X WORCAP
20 e 21 de outubro
Instituto Nacional de Pesquisas Espaciais - INPE
São José dos Campos - SP

Geração de Testes Caixa Branca 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

Abstract. *A computational model capable to perform the modeling of the source code of a multithreads software in Statecharts is proposed in this paper. One possible approach to implement software testing based on computer model is to perform the modeling in Finite State Machine (FSM). However, some software, such as communication protocols used in space applications have characteristics of concurrency and parallelism, and can't be easily modeled using FSM. An alternative is to perform modeling with Statecharts. The objective this work is to explore the white box testing using Statecharts. The case study is composed of multithreads processes automatically converted to Statecharts. Then, the tool WEB-PerformCharts converts the specification in a FSM flat. This makes possible the methods implementation to generate test cases for white box testing type and concurrent systems.*

Resumo. *A modelagem do código fonte de software multithreads em Statecharts é proposta neste trabalho. Uma possível abordagem para a execução de testes de softwares baseados em modelo é realizar a modelagem em Máquinas de Estados Finita (MEF). No entanto, alguns softwares, como protocolos de comunicação utilizados em aplicações espaciais possuem características de concorrência e paralelismo, e estas não podem ser facilmente modeladas utilizando MEF. Uma alternativa é realizar a modelagem com Statecharts. O objetivo deste trabalho é explorar testes caixa branca utilizando Statecharts. O estudo de caso é composto por processos multithreads automaticamente convertidos para Statecharts. Então, a ferramenta WEB-PerformCharts converterá a especificação para uma MEF plana. Isto torna possível a implementação de critérios que derivam casos de testes para testes tipo caixa branca e para sistemas concorrentes.*

Palavras-chave: *Statecharts, teste caixa branca, threads.*

1. Introdução

A garantia de qualidade de um software é composta por várias atividades, dentre elas está a atividade de teste [Pressman 2006]. As exigências por softwares com maior qualidade motiva pesquisadores a desenvolverem e aperfeiçoarem métodos e técnicas de desenvolvimento a fim de atender aos requisitos de qualidade e, também, elaborar estratégias com grande capacidade de revelar erros.

O desenvolvimento da tecnologia para computadores pessoais tem propiciado que softwares estejam presentes na maior parte das atividades humanas, e, em muitos casos, softwares são empregados em tarefas consideradas críticas, ou seja, envolvem vidas humanas e possivelmente altos valores financeiros. Como exemplos é possível citar:

- os profissionais da área médica têm a disposição softwares que auxiliam na emissão de diagnósticos a pacientes;
- Engenheiros químicos utilizam softwares para conduzirem experimentos científicos; e
- Na engenharia aeroespacial, softwares são empregados para simular em computador condições de vibração, comportamento em voo, impacto e controle.

Consta na literatura que erros de softwares já provocaram prejuízos de toda magnitude, e muitos dos acidentes provocados os prejuízos causados são conhecidos, como é o caso do acidente com o foguete Ariane 5, ocorrido devido a um erro de conversão de ponto flutuante de 64 *bits* para inteiro de 16 *bits* [Dowson 1997], que gerou um problema de imprecisão numérica. Neste contexto, a atividade de teste é considerada atualmente indispensável no processo de desenvolvimento de aplicações, visto que, ela atesta a qualidade de um software desenvolvido. Adicionalmente, a medida que os erros revelados pelos testes são corrigidos a confiabilidade é aprimorada e os eventuais prejuízos minimizados.

A atividade de teste consiste em executar um software com a intenção de encontrar erros [Pressman 2006]. Os erros encontrados podem ter origem, entre outras causas, na implementação incorreta de uma funcionalidade anteriormente especificada, ou por exemplo, ter como causa uma entrada não prevista durante a fase de modelagem e especificação que o software não é capaz de processar adequadamente. Após a realização dos testes um software pode apresentar um comportamento adequado, entretanto, segundo [Beizer 1995] não é possível garantir que ele esteja livre de erros, uma vez que o conjunto de casos de testes tenham sido insatisfatórios. Adicionalmente, o número de entradas ou percursos possíveis num software, mesmo muito simples, é tão diverso que torna o teste exaustivo uma tarefa impraticável, por isso a necessidade de se ter uma boa variedade de casos de testes [Myers 1979]. Um exemplo simples, é um software que realiza o cálculo da raiz quadrada de um valor informado pelo usuário, no qual, os valores que podem ser utilizados como entrada inviabilizam o teste exaustivo.

