PIBIC/CNPq/UFCG-2010

MANUTENÇÃO DA CONFORMIDADE ARQUITETURAL DE SOFTWARE ATRAVÉS DE TÉCNICAS DE RECUPERAÇÃO DA INFORMAÇÃO

Gustavo Jansen de Souza Santos[1], Dalton Dario Serey Guerrero[2]

RESUMO

Este trabalho teve o objetivo de investigar e implementar técnicas de Recuperação de Informação, com o intuito de auxiliar o processo de manutenção da conformidade arquitetural de software em evolução. IRUtils, a ferramenta implementada, extrai o vocabulário do software, representando-o em um modelo de espaço vetorial, do qual pode-se extrair métricas essenciais para atividades referentes à manutenção da conformidade arquitetural. Medidas típicas implementadas incluem a similaridade entre entidades de baixo nível com módulos de alto nível, assim como entre entidades de baixo nível existentes no código-fonte.

Palavras-chave: conformidade arquitetural, recuperação de infomação, engenharia de software

MAINTENANCE OF ARCHITECTURE COMPLIANCE THROUGH INFORMATION RETRIEVAL TECHNIQUES

ABSTRACT

This work aimed to investigate and implement techniques of Information Retrieval, in order to assist the process of maintaining the architecture compliance of software during its evolution. IRUtils, the implemented tool, extracts the software vocabulary and represents it in a vector space model, where it is possible to extract metrics for activities related to maintenance of the architecture compliance. Typical implemented measures include the similarity between software entities, both in high- and low-level.

Keywords: architecture compliance, information retrieval, software engineering

INTRODUÇÃO

Um ponto crítico no projeto e na construção de qualquer sistema de software complexo é a sua arquitetura; isto é, sua organização bruta como uma coleção de componentes que interagem entre si (GARLAN, 1995). Uma boa arquitetura pode fazer com que o sistema satisfaça requisitos nas áreas de performance, portabilidade, escalabilidade e interoperabilidade; ao mesmo tempo que uma arquitetura mal definida pode dificultar a organização do processo de desenvolvimento e a compreensão dos requisitos do sistema.

Durante o desenvolvimento e evolução de um sistema, preservar ou modificar adequadamente a arquitetura de software é um desafio. Além disso, quando um sistema é desenvolvido por grandes equipes e num período longo, é comum a diferença entre a arquitetura e o próprio software. Esta diferença causa a deterioração da arquitetura (PERRY E WOLF, 1992), levando ao envelhecimento precoce do software (PARNAS, 1994). Este fato ocorre por diversos motivos, tais como: desconhecimento técnico ou conceitual dos desenvolvedores em relação ao sistema, requisitos conflitantes e atividades realizadas sob pressão por cumprimento de prazos.

A verificação de conformidade entre a implementação atual e a arquitetura é, portanto, um problema importante. Ferramentas precisam ser desenvolvidas para automatizar o processo de manutenção dessa

[1] Aluno do Curso de Ciência da Computação, Depto. de Sistemas e Computação, UFCG, Campina Gande , PB, E-mail: gustavojss@lcc.ufcg.edu.br
[2] Ciência da Computação, Prof. Doutor, Depto. de Sistemas e Computação, UFCG, Campina Grande, PB, E-mail: dalton@dsc.ufcg.edu.br

conformidade no contexto de software em evolução. A extração e análise de informações não-estruturada do sistema, geralmente descritas em linguagem natural, podem ajudar nesse processo, através de técnicas de Recuperação da Informação.

Este trabalho teve como objetivo desenvolver uma ferramenta de suporte à manutenção e evolução de um software. O IRUtils é uma ferramenta que extrai o vocabulário do software e o representa em um modelo de espaço vetorial, onde pode-se extrair métricas de similaridade. Tais métricas são essenciais no mapeamento de entidades de design em módulos arquiteturais. Além disso, este relatório de pesquisa complementa a ferramenta desenvolvida, descrevendo-a assim como as técnicas implementadas nela.

METODOLOGIA

Este projeto de iniciação científica está inserido no contexto do projeto Design Checker, financiado pela FINEP - Financiadora de Estudos e Projetos, e da tese do aluno de doutorado do curso de Ciência da Computação, Roberto Almeida Bittencourt. O trabalho foi realizado no Laboratório de Redes de Petri e Métodos Formais do Departamento de Sistemas e Computação da Universidade Federal de Campina Grande.

Neste trabalho, serão investigadas e implementadas técnicas de recuperação de informação para auxiliar o processo de manutenção da conformidade entre arquitetura e implementação de software em evolução.

Mais especificamente, a metodologia deste trabalho consistirá dos seguintes passos:

- Revisão bibliográfica do estado da arte sobre técnicas de recuperação de informação aplicadas em manutenção e evolução de software (concluído);
- Concepção e projeto de uma ferramenta de extração de informação de grafos de *design* de software e de criação de vocabulário associado às entidades do *design* (concluído);
- Implementação de medidas de similaridade entre entidades de *design* (concluído);
- Implementação de algoritmos de mapeamento de entidades de *design* para módulos arquiteturais pré-existentes (concluído);
- Implementação de algoritmo para calcular a coesão conceitual de módulos arquiteturais, não realizada por mudanças de planos no trabalho de doutorado;
- Implementação de algoritmos para recuperação de módulos arquiteturais a partir da agregação de entidades de *design* similares (concluído);
- Escrita de um artigo e sua submissão à Revista Eletrônica de Iniciação Científica (REIC) da Sociedade Brasileira de Computação para divulgação e *feedback*;
- Escrita do relatório da bolsa de iniciação científica, para avaliação (concluído).

