Uso de Redes de Autômatos Estocásticos para Modelar Mecanismos Tolerantes a Falhas*

Luciano A. Cassol, Avelino F. Zorzo, Paulo Fernandes
{lcassol,zorzo,paulof}@inf.pucrs.br

[1]FACIN - PUCRS - Av. Ipiranga 6681 - 90619-900 - Porto Alegre - RS

Resumo. O uso de mecanismos de tolerância a falhas tem sido cada vez mais freqüente no desenvolvimento de sistemas críticos. Com este aumento, a precisão da descrição de tais mecanismos é fundamental para que seu uso não gere situações inesperadas aos desenvolvedores dos sistemas, devido a má interpretação destes mecanismos. Desta forma, este artigo apresenta a descrição de interações multi-participantes confiáveis (DMI) usando redes de autômatos estocásticos (SAN), um formalismo para descrição de sistemas distribuídos onde interações entre participantes pode ser modelado como um autômato estocástico.

1. Introdução

Programas paralelos são compostos de diferentes atividades concorrentes, padrões de comunicação e sincronização entre essas atividades. Dessa forma, a programação paralela é considerada difícil, pois somando-se as preocupações normais de programação, o programador tem que trabalhar com uma complexidade ainda maior gerada pelas diversas *threads* do sistema, controlando sua criação e destruição, e controlando suas interações através de sincronização e comunicação [1].

Somado a este contexto, a proliferação de sistemas distribuídos tem aumentado consideravelmente nos últimos anos. Apesar da distribuição tornar o sistema mais robusto, através da replicação de seus componentes, algumas vezes o aumento na distribuição pode introduzir falhas indesejáveis. Desta forma, é vital que a programação distribuída seja feita de maneira o rganizada.

Uma forma de organizar aplicações distribuídas é através do uso de operações que envolvam mais de um participante, como ações separadas que coordenam as interações entre esses participantes.

Um mecanismo que inclui diversos participantes (objetos ou processos) executando um conjunto de atividades em conjunto é chamado de Interações Multi-participantes (*Multiparty Interaction* - MI). Em uma MI, diversos participantes "reúnem-se" para produzir um estado combinado, intermediário e temporário. Usam este estado para executar algumas atividades e assim deixam a interação e continuam sua execução normal [2]. Neste artigo utilizamos um mecanismo para interações entre diversos participantes que provê recursos para tratamento de

*Este projeto tem apoio da CAPES, CNPq e FAPERGS.

exceções concorrentes e consistência de estado no término da interação. Este mecanismo é denominado Interações Multi-participantes Confiáveis (DMI - *Dependable Multiparty Interaction*) [3]. DMIs tem sido aplicadas na implementação de diversos sistemas críticos, porém a verificação de tais sistemas é na maioria das vezes realizada de maneira empírica.

Uma forma de verificar se as interações entre diversos participantes possuem as propriedades e comportamentos esperados é através da formalização das características e comportamentos que as interações entre diversos participantes têm. Para formalizar uma especificação de um sistema, é necessário uma linguagem de especificação formal.

Linguagens de especificação formal são utilizadas para fornecer uma descrição clara e precisa de sistemas, para testar e provar matematicamente que tais sistemas possuem as propriedades e comportamentos esperados, permitindo assim uma única interpretação de seu funcionamento. Esta interpretação única nem sempre é fornecida por descrições feitas em linguagem natural, onde diferentes interpretações podem gerar diferentes resultados. Para prover esta interpretação, as linguagens de especificação formal fornecem um modelo matemático para descrição de sistemas, a partir do qual podem ser realizadas verificações, a fim de provar que o sistema possui determinadas propriedades e comportamentos.

A descrição sintática e semântica das linguagens é essencial para auxiliar no desenvolvimento e programação de sistemas concorrentes com qualidade. A descrição sintática descreve a forma correta que os programas devem ser escritos, enquanto uma descrição semântica descreve o significado que é anexado as várias construções sintáticas.

A definição de sistemas complexos dificilmente pode ser abordada cientificamente sem que uma descrição formal seja empregada. No entanto, as qualidades de uma formalização de um sistema não se limitam a sua modelagem de forma inequívoca. A utilização de um formalismo torna possível a obtenção de conclusões sobre a viabilidade do sistema descrito através de índices de confiabilidade e desempenho [4, 5]. As observações sobre a viabilidade do sistema podem ser divididas em:

- informações que estimam confiabilidade; e
- informações que estimam o desempenho.

