UMA FERRAMENTA PARA MINERAÇÃO DE ASPECTOS

DELFIM, Fernanda Madeiral
Universidade Estadual Paulista “Júlio de Mesquita Filho” (UNESP)
fer.madeiral@gmail.com

GARCIA, Rogério Eduardo
Universidade Estadual Paulista “Júlio de Mesquita Filho” (UNESP)
rogerio@fct.unesp.br

RESUMO: A Mineração de Aspectos visa a identificar potenciais interesses transversais em código fonte de programa e a Refatoração para Aspectos visa a encapsulá-los em aspectos. A Mineração de Aspectos é um processo não-automático, pois o usuário precisa analisar e compreender os resultados gerados por técnicas/ferramentas e confirmar interesses transversais para refatorá-los em aspectos. Neste trabalho é proposta uma abordagem visual que lida com resultados gerados por duas técnicas de mineração de aspectos propostas na literatura. Por meio de múltiplas visões coordenadas, diferentes níveis de detalhe para explorar sistemas de software apóiam a análise e a compreensão de tais resultados para futura refatoração. O modelo de coordenação, implementado na ferramenta SoftVis4CA, é apresentado neste trabalho, juntamente com as visualizações e com os resultados preliminares obtidos.
PALAVRAS-CHAVE: Análise de Código, Mineração de Aspectos, Refatoração para Aspectos, Compreensão de Programa, Visualização de Software.

ABSTRACT: *Aspect Mining aims to identify potential crosscutting concerns in source code of program and Refactoring to Aspects aims to encapsulate them in aspects. Aspect Mining remains as non-automatic process, because the user needs to analyze and understand the results generated by techniques/tools and confirm crosscutting concerns to refactor them in aspects. In this paper is propose a visual approach that deals with results generated by two aspect mining techniques proposed in the literature. By multiple coordinated views, different levels of detail to explore software systems support analysis and comprehension of such results for further refactoring. The coordination model, implemented in SoftVis4CA tool, is presented in this paper, together the visualizations and the preliminary results obtained.*
KEYWORDS: *Code Analysis, Aspect Mining, Refactoring to Aspects, Program Comprehension, Software Visualization*

INTRODUÇÃO

A modularização de interesses transversais (do inglês Crosscutting Concerns - CCs) ainda é um desafio, pois a implementação desses interesses tende a ser espalhada por diversas unidades modulares do sistema e emaranhada com a implementação de outros interesses (KICZALES et al., 1997; SANT’ANNA et al., 2003; MASSICOTTE *et al.,* 2007), o que é um problema de compreensibilidade de sistema de software, e como resultado torna difícil a manutenção (CECCATO et al., 2006). A Programação Orientada a Aspectos (KICZALES *et al.*, 2001) provê mecanismos e abstrações para modularizar CCs, encapsulando-os em uma unidade de código separada, chamada aspecto, aumentando a coesão dos módulos (CLIFTON *et al.,* 2007) e possibilitando reúso de código (KICZALES *et al*., 1997).

A evolução de sistemas de software existentes para a tecnologia orientada a aspectos é também um desafio. Primeiro, é necessário identificar CCs em programa não orientado a aspecto que potencialmente podem ser encapsulados em aspectos e, se aceitos por um usuário, refatorar para aspectos. A Mineração de Aspectos é uma área de pesquisa cujo objetivo é identificar CCs. Diversas técnicas têm sido propostas para a mineração de aspectos (BREU; KRINKE, 2004; TONELLA; CECCATO, 2004; TOURWÉ; MENS, 2004; KRINKE; BREU, 2005; BRUNTINK *et al.,* 2005; MARIN et al., 2007; ABAIT *et al.*, 2008; HUANG et al., 2010; KATTI *et al.,* 2012). Em geral,

tais técnicas possuem limitações similares, como: precisão ruim dos resultados (isto é, a porcentagem de candidatos a aspectos relevantes relatados por uma dada técnica é relativamente baixa); subjetividade para analisar os resultados por causa de ambiguidade nos resultados produzidos; dificuldade em comparar os resultados de diferentes técnicas; e dificuldade em combinar diferentes técnicas. A apresentação inadequada de resultados é apontada como possível causa de tais limitações (MENS et al., 2008). Nesse contexto, abordagens para lidar com as limitações das técnicas propostas para minerar aspectos podem ser úteis.

O uso de Visualização de Software é uma alternativa que pode ajudar na análise e interpretação dos resultados de técnicas de mineração de aspecto, tornando explícito como o código fonte é organizado, como diferentes níveis de abstração são associados e como essas associações ocorrem no código fonte. Diversas abordagens e ferramentas têm sido propostas para apoiar compreensão de programa usando técnicas de visualização de software, como Asbro (PFEIFFER; GURD, 2006), CRISTA (PORTO *et al.,* 2009) e SoftVis$_{OAH}$ (D'ARCE *et al.*, 2012), mas elas não apóiam a identificação de CCs.

