UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

REPRESENTAÇÃO GRÁFICA DE DOCUMENTOS XML

TRABALHO DE GRADUAÇÃO

Matheus Koehler Zanella

Santa Maria, RS, Brasil

2011

REPRESENTAÇÃO GRÁFICA DE DOCUMENTOS XML

por

Matheus Koehler Zanella

Trabalho de Graduação apresentado ao Curso de Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de
Bacharel em Ciência da Computação

Orientador: Profª Drª Deise de Brum Saccol

Trabalho de Graduação Nº 335
Santa Maria, RS, Brasil

2011

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

REPRESENTAÇÃO GRÁFICA DE DOCUMENTOS XML

elaborado por
Matheus Koehler Zanella

como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA:

Prof^{a} Dr^{a} Deise de Brum Saccol
(Presidente/Orientador)

Prof^{a} Dr^{a} Juliana Kaizer Vizzoto (UFSM)

Prof Dr Eduardo Kessler Piveta (UFSM)

Santa Maria, 16 de dezembro de 2011.

RESUMO

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

REPRESENTAÇÃO GRÁFICA DE DOCUMENTOS XML
Autor: Matheus Koehler Zanella
Orientador: Profª Drª Deise de Brum Saccol
Local e data da defesa: Santa Maria, 16 de dezembro de 2011.

A linguagem XML é muito utilizada para armazenamento e troca de informações. Muitos são os fatores que levam a essa grande utilização, tais como: é uma linguagem estruturada em *tags*, o que facilita seu entendimento e realizações de consultas mais consistentes; possui mecanismos para garantir uma boa estruturação do documento e que o mesmo contenha as informações necessárias para ser válido; apresenta facilidade de compressão devido à natureza repetitiva das tags da estrutura de dados, entre outros fatores. Entretanto, apesar de XML ser uma linguagem bem estruturada, devido ao tamanho e ao número de tags que um documento pode ter, se torna complicado o entendimento de como é a estrutura do mesmo. Outro fator é a carência de ferramentas que possibilitem a visualização de arquivos XML em um formato gráfico. Assim, o objetivo deste trabalho é implementar uma ferramenta que possibilite a representação e visualização gráfica de um documento XML. Para isso, a estrutura do documento foi traduzida para uma linguagem de representação de grafos chamada DOT. Feita esta tradução, foi utilizado a ferramenta *GraphViz*, que desenhou o grafo formado pela estrutura do arquivo XML.

Palavras-chave: XML, DOM, *GraphViz*, DOT, Java.

ABSTRACT

Undergraduate Final Work
Undergraduate Program in Computer Science
Federal University of Santa Maria

GRAPHICAL REPRESENTATION OF XML DOCUMENTS
Author: Matheus Koehler Zanella
Advisor: Profª Drª Deise de Brum Saccol

XML is widely used for storing and exchanging information. There are many factors that led to this widespread use, such as: it is a structured tag-based language, which facilitates its understanding and allows more consistent queries; it has mechanisms to ensure a suitable document structure and to guarantee the necessary information to be valid; it presents good compression rates due to the tags' repetitive nature, among other factors. However, despite the XML is a well-structured language due to the size and number of tags that a document can have, it becomes difficult to understand the document structure. The objective of this work was to implement a tool to generate e visualize the graphical representation of an XML document. For this, the document structure was translated to a graph representation language, named DOT. Once this translation was done, the *GraphViz* toll was used to draw the graph composed by the structure of the XML file.

Keywords: XML, DOM , *GraphViz*, DOT, Java.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

DOM	Document Object Model
DTD	Document Type Definition
EBNF	Extended Backus–Naur Form
GML	Generalized Markup Language
HTML	HyperText Markup Language
ISO	International Organization for Standardization
SGML	Standard Generalized Markup Language
XHTML	eXtensible Hypertext Markup Language
XML	Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

A história da linguagem XML tem seu início no começo da década de 70 com a criação, pela IBM, da linguagem GML. A GML surgiu da necessidade que a empresa tinha para armazenar grandes quantidades de informações de temas diversos. Em 1986, a linguagem passou a ser adotada pela ISO e passou a se chamar SGML. Apesar do nome, a SGML não é uma linguagem em si, mas sim uma linguagem para a especificação de outras linguagens. Seu propósito era criar vocabulários que poderiam ser usados para marcar documentos com *tags* estruturais.

No fim dos anos 80, utilizando a SGML, foi criada a HTML, uma linguagem bastante simples que se tornou muito popular pelo seu uso na internet. Entretanto, quando falamos em tráfego e armazenamento de informações, a HTML não é uma boa opção, pois, apesar de ser simples, ela é restrita a um conjunto limitado de *tags*, enquanto a SGML é muito complexa para o uso em geral. Assim, em 1996, começa o desenvolvimento da linguagem XML que iria preencher a lacuna entre as linguagens SGML e HTML, isto é, criar uma linguagem mais poderosa que a HTML e mais simples de usar que a SGML (RAY. 2003).

Na linguagem XML, a criação de tags fica a critério do usuário, portanto não existe nenhum controle de quais tags poderão ser criadas no documento. Para que este controle fosse feito, foram, então, criadas a linguagem DTD e, posteriormente, a linguagem XML Schema. Para facilitar a manipulação de arquivos XML foram criadas implementações independentes de plataformas, dentre elas o DOM, que contêm diversos métodos para o acesso das informações contidas no arquivo.

Atualmente, pelas suas características e pelas ferramentas que dão suporte à linguagem, a XML está sendo muito utilizada no tráfego e armazenamento de dados. Portanto, a intenção deste trabalho é implementar uma ferramenta que permita a visualização gráfica de arquivos XML, a fim de facilitar o entendimento de como o documento está estruturado.

1.1 Justificativa

Documentos XML são amplamente utilizados para troca e/ou armazenamento de informações. Muitas vezes esses documentos podem ter tamanhos muito grandes e serem formados por muitas *tags*. Por exemplo, uma grande empresa pode fornecer através de um documento XML todas as informações de produtos que estão em estoque. Esta situação pode comprometer o entendimento do documento, inclusive para usuários que detêm o conhecimento da linguagem, além de demandar uma enorme quantidade de tempo. Outro fator é a carência de ferramentas que possibilitem a visualização de arquivos XML em um formato gráfico. A utilização de alguma ferramenta que crie uma visualização gráfica do documento pode facilitar muito este entendimento, principalmente para usuários que não conhecem a linguagem XML.

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo geral deste trabalho consiste em implementar uma ferramenta que permita a visualização gráfica de um documento XML. Ao final, espera-se um *software* funcional que irá traduzir a estrutura deste documento XML para uma linguagem de representação de grafos chamada DOT. Feita esta tradução, será utilizada uma API em Java da ferramenta *GraphViz* para gerar a visualização do grafo na ferramenta.

1.2.2 Objetivos Específicos

- Estudar a linguagem XML e formas de estruturar um documento em XML
- Estudar DOM para a manipulação de documentos XML.
- Estudar a linguagem DOT.
- Estudar o *software GraphViz*.
- Definir a proposta de visualização gráfica do documento.
- Programar a ferramenta.
- Realizar testes na ferramenta.

1.3 Organização do Texto

De modo a estruturar o texto, o Capítulo 2 apresenta uma revisão de literatura sobre conceitos envolvendo XML. As linguagens DTD e XML Schema para validação de documentos XML são explicadas com exemplos. A seguir, é feito um estudo sobre DOM, o qual será utilizado para manipulação de documentos XML. Por último, um estudo sobre a linguagem de representação de grafos DOT e o *software GraphViz*, para geração de grafos, se faz necessário.

O Capítulo 3 contém a proposta da ferramenta que irá gerar a visualização gráfica do documento XML, explicando sua finalidade. A abordagem adotada para simulação desta visualização é apresentada com exemplos.

O Capítulo 4 explica como a ferramenta foi desenvolvida, com uma breve descrição da implementação. Figuras mostram todas as funcionalidades presentes na interface e explica-se como é a utilização do *software* através de exemplos.

Por fim, o Capítulo 5 encerra o trabalho com as considerações finais e sugestões de melhorias para trabalhos futuros.

2 REVISÃO DE LITERATURA

Este capítulo tem por objetivo a apresentação de algumas tecnologias, tais com a linguagem XML, linguagens para validação de documentos XML, DOM, DOT e o *software GraphViz*. Após, será feita uma breve análise de alguns *softwares* de visualização gráfica de documentos XML.

2.1 XML

Extensible Markup Language (XML) é uma linguagem de marcação de documentos, sendo derivada da *Standard Generalized Markup Language* (SGML) (W3C, 2011). XML é baseado em elementos, onde um elemento é um par de *tags* que indicam o início e o fim de cada elemento. Toda *tag* de início deve ter sua respectiva *tag* de fim. Um texto está no contexto de um elemento se o texto estiver entre a *tag* de início e a *tag* de fim deste elemento

Na figura 2.1, temos um exemplo de um documento XML:

```
<aluno>
        <nome>
                <primeiro_nome>Matheus  </primeiro_nome>
                <sobrenome>Zanella</sobrenome>
        </nome>
        <cidade>Santa  Maria</cidade>
</aluno>
```

Figura 2.1 - Documento alunos.xml.

A figura 2.1 está representando um dado aluno que possui *nome* e *cidade* como sub-elementos pertencentes a ele. O elemento *nome*, por sua vez, também possui dois sub-elementos: *primeiro_nome* e *sobrenome*.

A grande vantagem de armazenar dados desta maneira é que para programas é muito mais fácil identificar e trabalhar com as informações, pois elas estão bem estruturadas e

separadas por *tags*. Trabalhar com um documento que não contém uma boa estrutura, além de dificultar para os programadores, pode resultar em uma má interpretação por parte do programa, pois ele pode não saber onde uma informação começa e acaba por falta de padrões na estrutura do documento. A linguagem XML também apresenta outras vantagens, tais como: número ilimitado de tags, facilitando a criação de sintaxes próprias; suporte ao padrão Unicode, possibilitando a utilização de um grande número de símbolos; linguagem simples, facilitando o entendimento tanto para humanos quanto para computadores, entre outras vantagens relativas à sua estruturação (McGRATH, 1999).

Por outro lado, a sintaxe da XML é redundante ou torna-se grande em relação a representações de dados semelhantes, isto é, não faz sentido utilizar toda uma estrutura de *tags* para dados que irão se repetir um grande número de vezes no documento. Neste caso, utilizar o XML pode afetar a eficiência no armazenamento, transmissão e processamento de informações. A utilização de uma formatação mais simples seria mais adequada. Como exemplo, podemos pensar numa estação meteorológica, onde informações sobre temperatura, pressão e ventos são captadas por sensores e salvas a todo o momento. Utilizar o XML nesta situação poderia aumentar o tamanho dos arquivos e atrasar o processamento desses dados (RAY. 2003).

Outra questão importante é que o XML em si não possui mecanismos para criação de regras de validação de documentos XML, isto é, não é possível definir quais blocos são válidos no documento. Para isto, é necessária a utilização de outras linguagens que irão definir estas regras. Nos próximos itens falaremos de duas delas: DTD e XML Schema.

2.2 Estruturas para arquivos XML

2.2.1 DTD

Uma *Document Type Definition* (DTD) pode ser definida como um conjunto de regras que define quais tipos de dados e entidades farão parte de um documento XML. O seu propósito é definir quais blocos são válidos na construção deste documento (RAY, 2003). Assim, não é mais necessário ao programador se preocupar com a validação do documento e a integridade dos dados, a DTD cumprirá este papel.

Uma DTD define um documento XML da seguinte maneira (GRONER, 2009):

- Declaração de um conjunto de elementos: não é possível utilizar outros elementos que não estejam definidos neste conjunto;
- Definição do conteúdo para cada elemento: a definição de conteúdo é um padrão que nos diz quais os elementos ou dados que aquele determinado elemento XML pode conter, em qual ordem, quantidade e se é opcional ou obrigatório;
- Declaração de um conjunto de atributos para cada elemento: cada declaração de atributo define o nome, tipo, valores padrões (se aplicável) e comportamento (obrigatório ou opcional) do atributo.

Na figura 2.2, temos uma DTD para o arquivo *alunos.xml* da figura 2.1. Esta DTD permite a criação de vários blocos de aluno onde cada *aluno* deve conter como sub-elementos os elementos *nome* e *cidade*. O elemento nome também deverá conter dois sub-elementos: *primeiro_nome* e *sobrenome*. Nas linhas 5, 6 e 7 temos que os elementos *primeiro_nome*, *sobrenome* e *cidade* poderão apenas conter informações, mas não outros elementos. Isto é garantido pela informação #PCDATA atribuídas a eles.

```
1 <!DOCTYPE alunos[
2       <!ELEMENT alunos (aluno+)>
3       <!ELEMENT aluno (nome, cidade)>
4       <!ELEMENT nome (primeiro_nome, sobrenome)>
5       <!ELEMENT primeiro_nome (#PCDATA)>
6       <!ELEMENT sobrenome (#PCDATA)>
7       <!ELEMENT cidade (#PCDATA)>
8 ]>
```

Figura 2.2 - Documento alunos.dtd.

