4 Arquitetura do *Framework*

4.1.Diagramas do SHDM .Net

A utilização de um diagrama para representar um modelo SHDM deve ser acompanhada de algum benefício imediato. Nos diagramas do SHDM .Net foram introduzidas abstrações na definição dos elementos e também alguma semântica baseada no posicionamento dos objetos no modelo. Foram retiradas repetições nas definições dos elementos ocasionadas, principalmente, pela definição textual, ou seja, pela ausência de posicionamento espacial introduzido pela representação diagramática. Todos os modelos do SHDM .Net já herdam, automaticamente, um pacote de propriedades gráficas pré-existente em qualquer modelo desenvolvido utilizando o Microsoft DSL-Tools. Estas propriedades e funções de layout visual são utilizadas para automatizar algumas definições, como definições baseada no posicionamento dos objetos, por exemplo. Esta automatização foi desenvolvida através da interpretação dos dados objetos visuais que compõem o diagrama.

4.2.Diagrama de Classes Navegacionais

No SHDM o diagrama de classes navegacionais é uma visão sobre o modelo conceitual. Ele é desenvolvido baseado nas classes e relacionamentos definidos no modelo conceitual de classes.

No SHDM .Net a estrutura das classes é definida diretamente no diagrama de classes navegacionais, ou seja, não existe o mapeamento das classes navegacionais em classes conceituais. O modelo de classes navegacionais do SHDM .Net engloba propriedades do modelo conceitual (OO) e são adicionadas as primitivas navegacionais definidas no OOHDM/SHDM.

O metamodelo do Diagrama de Classes Navegacionais é Apresentado na Figura 9.

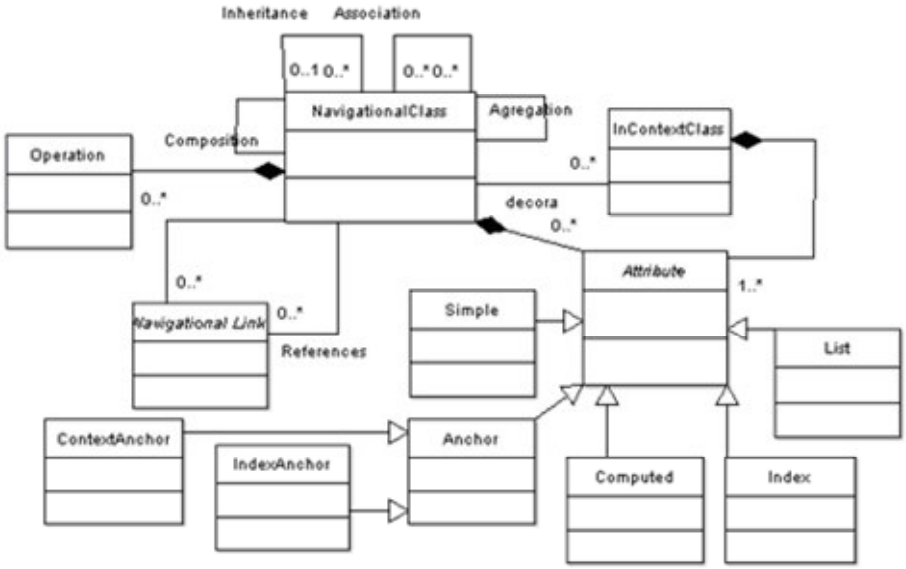

Figura 9 – Metamodelo do Diagrama de Classes Navegacionais do SHDM

No modelo do diagrama de classes navegacionais definido no DSL-Tools, apresentado na Figura 10, estão definidas e delimitadas todas as possibilidades de diagramação contidas em um diagrama de classes navegacionais. Conforme pode ser observado, o modelo do diagrama feito no DSL-Tools difere um pouco do metamodelo do SHDM . Isto ocorre devido a algumas limitações de representação na interface gerada pelo DSL-Tools ou por alguma melhoria ou otimização na diagramação do modelo de classes navegacionais proporcionada pelo ambiente de implementação. As classes que compõem o modelo do diagrama de classes navegacionais são descritas nos itens abaixo.

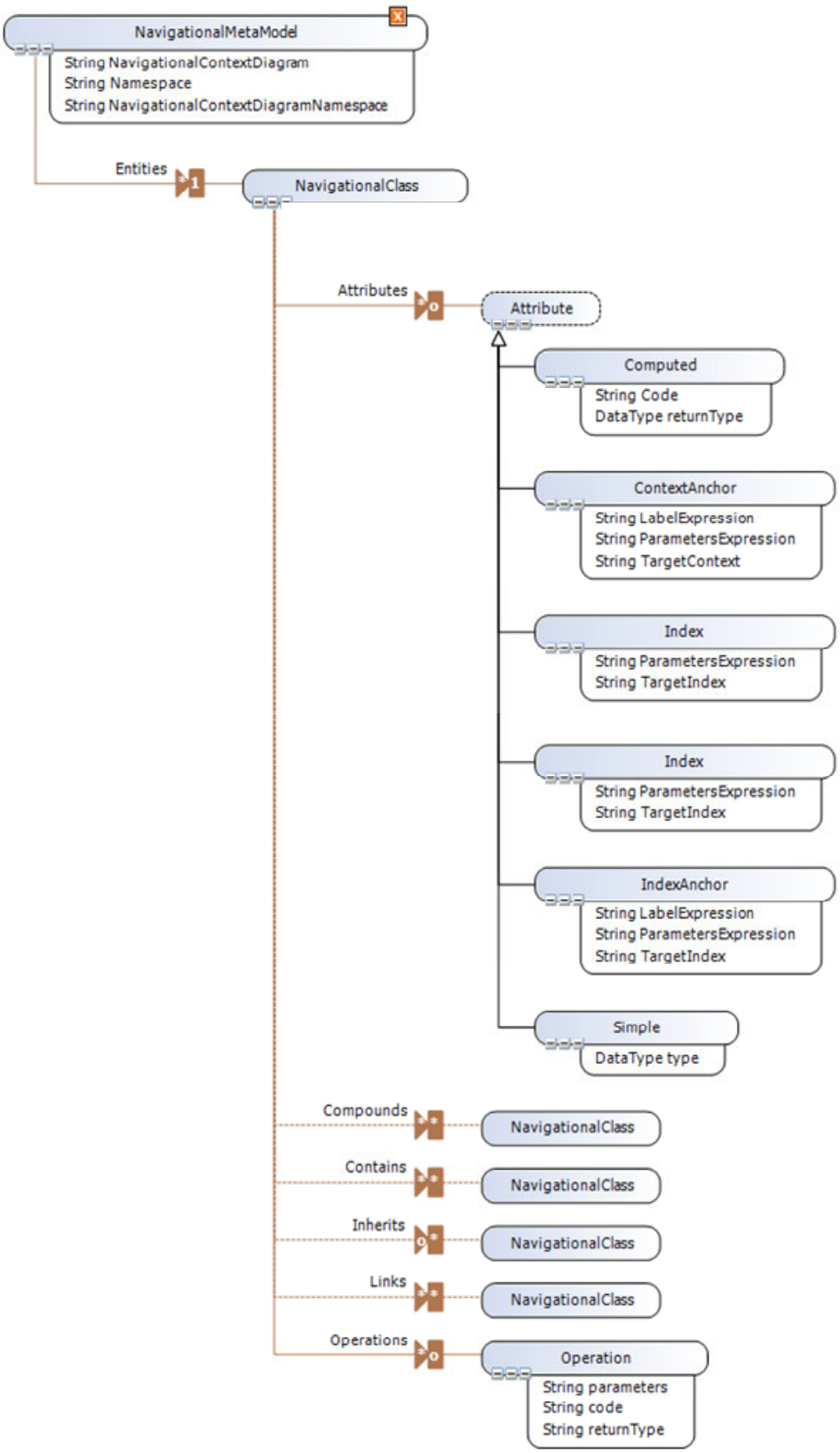

Figura 10 – Modelo do diagrama de classes navegacionais

4.2.1. Classe NavigationalMetaModel

A classe NavigationalMetaModel é responsável por armazenar todas as instâncias das classes NavigationalClass que compõem uma instância de um Diagrama de Classes Navegacionais.

Esta classe tem os seguintes atributos:

Namespace – identificação do namespace ou pacote onde serão geradas as classes C# referentes a cada classe navegacional do diagrama.

NavigationalContextDiagram – armazena o nome do diagrama de contextos navegacionais que será referenciado pelo diagrama de classes navegacionais. Esta referência é utilizada quando um atributo do tipo ContextAnchor, Index ou IndexAnchor é adicionado a uma NavigationalClass.

NavigationalContextDiagramNamespace – atributo não editável que recupera o namespace utilizado no diagrama de contextos navegacionais selecionado no atributo NavigationalContextDiagram.

4.2.1.1.Relacionamento Entities

O relacionamento Entities é um relacionamento de composição entre a classe NavigationalMetaModel e a classe NavigationalClass. A semântica deste relacionamento expressa que um NavigationalMetaModel é composto de NavigationalClasses.

Este relacionamento é representado na Classe NavigationalMetaModel através do atributo Entities. Este atributo, que tem o mesmo nome do relacionamento, armazena a coleção de classes NavigationalClass que compõe uma instância de um Diagrama de Classes Navegacionais.

4.2.2.Classe NavigationalClass

A classe NavigationalClass é a classe responsável por representar as classes navegacionais, ou seja, toda classe navegacional é uma instância da classe NavegationalClass. Esta classe tem apenas o atributo name que é responsável por armazenar o nome da classe navegacional.

A classe NavigationalClass tem diversos relacionamentos que serão descritos a seguir:

4.2.2.1.Relacionamento Attributes

Este relacionamento é equivalente ao relacionamento entre a metaclasse NavigationalClass e a metaclasse Attribute no metamodelo de classes navegacionais Figura 9. A semântica deste relacionamento expressa que uma Classe Navegacional é composta de atributos.

Este relacionamento não possui atributos, e é representado na Classe NavigationalClass através do atributo Attributes. Este atributo, que tem o mesmo nome do relacionamento, armazena a coleção de classes Attribute que compõe uma instância da uma Classe Navegacional.

4.2.2.2.Relacionamento Operations

O relacionamento Operations é equivalente ao relacionamento entre a metaclasse NavigationalClass e a metaclasse Operation no metamodelo de classes navegacionais Figura 9. A semântica deste relacionamento expressa que uma Classe Navegacional é composta de operações.

Este relacionamento não possui atributos, e é representado na Classe NavigationalClass através do atributo Operations. Este atributo, que tem o mesmo nome do relacionamento, armazena a coleção de classes Operation que compõe uma instância da uma Classe Navegacional.

4.2.2.3.Relacionamento Links

O relacionamento Links é um relacionamento de associação entre duas classes navegacionais. Este relacionamento é representado na Figura 9 pelo auto-relacionamento "Association" existente na metaclasse NavigationalClass. A semântica deste relacionamento é equivalente a semântica do relacionamento de associação do UML.

Este relacionamento é representado na Classe NavigationalClass através do atributo Links. Este atributo, que tem o mesmo nome do relacionamento, armazena a coleção de classes NavigationalClass com as quais uma instância de classe navegacional se relaciona através de relacionamentos de associação. Este relacionamento tem os seguintes atributos:

SourceMultiplicity: Cardinalidade da origem do relacionamento, os possíveis valores são: Zero_One, One, Zero_Many,One_Many

TargetMultiplicity: Cardinalidade do destino do relacionamento, os possíveis valores são: Zero_One, One, Zero_Many,One_Many

SourceName: Nome do atributo gerado no lado da origem do relacionamento. Caso este nome seja deixado em branco, atributo contendo a classe de destino não será criado na origem.

