4
O Framework

4.1
Tecnologias Utilizadas

O projeto foi todo desenvolvido utilizando-se o framework .Net na linguagem C# e a persistência dos dados é feita através de arquivos XML. Uma parte chave do projeto é a utilização da biblioteca *Piccolo*. Ela proporciona um conjunto de ferramentas que auxilia a renderização de gráficos 2D, além de um objeto de visualização que permite o zoom infinito.

O *Piccolo,* como já foi explicado, é um conjunto de ferramentas que oferece suporte ao desenvolvimento de programas de estruturas gráficas 2D em geral e a utilização de ZUIs, um tipo de interface que apresenta uma enorme quantidade de informação permitindo ao usuário fazer o "zoom in" para obter informação mais detalhada ou o "zoom out" para uma visão mais geral. Além disso, o Piccolo mantém uma estrutura hierárquica das câmeras e objetos facilitando o desenvolvimento de aplicações para manipulação de grupos. Com o *Piccolo* não existe a preocupação gráfica. A ferramenta suporta diversas funcionalidades como a eficiente renderização dos objetos, controle de tamanhos, controle de eventos, animação, etc.

O primeiro passo é distinguir os dois tipos de objetos que existem no projeto, os grupos e os itens. Os grupos consistem em uma forma abstrata de organizar os itens. Muito similar ao existente em qualquer sistema operacional, onde itens (arquivos) são organizados em grupos (diretórios), exceto pelo fato que neste projeto o tipo grupo nunca existe fisicamente em disco. Já os itens que podem ou não existir em disco, são representações de objetos em questão. Estes objetos, customizáveis, agregam informações referentes a um determinado contexto ou a diversos contextos.

Utilizando o recurso da câmera da biblioteca *Piccolo* de zoom infinito é possível exibir todos os grupos e todos os itens em uma única tela. Dessa forma é

PUC-Rio - Certificação Digital Nº 0410858/CA

possível criar uma navegação intuitiva onde o usuário pode ir de um grupo para qualquer outro através de uma única interação (um único clique).

4.2 Arquitetura

O projeto possui apenas uma interface de interação que é a janela de navegação. Isso é possível através do recurso de zoom infinito fornecido pela biblioteca *Piccolo*. Desta forma o projeto se divide em dois ramos. A aplicação e as classes. Dentre as classes temos duas principais, *PGroup* e *PItem* que definem grupos e itens, e algumas subclasses.

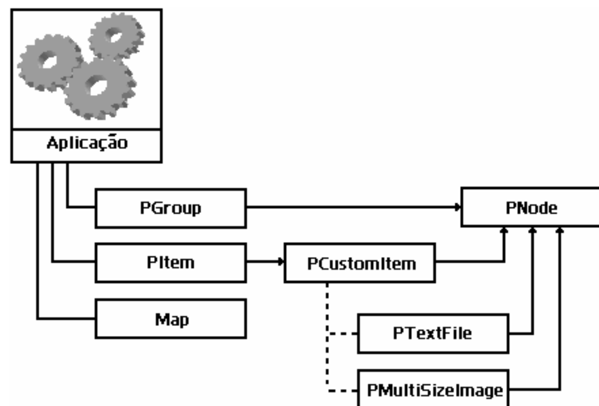

Figura 7 – Arquitetura da Aplicação.

PUC-Rio - Certificação Digital Nº 0410858/CA

4.2.1 Classes

PGroup

A classe *PGroup* implementa a organização dos objetos para o usuário, os grupos. Um grupo pode estar contido em outro grupo na hierarquia. Essa classe possui os atributos descritivos e métodos aplicáveis como, por exemplo, organizar os subitens, união de grupos, interseção, etc.

PItem

A classe *PItem* implementa os objetos a serem exibidos. Ela na verdade funciona como uma interface entre a aplicação e as classes de objetos. O objetivo disso é permitir aos futuros desenvolvedores utilizar o framework para outras classes de objetos sem a necessidade de alteração do código da aplicação. Desta

forma qualquer classe de objeto a ser utilizada deve herdar a classe *PItem*. A escolha de implementar a *PItem* como uma classe ao invés de uma interface foi feita para que exista a possibilidade de reutilização do código já escrito. Assim, métodos que são aplicáveis a todos os tipos de itens independentemente das classes não precisam ser reescritos. Também estão contidos na classe *PItem* atributos contendo informações pertinentes a qualquer tipo de classe que são necessários na aplicação. E, finalmente, métodos que coletam informações das subclasses e organizam de forma padronizada no formato XML para que seja analisado pela aplicação.

Para exemplificar isto estão inclusas no projeto algumas subclasses da classe *PItem*, que reutilizam seu código e implementam diferentes tipos de objetos: *PMultiSizeImage*, *PTextFile* e *PCustomItem*.

PMultiSizeImage

Na *PMultiSizeImage* o item é um arquivo de imagem, nela são implementadas sua renderização, suas transformações e mantidas informações específicas adicionais. Seus métodos de manipulação navegacional e hierárquica são herdados da classe *PItem*. Devido a uma implementação opcional, esta classe exibe a imagem de duas formas distintas: compactada e integral.

