Especificação e Verificação Formal de Sistemas Distribuídos*

Fernando L. Dotti[1], *Luciana Foss*[2], *Leila Ribeiro*[2] *e Osmar M. dos Santos*[1]

[1] Faculdade de Informática - Pontifícia Universidade Católica do Rio Grande do Sul
Porto Alegre, RS - Brasil
Email: {fldotti, osantos}@inf.pucrs.br

[2] Instituto de Informática - Universidade Federal do Rio Grande do Sul
Porto Alegre, RS - Brasil
Email: {lfoss, leila}@inf.ufrgs.br

Resumo

Neste artigo utilizamos uma linguagem visual de especificação formal, chamada Gramáticas de Grafos Baseadas em Objetos (GGBO), para especificar sistemas distribuídos. A GGBO é comparada com a linguagem PROMELA (*a PROcess MEta LAnguage*), usada como entrada pelo verificador de modelos SPIN. Com base nesta comparação, define-se um mapeamento de GGBO para PROMELA. É visto como especificar e verificar propriedades sobre os modelos mapeados para PROMELA. Um algoritmo de eleição em anel é utilizado, no decorrer do artigo, para ilustrar a GGBO, a tradução e a verificação de propriedades.

Palavras-chave: Gramática de grafos, verificação de modelos, sistemas distribuídos.

Abstract

In this paper we use a visual formal specification language, called Object-Based Graph Grammars (OBGG), suitable for the specification of distributed systems. The OBGG is compared with the language PROMELA (a PROcess MEta LAnguage), used as input by the SPIN model checker. Based on this comparison, a translation from OBGG to PROMELA is defined. It is shown how to specify and verify properties over the translated PROMELA models. An algorithm of leader election in a ring is used, throught the paper, to illustrate the OBGG, the translation and the property verification.

Keywords: Graph grammars, model checking, distributed systems.

1 Introdução

Ambientes para computação distribuída estão se tornando cada vez mais comuns e atingindo um número maior de pessoas e organizações. Devido à evolução rápida e contínua das capacidades de comunicação e processamento, assistimos, nos últimos anos, o surgimento de

*Este trabalho ´e parcialmente financiado por HP Brasil - PUCRS contrato CASCO (24 TA.), projetos de pesquisa ForMOS (FAPERGS e CNPq), PLATUS (CNPq) e IQ-MObile (CNPq/CNR).

diferentes plataformas para computação distribuída. Temos o uso de redes de longa distância como plataforma para processamento distribuído. Tais ambientes, como, por exemplo, a Internet, permitem formas variadas de cooperação e integração entre organizações e sistemas, sendo caracterizados por: alta distribuição geográfica, ultrapassar fronteiras administrativas; dinamismo (oferta e retirada de serviços e nós computacionais); inexistência de controle global; falhas parciais; falta de segurança; heterogeneidade tanto de nós computacionais (capacidade de processamento), como enlaces de comunicação (atraso, vazão e perda de pacotes); falta de qualidade na comunicação. Ambientes reunindo tais características são também chamados "ambientes abertos" [1].

A construção de um sistema distribuído não é uma tarefa trivial. Independente do ambiente computacional alvo para o sistema em questão, existem alguns requisitos importantes que se colocam durante a fase de construção do sistema: garantia de propriedades desejadas e avaliação do desempenho. Neste trabalho o primeiro requisito será abordado, ou seja, verificação.

Para se realizar a verificação formal de propriedades de um sistema, é necessário que o comportamento do mesmo seja descrito por um modelo matemático. Por isso, é recomendável que já na fase de especificação seja usada uma linguagem com sintaxe e semântica formais, permitindo que erros sejam encontrados antes da implementação do sistema. Existem basicamente duas abordagens de verificação formal: verificação de modelos e prova de teoremas [2]. Na primeira, constrói-se um modelo (geralmente um sistema de transição) contendo o comportamento do sistema, então este modelo é analisado. Na segunda, o comportamento do sistema deve ser descrito através de axiomas e regras, usa-se lógica para deduzir propriedades das computações do sistema. Com algumas restrições, essas duas abordagens podem ser automatizadas. Para utilizar uma ferramenta de verificação de modelos, o modelo que descreve o comportamento do sistema deve ser finito (o número de estados alcançáveis deve ser finito, o que não implica que não possa haver computações infinitas). Essa restrição não ocorre no uso de provadores de teoremas, mas estes requerem muito conhecimento de técnicas de prova do usuário, que normalmente precisa auxiliar a ferramenta a completar as provas (ou seja, a verificação não é totalmente automática). Como desenvolver um verificador de modelos é uma tarefa bastante complexa, várias linguagens de especificação tem sido traduzidas para as linguagens de entrada de verificadores conhecidos.

Como será discutido em maior detalhe na seção 2, encontramos basicamente duas abordagens bem difundidas para a especificação de sistemas distribuídos: baseada em objetos e baseada em processos, cada uma com importantes vantagens complementares. Tentando aliar os pontos positivos de ambas abordagens, como formalismo para especificação utilizamos a Gramáticas de Grafos Baseadas em Objetos (GGBO) [3]. GGBO é um formalismo gráfico e declarativo, oferecendo conceitos básicos de sistemas baseados em objetos, abstrações de não-determinismo e concorrência inerente. Por ser baseada em objeto, carrega vantagens como modularidade, facilitando o reuso e a construção de sistemas grandes. Por ser relativamente restrita em suas construções e ter uma semântica formal, permite seu mapeamento para ambientes de análise, tais como simuladores [4] [5] ou verificadores [6]. GGBO também se presta para o tratamento analítico necessário para avaliação de desempenho [7].

Neste artigo será realizada uma comparação entre as linguagens GGBO e PROMELA (*a PROcess MEta LAnguage*), a linguagem de entrada do verificador de modelos SPIN, com o objetivo de utilizar o SPIN para verificar sistemas modelados em GGBO. Para isso, será sugerida uma tradução de GGBO para PROMELA. Tanto a linguagem de especificação GGBO

como o mapeamento entre as linguagens será ilustrado através de um estudo de caso, um algoritmo de eleição em anel [8].

A seção 2 apresenta as duas linguagens, GGBO (seção 2.1) e PROMELA (seção 2.2), discutindo certas características de ambas. A seção 2.3 resume a comparação entre as linguagens. Na seção 3 é apresentada uma abordagem para a verificação de modelos descritos em GGBO. Por fim, na seção 4, são colocadas as conclusões e os principais trabalhos futuros.

