# 4
# Ambiente HyperDE

O HyperDE é a combinação de um framework MVC e um ambiente de desenvolvimento rápido que permite a arquitetos de informação construir em curto espaço de tempo um protótipo de uma aplicação modelada segundo os métodos OOHDM ou SHDM.

Como framework MVC, o HyperDE oferece uma pilha completa de camadas necessária a implementação de uma aplicação OOHDM/SHDM, que contemplam as funções de Modelo Navegacional, Controle e Visão de uma aplicação Web, modelada utilizando as primitivas descritas na seção anterior.

Como ambiente de desenvolvimento rápido, o HyperDE, através de sua interface Web e ferramentas de linha de comando, possibilita a construção, execução, testes e manutenção da aplicação de forma iterativa e incremental.

Atualmente, nos restringimos a caracterizar o HyperDE como útil apenas à **prototipação** de aplicações OOHDM/SHDM, basicamente por questões de performance e escalabilidade. A dinamicidade da camada de modelo navegacional na forma atualmente implementada exige constante introspecção em cima da base de dados sobre a qual o ambiente é construído. Isto impõe uma sobrecarga que se inviabiliza o uso das aplicações desenvolvidas com o HyperDE em ambientes de produção que requerem maior desempenho. Cabe observar que ainda não foram estabelecidos os parâmetros de desempenho das aplicações desenvolvidas com o HyperDE.

Por outro lado, essa arquitetura dá ao ambiente uma qualidade dinâmica e flexível o suficiente para que ele cumpra seu objetivo enquanto ferramenta de desenvolvimento ágil.

Futuramente, a falta de performance e escalabilidade poderão ser sanadas, possivelmente através do uso de, entre outras coisas, compiladores que gerariam um modelo navegacional baseado em uma base de dados estática, capaz de ser utilizado em ambientes de produção. Este assunto será abordado em mais detalhes na seção sobre trabalhos futuros.

Além disso, nesta primeira versão do framework, não foram embutidos mecanismos de segurança e controles de permissão, requisitos bastante comuns a aplicações web e que deverão ser suportados em versões futuras.

Uma visão geral do ambiente e do framework é apresentada no diagrama a seguir:

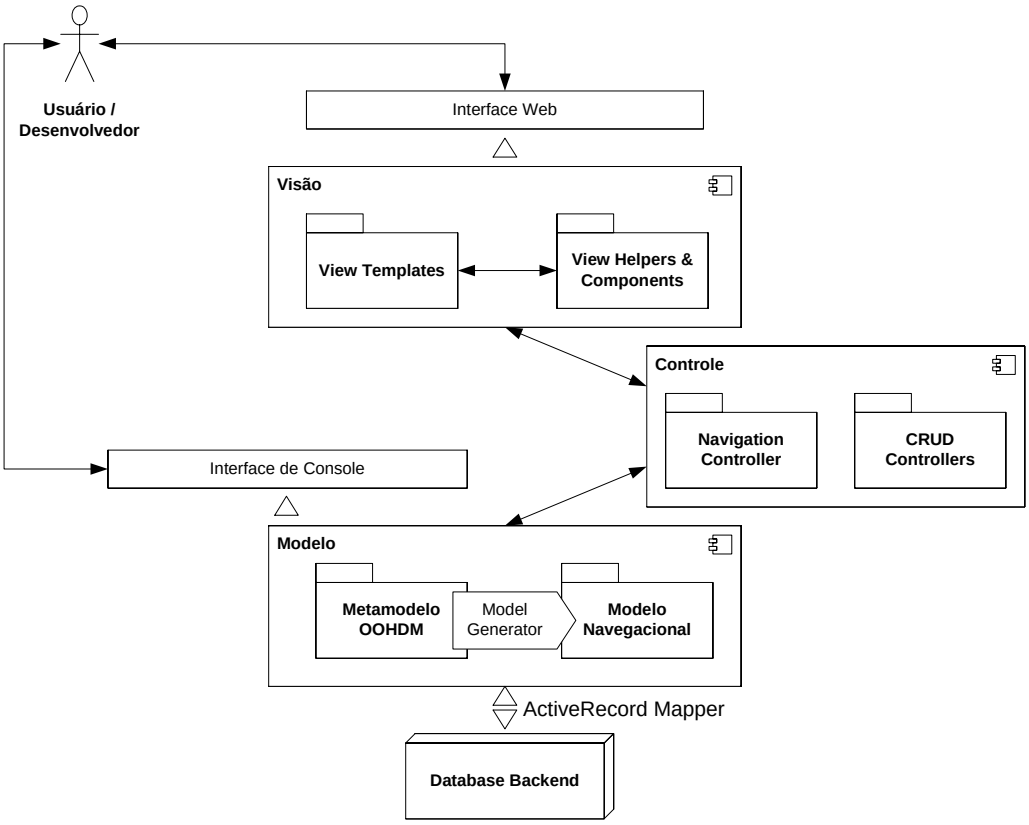


Figura 9 – Componentes da Arquitetura do HyperDE

Cada camada da arquitetura será explicada em maior detalhe na seção a seguir.

## 4.1. Framework MVC

O framework MVC do HyperDE é baseado no padrão de projeto modelo-visão-controle (MVC), que promove uma separação estrita entre a camada responsável pela representação do modelo de dados da aplicação e a camada responsável pela visualização desse modelo pelo usuário. Entre as duas camadas (modelo e visão), é construída uma camada de controle cujo objetivo primário é o de coordenar os eventos disparados pela interação do usuário com a interface da aplicação, atualizando o modelo de dados de acordo, assim como a visualização

da interface do usuário. Vale ressaltar que a camada de modelo do ambiente possui uma característica especial pois corresponde ao modelo “navegacional”, construído sobre as primitivas definidas nos métodos OOHDM e SHDM. Veremos agora uma descrição mais detalhada sobre cada uma das camadas do framework.

### 4.1.1. Camada de Modelo Navegacional

A camada de modelo navegacional do framework fornece ao arquiteto de informação as primitivas do metamodelo definidas para o OOHDM/SHDM. Portanto, nessa camada podemos definir e manipular:

- Contextos navegacionais
- Estruturas de acesso (Índices) e seus atributos
- Classes navegacionais, seus atributos e operações
- Elos
- Landmarks
- Nós, seus atributos e relacionamentos

As entidades listadas acima fazem parte do metamodelo OOHDM, cuja manipulação é feita através de classes de objetos definidas primitivamente pelo framework.

Por sua vez, a instanciação dessas classes permite construir o modelo navegacional da aplicação propriamente dita. O framework então cria dinamicamente as classes de objetos que representam o modelo navegacional da aplicação e permite a manipulação dessas classes da mesma forma que as classes do metamodelo.

Vejamos portanto um exemplo: se instanciarmos um objeto da metaclasse NavClass (que faz parte do metamodelo OOHDM) e atribuirmos à propriedade “name” dessa instância o valor “Pessoa”, o framework automaticamente irá gerar uma nova classe (essa integrante do modelo da aplicação) chamada “Pessoa”, que por sua vez poderá ser instanciada e manipulada como qualquer outra classe do framework.

Essa classe “Pessoa” possuirá ainda atributos, operações e relacionamentos nela definidos por outras metaclasses, NavAttribute, Link e NavOperation. Note que tudo isso ocorre dinamicamente em tempo de execução da aplicação. Não haverá em nenhum arquivo qualquer código-fonte referente à classe “Pessoa”, pois ela existirá apenas na memória da aplicação e poderá ser redefinida e refinada a qualquer momento.

Todas as metaclasses OOHDM, assim como as classes dinamicamente geradas na aplicação, são derivadas de SemanticRecord::Base e portanto possuem todos os seus mecanismos de persistência.

O componente SemanticRecord implementa o padrão de projeto “Active Record” como visto em [Fowler, 2002] e que promove um mapeamento direto entre um objeto em memória e um registro em um banco de dados, conferindo à classe que define o objeto a responsabilidade por todas as operações de persistência do mesmo. Isso torna a manipulação do objeto bastante simples em termos das 4 operações básicas de persistência (também conhecidas como operações *CRUD: Create-Recover-Update-Delete*), criando uma camada de abstração bastante útil e poderosa entre o banco de dados e o modelo da aplicação.

A base de dados tem função importante dentro da camada de modelo, pois proporciona não apenas os serviços de persistência dos objetos do modelo, mas como também a resolução de consultas, podendo ainda ser estendida com regras de inferência para realização de consultas mais inteligentes.



### 4.1.2. Camada de Controle

A camada de controle do HyperDE tem função de coordenar as interações do usuário com o modelo navegacional e o metamodelo da aplicação através da interface Web do ambiente.

Esta camada é composta de diversos controladores, do qual destacamos o controlador de navegação, responsável pelas ações de navegação contextual e construção das estruturas de acesso (índices). Essas duas são as ações principais da aplicação construída no ambiente.

Os outros controladores são responsáveis pelas ações de manipulação do metamodelo do ambiente e do modelo da aplicação. Estes cuidam basicamente das operações de listagem, inserção, alteração, exclusão de todas as entidades que

compõem o ambiente, tais como classes navegacionais, contextos, índices, elos, etc. assim como os nós, que formam a base de instâncias da aplicação, e que são efetivamente visualizados pelo controlador de navegação.

O controlador de navegação desempenha outra função importante com relação à seleção das visões que são apresentadas ao usuário na interface Web, que pode variar de acordo com o contexto e a classe navegacional correntemente exibida, no caso de ação de navegação contextual, assim como de acordo com o índice, no caso de ação de índice. Veremos esse mecanismo de seleção em mais detalhe em seção posterior da dissertação.

### 4.1.3. Camada de Visão

O HyperDE provê visões genéricas para as ações de navegação contextual e navegação em índice que estão embutidas no framework. No entanto, o ambiente permite que o arquiteto de informação defina novas visões customizadas que podem ser associadas a contextos, classes navegacionais e índices.

O framework então, em tempo de execução, seleciona dinamicamente a visão apropriada, dependendo do índice, contexto e/ou classe navegacional sendo correntemente acessada. Se não foi definida nenhuma visão customizada para o contexto corrente, o framework utiliza as visões genéricas embutidas no ambiente.

As visões customizadas podem ser definidas e construídas pelo usuário através da própria interface Web do ambiente. O framework fornece uma série de funções auxiliares (chamadas de “helpers”) para ajudar no desenho de primitivas do sistema. Essa biblioteca de funções fornece uma camada de abstração que evita que o usuário tenha que realizar uma implementação em HTML puro, e reduz a necessidade de conhecimento da linguagem de programação sob a qual o ambiente foi construído. Essas funções incluem auxiliadores para o desenho de índices, atributos de nós, âncoras para navegação e para disparar operações, entre outros. Essa biblioteca será documentada em detalhe em seção posterior da dissertação.

Estão disponíveis no ambiente três tipos de visão customizada:

- Visões Genéricas: podem ser utilizadas para que o usuário crie componentes visuais reutilizáveis em outros templates (como folhas

de estilo, funções em JavaScript e outros) e também para a definição de layouts (layouts serão explicados em detalhe em seção posterior da dissertação).

- Visões de Contexto: são utilizadas para navegação contextual e podem ser associadas a contextos e/ou classes específicas, podendo ainda estar relacionado a um layout definido através de visões genéricas.
- Visões de Índice: são utilizadas para navegação em índice e podem ser associadas a índices específicos. Também podem se utilizar de layout definido através de visões genéricas.

Na versão atual, esse mecanismo de customização e seleção dinâmica de visões está disponível apenas para as ações relacionadas ao controlador de navegação, que corresponde à aplicação criada no ambiente. Para as ações relacionadas aos controladores CRUD, que consideramos ações de retaguarda, o ambiente fornece apenas templates padronizados. No entanto, a customização de visões poderá ser estendida a todas as ações do ambiente em versões futuras.

A camada de visões foi projetada inicialmente para a construções de saídas baseadas em HTML, mas isso não impede o uso do mesmo mecanismo para a construção de saídas em outras linguagens textuais para descrição de interfaces, como XML, XAML ou XUL.



## 4.2. Ambiente de Desenvolvimento e Prototipação

O HyperDE como ambiente de desenvolvimento e prototipação foi projetado tendo como objetivos principais prover ao usuário/desenvolvedor uma forma ágil e produtiva de construir uma aplicação baseada nos métodos OOHDM/SHDM, tirando proveito dos conceitos e primitivas definidas nestes métodos.

O ambiente, através de sua interface Web, minimiza a necessidade do usuário de conhecer linguagens de programação para conseguir implementar uma aplicação hipermídia. De fato, um usuário do ambiente é capaz de prototipar uma aplicação apenas conhecendo o método OOHDM e utilizando o ambiente visual Web do HyperDE.

A necessidade de conhecimentos de programação se dá apenas para a definição de atributos computados e operações em classes, onde é necessário escrever código utilizando a linguagem Ruby e a linguagem específica de domínio do framework. Adicionalmente, este conhecimento também é necessário para a construção de templates para visões customizadas, onde é possível utilizar HTML, XML e as funções da biblioteca de auxílio ao desenho de interfaces do framework. De qualquer forma, mesmo essas funções podem ser realizadas utilizando a própria interface Web do ambiente.

O ambiente de desenvolvimento é capaz de executar a aplicação OOHDM/SHDM modelada no mesmo de forma totalmente dinâmica. Assim, qualquer alteração do modelo navegacional da aplicação é refletida instantaneamente na execução da mesma.

Com isso, é possível reduzir substancialmente o tamanho do ciclo de modelagem-implementação, propiciando iterações de desenvolvimento curtas o suficiente para possibilitar a utilização de processos ágeis de desenvolvimento [Cockburn, 2001]. Atualmente, processos ágeis encontram-se cada vez mais difundidos, principalmente no âmbito de aplicações altamente interativas, como ocorre na Web, por este tipo de aplicação em especial requisitar constante aprovação por parte de clientes e usuários finais. Prática em que os processos ágeis se apóiam fortemente.

Em nossa experiência, uma aplicação OOHDM/SHDM com um nível razoável de complexidade em seu modelo navegacional (que veremos em detalhe em seção posterior), pôde ser desenvolvida dentro do ambiente em questão de poucos dias. Exemplos mais simples puderam ser desenvolvidos em questão de horas.

### 4.2.1. Interfaces e Ferramentas do Ambiente

A principal interface com usuário ao HyperDE é a sua interface Web, que permite ao usuário executar todas as ações necessárias para construção de um modelo navegacional, contextos, índices, landmarks, elos e nós, assim como a própria execução da aplicação desenvolvida.

Além da interface Web, o usuário/desenvolvedor pode interagir e testar o modelo navegacional através de uma interface de console interativa. Para tanto,

ele deve ter um conhecimento básico da linguagem de programação Ruby e a linguagem específica de domínio que o ambiente fornece.

O usuário/desenvolvedor da aplicação pode contar ainda com uma ferramenta de linha de comando para importação e exportação do modelo navegacional da aplicação em um formato textual bastante conciso proporcionando maior velocidade na execução de cargas e descargas em lote do modelo navegacional. A forma de utilização dessa ferramenta será descrita em detalhe mais adiante.

## 4.3. Arquitetura de Implementação

O HyperDE foi construído utilizando a seguinte plataforma tecnológica:

- Ruby – linguagem de programação
- Rails – framework MVC genérico para web
- Sesame – banco de dados utilizando o modelo de dados RDF

A escolha dessa plataforma foi baseada acima de tudo em aspectos técnicos, como será detalhado nas seções a seguir. No entanto, devemos também destacar o aspecto subjetivo, baseado em uma escolha pragmática, onde se buscou uma plataforma que conferisse ao usuário e desenvolvedor uma grande produtividade na construção e manutenção, tanto do ambiente em si, como nas aplicações OOHDM/SHDM produzidas com o ambiente.

Como ambiente de desenvolvimento e prototipação, o HyperDE deve ser capaz de conferir ao usuário a qualidade de velocidade neste processo de construção, mostrando-se simples e conciso. Da mesma forma, buscamos uma plataforma com características semelhantes, como veremos a seguir.

### 4.3.1. Linguagem Ruby

Ruby é uma linguagem mais recente, criada no Japão em 1993 pelo engenheiro Yukihiro Matsumoto (conhecido como “Matz”), com o objetivo de ser uma linguagem de script dinâmica de alto nível e genuinamente orientada a

objetos. Ruby herda muitas características de linguagens como Smalltalk e Perl e é frequentemente comparada a Python.

Ruby ficou, durante muitos anos, restrita ao mercado japonês, por não existir documentação e livros em línguas ocidentais. No entanto, recentemente a linguagem tem ganhado popularidade com a publicações de livros em língua inglesa e a adoção da linguagem por profissionais respeitados da indústria do software como Martin Fowler [Fowler, 2002], Dave Thomas [Thomas & Hunt, 2001], entre outros.

Algumas características interessantes de Ruby como linguagem de programação na versão (1.8.2) usada neste trabalho são:

- Sintaxe intuitiva, simples e concisa, parcialmente inspirada em Eiffel e Ada.
- Tratamento de exceções no mesmo estilo de Java e C#.
- Os operadores são apenas uma forma amena ("*syntax sugar*") para métodos, que podem ser redefinidos à vontade.
- Totalmente orientada a objetos, sem qualquer exceção à regra. Mesmo constantes, como "true", "false", "nil" e números, são objetos.
- Utiliza herança simples, no entanto permite a mistura de módulos (que são coleções de métodos), que podem ser importados por classes, conferindo características de herança múltipla.
- Possui recursos avançados como "closures", blocos, continuações, iteradores, expressões regulares embutidas, "multi-threading", entre outros.
- É interpretada (não gera bytecode) e independente de plataforma de hardware e sistema operacional, tendo versões para DOS, Windows, Unix, Linux, Solaris, MacOS, OS/2, BSD, entre outros.
- Seu mecanismo de reflexão é trivial, o que possibilita a extensão de classes e objetos em tempo de execução muito facilmente.
- Suas classes são abertas, ou seja, você pode redefinir o comportamento de classes como Array e Integer sem ter que alterar o código-fonte original das mesmas.

- Tipagem forte, mas dinâmica, ou seja, não é necessário declarar o tipo de variáveis, mas uma vez definido, não pode ser mudado e não há conversões automáticas.
- Coletor de lixo do tipo “marca-e-limpa” que não necessita de liberação explícita de referências.

Algumas desvantagens de Ruby que podem ser citadas:

- Desempenho, por tratar-se de uma linguagem puramente interpretada. A versão 2.0 de Ruby introduzirá uma máquina virtual e o conceito de compilação “just-in-time” visando remediar este ponto.
- Variáveis globais com sintaxe pouco intuitiva, herdadas de Perl, tais como “$!”, “$@”, “$0” podem tornar a leitura difícil. No entanto, o uso dessas variáveis é opcional e equivalentes mais legíveis existem.
- O mecanismo de “multi-threading” é independente do sistema operacional e portanto, simulado, o que pode novamente causar problemas de performance.

Ruby foi escolhida como linguagem de programação para construção do HyperDE pelos seguintes aspectos técnicos:

- Por ser linguagem genuinamente orientada a objetos, Ruby permite um mapeamento de baixa impedância entre as classes navegacionais do OOHDM e SHDM para as classes em linguagem de programação. Lua, por exemplo, não possui primitivas de orientação a objetos como classes e objetos, tornando tal mapeamento mais indireto.
- Por ser linguagem de script e dinâmica, Ruby faz pouca ou nenhuma distinção entre tempo de desenvolvimento e tempo de execução, o que permite ao usuário do ambiente programar a aplicação enquanto a executa. Isso não seria possível realizar em linguagens estáticas

tais quais Java e C#, que necessitam de fase de compilação. Isto permite ciclos mais rápidos de desenvolvimento.

- Pelos seus poderosos recursos de reflexão, Ruby permite ao HyperDE a definição e redefinição de classes e métodos durante o uso do ambiente, com esforço mínimo de implementação, sem necessidade de fase de re-compilação. Mais uma vez, linguagens estáticas tais quais Java e C#, dão suporte a recursos de reflexão, no entanto requerem um grande esforço de implementação para seu uso.
- Sua simplicidade, leveza e tolerância na sintaxe permite que a curva de aprendizado seja suave nas áreas onde o usuário do HyperDE necessita utilizar linguagem de programação.

No aspecto subjetivo, podemos relatar que o aprendizado da linguagem Ruby foi feito enquanto construíamos o ambiente HyperDE, o que ocorreu de forma bastante rápida e agradável. Ruby possui um poder de expressividade alto e uma elegância de projeto e sintaxe que permite a construção de softwares utilizáveis em menos linhas de código do que outras linguagens. Vejamos a seguir exemplos de trechos de código de função equivalente em Java e em Ruby:

```
// ======== Java (song.java) ========
public class Song {

    protected String name; // atributos privado
    protected int lengthInSeconds; // atributos privado

    Song(String name, int len) { // construtor
        this.name = name;
        lengthInSeconds = len;
    }

    // acessores dos atributos
    public String getName() {
        return name;
    }

    public void setName(String str) {
        name = str;
    }

    public int getLengthInSeconds() {
        return lengthInSeconds;
    }
```

```
    public void setLengthInSeconds(int secs) {
        lengthInSeconds = secs;
    }

    // metodo para gerar representacao como String
    public String toString() {
        return name + " (" + lengthInSeconds + " seconds)";
    }

    // instanciar e imprimir
    public void main(String[] args) {
        s = new Song("name", 60);
        System.out.println(s);
    }
}
```

```
# ======== Ruby (song.rb) ========
class Song

    # acessores para escrita e leitura
    attr_accessor :name, :length_in_seconds

    # construtor
    def initialize(name, len)
        @name = name
        @length_in_seconds = len
    end

    # metodo para gerar representacao em String
    def to_s
        "#{@name} (#{@length_in_seconds} seconds)"
    end
end

# instanciar e imprimir
s = Song.new('name', 60)
puts s
```

Os trechos de código acima, absolutamente equivalentes, ilustram de forma superficial a poder expressivo da linguagem Ruby, conseguido neste caso específico através de sua capacidade de metaprogramação, onde a linha “attr_accessor :name, :length_in_seconds”, define 2 atributos privados, assim como os 4 métodos para leitura e escrita desses atributos. O comando “attr_accessor”, que aparenta ser uma palavra-chave da linguagem, na verdade é um método da meta-classe “Class” que visa criar atributos e os métodos acessores

correspondentes dinamicamente, como se fosse uma “macro”. Em Java, o equivalente só é conseguido em 10 linhas de código.

### 4.3.2. Framework Rails

Rails é um framework MVC “open-source”, completo para construção de aplicações Web genéricas. Totalmente escrito em linguagem Ruby, Rails é composto de 3 componentes independentes responsáveis por cada camada do padrão de projeto modelo-visão-controle. São eles:

- ActiveRecord: componente responsável por realizar o mapeamento objeto-relacional entre um banco de dados e objetos nativos da linguagem Ruby. ActiveRecord utiliza instrospecção sobre o esquema da base de dados para gerar automaticamente as declarações de atributos das classes, relacionamentos e outros métodos para operações CRUD. Este componente elimina quase completamente a necessidade de escrever código artesanal para realizar as operações básicas de persistência dos objetos da camada de modelo da aplicação.
- ActionController: componente responsável por receber as requisições HTTP e fornecer uma API de alto nível para o tratamento desses eventos. O ActionController mapeia requisições HTTP para classes (chamadas de “controllers”) e métodos (chamadas de “actions”), baseados na URL da requisição e fornece uma série de funções auxiliares para o tratamentos das requisições com o objetivo de coordenar a atualização entre as visões e objetos do modelo da aplicação. O ActionController é responsável ainda por funções de “cache”, gerenciamento de sessões, “cookies”, “layouts”, entre outros. Por fazer um mapeamento direto entre requisições HTTP e métodos de classes, o ActionController caracteriza um acesso a objetos através de protocolo REST.
- ActionView: componente responsável por construir a saída de uma requisição HTTP, utilizando mecanismo de “templates” denominado “Embedded Ruby”, onde o código HTML é intercalado com código

> Ruby para produzir saídas dinâmicas. Esse componente fornece ainda funções auxiliares para o desenho de saídas HTML e XML, facilitando a produção de páginas e formulários para Web.

A filosofia por trás do projeto de Rails é alinhada com a filosofia do ambiente HyperDE, em termos de praticidade, facilidade de uso e produtividade. A escolha por construir o framework MNVC do HyperDE baseada neste framework nos proporcionou maior robustez e agilidade no desenvolvimento do ambiente e permitiu nos concentrarmos diretamente na resolução de problemas do nosso domínio, ao invés de termos que investir tempo na construção de toda a infra-estrutura do ambiente.

Além disso, Rails tem código aberto e é suficientemente conciso para que pudesse ser adaptado às necessidades do HyperDE. As necessidades de adaptação de Rails se concentraram particularmente no componente ActiveRecord, cuja implementação original era destinada ao mapeamento de objetos em cima de bancos de dados relacionais, utilizando linguagem de consulta SQL e primitivas relacionais como tabelas e colunas.

Grande parte deste componente, portanto, precisou ser reescrito e acabou sendo renomeado para SemanticRecord, para dar suporte a base de dados semântica e as primitivas RDF(s), assim como a linguagem de consulta SeRQL utilizada no HyperDE. No entanto, a compatibilidade de interface com o componente original foi mantida.



### 4.3.3. Base de dados Sesame

Sesame é um banco de dados semântico de código-aberto, escrito em Java que utiliza RDF para armazenamento de informações e suporta nativamente consultas e inferências de declarações RDF(s).

Sesame possui uma linguagem de consulta denominada SeRQL (pronunciada como a palavra inglesa “circle”), que é baseada na linguagem RQL, com algumas extensões proprietárias e uma sintaxe mais concisa.

Escolhemos Sesame como base de dados para o ambiente HyperDE pelos seguintes aspectos:

- Por possuir suporte nativo a esquemas declarados via RDF(s), linguagem que acomoda perfeitamente o metamodelo do SHDM.
- Por possuir uma interface de acesso através de protocolo HTTP, o que permite arquiteturas físicas bastante flexíveis além de independência de plataforma de aplicações clientes. Jena2, por exemplo, não oferece este recurso, o que nos obrigaria a utilizar a plataforma Java no restante do framework do ambiente HyperDE ou ter que escrever um adaptador para interface via HTTP próprio.
- Pela sua linguagem de consulta SeRQL fornecer inferências nativas a esquemas RDF(s), como por exemplo, resolução de heranças entre classes. Além disso, SeRQL possui uma sintaxe mais simples e expressiva do que alternativas como RDQL e RQL.
- Por oferecer um nível bastante satisfatório de desempenho, principalmente nas operações de leitura, que são as operações predominantes utilizadas pelo HyperDE.
- Por oferecer uma interface visual Web de acesso aos repositórios de dados, o que facilitou imensamente a validação dos modelos do ambiente.



## 4.4. Modelagem Navegacional

Veremos a seguir os detalhes dos recursos oferecidos pelo ambiente HyperDE para construir uma modelagem navegacional OOHDM/SHDM.

A modelagem navegacional no HyperDE é desenvolvida através da instanciação de metaclasses representativas das seguintes primitivas do modelo OOHDM/SHDM:

- Classes navegacionais, seus atributos e operações
- Elos
- Contextos navegacionais
- Índices e seus atributos
- Landmarks
- Nós e seus relacionamentos

Veremos em detalhe cada uma dessas metaclasses nas seções a seguir.

### 4.4.1. Classes navegacionais, atributos, operações e elos

A seguir, apresentamos as metaclasses utilizadas para definição completa de uma classe navegacional em uma aplicação desenvolvida no HyperDE.

#### 4.4.1.1. Metaclasse NavClass – classes navegacionais

Classes navegacionais do metamodelo OOHDM/SHDM são representadas através de instâncias da metaclasse NavClass, que por sua vez possui 3 agregações:

- Objetos da metaclasse NavAttribute, para atributos navegacionais;
- Objetos da metaclasse NavOperation, para operações;
- Objetos da metaclasse Link, para elos;

O ambiente HyperDE gera, dinamicamente, o modelo navegacional da aplicação baseando-se nas instâncias dessas metaclasses. A cada modificação dessas instâncias, o modelo navegacional é re-gerado para refletir as alterações feitas. A geração dinâmica do modelo navegacional ocorre em tempo de execução do ambiente, sem que seja necessária qualquer interrupção de uso para reinício do ambiente ou fase de compilação.

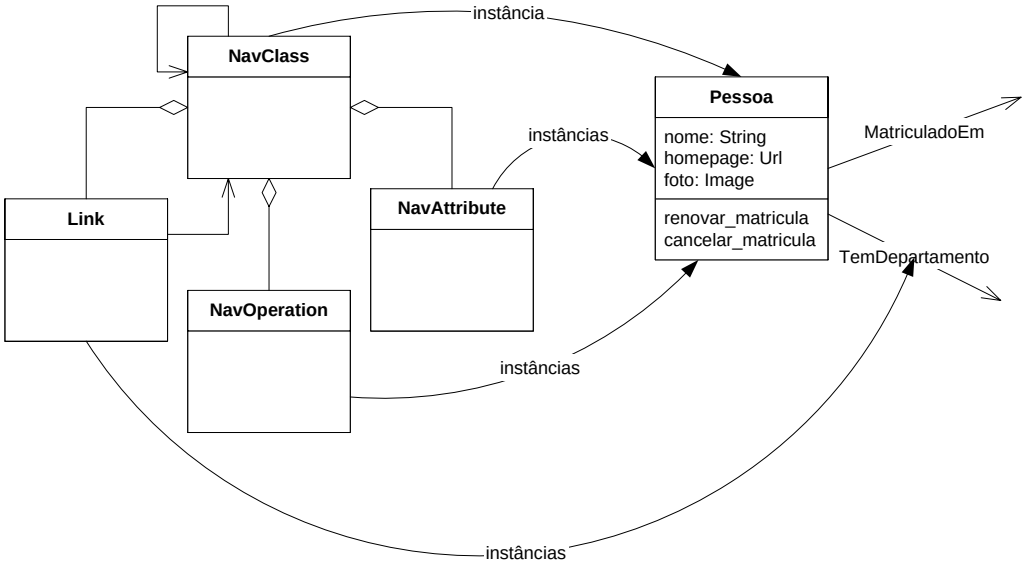


Figura 10 – Instâncias de metaclasses são geradoras das classes navegacionais

A geração do modelo navegacional é iniciado pela introspecção das instâncias da metaclasse NavClass, cujos atributos são:

- name, que deve conter o nome da classe navegacional que será gerada no modelo da aplicação e que deve ser iniciado por letra maiúscula.
- label_attribute, deve conter o nome do atributo navegacional a ser utilizado pelo HyperDE para representação de instâncias dessa classe como “string”.

NavClass possui os seguintes relacionamentos:

- base_class (associação), que aponta para uma instância de NavClass, que será utilizada como superclasse da instância corrente durante a geração do modelo navegacional.
- nav_attributes (agregação), que aponta para uma coleção de instâncias da metaclasse NavAttribute e representam os atributos navegacionais da classe gerada.
- nav_operations (agregação), que aponta para uma coleção de instâncias da metaclasse NavOperation e representam as operações da classe gerada.

