PROGRAMA INSTITUCIONAL DE BOLSAS DE INICIAÇÃO CIENTÍFICA – PIBIC/UFS

PROJETO DE PESQUISA

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

ORIENTADOR LÍDER DO PROJETO:		Alberto Costa Neto	
CENTRO:		Centro de Ciências Exatas e Tecnologia (CCET)	
DEPARTAMENTO:		Departamento de Computação (DCOMP)	
LOCAL DE EXECUÇÃO:		DCOMP	
FONTES DE FINANCIAMENTO:		-	
DATA DE INÍCIO:	**01/08/13**	**DATA DA CONCLUSÃO:**	**31/07/14**

	Nome / Código (CNPq)
GRANDE ÁREA DO CONHECIMENTO (CNPq):	Ciências Exatas e da Terra / 1.00.00.00-3
ÁREA DO CONHECIMENTO (CNPq):	Ciência da Computação
SUB-ÁREA DO CONHECIMENTO (CNPq):	Metodologia e Técnicas da Computação
ESPECIALIDADE DO CONHECIMENTO (CNPq):	Metodologia e Técnicas da Computação
NOME DO GRUPO DE PESQUISA:	Engenharia de Software
ÁREA DO QUALISE (CAPES) http://qualis.capes.gov.br/):	Ciência da Computação

EQUIPE EXECUTORA:	**DEPTO.**
Alberto Costa Neto	DCOMP
Lidiany Cerqueira Santos	Discente Metrado
Aluno 1	DCOMP
Aluno 2	DCOMP
Aluno 3	DCOMP

INTRODUÇÃO

Com a proposta de modularizar e encapsular corretamente os requisito transversais (*crosscutting concerns*), (Kiczales et al. 1997) apresenta a POA, 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 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 (http://www.eclipse.org/aspectj/), 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) (http://info.ufrn.br/wikisistemas/), como vem ocorrendo em Instituições Federais de Ensino Superior (IFES) (Passos et al. 2013).

De acordo com (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 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 execução 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 no 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) (http://abc.comlab.ox.ac.uk/) 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). 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 necessidade de adotar a LSD com 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 arterfatos 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.

Os alunos de graduação vinculados ao projeto terão oportunidade de vivenciar um problema real, complexo e relativamente comum que os profissionais da área enfrentam. Além disso, terão oportunidade de estudar e vivenciar técnicas que não fazem parte da grade curricular do seu curso. Além disso, estes alunos interagirão com a aluna de mestrado vinculada ao projeto, já que este projeto tem ligação com as suas dissertações de mestrado, o que estimula os alunos a ingressarem futuramente em cursos de mestrado.

Além disso, para a aluna de mestrado contar com alunos de iniciação científica é de extrema importância para atingir os objetivos de seu trabalho, já que parte do trabalho seria realizada pelos alunos de iniciação científica. Sem contar a experiência de formar um grupo de trabalho para tocar um projeto, o que faz parte da formação de qualquer bom pesquisador.

Finalmente, como o projeto ataca um problema real enfrentado pela equipe do CPD da UFS, seus resultados podem ser aplicados visando aprimorar o processo de adaptação e manutenção do SIG na UFS, trazendo benefícios para toda a comunidade acadêmica na forma de atualizações mais rápidas e agilidade na implantação de novos módulos. É importante destacar que este problema não é exclusivo do SIG ou da UFS, pois ele ocorre em qualquer sistema reutilizado por empresas ou instituições como base para criar sistemas customizados para clientes específicos, o que quase que na totalidade das vezes requer adaptações para se adequar aos requisitos específicos dos clientes.

OBJETIVOS

O principal objetivo deste projeto é avaliar a adequação da linguagem LSD como mecanismo para especificar as regras de projeto que devem ser respeitadas a fim de que os aspectos implementados em um sistema possam evoluir juntamente com as classes de forma que as dependências entre eles estejam claramente estabelecidas, reduzindo ou até evitando problemas de manutenção e evolução de sistemas ocasionados pela adoção da POA.

São objetivos específicos deste projeto:

- 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.

METODOLOGIA

Este projeto será coordenado pelo Prof. Alberto Costa Neto, cuja área de conhecimento envolve o desenvolvimento de aplicações corporativas como o SIG, além de ter desenvolvido sua tese de doutorado na área de POA, tendo também experiência na área de Linhas de Produtos de Software. Além disso, a aluna de mestrado que irá colaborar com o projeto é servidora do IFS e está lotada no CPD, tendo conhecimento sobre SIG. Sendo assim, a capacitação do pesquisador e da colaboradora é adequada para o desenvolvimento deste projeto. Além disso, o pesquisador leciona disciplinas que envolvem englobam a Programação Orientada a Aspectos e Linhas de Produtos de Software na graduação e no mestrado.

