Proposta de uma abordagem para a verificação formal de Sistemas Distribuídos Baseados em Objetos*

Osmar Marchi dos Santos†, Fernando Luís Dotti

Faculdade de Informática
Pontifícia Universidade Católica do Rio Grande do Sul
CEP 90619-900 - Porto Alegre, RS

{osantos, fldotti}@inf.pucrs.br

Resumo. *Este artigo propõe uma abordagem para a verificação formal, através da técnica de verificação de modelos, de Sistemas Distribuídos Baseados em Objetos (SDBOs). A linguagem de descrição utilizada para modelar SDBOs é uma forma restrita de Gramática de Grafos, um formalismo gráfico, declarativo, e que suporta paralelismo implícito. Esta linguagem, chamada Gramática de Grafos Baseada em Objetos (GGBO), é brevemente apresentada. Os modelos descritos na GGBO são mapeados para a linguagem de descrição PROMELA (PROtocol/PROcess MEta LAnguage), usada pelo verificador de modelos SPIN (Simple Promela INterpreter). Este mapeamento é descrito. A partir disso, propriedades sobre modelos descritos na GGBO podem ser especificadas e verificadas através do SPIN. Um exemplo de descrição na GGBO é apresentado e mapeado.*

1. Introdução

O constante crescimento na área de computação é refletido tanto em sistemas de *hardware* (novas tecnologias são lançadas no mercado) como em sistemas de *software* (novos paradigmas de aplicação tal como, por exemplo, mobilidade de código são inseridos). Estas novas funcionalidades quando incorporadas aos sistemas muitas vezes apresentam erros. Erro é uma parte do estado do sistema que pode levar o próprio sistema a um possível defeito. Defeito é um evento que ocorre quando a entrega de um serviço desvia do serviço correto [Avizienis et al., 2001]. Porém, sistemas livres de erros são difíceis de alcançar, senão impossíveis. Caso um sistema complexo, como por exemplo, de controle de aviões, apresente erros não antes verificados e tratados pessoas podem vir a morrer. Este é apenas um exemplo, outros tipos de catástrofes como perda de tempo ou dinheiro podem vir a ocorrer [Clarke et al., 1996].

Como pode ser visto, existe uma forte demanda pela criação de sistemas confiáveis. Em específico, a tarefa de desenvolver sistemas distribuídos é considerada complexa pois o sistema pode apresentar inúmeras configurações possíveis. A partir disto, argumenta-se pela necessidade de um método que permita garantir, ainda na fase de desenvolvimento, que um sistema distribuído mantém certas propriedades desejadas, tornando-o mais confiável.

*Este trabalho é parcialmente financiado pela FAPERGS e pelo CNPq.
†Este autor é parcialmente financiado pelo CNPq.

Um dos principais objetivos do projeto ForMOS (Métodos Formais para Código Móvel em Sistemas Abertos[1]) é o desenvolvimento de um *framework* onde diferentes métodos e ferramentas coexistem para facilitar a construção de sistemas distribuídos: simulação, geração de código e, de maior ênfase neste artigo, a verificação formal. De forma mais concreta, foi desenvolvida [Dotti and Ribeiro, 2000] uma linguagem de especificação formal voltada para sistemas distribuídos assíncronos. Esta linguagem, chamada Gramática de Grafos Baseada em Objetos (GGBO), é utilizada como formalismo integrador entre os diferentes métodos e ferramentas desenvolvidos no projeto. Atualmente o *framework* funciona da seguinte forma: o desenvolvedor define o seu modelo usando a GGBO e pode simular o comportamento deste modelo [Copstein et al., 2000]. Assim que o desenvolvedor estiver satisfeito com o comportamento do modelo ele pode gerar código para execução em um ambiente real [Duarte, 2001]. Além disto, em [Rödel, 2003] foi definida uma forma de inserir certos comportamentos falhos (e.g. modelo de falha *crash*) em uma descrição na GGBO existente. Nesta abordagem um modelo M_1 sem falhas é transformado em um modelo M_2 que apresenta o comportamento falho selecionado. A partir deste modelo M_2 o desenvolvedor pode raciocinar (atualmente via simulação) sobre o seu modelo M_1 na presença de certas falhas, possivelmente incluindo mecanismos de detecção e tolerância a falhas no modelo M_1. Uma vez que o desenvolvedor termina de analisar o comportamento do modelo M_2, i.e. o modelo M_1 com a presença de certas falhas, ele pode gerar, usando o modelo M_1 (o modelo que não contém a adição de comportamento falho), código para execução em um ambiente real.