O objetivo deste trabalho é propor a realização da modelagem em *Statecharts* de códigos fonte de softwares *multithreads* escritos na linguagem de programação Java. A partir do modelo obtido será possível realizar a implementação de critérios para sistemas concorrentes e testes caixa branca para derivar os casos de testes. As funcionalidades oriundas da contribuição deste trabalho serão incorporadas nas ferramentas: Geração Automática de Casos de Teste Baseada em Statecharts (GTSC) e WEB-PerformCharts.

1.1. Classificação dos Testes

Segundo [Delamaro et al. 2007] os métodos de seleção de casos de teste podem ser classificados em:

- Especificação (Caixa preta);
- Falhas; e
- Implementação (Caixa branca).

Os testes baseados na especificação, ou testes funcionais, têm por objetivo determinar se os aspectos requeridos foram implementados. Nessa técnica os detalhes de implementação não são considerados e o software é avaliado segundo o ponto de vista do usuário, ou seja, o foco de todas as atividades está somente na funcionalidade e não no que acontece internamente no software. Desta forma, o objetivo é garantir que o mesmo está trabalhando de acordo com os requisitos e que está em conformidade com as expectativas do usuário.

Para selecionar um conjunto de dados de entrada para posterior execução dos testes, a técnica de teste funcional utiliza alguns critérios, entre eles estão a Análise do Valor Limite, Grafo Causa-efeito e Particionamento de Equivalência[Delamaro et al. 2007]. Outros tipos de critérios, como *Syntax Testing* podem ser encontrados em [Kit and Finzi 1995, Vergilio et al. 2001].

Os testes baseados em falhas utilizam critérios que fornecem dados para compor uma medida da eficiência dos casos de testes. Segundo [Zhu et al. 1997] há uma série de formas de se aplicar estes testes e estabelecer critérios, entre elas estão: Introdução de erros, teste de mutação, teste de perturbação etc.

Nos testes baseados em implementação, também chamados de testes caixa branca ou estrutural, a seleção dos testes é baseada na informação obtida a partir do código fonte do software e visa mostrar que as diversas partes do código devem ser exercitadas sem violar a especificação [Myers 1979]. Critérios baseados no fluxo de controle, fluxo de dados e na complexidade, são utilizados pela técnica de teste caixa branca e um estudo detalhado pode ser encontrado em [Pressman 2006, Weyuker 1984, Maldonado 91].

1.2. Teste baseado em modelos

Testadores de software utilizam modelos para entender o sistema para o qual é necessário realizar toda a atividade de teste. A criação de modelos é uma tarefa comum entre os testadores, visto que, eles definem seus modelos, mesmo que usualmente não seja de uma maneira tão formal e rigorosa como uma especificação, porém, isto é realizado e é imprescindível para que seja possível determinar um comportamento inadequado. Conforme [Binder 1995], todo modelo possui quatro elementos:

- Assunto - é a ideia principal sobre a qual o modelo foi criado. Nas atividades de testes, o modelo do sistema auxilia na seleção de casos de testes;
- Ponto de vista - deve estabelecer a informação necessária para produzir e avaliar casos de testes;
- Representação - uma técnica de modelagem deve ter formas para expressar o comportamento de um sistema; e
- Técnica - modelos podem ser desenvolvidos usando qualquer técnica formal, assim como *Unified Modeling Language* (UML), ou utilizar simplesmente um artefato de uso geral.

Em [Usman et al. 2008] é realizado um estudo sobre as técnicas de verificação e consistência para modelos de aplicações orientado ao objeto utilizando a notação UML. No trabalho, as técnicas de controle são analisadas e alguns parâmetros de análise e construção tem sua coerência avaliada utilizando a estratégia de monitoramento. A pesquisa conclui que os diagramas de classe, sequência e *Statecharts* estão presentes na maioria das técnicas de verificação de consistência existente e que a maioria das técnicas de verificação analisadas são capazes de validar consistências em nível intra e inter modelos, ou seja, é possível atestar a exatidão de um modelo verificando a transição entre os estados do modelo, bem como a troca de mensagens com outros modelos.

