Uma Técnica de Indexação de Artefatos de Software Baseada em Dados Semi-Estruturados

Talles Brito, Thiago Ribeiro, Hugo Nóbrega, Glêdson Elias

COMPOSE – Component Oriented Software Engineering Group
Departamento de Informática – Universidade Federal da Paraíba
{talles,thiago,hugo}@compose.ufpb.br, gledson@di.ufpb.br

Resumo. *Sistemas de busca têm sido propostos na indústria e academia a fim de facilitar a busca de componentes de software. No entanto, diversos sistemas propostos adotam técnicas de indexação inadequadas e ineficientes. Neste contexto, este artigo apresenta uma nova técnica de indexação de artefatos de software que adota um modelo de representação baseado em dados semi-estruturados. Como principal contribuição, a técnica proposta permite eficientemente recuperar artefatos de software com base em suas propriedades sintáticas e semânticas.*

Abstract. *Search systems have been proposed by industry and academy in order to facilitate software component search. Despite that, several proposed systems adopt inadequate and inefficient indexing techniques. In such a context, this paper presents an asset indexing technique that adopts a representation model based on semi-structured data. As the main contribution, the proposed technique enables to efficiently retrieve software assets taking into account their syntactic and semantic properties.*

1. Introdução

Para tornar as práticas de reuso efetivas, os desenvolvedores devem estar aptos a buscar e selecionar componentes de software com facilidade [1]. Apesar do surgimento de novas técnicas e processos de reuso de software que se destacam por permitir a reutilização de maneira sistemática e mais previsível, ainda se faz necessária a adoção de uma infra-estrutura de ferramentas e técnicas para o armazenamento, indexação e busca dos inúmeros artefatos de software e produtos gerados ao longo dos processos de desenvolvimento baseados em componentes. Neste sentido, é importante ressaltar o desenvolvimento de técnicas de recuperação de informação que possam auxiliar os desenvolvedores na busca e seleção de componentes apropriados as suas necessidades.

Como conseqüência, vários sistemas de busca e recuperação de componentes de software têm sido propostos. Por definição [2], um sistema de recuperação de informações pode ser visto como sendo composto por dois componentes. O primeiro é o engenho de indexação, que através de uma técnica de indexação permite gerar índices. O segundo é o engenho de consulta, o qual cria planos de execução de consultas a fim de recuperar objetos que possam satisfazer as necessidades dos usuários.

Apesar das relevantes contribuições [3][4][5][6], tais sistemas de busca adotam diferentes técnicas de indexação e de acesso aos índices. No entanto, as técnicas de

indexação adotadas apresentam desvantagens em diferentes aspectos. Em relação à representação dos componentes, alguns sistemas provêm uma representação restrita a um pequeno domínio [3]. Além disso, outros sistemas adotam representações não realmente adequadas, tal como o código fonte dos componentes [4], não levando em consideração propriedades semânticas dos componentes.

Alguns sistemas de repositório de componentes adotam técnicas de indexação que automaticamente geram informações de representação dos componentes, mas, mesmo assim, os componentes são descobertos através de uma busca exaustiva em toda a coleção de informações [5]. Em geral, tais sistemas não levam em consideração que a técnica de indexação deve prover uma recuperação eficiente. Por exemplo, em trabalhos recentes que adotam abordagens baseadas em ontologias [6], a inferência sobre a coleção de conceitos é bastante custosa [7]. Vale ressaltar que sistemas que adotam métodos de recuperação clássicos, baseados na indexação de termos, se sobressaem em eficiência quando comparado aos sistemas que empregam mecanismos de inferência.

Por fim, como uma conseqüência da adoção de métodos de representação e técnicas de indexação inadequadas, os sistemas podem tratar um conjunto restrito de necessidades de informação. Por outro lado, os usuários também podem ter dificuldades ao converter suas necessidades de informação em consultas, quando estas são expressas em função de conceitos compartilhados pertencentes a um domínio abrangente.

Neste contexto, a fim de minimizar os problemas apresentados, este artigo apresenta uma nova técnica de indexação de componentes de software que adota um modelo de representação baseado em dados semi-estruturados. O modelo de representação adotado permite expressar não somente informações sintáticas textuais, mas também características semânticas associadas. Mais especificamente, a técnica de indexação proposta adota o modelo de representação X-ARM (*XML-based Asset Representation Model*) [8] que possibilita representar diferentes artefatos produzidos em quaisquer processos de DBC. Ao adotar uma técnica de indexação de dados semi-estruturados tem-se como objetivo minimizar os problemas de eficiência citados. Já a adoção de um modelo de representação específico para descrição de artefatos possibilita que um apropriado conjunto de conceitos seja empregado na construção de consultas.

O restante deste artigo está organizado da seguinte forma. Na Seção 2, o modelo de representação X-ARM é apresentado. Em seguida, na Seção 3, a técnica de indexação proposta é apresentada, bem como, algumas considerações sobre o processamento de consultas. Com o objetivo de avaliar a eficiência da técnica proposta, a Seção 4 apresenta e discute alguns resultados experimentais. Na seção 5, a técnica proposta é comparada com outras técnicas de indexação adotadas em sistemas de busca de componentes correlatos. Por fim, a Seção 6 apresenta algumas considerações finais.

2. Modelo de Representação dos Artefatos

Em um processo de DBC, diversos tipos de artefatos podem ser gerados, e, conseqüentemente, armazenados em um repositório a fim de potencializar o reuso. Apesar disso, no repositório, geralmente, os artefatos armazenados não são autodescritivos, e, além disso, podem ter diferentes formatos: desde documentações textuais e diagramas até código fonte. Como apontado por [9], características dos componentes obtidas diretamente a partir de dados contidos no próprio código fonte ou

documentação podem não expressar fielmente as funcionalidades e a semântica dos componentes. Logo, a fim de tratar esta dificuldade, segundo Frakes [10], os modelos de representação para descrição de componentes devem prover um conjunto de informações que possibilitem a indexação dos componentes por sistemas de busca.

