Ferramenta de Apoio ao Teste de Aplicações Java baseada em Reflexão Computacional

Fábio Fagundes Silveira [1]
Ana Maria de Alencar Price [2]

[1, 2] Universidade Federal do Rio Grande do Sul - Instituto de Informática
Av. Bento Gonçalves, 9500 - Campus do Vale - Bloco IV
Porto Alegre - RS - Brasil
CEP 91501-970 Caixa Postal: 15064
+55 (51) 3316-6159 Fax: +55 (51) 3316-7308
{ffs,anaprice}@inf.ufrgs.br

RESUMO

A atividade de teste constitui uma fase de grande importância no processo de desenvolvimento de software, tendo como objetivo garantir um alto grau de confiabilidade nos produtos desenvolvidos. Com o advento do paradigma da orientação a objetos, novos problemas foram introduzidos na atividade de teste de programas, tornando-a mais complexa do que para sistemas tradicionais. Com o objetivo de auxiliar o processo de teste de programas orientados a objetos, este trabalho aborda o desenvolvimento de uma ferramenta, para programas escritos em Java, orientada ao teste de estados com apoio da tecnologia de reflexão computacional. Através do emprego de asserções, especificadas pelo usuário, é possível verificar a integridade dos estados dos objetos durante a execução do programa em teste.

ABSTRACT

Software testing is a very important step in the software development cycle, whose goal is the obtainment of systems with a high degree of reliability. With the advent of the object oriented paradigm, new problems have been introduced in the software testing activity, by making it more complex than the traditional procedural validation. This paper focuses on the development of a tool to support the testing process of object oriented programs. KTest is oriented to state-based testing of Java-written programs, supported by the mechanism of computational reflection. By evaluating user specified assertions, KTest verifies the state integrity of the objects during the execution of the program being tested.

Keywords: *Object Oriented Software Testing, Computational Reflection, Computational Reflection Protocol, Java*

1. INTRODUÇÃO

O processo de teste de *software* abrange uma série de atividades realizadas com o objetivo de garantir a maior qualidade possível no desenvolvimento deste tipo de produto. Uma das razões pelo qual o processo de teste tem ganhado significativa importância, é o fato deste consumir até 40% do esforço despendido no desenvolvimento de *software* [PRE95].

O objetivo do paradigma da orientação a objetos (OO) é o de aumentar a produtividade e qualidade das aplicações, e sobretudo reduzir a ocorrência de determinados tipos de erros. Entretanto, apesar destas vantagens, este paradigma apresenta algumas características que, ao

mesmo tempo em que constituem aspectos positivos ao processo de desenvolvimento de software, caracterizam-se como fatores que tornam a atividade de teste e manutenção mais complexas que no paradigma procedimental. Entre essas características, citam-se a herança, o encapsulamento e o polimorfismo.

Com a finalidade de auxiliar a atividade de teste, utilizando especificamente a proposta de teste baseado em estados, modelou-se uma ferramenta que utiliza a tecnologia da reflexão computacional para teste de programas orientados a objetos escritos na linguagem Java. Esta ferramenta por sua vez, permite que se analise uma aplicação de forma dinâmica, sem a necessidade de instrumentar o código-fonte do programa em teste. Através da reflexão, é possível monitorar classes e objetos específicos, realizando uma intervenção na computação da aplicação escolhida para esse fim.

Trabalhos correlatos ao desenvolvido, são apresentados por Campo [CAM97], Pinto [PIN98] e Palavro [PAL00], implementados na linguagem Smalltalk. A ferramenta *ATeste* [PIN98], utiliza abordagem reflexiva para a realização de teste de estados, utilizando uma estratégia de teste denominada teste dinâmico de caminho em aplicações desenvolvidas na linguagem Smalltalk. *FATOO* [PAL00] estende *ATeste*, aceitando pré, pós-condições de métodos e invariantes de classes, gerando diagramas de eventos associados à execução da aplicação e informações relevantes para a aplicação de teste de regressão. Ambas utilizam o *framework LuthierMOPs* [CAM97], responsável pelo suporte à reflexão computacional, o qual permite monitorar a execução de frameworks OO.

