# 4
# Exemplos de uso

Para o melhor entendimento dos conceitos expostos no capítulo anterior, são apresentadas aqui duas aplicações que ilustram o poder da DSL criada e a facilidade de utilizar as novas funcionalidades para a manipulação de objetos e conjuntos. Também será descrita a aplicação que foi gerada a partir do modelo MVC.

## 4.1
## Aplicação Demo

Foi criada uma aplicação para demonstrar todas as operações disponíveis no ambiente de programação para o usuário. Foi desenvolvida em Asp.Net e utilizou as definições de classes e objetos listadas no Apêndice III.

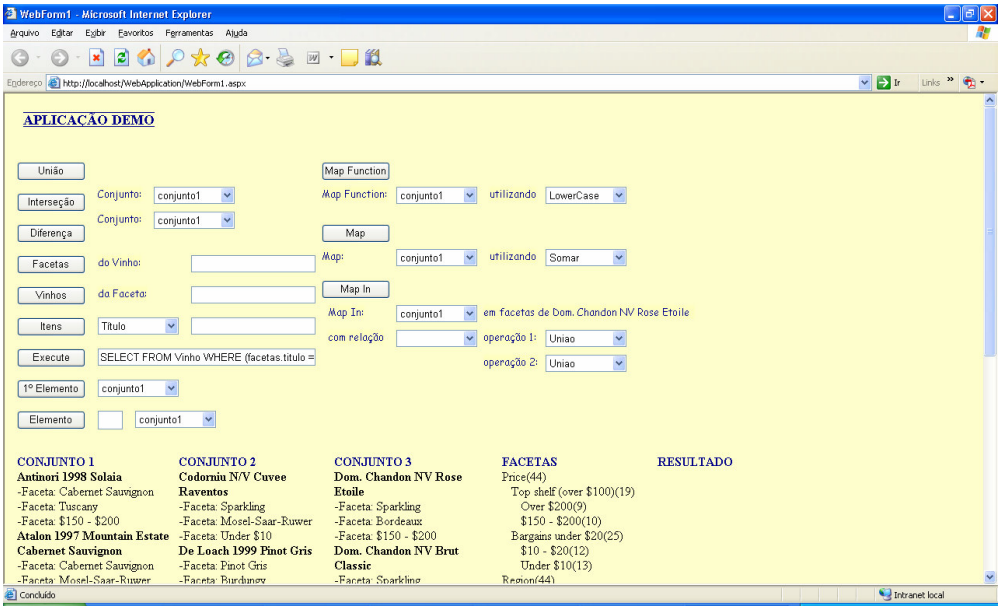


Figura 28: Tela da aplicação Demo

Foram disponibilizados 3 conjuntos definidos previamente através de consultas realizadas no repositório de elementos.

```
Set conjunto1 = Vinho.findSetBy("facetas.titulo", "Cabernet", "titulo", "ASC");

Set conjunto2 = Vinho.findSetBy("((numeroserie = 2) OR (facetas.titulo =
'Burgundy'))");

Set conjunto3 = Vinho.findSetBy("titulo", "V15");
```

Quadro 22: Definição dos 3 conjuntos utilizados na aplicação de teste

O primeiro conjunto foi definido fazendo uma consulta que retorna todos os *vinhos* que estão relacionados à categoria *Cabernet*, ordenados por *título* em ordem ascendente. O segundo conjunto está formado por todos os *vinhos* que tenha o número de séria igual a 2 ou que sejam da categoria *Burgundy*. E por fim, o terceiro conjunto foi definido pelo *vinho* que tem o *título* V15.

O usuário pode então realizar tipos de operações envolvendo estes conjuntos, *união* que irá fazer a união entre os elementos de dois conjuntos escolhidos; *interseção* que irá fazer a interseção entre os elementos de dois conjuntos escolhidos e *diferença* que irá fazer a diferença entre os elementos de dois conjuntos escolhidos.

