

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

**Representação e Análise
de Gramáticas de Grafos**

por

DANIELA TEREZA ASCENCIO RUSSI

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

Prof^a. Dr^a. Leila Ribeiro
Orientadora

Porto Alegre, janeiro de 2003.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Russi, Daniela Tereza Ascencio

Representação e Análise de Gramáticas de Grafos / por Daniela Tereza Ascencio Russi – Porto Alegre: PPGC da UFRGS, 2003.

92p.:il.

Dissertação (Mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, 2003. Orientadora: Leila Ribeiro.

1. Gramáticas de Grafos. 2. PLATUS. 3. Representação. 4. Análise. I. Ribeiro, Leila. II. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL.

Reitora: Profa. Wrana Maria Panizzi

Pró-Reitor de Ensino: Prof. José Carlos Ferraz Hennemann

Pró-Reitora Adjunta de Pós-Graduação: Prof^ª. Jocélia Grazia

Diretor do Instituto de Informática: Prof. Philippe Oliver Alexandre Navaux

Coordenador do PPGC: Prof. Carlos Alberto Heuser

Bibliotecária-Chefe do Instituto de Informática: Beatriz Regina Bastos Haro

*Dedico este trabalho ao meu filho:
João Vitor.*

Agradecimentos

Primeiramente, agradeço a Deus por proporcionar muita luz no meu caminho e a Nossa Senhora Aparecida a quem em momentos de angústia, recorri com muita fé e confiança, e sempre encontrei luz e força.

Agradeço ao Jefferson, pelo amor e o carinho, além de toda a dedicação mesmo nos momentos difíceis.

Agradeço ao meu filho João Vitor que foi tão presente, companheiro e responsável pela força que tive nos últimos meses. Apesar de tão juntos, fui tão ausente em vários momentos, onde meu pensamento se concentrava apenas na conclusão deste trabalho.

Agradeço aos meus pais José e Alzinir que foram sempre tão preocupados e carinhosos. As minhas irmãs Gracielle e Vanessa que ajudaram nesta caminhada dando o apoio que sempre precisei.

Agradeço à minha orientadora, Prof^a. Dr^a. Leila Ribeiro, pela disposição, atenção e principalmente compreensão.

Agradeço aos meus colegas e familiares que me incentivaram e apoiaram desde o início dos créditos até a conclusão da dissertação.

Agradeço aos amigos Roberto e Eliane que tornaram a estada em Porto Alegre mais agradável tornando-se amigos dedicados e preocupados.

Agradeço à Universidade do Oeste Paulista que forneceu ajuda financeira até a conclusão deste trabalho.

Sumário

Lista de Abreviaturas.....	6
Lista de Símbolos.....	7
Lista de Figuras.....	8
Lista de Tabelas.....	10
Resumo.....	11
Abstract.....	12
1 Introdução.....	13
1.1 Motivação e Objetivos.....	14
1.2 Trabalhos relacionados.....	15
1.3 Organização do Texto.....	16
2 Gramáticas de Grafos.....	18
2.1 Sintaxe de Gramáticas de Grafos.....	18
2.2 Semântica de Gramáticas de Grafos.....	23
2.3 Gramáticas de Grafos Baseada em Objetos.....	32
3 PLATUS.....	35
3.1 Arquitetura PLATUS.....	35
3.2 Estrutura de uma Especificação do PLATUS.....	37
3.3 Tipo de Gramáticas de Grafos usada no PLATUS.....	38
3.3.1 Aspectos de tempo.....	39
3.4 Entidades.....	40
3.4.1 Interface.....	40
3.4.2 Comportamento.....	41
3.4.3 Animação e Coleta de Estatísticas.....	42
3.5 Exemplo: Sistema da Ferrovia.....	42
3.6 Modelo de simulação da ferrovia.....	46
3.7 Simulação.....	48
3.7.1 Algoritmo.....	49
4 Representação de Gramáticas de Grafos.....	52
4.1 Definição de Critérios.....	52
4.2 Representação das Estruturas.....	53
5 Análise da Recuperação das Informações.....	67
5.1 Regras de uma mensagem.....	67
5.2 Regra habilitada.....	70
5.2.1 Definição.....	70
5.2.2 Complexidade.....	76
5.3 Conflito.....	76
5.3.1 Conflito Potencial.....	77
5.3.2 Conflito Real.....	79
6 Protótipo.....	82
6.1 Implementação.....	82
6.1.1 Cadastro.....	84
6.1.2 Estudo de caso.....	89
7 Conclusão.....	90
Referências.....	91

Lista de Abreviaturas

ADT	Abstract Date Type
AGG	Attributed Graph Grammar System
DTD	Document Type Definition
GXL	Graph Exchange Language
PROGRES	Programmed Graph Rewriting Systems
XML	Extensible Markup Language

Lista de Símbolos

- $f?$ Inclusão de domínio
- $f!$ Restrição de domínio
- Σ Alfabeto de rótulos
- ε Palavra vazia
- $\#$ Em conflito
- ~~$\#$~~ Sem conflito
- \parallel Paralelismo
- \mathbb{N} Conjunto dos números naturais

Lista de Figuras

FIGURA 2.1 – Grafo.	20
FIGURA 2.2 – (a) Morfismo de grafos (b) Não é morfismo de grafos.	20
FIGURA 2.3 – Grafo tipado I^T	21
FIGURA 2.4 – Morfismo de grafo tipado f^T	21
FIGURA 2.5 – Regra.	22
FIGURA 2.6 – Grafo-tipo e Grafo inicial.	22
FIGURA 2.7 – Regras.	23
FIGURA 2.8 – Ocorrência de regras.	24
FIGURA 2.9 – Passo de derivação.	25
FIGURA 2.10 – Derivação seqüencial.	25
FIGURA 2.11 – Derivação seqüencial σ_4	26
FIGURA 2.12 – Regras aplicáveis em um grafo G	27
FIGURA 2.13 – Conflito potencial entre regras.	28
FIGURA 2.14 – Ausência de conflito potencial entre regras.	29
FIGURA 2.15 – Regras em conflito.	30
FIGURA 2.16 – Regras sem conflito.	30
FIGURA 2.17 – Paralelismo entre regras.	32
FIGURA 2.18 – Grafo-tipo baseado em objeto.	33
FIGURA 2.19 – Grafo duplamente tipado.	34
FIGURA 3.1 – Arquitetura do PLATUS.	36
FIGURA 3.2 – Modelo de um sistema de controle.	37
FIGURA 3.3 – Interface em GGa.	41
FIGURA 3.4 – Regra em GGc.	42
FIGURA 3.5 – Regra em GGa.	42
FIGURA 3.6 – Grafo-tipo da entidade <i>Train</i>	43
FIGURA 3.7 – Grafo inicial da entidade <i>Train</i>	44
FIGURA 3.8 – Conjunto de regras da entidade <i>Train</i>	44
FIGURA 3.9 – Especificação da entidade <i>Rsegm</i>	45
FIGURA 3.10 – Especificação da entidade <i>Gate</i>	45
FIGURA 3.11 – Modelo de simulação da ferrovia.	46
FIGURA 3.12 – Grafo-tipo do modelo.	47
FIGURA 3.13 – Estado do <i>Train</i> (a) Estado do <i>Rsegm</i> (b) Estado do <i>Gate</i> (c).	48
FIGURA 4.1 – Grafo-tipo da entidade <i>Gate</i>	54
FIGURA 4.2 – Instância de <i>Train</i>	66
FIGURA 5.1 – Regra r_1 do <i>Train</i>	74
FIGURA 5.2 – Regra r_2 do <i>Train</i>	74
FIGURA 5.3 – Parte do estado atual do sistema (Ferrovia).	75
FIGURA 5.4 – Regra r	75

FIGURA 5.5 – Parte do estado atual.	75
FIGURA 6.1 – Parte do grafo-tipo baseado em objeto.....	83
FIGURA 6.2 – Protótipo.....	84
FIGURA 6.3 – Menu de cadastro do protótipo.	84
FIGURA 6.4 – Cadastro de modelo.....	85
FIGURA 6.5 – Cadastro de grafos.....	85
FIGURA 6.6 – Cadastro de vértices.	86
FIGURA 6.7 – Cadastro de arcos.	86
FIGURA 6.8 – Cadastro de grafo-tipo.....	87
FIGURA 6.9 – Cadastro de instâncias.	88
FIGURA 6.10 – Cadastro de regras.....	88
FIGURA 6.11 – Estudo de caso.....	89

Lista de Tabelas

TABELA 6.1 – Especificações Técnicas do InterBase. [IBS2001].....	83
---	----

Resumo

Os sistemas computacionais estão tomando proporções cada vez maiores envolvendo situações bastante complexas, onde muitas vezes erros são inaceitáveis, como em sistemas bancários, sistemas de controle de tráfego aéreo, etc... Para obter software confiável e com desempenho aceitável, pode-se aliar técnicas de desenvolvimento formal de software a técnicas de simulação de sistemas. O ambiente PLATUS reúne essas duas áreas: modelos de simulação são descritos usando gramáticas de grafos, uma linguagem de especificação formal.

Gramáticas de grafos são uma generalização de gramáticas de *Chomsky*, substituindo *strings* por grafos. Neste trabalho, serão tratadas gramáticas de grafos baseadas em objetos, um modelo onde vértices e arcos são tipados, e as especificações são modulares (a especificação de um sistema consiste em várias gramáticas de grafos combinadas). Assim, o modelo de um sistema pode ser descrito de forma precisa, e a linguagem de especificação é bastante abstrata e expressiva.

Num ambiente de simulação a questão da recuperação de dados merece uma atenção especial, uma vez que a eficiência da ação do simulador está diretamente ligada a agilidade na obtenção das informações.

Neste trabalho, o objetivo principal é definir uma representação para gramáticas de grafos que facilite o armazenamento, a recuperação e análise das estruturas identificadas no ambiente PLATUS, ou seja, gramáticas de grafos baseadas em objetos. São definidas também funções que implementam os procedimentos necessários para a recuperação de dados durante a simulação. A eficiência dessas funções é demonstrada através do cálculo de sua ordem de complexidade. As estruturas são validadas através da implementação de um protótipo de banco de dados.

Palavras Chaves: gramáticas de grafos, PLATUS, representação, análise.

TITLE: “REPRESENTATION AND ANALYSIS OF GRAPH GRAMMARS”

Abstract

Computer systems have become increasingly complex. Moreover, a great number of such systems deal with situations where errors are unacceptable, as banking systems, systems for air traffic control, etc. To provide reliable software with acceptable performance, techniques of formal software development can be associated with techniques of system simulation. The PLATUS environment gathers these two areas: simulation models are described using graph grammars, formal specification language.

Graph grammars are a generalization of *Chomsky's* grammars, where *strings* are substituted by graphs. This thesis deals with object based graph grammars, a model where vertices and arcs are typed and the specifications are modular (the specification of a system consists of a composition of graph grammars). The model of a system can be described in a precise form, and the specification language is at a suitable level of abstractness and expressiveness.

In a simulation environment, information retrieval requires special attention, since the efficiency of the simulation is directly linked to the speed of obtaining information.

The main aim of this thesis is to define a representation for graph grammars which will optimize the storage, retrieval and analysis of the structures identified in the PLATUS environment (object based graph grammars). We also define operations to implement the necessary procedures for the recovery of data during the simulation. The efficiency of each one of these operations is demonstrated through the calculation of its complexity. The structures are validated through the implementation of a prototype of a database based on the defined structures.

Keywords: graph grammars, PLATUS, representation, analysis.

1 Introdução

Os sistemas computacionais estão tomando proporções cada vez maiores envolvendo situações de controle complexas, onde muitas vezes erros são inaceitáveis, como em sistemas bancários, sistemas de tráfego aéreo, sistemas de controle de plantas de fábricas, etc... Buscando atingir uma especificação clara e objetiva, que nem sempre é possível através do uso de uma linguagem natural, faz-se necessária, a construção de especificações cada vez mais precisas para descrever o comportamento dos sistemas, bem como possibilitar a verificação de propriedades desses sistemas e da correção de suas implementações [DEH2000]. Além da correção, que pode ser verificada se existir um processo de desenvolvimento formal do sistema existem outros requisitos que dizem respeito ao tempo de resposta e desempenho dos sistemas. Esse tipo de requisitos pode ser validado através de simulação. A simulação é um processo no qual projeta-se um modelo de um sistema real e submete-se esse modelo a uma série de experimentos. Uma das grandes vantagens da simulação é caracterizada pela possibilidade de simular um sistema mesmo em fase de projeto, através do uso de modelos que possibilitam ajustes antes mesmo da implementação. A simulação também pode servir para validar um protótipo de sistema. O fato de ser possível fazer verificação formal e simulação em um mesmo ambiente oferece ferramentas poderosas para evitar erros nas fases iniciais de projeto, que são extremamente onerosos.

Existem atualmente vários ambientes para simulação, mas quase nenhum utiliza linguagens de especificação formal como base para a construção dos modelos de simulação. O ambiente que estará sendo usado neste trabalho será o PLATUS [BAR99, COP2000, RIB99b, RIB2001], ambiente esse que permite a construção de modelos de simulação baseados em técnicas de especificação formal que usam gramáticas de grafos como ferramenta descritiva [COP2000]. O fato de serem formais e intuitivas ao mesmo tempo, e também de poderem tratar com simplicidade aspectos de concorrência e distribuição de sistemas, faz de gramáticas de grafos um método promissor para o desenvolvimento de software confiável [DEH2000]. Segundo [RIB97], o uso de gramáticas de grafos no ambiente PLATUS é vantajoso, uma vez que nesse formalismo, os estados de um sistema são descritos por grafos e o comportamento do sistema é descrito de maneira operacional através de mudanças de estados, que são modeladas pelas regras. A representação dos sistemas através de gramáticas de grafos pode ser feita através de grafos simples, que consistem apenas de arcos e vértices sem tipo, para a modelagem dos estados [COR96]. Mas, geralmente, segundo [DEH2000], a utilização de mecanismos de tipagem simplifica significativamente a descrição das especificações.

PLATUS usa gramáticas de grafos como ferramenta descritiva, mas não gramáticas de grafos usuais e sim gramáticas chamadas de gramáticas de grafos baseadas em objeto, onde a especificação ganha um estilo baseado em objetos e os grafos são referenciados como grafos “especiais”, que possuem diferentes tipos de arcos e vértices. Para armazenar as especificações descritas no ambiente PLATUS, é necessário definir uma representação concreta, usando as estruturas de dados disponíveis no banco de dados existentes. Um mapeamento direto das definições clássicas de gramáticas de grafos para estruturas de dados concretas é difícil, pois essas definições são bastante abstratas, envolvendo conceitos matemáticos complexos de teoria das categorias c . Além disso, a recuperação de informações armazenadas desta forma provavelmente não seria eficiente. Como a recuperação de dados é bastante

utilizada durante a simulação de uma especificação no PLATUS, deve-se tornar esta recuperação o mais eficiente possível.

A próxima seção apresenta a motivação e os objetivos que levaram à elaboração deste trabalho, e o enfoque adotado nesta dissertação. Na seqüência, são apresentados os trabalhos relacionados que foram estudados e que auxiliaram no entendimento e no conhecimento da representação feita através de gramática de grafos. Por fim, é apresentada a organização do texto.

1.1 Motivação e Objetivos

Neste trabalho, os objetivos estão voltados para o ambiente PLATUS. Esse ambiente é composto por ferramentas que acessam dados e trocam informações, bem como armazenam dados sobre as estruturas.

A ausência de uma padronização na representação das estruturas, nas várias ferramentas serviu como motivação para elaboração deste trabalho. Uma representação adequada deveria facilitar as análises de consistência necessárias para a construção e simulação de modelos, bem como das outras ferramentas projetadas para o ambiente. Portanto, buscando a padronização na representação das estruturas do PLATUS e visando facilitar o armazenamento, a recuperação e a análise das especificações realizadas no ambiente, por intermédio da representação, o objetivo principal deste trabalho é obter uma definição de representação concreta para as estruturas do ambiente. Para alcançar esse objetivo, será necessário definir uma representação para estruturas do ambiente PLATUS que seja fiel ao formalismo de gramáticas de grafos baseadas em objetos e eficiente para agilizar a execução do simulador. Mais especificamente, os objetivos do trabalho são:

1. definir estruturas para a representação de modelos de simulação especificados usando gramáticas de grafos baseadas em objetos;
2. definir como os dados necessários para a simulação podem ser recuperados usando esta estrutura, e mostrar a eficiência desses procedimentos;
3. implementar um protótipo de banco de dados baseado nas estruturas propostas, que possa ser utilizado no ambiente PLATUS.

Para realizar o objetivo 1, foi necessário mapear definições usuais da abordagem algébrica de gramática de grafos, que são definidas usando teoria dos conjuntos e das categorias, para estruturas concretas encontradas usualmente em linguagens de programação e de modelagem de bancos de dados. As razões da escolha das estruturas concretas serão discutidas na seção 4.1, onde se definem critérios que devem ser atendidos pela representação proposta.

O objetivo 2 envolveu, além da tradução de conceitos de categorias para a representação concreta definida no item 1, a definição da ordem de complexidade dos procedimentos necessários para realizar as análises desejadas.

O objetivo 3 permitiu validar a proposta, e serviu de base para a integração de ferramentas no ambiente PLATUS.

1.2 Trabalhos relacionados

Nesta seção resume-se aos resultados de estudos realizados a partir de ambientes relacionados ao PLATUS, visando uma maior aproximação aos objetivos desta dissertação. Dentre os ambientes é possível referenciar:

- **AGG** (*Attributed Graph Grammar System*): ambiente composto de ferramentas visuais que consiste de editor, interpretador e depurador para transformação de grafos com atributos: computação de atributos através de Java [APP97]. Segundo [ERM2001a], AGG é uma ferramenta para editar grafos dirigidos, tipados e com atributos, onde o objetivo é definir GGs como entrada para o componente de transformação de grafos do sistema de execução de passos de transformação direta, isto é usado para produções e ocorrências selecionadas pelo usuário na abordagem *single-pushout* para transformação de grafos. A interface do AGG é dirigida por menu, e apresenta quatro editores diferentes: dois editores gráficos para grafos e regras, um editor de texto para expressões Java como atributos os quais são integrados no editor gráfico, e uma janela para editar os nomes das GGs e seus grafos iniciais e regras. Inicialmente, é preciso definir tipos para os objetos dos grafos, sendo que o editor gráfico apresenta um limitado conjunto de elementos gráficos capazes de representar vértices e arcos. Vértices podem ser representados de modo oval. Arcos podem ser representados por linhas. Tanto vértices como arcos podem possuir diferentes cores. O tipo de um objeto deve ser definido antes da definição do objeto. Na seqüência, podem ser definidas as regras, mediante o detalhamento do lado esquerdo da regra que é copiado para o lado direito, onde os objetos são deletados e criados. Uma regra pode conter computações em valores de atributos, sendo que um atributo possui três componentes: um tipo, um nome e um valor. Assim, através dos editores do AGG é possível definir não apenas as regras, mas os tipos de objetos (vértices e arcos) dos grafos, bem como os atributos. A última versão do AGG contém um analisador de grafo que possibilita a análise de conflito entre um par de regras, mediante as seguintes situações:

- A regra deleta um objeto que é acessado pela outra regra.
- A regra altera um atributo que é acessado pela outra regra.

Caso uma das situações seja identificada entre um determinado par de regras, isto determina a presença de conflito entre as regras.

Atualmente, o AGG encontra-se em processo de mudança de *java object streams* para XML, sendo que o objetivo é definir um DTD (*Document Type Definition*) comum a toda ferramenta de transformação de grafo [ERM2001b].

- **PROGRES** (*Programmed Graph Rewriting Systems*): ambiente que apresenta um conjunto integrado de ferramentas para edição, análise e transformação de grafos programados, e suporta prototipação rápida de ferramentas de manipulação de grafos [APP97]. Além disso, apresenta um editor dirigido à sintaxe que mistura texto e gráfico junto com uma bancada de trabalho incremental e um editor de layout, assim como oferece uma

forma de checar o tipo, uma vez que detecta todas as inconsistências com respeito à semântica estática da linguagem PROGRES, mostrando erros por meio de mensagens. Oferece uma interface de importação/exportação para editores de texto e sistemas de processamento de texto. Segundo [SCH95], PROGRES utiliza-se de um sistema de banco de dados (GRAS), o qual foi projetado para atender as necessidades dos sistemas que são especificados no ambiente. Esses sistemas são altamente interativos, apresentam estruturas de objetos complexos, deste modo o sistema de banco de dados a ser utilizado deve ser capaz de manipular de forma eficiente diferentes tipos de objetos, relacionar objetos hierárquicos ou não hierárquicos. Além disso, deve suportar a computação incremental de dados, refazer/desfazer modificações de dados e apresentar mecanismos de controle de versão.

De uma forma geral, os estudos permitiram a identificação de algumas características que são comuns aos ambientes:

- Várias ferramentas que cooperam umas com as outras.
- Estruturas de dados complexas são modeladas por grafos.
- Vértices e os arcos são definidos como objetos dos grafos, onde cada objeto está associado a um único tipo.
- O comportamento do sistema é representado através de um conjunto de regras.
- A aplicação de uma regra transforma a estrutura do grafo, uma vez que a regra é composta de um par de grafos que representam o estado anterior e posterior a ação da regra, ou melhor, uma regra representa uma transição de estado.
- As últimas pesquisas destinam-se ao intercâmbio de informações através da *web*, mediante o uso da XML/GXL (*Graph Exchange Language*).
- Possibilitam a verificação formal de GGs.
- Permitem a realização de simulação do sistema gerado através de GGs.

Porém, nenhum destes sistemas poderia servir como base para o ambiente PLATUS, porque no PLATUS a especificação de um sistema consiste de várias gramáticas de grafos, uma para cada entidade envolvida no modelo (seguindo o estilo baseado em objetos). Além disso, os modelos envolvem, além da descrição do comportamento do sistema, informações referentes à animação e a estatísticas de simulação, dados que não estão presentes nos outros ambientes.

1.3 Organização do Texto

Após este capítulo introdutório, a presente dissertação está organizada em capítulos intitulados respectivamente como: Gramáticas de Grafos, PLATUS, Representação de Gramáticas de Grafos, Análise da Recuperação das Informações, Protótipo e Conclusão.

O capítulo 2 apresenta algumas definições básicas, envolvendo a sintaxe e a semântica de gramáticas de grafos. Por fim, é feito um detalhamento sobre gramáticas de grafos baseada em objetos.

No capítulo 3, é feita uma descrição do ambiente PLATUS, onde inicialmente, descreve-se a arquitetura do ambiente, e na seqüência são feitos detalhamentos da estrutura de especificação, do tipo de GGs utilizada no PLATUS e das entidades. Por fim, apresenta-se um exemplo de um sistema, o modelo de simulação desse sistema (ferrovia), bem como a descrição da simulação propriamente dita, através de um algoritmo.

O capítulo 4 trata da representação de gramáticas de grafos, apresenta alguns critérios que foram definidos neste trabalho, bem como toda a representação das estruturas do ambiente PLATUS.

No capítulo 5, é possível ter uma idéia quanto à recuperação das informações, mediante a representação definida e as necessidades do simulador PLATUS.

O capítulo 6 apresenta detalhes quanto à idealização e a implementação do protótipo, bem como alguns exemplos de representações e recuperações de informações.

Por fim, o capítulo 7 apresenta as conclusões desta dissertação, bem como os aspectos que ficaram em aberto como sugestão a trabalhos futuros.

2 Gramáticas de Grafos

Gramáticas de grafos são uma generalização de gramáticas de *Chomsky*, substituindo *strings* por grafos [COR97]. O uso de gramáticas de grafos como método de especificação de sistema está sendo aplicado em uma grande variedade de sistemas complexos [RIB97].

Para formalização de um problema é desejável uma maneira natural e intuitiva de descrição. O fato de serem formais e intuitivas ao mesmo tempo, e também de poderem tratar com simplicidade aspectos de concorrência e distribuição de sistemas, faz de gramáticas de grafos um método promissor para o desenvolvimento de software confiável. Grafos são um meio natural para explicar situações complexas de modo intuitivo, partindo da idéia de que grafos representam estados do sistema e que cada regra aplicada, descreve uma transição de estado.

Pesquisas na área de gramáticas de grafos iniciaram nos anos 70, e métodos, técnicas e resultados nesta área já foram estudados e aplicados em uma grande variedade de campos da Informática como: teoria e linguagens formais, reconhecimento e geração de imagens, construção de compiladores, engenharia de software, modelagem de sistemas concorrentes e distribuídos, projeto e teoria de bancos de dados, etc. [NAG91, ROZ99].

Segundo [RIB97], para definir uma gramática de grafos, basicamente deve-se encontrar respostas para as seguintes perguntas:

1. O que é um grafo?
2. Como descrever uma regra usando grafos?
3. Como encontrar uma ocorrência do lado esquerdo L de uma regra $r: L \rightarrow R$ em um grafo G ?
4. Como descrever a substituição de L em G pelo lado direito R da regra?

Respostas para as duas primeiras perguntas definem a sintaxe da linguagem utilizada, ou seja, a estrutura dos grafos que podem ser utilizados para construir uma especificação e como esses grafos podem ser modificados. Respostas para as outras questões definem a semântica da linguagem, ou melhor, como as modificações descritas pelas regras são efetuadas.

As próximas seções apresentam detalhes e exemplos sobre sintaxe e semântica de gramáticas de grafos. Por fim, é feito um detalhamento sobre gramáticas de grafos baseada em objetos.

2.1 Sintaxe de Gramáticas de Grafos

Diferente de regras em gramáticas de *Chomsky*, uma regra de grafos $r: L \rightarrow R$ não consiste somente dos grafos L (lado esquerdo) e R (lado direito), mas também de uma parte adicional: um (homo) morfismo parcial de grafos r que mapeia vértices e arcos de L em vértices e arcos de R de maneira compatível. Compatibilidade aqui significa que cada vez que um arco a_L for mapeado em um arco a_R , então o vértice

origem de a_L deve ser mapeado para o vértice origem de a_R e o mesmo para o vértice destino.

A abordagem a ser seguida é que gramáticas de grafos especificam um sistema em termos de estados – modelados por grafos (ou estruturas parecidas com grafos) – e mudanças de estados – modeladas por derivações. Regras descrevem o comportamento do sistema (mudanças de estados), onde o lado esquerdo representa a situação do sistema no qual a regra pode ser aplicada, o lado direito representa a mudança que deve ocorrer com a aplicação da regra (situação resultante da aplicação da regra). A seguinte interpretação operacional de uma regra $r: L \rightarrow R$ descreve a base desta abordagem de especificação:

- Itens em L que não tem uma imagem em R são deletados.
- Itens em L que são mapeados em R são preservados.
- Itens em R que não tem uma pré-imagem em L são criados.

A aplicação de uma regra em um grafo (estado) só é possível se há um mapeamento do lado esquerdo desta regra, i.e., existe um subgrafo do estado que corresponde ao lado esquerdo da regra. Muitas regras podem ser aplicadas em paralelo, caso imagens dos seus lados esquerdos sejam encontradas no estado atual. Caso haja conflito, i.e., duas regras tentam deletar o mesmo item, a escolha de qual será aplicada é não determinística.

Definição 2.1 (Comutatividade fraca) Seja $f: A \rightarrow B$ uma função parcial com domínio $dom(f)$ onde a função f está definida, sendo $f?: A \rightarrow dom(f)$ e $f!: dom(f) \rightarrow B$ as funções de inclusão e restrição de domínio correspondente.

$$\begin{array}{ccc}
 A & \xleftarrow{f?} & dom(f) & \xrightarrow{f!} & B \\
 a \downarrow & & = & & \downarrow b \\
 A' & \xrightarrow{f'} & & & B'
 \end{array}
 \Leftrightarrow
 \begin{array}{ccc}
 A & \xrightarrow{f} & B \\
 a \downarrow & & \geq & & \downarrow b \\
 A' & \xrightarrow{f'} & & & B'
 \end{array}$$

Dadas funções, como mostram os diagramas acima, onde a e b são totais, escreve-se $f' \circ a > b \circ f$ e diz que o diagrama comuta fracamente se $f' \circ a \circ f? = b \circ f!$.

Observação: Se f e f' são totais, então comutatividade fraca coincide com comutatividade. Note que $((f?)^{-1}, f!)$ é uma fatorização de f . A compatibilidade condicional definida acima significa que tudo que é preservado (mapeado) pelo morfismo deve ser compatível. O termo “fraca” é usado porque a compatibilidade é requerida apenas a itens preservados, e não a todos os itens.

✓

Definição 2.2 (Grafo, Morfismo de Grafo) Um grafo $G = (V_G, A_G, o^G, d^G)$ consiste de um conjunto de vértices V_G , um conjunto de arcos A_G e funções totais o^G e $d^G: A_G \rightarrow V_G$, que representam a interdependência entre arcos e vértices e que são conhecidas como função de origem e destino respectivamente. $x \in G$ denota um item $x \in V_G \cup A_G$. Um grafo é finito se V_G e A_G são finitos.

Um **morfismo (parcial) de grafo** $g: G \rightarrow H$ de um grafo G para um grafo H é uma tupla $g = (g_V, g_A)$ que consiste de duas funções parciais $g_V: V_G \rightarrow V_H$ e $g_A: A_G \rightarrow A_H$, tal que os diagramas a seguir comutam fracamente.

$$\begin{array}{ccc}
 A_G & \xrightarrow{g_A} & A_H \\
 o^G \downarrow & \geq & \downarrow o^H \\
 V_G & \xrightarrow{g_V} & V_H
 \end{array}
 \qquad
 \begin{array}{ccc}
 A_G & \xrightarrow{g_A} & A_H \\
 d^G \downarrow & \geq & \downarrow d^H \\
 V_G & \xrightarrow{g_V} & V_H
 \end{array}$$

Um **morfismo (total) de grafo** $g: G \rightarrow H$ de um grafo G para um grafo H é uma tupla $g = (g_V, g_A)$ que consiste de duas funções totais $g_V: V_G \rightarrow V_H$ e $g_A: A_G \rightarrow A_H$.

Exemplo 2.1 (Grafo, Morfismo de Grafo) O grafo da figura 2.1 é formado por $G = (V, A, o, d)$, onde $A = \{a, b\}$, $V = \{ \blacksquare, \star, \blacktriangle \}$, $o = \{ a \rightarrow \blacksquare, b \rightarrow \blacktriangle \}$ e $d = \{ b \rightarrow \blacksquare, a \rightarrow \star \}$.

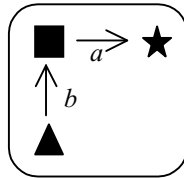


FIGURA 2.1 – Grafo.

A figura 2.2 mostra dois mapeamentos (a e b). O mapeamento (a) é um morfismo (parcial) de grafos. O mapeamento (b) não é um morfismo de grafos, uma vez que o arco b_L foi mapeado em um arco b_R e o vértice origem e/ou destino de b_L não foi mapeado de forma compatível como vértice origem e/ou destino de b_R .

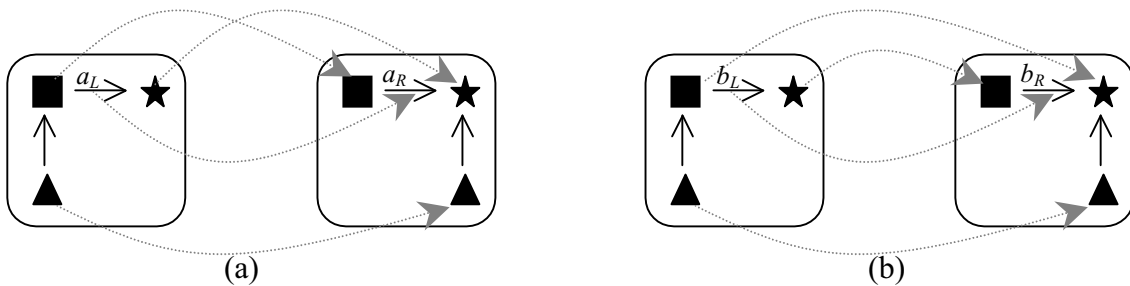
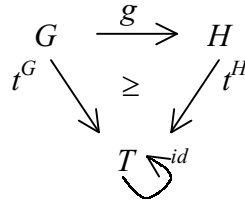


FIGURA 2.2 – (a) Morfismo de grafos (b) Não é morfismo de grafos.

Ao invés de usar grafos simples que consistem somente de arcos e vértices, geralmente usa-se algum mecanismo de tipagem nos grafos. A tipagem de grafos pode ser feita pelo uso de rótulos (onde cada arco e vértice é rotulado com um elemento de um alfabeto Σ), atributos (onde tipos abstratos de dados são usados) e grafo-tipo (onde um grafo é utilizado como tipo). O uso de mecanismos de tipagem faz as especificações se tornarem mais simples, compactas e fáceis de entender. Neste trabalho, as especificações são feitas através de um grafo denominado grafo-tipo, que especifica o tipo de todos os vértices e arcos. Assim, um grafo consiste de vértices e arcos do grafo-tipo.

Definição 2.3 (Grafo Tipado, Morfismo de Grafo Tipado) Um **grafo tipado** G^T é uma tupa $G^T = (G, t^G, T)$ onde G e T são grafos e $t^G: G \rightarrow T$ é um morfismo total de grafos. O uso $x \in G^T$ significa que x é um vértice ou um arco de G .