2. TESTE DE SOFTWARE OO

O paradigma da Orientação a Objetos (OO) surgiu trazendo consigo um novo enfoque, comparado aos métodos tradicionais de desenvolvimento de *software*. Entre as vantagens desta abordagem, pode-se citar a adoção de formas mais próximas aos mecanismos humanos com relação ao gerenciamento de complexidades inerentes ao desenvolvimento de produtos de *software*, buscando com isso um aumento de qualidade e maior produtividade, devido a uma de suas principais contribuições: a reutilização de código. Essa contribuição, entretanto, enfatiza que, assegurar que as classes desenvolvidas estejam corretas é essencial, pois erros podem propagar-se durante a reutilização de classes por suas subclasses.

Apesar da abordagem OO apresentar várias vantagens em relação ao paradigma procedimental, a atividade de teste constitui um dos principais problemas no desenvolvimento de aplicações OO. Existe a carência de técnicas bem estabelecidas para o teste de aplicações desenvolvidas sobre este paradigma, constituindo-se numa área nova de pesquisa e aplicação.

Do mesmo modo em que algumas das características encontradas em linguagens orientadas a objetos reduzem a probabilidade de determinados erros, outras favorecem o aparecimento de novas categorias dos mesmos [BIN95]. Entre as características favorecedoras, cita-se o encapsulamento, o polimorfismo e a ligação dinâmica.

Algumas facilidades do teste de *software* OO em relação ao procedimental são apresentadas por McGregor [MCG96]: i) métodos e *interfaces* de classes são explicitamente definidos; ii) número menor de casos de testes para cobertura são resultantes, devido ao número reduzido de parâmetros e iii) reutilização de casos de teste devido à presença da característica de herança. Porém, algumas desvantagens também devem ser consideradas [MCG96]: i) a avaliação da correteza da classe é dificultada pela presença do encapsulamento de informações; ii) o gerenciamento do teste é dificultado pelos múltiplos pontos de entrada (métodos) de uma classe e iii) as interações entre os objetos são expandidas pelo polimorfismo e pela ligação dinâmica.

O teste de *software* OO baseado em estados avalia as mudanças de estados sofridos pelos objetos de determinada(s) classe(s). Este teste é baseado no modelo dinâmico da classe [RUM94] (diagrama / máquina de estados), a qual é formada por estados, pré e pós-condições associadas e

transições, que são definidas como execução dos métodos, sendo este o teste utilizado pela ferramenta desenvolvida.

3. REFLEXÃO COMPUTACIONAL

Patti Maes [MAE87] definiu o conceito de reflexão computacional como sendo a atividade executada por um sistema computacional quando faz computações sobre (e possivelmente afetando) suas próprias computações. Desta maneira, reflexão pode ser entendida como uma forma de introspecção, pois o sistema pode tentar tirar conclusões sobre suas próprias computações, podendo estas serem posteriormente afetadas.

Conforme descrito por Rubira [RUB98], o objetivo da reflexão não se refere ao auxílio de atividades referentes ao domínio externo das aplicações, e sim na contribuição para sua organização interna bem como *interface* com o mundo externo. Disso resulta que o uso de reflexão é útil em atividades administrativas da aplicação, tais como estatísticas de desempenho, otimização, distribuição, tolerância a falhas e, evidentemente no processo de teste e depuração de *software*.

A arquitetura reflexiva é composta por dois níveis, um denominado meta-nível e outro denominado nível base [LIS98]. No meta-nível estão as estruturas de dados bem como ações a serem executadas sobre o sistema objeto presente no nível base. As computações realizadas no meta-nível são feitas sobre dados que representam informações para o programa de nível base, o qual realiza computações sobre seus dados, atendendo desta forma aos requisitos da aplicação (domínio externo).

A computação reflexiva, no modelo de objetos, pode ser realizada sobre classes ou objetos. Quando a reflexão é realizada sobre classes (denominado modelo de meta-classes), o meta-nível é composto por meta-classes, contendo estas informações estruturais sobre os componentes do nível base. Segundo Lisboa [LIS98], este modelo apresenta menor flexibilidade, pois o mesmo meta-objeto é compartilhado por todos os objetos de uma classe. No segundo caso, (denominado modelo de meta-objetos), o meta-nível é composto por meta-objetos, contendo estes informações (descrições) relacionadas ao comportamento dos componentes do nível base. Neste modelo a flexibilidade é maior, pois o meta-objeto possuirá as informações de um objeto específico.

