UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

Uma metodologia de modelagem de sistemas computacionais baseada em gramáticas de grafos

por

EDUARDO PRETZ

Dissertação submetida à avaliação,
como requisito parcial, para a obtenção do grau de Mestre
em Ciência da Computação

Prof[a]. Dr[a]. Leila Ribeiro
Orientadora

Porto Alegre, novembro de 2000.

Agradecimentos

À minha orientadora, professora Leila Ribeiro,
pela paciência e compreensão durante o
desenvolvimento do trabalho.

Aos meus pais, cujo ideal de vida foi o sucesso
de seus filhos, sendo esse o seu único tesouro.

Ao meu amigo Jorge Barbosa por me ensinar
que a realidade não se limita aos nossos
sentidos.

E principalmente à minha esposa Débora,
por me proporcionar os melhores
momentos de cada dia.
Dia após dia.

Sumário

Lista de Abreviaturas

RV	um conjunto contável de rótulos de vértices
RA	um conjunto contável de rótulos de arestas
VA	um conjunto infinito contável de valores de atributos
V	é o conjunto de vértices
$W \subset (V \times RA \times V)$	o conjunto de arestas
λ	a função de rotulagem de vértices
π	a função de atribuição de vértices
κ	a função de atribuição de arestas
s	a função de definição do estado de vértices $P\text{-}TIPO$
τ	a função de tipagem de arestas
$SORT_{DT}$	conjunto finito de nomes de classes
$OPER_{DT}$	conjunto finito de nomes de operações
$PRED_{DT}$	conjuntos finitos de nomes de predicados

Lista de Figuras

Resumo

Vários métodos de especificação procuram realizar a modelagem de sistemas sob três visões: uma visão funcional, que procura apresentar as informações que trafegam entre os diversos componentes do sistema, uma visão de dados, que apresenta as relações entre as estruturas de dados estáticas do sistema e a visão dinâmica, que mostra as transformações que o sistema pode sofrer ao longo do tempo.

Alguns modelos procuram integrar mais de uma visão, mas, em geral, os modelos possuem sérias deficiências ao tentarem representar mais de um aspecto do sistema ao mesmo tempo, sendo necessário o apoio de outros métodos.

Este trabalho apresenta um método de especificação de sistemas que procura integrar a modelagem de dados com a modelagem funcional e dinâmica utilizando-se, para isso, das Gramáticas de Grafos como método formal de especificação.

Sendo um grafo formado por vértices, arestas e rótulos, pode-se facilmente criar uma camada de abstração em que o usuário (em geral responsável pela análise de sistemas) manipule um método de especificação com o qual já convive, agora com uma semântica formal definida.

Espera-se, com a aplicação do método, gerar modelos passíveis de prova, não ambíguos e que promovam um incremento de qualidade no sistema gerado.

Palavras-chave: Gramáticas de Grafos, Sistemas de Transformação de Grafos, Métodos Formais, Especificação Formal.

Abstract

Several specification methods try to realize system modeling following three visions: the functional vision, which is based on representing the information exchange among the several components of the system; the data vision, which represents the relations among the static data structures of the system; and the dynamic vision, which presents the transformations the system may endure over the time.

Some models exist that try to integrate more than one of these visions, but, in general, they suffer from deficiencies when trying to represent more than one aspect of the system at the same time, in which case the use of other methods is necessary.

This work presents a novel method of systems specification that attempts to integrate data modeling with functional and dynamic modelings using, for this, Graph Grammars as its formal specification method.

A graph, being made of nodes, edges and labels, is appropriate for creating, easily, an abstraction layer in which the user (usually responsible for the system analysis) manipulates a specification method which is known to him, but now with a well defined formal semantics.

We hope, by applying this method, to generate provable, unambiguous models which promote an increase in the quality of the generated system.

Keywords: Graph Grammars, Graph Transformation Systems, Formal Methods, Formal Specification.

1 Introdução

Um grande salto na evolução dos métodos de construção de sistemas computacionais ocorreu na década de 70 com a introdução de métodos semi-formais no desenvolvimento de sistemas. Ao contrário dos métodos formais que são baseados em uma semântica formal, os métodos semi-formais procuram forçar o uso de linguagem formal no nível sintático [RIB 97]. Esses métodos tinham o objetivo básico de organizar o conhecimento sobre o assunto e proporcionar aos profissionais de desenvolvimento de sistemas padrões que, se seguidos, aumentariam as chances de êxito na automação de sistemas.

Com o passar dos anos observou-se que as equipes de desenvolvimento aumentaram a sua produtividade e a qualidade de seus sistemas, embora isso possa ter ocorrido mais pela sistematização e documentação do processo de construção do que pela validade dos métodos semi-formais sugeridos.

A qualidade na construção de sistemas computacionais só pode ser garantida através de métodos formais, visto que provas formais necessitam que a semântica de uma especificação ou programa seja determinada por modelos matemáticos.

Vários são os métodos que dão suporte formal a especificação de sistemas. Em geral esses métodos procuram gerar especificações segundo as abordagens operacional, baseada na construção de máquinas abstratas, denotacional, onde os sistemas são construídos usando domínios ou axiomática, baseada em álgebra abstrata e lógica formal.

As gramáticas de grafos e sistemas de transformação de grafos tem sido estudados há cerca de 30 anos com várias aplicações na ciência da computação. Atualmente tem sido mais utilizada para especificação de ambientes de desenvolvimento de sistemas e de diferentes tipos de sistemas computacionais. Também podem ser utilizada para avaliação de funções algébricas e expressões lógicas, geração de padrões e projeto de sistemas baseados em regras e em diversas áreas envolvendo paralelismo e concorrência [NAG 91a].

Grafos provêm uma representação de dados expressiva e versátil de um sistema. Do ponto de vista da interação entre o usuário e o modelo do sistema pode-se observar que os grafos são intuitivos e, em geral, de fácil compreensão por parte dos usuários. A rescrita de grafos também possui a característica de proporcionar uma representação de alto nível da solução de um problema computacional [BLO 95].

Outra vantagem da modelagem de uma sistema através de rescrita de grafos é a possibilidade de simulação (com o auxílio de ferramentas) o que proporciona além de sua verificação formal a validação do sistema, ou seja, a avaliação de sua conformidade com os requisitos funcionais.

Também, as gramáticas de grafos proporcionam um grande incremento na qualidade e produtividade na fase de manutenção de um sistema, em função da total previsibilidade do comportamento de um sistema em caso de alteração nos seus requisitos (novas especificações). Esse fator é de grande importância se considerarmos que a manutenção representa cerca de 60% da vida útil de um sistema [PRE 95].

A utilização de gramáticas de grafos como método para a modelagem de sistemas, pela sua flexibilidade de representação, pode ser uma poderosa ferramenta de modelagem dos diversos aspectos a serem representados durante o desenvolvimento de sistemas. Munida de suporte que lhe possibilite gerenciar as diversas abstrações observadas durante a construção de sistemas, uma semântica que lhe possibilite modelar

todos os aspectos do sistema e documentação que sirva como guia para o usuário durante a construção de sistemas, as gramáticas de grafos e os métodos de rescrita de grafos se apresentam como uma boa alternativa para a unificação dos diversos modelos semânticos utilizados no processo de construção de sistemas. Desta forma o chamado "*gap* semântico" [YOU 91] existente entre os diversos modelos utilizados para a construção de sistemas tende a ser diminuído, sendo necessária a utilização de modelos com outra semântica apenas para representar aspectos diretamente relacionados à tecnologia utilizada para a implementação do sistema.

Semânticas semelhantes proporcionam uma fácil integração entre os diversos modelos e permitem que o incremento de informação que caracteriza a diminuição do nível de abstração não gere modelos claramente distintos [PRE 99].

O **objetivo principal** deste trabalho é criar um método para desenvolvimento de software que se utilize das gramáticas de grafos como um método formal que incremente a qualidade do sistema gerado.

Para isso será necessário gerar uma descrição baseada em grafos que permita a representação dos modelos de dados, funcional e de controle considerando *programming in the small* e *programming in the large*, ou seja, os diversos níveis de abstração existentes durante o processo de desenvolvimento de sistemas aplicativos.

Utilizou-se como base para a modelagem de dados um modelo Entidade Relacionamento Extendido, e para modelagem funcional um Diagrama de decomposição funcional baseado em fluxo de dados.

Basicamente, a modelagem proposta é dividida em dois níveis. No primeiro nível foi definida a semântica e a sintaxe do modelo de dados, de modo a determinar as regras necessárias à manutenção da integridade do modelo. No segundo nível, o analista que realizar a modelagem deverá se preocupar com as regras que determinarão o sistema de rescrita de grafos, de modo a apresentar todas as possíveis transformações sofridas a partir de um grafo estado.

Os sistemas de rescrita de grafos foram criados como uma forma simples de representar as possíveis transformações pelas quais um modelo pode passar. Um modelo íntegro é aquele em que sempre será possível aplicar uma regra, exceto se esta for o estado final do modelo.

1.1 Sistemas de rescrita de grafos

Os sistemas de rescrita de grafos se baseiam no processo de transformação que um grafo pode sofrer em função de um conjunto de regras previamente definidas.

Ele é utilizado para demonstrar transformações que um sistema sofre após a conclusão de cada transição de estado. Esse processo parte de um Grafo Tipo e de um Grafo Inicial.

Um Grafo Tipo descreve as possíveis ocorrências em um grafo. É uma forma simplificada mas bastante representativa que substitui um (possivelmente) grande conjunto de regras que controlariam o grafo.

Um Grafo Inicial apresenta a primeira instância do grafo, nela esta representado a estado inicial do sistema aguardando a aplicação de regras que o transformarão.

Segundo Ehrig Hartmut [HAR 79], um sistema de rescrita de grafos é formado por dois grafos que se relacionam através de um morfismo de grafos que mapeia vértices e arestas do primeiro grafo para vértices e arestas do segundo. Um morfismo de grafo g entre dois grafos G e H é denotado por $g : G \quad H$. Para isso devem coincidir os *tipos*, *estruturas* e *valores de atributos*.

O processo de transformação de grafos se baseia em dois grafos, chamados de lado esquerdo *R* e lado direito *L* e um morfismo de grafo $r : L \quad R$ entre os diversos componentes dos grafos.

A principal característica deste método é a forma como a transformação do grafo é representada. As operações básicas são as seguintes:

- Deleção : A partir dos dois lados, *L* e *R*, objetos que aparecem em *L* e não possuem mapeamento *r* em *R* são deletados;
- Adição : Objetos do grafo em *R* que não possuem relação com nenhum elemento em *L* são criados,
- Preservação : Objetos em *L* os quais são mapeados para objetos em R através de *r* são mantidos.

Transformação de grafos define uma manipulação de grafos baseado em regras. Este trabalho será baseado na abordagem algébrica *single-pushout* para gramáticas de grafos [ROZ 97]. Regras de grafos podem ser utilizadas para capturar os aspectos dinâmicos do sistema. A idéia que se tem de gramáticas de grafos (consistindo de um grafo inicial e de um conjunto de regras de grafos) é a generalização da gramática de Chomsky de strings para grafos.

1.2 Organização do trabalho

O trabalho encontra-se dividido em 5 Capítulos.

No Capítulo 2 é apresentado formalismo que descreve o modelo de dados, modelo ER Extendido (EER). Este formalismo foi proposto por Marc Andries em [AND 96].

O Capítulo 3 introduz a definição formal do modelo funcional, chamado EFD (Esquema de Fluxo de Dados) integrado com o modelo EER. Baseado em um diagrama de decomposição funcional, o modelo propõe uma representação simples com um formalismo de fácil compreensão, de modo que mesmo profissionais sem grande fundamentação matemática consigam realizar a especificação formal de sistemas. Para simplificar esta tarefa, foi descrito um conjunto de regras formais que garantem a integridade do modelo.

O Capítulo 4 propõe um conjunto de atividades a serem desenvolvidas para gerar um modelo que segue a abordagem proposta. Neste capítulo também são descritos os cuidados as serem observados quando for realizada a geração das regras que determinarão as possíveis transformações sofridas pelo grafo.

Um estudo de caso que segue os passos do método é modelado no Capítulo 5. Baseado em um sistema de gerenciamento de clínicas médicas, o estudo de caso propõe a solução em termos de geração de um sistema de rescrita de grafos,

Por fim o Capítulo 6 apresenta um comparativo entre um conjunto de metodologias de desenvolvimento de sistemas baseados em modelagem de dados, modelagem funcional e modelagem orientada a objetos, a partir de onde é feita uma comparação apresentando as vantagens da representação de um sistema usando gramáticas de grafos.

2 Definição formal do Modelo EER

Neste capítulo serão apresentadas as definições de modelo ER Extendido de Hohemstein apud [AND 96].

Inicialmente será apresentada a definição de Grafo. A notação matemática utilizada nas definições pode ser encontrada no Apêndice A.

2.1 Definição de Grafo

Seja *RV* um conjunto contável de rótulos de vértices. *RA* um conjunto contável de rótulos de arestas e *VA* um conjunto infinito contável de valores de atributos.

Um grafo sobre (*RV, RA, VA*) é uma quádrupla (V, W, λ, π), onde

- V é o conjunto de vértices,
- $W \subset (V \times RA \times V)$ o conjunto de arestas,
- $\lambda : V \quad RV$ a função de rotulagem de vértices e
- $\pi : V \quad VA$ a função de atribuição de vértices.

Um modelo EER consiste basicamente de dois níveis. O nível superior consiste de um formalismo de esquemas de modelagem de dados e instâncias. Abaixo existe um nível que permite a especificação de tipos arbitrários de dados a serem usados como domínios de atributos na especificação do esquema EER. A presença deste nível mais baixo permite a definição de tipos de dados que não estão normalmente disponíveis em sistemas de gerenciamento de bancos de dados relacionais. Uma coleção de tipos de dados estão declarados em uma assinatura de tipos de dados

2.2 Definição da Assinatura de tipo de dados

Uma assinatura de tipo de dado *DT* é uma sêxtupla

($SORT_{DT}$,$OPER_{DT}$, $PRED_{DT}$, source, dest, arg) onde

- $SORT_{DT}$, $OPER_{DT}$ e $PRED_{DT}$ são conjuntos finitos de nomes de classes, operações e predicados, respectivamente.
- source, dest e arg são funções com as seguintes assinaturas:

$$\begin{aligned} \text{source} &: OPER_{DT} \quad SORT^{*}_{DT} \\ \text{dest} &: OPER_{DT} \quad SORT_{DT} \\ \text{arg} &: PRED_{DT} \quad SORT^{+}_{DT} \end{aligned}$$

2.3 Notação

- Se ω é um nome de operação com

source(ω) = <D_1, ..., D_n> e dest (ω) = D, então escreve-se

ω : $D_1 \times ... \times D_n$ D, o qual é chamado de assinatura de operação de ω.

- Se π é um nome de predicado com

  arg(π) = <D1, ..., Dn>, então escreve-se

  π : D1 × ... × Dn, o qual é chamado de assinatura de predicado de π.

e um conjunto $SORT_{DT}$ contendo as classes de tipos int, string, numérico, data, hora, dinheiro e endereço. Então pode-se definir uma operação calcula preço de consulta por $OPER_{DT}$, com as funções source e dest como seguem:

source(calcula preço de consulta) = (dinheiro,int)

dest(calcula preço de consulta) = dinheiro

O conjunto $PRED_{DT}$ pode conter, entre outros, o nome de predicado "$\leq_{dinheiro}$" com a função arg definida da seguinte forma:

arg($\leq_{dinheiro}$) = (dinheiro,dinheiro)

Assim a operação calcula preço de consulta tem a seguinte assinatura:

calcula preço de consulta : dinheiro × int dinheiro

Enquanto "$\leq_{dinheiro}$" tem a assinatura de predicado:

$\leq_{dinheiro}$: dinheiro × dinheiro

Os conjuntos de valores são associados às classes de tipos de dados através de uma interpretação. Da mesma forma, uma interpretação associa uma função para cada operação a uma relação para cada predicado, respeitando as suas respectivas assinaturas.

2.4 Interpretação de uma Assinatura de Tipo de Dado

Uma interpretação de uma assinatura de tipo de dado DT é uma tri-tupla

μ[DT] = (μ[$SORT_{DT}$], μ[$OPER_{DT}$], μ[$PRED_{DT}$]) de funções onde:

- μ[$SORT_{DT}$] associa um conjunto de valores para cada nome de classe em $SORT_{DT}$;
- μ[$OPER_{DT}$] associa para cada nome de operação em $OPER$ com assinatura
  ω : $d_1 \times ... \times d_n$ d uma função
  μ[$OPER_{DT}$](ω) : μ[$SORT_{DT}$](d_1) × ... × μ[$SORT_{DT}$](d_n) μ[$SORT_{DT}$](d);
- μ[$PRED_{DT}$] associa para cada nome de predicado em $PRED$ com assinatura
  π : $d_1 \times ... \times d_n$ d uma relação tal que

$\mu[PRED_{DT}](\pi) \subseteq \mu[SORT_{DT}](d_1) \times ... \times \mu[SORT_{DT}](d_n)$.

Na interpretação da assinatura de tipo de dado do exemplo, a função $\mu[SORT_{DT}]$ pode associar ao nome da classe dinheiro o conjunto de todos os números racionais com mais de duas casas decimais. A função $\mu[PRED_{DT}]$ pode associar ao predicado "$\leq_{\text{dinheiro}}$" a relação binária contendo os pares (d, d') de valores de dinheiro no qual d é menor ou igual a d'.

2.5 Expressão SORT

Para dar suporte a definição de estruturas complexas para atributos (como listas, *bags* e conjuntos), foi definida a seguinte expressão *SORT*.

