Experiência no Uso de Ferramentas Livres para o Teste Funcional de Componentes de Software[1]

Wilkerson de Lucena Andrade, Helton Souza Lima, Daniel Lima Barbosa, Patrícia D. L. Machado, Jorge C. A. Figueiredo

Universidade Federal de Campina Grande – Departamento de Sistemas e Computação
58109-970 – Campina Grande – PB – Brasil

{wilker,helton,daniel,patricia,abrantes}@dsc.ufcg.edu.br

Abstract. *We describe the experience of using Free Software to automate functional testing activities for software components in the SPACES tool. We present the free tools used, focusing on problems faced and solutions adopted. Furthermore, we analyze the advantages gained by the use of free software in the design of the SPACES tool.*

Resumo. *Descrevemos a experiência obtida no uso de Software Livre para auxiliar a automação das atividades de teste funcional de componentes de software na Ferramenta SPACES. Serão apresentadas as ferramentas livres utilizadas, abordando os problemas encontrados e as soluções adotadas. Por fim, fazemos uma análise das vantagens ocasionadas pela utilização de Software Livre no projeto da ferramenta SPACES.*

1. Introdução

Teste é uma atividade fundamental no processo de desenvolvimento e validação de software. Sua execução normalmente envolve altos custos e pode ser dificultada pela ausência de técnicas e ferramentas adequadas [Jorgensen 2002]. Neste sentido, automação tem se tornado um requisito indispensável a sua adoção. A ferramenta SPACES (*SPecification bAsed Component tESter*) [Barbosa el al 2004A, Barbosa et al 2004B] foi desenvolvida com o objetivo de dar suporte ao teste funcional de componentes de software. Trata-se de uma ferramenta de teste funcional, desenvolvida em Java, para componentes que faz uso de modelos na notação UML (*Unified Modeling Language*) e especificações em OCL (*Object Constraint Language*) para derivar casos de teste de forma automática.

Entre as características gerais de SPACES, destacam-se: integração com ferramentas de modelagem UML que exportem para o formato XMI (*XML Metadata Interchange*), para troca de informações; arquitetura adaptável para utilização de diferentes linguagens de especificação de restrições tais como OCL, assim como, a geração de código de teste para diferentes linguagens de programação e/ou plataformas; possibilidade de re-execução dos casos de teste gerados para novos dados fornecidos dinamicamente. Devido à complexidade inerente ao desenvolvimento de ferramentas deste tipo, reuso de software passa a ser um aspecto importante para maximizar a viabilidade de sua construção. É importante também mencionar o requisito de confiabilidade associado ao código final de teste gerado.

[1] Este trabalho é parte do Projeto Móbile, financiado pelo CNPq (Proc. 552190/2002-0) e do Projeto MOBILE-TEST, financiado pela FAPESQ/CNPq (Projeto 060/03).

O objetivo deste artigo é relatar a experiência com o uso de Software Livre no projeto de SPACES, destacando os problemas enfrentados, as soluções adotadas e os benefícios obtidos. Esperamos com isto contribuir para a disseminação do uso de Software Livre, em particular, das ferramentas adotadas, no desenvolvimento de projetos similares. Demonstrações do uso de Software Livre em projetos existentes têm se mostrado de grande importância na adoção da cultura de Software Livre por desenvolvedores [Robins 2003]. As seções seguintes apresentam as ferramentas livres abordadas e seu uso em SPACES.

2. Ferramentas Livres

Software Livre pode apresentar uma série de vantagens sobre o Software Proprietário [Fuggetta 2005, Robins 2003]. De acordo com a *Free Software Foundation* (FSF)[2], uma das vantagens mais importantes é a disponibilidade do código fonte que traz como conseqüência outras vantagens: (i) A comunidade de desenvolvimento fica espalhada pelo mundo e coopera através da Internet, contribuindo para o desenvolvimento de software com boa qualidade; (ii) A robustez e confiabilidade do Software Livre provocam reduções significativas em custos operacionais e a disponibilidade do código fonte permite que os sistemas sejam adaptados às condições e necessidades dos usuários; (iii) Transparência na codificação das informações tratadas pelos programas.