O formalismo de Redes de Autômatos Estocásticos (*Stochastic Automata Networks*) [6] é de grande interesse na modelagem de sistemas onde o paralelismo e sincronização são primitivas básicas de relacionamento. Esse formalismo baseia-se na descrição de sistemas complexos como um conjunto de entidades autônomas que interagem ocasionalmente.

Este artigo tem como objetivo descrever a utilização das DMIs através de Redes de Autômatos Estocásticos (SAN).

2. Redes de Autômatos Estocásticos (SAN)

Redes de autômatos estocásticos baseiam-se no formalismo matemático chamado cadeias de *Markov* [7]. Esse formalismo descreve o funcionamento do sistema como um conjunto de estados que se alteram segundo taxas definidas por leis exponenciais (autômatos estocásticos). Uma

vez modelado o sistema, a resolução do sistema consiste na solução de sistemas de equações lineares. O tamanho deste sistema é igual ao número de estados da Cadeia de Markov [7].

Redes de autômatos estocásticos são um formalismo para a modelagem de sistemas com grandes espaços de estados. Ao contrário das Cadeias de Markov que descrevem o sistema como um único autômato, as Redes de Autômatos Estocásticos baseiam-se na descrição de um sistema como um conjunto de subsistemas, em que cada subsistema é modelado como um autômato estocástico. A interação entre os subsistemas é descrita através das regras estabelecidas entre os estados internos de cada autômato. Dessa forma, o formalismo de redes de autômatos estocásticos é importante para a modelagem de sistemas distribuídos, nos quais as unidades de processamento não interagem entre si a todo instante [8].

Um autômato pode ser descrito como um conjunto de estados e um conjunto de transições entre esses estados. Essas transições são divididas em transições locais e transições sincronizadas. Um evento local é associado a uma única transição local e um evento sincronizante é associado a um conjunto de duas ou mais transições sincronizadas cada uma delas em um autômato.

O estado global de um sistema é definido como a combinação de todos os estados locais de cada autômato do sistema. Alterando o estado local de um autômato, o estado global da rede altera-se também. A mudança no estado global numa rede de autômatos estocásticos pode ser conseqüência da ocorrência de um evento local, ou de um evento sincronizante.

Um estudo mais aprofundado sobre SAN foi realizado em [9] e uma referência mais completa pode ser encontrada em [6, 8].

3. Estudo de Caso: formalização de DMI em SAN

De forma a modelar uma DMI em SAN, primeiramente devemos estabelecer um conjunto de regras a serem descritas de maneira precisa e inequívoca. Este conjunto de regras já foi previamente especificado em outro formalismo [10]. Além disto, um framework foi implementado a partir deste conjunto de regras e usado para o desenvolvimento de diversas aplicações críticas [2, 11, 12]. O conjunto de regras que estaremos mostrando neste artigo é o seguinte:

- a DMI é composta por um conjunto de papéis (*roles*) que são executados por participantes (*players*);
- os participantes ativam um papel para executar as instruções que compõem o papel;
- a DMI é iniciada somente quando todos os papéis da DMI tiverem sido ativados, e uma pré-condição (*guard*) estiver habilitada;
- a DMI somente é finalizada quando todos os participantes tiverem terminado de executar seus papéis, a pós-condição (*assertion*) for validada, e nenhuma exceção tiver sido levantada;
- exceções podem ser levantadas durante a execução da DMI. Se a exceção é levantada, então todos os papéis que não levantarem uma exceção são interrompidos, e um algoritmo de resolução é executado;
- se existe um tratador para a exceção que foi escolhida pelo algoritmo de resolução, então esse tratador é ativado por todos os papéis;

- se não existe um tratador para a exceção que foi escolhida pelo algoritmo de resolução, então a exceção é sinalizada[1] para quem invocou os papéis.

Para demonstrar como modelar sistemas que são implementados utilizando DMIs, adotamos, neste artigo, uma DMI utilizada na implementação de um sistema de controle para uma célula de produção [13]. Esta DMI será descrita em SAN.