Apesar de ser uma linguagem simples, a DTD é escrita na linguagem EBNF (*Extended Backus-Naur Form*) o que pode dificultar o seu entendimento. Ela também possui outras limitações como: imposição na ordem de ocorrência de elementos, tipos de dados limitados, um documento XML não pode estar associado a mais de uma DTD, entre outras limitações.

Uma forma de contornar a maioria dos problemas da DTD é a utilização da linguagem XML Schema apresentada no item seguinte.

2.2.2 XML Schema

XML Schema é uma linguagem baseada em XML e serve para, assim como a DTD, definir regras de validação de documentos XML (W3C, 2010). Ela foi criada para suprir as principais limitações da DTD, como explicadas no item anterior.

Assim como na DTD, a XML Schema define um documento através da criação de um conjunto de elementos válidos para o documento. Também é necessário informar qual será o conteúdo de cada elemento, se ele irá conter sub-elementos e/ou dados, em que ordem, em que quantidade e se é obrigatório. A declaração de atributos também deve ser feita para cada elemento, informando o nome, tipo, valores padrões (se aplicável), e comportamento (obrigatório ou opcional) do atributo.

Na figura 2.3, temos um documento XSD para o arquivo *alunos.xml*. Ele define a estrutura e regras para a criação de um documento XML onde deve ter um elemento principal *aluno*, que é formado por dois sub-elementos: *nome* e *cidade*. O sub-elemento *nome*, assim como o elemento *aluno*, é formado por dois sub-elementos: *primeiro_nome* e *sobrenome*. É informado através da *tag <xs:sequence>* que os elementos devem vir na seqüência em que foram dispostos no arquivo XSD. Também é definido que os dados dos elementos serão do tipo *string*, caracterizando uma cadeia de caracteres.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="aluno">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="nome">
          <xs:complexType>
            <xs:sequence>
                <xs:element name="primeiro_nome"  type="xs:string"/>
                <xs:element name="sobrenome"  type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="cidade"  type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figura 2.3 - Documento alunos.xsd.

A XML Schema fornece diversas funcionalidades. É possível utilizar tipos de dados (*strings*, *date*, entre outros), possui um maior controle sobre o número de ocorrências de um

determinado elemento, possibilita a reutilização de código, suporta a construção de tipos próprios derivados dos tipos básicos, realiza relacionamentos entre elementos de dados dentro do XML, entre outras funcionalidades (RAY,2003).

2.3 DOM

DOM (*Document Object Model*) é uma especificação, independente de plataforma e linguagem, definida pela W3C (*World Wide Web Consortium*) para acessar e manipular documentos HTML, XHTML e XML (W3Shools ,20011). Podemos dizer que o DOM é um conjunto de objetos e métodos específicos para interagir com arquivos estruturados em *tags*.

Para acessar e manipular documentos XML é construído, através de um *parser*, um modelo de árvore do arquivo. Assim, utilizando métodos é possível realizar operações nos nodos das árvores, tais como: remoção, adicionamento, substituição e clonagem de um nodo (VITTORI, 2000).

Exemplo, na linguagem JAVA, de construção de uma árvore XML:

```
try{
        Document meuDocumento  = XmlDocument.createXmlDocument(
                                                    "file:/Alunos.xml", true);
}
catch( IOException  err ) {...}
catch( SAXException  err ) {...}
catch( DOMException  err ) {...}
```

Figura 2.4 - Criação da árvore xml do documento *alunos.xml*.

Na figura 2.4, temos a criação, através da classe *XmlDocument* que executa o *parsing*, da árvore XML do documento Alunos.xml, isto é, temos a criação do objeto DOM na memória. É necessário a capturar os erros para realizar esta implementação.

```
if ( meuNodo.hasChildNodes()  ){
        // processar os filhos de meuNodo
}
```

Figura 2.5 - Exemplo de utilização de método do DOM.

Na figura 2.5 temos um exemplo de utilização do método *meuNodo.hasChildNodes( )* da interface *Node* do DOM. É testado se o nodo *meuNodo* possui filhos; se tiver, algum processamento especial poderá ser feito.

Para ilustrar a árvore criada pelo *parser* será utilizado o exemplo de arquivo XML da figura 2.6.

```
<livraria>
        <livro categoria="Romance">
                <titulo linguagem="portugues">O nome da rosa</titulo>
                <autor>Umberto Eco</autor>
                <ano>1980</ano>
                <valor>50.00</valor>
        </livro>
</livraria>
```

Figura 2.6 - Documento Livraria.xml.

Neste exemplo temos a representação de uma livraria que contém dados sobre livros. Cada livro contém informações sobre sua categoria, linguagem, título, autor, ano de lançamento e valor. Note que além de elementos temos informações armazenadas como atributos de alguns elementos, é o caso da categoria do livro e da linguagem do mesmo.

A seguinte imagem ilustra a árvore que será criada, pelo *parser*, do documento XML do exemplo anterior.

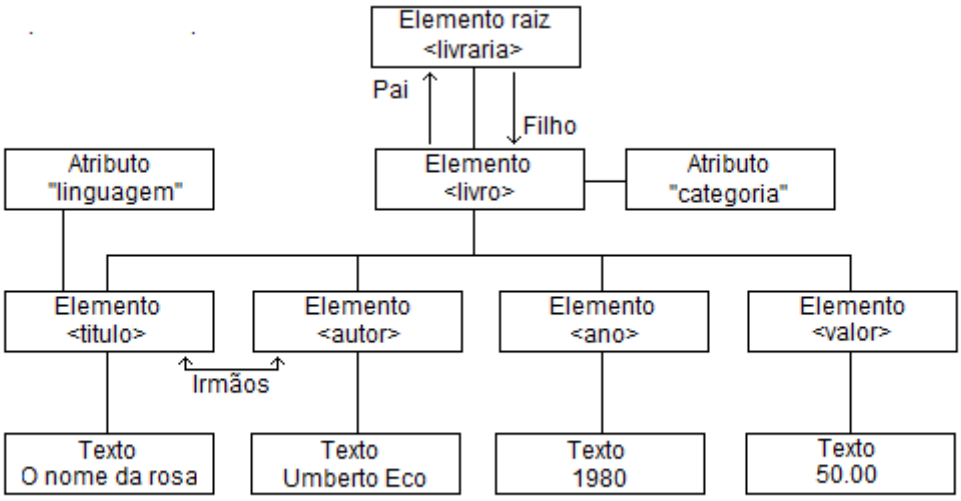

Figura 2.7 - Exemplo de árvore XML.

Na figura 2.7, temos o arquivo da figura 2.6 ilustrado na forma de árvore. A estrutura da árvore é a mesma do arquivo XML original. Para acessarmos os elementos que contêm informações textuais devemos percorrer a árvore a partir do elemento raiz até o elemento desejado. Por exemplo: para acessarmos o nome do autor de um *livro* devemos,

primeiramente, acessar o elemento raiz *livraria*, depois o elemento *livro* e então o elemento *autor* que conterá o nome do autor. Na ilustração, podemos perceber a relação entre os nodos da árvore, que podem ser de pai, filho e irmãos.

É possível, também, acessar elementos através do nome da *tag* desejada. Chamando a função *getElementsByTagName("nome")* o DOM retorna uma lista com os elementos que contenham a *tag nome*. Por exemplo, se adicionarmos o seguinte livro na coleção de livros do arquivo da figura 2.6,

```
<livro categoria="Romance">
        <titulo linguagem="portugues">O segredo</titulo>
        <autor>Rhonda Byrne</autor>
        <ano>2007</ano>
        <valor>60.00</valor>
</livro>
```

e chamássemos a função *getElementsByTagName("titulo")* o DOM retornaria a lista de nodos presentes na figura 2.8. Os nodos são retornados na mesma ordem em que são especificados no arquivo XML.

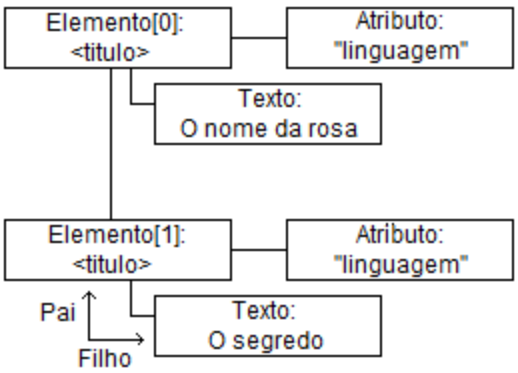

Figura 2.8 - Ilustração do vetor retornado pela função *getElementsByTagName("titulo")* .

A relação entre os nodos Elemento e o nodo de Texto, que contém a informação textual do título do livro, é de pai para filho. Portanto, caso desejarmos acessar o nome do título do livro do primeiro livro cadastrado é necessário chamar a seguinte função *getElementsByTagName("titulo")[0].childNodes[0].nodeValue()*. A função *getElementsBy TagName("titulo")*, como explicado anteriormente, retornará a lista de nodos com a *tag titulo*. Acessamos então o primeiro elemento da lista e então acessamos os nodos filhos deste elemento. Como o elemento *título* só tem como filho o nodo *Texto*, então acessamos o primeiro filho e utilizamos a função *nodeValue()* para acessar o valor textual deste elemento.

2.4 DOT

DOT é uma linguagem para representação de grafos. Há dois tipos de grafos suportados: grafos direcionados e não direcionados. A linguagem também oferece suporte para diversas opções de personalização do grafo, tais como: alterações na cor e estilo de arestas, forma de representação de nodos, entre outras (Wikipedia, 2011). A linguagem define o grafo, porém não oferece opção para desenho do mesmo. Assim, é necessário utilizar um *software* que o desenhe.

Na figura 2.9, temos um exemplo de um grafo direcionado na linguagem DOT:

```
digraph grafo{
        a -> b -> c;
        b -> d;
}
```

Figura 2.9 - Documento grafo.dot.

A figura 2.9 representa um grafo direcionado onde o elemento A será pai do elemento B, e o elemento B será pai de C e D.

A seguir, na figura 2.10, temos outro exemplo de grafo, nele estão feitas algumas personalizações de arestas e rótulos:

```
graph grafo {
        a [label="Alfa"];
        b [shape=box];
        a -- b -- c [color=blue];
        b -- d [style=dotted];
}
```

Figura 2.10 - Documento grafo2.dot.

Neste exemplo foram feitas modificações no exemplo da figura 2.8. O rótulo do nodo *a* foi alterado para *Alfa*, o formato do nodo *b* foi alterado para o formato de caixa, as arestas formadas entre os nodos *a* e *b* e entre os nodos *b* e *c* foram alteradas para a cor azul e o estilo da aresta entre os nodos *b* e *d* foi alterada para um estilo pontilhado.

2.5 GRAPHVIZ

GraphViz (GraphViz, 2011) é um *software* de código aberto para desenhos de grafos especificados na linguagem DOT. Ele oferece suporte para a criação de grafos complexos com diferentes cores de fontes, estilos de arestas, formatos personalizados de nodos, entre outras opções.

Na figura 2.11, temos um grafo, gerado pela ferramenta *GraphViz*, que representa a visualização gráfica do grafo da figura 2.9. O elemento *a* é pai de *b* que, por sua vez, é pai de *c* e *d*.

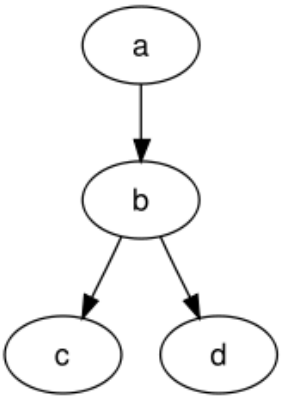

Figura 2.11 - Exemplo de grafo gerado pelo *GraphViz*.

Na figura 2.12, temos um grafo que representa a visualização gráfica do grafo da figura 2.10. É possível notar visualmente, através da figura 2.12, que a ferramenta *GraphViz* da suporte para personalização dos grafos.

