MoBRe: Refatoração de Modelos de Classes OO Anotados para Modelos de Classes OA

Paulo Afonso Parreira Júnior[1¥], Rosângela Aparecida Dellosso Penteado[1μ], Rhafik D. N. González[1], Valter Vieira de Camargo[1μ] Heitor Augustus Xavier Costa[2]

[1] Departamento de Computação – Universidade Federal de São Carlos
[2] Departamento de Ciência da Computação – Universidade Federal de Lavras

{paulo_junior, rosangela, valter}@dc.ufcar.br,
rhafik_daud@comp.ufscar.br, heitor@ufla.br

Abstract. *The automatic recovery of aspect-oriented (AO) models from object-oriented (OO) software is not a trivial task due to the difference between the concepts related to both approaches. In this paper, we propose the use of OO class models annotated with information of crosscutting concerns – UML class diagrams whose elements are stereotyped with indications of crosscutting concerns - to obtain AO class models. A computational support, called MoBRe (Model Based Refactoring), was developed to obtain AO class models from annotated OO class models through refactorings. The refactorings available in MoBRe are to following concerns: persistence, logging and the design patterns Singleton and Observer. MoBRe is an Eclipse plug-in and can be extended with new types of refactoring proposed by users.*

1. Introdução

A programação orientada a aspectos (POA) (Kiczales *et al*., 1997) tem como objetivo encapsular em unidades específicas chamadas 'aspectos', interesses (*concerns*) cuja implementação produz representações entrelaçadas e espalhadas pelos módulos funcionais do software. Tais interesses são conhecidos como Interesses Transversais (ITs) e sua identificação e modularização podem melhorar a qualidade do código fonte produzido e facilitar sua manutenção.

A reengenharia de sistemas Orientados a Objetos (OO) para sistemas Orientados a Aspectos (OA) tem como principal dificuldade a existência de diferentes abstrações no paradigma alvo (OA), fazendo com que seja necessária a existência de transformações das abstrações OO em abstrações OA. Uma técnica que pode ser usada são refatorações em nível de código-fonte (Silva *et al*. ,2009; Monteiro e Fernandes, 2006; Hannemann *et al*., 2005; Marin *et al*., 2005 e Iwamoto e Zhao, 2003). Entretanto, refatorações desse tipo possuem algumas características que as tornam inadequadas para um processo de reengenharia quando comparadas com refatorações em nível de modelos, tais como: i) em nível de código fonte as refatorações geralmente são feitas em apenas um passo, em que se tem como entrada o código OO e como saída o código OA. Isso faz com que esse processo tenha pouca flexibilidade, fazendo com que o sistema final tenha que seguir um determinado estilo arquitetural pré-definido. As refatorações com base em modelos introduzem, pelo menos, um passo a mais no processo antes de se obter o código fonte final. Esse passo intermediário consiste na obtenção de um modelo OA, em que o engenheiro de software pode refletir sobre ele e modificá-lo de acordo com os requisitos do ambiente operacional e necessidades dos *stakeholders*, introduzindo mais flexibilidade na transformação; ii) software legados, em

[¥] Apoio Financeiro CNPq pelo Proc no. 133140/2009-1
[μ] Apoio Financeiro CNPq pelo Proc no. 483106/2009-7

geral, possuem apenas o código fonte como artefato. Sendo assim, ao aplicar refatorações baseadas em código, tanto o software legado, quanto o novo software continuarão sem documentação; iii) diferentemente das refatorações baseadas em modelos, refatorações de código são dependentes da plataforma de implementação.

Neste artigo é apresentado um *plug-in* para o ambiente Eclipse, denominado MoBRe (*Model Based Refactoring*), que permite refatorar modelos de classes anotados com indícios de interesses transversais em modelos de classes orientados a aspectos. No contexto deste trabalho, "modelos de classes anotados" possuem estereótipos localizados em seus elementos (classes, interfaces, atributos e métodos) que simbolizam a presença de um determinado interesse transversal (Costa *et al*., 2009). Dessa forma, MoBRe permite que um modelo de classes afetado por interesses transversais seja transformado em um outro modelo de classes em que os interesses transversais encontra-se modularizados com aspectos. MoBre oferece suporte a três refatorações genéricas e seis específicas de interesses transversais para persistência (subdividida em gerenciamento de conexão, transação e sincronização), *logging* e para os padrões de projeto *Singleton* e *Observer* (Gamma *et al*., 1995).