A renderização compactada exibe um thumbnail da imagem e a renderização integral exibe a imagem em si.

Dos métodos aplicáveis a esta classe foram implementadas algumas transformações como, por exemplo, rotação da imagem, transformação sépia e transformação para monocromático. Para essas duas últimas foi utilizada a biblioteca angGoGo.PhotoController [18] que fornece diversas ferramentas para o trabalho com imagens.

PTextFile

Na *PTextFile,* o item é um arquivo texto. Nela são implementadas sua renderização e mantidas informações específicas adicionais. Seus métodos de manipulação navegacional e hierárquica são herdados da classe *PItem*. Devido a uma implementação opcional, esta classe é exibida de duas formas distintas: compactada e integral.

PUC-Rio - Certificação Digital Nº 0410858/CA

A renderização compactada exibe o nome do arquivo texto e a renderização integral exibe o conteúdo do arquivo.

Dos métodos aplicáveis a esta classe foi implementada a cópia do texto para a área de transferência (*clipboard)* do sistema.

PCustomItem

A classe *PCustomItem* é uma classe de item genérica que aceita um item como uma composição de outros itens. Desta forma um objeto *PCustomItem* pode ser apenas uma imagem (*PMultiSizeImage*), apenas um texto (*PTextFile*), desenhos de formas geométricas, ou uma composição com um ou mais objetos dessas diversas classes.

Devido a uma implementação opcional, esta classe pode ser renderizada de duas formas distintas: compactada e integral, sendo que essas duas formas de renderização são completamente desacopladas. Por exemplo, é possível exibir na forma compactada apenas uma imagem e na sua forma integral uma composição de diversas imagens, textos, etc., podendo se utilizar de classes também customizadas para isso, dentre elas as citadas anteriormente *PMultiSizeImage* e *PTextFile*.

O projeto foi desenvolvido para que a implementação dessas subclasses que definem o conteúdo de um item seja aberta ao desenvolvedor. Nelas devem ser definidas quais as informações pertinentes ao objeto e os atributos que podem ser filtrados. Por exemplo, para classe *PMultiSizeImage* foram definidos atributos de *description*, *grouping*, *date* e *size*, sendo os dois primeiros editáveis através da aplicação e os dois últimos *readonly* apenas exibindo a informação vinda do arquivo. Já os filtros definidos neste exemplo foram por *parentgroup*, *description*, *date*, *size*. Assim como os filtros possíveis, são também configuráveis os operadores possíveis de cada filtro dependendo do tipo da sua informação. Aqui foram definidas algumas informações de três tipos diferentes: texto, número e data. Já os operadores definidos para os respectivos tipos de dados foram:

- Texto - operadores de busca por igualdade (=), diferença (!=) e texto contido (LIKE).
- Número - operadores de busca por igualdade (=), diferença (!=), maior (>) e menor (<).

PUC-Rio - Certificação Digital Nº 0410858/CA

- Data - operadores de busca por igualdade (=), diferença (!=), maior (>) e menor (<).

Se, por um lado, permitir ao desenvolvedor definir todas essas configurações fornece um ambiente mais amplo com maiores possibilidades, por outro, isto aumenta relativamente a complexidade do desenvolvimento. Alguns métodos se tornam imprescindíveis nas subclasses e suas implementações devem receber e retornar alguns valores esperados. Em trabalhos futuros são discutidas as possíveis soluções para tal complexidade.

Map

A classe *Map* implementa os métodos de mapeamento do framework aplicação para alguma DSL. Ela deve ser customizada pelo desenvolvedor para que os mapeamentos funcionem em dois sentidos. Mapear ações e interações exercidas na interface do framework para que reflitam em seu domínio e mapear os resultados gerados pelo domínio para serem exibidos corretamente na interface. Por exemplo, a DSL em utilização fornece um método que faz a união de grupos. Para isso a classe que representa os grupos na interface *PGroup* deve chamar um método de mapeamento na classe *Map*. Este método de mapeamento fará as conversões necessárias entres os tipos da classe *PGroup* para a classe que representa grupos na DSL, chamará o método responsável pela execução da união passando os devidos parâmetros e retornará a resposta já convertida de volta para a classe *PGroup*.

Essa mesma lógica de mapeamento deve ser feita tanto para a classe de grupo quanto para a classe de itens.

4.2.2 Aplicação

Cabe à aplicação, ou à janela de interação, toda a lógica de organização, de navegação e de interação. Ela é responsável por manter a hierarquia dos objetos e exibi-los na interface conforme requisitado. Nesta janela são implementadas os sistemas de navegação por zoom e as interações possíveis. Dentre as interações existentes são elas:

PUC-Rio - Certificação Digital Nº 0410858/CA

- Interações com o mouse: clique simples, CTRL + clique simples, clique duplo, clique com botão direito, botão de rolagem, drag&drop, SHIFT+drag&drop.
- Interações com o teclado: CTRL+X, CTRL+C, CTRL+V, SHIFT+CTRL+V, Delete.