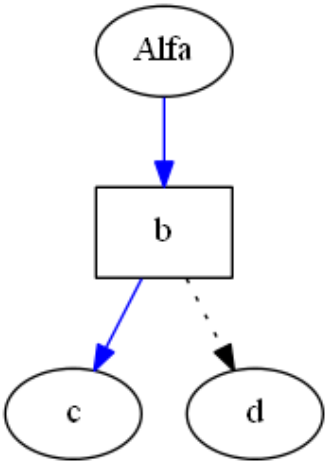

Figura 2.12 - Exemplo de grafo personalizado.

A *GraphViz* também fornece suporte à criação de sub-grafos, isto é, grafos dentro de um grafo. Estes sub-grafos podem ser personalizados para gerar uma melhor visualização, facilitar ou até salientar uma parte do grafo para melhorar o seu entendimento.

A seguir, na figura 2.13, temos a especificação de um grafo direcionado na linguagem DOT. Neste grafo, temos a criação de dois sub-grafos dentro do grafo principal. Cada sub-grafo pode ser personalizado de maneiras diferentes.

```
digraph G {
  subgraph cluster_0 {
        style=filled;
        color=lightgrey;
        node [style=filled,color=white];
        a0 -> a1 -> a2 -> a3;
        label = "processo #1";
  }

  subgraph cluster_1 {
        node [style=filled];
        b0 -> b1 -> b2 -> b3;
        label = "processo #2";
        color=blue
  }
  inicio -> a0;
  inicio -> b0;
  a1 -> b3;
  b2 -> a3;
  a3 -> a0;
  a3 -> fim;
  b3 -> fim;
  inicio [shape=Mdiamond];
  fim [shape=Msquare];
}
```

Figura 2.13 - Exemplo de grafo que contém dois sub-grafos.

Na figura 2.14, temos a visualização do grafo. Nela podemos ver como a criação de sub-grafos melhorou a visualização dos processos representados no grafo. Também é possível realizar ligação entre nodos de diferente sub-grafos, como pode ser visto nas arestas entre os nodos *a1* e *b3* e entre os nodos *b2* e *a3*.

A ferramenta *GraphViz* oferece suporte para geração de grafos com diferentes tipos de *layouts*. Cada *layout* altera a forma como os nodos do grafo são dispostos na imagem. São eles:

- DOT: grafo em formato hierárquico. A orientação do grafo pode ser *Top to Bottom* ou *Left to Right*.
- NEATO: utiliza métodos heurísticos para geração de grafos menores. Utilizado para grafos com poucos nodos.
- FDP: modelo de geração de grafo semelhante ao NEATO. Oferece suporte a subgrafos.
- SFDP: versão do modelo FDP para geração de grafos maiores. Não oferece suporte a subgrafos.
- TWOPI: modelo de geração de grafos em formato radial.
- CIRCO: modelo de geração de grafos em formato circular.

Durante a geração do grafo é possível que ocorram sobreposições entre os nodos do mesmo. Esta situação piora o entendimento do grafo, assim, é necessário configurar o grafo para impedir que elas ocorram. A ferramenta *GraphViz* possibilita a utilização de métodos que eliminarão a sobreposição de nodos, alguns desses métodos se encontram abaixo:

- SCALE: remove sobreposições escalando as posições X e Y dos nodos.
- FALSE: utiliza um algoritmo de proximidade de grafos para remover sobreposições de nodos.
- DEFAULT: utiliza a configuração padrão de sobreposição de nodos. A configuração padrão pode variar conforme o modelo de grafo gerado.

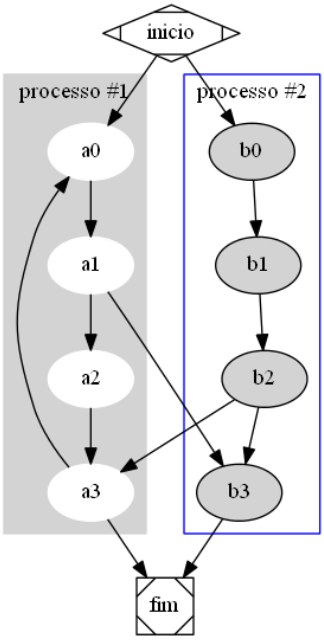

Figura 2.14 - Visualização do grafo da figura 2.13.

2.6 Ferramentas para visualização de documentos XML

Foram analisados diversos *softwares* para manipulação, edição e validação de documentos XML. Esta análise foi realizada com o intuito de verificar se esses *softwares* dão suporte à visualização gráfica de documentos XML, além de editarem e validarem estes documentos, como dito anteriormente.

As ferramentas *XML Spear* (XMLSpear, 2005), *XML Notepad* (XMLNotepad, 2007) e *XML Marker* (XMLMarker, 2011) dão suporte para a visualização do documento XML apenas no formato de árvore. Na figura 2.15, temos um exemplo de uma árvore gerada pela ferramenta *XML Spear* de um arquivo XML.

As ferramentas *Liquid XML Studio 2011* (Liquid Technologies, 2011) e *XMLSpy 2012* (Altova, 2011), além do suporte para visualização do documento XML em formato de árvore, oferecem uma visualização gráfica do esquema XSD de um documento XML, onde é possível realizar modificações nesta estrutura. Na figura 2.16, foi criada, no *software* Liquid XML Studio, uma visualização gráfica de um esquema XSD de um documento XML.

A desvantagem dos softwares que geram apenas a visualização de documentos XML em formato de árvore é que, para alguns usuários, que desconhecem o formato de estrutura de uma árvore, isso pode comprometer o entendimento da estrutura do documento XML. Para os *softwares* analisados que geram, além da visualização em árvore do documento XML, a visualização gráfica do esquema XSD do documento XML, a desvantagem é que é necessário que haja um esquema XSD para o documento XML. A ausência deste esquema impossibilita a sua visualização.

2.7 Considerações Finais

Após realizar esta pesquisa de fundamentação teórica, optou-se pela utilização da ferramenta *GraphViz* para a geração de grafos, pois esta ferramenta realiza o desenho de grafos na linguagem DOT, que é uma linguagem para descrição de grafos que apresenta muitas opções de personalização do grafo, opções estas suportadas pela *GraphViz*. Além disso, o *software GraphViz* conta com APIs em diversas linguagens, facilitando a integração da ferramenta com o software que será desenvolvido neste trabalho.

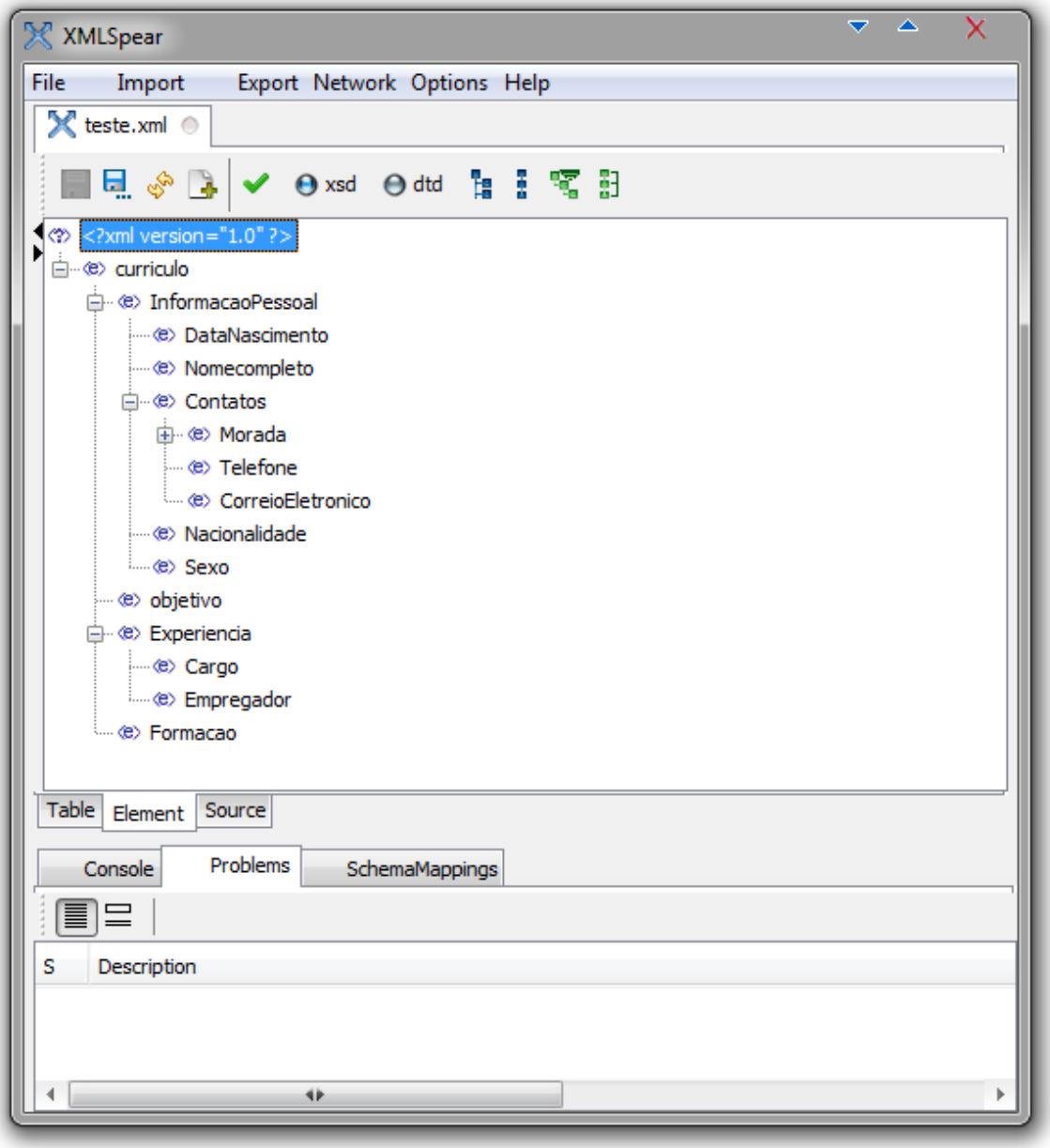

Figura 2.15 - Exemplo de árvore XML, gerada pelo programa *XMLSpear*.

Analisando os *softwares* da seção 2.6, é possível constatar que apenas a visualização em árvore, do documento XML, é oferecida aos usuários. Esta visualização, apesar de muito utilizada na área da computação, pode dificultar o entendimento do documento por parte de usuários que não estão habituados à esta estrutura e que não sabem como navegar pela mesma. A visualização em formato de grafo direcionado oferece um melhor entendimento entre as relações dos elementos do documento, facilitando a interpretação de usuários que não possuem conhecimento na área da computação.

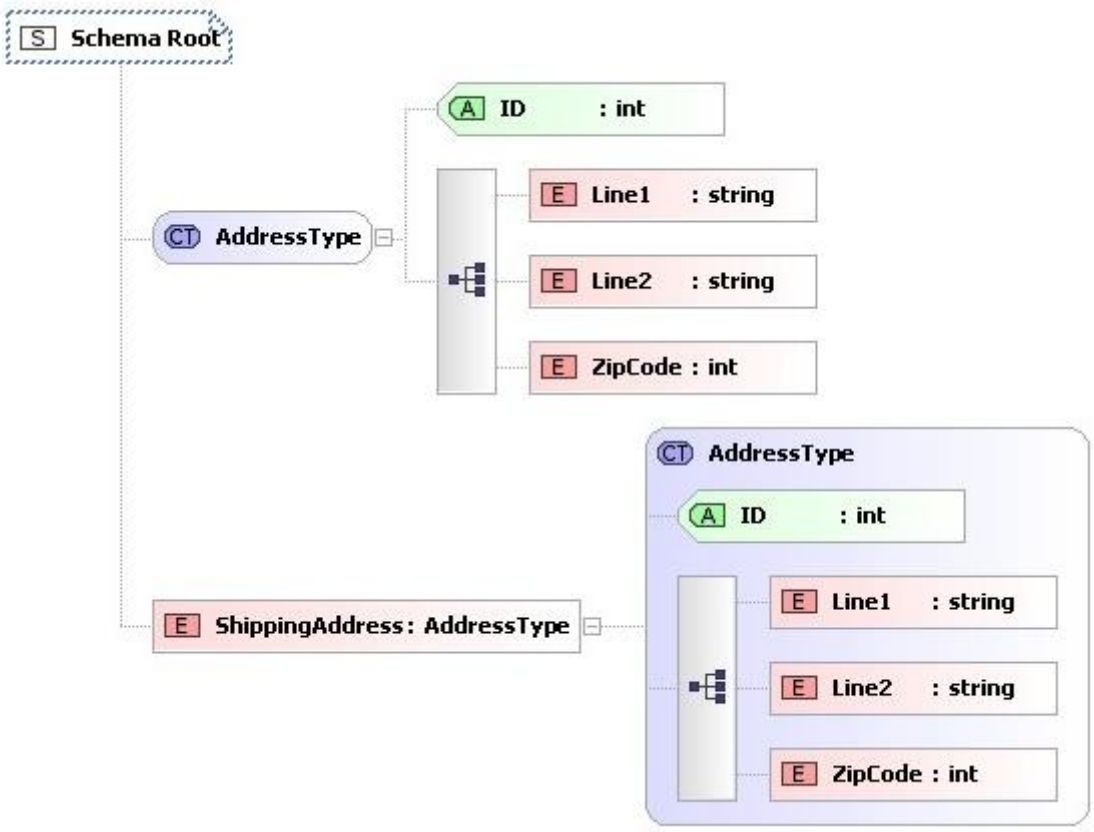

Figura 2.16 - Gráfico de um esquema XSD, gerada pelo *Liquid XML Studio*.