Neste trabalho é apresentada uma abordagem visual para apoiar a mineração de aspecto de programas orientados a objetos implementados em Java. A abordagem proposta inclui seis representações visuais coordenadas, sendo cada uma delas para apresentar uma perspectiva diferente de programa. Duas técnicas de mineração de aspectos foram escolhidas para serem implementadas: foi escolhido fatiamento de programa porque o seu uso foi proposto na literatura não somente para identificar interesses transversais, mas também para apoiar a refatoração para aspectos, e foram adicionadas características da análise de fan-in a fim de facilitar a análise e a compreensão. Nosso foco é investigar se a visualização contribui na compreensão de programas por meio dos resultados gerados usando as técnicas de fatiamento de programa e análise de *fan-in* de maneira complementar. Uma ferramenta de visualização de software, nomeada SoftVis$_{4CA}$ (*Software Visualization for Code Analysis*), foi desenvolvida para apoiar a abordagem visual proposta.

O restante deste artigo encontra-se organizado como segue. Na Seção 1 são apresentadas as técnicas de mineração de aspectos utilizadas neste trabalho. Na Seção 2 é apresentado o modelo de coordenação proposto juntamente com algumas considerações sobre a implementação da ferramenta desenvolvida e com as representações visuais obtidas com a ferramenta. Finalmente, as considerações finais são sumarizadas na última Seção.

1. TRABALHOS RELACIONADOS

Diversas técnicas têm sido propostas para minerar aspectos (BREU; KRINKE, 2004; TONELLA; CECCATO, 2004; TOURWÉ; MENS, 2004; KRINKE; BREU, 2005; BRUNTINK *et al.*, 2005; MARIN *et al.*, 2007; ABAIT *et al.*, 2008; HUANG *et al.,* 2010; KATTI *et al.*, 2012). Dentre elas, foram escolhidas as técnicas de fatiamento de programa e características da análise de fan-in para a visualização de resultados, como segue. KATTI et al. (2012) discutiram sobre o uso de fatiamento de programa para apoiar o processo de inserção de aspectos em um programa orientado a objetos. De acordo com eles, o processo de fatiamento consiste em identificar as instruções que formam uma fatia (identificação de fatia) e em isolar essas instruções em um programa independente (extração de fatia). Assim, eles vêem muita semelhança com o processo de Mineração de Aspectos e Refatoração para Aspectos.

Adicionalmente, foi utilizada a métrica fan-in em atributos visuais da abordagem visual proposta. A análise de fan-in tenta capturar código espalhado em nível de método, que é um sintoma de CCs porque métodos podem implementar CCs e, então, esses métodos são chamados por diversas unidades, resultando um alto valor fan-in. Em uma reimplementação orientada a aspectos, o método com alto valor fan-in constitui um advice, e as chamadas para ele corresponde ao contexto que precisa ser capturado por pointcut (MARIN et al., 2007).

2. O MODELO DE COORDENAÇÃO PROPOSTO

A abordagem visual proposta para apoiar a mineração de aspectos emprega um modelo de coordenação de seis visões, como ilustrado na Figura 1. A Visão Inter-Classes apresenta as chamadas de métodos entre classes usando projeção hiperbólica; a Visão *Inter-Métodos* apresenta as chamadas entre métodos usando, também, projeção hiperbólica; a Visão *Inter-Instruções* apresenta as dependências entre instruções de programa agrupadas pelos seus métodos, usando projeção de agregados; a Visão Estrutural apresenta a organização de programa em uma estrutura hierárquica usando a técnica *Treemap*; a Visão de Distribuição de Instruções apresenta a distribuição de instruções em classes usando a técnica Barras e Listras; e a Visão de Bytecode exibe as instruções bytecode de programa. Tais visões são coordenadas (as coordenações são indicadas pelas setas entre elas), permitindo uma exploração visual em diferentes níveis de detalhe.

