

Desenvolvimento de uma extensão para o navegador Firefox visando integração com o sistema *TagManager*

Antonio Pires de Almeida Junior¹

¹Departamento de Informática (DIN) – Universidade Estadual de Maringá (UEM)
Av. Colombo, 5790 – 87020-900 – Maringá – PR – Brasil

junioruhu@gmail.com

Abstract. *With the rapid expansion of the Web, brought about by the advent of Web 2.0, the manner in which information retrieval has been conducted has changed, especially with the advent of systems based on tagging. The TagManager system has the objective of managing personomies of a user in several systems based on tagging. This article discusses the development of an extension for the Firefox browser that provides a new way to interact with the system TagManager.*

Resumo. *Com a rápida expansão da Web, proporcionada pelo advento da Web 2.0, a maneira com que a recuperação de informação tem sido realizada sofreu alterações, principalmente com o surgimento de sistemas baseado em tagging. O sistema TagManager surgiu com o objetivo de gerenciar as personomias de um usuário em diversos sistemas baseado em tagging. Este artigo aborda o desenvolvimento de uma Extensão para o navegador Firefox que proporciona uma nova forma de interação com o sistema TagManager.*

1. Introdução

Com o aumento das informações disponíveis na *Web* tornou-se necessária a utilização de mecanismos de indexação de conteúdo (sistemas de busca) para auxiliar o usuário na recuperação de informação. Atualmente, esta indexação é realizada por sistemas como o *Google*¹, *Yahoo*², *Altavista*³, entre outros; os quais realizam, automaticamente, a indexação de conteúdo da *Web*, valendo-se de algoritmos complexos (DA SILVA, 2009).

O mais recente cenário da *Web* 2.0, formado por diversos sistemas que permitem uma maior interação do usuário, influencia na maneira como a recuperação de informações é realizada, visto que ele contribuiu para o aumento da quantidade de informação disponível na *Web*. O aumento dessa interação levou ao surgimento de novas aplicações marcadas pela colaboração, comunicação e interatividade, entre seus

¹ <http://www.google.com.br>

² <http://www.yahoo.com.br>

³ <http://www.altavista.com>

usuários de uma forma e em uma escala nunca antes observadas (PEREIRA; DA SILVA, 2008).

As aplicações baseadas em *tagging* são algumas das novas aplicações que surgiram após a *Web 2.0*. Elas permitem que seus usuários criem *tags* (metadados na forma de palavras-chave) e as atribua a recursos pessoais armazenados no sistema (CÔGO, 2008) denominando uma categorização. Um conjunto de categorizações utilizado por um usuário no sistema forma a sua *personomia* e um conjunto de *personomias*, quando compartilhadas por diversos usuários, formam o que é chamado de *folksonomia*.

Sistemas que são baseados em *tagging* permitem que os usuários realizem a categorização de um recurso por meio de uma ou mais *tags*. Os sistemas *Youtube*⁴, *Flickr*⁵ e *Delicious*⁶ podem ser considerados exemplos desses tipos de sistemas. Com a existência de vários sistemas baseados em *tagging* surgiu a necessidade da criação de um espaço único de nomes (um vocabulário), no qual o usuário poderia agrupar e gerenciar suas *tags*. O sistema denominado *TagManager* (TM) tem como objetivo o gerenciamento global dos termos utilizados como *tags* e também a realização de busca nos diversos sistemas baseados em *tagging* que empregam estas *tags*. Os autores do *TagManager* sugerem como trabalho futuro o desenvolvimento de um complemento⁷ (*plug-ins* ou *add-ons*) que intercepte a categorização do usuário, permitindo que as mudanças realizadas nos sistemas baseados em *tagging* sejam diretamente refletidas no TM, facilitando, assim, o trabalho de gerenciamento do vocabulário do usuário.

A utilização de um complemento é uma alternativa interessante para o usuário interagir com o sistema *TagManager*, tornando mais fácil o acesso aos seus recursos e permitindo, assim, uma maior interação. Esta pesquisa limita-se ao desenvolvimento de uma extensão para o navegador Firefox⁸ para o sistema *TagManager*, que permitirá ao usuário realizar categorizações no *Delicious*, sendo estas refletidas automaticamente ao sistema *TagManager*.

Este artigo está organizado da seguinte maneira: A seção 2 aborda a revisão bibliográfica. A seção 3 apresenta informações necessárias sobre o desenvolvimento de uma Extensão e apresenta também a Extensão para o *TagManager*. A seção 4 aborda as considerações finais e a 5 os trabalhos futuros.

2. Revisão Bibliográfica

Com a rápida expansão da *Web 2.0*, principalmente após a primeira conferência da *Web 2.0* em outubro de 2004 (O'REILLY, 2005), nunca se viu tanto conteúdo *on-line* disponível aos usuários. Este fato consolidou a utilização de mecanismos de indexação de conteúdo para auxiliar na recuperação de informação. Esses mecanismos utilizam algoritmos complexos para realizar a indexação de conteúdo. Uma alternativa no auxílio

⁴ <http://www.youtube.com>

⁵ <http://www.flickr.com>

⁶ <http://www.delicious.com>

⁷ Segundo a Fundação Mozilla, complemento (*add-ons*) são pequenos pedaços de programas que adicionam novas características ou funcionalidades ao programa.

⁸ <http://br.mozdev.org/>

na recuperação de informação é a utilização do processo de *tagging* (DA SILVA, 2009). Os sistemas baseados em *tagging* são amplamente utilizados para categorizar, de forma simples, qualquer tipo de conteúdo na *Web* (GONÇALVES; JUNIOR). Ou seja, ao invés de utilizar algoritmos complexos, o usuário interagirá com o sistema atribuindo *tags* (termos) que deseja aos objetos (recursos).

Categorizar, segundo o dicionário Aurélio de 1986 (FERREIRA, 1986), é a “representação de um objeto pelo pensamento, por meio de suas características gerais. Ação de formular uma idéia por meio de palavras, definição; caracterização”. A categorização, dependendo do ambiente e do contexto em que é utilizada, pode refletir na maneira em que uma informação é vista (LIMA, 2007). Quando a categorização passa a ser responsabilidade dos usuários, ou seja, os próprios usuários do sistema atribuem *tags* ao conteúdo presente e essas *tags* e os recursos utilizados nas categorizações são disponibilizados a todos os outros usuários do sistema (CÔGO, 2008), tem-se a categorização colaborativa.

A categorização colaborativa é mais conhecida como *folksonomia*, termo criado por Vander Wal em 2004 (MATHES, 2004). Segundo Pereira e da Silva (2007), o termo *folksonomia* passou a ser utilizado para representar a técnica na qual as pessoas utilizam *tags* para a organização de informação na *Web*, categorizando *URLs*, fotos, postagens de *blogs* ou qualquer outro objeto passível de ser referenciado. Nela existe o fator social do compartilhamento e da interação entre os usuários (PEREIRA; DA SILVA, 2007). Um aspecto importante da *folksonomia* é o fato de ela ser composta de diversas *tags* em um espaço livre. Ou seja, não existe uma hierarquia, nem relação entre as *tags* (MATHES, 2004). Dependendo do tipo de *folksonomia* o usuário pode categorizar, organizar e visualizar, além de seus próprios recursos, os de outros usuários.