- links (agregação), que aponta para uma coleção de instâncias da metaclasse Link e representam a definição dos elos entre as classes geradas.

Vejamos a seguir um exemplo de construção de uma classe navegacional utilizando a linguagem específica de domínio do *framework* HyperDE em Ruby:

```
professor = NavClass.new
professor.name = “Professor”
professor.label_attribute = “nome”
professor.base_class = Pessoa.nav_class
```

#### 4.4.1.2. Metaclasse NavAttribute – atributos de classes navegacionais

As instâncias da metaclasse NavAttribute e suas subclasses definem os atributos das classes navegacionais geradas dinamicamente pelo HyperDE. NavAttribute é utilizada para definir atributos de tipos de dados simples. Suas subclasses, tais como ComputedNavAttribute, AnchorNavAttribute, ContextAnchorNavAttribute, IndexAnchorNavAttribute, IndexNavAttribute definem, respectivamente, os atributos de tipo computado, âncora, âncora para contexto, âncora para índice e índice. A hierarquia de herança dessas metaclasses é ilustrada a seguir:

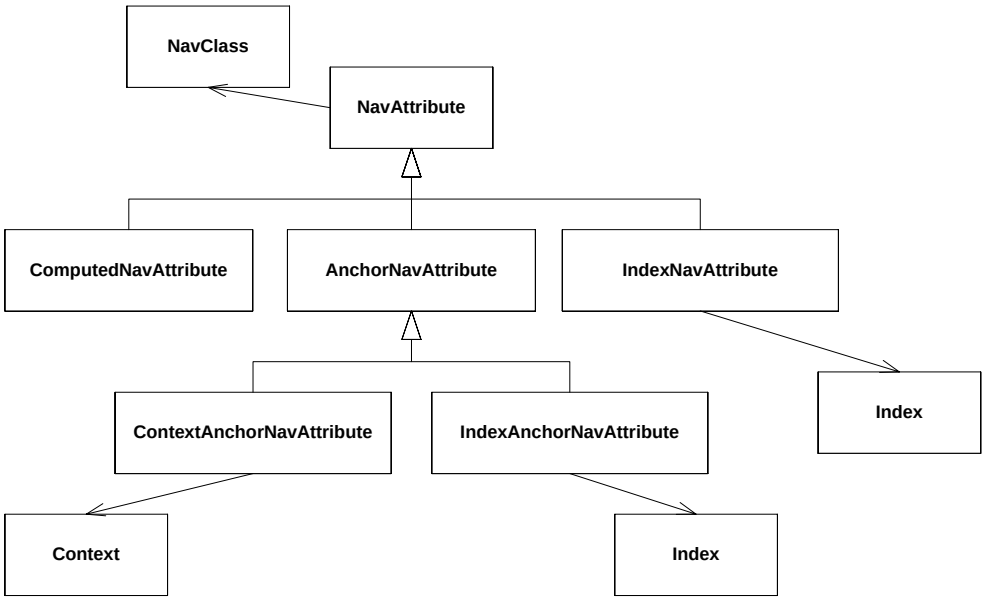


**Figura 11 – Hierarquia das metaclasses derivadas de NavAttribute e seus relacionamentos diretos**



Os atributos de NavAttribute são:

- name: nome do atributo a ser gerado, deve ser iniciado por letra minúscula.
- data_type: tipo do dado para o atributo, escolhido dentre as opções suportadas pelo ambiente.
- position: indicador da posição do atributo para ser usado em critérios de ordenação pelo ambiente de desenvolvimento.

Os tipos de dados suportados pelo HyperDE para NavAttribute são:

- String: textos curtos, geralmente de uma linha.
- Text: textos longos.
- Html: documentos em linguagem HTML.
- Xml: documentos em linguagem XML.
- Integer: números inteiros.
- Float: números de ponto flutuante.
- Boolean: verdadeiro ou falso.
- Date: apenas data (dia, mês e ano).

- Time: data e hora (incluindo hora, minuto, segundo, milissegundo).
- Array: uma lista de valores, heterogênea.
- Hash: uma lista composta de associações chave-valor.
- Url: um endereço na Web.
- Image: um endereço de uma imagem em qualquer formato suportado pelos navegadores atuais.
- Email: um endereço de correio eletrônico.
- Movie: um endereço de um filme em qualquer formato suportado pelos navegadores atuais.
- Áudio: um endereço de um áudio em qualquer formato suportado pelos navegadores atuais.
- Flash: um endereço de uma animação em formato Macromedia Flash.
- Applet: um endereço de um aplicativo Java do tipo Applet.
- Other: qualquer outro tipo de dado não definido pelos tipos acima.

NavAttribute possui relação que aponta para uma instância de NavClass.

A seguir, um exemplo em Ruby de uma instância de NavAttribute onde definimos um atributo “foto” de uma classe “Pessoa”:

```
atr_foto = NavAttribute.new
atr_foto.nav_class = Pessoa.nav_class
atr_foto.name = “foto”
atr_foto.data_type = “Image”
```

ComputedNavAttribute define um atributo de tipo computado, onde o valor do atributo é computado em tempo de execução a partir de um trecho de código Ruby. Ele possui um único atributo além dos herdados de NavAttribute:

- Code: trecho de código em Ruby para cálculo do valor do atributo em tempo de execução.

A seguir, um exemplo em Ruby de uma instância de ComputedNavAttribute onde definimos o cálculo da nota média para nós da classe “Aluno” (assumimos que a classe “Aluno” possui a definição do elo “notas” usada abaixo):

```
atr_nota = ComputedNavAttribute.new
atr_nota.nav_class = Aluno.nav_class
atr_nota.name = “nota_media”
atr_nota.code = “””
    soma = 0
    self.notas.each { |n| soma += n }
    media = soma / self.notas.length
”””
```

O AnchorNavAttribute é uma metaclasse abstrata e possui apenas uma associação com uma instância da classe Link, que define o elo a ser usado para calcular o nó alvo da âncora em relação ao nó de origem.

ContextAnchorNavAttribute é uma das subclasses concretas de AnchorNavAttribute e define um atributo do tipo “âncora para contexto”, onde o alvo da âncora é um nó em determinada posição de um dado contexto pré-definido. O nó do contexto alvo é calculado através do elo associado à âncora. Portanto, esta metaclasse possui relação de associação com a metaclasse Context e seus atributos adicionais (além dos herdados de NavAttribute) são:

- params: define uma expressão em linguagem Ruby, avaliada em tempo de execução sob o contexto do nó de origem, para cálculo dos valores dos parâmetros a serem passados para o contexto navegacional alvo. Esse atributo é opcional e só é utilizado quando o contexto alvo é parametrizado (grupos de contexto em OOHDM/SHDM).
- label: define uma expressão em linguagem Ruby, avaliada em tempo de execução sob o contexto do nó alvo, para determinar o valor da etiqueta da âncora a ser apresentada ao usuário.

A seguir, um exemplo em Ruby onde definimos um atributo de âncora para contexto na classe “Aluno” apontando para o contexto “ProfessorAlfa” (professores, em ordem alfabética), onde o nó alvo é calculado a partir do valor do elo “EhOrientadoPor” e a etiqueta da âncora definida como o atributo “nome” do nó alvo (classe “Professor”).

```
anc = ContextAnchorNavAttribute.new
anc.nav_class = Aluno.nav_class
```

```
anc.name = “orientador”
anc.link = Link.find_by_name “EhOrientadoPor”
anc.context = Context.find_by_name “ProfessorAlfa”
anc.label = “self.nome” # para exibir o nome do professor
```

IndexAnchorNavAttribute é a outra metaclasse concreta derivada de AnchorNavAttribute e define um atributo do tipo “âncora para índice”, onde o alvo da âncora é um índice pré-definido (instância da metaclasse Index). Esta metaclasse possui relação de associação com a metaclasse Index e seus atributos, além dos herdados, são:

- params: define uma expressão em linguagem Ruby, avaliada em tempo de execução sob o contexto do nó de origem, para cálculo dos valores dos parâmetros a serem passados para o índice alvo. Esse atributo é opcional e só é utilizado quando o índice alvo é parametrizado.
- label: define uma String a ser utilizada como valor da etiqueta da âncora para o índice. Este valor é opcional e caso não esteja definido, o HyperDE assumirá o nome do índice alvo como seu valor.

A seguir vemos um exemplo de IndexAnchorNavAttribute, onde definimos um atributo de âncora para índice para a classe “Departamento”, tendo como alvo o índice “AlunosPorDepartamento”. Este índice recebe como parâmetro o valor do identificador do departamento sendo navegado e a etiqueta da âncora é definida como “Alunos do Departamento”.

```
anc = IndexAnchorNavAttribute.new
anc.nav_class = Departamento.nav_class
anc.name = “alunos”
anc.index = Index.find_by_name “AlunosPorDepartamento”
anc.params = “self.id”
anc.label = “Alunos do Departamento”
```

IndexNavAttribute é a metaclasse que define um atributo do tipo índice onde os elementos do índice serão parte integrante do nó sendo navegado. Isto é diferente de âncora para índice, onde é exibida apenas uma âncora para navegar

ao índice alvo e não os elementos do índice em si. Os atributos de IndexNavAttribute são os mesmos de IndexAnchorNavAttribute, com exceção do atributo “label”.

A seguir vemos um exemplo de IndexNavAttribute, onde definimos que a classe Departamento possui um atributo de índice “ProfessoresPorDepartamento”. Este índice recebe como parâmetro o valor do identificador do próprio nó sendo navegado.

```
index = IndexNavAttribute.new
index.nav_class = Departamento.nav_class
index.index = Index.find_by_name “ProfessoresPorDepartamento”
index.params = “self.id”
```

#### 4.4.1.3. Metaclasse NavOperation – operações de classes navegacionais

Instâncias da metaclasse NavOperation definem operações que podem ser ativadas em instâncias de nós das classes navegacionais da aplicação. Ela possui relação de associação com a metaclasse NavClass e seus atributos são:

- name: nome da operação, que deve ser iniciada por letra minúscula.
- params: nome dos parâmetros, separados por vírgula, que devem ser passados para a operação.
- code: trecho de código em Ruby que é avaliado sob o contexto de execução do nó corrente.
- position: indicador opcional da posição da operação para ser usado em critérios de ordenação pelo ambiente de desenvolvimento.

A seguir vemos um exemplo de operação que cria uma instância de elo do tipo “MatriculadoEm” entre um nó da classe “Aluno” e um nó da classe “Curso”:

```
op = NavOperation.new
op.nav_class = Aluno.nav_class
op.name = “matricular”
op.params = “curso_id”
op.code = “self.matriculado_em << Curso.find(curso_id)”
```

Quando a operação de uma classe navegacional é ativada através do controlador de navegação durante ação de navegação contextual, o contexto de navegação corrente, através de uma instância da classe “Context”, fica disponível para o contexto de execução do código implementado na operação através da variável “@context”.

### 4.4.1.4. Metaclasse Link – definição de elos

Instâncias da metaclasse Link são usadas pelo HyperDE para gerar as classes de elos navegacionais da aplicação. Ela possui dois relacionamentos com a metaclasse NavClass, um para indicar a classe de origem do elo e outro para indicar a classe alvo.

Além disso, possui opcionalmente uma auto-associação para indicar uma outra classe de elo de semântica inversa. Esta associação de semântica inversa é utilizada pelo ambiente como facilitador da seguinte forma: quando uma instância de uma classe de elo é criada e a mesma possui um elo de semântica inversa, o ambiente automaticamente cria também a instância de elo inverso. Isso ocorre também nas operações de exclusão de elos.

As associações da metaclasse Link e suas relações com as classes navegacionais geradas na aplicação estão ilustradas a seguir:

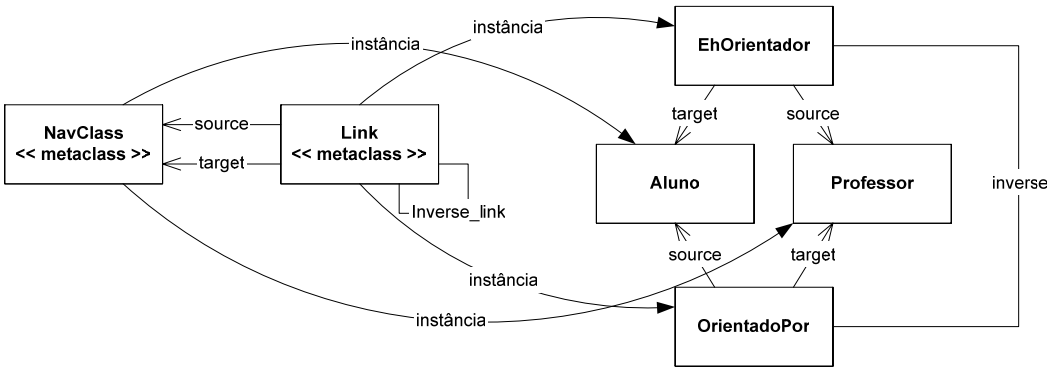


**Figura 12 – Associações da metaclasse Link e suas relações com as classes geradas na aplicação**

Os atributos da classe Link são:

- name: nome da classe de elo a ser gerada na aplicação. Deve ser iniciada por letra maiúscula.
- position: indicador opcional de posicionamento do elo para uso em critérios de ordenação pelo ambiente de desenvolvimento.

A seguir, vemos um exemplo da definição da classe de elo “OrientadoPor”, conforme ilustrado no diagrama anterior:

```
lnk = Link.new
lnk.name = “OrientadoPor”
lnk.nav_class = Aluno.nav_class # classe nav de origem do elo
lnk.target_nav_class = Professor.nav_class # classe destino
lnk.inverse_link = EhOrientador.link_class
```

A cardinalidade dos elos definidos no HyperDE é sempre um-para-muitos.

### 4.4.2. Contextos Navegacionais

Os contextos navegacionais formam a base de modelo navegacional de uma aplicação OOHDM/SHDM. Qualquer classe navegacional só pode ser navegada na forma de um nó, se este estiver dentro de um contexto navegacional.

Um contexto navegacional no HyperDE é formado basicamente por sua expressão de consulta, declarada na linguagem apropriada para que os nós que fazem parte do contexto sejam selecionados da base de dados, responsável pela persistência dos mesmos.

Contextos navegacionais também podem servir como base para construção de estruturas de acesso do tipo índice derivado de contexto, o qual veremos em seção posterior.

A expressão de consulta do contexto pode conter parâmetros, cujos valores serão fornecidos em tempo de execução durante a navegação na aplicação.

Um contexto navegacional é representando na camada de modelo navegacional do framework do HyperDE através da metaclasse “Context”. Esta metaclasse possui uma relação de agregação com instâncias da metaclasse “ContextParameter”, que por sua vez é a subclasse de “Parameter” e possui uma associação com a metaclasse “NavClass”. Esta associação indica o tipo de dado

do parâmetro, que pode ser um tipo de classe navegacional ou um dado simples. O diagrama a seguir ilustra este esquema:

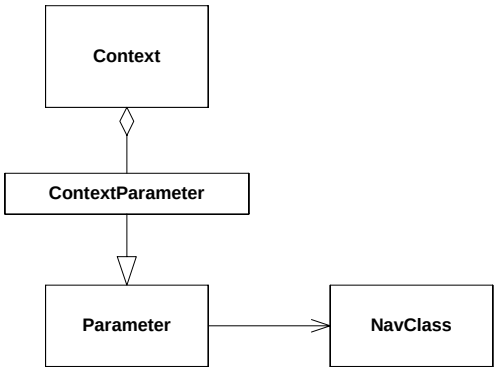


**Figura 13 – Metaclasses Context e ContextParameter e seus relacionamentos**

Os atributos de Context são:

- name: nome do contexto pelo qual o mesmo é identificado.
- query: expressão de consulta do contexto navegacional.

A metaclasse Parameter possui um único atributo, herdado por ContextParameter, além de suas relações:

- name: nome do parâmetro pelo qual o mesmo é identificado.

As expressões de consulta utilizadas no HyperDE são declaradas utilizando linguagem SeRQL, em virtude da utilização da base de dados para RDF, Sesame. Uma expressão de consulta para um contexto navegacional é válida, desde que o resultado da consulta, retornado pela base de dados, contenha apenas tuplas contendo referências a nós, podendo estes ser instâncias de qualquer classe navegacional.

Vejamos um exemplo de consulta em SeRQL que retorna nós da classe “Professor” que possuem elo do tipo “Trabalha” com um determinado nó da classe “AreaDePesquisa” que será passado como parâmetro em tempo de execução.

```
SELECT DISTINCT id, type FROM {id} rdf:type {<sr:Professor>};
serql:directType {type},
{link} sr:node_id {id}; rdf:type {<sr:Trabalha>};
sr:target_node_id {?} rdf:type {AreaDePesquisa}
USING NAMESPACE sr = <http://sr#>
```

Para facilitar a construção de consultas, o HyperDE fornece um módulo de auxílio chamado “QueryBuilder” que visa abstrair a sintaxe da linguagem de consulta utilizada para as expressões de consulta mais comuns. Essa abordagem procura minimizar a necessidade de conhecimento da linguagem de consulta e possibilita ainda a troca da linguagem de consulta sem que as consultas tornem-se inválidas. Vejamos um exemplo completo da construção de um contexto navegacional utilizando este módulo de auxílio a seguir:

```
query = QueryBuilder.x_by_y_sorted_by_attr(Professor, Trabalha,
Departamento, “nome”)
ctx = Context.new
ctx.name = “ProfessorPorDepartamento”
ctx.query = query.expression
ctx.parameters = query.parameters
```

A construção da expressão da consulta de contexto acima utiliza um *template* de consulta bem comum do tipo “classe por classe através de um elo, ordenada por um atributo” (contexto derivado de elo em OOHDM), e recebe os parâmetros adequados para construção da mesma: a classe navegacional de destino, o tipo de elo, a classe navegacional de origem e o atributo da classe de destino pelo qual será feita a ordenação.

A metaclasse “Context”, assim como outras metaclasses do *framework* HyperDE, contém uma coleção de métodos (tais como *position*, *nodes*, *next*, *previous*, *first?*, *last?*, *empty?*) que são avaliados apenas durante a execução da navegação, mas que não são relevantes para a definição de um contexto navegacional. Esses métodos serão tratados em detalhe na seção de documentação do API do *framework*.

### 4.4.3. Estruturas de Acesso: Índices

A forma mais comum de iniciar uma navegação contextual é dada através de estruturas de acesso, os índices. Os elementos que formam um índice

normalmente possuem âncoras que podem ter como alvo outros índices ou um nó em um contexto navegacional. Os elementos de um índice podem ser formados por uma composição de valores, que são definidos pelos atributos do índice.

Portanto, índices possuem atributos e elementos. Podemos fazer uma analogia entre índices e tabelas, onde a definição das colunas de uma tabela corresponde à definição dos atributos de um índice, as linhas da tabela correspondem aos elementos do índice e as células da tabela correspondem aos valores dos atributos em cada elemento do índice.

No *framework* HyperDE, essas primitivas de estruturas de acesso são representadas através das metaclasses abstratas "Index", "IndexAttribute", "IndexEntry" e "IndexEntryAttribute", respectivamente para índice, atributos de índice, elementos de índice e valores dos atributos em cada elemento do índice.

Vejamos a seguir um diagrama que ilustra esse modelo:

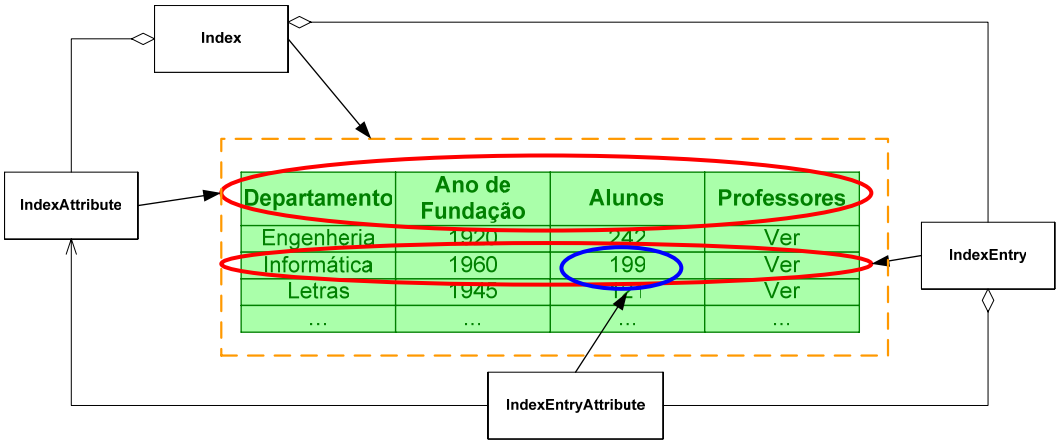


Figura 14 – Relações entre as metaclasses que formam uma estrutura de acesso

O HyperDE suporta 2 tipos de índices, representados por classes concretas derivadas da metaclasse abstrata "Index":

- Índice derivado de consulta: é o índice em que seus elementos são definidos com base em uma expressão de consulta livre, cujo resultado é retornado pela base de dados. É representado no *framework* pela metaclasse "QueryIndex".
- Índice derivado de contexto: é o índice em que seus elementos são selecionados com base nos elementos de um contexto navegacional.

Na prática, é similar a um índice derivado de consulta, onde a expressão de consulta é retirada de um contexto navegacional associado. É representado no *framework* pela metaclasse "ContextIndex".

A metaclasse "IndexAttribute" representa um atributo de índice simples, ou seja, um atributo cujos valores não resultam em uma âncora, mas apenas um valor informativo.

As subclasses de "IndexAttribute" são "ContextAnchorIndexAttribute" e "IndexAnchorIndexAttribute", cujos valores dos elementos resultam em âncoras para contexto e índice respectivamente.

A metaclasse "IndexEntry" representa um elemento de um índice. "IndexEntry" possui uma relação de agregação para instâncias da metaclasse "IndexEntryAttribute", representantes dos valores dos atributos em cada elemento do índice.

"IndexEntryAttribute" representa um valor de um atributo simples definido pela metaclasse "IndexAttribute". Por sua vez, as subclasses "ContextAnchorIndexEntryAttribute" e "IndexAnchorIndexEntryAttribute" são as representantes de valores para atributos definidos pelas metaclasses "ContextAnchorIndexAttribute" e "IndexAnchorIndexAttribute" respectivamente.

Para elucidar este modelo, vejamos o diagrama a seguir:

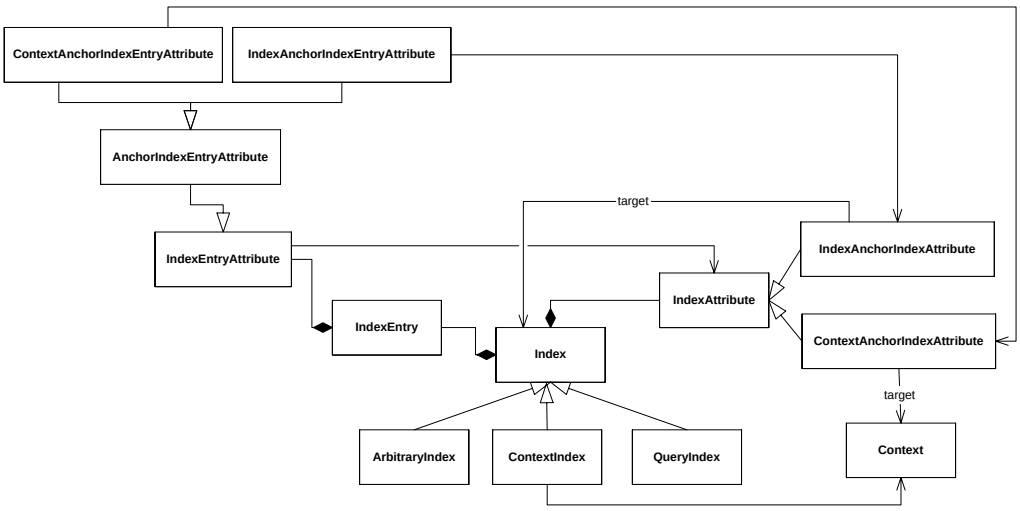


Figura 15 – Modelo do *framework* para definição de estruturas de acesso (índices)

Os atributos da metaclasse “Index” e suas subclasses são:

- name: nome do índice.
- query (apenas para QueryIndex): definição da expressão de consulta do índice.

Os atributos da metaclasse “IndexAttribute” e suas subclasses são:

- name: nome do atributo para o índice.
- label: expressão a ser avaliada em tempo de execução sob o contexto de execução de cada elemento do índice para determinar o valor textual do atributo em cada elemento índice.
- target_context_position (apenas para ContextAnchorIndexAttribute): posição do nó de entrada para o contexto navegacional alvo do atributo. Usado apenas quando se deseja apontar uma âncora para um nó fixo dentro de um contexto para todos os elementos do índice.
- target_context_expression (apenas para ContextAnchorIndexAttribute): expressão avaliada em tempo de execução sob o contexto de execução de cada elemento do índice para determinar dinamicamente a posição do nó de entrada para o contexto navegacional alvo do atributo.
- params (apenas para ContextAnchorIndexAttribute e IndexAnchorIndexAttribute): expressão avaliada em tempo de execução sob o contexto de execução de cada elemento do índice para determinar o valor dos parâmetros a serem passados para o contexto ou índice alvo caso sejam baseados em consultas parametrizadas.
- position: indicação do posicionamento do atributo do índice em relação aos demais atributos do mesmo índice. Este atributo permite definir, por exemplo, a ordem de desenho das colunas, caso o índice seja desenhado em forma de tabela.

As expressões de consultas nas quais os índices se baseiam são avaliadas dinamicamente a cada acesso ao índice, assim como os valores dos atributos de seus elementos. Isso garante que o índice sempre reflete entradas válidas para os nós em seus contextos e índices.

Vejamos a seguir alguns exemplos de definição de índices em Ruby. O primeiro exemplo define um índice derivado de contexto com um atributo:

```
idx = ContextIndex.new
idx.context = Context.find_by_name “AlunoPorArea”
idx.name = “AlunosPorArea”
idx.params = “self.id”
idx_attr = ContextAnchorIndexAttribute.new
idx_attr.name = “nome”
idx_attr.label = “self.nome” #avaliada em cada elemento do índice
idx.index_attributes << idx_attr
```

O índice definido acima lista todos os alunos de uma área, exibindo os nomes dos alunos como âncoras para o contexto “AlunoPorArea”.

O exemplo a seguir define um índice mais complexo, derivado de consulta, com 3 atributos:

```
idx = QueryIndex.new
# a query a seguir retorna uma tabela de 3 colunas, que sao,
# o `id`, `nome` e `email` do professor.
idx.query = “....” # uma expressão em SeRQL
idx.name = “Professores”
idx_attr1 = ContextAnchorIndexAttribute.new
idx_attr1.name = “nome”
idx_attr1.label = “self.nome”
idx_attr1.target_context = Context.find_by_name “ProfessorAlfa”
idx_attr1.target_context_expression = “self.id”
idx_attr2 = IndexAttribute.new
idx_attr2.name = “email”
idx_attr2.label = “self.email”
idx_attr3 = IndexAnchorIndexAttribute.new
idx_attr3.name = “alunos”
idx_attr3.label = “’Alunos’”
idx_attr3.params = “self.id”
idx_attr3.target_index = Index.find_by_name “AlunosPorProfessor”
idx.index_attributes << idx_attr1 << idx_attr2 << idx_attr3
```

O índice definido acima lista todos os professores, com cada elemento do índice tendo como atributos o nome do professor como âncora para o contexto

"ProfessorAlfa", o email do professor como um texto simples e uma âncora para o índice "AlunosPorProfessor" com a *string* 'Alunos' como etiqueta da âncora.

### 4.4.4. *Landmarks* – pontos de acesso globais

Landmarks são pontos de acesso globais sempre visíveis durante a navegação. Landmarks podem ter âncoras apontando para nós em contextos ou para índices. Landmarks são representados no framework HyperDE através da classe abstrata "Landmark".

A classe "Landmark" possui 2 subclasses: "ContextLandmark" e "IndexLandmark", representando landmarks com âncoras para nó em contexto ou para índice, respectivamente, conforme ilustrado no diagrama a seguir:

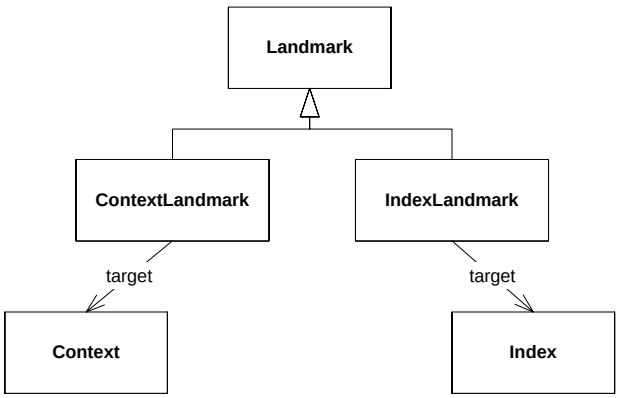


**Figura 16 – Relações da classe "Landmark"**

Os atributos da classe "Landmark" e suas subclasses são:

- name: nome do landmark.
- label: expressão avaliada em tempo de execução sob o contexto de execução da instância de "Landmark" para definir o texto de etiqueta para a âncora gerada. Geralmente é definida apenas como uma string constante.
- params: expressão avaliada em tempo de execução sob o contexto de execução da instância de "Landmark" para definir o valor dos parâmetros a serem passados para o contexto ou índice alvo, caso os mesmos estejam baseados de consultas parametrizadas.

- position: indicação de posicionamento para ordenação relativa aos outros landmarks definidos.
- target_context_position (apenas para ContextLandmark): posição do nó de entrada para o contexto navegacional alvo do landmark. Usado quando se deseja apontar uma âncora para um nó fixo dentro de um contexto.
- target_context_expression (apenas para ContextLandmark): expressão avaliada em tempo de execução sob o contexto de execução da instância do "Landmark" para determinar dinamicamente a posição do nó de entrada para o contexto navegacional alvo do âncora gerada no landmark.

A seguir vemos um exemplo de definição mais comum de um landmark apontando para um índice:

```
lm = IndexLandmark.new
lm.name = 'Areas'
lm.label = '"Áreas (#{Area.count})"'
lm.index = Index.find_by_name "Areas"
lm.position = 0
```

Note que o atributo "label" é definido como uma expressão que irá retornar a string "Áreas", seguido do número de áreas existentes na base de dados. Por exemplo: "Áreas (18)".



### 4.4.5.Nós – os objetos da aplicação

Os nós de uma aplicação OOHDM/SHDM são efetivamente os itens que são navegados durante a execução da aplicação no HyperDE. Nós representam objetos navegacionais, instâncias das classes navegacionais definidas através das metaclasses "NavClass", "NavAttribute", "NavOperation" e "Link".