Sendo assim, neste trabalho é utilizado o modelo de representação X-ARM [8], que adota um modelo de dados semi-estruturados baseado em XML. O X-ARM permite descrever não apenas produtos finais, ou seja, componentes implementados, mas também diversos tipos de artefatos produzidos durante o processo de desenvolvimento, provendo a semântica necessária para representar informações sobre a evolução, a certificação, o esquema de negócio, a visibilidade, o processo de desenvolvimento adotado e a classificação dos artefatos. Além disso, possibilita descrever especificações e implementações de componentes, interfaces, eventos e exceções. Por fim, também possibilita especificar processos de desenvolvimento, modelos de negócio, de certificação e de componentes, bem como, áreas de aplicação dos artefatos.

Como as descrições X-ARM são escritas sob a forma de documentos XML, temos que relações estruturais dos elementos XML provêem semântica aos valores textuais. Desta forma, a indexação de tais descrições X-ARM usando técnicas de indexação baseada em dados semi-estruturados torna possível selecionar os artefatos que satisfazem critérios informados através de restrições textuais e estruturais de acordo com as necessidades de informação do usuário. Vale ressaltar que, como o modelo de dados XML é semi-estruturado, técnicas de recuperação de informação baseadas somente em termos textuais não são eficientes quando adotadas para dados semi-estruturados. Logo, a adoção de documentos XML implica em desafios relacionados ao projeto de uma nova técnica de indexação que mantenha um nível de precisão apropriado, bem como, adote novos tipos de índices que mantenham eficiência no processamento de consultas que podem conter diversas restrições textuais e estruturais.

Desta forma, torna-se fundamental o projeto de técnicas de recuperação de informação capazes de tratar adequadamente a indexação de dados expressos em XML. Com o objetivo de tratar tais problemas, neste trabalho é proposta uma nova técnica para indexação de dados semi-estruturados. Atualmente, esta técnica já é empregada em um sistema de busca [11] que manipula artefatos de software representados em X-ARM. Na próxima seção, a técnica de indexação proposta é apresentada.

3. A Técnica de Indexação Baseada em Dados Semi-Estruturados

É importante ressaltar que os principais desafios envolvidos no desenvolvimento de técnicas de indexação para dados semi-estruturados estão relacionados com a indexação das propriedades estruturais dos documentos e não apenas informações textuais. Desta forma, manter tais propriedades possibilita aos usuários construir suas consultas levando em consideração relações tanto estruturais quanto textuais. Outro desafio diz respeito à eficiência do processamento das consultas e a precisão dos artefatos retornados.

Diversos trabalhos têm sido propostos para lidar com tais problemas. No entanto, apesar das relevantes contribuições, algumas técnicas propostas não empregam indexação exata. Como definido por Weigel [2], existem técnicas de indexação *exatas* e *inexatas*. Para uma determinada consulta, um índice exato retorna todos os dados

relevantes, e somente estes. Por outro lado, índices inexatos podem recuperar itens irrelevantes, afetando assim a precisão dos resultados da consulta com falsos positivos.

As técnicas de indexação classificadas como *baseadas em caminhos* tiram vantagem da redução de precisão de modo a reduzir o tamanho dos arquivos de índice. Isto é realizado por meio da seleção de caminhos (um dado caminho representa uma navegação na estrutura do documento XML) que satisfazem um padrão definido para a técnica, representando assim apenas um subconjunto de todos os caminhos que podem ser extraídos dos dados de entrada. Como conseqüência, os usuários sofrem com os efeitos da redução de precisão, pois itens que não satisfazem suas necessidades de informação podem ser retornados pelo sistema de busca.

A maior desvantagem das técnicas de indexação baseadas em caminhos, como o *Dataguide* [2], é que, geralmente, essas técnicas não indexam consultas com ramificação. Se um caminho é visto como uma seqüência de navegação, então um caminho simples é um caminho correspondente a uma única seqüência. Em contraste, uma consulta com ramificação pode ter várias seqüências (e, portanto, bifurcações), correspondendo a uma árvore de caminhos de navegação. A técnica *Dataguide* não indexa nenhuma consulta com ramificação. Desta forma, a adoção desta técnica de indexação tem como vantagem a redução dos tamanhos dos índices e a desvantagem da perda de precisão.

Além disso, outras técnicas propostas [12][13], denominadas índices de *interseção (join) estrutural*, apesar de serem exatas, precisam empregar um tipo de algoritmo de consulta comumente categorizado como *interseção (merge-join) estrutural*. Nas técnicas de interseção estrutural, os nomes dos elementos XML, seus atributos e valores textuais são indexados. As consultas são decompostas em várias partes, sendo uma consulta executada para cada parte. Cada consulta retorna um conjunto de resultados, que são unidos pelo processador de consultas usando um algoritmo de interseção estrutural, de modo a obter o resultado final.

De forma diferente das técnicas de indexação baseadas em caminhos, as técnicas baseadas em índices de interseção estrutural podem processar consultas com ramificação uma vez que qualquer consulta com ramificação (que pode ser vista como uma árvore) pode ser decomposta em partes menores, que, por sua vez, podem ser individualmente processadas e posteriormente combinadas. Apesar dessa vantagem, operações de interseção estrutural têm alto custo de processamento, e, como demonstrado experimentalmente em [14], a melhora de desempenho no processamento de consultas depende da redução da quantidade de operações de interseção estrutural.