2 Especificação de Sistemas Distribuídos

Tradicionalmente, pode-se observar duas abordagens utilizadas por desenvolvedores de sistemas distribuídos:

Baseada em objetos: utilizada pela comunidade não acadêmica para desenvolver aplicações distribuídas de grande porte. A linguagem mais usada é UML (*Unified Modeling Language*) [9], que se tornou um padrão na indústria. Na realidade, UML é um conjunto de linguagens para descrever os vários aspectos de um sistema. As linguagens mais utilizadas de UML são diagramas de classes para descrever a estrutura estática do sistema, *statecharts* para descrever o comportamento dinâmico de um objeto ou uma classe, e diagramas de interação (diagramas de seqüência ou diagramas de colaboração) para descrever as interações entre as classes. UML oferece métodos para permitir a estruturação de um sistema complexo usando conceitos de modularização e encapsulamento, facilitando o reuso de componentes. O grande problema da utilização de UML para modelar sistemas distribuídos é a dificuldade de verificação: a semântica da maioria das linguagens que compõem UML não está descrita de maneira formal, impossibilitando a realização de provas de propriedades de sistemas especificados nessas linguagens. Além disso, não há ferramentas para realizar simulações quantitativas que permitam avaliar, pelo menos de forma aproximada, o desempenho do sistema modelado.

Baseada em processos: por outro lado, a comunidade acadêmica da área de sistemas distribuídos utiliza com maior freqüência uma abordagem baseada em processos e troca de mensagens. Normalmente, são modelados algoritmos distribuídos para resolver uma função específica e necessária a um sistema distribuído. Não raro, a descrição do comportamento de um processo isolado é pequena e são poucos os tipos de mensagens trocadas. Mesmo assim, o entendimento do funcionamento concorrente de um conjunto de processos participando de um algoritmo distribuído não é trivial. Garantir que estes algoritmos realmente resolvem o problema proposto requer experiência em prova de teoremas (ver [8] para alguns exemplos de provas), ou o uso de ferramentas de suporte (como verificadores de modelos). Desta forma, encontramos com freqüência o emprego de formalismos de especificação que sirvam para uma posterior verificação do sistema modelado. O uso de verificadores de modelos é especialmente atrativo devido às facilidades de uso, se comparado a uma abordagem baseada em prova de teoremas.

A linguagem de especificação GGBO integra vantagens de ambas as abordagens. Esta linguagem foi proposta em [3] para modelar sistemas concorrentes reativos. Nesta seção iremos apresentar a GGBO (seção 2.1) e a linguagem PROMELA (seção 2.2). A seguir, faremos uma comparação de certos aspectos dessas duas linguagens (seção 2.3).

2.1 A Linguagem GGBO

As gramáticas de grafos [10] fornecem uma forma bastante natural de expressar situações complexas, onde os estados do sistema são descritos por grafos e os aspectos dinâmicos podem ser capturados pelas regras da gramática. Uma gramática de grafos é composta por:

- um grafo de tipos, que representa os tipos dos vértices e arestas permitidas no sistema;
- um grafo inicial, que representa o estado inicial do sistema;
- um conjunto de regras que descrevem as possíveis mudanças de estado que podem ocorrer em um sistema.

Em [3] é proposta uma restrição de gramática de grafos, chamada GGBO (Gramáticas de Grafos Baseadas em Objetos), para descrever sistemas baseados em objetos. Na GGBO um sistema consiste de entidades autônomas, chamadas de objetos. Os objetos possuem um estado interno e se comunicam através da troca de mensagens. Em [6] foi definido um modelo baseado na GGBO, onde os estados do sistema são descritos por hipergrafos. Hipergrafos são um tipo especial de grafos onde as arestas (hiperarcos) podem ter zero ou mais origens e destinos. Ao longo deste artigo utilizaremos este último modelo, chamando-o de GGBO.

Em uma GGBO, os estados de um sistema são descritos por grafos, onde os objetos são modelados como vértices e as mensagens são modeladas como (hiper)arcos. Os atributos de um objeto são arcos que partem do objeto e podem ligá-lo tanto a outros objetos como a valores de tipos de dados pré-definidos (estes também são modelados como vértices). Graficamente um objeto tem a notação de um retângulo, onde consta seu nome e o conjunto de atributos. Atributos de tipos de dados pré-definidos são listados dentro do retângulo, sendo que os atributos que referenciam outros objetos tem a notação de arestas que se ligam a outros objetos (ver figura 1).

O comportamento de um objeto corresponde às reações executadas por ele ao receber mensagens. Estas reações podem mudar o estado interno do objeto e/ou causar o envio de mensagens para outros objetos ou para si mesmo. As mensagens devem ter como destino um objeto e podem ter como argumentos outros objetos ou valores de tipos pré-definidos (ou seja, vértices do grafo). Graficamente uma mensagem tem a forma de um polígono como o da mensagem Msg na figura 1. O destino desta mensagem é dado por uma seta e os vários argumentos da mensagem são dados por linhas que ligam estes argumentos ao polígono da mensagem.

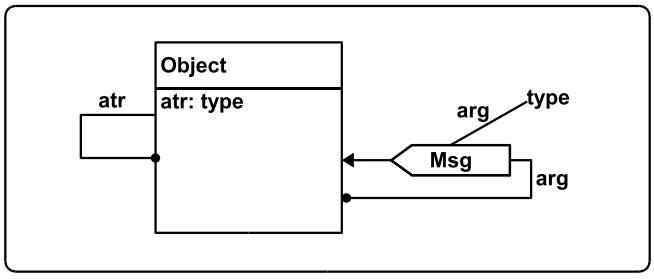


Figura 1: Grafo Modelo para Sistemas Baseados em Objetos

Em um sistema baseado em objetos podem existir vários tipos de objetos. Para descrever um sistema baseado em objetos usando GGBO é necessário definir o grafo tipo do sistema, que descreve os tipos de objetos que o compõe. O grafo tipo de um objeto define os atributos que um objeto tem e os tipos de mensagens que um objeto daquele tipo pode receber e enviar, bem como seus parâmetros. Como exemplo, vide figura 2 (a).

O grafo inicial de um sistema descreve todos os objetos, mensagens e valores de atributos que devem existir na situação inicial desejada do modelo. Estes objetos com seus atributos e mensagens são instâncias dos tipos de objetos do grafo tipo. Os objetos podem ser instanciados estaticamente (no grafo inicial) ou dinamicamente (através da execução de regras). Como exemplo de grafo inicial de um sistema, vide figura 2 (b).