Um **morfismo de grafo tipado** $g^t: G^T \rightarrow H^T$ entre grafos tipados G^T e H^T é um par de morfismos de grafo $g^{id} = (g, id)$ com $g: G \rightarrow H$ e $id: T \rightarrow T$ é a identidade de T . Neste caso, o mapeamento não deve preservar apenas vértices e arcos de maneira compatível, mas também o tipo desses arcos e vértices, tal qual o diagrama a seguir.



✓

Exemplo 2.2 (Grafo Tipado, Morfismo de Grafo Tipado) Considerando o grafo T da figura 2.3 como um grafo-tipo. Então um grafo I é uma instância deste grafo-tipo, i.e., um grafo no qual possivelmente muitas ocorrências de cada tipo serão encontradas. Para descrever o relacionamento entre as instâncias e seus tipos, usa-se um morfismo total de grafos $t^I: I \rightarrow T$. Este morfismo descreve a tipagem de todos os vértices e arcos de um grafo instância, e garante que o grafo instância é consistente com o grafo-tipo. Um grafo tipado usualmente será denotado por I^T , onde I é um grafo instância tendo T como tipo.

Como grafos representam estados, é natural que mudanças de estados sejam representadas por relacionamentos entre grafos. Neste caso, é requerido que não somente vértices e arcos sejam mapeados de forma compatível, mas também que as informações de tipagem sejam preservadas. Isto significa, por exemplo, que o vértice do tipo \blacksquare não pode ser mapeado através de um morfismo para um vértice do tipo \star . A figura 2.4 representa um morfismo de grafo tipado f^t entre os grafos (tipados) I^T e I'^T . Este morfismo deleta \blacktriangle , \star e os arcos correspondentes e cria \star .

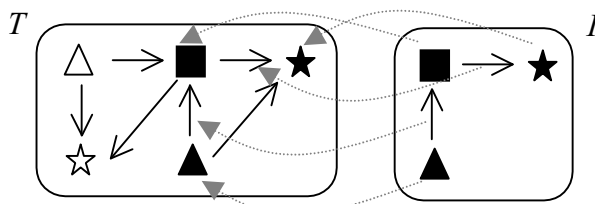


FIGURA 2.3 – Grafo tipado I^T .

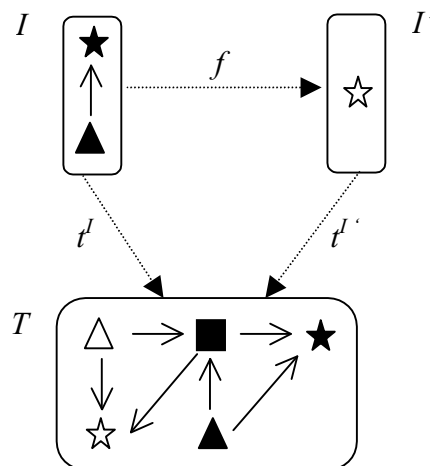


FIGURA 2.4 – Morfismo de grafo tipado f^t .

✓

Uma regra representa o comportamento do sistema, através de um morfismo entre dois grafos tipados L e R . O lado esquerdo (L) representa o estado do sistema que habilita a ação da regra e o lado direito (R) representa o estado do sistema após a execução da regra.

Definição 2.4 (Regra) Seja T um grafo-tipo. Então uma **regra** sobre T é um morfismo $r^T: L^T \rightarrow R^T$.

Notação: Para simplificar a representação gráfica das regras (morfismos) serão utilizados índices para vértices e arcos: elementos com mesmo índice (ou sem índice) representam itens preservados (mapeados).

Exemplo 2.3 (Regra) A figura 2.5 mostra uma regra $r: L \rightarrow R$, onde o arco com índice 1 foi deletado, os vértices ■, ★, ▲ e o arco sem índice foram preservados, e o arco com índice 2 foi criado. O grafo-tipo desta regra é o grafo T da figura 2.3.

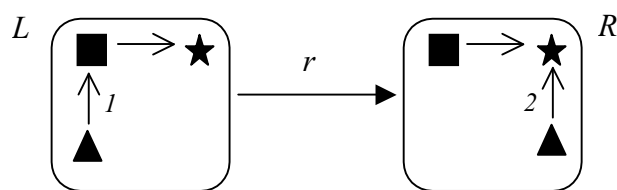


FIGURA 2.5 – Regra.

Uma gramática de grafos (tipada) consiste de um **grafo-tipo**, que especifica o tipo de todos os grafos envolvidos nesta gramática e, portanto chamado de tipo da gramática; um **grafo inicial**, que especifica o estado inicial do sistema e um **conjunto de regras**, que representa o comportamento do sistema.

Definição 2.5 (Gramática Grafos) Uma **gramática de grafos** (tipada) é uma tupla $GG=(GT, GI, N)$, onde:

- GT é um grafo-tipo (tipo da gramática),
- GI é um grafo inicial (tipado),
- N é um conjunto de regras.

Exemplo 2.4 (Gramática Grafos) As figuras 2.6 e 2.7 mostram a gramática de grafos $GG=(GT, GI, N)$.

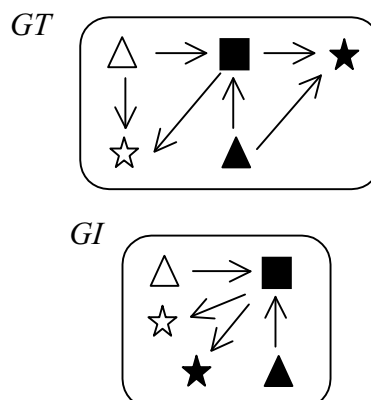


FIGURA 2.6 – Grafo-tipo e Grafo inicial.

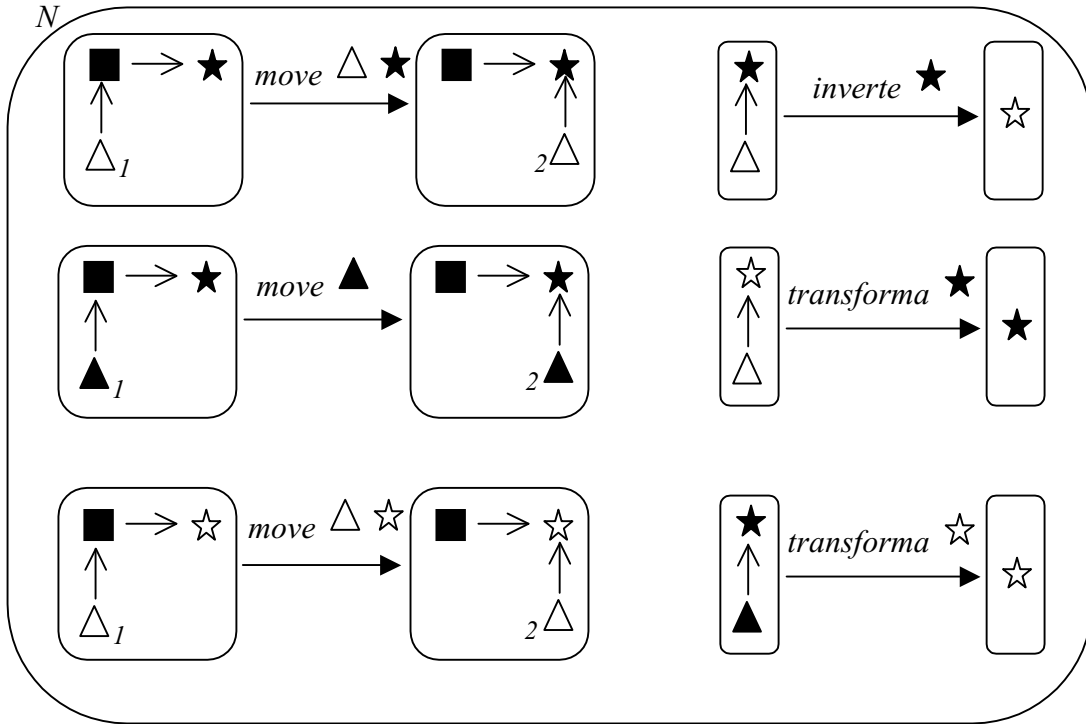


FIGURA 2.7 – Regras.

2.2 Semântica de Gramáticas de Grafos

O comportamento de uma gramática de grafos é determinado pelas aplicações das regras a grafos que representam os estados do sistema, tendo o grafo inicial como ponto de partida. A aplicação de uma regra em um estado atual do sistema só é possível se há um mapeamento para esta regra, i.e., há um subgrafo (no estado atual do sistema) que corresponde ao lado esquerdo da regra. O número de regras que podem ser identificadas como aplicáveis para um determinado estado do sistema pode ser bem diversificado. Muitas regras podem ser aplicáveis num mesmo estado do sistema e a escolha de qual regra/regras que será aplicada é não determinística. Diferentes estratégias para controlar qual/quais regras podem ser aplicadas em paralelo deram origem a diferentes modelos semânticos para GGs. Para gramáticas de grafos existem modelos semânticos baseados em computações seqüenciais e em diferentes tipos de computações concorrentes. A partir da definição do modelo é possível definir como serão identificadas as regras aplicáveis e a forma de aplicação de tais regras, visando ações que podem acontecer em paralelo e os relacionamentos possíveis entre essas ações.

Definição 2.6 (Ocorrência) Dada uma regra $r: L \rightarrow R$, e um grafo IN . Uma ocorrência de r em IN é um morfismo total de grafos tipados.

Notação: Para simplificar a representação gráfica das ocorrências das regras são utilizados índices para vértices e índices para arcos. Arcos são destacados com índices nos casos em que os vértices são preservados e os arcos são deletados, caso contrário os

arcos não apresentam índices. A deleção de um vértice (origem/destino) ocasiona a deleção de seus arcos.

✓

Uma ocorrência é modelada por um morfismo total de grafos ($m: L \rightarrow IN$), o que intuitivamente significa que todos os elementos do lado esquerdo da regra devem estar presentes em IN para que a regra possa ser aplicada.

Exemplo 2.5 (Ocorrência) Na figura 2.8 é possível identificar duas ocorrências (m_1 e m_2) da regra $move \blacktriangle$ no grafo IN , uma vez que o grafo IN apresenta \star_1 e \star_2 .

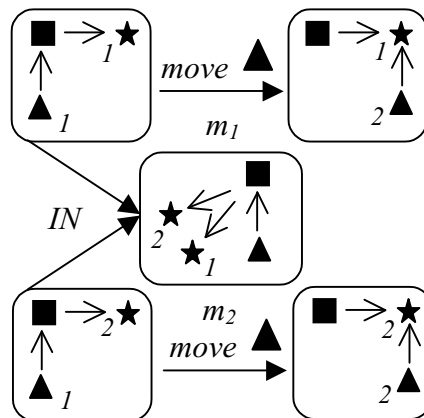


FIGURA 2.8 – Ocorrência de regras.

✓

Definição 2.7 (Passo de Derivação) Dada uma regra $r_s: L_s \rightarrow R_s$, e uma ocorrência $m_s: L_s \rightarrow IN_s$ de r_s em um grafo IN_s . Um **Passo de Derivação** s com a regra r_s na ocorrência m_s é uma tupla $s=(r_s, S)$, tal qual o diagrama a seguir, onde OUT_s é obtido através dos seguintes passos:

$$\begin{array}{ccc}
 L_s & \xrightarrow{r_s} & R_s \\
 m_s \downarrow & & \downarrow \\
 & S & \\
 IN_s & \longrightarrow & OUT_s
 \end{array}$$

1. Adicionar a IN_s tudo que for criado pela regra (itens que fazem parte do lado direito R_s da regra, mas não do lado esquerdo L_s).
2. Deletar do grafo resultante do passo 1, tudo que deve ser deletado pela regra (itens que fazem parte do lado esquerdo L_s e não do lado direito R_s).
3. Deletar arcos pendentes. Este passo é necessário no caso de haverem arcos conectados a vértices deletados no passo 2. Como o resultado da aplicação de uma regra deve ser um grafo, estes arcos devem ser deletados também.

Observação: O diagrama S corresponde a um *pushout* na categoria dos grafos e morfismos parciais [RIB96].

Note que, através da definição 2.7, se um vértice for deletado, todos os arcos conectados a este vértice também serão deletados, mesmo que isso não esteja explicitamente definido na regra.



Exemplo 2.6 (Passo de derivação) Na figura 2.9 *IN* é transformado em *OUT* através da aplicação da regra *move* ▲.

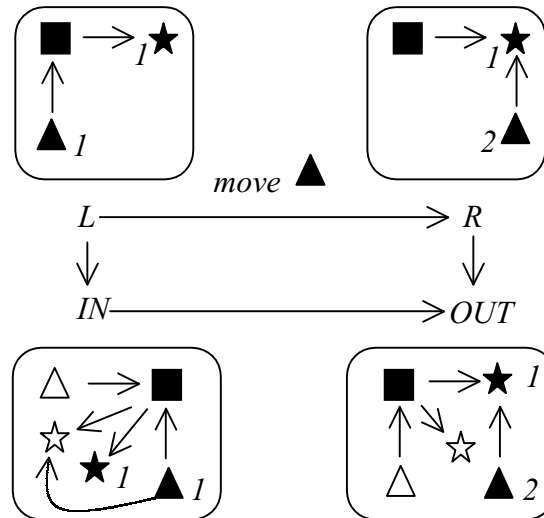


FIGURA 2.9 – Passo de derivação.



Definição 2.8 (Derivação Sequencial) Dada uma gramática de grafos $GG=(GT, GI, N)$. A **Derivação Sequencial** de GG é uma seqüência de passos $\sigma = s1; s2; s3; \dots$, onde $IN_{s1} = GI$, $OUT_{si} = IN_{si+1}$ e os passos usam regras de GG .

Uma derivação sequencial de uma gramática de grafos é uma seqüência composta por passos de derivação que podem ser obtidos tendo um grafo inicial e usando as regras da gramática. O grafo de saída de um passo é o grafo de entrada para o próximo passo, veja figura 2.10.

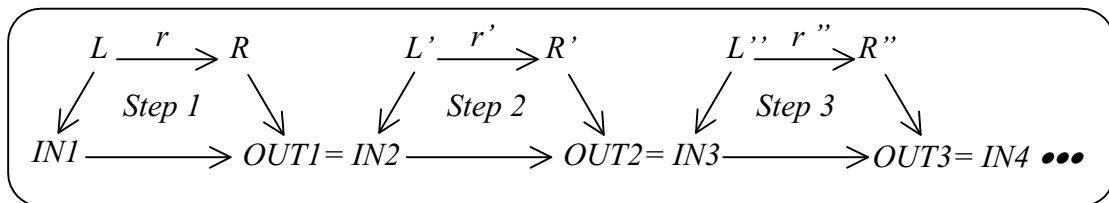


FIGURA 2.10 – Derivação sequencial.



Exemplo 2.7 As seguintes aplicações de regras, ou ações, são possíveis na gramática representada na figura 2.7.

1. O \triangle move de \blacksquare para \star .
2. O \blacktriangle move de \blacksquare para \star .
3. O \triangle move de \blacksquare para \star .
4. A \star inverte para \star .

Analisando o exemplo 2.7 é possível identificar que as ações 1 e 2 podem ocorrer em **paralelo** porque elas envolvem figuras diferentes. As ações 1 e 3 geram **conflito** porque ou o Δ se move para \star ou para \blackstar . A ação 4 **depende** da ação 1 para que seja possível inverter de \blackstar para \star . Uma derivação seqüencial para o exemplo 2.7, chamada de σ_4 , está ilustrada na figura 2.11. Nesta derivação, as ações 1 e 4 acontecem nesta ordem (são observadas como os passos **s11** e **s21** da derivação seqüencial). Seja σ_3 uma derivação seqüencial correspondendo à ocorrência da ação 3. Considerando uma semântica seqüencial para esta gramática, entre outras, as seguintes computações são possíveis (o símbolo (;) denota a composição seqüencial): $\sigma_1 = (s11)$, $\sigma_3 = (s3)$, $\sigma_4 = (s11; s21)$.

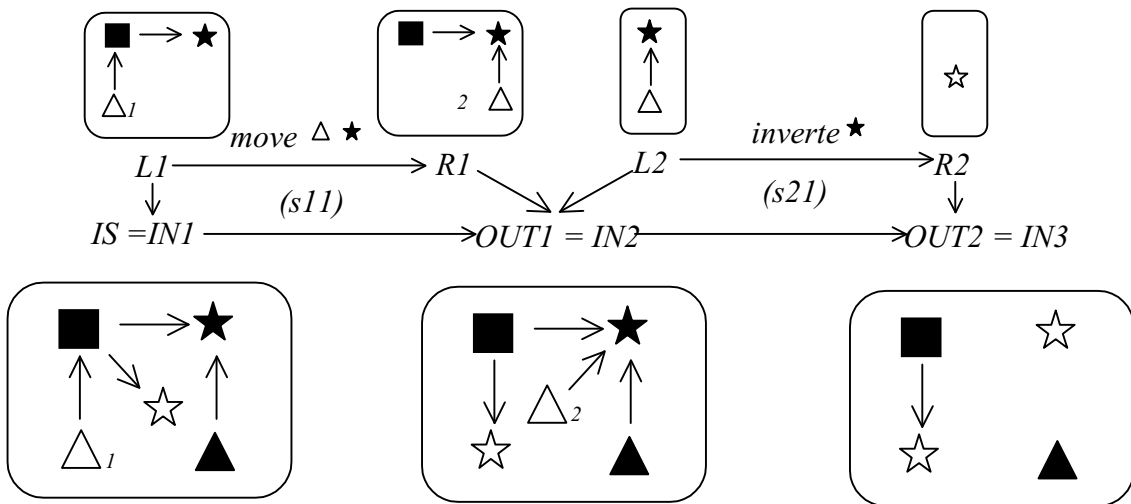


FIGURA 2.11 – Derivação seqüencial σ_4 .

✓

O comportamento de um sistema é baseado na aplicação de regras. Porém, a aplicação de uma regra está na maioria das vezes relacionada com a aplicação de outras regras e depende do modelo semântico adotado.

A escolha de um modelo semântico é feita baseada nas considerações que são relevantes para a especificação do sistema. A definição do modelo semântico provê meios para a análise das computações do sistema, de tal forma, que possibilita a obtenção de respostas para questões como: quais regras serão aplicáveis em um determinado estado do sistema, quais regras tidas como aplicáveis geram conflito e quais regras podem ocorrer em paralelo. Para que essas dúvidas sejam esclarecidas, é preciso que se faça uma análise sobre a especificação do sistema e o modelo semântico adotado. Assim, para entender como será possível realizar a análise e fornecer respostas para tais questões é necessário se ater a algumas definições que serão apresentadas a seguir. Essas especificações definem o modelo semântico a ser utilizado.

Definição 2.9 (Regras Aplicáveis) Dada uma gramática $GG = (GT, GI, N)$, um grafo G tipado sobre GT e o conjunto de regras de GG . O conjunto de regras aplicáveis em G é denotado por $RegrasAplicáveis_G$ definida por:

$$RegrasAplicáveis_G = \{(r, m) \mid r : L \rightarrow R \in N \text{ e } m : L \rightarrow G \text{ é uma ocorrência}\}$$

Para que uma regra seja considerada como uma regra aplicável em um estado atual do sistema é preciso que exista pelo menos uma ocorrência para esta regra, i.e., exista pelo menos um subgrafo (no estado atual do sistema) que corresponde ao lado esquerdo da regra (L).

Como uma regra pode ser aplicada de várias formas, o conjunto de regras aplicáveis consiste de pares contendo uma regra e uma ocorrência dessa regra.



Exemplo 2.8 (Regras Aplicáveis) Na figura 2.12 G representa o estado atual do sistema e N representa o conjunto de regras. As regras em destaque (cinza) são tidas como regras aplicáveis ao estado atual do sistema (juntamente com as respectivas ocorrências).

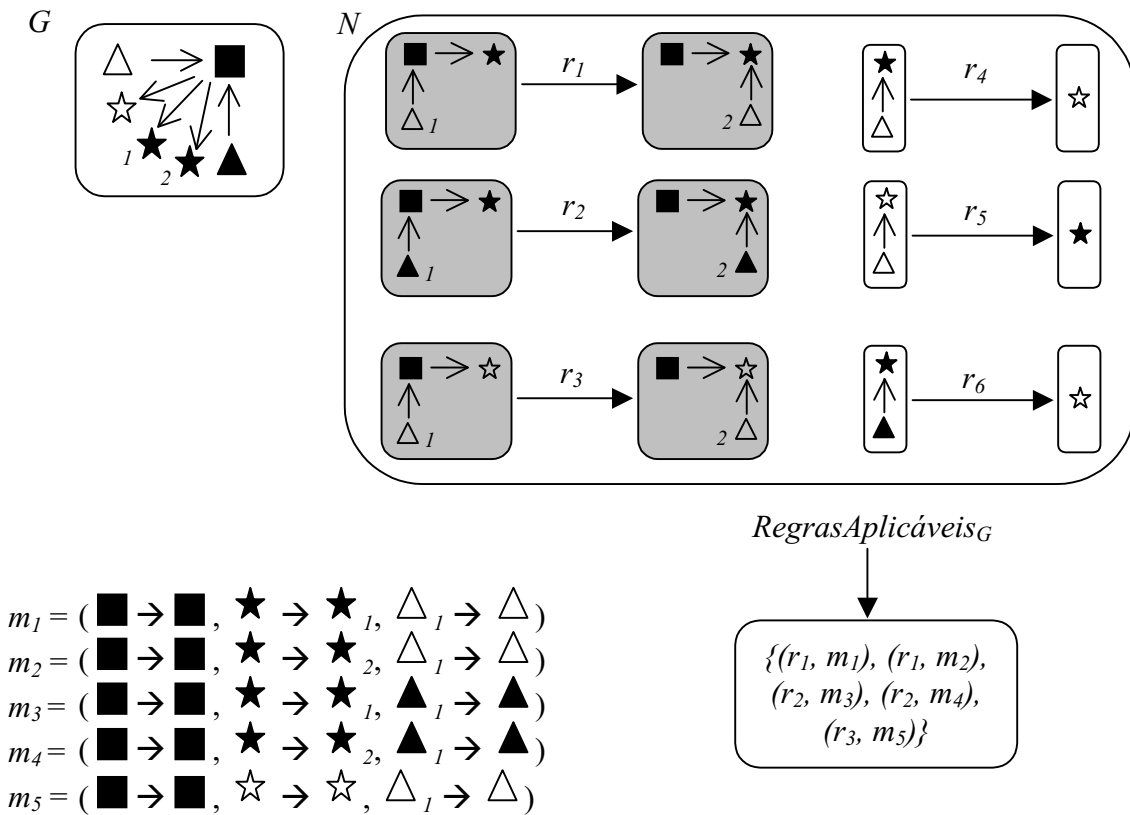


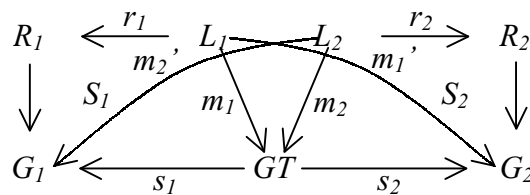
FIGURA 2.12 – Regras aplicáveis em um grafo G .



No exemplo 2.8 o número de regras que poderiam ser identificadas como aplicáveis para o estado atual do sistema (representado pelo grafo G), poderia ser bem diversificado (conforme ilustra o exemplo). Porém, dentre o conjunto de regras nem todas apresentavam no lado esquerdo da regra um grafo que pudesse ser caracterizado como um subgrafo de G . Assim, dentre seis regras, três foram selecionadas como regras aplicáveis ao estado atual do sistema, sendo que para r_1 e r_2 foram identificadas até duas ocorrências, ou melhor, foram encontrados dois subgrafos em G que correspondem ao lado esquerdo de r_1 e r_2 . Para que as regras r_1 , r_2 e r_3 possam ser aplicadas simultaneamente (em paralelo), é preciso que essas regras aplicáveis não estejam em conflito, i.e., não possuam acesso de escrita (deleção) em um mesmo item. Não basta

identificar as regras aplicáveis é preciso garantir que elas formam um conjunto consistente, e para isso utiliza-se o conceito de conflito. É possível identificar situações em que duas regras compartilham o acesso de escrita a um mesmo item, caracterizando o conflito entre regras. Considerando dois tipos de conflito: potencial (entre regras) e conflito propriamente dito (entre pares <regra, ocorrência>).

Definição 2.10 (Conflito Potencial) Dada uma gramática $GG = (GT, GI, N)$ e duas regras $r_1: L_1^{GT} \rightarrow R_1^{GT}$, $r_2: L_2^{GT} \rightarrow R_2^{GT} \in N$. Sejam $m_1: L_1 \rightarrow GT$ e $m_2: L_2 \rightarrow GT$ os morfismos de tipagem de L_1 e L_2 , respectivamente. A regra r_1 está em conflito potencial com r_2 , denotado por $r_1 \# r_2$, se m_2' ou m_1' não são totais, onde:



- S_1 e S_2 são aplicações de (r_1, m_1) e (r_2, m_2) , respectivamente
- $m_2' = s_1 \circ m_2: L_2 \rightarrow G_1$
- $m_1' = s_2 \circ m_1: L_1 \rightarrow G_2$

✓

Exemplo 2.9 (Conflito Potencial) Na figura 2.13 GT representa o grafo-tipo, r_1 e r_3 representam as regras a serem analisadas e G_1 e G_3 representam os estados resultantes da aplicação das regras. Analisando m_3' e m_1' é possível identificar que entre r_1 e r_3 há conflito potencial, uma vez que $s_1 \circ m_3: L_3 \rightarrow G_1$ e $s_3 \circ m_1: L_1 \rightarrow G_3$ não são totais. Na figura 2.14 GT representa o grafo-tipo, r_1 e r_2 representam as regras a serem analisadas e G_1 , G_2 representam os estados resultantes da aplicação das regras. Analisando m_2' e m_1' é possível identificar que entre r_1 e r_2 não há conflito potencial, uma vez que $s_1 \circ m_2: L_2 \rightarrow G_1$ e $s_2 \circ m_1: L_1 \rightarrow G_2$ são totais.

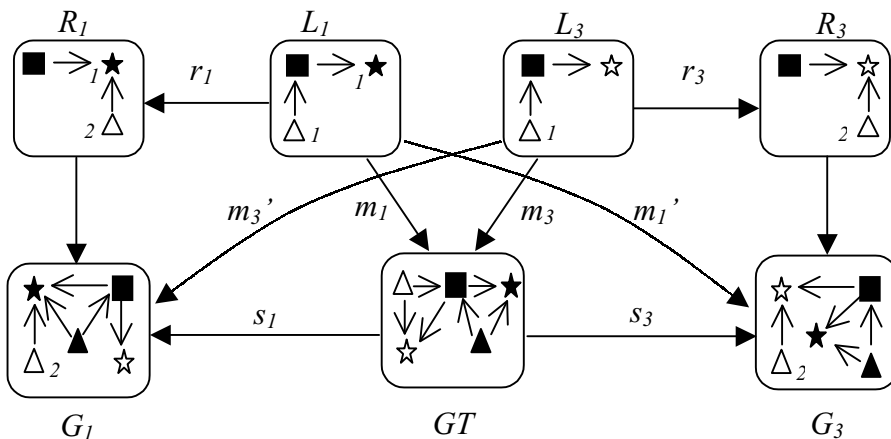


FIGURA 2.13 – Conflito potencial entre regras.

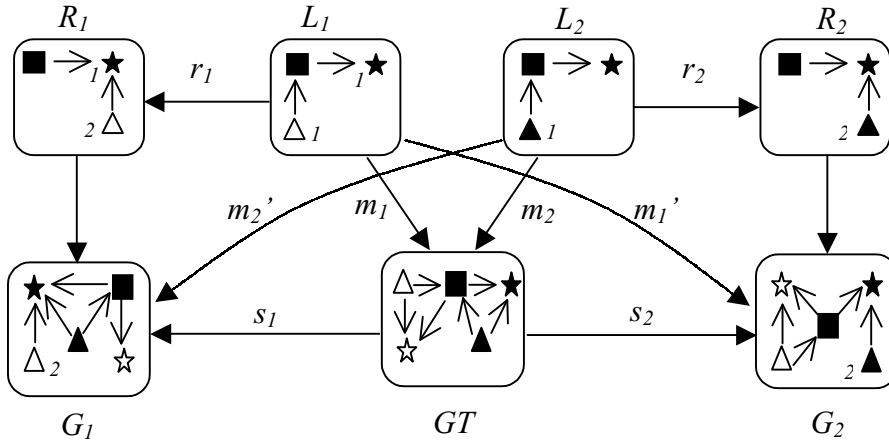
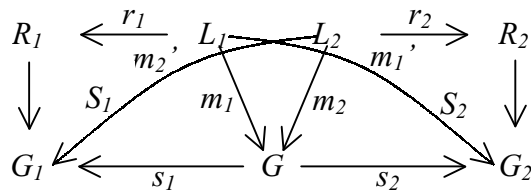


FIGURA 2.14 – Ausência de conflito potencial entre regras.

Conflito potencial possibilita uma análise sobre todas as regras definidas na gramática, sendo possível determinar quais regras podem vir a apresentar conflito quando executadas em paralelo. Regras que não apresentam conflito potencial nunca estarão em conflito real (veja figura 2.14). Porém, regras que apresentam conflito potencial não garantem a existência de conflito real para qualquer estado do sistema, é preciso testar se há ou não conflito em um determinado estado do sistema (veja figura 2.13).

Definição 2.11 (Conflito) Dadas duas regras aplicáveis (r_1, m_1) e (r_2, m_2) em um grafo G , (r_1, m_1) está em conflito com (r_2, m_2) , denotado por $(r_1, m_1) \# (r_2, m_2)$, se m_2' ou m_1' não são totais, onde:



- S_1 e S_2 são aplicações de (r_1, m_1) e (r_2, m_2) , respectivamente
- $m_2' = s_1 \circ m_2 : L_2 \rightarrow G_1$
- $m_1' = s_2 \circ m_1 : L_1 \rightarrow G_2$

Exemplo 2.10 (Conflito) Na figura 2.15 G representa o estado atual do sistema, r_1 e r_3 representam as regras aplicáveis em G (apresentam conflito potencial – figura 2.13), e G_1, G_3 representam os estados resultantes da aplicação das regras. Analisando m_5' e m_1' é possível identificar que r_1 e r_3 estão em conflito, uma vez que $s_1 \circ m_5 : L_3 \rightarrow G_1$ e $s_3 \circ m_1 : L_1 \rightarrow G_3$ não são totais.

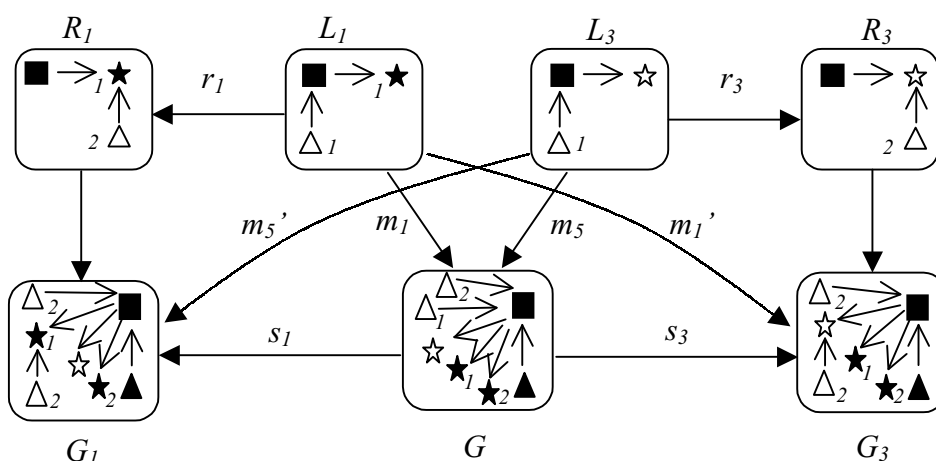


FIGURA 2.15 – Regras em conflito.

Exemplo 2.11 (Sem Conflito) Na figura 2.16 G representa o estado atual do sistema, r_1 e r_3 representam as regras aplicáveis em G , e G_1 , G_3 representam os estados resultantes da aplicação das regras. Analisando m_5' e m_1' é possível identificar que r_1 e r_3 não estão em conflito, uma vez que $s_1 \circ m_5 : L_3 \rightarrow G_1$ e $s_3 \circ m_1 : L_1 \rightarrow G_3$ são totais.

