Revista Eletrônica de Sistemas de Informação
ISSN 1677-3071

No 1 (8)

2009

Sumário

Revista hospedada em: http://revistas.facecla.com.br/index.php/reinfo
Forma de avaliação: *double blind review*

DERIVAÇÃO DE CASOS DE TESTES FUNCIONAIS: UMA ABORDAGEM BASEADA EM MODELOS UML[1]

FUNCTIONAL TEST CASE DEPLOYMENT: AN APPROACH BASED ON UML MODELS

(artigo submetido em maio de 2009)

Alex Mulattieri Suarez Orozco
Instituto Federal Farroupilha
alxorozco@gmail.com

Kleinner Oliveira
Pontifícia Universidade Católica do Rio de Janeiro
kfarias@inf.puc-rio.br

Flávio Oliveira
Pontifícia Universidade Católica do Rio Grande do Sul
flavio.oliveira@pucrs.br

Avelino Francisco Zorzo
Pontifícia Universidade Católica do Rio Grande do Sul
zorzo@pucrs.br

ABSTRACT

In this paper we present a model-based testing approach that aims at identifying, automating and deriving functional test cases in whole or in part from UML models that describe some aspects of the system under test. Our approach provides a considerable reduction of the effort on the test generation, increasing the effectiveness of the tests, shortening the testing cycle, and avoiding tedious and error-prone editing of a suite of hand-crafted tests. Finally, we present a case study to demonstrate the practicality and usefulness of the proposed approach.

Key-words: model-based testing; UML; functional testing.

RESUMO

Neste artigo é apresentada uma abordagem de teste de software baseada em modelos que se concentra na identificação, automatização e derivação completa, ou parcial, de casos de teste a partir da composição de modelos UML que descrevem alguns aspectos do sistema que está sendo testado. A abordagem adotada provê uma considerável redução de esforço na geração de testes, aumentando a sua eficiência, diminuindo o ciclo de testes e evitando a realização tediosa e propensa a erros de um conjunto de casos de testes. Por fim, é apresentado um estudo de caso para demonstrar de modo prático os benefícios da abordagem proposta.

Palavras-chave: testes baseados em modelos; UML; testes funcionais.

[1] Este trabalho foi desenvolvido em colaboração com a HP Brasil P&D.

1 INTRODUÇÃO

Um fator significante que evidencia a dificuldade encontrada durante o processo de desenvolvimento de software é a existência de um *gap* conceitual entre o domínio do problema e o domínio da solução (FRANCE, 2006; FRANCE, 2007). Diferentes paradigmas de desenvolvimento têm sido propostos com o objetivo de solucionar este *gap*, tais como: desenvolvimento de software orientado a objetos (OOP), desenvolvimento de software orientado a aspectos (AOP) e, mais recentemente, desenvolvimento de software dirigido por modelos (MDD).

Estes paradigmas de desenvolvimento são utilizados em alguns processos de desenvolvimento de software, como RUP (tradicional) (KRUCHTEN, 1999) e SCRUM (ágil) (SCHWABER, 2004), durante a execução das atividades inerentes ao processo. Porém, independente do paradigma de desenvolvimento e do tipo de processo utilizados, precisa-se, sem dúvida, da elaboração de produtos de software de qualidade. Logo, as atividades e técnicas relacionadas a testes de software têm se diversificado e vêm ganhando cada vez mais ênfase dentro dos processos de desenvolvimento visando a atender esta necessidade. Um exemplo destas técnicas é o teste de software dirigido por modelos. Se, por um lado, fica claro que a realização de testes tem um papel importante na aferição da qualidade de um software, por outro lado, a execução incompleta ou inadequada de testes pode proporcionar problemas que possivelmente comprometerão o bom funcionamento, o desempenho e a confiança no funcionamento do sistema.

Diante do dinamismo e da complexidade dos sistemas de software atuais, torna-se extremamente desafiante para os testadores criar, derivar casos de testes e colocar testes de software em prática dado o problema em mãos. Consequentemente, testar software exige extensiva habilidade dos testadores tanto em relação aos diferentes tipos de testes quanto às técnicas de automatização e aplicação de teste “antecipado”, ou seja, nas fases inicias do processo. Isto pode causar o surgimento de complexidades adicionais, o que torna a atividade de testes mais custosa e difícil. A automação dos testes pode reduzir o esforço requerido para as atividades de testes de software. Por meio da automação, os testes podem ser realizados em um menor tempo, ao invés de demorarem horas para serem executados manualmente, podendo alcançar uma diminuição de esforço em mais de 80% (FEWSTER e GRAHAM 1999). A aplicação de teste automatizado nas organizações pode reduzir gastos ou esforços de forma indireta. Além disso, a automação permite produzir softwares de melhor qualidade mais rapidamente do que seria possível utilizando-se testes manuais.

