Mineração de Aspectos: Conceitos, Métodos e Ferramentas

Leandro Maciel Almeida, Thereza Patrícia Pereira Padilha

Laboratório de Inteligência Computacional – Centro Universitário Luterano de Palmas
Caixa Postal 160 – 77.054-970 – Palmas – TO – Brasil

{leandro, thereza}@ulbra-to.br

Abstract. This paper presents an overview about Aspect Mining area describing concepts related to aspect-oriented programming, existing mining methods and functioning of the AMT/AMTEX and FEAT mining tools .

Resumo. *Este trabalho apresenta uma visão geral da área de Aspect Mining descrevendo os conceitos relacionados à programação orientada a aspectos, os métodos de mineração existentes, bem como o funcionamento das ferramentas de mineração AMT/AMTEX e FEAT.*

1. Introdução

A área de mineração de aspectos está diretamente relacionada com o advento do novo paradigma de programação chamado de programação orientada a aspectos (*aspect-oriented programming* - AOP). Apesar da maioria dos sistemas desenvolvidos seguir o paradigma orientado a objetos (OO), muitos desenvolvedores buscam a incorporação da AOP em seus sistemas, isto é, a adoção de aspectos devido à redução da complexidade, aumento da legibilidade e melhora da modularização, tornando-o mais maleável para futuras evoluções.

Para a incorporação de AOP em sistemas OO, é necessário realizar uma análise para a identificação de trechos de códigos a serem convertidos em aspectos. Essa análise de forma manual é uma tarefa bastante árdua e tendenciosa a erros. Diante da dificuldade de se explorar várias classes de um sistema que, possivelmente estão dispersas em muitos arquivos, tendo cada classe um tamanho variável em termos de atributos, métodos e linhas de códigos, surgiu a necessidade de automação do processo de exploração. Para automatizar esse processo, estão sendo desenvolvidos diversos métodos e ferramentas de mineração de aspectos.

De um modo geral, a mineração de aspectos visa descobrir uma lógica entrelaçada em grandes bases de códigos com ou sem a intervenção do usuário. A mineração de aspectos também pode ser vista como um processo de análise de códigos existentes, com o propósito de identificar alguns *crosscutting concerns* (conjunto de aspectos relacionados às propriedades que afetam o comportamento do sistema, tais como: persistência, distribuição e tolerância à falhas) que podiam ser extraídos ou re-implementados como um aspecto [Deursen et al. 2003; Loughran and Rashid 2002].

Devido ao crescente interesse dos desenvolvedores para com a programação orientada a aspectos, bem como os benefícios adquiridos com a incorporação de aspectos em sistemas OO, auxiliado por ferramentas de mineração, a busca por conhecimentos dessa nova área de pesquisa tem aumentado significativamente. Perante

esse contexto, este trabalho tem como finalidade realizar uma investigação sobre conceitos, métodos e ferramentas relativos à mineração de aspectos.

Este trabalho está organizado da seguinte forma: na seção 2 é apresentada uma visão geral da programação orientada a aspectos, juntamente com a linguagem AspectJ. Na seção 3 são apresentados alguns métodos e ferramentas de mineração de aspectos. Na seção 4 são descritos os testes realizados com as ferramentas de mineração de aspectos selecionadas (AMT/AMTEX e FEAT). Na seção 5 são apresentadas as conclusões referentes ao desenvolvimento desse trabalho.

2. Programação Orientada a Aspectos e AspectJ

A presença de modularização em um sistema consiste na existência de construções lógicas que agrupam classes, associações e generalizações, incorporando assim uma perspectiva ou visão de uma situação [Rumbaugh et al. 1994]. Dessa forma, o aumento da modularização provido pela utilização da AOP traz alguns benefícios importantes, tais como facilidades de compreensão do sistema antes, durante e depois do seu desenvolvimento pois, as propriedades do sistema possuem implementação em apenas uma unidade ou seção de código [Czarnecki and Eisenecker 2000 apud Piveta 2001].