No framework HyperDE, um nó é representado pela classe abstrata "Node". Todas as classes navegacionais geradas dinamicamente pela aplicação são de fato subclasses de "Node". Da mesma forma, as classes de elo, geradas dinamicamente a partir de instâncias da metaclasse "Link", são subclasses da classe abstrata "NodeLink", que representam portanto os valores dos elos entre nós. Os valores

dos atributos dos nós são representados através de instâncias da classe “NodeAttribute”. Estas relações estão ilustradas no diagrama a seguir:

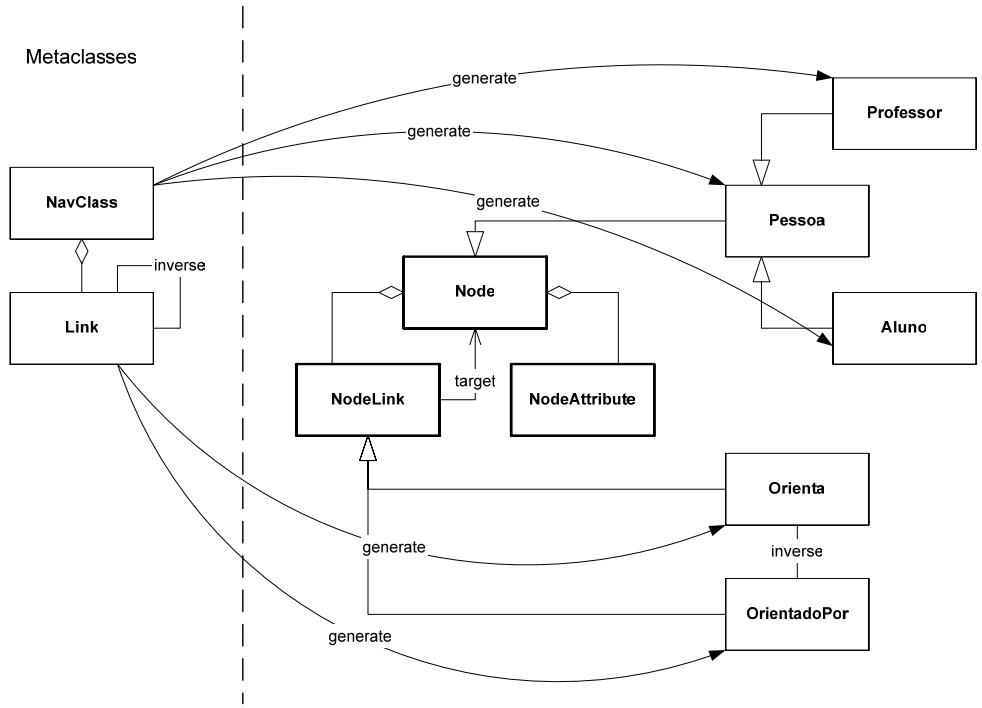


Figura 17 – Relação das classes “Node”, “NodeLink” e “NodeAttribute” como superclasses das classes navegacionais geradas a partir das metaclasses do framework.

A classe “Node” possui apenas 2 atributos nativos, somente de leitura. Os atributos restantes para cada classe navegacional são gerados dinamicamente a partir da metaclasse “NavAttribute”. São eles:

- created_at: data e hora de criação do nó.
- updated_at: data e hora da última atualização do nó.

A classe “NodeLink” possui, além das associações de nó de origem e nó alvo, apenas um atributo:

- position: para indicar opcionalmente o posicionamento do valor do elo relativo a outros elos do mesmo tipo originados de um nó, caso o conjunto de elos do nó seja ordenado.

A classe “NodeAttribute” possui, além das associações ao atributo e ao nó aos quais o valor do atributo se refere, apenas um atributo:

- value: valor do atributo.

Apesar do *framework* HyperDE utilizar internamente as classes “Node”, “NodeAttribute” e “NodeLink” para representação dos nós e os valores de seus atributos e elos, o usuário do HyperDE não precisa lidar diretamente com tais classes. O framework gera dinamicamente as classes navegacionais de forma a tornar transparente o mecanismo interno de representação. O usuário lida com as classes navegacionais da mesma forma que lidaria com qualquer outra classe do framework, acessando diretamente seus atributos e métodos.

Vejamos a seguir um exemplo de tal manipulação:

```
al = Aluno.new #jamais se instanciara diretamente “Node”

# os valores dos atributos manipulados a seguir são
# internamente representado por NodeAttribute

al.matricula = “0210503”
al.nome = “Demetrius Nunes”
al.email = “demetrius@interface-ti.com.br”
al.homepage = “http://www.interface-ti.com.br”
al.foto = “http://interface-ti.com.br/fotos/demetrius.jpg”
al.save

# os valores dos elos manipulados a seguir são internamente
# representados por NodeLink

al.eh_orientado_por << Professor.find_by_nome(“Daniel Schwabe”)
al.eh_orientado_por[0].nome # => “Daniel Schwabe”

# a operação executada a seguir é definida internamente
# por uma instância da metaclasse NavOperation

al.matricular Curso.find_by_titulo(“OOHDM”)
```

No exemplo anterior, podemos ver como o framework HyperDE fornece uma linguagem específica de domínio para manipulação de atributos, elos e operações, como se os mesmos fossem atributos e métodos nativos das classes navegacionais. A geração da classe navegacional obedece as seguintes regras:

- Instâncias da metaclasse “NavAttribute” são transformadas em atributos nativos simples da classe navegacional.
- Instâncias da metaclasse “Link” são transformadas em atributos nativos de tipo “Array”, onde estão disponíveis todas as operações oferecidas pela classe “Array”, incluindo iteração de elementos e adição de elementos através do operador “<<”.
- Instâncias da metaclasse “NavOperation” são transformadas em métodos nativos da classe navegacional.

É importante ressaltarmos que, como as classes navegacionais são subclasses de “Node”, esta por sua vez (assim como todas as classes do framework que necessitam de serviços de persistência) é subclasse de SemanticRecord::Base. Dessa forma os métodos de persistência (como “save”, “destroy” e outros) estão disponíveis para todas as classes navegacionais geradas pelo HyperDE. Esses métodos serão vistos em detalhe na seção de documentação da API do framework.

Note ainda que os métodos estáticos de busca do tipo “find_by_???” são gerados dinamicamente baseados nos atributos de cada classe navegacional.

A classe “Node” fornece ainda um mecanismo de “call-back” para a interceptação de eventos importantes, como o evento de navegação sobre o nó e eventos relacionados à persistência do mesmo. Dessa forma, o usuário do HyperDE pode implementar em suas classes navegacionais comportamentos atrelados a esses eventos através da adição de operações do mesmo nome.

Os eventos que oferecem call-backs são:

- “create”: evento que ocorre no ato de persistência de um novo nó. Seus call-backs são “before_node_create” e “after_node_create” chamados antes e depois da ocorrência do evento, respectivamente. Estes call-backs, assim como todos os outros, devem ser implementados na forma de operações da classe navegacional.
- “update”: evento que ocorre no ato de persistência de um nó já existente. Seus call-backs são “before_node_update”.

- “save”: evento similar a “create” e “update”, mas ocorre no ato de persistência independentemente se este nó é novo ou já existente. Seus call-backs são “before_node_save” e “after_node_save”.
- “destroy”: evento que ocorre no ato de exclusão de um nó já existente. Seus call-backs são “before_node_destroy” e “after_node_destroy.
- “add_link”: evento que ocorre quando um valor de elo é adicionado ao nó. Seus call-backs são “before_add_link” e “after_add_link”. O objeto da classe “NodeLink”, representante do valor do elo adicionado, fica disponível para a contexto de execução da operação que implementa o call-back através da variável “@node_link”.
- “remove_link”: evento que ocorre quando um valor de elo é removido do nó. Seus call-backs são “before_remove_link” e “after_remove_link”. O objeto da classe “NodeLink”, representante do valor do elo removido, fica disponível para o contexto de execução da operação que implementa o call-back através da variável “@node_link”.
- “navigation”: evento disparado pelo controlador de navegação e que ocorre no ato de navegação sobre o nó. Seus call-backs são “before_node_navigation” e “after_node_navigation”. Lembramos que o contexto navegacional corrente fica disponível no contexto de execução de toda operação de classe navegacional ativada durante a ação de navegação contextual, através de uma instância da classe “Context” atribuída a variável “@context”.

A seguir vemos um exemplo de utilização do call-back “after_node_navigation”:

```
op = NavOperation.new
op.name = “after_node_navigation”
op.code = %{
      v = Visita.new
      v.contexto = @context.name
      v.posicao = @context.position
      v.save
      self.visitado_em << v
      }
```

O call-back implementado acima cria um novo objeto de uma classe supostamente já definida chamada “Visita”, que teria como finalidade guardar a data e hora da visita em um nó (lembrando que todo nó possui os atributos nativos somente-leitura “created_at” e “updated_at” que são automaticamente preenchidos quando o nó é persistido), assim como o contexto de navegação da visita (o contexto navegacional corrente fica disponível às operações de classes navegacionais através da variável “@context”). Após a criação do objeto “Visita”, o mesmo é adicionado como valor do elo “VisitadoEm” ao objeto que implementou o call-back.

Os métodos de call-back formam uma base para uma eventual implementação de hipertextos adaptativos, nos quais diversas características, tais como valores dos atributos e estrutura de navegação, podem se alterar durante a navegação, em função de vários fatores.



## 4.5. Controladores

O HyperDE possui dois conjuntos de controladores, o controlador de navegação (representado no framework pela classe “NavigationController”), para execução das ações de navegação na aplicação desenvolvida e os controladores de retaguarda (representados no framework pelas classes derivadas de “CrudController”), para execução das ações de desenvolvimento da aplicação OODHM/SHDM.

Todos os controladores são implementados de forma a permitirem acesso via protocolo REST. Assim, o acesso a seus métodos pode ser feito remotamente, seja através de navegador Web ou qualquer outro aplicativo ou dispositivo que consiga realizar requisições e interpretar respostas feitas através do protocolo HTTP. Isso confere à camada de controle do HyperDE a qualidade de arquitetura orientada a serviços.

Para utilização remota de um controlador via protocolo REST, a aplicação cliente deve requisitar a execução de uma das ações implementadas pelo controlador, através do seguinte formato de URL:

```
http://{servidor_hyperde}/{controlador}/{ação}/[{id}]?[{params}]
```

O nome do controlador é o nome em minúsculas da classe que o implementa, retirando-se o sufixo “Controller”. O nome da ação é nome do método que o implementa na classe controladora. O “id” é passado quando a ação referencia um objeto já existente que deve ser recuperado da base de dados. Os parâmetros restantes para a ação, podem ser passados na própria URL, após “?”, caso seja uma requisição do tipo “GET”. Caso seja uma requisição do tipo “POST”, os parâmetros podem ser passados no corpo da requisição.

Por exemplo, para fazermos uma requisição de atualização de um contexto com “id” 433, devemos fazer uma requisição “POST” para o seguinte endereço:

```
POST http://servidor/contexto/update/433
Corpo: context[name]=”AlunoAlfa”&context[query]=”…”
```

Veremos a seguir os detalhes de cada ação implementada por cada controlador do framework HyperDE.



### 4.5.1. CrudControllers – controladores de retaguarda

Os controladores de retaguarda do ambiente fornecem as funcionalidades para desenvolvimento da aplicação OOHDM/SHDM. Através deles é possível criar e gerenciar as entidadades de contextos navegacionais, classes navegacionais, índices, landmarks, elos, nós e visões (templates).

Cada entidade possui seu próprio controlador e cada controlador é subclasse da classe abstrata “CrudController”. São eles:

- ContextController: implementa as ações para gerenciamento de contextos navegacionais.
- IndexController: implementa as ações para gerenciamento de índices e seus atributos.
- LandmarkController: implementa as ações para gerenciamento de landmarks.
- LinkController: implementa as ações para gerenciamento de elos.
- NavClassController: implementa as ações para gerenciamento de classes navegacionais, seu atributos, elos e operações.

- NodeController: implementa as ações para gerenciamento de nós e elos entre nós.
- ViewController: implementa as ações para gerenciamento de visões customizadas (templates) para classes em contexto e índices.

As ações implementadas por esses controladores que são herdadas de "CrudController" são:

- list: fornece a listagem das instâncias da entidade sendo gerenciada.
- index: o mesmo que a ação "list".
- new: fornece os dados para inicialização de uma instância da entidade para finalidade de criação da mesma (ação "create").
- edit: fornece os dados de uma instância existente de uma entidade com finalidade de atualização da mesma (ação "update").
- create: executa a ação de criação de uma nova instância de uma entidade.
- update: executa a ação de atualização de uma instância existente de uma entidade.
- delete: executa a ação de exclusão de uma instância existente de uma entidade.

Cada controlador de retaguarda implementa essas ações relativas à entidade que ele gerencia. Além disso, cada controlador pode implementar outras ações específicas. Por exemplo, "NavClassController" implementa além das ações básicas, ações relativas a atributos navegacionais, como "add_attribute", "remove_attribute", entre outras.

Os detalhes de cada ação implementada pelos controladores do framework, como os parâmetros que eles recebem e os dados fornecidos, podem ser vistos na seção de API do framework HyperDE.

Todas as ações que fornecem dados, como "list", "new", "edit" e outras, o fazem através de formato XML ou HTML através de templates customizáveis para cada visão relativa a essas ações.

O diagrama a seguir apresenta uma visão geral sobre os controladores de retaguarda e ações implementadas por cada um deles, assim como as classes a que eles estão relacionados:

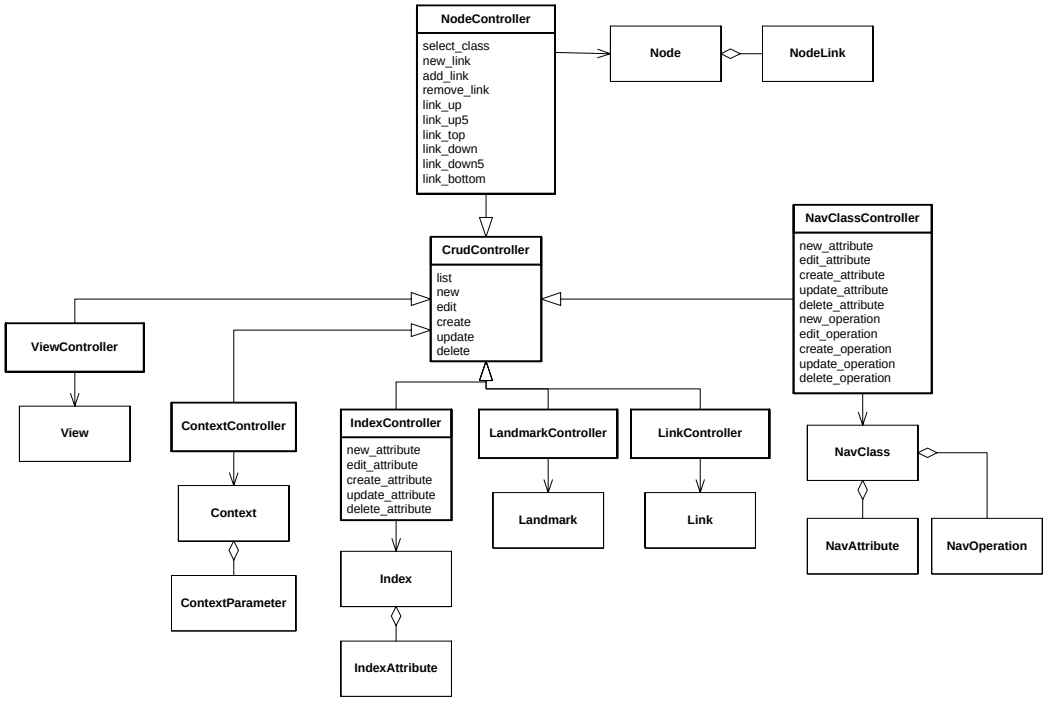


Figura 18 – Controladores de retaguarda e suas ações e as classes que eles manipulam

Veremos a seguir em maior detalhe, cada um dos controladores de retaguarda, subclasses de “CrudController”.

### 4.5.1.1. ContextController

O “ContextController” manipula os objetos das classes “Context” e “ContextParameter”. Este controlador não implementa nenhum método adicional além dos implementados em “CrudController”. As ações de “new” e “edit” fornecem uma lista das classes navegacionais que podem ser utilizadas como tipos para os parâmetros do contexto. A ação de “edit” fornece ainda os atributos do contexto. A ação “list” fornece a lista de todos os contextos navegacionais já cadastrados na aplicação.

Para maiores informações sobre cada ação e seus parâmetros, ver a documentação da API.

#### 4.5.1.2. IndexController

O “IndexController” manipula os objetos das classes “Index” e suas subclasses e também “IndexAttribute” e suas subclasses. As ações “new” e “edit” fornecem uma lista de classes navegacionais e contextos que podem ser usados para definição dos tipos de parâmetros do índice e para definição do contexto associado, caso o objeto manipulado seja instância de “ContextIndex”. A ação “list” fornece a lista de todos os índices já cadastrados na aplicação.

“IndexController” implementa as seguintes ações adicionais às ações implementadas por “CrudController”:

- new_attribute: fornece os dados para a criação de um novo atributo do índice, que são as listas de contextos e de índices para os tipo de atributos referentes a âncoras para contexto e índice, respectivamente.
- edit_attribute: fornece os mesmo dados que “new_attribute” além dos atributos do atributo de índice existente que será atualizado.
- create_attribute: executa a ação de criar um novo atributo de índice, de acordo com os parâmetros recebidos.
- update_attribute: executa a ação de atualizar um atributo de índice existente, de acordo com os parâmetros recebidos.
- delete_attribute: executa a ação de excluir um atributo de índice existente.

Para maiores informações sobre cada ação e seus parâmetros, ver a documentação da API.

#### 4.5.1.3. LandmarkController

O controlador “LandmarkController” é responsável por manipular os objetos instanciados a partir da classe “Landmark”. Este controlador não implementa nenhum método adicional além dos implementados por “CrudController”. Suas ações “new” e “edit” fornecem listas de contextos e índices disponíveis no sistema para associação como alvo da âncora resultante do

landmark. A ação “edit” fornece ainda os atributos do landmark a ser atualizado. A ação “list” fornece a lista de todos os landmarks já cadastrados na aplicação.

Para maiores informações sobre cada ação e seus parâmetros, ver a documentação da API.

#### 4.5.1.4. LinkController

O controlador “LinkController” é responsável por manipular os objetos instanciados a partir da metaclasse “Link”. Este controlador não implementa nenhum método adicional além dos implementados por “CrudController”. Suas ações “new” e “edit” fornecem a lista de classes navegacionais para serem associadas como origem e alvo do elo resultante, assim como a lista de elos, para associação opcional como elo inverso ao elo resultante. A ação “edit” fornece ainda os atributos do elo a ser atualizado. A ação “list” fornece a lista de todos os elos já cadastrados na aplicação.

As ações executadas por este controlador são monitoradas pela aplicação através de uma classe “ModelObserver” com o objetivo de atualizar o modelo navegacional da aplicação desenvolvida a cada modificação nas metaclasses manipuladas.

Para maiores informações sobre cada ação e seus parâmetros, ver a documentação da API.

#### 4.5.1.5. NavClassController

O controlador “NavClassController” é responsável por manipular os objetos instanciados a partir das metaclasses “NavClass”, “NavAttribute” e “NavOperation”. Suas ações “new” e “edit” fornecem uma lista de classes navegacionais para que possa ser feita atribuição de classe base (superclasse) da classe navegacional a ser criada ou atualizada. A ação “edit” fornece ainda os dados da classe navegacional a ser atualizada. A ação “list” fornece a lista de todas classes navegacionais já cadastradas na aplicação.

Este controlador implementa ainda as seguintes ações adicionais às ações implementadas por “CrudController”:

- new_attribute: fornece os dados para criação de um novo atributo da classe navegacional, que incluem a lista de tipos de dados disponíveis para atributos simples, a lista de contextos disponíveis e a lista de elos disponíveis para atributos do tipo âncora para contexto e a lista de índices disponíveis para atributos do tipo índice ou âncora para índice.
- edit_attribute: fornece os dados para atualização de um atributo existente, que são os mesmos que “new_attribute” além dos dados do atributo a ser atualizado.
- create_attribute: executa a ação de criar um novo atributo em uma classe navegacional, de acordo com os parâmetros recebidos.
- update_attribute: executa a ação de atualizar um atributo de classe navegacional existente, de acordo com os parâmetros recebidos.
- delete_attribute: executa a ação de excluir um atributo de classe navegacional existente.
- new_operation: fornece os dados para criação de uma nova operação em uma classe navegacional.
- edit_operation: fornece os dados para atualização de uma operação existente em uma classe navegacional, que são os mesmos que “new_operation” além dos dados da operação a ser atualizada.
- create_operation: executa a ação de criar uma nova operação em uma classe navegacional, de acordo com os parâmetros recebidos.
- update_operation: executa a ação de atualizar uma operação de classe navegacional existente, de acordo com os parâmetros recebidos.
- delete_operation: executa a ação de excluir uma operação de classe navegacional existente.

As ações executadas por este controlador são monitoradas pela aplicação através de uma classe “ModelObserver” com o objetivo de atualizar o modelo navegacional da aplicação desenvolvida a cada modificação nas metaclasses manipuladas.

Para maiores informações sobre cada ação e seus parâmetros, ver a documentação da API.

#### 4.5.1.6. NodeController

O controlador “NodeController” é responsável pela manipulação dos nós da aplicação. Os nós são instâncias das classes navegacionais dinamicamente geradas pelo HyperDE de acordo com as metaclasses criadas pelos controladores “NavClassController” e “LinkController”.

As ações “new” e “edit” fornecem os dados de um novo nó ou um já existente com finalidade de criação ou atualização do mesmo. A ação “list” oferece a listagem dos nós de uma classe navegacional, passada como parâmetro para a ação. Se o parâmetro que identifica a classe não for passado, “list” fornece apenas a listagem das classes navegacionais disponíveis na base de dados.

As seguintes ações adicionais são implementadas por “NodeController”, além das implementadas por “CrudController”:

- “select_class”: fornece a lista de classes navegacionais existentes para seleção visando uma posterior ação de criação de novo nó.
- “new_link”: fornece a lista de nós disponíveis para associação como valor de um determinado tipo de elo a um nó já existente. O tipo de elo e o nó a que será associado no novo elo devem ser passados como parâmetros para a ação (mais detalhes na documentação da API).
- “add_link”: executa a operação de inserção de um novo valor de elo a um nó já existente de acordo com os parâmetros recebidos (tipo de nó, nó de origem, nó de destino). O parâmetro de nó de destino também pode ser uma listas de nós, o que possibilita a associação de vários elos na execução de uma única ação.
- “remove_link”: executa a operação de remoção de um valor de elo já existente de acordo com os parâmetros recebidos, que pode ser o elo a ser removido ou uma lista de elos, para remoção em lote.
- “link_up”, “link_down”, “link_to_top”, “link_to_bottom”: ações de re-posicionamento de um valor de elo relativo a outros elos de

mesmo tipo dentro de um nó. O parâmetro necessário para essas ações é o valor do elo a ser re-posicionado e o reposicionamento pode ser: 1 posição para cima, 1 posição para baixo, primeira posição e última posição, respectivamente.

Para maiores detalhes sobre cada ação e seus parâmetros, ver a documentação da API do framework.

#### 4.5.1.7. ViewController

O controlador "ViewController" é responsável pela manipulação das visões customizadas que podem ser definidas pelo usuário para navegação em índices e contextos navegacionais. Estas visões são representadas por instâncias da classe "View", que será vista em maior detalhe em seção posterior. Este controlador implementa apenas as ações definidas por "CrudController".

Suas ações "new" e "edit" fornecem os dados para criação e atualização de uma visão nova ou já existente. A ação "edit" fornece ainda os atributos da visão a ser atualizada. A ação "list" fornece a lista de todas as visões customizadas já cadastradas na aplicação.

Para maiores informações sobre cada ação e seus parâmetros, ver a documentação da API.

### 4.5.2. NavigationController – controlador de execução da aplicação

O controlador de navegação, implementado pela classe "NavigationController", é responsável pelas ações que correspondem à execução da aplicação OOHDM/SHDM desenvolvida no ambiente HyperDE. As ações implementadas nesse controlador são:

- "index": ação que corresponde ao ponto inicial de execução da aplicação.
- "show_index": ação que fornece os dados para construção de uma estrutura de acesso, um índice, que são objetos derivados de subclasses da classe "Index".

- “context”: ação que fornece os dados para construção de um contexto navegacional e o nó corrente neste contexto.

O controlador de navegação também é responsável por manter o rastro de navegação do usuário na aplicação através de um objeto representado pela classe “Breadcrumb”. A cada mudança de contexto navegacional ou mudança de índice, o rastro é atualizado. Dessa forma, é possível oferecer ao usuário âncoras que lhe possibilitem retroceder a navegação para contextos e índices previamente acessados. Esse tipo de recurso é bastante recomendado quando a modelagem navegacional da aplicação inclui objetos que impõem um acesso hierárquico entre eles. O rastro de navegação é mantido através de “cookie” e mais detalhes podem ser obtidos sobre a classe “Breadcrumb” através da documentação da API do framework.

Veremos cada ação do controlador navegacional em detalhe a seguir.



#### 4.5.2.1. Ação “index” – ponto de entrada da aplicação

A ação “index” calcula o ponto de entrada da aplicação, que pode ser um índice ou um nó em contexto navegacional. Após calcular o ponto de entrada, esta ação simplesmente redireciona para a ação correspondente.

O algoritmo para cálculo do ponto de entrada leva em conta a definição de landmarks, índices e contextos da aplicação e opera da seguinte forma:

1. Recupera o 1º landmark da aplicação.
   1.1. O ponto de entrada da aplicação será o alvo da âncora correspondente ao landmark encontrado.
2. Caso não haja nenhum landmark definido na aplicação:
   2.1. Recupera o 1º índice da aplicação e o define como ponto de entrada.
   2.2. Caso não haja nenhum índice definido na aplicação:
      2.2.1. Recupera o 1º contexto da aplicação e define o primeiro nó como ponto d e entrada.
      2.2.2. Caso não haja nenhum contexto definido na aplicação, exibe mensagem de erro.

A URL de entrada da aplicação é:

```
http://[servidor]/navigation/
ou
http://[servidor]/navigation/index
```

#### 4.5.2.2. Ação “show_index” – estruturas de acesso

A ação “show_index” é responsável por recuperar um índice (identificado pela URL) cadastrado para a aplicação, computar as entradas do índice passando eventuais parâmetros para a execução do mesmo.

Esta ação também atualiza o rastro de navegação, caso o índice requisitado seja diferente do da última requisição ou a última requisição tenha sido para um contexto navegacional (ação “context”).

Por último, a ação “show_index” faz a seleção do template de índice a ser utilizado, verificando se existe visão associada ao índice requisitado. Caso não exista, é verificado se existe uma visão genérica de índice, que é utilizada no caso afirmativo. Em último caso é utilizada a visão padrão nativa do ambiente.

A URL para execução desta ação é:

```
http://[servidor]/navigation/show_index/[id][?params=...]
```

Caso o código identificador do índice (“id”) não seja passado, a ação irá utilizar o primeiro índice cadastrado na aplicação. Caso não existam índices cadastrados ou o código identificado seja inválido, será exibida mensagem de erro. Os parâmetros para o índice, caso o mesmo seja baseado em consulta parametrizada, devem ser passados na própria URL através do campo “params”.

Para maiores detalhes sobre esta ação, ver a documentação de API do ambiente.

#### 4.5.2.3. Ação “context” – navegação contextual

A ação “context” é a ação que possibilita o acesso a um nó da aplicação em determinado contexto navegacional. É a ação mais significativa e freqüente durante a execução de uma aplicação OOHDM/SHDM desenvolvida no ambiente HyperDE.

Uma requisição usual à ação “context” executa os seguintes passos:

- Recupera um contexto navegacional disponível na aplicação, identificado por código passado na URL.
    - Caso não tenha sido passado o código, recupera o primeiro contexto navegacional cadastrado na aplicação.
    - Caso não existam contextos cadastrados na aplicação, exibe mensagem de erro.
- Computa as entradas (nós) do contexto navegacional, passando eventuais parâmetros para o cálculo.
- Define o nó corrente, ou seja, a posição atual dentro do contexto, se o mesmo não estiver vazio.
- Atualiza o rastro de navegação, caso a requisição seja para um contexto diferente ao feito pela requisição anterior ou caso a requisição anterior tenha sido feita para a ação “show_index”.
- Seleciona o template a ser utilizado para desenho do nó corrente dentro do contexto navegacional recuperado.

A seleção do template da visão de contexto a ser utilizado é feito da seguinte forma:

- Verifica-se a existência de um template de visão associado ao contexto navegacional requisitado e também a classe navegacional do nó corrente. Se existir uma visão associada aos dois elementos, seu template é utilizado.
- Caso não haja uma visão associada ao contexto e à classe navegacional do nó corrente, verifica-se a existência de uma visão associada somente a classe navegacional do nó corrente. Se existir, seu template é utilizado.
- Caso não haja uma visão associada à classe navegacional do nó corrente, verifica-se a existência de uma visão associada somente ao contexto navegacional requisitado, especificamente. Se existir, seu template é utilizado.

- Caso não haja uma visão associada ao contexto navegacional requisitado, verifica-se a existência de uma visão genérica de contextos definida pelo usuário e seu template é utilizado em caso positivo.
- Em último caso, utiliza-se o template da visão nativa padrão do ambiente.

Em resumo, a visão selecionada segue a precedência de associação para contexto e classe, depois apenas para classe, depois apenas para o contexto específico, seguido de contexto genérico, acabando finalmente na visão padrão do ambiente.

Esse mecanismo de “fall-back” utilizado na seleção de visões, tanto para nós em contexto como para índices, possibilita ao desenvolvedor da aplicação, definir visões customizadas cada vez mais especializadas de forma incremental e gradativa. Dessa forma, é possível avaliar o funcionamento da aplicação, mesmo que a definição de visões customizadas esteja incompleta ou mesmo inexistente.

A ação “context” também é responsável por executar as operações definidas na classe navegacional do nó corrente, ativadas através de elemento de interface. A seqüência de execução de uma ou mais operações, assim como uma possível transição navegacional é definida no elemento ativador e executada pelo controlador navegacional. O controlador navegacional pode executar a seqüência de forma síncrona e seqüencial ou assíncrona e paralela, conforme for definido pelo elemento ativador. O controlador navegacional disponibiliza para cada operação executada o contexto navegacional corrente (instância da classe “Context”), que fica acessível através da variável “@context”.

