ISSN 0103-2509

Especificações formais em VDM: um estudo de caso

MEIRA, C.A.A.; MASIERO, P.C.

nº 6

São Carlos (SP)
1990

ESPECIFICAÇÕES FORMAIS EM VDM: UM ESTUDO DE CASO

Carlos Alberto Alves Meira

Paulo Cesar Masiero

ICMSC-USP

Sumário

Este trabalho apresenta uma breve introdução ao método de especificação formal de sistemas denominado VDM (Vienna Development Method) e o aplica integralmente na especificação de um sistema exemplo, ilustrando, dessa forma, a notação VDM.

KEYWORDS:
Especificação
VDM

1 Introdução

Dentre as fases que compõem o ciclo de desenvolvimento de software, a fase de especificação dos requisitos do sistema desempenha um papel chave. Todas as demais fases dependem de uma boa especificação para que o resultado final seja satisfatório. Qualquer falha ou má interpretação na definição dos requisitos do sistema acarretará na construção de um produto que não atende às necessidades do usuário. Sendo assim, a especificação deve ser precisa e completa.

Alguns sistemas de software requerem maior precisão na especificação do que outros. Em sistemas de tempo real, os danos causados por erros de especificação são potencialmente muito mais prejudiciais do que os causados, por exemplo, em sistemas comerciais.

Especificação formal, através de formalismos matemáticos, é um meio de se conseguir maior rigor na especificação de sistemas. A especificação é formal quando a linguagem utilizada para definir os requisitos do sistema possui sintaxe (regras de construção da linguagem) e semântica (significado) bem definidas, usualmente envolvendo notação matemática. A especificação formal é uma abstração do objeto real sendo especificado onde apenas os detalhes relevantes são considerados.

Os métodos de especificação formal podem ser classificados em duas grandes classes: métodos baseados em propriedades e métodos baseados em modelos [Mar89]. Nos métodos baseados em propriedades, como o próprio nome diz, descrevem-se as propriedades que o sistema deve satisfazer. Como exemplo, existem os métodos de especificação algébrica [Gut80]. Nos métodos baseados em modelos procura-se construir, a partir de objetos já conhecidos, um modelo que reflita as propriedades do sistema em discussão. Um exemplo de método baseado em modelos é o método VDM (Vienna Development Method) [Bjo89].

Este trabalho tem o objetivo de ilustrar a aplicação do método VDM, para isso utilizando como caso de estudo o sistema EVENT LOG, cujo enunciado encontra-se em [Geh82], onde é apresentada sua especificação em forma algébrica. Assim, a especificação em VDM apresentada neste trabalho possibilita a comparação entre os dois tipos de especificação, a ser realizada num trabalho futuro.

2 O Método VDM

Em VDM, como já foi mencionado anteriormente, a especificação de um sistema é representada como um modelo explícito do sistema. O modelo em VDM é dado por um estado global do sistema que está sendo modelado e por operações sobre esse estado.

A linguagem utilizada para a especificação de sistemas no método VDM é conhecida como META-IV. Na linguagem META-IV estão definidos os objetos matemáticos que podem ser usados para construir o modelo de um sistema. Nas seções seguintes é feita uma apresentação sucinta desses objetos. Uma descrição mais detalhada dessa linguagem pode ser encontrada em [Bjo89].

2.1 Estados e Domínios Semânticos.

Estados e Domínios Semânticos representam o modelo dos dados internos do sistema. A semântica do que se está definindo é dada em termos da semântica de objetos matemáticos já conhecidos, tais como: conjuntos, seqüências ou tuplas, produtos cartesianos ou árvores e mapeamentos.

Além da definição dos domínios, é usual definir também regras de boa formação e invariantes. Regras de boa formação são regras que restringem quais elementos efetivamente devem ser considerados como pertencentes ao objeto em especificação; invariantes são as restrições de consistência interna que devem ser mantidas sobre o objeto.

2.2 Tipos Abstratos de Dados

Tipos Abstratos de Dados são os objetos matemáticos que podem ser utilizados para modelar um sistema.

Domínios Básicos

Definem elementos não decomponíveis, tais como: valores lógicos, naturais, inteiros, racionais e outros conjuntos obtidos por enumeração de elementos atômicos.

Conjuntos

Esta abstração de tipo de dados deve ser usada quando o objeto sujeito à modelagem possuir propriedades tais que ele possa ser considerado como uma coleção de elementos desordenados, não distinguíveis, mas distintos. Além disso, as operações sobre esse objeto podem envolver a seleção arbitrária de objetos componentes e a remoção ou adição de objetos distintos.