Por fim, as regras modelam o comportamento de um tipo de objeto. Em uma gramática de grafos uma regra $r : L \rightarrow R$ especifica uma mudança de estado do sistema que ocorre da seguinte forma:

- Todos os ítens que estão no lado esquerdo da regra L devem estar presentes no estado atual do sistema para possibilitar a aplicação da regra;
- Todos os ítens que são mapeados de L para R (através do mapeamento r) são preservados;
- Todos os ítens que não são mapeados de L para R são apagados do estado atual;
- Todos os ítens que estão em R e não estão em L são adicionados para a obtenção do novo estado.

As regras de um tipo de objeto são as que apresentam, no lado esquerdo da regra, uma mensagem sendo recebida por um objeto do tipo em questão. Esta regra especifica a reação de um objeto daquele tipo à recepção daquela mensagem. No lado direito da regra, esta mensagem terá sido consumida, atributos do objeto podem ter mudado de valor, e novas mensagens podem ter sido geradas. Cada regra descreve o tratamento de apenas uma mensagem. Todas as ações descritas em uma regra ocorrem de forma atômica. Como exemplo, vide figura 3.

Uma GGBO fornece uma linguagem baseada em objetos para a especificação de sistemas. Essas especificações apresentam algumas características que facilitam o desenvolvimento do sistema que descrevem. Um sistema baseado em objetos é modular, uma vez que é composto por entidades autônomas (objetos) conectados via interfaces bem definidas (mensagens). Os atributos e as operações que manipulam os objetos são descritos juntos no objeto e não podem ser acessados por outros objetos (encapsulamento).

Um modelo baseado em objetos facilita o reuso das especificações devido ao encapsulamento e às interfaces de importação (operações que utiliza) e de exportação (operações que disponibiliza) de cada objeto. Assim objetos especificados em um modelo podem ser utilizados em outros modelos, sem a necessidade de conhecer a sua estrutura interna. Por questões de espaço, não foram mostradas neste artigo como são descritas as interfaces de importação e exportação de GGBO.

Um sistema descrito por um modelo baseado em objetos apresenta uma arquitetura simples (devido à abstração) e descentralizada (devido à sua estrutura de objetos), que são dois

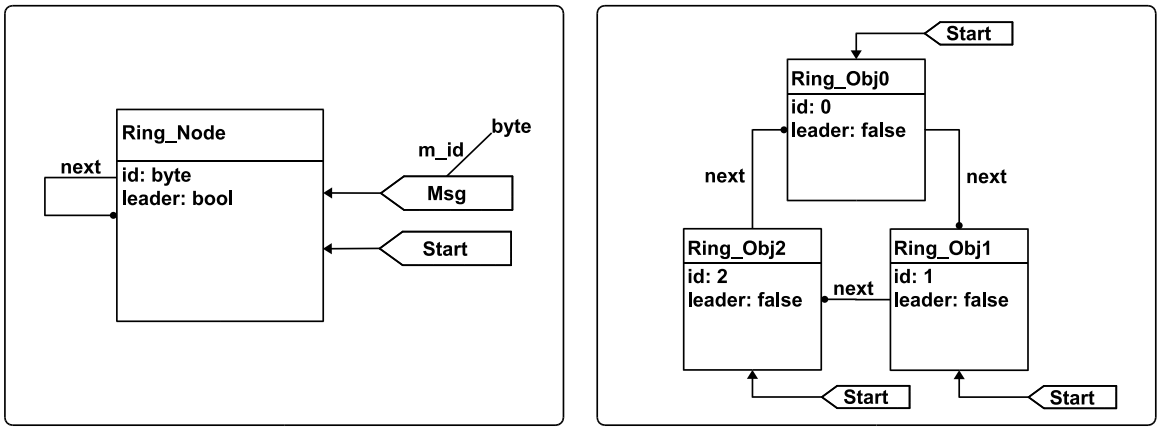
princípios necessários para a extensibilidade, onde novas funcionalidades podem ser incluídas no sistema com pouco esforço, sem que o resto do sistema tenha que ser alterado. Na GGBO, a extensão das funcionalidades de um objeto ou do sistema é alcançada através da inclusão de novos tipos de mensagens e objetos e da inclusão de novas regras.

Além disso, a GGBO permite modelar concorrência e não-determinismo. A concorrência entre objetos e a concorrência interna (um objeto pode tratar diversas mensagens ao mesmo tempo) são modeladas pela possibilidade da aplicação de diversas regras ao mesmo tempo, quando elas não são conflitantes. Uma regra r_1 é conflitante com uma regra r_2 se consomem (deletam) mesmo itens. O não-determinismo é modelado na escolha da regra para aplicação, isto é, se mais de uma regra puder ser aplicada em uma situação, uma dessas é escolhida não-deterministicamente para executar.

2.1.1 Exemplo: Algoritmo de Eleição em Anel

Um algoritmo de eleição em anel tem como objetivo eleger um entre um grupo de processos cooperantes para desempenhar uma função qualquer de interesse para o grupo. Segundo o algoritmo de eleição em anel encontrado em [8], o nodo com maior número de identificação deve ser eleito. Inicialmente cada nodo participante envia seu número de identificação ao próximo nodo do anel. Na recepção dessa mensagem um nodo pode: i) mandar a mensagem adiante caso o número de identificação for maior que o seu, ii) declarar-se líder caso o número de identificação for o seu, e iii) descartar a mensagem, caso contrário.

Para a especificação do algoritmo de eleição em anel em GGBO, define-se o tipo de objeto *Ring_Node* (figura 2 (a)) que compõe o modelo. O tipo de objeto *Ring_Node* é composto por três atributos: *next* (referência para o próximo nodo no anel), *id* (número de identificação do objeto), e *leader* (indica se o objeto é o líder ou não). Um grafo inicial para este modelo é apresentado na figura 2 (b), onde são instanciados três objetos do tipo *Ring_Node* e estes iniciam o seu funcionamento devido a recepção das mensagens *Start* definidas. Ao final da execução deste cenário, o objeto *Ring_Obj2* deverá tornar-se líder, pois apresenta o maior número de identificação.



(a) (b)

Figura 2: Grafo de Tipos *Ring_Node* (a) Grafo Inicial (b)

As regras que definem o comportamento dos objetos do tipo *Ring_Node* são definidas

na figura 3. Um objeto do tipo *Ring_Node*, ao receber uma mensagem *Start* (ver regra *Rule_Start*), inicia o seu funcionamento, enviando uma mensagem *Msg* (carregando o seu número de identificação) ao seu vizinho. Quando um objeto do tipo *Ring_Node* recebe uma mensagem *Msg* ele pode: reenviar a mensagem ao seu vizinho (regra *Rule_Msg0*, se o número de identificação da mensagem for maior que o seu), não fazer nada (Regra *Rule_Msg2*, se o número de identificação da mensagem for menor que o seu), ou se tornar líder (Regra *Rule_Msg1*, se o número de identificação da mensagem for igual ao seu).