Alguns críticos como Vander Wal⁹ e Shirky¹⁰ levantaram questões sobre duas formas de *folksonomia*, a estreita e a larga. A **folksonomia estreita** permite que um único usuário atribua *tags* a recursos próprios, as quais podem ser utilizadas por outros usuários para recuperação do recurso armazenado. Enquanto a **folksonomia larga** permite que muitos usuários atribuam *tags* a um mesmo recurso armazenado no sistema, podendo este ser recuperado por qualquer uma das *tags* atribuídas. Observa-se que o usuário compartilhará suas *tags*, seus recursos e suas caracterizações no sistema. Esses dados no sistema formarão sua *personomia*. Deste forma, conclui-se que a *folksonomia* pode ser compreendida como uma coleção de *personomia* compartilhada e interligada.

Segundo DA SILVA (2009) “Uma *personomia* pode ser vista como a relação composta por um usuário, que utiliza seu vocabulário pessoal como *tags* para marcar os recursos (objetos) de seu interesse”. A *personomia* do usuário no sistema é criada para a organização da informação, porém, um usuário pode utilizar vários sistemas e em cada um deles possuir uma *personomia*. Diante deste fato surgiu à necessidade de haver uma maneira do usuário gerenciar todas suas *personomia*, ou seja, seu vocabulário. O objetivo do sistema *TagManager* é o gerenciamento global dos termos utilizados como *tags* e permitir ao usuário realizar busca nos diversos sistemas baseados em *tagging*. Ele

⁹ <http://www.vanderwal.net>

¹⁰ <http://shirky.com>

é um sistema *Web* que, até o momento, apresenta uma única forma de interação (i.e. por meio de uma página *Web*) entre o usuário e os sistemas baseados em *tagging*. Durante o seu desenvolvimento foi levantado o problema de comunicação entre o *TagManager* e os sistemas baseados em *tagging*, esse problema consiste na maneira em que as modificações realizadas nesses sistemas serão refletidas no *TagManager*, então, visando resolver o problema citado e oferecer uma nova forma de interação, os autores do sistema *TagManager* sugerem como trabalho futuro o desenvolvimento de um *complemento* para os navegadores da *Web* na forma de uma Extensão que intercepte o uso dos sistemas baseados em *tagging* cadastrados e comunique as mudanças efetuadas ao *TagManager*.

2.1 Linguagem XUL

A *XML User Interface Language (XUL)*, em português, linguagem *XML* de interface de usuário, é uma linguagem proprietária desenvolvida para suportar aplicações do Projeto Mozilla. Ela possui todas as características disponíveis para *XML*, podendo ser considerada o futuro das interfaces gráficas, popularmente chamadas de *GUI (Graphical User Interface)*. Por ser desenvolvida pelo Projeto Mozilla, se faz necessário utilizar o mesmo para poder executá-la. A programação em *XUL* separa a interface do usuário em quatro camadas (MOZILLA, 2007) (ALEXANDRE, 2005), sendo elas:

- **Conteúdo:** na qual são descritos os elementos da interface em um arquivo de extensão *XUL*;
- **Aparência:** na qual a apresentação é descrita em um arquivo de extensão *css* e em arquivos de imagens (*GIF*, *JPG* e *PNG*)
- **Comportamento:** na qual é descrita a lógica que atua na interface em um arquivo *Javascript* (extensão *js* e *xbl*);
- **Local:** a qual apresenta as informações necessárias para o funcionamento correto do idioma (linguagem natural) em que foi desenvolvida a aplicação.

A estrutura básica de um arquivo *XUL* para criar uma Janela pode ser vista na Figura 1.

```
1 <?xml version="1.0"?>
2 <?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
3 <window
4     id="Teste"
5     title="Titulo"
6     orient="horizontal"
7     xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
8     ...
9 </window>
```

Figura 1. Estrutura básica de uma janela em XUL

Realizando uma análise linha a linha da estrutura básica tem-se:

- Linha 1: Declara que é um arquivo *XML*, normalmente, será adicionada no topo de cada arquivo *XUL*;

- Linha 2: Especifica o arquivo *CSS* que será utilizado pelo arquivo, utiliza-se a mesma sintaxe para importar folhas de estilo em *XML*;
- Linha 3: Declara que está descrevendo uma janela (*window*). Cada janela é descrita em um arquivo separado, neste exemplo quatro atributos são adicionados a *tag*;
- Linha 4: O atributo *id* geralmente é utilizado em todos os elementos, pois é utilizado com um identificador por *scripts* para ser referenciado.
- Linha 5: O atributo *title* descreve o texto que aparecerá na barra de título da janela.
- Linha 6: O atributo *orient* especifica o posicionamento dos elementos na janela. Pode-se utilizar o valor **vertical** para informar que os elementos serão exibidos em forma de colunas ou utilizar o valor **horizontal** para informar que os elementos serão exibidos horizontalmente na janela.
- Linha 7: Declara o *namespace* para *XUL*, indicando que todos os outros elementos são filhos *XUL*;
- Linha 8: Nesse espaço pode ser adicionado qualquer elemento (botão, menus, etiqueta, imagens e etc);
- Linha 9: Fechamento da *tag window*, sinalizando o final do arquivo.

É possível adicionar diversos tipos de elementos na janela, porém neste artigo serão abordados apenas os elementos básicos ou considerados importantes para elaboração da interface da Extensão em questão, outros elementos podem ser encontrados em MOZZILA (2007).

2.1.1 Principais Elementos Utilizados

Diversos elementos gráficos, encontrados nas principais interfaces gráficas, podem ser adicionados em uma janela *XUL*, porém nesta seção serão apresentados somente os principais elementos utilizados no desenvolvimento da Extensão em questão, sendo estes: os Botões, os Textos, as Imagens, os Controles de Entrada, as Caixas de Checagem, os Botões Radio, os Painéis com Abas e as Caixas.

Botões

Para adicionar um simples botão, geralmente, utiliza-se o elemento *button*. Esse elemento botão possui três propriedades principais, além de propriedades comuns: uma de etiqueta (*label*), uma de imagem (*image*) e uma de ação (*oncommand*). Elas podem ser usadas simultaneamente. A Figura 2 mostra a sintaxe e, a Figura 3, o código compilado.

```

1 <button
2     id="identifier"
3     class="dialog"
4     label="Button"
5     image="images/image.jpg"
6     default="true"
7     disabled="false"
8     accesskey="t"
9     oncommand="alert('Botão Clicado');"
10 />

```

Figura 2. Estrutura de um elemento *button*



Figura 3. Elemento *button* em execução

Texto

Não é possível inserir texto diretamente no *XUL*, é necessário inseri-los utilizando *tags*. Para isso, utilizam-se as *tags description* e *label*. A diferença entre elas é que a *label* permite atribuir um controle que informará a qual elemento ele pertence. A Figura 4 mostra a sintaxe das *tags* e a Figura 5 mostra o código compilado.

```

1 <description>
2     O texto deve ser inserido aqui!
3 </description>
4
5 <label value="Este label está relacionado ao botão a baixo" control="button"/>
6 <button id="button" label="Button"/>

