APIMiner 2.0: Recomendação de Exemplos de Uso de APIs usando Regras de Associação

Hudson S. Borges[1], Daniel Carlos H. Felix[1], João E. Montandon[1], Heitor Augustus Xavier Costa[2], Marco Tulio Valente[1]

[1]Departamento de Ciência da Computação, UFMG
[2]Departamento de Ciência da Computação, UFLA

{hsborges,dfelix,joao.montandon,mtov}@dcc.ufmg.br, heitor@dcc.ufla.br

Resumo. *A utilização de exemplos de uso é um recurso chave para a compreensão de APIs. Neste artigo, apresenta-se a plataforma APIMiner 2.0, que se baseia em regras de associação envolvendo elementos de APIs para extração e agregação automatizada de exemplos em documentações no formato JavaDoc. Tais regras são obtidas a partir da mineração de um conjunto de sistemas cliente. Uma instância da plataforma para a API de desenvolvimento do Android foi criada e também é descrita neste artigo.*

Abstract. *An interesting resource for understanding APIs is to present usage examples. In this paper, APIMiner 2.0 platform is presented. This platform is based on association rules involving API elements for automated extracting and aggregating of examples in JavaDoc documents. These rules are obtained from mining of set of client systems. An instance of APIMiner 2.0 platform was created for Android and it was presented.*

1. Introdução

Atualmente, a utilização de código de terceiros no processo de desenvolvimento de sistemas consolidou-se como uma prática comum. Geralmente disponíveis na forma de bibliotecas e *frameworks*, a utilização desse tipo de código ocorre por meio de uma API (*Application Programming Interface*). Entretanto, a inexistência de documentação adequada leva a um contexto em que a compreensão e domínio de APIs têm se mostrado cada vez mais difícil [Scaffidi 2006, Robillard and DeLine 2011, Buse and Weimer 2012].

Nesse artigo, apresenta-se a plataforma APIMiner 2.0, que estende a plataforma APIMiner [Montandon and Valente 2012] utilizando regras de associação envolvendo elementos de APIs para extração e agregação automatizada de exemplos de uso em documentação, disponibilizada no formato JavaDoc. Dentre suas principais características, destacam-se:

- Fornecimento de exemplos de uso que envolvem mais de um método da API que permitem aos desenvolvedores identificar diferentes formas de usar essa API. Essa característica faz com que os exemplos apresentados pela plataforma APIMiner 2.0 sejam mais úteis aos desenvolvedores que exemplos de uso incluindo métodos únicos [Robillard and DeLine 2011, Buse and Weimer 2012];
- A utilização de diversos artifícios para melhorar a legibilidade de exemplos de uso obtidos por meio de técnicas de *slicing* estático. Por exemplo, problemas como a não identificação de variáveis em um *slicing* tradicional são contornados com a inserção de declarações abstratas de variáveis;

- A utilização de técnicas de mineração de dados, mais precisamente mineração de conjuntos de itens frequentes e regras de associação, com o objetivo de identificar co-ocorrências de chamadas de métodos de uma API. Por exemplo, se um determinado usuário de uma API solicitar exemplos de uso para o método `M1()` e, no processo de mineração, identificou-se que na maioria das vezes esse método é utilizado em conjunto com o método `M2()`. Assim, são apresentados exemplos de uso que envolvem esses dois métodos como recomendações ao usuário.

Uma instância da plataforma APIMiner 2.0 foi criada para a API de desenvolvimento do Android, denominada Android APIMiner. Nessa instância, há 287.263 exemplos de uso extraídos de 155 projetos, sendo 21.598 exemplos de uso para 1.952 regras de associação envolvendo conjuntos de métodos que são frequentemente chamados juntos.

O restante deste artigo está organizado conforme descrito a seguir. Na Seção 2, apresenta-se uma visão geral da solução proposta pela plataforma APIMiner 2.0. Na Seção 3, é apresentada a instância da plataforma APIMiner 2.0 para a API de desenvolvimento do Android. Na Seção 4, descreve-se brevemente algumas ferramentas relacionadas. Na Seção 5, são apresentadas conclusões finais do artigo.