A URL de requisição de um nó em contexto navegacional é feita no seguinte formato:

```
http://[servidor]/navigation/context/[id@pos][?params=...]
```

Caso a posição corrente não seja especificada na URL de requisição, é assumido o primeiro nó do contexto requisitado (equivalente à posição zero). Os valores dos parâmetros a serem passados para o contexto navegacional, caso o esmo esteja baseado em consulta parametrizada, são passados na própria URL através do campo “params”.

Para maiores detalhes sobre esta ação e a execução de operações, ver a documentação de API do ambiente.

## 4.6. Modelagem de Interface

A modelagem de interfaces suportada pelo HyperDE pode ser considerada uma solução de compromisso entre o desenvolvimento de interfaces em um framework MVC tradicional e a etapa de modelagem de interface abstrata (ADV - abstract data view) promovido pelo método OOHDM/SHDM [Moura, 2004].

O usuário do ambiente pode definir visões customizadas para cada combinação de classe em contexto e índice que estiver disponível no modelo navegacional da aplicação desenvolvida. Ele faz isso através de objetos instanciados a partir de subclasses da classe abstrata “View” do framework HyperDE. Existem subclasses de “View” específicas para a construção de visões genéricas reutilizáveis (componentes visuais) ou visões associadas à classe em contexto, e associadas a índice, cujas instâncias são utilizadas respectivamente pelas ações “context” e “show_index” do controlador navegacional. O diagrama a seguir ilustra esse modelo:

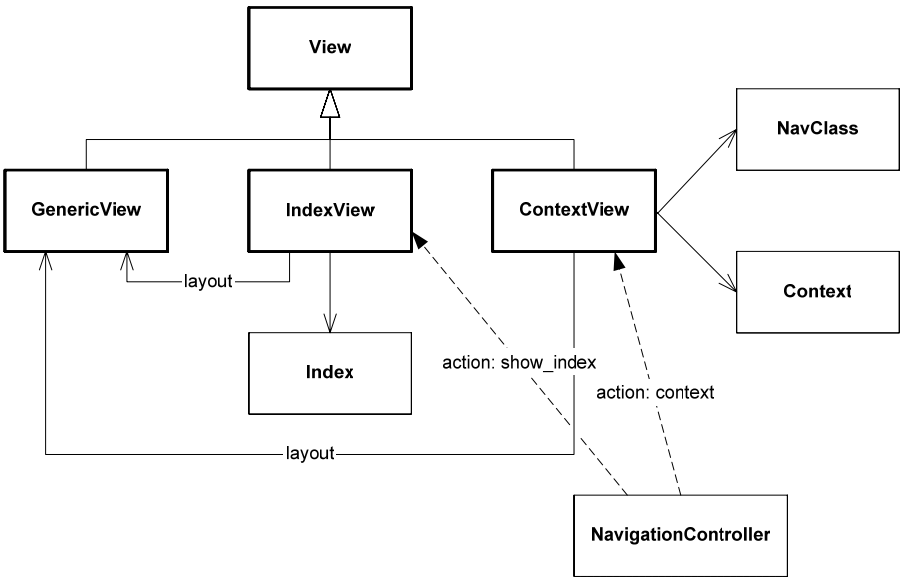


Figura 19 – Modelo de visões customizadas do HyperDE

Como podemos ver no diagrama, visões genéricas podem ser associadas como layouts de visões de contexto e índice. Veremos a utilização de visões genéricas em mais detalhe na próxima seção.

Os atributos da classe “View” e de suas subclasses são:

- name: nome da visão pelo qual a mesma pode ser referenciada pelas funções de auxílio de interface, como veremos adiante.
- template: definição concreta da interface, feita através da mistura de código-fonte HTML ou XML com chamadas a funções de auxílio de interface.

A sintaxe do HTML misto utilizada para o código-fonte do template é similar à utilizada em outras plataformas deste tipo como ActiveServerPages (ASP) e JavaServerPages (JSP) e utilizam os símbolos “<%”, “<%=” e “%>” para delimitar os trechos de código em Ruby.

A camada de interface do framework HyperDE oferece uma biblioteca de funções auxiliares para a construção de interfaces visuais baseadas nas primitivas do modelo navegacional, como contextos, índices, nós, atributos, operações, landmarks e outros. Essa biblioteca tem o objetivo de criar uma camada de abstração de modo a tentar minimizar a necessidade de uso de código HTML puro, assim como o conhecimento da linguagem de programação Ruby. Os detalhes de uso da biblioteca podem ser vistos em detalhe em subseção posterior e na documentação de API do framework. Através de um uso adequado dessa biblioteca torna-se possível ao desenvolvedor da aplicação, aproximar-se no modelo abstrato de interfaces definido pelo método OOHDM/SHDM.

Nas próximas seções, veremos o uso para cada tipo de visão customizada disponível no ambiente, seguida do uso da biblioteca de funções auxiliares.

### 4.6.1. Visões Genéricas (componentes)

As visões genéricas, representadas no framework HyperDE pela classe “GenericView”, têm o objetivo de oferecer um mecanismo de reuso de componentes de interface. Esta reutilização pode ser realizada nas visões

específicas para classe em contexto e índice, como também em outras visões genéricas.

Por exemplo, podemos utilizar uma visão genérica para definir uma folha de estilo CSS reutilizável ou mesmo um elemento gráfico como um botão ou uma imagem. A utilização destas visões se dá através da função auxiliar “component”, que veremos em mais detalhe a seguir e que recebe como argumento o nome da visão genérica a ser reutilizada.

A seguir vemos um exemplo de uso de uma visão genérica para construção de uma barra de navegação para a aplicação. Note que neste exemplo, utilizamos algumas funções auxiliares cuja forma de utilização deve ser desconsiderada neste momento.

```
<hr>
<div>
<span><%= context_previous %></span>
<span><%= node_label %></span>
<span><%= context_next %></span>
<div>
```

Supondo que demos a essa visão o nome de “BarraNavegacao”, podemos utilizá-la da seguinte forma:

```
<!-- ... código html de uma visão para classe em contexto ... -->

<%= component(“BarraNavegacao”) %>

<!-- restante do código html da visão de contexto ... -->
```

Uma outra utilização que pode ser dada a visões genéricas é na criação de layouts. Um layout é uma inversão da forma padrão de reutilização de componentes, onde para adicionar, por exemplo, um cabeçalho e um rodapé a todas as páginas da aplicação, deveríamos escrever algo do tipo:

```
<%= component(“Cabecalho”)
<!-- conteúdo da página -->
<%= component(“Rodape")
```

Neste padrão de reutilização, é necessário referenciar os componentes denominados “Cabecalho” e “Rodape" em todas as páginas em que eles são

aplicados. Uma mudança na estrutura desses dois elementos poderia potencialmente resultar na necessidade de se alterar todas as suas referências.

Um layout, por outro lado, define áreas constantes das páginas de uma aplicação e um ponto de inserção para o conteúdo dinâmico, de forma que o exemplo anterior ficaria:

```
<!-- conteúdo para o cabeçalho -->
@content_for_layout
<!-- conteúdo para o rodapé -->
```

Onde a variável "@content_for_layout" define o ponto de inserção do conteúdo dinâmico. O conteúdo da página a que o layout é associado precisaria ser apenas:

```
<!-- conteúdo da página -->
```

Em tempo de execução a página resultante portanto seria a combinação do layout com a página a qual o layout foi associado, resultando em:

```
<!-- conteúdo para o cabeçalho -->
<!-- conteúdo da página -->
<!-- conteúdo para o rodapé -->
```

Visões genéricas podem ser associadas a visões específicas de contexto e índice como layout da visão, conforme veremos a seguir.

### 4.6.2. Visões de Índice

Visões de índice, representadas pela classe "IndexView", podem ser utilizadas para a customização da apresentação visual de índices da aplicação. Uma visão de índice pode estar associada a um índice da aplicação, cujo template será então utilizado quando aquele índice for especificamente requisitado através de uma ação "show_index" do controlador navegacional.

Uma visão de índice pode também não estar associada a nenhum índice específico a qual chamamos de visão genérica de índice. O template desta visão será utilizado quando o índice requisitado na execução da aplicação não possuir uma visão específica associada.

Este comportamento caracteriza um mecanismo de *"fall-back"*, onde é possível definir uma visão genérica para todos os índices da aplicação e gradativamente ir especializando a apresentação de cada índice através de visões específicas.

Uma visão de índice também pode ser associada a um layout, definida através de uma visão genérica.

### 4.6.3. Visões de Classe em Contexto

Visões de classe em contexto, representadas pela classe "ContextView", podem ser utilizadas para a customização da apresentação de nós em contextos navegacionais durante a execução da aplicação. A associação de uma visão de contexto pode ocorrer em diferentes níveis, sob a seguinte ordem de precedência:

- Associada a um contexto navegacional e uma classe navegacional específica: o template desta visão será utilizado apenas quando ocorrer navegação do contexto navegacional associado e o nó corrente do contexto for derivado da classe navegacional associada.
- Associada a nenhum contexto em específico e a uma classe navegacional: o template desta visão será utilizado em qualquer contexto navegacional onde o nó corrente é derivado da classe navegacional associada.
- Associado a um contexto e a nenhuma classe navegacional em específico: o template desta visão será utilizando quando ocorrer navegação no contexto navegacional associado, independente da classe navegacional do nó corrente.
- Associado a nenhum contexto e nenhuma classe navegacional em específico: o template desta visão será utilizado quando ocorrer navegação em qualquer contexto navegacional e para qualquer classe navegacional do nó corrente.

Mais uma vez, a aplicação dos templates é feita pelo controlador navegacional na ordem de precedência, partindo da associação mais específica até a mais genérica, caracterizando também um mecanismo de *fall-back* para

navegação em contextos navegacionais, onde é possível desenvolver visões genéricas para classes e contextos e gradativamente ir especializando a apresentação das mesmas.

A visão de contexto navegacional também pode estar associada a um layout, definida através de uma visão genérica.

### 4.6.4. Biblioteca de funções auxiliares para interfaces

A biblioteca de camada de visão do framework HyperDE oferece funções que visam auxiliar o desenvolvedor a escrever um código-fonte para os templates em linguagem HTML ou XML de forma concisa. Suas funcionalidades cobrem o escopo necessário para a construção de visões de índices e de navegação em contextos e é composto pelas seguintes tarefas:

- Desenho do rastro de navegação (“breadcrumb”).
- Desenho de âncoras para os landmarks.
- Desenho de índices, suas entradas e atributos, incluindo o desenho de âncoras dos valores.
- Desenho de nós, seus atributos e âncoras para contexto e índice, como também para ativação de operações.
- Desenho de controles de navegação em contexto e informações sobre o contexto corrente.
- Desenho de componentes reutilizáveis definidos por visões genéricas.

Nas seções posteriores veremos alguns exemplos de usos de cada função; a forma de utilização detalhada de cada uma delas pode ser obtida na documentação de API do framework. Antes de iniciar no entanto, vejamos um exemplo simples, mas completo, de uso dessas funções para construção de um template para contexto navegacional:

```
<!-- folha de estilo css -->
<%= component "css" %>
<!-- landmarks -->
<div id=landmarks>
       <%= landmarks "<span class='landmark'>[ %s ]</span>" %>
</div>
<!-- breadcrumb -->
<div id=breadcrumb><%= breadcrumb "  :: %s" %></div>
<!-- contexto -->
<div id=context>
<% unless @context.empty? %>
       <!-- titulo do nó corrente -->
       <div class=node_title><%= node_label %></div>
       <!-- informações sobre o contexto atual -->
       <div class=context_info>
              [em <%= @context.name %> (<%= context_params %>) ]
       </div>
       <!-- barra de navegação do contexto -->
       <div class=context_navigation>
       <span class=context_previous><%= context_previous "<< %s" %></span>
       <span class=context_next><%= context_next "%s >>" %></span>
       <div>
       <!-- dados do nó corrente -->
       <div class=node>
              <div class=operations>
              <span class=operation>
              <%= node_operations "[%s]  " %>
              </span>
              </div>
              <div class=attributes>
              <%= node_attributes %>
              </div>
       </div>

       <!-- índice de navegação do contexto atual -->
       <div id=context_index>
       <%= context_index %>
       </div>
<% else %>
       <!-- caso o contexto esteja vazio -->
       <h1>Este contexto está vazio</h1>
<% end %>
</div>
```

O código-fonte acima em conjunto com definições de formatação de uma folha de estilo CSS tem como resultado:

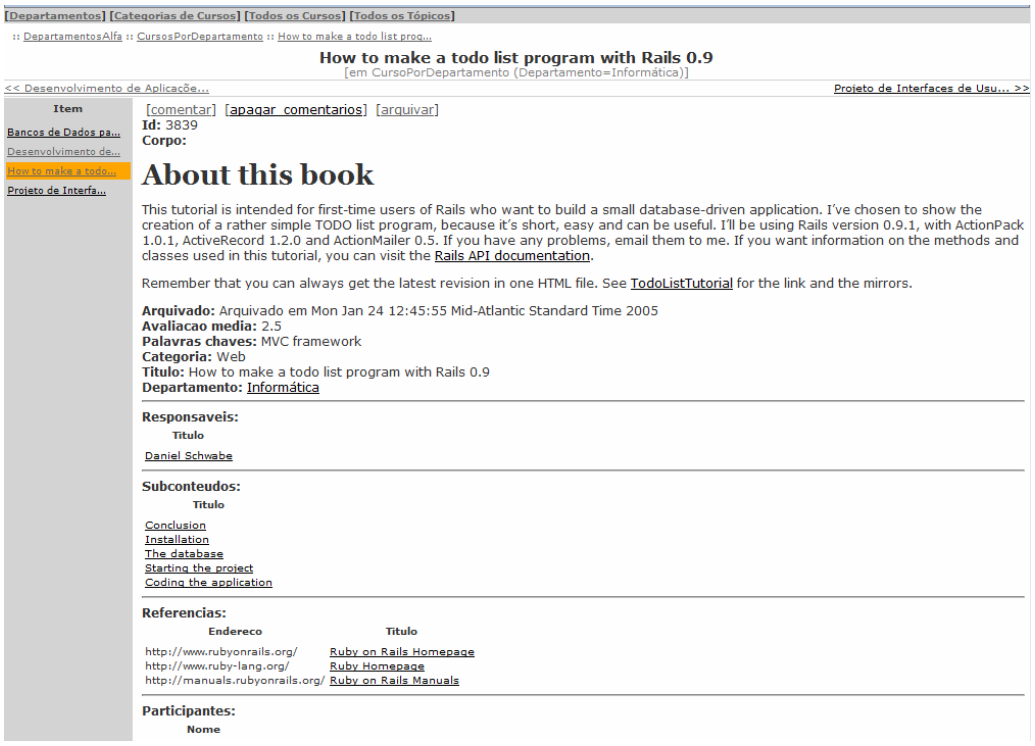


Figura 20 – Resultado de um template utilizando as funções auxiliares de interface

O template apresentado como exemplo assim como o resultado obtido é justamente o que compõe o código-fonte do template genérico nativo do ambiente. Note que por tratar-se de um template genérico, a formatação e posicionamento de atributos e índices do nó corrente são feitos de forma padronizada. Em uma aplicação mais elaborada, este template seria substituído por um especificamente talhado para a classe navegacional do nó a ser exibido.

De qualquer forma, é importante ressaltar que o código-fonte do template, por utilizar-se de uma folha de estilo para formatação de seus elementos e as funções auxiliares para desenho das primitivas do ambiente, aproxima-se bastante da definição abstrata da interface.

Em seguida veremos um pouco mais sobre cada uma das funções auxiliares para interface disponíveis no HyperDE.

#### 4.6.4.1. Função “breadcrumb”

A função “breadcrumb” auxilia no desenho do rastro de navegação da sessão corrente do usuário da aplicação OOHDM. O controlador navegacional mantém este rastro através de um “cookie” e cada requisição o reconstrói e o

atualiza. O estado do rastro de navegação é mantido através de uma instância do objeto “Breadcrumb”. Existem 3 formas básicas de uso da função breadcrumb:

- breadcrumb: sem argumentos, utiliza a formatação nativa do ambiente, que gera um rastro de navegação horizontal, separados por “::”, tendo como etiqueta das âncoras o título da página navegada.
- breadcrumb(template): neste modo, o usuário fornece um template que será utilizado no desenho de cada elemento do rastro de navegação. O parâmetro template pode ser do tipo String, quando a função se encarregará de gerar a âncora completa para cada elemento, ou do tipo Array, onde cada elemento do rastro pode ser decomposto em âncora, etiqueta da âncora e URL da âncora. Nesta forma, o primeiro elemento do Array passado é o template e os restantes são símbolos que identificam o componente a ser utilizado no template. O posicionamento da âncora é definido através de caracteres de formatação de strings no estilo “%s”. Veremos o uso dessa forma nos exemplos a seguir.
- breadcrumb { block }: neste modo, o usuário fornece um bloco de código para o desenho de cada elemento do rastro de navegação, podendo misturar código HTML e Ruby. Veremos o uso dessa forma nos exemplos a seguir.

Os exemplos de utilização da função “breadcrumb” abaixo produzem resultados similares, no entanto sua forma difere e pode ser utilizada conforme preferência ou necessidade do desenvolvedor.

```
<!-- a forma mais compacta -->
<%= breadcrumb %>

<!-- forma usando template simples
     nesta forma, a função gera a ancora completa -->
<%= breadcrumb “:: %s“ %>

<!-- forma usando template decomposto
     nesta forma, a ancora pode ser decomposta -->
<%= breadcrumb [ “:: <a href=’%s’>%s</a>”, :url, :label ]%>
```

```
<!-- forma utilizando template na forma de bloco -->
<% breadcrumb { |e| %>
      ::  <a href=”<%= e.url%>”><%= e.label %></a>
<% } %>
```

Note que nas formas de utilização funcionais (quando não é utilizado bloco de código intercalado), deve ser utilizado o delimitador “<%=” pois a função efetivamente retornará o código HTML gerado completo. Na forma de bloco, é usado “<%” pois o resultado da função virá do bloco fornecido a mesma.

Na 3ª forma do exemplo, vimos também como cada elemento de âncora do rastro de navegação pode ser decomposto em etiqueta (símbolo “:label”) e a URL alvo da âncora (símbolo “:url”). Esta é uma forma bastante comum de decomposição de várias outras funções da biblioteca, como veremos a seguir.

#### 4.6.4.2. Função “landmarks”

A função “landmarks” auxilia na construção do código HTML das âncoras referentes aos landmarks cadastrados no ambiente. Cada landmark é representado por instâncias da classe “Landmark”. A forma de utilização desta função é similar ao da função “breadcrumb” e pode ocorrer em 3 formas:

- landmarks: sem argumentos. Utiliza uma formatação nativa do ambiente para a geração de âncoras horizontais, separadas por “[“ e “]”.
- landmarks(template): nesta forma, deve-se fornecer um template em forma de String (forma compacta) ou Array (forma decomposta), conforme visto na função “breadcrumb”, indicando a posição de inserção das âncoras através do código de formatação “%s”.
- landmarks { block }: nesta forma o template é fornecido através de um bloco de código intercalado com HTML que será chamada para cada elemento da coleção de landmarks do da aplicação.

A seguir vemos exemplos das formas de utilização da função “landmarks”, produzindo resultados similares:

```
<!-- a forma mais compacta -->
<%= landmarks %>

<!-- forma usando template simples
     nesta forma, a função gera a ancora completa -->
<%= landmarks “ [ %s ] “ %>

<!-- forma usando template decomposto
     nesta forma, a ancora pode ser decomposta -->
<%= landmarks [ “ [ <a href=’%s’>%s</a> ] ”, :url, :label ]%>

<!-- forma utilizando template na forma de bloco -->
<% landmarks { |lm| %>
       [ <a href=”<%= lm.url %>”><%= lm.label %></a> ] 
<% } %>
```

Note que no exemplo de forma de uso de bloco, o que é passado para o bloco em cada iteração através da variável “lm” é uma instância de uma classe Landmark decorado com os métodos “label” e “url” para facilitar seu uso dentro do bloco. A classe Landmark não possui esses métodos, mas a função “landmarks” os insere dinamicamente ao executar cada iteração. Isso não anula os outros métodos e atributos da classe Landmark como “name” e “position” que poderiam ser acessados e usados normalmente para construção do template, como nos exemplos a seguir:

```
<% landmarks { |lm| %>
      <li><%= lm.position %>. <%= lm.ahref %></li>
<% } %>
```

Neste último exemplo podemos ver ainda o uso de outro método adicionado dinamicamente pela função “landmarks” denominado “ahref”, que gera o código HTML de uma âncora completa, utilizando os outros dois métodos já apresentados “label” e “url”. Estes 3 métodos (“ahref”, “label” e “url”) estão disponíveis em todas as funções auxiliares de interface que iteram sobre objetos que representam a definição de uma âncora.

#### 4.6.4.3. Funções “context_previous” e “context_next”

As funções “context_previous” e “context_next” auxiliam na geração de âncoras para os nós vizinhos do nó corrente dentro de um contexto navegacional. Suas formas de utilização podem ser feitas de 3 maneiras: sem argumentos, com

template compacto ou decomposto como argumento ou com template através de bloco. Essas formas são análogas às formas utilizadas pelas funções “breadcrumb” e “landmark” já vistas.

Vejamos alguns exemplos de uso dessas funções para criar uma barra de navegação contextual:

```
#uso de template simples como parametro
<table><tr>
    <td><%= context_previous “<< %s” %></td>
    <!-- titulo do nó corrente -->
    <td><%= node_attribute @context.current, “label” %></td>
    <td><%= context_next “>> %s” %></td></tr></table>

#uso com template em bloco
<% context_previous { |anc| %>
    <a href=”<%= anc.url %>”><img src=prev.gif border=0></a>
<% } %>

#uso com template decomposto
<%= context_next [“<a href=’%s’><img src=nxt.gif alt=’%s’></a>”, :url, :label ]%>
```

Repare que na forma de utilização em bloco, o objeto passado para o bloco a cada iteração através da variável “anc” é uma instância da classe “ContextAnchor” que possui os atributos “url”, “label” e “ahref” já mencionados.

#### 4.6.4.4. Função “context_params”

A função “context_params” fornece o auxílio para exibição e formatação dos valores dos parâmetros passados para o contexto corrente. Mais uma vez, sua forma de utilização pode ocorrer de 3 maneiras: sem argumentos, templates compactos e decompostos e template em bloco. Vejamos alguns exemplos:

```
<!-- nome do contexto atual seguido de seus parametros -->
<%= @context.name %><br>

<!-- sem argumentos -->
 (<%= context_params %>)

<!-- ou com template compacto -->
<%= context_params “(%s)” %>

<!-- ou com template decomposto -->
<%= context_params [ “(%s: %s)”, :param, :value ] %>

<!-- ou com template em bloco -->
```

```
<%= context_params { |param, value| %>
(<%= param %>: <%= value %>) <% } %>
```

Podemos notar que no caso desta função, os valores fornecidos a cada iteração são objetos da classe String, passados através de um par de variáveis: a primeira contendo o nome do parâmetro, a segunda contendo seu valor.

#### 4.6.4.5. Função “context_index”

A função “context_index” auxilia na construção do índice referente ao contexto corrente. Sua utilização também ocorre nas 3 formas já apresentadas: sem argumentos, quando é utilizada formatação nativa do ambiente, com template compacto ou decomposto e com template em bloco de código. O template que pode ser passado para a função é referente a cada entrada do índice, que é sempre composto de apenas um atributo contendo âncoras referentes aos nós do contexto corrente.

Vejamos alguns exemplos de utilização:

```
<!-- sem argumentos -->
<%= context_index %>

<!-- argumento compacto -->
<%= context_index “<tr><td class=index_entry>%s</td></tr>” %>

<!-- argumento decomposto -->
<%= context_index [ “<li><a href=’%s’>[ %s ]</a>”, :url, :label ]

<!-- template em bloco -->
<%= context_index { |anc| %>
<tr>
    <% if anc.current? %>
        <td class=”index_entry_current”><b><%= anc.label %></b></td>
    <% else %>
        <td class=”index_entry”><%= anc.ahref %></td>
    <% end %>
</tr>
<% } %>
```

Podemos notar na forma de utilização com template em bloco, que o iterador desta função passa como parâmetro do bloco (através da variável “anc”) um objeto da classe “ContextAnchor” estendido de alguns métodos, como “url”, “label” e “ahref”, já vistos anteriormente, e um novo método “current?”, particular desta função. O método “current?” indica se o elemento iterado corresponde ao nó

corrente e pode ser usado para definir uma formatação diferenciada para este elemento, como por exemplo, um efeito de seleção.

#### 4.6.4.6. Funções “node_attr” e “node_attributes”

A função “node_attr” auxilia a geração de código HTML para a exibição do valor de atributos do nó corrente. Suas formas de utilização são similares as das funções já apresentadas, no entanto, existe um primeiro argumento obrigatório, que é o nome do atributo a ser exibido.

```
<!-- atributo do tipo Image -->
<label>Foto:</label><%= node_attr “foto” %>

<!-- atributo do tipo Url -->
<label>HomePage:</label><%= node_attr “homepage” %>

<!-- atributo do tipo Email -->
<label>Email:</label><%= node_attr “email” %>

<!-- atributo do tipo âncora -->
<label>Orientador:</label>
<%= node_attr “orientador” %>

<!-- ou -->
<%= node_attr “orientador”, “Ir para: %s “ %>

<!-- ou -->
<%= node_attr “orientador”, [ “Ir para: <a href=’%s’>%s</a>”, :url, :label ] %>

<!-- ou -->
<% node_attr “orientador” { |anc| %>
    Ir para: <a href=”<%= anc.url %>”> <%= anc.label %></a>
<% } %>

<!-- atributo do tipo índice -->
<label>Publicações</label><br>
<%= node_attr(“publicacoes”) %>

<!-- ou -->
<% node_attr(“publicacoes") { |idx_pubs| index(idx_pubs) { %>
<!-- template do índice: veja função auxiliar “index” -->
<% } %>
```

Nos 3 primeiros exemplos, vemos a utilização simples, sem argumentos. Nesta forma, a função “node_attr” examina o tipo de dado do atributo e formata seu valor de acordo com os templates nativos do sistema. Existe um template específico associado a cada tipo de dado suportado pelo framework, o que permite ao ambiente fornecer o código HTML adequado para cada tipo de atributo.

Para os atributos de tipo âncora, apresentados no exemplo, vê-se as outras três formas de utilização, utilizando template compacto e decomposto passado por parâmetro e utilizando template com bloco de código. Repare que nesta última forma, o parâmetro passado ao bloco é o valor do atributo, neste caso, na forma de objeto da classe “ContextAnchor” ou “IndexAnchor”.

No último exemplo vemos o caso mais complexo, pois se trata de um atributo de tipo índice. As formas de utilização para este tipo de atributo são apenas duas: a forma sem argumentos, quando é utilizando o template nativo do ambiente para gerar todo o código HTML do índice, ou a forma de template em bloco, utilizando uma segunda função auxiliar encadeada chamada “index”. Esta função é utilizada para definição de um template customizado para o índice (onde se define cabeçalho, rodapé e formatação dos atributos) e para sua geração utilizando este template. Veremos sua forma de utilização em seção posterior da dissertação.

A função “node_attributes” gera a saída para exibição em HTML das etiquetas e valores de todos os atributos do nó corrente, utilizando a função “node_attr” com o template nativo do ambiente. Esta função é útil no desenvolvimento de um template customizado, pois rapidamente podemos obter uma visão sobre todos os atributos de um nó.

A seguir temos um exemplo do resultado obtido com essas funções:


**Figura 21 - Exemplo de usos da função “node_attr”**

Na imagem acima podemos ver a utilização de várias funções auxiliares já descritas como “landmarks”, “breadcrumb”, “context_index”, “context_previous”, “context_next”, “context_params” e “node_attributes”.

Podemos notar que a função “node_attributes” gerou o HTML adequado para exibição de todos os atributos do nó (tanto as etiquetas como seus valores), de acordo com seu tipo de dado. No exemplo vemos atributos do tipo “Image” (foto), “Url” (homepage), “Email” (email), “Anchor” (departamento, conteudos_criados e comentarios) e “Index” (cursos).

É interessante ressaltar que os templates nativos utilizam apenas folhas de estilo CSS para sua formatação e posicionamento (layout), portanto, torna-se simples customizar essa formatação apenas redefinindo os estilos utilizados pelos elementos das interfaces geradas. Os nomes das classes CSS utilizadas para os elementos gerados pelas funções auxiliadores estão descritos na documentação técnica em formato eletrônico do ambiente (ver apêndice D).

### 4.6.4.7. Funções “node_op” e “node_operations”

A função “node_op” pode ser usada para auxiliar na geração de âncoras para ativação de operações executadas pelo nó corrente. Sua forma de utilização é apenas uma:

- node_op(template_etiqueta, *operacoes): o argumento “template_etiqueta” deve receber o template referente à etiqueta da âncora que será gerada para ativação de uma ou mais operações. Caso o template da etiqueta contenha apenas um texto simples (sem código de formatação do tipo “%s”), a âncora terá este texto como etiqueta. Caso o template da etiqueta contenha um código de formatação ou caso seja de tipo decomposto (um Array), a âncora será o resultado do template. O argumento “*operacoes" é um argumento multivalorado que deve ser composto de um ou mais objetos da classe “Hash”, contendo pares de chave/valor que descrevem a operação a ser executada, seus parâmetros e forma de execução (síncrona ou assíncrona).

Vejamos alguns exemplos de utilização desta função:

```
<!-- ativação de apenas uma operação simples -->
<%= node_op “Arquivar”, { :op => “arquivar” } %>

<!-- ativação de operação usando template decomposto para etiqueta -->
<%= node_op [ “<a href=\”window.open(‘%s’)\”><img src=’arq.gif’></a>”, :url ],
            { :op => “arquivar” } %>

<!-- 2 operações seguidas de uma navegação para edição do nó -->
<%= node_op "Limpar e Comentar",
   { :op => "limpar_comentarios", :params => "true", :async => true },
   { :op => "comentar", :params => "'comentário para #{@node.label}'"},
   { :op => :navigate, :controller => "node", :action => "edit", :id => @node.id }
%>
```

As opções para o objeto “Hash” de definição de ativação das operações apresentadas nos exemplos acima são:

- :op - identifica o nome da operação do nó corrente a ser executada.

- :params - define os valores dos parâmetros, através de um objeto “String”, que serão passados como argumentos da operação a ser executada.
- :async - define se a operação será executada de forma síncrona ou assíncrona.

No último exemplo, podemos ver a definição de uma operação especial de navegação através de “:op => :navigate”. Esta operação especial está sempre disponível, independente da classe navegacional do nó corrente, e define que será executada uma navegação após a execução das operações. Suas opções são:

- :controller - identifica um controlador externo, caso seja uma navegação para fora do controlador de navegação. Se esta opção for omitida, assume-se que a navegação será dentro do controlador de navegação (“navigation”).
- :action - identifica a ação do controlador a ser executada. Se esta opção for omitida, assume-se que a navegação será para a mesma ação corrrente (“context”).
- :id - identifica o código de identificação para ação a ser executada. Se esta opção for omitida, assume-se que a navegação utilizará o mesmo código identificador corrente.