```

Figura 4. Estrutura do elemento texto – *description* e *label*

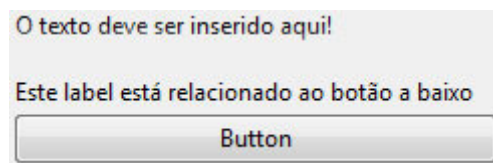


Figura 5. Elemento Texto – *description* e *label* em execução

Imagens

O *XUL* possui especificamente o elemento *image* para adicionar imagens. Esse elemento possui o atributo *src* e *id* além de atributos de estilo. A localização do endereço deve ser atribuída no *src*, entretanto, com a utilização do *id*, pode-se adicionar a localização da imagem na folha de estilo. A Figura 6 mostra a sintaxe desse elemento, nas duas situações citada.

```

1 <image src="images/exemplo.jpg"/>
2
3 <image id="imagem_exemplo"/>
4
5
6 Exemplo da folha de estilo:
7   #imagem_exemplo {
8     list-style-image="chrome://tagmaneger/skin/imagem.jpg";
9   }

```

Figura 6. Estrutura do elemento *image*

Caixas de Entrada de Texto

O *XUL* possui o elemento *textbox* para controlar a entrada de texto introduzida pelo usuário. O elemento citado possui diversos atributos que moldam seu funcionamento, permitindo ao usuário definir a quantidade de caracteres e, de linhas, se é um campo de digitação privada, etc. A Figura 7 mostra a sintaxe do elemento com diversas configurações de atributos e a Figura 8 mostra o código compilado.

```

1 <label control="somente-texto" value="Somente texto, sem configuração:"/>
2 <textbox id="somente-texto"/>
3
4 <label control="somente-password" value="Campo de senha, com no maximo 8 caracteres"/>
5 <textbox id="somente-password" type="password" maxlength="8"/>
6
7 <label control="somente-password" value="Texto Multi-linha"/>
8 <textbox multiline="true" value="Qual quer texto pode ser digitado aqui."/>

```

Figura 7. Estrutura do elemento *textbox* em diversas variações de atributos

Figura 8. Elemento *textbox*, diversos atributos, em execução

Caixas de Checagem

O elemento *checkbox* permite que uma caixa de checagem seja criada. Ela é utilizada para opções que podem ser habilitadas ou inabilitadas. A Figura 9 mostra a sintaxe do elemento e a Figura 10 mostra o código compilado.

```

1 <checkbox id="teste-checkbox" checked="true" label="Testando o checkbox"/>
2
3 <checkbox id="teste-checkbox2" checked="false" label="Testando o checkbox 2"/>

```

Figura 9. Estrutura do elemento *checkbox* em duas diversas variações de atributos

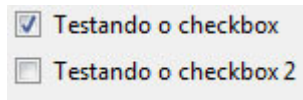


Figura 10. Elemento *checkbox* em execução

Botões Radio

O elemento *radio* diferencia-se do *checkbox* por permitir que apenas uma opção possa ser selecionada. Com a finalidade de agrupar vários botões *radio* é necessário utilizar o elemento *radiogroup*. A Figura 10 mostra a sintaxe do elemento e a Figura 11 mostra o código compilado.

```
1 <radiogroup>
2   <radio id="santos" label="Santos"/>
3   <radio id="corinthians" label="Corinthians" selected="true"/>
4   <radio id="palmeiras" label="Palmeiras"/>
5 </radiogroup>
```

Figura 11. Estrutura do elemento *radiogroup*, com o radio com *label* Corinthians sendo selecionado como padrão

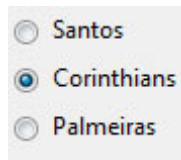


Figura 12. Elemento *radiogroup* em execução, somente um poderá ser selecionado

Caixas

A maneira mais comum de desenvolver uma interface *XUL* é dividindo a janela em caixas. Os elementos inseridos dentro de uma caixa poderão ser ajustados horizontalmente e verticalmente. A Figura 13 mostra a estrutura de uma caixa horizontal e uma caixa vertical. A Figura 14 apresenta o código compilado. Os elementos que forem filhos de uma caixa horizontal serão organizados em uma linha e os elementos que forem filhos de uma caixa vertical serão organizados em coluna.

```
1 <hbox>
2   <button label="Botão Horizontal 1"/>
3   <button label="Botão Horizontal 2"/>
4 </hbox>
5
6 <vbox>
7   <button label="Botão Vertical 1" />
8   <button label="Botão Vertical 2" />
9 </vbox>
```

Figura 13. Estrutura básica do elemento *hbox* e *vbox*

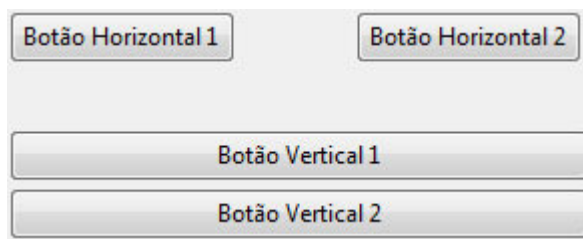


Figura 14. Elemento *hbox* e *vbox* em execução

Essa estrutura torna-se mais eficiente com a utilização de outras caixas, espaçadores, elementos que possuam atributos *flex*¹¹ e algumas propriedades de estilo *HTML* (*width* e *height*). A Figura 15 mostra a estrutura de uma caixa mais complexa e a Figura 16 o código compilado.

```

1 <vbox flex="1">
2
3   <description>
4     Campo de teste
5   </description>
6
7   <hbox>
8     <label value="Digite o texto:" control="text"/>
9     <textbox id="text"/>
10  </hbox>
11
12  <hbox>
13    <spacer flex="1"/>
14    <button id="texto-button" label="OK" default="true" style="min-width: 100px;"/>
15    <button id="cancel-button" label="Cancel"/>
16  </hbox>
17
18 </vbox>

```

Figura 15. Estrutura elemento *hbox* e *vbox* mais elaborada

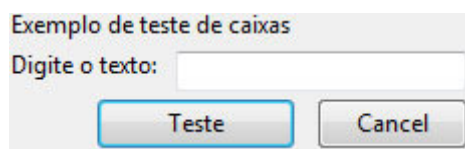


Figura 16. Elemento *hbox* e *vbox*, em uma estrutura mais elaborada, em execução

Caixas com Abas

Caixas com abas são utilizadas quando existe a intenção de expor mais opções do que comportaria uma só tela. Para criar uma caixa com abas necessita-se de cinco elementos, sendo eles:

- *tabbox*: A caixa principal que conterá as abas;
- *tabs*: A caixa onde as abas individuais se encontram;

¹¹ Quando o atributo *flex* for maior que 0, os elementos crescem ou encolhem para se adequarem ao espaço disponível na janela.

- *tab*: Uma aba específica;
- *tabpanel*: O *container* para as páginas.
- *tabpanel*: O corpo de uma simples página.