Os operadores fornecidos pela linguagem para a manipulação de conjuntos são os operadores usuais da teoria dos conjuntos:

i) união.

ii) intersecção.

iii) diferença.

iv) conjunto potência (conjunto de todos os subconjuntos).
Seja A = $\{a, b\}$, então A-set = $\{\emptyset, \{a\}, \{b\}, \{a, b\}\}$

v) união distribuída.
union A = $\{x | (\exists C \subseteq A)(x \in C)\}$, onde A é um conjunto de conjuntos.
Exemplo: union $\{\{a, b\}, \{c\}, \{d, e\}\} = \{a, b, c, d, e\}$

vi) cardinalidade (número de elementos de um conjunto).
card $\{a, b, c\} = 3$

Seqüências

O tipo de dado seqüência deve ser usado quando o objeto sujeito à abstração possuir propriedades tais que seus componentes podem melhor ser tratados como sendo ordenados e que seja natural se falar de um primeiro elemento, um segundo elemento e assim por diante. O uso de seqüências também é recomendado quando as manipulações do objeto envolverem composição de pares ou seqüências de tais objetos, ou envolverem indagações sobre seu comprimento ou sobre o conjunto de seus elementos.

A linguagem fornece os seguintes operadores para a manipulação de seqüências:

i) comprimento.
len $< a, b, c >= 3$

ii) aplicação.

Seja $L =< a, b, c >$, então $L[2] = b$

iii) conjunto de índices.

Seja $L =< a, b, c >$, então $\underline{\text{inds}}\ L = \{1, 2, 3\}$

iv) conjunto de elementos.

$\underline{\text{elems}}\ L = \{L[i]\ |i \in \underline{\text{inds}}\ L\}$

v) primeiro elemento (cabeça).

$\underline{\text{hd}}\ L = L[1]$

vi) cauda de uma seqüência.

$\underline{\text{tl}} < a, b, c > \ = \ < b, c >$

vii) concatenação de duas seqüências.

Sejam $L1 =< a, b, c >$ e $L2 =< d, e >$, então $L1^{\frown}L2 =< a, b, c, d, e >$

Mapeamentos

A abstração mapeamento é usada quando o objeto sujeito à abstração possuir propriedades tais que ele possa ser considerado como uma coleção finita de elementos unicamente distinguíveis. Se, além disso, manipulações do objeto envolverem buscar um elemento ou estender o objeto com um novo elemento, então a abstração mapeamento pode ser ainda mais apropriada.

Um mapeamento pertence à classe de funções que satisfazem a restrição de serem finitas. Portanto, as operações entre mapeamentos são operações entre funções. São usadas as operações: domínio ($\underline{\text{dom}}$), contra-domínio ($\underline{\text{rng}}$), união, etc. Em seguida, são dados dois exemplos que mostram o uso dos operadores de restrição e um exemplo para o operador de sobreposição. Os mapeamentos podem ser totais ($\mapsto$) ou parciais ($\xrightarrow{\sim}$).

Sejam os mapeamentos $M = [a \mapsto 1, b \mapsto 3, c \mapsto 2, d \mapsto 2]$, $N = [c \mapsto 4]$ e o conjunto $C = \{a, b, c\}$:

i) "\" (restrito por).

$M \backslash C = [d \mapsto 2]$

ii) "/" (restrito a).

$M/C = [a \mapsto 1, b \mapsto 3, c \mapsto 2]$

iii) "+" (sobreposição).

$M + N = [a \mapsto 1, b \mapsto 3, c \mapsto 4, d \mapsto 2]$

Árvores

Quando o objeto a ser modelado possuir uma estrutura composta e essa estrutura consistir de um número fixo de componentes, então o uso da abstração árvore é recomendado. Justifica-se também o uso de árvores quando se quiser pensar nos componentes do objeto individualmente ou na comparação de duas estruturas.

Árvores, na verdade, correspondem ao conceito de produto cartesiano. São fornecidos dois operadores para árvores: um para construir registros (mk) e outro para se ter acesso aos campos (seletor). Nomes de seletores podem ser definidos explicitamente ou implicitamente.

i) definição explícita.

Dado o domínio de objetos do tipo árvore abaixo, temos:

$D :: {}^{1} s\text{-}id_1 : D_1 \times s\text{-}id_2 : D_2 \times \cdots \times s\text{-}id_n : D_n$

$s\text{-}id_1(\underline{\text{mk}}\text{-}D(d_1, d_2, \ldots, d_n)) = d_1$

ii) definição implícita.

Assumindo, para algum ou todo i com $i = 1, \ldots, n$, a unicidade de D_i e em particular que D_i é um identificador em $D :: D_1 \times D_2 \times \cdots \times D_n$, temos:

$s\text{-}D_i(\underline{\text{mk}}\text{-}D(d_1, d_2, \ldots, d_n)) = d_i$

2.3 Sintaxe Abstrata e Domínios Sintáticos

A manipulação dos objetos semânticos é efetivada através da execução de comandos. Na especificação deve ser incluída então, a sintaxe abstrata dos comandos. Por sintaxe abstrata entende-se a definição e construção dos domínios envolvidos na sintaxe dos comandos, que são chamados de Domínios Sintáticos.

Da mesma forma que na definição da semântica, devem ser apresentadas regras de boa formação que caracterizam os objetos em questão.

[1] O símbolo :: é o símbolo usado na definição de domínios sintáticos de objetos do tipo árvore.

2.4 Funções

Para finalizar, devem ser definidas as funções que serão executadas pelo sistema. Junto com as funções que implementam os aspectos principais da especificação costuma-se também definir funções auxiliares que modularizam e tornam mais legíveis as especificações. Existem dois estilos para a construção dessas funções: o estilo aplicativo e o estilo imperativo.