O cronograma a seguir destaca os passos realizados durante este trabalho:

	Meses											
	2009					2010						
Atividade	08	09	10	11	12	01	02	03	04	05	06	07
Revisão bibliográfica do estado da arte sobre técnicas de RI em manutenção e evolução de software												
Concepção e projeto da ferramenta de extração de informação e criação de vocabulário												
Implementação de protótipo da ferramenta de extração de informação e criação de vocabulário												
Implementação de medidas de similaridad entre entidades de *design*												
Implementação de algoritmos de mapeamento de entidades de *design* para módulos arquiteturais pré-existentes												
Implementação de algoritmos para recuperação de módulos arquiteturais a partir da agregação de entidades de *design* similares												
Escrita do relatório da bolsa de iniciação científica												

Revisão Bibliográfica

Nesta seção estão relacionados os conceitos estudados neste trabalho sobre Verificação de Conformidade Arquitetural, Recuperação Arquitetural e Recuperação de Informação.

Verificação de Conformidade Arquitetural

Murphy et al. (1995, 2001) realizaram um dos trabalhos mais importantes sobre verificação de conformidade, utilizando os chamados modelos de reflexão. A técnica proposta parte de um modelo de alto nível proposto pelo engenheiro de software, de acordo com a compreensão do sistema. O engenheiro propõe o mapeamento das entidades de alto nível para entidades de baixo nível, estas extraídas do código-fonte através de engenharia reversa. O mapeamento é realizado a partir de expressões regulares, agrupando entidades de baixo nível em módulos de alto nível e extraindo as relações de baixo nível para relações de alto nível.

O modelo de reflexão, portanto, determina quais relações estão corretas de acordo com o modelo de alto nível do engenheiro. Em essência, um modelo de reflexão resume o modelo de baixo nível de um sistema de software do ponto de vista de um modelo de alto nível particular (MURPHY et al., 1995). Dessa forma, é possível modificar o modelo de alto nível para melhorar sua conformidade com a implementação; assim como também é possível alterar o mapeamento de forma a construir modelos de reflexão adicionais, a partir de diferentes pontos de vista do software.

Os modelos de reflexão então descritos demandam que o engenheiro de software conheça o mapeamento das entidades encontradas no código-fonte para entidades de alto nível. Mesmo sendo possível utilizar convenções baseadas em nomes de entidades e localização dos arquivos em diretórios, o processo de mapeamento demanda grande esforço manual por parte do engenheiro de software. O mapeamento com auxílio computacional foi proposto através de técnicas de agrupamento automático (CHRISTL et al., 2005) e não pretende substituir o processo manual, mas dar suporte ao usuário para alcançar um mapeamento correto de forma mais rápida.

Lindvall e Muthig (2008) sintetizam algumas das idéias mais importantes em verificação de conformidade arquitetural através de um processo composto por seis passos:

- capturar e modelar a arquitetura planejada, de acordo com um alto nível compreensível do sistema;
- extrair a arquitetura real a partir do código-fonte;
- definir o mapeamento dos componentes da arquitetura real para componentes da arquitetura planejada;
- comparar automaticamente a arquitetura real com a planejada, baseando-se no mapeamento realizado no passo anterior;
- analisar cada desvio para determinar se ele é crítico;
- planejar a remoção das violações consideradas críticas.

A visão dos autores é de um ambiente dinâmico, relacionando a arquitetura, regras e restrições arquiteturais e a implementação do sistema ao longo de todo o ciclo de vida, inclusive a evolução, do software.

Dessa forma, conformidade arquitetural pode ser definida como uma medida de em que grau uma arquitetura implementada no código-fonte de um software se conforma com a arquitetura planejada (KNODEL e POPESCU, 2007). Assim, um grau alto de conformidade significa que há poucas violações arquiteturais, enquanto um grau baixo significa que há muitas violações. Este projeto se interessa na conformidade entre uma visão estática estrutural (modular) de uma arquitetura e sua implementação (código-fonte).

No contexto de verificação estática, os trabalhos existentes abordam a conformidade nas relações entre componentes. Três casos podem ocorrer entre dois ou mais componentes arquiteturais planejados e componentes implementados equivalentes:

- Convergência: a relação entre dois componentes existe tanto na arquitetura planejada quanto na implementada, significando que a implementação obedece à arquitetura planejada;
- Divergência: a relação não existe na arquitetura planejada, mas foi implementada, indicando a não obediência à arquitetura planejada;
- Ausência: a relação foi planejada mas não foi implementada, indicando que relações planejadas não foram encontradas na implementação.

Recuperação Arquitetural

Para verificar a conformidade entre arquitetura planejada e implementada, é preciso extrair uma abstração do modelo de alto nível a partir da estrutura e do comportamento das entidades que compõem o software. Este processo de abstração é conhecido por recuperação arquitetural, que é um passo no processo de engenharia reversa (CHIKOFSKY e CROSS, 1990). Recuperação arquitetural é um problema complexo, pois é difícil relacionar conceitos existentes em código-fonte, dependentes de linguagem de programação, e conceitos de alto nível, mais adequados ao entendimento humano e ao modelo conceitual

do sistema. Além disso, uma arquitetura costuma ser documentada como um conjunto de diferentes visões, de acordo com algum padrão de documentação arquitetural (CLEMENTS et al., 2002). Portanto, diferentes técnicas de abstração devem ser usadas para recuperar visões arquiteturais diferentes.