A Figura 17 mostra a estrutura de um painel com abas e a Figura 18 mostra o código compilado.

```

1 <tabbox>
2   <tabs>
3     <tab label="Aba 1"/>
4     <tab label="Aba 2"/>
5   </tabs>
6   <tabpanel>
7     <tabpanel id="aba1">
8       <description>Conteudo aba 1</description>
9     </tabpanel>
10    <tabpanel id="aba2">
11      <description>Conteudo aba 2</description>
12    </tabpanel>
13  </tabpanel>
14 </tabbox>

```

Figura 17. Estrutura de uma Caixa com Abas



Figura 18. Caixas com Abas em execução

2.2 Javascript

Javascript é uma linguagem de programação que foi desenvolvida pela *Netscape*. Inicialmente ela foi chamada de *LiveScript*, porém após uma parceria com *Sun Microsystems* o projeto foi renomeado em 1995 (MOZILLA, 2009) (WIKIPÉDIA,2010). Ela foi criada inicialmente para realizar pequenos processamentos no modo cliente, ou seja, a execução de programas simples, tais como validação de formulários que fossem executados no computador do usuário sem a necessidade de troca de informações com o servidor. Segundo Galdino (2006), a linguagem *Javascript* foi criada para:

- alterar valores de elementos *HTML*;
- criar elementos *HTML*; e
- validar formulários e controlar interação entre molduras.

Após o advento da tecnologia *Ajax* e da *Web 2.0*, ela passou a ser utilizada não somente para aplicações banais. O objetivo desta seção não é o de ensinar conceitos básicos da linguagem *JavaScript*, mas sim abordar conceitos importantes para o

desenvolvimento da Extensão em questão (como a utilização de *DOM* e de *XMLHttpRequest*), pois o *Javascript* é responsável em realizar as ações na linguagem *XUL*. Descrições de conceitos básicos podem ser encontrado em GALDINO (2006) e MOZILLA (2010 A).

2.2.1 Document Object Model (DOM)

O *Document Object Model (DOM)* é uma *API* para documentos *HTML* e *XML*. Ele fornece uma representação estrutural do documento, permitindo alterar seu conteúdo e apresentação visual. Essencialmente, ela liga a página *Web* a *scripts* ou linguagens de programação. O *DOM* não é uma linguagem de programação, porém, sem ele a linguagem *Javascript* não poderia acessar o documento e seus elementos. No início, a *Javascript* e o *DOM* eram fortemente ligados, mas eles evoluíram separadamente. O conteúdo de uma página é armazenado em forma de árvore no *DOM* e pode ser acessado e manipulados via *Javascript*. Para acessar os elementos de uma página basta inserir um *script* ou criá-lo na página *Web* que todas as funcionalidades do *DOM* passam a ser acessíveis. Abaixo será apresentada a estrutura de nós e suas principais utilizações.

Nós

No *DOM* cada elemento, seja ele qual for, é considerado um nó. Quando este nó possuir em sua estrutura um outro elemento este será considerado um nó filho. A Figura 19 mostra essa situação.

```
1 <P>Isto é um <B id="negrito1">teste</B></P>
```

Figura 19. Exemplo de um nó

Analisando a Figura 19, observa-se que a árvore formada pelo o elemento parágrafo “*p*” possuirá dois filhos, um com texto “*Isso é um*” e outro com a *tag* negrito “*b*”, sendo que está possuirá o filho “*teste*”.

Navegando entre Nós

É necessário conhecer a estrutura exata de uma árvore *DOM* para poder acessar e manipular seus elementos. Tomando ainda o exemplo da Figura 19, supondo que ele estivesse armazenado em uma variável *X* e se desejasse acessar seu conteúdo. Para tal seria necessários utilizar o código *x.parentNode*, porém, caso se quisesse acessar o valor de seus filhos se utilizaria *x.childNodes[Y]*, onde o “*Y*” seria o número do filho que desejaria-se recuperar, sendo este sempre começado de 0. No exemplo da Figura 19, caso se utilize *x.childNodes[0]*, o valor “*Isso é um*” será retornado.

Existe ainda o código *x.firstChild* e *x.lastChild* que retornam, respectivamente, o primeiro filho e o último filho da árvore.

Obtendo um elemento

Muitas vezes percorrer um documento inteiro em busca de um elemento não é uma tarefa fácil, pois é necessário conhecer em detalhe a estrutura da árvore *DOM*, sendo que esta pode sofrer alterações e gerar problemas. Portanto para facilitar a busca de um elemento existe a possibilidade de saltar diretamente para ele utilizando a busca pelo *TagName* ou pelo seu *ID*.

Retornando ainda à Figura 19, é possível recuperar o valor que está em negrito utilizando o comando `document.getElementsByTagName("B")[0]` ou `document.getElementById("negrito1")` atribuindo essa expressão a uma variável.

Alterando um nó

Com o *DOM* é possível alterar o valor e/ou um atributo de um nó. Ainda utilizando o exemplo da Figura 19, deseja-se alinhar o texto para esquerda e mudar o valor que está dentro da *tag* negrito "B". Para alterar o valor utiliza-se o comando `document.getElementById("negrito1").firstChild.nodeValue = "teste 2"` e para realizar o alinhamento utiliza-se `document.getElementsByTagName("p").setAttribute("align", "left")`.

2.2.2 XMLHttpRequest

O *XMLHttpRequest* é um objeto *Javascript* que foi desenvolvido pela *Microsoft*, porém, foi adotado pelo *Mozilla* e também passou por uma padronização pela *W3C* (*MOZILLA*,2010 B). Ele implementa uma interface exposta por um motor de *script* que permite executar *scripts* com funcionalidades *HTTP* cliente, tais como a apresentação de formulário de dados ou carregar dados de um servidor. Ele pode ser utilizado para requisições e respostas *HTTP* e *HTTPS* (*W3C*, 2010).

No desenvolvimento da Extensão discutida neste artigo, o *XMLHttpRequest* é utilizado para fazer requisição ao servidor *Delicious* para o qual a integração será realizada. A Figura 20 mostra a sintaxe utilizada para que as requisições possam ser realizadas.

```
1 var req = new XMLHttpRequest();
2 req.open('GET', 'http://www.mozilla.org/', true);
3 req.onreadystatechange = function (aEvt) {
4     if (req.readyState == 4) {
5         if (req.status == 200)
6             dump(req.responseText);
7         else
8             dump("Error loading page\n");
9     }
10 };
11 req.send(null);
```

Figura 20. Sintaxe utilizada para requisições assíncronas

3. Complemento - Extensão

Um complemento (*add-on* – Extensão, *plugin*, *template* e etc) para o *Firefox* é uma coleção de arquivos e pastas que foram compactados em um arquivo com uma Extensão **xpi**. O arquivo **xpi** é nada mais do que um **zip** que foi renomeado. Complementos para o *Firefox* são independentes de plataforma. Isso significa que uma única codificação pode ser utilizada em qualquer sistema operacional (*Windows*, *Mac OS X* e *Linux*).

Recordando que, a proposta, do presente trabalho, consiste no desenvolvimento de uma Extensão para o navegador *Firefox*. Vale ressaltar a diferença entre Extensão e *Plugin* para navegador *web Firefox*. O *Plugin* adiciona recursos que podem ser

utilizados pela página *Web*, enquanto a Extensão é instalada especificamente para o Firefox para criar ou alterar alguma funcionalidade. Para o desenvolvimento da extensão é necessário conhecer a Linguagem *XUL* e *Javascript*.