A DMI escolhida foi a `CarregaGuindaste1` (veja Figura 1). Esta DMI é responsável por executar a leitura do código de barras existente em um bloco de metal que se encontra em uma esteira alimentadora, e posteriormente fazer com que um guindaste retire este bloco de metal da esteira. Para executar estas tarefas, a DMI `CarregaGuindaste1` é modelada com três papéis: `EsteiraAlimentadora`, `LeitorCodigoBarras` e `Guindaste1`. Estes papéis são ativados (executados) pelos controladores de cada um dos dispositivos correspondentes. Veja na figura as atividades executadas em cada um dos papéis. As setas pontilhadas entrando ou saindo da DMI representam os acessos aos objetos externos à DMI (estes objetos representam os dispositivos físicos na célula de produção).

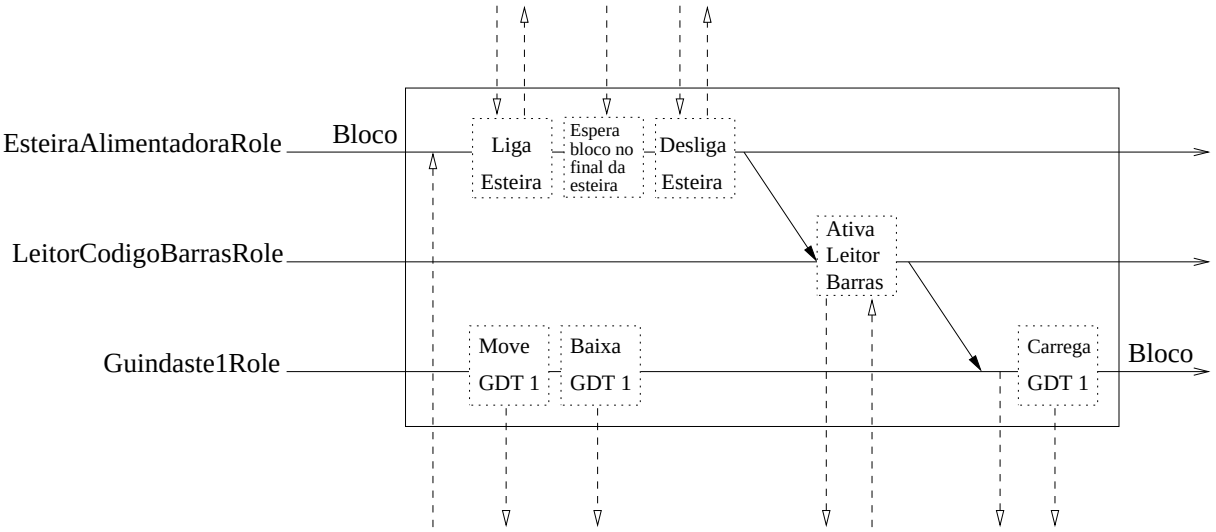

Figura 1: Exemplo da DMI CarregaGuindaste [13]

A Figura 2 representa parte do modelo gráfico da especificação da DMI `CarregaGuindaste1` em SAN. Nesta figura, um conjunto de autômatos é utilizado para representar uma `DMI`: cada um dos participantes (`PLAYER`$_i$), a pré-condição (`GUARD`), a pós-condição (`ASSERT`), o tratador de exceção (`HANDLER`), e o resultado e estado de cada um dos papéis (`ROLE`$_i$). A representação textual, para este modelo gráfico, que usa a ferramenta PEPS[2] [14] para avaliação de desempenho e confiabilidade, é apresentado na Figura 3.

O funcionamento da DMI, ou seja a troca de estados nos autômatos, durante a execução normal de uma DMI é feita conforme descrito a seguir. Um participante antes de querer executar

[1] O termo levantar exceção refere-se a uma exceção no contexto onde ela aconteceu, enquanto o termo sinalizar exceção refere-se a propagação da exceção para o nível superior do serviço que está sendo executado.

[2] PEPS -*Performance Evaluation of Parallel Systems, software* que resolve numericamente Cadeias de Markov com grande espaço de estados. O modelo em Redes de Autômatos Estocásticos é baseado em Cadeias de Markov.