No modelo de reflexão de meta-classes, ocorre a reflexão estrutural, a qual permite que se obtenha informações (permitindo também alterações) sobre a estrutura de determinadas classes. Entre as informações e alterações suportadas pela reflexão estrutural estão: obter subclasses, superclasses, atributos e métodos de uma classe, *interfaces* de uma classe, alterar o comportamento de classes, além de criar novas classes e redefinir classes existentes, bem como eliminá-las. Já o modelo de reflexão de meta-objetos utiliza a Reflexão Comportamental, permitindo que um meta-objeto interfira no comportamento de um objeto. Segundo Lisboa [LIS98], a reflexão comportamental de um objeto consiste na atividade realizada pelo seu meta-objeto, visando obter informações e realizar transformações sobre o comportamento do objeto. Através da busca e coleta destas informações sobre o processo de execução, pode-se obter: estatísticas de desempenho, informações para fins de depuração e monitoração, entre outras.

4. USO DA REFLEXÃO COMPUTACIONAL NO TESTE DE SOFTWARE

A utilização da reflexão computacional no processo de teste possibilita analisar a aplicação de forma dinâmica, não sendo necessária a instrumentação do código-fonte da mesma. Através da reflexão, é possível monitorar classes e objetos específicos, realizando uma intervenção na computação da aplicação em teste, constituindo uma técnica bastante flexível.

Este monitoramento é feito através da interceptação na computação da aplicação, com o intuito de examinar objetos selecionados, os quais podem ser indicados pelo usuário através do uso de uma ferramenta de teste, em tempo de execução.

Implementações de técnicas de teste de software OO tem sido desenvolvidas com apoio de reflexão, tendo como base a utilização de invariantes associadas a classes, bem como pré e pós condições associadas aos métodos. Através da utilização de protocolos de reflexão que suportem o modelo de reflexão comportamental, ou seja, que permitam que se realize interceptação de mensagens, estas técnicas poderiam ser utilizadas. Assim, seria possível verificar a integridade dos objetos, consultando os valores dos seus atributos e, analisando se os estados dos mesmos são válidos, de acordo com pré e pós condições especificadas pelo usuário em tempo de execução.

5. PROTOCOLO REFLEXIVO GUARANÁ

O Guaraná consiste numa arquitetura reflexiva, no qual seu protocolo de meta-objetos permite a reutilização do código do meta-nível através da composição de meta-objetos [OLI98]. Este protocolo possibilita a composição dinâmica de meta-objetos, na qual permite uma maneira mais simples de composição destes objetos, tornando possível sua reconfiguração dinâmica.

O protocolo Guaraná foi implementado através da modificação da *Kaffe OpenVM*, uma implementação de domínio público da especificação da JVM padrão [OLI98]. Segundo Oliva [OLI98], a maior parte do protocolo é codificado em Java, tendo sua máquina virtual sofrido modificações localizadas, com o objetivo de prover operações de interceptação, materialização e criação de operações. Entretanto, a linguagem de programação Java não foi modificada: qualquer programa em Java, compilado por qualquer compilador Java, poderá ser executado nesta implementação, sendo ainda possível, estender suas capacidades através do uso de reflexão. As alterações no interpretador, permitem: a) interceptação de operações, como invocação de métodos; b) leitura e escrita em variáveis, bem como em elementos de *arrays*; c) criação de objetos e *arrays* e d) entrada/saída de monitores.

Em Java 2, a API de reflexão permite a um objeto realizar somente as operações concedidas a ele, sendo estas diretamente no código fonte, isto é, o controle de acesso é baseado em permissões da classe. O protocolo Guaraná incrementa esta característica [OLI98b], introduzindo mecanismos de interceptação que não estão presentes em Java, além de mecanismo de segurança por objeto (contrário ao de classes), resultando que meta-objetos podem obter privilégio de acesso para objetos que eles controlam.

O *kernel* do protocolo Guaraná constitui a base de sua arquitetura, tendo como funções básicas a realização das seguintes tarefas [OLI98]: (i) operações de interceptações e reificações; (ii) ligação dinâmica e invocação para objetos do meta-nível e (iii) manutenção da meta-informação estrutural.

5.1. Meta-Objetos do Protocolo

De acordo com Oliva [OLI98], cada objeto pode estar diretamente associado com zero ou um meta-objeto, chamado de meta-objeto primário. Seu papel é observar todas as operações endereçadas ao seu objeto associado (aqui denominado para-objeto), bem como seus resultados. Tais funções são garantidas pelos mecanismos de interceptação e reificação implementados no *kernel* do protocolo. É possível também, que a classe esteja associada a um meta-objeto primário, no qual observará todas as operações referentes à classe associada e não com suas instâncias, resultando disto que, os meta-objetos das classes e suas instâncias são independentes. Desta maneira, não serão interceptadas operações atribuídas a instâncias que não estejam ligadas a este meta-objeto [OLI98b].