O estilo aplicativo corresponde à programação funcional, onde as funções são definidas com construtores aplicativos, não exibindo a noção de seqüencialidade e mudanças de estados. Uma especificação tem estilo imperativo se essas noções aparecem e existe ao menos um comando imperativo (Ex: for, while, atribuição, etc.). No exemplo apresentado neste trabalho utiliza-se o estilo aplicativo.

3 Especificação em VDM do sistema *EVENT LOG*

O sistema *EVENT LOG* é um subsistema do sistema de gerenciamento de software denominado 'Change Management Automated Build System'. A especificação informal do sistema *EVENT LOG*, enunciada por Gehani em [Geh82], é a seguinte:

Associado com cada *release* de um produto há um arquivo *eventlog* no qual informações sobre a situação do produto são registradas, referindo-se a todas as versões (chamadas *builds*) de um release. O desenvolvimento de um novo release do produto é visto como a construção de uma série de versões começando do release anterior. A versão final é designada como o novo release. O eventlog é atualizado e consultado através das seguintes operações:

create(r): cria um eventlog r nulo.

newbuild(r): atualiza o eventlog r para permitir o registro de modificações para construir a nova versão (build). O novo número de build é escrito no arquivo padrão. Números de novas versões variam de 1 a 999. Nota: várias versões são associadas a um release.

append(r,b,e,t): o texto t é associado com o evento e do build b no eventlog r. Textos anteriores associados com o evento e não são afetados.

extract(r,b,le): recupera o texto correspondente aos eventos na lista *le*, associados com o build *b* do eventlog *r*. O texto é prefixado pelo evento correspondente. A ordem de recuperaçao é idêntica à ordem na qual os eventos foram fornecidos. Se a lista de eventos *le* não é informada, então a lista de eventos é assumida como sendo a lista de todos os eventos associados com o build *b*.

delete(r,b,e): o evento *e* do build *b* e o texto associado a ele são removidos do eventlog *r*.

A destruição de um eventlog é realizada pela destruição de sua representação subjacente, que supõe-se ser um arquivo.

3.1 Domínios Semânticos e Sintáticos

Abaixo, têm-se os Domínios Semânticos e Sintáticos do subsistema *EVENT LOG*. A especificação foi feita em inglês, para preservar os termos chaves do sistema, como "event", "build", etc. e facilitar a comparação com a especificação algébrica do mesmo problema, apresentada por Gehani.

Domínios Semânticos.

1. $BD = Name \rightarrow\!\!\!\!\!\!\pi \ \ Eventlog$
2. $Eventlog = \#Build \rightarrow\!\!\!\!\!\!\pi \ \ Annotation$
3. $Annotation = (Event \times Texts)^*$
4. $Texts = Text^+$
5. $\#Build = \{0, 1, 2, ..., 999\}$
6. $Event = TOKEN$
7. $Text = TOKEN$
8. $Name = TOKEN$

Domínios Sintáticos.

9. $EVL = Createbd \mid Create \mid Newbuild \mid Append \mid Extract \mid Delete \mid Destroy$
10. $Create :: Name$

11. $Newbuild :: Name$

12. $Append :: Name \times \#Build \times Event \times Text$

13. $Extract :: Name \times \#Build \times Eventlist$

14. $Delete :: Name \times \#Build \times Event$

15. $Destroy :: Name$

16. $Eventlist :: Event\text{-}\underline{\text{set}}$

No Apêndice é apresentado um exemplo hipotético do domínio semântico, ilustrando-se também a execução das funções que alteram o estado do domínio semântico.

No restante do texto, as seguintes variáveis serão utilizadas, com seus respectivos tipos: n: $Name$, bd: BD, evl: $Eventlog$, e: $Event$, t: $Text$, m: $Annotation$, b: $\#Build$ e le: $Eventlist$.

Todos os números de versões (builds) de um determinado eventlog da base de dados devem estar no intervalo de 1 a 999. Isso dá origem ao seguinte invariante do domínio semântico:

17. $\underline{\text{inv}}\text{-}BD(bd)(n) \triangleq$
.1 $\quad \underline{\text{let}}\ evl = bd(n)\ \underline{\text{in}}$
.2 $\quad\quad (evl = [\,]) \vee (\forall b \in \underline{\text{dom}}\ evl)(1 \leq b \leq 999)$
17.3 $\underline{\text{type}} : BD \rightarrow Name \rightarrow BOOL$

3.2 FUNÇÕES

A seguir, são apresentadas as especificações das funções do sistema *EVENT LOG*.

Função $Elab\text{-}Createbd$:

18. $\underline{\text{type}} : Elab\text{-}Createbd : \rightarrow BD$
18.1 $Elab\text{-}Createbd(\,) \triangleq [\,]$

Anotações para $Elab\text{-}Createbd$.

.1 $Elab\text{-}Createbd$ cria uma base de dados nula.