Caso esta operação não seja declarada, ocorrerá apenas a execução das operações definidas, sem que haja uma transição navegacional.

Neste último exemplo, também vimos que a navegação que ocorrerá após a ativação das operações tem como alvo um controlador de retaguarda do ambiente (“NodeController”). O controlador navegacional do HyperDE possibilita estas transições, no entanto, a posição atual de navegação contextual é armazenada para que o usuário da aplicação, ao retornar para o controlador de navegação, retome a navegação do ponto onde ocorreu a transição.

Todas as opções disponíveis para a função “node_op” na ativação de operações estão detalhadas na documentação da API do ambiente.

A função “node_operations” gera âncoras de ativação para todas as operações definidas na classe navegacional do nó corrente, utilizando o template nativo do ambiente e ativando apenas uma operação por vez, em modo síncrono. Esta função é útil na rápida elaboração de um template temporário que será refinado posteriormente.

#### 4.6.4.8. Funções “index” e “indexes”

A função “index” auxilia a construção de código HTML para exibição de índices. Sua forma de uso é diferente das outras funções auxiliares já vistas, pois ela depende da definição de sub-templates para as diferentes partes de um índice para sua correta exibição. Existem duas formas de utilização:

- index(index_obj) : nesta forma, o índice, definido através do argumento “index_obj”, que deve ser uma instância de uma subclasse de “Index”, é construído utilizando o template nativo do ambiente.
- index(index_obj) { |template| ... } : nesta forma de utilização, deve ser utilizando um bloco de código, que recebe como parâmetro um objeto da classe “IndexTemplate”, passado através da variável “template”, onde se definirá os sub-templates para cada área do índice, que são:
    - “header”: sub-template opcional de cabeçalho do índice, que precede a seção de entradas do índice.
    - “footer”: sub-template opcional de rodapé do índice, que sucede a seção de entradas do índice.
    - “entry”: sub-template obrigatório que será utilizando em iteração de cada entrada do índice.

Além dos sub-templates descritos, é necessário definir sub-templates para a exibição dos valores referentes a cada atributo do índice. Para melhor entendimento, veremos um exemplo a seguir:

```
<% index(@index) { |template| %>

   <!-- definicao do template para cabecalho -->

   <% template.header { %>
      <table><tr><th>Nome</th>
                <th>Cursos</th>
                <th>Professores</th>
                <th>Alunos</th></tr>
   <% } %>


   <!-- definicao do template para rodape -->
   <% template.footer { %>
      <tr><td colspan=4>
      <%= @index.entries.size %> entry(s).
      </td></tr></table>
   <% } %>


   <!-- definicao do template para cada entrada do indice -->
   <% template.entry { |entry| %>
     <tr><td>#nome#</td>
         <td>#cursos#</td>
         <td>#professores#</td>
         <td>#alunos#</td></tr>
   <% } %>



   <!-- definição do template para cada atriburo do índice -->
   <% template.attr "nome" %>
   <% template.attr "cursos", "Ver Cursos: %s" %>
   <% template.attr "professores",
      [ "<a href=\"w=window.open('%s');\">%s</a>", :url, :label ] %>
   <% template.attr "alunos", "Ver Alunos: %s" %>

<% } %>
```

A classe “IndexTemplate” possui 4 métodos, “header”, “footer”, “entry” e “attr” para definição dos sub-templates para as seções de cabeçalho, rodapé, entradas do índice e atributos do índice, respectivamente.

Os templates para os métodos “header”, “footer”e “entry” pode ser passado através de um objeto String passado como argumento da função ou como bloco de código, conforme os exemplos.

Note que o template para o método “entry” possui marcadores na forma “#nome_do_atributo#”, que definem a posição de entrada dos valores de cada atributo do índice, para cada entrada.

O método “attr” recebe como primeiro parâmetro o nome do atributo do índice a que ele se refere. O template para cada atributo pode ser passado através de objeto do tipo String utilizando marcação através do código de formatação “%s” ou Array, na sua forma decomposta. Se o template não for passado para este método, o ambiente utilizará o template nativo, que pode ser uma âncora ou texto simples, dependendo do tipo do atributo.

Abaixo temos o resultado de uma função “index” utilizando o template nativo do ambiente:

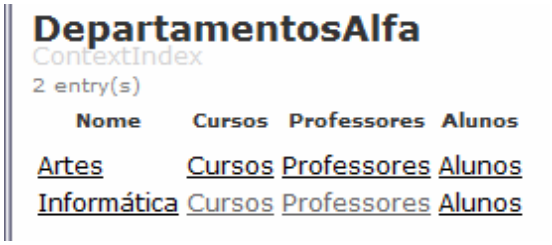


Figura 22 - Resultado de uso da função “index”

A função “index” pode ser utilizada em templates de visão de índice, para construção do índice requisitado, ou em templates de visão de classe em contexto, para construção dos atributos do nó corrente de tipo “Index”.

Ainda para este último tipo de visão, o framework fornece também a função “indexes”, que é usada sem argumentos e produz o código HTML utilizando template nativo do ambiente, para todos os atributos de tipo “Index” do nó corrente.

Vejamos abaixo essas formas de utilização:

```
<!-- em visão de índice para exibição do índice corrente -->
<%= index @index %>

<!-- em visão de contexto para exibição de um atributo de tipo “Index” -->
<%= index @node.publicacoes %>

<!-- ou -->
<%= node_attr(“publicacoes") { |idx_pubs| index(idx_pubs) } %>

<!-- exibindo todos os índices do nó corrente -->
<%= indexes %>
```

### 4.6.4.9. Função “component”

A função auxiliar “component” tem o objetivo de executar templates externos, definidos em visões genéricas, fornecendo assim um mecanismo de reuso e componentização de templates. Sua forma de uso é:

- componente(visao, parametros): o argumento “visao" é obrigatório e deve fornecer o nome da visão genérica a ser invocada pela função. O argumento “variaveis" é opcional, do tipo Hash, que pode conter pares de chave/valor indicando valores de parâmetros que devem ser passados para o template da visão sendo executada pela função. Neste caso, a chave, deve ser um símbolo correspondente ao nome

da variável que conterá o valor do parâmetro e que será passada ao template.

Vejamos alguns exemplos de utilização:

```
<!-- chamando um template sem parâmetros -->
<%= component “Logotipo” %>

<!-- chamando um template de índice e passando o índice como parâmetro -->
<%= component “RedIndex”, :index => @node.publicacoes %>

<!-- chamando um template de contexto de índice e definindo parametros -->
<%= component “SmallContextIndex”, :position => “right”, :width => “50px” %>
```

O template da visão genérica “SmallContextIndex” do último caso apresentado no exemplo poderia ser:

```
<div style=”float:<%= position %>;width:<%= width %>”><%= context_index %></div>
```

Neste template, percebe-se que as variáveis “position” e “width” devem ser passadas como parâmetros, conforme feito no exemplo anterior.

Todas as funções auxiliares apresentadas neste capítulo podem ser vistas em detalhes na documentação de API do framework.

## 4.7. Interfaces e Ferramentas do Ambiente

O ambiente de desenvolvimento HyperDE é composto pelo framework MVC, apresentado nas seções anteriores e através do qual pode-se construir um modelo navegacional de forma programática, e por ferramentas e interfaces com o usuário que têm como objetivo possibilitar a prototipação rápida de uma aplicação OOHDM/SHDM.

Atualmente, o ambiente fornece duas interfaces de acesso, a principal delas sendo em plataforma Web, acessível por navegador, e a segunda, por console acessível por terminal de linha de comando. Além disso, o ambiente fornece uma ferramenta para importação e exportação dos modelos navegacionais suportados no ambiente, de forma a facilitar cargas de dados em lote. Veremos cada uma dessas possibilidades a seguir.

### 4.7.1. Interface Web

A interface Web é a forma mais interativa oferecida pelo ambiente HyperDE de desenvolver uma aplicação OOHDM/SHDM. Através dela é possível alimentar a base de dados da aplicação com todas as primitivas necessárias para criação de um modelo navegacional descrito pelo método. Ainda, através da interface Web é possível executar a aplicação ao mesmo tempo em que ocorre sua construção, podendo-se testar e validar cada passo do desenvolvimento, acelerando o ciclo deste processo.

O uso da interface Web possui também a qualidade de poder ser utilizada remotamente, pois para acessá-la basta um navegador Web padrão. A URL desta interface é:

```
http://[servidor]/[crud_controller]/
```

Onde “crud_controller” é o endereço de qualquer controlador de retaguarda disponível no ambiente: “context”, “index”, “nav_class”, “link”, “landmark”, “view” e “node”, correspondente às primitivas de contextos navegacionais, índices, classes navegacionais, elos, landmarks, visões e nós, respectivamente.

As páginas que compõem a interface Web do ambiente seguem um padrão consistente, oferecendo sempre uma listagem das instâncias de cada primitiva descrita, possibilitando a adição de novas instâncias e alteração ou exclusão de instâncias existentes. As páginas de edição das instâncias oferecem diferentes recursos, dependendo da complexidade da primitiva sendo editada, mas são constantes as funções de persistir as alterações, através de botão “Save” ou cancelar, voltando pra listagem correspondente, através da âncora “<< Back”.

A seguir veremos uma descrição das tarefas que podem ser executadas sobre cada primitiva do ambiente por meio desta interface. Para acesso a cada uma delas, devemos utilizar as âncoras correspondentes a cada primitiva localizadas na barra superior da interface.

#### 4.7.1.1. Contextos

O gerenciamento de contextos navegacionais pela interface Web do ambiente se inicia pela listagem em ordem alfabética dos contextos já cadastrados para a aplicação, caso existam, onde podemos editá-los ou excluí-los, clicando na

âncora correspondente. Também é possível adicionarmos um novo contexto utilizando a âncora “Add New Context” localizada ao final da lista.

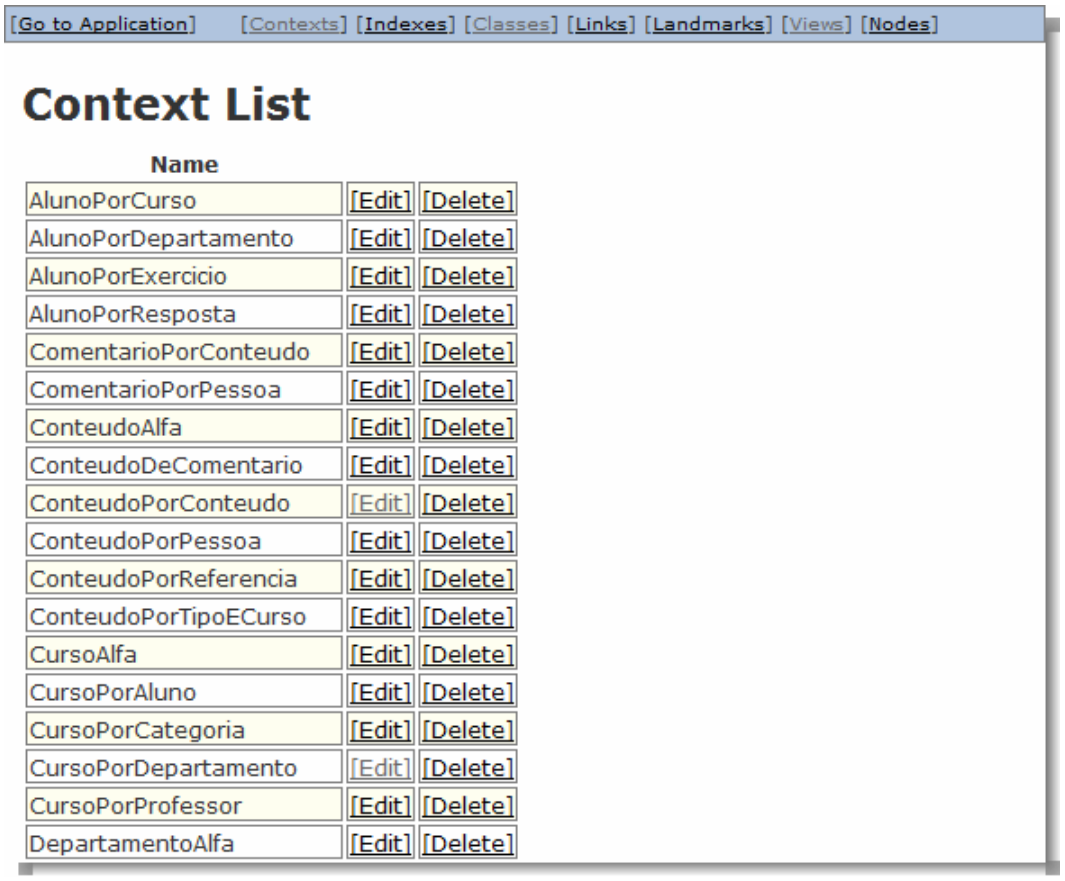


Figura 23 - Tela de listagem de contextos navegacionais

Através da tela de edição de contexto, devemos informar o nome do contexto e a expressão de consulta cujo resultado deve ser composto dos nós que irão participar do contexto.

Se a expressão de consulta contiver parâmetros cujos valores serão informados em tempo de execução do contexto, estes devem ser declarados, indicando o nome do parâmetro e seu tipo, caso o tipo seja um nó derivado de uma classe navegacional. Se o tipo do parâmetro for um valor literal, deve-se informar que o tipo é “Literal”.

Como exemplos, podemos citar um contexto “CursoPorDepartamento”, onde o valor do parâmetro é um nó do tipo “Departamento”. Já no contexto “CursoPorCategoria”, o parâmetro é um literal, pois “Categoria” não é representado por um nó, mas apenas como uma String.

Um contexto navegacional pode conter zero ou mais parâmetros, sem limite superior de parâmetros. Na prática, no entanto, este número tende a variar entre

zero a três parâmetros e por isso a interface provê sempre espaço para a declaração de três parâmetros “por vez”. Isso não significa que podemos apenas declarar três parâmetros, mas se preenchermos todos os espaços disponíveis para declaração dos parâmetros, o ambiente irá fornecer mais três espaços na próxima interação, após apertarmos o botão “Save”.

Para excluir um ou mais parâmetros já declarados, devemos marcá-los na coluna “Remove” e clicar em “Save” para efetuar a exclusão.

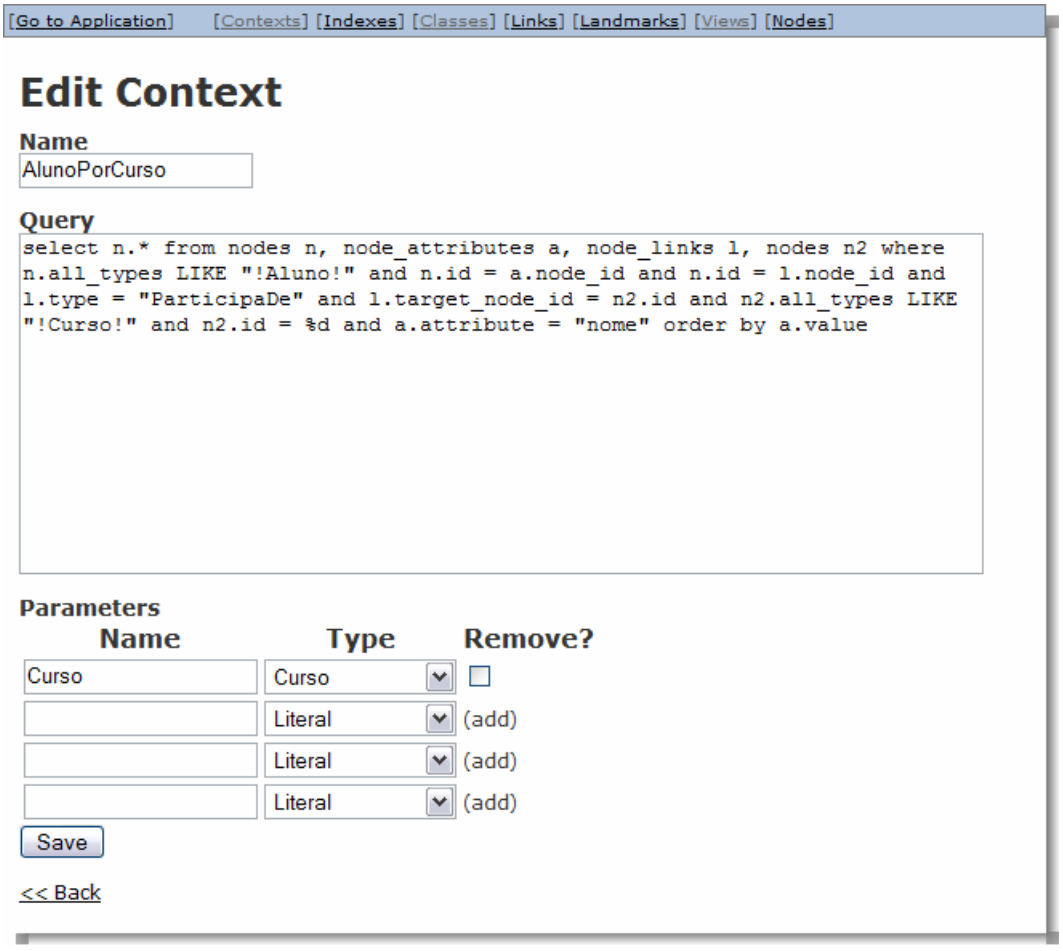


Figura 24 - Tela para edição de um contexto navegacional

### 4.7.1.2. Índices

O gerenciamento das estruturas de acesso da aplicação sendo desenvolvida no ambiente é iniciado pela listagem, em ordem alfabética, dos índices já cadastrados, se já existirem. Novamente, temos as operações de edição, exclusão e inserção dessas primitivas estão disponíveis através das âncoras “Edit”, “Delete” e “Add New Index”, respectivamente.

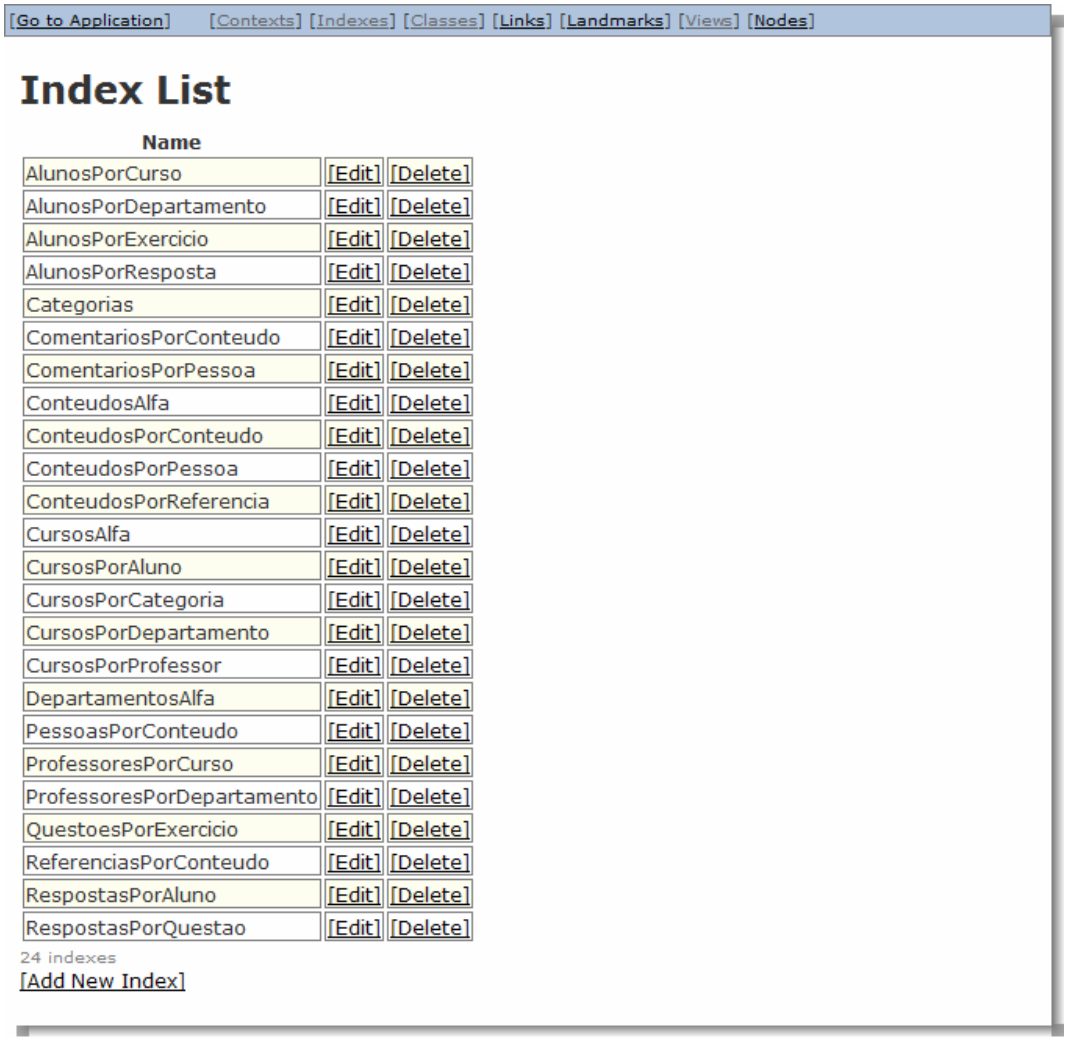


Figura 25 - Tela de listagem de índices da aplicação

Os tipos de índices suportados pela interface Web são os índices baseados em contextos e os índices baseados em consulta. Ao editarmos um índice, devemos selecionar o tipo de índice a ser construído, quando então a interface apresentará os atributos apropriados para cada tipo.

Para a edição de um índice baseado em consulta, assim como na edição de um contexto navegacional, devemos informar o nome do índice e a expressão de consulta cujo resultado fornece as entradas para o índice, assim como declarar os parâmetros utilizados nesta consulta, caso esta seja uma consulta parametrizada.

Para completar a edição do índice, devemos informar seus atributos. Veremos como fazer isso mais adiante.

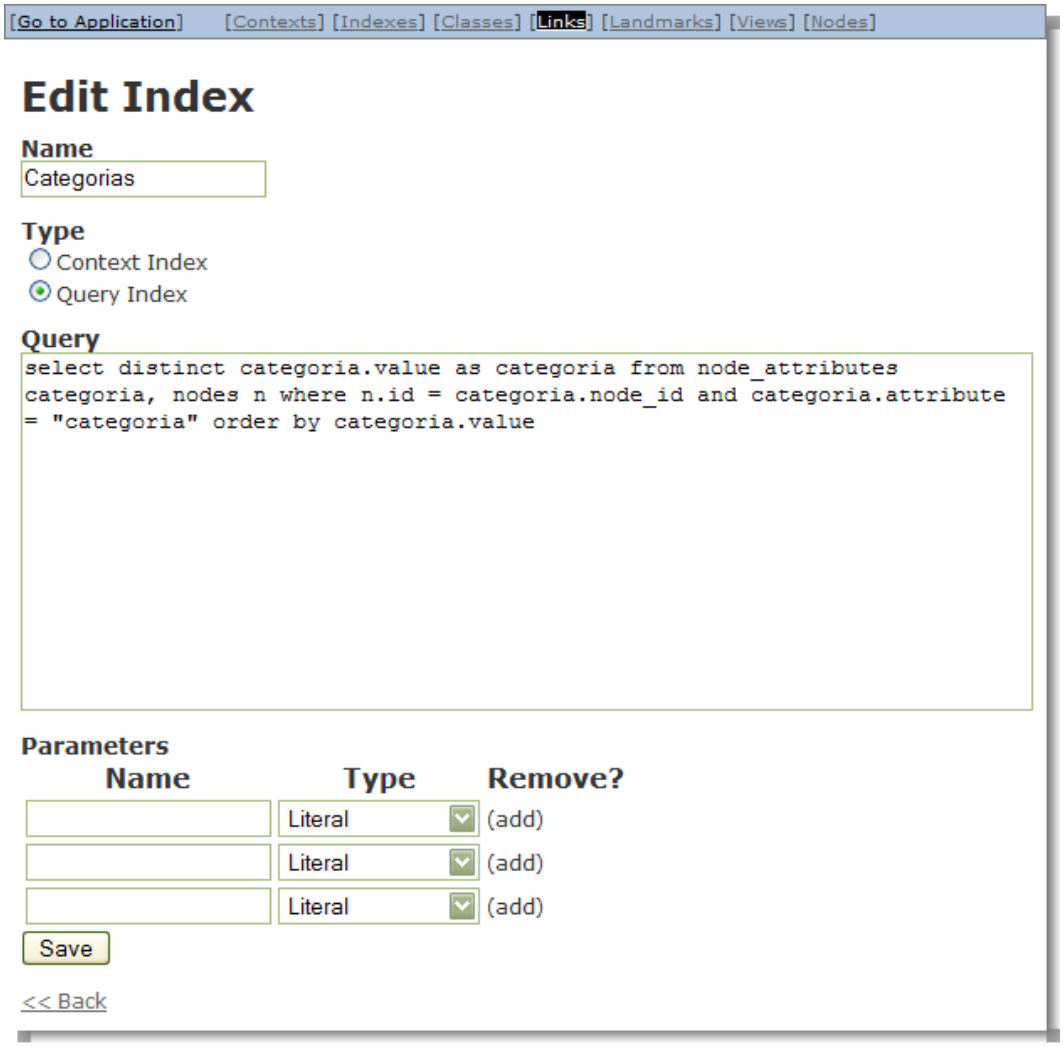


Figura 26 - Tela de edição de um índice baseado em consulta

Em um índice derivado de contexto, a expressão de consulta é obtida a partir do contexto navegacional do qual o índice é derivado, e, portanto, devemos informá-lo. Também não é necessário informar os parâmetros do índice, mesmo que ele seja parametrizado, pois neste caso, os parâmetros serão os mesmos do contexto navegacional associado.

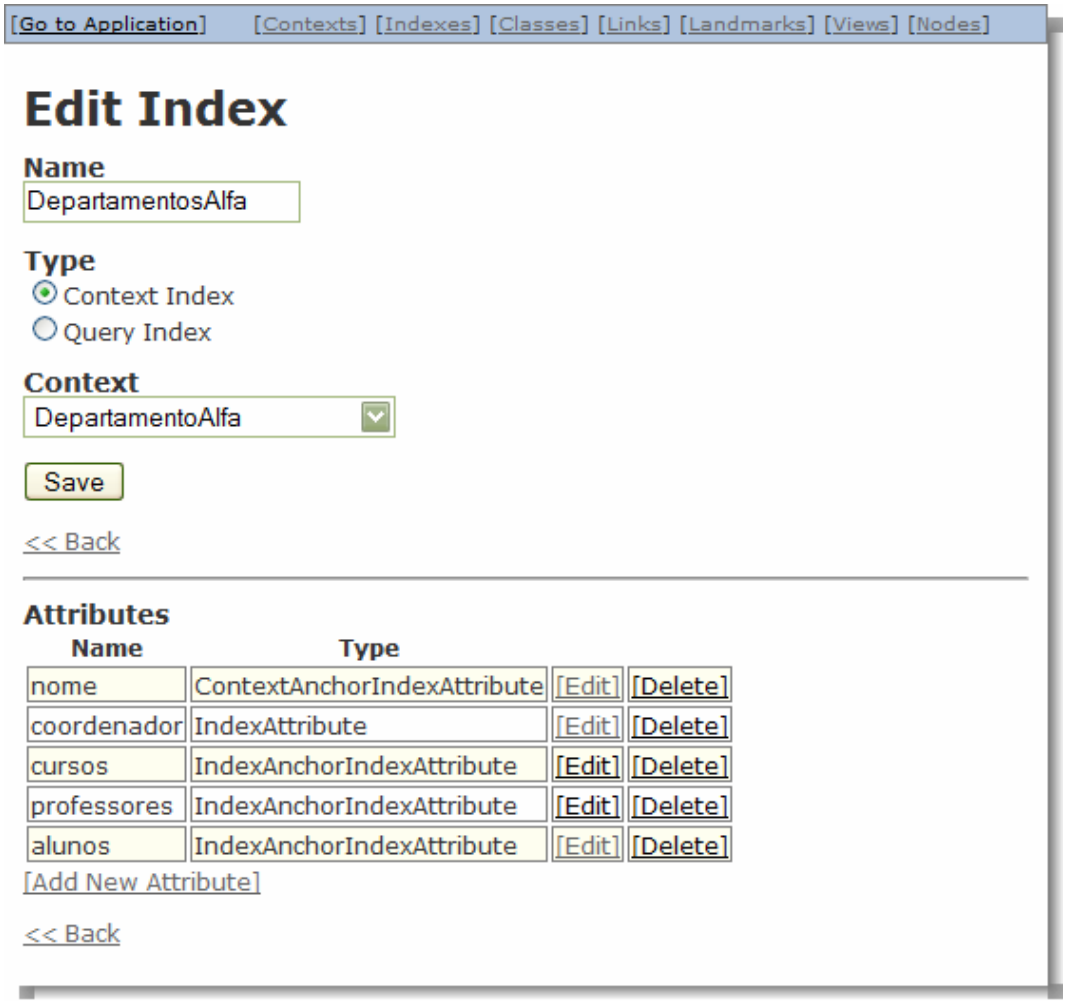


Figura 27 - Tela de edição para índice derivado de contexto

Abaixo dos dados básicos do índice, como podemos ver na imagem acima, o ambiente apresenta a lista de atributos do índice, oferecendo âncoras para inserção, edição e exclusão desses atributos.

Ao editarmos um atributo do índice, devemos primeiramente selecionar o tipo do atributo, que pode ser um atributo de tipo literal (“Simple Attribute”), um atributo de tipo âncora para contexto (“Context Anchor”) ou um atributo de tipo âncora para índice (“Index Anchor”). Ao selecionarmos o tipo de atributo, a interface apresenta os metadados apropriados a serem preenchidos.

