Uma Abordagem para a Geração de Casos de Teste Baseada em Modelos

A Model-based Approach for Test Cases Generation

J. C. Silva
Escola Superior de Tecnologia
Instituto Politécnico do Cávado e do Ave
Portugal
jcsilva@ipca.pt

J. L. Silva
IRIT
Université Paul Sabatier
France
silva@irit.fr

J. C. Campos
HASLab / INESC TEC &
Universidade do Minho
Portugal
jose.campos@di.uminho.pt

J. A. Saraiva
HASLab / INESC TEC &
Universidade do Minho
Portugal
joao.saraiva@di.uminho.pt

Resumo — **Os métodos analíticos baseados em modelos de avaliação de sistemas interativos foram propostos como alternativa aos testes com o utilizador nas últimas fases de desenvolvimento de software devido aos custos destes últimos. No entanto, a utilização isolada de modelos do comportamento do sistema limita os resultados fornecidos pelos métodos analíticos. Um exemplo dessas limitações relaciona-se com o facto destes serem incapazes de identificar problemas de implementação que terão impacto na usabilidade. Com a introdução de testes baseados em modelos é possível contornar o problema e testar se o software implementado vai de encontro ao modelo especificado. Este artigo apresenta uma abordagem para a geração de casos de testes baseados em modelos a partir da análise estática do código fonte.**

Usabilidade, Sistemas Interativos, Teste baseado em Modelos

Abstract — ***The analytical methods based on evaluation models of interactive systems were proposed as an alternative to user testing in the last stages of the software development due to its costs. However, the use of isolated behavioral models of the system limits the results of the analytical methods. An example of these limitations relates to the fact that they are unable to identify implementation issues that will impact on usability. With the introduction of model-based testing we are enable to test if the implemented software meets the specified model. This paper presents an model-based approach for test cases generation from the static analysis of source code.***

Usability, Interactive Systems, Model-based testing

I. INTRODUÇÃO

Os sistemas de software fornecem habitualmente interfaces gráficas ao utilizador. Estas representam o meio comum de interacção com o software. Nos últimos anos deparamo-nos com uma explosão das possibilidades de interacção com estes sistemas, sendo o teste de interfaces com o utilizador essencial para avaliar a sua qualidade [9]. Numa tentativa de abordar esse problema foram desenvolvidas várias técnicas de teste de interfaces gráficas. Por exemplo os testes do utilizador fornecem um elevado nível de fidelidade para avaliar a resposta do utilizador ao sistema. No entanto, esta técnica não é exaustiva em termos de cobertura de todos os potenciais problemas da interface e, além disso, os testes do utilizador são caros. Os métodos analíticos foram propostos como alternativa aos testes do utilizador. Usando protótipos ou modelos do sistema, potenciais erros de usabilidade são identificados e alvo de atenção durante o processo de desenvolvimento. Os protótipos/modelos permitem capturar os aspectos chaves da interacção dos utilizadores com as interfaces assim como os seus relacionamentos [19].

Os modelos de comportamento, em particular, podem ser obtidos através da implementação de um processo de engenharia reversa da interacção com o utilizador a partir do código fonte. Explorar a adopção da engenharia reversa de forma a extrair modelos da interface que esse código implementa representa um salto de abstracção maior que o necessário para obter um modelo do código. A engenharia reversa de sistemas interactivos pretende deduzir por análise do código, não quais os objectos presentes na interface (campos de texto, botões, etc.) e como estão organizados, mas antes que informação está presente na interface (o número, o nome, etc.) e as acções disponíveis na mesma (adicionar, consultar, etc.). O processo de engenharia reversa assenta num certo número de assunções e heurísticas que torna o processo viável. Estas abordam basicamente aspectos relacionados com a arquitectura da aplicação a reverter e aspectos relacionados com o nível de abstracção a utilizar. Vários autores investigaram a análise da usabilidade dos sistemas interactivos utilizando técnicas envolvendo o uso de engenharia reversa. Uma abordagem típica consiste em executar o sistema interactivo e automaticamente armazenar os seus estados e acções. Por exemplo, Memon [8] descreve uma ferramenta que extrai automaticamente dados relacionados com os componentes gráficos, as suas propriedades e valores. Uma alternativa que surge no âmbito da engenharia reversa de sistemas interactivos é a análise estática [4]. O processo baseia-se na análise do código de uma aplicação em vez da sua execução, tal como acontece no caso anterior.