Vale ressaltar que os modelos de classes anotados com indícios de interesses transversais são recuperados automaticamente a partir de código fonte Java utilizando-se dois apoios computacionais: ComSCId (*Computational Support for Concern Identification*) e DMAsp (*Design Model for Aspect*) (Parreira Júnior *et al*., 2010), mas que o enfoque deste artigo é apenas nas refatorações desses modelos anotados para modelos orientados a aspectos.

Uma breve descrição do ComSCId e do DMAsp é apresentada na Seção 2. O apoio computacional MoBRe é apresentado na Seção 3. Alguns trabalhos relacionados são comentados na Seção 4 e as considerações finais estão na Seção 5.

2. ComSCId e DMAsp

ComSCId é um *plug-in* Eclipse que possibilita a automatização da identificação de indícios de ITs em softwares implementados em Java. Para isso são utilizadas regras compostas por cadeias de caracteres ou tipos de dados. Possui também um repositório de regras que permite o armazenamento de novas regras, a remoção e a atualização das regras existentes para quaisquer ITs. DMAsp também é um *plug-in* para o Eclipse que gera modelos de classes anotados com indícios de ITs que foram identificados a partir do código fonte pelo ComSCId. Essa anotação é representada usando estereótipos localizados acima e do lado esquerdo do identificador de classes, interfaces, atributos e métodos, como mostrado na Figura 1.

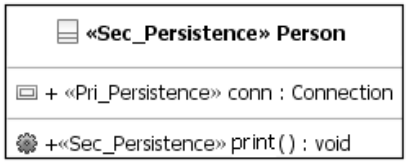

Figura 1. Classe UML Estereotipada com Indícios de IT.

As definições a seguir, propostas por Figueiredo *et al*. (2009), foram adaptadas ao contexto dos modelos de classes anotados gerados pelo DMAsp e são utilizadas nas refatorações do MoBRe: *a) interesse primário* é o interesse principal de um componente e está relacionado com a razão pela qual ele foi criado. No modelo de classes anotado os estereótipos relacionados a esses interesses primários são precedidos do prefixo "Pri_".

Por exemplo, o atributo `conn` da classe `Person` (Figura 1) foi criado para realizar conexão com o banco de dados, então "persistência" é o seu interesse primário (`<<Pri_Persistence>>`); e *b) interesse secundário de um componente* corresponde a determinadas funções que esse componente desempenha, mas que não estão diretamente relacionadas com a razão pela qual ele foi criado. No modelo de classes anotado os estereótipos relacionados aos interesses secundários são precedidos do prefixo "Sec_". Por exemplo, o método `print()` da classe `Person`, criado para imprimir os dados de uma pessoa na tela, pode possuir também um interesse secundário de persistência (`<<Sec_Persistence>>`).

A classificação dos interesses é realizada de forma automática pela ferramenta ComSCId, com o cadastro de um conjunto de regras para identificação de ITs Esse conjunto pode ser modificado se houver necessidade, pelo engenheiro de software. Após a identificação e classificação dos ITs, o *plug-in* DMAsp pode ser executado para obtenção do modelo de classes OO anotado. O DMAsp realiza a engenharia reversa do código Java, construindo modelos de classes da UML com anotações dos ITs a partir dos que foram identificados pelo ComSCId.

3. MoBRe – *Model-Based Refactoring*

MoBRe é um *plug-in* Eclipse capaz de transformar um modelo OO anotado em um modelo OA. Como pode ser visto na Figura 2, MoBRe trabalha de forma integrada ao ComSCId e DMAsp, permitindo: i) identificar indícios de interesses transversais em código fonte Java (ComSCId); ii) recuperar o modelo de classes OO anotado com os indícios identificados, com a utilização de estereótipos (DMAsp); e iii) aplicar refatorações sobre os modelos OO para construir modelos de classes OA (MoBRe). Além disso, MoBRe é um *plug-in* extensível, permitindo que novas refatorações possam ser adicionadas pelo engenheiro de software para atender as suas necessidades.