Neste artigo é apresentada uma proposta para a verificação formal, baseada na técnica de verificação de modelos, de Sistemas Distribuídos Baseados em Objetos (SDBOs). A abordagem usada para a verificação formal consiste em mapear descrições na GGBO para a linguagem de descrição PROMELA (*PROtocol/PROcess MEta LAnguage*), utilizada pelo verificador de modelos SPIN (*Simple Promela INterpreter*). A partir disto, as propriedades desejadas do sistema podem ser especificadas em lógica temporal e verificadas automaticamente. Apesar de se tratar de uma proposta e estar com vários pontos em desenvolvimento espera-se, futuramente, se obter uma forma de integrar esta abordagem no *framework* apresentado acima. Assim o desenvolvedor pode utilizar, além da simulação, a verificação formal para raciocinar sobre o seu modelo. Em especial, a integração deste trabalho com a abordagem definida em [Rödel, 2003] facilitaria bastante a construção de sistemas distribuídos considerando certos tipos de falhas.

Na Seção 2. são comentados os trabalhos relacionados. Já na Seção 3. são apresentadas noções básicas sobre a técnica de verificação de modelos e o verificador de modelos SPIN. Na Seção 4. é apresentado o formalismo GGBO e a modelagem de um algoritmo de eleição em anel, ilustrando o uso da GGBO. A proposta deste artigo é vista na Seção 5., onde o exemplo definido é mapeado para a linguagem de descrição PROMELA. Por fim, na Seção 6. são colocadas as conclusões e definidos os principais trabalhos futuros.

2. Trabalhos Relacionados

Encontra-se na literatura um conjunto de trabalhos recentes voltados à verificação de sistemas distribuídos baseados/orientados em/a objetos. O trabalho proposto em

[1]Suporte FAPERGS e CNPq.

[Leue and Holzmann, 1999] 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 [Cho et al., 1999] a linguagem de descrição PROMELA é extendida considerando o modelo de concorrência *actors*.

Em [Lilius and Paltor, 1999] é proposta uma ferramenta que tenta disponibilizar a verificação automática de sistemas descritos na linguagem de modelagem UML (*Unified Modelling Language*). Esta abordagem consiste em mapear as descrições na UML para a linguagem de descrição PROMELA. Outro trabalho encontrado na literatura [Winter and Duke, 2002] consiste em integrar a 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 (*Symbolic Model Verifier*).

A GGBO é um formalismo com alto nível de abstração, permitindo a modelagem de atributos básicos, assim como de tipos abstratos de dados. Além disso, a GGBO apresenta as mesmas abstrações de encapsulamento e troca de mensagens dos trabalhos acima apresentados. Ainda, conforme já mencionado, a partir de um sistema descrito na GGBO pode-se aplicar técnicas para simulação e/ou geração de código para execução em um ambiente real, além da abordagem de verificação formal, inicialmente proposta neste artigo.

3. Verificação de Modelos

A verificação de modelos consiste em uma técnica automática, sendo empregada para a verificação de sistemas reativos com estados finitos, tais como projetos de protocolos de comunicação [Clarke et al., 1999]. Na maioria das vezes as propriedades a serem verificadas são expressas através de lógica temporal, um tipo de lógica modal que possibilita descrever certas ocorrências de eventos sobre o tempo.

Os verificadores de modelos trabalham sobre um modelo definido através de uma linguagem de descrição (que deve apresentar uma semântica formal) e certas propriedades especificadas em lógica temporal [Manna and Pnueli, 1992]. As propriedades são verificadas sobre o modelo e, ao final desta verificação, é informado ao usuário se a propriedade é válida (verdadeira), ou não (falsa). Caso não seja válida, o verificador de modelos fornece um contra-exemplo da propriedade que consiste na sequência de execução do modelo até o local onde a propriedade é dada como falsa.

3.1. Verificador de Modelos SPIN

O ambiente SPIN [Holzmann, 1997] possibilita a verificação de modelos descritos utilizando a linguagem de descrição PROMELA. Para a especificação de propriedades o SPIN utiliza a lógica de tempo linear (LTL - *Linear Temporal Logic*).