Segundo [Harel 1987] um estado captura uma situação ou um contexto de um objeto, ou seja, descreve a atividade realizada ou a espera de um evento. Transição é um relacionamento entre dois estados que é disparado por algum evento que acarreta na execução de uma ação que leva a um estado específico. Uma MEF é composta por estados vinculados por transições.

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. Estes geralmente se utilizam de algum algoritmo para realizar percursos em grafos. O funcionamento e propósito destes critérios, numa visão alto nível, podem ser explicados como: a partir de uma MEF é criada uma árvore de transição de estados. Nesta árvore, é aplicado um critério que irá utilizar as entradas dos estados para a derivação dos casos de testes. Neste caso, os casos de testes são sequências de fluxo extraídas do modelo e podem ser utilizados, por exemplo, para validar se a implementação corresponde aos fluxos previstos no modelo. Dentre os critérios mais comuns para a geração de sequências de testes encontrados na literatura estão: *Transition Tour* (TT), *Unique-Input-Output sequence* (UIO), *Distinguishing Sequence* (DS), *Switch-Cover* e *Round-Trip Path* [Lee and Yannakakis 1996, Myers 1979, Binder 1995]. Segundo [Binder 1995], estes critérios permitem que sejam detectadas transições, respostas e estados incorretos ou não implementados.

Utilizando a abordagem de Teste Baseado em Modelo (TBM), o teste pode ser sistemático, focado e automatizado. Neste tipo de teste, a combinação entre entradas e estados pode ser enumerada. Com o objetivo de melhor entender o sistema e consequentemente aplicar os TBM, a equipe de teste pode criar seus próprios modelos, ou ainda, criar um novo modelo para substituir um modelo existente. Como por exemplo, com o intuito de especificar um sistema complexo que contenha concorrência e paralelismo, anteriormente especificado em MEF, um novo modelo em *Statecharts* pode ser criado.

1.3. Statecharts

Alguns softwares, como os utilizados em sistemas críticos de tempo real, protocolos de comunicação e sistemas embarcados exigem formas eficazes de verificação da qualidade. Estes tipos de aplicações podem ser classificadas como sistemas reativos, os quais, caracterizam-se por interagirem fortemente com o ambiente em que estão inseridos e uma mesma entrada pode provocar reações distintas dependendo do contexto ou estado na qual ocorre. Sistemas reativos podem ser representados através de MEF, onde a característica principal é a possibilidade de se definir a ordem temporal das interações.

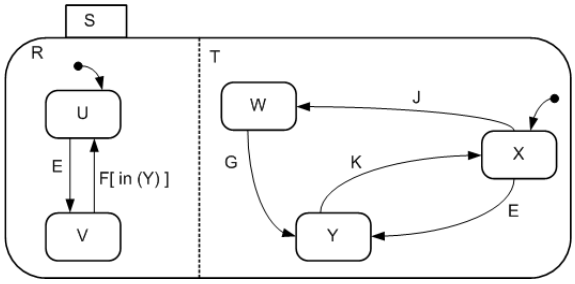

Figura 1. *Statecharts* - Comunicação entre estados [Harel and Politi 1998].

Diagrama de Transição de Estado (DTE) ou MEF são técnicas capazes de descrever o comportamento de um sistema, mostrar os estados possíveis que um objeto pode estar e como o estado do objeto muda ao receber eventos. No entanto, não permitem hierarquia, modularidade, profundidade e concorrência, ou seja, são planas. Desta forma, à medida que a complexidade do sistema cresce linearmente, o número de estados e transições cresce exponencialmente, gerando diagramas grandes e confusos. Adicionalmente, devido a dificuldade de representação de algumas características de sistemas complexos em MEF, foram propostas algumas técnicas para aprimoramento, como Stelle [Budkowski and Dembinski 1987] e *Statecharts* [Harel 1987]. A técnica gráfica de *Statecharts* é uma extensão a MEF, porém, permite a representação da composição hierárquica de estados, atividades paralelas, noções de profundidade, sincronismo e interdependência através de comunicação do tipo *broadcasting*. A Figura 1 mostra um estado *S* composto por dois outros estados *R* e *T* paralelos. O estado *R* contém os estados *U* e *V* e estes possuem as transições *E* e *F*. O estado *T* é composto pelos estados *W*, *X* e *Y* os quais são vinculados através das transições *E*, *G*, *J* e *K*. Os estados iniciais definidos em *R* e *T* são *U* e *X* respectivamente.