3. 1. Configuração de uma Extensão

Para desenvolver uma Extensão para o *Firefox* é necessário conhecer a estrutura básica que o forma, sendo estas pastas e arquivos, demonstrados na Figura 21 e 22. O *software Extension Wizard*¹² permite ao desenvolvedor, criar de forma rápida um esqueleto de uma Extensão.

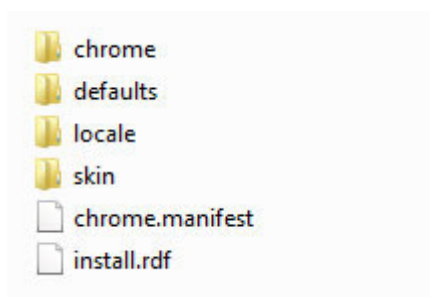


Figura 21. Estrutura de uma Extensão

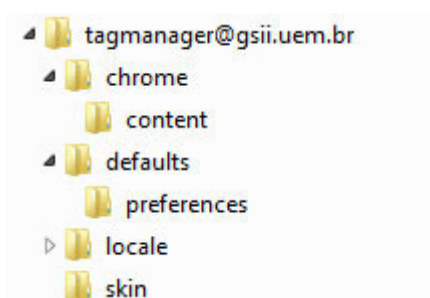


Figura 22. Estrutura de uma Extensão

Os arquivos mostrados na Figura 21, são arquivos de configuração que fornecem informações para o *Firefox*. Cada um deles será descrito a seguir.

O *Install Manifest* é um arquivo texto criado com o nome “*install.rdf*”. Este arquivo é utilizado para fornecer informações sobre a Extensão para o Firefox no momento em que a instalação ocorre. O arquivo contém metadados que identifica a Extensão proporcionando informações sobre quem o criou, onde encontrar mais informações, quais as aplicações e versões são compatíveis. A Figura 23 mostra a sintaxe referente ao “*install.rdf*” da Extensão proposta.

¹² <http://ted.mielczarek.org/code/mozilla/extensionwiz/>

```

1 <?xml version="1.0"?>
2 <RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3   xmlns:em="http://www.mozilla.org/2004/em-rdf#">
4
5   <Description about="urn:mozilla:install-manifest">
6     <em:id>tagmanager@gsii.uem.br</em:id>
7     <em:version>1.0</em:version>
8     <em:type>2</em:type>
9
10    <!-- Target Application this extension can install into,
11         with minimum and maximum supported versions. -->
12    <em:targetApplication>
13      <Description>
14        <em:id>[ec8030f7-c20a-464f-9b0e-13a3a9e97384]</em:id>
15        <em:minVersion>2.0</em:minVersion>
16        <em:maxVersion>3.5.3</em:maxVersion>
17      </Description>
18    </em:targetApplication>
19
20    <!-- Front End MetaData -->
21    <em:name>Tag Manager</em:name>
22    <em:description>Extensão do Tag Manager</em:description>
23    <em:creator>GSII - Grupo de Sistemas Interativos Inteligentes</em:creator>
24    <em:homepageURL>http://www.din.uem.br/gsii/</em:homepageURL>
25    <em:optionsURL>chrome://tagmanager/content/options.xul</em:optionsURL>
26  </Description>
27 </RDF>

```

Figura 23 – Estrutura do *Install Manifest* (install.rdf)

Neste arquivo é necessário modificar as informações essenciais. Observando a primeira seção (linhas 6, 7 e 8) tem-se:

- *id*: Este é o valor que identifica a Extensão. A identificação precisa estar no formato de um endereço de *e-mail*, no entanto, ela não precisa ser um *e-mail* válido. É possível utilizar um *GUID*, mas isso não é necessário ou recomendado;
- *version*: Este texto identifica a versão da Extensão que está sendo instalada. Recomenda-se utilizar valores menores que 1.0 para aplicações em desenvolvimento;
- *type*: Este é um valor inteiro que representa o tipo do complemento a ser instalado. No caso de um complemento o valor deve ser sempre 2 ou, caso o desenvolvimento seja para um tema do *Firefox*, o valor deve ser sempre 4.

A seção “*targetApplication*” (linhas 12 à 18) define como a aplicação é orientada. Os campos:

- *id*: define para qual aplicação o *add-on* é desenvolvido, neste caso, está definido para *Firefox*;
- *minVersion*: define a versão mínima do *Firefox* para a Extensão funcionar;
- *maxVersion*: define a versão máxima do *Firefox* para a Extensão funcionar.

E, por fim, a última seção (linhas 21 à 25) descreve a Extensão. Tem-se:

- *name*: Esta linha simplesmente define o nome da Extensão e é destinada para exibição na interface do *Firefox*.
- *description*: Esta linha deve descrever a funcionalidade da Extensão e destina-se a ser exibido na interface do usuário. Deve caber em uma linha, a fim de mostrar-se, na íntegra, na interface do usuário.
- *creator*: Esta linha deve conter o nome de quem desenvolveu e será usado para exibir seu nome na interface do usuário.
- *homepageURL*: Esta linha é opcional, sendo apenas um *link* para a página do autor.

Antes de abordar o *Chrome Manifest* é necessário entender o significado de *Chrome*. Ele é o termo utilizado para referenciar o pacote de interface de usuário criado para o *Firefox*. Ele utiliza *URLs* exatamente como é usado na *Web (HTTP)*. No entanto, para que o *Firefox* interprete corretamente que ele está trabalhando com um pacote *chrome*, o prefixo “*chrome*” deve ser utilizado. O *Chrome* do *Firefox* está presente na janela do aplicativo localizado fora da área da janela de conteúdo. Como exemplo pode-se citar: barra de ferramentas, barras de menu, barra de título do navegador *Web* e etc.

O *Chrome Manifest* é um arquivo texto criado com o nome “*chrome.manifest*” que possui informações para que o *Firefox* possa encontrar o conteúdo necessário para exibir e executar a Extensão. A figura 24 mostra a sintaxe do “*chrome.manifest*” da Extensão proposta.

```

1 content    tagmanager  chrome/content/
2 content    tagmanager  chrome/content/ contentaccessible=yes
3 overlay chrome://browser/content/browser.xul chrome://tagmanager/content/browser.xul
4
5 locale    tagmanagerUS  en-US  locale/en-US/
6 locale    tagmanagerBR  pt-BR  locale/pt-BR/
7
8 skin      tagmanager  classic/1.0 skin/
9 style     chrome://global/content/customizeToolbar.xul  chrome://tagmanager/skin/skin.css

```

Figura 24 – Estrutura do *Chrome Manifest* (*chrome.manifest*)

Analisando o arquivo observa-se 5 tipos de pacotes: *content*, *locale*, *skin*, *overlay* e *style*.

