Testes de Software de Aplicações Espaciais: Abordagem por Modelos Formais e Oráculos

Alessandro O. Arantes[1,2], Nandamudi L. Vijaykumar[1], Valdivino A. de Santiago Júnior[1]

[1]Programa de Doutorado em Computação Aplicada – CAP
Instituto Nacional de Pesquisas Espaciais – INPE

[2]Instituto de Estudos Avançados – IEAv
Departamento de Ciência e Tecnologia Aeroespacial – DCTA

aarantes@ieav.cta.br, vijay@lac.inpe.br, valdivino@das.inpe.br

Abstract. *INPE develops embedded systems for scientific satellites and stratospheric balloons, hence the verification and validation processes require special attention in detecting and preventing failures. In this scenario, techniques such as formal modeling based on requirements assists in the development process, allowing a systematic and automated generation of test cases for such systems even before its final implementation. The automatic generation and execution of test cases provides a significant gain in productivity for software testers, and combined with a test oracle, enables evaluating reactive system's behavior in face of its inputs.*

Resumo. *O INPE desenvolve complexos sistemas embarcados para satélites científicos e balões estratosféricos; consequentemente, os processos de verificação e validação exigem cuidados especiais na detecção e prevenção de falhas. Neste cenário, aplicam-se técnicas especiais como a modelagem formal baseada em requisitos, que auxilia no processo de desenvolvimento permitindo a geração sistemática e automatizada de casos de teste para sistemas antes mesmo de sua implementação final. A geração e execução automática de casos de teste proporciona um ganho significativo de produtividade no trabalho de especialistas que, aliado a um oráculo de resultados de testes, pode avaliar o comportamento de sistemas reativos diante dos eventos aos quais estará sujeito.*

Palavras-chave: *Modelagem de Software, Teste de Software Baseado em Modelo, Caso de Teste, Oráculo de Teste.*

1. Introdução

Os custos de V&V (Verificação e Validação) [Sommerville 2007] de sistemas críticos em geral são muito altos, podendo alcançar mais de 50% do total de custos do desenvolvimento. Isso é ainda mais comum em sistemas espaciais que desempenham tarefas complexas e envolvem um hardware de engenharia complexa e de alto custo. Tendo em vista esse grande investimento, as agências espaciais não podem abrir mão de um bom plano de testes do software embarcado, até porque o sucesso da missão depende por completo do software. Agências de pesquisas espaciais como o INPE desenvolvem software para computadores embarcados em satélites e balões estratosféricos.

O que este tipo de software tem como principal característica é o fato de interagir constantemente com sensores, atuadores e outros dispositivos. A partir destes dispositivos

estes computadores de bordo respondem constantemente a estímulos e eventos tornando-se um sistema reativo. Além disso, o hardware envolvido muitas vezes precisa transpor problemas de limitação de espaço, peso e consumo de energia; isso faz com que o desenvolvimento de software para aplicações espaciais seja uma área muito restrita e que exige um alto grau de especialização da mão de obra, exigindo um processo de teste formal. Neste cenário, a utilização de ferramentas de suporte a geração e execução de casos de teste, além da análise de resultados baseado em um oráculo de teste, trazem grandes benefícios para os testadores e para a qualidade final do produto.

2. Sistemas Críticos

Quando se constata uma falha em sistemas convencionais, geralmente seu resultado é o surgimento de contratempos e inconveniência para os usuários ou operadores. No entanto, existem outros tipos de sistemas cujas consequências podem ser muito piores podendo envolver danos ambientais, prejuízos financeiros ou até mesmo um risco às vidas humanas, e estes são chamados de sistemas críticos. Falhas em sistemas críticos têm geralmente um alto custo de reparo, e além do custo provocado diretamente pela falha em si, podem estar inclusos a substituição do sistema e custos indiretos devido à indisponibilidade, processos judiciais, etc. Por isso é interessante mencionar que o desenvolvimento de sistemas críticos pode apresentar duas características um tanto incomuns. Primeiro, são em geral desenvolvidos com a utilização de técnicas bem experimentadas e testadas ao invés de adotar inovações sem muita base de experiência em aplicações práticas. E segundo, técnicas de engenharia de software que não são normalmente economicamente viáveis para projetos convencionais são utilizadas no desenvolvimento de sistemas críticos, como o uso de especificação e verificação formal em relação aos seus requisitos, e ferramentas de suporte as tarefas.

