Geração de Testes Estruturais para aplicações Multithreads: Abordagem por Statecharts

Rogério Marinke[1,2], **Nandamudi L. Vijaykumar**[1], **Edson L. F. Senne**[1,3]

[1]Laboratório Associado de Computação e Matemática Aplicada – LAC
Instituto Nacional de Pesquisas Espaciais – INPE

[2]Faculdade de Tecnologia de Ourinhos – FATEC

[3]Universidade Estadual Paulista Júlio de Mesquita Filho – UNESP

{rogerio.marinke,vijay}@lac.inpe.br, elfsenne@feg.unesp.br

Resumo. *A modelagem do código fonte de software multithreads em Statecharts é proposta neste trabalho. Uma possível abordagem para a execução de Teste Baseado em Modelo (TBM) é realizar a modelagem em Máquina de Estados Finita (MEF). No entanto, alguns softwares, como protocolos de comunicação utilizados em aplicações espaciais possuem características de concorrência e paralelismo, as quais não são facilmente modeladas utilizando MEF. Após realizar a modelagem, uma ferramenta desenvolvida no INPE, chamada WEB-PerformCharts converterá a especificação obtida para uma MEF plana. Isto torna possível a implementação de critérios para derivar casos de testes para testes tipo caixa branca para sistemas concorrentes.*

1. Introdução

O processo de qualificação espacial é composto por diversos ambientes de teste que possuem como objetivo emular a maioria das condições presentes desde o pré-lançamento até o início da vida útil do espécime. Os diversos subsistemas do satélite são testados isoladamente e em conjunto com o objetivo de verificar o comportamento de todo o sistema. Os inúmeros softwares embarcados nestes sistemas devem de forma similar serem submetidos a rigorosos procedimentos de testes.

Os softwares utilizados nas missões espaciais da National Aeronautics and Space Administration (NASA), por exemplo, estão cada vez maiores em número de linhas e mais complexos conforme as novas exigências vão surgindo. A Tabela 1 exibe algumas missões espacias e os respectivos números de linhas dos softwares utilizados [Thompson et al. 2010].

Além da grande quantidade eminente de linhas, na maioria dos casos estes softwares possuem trechos de código *multithreads*. Neste contexto, muitos trabalhos tem sido realizados na tentativa de investigar e propor novas abordagens que explorem as características de sistemas concorrentes [Ezekiel et al. 2007, Curtis 2009].

Muitos pesquisadores têm aplicado Teste Baseado em Modelo (TBM) para derivar casos de teste e realizar diversas análises decorrentes da atividade de teste. A comunidade científica geralmente considera o TBM como um tipo de teste caixa preta, entretanto, segundo a classificação proposta por [Utting et al. 2006] o TBM pode ser considerado um

Tabela 1. Softwares utilizados em missões espaciais

Missão *Voyager*	3 mil linhas.
Missão *Cassini*	30 mil linhas.
Missão *Mars Path Finder*	160 mil linhas.
Space Shuttle	500 mil linhas.
Estação Espacial Internacional	3 milhões de linhas.
Constellation Project	50 milhões de linhas.

teste estrutural (caixa branca) quando critérios de cobertura são adaptados para trabalhar com modelos.

O objetivo deste trabalho é realizar a automatização da modelagem em *Statecharts* de códigos fonte de softwares *multithreads* escritos na linguagem de programação Java. O modelo proposto deriva casos de testes para sistemas concorrentes. Adicionalmente, serão realizadas as análises dos casos de testes obtidos e as funcionalidades oriundas da contribuição deste trabalho serão incorporadas nas ferramentas: Geração Automática de Casos de Teste Baseada em *Statecharts* (GTSC) e WEB-PerformCharts.

2. Teste baseado em modelo

Os critérios de geração de casos de teste baseados em MEF normalmente tratam o fluxo do controle de protocolos de comunicação e são baseados em técnicas de testes de transições de estados que possuem como objetivo detectar falhas de transição e de transferência[Bochmann and Petrenko 1994]. Dentre os critérios mais comuns para a geração de sequências de testes encontrados na literatura estão: Transtion Tour (TT), Unique Input/Output(UIO), Distinguishing Sequence (DS), *Switch-Cover* e critério *W* [Chow 1978, Sidhu and Leung 1989, Lee and Yannakakis 1996].

