4
O ambiente de desenvolvimento Synth

O Synth é um ambiente de desenvolvimento que dá suporte à construção de aplicações modeladas segundo o método SHDM, fornecendo um conjunto de módulos capazes de receber como entrada os modelos gerados na execução das etapas do método SHDM e produzir como saída uma aplicação hipermídia descrita por estes modelos. O Synth também dispõe de um ambiente de autoria que facilita a inserção e edição destes modelos através de uma interface gráfica de formulários que pode ser executada em qualquer navegador de internet.

4.1.
O Synth e o 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 [Nunes, 2005].

O Synth foi construído tendo como base as experiências obtidas com o HyperDE e pode ser considerado uma evolução do mesmo. Entretanto, o Synth apresenta uma arquitetura de software completamente nova e uma série de características e funcionalidades que o tornam mais alinhado com as mudanças recentes introduzidas no método SHDM, além de resolver alguns problemas encontrados no HyperDE.

Entre os principais problemas encontrados no HyperDE, podemos destacar os seguintes:

- Seu meta modelo de domínio é implementado com uma ontologia própria que, por não ser RDFS nem OWL, dificulta o uso de instâncias de recursos descritos com ontologias conhecidas e disponíveis na *web;*
- Embora empregue o uso de uma base de dados RDF, que é um modelo de dados em grafo, seus dados são manipulados de forma similar ao

modelo relacional, fazendo com que se perca características importantes do modelo RDF;

- Não oferece suporte a modelagem de classes em contexto;
- Não oferece suporte à inferências OWL.

Sobre as mudanças recentes no SHDM suportadas pelo Synth, destacam-se:

- Suporte ao uso de repositórios externos baseados no protocolo SPARQL;
- Suporte a modelagem de atributos navegacionais como extensão do modelo de domínio;
- Suporte ao projeto comportamental de acordo com o modelo proposto por [Santos, 2010];
- Suporte ao projeto de interface de acordo com o proposto por [Luna, 2009].

4.2. Uma arquitetura de software modular

Foi projetada uma arquitetura de software modular [Gauthier & Ponto, 1970; Parnas, 1972], independente de tecnologia de implementação, capaz de oferecer suporte ao desenvolvimento de aplicações modeladas segundo o método SHDM.

Esta arquitetura de software é composta por um conjunto de módulos responsáveis por manter e interpretar os modelos gerados em cada fase do método SHDM.

Cada módulo é composto por uma ontologia em RDFS ou OWL para a representação de seus modelos e um interpretador que dá semântica a esses modelos, além da própria semântica básica das linguagens RDFS e OWL nas quais são representados.

Os módulos trabalham em conjunto, interpretando os seus modelos e comunicando-se entre si, para gerar o ambiente de execução da aplicação de acordo com as definições de cada modelo.

4.2.1.
Visão conceitual da arquitetura

A Figura 13 apresenta a visão conceitual da arquitetura de software modular para aplicações modeladas segundo o método SHDM. As caixas cinzas com cantos arredondados representam os módulos e as caixas brancas representam os componentes de cada módulo. As setas não rotuladas têm o significado definido pela legenda correspondente.