3. Modelagem de Sistemas

Os modelos têm se tornado comuns na especificação de aplicações de um domínio específico, e em particular no desenvolvimento de software. Em nosso cenário de estudo composto de aplicações espaciais que trabalham em tempo-real e são essencialmente reativas, o modelo comportamental é muito mais condizente com a realidade, pois representa de forma muito mais natural o comportamento do sistema quando é afetado por entradas ou estímulos. Dentre as técnicas de modelagem comportamental, a mais adequada nesses casos seria a modelagem em Statecharts proposta por Harel (1987), e que foi a base para a notação de modelagem de Statecharts na UML (Unified Modeling Language) [Hartman 2005].

Statecharts é um grafismo visual baseado nas MEF convencionais e muito utilizado para a especificação de sistemas reativos [Harel 1987]. Na verdade, é uma técnica que estende as MEF possibilitando a representação de recursos mais avançados e comuns na especificação de sistemas complexos como noções de hierarquia e paralelismo. Essa notação vem sendo utilizada por alguns grupos do INPE para especificação formal de sistemas embarcados que, em sua essência, são reativos devido ao fato de seu funcionamento ser baseado em captação de sensores e atuadores que alteram seu comportamento. Esse tipo de aplicação requer testes cuidadosos, ou seja, organizados e formalizados e, de preferência, com a automação dos processos envolvidos.

4. Testes de Software Baseado em Modelo

A modelagem comportamental do software pode se tornar uma importante técnica para que, a partir dela, possam ser obtidos casos de teste mais rapidamente permitindo a detecção de falhas através de testes funcionais [Myers 2004]. Casos de teste podem ser descritos de forma geral como uma sequencia de ações que afetam o comportamento de um sistema e, como nosso foco é em sistemas reativos, a sequencia de teste pode ser definida como uma sequencia de eventos que provêm estímulos (entradas) ao sistema fazendo com ele mude seu comportamento. A literatura especializada já descreveu várias metodologias para geração de casos de teste a partir de uma representação formal de software e várias ferramentas comerciais ou acadêmicas geram casos de teste automaticamente baseado em algum tipo de representação do software como MEF, UML, SDL e Statecharts.

Para se gerar casos de teste a partir da MEF, por exemplo, faz-se o uso de alguns métodos já descritos na literatura, como: Transition Tour [Lee and Yannakakis 1996], Switch Cover [Martins et al. 2000], Método UIO (*Unique Input/Output*) [Myers 2004] e Método DS (*Distinguishing Sequence*) [Pimont and Rault 1979]. Entretanto, é necessário lembrar que alguns recursos comumente presentes em sistemas complexos, como atividades paralelas e encapsulamento, são muito difíceis de serem representados utilizando modelos de máquina de estados podendo induzir o especialista ao erro. Isso pode prejudicar a formulação do modelo fazendo com que ele não resulte em uma representação fiel ao sistema real e, consequentemente, compromete a obtenção dos casos de teste. É na prevenção deste problema que um grupo do INPE optou pela utilização de uma notação de alto nível e com mais recursos de modelagem , no caso o Statecharts. E como Statecharts é um modelo baseado em MEF, optou-se para converter a modelagem Statecharts em uma MEF já que é possível fazer esta conversão através de um algoritmo computacional e, consequentemente, utilizar métodos geradores de casos de teste a partir de MEF como a ferramenta WEB-PerformCharts.