A linguagem de descrição PROMELA apresenta uma sintaxe *C-like*, com características da linguagem de especificação formal CSP (*Communicating Sequential Processes*). Descrições em PROMELA consistem de processos, que podem trocar informações através de canais de mensagem e/ou variáveis globais. O não-determinismo em PROMELA é modelado nas estruturas de repetição/condição. Além disto é possível definir sequências atômicas na linguagem, ou seja, a execução de um conjunto de declarações em um único passo.

4. Gramática de Grafos Baseada em Objetos

Uma Gramática de Grafos [Ehrig, 1979] é composta por um Grafo de Tipos (representa os tipos dos vértices e arestas permitidas na descrição), um Grafo Inicial (representa o estado inicial da descrição), e um conjunto de Regras (descrevem as possíveis mudanças de estado que podem ocorrer numa descrição).

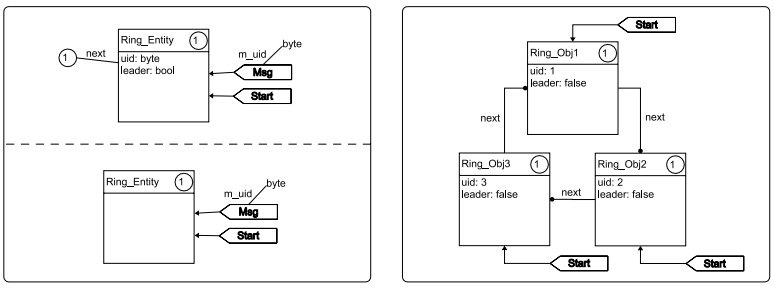
Em [Dotti and Ribeiro, 2000] é proposta a GGBO, uma restrição de Gramática de Grafos para descrever SBOs (Sistemas Baseados em Objetos). Na GGBO uma descrição consiste de objetos que possuem um estado interno e se comunicam através da troca de mensagens. Um objeto é definido segundo um Grafo de Tipos do objeto. O comportamento do objeto corresponde às reações executadas por ele (Regras do objeto) ao receber uma mensagem. Estas reações podem vir a mudar o estado interno do objeto e/ou causar o envio de mensagens para outros objetos e/ou para si mesmo. A instanciação dos objetos que definem uma descrição ocorrem no Grafo Inicial.

Algumas características importantes da GGBO são: apenas uma única mensagem é consumida na aplicação de uma Regra; o não-determinismo é modelado através da escolha implícita de uma Regra, i.e. quando mais de uma Regra se aplica a mesma mensagem, uma destas Regras é escolhida não-deterministicamente para executar; mais de uma Regra pode ser aplicada em paralelo quando as mensagens que as disparam estão presentes no atual estado do sistema (o Grafo) e as Regras não são conflitantes[2].

4.1. Algoritmo de eleição em anel

Nesta Seção é modelado o algoritmo de eleição em anel definido em [Lynch, 1996]. Neste algoritmo os objetos (do tipo *Ring_Entity*, Figura 1 (a)) que compõem a descrição são compostos por três atributos: *next* (referência[3] para o próximo nodo no anel), *uid* (número de identificação do objeto), e *leader* (indica se o objeto é o líder ou não).

O Grafo Inicial para esta descrição é apresentado na Figura 1 (b), onde são instanciados três objetos do tipo *Ring_Entity* e estes iniciam o seu funcionamento devido a recepção das mensagens *Start* definidas. Ao final da execução deste cenário, o objeto *Ring_Obj3* torna-se o líder pois apresenta o maior número de identificação. Devido a restrições de espaço, neste artigo não são definidas e verificadas propriedades sobre este exemplo.



(a) (b)

Figura 1: Grafo de Tipos *Ring_Entity* (a). Grafo Inicial (b).

[2]Neste caso, uma Regra R_1 é conflitante com uma Regra R_2 se consomem (apagam) mesmos ítens.
[3]Por uma questão de sintaxe as referências na GGBO não são definidas dentro do objeto.

As Regras que definem o comportamento dos objetos do tipo *Ring_Entity* são definidos na Figura 2. Como pode ser visto (Figura 2), um objeto do tipo *Ring_Entity* ao receber uma mensagem *Start* (Regra *RuleStart*) inicia o seu funcionamento, enviando uma mensagem *Msg* (carregando o seu número de identificação) ao seu vizinho. Quando um objeto do tipo *Ring_Entity* recebe uma mensagem *Msg* ele pode: reenviar a mensagem ao seu vizinho (Regra *RuleMsg_1*, se o número de identificação da mensagem for maior que o seu), não fazer nada (Regra *RuleMsg_3*, se o número de identificação da mensagem for menor que o seu), ou se tornar líder (Regra *RuleMsg_2*, se o número de identificação da mensagem for igual ao seu).