Em cada representação visual, as cores dos itens visuais são pré-definidas, mas o usuário pode modificar tais cores se desejar. Adicionalmente, há um mapeamento de cor baseado em chamadas de métodos usado em visões – para cada unidade (i.e., pacotes, classes e métodos) é calculado o valor da métrica fan-in, e um gradiente de cor do cinza claro para vermelho é usado para representar os valores obtidos (vermelho representa o maior valor). Além disso, é possível filtrar unidades de código com base em valor de fan-in por meio da definição de um limiar, permitindo observar as

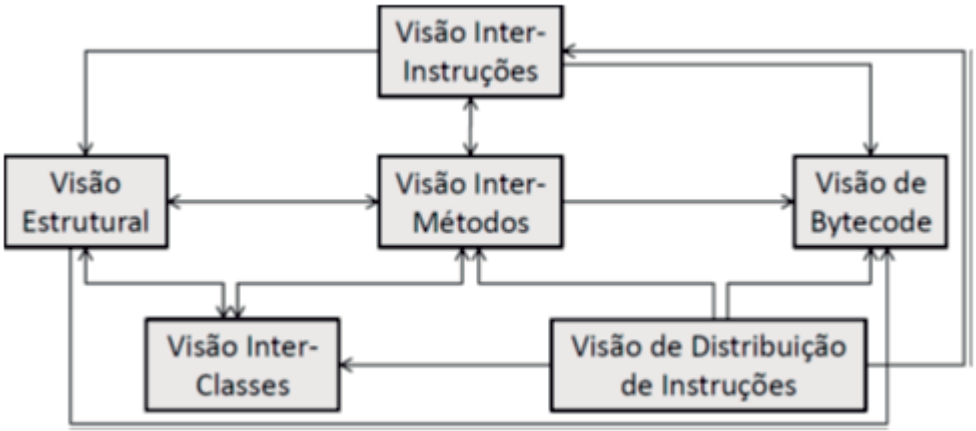

Figura 1: Modelo de Coordenação.

unidades correspondentes nas projeções e, assim, é possível visualizar as unidades com alta frequência de chamadas.

2.1. Questões de Implementação

A abordagem visual proposta foi implementada em uma ferramenta de Visualização de Software desktop, nomeada SoftVis4CA. Na Figura 2 é ilustrada a arquitetura da ferramenta, que é organizada em três camadas: Camada de Dados, Camada de Controle e Camada de Visualização. Usando como entrada um arquivo .jar contendo a implementação de um programa desenvolvido em Java e casos de teste criados usando JUnit, na Camada de Dados são realizadas as análises estática e dinâmica para coletar informações sobre o código e criar todos os conjuntos de dados necessários para gerar as representações visuais.

Na Camada de Visualização, os conjuntos de dados obtidos na Camada de Dados são mapeados para atributos visuais e, a partir desses atributos, são providas as representações visuais. Na Camada de Controle é realizada a coordenação das re-

presentações visuais, isto é, por meio do recebimento de eventos da interação do usuário em alguma representação visual, são obtidos dados relacionados ao item visual com o qual houve interação, pela consulta aos dados da Camada de Dados, e é disparado um evento para a Camada de Visualização informando o novo estado da representação visual, como itens destacados, novo foco e nova posição das informações na projeção. Assim, é permitida uma exploração visual de programa em diferentes níveis de detalhe.

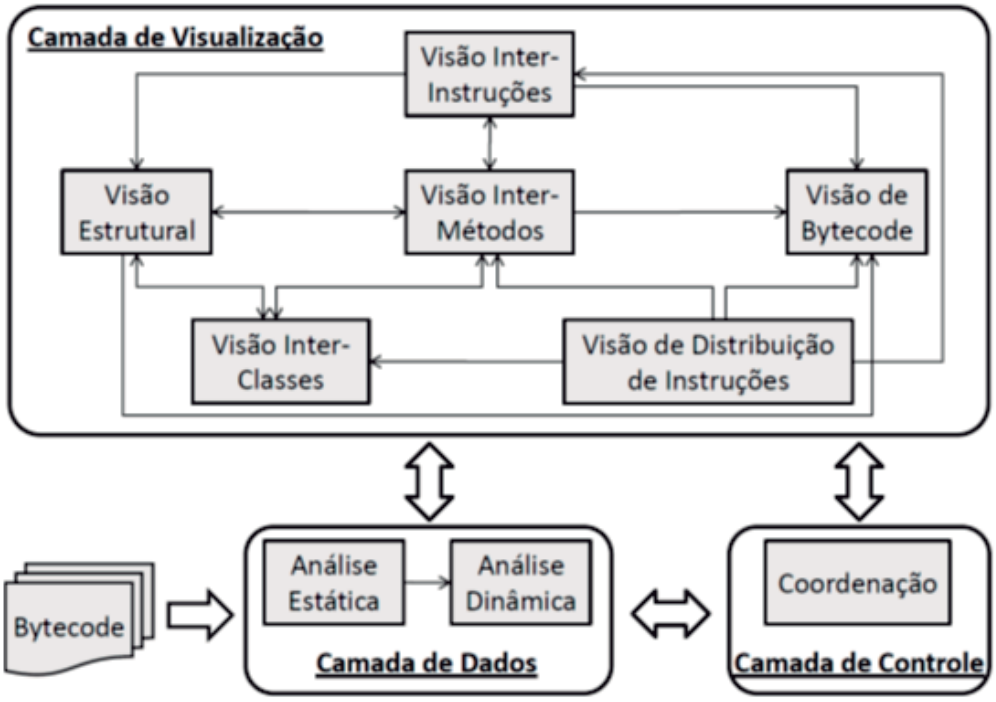

Figura 2: Arquitetura da *SoftVis4CA*.