4.1. WEB-PerformCharts

A WEB-PerformCharts [Arantes 2008], como o próprio nome sugere, tem como sua principal característica o acesso pela internet via browser. A ideia principal durante sua concepção foi permitir que testadores de software em qualquer lugar do planeta pudessem submeter suas especificações em Statecharts (escritos em XML) para o servidor e executar ferramentas de geração de casos de teste on-line obtendo as sequências de teste imediatamente e sem o excessivo trabalho manual da primeira metodologia. A WEB-PerformCharts tem algumas funcionalidades como a capacidade de compartilhar projetos e gerenciamento de usuários pelos administradores do sistema, pois foi desenvolvida visando dar suporte para uma possível metodologia de trabalho cooperativo em testes. A plataforma faz uso de um banco de dados onde são armazenados todos os dados das especificações em Statecharts, saídas, MEF e casos de teste. Esta ferramenta pode fazer a geração de casos de teste utilizando quatro critérios que foram incorporados, são eles o *Transition Tour*, *Switch Cover*, UIO e DS [Arantes 2008] [Ferreira et al. 2010].

5. Metodologia de Testes

5.1. Geração de Casos de Teste com a WEB-PerformCharts

A geração de casos de teste com a ferramenta WEB-PerformCharts segue a metodologia ilustrada na Figura 1.

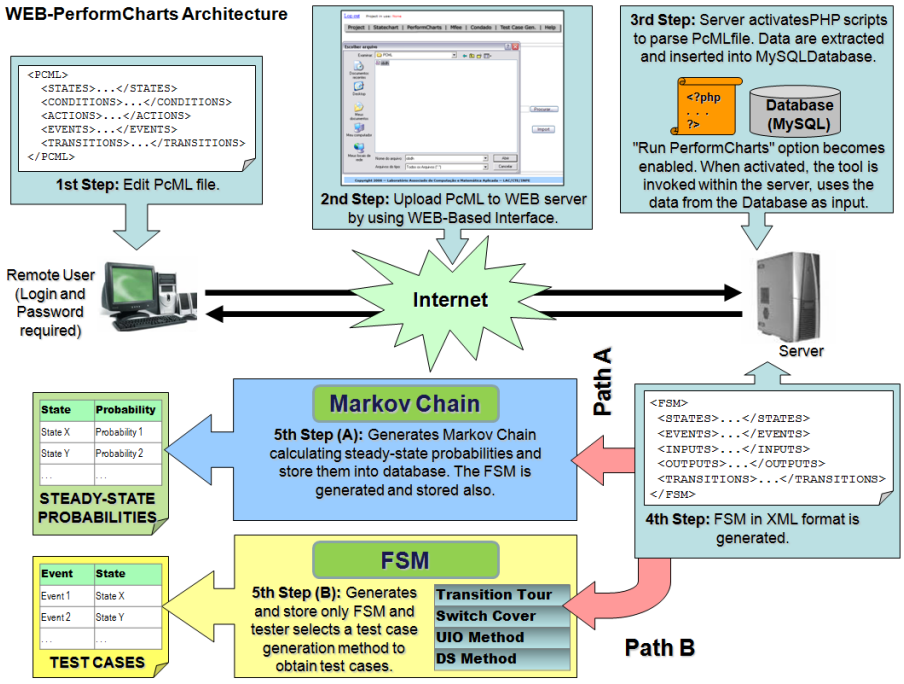

Figura 1. Metodologia de geração de casos de testa na WEB-PerformCharts. [Arantes 2008].

Todas as informações manipuladas pela ferramenta como as especificações em Statecharts, MEF, casos de teste, etc; são armazenadas em banco de dados on-line e disponíveis em tempo real para o acesso de qualquer parte do globo. O servidor com a aplicação e o banco de dados encontra-se no INPE e acessível pelo seguinte endereço: http://www.lac.inpe.br/WEB-PerformCharts

6. Execução dos casos de teste

O objetivo da execução dos casos de teste é submeter a IUT (*Implementation Under Test*) a um conjunto de entradas e condições de execução, e então analisar resultados com um objetivo particular. Em se tratando de software complexo, a execução dos casos de teste pode se tornar uma atividade extremamente onerosa em virtude da quantidade de sequências sucessivas de testes que podem ser necessárias para o teste dos instrumentos, subsistemas e sistemas de uma aplicação espacial. Com isso, para melhorar a produtividade da atividade de testes, uma metodologia de automatização de execução de casos de teste foi estudada e implementada resultando na ferramenta QSEE-TAS (Qualidade do Software Embarcado em aplicações Espaciais - Teste Automatizado de Software) desenvolvida em LabView [Silva 2009]. A interface principal do sistema pode ser visualizado na Figura 2.

