Anais da 2ª Escola Regional de Engenharia de Software (ERES 2018)
em Dois Vizinhos, PR, Brasil, 22 a 24 de outubro de 2018

Em Direção à Engenharia Reversa Adaptativa de Binários e Códigos para Modelo

Giliardi Schmidt[1], **Guilherme Bolfe**[1],
Fábio Paulo Basso[1], **Elder Rodrigues**[1], **Maicon Bernardino**[1]

[1]Universidade Federal do Pampa (UNIPAMPA)
Av. Tiarajú, 810, Ibirapuitã – Alegrete, RS – Brasil
Laboratory of Empirical Studies in Software Engineering (LESSE)

gili.schmidt@hotmail.com
guilhermebolfe11@gmail.com, fabiopbasso@gmail.com,
eldermr@gmail.com, bernardino@acm.org

Abstract. *Through code-to-model reverse engenireering techniques it is possible to extract structural information from source code to a level independent of the programming language adopted. Some benefits assossiated with these techniques include ease of understanding of poorly documented systems as well as the (semi-) automatic migration of applications from one technology to another. This paper investigates the possibility for the adaptative extraction of structural characteristics from source codes written in the C++ laguague to a UML model, presenting a proof concept and discussions about future works.*

Resumo. *Por meio de técnicas de engenharia reversa de código-para-modelos é possível extrair informações estruturais de códigos-fonte para um nível independente da linguagem de programação adotada. Alguns benefícios associados com essas técnicas incluem facilidade para o entendimento sobre sistemas mal documentados e também a migração (semi-)automática de aplicações de uma tecnologia para outra. Este artigo investiga a possibilidade para a extração adaptativa de características estruturais de códigos-fonte escritos em C++ para modelos UML, apresentando uma prova de conceito e discussões sobre trabalhos futuros.*

1. Introdução

Muitos produtos de software com o passar do tempo ficam obsoletos, devido à constante evolução das necessidades dos usuários e da evolução natural das tecnologias disponíveis, necessitando assim manutenções e/ou adições de novas funcionalidades. Para realizar estas alterações no software, faz-se necessário entender o código-fonte do sistema: sua estrutura, algoritmos e fluxos de execução. Porém, há situações em que este código-fonte está documentado de maneira inconsistente ou superficial, ou até mesmo não possua nenhum documento de apoio, tornando o entendimento do sistema uma tarefa custosa.

Além da necessidade de alterar o código-fonte do software, há casos em que deseja-se efetuar as tarefas de *Verificação e Validação* (V&V) no sistema. Estas duas tarefas podem necessitar de uma documentação mais abrangente sobre o código-fonte, como a estrutura estática do mesmo [Korshunova et al. 2006]. O processo de verificação e validação analisa exaustivamente e testa o software para determinar se ele executa

suas funções, para garantir que ele realiza suas funções de forma correta e mensura sua qualidade e confiabilidade [D. R. Wallace 1989].

Para obter esta documentação faltante, é possível utilizar a engenharia reversa, onde parte-se de um código-fonte pronto e eleva-se para um nível mais abstrato, um modelo. O processo de obtenção de representações úteis de alto nível que se da a partir de código-fonte é chamado engenharia reversa [Brunelière et al. 2014]. Ferramentas de modelagem UML - *Unified Modeling Language* como Astah e EA, peritem ao engenheiro de software aplicar a engenharia reversa de código para diagramas de classes. No entanto, o modelo revertido é poluído com detalhes de APIs, o que dificulta sua legibilidade em atividades de V&V. Além disso, para ser utilizado com propósitos de migração, o modelo revertido precisa ser constantemente e manualmente ser refinado para um outro livre de detalhes de implementação de plataformas alvo.

Neste artigo apresenta-se uma nova abordagem para engenharia reversa no refinamento automatizado de modelos em níveis independentes de plataforma. O refinamento deve-se dar de modo adaptativo para contextos inter-organizacionais [Basso et al. 2017]. Como resultado, espera-se gerar modelos em níveis de abstração úteis para a execução de atividades de V&V e de migração de aplicações. Portanto, apresenta-se uma nova perspectiva de investigação em relação à outras abordagens para engenharia reversa existentes [Korshunova et al. 2006, Brunelière et al. 2014, Méndez-Acuña et al. 2017].