TargetName: Nome do atributo gerado no destino do relacionamento. Caso este nome seja deixado em branco, atributo contendo a classe de origem não será criado no destino.

Name: Nome do relacionamento

4.2.2.3.1.Relacionamento Compounds

O relacionamento Compounds é um relacionamento de associação entre duas classes navegacionais. Este relacionamento é representado na Figura 9 pelo auto-relacionamento "Composition" existente na metaclasse NavigationalClass. A semântica deste relacionamento é equivalente a semântica do relacionamento de composição do UML.

As composições são associações que representam agregações muito fortes. Isto significa que as composições formam também relações do 'todo' para as partes, mas a relação é tão forte que as partes não podem existir por si só. Elas só existem dentro do todo e se o todo for destruído, as partes desaparecem também.

Este relacionamento é representado na Classe NavigationalClass através do atributo Compounds. Este atributo, que tem o mesmo nome do relacionamento, armazena a coleção de classes NavigationalClass com as quais uma instância de classe navegacional se relaciona através de relacionamentos de composição. O relacionamento Compounds tem os seguintes atributos:

TargetMultiplicity: Cardinalidade do destino do relacionamento, os possíveis valores são: Zero_One, One, Zero_Many,One_Many

SourceName: Nome do atributo gerado no lado da origem do relacionamento. Caso este nome seja deixado em branco, atributo contendo a classe de destino não será criado na origem.

TargetName: Nome do atributo gerado no destino do relacionamento. Caso este nome seja deixado em branco, atributo contendo a classe de origem não será criado no destino.

Name: Nome do relacionamento

4.2.2.3.2. Relacionamento Contains

O relacionamento Contains é um relacionamento de associação entre duas classes navegacionais. Este relacionamento é representado na Figura 9 pelo auto-relacionamento “Aggregation” existente na metaclasse NavigationalClass. A semântica deste relacionamento é equivalente à semântica do relacionamento de agregação do UML.

A agregação é um tipo especial de associação, na qual duas classes participantes não têm um estado igual, mas tem uma relação “do 'todo' para as partes”. Uma agregação diz como é que a classe que tem o papel do 'todo' é composta (ou tem) as outras classes, que têm o papel das partes. Para as agregações, a classe que atua como o 'todo' tem sempre uma multiplicidade de um.

Este relacionamento é representado na Classe NavigationalClass através do atributo Contains. Este atributo, que tem o mesmo nome do relacionamento, armazena a coleção de classes NavigationalClass com as quais uma instância de classe navegacional se relaciona através de relacionamentos de agregação. O relacionamento Contains tem os seguintes atributos:

TargetMultiplicity: Cardinalidade do destino do relacionamento, os possíveis valores são: Zero_One, One, Zero_Many,One_Many

SourceName: Nome do atributo gerado no lado da origem do relacionamento. Caso este nome seja deixado em branco, atributo contendo a classe de destino não será criado na origem.

TargetName: Nome do atributo gerado no destino do relacionamento. Caso este nome seja deixado em branco, atributo contendo a classe de origem não será criado no destino.

Name: Nome do relacionamento

4.2.2.3.3.Relacionamento Inherits

O relacionamento Inherits é um relacionamento de generalização entre duas classes navegacionais. Este relacionamento é representado na Figura 9 pelo auto-relacionamento "Inherits" existente na metaclasse NavigationalClass. A semântica deste relacionamento é equivalente a semântica do relacionamento de herança do UML.

A herança é um dos conceitos fundamentais da programação orientada a objetos, nos quais uma classe "ganha" todos os atributos e operações da classe que herda, podendo sobrepor ou modificar algumas delas, assim como adicionar mais atributos ou operações próprias.

Este relacionamento não possui atributos, e é representado na Classe NavigationalClass através do atributo Inherits. Este atributo, que tem o mesmo nome do relacionamento, armazena a uma instância de sua superclasse navegacional, ou seja, armazena uma instância da classe "Pai", de quem herda os atributos e comportamentos.

4.2.2.4. Classe Attribute

A classe NavigationalClass está relacionada com a classe abstrata Attribute. A semântica deste relacionamento indica que uma classe navegacional tem atributos. O único atributo da classe Attribute é o "name" que armazena o nome do atributo instanciado.

Os tipos destes atributos podem ser de qualquer classe que seja subclasse da classe abstrata Attribute, conforme demonstrado abaixo:

4.2.2.4.1. Classe Simple

A classe Simple é utilizada para representar atributos com tipo de dado primitivo. O único atributo da classe Simple é o "type" que contém o tipo do dado primitivo. Os tipos de dados considerados primitivos no *framework* SHDM .Net são: String, Text, Html, Xml, Integer, Float, Boolean, Date, Time, DateTime, List, Hash, URL, Image, Email, Movie, Audio, Flash, Applet, Other.

4.2.2.4.2. Classe ContextAnchor

Uma âncora para um contexto é um atributo que contém um apontador para um elemento específico dentro de um contexto previamente definido. Este contexto tem que estar definido no Diagrama de Contextos Navegacionais, que será detalhado no Item 4.3. A classe ContextAnchor tem os seguintes atributos:

LabelExpression: responsável por armazenar o valor texto que será colocado na âncora para o contexto, pode ser um valor string, fixo, delimitado por aspas como, por exemplo “valor”, ou uma expressão na DSL-SHDM que irá, em tempo de execução, obter o valor do dado a ser utilizado como etiqueta da âncora.

ParametersExpression: armazena os parâmetros a serem passados para o contexto de destino da âncora para que seja identificado um nó, unívoco, dentro do contexto. Este atributo apenas é utilizado quando o contexto de destino da âncora recebe parâmetros. Os valores destes parâmetros podem ser expressões na DSL-SHDM, que serão avaliadas em tempo de execução, ou um valor fixo que deverá ser delimitado por aspas. Ex.: “valor”.

TargetContext: atributo que armazena o nome do contexto de destino da âncora. Automaticamente são apresentados os contextos possíveis que estão definidos no Diagrama de Contextos Navegacionais associado ao Diagrama de Classes Navegacionais definido na classe NavigationalMetaModel, descrita no item 4.2.1, através do atributo NavigationalContextDiagram.

A Figura 11 mostra a definição de um atributo do tipo âncora. Neste exemplo o atributo é uma âncora para um nó no contexto StudentByResearchArea.

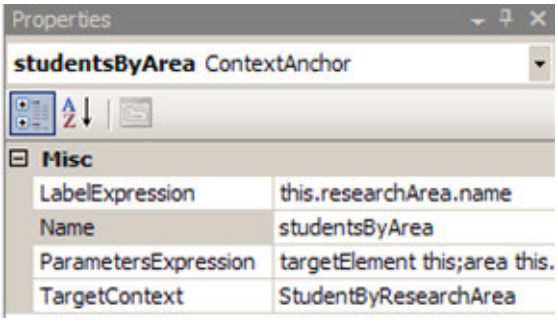

Figura 11 – Propriedades de um atributo ContextAnchor

A Figura 12 mostra um exemplo dos parâmetros passados para o contexto StudentByResearchArea através da âncora definida pelo atributo studentsByArea. Este contexto recebe o nó que será apresentado e a área de pesquisa para que o contexto seja montado. O nó de destino no contexto é identificado pelo parâmetro "targetElement" , que no caso apresentará a mesma instância da classe Student. No parâmetro "area" é passado o valor existente na propriedade *this.researchArea*. esta propriedade armazena a área de pesquisa do aluno.

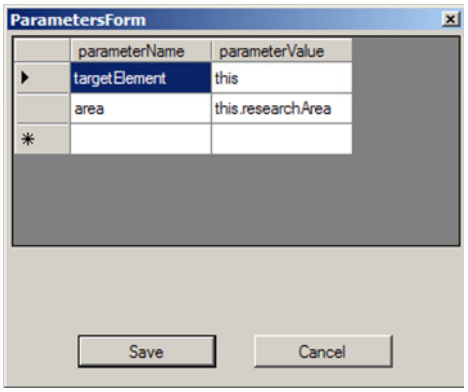

Figura 12 – Parâmetros passados para o contexto StudentByResearchArea

4.2.2.4.3. Classe IndexAnchor

Uma âncora para um índice é um atributo que contém um link para um índice previamente definido. Este índice tem que estar definido no Diagrama de Contextos Navegacionais, que será detalhado no Item 4.3. A classe IndexAnchor tem os seguintes atributos:

LabelExpression: responsável por armazenar o valor texto que será colocado na âncora para o índice, pode ser um valor string, fixo, delimitado por aspas como, por exemplo "valor", ou uma expressão na DSL-SHDM que irá, em tempo de execução, obter o valor do dado a ser utilizado como etiqueta da âncora.

ParametersExpression: armazena os parâmetros a serem passados para o índice de destino da âncora para que seja identificado um nó, unívoco, dentro do índice. Este atributo apenas é utilizado quando o índice de destino da âncora recebe parâmetros. Os valores destes parâmetros podem ser expressões na DSL-SHDM, que serão avaliadas em tempo de execução, ou um valor fixo que deverá ser delimitado por aspas. Ex.: "valor".

TargetIndex: atributo que armazena o nome do índice de destino da âncora. Automaticamente são apresentados os índices possíveis que estão definidos no

Diagrama de Contextos Navegacionais associado ao Diagrama de Classes Navegacionais definido na classe NavigationalMetaModel, descrita no item 4.2.1, através do atributo NavigationalContextDiagram.

A Figura 13 mostra a definição de um atributo do tipo âncora. Neste exemplo o atributo é uma âncora para um índice StudentsByProfessor.

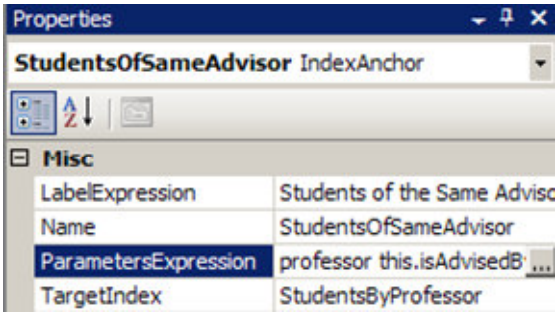

Figura 13 – Propriedades de um atributo IndexAnchor

A Figura 14 mostra um exemplo dos parâmetros passados para o índice StudentsByProfessor através da âncora definida pelo atributo StudentsOfSameAdvisor. Este índice recebe um parâmetro *professor*. Baseado no professor passado por parâmetro, são obtidos seus orientados.