Para um melhor entendimento da AOP é preciso conhecer as duas visões possíveis de uma propriedade a ser implementada no desenvolvimento de um sistema, componente e aspecto [Irwin et al. 1997 apud Piveta 2001]:

- **componente** é uma visão de uma propriedade se a mesma puder ser encapsulada em um procedimento genérico (método, objeto, API ou função). Os componentes normalmente são unidades da decomposição funcional do sistema, que podem ser: usuários, mensagens e entre outros.
- **aspecto** é outra visão não restrita apenas a unidades de decomposição funcional do sistema. Os aspectos são propriedades que envolvem diversas partes de um sistema, afetando sistematicamente os componentes funcionais existentes.

A AOP procura solucionar a deficiência encontrada na captura de importantes decisões de projeto que um sistema deve implementar, normalmente não capturadas com o emprego da POO. Isso ocorre com o aumento da modularização obtido com AOP com a separação do código que implementa funções específicas que afetam diferentes partes do sistema, chamadas de interesses ortogonais (*crooscutting concerns*) ou aspectos, do código de componentes ou código de negócio. Normalmente a AOP é utilizada para a implementação de aspectos que geralmente são requisitos não-funcionais de um sistema. Conseqüentemente tem-se dos requisitos funcionais um conjunto de componentes expresso em uma linguagem de programação. Enquanto dos requisitos não-funcionais tem-se geralmente um conjunto de aspectos relacionados às propriedades que afetam o comportamento do sistema.

No desenvolvimento orientado a aspectos, além da identificação de requisitos funcionais e não-funcionais, são encontrados: uma linguagem de programação de componentes e uma de aspectos, um conjunto de classes escritas em linguagem de componentes e outro conjunto em linguagem de aspecto e um combinador de aspectos. Para o trabalho com AOP existem implementações em diferentes linguagens como, por exemplo, C++, Smalltalk, C#, C, J#, VB.NET e Java. Dentre essas linguagens, a linguagem que ganhou mais interesse pela comunidade foi Java e, conseqüentemente,

surgiram várias ferramentas que oferecem suporte para AOP em Java. A mais conhecida é AspectJ [Kiczales et al. 2001].

AspectJ é uma extensão de Java que permite a programação orientada a aspectos. Em AspectJ, um aspecto é visto como a principal construção, podendo afetar diversas partes de um sistema. Um aspecto pode conter atributos e métodos, assim como classes em Java, além de uma hierarquia de aspectos por meio da definição de aspectos especializados [Soares and Borba 2003]. Os aspectos em AspectJ podem afetar tanto a estrutura estática quanto à estrutura dinâmica de um sistema. A alteração da estrutura estática de sistema por aspectos é caracterizada pela adição de atributos, métodos e construtores em uma ou mais classes, alteração da hierarquia de classes entre outras. A estrutura dinâmica de um sistema pode ser afetada por um aspecto através da interceptação de mensagens por meio de pontos no fluxo de execução (*join points*) com a adição de comportamentos, além da obtenção de pleno controle sobre um ponto de execução [Soares and Borba 2003].

Um aspecto em AspectJ, usualmente, contém a definição de *pointcuts* que selecionam os *join points* e os seus valores, juntamente com os comportamentos especificados nos *advices*. Um *join point* é especificado como um ponto bem definido no fluxo de execução de um programa, além de ser considerado um elemento fundamental para a construção de mecanismos com AspectJ [Kiczales et al. 2001]. Os *join points* são utilizados para a programação de aspectos que afetam a estrutura dinâmica de um programa, e podem ser enxergados como o ponto exato onde o fluxo de execução de um programa será interrompido para a atuação de um referido aspecto. Um *pointcut* é visto como um conjunto de *join points* usando alguns operadores lógicos como, por exemplo, && (e), || (ou) e ! (não) [Kiczales et al. 2001]. Os *advices* são mecanismos utilizados para declaração de códigos a serem executados em cada *join point* de um *pointcut* [Kiczales et al. 2001].