Assim, no capítulo a seguir, será apresentada uma proposta de ferramenta que gerará uma visualização gráfica em formato de grafo de um documento XML, sem a necessidade da existência de um esquema XSD do documento XML.

3 A FERRAMENTA *XMLGRAPH*

Este capítulo apresenta uma proposta para criação de uma ferramenta para visualização gráfica de documentos XML no formato de grafo.

3.1 Visão geral

A ferramenta *XMLGraph* tem como finalidade a visualização gráfica de documentos XML. Para isto, a ferramenta cria um documento na linguagem DOT que irá representar o grafo do documento XML. A criação deste grafo é feita observando-se a relação entre os elementos que se encontram na estrutura do documento XML. A interface da ferramenta é de fácil entendimento, contando com menus com opções para configuração do grafo e painéis onde as informações são mostradas ao usuário. No tratamento de exceções, mensagens de erro são exibidas para informar ao usuário o problema ocorrido. Além disso, o menu de ajuda auxilia em caso de dúvidas quanto à utilização da ferramenta.

A seguir, na figura 3.1, é apresentado o fluxo de funcionamento do programa.

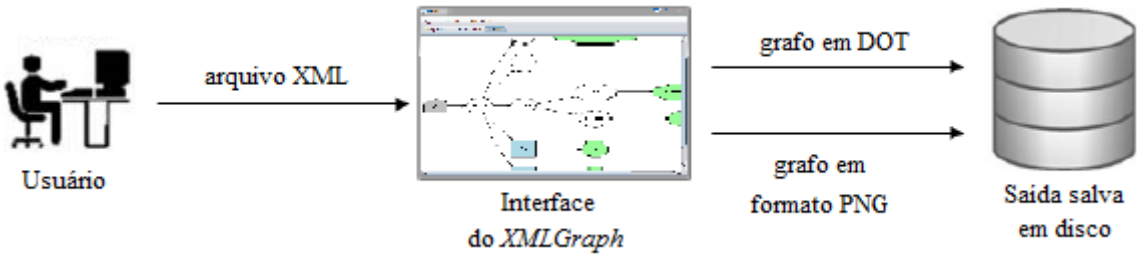

Figura 3.1 - Fluxo de Funcionamento.

Primeiramente o usuário escolhe um arquivo XML qualquer para ser enviado ao programa. Ao receber o arquivo, a ferramenta *XMLGraph* analisa o documento XML e cria

um documento DOT que irá representar o grafo, formado pelos elementos do documento XML. Logo após a criação do documento DOT, é desenhado , através de uma API em Java do *GraphViz*, o grafo representado no documento DOT. O desenho do grafo, assim como o documento DOT, podem ser analisados na ferramenta e se o usuário optar é possível salvar estes documentos em disco.

3.2 Mapeamento de XML para a linguagem DOT

A ferramenta se baseia no mapeamento da estrutura e conteúdo de um documento XML para uma representação de um grafo na linguagem DOT. Para isto, é realizado um mapeamento nos elementos do arquivo XML e , assim, são criados nodos que representarão os elementos, atributos e informações de conteúdo e arestas que representarão a relação entre os nodos. Desta forma, o usuário fornece um documento XML à ferramenta que irá criar o documento DOT e os resultados, documento DOT e a imagem do grafo, são mostrados em painéis presentes na interface.

Os mapeamentos utilizados na tradução, da estrutura e conteúdo, do documento XML para um grafo representado num documento DOT podem ser vistos a seguir, primeiro mostrando o arquivo XML de entrada, depois a sua tradução para DOT e enfim o desenho do grafo.

1. O nodo elemento raiz do documento XML é mapeado para um nodo raiz no grafo:

   XML: `<usuarios></usuarios>`

   DOT:
   ```
   usuarios [shape=house,style=filled, fillcolor=gray];
   graph [root="usuarios"];
   ```

   A figura 3.2 ilustra o grafo gerado a partir do documento XML.

Figura 3.2 - Grafo de um elemento raiz.

2. Um nodo elemento no documento XML é mapeado para um novo nodo no grafo, quando este novo nodo é criado é também criada uma aresta ligando este nodo ao seu nodo pai:

   XML:
   ```
   <usuarios>
       <usuario> </usuario>
   ```

```
</usuarios>
```

DOT:
```
nomeFilho [label = "usuario" ];
nomePai -> nomeFilho;
```

A variável *nomeFilho* é gerada a partir de uma combinação de informações que tornam o nodo único, pois se houvesse mais de um usuário no documento XML ambos seriam referenciados da mesma maneira, gerando um resultado inesperado no grafo. A variável *nomePai* é a referência de nome para o nodo pai do nodo Filho, esta variável também é gerada a partir de uma combinação de informações que tornam o nodo único.

A figura 3.3 ilustra o grafo gerado a partir do documento XML.

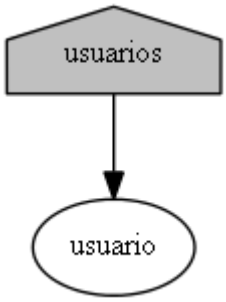

Figura 3.3 - Adição de um elemento no grafo da figura 3.2.

3. Um nodo texto no documento XML é mapeado para um novo nodo no grafo com uma aresta ligando ao seu pai:

XML:
```
<usuarios>
   <usuario>Fulano</usuario>
</usuarios>
```

DOT (para adicionar o nodo texto):
```
nomeInfo [label = "Fulano];
nomePai -> nomeInfo ;
nomeInfo  [style=filled, fillcolor= palegreen  , shape=octagon];
```

A figura 3.3 ilustra o grafo gerado a partir do documento XML.

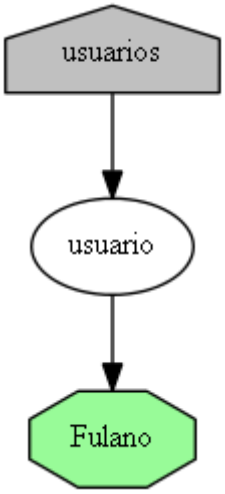

Figura 3.4 - Adição de um nodo texto no grafo da figura 3.3.

4. Um nodo atributo de um elemento no documento XML é mapeado para um novo nodo no grafo com aresta entre o nodo atributo e o nodo elemento. Caso o nodo atributo a ser inserido contenha informações, um novo nodo é criado e uma aresta é criada para ligar o nodo atributo com o nodo de informações do atributo:

   XML:
   ```
   <usuarios>
     <usuario id="1"> Fulano </usuario>
   </usuarios>
   ```

   DOT: (para adicionar nodo atributo):

   ```
   nomeFilho [label = "id" ];
   nomePai  -> nomeFilho;
   nomeFilho  [style=filled, fillcolor= lightblue, shape=box];
   nomeInfo [label = "1"];
   nomeFilho -> nomeInfo ;
   nomeInfo  [style=filled, fillcolor= palegreen  , shape=octagon];
   ```

   A figura 3.5 ilustra o grafo gerado a partir do documento XML.

5. Um nodo CDATA é mapeado para um novo nodo do grafo com uma aresta ligando-o ao seu pai:

   XML:
   ```
   <usuarios>
     <usuario id="1">
       <nome> Fulano </nome>
       <tag_usuario> <![CDATA[ <usuario 1>]]></tag_usuario>
     </usuario>
   </usuarios>
   ```

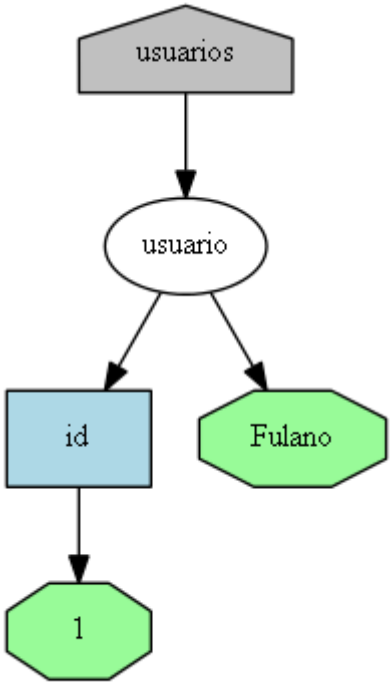

Figura 3.5 - Adição de um nodo atributo com informação no grafo da figura 3.4.

DOT(para adicionar nodo CDATA):

```
nomeFilho [label = "tag_usuario" ];
nomePai  -> nomeFilho;
nomeInfo [label = "<usuario 1>"];
nomeFilho -> nomeInfo ;
nomeInfo  [style=filled, fillcolor= darksalmon, shape=octagon];
```

A figura 3.6 ilustra o grafo gerado a partir do documento XML.

Nodos do tipo *CDATA* não devem conter os caracteres: " (aspas duplas) e / (barra) pois, apesar do *parser* aceitar, quando o grafo for gerado estes caracteres entrarão em conflito com a sintaxe do software *GraphViz* gerando resultados inesperados.

6. Um nodo elemento misto é mapeado para dois novos nodos no grafo, um contendo o nome do elemento e o outro contendo a informação textual do elemento. O nodo contendo o nome do elemento é ligado ao seu pai e o nodo contendo a informação textual é ligado ao nodo que contém o nome do elemento:

XML:
```
<usuarios>
  <usuario id="1">
    <nome>
      Fulano <sobrenome> Beltrano </sobrenome>
    </nome>
  </usuario>
</usuarios>
```

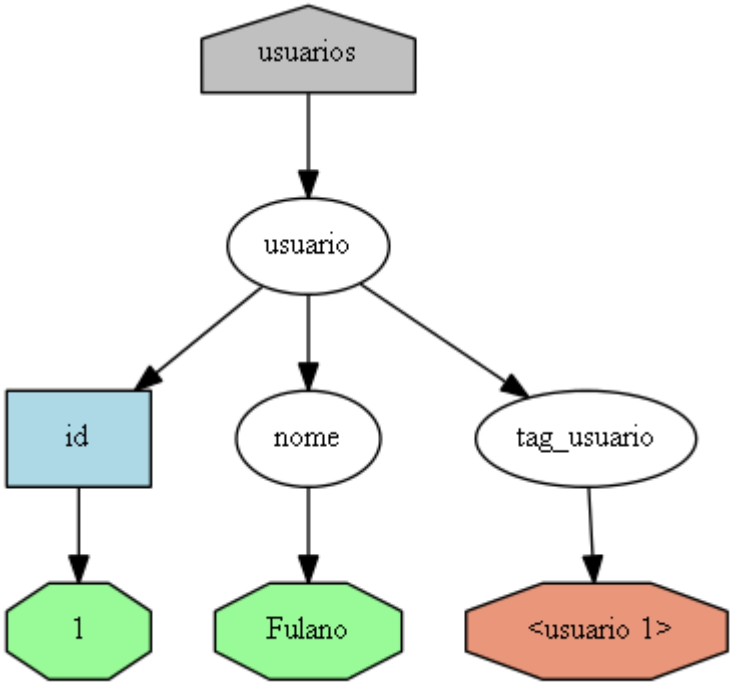

Figura 3.6 - Adição de um nodo elemento *nome* e um nodo CDATA no grafo da figura 3.5.

DOT (para adicionar nodo elemento misto):

```
nomeFilho [label = "nome" ];
nomePai  -> nomeFilho;
nomeInfo [label = getText(nodoFilho )];
nomeFilho -> nomeInfo ;
nomeInfo  [style=filled, fillcolor= lightgoldenrodyellow, shape=octagon];
```

A figura 3.7 ilustra o grafo gerado a partir do documento XML.

