Ferramenta para Extração de Transações de Sistemas de Transformação de Grafos Transacionais

ERICK C. ESPINDULA[1]; TAMARA M. DA CRUZ[1]; SIMONE A. C. CAVALHEIRO[1]; LUCIANA FOSS[1]

[1]Universidade Federal de Pelotas
{ecespindula,tmdcruz,simone.costa,lfoss}@inf.ufpel.edu.br

1. INTRODUÇÃO

Atualmente, a exigência quanto a qualidade de um software é alta, tanto para seu custo, funcionamento, manutenção ou modificações. É essencial que determinado software funcione corretamente, pois, dependendo do caso, erros no software podem causar desde atrasos em aplicações de um computador a perdas financeiras em bancos ou até mesmo catástrofes envolvendo risco de vida, como por exemplo um software ligado a uma função de um avião.

Qualidade do software inclui diversos aspectos que envolvem o processo de desenvolvimento e neste trabalho são tratados dois aspectos: correção e flexibilidade [CAVANO, 1978]. Estes aspectos são essenciais para que o software produzido seja confiável e adaptável ao ambiente complexo das aplicações. A garantia da qualidade destes aspectos está intrinsecamente ligada às atividades de Verificação, Validação e Teste. Neste trabalho, nos focaremos na atividade de Verificação [CLARKE, 1996] [ROBINSON 2001], a qual visa a garantir que o software produzido segue a especificação definida.

Apesar de geralmente haver uma etapa específica para verificar a correção, idealmente, esta atividade deveria ser desempenhada ao longo de todo o ciclo de vida do software. Desta forma, esta atividade poderia ser utilizada para garantir a preservação de certos aspectos do software após cada etapa, permitindo a identificação precoce de erros em etapas onde eles fossem mais simples de corrigir, reduzindo custos.

Uma das formas de se aumentar a flexibilidade do software é aumentar o nível de abstração do projeto e usar ferramentas que permitam construir o software de forma automática. O uso de modelos abstratos tem se mostrado um mecanismo importante para controlar a complexidade do software e facilitar sua evolução, mas a integração de modelos e técnicas de verificação é bastante limitada. O ideal seria garantir que os resultados de uma verificação realizada no nível de especificação são válidos para o código gerado. Para tentar administrar as mudanças em um projeto e manter a qualidade do software final, surgiu uma abordagem que permite tratar essas mudanças reusando projeto de software, e até mesmo código. Esta abordagem é a Engenharia de Software Guiada por Modelos (MDE) [KLEPPE, 2003] [SCHMIDT, 2006]. Frente a este cenário, o projeto no qual este trabalho está inserido tem como objetivo propor técnicas de desenvolvimento de software que visam integrar as técnicas de verificação com MDE.

Transformações sobre modelos podem ser simples (especificando somente pequenas mudanças) ou muito complexas (especificando mudanças para um modelo completamente diferente ou para um passo complexo de refinamento). Usualmente, as transformações são descritas por um conjunto de regras e para garantir que os resultados produzidos são válidos, deve ser assegurado que o

processo de aplicação destas regras sempre termina, é confluente e semanticamente correta.

Como muitas linguagens de especificação são visuais, é natural considerar abordagens que permitam que transformações de modelos sejam transformações sobre grafos. Existem várias abordagens que usam gramáticas de grafos para descrever tipos diferentes de transformações de modelos [MENS, 2002] [BIERMANN, 2006] [EHRIG, 2008] [RANGEL, 2008]. A ideia é representar o modelo de origem através de um grafo e descrever a transformação deste em um outro através de regras de transformação de grafos. A área de gramáticas de grafos (ou transformação de grafos) [ROZENBERG, 1997] oferece vários resultados para análise (como terminação, confluência, independência) e portanto parece adequada para servir de embasamento formal para transformações de modelos. Porém, grande parte dessas análises se baseia em propriedades de cada regra separadamente. Para transformações simples, é possível que esse tipo de análise seja suficiente, mas qualquer transformação de modelos um pouco mais elaborada exigirá que o processo seja realizado em passos, descritos por diferentes regras que serão aplicadas em sequência (e/ou concorrentemente) para completar a transformação do modelo. Provar correção ou outra propriedade desejada para este tipo de transformação é uma tarefa difícil.

