5
Detalhes da Implementação

Neste capítulo descreveremos detalhes da implementação do Explorator que foi desenvolvido usando o paradigma MVC[45] e o paradigma de CoC[46] (*Convention over Configuration*) que é definido no *framework* Ruby-on-Rails[47].

Toda a plataforma foi desenvolvida na linguagem Ruby[48].

5.1.Arquitetura Geral

Segue abaixo uma visão geral da arquitetura de implementação do Explorator:

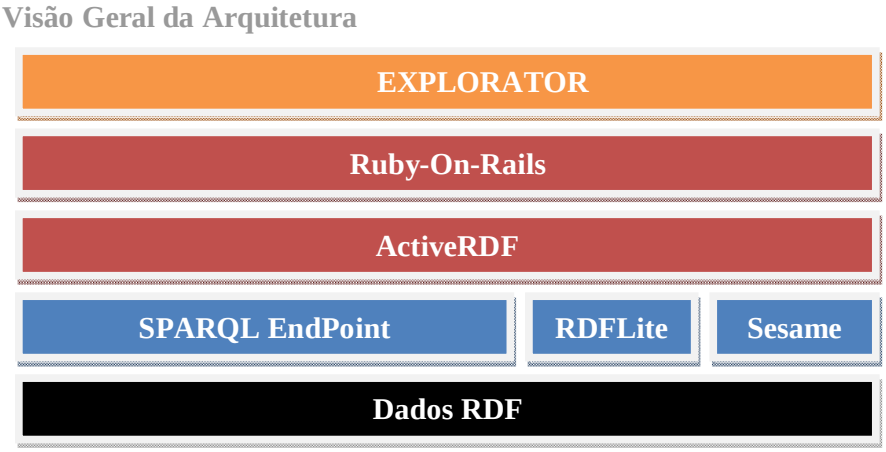

Figura 56 – Arquitetura do Explorator

[45] http://en.wikipedia.org/wiki/Model-view-controller

[46] http://en.wikipedia.org/wiki/Convention_over_Configuration

[47] http://www.rubyonrails.org/

[48] http://www.ruby-lang.org/pt/

5.1.1. Dados RDF

A base de dados do sistema será uma base de dados RDF. A arquitetura proposta é totalmente independente do conteúdo e esquema RDF, dado que o objetivo do sistema é navegar em qualquer base de dados RDF, não importando qual o esquema que a define. A única consideração a ser feita é que a base seja uma base válida e que todas as classes sejam explicitamente definidas. Caso a base não forneça tais características, a experiência do usuário poderá ser limitada, porém o sistema não apresentará problemas ou bugs.

Na versão implementada utilizaremos duas bases como exemplos: *Mondial* e *Nokia*. Tais bases estão armazenadas em repositórios da plataforma SESAME[49].

5.1.2. ActiveRDF

O *ActiveRDF* é um *framework* para acessar dados RDF dentro de programas *Ruby*. O *ActiveRDF* possui uma API (*Application Programming Interface*) que encapsula o modelo RDF no modelo de orientação a objetos. Utilizamos esta API como base para a implementação do modelo de operações do *Explorator*.

ActiveRDF é um modelo separado do *Ruby-on-Rails* e deve ser instalado independentemente e posteriormente integrado ao *Rails*. Descreveremos mais à frente esta integração.

5.1.3. Ruby-on-Rails

Ruby-on-Rails é um *framework* para desenvolvimento de aplicações Web em *Ruby*. *Ruby-on-Rails* utiliza o conceito de convenção sobre configuração (CoC) e é totalmente baseado no modelo MVC. Além disso, o ambiente *Rails* provê a API *Prototype* e *Scriptaculous*, que são APIs em Javascript para codificação de interfaces Web avançadas. A biblioteca *Prototype* dá suporte ao

[49] http://www.openrdf.org/

AJAX[50] que foi utilizado nas requisições de usuário na interface do *Explorator*. A tecnologia *AJAX* melhora a usabilidade da interface permitindo o usuário atualizar um trecho de uma página HTML sem a necessidade de recarregar todo o seu conteúdo.