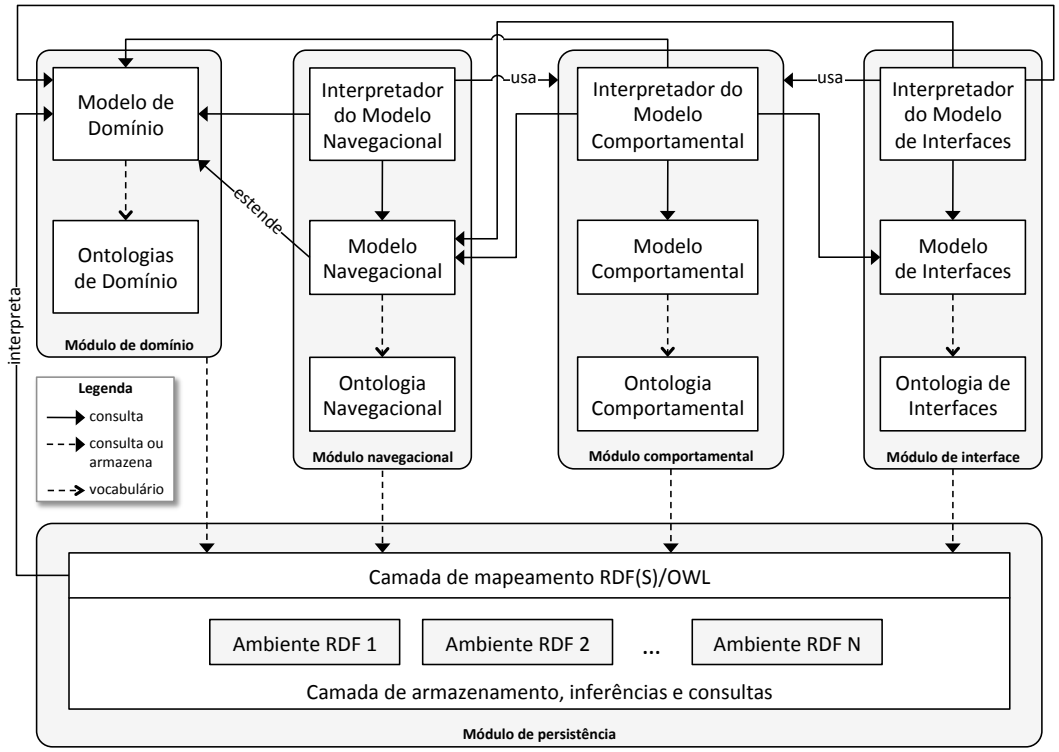


Figura 13 – Visão conceitual da arquitetura

O módulo de persistência trata do acesso e manipulação dos dados da aplicação. Este módulo é composto por duas camadas: uma camada de armazenamento, inferências e consultas e uma camada de mapeamento RDF(S)/OWL.

A camada de armazenamento, inferências e consultas deve fornecer uma interface única para o acesso aos vários ambientes e plataformas de dados RDF existentes no mercado. Pela aplicação do padrão de projeto *Adapter* [Gamma et al., 1995], esta camada converte as interfaces de acesso dos ambientes de dados RDF em uma interface única conhecida pela camada de mapeamento RDF(S)/OWL. Como exemplos de ambientes de dados RDF, podemos citar os

frameworks Sesame[61] e Jena[62] e bases de dados que implementam o protocolo SPARQL[63].

A camada de armazenamento, inferências e consultas também deve ser capaz de distribuir as consultas por vários repositórios de dados, locais ou remotos, combinando seus resultados na mesma resposta. Este requisito visa tornar possível a utilização dos dados distribuídos segundo as recomendações do projeto *Linking Open Data*, principalmente aqueles disponibilizados segundo o protocolo SPARQL.

A camada de mapeamento RDF(S)/OWL fornece uma visão dos dados e meta dados da aplicação, persistidos originalmente em RDF, na forma de primitivas do ambiente de programação utilizado. Em geral, essas primitivas são objetos, atributos e classes de ambientes de programação orientados a objetos baseados em classes. Porém esta arquitetura não obriga que a implementação seja feita em um ambiente de programação específico, podendo-se utilizar ambientes que possuem outras primitivas como vetores associativos [Luckham & Suzuki, 1979] e protótipos [Lieberman, 1986].

Além do módulo de persistência, foram definidos mais quatro outros módulos: módulo de domínio, módulo navegacional, módulo comportamental e módulo de interface. Cada um desses módulos é responsável por manter e interpretar seus modelos de acordo com a fase do método SHDM a qual está relacionado.

O módulo de domínio mantém o modelo de domínio da aplicação, composto pelas ontologias de domínio e instâncias de dados (recursos RDF) da aplicação. Por possuir a mesma semântica do RDF(S) e OWL, o modelo de domínio é interpretado diretamente pela camada de mapeamento RDF(S)/OWL.