A visão modular é utilizada para dividir um sistema em módulos com responsabilidades específicas e interfaces apropriadas para comunicação, e é a mais utilizada das visões arquiteturais. Armstrong e Trudeau (1998) descrevem e comparam ferramentas que extraem informação do código-fonte e abstraem-na em visões modulares de alto nível, permitindo modificações manuais nas visões através de interface com o usuário. Portanto, estas ferramentas permitem realizar recuperação arquitetural assistida pelo engenheiro de software, de forma automática ou semi-automática.

Em nosso trabalho, esperamos que o engenheiro seja capaz de reconhecer, modificar e confirmar uma visão modular a partir de visões extraídas do código-fonte, de acordo com o seu conhecimento técnico e o contexto no qual o software está inserido. As visões extraídas independem da técnica utilizada, através de expressões regulares ou localização em diretórios, por exemplo. Uma vez que a visão obtida é confirmada, pode-se checar automaticamente a conformidade do software com a arquitetura planejada.

Recuperação de Informação

Recuperação de Informação é a área de pesquisa que se preocupa com a estrutura, análise, organização, armazenamento, recuperação e busca de informação (SALTON e MCGILL, 1986). Uma das suas principais áreas de atuação é a busca de informações relevantes em documentos e páginas Web, ou seja, informação não estruturada e geralmente em linguagem natural, extremamente complexa de ser analisada. Dada a convenção de documentação do código por parte dos programadores, é percebido que as informações de documentação, tais como nomes de atributos, uma vez extraídas, são de grande importância para análise em diversas linhas de pesquisa.

Um dos primeiros trabalhos a utilizar nomes de entidades com fins de manutenção de software foi a técnica de agrupamento baseada em nomes de arquivos, de Anquetil e Lethbridge (1997). Nesta técnica, partes dos nomes dos arquivos (n-gramas) são extraídos, permitindo agrupar os arquivos que possuem mais n-gramas similares. Posteriormente, estes autores realizaram estudos comparativos entre várias técnicas de agrupamento, concluindo que atributos descritivos não formais (e.g.: nomes de arquivos) são tão poderosos quanto atributos formais de código-fonte (e.g.: dependências entre entidades) (ANQUETIL e LETHBRIDGE, 1999).

Linstead et al. (2009) realizaram um estudo com o vocabulário de milhares de projetos de código aberto em linguagem de programação Java. Foram coletados nomes de classes, interfaces, métodos e atributos, e organizados de modo a observar padrões e convenções na nomeação dessas estruturas. O trabalho teve o objetivo de enfatizar a importância das informações descritas em documentação para trabalhos futuros, incluindo compreensão de software, busca e mineração de dados.

Abebe et al. (2009), por sua vez, analisaram o vocabulário de softwares de grande porte durante sua evolução. Os autores concluíram que o vocabulário dos softwares cresciam de acordo com o crescimento do próprio software, ou seja, com o surgimento de novos identificadores para as entidades, eram incluidos também novos termos ao vocabulário. Além disso, concluíram que termos frequentes são associados com o domínio o qual o software está inserido, sugerindo que, a partir de análise simples é possível extrair conhecimento do sistema desenvolvido.

Diversos outros trabalhos têm usado recuperação de informação e, mais especificamente LSI como técnica para automatizar tarefas de manutenção de software. LSI, ou indexação por semântica latente, é uma técnica bastante popular na área que constrói um corpus a partir do vocabulário de um conjunto de documentos e atribui um vetor a cada documento, a partir de estatísticas de cada documento e do conjunto de documentos (DEERWESTER et al., 1990). Com o objetivo de agrupar artefatos de código-fonte que utilizam vocabulário similar, no trabalho de Kuhn et al. (2007) cada entidade (e.g.: método, classe) é vista como um documento, onde o vocabulário é extraído dos identificadores e/ou comentários englobados por dada entidade.

Ferramenta Desenvolvida

Nossa solução para auxiliar o processo de manutenção da conformidade entre arquitetura e implementação de software no contexto de evolução consiste em implementar uma ferramenta, o IRUtils, que utilize conceitos de recuperação de informação, incluindo LSI. Os tópicos a seguir contém os conceitos básicos para o entendimento da ferramenta desenvolvida neste trabalho.

IRUtils

IRUtils utiliza a representação em grafo do sistema para extrair seu vocabulário. Como representação do sistema de software a ser analisado, o IRUtils recebe como entrada um grafo de design de software,

onde cada vértice do grafo representa uma entidade de design, e as arestas representam as dependências estáticas entre as entidades.

IRUtils permite realizar o mapeamento de entidades de design para módulos arquiteturais pré-existentes e descobrir novos módulos arquiteturais a partir da agregação de entidades de design similares. A Figura 1 apresenta as etapas de obtenção do vocabulário e espaço vetorial do software; os conceitos citados serão aprofundados nas seções seguintes.