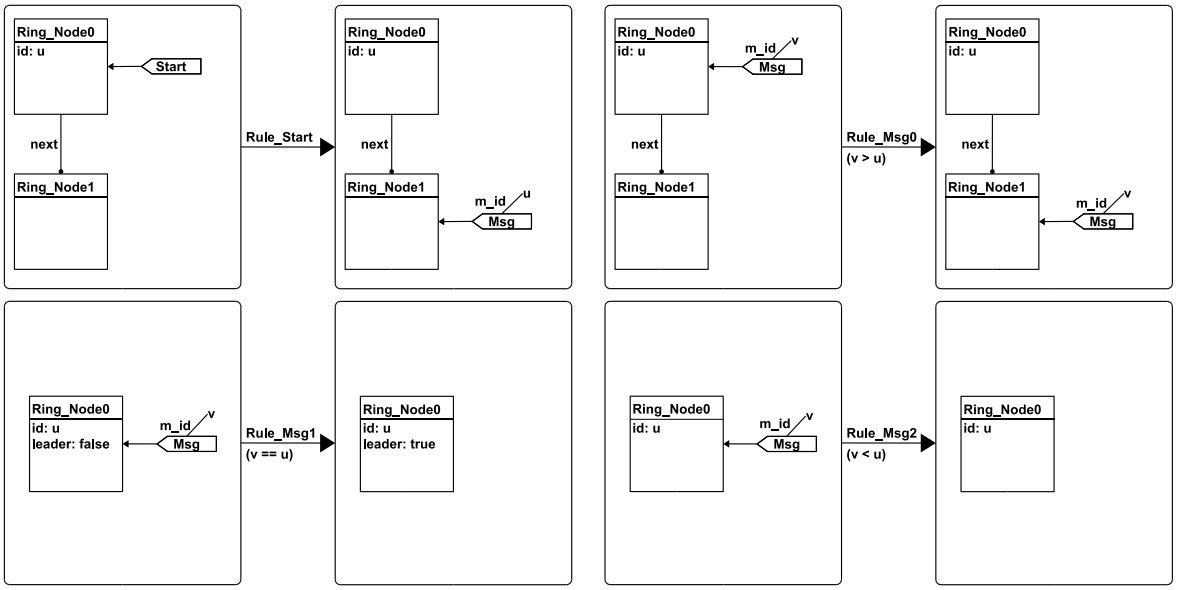


Figura 3: Regras para o Tipo de Objeto *Ring_Node*

2.2 A Linguagem PROMELA

PROMELA é uma linguagem de especificação formal baseada em processos, sendo utilizada como linguagem de entrada pelo verificador de modelos SPIN [11]. No verificador de modelos SPIN, a partir de especificações na linguagem PROMELA e propriedades especificadas usando a Lógica de Tempo Linear (LTL), é possível verificar especificações.

A linguagem PROMELA [12] caracteriza-se por possuir uma sintaxe *C-like* e usar construções para o envio e recepção de mensagens parecidas com as encontradas na linguagem de especificação formal *Communication Sequential Processes* (CSP). Como unidade básica de computação, PROMELA oferece processos. Os processos podem ser criados tanto de forma estática como dinâmica, i.e., um processo pode ser criado a qualquer momento dentro de uma especificação. Na linguagem utiliza-se um processo especial (*init*) para inicializar uma especificação.

Os processos podem trocar informações através de canais de mensagem ou variáveis globais (variáveis declaradas fora do escopo de um processo). Os canais de mensagem podem ser síncronos ou assíncronos. A diferença entre canais síncronos e assíncronos está na noção de *buffer*. Um canal assíncrono pode apresentar um *buffer* de **N** mensagens, enquanto um canal síncrono não apresenta *buffer* algum. Além disto, os canais são compostos pelos tipos de variáveis que podem trafegar no canal.

Na linguagem PROMELA o não-determinismo, é modelado através das estruturas de condição e de repetição. Uma estrutura de condição e de repetição é composta por comandos guardados. Um comando guardado, por sua vez, é um comando onde a não satisfação de sua condição implica no bloqueio do comando e, conseqüentemente, do processo que contém esta condição. O bloqueio ocorre até que a condição do comando guardado se torne verdadeira. Em uma estrutura de repetição ou de condição, se ao menos uma expressão for verdadeira, o comando é executado. Caso nenhuma expressão for verdadeira o processo é bloqueado até que alguma das expressões se torne verdadeira. O comportamento de não-determinismo ocorre quando mais de uma das expressões da estrutura de condição ou de repetição for verdadeira. Neste caso um dos caminhos possíveis é escolhido de forma não-determinística.

Também é possível definir seqüências atômicas em uma especificação. As seqüências atômicas caracterizam-se pela execução de um conjunto de declarações em um único passo, não intercalando a execução destas declarações com as de outros processos. As seqüências atômicas em PROMELA são definidas com o uso da estrutura *atomic*, onde as declarações dentro do escopo *atomic* são executadas em um único passo. Um ponto importante é que, se existirem comandos guardados dentro de uma estrutura *atomic* e estes não estiverem habilitados, irá ocorrer a perda da atomicidade, acarretando na execução intercalada de comandos do escopo *atomic* com comandos de outro(s) processo(s).

2.3 Resumo Comparativo

Na tabela 1 são apresentadas as principais características das linguagens de especificação apresentadas.

Tabela 1: Características das Linguagens de Especificação Apresentadas

Linguagem de especificação	**GGBO**	**PROMELA**
Verificação formal automatizada	N˜ao	Sim
Unidade básica de computação	Objeto	Processo
Criação de unidades básicas	Dinˆamica e est´atica	Dinˆamica e est´atica
Inicialização de especificações	Grafo inicial	Processo inicial
Forma de comunicação	Troca de mensagens	Troca de mensagens Mem´oria compartilhada
Não-determinismo	Escolha de regras	Estruturas de condiç ˜ao e repetiç ˜ao
Sequências atômicas	Regra	Estruturas do tipo *atomic*
Concorrência	Dentro de objetos Entre objetos	Entre processos

A fim de aliar as facilidades de desenvolvimento de GGBO com a verificação formal automatizada disponível em PROMELA (usando o SPIN), na próxima seção é apresentado o mapeamento de especificações na GGBO para PROMELA. A tabela 1 é usada como base para o mapeamento. Porém, como será explicado na próxima seção, para conservar a semântica de GGBO, as características de GGBO não são tão diretamente mapeadas para PROMELA.

