Gramáticas de Grafos e Objetos para Suporte a Notações Diagramáticas em Ambientes de Desenvolvimento de Software

Elisiane Andreatta Ferreira De Macedo *,+

Ana Maria De Alencar Price **,+

Resumo

A especificação de editores dirigidos por sintaxe, para notações diagramáticas de metodologias de desenvolvimento de software é abordada do ponto de vista de que um diagrama é um grafo e que as operações possíveis sobre esse grafo são determinadas através das produções de uma gramática de grafos. Os elementos léxicos são definidos sobre um framework orientado a objetos, sendo possível determinar regras de layout através de equações matemáticas, que regem o comportamento dos mesmos.

Abstract

The specification of syntax directed editors for diagrams of software development methodologies are made from the point of view that a diagram is a graph, and that operations on this graph can be specified by graph grammars productions. Lexical elements are defined from a object oriented framework, with layout rules determined by mathematical equations that manage the behavior of them.

Palavras-chave: Notações Diagramáticas, Linguagens Visuais, Gramáticas de Grafos, Framework Orientado a Objetos, Editores dirigidos por sintaxe.

* Mestranda em Ciências da Computação, UFRGS-CPGCC. E-mail: lise@inf.ufrgs.br

** Doutora em Ciências da Computação (Sussex-Inglaterra); Professora Adjunta do Instituto de Informática - UFRGS; Professora/Pesquisadora do CPGCC/UFRGS. E-mail: anaprice@inf.ufrgs.br

+ Instituto de Informática - UFRGS - Curso de Pós-Graduação em Ciências da Computação, Caixa Postal nº 15064, Porto Alegre, RS.

1. Introdução

Ambientes de desenvolvimento de software (ADSs) permitem o desenvolvimento de novos artefatos de software seguindo uma ou mais metodologias de desenvolvimento (MDS). Os ADSs podem dar suporte a todo o processo de desenvolvimento ou então oferecer ferramentas de auxílio para algumas etapas desse processo.

Com o freqüente surgimento de novas metodologias ou de modificações nas metodologias em uso, faz-se necessária a possibilidade de estender os ADSs (MetaADS) de maneira a dar suporte a essas modificações. Os ADSs que permitem esse tipo de extensão são denominados de MetaADS e a extensão pode se dar de duas formas: a primeira e mais simples é adquirir um produto disponível no mercado que consiga interfacear com o restante das ferramentas do ambiente; outra é que o próprio ADS possibilite que o administrador projete novas ferramentas, através, por exemplo, de um editor de especificação baseado em gramáticas. A primeira solução apesar de mais simples pode ser mais difícil de ser posta em prática, uma vez que, o produto desejado pode não estar disponível, estar disponível mas não possuir uma boa integração (dados, interface, controle e operacional) com o restante do ADS, além do fator relativo ao custo monetário de aquisição de um novo software. A segunda solução não possui os problemas de integração, de custos monetários e de não estar disponível, no entanto, a concepção de uma forma de especificação suficientemente genérica, simples de ser utilizada e completa ainda requer estudos profundos. Essa forma de especificação deve se preocupar em contemplar não somente a especificação de novas ferramentas, mas também a definição do próprio processo de desenvolvimento, ou seja, o relacionamento da nova ferramenta com as existentes.

As MDSs utilizam amplamente notações gráficas como recurso para melhor entender a realidade a ser modelada. Assim, um uso comum dos MetaADSs é a definição de uma nova notação diagramática. As notações diagramáticas são linguagens visuais e portanto a definição de uma nova notação não é um processo trivial. A construção de um gerador de editores diagramáticos para a geração de novos editores (para essas notações) se apresenta como uma solução para o problema da definição de novas notações para o ADS. O gerador necessita de uma forma de especificação que defina completamente a nova notação.

O artigo apresenta uma forma de especificação que alia as características de um modelo formal com grande poder de expressividade, gramáticas de grafos, com conceitos de orientação a objetos. O gerador de editores possibilita a construção de um novo editor a partir da

especificação dos elementos léxicos e das regras de layout que os governam e da definição, através de produções de uma gramática de grafos, das regras de sintaxe e semântica.

2. Notações Diagramáticas como Linguagens Visuais

Segundo a definição de Myers [MYE90] uma linguagem é visual quando os elementos componentes da linguagem estiverem dispostos num espaço bi-dimensional. As notações gráficas utilizadas nos ADS, como por exemplo os diagramas de fluxos de dados (DFD) e entidade relacionamento (ER) utilizados na metodologia estruturada [YOC78] são linguagens visuais segundo essa definição. Assim a especificação de uma nova notação consiste em definir uma nova linguagem visual, ou seja, determinar sua sintaxe e semântica.