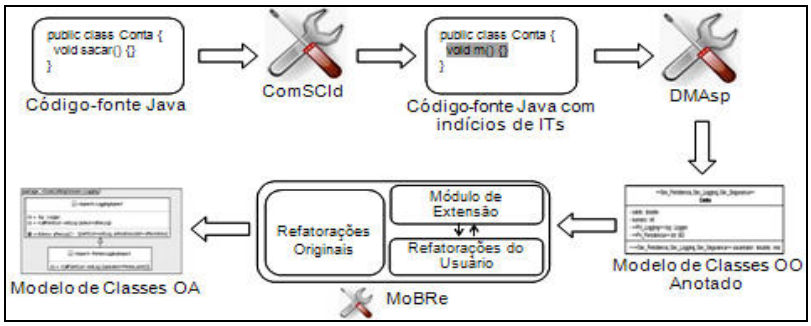

Figura 2. Arquitetura do *plug-in* MoBRe.

Na Figura 3 é apresentada a interface do MoBRe que é composta por dois menus localizados na barra de menu principal do Eclipse, *Refactoring* (1) e *My Refactoring* (2), e uma visão denominada *Refactoring View* (3).

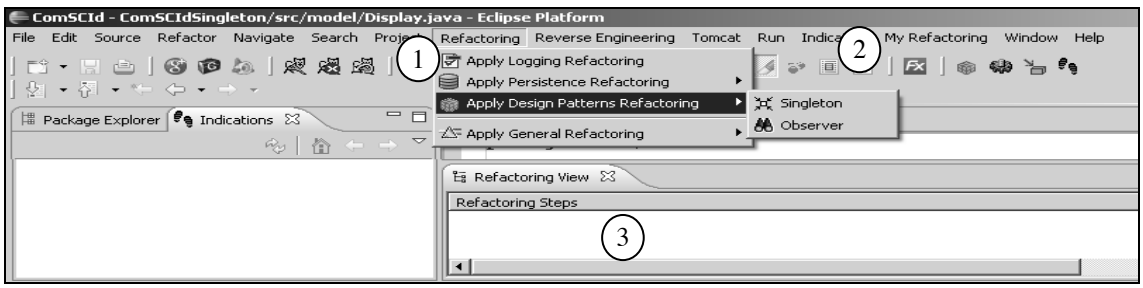

Figura 3. Interface do *plug-in* MoBRe.

O menu *Refactoring* disponibiliza os itens necessários para aplicação das refatorações sobre os modelos de classes OO anotados. MoBRe disponibiliza dois tipos de refatorações: genéricas (aplicáveis a qualquer tipo de IT) e específicas do tipo de IT (possuem um conjunto de passos específicos para modularização de um determinado tipo de interesse). As refatorações presentes no menu *Refactoring* e disponibilizadas originalmente com o *plug-in* são três tipos de refatorações genéricas R-1 (Entrelaçamento sem Generalização/Especialização), R-2 (Entrelaçamento com Generalização/Especialização) e R-3 (Entrelaçamento sem Interesse Primário) e seis específicas, R-*Connection*, R-*Transaction*, R-*Syn*, R-*Logging*, R-*Singleton* e R-*Observer*. O menu *My Refactoring* contém refatorações implementadas por usuários. Para elaboração dessas refatorações foi realizado um estudo prospectivo. As refatorações propostas foram suficientes para modularização dos interesses existentes nessas aplicações, porém é necessário um estudo mais aprofundado para averiguar a abrangência do conjunto de refatorações proposto para outras aplicações e domínios. Em virtude da restrição de espaço, a descrição dessas refatorações foi omitida deste artigo, detalhes sobre elas podem ser obtidas em Parreira Júnior (2011).

Além dos menus, MoBRe possui uma visão, *Refactoring View*, responsável por exibir os passos aplicados sobre o modelo de classes OO anotado quando uma refatoração, genérica ou específica, é aplicada. Esses passos descrevem as modificações realizadas no modelo de classes OO para transformá-lo em um modelo OA.

3.1 Executando Refatorações no MoBRe

Os passos necessários para uso do MoBRE são exemplificados pela aplicação de uma refatoração genérica sobre um modelo de classes OO anotado.