O módulo navegacional mantém as declarações sobre o modelo navegacional, descritas de acordo com a ontologia navegacional, e interpreta este modelo, gerando o ambiente de execução da navegação do usuário na aplicação. O interpretador do modelo navegacional consulta o modelo de domínio para obter os conjuntos recursos RDF nos quais se baseiam os nós dos contextos navegacionais e as entradas de índices navegacionais. Os nós e entradas de índices

[61] http://www.openrdf.org
[62] http://jena.sourceforge.net
[63] http://www.w3.org/TR/rdf-sparql-protocol/

são recursos RDF do domínio da aplicação enriquecidos com as primitivas do modelo navegacional pela aplicação do padrão de projeto *Decorator* [Gamma et al., 1995]. Essas primitivas são os atributos navegacionais, como âncoras e estruturas de acesso, que possibilitam que o usuário navegue nos nós da aplicação. O interpretador do modelo navegacional também é composto por um conjunto de instâncias de operações externas do modelo comportamental interpretadas pelo interpretador do modelo comportamental. Nesse sentido, dizemos que ele usa o módulo comportamental. É através das operações externas que o usuário interage com a aplicação navegando em seus nós.

O módulo comportamental mantém as declarações sobre o modelo comportamental, descritas de acordo com a ontologia do modelo comportamental, e interpreta este modelo, gerando o ambiente de execução das regras de negócio e do controle da interação entre os agentes externos e a aplicação. O modelo comportamental é composto por operações internas, que tratam das regras de negócio da aplicação e não podem ser invocadas por agentes externos e por operações externas, que podem ser invocadas por agentes externos [Santos, 2010].

O módulo de interfaces mantém as declarações sobre o modelo de interfaces, descritas de acordo com as ontologia de descrição de interfaces ricas [Luna, 2010] e é responsável pela geração das interfaces da aplicação. O interpretador do modelo de interface acessa o modelo navegacional para compor as interfaces baseadas nos contextos e nos índices navegacionais. Este interpretador também usa o módulo comportamental de forma similar ao interpretador do modelo navegacional.

4.2.2. Colaboração entre os módulos

O diagrama de sequências apresentado na Figura 14 ilustra a sequência de colaboração entre os módulos em um caso normal de navegação na aplicação.

As interações entre o usuário e a aplicação sempre ocorrem através da execução de operações externas do modelo comportamental, disponibilizadas pelo módulo comportamental na forma de *Web Services*. As operações externas são invocadas pelos agentes externos por meio do envio de requisições do protocolo HTTP [Berners-Lee et al., 1999] (*Hypertext Transfer Protocol*) à aplicação. Nesse

sentido, as operações externas cumprem o mesmo papel do controlador do padrão de arquitetura de software MVC [Burbeck, 1987, 1992] (modelo-visão-controlador), coordenando as interações do usuário, obtendo os dados do modelo de domínio, acionando a visão para gerar a interface e, finalmente, entregando o resultado ao usuário ou agente externo.