Dessas interações, as configuráveis pelo desenvolvedor são as de *SHIFT+drag&drop*, *clique com botão direito*, *CTRL+V*, *SHIFT+CTRL*+V e *Delete*. *Delete*, comumente, é a interação básica de remover objetos selecionados, contudo, sua semântica é aberta e seu resultado assim como estas outras interações dependem de uma resposta do controlador. Todas as outras representam interações básicas do sistema ou operações de navegação. Por exemplo, o botão de rolagem do mouse está diretamente ligado ao "zoom in" e "zoom out" da câmera de visualização, o *clique* simples representa a seleção de algum item, assim como o *CTRL + clique* simples a seleção de múltiplos objetos. O *drag&drop* simples de um item ou grupo apenas reposiciona o objeto em questão em outra posição da tela. Maiores detalhes das interações são descritos na secção 4.4.3 Guia de Uso da Interface.

São três as classes visualizadas pela aplicação: *PGroup*, *PItem* e *Map*. Todas as interações são chamadas a partir de métodos descritos em uma dessas classes. Contudo, alguns métodos customizáveis existentes nas classes superiores a *PItem* são chamadas diretamente pela aplicação. Estes métodos devem estar presentes nas classes customizáveis.

A aplicação também é responsável pela persistência de dados. Durante a utilização, a interface pode salvar o seu presente estado para depois iniciar esta mesma sessão da mesma forma e posicionamento que foi salva. A aplicação gera e carrega um arquivo XML com as informações necessárias. Além das informações básicas, este XML também guarda informações adicionais dos atributos customizáveis dentro da tag <custom> do item. Abaixo segue exemplo do XML gerado para a persistência dos dados. Pode se identificar no trecho abaixo um grupo em uma determinada posição e um item posicionado dentro deste grupo com algumas informações de atributos customizáveis. Dentre eles, *Título*, *Descrição* e *Data*.

PUC-Rio - Certificação Digital Nº 0410858/CA

```
<xml>
  <node type="ZoomNavigator.PGroup">
    <id>0</id>
    <description>Arquivos</description>
    <color>System.Drawing.SolidBrush</color>
    <position>
    <x>270,0925</x>
    <y>-300,6145</y>
    </position>
    <subnodes>
      <node type=" ZoomNavigator.PItem">
      <id>3467a38b-f925-4bd5-8341-1e7a9eaf8166</id>
      <path>C:\texto.txt</path>
      <custom>
        <Título>titulo</Título>
        <Descrição>descrição do item</Descrição>
        <Data>16/5/2006 22:35:35</Data>
      </custom>
      <position>
      <x>0</x>
      <y>0</y>
      </position>
      </node>
      ...
    </subnodes>
  </node>
  ...
</xml>
```

Quadro 2 – Arquivo XML para persistência de dados.

4.3 Utilizando o framework

4.3.1 O Desenvolvedor

O primeiro passo do usuário desenvolvedor é definir qual o tipo de objeto que irá utilizar e qual o seu repositório de objetos. Tendo definido isto, o próximo passo é mapear esses dados para a interface gerada pelo framework. Isto é feito

PUC-Rio - Certificação Digital Nº 0410858/CA

através da codificação das classes customizáveis. São elas: *PGroup*, *PCustomItem* e *Map*.

No caso onde o desenvolvedor não possui uma DSL, ele deve definir os algoritmos das operações e manipulações sobre os grupos e objetos dentro da classe *Map*. Já no caso de existir uma DSL que gere os resultados a partir de parâmetro, a classe *Map* fica encarregada apenas de estabelecer a conexão entre as classes *PCustomItem* e *PGroup* com os métodos da DSL.

A classe *PGroup* por representar apenas uma organização hierárquica dos itens, logo não necessita da customização de sua renderização, apenas dos métodos aplicáveis aos grupos:

- Nos métodos *GetClassFunctionsNames()*,*GetClassFunctions()*, e *GenericFunctionMenu()*, são feitas as alterações para que o *drop down menu* exiba e execute as funções. No método *GetClassFunctionsNames()* devem ser adicionados os nomes que serão exibidos no menu.

PUC-Rio - Certificação Digital Nº 0410858/CA

```
public ArrayList GetClassFunctionNames()
  {
        ArrayList retorna = new ArrayList();retorna.Add("Arrange Cascade");
        retorna.Add("Arrange Side by Side");
        retorna.Add("Change Color");
        retorna.Add("-");
        retorna.Add("Union");
        retorna.Add("Intersection");
        retorna.Add("Difference");
        return retorna;
  }
```

Quadro 3 – Configurando nomes das funções na classe grupo.