2.2. Representações Visuais

Os resultados (representações visuais) apresentados neste artigo foram obtidos com a SoftVis4CA usando um programa orientado a objetos de simulação de elevador (DO et al., 2005). Nas Figuras 3 e 4 são mostradas duas projeções hiperbólicas baseadas em chamadas de métodos: a primeira em nível de classe e a segunda em nível de método. Os nós representam unidades (classes ou métodos), e as arestas direcionadas representam as chamadas entre elas. A projeção hiperbólica mostra um nó em foco (escolhido pelo usuário), maior do que os outros e posicionado no centro do espaço de projeção. Os nós vizinhos do nó em foco são coloridos em rosa, e os outros nós não são coloridos. A classe Logger e os métodos write e action estão coloridos usando o gradiente de cor baseado na métrica fan-in. Tais visões ajudam a ter evidências de possíveis interesses transversais por meio da visualização e da análise de resultados da métrica fan-in, e se o usuário decidir refatorar métodos para aspectos, ajudam a identificar quais unidades são afetadas – analisando as chamadas mostradas em nível de método é possível observar o método a ser considerado na criação de advices e os que devem ser entrecortados.

A Visão Inter-Instruções apresenta grafos em nível de instruções de programa. Cada método do programa tem seu próprio grafo, e grafos podem ser combinados por chamadas de métodos a partir de ou para um método selecionado (forward ou backward). Nas Figuras 5 e 6 são mostrados os dois tipos de Visão Inter-Instruções: vi-

são do grafo de um método (intra-método) e visão de um grafo combinado (inter-métodos), respectivamente. Existem dois tipos de nós nessas visões: nós físicos e nós virtuais. Os nós físicos representam instruções bytecode de programa (coloridos em rosa). Os nós virtuais são introduzidos para representar entradas de métodos (coloridos em verde escuro), saídas de métodos (coloridos em verde claro) ou parâmetros (coloridos em preto). As arestas representam fluxo de controle (linhas contínuas coloridas em preto), exceção (linhas tracejadas coloridas em preto), dependência de controle (linhas contínuas representam intra-método e linhas tracejadas representam inter-método, ambas coloridas em azul) ou dependência de dados (linhas contínuas representam intra-método e linhas tracejadas representam inter-método, ambas coloridas