O teste de implementações baseado em modelos testa se a aplicação vai de encontro ao modelo especificado. Este teste é relevante no contexto da engenharia de software, já que se está interessado em ter a certeza que a implementação não introduz problemas adicionais ao proposto na fase de *design* [16,17].

No contexto da manutenção e *redesign* pretende-se garantir que alterações ao sistema não criem efeitos inesperados. O teste de implementações baseado em modelos também é aplicada ao teste de interfaces com o utilizador (UI) de forma a testar se a interface desenvolvida vai de encontro ao modelo pretendido. Algumas abordagens foram efectuadas nesse sentido como é o caso do trabalho desenvolvido por Paiva [10,11]. No seu trabalho métodos de teste baseados em especificações formais foram utilizados de forma a sistematizar e automatizar o processo de teste de interfaces gráficas com o utilizador.

A abordagem descrita neste artigo incorpora modelos gerados a partir da análise do código fonte, servindo de base para uma ferramenta de teste de interfaces gráficas via modelos Spec# [1] e usando a ferramenta Spec Explorer [3].

II. TESTE DE SOFTWARE

A. *Uso de Modelos*

O teste de software é uma técnica que visa aumentar a confiança na exactidão do software desenvolvido. O teste de software baseado em modelos aumenta a sistematização e automatização, comparando o estado e comportamento de um produto de software com o seu modelo. Nessa comparação, são realizados testes através dos quais é possível averiguar se a aplicação desenvolvida vai de encontro ao seu modelo. Os modelos utilizados podem variar em termos de formalidade: formais, semi-formais ou informais [6]. Os modelos formais podem ser usados como oráculos de teste determinando-se se os resultados de casos de testes são correctos. Os casos de testes são sequências de possíveis acções com os respectivos parâmetros e resultados esperados. Podem ser gerados automaticamente os casos de teste a partir de um modelo, sendo estes executados sobre a aplicação e sobre o seu modelo. Ambos os resultados são comparados e caso surgem disparidades entre eles os potenciais problemas são identificados. Existem diversos exemplos de ferramentas baseadas em modelos para teste de APIs (Application Programming Interface) [3,5]. Estas têm de tratar dois problemas principais: controlar a explosão do número de estados e efectuar a ponte entre os modelos e as suas implementações. O problema da explosão do número de estados deve-se à enorme quantidade de estados que uma aplicação pode ter variando as entradas. A ordem de grandeza pode ser de tal ordem que se torna impraticável analisar todos os casos. Para solucionar tal situação são utilizados critérios de cobertura, com os quais se pretende identificar quais os testes adequados com um número de casos que torna o teste praticável.

B. *O Sistema de Programação Spec#*

O sistema de programação *Spec#* (Figura 1) é uma tentativa de desenvolver e manter de maneira eficaz *software* de alta qualidade [2] . Este foi desenvolvido pela *Microsoft Research lab* e é constituído por três partes, i.e. a linguagem de programação *Spec*#, o compilador e o verificador estático. A linguagem de programação *Spec#* é uma extensão à linguagem orientada aos objectos C#. Esta estende o C# introduzindo, tipos não nulos, excepções validadas e fornece ainda métodos na forma de pré-condições e pós-condições que funcionam como invariantes sobre os objectos.