5.2. Explorator

O *Explorator* foi construído sobre o *Rails* e utiliza a API do *ActiveRDF* para acessar as bases RDF. A interface do *Explorator* é totalmente baseada em Javascript e *AJAX*. Segue abaixo uma visão geral da arquitetura de interface do *Explorator*:

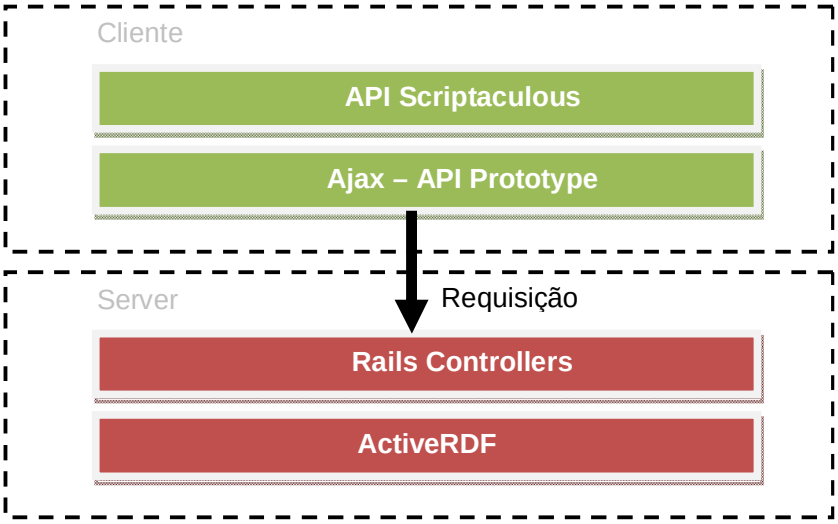

Figura 57 – Visão da chamada AJAX feita pela interface do Explorator.

No diagrama acima fica explicito que o cliente poderá realizar requisições via *AJAX* para o servidor. A chamada *AJAX* não adiciona nenhuma complexidade extra, o único efeito é que ele permite que um conteúdo seja carregado no *browser* sem a necessidade de recarregar toda a página.

[50] http://pt.wikipedia.org/wiki/AJAX_(programa%C3%A7%C3%A3o)

5.3. Diagramas de Pacotes de Classes

Abaixo seguem as estruturas de pacotes que utilizadas pelo *Explorator*. Basicamente elas seguem a mesma estrutura da plataforma *Rails.* Foram adicionados a estrutura do *Rails* os pacotes: de interface com o usuário, *ActiveRDF* e uma biblioteca de inicialização do sistema.

Por ser um ambiente MVC, o *Rails* é subdividido em 4 grupos de pacotes de classes:

- **Controller** –contém classes que tratam as requisições da interface.
- **View** – contém *templates* da interface gráfica.
- **Helper** – contém métodos que auxiliam a *view* a *renderizar* a interface.
- **Model** – contém classes do domínio do *Explorator*.

Todas as aplicações desenvolvidas no *Rails* seguem esta estrutura. A estrutura final de pacotes do *Explorator* segue abaixo:

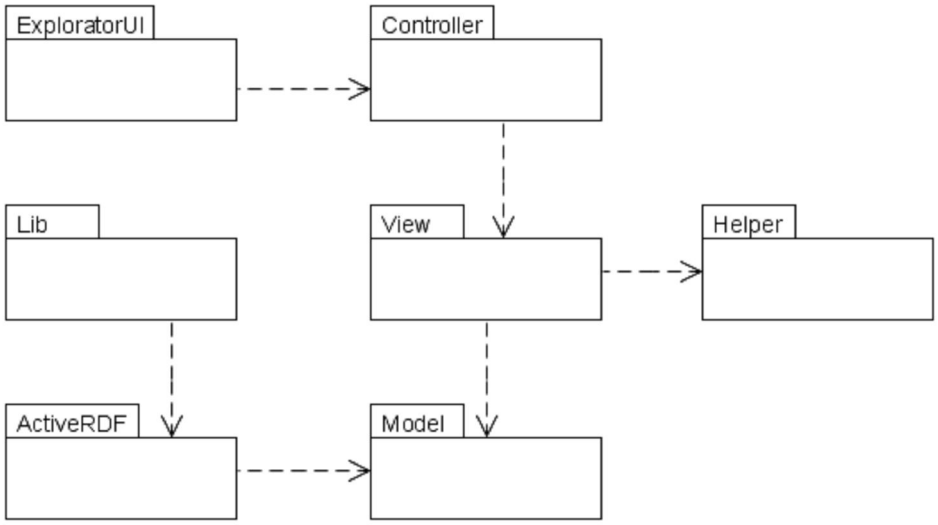

Figura 58 – Diagrama de Pacotes

Segue abaixo uma breve descrição dos pacotes do *Explorator*:

- **ExploratorUI** – Este pacote contém uma série de módulos que são utilizados para construir a interface gráfica do *Explorator*.
- **Libs** – Classes e módulos adicionais que realizam a integração do *ActiveRDF* com o *Rails.*
- **ActiveRDF** – Refere-se à aplicação *ActiveRDF* em si.

5.3.1. Diagrama de Classes

Segue a estrutura de classes do Explorator. Abaixo temos as classes do pacote *controller* e *model.*

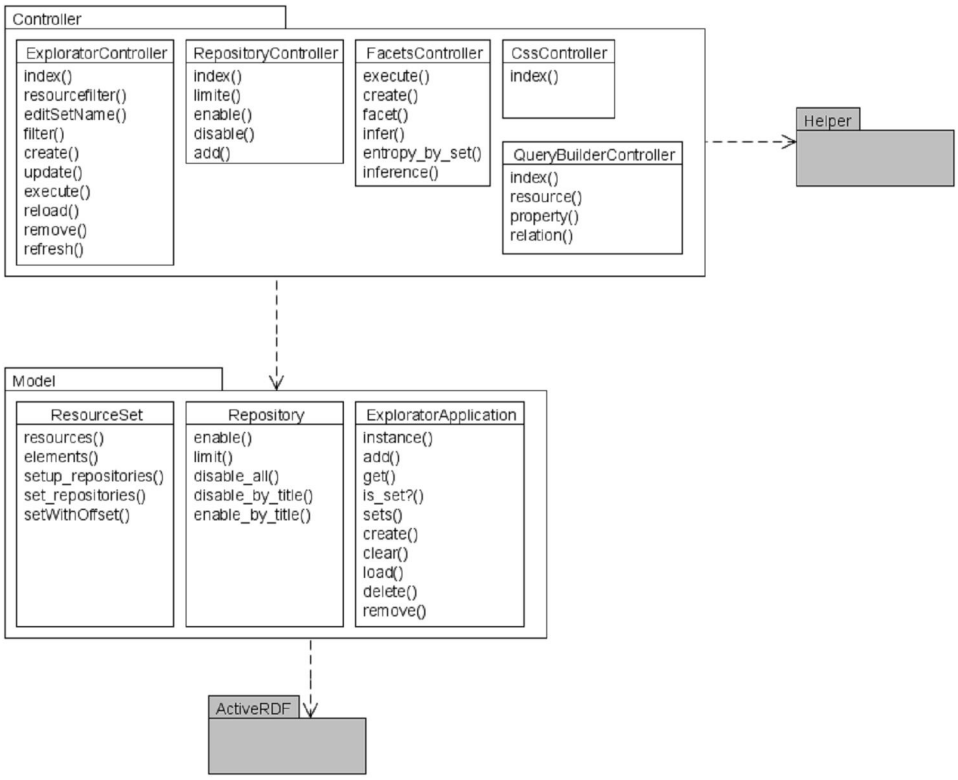

Figura 59 – Diagrama de Classes do pacote *Controller* e *Model*

Abaixo seguem as classes do pacote *helper*.