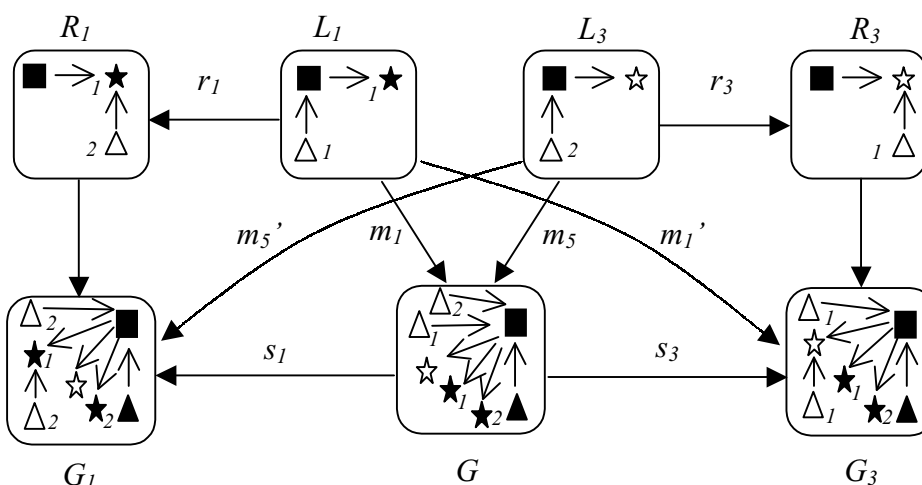


FIGURA 2.16 – Regras sem conflito.

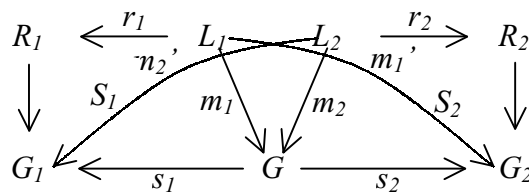
Naturalmente muitas das regras tidas como aplicáveis a um mesmo estado podem ser executadas em paralelo, i.e., são acionadas simultaneamente. Considerando as ocorrências dessas regras, pode-se ter 4 situações [RIB97]:

- Ocorrências que não se sobrepõem em G :** Neste caso as regras r_1 e r_2 podem ser aplicadas obviamente em paralelo uma vez que agem independentemente.
- Ocorrências que se sobrepõem em itens preservados:** Neste caso as regras também podem ser aplicadas em paralelo porque os itens compartilhados são preservados por ambas as regras, uma vez que o acesso é de somente leitura.

- c) **Ocorrências que se sobrepõem em itens que são preservados por uma regra e deletados pela outra:** Neste caso a permissão da aplicação paralela das regras pode ou não ser permitida. Caso seja permitida significa que houve uma permissão de somente leitura e uma de escrita para que as regras possam agir juntas em itens compartilhados.
- d) **Ocorrências que se sobrepõem em itens que são deletados por ambas as regras:** Neste caso também é possível que seja permitida ou proibida a aplicação paralela das regras. A permissão significa que ambas as regras tem acesso de escrita, podendo agir juntas em itens compartilhados.

Gramáticas de grafos possibilitam a escolha de qual tipo de paralelismo será possível no sistema. Cada escolha provavelmente conduzirá a um tipo diferente de semântica concorrente para este sistema. Normalmente os casos a) e b) são permitidos. Estes casos representam um paralelismo forte porque é um tipo simétrico de paralelismo: se dois passos de derivação s_1 e s_2 podem acontecer em paralelo também podem acontecer consecutivamente em qualquer ordem e vice-versa, sem qualquer alteração no resultado da aplicação da regra. Caso c) é chamado paralelismo fraco e representa paralelismo assimétrico: se dois passos de derivação s_1 e s_2 podem acontecer em paralelo eles também podem acontecer em pelo menos uma ordem. Caso d) normalmente é proibido, embora sob algumas restrições possa ser aplicado. Neste trabalho, considera-se paralelismo forte, proibindo a execução paralela de regras nas situações c) e d).

Definição 2.12 (Paralelismo) Dadas duas regras aplicáveis (r_1, m_1) e (r_2, m_2) em um grafo G , (r_1, m_1) é paralela a (r_2, m_2) , denotado por $(r_1, m_1) \parallel (r_2, m_2)$, se m_2' e m_1' são totais, onde:



- S_1 e S_2 são aplicações de (r_1, m_1) e (r_2, m_2) , respectivamente
- $m_2' = s_1 \circ m_2 : L_2 \rightarrow G_1$
- $m_1' = s_2 \circ m_1 : L_1 \rightarrow G_2$

✓

Exemplo 2.12 (Paralelismo) Na figura 2.17 G representa o estado atual do sistema, r_1 e r_2 representam as regras aplicáveis em G , e G_1, G_2 representam os estados resultantes da aplicação das regras. Analisando m_3' e m_1' é possível identificar que r_1 e r_2 podem ser executadas em paralelo, uma vez que $s_1 \circ m_3 : L_2 \rightarrow G_1$ e $s_2 \circ m_1 : L_1 \rightarrow G_2$ são totais.

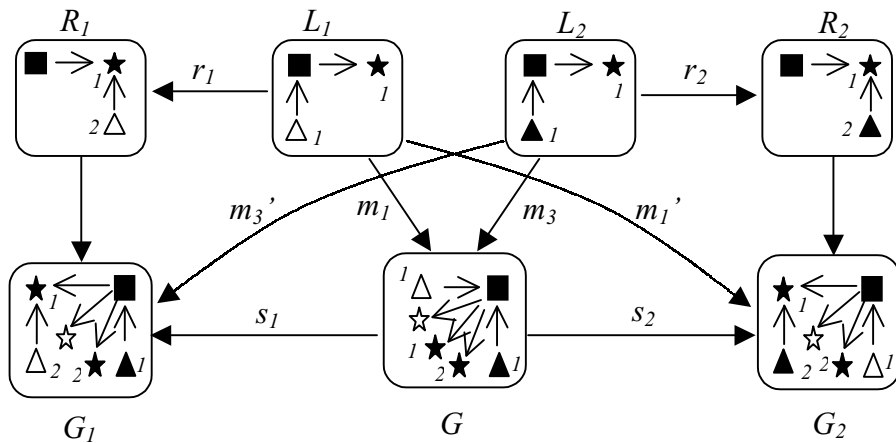


FIGURA 2.17 – Paralelismo entre regras.

Definição 2.13 (Conjunto Consistente) Dado o conjunto $R \subseteq \text{RegrasAplicáveis}_G$. Este conjunto é consistente se $\forall (r_1, m_1) \text{ e } (r_2, m_2) \in R (r_1, m_1) \not\# (r_2, m_2)$.

Notação: O símbolo $\not\#$ representa que não há conflito entre regras aplicáveis enquanto o símbolo $\#$ representa conflito.

Exemplo 2.13 (Conjunto Consistente) Na figura 2.12 o conjunto de $\text{RegrasAplicáveis}_G$ é formado por: $\{(r_1, m_1), (r_1, m_2), (r_2, m_3), (r_2, m_4), (r_3, m_5)\}$, sendo que dentre essas regras $(r_1, m_1) \# (r_3, m_5)$ e $(r_1, m_2) \# (r_3, m_5)$ e $(r_1, m_1) \# (r_1, m_2)$. Assim, os seguintes conjuntos, entre outras, são consistentes.

$$\text{Conjunto Consistente}(1) = \{(r_1, m_1), (r_2, m_3)\}$$

$$\text{Conjunto Consistente}(2) = \{(r_2, m_4), (r_3, m_5)\}$$

2.3 Gramáticas de Grafos Baseada em Objetos

Um sistema baseado em objetos consiste de entidades autônomas (objetos) que comunicam e cooperam com outras entidades através de mensagens. O comportamento de uma entidade pode ser descrito através de regras que são acionadas pelo recebimento de mensagens. Regras podem mudar o estado de uma entidade e/ou enviar mensagens a outras entidades, sendo que uma entidade pode desempenhar muitas ações em paralelo.

Para especificação de um sistema baseado em objetos é necessária uma linguagem de especificação que seja capaz de representar entidades, ou melhor, os objetos do sistema e comportamento de tais objetos. Gramáticas de grafos possibilitam a especificação de sistemas baseados em objetos através de um tipo especial de gramática chamada gramáticas de grafos baseada em objetos (*object-based graph grammars*).

Segundo [RIB2001], gramáticas de grafos baseada em objetos apresentam vantagens: no lado prático, permitem que a especificação ganhe um estilo baseado em objeto, muito familiar para a maioria dos usuários, e desta forma facilitam a construção,

o entendimento e conseqüentemente o uso como base de implementação; no lado teórico, as restrições permitem uma implementação eficiente do comportamento de uma gramática de grafos, assim como facilita as análises da gramática. Basicamente, foram impostas restrições nos tipos de grafos que são usados e nos tipos de comportamentos que as regras podem especificar. Cada grafo baseado em objeto pode ser composto por instâncias de vértices e arcos que representam entidades, mensagens, atributos e parâmetros. Além disso, são utilizados tipos abstratos de dados como atributos para vértices, permitindo uma forma melhor de descrever o sistema.

Na figura 2.18 pode-se observar que os vértices representam entidades, mensagens e tipos abstratos de dados, onde os elementos de tipos abstratos de dados são admitidos como atributos de entidades (atr1 e atr2) e/ou parâmetros de mensagens (par1 e par2).

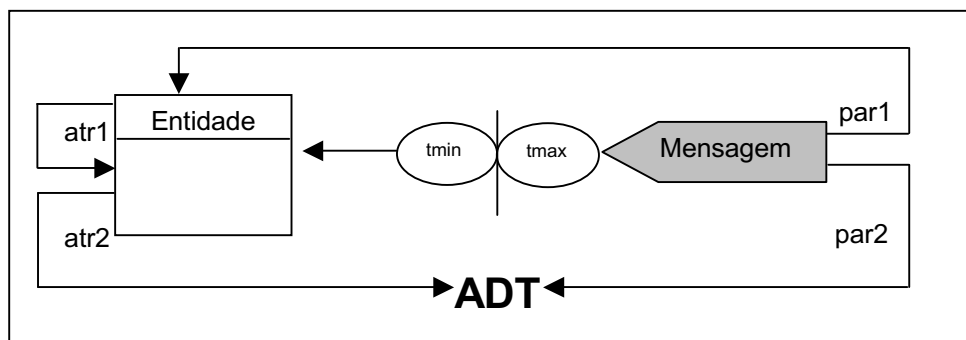


FIGURA 2.18 – Grafo-tipo baseado em objeto.

Na seção 2.1 gramática de grafos foi definida como uma tupla composta de um grafo-tipo, um grafo inicial e um conjunto de regras. Assim, pode-se dizer que gramáticas de grafos baseadas em objetos são duplamente tipadas. Pois, além da tipagem estabelecida pelo grafo-tipo da gramática, serão utilizados conjuntos de rótulos (tipos de vértices e tipos de arcos) estabelecidos a partir do grafo-tipo baseado em objeto, e algumas restrições (um arco que tem como origem um vértice do tipo mensagem deve ter como destino um vértice do tipo entidade) para garantir que a origem e o destino dos arcos sejam consistentes quanto aos tipos estabelecidos no grafo-tipo baseado em objeto (veja o exemplo 2.14).

Exemplo 2.14 Na figura 2.19 *GTBO* representa o grafo-tipo baseado em objeto, *GT* o grafo-tipo da entidade *Train* e *GI* o grafo inicial da entidade *Train*, sendo que *GI* é tipado a partir das representações feitas em *GT*. Para simplificar o desenho dos grafos, os arcos que tem como destino tipos abstratos de dados serão representados dentro das caixas que representam as entidades. *GT* é tipado a partir de definições feitas em *GTBO*, onde os vértices devem ser dos tipos: entidade, mensagem ou ADT, os arcos devem ser dos tipos: atributo, parâmetro ou destino, sendo que o tipo do vértice origem e o tipo do vértice destino de cada arco apresenta restrições. O morfismo de *GT* para *GTBO* mapeia os arcos *curr*, *next*, *gate* e *n* para *atr1* e os arcos *posNext*, *posGate*, *wait* e *station* para *atr2*.

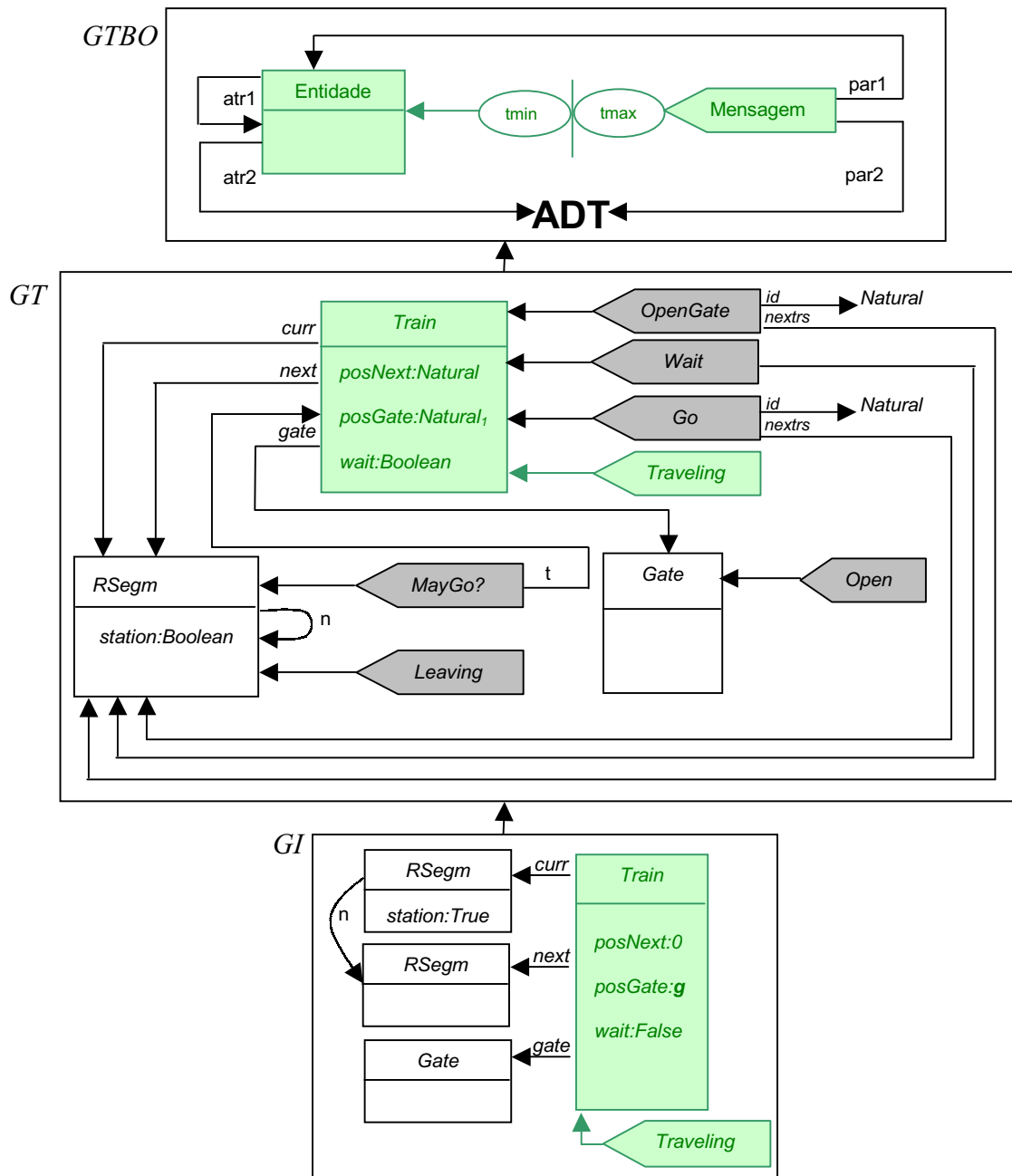


FIGURA 2.19 – Grafo duplamente tipado.

Para maiores detalhes e definições formais de GTBO veja [DOT2000].

3 PLATUS

PLATUS é um ambiente que permite a construção de modelos de simulação baseados em técnicas de especificação formal.

Segundo [COP2000], o modelo de um sistema de controle construído no ambiente PLATUS não é composto por um único módulo, mas por uma composição de componentes individuais, que facilitam a modelagem de sistemas complexos. Isto implica que a linguagem de especificação formal escolhida para especificar o modelo deve apresentar operadores de composição. O uso de operadores de composição define uma maneira de especificar modelos de simulação nos quais o comportamento do sistema como um todo pode ser derivado do comportamento de suas partes.

A arquitetura PLATUS é modular. Ferramentas independentes permitem a especificação das entidades, a definição dos elementos da coleta de dados para estatística e a definição dos elementos de visualização da simulação. A independência entre as definições melhora a flexibilidade da especificação de uma entidade, uma vez que a especificação pode ser reusada facilmente através de mudanças nas implementações de animação e estatística.

PLATUS usa gramáticas de grafos como ferramenta descritiva, mas não uma gramática de grafos usual e sim uma gramática chamada de gramáticas de grafos baseadas em objetos, onde a especificação ganha um estilo baseado em objetos. Originalmente, o formalismo de gramáticas de grafos não inclui a noção de tempo. Mas a necessidade de descrever modelos de simulação fez com que o conceito de tempo fosse incorporado.

As próximas seções apresentam detalhes sobre: o ambiente, a estrutura, o tipo de gramática de grafos utilizada no PLATUS, definições para as entidades, o modelo de simulação, e da simulação. A seção 3.1 apresenta a arquitetura do ambiente. A seção 3.2 trata da estrutura de uma especificação do PLATUS. A seção 3.3 define o tipo de gramáticas de grafos utilizada no PLATUS. A seção 3.4 trata das entidades. A seção 3.5 apresenta um exemplo de um sistema de ferrovia. A seção 3.6 define o modelo de simulação da ferrovia e a seção 3.7 trata da simulação.

3.1 Arquitetura PLATUS

O ambiente PLATUS é um ambiente de simulação que permite a realização de simulações sobre os modelos descritos através de gramáticas de grafos baseadas em objetos. PLATUS apresenta uma arquitetura modular, formada por ferramentas independentes que permitem a especificação das entidades, a definição dos elementos da coleta de dados para estatística e a definição dos elementos de visualização da simulação, possibilitando a construção dos modelos através da ferramenta de edição de modelos. Para que a simulação seja possível é preciso que haja uma descrição do modelo, junto com a descrição da simulação. Assim, lotes de dados serão gerados a partir da simulação, possibilitando a análise através da ferramenta analisadora de resultados. A análise permitirá a validação, garantindo que as propriedades do sistema sejam atendidas antes mesmo da implementação. A figura 3.1 apresenta o esquema da arquitetura do PLATUS.

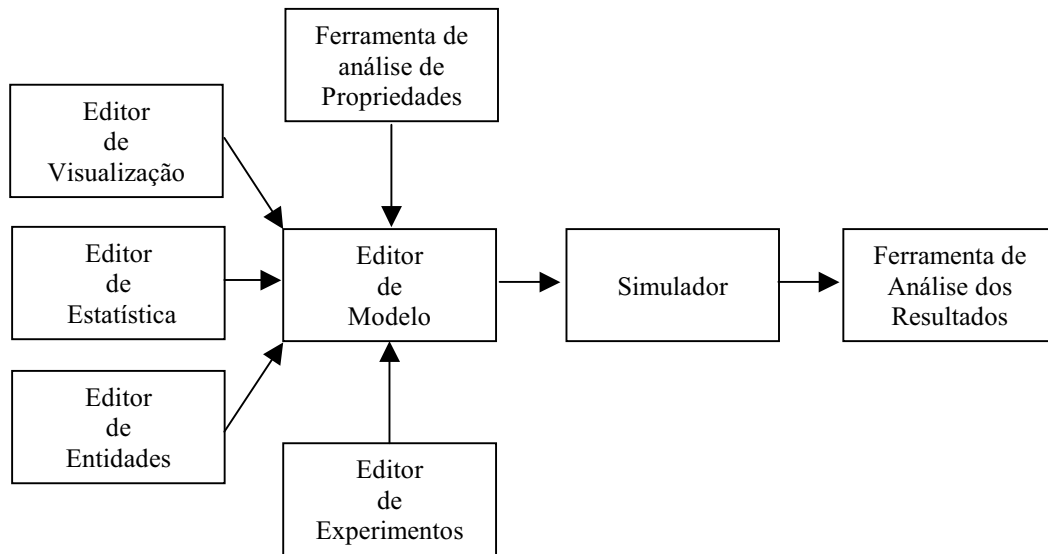


FIGURA 3.1 – Arquitetura do PLATUS.

A ferramenta de edição das entidades permite a especificação das entidades isoladas que podem ser armazenadas numa biblioteca para uso posterior (para compor modelos). O uso do formalismo de gramáticas de grafos permite a descrição de todos os aspectos de uma entidade, sendo elas: estrutura estática (atributos), estrutura dinâmica (comportamento) e interface de passagem de mensagem mais a interface com a coleção estatística e elementos visualizadores. A ferramenta de edição das entidades é basicamente composta por três módulos: editor do grafo-tipo, editor do estado inicial do grafo e editor de regras. O editor de regras tem vários serviços que simplificam a edição das regras. Estes serviços são tão simples como a capacidade de copiar e mover regras, bem como a geração automática do “lado esquerdo” da regra. Além disso, todos os vértices e arcos são instanciados a partir da definição feita para o grafo-tipo (veja detalhes na seção 4.2). Regras representam o comportamento do sistema. A descrição do comportamento precisa ser independente das outras (visualização e estatística) porque esta é realmente a especificação de um modelo semântico do sistema que será simulado (a verificação formal deve ser baseada somente nesta descrição). Enquanto os comportamentos são descritos com gramáticas de grafos, a especificação da animação e a estatística de uma entidade são descritas como refinamentos do comportamento de gramáticas de grafos (pela adição do comportamento de gramáticas de grafos correspondendo às chamadas reais de procedimentos de animação e estatísticas). Isto é feito, para melhorar a flexibilidade da especificação de uma entidade, que pode ser reusada facilmente através de mudanças nas implementações de animação e estatística. Para obter a especificação de um componente, o ambiente correspondente às gramáticas de grafos, animação e estatística é composto de operadores de composição de gramáticas de grafos definido em [RIB96]. O uso de uma interface garante que a especificação de componentes seja ocultada de outros componentes. Durante a simulação, isto também permite a substituição de componentes simulados por correspondentes reais sem ter que fazer nenhuma mudança em outros componentes (o componente real tem que se comportar, proceder de acordo com a interface da entidade que está sendo substituída por ele).

3.2 Estrutura de uma Especificação do PLATUS

Segundo [COP2000], a descrição de um sistema de controle construído no ambiente PLATUS não é composta por um único módulo, mas por uma composição de componentes individuais, que facilitam a modelagem de sistemas complexos. Isto implica que a linguagem de especificação formal escolhida para descrever o modelo deve apresentar operadores de composição. Pois, através do uso de operadores de composição, define-se uma maneira de especificar modelos de simulação nos quais o comportamento do sistema como um todo pode ser derivado do comportamento de suas partes.

Em diversos formalismos de especificação de sistemas concorrentes, o comportamento de componentes individuais isolados não pode ser determinado [COP2000]. Isso acaba por caracterizar um problema, porque qualquer consideração sobre o comportamento do sistema só pode ser feita depois da modelagem completa. Em [RIB96] [RIB99a] foi definida uma maneira de construir especificações usando componentes, descritos com gramáticas de grafos, onde o comportamento individual pode ser determinado e preservado pelos operadores de composição que compõem o sistema. Assim, segundo [COP2000], foi possível achar um caminho para modelar sistemas concorrentes onde o comportamento do sistema inteiro pode ser derivado de comportamentos individuais.

No PLATUS um sistema é composto por entidades, onde uma entidade é descrita por uma gramática e a interface por um conjunto de regras descrevendo as mensagens e parâmetros correspondentes que são usados por essas entidades. A gramática que descreve os componentes é obtida pela composição de gramáticas que descrevem o comportamento, a estatística e a animação (veja figura 3.2). O comportamento é a estrutura dinâmica da entidade descrita através de gramáticas de grafos, sendo caracterizada por um grafo-tipo, um grafo inicial e um conjunto de regras que são ativadas pelo envio de mensagens (*triggers*) e que acabam por descrever a reação de uma entidade ao receber uma determinada mensagem. Já a estatística e a animação são refinamentos do comportamento, ou seja, não podem incluir o tratamento de novos tipos de mensagens pela entidade, o que implica que não são recebidas mensagens provenientes dos módulos de animação e/ou estatística, apenas são enviadas mensagens para estes módulos, desta forma o operador de composição garante, por exemplo, que qualquer procedimento usado para coletar estatísticas não tenha influência no comportamento do sistema.

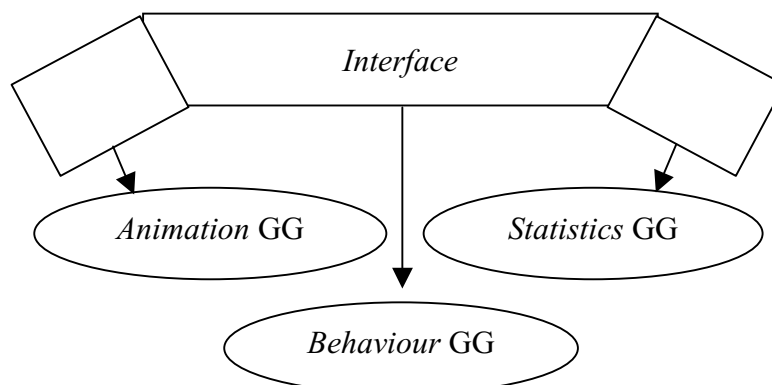


FIGURA 3.2 – Modelo de um sistema de controle.

3.3 Tipo de Gramáticas de Grafos usada no PLATUS

A representação de sistemas através de gramáticas de grafos pode ser feita através de grafos simples, que consistem apenas de arcos e vértices. Mas geralmente, segundo [DEH2000], são utilizados alguns mecanismos de tipagem nos grafos. A tipagem de grafos pode ser feita pelo uso de rótulos (cada arco/vértice é rotulado com um elemento de um alfabeto Σ de rótulos), atributos (onde tipos abstratos de dados são usados) ou grafo-tipo (onde um grafo é utilizado como tipo).

No PLATUS a tipagem de grafos é feita através de um mecanismo mais poderoso, que torna a linguagem de especificação mais abstrata. Em vez de usar uma gramática de grafos usual, PLATUS utiliza gramáticas chamadas de gramáticas de grafos baseadas em objetos (veja seção 2.3), onde cada entidade descrita pode ser referenciada como uma classe, e cada instância pode ser vista como um objeto. Assim, a representação dos sistemas ganha um estilo baseado em objetos.

- **Grafo:** Cada grafo descrito pela gramática de grafos baseada em objeto é composto por instâncias de vértices e arcos (veja figura 2.18). Os vértices representam entidades, tipos abstratos de dados, tempo e mensagens, onde *time stamps*¹ - *tmin* e *tmax* são atributos especiais de uma mensagem. Elementos de tipos abstratos de dados são admitidos como atributos de entidades e/ou parâmetros de mensagens. Um vértice do tipo mensagem deve ser conectado a pelo menos uma entidade. *Time stamps* descrevem o intervalo de tempo no qual a mensagem deve ocorrer em termos de unidades de tempo mínima e máxima relativas ao tempo corrente (veja seção 3.3.1). Esses *time stamps* (*tmin* e *tmax*) são associados para cada mensagem (na omissão são usados valores padrões). Note que o grafo da figura 2.18 define somente os tipos possíveis de vértices e arcos, não obriga entidades ou mensagens a terem atributos, e sim especifica que se uma entidade tem atributos, eles devem ser do tipo ADT ou do tipo entidade, por exemplo.
- **Grafo-tipo:** Para cada entidade existe um grafo denominado grafo-tipo da entidade que contém informações sobre todos os atributos desta entidade, relacionamentos com outras entidades, e mensagens enviadas/recebidas pela entidade. Este grafo-tipo é uma instanciação de um grafo-tipo baseado em objetos descrito na figura 2.18. Todas as regras que descrevem o comportamento da entidade só podem referenciar itens definidos neste grafo-tipo.
- **Regras:** Uma regra deve expressar a reação de uma entidade ao receber uma determinada mensagem. Portanto, o lado esquerdo da regra deve apresentar apenas um vértice do tipo mensagem, que é deletado pela regra (significando que a mensagem foi processada). O lado esquerdo e direito da regra devem apresentar apenas atributos da entidade que recebe a mensagem e parâmetros dessa mensagem (isto garante que o principal encapsulamento não será quebrado). No lado direito da regra os atributos podem assumir valores diferentes e novas mensagens podem ser criadas.

¹ Descreve o intervalo de tempo (fator interessante para a simulação) no qual a mensagem deve ocorrer em termos de unidade de tempo mínima (*tmin*) ou máxima (*tmax*) em relação ao tempo corrente.


- **Gramática de Grafos:** Uma gramática de grafos consiste de um grafo-tipo, um grafo inicial e um conjunto de regras. O grafo-tipo é descrito por tipos que serão usados nesta gramática (especifica os tipos de entidades, mensagens, ADT, atributos e parâmetros que são possíveis – como parte da estrutura de uma classe descritiva). O grafo inicial especifica o estado inicial do sistema (dentro da especificação de uma entidade, este estado pode ser especificado de uma forma abstrata, tornando-se concreto quando for construído o modelo de simulação, contendo instâncias de todas as entidades envolvidas neste sistema). Regras são especificadas através de instâncias das entidades reagindo a mensagens recebidas.



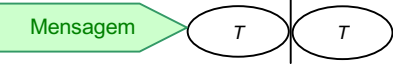


3.3.1 Aspectos de tempo

Originalmente, o formalismo de gramáticas de grafos não inclui a noção de tempo. Mas a necessidade de modelar sistemas em tempo real fez com que a opção de incorporar este conceito dentro do modelo, fosse escolhida. Segundo [RIB2001], esta escolha foi guiada pela semântica de gramáticas de grafos e o comportamento esperado para cada tipo de sistema que se deseja modelar. O seguinte questionamento tem sido considerado (como em [TAN97]):

- **Como o tempo deve ser incorporado?** Segundo [RIB2001], há muitas escolhas de onde colocar o tempo: regras, mensagens, entidades. A decisão foi colocar *time stamps* nas mensagens descrevendo quando elas serão enviadas/tratadas. Assim, pode-se programar certos eventos para acontecer em algum tempo específico no futuro. Como regras não tem *time stamps*, assume-se que a aplicação de uma regra é instantânea.
- **Que tipo de semântica deve ser adotada?** As possibilidades principais são: semântica de tempo fraca e forte. No primeiro caso, a mensagem não é forçada a ser tratada (mas se forçada, ela deve estar dentro do intervalo de tempo especificado). No último caso, se a mensagem é enviada, ela deve ser tratada. Adotou-se o segundo caso, pois um dos principais interesses é modelar sistemas fortes de tempo real (sistemas no qual a violação de *deadline* é fatal). Contudo, a escolha de quais mensagens devem ser tratadas no tempo corrente é feita por cada entidade, isto é possível considerando que modelos tem diferentes tipos de comportamentos para as entidades.

A unidade de tempo deve ser definida na especificação. Os *time stamps* das mensagens estão na forma: $\langle tmin, tmax \rangle$, com $tmin \leq tmax$, onde *tmin* e *tmax* representam o número mínimo/máximo em unidades de tempo, inicializado com o tempo corrente, dentro o qual a mensagem deve ser tratada. Os possíveis *time stamps* são:

-  : Esta mensagem deve ser tratada no mínimo em *tmin* unidades de tempo e no máximo em *tmax* unidades de tempo.

-  : Se $tmin$ é omitido, a soma do tempo corrente mais as unidades de tempo é assumida como o tempo mínimo para que essa mensagem seja tratada.
-  : Se $tmax$ é omitido, o infinito é assumido (isto é, está mensagem não tem tempo limite para ser tratada).
-  : Se $tmin = tmax$, esta mensagem deve ser tratada em um tempo específico durante a simulação.
-  : Se $tmin$, $tmax$ e a barra | forem omitidos, a mensagem será tratada num próximo tempo de simulação, e não tem tempo limite para ser tratada.
-  : Esta notação é equivalente a ter $tmin = tmax = T$, onde T é inicializado com o próximo tempo de simulação. Isto significa que esta mensagem deve ser tratada imediatamente.

Como os *time stamps* são sempre referentes ao tempo corrente, não é possível enviar mensagens que devem ser aplicadas no tempo corrente.

3.4 Entidades

O ambiente PLATUS é composto por entidades que são modeladas como objetos, que através do uso de gramáticas de grafos permite a descrição de todos os aspectos de uma entidade, sendo: estrutura estática (atributos), estrutura dinâmica (comportamento) e interface de passagem de mensagem mais a interface com a coleção estatística e elementos visualizadores.

A representação das entidades do ambiente PLATUS utiliza-se de grafos tipados para representar os estados e comportamento dos sistemas, sendo que as entidades são composta de quatro elementos distintos:

- Interface (GGi)
- Descrição de comportamento (GGc)
- Descrição de animação/visualização (GGa)
- Descrição da coleta de estatísticas (GGe)

Esta arquitetura visa manter a descrição da entidade livre de aspectos relativos a visualização e coleta de estatística que não são facilmente reusáveis em contextos distintos.

3.4.1 Interface

A interface de uma entidade do PLATUS é composta por conjuntos de regras, que apresentam:

- **Regras da interface de comportamento:** contém os tipos de entidades e mensagens de GGc, mas não podem conter atributos de entidades uma vez que estes representam o estado interno da entidade. Estas regras visam na verdade representar o comportamento da entidade em termos de trocas de mensagens a serem realizadas pela mesma.
- **Regras da interface de animação:** para cada mensagem que pode ser enviada para o módulo de animação, deve haver uma regra como a da figura 3.3, onde o lado direito será semelhante em todas as regras, já que nunca é exigida uma resposta por parte do módulo de animação. Isto ocorre porque se o módulo de animação enviasse algum tipo de resposta, fatalmente acabaria por influenciar o comportamento original da entidade.
- **Regras da interface de coleta de estatísticas:** são análogas as regras da interface de animação.

