PUC RIO - DEPARTAMENTO DE INFORMÁTICA

Monografias em Ciência da Computação

Editor: Carlos J. P. Lucena

Dezembro, 97

ESPECIFICAÇÃO DE ESTRUTURAS CONCEITUAIS BÁSICAS ATRAVÉS DA LÓGICA DE PREDICADOS

Eliana Silva de Almeida
Bruno Maffeo

Departamento de Informática

SUMÁRIO

ESPECIFICAÇÃO DE ESTRUTURAS CONCEITUAIS BÁSICAS ATRAVÉS DA LÓGICA DE PREDICADOS

Eliana Silva de Almeida
DI/PUC - Rio
e-mail: eliana@inf.puc-rio.br
Bruno Maffeo
DI/PUC - Rio
e-mail: maffeo@inf.puc-rio.br

PUC-Rio.Inf.MCC 44/97

Abstract:
This work defines a representation language and illustrates procedures to formally specify each primitive operational activity contained in activity schemes that model systems behavior when the Essential Model is used as conceptual framework. The representation language is based on predicate logic and, for each procedure illustrated, the primitive operational activity to be specified belongs to a conceptual pattern typically found in essence modeling. It is suggested that these procedures should be employed in formal specifications of large systems based on logic languages, since its use in connection with the structures commonly found in the Essential Model allows a gradual control of representational complexity, otherwise excessive. It is also believed that significant gains of quality and productivity in the systems development process should be achieved through reuse of these conceptual pattern specifications.

Keywords :
Declarative Specification, Predicate Logic, Essential Model, Conceptual Framework, Conceptual Pattern, Activity Scheme, Primitive Operational Activity, Reuse.

Resumo:
Neste trabalho, é definida uma linguagem de representação e são ilustrados procedimentos para especificar formalmente cada atividade operacional primitiva, contida em esquemas de atividades que modelam o comportamento de sistemas quando o arcabouço conceitual (“conceptual framework”) baseia-se no Modelo da Essência. A linguagem de representação (LLED — Linguagem Lógica para Especificação Declarativa) baseia-se na lógica de predicados e, em cada procedimento ilustrado, a atividade operacional primitiva a ser especificada faz parte de uma estrutura conceitual básica (“conceptual pattern”) encontrada tipicamente na modelagem da essência. É sugerido que esses procedimentos sejam utilizados em especificações formais de sistemas de grande porte, expressas através de linguagens baseadas em lógica, pois seu emprego, relacionado a esquemas de atividades obtidos na modelagem da essência, permite graduar a complexidade da representação a ser construída, em geral excessiva. Acredita-se também que ganhos significativos de produtividade e qualidade no processo de desenvolvimento de sistemas podem ser obtidos através do reúso das especificações formais associadas a essas estruturas conceituais básicas.

Palavras Chaves:
Especificação Declarativa, Lógica de Predicados, Modelo da Essência, Arcabouço Conceitual, Estrutura Conceitual Básica, Esquema de Atividades, Atividade Operacional Primitiva, Reúso.

1. INTRODUÇÃO

Durante as últimas duas décadas, a assim chamada área de Métodos Estruturados vem trazendo, para as etapas de Análise e de Design do processo de desenvolvimento de sistemas de software, um conjunto poderoso e diversificado de ferramentas conceituais e técnicas de modelagem. Desde o advento da Análise Estruturada [1, 2], passando pela Análise Essencial [3] e chegando a análise Estruturada Moderna [4, 5, 6, 7], diferentes autores têm investido considerável esforço no sentido de prover instrumentos de modelagem adequados ao desenvolvimento tanto de sistemas de informação convencionais de grande porte, quanto de sistemas de tempo-real para controle de processos [8, 9, 10].

Na Análise Estruturada Moderna, a especificação conceitual do comportamento de um sistema emprega linguagens de representação gráficas para construir **esquemas de atividades** (originalmente conhecidos como **diagramas de fluxos de dados**) que modelam os elementos funcionais ativos do sistema e as interfaces desses elementos, internas ao sistema e com o ambiente externo. Em seu formato hierarquizado, esses esquemas evidenciam uma decomposição funcional, correspondente a um refinamento descendente ("top-down") da visão do sistema ativo. Nessa abordagem, esse refinamento termina em atividades-folha denominadas **atividades operacionais primitivas**. Por definição, essas atividades não são decompostas funcionalmente em termos de esquemas de atividades; sua especificação detalhada, necessariamente, deve ser realizada em termos de outra linguagem de representação, desejavelmente formal.

O presente trabalho adota o mesmo objetivo definido em [14], ou seja, apresentar um tratamento formal para a **especificação declarativa** de atividades operacionais primitivas (originalmente conhecida como **mini-especificação**) contidas em um esquema de atividades organizado hierarquicamente resultante da modelagem do comportamento de um sistema de software a ser construído. Na prática, cada especificação declarativa desse tipo refere-se a uma **estrutura conceitual básica** (**conceptual pattern**) comumente encontrada na modelagem de sistemas com base no **Modelo da Essência** [7] — esta abordagem constituindo instância específica, freqüentemente utilizada, de **arcabouço conceitual** (**conceptual framework**) para especificação de sistemas —.

A distinção básica relativamente à abordagem adotada em [14], onde são utilizadas Redes de Petri Compactas [11, 12, 13], consiste em que, neste trabalho, essa especificação declarativa é realizada em termos de uma ferramenta conceitual estritamente textual baseada na lógica de predicados: a **L**inguagem **L**ógica para **E**specificação **D**eclarativa (LLED). Procura-se demonstrar a viabilidade dessa linguagem através de exemplos em que especificações semi-formais baseadas em listas de Pré/Pós-Condições são convertidas para a LLED.

Na seção 2, serão definidas a sintaxe e a semântica da LLED. Na seção 3, a utilização da LLED é ilustrada em exemplos específicos e na ultima seção, são apresentadas as conclusões deste trabalho.

2. UMA LINGUAGEM LÓGICA PARA ESPECIFICAÇÃO DECLARATIVA DE ESTRUTURAS CONCEITUAIS BÁSICAS

Nesta seção, é definida uma linguagem baseada na lógica de predicados, designada por LLED — **L**inguagem **L**ógica de **E**specificação **D**eclarativa —, para especificar **estruturas conceituais básicas** (**conceptual patterns**) contendo atividades operacionais primitivas de um esquema de atividades do Modelo da Essência. Cada estrutura desse tipo, evidenciando uma atividade operacional primitiva e sua interface (com o ambiente externo e com os demais elementos funcionais ativos do sistema), constitui uma construção gráfica que representa uma das possíveis formas sob as quais uma atividade operacional primitiva pode apresentar-se em esquemas de atividades construídos para especificar requisitos essenciais de um sistema de software.