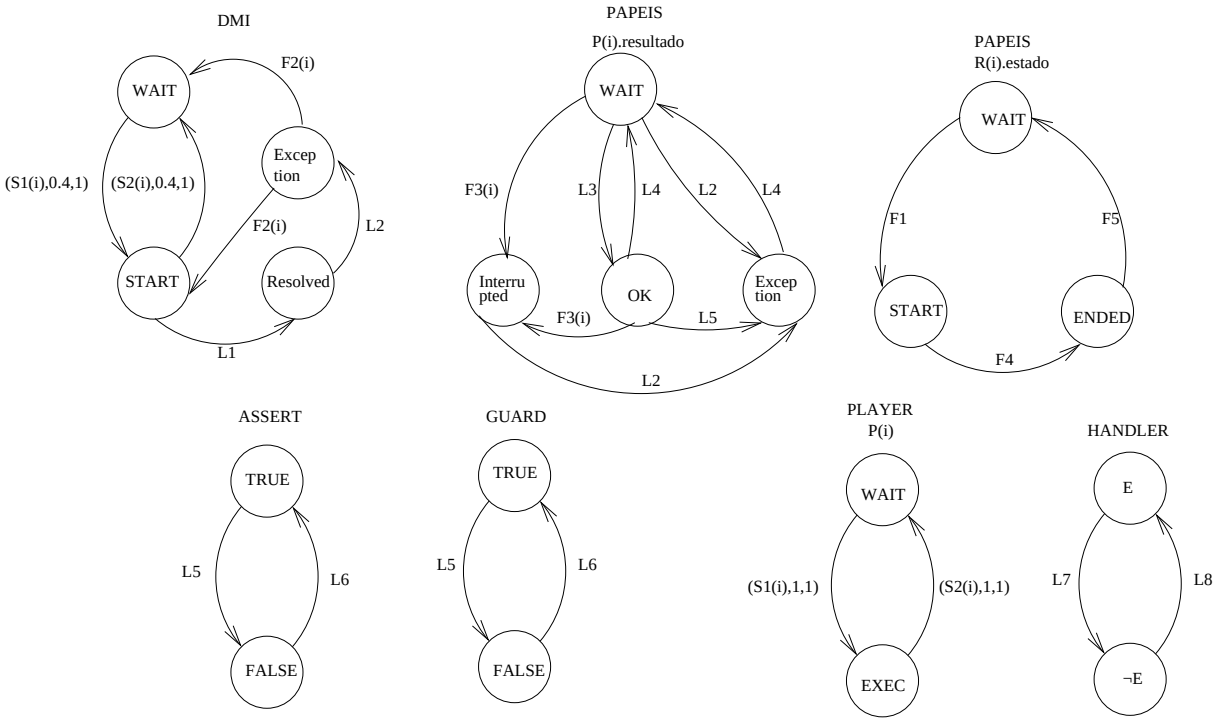

Figura 2: Modelo Gráfico da Especificação da DMI `CarregaGuindaste` em SAN

um papel em uma DMI, encontra-se no estado `WAIT`. A partir do momento que o mesmo deseja executar um papel ele passa do estado `WAIT` e habilita a transição sincronizada correspondente, ou seja S_i. Para que uma DMI esteja habilitada, ou seja passe do estado `WAIT` para o estado `START`, todos os participantes devem ter sincronizado. Os papéis, por outro lado, só serão executados, ou seja passarão do estado `WAIT` para o estado `START`, quando todos os participantes estiverem no estado `EXEC` e a pré-condição (autômato `GUARD`) estiver no estado `TRUE`. Estes testes são realizados pela transição funcional `F1`.

Quando um participante termina a execução de um papel de maneira correta, o autômato `PAPEIS` P_i`.resultado` correspondente passará para o estado `OK`. Quando todos autômatos `PAPEIS` P_i`.resultado` estiverem no estado `OK`, então a transição funcional `F4` será ativada e todos os papéis passarão do estado `START` para o estado `ENDED`. O término da execução de todos os papéis será efetuado quando a pós-condição (autômato `ASSERT`) estiver no estado `TRUE` e todos os papéis estiverem no estado `ENDED`. Este teste é realizado pela transição funcional `F5`.

Quando todos papéis estiverem novamente no estado `WAIT`, então os participantes podem executar suas transições sincronizadas $S2_i$, passando novamente para o estado `WAIT`, e também passando a DMI do seu estado `START` para o estado `WAIT`.