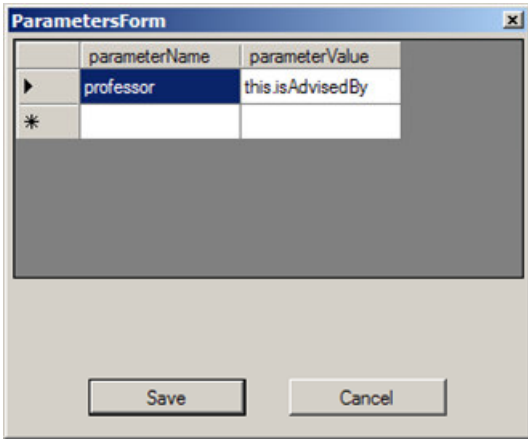

Figura 14 - Parâmetros passados para o índice StudentsByProfessor

4.2.2.4.4.Classe Index

É um atributo que contém um conjunto de elementos que formam um índice, ou seja, este atributo comporta um índice de acesso e não apenas uma âncora para um índice. Este índice tem que estar definido no Diagrama de Contextos Navegacionais, que será detalhado no Item 4.3. A classe IndexAnchor tem os seguintes atributos:

ParametersExpression: armazena os parâmetros a serem passados para o índice de destino da âncora para que seja identificado um nó, unívoco, dentro do índice. Este atributo apenas é utilizado quando o índice de destino da âncora recebe parâmetros. Os valores destes parâmetros podem ser expressões na DSL-SHDM, que serão avaliadas em tempo de execução, ou um valor fixo que deverá ser delimitado por aspas. Ex.: “valor”.

TargetIndex: atributo que armazena o nome do índice de destino da âncora. Automaticamente são apresentados os índices possíveis que estão definidos no Diagrama de Contextos Navegacionais associado ao Diagrama de Classes Navegacionais definido na classe NavigationalMetaModel, descrita no item 4.2.1, através do atributo NavigationalContextDiagram.

A Figura 15 mostra a definição de um atributo do tipo índice. Como pode ser observado este atributo não tem a propriedade Label existente na propriedade dos atributos ContextAnchor e IndexAnchor definidos na Figura 11 e Figura 13, respectivamente. Neste exemplo o atributo é o índice StudentsByProfessor.

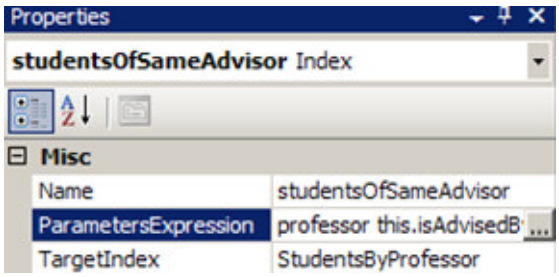

Figura 15 - Propriedades de um atributo Index

A Figura 16 mostra um exemplo dos parâmetros passados para o índice StudentsByProfessor seja criado. Este índice recebe um parâmetro *professor*. Baseado no professor passado por parâmetro, o atributo índice é criado.

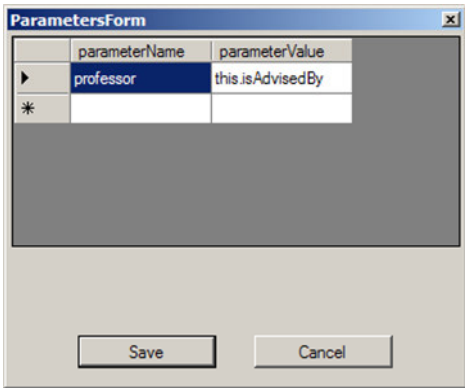

Figura 16 - Parâmetros passados para o índice StudentsByProfessor

4.2.2.4.5. Classe Computed

Um Atributo computado é um atributo que contém um código que é responsável por, em tempo de execução, retornar o valor do atributo. Esta classe tem, além dos atributos herdados, o atributo Code, que é responsável por armazenar o código, escrito na DSL-SHDM, que obtém o valor do atributo. A seguir um exemplo de um código do atributo Code em uma instância da classe Computed. O Exemplo no Quadro 1 retorna o ano semestre de ingresso do aluno baseado na registrationId, ou seja, retorna os três primeiros dígitos do registrationId do aluno.

```
string regID = this.registrationId.ToString();
return regID.Substring(0, 3);
```

Quadro 1 – Exemplo de preenchimento do atributo code na classe Computed

4.2.3. Classe Operation

A Classe Operation é responsável por definir operações nas classes navegacionais onde são criadas.

Esta classe tem os seguintes atributos:

Name: Nome da função, sem os parâmetros.

Params: parâmetros recebidos pela função, estes parâmetros devem ser descritos na forma de Tipo do Parâmetro e nome do parâmetro separado por um espaço em branco. Entre os parâmetros utiliza-se vírgula como separador. Exemplo: uma função que recebe como parâmetro dois valores inteiros para que

seja efetuada a soma destes valores deverá ter o seguinte valor para o atributo params: int valor1, int valor2.

Code: código do corpo da função escrito na DSL-SHDM.

returnType: tipo retornado na chamada da função, caso a função não retorne nenhum valor, deverá ser colocada a palavra “void” no campo de retorno.

Na Figura 17, temos um diagrama de classes navegacionais desenvolvido no editor que produzido pelo Microsoft DSL Tools após a interpretação do modelo previamente apresentado na Figura 10.

Neste diagrama temos um cenário onde professores e estudantes são subclasses de pessoas; um professor orienta estudantes, e professores e estudantes trabalham em áreas de pesquisa.

Quando o diagrama exemplificado na Figura 17 tiver sido criado, uma parte da DSL-SHDM é produzida, contendo todas as classes existentes no modelo criado pelo usuário.

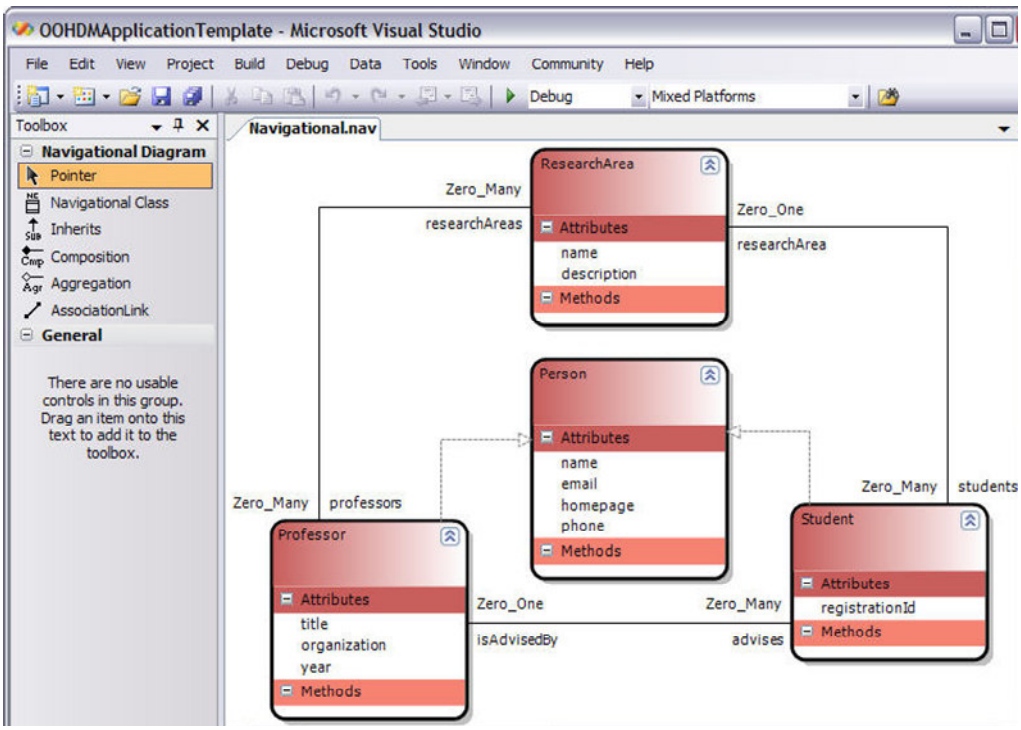

Figura 17 – Exemplo de diagrama de classes navegacionais

4.3.Diagrama de Contextos Navegacionais

O Diagrama de Contextos Navegacionais tem por objetivo representar as alternativas navegacionais da aplicação, ou seja, quais são os possíveis caminhos

que o usuário pode tomar em cada passo da navegação. O Diagrama de Contextos Navegacionais tem basicamente dois elementos – Estruturas de Acesso e Contextos e o relacionamento entre eles. Estruturas de Acesso são índices, que contém âncoras para Contextos ou outras Estruturas de Acesso. Contextos são responsáveis por representar grupos de objetos que são manipulados pelo usuário. Cada Contexto tem diversas propriedades, como estrutura de navegação interna, ponto de entrada e restrições de acesso. Todas estas informações são detalhadas em um cartão de especificação. No SHDM .Net as informações existentes nos cartões de especificação são representadas através de propriedades nos elementos gráficos do diagrama. O diagrama de contextos navegacionais tem um terceiro elemento: o Escopo. Este elemento não tem uma função semântica específica. Sendo apenas utilizado para agrupar relacionamentos comuns entre contextos com o objetivo de simplificar e retirar a repetição do diagrama. O modelo do Diagrama de Contextos Navegacionais é apresentado na Figura 19.