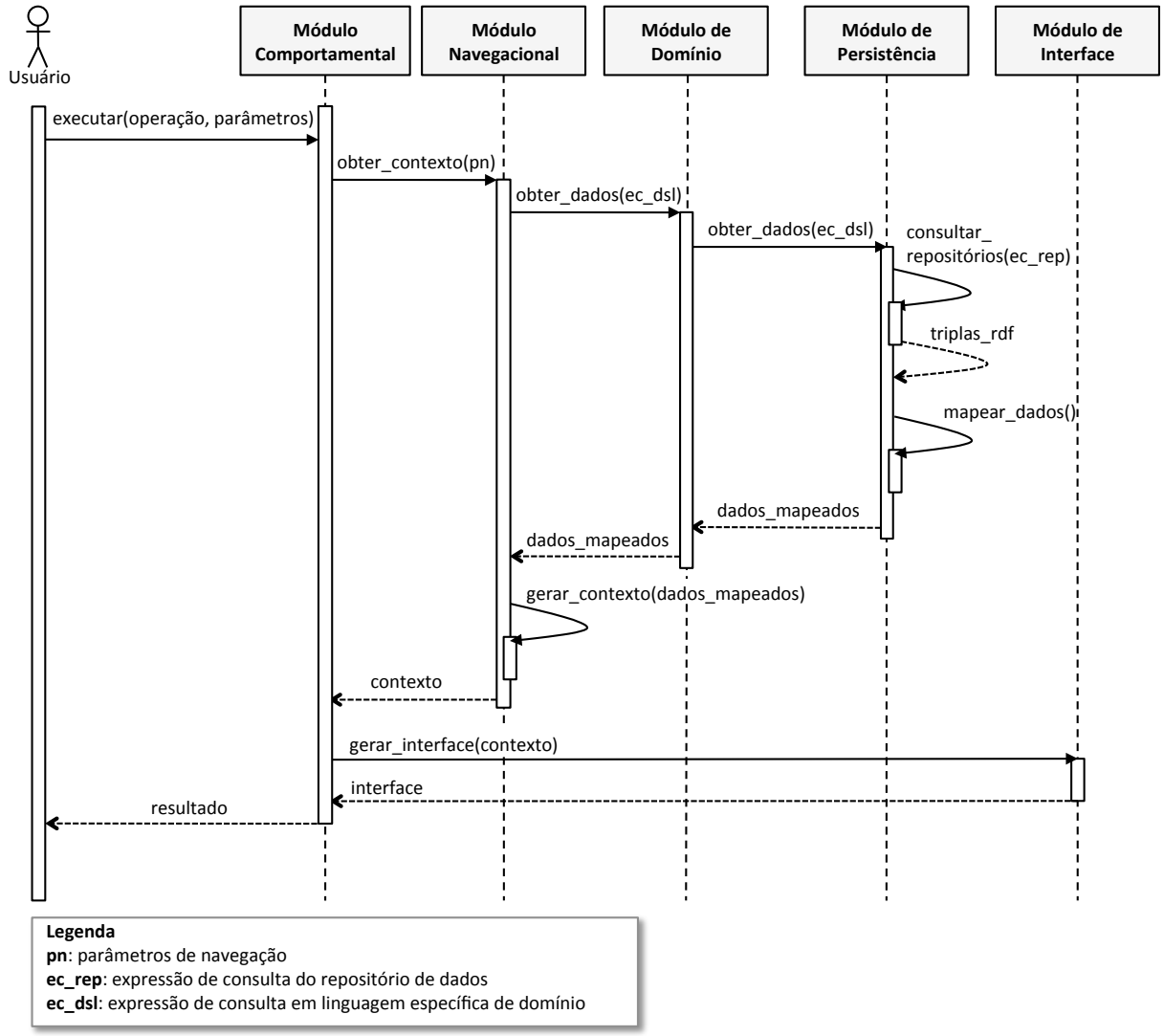


Figura 14 – Colaboração entre os módulos

A seguir descrevemos passo-a-passo a sequência de colaboração ilustrada na Figura 14:

Passo 1: executar(operação, parâmetros) – O usuário aciona o método "executar" do módulo comportamental, informando o nome da operação externa a ser executada e um conjunto de parâmetros. Em seguida, o módulo

comportamental invoca a execução da operação externa solicitada, repassando os parâmetros que foram informados na chamada anterior.

Passo 2: obter_contexto(pn) – Depois, a operação externa de navegação invoca o método "obter_contexto" do módulo navegacional, passando um conjunto de parâmetros de navegação, identificado na figura como "pn", para obtenção desse contexto. Entre os parâmetros de navegação estão o nome do contexto a ser usado, o identificador do nó navegacional a ser acessado e outros parâmetros que possam ser requeridos pelo contexto, no caso de contextos parametrizados.

Passo 3: obter_dados(ec_dsl) – Em seguida, o método "obter_contexto" do módulo navegacional obtém a descrição do contexto, cujo nome foi informado na chamada anterior, e invoca o método "obter_dados" do módulo de domínio, passando como parâmetro uma expressão de consulta expressa em uma linguagem específica de domínio (DSL), declarada no contexto obtido.

Passo 4: obter_dados(ec_dsl) (Módulo de persistência) – O módulo de domínio apenas repassa a chamada acionando o método de mesmo nome ("obter_dados"), com os mesmos parâmetros, para a camada de mapeamento RDF(S)/OWL do módulo de persistência, que transforma a expressão de consulta expressa na linguagem específica de domínio em uma expressão de consulta compatível com a base de dados utilizada. Em geral, expressão de consulta final é expressa na linguagem SPARQL.