2. APIMiner 2.0: Visão Geral

A plataforma APIMiner 2.0 está organizada em duas fases principais: i) Pré-processamento; e ii) Apresentação. Na fase de pré-processamento, são realizadas a análise, a extração e o armazenamento dos padrões de uso e dos exemplos de uma API. Na fase de apresentação, ocorre a instrumentação do JavaDoc original da API, permitindo que recomendações de padrões e exemplos sejam recuperados por um usuário final da API. Na Figura 1, apresenta-se a arquitetura interna da plataforma, composta por cinco módulos principais:

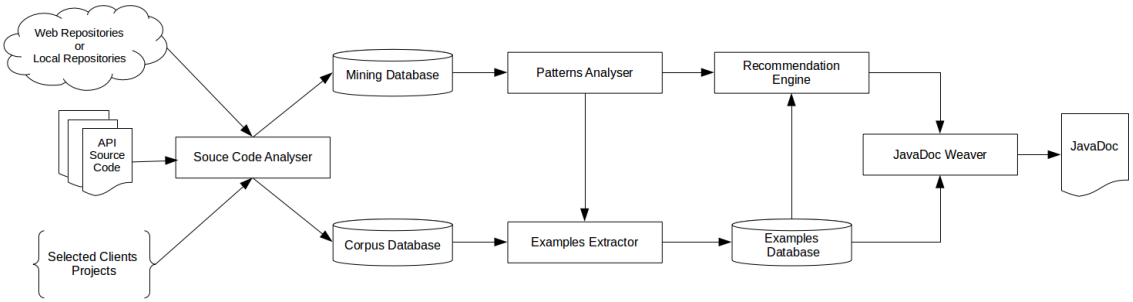

Figura 1. Arquitetura da Ferramenta

I **Source Code Analyser**. Nesse módulo, é analisado o código fonte da API e de sistemas cliente da API. Inicialmente, o módulo recebe o código fonte da API, extrai informações sobre suas classes e métodos e replica essas informações em duas bases de dados distintas. Os sistemas cliente são classificados em dois grupos: i) Sistemas para extração de padrões de uso; e ii) Sistemas para extração de exemplos. O primeiro grupo engloba sistemas que podem estar em fase de desenvolvimento, por exemplo, em uma versão *alpha* ou *beta*, e compõem a base de mineração de padrões da API (`Mining Database`). O segundo grupo engloba sistemas estáveis e bem conhecidos por usuários da API; esses sistemas compõem a base de sistemas para

extração de exemplos de uso de métodos da API (`Corpus Database`). A análise de código fonte é realizada utilizando o componente JDT da plataforma Eclipse;

II **Patterns Analyser**. Nesse módulo, são extraídos padrões de uso a partir das informações presentes na base de mineração. Um padrão de uso é representado por um conjunto de chamadas da API frequentemente utilizado no mesmo escopo. Além disso, esse módulo extrai outras informações e métricas que permitem avaliar e classificar esses padrões. Na Seção 2.1, são apresentados detalhes sobre as técnicas e os algoritmos utilizados nessa extração. Esse módulo utiliza os algoritmos disponibilizados pela ferramenta Weka para mineração de dados;

III **Examples Extractor**. Nesse módulo, são extraídos exemplos de uso da API na base de sistemas. Para isso, são analisados sequencialmente os métodos dos sistemas cliente; para cada método identificado, são extraídas informações sobre as chamadas a métodos da API. Para cada chamada identificada executa-se o processo de extração de exemplos. Para o conjunto de chamadas identificadas e seus subconjuntos, verifica-se se existem padrões de uso que envolvem tais métodos. Caso existam, é realizada a extração do exemplo para o subconjunto. O processo de extração de exemplos é detalhado na Seção 2.2;

IV **Recommendation Engine**. Nesse módulo, são estruturados os padrões de uso na forma de recomendações e são identificados e ordenados os exemplos de cada recomendação. Uma recomendação é constituída por uma premissa e uma consequência, denotando conjuntos de métodos da API que possuem relação de co-ocorrência. Para cada recomendação, são listados e ordenados os exemplos, sendo a ordenação feita por uma função que considera o número de chamadas e o número de linhas de código do exemplo;

V **JavaDoc Weaver**. Nesse módulo, é realizado o processo de instrumentação da documentação original da API no formato JavaDoc, gerando uma interface de apresentação dos exemplos e das recomendações extraídos dos módulos anteriores.