Numa linguagem linear a composição dos elementos léxicos para formar uma expressão é uma cadeia, enquanto que numa linguagem visual a composição dos elementos gráficos é não linear. Segundo Golin [GOR89] os tipos de composição para linguagens visuais podem ser classificados em três grupos:

- **Por adjacência**: a composição de elementos gráficos adjacentes pode ser definida assim como *A acima de B* e a *base de A está tocando o topo de B*, ou de forma menos específica, como *A é vizinho de B*. Um exemplo prático são os diagramas estruturados [YOC78] onde um nodo pai deve estar acima dos seus nodos filhos.
- **Por conexão**: dois ou mais elementos gráficos podem estar conectados, como por exemplo, dois segmentos de linha conectados através de um ponto terminal comum ou o elemento *A* conectado com o elemento *B* através de uma seta que tem início no elemento *A* e término no elemento *B*. Um exemplo prático é o fluxo de dados presente nos diagramas de fluxo de dados que conecta um nodo a outro.
- **Por embutimento**: a composição de embutimento permite que um elemento gráfico esteja embutido dentro de outro, como por exemplo um círculo que possui um texto dentro. Um exemplo prático aparece no diagrama de objetos do Booch [BOO91], onde uma classe pode conter uma instância.

Em consequência da complexidade advinda das diferentes formas de composição de elementos, que não se restringem a simples concatenação, a constatação de que uma expressão é válida, segundo uma linguagem, é um processo complicado (análise). Para diminuir essa

complexidade, uma possibilidade é a construção de editores que só permitam a geração de expressões válidas de acordo com a linguagem (síntese).

As propostas de análise normalmente seguem o mesmo caminho que as linguagens lineares, ou seja, uma expressão completa da linguagem é analisada por um "parser" que verifica se a mesma está de acordo ou não com a linguagem. Gramáticas de atributos [GOR89 GOM93] e gramáticas posicionais [CHA89] foram utilizadas para a construção de "parsers" para linguagens visuais.

Com o uso de síntese, ao invés de análise, as expressões são geradas de forma que a sintaxe vai sendo avaliada a cada interação do usuário de forma a garantir que a expressão está sendo sintetizada de acordo com a linguagem desejada. O Sinthetizer Generator [RET89] gera editores de linguagens de programação lineares dirigidos pela sintaxe.

Os editores criados a partir do gerador diagramático proposto são dirigidos pela sintaxe.

3. Gramáticas de Grafos e Linguagens Visuais

Quando se trata de linguagens visuais, a exemplo das lineares [AHO86], também se pode utilizar gramáticas para a especificação de uma nova linguagem. As linguagens visuais, compostas de elementos bi-dimensionais, requerem uma teoria de linguagens que permita a definição desses elementos e dos diferentes tipos de relacionamentos existentes entre eles (adjacência, embutimento, etc.). As gramáticas de grafos podem acomodar estruturas mais gerais que "strings" e portanto são um formalismo de representação que se adequa às linguagens visuais.

Uma expressão (figura) de uma linguagem visual pode ser descrita como um grafo, ou seja, cada elemento gráfico da expressão é um nodo do grafo e as arestas do grafo correspondem aos diversos relacionamentos entre estes elementos. Os nodos e as arestas do grafo são rotulados de tal forma que possam representar diferentes elementos gráficos e diferentes tipos de relacionamentos.

Utilizaremos o formalismo de gramática de grafos operacional de Göttler [GOT87]. Esse formalismo utiliza a notação em X para a descrição das produções da gramática. Uma produção na notação X, figura 1, é composta de quatro partes: lado esquerdo da produção, lado direito da produção, contexto obrigatório e contexto opcional (ou transformação de inserção).

O processo de aplicar uma produção em notação X consiste em localizar, no grafo hospedeiro, um padrão que coincida com o lado esquerdo da produção a ser aplicada e verificar se o contexto obrigatório existe no grafo. Caso positivo, substituir o padrão equivalente ao lado esquerdo da produção pelo lado direito da mesma. O último passo é garantir que, quando existir um contexto opcional, a transformação de inserção seja aplicada. A transformação de inserção é que determina com quais nodos as arestas antes ligadas com o padrão substituído serão agora conectadas. Arestas podem ser mantidas, adicionadas ou excluídas de acordo com o que foi determinado na transformação de inserção.