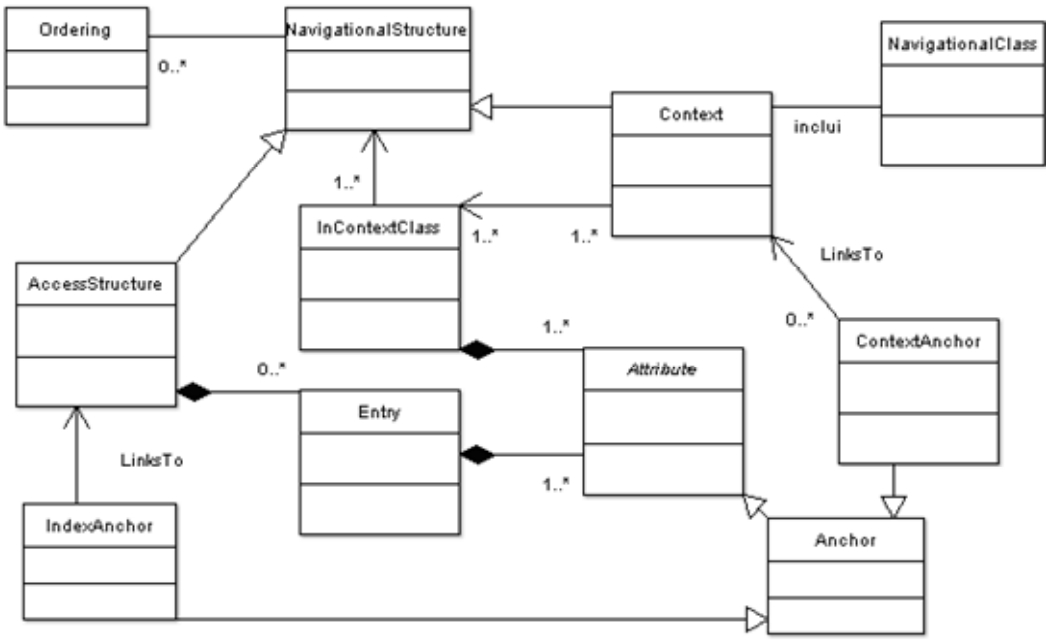

Figura 18 – Metamodelo de Contextos Navegacionais

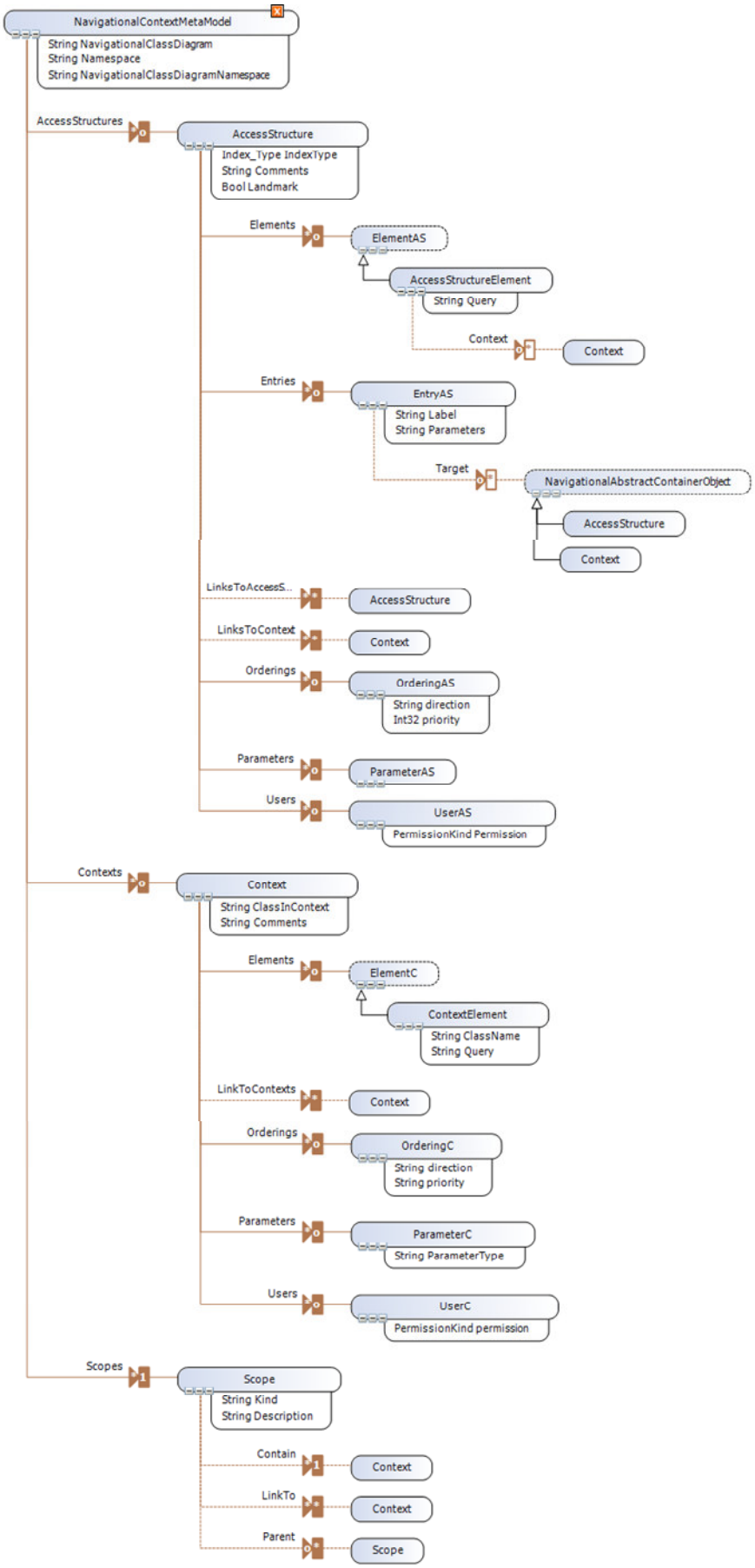

Figura 19 – Modelo do diagrama de contextos navegacionais

4.3.1.Classe NavigationalContextMetaModel

A classe NavigationalContextMetaModel é responsável por armazenar todas as instâncias das classes Context, AccessStructure e Scope que compõem uma instância de um Diagrama de Contextos Navegacionais.

Esta classe tem os seguintes atributos:

Namespace – identificação do namespace ou pacote onde serão geradas as classes C# referentes a cada classe do diagrama de contextos navegacionais.

NavigationalClassDiagram – armazena o nome do diagrama de classes navegacionais que será referenciado pelo diagrama de classes navegacionais.

4.3.1.1.Relacionamento AccessStructures

O relacionamento AccessStructures é um relacionamento de composição entre a classe NavigationalContextMetaModel e a classe AccessStructure. A semântica deste relacionamento expressa que um NavigationalContextMetaModel é composto de AccessStructures.

Este relacionamento é representado na Classe NavigationalMetaModel através do atributo AccessStructures. Este atributo, que tem o mesmo nome do relacionamento, armazena a coleção de classes AccessStructure que compõe uma instância de um Diagrama de Contextos Navegacionais.

4.3.1.2.Relacionamento Contexts

O relacionamento Contexts é um relacionamento de composição entre a classe NavigationalContextMetaModel e a classe Context. A semântica deste relacionamento expressa que um NavigationalContextMetaModel é composto de Contexts.

Este relacionamento é representado na Classe NavigationalMetaModel através do atributo Contexts. Este atributo, que tem o mesmo nome do relacionamento, armazena a coleção de classes Context que compõe uma instância de um Diagrama de Contextos Navegacionais.

4.3.1.3.Relacionamento Scopes

O relacionamento Scopes é um relacionamento de composição entre a classe NavigationalContextMetaModel e a classe Scope. A semântica deste relacionamento expressa que um NavigationalContextMetaModel é composto de Escopos.

Este relacionamento é representado na Classe NavigationalMetaModel através do atributo Scopes. Este atributo, que tem o mesmo nome do relacionamento, armazena a coleção de classes Scope que compõe uma instância de um Diagrama de Contextos Navegacionais.

4.3.2.Classe AccessStructure

A classe AccessStructure é a classe responsável por representar os índices. Índices são estruturas responsáveis por prover um meio de acesso a uma informação em determinado contexto navegacional. Desta forma, um índice pode ter âncoras para um nó em um determinado contexto ou para outro índice.

A classe AccessStructure tem os seguintes atributos:

IndexType: Atributo responsável por armazenar o tipo do Índice. Os valores possíveis são: SimpleIndex, DynamicIndex e MultipleOrdering

Landmark: Atributo que indica se este índice estará disponível a qualquer momento na aplicação, ou seja, se este índice é um ponto de acesso global. Este atributo pode assumir dois valores: true, caso o índice seja um landmark, e false, caso seja necessário uma navegação anterior até que o índice seja atingido.

4.3.2.1.Classe AccessStructureElement

A classe AccessStructureElement é uma classe que compõe a estrutura de uma estrutura de acesso. Esta classe armazena o contexto base de formação do índice, no caso de um índice baseado em contexto, ou caso seja um índice baseado em consulta, armazena uma expressão de consulta que será responsável por selecionar as entradas do índice. Esta classe está relacionada com a classe AccessStructure através da relação de composição Elements.

4.3.2.1.1.Relacionamento Context

O relacionamento Context é um relacionamento de agregação entre a classe AccessStructureElement e uma classe Context. Este relacionamento é preenchido no caso o índice seja baseado em um contexto.

4.3.2.2.Classe EntryAS

A classe EntryAS é responsável por armazenar o valor que irá aparecer para o usuário quando este visualizar a estrutura de acesso, ou seja, cada informação que aparece dentro de um índice é armazenada em uma instância da classe EntryAS. Uma Entry pode ser representada como um ativador ou âncora para um contexto ou para outro índice, ou pode ser apenas um exibidor. Porém todo índice precisa de, no mínimo, uma EntryAS sendo um atributo do tipo âncora.

A classe EntryAS não tem um mapeamento direto no metamodelo de contextos navegacionais do SHDM. No metamodelo uma Entry armazena todos os valores que compõem um índice, em outras palavras, toda a tabela de dados da estrutura de acesso. Já no modelo do diagrama de contextos navegacionais do SHDM .Net, uma Entry equivale ao titulo de uma coluna da tabela que produzirá o índice. As tuplas da tabelas serão obtidas em tempo de execução.

A classe EntryAS tem os seguintes atributos:

Label: responsável por armazenar o texto que aparecerá para o usuário quando este visualizar o índice. Pode ser preenchido com um valor fixo, que deve ser delimitado por aspas. Ex.: “valor”, ou uma expressão na DSL-SHDM que obtém em tempo de execução o texto a ser mostrado.

Parameters: responsável por armazenar os parâmetros a serem passados para o contexto ou estrutura de acesso. Este atributo apenas é utilizado quando o índice ou contexto de destino da âncora recebe parâmetros. Os valores destes parâmetros podem ser expressões na DSL-SHDM, que serão avaliadas em tempo de execução, ou um valor fixo que deverá ser delimitado por aspas. Ex.: “valor”.

4.3.2.2.1.Relacionamento Target

Este relacionamento, entre a classe EntryAS e a classe abstrata NavigationalAbstractContainerObject, armazena o contexto ou estrutura de acesso

de destino da classe EntryAS. Quando este relacionamento está preenchido significa que a informação mostrada nesta EntryAS é uma âncora para o contexto ou índice existente neste relacionamento.

4.3.2.3.Classe OrderingAS

A classe OrderingAS armazena os campos pelos quais as informações existentes no índice serão ordenadas e sua ordem de ordenação dentro de um índice .

A classe OrderingAS tem os seguintes atributos:

OrderingField: campo do índice pelo qual será ordenado.

Direction: direção da ordenação. Dois valores possíveis Asc – Ascendente e Desc – Decrescente.

Priority: armazena a prioridade de ordenação junto com os outros campos.

4.3.2.4.Classe ParameterAS

A classe ParameterAS armazena as variáveis que receberão as informações passadas por parâmetro. Este campo é utilizado quando a estrutura de acesso precisa de um parâmetro de filtragem das informações a serem exibidas.

A classe ParameterAS tem apenas o atributo nome do parâmetro.

4.3.2.5.Relacionamentos LinksToAccessStructure e LinksToContext

Os relacionamentos LinksToAccessStructure e LinksToContext são relacionamentos meramente visuais, estes são necessários para que se possa expressar visualmente, através de linhas de relacionamento entre os objetos visuais, as relações entre EntryAS e uma outra AccessStructure ou Context.

4.3.2.6.Classe UserAS

Armazena o usuário que tem permissão de acesso à informação da estrutura de acesso e o tipo de permissão que o usuário possui.

A classe UserAS tem os seguintes atributos:

Name: nome do usuário ou perfil de acesso

Permission: tipo de permissão de acesso do usuário. Este tipo de permissão tem três valores possíveis: Read – permissão apenas para leitura; Write – Permissão apenas para escrita; Read_Write – permissão para escrita e leitura.

4.3.3.Classe Context

A classe Context é a classe responsável pela definição do contexto no qual uma classe navegacional será exibida para o usuário. A classe Context também pode ser utilizada na definição de índices baseados em contextos através da classe AccessStructureElement descrita no item 4.3.2.1.

4.3.3.1.Classe ContextElement

A classe ContextElement é uma classe que compõe a estrutura de um contexto. Nesta classe é definida a consulta que dará origem ao contexto. No diagrama, foram introduzidas duas novas formas de se especificar a consulta, ou seja, é possível gerar o conjunto de nós que formarão o contexto de três formas demonstradas abaixo.

- Contexto baseado em todas as instâncias de uma classe.

O contextElement "studentAlpha" abaixo faz parte do contexto "StudentAlpha" do diagrama de contextos navegacionais da Figura 23 e foi definido como sendo formado por todas as instâncias da classe Student.