A função *getText* irá percorrer todos os filhos do nodo *nodoFilho* e irá criar uma *string* que conterá a soma de toda a informação textual de *nodoFilho*. Esta opção de juntar toda a informação num nodo texto foi escolhida, pois, na maioria dos casos a utilização de um elemento misto é feita para representar melhor a informação do nodo em questão. Por exemplo, a utilização da *tag* <*b*> da linguagem HTML, utilizada para deixar um texto em *negrito*, não é uma informação relevante para se criar um novo nodo no grafo.

A ferramenta considera elemento misto aquele elemento que contém ocorrências de pelos menos dois dos seguintes tipos de nodos : nodo texto, nodo elemento e nodo CDATA.

3.3 Criação de sub grafos

Dependendo do número de nodos presentes no grafo gerado, a sua visualização pode ficar muito confusa, pois apesar do *GraphViz* dar suporte à eliminação de sobreposições de nodos, estes mesmos nodos podem ficar em posições ruins para o entendimento do grafo.

A ferramenta *XMLGraph* dá suporte para a criação de sub grafos. Desta maneira, é possível para a ferramenta *GraphViz* eliminar sobreposições de sub-grafos, melhorando o entendimento do grafo. Um ponto negativo é que com a criação destes sub grafos o grafo pode aumentar de tamanho.

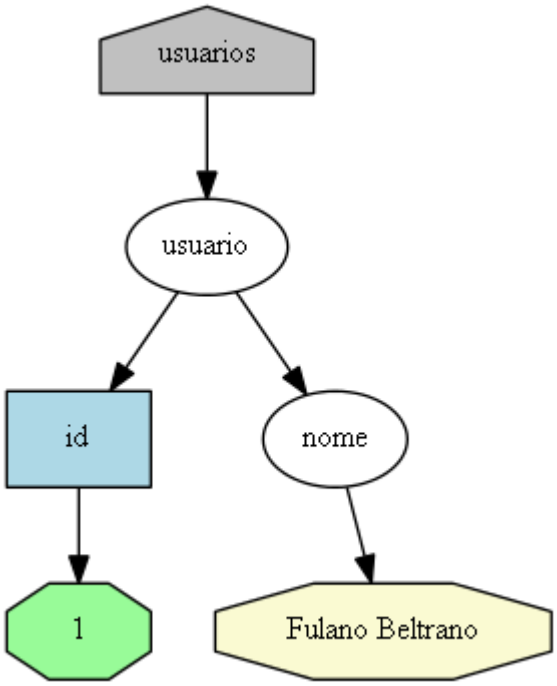

Figura 3.7 - Adição de um nodo elemento misto no grafo da figura 3.5.

A criação de um sub grafo é feita assim que um novo nodo elemento, nodo atributo, ou nodo elemento misto é adicionado ao grafo. O sub grafo de um nodo só é fechado quando todos os elementos filhos do nodo forem analisados. Assim, é garantido que todos os nodos filhos do nodo estão contidos no sub grafo.

Para visualizar a criação de sub grafos utilizaremos o arquivo XML da figura 3.8.

```
<usuarios>
   <usuario id="1"> Fulano </usuario>
</usuarios>
```

Figura 3.8 - Arquivo XML com nodos do tipo texto, atributo e elemento.

A figura 3.9 contém o grafo gerado em DOT para o documento XML do item 4 da seção de mapeamento. Este grafo contém algumas configurações iniciais de visualização e de tratamento de sobreposições.

A Figura 3.10 mostra os sub grafos criados para o documento XML da figura 3.8 Observe que os sub grafos estão com um contorno preto para facilitar a visualização.

```
digraph grafo {
  layout=dot;
  overlap=false;
  edge  [fontsize=12];
  node  [fontsize=12];
  usuarios [shape=house,style=filled,  fillcolor=gray];
  graph [root="usuarios"];

  usuarios->usuarios_usuario1;
  subgraph cluster_usuarios_usuario1{
    color=white;
    usuarios_usuario1  [label="usuario"];
    usuarios_usuario1->usuarios_usuario1_id;

    subgraph cluster_usuarios_usuario1_id{
      color=white;
      usuarios_usuario1_id  [label="id"];
      usuarios_usuario1_id  [style=filled, fillcolor="lightblue"  , shape="box"];
      usuarios_usuario1_id->usuarios_usuario1_id_info;
      usuarios_usuario1_id_info  [label="1"];
      usuarios_usuario1_id_info  [style=filled, fillcolor="palegreen"  , shape="octagon"];
    }
    usuarios_usuario1->usuarios_usuario1_info;
    usuarios_usuario1_info  [label=" Fulano "];
    usuarios_usuario1_info  [style=filled, fillcolor="palegreen"  , shape="octagon"];
  }
}
```

Figura 3.9 - Grafo em DOT para o XML da figura 3.8.

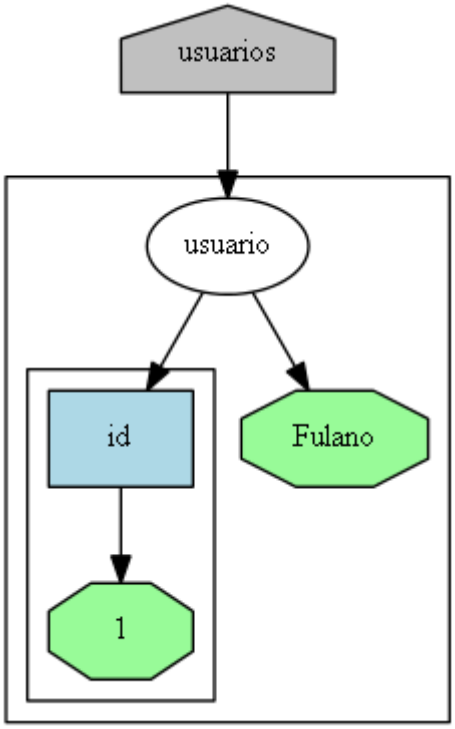

Figura 3.10 - Exemplo de grafo com sub-grafos.

O capítulo a seguir, mostrará detalhes da implementação da ferramenta *XMLGraph*, através da explicação do seu diagrama de classes e exemplos de utilização da ferramenta.

4 DESENVOLVIMENTO DA APLICAÇÃO

Neste capítulo será discutida a implementação da ferramenta *XMLGraph*. Ao final, será feita uma análise do software desenvolvido.

4.1 Tecnologias utilizadas

A ferramenta foi desenvolvida em linguagem Java no ambiente de desenvolvimento integrado (IDE) *NetBeans 7.0.1* (NETBEANS, 2011) devido à sua facilidade para a criação de interfaces gráficas. Além disto, a ferramenta *Netbeans* já vem com bibliotecas que dão suporte a utilização de DOM e de um *parser* para gerar o modelo de árvore do documento XML. Para integrar a opção de visualização do grafo na ferramenta, foi utilizado a API *GraphViz* Java API (GRAPHVIZ JAVA API, 2011) por contar com métodos que facilitam a utilização das funcionalidades do *GraphViz* dentro da ferramenta.

4.2 Implementação

O diagrama de classes apresentado na figura 4.1 mostra como o programa está organizado. Os atributos das classes foram omitidos por motivos de simplificação.

4.2.1 Classes implementadas

As dez classes do programa e os principais aspectos da implementação são detalhadas nas seções a seguir.

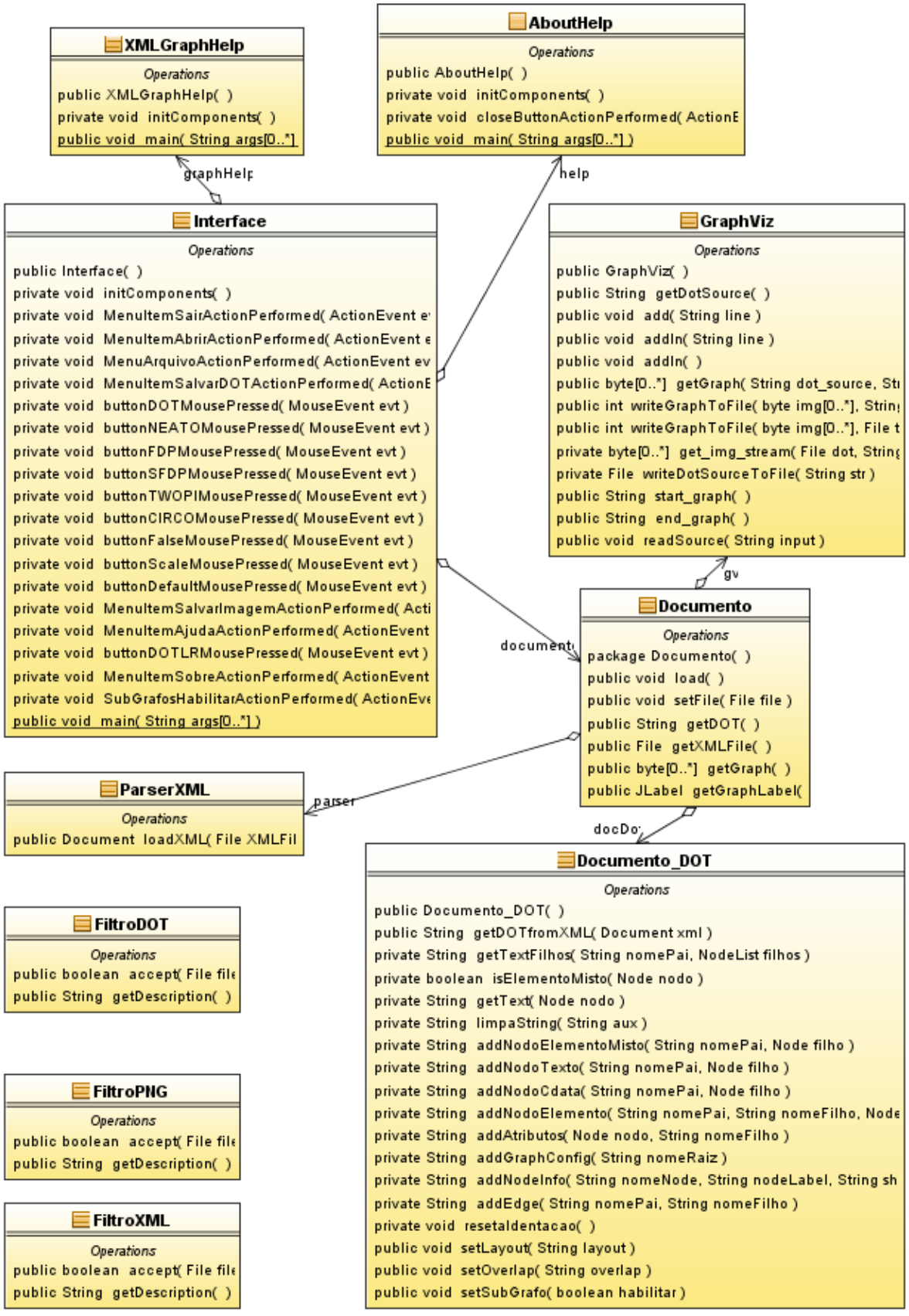

Figura 4.1 - Diagrama de classes do *XMLGraph*.

4.2.1.1 Classe *Interface*

Essa classe contém a *main* e todos os componentes visuais do programa.

- Carregamento do arquivo XML: quando o usuário carrega, através de um item de menu, um arquivo XML na ferramenta, este arquivo é enviado para um método da classe *Documento* onde será salvo e irá gerar o documento DOT e a imagem em grafo do arquivo XML.

- Salvamento do arquivo DOT e da imagem do grafo: quando o usuário opta por salvar, através de itens de menu, o arquivo DOT ou a imagem do grafo, métodos da classe *Documento* são acessados para receber estas informações e então salvar os arquivos em disco.

- Opções de configuração do grafo**:** como a ferramenta *XMLGraph* apresenta menus com opções para personalização do grafo, a cada mudança de configuração, seja ela de *layout*, *overlap* e de sub grafos, essas informações são enviadas a classe *Documento* para que seja gerado um novo arquivo DOT e, consequentemente, um novo grafo.

- Visualização do arquivo DOT e do grafo gerado: o painel com o conteúdo do arquivo DOT é preenchido com informação proveniente da invocação do método *getDot*. O painel com a imagem do grafo é preenchido com informação proveniente da invocação do método *getGraphLabel*. Ambos os métodos estão implementados na classe *Documento*.

4.2.1.2 Classe *Documento*

Essa é a classe onde o documento XML, o documento DOT e a imagem do grafo ficam salvos durante a execução do programa. Métodos para acessar estes documentos estão disponíveis na classe. É também nessa classe onde são acessados os métodos quer irão gerar o arquivo DOT e o grafo.