3 Verificação de Sistemas Distribuídos

Esta seção apresenta uma abordagem para a verificação formal de sistemas distribuídos especificados em GGBO. Isto é feito através do mapeamento de um modelo GGBO para PROMELA. Este mapeamento é apresentado na seção 3.1, usando como estudo de caso o algoritmo de eleição em anel apresentado na seção 2.1.1. A partir disto é possível verificar propriedades, usando o SPIN, sobre as especificações GGBO mapeadas. Na seção 3.2 são apresentadas e verificadas duas propriedades sobre o estudo de caso mapeado.

3.1 Mapeamento de Especificações

O mapeamento de GGBO para PROMELA define como as abstrações de GGBO são mapeadas para elementos de PROMELA, de forma a preservar a semântica do modelo definido em GGBO. O quadro comparativo da tabela 1 pode sugerir que o mapeamento de abstrações de GGBO aconteça de maneira direta para as correspondentes em PROMELA. Será visto que para preservar a semântica de GGBO este mapeamento não é direto. A seguir são discutidas cada uma das abstrações de GGBO e sua correspondência em PROMELA.

Mapeamento de Objetos

Objetos de GGBO são mapeados para processos em PROMELA. É importante evidenciar que, para esse mapeamento ser possível, os modelos GGBO podem utilizar apenas os tipos básicos de dados existentes em PROMELA: bit, bool, byte, short, unsigned e int, e também vetores desses tipos de dados. Assim, atributos de objetos são mapeados diretamente para variáveis do processo representando o objeto. A concorrência entre objetos GGBO fica naturalmente preservada pela concorrência dos processos representantes em PROMELA. Porém, em GGBO é possível existir concorrência interna em objetos. Caso existam várias mensagens a serem processadas por um objeto no grafo de estado do sistema, pode-se ter a aplicação concorrente de regras para um mesmo objeto, cada uma tratando uma mensagem, desde que estas regras não sejam conflitantes. Assim, regras que não modificam o estado de um objeto podem ser executadas concorrentemente e regras que modificam o objeto devem executar de maneira exclusiva. As únicas operações possíveis para regras que não modificam o estado de um objeto são gerar novas mensagens e criar novos objetos. Segundo o mapeamento proposto, ambas operações se traduzem em criação de novos processos PROMELA, respectivamente representando mensagens e objetos.Como não se pode assumir nada sobre o progresso dos processos criados, a execução de regras não conflitantes pode se dar em qualquer ordem, modelando com fidelidade a semântica desejada. Com isso, um processo representando um objeto assume um comportamento cíclico que é receber uma mensagem, achar uma regra para tratá-la (executar a regra se estiver habilitada), e retornar para ler outra mensagem no seu canal de entrada. A procura de uma regra para tratar uma mensagem respeita o não-determinismo de GGBO, ou seja, se existe mais de uma regra habilitada para tratar uma mensagem então uma delas é escolhida não-deterministicamente. Isto é feito através da estrutura de condição em PROMELA que oferece escolha não-determinística.

Estas abstrações podem ser exemplificadas com o estudo de caso de eleição em anel. Este exemplo também serve para entrar em maiores detalhes do mapeamento, como conversão de nomes de objetos para processos, de atributos de objetos para variáveis de processos,

de parâmetros de inicialização de processos, além de explicar os canais de comunicação e sincronização necessários. Estes detalhes do mapeamento são genéricos para outros casos também.

Mapeamento do objeto *Ring_Node*: o objeto *Ring_Node* mapeado de GGBO para PROMELA é apresentado na figura 4 (linhas 1 a 11). Os parâmetros de criação deste processo são: o canal usado para a recepção de mensagens (linha 1) e os atributos mapeados do tipo de objeto GGBO, apresentando o prefixo *atr* (linha 1). Na linha 2 é declarada a variável *msg_name* que contém, após a recepção de uma mensagem, o nome da mensagem recebida. O canal definido na linha 3, após a recepção de uma mensagem, irá conter a referência para o canal de confirmação de uma mensagem (ver **Mapeamento de Mensagens**). Os parâmetros das mensagens são mapeados de forma direta, mas apresentam o prefixo *par* (linha 4). Após estas declarações, a entidade entra em uma estrutura de repetição (linhas 5 a 10). Esta estrutura tem como objetivo receber uma mensagem pelo canal de recepção de mensagens (linha 7), e tentar encontrar regras que possam tratar a mensagem (linha 8). Este tratamento continua na figura 6, onde tenta-se encontrar uma regra para tratar a mensagem e, ao mesmo tempo, verificar se o lado esquerdo da regra é satisfeito. Note que se mais de uma regra puderem tratar uma mesma mensagem (elas têm seu lado esquerdo satisfeito), uma das regras é escolhida para aplicação de maneira não-determinística, preservando a semântica de GGBO.

```
1  proctype Ring_Node(chan this_chan; bool atr_leader; chan atr_next; byte atr_id) {
2    mtype msg_name;
3    chan msg_confirm;
4    byte par_m_id;
5    do
6    ::atomic {
7        this_chan?msg_name, msg_confirm, par_m_id;
8        rules_Ring_Node();
9      }
10   od;
11 }
```

Figura 4: Mapeamento do Objeto *Ring_Node*

Mapeamento de Mensagens

No grafo de estado de um sistema GGBO, um objeto pode ter várias mensagens endereçadas a ele. Estas mensagens são consumidas de forma não-determinística. Mensagens para as quais não existe uma regra habilitada permanecem no grafo de estado do sistema. Em PROMELA, as mensagens endereçadas a um processo são consumidas na ordem de chegada ao canal de comunicação daquele processo. Assim, não é suficiente mapear mensagens endereçadas a um objeto para mensagens enviadas ao canal do processo representando o objeto. Este mapeamento não suportaria o não-determinismo na escolha da(s) mensagem(ns) a tratar em um instante. Segundo a abordagem definida neste artigo, mensagens de GGBO são mapeadas para processos PROMELA. Enviar uma mensagem significa criar um processo que tem como parâmetro todos os argumentos da mensagem GGBO, além do canal de envio do processo representando o objeto destino. O processo representando uma mensagem fica bloqueado tentando enviar uma mensagem (em PROMELA) ao canal do processo representando o objeto destino. Como este canal é síncrono, quando o processo representando o objeto lê uma mensagem no seu canal de entrada, acontece a sincronização com um processo representando uma mensagem postada àquele objeto. Como podem existir vários

processos representando mensagens tentando sincronizar (enviar mensagem) com o mesmo processo representando o objeto, a escolha de qual processo sincroniza é não-determinística em PROMELA, modelando assim a semântica desejada de não-determinismo na escolha de mensagens a consumir em GGBO.