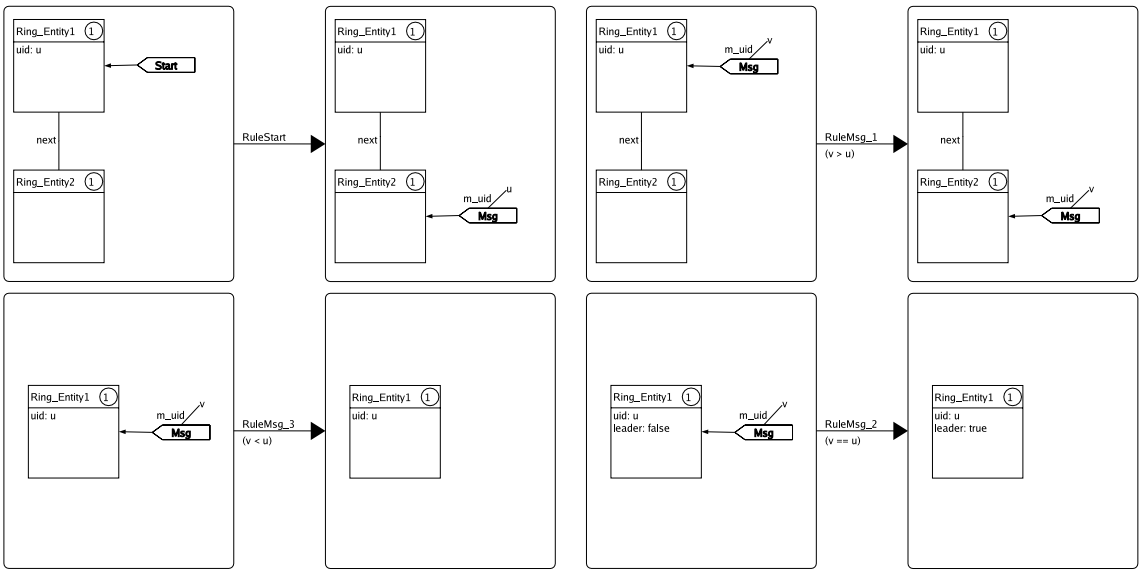


Figura 2: Regras para o tipo de objeto *Ring_Entity*.

5. Proposta para a verificação formal de Sistemas Distribuídos Baseados em Objetos

Esta Seção tem por objetivo apresentar o mapeamento de descrições na GGBO para a linguagem de descrição PROMELA. O exemplo de algoritmo de eleição definido anteriormente é usado para ilustrar este mapeamento.

Na Figura 3 é apresentado parte do tipo de objeto *Ring_Entity* mapeado, assim como parte do Grafo Inicial do exemplo definido. O código visto na Figura 3 contém inúmeros comentários (“/* ... */”), tornando mais fácil a compreensão do mapeamento realizado.