Neste artigo apresenta-se uma abordagem de teste de software baseada em modelos que se concentra na identificação, automatização e derivação completa, ou parcial, de casos de teste funcional, a partir do diagrama de atividades da UML (Unified Modeling Language) (OMG, 2007). O objetivo

é realizar teste funcional nas fases iniciais do processo de desenvolvimento. A abordagem adotada provê uma considerável redução de esforço na geração de testes por meio da composição dos diferentes diagramas de atividades do sistema a ser testado em um único diagrama, evitando assim atividades redundantes e, conseqüentemente, casos de teste redundantes. É apresentado um estudo de caso para demonstrar, de modo prático, os benefícios da abordagem proposta.

Este artigo é organizado da seguinte forma. A seção 2 apresenta o referencial teórico desta pesquisa. A seção 3 e seção 4 apresentam a abordagem proposta. A seção 5 apresenta um estudo de caso, mostrando a abordagem na prática. A seção 6 apresenta os trabalhos relacionados. Finalmente, na seção 7, são apresentadas as conclusões e sugestões para trabalhos futuros.

2 REFERENCIAL TEÓRICO

Nesta seção são discutidos os principais conceitos necessários para a compreensão da técnica de automatização de testes proposta neste artigo.

2.1 TESTE FUNCIONAL

Teste funcional consiste na realização de um conjunto de atividade que visam a localizar discrepâncias entre o comportamento do sistema sob teste e as especificações de comportamento esperado. Tais especificações são descrições precisas do comportamento do programa do ponto de vista do usuário final. Em outras palavras, o teste funcional tenta verificar as funcionalidades do sistema considerando uma dada entrada de dados, processamento e resposta. Para prover um teste funcional, as especificações são analisadas para derivarem um conjunto de casos de testes. Exceto quando utilizado em pequenos programas, o teste funcional é uma atividade conhecida como caixa preta, ou seja, se confia que um processo de teste unitário anterior alcançou um critério de cobertura lógica aceitável (MYERS 2004).

2.2 UML

A UML (OMG 2007) trata-se de uma linguagem padrão para modelagem de sistemas orientados a objetos, sendo usada tanto na academia quanto na indústria. Apresenta um conjunto de diagramas que são usados para representar aspectos estáticos e dinâmicos de um sistema em desenvolvimento. Dentre estes diagramas, encontra-se o diagrama de atividades que, na sua essência, representa os fluxos conduzidos por algum tipo de processamento. Trata-se de um gráfico de fluxo, mostrando o fluxo de controle de uma atividade para outra. Neste artigo, este diagrama é um elemento central para permitir a derivação automática dos casos de teste funcional.

2.3 MÁQUINA DE ESTADOS FINITOS COM ENTRADAS E SAÍDAS

Uma máquina de estados finitos (MEF) com entradas e saídas é definida como uma 6-tupla (S,I,A,R,Δ,T), onde S é o conjunto finito de estados, $i \subset S$, é o conjunto de estados iniciais, A é o alfabeto

finito de símbolos de entrada e R é o conjunto de possíveis saídas ou respostas. O conjunto $\Delta \subset S \times A$ é o domínio da relação de transição T, que é uma função de Δ para $S \times R$.

A relação de transição descreve como a máquina reage ao receber entrada $a \in A$ quando $s \in S$, assumindo que $(s,a) \in \Delta$. Interpreta-se $(s,a) \notin \Delta$ como: o símbolo de entrada não pode ser aceito no determinado estado. Quando $T(s,a) = (s',r)$, o sistema se move para o novo estado s' e responde como saída r. Se T não é uma função, mas mais exatamente uma relação que associa cada par de estado de entrada com um conjunto não vazio de pares de estados de saída, é dito que o autômato finito é não determinístico, e é interpretado como o conjunto de possíveis respostas para um estímulo de entrada em um certo estado (FRIEDMAN *et al.*, 2002).

2.4 MÉTODO UIO

O método UIO (*Unique Input/ Output*) é um método para a geração de um conjunto de sequências de entrada para testar uma MEF (DELAMARO *et al.,* 2007). Para a geração destas sequências, este método utiliza sequências UIO.

As sequências UIO são utilizadas para verificar se a MEF está em um determinado estado em particular. Sendo assim, cada estado da MEF poderá possuir uma sequência UIO distinta.

Desta forma, como afirmam Delamaro *et al.* (2007, p. 42), "para cada transição de um estado s_i para s_j, $f_S(s_i, x)$, com algum x, é definida uma sequência que conduz do estado inicial a s_i, aplica-se o símbolo de entrada e, em seguida, aplica-se a sequência UIO do estado que deveria ser atingido. Sendo assim, cada sequência é da forma $P(s_i)$ x UIO(s_j), sendo que $P(s_i)$ é uma sequência que leva a MEF do estado inicial ao estado s_i e UIO(s_j) é uma sequência para s_j".