A técnica gráfica de *Statecharts* é uma extensão à MEF, porém, permite a representação da composição hierárquica de estados(profundidade), atividades paralelas(ortogonalidade), sincronismo e interdependência através de comunicação entre componentes *broadcast* [Harel 1987]. O trabalho proposto por [Amaral 2005] utiliza uma metodologia que permite especificar modelos *Statecharts* em XML. A linguagem denominada *PerformCharts Markup Language* (PcML) desenvolvida no trabalho é capaz de representar o *Statecharts* de um sistema reativo.

No trabalho realizado por [Santiago et al. 2008] é proposto o ambiente denominado GTSC, o qual permite ao projetista de teste modelar o comportamento de um software utilizando MEF ou *Statecharts* com o objetivo de gerar automaticamente sequências de casos de teste. [Arantes et al. 2008] apresenta em seu trabalho a ferramenta WEB-PerformCharts, a qual permite a execução através da Web. Neste trabalho o critério para geração de casos de testes chamado *Transition Tour* foi implementado e integrado a ferramenta.

2.1. Teste para sistemas concorrentes

No trabalho realizado por [Vergilio et al. 2005] os autores propõem o modelo *Parallel Control Flow Graph* (PCFG) para aplicar critérios de teste em softwares concorrentes com passagem de mensagem. Algumas das considerações realizadas pelos autores a respeito do PCFG são apresentadas a seguir e são utilizadas para a realização deste trabalho.

Tabela 2. Família de critérios baseados em fluxo de controle e comunicação.

All-nodes-r	requer que todos os nós $n_i^p \in N_r$, sejam exercitados ao menos uma vez.
All-nodes-s	requer que todos os nós $n_i^p \in N_s$, sejam exercitados ao menos uma vez.
All-nodes	requer que todos os nós $n_i^p \in N$ sejam exercitados.
All-edges-s	requer que todas as arestas $(n_i^f, n_j^g) \in E_s$ sejam exercitadas.
All-edges	requer que todas as arestas $(n_i^f, n_j^g) \in E$ sejam exercitadas.

- o modelo considera que o software concorrente *SC*= $\{p^0, p^1, ...p^{n-1}\}$ é um conjunto de *n* processos que executam concorrentemente;
- o número de processos *n* é fixo e conhecido em tempo de compilação e são criados apenas uma vez durante a inicialização do software concorrente; e
- a comunicação/sincronização dos processos é realizada por meio das arestas do Grafo de Fluxo de Controle (GFC) do PCFG que é construído para representar o software concorrente.

Considerando que no contexto de testes para softwares concorrentes é necessário testar a comunicação/sincronização existente no PCFG derivado, este trabalho utiliza uma família de critérios baseados em fluxo de controle e comunicação adaptados para gerar sequências de testes para softwares concorrentes, os quais foram propostos por [Souza et al. 2008] e são exibidos na Tabela 2.

3. Modelo proposto

A abordagem proposta para realizar os TBM utilizando uma metodologia caixa branca foi dividida nas etapas exibidas na Figura 1 e são detalhadas nas seções a seguir.

3.1. Leitura de código *multithread*

Consiste em realizar a leitura de todo o código fonte do software submetido a modelagem e à medida que esta leitura evolui pelo código é criado um PCFG em tempo de execução.

O modelo de derivação do PCFG proposto neste trabalho considera o conhecimento a priori da quantidade de arquivos do software *multithreads* submetido ao modelo proposto para leitura do código fonte. Esta simplificação é realizada para reduzir a complexidade do modelo ao tratar as questões de sincronização e concorrência oriundas do código fonte *multithread*.

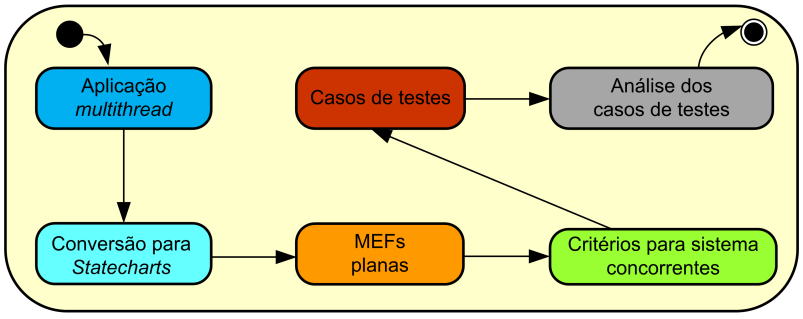

Figura 1. Abordagem proposta.