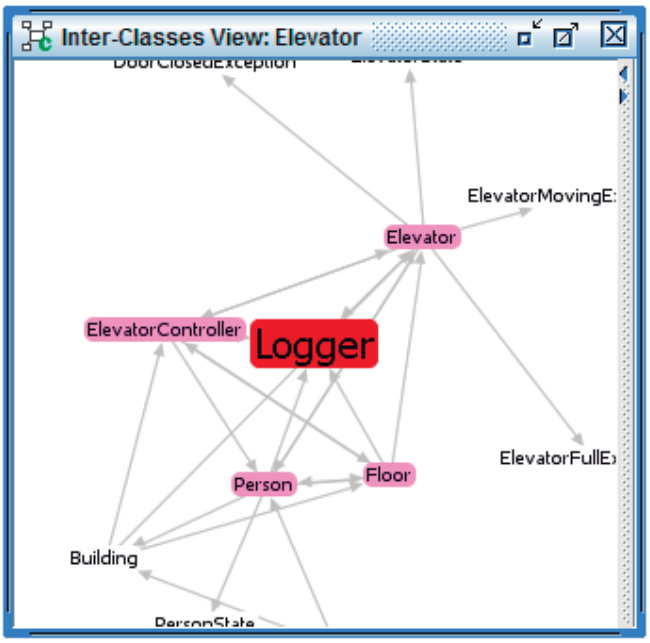

Figura 3: Visão *Inter-Classes.*

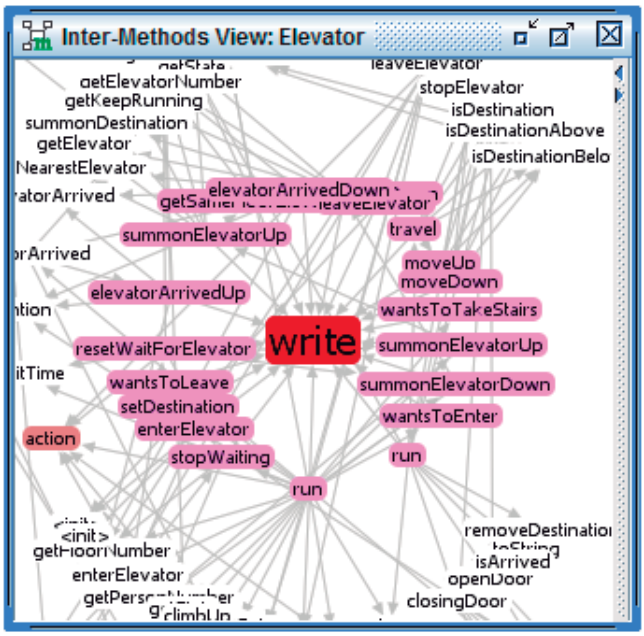

Figura 4: Visão *Inter-Métodos.*

em verde). Além disso, os nós são agregados por métodos, representados por polígonos curvos (nomeado região de agregado) coloridos por uma cor específica para cada método (foi utilizada transparência por causa de sobreposição de regiões e de nós).

Fatias podem ser visualizadas por meio de interação com a Visão Inter-Instruções. O usuário pode escolher um item visual que representa uma instrução em específico (critério de fatia) e o tipo de fatia (estática/dinâmica e backward/forward). Assim, uma outra Visão Inter-Instruções é criada para apresentar a fatia usando os parâmetros escolhidos.

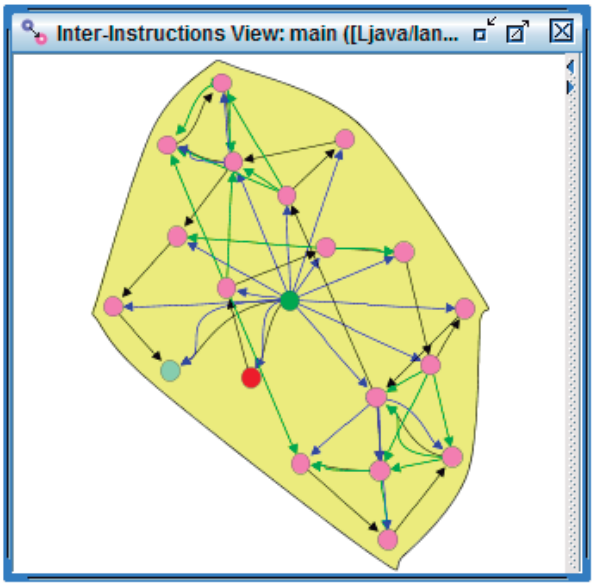

Figura 4: Visão *Inter-Instruções (Intra-Método).*

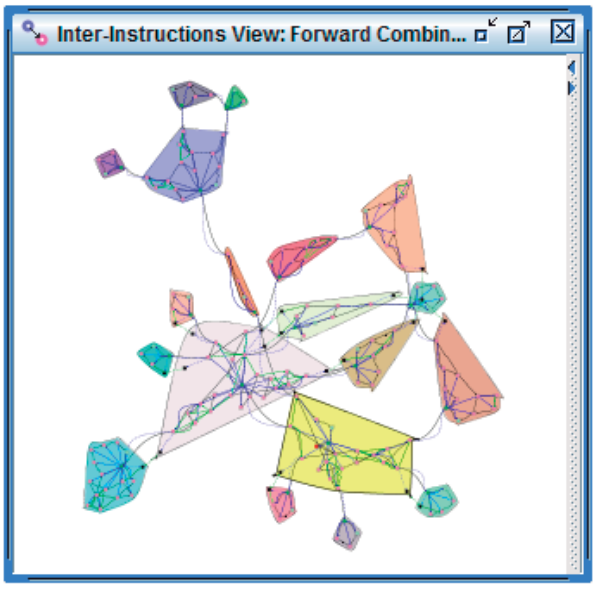

Figura 5: Visão *Inter-Instruções (Inter-Métodos).*

A Visão Estrutural apresenta a organização de programa em uma estrutura hierárquica. As unidades de programa são representadas por retângulos aninhados. O retângulo externo representa o programa todo (raiz), e os retângulos internos representam pacotes, classes, métodos e casos de testes. Os retângulos são coloridos usando um gradiente linear simples de cinza (cinza escuro representa o programa todo). Cada retângulo folha representa um método e o seu tamanho é proporcional a quantidade de instruções bytecode. Na Figura 7 está destacado o método selecionado na Figura 4 (sua classe em vermelho) e os métodos relacionados em rosa.

Na Figura 8 é mostrada a Visão de Distribuição de Instruções. As barras representam classes – a altura de cada barra é proporcional a quantidade de instruções bytecode da classe que ela representa. As listras representam instruções bytecode usando cores diferentes atribuídas para cada fatia. Tal visão provê uma visão geral de como uma fatia é distribuída em múltiplas classes.

