UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

WILLIAM BOMBARDELLI DA SILVA

Towards Synchronizing Relations Between Artifacts in the Java Technological Space

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação

Trabalho realizado na Technische Universität Berlin dentro do acordo de dupla diplomação UFRGS - TU Berlin.

Orientadora brasileira: Prof.ª Dra. Leila Ribeiro
Orientador alemão: Dr.-Ing. Frank Trollmann

Porto Alegre
2016

CIP — CATALOGAÇÃO NA PUBLICAÇÃO

Bombardelli da Silva, William

Towards Synchronizing Relations Between Artifacts in the Java Technological Space / William Bombardelli da Silva. – Porto Alegre: CIC da UFRGS, 2016.

9 f.: il.

Trabalho de conclusão (graduação) – Universidade Federal do Rio Grande do Sul. Curso de Ciência da Computação, Porto Alegre, BR–RS, 2016. Orientador: Leila Ribeiro.

1. Model Synchronization. 2. Java Metamodels. 3. Network of Models. 4. Iterative Model Transformation. 5. Model Transformation. 6. Model-driven Engineering. 7. Software Engineering. I. Ribeiro, Leila. II. Título.

RESUMO

O uso de modelos em processos de engenharia de software tem crescido nos últimos anos. Ao passo que o uso cresce, cresce também a relevância de alguns problemas relacionados à área. Um deles é o problema de sincronização de modelos, que consiste basicamente em manter todos os modelos de uma aplicação de software consistentes entre si. Em outras palavras, os modelos de um software tendem a ser mudados ao longo do seu tempo de vida, e quando isto acontece, estas mudanças têm que ser devidamente despachadas a todos os modelos. Para aplicações de software de grande porte é inviável realizar tal procedimento de sincronização manualmente, portanto, é desejado a criação de métodos automáticos capazes de sincronizar os modelos do software.

Não exploramos este problema para qualquer tipo de software, ao invés disso limitamos nosso domínio ao espaço tecnológico Java, de maneira que o escopo deste trabalho permaneça factível. Esta tese procede portanto (1) identificando e definindo formalmente alguns modelos do espaço tecnológico Java, (2) identificando e formalizando algumas relações entre eles, criando uma rede de metamodelos a ser mantida sincronizada, e mostrando como estas relações trabalham através de um caso representativo, e finalmente (3) discutindo a sincronização desta rede de metamodelos. Os resultados incluem a implementação destas relações mais o relatório sobre a experiência de desenvolver isso neste trabalho de conclusão de curso.

Palavras-chave: Model Synchronization. Java Metamodels. Network of Models. Iterative Model Transformation. Model Transformation. Model-driven Engineering. Software Engineering.

RESUMO ESTENDIDO

Este é um resumo estendido em português para a Universidade Federal do Rio Grande do Sul. O trabalho de conclusão original, em inglês, foi apresentado na Technische Universität Berlin através do programa de dupla diplomação UNIBRAL II entre as duas universidades.

1 Introdução

Os últimos avanços relacionados a técnicas de engenharia de software incluem a chamada **engenharia orientada a modelos** (inglês: Model-driven Engineering ou MDE), sinônimo para **desenvolvimento orientado a modelos** (inglês: Model-driven Development ou MDD), onde modelos de software são os artefatos primários do desenvolvimento do software. Em MDE, um sistema de software tem possivelmente diversos modelos abstraindo-o, cada um representando certos aspectos de todo o sistema. Esses modelos têm relações entre si, no sentido de que devem descrever o sistema consistentemente, não apresentando contradições lógicas. Exemplos de modelos de software são *diagramas de classes UML*, *diagramas de casos de uso* ou até mesmo código-fonte. Muito embora o uso de modelos pode contribuir positivamente para o desenvolvimento de aplicações de software, ele também introduz alguns problemas, entre eles está o **problema de sincronização de modelos**, que consiste basicamente em manter todos os modelos de um software consistentes entre eles mesmos. Em outras palavras, ao passo que uma aplicação de software evolui, seus modelos sofrem alterações que devem ser encaminhadas corretamente a todos os outros modelos relacionados a esta aplicação (DISKIN, 2011).

Propõe-se então neste trabalho uma abordagem ao problema baseada na criação de uma **rede de modelos interconectados**, que deve ser mantida consistente, i.e. todos os modelos devem estar sincronizados. Para exemplificar, suponha um *diagrama de classes UML*, uma séria de *diagramas de sequência UML* e um código-fonte. Se um método tem seu nome alterado no diagrama de classes, todas as ocorrências deste método devem ter seus nomes atualizados nos diagramas de sequência e no código-fonte. Acontece, porém, que uma ferramenta de sincronização de modelo automática e genérica capaz de ser aplicada na prática não é conhecida por nós, apesar de que um esforço expressivo tem sido feito pela comunidade acadêmica para criar solução para este problema.

