a construção de especificações não-ambíguas (entendidas da mesma maneira por todos que as lêem). Como uma consequência direta dessa precisão são descobertos erros que normalmente seriam descobertos apenas em fases mais avançadas do ciclo de vida do software;

- a especificação formal serve como base para o desenvolvimento de programas corretos (entendendo-se como "correto" aqui um programa que atende sua especificação).

5 Referências Bibliográficas

[BJO 78] BJORNER, D. & JONES, C. (eds.) The Vienna development method: The meta-language. Lecture Notes in Computer Science, 61, 1978.

[JON 86] JONES, C. Systematic software development using VDM. Inglaterra, Prentice-Hall International,1986. 300p.

[RIB 88] RIBEIRO, L. O método VDM e sua aplicação na especificação formal de um programa de matrícula. Porto Alegre, 1988.
[Trabalho de Diplomação - UFRGS]

[VDM 87] VDM'87: VDM - A formal method at work.Lecture Notes in Computer Sciences, 252, 1987.