3 DERIVAÇÃO DE CASOS DE TESTES FUNCIONAIS

Para possibilitar a geração de casos de testes, este trabalho propõe a utilização de um diagrama de atividades dos sistemas que está sendo testado. Por meio deste modelo, é possível, ainda na fase de requisitos, começar a pensar em como o plano de testes deve ser criado. Porém, o diagrama de atividades não contém informações suficientes para a geração dos casos de teste. É necessário inserir as características relacionadas a teste funcional. A Figura 1 exibe o *overview* da abordagem proposta.

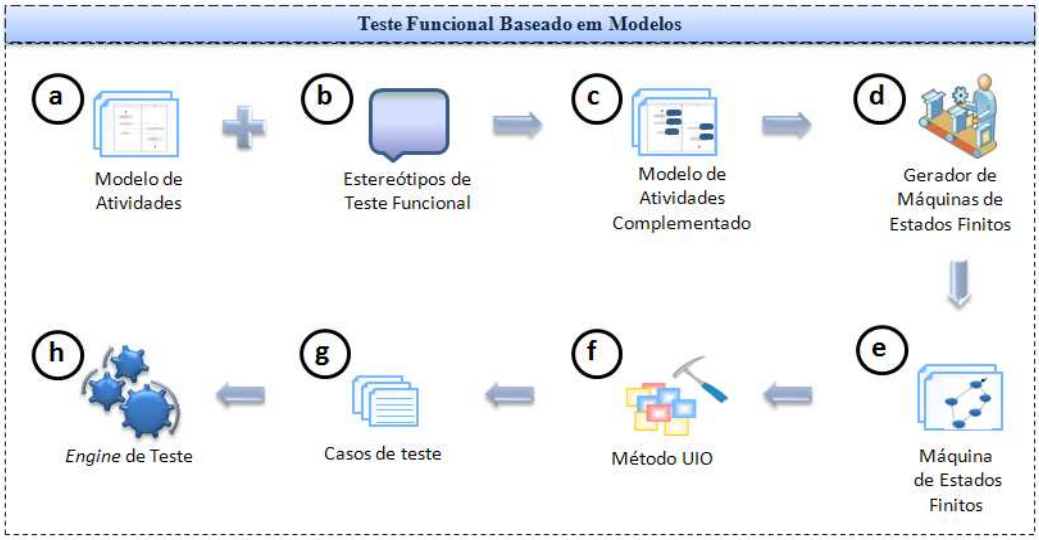

Figura 1. Um *overview* da abordagem proposta.

De posse dos modelos de atividades do sistema em teste (a), é inserido em cada atividade relevante para o teste um estereótipo definido como *<FTStep>* (b). Este estereótipo é composto de duas marcações, uma denominada *FTAction*, responsável por especificar a ação a ser realizada no sistema, e a outra denominada *FTExpectResult*, responsável por especificar o resultado esperado da ação definida na *FTAction*.

Após a inclusão destas informações, o modelo de atividades complementado (c) é convertido em uma máquina de estados finitos com entradas e saídas (d), onde o alfabeto de entrada é formado pelo conteúdo das marcações *FTAction*, o alfabeto de saída é formado pelo conteúdo das marcações *FTExpectedResult* e os estados são formados pelas atividades do diagrama de atividades. Considerando um diagrama de atividades composto por duas atividades *A1* e *A2*, onde *A1* leva a *A2*, a máquina de estados finitos equivalente será formada pelos estados *A1* e *A2*, a entrada da transição *A1→A2* será o conteúdo da marcação *FTAction* da atividade *A2* e a saída da transição será o conteúdo da marcação *FTExpected Result* da atividade *A2*.

Nesta máquina de estados finitos (e) é aplicado o método de geração de sequência de entrada para testar a máquina conhecido como método UIO (f) (DELAMARO *et al.*, 2007). Este método gera uma sequência única de entrada e saída para alcançar cada estado da máquina de estados finitos. Desta forma, cada sequência UIO resulta em um caso de teste (g). Com isto, ainda durante a fase de requisitos, é possível ter uma idéia sobre os prováveis casos de teste a serem realizados quando a aplicação estiver implementada, sendo estes casos de teste refinados à medida que o processo de desenvolvimento evolui, bastando atualizar os dados existentes nas marcações do modelo e gerar os casos de teste novamente. Esta abordagem resulta na vantagem que, para cada atividade exposta no diagrama de atividades, é gerado um caso de teste, provendo um critério de cobertura bastante amplo.

Quando a aplicação estiver implementada, o modelo de atividades é atualizado com as informações relevantes para automatizar a geração de *scripts* de teste, sendo inseridas nas marcações do estereótipo *<FTStep>,* em linguagem de *script*, a ação que o objeto da interface da aplicação representada pela atividade do modelo deve realizar e o seu respectivo resultado esperado. Desta forma, os casos de teste gerados com estas informações podem ser executados em uma *engine* de teste (h), automatizando o processo de teste.