Em seguida o método *GetClassFunctions()* deve conter as respectivas chamadas para os nomes das funções. Essas chamadas devem ser alinhadas aos nomes dados às funções, isto é, obedecer a mesma ordenação. O método a ser executado pode ser chamado diretamente neste momento ou através de um método de chamada genérico.

```
public ArrayList GetClassFunctions()
{
        // Cria array de retorno
        ArrayList retorna = new ArrayList();
```

```
                // Adiciona médotos ao array
                retorna.Add(new System.EventHandler(ArrangeCascade));
                retorna.Add(new System.EventHandler(ArrangeSideBySide));
                retorna.Add(new System.EventHandler(ChangeGroupColor));
                retorna.Add(new System.EventHandler(GenericFunctionMenu));
                retorna.Add(new System.EventHandler(GenericFunctionMenu));
                retorna.Add(new System.EventHandler(GenericFunctionMenu));
                retorna.Add(new System.EventHandler(GenericFunctionMenu));
                return retorna;
        }
```

Quadro 4 – Configurando chamadas das funções na classe grupo.

Para o caso de ter sido referenciado, o método genérico de chamada *GenericFunctionMenu()* deve ser implementado passando a chamada da função para o método responsável pela ação.

```
protected void GenericFunctionMenu(object sender, System.EventArgs e)
{
  string optionText = (sender as MenuItem).Text;
  if(optionText=="Union")
  {
       if(this == MainForm.mainForm.currentGroup)
         Map.doGroupActionMenu("union", MainForm.mainForm.selectedGroups);
   }
   else if(optionText=="Intersection")
   {
       if(this == MainForm.mainForm.currentGroup)
         Map.doGroupActionMenu("intersection", MainForm.mainForm.selectedGroups);
   }
  else if(optionText=="Difference")
   {
       if(this == MainForm.mainForm.currentGroup)
          Map.doGroupActionMenu("difference", MainForm.mainForm.selectedGroups);
    }
}
```

Quadro 5 – Configurando chamadas das funções na classe grupo.

- O método *doAction()* é responsável pelas chamadas de manipulação direta feitas na interface. Nele, da mesma forma que nos métodos anteriores,

PUC-Rio - Certificação Digital Nº 0410858/CA

devem ser feitas as implementações relativas às ações associadas às interações. Por exemplo, para executar a união de dois grupos chamando o método da DSL. Primeiramente deve-se criar a opção união dentro do método *doAction()* da classe *PGroup*.

```
…
 if (actionName == "union")
 {
   if(targetGroup!=null)
     resultGroup = this.Union(RootLayer, oldParent, targetGroup, true, lastGroup);lastGroup);
   else
     resultGroup = null;
 }
...
```

Quadro 6 – Mapeando operações.

O método *doAction()* recebe diversos parâmetros, dentre eles o *actionName* o qual indica a interação efetuada. O nome desta interação é o mesmo que foi escolhido no arquivo de configuração. Neste trecho do código, cabe agora ao método *Union()* da classe *PGroup* chamar e retornar os resultados de um método de união na classe *Map.* Para este caso, o algoritmo que gera a união dos grupos é implementado na própria classe *Map*, pois este repositório não possui uma DSL com a semântica e com o método de união,

Já classe *PCustomItem* deve passar por uma customização mais detalhada, dentre elas, customização da forma de renderização, das propriedades, dos filtros e dos métodos aplicáveis à classe.

- Renderização do objeto em questão. Neste exemplo o objeto será um arquivo texto e sua exibição inicial será apenas o nome do arquivo.

```
public PCustomItem(string path)
{
        // Guarda caminho do arquivo
        filePath = path;
        this.Text = GetFileName();
```

PUC-Rio - Certificação Digital Nº 0410858/CA

```
            this.fileName = GetFileName();
    }
```

Quadro 7 – Configurando o construtor PCustomItem.

- Estados de renderização do objeto. O framework possibilita a exibição dos objetos em dois estados distintos: compactado e integral. Estes dois estados de exibição podem ser definidos pelo desenvolvedor como o seguinte exemplo onde o código implementa a forma compacta para exibir apenas o nome do arquivo texto e a forma integral para exibir o conteúdo do arquivo texto.

```
public bool ShowCompact
{
  get { return Text == fileName; }
  set {
            //Caso ShowCompact = true
            if(value)
            {
                    // Mostra nome
                    Text = fileName;
                    this.Brush =  null;
            }
            // Senão
            else
            {
                    // Mostra conteúdo
                    Text = GetTextFile();
            }
            // Invalida renderização
            InvalidatePaint();
   }
  }
```

Quadro 8 – Configurando os dois estados de renderização.

- Enumeração dos atributos desejados para o objeto. Os atributos são guardados em três vetores. O *customInfoTag* onde são guardados os nomes dos atributos, o customInfoValue onde são guardados os valores e o *customInfoReadyOnly* que guarda a definição se o atributo é ou não

PUC-Rio - Certificação Digital Nº 0410858/CA

editável pelo usuário final dentro da interface. Neste exemplo são adicionados três atributos para o objeto: *título*, *descrição* e *data*. Tí*tulo* e *descrição* são editáveis e seus valores não são iniciados neste primeiro momento. Já o atributo data não é editável pelo usuário final e seu valor já é preenchido com a data do arquivo.

```
protected void GetCustomInfo()
{
        customInfoTag.Clear();
        customInfoValue.Clear();

        customInfoTag.Add("Título");
        customInfoValue.Add("");
        customInfoReadOnly.Add(false);

        customInfoTag.Add("Descrição");
        customInfoValue.Add("");
        customInfoReadOnly.Add(false);

        customInfoTag.Add("Data");
        customInfoValue.Add(this.infoFileDate.ToString());
        customInfoReadOnly.Add(true);
}
```

Quadro 9 – Adicionando atributos.