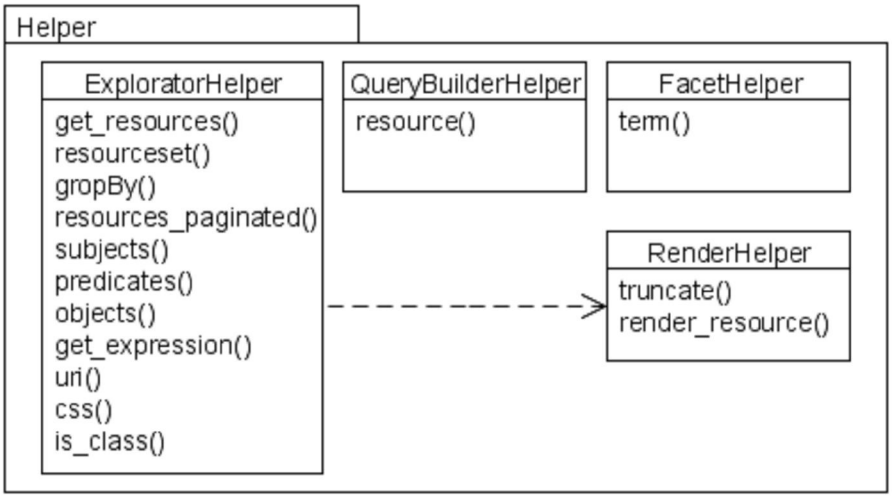

Figura 60 – Diagrama de classes do pacote *Helper.*

5.3.1.1. Classes sobre a estrutura do Rails Controllers

As seguintes classes foram implementadas no modelo de *controller* do Rails:

- **ExploratorController** – trata as requisições do usuário referente ao modelo de operações do *Explorator*. Ou seja, é responsável por executar as operações subjacentes do modelo descrito na especificação.
- **FacetController** – este *controller* trata as requisições relativas à construção de facetas. Tal operação também é descrita em mais detalhes na especificação do sistema.
- **CssController** – neste *controller* unicamente existe a *view* de edição de CSS. Toda a operação sobre CSS é feita direta na interface sem necessidade de invocar uma operação no servidor.
- **RepositoryController** – responsável por habilitar/desabilitar um repositório de dados.
- **QueryBuilderController** – trata as requisições da interface de construção de consulta disponível no Explorator.

5.3.1.2. Classes sobre a estrutura do Rails Helper

O pacote *Helper* possui classes com métodos utilizados para auxiliar a exibição dos recursos RDF na interface.

5.3.1.3. Classes sobre a estrutura do Rails Model

- **ResourceSet** - esta classe armazena uma expressão que é utilizada para obter um conjunto de recursos RDF correspondentes.
- **ExploratorApplication** – armazena um conjunto de **ResourceSet**
- **Repository** – representa um repositório do *ActiveRDF*. Esta classe é um objeto (*value object*) que é passado para ser exibido pela interface.

5.3.1.4. Classes de infra-estrutura

As classes e módulos representados no diagrama abaixo serão utilizadas principalmente para integrar o ambiente Rails com o ActiveRDF.

- **Dataload.rb** – *Dataload.rb* é um módulo contendo uma série de funções que permite o carregamento de um novo repositório no sistema. Um repositório pode ser considerado um *SparqlEndpoint* ou um arquivo na notação NT a ser carregado no *RDFLite*.
- **QueryFactory** – Encapsula a classe *Query* do *ActiveRDF* para dar acesso público a alguns atributos.
- **SemanticExpression** – Implementa o modelo de operações descrito na especificação do *Explorator*.