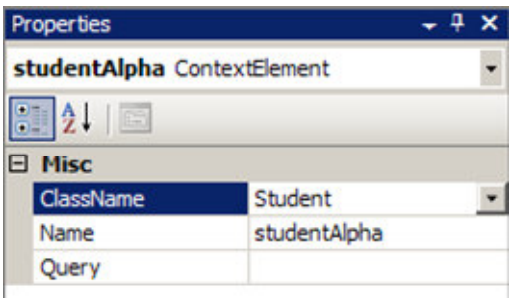

Figura 20 – Exemplo de ContextElement baseado em todas as instâncias de uma classe

- Contexto que seleciona as instâncias de uma classe com o auxilio de uma expressão de seleção escrita na DSL-SHDM

Como exemplo na Figura 21 foi utilizado o contexto "StudentByProfessor" da Figura 23. O campo "ClassName" do ContextElement "studentByProfessor" foi preenchido com a classe *Student* e o campo "Query" foi preenchido com uma expressão na DSL-SHDM como demonstrado na Figura 22. No valor da expressão DSL-SHDM a expressão "%SELECT THIS ELEMENT%" é um comando na DSL-SHDM que, em tempo de execução, seleciona apenas as instâncias da classe definida que atendem a expressão de filtro. No exemplo abaixo apenas os alunos que sejam orientados por um determinado professor passado por parâmetro serão selecionados.

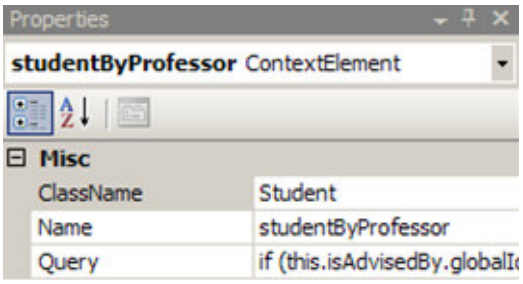

Figura 21 - Exemplo de ContextElement com filtro nas instâncias de uma classe

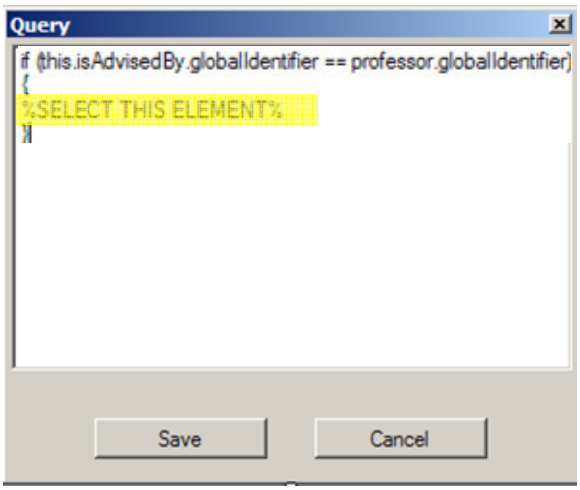

Figura 22 – Campo "Query" preenchido com uma expressão de filtro na DSL-SHDM

- Contexto que seleciona as instâncias de uma classe baseado em uma consulta SPARQL.

  Uma consulta SPARQL é a forma mais poderosa para criar um contexto navegacional, pois ao fazer uma consulta na base de dados RDF além dos dados podem ser consultados os metadados. Porém o resultado das consultas tem que ser o identificador único do objeto. No exemplo do Quadro 2 a

consulta SPARQL retorna todos os identificadores das instâncias da classe Student.

```
SELECT ?id  \n" +
"Where { ?id <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<OOHDMClassLibrary#Student>      . }";
```

Quadro 2 – Consulta Sparql que retorna todas as instâncias da classe Student

4.3.3.2.Classe OrderingC

A classe OrderingC armazena os campos pelos quais as informações existentes no contexto serão ordenadas. Baseado neste campo que são definidos quais os nós que serão apresentados através das âncoras: “próximo” e “anterior”.

A classe OrderingC tem os seguintes atributos:

OrderingField: campo da classe em contexto pelo qual o contexto será ordenado.

Direction: direção da ordenação. Dois valores possíveis Asc – Ascendente e Desc – Decrescente.

Priority: armazena a prioridade de ordenação em relação às outras instâncias da classe OrderingC dentro do contexto.

4.3.3.3.Classe ParameterC

A classe ParameterC armazena as variáveis que receberão as informações passadas por parâmetro. Este campo é utilizado quando o contexto precisa de um parâmetro de filtragem das informações a serem exibidas.

A classe ParameterC tem apenas os seguintes atributos:

Name: nome do atributo a ser passado por parâmetro. Este nome será utilizado na query do contexto. No exemplo da Figura 21, a variável “professor” foi definida como parâmetro, como mostra o contexto StudentByProfessor na Figura 23.

ParameterType: tipo do atributo esperado como parâmetro.

4.3.3.4.Classe UserC

Armazena o usuário que tem permissão de acesso à informação do contexto e o tipo de permissão que o usuário possui.

A classe UserC tem os seguintes atributos:

Name: nome do usuário ou perfil de acesso

Permission: tipo de permissão de acesso do usuário. Este tipo de permissão tem três valores possíveis: Read – permissão apenas para leitura; Write – Permissão apenas para escrita; Read_Write – permissão para escrita e leitura.

4.3.3.5.Relacionamento LinkToContexts

O relacionamento LinkToContext é um relacionamento meramente visual, este é necessário para que se possa expressar visualmente, através de linhas de relacionamento entre os objetos visuais, as relações existentes entre as classes navegacionais e os contextos. Por exemplo, se a classe navegacional em contexto tiver um atributo do tipo ContextAnchor, conforme definido no item 4.2.2.4.2, este relacionamento será expresso visualmente através do relacionamento LinkToContexts.

4.3.4.Classe Scope

A classe Scope é uma classe que representa um escopo navegacional. A função do escopo é facilitar a interpretação do diagrama através da retirada de âncoras redundantes, ou seja, se todos os contextos inseridos dentro de um escopo tiverem uma âncora em comum para algum outro contexto, esta âncora será representada utilizando, como origem, o escopo no qual os contextos estão inseridos e tendo como destino o contexto para o qual as âncoras em comum apontam.

4.3.4.1.Relacionamento Contain

Relacionamento entre a classe Scope e a classe Context que indica se o contexto está contido dentro do escopo. Tomando por base os elementos existentes neste relacionamento, ou seja, os contextos que estão inseridos dentro do escopo,

4.3.4.2.Relacionamento LinkTo

O relacionamento LinkTo é um relacionamento meramente visual, utilizado para representar âncoras que todos os contextos inseridos dentro do escopo têm em comum.

4.3.4.3.Relacionamento Parent

Relacionamento utilizado para identificar escopos aninhados, ou seja, um escopo dentro de outro. Este relacionamento aponta para escopo pai de um escopo.

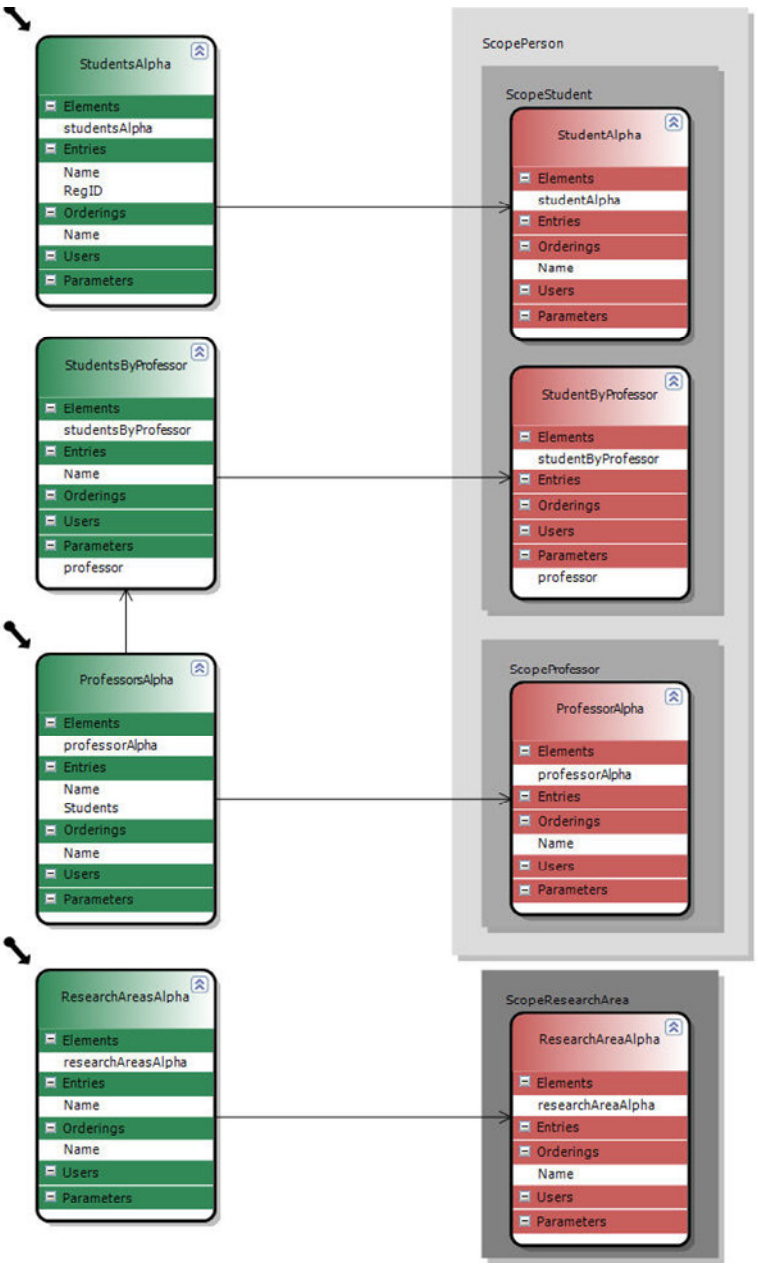

Figura 23 – Exemplo do diagrama de contextos navegacionais

4.4.Diagrama de Interface Abstrata

O Diagrama de Interface Abstrata (DIA) tem por objetivo a definição da interface em um nível abstrato, desta forma o elemento tecnológico que representará a informação é abstraído, sendo utilizando na representação do diagrama apenas o seu grupo funcional, obtendo independência da tecnologia utilizada na construção da interface da aplicação.

As classes de interface são definidas como agregações de objetos primitivos de interface (como exibidores ou ativadores). Os objetos navegacionais são mapeados em objetos de interface abstratos. Este mapeamento permite definir quais objetos ou informações estarão disponíveis para serem apresentados para o usuário, também define quais objetos farão o papel de ativadores da navegação, porém sem se importar com o elemento concreto responsável pela representação deste ativador.