Neste sentido, a fim de minimizar os problemas anteriormente citados, este artigo propõe uma nova técnica de indexação de dados semi-estruturados baseada em caminhos. A técnica proposta foi concebida para manipular uma determinada classe de consultas com ramificações, que pode ser utilizada em variados tipos de sistemas de busca, incluindo aqueles capazes de recuperar artefatos descritos em documentos X-ARM, como detalhado na próxima seção. Além disso, é importante ressaltar que a técnica proposta foi desenvolvida com o objetivo de evitar o uso de operações de interseção estrutural, de modo a obter um mecanismo eficiente de processamento de consultas, bem como, não retornar falsos positivos na execução da classe de consultas com ramificação suportada, e assim, alcançar um melhor nível de precisão.

3.1. Tipo de Consulta

O problema tratado será primeiramente descrito em um nível mais alto de abstração. Suponha que uma coleção de objetos precisa ser indexada, na qual cada objeto é semanticamente descrito usando um modelo de representação. Cada objeto descrito pode então ser armazenado em uma base de dados juntamente com sua descrição. Por exemplo, todos os artefatos de software armazenados em um repositório também devem ter suas descrições armazenadas (no formato X-ARM, por exemplo). Além disso, considere que é fornecida uma interface de busca onde tais objetos podem ser encontrados de acordo com as necessidades de informação do usuário. Tal interface de busca permite que usuários procurem por artefatos de acordo com restrições de consulta como, por exemplo: nome do artefato, funcionalidades ou certificação.

Como ilustração, a Figura 1 mostra um documento XML que descreve um componente de nome *Calculadora* que possui duas operações: *negativo* e *incremento*. A primeira operação possui um parâmetro de nome *operando*. Por sua vez, a segunda possui um parâmetro de nome *valor*. É importante ressaltar que, embora o documento da Figura 1 seja suficiente para desenvolver a linha de raciocínio deste artigo, este documento não representa uma descrição X-ARM real, mas uma descrição em versão resumida que facilita a compreensão do leitor.

```
<asset name="Calculadora">
    <operation name="negativo">
        <parameter name="operando"/>
    </operation>
    <operation name="incremento">
        <parameter name="valor"/>
    </operation>
</asset>
```

Figura 1. Exemplo de descrição de um componente de software

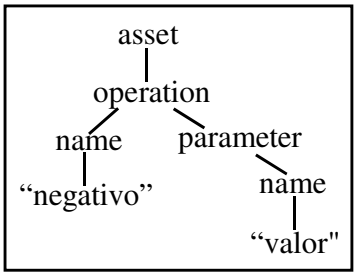

Figura 2. Árvore representando uma necessidade de informação

Suponha que um usuário deseja encontrar um componente de software de acordo com a seguinte necessidade de informação: "*Encontrar um componente que possua uma operação de nome negativo e esta operação possua um parâmetro chamado valor"*. Esta necessidade de informação pode ser representada por meio de um conjunto de caminhos de navegação, especificamente, uma árvore, como ilustrada na Figura 2. Este conjunto de caminhos representa uma consulta com ramificações. Um índice gerado com a técnica *Dataguide* não pode resolver consultas com ramificações (como previamente citado na Seção 3). Neste contexto, essencialmente, o problema que é tratado neste trabalho é: como desenvolver um índice baseado em caminhos que suporte o tratamento de consultas com ramificação de modo que falsos positivos não ocorram?

Como uma direção para a solução, neste trabalho, ao contrário de outros índices baseados em caminhos, a técnica de indexação proposta não seleciona apenas caminhos simples para a indexação. Neste trabalho é definido um padrão para a seleção de caminhos considerando os seguintes requisitos: **i)** Devem ser selecionados caminhos com o objetivo de possibilitar o tratamento de consultas com ramificação; **ii)** Estes caminhos com ramificações podem ser extraídos de um amplo conjunto de diferentes bases de dados XML e devem possibilitar o tratamento de consultas com ramificação

em um amplo conjunto de sistemas de busca; **iii)** Os caminhos selecionados devem permitir o tratamento de diversos conjuntos de necessidades de informação dos usuários.

A fim de satisfazer estes requisitos, a interface provida por um engenho de indexação que suporte a técnica de indexação proposta neste trabalho deve aceitar como entrada qualquer consulta com ramificação que seja do tipo definido pela expressão regular $(/elemento([atributo="texto"])^{0..1})^{+}$. Esta notação de expressão de consultas é semelhante à definida com o *XPath* [15].

Por exemplo, a consulta com ramificação expressa na linguagem *XPath* como */asset/operation[@name="negativo"]/parameter[@name="valor"]* satisfaz o padrão de expressão regular definido. Além disso, esta expressão *XPath* corresponde à árvore mostrada na Figura 2. Perceba que */asset/operation[@name* refere-se à restrição estrutural *asset-operation-name*, enquanto que *[@name="negativo"]*, refere-se à restrição textual *...-name-"negativo"*. Neste sentido, a técnica de indexação deve processar expressões que estejam de acordo com o padrão definido, e, assim, retornar os documentos indexados que satisfazem as relações estruturais e textuais impostas.

3.2. Técnica de Indexação

Nesta seção, serão inicialmente apresentados os fundamentos necessários ao entendimento da técnica proposta. Em seguida, serão apresentadas a técnica de indexação e algumas considerações sobre a implementação da mesma.

3.2.1. Grafo de Dados

Intuitivamente, um documento XML pode ser visto como uma árvore na qual os nós representam elementos ou atributos e as relações pai-filho na árvore representam as relações existentes entre cada elemento e seus elementos/atributos filhos no documento XML. Neste sentido, como apontado por Weigel [2], comumente, ao considerarmos uma coleção de documentos XML, todas as árvores correspondentes aos documentos desta coleção podem ser concatenadas em um grafo definido como o *grafo de dados*.