Adicionalmente, a SoftVis4CA permite visualizar instruções bytecode usando duas rolagens sincronizadas – a rolagem da direita para visão geral e a rolagem da esquerda para detalhes – como mostrado na Figura 9. Na rolagem da direita os retângulos representam um conjunto de instruções agrupadas por labels de bytecode de programa. Na rolagem da esquerda são mostradas as instruções bytecode de retângulos selecionados na rolagem da direita, agrupadas pelo método que as possuem. Similar à Visão Inter-Instruções, cada método é mostrado usando uma cor diferente e o mapeamento de cores é o mesmo da Visão Inter-Instruções.

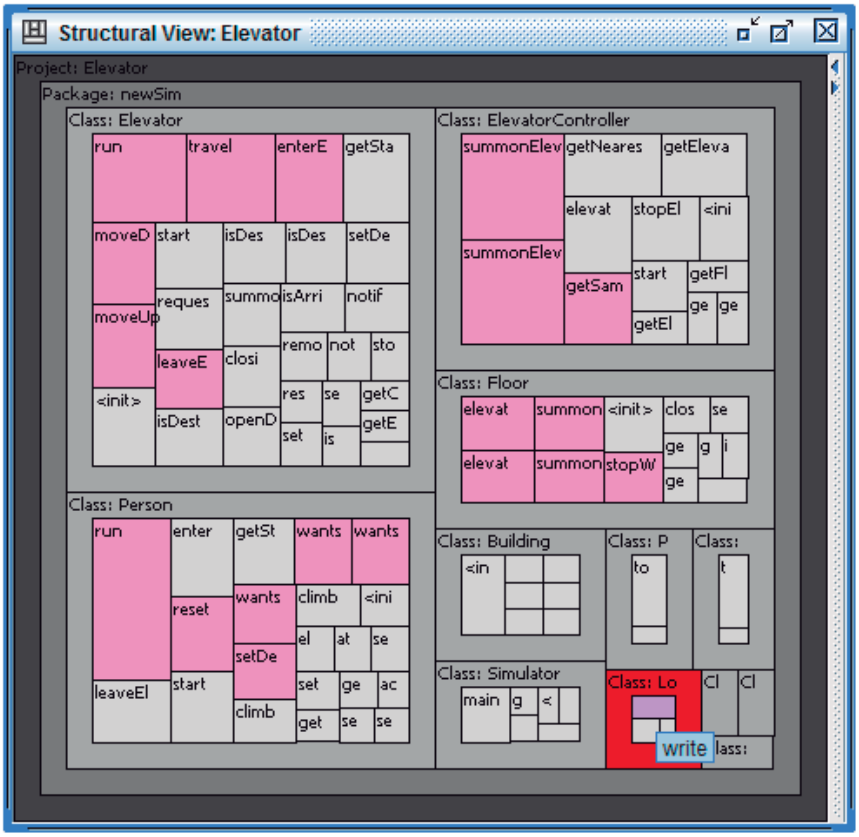

Figura 6: Visão Estrutural.

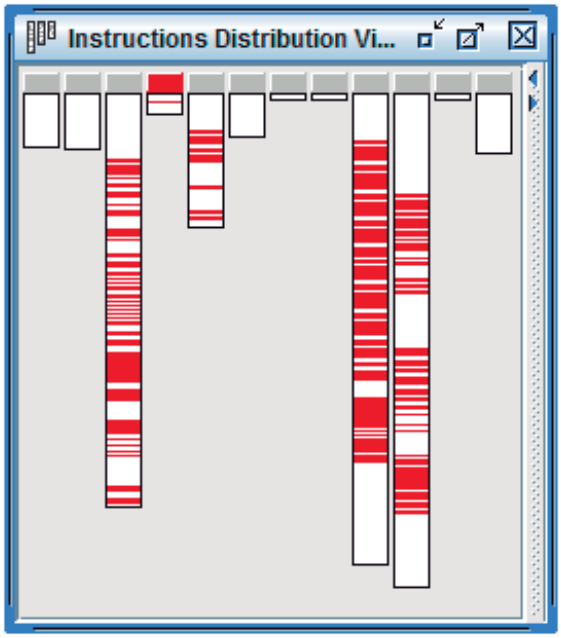

Figura 7: Visão de Distribuição de Instruções.

