Um Estudo sobre o Emprego de Quantificação em Sistemas Orientados por Aspectos: Resultados Preliminares

Marcelo Nassau Malta, Tarik Melo,
Leonardo Humberto Guimarães Silva e Marco Túlio de Oliveira Valente

Instituto de Informática
Pontifícia Universidade Católica de Minas Gerais
{nassau,leonardosilva,mtov}@pucminas.br, tarik.melo@gmail.com

Resumo. *Neste position paper, descrevem-se os resultados preliminares de um estudo que está sendo desenvolvido com o objetivo de avaliar o emprego e os benefícios de quantificação em sistemas AspectJ. Propõe-se uma métrica para medir o grau de quantificação de um sistema e descreve-se a sua aplicação em três sistemas reais. Conclui-se que quantificação contribui com critérios como facilidade de mudanças, facilidade de compreensão e desenvolvimento independente. Por outro lado, quantificação exige que o interesse transversal seja bastante homogêneo, o que a princípio não é tão comum em sistemas reais.*

1 Introdução

Desde um trabalho de Filman e Friedman, quantificação e *obliviousness* são consideradas como as características distintivas de orientação por aspectos [3]. Basicamente, linguagens orientadas por aspectos permitem a definição de abstrações de comandos que afetam (ou quantificam) de forma não-invasiva múltiplas partes de um sistema alvo. Mais recentemente, o conceito de *obliviousness* tem sido objeto de críticas e propostas de revisão, por comprometer princípios clássicos de Engenharia de Software, como desenvolvimento em paralelo e raciocínio modular [9]. Por outro lado, o conceito de quantificação tem sido menos analisado e desafiado até o presente momento.

Assim, neste *position paper*, descrevem-se os primeiros resultados de um estudo que está sendo desenvolvido com o objetivo de avaliar o uso e benefícios, tais como a melhoria na facilidade de manutenção e entendimento do código, que podem ser proporcionados por quantificação em sistemas implementados em AspectJ. Inicialmente, na Seção 2, introduz-se uma nova métrica, chamada grau de quantificação, usada para dimensionar o emprego de quantificação em sistemas orientados por aspectos. Na Seção 3, são descritos e discutidos os resultados da aplicação dessa métrica em três sistemas orientados por aspectos de médio porte. A Seção 4 discute trabalhos relacionados. Por fim, a Seção 5 conclui o artigo e apresenta brevemente os próximos passos do trabalho.

2 Grau de Quantificação

A fim de avaliar o emprego de quantificação em sistemas orientados por aspectos, propõe-se neste artigo uma nova métrica, chamada grau de quantificação. Suponha que o conjunto $\{A_1, A_2, \ldots A_n\}$ contenha todos os adendos (*advices*) definidos em todos os aspectos de um determinado sistema. Seja $\mathcal{PC}$ uma função que retorna o conjunto de junção (*pointcut*) associado a um determinado adendo A_i e $\mathcal{JPS}$ uma função que retorna o número de

sombras de pontos de junção (*join point shadows*) associadas a um determinado conjunto de junção[1]. O grau de quantificação $\mathcal{QD}$ de um sistema é definido da seguinte forma:

$$\mathcal{QD} = \frac{\sum_{i=1}^{n} \mathcal{JPS}\left(\mathcal{PC}\left(A_i\right)\right)}{n}$$

O grau de quantificação de um sistema representa, em média, o número de pontos do código de um programa afetados pelos comandos quantificados (ou adendos) de um sistema orientado por aspectos. Assim, quanto maior o grau de quantificação de um sistema, menor o trabalho requerido para implementação de seus interesses transversais, pois o código antes implementado de forma espalhada e entrelaçada encontra-se agora modularizado em um ou poucos adendos. Ou seja, quanto maior o grau de quantificação, menor o esforço de codificação do programador para implementação de interesses transversais e maior o trabalho do compilador de aspectos para espalhamento desses interesses no código base do sistema. Por outro lado, caso o sistema possua grau de quantificação mínimo (isto é, QD=1), então cada um dos seus adendos afeta um único ponto distinto do código base. Nesse caso, considera-se que o interesse transversal é modularizado, porém não ocorre uma economia de esforço de codificação (já que essencialmente o que ocorre é uma migração "um-para-um" de código antes pertencente a métodos para adendos).