Quando um papel levantar uma exceção, então o autômato `PAPEIS` P_i`.resultado` passa ou do estado `WAIT` ou do estado `OK` para o estado `EXCEPTION`, fazendo que esta exceção seja tratada pela DMI. Caso uma exceção seja levantada por outro papel, então os demais papéis que estiverem ou no estado `WAIT` ou no estado `OK` passam para o estado `Interrupted`

```
declarations {}
  constant tx := 2;
  constant me := 2;
  constant es := 1;

functional F1:= (((st Player1 == EXEC) }
               && (st Player2 == EXEC) \&\& (st Player3 == EXEC)
               && (st Guard == TRUE)){*}2);}
functional F2:= ((st Handler == EXIST) {*}2);
functional F21:= ((st Handler == NOTEXIST) {*}2);
functional F31:=(((st RoleResult2 == EXCEPTION) || (st RoleResult3 == EXCEPTION)){*}2);
functional F32:=(((st RoleResult1 == EXCEPTION) || (st RoleResult3 == EXCEPTION)){*}2);
functional F33:=(((st RoleResult1 == EXCEPTION) || (st RoleResult2 == EXCEPTION)){*}2);
functional F4:=(((st RoleResult1 == OK) && (st RoleResult2 == OK) && (st RoleResult3 == OK)){*}2);
functional f5:=(((st RoleState1==ENDED) && (st RoleState2==ENDED) && (st RoleState3 == ENDED)
                && (st Assert == TRUE)){*}2);}
reachability := 1;

network dmi (continuous, 13){

automaton dmi1(1,4){
  node WAIT (1) { arc (START,3){sync(S11,1,0.4) sync(S12,1,0.3) sync(S13,1,0.3)}}
  node START (2) {arc (WAIT,3){sync(S21,1,0.4) sync(S22,1,0.3) sync(S23,1,0.3)}
  node PAPEIS P_i.resultado {arc (RESOLVED,3){sync(F31,1,0.4) sync(F32,1,0.3) sync(F33,1,0.3)}}
  node EXCEPTION (2){ arc (WAIT,0,F2) arc (START,0,F21)}
  node RESOLVED (1){ arc (EXCEPTION,0,1)}}

automaton Guard(1,2){
 node TRUE (1){ arc (FALSE,0,1)}
 node FALSE (1){ arc (TRUE,0,1)}}

automaton Assert(1,2){
 node TRUE (1){ arc (FALSE,0,1)}
 node FALSE (1){ arc (TRUE,1){ sync(F5,1,1)}}}

automaton Handler(1,2){
 node EXIST (1){ arc (NOTEXIST,0,1)}
 node NOTEXIST (1){ arc (EXIST,0,1)}}

//=============== Para cada automato i ======================

automaton Player_i(1,2){
 node WAIT (1){ arc (EXEC,1){sync(S1_i,me,1)}}
 node EXEC (1){ arc (WAIT,2){sync(S2_i,me,0.5) sync(F5,1,0.5)}}}

//=============== Para cada automato i ======================

automaton RoleState_i(1,3){
 node WAIT (1){ arc (START,1){ sync(F1,1,1)}}
 node START (1){ arc (ENDED,1){ sync(F4,1,1)}}
 node ENDED (1){ arc (WAIT,1){ sync(F5,1,1)}}}

//=============== Para cada automato i ======================

automaton RoleResult_i(1,4){
 node WAIT (3){ arc (INTERRUPTED,1){sync(F3_i,F3_i,1)} arc (OK,0,1) arc (EXCEPTION,0,1)}
 node INTERRUPTED (1){ arc (EXCEPTION,0)}
 node OK (3){ arc (WAIT,0,1) arc (INTERRUPTED,1){sync(F3_i,F3_i,1)} arc (EXCEPTION,0,1)}
 node EXCEPTION (1){ arc (WAIT,0,1)} }

//=============== RESULTS ======================
results { teste := st dmi1 == WAIT;}
```

Figura 3: Especificação textual da DMI CarregaGuindaste1 em SAN

para que a exceção levantada seja tratada por todos os papéis pertencentes a mesma DMI. As transições $\mathsf{L}_{(}i)$ representam transições locais dos autômatos, isto é, dependem apenas da sua taxa de disparo e não da transição de estado de outros autômatos.

Na Figura 2 é representado a possibilidade de uma ou mais exceções serem levantadas durante a execução de dos papéis pelos participantes.

Através dos resultados obtidos no PEPS observou-se que todos os estados da especificação são alcançáveis, pois obtivemos probabilidade de ocorrência desses estados maior que zero o que demonstra a alcançabilidade dos mesmos. A análise dos resultados baseou-se na verificação da alcançabilidade, isto é, verificou-se apenas se os estados são alcançáveis ou não, não levando em consideração os índices de desempenho que foram gerados.