Outra operação interessante: dado um *vinho* mostrar as categorias ao qual este está associado, assim como dada uma *categoria* mostrar os vinhos que esta contém. Pode-se também encontrar elementos similares a outros, como por exemplo, todos os *vinhos* que tenham sido fabricados em um mesmo ano.

É possível aplicar funções sobre conjuntos. Dada uma função definida pelo usuário, aplicá-la sobre todos os elementos de um conjunto, como por exemplo, duplicar o preço de todos os *vinhos* ou calcular o somatório de todos os preços dos *vinhos* de um conjunto que contém os *vinhos* do tipo *Sauvignon*.

Este tipo de método pode ser caracterizado de duas formas. Na primeira é passada como parâmetro a função a ser aplicada a cada elemento de um conjunto, alterando-os, onde o resultado é o conjunto modificado. Como ilustração, vejamos o exemplo abaixo:

```
/// <summary>
/// Método que será utilizado pelo delegate para ser aplicado em todos os
elementos de um conjunto.
/// Função para transformar os títulos dos vinhos para lowercase.
/// </summary>
/// <param name="x">objeto ao qual será aplicada a função.</param>
/// <returns>objeto após ser aplicada a função.</returns>
private object LowerCase(object x)
{
        Vinho vinho = (Vinho) x;
        Vinho vinhoResposta = vinho;
        vinhoResposta.Titulo = vinhoResposta.Titulo.ToLower();
        return vinhoResposta;
}
```

Quadro 23: Método de exemplo para ser utilizado no MapSet.

Este método modifica os títulos dos vinhos deixando-os em caixa baixa. Após a sua definição, ele é utilizado como parâmetro para o método FunctionMapSet, que por sua vez aplicará a função definida acima em todos os elementos do *conjunto1*, veja abaixo:

```
conjunto1.MapSet(new Set.FunctionMapSet(LowerCase));
```

Quadro 24: Método MapSet.

Na segunda forma, um método também é definido, mas este será aplicado no conjunto como um todo, ou seja, não teremos um conjunto como resultado e sim um valor escalar, por exemplo, vejamos o método que segue:

```
/// <summary>
/// Método que será utilizado pelo delegate para ser aplicado em todos os
elementos de um conjunto.
/// Função para calcular o somatório de todos os números de série os vinhos.
/// </summary>
/// <param name="x">objeto ao qual será aplicada a função.</param>
/// <returns>objeto após ser aplicada a função.</returns>
private object Sum(object x, object y)
{
        int temp1, temp2;
        if (x is Vinho)
                temp1 = ((Vinho) x).Numeroserie;
        else
        {
                if (x != null)
                        temp1 = (int) x;
                else
                        temp1 = 0;
        }
        if (y is Vinho)
                temp2 = ((Vinho) y).Numeroserie;
        else
        {
                if (y != null)
                        temp2 = (int) y;
                else
                        temp2 = 0;
```

```
        }
                return temp1 + temp2;
}
```

Quadro 25: Método de exemplo para ser utilizado no Map.

Este método faz o somatório do número de série de todos os elementos de um conjunto. É passado como parâmetro para o método FunctionMap:

```
string concat = (string)conjunto1.Map(new Set.FunctionMap(Concat));
```

Quadro 26: Método Map.

Está também disponível para o usuário programador um método que mapeia um conjunto em outro, sendo estes diferentes entre si. Por exemplo, mapear um conjunto de Pessoas em um conjunto de Fotos. Para isso o usuário deve definir qual o conjunto de origem e o de destino, então a relação desejada entre os 2 conjuntos, por exemplo Pessoa possui Foto. E por fim as operações que serão executadas sobre os conjuntos. O método encontra todas as Fotos que uma Pessoa possui, e assim é necessário definir qual a operação que será utilizada para juntar as fotos de cada pessoa (união, interseção, etc.) e depois a operação que irá fazer a interação entre as fotos das pessoas e o conjunto de destino, que contém fotos.

Para colaborar com essa funcionalidade, há também um método que dado um conjunto em que seus elementos são do mesmo tipo, é possível saber todas as relações que estes mantêm com outros elementos.