Seja S um conjunto de símbolos com mapeamento $\mu[S]$ o qual mapeia cada elemento de S para um conjunto finito. O conjunto expr(*SORT*) de expressões *SORT* sobre S é definido como segue:

$$\text{expr}(S) = \quad SORT \quad \{conjunto(s) \mid \forall s \in SORT\} \quad \{bag(s) \mid \forall s \in SORT\} \quad \{lista(s) \mid \forall s \in SORT\}$$

De modo que $\forall s \in \text{expr}(SORT)$

$$\mu[S]s = \begin{cases} \mu[s_0], s \in SORT \\ P\,(\mu[s_0]), s = conjunto(s_0) \\ B\,(\mu[s_0]), s = bag(s_0) \\ \mu[s_0]^*, s = lista(s_0) \end{cases}$$

2.6 Definição do Grafo Tipo do Esquema EER

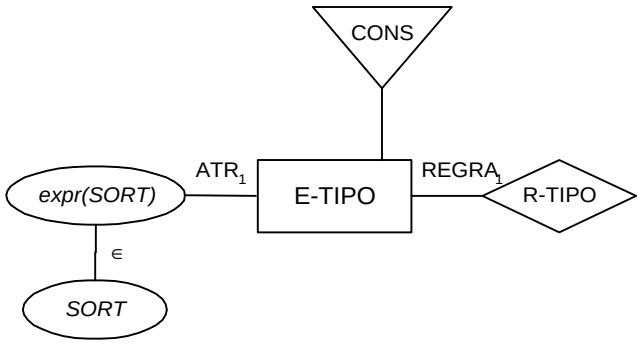

FIGURA 2.1 - Grafo Tipo de um esquema EER

Um grafo que representa um esquema EER é formado por instâncias de vértices que representam entidades, relacionamentos, construções, estruturas e classes e por mapeamentos entre estes vértices através de arestas que representam atributos e regras. Os vértices que representam as *entidades* não podem possuir nenhuma aresta ligando-os entre si. A identificação de que uma entidade possui algum tipo de relação com outra é

indicada com o auxílio de um vértice representando um *relacionamento* e duas arestas representando as ligações entre este relacionamento e as duas entidades, chamadas *regras*. A definição dos *atributos* que compõem uma entidade é realizada através de uma ou mais arestas mapeando o atributo para uma *classe de dados* ou para uma *estrutura de dados* a qual o atributo está vinculado, neste caso, a estrutura deverá estar mapeada para uma classe de tipo de dados. Uma *construção* representa a classificação de *uma entidade em duas ou mais entidades que* passam a herdar seus atributos. Todos os vértices possuem rótulo e atributos, todas as arestas, exceto as ligadas às construções, possuem atributos. Não há necessidade da representação de todos os atributos de uma entidade ou relacionamento, sendo utilizado para isso uma tabela auxiliar.

2.7 Definição do Esquema EER

Um esquema EER é um grafo, conforme definido na seção 2.1, contendo instâncias dos vértices e arestas da fig. 2.1 que obedece certas restrições.

Seja *DT* um tipo de dado assinatura. Um esquema EER S_{DT} sobre *DT* consiste de

- cinco conjuntos finitos *E-TIPO, R-TIPO, REGRA, ATR, CONS* baseados em ***RV*** e ***RA***.
- É proposta uma expansão de *W* em sete diferentes funções especializadas de acordo com o tipo do vértice.

participantes :	*R-TIPO*	$E\text{-}TIPO^+$
relacionam :	*REGRA*	*R-TIPO*
entidade :	*REGRA*	*E-TIPO*
proprietário :	*ATR*	F (*E-TIPO* *R-TIPO*)
domínio :	*ATR*	expr($SORT_{DT}$)
entrada,saída :	*CONS*	F(*E-TIPO*)

O grafo é submetido a três restrições, relacionadas a seguir.

A existência de um vértice rotulado como *relacionamento* obriga a existência de arestas rotuladas como *regras* em igual número ao de vértices do tipo *entidade*. De modo que cada regra esteja mapeada para o *relacionamento* e para uma *entidade*. Tornando obrigatória relação indicada na fig. 2.2 e a regra 1.

FIGURA 2.2 - Representação correta de uma formação de um relacionamento.

1. Para cada $R \in$ *R-TIPO* com participantes(R) = $<E_1,...,E_m>$, este deve possuir *m* diferentes $P_i \in$ *REGRA* (*1* *i* *m*) com relacionam(P_i) = R e entidade(P_i) = E_i.

Para que uma construção seja considerada válida, não é admitido que um vértice *entidade* seja mapeado por dois vértices construção[1], também não é admitido que um vértice *entidade* seja mapeado pelas funções de entrada e saída para uma mesma construção, como na fig. 2.3, obedecendo as restrições 2 e 3 .

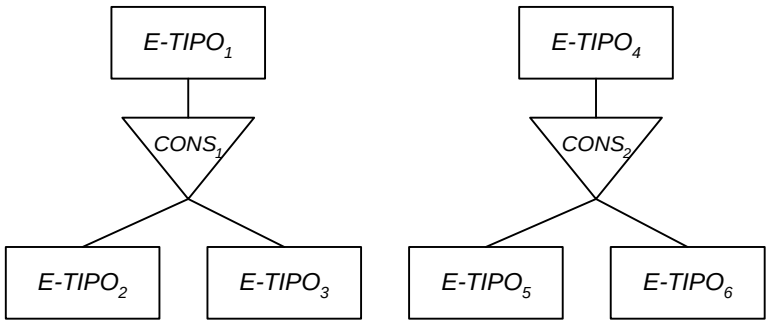

FIGURA 2.3 - Representação correta de uma formação de construções.

2. Para C_1, $C_2 \in CONS$, se C_1 C_2, então saída(C_1) saída(C_2)=ĭ
3. Seja $E \in E\text{-}TIPO$. Então (E, E) não deve estar em um fecho transitivo da relação {(I,O) C $CONS$, $I \in$ entrada (C) , $O \in$ saída (C)}

O fato de *relacionam* e *entidade* serem funções que mapeiam regras para um tipo de relacionamento simples e para um tipo de entidade, implica que nomes de regras devem ser únicas dentro de um esquema EER.

Da mesma forma, cada atributo ou componente é associado a um único domínio pela função correspondente. Já a função proprietário mapeia cada atributo e componente a um conjunto de tipos de entidades e relacionamentos. Consequentemente, entidades e relacionamentos que precisam possuir atributos e componentes de mesmo nome necessitam possuir o mesmo domínio.

Na ilustração a seguir, será apresentado parte do esquema EER correspondente ao diagrama EER apresentado na fig. 5.1.

E_TIPO = {Pessoa, Paciente, Convênio, Médico, Agenda, ...}
$R\text{-}TIPO$ = {dispõe, tem hora marcada, está alocado, ...}
$REGRA$ = {marcação, reserva, alocação, ocupação, ...}
ATR = {tipoCID, nome, códigoEspec, ...}
$CONS$ = {tipo}

participantes (está alocado) = (MÉDICO, AGENDA)
relacionam (alocação) = está alocado
entidade(alocação) = AGENDA
proprietário(CID) = {DOENÇA}
domínio(CID) = tipoCID
proprietário (residência) = {PACIENTE}

[1] Tradicionalmente, modelos orientados a objetos definem isso como a impossibilidade de herança múltipla.

domínio (residência) = lista(endereço)
proprietário (nome) = {CONVÊNIO, PESSOA}
entrada (tipo) = {PESSOA}
saída (tipo) = {PACIENTE, MÉDICO}

2.8 Definição de Universo de Entidades

Seja s um esquema EER. $E = \bigcup_{E \in E\text{-}TIPO} E_E$ é um conjunto infinito chamado universo de entidades. Isto inclui para cada tipo de entidade E em s um conjunto infinito EE de entidades do tipo E.

Para diferentes tipos de entidades E e E', $E_E \cap E_{E'} = \varnothing$

2.9 Definição de Instância de EER

Seja s_{DT} um esquema EER sobre uma assinatura de tipo de dado *DT*. Uma instância de EER I sobre s_{DT} consiste de uma detalhamento da função π (função de atribuição de vértices) e das seis seguintes funções:

- $\mu[E\text{-}TIPO]$, o qual mapeia cada tipo de entidade $E \in E\text{-}TIPO$ para um subconjunto finito de s_E;
- $\mu[R\text{-}TIPO]$, mapeia cada tipo de relacionamento $R(P_1 : E_1,...,P_m : E_m) \in R\text{-}TIPO$ para um conjunto finito de tuplas de entidades chamadas relacionam, tal que $\mu[R\text{-}TIPO](R) \subseteq \mu[E\text{-}TIPO](E_1) \times ... \times \mu[E\text{-}TIPO](E_m)$;
- $\mu[REGRA]$, o qual mapeia cada regra $P_i : R \quad E$ para a função $\mu[REGRA](P_i) : \mu[R\text{-}TIPO] \quad \mu[E\text{-}TIPO]$ satisfazendo $\mu[REGRA](P_i)(r) = e_i$ $(1 \le i \le m)$ para cada $e = (e_1, ... ,e_m) \in \mu[R\text{-}TIPO](r)$
- $\mu[ATR]$, o qual mapeia cada atributo $A : E \quad D'$ para uma função $\mu[ATR](A) : \mu[E\text{-}TIPO](E) \quad \mu[expr(SORT_{DT})](D')$ e $A : R \quad D'$ para uma função $\mu[ATR](A) : \mu[R\text{-}TIPO](R) \quad \mu[expr(SORT_{DT})](D')$;
- $\mu[CONS]$, o qual mapeia cada construção $T(I_1, ... , I_n; O_1, ... , O_m)$ para uma função $\mu[CONS](T) : \bigcup^{m}_{j=1} \mu[E\text{-}TIPO](O_j) \quad \bigcup^{n}_{k=1} \mu[E\text{-}TIPO](I_k)$.

Como uma ilustração, será apresentado parte de uma instância de EER sobre o exemplo proposto.

Neste caso *VA* (conjunto de valores de atributos) é definido por:

$$VA = \{p1, p2, p3, m1, m2, m3, e1, e2, a1, a2, a3, 13{:}30\}$$

$$\mu[E\text{-}TIPO](\text{PESSOA}) = \{p_1, p_2, p_3\}$$
$$\mu[E\text{-}TIPO](\text{MÉDICO}) = \{m_1, m_2, m_3\}$$
$$\mu[E\text{-}TIPO](\text{ESPECIALIDADE}) = \{e_1, e_2\}$$
$$\mu[E\text{-}TIPO](\text{AGENDA}) = \{a_1, a_2, a_3\}$$

$$\mu[R\text{-}TIPO](\text{está alocado}) = \{(m_1, a_2), (m_2, a_3)\}$$
$$\mu[REGRA](\text{ocupação})(m_1, a_2) = m_1$$
$$\mu[REGRA](\text{alocação})(m_1, a_2) = a_2$$
$$\mu[ATR](\text{Hora da Consulta})(a_2) = 13{:}30$$
$$\mu[CONS](\text{tipo})(m_1) = p_1$$
$$\mu[CONS](\text{tipo})(m_2) = p_3$$

Esta instância inclui dados sobre três pessoas p_1, p_2, p_3 sendo p_1 dados do médico m_1 e p_3 dados do médico m_2. O médico m_3 não possui os dados de pessoa alguma. O médico m_1 possui as especialidades e_1 e e_2 e o agendamento a_2 que possui horário de consulta marcado para as 13:30.

As instâncias são representadas através de seu conteúdo colocado sob ou sobre o seu domínio, como na figura abaixo:

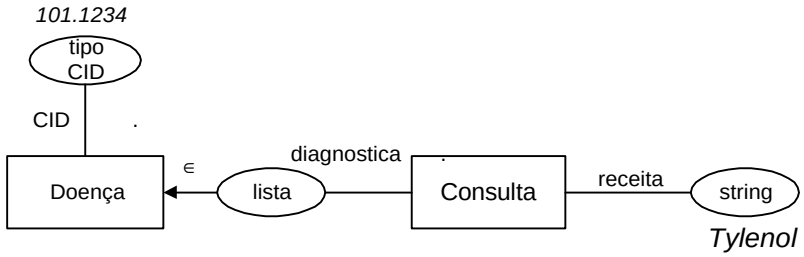

FIGURA 2.4 - Representação para instância em um modelo EER

2.10 Atualização de atributo

Dado um esquema ER s, uma atualização de atributo sobre s possui como entrada:

- uma instância ER $I = (V, W, \lambda, \pi)$ sobre s
- $A \in$ ATR
- $e \in V$ tal que proprietário(A) $\in \lambda(e)$
- $val \in VA$
- um dos seguintes
  1. $v \in V$ tal que domínio(A) = $\lambda(v)$ e $\pi(v)$ sejam definidos
  2. $v \in \mu$[ATR](domínio(A))

O resultado da aplicação da atualização de atributo

ATRupd [*P, A, e, val*]

para um padrão ER I sobre *s* é definida como o resultado do seguinte algoritmo:

I´:=I
$VA_{I'} := VA_{I'} \cup \{ val \}$
$\pi_{I'} := \pi_{I'} - \{ (v, x) \}$ (para algum $x \in VA_I$)
$\pi_{I'} := \pi_{I'} \cup \{ (v, val) \}$
$VA_{I'} := VA_{I'} - \{ x \mid v \in V_{I'}: (v, x) \in \pi_{I'} \}$
retorna (I´)

Abaixo está um exemplo de atualização de atributo ATRupd [*P, A, e, v*] com

- o padrão *P* = (V, W, λ, π) consistindo de
  - V = {e_1,v_1}
  - W = {(e_1,receita, v_1)}
  - λ é definido por meio da tabela

vértice	rótulo
e_1	CONSULTA
v_1	string

  - π mapeia v_1 para 0.

- o tipo de atributo *A* é igual a receita, sendo o tipo de atributo a ser atualizado.
- a entidade *e* é igual a e_1, sendo a entidade na qual o atributo será atualizado.
- o valor *val* é igual a *´Aspirina´*, sendo o novo valor do atributo.

A atualização poderia se representada através da seguinte regra:

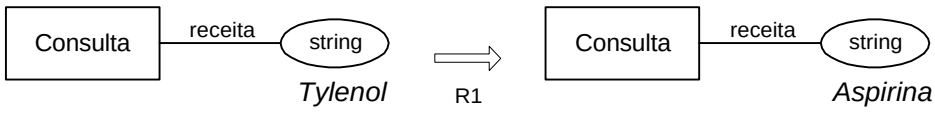

FIGURA 2.5. Exemplo de atualização de atributo sob forma de regra.

3 Definição formal do Esquema de Fluxo de Dados

Um esquema de fluxo de dados promove o dinamismo e representa a transformação dos dados em um determinado sistema de banco de dados.

Ele pode ou não ser utilizado relacionado às estruturas do modelo de dados. No entanto, a sua utilização em conjunto é benéfica, pois permite que detalhes referentes à manipulação das informações, usualmente representados sob forma de outros diagramas, possam, através de uma semântica unificada, apresentar os contextos estático e dinâmico do sistema.

Os diagramas que promovem a representação da transformação dos dados em um sistema se baseiam nos seguintes componentes:

- Fluxos de dados
  Representam as informações que trafegam pelos diversos módulos de um sistema. Os fluxos de dados necessitam de *processos* para se movimentarem no modelo. A partir de um *agente externo*, um *processo* recebe um *fluxo de dados* e o envia para um outro *processo* ou para uma *entidade*. Os fluxos de dados fazem parte de classes de dados que devem ser as mesmas dos componentes equivalentes nas *entidades*. Na fig. 3.1 podemos observar dois fluxos de dados, um representando um ramo aferente (*a*) e outro representando um ramo eferente (*b*).
- Agentes externos
  São os terminadores, de onde as informações surgem ou para onde as informações são enviadas. São representados em geral ao redor do modelo.
- Processos
  Eqüivalem aos módulos de um sistema. São os responsáveis pela movimentação dos fluxos de dados no sistema. Possuem normalmente ramos aferentes e eferentes, a ausência de um ramo aferente é admitida em casos de módulo de geração dados aleatórios, por exemplo. A ausência de pelo menos um ramo eferente não é admitida.

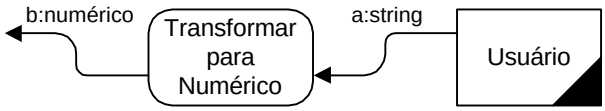

FIGURA 3.1 - Exemplo de processo de transformação de dados.

Na proposta apresentada, os *fluxos de dados* são representados através de linhas curvas, cada linha com o seu nome de fluxo e seu tipo. Os *processos* são representados através de retângulos com as bordas arredondadas e os *agentes externos* por retângulos com o canto inferior direito marcado por uma tarja.

Se observado como um grafo, um esquema de fluxo de dados é um grafo rotulado e com atributos, onde os fluxos de dados representam as arestas do grafo e os demais componentes representam os vértices.

Os fluxos de dados que chegam ou partem de um processo devem, obrigatoriamente, estar contidos na expansão deste processo. A isso chama-se *balanceamento*.

O balanceamento é a chave que garante a integridade dos diversos níveis de abstração dentro de um modelo, ele garante que os processos de nível mais baixo transformam as mesmas informações recebidas pelo processo de nível mais alto, produzindo a mesma resposta.

Em geral isso é representado como um processo da expansão recebendo a mesma informação recebida pelo processo de nível superior e um outro processo enviando a informação enviada pelo processo de nível superior, como no exemplo abaixo.