O objetivo dessa linguagem é dar suporte a uma especificação, que visa definir relações, envolvendo as entradas e saídas de uma atividade operacional primitiva, a serem respeitadas por qualquer procedimento que constitua uma implementação da atividade. Essas relações serão representadas por fórmulas da linguagem. Associadas às fórmulas, encontram-se os elementos funcionais passivos que constituem a interface sobre a qual a atividade está atuando. Esses elementos são representados agregadamente no esquema de atividades pelos fluxos de dados e pelos depósitos de dados. Assim sendo, cada valor transportado por um fluxo de dados ou armazenado em um depósito de dados corresponde a um valor de um domínio, ou seja, de um conjunto englobando todos os valores que podem ser assumidos por um componente (atributo) específico do fluxo de dados ou do depósito de dados.

Ao definir a LLED, focalizou-se inicialmente a escolha dos elementos que se deseja modelar. Como o objetivo é especificar declarativamente cada atividade operacional primitiva de um esquema de atividades organizado hierarquicamente, esses elementos são os componentes (atributos) de fluxos/depósitos de dados. A figura abaixo [figura 1] ilustra uma estrutura conceitual básica incorporando uma instância genérica para esse tipo de atividade, apresentando elementos funcionais passivos (fluxos de dados e depósito de dados) com os quais uma atividade operacional primitiva pode interagir:

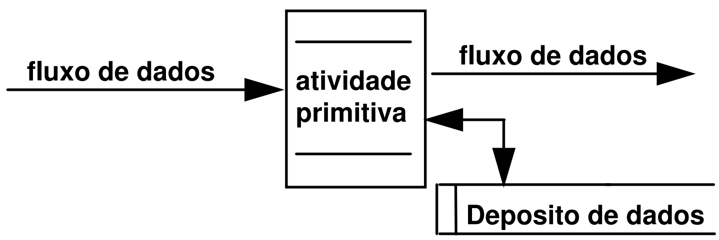

figura 1 - Atividade Operacional Primitiva e sua Interface

Para realizar essa especificação, necessita-se:

- ♦ definir os domínios, a serem considerados na LLED, associados aos componentes (atributos) dos fluxos/depósitos de dados que constituem a interface da atividade operacional primitiva a ser especificada;

- estabelecer as formas de referenciamento aos atributos cujos valores são manipulados por essa atividade operacional primitiva;
- selecionar os elementos básicos da lógica de predicados que permitirão construir as relações matemáticas que constituirão a especificação declarativa desejada;
- estabelecer as formas de associação válidas para os elementos que constituem o léxico da LLED.

Assim sendo, deve-se definir a sintaxe da LLED cuja forma de apresentação obedece a seguinte convenção:

- nomes entre < >, são símbolos não terminais, designando entidades que se deseja especificar;
- símbolos escritos em negrito e entre aspas são símbolos terminais da linguagem que está sendo definida;
- o símbolo "|" indica uma alternativa de especificação.

2.1. Sintaxe da LLED

<ESPECIFICAÇÃO>	::=	"**ESPECIFICAÇÃO:** "<COMENTÁRIO> "**;**"
		<INTERFACE> "**;**"
		<ATIVIDADE>
	|	"**ESPECIFICAÇÃO:**"
		<INTERFACE> "**;**"
		<ATIVIDADE>
<COMENTÁRIO>	::=	"texto escrito entre aspas"
<INTERFACE>	::=	"**Domínios:**" <DOMÍNIOS>"**;**"
		"**Fluxos:**" <FLUXOS>"**;**"
		"**Depósitos:**" <DEPÓSITOS>"**;**"
	|	"**Domínios:**" <DOMÍNIOS>"**;**"
		"**Fluxos:**" <FLUXOS> "**;**"
<ATIVIDADE>	::=	"**Atividade:**" <FÓRMULA>
<DOMÍNIOS>	::=	<CONJ-VAR> "**:**" <NOME-DOM> "**=**" <CONJUNTO>
		<COMENTÁRIO>
	|	<CONJ-VAR> "**:**" <NOME-DOM> "**=**" <CONJUNTO>
	|	<DOMÍNIOS>$_1$ "**;**" <DOMÍNIOS>$_2$
<CONJUNTO>	::=	<NOME-CONJ>
	|	<NOME-DOM>
	|	<CONJUNTO>$_1$ "**x**" <CONJUNTO>$_2$
	|	<CONJUNTO>$_1$ "$\rightarrow$" <CONJUNTO>$_2$
<NOME-DOM>	::=	"palavra escrita com letras maiúsculas"
<FLUXOS>	::=	<NOME-FLUXO>"("<NOME-DOM>")"
		<COMENTÁRIO>
	|	<NOME-FLUXO>"("<NOME-DOM>")"
	|	<FLUXOS>$_1$ "**;**" <FLUXOS>$_2$

<NOME-FLUXO> ::= "palavra iniciada por **F_** e complementada pelo descritor de um fluxo de dados"

<CONJ-VAR> ::= <VAR>
| <CONJ-VAR>$_1$"**,**" <CONJ-VAR>$_2$

<VAR> ::= "palavra contendo uma ou mais letras minúsculas"

<NOME-CONJ> ::= "palavra iniciada por **C_** e complementada por um nome iniciado por letra maiúscula"

<DEPÓSITOS> ::= <FUNÇÃO-DEP> "**:**" <NOME> "**=**" <NOME-DOM>$_1$ "→" <NOME-DOM>$_2$
| <DEPÓSITOS>$_1$ "**;**" <DEPÓSITOS>$_2$

<FUNÇÃO-DEP> ::= "letra grega minúscula"

<NOME> ::= "palavra iniciada por **D_** e complementada pelo descritor de um depósito de dados"

<FÓRMULA> ::= "**T**"
| "**F**"
| "∀"<VAR>"("<FÓRMULA>")"
| "∃"<VAR>"("<FÓRMULA>")"
| "¬" <FÓRMULA>
| <FÓRMULA>$_1$ <CONEC> <FÓRMULA>$_2$
| <PRED>"("<TERMO>$_1$, ... , <TERMO>$_n$")"
| <TERMO>$_1$ <PRED-BINÁRIO> <TERMO>$_2$

<CONEC> ::= "∧"
| "∨"
| "→"