A partir desses componentes de AspectJ, *join pont*, *pointcut* e *advice*, é possível construir um aspecto em AspectJ que possua pontos de execução específicos e claros, bem como atuações a serem realizadas na localização dos pontos de fluxo de execução de um sistema. A linguagem AspectJ fornece um conjunto extenso de variações para *join points*, *pointcuts* e *advices*, a relação completa dessas variações e das suas respectivas definições são encontradas em]Kiczales et al. 2001].

3. *Aspect Mining*

A mineração de aspectos visa identificar linhas de código candidatas a aspetos presentes em um código legado[1] e, com a interação de um usuário (desenvolvedor), elicitar os aspectos que realmente refletem a sua intenção. A identificação de aspectos é considerada a fase inicial para a transformação de sistemas OO em sistemas orientados a aspectos. Os principais motivos da busca pela transformação de sistemas orientados a objetos para orientados a aspectos é que com a adoção de AOP se ganha uma redução da complexidade, aumento da legibilidade e modularidade [Shepherd et al. 2004].

[1] Código fonte mantido para o funcionamento e evolução do sistema.

Os principais métodos de mineração são:

- **baseado em texto**: procura a partir de uma convenção de nomes para tipos, métodos, variáveis e classes [Hanneman and Kiczales 2001];
- **baseado em tipo**: procura a partir dos tipos de dados utilizados no código-fonte [Hannemann and Kiczales 2001];
- **multi-modal**: é uma combinação de vários tipos de métodos. Um exemplo de método multi-modal é a união do método baseado em texto e baseado em tipo. Dessa maneira, a análise é feita sob dois parâmetros: texto e tipo;
- **baseado em comparação e filtragem**: é um método automático que não necessita da interação do usuário. Esse método procura clones presentes no sistema e, por fim, ao usuário são apresentados trechos de códigos candidatos à reformulação em aspectos [Shepherd et al. 2004].

3.1. Principais Ferramentas de *Aspect Mining*

As ferramentas de *Aspect Mining* são usadas para suportar uma análise quantitativa e qualitativa de entrelaçamento e espalhamento de implementação em códigos legados, principalmente. Para este trabalho foram escolhidas três ferramentas de *Aspect Mining*. Os critérios utilizados para a seleção dessas ferramentas se basearam fundamentalmente pela existência de uma documentação e pelo respalde em trabalhos relacionados.

AMT/AMTEX

AMT é a ferramenta precursora para mineração de aspectos, desenvolvida por Hannemann e Kiczales em 2001 [Hannemann and Kiczales 2001]. AMT é um *framework* aberto de análise multi-modal e, atualmente, dispõe dois métodos de mineração, que são: baseado em texto e em tipo. O método baseado em texto oferece uma simples unificação de padrões. O método baseado em tipo, por outro lado, detecta códigos entrelaçados e apresenta medidas de qualidade de modularidade tal como coerência.

Os códigos dos programas são representados usando o conceito de Seesoft [Eick 1992]. Cada unidade de compilação (classe) é representada como uma coleção de faixas horizontais que correspondem as linhas “relevantes” do código-fonte. Uma linha de código é considerada relevante quando não possui comentário ou em branco. A ferramenta possibilita ao usuário realizar consultas a partir de expressões regulares ou de tipos usados, apresentando as linhas que unificam (*matching*) com cores específicas.

A ferramenta AMT utiliza, basicamente, dois componentes: analisador (AMTAnalyzer) e visualizador (AMTVisualizer). O analisador usa uma versão modificada do compilador do AspectJ para extrair informações de cada linha do código-fonte. Esse analisador captura todas as informações “relevantes” em cada linha de código e, depois as armazena em um arquivo de texto denominado AMTResults.amt. O visualizador utiliza o arquivo criado pelo analisador para apresentar o sistema graficamente em forma de linhas.

AMTEX é uma ferramenta estendida da AMT que possui novas funcionalidades de visualização através de um mecanismo de multi-visualizador [Zhang and Arno 2003]. Na AMTEX é possível, por exemplo, solicitar uma lista de classes que contém sincronização ou requerer as N classes mais utilizadas na aplicação, isto é, as classes que estão mais entrelaçadas.