1. **Executar o *plug-in* ComSCId** em um código Java para identificar os indícios de ITs. Após a identificação dos indícios dos ITs existentes no código fonte, as refatorações já podem ser aplicadas. Entretanto, as refatorações específicas implementadas no MoBRe foram desenvolvidas para serem executadas após a aplicação das refatorações genéricas, por isso, é necessário descobrir qual refatoração aplicar.
2. **Gerar modelo de classes OO anotado**. Para saber qual refatoração genérica deve ser aplicada, o usuário precisa observar o modelo de classes OO anotado com indícios de interesses transversais, obtido pelo DMAsp.

Considerando o modelo de classes apresentado na Figura 1, ele apresenta um cenário de entrelaçamento/espalhamento do interesse transversal de persistência apropriado para a aplicação da refatoração R-3. A refatoração genérica R-3 deve ser aplicada quando há classes com somente com Interesses Secundários.

Caso o usuário não saiba identificar os cenários relacionados a cada refatoração genérica, ele pode clicar no *menu Refactoring -> Apply General Refactoring* e selecionar a opção *Analyse OO Class Model*. O modelo de classes do software será analisado pelo MoBRe, que apresentará uma mensagem descrevendo as possíveis refatoração para cada tipo de interesse. A análise do modelo de classes OO anotado é feita com base na configuração dos interesses existentes nesse modelo. As informações necessárias para visualização do modelo de classes OO anotado, como classes/interfaces, atributos, métodos e ITs, são armazenadas em um arquivo XML. Dessa forma, para identificar o cenário adequado para aplicação da refatoração R-3, o *plug-in* MoBRe percorre esse arquivo procurando por interesses classificados como

secundários de algumas classes e verificando se esses não são primários em outras classes. Se o resultado dessa análise for positivo, significa que há um cenário de entrelaçamento adequado para aplicação da refatoração R-3. Para o modelo apresentado na Figura 1, a mensagem exibida ao usuário é a apresentada na Figura 4.

3. **Aplicar um tipo de refatoração genérica**. Após identificar qual refatoração genérica aplicar, o usuário deve, clicar sobre o *menu Refactoring -> Apply General Refactoring* e selecionar a opção correspondente à refatoração desejada. No exemplo em questão, ao clicar na opção *Apply Refactoring 3*, a caixa de diálogo da Figura 5 será exibida.

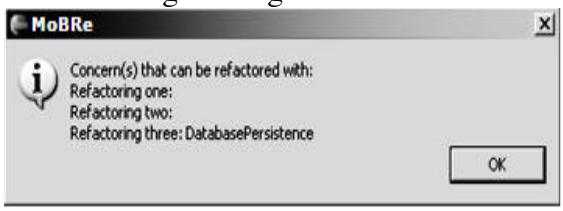

Figura 4. Refatorações Passíveis de serem Aplicadas.

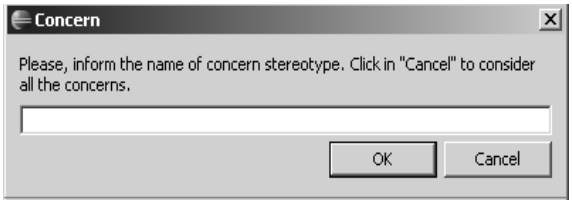

Figura 5. Caixa de Diálogo para Informar o Interesse a ser Analisado.

O usuário deverá informar para qual tipo de interesse ele deseja aplicar a refatoração escolhida. Essa característica é importante, pois quando há vários interesses no modelo OO anotado que podem ser refatorados com uma determinada refatoração, o usuário pode escolher aplicá-la a cada interesse individualmente. Neste exemplo, como existe apenas um interesse sobre o qual se deseja aplicar a refatoração R-3, basta clicar em "Cancel". O modelo resultante dessa aplicação pode ser visualizado na Figura 6. Além disso, após a aplicação de qualquer refatoração, a visão de refatorações (Figura 7) é atualizada com os passos realizados pela refatoração.

Figura 6. Modelo OA Parcial Obtido Após a Aplicação da Refatoração 3.