Assim, a interface de uma entidade do PLATUS apresenta uma composição formada por três conjuntos de regras, sendo que cada conjunto representa uma parte (comportamento, animação e coleta de estatísticas).

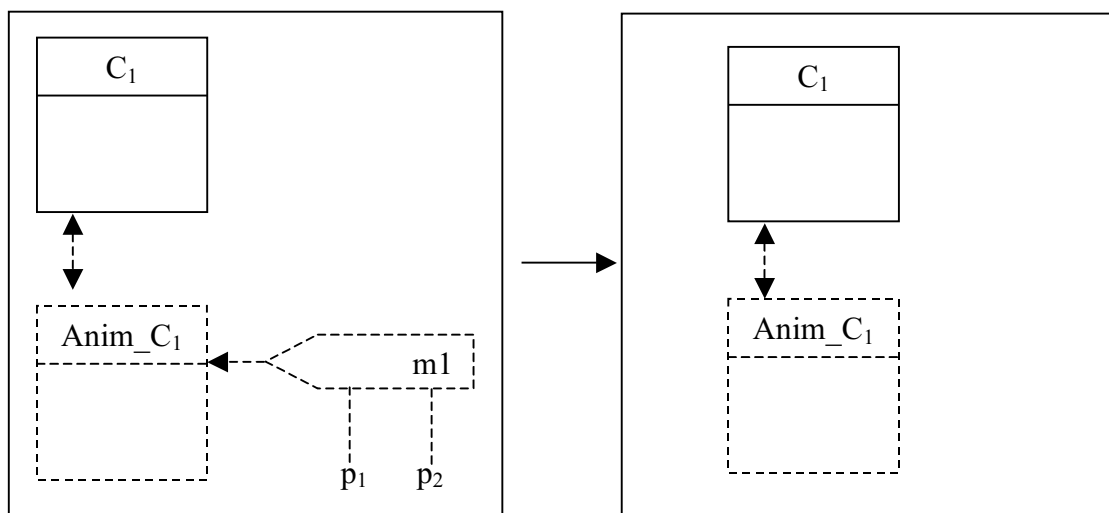


FIGURA 3.3 – Interface em GGa.

3.4.2 Comportamento

A estrutura dinâmica da entidade é toda representada através da GGc, onde o comportamento é descrito através de gramáticas de grafos. Assim, a descrição de comportamento (GGc) é caracterizada por um grafo-tipo, um grafo inicial e um conjunto de regras que são ativadas pelo envio de mensagens e que acabam por descrever a reação de uma entidade ao receber uma determinada mensagem, representando assim o comportamento do sistema. O fato do comportamento do sistema ser dinâmico torna necessário que a GGc trate novos tipos de mensagens. A figura 3.4 apresenta um exemplo de regra em GGc.

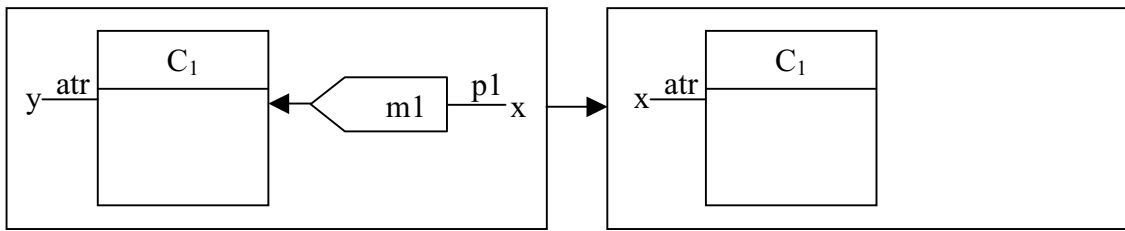


FIGURA 3.4 – Regra em GGc.

3.4.3 Animação e Coleta de Estatísticas

De maneira a manter as características descritas, GGa e GGe são refinamentos de GGc, ou seja, não podem incluir o tratamento de novos tipos de mensagens pela entidade. Isto implica que não são recebidas mensagens provenientes dos módulos de animação e/ou estatística, somente são enviadas mensagens para estes módulos. A figura 3.5 apresenta um exemplo de regra em GGa.

O uso de atributos de animação em GGa, como é o caso do atributo *cor* na figura 3.5, não pode inibir a aplicação da regra em GGc, ou seja, só podem ser usadas variáveis (no caso, *b* é uma variável).

Na descrição de uma regra em GGa, os itens pontilhados representam a parte da regra referente à animação.

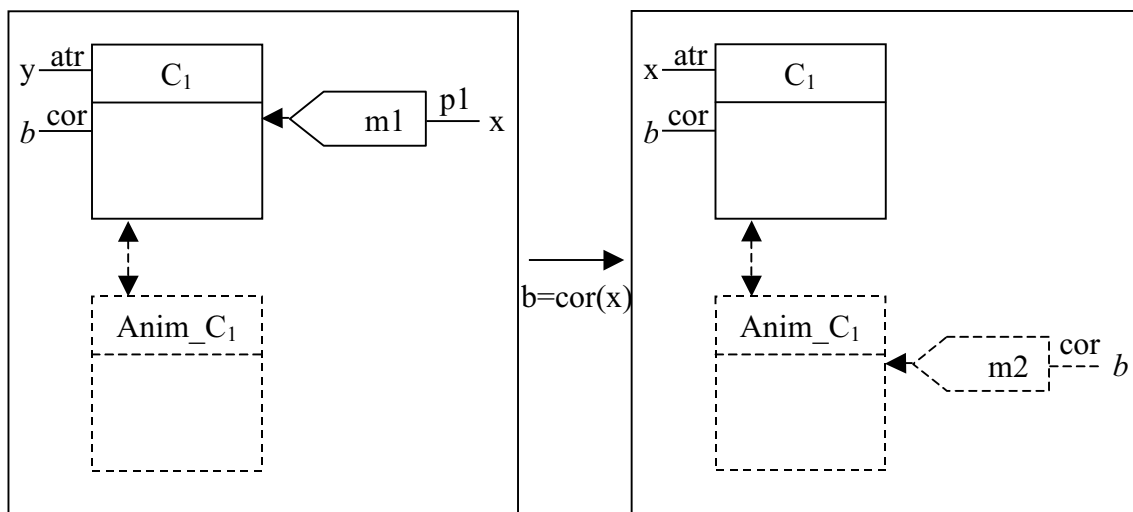


FIGURA 3.5 – Regra em GGa.

3.5 Exemplo: Sistema da Ferrovia

A construção de um modelo de simulação no ambiente PLATUS requer, inicialmente, uma definição das entidades que serão representadas no modelo, através de gramáticas de grafos. A definição visa uma especificação para grafo-tipo, grafo inicial e conjunto de regras de cada entidade que será representada no modelo. Assim, o sistema da ferrovia deverá apresentar definições para três entidades: *Train* (Trem), *RSegm* (Segmento de Trilho) e *Gate* (Portão).

Notação: Para facilitar a representação gráfica das entidades do sistema da ferrovia são feitas distinções apenas quanto aos tipos de vértices e arcos, não há diferencial para os grafos tidos como tipados, ou melhor, grafos que apresentam instâncias de vértices e arcos do grafo-tipo.

- **Entidade *Train*:** Os componentes desta entidade estão representados na figura 3.6 (grafo-tipo), figura 3.7 (grafo inicial) e na figura 3.8 (regras). O grafo-tipo mostra que cada *Train* tem seis atributos: posição corrente (*curr*), uma referência da posição para qual o *Train* pode mover-se (*next*), uma referência para um portão (*gate*), o identificador da próxima posição (*posNext*), o identificador do portão (*posGate*) e o status do atributo que descreve se o *Train* esta parado ou em movimento (*wait*). Os tipos de atributos ADT (*posNext*, *posGate*, *wait*) são estabelecidos a partir do nome do atributo (números naturais (*Natural*), excluindo o zero (*Natural₁*) e os booleanos dos números naturais (*Boolean*)). Os atributos que são referenciados por outros objetos são conectados por arcos a entidade *Train* e as entidades que possuem atributos que pertencem à entidade *Train*. Note que, embora a entidade descrita seja a entidade *Train*, alguns atributos de outras entidades aparecem no grafo-tipo (atributos *station* e *n* da entidade *RSegm*). Assim, ao construir o modelo deve-se escolher uma entidade *RSegm* que tenha estes atributos como atributos instanciados. Todavia, estes atributos podem nem ser lidos tampouco alterados por qualquer regra da entidade *Train*. Entidades *Train* podem receber quatro tipos de mensagens: *Wait*, *Go*, *OpenGate* e *Traveling*. Já as mensagens *MayGo?* e *Leaving* são enviadas pela entidade *Train* para instâncias da entidade *RSegm*, assim como a mensagem *Open* é enviada para as instâncias da entidade *Gate*.

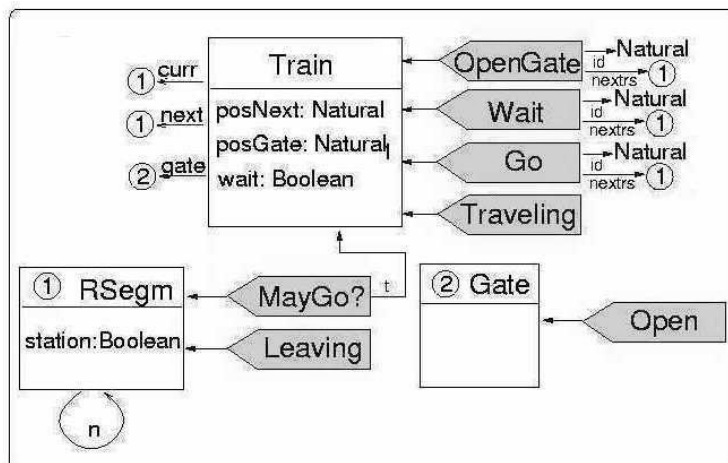
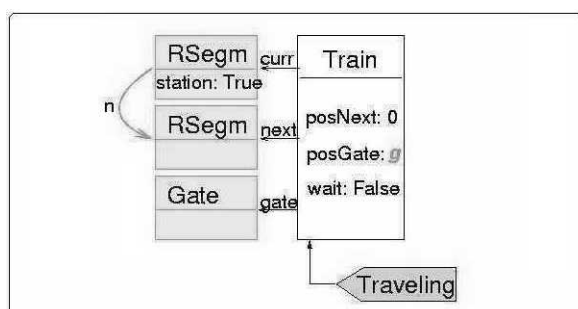
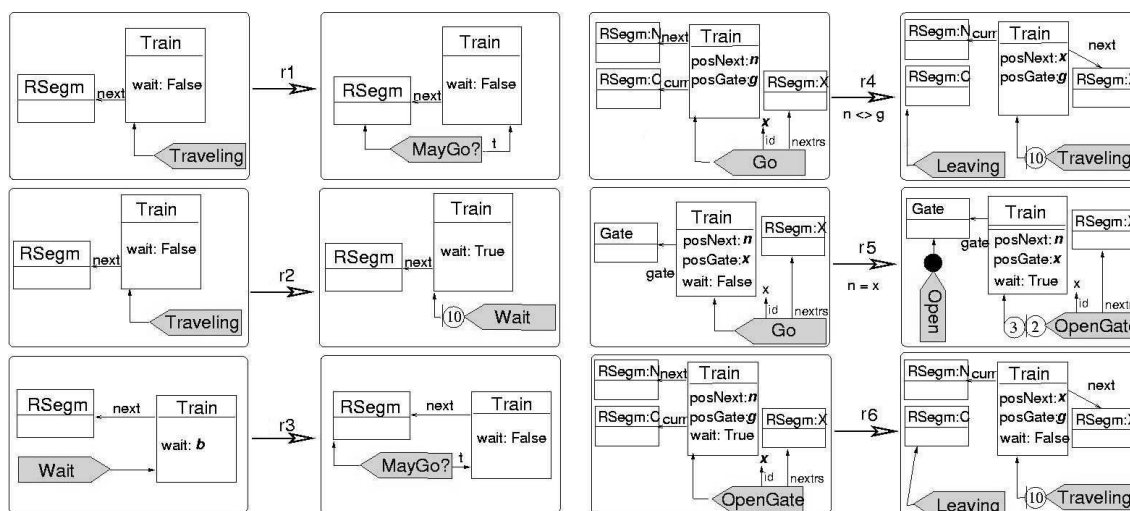


FIGURA 3.6 – Grafo-tipo da entidade *Train*.

O grafo inicial descreve que inicialmente o *Train* deve estar conectado a duas instâncias de *RSegm*, uma instância de *Gate* e ter uma mensagem *Traveling* pendente (esta mensagem irá disparar o movimento do *Train*). Os atributos *posNext* e *wait* são inicializados de 0 e *False*, respectivamente. Para criar um objeto da classe *Train* são necessários dois segmentos de trilho, um portão e um número natural (excluindo zero) que devem ser passados como parâmetros. Todavia, há uma restrição nestes parâmetros: o segmento de trilho designado pelo atributo *curr* deve ter os atributos *station* setado de *True* e *n* setado para outro segmento de trilho passado como parâmetro.

FIGURA 3.7 – Grafo inicial da entidade *Train*.

O comportamento da entidade *Train* é modelado por seis regras. A regra *r1* descreve que quando uma mensagem *Traveling* é recebida, um *Train* tenta viajar para um segmento de trilho apontado pelo atributo *next*. A resposta para mensagem *Traveling* é modelada no lado direito da regra, onde há uma mensagem que responde com uma permissão de entrada para o próximo segmento (*next*). Na regra *r2* a escolha feita é esperar por pelo menos 10 unidades de tempo antes de tentar viajar. Como ambas as regras (*r1* e *r2*) deletam o mesmo atributo, somente uma poderá ser aplicada para cada mensagem. A escolha de qual regra será aplicada é não determinística (veja seção 2.2). A regra *r3* representa o comportamento do sistema quando uma mensagem *Wait* é recebida, *Train* responde com uma permissão para entrar no próximo segmento de trilho. A regra *r4* descreve o movimento do *Train*: atualiza os atributos, envia uma mensagem *Traveling* para se mesmo que será recebida em (pelo menos) dez unidades de tempo e envia uma mensagem para o segmento de trilho avisando que este *Train* está partindo. Note que, para esta regra ser aplicada, tem-se a condição $n \triangleleft g$, que significa que este movimento pode ocorrer somente se não há um *Gate* para entrar na próxima posição. Se há um *Gate*, então a mensagem *Go* será tratada pela regra *r5*, que responde ao *Gate* para abrir imediatamente (a mensagem *Open* é discriminada para chegar exatamente na próxima unidade de tempo (sem atraso)). A aplicação da regra *r5* também irá gerar uma mensagem *OpenGate* para o *Train*, que deve chegar entre duas e três unidades de tempo (o tempo necessário para abrir o *Gate*), e irá disparar a regra *r6*, que irá então mover o *Train* para a próxima posição.

FIGURA 3.8 – Conjunto de regras da entidade *Train*.

- Entidade *Rsegm*:** A gramática de grafos que especifica esta entidade está descrita na figura 3.9. Cada segmento do trilho mantém a informação de seus identificadores, vizinhos, *status* (ocupado ou livre) e sabe se é uma posição ou não. Eles podem reagir às mensagens *MayGo?* (dizendo ao *Train* se este pode mover ou esperar) e *Leaving* (alterando o atributo *busy*). O segmento de trilho pode enviar mensagens do tipo *Go* para as entidades *Train*. Para criar uma instância da entidade *Rsegm*, é necessário fornecer como parâmetros o segmento vizinho, o estado inicial (ocupado ou livre), o identificador e dizer se é uma posição ou não.

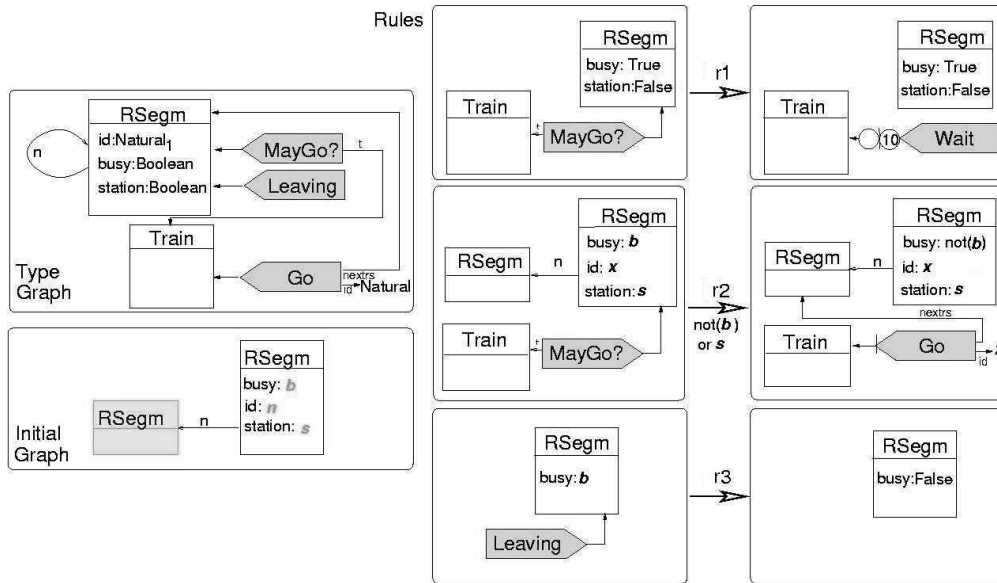


FIGURA 3.9 – Especificação da entidade *Rsegm*.

- Entidade *Gate*:** A especificação desta entidade está descrita na figura 3.10. Entidades *Gate* têm dois atributos: *open* e *stay*. O primeiro atributo descreve se o *Gate* está aberto e o segundo conta as requisições enviadas de *open* para o *Gate*. Note que as entidades *Gate* não apresentam parâmetros instanciados: eles são sempre criados no estado *close* e sem qualquer mensagem *open* pendente. Uma vez o *Gate* aberto, a conclusão, ou melhor, o fechamento do *Gate* deve ocorrer no mínimo dentre dez unidades de tempo. Enquanto o *Gate* está aberto, todas as requisições de *open* causam um atraso no fechamento do *Gate*.

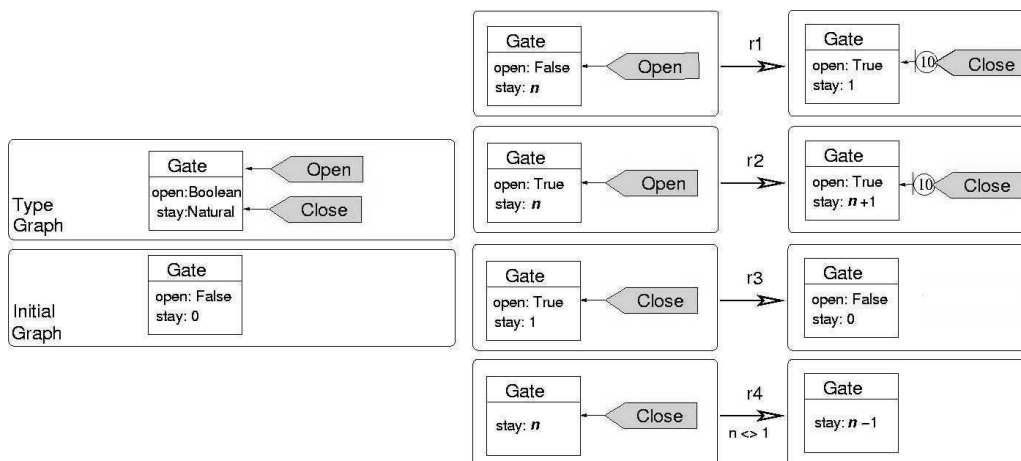


FIGURA 3.10 – Especificação da entidade *Gate*.



3.6 Modelo de simulação da ferrovia

No sistema da ferrovia, por exemplo, é possível criar um modelo como descrito na figura 3.11, no qual há seis segmentos de trilho, dois trens e um portão, podendo ser este o estado inicial da simulação.

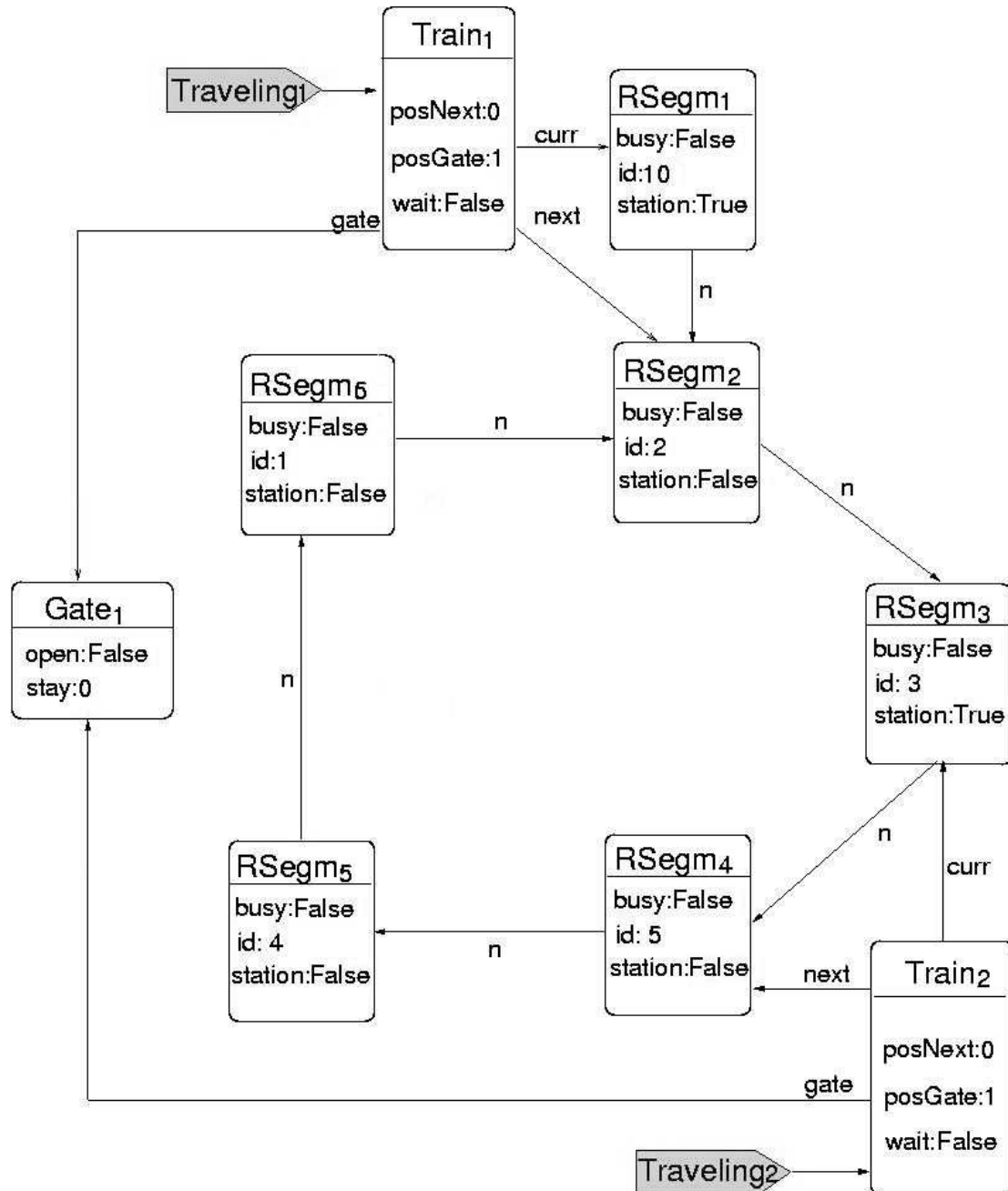


FIGURA 3.11 – Modelo de simulação da ferrovia.

Porém, para construir um modelo de simulação a ser simulado no PLATUS é preciso definir algumas condições. Estas condições precisam ser estabelecidas para que seja possível garantir que o modelo elaborado corresponde ao estado inicial da simulação. As condições podem ser definidas como:

1. As entidades que compõem o modelo devem ser instâncias das entidades definidas na especificação.
2. O modelo deve estar tipado por um grafo definido como grafo-tipo² do modelo.
3. Para garantir que cada entidade foi representada de maneira correta, é preciso construir o estado de cada uma delas e verificar se a representação da entidade feita no modelo, está de acordo com o seu estado. Por exemplo, se o estado da entidade *Train* indica que ela deve estar conectada a dois *RSegm*, um *Gate* e a uma mensagem *Traveling*, esta representação deve ser identificada no modelo.

Assim, para garantir que a figura 3.11 corresponde a um modelo de simulação válido que equivale ao estado inicial do sistema especificado, será necessária uma análise sobre o modelo e as condições que devem ser atendidas (veja o exemplo 3.1).

Exemplo 3.1 Analisando a figura 3.11 é possível observar que as instâncias apresentadas correspondem apenas às entidades que foram definidas na especificação feita na seção 3.5, o que satisfaz a primeira condição. A figura 3.12 apresenta o grafo definido como grafo-tipo do modelo. A partir do grafo-tipo do modelo é possível verificar que o modelo apresentado na figura 3.11 está tipado, satisfazendo a segunda condição. A figura 3.13 (a) apresenta o estado do *Train* (*Train* está conectado a duas entidades *RSegm*, uma entidade *Gate* e uma mensagem *Traveling*), a figura 3.13 (b) apresenta o estado do *RSegm* (*RSegm* está conectado a outro *RSegm*) e a figura 3.13 (c) apresenta o estado do *Gate*. Analisando as condições é possível identificar que o modelo definido na figura 3.11 corresponde a um modelo válido. Pois, todas as entidades instanciadas no modelo foram definidas na especificação, o modelo apresenta vértices (entidades, mensagens, ADT) e arcos (atributos, parâmetros, *time stamps*, destinos) tipados a partir do grafo-tipo do modelo, e os estados das entidades correspondem com o estado de cada entidade representada no modelo.

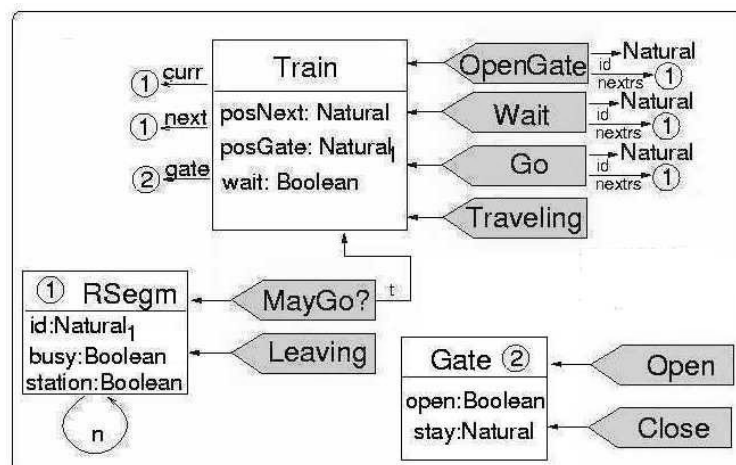


FIGURA 3.12 – Grafo-tipo do modelo.

² O grafo-tipo do modelo é obtido pela união dos grafos-tipos de todas as entidades instanciadas.

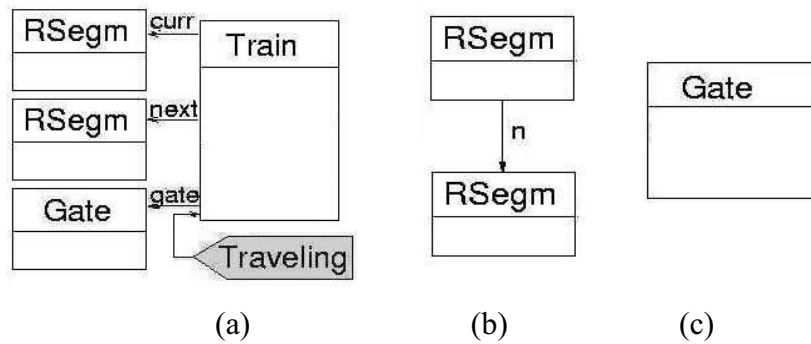


FIGURA 3.13 – Estado do *Train* (a) Estado do *RSegm* (b) Estado do *Gate* (c).

3.7 Simulação

Uma especificação de gramáticas de grafos baseadas em objetos pode ser vista como um conjunto de entidades que são executadas de forma colaborativa. Cada entidade mantém o seu estado particular e comunica-se com outras entidades através de troca de mensagens. O comportamento de cada entidade é descrito por um conjunto de regras. Porém, mais de uma regra pode ser candidata para tratar uma mensagem, sendo que somente uma será considerada *trigger* da mensagem.

No PLATUS um modelo de sistema é definido por um conjunto de entidades, desta forma estas entidades são reunidas no modelo (veja seção 3.6). O modelo de simulação pode ser usado para verificar o comportamento do sistema ou para coletar estatísticas, ou melhor, para analisar a performance ou para aproximar-se de diferentes direções empregadas.

Para que a simulação do modelo seja possível, é preciso se ater a algumas definições: entidade, regras de uma entidade e uma mensagem enviada para entidade. A partir destas definições é importante deixar bem claro que cada estado de uma computação descrita pela gramática de grafos é um grafo que contém instâncias de entidades (com valores concretos de seus atributos), mensagens a serem tratadas, e o tempo corrente. Várias regras podem estar habilitadas, a cada estado de execução, mas nem todas podem ser aplicadas. Assim, a identificação de uma regra como regra habilitada não garante a sua aplicação em um determinado instante de tempo, ou melhor, não define a regra como regra aplicável. Para definir uma regra como regra aplicável é preciso usar o conceito de conflito (como visto na seção 2.2), sendo que se duas regras possuem acesso de escrita em um mesmo atributo é possível afirmar que tais regras estão em conflito. Conflito indica que a aplicação das regras não deve ocorrer num mesmo instante de tempo. Assim, o algoritmo deve realizar uma análise sobre as regras e identificar situações de conflito em tempo de execução, para que a simulação do modelo seja possível.

A próxima seção apresenta detalhes sobre o algoritmo de simulação usado no PLATUS para simular os modelos definidos através de gramáticas de grafos.

3.7.1 Algoritmo

Segundo [RIB2001], o principal objetivo do algoritmo desenvolvido para o PLATUS é assegurar que o formalismo de gramáticas de grafos é fielmente simulado. O formalismo permite que todas as entidades possam ser executadas em um mesmo tempo e uma implementação *multi-thread* parece ser a melhor escolha. Além disso, dentro de uma entidade todas as regras habilitadas não conflitantes podem ser aplicadas em paralelo. Para analisar as dependências entre as regras habilitadas, pode-se construir um conjunto de regras habilitadas não conflitantes a partir de um conjunto arbitrário de regras habilitadas. Mensagens que não podem ser tratadas em um tempo corrente, devido ao fato de estarem em conflito, devem ser posteriormente colocadas novamente. Se não é possível colocar a mensagem novamente (porque o *tmax* foi excedido), a simulação deve ser abortada devido à especificação de erro: não é possível tratar as mensagens no tempo em que elas foram especificadas.

Cada entidade no modelo é mapeada para um processo lógico (LP). O LP é responsável pela seleção das mensagens que podem ser tratadas em um tempo corrente, pela escolha das regras que podem ser ativadas para tratar estas mensagens, e pela seleção do conjunto de regras que serão efetivamente disparadas. Para cada regra disparada, um novo LP será criado. O LP tem uma vida curta, sendo excluído ao término da execução da regra. Isto assegura um alto grau de paralelismo entre as regras que podem ser disparadas em uma mesma entidade (e conseqüentemente em todas as entidades) num ponto específico do tempo.

Para simplificar, entretanto, apenas um relógio centralizado é mantido. Então todas as entidades estão sincronizadas por este relógio e o grau de paralelismo é restrito às regras que ocorrem em um mesmo tempo. Segundo [RIB2001], isto não é muito eficiente se pensar sobre o aspecto de tempo de processamento, mas é suficiente para assegurar uma simulação correta do modelo e é uma boa base para melhorar futuramente o algoritmo.

Uma vez que o relógio é único, tem-se um *kernel*. O *kernel* é uma entidade especial que mantém a lista de eventos (EVL) e o relógio do simulador, sendo também um LP. Quando uma entidade quer enviar uma mensagem para qualquer outra entidade (incluindo ela mesma), essa mensagem deve ser enviada para o *kernel*. O listador irá inserir a mensagem sorteada pelo *time stamp* na EVL. A cada avanço no tempo do relógio, todas as mensagens cujo $tmin(m) \geq T$, onde T é o tempo corrente, serão enviadas para *target* das entidades delas. Assim, a EVL de uma entidade armazena somente mensagens que podem ser selecionadas para serem executadas no tempo corrente. Em cada tempo, todas as entidades esvaziam suas EVLs. O tempo irá avançar somente quando todas as regras disparadas forem executadas por completo.