A estratégia para geração dos vértices a serem adicionados ao PCFG consiste em verificar para cada linha do código fonte o tipo de instrução (tomada de decisão, declaração e inicialização de variáveis e laços de repetição) contido. Entretanto, os vértices e arcos criados são conectados de acordo com o tipo de instrução detectado, que podem ser: sequenciais, tomada de decisão *if*, laço *while/for*, laço *until* e tomada de decisão *switch case*.

É importante destacar que, para a posterior geração automática dos casos de teste é necessário que exista um rótulo para as transições existentes entre os estados. Para realizar adequadamente a geração dos rótulos das transições detectadas foi implementado neste trabalho um algoritmo que fornece rótulos de entrada e saída. Antes de realizar a atribuição de rótulo na transição, o algoritmo verifica se o rótulo gerado não foi atribuído a nenhuma outra transição do grafo. Caso o rótulo gerado tenha sido atribuído à uma transição um novo rótulo é gerado.

Após realizar a leitura completa do código fonte os objetos Java, estados e transições, que representam o PCFG são mantidos na memória e inicia-se a próxima etapa.

3.2. Modelagem em *Statecharts*

Nesta etapa é realizada a leitura do PCFG passando por todos os vértices e arcos do grafo. A leitura tem como objetivo especificar o grafo em PcML. PcML é uma representação textual em XML de um diagrama *Statecharts*. Em PcML é possível especificar todos os componentes contido num diagrama *Statecharts*, como estados, transições, condições, eventos de entrada e saída e super estados. Após a execução dos passos descritos abaixo é obtido o modelo em *Statecharts* do software especificado em PcML.

- é criado um arquivo que irá armazenar o código PcML. Neste arquivo adiciona-se o código de cabeçalho PcML;
- percorrer cada PCFG contido na lista de grafos detectados conforme explicado na seção anterior. Cada PCFG representa um super estado no arquivo PcML;
- utilizando o algoritmo busca em profundidade são percorridos todos os estados do grafo e para cada estado é gerado uma linha de código no arquivo PcML, a qual é adicionada no super estado correspondente;
- percorre-se a lista de transição de cada estado. Para cada transição é gerado o código PcML correspondente. Cada transição contém os seguintes elementos: estado origem, estado destino e os rótulos de entrada e saída;
- caso exista mais de um PCFG na lista de grafos são adicionadas as condições para as transições; e
- são adicionadas os códigos de fechamento ao arquivo PcML.

3.3. Obtendo MEFs planas

Considerando o contexto e as ferramentas de TBM desenvolvidos pelo grupo de pesquisa de testes de software do INPE é necessário que a especificação PcML dos *Statecharts*, que eventualmente é composta por processos paralelos e outras características de softwares concorrentes, seja convertida em MEFs planas. Esta conversão é realizada neste trabalho devido ao fato de que a maioria dos critérios estruturais para geração de sequencias de teste são adaptados para MEF planas e alguns destes critérios estão implementados nas ferramentas GTSC e WEB-PerformCharts desenvolvidas no INPE.

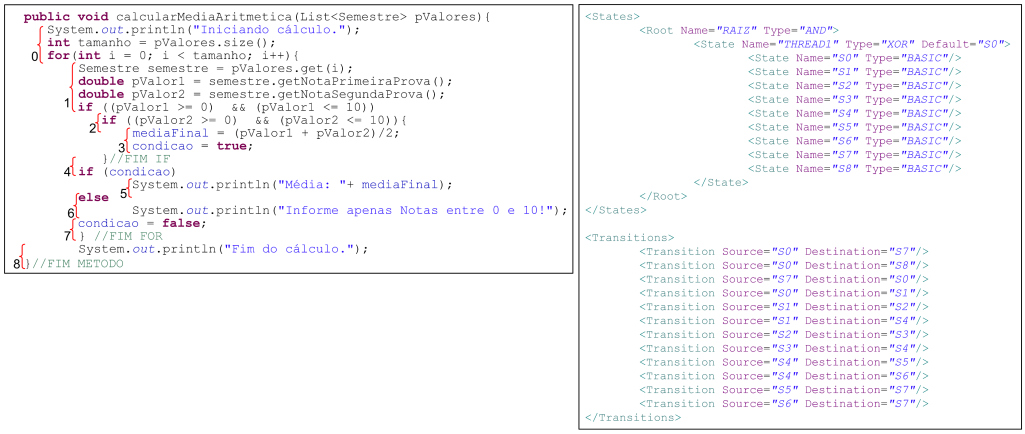

Figura 2. Código fonte e respectivo *Statecharts* especificado em PcML.