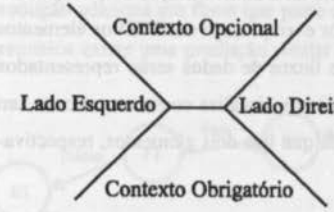

Figura 1 - Notação X

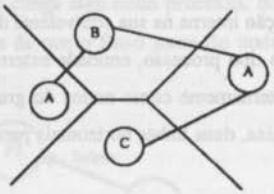

Figura 2 - Exemplo de produção

A figura 2, mostra um exemplo de uma produção de uma gramática de grafos sensível ao contexto. Para que essa produção possa ser aplicada em um grafo hospedeiro é necessário que no grafo exista um nodo do tipo **A** e um nodo do tipo **C**. Se existir, a produção será aplicável e o resultado é a criação de um relacionamento (aqui não rotulado) entre os nodos **A** e **C**. A transformação de inserção determina que se existirem um ou mais relacionamentos entre **A** e **B**, no grafo hospedeiro, estes devem ser mantidos depois da aplicação da produção.

Os atributos da gramática de grafos são equivalentes aos existentes nas linguagens lineares e representam aspectos referentes ao layout dos diagramas. Uma produção **p** de um grafo de atributos é um par **p=(X,F)**, onde **X** é uma produção de um grafo e **F** é um conjunto de fórmulas que descrevem as regras de avaliação dos atributos dos nodos de **X**.

Na aplicação de gramáticas de grafos para a geração de editores diagramáticos, o grafo hospedeiro representa a estrutura da figura, e as produções da gramática de grafos as operações que podem ser realizadas sobre a figura. Cada produção eqüivale a uma única operação do usuário do editor, no entanto operações mais complexas exigirão duas ou mais produções.

4. Visão Geral dos Editores Gerados a partir de Gramáticas de Grafos

O Sinthetizer Generator [RET89] utiliza uma representação interna, sua estrutura de dados, em forma de árvore. As possíveis operações sobre a árvore são determinadas a partir de uma gramática de atributos. De forma similar o gerador proposto se utiliza de uma representação interna na forma de um grafo e de uma gramática de grafos de atributos para as operações.

O grafo mantêm informações sobre os elementos léxicos da linguagem em questão e para que um elemento seja visualizado pelo usuário existirá uma transformação dessa representação interna na sua equivalente dentro da linguagem. Por exemplo, em um DFD os elementos do tipo processo, entidade externa, depósito de dados e fluxos de dados serão representados internamente como nodos do grafo, no entanto, o usuário os visualiza como um círculo, um caixa, duas linhas horizontais paralelas ou como uma seta que liga dois elementos, respectiva-

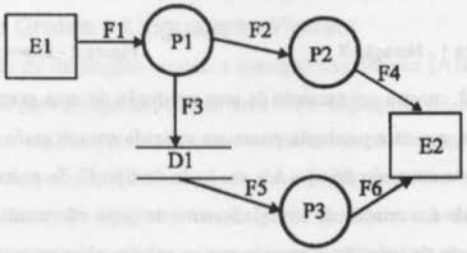

Figura 3 - Exemplo de DFD

mente. A figura 3 apresenta um DFD e na figura 4 mostra-se seu o grafo de representação. O único relacionamento existente entre nodos num DFD é o nodo início e o nodo fim de um fluxo de dados.

As operações definidas através de uma gramática de grafos agem sobre a estrutura interna dos dados que uma vez alterada deve também modificar a sua visão. Assim, uma interação do usuário no editor, é a execução de uma operação definida na gramática sobre o grafo de representação. Dessa forma o editor garante que só sejam aplicadas ações que mantenham o grafo de acordo com a sintaxe da linguagem.

Para exemplificar como as operações são definidas na produções de uma gramática de grafos, apresenta-se as figuras 5, 6 e 7 que definem parte da sintaxe de DFDs. A produção da figura 5 é responsável pela operação de incluir um novo nodo no diagrama, mantendo os relacionamentos de início e fim ligados com o nodo que foi substituído. A figura 6 mostra uma produção que transforma um nodo genérico (incluído através da aplicação da produção da figura 5) num nodo do tipo processo. Existem na gramática produções similares para transformação de nodo genérico para entidade externa ou para depósito de dados.