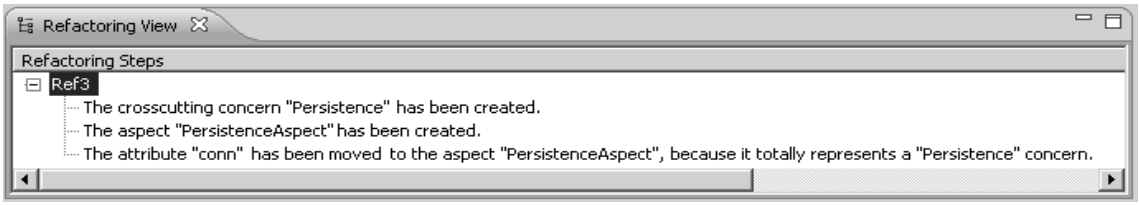

Figura 7. Passos Executados pela Refatoração 3.

4. **Aplicar um tipo de refatoração específica**. Após ter aplicado a refatoração genérica, percebe-se que o método `print()` da classe `Person` continua com o estereótipo `<<Sec_Persistence>>`. Para solucionar esse problema, o engenheiro de software pode aplicar uma refatoração específica para o interesse de persistência. A aplicação dessa refatoração não é apresentada em consequência da limitação de espaço.

Salienta-se que a obtenção de um modelo OA final deve ser realizada sob uma perspectiva iterativa e incremental. Dessa forma, usuários podem aplicar refatorações sobre cada interesse existente no software individualmente, observando, por meio do modelo de classes e da visão de refatorações, as modificações realizadas e progredindo

até que todos os interesses identificados pela e ferramenta e que possuam refatorações sejam refatorados. Além disso, caso seja necessário, pode-se reiniciar o processo por um interesse diferente ou aplicando outros tipos de refatorações.

É importante comentar ainda que após a aplicação das refatorações disponíveis no MoBRe não é feita checagem da preservação de comportamento do software, nem de sua estrutura. Para realizar checagem comportamental seriam necessários outros tipos de artefatos de software como diagramas de sequência ou código fonte, que não são gerados pela ferramenta. A checagem estrutural do software com base nos diagramas de classes OO e OA não é realizada nesta versão da ferramenta, e será avaliada em trabalhos futuros.

3.2 Estendendo o MoBRe

A elaboração de refatorações específicas para todos os tipos de interesses transversais não seria possível. Dessa forma, foi construído um módulo que permite ao usuário acrescentar novos tipos de refatorações genéricas e/ou específicas. Por exemplo, um engenheiro de software que trabalha com refatorações para interesse transversais de segurança pode fazer o *download* do MoBRe e implementar a sua própria refatoração para esse tipo de interesse, utilizando como suporte a API do *plug-in*. Essa API oferece suporte para implementação de consultas, alterações, remoções e/ou adições de elementos em um determinado modelo de classes OO utilizando a linguagem de programação Java.

Após implementar a refatoração desejada, basta gerar uma nova versão do *plug-in* MoBRe no ambiente Eclipse, que as modificações realizadas pelo usuário serão incorporadas automaticamente e organizadas no menu *My Refactoring* (destaque 2 na Figura 3). Além disso, com MoBRe o usuário pode implementar novos tipos de refatorações para um interesse já contemplado no *plug-in* (por exemplo, persistência), podendo assim, comparar duas ou mais soluções de modularização para um mesmo tipo de interesse.

4. Trabalhos Relacionados

Vários trabalhos têm sido propostos para refatoração de software OO para software OA, somente em nível de código: Silva *et al*. (2009); Monteiro e Fernandes (2006); Hannemann *et al*. (2005); Marin *et al*. (2005) e Iwamoto e Zhao (2003). Porém, notou-se escassez de trabalhos relacionados à refatoração em nível de modelos.

Boger *et al*. (2002) desenvolveram um *plug-in* para a ferramenta ArgoUML que apoia a refatoração de modelos UML: diagramas de classes, de estados e de atividades. Desse modo, o usuário pode realizar refatorações que não são simples para serem aplicadas em nível de código, por exemplo, substituir um relacionamento de herança por delegação. Van Gorp *et al*. (2003) propuseram um perfil UML para expressar pré e pós-condições de refatoração de código fonte utilizando restrições OCL (*Object Constraint Language*). Esse perfil permite que uma ferramenta CASE: i) verifique pré e pós-condições para composição de sequências de refatorações; e ii) use o mecanismo de consulta OCL para detectar *bad smells*. O diferencial deste trabalho em relação aos demais é a proposta de construção de um modelo OA considerando modelos de classes OO anotados com estereótipos representando interesses transversais.