O artigo é organizado conforme segue: A Seção 2 discute nossa proposta e a Seção 3 apresenta um resumo dos principais trabalhos relacionados. A abordagem atual para engenharia reversa é demonstrada na prova de conceito apresentada na Seção 4. Na Seção 5 apresenta-se uma discussão sobre a viabilidade para aplicação da abordagem FOMDA para engenharia reversa adaptativa (FOMDA-reverse) e os próximos passos para estendê-la e viabilizá-la em contextos inter-organizacionais de reutilização. For fim, a Seção 6 apresenta as conclusões.

2. Visão Geral da Abordagem FOMDA-Reverse

A contribuição deste trabalho é um estudo analítico-prático discutindo sobre a viabilidade de aplicação da engenharia reversa adaptativa. Inicialmente, extraímos de um código na linguagem C++ as características estruturais estáticas, como classes, atributos de classes e assinaturas de métodos, para um diagrama de classes UML. Este diagrama UML, por fim, é salvo em um arquivo no formato XMI.

Para conseguir elevar o código a um nível mais alto é necessário criar uma árvore sintática abstrata - AST (*Abstract Syntax Tree*), esta que por sua vez possui diversas formas de ser construída, utilizando diferentes ferramentas. Neste trabalho optou-se por utilizar a ferramenta ANTLr [Parr 2013], o qual se adequou mais à necessidade da elaboração de uma prova de conceito devido o fato de possuir gramaticas prontas. Após sua construção, a AST pode ser utilizada para analise de softwares, desenvolvimento de softwares e construção de compiladores [G. Fischer 2007].

Para a implementação de uma nova abordagem para engenharia reversa adaptativa, parte-se de um trabalho anterior já consolidado. Em experiencias anteriores executadas por meio da abordagem FOMDA [Basso et al. 2013], modelos UML foram adotados em processos adaptativos de transformações de modelo-para-código (M2C), portanto permitindo a customização na geração de código para contextos inter-organizacionais.

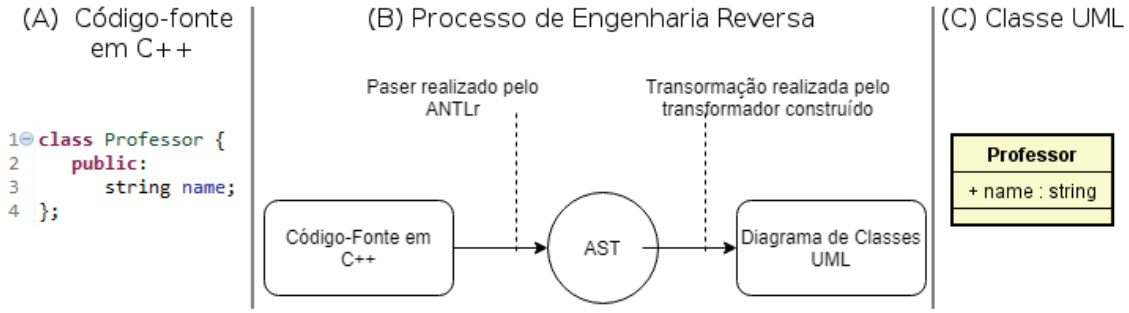

Figura 1. Um simples processo de transformação do código-fonte para modelo

Neste artigo apresenta-se uma prova de conceito que investiga a viabilidade para se aplicar o processo contrário, em transformações de código-para-modelo (C2M). Como resultado, espera-se o desenvolvimento de suporte ferramental que permita reverter código para modelos em níveis independentes de características de plataformas. Portanto, nossa proposta é chamada de FOMDA-reverse e permitirá executar atividades automáticas de refinamento de modelos independentes de plataforma baseadas em contextos.