Finalmente a figura 7 mostra a operação de adicionar um fluxo entre dois nodos. A produção adiciona um fluxo que parte de um nodo qualquer e chega num nodo processo. Na gramática existe uma produção similar a esta com a diferença de que o fluxo parte do nodo

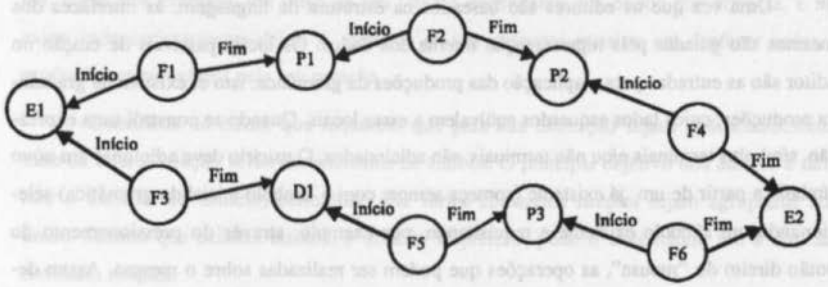

Figura 4 -Grafo de representação do DFD da figura 2

processo e chega em outro nodo. É garantida a restrição de que um fluxo deve ter numa de suas pontas um nodo do tipo processo.

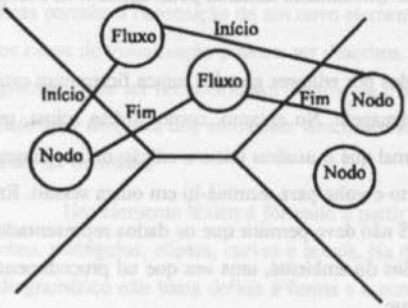

Figura 5 - Insere Nodo

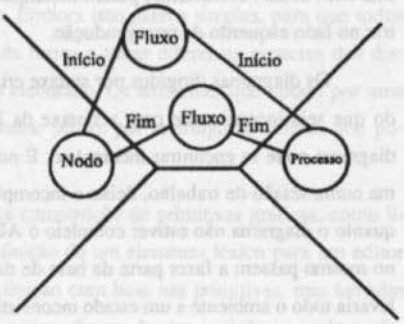

Figura 6 - Nodo genérico p/ processo

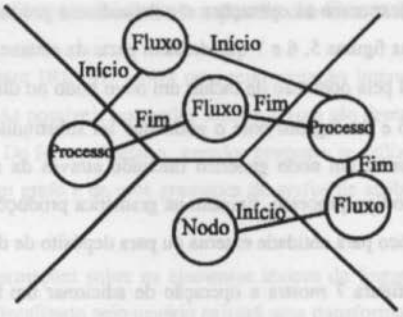

Figura 7 - Adição de fluxo de dados

Uma vez que os editores são baseados na estrutura da linguagem, as interfaces dos mesmos são guiadas pela representação interna dos dados. Os locais passíveis de edição no editor são as entradas para a aplicação das produções da gramática, isto é, existem na gramática produções, cujos lados esquerdos eqüivalem a esses locais. Quando se constrói uma expressão, símbolos terminais e/ou não terminais são adicionados. O usuário deve adicionar um novo símbolo a partir de um já existente (começa sempre com o símbolo inicial da gramática) selecionando um símbolo existente e requisitando, por exemplo, através do pressionamento do botão direito do "mouse", as operações que podem ser realizadas sobre o mesmo. Assim dependendo do contexto selecionado varia a gama de produções que pode ser aplicada.

As gramáticas de grafos não fazem uma distinção muito clara entre símbolos terminais e não terminais. Enquanto uma expressão está sendo construída aparecem no diagrama tanto não terminais como terminais, porém enquanto existirem símbolos não terminais o diagrama está num estado consistente, porém incompleto. Um símbolo terminal pode também se encontrar no lado esquerdo de uma produção.

Os diagramas dirigidos por sintaxe criados por editores gerados nunca ficam num estado que seja inconsistente com a sintaxe da linguagem. No entanto, como já dito acima, um diagrama pode se encontrar incompleto. É normal que o usuário inicie a edição de um diagrama numa sessão de trabalho, deixe-o incompleto e volte para terminá-lo em outra sessão. Enquanto o diagrama não estiver completo o ADS não deve permitir que os dados representados no mesmo passem a fazer parte da base de dados do ambiente, uma vez que tal procedimento levaria todo o ambiente a um estado inconsistente.

4.1 Interface dos Editores Gerados

Editores dirigidos por sintaxe apresentam interfaces altamente influenciadas pelas produções da gramática e só permitem a edição por estruturas inteiras e bem formadas. Assim cada operação do editor deve eqüivaler a pelo menos uma produção da gramática, não sendo possível desfazer parte de uma produção.

