Protótipos Funcionais a partir de Especificações Formais em VDM

Paulo Henrique Monteiro Borba, Silvio Lemos Meira

Departamento de Informática
Universidade Federal de Pernambuco
CP 7851, 50739 Recife - PE - Brasil

Resumo

Descrevemos neste trabalho a transformação de uma especificação formal em VDM (*Vienna Development MethodD* [1] para um protótipo correspondente em Standard ML (SML) [2]. Para facilitar a transformação, utilizamos uma implementação (descrita em [3]) em SML dos objetos matemáticos (*sets*, *maps*, *seqs*) e operações associadas de VDM. O enfoque utilizado para a transformação é completamente formal.

Abstract

We describe the transformation of a VDM (*Vienna Development MethodD* [1] formal specification into a Standard ML (SML) [2] prototype. To help the transformation, we use a SML implementation (described in [3]) of the mathematical objects (*sets*, *maps*, *seqs*) and related operations of VDM. The approach used in the transformation is completly formal and general.

1. Introdução

A utilização de métodos formais no processo de desenvolvimento de sistemas complexos (como compiladores, sistemas operacionais e geradores de aplicativos) é fundamental quando se deseja reduzir os custos de desenvolvimento e ao mesmo tempo projetar sistemas seguros e corretos [4].

O uso de métodos formais envolve a especificação formal do sistema, o projeto verificado com base na especificação e a implementação verificada com base no projeto.

Através de uma especificação formal é possível atingir um bom nível de precisão e concisão na especificação do sistema. Por meio de um projeto verificado prova-se a corretude do projeto em relação à especificação, e uma implementação verificada torna possível provar que o sistema implementado satisfaz o projeto.

Além de permitir o projeto e a implementação verificada, e assim viabilizar o desenvolvimento formal de software, a especificação formal possibilita a passagem formal (ou pelo menos rigorosa) da especificação para um protótipo do sistema.

Para que a transformação da especificação formal em protótipo seja modelada formalmente é interessante, para facilitar o processo, que a linguagem de especificação e a linguagem de prototipagem tenham sintaxe e semântica bem definidas e sejam semanticamente próximas uma da outra. Suportando os mesmos objetos e operações da linguagem de especificação na linguagem de protótipo contribui neste sentido.

Por este motivo, neste trabalho utilizamos a implementação descrita em [3], onde é apresentada a notação para representar os objetos e operações de VDM em SML, e é provado que a implementação é correta com relação a notação VDM, isto é, verifica-se que a implementação é um modelo da teoria definida pela notação VDM. A seguir, apresentamos uma pequena parte da notação, onde para um tipo T qualquer considera-se que:

$$e : T,$$
$$R, S : \text{set of } T,$$
$$i, j : \mathbb{N},$$
$$p : T \rightarrow \text{bool}.$$

A notação VDM é mostrada no lado esquerdo da seta, e a notação correspondente em SML é apresentada do lado direito:

```
e ∈ R → member(e,R)
R ∪ S → union(R,S)
(i,...,j) → setint (i,j)
(x ∈ S | p(x)) → setcomp (S,p)
```

O tipo *set* é representado em SML por um tipo abstrato implementado por um par (tupla de dois elementos) formado de uma lista (contém os elementos do conjunto) e de um operador de igualdade entre os elementos desta lista. Logo, set(l,eq) é em SML o conjunto de VDM cujos elementos são os presentes na lista l. Abaixo apresentamos duas das funções que implementam o tipo. O resto da implementação e detalhes sobre a mesma pode ser encontrado em [3].

```
abstype 'a set = set of ('a list * ('a * 'a -> bool))
  with   fun member (e,set ([],equ)) = false |
         member (e,set (h::t,equ)) = equ (e,h) orelse
                                     member (e,set (t,equ))
       fun setcomp (set (l,equ),p) = set (filter p l,equ)
  end
```

A função filter, utilizada na implementação, pode ser definida em SML como:

```
fun  filter p [] = [] |
     filter p (h::t) = if (p h) then h::(filter p t)
                       else filter p t
```

e é juntamente com map, fold e outras considerada pré-definida.