Os nomes de mensagens que compõem a descrição são definidas na Linha 3, estes nomes são usados na escolha de Regras a serem aplicadas. Um tipo de objeto na GGBO é mapeado para um processo em PROMELA (Linhas 26 a 36). O processo em PROMELA tem adicionado a ele um canal de comunicação assíncrono (Linha 26), definido a partir de todos os parâmetros de mensagens que este objeto pode receber (Linha 39). A partir da recepção de uma mensagem (Linha 33) o objeto escolhe (Linhas 5 a 17) uma mensagem que possa ser consumida, aplicando a Regra correspondente.

```
1  #define SIZE 3                  /* Tamanho dos canais de comunicação, definido pelo usuário */
2
3  mtype = { Msg, Start };                                /* Mensagens pertencentes a descrição */
4
5  inline rules_Ring_Entity() {                           /* Escolha de um Regra para aplicação */
6    if
7    ::(msg_name == Msg) && (par_m_uid > atr_uid);              /* Escolha da Regra RuleMsg_1 */
8        run Ring_Entity_RuleMsg_1(atr_next, atr_uid, par_m_uid, ver_id);
9    ::(msg_name == Msg) && (par_m_uid == atr_uid);             /* Escolha da Regra RuleMsg_2 */
10       atr_leader = true;                                  /* Aplicação da Regra RuleMsg_2 */
11
12   ...                                          /* Escolha das Regras RuleMsg_3 e RuleStart */
13
14   ::else;
15     this_chan!msg_name, par_m_uid;                  /* Reenvio de mensagens não aplicáveis */
16   fi;
17 }
18
19 proctype Ring_Entity_RuleMsg_1(chan atr_next; byte atr_uid;
20                                byte par_m_uid; byte ver_id) {
21   atr_next!Msg, par_m_uid;                                /* Aplicação da Regra RuleMsg_1 */
22 }
23
24 ...                                         /* Aplicação das Regras RuleMsg_3 e RuleStart */
25
26 proctype Ring_Entity(chan this_init; chan this_chan) {       /* Tipo de objeto Ring_Entity */
27   mtype msg_name;                                  /* Contém o nome das mensagens recebidas */
28   byte ver_id = _pid;                           /* Identificação usada na verificação formal */
29   bool atr_leader; chan atr_next; byte atr_uid;                    /* Atributos do objeto */
30   byte par_m_uid;                                  /* Parâmetros das mensagens recebidas */
31   this_init?atr_leader, atr_next, atr_uid;      /* Inicialização dos atributos dos objetos */
32   do
33   ::this_chan?msg_name, par_m_uid;                              /* Recepção de mensagens */
34     rules_Ring_Entity();                           /* Seleção para aplicação de um Regra */
35   od;
36 }
37
38 init {                                                  /* Mapeamento do Grafo Inicial */
39   chan chan_Ring_Entity_1 = [SIZE] of {mtype, byte};          /* Canal do objeto Ring_Obj1 */
40   chan chan_Ring_Entity_1_init = [0] of {bool, chan, byte};    /* Canal para inicialização */
41
42   ...                     /* Criação de outros canais, pertencentes a Ring_Obj2 e Ring_Obj3 */
43
44   atomic {
45     run Ring_Entity(chan_Ring_Entity_1_init, chan_Ring_Entity_1);  /* Criação de Ring_Obj1 */
46     chan_Ring_Entity_1_init!false, chan_Ring_Entity_2, 1;    /* Inicialização de Ring_Obj1 */
47
48     ...                                 /* Criação e inicialização de Ring_Obj2 e Ring_Obj3 */
49
50     chan_Ring_Entity_1!Start, 0;                 /* Envio da mensagem Start para Ring_Obj1 */
51
52     ...                       /* Envio das outras mensagens Start para Ring_Obj2 e Ring_Obj3 */
53   }
54 }
```

Figura 3: Mapeamento do exemplo na GGBO definido anteriormente.

Neste mapeamento, a semântica de paralelismo implícito para a aplicação de Regras na GGBO não é completamente respeitado, sendo que as Regras que modificam o estado interno do objeto são aplicadas de forma serial (Linha 10). Já as Regras que não modificam o estado interno do objeto podem ser aplicadas de forma paralela, sendo estas criadas na forma de processos (Linhas 19 a 22 e 24).

O Grafo Inicial da GGBO é mapeado para um processo inicial em PROMELA (Linhas 38 a 54). Este processo tem como objetivo definir canais usados pelos objetos e canais para a inicialização dos atributos dos objetos (Linhas 39 a 42). Além disto, são criados os processos que correspondem aos objetos definidos no Grafo Inicial (Linhas 45 a 48) e enviadas as mensagens *Start* definidas no Grafo Inicial (Linhas 50 a 52).

6. Conclusão

Este artigo apresentou uma proposta de abordagem para a verificação formal, baseada na verificação de modelos, de SDBOs. A verificação é viabilizada a partir do mapeamento de um sistema descrito na linguagem de especificação formal GGBO (usada como linguagem de descrição) para um modelo na linguagem de descrição PROMELA.

A partir deste mapeamento o desenvolvedor tem de especificar as propriedades desejadas para o modelo e verificá-las usando o SPIN. Assim, o desenvolvedor inicia um processo de especificação $\Rightarrow$ verificação $\Rightarrow$ correção. Quando uma propriedade desejada é verificada e retorna o valor falso, o desenvolvedor, com base no contra-exemplo gerado, corrige o modelo para que a propriedade venha a ser verdadeira numa próxima verificação. No momento em que as propriedades desejadas são verdadeiras sobre o modelo, diz-se que o modelo é mais confiável. Isto se deve a garantia formal de que as propriedades desejadas são mantidas.