<TERMO> ::= <CONSTANTE>
| <VAR>
| <FUNÇÃO>"("<TERMO>$_1$, ... , <TERMO>$_n$")"
| "⟨"<TERMO>$_1$, ... , <TERMO>$_n$"⟩"
| <TERMO>$_1$ <FUNÇÃO-BINÁRIA> <TERMO>$_2$
| <FUNÇÃO-DEP>

<FUNÇÃO> ::= <FUNÇÃO-DEP>
| "**PROJ**"
| "outra definida de acordo com a atividade a ser especificada"

<FUNÇÃO-BINÁRIA>::= "definidas de acordo com a atividade a ser especificada"

<PRED> ::= "**Incluído**"
| "**Excluído**"
| "**Alterado**"
| "**Acesso-leitura**"
| "**Acesso-escrita**"
| "outro definido de acordo com a atividade a ser especificada"

<PRED-BINÁRIO> ::= "definidos de acordo com a atividade a ser especificada"

<CONSTANTE> ::= "**T**"
| "**F**"
| "número"
| "nome escrito entre aspas"

<FUNÇÃO> e <FUNÇÃO-BINÁRIA>, definidas de acordo com a atividade a ser especificada, são representadas por palavras escritas em letras maiúsculas ou podem também ser símbolos especiais tais como + e = . Para distinguir visualmente de <FUNÇÃO> e <FUNÇÃO-BINÁRIA>, <PRED> e <PRED-BINÁRIO> são representados por palavras escritas em letras minúsculas e iniciadas por letra maiúscula.

É importante notar que a sintaxe só estará completa quando todos os seus elementos estiverem definidos, o que depende do particular conjunto de atividades operacionais primitivas que se deseja especificar.

2.2. Semântica da LLED

Para apresentar o significado dos elementos sintáticos da linguagem, são identificados os elementos funcionais passivos que definem a interface da atividade operacional primitiva que se deseja especificar. Como já foi observado, esses elementos são (componentes de) fluxos de dados e depósitos de dados. Assim sendo, a semântica dos elementos sintáticos da linguagem é a seguinte:

1. <ESPECIFICAÇÃO> : designa a atividade operacional primitiva do Esquema de Atividades a ser especificado. Seu conteúdo é a declaração dos elementos funcionais passivos apresentada em <INTERFACE>, que define os elementos da interface da atividade, e a fórmula apresentada em <ATIVIDADE>, que define as relações entre os elementos dessa interface.

2. <INTERFACE> : designa os elementos funcionais passivos que definem a interface da atividade operacional primitiva a ser especificada. Seu conteúdo é a declaração dos elementos de dados que são manipulados, apresentada em <DOMÍNIOS>, a declaração dos fluxos de dados, presente em <FLUXOS> e a declaração dos depósitos de dados, presente em <DEPÓSITOS>. Caso a atividade modelada não faça acesso a depósito de dados, esta parte da declaração é excluída da especificação.

3. <DOMÍNIOS> : designa os conjuntos de valores (domínios) manipulados pela atividade operacional primitiva a ser especificada. Uma variável designada por <VAR> pode assumir cada valor do domínio cujo nome é designado por <NOME-DOM>. O conjunto de valores do domínio é designado por <CONJUNTO>. Logo, em <DOMÍNIOS> encontra-se a declaração de todos os conjuntos de valores processados pela atividade operacional primitiva considerada.

4. <VAR> : designa algum valor de domínio. Uma variável deve estar associada a um conjunto designado por <NOME-DOM>.

5. <NOME-DOM> : designa informalmente o conjunto de valores associados a um domínio.

6. <CONJUNTO> : designa formalmente o conjunto de valores identificado informalmente por <NOME-DOM>. Seu identificador inicia-se por "C_" ou pode ser um nome já designado em <NOME-DOM>. Como "formadores" de conjuntos, apresentam-se os símbolos x e $\rightarrow$, onde:

- <CONJUNTO>$_1$ **x** <CONJUNTO>$_2$ representa o conjunto de todos os pares que podem ser formados tomando qualquer elemento do conjunto definido por <CONJUNTO>$_1$ e um elemento qualquer do conjunto definido por <CONJUNTO>$_2$
- <CONJUNTO>$_1$ $\rightarrow$ <CONJUNTO>$_2$ representa o conjunto de pares que definem uma função cujo domínio é <CONJUNTO>$_1$ e a imagem é <CONJUNTO>$_2$.

7. <FLUXOS> : designa predicados que permitem modelar a disponibilidade de valores associados a fluxos de dados. Um nome desse tipo de predicado, designado por <NOME-FLUXO>, inicia-se por "**F_**" e é complementado pelo descritor do fluxo. A cada um desses predicados associa-se nome designado por <NOME-DOM> para indicar o domínio em que o predicado pode aplicar-se.

8. <DEPÓSITOS> : designa os depósitos de dados acessados pela atividade operacional primitiva a ser especificada. Um depósito é definido como uma função, cujo nome é designado por <FUNÇÃO-DEP> e é informalmente identificado por <NOME>. O domínio da função é designado por <NOME-DOM>$_1$ e a imagem por <NOME-DOM>$_2$. Em geral, <NOME-DOM>$_1$ corresponde a um índice pertencente à classe de números naturais. <NOME-DOM>$_2$ corresponde a um domínio declarado em <DOMÍNIOS>.

9. <ATIVIDADE> : designa a operação da atividade operacional primitiva a ser especificada. Essa operação, cujo resultado é o estabelecimento de relações matemáticas entre elementos da entrada e elementos da saída da atividade, fica especificada declarativamente através de uma fórmula, representada por <FÓRMULA>, que designa os valores lógicos *verdadeiro* e *falso*.

10. T e F são constantes que designam os valores lógicos *verdadeiro* e *falso*, respectivamente.

11. $\neg$ <FÓRMULA> e <FÓRMULA>$_1$ <CONEC> <FÓRMULA>$_2$, onde <CONEC> representa um dos conectivos lógicos $\wedge$, $\vee$ e $\rightarrow$, designam as relações lógicas usuais. Da mesma forma, os símbolos $\forall$ e $\exists$ representam os quantificadores usuais, universal e existencial respectivamente, da lógica de predicados.

12. <PRED> (<TERMO>$_1$, ... , <TERMO>$_n$) e <TERMO>$_1$ <PRED-BINÁRIO> <TERMO>$_2$ representam relações entre os elementos designados por <TERMO>$_i$, onde i = 1,...,n. Alguns predicados já estão predefinidos nesta linguagem:

- **Incluído** (<FUNÇÃO-DEP>; <TERMO>) : representa a inclusão do valor designado por <TERMO> no depósito designado por <FUNÇÃO-DEP>.