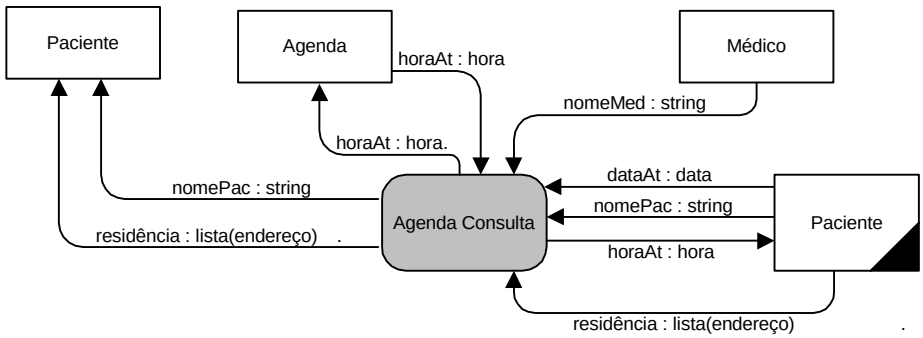

FIGURA 3.2 - Processo de nível 1

Que, expandida, deveria apresentar a união de seus fluxos de dados da seguinte forma:

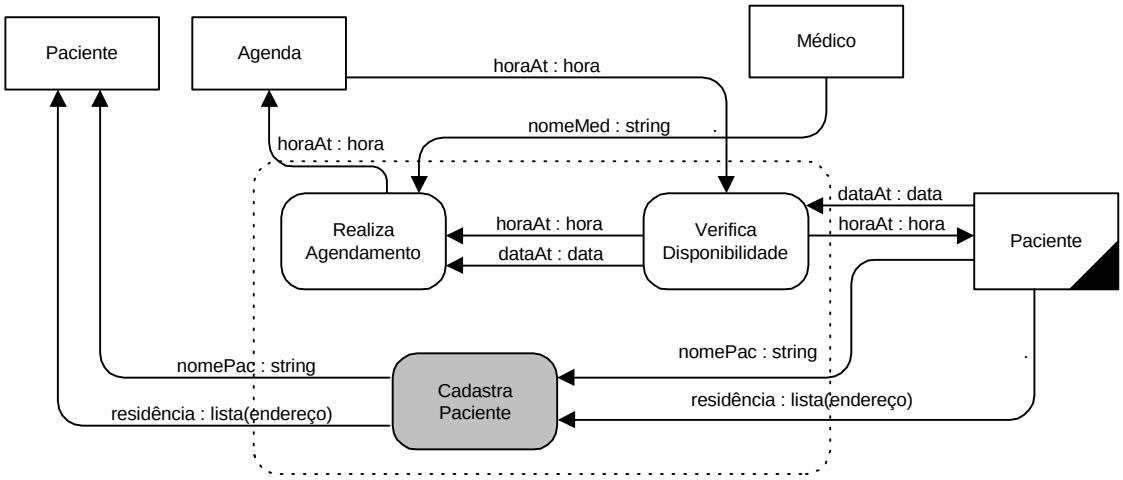

FIGURA 3.3 - Expansão do processo *Agenda Consulta*

O fluxo de dados é representado por uma aresta com a seta determinando o sentido do fluxo. As funções origem e destino relacionam o fluxo com a sua origem e destino.

Devido ao fato da principal função de um processo ser promover a transformação das informações existentes em um sistema, fluxos de dados entre duas entidades, entre uma entidade e um agente externo e entre agentes externos não são admitidos.

A expansão de um processo é caracterizada pela necessidade de execução de módulos de nível mais baixos para a total execução de um módulo de nível superior.

Um processo pode possuir ou não expansão. Em caso de necessidade de expansão, pode-se identificar quais os sub-processos relacionados ao processo de nível superior através da função expansão.

Observando o Esquema de Fluxo de Dados como um grafo, os processos podem possuir três estados diferentes, habilitados, executados e não habilitados (ou em espera), conforme a notação apresentada na fig. 3.4, sendo esses, atributos especiais.

FIGURA 3.4 - Notação para processos habilitados, executados e não habilitados

3.1 Definição de Grafo

Existem duas variações na definição do grafo para o modelo EFD em comparação com o modelo EER. No modelo EER apenas vértices possuíam atributos, no modelo EFD algumas arestas também possuem, aquelas relacionadas a dados que fluem pelo sistema. Também no modelo EFD as arestas possuem tipos.

Na proposta a ser apresentada os dois modelos já se encontram relacionados, assim, para construir a definição de grafo do modelo EFD incorporou-se o mapeamento entre arestas e atributo e entre arestas e tipos. Assim:

Seja *RV* um conjunto contável de rótulos de vértices. *RA* um conjunto contável de rótulos de arestas, *VA* um conjunto contável de valores de atributos e $SORT_{DT}$ o conjunto de tipos.

Um grafo sobre (*RV, RA, VA*) é uma sêxtupla (V, W, λ, κ, s, τ), onde

- V é o conjunto de vértices,
- $W \subset (V \times \mathrm{RA} \times V)$ o conjunto de arestas,
- $\lambda : V \quad RV$ a função de rotulagem de vértices,
- $\kappa : W \quad VA$ a função de atribuição de arestas,
- $s : V \quad \{habilitado, executado, espera\}$
- $\tau : W \quad \mathrm{expr}(SORT_{DT})$ a função de tipagem de arestas.

3.2 Definição do Grafo Tipo do Esquema EFD

No Grafo Tipo representado na fig. 3.5, pode-se observar que não são admitidos *fluxos de dados* entre duas *entidades*, entre uma *entidade* e um *agente externo* e entre *agentes externos* (pois não existem arestas curvas entre vértices). As outras restrições necessárias Todo *processo* deve ser, pelo menos, origem de uma aresta. Mais de uma aresta pode relacionar dois vértices, no entanto estas deverão possuir rótulos diferentes.

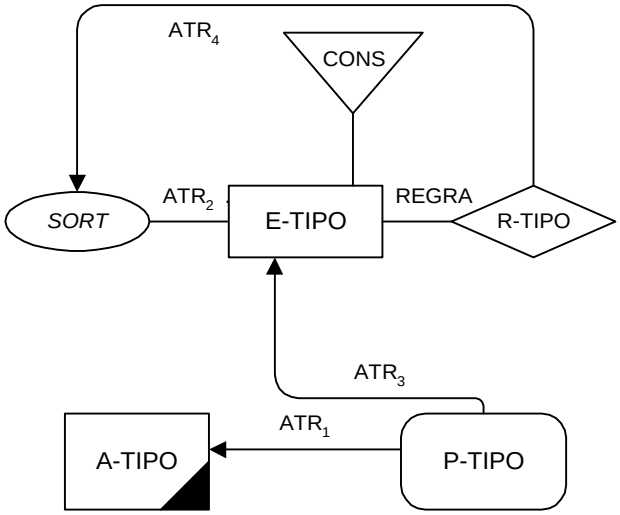

FIGURA 3.5 - Grafo tipo do esquema EFD

3.3 Definição do Esquema de Fluxo de Dados

(Seja *DT* um tipo de dado assinatura. Um esquema EFD F_{DT} sobre *DT* consiste de um grafo, conforme definido na seção

- cinco conjuntos finitos *E-TIPO, R-TIPO, REGRA, ATR, CONS* gerados a partir do modelo EER e dois conjuntos finitos *P-TIPO* (representando o conjunto dos tipos de processos) e *A-TIPO* (representando os tipos de Agentes Externos) referentes ao modelo funcional, ambos baseados em *RV* e *RA*.
- é proposta uma expansão da definição de *W* em quatro funções.

fluxo : W $\rightarrow$ *ATR* tal que $\forall (v, atr, v') \in W$: fluxo $(v, atr, v') = atr$

origem : W $\rightarrow$ *V* tal que $\forall (v, atr, v') \in W$: origem $(v, atr, v') = v$

destino : W $\rightarrow$ *V* tal que $\forall (v, atr, v') \in W$: destino $(v, atr, v') = v'$

expansão : *P-TIPO* $\rightarrow$ *F*(*P-TIPO*)

Além da mesma função domínio do modelo EER, aqui relacionada à definição das classes referentes aos fluxos de dados

domínio : *ATR* $\rightarrow$ expr($SORT_{DT}$)

Satisfazendo as seguintes regras:

1. Para garantir que um processo sempre possua um ramo eferente, utiliza-se a seguinte expressão:

$$\exists\, w \in W : \text{origem (w)} = P$$

2. Para que o modelo esteja balanceado, é necessário que as arestas cuja origem ou destino são um determinado processo *P*, façam parte da expansão deste processo *P*. Isto é garantido através da seguinte regra.

   Para cada $P \in$ *P-TIPO* se expansão(*P*) ĭ, então considera-se $arestas_P$ àquelas arestas cuja origem ou destino são *P*.

$$arestas_P = \{w \in W \mid origem(w) = P \vee destino(w) = P\}$$

As arestas pertencentes à expansão de *P* que não ligam dois vértices da própria expansão de *P* são definidas como

$$arestas_{exp(P)} = \{ w \in W \mid origem(w) = p_i \wedge destino(w) \notin expansão(P)) \vee destino(w) = p_i \wedge origem(w) \notin expansão(P))\}$$

Para que o modelo esteja balanceado, é necessário que

$$arestasP = arestas_{exp(P)}$$

Um exemplo de instâncias em um esquema EFD, baseado nas figuras 3.2 e 3.3, poderia ser o seguinte:

P_TIPO = {Agenda Consulta, Realiza Agendamento, Cadastra Paciente, ...}
A-TIPO = {Paciente}
ATR = {dataAt, horaAt, nomePac, residência, ...}

fluxo (Cadastra Paciente, Paciente, nomePac) = nomePac
fluxo (Agenda Consulta, Paciente, residência) = residência
origem (Cadastra Paciente, Paciente, nomePac) = { Cadastra Paciente }
destino (Cadastra Paciente, Paciente, nomePac) = { Paciente}
origem (Agenda Consulta, Paciente, residência) = { Agenda Consulta }
destino (Agenda Consulta, Paciente, residência) = { Paciente }
estado(Agenda Consulta) = habilitado
estado(Realiza Agendamento) = espera
expansão (Agenda Consulta) = { Realiza Agendamento, Verifica Disponibilidade, Cadastra Paciente }

3.4 Definição de Instância de EFD

Seja S_{DT} um esquema EFD sobre uma assinatura de tipo de dado *DT*. Uma instância de EFD I sobre S_{DT} consiste de

- uma instância do modelo EER de S_{DT}
- dos conjuntos *P-TIPO* e *A-TIPO* de S_{DT}
- das funções origem, destino, expansão, domínio e fluxo de S_{DT}
- e da função

$$\text{valor} : W \quad \mu[\text{expr}(SPORT_{DT})] \quad \{nulo\}$$

tal que $\forall w \in W$

$$S = \text{domínio}\,(\text{fluxo}(w)) \wedge \text{valor}(w) = v \;\Rightarrow\; v \in \mu[S]$$

Garantindo que o atributo possui o mesmo tipo de dado do valor correspondente.

A seguir são apresentados dois exemplos da função valor de instâncias de EFD:

valor (Cadastra Paciente, nomePac, Paciente) = ´João da Silva´
valor (Agenda Consulta, residência, Paciente) = ´Rua das hortências, 20´

3.5 Regras que regem o EFD

Aqui será apresentado um conjunto de regras de rescrita baseadas no Grafo Tipo (fig. 3.5) e geradas a partir de um Grafo Inicial (fig. 3.6).

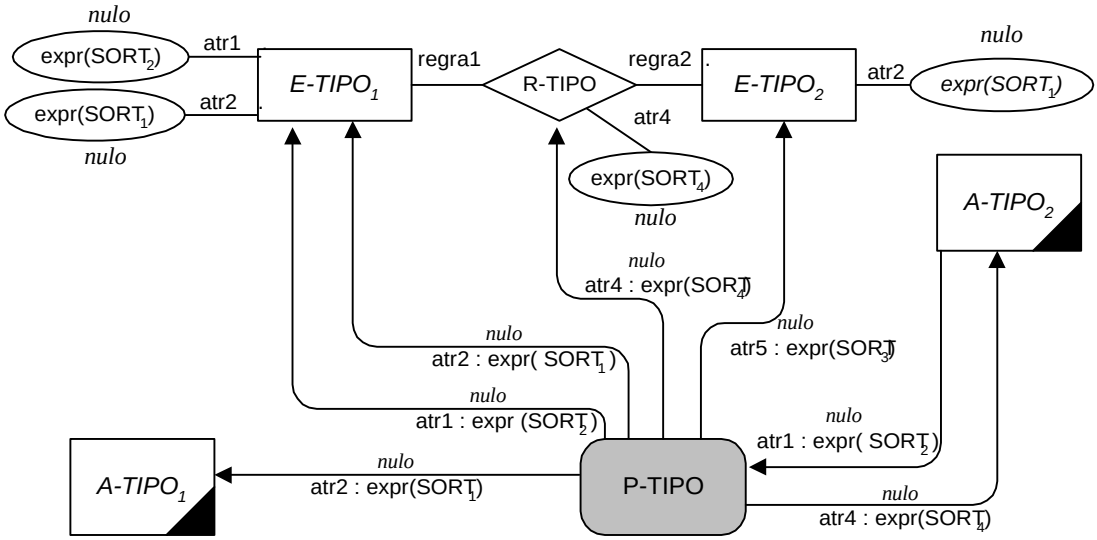

FIGURA 3.6 - Grafo Inicial

Os grafos a seguir procuram apresentar um exemplo das diversas situações possíveis de ocorrer em termos de transição de estados em um grafo representando

fluxo de dados. É importante lembrar que em um modelo de fluxo de dados, os estados são caracterizados pela execução dos diversos processos que compõem o sistema. Cada processo transformado pode significar a geração de um subgrafo, a transformação de uma informação ou a atualização de uma estrutura de dados estática.

A notação utilizada inclui a utilização das instâncias de valores colocadas acima do nome do atributo. No grafo inicial, todos valores são definidos como *nulos*.

a) Regra 1 – Expansão de processos

Na expansão de um processo, observa-se que as arestas que partem ou chegam nos dois vértices *A-TIPO* representados se mantém. Neste exemplo, *P-TIPO1* está habilitado a ser executado, enquanto que *P-TIPO2* está em estado de espera. Em nenhum momento um processo pode estar representado sem que uma aresta esteja mapeada para outro vértice a partir dele (ramo aferente), no entanto, é possível que um processo possua apenas saída (ramo eferente).

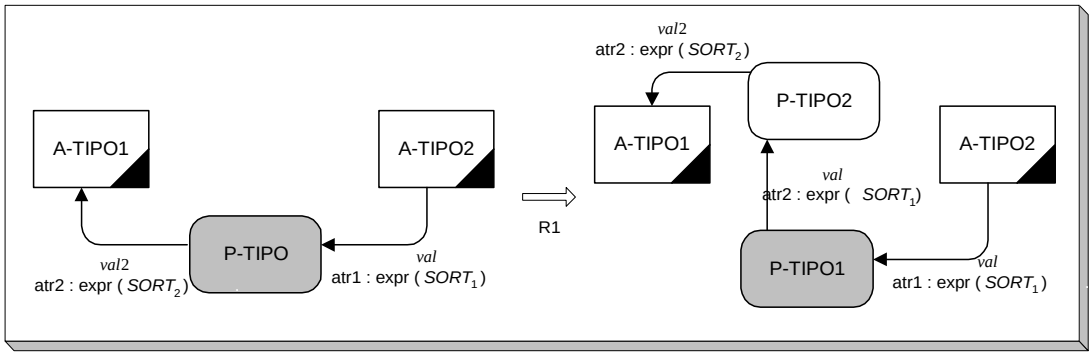

FIGURA 3.7 - Primeira regra para expansão do EFD – Expansão de processo

b) Regra 2 - Definição da ordem de execução

A Regra 2 apresenta o mecanismo de definição da ordem em que os processos são executados. Um *P-TIPO* só é considerado executado quando todos os seus subprocessos concluíram a sua execução ou quando a transformação da informação recebida foi concluída. A identificação de uma expansão ocorre pela possibilidade de aplicação da respectiva regra.

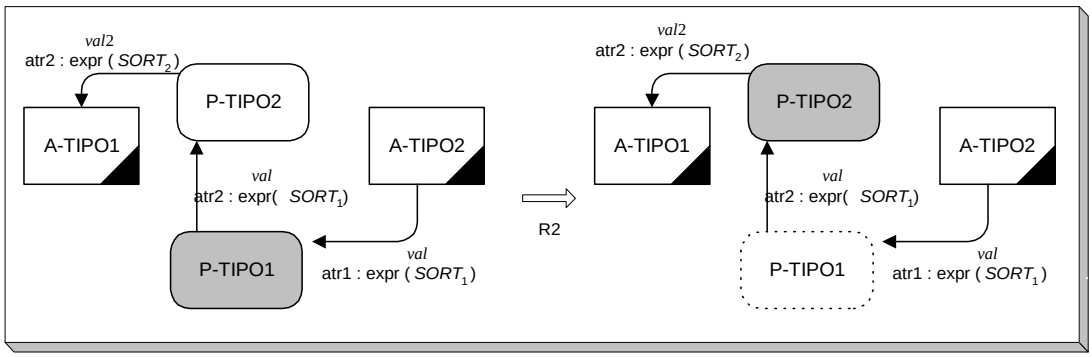

FIGURA 3.8- Regra com mecanismo de definição de ordem de execução.

c) Regra 3 - Execução do *Processo 2*

Na continuidade da execução, o *P-TIPO2* conclui o seu processamento, passando para o estado de *executado*.

Não deve existir nenhuma regra de expansão ou atualização com origem em um vértice em estado de executado. A única ação possível de ser executada sobre ele é a de *habilitação*.

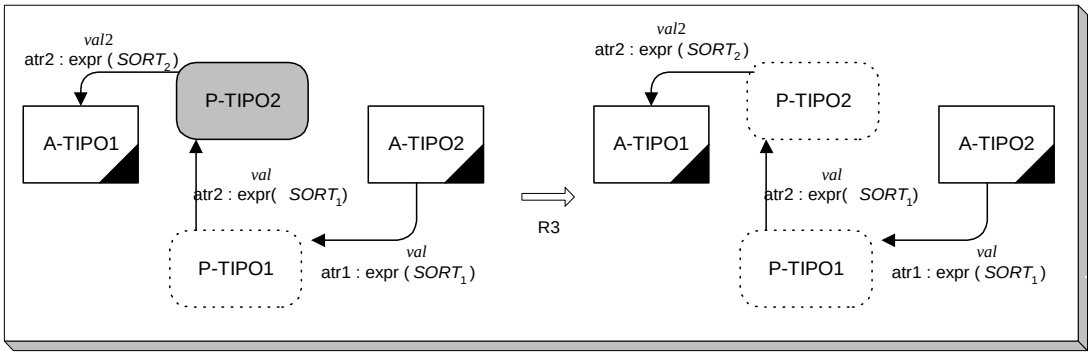

FIGURA 3.9 - Fim da execução de *P-TIPO2*.

d) Regra 4 - Compressão dos *processos 1 e 2*.