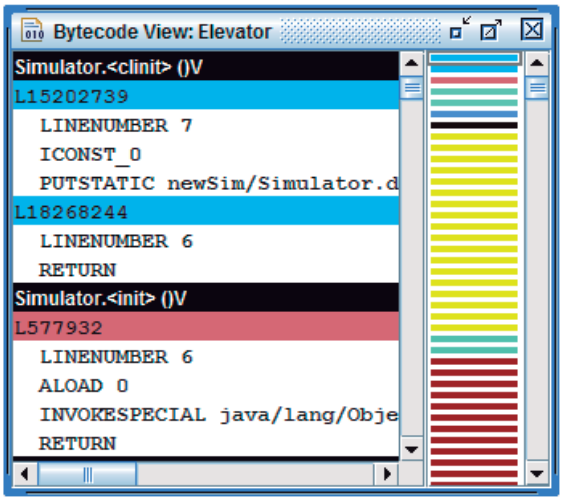

Figura 8: Visão de Bytecode.

CONSIDERAÇÕES FINAIS

Neste trabalho foi apresentada uma abordagem visual e sua implementação, que consiste na coordenação de múltiplas visões, o mapeamento de cores baseado na métrica fan-in e a visualização de fatias, permitindo: 1) uma exploração visual de software em diferentes níveis de detalhe; 2) a visualização dos métodos mais frequentemente chamados; 3) e a análise de dependência de controle e de dados.

Pela interação com as visões, o usuário pode obter informações sobre métodos e decidir sobre refatorá-los para aspecto. Métodos considerados candidatos a CCs, suas classes e métodos chamadores, e como eles estão relacionados devem ser analisados em diferentes níveis de detalhe. Em baixo nível, o usuário pode observar dependências de controle e de dados, o que é importante para avaliar como um CC pode ser implementado em um aspecto (por exemplo, o aspecto deve ter acesso aos dados do método?). Nós observamos que fatiamento de programa pode ser útil

não somente para identificar e extrair uma fatia, mas também para ajudar na definição de pointcuts (uma etapa da Refatoração para Aspectos).

Nosso estudo preliminar mostrou que o modelo de coordenação proposto apóia a análise de código pela exploração de diferentes níveis de detalhe. Além disso, foi concluído que a ferramenta ajuda a lidar com limitações de algumas projeções (escalabilidade). As visões que utilizam projeção hiperbólica e a Visão Inter-Instruções podem apresentar uma grande quantidade de nós/arestas, resultando em sobreposição de itens visuais e/ou dificultando a compreensão das informações apresentadas pelas visões. Isso pode ser contornado por meio da modificação de preferências e por meio da utilização de filtros.

Os resultados visuais e as observações apresentadas foram obtidos pelo estudo preliminar. No entanto, para avaliar a eficácia da abordagem, um experimento controlado está sendo planejado para avaliar a identificação de CCs e a definição de pointcuts em programas orientados a objetos usando a ferramenta.

REFERÊNCIAS BIBLIOGRÁFICAS

ABAIT, E. S.; VIDAL, S. A.; MARCOS, C. A. Dynamic Analysis and Association Rules for Aspects Identification. In: Proceedings of the II Latin American Workshop on Aspect-Oriented Software Development (LA-WASP '08). [S.l.: s.n.], 2008. p. 31–39.

BREU, S.; KRINKE, J. Aspect Mining Using Event Traces. In: 19th IEEE International Conference on Automated Software Engineering (ASE '04). Washington, DC, USA: IEEE Computer Society, 2004. p. 310–315.

BRUNTINK, M.; DEURSEN, A. van; ENGELEN, R. van; TOURWE, T. On the Use of Clone Detection for Identifying Crosscutting Concern Code. IEEE Transactions on Software Engineering, IEEE Press, Piscataway, NJ, USA, v. 31, n. 10, p. 804–818, 2005.

CECCATO, M.; MARIN, M.; MENS, K.; MOONEN, L.; TONELLA, P.; TOURWÉ , T. Applying and combining three different aspect Mining Techniques. Software Quality Control, Kluwer Academic Publishers, Hingham, MA, USA, v. 14, n. 3, p. 209–231, September 2006.

CLIFTON, C.; LEAVENS, G. T.; NOBLE, J. MAO: Ownership and Effects for More Effective Reasoning About Aspects. In: Proceedings of the European Conference on Object-Oriented Programming (ECOOP '07). [S.l.]: Springer-Verlag, 2007. p. 451–475.

D'ARCE, A. F.; GARCIA, R. E.; CORREIA, R. C. M.; ELER, D. M. Coordination Model to Support Visualization of Aspect-Oriented Programs. In: Proceedings of the 24th International Conference on Software Engineering and Knowledge Engineering (SEKE '12). [S.l.: s.n.], 2012. p. 168–173.

DO, H.; ELBAUM, S. G.; ROTHERMEL, G. Supporting Controlled Experimentation with Testing Techniques: An Infrastructure and its Potential Impact. Empirical Software Engineering: An International Journal, v. 10, n. 4, p. 405–435, 2005.