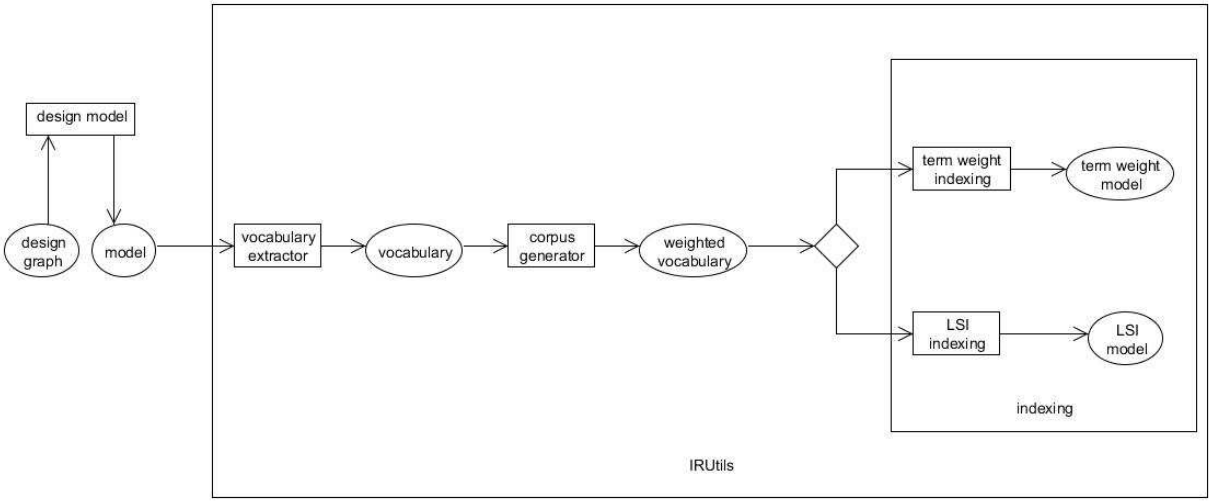

Figura 1. Visão Geral do IRUtils

- ***design model***: módulo responsável pela transformação do software, representado por um arquivo em formato padrão de intercâmbio para reengenharia de software, conhecido como GXL (HOLT et al., 2000). Este formato permite que o software seja representado como um grafo de entidades conectadas por dependências estáticas. O resultado obtido nessa etapa é uma representação do grafo em um modelo de *design* armazenado em memória;
- ***vocabulary extractor***: realiza a extração do vocabulário do modelo de *design*, resultando em uma lista de termos ocorridos em cada entidade;
- ***corpus generator***: responsável pela atribuição de pesos aos termos obtidos na etapa anterior, de acordo com a função escolhida pelo usuário: *term frequency* (tf), *inverse document frequency* (idf), ou tf-idf. Tem-se como resultado uma lista de tuplas para cada entidade, onde cada tupla representa um termo e seu determinado peso no documento;
- ***indexing***: módulo responsável pela representação do sistema em um modelo de espaço vetorial, assim como a representação de qualquer busca ou documento em seu vetor equivalente nesse espaço. A representação vetorial pode ser feita com as aproximações feitas pela técnica LSI. Além disso, este módulo também informa a similaridade entre dois vetores no espaço, através Das seguintes funções: Cosseno, Euclidean, Canberra, Bray-Curtis e Tanimoto.

Extração de Termos

A Figura 2 representa o processo de extração de termos, i.e. sequências de caracteres que podem ser incluídos no dicionário de um sistema de recuperação de informação, para posterior consulta, análise e recuperação. Os detalhes das etapas desse processo são descritas a seguir.

A etapa de **extração** consiste em recuperar informações úteis de uma coleção de documentos. Assim, dado um modelo de alto-nível proposto pelo usuário (e.g.: classe ou pacote), o IRUtils extrai os identificadores das entidades desse modelo. Cada entidade desse modelo é chamado de documento na área de recuperação da informação. Caso considerarmos um grafo hierárquico, em que entidades possuem subgrafos com entidades de baixo nível, serão incluídos os identificadores destas sub-entidades no vocabulário do sistema. O resultado desta etapa é uma lista de identificadores do sistema, organizados pelas entidades básicas do modelo proposto.

Dada uma sequência de caracteres e um documento definido, ***tokenização*** é a tarefa de separar essas sequências em pedaços (*tokens*), geralmente eliminando pontuações. As sequências são divididas sob o critério de documentação *camel case*, que consiste numa prática comum de diversas linguagens de programação em escrever palavras compostas, onde cada palavra é iniciada com letras maiúsculas e unidas sem espaços. O resultado desta etapa consiste em uma lista de palavras do sistema, ainda organizadas pelas entidades do modelo.