A regra 4 apresenta um processo de compressão. Onde, a partir da identificação de que todos os processos passaram para o estado de *executados* (com seus devidos rótulos), o grafo retorna para o estado anterior à expansão, exceto pelo fato do processo pai estar agora também em estado *executado*.

Quando ocorre a compressão, as arestas entre os *A-TIPOs* e os *P-TIPOs* se mantém, caracterizando um bom balanceamento do modelo.

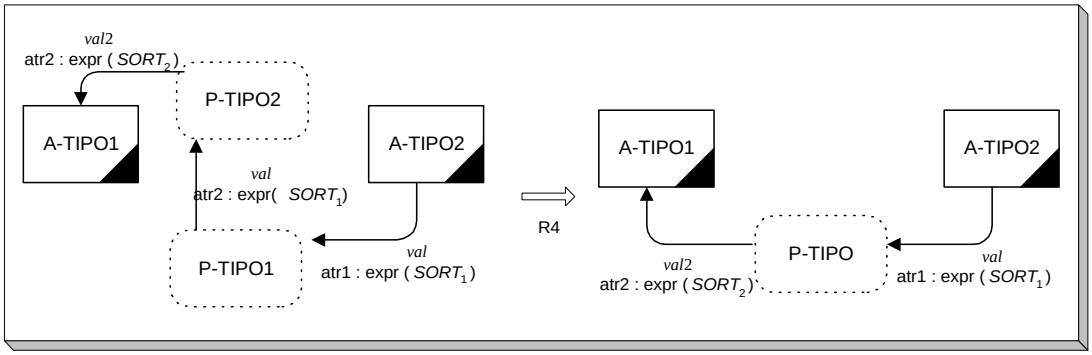

FIGURA 3.10 - Compressão dos *processos 1 e 2*.

e) Regra 5 - Atualização de *Relacionamento*

A Regra 5 mostra a atualização de um *relacionamento*. A atualização é realizada através da aplicação de uma função ATRupd( ) que instancia um atributo de uma estrutura de dados (neste caso um relacionamento). A definição da função ATRupd() pode ser encontrada na seção 2.10. Para ocorrer a atualização, é necessário que o valor informado seja do mesmo tipo da expr(*SORT*) relacionada ao atributo que faz parte do *tipo de entidade* ou *de relacionamento*. No exemplo abaixo o valor é representado por variáveis val_n que aparecem acima do seu atributo correspondente.

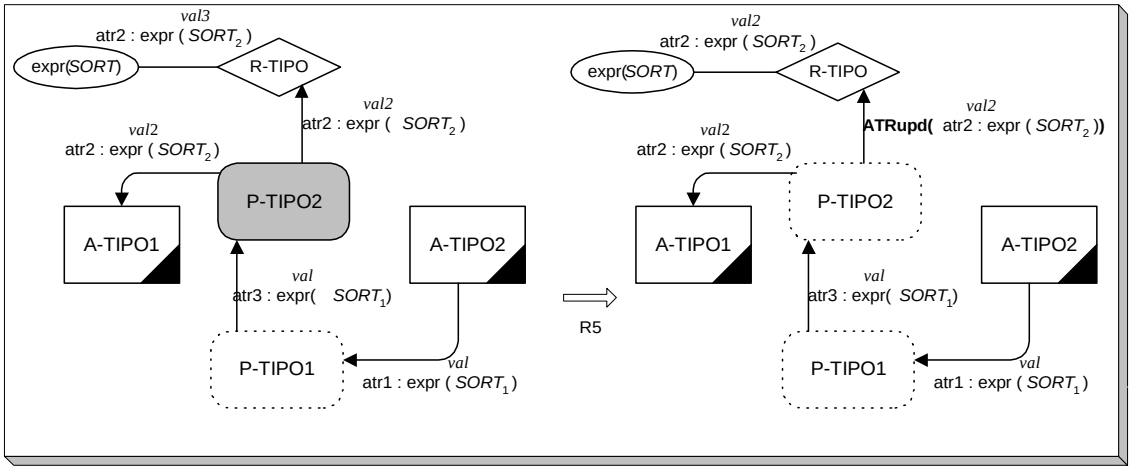

FIGURA 3.11 - Regra para atualização de atributo em relacionamento.

f) Regra 6 - Atualização de *Entidade*

Assim como na Regra 5 aqui é descrito a atualização de uma estrutura de dados, neste caso uma *entidade*. O exemplo foi gerado com a atualização de um atributo.

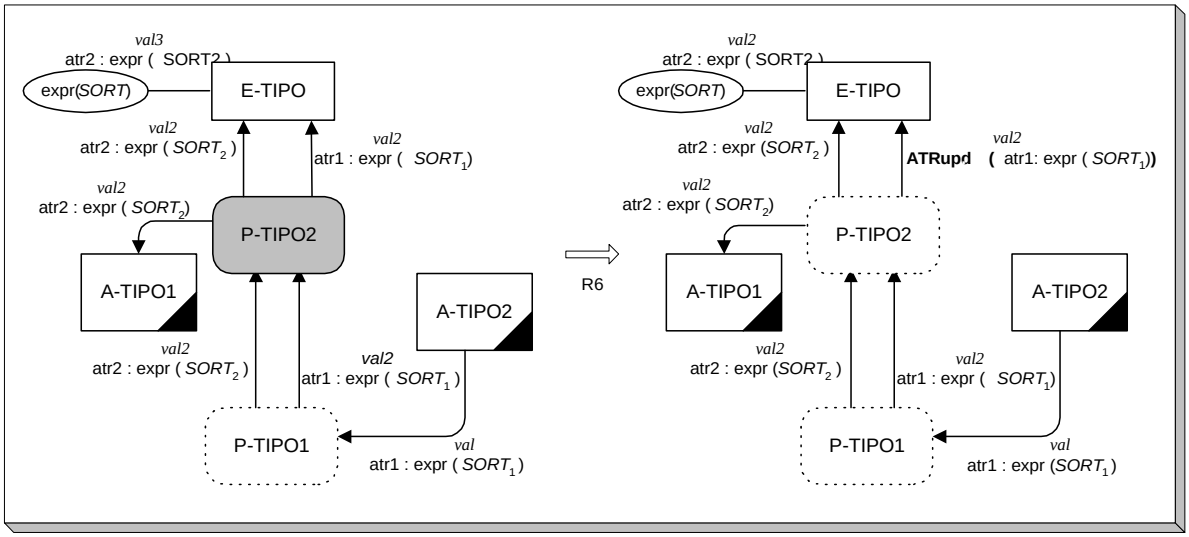

FIGURA 3.12 - Regra para atualização de atributo em entidade.

g) Regras 7 e 8 - Compressão

As regras 7 e 8 apresentam o processo de compressão para a atualização de *relacionamento* e de *entidade*.

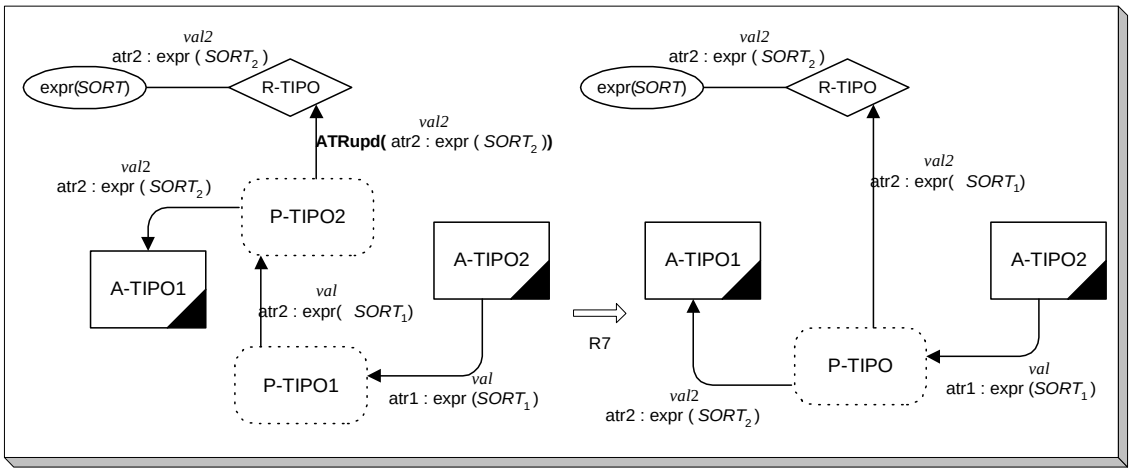

FIGURA 3.13 - Compressão de dois processos após uma atualização de *relacionamento*.

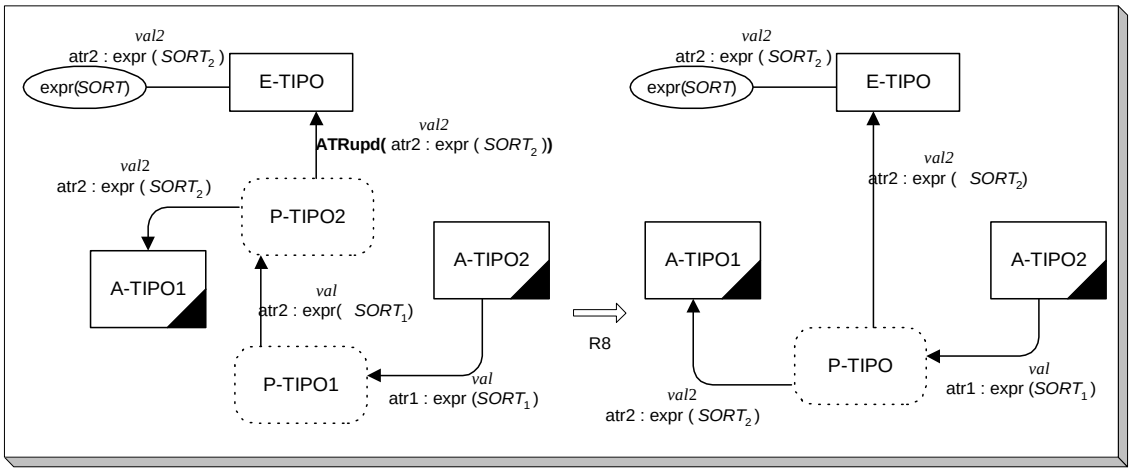

FIGURA 3.14 - Compressão dos *processos 1 e 2* após uma atualização de *entidade*.

4 Aplicação do método de modelagem baseado em rescrita de grafos.

Neste capítulo será apresentado o processo de modelagem funcional, de dados e dinâmica de sistemas através da aplicação de sistemas de rescrita de grafos.

Essa modelagem se baseia na definição dos modelos EER e EFD e na sua integração.

4.1 Processo de Modelagem – passos do método

Embora a geração dos modelos possa ocorrer em paralelo, sugere-se que seja definido inicialmente o modelo de dados, sendo posteriormente agregado o modelo funcional.

O motivo desta escolha é o fato do modelo de dados, por ser mais estável, fornecer uma base sólida para o posterior acoplamento do modelo funcional. O modelo de dados é também bastante útil na identificação das principais características do sistema, principalmente no que se refere ao seu estado inicial

4.1.1 Geração do Modelo de Dados (EER)

Para a geração do modelo de dados, é necessário extrair da realidade seus componentes (entidades, relacionamentos, atributos e estruturas).

A construção do modelo EER não diferencia muito do modelo ER tradicional, exceto pela ausência de cardinalidade. No entanto é necessário atentar para a necessidade de definição da totalidade dos objetos (vértices e arestas) que compõem o modelo, exceção à definição dos atributos que podem ser parcialmente representados no modelo e parcialmente em uma tabela auxiliar. Em geral a representação da totalidade dos atributos não é realizada nem mesmo em modelo ER tradicionais.

Pode-se destacar como precauções a serem tomadas durante o processo de modelagem:

- associar nomes às regras que unem as entidades e os relacionamentos;

Não é necessário a geração de um nome representativo para as regras (embora seja interessante), mais importante é definir um nome significativo para as entidades e relacionamentos.

- identificar corretamente as estruturas de dados complexas utilizadas pelo sistema e representá-las sob forma de conjunto, lista ou *bag*;

A definição incorreta desses tipos complexos pode comprometer o formalismo;

- definir corretamente os tipos dos atributos relacionados às estruturas de dados.

Como os tipos de dados do modelo EER e do modelo EFD se relacionam intimamente, a definição incorreta do tipo de dados pode comprometer a integridade do

modelo. O que não ocorreria se a modelagem ocorresse com o auxílio de uma ferramenta CASE de modelagem formal.

4.1.2 Geração do Modelo Funcional (EFD) incorporado ao modelo funcional

O modelo de funcional possui a característica de procurar representar prioritariamente a transformação que os dados sofrem ao longo do tempo.

O relacionamento entre os modelos de dados e funcional se dá através de seus componentes comuns, as estruturas de dados estáticas.

Em um modelo de fluxo de dados, a estrutura que armazena os dados é o depósito de dados, em um modelo de dados são a entidade e o relacionamento (em caso de relacionamentos com atributos).

Antes de iniciar o processo de modelagem é preciso extrair da realidade o tipo de informação que mais tarde será utilizado na definição dos componentes do grafo.

Um esquema de fluxo de dados manipula basicamente, *terminadores* (geradores ou consumidores de informações) chamados *agentes externos*, *informações* que fluem pelo sistema e são transformadas por *processos* que é o componente funcional do modelo.

Para gerenciar a complexidade, os processos podem ser detalhados em diversos níveis de abstração. Esse conceito possui íntima relação com os modelos baseados em estruturas modulares, que realizam o processo de decomposição funcional.

Considerando que a geração dos modelos funcional e de dados é de domínio já estabelecido, será dada ênfase nos aspectos relacionados ao dinamismo do modelo.

Algumas das regras necessárias à modelagem de um sistema foram definidas no processo de formalização do esquema EFD.

a) Regras pré-definidas pelo formalismo.

- Existem três tipos de processos: *habilitados*, não *habilitados* e *executados*;
- um processo não pode estar em dois estados diferentes;
- todo processo deve possuir, pelo menos, um ramo eferente;
- no caso de uma expansão, os fluxos de dados de entrada e saída do processo de nível superior devem fazer parte da expansão.

b) Regras pré-definidas pelo Grafo Tipo.

- O Grafo Tipo (fig. 3.5) descreve um conjunto de regras relacionadas às possibilidades de ligações entre os diversos componentes do modelo.

c) Regras de produção pré-definidas.

- As regras genéricas descritas na seção 3.4 gerenciam um conjunto de situações padrão que ocorrem durante o processo de modelagem.

Para gerar o modelo é necessário ter em mente estas regras, de modo que em nenhum momento do sistema alguma delas seja contrariada. As regras de integridade que não foram definidas através destes mecanismos devem ser expressas nas regras de produção do sistema de rescrita de grafos.

4.1.3 Geração das regras de produção

As regras de produção deverão ser geradas a partir do conceito de estados e transições, sendo que cada regra determina uma transição.

A definição das regras possui a seguinte notação,

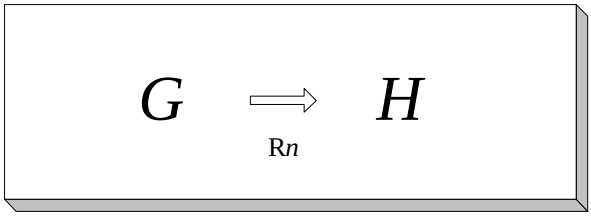

FIGURA 4.1 - Notação para definição das regras de produção

onde:
G é o grafo origem
H é o grafo resultante
R é a regras que rege a produção e
n é o número da regra

O grafo *G* deverá apresentar apenas aqueles elementos que participam da produção, sejam como identificadores de componentes que irão se manter, sejam como identificadores de elementos que não aparecerão após a produção.

O grafo *H* deverá apresentar os componentes que se manterão após a produção e aqueles elementos que serão agregados ao grafo *G*.

Cada estado do sistema possui uma ou mais regras. Todos os estados (exceto o estado final, se houver) deverão possuir regras.

Deve-se levar em consideração que, após definida uma regra, esta poderá gerar uma produção de modo independentemente, sempre que encontrar as condições definidas em *G*.

Convenciona-se que uma condição para que ocorra uma produção é a instanciação dos fluxos de dados dos ramos aferentes ou eferentes de um processo, ou seja, uma regra só pode ser aplicada se existirem valores a serem manipulados, caso contrário as regras não estão habilitadas. Como no exemplo a seguir, em que a produção ocorre apenas quando nomePac e residência possuírem instâncias associadas.

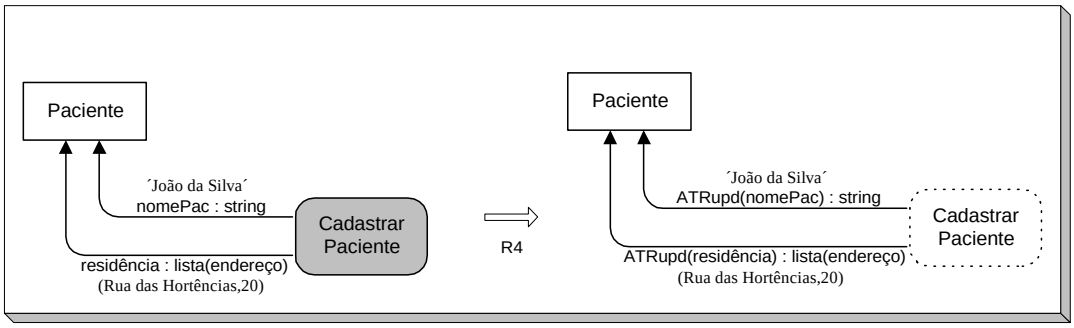

FIGURA 4.2 - Exemplo de execução de regra a partir da instanciação de atributos.