- Descrição de possíveis filtros. Os filtros possíveis são armazenados em um vetor e estes só podem existir sobre propriedades definidas para o objeto. No caso abaixo são adicionados os filtros para *título*, *descrição* e *data* do objeto.

```
protected void GetPossibleFilters()
{
        possibleFilters.Clear();
        possibleFilters.Add("Título");
        possibleFilters.Add("Descrição");
        possibleFilters.Add("Data");
}
```

Quadro 10 – Configurando os possíveis filtros.

PUC-Rio - Certificação Digital Nº 0410858/CA

Então devem ser definidos os possíveis operadores para cada um dos filtros. Abaixo são definidos os operadores =*, !=* e *like* para os filtros *título* e *descrição*. Já para o filtro de *data* são definidos os operadores =, *!*=, > e <.

```
public void GetPossibleFiltersOperators(string filter)
{
        possibleFilters.Clear();
        if(filter=="Título "||filter==" Descrição")
        {
                possibleFiltersOp.Add("=");
                possibleFiltersOp.Add("!=");
                possibleFiltersOp.Add("like");
        }
        if(filter=="Data")
        {
                possibleFiltersOp.Add("=");
                possibleFiltersOp.Add("!=");
                possibleFiltersOp.Add(">");
                possibleFiltersOp.Add("<");
        }
}
```

Quadro 11 – Configurando os possíveis operadores dos filtros.

PUC-Rio - Certificação Digital Nº 0410858/CA

Finalmente deve ser implementado o teste realizado pelos filtros. Aqui, para cada filtro existente são passados para a operação de filtragem o parâmetro de entrada, o valor da propriedade do objeto, o operador e o tipo do dado (*string*, *inteiro*, *data*, etc.).

```
protected bool TestFilters()
{
        bool itemVisible=true;
        try
        {
          for(int i=0; i<=filters.Count;i++)
          {
          // Descrição ou Título
          if( ((filters[i] as String) == "Descrição") || ((filters[i] as String) == "Título") &&
itemVisible)
          {
```

```
                    itemVisible = FilterOperation(GetCustomInfoValue((filters[i] as
                    String)),filtersParam[i].ToString(),filtersOp[i].ToString(),"string");
          }
          // Data
          if((filters[i] as String) == "Data"  && itemVisible)
          {
                    itemVisible = FilterOperation(GetCustomInfoValue((filters[i] as
                    String)),filtersParam[i].ToString(),filtersOp[i].ToString(),"date");
          }
          if(!itemVisible)
                    return itemVisible;
          }
                    return itemVisible;
        }
        catch
        {
          return itemVisible;
        }
}
```

Quadro 12 – Testando os filtros.

- Para a customização dos métodos aplicáveis, assim como na classe *PGroup*, os métodos *GetClassFunctionsNames()*, *GetClassFunctions()*, e *GenericFunctionMenu()*, devem ser alterados para que o *drop down menu* exiba e execute as funções. No método *GetClassFunctionsNames()* devem ser adicionados os nomes que serão exibidos no menu.

```
        public  void GetCustomClassFunctionsNames(ArrayList functionNames)
        {
          functionNames.Add("Copy Text");
        }
```

Quadro 13 – Configurando nomes das funções na classe item.

Em seguida o método *GetClassFunctions()* deve conter as respectivas chamadas para os nomes das funções. Essas chamadas devem ser alinhadas aos nomes dados às funções. O método a ser executado pode ser chamado diretamente neste momento ou através de um método de chamada genérico.

```
        public void GetCustomClassFunctions(ArrayList functions)
```

PUC-Rio - Certificação Digital Nº 0410858/CA

```
        {
          functions.Add(new System.EventHandler(GenericFunctionMenu));
        }
```

Quadro 14 – Configurando chamadas das funções na classe item.

Para o caso de ter sido referenciado o método genérico de chamada *GenericFunctionMenu()* deve ser implementado passando a chamada da função para o método responsável pela ação.

```
protected void GenericFunctionMenu(object sender, System.EventArgs e)
{
        string optionText = (sender as MenuItem).Text;
        if (optionText == "Copy Text")
                (this.GetChild(0) as PTextFile).CopyToClipBoard();
}
```

Quadro 15 – Configurando chamadas das funções na classe grupo.

Com essas poucas implementações, o desenvolvedor consegue utilizar o framework para criar a interface ZUI de visualização de seu objeto. Para um segundo caso onde o desenvolvedor queira utilizar-se de sua DSL e de suas operações as classes *PGroup* e *Map* devem ser codificadas. Tirando proveito dessa funcionalidade e extensibilidade das classes, o desenvolvedor pode utilizar os métodos de manipulação existentes e pré-definidos em sua DSL, sem a necessidade de toda uma recodificação para o ambiente do framework. A exemplificação de utilização será abordada melhor no próximo capítulo e para maiores detalhes de programação o código do framework se encontra comentado e bem explicativo.