A noção de transações para transformações de grafos [FOSS, 2008] permite relacionar sequências de passos de transformação a um passo abstrato que descreve o efeito de toda a sequência. Com o uso deste conceito, pode-se obter uma única regra abstrata que representa uma transação (sequência mais complexa de transformações). Assim, usando a regra abstrata, podemos usar as técnicas de análise já existentes para verificar propriedades de sequências mais complexas de transformações. Porém, a tarefa de se obter todas as regras abstratas correspondentes às transações de um sistema é uma tarefa bastante complexa e uma ferramenta que realizasse automaticamente este processo seria muito útil para que o uso desta abordagem se tornasse mais efetivo. Com isso, este trabalho tem como objetivo apresentar um algoritmo que permita extrair as regras abstratas correspondentes às transações de um sistema de transformação de grafos transacionais (T-GTS), bem como implementá-lo.

2. METODOLOGIA

Para que o objetivo deste trabalho possa ser alcançado, primeiramente, será realizado um estudo sobre o formalismo de T-GTS, bem como sua extensão para descrição de transformações de modelos. Baseando-se nesses estudos, será realizada a definição de um algoritmo para a extração das transações de um sistema especificado em T-GTS. Também será realizada a implementação deste algoritmo usando uma linguagem de programação adequada, que permita a integração com as ferramentas de especificação já existentes.

Além disso, a elaboração de um estudo de caso será realizada, no qual um sistema será especificado usando T-GTS, bem como a ferramenta desenvolvida. Em um momento final, uma análise dos resultados do estudo de caso deverá ser feita para verificar a adequação deste formalismo e da ferramenta criada para a transformação de modelos.

3. RESULTADOS E DISCUSSÃO

Este trabalho tem como objetivo a criação de uma ferramenta que permite extrair regras abstratas que correspondem a transações de um sistema descrito

com T-GTS, integrando esta a alguma ferramenta de especificação já existente.

O trabalho é recente e até dado momento, apenas os estudos sobre gramática de grafos e T-GTSs foram abordados, tal como a discussão da linguagem que será usada para o desenvolvimento da ferramenta. A seguir serão brevemente apresentados os conceitos já estudados.

Sistema de transformação de grafos *transacionais* (T-GTS), proposto em [FOSS, 2008], como uma extensão de sistemas de transformação de grafos (GTS), onde foi introduzido o conceito de transações. As transações são computações que realizam uma tarefa, o início e o fim são descritos por estados chamados estáveis, enquanto estados intermediários são chamados instáveis. Uma transação é definida por um conjunto de ações que são realizadas atomicamente, ou seja, de modo indivisível, onde todas as ações devem ser realizadas antes de qualquer outra aplicação seja feita [DOWNEY, 2013]. Sendo assim, uma transação é efetivada somente se todas as suas ações forem realizadas, caso contrário, nenhuma de suas ações terá efeito.

Na Figura 1, vemos um exemplo de um sistema especificado com T-GTS, o qual descreve o comportamento de um cliente em um posto de gasolina. Este T-GTS é composto de um grafo-tipo (exibido na parte inferior da figura) e um conjunto de regras (mostradas na parte superior da figura). O grafo-tipo nos mostra todas as entidades que aparecem no sistema: o cliente (Customer), o operador (Operator) e a bomba (Pump). As regras descrevem o comportamento deste sistema. O comportamento do cliente permite que ele realize as seguintes tarefas (ou transações):

1. Liberar o abastecimento fazendo um pré-pagamento (regra Init);
2. Iniciar o abastecimento e prosseguir até atingir o volume de combustível desejado (regras StartSupply e EndSupply);
3. Tentar reiniciar processo para liberação do abastecimento caso o operador da bomba esteja ocupado (regra Retray)
4. Ou retornar ao estado inicial, podendo recomeçar o processo de abastecimento em qualquer momento (regra Restart).

As tarefas 1,2 e 4 são realizadas em um único passo, porém a tarefa 3 necessita de dois passos para ser concluída. Nesta transação podemos ver um elemento instável, isto é, faz parte de um estado intermediário desta transação. Este elemento é a mensagem Stop (representada por linhas tracejadas). Assim, para realizar a tarefa 3, a regra StartSupply deve ser aplicada em um estado estável (início da transação), levando a um estado instável (estado intermediário) e após a regra EndSupply deve ser aplicada, levando a um estado estável (fim da transação).

4. CONCLUSÕES

Este trabalho está em seu estágio inicial, onde estão sendo estudados os conceitos teóricos que devem dar suporte ao seu desenvolvimento. Como contribuição final deste trabalho pretende-se que a ferramenta desenvolvida possa dar suporte à especificação de transformações de modelos complexas, que necessitem de diversas etapas para serem concluídas.