HUANG, J.; LU, Y.; YANG, J. Aspect Mining Using Link Analysis. In: Proceedings of the 5th International Conference on Frontier of Computer Science and Technology (FCST '10). Washington, DC, USA: IEEE Computer Society, 2010. p. 312–317.

KATTI, A.; BINGI, V.; CHAVAN, V. Application of Program Slicing for Aspect Mining and Extraction – A Discussion. International Journal of Computer Applications, Foundation of Computer Science, New York, USA, v. 38, n. 4, p. 12–15, January 2012.

KICZALES, G.; HILSDALE, E.; HUGUNIN, J.; KERSTEN, M.; PALM, J.; GRISWOLD, W. G. An Overview of AspectJ. In: Proceedings of the 15th European Conference on Object-Oriented Programming (ECOOP '01). London, UK, UK: Springer-Verlag, 2001. p. 327–354.

KICZALES, G.; LAMPING, J.; MENDHEKAR, A.; MAEDA, C.; LOPES, C. V.; LOINGTIER, J.-M.; IRWIN, J. Aspect-Oriented Programming. In: Proceedings of the European Conference on Object-Oriented Programming (ECOOP '97). [S.l.]: Springer-Verlag, 1997. p. 220–242.

KRINKE, J.; BREU, S. Aspect Mining Based on Control-Flow. In: Proceedings of the 7th Workshop Software Reengineering (WSR '05). Bad Honnef, Germany: GI-Softwaretechnik-Trends, 2005. v. 25, p. 39–40.

MARIN, M.; DEURSEN, A. V.; MOONEN, L. Identifying Crosscutting Concerns Using Fan-In Analysis. ACM Transactions on Software Engineering and Methodology (TOSEM), v. 17, n. 1, p. 3:1–3:37, December 2007.

MASSICOTTE, P.; BADRI, L.; BADRI, M. Towards a Tool Supporting Integration Testing of Aspect-Oriented Programs. Journal of Object Technology, v. 6, n. 1, p. 67–89, 2007.

MENS, K.; KELLENS, A.; KRINKE, J. Pitfalls in Aspect Mining. In: Proceedings of the 15th Working Conference on Reverse Engineering (WCRE '08). Washington, DC, USA: IEEE Computer Society, 2008. p. 113–122.

PFEIFFER, J.-H.; GURD, J. R. Visualisation-Based Tool Support for the Development of Aspect-Oriented

Programs. In: Proceedings of the 5th International Conference on Aspect-Oriented Software Development (AOSD ’06). New York, NY, USA: ACM, 2006. p. 146–157.

PORTO, D.; MENDONÇA, M. G.; FABBRI, S. C. P. F. CRISTA: A Tool to Support Code Comprehension Based on Visualization and Reading Technique. In: Proceedings of the 17th IEEE International Conference on Program Comprehension (ICPC ’09). [S.l.: s.n.], 2009. p. 285–286.

SANT’ANNA, C. N.; GARCIA, A. F.; CHAVEZ, C. v. F. G.; LUCENA, C. J. P. de; STAA, A. v. On the Reuse and Maintenance of Aspect-Oriented Software: An Assessment Framework. In: Proceedings XVII Brazilian Symposium on Software Engineering. [S.l.: s.n.], 2003.

TONELLA, P.; CECCATO, M. Aspect Mining Through the Formal Concept Analysis of Execution Traces. In: Proceedings of the 11th Working Conference on Reverse Engineering (WCRE ’04). Washington, DC, USA: IEEE Computer Society, 2004. p. 112–121.

TOURWÉ , T.; MENS, K. Mining Aspectual Views Using Formal Concept Analysis. In: Proceedings of the 4th IEEE International Workshop Source Code Analysis and Manipulation (SCAM ’04). Washington, DC, USA: IEEE Computer Society, 2004. p. 97–106.

Fernanda Madeiral Delfim é bacharel em Ciência da Computação pela Universidade do Oeste Paulista (2010). Atualmente é aluna regular do mestrado em Ciência da Computação da Universidade Estadual Paulista “Júlio de Mesquita Filho” (UNESP), atuando principalmente nos seguintes temas: Engenharia de Software, Refatoração de Código, Programação Orientada a Aspectos e Visualização de Software.

Rogério Eduardo Garcia concluiu o doutorado em Ciência da Computação e Matemática Computacional pela Universidade de São Paulo [ICMC-USP-São Carlos] em 2006. Atualmente é Professor Assistente da Universidade Estadual Paulista “Júlio de Mesquita Filho” (UNESP), lotado no Departamento de Matemática e Computação da Faculdade de Ciência e Tecnologia de Presidente Prudente. Atua na área de Ciência da Computação, com ênfase em Metodologia e Técnicas da Computação, principalmente nos seguintes temas: Engenharia de Software e Visualização de Informação e Científica.