FEAT

A ferramenta FEAT é um *plug-in* para o ambiente de desenvolvimento Eclipse e foi desenvolvida com o propósito de oferecer suporte para a tarefa de busca por aspectos presentes no código-fonte usando grafos de aspecto (*concern graphs*). Segundo Robillard e Murphy, grafos de aspecto constituem em uma representação mais efetiva do que a simples ilustração de linhas do código-fonte para a análise de aspectos em um sistema [Robillard and Murphy 2002]. Um grafo de aspecto separa detalhes da implementação de um aspecto através do armazenamento da sua estrutura chave. Nessa estrutura são apresentados os relacionamentos existentes entre os diferentes elementos de um aspecto, classes, métodos e atributos [Robillard and Murphy 2002].

A formalização de um grafo de aspecto é baseada em um modelo de programa que pode ser extraído automaticamente a partir do código-fonte e em diferentes níveis de proporção. Um modelo de programa descreve a declaração e o uso de diversos elementos de um sistema desenvolvido em OO. As definições dos formalismos de um modelo de programa são discutidas em [Robillard and Murphy 2002]. Dentre esses formalismos, existe a especificação de um vértice *all-of* ou *part-of* que indica, respectivamente, se um vértice pertence totalmente ou parcialmente a um aspecto. Existe também a especificação de um arco que usualmente define operações realizadas a partir da sua origem, criando, alterando ou lendo informações dos seus destinatários.

FEAT auxilia também no gerenciamento de possíveis aspectos encontrados e reformulados. A FEAT possui, por conseqüência do uso de grafos de aspecto, um resumo simples, compacto e descritivo a respeito de aspectos. Esse resumo é simples porque os grafos de aspecto resumem detalhes de implementação de classes separando sintaticamente sentenças como palavras-chave *call* ou *reads*. O resumo é compacto porque grafos de aspecto permitem uma visão local de aspectos eliminando códigos que não representam interesses, e, por fim, é descritivo, pois existe uma explícita documentação das relações encontradas entre diferentes elementos do programa [Robillard and Murphy 2002].

4. Resultados e Discussões

Nesta seção serão apresentados alguns experimentos realizados com as ferramentas AMT/ATMEX e FEAT usando o JHotDraw. Esses experimentos buscam uma identificação de códigos candidatos a aspectos que alteram o valor do atributo `figura` e que armazenam o valor do atributo `figura`. Vale ressaltar que o JHotDraw permite ao usuário criar desenhos tradicionais como elipses, retângulos etc, logo, para isso, existe o atributo `figura` que representa o que é criado/manipulado pelo usuário. A fase em que os valores do atributo `figura` são armazenados representa a etapa em que

a figura é salva em um arquivo pelo usuário. A fase em que ocorrem alterações na figura criada pelo usuário corresponde às alterações realizadas no atributo `figura`.

AMT/AMTEX

Para realizar os experimentos utilizando a ferramenta AMT e o seu multi-visualizador AMTEX é necessário submeter todas a classes ao *AMTAnalyzer* para, posteriormente, realizar a construção de consultas usando expressões regulares. Para executar o primeiro experimento, após a análise em todas as classes pelo *AMTAnalyzer*, foi construída uma consulta que busca por todas as classes que tenham presentes em seu nome a palavra Attribute. Essa consulta é formalizada pela seguinte expressão regular: .**class* .**Attribute.**. A partir disso, o componente *AMTAnalyzer* realiza a busca e o AMTVisualizer apresenta o resultado, assim como ilustrado na Figura 1. As barras brancas representam as classes e as linhas destacadas exibem os pontos que possuem ocorrências das buscas submetidas.

No exemplo da Figura 1, foi identificada, a princípio, a classe `ChangeAttributeCommand` que realiza alterações no atributo `figura` por meio do método `execute()`. Essa identificação foi realizada com a execução da primeira consulta e com a busca por utilizações da classe `FiguresAttributes`. Após isso, entre as classes encontradas, foram identificadas as classes que estão relacionadas com alterações dos valores do atributo `figura`. Com a identificação da classe `ChangeAttributeCommand`, realizavam-se consultas para identificar os pontos de utilização no sistema da mesma, encontrando assim um novo conjunto de classes. Esse conjunto consiste em dez classes como é apresentado na opção CUs matching search, Figura 1.