Caso ocorra a entrada E em (U,X), o sistema simultaneamente transfere para (V,Y), ou seja, um simples evento dispara duas ações simultâneas, caracterizando uma concorrência sincronizada.

O trabalho proposto por [Amaral 2005] utiliza uma metodologia que permite especificar modelos *Statecharts* em XML(Extensible Markup Language). A linguagem desenvolvida no trabalho, PerformCharts Markup Language (PcML) representa o Statecharts. A especificação XML produzida é convertida para uma MEF plana para que seja possível aplicar critérios para derivar casos de testes.

Com o objetivo de gerar automaticamente casos de testes a partir da especificação PcML, algumas abordagens foram propostas, como por exemplo, as ferramentas GTSC[Santiago 2008] e Web-PerformCharts[Arantes et al. 2008]. Estas ferramentas convertem o modelo *Statecharts* numa MEF plana e permitem que a partir deste modelo sejam aplicados critérios para obtenção dos casos de testes. Estas ferramentas são utilizadas pelo grupo de pesquisa de testes de software do INPE e sofrem atualizações a medida que as pesquisas evoluem.

Em [Souza et al. 2000] é proposto um conjunto de critérios de testes de cobertura para especificações baseadas em *Statecharts*. Os critérios propostos utilizam uma árvore de alcançabilidade e tem como objetivo complementar as abordagens de simulação e análise de propriedades utilizadas para validação e teste de especificações *Statecharts*.

E alguns destes critérios foram implementados e integrados à ferramenta GTSC.

2. Teste para Sistemas Concorrentes

A extração de MEF a partir do código fonte de uma aplicação é proposto em [Corbett et al. 2000]. Segundo os autores a ferramenta desenvolvida, **Bandera**, recebe o código de um software e deriva o modelo em MEF. No trabalho também é abordado como os resultados podem ser utilizados para a verificação e correção do modelo de um software Java.

[Luo et al. 2006] apresentam uma abordagem para TBM de sistemas concorrentes especificados em *Statecharts*. O trabalho descreve um processo automatizado para geração de sequências de teste.

O trabalho realizado por [Vergilio et al. 2005] apresenta uma família de critérios de teste para softwares concorrentes que utilizam o paradigma de passagem de mensagem. O modelo de teste proposto considera aspectos de sincronização e comunicação que são independentes da linguagem de programação.

[Sarmanho 2010] propõe um modelo de teste para softwares concorrentes que utilizam memória compartilhada. O trabalho trata a sincronização e a comunicação de *threads* (ou processo leve) de forma independente e entre outras características apresenta um método baseado em *timestamps* (ou sequência de caracteres para rotular data e/ou hora) para determinar as comunicações exercitadas numa dada execução do software.

3. Metodologia

Para a realização deste trabalho foi realizado inicialmente um levantamento bibliográfico sobre sistemas críticos e suas características. Adicionalmente, a importância de se realizar testes de softwares em sistemas espaciais também foi investigada.

3.1. Abordagem para realizar TBM para testes caixa branca

A proposta para realizar os TBM utilizando uma abordagem caixa branca foi dividida neste trabalho em algumas etapas, a saber:

- modelar em *Statecharts* softwares *multithread*;
- implementar um critério, passar por todos os nós ou vértices, para derivar casos de testes baseados na metodologia caixa branca;
- converter os *Statecharts* especificados em PcML em MEF utilizando a ferramenta WEB-PerformCharts;
- no trabalho realizado por [Sarmanho 2010] foi proposta uma abordagem para testar e analisar os resultados de testes para sistemas concorrentes com memória compartilhada. Esta abordagem será em parte utilizada neste trabalho, entretanto, os casos de testes a serem utilizados serão derivados através dos critérios propostos em [Vergilio et al. 2005]; e
- os critérios implementados serão modificados conforme a arquitetura dos ambientes de testes GTSC e WEB-PerformCharts e finalmente integrados aos mesmos.