Basicamente existem três tipos de operações no editor: aplicar uma produção (comando simples), desfazer uma produção (comando simples), aplicar/desfazer um conjunto de produções (atalho). A aplicação de uma produção permite que novos elementos sejam adicionados ao diagrama, e para remover elementos, produções precisam ser desfeitas. Quando um novo elemento é adicionado ele pode guardar relações de dependência com outros elementos, como por exemplo, um fluxo de um DFD é dependente dos nodos que conecta, e não existe independentemente deles. A remoção de um elemento consiste em desfazer todas as produções responsáveis pela sua geração.

Comandos do editor que requerem que para sua execução sejam aplicadas/desfeitas mais de uma produção serão denominados de atalhos. O principal objetivo dos atalhos é diminuir o trabalho do usuário, permitindo que vários comandos simples sejam agrupados num único. Mesmo que existam atalhos, o usuário se preferir, pode ir executando um a um cada comando simples.

5. Visualização dos Elementos Léxicos

Na definição de uma nova forma de especificação para editores diagramáticos para ADS, um aspecto importante a ser considerado é a definição dos léxicos. A forma de especificação pode ter disponível um editor de léxicos, que a partir de um conjunto de primitivas gráficas permita a construção de um novo elemento. Embora isso pareça simples, para que todos os casos de visualização possam ser descritos, de forma a tratar diferentes aspectos dos diagramas, pode ser necessário um tratamento mais elaborado. Os ambientes que optam por uma definição simplista dos elementos léxicos geralmente pecam por restringir as formas que podem ser definidas.

Um elemento léxico é formado a partir da composição de primitivas gráficas, como linhas, retângulos, elipses, curvas e textos. Na definição de um elemento léxico para um editor diagramático não basta definir a forma e a composição com base nas primitivas, mas também deve-se descrever o comportamento de cada primitiva, dentro da composição, quando o ele-

mento léxico sofrer operações de edição por parte do usuário do editor gerado. As principais operações de edição são redimensionar e mover. A descrição do comportamento de cada componente pode ser feita através de um conjunto de restrições, definidas através do editor de léxicos.

Na definição de elementos léxicos é possível descrever relações matemáticas entre as primitivas, tais como, *é igual a*, *é a metade de*, *é duas vezes*, etc., através das restrições, de forma a estabelecer regras de layout que deverão ser mantidas quando da execução de operações de mover e redimensionar. Cruz [CRU93] propôs um modelo baseado em restrições para a definição de visualizações de objetos de banco de dados. Esse modelo não oferece suporte para edição e definição de características por parte do usuário, tais como mover e redimensionar, mas somente figuras com layout fixo. Propomos um modelo que permita alguns parâmetros em aberto para que o usuário possa mover/redimensionar uma primitiva dentro de certas restrições.

As primitivas gráficas do tipo texto podem ser variáveis ou não. As primitivas texto variáveis não têm tamanho fixo, uma vez que podem ser editadas textualmente ou então representarem, por exemplo, um campo do banco de dados do ADS. As primitivas texto não variáveis se comportam como um elemento gráfico comum e não são editadas através de operações sobre caracteres.

A operação de mover um elemento léxico compreende a descrição de como cada primitiva da composição deve se comportar. O caso mais comum é que o elemento léxico seja movido como um todo, ou seja, cada primitiva vai ser movida proporcionalmente para a nova posição de tela. No entanto é possível que alguma componente seja movida sem que as demais também o sejam. Nesse último caso, normalmente existe uma faixa ao redor do léxico na qual a primitiva pode se posicionar, assim existe um grau de dependência entre as primitivas componentes de um mesmo léxico. Na figura 8 apresenta-se um exemplo de um elemento léxico para troca de mensagens de Booch [BOO91], onde as setas representam as diferentes trocas de mensagens entre dois objetos e a linha que termina num **F** representa o relacionamento através do qual as mensagens são trocadas. O usuário do editor pode reposicionar as setas, ao redor do relacionamento, para dar a estética desejada.