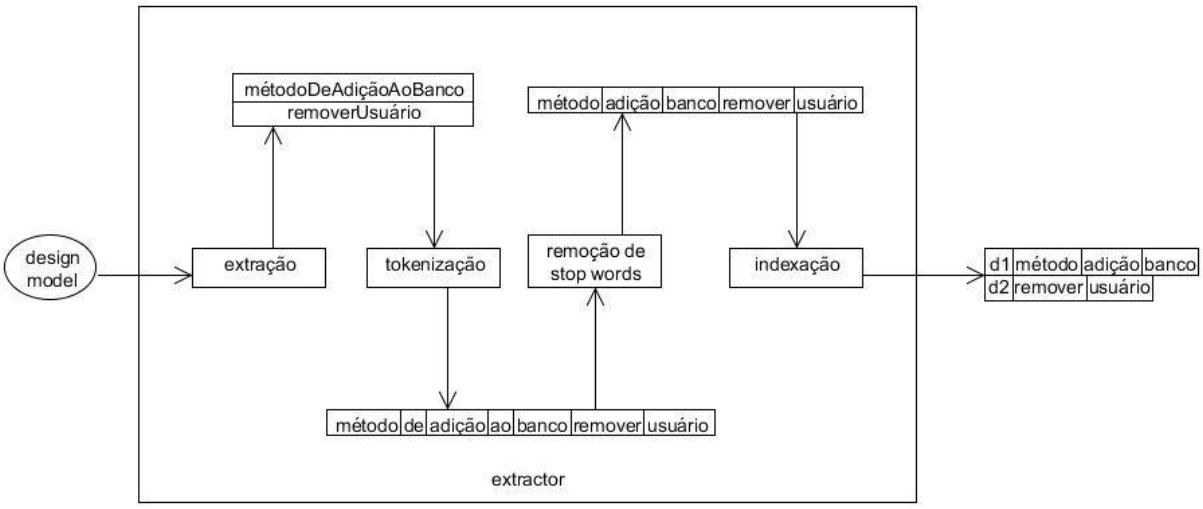

Figura 2. Processo de extração de termos do software

Após a etapa de *tokenização*, uma etapa adicional é realizada: **a remoção de *stop words***, i.e. palavras usadas frequentemente que têm pouco valor de busca. Por exemplo, conectivos como conjunções e preposições (e, para, de, com, entre outros). Lucene (LUCENE, 2010) é uma biblioteca em código aberto para consulta e *indexação* de textos, e foi utilizada para realizar a remoção dos *stop words*.

A última etapa de obtenção dos termos do vocabulário é chamada de ***indexação***, que consiste na organização dos termos de modo a tornar possível a busca destes no sistema. Para isso, é construída uma estrutura de mapa, onde para cada documento há uma lista de termos encontrados. Esta estrutura de dados, chamada de corpus, representa o vocabulário do sistema e serve de base para operação de manutenção de software baseadas no vocabulário extraído.

Determinação de Pesos a Termos (*Term Weighting*)

Até o momento, a busca de um termo no corpus do sistema informava apenas se o termo ocorria ou não em determinado documento. Um próximo passo determina se um documento tem mais importância na busca, dependendo da quantidade de ocorrências do termo e de uma função de valoração. Por exemplo, um documento cujo termo a ocorreu 10 vezes pode ser mais importante que um documento em que o mesmo termo ocorreu 3 vezes. Este passo é chamado de *term weighting*.

A abordagem mais simples dá ao termo o valor igual à quantidade de vezes que o mesmo ocorreu no documento. Esta função é chamada de *term frequency* e denotada tf(t,d), onde *t* e *d* denotam o termo e o documento, respectivamente. Uma abordagem posterior extrai a frequência relativa do termo, ou seja, a razão entre a quantidade de ocorrências em um determinado documento pela quantidade de ocorrências de termos em todo o documento. Esta nova função é chamada de *relative term frequency*.

As abordagens de contagem descritas consideram igualmente relevantes todos os termos durante uma busca. De fato, pode-se perceber que alguns termos tem pouca ou nenhuma relevância, ao ocorrer na maioria dos documentos. O trabalho de Linstead et al. (2009) mostra que este fenômeno acontece em softwares reais, ou seja, certos termos tendem a ocorrer em diversas partes do sistema, e sua busca acaba retornando muitos resultados. A função chamada *inverse document frequency*, denotada por idf(t), tende a possuir valor alto para termos raros, enquanto possui valores mais baixos quanto mais o termo é frequente. A função é descrita a seguir:

$$\mathrm{idf}_t = \log \frac{N}{\mathrm{df}_t}$$

, onde *N* denota a quantidade de documentos e *df(t)*, a quantidade de documentos em que o termo *t* ocorreu.

Por fim, pode-se combinar as abordagens *term frequency* e *inverse document frequency*, definindo assim uma nova função, chamada de tf-idf. A função tf-idf(t,d) atribui peso a um termo *t* em um documento *d*, da seguinte forma:

$$\text{tf-idf}_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t$$

Em outras palavras, a função tf-idf(t,d) tende a ter valores:

- mais altos quando o termo ocorre mais vezes em uma pequena quantidade de documentos;
- baixos quando o termo ocorre poucas vezes em um documento, ou ocorre em muitos documentos;
- mais baixos quando o termo ocorre em praticamente todos os documentos.

Com o objetivo de atribuir peso a um termo em um determinado documento, uma série de funções alternativas a *term frequency* e tf-idf podem ser consideradas. Manning et al. (2008) apresenta em seu trabalho algumas das principais.

Modelo de Espaço Vetorial (*Vector Space Indexing*)

Na seção anterior, foi desenvolvida a noção de que um documento ou uma busca (*query*) pode ser representado como um conjunto finito de termos. A representação de um conjunto de documentos em um espaço vetorial comum é conhecido como modelo de espaço vetorial (*vector space model*) e é fundamental para fornecer operaçõe de recuperação de informação, como classificação e agrupamento (*clustering*) de documentos.

Dessa forma, *V(d)* é o vetor derivado do documento *d*, e cada componente do vetor representa um termo do dicionário. Pode-se assumir que é usada a função tf-idf para a definição de pesos aos componentes do vetor V(d), porém a representação em modelo de espaço vetorial independe da função de pesos. O conjunto de documentos do sistema pode então ser vista como um conjunto de vetores em um espaço. Esta representação não considera a ordem dos termos, fazendo com que os documentos "José é melhor que Maria" e "Maria é melhor que José" sejam idênticos nessa representação.

***Indexação* por Semântica Latente (*LSI Indexing*)**

Indexação por Semântica Latente é um método em recuperação de informação que visa facilitar a busca de termos em um conjunto de documentos a partir de palavras similares, de modo parecido com as estratégias de busca em documentos e páginas Web. A motivação parte do princípio que o vocabulário de sistemas de busca de grande porte tendem a ser muito grandes, muitos desses termos ocorrendo em poucos documentos e, portanto, ocupando espaço desnecessário.