A etapa de execução dos casos de teste observa o comportamento da IUT à medida que os estímulos são transmitidos via hardware. A interação do executor de testes com a IUT é traduzida em dados que são coletados para um relatório utilizado para análise do especialista responsável por dar o veredito ao teste. Para auxiliar o especialista no julgamento do veredito, pode-se utilizar como auxílio oráculo de teste.

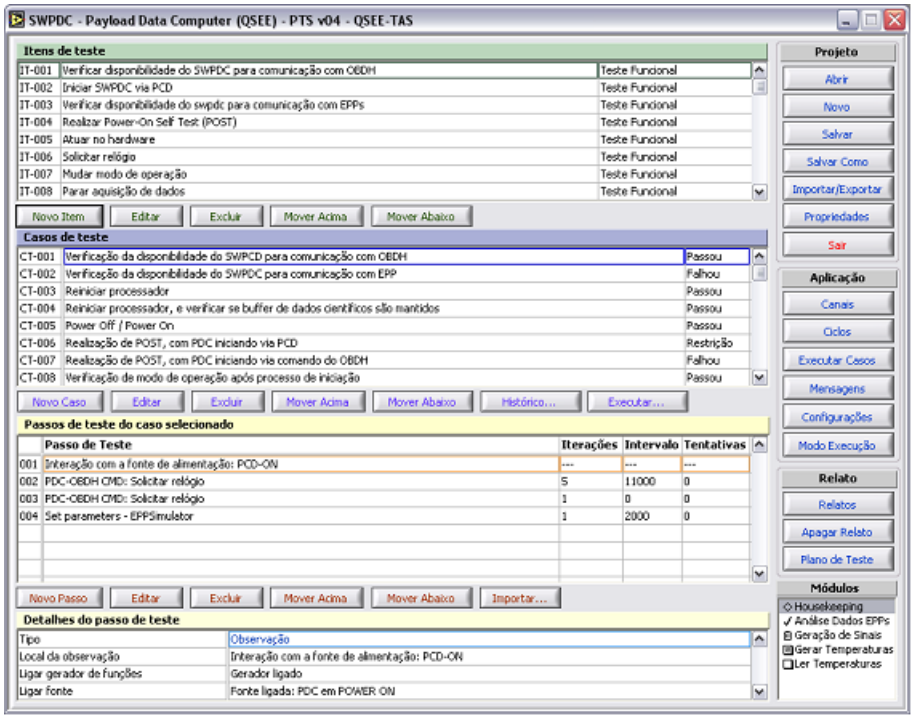

Figura 2. Interface principal do QSEE-TAS.
[Silva 2009]

7. Análise de Resultados com Auxílio de Oráculo

Em testes de software, o oráculo é um instrumento que vem sendo largamente utilizado como arcabouço de resultados esperados para determinados casos de teste aplicados ao sistema em teste [Binder 2000]. É como um conjunto de tuplas onde para cada entrada possível, uma saída esperada é associada. Com isso, é possível comparar a saída obtida de um sistema com a saída esperada, facilitando o trabalho do testador na detecção de falhas. Podemos ilustrar essa definição, utilizando como exemplo uma abordagem de teste caixa-preta, conforme a Figura 3.

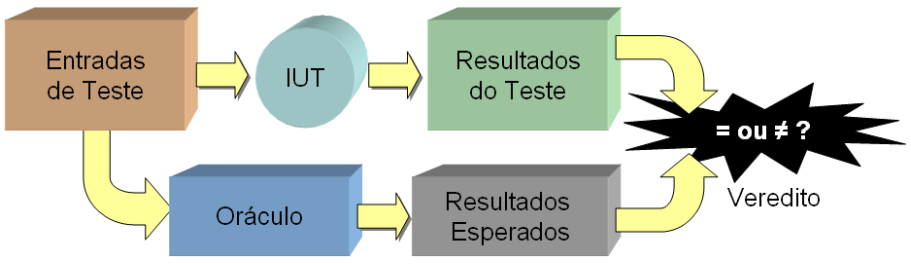

Figura 3. Esquema básico de um oráculo de teste.
[Shahamiri et al. 2009].