Função *Elab−Create*:

19. type : $Elab-Create : Create \rightarrow (BD \rightarrow BD)$
.1 $Elab-Create[\![\text{mk}-Create(n)]\!](bd) \triangleq$
.2 if $n \in \text{dom } bd$
.3 then error
19.4 else $bd \cup [n \mapsto [\,]]$

Anotações para *Elab−Create*.

.1 *Elab−Create* toma uma base de dados *bd* e um nome *n*

.2 se já existe um eventlog na base de dados identificado por *n*

.3 então, erro

.4 senão, a base de dados é atualizada com *n* identificando um novo eventlog nulo.

Função *Elab−Newbuild*:

20. type : $Elab-Newbuild : Newbuild \rightarrow (BD \rightarrow BD)$
.1 $Elab-Newbuild[\![\text{mk}-Newbuild(n)]\!](bd) \triangleq$
.2 if $n \in \text{dom } bd$
.3 then let $evl = bd(n)$ in
.4 if $lastbuild(evl) = 999$
.5 then error
.6 else (let $evl' = evl \cup [\,(lastbuild(evl) + 1) \mapsto <>\,]$ in
.7 $bd + [\,n \mapsto evl'\,]$)
20.8 else error

Anotações para *Elab−Newbuild*.

.1 *Elab−Newbuild* toma uma base de dados *bd* e um nome *n*

.2 se *n* é o nome de um eventlog já existente na base de dados

.3 então, seja *evl* o eventlog identificado por *n*

.4 se o número da última versão alcançou o limite 999

.5 então, erro

.6 senão, crie uma nova versão para o eventlog *evl*, identificada pelo sucessor do número da última versão

.7 e atualize a base de dados

.8 senão, erro.

21. type : *lastbuild* : *Eventlog* → #*Build*
.1 *lastbuild*(*evl*) ≜
.2 if *evl* = [ ]
.3 then 0
21.4 else card (dom *evl*)

Anotações para *lastbuild*.

.1 A função *lastbuild* retorna o número da última versão de um dado eventlog *evl*.

.2 se *evl* for nulo

.3 então, retorna zero (não existe nenhuma versão)

.4 senão, retorna o número de versões existentes do eventlog *evl*.

Função *Elab*−*Append*:

22. type : *Elab*−*Append* : *Append* → (*BD* → *BD*)
.1 *Elab*−*Append*[mk−*Append*(*n*, *b*, *e*, *t*)](*bd*) ≜
.2 if (*n* ∈ dom *bd*) ∧ (*b* ∈ dom *bd*(*n*))
.3 then let *evl* = *bd*(*n*) in
.4 (let *m* = *evl*(*b*) in
.5 if (∃*i* ∈ inds *m*)(*m*[*i*] = (*e*,))
.6 then (let *j* ∈ inds *m* be s.t. *m*[*j*] = (*e*, *t'*) in

```
.7                    (let m' = m + [ j ↦ (e, t' ^ < t >) ] in
.8                        bd + [ n ↦ evl + [ b ↦ m' ] ] ))
.9                else (let m' = m ^ < (e, < t >) > in
.10                       bd + [ n ↦ evl + [ b ↦ m' ] ] ))
22.11         else error
```

Anotações para *Elab−Append*.

.1 *Elab−Append* toma uma base de dados bd, um nome n, um número de versão b, um evento e e um texto t

.2 se n é o nome de um eventlog já existente na base de dados e b é um número de versão válido, isto é, o número de alguma versão do eventlog identificado por n

.3 então, seja evl o eventlog identificado por n

.4 com relação à versão b, seja m a seqüência de pares do tipo evento e seus textos correspondentes

.5 se já existe em m um par correspondendo ao evento e

.6 então, seja este par (e, t')

.7 troque (e, t') por $(e, t' \hat{} < t >)$

.8 e atualize a base de dados

.9 senão, concatene com m o par $(e, < t >)$

.10 e atualize a base de dados

.11 senão, erro.

Função *Elab−Extract*:

```
23.  type : Elab−Extract : Extract → (BD → Annotation)
 .1  Elab−Extract[[mk−Extract(n, b, le)]](bd) ≜
 .2        if (n ∈ dom bd) ∧ (b ∈ dom bd(n))
 .3        then let evl = bd(n) in
 .4                (let m = evl(b) in
 .5                   if le = { }
```

.6 then m
.7 else (let $le' = \{e | (e \in le) \wedge (\exists (e', t') \in \underline{\text{elems}}\ m)(e' = e)\}$ in
.8 $extractlist(m, le')$))
23.9 else error

Anotações para $Elab-Extract$.

.1 $Elab-Extract$ toma uma base de dados bd, um nome n, um número de versão b e uma lista de eventos le

.2 se n é o nome de um eventlog já existente na base de dados e b é um número de versão válido

.3 então, seja evl o eventlog identificado por n

.4 com relação à versão b, seja m a seqüência de pares do tipo evento e seus textos correspondentes

.5 se não existir nenhum evento em le

.6 então, retorne todos os pares de eventos com seus textos correspondentes, isto é, m

.7 senão, seja le' o conjunto de eventos de le que fazem parte de pares em m, isto é, o conjunto de eventos em le referentes à versão b

.8 extraia de m os eventos em le' com seus textos correspondentes, usando a função auxiliar $extractlist$

.9 senão, erro.