4. Conclusão

Utilizar modelos para a realização de testes de software *multithreads* pode tornar possível a obtenção de um maior rigor das expectativas que o sistema deve satisfazer, sendo que a ambiguidade (considerando a hipótese de existirem estados com comportamento idêntico), por exemplo, é reduzida a medida que é utilizada uma técnica formal, como *Statecharts*, para representar corretamente os comportamentos do sistema.

A proposta exposta neste trabalho de realizar a modelagem em *Statecharts* do código fonte de um software, possibilita a correta especificação de softwares que contenham as características de concorrência e paralelismo. Adicionalmente, a abordagem caixa branca utilizada 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.

Finalmente, a pesquisa pretende, com a implementação dos critérios de teste para sistemas concorrentes derivar os casos de testes e contribuir para o aprimoramento das ferramentas GTSC e WEB-PerformCharts.

Referências

Amaral, A. S. M. S. d. (2005). Geração de casos de teste para sistemas especificados em statecharts. Master's 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 '08: Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services*, pages 374–381, New York, NY, USA. ACM.

Beizer, B. (1995). *Black-Box Testing: Techniques for functional testing of software and systems*. John Wiley and Sons.

Binder, R. (1995). *Testing Object-Oriented Systems. Models, Patterns, and Tools*. Addison-Wesley.

Budkowski, S. and Dembinski, P. (1987). An introduction to estelle: a specification language for distributed systems. *Comput. Netw. ISDN Syst.*, 14(1):3–23.

Corbett, J. C., Dwyer, M. B., Hatcliff, J., Laubach, S., Pasareanu, C. S., and Zheng, H. (2000). Bandera: Extracting finite-state models from java source code. In *In Proceedings of the 22nd International Conference on Software Engineering*, pages 439–448. ACM Press.

Delamaro, M. E., Maldonado, J. C., and Jino, M. (2007). *Introdução ao teste de software*. Elsevier.

Dowson, M. (1997). The ariane 5 software failure. *SIGSOFT Softw. Eng. Notes*, 22(2):84.

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

Harel, D. and Politi, M. (1998). *Modeling Reactive Systems with Statecharts: The Statemate Approach*. McGraw-Hill, Inc., New York, NY, USA.

Kit, E. and Finzi, S. (1995). *Software testing in the real world: improving the process*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA.

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

Luo, G., Bochmann, G. V., and Petrenko, R. (2006). Test selection based on communicating nondeterministic finite state machines using a generalized wp-method.

Maldonado, J. C. (91). *Critérios Potenciais Usos:Uma Contribuição ao Teste Estrutural de Software.* PhD thesis, DCA/FEEC/UNICAMP.

Myers, G. J. (1979). *The Art of Software Testing*. John Wiley and Sons, New York.

Pressman, R. S. (2006). *Engenharia de Software*. McGraw-Hill, Inc., Rio de Janeiro.

Santiago, V.; Vijaykumar, N. G. D. A. A. F. E. (2008). An environment for automated test case generation from statechart-based and finite state machine-based behavioral models. In *Fisrt IEEE International Conference on Software Testing Verification and Validation (ICST 2008)*, pages 63–72.

Sarmanho, F. S. (2010). Teste de programas concorrentes com memória compartilhada. Master's thesis, ICMC/USP.

Souza, S. R. S., Fabbri, S. C. P. F., Maldonado, J. C., and Masiero, P. C. (2000). Statecharts specifications: A family of coverage testing criteria. *CLEI 2000 - XXVI Conferência Latinoamericana de Informática*.

Usman, M., Nadeem, A., Kim, T.-h., and Cho, E.-s. (2008). A survey of consistency checking techniques for uml models. In *ASEA '08: Proceedings of the 2008 Advanced Software Engineering and Its Applications*, pages 57–62, Washington, DC, USA. IEEE Computer Society.

Vergilio, S. R., Maldonado, J. C., and Jino, M. (2001). Constraint based criteria: An approach for test case selection in the structural testing. *J. Electron. Test.*, 17(2):175–183.

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.

Weyuker, E. J. (1984). The complexity of data flow criteria for test data selection. *Inf. Process. Lett.*, 19(2):103–109.

Zhu, H., Hall, P. A. V., and May, J. H. R. (1997). Software unit test coverage and adequacy. *ACM Comput. Surv.*, 29(4):366–427.