O pacote *content* registra a localização dos arquivos da extensão, ele possui a seguinte estrutura:

1. nome do pacote *chrome* (*tagmanager*) ;
2. localização dos arquivos do pacote *chrome* (*chrome/content/*);

O pacote *Locale* é responsável por informar as localizações dos arquivos de idioma; o que permite ligá-lo em um pacote *chrome* diferente para traduzir um aplicativo sem alterar o resto do código fonte. Ele possui a seguinte estrutura:

1. nome do estilo de idioma(*tagmanagerUS*);
2. idioma a ser utilizado (*en-US*);
3. localização dos arquivos de idioma (*locale/en-us*).

Um pacote *Skin* é responsável por proporcionar um conjunto de arquivos que descrevem a aparência visual do *chrome*.

Um *overlay* (*XUL overlays* ou sobreposições *XUL*) permite que ocorra uma sobreposição entre arquivos, na Figura 24 observa-se que o Firefox recebe a instrução para sobrepor o “*browser.xul*” do Browser com o “*browser.xul*” da Extensão. O *overlay* possui a seguinte forma de registro:

1. localização do arquivo do *Firefox*;
2. localização do arquivo da Extensão.

Por fim, um *style* (*Style overlays* ou sobreposições de estilo) permite que um CSS customizado seja aplicado à página *chrome*. Ele possui, praticamente, a mesma forma de registro do *overlay*.

3. 2. A Pasta Chrome

No desenvolvimento de uma Extensão, a pasta *Chrome* é responsável por armazenar os arquivos específicos da Extensão, tais como arquivo de interface *XUL* e arquivos *Javascript*. Entretanto, também existirá um arquivo de configuração chamado “*browser.xul*” que possuirá a função de *overlay*, substituindo a aparência padrão do navegador Firefox. Nesse arquivo, por exemplo, é possível adicionar códigos que possibilitam adicionar um botão à barra de ferramentas, um item ao menu, um ícone na barra de *status* e etc. A Figura 25 mostra a sintaxe do arquivo do presente projeto. Analisando a sintaxe observam-se as seguintes informações relevantes:

- O *overlay* inicia-se na linha 4;
- Na linha 5 um arquivo *Javascript* é adicionado ao arquivo;
- Da linha 7 a 13 teclas de atalho são definidas pelo *keyset*, onde o desenvolvedor escolherá o *id* a utilizar, as teclas modificadoras (**alt**, **ctrl**, **Shift** e etc), a tecla de atalho e o comando a ser realizado pelo atalho;
- Da linha 15 a 18 dois itens são adicionados ao *menu* Ferramentas do *Firefox*;
- Da linha 21 a 30 o *menu TagManager* é adicionado na barra de *menu*, sendo posicionado na mesma logo após o *menu* ajuda;
- Da linha 32 a 35 um ícone é adicionado à barra de *status* do *Firefox*;
- Da linha 37 a 40 um ícone é adicionado na barra de ferramentas do *Firefox*;

```

1 <?xml version="1.0"?>
2 <?xml-stylesheet href="chrome://tagmanager/skin/skin.css" type="text/css"?>
3 <!DOCTYPE tagmanager SYSTEM "chrome://tagmanagerUS/locale/translations.dtd">
4 <overlay id="sample" xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
5   <script src="tagmanager.js" />
6
7   <keyset>
8     <key id="tag-manager-run-key" modifiers="accel" key="D" oncommand="tagmanager.run()"/>
9     <key id="tag-manager-run-logof" modifiers="accel,shift" key="P" oncommand="tagmanager.logof()"/>
10    <key id="addBookmarkAsKb" key="U" command="Browser:AddBookmarkAs" modifiers="accel,shift"/>
11    <key id="tag-manager-run-flickr" modifiers="accel,shift" key="F" oncommand="tagmanager.logof()"/>
12    <key id="tag-manager-run-youtube" modifiers="accel,shift" key="Y" oncommand="tagmanager.logof()"/>
13  </keyset>
14
15  <menupopup id="menu_ToolsPopup">
16    <menuitem label="&runtagmanager;" key="tag-manager-run-key" oncommand="tagmanager.run()"/>
17    <menuitem label="&logof&tagmanager;" key="tag-manager-run-logof" oncommand="tagmanager.logof()"/>
18  </menupopup>
19
20
21  <menubar id="main-menubar">
22    <menu id="tag_manager-menu" label="TagManager" insertafter="helpMenu" >
23      <menupopup id="dpopup" onpopupshowing="" >
24        <menuitem label="Run Tag Manager" modifiers="accel,shift" key="D" oncommand="tagmanager.run()"/>
25        <menuitem label="Add Flickr" key="tag-manager-run-flickr" oncommand="tagmanager.run()"/>
26        <menuitem label="Add Youtube" key="tag-manager-run-youtube" oncommand="tagmanager.run()"/>
27        <menuitem label="Logoff Tag Manager" key="tag-manager-run-logof" oncommand="tagmanager.run()"/>
28      </menupopup>
29    </menu>
30  </menubar>
31
32  <statusbar id="status-bar">
33    <statusbarpanel id="tag-manager-status-bar-icon" class="statusbarpanel-iconic"
34      src="chrome://tagmanager/skin/status-bar.png" tooltip="&runtagmanager;" onclick="tagmanager.run()" />
35  </statusbar>
36
37  <toolbarpalette id="BrowserToolbarPalette">
38    <toolbarbutton id="tag-manager-toolbar-button" label="Tag Manager" tooltip="&runtagmanager;"
39      oncommand="tagmanager.run()"/>
40  </toolbarpalette>
41 </overlay>

```

Figura 25 – Estrutura do browser.xul da extensão *TagManager*

3. 3. A Pasta *Default*

A pasta *Default* é utilizada quando se tem a intenção de adicionar preferências de usuário, automaticamente, ao *Firefox* durante o processo de instalação do complemento. Para que isso funcione perfeitamente é necessário criar uma subpasta “*preferences*” e adicionar um arquivo *Javascript(js)* com as configurações. A Figura 26 mostra o arquivo *js* do presente complemento. Neste caso, tem-se a intenção de gravar Usuário e a Senha do usuário para acessar o *Delicious* e o *TagManager*.

```

3 pref("extensions.tagmanager.usuarioDel", '');
4 pref("extensions.tagmanager.senhaDel", '');
5 pref("extensions.tagmanager.usuarioTag", '');
6 pref("extensions.tagmanager.senhaTag", '');

```

Figura 26 – Modelo de *preferences* do complemento *TagManager*

3. 4. A Pasta *Locale* e a *Skin*

Na pasta *Locale* encontram-se os arquivos que servirão para modificar a linguagem do complemento. É necessário criar uma pasta para cada idioma; elas possuirão arquivos

com os mesmos nomes, porém mudando apenas os valores das variáveis, conforme mostram as Figuras 27 e 28.

```
1 <!ENTITY runtanager "Run Tag Manager">
2 <!ENTITY logoftanager "Logoff Tag Manager">
```

Figura 27 – Arquivo translations.dtd da pasta en-US

```
1 <!ENTITY runtanager "Executar Tag Manager">
2 <!ENTITY logoftanager "Sair Tag Manager">
```

Figura 28 – Arquivo translations.dtd da pasta pt-BR

Como já foi dito, é necessário adicionar a referência das pastas de idioma no arquivo “*chrome.manifest*” e também definir no cabeçalho do arquivo *XUL* qual idioma será utilizado.