4.3.2
O Usuário Final

Após o processo de desenvolvimento e implementação, o resultado é apresentado para o usuário final na forma de uma interface de navegação e manipulação direta. Com a finalidade de oferecer maior aplicabilidade e usabilidade da interface gerada, esta possui um arquivo de configuração que pode ser alterado pelo usuário final. Através deste arquivo é possível adicionar filtros e

PUC-Rio - Certificação Digital Nº 0410858/CA

operadores para as interfaces e alterar a semântica de suas ações (estes a partir de opções implementadas pelo usuário desenvolvedor).

O exemplo a seguir ilustra a configuração da semântica para a interação de drag&drop na interface. A interação de arrastar um item sobre um grupo ou arrastar um grupo sobre um grupo é implementação fixa do framework. Já os seus possíveis resultados são opções que devem ser implementadas pelo desenvolvedor. Aqui a interação de arrastar um item sobre um grupo tem como resultado a operação *cutpaste* implementada pelo desenvolvedor. Considerando como descritivo o nome desta operação pode-se dizer que o resultado de arrastar um item sobre um grupo retira o item do seu grupo original (*cut*) e o insere no grupo de destino (*paste*).

Da mesma forma, a interação de arrastar um grupo sobre outro grupo é implementação fixa do framework e seus possíveis resultados dependem de implementação por parte do desenvolvedor. Neste mesmo exemplo, o resultado para esta interação está selecionado como sendo a união dos grupos em questão.

```
<config>
  <semantic>
    <actions>
      <action name="dragdropitemtogroup" >
        <operation>cutpaste</operation>
        <!--options>cutpaste, copypaste, copypastespecial<options-->
      </action>
      <action name="dragdropgrouptogroup" >
        <operation> union </operation>
        <!--options>cutpaste, copypaste, copypastespecial, union, intersection<options-->
      </action>
      ...

    </actions>
    <filters>
    ...
    </filters>
  </semantic>
</config>
```

Quadro 16 – Arquivo de Configuração (Ações)

PUC-Rio - Certificação Digital Nº 0410858/CA

Outras opções do usuário final são os filtros existentes e seus respectivos operadores. Os filtros que aqui podem ser selecionados, assim como os operadores, são os mesmos que foram implementados pelo usuário desenvolvedor. Neste momento cabe ao usuário final apenas selecionar quais são as opções desejadas. No exemplo abaixo é definido o filtro sobre o atributo *descrição* e os seus possíveis operadores são = != e *like*.

```
<config>
<semantic>
  <actions>  ... </actions>
  <filters>
    <filter field="Descrição">
      <operators>
        <operator>=</operator>
        <operator>!=</operator>
        <operator>like</operator>
      </operators>
    </filter>
     ...
  </filters>
</semantic>
</config>
```

Quadro 17 – Arquivo de Configuração (Filtros)

PUC-Rio - Certificação Digital Nº 0410858/CA

4.3.3 Guia de Uso da Interface

Após as configurações feitas pelo usuário final, a utilização da interface se resume em ínfimas opções e interações. Alguns itens de menu, interações sobre a interface e poucas entradas de dados como listado abaixo:

Menu

- File
  - New Group - Adiciona um grupo vazio à área de trabalho.
  - Add Item - Adiciona itens ao grupo atual da área de trabalho.
  - Save Workspace - Grava o estado atual da área de trabalho.

- o Load Workspace - Carrega o estado da última área de trabalho salva.
- o Load Config - Carrega o arquivo de configuração.

- View
  - o View All - Ajusta o zoom e centraliza câmera para que todos os grupos e itens fiquem visíveis.
  - o Arrange Cascade - Organiza subitens do grupo atual em cascata.
  - o Side By Side - Organiza subitens do grupo atual lado a lado

Interações na área de trabalho

- Clique
  - o Seleciona item sob o ponteiro. O item clicado passa a ser o item atual para efeitos de transformações.
- CTRL + Clique
  - o Modo para múltipla seleção.
- Duplo Clique
  - o Sobre a área de trabalho - Ajusta o zoom e centraliza câmera para que todos os grupos e itens fiquem visíveis.
  - o Sobre um grupo - Ajusta o zoom e centraliza câmera para que todo o grupo fique visível.
  - o Sobre um item - Ajusta o zoom, centraliza câmera sobre o item e altera o estado do item para sua forma integral.
- Clique Direito
  - o Monta o drop down menu de métodos específicos da respectiva classe onde foi clicado (grupo ou item).
- Drag&Drop
  - o Reposiciona objeto selecionado.
- SHIFT + Drag&Drop
  - o Executa o método específico da classe, o qual foi configurado, sobre os objetos dragados.
- Botão de rolagem do mouse
  - o Aproxima ou afasta o zoom da câmera.
- CTRL + A
  - o Seleciona todos os itens do grupo atual.