Existem ferramentas livres para praticamente todos os usos possíveis e algumas destas podem ser utilizadas para promover o teste funcional. A seguir, descreveremos as ferramentas livres que foram utilizadas no desenvolvimento da ferramenta SPACES.

2.1. Dresden OCL

Dresden OCL [Hussmann 2000] é um conjunto de módulos de suporte à análise e ao processamento de restrições OCL. Entre os módulos disponíveis estão: (1) Um *Parser*, gerado pela ferramenta SableCC[3], a partir da especificação 1.3 da linguagem OCL, responsável pela análise léxica e sintática das restrições; (2) Um *Analisador Semântico*, que realiza verificações de consistência e de tipos de acordo com o modelo UML; (3) Um *Normalizador*, que simplifica as restrições OCL para facilitar a geração de código; (4) Um *Gerador de Código*, responsável pela geração de código fonte Java para ser usado em instrumentação (inserção de código de teste na própria implementação a ser testada). Este código é baseado em uma biblioteca de classes própria que representa o sistema de tipos OCL. Cada módulo implementa interfaces bem definidas, o que torna possível a redefinição de suas funcionalidades de acordo com as necessidades dos usuários. A comunicação entre os módulos se dá a partir de uma estrutura de dados denominada *Árvore Sintática Abstrata* que é construída durante o parser.

Dresden OCL é licenciado segundo a *GNU Lesser General Public License* (Licença LGPL), que está hospedado em [Open Source Technology Group], um conhecido repositório de Software Livre.

2.2. JUnit

A ferramenta JUnit [Beck & Gamma] é um *framework* bastante conhecido para testes de unidade automáticos em código Java, e que funciona da seguinte forma: ao

[2] http://www.fsf.org/
[3] http://www.sablecc.org

escrevermos o código de teste, que executa partes do código do programa principal (programa que se quer testar), informamos ao *framework* os resultados esperados provenientes dessa execução. Esses resultados podem ser informados de várias formas, como, por exemplo, sobre valores esperados para atributos, valores de retorno de determinados métodos, lançamento de exceções, e estados de objetos. Ao executar o teste, o *framework* compara os resultados esperados com os resultados obtidos e, ao finalizar a execução, lista todas as eventuais disparidades que ocorreram. JUnit é licenciado de acordo com a *Common Public License*, versão 1.0, e está hospedado em [Open Source Technology Group].

2.3. JTestCase

A ferramenta JTestCase [Kölle & Wang] é um *framework* para teste de software escrito em Java, que faz a separação entre os dados do teste e o código do teste. Dessa forma, os dados do teste podem ser modificados sem a necessidade da alteração e compilação do código do teste e os casos de teste ficam mais flexíveis. Através de arquivos bem definidos no formato XML, a API de JTestCase permite um fácil acesso aos dados dentro do próprio código de teste. JTestCase tem fácil integração com códigos de teste baseados no *framework* JUnit, pois foi desenvolvido com este propósito, apesar de não estarem diretamente associados. JTestCase é licenciado de acordo com a *Common Public License*, versão 0.5, e está hospedado em [Open Source Technology Group].

3. A Ferramenta SPACES

A Figura 1 apresenta a arquitetura da ferramenta SPACES. Como se pode observar, os artefatos UML necessários à geração de casos de teste são: Diagrama de Casos de Uso; Diagrama de Classes; Diagramas de Seqüência; e Restrições OCL.