A necessidade em se definirem regras genéricas impedem que valores de atributos sejam incluídos na regra. A utilização da função ATRupd ( ) garante que o atributo equivalente na entidade *Paciente* foi atualizado, gerando uma instância. Isso pode ou não ser representado no grafo, depende da sua existência no grafo inicial ou na tabela auxiliar.

4.3 Avaliação do modelo gerado

Por último, dever ser feita uma avaliação das regras geradas a partir do estado inicial do sistema.

A realização desta tarefa deveria ser acompanhada de uma ferramenta de prova que avaliasse todas as possibilidades de aplicação de regras e a manutenção da integridade do sistema em função de sua aplicação.

Na falta de tal ferramenta, o responsável pela modelagem deverá ter cuidado especial com a complexidade inserida por regras que podem gerar produções que ocorrem em paralelo. Esse tipo de caso pode gerar situações inesperadas e, ocasionalmente, alcançar um estado (que não o final) sem regra aplicável.

5 Estudo de Caso – Sistema de Controle de Clínicas Médicas

Neste capítulo será apresentado o processo de modelagem de um sistema de automação de clínicas médicas.

5.1 – Descrição do Domínio do Problema

A domínio do problema foi gerado a partir de uma simplificação do estudo de caso apresentado por Heloísa Hertzog [HER 99], a partir de onde é proposto um modelo utilizando-se uma representação baseada em grafos no qual tanto os dados quanto as ações que os transformam são representados de modo integrado.

Inicialmente será apresentado o modelo de dados, sendo posteriormente apresentadas as ações que o transformará.

Com o objetivo de orientar a leitura, proporcionando uma visão geral do domínio do problema, este modelo será dividido em três assuntos, clínica, recepção e assunto.

a) Clínica

Na clínica podem trabalhar um ou mais médicos. Os médicos são profissionais formados e possuem um Registro Profissional que os identificam perante o CRM (Conselho Regional de Medicina), atendendo na parte de clínica geral ou dentro de suas especialidades.

No âmbito da saúde, os médicos têm como objetivo atender a todos os pacientes que os procuram, proporcionando tratamentos para solucionar os seus problemas de saúde.

Os pacientes que procuram a clínica selecionam os médicos cuja especialidade atenda aos seus problemas de saúde. Dentro da clínica, os pacientes podem solicitar consultas com diferentes médicos.

Tanto o médico como os pacientes possuem dados em comum como por exemplo: (Nome, Identidade e CPF)

Os pacientes podem possuir vários endereços.

b) Recepção

Na recepção é realizado o atendimento de pacientes que desejam agendar, confirmar ou cancelar consultas. Assim, tem como objetivo controlar todas as consultas marcadas para os respectivos médicos.

Para efetuar o agendamento de uma consulta para um paciente, quatro informações fundamentais são solicitadas (nome do paciente, nome do médico e data e hora p/ consulta).

A data e a hora identificam o dia, mês e ano em que vai ser realizada a consulta e o horário correspondente.

Uma consulta agendada pode possuir vários estados. Esses são classificados como status das consultas.

Os status podem assumir posições como: (aguardando, em consulta, atendido, não compareceu e cancelado).

Essas posições assumidas pelo status proporcionam ao médico um controle bem preciso sobre as suas consultas, podendo tirar informações do tipo: Quantos e quais são os pacientes que estão aguardando na recepção. Qual o índice de cancelamento de consultas. Quais os pacientes que já foram atendidos.

As consultas podem ser do tipo particular ou através de convênios, assim uma outra característica importante da agenda é chamada de tipo de consulta. Essa característica permite ao médico identificar os tipos de consultas já efetuadas.

Para uma melhor organização da sua agenda, o médico pode definir os seus horários para atendimento, ficando totalmente a seu critério o intervalo de tempo entre as consultas e os horários iniciais e finais para atendimento.

Por motivos diversos, o médico pode ausentar-se do consultório em dias que normalmente estaria atendendo. Como a secretária é avisada com antecedência sobre os períodos em que o médico estará ausente, ela realiza observações na agenda para que não sejam marcadas consultas nesses dias.

Essas observações ou notas da agenda possuem como características a data solicitada e a própria observação ou nota que será elaborada pela secretária.

c) Consulta

A consulta caracteriza-se pelo atendimento médico do paciente denominado de anamnese.

O termo anamnese designa o conjunto de informações recolhidas sobre os fatos de interesse médico que dizem respeito à vida de um determinado paciente. Dentre as características da anamnese, podemos observar:

1. Queixa Principal (QP): o problema que levou o paciente até o consultório.
2. História da Doença Atual (HDA): o histórico da doença atual.
3. Medicamento: medicamentos tomados.
4. Cirurgias: cirurgias realizadas.
5. Antecedentes Obstétricos (AO): ocorrência de gravidez quando pacientes do sexo feminino.
6. Antecedentes Familiares (AF): ocorrência de doenças em familiares próximos.
7. Antecedentes Pessoais (AP): ocorrência de doenças anteriores.
8. Exame Físico: exame físico realizado no paciente.
9. Conduta: procedimento tomado pelo médico.

As doenças existentes atualmente são catalogadas no CID-10. Essas recebem códigos que são padronizados internacionalmente, de modo que, em qualquer lugar, um médico é capaz de distinguir uma doença a partir do seu código.

Como as doenças, os exames e procedimentos também possuem um cadastro padronizado, realizado pela AMB (Associação Médica Brasileira). Para efeitos de documentação médica, são utilizados os códigos da AMB quando necessário referenciar algum exame ou procedimento.

Os medicamentos são organizados na EBPF (Enciclopédia Brasileira de Produtos Farmacêuticos). Nesta enciclopédia são encontrados todos os medicamentos com os seus respectivos códigos.

Cada consulta realizada apresenta como características a hora de atendimento, o valor da consulta e o dia para retorno.

A hora de atendimento é importante pois, a partir dela, pode-se levantar comparações entre a hora que o paciente solicitou uma consulta e o horário que ele foi atendido, para que o médico verifique o tempo de espera dos seus pacientes na sala de recepção.

5.2 Processo de Modelagem – passos do método

5.2.1 – Geração do Modelo EER

As características estruturais de dados relevantes à aplicação conforme descrita a seguir, podem ser expressas por meio de um esquema ER Extendido(EER), cuja definição formal foi apresentada no Capítulo 2. Embora a representação do sistema através de uma notação matemática garanta uma definição precisa e não ambígua dos requisitos do sistema, ela não é uma forma apropriada de comunicação com o usuário para representação de sistemas no nível de abstração pós especificação de requisitos. Para isso são necessários esquemas que facilitem a visualização do modelo sem, no entanto, perder-se as vantagens obtidas com a formalização da especificação.

Na fig. 5.1 é apresentado um modelo EER parcial do exemplo proposto, sendo seguido de uma breve descrição de seus componentes.

O principal componente de um esquema EER é o *tipo de entidade*. Um tipo de entidade é uma abstração para um conjunto de componentes da realidade (*entidades*), compartilhando com esta algumas características comuns. Em um sistema de controle de clínicas médicas, o papel do médico é de extrema importância, sendo necessário a definição de um tipo de entidade MÉDICO.

Para representar o relacionamento entre entidade de um dado tipo de entidade, utilizam-se *tipos de relacionamentos*, por exemplo um PACIENTE tem hora marcada em uma AGENDA. Os tipos de entidade PACIENTE e AGENDA estão sujeitas às regras de marcação e reserva respectivamente no relacionamento tem hora marcada. Tipos de relacionamentos são representados através de losangos. Um vértice não dirigido unindo o losango representando o tipo de relacionamento e o retângulo representando o tipo de entidade (como o vértice denominado marcação) representa uma regra. É importante salientar que os relacionamentos denotam dados estáticos armazenados em um banco de dados.

Além dos relacionamentos, também podem ser estabelecidos ligação de *componentes* entre entidades. Os relacionamentos em um modelo relacional correspondem à *associação* em modelo orientados a objetos, assim como os componentes correspondem à *agregação*. Enquanto a agregação é utilizada para expressar o fato de entidades de um tipo serem uma propriedade de entidades de algum outro tipo, uma associação é utilizada para modelar uma correspondência mais geral entre entidades de dois ou mais tipos.

No exemplo apresentado, um MÉDICO atua em um conjunto de ESPECIALIDADES, ordenados em uma lista. Isto pode ser observado no diagrama utilizando-se um nodo extra rotulado lista, unido ao nodo MÉDICO por meio de um vértice rotulado atua e ESPECIALIDADE por meio de um vértice rotulado $\in$.

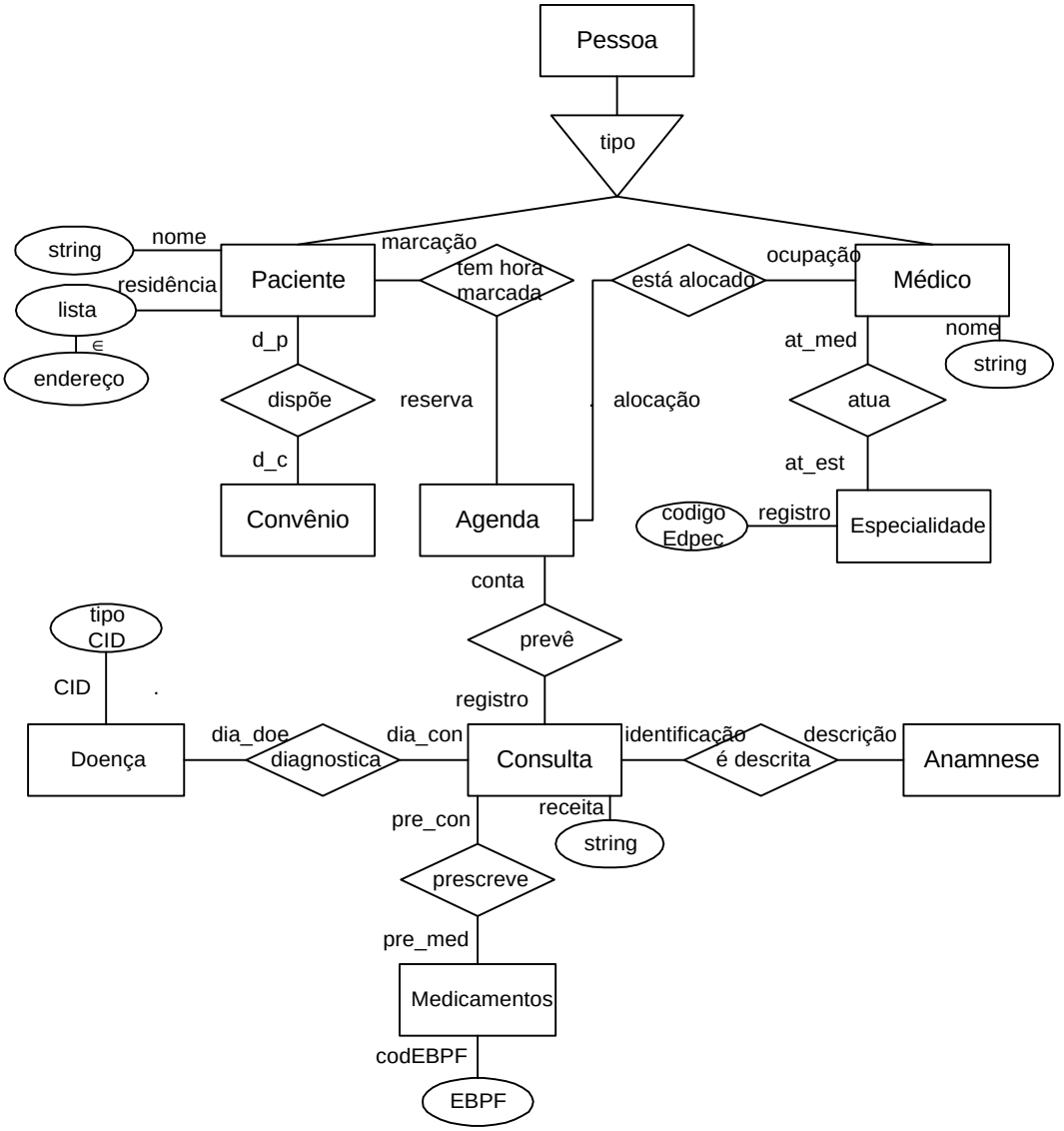

FIGURA 5.1 - Diagrama EER de um sistema de controle de clínicas médicas

Tanto as entidades como os relacionamentos são caracterizados por meio de *atributos*. Atributos são valores simples ou coleções de valores descrevendo propriedades de entidades ou relacionamentos. Por exemplo podemos observar que a entidade ANAMNESE possui o atributo receita do tipo string, enquanto PACIENTE possui uma lista de endereços como suas residências.

Os atributos são indicados em um diagrama EER por um vértice não dirigido, rotulado com o nome do atributo, unindo o retângulo correspondente ao tipo de entidade (ou relacionamento) para um vértice oval rotulado com o domínio do atributo (no caso de atributo simples) ou com a palavra chave *lista*, *conjunto* ou *bag* (no caso de atributos complexos).

De modo a facilitar a leitura do diagrama, não é representada no diagrama a totalidade dos atributos, apenas os atributos mais importantes. Os demais deverão ser representados em uma tabela apresentada abaixo que coexistirá como diagrama.

Tabela 1- Relação de atributos para o sistema de controle de cínicas médicas

Tipo de Entidade	Atributo	Tipo de Dado
Agenda	Nome do Médico	string
	Nome do Paciente	string
	Data da Consulta	data
	Hora da Consulta	hora
	Status	tipoStatus
	Notas	string
Anamnese	Receita	string
Consulta	Tipo	tipoConsulta
	Hora	hora
	Valor	dinheiro
	Data retorno	data
Convênio	Nome	string
Doença	CID	tipoCID
Especialidade	Registro	tipoEspecialidade
Medicamentos	CodigoEBPF	EBPF
Médico	CRM	numérico
Paciente	Residência	lista(endereço)
Pessoa	Nome	string
	Identidade	numérico
	CPF	numérico

Outra construção representada no diagrama é a herança, que também é bastante comum em modelos orientados a objetos. Isto permite representar novos tipos de entidades que são casos especiais de outras já existentes. Para isso é utilizado um tipo de construção representada por um triângulo invertido que recebe um número de tipos de entidade de *entrada* e redistribui algumas entidades desse tipo em um número de tipos de *saída*. Na terminologia orientada a objetos tipos de entrada poderiam ser chamados de superclasses e tipos de saída poderiam ser chamados de subclasses. No exemplo podemos observar que Paciente e Médico são definidos a partir de um tipo de Pessoa por meio de uma construção chamada tipo.

Tipos de *saída* de uma construção de herdam todas as propriedades (atributos, componentes e regras em relacionamentos) dos seus tipos de *entrada*. No exemplo, tanto PACIENTES quanto MÉDICOS possuem um atributo nome.

5.2.2 – Descrição do Modelo EFD incorporado ao modelo EER

Do ponto de vista funcional, podemos identificar as seguintes ações com sendo as principais que transformam o sistema:

1) Atendimento

Para a efetivação das atividades relacionadas ao atendimento é necessário:

1.1 - Notificar Afastamentos
1.2 – Agendar Consulta
    1.2.1 - Cadastrar Pacientes
    1.2.2 - Verificar a disponibilidade do médico

1.2.3 - Realiza agendamento
1.3 - Gerar relatório de atendimentos

2) Consulta

Com as seguintes subatividades:

2.1 - Definir o estado da consulta
2.2 - Identificar a doença.
2.3 - Gerar a Amamnese
2.3.1 - Definir a Conduta
2.3.2- Emitir a receita
2.4 - Avaliar o Tempo de Espera

Essa mesma estrutura que representa a atividades de forma nivelada (subatividades) será a estrutura dos processos que manipulam o sistema. Assim, basicamente são dois os processos de nível mais alto do sistema: Consulta e Atendimento.

Devido ao fato dos dois processos possuírem as mesmas ocorrências, o exemplo será desenvolvido sobre a parte referente ao *Atendimento*.

Na fig. 5.2. podemos observar o Grafo Inicial referente ao Atendimento.

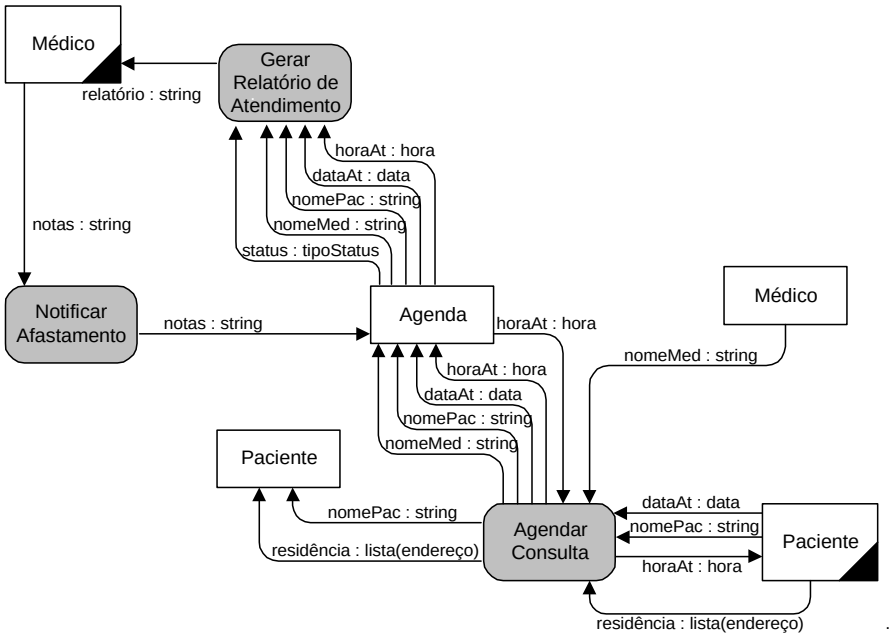

FIGURA 5.2 - Grafo Inicial do processo de Atendimento.