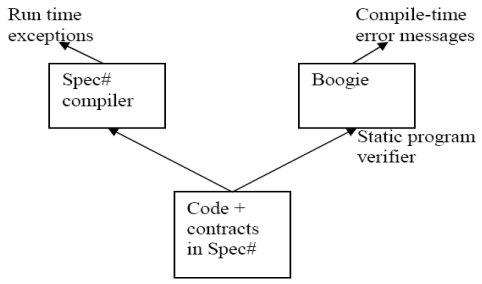

Figura 1: O sistema de programação Spec#

Um modelo Spec# descreve uma máquina de transição de estados possivelmente infinita, sendo a modelação dos seus estados efectuada através do uso de variáveis de estado.

C. *Modelos usando o Spec Explorer*

O *Spec Explorer* [3] é um software de modelação e teste proveniente da *Microsoft Research lab*. Esta ferramenta pode ajudar as pessoas que desenvolvem software a detectar erros nas fases de *design*, especificação e implementação das suas aplicações. Esta usa um modelo formal executável que pode ser escrito em *Spec#* ou *AsmL* (*Abstract State Machine Language*). O *Spec Explorer* permite codificar o comportamento pretendido de um sistema num modelo. O modelo permite representar os estados relevantes do sistema e identificar os comportamentos que uma execução correcta deve seguir. O objectivo consiste em determinar quais os passos executados pela aplicação, o que deve e o que não deve executar. O modelo permite também explorar as possíveis sequências de utilização de uma aplicação e comparar o comportamento de uma aplicação com o seu modelo. O *Spec Explorer* fornece um conjunto de características para lidar com a explosão de estados que pode resultar da exploração do modelo. Uma outra funcionalidade do *Spec Explorer* é a capacidade de gerar uma visualização gráfica da máquina de estados finita obtida pela exploração do oráculo em *Spec#* ou *AsmL*.

III. A ABORDAGEM IMPLEMENTADA

A. Extracção de Modelo de Diálogo

A abordagem seguida neste trabalho tem como ponto de partida código fonte de sistemas interactivos (e.g. *Java* ou *Haskell*). Tal como referido na secção I, existem dois objectivos essenciais a satisfazer: engenharia reversa do código para um nível adequado de abstracção, e modelação de interacção.

A extracção do modelo de uma interface a partir do código fonte foi implementada através da execução de travessias da árvore abstracta de sintaxe (AST) do programa em análise, obtendo como resultado uma sub-árvore com as instruções da interface. Esta abordagem permite ter em conta os aspectos de interesse, permitindo assim considerar unicamente as partes relevantes relacionadas com a camada de interacção com o utilizador [18]. O protótipo implementado, designado *GUIsurfer,* permite extrair e manipular todos os componentes gráficos da aplicação, por exemplo em *Java*, objectos das classes *JButton*, *Jlabel*, *JTextField*, *JProgressBar*, *JComboBox*, *JSlider*, etc. Após extracção dos dados dos objectos gráficos assim como todos os dados dos métodos sobre estes executados, é possível gerar representações abstractas do comportamento das interfaces através de máquinas de estados [22]. Uma máquina de estados permite abstrair o comportamento dos sistemas interactivos. Com esta representação pretende-se abstrair todos os estados gráficos de um sistema interactivo bem como as acções que interligam esses estados. Assim, a cada estado gráfico deduzido da árvore corresponde um estado na máquina de estados e cada acção sobre a aplicação dá origem a uma transição entre dois estados. Acrescentam-se também na máquina outros dados específicos do código fonte como por exemplo a inicialização de valores, as condições associadas às acções, etc. A primeira fase desta abordagem permite assim extrair modelos de diálogo.

B. Especificação do Modelo de Tarefas

Numa segunda fase, para efectuar os testes de *Graphical User Interfaces* (*GUIs*) é necessário codificar um oráculo num nível de abstracção baixo [7]. Sendo o processo moroso e caro, a abordagem definida efectua a geração automática desse oráculo com base num modelo de tarefas a definir manualmente com o apoio dos modelos de diálogo gerados pelo *GUIsurfer*. Usou-se a notação de modelação de tarefas *ConcurTaskTrees* (*CTT*) [13,15]. Esta fornece duas principais vantagens, i.e. os modelos estão ao mesmo nível de abstracção que os usados no *design* de interfaces, e o custo do desenvolvimento do oráculo é consideravelmente menor.