Nas próximas etapas, serem estudados com maior profundidade os conceitos de transformações de grafos e transações. Além disso, tem-se que estudar a extensão do conceito de transações proposta para especificação de transformação de modelos, para que se possa desenvolver a ferramenta para esta nova noção de transações.

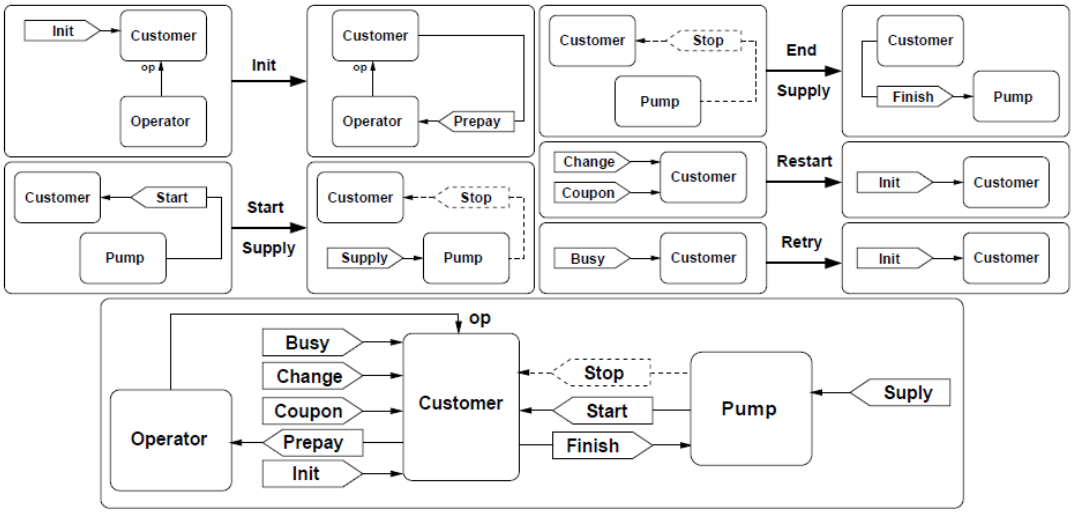

Figura 1: T-GTS de um cliente em um posto de gasolina.

5. REFERÊNCIAS BIBLIOGRÁFICAS

BIERMANN, E.; EHRIG, K.; KÖHLER, C.; KUHNS, G.; TAENTZER, G.; WEISS, E. **EMF model refactoring based on graph transformation concepts**, ECEASST 3, 2006.

CAVANO, J. P.; MCCALL, J. A. A framework for the measurement of software quality. **SIGSOFT Software Engineering Notes**, New York, v. 3, n. 5, p. 133-139, 1978.

CLARKE, E. M.; WING, J. M. Formal methods: State of the art and future directions. **ACM Computing Surveys**, Dakota, v. 28, p. 626-643, 1996.

DOWNEY, A. B. **The Little Book of Semaphores**. Version 2.1.5. Acessado em 25 set. 2013. Online. Disponível em: http://greenteapress.com/semaphores/downey08semaphores.pdf.

EHRIG H.; ERMEL, C. **Semantical correctness and completeness of model transformations using graph and rule transformation**. LNCS, v. 5214, Springer, pp. 194–210, 2008.

FOSS, L. **Transactional Graph Transformation Systems**. 2008. Tese (Doutorado em Ciência da Computação) – Programa de Pós-Graduação em Computação, Universidade Federal do Rio Grande do Sul.

KLEPPE, A. G.; WARMER, J.; BAST, W. **MDA explained: The model driven architecture: Practice and promise**. Boston: Addison-Wesley Longman, 2003.

MENS, T.; DEMEYER, S,; JANSSENS, D. **Formalising behavior preserving program transformations**. LNCS, v. 2505, Springer, p. 286–301, 2002.

RANGEL, G.; LAMBERS, L.; KÖNIG, B.; EHRIG, H.; BALDAN, P. **Behavior preservation in model refactoring using DPO transformations with borrowed contexts**. LNCS, v. 5214, Springer, p. 242–256, 2008.

ROBINSON, A.; VORONKOV, A. (Eds.). **Handbook of automated reasoning**. Amsterdam: Elsevier, 2001.

ROZENBERG, G. (ed.). **Handbook of graph grammars and computing by graph transformation: Fondations**. World Scientific, 1997.

SCHMIDT, D. C. Model-driven engineering. **IEEE Computer**, v. 39, n. 2, 2006.