- **Excluído** (<FUNÇÃO-DEP>; <TERMO>) : representa a exclusão do valor designado por <TERMO> do depósito designado por <FUNÇÃO-DEP>.

- **Alterado** (<FUNÇÃO-DEP>; <TERMO>$_1$; <TERMO>$_2$) : representa a alteração do valor designado por <TERMO>$_1$ pelo valor designado por <TERMO>$_2$ no depósito designado por <FUNÇÃO-DEP>.

- **Acesso-leitura** (<FUNÇÃO-DEP>) : representa um acesso de leitura a um depósito designado por <FUNÇÃO-DEP>.

- **Acesso-escrita** (<FUNÇÃO-DEP>) : representa um acesso de escrita a um depósito designado por <FUNÇÃO-DEP>.

13. <TERMO> : os termos podem ser:

- <CONSTANTE> : designa valor fixo de um domínio declarado em <DOMÍNIOS>.

- <VAR> : designa valor variável de um domínio declarado em <DOMÍNIOS> .

- <FUNÇÃO> (<TERMO>$_1$, ... , <TERMO>$_n$) e <TERMO>$_1$ <FUNÇÃO-BINÁRIA> <TERMO>$_2$: representam funções envolvendo valores designados por <TERMO>$_i$ (i=1, ..., n). Na primeira forma de representação, o primeiro termo designa o valor associado, através da função, aos valores designados pelos demais termos. A função predefinida PROJ$_i$ (n-upla) designa o i-ésimo valor da n-upla.
- <TERMO>$_1$, ... , <TERMO>$_n$⟩ : representa uma n-upla de valores, onde cada <TERMO>$_i$ (i = 1, ..., n) designa valor que pertence a um domínio declarado em <DOMÍNIOS>.

14. <COMENTÁRIO> : designa um texto escrito entre aspas, correspondendo a um comentário. Sua presença na especificação não é obrigatória.

3. EXEMPLOS DE ESPECIFICAÇÃO DECLARATIVA TEXTUAL UTILIZANDO A LLED

Nesta seção, a especificação formal de algumas estruturas conceituais básicas é ilustrada através de exemplos de como utilizar a LLED para especificar declarativamente atividades operacionais primitivas contidas no esquema de atividades organizado hierarquicamente de um Modelo da Essência. Para cada exemplo, é feita uma descrição sumária da operação da atividade e, em seguida, apresentada uma especificação declarativa utilizando a LLED. A título comparativo, é também apresentada a especificação declarativa da atividade baseada em Lista de Pré/Pós-Condições, instrumento textual de modelagem semi-formal freqüentemente empregado.

EXEMPLO 1 - CALCULA RAIZ QUADRADA

Dado um número maior que ou igual a zero, a atividade efetua o cálculo da raiz quadrada. Se o número for menor que zero, emite mensagem de rejeição [figura 2].

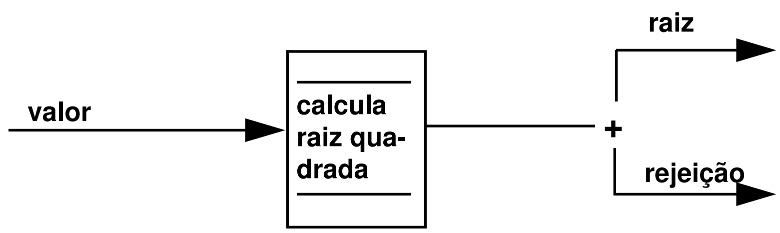

figura 2 - Esquema de Atividades para o exemplo
calcula raiz quadrada

pré-condição 1: valor ≥ 0
pós-condição 1: raiz TALQUE($raiz^2$ = valor) $\wedge \neg$rejeição

pré-condição 2: valor < 0
pós-condição 2: rejeição indicando que não existe raiz real $\wedge \neg$raiz

ESPECIFICAÇÃO:

Domínios:
x,y: NÚMERO = C_Real
z : MENSAGEM = C_Mensagem

Fluxos:

F_valor (NÚMERO)	"ocorre valor de entrada"
F_raiz (NÚMERO)	"ocorre valor de saída fornecendo a raiz quadrada"
F_rejeição (MENSAGEM)	"ocorre valor de saída fornecendo mensagem de rejeição"

Atividade:

$\forall x\ ((F\_valor(x) \wedge x \geq 0 \rightarrow \exists y(F\_raiz(y) \wedge y^2 = x\ ) \wedge \neg\exists z(F\_rejeição(z))$

$\vee$

$(F\_valor(x) \wedge x < 0 \rightarrow \exists z(F\_rejeição(z)) \wedge$ z indica inexistência de raiz real $\wedge$

$\neg \exists y(F\_raiz(y))))$

EXEMPLO 2 - EMITE HISTÓRICO ESCOLAR

Dado o número de matrícula do aluno, a atividade verifica se aluno consta no cadastro e se há disciplinas associadas a ele. Em caso afirmativo, obtém dados do aluno e emite seu histórico. Em caso negativo, emite uma mensagem de rejeição [figura 3].

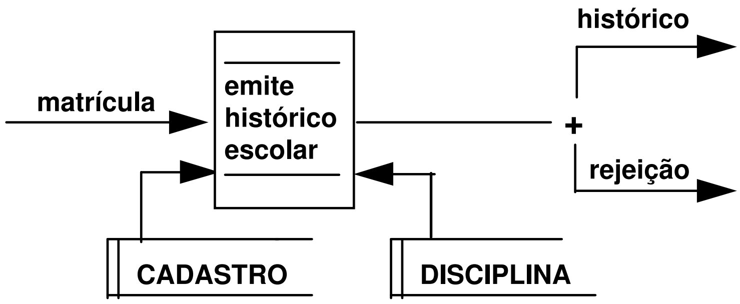

figura 3 - Esquema de Atividades para o exemplo
Emite Histórico Escolar

pré-condição 1: matrícula $\in$ CADASTRO $\wedge$ matrícula $\in$ DISCIPLINA

pós-condição 1: histórico TALQUE(valor histórico =
valor(CADASTRO(matrícula).nome +
DISCIPLINA(matrícula).{<codmat,nota>})) $\wedge$ $\neg$rejeição

pré-condição 2: matrícula $\notin$ CADASTRO

pós-condição 2: rejeição indicando que aluno não está cadastrado $\wedge$ $\neg$histórico

pré-condição 3: matrícula $\in$ CADASTRO $\wedge$ matrícula $\notin$ DISCIPLINA

pós-condição 3: rejeição indicando que o aluno não cursou disciplina alguma $\wedge$
$\neg$histórico