2.1. Padrões de Uso de Elementos de APIs

Padrões de uso de elementos de APIs representam relações de co-ocorrência de duas ou mais chamadas de métodos de uma API em um determinado escopo. Um exemplo típico de padrões de uso envolve a co-ocorrência de chamadas `open()` e `close()` no código fonte de sistemas cliente.

Em mineração de dados, a identificação de tais padrões é normalmente feita utilizando a técnica de mineração de itens frequentes. Essa técnica permite identificar conjuntos de elementos utilizando relações de co-ocorrência em uma base de dados. Um conjunto de elementos pode ser considerado frequente caso ocorra uma quantidade mínima de vezes. Além disso, o conhecimento desses conjuntos permite derivar regras de associação entre seus elementos [Agrawal and Srikant 1994]. Por exemplo, seja $A = \{m_1, m_2, ..., m_{100}\}$ o conjunto de métodos de uma API e $I = \{m_{17}, m_{22}, m_{65}\}$ um conjunto de itens frequentes minerado de uma base D contendo informações de uso da API em sistemas cliente. Seja ainda uma possível regra derivada desse conjunto como a seguinte: $m_{17} \Longrightarrow m_{22}, m_{65}$. O uso de regras de associação no contexto de recomendação de exemplos é relevante pois fornece informações importantes sobre relações de co-ocorrência no

uso da API. Essas relações podem ser utilizadas para extração e apresentação de exemplos mais úteis aos usuários.

2.2. Extração e Recomendação de Exemplos

O algoritmo para extração de exemplos utilizado pela plataforma APIMiner (desde sua versão 1.0 [Montandon and Valente 2012]) baseia-se no conceito de *slicing* estático [Weiser 1981]. Na versão 2.0, esse algoritmo foi aperfeiçoado para incrementar legibilidade dos exemplos extraídos. Para isso, foram incorporadas algumas soluções propostas originalmente no trabalho de Buse e Weimer [Buse and Weimer 2012], como inicialização abstrata para variáveis não identificadas no processo de *slicing* e identificação de padrões. No algoritmo de *slicing* estático, o objetivo é extrair as linhas de código relevantes do código fonte (*slice*). Para isso, o algoritmo recebe a linha de código que contém a chamada de interesse e extrai do código fonte as linhas anteriores que modificam as variáveis utilizadas pela chamada, visando preservar o comportamento do *slice*, e as linhas posteriores que fazem uso da variável retornada pela chamada [Lucia 2001]. A granularidade tratada no algoritmo implementado é método. O algoritmo de *slicing* recebe como parâmetro os métodos da API (incluindo construtores), os padrões de uso identificados por meio de uma técnica de mineração de dados e o código fonte do sistema cliente. Como resultado, o algoritmo fornece uma lista de exemplos que envolvem um ou mais métodos da API com base nos padrões de uso fornecidos na entrada.

O processo de recomendação de padrões e de exemplos é realizado com base em métricas extraídas durante o pré-processamento. As recomendações de padrões são feitas utilizando média ponderada dos valores de suporte (quantidade de vezes que o padrão aparece na base de dados), confiança (relação entre o suporte da premissa pelo suporte da regra), quantidade de elementos que compõem a regra e o coeficiente de similaridade de Jaccard. Esse coeficiente avalia a similaridade entre a premissa e a consequência. O critério de ordenação dos exemplos se baseia na quantidade de linhas de código e na quantidade de métodos presentes no exemplo. Além disso, há penalização para exemplos muito pequenos ou muito grandes. Para cada método da API e para cada padrão de uso recomendado, gera-se uma lista com os 20 exemplos com maior relevância.