C. Construção Automática do Oráculo

A partir do modelo de tarefas *CTT* é possível definir uma especificação formal que represente uma aplicação interactiva. Uma ferramenta foi desenvolvida, designada TOM (task to oracle mapping) para a geração automática de oráculos a partir de modelos de tarefas [7]. Os oráculos descrevem as acções que o utilizador pode executar (carregar num botão, preencher um campo, etc.) e efeitos que tais acções podem desencadear no sistema em termos de estado da aplicação. Os efeitos de acções do utilizador não dependem exclusivamente do actual estado da aplicação mas também de condições de ambiente [7].

Os modelos são construídos seguindo as linhas de construção delineadas por Paiva [12] com as quais é possível construir-se um oráculo em *Spec#* para modelar sistemas interactivos. Sendo o nível de abstracção utilizado bastante baixo, a geração automática destes oráculos a partir de modelos de tarefas representa uma vantagem significativa [7].

D. Usando o Oráculo

Uma vez o oráculo completo, o *Spec Explorer* permite gerar uma máquina de estados finita a partir do modelo *Spec#* e gerar casos de testes. Três critérios de cobertura podem ser aplicados, i.e. *Full Transition Coverage* onde os casos de teste gerados cobrem todas as transições da máquina de estados finita, *Shortest Path* onde a sequência de casos de teste corresponde a uma sequência de transições para atingir determinado estado, e *Random Walk* com uma sequência de casos de teste obtida de forma aleatória.

A Figura 2 ilustra a abordagem seguida neste trabalho. O processo, tendo como ponto de partida o código fonte da aplicação em análise, envolve vários modelos, nomeadamente modelos de diálogo e de tarefas. Por fim é feito uso do *Spec Explorer* para geração e execução dos casos de teste.

Após a geração dos casos de testes, os mesmos são executados com o *Spec Explorer* para determinação das inconsistências e/ou conformidades entre a especificação e a aplicação em análise [12].

De forma a usar o oráculo gerado alguns ajustes devem ser feitos para adicionar mais alguma semântica que não tenha sido introduzida no modelo de tarefas. Quando o modelo *CTT* não descreve por completo o comportamento da GUI a ser testada, a pessoa que irá testar a aplicação tem neste passo a oportunidade de refinar manualmente o oráculo preliminar gerado. Por exemplo, o modelador pode optar por manter um modelo *CTT* com um baixo nível de detalhe e refinar mais tarde o modelo gerado em *Spec#*. Uma vez o oráculo pronto, casos de teste são gerados automaticamente e executados sobre a GUI a testar.

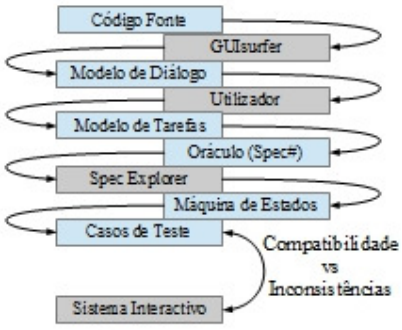

Figura 2: Geração de casos de teste baseado em modelos

IV. ANÁLISE DA ABORDAGEM

Diversos autores trabalharam na geração de sistemas interactivos a partir dos modelos, por exemplo *TERESA* [9] ou *Mobi-D* [14]. O trabalho desenvolvido concentra-se noutro sentido: testar os sistemas verificando se estes vão de encontro ao modelo gerado. Nesta abordagem o modelo gerado representa o comportamento do sistema implementado e não o comportamento previsto. Tendo em conta esta consideração esta abordagem permite executar testes de regressão já que permite comparar o comportamento do sistema, depois de alterado, com o modelo de como ele se comportava inicialmente. A execução de testes de regressão permite assim garantir que a alteração do sistema não afectou a sua estabilidade. Sendo assim, esta abordagem permite garantir que o sistema ainda contempla os requisitos iniciais.