Por exemplo, a Figura 3 mostra dois documentos XML, cada um descreve um componente de software hipotético. O primeiro, descrito pelo *Documento 1* é o componente *Calculadora1*, que possui a operação *negativo* cujo único parâmetro chama-se *valor*. O segundo, descrito pelo *Documento 2* é o componente *Calculadora2*, que possui não apenas a operação *negativo* como também a operação *incremento*, ambos recebendo parâmetros de nome *valor*. Estes dois documentos podem ser vistos no grafo de dados ilustrado na Figura 4.

Documento 1:

```
<asset name="Calculadora1">
        <operation name="negativo">
            <parameter name="valor"/>
        </operation>
</asset>
```

Documento 2:

```
<asset name="Calculadora2">
        <operation name="negativo">
            <parameter name="valor"/>
        </operation>
        <operation name="incremento">
            <parameter name="valor"/>
        </operation>
</asset>
```

Figura 3. Documentos XML descrevendo artefatos

Um grafo de dados é definido como um grafo direcionado *G = (V, E, rótulos, valores, tipos, RAIZ)*, tal que *V* é um conjunto de vértices, no qual a função *valores* mapeia cada vértice para diferentes informações que são associadas em cada um dos vértices de *G*. Além disso, *E* é um conjunto de arestas e cada nome de aresta é mapeado através da função *rótulos*. Por sua vez, *RAIZ* é o vértice raiz de *G*. Cada vértice está associado a um elemento ou atributo dos documentos XML da fonte de dados, neste sentido, a função *tipos* indica se um dado vértice é associado com um elemento ou atributo XML. Por fim, é importante ressaltar que as direções das arestas seguem a direção do relacionamento pai-filho entre elementos e atributos. Veja que atributos nunca têm filhos, logo, sempre estão relacionados a vértices folha de *G*.

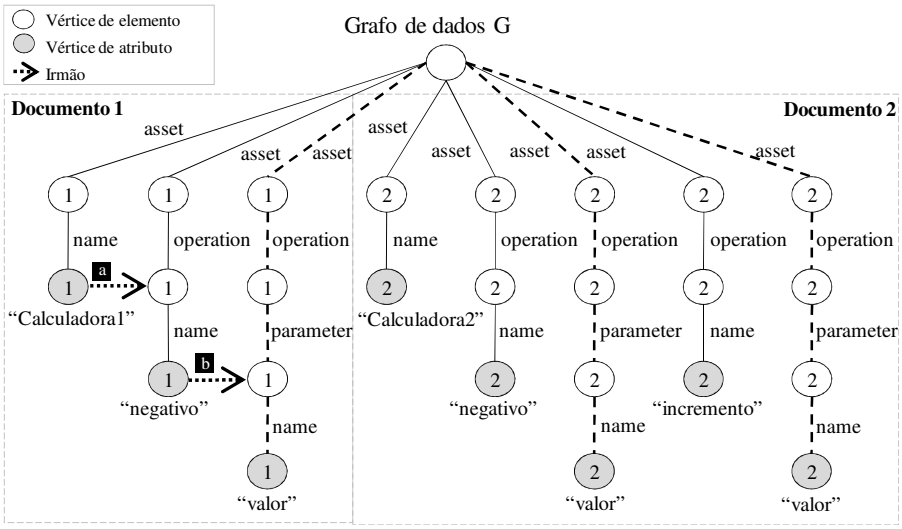

Figura 4. Exemplo de grafo de dados

Mais especificamente, considere o *Documento 1* da Figura 3. Este documento contém a seguinte relação no sentido de navegação pai→filho a partir do elemento raiz: *asset−operation−parameter−name−"valor"*. Então, este caminho deve estar contido no grafo de dados mostrado na Figura 4. Cada elemento ou atributo de um caminho é associado a uma aresta nomeada por meio da função *rótulo*. Por exemplo, as arestas *operation, parameter* e *name*. Além disso, cada vértice possui uma aresta incidente (no sentido pai→filho). Por meio da função *valores*, no caso de arestas de elementos (*asset*, *operation* e *parameter*), identificadores de documentos são mapeados e armazenados nos vértices nos quais as arestas incidem. Na Figura 4, os identificadores *1* e *2* representam os dois documentos usados para gerar o grafo. Por outro lado, no caso das arestas de atributos (*name*), além dos identificadores de documentos, é armazenado o valor textual do atributo no vértice incidente. O caminho de navegação mencionado anteriormente ocorre simultaneamente nos documentos *1* e *2*, conforme pode ser observado na Figura 4. Portanto, um mesmo caminho de navegação é replicado no grafo de dados quando vários documentos descrevem a mesma informação.

3.2.2. Grafo de Índice

Os caminhos de navegação estão relacionados com as expressões de consulta (por exemplo, expressões *XPath*). Dada uma consulta *XPath*, tal como

/asset/operation/parameter[@name="valor"], esta pode ser relacionada ao caminho de navegação *asset−operation−parameter−name−"valor"*. Este caminho de navegação pode ser encontrado diversas vezes no grafo de dados, navegando pelos vértices a partir da raiz até o vértice contendo o último nome incluído no caminho.

Para o caminho de navegação mencionado anteriormente, ao navegar pelo grafo de dados mostrado na Figura 4, a ocorrência replicada do caminho no grafo de dados (destacada com linhas pontilhadas) resulta na recuperação de dois identificadores de documentos, representados especificamente pelo conjunto {1, 2}. Vale ressaltar que procurar ocorrências de caminhos no grafo de dados é um procedimento caro do ponto de vista de processamento. Portanto, técnicas de indexação baseada em dados semi-estruturados devem evitar procurar caminhos nos documentos da fonte de dados.