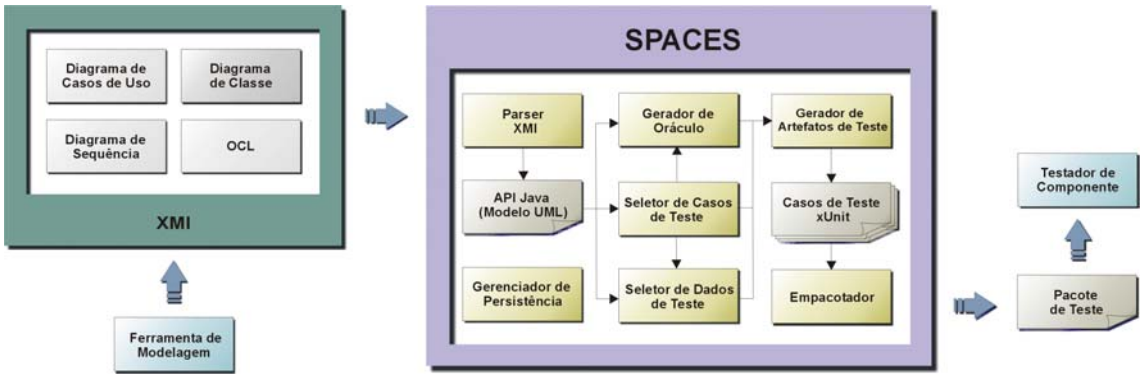

Figura 1: Arquitetura da Ferramenta SPACES.

SPACES possui os seguintes módulos: Parser XMI, Seletor de Casos de Testes, Gerador de Oráculo, Seletor de Dados de Testes, Gerador de Artefatos de Teste e Empacotador. Além disso, temos o Testador de Componente, que é responsável pela execução dos testes.

De uma forma geral, o funcionamento da ferramenta pode ser descrito como a seguir. Inicialmente, a especificação UML, em formato XMI, é processada pelo *parser* que instancia objetos correspondentes às informações do modelo UML (Diagramas, Classes, Operações, Cláusulas OCL, etc.). Essas informações são processadas pelos módulos: Seletor de Casos de Teste, Gerador de Oráculo e Seletor de Dados de Teste.

As informações obtidas (Casos de Teste, Oráculos e Dados) são então utilizadas pelo módulo Gerador de Artefatos de Teste, que se encarrega de gerar o código de teste, os arquivos com os dados do teste e os arquivos de configuração do teste. O código resultante está de acordo com o *framework* de teste JUnit e com a linguagem de programação Java. Ao final do processo, os artefatos do teste, constituídos de classes de teste e arquivos de dados, são empacotados para serem executados no Testador de Componente, que é então distribuído juntamente com o sistema, possibilitando, além da execução dos casos de teste, a alteração dos dados utilizados no teste.

4. Utilização de Software Livre no Desenvolvimento de SPACES

As ferramentas livres apresentadas na Seção 2 auxiliaram no desenvolvimento de alguns módulos de SPACES (Gerador de Oráculo e Gerador de Artefatos de Teste) e do Testador de Componente. A seguir, descrevemos como essas ferramentas foram utilizadas e analisamos a experiência de uso sob aspectos como a dificuldade de utilização, problemas encontrados e soluções adotadas.

4.1 Gerador de Oráculo

Após a definição do conjunto de casos de teste selecionados para as funcionalidades do componente, é preciso gerar oráculos que sejam responsáveis pelo veredicto destes casos de teste. Esta funcionalidade consiste na derivação de fragmentos de código a partir de restrições OCL, invariantes, pré e pós-condições. Estes fragmentos são responsáveis, em tempo de execução, pela determinação do resultado de cada teste.

Para proceder tal geração era imprescindível a precisão e a correção, tanto sintática quanto semântica, das restrições OCL. Dessa forma o gerador precisava prover meios de detectar antecipadamente qualquer problema existente nas restrições. Em função disso, optamos pela utilização da ferramenta Dresden OCL.

Para o correto funcionamento, a ferramenta Dresden OCL precisa acessar, além das restrições OCL, o modelo UML sob o qual foram especificadas as restrições. Isso é feito a partir da implementação de uma interface denominada *ModelFacade*. Esta interface define a operação *getClassifier* que recebe o nome do classificador e, caso este exista no modelo UML, retorna um objeto correspondente do tipo *Any* da biblioteca de classes da ferramenta. Assim sendo, para possibilitar a integração desta ferramenta com SPACES foi preciso criar uma implementação da interface *ModelFacade* que permitisse o acesso às informações do modelo UML usado por SPACES. Esta classe, denominada *SPACESModelFacade*, assim como as interfaces que constituem o mecanismo de acesso ao modelo UML da ferramenta Dresden OCL, pode ser vista na Figura 2.

