UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO
PROPOSTA DE TRABALHO DE GRADUAÇÃO

CONSOLIDANDO *DESIGN BY CONTRACT* ATRAVÉS DO *ASPECTJML WEB*

ALUNO: IGOR VINÍCIUS PINHEIRO CORDEIRO LEÃO
ORIENTADOR: HENRIQUE REBÊLO

RECIFE, ABRIL DE 2017

SUMÁRIO

RESUMO E CONTEXTUALIZAÇÃO

Com o surgimento da programação orientada a objetos, cresce naturalmente uma preocupação a respeito da confiabilidade dos sistemas escritos neste paradigma, onde por confiabilidade entende-se a combinação de corretude e robustez: a ausência de bugs [1]. Essa preocupação é justificada por uma das ideias centrais do paradigma, o reuso. A reutilização de um componente não confiável em larga escala afeta não apenas o próprio componente, mas o funcionamento de vários sistemas. Desta forma, para abordar sistematicamente essa questão, surge a teoria de *Design by Contract* (DbC).

O *Design by Contract* se desenvolve como uma nova forma de tratar a confiabilidade. Ele introduz o conceito de contrato, um conjunto de restrições de comportamento que devem ser atendidas antes, durante e após a execução de uma determinada rotina, através das pré-condições, declarações de invariante e pós-condições. Se explorarmos um pouco esses conceitos, nas pré-condições temos a definição das propriedades que devem ser satisfeitas por quem chama uma determinada rotina para que ela possa ser satisfatoriamente executada, a exemplo de um parâmetro de entrada que deve ter um valor válido. Já nas pós-condições temos a definição de um conjunto de restrições que devem ser garantidas pela rotina após a sua execução, a exemplo da real atualização de determinado atributo. Por fim, nas declarações de invariante temos as propriedades que devem ser sempre verdadeiras independente da chamada [1].

A ideia de fazer a verificação dos contratos em tempo de execução ficou popular nos anos 80 com o *Eiffel* [7], ganhando ao longo dos anos representatividade por meio de várias outras linguagens de *DbC* como o *PyContract* [5] e o *JML* [6], sendo esta última uma especificação para Java. Enquanto essas linguagens se desenvolviam, por outro lado, a literatura reconheceu que o contrato seria um interesse transversal, sendo melhor modelado por meio de aspectos [3]. Tomaremos aqui como base o universo Java e, por isso, analisaremos a implementação de contratos modelados como aspectos através da extensão mais famosa de Java ao paradigma, o *AspectJ* [4]. Modelar contratos com *AspectJ*, entretanto, traz consigo alguns novos problemas.

Um primeiro problema seria o raciocínio modular [2]. O raciocínio modular se caracteriza pela capacidade de tomar decisões a respeito de um módulo olhando apenas para sua implementação. Por sua própria estrutura, os *adivices* em *AspectJ* são aplicados sem explicitamente serem referenciados por um módulo, afetando o raciocínio modular. O raciocínio modular, entretanto, é um princípio defendido pelo *Design by Contract* [1] e por isso um impasse é criado. Ao modelar contratos como um interesse transversal através do *AspectJ* acabamos contrariando uma primitiva do *DbC*.

Um segundo problema seria a nível de documentação [2]. Por via de regra, a exemplo de linguagens como *Eiffel* e *JML*, as pré-condições, pós-condições e declarações de

invariante são declaradas em, ou próximas, o código que elas estão especificando, aumentando a documentação do sistema. Em *AspectJ* isso não é possível.

Para resolver esses problemas, Rebêlo et al. propuseram o *AspectJML* [2], uma extensão do *JML* ao paradigma orientado a aspectos, mantendo a separação de interesses e evitando incorrer nos dois problemas acima citados.

Este trabalho, então, assume a responsabilidade de estender o *AspectJML* a categoria de serviço, fazendo uma contribuição à linguagem por tornar possível que códigos *AspectJML* sejam compilados e executados remotamente através da *World Wide Web*.

OBJETIVOS

Rebêlo et Al. criaram o *AspectJML* como uma extensão orientada a aspectos do *JML*, mantendo qualidades positivas tanto do JML quando da orientação a aspectos. A linguagem, por exemplo, já é adotada para o ensino de *DbC*. A medida que a ela foi se consolidando, observou-se a necessidade de aumentar seu poder de penetração tanto na comunidade acadêmica como na indústria. Este trabalho, então, se compromete a revisar de maneira sistemática o *AspectJML*, analisando em detalhes os fatores que motivaram sua definição, incluindo seus pontos positivos e eventuais problemas. Por fim, como resultado desse processo, será proposto a transformação *AspectJML* em um serviço, afirmando a linguagem, tornando possível sua maior disseminação e facilitando o seu uso, em especial para o ensino.

Desta forma, será proposto uma interface *Web* onde se tornará possível a compilação e execução remota de códigos escritos em *AspectJML*.

CRONOGRAMA DE ATIVIDADES

Atividades	Abril	Maio	Junho	Julho
Pesquisa de bibliografia básica	X	X		
Escrita da monografia	X	X	X	
Desenvolvimento da parte prática	X	X	X	
Revisão da monografia		X	X	X
Preparação para a defesa			X	X

POSSÍVEIS AVALIADORES

NOME	ÁREA DE PESQUISA (INFERIDA)
LEOPOLDO TEXEIRA	Compiladores
RICARDO MASSA	Engenharia de Software, com pesquisas em *JML*
SERGIO SOARES	Engenharia de Software, com pesquisas em *AOP*

ASSINATURAS

_____

Henrique Rebêlo
Orientador

_____

Igor Vinícius Pinheiro Cordeiro Leão

Orientando

REFERÊNCIAS

[1] B. Meyer. Applying “design by contract”. Computer, 25(10):40–51, 1992.

[2] H. Rebelo, G. T. Leavens, M. Bagherzadeh, H. Rajan, R. Lima, D. M. ˆ Zimmerman, M. Cornelio, and T. Th ´ um. AspectJML: Modular specifi- ¨ cation and runtime checking for crosscutting contracts. In Proceedings of the 13th International Conference on Modularity, MODULARITY ’14, pages 157–168, New York, NY, USA, 2014. ACM.

[3] Y. A.Feldman etal. Jose:Aspects forDesignbyContract80-89. IEEE SEFM, 0:80–89, 2006.

[4] G.Kiczales, E.Hilsdale, J.Hugunin, M.Kersten,J.Palm,andW.Griswold. Getting Started with AspectJ. Commun. ACM, 44:59–65, October 2001

[5] Webb, Jason. "A declarative data contract container type for Python." pycontract 0.1.4 (2010). Python Package Index. Web. 5 Dec 2012.

[6] G. T. Leavens, A. L. Baker, and C. Ruby. Preliminary design of JML: Abehavioral interface specification language forJava. ACMSIGSOFT Software Engineering Notes, 2006.

[7] B. Meyer. Eiffel: The Language. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1992.