Relativamente ao desenvolvimento destes modelos tornou-se claro que teriam de suportar um nível de detalhe complexo para o contexto de GUI. Modelar diálogos envolvendo *wizards* e janelas modais cria problemas consideráveis. De facto, os modelos pretendem capturar como os utilizadores interagem com o sistema para atingir determinado objectivo, e não capturem o desenho do sistema em si.

Os modelos de tarefas podem ser uma solução viável na geração de oráculos de teste. O que deve ser tido em consideração é que, estes oráculos testam somente um uso "normal" das aplicações. Por "normal" entende-se que os comportamentos esperados são resultado de utilizações de utilizadores típicos e que vão de encontro à aplicação que foi desenhada. Daí que estes oráculos são úteis numa primeira fase de teste onde o objectivo é testar se a aplicação cumpre as condições de utilização típica. Numa segunda fase, pretende-se testar a capacidade da GUI em lidar com erros do utilizador e/ou comportamento inesperados. Para tal, uma possibilidade é introduzir alterações no modelo de tarefas para permitir que comportamentos alternativos estejam reflectidos no modelo de tarefas. Alternativamente, um oráculo baseado em controlo de diálogos deverá ser usado.

V. Conclusões e Trabalho Futuro

O teste baseado em modelos pode ter um papel importante na garantia da qualidade e usabilidade dos sistemas interactivos. Apesar de não permitir fornecer uma avaliação correcta das respostas dos utilizadores ao sistema, pode ajudar na identificação de problemas que não foram detectados com outras técnicas de teste de sistemas interactivos. Isto é realizado através de um processo maioritariamente automatizado e sem o custo de testes completos de usabilidade. A técnica será relevante no contexto da engenharia, especificamente na manutenção e redesign. O teste baseado em modelos funciona comparando a execução de uma aplicação executável com um modelo que representa como o sistema se deve comportar.

A abordagem apresentada incide inicialmente na geração de modelos de diálogo a partir de código fonte, i.e. uso da ferramenta *GUisurfer*, sendo estes modelos utilizados como apoio na geração de modelos de tarefas. Finalmente o teste baseado em modelos é efectuado usando a ferramenta desenvolvida por Paiva [11] em conjunto com o *Spec Explorer*.

A abordagem apresentada permite gerar e executar casos de teste de forma semi-automatizada, mas não dispõe de informações que possam contribuir na automatização da análise dos resultados. Neste contexto, seria interessante que a abordagem implementasse funções que desempenhassem o papel de oráculos, apresentando informações do teste relativas ao cumprimento ou não dos requisitos de teste especificados.

Agradecimentos

Este trabalho é financiado por Fundos FEDER através do Programa Operacional Factores de Competitividade – COMPETE e por Fundos Nacionais através da FCT – Fundação para a Ciência e a Tecnologia no âmbito do projecto FCOMP-01-0124-FEDER-020554.

Referências Bibliográficas

[1] M. Barnett, R. DeLine, B. Jacobs, M. Fhndrich, K. R. M. Leino, W. Schulte, and H. Venter. The spec# programming system: Challenges and directions. In VSTTE2005 Verified Software: Theories, Tools, Experiments, ETH, Zurich, Switzerland, October 2005.

[2] M. Barnett, K. R. M. Leino, and W. Schulte. The Spec# programming system: An overview. In Gilles Barthe, Lilian Burdy, Marieke Huisman, Jean-Louis Lanet, and Traian Muntean, editors, Construction and Analysis of Safe, Secure, and Interoperable Smart Devices, volume 3362 of Lecture Notes in Computer Science, pages 49-69. Springer, 2005.

[3] C. Campbell, W. Grieskamp, L. Nachmanson, W. Schulte, N. Tillmann, and M. Veanes. Model-based testing of object-oriented reactive systems with Spec Explorer. Technical Report MSR-TR-2005-59, Microsoft Research, May 2005.