Para a identificação da real utilização, por parte de outros componentes do sistema, do método `execute()`é necessário realizar a abertura de todos os arquivos encontrados, verificando a existência ou não de chamadas ao método `execute()`. Esse processo é demorado, cansativo e tendencioso a erros, pois caso sejam identificadas chamadas ao método, é necessário então relatá-las para, posteriormente, serem úteis na definição de um aspecto.

A ferramenta AMT dispõe de poucos métodos e variações de buscas, isso, a princípio, é devido à ferramenta ser a precursora na identificação de códigos candidatos a aspectos. Com o objetivo de suportar o trabalho em sistemas expressivamente grandes surgiu a AMTEX que é um multi-visualizador que trabalha com o analisador da AMT. Tanto a AMT quanto AMTEX fornecem buscas por tipos definidos e buscas expressas por expressões regulares, o que degrada largamente o trabalho do desenvolvedor, caso o mesmo não conheça com exímio a utilização de expressões regulares.

Figura 1. Execução de Consultas e Visualização de Resultados na Ferramenta AMT.

Os benefícios são limitados utilizando a ferramenta AMT/AMTEX para a identificação de códigos candidatos a aspectos, pois os meios de buscas oferecidos para o usuário são restritos. Um exemplo dessa restrição é a incapacidade de construções complexas de consultas para sanar as deficiências da não localização de métodos específicos de uma classe. Como conseqüência disso, o desenvolvedor pode não conseguir um auxílio efetivo na identificação de aspectos, ocasionando, inclusive, em certos momentos, uma complicação pelo fato de ser necessário o conhecimento da versão das expressões regulares da ferramenta.

A ferramenta AMT/AMTEX fornece uma pequena quantidade de informações a respeito dos detalhes de implementação de um sistema. Isso decorre da existência de poucos métodos de mineração e, conseqüentemente, da ausência de construtores de relacionamentos entre os diversos elementos do sistema, tais como atributos, métodos e classes.

FEAT

Para executar a ferramenta FEAT é necessário criar um grafo de aspectos raiz que comporta todos os grafos responsáveis pelo agrupamento de informações sobre a alteração e o armazenamento do atributo `figura`. Para isso, foi criado um grafo denominado Figura que agrega os grafos de aspecto Alterações e Armazenamento.

Com o intuito de identificar as classes que implementam métodos de alteração e armazenamento dos valores do atributo `figura`, foi realizada uma busca por `*Attribute*`. Com essa busca, as classes `AttributeFigure`, `FigureAttributes` e `ChangeAttributeCommand` foram encontradas.

A classe `ChangeAttributeCommand`, por exemplo, possui implementações que alteram valores do atributo `figura` e, com isso foi adicionada no grafo de aspecto Alterações. A partir disso inicia-se a busca por pontos existentes no sistema que utilizam essa classe para realizar alterações no atributo `figura`. Para essa tarefa, a ferramenta disponibiliza várias espécies de consultas que retornam, por exemplo, relacionamentos de um elemento com todos os outros componentes do sistema (*Fan-in*) e relacionamentos de componentes do sistema com um elemento (*Fan-out*).

Para detectar os pontos no sistema que realizam alterações no atributo `figura` por meio da classe `ChangeAttributeCommand`, foi realiza, primeiramente, uma busca por todos os elementos presentes no sistema que fazem chamadas ao construtor da classe `ChangeAttributeCommand`. A partir disso, obteve-se uma relação de todas as classes e métodos que fazem chamadas ao referido construtor, como ilustrado na Figura 2 em *Projection*. A visão *Projection* é responsável, principalmente, por apresentar os resultados de consultas realizadas pelo usuário.