LSI consiste no processo de representar os documentos num espaço vetorial menor, i.e. com menos termos. Termos que ocorrem juntos com frequência podem então ser agrupados, pois assume-se que estes termos são semanticamente correlatos, podendo abranger casos em que termos diferentes possuem mesmo significado (sinonímia) e casos em que um termo pode ter vários significados (polisemia). Assim, a modificação no espaço vetorial faz com que cada documento seja representado de outra forma, tal como qualquer busca feita nessa nova representação.

A transformação do espaço vetorial é feita através de decomposição de matrizes e aproximações de baixo posto. Considere o espaço sendo representado por uma matriz MxN, onde cada linha representa um termo do vocabulário, e cada coluna representa um documento; esta matriz é chamada de matriz termo-por-documento. Então, pode-se dizer que o elemento da linha *i* e coluna *j* desta matriz, representa o peso do termo t_i no documento d_j. A função de determinação de pesos não importa nesse processo.

Seja uma matriz termo-por-documento A, há uma decomposição de valor singular de A da forma:

$A = USV^T$, onde *U* é uma matriz MxM cujas colunas são os auto-valores de AA^T, a matriz *S* é uma matriz MxN diagonal formada pelos auto-valores, ou valores singulares, de *A* em ordem decrescente. Por fim, *V* é uma matriz NxN cujas colunas são auto-valores de A^TA.

A decomposição da matriz termo-por-documento *A* é realizada pela ferramenta Colt (COLT, 2010). O próximo passo consiste em escolher um número inteiro positivo *k* e determinar a aproximação de posto *k* da matriz *A*, e é feita da seguinte forma:

- Dada uma matriz *A*, construir sua decomposição de valor singular; portanto, $A = USV^T$;
- Seja *r* o posto da matriz *S*;
- Derivar a matriz S_k da matriz *S*, substituindo por zero os *(r – k)* menores valores da diagonal principal de *S*;
- Derivar a matriz A_k, $A_k = US_kV$ como a aproximação de posto *k* da matriz *A*.

Aproximações também podem ser feitas nas matrizes *U* e *V*. Uma vez que os *(r – k)* valores da matriz A foram substituidos por zero, pode-se definir U_k e V_k como matrizes reduzidas de *U* e *V*, mantendo apenas as *k* primeiras colunas de *U* e *V*, respectivamente. A vantagem é de evitar a multiplicação de valores que não irão influir no resultado de $A_k = US_kV$, já que as *k* linhas e colunas de S_k são iguais a zero.

A escolha do número *k* deve ser suficiente para minimizar a diferença entre a aproximação de posto *k* e a matriz original. Através de demonstrações matemáticas, Manning et al. (2008) mostram que o efeito de

valores singulares pequenos no produto da matriz resultante A_k é pequeno e, portanto, a remoção destes valores na matriz *S* mantém a proximidade entre *A* e A_k. Também, segundo os autores, para dezenas de milhares de documentos, um valor de *k* em torno de poucas centenas pode aumentar a precisão sobre algumas buscas. Em nossa abordagem, o valor de aproximação *k* tem valor padrão igual a zero, permitindo ao usuário modificar este valor, de acordo com o vocabulário do sistema e a eficiência da aplicação.

Para seguintes consultas sobre o espaço vetorial produzido pela decomposição de valor singular, a matriz A_k é utilizada. Um vetor de busca, representada pelo vetor *q*, portanto, pode ser representado no espaço vetorial LSI através da transformação:

$$\vec{q}_k = S_k^{-1} U_k^T \vec{q}.$$

Assim como representamos o vetor de busca em um novo espaço, também é possível realizar a tranformação em um vetor de documento, de maneira análoga. Dessa forma, pode-se usar as funções de similaridade para determinar a semelhança entre um vetor de busca e um documento, ou entre dois documentos. Esta transformação permite que novos documentos sejam adicionados para a representação LSI, com a ressalva que a qualidade desta representação tende a degradar quanto mais documentos são adicionados; isto requere que seja computado um novo espaço, de modo a incluir os novos documentos.

Similaridade

A partir de um modelo de espaço vetorial, onde tantos os documentos, quanto qualquer conjunto de termos pode ser transformado em um vetor equivalente, pode ser medida e proximidade entre dois vetores através do cosseno entre eles. Documentos e/ou buscas correlatas, portanto, apresentarão maior similaridade nesta medida. A função de similaridade baseada no cosseno dos vetores é definida a seguir:

$$\mathrm{sim}(d_1, d_2) = \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{|\vec{V}(d_1)||\vec{V}(d_2)|}$$

, onde o numerador representa o produto vetorial dos vetores derivados dos documentos d_1 e d_2, enquanto o denominador representa o produto das normas euclidianas destes vetores.

Outra abordagem para definir a similaridade entre dois vetores de documentos utiliza a distância euclidiana entre estes vetores, e é definida a seguir:

$$|\vec{x} - \vec{y}| = \sqrt{\sum_{i=1}^{M} (x_i - y_i)^2}$$

, onde *M* representa o número de termos do dicionário.

Agrupamento e Mapeamento

Agrupamento, *clusterização* ou *clustering* é a técnica de organizar objetos em categorias de acordo com sua similaridade. Este tipo de atividade é bastante comum, principalmente em organização de arquivos e entidades de *design* em grupos coesos, e vem sendo intensificada devido ao número elevado de dados disponíveis em um sistema de software.