4 ESPECIALIZAÇÃO DA ABORDAGEM PROPOSTA

A abordagem descrita na seção 3 tenta proporcionar a derivação de casos de testes funcionais a partir do diagrama de atividades do sistema em teste, sendo necessário que cada diagrama de atividades seja exposto aos passos descritos. Para sistemas simples, esta abordagem é bastante eficiente, mas, para sistemas de porte expressivo, começa a apresentar deficiências.

Suponha-se um sistema com três diagramas de atividades, onde cada diagrama possui quatro atividades em sequência e os três diagramas possuem uma atividade denominada “Inserir nome de usuário”. Ao expor estes diagramas à abordagem da seção 3, cada atividade resulta em um caso de teste. Sendo assim, como resultados existirão doze casos de teste. Se a atividade “Inserir nome de usuário” for idêntica para os três diagramas, a abordagem gera três casos de teste redundantes, um caso de teste para cada vez que a atividade é exposta nos diagramas. Se situações como esta existirem em sistemas com um grande volume de diagramas de atividades, a quantidade de casos de teste redundantes será de um número elevado.

Para solucionar esta deficiência, é proposta a realização da composição de todos os diagramas de atividades em um único diagrama, como exibido na Figura 2.

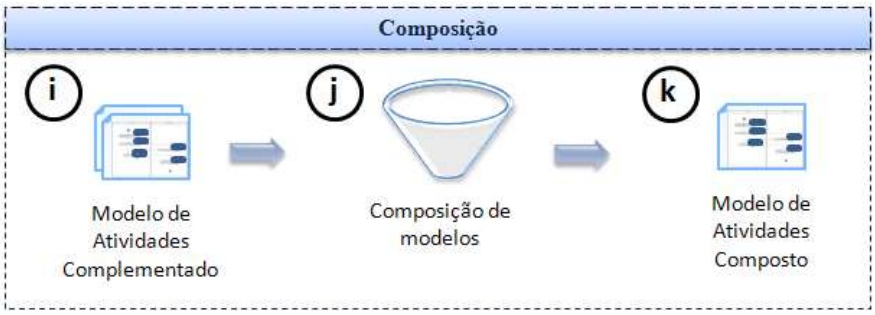

Figura 2. Processo de composição dos diagramas de atividades.

A composição pode ser definida como uma atividade de transformação, onde dois modelos *Ma* e *Mb* são transformados em *Mab*, o resultado da composição. Na abordagem aqui utilizada *Ma* e *Mb* são diagramas de atividades. Desse modo, para realizar a composição, é preciso definir quando as atividades do diagrama são consideradas iguais. Neste trabalho foi definido que duas atividades

são iguais quando a ação a ser realizada na atividade (conteúdo da marcação FTAction) e o resultado esperado desta ação (conteúdo da marcação FTExpectedResult) são iguais.

Desta forma, aplica-se a todos os diagramas de atividades do sistema em teste (i) esta regra de composição (j), resultando em um único modelo de atividades (k). Com isso eliminam-se as atividades redundantes, do ponto de vista do teste funcional. Com esta solução, a abordagem apresentada na Figura 1 é atualizada para a exibida na Figura 3, onde o passo (c) é substituído pelo processo da Figura 2, sendo que os modelos de atividades (a) com o estereótipo inserido (b) passam pelo processo de composição (c) e o modelo resultante da composição é convertido em uma máquina de estados finitos (d).

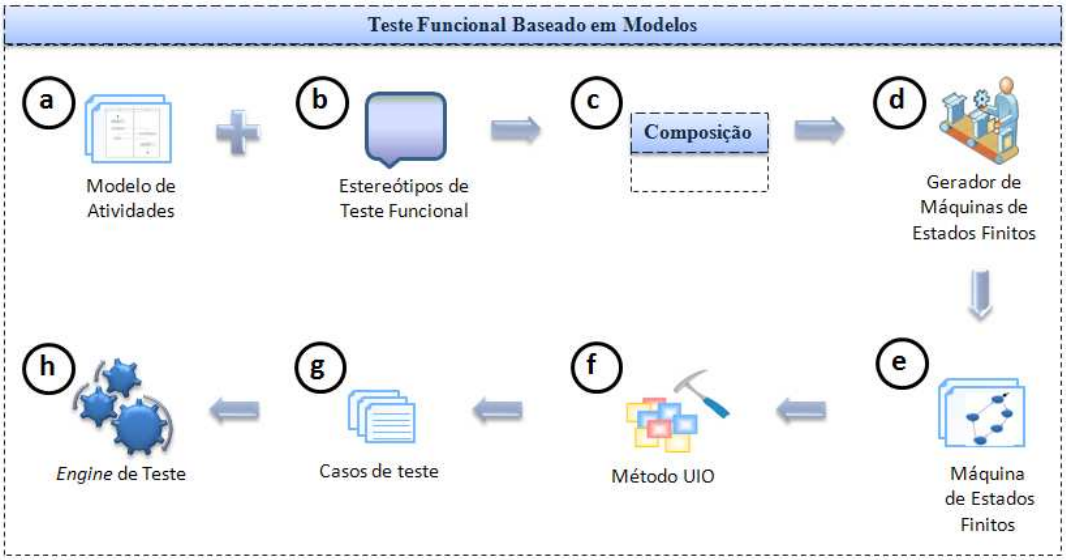

Figure 3. Abordagem completa.