Vejamos as assinaturas para os métodos mencionados anteriormente:

```
public Set MapIn(Set sourceSet, Set targetSet, string relationName, string
firstOperation, string secondOperation)

public ArrayList ReturnRelations()
```

Quadro 27: Método MapIn e ReturnRelations.

Dessa forma pode-se pensar na utilização desta funcionalidade, por exemplo, ao tentar fazer uma interação em uma interface gráfica entre 2 conjuntos onde cada um contém diferentes tipos de objetos. Nesse momento o usuário final pode definir qual o conjunto de origem, qual a relação que deseja estar utilizando, para os objetos de origem, qual o conjunto de destino e as operações entre os

elementos encontrados da relação no conjunto de origem e entre este conjunto encontrado e o conjunto de destino.

Por fim devemos ressaltar uma funcionalidade de consulta onde o usuário programador entra com uma *query*, similar à linguagem SQL aceita pelo SQL Server da Microsoft. O método desenvolvido para tal retorna um conjunto com os elementos que atendem às restrições definidas nas *query*.

Esta linguagem tem a seguinte sintaxe:

```
SELECT FROM <tipo do objeto> {WHERE (<campo> = <valor>) [AND (<campo> = <valor>)] [OR (<campo> = <valor>)]} {ORDER BY <campo>}.
```

Quadro 28: Linguagem de consulta.

Primeiramente devemos definir qual o tipo de objeto que queremos consultar, e então se é necessário uma restrição, como por exemplo, que o tipo do vinho seja Cabernet ou que o preço seja menos que 100. E se necessário retornar os elementos ordenados ascendentemente ou descendentemente.

Vejamos alguns exemplos de utilização:

```
Sets sets = Sets.Open(true);

Set.Set resultSet1 = sets.ExecuteQuery("SELECT FROM Vinho WHERE (facetas.titulo = 'Cabernet') ORDER BY titulo DESC");

Set.Set resultSet2 = sets.ExecuteQuery("SELECT FROM Faceta WHERE ((titulo = 'Varietal') OR (titulo = 'Region') OR (titulo = 'Price'))");

sets.Close();
```

Quadro 29: Exemplos de utilização do método ExecuteQuery.

Na primeira consulta estamos buscando pelos vinhos do tipo Cabernet, ou seja, que estão associados à faceta cujo título é Cabernet. E requisita-se também que estes vinhos estejam ordenados por título de forma descendente.

Já na segunda consulta, estamos buscando pelas facetas que tenham como título "Varietal" *ou* "Region" *ou* "Price".

A versão desta linguagem desenvolvida nesta dissertação necessita que sempre as cláusulas do *WHERE* estejam parentetisadas corretamente. Não há um verificador para validar a sintaxe.

## 4.2
## Aplicação Teste – Navegação Facetada

Esta aplicação define uma interface de navegação baseada no modelo proposto. Todas as informações são interpretadas como elementos e seus agrupamentos como conjuntos. Também foi desenvolvida utilizando Asp.Net.

O usuário deve definir o RDF com os elementos que serão utilizados pela aplicação. Devem ser definidas as categorias e os elementos e cada um deve estar associado a uma ou mais categorias. O exemplo abaixo utiliza o modo de navegação conhecido como navegação facetada, citada anteriormente. Os elementos são agrupados em categorias organizadas hierarquicamente. À medida que o usuário navega por estas categorias, os itens são filtrados e retornados ao mesmo. O usuário também pode retirar uma categoria já selecionada, diminuindo assim as restrições da busca feita.

A aplicação está preparada para aceitar outras definições de elementos e gerar a interface automaticamente, servindo como um exemplo concreto do que o modelo é capaz. Neste exemplo apenas a operação de *interseção* está sendo utilizada, pois como foi visto, as categorias selecionadas pelo usuário são utilizadas como restrições para os elementos a serem retornados.

A imagem que segue é um exemplo que foi gerado a partir do RDF de definição de elementos que se encontra no apêndice III deste documento.