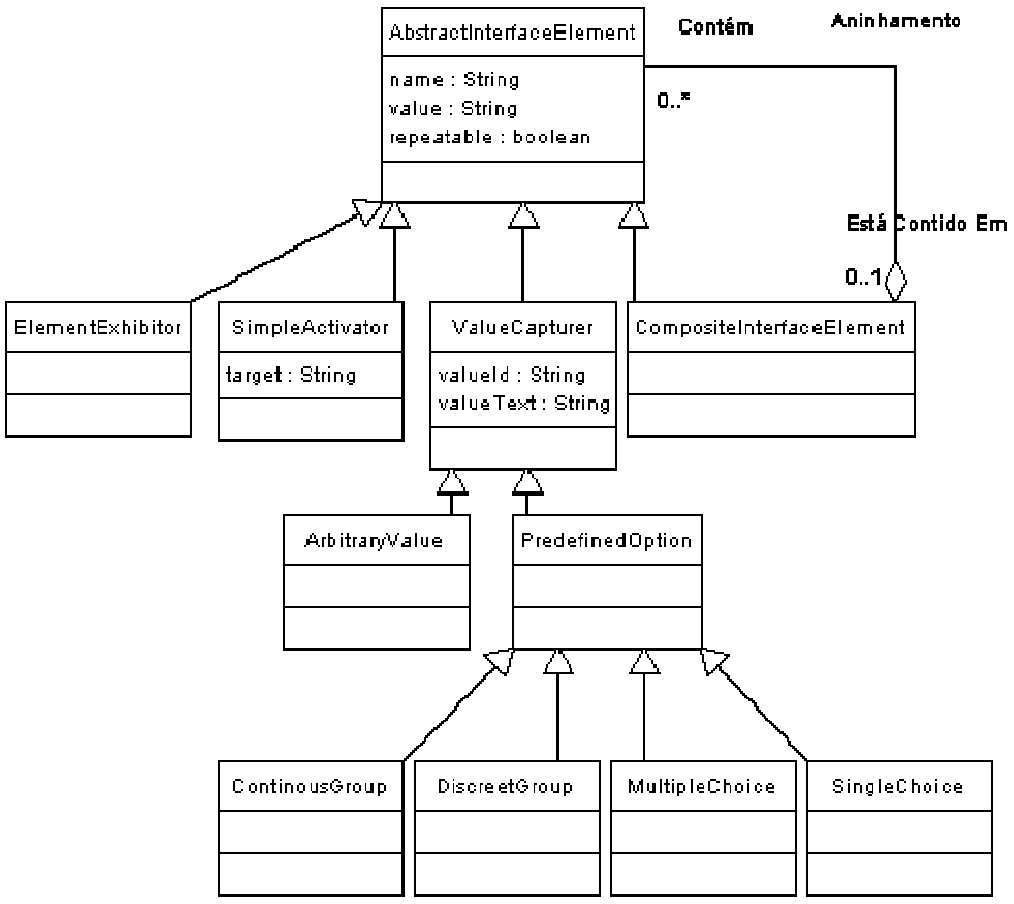

Figura 24 – Metamodelo do Diagrama de Interface Abstrata

O metamodelo SHDM do Diagrama de Interface Abstrata é apresentado na Figura 24 e o modelo do diagrama produzido no DSL-Tools é apresentado na Figura 25. O modelo do diagrama de interface abstrata é semelhante ao metamodelo SHDM do diagrama de interface abstrata. Foram adicionados apenas alguns atributos tecnológicos.

Em outros trabalhos como o Teresa XML [Mori et al., 2003], UIML [Abrams et al., 1999], USIXML [Limbourg et al., 2004] e XIML [Puerta & Eisenstein, 2001], o nível de abstração da descrição é baixo, ou seja, a forma de descrição é muito próxima da interface concreta. O modelo de interface abstrata utilizado nesta dissertação é fundamentado por uma abordagem muito mais abstrata que as ferramentas citadas acima. A descrição da interface abstrata do SHDM .Net segue o padrão proposto por [Moura, 2004]. Baseado neste padrão foi definido o Diagrama de Interface Abstrata do SHDM .Net. Desta forma o DIA é completamente independente de tecnologia, sendo a parte dependente de tecnologia obtida através do diagrama de interface concreto e da utilização de uma ontologia de interface concreta apresentados respectivamente nos itens 4.5 e 4.6

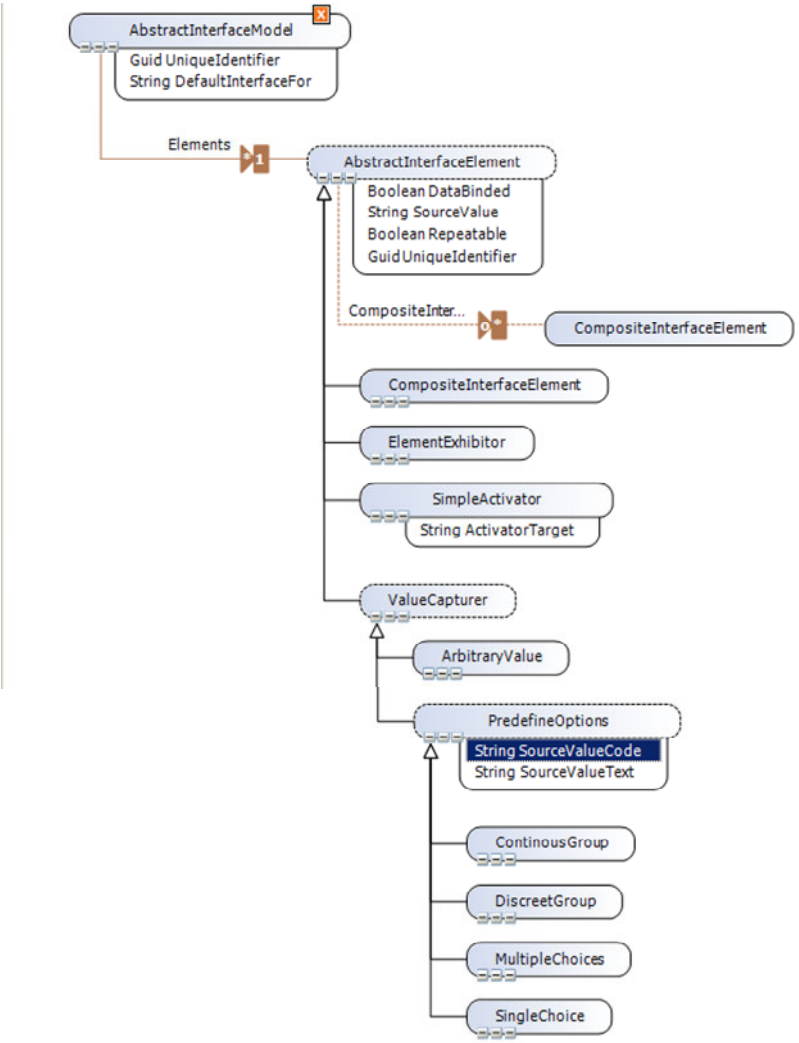

Figura 25 – Modelo do diagrama de interface abstrata no DSL-Tools

4.4.1.Classe AbstractInterfaceModel

A classe AbstractInterfaceModel é responsável por armazenar todas as instâncias da classe AbstractInterfaceElement que compõem uma instância de um Diagrama de Interface Abstrata.

Esta classe tem os seguintes atributos:

Name – identificação do nome da interface abstrata

DefaultInterfaceFor – armazena a lista de estruturas de acesso ou contextos para os quais esta interface é a interface padrão. O nome dos contextos e/ou estruturas de acesso devem ser separados por ponto e vírgula ‘; ’.

UniqueIdentifier – atributo tecnológico, preenchido automaticamente pelo sistema, que auxilia no mapeamento dos elementos abstratos nos concretos.

4.4.1.1.Relacionamento Elements

O relacionamento Elements é um relacionamento de composição entre a classe AbstractInterfaceModel e a classe AbstractInterfaceElement. A semântica deste relacionamento expressa que um AbstractInterfaceModel é composto de AbstractInterfaceElement.

Este relacionamento é representado na Classe AbstractInterfaceModel através do atributo Entities. Este atributo, que tem o mesmo nome do relacionamento, armazena a coleção de classes AbstractInterfaceModel que compõe uma instância de um Diagrama de Interface Abstrata.

4.4.2.Classe AbstractInterfaceElement

Classe abstrata que representa um objeto visual genérico. A partir desta classe todas as outras classes da interface abstrata são derivadas.

Esta classe tem por atributos:

DataBound: Atributo que guarda se o elemento abstrato está associado com uma classe navegacional, ou seja, se seus dados tem como origem uma instância de classe. Possíveis valores: true ou false.

SourceValue: Atributo que guarda o valor do elemento, caso o elemento de interface seja Databound = true, o valor deverá ser uma expressão na DSL-SHDM que seja avaliado em tempo de execução. Caso o elemento seja DataBound = false, o atributo SourceValue deve ser preenchido com o valor a ser apresentado.

Repeatable: Atributo cuja semântica indica que se o valor do atributo tem repetição ou não, ou seja, se é um conjunto de instâncias de objetos a serem percorridas.

UniqueIdentifier – atributo tecnológico, preenchido automaticamente pelo sistema, que auxilia no mapeamento dos elementos abstratos nos concretos.

4.4.2.1.Relacionamento CompositeInterfaceElement

Este relacionamento equivale ao relacionamento “Aninhamento” do metamodelo de interface abstrata, Figura 24, indica se a classe AbstractInterfaceElement está aninhada dentro de algum CompositeInterfaceElement. Este relacionamento produz dois atributos, um vetor de AbstractInterfaceElements na classe CompositeInterfaceElement, onde são armazenados todos os elementos aninhados dentro de uma instância de CompositeInterfaceElement, e um atributo CompositeInterfaceElement na classe AbstractInterfaceElement que representa o elemento de interface no qual a instância de AbstractInterfaceElement está inserida.

4.4.3.Classe CompositeInterfaceElement

A classe CompositeInterfaceElement, que herda de AbstractInterfaceElement, é responsável por armazenar todas as instâncias das subclasses de AbstractInterfaceElement que estão aninhadas dentro de uma instância da classe CompositeInterfaceElement.

Esta classe tem apenas um atributo, além dos herdados de AbstractInterfaceElement, proveniente do relacionamento CompositeInterfaceElement, descrito no item 4.4.2.1. Este atributo comporta um conjunto de instâncias da classe AbstractInterfaceElements que estão aninhadas dentro da instância do objeto CompositeInterfaceElement.

4.4.4.Classe ElementExhibitor

A classe ElementExhibitor é responsável por representar elementos que exibem algum tipo de conteúdo, como, por exemplo, um texto ou uma imagem.

Esta classe tem como superclasse a classe AbstractInterfaceElement e não tem nenhum atributo adicional.

4.4.5.Classe SimpleActivator

A classe SimpleActivator é responsável por representar qualquer elemento capaz de reagir a eventos externos – caso típico do clicar o mouse – que possua um evento associado a ele, tais como ativar um link, um botão de ação, o envio de

um e-mail, dentre outros. Esta classe tem como superclasse a classe AbstractInterfaceElement e tem o atributo ActivatorTarget que armazena a âncora ou o destino da ativação.

4.4.6.Classe ValueCapturer

Classe abstrata ValueCapturer é responsável por representar os elementos capazes de capturar um valor, ou seja, todos os campos de entrada de dado, por exemplo, caixas de texto, combo boxes e selecionadores como check boxes. Esta classe tem como superclasse a classe AbstractInterfaceElement e não tem nenhum atributo adicional.

4.4.7.Classe ArbitraryValue

Classe responsável por representar elementos que permitem que o usuário insira dados utilizando o teclado. Pode representar um campo de formulário, ou seja, uma caixa de texto. Esta classe tem como superclasse a classe ArbitraryValue e não tem nenhum atributo adicional.

4.4.8.Classe PredefinedOptions

Classe abstrata responsável por representar a seleção de um subconjunto a partir de um conjunto de valores pré-definidos; muitas vezes esse subconjunto poderá ser unitário.