Devido a restrições de espaço não foi possível apresentar uma metodologia para a especificação e verificação de propriedades. Este tipo de metodologia existe no contexto do trabalho, porém, ainda exige que o usuário conheça o mapeamento de GGBO para PROMELA apresentado neste artigo. No entanto, como pode ser visto na Seção 5., as construções da GGBO puderam ser mapeadas de maneira bastante direta para as construções em PROMELA, o que facilita a compreensão da descrição mapeada.

Além disto, o mapeamento aqui proposto ainda não apresenta uma prova formal. Esta prova formal está sendo estudada e é vista como um importante trabalho futuro. Entretanto, pode-se observar que, apesar de não existir uma prova formal o mapeamento mantém (de acordo com a restrição de paralelismo evidenciada na Seção 5.) a semântica da GGBO.

Com relação a ferramentas computacionais, existe uma interface básica para a definição, armazenamento e recuperação de modelos na GGBO. A partir de descrições armazenadas é possível gerar: **i)** código para simulação, **ii)** código para execução em um ambiente real, ou **iii)** código para verificação formal, usando a abordagem proposta neste artigo.

Referências

Avizienis, A., Laprie, J.-C., and Randell, B. (2001). Fundamental concepts of dependability. Technical Report 01-145, LAAS (Laboratory for Analysis and Architecture of Systems), Taulosse, France.

Cho, S. M., Bae, D. H., Cha, S. D., Kim, Y. G., Yoo, B., and Kim, S. (1999). Applying Model Checking to Concurrent Object-oriented Software. In *4th International Symposium on Autonomous Decentralized Systems*, pages 380–383, Tokyo, Japan. IEEE Computer Society Press.

Clarke, E. M., Grumberg, O., and Peled, D. A. (1999). *Model Checking*. MIT Press, Cambridge, MA, USA.

Clarke, E. M., Wing, J. M., Alur, R., Cleaveland, R., and et al (1996). Formal Methods: State of the Art and Future Directions. *ACM Computing Surveys*, 28(4):626–643.

Copstein, B., da Costa Móra, M., and Ribeiro, L. (2000). An Environment for Formal Modeling and Simulation of Control Systems. In *33rd Annual Simulation Symposium*, pages 74–82, Washington, USA. IEEE Computer Society Press.

Dotti, F. L. and Ribeiro, L. (2000). Specification of Mobile Code Systems Using Graph Grammars. In *4th International Conference on Formal Methods for Open Object-Based Distributed Systems*, volume 177, pages 45–63, Stanford, CA, USA. Kluwer - IFIP Conference Proceedings.

Duarte, L. M. (2001). Desenvolvimento de Sistemas Distribuídos com Código Móvel a partir de Especificação Formal. Dissertação de mestrado, Pontifícia Universidade Católica do Rio Grande do Sul, Faculdade de Informática - Programa de Pós-Graduação em Ciência da Computação, Porto Alegre, RS, Brasil.

Ehrig, H. (1979). 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, pages 1–69, Berlin, Germany. Springer-Verlag - Lecture Notes in Computer Science.

Holzmann, G. J. (1997). The Model Checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295.

Leue, S. and Holzmann, G. (1999). v-Promela: a visual, object oriented language for SPIN. In *2nd International Symposium on Object-Oriented Real-Time Distributed Computing*, Saint-Malo, France. IEEE Computer Society Press.

Lilius, J. and Paltor, I. P. (1999). vUML: A Tool for Verifying UML Models. In *14th International Conference on Automated Software Engineering*, pages 255–258, Cocoa Beach, Florida, USA. IEEE Computer Society Press.

Lynch, N. A. (1996). *Distributed Algorithms*, pages 476–482. Morgan Kaufmann Publishers, San Francisco, CA, USA.

Manna, Z. and Pnueli, A. (1992). *The Temporal Logic of Reactive and Concurrent Systems - Specification*. Springer-Verlag, New York, NY, USA.

Rödel, E. T. (2003). Modelagem Formal de Falhas em Sistemas Distribuídos envolvendo Mobilidade. Dissertação de mestrado, Pontifícia Universidade Católica do Rio Grande do Sul, Faculdade de Informática - Programa de Pós-Graduação em Ciência da Computação, Porto Alegre, RS, Brasil.

Winter, K. and Duke, R. (2002). Model Checking Object-Z using ASM. In *3rd International Conference on Integrated Formal Methods*, volume 2335, pages 165–184. Lecture Notes in Computer Science, Spinger-Verlag.