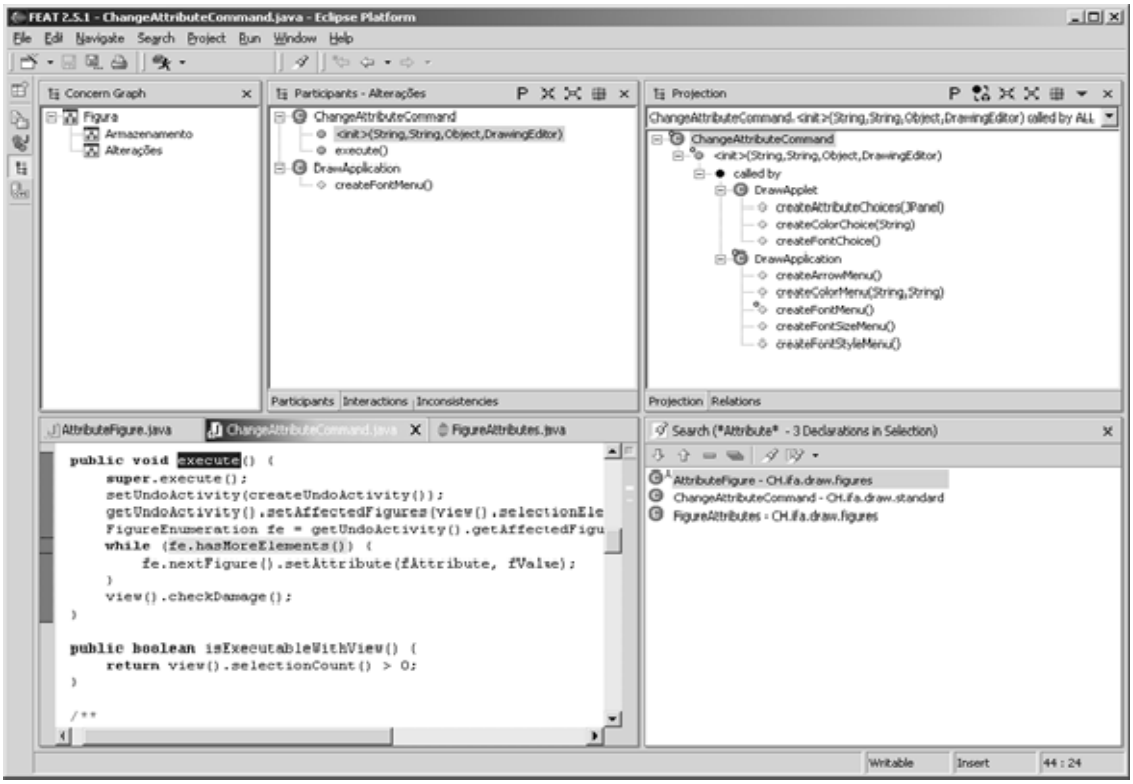

Figura 2. Busca de Códigos Candidatos a Aspectos na Ferramenta FEAT

Para ter acesso ao código-fonte a partir dos elementos das visões *Participants*, *Projection* ou *Relations*, o usuário precisa selecionar o elemento desejado e a ferramenta apresenta o seu código no editor. Essa facilidade auxilia bastante o trabalho do usuário, pois em qualquer momento, se necessário, é possível visualizar os elementos

encontrados em uma consulta realizada, mesmo que essa consulta não tenha nenhum de seus elementos presentes em um grafo de aspecto.

5. Conclusões

Os aspectos em uma aplicação não são tão fáceis de serem identificados, pois exige uma interação do desenvolvedor para decidir se um método ou atributo faz parte de um determinado interesse. Realizar a identificação manual de aspectos é uma tarefa tendenciosa a erros, cansativa e desgastante que, em muitos casos, pode não identificar claramente a presença ou não de aspectos, principalmente quando se trabalha com sistemas legados. Para essa tarefa o desenvolvedor pode contar com o auxílio de uma das várias ferramentas que estão disponíveis para localização ou mineração de aspectos. As ferramentas AMT/AMTEX, FEAT, Aspect Browser e JQuery se encaixam no quadro das ferramentas que necessitam de interação com o usuário para a localização de aspecto. Buscas por aspectos sem a necessidade do desenvolvedor são encontradas somente em métodos automáticos considerados pela comunidade bastante promissores, mas que ainda encontram-se em estudo.