3 Estudos de Caso

Até o presente momento, a métrica de quantificação proposta foi calculada para os seguintes sistemas:

- JHotDraw[2]: um *framework* para criação de editores de figuras geométricas. Foi avaliado o grau de quantificação de aspectos implementados por Binkley et al. para modularização da funcionalidade de *undo* desse sistema [1].

- JAccounting[3]: um sistema Web para controle de pedidos e pagamentos. Foi avaliado o grau de quantificação de aspectos também implementados por Binkley et al. para controle de transações nesse sistema [1].

- Tomcat[4]: um servidor e *container* de aplicações Web. Foi avaliado o grau de quantificação de aspectos responsáveis pela funcionalidade de *logging* do Tomcat. Essa funcionalidade é usualmente citada como um exemplo de interesse transversal em textos de introdução a orientação por aspectos (usualmente, por meio de um diagrama que mostra verticalmente os pacotes do sistema e horizontalmente o entrelaçamento e espalhamento do código de *logging* [5]). A modularização em

[1] Um ponto de junção (*join point*) corresponde a um ponto bem definido da *execução* de um sistema. Já uma sombra de ponto de junção (*join point shadow*) corresponde *estaticamente* ao trecho de código responsável pela ativação de um ponto de junção [4]. Em outras palavras, pontos de junção podem ser vistos como eventos que ocorrem durante a execução de um sistema orientado por objetos; já sombras de pontos de junção são os pontos do código onde tais eventos são disparados. Ou seja, uma sombra de ponto de junção pode gerar diversos pontos de junção.

[2] `http://www.jhotdraw.org`.

[3] `https://jaccounting.dev.java.net`.

[4] `http://tomcat.apache.org`.

aspectos do código de *logging* foi implementada pelos autores desse artigo, com o objetivo principal de contribuir com o estudo de caso aqui descrito.

A Tabela 1 apresenta o grau de quantificação dos sistemas avaliados. No sistema JHotdraw, o grau de quantificação é mínimo (QD=1), pois o número de adendos é igual ao número de sombras de pontos de junção capturadas na aplicação. No Tomcat, o grau de quantificação também é bastante reduzido (igual a 1.09). Já no sistema JAccounting, o grau de quantificação é significativo (igual a 11.25), isto é, cada adendo declarado no sistema afeta em média 11.25 pontos do código base.

	JHotDraw	**Tomcat**	**JAccounting**
Sombras pontos junção	86	258	45
Adendos	86	236	4
Grau de quantificação	1	1.09	11.25
LOC (versão OO)	40022	45107	11676
LOC (versão AO)	40805 (+2%)	45642 (+1.2%)	11477 (-1.7%)

Tabela 1. Grau de quantificação nos sistemas JHotDraw, Tomcat e JAccounting

Análise: Os principais resultados deste estudo de caso são analisados a seguir:

- O grau de quantificação de um sistema depende essencialmente da homogeneidade do interesse transversal modularizado em aspectos. Interesses transversais homogêneos são aqueles que prescrevem o emprego do mesmo bloco de código em diferentes partes de um sistema [2]. Já interesses heterogêneos demandam o emprego de códigos diferentes em pontos diferentes do sistema. Assim, não surpreende que controle de transações – um interesse reconhecidamente homogêneo – tenha apresentado um grau de quantificação significativo no sistema JAccounting. Por exemplo, um único adendo nesse sistema é capaz de modularizar todo código de início de transação, conforme mostrado na Figura 1. Por outro lado, *undo* (interesse modularizado no sistema JHotDraw) e *logging* (interesse modularizado no sistema Tomcat) são bastante heterogêneos e, portanto, pouco se beneficiaram dos recursos de quantificação de AspectJ. No Tomcat, apesar da funcionalidade de *logging* ser implementada chamando sempre o mesmo método, existe uma grande variabilidade nos argumentos do tipo *string* desse método que representam as mensagens de *logging* (conforme ilustrado pelos adendos da Figura 2). Já no sistema JHotDraw, a funcionalidade de *undo* requer um código específico para cada tipo de figura que se pretende restaurar.

- Para que o grau de quantificação seja devidamente avaliado, é necessário considerar a qualidade da implementação dos aspectos. Aspectos mal projetados, onde conjuntos de junção implementados separadamente poderiam estar agrupados, certamente vão influenciar negativamente no grau de quantificação.