A partir da implementação da interface *ModelFacade* pela classe *SPACESModelFacade*, e o conseqüente acesso da ferramenta Dresden OCL às informações do modelo UML contidas nos objetos da API UML, SPACES passou a se beneficiar das funcionalidades de verificação (sintática e semântica) e de normalização de restrições OCL, implementadas pelos módulos desta ferramenta. Entretanto, não foi possível utilizar o módulo Gerador de Código da ferramenta para gerar os oráculos de teste, visto que o módulo gera código apenas para instrumentação. A solução adotada foi a redefinição do módulo Gerador de Código da ferramenta Dresden OCL dentro do módulo Gerador de Oráculos da ferramenta SPACES.

De uma forma geral, o processo de integração de SPACES com o Dresden OCL

foi realizado sem problemas devido à existência de uma boa documentação da segunda.

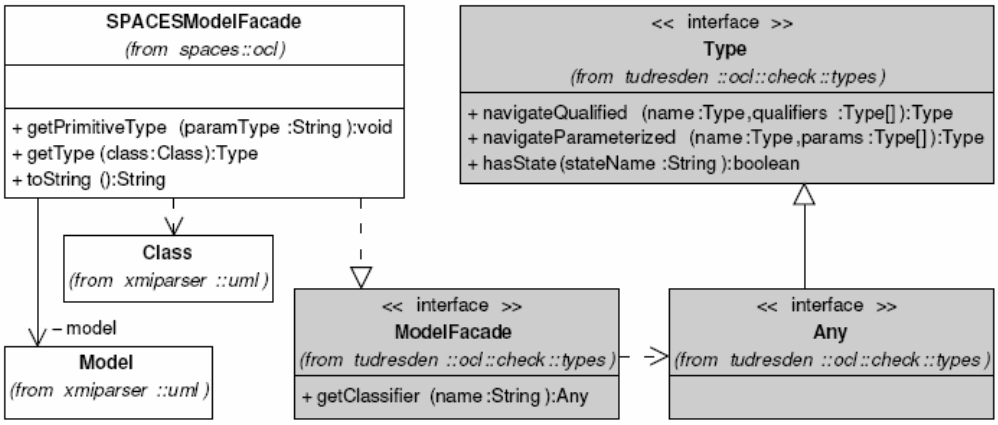

Figura 2: Integração Dresden OCL - SPACES: Acesso ao Modelo UML.

4.2 Gerador de Artefatos de Teste

A execução do teste de componentes demanda a seleção de dados que são usados como entrada, a fim de instanciar variáveis e construir objetos do programa a se testar. SPACES faz a seleção automática de dados de teste, porém, como esta é uma tarefa que continua sendo um dos maiores problemas enfrentados por testadores e ferramentas de teste, é disponibilizado ao usuário a possibilidade de revisão, alteração e adição no conjuntos de dados. Para isso, precisamos que o código de teste acesse o conjunto de dados de teste localizados em arquivos distintos, a fim de podermos alterar os dados sem alterar o código, e não necessitarmos de uma nova compilação a cada alteração.

Essa funcionalidade é oferecida por JTestCase, que abstrai, para o usuário, todo o trabalho de leitura e conversão de conjunto de caracteres para os diversos tipos de variáveis que utilizamos no código de teste. Então, o Gerador de Artefatos de Teste apenas faz a geração dos arquivos XML no formato padrão do JTestCase e adiciona, no código de teste, o acesso aos dados que estão no XML através da API do JTestCase.

A ferramenta JTestCase nos ajudou bastante nessa etapa, pois, basicamente, só precisamos nos familiarizar com sua API e o formato padrão de seus arquivos, ganhando um tempo razoável no desenvolvimento de SPACES. Tivemos algumas dificuldades ao iniciarmos o manuseio com a ferramenta, entretanto, essas dificuldades foram sanadas com o auxílio dos próprios desenvolvedores do JTestCase, através do fórum de ajuda da ferramenta, disponibilizado em [Open Source Technology Group].

