Avaliação de uma Linguagem de Definição de Regras de Projeto no Contexto da Evolução de Sistemas Orientados a Aspectos

Lidiany Cerqueira Santos
Orientador: Alberto Costa Neto

lidianycs@gmail.com, alberto@ufs.br
Mestrado - Programa de Pós-Graduação em Ciência da Computação
Universidade Federal de Sergipe
Ingresso: Março de 2013
Previsão de conclusão: Fevereiro de 2015

Etapas futuras:

- exame de qualificação - Julho de 2014
- defesa - Janeiro de 2015.

Abstract: *Aspect-oriented development allows to separate and encapsulate crosscutting-concerns that cannot be effectively modularized when using for example, object-oriented programming. AOP may be used as a technique for introducing variations to customizing systems. However, problems inherent to the evolution of aspect-oriented software, can become obstacles in system updates. This work aims to assess a language for specification of design rules that can assist the developers in the evolution of aspect-oriented systems. As result, in addition to evaluation, it also intends to provide a solution for a real problem.*

Palavras-chave: programação orientada a aspectos, regras de projeto, evolução de sistemas.

Avaliação de uma Linguagem de Definição de Regras de Projeto no Contexto da Evolução de Sistemas Orientados a Aspectos

Lidiany Cerqueira Santos[1]
Alberto Costa Neto[1]

[1] Departamento de Computação – Universidade Federal de Sergipe
Av. Marechal Rondon, Jardim Rosa Elze, 49100-000 – São Cristóvão – SE – Brasil

lidianycs@gmail.com,alberto@ufs.br

1. Introdução

Com a proposta de modularizar e encapsular corretamente os requisitos transversais (*crosscutting concerns*), Kiczales apresenta a POA [Kiczales et al. 1997], a qual utiliza Aspectos como uma forma de modularizar esses requisitos. A programação orientada a objetos não permite uma modularização adequada de interesses transversais de um software [Kiczales et al. 2001], essa limitação ocorre quando a implementação de funcionalidades tende a se espalhar pelo código, entrelaçando-se com a implementação de outras funcionalidades, aumentando o acoplamento e reduzindo a coesão do software. Na POA esses requisitos são tratados como uma dimensão distinta dos demais requisitos do sistema. Através da sua integração à POO, é possível modularizar os interesses transversais e consequentemente melhorar o projeto e facilitar a manutenção do software.

Entretanto, apesar dos benefícios apresentados, a POA pode introduzir problemas como o alto acoplamento entre classes e aspectos [Kellens et al. 2006, Rashidi and Alexander 2008, Kojima and Aotani 2013], a quebra de modularidade de classes [Fernandes et al. 2009, Steimann et al. 2010, Gasiunas et al. 2011, Rebelo et al. 2013] e restringir o desenvolvimento em paralelo [Rebelo et al. 2013, Neto et al. 2013]. Com a finalidade de tratar estes problemas, [Neto 2010] apresenta uma linguagem para especificação de regras de projeto. A adoção desta especificação proporciona uma melhoria na modularidade dos sistemas orientados a aspectos (OA) implementados com AspectJ [1], através da especificação de uma interface com a estrutura e o comportamento essencial que cada componente desenvolvido deve fornecer. Porém, a linguagem proposta ainda precisa de uma avaliação maior em outros cenários e sistemas.

Diante do cenário apresentado, pretende-se avaliar os benefícios da adoção de uma linguagem de especificação de regras de projeto (LSD), proposta por [Neto 2010] em seu trabalho de doutorado, possibilitando a avaliação deste trabalho em um sistema real, bem como fornecendo uma solução para os problemas encontrados na adaptação e atualização de Sistemas Integrados de Gestão (SIG) [2], como vem ocorrendo em Instituições Federais de Ensino Superior (IFES) [Passos et al. 2013].

[1] http://eclipse.org/aspectj
[2] http://info.ufrn.br/wikisistemas/

2. Fundamentação Teórica e Trabalhos Relacionados

De acordo com Kiczales [Kiczales et al. 2001], a POA fez pelos interesses transversais o mesmo que a POO fez por um objeto, com o encapsulamento e a herança, fornecendo mecanismos de linguagem para capturar explicitamente requisitos transversais. Através da POA, interesses transversais podem ser escritos de forma modular, possibilitando a escrita de código mais simples e mais fácil de desenvolver e manter, e ainda com maior potencial de reutilização.