24. type : $extractlist : Annotation \times Eventlist \rightarrow Annotation$
.1 $extractlist(m, le) \triangleq$
.2 if $le = \{\ \}$
.3 then $< >$
.4 else (let $e \in le$ be s.t. $(\forall x \in le)(x \neq e \Rightarrow before(e, x, m))$ in
.5 (let $(e', t') \in$ elems m be s.t. $e' = e$ in
24.6 $< (e, t') > \hat{}\ extractlist(m, le \backslash \{e\})$))

Anotações para *extractlist*.

.1 Dados uma seqüência m de pares do tipo evento e seus textos correspon-

dentes e um conjunto de eventos le

.2 se o conjunto for vazio

.3 então, retorne a seqüência vazia

.4 senão, seja e o evento que pertence ao conjunto le e que foi registrado antes

de qualquer outro evento em le

.5 seja (e, t') o par registrado

.6 retorne o par (e, t') concatenado com o resultado da aplicação recursiva de *extractlist* em m e o conjunto le sem o evento e.

25. type : $before : Event \times Event \times Annotation \rightarrow BOOL$
.1 $before(e, e', m) \triangleq$
.2 let $i, j \in$ inds m be s.t. $(m[i] = (e,)) \wedge (m[j] = (e',))$ in
25.3 $i < j$

Anotações para *before*.

.1 Um evento e foi registrado antes que outro e', se o índice i correspondente ao par ao qual ele pertence for menor que o índice j correspondente ao par ao qual o evento e' pertence.

Função *Elab−Delete*:

26. type : $Elab-Delete : Delete \rightarrow (BD \rightarrow BD)$
.1 $Elab-Delete[\text{mk}-Delete(n, b, e)](bd) \triangleq$
.2 if $(n \in \text{dom } bd) \wedge (b \in \text{dom } bd(n))$
.3 then let $evl = bd(n)$ in
.4 (let $m = evl(b)$ in
.5 $bd + [\, n \mapsto evl + [\, b \mapsto del(e, m)\,]\,]\,)$
26.6 else error

Anotações para *Elab–Delete*.

.1 *Elab–Delete* toma uma base de dados *bd*, um nome n, um número de versão b e um evento e

.2 se n é o nome de um eventlog já existente na base de dados e b é um número de versão válido

.3 então, seja *evl* o eventlog identificado por n

.4 com relação à versão b, seja m a sequência de pares do tipo evento e seus textos correspondentes

.5 remova e da sequência m através da função auxiliar *del* e atualize a base de dados

.6 senão, erro.

```
27.  type : del : Event × Annotation → Annotation
 .1 del(e, m) ≜
 .2        if m = < >
 .3        then  < >
 .4        else (let hd m = (e', t') in
 .5                  if e = e'
 .6                  then tl m
27.7                  else  < (e', t') > ^ del(e, tl m) )
```

Anotações para *del*.

.1 *del* é uma função recursiva que busca o evento e na seqüência m e se encontrar retira o par correspondente.

.2 se m for vazia

.3 então, retorne a seqüência vazia

.4 senão, obtenha a cabeça da seqüência m

.5 se o evento e estiver na cabeça de m

.6 então, retorne a cauda de m

.7 senão, retorne a cabeça de m concatenada com o resultado da aplicação recursiva de $del(e, \underline{\text{tl}}\ m)$.

Função *Elab–Destroy*:

28. $\underline{\text{type}} : Elab\text{–}Destroy : Destroy \rightarrow (BD \rightarrow BD)$
.1 $Elab\text{–}Destroy[\underline{\text{mk}}\text{–}Destroy(n)](bd) \triangleq$
.2 $\underline{\text{if}}\ n \in \underline{\text{dom}}\ bd$
.3 $\underline{\text{then}}\ bd \backslash \{n\}$
28.4 $\underline{\text{else}}\ bd$

Anotações para *Elab–Destroy*.

.1 Dados um nome n e uma base de dados bd

.2 se n identifica algum eventlog em bd

.3 então, remova n e o eventlog correspondente

.4 senão, a base de dados não se modifica.

4 Outra interpretação para a especificação informal

A especificação informal da função *extract* requer que "a ordem de recuperação seja idêntica à ordem na qual os eventos foram fornecidos", o que pode dar margem a duas interpretações distintas : a primeira seria a ordem em que os eventos aparecem na lista *le* e a segunda interpretação seria a ordem na qual os eventos foram registrados na base de dados. Esta última interpretação é mais comum e foi utilizada na especificação mostrada na seção anterior.

A primeira interpretação, menos usual, tem alguns pontos distintos que valem a pena ser mostrados. As principais diferenças são: na primeira interpretação se está interessado na ordem em que os eventos estão na lista *le*, sendo assim, elementos do domínio *Eventlist* são abstraídos como sendo seqüências (regra 44); torna-se desnecessário saber a ordem em que os eventos foram registrados, portanto, elementos do domínio *Annotation* podem ser abstraídos por

mapeamentos de eventos em seus textos correspondentes (regra 31). Os domínios semânticos e sintáticos para esta nova interpretação são apresentados a seguir.