Três possíveis possibilidades são retornadas pelo meta-objeto primário ao *kernel*, após suas operações de inspeção e reflexão sobre seus conteúdos [OLI98b]:

a) um resultado: considerado pelo *kernel* como se fosse produzido pela execução da operação atual;
b) uma operação de substituição: o *kernel* irá repassar ao para-objeto, desconsiderando o original;

c) nenhuma das anteriores: o *kernel* devolverá a operação original para o objeto da aplicação.

Nas alternativas “a” e “b”, onde o meta-objeto não fornece resultado, este poderá sinalizar ao *kernel* que pretende inspecionar ou alterar o resultado da operação. Diante desta situação, após a realização da operação, o *kernel* reificará o resultado e apresentará ao meta-objeto primário, sendo possível neste ponto, para este, realizar qualquer operação apropriada. Cabe ressaltar que, o *kernel* somente aceitará este resultado modificado se o meta-objeto tiver indicado que ele poderia modificá-lo.

Assim que uma aplicação é inicializada, todos os objetos possuem uma meta-configuração vazia, isto é, nenhum objeto está passível de reflexão. A aplicação é que pode criar objetos e meta-objetos e então associá-los [OLI98]. O protocolo Guaraná foi desenvolvido através de um pacote denominado *BR.unicamp.guarana*. Uma completa referência sobre a hierarquia das classes pode ser encontrada em [OLI98] [SEN01].

6. A FERRAMENTA KTEST

A ferramenta KTest objetiva fornecer apoio às atividades de teste e validação de aplicações escritas na linguagem Java, dando suporte ao teste de *software* baseado em estados. Utiliza para tanto, a tecnologia da Reflexão Computacional para fazer a análise dos estados dos objetos de forma dinâmica, ou seja, durante a execução da aplicação em teste.

Através da especificação de asserções, feitas pelo usuário (testador) da ferramenta, na forma de invariantes de classe, pré e pós-condições, é possível verificar os estados dos objetos da aplicação em teste.

A KTest, que encontra-se em fase de conclusão, está sendo desenvolvida na linguagem Java, utilizando-se o Protocolo Reflexivo Guaraná versão 1.6 [OLI98]. Esta versão do protocolo está implementada na JVM *Kaffe OpenVM* 1.0.5, sendo um híbrido entre o versão 1.1 e 1.2 do JDK.

A escolha desse protocolo deu-se por ele apresentar características benéficas ao teste e depuração de programas em tempo de execução. É possível realizar testes de “caixa-preta” e, até certo ponto, teste de “caixa-branca”, visto que este protocolo, com modelo de reflexão por meta-objetos, permite que seja aberta a implementação dos objetos, sendo possível a realização de injeção de falhas para posterior análise. Outra vantagem do protocolo Guaraná é a capacidade de poder projetar diversos meta-objetos de testes distintos, combinando-os de forma a compô-los numa única aplicação, fazendo estes operarem sobre um mesmo objeto ou um mesmo grupo de objetos. Desta forma, vários tipos de teste podem ser aplicados sobre um ou mais objetos selecionados. Como neste protocolo, a introdução de novos meta-objetos é totalmente realizada no meta-nível, decorre que a aplicação a ser testada não necessita ser modificada, garantindo com isto que o teste verificará o comportamento real do programa. A desvantagem deste protocolo, é que, como é implementado sobre a modificação de uma máquina virtual, esta torna-se necessária para a execução de uma aplicação (ferramenta de teste) que utilize seus recursos, contradizendo em parte, com a filosofia da linguagem Java.

6.1. Características da Ferramenta

Utilizando um meta-nível para monitorar objetos de classes selecionadas pelo testador, a ferramenta KTest intercepta toda a interação realizada entre para-objetos. Através da utilização de reflexão comportamental sobre as classes escolhidas para teste, mensagens enviadas a objetos instanciados destas classes são interceptadas pelo gerenciador do protocolo, o qual verifica qual meta-objeto está associado ao para-objeto receptor da mensagem, entregando-lhe o controle da aplicação. Este meta-objeto realiza então as computações necessárias à verificação das asserções.