A pasta *Skin* irá conter os arquivos de imagens e folhas de estilos para serem utilizados na configuração do *layout* desenvolvido em *XUL*;

3. 5. A Extensão *TagManager*

Os autores do *TagManager* sugeriram como trabalho futuro o desenvolvimento de uma Extensão que tenha como objetivo a sincronização da *personomia* do usuário em diversos sistemas baseados em *tags* (*Delicious*, *Youtube*, *Flicker*) com o *TagManager*. Ou seja, qualquer atualização da *personomia* do usuário em um dos sistemas deve ser refletido automaticamente no *TagManager*. A Extensão possuirá a seguinte arquitetura, mostrada na Figura 29, lembrando que nesse trabalho somente será feita a integração com o *Delicious*. A arquitetura completa foi elaborada já pensando em trabalhos futuros.

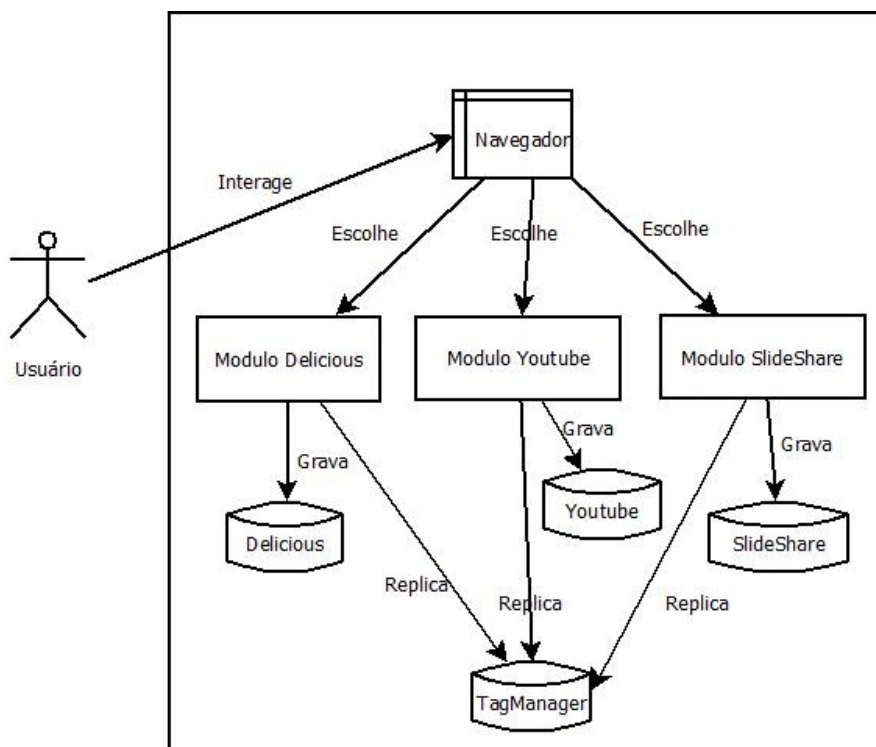


Figura 29 – Arquitetura da Extensão do TagManager

Entendendo a arquitetura da Extensão: O usuário interagirá com o navegador Firefox expressando qual atividade deseja realizar - adicionar um vídeo ao *Youtube* ou um endereço favorito ao *Delicious* ou uma apresentação no *Slideshare*. Após a interação o módulo responsável por realizar a tarefa escolhida será acionado, solicitando mais informações para que a tarefa seja finalizada. Ao terminar a tarefa os dados serão enviados para o sistema original e replicados ao *TagManager*.

O módulo *Delicious* funciona da seguinte maneira: O usuário interage com o navegador Firefox expressando que deseja adicionar o endereço acessado à sua conta no *Delicious*. Em um uso normal, caso o usuário utilize a Extensão do próprio *Delicious*, uma janela para categorização será aberta, porém, por utilizar a Extensão do *TagManager* será aberta uma janela de categorização do *TagManager* substituindo a janela original. O usuário irá preencher o formulário normalmente e ao salvar, os dados serão enviados ao *Delicious* e replicados ao *TagManager*. A função para receber os dados vindo da replicação está em desenvolvimento no *TagManager*, por esse motivo ainda não está em funcionamento.

O módulo *Delicious* possui uma particularidade: a janela de categorização é uma cópia da janela original da Extensão do *Delicious*, sendo esta responsável por realizar a mesma tarefa da Extensão do próprio *Delicious* sem que o usuário perceba a modificação, conforme mostra a Figura 30.

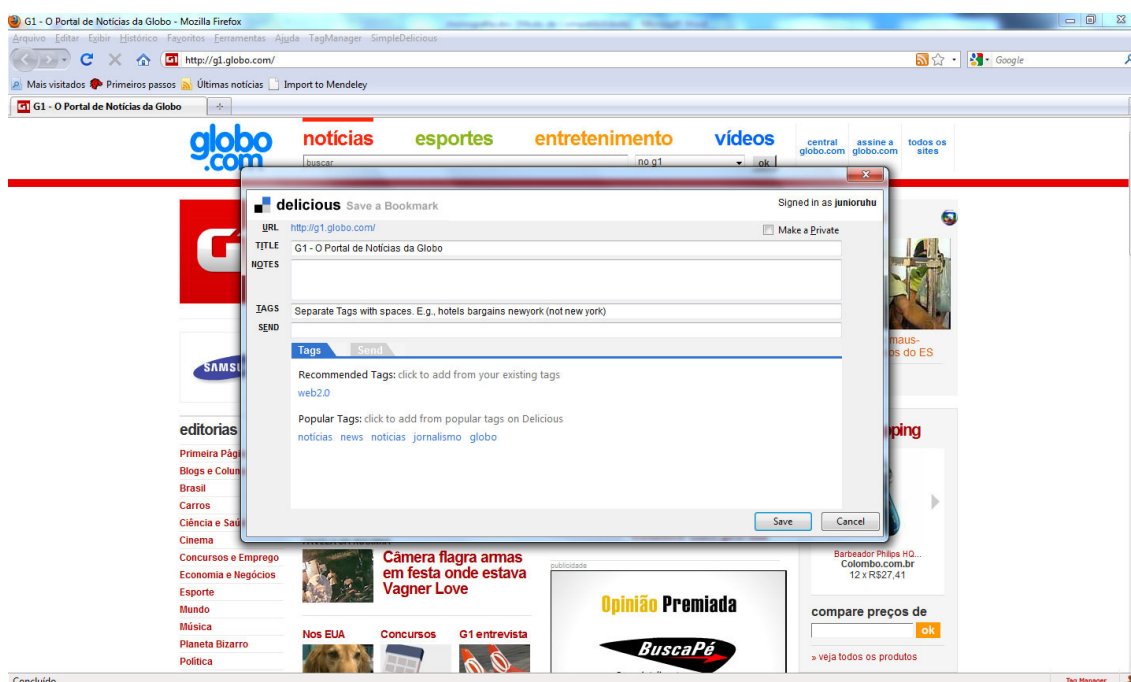


Figura 30 – Módulo Delicious em funcionamento

Seguindo a mesma idéia de desenvolvimento do *TagManager*, que priorizou a utilização do *Delicious*, por este possuir uma *API* com boa documentação e de fácil integração, somente o módulo de integração com *Delicious* foi desenvolvido (como já foi dito), nessa fase do projeto. Para realizar as demais integrações é necessário

conhecer em detalhe as *APIs* dos sistemas, como também as minúcias da utilização do mesmo, pois é de suma importância que o usuário não sinta diferença na utilização da Extensão *TagManager* em relação a aplicação do próprio sistema. Algumas *APIs* não possuem um *WebService* que permita que o usuário “final” acesse as informações por meio de requisição assíncrona, utilizando o seu próprio *login* e senha, tornando-se isso uma barreira no desenvolvimento do módulo.