5. Considerações Finais

Neste artigo foi apresentado o apoio computacional MoBRe que auxilia desenvolvedores/mantenedores na tarefa refatoração de software OO para OA com a geração de modelos de classes. A geração de um modelo de classes OA com MoBRe pode trazer benefícios para o processo de reengenharia de software, como: (i) facilitar a visualização dos interesses transversais existentes no software e de seu nível de espalhamento/entrelaçamento com os demais módulos do software; (ii) permitir ao engenheiro de software articular e raciocinar como projetar os interesses transversais identificados; e (ii) minimizar o *gap* semântico existente entre o código fonte de um software OO e um modelo OA.

Atualmente, MoBRe gera apenas diagramas de classes, porém, como trabalho futuro, pretende-se: i) estender esse apoio computacional para permitir a obtenção de outros diagramas, por exemplo o de sequência, bem como para geração de código fonte OA; ii) realizar a checagem estrutural do software após a aplicação das refatorações; e iii) realizar a avaliação da ferramenta, bem como das refatorações propostas por meio de experimentos controlados. Salienta-se que os apoios computacionais apresentados neste artigo são softwares livre e estão disponíveis para download por meio do link: http://www.dc.ufscar.br/~paulo_junior.

Referências Bibliográficas

Boger, M.; Sturm, T.; Fragemann, P. "Refactoring Browser for UML". In: Revised Papers From The International Conference Netobjectdays On Objects, Components, Architectures, Services, And Applications For A Networked World – NODe'02, 2002.

Costa, H. A. X. ; Parreira Júnior, P. A. ; Camargo, V. V. ; Penteado, R. A. D. . "Recovering Class Models Stereotyped With Crosscutting Concerns". In: Session Tool of XVI Working Conference on Reverse Engineering (WCRE), Lille, França. 2009.

Figueiredo E., Sant'Anna C., Garcia A., and Lucena C. Applying and Evaluating Concern-Sensitive Design Heuristics. In proceedings of the 23rd Brazilian Symposium on Software Engineering (SBES). Fortaleza, 2009.

Gamma, E., Helm, R., Johnsn, R., Vlisside, J. "Design Patterns: Elements of Reusable Object-Oriented Software". Addison-Wesley, 1995.

Hannemann, J.; Murphy, G. C.; Kiczales, G. "Role-based refactoring of crosscutting concerns". In: Aspect-Oriented Software Development - AOSD, New York, USA, 2005.

Iwamoto, M.; Zhao, J. "Refactoring Aspect-Oriented Programs". In: International Workshop On Aspect-Oriented Modeling With UML, Boston, USA, AOSD 2003.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J., Irwin, J. "Aspect-Oriented Programming". 11th European Conference on Object-Oriented Programming.LNCS, v. 241, 1997.

Marin, M.; Moonen, L.; Van Deursen, A. "An Approach to Aspect Refactoring based on Crosscutting Concern Types". Software Engineering Notes, v.30, n.4, 2005.

Monteiro, M. P.; Fernandes, J. M. L. "Towards a catalogue of refactorings and code smells for aspect". Transactions on Aspect Oriented Software Development (TAOSD) - LNCS, n.3880, 2006.

Parreira Júnior, P. A. *et al*. "ComSCId & DMAsp: Identificação de Interesses Transversais e Recuperação de Modelos de Classes Anotados a partir Softwares OO Java". In: Sessão de Ferramentas do CBSoft, 2010, Salvador - BA. 2010.

Parreira Júnior, P. A. "Recuperação de Modelos de Classes Orientados a Aspectos a partir de Sistemas Orientados a Objetos usando Refatorações de Modelos". Dissertação de Mestrado. Departamento de Computação – Universidade Federal de São Carlos (UFSCar). 2011.

Silva, B. *et al*. "Refactoring of Crosscutting Concerns with Metaphor-Based Heuristics". Electronic Notes in Theoretical Computer Science (ENTCS), vol. 233, 2009.

Van Gorp, P. et al. "Towards Automating Source Consistent UML Refactorings". In: The Unified Modeling Language Conference – UML 2003. 144-158.