A ferramenta KTest, possui as seguintes características e/ou funcionalidades:

- Executa o teste baseado em estados. Dessa maneira, podem ser avaliadas as várias mudanças de estados pelas quais passam os objetos de determinada classe, baseando-se no modelo dinâmico da classe (diagrama/máquina de estados);
- Verifica invariantes de classe, pré e pós-condições de métodos. Assim, torna-se possível a verificação da integridade dos estados de um objeto durante a execução do programa. Com a interceptação de mensagens entre objetos, possível com a reflexão destes, tal verificação pode ser realizada através de consultas aos valores dos atributos dos objetos;
- Armazena a seqüência de métodos chamados pelos para-objetos, tornando possível ao testador visualizar o histórico de interações no nível-base;
- Possui uma *interface* gráfica para interação com usuário, o que lhe possibilita escolher as classes, métodos, especificar asserções e visualizar resultados sobre a aplicação em teste.

Todas essas funções descritas, são realizadas sem a necessidade de instrumentação do código-fonte da aplicação em teste, constituindo esta, uma das mais importantes características da ferramenta KTest. Com a utilização da Reflexão Computacional é possível monitorar os objetos, implementando diferentes mecanismos de análise na aplicação, sem que isso venha a interferir no código do *software* em teste.

A classe *KTest* e suas especializações, definem a *interface* de interação com o usuário, gerenciando as informações por ele providas, sendo responsáveis também pela apresentação dos resultados da aplicação monitorada. Através da *interface* para interação com o usuário é possível escolher a aplicação para teste. Logo após, é apresenta a hierarquia de classes da aplicação, sendo relacionados também os métodos e atributos de cada classe. Em seguida, o usuário especifica quais classes e/ou métodos ele deseja selecionar para serem monitorados. Para cada classe escolhida pode ser especificada uma invariante associada a esta classe e, para cada método pode ser especificada a pré e pós-condição para avaliação, não sendo, entretanto, obrigatória a especificação dessas asserções. Na Figura 6.1 é apresentado o diagrama de funcionamento da KTest.