Em seu capítulo sobre oráculos, Binder (2000) comenta 16 padrões de abordagens para oráculos divididos em 4 grandes grupos. Baseado nas várias abordagens, pode-se dizer que não existe uma regra geral para o desenvolvimento do oráculo, e a escolha da abordagem deve depender de vários fatores. Mas é certo que, independente da abordagem utilizada, para que possa ser utilizado adequadamente como ferramenta de apoio o TOG

(*Test Oracle Generator*) deve representar uma fonte confiável de resultados esperados para cada comportamento previsto em suas especificações [Shahamiri et al. 2009], e este é o chamado "Problema do oráculo" [Singh et al. 2011].

Oráculos de teste são formados basicamente por dois componentes, *oracle information* e *oracle procedure* [Xie and Memon 2007]. *Oracle information* é sua fonte formadora de resultados esperados, enquanto *oracle procedure* representa o comparador. O seu mecanismo comparador é responsável pela análise dos resultados e veredito [Xie and Memon 2007]. Além de tipos primitivos, é comum em sistemas modernos a produção de saídas como arquivos, instâncias de classes, bancos de dados, *dumps* de memória e comandos de hardware. Binder (2000) classifica os comparadores em 3 tipos básicos:

- *System utilities:* comparadores básicos disponíveis nas linguagens de programação.
- *Smart comparators:* software desenvolvidos por terceiros como comparadores de arquivos, bancos de dados e documentos.
- *Application-specific comparator:* rotinas desenvolvidas com propósito específico da aplicação como comparar objetos de uma classe, por exemplo.

Tendo em vista a complexidade que o desenvolvimento de um TOG pode oferecer, é justificável uma reflexão a respeito de seus benefícios em relação ao esforço em implementação. Um grupo do INPE atualmente faz uso das abordagens *Solved Examples Oracle* e *Judging* e, mesmo sendo processos que requerem trabalho manual, já se agrega melhorias na detecção de falhas [Santiago et al. 2007]. No caso do INPE, os casos de teste são importados em XML pela ferramenta QSEE-TAS e o testador entra com os resultados esperados para cada caso manualmente. A diferença em relação à abordagem de Binder (2000) é que o veredito final não é feito por apenas uma pessoa. Uma equipe observa as saídas e fornecem seu veredito preliminar baseado em documentos de requisitos e especificações. Reuniões semanais são feitas por uma equipe de IVV (*Independent Verification and Validation*) com a finalidade de avaliar os relatórios de testes. Caso a equipe julgue um veredito como *no pass*, uma não conformidade com a especificação é registrada. Vereditos divergentes são revisados com o objetivo de se determinar uma saída esperada com o consenso da equipe.

8. Conlusões

A metodologia de geração e execução de casos de testes estudada, foi baseada nos trabalhos desenvolvidos na área de desenvolvimento de sistemas embarcados do INPE. O estudo sobre avaliação de resultados de teste com o auxílio de oráculos automatizados surgiu em consequência ao avanço das pesquisas do que se diz respeito a automatização de processo de teste com as ferramentas WEB-PerformCharts e QSEE-TAS.

Foi constatado que a modelagem de sistemas é um recurso formidável para obtenção de casos de teste e, como já mencionado em seu devido capítulo, a escolha do formalismo está diretamente ligada à facilidade de implementação para o escopo da aplicação sob teste. Tais modelos são concebidos baseados na especificação dos requisitos do sistema, e interpretar especificações em LN e formalizar um modelo não é uma tarefa trivial e, ao contrário disso, requer tempo e experiência por parte do testador que possivelmente terá que lidar com informações ambíguas, inconsistentes ou incompletas. Neste caso, a

metodologia SOLIMVA [Santiago 2011] pode ser utilizada com o intuito de amenizar o esforço despendido pelo testador na compreensão e especificação formal da IUT.