Para processar as consultas de maneira eficiente, técnicas de indexação baseadas em dados semi-estruturados devem gerar um índice contendo um resumo dos documentos que compõem a fonte de dados [2]. A forma como um índice baseado em caminhos é criado pode ser definida através de regras. No momento da criação de tais resumos das informações presentes no grafo de dados, essas regras definem uma transformação do grafo de dados em um novo grafo, chamado de *grafo do índice*. Um processo de indexação pode ser definido como uma função $I: G \rightarrow G'$ que mapeia um grafo de dados G para um grafo de índice G' de acordo com as regras definidas por I.

3.2.3. A Função de Transformação Proposta

Primeiramente, a fim de definir formalmente a proposta da função de transformação de grafo (I) é necessário incluir uma nova informação no grafo de dados, que é mapeada usando a função *valores* do grafo *G*. Deste ponto em diante, para cada vértice do tipo atributo, a função *valores* mapeia o vértice para o seu respectivo identificador de documento, valor textual do atributo e um conjunto de vértices irmãos.

Por exemplo, no documento *1* ilustrado na Figura 3, considere o elemento *asset*. Neste caso, o elemento *operation* é um filho direto do elemento *asset*, bem como o atributo *name*. Sendo assim, é definida outra relação entre elementos e atributos (não é considerada tal relação entre um atributo e outro atributo) denominada relação de *irmãos*. Como *name* e *operation* são filhos do mesmo pai, então *name* e *operation* são irmãos. As ocorrências deste tipo de relação no documento *1* estão destacadas na Figura 4 com setas. Desta forma, a função *valores* mapeia o vértice no qual a aresta *name* incide para o respectivo identificador do documento, e, em adição, também mapeia o vértice para a lista de vértices irmãos. No exemplo, a função *valores* mapeia o vértice no qual a aresta *name* incide para o vértice no qual a aresta *operation* incide.

Para indexar caminhos com ramificação associados às expressões *XPath* que satisfazem a expressão regular $(/elemento([atributo="texto"])^{0..1})^{+}$, este artigo propõe uma função de transformação $I: G \rightarrow G'$ definida pelas seguintes regras: **i)** Todo caminho indexado a partir da fonte de dados é representado uma única vez no índice sem replicação, sendo que cada vértice contém a lista de documentos onde ocorre o respectivo caminho; **ii)** Todo vértice do grafo de dados do tipo atributo gera uma sub-árvore para cada um dos seus irmãos, sendo que a aresta raiz desta sub-árvore tem o nome da aresta que incide no vértice irmão; **iii)** A sub-árvore gerada corresponde aos

caminhos com ramificação, e, assim, a lista de documentos de cada vértice desta sub-árvore contém somente documentos nos quais ocorrem o caminho com ramificação.

Por exemplo, veja que as duas relações de irmão do documento *1* na Figura 4 estão destacadas, e, a propósito de explanação, cada relação é identificada como relação *a* e *b***.** A relação *b* existe entre o vértice de atributo *name-"negativo"* e o vértice no qual a aresta *parameter* incide. Assim, este último vértice e sua sub-árvore devem ser indexados. O vértice raiz da sub-árvore está destacado na Figura 5 com uma linha dupla incidente e identificada com o rótulo *b***.** Analogamente, o irmão identificado com a relação *a* tem uma sub-árvore relacionada que está destacada com o rótulo *a* na Figura 5.

Observe que a adoção da transformação proposta não requer decompor a expressão de consulta */asset/operation[@name="negativo"]/parameter/[@name="valor"]* em expressões mais simples. Para processar tal consulta, o caminho com ramificação *asset−operation−name−"negativo"−parameter−name−"valor"* (destacado com linhas pontilhadas na Figura 5) pode ser acessado diretamente para recuperar a respectiva lista de identificadores de documentos. Como o caminho com ramificação é individualmente indexado, a transformação proposta assegura que não será retornado qualquer falso positivo, caracterizando assim uma técnica de indexação exata.

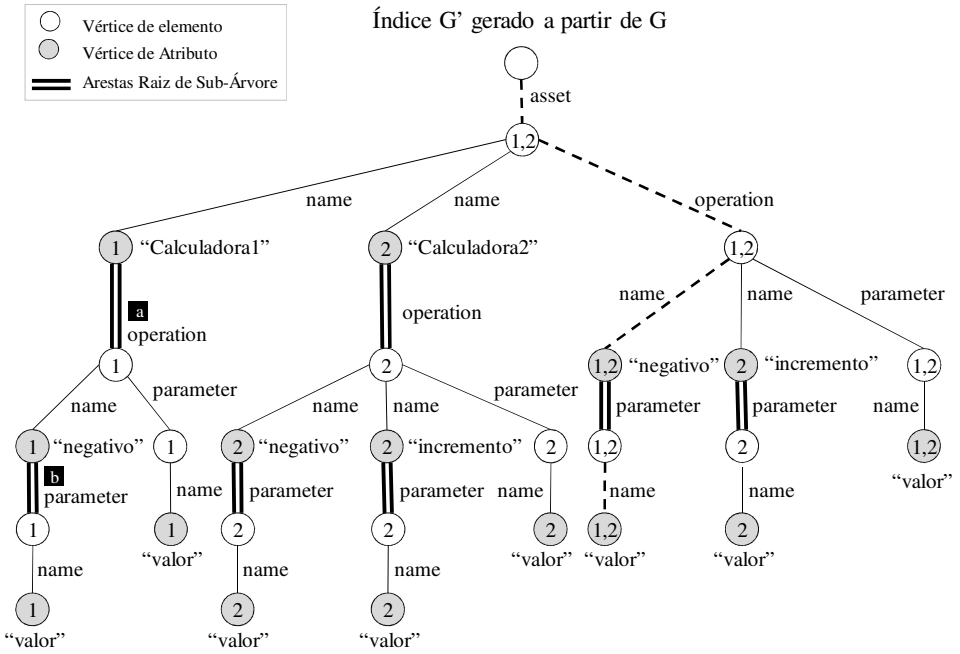

Figura 5. Índice G' gerado a partir de G

3.2.4. Implementação da Técnica Proposta

A partir da técnica proposta, um engenho de indexação protótipo foi desenvolvido na linguagem Java. Neste protótipo, todos os caminhos a partir do vértice raiz do grafo de índice em direção às folhas são armazenados. Cada caminho armazenado possui um ponteiro para a lista de identificadores de documentos contidos no vértice mais profundo do caminho. Estas informações são armazenadas em memória secundária usando uma estrutura de dados conhecida como índice invertido [16], mapeando cada caminho do