4. Conclusão

Neste trabalho foi apresentada uma descrição geral do formalismo de Redes de Autômatos Estocásticos que foi utilizado para especificar formalmente uma DMI. Foi criado e demonstrado um modelo de especificação formal, e validado com o *software* PEPS [14].

A especificação formal de sistemas torna mais fácil o seu processo de desenvolvimento, pois dessa forma é possível corrigir erros de projeto antes da sua implementação, evitando assim desperdícios de recursos.

Um cuidado que deve-se tomar antes de especificar formalmente um sistema, é na escolha da linguagem de especificação formal, pois uma escolha errada pode ocasionar inconsistência na especificação.

Uma análise informal dos resultados obtidos nas especificações de DMI em TLA - *Temporal Logic of Actions* [10] e SAN (apresentada neste artigo), nos leva a concluir que em TLA a possibilidade de desenvolver especificações genéricas torna a modelagem de um determinado sistema mais abrangente pois por exemplo, num sistema multi-thread é possível modelar o sistema levando em consideração apenas as ações que devem ser executadas e não o número de threads que o sistema contém.

A modelagem de sistemas em SAN, por outro lado, deve ser feita de forma específica, isto é, num sistema multi-thread o número de threads tem de ser levado em consideração e não apenas o comportamento que cada thread tem. Em conpensação utilizando SAN para modelar sistemas é possível realizar simulações através de modelos matemáticos e gerar índices de desempenho e confiabilidade. Essas três características tornam SAN uma ferramenta muito robusta, pois além de validar o sistema é possível simular o modelo proposto.

Este modelo está correntemente sendo aplicado na modelagem de um sistema de controle de um sistema de manufatura controlado por computador existente na PUCRS.

Referências

[1] I Foster. Compositional parallel programming language. *ACM Transactions on Programming Languages and Systems*, 18(4):454–476, 1996.

[2] A. F. Zorzo and R. J. Stroud. A distributed object-oriented framework for dependable multiparty interactions. In *OOPSLA Conference Proceedings*, volume 34, pages 435–446, Denver, Colorado - USA, November 1999. The Association for Computing Machinery (ACM), Inc.

[3] A. F. Zorzo. *Multiparty Interactions in Dependable Distributed Systems.* PhD thesis, University of Newcastle Upon Tyne, UK, 1999.

[4] E. A de Souza e Silva and R. R Muntz. Métodos computacionais de solução de cadeias de markov: aplicações a a sistemas de computação e comunicação. Porto Alegre - Brasil, 1992. Instituto de Informática UFRGS.

[5] K. Trivedi. Probability e statistic with reliability, queueing and computer science aplications. *Englewwod Cliffs*, 1982.

[6] B. Plateau and K. Atif. Stochastic automata networks for modelling parallel systems. *IEEE Transactions on Software Engineering*, 17(10):1093–1108, 1991.

[7] W. J. Stewart. Introduction to numerical solutions of markov chains. *Princeton University Press*, 1994.

[8] P. Fernandes, B. Plateau, and W.J. Stewart. Eficient descriptor-vector multiplication in stochastic automata networks. *Journal of the ACM*, 45(3):381–414, 1998.

[9] L. A. Cassol. Especificação formal de interações multi-participantes confiáveis em lógica temporal de ações e redes de autômatos estocásticos. Trabalho individual I, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre - Brasil, 2001.

[10] A. F. Zorzo, A. Romanovsky, and B. Randell. *TLA Specification of a Mechanism for Concurrent Exception Handling.* Kluwer Publ., Holland, 2002.

[11] A. F. Zorzo. Dependable multiparty interactions: A case study. In *29th Conference on Technology of Object-Oriented Languages and Systems - TOOLS29-Europe*, pages 319–328, Nancy, France, 1999. IEEE Computer Society Press.

[12] J. Xu, B. Randell, A. Romanovsky, R. J. Stroud, A. F. Zorzo, E. Canver, and F. von Henke. Rigorous development of an embedded fault-tolerant system based on coordinated atomic actions. *IEEE Transactions on Computers*, 51(2):164–179, 2002.

[13] L. Cassol. Controle de célula de produção de tempo real com DMIs. In *III Workshop de Testes e Tolerância a Falhas*, Búzios, RJ, Brasil, Maio 2002.

[14] PEPS Project. PEPS project: Performance evaluation of parallel system, ID-INRIA/IMAG/PUCRS. WWW, 30 mar. 2002.