[4] Bruno d'Ausbourg, Guy Durrieu, and Pierre Roch_e. Deriving a formal model of an interactive system from its UIL description in order to verify and to test its behaviour. In F. Bodart and J. Vanderdonckt, editors, Design, Specification and Verification of Interactive Systems '96, Springer Computer Science, pages 105-122. Springer-Verlag/Wien, June 1996.

[5] A. Hartman and K. Nagin. The AGEDIS tools for model based testing. In Gregg Rothermel, editor, ISSTA '04: Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis, pages 129-132, New York, NY, USA, 2004. ACM Press.

[6] Alexander Pretschner. Model-based testing. In Proceedings of the 27th international conference on Software engineering (ICSE '05), 2005. ACM, New York, NY, USA, 722-723.

[7] José L. Silva, José Creissac Campos, and Ana C. R. Paiva. 2008. Model-based User Interface Testing With Spec Explorer and ConcurTaskTrees. *Electron. Notes Theor. Comput. Sci.* 208 (April 2008), 77-93.

[8] Atif M. Memon, Martha E. Pollack, and Mary Lou Soa. Automated test oracles for UIs. In David S. Rosenblum, editor, SIGSOFT '00/FSE-8: Proceedings of the 8th ACM SIGSOFT international symposium on Foundations of software engineering, pages 30-39, New York, NY, USA, 2000. ACM Press.

[9] G. Mori, F. Paternó and C. Santoro. Design and development of multi-device user interfaces through multiple logical descriptions. IEEE Transactions on Software Engineering, 30(8):507-520, August 2004.

[10] A. C. R. Paiva P. Automated Specification-Based Testing of Graphical User Interfaces. PhD thesis, Engineering Faculty of Porto University, Department of Electrical and Computer Engineering, 2007.

[11] Ana Cristina Paiva, João Pascoal Faria, Nikolai Tillmann, and Raul Moreira Vidal. A model-to-implementation mapping tool for automated model-based GUI testing. In Kung-Kiu Lau and Richard Banach, editors, Formal Methods and Software Engineering, volume 3785 of Lecture Notes in Computer Science, pages 450-464. Springer, 2005.

[12] Ana Cristina Paiva, Nikolai Tillmann, João Pascoal Faria, and Raul Moreira Vidal. Modeling and testing hierarchical guis. In Proceedings of the 12th International Workshop on Abstract State Machines, ASM'05, pages 329-344, 2005.

[13] F. Paternó. ConcurTaskTrees: An engineered notation for task models. In D. Diaper and N. Stanton, editors, The Handbook of Task Analysis for Human-Computer Interaction, chapter 24, pages 483-503. LEA, 2003.

[14] A. Puerta and J. Eisenstein. Towards a general computational framework for model-based interface development systems. In Proceedings ACM IUI99, pages 171-178. ACM Press, 1999.

[15] Clive Warren. The task oriented modelling (TOM) approach to the development of real-time safety-critical systems. In Stacey Ashlund, Kevin Mullet, Austin Henderson, Erik Hollnagel, and Ted White, editors, INTERACT '93 and CHI '93 conference companion, pages 167-168. ACM Press, 1993.

[16] Clayton Lewis, Peter Polson, Cathleen Wharton, and John Rieman. Testing a walkthrough methodology for theory-based design of walk-up-and-use interfaces. In CHI '90 Proceedings, pages 235–242, New York, April 1990. ACM Press.

[17] Jakob Nielsen and Rolf Molich. Heuristic evaluation of user interfaces. In CHI '90 Proceedings, pages 249–256, New York, April 1990. ACM Press.

[18] João Carlos Silva, Carlos Silva, Rui D. Gonçalo, João Saraiva, and José Creissac Campos. The GUISurfer tool: towards a language independent approach to reverse engineering GUI code. In Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems (EICS'10). ACM, New York, NY, USA, 181-186.

[19] Fabio Paterno. Model-Based Design and Evaluation of Interactive Applications (1st ed.), 1999. Springer-Verlag, London, UK.