grafo do índice para um conjunto de identificadores de documentos. Sendo assim, no processamento de consultas, o caminho representado pela consulta é localizado no índice invertido. Se o caminho é encontrado, os identificadores de documentos associados ao caminho são retornados.

3.3. Decomposição de Consultas

De acordo com Manning [16], ao utilizar técnicas de recuperação de informação, o usuário pode buscar por uma classe de objetos que contém apenas um percentual do total de atributos de todas as instâncias da classe de objetos. Portanto, o usuário pode encontrar objetos similares ao que inicialmente estava procurando. Adotar uma abordagem de recuperação de informação é importante porque, em geral, antes do usuário iniciar as atividades de busca, ele tem apenas uma noção parcial e imprecisa do que está procurando ou do problema que está tentando resolver. Portanto, técnicas de recuperação devem permitir o controle de similaridade dos resultados a serem obtidos.

Neste sentido, a técnica de indexação proposta possibilita que consultas com um nível controlado de precisão sejam processadas. Para tal, um processador de consultas deve prover meios para decompor consultas mais complexas com operadores de controle de precisão em expressões que podem ser encontradas no índice. Mais especificamente, o controle de precisão é definido em função de operadores lógicos de disjunção (OR) e conjunção (AND). Por exemplo, suponha que o usuário está procurando por interfaces que possuam uma operação denominada "*soma*" contendo um parâmetro denominado "*a*" do tipo "*Inteiro*". Uma expressão de consulta que representa esta necessidade de informação é mostrada na Figura 6.

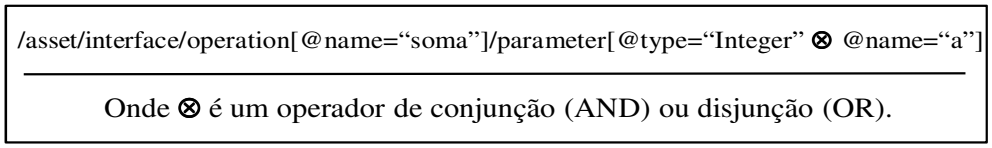

Figura 6. Controle de similaridade através de operadores

É importante ressaltar que o usuário pode especificar o grau de precisão dos resultados, definindo explicitamente o operador ⊗ como sendo AND ou OR, bem como, não indicando o operador de forma que o engenho de consulta retorne itens com diversos graus de similaridade. No caso do usuário não definir explicitamente os operadores de precisão, primeiramente, é gerada uma expressão de consulta mais restrita que utiliza o operador AND. Posteriormente, é gerada uma segunda expressão de consulta, sendo esta menos restrita que utiliza o operador OR. Para concluir o processamento, o resultado da execução de cada uma das duas consultas é concatenado em uma única lista, que então é retornada ao usuário. Na lista resultante, os itens mais restritos (mais similares) aparecem no topo da lista, enquanto que os itens menos restritos (menos similares) aparecem na base da lista.

A técnica de indexação proposta não suporta consultas que adotam operadores lógicos. No entanto, consultas com operadores lógicos podem ser decompostas em consultas suportadas pela técnica de indexação que não possuem tais operadores, e, posteriormente, um processamento adicional do engenho de busca pode combinar os resultados e retornar os objetos que satisfazem a consulta original.

Por exemplo, considere expressão da Figura 6. Considerando o operador ⊗ como sendo o operador lógico AND, a expressão de consulta final é mostrada Figura 7. Para processar tal expressão, ela é decomposta em duas expressões mais simples que também são mostradas na Figura 7. As duas expressões são individualmente consultadas no índice, e, para cada expressão, é retornada uma lista de documentos satisfazendo a respectiva consulta. Obtidas as duas listas, uma operação de interseção é executada entre as duas listas de forma que apenas artefatos que satisfaçam as duas expressões sejam simultaneamente selecionados.

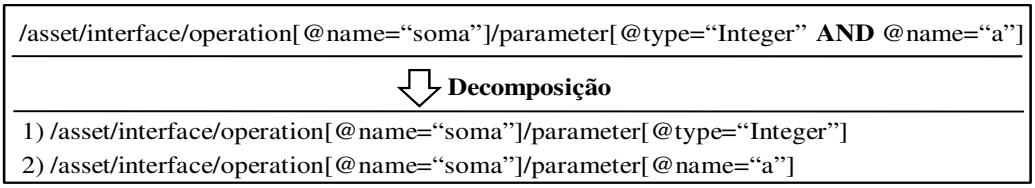

Figura 7. Decomposição de expressão de consulta

4. Avaliação Experimental

Para avaliar a técnica proposta foram realizados alguns experimentos. O objetivo dos experimentos é medir a eficiência da técnica de indexação proposta e compará-la à técnica de indexação baseada em interseção estrutural adotada pela ferramenta *eXist* [13]. Como citado anteriormente, técnicas de interseção estrutural indexam elementos/atributos ao invés de caminhos.