2. Especificação Formal em VDM

Para exemplificar a transformação de uma especificação formal em protótipo, utilizaremos uma parte da especificação formal de um sistema de folha de pagamento [5]. O sub-sistema considerado aqui tem como objetos o valor da dedução por dependente, tabela de imposto de renda (IR) e um cadastro de funcionários. A tabela de IR tem informações necessárias para o cálculo do IR de várias faixas salariais. Apresentamos apenas a especificação da operação que calcula o IR de um funcionário. O IR é calculado pela fórmula:

$$ir = (bir * aliq) - ded - (n * dep)$$

Na fórmula, *bir* é o valor base (salário base + vencimentos - débitos) do funcionário para cálculo do IR, *aliqt* e *ded* são respectivamente a alíquota e a dedução (obtidas na tabela de IR) correspondentes à faixa salarial a qual o valor base do IR pertence, *n* é o número de dependentes do funcionário e *dep* é o valor da dedução por dependente. Agora, apresentamos a especificação formal do sistema.

O estado do sistema é modelado pelo seguinte objeto composto:

```
Estadofp :: deddep : ℝ      /* Valor de Dedução por dependente */
            tabir : Tabir                  /* Tabela do IR */
            func : map Cfun to Dfun        /* Funcionários */
```

A tabela de IR é modelada como uma seqüência de dados relativos ao IR (*Dir*) que tem a propriedade especificada por *inv-Tabir*.

```
Dir :: aliqt, ded, teto : ℝ
Tabir = seq of Dir
where inv-Tabir(t) ≜ ∀ i ∈ {1,...,len(t)-1} •
                        teto(t(i)) < teto(t(i+1))
```

O valor do *teto*, para o *i*-ésimo elemento da seqüência é sempre menor que o o *teto* do *i*+1-ésimo elemento. O primeiro elemento da seqüência determina a alíquota (*alqt*) e dedução (*ded*) correspondente à faixa salarial de 0 até o primeiro *teto*. Os outros elementos da seqüência determinam a alíquota e dedução correspondente à faixa salarial entre o *teto* anterior e o *teto* atual.

O cadastro de funcionários é modelado como um mapeamento entre o código de um funcionário e os dados relativos a este.

```
Cfun = ℕ                          /* Código do Funcionário */
Sit = {A,D}
Dfun :: situacao : Sit   /* Funcionário Admitido ou Demitido */
        nome : Text                    /* Nome do Funcionário */
        dep : ℕ                    /* Número de dependentes */
        valorsal : ℝ                    /* Valor do salário */
```

O cálculo do IR de um funcionário é feito pela operação *IR*, que recebe como argumentos o código do funcionário (*cf*) e o valor base do IR (*bir*) e devolve como resultado o valor do IR do funcionário (*ir*). Esta operação tem acesso de leitura à tabela de IR (*tabir*), ao cadastro de funcionários (*func*) e ao valor da dedução por dependente (*deddep*).

```
1 IR(bir:ℝ,cf:Cfun) ir:ℝ
2 ext rd tabir:Tabir, func:map Cfun to Dfun, deddep:ℝ
3 pre len(tabir) ≥ 1 ∧
4     teto(tabir(len(tabir))) ≥ bir ∧
5     cf ∈ dom func
6 post ∃ i ∈ dom tabir •
7  let t = tabir(i)
8  in teto(t) ≥ bir ∧
9     (∀ j ∈ {1,...,i-1} • teto(tabir(j)) < bir) ∧
1     ir = (bir * aliqt(t)) - ded(t) - (dep(func(cf)) * deddep)
```

A pre condição (linhas 3-5) de *IR* especifica as propriedades que têm que ser satisfeitas para que a operação seja realizada. A pós condição (linhas 6-10) especifica a propriedade associada ao estado depois da operação e o resultado desta.

O número de elementos da seqüência *tabir* é dado por len(*tabir*) (linha 3). O *i*-ésimo elemento de *tabir* é *tabir*(*i*) (linha 4). A função dom (linha 5-6) é sobrecarregada e pode receber como parâmetro um mapeamento (map) e retornar o domínio deste mapeamento (5), ou receber uma seqüência (*s*) e retornar {1,...,len *s*} (linha 6). O elemento do contra-domínio do mapeamento *func* correspondente ao elemento do domínio *d* é *func*(*d*) (linha 10).

Para *IR* ser realizada é necessário que a tabela de IR tenha pelo menos um elemento (3), que o valor do campo *teto* do último elemento da seqüência seja maior que o valor base do IR (4) e que o código do funcionário pertença ao domínio do cadastro (5). Se a pré condição for válida, a operação *IR* verifica a que faixa salarial da tabela de IR *bir* pertence e calcula o IR (linha 10) com a alíquota e a dedução desta faixa.