Esta classe herda da classe ValueCapturer e tem os seguintes atributos adicionais:

SourceValueId = este atributo armazena o conjunto de identificadores dos valores texto que são atribuídos ao parâmetro SourceValueText. Esse parâmetro pode receber uma exressao SHDM .NET ou um string com os identificadores sepadados por ponto e virgula (;);

SourceValueText = este atributo armazena o conjunto de valores que serão apresentados ao usuário para que este faça a seleção. Este parâmetro pode receber uma expressão SHDM .NET ou um string com os textos separados por ponto e virgula (;).

4.4.9.Classe ContinousGroup

Classe responsável pela representação de um elemento concreto que permite a seleção de um valor de um conjunto infinito de valores, como por exemplo, uma barra de volume  . Esta classe tem como superclasse a classe PredefinedOptions e não tem nenhum atributo adicional.

4.4.10.Classe DiscreteGroup

Classe responsável pela representação de um elemento concreto que permite a seleção de um valor de um conjunto enumerável de valores, como por exemplo, um spin button  . Esta classe tem como superclasse a classe PredefinedOptions e não tem nenhum atributo adicional.

4.4.11.Classe MultipleChoices

Classe responsável pela representação de um elemento concreto que permite a seleção de mais de um elemento de um conjunto enumerável de valores, como por exemplo, uma listbox. Esta classe tem como superclasse a classe PredefinedOptions e não tem nenhum atributo adicional.

4.4.12.Classe SingleChoice

Classe responsável pela representação de um elemento concreto que permite a seleção de apenas um elemento de um conjunto enumerável de valores, como por exemplo, uma Combo box. Esta classe tem como superclasse a classe PredefinedOptions e não tem nenhum atributo adicional.

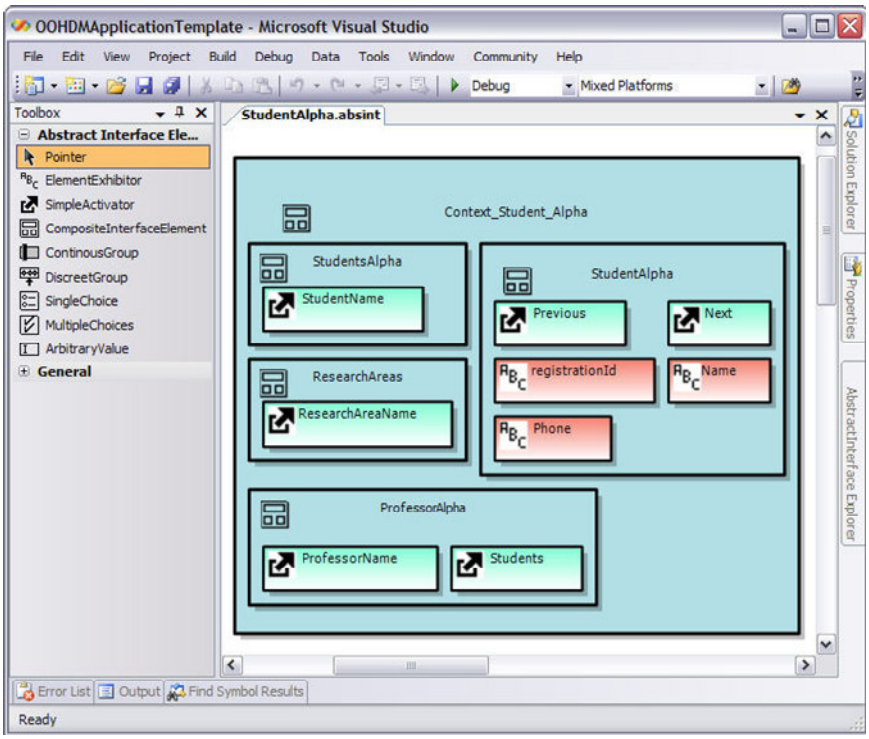

Figura 26 – Diagrama de Interface Abstrata

4.5. Diagrama de Interface Concreta

O Diagrama de Interface Concreta (DIC) é um diagrama produzido automaticamente pelo SHDM .Net. Quando o desenvolvedor finaliza o DIA um DIC é produzido. O DIC foi automaticamente produzido pelo *framework* sem os elementos concretos responsáveis por apresentar cada elemento abstrato em uma determinada tecnologia de interface. Este mapeamento deve ser efetuado pelo desenvolvedor da aplicação baseado em uma ontologia de *widgets* concretos a ser descrita no item 4.6.

Quando o usuário seleciona a ontologia sobre a qual o mapeamento será feito, automaticamente a ontologia é carregada e ao selecionar um elemento abstrato os possíveis *widgets* concretos para aquele elemento são lidos da ontologia e apresentados. Uma vez que o desenvolvedor tenha efetuado todos os mapeamentos, a interface concreta é produzida. A interface concreta produzida deve ser formatada com o auxílio de arquivo CSS.

O Cascading Style Sheet permite uma versatilidade maior na programação do layout de páginas web. Basicamente, o CSS permite ao designer um controle maior sobre os atributos de uma homepage, como tamanho e cor das fontes, espaçamento entre linhas e caracteres, margem do texto, caixas de texto, botões de

formulário, entre outros. Introduziu também às páginas a utilização de layers, permitindo a sobreposição de objetos como textos e imagens em camadas uma sobre as outras. Ao invés de colocar a formatação dentro do código, o programador cria um link para uma página que contém os estilos, procedendo de forma idêntica para todas as páginas de um portal. Quando quiser alterar a aparência do portal basta, portanto modificar somente os arquivos de estilo. Mantendo os arquivos fonte das páginas inalterados. Através do uso de CSSs uma página produzida utilizando o SHDM .Net tem a mesma flexibilidade de interface do site Css Zen Garden[1], conforme demonstrado no Apêndice 1 – Exemplo da interface do site Css Zen Garden.

Todos os atributos do DIC que são "importados" do DIA tornam-se atributos de leitura apenas, podendo ser editados apenas os atributos pertinentes a este diagrama.

O modelo do DIC, apresentado na Figura 27, tem as mesmas classes que o DIA, porém foram adicionados atributos dando suporte ao mapeamento dos elementos concretos.

Abaixo serão descritas as classes do DIC cuja estrutura difere da estrutura do DIA.

4.5.1.ConcreteInterfaceModel

Classe responsável por armazenar todas as classes AbstractInterfaceElement do diagrama de interface concreta.

Esta classe é análoga a classe AbstractInterfaceModel com o acréscimo dos atributos listados abaixo:

InterfaceTargetProject: Atributo responsável por armazenar o projeto de destino da interface concreta produzida, ou seja, o projeto no qual a interface concreta, produzida pelo SHDM .Net, será adicionada. Neste atributo será apresentado uma lista dos projetos do Visual Studio .Net nos quais o arquivo da interface concreta produzido será colocado.

[1] http://www.csszengarden.com

ConcreteWidgetsOntology: Atributo responsável pelo armazenamento do nome da instância da ontologia de *widgets* concretos utilizada para produzir a interface concreta.

CssFile: Caminho do arquivo CSS utilizado para formatar a interface produzida.

4.5.2.AbstractInterfaceElement

Classe abstrata a partir da qual todos os elementos abstratos de interface herdam. Os atributos adicionados nesta classe serão preenchidos pelas instâncias dos elementos concretos que herdam esta classe.

Esta classe é análoga a classe AbstractInterfaceElement, existente no Diagrama de Interface Concreta, com o acréscimo dos atributos listados abaixo:

ConcreteWidget: Atributo responsável por armazenar o nome do elemento concreto, carregado da ontologia de *widgets* concretos, no qual o elemento abstrato foi mapeado.

CssClass: nome da classe Css responsável pela formatação do deste elemento na interface produzida.

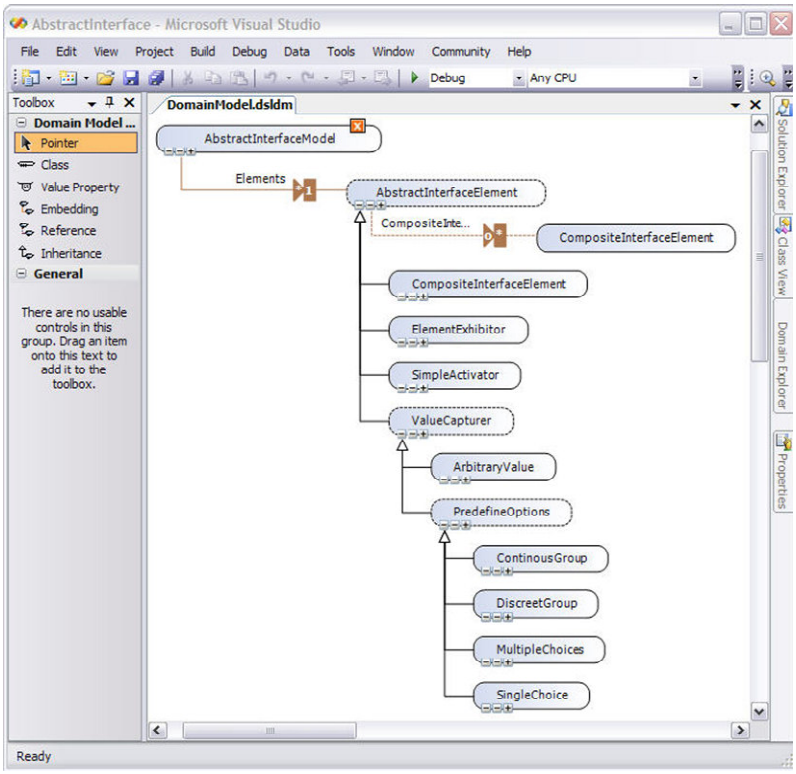

Figura 27 – Modelo do Diagrama de Interface Concreta.

4.6.Ontologia de *widgets* concretos

A ontologia de *widgets* concretos tem por objetivo definir os elementos visuais concretos que podem ser usados para implementar um *widget* abstrato em uma plataforma ou tecnologia específica.

A ontologia é composta de duas classes: ConcreteInterface e ConcreteWidget que são descritas abaixo:

4.6.1.ConcreteInterface

Classe que armazena a definição da interface concreta é composta dos seguintes slots:

name: nome identificador da interface concreta ou tecnologia.

concrete*Widgets*: conjunto de *widgets* concretos que compõem a interface.

dataSourceFunction: função utilizada para obter os dados de uma coleção na tecnologia a ser utilizada na implementação.

evaluateFunction: função a ser utilizada para retirar um valor de uma coleção na qual tem uma dataSourceFunction.

fileExtension: extensão do arquivo de interface concreta a ser produzido.

4.6.2.ConcreteWidget

Classe que armazena a implementação do *widget* em uma determinada tecnologia. Possui os seguintes atributos:

abstractElements: armazena o conjunto de *widgets* abstratos que podem ser representados pelo ConcreteWidget. Estas classes de *widgets* abstratos estão definidas na ontologia de *widgets* abstratos.

code: código fonte do *widget* concreto.

name: nome do *widget* concreto.

A Figura 28 mostra a ontologia de *widgets* abstratos juntamente com a ontologia de *widgets* concretos. Os elementos que fazem parte da ontologia de *widgets* abstratos têm o prefixo “awo” e os elementos que fazem parte da ontologia de *widgets* concretos têm o prefixo “cwo”.