5 ESTUDO DE CASO

Para avaliar a abordagem proposta, foi definido e desenvolvido um estudo de caso que é descrito nesta seção. Este estudo é concentrado nas funcionalidades presentes em calculadoras tradicionais, especificamente as funcionalidades apresentadas pelas operações de soma e divisão. Basicamente, são definidos dois casos de uso, *somar* e *dividir*. Os respectivos diagramas de atividades para estes dois casos de uso são exibidos na Figura 4.

Analisando as atividades dos diagramas para aplicar a composição, verifica-se que somente a primeira atividade de cada um dos dois diagramas são iguais, baseado na regra de composição que para duas atividades serem iguais, o conteúdo das marcações *FTAction* e *FTExpectedResult* precisam ser iguais. Nas atividades de "selecionar a operação", embora o resultado esperado seja o mesmo, a ação é diferente. O mesmo ocorre com a atividade "calcular", embora ambas possuam o mesmo conteúdo em *FTAction*, o resultado esperado é diferente.

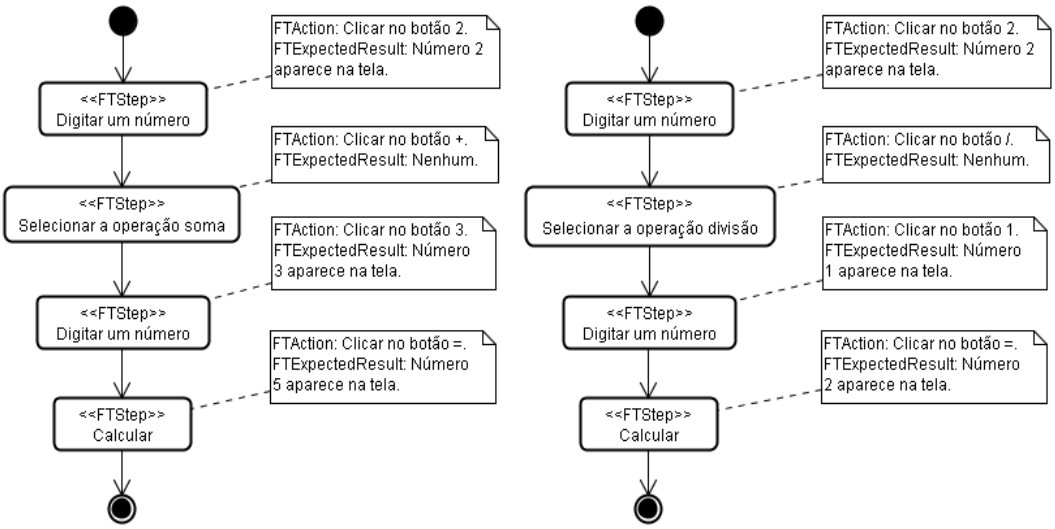

Figura 4. Diagramas de atividades referentes aos casos de uso *somar* e *dividir*.

O diagrama resultante da composição é exibido na Figura 5.

Após a composição, este diagrama resultante é inserido em um gerador de máquinas de estados finitos com entrada e saída. A máquina de estados finitos resultante é exibida na Figura 6.

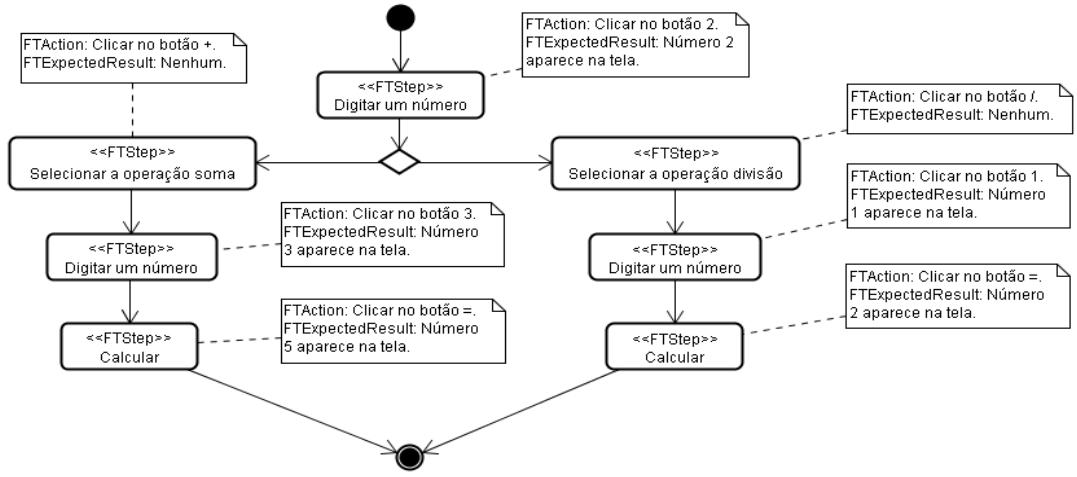

Figura 5. Diagrama de atividades após a composição.