Passo 5: consultar_repositórios(ec_rep) – A camada de armazenamento, inferências e consultas do módulo de persistência executa a consulta em todos os repositórios de dados da aplicação, consolida seus resultados em um conjunto de triplas RDF e retorna esse conjunto para a camada de mapeamento RDF(S)/OWL.

Passo 6: mapear_dados() – Em seguida, a camada de mapeamento RDF(S)/OWL mapeia as triplas RDF em primitivas do ambiente de programação que representam recursos RDF. O módulo de persistência retorna esses dados mapeados para o módulo de domínio, que repassa os mesmos dados de volta para o método "obter_contexto".

Passo 7: gerar_contexto(dados_mapeados) – O método "obter_contexto" aciona o método "gerar_contexto", passando como parâmetro os dados mapeados. O método "gerar_contexto" gera a estrutura de dados de um contexto baseada nos dados mapeados obtidos do módulo de domínio e retorna esta estrutura para o método "obter_contexto" que repassa para a operação externa que a acionou.

Passo 8: gerar_interface(contexto) – Em seguida, a operação externa invoca o método "gerar_interface" do módulo de interface repassando como parâmetro o contexto obtido. O método "gerar_interface" retorna a interface resultante da combinação dos dados do contexto com as regras de formatação definidas pelo modelo de interface.

Finalmente, a operação externa devolve para o usuário o resultado final da operação de navegação.

A Figura 15 apresenta um caso concreto da sequência de colaboração entre os módulos. Nesta ilustração são utilizados valores de parâmetros semelhantes a valores reais.