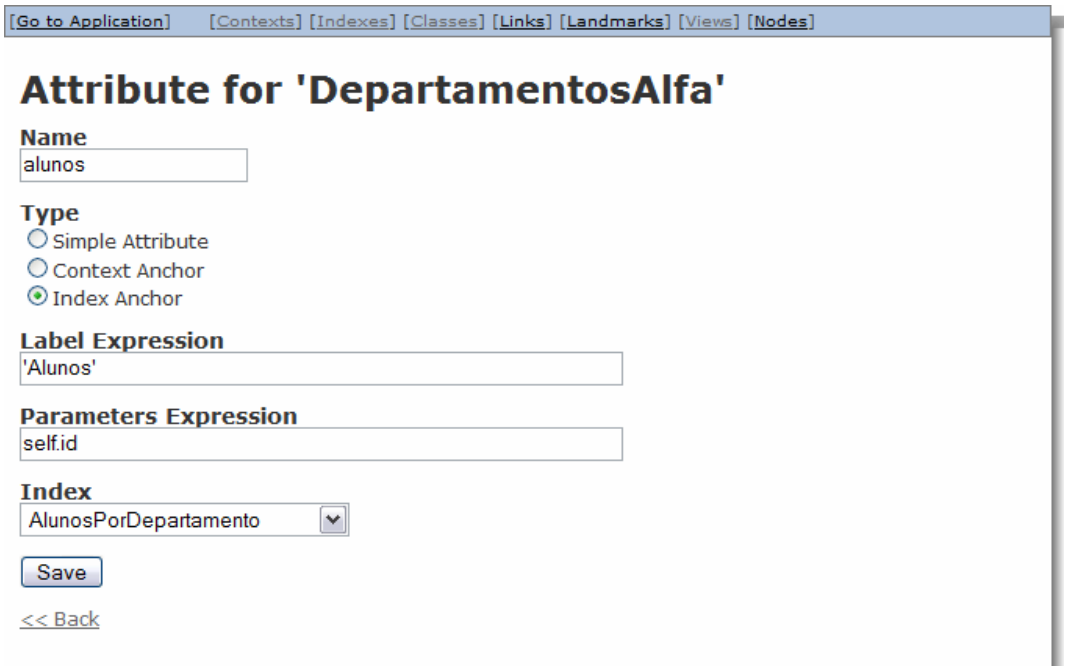


Figura 28 - Tela de edição para um atributo de índice do tipo âncora para índice



Para um atributo de tipo âncora para índice, como o apresentado na tela acima, devemos informar, além do nome do atributo, a expressão para etiqueta da âncora (“Label Expression”), escrita em linguagem Ruby. Esta expressão será avaliada em tempo de execução, no contexto de execução de cada elemento do índice. No exemplo acima, vemos que essa expressão é uma String com valor “Alunos” (por ser uma expressão dinâmica, devemos utilizar delimitadores de string se quisermos que o texto da etiqueta seja constante para todos os elementos do índice, por isso o uso de ‘’). Exemplos de expressões mais dinâmicas para a etiqueta desta âncora poderiam ser:

```
“Alunos em #{DateTime.now}” # => “Alunos em 2005-02-04T19:05”
”Alunos do Departamento #{self.nome}” # => “Alunos do Departamento Informática”
```

Além disso, devemos informar ainda o índice alvo da âncora e, caso o índice seja parametrizado, devemos informar a expressão de parâmetros que será avaliada no escopo de execução de cada elemento do índice, e que irá fornecer os valores dos parâmetros a serem usados na consulta formadora do índice.

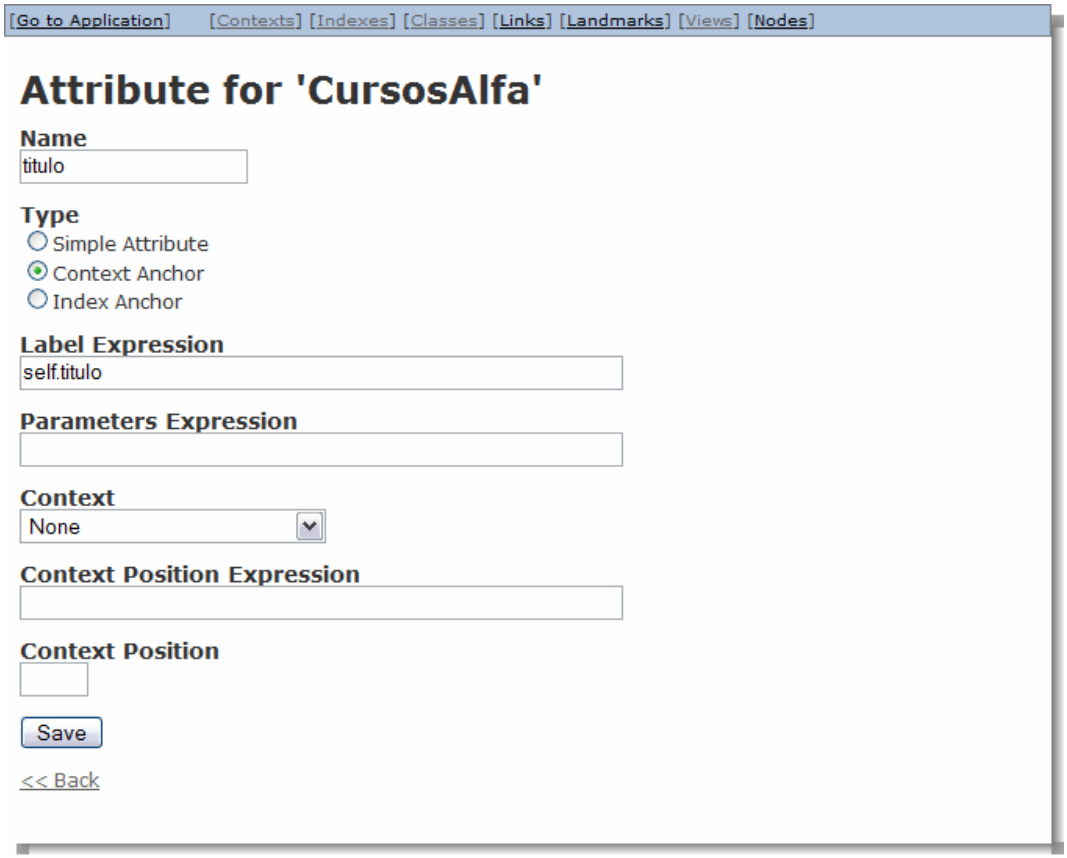


Figura 29 - Tela para edição de um índice de tipo âncora para contexto

Um atributo de índice do tipo âncora para contexto pode ser configurado de duas formas, dependendo do tipo do índice e do contexto alvo da âncora. Se o índice for derivado de contexto e o alvo da âncora deste atributo for direcionado aos elementos desse mesmo contexto, como na tela acima, não é necessário informar o contexto alvo para o atributo, pois o ambiente irá utilizar o contexto já associado ao índice. Da mesma forma, não é necessário neste caso, informar a expressão de parâmetros para o contexto (“Parameters Expression”), nem a expressão para localizar o nó alvo dentro do contexto (“Context Position Expression”) ou a posição do nó alvo dentro do contexto (“Context Position”). Restando apenas informar a expressão de etiqueta da âncora. O endereço de alvo da âncora será associado ao nó do contexto do qual o índice é derivado.

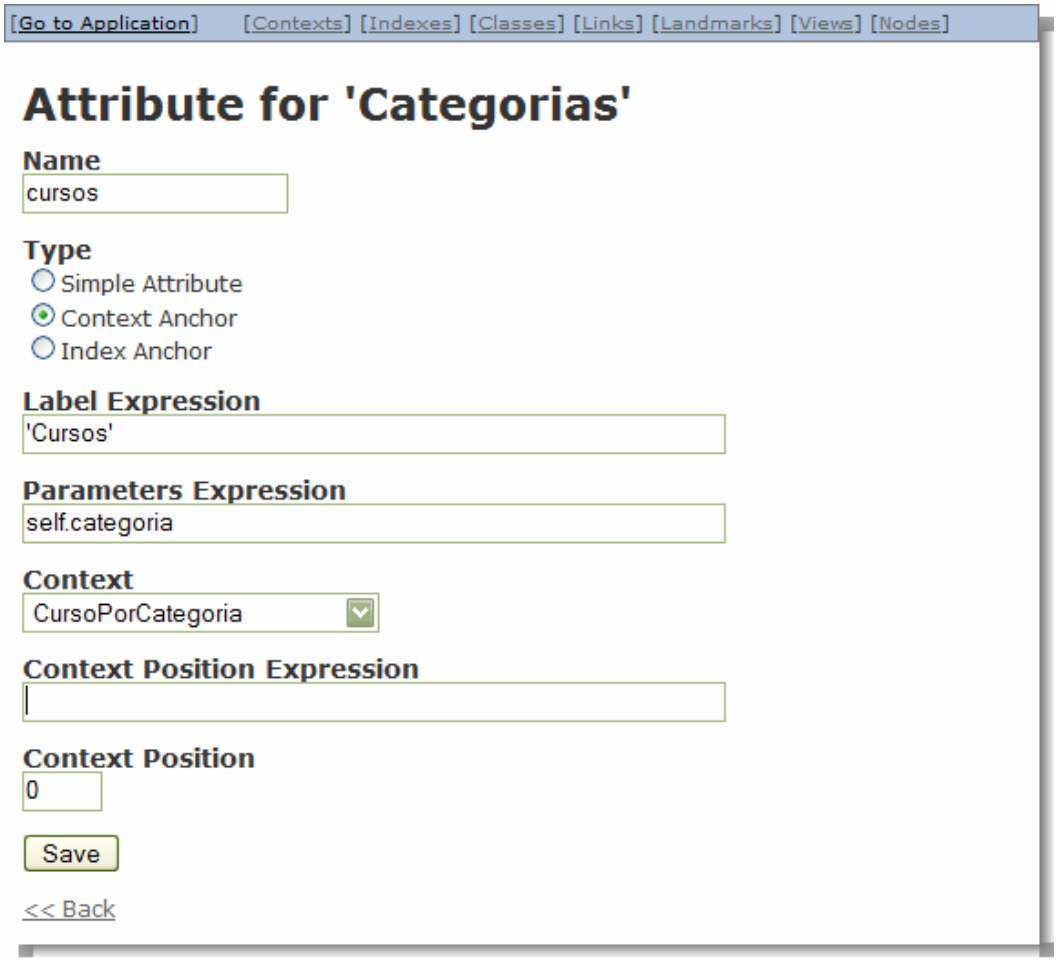


Figura 30 - Tela para edição de atributo de índice (âncora para contexto) em índice baseado em consulta

Se o contexto alvo da âncora deste atributo de índice for um contexto diferente do qual o índice é derivado ou o atributo for de um índice baseado em consulta, como na tela acima, então devemos informar o contexto alvo da âncora do atributo. Além disso, se o contexto alvo for parametrizado, devemos informar a expressão que será avaliada no escopo de execução de cada elemento do índice, para fornecer os valores desses parâmetros para a expressão de consulta do contexto.

A seguir, deveremos indicar qual a posição do nó corrente dentro do contexto que será alvo da âncora. Para isso, podemos fazê-lo informando diretamente a posição do nó corrente (“Context Position”), como no exemplo acima, que será então constante para todos os elementos do índice.

Alternativamente, podemos informar uma expressão em Ruby para determinação da posição do contexto (“Context Position Expression”) que será

avaliada no escopo de execução de cada elemento do índice e cujo resultado deverá ser o código de identificação do nó alvo no contexto.

Para este caso, vejamos um exemplo: suponha um índice baseado em consulta, em que temos um dos campos da consulta contendo o código de identificação de nós do tipo Professor e cujo campo foi denominado "professor_id". Se tivermos um atributo de índice apontando para um contexto "ProfessorAlfa", contendo nós do tipo Professor em ordem alfabética, a expressão para determinação da posição do nó corrente neste contexto seria "self.professor_id".

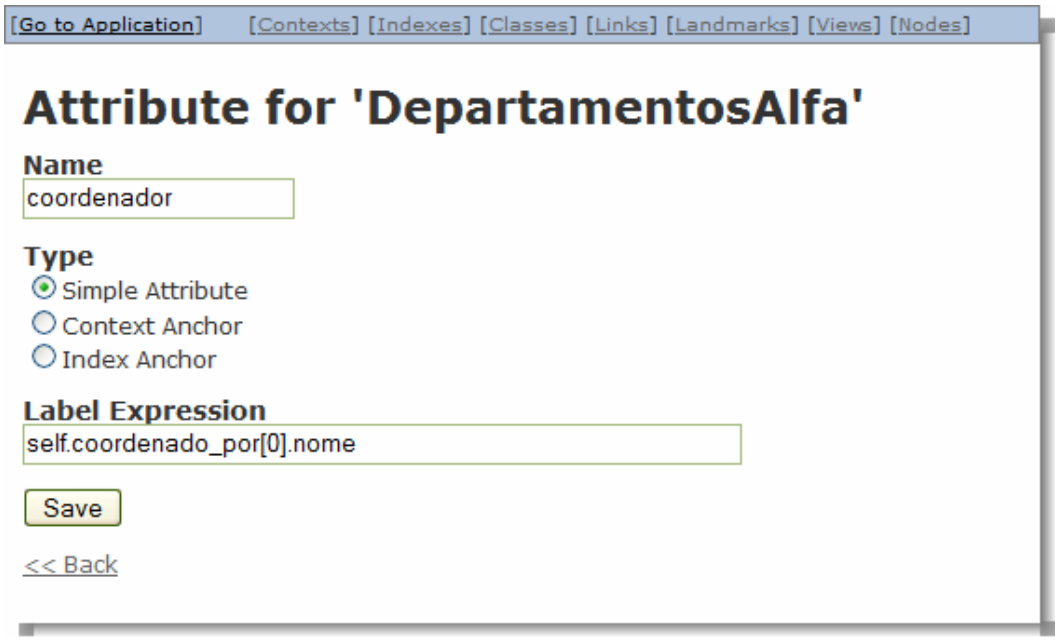


Figura 31 - Tela para edição de um atributo de índice de tipo simples

Para edição de um tipo de atributo simples, ou seja, que não seja uma âncora para contexto ou índice, mas apenas um texto informativo ou qualquer outro tipo de elemento visual, devemos apenas informar a expressão em Ruby que será avaliada no escopo de execução de cada entrada do índice para fornecer a etiqueta do atributo do índice, cujo resultado pode ser simplesmente textual ou código HTML. No exemplo visto acima, a expressão avalia o atributo "nome" em um nó ligado ao nó corrente através de um elo "CoordenadoPor".

### 4.7.1.3. Classes Navegacionais

As classes navegacionais são as primitivas mais complexas para serem definidas através da interface Web do ambiente, pois possuem, além de seus dados básicos, atributos, elos e operações.

A listagem das classes navegacionais é ordenada alfabeticamente pelo nome da classe, que apresenta também o atributo da classe usado como representação da mesma através de String e sua classe base, da onde é derivada, herdando seus atributos, elos e operações.

As operações habituais de inserção, edição e exclusão estão disponíveis nas âncoras “Add New Class” ao fim da lista, “Edit” e “Delete” ao lado de cada item da lista.

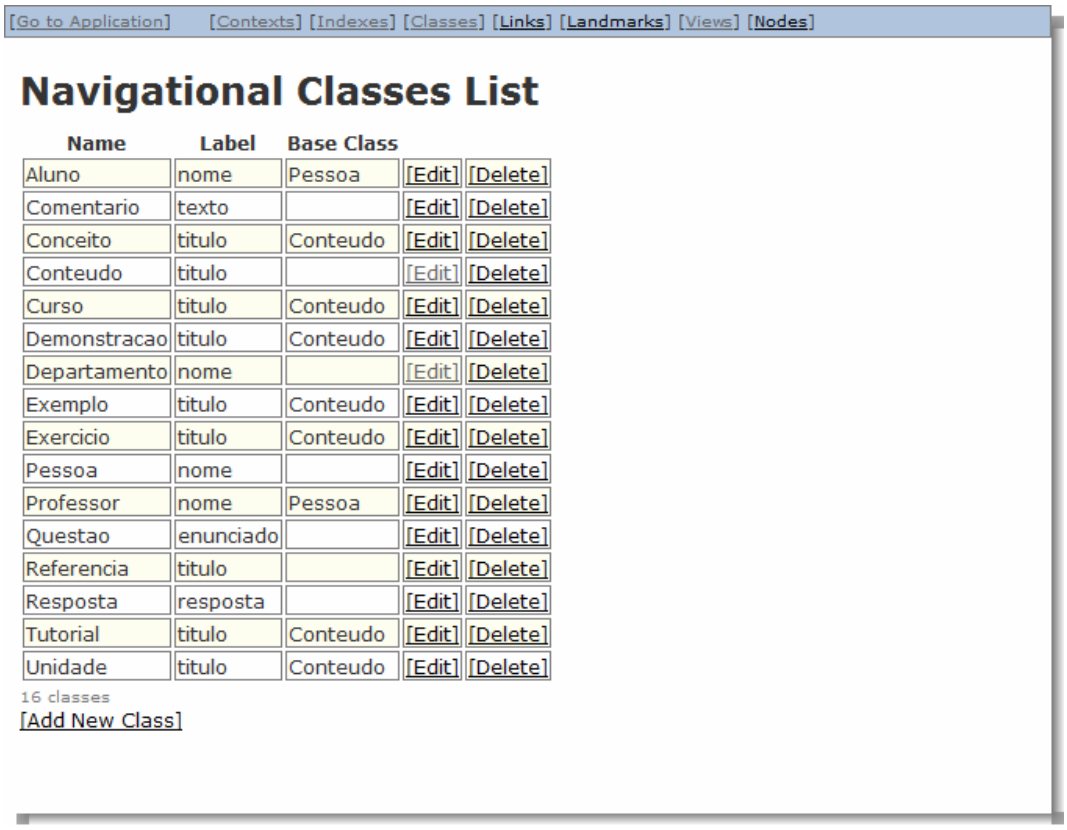


Figura 32 - Tela de listagem das classes navegacionais

Para inserção de uma nova classe navegacional ou edição de uma já existente, devemos informar apenas seu nome, iniciado com letra maiúscula, e opcionalmente sua classe base, caso esta seja derivada de alguma outra classe já cadastrada no ambiente.

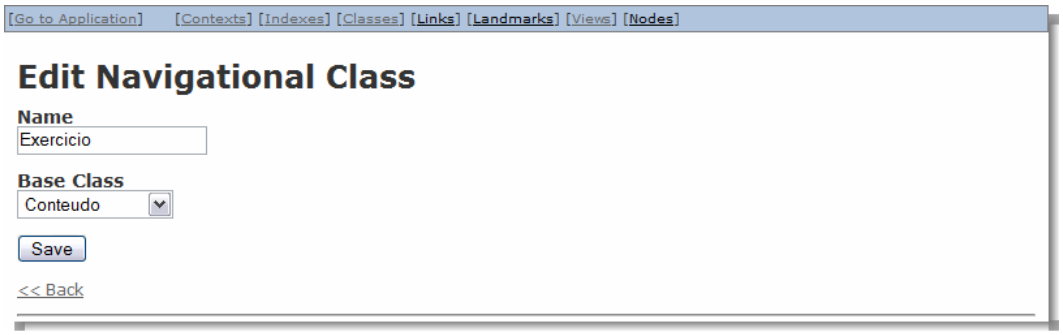


Figura 33 - Tela de edição dos dados básicos de uma classe navegacional

Se estivermos inserindo uma nova classe navegacional, apenas estes campos estarão disponíveis. Após confirmarmos os dados, através do botão “Save”, teremos acesso ao cadastro de atributos, elos e operações da classe.

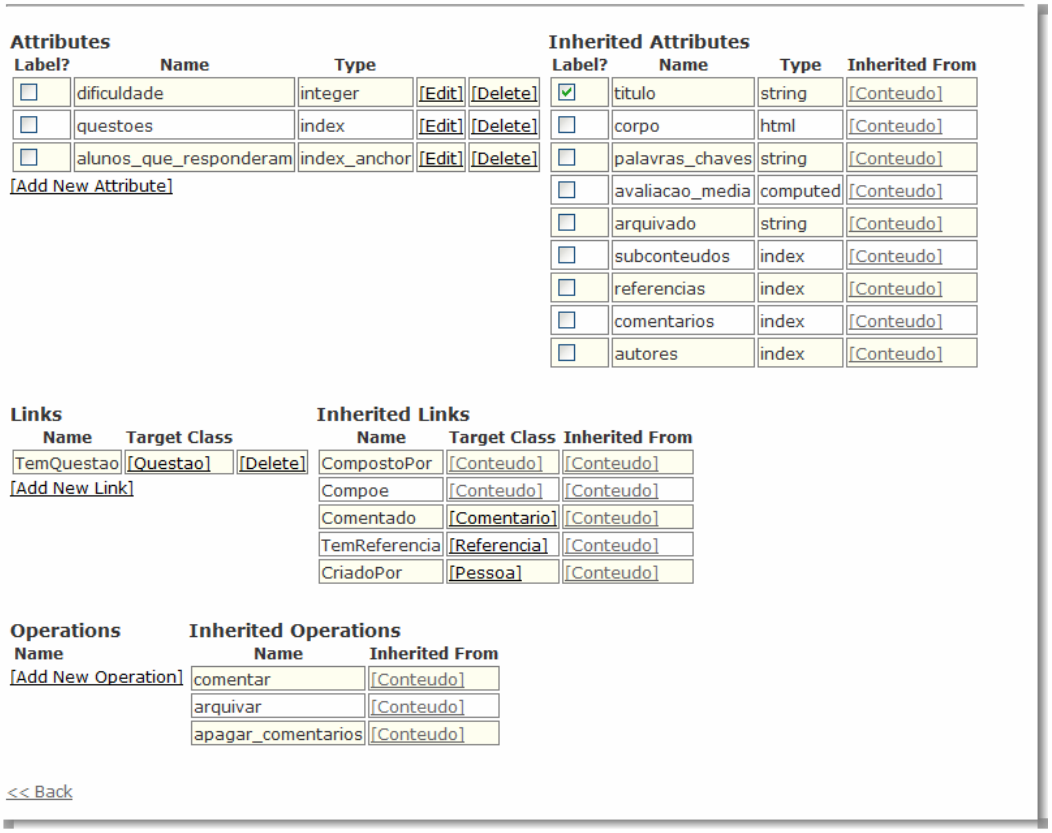


Figura 34 - Tela de listagem dos atributos, elos e operações de uma classe navegacional

Na tela de edição de uma classe navegacional também nos são apresentadas as listagens de atributos, elos e operações para aquela classe. Cada listagem possui duas tabelas: do lado direito vemos a tabela contendo os atributos, elos e operações herdados da classe base, caso alguma tenha sido informada, enquanto

que do lado esquerdo vemos a tabela contendo os atributos, elos e operações específicas desta classe.

Podemos apenas manipular efetivamente as tabelas do lado esquerdo, através de operações de inserção, edição e exclusão de seus elementos, enquanto que as do lado direito são informativas, fornecendo âncoras para edição das classes dos quais esses elementos foram herdados (coluna “Inherited From”).

Na lista de atributos, temos ainda a possibilidade de informar, através de marcação na coluna “Label?”, qual (apenas um deve ser marcado) destes atributos deve ser usado para resposta ao método “label”, que é utilizado pelo ambiente para gerar uma representação do nó em forma de String. Se nenhum for informado, o código identificador do nó será então usado para isso.

A lista de elos também informa a classe navegacional alvo do elo (coluna “Target Class”), oferecendo uma âncora para iniciar a edição desta.



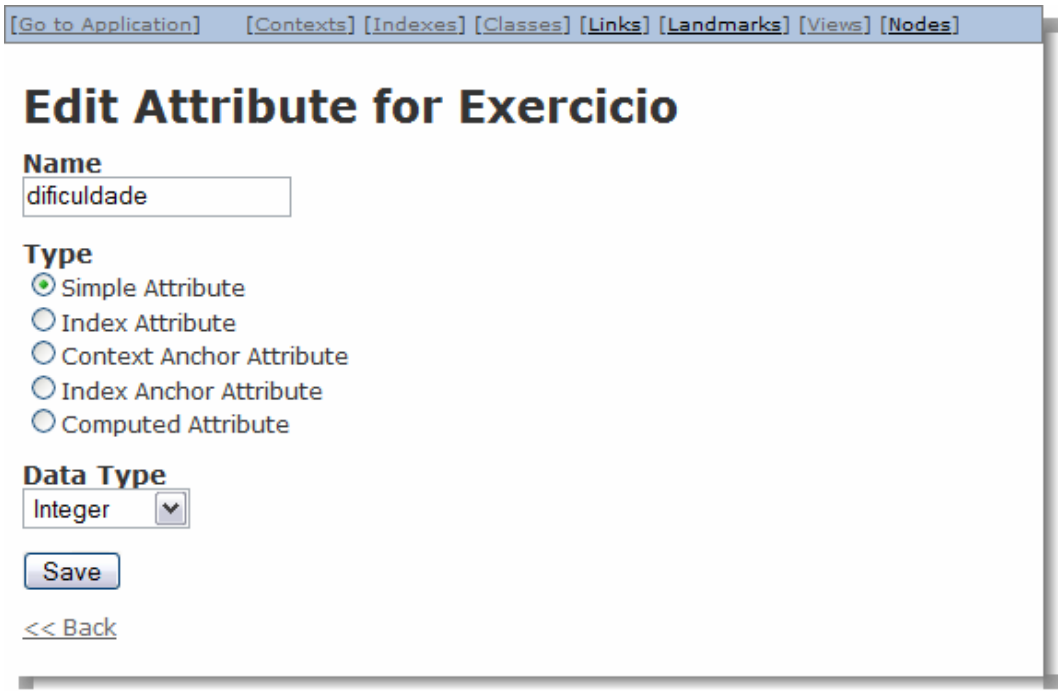


Figura 35 - Tela para edição de um atributo de tipo simples

Ao editar um atributo de classe navegacional, devemos escolher entre os 5 tipos de atributos suportados pelo ambiente, que são: atributos simples (de tipos literais, como String, Integer, Image e outros), atributos de tipo índice, atributos de tipo âncora para contexto ou âncora para índice e atributos computados. Após a

seleção do tipo, a interface exibirá os campos adequados para configuração completa do atributo.

Para atributos do tipo simples, devemos simplesmente informar o nome do atributo, que deve ser iniciado por letra minúscula, e o tipo de dado do atributo, dentre os tipos suportados pelo ambiente.

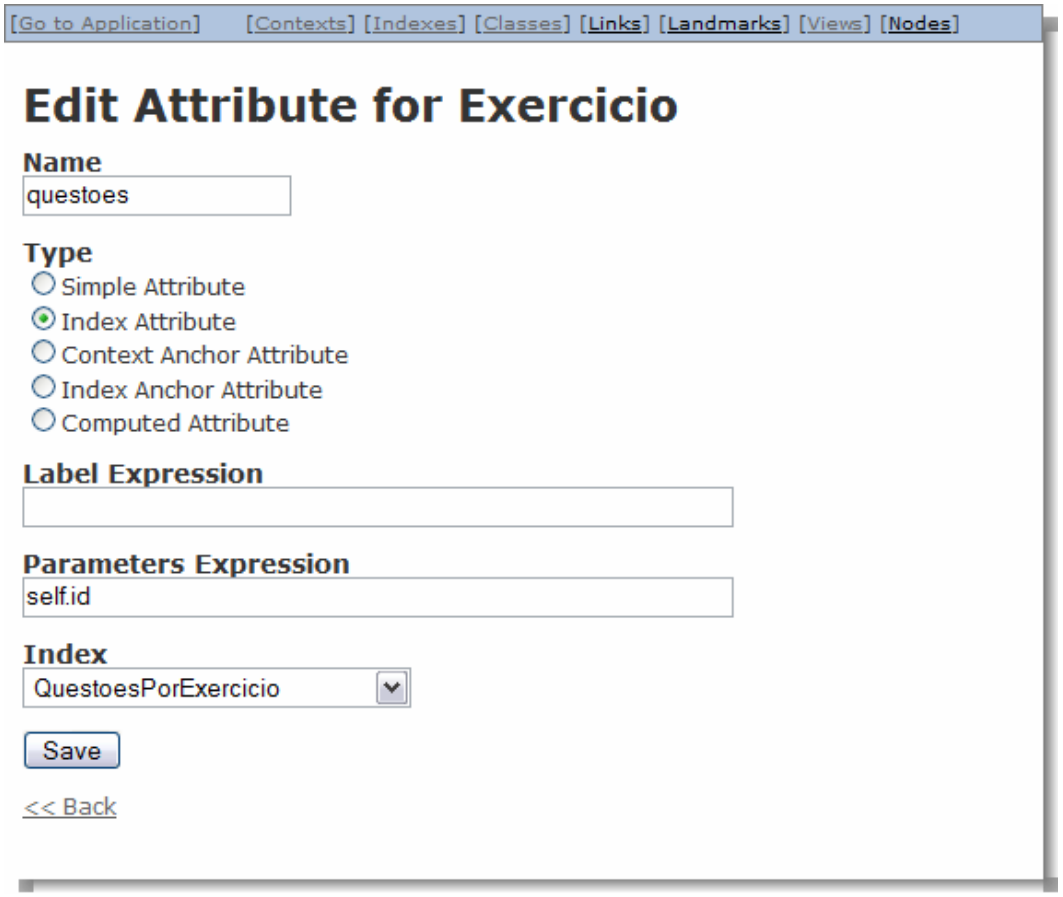


Figura 36 - Tela para edição de um atributo de classe navegacional do tipo índice

Para edição de um atributo de tipo índice, devemos informar obrigatoriamente o índice associado a este atributo. Opcionalmente podemos informar a expressão de parâmetros que será avaliada no escopo de execução do nó e que fornecerá os valores dos parâmetros para o índice associado, caso este esteja baseado em consulta ou contexto parametrizado. Podemos informar ainda a expressão avaliada também no escopo de execução do nó, para determinação da etiqueta/título do índice. Caso não seja informada esta expressão, o ambiente assumirá como etiqueta o próprio nome do índice associado.