- Geração do DOT e do grafo: após o arquivo XML ser salvo na classe, é feito, dentro da função *load* e através da classe *ParserXML*, a criação do documento XML no formato de árvore DOM, após são chamadas as funções que irão gerar o documento DOT e o grafo deste documento. Para gerar o arquivo DOT é chamado o método *getDOTfromXML* da classe *Documento_DOT*, e para gerar o grafo é chamado o método *getGraph* da classe *GraphViz*.

4.2.1.3 Classe *ParserXML*

Classe onde, utilizando o método *loadXML*, é criado o *Document* que contém o arquivo XML em formato de árvore DOM. Esta classe tem por objetivo a abstração da forma como este *Document* é gerado.

4.2.1.4 Classe *Documento_DOT*

Essa é a classe que ira gerar o documento DOT a partir do arquivo XML em formato de árvore DOM. Tem por objetivo a abstração da forma como o documento DOT é gerado.

- Geração do documento DOT: o processo de geração do documento DOT começa no método *getDOTfromXML*, onde são inseridas as primeiras informações de configuração do grafo. Após é inserida as informações do nodo raiz (se ele existir) e então é chamado o método recursivo *getTextFilhos*, que irá adicionar as informações dos nodos filhos do nodo passado ao método.

- Adição de nodos: Dentro da função *getTextFilhos*, é feito um teste, para cada filho, para identificar o tipo de nodo. Conforme o resultado, são chamadas as funções *addNodoTexto*, *addNodoCdata*, *addNodoElementoMisto* e *addNodoElemento*, se o nodo for do tipo elemento ou elemento misto então é feita uma verificação se existem atributos no nodo e, se eles possuem informação, são adicionados nodos atributos, através da função *addAtributos*. As funções para adicionar novos nodos e atributos fazem o uso dos métodos *addNodeInfo* e *addEdge* para criar os nodos no documento e realizar as ligações com os nodos pais. Antes de adicionar a informação no *label* dos

nodos, a *string* contendo as informações é processada afim de diminuir o tamanho dos nodos e melhorar a visualização do grafo, com a função *limpaString*, para não conter caracteres de fim de linha e de dois, ou mais, caracteres de espaço juntos, além de permitir um tamanho máximo de 20 caracteres.

- Alteração nas opções do grafo: é possível, na interface da ferramenta *XMLGraph*, escolher opções de configuração do grafo. Quando isto é feito, utiliza-se as funções *setLayout*, *setOverlap* e *setSubGrafo*, para alterar as configurações do grafo.

4.2.1.5 Classes *XMLGraphHelp e AboutHelp*

Essas duas classes, que herdam de *JFrame*, compõem a interface das janelas de ajuda.

4.2.1.6 Classes *FiltroDOT*, *FiltroPNG* e *FiltroXML*

Essas três classes, que herdam de *FileFilter*, realizam filtros de arquivos nas pastas, na hora de salvar ou abrir algum arquivo, que permitem mostrar apenas os tipos de arquivos desejados.

4.2.1.7 Classe *GraphViz*

Essa classe encontra-se disponível na API *GraphViz Java API*. Ela facilita o uso do *software GraphViz* através de uma classe que conta com métodos para criação de grafos especificados na linguagem DOT e para a geração do grafo do documento DOT. A função *getGraph* é utilizada na ferramenta *XMLGraph* para gerar a imagem do grafo.

4.3 Utilização do programa

A aplicação desenvolvida permite carregar um documento XML e com ele criar um documento DOT, que conterá a especificação do grafo, e a visualização do grafo gerado a partir do documento DOT.

A figura 4.2 representa o documento XML que será utilizado nos exemplos de grafos para os próximos item desta seção.

```
<usuarios>
   <usuario id="0">
      <nome>
         <primeiro>Fulano</primeiro>
         <sobrenome>De Tal </sobrenome>
      </nome>
      <email>fulanotal@gmail.com</email>
   </usuario>
   <usuario id="1">
      <nome>
         <primeiro>Beltrano </primeiro>
         <sobrenome>De Tal </sobrenome>
      </nome>
      <email>beltranotal@gmail.com</email>
   </usuario>
</usuarios>
```

Figura 4.2 - Documento usuarios.xml.

4.3.1 Carregamento do arquivo XML de entrada

No menu *Arquivo*, presente na *interface*, é possível escolher a opção para abrir um arquivo XML. A figura 4.3 apresenta o menu para seleção do arquivo XML de entrada. O menu conta com um filtro para arquivos XML, facilitando a procura. Após o arquivo XML ser aberto, sua informação é carregada para um painel na *interface* onde o usuário poderá visualizá-lo. A figura 4.4 mostra o painel de visualização do arquivo XML aberto.

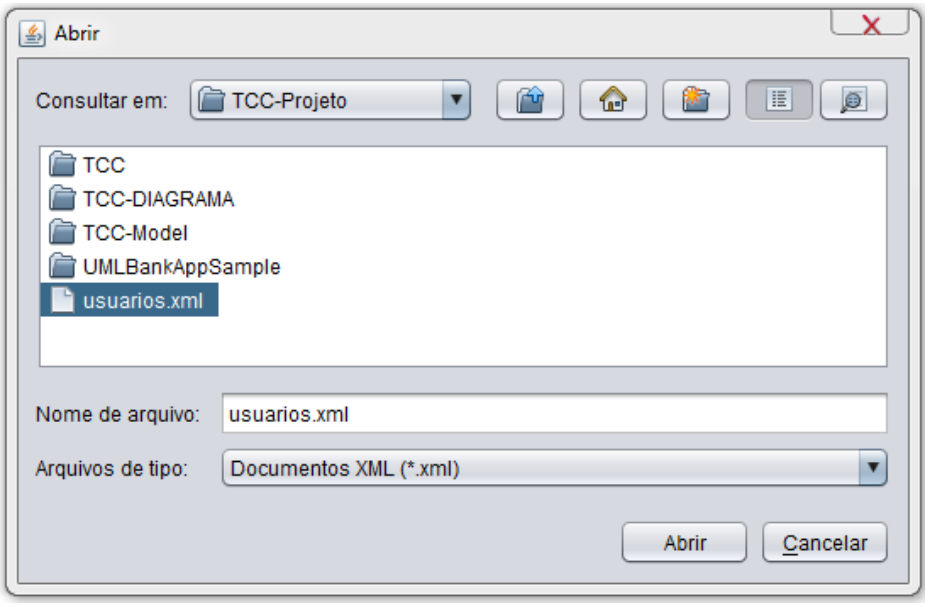

Figura 4.3 - Menu Abrir.

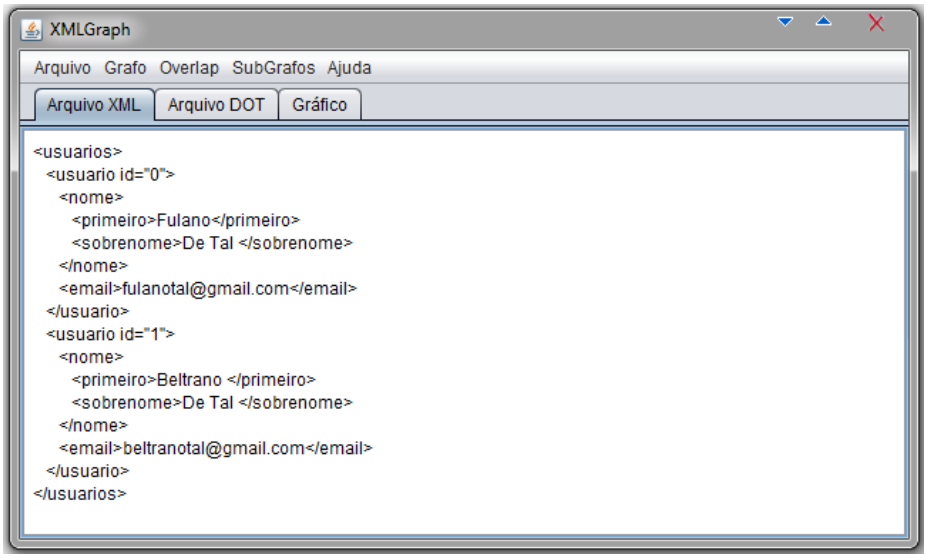

Figura 4.4 - Painel mostrando o conteúdo do arquivo XML.

4.3.2 Visualização do documento DOT

Após o arquivo XML ser aberto é criado um documento que conterá a especificação do grafo na linguagem DOT. A figura 4.5 mostra o painel contendo o documento DOT gerado a partir do arquivo XML de entrada e com as configurações escolhidas pelo usuário. Note que o documento gerado é maior que o tamanho do painel, para visualizar o restante do arquivo é necessário navegar no painel através de barras de rolagem.

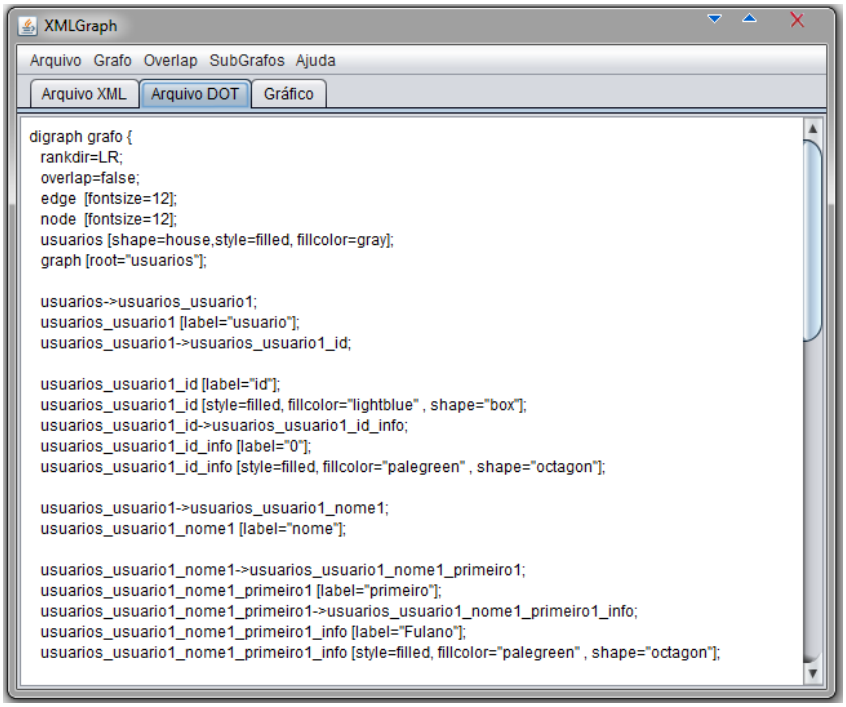

Figura 4.5 - Painel mostrando o documento DOT gerado.

4.3.3 Opções para visualização do grafo

Na interface da ferramenta, está disponível para o usuário opções para alterar as configurações do grafo gerado. Na criação do grafo, é possível escolher entre sete tipos de *layouts* , três tipos de tratamento de sobreposições de nodos e escolher se o grafo apresentará sub grafos ou não.

4.3.3.1 Tipos de *layouts* do grafo

Os sete tipos de *layouts* de grafo disponíveis são: DOT, DOT - LR, NEATO, FDP, SFDP, TWOPI e CIRCO. A figura 4.6 mostra o grafo resultante do documento XML da figura 4.2 com a escolha do layout DOT - LR.

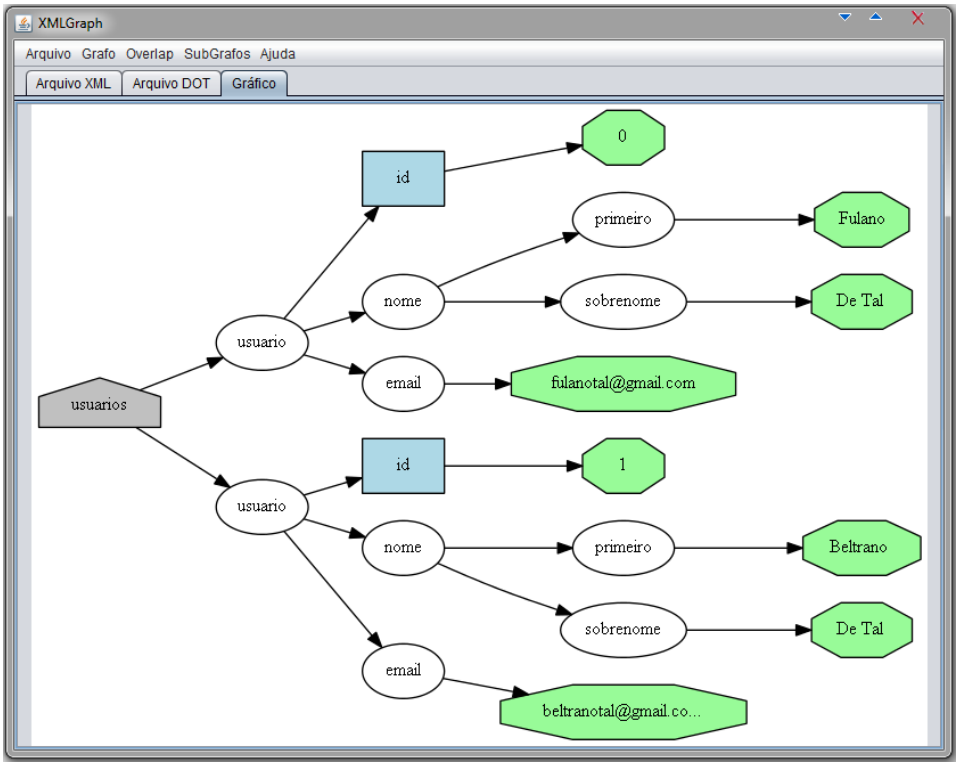

Figura 4.6 - Visualização do documento *usuarios.xml* no *layout* DOT - LR.