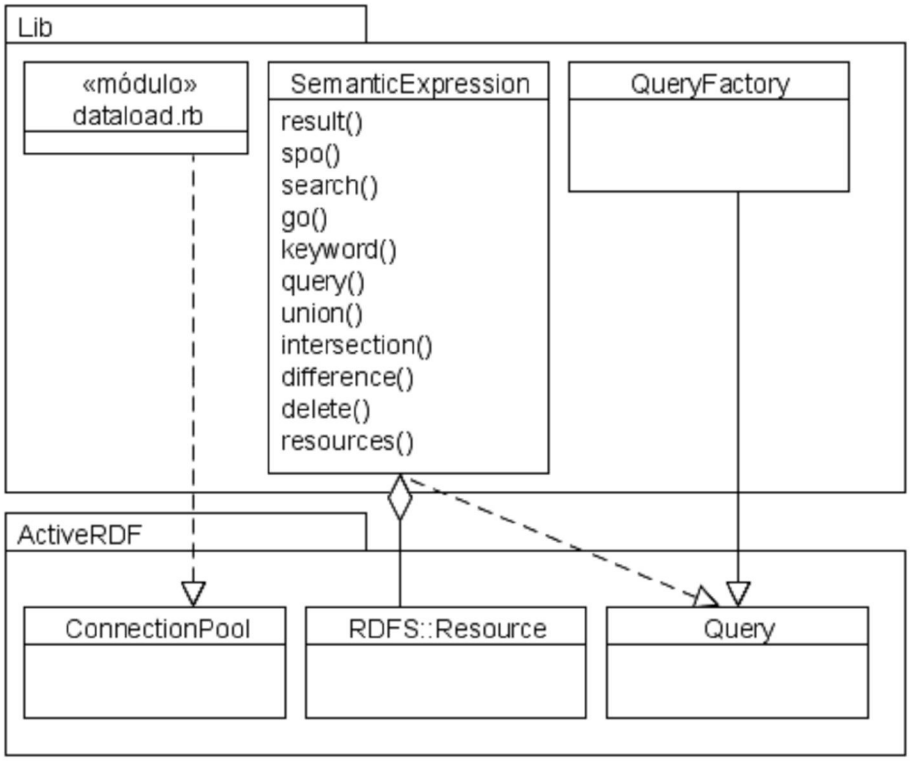

Figura 61 – Classes do pacote Lib

5.3.1.5. Classes de Interface

A interface do Explorator foi baseada na biblioteca *Scriptaculous* e *Prototype*. Tais bibliotecas foram construídas em *Javascript* e adicionam uma série de efeitos visuais a uma interface Web, tais como a funções de arrastar. Abaixo segue o diagrama de módulos de interface para o Explorator:

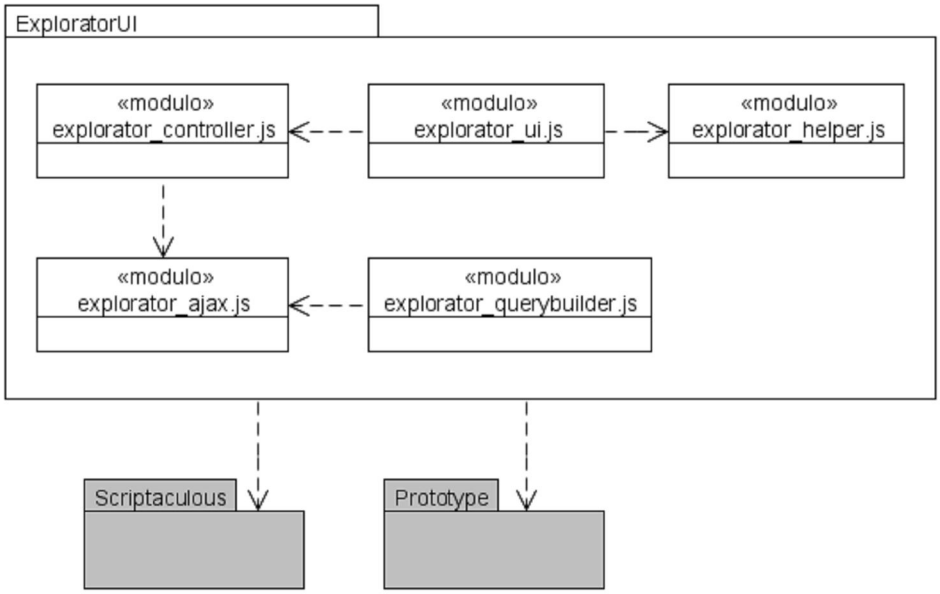

Figura 62 – Classes do pacotes ExploratorUI

- **Explorator_ajax.js** – implementa métodos de chamadas *AJAX*.
- **Explorator_controller.js** - implementa funções que refletem as operações no modelo do Explorator.
- **Explorator_ui.js** – implementa todas as funções de controle dos *widgets* de interface. Dentre estas funções temos funções de arrastar, maximiza e minimiza janela, etc.
- **Explorator_helper.js** – implementa funções de auxílio à execução de operações na interface e no controlador.
- **Explorator_querybuilder.js** – implementa uma série de métodos utilizados na funcionalidade de construção de consulta visual do *Explorator*.

5.4. Interface

Aqui descrevemos o modelo e convenções utilizadas para a construção da interface de usuário do Explorator. Basicamente, a interface do Explorator é composta de três elementos:

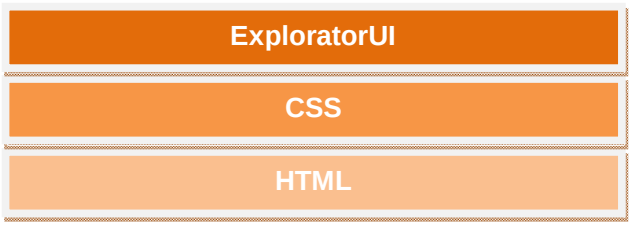

Figura 63 – Visão geral da arquitetura da interface do Explorator

5.4.1. Anotações Semânticas

Adotamos uma abordagem de adicionar notações semânticas no código HTML para definir o comportamento dos *widgets* de interface. Basicamente, esta abordagem consiste em adicionar palavras no atributo *Class* de um elemento HTML. Cada palavra poderá ser utilizada para definir formatação ou comportamento do elemento HTML. A formatação é feita através folhas de estilo CSS, processo padrão já definido pela especificação do CSS (*Cascading Style Sheets*)[51]. Já o comportamento é programado e adicionando ao elemento HTML via Javascript fazendo uso da biblioteca *Prototype* e *Scriptaculous*. Por exemplo, podemos definir a semântica para a palavra 'close' como sendo a seguinte: ao clicar no elemento anotado com o valor 'close', remova o elemento da árvore DOM[52] (*Document Object Model*) HTML. Segue um exemplo prático abaixo:

Código HTML:

```
<HTML><body>
<div class = "close">x</div>
</body></HTML>
```

[51] http://www.w3.org/Style/CSS/

[52] http://www.w3.org/DOM/

Quadro 29 – Exemplo de um trecho de código HTML com anotação semântica.

Comportamento codificado em Javascript:

```
$$('.close').each(function(item){
onclick = item.remove();
}; //Vide documentação do prototype para compreender esta sintaxe.
```

Quadro 30 – Exemplo de um trecho de código Javascript que adiciona comportamento.

Com esta linha de código, qualquer elemento anotado no atributo 'class' com o valor 'close' será removido da interface ao ser clicado. Adotamos esta mesma técnica para adicionar outros comportamentos que a interface pode assumir. É uma estratégia simples e robusta, pois permite a separação completa da codificação do layout e do comportamento da interface.