Ao instalar a Extensão *TagManager* será adicionado o **Menu** *TagManager* na barra de *menus* e também um **ícone** na barra de *status* do *Firefox*, todas as funcionalidades poderão ser acessadas por teclas de atalho, além da interação por *mouse*.

4. Considerações Finais

O desenvolvimento da Extensão do *TagManager* proporcionou o conhecimento de áreas distintas, pois foi necessário integrar três conceitos diferentes: o de Extensão, e as linguagens *XUL* e *Javascript*. A linguagem *XUL* mostrou ser uma forte aliada para o desenvolvimento de interface, principalmente, quando agregada ao *Javascript*, que mostrou ser versátil na tarefa de realizar ações e comunicações.

A utilização das *APIs* dos sistemas baseados em *tagging* é necessária para realizar a troca de informações entre a Extensão e os sistemas, porém, elas podem vir a ser um fator limitante no desenvolvimento. Por exemplo, a *API* do *Youtube* não permite acesso direto do usuário ao sistema, com isso torna-se fundamental a realização de um estudo para encontrar uma maneira de contornar essa limitação.

Um fator observado no desenvolvimento da Extensão é a falta de ferramentas específicas para o desenvolvimento e depuração de uma Extensão. Assim, foi necessário utilizar-se de um simples processador de texto e um *compilador on-line*¹³, porém, este deixou de funcionar após a inserção do *javascript* no código, com isso, tornou-se necessário a cada modificação do código a realização do teste de funcionamento no próprio *Firefox*, o que tornou-se extremamente maçante, pois era necessário realizar todo o processo de desinstalação e instalação da Extensão. Em suma o desenvolvimento de uma Extensão não possui um alto grau de complexidade, porém, é necessário um bom conhecimento do domínio em que deseja trabalhar.

5. Trabalhos Futuros

É necessário estudar detalhadamente as outras *APIs* para que os demais módulos da Extensão sejam desenvolvidos. Também será necessária uma análise para decidir de que maneira a interação do usuário ocorrerá com esses sistemas. Por fim, realizar testes com usuários e enviar a Extensão para homologação pela Fundação Mozilla.

Agradecimentos

Primeiramente a Deus, que é o Senhor da minha vida, aos meus pais Antonio e Dalva que me deram oportunidade de estudar e o apoio necessário para isto. A minha

¹³ <http://ted.mielczarek.org/code/mozilla/xuledit/xuledit.xul>

namorada Ana Paula que entendeu os momentos em que estive ausente e sempre me apoiou em tudo. Ao meu orientador Dr. Sergio Silva por ter me auxiliado nessa caminhada.

Referências

- ALEXANDRE (2005), G. Introdução Básica em XUL. Obtido via internet: <http://imasters.uol.com.br/artigo/3046>. Acesso em março de 2010.
- DA SILVA (2009), Jose V.. Gerenciamento do Vocabulário do usuário em sistemas baseados em *tagging*, Maringá, fevereiro de 2009. Dissertação (Mestrado em Ciência da Computação) – Departamento de Informática, Universidade Estadual de Maringá.
- DA SILVA (2008), R., PEREIRA, R.. Aspectos da Interação Humano-Computador na *Web Social*. IHC 2008 – VIII Simpósio Sobre Fatores Humanos em Sistemas.
- FERREIRA (1986), Aurélio Buarque de Holanda. *Novo dicionário da língua portuguesa*. Rio de Janeiro: Nova Fronteira.
- FERREIRA (2005), Rafael de F.. Folksonomies e Ontologias, obtido via internet: <http://blog.rafaelferreira.net/2005/01/folksonomies-e-ontologias.html>, Acesso em março 2009.
- GALDINO (2006), Cárliston. Manual do Javascript – O poder da simplicidade. Obtido via internet: <http://bardo.castelodotempo.com/Javascript>. Acesso em março de 2010
- GÔGO (2008), Filipe R. Uma proposta de organização do vocabulário de *tags* dos usuários de sistemas baseados em *folksonomia*. Maringá, outubro de 2008. Monografia (Bacharelado em Ciência da Computação) – Departamento de Informática, Universidade Estadual de Maringá.
- GONÇALVES, Eduardo Machado Junior, VAHL, José Cláudio. *Tagging e Folksonomia*, obtido via internet: http://www.sensedia.com/br/anexos/WP_Folksonomia.pdf, Acesso em março de 2009.
- MATHES (2004), A.. *Folksonomies - Cooperative Classification and Communication Through Shared Metadata*. University of Illinois Urbana-Champaign. Obtido via internet: http://blog.namics.com/archives/2005/Folksonomies_Cooperative_Classification.pdf, Acesso em março de 2009.
- MOZILLA, Fundação. *Plugin*: para que serve e como instalar? Obtido via internet: <http://br.mozdev.org/firefox/extensoes>, Acesso em março de 2009.
- MOZILLA (2007), Fundação. Tutorial XUL – Introdução. Obtido via internet: https://developer.mozilla.org/pt/Tutorial_XUL/. Acesso em março de 2010 .
- MOZILLA (2009), Fundação. *About Javascript*. Obtido via internet: https://developer.mozilla.org/en/About_Javascript#Javascript_resources. Acesso em março de 2010

- MOZILLA (2010 A), Fundação. *Javascript*. Obtido via internet: <https://developer.mozilla.org/en/Javascript>. Acesso em março de 2010 .
- MOZILLA (2010 B), Fundação. *XMLHttpRequest*. Obtido via internet: <https://developer.mozilla.org/en/XMLHttpRequest>. Acesso em março de 2010
- LIMA (2007), G.A.B.. Categorização como um processo cognitivo. Obtido pela internet; <http://www.cienciasecognicao.org/artigos/v11/337170.html>. Acesso em março de 2009.
- O'REILY (2005), Tim. *What Is Web 2.0*, obtido via internet: <http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html>
- PEREIRA (2008), R., DA SILVA, R. *Folksonomias: Uma Análise Crítica Focada na Interação e na Natureza da Técnica*. IHC 2008 – VIII Simpósio Sobre Fatores Humanos em Sistemas.
- WIKIPÉDIA, *Javascript*. Obtido via internet: <http://pt.wikipedia.org/wiki/Javascript>. Acesso em março de 2010.
- W3C(2009). *XMLHttpRequest*. Obtido via internet: <http://www.w3.org/TR/XMLHttpRequest/#introduction>. Acesso em março de 2010