Em um estudo conduzido por [Passos et al. 2013] foram levantadas técnicas e abordagens adequadas para a introdução de variações no SIG desenvolvido pela Universidade Federal do Rio Grande do Norte (UFRN), o qual vem sendo adquirido por diversas IFES do Brasil. Dentre as técnicas levantadas, observou-se que a adoção da programação OA trouxe impactos positivos para a customização do SIG. Como principal benefício, a POA possibilita a alteração da estrutura estática e dinâmica de sistemas sem a necessidade de alterar o código base do SIG, mantendo-o intacto.

Por outro lado, alguns problemas que podem se tornar obstáculos durante a atualização para novas versões de um sistema, foram identificados, como por exemplo: a mudança semântica, a complexidade de manutenção e a fragilidade de *pointcuts* foram discutidas por [Stoerzer and Graf 2005, Anbalagan and Xie 2006, Kellens et al. 2006, Sakurai and Masuhara 2007, Rashidi and Alexander 2008, Bynens et al. 2011b, Rebelo et al. 2013, Brichau et al. 2007, Sakurai and Masuhara 2008, Wloka et al. 2008, Kojima and Aotani 2013]. O alto acoplamento, a violação de integridade e a quebra de modularidade entre classes e aspectos foram tratados por [Fernandes et al. 2009, Camilleri et al. 2009, Chiba et al. 2010, Steimann et al. 2010, Gasiunas et al. 2011, Bynens et al. 2011a, Rebelo et al. 2013]. A restrição ao desenvolvimento em paralelo é um problema abordado por Rebelo [Rebelo et al. 2013]. Estes trabalhos buscam garantir maior estabilidade para a evolução de sistemas orientados a aspectos e conseguem minimizar a ocorrência destes problemas. Contudo, a exceção de [Kellens et al. 2006, Fernandes et al. 2009], os demais trabalhos não possibilitam a verificação de conformidade ao longo da evolução dos sistemas. Estes dois trabalhos, tratam dos problemas a nível de modelos conceituais. Em um levantamento sistemático de literatura realizado, observou-se que, apesar de detectados e analisados desde 2005, ainda não se encontrou uma solução definitiva para estes problemas, e por isso continuam sendo objetos de estudo.

A proposta apresentada por [Neto et al. 2013] é uma linguagem para especificação de regras de projeto (LSD) com o objetivo principal de apoiar o desenvolvimento modular de classes e aspectos, permitindo a implementação de interesses transversais sem quebrar a modularidade de classes. De acordo com [Neto et al. 2013], nenhuma abordagem prévia apresenta o propósito específico de descrever regras de design em sistemas orientados a aspectos. A linguagem é implementada através de uma extensão do abc (*AspectBench Compiler*) [3] e possibilita a especificação da estrutura e do comportamento essencial que cada componente desenvolvido deve fornecer. Uma regra de projeto contém um conjunto de restrições que devem ser seguidas por componentes que declaram implementá-las. Estas restrições são verificadas automaticamente por uma ferramenta (analisador estático) que alerta quando alguma restrição é desrespeitada [Neto et al. 2013].

[3] `http://abc.comlab.ox.ac.uk/`

A LSD tem como proposta auxiliar os desenvolvedores a escrever as regras de projeto sem ambiguidades, prover a modularidade de interesses transversais sem quebrar a modularidade de classes, reduzir a complexidade encontrada em outras abordagens e ainda possibilitar o desenvolvimento independente de classes e aspectos, após a definição das regras de projeto. Sendo esta última, uma característica importante e necessária para atualização do SIG, uma vez que as atualizações de versões produzidas pela UFRN são desenvolvidas de forma paralela e totalmente independente das variações introduzidas pelos desenvolvedores da Universidade Federal de Sergipe (UFS). A linguagem possui ainda uma semântica definida e suporte à verificação automática de conformidade com o código.