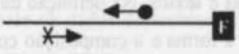

Figura 8 - Léxico p/ troca de mensagens

Quanto à operação de redimensionar é preciso definir se todos as primitivas serão redimensionadas igualmente (respeitando as relações matemáticas já especificadas) ou se algumas primitivas têm comportamento especial de se manterem sempre com um mesmo tamanho. Primitivas do tipo texto variável são modificadas através de operações sobre caracteres, tais como, inserção ou remoção de caracter, mudança de tamanho da letra, atualização do campo de dados que representam, etc., não sofrendo redimensionamento quando o léxico muda de tamanho. Primitivas do tipo texto influenciam no tamanho das demais primitivas do léxico, uma vez que, não cabendo mais dentro da área destinada, o elemento léxico poderá ser redimensionado de forma a ainda comportar o texto.

Como um diagrama é formado a partir da composição de elementos léxicos, segundo a sintaxe de uma linguagem, cada elemento léxico deve possuir um conjunto de pontos, denominados de *pontos de* relação, através dos quais o mesmo se relacionará com outros elementos. O editor de léxicos deve permitir a definição desses pontos. Um diagrama, como linguagem visual, permite que seus elementos léxicos se relacionem no espaço de três formas: por adjacência, por conexão ou por embutimento. Existindo, assim, pontos de relação de adjacência, pontos de relação de embutimento e pontos de relação de conexão. A composição de um elemento léxico com outro só pode ocorrer através de um ponto de relação.

Vários tipos de diagramas existentes nas MDS exigem regras de layout entre os léxicos, tais como, um léxico tem que estar posicionado acima de outro, um léxico deve comportar o embutimento de uma lista de léxicos, etc. É evidente que o próprio usuário do editor gerado pode se encarregar manualmente de manter os elementos léxicos dispostos coerentemente dentro do diagrama, como por exemplo ele pode primeiro aumentar o tamanho do léxico (quando necessário) que receberá um embutimento e depois realizar o embutimento de forma a sempre ter espaço para embutir outro elemento. No entanto, a especificação de regras de layout, através do editor de léxicos facilita e automatiza o trabalho de construir novos diagramas.

As regras que determinam o layout dos diagramas são escritas com base nos pontos de relação. Uma vez que os pontos de relação podem ser do tipo adjacência, embutimento e conexão, todas as possíveis situações de layout podem ser descritas. Pontos de relação de adjacência permitem a construção de regras para diagramas do tipo estruturado, árvore binária, etc., nos quais, é exigido que nodos pai se encontrem posicionados acima dos respectivos nodos filhos. Pontos de relação de embutimento permitem a construção de regras para diagramas

onde um léxico pode conter outros léxicos (embutimento) e aumentar de tamanho automaticamente toda vez que for necessário. O diagrama de objetos do Booch [BOO91] é um exemplo de diagrama que necessita de regras de layout para pontos de embutimento.

As operações de mover e redimensionar também devem ser pensadas dentro de um diagrama, do ponto de vista de léxico para léxico. Se entre dois ou mais elementos existirem relações de adjacência controladas por regras de layout, mover/redimensionar um elemento significa mover/redimensionar tanto o elemento alvo da operação como todos os outros que serão afetados por causa da relação de adjacência existente, de forma a manter o diagrama correto quanto ao layout.

Se um elemento tem em si embutidos outros elementos, movê-lo significa mover também todos os elementos contidos dentro dele. Porém mudar o seu tamanho ocorre de forma independente dos elementos embutidos, desde que seu novo tamanho ainda comporte todos os elementos embutidos. Se um elemento embutido é redimensionado de forma que não caiba mais dentro do elemento que o contém, este último deve automaticamente ser redimensionado para refletir a nova situação de layout.

É necessário definir o comportamento dos léxicos envolvidos numa relação de conexão quando ocorrer uma operação para mover ou redimensionar um dos elementos componentes dessa relação. É comum a existência de elementos léxicos nos diagramas das MDS próprios para conectar dois ou mais elementos, como é o caso do fluxo de dados nos DFD. Nesse caso mover um elemento da conexão resultará também em modificar o conector de forma que continue ligando os elementos. Não existindo elementos próprios para a conexão, a movimentação de um elemento resultará na movimentação dos pontos de relação contidos nos demais elementos envolvidos na conexão, o que poderá causar uma alteração na forma espacial desses elementos. A operação de redimensionar elementos componentes de um conexão não apresenta particularidades.

O editor de léxicos deve, portanto, fornecer formas de definir as regras de layout para os pontos de relação, assim como, o comportamento de cada primitiva componente de um elemento léxico. O modelo de restrições que será utilizado para descrever o comportamento das primitivas também pode ser estendido para comportar a definição de regras de layout.