5.4.2. CSS

O *Explorator* possui um arquivo de CSS que define toda a aparência da interface. Algumas classes CSS definem tanto formatação quanto comportamento, no entanto a formatação pode ser redefinida quando existe a necessidade de apresentar o mesmo comportamento, porém com visualizações diferenciadas na interface.

Usamos também a convenção de que cada recurso RDF exibido na interface deve ter uma classe CSS para cada valor do seu predicado *rdfs:type*. Para representar o *Recurso1* abaixo em HTML, devemos anotar o atributo *class* com o valor *book*.

```
Recurso1, rdfs:label , "O alquimista"
Recurso1, rdfs:type , rdfs:book
```

Quadro 31 – Triplas utilizadas no exemplo de aplicação de CSS e RDF.

Ou seja:

```
<div class = ‘book’>Recurso1</div>
```

Quadro 32 – Exemplo da representação de um recurso RDF em HTML usando anotações CSS.

Dessa forma, estilos de formatação podem ser aplicados baseados no tipo de um recurso. Ou seja, recurso do tipo livro pode ter uma representação gráfica diferenciada de um recurso do tipo pessoa.

5.4.3. Comportamento

Aqui detalharemos um pouco mais a estratégia de anotar os elementos HTML para adicionar comportamento na interface. Descreveremos alguns *widgets* definidos no *Explorator* para ajudar na compreensão deste mecanismo.

O principal objetivo do *Explorator* é explorar uma base RDF. Esse processo inclui a visualização de recursos e triplas RDF na interface. Além disso, precisamos representar tais elementos dentro da estrutura proposta pelo nosso modelo de operações, e neste caso significa que os recursos e triplas devem ser representados em conjuntos na interface.

Para dar a noção de conjunto de recursos e triplas, apresentaremos tais elementos em *frames* de conteúdo ou janelas. A idéia é permitir o usuário visualizar blocos de informações distintos, podendo operá-los arbitrariamente. Dessa forma, criaremos o conceito de janela na interface do *Explorator*. Tal *widgets* representa um *container* de informação similar a uma janela do *Windows*, no que tange as operações de maximizar, minimizar, fechar, etc. Uma janela também pode ter barra de ferramentas e um menu. Abaixo segue um esquema de como uma janela pode ser representada utilizando anotações:

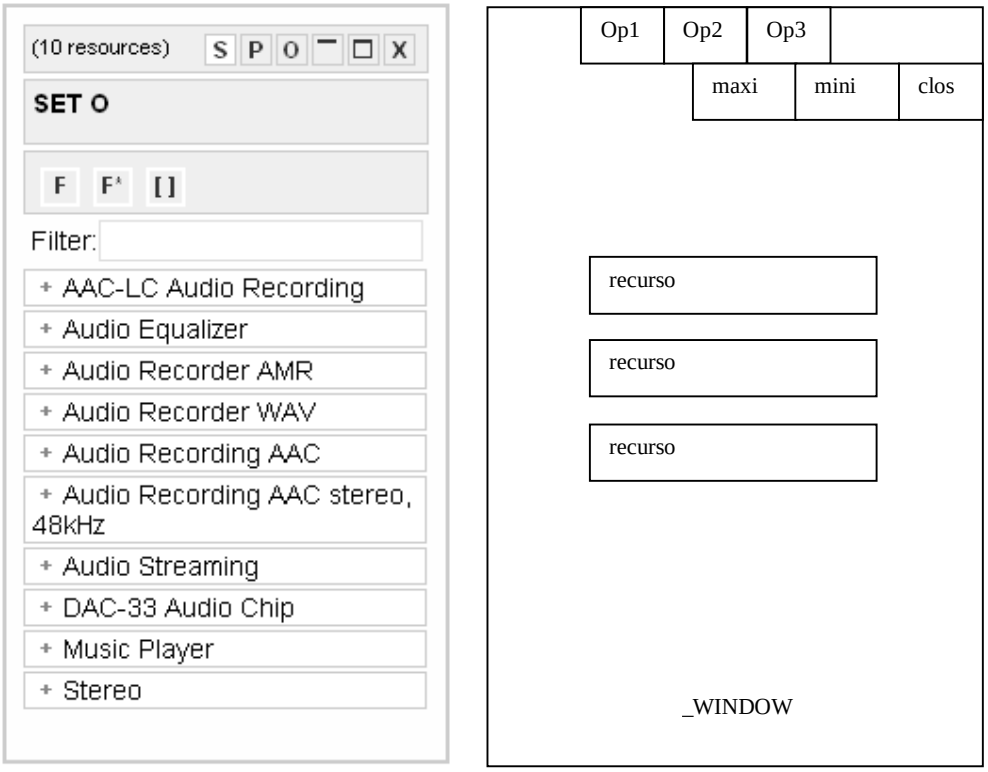

Figura 64 – Esquema exemplificando uso de anotações semânticas na interface do Explorator.

As anotações mini, maxi, close adicionam o comportamento de minimizar, maximizar e fechar, respectivamente. Note que anotações que definem comportamento não foram utilizadas para definir formação.

5.4.3.1. Propriedades

Algumas anotações definem status e/ou propriedades dos elementos na interface, tais como se o elemento está selecionado, se pode ser minimizado, se é uma janela, etc. A sintaxe utilizada para esta anotação será sempre igual a: _VALOR.

Abaixo seguem algumas das propriedades pré-definidas:

- _SELECTED – define se o elemento foi selecionado
- _WINDOW – define se o elemento é uma janela.
- _NO_MINIMIZE – define que o elemento não pode ser minimizado.

5.5. Desempenho

Durante a implementação e testes, diversos detalhes referentes ao desempenho do *Explorator* foram observados, que valem a pena serem descritos para futura referência. Durante o processo de exploração de bases extensas com mais de 100 mil triplas, a ferramenta leva um tempo considerável para dar uma resposta satisfatória ao usuário. Estudos comprovam que sem um ajuste fino na base RDF, o tempo de resposta de uma consulta pode variar consideravelmente em função do tamanho da base e do tipo da consulta (Bizer et al., 2008).

Atualmente, para cada consulta formulada na interface, o Explorator executa uma série de consultas, que foram baseadas na seguinte heurística:

1. Execute a consulta formulada pelo usuário para cada *SPARQL Endpoint* habilitado no repositório do *Explorator;*
2. Para cada recurso resultante da consulta que for exibido na interface, realize um consulta para obter suas propriedades e valores, ou as triplas em que ele é sujeito.

O primeiro passo dessa heurística é necessário, pois o sistema não tem como identificar para qual *Sparql Endpoint* esta sendo direcionada aquela consulta. Inclusive o resultado de tal consulta pode ser composto por triplas existentes em diversos *Sparql Endpoints*. A execução de uma mesma consulta em diversos *SPARLQ Enpoint* aumenta linearmente o tempo de resposta do sistema. No entanto, o usuário tem controle de quais repositórios deseja manter habilitado durante seu processo de exploração. O segundo passo dessa heurística é necessário, pois uma vez obtido um conjunto de triplas, é necessário exibi-las na interface de forma amigável. Para cada recurso da tripla (sujeito, predicado e objeto), exibimos seu *rdfs:label*, ou uma propriedade *title, name* ou seu *localname.* Sendo assim, tendo uma única base habilitada no repositório do

Explorator, uma consulta que retorne 10 triplas, contendo todos os recursos distintos, irá disparar mais 30 consultas, uma para cada recurso, somente para obter suas propriedades que serão utilizadas como um rótulo na interface.

É importante frisar que explorar uma base com mais de um milhão de triplas sem o mínimo de ajuste de performance, pode se tornar impraticável em quaisquer mecanismos de exploração.