Caso um processo representando uma mensagem consiga enviar uma mensagem (em PROMELA), ele espera pela confirmação da aplicação de uma regra que consuma esta mensagem. Se uma regra em PROMELA estiver habilitada para tratar esta mensagem, a regra em PROMELA envia uma confirmação ao processo representando a mensagem informando que ela foi consumida. Isso culmina na finalização do processo representando a mensagem. Caso nenhuma regra em PROMELA estiver habilitada para a mensagem (em PROMELA), o processo que representa o objeto volta a ler uma nova mensagem do seu canal de entrada. Desta forma, irá ocorrer um *time-out* pelo processo que representa a mensagem, levando ao reenvio da mensagem (em PROMELA) ao canal do processo que representa o objeto.

Mapeamento da mensagem *Start*: a mensagem *Start* mapeada de GGBO para PROMELA pode ser vista na figura 5. Na linha 1 são definidos todos os nomes das mensagens que fazem parte da especificação, estes nomes simbólicos seguem a sintaxe ***nomeObjeto_nomeMensagem***. Na linha 3 é declarada uma variável global que contém, a cada instante, o número de mensagens presentes no modelo (isso é usado para o reenvio de mensagens). O processo que representa a mensagem *Start* de GGBO em PROMELA (linhas 5 a 17) tem como parâmetro (linha 5) seu destino (*msg_send*), e apresenta a sintaxe *msg_**nomeObjeto_nomeMensagem***. Como a mensagem *Start* não tem parâmetros, seu processo correspondente também não apresenta parâmetros. É criado um canal (linha 6) usado para confirmar se a mensagem foi consumida. O envio da mensagem se dá através do canal *msg_send* (linha 8), onde são passados o nome da mensagem (usando a sintaxe mapeada de GGBO para PROMELA), o canal usado para confirmação, e os parâmetros das mensagens. Neste caso, como não existem parâmetros na mensagem *Start*, os parâmetros, que não são usados, têm valor 0. Após o envio da mensagem, o processo que representa a mensagem GGBO espera pela confirmação de que a mensagem foi consumida (linha 10), decrementando em 1 o número de mensagens atuais no modelo e terminando sua execução. Caso a mensagem não tenha sido consumida, o processo tenta reenviar a mensagem (linhas 14 e 15).

```
1  mtype = { Ring_Node_Msg, Ring_Node_Start };
2
3  unsigned messages:8;
4
5  proctype msg_Ring_Node_Start(chan msg_send)  {
6    chan mc = [0] of { bool };
7  RM:
8    msg_send!Ring_Node_Start, mc, 0;
9    if
10   ::atomic {
11       mc?_;
12       messages = messages - 1;
13     }
14   ::(timeout) && (messages == 1) -> goto RM;
15   ::(timeout) && (_last != _pid) -> goto RM;
16   fi;
17 }
```

Figura 5: Mapeamento da Mensagem *Start*

Mapeamento de Regras

A escolha de regras a serem aplicadas na GGBO se dá a partir de uma ocorrência da regra, que é um mapeamento do lado esquerdo da regra para o grafo estado. Caso mais de uma regra esteja habilitada, uma destas regras é escolhida de forma não determinística para aplicação. A abordagem definida para mapear a escolha de regras, seguindo esta semântica de GGBO, foi de criar uma estrutura de condição em PROMELA. As entradas desta estrutura de condição PROMELA são compostas pelas ocorrências das regras GGBO. Assim, se mais de uma entrada estiver habilitada, uma delas é escolhida de forma não determinística para executar, seguindo a semântica de GGBO.

Mapeamento das regras *Rule_Msg1* e *Rule_Start*: as regras *Rule_Msg1* e *Rule_Start* mapeadas de GGBO para PROMELA são apresentadas na figura 6. A escolha de regras a serem aplicadas é feita segunda uma estrutura de condição (linhas 2 a 13). Observa-se que, logo após a aplicação de uma regra, é enviada uma confirmação positiva ao processo representando a mensagem, indicando que a mensagem (em GGBO) foi consumida e a regra aplicada (linhas 6 e 11). Além disso, se a regra gerar mensagens, o número de mensagens atuais no modelo é incrementado de acordo com esse valor (linha 10). Caso uma mensagem (em PROMELA) seja recebida e não exista nenhuma regra habilitada para a mesma, o processo representando o objeto (linha 12) retorna para receber uma nova mensagem pelo seu canal de entrada.

```
1  inline rules_Ring_Node() {
2    if
3    ...
4    ::(msg_name == Ring_Node_Msg) && (par_m_id == atr_id);
5        atr_leader = true;
6        msg_confirm!true;
7    ...
8    ::(msg_name == Ring_Node_Start);
9        run msg_Ring_Node_Msg(atr_next, atr_id);
10       messages = messages + 1;
11       msg_confirm!true;
12   ::else;
13   fi;
14 }
```

Figura 6: Mapeamento das Regras *Rule_Msg1* e *Rule_Start*

Mapeamento do Grafo Inicial

O grafo inicial de GGBO é composto pelas instâncias dos objetos que representam o modelo e pelas mensagens iniciais do modelo. Estas instâncias têm os seus atributos internos inicializados. No mapeamento para PROMELA o grafo inicial de GGBO é mapeado para um processo inicial. Este processo é composto por: i) criação de canais de comunicação síncronos usados na recepção de mensagens pelos processos que representam objetos; ii) criação dos processos que representam os objetos, passando como parâmetros os valores iniciais dos atributos; iii) criação dos processos que representam as mensagens definidas no grafo inicial.