Dessa forma, um bom agrupamento deverá manter elevada semelhança entre entidades do mesmo grupo, enquanto deve manter baixa semelhança entre entidades de grupos diferentes. O critério de semelhança é subjetivo, depende tanto do contexto quanto da função que determinará a similaridade. A técnica de agrupamento pode ser considerada uma técnica que utiliza aprendizado não supervisionado, ou seja, o funcionamento independe da intervenção de um especialista que obtenha conhecimento sobre as entidades.

Anquetil e Lethbridge (1997) apresentaram uma técnica de agrupamento de entidades de *design* baseada em nomes de arquivos que, posteriormente e através de estudos comparativos, mostrou que atributos descritivos podem ser tão poderosos quanto atributos formais de código-fonte. Utilizando recuperação de informação, especialmente similaridade de documentos, espera-se que seja possível identificar relações fortes, a partir de análise semântica.

A partir do espaço vetorial semântico fornecido pelas técnicas de LSI, pode-se verificar a similaridade entre os vetores de documentos, independente da função de similaridade utilizada, agrupando aqueles que tem maiores valores nessa medida. Neste trabalho, utilizamos o agrupamento hierárquico aglomerativo, que parte de um conjunto de *clusters* unitários, realizando a união de pares de *clusters* com alta similaridade; esta união resulta numa estrutura de árvore, chamada de dendrograma. O resultado esperado é um

conjunto de grupos de entidades de alta semelhança semântica, e eventualmente um conjunto de entidades que não apresentam semelhança suficiente para serem adicionados em algum grupo; estas entidades são chamadas de entidades órfãs.

Independente da técnica utilizada para extração da arquitetura planejada, os modelos de reflexão tendem a ser desatualizados com a evolução do software, onde novas entidades de *design* podem ser adicionadas. Além disso, entidades existentes podem ser removidas, assim como suas relações, modificando a similaridade entre as demais entidades do sistema.

No intuito de atualizar um modelo de reflexão durante a evolução de um software, a técnica de mapeamento (*mapping*) é uma opção para direcionar entidades de *design* a módulos arquiteturais já existentes, de acordo com uma função de atração que determina o quão forte é a relação entre a entidade e o módulo. Entidades órfãs resultadas de atividades de agrupamento também podem ser mapeadas para um módulo existente.

No contexto de recuperação de informação, cada entidade a ser mapeada pode ser representada como um vetor. Assim, é possível verificar a similaridade com os módulos pré-existentes, descritos como vetores de documentos. Similaridades altas, portanto, indicam que a entidade deve pertencer ao módulo. O resultado esperado é um novo modelo de reflexão, incluindo entidades nos módulos de mais alta similaridade.

Ao contrário do agrupamento, as técnicas de mapeamento podem ser assistidas e direcionadas por um especialista. Uma vez que ela indica o fator de semelhança entre a entidade de *design* e os módulos existentes no modelo de reflexão atual, cabe ao engenheiro, de acordo com seu conhecimento prévio no sistema, escolher qual módulo arquitetural a entidade deve ser adicionada. Este mapeamento, com intervenção do usuário, é chamado de mapeamento semi-automático.

Ambas as técnicas de agrupamento e mapeamento ajudam a tornar os modelos de reflexão atuais e coesos, permitindo que estes modelos possam ser atualizados durante a evolução do software. De um ponto de vista mais específico, conforme o sistema e seu vocabulário cresce, novos termos surgem e elaboram novas relações, semânticas inclusive, entre as entidades, permitindo uma nova orgaização de como o software está sendo implementado.

CONCLUSÕES

Atualmente, recuperação de informação é uma área em ascenção, devido à grande quantidade de informação gerada nos mais diversos sistemas de software. Métodos e técnicas vêm surgindo no objetivo de interpretar e analisar esse volume de informação de forma rápida, com menor intervenção de um especialista. LSI é uma técnica de grande importância na área, pois permite que a informação seja interpretada pela sua similaridade em um contexto específico, ou seja, pelo seu significado.

Utilizar o vocabulário do software para extrair informações e, especialmente, para facilitar sua manutenção e evolução, é uma linha de pesquisa recente, a qual depende de boas práticas no desenvolvimento. Uma ferramenta que explora interativamente este vocabulário, constantemente modificado durante a evolução do software, no intuito de manter a coesão entre os módulos arquiteturais, é portanto de grande auxílio na manutenção da integridade conceitual no contexto de software em evolução.

Por fim, o conhecimento adquirido em recuperação de informação é bastante abrangente, gerando um conhecimento essencial para qualquer sistema moderno de busca, em áreas que ultrapassam a simples busca de uma palavra num conjunto de documentos. Por exemplo, no Laboratório de Redes de Petri e Métodos Formais do Departamento de Sistemas onde foi desenvolvido esse trabalho, existem hoje duas pesquisas adicionais que utilizam conceitos de recuperação de informação: um estudo estatístico do vocabulário de softwares escritos em linguagem de programação Java e a investigação de técnicas para compreensão de programas e localização de requisitos escritos em linguagem natural.

Espera-se que o IRUtils seja uma ferramenta de base para trabalhos futuros que utilizem o vocabulário de software para sua compreensão, manutenção e análise. Espera-se também quem o IRUtils incorpore outras funções de atribuição de pesos a termos, permitindo estudos comparativos entre os diversos métodos propostos.