ESPECIFICAÇÃO:

Domínios:

x: MATRICULA	= C_MatrAl
y: NOME	= C_NomeAl
e: ENDEREÇO	= C_EndAl
n: NOTA	= C_Nota

c: CÓDIGO-DISCIPLINA = C_CodDis
r: MENSAGEM = C_Mensagem
a: AVALIAÇÃO = CÓDIGO-DISCIPLINA $\rightarrow$ NOTA
h: HISTÓRICO = MATRÍCULA x NOME x AVALIAÇÃO
i, j: ÍNDICE = C_Natural
cad: CADASTRO = MATRÍCULA x NOME x ENDEREÇO
dis: DISCIPLINA = MATRÍCULA x AVALIAÇÃO

Fluxos:

F_matrícula (MATRÍCULA)	"ocorre valor de entrada fornecendo número de matrícula"
F_rejeição (MENSAGEM)	"ocorre valor de saída fornecendo mensagem de rejeição"
F_histórico(HISTÓRICO)	"ocorre valor de saída fornecendo histórico escolar"

Depósitos:

σ : D_CADASTRO = ÍNDICE $\rightarrow$ CADASTRO
δ : D_DISCIPLINA = ÍNDICE $\rightarrow$ DISCIPLINA

Atividade:

$\forall x((\text{F\_matrícula}(x) \wedge \exists i \exists j \exists cad \exists dis((\sigma(i) = cad \wedge PROJ_1(cad) = x) \wedge (\delta(j) = dis \wedge PROJ_1(dis) = x)) \rightarrow \exists h(\text{F\_histórico}(h) \wedge h = <x, PROJ_2(cad), PROJ_2(dis)>) \wedge$
$\neg \exists z(\text{F\_rejeição}(z))$

$\vee$

$(\text{F\_matrícula}(x) \wedge (\neg \exists i(\sigma(i) = cad \wedge PROJ_1(cad) = x) \vee$
$\neg \exists j(\delta(j) = dis \wedge PROJ_1(dis) = x) \rightarrow \exists r(\text{F\_rejeição}(r) \wedge$ r indicando que aluno não está cadastrado ou que aluno não cursou disciplina $\wedge \neg \exists h(\text{F\_histórico}(h)))$

$\vee$

$(\text{F\_matrícula}(x) \wedge \exists i \exists j \exists cad \neg \exists dis((\sigma(i) = cad \wedge PROJ_1(cad) = x) \wedge (\delta(j) = dis \wedge PROJ_1(dis) = x)) \rightarrow \exists r(\text{F\_rejeição}(r) \wedge$ r indicando que aluno está cadastrado e não cursou disciplina alguma $\wedge \neg \exists h(\text{F\_histórico}(h)))$

EXEMPLO 3 - ALTERA NOTA DE ALUNO

Dado o número de matrícula do aluno, o código da disciplina e a nova nota, a atividade verifica se matrícula do aluno consta no arquivo de disciplinas. Em caso negativo, emite mensagem de rejeição. Em caso positivo, verifica se o código da disciplina está presente no registro correspondente ao aluno. Caso esteja, efetua a alteração da nota e emite mensagem de confirmação. Caso contrário, emite mensagem de rejeição [figura 4].

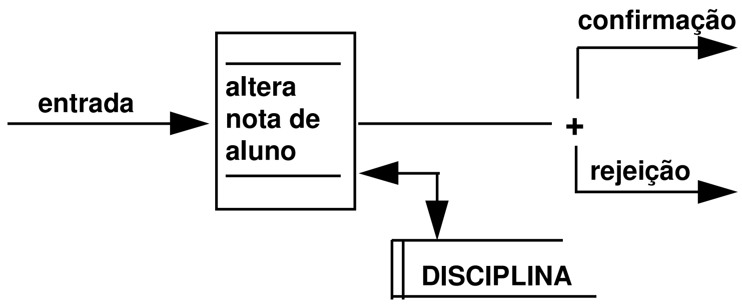

figura 4 - Esquema de Atividades para o exemplo *Altera Nota de Aluno*

pré-condição 1: aluno(matricula $\notin$ DISCIPLINA)
pós-condição 1: mensagem de rejeição indicando que a matrícula do aluno não consta no arquivo de disciplinas

pré-condição 2: aluno(matrícula $\in$ DISCIPLINA) $\wedge$ $\neg\exists$código da disciplina, associado à matrícula, em DISCIPLINA
pós-condição 2: mensagem de rejeição indicando que a disciplina não consta no arquivo de disciplinas

pré-condição 3: aluno(matrícula $\in$ DISCIPLINA) $\wedge$ $\exists$código da disciplina, associado a matrícula, em DISCIPLINA
pós-condição 3: altera nota em DISCIPLINA, do aluno cuja matrícula consta na "entrada", associada ao código que consta na "entrada" $\wedge$ mensagem de confirmação indicando que nota foi alterada

ESPECIFICAÇÃO:

Domínios:

x: MATRÍCULA = C_MatAl
n: NOTA = C_Nota
c: CÓDIGO-DISCIPLINA = C_CodDis
m: MENSAGEM = C_Mensagem
a: AVALIAÇÃO = CÓDIGO-DISCIPLINA $\rightarrow$ NOTA
i, j: ÍNDICE = C_Natural
e: DADOS-ENTRADA = MATRÍCULA x CÓDIGO-DISCIPLINA x NOTA
dis: DISCIPLINA = MATRÍCULA x AVALIAÇÃO

Fluxos:

F_entrada (DADOS-ENTRADA)	"ocorre valor de entrada fornecendo matrícula, código da disciplina e nota"
F_rejeição (MENSAGEM)	"ocorre valor de saída fornecendo mensagem de rejeição"
F_confirmação (MENSAGEM)	"ocorre valor de saída fornecendo mensagem de confirmação"

Depósitos:

δ : D_DISCIPLINA = ÍNDICE $\rightarrow$ DISCIPLINA

Atividade:

$\forall$e((F_entrada(e) $\wedge$ $\neg\exists$i(PROJ$_1$(δ (i)) = PROJ$_1$(e)) $\rightarrow$
$\exists$m(F_rejeição(m) $\wedge$ m indicando que a matrícula do aluno não consta do arquivo de disciplina) $\wedge$ $\neg\exists$z(F_confirmação(z)))