Um estudo conduzido por [Passos et al. 2013] apresentou um levantamento e catalogação de variações introduzidas pela UFS no código base da UFRN. Com base neste levantamento, foram implementadas algumas variações através de Aspectos codificados em AspectJ. Observou-se que todas as variações encontradas e contabilizadas puderam ser convertidas em aspectos com comportamento equivalente. Contudo, observou-se que devido aos problemas evolutivos inerentes à POA, existem alguns riscos associados à sua adoção, como por exemplo mudanças no código fonte da UFRN, causando o desaparecimento ou o surgimento inesperado de um join point. Diante disso, identificou-se a possibilidade de adotar a LSD com o objetivo de mitigar esses riscos. Os benefícios apresentados pela LSD demonstram que esta é uma abordagem interessante para lidar com os problemas inerentes a atualizações necessárias ao longo da evolução de SIG's, contudo ainda carece de uma avaliação formal em outros cenários. Esta linguagem requer a criação de novos artefatos e exige certa experiência dos projetistas de software, bem como o conhecimento das novas construções da linguagem. Por isso, a importância de avaliar a LSD apresentada no contexto descrito.

3. Estado atual

Foi realizado um levantamento sistemático de obstáculos para a evolução de sistemas orientados a aspectos e das propostas existentes para solucionar esses problemas. Está sendo configurado um ambiente de desenvolvimento necessário para a condução dos experimentos, semelhante ao adotado atualmente pela UFS, utilizando-se o código base da UFRN e as variações catalogadas e produzidas pela UFS para customização do SIG. Ao ambiente de desenvolvimento será integrado o compilador implementado por [Neto 2010] para permitir a definição das regras de projeto necessárias para implementação das variações no cenário atual. Este ambiente, versões e variações serão utilizados para a especificação das interfaces através da LSD adotada e para a realização de um experimento, no qual será realizada a atualização do sistema para novas versões mediante a utilização das especificações projetadas.

4. Desenvolvimento necessário para a conclusão

- Especificação das interfaces relacionadas às variações do SIG utilizando a linguagem de regras de projeto.
- Definir e realizar um novo experimento controlado com objetivo de atualizar o SIG para novas versões e avaliar a especificação projetada com a LSD e seu impacto sobre a evolução do SIG.

- Coletar e validar os dados dos experimentos, analisar os resultados e empacotar os dados.

5. Avaliação dos resultados

O estágio atual deste trabalho encontra-se na fase de estudo da linguagem para especificação de regras de projeto e configuração do ambiente de desenvolvimento para realização dos experimentos. Foi conduzido um levantamento sistemático de literatura envolvendo os obstáculos e soluções para evolução de sistemas OA, bem como um estudo necessário para embasar o desenvolvimento deste trabalho. Espera-se publicar os resultados das atividades conduzidas até o momento, e como resultado final, além da avaliação da linguagem no contexto descrito, apresentar uma abordagem para melhorar o processo de atualização, e consequentemente auxiliar na adaptação e na evolução de SIG's.

References

Anbalagan, P. and Xie, T. (2006). Apte: Automated pointcut testing for aspectj programs. In *Proceedings of the 2Nd Workshop on Testing Aspect-oriented Programs*, WTAOP '06, pages 27–32, New York, NY, USA. ACM.

Brichau, J., Kellens, A., Gybels, K., Mens, K., Hirschfeld, R., and DHondt, T. (2007). Application-specific models and pointcuts using a logic meta language. In Meuter, W., editor, *Advances in Smalltalk*, volume 4406 of *Lecture Notes in Computer Science*, pages 1–22. Springer Berlin Heidelberg.

Bynens, M., Truyen, E., and Joosen, W. (2011a). A sequence of patterns for reusable aspect libraries with easy configuration. In Apel, S. and Jackson, E., editors, *Software Composition*, volume 6708 of *Lecture Notes in Computer Science*, pages 68–83. Springer Berlin Heidelberg.

Bynens, M., Truyen, E., and Joosen, W. (2011b). A system of patterns for reusable aspect libraries. In Katz, S., Mezini, M., Schwanninger, C., and Joosen, W., editors, *Transactions on Aspect-Oriented Software Development VIII*, volume 6580 of *Lecture Notes in Computer Science*, pages 46–107. Springer Berlin Heidelberg.