AGRADECIMENTOS

Ao CNPq pela bolsa de Iniciação Científica e a Dalton Serey, Roberto Bittencourt e Raquel Guimarães pela dedicação e orientação empregados para que o projeto fosse realizado.

REFERÊNCIAS BIBLIOGRÁFICAS

ABEBE et al., 2009. Abebe, S. L., Haiduc, S., Marcus, A., Tonella, P., and Antoniol, G. 2009. **Analyzing the Evolution of the Source Code Vocabulary**. In Proceedings of the 2009 European Conference on Software Maintenance and Reengineering (March 24 - 27, 2009). CSMR. IEEE Computer Society, Washington, DC, 189-198. DOI=http://dx.doi.org/10.1109/CSMR.2009.61

ANQUETIL e LETHBRIDGE, 1997. Anquetil, N. and Lethbridge, T. 1997. **File clustering using naming conventions for legacy systems**. In Proceedings of the 1997 Conference of the Centre For Advanced Studies on Collaborative Research (Toronto, Ontario, Canada, November 10 - 13, 1997). J. H. Johnson, Ed. IBM Centre for Advanced Studies Conference. IBM Press.

ANQUETIL e LETHBRIDGE, 1999. Anquetil, N., Fourrier, C., and Lethbridge, T. C. 1999. **Experiments with Clustering as a Software Remodularization Method**. In Proceedings of the Sixth Working Conference on Reverse Engineering (October 06 - 08, 1999). WCRE. IEEE Computer Society, Washington, DC, 235.

CHIKOFSKY e CROSS, 1990. Chikofsky, E. J. and Cross II, J. H. (1990). **Reverse Engineering and Design Recovery: a taxonomy**. IEEE Software, 7(1):13–17.

CHRISTL et al., 2005. Christl, A., Koschke, R., and Storey, M. 2005. **Equipping the Reflexion Method with Automated Clustering**. In Proceedings of the 12th Working Conference on Reverse Engineering (November 07 - 11, 2005). WCRE. IEEE Computer Society, Washington, DC, 89-98.

CLEMENTS et al., 2002. Clements, P., Garlan, D., Bass, L., Stafford, J., Nord, R., Ivers, J., and Little, R. (2002). **Documenting Software Architectures: Views and Beyond**. Pearson Education.

DEERWESTER et al., 1990. Deerwester, S., Dumais, S., Furnas, G., Landauer, T., and Hashman, R. (1990). **Indexing by latent semantic indexing**. Journal of the American Society for Information Science, 41(6).

GARLAN e PERRY, 1995. Garlan, D. and Perry, D.E. (1995). **Introduction to the Special Issue on Software Architecture**. IEEE Transactions on Software Engineering, 21(4):269–274.

HOLT et al., 2000. Holt, R. C., Winter, A., and Schürr, A. (2000). **GXL: Toward a Standard Exchange Format**. In Proceedings of the 7th Working Conference on Reverse Engineering, pages 162–171. WCRE, 23–25 November 2000. Washington, DC, IEEE Computer Society.

KNODEL e POPESCU, 2007. Knodel, J. and Popescu, D. 2007. **A Comparison of Static Architecture Compliance Checking Approaches**. In Proceedings of the Sixth Working IEEE/IFIP Conference on Software Architecture (January 06 - 09, 2007). WICSA. IEEE Computer Society, Washington, DC, 12.

KUHN et al., 2007. Kuhn, A., Ducasse, S., and Gírba, T. 2007. **Semantic clustering: Identifying topics in source code**. Inf. Softw. Technol. 49, 3 (Mar. 2007), 230-243.

LINDVALL e MUTHIG, 2008. Lindvall, M. and Muthig, D. 2008. **Bridging the Software Architecture Gap**. Computer 41, 6 (Jun. 2008), 98-101.

LUCENE, 2010. **Apache Lucene**, 2010. http://lucene.apache.org/. Visitado em 21 de fevereiro de 2010.

MANNING et al., 2008. Manning, C., Raghavan, P., and Schtze, H. 2008. **Introduction to Information Retrieval**. Cambridge University Press.

MARCUS e POSHYVANYK, 2005. Marcus, A. and Poshyvanyk, D. (2005). **The Conceptual Cohesion of Classes**. In Proceedings of the 21st IEEE international Conference on Software Maintenance (September 25 - 30, 2005). ICSM. IEEE Computer Society, Washington, DC, 133-142.

MURPHY et al., 1995. Murphy, G. C., Notkin, D. and Sullivan, K. (1995). **Software Reflexion Models: Bridging the Gap between Source and High-Level Models**. ACM SIGSOFT Software Engineering Notes, 20(4):18–28.

MURPHY et al., 2001. Murphy, G. C., Notkin, D., and Sullivan, K. J. 2001. **Software Reflexion Models: Bridging the Gap between Design and Implementation**. IEEE Trans. Softw. Eng. 27, 4 (Apr. 2001), 364-380.

PARNAS, 1994. Parnas, D. L. (1994). **Software Aging**. In Proceedings of the 16th International Conference on Software Engineering, pages 279–287. ICSE, Sorrento, Italy, 16–21 May 1994. Los Alamitos, CA, IEEE Computer Society Press.

PERRY e WOLF, 1992. Perry, D. E. and Wolf, A. L. 1992. **Foundations for the study of software architecture**. SIGSOFT Softw. Eng. Notes 17, 4 (Oct. 1992), 40-52.

SALTON e MCGILL, 1986. Salton, G. and McGill, M. J. 1986. **Introduction to Modern Information Retrieval**. McGraw-Hill, Inc.