O algoritmo do simulador é dividido em duas partes. Uma é o *kernel* e a outra compreende o procedimento das entidades.

Algoritmo do *kernel*: As entidades e o *kernel* precisam trocar mensagens. Todas as entidades armazenam uma referência para o *kernel*. Usando estas referências, entidades são capazes de adicionar mensagens num conjunto denominado *buffer* de mensagens (KSBM) do *kernel*. Este conjunto armazena todas as mensagens colocadas pelas entidades desde o último avanço do relógio da simulação. Quando o *kernel* é inicializado, todas as mensagens do sistema inicial devem ser adicionadas neste

conjunto. Se KSBM está vazio para a inicialização da simulação, o *kernel* interrompe a simulação de qualquer aplicação de regra. Do mesmo modo, cada entidade tem seu próprio *buffer* de mensagens (ESBM) e o *kernel* tem mais duas estruturas: um contador (C) e uma lista de eventos (KEVL). O contador (C) é usado para contar o número de mensagens enviadas para as entidades. Este contador é incrementado pelo *kernel* a cada instante de tempo que uma mensagem é colocada e decrementado pela entidade a cada instante que termina a execução de uma regra. A lista de eventos (KEVL) é uma lista que contém todas as mensagens (com *time stamps*) de um tempo futuro.

Dadas estas definições, pode-se descrever o algoritmo do *kernel* iniciado em um instante de tempo $T = 0$, como:

```

1  Construir (KEVL, (Mensagens Iniciais do Sistema))
2  T := Menor(Time stamp(m ∈ KEVL))
3  Enquanto (KEVL ≠ ∅) faça
4      Enquanto (∃ (m ∈ KEVL.tmin(m) ≥ T ))
5          Target(m) // envia m para o Target
6          C := C + 1 // incrementa contador
           /* C é decrementado ao término da execução da regra */
7      Esperar até que C = 0
8      KEVL := KSBM
9      Testar (m ∈ KEVL, Time stamp = T, error)
10     Zerar (listas das entidades)
11     T := Próximo (Time stamp(m ∈ KEVL))
12 Excluir (LPs das Entidades)

```

✓

Algoritmo da Entidade: O comportamento de uma entidade é descrito pelo algoritmo a seguir. O algoritmo implementa cada entidade como uma *thread*, que é iniciada com a chegada de mensagens e finalizada quando todas as mensagens são processadas. Para cada mensagem recebida serão selecionadas regras que tratam a mensagem. A escolha de uma regra dentre as regras selecionadas para tratar a mensagem é feita de forma aleatória, até que todas as regras tenham sido selecionadas. Então, a cada escolha de regra será feita uma análise, sendo que a regra escolhida só será selecionada como regra aplicável se estiver habilitada e não estiver em conflito com nenhuma outra regra já selecionada como regra aplicável à mensagem.

Dadas estas definições, pode-se descrever o algoritmo que implementa a execução da gramática de grafos. Seja o tempo T o valor da variável de tempo corrente do *kernel*:

```

1  Enquanto (Kernel não é notificado (FIM DA SIMULAÇÃO))
2    M := KSBM
3    error := False
4    RegrasAplicáveis :=  $\phi$ 
5    RegraSelecionada :=  $\phi$ 
6    Enquanto ( $M \neq \phi$ ) e (not error) faça
7      Escolha uma mensagem msg de M;
8      M:=M – msg;
9      Construa o conjunto de regras RegrasDeUmaMensagem que tratam msg;
10     Escolha:= False;
11     Enquanto (RegrasDeUmaMensagem  $\neq \phi$ ) e (not Escolha) faça
12       Escolha uma regra  $r \in$  RegrasDeUmaMensagem // seleção randômica
13       Se (regra r not habilitada) ou (conflita com  $r \in$  RegrasAplicáveis)
14         Então RegrasDeUmaMensagem := RegrasDeUmaMensagem – r
15         Senão RegraSelecionada:= r
16         Escolha:= True
17       Se (Escolha = False)
18         Então Se ( $t_{\max}(\text{msg}) > T$ )
19           Então Buffer:= msg (Time stamp(T + tunit,  $t_{\max}(\text{msg})$ ))
20           Senão error := True
21         Senão RegrasAplicáveis:= RegrasAplicáveis  $\cup$  (RegraSelecionada, msg)
22     Se error
23       Então Aborta sistema
24       Senão Executa processo( $\forall(r, \text{msg}) \in$  RegrasAplicáveis)
25     Testar (Finalização de cada Thread, Thread = FIM, C:=C-1)
26     Esperar a finalização de todas as Threads

```

✓

4 Representação de Gramáticas de Grafos

A representação das entidades que compõem os sistemas modelados no PLATUS é feita através do uso de gramáticas de grafos. As gramáticas utilizam grafos para representar os estados e regras para modelar o comportamento dos sistemas.

O ambiente PLATUS é composto por ferramentas independentes, e o problema encontrado no ambiente é de representação, uma vez que as entidades e modelos precisam ser manipulados nas diversas ferramentas de maneira uniforme. A necessidade de obter uma representação das entidades e modelos fiel ao formalismo de gramáticas de grafos que seja eficiente para agilizar a execução do simulador, leva a uma representação não trivial.

As próximas seções apresentam detalhes sobre os critérios que foram analisados para chegar a uma definição da representação das entidades e modelos do PLATUS e por fim apresenta a representação das estruturas. A seção 4.1 apresenta definições de critérios. A seção 4.2 define a representação das estruturas.

4.1 Definição de Critérios

O ambiente PLATUS é composto por ferramentas independentes. As ferramentas permitem a modelagem de entidades, a definição dos elementos da coleta de dados para estatística e a definição dos elementos de visualização da simulação. A partir de todas essas definições, é possível construir modelos na ferramenta de edição de modelos. Para que um modelo do PLATUS possa ser simulado é necessário o uso do conjunto de ferramentas. Assim, é preciso que haja uma padronização na representação. Porém, a representação concreta das entidades do ambiente PLATUS não é algo trivial. A representação precisa ser fiel ao formalismo de gramáticas de grafos e eficiente para agilizar a execução do simulador. Pensando sobre tal aspecto, é preciso definir critérios, que devem ser analisados para que a definição da representação atenda não só ao formalismo e as necessidades do ambiente PLATUS, mas também que permita a conversão dos dados para outros ambientes.

A princípio, foram levantados alguns critérios quanto à representação como:

1. Padronização na representação para todas as ferramentas do PLATUS.
2. Possibilitar uma representação compacta das gramáticas de grafos baseadas em objetos.
3. Viabilizar o tempo de resposta das análises requisitadas pelo simulador.
4. Facilitar a conversão para XML (*Extensible Markup Language*).

Todos os critérios têm seu grau de importância e relevância, que precisam ser analisados cuidadosamente, para que a forma de representação adotada não acabe por favorecer um ou outro critério apenas. Inicialmente, pode-se pensar na possibilidade de obter uma representação tanto na forma de arquivos, como num banco de dados (tabelas). Porém, olhando pelo aspecto da padronização e segundo [KSW95], é necessário um banco de dados para assegurar que informações de um ambiente como o PLATUS, sejam gerenciadas de uma forma consistente. Arquivos não permitem acessos concorrentes, não controlam a redundância, não garantem a integridade dos dados, não

provêm múltiplas visões, enquanto num banco de dados tudo isso é possível. Para uma representação em nível de arquivos, seria necessária uma interface bem elaborada. Pois, todas as restrições seriam estabelecidas através da interface (limitando a especificação). Enquanto que se a representação for feita num banco de dados, muitas das restrições serão alcançadas através da integridade referencial, ou até mesmo por meio de funções definidas no banco. Os sistemas de banco de dados oferecem um grande número de benefícios, que seriam necessários para atender a maioria dos critérios relacionados. Por exemplo, analisando o primeiro critério é possível identificar sem problemas que a escolha do banco de dados como forma de representação seria ideal. Todas as ferramentas estariam manipulando uma mesma base de dados de forma integrada. Porém, o segundo critério exige um pouco mais de análise. Pois, gramáticas de grafos baseadas em objetos impõem algumas restrições às representações. A representação das estruturas de objetos, mesmo complexos, deve ser feita de forma natural e compacta, deve garantir a integridade dos dados (quando um vértice for deletado, os arcos conectados a esse vértice devem ser também deletados). Assim, as deleções necessárias em tabelas correlatas, seriam implementadas de uma forma eficiente em um banco de dados (*triggers*). Enquanto que em arquivos, isso só seria possível através da aplicação. No terceiro critério, é preciso ressaltar que as análises devem acontecer em tempo de execução, as mensagens enviadas não têm uma ordem pré-estabelecida. Para que novas mensagens sejam enviadas é necessário que regras de outras mensagens sejam executadas e analisadas, é preciso selecionar as regras que podem tratar uma mensagem, pois mais de uma regra pode ser aplicável a uma mesma mensagem. As regras selecionadas precisam estar habilitadas (por uma condição ou atributo). Além disso, das regras tidas como habilitadas é preciso que sejam identificadas as que não estão em conflito, uma vez que uma regra só poderá ser aplicada para uma mensagem, caso seja selecionada, esteja habilitada e não esteja em conflito com outras regras em execução, sendo que ao final da análise apenas uma será disparada. Numa representação feita em banco de dados isso poderia ser resolvido através de visões. Visões efetivamente transformam a estrutura de uma ou mais tabelas numa outra estrutura mais apropriada às demandas de aplicação específica.

O fato de XML ser o formato de intercâmbio de dados na *web* torna o quarto critério imprescindível. O desempenho na conversão pode ser obtido igualmente tanto na forma de arquivos quanto em banco de dados, se for levado em consideração apenas a conversão. Mas se o objetivo visa atualização dos dados por intermédio de páginas da *web* com certeza a representação em banco de dados seria a mais indicada.

4.2 Representação das Estruturas

Um grafo (G) pode ser definido como um conjunto de vértices (V) e arcos (A), que possuem uma origem (o) e um destino (d) (ou seja, arcos direcionados). Assim, um grafo pode ser representado por $G = \langle V, A, o, d \rangle$, onde: V é um conjunto de vértices, A é um conjunto de arcos, e $o: A \rightarrow V$ e $d: A \rightarrow V$ são funções totais. Mas, os grafos que serão utilizados no ambiente PLATUS são grafos “especiais”, possuem diferentes tipos de arcos e vértices. Assim, as gramáticas que devem ser utilizadas não podem ser gramáticas que tratam de grafos simples, mas sim, gramáticas que apresentam um mecanismo especial de tipagem. Um dos mecanismos de tipagem de grafos baseia-se em um grafo-tipo, onde a partir do grafo-tipo, é possível definir grafos tipados para representar o grafo inicial e o conjunto de regras da gramática.

O grafo-tipo (GT) utilizado no PLATUS pode ser definido com um conjunto de vértices e arcos. Os vértices representam os nomes das: entidades (E), mensagens (M), tipos abstratos de dados (ADT) e o tempo (T). Arcos representam: atributos (atr) de uma determinada entidade ou parâmetros (par), destinos ($dest$), tempo mínimo/máximo ($time\ stamp$) de uma determinada mensagem, sendo que a determinação da entidade e da mensagem se dá através do direcionamento dos arcos, ou melhor, da definição de origem e destino de cada arco. Para o tipo de vértice identificado como tempo (T), é importante definir que $T = \mathbb{N} \times \mathbb{N}$, sendo que \mathbb{N} representa o conjunto dos números naturais. Assim, o tempo (T) será representado por um par de números naturais.

As definições que serão apresentadas para grafo-tipo e grafos tipados nesta seção serão, em grande parte, compostas por muito mais relações do que funções (relações especiais que garantem naturalmente algumas restrições), uma vez que matematicamente, define-se uma relação como um subconjunto de um produto cartesiano de uma lista de domínios, e essa definição, segundo [SKS97], corresponde quase que exatamente à definição de uma tabela. Assim, como as tabelas em essência são relações, tudo se torna mais fácil ao pensar numa representação em um banco de dados, desta forma é possível representar um grafo-tipo através dos conjuntos V_{tipo} e A_{tipo} (veja exemplo 4.2), ao invés de quatro como na definição clássica (veja exemplo 4.1). O fato das funções o e d não serem apresentadas explicitamente na definição do grafo-tipo não significa que sejam desnecessárias, mas que há uma codificação das informações contidas nas funções o e d como parte dos arcos. Pois, se a representação for feita com a definição clássica será preciso que se façam algumas restrições que não são necessárias na representação através de V_{tipo} e A_{tipo} , sendo que cada arco tem apenas uma origem, uma vez que não é possível mapear o mesmo elemento do domínio para dois da imagem, e na representação clássica, para obter este efeito seria necessário colocar restrições nos arcos.

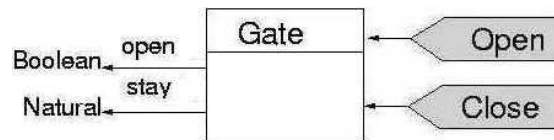


FIGURA 4.1 – Grafo-tipo da entidade *Gate*.

Exemplo 4.1 A figura 4.1 mostra a representação gráfica do **grafo-tipo** da entidade *Gate*, que será definida através da definição clássica, onde um grafo pode ser representado por $G = \langle V, A, o, d \rangle$, sendo: V um conjunto de vértices, A um conjunto de arcos, e $o: A \rightarrow V$ e $d: A \rightarrow V$ funções totais.

$$V = \{ \langle Gate \rangle, \langle Open \rangle, \langle Close \rangle, \langle Boolean \rangle, \langle Natural \rangle \}$$

$$A = \{ \langle open \rangle, \langle stay \rangle, \langle a_1 \rangle, \langle a_2 \rangle \}$$

$$o = \{ \langle open \rangle \rightarrow \langle Gate \rangle, \\ \langle stay \rangle \rightarrow \langle Gate \rangle, \\ \langle a_1 \rangle \rightarrow \langle Open \rangle, \\ \langle a_2 \rangle \rightarrow \langle Close \rangle \}$$

$$d = \{ \langle open \rangle \rightarrow \langle Boolean \rangle, \\ \langle stay \rangle \rightarrow \langle Natural \rangle, \\ \langle a_1 \rangle \rightarrow \langle Gate \rangle, \\ \langle a_2 \rangle \rightarrow \langle Gate \rangle \}$$

✓

O conjunto $Vtipo$ é um subconjunto do produto cartesiano³ Vid (identificador de vértices) $\times T_V$ (tipos de vértices) $\times T_{GG}$ (tipos de gramáticas de grafos), onde cada vértice representado em Vid deve estar associado a um único tipo de vértice de T_V e a um único tipo de gramáticas de grafos de T_{GG} . O conjunto T_{GG} é um conjunto composto por tipos de gramáticas de grafos, que identificam vértices e arcos usados nas diferentes gramáticas (veja seção 3.2). Assim, $T_{GG} = \{A, B, S\}$, onde A representa vértices e arcos da animação, B representa do comportamento e S da estatística.

O conjunto $Atipo$ é um subconjunto do produto cartesiano Aid (identificadores dos arcos) $\times T_A$ (tipos dos arcos) $\times T_{GG}$ (tipos das gramáticas) $\times Vtipo$ (vértice origem) $\times Vtipo$ (vértice destino), onde cada arco representado em Aid deve estar associado a apenas um tipo de arco de T_A , a um único tipo de gramática de T_{GG} , a um vértice origem e a um vértice destino. Além disso, para garantir a consistência dos arcos utilizados no PLATUS, é preciso que sejam estabelecidas algumas restrições: se o arco é do tipo atributo, é preciso restringir que o tipo do vértice de origem seja uma entidade e o tipo do vértice de destino seja do tipo entidade ou ADT; se o arco é do tipo parâmetro, a origem deve ser identificada com vértices do tipo mensagem e o destino pode ser uma entidade ou um ADT; arcos do tipo destino têm origem em uma mensagem e destino em uma entidade; arcos do tipo *time stamp* têm origem em um tipo de vértice especificado como tempo e destino em um vértice do tipo mensagem. Além disso, é preciso garantir que uma mensagem deve ter um único destino especificado.

Exemplo 4.2 A figura 4.1 mostra a representação gráfica do **grafo-tipo** da entidade *Gate*, que será definida através da definição desenvolvida neste trabalho, onde um grafo pode ser representado por $G = \langle Vtipo, Atipo \rangle$, sendo: $Vtipo$ um conjunto de vértices, $Atipo$ um conjunto de arcos que apresentam uma codificação das funções o e d .

$$Vtipo = \{ \langle Gate, \underline{E}, \underline{B} \rangle, \langle Open, \underline{M}, \underline{B} \rangle, \langle Close, \underline{M}, \underline{B} \rangle, \langle Boolean, \underline{ADT}, \underline{B} \rangle, \langle Natural, \underline{ADT}, \underline{B} \rangle \}$$

$$Atipo = \{ \langle open, \underline{atr}, \underline{B}, \langle Gate, \underline{E}, \underline{B} \rangle, \langle Boolean, \underline{ADT}, \underline{B} \rangle \rangle, \langle stay, \underline{atr}, \underline{B}, \langle Gate, \underline{E}, \underline{B} \rangle, \langle Natural, \underline{ADT}, \underline{B} \rangle \rangle, \langle a_1, \underline{dest}, \underline{B}, \langle Open, \underline{M}, \underline{B} \rangle, \langle Gate, \underline{E}, \underline{B} \rangle \rangle, \langle a_2, \underline{dest}, \underline{B}, \langle Close, \underline{M}, \underline{B} \rangle, \langle Gate, \underline{E}, \underline{B} \rangle \rangle \}$$

$$\begin{array}{ccc} \uparrow & & \uparrow \\ origem & & destino \end{array}$$

Notação: Para facilitar a definição foram usadas formatações diferenciadas na representação dos conjuntos. O sublinhado indica uma referência feita aos tipos de vértices, arcos e gramáticas de grafos, e o negrito destaca a origem e o destino dos arcos, sendo que tanto origem quanto destino devem estar definidos em $Vtipo$. Assim, cada vértice de origem/destino definido em $Atipo$ é um vértice tipado de $Vtipo$.

✓

³ Representado pelo \times , o produto cartesiano (ou simplesmente produto) de dois (ou mais) conjuntos é o conjunto de todos os pares (tuplas) ordenados possíveis tal que, em cada par (tupla), o primeiro elemento vem do primeiro conjunto e o segundo elemento vem do segundo conjunto (e assim por diante).

Definição 4.1 (Grafo-tipo) Um **grafo-tipo** é uma tupla $GT = \langle V_{tipo}, A_{tipo} \rangle$ onde:

$V_{id} =$ conjunto de vértices que representam nomes de entidades, mensagens, tipos abstratos de dados e o tempo.

$A_{id} =$ conjunto de arcos que representam nomes de atributos, parâmetros ou simplesmente o destino ou o time stamp de uma mensagem.

$$T_V = \{ \underline{E}, \underline{M}, \underline{ADT}, \underline{T} \}$$

$$T_A = \{ \underline{atr}, \underline{par}, \underline{dest}, \underline{time\ stamp} \}$$

$$T_{GG} = \{ \underline{A}, \underline{B}, \underline{S} \}$$

- $V_{tipo} \subseteq V_{id} \times T_V \times T_{GG}$ tal que $\forall v \in V_{id}; \forall t_1, t_2 \in T_V; \forall t_{GG1}, t_{GG2} \in T_{GG}$

i) Cada vértice só tem um tipo de vértice e um tipo de gramática:

$$\langle v, t_1, t_{GG1} \rangle \in V_{tipo} \wedge \langle v, t_2, t_{GG2} \rangle \in V_{tipo} \Rightarrow t_1 = t_2 \wedge t_{GG1} = t_{GG2}$$

- $A_{tipo} \subseteq A_{id} \times T_A \times T_{GG} \times V_{tipo} \times V_{tipo}$ tal que $\forall a \in A_{id}; \forall v_o, v_d \in V_{id}; \forall t_o, t_d \in T_V; \forall t_{GGo}, t_{GGd} \in T_{GG}; \forall o_1, o_2, d_1, d_2 \in V_{tipo}$

i) Cada arco tem apenas um tipo de arco, um tipo de gramática, uma origem e um destino:

$$\langle a, t_1, t_{GG1}, o_1, d_1 \rangle \in A_{tipo} \wedge \langle a, t_2, t_{GG2}, o_2, d_2 \rangle \in A_{tipo} \Rightarrow t_1 = t_2 \wedge t_{GG1} = t_{GG2} \wedge o_1 = o_2 \wedge d_1 = d_2$$

ii) Origem e destino de um atributo são consistentes:

$$\langle a, \underline{atr}, t_{GG}, \langle v_o, t_o, t_{GGo} \rangle, \langle v_d, t_d, t_{GGd} \rangle \rangle \in A_{tipo} \Rightarrow t_o = \underline{E} \wedge (t_d = \underline{E} \vee t_d = \underline{ADT})$$

iii) Origem e destino de um parâmetro são consistentes:

$$\langle a, \underline{par}, t_{GG}, \langle v_o, t_o, t_{GGo} \rangle, \langle v_d, t_d, t_{GGd} \rangle \rangle \in A_{tipo} \Rightarrow t_o = \underline{M} \wedge (t_d = \underline{E} \vee t_d = \underline{ADT})$$

iv) Origem e destino de uma mensagem são consistentes:

$$\langle a, \underline{dest}, t_{GG}, \langle v_o, t_o, t_{GGo} \rangle, \langle v_d, t_d, t_{GGd} \rangle \rangle \in A_{tipo} \Rightarrow t_o = \underline{M} \wedge t_d = \underline{E}$$

v) **Origem e destino do time stamp são consistentes:**

$$\langle a, \underline{\text{time stamp}}, t_{GG}, \langle v_o, t_o, t_{GGo} \rangle, \langle v_d, t_d, t_{GGd} \rangle \rangle \in \text{Atipo} \Rightarrow t_o = \underline{T} \wedge t_d = \underline{M}$$

vi) **Cada mensagem tem exatamente um destinatário:**

$$\langle v_o, \underline{M}, t_{GG} \rangle \in \text{Vtipo} \Rightarrow \exists! \langle a, \underline{\text{dest}}, t_{GG}, \langle v_o, \underline{M}, t_{GGo} \rangle, \langle v_d, \underline{E}, t_{GGd} \rangle \rangle \in \text{Atipo}$$

✓

Exemplo 4.3 (Grafo-Tipo) A figura 3.6 mostra a representação gráfica do **grafo-tipo** da entidade *Train* definido neste exemplo.

$$\text{Vid} = \{\text{Train}, \text{RSegm}, \text{Gate}, \text{Natural}, \text{Natural}_1, \text{Boolean}, \text{OpenGate}, \text{Wait}, \text{Go}, \text{Traveling}, \text{MayGo?}, \text{Leaving}, \text{Open}\}$$

$$\text{Aid} = \{\text{curr}, \text{next}, \text{gate}, \text{posNext}, \text{posGate}, \text{wait}, \text{id}_{og}, \text{nexts}_{og}, \text{id}_w, \text{nexts}_w, \text{id}_g, \text{nexts}_g, \text{station}, \text{n}, \text{t}, \text{dest}_{og}, \text{dest}_w, \text{dest}_g, \text{dest}_t, \text{dest}_{mg}, \text{dest}_l, \text{dest}_o\}$$

$$\begin{aligned} \text{Vtipo} = \{ & \langle \text{Train}, \underline{E}, \underline{B} \rangle, \\ & \langle \text{RSegm}, \underline{E}, \underline{B} \rangle, \\ & \langle \text{Gate}, \underline{E}, \underline{B} \rangle, \\ & \langle \text{Natural}, \underline{\text{ADT}}, \underline{B} \rangle, \\ & \langle \text{Natural}_1, \underline{\text{ADT}}, \underline{B} \rangle, \\ & \langle \text{Boolean}, \underline{\text{ADT}}, \underline{B} \rangle, \\ & \langle \text{OpenGate}, \underline{M}, \underline{B} \rangle, \\ & \langle \text{Wait}, \underline{M}, \underline{B} \rangle, \\ & \langle \text{Go}, \underline{M}, \underline{B} \rangle, \\ & \langle \text{Traveling}, \underline{M}, \underline{B} \rangle, \\ & \langle \text{MayGo?}, \underline{M}, \underline{B} \rangle, \\ & \langle \text{Leaving}, \underline{M}, \underline{B} \rangle, \\ & \langle \text{Open}, \underline{M}, \underline{B} \rangle \} \end{aligned}$$

$$\begin{aligned} \text{Atipo} = \{ & \langle \text{curr}, \text{atr}, \underline{B}, \langle \text{Train}, \underline{E}, \underline{B} \rangle, \langle \text{RSegm}, \underline{E}, \underline{B} \rangle \rangle, \\ & \langle \text{next}, \text{atr}, \underline{B}, \langle \text{Train}, \underline{E}, \underline{B} \rangle, \langle \text{RSegm}, \underline{E}, \underline{B} \rangle \rangle, \\ & \langle \text{gate}, \text{atr}, \underline{B}, \langle \text{Train}, \underline{E}, \underline{B} \rangle, \langle \text{Gate}, \underline{E}, \underline{B} \rangle \rangle, \\ & \langle \text{posNext}, \text{atr}, \underline{B}, \langle \text{Train}, \underline{E}, \underline{B} \rangle, \langle \text{Natural}, \underline{\text{ADT}}, \underline{B} \rangle \rangle, \\ & \langle \text{posGate}, \text{atr}, \underline{B}, \langle \text{Train}, \underline{E}, \underline{B} \rangle, \langle \text{Natural}_1, \underline{\text{ADT}}, \underline{B} \rangle \rangle, \\ & \langle \text{wait}, \text{atr}, \underline{B}, \langle \text{Train}, \underline{E}, \underline{B} \rangle, \langle \text{Boolean}, \underline{\text{ADT}}, \underline{B} \rangle \rangle, \\ & \langle \text{id}_{og}, \text{par}, \underline{B}, \langle \text{OpenGate}, \underline{M}, \underline{B} \rangle, \langle \text{Natural}, \underline{\text{ADT}}, \underline{B} \rangle \rangle, \\ & \langle \text{nexts}_{og}, \text{par}, \underline{B}, \langle \text{OpenGate}, \underline{M}, \underline{B} \rangle, \langle \text{RSegm}, \underline{E}, \underline{B} \rangle \rangle, \\ & \langle \text{id}_w, \text{par}, \underline{B}, \langle \text{Wait}, \underline{M}, \underline{B} \rangle, \langle \text{Natural}, \underline{\text{ADT}}, \underline{B} \rangle \rangle, \\ & \langle \text{nexts}_w, \text{par}, \underline{B}, \langle \text{Wait}, \underline{M}, \underline{B} \rangle, \langle \text{RSegm}, \underline{E}, \underline{B} \rangle \rangle, \end{aligned}$$

$$\begin{aligned}
&\langle id_g, \underline{par}, \underline{B}, \langle \underline{Go}, \underline{M}, \underline{B} \rangle, \langle \underline{Natural}, \underline{ADT}, \underline{B} \rangle \rangle, \\
&\langle nexts_g, \underline{par}, \underline{B}, \langle \underline{Go}, \underline{M}, \underline{B} \rangle, \langle \underline{RSegm}, \underline{E}, \underline{B} \rangle \rangle, \\
&\langle station, \underline{atr}, \underline{B}, \langle \underline{RSegm}, \underline{E}, \underline{B} \rangle, \langle \underline{Boolean}, \underline{ADT}, \underline{B} \rangle \rangle, \\
&\langle n, \underline{atr}, \underline{B}, \langle \underline{RSegm}, \underline{E}, \underline{B} \rangle, \langle \underline{RSegm}, \underline{E}, \underline{B} \rangle \rangle, \\
&\langle t, \underline{par}, \underline{B}, \langle \underline{MayGo?}, \underline{M}, \underline{B} \rangle, \langle \underline{Train}, \underline{E}, \underline{B} \rangle \rangle, \\
&\langle dest_{og}, \underline{dest}, \underline{B}, \langle \underline{OpenGate}, \underline{M}, \underline{B} \rangle, \langle \underline{Train}, \underline{E}, \underline{B} \rangle \rangle, \\
&\langle dest_w, \underline{dest}, \underline{B}, \langle \underline{Wait}, \underline{M}, \underline{B} \rangle, \langle \underline{Train}, \underline{E}, \underline{B} \rangle \rangle, \\
&\langle dest_g, \underline{dest}, \underline{B}, \langle \underline{Go}, \underline{M}, \underline{B} \rangle, \langle \underline{Train}, \underline{E}, \underline{B} \rangle \rangle, \\
&\langle dest_t, \underline{dest}, \underline{B}, \langle \underline{Traveling}, \underline{M}, \underline{B} \rangle, \langle \underline{Train}, \underline{E}, \underline{B} \rangle \rangle, \\
&\langle dest_{mg}, \underline{dest}, \underline{B}, \langle \underline{MayGo?}, \underline{M}, \underline{B} \rangle, \langle \underline{RSegm}, \underline{E}, \underline{B} \rangle \rangle, \\
&\langle dest_l, \underline{dest}, \underline{B}, \langle \underline{Leaving}, \underline{M}, \underline{B} \rangle, \langle \underline{RSegm}, \underline{E}, \underline{B} \rangle \rangle, \\
&\langle dest_o, \underline{dest}, \underline{B}, \langle \underline{Open?}, \underline{M}, \underline{B} \rangle, \langle \underline{Gate}, \underline{E}, \underline{B} \rangle \rangle \}
\end{aligned}$$

Notação: Para simplificar a representação, cada linha representa um vértice de V_{tipo} e um arco de $Atipo$.

✓

A partir da definição do grafo-tipo é possível definir todos os grafos que estarão representando os estados do sistema: estes grafos constituem-se de instâncias de elementos do grafo-tipo, respeitando as relações de coexistência dos arcos. Assim, é possível definir o grafo inicial que é composto por instâncias dos tipos especificados no grafo-tipo, onde em cada entidade, todos os seus atributos devem ser especificados, e em cada mensagem, todos os seus parâmetros também devem ser especificados de acordo com a definição do grafo-tipo. Estes grafos serão chamados de grafos tipados ou simplesmente grafos.

O conjunto V_G é um subconjunto do produto cartesiano $Vid_G \times V_{tipo}$, lembre que $V_{tipo} = Vid \times T_V \times T_{GG}$, de forma que todos os vértices representados em Vid_G devem estar associados a um único tipo de vértice de V_{tipo} , ou melhor, a um vértice tipado pertencente a V_{tipo} .

Já o conjunto A_G é um subconjunto do produto cartesiano $Aid_G \times Atipo \times V_G \times V_G$, lembre que $Atipo = Aid \times T_A \times T_{GG} \times V_G \times V_G$, de forma que todos os arcos representados em Aid_G devem estar associados a um tipo de arco de $Atipo$ (arco tipado pertencente a $Atipo$), a um vértice origem e a um vértice destino, sendo que tanto o vértice de destino como o de origem pertencem a V_G (logo são resultantes de um produto cartesiano ($Vid_G \times V_{tipo}$)). Mas para garantir a consistência dos arcos é preciso que sejam estabelecidas algumas restrições, permitindo a cada arco apenas um único tipo, uma única origem e um único destino. Além desta restrição que estabelece limites a todos os arcos, é necessário restringir um pouco mais, de acordo com a definição do grafo-tipo. Uma mensagem, por exemplo, só pode ter um único destino e deve possuir todos os parâmetros que foram definidos no grafo-tipo. Como as mensagens, as entidades também devem possuir todos os atributos definidos no grafo-tipo, caso o grafo a ser especificado, seja o grafo inicial.

Assim, como tudo depende da especificação do grafo-tipo, são utilizadas algumas funções auxiliares que facilitam a obtenção de informações como: tipo do vértice, tipo do arco, nome de um vértice ou nome de um arco, que são necessárias para obtenção do conjunto A_G .

Para definir grafos tipados serão utilizadas algumas funções auxiliares:

$$F_{\text{tipo}}: V_{\text{tipo}} \rightarrow T_V, \text{ onde: } F_{\text{tipo}}(\langle \text{vid}, t_v, t_{GG} \rangle) = t_v$$

$$F_{\text{tipoMsg}}: A_{\text{tipo}} \rightarrow T_A, \text{ onde: } F_{\text{tipoMsg}}(\langle \text{aid}, t_a, t_{GG}, v_o, v_d \rangle) = t_a$$

$$\text{NomeV}: V_{\text{tipo}} \rightarrow \text{Vid}, \text{ onde: } \text{NomeV}(\langle \text{vid}, t_v, t_{GG} \rangle) = \text{vid}$$

$$\text{Nome}: A_{\text{tipo}} \rightarrow \text{Aid}, \text{ onde: } \text{Nome}(\langle \text{aid}, t_a, t_{GG}, v_o, v_d \rangle) = \text{aid}$$

✓

Definição 4.2 Grafo (ou Grafo Tipado) Seja $GT = \langle V_{\text{tipo}}, A_{\text{tipo}} \rangle$ um grafo-tipo. Um grafo tipado sobre GT é um par $G = \langle V_G, A_G \rangle$ onde:

$Vid_G =$ conjunto de vértices que representam as identidades das instâncias dos tipos.

$Aid_G =$ conjunto de arcos que representam as identidades das instâncias dos atributos, parâmetros, destinos e time stamps concretos de entidades e mensagens.