PUC-Rio - Certificação Digital Nº 0410858/CA

- CTRL + C
  - Copia itens selecionados para a área de transferência.
- CTRL + X
  - Recorta itens selecionados (copia itens selecionados para.a área de transferência e os remove da área de trabalho)
- CTRL + V
  - Cola itens da área de transferência no grupo atual.
- DELETE
  - Remove itens selecionados.

Botões e Entrada de Dados

- Caixa de entrada de dados:
  - Edição da tag identificadora dos grupos.
- Botão *Pan*
  - Habilita/desabilita modo de arrasto da câmera de visão.
- Lens
  - Habilita/desabilita lente de filtragem sobre objetos.

  Ao habilitar a lentes surge um *popup* de configuração para a seleção da propriedade a ser filtrada, do operador e do seu valor de filtragem.
    - Botão "+" – adiciona nova condição de filtragem.
    - Botão "-" – remove última condição de filtragem.

PUC-Rio - Certificação Digital Nº 0410858/CA

4.4
O Resultado

A aplicação resultante da compilação do framework com um modelo de dados e um controlador é uma interface de navegação e interação para manipulação direta. Uma aplicação estruturada à luz da arquitetura *MVC* e flexível o suficiente para que o usuário final possa selecionar as semânticas das operações a serem realizadas em interações na interface.

Primeiramente o programa possibilita ao usuário configurar certos aspectos de interação e navegação como explicados na sessão anterior. Feito isso, com a aplicação em execução o usuário pode adicionar grupos e itens, e atribuir valores às suas propriedades. Pode navegar por estes grupos criados, manipular

diretamente os grupos e gerar novos grupos a partir dos já existentes. Filtrar os itens por parâmetros definidos, aproximar e afastar a câmera, aplicar N transformações da quais foram implementadas pelo desenvolvedor, sobre itens ou conjuntos e finalmente salvar e/ou carregar sua área de trabalho atual. As figuras a seguir ilustram a interface gerada e algumas ações possíveis.

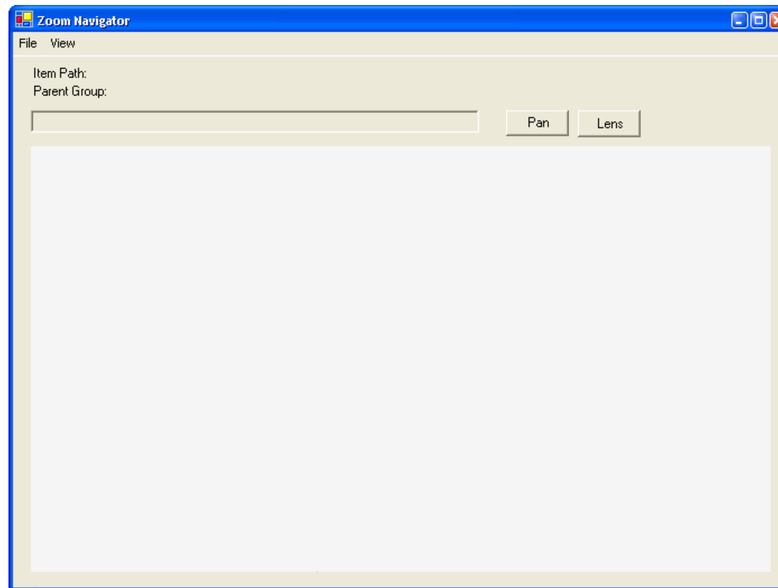

Figura 8 – Interface de navegação.

Depois de adicionados dois grupos e alguns itens do tipo arquivo de texto. Os itens são exibidos no formato compactado mostrando apenas o nome do arquivo.

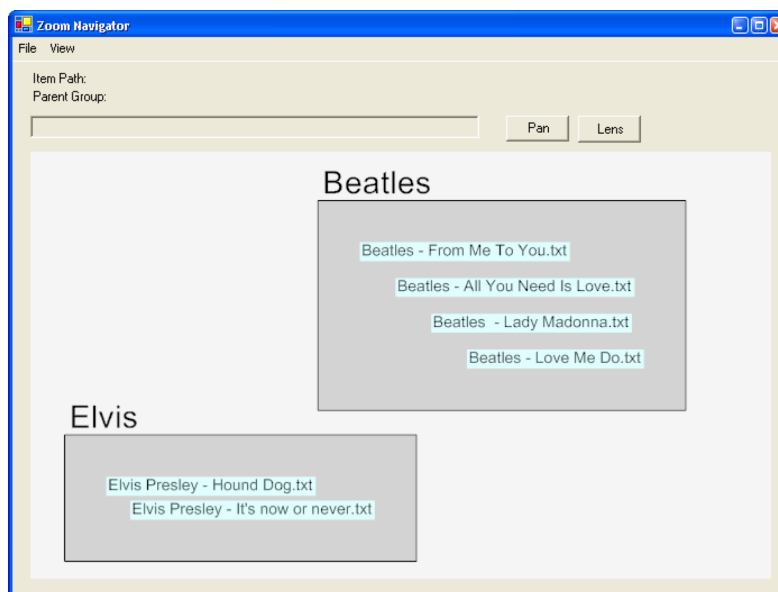

Figura 9 – Interface de navegação - grupos e itens.