Quanto às atividades do projeto, descritas logo abaixo, procurou-se seguir a ordem cronológica aproximada, sendo que algumas ocorrerão em paralelo. Os períodos previstos de realização das mesmas encontram-se detalhados no cronograma:

- **A1: Estudar o SIG e as variações existentes no mesmo.** Esta atividade contará com a colaboração intensa da aluna de mestrado, já que a mesma conhece a fundo alguns módulos do SIG;
- **A2: Estudar o paradigma de POA.** O coordenador do projeto poderá ajudar bastante neste ponto já que ministra disciplinas na área, fornecendo material para estudo e até sanando determinadas dúvidas;
- **A3: Estudar a linguagem de definição de regras de projeto (LSD).** Nesta atividade o aluno deverá estudar a linguagem LSD, o qual será auxiliado pelo coordenador do projeto;
- **A4: Implementar as variações no SIG utilizando a POA e especificando as regras de projeto com LSD.** Esta atividade envolve a criação das regras de projeto que podem ajudar no processo de manutenção e evolução do SIG;
- **A5: Aprimorar o compilador de LSD.** À medida que as regras de projeto forem especificadas, pode ser necessário corrigir alguma falha no compilador de LSD, já que o mesmo também não foi submetido a uma avaliação.
- **A6: Realizar experimento controlado visando avaliar a utilização de LSD no contexto do projeto.** Esta atividade envolve a realização de um experimento utilizando um módulo significativo do SIG para confirmar se a Linguagem LSD e seu compilador ajudam no processo de evolução e manutenção do SIG;
- **A7: Analisar os resultados obtidos com a utilização de LSD.** Nesta atividade deve ser escrito um artigo descrevendo os resultados obtidos;

O acompanhamento do projeto dar-se-á através de reuniões semanais entre o coordenador do projeto, a aluna de mestrado (colaboradora) e os alunos envolvidos. Os alunos deverão apresentar alguns seminários sobre o andamento do projeto. Espera-se também que a equipe elabore pelo menos dois artigos científicos sobre os resultados obtidos no trabalho.

É importante que a depender da complexidade do SIG e suas variações, além, obviamente, do desempenho individual do aluno este projeto pode ser estendido por 1 ano, seguindo um processo iterativo e incremental em relação às variações tratadas e as técnicas avaliadas, casando assim com o tempo do mestrado da aluna colaboradora.

IMPACTOS ESPERADOS

Dentre os impactos esperados do projeto, podemos destacar:

- Difundir a programação orientada a aspectos entre os alunos de graduação;
- Estimular a interação entre alunos de graduação e mestrado;
- Publicação de artigos em científicos relatando os resultados;
- Aprimoramento do processo de adaptação e manutenção do SIG da UFS;
- Compartilhar os resultados com outras IFES adquirentes do SIG;
- Beneficiar toda a comunidade acadêmica através da viabilidade de atualizações mais rápidas e agilidade na implantação de novos módulos do SIG na UFS.

CRONOGRAMA

Atividade	**Mês**											
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
A1: Estudar o SIG e as variações existentes no mesmo.	X	X	X									
A2: Estudar o paradigma de POA.		X	X	X								
A3: Estudar a linguagem de definição de regras de projeto (LSD).			X	X	X	X						
A4: Implementar as variações no SIG utilizando a POA e especificando as regras de projeto com LSD.					X	X	X	X	X	X		
A5: Aprimorar o compilador de LSD.					X	X	X	X	X	X		
A6: Realizar experimento controlado visando avaliar a utilização de LSD no contexto do projeto.							X	X	X	X		
A7: Analisar os resultados obtidos com a utilização de LSD.											X	X

REFERÊNCIAS BIBLIOGRÁFICAS

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 evolução de pointcuts. In SBCARS09: Proceedings of 2nd Brazilian Symposium on Components, Architecture and Software Reuse.

Gasiunas, V., Satabin, L., Mezini, M., N´u˜nez, A., and Noy´e, 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., Bonif´acio, 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). Adaptação e manutenção de sistemas integrados de gestão apoiados pela programação orientada a aspectos. In Proceedings of The IX Simpósio Brasileiro de Sistemas de Informaçãoo - SBSI 2013.

Rashidi, P. and Alexander, R. T. (2008). Onspect: Ontology based aspects. In Proceedings of the 7th Workshop on Foundations of Aspect-oriented Languages, 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 Workshop on Foundations of Aspect-oriented Languages, 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 K¨astner, 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 H¨ansel, 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.

PLANOS DE TRABALHO DOS BOLSISTAS

ALUNO 1: Implementação de Regras de Projeto no SIG da UFS com a linguagem LSD.

As atividades do aluno serão um subconjunto das constantes no cronograma, as quais são listadas abaixo:

- A1: Estudar o SIG e as variações existentes no mesmo.
- A2: Estudar o paradigma de POA.
- A3: Estudar a linguagem de definição de regras de projeto (LSD).
- A4: Implementar as variações no SIG utilizando a POA e especificando as regras de projeto com LSD.
- A7: Analisar os resultados obtidos com a utilização de LSD.

ALUNO 2: Aprimoramento do Compilador da linguagem de definição de regras de projeto LSD.

As atividades do aluno serão um subconjunto das constantes no cronograma, as quais são listadas abaixo:

- A1: Estudar o SIG e as variações existentes no mesmo.
- A2: Estudar o paradigma de POA.
- A3: Estudar a linguagem de definição de regras de projeto (LSD).
- A5: Aprimorar o compilador de LSD.
- A7: Analisar os resultados obtidos com a utilização de LSD.

ALUNO 3: Realização de experimento controlado para avaliar a linguagem LSD no contexto do SIG.

As atividades do aluno serão um subconjunto das constantes no cronograma, as quais são listadas abaixo:

- A1: Estudar o SIG e as variações existentes no mesmo.
- A2: Estudar o paradigma de POA.
- A3: Estudar a linguagem de definição de regras de projeto (LSD).
- A6: Realizar experimento controlado visando avaliar a utilização de LSD no contexto do projeto.
- A7: Analisar os resultados obtidos com a utilização de LSD.