- $V_G \subseteq Vid_G \times V_{\text{tipo}}$ tal que

i) **Cada vértice tem apenas um tipo:**

$$\langle v, t_1 \rangle \in V_G \wedge \langle v, t_2 \rangle \in V_G \Rightarrow t_1 = t_2$$

- $A_G \subseteq Aid_G \times A_{\text{tipo}} \times V_G \times V_G$ tal que

i) **Tipo, origem e destino de um arco são consistentes:**

$$\langle a, t_1, o_1, d_1 \rangle, \langle a, t_2, o_2, d_2 \rangle \in A_G \Rightarrow t_1 = t_2 \wedge o_1 = o_2 \wedge d_1 = d_2$$

ii) **Tipos são consistentes com o grafo-tipo:**

$$\langle a, \langle at, t_b, t_{GGt}, o_b, d_t \rangle, \langle v_{og}, t_{og} \rangle, \langle v_{dg}, t_{dg} \rangle \rangle \in A_G \Rightarrow t_{og} = o_t \wedge t_{dg} = d_t$$

iii) **Mensagens possuem um único destino:**

$$\forall \langle m, mt \rangle \in V_G \wedge F_{\text{tipo}}(mt) = \underline{M} \Rightarrow \exists! \langle a, t_a, o, d \rangle \in A_G \text{ tal que } o = \langle m, mt \rangle \wedge F_{\text{tipoMsg}}(t_a) = \underline{\text{dest}}$$

iv) *Mensagens possuem todos os parâmetros definidos no grafo-tipo:*

$$\forall \langle m, mt \rangle \in V_G \wedge Ftipo(mt) = \underline{M}: partipo = \langle a, \underline{par}, t_{GG}, \langle v_o, t_o, t_{GG} \rangle, \langle v_d, t_d, t_{GG} \rangle \rangle \in Atipo \wedge NomeV(mt) = v_o \Rightarrow \exists \langle a_g, t_g, o_g, d_g \rangle \in A_G \text{ tal que } t_g = partipo \wedge o_g = \langle m, mt \rangle$$

✓

Exemplo 4.4 (Grafo) A figura 3.7 mostra a representação gráfica do **grafo** inicial da entidade *Train* definido neste exemplo.

$$Vid_G = \{Train_1, RSegm_1, RSegm_2, Gate_1, Traveling_1, 0, g, True, False\}$$

$$Aid_G = \{curr_1, next_1, gate_1, posNext_1, posGate_1, wait_1, n_1, station_1, dest_{t1}\}$$

$$V_G = \{ \langle Train_1, \langle \underline{Train}, \underline{E}, \underline{B} \rangle \rangle, \\ \langle RSegm_1, \langle \underline{RSegm}, \underline{E}, \underline{B} \rangle \rangle, \\ \langle RSegm_2, \langle \underline{RSegm}, \underline{E}, \underline{B} \rangle \rangle, \\ \langle Gate_1, \langle \underline{Gate}, \underline{E}, \underline{B} \rangle \rangle, \\ \langle Traveling_1, \langle \underline{Traveling}, \underline{M}, \underline{B} \rangle \rangle, \\ \langle 0, \langle \underline{Natural}, \underline{ADT}, \underline{B} \rangle \rangle, \\ \langle g, \langle \underline{Natural}_1, \underline{ADT}, \underline{B} \rangle \rangle, \\ \langle True, \langle \underline{Boolean}, \underline{ADT}, \underline{B} \rangle \rangle \} \\ \langle False, \langle \underline{Boolean}, \underline{ADT}, \underline{B} \rangle \rangle \}$$

$$A_G = \{ \langle curr_1, \langle curr, \underline{atr}, \underline{B}, \langle \underline{Train}, \underline{E}, \underline{B} \rangle, \langle \underline{RSegm}, \underline{E}, \underline{B} \rangle \rangle, \langle Train_1, \langle \underline{Train}, \underline{E}, \underline{B} \rangle \rangle, \langle RSegm_1, \langle \underline{RSegm}, \underline{E}, \underline{B} \rangle \rangle \rangle, \\ \langle next_1, \langle next, \underline{atr}, \underline{B}, \langle \underline{Train}, \underline{E}, \underline{B} \rangle, \langle \underline{RSegm}, \underline{E}, \underline{B} \rangle \rangle, \langle Train_1, \langle \underline{Train}, \underline{E}, \underline{B} \rangle \rangle, \langle RSegm_2, \langle \underline{RSegm}, \underline{E}, \underline{B} \rangle \rangle \rangle, \\ \langle gate_1, \langle gate, \underline{atr}, \underline{B}, \langle \underline{Train}, \underline{E}, \underline{B} \rangle, \langle \underline{Gate}, \underline{E}, \underline{B} \rangle \rangle, \langle Train_1, \langle \underline{Train}, \underline{E}, \underline{B} \rangle \rangle, \langle Gate_1, \langle \underline{Gate}, \underline{E}, \underline{B} \rangle \rangle \rangle, \\ \langle posNext_1, \langle posNext, \underline{atr}, \underline{B}, \langle \underline{Train}, \underline{E}, \underline{B} \rangle, \langle \underline{Natural}, \underline{ADT}, \underline{B} \rangle \rangle, \langle Train_1, \langle \underline{Train}, \underline{E}, \underline{B} \rangle \rangle, \langle 0, \langle \underline{Natural}, \underline{ADT}, \underline{B} \rangle \rangle \rangle, \\ \langle posGate_1, \langle posGate, \underline{atr}, \underline{B}, \langle \underline{Train}, \underline{E}, \underline{B} \rangle, \langle \underline{Natural}_1, \underline{ADT}, \underline{B} \rangle \rangle, \langle Train_1, \langle \underline{Train}, \underline{E}, \underline{B} \rangle \rangle, \langle g, \langle \underline{Natural}_1, \underline{ADT}, \underline{B} \rangle \rangle \rangle, \\ \langle wait_1, \langle wait, \underline{atr}, \underline{B}, \langle \underline{Train}, \underline{E}, \underline{B} \rangle, \langle \underline{Boolean}, \underline{ADT}, \underline{B} \rangle \rangle, \langle Train_1, \langle \underline{Train}, \underline{E}, \underline{B} \rangle \rangle, \langle False, \langle \underline{Boolean}, \underline{ADT}, \underline{B} \rangle \rangle \rangle, \\ \langle n_1, \langle n, \underline{atr}, \underline{B}, \langle \underline{RSegm}, \underline{E}, \underline{B} \rangle, \langle \underline{RSegm}, \underline{E}, \underline{B} \rangle \rangle, \langle RSegm_1, \langle \underline{RSegm}, \underline{E}, \underline{B} \rangle \rangle, \langle RSegm_2, \langle \underline{RSegm}, \underline{E}, \underline{B} \rangle \rangle \rangle, \\ \langle station_1, \langle station, \underline{atr}, \underline{B}, \langle \underline{RSegm}, \underline{E}, \underline{B} \rangle, \langle \underline{Boolean}, \underline{ADT}, \underline{B} \rangle \rangle, \langle RSegm_1, \langle \underline{RSegm}, \underline{E}, \underline{B} \rangle \rangle, \langle True, \langle \underline{Boolean}, \underline{ADT}, \underline{B} \rangle \rangle \rangle, \\ \langle dest_{t1}, \langle dest, \underline{dest}, \underline{B}, \langle \underline{Traveling}, \underline{M}, \underline{B} \rangle, \langle \underline{Train}, \underline{E}, \underline{B} \rangle \rangle, \langle Traveling_1, \langle \underline{Traveling}, \underline{M}, \underline{B} \rangle \rangle, \langle Train_1, \langle \underline{Train}, \underline{E}, \underline{B} \rangle \rangle \rangle \}$$

Notação: Para simplificar a representação do conjunto A_G , cada arco pertencente ao conjunto $Atipo$ que especifica o tipo de um arco de A_G é representado com uma formatação diferenciada.

✓

Segundo [RIB00], um grafo inicial especifica um estado inicial do sistema (dentro da especificação de uma entidade), e este estado pode ser especificado abstratamente, e tornará se concreto quando for construído um modelo de simulação contendo todas as instâncias de entidades envolvidas no sistema.

Definição 4.3 (Grafo Estado) Um **grafo estado** é um par $G = \langle V_G, A_G \rangle$ onde:

i) **Entidades possuem todos os atributos definidos no grafo-tipo:**

$$\forall \langle e, et \rangle \in V_G \wedge Ftipo(et) = \underline{E}: atrtipo = \langle a, \underline{atr}, t_{GG}, \langle v_o, t_o, t_{GG} \rangle, \langle v_d, t_d, t_{GG} \rangle \rangle \in Atipo \wedge NomeV(et) = v_o \Rightarrow \exists \langle a_g, t_g, o_g, d_g \rangle \in A_G \text{ tal que } t_g = atrtipo \wedge o_g = \langle e, et \rangle$$

✓

Definição 4.4 (Grafo Tipado Parcial) Dado um grafo tipado $G = \langle V_G, A_G \rangle$, um **grafo tipado parcial** sobre G é qualquer tupla $GP = \langle V_{GP}, A_{GP} \rangle$ onde $V_{GP} \subseteq V_G$ e $A_{GP} \subseteq A_G$.

✓

Para definir regras, é preciso saber que uma regra é formada por grafos tipados que representam o lado esquerdo (L) e o lado direito da regra (R). Uma regra descreve a reação da entidade ao receber uma determinada mensagem, ou melhor, o comportamento do sistema. Esta reação é descrita em termos de mudança de valores de atributos e envio de novas mensagens. Cada regra pode ser vista como uma composição formada por:

- **Lado esquerdo da regra (L)** representa a situação necessária para habilitar a ocorrência da regra, desta forma, quanto mais elementos do lado esquerdo, maiores as restrições à ativação da regra. L é um grafo tipado contendo:
 - uma única mensagem com todos os seus parâmetros (exceto *time stamp*);
 - atributos da entidade que recebe a mensagem, sendo que não é necessária a representação dos valores de todos os atributos, apenas dos que são relevantes para o tratamento da mensagem;
 - atributos do tipo ADT podem ser representados por variáveis a partir das quais podem ser ativadas as operações especificadas para este ADT.
- **Lado direito da regra (R)** é um grafo tipado que:
 - pode apresentar ou não mensagens, mas se apresentar é necessário que todos os parâmetros da mensagem sejam especificados, inclusive o *time stamp*;
 - pode apresentar várias mensagens e entidades.

- **Condições (e)** são equações que representam condições que devem ser verdadeiras para a regra ser aplicada. As condições são posicionadas, em geral, sob a seta que referencia a regra.
- **Mapeamento (M)** é o morfismo entre grafos tipados que:
 - define itens que são preservados, deletados e criados pela aplicação da regra;
 - necessariamente a mensagem contida em L não deve estar em R (pode haver novas instâncias deste tipo de mensagem, mas a mensagem tratada deve ser consumida);
 - todos os atributos das entidades que aparecem em L devem aparecer em R (possivelmente com valores modificados). Isto evita que um atributo de uma entidade “desapareça”;
 - não podem aparecer em R entidades que não estão em L .

Assim, para que a representação da regra seja feita de acordo com as definições do grafo-tipo e obedecendo ao formalismo de gramáticas de grafos, são necessárias algumas validações, para os grafos L , R e o mapeamento $L \rightarrow R$.

Considerando que o mapeamento $L \rightarrow R$ é formado por vértices e arcos que são preservados (aparecem em L e R), criados (não aparecem em L e aparecem em R) ou deletados (aparecem em L e não aparecem em R) é possível chegar a uma definição da regra. Assim, o lado esquerdo da regra seria definido através da união dos vértices (V_{GD}) e arcos (A_{GD}) deletados e o grafo tipado (P), que representa a parte preservada da regra. Logo, o lado esquerdo da regra seria representado por $L = (V_{GL}, A_{GL})$. Já o lado direito seria definido através da união dos vértices (V_{GC}) e arcos (A_{GC}) criados e o grafo tipado (P), onde R seria representado por $R = (V_{GR}, A_{GR})$. Chama-se de grafo-núcleo a união $L \cup R$.

Definição 4.5 (Regra) Seja o $GT = \langle V_{tipo}, A_{tipo} \rangle$ um grafo-tipo e $G = \langle V_G, A_G \rangle$ um grafo tipado sobre GT , chamado grafo-núcleo, então uma **regra** tipada com GT é uma quintupla $r = \langle id, D, C, P, e \rangle$ onde:

- id é um identificador da regra
- o conjunto de itens deletados D é um grafo tipado parcial sobre G
- o conjunto de itens criados C é um grafo tipado parcial sobre G
- o conjunto de itens preservados P é um subgrafo de G
- e é uma string (representada por uma expressão booleana, que não será tratada neste trabalho)

tal que

$$i) \quad D \cap C = P$$

$$ii) \quad L = D \cup P \text{ é um subgrafo de } G, \text{ chamado lado esquerdo da regra.}$$

iii) $R = C \cup P$ é um subgrafo de G , chamado lado direito da regra.

iv) ***L tem apenas uma mensagem:***

$$\exists! \langle m, mt \rangle \in V_{GL} \Rightarrow Ftipo(mt) = \underline{M}$$

v) ***L só possui atributos da entidade que recebe a mensagem:***

$$\exists! \langle a, \langle aid, \underline{dest}, t_{GG}, v_o, v_d \rangle, \langle m, mt \rangle, \langle e, et \rangle \rangle \in A_{GL} \Rightarrow \forall \langle a, \langle aid, \underline{atr}, t_{GG}, v_o, v_d \rangle, \langle e', et' \rangle, \langle vd, td \rangle \rangle \in A_{GL} \wedge \langle e, et \rangle = \langle e', et' \rangle$$

vi) ***Mensagem de L não tem time stamp:***

$$\langle m, mt \rangle \in V_{GL} \wedge Ftipo(mt) = \underline{M} : \not\exists \langle a_g, t_g, o_g, d_g \rangle \in A_{GL} \text{ tal que } t_g = \underline{time stamp} \wedge o_g = \langle m, mt \rangle, \text{ onde } \underline{time stamp} = \langle a, \underline{time stamp}, t_{GG}, \langle v_o, t_o, t_{GG} \rangle, \langle v_d, t_d, t_{GG} \rangle \rangle \in A_{tipo} \wedge NomeV(mt) = v_o$$

vii) ***Mensagem de L é deletada:***

$$\langle m, mt \rangle \in V_{GL} \wedge Ftipo(mt) = \underline{M} \Rightarrow \langle m, mt \rangle \in V_{GD}$$

viii) ***Atributos das entidades não podem ser deletados:***

$$\forall \langle a, \langle aid, \underline{atr}, t_{GG}, v_o, v_d \rangle, \langle e, et \rangle, \langle vd, td \rangle \rangle \in A_{GL} \wedge Ftipo(et) = \underline{E} \Rightarrow \not\exists \langle a, \langle aid, \underline{atr}, t_{GG}, v_o, v_d \rangle, \langle e, et \rangle, \langle vd, td \rangle \rangle \in A_{GD}$$

ix) ***Todas as entidades são preservadas:***

$$\forall \langle e, et \rangle \in V_{GL} \wedge Ftipo(et) = \underline{E} \Rightarrow \langle e, et \rangle \in V_{GR}$$

✓

Exemplo 4.5 (Regra) A figura 3.8 mostra a representação gráfica da regra r_1 da Entidade *Train* definida neste exemplo.

$$V_{GD} = \{ \langle \text{Traveling}_1, \langle \text{Traveling}, \underline{M}, \underline{B} \rangle \rangle \}$$

$$A_{GD} = \{ \langle \text{dest}_{t1}, \langle \text{dest}_b, \underline{dest}, \underline{B}, \langle \text{Traveling}, \underline{M}, \underline{B} \rangle, \langle \text{Train}, \underline{E}, \underline{B} \rangle \rangle, \langle \text{Traveling}_1, \langle \text{Traveling}, \underline{M}, \underline{B} \rangle \rangle, \langle \text{Train}_1, \langle \text{Train}, \underline{E}, \underline{B} \rangle \rangle \}$$

$$V_{GC} = \{ \langle \text{MayGo?}_1, \langle \text{MayGo?}, \underline{M}, \underline{B} \rangle \rangle \}$$

$$A_{GC} = \{ \langle \text{dest}_{mg1}, \langle \text{dest}_{mg}, \underline{dest}, \underline{B}, \langle \text{MayGo?}, \underline{M}, \underline{B} \rangle, \langle \text{RSegm}, \underline{E}, \underline{B} \rangle \rangle, \langle \text{MayGo?}_1, \langle \text{MayGo?}, \underline{M}, \underline{B} \rangle \rangle, \langle \text{RSegm}_2, \langle \text{RSegm}, \underline{E}, \underline{B} \rangle \rangle, \langle t_1, \langle t, \underline{par}, \underline{B}, \langle \text{MayGo?}, \underline{M}, \underline{B} \rangle, \langle \text{Train}, \underline{E}, \underline{B} \rangle \rangle, \langle \text{MayGo?}_1, \langle \text{MayGo?}, \underline{M}, \underline{B} \rangle, \langle \text{Train}_1, \underline{B}, \langle \text{Train}, \underline{E}, \underline{B} \rangle \rangle \}$$

$$V_{GP} = \{ \langle \text{Train}_1, \langle \text{Train}, \underline{E}, \underline{B} \rangle \rangle, \\ \langle \text{RSegm}_2, \langle \text{RSegm}, \underline{E}, \underline{B} \rangle \rangle, \\ \langle \text{False}, \langle \text{Boolean}, \underline{ADT}, \underline{B} \rangle \rangle \}$$

$$A_{GP} = \{ \langle \text{wait}_1, \langle \text{wait}, \underline{atr}, \underline{B}, \langle \text{Train}, \underline{E}, \underline{B} \rangle, \langle \text{Boolean}, \underline{ADT}, \underline{B} \rangle \rangle, \langle \text{Train}_1, \\ \langle \text{Train}, \underline{E}, \underline{B} \rangle \rangle, \langle \text{False}, \langle \text{Boolean}, \underline{ADT}, \underline{B} \rangle \rangle \rangle, \\ \langle \text{next}_1, \langle \text{next}, \underline{atr}, \underline{B}, \langle \text{Train}, \underline{E}, \underline{B} \rangle, \langle \text{RSegm}, \underline{E}, \underline{B} \rangle \rangle, \langle \text{Train}_1, \\ \langle \text{Train}, \underline{E}, \underline{B} \rangle \rangle, \langle \text{RSegm}_2, \langle \text{RSegm}, \underline{E}, \underline{B} \rangle \rangle \rangle \}$$

$$e = \varepsilon$$

$$V_{GL} = \{ \langle \text{Train}_1, \langle \text{Train}, \underline{E}, \underline{B} \rangle \rangle, \\ \langle \text{RSegm}_2, \langle \text{RSegm}, \underline{E}, \underline{B} \rangle \rangle, \\ \langle \text{Traveling}_1, \langle \text{Traveling}, \underline{M}, \underline{B} \rangle \rangle \}$$

$$A_{GL} = \{ \langle \text{dest}_{t1}, \langle \text{dest}_b, \underline{dest}, \underline{B}, \langle \text{Traveling}, \underline{M}, \underline{B} \rangle, \langle \text{Train}, \underline{E}, \underline{B} \rangle \rangle, \\ \langle \text{Traveling}_1, \langle \text{Traveling}, \underline{M}, \underline{B} \rangle \rangle, \langle \text{Train}_1, \langle \text{Train}, \underline{E}, \underline{B} \rangle \rangle \rangle, \\ \langle \text{wait}_1, \langle \text{wait}, \underline{atr}, \underline{B}, \langle \text{Train}, \underline{E}, \underline{B} \rangle, \langle \text{Boolean}, \underline{ADT}, \underline{B} \rangle \rangle, \langle \text{Train}_1, \\ \langle \text{Train}, \underline{E}, \underline{B} \rangle \rangle, \langle \text{False}, \langle \text{Boolean}, \underline{ADT}, \underline{B} \rangle \rangle \rangle, \\ \langle \text{next}_1, \langle \text{next}, \underline{atr}, \underline{B}, \langle \text{Train}, \underline{E}, \underline{B} \rangle, \langle \text{RSegm}, \underline{E}, \underline{B} \rangle \rangle, \langle \text{Train}_1, \\ \langle \text{Train}, \underline{E}, \underline{B} \rangle \rangle, \langle \text{RSegm}_2, \langle \text{RSegm}, \underline{E}, \underline{B} \rangle \rangle \rangle \}$$

$$V_{GR} = \{ \langle \text{Train}_1, \langle \text{Train}, \underline{E}, \underline{B} \rangle \rangle, \\ \langle \text{RSegm}_2, \langle \text{RSegm}, \underline{E}, \underline{B} \rangle \rangle, \\ \langle \text{MayGo?}_1, \langle \text{MayGo?}, \underline{M}, \underline{B} \rangle \rangle \}$$

$$A_{GR} = \{ \langle \text{dest}_{mg1}, \langle \text{dest}_{mg}, \underline{dest}, \underline{B}, \langle \text{MayGo?}, \underline{M}, \underline{B} \rangle, \langle \text{RSegm}, \underline{E}, \underline{B} \rangle \rangle, \\ \langle \text{MayGo?}_1, \langle \text{MayGo?}, \underline{M}, \underline{B} \rangle \rangle, \langle \text{RSegm}_2, \langle \text{RSegm}, \underline{E}, \underline{B} \rangle \rangle \rangle, \\ \langle t_1, \langle t, \underline{par}, \underline{B}, \langle \text{MayGo?}, \underline{M}, \underline{B} \rangle, \langle \text{Train}, \underline{E}, \underline{B} \rangle \rangle, \langle \text{MayGo?}_1, \\ \langle \text{MayGo?}, \underline{M}, \underline{B} \rangle \rangle, \langle \text{Train}_1, \underline{B}, \langle \text{Train}, \underline{E}, \underline{B} \rangle \rangle \rangle, \\ \langle \text{wait}_1, \langle \text{wait}, \underline{atr}, \underline{B}, \langle \text{Train}, \underline{E}, \underline{B} \rangle, \langle \text{Boolean}, \underline{ADT}, \underline{B} \rangle \rangle, \\ \langle \text{Train}_1, \langle \text{Train}, \underline{E}, \underline{B} \rangle \rangle, \langle \text{False}, \langle \text{Boolean}, \underline{ADT}, \underline{B} \rangle \rangle \rangle, \\ \langle \text{next}_1, \langle \text{next}, \underline{atr}, \underline{B}, \langle \text{Train}, \underline{E}, \underline{B} \rangle, \langle \text{RSegm}, \underline{E}, \underline{B} \rangle \rangle, \langle \text{Train}_1, \\ \langle \text{Train}, \underline{E}, \underline{B} \rangle \rangle, \langle \text{RSegm}_2, \langle \text{RSegm}, \underline{E}, \underline{B} \rangle \rangle \rangle \}$$

✓

Como visto na seção 2.1, uma gramática de grafos é composta de um grafo-tipo, um grafo inicial e um conjunto de regras. Assim, para definir a gramática de grafos de uma determinada entidade é preciso definir o grafo-tipo da entidade, um grafo de estado inicial que representa o estado inicial da entidade, um conjunto de regras que estabelece as possíveis mudanças de estado da entidade, e um identificador que representa a entidade. Desta forma, é possível definir também que uma entidade é uma gramática de grafos que só apresenta um tipo de objeto, ou melhor, é uma classe de entidade.

Definição 4.6 (Gramática de Grafos) Dado o nome de uma entidade E , um grafo-tipo $GT = \langle V_{tipo}, A_{tipo} \rangle$, um grafo de estado inicial $GE = \langle V_{GE}, A_{GE} \rangle$ e um conjunto de regras $N = \{r_1 \dots r_n\}$. Uma **gramática de grafos** tipada sobre GT é uma quádrupla $GG = (E, GT, GE, N)$, onde E é um identificador.

✓

Exemplo 4.6 (Gramática Grafos) Utilizando as definições feitas nos exemplos 4.3, 4.4 e 4.5 é possível definir a **gramática de grafos** da entidade *Train* neste exemplo. O exemplo 4.3 apresenta a definição do grafo-tipo (GT) do *Train*, o exemplo 4.4 define o grafo de estado inicial (GE) do *Train* e o exemplo 4.5 apresenta a definição de uma das regras (r_1) da entidade *Train*. Assumindo que todas as outras regras da entidade *Train* representadas na figura 3.8, serão definidas como a regra (r_1), foi possível definir o conjunto de regras de *Train* (N).

$$E = Train$$

$$GT = \langle V_{tipo}, A_{tipo} \rangle$$

$$GE = \langle V_G, A_G \rangle$$

$$N = \{r_1, r_2, r_3, r_4, r_5, r_6\}$$

✓

Definição 4.7 (Entidade) Uma **entidade** E é uma gramática de grafos, onde só há regras para tratar mensagens enviadas a um dos tipos de objetos do grafo-tipo. Este é o tipo de objeto especificado por esta entidade.

✓

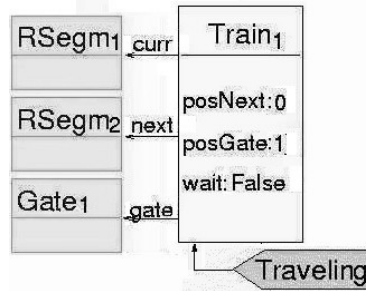
Partindo da definição de que uma entidade E é uma gramática de grafos que descreve um único tipo de objeto, é possível definir uma instância da entidade. A instância da entidade pode ser definida através de um identificador que diferencia uma instância da outra, uma lista de valores que serão atribuídos a todas as instâncias de atributos da entidade e um conjunto de instâncias de mensagens destinadas à entidade.

Definição 4.8 (Instância da Entidade) Dada uma entidade $E = (id, GT, GE, N)$, uma **instância da entidade** é uma tupla $IE = (id_{IE}, G, M)$ onde id_{IE} é um identificador da instância da entidade e G um mapeamento associando valores para todos os atributos da entidade e M é um conjunto de instâncias de mensagens com destino id_{IE} . Uma instância de mensagem IM é um par $IM = (id_M, P)$ onde id_M é um identificador de instâncias de mensagem e P é um mapeamento associando valores a todos os parâmetros da mensagem.

Observação: Um mapeamento é um conjunto de pares (veja exemplo 4.7), onde os valores associados podem ser pertencentes a tipos de dados pré-definidos ou a identificadores de instâncias.

✓

Exemplo 4.7 (Instância da Entidade) A figura 4.2 apresenta uma representação gráfica de uma **instância da entidade** *Train*, onde: $id_{IE} = Train_1$, $G = \{ \langle posNext, 0 \rangle, \langle posGate, 1 \rangle, \langle wait, False \rangle, \langle curr, Rsegm_1 \rangle, \langle next, Rsegm_2 \rangle, \langle gate, Gate_1 \rangle \}$ e $M = \{ \langle Traveling1, \emptyset \rangle \}$

FIGURA 4.2 – Instância de *Train*.

A construção de um modelo requer, inicialmente, uma definição das entidades que serão representadas no modelo. Após a definição das entidades é possível criar as instâncias que apresentam valores concretos e que serão reunidas no modelo de acordo com o conjunto de entidades, representando o estado inicial do sistema. O conjunto de entidades representa as gramáticas das diversas entidades instanciadas no modelo, visando que cada entidade seja representada de acordo com a sua gramática. Por exemplo, se a entidade *Train* for representada no modelo pode-se ter uma instância da entidade (veja exemplo 4.7) e essa instância deve estar conectada a dois *Rsegm*, um *Gate* e a uma mensagem *Traveling*.

Definição 4.9 (Modelo) Um **modelo** é uma tupla $M_o = \langle GG_E, C_{IE} \rangle$, onde:

- GG_E : é um conjunto de entidades.
- C_{IE} : é um conjunto de instâncias de entidades de GG_E .

Exemplo 4.8 (Modelo) A figura 3.11 apresenta uma representação gráfica de um **modelo** de ferrovia definido neste exemplo.

$$GG_E = (E_{(Train)}, E_{(RSegm)}, E_{(Gate)})$$

$$C_{IE} = ((Train_1, \{ \langle posNext, 0 \rangle, \langle posGate, 1 \rangle, \langle wait, False \rangle, \langle curr, Rsegm_1 \rangle, \langle next, Rsegm_2 \rangle, \langle gate, Gate_1 \rangle \}, \{ \langle Traveling_1, \emptyset \rangle \}), \\ (Train_2, \{ \langle posNext, 0 \rangle, \langle posGate, 1 \rangle, \langle wait, False \rangle, \langle curr, Rsegm_3 \rangle, \langle next, Rsegm_4 \rangle, \langle gate, Gate_1 \rangle \}, \{ \langle Traveling_2, \emptyset \rangle \}), \\ (RSegm_1, \{ \langle busy, False \rangle, \langle id, 10 \rangle, \langle station, True \rangle, \langle n, Rsegm_2 \rangle \}, \emptyset), \\ (Rsegm_2, \{ \langle busy, False \rangle, \langle id, 2 \rangle, \langle station, False \rangle, \langle n, Rsegm_3 \rangle \}, \emptyset), \\ (Rsegm_3, \{ \langle busy, False \rangle, \langle id, 3 \rangle, \langle station, True \rangle, \langle n, Rsegm_4 \rangle \}, \emptyset), \\ (Rsegm_4, \{ \langle busy, False \rangle, \langle id, 5 \rangle, \langle station, False \rangle, \langle n, Rsegm_5 \rangle \}, \emptyset), \\ (Rsegm_5, \{ \langle busy, False \rangle, \langle id, 4 \rangle, \langle station, False \rangle, \langle n, Rsegm_6 \rangle \}, \emptyset), \\ (Rsegm_6, \{ \langle busy, False \rangle, \langle id, 1 \rangle, \langle station, False \rangle, \langle n, Rsegm_2 \rangle \}, \emptyset), \\ (Gate_1, \{ \langle open, False \rangle, \langle stay, 0 \rangle \}, \emptyset))$$

5 Análise da Recuperação das Informações

Com a idealização deste trabalho surgiram questões não só quanto ao armazenamento das especificações feitas através de gramáticas de grafos, mas também quanto à recuperação. PLATUS é um ambiente de simulação, o que torna a questão da recuperação das informações tão importante quanto o armazenamento. Não basta definir uma forma de representação das informações, é preciso garantir que a recuperação será feita de forma eficiente, visando agilizar a ação do simulador.

Num ambiente de simulação o próximo estado de um sistema depende da última execução, ou melhor, as regras que serão selecionadas como regras aplicáveis a uma determinada mensagem dependem do estado gerado pela última regra aplicada. Assim, é necessário realizar um exame detalhado, uma observação minuciosa sobre a representação das informações, para garantir que a recuperação das informações será feita de forma rápida e condizente com o estado atual do sistema.

Regras são especificadas através de instâncias das entidades que reagem a mensagens recebidas. Entidades apresentam atributos que podem ser definidos por variáveis. As variáveis são instanciadas no momento de execução da regra, onde pode ser que em um determinado estado, não exista uma instanciação possível, i.e., dois atributos podem estar associados a uma mesma variável na regra, e no estado atual esses atributos apresentam valores diferenciados. A instanciação numa regra não é necessária apenas para definir o valor dos atributos da entidade, mas também para checar a condição de uma regra (que normalmente envolve variáveis), bem como para gerar o próximo estado, ou melhor, o lado direito da regra (a mesma instanciação do lado esquerdo será utilizada no lado direito da regra na hora de aplicá-la).

A cada mensagem enviada é necessário realizar uma análise que determine o conjunto consistente de regras aplicáveis ao estado atual do sistema. A análise deve ser realizada em tempo de execução, sendo que dentre as regras tidas como regras de uma mensagem (regras que apresentam no lado esquerdo um tipo de mensagem igual ao tipo de mensagem enviada), é preciso determinar quais regras podem ser consideradas como habilitadas (regras que apresentam correspondência de valores entre todos os atributos e parâmetros presentes no lado esquerdo da regra, com atributos e parâmetros presentes no estado atual do sistema) e dentre as regras tidas como habilitadas quais não estão em conflito (regras que não compartilham vértices tidos como deletados com nenhuma outra regra já definida como regra aplicável à mensagem), gerando assim um conjunto consistente de regras aplicáveis a uma determinada mensagem.

As próximas seções apresentam detalhes e exemplos da análise realizada a partir de uma mensagem e um conjunto de regras, onde as regras poderão ser caracterizadas como: regras de uma mensagem, regras habilitadas e regras conflitantes.

5.1 Regras de uma mensagem

A partir de informações como: entidade (recebe a mensagem) e tipo de mensagem, que são fornecidas como dados de entrada, é possível realizar uma análise sobre o conjunto de regras (N) da entidade para identificar se há regras para tratar tal tipo de mensagem. Assim, dentre as regras de N todas serão analisadas, mas apenas as

regras que apresentarem em V_{GD} (conjunto de vértices consumidos pela regra) um vértice com o tipo de mensagem fornecido serão inseridas no conjunto *RegrasDeUmaMensagem* (veja exemplo 5.1). Afinal, uma mensagem recebida é uma mensagem que deve ser tratada, ou melhor, consumida pela regra.

Função *RegrasDeUmaMensagem*:

Dados de Entrada: Entidade $E = (id_E, GT, GE, N)$, Tipo de mensagem $tipo_msg = (id_M, \underline{M}, \underline{T_{GG}})$, onde id_M é um identificador, \underline{M} representa o tipo de vértice (mensagem) e $\underline{T_{GG}}$ representa o tipo de gramática de grafos (veja definição 4.1).