3. Instância Android APIMiner

Uma instânciação da plataforma para a API do Android está disponível publicamente na web (`www.apiminer.org`). Ela dispõe de 287.263 exemplos de uso extraídos de 155 projetos, sendo 21.598 exemplos de uso para 1.952 recomendações de padrões de uso. Junto com os exemplos, fornecem-se informações sobre o projeto e o arquivo do qual foram obtidos. Os exemplos são extraídos de sistemas de código aberto.

A Figura 2 ilustra a utilização da plataforma APIMiner 2.0 para obter exemplos sobre o método `beginTransaction()` da classe `SQLiteDatabase`. Nesse caso, há recomendações de outros métodos frequentemente chamados em conjunto no *menu* `Frequently called with`, como `setTransactionSuccessful()` e `endTransaction()`, `endTransaction()` e `setTransactionSuccessful()`, respectivamente.

Ao selecionar a primeira recomendação, é apresentado ao usuário o exemplo de código mostrado na Figura 3. As chamadas da recomendação são destacadas no fragmento de código, que apresenta um uso típico destes métodos: i)

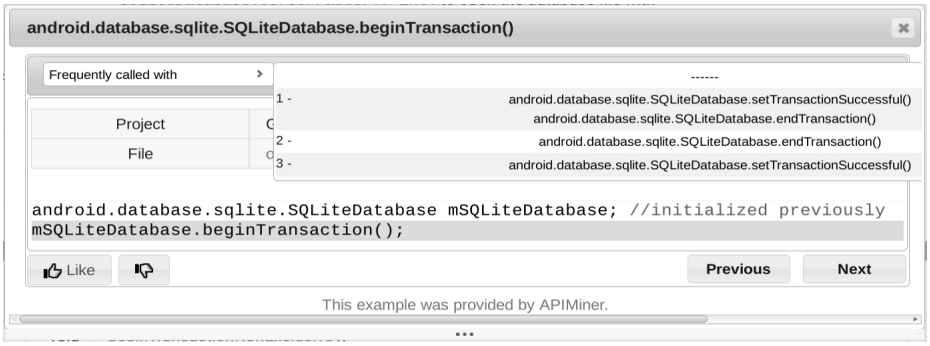

Figura 2. Exemplo de uso do método `beginTransaction()`

`beginTransaction()` para iniciar uma transação com o banco de dados; ii) `setTransactionSuccessful()` no interior de uma cláusula `try` para indicar que a transação foi realizada com sucesso; e iii) o uso de `endTransaction()` em uma cláusula `finally` para encerrar a transação com banco de dados.

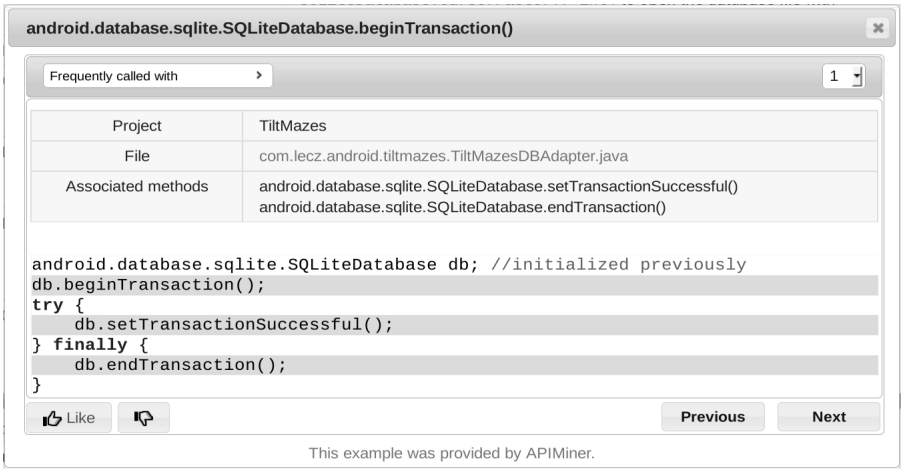

Figura 3. Exemplo para a recomendação conjunta de `beginTransaction()`, `setTransactionSuccessful()` e `endTransaction()`

Em ambos os exemplos, com a utilização do algoritmo de *slicing*, foram eliminadas linhas de código não diretamente relacionadas ao método analisado (o código completo pode ser observado ao clicar no nome do arquivo). Além disso, como o processo de *slicing* tradicional não conseguiu identificar a declaração da variável `db` no escopo do método de onde o exemplo foi extraído, acrescentou-se na linha 1 uma declaração de `db`.