O elemento da seqüência que corresponde à faixa salarial de *bir* tem o valor do campo *teto* maior ou igual a *bir* (8) e todos os outros elementos da seqüência anteriores a este têm o valor do campo *teto* menor que *bir* (linha 9).

3. Protótipo em SML

Nesta seção apresentamos o protótipo em SML do "sistema" especificado em VDM na seção anterior, utilizando a notação descrita em [3]. Desta forma, o estado, os objetos utilizados na especificação formal e as funções definidas explicitamente (construtivamente) podem ser traduzidas diretamente para SML.

Para transformar uma especificação em VDM de uma operação (implementável) para uma função em SML, transformamos primeiro a especificação em uma função implícita (definida com pré e pós condição), ainda em VDM, onde o estado (da operação) antes de realizada a operação é passado como parâmetro para a função e o estado depois de realizada a operação e o resultado da mesma são retornados como resultado da função. A pré e pós condições da função são idênticas às da operação.

A partir desta função em VDM, definimos uma função em SML que corresponde à definição de VDM. Para verificar que a função em SML corresponde (satisfaz) à especificação em VDM utilizamos a obrigação de prova [1] (*satisfaction of specification*) de VDM.

A seguir, descrevemos o protótipo e na seção seguinte apresentamos a prova de satisfação da especificação.

A função em VDM correspondente à operação *IR* é especificada a seguir, onde o estado da operação *IR* é passado como parâmetro para a função *IRFUN*. A pré e pós condições da função são idênticas às da operação.

```
1 IRFUN(bir:ℝ,cf:Cfun,tabir:Tabir,
        func:map Cfun to Dfun,deddep:ℝ) ir:ℝ
2 pre len(tabir) ≥ 1 ∧
3     teto(tabir(len(tabir))) ≥ bir ∧
4     cf ∈ dom func
5 post ∃ i ∈ dom tabir • let t = tabir(i)
6  in teto(t) ≥ bir ∧
7     (∀ j ∈ {1,...,i-1} • teto(tabir(j)) < bir) ∧
8     ir = (bir * aliqt(t)) - ded(t) - (dep(func(cf)) * deddep)
```

No protótipo, representamos o tipo *Text* como um *string*. Os objetos compostos de VDM são modelados como *records* de SML. Assim, *Dir* é implementado por:

```
type Text = string
type Dir = (aliqt:real, ded:real, teto:real)
```

Em SML, (aliqt = a,...):Dir é uma abreviação de (aliqt = a,ded = d,teto = t):Dir. Daí, os seletores do tipo *Dir* são:

```
fun aliqt ((aliqt = a,...):Dir) = a
fun ded ((ded = d,...):Dir) = d
fun teto ((teto = t,...):Dir) = t
```

A definição do tipo *Tabir* é traduzida a seguir. Como na definição deste tipo é especificado um invariante e em SML não é possível especificar tipos com invariantes, apresentamos uma função em SML equivalente ao invariante em VDM.

Assim, as funções de VDM que recebem como parâmetro um elemento de tipo *Tabir*, quando traduzidas para SML receberão um elemento de tipo Tabir e terão que verificar (através da função inv_Tabir) se o parâmetro recebido tem a propriedade estabelecida pelo invariante do tipo *Tabir*. O equivalente em SML a "seq of Dir" é "Dir seq".

```
type Tabir = Dir seq
fun inv_Tabir(t:Tabir)
   = forall setint(1,(lenseq t)-1)
           (fn x => teto(aplseq(x,t)) < teto(aplseq(x+1,t)))
```

A função lenseq retorna o tamanho de uma seqüência e aplseq(x,t) é t(x) em VDM (o x-ésimo elemento da seqüência t). O correspondente a (i,...,j) em SML é setint(i,j).