4.1 Domínios Semânticos e Sintáticos

Domínios Semânticos.

29. $BD = Name \pitchfork Eventlog$

30. $Eventlog = \#Build \pitchfork Annotation$

31. $Annotation = Event \pitchfork Texts$

32. $Texts = Text^{+}$

33. $\#Build = \{0, 1, 2, ..., 999\}$

34. $Event = TOKEN$

35. $Text = TOKEN$

36. $Name = TOKEN$

Domínios Sintáticos.

37. $EVL = Createdb \mid Create \mid Newbuild \mid Append \mid Extract \mid Delete \mid Destroy$

38. $Create :: Name$

39. $Newbuild :: Name$

40. $Append :: Name \times \#Build \times Event \times Text$

41. $Extract :: Name \times \#Build \times Eventlist$

42. $Delete :: Name \times \#Build \times Event$

43. $Destroy :: Name$

44. $Eventlist :: Event^{*}$

4.2 Funções

As funções *Elab−Createbd*, *Elab−Create*, *lastbuild* e *Elab−Destroy* não se alteram. As demais são apresentadas em seguida, com as modificações causadas pela interpretação diferente:

Função *Elab−Newbuild*:

45. type : $Elab-Newbuild : Newbuild \rightarrow (BD \rightarrow BD)$
.1 $Elab-Newbuild[\![\underline{\text{mk}}-Newbuild(n)]\!](bd) \triangleq$
.2 if $(n \in \underline{\text{dom}}\ bd) \wedge (b \in \underline{\text{dom}}\ bd(n))$
.3 then let $evl = bd(n)$ in
.4 if $lastbuild(evl) = 999$
.5 then error
.6 else (let $evl' = evl \cup [\,(lastbuild(evl) + 1) \mapsto [\,]\,]$ in
.7 $bd + [\,n \mapsto evl'\,]$)
45.8 else error

.1 A diferença entre a primeira versão é que o mapeamento agora vai do novo número de versão para o mapeamento vazio (.6), ao invés da seqüência vazia.

Função *Elab−Append*:

46. type : $Elab-Append : Append \rightarrow (BD \rightarrow BD)$
.1 $Elab-Append[\![\underline{\text{mk}}-Append(n, b, e, t)]\!](bd) \triangleq$
.2 if $(n \in \underline{\text{dom}}\ bd) \wedge (b \in \underline{\text{dom}}\ bd(n))$
.3 then let $evl = bd(n)$ in
.4 (let $m = evl(b)$ in
.5 if $e \in \underline{\text{dom}}\ m$
.6 then (let $m' = m + [\,e \mapsto (m(e)^{\frown} < t >)\,]$ in
.7 $bd + [\,n \mapsto (evl + [\,b \mapsto m'\,])\,]$)
.8 else (let $m' = m \cup [\,e \mapsto < t >\,]$ in
.9 $bd + [\,n \mapsto (evl + [\,b \mapsto m'\,])\,]$))
46.10 else error

Anotações para *Elab−Append*.

.1 *Elab−Append* toma uma base de dados *bd*, um nome n, um número de versão b, um evento e e um texto t

.2 se n é o nome de um eventlog já existente na base de dados e b é um número de versão válido

.3 então, seja *evl* o eventlog identificado por n

.4 com relação à versão b, seja m o mapeamento de eventos aos seus textos correspondentes

.5 se o evento e é um evento do domínio de m, isto é, se já existe algum texto correspondente ao evento e

.6 então, associe o texto t com o evento e

.7 e atualize a base de dados

.8 senão, inclua o evento e com seu texto correspondente t em m

.9 e atualize a base de dados

.10 senão, erro.

Função *Elab−Extract*:

```
47.  type : Elab−Extract : Extract → (BD → Annotation)
 .1  Elab−Extract[[mk−Extract(n, b, le)]](bd) ≜
 .2        if (n ∈ dom bd) ∧ (b ∈ dom bd(n))
 .3        then let evl = bd(n) in
 .4                  if le =<>
 .5                  then let m = evl(b) in
 .6                            m
 .7                  else extractlist(evl, b, le)
47.8       else error
```

Anotações para *Elab−Extract*.

.1 *Elab−Extract* toma uma base de dados *bd*, um nome n, um número de versão b e uma lista de eventos *le*

.2 se n é o nome de um eventlog já existente na base de dados e b é um número de versão válido

.3 então, seja evl o eventlog identificado por n

.4 se a lista de eventos le for vazia

.5 então, seja m o mapeamento de eventos aos seus textos correspondentes

.6 o resultado é o próprio mapeamento m

.7 senão, extrair os textos correspondentes aos eventos na lista le

.8 senão, erro.

48. type : $extractlist : Eventlog \times \#Build \times Eventlist \rightarrow Annotation$

.1 $extractlist(evl, b, le) \triangleq$
.2 if $le = <>$
.3 then [ ]
.4 else (let $m = evl(b), e =$ hd le in
.5 if $e \in$ dom m
.6 then $[e \mapsto m(e)] + extractlist(evl, b,$ tl $le)$
48.7 else $extractlist(evl, b,$ tl le))

Anotações para *extractlist*.