A amostra de documentos de descrição, utilizada nos experimentos, foi derivada a partir do X-ARM. As descrições representam especificações de interface geradas aleatoriamente, apresentando, no entanto, uma exaustiva quantidade de operações e parâmetros por interface para que se comporte como uma amostra de pior caso.

Foram definidas consultas de profundidade baixa, média e alta. Quanto maior a profundidade da consulta, maior a quantidade de nomes de elementos/atributos e maior a quantidade de ramificações. Para cada tipo de consulta (alta, baixa, média) foram definidas três consultas diferentes (I, II, e III) para avaliação. As mesmas consultas também foram executadas na ferramenta *eXist*. Os resultados estão listados na Tabela 1.

Tabela 1. Tempos de consulta em milissegundos

Profundidade da consulta	Técnica proposta			eXist		
	I	II	III	I	II	III
Baixa	1, 456	0, 176	0, 155	70,526	102,871	63,548
Média	0, 301	0, 190	0,159	175,984	61,290	70,826
Alta	0,358	0,152	0,146	83,928	84,831	83,815

Como pode ser observado na Tabela 1, os tempos de consulta da técnica proposta são consideravelmente menores que os tempos de consulta da ferramenta *eXist*. Isto se dá pelo fato que, na técnica proposta, a execução de uma consulta com ramificação implica em um simples acesso ao índice. Por outro lado, na ferramenta *eXist*, uma consulta com ramificação executa várias etapas de interseção estrutural para

obter o resultado final da consulta. Além disso, nas duas técnicas, não existe uma relação direta entre a profundidade da consulta e o tempo de execução, pois o tempo de execução é também dependente da recuperação da lista de identificadores que satisfazem a consulta, que pode variar em tamanho em diferentes consultas.

Vale ressaltar que, utilizando operadores lógicos, o tempo de processamento das consultas aumenta devido à decomposição das mesmas. Apesar disso, na técnica proposta, os tempos de consulta são significativamente menores (não apresentados neste artigo por limitações de espaço) que os tempos obtidos com a técnica de interseção estrutural adotada pela ferramenta *eXist*. Observa-se que, apesar da decomposição das várias consultas e das interseções necessárias para obter o resultado final, o tempo de execução total de uma consulta com operadores lógicos apresenta-se baixo em relação ao *eXist* devido aos menores tempos de acesso das consultas sem operadores lógicos.

5. Trabalhos Relacionados

Prieto-Diaz [3] propõe um mecanismo de classificação de componentes baseado em facetas. Uma faceta pode ser vista como uma propriedade pré-definida do componente que deve aceitar valores pré-definidos. Internamente, no repositório, os possíveis valores de cada faceta são relacionados através de um grafo conceitual que permite relacionar valores sinônimos ou conceitualmente semelhantes. A partir disto, é possível localizar componentes que se aproximam conceitualmente do componente pretendido. Entretanto, este tipo de classificação é ideal somente para repositórios que armazenam um conjunto homogêneo de componentes. Conforme novos componentes são armazenados eles devem ser classificados em função dos valores pré-definidos, sendo um método adequado apenas para componentes de um domínio restrito. Em contrapartida, a técnica proposta neste artigo pode ser utilizada em um repositório que mantém artefatos de qualquer domínio. Tal benefício é possível porque a técnica de indexação proposta não impõe restrições sobre valores textuais que podem ser descritos.

Em outra proposta, Girardi [5] apresenta um sistema para reuso de componentes de software baseado na indexação de descrições de artefatos utilizando linguagem natural. A descrição de cada artefato é expressa através de uma sentença imperativa. No processo de indexação, através da análise sintática e semântica das sentenças imperativas, de acordo com a presença de certos termos nas descrições, as mesmas são associadas a diferentes casos semânticos (por exemplo, a descrição do propósito de uso do componente ou a ação que um componente executa). Como saída do processo de análise semântica, a técnica de indexação gera uma estrutura que segue um modelo de representação próprio. Consultas dos usuários também são sentenças imperativas e são mapeadas no mesmo modelo de representação. Baseado em uma métrica de similaridade entre as representações da consulta e dos componentes, o sistema recupera componentes que são adequados em vários graus de similaridade. Do ponto de vista de representação de informações, um problema evidente é a dificuldade para descrever diversas propriedades dos artefatos utilizando sentenças imperativas. Já do ponto de vista da técnica de indexação, o sistema também tem um problema relacionado com a execução de consultas. Para cada consulta, esta é comparada com todo o conjunto de componentes indexados, tornando-se assim uma abordagem ineficiente. Em contraste, a técnica de indexação proposta adota índices invertidos, que são estruturas de dados reconhecidamente eficientes para recuperação de informações [16].

Ao contrário de propostas que adotam a indexação de código fonte, tal como [4], este artigo propõe uma técnica de indexação para metadados de artefatos representados com o X-ARM. Como apontado por Marrek [9], indexação de código fonte é bastante complicada e nem sempre traz os resultados esperados. É difícil distinguir quais partes do código representam funcionalidades. Além disso, os identificadores escolhidos pelos programadores para representar um mesmo conceito podem ser diferentes.

Por fim, Sivashanmugam [6] propõe uma abordagem na qual Web Services são semanticamente descritos com base em ontologias compartilhadas. Em [6], as consultas são especificadas pelos usuários através de modelos (grafos conceituais) construídos a partir dos conceitos das ontologias compartilhadas. Tal modelo é fornecido como entrada para um algoritmo que, através de inferência, recupera serviços semanticamente similares ao mesmo. Do ponto de vista do usuário, esta abordagem apresenta algumas dificuldades, pois requer que os usuários conheçam e entendam os conceitos de uma ontologia compartilhada que pode descrever conceitos de um domínio abrangente.