A realização desta etapa consiste em utilizar o arquivo PcML obtido na etapa anterior como entrada para a ferramenta WEB-PerformCharts, a qual fornecerá como saída um arquivo no formato XML que representa a MEF plana.

3.4. Derivando os casos de testes

Em resumo, numa abordagem TBM para gerar casos de testes são realizadas as seguintes etapas:(a)utilizando o algoritmo busca em largura adaptado para atender as restrições do critério a MEF plana derivada é percorrida;(b)com base nos estados e transições do software e considerando as restrições do critério são derivados os casos de teste.

3.5. Análise dos casos de teste obtidos

Uma análise dos casos de testes gerados pode, por exemplo, possibilitar que seja verificada a eficácia dos critérios dentro de determinados contextos do software submetido aos testes. Esta análise preliminar de cobertura é realizada nas seguintes perspectivas:

- são comparados a quantidade de estados e transições percorridas ao se utilizar os critérios: *All-nodes-r*, *All-nodes-s*; e
- as quantidades de estados e transições percorridas são comparadas ao total de estados e transições existentes na máquina plana derivada.

4. Resultados

Esta seção exibe os resultados preliminares obtidos até o momento. O código fonte submetido ao modelo e o arquivo *Statecharts* correspondente especificado em PcML obtido a partir do grafo derivado são mostrados na Figura 2.

5. Considerações finais

O modelo proposto neste trabalho possibilita a correta especificação de softwares *multithreads* em *Statecharts*. O uso da ferramenta WEB-PerformCharts e a implementação dos critérios de testes para sistemas concorrentes contribui para que a equipe de testes do INPE, por exemplo, possa ter uma alternativa ao realizar os TBM, visto que, atualmente estes testes são realizados utilizando a metodologia caixa preta.

Como trabalhos futuros sugere-se a integração dos critérios de teste para sistemas concorrentes implementados às ferramentas GTSC e WEB-PerformCharts, a implementação de outros critérios e o aprofundamento das análises dos resultados.

Referências

Amaral, A. S. M. S. d. (2005). *Geração de casos de teste para sistemas especificados em Statecharts*. PhD thesis, Instituto Nacional de Pesquisas Espaciais, São José dos Campos.

Arantes, A. O., Vijaykumar, N. L., de Santiago Junior, V. A., and Guimar aes, D. (2008). Web-performcharts: a collaborative web-based tool for test case generation from statecharts. In *iiWAS*, pages 374–381, New York, NY, USA. ACM.

Bochmann, G. and Petrenko, A. (1994). Protocol testing: Review of methods and relevance for software testing. *Proc. Int l Symposium on Software Testing and Analysis*, pages 109–124.

Chow, T. (1978). Testing software desing modeled by finite-state machines. *IEEE Transaction on Software Engineering*, 4(3):178–187.

Curtis, S. A. (2009). Evolvable neural software system. Technical report, NASA Goddard Space Flight Center.

Ezekiel, J., Lüttgen, G., and Siminiceanu, R. I. (2007). A parallel saturation algorithm on shared memory architectures. Technical report, NASA Langley Research Center.

Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Sci. Comput. Program.*, 8(3):231–274.

Lee, D. and Yannakakis, M. (1996). Principles and methods of testing finite state machines, a survey. In *Proceedings of the IEEE*.

Santiago, V., Vijaykumar, N. L., Guimaraes, D., Amaral, A. S., and Ferreira, E. (2008). An environment for automated test case generation from statechart-based and finite state machine-based behavioral models. In *ICST*, pages 63–72.

Sidhu, D. P. and Leung, T.-k. (1989). Formal methods for protocol testing: A detailed study. *IEEE Trans. Softw. Eng.*, 15:413–426.

Souza, S. R. S., Vergilio, S. R., Souza, P. S. L., Simão, A. S., and Hausen, A. C. (2008). Structural testing criteria for message-passing parallel programs. *Concurr. Comput. : Pract. Exper.*, 20:1893–1916.

Thompson, S., Brat, G., and Venet, A. (2010). Software model checking of arinc-653 flight code with mcp. In Muñoz, C., editor, *Proceedings of the Second NASA Formal Methods Symposium*, pages 171–181, Langley Research Center, Hampton VA 23681-2199, USA. NASA.

Utting, M., Pretschner, A., and Legeard, B. (2006). A taxonomy of model-based testing. *Working Paper Series*.

Vergilio, S. R., Souza, S. R. S., and de Souza, P. S. L. (2005). Coverage testing criteria for message-passing parallel programs. *LATW2005-6th IEEE Latin-American Test Workshop*, 1(2):161–166.