Os tipos *Cfun*, *Sit* e *Dfun* são implementados em seguida. Para o tipo *Dfun* apresentamos apenas um seletor, já que os outros podem ser definidos de forma similar. Usamos um tipo algébrico de SML para implementar o tipo *Sit*.

```
type Cfun = int
datatype Sit = Adm | Dem
type Dfun = (situacao:Sit, nome:Text, dep:int, valorsal:real)
fun dep ((dep = s,...):Dfun) = s
```

A função em SML (IRSML) correspondente à função *IRFUN* é apresentada abaixo, onde (∀ x ∈ S · p(x)) é implementado por (forall S p). As funções member e setcomp foram definidas na seção 1. A função choiceset recebe um *set* como parâmetro, e se este não for vazio retorna um elemento qualquer do mesmo. A função cond (linha 1), local à IRSML, é definida a partir das linhas 5-7 de *IRFUN*, e recebe como parâmetros um valor base de IR (bir), uma tabela de IR e um elemento do domínio da tabela (x). O resultado é true se o elemento da tabela indexado por x tiver o teto maior ou igual a bir (linha 2) e se todos os outros elementos da tabela abaixo dele tiverem o teto menor que bir (linha 3).

```
1 local fun cond (bir:real) (tabir:Tabir) (x:int) =
2       (teto(aplseq(x,tabir)) >= bir) andalso
3       forall (setint(1,x-1))
                (fn y => teto(aplseq(y,tabir)) < bir)
4 in
```

```
5 fun IRSML (bir:real,cf:Cfun,tabir:Tabir,
            func:(Cfun,Dfun) maps,deddep:real):real =
6       if member(cf,dom func) andalso
7          (lenseq(tabir) >= 1) andalso
8          (teto(aplseq(lenseq tabir,tabir)) >= bir) andalso
9          inv_Tabir(tabir)
10      then let val s = setcomp(domseq tabir,cond bir tabir)
11               val id = choiceset(s)
12               val tt = aplseq(id,tabir)
13           in (bir * aliqt(tt)) - ded(tt) -
                 (dep(apl(func,cf)) * deddep)
14           end
15      else 0.0
16 end
```

A função IRSML recebe os mesmos parâmetros (a menos de notação) que a função em VDM e também retorna um resultado do mesmo tipo (5). A condição do if-then-else (6-9) é uma tradução da pré condição de *IRFUN*, acrescentando o invariante da tabela de IR (9). Se a condição for falsa (a pré condição da função em VDM é falsa), o resultado é 0.0. Caso contrário, define-se s (linha 10) como o conjunto de todos elementos pertencentes ao domínio de tabir que satisfazem cond (só existirá um elemento neste conjunto). Define-se então id (linha 11) como um elemento deste conjunto e tt (linha 12) como o id-ésimo elemento de tabir. Em SML f(d) é representado por apl(f,d), onde f é um mapeamento e d e um elemento do seu domínio. Finalmente na linha 13, calcula-se o valor do IR da mesma maneira que em *IRFUN*.

4. Corretude da Transformação

Antes de provarmos que a implementação em SML satisfaz a especificação em VDM teríamos que provar que a função especificada em VDM é implementável, o que é estabelecido por uma obrigação de prova (*implementability* [1]) de VDM. Em VDM, esta obrigação é geralmente provada de forma rigorosa.

Para a função *IRFUN* a obrigação de prova é:

$$\forall\ bir \in \mathbb{R},\ cf \in Cfun,\ tabir \in \text{seq of } Dir,\ deddep \in \mathbb{R},$$
$$func \in \text{map } Cfun \text{ to } Dfun \cdot$$
$$\text{pre-}IRFUN(bir, cf, tabir, func, deddep) \Rightarrow$$
$$\exists\ ir \in \mathbb{R} \cdot \text{post-}IRFUN(bir, cf, tabir, func, deddep, ir)$$

Esta fórmula estabelece que se a pré condição da operação for verdadeira então existe um número real que é resultado da operação, para o qual a pós condição é válida. Substituindo a pré e pós condições da operação *IRFUN* na proposição acima chegamos ao Teorema abaixo.

Teorema (Implementabilidade de IRFUN):

$$\forall\ bir \in \mathbb{R},\ cf \in Cfun,\ tabir \in \text{seq of } Dir,\ deddep \in \mathbb{R},$$
$$func \in \text{map } Cfun \text{ to } Dfun \cdot$$
$$(\text{len}(tabir) \geq 1 \wedge$$
$$teto(tabir(\text{len}(tabir))) \geq bir \wedge$$
$$cf \in \text{dom } func) \Rightarrow$$
$$(\exists\ ir \in \mathbb{R} \cdot$$
$$\exists\ i \in \text{dom } tabir \cdot$$
$$\text{let } t = tabir(i)$$
$$\text{in } teto(t) \geq bir \wedge$$
$$(\forall\ j \in \{1, \ldots, i-1\} \cdot teto(tabir(j)) < bir) \wedge$$
$$ir = (bir * aliqt(t)) - ded(t) - (dep(func(cf)) * deddep))$$

Na prova, são utilizadas algumas propriedades das funções que implementam as operações sobre os objetos matemáticos de VDM. A seguir enunciamos as proposições e as provas completas do Teorema e Lemas podem ser encontradas em [3].