$\vee$

(F_entrada(e) $\wedge$ $\neg\exists$i(PROJ$_1$(δ (i)) = PROJ$_1$(e)) $\wedge$ (PROJ$_2$(δ (i)) = (PROJ$_2$(e) $\rightarrow$ PROJ$_3$(e))) $\rightarrow$
$\exists$m(F_rejeição(m) $\wedge$ m indicando disciplina não cursada) $\wedge$
$\neg\exists$z(F_confirmação(z)))

$\vee$

(F_entrada(e) $\wedge$ $\exists$i(PROJ$_1$(δ (i)) = PROJ$_1$(e)) $\wedge$ (PROJ$_2$(δ (i)) = (PROJ$_2$(e))$\rightarrow$
(Alterado(δ;PROJ$_2$(δ (i)) ; PROJ$_2$(e) $\rightarrow$ PROJ$_3$(e)) $\wedge$ $\exists$m(F_confirmação(m) $\wedge$ m indicando nota alterada) $\wedge$ $\neg\exists$z(F_rejeição(z)))

EXEMPLO 4 - SEPARA MENSAGEM

Dada uma mensagem com valores consistentes, a atividade identifica se trata-se de uma mensagem comum ou de uma mensagem especial e separa a respectiva mensagem. Em caso de valores inconsistentes, emite mensagem de rejeição [figura 5].

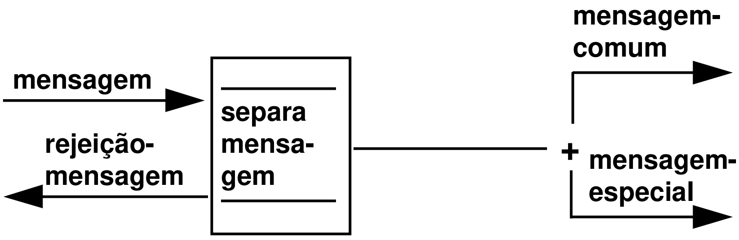

figura 5 - Esquema de Atividades para o exemplo *Separa Mensagem*

pré-condição 1: mensagem(∃ valor para texto ∧ valor para tipo indica mensagem - comum ∧ ¬∃ valor para classe)
pós-condição 1: mensagem-comum ∧ ¬rejeição-mensagem ∧ ¬mensagem-especial

pré-condição 2: mensagem(∃ valor para texto ∧ valor para tipo indica mensagem-especial ∧ ∃ valor para classe)
pós-condição 2: mensagem-especial ∧ ¬rejeição-mensagem ∧ ¬mensagem-comum

pré-condição 3: mensagem(¬∃ valor para (texto ∨ tipo) ∨ (∃ valor para texto ∧ valor para tipo indica mensagem especial ∧ ¬∃ valor para classe) ∨ (∃ valor para texto ∧ valor para tipo indica mensagem comum ∧ ∃ valor para classe))
pós-condição 3: rejeição-mensagem ∧ ¬mensagem-comum ∧ ¬mensagem-especial

ESPECIFICAÇÃO:

Domínios:

m: MENSAGEM	= C_Texto x C_Tipo x C_Classe
c: MENSAGEM-COMUM	= C_ Texto
x, z: MENSAGEM-REJEICAO	= C_Texto
y: MENSAGEM-ESPECIAL	= C_Texto x C_Classe

Fluxos:

F_mensagem (MENSAGEM)	"ocorre valor de entrada fornecendo mensagem"
F_rejeição-mensagem (MENSAGEM-REJEIÇÃO)	"ocorre valor de saída fornecendo mensagem de rejeição"
F_mensagem-comum (MENSAGEM-COMUM)	" ocorre valor de saída fornecendo mensagem comum"
F_mensagem-especial (MENSAGEM-ESPECIAL)	"ocorre valor de saída fornecendo mensagem especial"

Atividade:

$\forall m(F\_\,mensagem(m) \;\wedge\; PROJ1(m) \;\neq\; "\;" \;\wedge\; PROJ3(m) \;\neq\; "\;" \;\wedge$
$PROJ2(m)$ indicando mensagem especial$) \;\rightarrow\; \exists y(F_\,mensagem - especial(y)) \;\wedge$
$\neg\exists x(F\_\,rejeição - mensagem(x)) \;\wedge\; \neg\exists c(F\_\,mensagem - comum(c))$

$\vee$

$(F\_\,mensagem(m) \;\wedge\; ((PROJ1(m) \;=\; "\;" \;\vee\; PROJ2(m) \;=\; "\;") \;\vee$
$(PROJ1(m) \;\neq\; "\;" \;\wedge\; PROJ2(m)$ indicando mensagem especial $\wedge$
$PROJ3(m) \;=\; "\;") \;\vee\; (PROJ1(m) \;\neq\; "\;" \;\wedge\; PROJ2(m)$
indicando mensagem comum $\wedge\; PROJ3(m) \;\neq\; "\;") \;\rightarrow\; \exists x(F\_\,rejeição - mensagem(x))$
$\wedge\; \neg\exists c(F\_\,mensagem - comum(c)) \wedge\; \neg\exists y(F\_\,mensagem - especial(y)))$

EXEMPLO 5 - DECODIFICA MENSAGEM COMUM

Dado uma mensagem comum, a atividade decodifica esta mensagem. A aplicação do código ao texto da mensagem é especificada através da função DECOD-COMUM [figura 6].

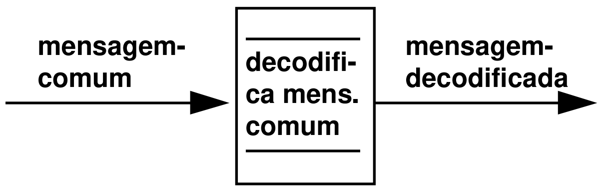

figura 6 - Esquema de Atividades para o exemplo
Decodifica Mensagem Comum

pré-condição 1: mensagem-comum
pós-condição 1: mensagem-decodificada (mensagem-decodificada.texto =
DECOD-COMUM (mensagem-comum.texto))

ESPECIFICAÇÃO:

Domínios:

x, y: MENSAGEM = C_Texto

Fluxos:

F_mensagem-comum (MENSAGEM) "ocorre mensagem comum"
F_mensagem-decodificada (MENSAGEM) "ocorre mensagem decodificada"

Atividade:

$\forall x(F\_mensagem\text{-}comum(x) \rightarrow \exists y(DECOD\text{-}COMUM(x) = y \wedge F\_mensagem\text{-}decodificada(y)))$

EXEMPLO 6 - DECODIFICA MENSAGEM ESPECIAL

Dado uma mensagem especial, a atividade verifica se existe no depósito DI 2 o código correspondente à classe da mensagem. Em caso positivo, decodifica o texto da mensagem especial, utilizando a função DECOD-ESPECIAL, sem realizar nenhum acesso ao depósito MENS.ESPECIAL. Em caso negativo, inclui a mensagem no depósito MENS.ESPECIAL [figura 7].