Camilleri, A., Coulson, G., and Blair, L. (2009). Cif: A framework for managing integrity in aspect-oriented composition. In Oriol, M. and Meyer, B., editors, *Objects, Components, Models and Patterns*, volume 33 of *Lecture Notes in Business Information Processing*, pages 18–36. Springer Berlin Heidelberg.

Chiba, S., Igarashi, A., and Zakirov, S. (2010). Mostly modular compilation of crosscutting concerns by contextual predicate dispatch. In *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA '10, pages 539–554, New York, NY, USA. ACM.

Fernandes, V., Delicato, F., and Pires, P. (2009). Crossmda2: Uma abordagem baseada em modelos para gerência de evoluçao de pointcuts. In *SBCARS09: Proceedings of 2nd Brazilian Symposium on Components, Architecture and Software Reuse.*

Gasiunas, V., Satabin, L., Mezini, M., Núñez, A., and Noyé, J. (2011). Escala: Modular event-driven object interactions in scala. In *Proceedings of the Tenth International Conference on Aspect-oriented Software Development*, AOSD '11, pages 227–240, New York, NY, USA. ACM.

Kellens, A., Mens, K., Brichau, J., and Gybels, K. (2006). Managing the evolution of aspect-oriented software with model-based pointcuts. In Thomas, D., editor, *ECOOP 2006 Object-Oriented Programming*, volume 4067 of *Lecture Notes in Computer Science*, pages 501–525. Springer Berlin Heidelberg.

Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., and Griswold, W. G. (2001). An overview of aspectj. pages 327–353. Springer-Verlag.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., marc Loingtier, J., and Irwin, J. (1997). Aspect-oriented programming. In *ECOOP*. SpringerVerlag.

Kojima, T. and Aotani, T. (2013). Hierarchical concern-based pointcuts. In *Proceedings of the 8th International Workshop on Advanced Modularization Techniques*, AOAsia '13, pages 19–22, New York, NY, USA. ACM.

Neto, A. C. (2010). *Specifying Design Rules in Aspect-Oriented Systems*. PhD thesis, Universidade Federal de Pernambuco.

Neto, A. C., Bonifácio, R., Ribeiro, M., Pontual, C. E., Borba, P., and Castor, F. (2013). A design rule language for aspect-oriented programming. *Journal of Systems and Software*, 86(9):2333 – 2356.

Passos, F. A., Santos, J., and C, N. A. (2013). Adaptacao e manutencao de sistemas integrados de gestao apoiados pela programacao orientada a aspectos. In *In Proceedings of The IX Simpósio Brasileiro de Sistemas de Informação - SBSI 2013*.

Rashidi, P. and Alexander, R. T. (2008). Onspect: Ontology based aspects. In *Proceedings of the 7th FOAL*, FOAL '08, pages 41–41, New York, NY, USA. ACM.

Rebelo, H., Leavens, G. T., Lima, R. M. F., Borba, P., and Ribeiro, M. (2013). Modular aspect-oriented design rule enforcement with xpidrs. In *Proceedings of the 12th FOAL*, FOAL '13, pages 13–18, New York, NY, USA. ACM.

Sakurai, K. and Masuhara, H. (2007). Test-based pointcuts: A robust pointcut mechanism based on unit test cases for software evolution. In *Proceedings of the 3rd Workshop on Linking Aspect Technology and Evolution*, LATE '07, New York, NY, USA. ACM.

Sakurai, K. and Masuhara, H. (2008). Test-based pointcuts for robust and fine-grained join point specification. In *Proceedings of the 7th International Conference on Aspect-oriented Software Development*, AOSD'08, pages 96–107, New York, NY, USA. ACM.

Steimann, F., Pawlitzki, T., Apel, S., and Kästner, C. (2010). Types and modularity for implicit invocation with implicit announcement. *ACM Trans. Softw. Eng. Methodol.*, 20(1):1:1–1:43.

Stoerzer, M. and Graf, J. (2005). Using pointcut delta analysis to support evolution of aspect-oriented software. In *Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on*, pages 653–656.

Wloka, J., Hirschfeld, R., and Hänsel, J. (2008). Tool-supported refactoring of aspect-oriented programs. In *Proceedings of the 7th International Conference on Aspect-oriented Software Development*, AOSD '08, pages 132–143, New York, NY, USA. ACM.