.1 Dados o eventlog evl, o número de versão b e a lista de eventos le, a função *extractlist* é chamada recursivamente até que seja satisfeita a condição de parada.

.2 se a lista de eventos le for vazia

.3 então, fim da recursão e retorna o mapeamento vazio

.4 senão, seja m o mapeamento de eventos aos seus textos correspondentes e seja e a cabeça da lista le

.5 se o evento e pertence ao domínio de m

.6 então, retorna os textos correspondentes ao evento e mais os textos correspondentes aos eventos na cauda da lista le

.7 senão, retorna apenas os textos correspondentes aos eventos na cauda da lista *le*.

Função *Elab–Delete*:

```
49.  type : Elab–Delete : Delete → (BD → BD)
 .1  Elab–Delete[mk–Delete(n, b, e)](bd) ≜
 .2        if (n ∈ dom bd) ∧ (b ∈ dom bd(n))
 .3        then let evl = bd(n) in
 .4                  (let m = evl(b) in
 .5                    if e ∈ dom m
 .6                    then (let evl' = evl + [ b ↦ m\{e} ] in
 .7                              bd + [ n ↦ evl' ] )
 .8                    else bd )
49.9       else error
```

Anotações para *Elab–Delete*.

.1 *Elab–Delete* toma uma base de dados *bd*, um nome *n*, um número de versão *b* e um evento *e*

.2 se *n* é o nome de um eventlog já existente na base de dados e *b* é um número de versão válido

.3 então, seja *evl* o eventlog identificado por *n*

.4 seja *m* o mapeamento de eventos aos seus textos correspondentes

.5 se o evento *e* pertence ao domínio de *m*

.6 então, remova o evento *e* com seus textos correspondentes

.7 e atualize a base de dados

.8 senão, a base de dados não se modifica

.9 senão, erro.

5 Considerações Finais

Este trabalho apresentou a especificação formal em VDM do sistema *EVENT LOG*, enunciado por Gehani em [Geh82]. A especificação realizada utilizou as principais abstrações de VDM, ilustrando também a dificuldade de interpretação de especificações informais (como a linguagem natural) através da apresentação de uma segunda versão da especificação, causada por dificuldades no entendimento do significado da função *extract*.

A partir dessa especificação, segue-se um processo no qual domínios mais concretos podem ser desenvolvidos, devendo-se provar que o domínio mais concreto é equivalente, isto é, é uma implementação do domínio mais abstrato anterior e assim por diante, até que se atinja um nível de detalhamento em que seja possível a implementação da especificação em alguma linguagem de programação.

Referências

[Bjo89] Bjorner, D. - 'The VDM Specification & Implementation Methodology', State of the Art Seminar on Formal Description of Programming Concepts, Petrópolis, 1989.

[Geh82] Gehani, N.H. - 'Specifications: Formal and Informal - A Case Study' Software - Practice and Experience, 12, pp 433-444, 1982. Reprinted in: Gehani, N.H. and McGettrick, A. - Software Specifications Techniques, Addison-Wesley, pp 408-419, 1986.

[Gut80] Guttag, J.V. and Horning, J.J. - 'Formal Specification as a Design Tool', Seventh Annual ACM Symposium on Principles of Programming Languages, pp 251-261, 1980. Reprinted in: Gehani, N.H. and McGettrick, A. - Software Specifications Techniques, Addison-Wesley, pp 187-207, 1986.

[Mar89] Martins, R.C.B., Annarumma, G.O., Carneiro, L.M.F., Lima, A.V., Molinari, P., Quintana, E.M.B. e Stern, R. - 'Métodos Formais para Desenvolvimento de Software: Um estudo de caso, o projeto GARDEN', III Simpósio Brasileiro de Engenharia de Software, pp 201-213, 1989.

A Exemplo hipotético do Domínio Semântico

Suponha que se tenha num determinado instante a seguinte base de dados:

$$
\begin{array}{l}
bd = [\ evl_1 \mapsto [1 \mapsto < (e_1, < t_1, t_2, t_3 >) >, \\
\qquad\qquad\quad 2 \mapsto < (e_2, < t_4 >) >, \\
\qquad\qquad\quad 3 \mapsto < (e_3, < t_5, t_6 >) > \\
\qquad\qquad\quad], \\
\qquad evl_2 \mapsto [1 \mapsto < (e_4, < t_7, t_8 >), (e_5, < t_9 >) > \\
\qquad\qquad\quad], \\
\qquad evl_3 \mapsto [1 \mapsto < (e_6, < t_{10}, t_{11} >) >, \\
\qquad\qquad\quad 2 \mapsto < (e_7, < t_{12}, t_{13}, t_{14} >), (e_8, < t_{15}, t_{16} >) > \\
\qquad\qquad\quad] \\
\quad]
\end{array}
$$