A figura 4.7 mostra o grafo resultante do documento XML da figura 4.2 com a escolha do *layout* NEATO.

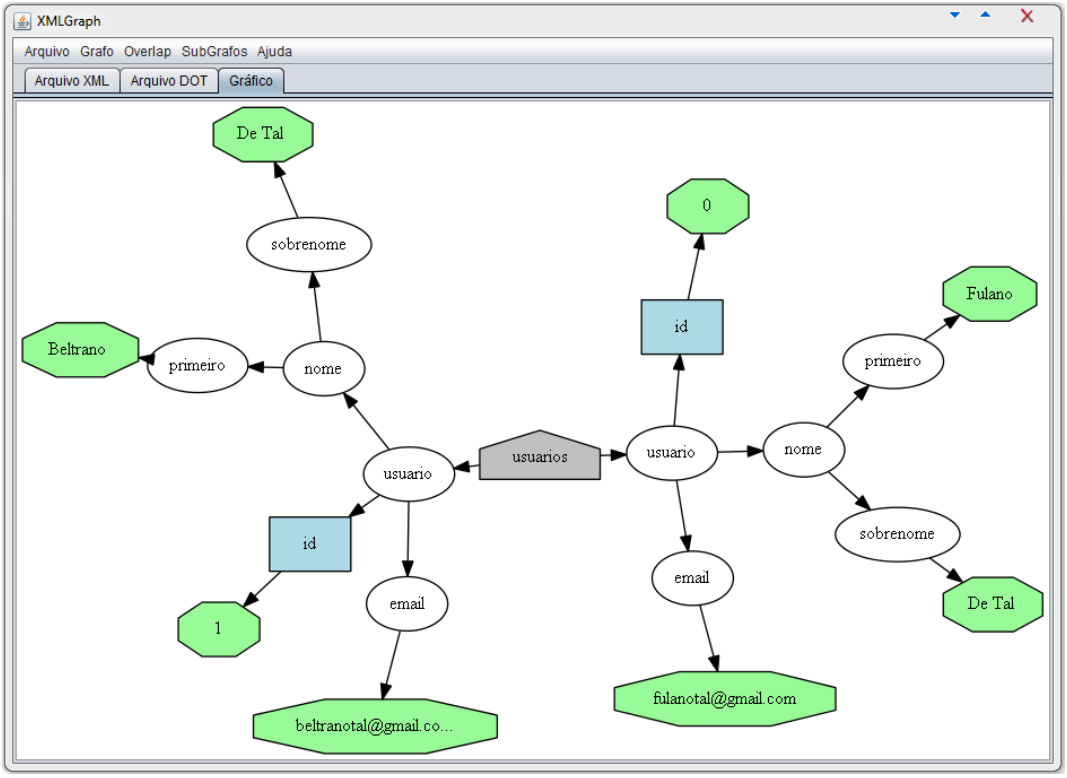

Figura 4.7 - Visualização do documento *usuarios.xml* no *layout* NEATO.

4.3.3.2 Tipos de tratamento de sobreposições de nodos

Os três tipos de tratamento de sobreposições de nodos são: FALSE, SCALE e DEFAULT. Dependendo do grafo, é interessante visualizá-lo para cada um dessas opções para achar grafos de tamanhos menores sem prejudicar o entendimento do mesmo.

4.3.3.3 Criação de sub grafos

Para melhorar o entendimento do grafo gerado, é possível optar pela criação de sub grafos. A utilização de sub grafos surtirá efeitos apenas nos métodos de *layout* DOT, DOT - LR e FDP. Por motivos de simplificação, o elemento *nome* do documento *usuarios.xml* da figura 4.2 foi simplificado para os exemplos das figuras 4.8 e 4.9.

A figura 4.8, mostra um grafo com *layout* FDP sem a criação de sub grafos e a figura 4.9 mostra o mesmo grafo com a criação de sub grafos. Note que a criação de sub grafos evitou a sobreposições de arestas que dificultavam o entendimento do grafo.

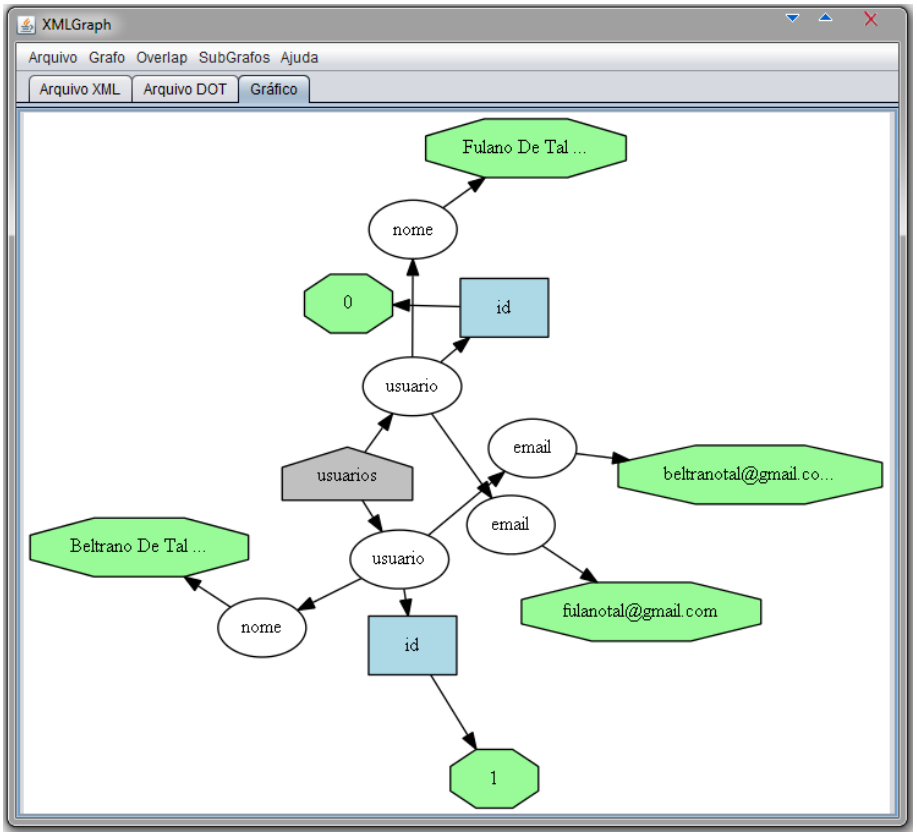

Figura 4.8 - Visualização de um grafo no formato FDP sem sub-grafos.

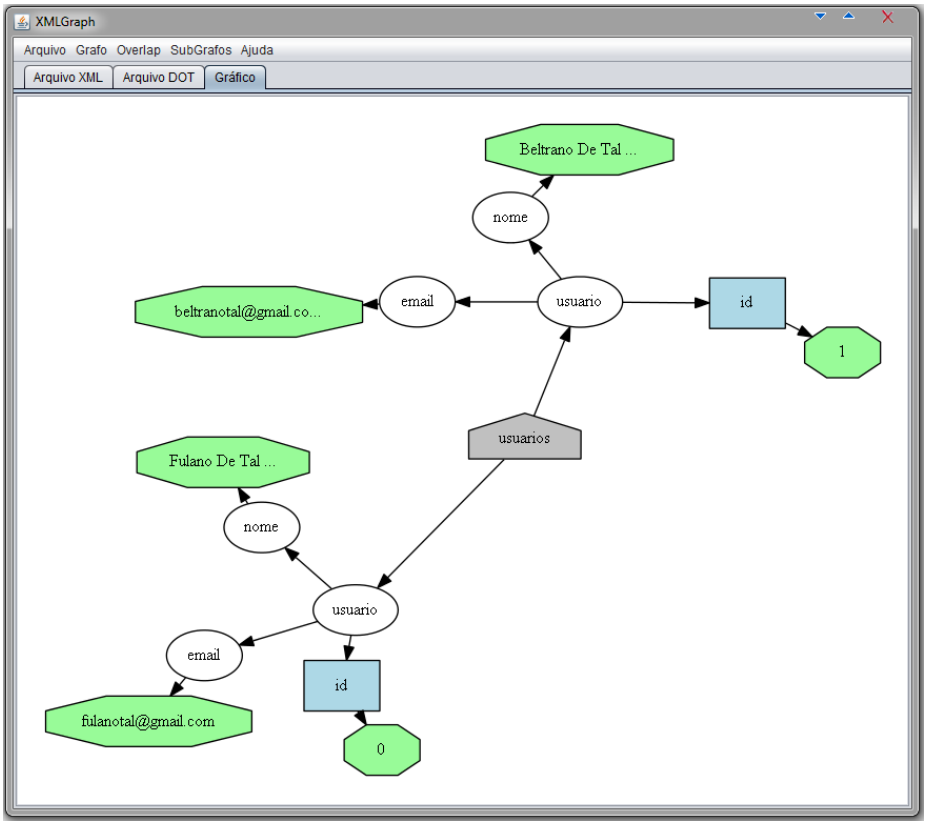

Figura 4.9 - Visualização do grafo da figura 4.8 com sub-grafos.

4.3.4 Salvamento do documento DOT e da imagem do grafo

No menu *Arquivo*, presente na *interface*, é possível escolher a opção para salvar o documento DOT ou a imagem do grafo, em formato PNG. Cada opção irá abrir um menu, semelhante ao da figura 4.3, para seleção do local e do nome do arquivo a ser salvo. Cada menu conta com um filtro de arquivos próprio, filtro de extensão DOT para salvar o documento DOT e filtro de extensão PNG para salvar a imagem do grafo.

4.3.5 Menus de Ajuda

A interface possui um menu de ajuda com dois itens. O primeiro explica os métodos de *layout*, de sobreposições de nodos, da criação de sub grafos e instruções para o correto funcionamento da ferramenta, conforme a figura 4.10. Nesta, explica-se as opções de *layout* disponíveis no *XMLGraph* para geração de grafos. O segundo item mostra o nome do desenvolvedor e o *link* onde podem ser encontradas mais informações sobre o programa.

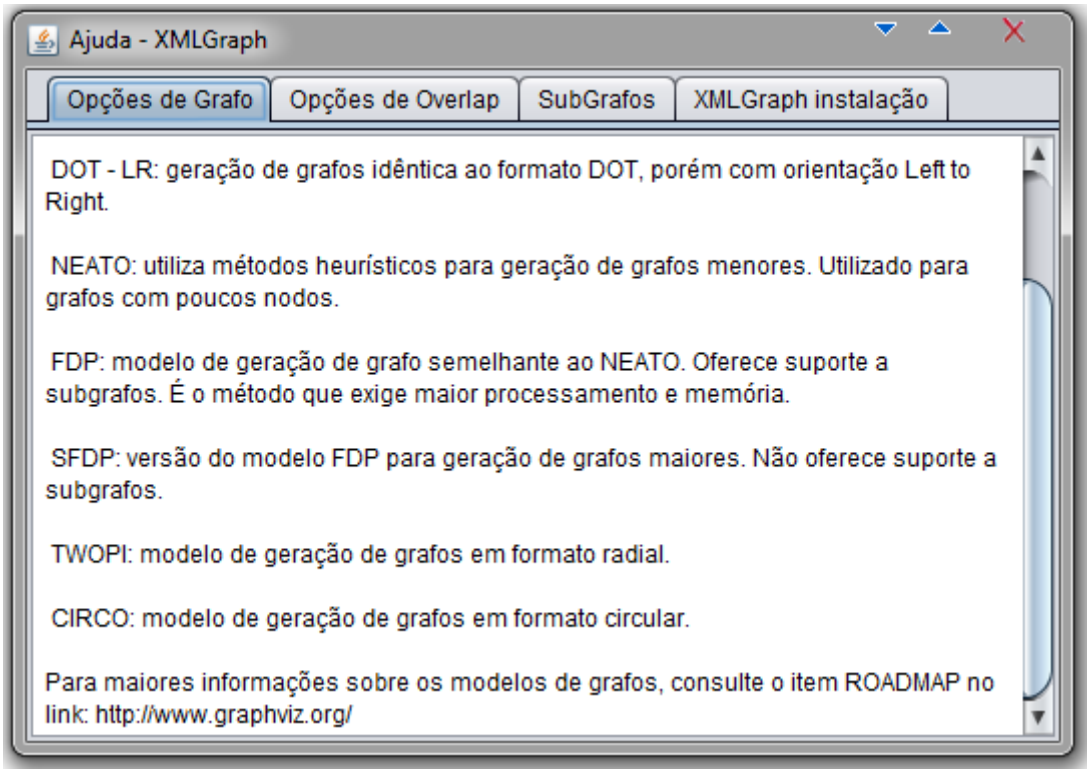

Figura 4.10 - Janela de ajuda para as opções da ferramenta *XMLGraph*.