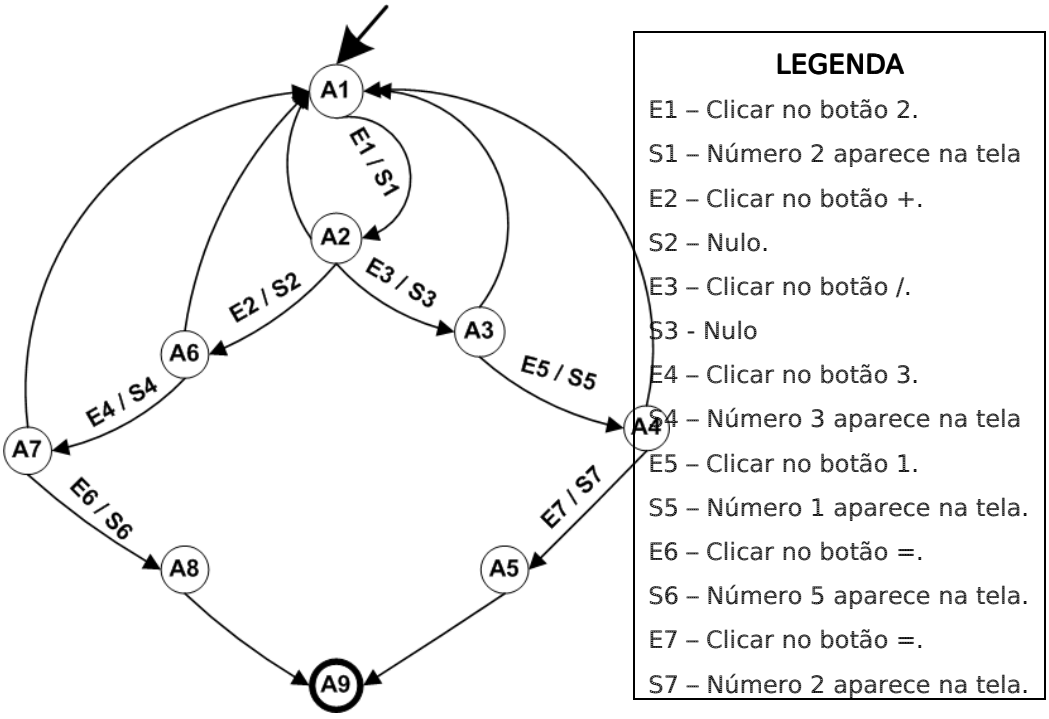

Figura 6. Máquina de estados finitos equivalente.

Aplicando o método UIO nesta máquina de estados finitos, o resultado será {(E1/S1), (E1/S1, E2/S2), (E1/S1, E2/S2, E4/S4), (E1/S1, E2/S2, E4/S4, E6/S6), (E1/S1, E3/S3), (E1/S1, E3/S3, E5/S5), (E1/S1, E3/S3, E5/S5, E7/S7)}. Desta forma, cada sequência UIO resulta em um caso de teste. Para a sequência (E1/S1, E2/S2, E4/S4, E6/S6) a descrição do caso de teste resultante seria:

1. Clicar no botão 2.

Resultado esperado: número 2 aparece na tela.

2. Clicar no botão +.

3. Clicar no botão 3.

Resultado esperado: número 3 aparece na tela.

4. Clicar no botão =.

Resultado esperado: número 5 aparece na tela.

Para a execução dos casos de teste de forma automatizada, foi utilizada a ferramenta de teste funcional *HP Quick Test Professional* (QTP, s.d.), por ser uma ferramenta para automação de testes funcionais largamente adotada pela indústria. Para a utilização desta ferramenta, os conteúdos das marcações *FTAction* e *FTExpectedResult* foram substituídos, nos diagramas de atividades dos casos de uso *somar* e *dividir*, por comandos do QTP. A abordagem aqui proposta não se limita apenas para aplicação no QTP, podendo ser utilizada em qualquer ferramenta de automação de testes funcionais baseada em *scripts*, onde os comandos da ferramenta são inseridos da mesma forma que este trabalho exemplifica para o QTP.

A Figura 7 exemplifica o diagrama de atividades do caso de uso *somar* dotado dos comandos do QTP.