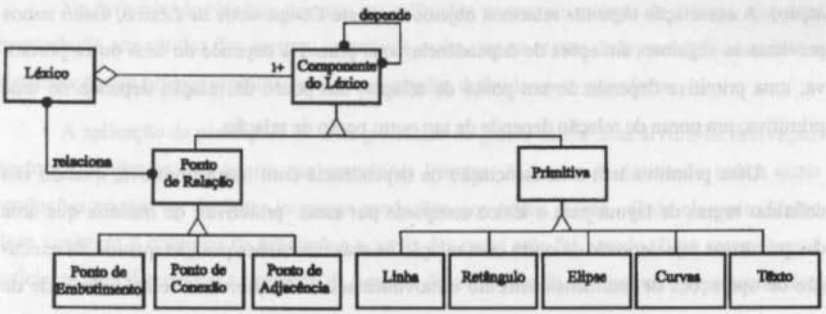

Figura 9 - Framework para definição de léxicos

6. Aspectos de Implementação

Como uma notação diagramática é uma linguagem visual, a definição de um novo editor para essa notação consiste em especificar os elementos léxicos, através de um editor de léxicos, a sintaxe, através de uma gramática de grafos, e a semântica, através dos atributos de uma gramática de grafos, da linguagem por ele representada.

6.1 Léxicos

A definição dos elementos léxicos do editor e de um conjunto de restrições, baseadas em equações matemáticas, que governam o layout dos léxicos, como discutido no tópico 5. O gerador de editores opera sobre um framework orientado a objetos para editores bi-dimensionais, padrão MVC [KRP88], responsável pela implementação dos elementos léxicos. Na figura 9 apresentamos o framework para léxicos.

Os elementos léxicos possuem pontos de relação, através dos quais, é possível relacioná-los. Pontos de relação de adjacência e de embutimento são considerados como áreas nas quais outros elementos podem estar contidos. A diferença entre os dois tipos de pontos é que o primeiro é uma área externa e o segundo é uma área interna ao elemento que detêm a dependência de outro. Outra diferença é que a área definida por um ponto de adjacência contêm apenas um elemento léxico, enquanto que um ponto de embutimento pode conter uma lista. Cada ponto de relação de conexão permite a ligação com um ou mais léxicos.

A classe *Léxico* é um agregado de componentes da classe *Componente de Léxico*. Para cada elemento léxico definido para o editor é criada uma sub-classe da classe *Léxico* que é um agregado de primitivas gráficas (classe *Primitiva)* e de pontos de relação (classe *Ponto de Re-*

lação). A associação *depende* relaciona objetos da classe *Componente de Léxico*, assim temos previstas as seguintes situações de dependência: uma primitiva depende de uma outra primitiva; uma primitiva depende de um ponto de relação; um ponto de relação depende de uma primitiva; um ponto de relação depende de um outro ponto de relação.

Uma primitiva tem uma associação de dependência com outra primitiva, quando são definidas regras de layout para o léxico composto por essas primitivas, de maneira que uma das primitivas seja depende da outra com relação às suas dimensões/posição quando da execução de operações de redimensionamento e movimentação. Um ponto de relação depende de uma primitiva ou uma primitiva depende de um ponto de relação quando forem definidas regras de layout para o diagrama que forcem a dependência entre dois ou mais léxicos.

A associação *relaciona* define que um léxico pode estar relacionado com um ou mais pontos de relação de outro léxico, sendo que esses pontos podem ser de embutimento (classe *Ponto de Embutimento*), de conexão (classe *Ponto de Conexão*) ou de adjacência (classe *Ponto de Adjacência*).

Existe, portanto, um mecanismo de dependência entre objetos componentes de um editor. Esse mecanismo pode ser implementado através de uma tabela de dependências, extraída das regras de layout. Toda vez que um objeto é movido ou redimensionado todos os seus dependentes receberão uma mensagem para recalculo dos atributos de layout, face à mudança ocorrida.

6.2 Gramática

Para a definição da sintaxe e da semântica de uma linguagem visual o gerador possui um editor de gramáticas de grafos, para notação X. A partir das produções é gerada uma representação interna, que é utilizada para determinar os comandos existentes na interface do editor gerado.

A aplicação de uma produção num grafo, ou seja, a execução de um comando no editor de diagramas, resulta na criação de uma instância para uma sub-classe de léxico quando essa produção adiciona um novo nodo no grafo; e/ou na criação de associações do tipo *relaciona* entre pontos de relação e léxicos quando é adicionada uma nova aresta. Dessa forma o grafo utilizado na representação interna será formado por instâncias de léxico e relacionamentos do tipo *relaciona*.