O objetivo amplo deste trabalho é explorar o problema de sincronização de mo-

delos, analisando modelos e suas relações com outros modelos e estabelecendo técnicas de sincronização. Todavia, para manter o escopo factível, restringimos o domínio ao **espaço tecnológico Java** e fixamos as seguintes metas: (1) apresentar e definir formalmente alguns **modelos** do espaço tecnológico Java; (2) formalizar e explicar algumas **relações** entre esses modelos, criando uma rede de metamodelos. e (3) discutir como **sincronização** pode ser alcançada nessa rede. Está fora, portanto, do escopo deste trabalho a criação de definições completas de metamodelos ou a implementação de um algoritmo totalmente funcional de sincronização, assim como uma análise teórica profunda do problema ou da performance das relações desenvolvidas. Contudo, os resultados deste trabalho contribuem para um melhor intendimento do problema de sincronização de modelos de espaços tecnológicos complexos.

2 Fundamentação

Abaixo está o resumo de algumas definições importantes para o restante do texto.

2.1 Modelos

De acordo com Favre (2004b), um sistema é o elemento de discurso principal ao falar de MDE. Um modelo é um sistema que representa um outro sistema. Um modelo é expresso usando uma linguagem de modelagem, que por sua vez nada mais é que um conjunto de todos os modelos expressados naquela linguagem. Por final, um metamodelo é então um modelo de uma linguagem de modelos, isto é ele especifica o que pode ser escrito usando uma certa linguagem. Um modelo M está em conformidade com um metamodelo MM, se e somente se, M pertence à linguagem especificada po MM (FAVRE, 2004a).

2.2 Sincronização de Modelos

Uma relação de modelos é definida aqui abstratamente como qualquer relacionamento ou restrição possível de acontecer entre um modelo fonte e um modelo alvo. Sincronização de modelos pode ser vista como uma função $s : M \times M \times \Delta_M \times N \times N \times \Delta_N \rightarrow M \times N$, onde $s(m_0, m_1, \delta_m, n_0, n_1, \delta_n) = (m_2, n_2)$ significa que os modelos sincroniza-

dos m_2 e n_2 são criados a partir dos modelos inicialmente sincronizados m_0 e n_o e os modelos modificados não sincronizados m_1 e n_1, considerando as modificações (respectivamente δ_m e δ_n) executadas sobre ambos (DISKIN, 2011). Uma rede de modelos de um sistema S é um grafo não-direcionado $G = (V, E)$, onde cada vértice $v_i \in V$ representa um modelo único i abstraindo S, e uma aresta $\{v_i, v_j\}$ existe se, e somente se, há uma relação entre ambos os modelos i e j.

2.3 Gramática de Grafos Triplos

Com o uso de um grafo triplo, uma relação entre um modelo fonte S e um modelo alvo T é abstraída em uma tripla (G^S, G^C, G^T) – onde G^S é a representação em grafo dos elementos do modelo fonte, G^T é a representação dos elementos do modelo alvo, e G^C representa a correspondência entre os dois conjuntos de elementos – juntamente com dois mapeamentos $s_G : G^C \rightarrow G^S$ e $t_G : G^C \rightarrow G^T$, que conectam os três grafos (HERMANN et al., 2011). Neste caso, uma adição no grafo triplo $G = (G^S, G^C, G^T)$, que leva a um novo grafo triplo $H = (H^S, H^C, H^T)$, consiste em um morfismo de grafo triplo $m : G \rightarrow H$, com $m = (m^S, m^C, m^T)$, de acordo com a Figura 1.

Figura 1: O morfismo $m : G \rightarrow H$ é um grafo triplo $m = (m^S, m^C, m^T)$.

$$\begin{array}{ccccc} G = (G^S & \xleftarrow{s_G} & G^C & \xrightarrow{t_G} & G^T) \\ {\scriptstyle m}\downarrow & {\scriptstyle m^S}\downarrow & & {\scriptstyle m^C}\downarrow & {\scriptstyle m^T}\downarrow \\ H = (H^S & \xleftarrow[s_H]{} & H^C & \xrightarrow[t_H]{} & H^T) \end{array}$$

Fonte: (HERMANN et al., 2011)

Uma regra tripla é um morfismo de grafo triplo $t = (t^S, t^C, t^T) : L \rightarrow R$. Um axioma triplo é uma regra tripla $t_a = (t^S, t^C, t^T) : \varnothing \rightarrow R$ (EHRIG et al., 2007). Uma gramática de grafo triplo(inglês: Triple Graph Grammar ou TGG) $TGG = (t_a, T_{rules})$ consiste de um axioma triplo t_a e um conjunto de regras triplas T_{rules} (GIESE; HILDEBRANDT; LAMBERS, 2010, p. 4). Grafos triplos são utilizados neste trabalho como descrição de uma relação entre dois metamodelos, enquanto que TGGs descrevem uma espécie de linguagem da relação consistente entre estes dois metamodelos. Entretanto, regras extras podem ser derivadas de uma TGG para criar a semântica operacional do procedimento de sincronização da relação (GIESE; HILDEBRANDT; LAMBERS, 2010).