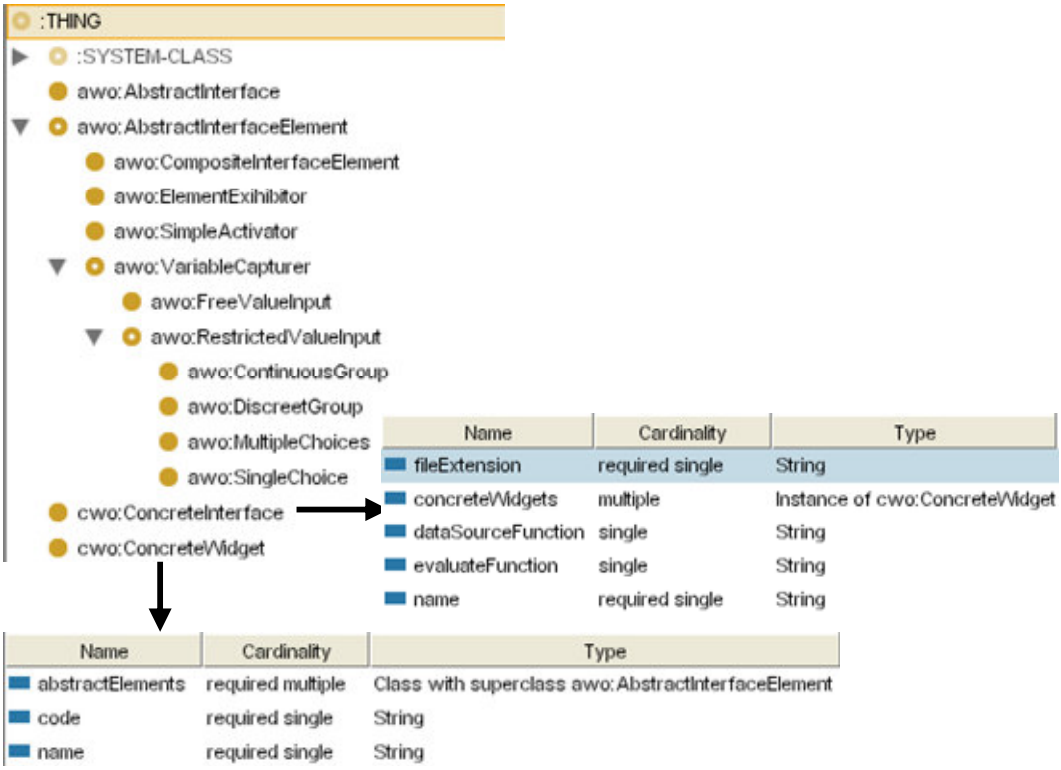

Figura 28 – Ontologia de *widgets* concretos

4.7.Geração da DSL-SHDM

A geração da DSL-SHDM é efetuada baseada em um *template* de código que interpreta das instâncias dos modelos desenvolvidos pelo usuário, produzindo o código fonte da DSL-SHDM em C#. Quando este *template* é executado recebe como parâmetro o programa na DSL-Microsoft produzida quando o usuário cria um diagrama. A estrutura da DSL-Microsoft foi produzida automaticamente pelo DSL-Tools quando o *framework* foi desenvolvido. O *template* de forma similar a uma página Aspx onde são definidos blocos de controle. Dentro destes blocos a lógica de geração do código é colocada. Estes blocos podem ser identificados no Quadro 3, pois são delimitados pelos caracteres <# e #>. O Quadro 3 apresenta uma parte do código de geração das classes navegacionais. Neste código são criados os diversos tipos de atributos simples definidos pelo usuário em uma classe navegacional. Este *template* poderá ser alterado em versões futuras deste *framework* dando suporte a geração do código da DSL-SHDM em outras tecnologias ou linguagens de programação.

```
<#@ import namespace="System.Collections" #>
<#@ template
inherits="Microsoft.VisualStudio.TextTemplating.VSHost.ModelingTe
xtTransformation"#>
<#@ output extension=".cs" #>
<#@ navigationalmetamodel
processor="NavigationalDirectiveProcessor"
requires="fileName='%NAVIGATIONALMODELFILENAME%'"
provides="NavigationalMetaModel=NavigationalMetaModel" #>


<#     foreach ( NavigationalClass navclass in
       this.NavigationalMetaModel.Entities )
         {
             if (navclass.Inherits != null )
             {
                   hasSuperClass = true;
                   superClassName = navclass.Inherits.Name;
             }
             else
             {
                   hasSuperClass = false;
                   superClassName = "";
                   }
#>
        #region <#=navclass.Name#>


        public partial class <#=navclass.Name#>:
<#WriteLine(IheritanceFormat(hasSuperClass, superClassName)); #>
        {
            #region Attributes
            private int _globalIdentifier;
            public int Identifier()
            {
                  return this.globalIdentifier;
            }
            public void GenerateNewIdentifier()
            {
                  Random randomicNumber = new Random();
                  this._globalIdentifier = randomicNumber.Next();
            }
            public int globalIdentifier
            {
                  get { return this._globalIdentifier;}
            }
            public <#=navclass.Name#>(int globalIdentifier)
            {
                  this._globalIdentifier = globalIdentifier;
```

```
            }
            public <#=navclass.Name#>()
            {
                  Random randomNumber = new Random();
                  this._globalIdentifier = randomNumber.Next();
            }
            public static List<<#=navclass.Name#>> getList()
            {
               return new List<<#=navclass.Name#>>();
            }
            <# foreach (
      OOHDM.Diagrams.Navigational.DomainModel.Attribute attrib in
navclass.Attributes )
            {
               if (attrib is
      OOHDM.Diagrams.Navigational.DomainModel.Simple)
               {
                  simpleAttrib =
      (OOHDM.Diagrams.Navigational.DomainModel.Simple) attrib;
                  convertDataType(simpleAttrib.type,out
      tipoClasse, out inicializacao, out property);
                        #>
      // Atributo <#=attrib.Name #>
      <#if (property) {#>
                  private <#=tipoClasse #> a_<#=attrib.Name #> =
<#=inicializacao#>;
                  public <#=tipoClasse #> <#=attrib.Name #>
                  {
                        get{return this.a_<#=attrib.Name #>;}
                        set{this.a_<#=attrib.Name #> = value;}
                  }
      <#}else{#>
                  public <#=tipoClasse #> <#=attrib.Name #> =
<#=inicializacao#>;
      <#}#> // Fim Atributo <#=attrib.Name #>
```

Quadro 3 – Exemplo do *template* de geração da DSL-SHDM de uma classe navegacional

4.8.Arquitetura de Runtime

O fluxo de uma requisição tem início quando um usuário solicita alguma informação. Esta solicitação é interpretada por um controlador, que é responsável por capturar todas as solicitações do *framework*. O controlador identifica a requisição e transfere o controle para a estrutura de acesso ou contexto responsável por processar a requisição. O contexto ou estrutura de acesso recebe

os parâmetros passados pela requisição e começa a coletar, no banco de dados semântico, toda a informação usada para criar a resposta da requisição. O critério de seleção é definido em cada contexto ou estrutura de acesso. As instâncias de classes que compõem a resposta são armazenadas na sessão do usuário. O controlador seleciona a interface responsável por mostrar a resposta da requisição. Esta interface responsável pode ser passada como um parâmetro da requisição ou, se nenhuma interface for passada, uma interface padrão é selecionada pelo controlador para o contexto ou estrutura de acesso, como especificado no modelo SHDM. Os elementos da interface são associados aos elementos que compõem o pacote de resposta à requisição, que foram previamente colocados na sessão do usuário. Por fim a interface é apresentada.

A Figura 29 ilustra um fluxo de requisição, de uma aplicação produzida pelo *framework*, quando um usuário executa uma ação.

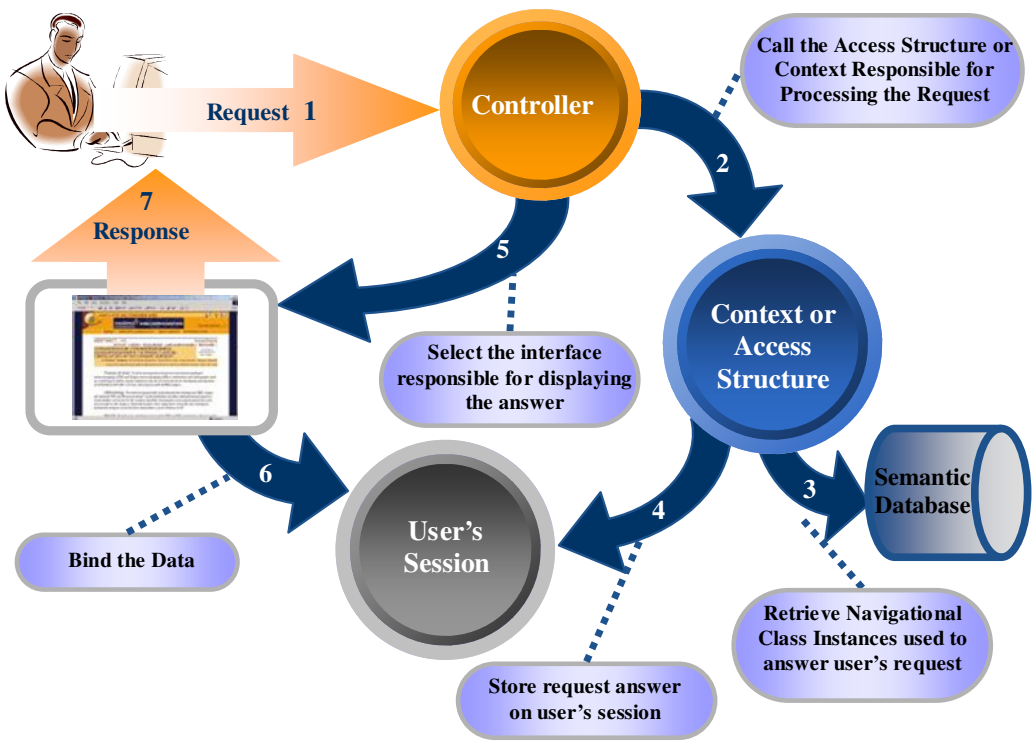

Figura 29 – Fluxo de requisição de uma aplicação produzida pelo SHDM .Net

4.9.Restrições de Implementação

Algumas restrições de implementação foram encontradas no decorrer do desenvolvimento desta ferramenta. Podemos classificar estas restrições em duas categorias descritas abaixo:

4.9.1.Restrições no design dos diagramas

Foram encontradas as seguintes restrições de design.

- Número pequeno de possibilidades de representação visual de um elemento, e a customização de cada elemento apresentado é extremamente limitada.
- Impossibilidade de criação de um relacionamento entre links, impossibilitando uma representação visual do conceito de sub-relacionamento conforme proposto [Szundy, 2004]
- Impossibilidade de representar nativamente elementos visuais aninhados. Limitação esta contornada através da manipulação dos objetos visuais de forma programática.

4.9.2.Restrições tecnológicas

Foi encontrada a seguinte restrição tecnológica

- Ausência de meta-relacionamento entre modelos, ou seja, impossibilidade de uma validação nativa de atributos utilizados entre instâncias de modelo. Toda a validação foi efetuada pela alteração da API do DSL-Tools bem como pela inclusão de códigos responsáveis pela verificação em tempo real das inconsistências entre os diagramas.