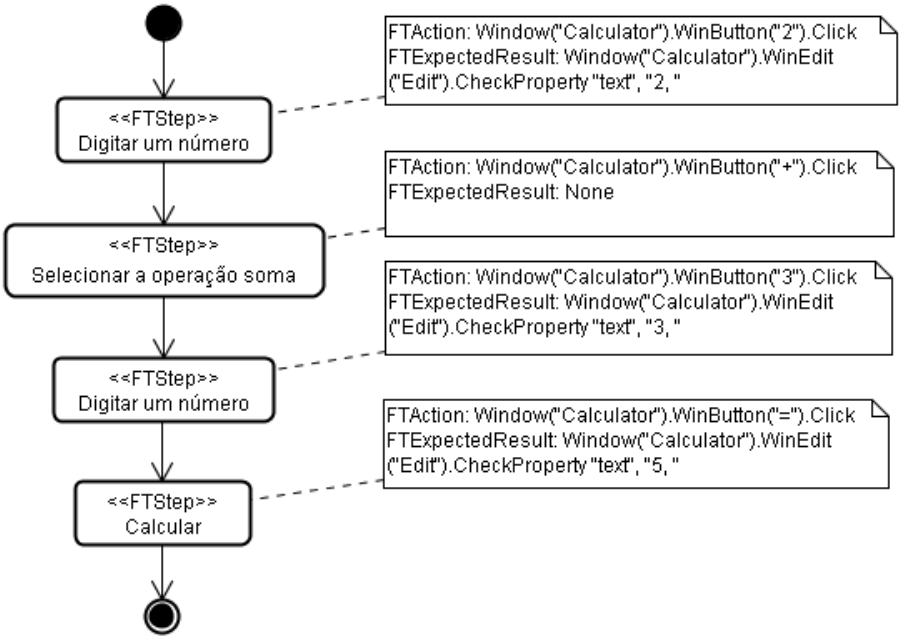

Figura 7. Diagrama de atividades referente aos casos de uso *somar*, dotado de comandos da ferramenta *HP Quick Test Professional*.

Após estes diagramas serem compostos, ser gerada a máquina de estados finitos equivalente e aplicado o método UIO, o resultado do mesmo caso de teste descrito anteriormente, agora no formato de *script* para o QTP, é apresentado abaixo:

```
Window("Calculator").WinButton("2").
Click
Window("Calculator").WinEdit("Edit")
.CheckProperty "text", "2, "
Window("Calculator").WinButton("+").
Click
Window("Calculator").WinButton("3").
Click
Window("Calculator").WinEdit("Edit")
.CheckProperty "text", "3, "
Window("Calculator").WinButton("=").
Click
Window("Calculator").WinEdit("Edit")
.CheckProperty "text", "5, "
```

Cabe ressaltar que neste estudo de caso foi necessário determinar o valor a ser inserido para a realização da soma, pois um número é representado na calculadora como um objeto botão e não como um campo de texto, ou seja, está sendo testada a funcionalidade do botão 2 da calculadora, e não somente o valor 2 para a realização da soma.

6 TRABALHOS RELACIONADOS

Alguns trabalhos apontam tentativas de facilitar o processo de testes derivando os casos de teste a partir de modelos. Oliveira *et al.* (2007) propõem a derivação de casos de teste de desempenho a partir de diagramas de atividades e casos de uso. Tais diagramas são complementados com estereótipos da *UML Profile for Schedulability, Performance and Time* (OMG, s. d.), onde os dados existentes nos diagramas mais os estereótipos provêem informações suficientes para a geração de uma rede de Petri estocástica generalizada (MARSAN *et al.*, 1984), podendo assim simular as características de

desempenho tanto de forma determinística quanto estocástica.

Vários outros trabalhos abordam o tema, tais como testes funcionais a partir de modelos de máquinas de estados finitos estendidas (PRETSCHNER *et al.,* 2005), testes de integração a partir de diagramas UML de estados (*State Chart Diagrams*) (HARTMANN *et al.,* 2000). Neto (2007) analisa 78 trabalhos sobre testes baseados em modelos, sendo 47 deles baseados em modelos UML. Rodrigues (2008) utiliza testes de desempenho baseados em modelos UML para auxiliar na realocação de recursos em ambientes virtualizados. Peralta (2008) especifica e gera casos de testes de segurança baseados em modelos UML.

Um diferencial da abordagem aqui proposta em relação aos demais trabalhos é o uso de mecanismos de composição de modelos com o objetivo de diminuir redundâncias na geração de casos de teste, visto que freqüentemente uma atividade encontra-se em diferentes diagramas de atividades do sistema a ser testado, fazendo com que ao utilizar estes diagramas individualmente para a geração de casos de teste, esta mesma atividade se repita, resultando em casos de teste redundantes.

7 CONCLUSÃO

Se os testes são vistos como elementos centrais para a garantia de qualidade de um produto de software, então um processo de desenvolvimento de software deve naturalmente se preocupar com como e quando estes testes são realizados. A fim de serem usados de uma forma mais eficiente, tais testes devem ser executados de uma forma automatizada e, preferencialmente, aplicada nas primeiras etapas do processo de desenvolvimento.