4. Ferramentas Relacionadas

As ferramentas relacionadas com a plataforma APIMiner 2.0 podem ser organizadas em dois grupos principais: i) ferramentas de recomendação de exemplos; e ii) ferramentas de recomendação de uso de métodos de APIs. Nas ferramentas do primeiro grupo, são utilizadas técnicas de sumarização para obter e apresentar exemplos. Um exemplo dessas ferramentas é eXoaDocs [Kim et al. 2010] que insere exemplos de uso na documentação sumarizando código obtido por máquinas de busca. Nas ferramentas do segundo grupo, são identificados padrões de uso de elementos de APIs para recomendar exemplos por meio de IDEs de desenvolvimento. Um exemplo dessas ferramentas é

VCC [da Silva Jr et al. 2012] que utiliza mineração de sequências para extrair e recomendar trechos de código aos usuários. Entretanto, a mineração é realizada utilizando como entrada o próprio código do projeto que está sendo utilizado. A ferramenta foi implementada na forma de *plugin* para a IDE Eclipse e se vale de informações de contexto, fato que não ocorre com documentações abertas na web, como Java Docs.

5. Considerações Finais

Neste artigo, foi apresentada a plataforma APIMiner 2.0, que se baseia em padrões de uso de métodos de uma API, para recomendar exemplos mais úteis aos usuários. Tais padrões são obtidos por meio de mineração de itens frequentes, sumarizados usando um algoritmo de *slicing* estático combinado com características que ajudam a incrementar legibilidade dos exemplos, e recomendados a partir de dados obtidos por regras de associações derivadas de conjuntos de itens frequentes minerados previamente.

Um estudo de caso foi apresentado, utilizando uma instância da plataforma configurada para a API do sistema operacional Android que pode ser publicamente acessada no endereço `http://www.apiminer.org`. Atualmente, a plataforma tem recebido cerca de 10 mil acessos mensais. Como próximos passos, tem-se o planejamento e a execução de um experimento controlado para avaliar a relevância dos exemplos e dos padrões recomendados pela plataforma.

Agradecimentos: Este trabalho foi apoiado pela FAPEMIG, CAPES e CNPq.

Referências

Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules in large databases. In *20th International Conference on Very Large Data Bases*, pages 487–499.

Buse, R. P. L. and Weimer, W. (2012). Synthesizing API usage examples. In *34th International Conference on Software Engineering*, pages 782–792.

da Silva Jr, L. L. N., de Oliveira Alexandre Plastino, T. N., and Murta, L. G. P. (2012). Vertical code completion: Going beyond the current ctrl+ space. In *6th Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software*, pages 81–90.

Kim, J., Lee, S., won Hwang, S., and Kim, S. (2010). Towards an intelligent code search engine. In *AAAI Conference on Artificial Intelligence*.

Lucia, A. D. (2001). Program slicing: methods and applications. In *I International Workshop on Source Code Analysis and Manipulation*, pages 142–149.

Montandon, J. E. and Valente, M. T. (2012). APIMiner: Uma plataforma para recomendação de exemplos de uso de APIs. *III Congresso Brasileiro de Software: Teoria e Prática (Sessão de Ferramentas)*, pages 51–56.

Robillard, M. P. and DeLine, R. (2011). A field study of API learning obstacles. *Empirical Software Engineering*, 16(6):703–732.

Scaffidi, C. (2006). Why are APIs difficult to learn and use? *ACM Crossroads*, 12(4):4–4.

Weiser, M. (1981). Program slicing. In *5th International Conference on Software Engineering*, pages 439–449.