Aplicando-se as funções *Elab−Create*, *Elab−Newbuild*, *Elab−Append*, *Elab−Delete* e *Elab−Destroy*, nesta ordem, têm-se as seguintes mudanças na base de dados:

a) $Elab-Create[\underline{\text{mk}}-Create(evl_4)](bd)$

$$
\begin{array}{l}
bd = [\ evl_1 \mapsto [1 \mapsto < (e_1, < t_1, t_2, t_3 >) >, \\
\qquad\qquad\quad 2 \mapsto < (e_2, < t_4 >) >, \\
\qquad\qquad\quad 3 \mapsto < (e_3, < t_5, t_6 >) > \\
\qquad\qquad\quad], \\
\qquad evl_2 \mapsto [1 \mapsto < (e_4, < t_7, t_8 >), (e_5, < t_9 >) > \\
\qquad\qquad\quad], \\
\qquad evl_3 \mapsto [1 \mapsto < (e_6, < t_{10}, t_{11} >) >, \\
\qquad\qquad\quad 2 \mapsto < (e_7, < t_{12}, t_{13}, t_{14} >), (e_8, < t_{15}, t_{16} >) > \\
\qquad\qquad\quad], \\
\qquad evl_4 \mapsto [\,] \\
\quad]
\end{array}
$$

b) $Elab\text{-}Newbuild[\![\underline{\text{mk}}\text{-}Newbuild(evl_4)]\!](bd)$

$$
\begin{aligned}
bd = [\ & evl_1 \mapsto [1 \mapsto < (e_1, < t_1, t_2, t_3 >) >, \\
& \qquad 2 \mapsto < (e_2, < t_4 >) >, \\
& \qquad 3 \mapsto < (e_3, < t_5, t_6 >) > \\
& \qquad], \\
& evl_2 \mapsto [1 \mapsto < (e_4, < t_7, t_8 >), (e_5, < t_9 >) > \\
& \qquad], \\
& evl_3 \mapsto [1 \mapsto < (e_6, < t_{10}, t_{11} >) >, \\
& \qquad 2 \mapsto < (e_7, < t_{12}, t_{13}, t_{14} >), (e_8, < t_{15}, t_{16} >) > \\
& \qquad], \\
& evl_4 \mapsto [1 \mapsto < > \\
& \qquad] \\
]
\end{aligned}
$$

c) $Elab\text{-}Append[\![\underline{\text{mk}}\text{-}Append(evl_4, 1, e_9, t_{17})]\!](bd)$

$$
\begin{aligned}
bd = [\ & evl_1 \mapsto [1 \mapsto < (e_1, < t_1, t_2, t_3 >) >, \\
& \qquad 2 \mapsto < (e_2, < t_4 >) >, \\
& \qquad 3 \mapsto < (e_3, < t_5, t_6 >) > \\
& \qquad], \\
& evl_2 \mapsto [1 \mapsto < (e_4, < t_7, t_8 >), (e_5, < t_9 >) > \\
& \qquad], \\
& evl_3 \mapsto [1 \mapsto < (e_6, < t_{10}, t_{11} >), >, \\
& \qquad 2 \mapsto < (e_7, < t_{12}, t_{13}, t_{14} >), (e_8, < t_{15}, t_{16} >) > \\
& \qquad], \\
& evl_4 \mapsto [1 \mapsto < (e_9, < t_{17} >) > \\
& \qquad] \\
]
\end{aligned}
$$

d) $Elab-Delete[\underline{\text{mk}}-Delete(evl_3, 2, e_8)](bd)$

$$
\begin{aligned}
bd = [\ & evl_1 \mapsto [1 \mapsto < (e_1, < t_1, t_2, t_3 >) >, \\
& \qquad 2 \mapsto < (e_2, < t_4 >) >, \\
& \qquad 3 \mapsto < (e_3, < t_5, t_6 >) > \\
& \qquad], \\
& evl_2 \mapsto [1 \mapsto < (e_4, < t_7, t_8 >), (e_5, < t_9 >) > \\
& \qquad], \\
& evl_3 \mapsto [1 \mapsto < (e_6, < t_{10}, t_{11} >) >, \\
& \qquad 2 \mapsto < (e_7, < t_{12}, t_{13}, t_{14} >) > \\
& \qquad], \\
& evl_4 \mapsto [1 \mapsto < (e_9, < t_{17} >) > \\
& \qquad] \\
]
\end{aligned}
$$

e) $Elab-Destroy[\underline{\text{mk}}-Destroy(evl_2)](bd)$

$$
\begin{aligned}
bd = [\ & evl_1 \mapsto [1 \mapsto < (e_1, < t_1, t_2, t_3 >) >, \\
& \qquad 2 \mapsto < (e_2, < t_4 >) >, \\
& \qquad 3 \mapsto < (e_3, < t_5, t_6 >) > \\
& \qquad], \\
& evl_3 \mapsto [1 \mapsto < (e_6, < t_{10}, t_{11} >) >, \\
& \qquad 2 \mapsto < (e_7, < t_{12}, t_{13}, t_{14} >) > \\
& \qquad], \\
& evl_4 \mapsto [1 \mapsto < (e_9, < t_{17} >) > \\
& \qquad] \\
]
\end{aligned}
$$

Data 24 / 08 / 90

Proc.

Ped

Liv. Doação

r$