Com isso, foi apresentada uma proposta de derivação de casos de testes funcionais a partir de modelos de atividades da UML. Para realizar esta derivação, foi definida uma abordagem baseada na inserção de informações relevantes aos testes funcionais por meio da aplicação de estereótipos. Após isto, é realizada a composição dos diferentes modelos de atividades do sistema sob teste em um único modelo, o modelo composto. A partir deste modelo composto, é gerada uma máquina de estados finitos com entradas e saídas. Nesta máquina é aplicado um algoritmo para a criação de sequências de entradas e saídas que são convertidas em casos de teste para o sistema sob teste.

Foi implementada uma ferramenta que possibilitou colocar a abordagem em prática e realizar um estudo de caso. A avaliação inicial da abordagem tem demonstrado a aplicabilidade e a viabilidade da técnica desenvolvida. Obviamente, mais investigações são necessárias sobre sua eficiência para a derivação de casos de testes com diagramas de atividades da UML de tamanho elevado. Neste sentido, os trabalhos futuros serão concentrados em projetar e executar avaliações da abordagem por meio de testes na indústria. Pretende-se refinar o mecanismo de composição de modelos, visando a diminuir ainda mais as redundâncias na geração dos casos de teste.

REFERÊNCIAS

DELAMARO, M.; MALDONADO, J.; JINO, M. *Introdução ao teste de software*. Rio de Janeiro: Campus, 2007.

FEWSTER, M. e GRAHAM, D. *Software test automation*: effective use of test execution tools. New York: ACM Press/Addison-Wesley Publishing Co, 1999.

FRANCE, R. e RUMPE, B. Model-driven development of complex software: a research roadmap. In: Future of Software Engineering, co-located with ICSE'07, Minnesota, EUA, p. 37–54. 2007.

FRANCE, R., GHOSH, S. e DINH-TRONG, T. Model driven development using UML 2.0: Promises and pitfalls, *IEEE Computer Society*, v. 39, n. 2, p. 59–66. 2006.

FRIEDMAN, G.; HARTMAN, A.; NAGIN, K. e SHIRAN, T. Projected state machine coverage for software testing, In: International symposium on software testing and analysis. 2002, New York. *Anais...* ACM, p.134–143. 2002.

HARTMANN, J., IMOBERDORF, C. e MEISINGER, M. UML-based integration testing. In: International symposium on software testing and analysis. 2000, New York. *Anais...* ACM. p. 60–70. 2000.

KRUCHTEN, P. *The rational unified process*: an introduction. Boston: Addison-Wesley, 1999.

MARSAN, M.; CONTE, G. e BALBO, G. A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. *ACM Transactions on Computer Systems*, v. 2, n. 2, p. 93–122. 1984.

MYERS, G. *The Art of Software Testing*. Hoboken, USA: John Wiley & Sons, 2004.

NETO, A.; SUBRAMANYAN, R.; VIEIRA, M. e TRAVASSOS, G. A survey on model-based testing approaches: a systematic review. In: International workshop on empirical assessment of software engineering languages and technologies. 2007, New York. *Anais...* ACM Press. p. 31–36. 2007.

Object Management Group – OMG. UML profile for schedulability, performance and time. Disponível em http://www.omg.org/technology/documents/formal/schedulability.htm. Acesso em: 17/11/2008.

OLIVEIRA, F.; MENNA, R.; VIEIRA, H. e RUIZ, D. Performance testing from UML models with resource descriptions. In: Brazilian workshop on systematic and automated software testing. 2007, João Pessoa. *Anais...* Brazilian Computer Society. p. 47–54. 2007.

OMG (Object Management Group). Unified Modeling Language: infra-structure version 2.1. s.d. Disponível em http://www.omg.org/docs/formal/07-02-06.pdf. Acesso em: 17/11/2008.

PERALTA, K. Uma estratégia para especificação e geração de casos de teste de segurança usando modelos UML. 2008. Dissertação - Faculdade de Informática, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, 2008.

PRETSCHNER, A.; PRENNINGER, W.; WAGNER, S.; KÜHNEL, C.; BAUMGARTNER, M.; SOSTAWA, B.; ZÖLCH, R. e STAUNER, T. One

evaluation of model-based testing and its automation. In: International conference on software engineering. 2005, Saint Louis. *Anais...* ACM Press. p. 392–401. 2005.

QTP. Quick test professional. Disponível em https://h10078.www1.hp. com/cda/hpms/display/main/hpms_ content.jsp?zn=bto&cp=1-11-12724%5E135 2_4000_100__. Acesso em: 17/11/2008.

RODRIGUES, E. Alocação de recursos em ambientes virtualizados. 2008. Dissertação - Faculdade de Informática, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, 2008.

SCHWABER, K. *Agile project management with Scrum*. Redmond: Microsoft Press. 2004.