Nas proposições abaixo, tomando T como um tipo qualquer, assumimos

$$l : T\ List,$$
$$a, b, c, e : T,$$
$$x, y : T\ set,$$
$$cond : T \rightarrow Bool,$$
$$eq : T \rightarrow (T \rightarrow Bool),$$

eq é um operador de igualdade reflexivo, simétrico e transitivo.

O operador de igualdade para conjuntos (*set*) é *setequal*. O conjunto vazio é representado por *set(l,eq)*.

Lema 1: *Para todo x, eq,*

setequal(x,set(l,eq)) = false →

member(choiceset(x),x) = true.

Lema 2: *Para todo a, l, eq, member(a,set(l,eq)) = in(a,l,eq),*

onde fun in(a,[],eq) = false ;

in(a,h::t,eq) = eq(a,h) orelse in(a,t,eq)

Lema 3: *Para todo e, x, y,*

setequal(x,y) = true → member(e,x) = member(e,y).

Lema 4: *Para todo a, b, c, eq, x, cond,*

(eq(a,b) = true → cond a = cond b) →

member(c,setcomp(x,cond)) = member(c,x) andalso (cond c).

Para verificarmos que a função *IRSML* (em SML) satisfaz a função *IRFUN* (em VDM) temos que provar a seguinte proposição:

∀ *bir* ∈ ℝ, *cf* ∈ *Cfun*, *tabir* ∈ seq of *Dir*, *deddep* ∈ ℝ,
func ∈ map *Cfun* to *Dfun* •
pre-*IRFUN*(*bir*,*cf*,*tabir*,*func*,*deddep*) →
(*IRSML*(*bir*,*cf*,*tabir*,*func*,*deddep*) ∈ ℝ ∧
post-*IRFUN*(*bir*,*cf*,*tabir*,*func*,*deddep*,
IRSML(*bir*,*cf*,*tabir*,*func*,*deddep*)).

Substituindo a pré e pós condições de *IRFUN*, temos o Teorema abaixo.

Teorema (Satisfação da Especificação):

Para quaisquer *bir* ∈ ℝ, *cf* ∈ *Cfun*, *tabir* ∈ seq of *Dir*, *deddep* ∈ ℝ, *func* ∈ map *Cfun* to *Dfun*:

(len(*tabir*) ≥ 1 ∧ *teto*(*tabir*(len(*tabir*))) ≥ *bir* ∧
cf ∈ dom *func*) →
(*IRSML*(*bir*,*cf*,*tabir*,*func*,*deddep*) ∈ ℝ ∧
(∃ *i* ∈ dom *tabir* • let *t* = *tabir*(*i*)
in *teto*(*t*) ≥ *bir* ∧
(∀ *j* ∈ {1,...,*i*-1) • *teto*(*tabir*(*j*)) < *bir*) ∧
IRSML(*bir*,*cf*,*tabir*,*func*,*deddep*)
= (*bir* * *aliqt*(*t*)) - *ded*(*t*) - (*dep*(*func*(*cf*)) * *deddep*)))

Prova.

Assumimos que o antecedente é verdadeiro e provamos que o consequente também é. Logo, pela Teorema da Implementabilidade de *IRFUN*, para algum $i \in \text{dom } tabir$ e $ir \in \mathbb{R}$, onde $t = tabir(i)$ temos:

$$i \in \text{dom } tabir \wedge$$
$$(teto(t) \geq bir \wedge (\forall j \in \{1,\ldots,i-1\} \cdot teto(tabir(j)) < bir) \wedge$$
$$ir = (bir * aliqt(t)) - ded(t) - (dep(func(cf)) * deddep)).$$

Como $\vdash i \in X \wedge p(i) \Leftrightarrow i \in \{h \in X \mid p(h)\}$, temos:

$$(\xi)\ i \in \{h \in \text{dom } tabir \mid teto(tabir(h)) \geq bir \wedge$$
$$(\forall j \in \{1,\ldots,h-1\} \cdot teto(tabir(j)) < bir)\}.$$

Pela definição de IRSML, temos:

s = setcomp(domseq tabir,cond bir tabir). Observe que s, a menos de notação, é exatamente o conjunto em ξ. Logo, de ξ concluimos que: (Ω) member(i,s) = true, para algum $i \in \mathbb{N}$.