Dados de Saída: Conjunto de identificadores de regras que tratam o tipo de mensagem (*RegrasDeUmaMensagem*).

Definição: $\forall r = (id, \langle V_{GD}, A_{GD} \rangle, C, P, e) \in N:$

$$(vid, tipo_msg) \in V_{GD} \Rightarrow id \in \text{RegrasDeUmaMensagem}$$

Complexidade: polinomial

Para implementar esta função, são necessárias, no máximo, $|N| \times |V_{GD}|$ comparações, onde N é o conjunto de regras da entidade, e V_{GD} é o maior conjunto de vértices consumidos por uma regra de N . É necessário percorrer o conjunto N uma vez para calcular a saída desta função.

✓

Exemplo 5.1 (Regras de uma mensagem) A partir da entidade E e do tipo da mensagem $tipo_msg$ é possível obter um conjunto de regras *RegrasDeUmaMensagem* que podem tratar a mensagem enviada. Dentre as regras de *Train* foi possível selecionar duas regras como regras de uma mensagem, sendo essas regras identificadas como r_1 e r_2 . As regras r_1 e r_2 foram selecionadas por apresentarem no conjunto de vértices deletados, um vértice que possuía o mesmo tipo especificado como tipo da mensagem (*Traveling*, \underline{M} , \underline{B}) enviada a entidade *Train*.

Dados de Entrada: $E = (Train, GT_{(Train)}, GE_{(Train)}, \{r_1, r_2, r_3, r_4, r_5, r_6\})$, $tipo_msg = (Traveling, \underline{M}, \underline{B})$

- **Regra r_1 :**

$$V_{GD1} = \{\langle Traveling_1, \langle Traveling, \underline{M}, \underline{B} \rangle \rangle\}$$

$$(r_1, \langle V_{GD1}, A_{GD1} \rangle, C, P, e): (Traveling_1, \langle Traveling, \underline{M}, \underline{B} \rangle) \in V_{GD1} \Rightarrow r_1 \in \text{RegrasDeUmaMensagem}$$

- **Regra r_2 :**

$$V_{GD2} = \{ \langle \text{Traveling}_2, \langle \text{Traveling}, \underline{\mathbf{M}}, \underline{\mathbf{B}} \rangle \rangle \}$$

$$(r_2, \langle V_{GD2}, A_{GD2} \rangle, C, P, e): (\text{Traveling}_2, \langle \text{Traveling}, \underline{\mathbf{M}}, \underline{\mathbf{B}} \rangle) \in V_{GD2} \Rightarrow r_2 \in \text{RegrasDeUmaMensagem}$$

- **Regra r_3 :**

$$V_{GD3} = \{ \langle \text{Wait}_1, \langle \text{Wait}, \underline{\mathbf{M}}, \underline{\mathbf{B}} \rangle \rangle \}$$

$$(r_3, \langle V_{GD3}, A_{GD3} \rangle, C, P, e): (\text{Wait}_1, \langle \text{Traveling}, \underline{\mathbf{M}}, \underline{\mathbf{B}} \rangle) \notin V_{GD3} \Rightarrow r_3 \notin \text{RegrasDeUmaMensagem}$$

- **Regra r_4 :**

$$V_{GD4} = \{ \langle \text{Go}_1, \langle \text{Go}, \underline{\mathbf{M}}, \underline{\mathbf{B}} \rangle \rangle \}$$

$$(r_4, \langle V_{GD4}, A_{GD4} \rangle, C, P, e): (\text{Go}_1, \langle \text{Traveling}, \underline{\mathbf{M}}, \underline{\mathbf{B}} \rangle) \notin V_{GD4} \Rightarrow r_4 \notin \text{RegrasDeUmaMensagem}$$

- **Regra r_5 :**

$$V_{GD5} = \{ \langle \text{Go}_2, \langle \text{Go}, \underline{\mathbf{M}}, \underline{\mathbf{B}} \rangle \rangle \}$$

$$(r_5, \langle V_{GD5}, A_{GD5} \rangle, C, P, e): (\text{Go}_2, \langle \text{Traveling}, \underline{\mathbf{M}}, \underline{\mathbf{B}} \rangle) \notin V_{GD5} \Rightarrow r_5 \notin \text{RegrasDeUmaMensagem}$$

- **Regra r_6 :**

$$V_{GD6} = \{ \langle \text{OpenGate}_1, \langle \text{OpenGate}, \underline{\mathbf{M}}, \underline{\mathbf{B}} \rangle \rangle \}$$

$$(r_6, \langle V_{GD6}, A_{GD6} \rangle, C, P, e): (\text{OpenGate}_1, \langle \text{Traveling}, \underline{\mathbf{M}}, \underline{\mathbf{B}} \rangle) \notin V_{GD6} \Rightarrow r_6 \notin \text{RegrasDeUmaMensagem}$$

Dados de Saída: $\text{RegrasDeUmaMensagem} = \{r_1, r_2\}$

✓

Porém, para uma regra poder ser aplicada em um determinado estado, a informação obtida da análise das regras de uma mensagem não é suficiente: dentre as regras selecionadas como regras de uma mensagem é preciso identificar quais regras estão habilitadas, ou melhor, quais regras apresentam atributos, parâmetros e/ou condições⁴ que são satisfeitos pelo estado atual da entidade. Buscando um novo conjunto de regras identificado como *RegrasAplicáveis*, serão realizadas análises com as regras do conjunto de *RegrasDeUmaMensagem* e o estado atual, onde as regras serão classificadas em habilitadas ou desabilitadas. Assim, foi definida uma função que verifica se uma determinada regra está habilitada ou não em um estado.

⁴ A validação da condição da regra não será abordada neste trabalho.

5.2 Regra habilitada

A regra está habilitada? Essa questão certamente envolve muito mais do que uma simples análise, que identifica a presença de um tipo de mensagem no lado esquerdo da regra. O fato de uma regra tratar uma mensagem passada como parâmetro não garante que a regra pode ser aplicada. A aplicação de uma regra depende também do estado atual do sistema, ou melhor, é preciso que haja uma correspondência entre todos os atributos e parâmetros presentes no lado esquerdo da regra, com atributos e parâmetros instanciados no estado atual do sistema.

Tendo uma instância da entidade, uma regra e uma lista de parâmetros de uma mensagem como dados de entrada, é possível caracterizar a regra como habilitada ou desabilitada.

Função RegraHabilitada:

Dados de Entrada: Instância $IE = (id_{IE}, G, M)$, Regra $r = (id, D, C, P, e)$, Lista de parâmetros da mensagem par_msg , onde cada elemento da lista é um par $(tipopar, valor)$.

Dados de Saída: Booleano, Tabela (lista) de atribuições feitas para as variáveis tab_var , onde cada elemento da tabela é um par $(var, valor)$.

A definição desta função, bem como sua complexidade, serão mostradas nas seções 5.2.1 e 5.2.2, respectivamente.

5.2.1 Definição

O objetivo desta função é analisar a regra e caracteriza-la em habilitada ou desabilitada para o estado atual do sistema. Para isso, foi esboçado um algoritmo, que está estruturado em duas partes, visando os tipos diferentes de atributos e parâmetros encontrados nas gramáticas de grafos baseadas em objeto, além das diferentes atribuições feitas para as variáveis encontradas.

Gramáticas de grafos baseadas em objeto apresentam arcos que podem representar valores de atributos e parâmetros, sendo que esses valores só podem ser especificados com o tipo de vértice ADT ou instância de Entidade. Para executar o algoritmo, construiu-se quatro conjuntos:

X_{ADT} : contém os atributos da regra que apresentam valores do tipo ADT;

Y_{ADT} : contém os parâmetros que apresentam valores do tipo ADT;

X_E : contém os atributos da regra que apresentam valores do tipo instância de Entidade;

Y_E : contém os parâmetros que apresentam valores do tipo instância de Entidade.

Os elementos destes conjuntos que representam valores tipo ADT terão o formato $\langle \text{tipo}, \text{vid} \rangle$, onde *tipo* é o nome do atributo ou parâmetro, e *vid* o seu valor. Esse valor é um elemento de algum conjunto carregador da álgebra nesta entidade. No caso de regras, a álgebra pode ser uma álgebra de termos, permitindo assim a utilização de variáveis e expressões (termos) como atributos ou parâmetros. O uso de variáveis é bastante interessante para uma representação mais compacta da especificação (ao invés de se escrever uma regra para cada valor concreto, escreve-se uma genérica usando variáveis). No momento de se aplicar uma regra, deve-se instanciar essas variáveis com valores concretos e verificar se essa instanciação corresponde aos valores existentes no estado corrente (formalmente, isso corresponde a verificar se existe um homomorfismo entre a álgebra da regra e a do estado atual). Quando não existe uma instanciação possível para as variáveis da regra que corresponda aos valores existentes no estado atual, a regra não está habilitada (veja o exemplo 5.3). Assim, não basta que a regra seja caracterizada como habilitada ou desabilitada, é preciso saber mediante a que atribuições de variáveis essa regra foi analisada.

Os elementos dos conjuntos X_E e Y_E têm o formato $\langle \text{tipo}, \text{vid} \rangle$, onde *tipo* é o nome do atributo ou parâmetro, e *vid* é o identificador da entidade que corresponde a esse atributo ou parâmetro.

A seguir, estão as definições formais destes conjuntos:

$$X_{\text{ADT}} = \{ \langle \text{tipoatr}, \text{vid} \rangle \mid \langle \text{aid}, \langle \text{tipoatr}, \underline{\text{atr}}, t_{GG}, o, d \rangle, o', \langle \text{vid}, \langle \text{tipovert}, \underline{\text{ADT}}, t_{GG} \rangle \rangle \in A_{GL} \}$$

$$X_E = \{ \langle \text{tipoatr}, \text{vid} \rangle \mid \langle \text{aid}, \langle \text{tipoatr}, \underline{\text{atr}}, t_{GG}, o, d \rangle, o', \langle \text{vid}, \langle \text{tipovert}, \underline{E}, t_{GG} \rangle \rangle \in A_{GL} \}$$

$$Y_{\text{ADT}} = \{ \langle \text{tipopar}, \text{vid} \rangle \mid \langle \text{aid}, \langle \text{tipopar}, \underline{\text{par}}, t_{GG}, o, d \rangle, o', \langle \text{vid}, \langle \text{tipovert}, \underline{\text{ADT}}, t_{GG} \rangle \rangle \in A_{GL} \}$$

$$Y_E = \{ \langle \text{tipopar}, \text{vid} \rangle \mid \langle \text{aid}, \langle \text{tipopar}, \underline{\text{par}}, t_{GG}, o, d \rangle, o', \langle \text{vid}, \langle \text{tipovert}, \underline{E}, t_{GG} \rangle \rangle \in A_{GL} \}$$

Notação: A formatação em negrito procura destacar o tipo do atributo ou parâmetro e o tipo do vértice de destino.

✓

Com a identificação dos atributos e parâmetros presentes no lado esquerdo da regra é possível realizar uma análise para verificar se o estado atual G e a lista de parâmetros *par_msg* satisfazem esses atributos. Para essa análise é necessário utilizar um algoritmo que tem como resultado final um valor booleano (indicando se o estado atual satisfaz os atributos) e uma tabela de atribuições feitas as variáveis encontradas (que conterá uma atribuição que satisfaz todos os atributos, caso ela exista).

- 1 Inicializa tab_var
- 2 $Habilitada := True$
- 3 Constrói conjuntos X_{ADT} , Y_{ADT} , X_E e Y_E
- 4 $C_{ADT} := X_{ADT} \cup Y_{ADT}$
- 5 $C_E := X_E \cup Y_E$
- 6 $C_{G_par} := G \cup par_msg$
- 7 **Algoritmo**_ C_{ADT}
- 8 **Algoritmo**_ C_E
- 9 **Se** (r not $Habilitada$)
- 10 **Então** Retorna ($False$, tab_var)
- 11 **Senão** Retorna ($True$, tab_var)

Algoritmo_ C_{ADT} : O objetivo do algoritmo_ C_{ADT} é analisar os atributos e parâmetros presentes no lado esquerdo de r que apresentam valores do tipo ADT. Nesse caso, para cada atributo de X_{ADT} e para cada parâmetro de Y_{ADT} é necessário encontrar um elemento pertencente a C_{G_par} que seja do mesmo tipo e que tenha o mesmo valor concreto presente no lado esquerdo de r . Caso o valor seja uma variável, é feita uma atribuição do valor de *tipo* pertencente a C_{G_par} para tal variável em uma tabela de atribuições. Se já existir uma atribuição a essa variável na tabela, esta deve ser considerada.

- 1 **Enquanto** ($(C_{ADT} \neq \emptyset)$ e ($Habilitada$)) **faça**
 / verificar todos os atributos e parâmetros enquanto ainda for possível encontrar atribuições de valores atuais para as variáveis da regra */*
- 2 Seleciona atributo ou parâmetro $\langle tipo, valor_regra \rangle \in C_{ADT}$
 / escolhe um atributo ou parâmetro para verificar, a ordem de escolha não muda o resultado */*
- 3 **Se** ($valor_regra, vtab \in tab_var$)
 / testa se o atributo ou parâmetro escolhido é uma variável para a qual já foi feita uma atribuição */*
- 4 **Então** $valor_regra := vtab$
 / valor_regra recebe o valor atribuído a esta variável em tab_var */*
- 5 **Senão** $valor_atual := v$ tal que $\langle tipo, v \rangle \in C_{G_par}$
 / valor_atual é o valor em C_{G_par} do atributo ou parâmetro que está sendo analisado */*
- 6 **Se** ($valor_atual = valor_regra$)
 / testa se o valor no estado atual é o mesmo da regra */*


```

7      Então  $C_{ADT} = C_{ADT} - \langle tipo, valor\_regra \rangle$ 
      /* simplesmente retira-se esse atributo ou parâmetro do conjunto
      a ser analisado e passa-se para o próximo */
8      Senão Se (valor_regra é uma variável)
      /*se o atributo ou parâmetro da regra for uma variável*/
9          Então  $var := valor\_regra$ 
10              $tab\_var := \langle var, valor\_atual \rangle$ 
11              $C_{ADT} = C_{ADT} - \langle tipo, valor\_regra \rangle$ 
      /* coloca-se o valor desse atributo ou parâmetro
      associado a essa variável na tabela tab_var, retira-se
      o atributo ou parâmetro do conjunto a ser analisado e
      passa-se para o próximo */
12      Senão  $Habilitada := False$ 
      /* se não for variável, então a regra pode ser
      aplicada, pois o valor desse atributo ou parâmetro no
      estado atual é diferente da constante na regra */

```

Algoritmo C_E : O objetivo do algoritmo C_E é analisar os atributos e parâmetros presentes no lado esquerdo de r que apresentam valores do tipo instância de entidade. Nesse caso, para cada atributo de X_E e para cada parâmetro de Y_E é necessário encontrar um elemento pertencente a C_{G_par} que seja do mesmo tipo e que tenha o mesmo valor concreto presente no lado esquerdo de r . Caso o valor seja uma variável, é feita uma atribuição do valor de *tipo* pertencente a C_{G_par} para tal variável em uma tabela de atribuições. Se já existir uma atribuição a essa variável na tabela, esta deve ser considerada.

```

1  Enquanto ( $(C_E \neq \emptyset)$  e (Habilitada)) faça
2      Seleciona atributo ou parâmetro  $\langle tipo, valor\_regra \rangle \in C_E$ 
3      Se (valor_regra, vtab)  $\in tab\_var$ 
4          Então  $valor\_regra := vtab$ 
5          Senão  $valor\_atual := v$  tal que  $\langle tipo, v \rangle \in C_{G\_par}$ 
6          Se ( $valor\_atual = valor\_regra$ )
7              Então  $C_E = C_E - \langle tipo, valor\_regra \rangle$ 
8              Senão Se (valor_regra é uma variável)
9                  Então  $var := valor\_regra$ 
10                      $tab\_var := \langle var, valor\_atual \rangle$ 
11                      $C_E = C_E - \langle tipo, valor\_regra \rangle$ 
12      Senão  $Habilitada := False$ 

```

Exemplo 5.2 (Regra habilitada) A partir da instância da entidade IE , da regra r e da lista de parâmetros da mensagem par_msg é possível caracterizar a regra do conjunto $RegrasDeUmaMensagem$ como habilitada ou desabilitada. Dentre as regras do conjunto $RegrasDeUmaMensagem$ foi possível caracterizar r_1 e r_2 como habilitadas. Pois, r_1 e r_2 apresentam do lado esquerdo apenas os atributos $wait$ e $next$. O atributo $wait$ foi instanciado como $False$ e o atributo $next$ foi instanciado com um valor do tipo $RSegm$, o valor $RSegm_2$, em ambas as regras (veja as figuras 5.1 e 5.2). Assim, considerando que o estado atual do sistema seja o estado representado na figura 5.3 é possível identificar que há correspondência entre os atributos das regras e os atributos identificados no estado atual. No estado atual, a instância da entidade identificada como $Train_1$ apresenta o atributo $wait$ com valor $False$ e o atributo $next$ com um valor do tipo $Rsegm$.

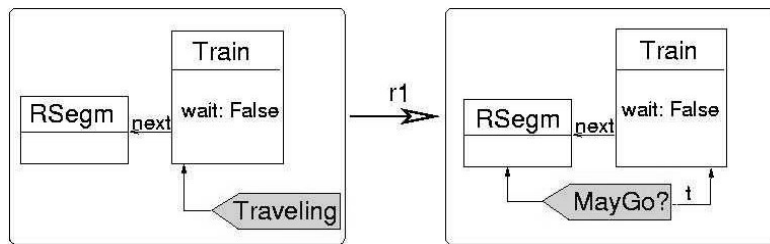


FIGURA 5.1 – Regra r_1 do $Train$.

Dados de Entrada: $IE = (Train_1, \{ \langle posNext, 0 \rangle, \langle posGate, 1 \rangle, \langle wait, False \rangle, \langle curr, Rsegm_1 \rangle, \langle next, Rsegm_2 \rangle, \langle gate, Gate_1 \rangle \}, \langle Traveling_1, \emptyset \rangle)$,
 $r = \langle r_1, \langle V_{GD1}, A_{GD1} \rangle, \langle V_{GC1}, A_{GC1} \rangle, \langle V_{GP1}, A_{GP1} \rangle, e_1 \rangle, par_msg = \emptyset$

Dados de Saída: $True, tab_var = \{ \langle RSegm, RSegm_2 \rangle \}$

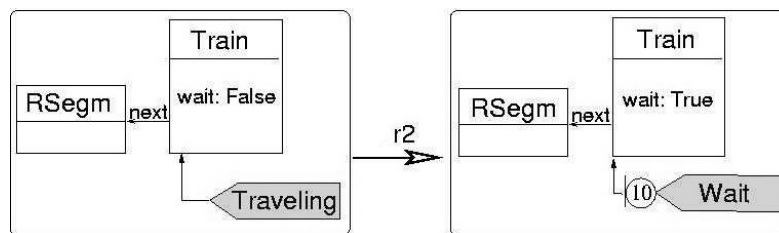


FIGURA 5.2 – Regra r_2 do $Train$.

Dados de Entrada: $IE = (Train_1, \{ \langle posNext, 0 \rangle, \langle posGate, 1 \rangle, \langle wait, False \rangle, \langle curr, Rsegm_1 \rangle, \langle next, Rsegm_2 \rangle, \langle gate, Gate_1 \rangle \}, \langle Traveling_1, \emptyset \rangle)$,
 $r = \langle r_2, \langle V_{GD2}, A_{GD2} \rangle, \langle V_{GC2}, A_{GC2} \rangle, \langle V_{GP2}, A_{GP2} \rangle, e_1 \rangle, par_msg = \emptyset$

Dados de Saída: $True, tab_var = \{ \langle RSegm, RSegm_2 \rangle \}$

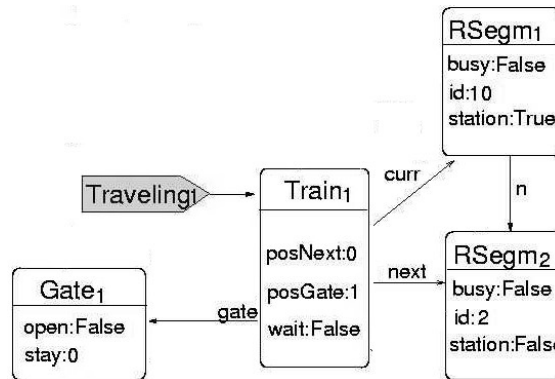
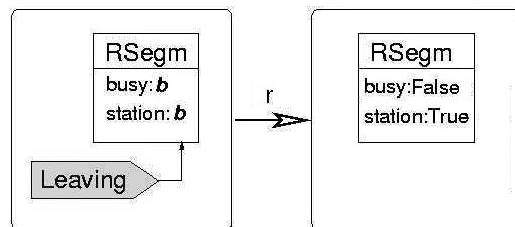


FIGURA 5.3 – Parte do estado atual do sistema (Ferrovia).

Exemplo 5.3 A figura 5.4 representa a regra r , onde os atributos *busy* e *station* foram definidos pela mesma variável b . Assim, considerando que o estado atual de um sistema esteja representado na figura 5.5 é possível identificar que não há correspondência entre os atributos da regra e os atributos identificados no estado atual. Nesse caso, não existe uma instânciação possível. Numa primeira análise b será identificado como variável, onde será feita uma atribuição de valor *False* para b , uma vez que no estado atual o atributo *busy* foi instanciado com valor *False*. Como a variável b define também o atributo *station*, isso leva a concluir que o estado atual do sistema deve apresentar o valor *False* para o atributo *station*. Observando a figura 5.5 é possível identificar que o atributo *station* foi instanciado com o valor *True*. Logo, a regra r não está habilitada para tratar uma mensagem do tipo *Leaving*.

FIGURA 5.4 – Regra r .

Dados de Entrada: $IE = (RSegm_1, \{<busy, False>, <id, 10>, <station, True>\}, <Leaving_1, \emptyset>), r = <r, <V_{GD}, A_{GD}>, <V_{GC}, A_{GC}>, <V_{GP}, A_{GP}>, e >, par_msg = \emptyset$

Dados de Saída: $False, tab_var = \{<b, False>\}$

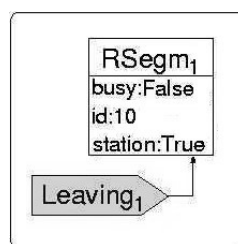


FIGURA 5.5 – Parte do estado atual.

Uma regra habilitada não equivale a uma regra aplicada. A aplicação de uma regra esta vinculada à questão do conflito. Para que uma regra seja aplicada é preciso garantir que essa regra não esteja em conflito com nenhuma outra regra identificada como regra aplicável no estado atual do sistema. Assim, com o intuito de obter um conjunto consistente de regras aplicáveis foi definida uma função que identifica se a regra está ou não está em conflito no estado atual do sistema.

5.2.2 Complexidade

A complexidade desse algoritmo é polinomial:

- Os conjuntos X_{ADT} , Y_{ADT} , X_E e Y_E são construídos analisando uma vez o lado esquerdo da regra, a lista de parâmetros e atributos que fazem parte da entrada do algoritmo.
- Para cada elemento desses conjuntos, deve ser verificado o seu valor (no estado atual, grafo G , ou na tabela que está sendo construída, tab_var , e depois no grafo G). Assim, é preciso fazer no máximo o número de comparações correspondente à soma dos elementos dos conjuntos de atributos e parâmetros (para cada um, analisa-se se ele está ou não satisfeito). Se o atributo for uma variável que não tem atribuição ainda, coloca-se na tabela tab_var e assume-se que ele está satisfeito (com o valor que foi colocado na tabela). Se já houver uma atribuição, só tem-se que verificar se o valor da tabela corresponde ao valor em G .
- Note que não existem duas atribuições possíveis, pois cada atributo ocorre exatamente uma vez no estado atual. O que o algoritmo faz é simplesmente verificar se os valores dos atributos da entidade e parâmetros da mensagem satisfazem as condições impostas pela regra.

5.3 Conflito

O conflito entre regras é caracterizado a partir do tipo de acesso realizado pelas regras em atributos de entidades. Regras que compartilham o acesso de escrita ou leitura e escrita em um mesmo atributo estão em conflito (veja seção 2.2).

Existem dois tipos de conflito: potencial e real. Para detecção de conflitos potenciais, pode-se fazer uma análise prévia sobre todas as regras definidas na gramática, antes que seja iniciada qualquer execução; enquanto para conflitos reais, o estado atual do sistema é necessário para essa análise, ou seja, ela depende de informações que são obtidas em tempo de execução. Assim, mediante o conceito de conflito potencial é possível determinar os pares de regras que podem estar ou que nunca estarão em conflito real. Pares de regras estão em conflito potencial não garantem a existência de conflito real para qualquer estado do sistema. Porém, os pares de regras que não estão em conflito potencial equivalem às regras que nunca estarão em conflito real, quando executadas em paralelo. Logo, a vantagem de identificar conflitos potenciais é que quando eles existem, eles são genéricos, valem para quaisquer aplicações de regras. Assim, a relação de regras que não apresentam conflito potencial tornaria a análise mais eficiente, visto que a ausência de conflito potencial permite a afirmação da ausência de conflito real entre as regras, independente do estado do

sistema. A determinação da ausência de conflito real necessita de uma análise não só dos acessos feitos aos atributos, mas também do estado do sistema, uma vez que muitas regras tidas como aplicáveis em um determinado estado do sistema podem ser executadas em paralelo, i.e., podem ser acionadas simultaneamente em instâncias diferentes sem qualquer conflito (veja o exemplo 5.4). Assim, para realizar a análise de conflito real é interessante realizar inicialmente uma análise de conflito potencial a partir da entidade. Através da análise de conflito potencial seria possível obter uma tabela com identificadores de regras que estão em conflito potencial, quando executadas em paralelo. Desta forma, a partir de informações como: modelo do sistema, conjunto de regras aplicáveis e tabela de identificadores de regras que estão em conflito potencial, é possível verificar se este conjunto de regras apresenta conflitos reais.

A seguir são apresentados detalhes sobre maneiras de identificar conflitos potenciais e reais, bem como exemplos.

5.3.1 Conflito Potencial

A análise de conflito potencial tem como entrada uma entidade E . Essa análise é realizada para todos os pares de regras pertencentes à entidade, retornando uma tabela composta pelos pares que estão em conflito potencial. A identificação de conflito potencial entre r e r' é determinada a partir de três verificações:

1. r e r' tratam o mesmo tipo de mensagem?
2. r e r' apresentam acesso de escrita em um mesmo tipo de atributo?
3. r apresenta acesso de leitura e r' acesso de escrita em um mesmo tipo de atributo ou vice-versa?

Se todas as verificações resultarem em uma resposta negativa, certamente o par r e r' não será encontrado na tabela resultante da análise, uma vez que r e r' não apresentam conflito potencial. Mas, se qualquer uma das verificações resultar em uma resposta afirmativa, então r e r' apresentam conflito potencial (veja o exemplo 5.4).

Função Conflito Potencial:

Dados de Entrada: Entidade $E = (id, GT, GE, N)$

Dados de Saída: Tabela tab_conf_pot de identificadores de regras que não apresentam conflito potencial quando executadas em paralelo, onde cada elemento da tabela é um par de identificadores de regras pertencentes ao conjunto N . **Observação:** se um par (r_1, r_2) está na tabela, o par (r_2, r_1) também deve estar.

Definição: Para cada par $r = (id, (V_{GD}, A_{GD}), C, (V_{GP}, A_{GP}), e) \in N$ e $r' = (id', (V_{GD}', A_{GD}'), C', (V_{GP}', A_{GP}'), e') \in N$, com $r \neq r'$: Sejam $AT_{GD} = \{at \mid \langle a, at, o, d \rangle \in A_{GD}\}$, $AT_{GD}' = \{at \mid \langle a, at, o, d \rangle \in A_{GD}'\}$, $AT_{GP} = \{at \mid \langle a, at, o, d \rangle \in A_{GP}\}$

A_{GP} }, $AT_{GP}' = \{at \mid \langle a, at, o, d \rangle \in A_{GP}'\}$, então o conjunto tab_conf_pot é definido da seguinte forma:

i) r e r' tratam o mesmo tipo de mensagem

$$\forall \langle v, vt \rangle \in V_{GD} \wedge \langle v', vt' \rangle \in V_{GD}' : vt = vt' \Rightarrow \langle id, id' \rangle \in tab_conf_pot$$

ii) r e r' apresentam acesso de escrita em um mesmo tipo de atributo

$$AT_{GD} \cap AT_{GD}' \neq \emptyset \Rightarrow \langle id, id' \rangle \in tab_conf_pot$$

iii) r apresenta acesso de leitura e r' acesso de escrita em um mesmo tipo de atributo ou vice-versa

$$AT_{GP} \cap AT_{GD}' \neq \emptyset \vee AT_{GD} \cap AT_{GP}' \neq \emptyset \Rightarrow \langle id, id' \rangle \in tab_conf_pot$$

Complexidade: Para montar a tabela tab_conf_pot , são consideradas todas as regras da entidade, duas a duas, ou seja, assumindo que o número de regras da entidade seja n (cardinalidade do conjunto N), tem-se $(n*(n-1))/2$ pares de regras para comparar (combinação de n elementos, dois a dois). Para cada um desses pares, devem ser construídos os conjuntos AT_{GD} , AT_{GD}' , AT_{GP} e AT_{GP}' , que contém apenas os tipos dos atributos deletados ou preservados por cada regra. Para isso, é preciso acessar cada arco da regra, ou seja, fazer, no máximo, $2m$ testes, onde m é o número de atributos da entidade (um atributo não pode ser preservado e deletado pela mesma regra). Depois, é preciso realizar os 3 testes definidos acima:

1. comparar a mensagem tratada por cada regra: somente uma comparação, pois cada regra só trata de uma mensagem (apenas um vértice é deletado em cada regra);
2. comparar os conjuntos de atributos atualizados pelas regras: cada atributo atualizado é um arco de A_{GD} . Assumindo que o número de atributos da entidade seja m , ter-se-ia no máximo, $(m*(m-1))/2$ comparações. Na realidade, esse máximo é maior que o número de comparações necessárias, pois quando for encontrado o primeiro elemento que está nos dois conjuntos, a busca pode parar, pois as regras já estão em conflito potencial;
3. Análogo ao item 2, considerando agora os conjuntos de itens deletados por uma regra e preservados pela outra, e vice-versa.

Como a complexidade tanto para construir os conjuntos quanto para fazer os testes necessários é polinomial em relação ao tamanho da entrada, esta função tem ordem polinomial de complexidade.

✓

Exemplo 5.4 (Conflito Potencial) A partir da entidade E , é possível realizar uma análise visando identificar conflitos potenciais entre as regras de E . Dentre as regras da entidade todas devem ser relacionadas e analisadas gerando uma tabela denominada tab_conf_pot . A tabela tab_conf_pot é composta por pares de identificadores de regras, sendo que essas regras podem tratar um mesmo tipo de mensagem, ou apresentarem acesso de escrita ou leitura e escrita em um mesmo tipo de atributo. Nesse exemplo, a

entidade considerada é denominada *Train*. Após relacionar todos os pares de regras pertencente ao conjunto N da entidade foi feita uma análise para cada par que resultou em uma tabela composta por diversos pares de regras. Pois, vários pares de regras atenderam as condições estabelecidas. Por exemplo, o par $\langle r_1, r_2 \rangle$ atendeu a primeira e a terceira condição, uma vez que tanto r_1 como r_2 tratam o tipo de mensagem *Traveling* e em r_1 o acesso aos atributos *wait* e *next* é de leitura, enquanto r_2 apresenta acesso de leitura para o atributo *next* e acesso de escrita para o atributo *wait*.

Dados de Entrada: Entidade $E = (Train, GT, GE, N)$

Dados de Saída: $tab_conf_pot = \{ \langle r_1, r_2 \rangle, \langle r_2, r_1 \rangle, \langle r_1, r_3 \rangle, \langle r_3, r_1 \rangle, \langle r_1, r_4 \rangle, \langle r_4, r_1 \rangle, \langle r_1, r_5 \rangle, \langle r_5, r_1 \rangle, \langle r_1, r_6 \rangle, \langle r_6, r_1 \rangle, \langle r_2, r_3 \rangle, \langle r_3, r_2 \rangle, \langle r_2, r_4 \rangle, \langle r_4, r_2 \rangle, \langle r_2, r_5 \rangle, \langle r_5, r_2 \rangle, \langle r_2, r_6 \rangle, \langle r_6, r_2 \rangle, \langle r_3, r_4 \rangle, \langle r_4, r_3 \rangle, \langle r_3, r_5 \rangle, \langle r_5, r_3 \rangle, \langle r_3, r_6 \rangle, \langle r_6, r_3 \rangle, \langle r_4, r_5 \rangle, \langle r_5, r_4 \rangle, \langle r_4, r_6 \rangle, \langle r_6, r_4 \rangle, \langle r_5, r_6 \rangle, \langle r_6, r_5 \rangle \}$

✓

5.3.2 Conflito Real

O objetivo desta função é identificar se uma determinada regra está em conflito com alguma outra regra selecionada como regra aplicável no estado atual do sistema. Assim, a função tem como entrada o modelo do sistema, um conjunto de regras aplicáveis, a tabela resultante da análise de conflito potencial, e a regra aplicável que se quer verificar se está em conflito com as outras regras aplicáveis já selecionadas (pertencentes ao conjunto passado como parâmetro para a função). O resultado é um valor booleano, que indica se há conflito ou não.