- Observou-se um aumento no número de linhas de código dos sistemas orientados por aspectos com baixo grau de quantificação – caso dos sistemas JHotdraw (2% a mais de linhas de código) e Tomcat (1.2% a mais de linhas de código). Sendo

```
pointcut p0(): call(Session SessionFactory.openSession()));
after() returning (Session s) throws HibernateException: p0() {
   tx= s.beginTransaction();
}
```

Figura 1. Exemplo de adendo do sistema JAccounting

```
pointcut p_48(Digester _this):
   execution(void startElement3(boolean))&&this(_this);
void around(Digester _this): p_48(_this) {
   _this.log.debug("No rules found matching " + _this.match);
}
pointcut p_49(Digester _this, String URL):
   execution(void resolveEntity(String))&&args(URL)&&this(_this);
void around(Digester _this, String URL):p_49(_this, URL) {
   _this.log.debug("Cannot resolve entity: " + entityURL);
}
pointcut p_50(Digester _this, String sysId):
   execution(void resolveEntity2(String))&&args(sysId)&&this(_this);
void around(Digester _this, String sysId): p_50(_this, sysId) {
   _this.log.debug("Trying to resolve using system ID " + sysId);
}
```

Figura 2. Exemplos de adendos do sistema Tomcat

o grau de quantificação desses sistemas mínimo, seus aspectos incluem adendos com código que afeta um único ponto do sistema base (conforme mostrado na Figura 2). Como a implementação de adendos em AspectJ possui uma sintaxe própria, requerendo código para declaração de seus parâmetros, para declaração de conjuntos de junção etc, justifica-se o aumento de linhas constatado.

- O baixo grau de quantificação observado nos sistemas JHotDraw e Tomcat não se deve a uma limitação das abstrações para modularização de interesses transversais disponíveis em AspectJ, mas sim à variabilidade observada na implementação dos interesses de *undo* e *logging*, respectivamente. Mais especificamente, se um interesse transversal possui variações relevantes em cada ponto onde o mesmo é implementado em um determinado sistema alvo – por exemplo, se faz obrigatoriamente uso de *strings* diferentes em cada um desses pontos, como ocorre no *logging* do Tomcat (Figura 2)– então é esperado que não se consiga prover uma implementação genérica para o mesmo.

Quantificação e Critérios para Avaliação de Modularização: Apesar de preliminares, os estudos de casos permitem concluir que sistemas com alto grau de quantificação atendem aos seguintes critérios propostos por Parnas para se avaliar estratégias de modularização [8]: facilidade de compreensão (*comprehensibility*), facilidade de mudanças (*changeability*) e desenvolvimento independente. De forma oposta, sistemas com baixo grau de quantificação tendem a não contemplar ou a contemplar de forma parcial esses mesmos critérios. Uma análise sobre o atendimento a esses critérios nos sistemas analisados é realizada a seguir.

- Facilidade de compreensão: Aspectos com alto grau de quantificação podem facilitar o entendimento de um sistema. Por exemplo, é relativamente simples inferir

que o adendo da Figura 1, extraído do sistema JAccounting, determina que após chamadas do método `openSession`, deve-se iniciar uma transação (chamando-se o método `beginTransaction`). Já os adendos da Figura 2, extraídos do Tomcat, são mais complexos, demandando inclusive conhecimento detalhado do código base para seu entendimento.

- Facilidade de mudanças: Ao migrar código espalhado, por exemplo, por n pontos de um sistema para um único adendo, pode-se facilitar a realização de mudanças que afetem esse código (um exemplo dessa situação é o código de abertura de transação, modularizado no adendo da Figura 1). Por outro lado, se o sistema apresentar grau de quantificação mínimo, uma mudança como a descrita deve ser realizada em n pontos do sistema (como seria o caso de uma mudança no *logging* do Tomcat, conforme ilustrado na Figura 2). A principal vantagem nesse caso é que esses n pontos estarão lexicamente localizados em um único aspecto.

- Desenvolvimento independente: de forma geral, essa propriedade tem sido considerada como uma deficiência importante de sistemas orientados por aspectos. No entanto, se um interesse apresenta alto grau de quantificação, essa deficiência é suavizada, já que torna-se mais simples a definição de regras de projeto (*design rules*) que devem ser seguidas pelo código base a fim de facilitar a implementação independente de aspectos [9]. Em outras palavras, quantificação não reduz a dependência e o acoplamento entre aspectos e código base, porém favorece a definição de regras de projeto genéricas que documentem essa dependência.