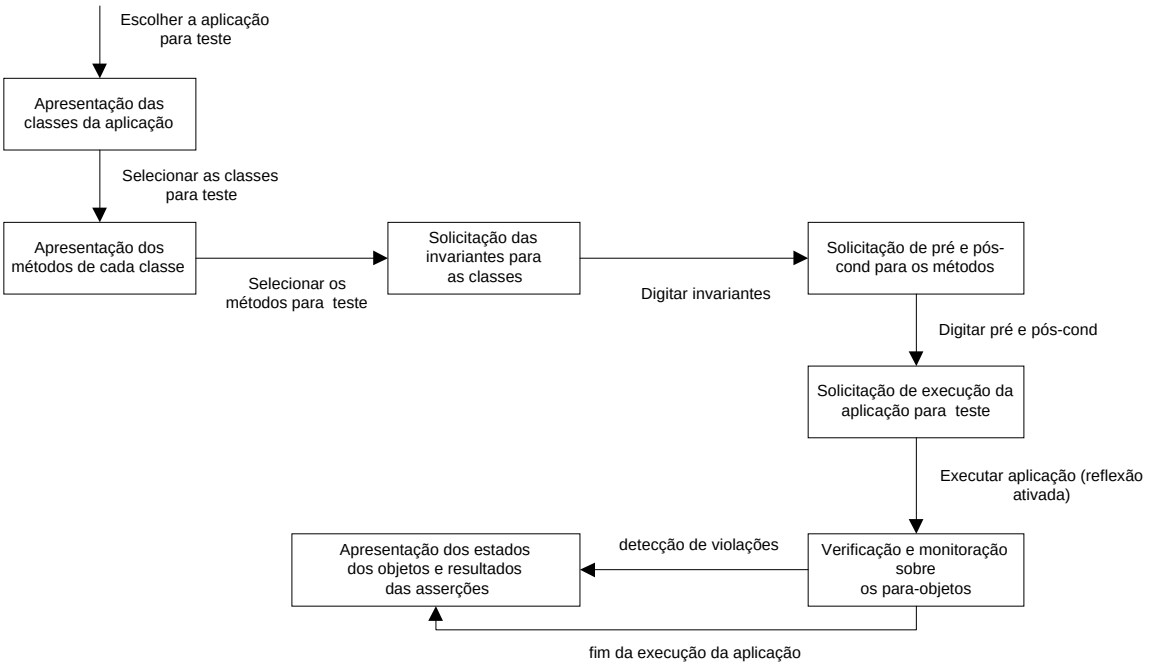

FIGURA 6.1 – Diagrama de Funcionamento da KTest

A identificação da hierarquia de classes de uma aplicação é obtida através de reflexão computacional estrutural, através da qual são coletadas todas as informações das classes. Tais informações, adicionadas das seleções feitas pelo usuário (classe/método), bem como as especificações das asserções, são armazenadas numa estrutura de classes, presente no meta-nível, apresentada na Figura 6.2.

Esta estrutura, formada pelas classes *ClassData*, *Metodo*, *Atributo*, *Construtor* e suas sub-classes, constituem o principal alvo de consultas realizadas pelos métodos dos meta-objetos, os quais a utilizam como fonte de informações para o embasamento das decisões computacionais realizadas no meta-nível.

A próxima etapa consiste na reconfiguração do meta-nível, com o objetivo de instanciar meta-objetos e associá-los às instâncias das classes escolhidas para teste. A meta-classe da ferramenta KTest, chamada *KMeta*, que estende a classe *MetaObject* do protocolo Guaraná, provê os recursos responsáveis para estas instanciações e operações do meta-nível. É apresentada na Figura 6.3, os relacionamentos que *KMeta* possui com as classes *Operation*, *Result*, *OperationFactory* e *Message*, definidas no protocolo Guaraná.

Executando-se a aplicação, esta é interrompida quando mensagens forem encaminhadas a objetos de classes selecionadas para teste, sendo transferido o controle da aplicação para o meta-nível, onde os métodos dos meta-objetos associados fazem as computações necessárias para a verificação das asserções e, consequentemente, validação sobre os estados dos para-objetos.

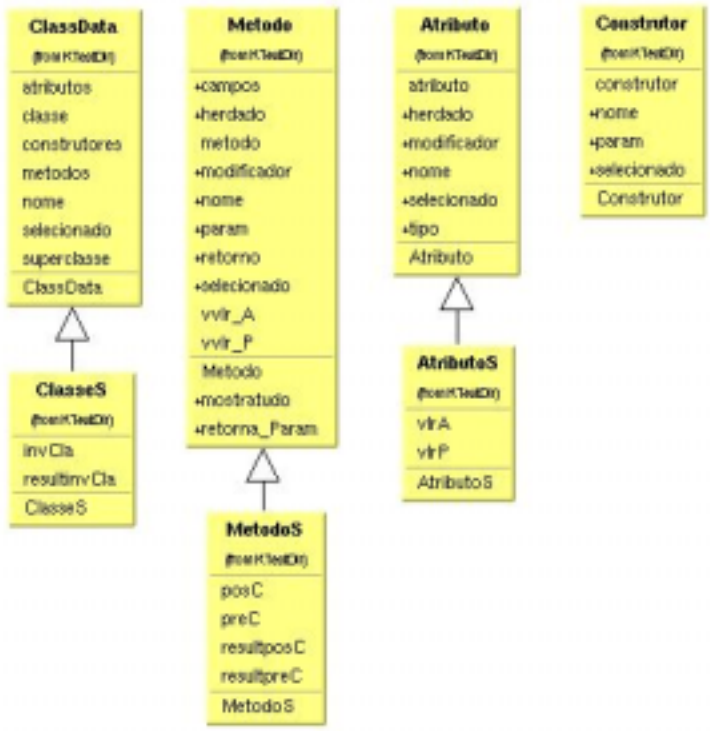

FIGURA 6.2 –Classes que armazenam informações sobre a aplicação em teste

6.2. Verificação das asserções

Através da verificação das asserções é possível determinar se o estado do objeto está ou não de acordo com a especificação feita pelo usuário. Conforme explanado anteriormente, são três os tipos de asserções especificadas pelo usuário: i) invariantes de classe ii) pré-condições de métodos e iii) pós-condições de métodos.