A partir deste grafo, pode-se definir as seguintes regras:

a) Regra 1. Execução do processo Notificar Afastamento

Como o processo Notificar Afastamento não possui expansão, a sua única ação possível é atualizar a agenda com as notas de afastamento. A atualização é realizada com o auxílio da função ATRupd( ). É importante salientar que, o fato de nem todos as entidades possuírem atributos representados no grafo faz com que algumas operação ATRupd( ) não indiquem a inclusão do atributo no grafo. Nesses casos considera-se que o atributo foi incluído.

Os valores que instanciam os atributos são representados através da variável *val*, que deve, obrigatoriamente apresentar um tipo de dado compatível com o tipo de dado do atributo.

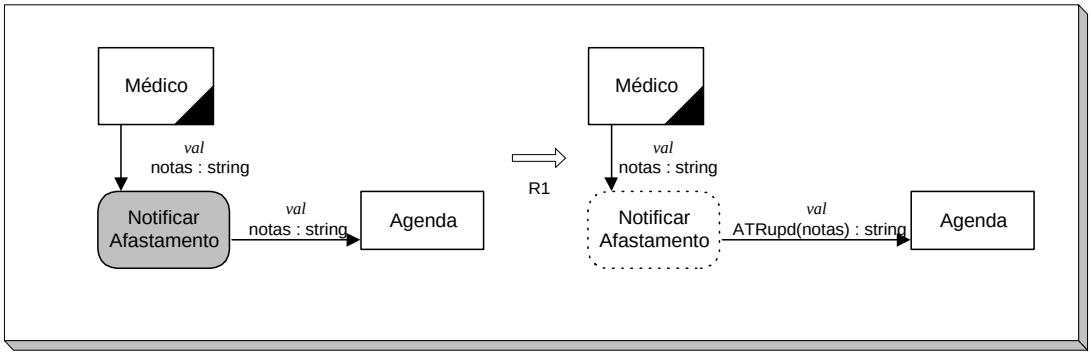

FIGURA 5.3 - Regra 1, para execução do processo Notificar Afastamento

b) Regra 2. Habilitação do processo *Notificar Afastamento*

Após a sua execução, é necessário mudar o estado do processo para que ele possa ser novamente executado. Para isso aplica-se a seguinte regra:

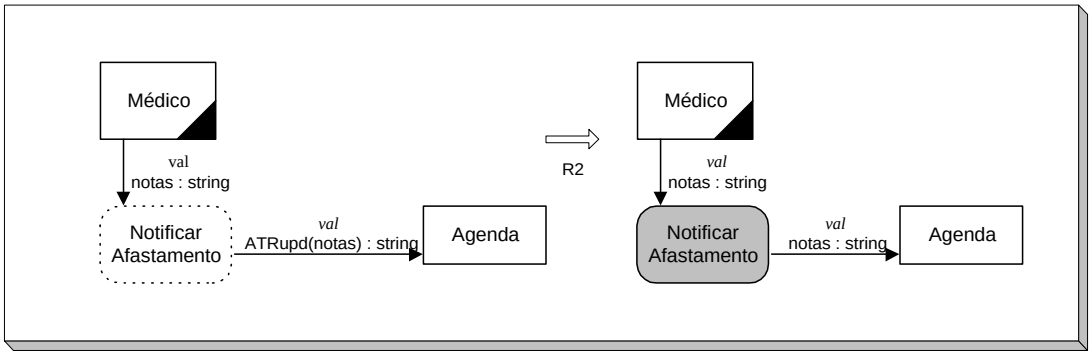

FIGURA 5.4 - Regra 2, para habilitação do processo *Notificar Afastamento*

Para diminuir a complexidade do grafo, a partir desta parte do trabalho não será apresentada a variável que instancia cada atributo, no entanto é importante salientar a necessidade de sua existência, indicando os valores dos atributos dos fluxos de dados.

c) Regra 3. Expansão do processo Agendar Consulta

Na regra 3 será apresentada a expansão do processo *Agendar Consulta*. Durante o processo de expansão é determinado o primeiro processo a ser executado, neste caso será o processo *Cadastrar Paciente*. Nota-se que os fluxos de dados de entrada e de saída se mantém na expansão.

Nenhum dos processos da expansão de Agendar Consulta possui uma nova expansão. Caso possuísse, novas regras deverias ser definidas para realizar esta tarefa.

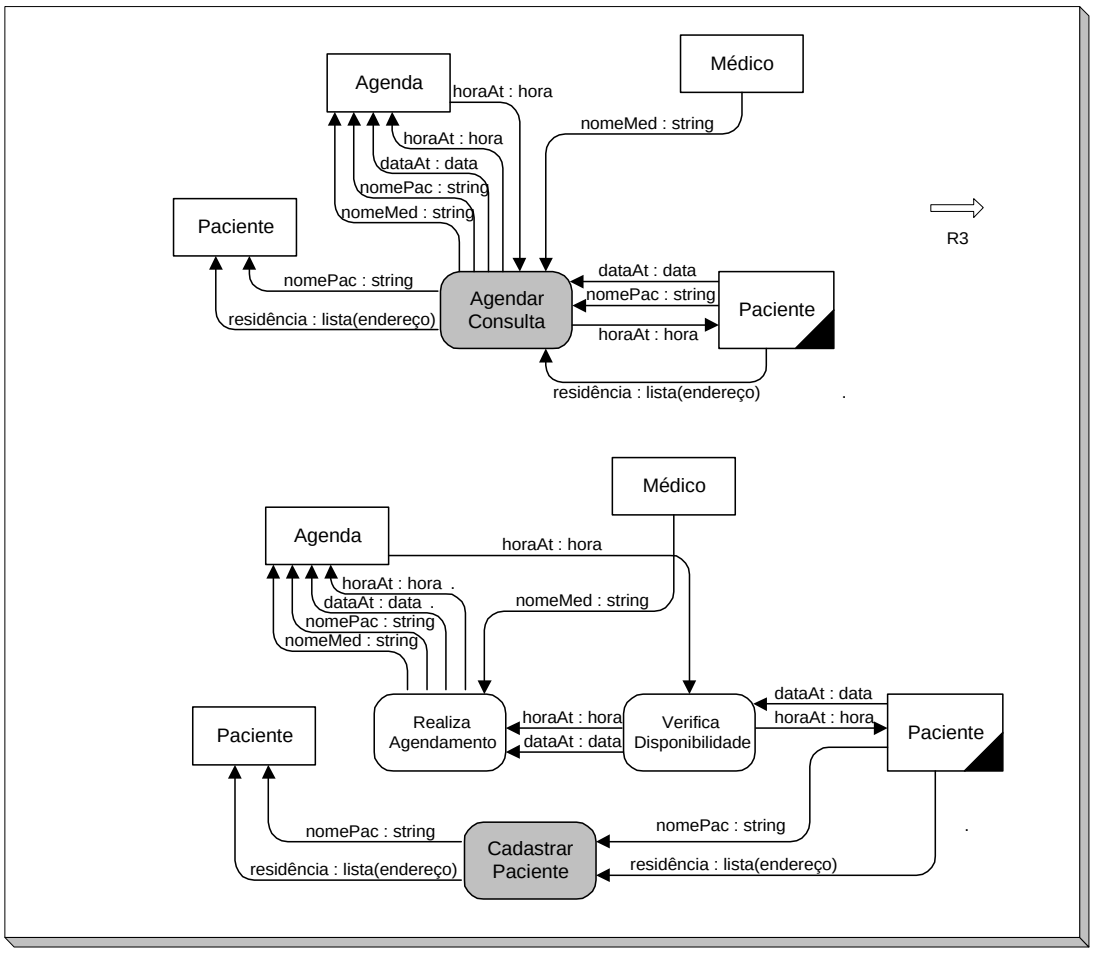

FIGURA 5.5 - Regra 3, para expandir o processo *Agendar Consulta*

d) Regra 4. Atualização da Entidade *Paciente*

O processo *Cadastrar Paciente* é um processo atômico, nenhum subprocesso derivará dele. Assim, as únicas atividades que ele pode realizar é enviar uma informação para outro processo ou agente externo ou atualizar alguma estrutura de dados estática, neste caso a entidade *Paciente*, como apresentado na regra 4.

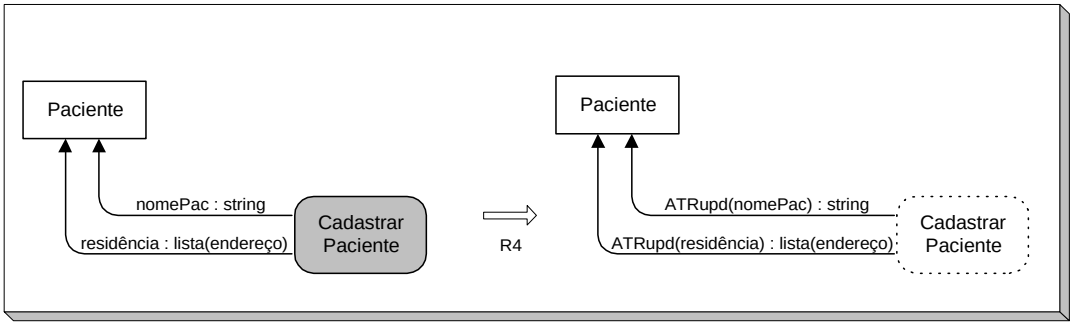

FIGURA 5.6 - Regra 4, para atualizar entidade Paciente.

Após a execução do processo, seus fluxos de dados são ajustados para que reste apenas a notação de processo em estado *executado*. Quando isso ocorre, deve ser definido o próximo processo a estar no estado *habilitado*.

Isso pode ocorrer de duas formas: a partir da expansão, como na fig. 5.7, ou a partir de regras de *habilitação*, regras 6 e 7.

A regra a seguir apresenta o processo *Verifica Disponibilidade* sendo transformado de modo a determinar a ordem de execução de processos do nível superior.

Esta regra é apenas um exemplo e não será utilizada no estudo de caso.

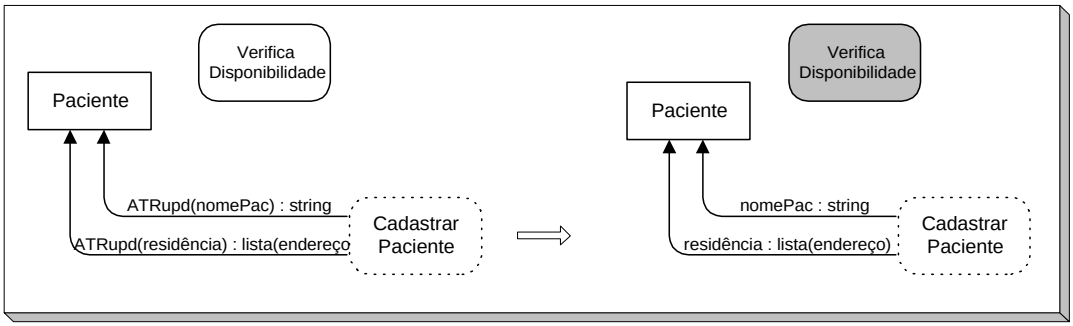

FIGURA 5.7 - Habilitação de processo de nível superior a partir da expansão.

No entanto, como a opção foi controlar a ordem de execução no nível superior, a regra a ser aplicada é a seguinte:

e) Regra 5. Desabilitação do processo *Cadastrar Paciente*.

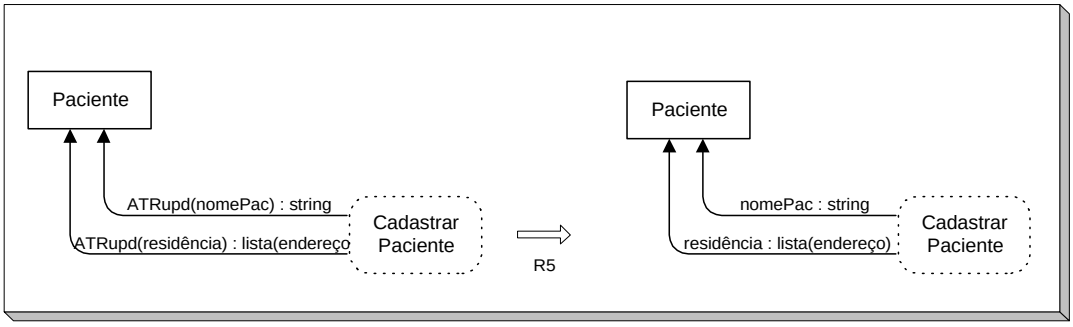

FIGURA 5.8 - Desabilitação do processo *Cadastrar Paciente*.

f) Regra 6. Definição da ordem de execução (Habilitação do processo *Verifica Disponibilidade).*

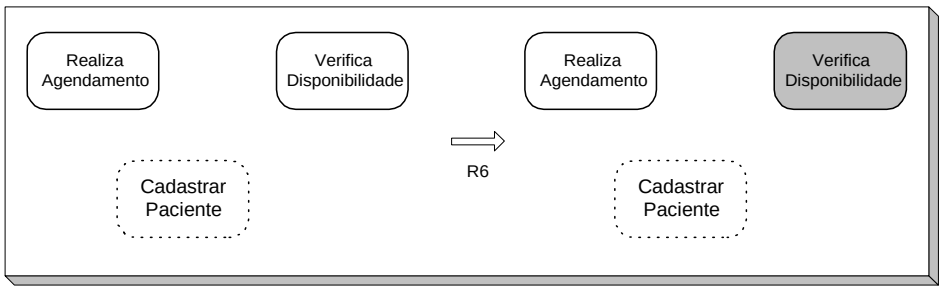

FIGURA 5.9 - Regra 6, habilitação do processo *Verifica Disponibilidade*.

g) Regra 7. Definição da ordem de execução (Habilitação do processo *Verifica Disponibilidade).*

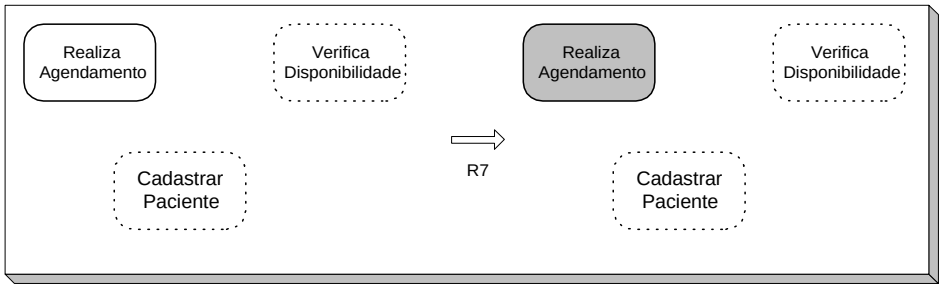

FIGURA 5.10 - Regra 7, habilitação do processo *Realiza Agendamento*.

Foi dada preferência pela utilização da habilitação de processos no nível superior, de modo que todo o controle da ordem de execução seja independente do nível inferior (expansão) do processo.

h) Regra 8. Atualização da Entidade *Agenda* a partir da execução do processo *Realiza Agendamento*.

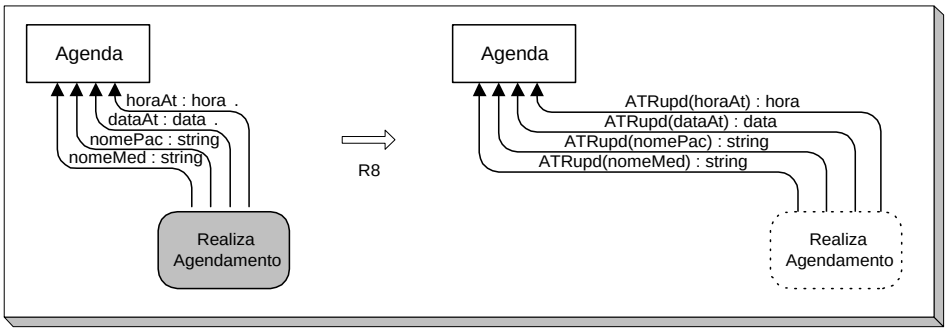

FIGURA 5.11 - Regra 8, execução do processo *Realiza Agendamento*.

i) Regra 9. Desabilitação do processo *Realiza Agendamento*.

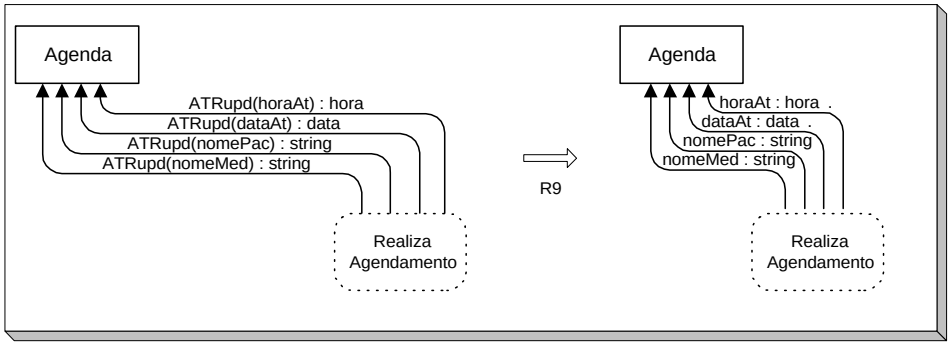

FIGURA 5.12 - Desabilitação do processo *Realiza Agendamento*.

j) Regra 9. Compressão do Processo *Agendar Consulta*.

A compressão do processo ocorre quando for concluída a execução de todos os seus subprocessos. No caso atual, o processo *Agendar Consulta* volta a estar habilitado após a execução de sua expansão.