4 Trabalhos Relacionados

Um conjunto de métricas foi proposto por Herrejon e Apel para categorizar interesses transversais de acordo com o número de classes que são atravessadas pelos mesmos [7]. Dentre as métricas propostas, destaca-se o Quociente de Homogeneidade (HQ). Influenciada pelos conceitos de programação orientada por *features*, a métrica HQ considera como homogêneo um interesse cuja modularização usa mais transversalidade dinâmica do que transversalidade estática. Ou seja, a métrica proposta não considera a abrangência de adendos, conforme ocorre com o conceito de grau de quantificação. É interessante observar, no entanto, que nos estudos de casos nos quais a métrica HQ foi aplicada também constatou-se que adendos afetam um pequeno percentual das classes de um sistema.

Kästner, Apel e Batory realizaram um estudo com o objetivo de modularizar em aspectos um conjunto de *features* do banco de dados Oracle Berkeley DB, de forma a gerar uma linha de produtos de *software* [6]. O estudo mostrou que a maioria dos aspectos extraídos correspondem a interesses bastante heterogêneos – mais precisamente, informa-se que dos 482 adendos implementados, apenas sete incluem mais de uma sombra de ponto de junção. Em conseqüência, os autores concluem que aspectos, como suportados por AspectJ, não são a melhor tecnologia para atender a seus objetivos iniciais. Ou seja, essa experiência evidencia que o grau de quantificação é um fator importante para apoiar uma decisão de migração para tecnologia de aspectos.

5 Conclusões

Quantificação é realmente uma característica distintiva de orientação por aspectos, conforme sugerido por Filman e Friedman. Para comprovar esse fato, foi proposto neste artigo uma nova métrica para expressar o grau de quantificação de um sistema orientado por aspectos. Os estudos de caso realizados evidenciaram que sistemas com maior grau de quantificação são mais fáceis de entender, de manter e de serem desenvolvidos em paralelo. No entanto, duas questões permanecem em aberto. Primeiro, quantificação exige elevados níveis de homogeneidade na implementação de interesses transversais, os quais parecem não ocorrer com freqüência em interesses de sistemas reais. Por outro lado, não é tão claro também se vale a pena usar aspectos no caso de sistemas com baixo grau de quantificação, como é o caso, por exemplo, dos sistemas Tomcat e JHotDraw.

No curto prazo, pretende-se implementar uma ferramenta para calcular o grau de quantificação de um sistema. De posse dessa ferramenta, pretende-se ampliar os estudos de caso com novos sistemas. A médio prazo, pretende-se evoluir essa ferramenta para calcular o grau de quantificação mesmo antes da implementação de aspectos.

Agradecimentos: Este trabalho foi desenvolvido como parte de um projeto de pesquisa financiado pela FAPEMIG (processo CEX-817/05 - Edital Universal 2005).

Referências

[1] David Binkley, Mariano Ceccato, Mark Harman, Filippo Ricca, and Paolo Tonella. Tool-supported refactoring of existing object-oriented code into aspects. *IEEE Transactions Software Engineering*, 32(9):698–717, 2006.

[2] Adrian Colyer and Andrew Clement. Large-scale AOSD for middleware. In *3rd International Conference on Aspect-Oriented Software Development*, pages 56–65. ACM Press, 2004.

[3] Robert E. Filman and Daniel P. Friedman. Aspect-oriented programming is quantification and obliviousness. In *OOSPLA Workshop on Advanced Separation of Concerns*, October 2000.

[4] Erik Hilsdale and Jim Hugunin. Advice weaving in AspectJ. In *3rd International Conference on Aspect-Oriented Software Development (AOSD)*, pages 26–35, 2004.

[5] Erik Hilsdale and Mik Kersten. Aspect-oriented programming with AspectJ, 2001.

[6] Christian Kästner, Sven Apel, and Don Batory. A case study implementing features using aspectj. In *International Software Product Line Conference (SPLC'07)*, September 2007.

[7] Roberto Lopez-Herrejon and Sven Apel. Measuring and characterizing crosscutting in aspect-based programs: Basic metrics and case studies. In *ETAPS International Conference on Fundamental Approaches to Software Engineering (FASE'07)*, March 2007.

[8] David Lorge Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972.

[9] Kevin J. Sullivan, William G. Griswold, Yuanyuan Song, Yuanfang Cai, Macneil Shonle, Nishit Tewari, and Hridesh Rajan. Information hiding interfaces for aspect-oriented design. In *13th International Symposium on Foundations of Software Engineering (FSE)*, pages 166–175, 2005.