Por member, member(i,set([],eq)) = false, logo por Ω e pelo Lema 3,

setequal(s,set([],eq)) = false, e pelo Lema 1,

(Θ) member(choiceset(s),s) = true.

Como em IRSML id = choiceset(s), por Θ concluimos que

member(id,s) = true.

Logo, pelo Lema 4,

member(id,domseq tabir) = true e

cond bir tabir id = true.

Por cond, member(id,domseq tabir) = true,

(teto(aplseq(id,tabir)) >= bir) = true e

forall (setint(1,id-1))

(fn y => teto(aplseq(y,tabir)) < bir) = true.

Logo,considerando tt = $tabir(id)$, traduzindo de SML para VDM,

(θ) $id \in \text{dom } Tabir \wedge$

$teto(tt) \geq bir \wedge$

$(\forall j \in \{1,\ldots,id-1\} \cdot teto(tabir(j)) < bir)$.

De θ, pela reflexividade de =,

$id \in \text{dom } Tabir \wedge$
$teto(tt) \geq bir \wedge$
$(\forall j \in \{1,\ldots,id-1\} \cdot teto(tabir(j)) < bir) \wedge$
$(bir * aliqt(tt)) - ded(tt) - (dep(func(cf)) * deddep))$
$= (bir * aliqt(tt)) - ded(tt) - (dep(func(cf)) * deddep))).$

Por IRSML, como a condição do if é verdadeira (assumimos que pré condição de *IRFUN* é verdadeira),

$id \in \text{dom } Tabir \wedge$
$teto(tt) \geq bir \wedge$
$(\forall j \in \{1,\ldots,id-1\} \cdot teto(tabir(j)) < bir) \wedge$
$IRSML(bir, cf, tabir, func, deddep)$
$= (bir * aliqt(tt)) - ded(tt) - (dep(func(cf)) * deddep))).$

Logo, pela introdução do ∃, tomando *i* como id, onde *t* = *tabir*(*i*), temos:

$(\exists i \in \text{dom } tabir \cdot$
$teto(t) \geq bir \wedge$
$(\forall j \in \{1,\ldots,i-1\} \cdot teto(tabir(j)) < bir) \wedge$
$IRSML(bir, cf, tabir, func, deddep)$
$= (bir * aliqt(t)) - ded(t) - (dep(func(cf)) * deddep))).$

Da proposição acima podemos trivialmente concluir que:

$IRSML(bir, cf, tabir, func, deddep) \in \mathbb{R}$. Das duas proposições acima provamos o Teorema. ■

5. Conclusões

A implementação dos objetos matemáticos e operações de VDM realmente facilita a transformação das especificações em VDM para protótipos em SML.

Para especificações simples e não muito abstratas a passagem é direta. Para as especificações mais abstratas a transformação é também natural mas faz-se necessário provar que o protótipo satisfaz a especificação.

Nestes casos se a especificação é razoavelmente grande torna-se inviável conduzir as provas de maneira formal. Para tal, seria fundamental dispor de provadores e verificadores (semi) automáticos de teoremas com uma biblioteca de teoremas bem maior que a apresentada em [3].

Também seria interessante ter um meio (provado correto) de transformar classes de especificações em funções escritas em SML. Particularmente na especificação mostrada em [5] identificamos um conjunto de operações que podem ser implementadas em SML exatamente da mesma forma que a operação aqui apresentada. Neste caso não precisaríamos provar que elas implementam a especificação, já que isto foi provado para a operação *IR*.

Referências

[1] Jones, C.B.: Systematic Software Development Using VDM. Prentice Hall International (UK) Ltd, 1986.

[2] Harper, R., D. MacQueen and R. Milner: Standard ML. Edinburgh University, Internal Report ECS-LFCS-86-2, March, 1986.

[3] Borba, P.H.M., S.L. Meira: Modelo Funcional de um Subconjunto de VDM. Departamento de Informática, UFPE, 1989.

[4] Cohen, B., W.T. Harwood and M.I. Jackson: The Specification of Complex Systems. Addison-Wesley Publishing Company, 1986.

[5] Borba, P.H.M.: Especificação Formal em VDM de um Sistema de Folha de Pagamento. Departamento de Informática, UFPE, 1989.