De momento, esta pesquisa encontra-se em estágio inicial de caracterização do problema. Assim, este estudo investigou os elementos necessários para o desenvolvimento de componentes que aplicam transformação de código-para-modelo seguindo uma técnica comum aos trabalhos relacionados. Ou seja, adotou-se uma abordagem simples de engenharia reversa, utilizando-se de duas etapas para a transformação do código-fonte para o modelo UML. Portanto, no estágio atual não foram considerados diferentes contextos e componentes adaptativos para reverter modelos de código.

A Figura 1 apresenta uma visão geral deste simples processo de transformação como segue:

Passo 1: Levantamento de características estruturais para uma AST. Nesse passo é gerada uma AST utilizando a ferramenta ANTLr. Focou-se no desenvolvimento de um componente para transformação construído para extrair informações de estrutura estática de um código-fonte escrito na linguagem C++ para um diagrama de classes UML, utilizando a ferramenta ANTLr.

Passo 2: Transformação dos dados em conformidade com a AST para uma representação comum a plataformas/linguagens cruzadas em programação orientada à objetos (C++, Java, C#, PHP, etc). Nesse sentido, o transformador construído navega por esta AST, extraindo informações sobre as classes, atributos e métodos destas classes, gerando um modelo composto de um diagrama de classes UML. Adotou-se a UML como linguagem de representação comum devido à sua grande expressividade para representar sistemas orientados à objetos.

3. Trabalhos Relacionados

No trabalho apresentado por [Korshunova et al. 2006] é feita a engenharia reversa de código-fonte para modelos UML. A ferramenta CPP2XMI possibilita gerar diversos diagramas, tais como: classes, sequência e atividades. Estes diagramas são salvos no formato XMI.

No trabalho apresentado por [Fleurey et al. 2007] foi realizada a migração de um conjunto de aplicações de uma instituição financeira. Parte do trabalho de migração consistiu em extrair informações sobre o código-fonte das aplicações existentes para modelos utilizando ASTs.

O projeto *open-source* MoDisco [Brunelière et al. 2014] provê um *framework* genérico e extensível para engenharia reversa. Diferentemente de outras soluções de engenharia reversa que apenas geram modelos UML a partir de tecnologias específicas, e vice-versa, este *framework* tem como objetivo prover suporte para a geração de modelos a partir de diferentes metamodelos, portanto tendo como entrada diversos tipos de artefatos como código-fonte, banco de dados, etc.

Estes trabalhos aplicam engenharia reversa utilizando-se de processos de extração de informações do código sem considerar o contexto. MoDisco é um suporte ferramental rico e flexível para permitir customização no processo de engenharia reversa, mas ele ainda é limitado para a execução adaptativa dos componentes de transformação reversa. Isso acarreta em modelos revertidos com informações desnecessárias, que atrapalham a execução de atividades de V&V e de migração de aplicações.

Para resolver esse problema, é necessário tornar o processo de engenharia reversa adaptativo. Essa necessidade é diferente de se reverter um modelo para arquiteturas de linhas de produto, por exemplo, como o realizado em [Méndez-Acuña et al. 2017]. Neste trabalho, apesar de tratar do termo "contexto", o processo de extração de detalhes de implementação continua sendo estático. Diferentemente, FOMDA-reverse tornará este processo dinâmico, permitindo a inclusão de componentes adaptativos que extraem dados independentemente de se tratar de um código para linha de produtos ou um software simples.

4. Implementação do Componente para Transformação C2M

Com a utilização da ferramenta ANTLr, foi gerado um *parser*, para a linguagem de programação Java. Para a geração deste *parser*, utilizou-se uma gramática pronta [Sanchez]. A partir do *parser* gerado pelo ANTLr, estendeu-se a classe abstrata *BaseVisitor*, gerada automaticamente pelo ANTLr, e criou-se diversos *Visitors* para extrair as informações necessárias da AST gerada. A Figura 2 apresenta um dos *Visitors* criados, sendo o responsável por extrair informações sobre uma classe.

Assim, o transformador construído neste trabalho aplica os *Visitors* gerados na AST e cria os elementos do diagrama UML com base no tipo de nó visitado.

O transformador gerado neste trabalho tem a capacidade de extrair as informações estruturais de classes: o nome da mesma, quais são seus atributos e funções, a visibilidade destes atributos e funções e a assinatura (parâmetros de entrada e tipo de retorno) destas funções.