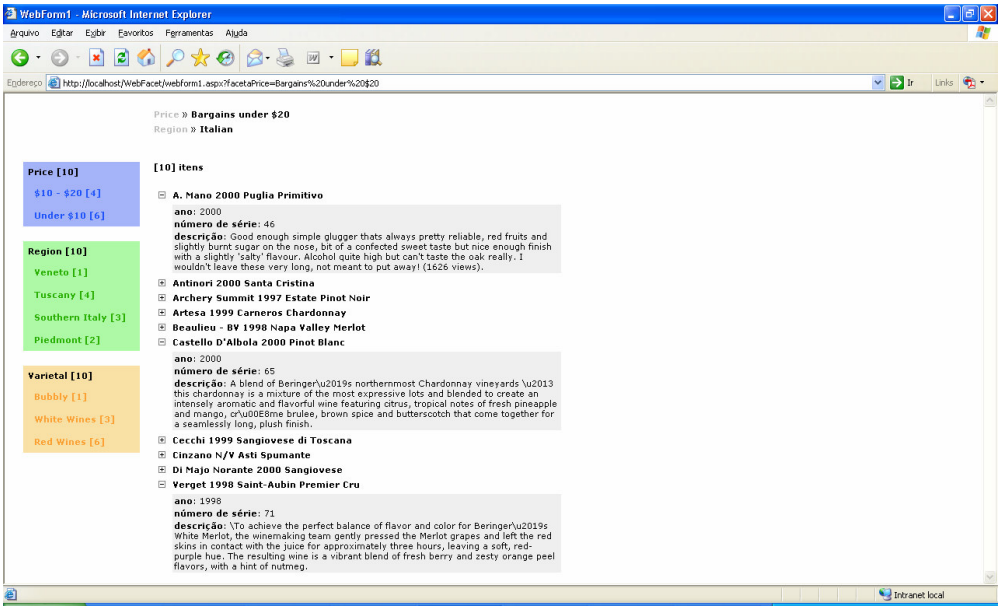


Figura 29: Tela da aplicação Teste

A interface gerada está formada por 3 categorias principais: *Price*, *Region* e *Varietal*. Cada uma desta está formada por outras categorias. Os elementos definidos estão associados a uma ou mais categorias.

No exemplo dado, o usuário já incluiu duas restrições: Region > Italian e Varietal > White Wine, que podem ser facilmente removidas. Para isso basta clicar sobre a restrição desejada.

Quando uma categoria é escolhida para compor uma nova restrição, um novo grupo é gerado utilizando o grupo atualmente definido por uma eventual busca já realizada e a categoria selecionada. Então é executada a operação de *interseção* para que seja obtido o novo conjunto resposta com os *vinhos* que atendem a todas as restrições já definidas.

## 4.3
## Aplicação resultante do uso do modelo MVC

Neste exemplo temos uma aplicação resultado do uso do modelo MVC descrito no capítulo anterior. Foi utilizado o framework Zoom Navigator como o componente da interface e o modelo de informação criado nesta dissertação e sua DSL como o componente do modelo.

Esta aplicação ilustra a capacidade que os dois componentes desenvolvidos têm de se comunicar e interagir entre si para poderem formar uma aplicação típica MVC. A interface disponibiliza que diferentes tipos de manipulações direta sejam realizadas e a semântica destas operações podem ser definidas pelo usuário que irá utilizar o sistema.

A semântica a ser definida utiliza em *background* a DSL desenvolvida sobre o modelo de informação basado em conjuntos. Desta forma o usuário pode utilizar todas as funcionalidades presentes na DSL como semântica para as operações realizadas na interface. Por exemplo, definir que ao arrastar um grupo para cima de outro, estará realizando uma união entre eles.