O algoritmo funciona da seguinte forma: Inicialmente, supõe-se que a regra não está em conflito com nenhuma outra regra. O conjunto *Regras* será inicializado com todas as regras já analisadas e definidas como regras aplicáveis. A análise é realizada, enquanto não for identificado conflito e houver regras aplicáveis a serem analisadas. Para cada regra selecionada do conjunto *Regras*, será feita uma busca na tabela resultante da análise de conflito potencial. Se o par entre a regra selecionada e a regra que esta sendo analisada não for encontrado na tabela, então o processo se iniciará para uma nova regra a ser selecionada, pois, entre a regra selecionada e a regra analisada nunca haverá conflito, visto que a ausência de conflito potencial determina a ausência de conflito real. Caso contrário (se existir conflito potencial) é preciso testar se as ocorrências das regras se sobrepõem no estado atual.

Função Conflito Real:

Dados de Entrada: Modelo $M_o = (GG_E, C_{IE})$, conjunto de regras aplicáveis *RegrasAplicáveis*, tabela de identificadores de regras que estão em conflito potencial tab_conf_pot , regra aplicável $\langle r = \langle id_r, D, C, P, e \rangle, m \rangle$

Dados de Saída: Booleano

Algoritmo:

```

1  Conflito := False
2  Regras := RegrasAplicáveis
3  Enquanto ((Regras ≠ ∅) e (not Conflito)) faça
4    Seleciona regra  $\langle r' = \langle id_r', D', C', P', e' \rangle, m' \rangle \in \textit{Regras}$ 
5    Se ( $id_r, id_{r'}$ ) ∉ tab_conf_pot
6      Então Regras := Regras -  $r'$ 
7      Senão Se  $m$  e  $m'$  não se sobrepõem
8        Então Regras := Regras -  $r'$ 
9        Senão Conflito := True
10  Retorna Conflito

```

Complexidade: Nesta função, deve-se comparar a regra $\langle r, m \rangle$ com cada uma do conjunto *RegrasAplicáveis*, ou seja, tem-se $n = |\textit{RegrasAplicáveis}|$ comparações, no máximo. Para cada par $\langle \langle r, m \rangle, \langle r', m' \rangle \rangle$, verifica se essas regras não estão em conflito potencial (procurar um elemento em um conjunto), e depois verifica se as ocorrências não se sobrepõem (caso haja conflito potencial). Para verificar isso, é preciso fazer uma análise similar à da função que identifica conflitos potenciais, mas considerando agora se os vértices e arcos deletados e preservados pelas regras são os mesmos (ao invés de considerar os tipos de vértices e arcos, como na função conflito potencial). Portanto, a complexidade desta função é também polinomial.

Observação: Dada uma regra $r: L \rightarrow R$, onde $L = D \cup P$ e $R = C \cup P$, uma ocorrência de r em M_o é um morfismo total de grafos representado por $m: L \rightarrow M_o$.

✓

Exemplo 5.5 (Conflito) Considere o modelo M_o definido no exemplo 4.8, o conjunto de regras aplicáveis $\textit{RegrasAplicáveis} = \{ \langle r_1, m_1 \rangle \}$, onde r_1 está ilustrada na figura 5.1 e a ocorrência m_1 é o mapeamento do lado esquerdo desta regra para o trem \textit{Train}_1 (com correspondentes atributos e mensagem), a tabela de identificadores de regras que estão em conflito potencial *tab_conf_pot* (exemplo 5.4) e a regra aplicável denominada $\langle r_2, m_2 \rangle$, onde r_2 está ilustrada na figura 5.2 e a ocorrência m_2 mapeia a entidade trem do lado esquerdo da regra para \textit{Train}_2 . Agora, é preciso verificar se existe conflito real entre $\langle r_2, m_2 \rangle$ e o conjunto *RegrasAplicáveis*. O fato da tabela *tab_conf_pot* apresentar o par $\langle r_1, r_2 \rangle$, torna a identificação de conflito dependente do estado do sistema. Assim, para descartar a hipótese de conflito é necessário verificar se no estado do sistema, ou melhor, no modelo M_o , as ocorrências m_1 e m_2 se sobrepõem. Observando o modelo, representado na figura 3.11, é fácil verificar que, apesar das duas regras deletarem o mesmo tipo de mensagem, as mensagens concretas utilizadas por elas em m_1 e m_2 são diferentes. Logo, a regra aplicável $\langle r_2, m_2 \rangle$ não está em conflito podendo ser executada em paralelo com as regras do conjunto *RegrasAplicáveis* (que, neste exemplo, é formado apenas da regra aplicável $\langle r_1, m_1 \rangle$).

Dados de Entrada: $M_o = (GG_E, C_{IE})$, $RegrasAplicáveis = \{ \langle r_1, m_1 \rangle \}$,
 $tab_conf_pot = \{ \langle r_1, r_2 \rangle, \langle r_2, r_1 \rangle, \langle r_1, r_3 \rangle, \langle r_3, r_1 \rangle, \langle r_1, r_4 \rangle, \langle r_4, r_1 \rangle, \langle r_1, r_5 \rangle, \langle r_5, r_1 \rangle, \langle r_1, r_6 \rangle, \langle r_6, r_1 \rangle, \langle r_2, r_3 \rangle, \langle r_3, r_2 \rangle, \langle r_2, r_4 \rangle, \langle r_4, r_2 \rangle, \langle r_2, r_5 \rangle, \langle r_5, r_2 \rangle, \langle r_2, r_6 \rangle, \langle r_6, r_2 \rangle, \langle r_3, r_4 \rangle, \langle r_4, r_3 \rangle, \langle r_3, r_5 \rangle, \langle r_5, r_3 \rangle, \langle r_3, r_6 \rangle, \langle r_6, r_3 \rangle, \langle r_4, r_5 \rangle, \langle r_5, r_4 \rangle, \langle r_4, r_6 \rangle, \langle r_6, r_4 \rangle, \langle r_5, r_6 \rangle, \langle r_6, r_5 \rangle \}$, $regra\ aplicável = \langle r_2, m_2 \rangle$

Dados de Saída: *False*



6 Protótipo

A representação concreta das estruturas do ambiente PLATUS não é algo trivial, visto que toda a representação precisa ser fiel ao formalismo de GGs e eficiente para agilizar a execução do simulador. Buscando atender tais considerações, foi necessário realizar definições e análises, que possibilitaram por fim o desenvolvimento do protótipo.

O protótipo visa uma solução aplicável ao ambiente de simulação, onde não só a representação das estruturas será possível, como também a validação do modelo proposto, através de estudo de caso. Considerando que se trata de um protótipo, é possível ressaltar algumas limitações:

- **Representação de um único tipo de GGs:** não é feita qualquer distinção quanto ao tipo de GGs na representação do protótipo. A representação destina-se a gramática de grafos de comportamento (GGc), até mesmo porque, os vários tipos de gramáticas que permitem a descrição de todos os aspectos de uma entidade, são refinamentos de GGc.
- **Validação parcial do modelo:** caracterizada como parcial, uma vez que o protótipo considera no estudo de caso, o grafo inicial de uma entidade como estado inicial do sistema, e não um modelo do sistema como um todo (composto por vários grafos iniciais de diversas entidades).

A próxima seção apresenta detalhes sobre a implementação do protótipo que foi idealizada neste trabalho, e desenvolvida pela aluna Tatiana DelTrejo Bezerra [BEZ2001], como trabalho de conclusão do curso de Bacharelado em Ciência da Computação.

6.1 Implementação

O uso de um banco de dados como forma de representação concreta das estruturas definidas no protótipo, foi considerada, desde o início, como a solução mais indicada. Porém, o número de banco de dados existentes é bem diversificado, o que poderia gerar uma dúvida, não quanto à abordagem, mas quanto à distribuição, uma vez que se almeja um sistema de abordagem relacional com distribuição livre, ou melhor, um sistema relacional de código aberto e acessível a todos os pesquisadores do projeto PLATUS. Assim, mediante as características de armazenamento (como mostra a tabela 6.1), disponibilidade (código aberto) e multi-plataforma (Linux, Unix, Solaris, Windows e outras) do InterBase Server 6.0, foi possível justificar a escolha do banco de dados do protótipo, dentre os vários sistemas existentes.

O InterBase Server 6.0 é caracterizado como um sistema de banco de dados robusto, capaz de representar sistemas complexos. Entretanto, é preciso ressaltar que através de um banco de dados, não é possível garantir todas as restrições necessárias para assegurar uma representação fiel ao formalismo de GGs. Portanto, foi preciso utilizar uma ferramenta de programação, denominada Delphi. A Delphi juntamente com o InterBase possibilitou o armazenamento e a recuperação da representação das estruturas do PLATUS (veja o exemplo 6.1). Assim, a implementação do protótipo foi

desenvolvida, sendo convenientemente dividida em duas partes denominadas: cadastro e estudo de caso (veja figura 6.2).

TABELA 6.1 – Especificações Técnicas do InterBase. [IBS2001]

Tamanho máximo do banco	32 Terabytes
Tamanho máximo de um arquivo GDB	4 GB na maioria das plataformas;
Número máximo de tabelas	2^{16} Tabelas
Tamanho máximo de uma tabela	32 Terabytes
Número máximo de linhas por tabelas	2^{32} linhas
Tamanho máximo da linha	64 KB
Número máximo de índices por tabela	2^{16} Índices
Número máximo de índices por banco	2^{32} Índices

Exemplo 6.1 A figura 6.1 representa apenas parte do grafo-tipo baseado em objeto. Analisando a figura é possível verificar que tanto a origem, como o destino dos arcos, apresentam restrições quanto ao tipo de objeto. Arcos só podem ter como vértice de origem, objetos do tipo entidade, conforme mostra a figura 6.1 (através de atr1 e atr2). Enquanto o vértice de destino restringe-se a dois tipos de objeto, podendo ser especificado como um vértice do tipo entidade (representado por atr1) ou do tipo ADT (representado por atr2). Portanto, de acordo com as definições, tais restrições jamais poderiam ser representadas apenas por um relacionamento entre tabelas do banco de dados.

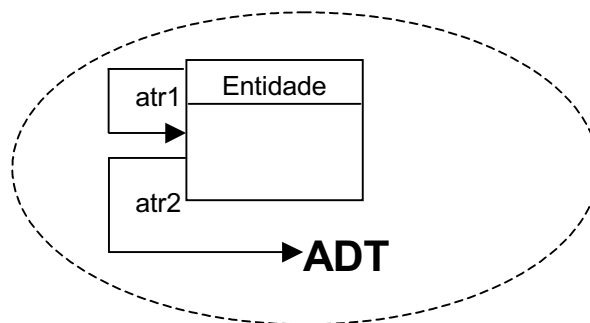


FIGURA 6.1 – Parte do grafo-tipo baseado em objeto.



FIGURA 6.2 – Protótipo.

6.1.1 Cadastro

O módulo cadastro consiste na definição do sistema, ou seja, nessa parte o aplicativo trata da identificação do modelo do sistema, dos grafos, dos objetos (vértices e arcos) que definem os grafos, do grafo-tipo, das instâncias e das regras, ou melhor, da representação da gramática de grafos que especifica um determinado sistema (veja figura 6.3).

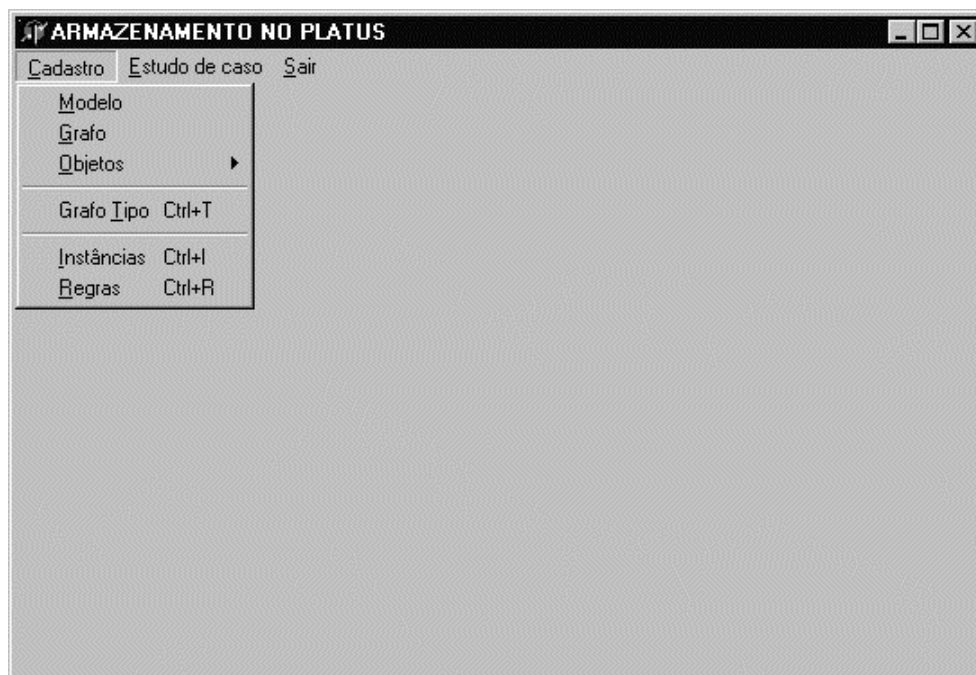


FIGURA 6.3 – Menu de cadastro do protótipo.

- **Modelo:** identifica o modelo do sistema a ser especificado (veja figura 6.4).

The image shows a dialog box titled "Cadastro de Modelo". It has a standard Windows-style title bar with a close button. The main area contains a text input field labeled "Nome do modelo:". Below the input field is a horizontal row of five buttons: "Alterar" (with a document icon), "Apagar" (with a trash can icon), "Confirmar" (with a checkmark icon), "Cancelar" (with a circle and slash icon), and "Sair" (with a door icon).

FIGURA 6.4 – Cadastro de modelo.

- **Grafo:** identifica os grafos que compõem o modelo, sendo que cada grafo referencia um modelo cadastrado e pode ser classificado como: grafo-tipo ou grafo inicial ou grafo tipado (veja figura 6.5).

The image shows a dialog box titled "Cadastro de Grafo". It has a standard Windows-style title bar with a close button. The main area contains a text input field for "Grafo:" and a dropdown menu for "Modelo:". Below these is a section labeled "Classificação:" containing three radio button options: "Grafo Tipo", "Grafo Inicial", and "Grafo Tipado". At the bottom is a horizontal row of five buttons: "Alterar", "Apagar", "Confirmar", "Cancelar", and "Sair", identical to the previous dialog box.

FIGURA 6.5 – Cadastro de grafos.

- **Objetos:** identifica vértices e arcos que compõem os grafos do modelo.
 - **Vértices:** identifica os tipos de vértices que serão utilizados como origem/destino dos arcos. Cada vértice apresenta apenas um tipo que pode ser classificado como ADT ou entidade ou mensagem ou tempo (veja figura 6.6).
 - **Arcos:** identifica os tipos de arcos que serão utilizados para definir o grafo-tipo e conseqüentemente outros grafos tipados. Cada arco apresenta apenas um tipo que pode ser classificado como atributo ou parâmetro ou destino ou *time stamp* (veja figura 6.7). Arcos apresentam uma identificação de vértice de origem e outra para vértice de destino, sendo que a consistência tanto da origem quanto do destino é garantida na implementação. Pois, dependendo da classificação do arco serão feitas verificações

quanto ao tipo de vértice identificado como origem e como destino.

FIGURA 6.6 – Cadastro de vértices.

FIGURA 6.7 – Cadastro de arcos.

- **Grafo-tipo:** seleciona vértices e arcos cadastrados para então definir o grafo-tipo do modelo. Assim, para cada grafo-tipo definido são feitos vários acessos de leitura e gravação em diversas tabelas, uma vez que todos os vértices e arcos que compõem o grafo-tipo já foram cadastrados anteriormente. Desta forma, ficou estabelecido que as informações referentes ao grafo-tipo só seriam gravadas definitivamente no banco de dados após toda a definição (veja a figura 6.8).
- **Instâncias:** identifica cada instância que compõe os grafos classificados como inicial ou tipados. Assim, cada instância de um determinado grafo-tipo requer uma validação, uma vez que nenhum vértice ou arco pode ser definido se não estiver de acordo com o que foi especificado no grafo-tipo. Desta forma, apenas os vértices e arcos que compõem o grafo-tipo eram disponibilizados. Para instanciar um vértice ou arco, basta selecionar o objeto pertencente ao grafo-tipo e fornecer uma nova descrição. A solução encontrada possibilitou a checagem dos objetos com o grafo-tipo durante a construção da instância, ou seja, a instância é criada e verificada a cada inserção de um elemento. (veja figura 6.9).

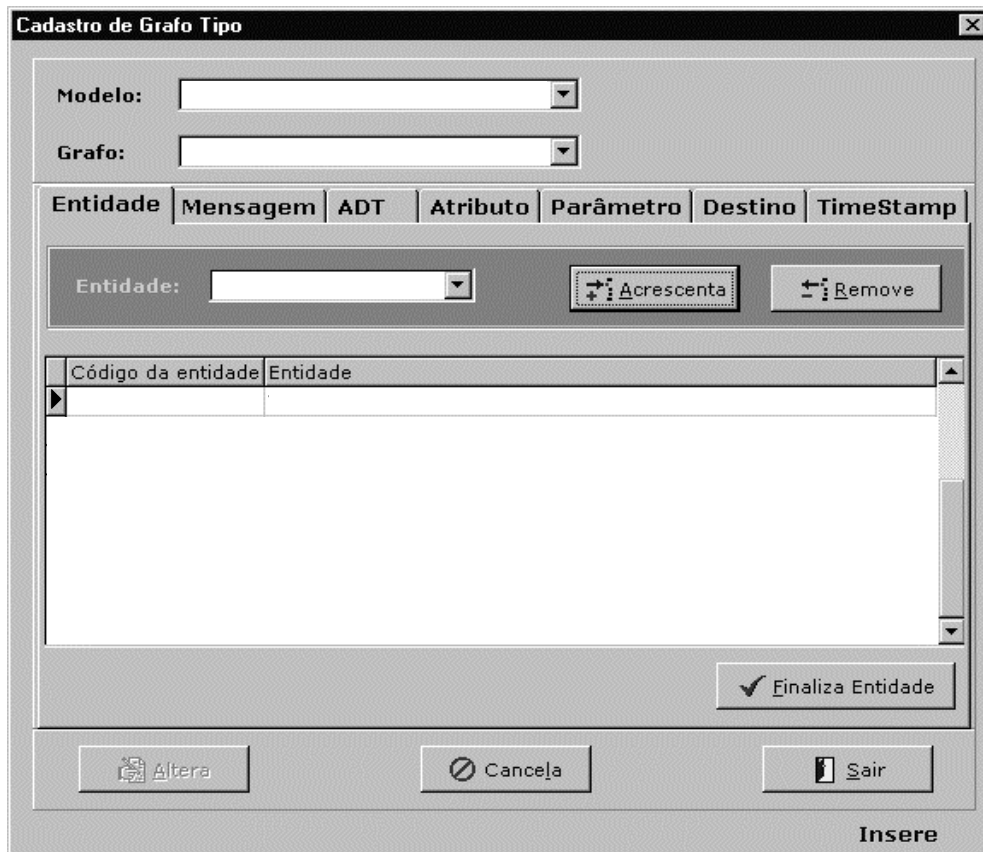


FIGURA 6.8 – Cadastro de grafo-tipo.

- **Regras:** a definição da regra não é formada por partes, ou melhor, lado esquerdo, lado direito e mapeamento. A solução adotada consiste na construção do lado esquerdo e o direito ao mesmo tempo, onde o mapeamento é estabelecido porque o lado esquerdo e o direito seriam armazenados em um único registro. Assim, este módulo define os itens que são preservados, deletados e criados pela aplicação da regra. O cadastro de regras disponibiliza apenas os vértices e arcos que compõem o grafo tipado, i.e., instância de um grafo-tipo. Para cadastrar uma regra, basta selecionar um objeto que faça parte do grafo tipado e informar o *status* do vértice ou arco (veja figura 6.10).
- **Preservado:** identifica os objetos que aparecem em ambos os lados da regra.
 - **Deletado:** identifica os objetos que aparecem apenas no lado esquerdo da regra.
 - **Criado:** identifica os objetos que aparecem apenas no lado direito da regra.
 - **Deletado e criado⁵:** identifica os objetos que aparecem em ambos os lados da regra sem preservar o valor, i.e., objetos modificados.

⁵ Facilita a identificação de conflito entre regras.

Cadastro de Instâncias

Modelo: Grafo Tipo:

Grafo da instância:

Entidade | Mensagem | ADT | Atributo | Parâmetro | Destino | TimeStamp

Entidade:

Descrição:

Descrição da instância entidade

Inserir

FIGURA 6.9 – Cadastro de instâncias.

Cadastro de Regras

Modelo:

Grafo: N° da Regra:

Entidade | Mensagem | ADT | Atributo | Parâmetro | Destino | TimeStamp | Condição

Entidade:

Status: criado preservado
 deletado criado e deletado

Código da entidade	Entidade	Criado	Deletado	Preservado

Inserir

FIGURA 6.10 – Cadastro de regras.

6.1.2 Estudo de caso

O módulo do estudo de caso consiste de uma análise em relação à recuperação das informações necessárias ao simulador. A análise é realizada mediante um conjunto de regras e o grafo inicial de uma entidade. Desta forma, a validação é feita apenas para uma entidade do modelo proposto, uma vez que o estado inicial do sistema considerado não abrange todas as entidades especificadas no modelo.

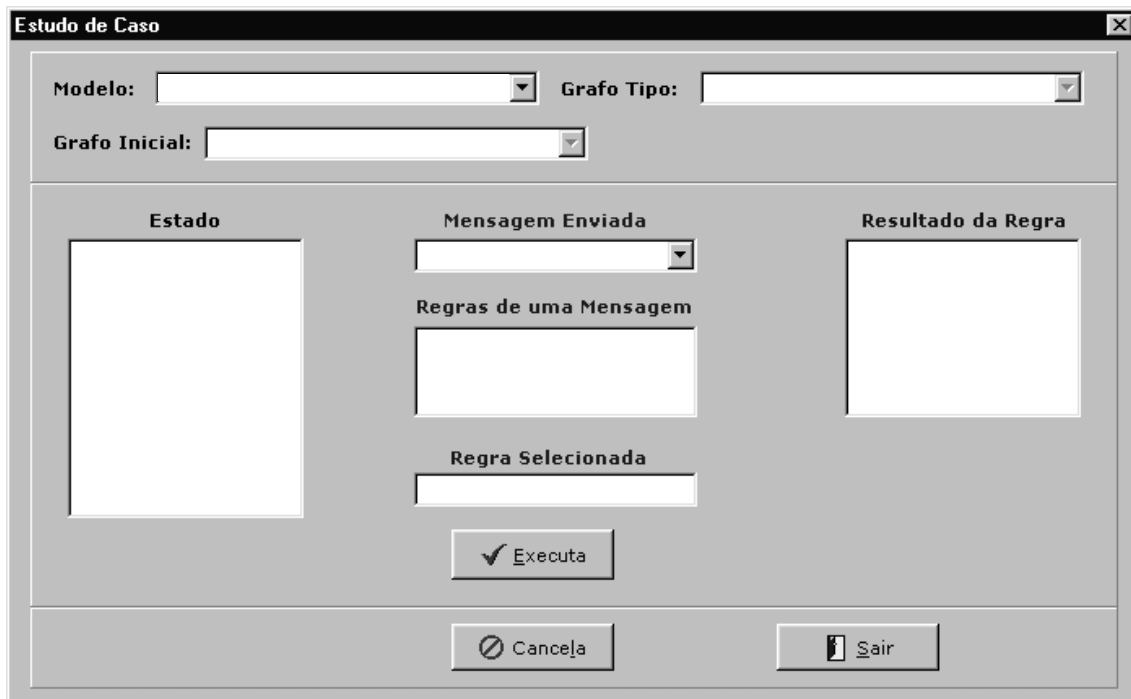


FIGURA 6.11 – Estudo de caso.

Para maiores detalhes sobre a implementação do protótipo veja [BEZ2001].

7 Conclusão

O uso da especificação formal está crescendo cada vez mais, de acordo com a proporção do tamanho dos sistemas e está juntamente com a simulação fazendo com que o desenvolvimento dos sistemas de controle avance cada vez mais em busca de requisitos como estimativa de tempo e desempenho, ainda em tempo de projeto.

Neste contexto, o PLATUS permite a construção de modelos de simulação baseados em técnicas de especificação formal que usam gramáticas de grafos como ferramenta descritiva. Entretanto, a representação das estruturas do PLATUS não é algo trivial, uma vez que tudo é representado por grafos, que representam desde aspectos estáticos, dinâmicos, até aspectos de interface.

Esta dissertação foi motivada pela necessidade de uma padronização na representação das estruturas do PLATUS, visando facilitar não apenas o armazenamento, mas também a recuperação e a análise das estruturas de forma eficiente, uma vez que se trata de um ambiente de simulação.

A partir dos estudos que foram realizados mediante outros ambientes, foi possível elaborar definições para grafo-tipo e grafos tipados que são, em grande parte, compostas por muito mais relações do que funções (relações especiais que garantem naturalmente algumas restrições), uma vez que matematicamente, define-se uma relação como um subconjunto de um produto cartesiano de uma lista de domínios e essa definição, segundo [SKS97], corresponde quase que exatamente à definição de uma tabela. Assim, devido ao fato de tabelas em essência serem relações, a conclusão das definições acabaram facilitando a elaboração de um modelo de dados que possibilitou a representação das estruturas do PLATUS, facilitando e agilizando o acesso e a troca das informações no ambiente.

Para que a recuperação das informações fosse realizada de forma eficiente, métodos foram propostos e implementados, com o objetivo de analisar as estruturas representadas e retornar ao simulador, informações que são necessárias à simulação de um modelo. Segundo [TOS2001], às vezes, o algoritmo mais imediato está longe de ser razoável em termos de eficiência. Desta forma, foram realizados alguns cálculos da complexidade como meio de garantir que a recuperação de informações seria eficiente. Assim, mediante as definições e análises foi possível elaborar um protótipo de uma implementação em um banco de dados, com o objetivo de validar o modelo proposto, através de um estudo de caso.

Espera-se que, com o estudo e resultados alcançados, este trabalho venha a somar-se às atividades que estão sendo desenvolvidas no projeto PLATUS, contribuindo para a representação das estruturas das diversas ferramentas do ambiente PLATUS. A concepção deste trabalho foi muito importante para o aprendizado teórico-prático de representação e análise de gramáticas de grafos.

Acredita-se que em trabalhos futuros seja possível abordar o tratamento da expressão booleana da regra, assim como possibilitar o intercâmbio de informações através da *web*, mediante o uso da XML ou GXL.

Referências

- [APP97] APPLIGRAPH. **Applications of Graph Transformation**. 1997. Disponível em: <<http://www-i3.informatik.rwth-aachen.de/gragra/tools.html>>. Acesso em: 30 mar. 2001.
- [BAR99] BARDOHL, R.; ERMEL, C.; RIBEIRO, L. A Modular Approach to Animation of Simulation Models. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, SBES, 13., Florianópolis: UFSC, 1999; **Anais...** Florianópolis: UFSC, 1999.
- [BEZ2001] BEZERRA, D. T. **Representação de Gramática de Grafos em Banco de Dados**. 2001. Trabalho de Conclusão (Bacharelado em Ciência da Computação) – Universidade do Oeste Paulista, Faculdade de Informática de Presidente Prudente, Presidente Prudente.
- [COP2000] COPSTEIN, B.; RIBEIRO, L.; MORA, M. C. An Environment for Formal Modeling and Simulation of Control Systems. In: ANNUAL SIMULATION SYMPOSIUM, 2000. **Proceedings...** Washington:IEEE, 2000. p.74-82.
- [COR96] CORRADINI, A.; MONTANARI, U.; ROSSI, F. Graph processes. **Fundamentae Informatica**, [S.l.], v.26, n.3-4, p.241-265, 1996.
- [COR97] CORRADINI, A.; MONTANARI, U.; ROSSI, F. et al. Algebraic Approaches to Graph Transformation I: basic concepts and double pushout approach. In: **Handbook of Graph Grammars and Computing by Graph Transformation**. Singapore:World Scientific, 1997. v.1.
- [DEH2000] DÉHARBE, D.; MOREIRA, A. M.; RIBEIRO, L. et al. Introdução a Métodos Formais: especificação, semântica e verificação de sistemas concorrentes. **Revista de Informática Teórica e Aplicada**, Porto Alegre, v.7, n.1, p.7-48, set. 2000.
- [DOT2000] DOTTI, F. L.; RIBEIRO, L. Specification of Mobile Code Using Graph Grammars. In: IFIP TCG WGG.1 INTERNATIONAL CONFERENCE ON FORMAL METHODS FOR OBJECT-BASED DISTRIBUTED SYSTEMS, 4., 2000, Stanford. **Formal Methods for Open Object-Based Distributed Systems IV**. Boston:Kluwer Academic, 2000. p.45-64.
- [ERM2001a] ERMEL, C. **The AGG environment**. Disponível em: <<ftp://ftp-i3.informatik.rwth-aachen.de/pub/reports/SWZ95c.ps.gz>>. Acesso em: 30 mar. 2001.
- [ERM2001b] ERMEL, C. **XML and AGG**. Disponível em: <<http://tfs.cs.tu-berlin.de/agg/xml.html>>. Acesso em: 30 mar. 2001.
- [IBS2001] IBSUPER. **Interbase Superpage – Estatísticas do banco**. 2001. Disponível em: <<http://ibsuper.net/dicas3.htm>>. Acesso em: 03 ago. 2001.

- [KSW95] KIESEL, N.; SCHUERR, A; WESTFECHTEL, B. GRAS, a Graph-Oriented (software) Engineering Database System. **Information Systems**, Germany, v.20, n.1, Feb.1995.
- [NAG91] NAGL, M. The Use of Graph Grammars in Applications. In: **Graph Grammars and Their Applications to Computer Science**. [S:l]:Springer-Verlag, 1991. p.41-60. (Lecture Notes in Computer Science, 532).
- [RIB96] RIBEIRO, L. **Parallel Composition and Unfolding Semantics of Graph Grammars**. 1996. Ph. D. thesis, Technical University of Berlin, Germany.
- [RIB97] RIBEIRO, L.; KORFF, M. **Métodos Formais para Especificação: Gramática de Grafos**. Pelotas:UFPel, 1997.
- [RIB99a] RIBEIRO, L. Parallel Composition of Graph Grammars. **Applied Categorical Structures**, Dordrecht, v.7, n.4, p.405-430, Dec.1999.
- [RIB99b] RIBEIRO, L.; COPSTEIN, B. Compositional Construction of Simulation Models Using Graph Grammars. In: INTERNATIONAL WORKSHOP ON APPLICATIONS OF GRAPH TRANSFORMATION WITH INDUSTRIAL RELEVANCE, 1999, Kerkrade. **Applications of Graph Transformation with Industrial Relevance: proceedings**. Berlin:Springer – Verlag, 2000. p.87-94.
- [RIB2001] RIBEIRO, L.; MORA, M. C.; COPSTEIN, B. Event Driven Simulation of Object-Based Graph Grammar Models. In: ANNUAL SIMULATION SYMPOSIUM, 2001. **Proceedings...**Seattle: [s.n.], 2001.
- [ROZ99] ROZENBERG, G.; ENGELS, G.; EHRIG, H. et al. Term Rewriting and Functional Languages. In: **Handbook of Graph Grammars and Computing by Graph Transformation: Applications, Languages and Tools**. Singapore: World Scientific, 1999. v. 2.
- [SCH95] SCHÜRR, A.; WINTER, A. J.; ZÜNDORF, A. Graph Grammars Engineering with Progres. In: INTERNATIONAL WORKSHOP ON GRAPH GRAMMARS AND THEIR APPLICATION TO COMPUTER SCIENCE, 1995. **Graph Grammars and Their Applications to Computer Science**. Berlin:Springer - Verlag, 1995. p.219-234. (Lecture Notes in Computer Science, 989). Disponível em: <ftp://ftp-i3.informatik.rwth-aachen.de/pub/reports/SWZ95c.ps.gz>. Acesso em: 30 mar. 2001.
- [SKS97] SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Sistema de Bancos de Dados**. São Paulo:Makron Books, 1997. p.61-108.
- [TAN97] TANABE, M. Timed Petri Nets and Temporal Linear Logic. In: INTERNATIONAL CONFERENCE ON APPLICATION AND THEORY OF PETRI NETS, 1997. Toulouse:1997. p.156-174. (Lecture Notes in Computer Science, 1248).
- [TOS2001] TOSCANI, L. V.; VELOSO, P. A. S. **Complexidade de Algoritmos**. Porto Alegre:Sagra Luzzatto, 2001. 202p.