Apesar da principal metodologia de teste estudada ser baseada na abordagem caixa-preta, é necessário mencionar também que é possível fazer a geração de casos de teste utilizando a abordagem caixa-branca. Afinal, Marinke (2011) demonstrou em seu trabalho que é possível converter código fonte de software concorrente escrito em Java, em uma especificação equivalente em Statecharts. E a partir desta especificação, é possível obter a MEF e gerar casos de teste usando a ferramenta WEB-PerformCharts ou GTSC.

No caso específico dos oráculos de testes, foi possível primeiramente observar uma gama muito grande de abordagens como solução propostas por Binder (2000) e na pesquisa bibliográfica que menciona até soluções complexas utilizando recursos estudados em inteligência artificial como as RNA e *machine learning*. No entanto, dentre as propostas que apresentam um processo de geração automatizado, é possível observar que a maioria é baseada em alguma especificação ou linguagem formal para a descrição detalhada do oráculo. Assim como ocorre na geração de casos de testes, a escolha do formalismo utilizado ocorre de forma muito particular dependendo da aplicação envolvida e, principalmente, da disponibilidade de recursos que definem os requisitos para a construção da especificação.

Em trabalhos futuros, seria interessante intensificar as pesquisas em torno da modelagem de oráculos automatizados para testes, bem como as abordagens possivelmente utilizáveis para a sua construção voltada às aplicações espaciais complexas no âmbito do INPE. Afinal, a automatização de avaliação de testes reduz a necessidade de mão de obra em trabalhos manuais durante o processo acarretando economia de recursos. E além disso, somente com a automatização da avaliação dos resultados seria possível realizar testes exaustivos em sistemas complexos, tendo em vista de que a geração e execução de casos de teste já se encontram neste patamar.

Referências

Arantes, A. O. (2008). Web-performcharts: a web-based test case generator from statecharts modeling. (INPE-15379-TDI/1398).

Binder, R. V. (2000). *Testing object-oriented systems: models, patterns and tools.* Addison-Wealey, Boston, MA.

Ferreira, D. F., Nunes, M. K. P., Ferreira, E., Arantes, A., and Vijaykumar, N. L. (2010). Integração de métodos de teste à ferramenta web-performcharts. *Simpósio de Iniciação Científica e Tecnológica*. sid.inpe.br/mtc-m19/2012/01.11.14.05.

Harel, D. (1987). Statecharts: a visual formalism for complex systems. 8:237–274. Science of Computer Programming.

Hartman, A. (2005). Uml 2.0 infrastructure specification.

Lee, D. and Yannakakis, M. (1996). Principles and methods of testing finite state machines - a survey. (84(8)). Proceedings of the IEEE.

Martins, E., Sabião, S. B., and Ambrósio, A. M. (2000). Condata: a tool for automating specification-based test case generation for communication systems. 33rd Hawaii International Conference on System Sciences.

Myers, G. (2004). *The art of software testing.* John Wiley & Sons.

Pimont, S. and Rault, J. C. (1979). An approach towards reliable software. pages 220–230. Proceedings of the 4th International Conference on Software Engineering.

Santiago, V., Mattiello-Francisco, M. F., Costa, R., Silva, W. P., and Ambrósio, A. M. (2007). Qsee project: an experience in outsourcing software development for space applications. pages 51–56.

Santiago, V. A. d. (2011). Solimva: A methodology for generating model-based test cases from natural language requirements and detecting incompleteness in software specifications. sid.inpe.br/mtc-m19/2011/11.07.23.30-TDI.

Shahamiri, S. R., Kadir, W. M. N. W., and Mohd-HashimSidhu, S. Z. (2009). A comparative study on automated software test oracle methods. pages 140–145. ICSEA Ó9 Proceedings of the 2009 Fourth International Conference on Software Engineering Advances.

Silva, W. P. (2009). Qsee-tas: execução automatizada de casos de teste para software embarcado em aplicações espaciais.

Singh, A., Kang, S., and Bajwa, S. (2011). Metamorphic testing: Using the properties of sut.

Sommerville, I. (2007). *Software engineering.* Addison Wesley. 568 p.

Xie, Q. and Memon, A. M. (2007). Designing and comparing automated test oracles for gui-based software applications. 16(1):4. ACM Transactions on Software Engineering and Methodology.