Na definição dos léxicos deverão ser atribuídos nomes aos pontos de relação que corresponderão aos rótulos das arestas, garantindo assim que as regras de layout definidas para a visualização dos léxicos representarão as associações definidas pela gramática.

A aplicação de produções de uma gramática de grafos forma uma árvore de derivação, de forma similar ao que ocorre nas gramáticas lineares. A árvore de derivação indica quais produções podem ser desfeitas (somente produções nos nodos folha). Cada elemento léxico deve conhecer o conjunto de produções a partir das quais este foi gerado. Esse mecanismo é suficiente para determinar a operação de funções de retrocesso da aplicação de uma produção.

O formalismo de gramáticas de grafos, devido ao seu poder de expressão e ao seu comportamento formal bem definido para geração de grafos (síntese) proporciona uma implementação relativamente simples e direta a partir de uma gramática.

7. Conclusões

A forma de especificação proposta para o gerador de editores é suficientemente genérica e permite a definição de uma grande gama de editores para notações diagramáticas. A possibilidade de especificar regras de layout para léxicos e diagramas é uma importantes contribuição para a área de geradores de editores diagramáticos.

A construção de editores de diagramas através do gerador de editores força a uniformidade de interfaces de usuário, que podem ser mais flexíveis ou mais rígidas de acordo o que for especificado nas produções da gramática de grafos.

A utilização de um modelo formal, gramáticas de grafo, para a representação da sintaxe da notação diagramática faz com que seja possível definir de forma precisa o que é um diagrama correto e o que é um diagrama incompleto. Como o ADS possui um banco de dados compartilhado por todas as ferramentas, o conhecimento através da gramática de que um diagrama está incompleto garante que a transação de adicioná-lo a essa base só será concluída quando o diagrama estiver completo.

A possibilidade de construir editores dirigidos por sintaxe com o auxílio de uma forma de especificação completa, cria uma expectativa positiva com relação a elaboração de notações mais complexas para MDS.

8. Referências

[AHO86] Aho, Alfred et ali. - *Compiler, Principles, Techniques and Tools* - Addison-Wesley. Massachusets. 1986.

[BOO91] Booch, Grady - *Object Oriented Design with Applications* - Benjamin/Cummings. 1991.

[CHA90] Chang, Shi-Kuo e Costagliola, Gennaro - *DR Parsers: a generalization of LR parsers* - IEEE Workshop on Visual Languages, pp: 174 a 180, 1990.

[CRU93] Cruz, Isabel F. - *User-Defined Visual Languages for Querying Data* - Technical Report CS-93-58, Brown University, 1993.

[GOM93] Golin, Eric J.; Magliery,Tom - *A Compiler Generator for Visual Languages* - Proceedings of IEEE Symposium on Visual Languages, IEEE Computer Society Press, August, 1993.

[GOR89] - Golin, Eric J. E Reiss, Steven P. - *Parsing in a Visual Language Environment* - Technical Report Nº CS-89-06, Brown University, February,

[GOT87] Gottler, Herbert - *Graph-Grammars and Diagram Editing* - Third International Workshop Graph-Grammars and Their Application to Computer Science, pp: 216 a 231, 1987.

[JOH91] Johnson, Ralph E.; Foot, Brian - *Designing Reusable Classes*. University of Illinois - Urbana-Champaign. 1991.

[KRP88] Krasner, G. E.; Pope, S. T. - *A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80* - Journal of Object-Oriented Program, 1(3), August-September 1988.

[MYE90] Myers, B. A. - *Taxonomies of Visual Programming and Program Visualization* - Journal of Visual Language and Computing, pp: 97 a 123, 1990.

[RET89] Reps, Thomas W.; Teitelbaum, Tim - *The Synthesizer Generator: A System For Constructing Language-Based Editors* - Springer-Verlag, New York, 1989.

[RUM91] Rumbaugh, J.; Blaha, M.; Premerlani, W.; Eddy, F.; Lorensen, W. - *Object-Oriented Modeling and Design* - Prentice Hall, Englewood Gliffs, 1991.

[WIR90] Wirfs-Brock, Rebecca J.; Johnson, Ralph E. - *Surveying Current Research in Object Oriented Design* - Communications of ACM. v.33, n.9, p. 104-124. Set, 1990.

[YOC78] Yourdon, Edward; Constantine, Larry L. - *Structured Design; Fundamentals of a Discipline of Computer Program and Systems Design* - 2ª edição, New York, Yordon Press, 1978.