Com realização dos experimentos, pôde-se observar que a ferramenta FEAT fornece maiores informações a respeito de detalhes de implementação, facilitando assim a definição de um aspecto. Um aspecto de sincronização para o atributo `figura`, por exemplo, poderia ser incorporado facilmente de acordo com as informações da utilização do atributo fornecido pela ferramenta. Em contrapartida, a ferramenta AMT/AMTEX não prove esse tipo de suporte para a incorporação de aspectos, tornando a formalização do aspecto de sincronização mais complicada e possivelmente menos eficiente.

A ferramenta AMT/AMTEX diante dos experimentos realizados apresentou-se como complexa em termos de utilização, incompleta em termos de métodos de mineração e pouco efetiva no auxílio provido ao desenvolvedor. Porém, a FEAT incorpora vários outros mecanismos que possibilitam a busca por relacionamentos que vão desde uma classe até a um atributo. A detecção de relacionamentos de um determinado elemento para com os demais acontece a partir da especificação de uma busca, que na prática contém maiores informações no seu retorno do que as buscas realizadas com a AMT.

Nesses experimentos, de um modo geral, observou-se que a ferramenta FEAT possui superioridade em relação à AMT/AMTEX devido à flexibilidade de consultas e auxílio efetivo à identificação de aspectos. Por outro lado, a ferramenta AMT fornece uma visão geral do espalhamento e entrelaçamento do código no sistema devido às características visuais. Portanto, é difícil determinar a melhor ferramenta para a mineração de aspectos em sistemas OO, visto que cada ferramenta possui características importantes e também porque existem outras ferramentas disponíveis na literatura.

Referências Bibliográficas

Czarnecki K. and Eisenecker, U. (2000) “Generative Programming: Methods, Tools, and Applications”. Addison-Wesley, Boston.

Deursen, A., Marin, M. and Moonen, L. (2003) “Aspect Mining and Refactoring”. The First International Workshop on Refactoring: Achievements, Challenges, Effects (REFACE), Canadá.

Eick, G., Steffen, L. and Summer, E. (1992) “Seesoft – A Tool for Visualizing Line-Oriented Software Statistics”. IEEE Transactions of Software Engineering.

Hannemann, J. and Kiczales, G. (2001) “Overcoming the Prevalent Decomposition in Legacy Code”. Workshop on Advanced Separation of Concerns in Software Engineering at (ICSE).

Irwin, J., Kickzales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C. and Loingtier, J. (1997) “Aspect-Oriented Programming”. Proceeding da ECOOP’97, Finland: Springer-Verlag.

Kiczales, G., Hilsdale, E., Hugunin, J., Kerten, M., Palm, J. and Griswold. (2001) “An Overview of AspectJ”. Proceeding da ECOOP, Springer-Verlag.

Loughran, N. and Rashid, A. (2002) “Mining Aspects”. Workshop on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design.

Piveta, E. K. (2001) “Um Modelo de Suporte a Programação Orientada a Aspectos”. Dissertação de Mestrado, Universidade Federal de Santa Catarina.

Robillard, M. and Murphy, G. (2002) “Concern graphs: Finding and describing concerns using structural program dependencies”. Proceedings of the Internacional Conference on Object-Oriented Programming (ICSE).

Rumbaugh, J., Blaha, M., Permerlani, W., Eddy, F. and Lorense, W. (1994) “Modelagem e Projetos Baseados em Objetos”. Rio de Janeiro, Campus.

Shepherd, D., Gibson, E. and Pollock, L. (2004) “Desing and Evaluation of an Automated Aspect Mining Tool”. IEEE Mid-Atlantic Symposium on Program Languages and Systems (MASPLAS).

Soares, S. and BORBA, P. (2003) “AspectJ – Programação orientada a aspectos em Java”. Simpósio Brasileiro de Linguagens de Programação, Rio de Janeiro.

Zhang, C. and Arno, J. (2003) “A Prism for Research in Software Modularization Through Aspect Mining”. Technical Communication, Middleware Systems Research Group, University of Toronto.