Mapeamento do grafo inicial: a figura 7 apresenta o mapeamento do grafo inicial do estudo de caso. Como dito anteriormente, o grafo inicial é traduzido para um processo inicial em PROMELA (linhas 1 a 17). Das linhas 3 a 5 aparece a criação dos canais usados, por processos que representam objetos, na recepção de mensagens. Estes canais apresentam o mesmo nome dos objetos definidos no grafo inicial. Os processos que representam os objetos

do grafo inicial são criados nas linhas 7 a 9, onde são passados como parâmetros os valores dos atributos iniciais. Nas linhas 11 a 13 são criados os processos que representam as mensagens do grafo inicial. Por fim, na linha 15 é atribuído o valor de mensagens atuais no modelo (de acordo com o número de mensagens definidas no grafo inicial).

```
1  init {
2    atomic {
3      chan Ring_Obj0 = [0] of { mtype, chan, byte };
4      chan Ring_Obj1 = [0] of { mtype, chan, byte };
5      chan Ring_Obj2 = [0] of { mtype, chan, byte };
6
7      run Ring_Node(Ring_Obj0, false, Ring_Obj1, 0);
8      run Ring_Node(Ring_Obj1, false, Ring_Obj2, 1);
9      run Ring_Node(Ring_Obj2, false, Ring_Obj0, 2);
10
11     run msg_Ring_Node_Start(Ring_Obj0);
12     run msg_Ring_Node_Start(Ring_Obj1);
13     run msg_Ring_Node_Start(Ring_Obj2);
14
15     messages = 3;
16   }
17 }
```

Figura 7: Mapeamento do Grafo Inicial

3.2 Especificação e Verificação de propriedades

Na verificação usando o SPIN, as fórmulas em LTL que representam as propriedades a serem verificadas podem usar variáveis globais. Com o uso destas variáveis globais são definidas proposições que irão formar, junto a operadores lógicos e temporais, as especificações das propriedades. Devido a GGBO ser uma linguagem baseada em objetos, e não apresentar a noção de variáveis globais, é necessário utilizar uma forma para representar as propriedades em PROMELA que seja, sobre um modelo mapeado, o menos intrusiva possível.

A abordagem escolhida aqui foi tornar visível, em termos de variáveis globais do modelo mapeado, as variáveis dos processos (atributos de objetos) que se quer utilizar para verificação. Assim, sempre que há uma mudança em uma variável do processo que representa o objeto que seja de interesse para verificação, deve-se atualizar uma variável global correspondente. Com estas variáveis globais pode-se, então, enunciar propriedades em LTL para o modelo.

Para o algoritmo de eleição em anel a principal propriedade que deve ser satisfeita é de que "nunca existirá mais de um líder". Outra propriedade mais específica para o modelo em questão é a de que "o objeto *Ring_Obj2* irá se tornar líder", já que ele é o objeto com maior identificação no anel.

Mais especificamente, para o estudo de caso define-se um vetor de valores booleanos (*bool leader[3]*) que deve conter o valor da variável *leader* referente a cada um dos objetos *Ring_Obj0*, *Ring_Obj1* e *Ring_Obj2*. Para isso, a cada mudança do valor da variável *leader* (ver regra *Rule_Msg1*, figura 3), o valor da variável é copiado para o vetor, usando a variável *id* de cada objeto como índice. Desta forma as seguintes proposições são definidas sobre o modelo: $l0f$ ($leader[0] == false$); $l1f$ ($leader[1] == false$); $l2f$ ($leader[2] == false$); $l0t$ ($leader[0] == true$); $l1t$ ($leader[1] == true$); e $l2t$ ($leader[2] == true$). A partir destas definições é possível especificar as fórmulas em LTL que irão representar as propriedades definidas acima. Com isso, as propriedades acima tornam-se as seguintes fórmulas:

- $[]\ (l0f\ \&\&\ l1f\ \&\&\ l2t)\ ||\ (l0f\ \&\&\ l1t\ \&\&\ l2f)\ ||\ (l0t\ \&\&\ l1f\ \&\&\ l2f)\ ||\ (l0f\ \&\&\ l1f\ \&\&\ l2f)$ - esta fórmula diz que ou não existe líder (no início do algoritmo nenhum processo é líder) ou somente um processo (representando um objeto) será líder em um momento. Esta fórmula é válida após a verificação usando o SPIN;

- $<>\ \ l2t$ - esta fórmula é mais específica ao estudo de caso, sendo válida após a verificação usando o SPIN. Ela diz que em todas as execuções o eleito será o objeto *Ring_Obj2*.

Esta abordagem para especificar propriedades sobre modelos GGBO mapeados, foi utilizada por apenas inserir no modelo PROMELA a atribuição de valores a variáveis globais usados para verificação, sendo, segundo nossa análise, a menos intrusiva possível. No entanto, esta abordagem se presta para a verificação de propriedades sobre modelos GGBO sem criação dinâmica de objetos. A criação dinâmica de objetos implica na criação dinâmica de variáveis globais (caso se queira utilizar na verificação estados de objetos criados dinamicamente) o que não é possível. Uma abordagem para a verificação de modelos com criação dinâmica de objetos está sendo investigada.

Na tabela 2 são apresentados resultados de complexidade (número de estados, memória utilizada e tempo) sobre a verificação da segunda fórmula ($<>\ \ l2t$). Visto que estes resultados oferecem uma visão geral sobre a complexidade de verificação usando o mapeamento proposto, e devido a restrições de espaço, resultados relativos a primeira propriedade não são apresentados. Utiliza-se um número crescente de participantes (definidos no grafo inicial), onde a fórmula vai mudando para refletir o novo líder (o objeto com maior identificador). Estas verificações foram realizadas em uma máquina Compaq ProLiant ML350, usando um limite de 500 *Mb* de memória no SPIN. Como pode ser visto na tabela 2, quando o número de objetos é igual a 6 a memória chega ao seu limite. Em parte, este alto número de estados e de memória, se deve as mensagens tornarem-se processos (criados dinamicamente) no mapeamento.

Tabela 2: Resultados sobre a verificação da segunda fórmula

Número de objetos	**Número de estados**	**Memória utilizada**	**Tempo total**
3	858	2,622 Mb	0,05 seg.
4	32.602	5,899 Mb	1 seg.
5	1.653114e+06	216,331 Mb	38 seg.
6	-	>500 Mb	1,30 min.

4 Conclusão

Neste artigo foram comparadas as linguagens de especificação de sistemas distribuídos GGBO e PROMELA. A primeira é uma linguagem que segue o paradigma de objetos, e a segunda é uma linguagem orientada a processos. Por ser uma linguagem visual com primitivas simples e intuitivas, GGBO parece ser adequada para a modelagem de sistemas distribuídos complexos. Para esta linguagem, já está disponível um simulador [4] [5] e uma ferramenta de geração

automática de código [13]. Além disso, foram realizados estudos sobre a complexidade das especificações [7], considerando a troca de mensagens como operação principal, ou seja, a complexidade é medida em termos de número de trocas de mensagens necessárias para completar alguma tarefa. Essa medida de complexidade é bastante útil como medida de desempenho. Porém, ainda não existia a possibilidade de utilizar um verificador de modelos para analisar propriedades funcionais de especificações em GGBO. Com a tradução apresentada neste artigo, isto se torna possível.