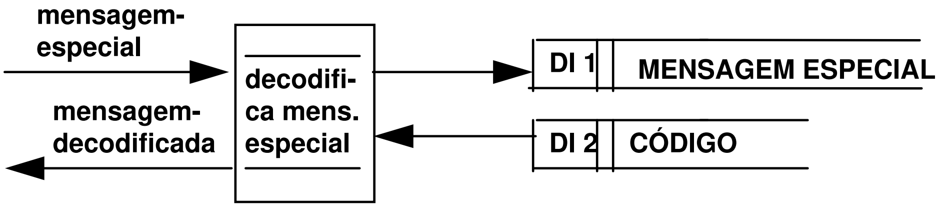

figura 7 - Esquema de Atividades para o exemplo
decodifica Mensagem Especial

pré-condição 1: mensagem-especial $\wedge$ $\exists$i(CÓDIGO(i). classe = mensagem-especial.classe)
pós-condição 1: mensagem-decodificada (mensagem-decodificada.texto = DECOD-ESPECIAL(i) (mensagem-especial.texto)) $\wedge$ $\neg$ acesso a DI 1

pré-condição 2: mensagem-especial $\wedge$ $\neg\exists$i(CÓDIGO(i). classe = mensagem-especial.classe)
pós-condição 2: $\neg$ mensagem-decodificada $\wedge$ $\exists$j(MENS.ESPECIAL(j) = mensagem-especial)

ESPECIFICAÇÃO:

Domínios:

m, c, x: MENSAGEM	= C_Texto x C_Classe
y: TEXTO	= C_Texto
i: ÍNDICE	= C_Natural

Fluxos:

F_mensagem-especial (MENSAGEM)	"ocorre mensagem especial"
F_mensagem-decodificada (TEXTO)	"ocorre mensagem decodificada"

Depósitos:

γ : D_MENS.ESPECIAL = ÍNDICE $\rightarrow$ MENSAGEM
η : D_CÓDIGO = ÍNDICE $\rightarrow$ CÓDIGO

Atividade:

$$\forall x((F\_mensagem\text{-}especial(x) \wedge \exists i PROJ_2(\eta(i)) = PROJ_2(x))$$
$$\rightarrow \exists y(F\_mensagem\text{-}decodificada(y) \wedge y = DECOD\text{-}ESPECIAL(PROJ_1(x))$$
$$\wedge \neg Acesso\text{-}leitura(\gamma)))$$
$$\vee$$
$$(F\_mensagem - especial(x) \wedge \neg\exists i\ (PROJ_2(\eta(i)) = PROJ_2(x)) \rightarrow$$
$$\neg\exists y(F\_Mensagem\text{-}decodificada(y)) \wedge Incluído(\gamma;\ x)))$$

EXEMPLO 7 - TRATA CÓDIGO

Dado código, com valores atribuídos a todos os seus componentes, a atividade verifica em DI 1 a existência de mensagens da mesma classe. Caso exista, decodifica todas aplicando ao texto de cada uma a função DECOD-ESPECIAL e gera a saída mensagem-decodificada, além de armazenar a entrada em DI 2 e excluir de DI 1 cada mensagem tratada. Caso não existam mensagens de uma mesma classe, é produzida apenas a resposta de armazenamento da entrada em DI 2. Haverá rejeição caso esteja faltando algum componente da entrada [figura 8].

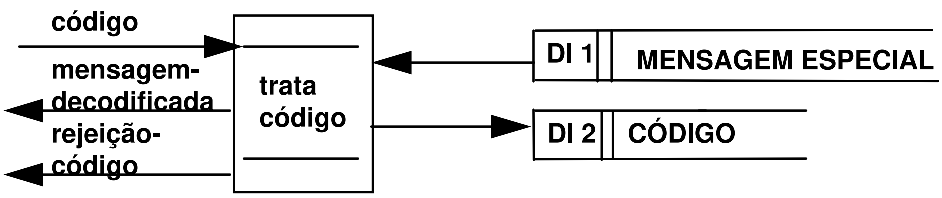

figura 8 - Esquema de Atividades para o exemplo *Trata Código*

pré-condição 1: código(∃(texto + classe)) ∧∃i(MENS.ESPECIAL(i).classe = código.classe)
pós-condição 1: mensagem-decodificada(mensagem-decodificada.texto = DECOD-ESPECIAL(mensagem-especial(i).texto)) ∧ ∃j(CÓDIGO(j) = código) ∧ ¬∃i(MENS.ESPECIAL(i).classe = código.classe) ∧ ¬rejeição-código

pré-condição 2: código(∃(texto + classe)) ∧ ¬∃i(MENS. ESPECIAL(i).classe= código.classe)
pós-condição 2: incluído(DI 2, código) ∧ ¬rejeição-código ∧ ¬mensagem-decodificada

pré-condição 3: código(¬∃(texto ∨ classe))
pós-condição 3: rejeição-código ∧ ¬mensagem-decodificada ∧ ¬acesso-escrita(CÓDIGO)

ESPECIFICAÇÃO:

Domínios:

e : MENSAGEM	= C_Texto x C_Classe
c : CÓDIGO	= C_Texto x C_Classe
x, y: TEXTO	= C_Texto
i,j : ÍNDICE	= C_Índice

Fluxos:

F_código (CÓDIGO)	"ocorre valor de entrada fornecendo código"
F_mensagem-decodificada (TEXTO)	"ocorre valor de saída fornecendo mensagem decodificada"
F_rejeição-código (TEXTO)	"ocorre valor de saída fornecendo rejeição de código"

Depósitos:

γ : D_MENSAGEM-ESPECIAL = ÍNDICE $\rightarrow$ MENSAGEM
η : D_CÓDIGO = ÍNDICE $\rightarrow$ CÓDIGO