4.3 Testador de Componente

Como o código de teste gerado por SPACES está de acordo com o *framework* JUnit, necessita-se de uma integração do código com o *framework* para que se possa executar o teste. Se optássemos por disponibilizar ao usuário apenas o pacote de teste gerado por SPACES e o JUnit, iríamos atribuir ao cliente a responsabilidade de realizar a integração necessária à execução do teste. Além disso, a alteração dos dados no arquivo XML ficaria complicada. Por isso, resolvemos alterar o código do JUnit para atender nossas necessidades específicas. Como o JUnit possui três interfaces com o usuário, escolhemos uma delas (a gráfica) para servir de alvo às nossas modificações.

Para resolver o problema da alteração dos dados, incrementamos na interface

gráfica do JUnit, um botão que abre uma tabela que contém os conjuntos de dados permitindo editá-los, removê-los ou adicionar novos conjuntos de dados. Ao fechar a janela, o usuário escolhe a opção de salvar os dados ou de cancelar suas alterações e volta à tela principal do JUnit, podendo já executar os testes com os novos dados.

5. Considerações Finais

Durante o desenvolvimento da ferramenta SPACES, lidamos com duas situações: a reutilização de software de forma gratuita e a necessidade de acesso ao código-fonte para integração com SPACES. Neste contexto, Software Livre surgiu como a única alternativa capaz de suprir essas necessidades. Ao fazer uso de Software Livre na implementação das funcionalidades de SPACES, conseguimos agilizar o desenvolvimento e, para cada dificuldade encontrada, adotamos soluções e/ou propomos adaptações para a comunidade de desenvolvedores, contribuindo para o aperfeiçoamento das próprias ferramentas.

SPACES encontra-se em fase final de desenvolvimento e sua primeira versão será disponibilizada como Software Livre sobre uma licença a ser definida em acordo com as licenças das ferramentas usadas. Como Software Livre, SPACES irá adquirir rapidamente um bom nível de qualidade devido ao *feedback* da comunidade de usuários e desenvolvedores. Por fim, esperamos que uma ferramenta livre de geração e execução de teste para componentes de software a partir de especificações UML contribua para que desenvolvedores e empresas não habituados a outras especificações menos usuais, tenham acesso a mecanismos eficientes para melhorar a qualidade de seus produtos de desenvolvimento sem a necessidade de aquisição de licenças de softwares proprietários.

Referências Bibliográficas

Barbosa, D. L.; Andrade, W. L.; Machado, P. D. L.; Figueiredo, J. C. A. (2004A) "Um Método Automático para Verificação Funcional de Componentes". Em Anais do **IV Workshop de Desenv. Baseado em Componentes** (WDBC), 23-28.

Barbosa, D. L.; Andrade, W. L.; Machado, P. D. L.; Figueiredo, J. C. A. (2004B) "SPACES - Uma Ferramenta para Teste Funcional de Componentes". Em **Sessão de Ferramentas do XVIII SBES**, Brasília, 55-61 (Prêmio de Melhor Ferramenta).

Fuggetta, A., (2004) "Open Source and Free Software: A New Model for The Software Development Process?", Software Process Technology, Vol V, N. 5.

Beck, K., Gamma, E. **JUnit** . Em http://junit.org. Último acesso em 11/042005.

Hussmann, H.; Demuth, B.; Finger, F. (2000) "Modular Architecture for a Toolset Supporting OCL". Em Proceedings of **UML 2000 - The Unified Modeling Language**. Advancing the Standard. Third Int. Conf., pages 278- 293, York, UK.

Jorgensen, P. C. **Software Testing: A Craftsman's Approach**. CRC, 2002.

Kölle, C.; Wang, Y. **JTestCase**. Em http://jtestcase.sourceforge.net. Último acesso em 11/04/2005.

Open Source Technology Group. ***Source Forge***. Em http://sourceforge.net. Último acesso em 11/04/2005.

Robins, J. (2003), "Adopting Open Source Software Engineering (OSSE) Practices by Adopting OSSE Tools", In: Perspectives on Open Source and Free Software, Ed by J. Feller, B. Fitzgerald, S. Hissan & K. Lakham, O'Reilly & Associates.