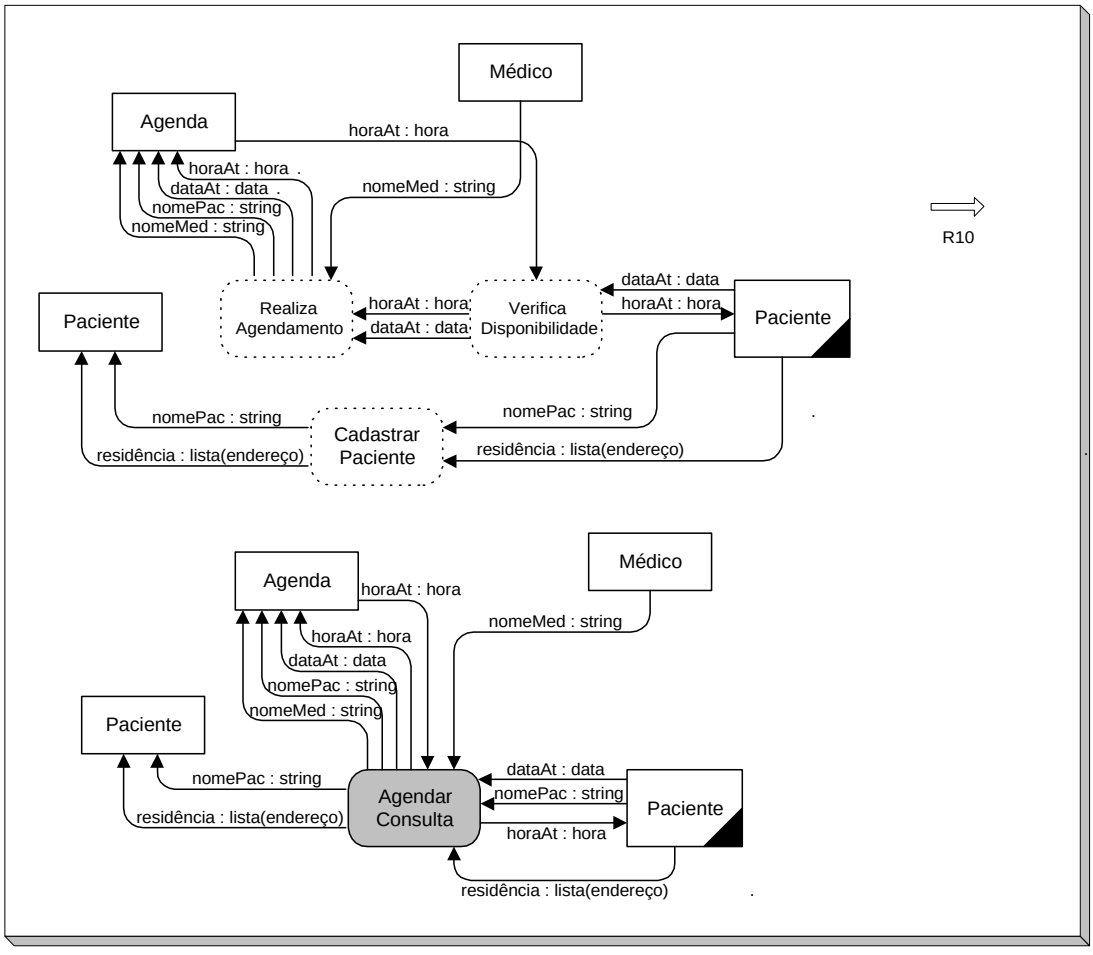

FIGURA 5.13 - Compressão do processo *Agendar Consulta*.

6 Análise comparativa entre as abordagens semi-formais de desenvolvimento e a modelagem através de sistemas de rescrita de grafos.

Neste capítulo será apresentada uma análise comparativa entre as características dos principais métodos de desenvolvimento e a abordagem de integração do modelo de dados e modelo funcional através de grafos.

Inicialmente será feita uma descrição dos princípios que regem a modelagem através de sistemas de rescrita de grafos, logo após serão analisadas as características dos principais modelos semi-formais que procuram gerar o mesmo tipo (ou parte) da solução que o modelo proposto neste trabalho. Para cada modelo serão definidos os seus princípios de modelagem, principais notações utilizadas e as vantagens do modelo proposto sobre as alternativas existentes.

Existe uma infinidade de modelos que pretendem criar representações de sistemas de modo a formalizar o seu conceito abstrato[2]. Dentre esses, foram selecionados a Modelagem de Dados, pela sua dissiminação e grande capacidade de representação de estruturas de dados[HEU 00]. A Análise Estruturada Moderna foi escolhida também por sua disseminação, principalmente devido ao fato de ter sido uma das primeiras metodologias consistentes a serem criadas, e pelo conjunto de ferramentas que proporcionam uma forte representação modular em sistemas estruturados. A Orientação a Objetos foi escolhida por proporcionar uma ampla representação de sistema integrando a modelagem de dados e funcional.

6.1 Princípios da modelagem de dados e funcional através de gramática de grafos

A forma mais genérica de descrever um diagrama formado por símbolos que se relacionam é através de um grafo. Um grafo, que é formado por arestas e mapeamentos, consegue prover um formalismo para a representação de componentes que se relacionam ou interagem em um sistema, comuns em métodos semi-formais de especificação.

Para Roger Pressman, qualquer método cuja sintaxe e semântica formal não é descrita para especificar a função e o comportamento do sistema é chamado de método informal de especificação [PRE 95]. No entanto, neste trabalho optou-se por tratar estes métodos por semi-formais, visto que existe uma preocupação com a sintaxe dos métodos, que garante uma parcial compreensão de sua representação, embora não permitam uma total compreensão das regras internas que o regem. Outro elemento que motiva esta classificação é a notação limitada que é utilizada que, embora com sérias limitações, permite uma substancial redução na ambigüidade da especificação, permitindo que o analista de sistemas e o usuário possuam uma compreensão bastante aproximada da realidade do que será o sistema após implementado.

[2] Cerca de uma dúzia de modelos foram definidos apenas por James Martin e Carma McClure em [MAR 91]. A Unified Modeling Language propõe outros 12 modelos [HAR 99].

Em geral, os métodos de especificação de sistemas baseiam-se em componentes que podem ser dados, ações, estados, atores, estruturas, tipos abstratos, dentre outros. Esses componentes podem facilmente ser representados em um grafo sob forma de vértices com seus devidos rótulos. Esses componentes estão ligados através de relações, que podem significar transições, fluxos de dados, conexões, ocorrências, regras etc.

Componentes e relações são a base de um grafo. Agregando a isso a possibilidade de associar rótulos aos vértices e atributos tipados aos diversos componentes existentes no modelo, têm-se uma poderosa representação que, com sua base formal, proporciona uma especificação sem ambigüidade e uma possibilidade de prova formal de um sistema.

Essa flexibilidade apresentada pelos grafos permite inclusive a definição de modelos que inicialmente parecem incompatíveis, como o modelo de dados e o modelo funcional. É possível definir uma única semântica de modo a diminuir a necessidade de compor a modelagem de um sistema com diversos métodos de semânticas distintas, como apresentado tanto nos métodos orientados a objetos quanto na análise estruturada.

Basicamente um sistema deve possuir três modelos para garantir a sua total representação: modelo de dados, modelo funcional e modelo dinâmico [RUM 94] [YOU 91][LAR 00][COL 96]. Em geral as metodologias de desenvolvimento necessitam de diversos diagramas para compor uma imagem abrangente do sistema.

6.2 Modelagem através de Gramática de Grafos x Modelagem de Dados

6.2.1 Princípios da modelagem de dados

Um modelo de dados é uma descrição dos tipos de informações que estão armazenadas em um banco de dados [HEU 00]. Chris Date define como um sistema de manutenção de registros por computador, cuja principal função é manter informações e torná-las disponíveis no computador [DAT 91]. Em geral é descrito por um conjunto de representações diagramáticas e textuais cujo principal componente é o Diagrama Entidade-Relacionamento (DER). Basicamente ele apresenta os componentes estáticos de informações de um sistema, suas relações e propriedades. Seus componentes básicos são: Entidades, relacionamentos com suas cardinalidades, atributos e estruturas de generalização/especialização.

Do ponto de vista de grafos, um DER é um grafo rotulado não dirigido e com atributos, referentes aos componentes que estão presentes no grafo.

A relação gráfica entre um grafo e um DER (apresentado na fig. 5.2) pode ser definida da seguinte forma: Cada entidade equivale a um vértice rotulado com atributo referente a seu nome. Os atributos do modelo ER podem ser representados através de arestas que mapeiam uma entidade para um determinado tipo. Os relacionamentos são também representados por vértices rotulados e com um atributo nome. As linhas que unem as entidades a um relacionamento também são arestas (mapeamentos) e podem possuir a cardinalidade como atributo. A estrutura de Generalização/Especialização é associada aos vértices rotulados como entidades através de arestas sem atributos.

A aplicação da gramática de grafos neste caso se dá através da transição de estados que ocorre no sistema em função de seu processo de atribuição de valores. Esta transição de estados pode ocorrer tanto observando o sistema como um todo quanto a partir de uma parte específica.

Date continua apresentando três principais aspectos relacionados ao modelo relacional: *Estruturas de dados*, compreendendo domínios e relações *n*-árias, *Integridade dos dados*, abrangendo valores não nulos, equivalência entre chaves e *Manipulação de dados*, incluindo álgebra relacional e atribuição relacional.

Quaisquer dos três aspectos podem ser representados utilizando-se grafos.

Existem vários métodos formais que utilizam uma notação matemática para realizar uma especificação de sistema baseado em suas estruturas de dados. Dentre eles podemos destacar os métodos formais Z e VDM, ambos baseados na teoria dos conjuntos [HAR 96][DIL 94].

6.2.2 Principais notações para geração de modelos de dado

Existem dois conjuntos de notações que são comumente utilizadas para representar um diagrama ER. A notação de James Martin e a notação de Peter Chen.

A notação de James Martin [MAR 91] dificulta a representação de relacionamentos com atributos, conforme pode ser visto na fig. 6.1.

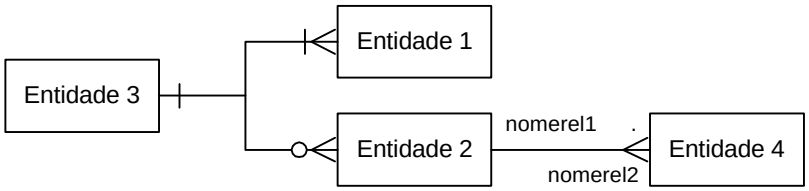

FIGURA 6.1 - Exemplo da notação de James Martin para diagrama ER.

Já a notação de Peter Chen, [CHE 76] que pode ser vista na fig. 6.2, é mais apropriada para este tipo de representação, sendo a mais utilizada.

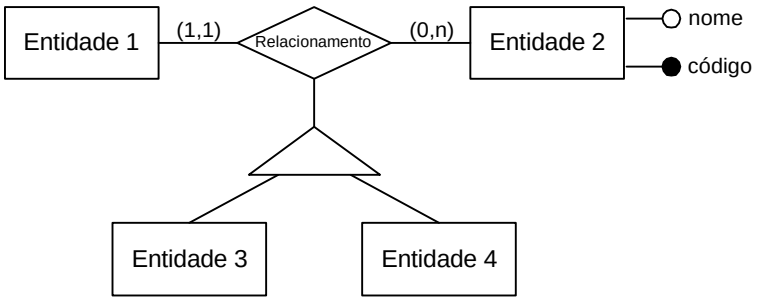

FIGURA 6.2 - Exemplo da notação de Peter Chen para diagrama ER.

A principal diferença entre as duas representações está na adaptabilidade em função do Sistema de Gerenciamento de Banco de Dados (SGDB) a ser utilizado para implementação. O modelo ER de Chen, embora aplicável nos dois casos, é mais próprio para banco de dados que consideram um relacionamento como uma estrutura armazenável da realidade, ex.: ZIM. O modelo de James Martin é mais apropriado em SGBD que gerenciam o relacionamento entre entidades através de informações contidas no Dicionário de Dados, ex.: Oracle.

6.2.3 Vantagens da modelagem através de gramática de grafos

a) Formalismo

Embora a utilização de DER no processo de modelagem de sistemas promova uma compreensão, por parte do usuário e da equipe de desenvolvimento, da estrutura do sistema, ele não é uma ferramenta própria para representar todos os aspectos de

integridade necessários ao bom funcionamento do sistema, principalmente aqueles que transcendem à álgebra relacional.

A possibilidade de representar regras que determinam o comportamento possível ao sistema provê às gramáticas de grafos a capacidade de garantir, ao final de cada transação, que o sistema mantém a sua *integridade*.

Sendo baseado em um formalismo matemático, a modelagem através de gramáticas de grafos garante também a possibilidade de serem aplicadas *provas formais* sobre o sistema, permitindo que erros de especificação sejam descobertos antes mesmo que se inicie a sua construção.

b) Unidade Semântica

A modelagem de dados, por não abranger uma gama muito grande de modelos, mantém uma unidade semântica que facilita o seu processo de representação. No entanto o fato do DER não se preocupar com a representação dos processo que transformam as informações, faz com que seja necessária a utilização de outros diagramas que realizam esta função, proposta defendida pelos autores que criaram a Análise Estruturada [GAN 83][YOU 91].

c) Comunicabilidade

Essa mesma representação das regras que regem um sistema proporciona ao usuários e equipe de desenvolvimento uma *especificação total e não ambígua* do sistema.

Também o fato do DER ser uma diagrama que representa a estrutura estática de um sistema não facilita a compreensão de como o sistema se comportará ao longo do tempo. Para resolver esse problema, freqüentemente os analista de sistemas fazem uso de algum tipo de diagrama de transição de estados.

Os analistas alegam que a representação de um sistema através de grafos dificulta a compreensão por parte dos usuários, problema este que pode ser solucionado utilizando-se grafos rotulados, que proporcionam uma notação de fácil compreensão.

A utilização de grafos permite a *abstração* de aspectos que não são relevantes em determinados momentos da modelagem, como por exemplo pode-se utilizar apenas o grafo para representar a estrutura estática do sistema, ou as suas transições de estados ou mesmo o seu processo de transformação das informações.

6.3 Modelagem através de gramática de grafos x Análise Estruturada

6.3.1 Princípios da Análise Estruturada.

O nome “Análise Estrutura” foi popularizado por Tom DeMarco [DEM 89] quando introduziu e nomeou símbolos gráficos que possibilitam ao analista criar modelos de fluxo de informações, um dicionário de dados e narrativas de processamento. Algumas extensões foram criadas por outros autores, como Chris Gane e Trish Sarson [GAN 85]. No entanto apenas no meio da década de 80 foi proposto um modelo que representasse o modelo comportamental e de controle de um sistema [WAR 85] apud [PRE 95].

A análise estruturada se propõe a representar um sistema através de seu fluxo de informações, enfatizando o processo de geração, transformação e armazenamento das informações de um sistema. O principal diagrama que procura dar suporte a esta abordagem é o Diagrama de Fluxo de Dados (DFD), embora, para representar a estrutura modular de um sistema, seja também utilizado um Diagrama de Estrutura (DE) e um Diagrama de Transição de Estados (DTE) representando o dinamismo do sistema.

O tempo demonstrou que apenas esta abordagem não era suficiente para manter uma representação de um sistema, principalmente pela instabilidade de modelos baseados em processos, que rapidamente eram substituídos em um empresa.

Alguns autores procuram suprir esta deficiência utilizando o diagrama ER como apoio ao processo de representação do sistema. [YOU 89][YOU 91], sem, no entanto, conseguir manter uma relação formal entre as duas abordagens. A falta desta relação fez com que os analistas de sistemas optasses por manter o DER e abandonar o DFD, pela sua difícil manutenção.

Este abandono teve o seu preço, que foi a diminuição do volume de informações a respeito do domínio do problema, principalmente as informações dinâmicas.

6.3.2 Principais notações para geração de um DFD.

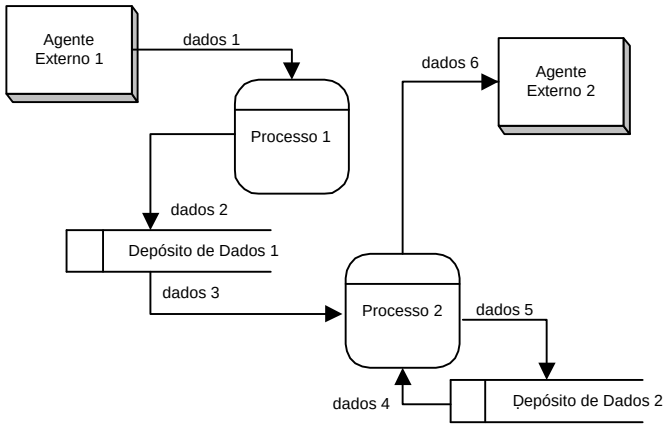

FIGURA 6.3 - Notação de DFD de Chris Gane.

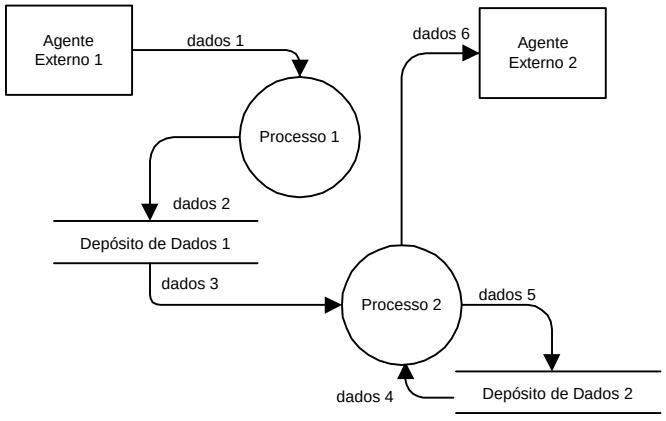

FIGURA 6.4 - Notação de DFD de Tom DeMarco.

Em ambas notações, observa-se novamente um conjunto de símbolos com rótulos que se relacionam através de arcos com atributos, conceitos facilmente representáveis em um grafo.

Estes modelos poderiam ser representados sob forma de um grafo dirigido, rotulado e com atributos, através de um conjunto de vértices (representando os agentes externos, processos e depósitos de dados) e por arestas representando os mapeamentos entre esses componentes.