3 Relações de Metamodelos no Espaço Tecnológico Java

Foi selecionado neste trabalho um conjunto de metamodelos típicos do espaço tecnológico Java e foram identificadas relações entre eles, criando-se assim uma **rede de metamodelos**. Dessa rede, quatro metamodelos e suas relações são trabalhadas mais profundamente, a saber *Java*, *UMLClassDiagram*, *UMLSequenceDiagram* e *UMLContract*.

Um modelo ***Java*** (i.e. o código-fonte) contem informações tanto estruturais quanto comportamentais, e por isso se relaciona com os outros três metamodelos analisados. O metamodelo *Java* criado não é completo, pois não modela toda a gramática da linguagem Java, entretanto aspectos arquiteturais de um programa orientado a objetos (e.g. classes, interfaces, métodos, atributos, etc.) são representados. O metamodelo ***UMLClassDiagram*** é construído a partir dos conceitos de classe, propriedade, operação, interface e pacote. Ele é usado para descrever a estrutura de um programa orientado a objetos e sua relação com o metamodelo *Java* é dado por uma tradução quase direta entre seus elementos.

Para representar alguns aspectos comportamentais, ***UMLSequenceDiagram*** é usado, dado que um diagrama de sequência representa tempos-de-vida de objetos e mensagens trocadas entre eles. No modelo *Java* isto pode ser representado por chamadas de métodos entre classes ou por anotações sobre métodos, representando a ordem em que as trocas de mensagens ocorrem. Portanto, na nossa abordagem, cada *linha-de-vida* (*lifeline*) representa o comportamento de um método Java e cada mensagem enviada a partir desta *linha-de-vida* é modelada em uma anotação (*Java annotation*) sobre o respectivo método, que codifica a metainformação da ordem de chamadas a outros métodos.

O metamodelo ***UMLContract*** é baseado nas ideias de programação por contrato, onde operações têm restrições (*constraints*) de pré e pós-condição assim como de invariante. Sua relação com *Java* é que cada restrição dos contratos pode ser (1) testada através de asserções (*assertions*) e (2) expressa em termos de anotações no modelo *Java*. Além disso, pode-se ter métodos de checagem de consistência interna de classe que verificam as restrições da classe e, portanto, devem ser mantidas sincronizadas com os modelos de *UMLContract* (i.e. os contratos). Desta maneira, as relações desenvolvidas neste trabalho estabelecem que cada restrição de *UMLContract* corresponde a uma anotação (sobre um método ou atributo) de *Java*, assim como uma asserção pertencente a um método de checagem da classe.

4 Sincronização de Modelos no Espaço Tecnológico Java

5 Conclusão

REFERÊNCIAS

DISKIN, Z. Model synchronization: Mappings, tiles, and categories. In: **Generative and Transformational Techniques in Software Engineering III**. [S.l.]: Springer, 2011. p. 92–165.

EHRIG, H. et al. Information preserving bidirectional model transformations. In: **Fundamental Approaches to Software Engineering**. [S.l.]: Springer, 2007. p. 72–86.

FAVRE, J.-M. Foundations of meta-pyramids: languages vs. metamodels. In: CITESEER. **Episode II. Story of Thotus the Baboon, Procs. Dagstuhl Seminar**. [S.l.], 2004. v. 4101.

FAVRE, J.-M. Foundations of model (driven)(reverse) engineering: Models. In: **Proceedings of the International Seminar on Language Engineering for Model-Driven Software Development, Dagstuhl Seminar 04101**. [S.l.: s.n.], 2004.

GIESE, H.; HILDEBRANDT, S.; LAMBERS, L. Toward bridging the gap between formal semantics and implementation of triple graph grammars. In: IEEE. **Model-Driven Engineering, Verification, and Validation (MoDeVVa), 2010 Workshop on**. [S.l.], 2010. p. 19–24.

HERMANN, F. et al. Correctness of model synchronization based on triple graph grammars. In: **Model Driven Engineering Languages and Systems**. [S.l.]: Springer, 2011. p. 668–682.