Por outro lado, a técnica de indexação proposta neste artigo adota um modelo de representação que semanticamente mantém propriedades de diversos tipos de artefatos gerados em processos de DBC, não requerendo que os usuários conheçam e entendam conceitos expressos em ontologias compartilhadas. Do ponto de vista de eficiência, como apontado por Ding [7], executar inferência sobre uma ampla coleção de conceitos pode ser computacionalmente caro. A técnica de indexação proposta e avaliada neste artigo não executa procedimentos de inferência já que adota técnicas de indexação de dados semi-estruturados, possibilitando uma eficiente descoberta de artefatos.

6. Considerações Finais

O X-ARM é um modelo de representação de artefatos que adota dados semi-estruturados e possibilita descrever um conjunto de propriedades semânticas de artefatos gerados em processos de DBC. No projeto inicial de um sistema de busca baseado em descrições X-ARM, optou-se por selecionar uma técnica de indexação de dados semi-estruturados já existente. No entanto, não foi obtido sucesso nesta fase inicial uma vez que as técnicas encontradas, em geral, sofrem de problemas de precisão e desempenho. Logo, optou-se pelo projeto de uma nova técnica de indexação.

A técnica de indexação proposta não tem problemas de precisão na indexação de caminhos com ramificação, de forma que as diversas relações estruturais que definem a semântica dos artefatos X-ARM não são perdidas após o processo de indexação. Além disso, a técnica proposta é eficiente do ponto de vista de processamento, fator que não é priorizado em diversos trabalhos relacionados, mas que na prática é um requisito fundamental para permitir a utilização adequada de um sistema de busca. Neste sentido, a principal contribuição deste artigo é a apresentação e avaliação da nova técnica de indexação de artefatos baseada em dados semi-estruturados e com suporte a consultas com ramificação. A técnica proposta provê um mecanismo de indexação de artefatos de software que mantém propriedades semânticas dos componentes indexados, e, mesmo assim, consegue ter excelentes resultados de desempenho. Vale ressaltar que a técnica proposta pode ser empregada em quaisquer sistemas de repositório de componentes que adotem modelos de representação baseados em dados semi-estruturados, e não apenas aqueles que adotam o modelo X-ARM.

Apesar dos excelentes resultados obtidos, é importante mencionar que a indexação de caminhos com ramificação gera um problema relacionado com a quantidade de informação indexada. Considerando que o número de caminhos que pode ser extraído da fonte de dados é grande, os índices podem se tornar muito maiores que a própria fonte de dados. Desta forma, como trabalho futuro, a técnica proposta ainda precisa ser melhorada para reduzir os requisitos de armazenamento, mas sem impor perda de precisão. Uma vez concluída esta melhoria, a técnica de indexação deverá ser integrada em um sistema de repositório, provendo a recuperação eficiente de artefatos, e, assim, potencializando o reuso em larga escala de uma ampla quantidade de artefatos.

7. Agradecimentos

Este trabalho foi apoiado [em parte] pelo Instituto Nacional de Ciência e Tecnologia para Engenharia de Software (INES)[1], financiado pelo CNPq processo 573964/2008-4.

8. Referências

[1] Frakes, W.; Pole, T. (1994) "An Empirical Study of Representation Methods for Reusable Software Components". IEEE Transactions on Software Engineering.

[2] Weigel, F. (2002) "A Survey of Indexing Techniques for Semistructured Documents". Project Thesis, Institute for Computer Science, University of Munich.

[3] Prieto-Diaz, R. (1991) "Implementing Faceted Classification for Software Reuse". Communications of ACM, Vol. 34, May.

[4] Garcia, V. *et al*. (2005) "Especificação, Projeto e Implementação de uma Arquitetura para um Engenho de Busca de Componentes". 5o. Workshop de Desenvolvimento Baseado em Componentes, Juiz de Fora - MG.

[5] Girardi, M.; Ibrahim, B. (1994) "Automatic Indexing of Software Artifacts". 3rd International Conference on Software Reuse.

[6] Sivashanmugam, K.; Verma, K.; Sheth, A.; Miller, J. (2003) "Adding Semantics to Web Services Standards". International Conference on Web Services.

[7] Ding, L. *et al*. (2004) "Swoogle: A Semantic Web Search and Metadata Engine". 13th ACM Conference on Information and Knowledge Management.

[8] Schuenck, M. (2006) "X-ARM: Um Modelo de Representação de Artefatos de Software". Dissertação de Mestrado, DIMAp-UFRN, Natal-RN.

[9] Maarek, S.; Berry, D.; Kaiser, G. (1991) "An Information Retrieval Approach for Automatically Constructing Software Libraries". IEEE Transactions on Software Engineering, Vol. 17.

[10] Frakes, W.; Kang, K. (2005) "Software Reuse Research: Status and Future". IEEE Transactions on Software Engineering, Vol. 31, Nº 7, July.

[11] Brito, T.; Nóbrega, H.; Ribeiro, T.; Elias, G. (2009) "A Search Service for Software Components based on a Semi-Structured Data Representation Model". 6th International Conference on Information Technology, Las Vegas.

[12] Li, Q.; Moon, B. (2001) "Indexing and Querying XML Data for Regular Path Expressions". 27th International Conference on Very Large Data Bases.

[13] Meier, W. (2002) "eXist: An Open Source Native XML Database". NODe 2002 Web and Database-Related Workshops on Web, Web-Services, and Database Systems, October.

[14] Kaushik, R. *et al*. (2004) "On the Integration of Structure Indexes and Inverted Lists". ACM SIGMOD International Conference on Management of Data, June, Paris.

[15] W3C. (1999) "XML Path Language (XPath)". http://www.w3.org/TR/xpath.

[16] Manning, C.; Raghavan, P.; Schütze, H. (2008) "Introduction to Information Retrieval". Cambridge University Press.

[1] www.ines.org.br