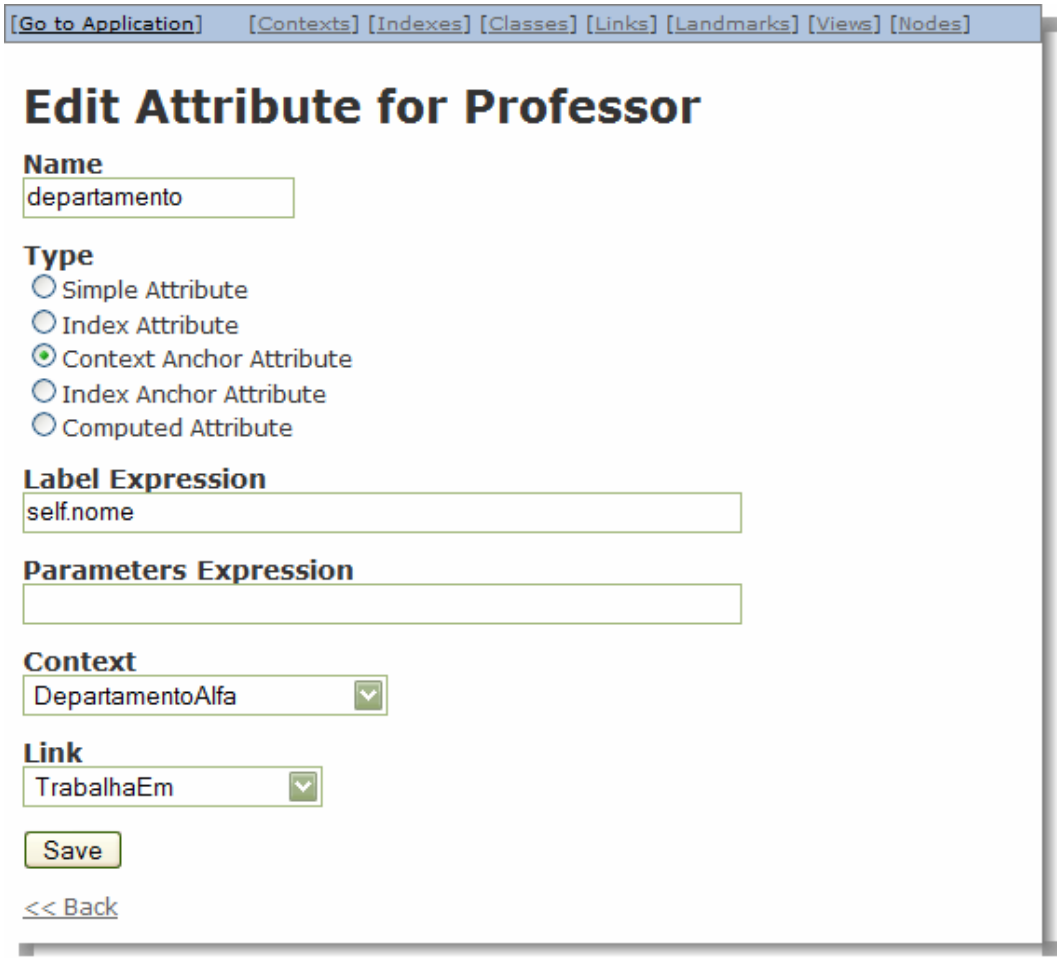


Figura 37 - Tela de edição de atributo de classe naveg. de tipo âncora para contexto

Para edição de um atributo de tipo âncora para contexto, devemos obrigatoriamente informar o contexto alvo da âncora, assim como o elo que determinará o nó corrente neste contexto alvo. Devemos ainda informar a expressão, avaliada sob o escopo de cada entrada do índice e cujo resultado será usado como etiqueta da âncora. Opcionalmente podemos informar a expressão de parâmetros que será avaliada sob o escopo de execução de cada entrada do índice e cujos valores serão passados para o contexto alvo.

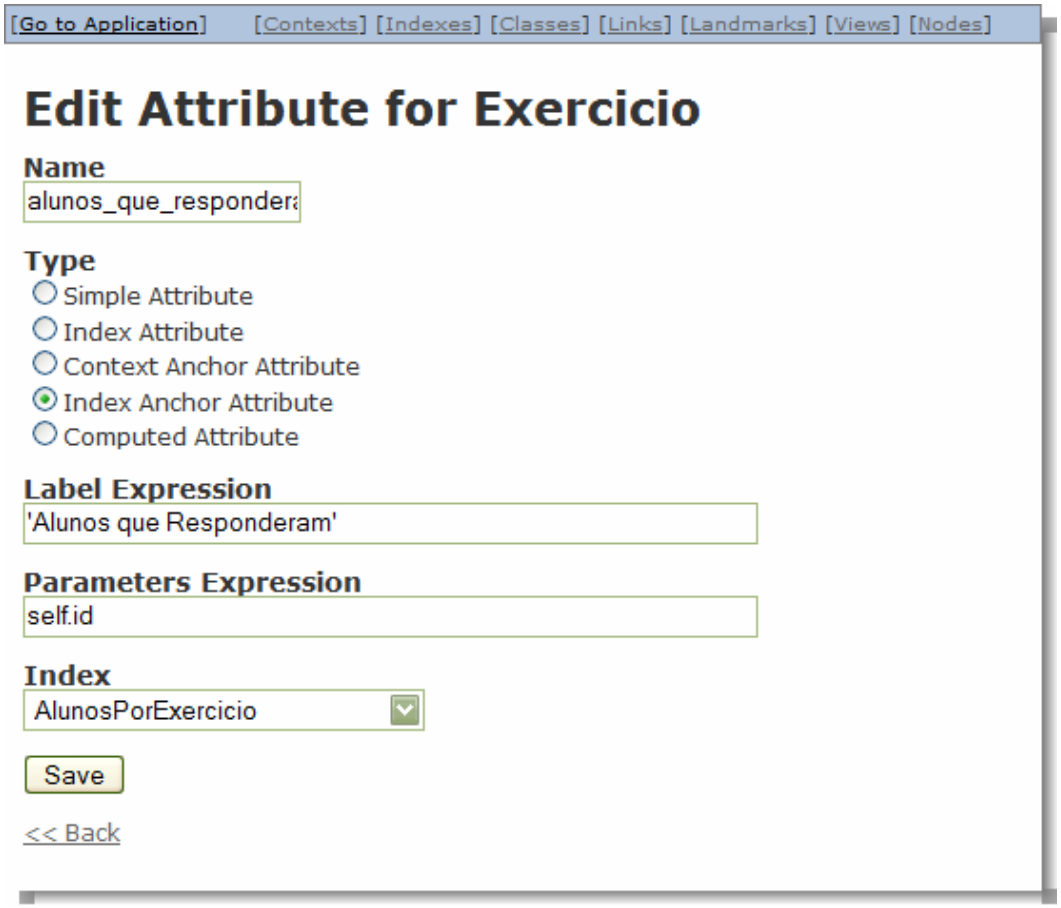


Figura 38 - Tela de edição de atributo de classe naveg. do tipo âncora para índice

Para edição de um atributo do tipo âncora para índice, devemos proceder da mesma forma que para edição de atributo do tipo índice, informando o índice alvo da âncora e opcionalmente a expressão de parâmetros para passagem ao índice, caso sua expressão de consulta seja parametrizada, e também a expressão de etiqueta, avaliada sob escopo do nó que contém a âncora, e que se deixada em branco, será assumido o nome do índice alvo como etiqueta da âncora.

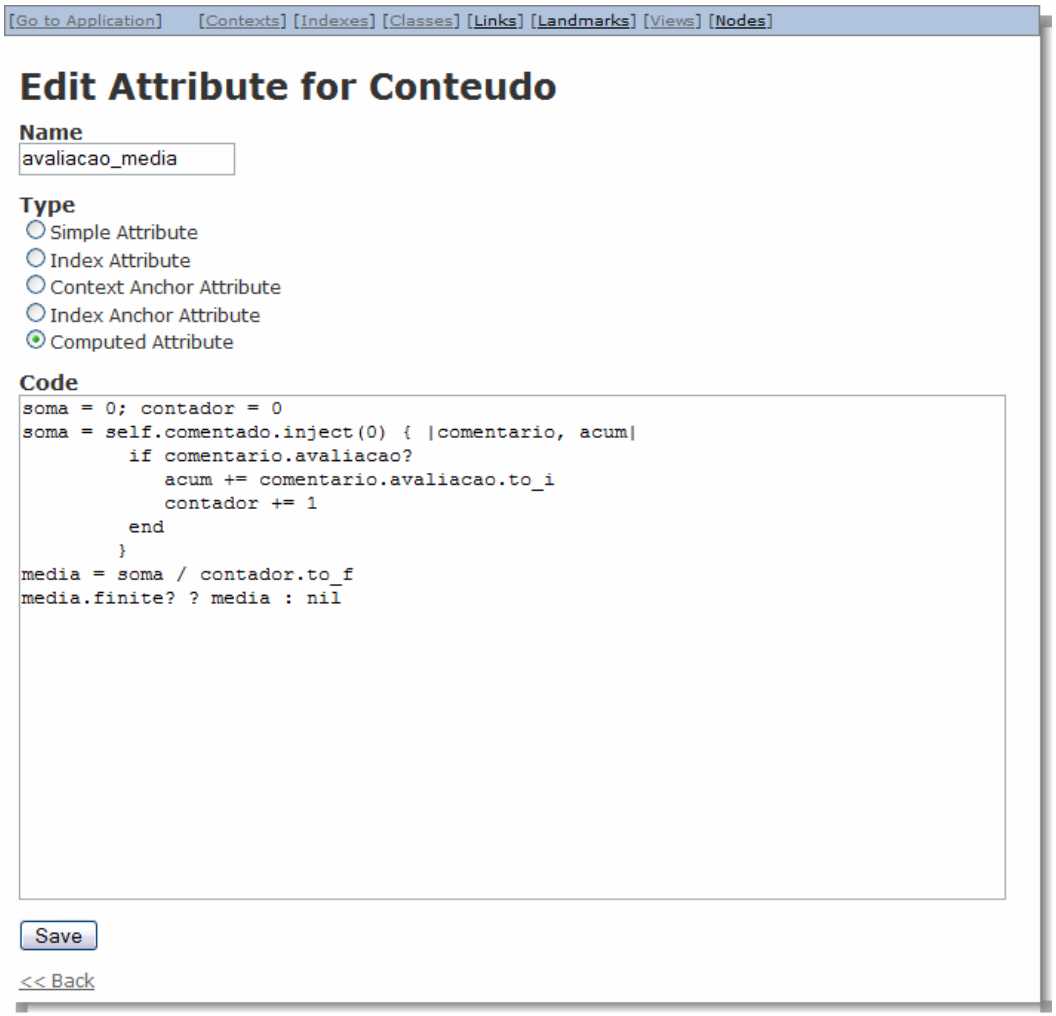


Figura 39 - Tela de edição de um atributo de classe navegacional de tipo computado

Para edição de um atributo de tipo computado, devemos informar além do nome do atributo, o trecho de código em Ruby que será avaliado sob o escopo de execução do nó para fornecer o resultado que será utilizado para apresentação do valor do atributo.

Na manipulação de operações de classes navegacionais, utilizamos as âncoras de inserção, edição e exclusão na listagem correspondente.

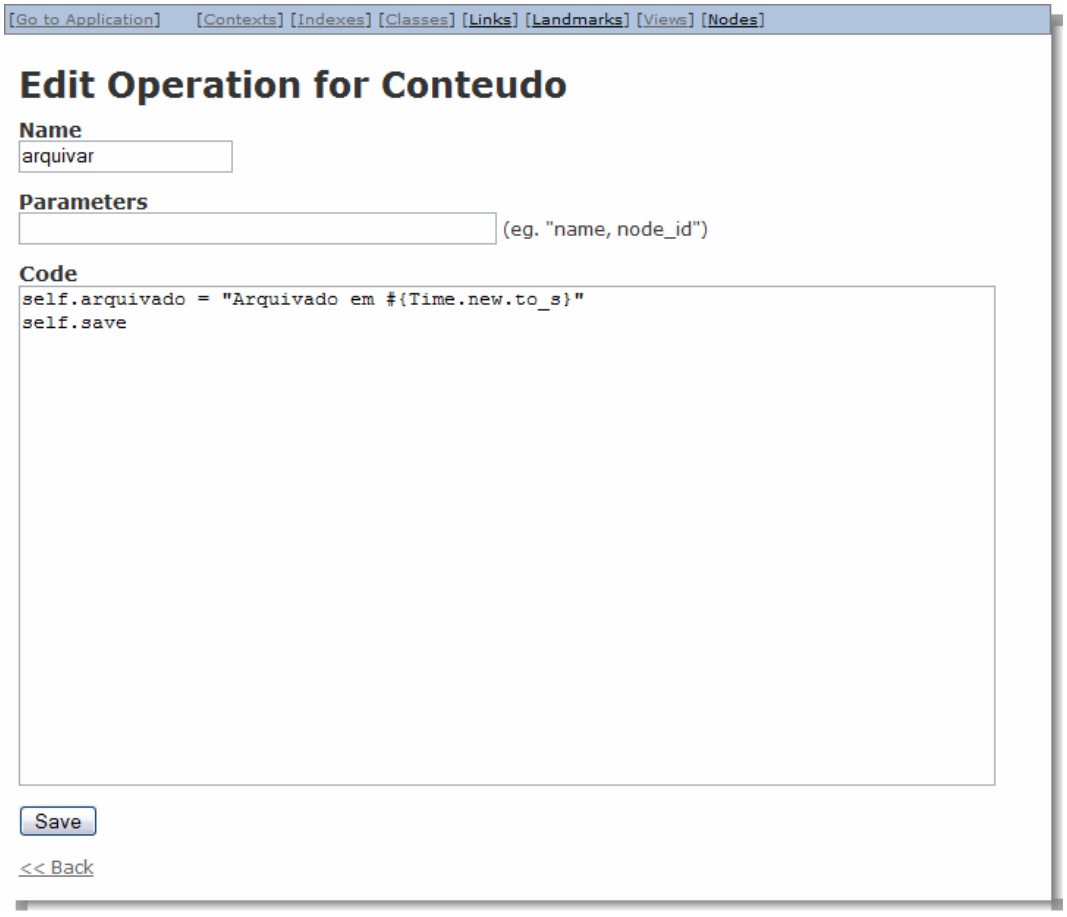


Figura 40 - Tela de edição de operação de classe navegacional

Para adicionar ou editar uma operação a uma classe navegacional, devemos informar o nome da operação, iniciado por letra minúscula e o trecho de código em Ruby que será executado sob escopo do nó sob o qual a operação foi invocada. Opcionalmente, podemos informar uma lista de parâmetros, separados por vírgula, que deverão ser passados à operação como argumentos.

Como já mencionado, operações invocadas sob o contexto de navegação recebem na variável "@context", uma instância da classe "Context" contendo o contexto navegacional corrente.

### 4.7.1.4.Elos

A listagem de elos apresenta em ordem alfabética todas as classes de associação definidas para a aplicação, indicando a classe navegacional de origem e de destino, e opcionalmente, o elo de semântica inversa. As operações de inserir, editar e excluir estes elos estão disponíveis, como de costume, através das âncoras "Edit", "Delete" e "Add New Link" ao fim da listagem.

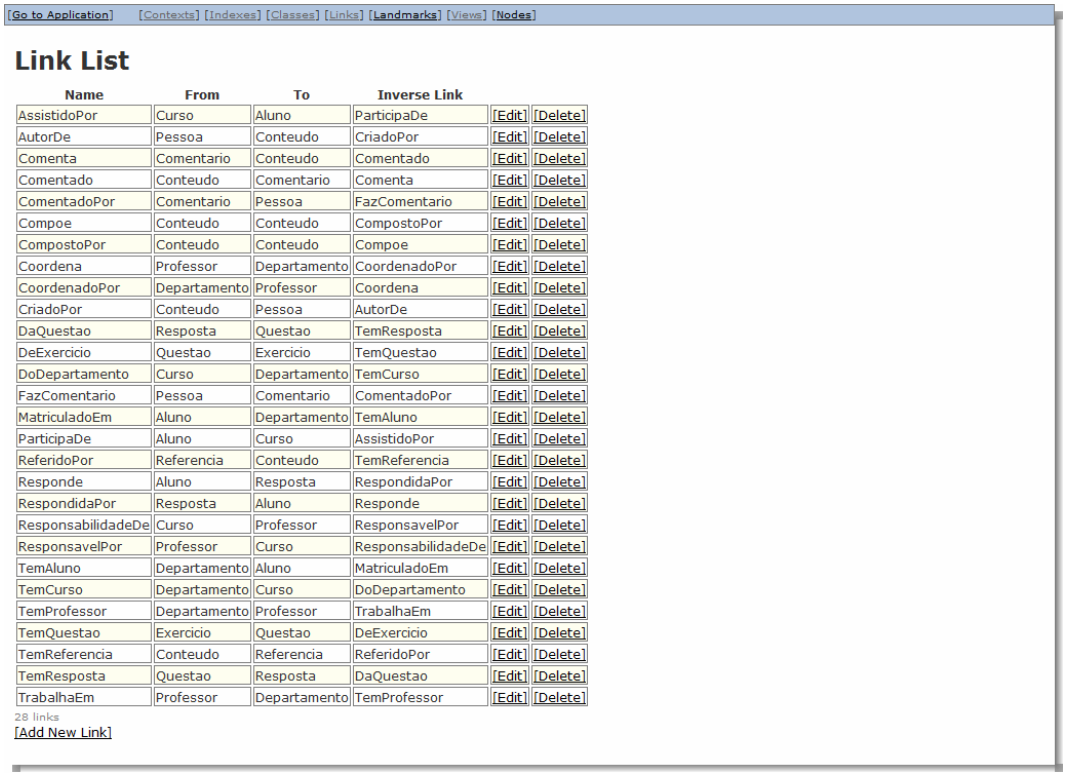


Figura 41 - Tela de listagem de elos da aplicação

Vale notar que as mesmas operações de inserção, edição e exclusão de elos também encontram-se disponíveis na tela de edição de classe navegacional, na listagem de elos. No entanto, lá vemos apenas os elos cuja classe de origem é a que está sendo editada. Na tela acima, podemos obter uma visão de todos os elos já cadastrados para a aplicação.

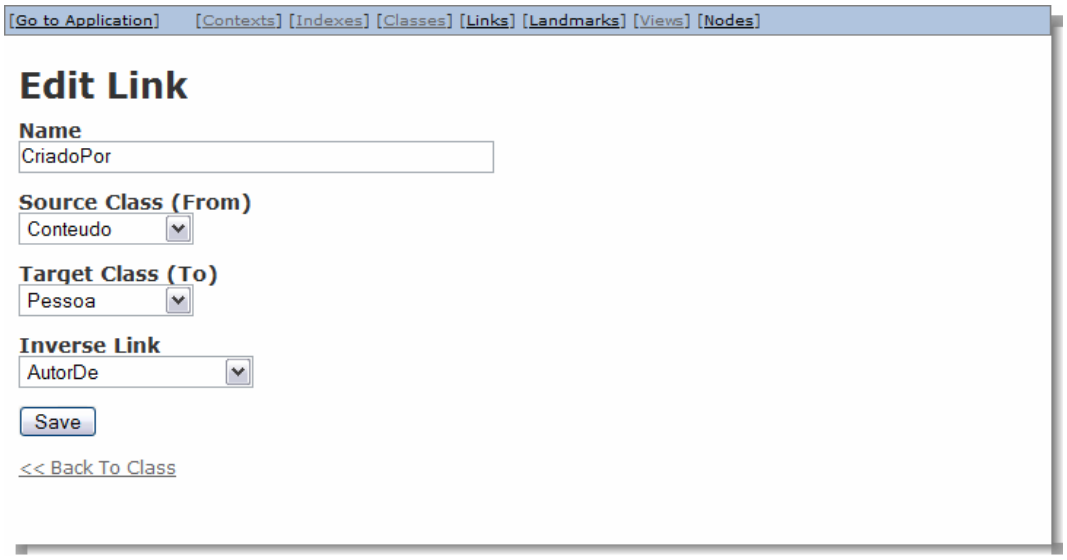


Figura 42 - Tela de edição de elo

Para editar um elo, devemos informar seu nome, iniciado por letra maiúscula, e obrigatoriamente informar a classe navegacional de origem e a classe navegacional de destino do elo. Opcionalmente podemos informar o elo de semântica inversa ao do elo correntemente sendo editado.

### 4.7.1.5. Landmarks

Na listagem de landmarks podemos visualizar todos os landmarks já cadastrados para a aplicação, ordenados pelo seu atributo de posição. Temos disponíveis, assim como nas outras listagens, as funções de inserção, edição e exclusão de landmarks através das âncoras “Add New Landmark”, “Edit” e “Delete”.

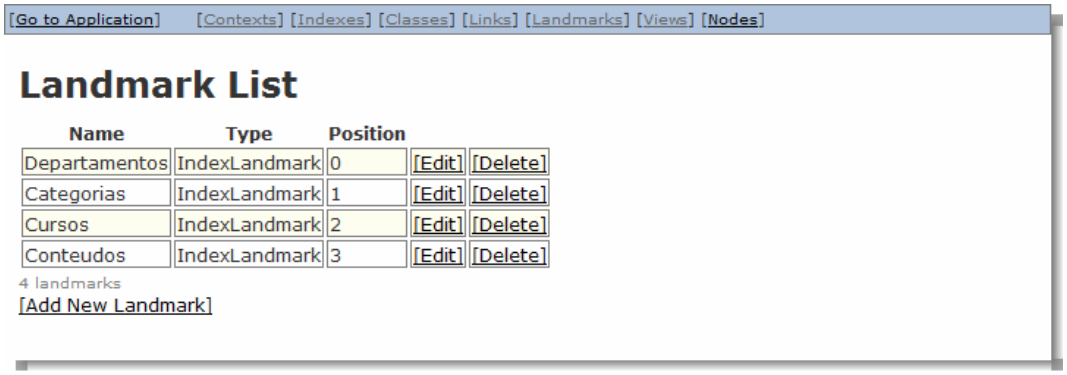


Figura 43 - Tela de listagem de landmarks

Existem 2 tipos de landmarks suportados pelo ambiente e que são representados através de âncoras: âncora para índice e âncora para contexto. Ao editar um landmark, devemos selecionar primeiramente o tipo de landmark e a interface apropriada nos será apresentada pelo ambiente.

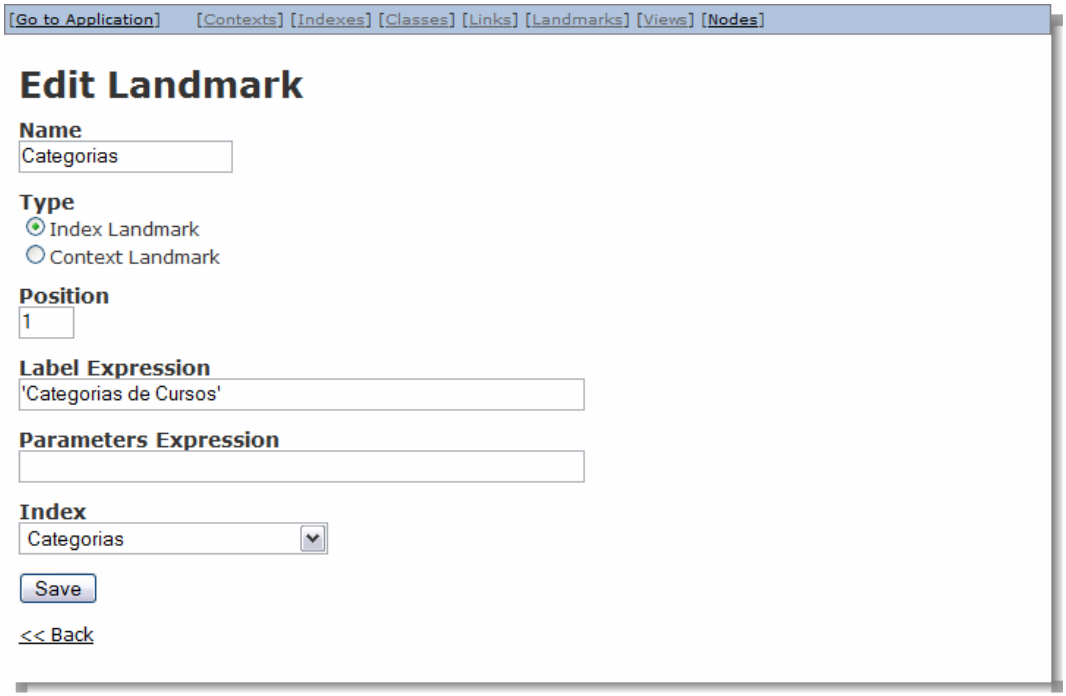


Figura 44 - Tela de edição de landmark do tipo âncora para índice

Para editarmos um landmark do tipo âncora para índice, devemos fornecer o nome do landmark e sua posição relativa aos outros landmarks da aplicação. Devemos também selecionar qual o índice que será alvo da âncora gerada através dele e devemos preencher a expressão da etiqueta da âncora, avaliada sob o escopo de execução do landmark. Opcionalmente podemos fornecer a expressão de parâmetros, avaliada sob o escopo de execução do landmark, cujo resultado será passado ao índice associado na forma de argumentos para a expressão de consulta parametrizada do mesmo.

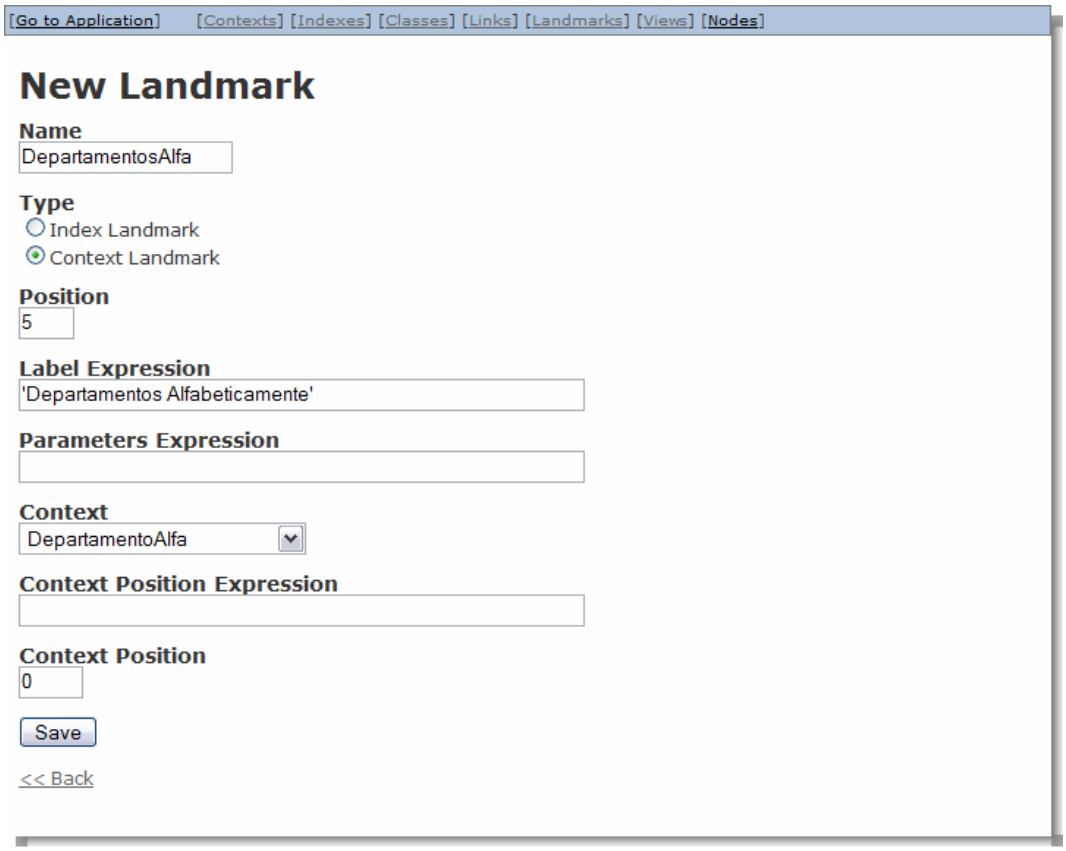


Figura 45 - Tela de edição de um landmark do tipo âncora para contexto

Em um landmark do tipo âncora para contexto, devemos fornecer além do nome, posição do landmark e a expressão de etiqueta da âncora, o contexto alvo da âncora. Para determinar o nó alvo do contexto associado, devemos fornecer ainda a posição deste nó diretamente ou a expressão avaliada sob o escopo de execução de cada nó do contexto e que se verdadeira, irá definir a posição da posição corrente no contexto. Opcionalmente, devemos fornecer a expressão de parâmetros cuja avaliação fornecerá os argumentos para a expressão de consulta sobre a qual o contexto associado se baseia.

#### 4.7.1.6. Visões

A listagem de visões fornece todas as visões customizadas já cadastradas para aplicação, ordenadas alfabeticamente e indicando seu tipo. Como nos outros casos, temos a possibilidade de inserir, editar ou excluir visões através das âncoras “Add New View”, “Edit” e “Delete”.

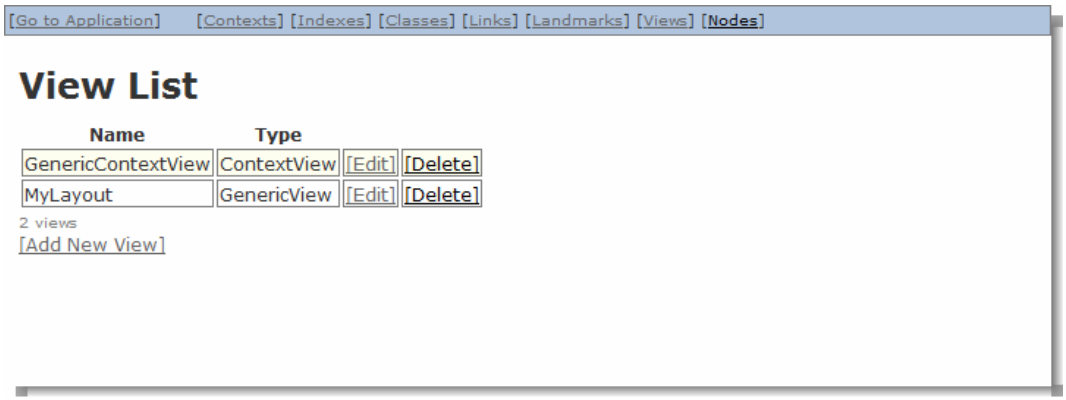


Figura 46 - Tela de listagem das visões customizadas da aplicação

O ambiente suporta 3 tipos de visões: visões genéricas, utilizadas na definição de componentes visuais reutilizáveis ou layouts, visões de classe em contexto, utilizadas para determinadas combinações de classes navegacionais e contextos, e visões de índices, utilizadas na exibição de índices.

Para editar visões devemos fornecer seu nome e indicar o tipo desejado. Após a indicação do tipo, o ambiente adequará a interface de acordo com nossa escolha.