Para que o modelo MVC pudesse ser implementado, foi necessária a criação do terceiro componente, o *controller*, que é responsável por essa troca de informações entre a interface e o modelo que esta utiliza. Este componente é responsável pelo mapeamento dos dados entre duas aplicações distintas, utilizando diferentes estruturas de dados para representar seus elementos. Quando uma ação é executada na interface, esta informa ao modelo o que deve ser realizado, e este então retorna a resposta do que foi solicitado. Por fim, a interface interpreta o retorno e modifica seus elementos na tela para que reflitam o que foi calculado.

Esta interação entre os dois componentes é constante, e quanto mais esta ocorrer, melhor estarão integrados e funcionando como um puro modelo MVC. Na aplicação de teste que foi desenvolvida, todas as operações de adicionar, editar, remover grupos e itens estão contempladas. Assim como algumas manipulações de drag & drop de grupos e itens e também copiar e colar. Para cada uma destas operações é possível definir em um XML de configuração sua semântica.

Abaixo segue a imagem capturada em uma interação no sistema:

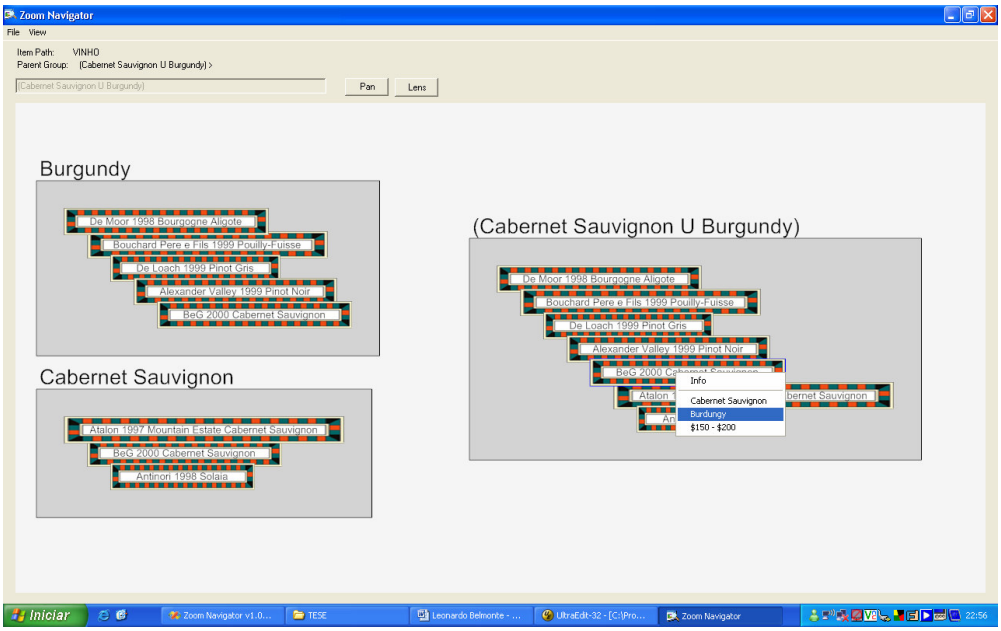


Figura 30: Aplicação resultante do uso do modelo MVC

Na tela acima, inicialmente foram adicionados dois conjuntos, [1] *Burgundy* e [2] *Cabernet* que representam os vinhos da região *Burgundy* e do tipo *Cabernet*, respectivamente.

Depois foi realizada uma operação de interseção entre os conjuntos, obtendo como resposta o conjunto [3] que contém somente os vinhos *V12* e *V2*, vinhos estes que estão presentes em ambos os conjuntos, respeitando a teoria dos conjuntos.

É possível também a partir de um elemento identificar aos quais outros grupos ele pertence. No exemplo, ao clicar no elemento com o botão direito do mouse, podemos ver suas informações e todos os grupos nos quais está contido e ao clicar em um destes grupos, é gerado um novo conjunto com todos os elementos que percentem a ele.

Inúmeras outras interações poderiam ser desenvolvidas, como por exemplo, selecionar alguns itens de um conjunto e aplicar alguma função para alterar os títulos dos elementos para caixa alta ou então somar aum valor nos seus preços. Estas outras interações ficaram para trabalhos futuros.