Na verdade apenas o conjunto de rótulos seria substituído, caso se desejasse alterar a notação utilizada.

Diferente do modelo ER, a modelagem através do DFD permite a observação das ações que transformam o sistema, representadas pelos processos. Estes processos podem estar associados a outros processos e informações de nível mais baixo. O detalhamento da estrutura interna de um processo chama-se *expansão*.

6.3.3 Vantagens da modelagem através de gramática de grafos

a) Formalismo

O Diagrama de Fluxo de Dados e o Diagrama de Estrutura, ao contrário do DER que é baseado na álgebra relacional não possuem qualquer fundamentação formal, além da definição sintática. São apenas representações gráficas geradas a partir das experiências de analistas de sistemas com o intuito de representar o tráfego de informações no sistema e a sua estrutura modular. Esta ausência de formalismo faz com que toda a responsabilidade sobre a qualidade da especificação fique nas mãos das pessoas que gerarão modelo.

Embora a relação entre as ações seja representada, não há como realizar qualquer tipo de prova a respeito do estado final do modelo após a realização de qualquer transação. A integração do modelo de dados com o modelo funcional (proposta por este trabalho) garante a verificação de ambos modelos em qualquer ponto do sistema.

b) Unidade Semântica

Os diversos diagramas propostos na Análise Estruturada não possuem compromisso semântico. As poucas tentativas de relacioná-los (como a análise de transação e transformação para transformar DFDs em DEs) tem se mostrado fracas e, salvo raras exceções, necessitam de ajustes no produto final.

O relacionamento entre o modelo funcional e o modelo de dados é baseado nos componentes *depósitos de dados* e *entidades*, não havendo compromisso com a álgebra relacional, tampouco com a relação entre os tipos dos dados que trafegam no sistema e aqueles que são armazenados.

Relação mais difícil ainda é a existente entre o Diagrama de Transição de Estados como DFD e DER. O que é facilmente resolvido na aplicação de modelos baseados em gramáticas de grafos pela sua natural tendência de representar transições entre estados do sistema.

Essa diversidade semântica citada acima foi um dos principais elementos que motivaram o crescimento do paradigma Orientado a Objetos.

c) Comunicabilidade

Embora a Análise Estruturada possua sérios problemas em termos de formalismo, a sua representação de ações que ocorrem no dia-a-dia do usuário faz dele um diagrama de fácil compreensão por parte deste. Embora a impossibilidade de acompanhar os diversos estados gerados na base de dados faça dele um modelo que facilmente pode corromper a integridade do banco de dados. Sua falta de formalismo faz também com que um modelo gerado possa ser mal compreendido pelo usuário, o que não ocorreria em um modelo baseado em grafos.

6.4 Modelagem através de gramática de grafos x Orientação a Objetos

6.4.1 Princípios da modelagem Orientada a Objetos

O modelagem orientada a objetos surgiu a partir de meados da década de 80 [BOO 86], tendo se popularizado firmemente no início da década de 90 quando seus principais autores escreveram os primeiros livros a seu respeito [BOO 90][COA 90] [RUM 91].

Um modelo Orientado a Objetos observa a realidade a partir de um conjunto de elementos a respeito dos quais se deseja obter informações e as operações que podem ser realizadas sobre esses elementos.

O seu diagrama mais importante é o diagrama que representa as classes, objetos, estruturas e associações entre objetos, que, neste trabalho, será chamado de Diagrama de Classe&Objeto de acordo com Coad [COA 90].

Este diagrama representa as informações estáticas existentes no sistema (semelhante ao DER) com as ações que transformam essas informações. Esta representação determina uma forte coesão e um encapsulamento que aumenta significativamente a reusabilidade e o controle sobre um sistema.

Um Modelo de Objetos, pode ser observado com um grafo não dirigido, rotulado, com atributos em que os vértices correspondem a *objetos* que possuem associado um conjunto de *atributos* e um conjunto de *serviços* através de funções que mapeamentos (arestas).

Os vértices que representam *objetos* relacionam-se entre si através de funções de *associações* e *mensagens* e com vértices que representam estruturas, como *agregação* e *generalização*.

6.4.2 Principais notações para geração de modelos orientados a objetos.

A seguir observa-se duas das diversas notações utilizadas para representar um modelo de objetos. A fig. 6.5 representa um exemplo utilizando a notação da Unified Modelling Language (UML) [HAR 99].

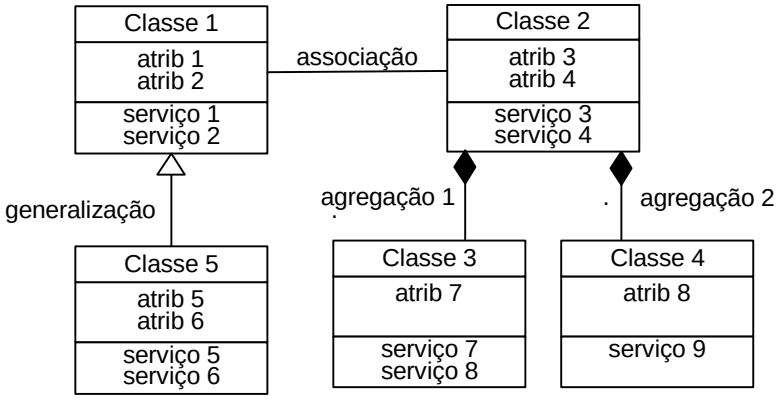

FIGURA 6.5 - Notação segundo a UML

Já a fig. 6.6 apresenta um exemplo de modelo de objetos utilizando-se a notação de Peter Coad e Edward Yourdon [COA 90].

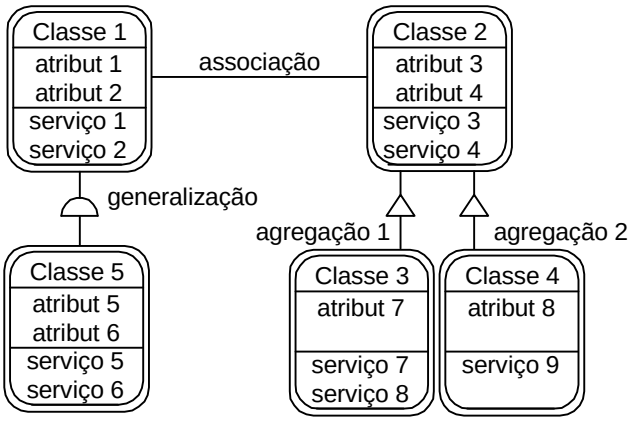

FIGURA 6.6 - Notação segundo a Coad e Yourdon

6.4.3 Vantagens da modelagem através de gramática de grafos

a) Formalismo

O Diagrama de Classe&Objeto, como apresentado, possui uma série de características que associam o seu significado ao diagrama ER. Generalização (com herança) e associações são conceitos já tratados pela álgebra relacional. Assim, sofre pela falta da capacidade de representação de regras de integridade que transcendem a álgebra relacional, mesmo problema do diagrama ER.

Ainda neste caso poderia ser utilizada a gramática de grafos para prover *integridade* e possibilidade de provas formais ao modelo do sistema.

b) Unidade Semântica

Também, a sua incapacidade de representar transições entre estados e atividades faz com que as metodologias de modelagem orientadas a objetos proponham uma série de outros diagramas para compensar suas deficiências.

Com isso o analista necessita conhecer uma série de diagramas com notações e significados distintos para gerar um modelo completo do sistema. Boa parte deles poderiam ser substituídos por um modelo unificado baseado em grafos.

c) Comunicabilidade

O grande número de diagramas necessários para a modelagem orientada a objetos confunde também o usuário que necessita compreendê-los. Embora essa diversidade proporcione um potencial de representação bastante significativo, ela ocorre principalmente pelo alto grau de especialização dos modelos, o que poderia ser resolvido utilizando-se um número reduzido de modelos que possam abstrair aspectos relacionados ao sistema.

7 Conclusão

Este trabalho teve a pretensão de apresentar uma proposta de integração de modelos através de uma notação baseada em grafos e com comportamento ditado por um sistema de rescrita de grafos.

Do ponto de vista do formalismo, o modelo demonstrou uma grande capacidade de relacionar as estruturas de dados com as estruturas funcionais e dinâmicas de um sistema. Uma grande preocupação que existia se referia à possibilidade de se utilizar uma única notação, de modo que o analista responsável pelo processo de especificação não necessitasse tomar conhecimento de um grande número de métodos para especificar um sistema. Após a aplicação do método proposto no estudo de caso, observou-se um grande poder de representação, utilizando apenas uma notação.

No entanto, embora com um notação simples de ser utilizada, o modelo se torna muito grande para ser compreendido como um elemento atômico. Na verdade, a modelagem através de gramáticas de grafos é bastante prática em modelos com um reduzido conjunto de regras, já em modelos da realidade com um grande conjunto de variações, a sua utilização pode se tornar bastante complexa, principalmente sem o apoio de uma ferramenta automatizada.

É importante avaliar os benefícios da utilização do formalismo das gramáticas de grafos. Sempre que se desejar especificar formalmente um sistema se arca com o ônus dessa especificação. Dentre os métodos disponíveis para especificação formal, o sistema de rescrita de grafos é um dos que causa menor desgaste no processo de modelagem. Existem outros métodos, talvez até mais apropriados para o desenvolvimento de sistemas aplicativos baseados em modelos de dados, como o Z ou o VDM [DIL 94][HAR 96], no entanto esses métodos são muito pobres em sua representação gráfica, o que aumenta a complexidade da compreensão do sistema como um todo. Outro problema encontrado nesses métodos é o fato de serem baseados em uma rígida notação matemática, o que os afasta da maior parte dos profissionais que realizam modelagem de sistemas aplicativos.

A falta da definição de um formalismo simples mantém os métodos formais de especificação acessíveis a um reduzido círculo de profissionais que possuem uma boa formação matemática.

Salienta-se a necessidade de uma ferramenta gráfica, que proporcione uma rápida compreensão do sistema pela identificação de componentes da realidade claramente distintos. Os métodos baseados em axiomas matemáticos explícitos pecam pela dificuldade em se compreender um universo maior a partir da avaliação de seus esquemas. A compreensão da realidade está sempre limitada ao esquema que se está observando

Como trabalhos futuros, faz-se necessária a geração de ferramentas automatizadas que apoiem ao processo de modelagem e prova formal. Essas ferramentas devem ser adaptáveis a vários projetos, possuir flexibilidade na representação e permitir a abstração dos aspectos matemáticos de mais baixo nível. Também é necessário que as ferramentas explorem mais a flexibilidade dos métodos em representar modelos diferentes e de integrar modelos.

Também a bibliografia disponível deve ser adaptada para que não apenas a comunidade acadêmica compreenda claramente o seu conteúdo, mas que seja compreensível pelos profissionais que não possuem forte embasamento matemático.

Isso é perfeitamente realizável pelo fato das gramáticas de grafos e sistemas de rescritas de grafos apresentarem uma teoria inicial pouco complexa.

O modelo pode ser também expandido de modo a dar suporte a esquemas orientados a objetos. De modo que também neste esquema, aspectos relacionados às transformações que o sistema sofre e as ações que o manipulam possam ser representadas em um modelo integrado. Lembrando que hoje, para representar os três aspectos importantes em um modelo computacional (dados, processos e transições de estados) as metodologias se utilizam de uma vasta gama de métodos distintos.

Espera-se, com este trabalho, ter-se atingido o objetivo de propor uma solução gráfica, matematicamente fundamentada, de fácil compreensão e passível de prova, que proporcione às equipes de desenvolvimento de sistemas um método que realmente garante a qualidade final de seus sistemas.

Apêndice A - Notação utilizada

$\mathcal{F}(S)$		o conjunto de todos os conjuntos finitos com elementos de S
$\{x_1,...,x_n\}$		o conjunto contendo os elementos de x_1 até x_n
$\varnothing$		o conjunto vazio
$\mathcal{B}(S)$		o conjunto de todos os bags finitos com elementos do conjunto S
$\mathcal{P}(S)$		o conjunto de todos os subconjunto do conjunto S
S^*		o conjunto de todas as listas finitas de elementos de S
S^+		o conjunto de todas as listas não vazias de elementos do conjunto S
$< x_1,...,x_n >$		uma lista com elementos de x_1 até x_n
$S_1 \times ... \times S_n$		o produto cartesiano entre os conjuntos S_1 e S_n
$f: S_1 \times ... \times S_n$	T	uma função total com o domínio $S_1 \times ... \times S_n$ e imagem T
$f: S_1 \times ... \times S_n$	T	uma função parcial com o domínio $S_1 \times ... \times S_n$ e imagem T
$f \mid v$		uma restrição da função f para o subconjunto V de seu domínio
$\mathcal{E}$		um conjunto infinito chamado universo de entidades

Bibliografia

[AND 96] ANDRIES, Adolf A. **Graph rewrite systems and visual database languages**. Tese de doutorado. College van Dekanen, Bélgica, 1996.

[BLO 95] BLOSTEIN, Dorothea.; FAHMY, Hoda & GRBAVEC, Ann. **Practical use of graph rewriting**. Technical Report No. 95-373, Computing and Information Science, Queen's University, Janeiro, 1995. Disponível por WWW em (http://www.qucis.queensu.ca/ TechReports/Reports/95-373.ps) (15/12/98)

[BOO 86] BOOCH, Grady. Object-Oriented development. **IEEE Transaction on Software Engineering SE-12**, 2. pp. 211-221.

[BOO 90] BOOCH, Grady. **Object-Oriented design**. Benjamin/Cummings, 1990.

[CHE 76] CHEN, Peter. The Entity Relationship Model - toward a unified view of data. **ACM transaction on database systems**, 1 : 1º de Marçco de 1976, pp9-37.

[COA 90] COAD, Peter e YOURDON, Edward. **Object-Oriented analysis**. Prentice Hall, 1990.

[COL 96] COLEMAN, Derek [et al.] **Desenvolvimento orientado a objetos - o método fusion**. Rio de Janeiro: Campus, 1996.

[DAT 91] DATE, C. J. **Introdução a sistemas de bancos de dados**. Rio de Janeiro: Campus, 1991.

[DEM 89] DeMARCO, Tom. **Análise estruturada e especificação de sistemas**. Rio de Janeiro: Campus, 1989.

[DIL 94] DILLER, Antoni. **Z, an introduction to formal methods**. West Sussex, England. John Willey & Sons. 1994.

[EHR 79] EHRIG, Hartmut. Introduction to the algebraic theory of graph grammars. 1st Graph Grammar Workshop, **Lecture notes in computer science 73**. Springer Verlag, 1979, 1-69.

[GAN 83] CHRIS, Gane & TRISH, Sarson. **Análise estruturada de sistemas**. Rio de Janeiro: LTC - Livros Técnicos e Científicos, 1983.

[HAR 96] HARRY, Andrew. **Formal Method - VDM and Z**. West Sussex, England. John Willey & Sons. 1996.

[HAR 99] HARMON, Paul & SAWYER, Brian. **UML for Visual Basic 6.0 developers**. San Francisco, CA: Morgan Kaufmann Publisher, Inc. 1999.

[HER 99] HERTZOG, Heloísa. **Sistema para gerenciamento de clínicas médicas** - TCC Análise de Sistemas. Universidade Luterana do Brasil. Torres, 1999.

[HEU 00] HEUSER, Carlos A. **Projeto de Banco de Dados**. Porto Alegre: Sagra Luzzato, 2000.

[LAR 00] LARMAN, Craig. **Utilizando UML e padrões**. Porto Alegre: Bookman, 2000.

[MAR 91] MARTIN, James & McCLURE, Carma. **Técnicas estruturadas e CASE**. São Paulo: Makron, McGraw-Hill, 1991.

[NAG 91a] NAGL, Manfred. (organizador), **The use of graph grammars in applications**, Graph Grammars and Their Applications to Computer Science, Springer-Verlag, LNCS 532, 1991, pp. 41-60.

[NAG 93] NAGL, Manfred. **Uniform-modelling in graph grammar specifications**, Graph Transformation in Computer Science, Springer-Verlag, LNCS 776, 1993, pp. 296-311.

[NAG 96] NAGL, Manfred. **Overview: introduction, classification and global approach**. Building tightly integrated software development environments: the IPSEN approach, Graph Transformation in Computer Science, Springer-Verlag, LNCS 1170, 1996, pp. 3-168.

[PRE 95] PRESSMAN, Roger. **Engenharia de *software***. São Paulo: Makron Books, 1995.

[PRE 99] PRETZ, Eduardo. **Estudo sobre métodos e ambientes de especificação para Gramáticas de Grafo**. Porto Alegre: CPGCC-UFRGS, 1999. 40p. (Trabalho Individual, 781).

[RIB 97] RIBEIRO, Leila e KORFF, Martin. **Métodos formais para especificação: Gramática de Grafos**. I Escola de Métodos Formais para Qualidade de Software. Curso de Informática da UFPel e Escola de Informática da UCPel, Pelotas, 1997.

[ROZ 97] ROZENBERG, G. **Handbook os graph grammars and computing by graph transformation, Volume 1: Foundations**. World Scientific. Singapore, 1997.

[RUM 94] RUMBAUGH, James [et al]. **Modelagem e projeto baseados em objetos**. Rio de Janeiro: Campus, 1994.

[RUM 91] RUMBAUGH, James et al. **Object-Oriented modelling and design**. Englewood Cliffs, NJ: Prentice Hall, 1991.

[SCH 95] SCHÜRR Andy, WINTER Andreas J., ZÜNDORF Albert. **Graph Grammar Engineering with PROGRES**, Graph Grammars and Their Applications to Computer Science, Springer, LNCS 989, 1995, pp. 219-234. Disponível por WWW em (ftp://ftp-i3.informatik.rwthaachen.de/ pub/reports/SWZ95c. ps.gz)

[WAR 85] WARD, P. T. & MELLOR, S. J. **Structured development for Real-Time Systems**. Yourdon Press, 1985.

[YOU 91] YOURDON, Edward. **Análise estruturada moderna.** Rio de Janeiro: Campus, 1991.