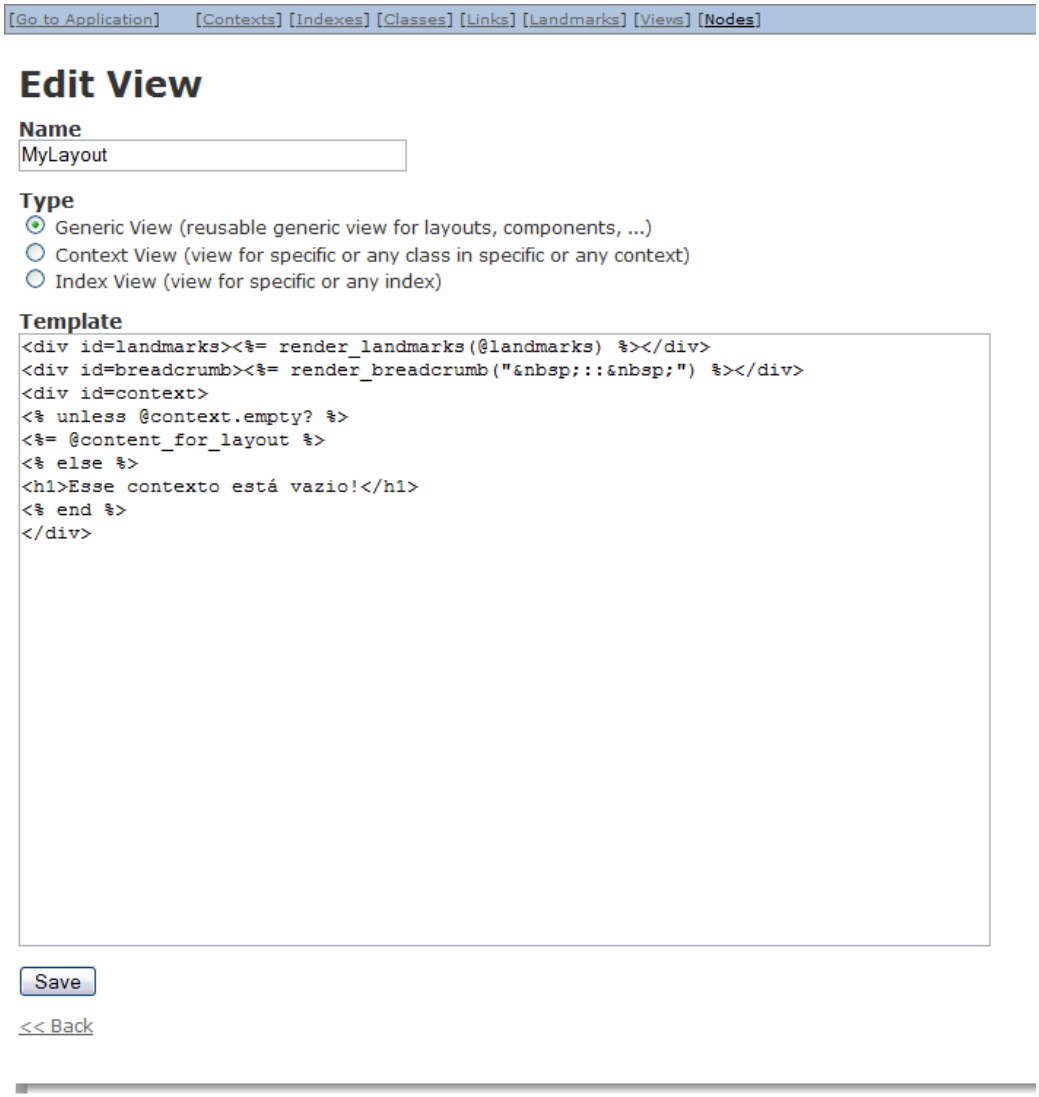


Figura 47 - Tela para edição de uma visão customizada genérica

Na definição de uma visão genérica, devemos apenas fornecer o código misto, utilizando Ruby (tirando proveito da biblioteca de funções de auxílio já apresentada) e HTML, para preencher o template da visão.

Se a visão genérica for utilizada como layout para outras visões, devemos utilizar no template a variável "@content_for_layout" para indicar a posição de inserção do resultado do processamento do template da visão associada, como podemos ver acima.

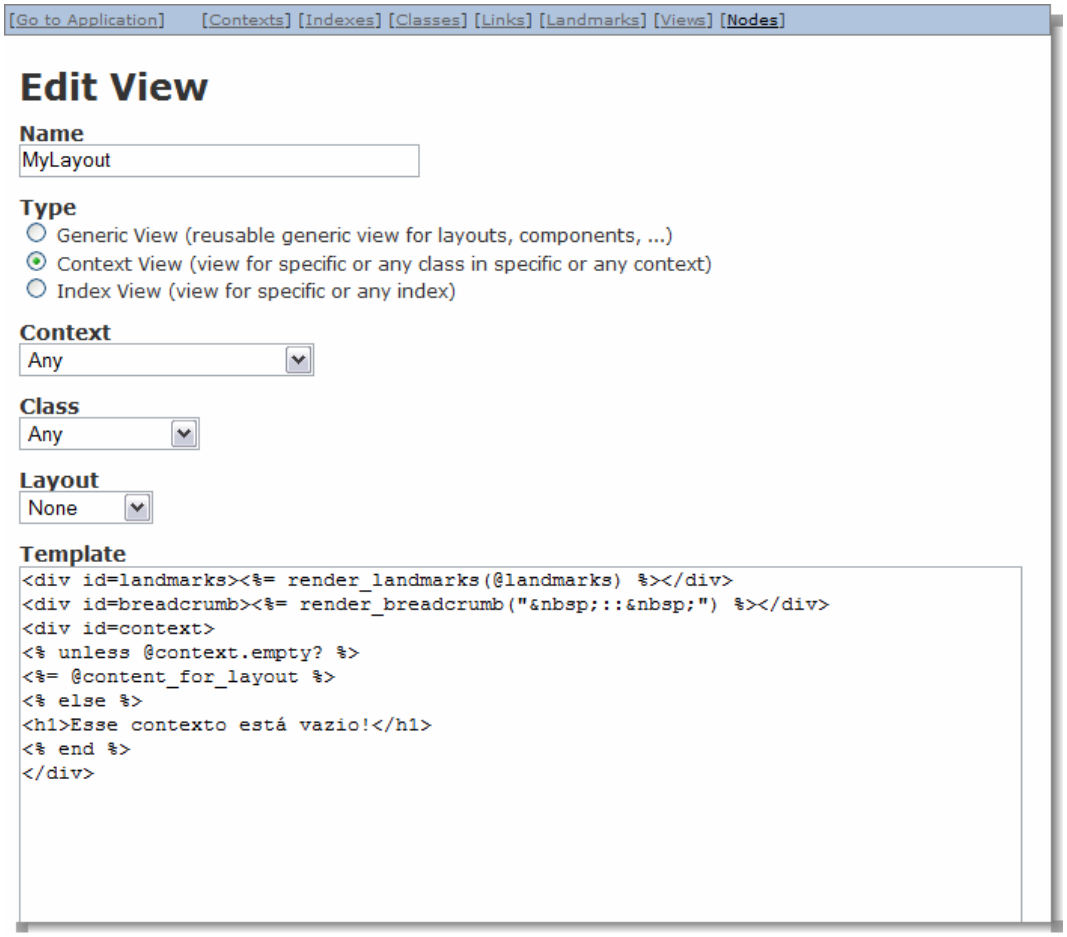


Figura 48 - Tela de edição de visão customizada para classe em contexto

Para a definição de uma visão customizada para classe em contexto, devemos, além de fornecer o nome e o template para a visão, indicar a combinação da classe navegacional e do contexto sobre os quais essa visão deverá ser aplicada. Podemos indicar a generalização da combinação através da opção “Qualquer” (“Any”), quando a visão será então aplicada a todos os contextos e/ou classe navegacionais. Opcionalmente podemos indicar uma visão genérica cujo template será utilizado como layout desta visão.

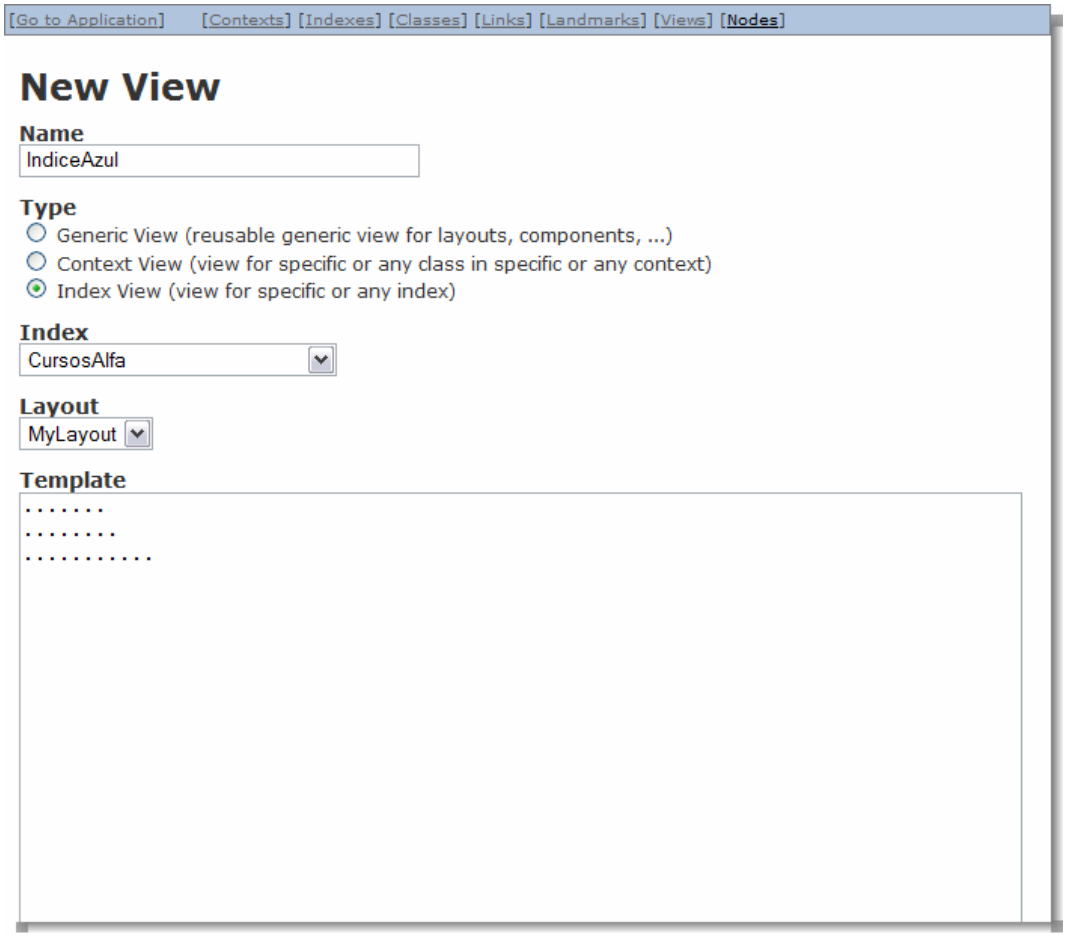


Figura 49 - Tela de edição de uma visão customizada para índice

Para edição de uma visão customizada para índice, devemos, além de fornecer o nome e o código do template, indicar o índice sobre ao qual esta visão será aplicada. Se nesta indicação, escolhermos a opção “Qualquer” (“Any”), então esta visão será aplicada a todos os índices da aplicação.

Opcionalmente, podemos indicar a visão genérica cujo template será utilizado como layout desta visão.

### 4.7.1.7. Nós

A listagem de nós da aplicação apresenta uma característica diferenciada das outras telas de listagem do ambiente. Esta listagem não apresenta todos os nós já cadastrados para aplicação, mas os filtra pela classe navegacional. Dessa forma, para visualizarmos a lista de nós, devemos antes indicar de qual classe navegacional serão os nós visualizados.

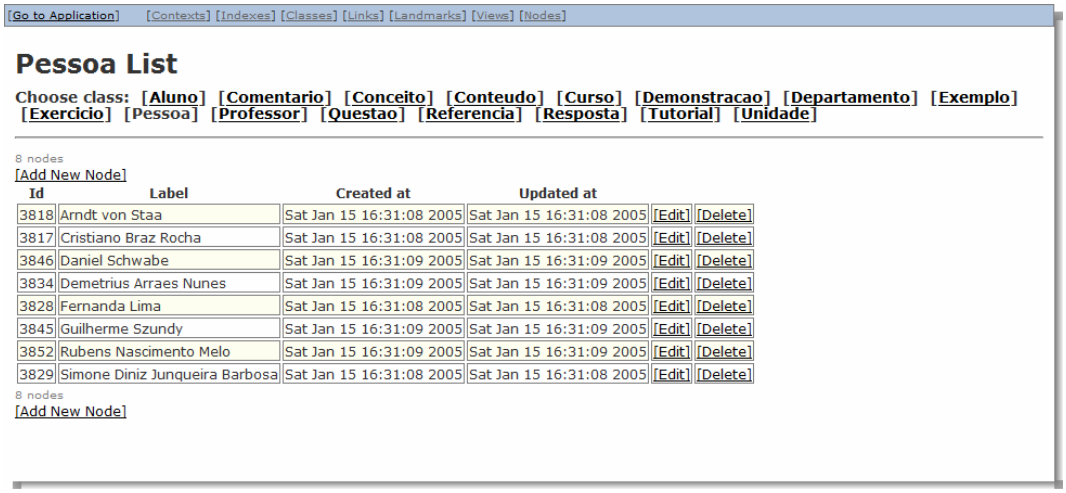


Figura 50 - Tela de listagem de nós de uma classe navegacional escolhida

Após a exibição, temos à nossa disposição a lista dos nós correspondentes à classe navegacional selecionada, ordenada pelo atributo de etiqueta indicado na definição da classe navegacional, assim como o código identificador de cada nó e a data/hora de criação e útima atualização do nó.

As funções de inserção, edição e exclusão de nós nesta listagem estão disponíveis através das âncoras “Add New Node”, “Edit” e “Delete”.

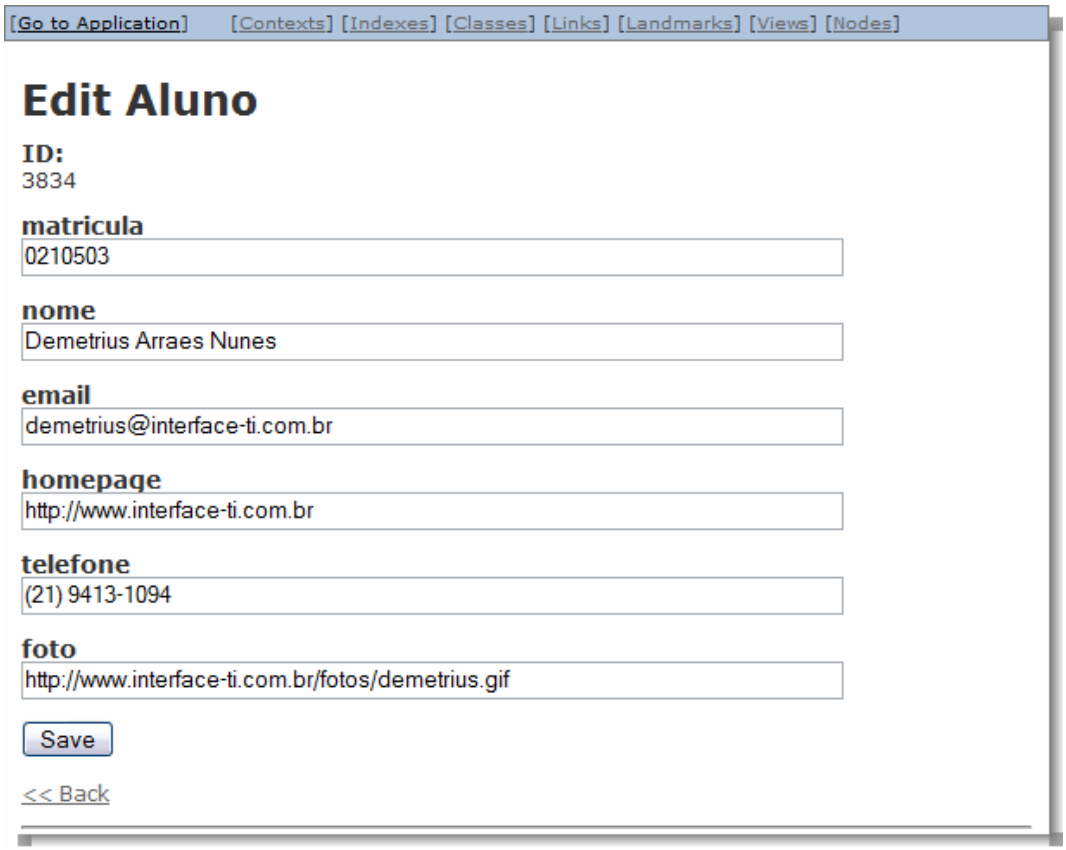


Figura 51 - Tela de edição dos atributos de um nó

A tela de edição de um nó varia conforme a definição de sua classe navegacional. O ambiente constrói dinamicamente a interface de edição baseado nos atributos e elos que compõe sua classe navegacional. Na parte superior da tela de edição são apresentados os atributos, conforme mostra a tela acima. Não existe obrigatoriedade de preenchimento de nenhum atributo. Na parte inferior são apresentadas as listas contendo os valores de elos para este nó.

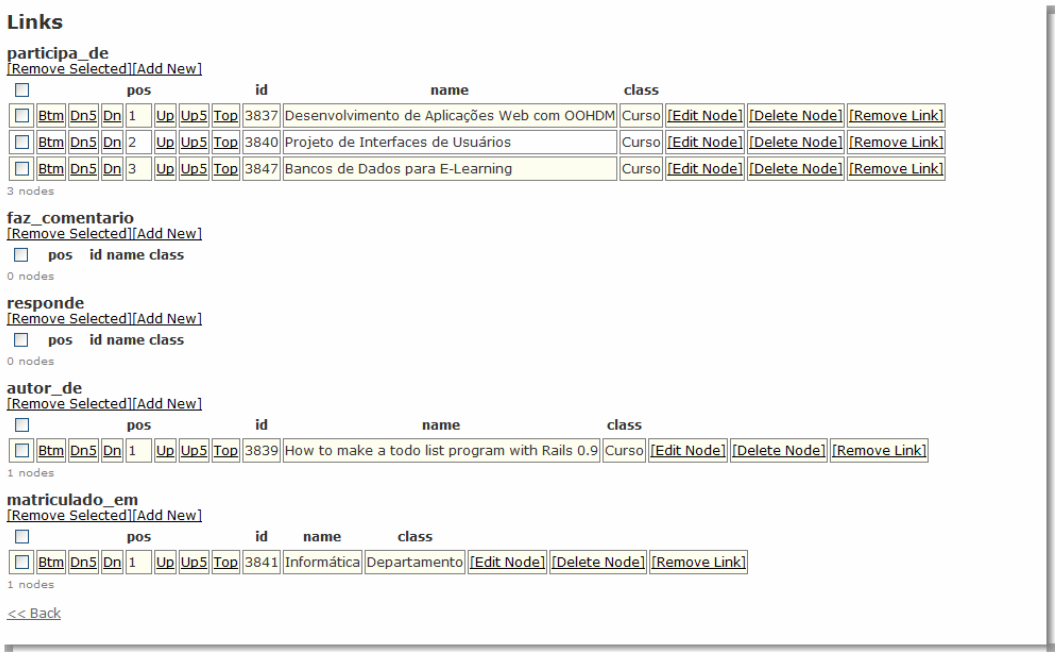


Figura 52 - Tela de edição de nó apresentando as listas de valores dos elos

Cada elo cuja classe navegacional de origem é a classe navegacional do nó editado gera uma listagem que pode ser manipulada através de diversas operações. Podemos excluir valores de elos de duas formas, selecionando os itens desejados para exclusão na listagem e ativando a âncora “Remove Selected”, posicionada acima  ou item a item, através da âncora lateral “Remove Link”.

Podemos editar ou mesmo excluir o nó alvo (neste caso, excluindo o valor de elo como consequência) através das âncoras “Edit Node” e “Delete Node”, respectivamente.

Podemos ainda reposicionar os itens de cada lista através das âncoras “Btm”, “Dn5”, “Dn”, “Up”, “Up5”, “Top” que correspondem aos comandos de posicionar o item como última, descer 5 posições, descer uma posição, subir uma posição, subir 5 posições ou posicionar como o primeiro item da lista, respectivamente.

Finalmente, podemos adicionar novos valores de elos através da âncora “Add New”, posicionada acima de cada listagem.

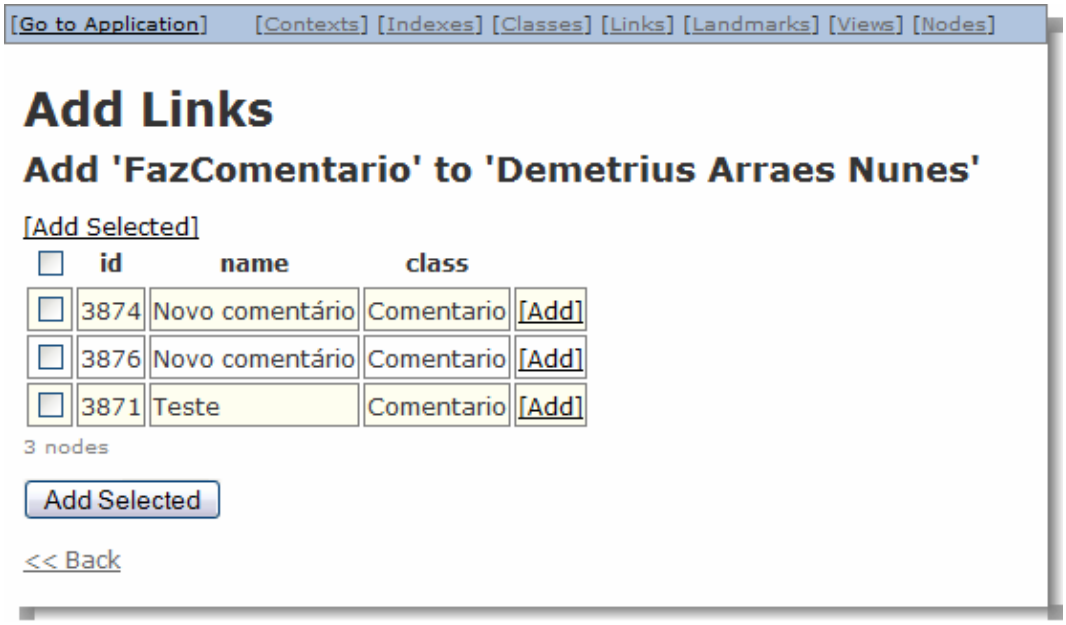


Figura 53 - Tela de adição de valores de elos a um nó

Para adicionarmos um valor de elo a um nó, devemos selecionar os nós a serem ligados ao nó de origem através do elo em uma listagem que é composta de todos os nós da aplicação cuja classe navegacional corresponde à classe navegacional de destino definido para o elo em questão e ainda não foram ligados ao nó de origem através deste elo. Esta seleção pode ser feita em lote, através de marcação e posterior ativação da âncora ou botão “Add Selected” ou um a um, através da âncora lateral “Add”.

### 4.7.1.8. Execução da Aplicação

Os ambientes de desenvolvimento e execução da aplicação desenvolvida no HyperDE estão ligados através da âncora “Go to Application”, que ativa o controlador de navegação e suas ações de exibição de índice e navegação contextual e que está visível apenas nos controladores de retaguarda, e através da âncora “Go to Metamodel”, que ativa os controladores de retaguarda. Ambas ficam localizadas na barra superior da interface Web do ambiente.

O resultado da transição navegacional do metamodelo para a aplicação varia conforme a origem da navegação para o metamodelo. Se o usuário encontrava-se já na aplicação e vai para o metamodelo, o ambiente armazena o ponto de navegação corrente e ao voltar para a aplicação, o usuário retorna ao mesmo ponto onde interrompeu sua navegação. Se o usuário não veio da aplicação e foi

diretamente para o metamodelo, ao navegar para a aplicação será exibido a primeira página da aplicação conforme o algoritmo de definição apresentado previamente.


Figura 54 - Tela de exibição de um índice da aplicação

Toda a área abaixo da barra superior de funções do ambiente, que está sempre visível, é reservada aos templates das visões nativas ou customizadas da aplicação. A tela acima ilustra a exibição de um índice, onde são apresentados os landmarks na parte superior, o rastro de navegação logo abaixo e o índice em si, com seus atributos e entradas na forma de tabela.

Note que as âncoras disponíveis agora na barra superior fornecem apenas a possibilidade de adição de novos nós, de qualquer classe navegacional, ou da navegação para o metamodelo e os controladores de retaguarda.


Figura 55 - Tela de exibição de um nó em contexto navegacional da aplicação

Ao ativarmos a ação de navegação contextual, ganhamos a visão sobre um nó da aplicação de determinada classe navegacional sob um contexto navegacional específico. Acima vemos o exemplo de tal ação utilizando o template nativo do ambiente.

Neste momento, a barra superior oferece âncoras para edição do nó correntemente exibido, exclusão do nó corrente, adição de um nó da mesma classe navegacional do nó corrente e as funções já apresentadas para adição de um nó de qualquer classe navegacional e a navegação para o metamodelo.

### 4.7.2. Interface de Console

Além da interface Web, o ambiente HyperDE oferece uma interface adicional, via console local, onde é possível manipular o metamodelo, modelo e instâncias de uma aplicação OOHDM/SHDM de forma interativa.

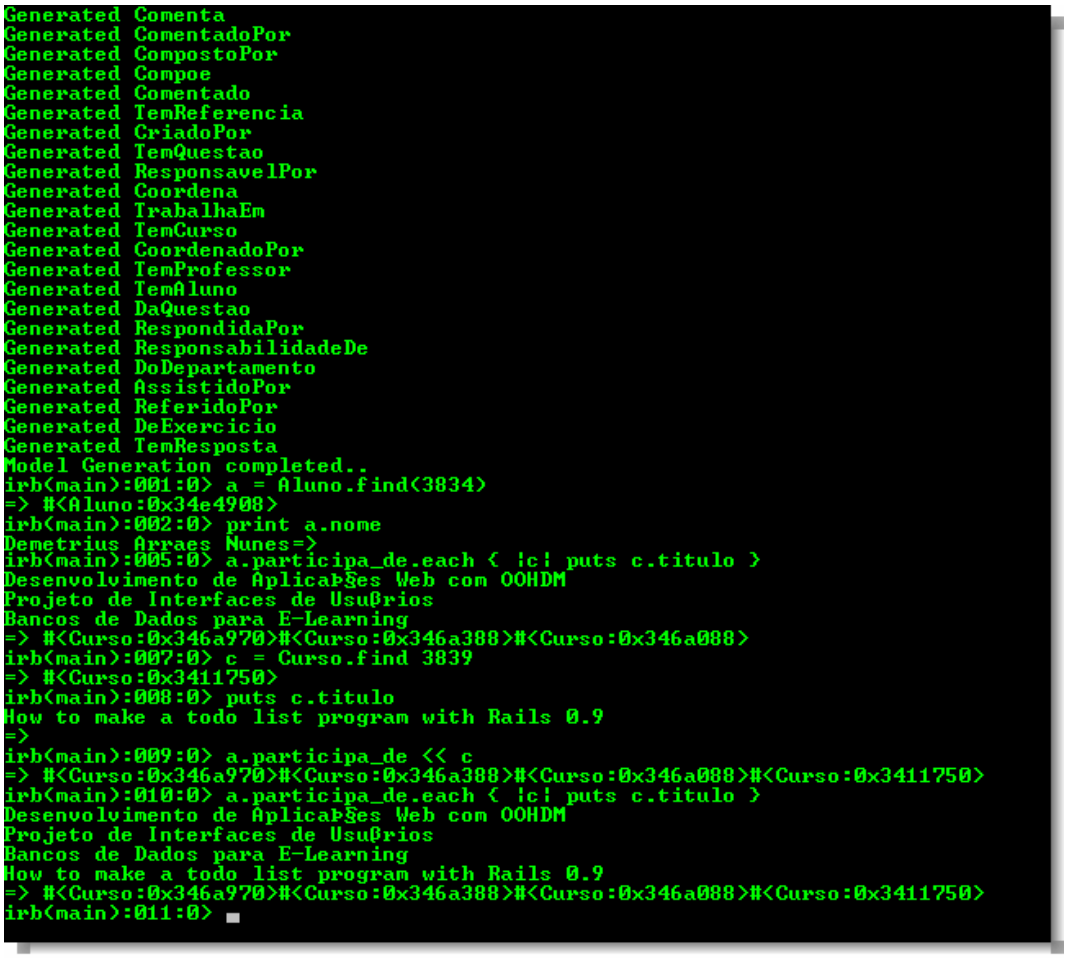


Figura 56 - Tela de uma sessão de uso da interface de console interativo

O console interativo pode ser utilizando apenas localmente e realiza a conexão com a base de dados configurada para o ambiente, gerando todas as classes navegacionais e classes de elos já cadastrados na aplicação de forma automática.

Através desta interface é possível interagir programaticamente com todas as primitivas do ambiente, tornando-se uma ferramenta útil para realizar inspeções, sessões de depuração e experimentações com o modelo da aplicação, obtendo-se um feedback instantâneo para todas as operações executadas.

Mais detalhes da utilização desta interface podem ser encontrados no apêndice relativo a ferramentas do ambiente.

### 4.7.3. Ferramenta para Importação/Exportação de Modelos Navegacionais

As interfaces Web e de console mostram-se bastante úteis na realização de ajustes e refinamentos de um modelo de uma aplicação OOHDM/SHDM já existente. No entanto, por serem interfaces interativas em que cada operação de manipulação do modelo só pode ser feita uma por vez, estas interfaces tornam-se pouco produtivas na criação inicial de um modelo OOHDM/SHDM de complexidade moderada.

Visando preencher esta necessidade, o ambiente HyperDE oferece uma ferramenta para importação e exportação de modelos navegacionais utilizando arquivo em formato textual simples e compacto, facilitando a criação destes modelos através da execução de diversas operações de manipulação em lote.

A seguir podemos ver fragmentos de um arquivo utilizado pela ferramenta para criação de um modelo navegacional:

```
Classes:
  Pessoa:
    attrs: [ nome: String, email: Email, homepage: Url, foto: Image ]
    links: [ [FazComentario: Comentario], [AutorDe: Conteudo, CriadoPor] ]
    label: nome
    anchors:
      conteudos_criados:
        index: ConteudosPorPessoa
        label: Conteúdos Criados
        params: self.id
      comentarios:
        index: ComentariosPorPessoa
        label: Comentários Feitos
        params: self.id
(...)
Contexts:
    CursoPorProfessor:
      query: [ X_by_Y_sorted, Curso, ResponsabilidadeDe, Professor, titulo ]
      params: [ [Professor, Professor] ]
(...)
Indexes:
  CursosPorCategoria:
    context: CursoPorCategoria
    params: [ [ Categoria ] ]
    attrs:
      titulo:
        type: ContextAnchor
        label: self.titulo
(...)
Landmarks:
  Departamentos:
    index: DepartamentosAlfa
    label: "'Departamentos'"
    position: 0
(...)
Nodes:
  prof_schwabe:
    type: Professor
    attrs:
      nome: Daniel Schwabe
      foto: http://www-nt.inf.puc-rio.br/foto/daniel.gif
      email: dschwabe@inf.puc-rio.br
      homepage: http://www-di.inf.puc-rio.br/~schwabe/
      telefone:  3114-1500 Ramal 4356
```

Através deste formato podemos descrever de forma concisa todas as primitivas do modelo navegacional de uma aplicação - classes, contextos, índices, elos, landmarks, como também a descrição de nós neste modelo.

A ferramenta pode ser utilizada tanto para ler um arquivo para importação de dados do modelo, como também para exportar o modelo já existente na base de dados do ambiente para um novo arquivo.

O formato deste arquivo texto e a utilização em detalhes desta ferramenta de importação/exportação podem ser vistos no apêndice de ferramentas do ambiente.