PUC-Rio - Certificação Digital Nº 0410858/CA

Quando item é clicado pela segunda vez, passa a ser exibido a sua forma integral.

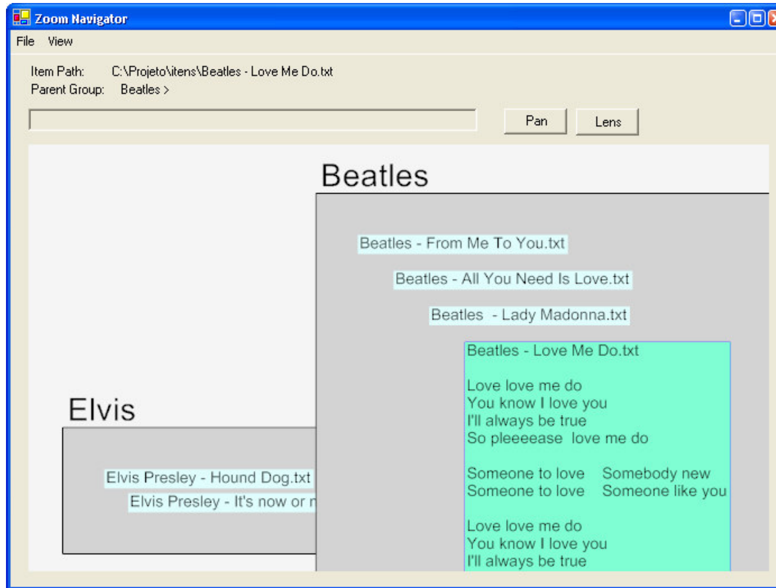

Figura 10 – Interface de navegação – visualizando item.

Ao clicar no botão *Lens* uma nova janela se abre para configurar as opções de filtragem.

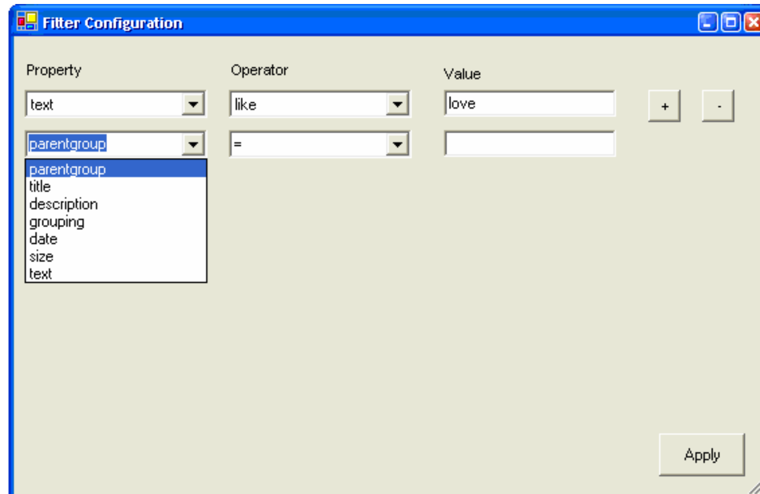

Figura 11 – Interface de navegação – Selecionando filtros da lente.

PUC-Rio - Certificação Digital Nº 0410858/CA

A lente pode ser dragada e ampliada. Por ela são exibidos apenas os itens que atendem as condições de filtragem.

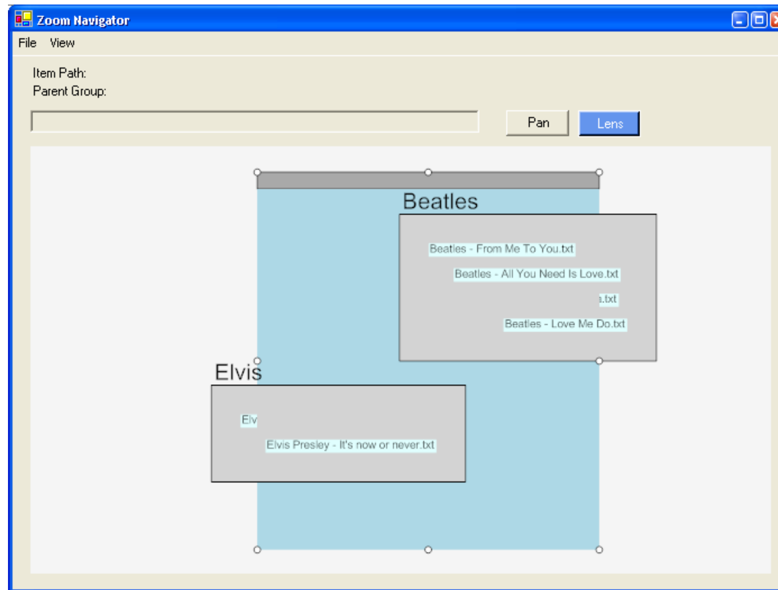

Figura 12 – Interface de navegação com lente de filtragem.

PUC-Rio - Certificação Digital Nº 0410858/CA