Atividade:

$$
\begin{aligned}
&\forall c((\text{F\_Código}(c) \wedge \text{PROJ}_1(c) \neq \text{" "} \wedge \text{PROJ}_2(c) \neq \text{" "} \\
&\wedge \exists i(\text{PROJ}_2(\gamma(i)) = \text{PROJ}_2(c)) \rightarrow \forall j(\text{PROJ}_2(\gamma(j) = \text{PROJ}_2(c) \\
&\rightarrow \exists y(\text{F\_mensagem-decodificada}(y) \wedge y = \text{DECOD-ESPECIAL}(\text{PROJ}_1(\gamma(j)))) \\
&\wedge \text{Incluído}(\eta;c) \wedge \text{Excluído}(\gamma;\gamma(j)) \wedge \neg\exists x(\text{F\_rejeição-código}(x)) \\
&\qquad\qquad\qquad\qquad \vee \\
&(\text{F\_Código}(c) \wedge \text{PROJ}_1(c) \neq \text{" "} \wedge \text{PROJ}_2(c) \neq \text{" "} \\
&\wedge \neg\exists i(\text{PROJ}_2(\gamma(i)) = \text{PROJ}_2(c)) \rightarrow \text{Incluído}(\gamma;c) \wedge \\
&\neg\exists x(\text{F\_rejeição-código}(x)) \wedge \neg\exists y(\text{F\_mensagem-decodificada}(y))) \\
&\qquad\qquad\qquad\qquad \vee \\
&(\text{F\_Código}(c) \wedge (\text{PROJ}_1(c) = \text{" "} \vee \text{PROJ}_2(c) = \text{" "}) \rightarrow \\
&\exists x(\text{F\_rejeição-código}(x)) \wedge \neg\exists y(\text{F\_mensagem-decodificada}(y)) \\
&\wedge \neg\text{Acesso-escrita}(\eta))
\end{aligned}
$$

4. CONCLUSÃO

Neste trabalho, foi definida uma Linguagem Lógica para Especificação Declarativa (LLED), baseada na lógica de predicados, visando dar suporte à especificação de estruturas conceituais básicas, contendo as atividades operacionais primitivas do esquema de atividades hierarquizado, obtido na modelagem da essência de um sistema. O objetivo foi contribuir para a formalização desse tipo de especificação, oferecendo uma alternativa puramente textual à formalização proposta em [14], baseada em Redes de Petri Compactas.

A abordagem proposta, além de capturar toda a dinâmica inerente à execução de uma atividade operacional primitiva, formaliza apenas a relação matemática bem definida que deve existir entre a entrada e a saída da atividade especificada. Dessa forma, presta-se eficazmente à especificação declarativa dessa atividade pois não apresenta qualquer viés procedimental que indicaria como, numa dada alternativa de implementação, essa relação seria estabelecida.

Da mesma forma que no caso de uma especificação baseada em Redes de Petri Compactas [14], vale indicar que a presente abordagem permite:

- graduar a complexidade da representação associada a uma especificação formal de sistemas de grande porte totalmente baseada em textos formalizados através de linguagens da lógica.

  É bastante razoável supor que, utilizando unicamente Lógica de Predicados, dar-se-ia origem a uma representação de grande complexidade, difícil de validar e manter consistente. Por outro lado, esquemas de atividades, tais como os encontrados na modelagem da essência de sistemas, possuem estruturas conceituais básicas simples e intuitivas por se situarem em nível de abstração suficientemente elevado para tornar possível abstrair grande parte de sua complexidade formal. Assim sendo, de acordo com a abordagem aqui preconizada, o comportamento de um sistema de grande porte possuiria uma representação formal abstrata mais simples e intuitiva através desses esquemas, e o detalhamento formal que restaria a fazer ficaria circunscrito às atividades operacionais primitivas neles contidas e seria obtido empregando-se os procedimentos ilustrados, através dos exemplos contidos no presente trabalho;

- através do reúso das especificações formais associadas a essas estruturas conceituais básicas, aumentar a produtividade da etapa de especificação do sistema a ser construído, bem como aprimorar a qualidade dos produtos nela gerados.

  Sem dúvida, um dos aspectos mais complexos e consumidores de tempo, durante a modelagem da essência de um sistema, refere-se à sub-etapa de especificação detalhada das atividades operacionais primitivas que compõem a solução abstrata proposta para o problema a ser resolvido, através da construção do sistema. Essa é uma sub-etapa que, inclusive, proporciona a melhor oportunidade para detectar possíveis erros e omissões cometidos durante a fase de elaboração do enunciado desse problema. O reúso de especificações formais depuradas e refinadas ao longo tempo, associadas a estruturas conceituais básicas bem definidas e de ocorrência freqüente, acredita-se, contribuirá para aperfeiçoar, globalmente, a execução da etapa de especificação.

5. REFERÊNCIAS BIBLIOGRÁFICAS

[1] Gane, C. e Sarson T., *Structured Systems Analysis*. Prentice Hall, 1979.

[2] DeMarco, T., *Structured Essential Systems Analysis*. Prentice Hall, 1984.

[3] McMenamin, S. e Palmer J., *Structured Systems Analysis and System Specification*. Prentice Hall, 1979.

[4] Ward, P. T., *Systems Development Without Pain.* Yourdon Press, 1984.

[5] Ward, P. T. e Mellor S. J., *Structured Development of Real-Time Systems*. Yourdon Press, 1985.

[6] Yourdon, E., *Modern Structured Analysis*. Yourdon Press, 1989.

[7] Maffeo, B., *Engenharia de Software e Especificação de Sistemas*. Campus, 1992.

[8] Tao, Y. e Kung C., *Formal Definition and Verification of Data Flow Diagrams*. The Journal of Systems Software (16)1, 29 (1991).

[9] Richter, G. e Maffeo B., *Towards a Rigorous Interpretation of ESML - Extended Systems Modeling Language*. IEEE Transactions on Software Engineering (19)2, 165 (1993).

[10] Sanchez, M. L. e Maffeo B., *Modelagem Conceitual de Sistemas de Tempo-Real — Um Estudo de Caso* —. Monografias em Ciência da Computação do Departamento de Informática da PUC-Rio, 18/96.

[11] Heuser, Carlos A., *Modelagem Conceitual de Sistemas*. Edição Preliminar, V EBAI - Nova Friburgo, Brasil.

[12] Heuser, Carlos A. e Richter G., Constructs for Modeling Information Systems with Petri Nets. *Aplication and Theory of Petri Nets*. Springer-Verlag, 1992 (224 - 243).

[13] Heuser, Carlos A., Peres, Eduardo M. e Richter, G., Towards a complete conceptual model: Petri nets and entity-relationship diagrams. *Information Systems*, volume 18, número 5 (1993).

[14] de Almeida, Eliana S. e Maffeo, B., Especificação de Estruturas Conceituais Básicas através de Redes de Petri Compactas. Monografias em Ciência da Computação do Departamento de Informática da PUC-Rio, 26/97