Sempre que existe interação entre objetos do nível-base, ou seja, quando mensagens são enviadas aos para-objetos, a chamada ao método é reificada como uma operação e entregue ao

meta-objeto correspondente ao para-objeto receptor da mensagem. Nesse momento, o meta-nível encarrega-se, primeiro, de verificar se tal operação refere-se a um objeto cuja classe foi previamente escolhida para teste, através de uma pesquisa na estrutura (classes) presente no meta-nível, a qual contém a relação de classes e métodos escolhidas para teste, além de outras informações. Se a classe é encontrada nesta estrutura, ocorre então uma busca referente ao método que foi invocado, de forma a verificar se o mesmo também foi selecionado para teste. Em caso afirmativo, o meta-nível recupera a pré-condição especificada para esse método e, se houver, a mesma é avaliada. As informações sobre o estado do objeto (valores dos seus atributos) são, então, captadas para verificar se o estado do para-objeto atende à pré-condição para a ativação do método solicitado. Caso essa pré-condição seja falsa, o meta-nível se encarregará de apresentar ao usuário a violação da asserção.

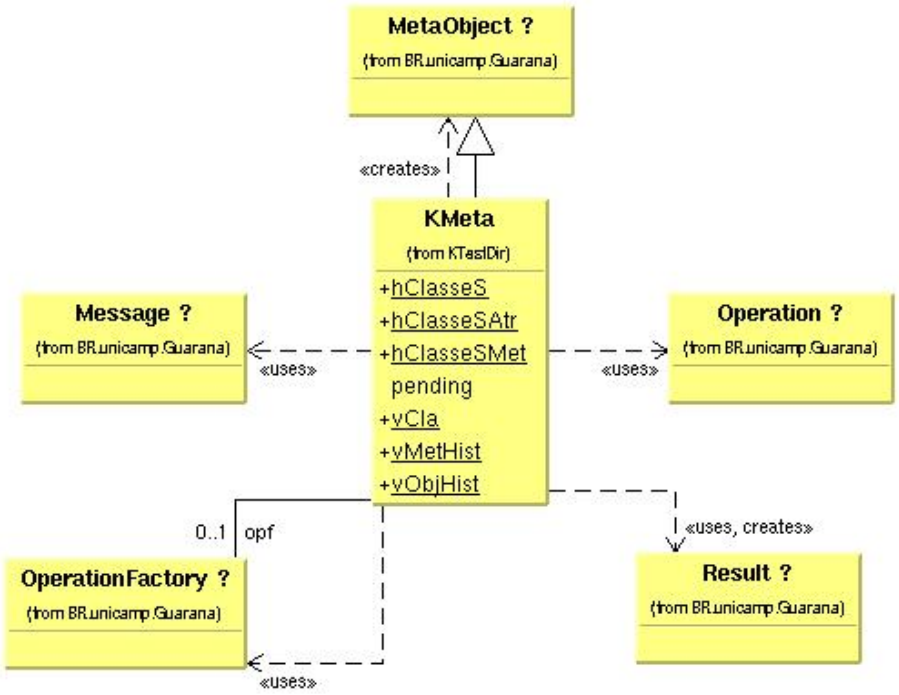

FIGURA 6.3 – Classe do Meta-Nível da KTest

Sendo a pré-condição verdadeira, o método original é chamado e, logo após o final de sua execução, o controle da aplicação é repassado novamente ao meta-nível, que captará na mesma estrutura, a pós-condição relativa a esse método, se esta existir. Em caso afirmativo, novamente são recuperados os valores dos atributos do objeto e a asserção é verificada, com o objetivo de saber se o estado do objeto atende à pós-condição, que define o estado final do objeto após a ativação do método em questão. Caso a pós-condição seja violada, o meta-nível apresentará ao usuário a situação.

Torna-se mister ressaltar que, sempre que mensagens são interceptadas, independentemente de ativarem ou não métodos escolhidos para monitoração, o meta-nível fará a recuperação da invariante de classe, caso seja uma classe escolhida para teste, com o propósito de verificar, após o término da execução do método, a invariante da classe. As informações do estado do objeto são, então, captadas e a respectiva invariante é validada para constatar se o objeto atende a essa condição.

Quando do término da execução da aplicação em teste ou, quando da ocorrência de uma violação sobre uma asserção, seja ela uma invariante, uma pré ou pós-condição, o usuário pode verificar a seqüência de ativação dos métodos da aplicação em teste e os estados dos objetos (valores dos seus atributos) antes e após a execução dos métodos escolhidos para teste, além do resultado da avaliação de cada uma das asserções.