Para exemplificar o seu funcionamento, considere o que é apresentado nas Figuras 1(A) e 2. Ao fornecer como entrada o código-fonte da Figura 1 (A) ao *parser* criado pelo ANTLr, é gerada de forma automatizada uma AST.

Ao aplicar o *Visitor ClassSpecifierVisitor* ao nó raiz de uma AST, sempre que houver um nó do tipo *classspecifier* (que corresponde à uma classe), o método *visitClassSpecifier(ClassSpecifierContext ctx)* será invocado. Para extrair o nome desta classe,

```
9  public class ClassSpecifierVisitor extends BaseVisitor<Class> {
10
11     public ClassSpecifierVisitor(Model model) {
12         super(model);
13     }
14
15     @Override
16     public Class visitClassspecifier(ClassspecifierContext ctx) {
17         Class c = null;
18
19         String name = ctx.classhead().classheadname().getText();
20
21         try {
22             c = super.createClass(name);
23
24             if (ctx.classhead().baseclause() != null) {
25                 ClassNameVisitor classNameVisitor = new ClassNameVisitor(model, c);
26                 ctx.classhead().baseclause().accept(classNameVisitor);
27             }
28
29             MemberSpecificationVisitor memberSpecificationVisitor = new MemberSpecificationVisitor(model, c);
30             ctx.accept(memberSpecificationVisitor);
31
32         } catch (Exception e) {
33             e.printStackTrace();
34         }
35
36         return c;
37     }
```

Figura 2. Visitor gerado para visitar nós do tipo *classspecifier*

navega-se pelos filhos do nó em questão, como exemplificado na linha 19 da Figura 2. Após isso, são aplicados novos *Visitors* conforme necessário. Por exemplo, é aplicado um novo *Visitor*, que é responsável por extrair as informações de atributos e funções da classe (linhas 29 e 30 da Figura).

Como resultado, nossas transformações resultam em um modelo UML que é independente da plataforma de desenvolvimento e da própria linguagem C++. Para este modelo se dá a definição de: Modelo Independente de Plataforma (PIM) [Basso et al. 2013]. Por fim, o transformador construído chamado de "visitor" ainda não é adaptativo. Para tal, a seção 4.1 apresenta o planejamento para incremento de pesquisa existente que já traz suporte ferramental para a execução de transformações adaptativas.

4.1. Um Paralelo com a Abordagem FOMDA

Uma vez que o modelo UML é extraído a partir de um código-fonte em C++, ele é refinado para a geração de código em outra plataforma alvo, como ilustrado na Figura 3. Assim, o modelo é a entrada "PIM do Sistema A" para um processo de transformação, também chamado de *assembly*, configurado em FOMDA DSL [Basso et al. 2013].

De modo à permitir a execução adaptativa de componentes de transformação de modelos, a abordagem FOMDA oferece uma DSL para a representação de um Modelo de Domínio de Plataformas (PDM) [Basso et al. 2017], que é integrado em uma cadeia de transformação e com o modelo de entrada da seguinte forma:

1) O projetista do plano de migração utiliza as configurações (PDM + transformadores localizados em "Transformações M2M"), para refinar um modelo do sistema (PIM) para modelos em nível intermediário [Basso et al. 2016]. Para tanto, o projetista de aplicação precisa: A) selecionar as características não funcionais do PDM, gerando assim uma instância do PDM; B) indicar qual é o PIM que representa os requisitos funcionais do sistema; C) indicar os transformadores que ele vai utilizar para converter o PIM em