Traduzir linguagens de especificação ou programação para linguagens de entrada de verificadores de modelos é uma prática que está se tornando padrão, pois muitas vezes fazer tal tradução é mais simples do que desenvolver um verificador específico. Porém, estas traduções envolvem comparações detalhadas, principalmente no nível semântico, das linguagens envolvidas. Em [6] foi definida uma tradução de GGBO para o cálculo π, que possibilitaria o uso de verificadores de propriedades de cálculo π para analisar especificações GGBO. A abordagem de tradução para PROMELA parece mais interessante, porque o verificador de modelos SPIN é um dos mais utilizados e otimizados atualmente, permitindo verificar sistemas com tamanho razoável.

Encontra-se na literatura um conjunto de trabalhos recentes voltados à verificação de sistemas distribuídos baseados/orientados em/a objetos. O trabalho proposto em [14] tem como objetivo definir uma linguagem de descrição visual e orientada a objetos que possa ser mapeada para o verificador de modelos SPIN. Já em [15] a linguagem de descrição PROMELA é estendida, considerando o modelo de concorrência *actors*. Em [16] é proposta uma ferramenta que tenta disponibilizar a verificação automática de sistemas descritos na linguagem de modelagem UML. Esta abordagem consiste em mapear as descrições em UML para a linguagem de descrição PROMELA. Em [17] é proposta a integração da linguagem de especificação formal Object-Z com ASM (*Abstract State Machine*), criando uma notação chamada OZ-ASM. Esta linguagem passa por uma série de conversões, sendo possível verificá-la no verificador de modelos SMV [18]. Ao contrário da maioria dessas traduções, estamos apresentando explicitamente a tradução de GGBO para PROMELA, e também comparando as linguagens do ponto de vista semântico (pelas restrições de espaço, não foi possível entrar em detalhes sobre a compatibilidade semântica da tradução proposta). A GGBO é um formalismo com alto nível de abstração, permitindo a modelagem de atributos básicos, assim como de tipos abstratos de dados. A GGBO apresenta as mesmas abstrações de encapsulamento e troca de mensagens dos trabalhos acima apresentados. Conforme mencionado acima, a partir de um sistema descrito em GGBO pode-se aplicar técnicas para simulação e/ou geração de código para execução em um ambiente real, fazer análises de complexidade, além da abordagem de verificação formal, proposta neste artigo.

Como trabalhos futuros, pretende-se automatizar essa tradução, definir uma forma de especificar propriedades diretamente sobre modelos em GGBO (ao invés de usar o modelo mapeado), investigar maneiras de realizar verificação em sistemas com criação dinâmica de objetos, e realizar estudos de caso de maior porte para analisar a escalabilidade da abordagem.

Referências

[1] F. M. A. Silva. *A transaction model based on mobile agents*. Tese de doutorado, Technical University Berlin - FB-Informatik, Germany, 1999.

[2] E. M. Clarke, J. M. Wing, R. Alur, et al. Formal methods: state of the art and future directions. *ACM Computing Surveys*, 28(4):626–643, 1996.

[3] F. L. Dotti and L. Ribeiro. Specification of mobile code systems using graph grammars. In *4th International Conference on Formal Methods for Open Object-Based Distributed Systems*, volume 177 of *IFIP Conference Proceedings*, pages 45–63, USA, 2000. Kluwer.

[4] B. Copstein, M. C. M´ora, and L. Ribeiro. An environment for formal modeling and simulation of control systems. In *33rd Annual Simulation Symposium*, pages 74–82, USA, 2000. IEEE Computer Society.

[5] F. L. Dotti, L. M. Duarte, B. Copstein, and L. Ribeiro. Simulation of mobile applications. In *Communication Networks and Distributed Systems Modeling and Simulation Conference*, pages 261–267, USA, 2002. The Society for Modeling and Simulation International.

[6] L. Foss. Uma traduç ˜ao de gram´aticas de hipergrafos baseados em objetos para c´alculo-π. Dissertaç ˜ao de mestrado, UFRGS - II - PPGC, Brasil, 2003.

[7] A. B. Loreto, L. Ribeiro, and L. Toscani. Complexity analysis of reactive graph grammars. *Revista de Informática Teórica e Aplicada - UFRGS*, 7(1):109–128, 2000.

[8] N. A. Lynch. *Distributed algorithms*, pages 476–482. Morgan Kaufmann, USA, 1996.

[9] G. Booch, J. Rumbaugh, and I Jacobson. *The unified modeling language users guide*. Addison Wesley, USA, 1999.

[10] H. Ehrig. Introduction to the algebraic theory of graph grammars. In *1st International Workshop on Graph Grammars and Their Application to Computer Science and Biology*, volume 73 of *LNCS*, pages 1–69. Springer, 1979.

[11] G. J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.

[12] Research Bell-Labs. PROMELA language reference. http://cm.bell-labs.com/cm/cs/what/spin/Man/promela.html, 2003.

[13] L. M. Duarte. Desenvolvimento de sistemas distribu´ıdos com c´odigo m´ovel a partir de especificaç ˜ao formal. Dissertaç ˜ao de mestrado, PUCRS - FACIN - PPGCC, Brasil, 2001.

[14] S. Leue and G. Holzmann. v-Promela: a visual, object oriented language for SPIN. In *2nd International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 14–23, France, 1999. IEEE Computer Society.

[15] S. M. Cho, D. H. Bae, S. D. Cha, et al. Applying model checking to concurrent object-oriented software. In *4th International Symposium on Autonomous Decentralized Systems*, pages 380–383, Japan, 1999. IEEE Computer Society.

[16] J. Lilius and I. P. Paltor. vUML: a tool for verifying UML models. In *14th International Conference on Automated Software Engineering*, pages 255–258, USA, 1999. IEEE Computer Society.

[17] K. Winter and R. Duke. Model checking Object-Z using ASM. In *3rd International Conference on Integrated Formal Methods*, volume 2335 of *LNCS*, pages 165–184. Springer, 2002.

[18] K. L. MacMillan. *Symbolic model checking: an approach to the state explosion problem*. Tese de doutorado, Carnegie Mellon University, USA, 1992.