7. CONCLUSÕES

Automatizar a fase de teste de software constitui-se numa relevante área de pesquisa, pois verifica-se um constante crescimento na complexidade das aplicações desenvolvidas. O paradigma OO tem contribuído para a reutilização de soluções, evidenciando que o teste deve ser realizado com o intuito de garantir que defeitos presentes nestas soluções não sejam propagados nestas reutilizações em futuras aplicações.

A utilização da reflexão computacional contribui de maneira significativa no processo de teste, permitindo que se monitore uma aplicação em tempo de execução, sem a necessidade da instrumentação do código-fonte. A API de reflexão da linguagem Java permite somente a realização de reflexão estrutural, não possibilitando reflexão comportamental. A reflexão comportamental permite, através de interceptação de mensagens entre objetos, uma monitoração das interações destes, tornando possível a verificação da integridade de objetos e/ou classes escolhidas para teste.

A ferramenta KTest aplica a reflexão comportamental para auxiliar o teste de programas escritos em Java. KTest utiliza o protocolo de reflexão *Guaraná*, para a monitoração de interações entre os objetos, verificando a integridade dos objetos, consultando os valores de seus atributos, analisando se os estados dos objetos são válidos, de acordo com pré, pós-condições e invariantes de classes especificadas pelo testador. Ressalta-se, que é possível estender KTest adicionando-se funcionalidades, tais como: inclusão de novas técnicas de teste, implementação em outra linguagem de programação, dependendo das características reflexivas desta, geração de gráficos demonstrando métodos ou variáveis mais acessados e preparação para o teste de aplicações reflexivas e distribuídas.

REFERÊNCIAS BIBLIOGRÁFICAS

[BIN95] BINDER, R. V. **Testing Object-Oriented Systems: A Status Report** Chicago: RBSC Corporation., 1994. Disponível em: <http://www.rbsc.com/pages/site_map.html>. Acesso em: 12 dez. 1999.

[CAM97] CAMPO, M. R. **Compreensão Visual de Frameworks através da Introspeção de Exemplos**. Porto Alegre: CPGCC da UFRGS, 1997. Tese de Doutorado.

[LIS98] LISBÔA, M.L.B. **Reflexão computacional no modelo de objetos**. Porto Alegre: CPGCC da UFRGS, 1998.

[MAE87] MAES, P. **Concepts and experiments in computational reflection**. In: OOPSLA'87, p. 147-169. Orlando, Flórida, 1987.

[MCG96] McGREGOR, J. **Testing Object-Oriented Components**. In: 10th European Conference on Object-Oriented Programming. Tutorial Notes. Jul. 1996.

[OLI98] OLIVA, A.; BUZATO, L. E.; GARCIA, I. C. **The Reflexive Architecture of Guaraná**. Instituto de Computação, Universidade Estadual de Campinas – UNICAMP, Campinas: SP, 1998. Disponível em: <http://www.dcc.unicamp.br/~oliva/guarana>. Acesso em: 10 jan. 2000.

[OLI98b] OLIVA, A.; BUZATO, L. E. **Composition of Meta-Objects in Guaraná**. Instituto de Computação, Universidade Estadual de Campinas – UNICAMP, Campinas: SP, 1998. Disponível em: <http://www.dcc.unicamp.br/~oliva/guarana>. Acesso em: 10 jan. 2000.

[PAL00] PALAVRO, I. **Ferramenta de Teste de Aplicações Orientada a Objetos Baseada em Estados**. Porto Alegre: PPCG da UFRGS, 2000. Dissertação de Mestrado.

[PIN98] PINTO, I. M. **Um Sistema de Apoio ao Teste de Aplicações Smalltalk**. Porto Alegre: CPGCC da UFRGS, 1998. Dissertação de Mestrado.

[PRE95] PRESSMAN, R. S. **Engenharia de Software**. São Paulo: Makron Books, 1995.

[RUB98] RUBIRA, C.M.F., SILVA, R.C., CORREA, S.L., BUZATO, L.E. **Reflexão Computacional em Linguagens de Programação: Um Estudo Comparativo**. In: Simpósio Brasileiro de Linguagens de Programação, 2. Campinas, SP, 1998.

[RUM94] RUMBAUGH, J. et al. **Modelagem e Projetos Baseados em Objetos**. São Paulo: Editora Campus, 1994.

[SEN01] SENRA, R. D. A. **Programação Reflexiva sobre o MOP Guaraná**. Campinas: Instituto de Computação da UNICAMP, 2001. Dissertação de Mestrado.