4.3.6 Tratamento de erros

Nos testes realizados com o programa, foram identificados alguns erros que o usuário pode cometer. O primeiro trata-se de erros no arquivo de entrada. O usuário pode escolher um documento XML contendo erros de estruturação. Outro erro possível ocorre quando o usuário tenta salvar o documento DOT ou a imagem do grafo sem que um arquivo XML tenha sido carregado no programa. Por fim, o último caso de erro observado é quando o arquivo de entrada é muito grande e durante a criação do documento DOT e da imagem do grafo o espaço de memória para o programa não seja suficiente. Em todos esses casos, mensagens de erro em janelas *pop-up* alertam o usuário sobre o problema.

Como a ferramenta *XMLGraph* utiliza uma API para acessar as funcionalidades da ferramenta *GraphViz*, podem ocorrer erros relacionados a esta ferramenta. O primeiro erro é quando o usuário insere o caractere ″ (aspas dupla) ou o caractere / (barra) em nodos CDATA. Isto acarretará em uma interpretação errada, no momento da criação deste nodo no grafo, por parte da ferramenta *GraphViz*, que considera o caractere " (aspas dupla) como caractere de fim de string, e o caractere / (barra) como um caractere especial e não como informação. Neste caso, uma janela *pop-up* alerta o usuário da existência deste erro no documento XML. Outro erro que pode ocorrer é na criação de grafos com um número muito elevado de nodos. A ferramenta utiliza métodos, para evitar sobreposições de nodos e de sub grafos, que aumentam de complexidade conforme o número de nodos e de sub grafos, podendo ocasionar erros no processo da ferramenta *GraphViz* para criação do grafo.

4.4 Análise do *XMLGraph*

Esta seção apresenta uma análise das funcionalidades implementadas na ferramenta *XMLGraph*. Após, é feito uma comparação com os programas semelhantes previamente listados.

4.4.1 Criação do grafo em DOT

Para gerar a imagem do grafo do documento XML, a ferramenta *XMLGraph* realiza, primeiramente, o mapeamento do XML para a linguagem DOT. Com a criação deste documento, é possível utilizar todas as funcionalidades da ferramenta *GraphViz* para desenho

de grafos, pois ela realiza o desenho de grafos especificados na linguagem DOT. Também é possível salvar este documento DOT em disco, com a posse deste documento o usuário, que tiver conhecimento da linguagem DOT, pode realizar modificações para personalizar o grafo da maneira que desejar e então gerá-lo na ferramenta *GraphViz*.

4.4.2 Visualização do Grafo

Através de uma API em Java da ferramenta *GraphViz*, é possível visualizar o grafo gerado na ferramenta *XMLGraph*. A geração deste grafo através da API pode demorar dependendo do número de nodos no grafo. Esta visualização pode ser alterada pelo usuário através das opções presentes nos menus da ferramenta *XMLGraph*. Entretanto, as opções para visualização, implementadas na ferramenta, são poucas se comparadas com as opções que a ferramenta *GraphViz* suporta. Essas opções foram escolhidas por serem as que mais modificam e melhoram a visualização do grafo e são de mais fácil entendimento, não requerendo que o usuário realize leituras na documentação da ferramenta *GraphViz* para entender as opções.

4.4.3 Comportamento dos grafos

Durante os testes realizados na ferramenta desenvolvida, foi constatado que alguns métodos de *layout* são melhores para a visualização de grafos de arquivos de XML. O método DOT garante uma boa estruturação e visualização do grafo, sendo o mais aconselhado para geração de grafos de documentos XML pequenos. Para grafos maiores, o método que apresentou melhores resultados foi o método FDP. Este método dá suporte a utilização de sub grafos e evita a sobreposição dos mesmos, o que garante que as informações não se misturem no grafo, facilitando a visualização. Os demais métodos, apesar de, para alguns casos, gerarem grafos de bom entendimento, por não darem suporte à sub grafos não é possível garantir que os grafos gerados serão de fácil visualização.

4.4.4 Comparação entre *XMLSpear*, *Liquid XML Studio* e *XMLGraph*

O propósito para a criação de uma ferramenta de visualização de documentos XML é o fato de que as atuais ferramentas, que trabalham com XML, não apresentam uma

visualização que facilite o entendimento do documento XML para usuários que não conhecem a linguagem.

Assim, analisando as ferramentas *XMLSpear* e *Liquid XML Studio*, que são editores de arquivos XML, é possível constatar que são ferramentas bastante completas que possuem métodos para criação e validação de documentos XML, porém, como dito anteriormente, a visualização das informações não é satisfatória.

Na ferramenta *freeware XMLSpear*, a única visualização disponível é da árvore gerada pelo arquivo XML, como pode ser visto no exemplo da figura 2.15. Para acessar o conteúdo nesta árvore é necessário que o usuário navegue pelos nodos e ache a informação, para isto o usuário necessita estar acostumado com este tipo de estrutura.

Na ferramenta proprietária *Liquid XML Studio*, além da visualização em formato de árvore, é possível visualizar a estrutura do documento XML, se o esquema XLS do XML for carregado na ferramenta. Entretanto, como pode ser visto na figura 2.16, esta visualização é um tanto confusa, pois conta com informações, sobre tipos de elementos e o tipo da informação armazenada (*int*, *float*, *string*, entre outros)em nodos folhas. Estas informações não são relevantes para usuários que não entendam sobre a linguagem XML Schema e apenas confundem a visualização. Esta visualização não mostra o conteúdo do arquivo XML, isto é, não é mostrado ao usuário as informações presentes nos elementos folhas do documento XML.

A ferramenta *XMLGraph* não é um editor para documentos XML, sendo assim, nela não é possível criar, editar e validar documentos XML. O foco da ferramenta é a visualização gráfica do documento XML. Nela é possível realizar a visualização, da estrutura juntamente com o conteúdo, do documento XML em formato de grafo. É possível gerar este grafo de diversas formas, cabendo ao usuário escolher a que mais lhe agradar. Caso o usuário queira, é possível salvar a imagem do grafo em formato PNG para ser utilizada por outras aplicações, o que não é possível nas ferramentas *XMLSpear* e *Liquid XML Studio*, a não ser que seja retirado um *print screen* da tela.

Os códigos fonte e a aplicação estão disponíveis em http://www.inf.ufsm.br/~mzanella/XMLGraph.

5 CONCLUSÃO E TRABALHOS FUTUROS

A necessidade da implementação de uma nova ferramenta para visualizar graficamente arquivos XML foi constatada através da análise dos programas existentes que trabalham com documentos XML. As ferramentas analisadas, *XMLSpear* e *Liquid XML Studio*, possuem métodos de visualização que são de difícil entendimento para usuários que não estão habituados à estruturas em formato de árvore ou que não possuem conhecimento da linguagem XML Schema. Outro fator é que não há como exportar estas visualizações para um arquivo de imagem e utilizar em outras aplicações. Após constatadas estas deficiências, fez-se uma proposta de uma nova ferramenta que suprisse tais necessidades.

Implementou-se, então, o *XMLGraph*, tendo como objetivos principais a visualização de arquivos XML em um formato de fácil entendimento, opções para o usuário alterar o formato de visualização e a possibilidade de salvar esta imagem em disco para ser utilizada em outras aplicações. Espera-se que estas vantagens sirvam de incentivo para a utilização do *XMLGraph* como ferramenta de visualização de arquivos XML.

Entretanto, ainda há o que ser feito. Sugere-se, para trabalhos futuros, que seja suportada a opção para edição e validação do arquivo XML de entrada, com o carregamento, se houver, da DTD ou XML Schema do arquivo XML. Com este recurso, não é necessário que o usuário recorra a outra ferramenta para editar ou validar o seu arquivo XML.

Outra melhoria a ser realizada diz respeito à utilização da *GraphViz* como ferramenta para desenhar o grafo. Ela apresenta suporte para inúmeras opções de personalização no grafo, muitas destas não compreendidas na ferramenta *XMLGraph*. Oferecer um suporte melhor para personalizações no grafo tornaria o *XMLGraph* uma ferramenta para criação de grafos altamente personalizados pelo próprio usuário.

Por último, outras melhorias a serem realizadas dizem respeito à criação de grafos menores que facilitariam o entendimento para os usuários da ferramenta. São elas:

- Oferecer ao usuário uma opção para delimitar o número de níveis hierárquicos que serão criados no grafo, diminuindo o tamanho tanto do grafo quanto do arquivo DOT gerado.
- Implementação da visualização apenas da estrutura do documento XML. A visualização de estrutura e de conteúdo no mesmo grafo pode gerar grafos com um tamanho muito grande, dificultando o entendimento. Caso o conteúdo do documento não seja de grande importância, a visualização apenas da sua estrutura geraria um grafo menor.
- Integrar à ferramenta *XMLGraph* a opção de consultas XQuery em documentos XML e gerar o grafo do resultado da consulta, assim seria possível para os usuários delimitar ainda mais a criação do grafo, retirando informações não relevantes.

REFERÊNCIAS

ALTOVA. **XMLSpy 2012**. Disponível em: http://www.altova.com/xml-editor/. Acesso em: setembro de 2011.

DONKEY DEVELOPMENT. **XMLSpear** Disponível em: http://www.donkeydevelopment.com/. Acesso em: setembro de 2011.

GraphViz. **GraphViz – Graph Visualization Software**. Disponível em http://www.GraphViz.org/. Acesso em agosto de 2011.

GRAPHVIZ JAVA API. Disponível em: http://www.loria.fr/~szathmar/off/projects/java/GraphVizAPI/index.php. Acesso em: outubro de 2011.

GRONER, L. **Construindo um DTD – Introdução ao XML** – Parte VI. Disponível em: http://www.loiane.com/2009/04/construindo-um-dtd-%E2%80%93-introducao-ao-xml-parte-vi/. Acesso em: agosto 2011.

LIQUID TECHNOLOGIES. **Liquid XML Studio 2011**. Disponível em: http://www.liquid-technologies.com/xml-studio.aspx. Acesso em: setembro de 2011.

McGRATH, Sean. **XML**: aplicações práticas. Tradução de Vitor Hugo da Paixão Alves. Rio de Janeiro: Campus, 1999. 368 p.

NETBEANS. Disponível em: http://netbeans.org. Acesso em: setembro de 2011.

RAY, T. E. **Learning XML**. Sebastopol: O'Reilly, 2003. 416 p.

VITTORI, C. **APIs XML – DOM** (Modelo de Objeto Documento). Disponível em : http://www.ufpa.br/sampaio/curso_de_sbd/semin_data_web/Transparencias/03-b-domshort.pdf. Acesso em agosto de 2011.

Wikipedia. **Document Object Model**. Disponível em http://en.wikipedia.org/wiki/Document_Object_Model. Acesso em agosto de 2011.

Wikipedia. **DOT Language**. Disponível em http://en.wikipedia.org/wiki/DOT_language. Acesso em agosto de 2011.

W3C. **Extensible Markup Language (XML)**. Disponível em http://www.w3.org/XML. Acesso em agosto de 2011.

W3C. **XML Schema**. Disponível em http://www.w3.org/XML/Schema. Acesso em agosto de 2011

W3Schools. **XML DOM Tutorial**. Disponível em: http://www.w3schools.com /dom/default.asp . Acesso em agosto de 2011

XML MARKER. Disponível em: http://symbolclick.com/. Acesso em: setembro de 2011.

XML NOTEPAD. Disponível em: http://xmlnotepad.codeplex.com/. Acesso em: setembro de 2011.