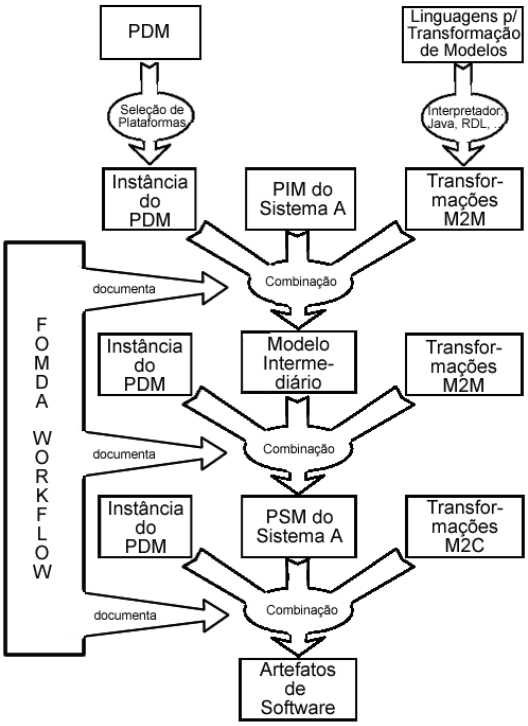

Figura 3. Modelos e Transformações Utilizadas pela Abordagem FOMDA

um novo modelo. Estas três tarefas do projetista de aplicação são identificadas na Figura 3 pela seta que combina a saída dos três modelos (a instância do PDM, o PIM e uma transformação M2M).

2) Tais configurações definem uma receita que organiza e documenta o processo de transformações de modelos que o projetista de aplicações deve seguir. Essa ordem precisa organizar as características do PDM, os elementos do PIM e as transformações.

3) Para organizar os refinamentos intermediários de modelos entre um PIM e um PSM, o projetista de transformações especifica e configura um Domínio de Cadeia de Transformações (TCDM). Ele é utilizado para fazer a combinação entre as características selecionadas no PDM, os elementos do PIM e as transformações.

4) O TCDM oferece ao projetista de aplicação uma receita para gerar cadeias de transformações para contextos intra-organizacionais. As cadeias, por fim, permitem transformar PIMs em PSMs por meio do mapeamento de características de plataformas que ditam como a composição de transformações adaptativas acontece.

5) O resultado final é ilustrado pela geração dos artefatos do sistema para uma nova plataforma de execução. Para mais informações, as contribuições em [Basso et al. 2013, Basso et al. 2014] demonstram elementos da FOMDA DSL que permitem execuções adaptativas de componentes de transformação.

5. Discussões

A abordagem FOMDA vem se mostrando efetiva para flexibilizar a geração de modelos e código no desenvolvimento de sistemas para múltiplos domínios [Basso et al. 2017].

A mesma foi utilizada em contextos reais de fábricas de software e se mostrou útil para reutilização de componentes de transformação de modelos [Basso et al. 2013]. Logo, para o próximo trabalho, planeja-se o desenvolvimento de geradores de código para diferentes plataformas, complementando assim o *framework* disponível nesta abordagem.

Além disso, como a abordagem foi concebida para executar processos de transformações adaptativos, a pergunta comum de pesquisa que pretende-se investigar é: Quais são as vantagens e desvantagens ao se introduzir os conceitos de variabilidade de comunalidade também no processo de engenharia reversa de um processo de migração de aplicações motivado neste artigo?

Como exemplificado em [Basso et al. 2016], realizar a engenharia reversa de código Java para modelos é uma tarefa crítica para o sucesso de abordagens para *Rountrip Engineering* [Kelly and Tolvanen 2008]. *Rountrip* permite a execução de uma abordagem de MDE que, tanto pode gerar o código a partir de um modelo de entrada, quanto reverter o código gerado para modelos. No entanto, a abordagem FOMDA nunca foi aplicada para processos de migração de aplicações.

Assim, propõem-se como objetivo de pesquisa derivado deste artigo, que a ordem da Figura 3 seja executada em ordem reversa. A meta para trabalhos futuros é estender a API FOMDA [Basso et al. 2014] para tornar os transformadores de engenharia reversa, como o demonstrado na Figura 2, adaptativos para o contexto do código. Isso terá implicações positivas para se executar *Rountrip Engineering* no futuro, sendo este um tópico de investigação bastante recente na literatura [Méndez-Acuña et al. 2017]. Assim, planeja-se esta meta de acordo com três pesquisas dirigidas:

- Engenharia reversa adaptativa de diretivas de APIs misturadas com sintaxe de linguagens de programação. A prova de conceito desenvolvida neste trabalho levou em conta um código puramente basado em C++. No entanto, os códigos reais são repletos de bibliotecas adicionadas ao conjunto de comandos padrão da linguagem. Logo, estes detalhes específicos de APIs devem ser recuperados também numa abordagem adaptativa para engenharia reversa;
- Engenharia reversa adaptativa de diretivas de *tag-libraries* misturadas com sintaxe de código de interfaces gráficas de usuário Web (JSP, JSF, etc.). Aqui o problema se caracteriza do mesmo modo que na proposta anterior, porém as informações estão sob a forma de *tags* XML, Javascript e arquivos semi-estruturados, e;
- Engenharia reversa adaptativa de binários. Também ruma-se nessa implementação para o desenvolvimento de casos de testes adaptativos [Basso et al. 2014], com a utilização de Java *Reflection* e *Annotations*.

6. Conclusão

O transformador gerado tem a capacidade de extrair as informações estruturais estáticas de códigos fonte escritos em C++. Porém, sua efetividade não foi testada e avaliada. Ou seja, pode haver situações em que o transformador não consiga extrair as informações corretas, ou até mesmo não funcione.

É possível citar como vantagem do trabalho realizado que, após abstrair as informações do código-fonte para um modelo, é possível exportá-lo para outras linguagens de programação. Assim, facilitando a migração de aplicações. Além de exportações e

migrações, após o processo de engenharia reversa estar aplicado, pode-se utilizar técnicas de verificação e validação nos modelos gerados, garantindo uma melhor qualidade das aplicações.

Referências

Basso, F. P., Oliveira, T. C., and Farias, K. (2014). Extending junit 4 with java annotations and reflection to test variant model transformation assets. In *29th Symposium On Applied Computing*, SAC'14, pages 1601–1608.

Basso, F. P., Oliveira, T. C., Werner, C. M., and Becker, L. B. (2017). Building the foundations for 'mde as service'. *IET Software*, 11:195–206(11).

Basso, F. P., Pillat, R. M., Oliveira, T. C., and Becker, L. B. (2013). Supporting large scale model transformation reuse. In *12th International Conference on Generative Programming: Concepts & Experiences.*, GPCE'13, pages 169–178.

Basso, F. P., Pillat, R. M., Oliveira, T. C., Roos-Frantz, F., and Frantz, R. Z. (2016). Automated design of multi-layered web information systems. *Journal of Systems and Software*, 117:612 – 637.

Brunelière, H., Cabot, J., Dupé, G., and Madiot, F. (2014). Modisco: A model driven reverse engineering framework. *Information and Software Technology*, 56(8):1012–1032. cited By 65.

D. R. Wallace, R. U. F. (1989). Software verification and validation: an overview. 6(3):10–17.

Fleurey, F., Breton, E., Baudry, B., Nicolas, A., and Jézéquel, J.-M. (2007). Model-driven engineering for software migration in a large industrial context. In *International Conference on Model Driven Engineering Languages and Systems*, pages 482–497. Springer.

G. Fischer, J. Lusiardi, J. v. G. (2007). Abstract syntax trees – and their role in model driven software development. pages 1–6.

Kelly, S. and Tolvanen, J.-P. (2008). *Domain Specific Modeling: Enabling Full Code Generation*. IEEE Computer Society - John Wiley & Sons.

Korshunova, E., Petkovic, M., g. j. V. Den Brand, M., and Mousavi, M. R. (2006). Cpp2xmi: Reverse engineering of uml class, sequence, and activity diagrams from c++ source code. In *2006 13th Working Conference on Reverse Engineering*, pages 297–298.

Méndez-Acuña, D., Galindo, J. A., Combemale, B., Blouin, A., and Baudry, B. (2017). Reverse engineering language product lines from existing dsl variants. *Journal of Systems and Software*, 133:145 – 158.

Parr, T. (2013). *The Definitive Antlr 4 Reference*. Pragmatic Bookshelf; Edição: 2 (25 de janeiro de 2013).

Sanchez, C. Cpp14 grammar. https://github.com/antlr/grammars-v4/tree/master/cpp. Acessado em: 03/07/2018.