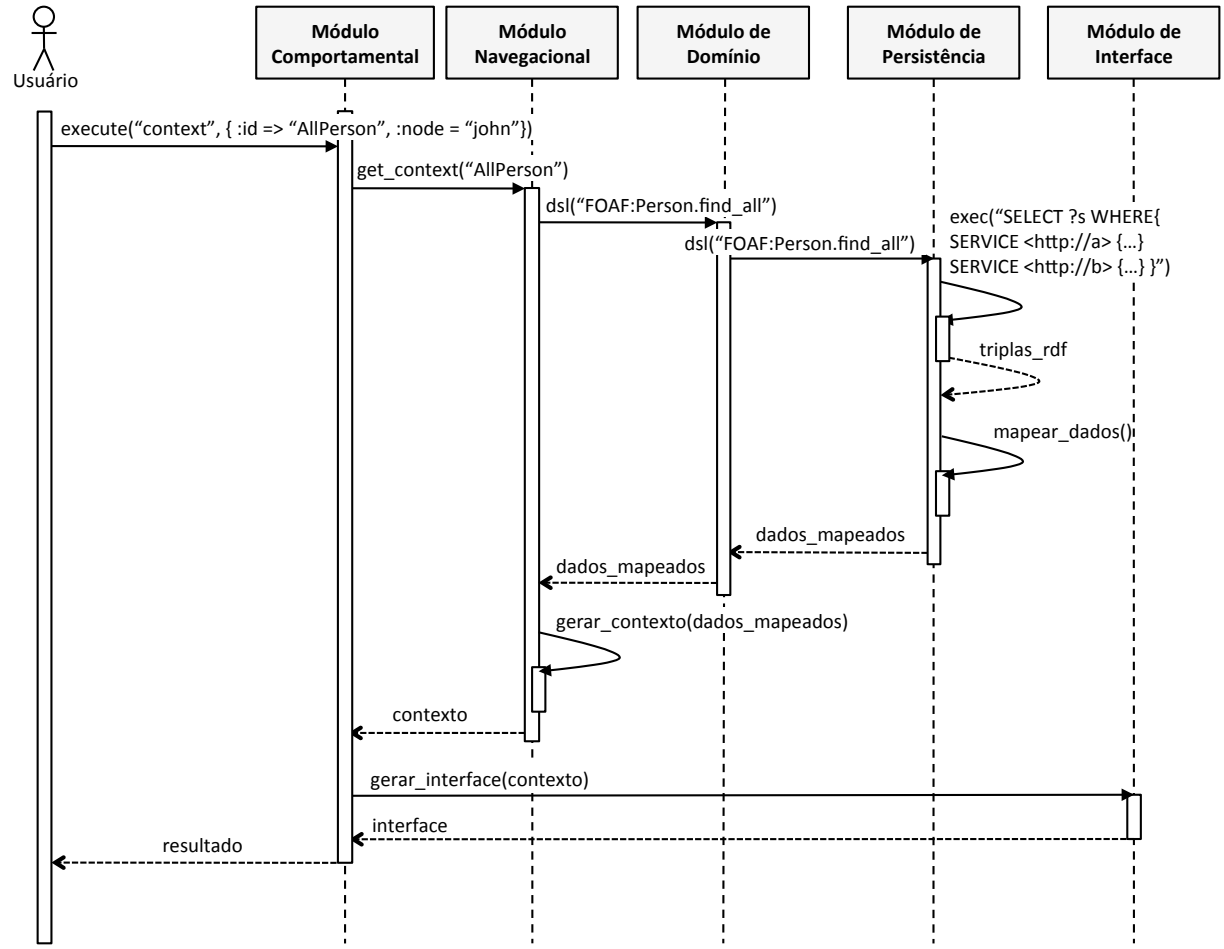


Figura 15 – Colaboração entre módulos em um caso concreto

4.3. Arquitetura de implementação

O Synth foi implementado sobre o Ruby on Rails, um *framework* MVC para o desenvolvimento de aplicações *web*. Sob esse aspecto, o Synth é uma aplicação *web* desenvolvida segundo o paradigma MVC. Contudo, foi mantida uma organização de software modular, onde cada módulo descrito na seção 4.2 é implementado como uma composição de um ou mais componentes do paradigma MVC.

A Figura 16 apresenta a pilha de componentes de software que compõe a arquitetura geral de implementação do Synth.

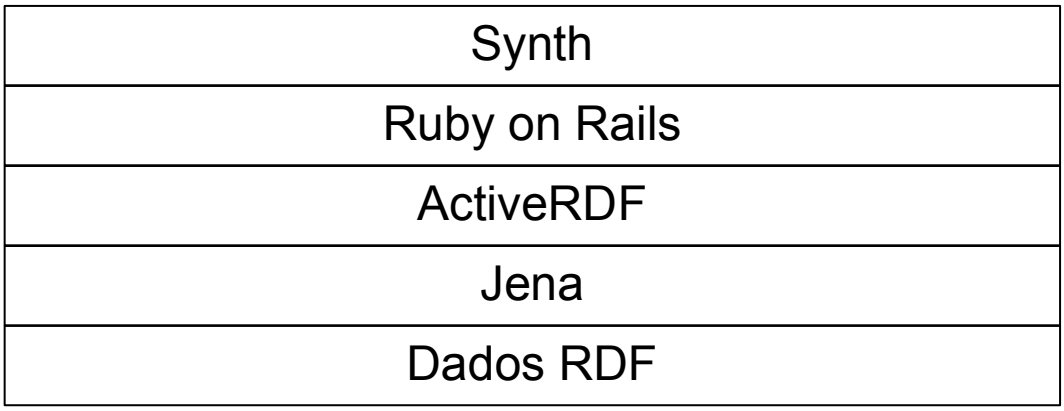


Figura 16 – Arquitetura de implementação Synth

4.3.1. Dados RDF

Todos os artefatos gerados durante a modelagem de uma aplicação SHDM são persistidos em uma base de dados RDF de acordo com uma ontologia de meta modelo desenvolvida para expressar conceitos como classes, propriedades e relacionamentos de cada artefato a ser descrito. Nesta mesma base de dados estão persistidas os ontologias de domínio e as instâncias (recursos) de dados do domínio da aplicação. A base de dados RDF do Synth não faz distinção entre dados e meta dados da aplicação.

O Synth ainda é capaz de utilizar dados de domínio da aplicação disponíveis em base de dados RDF externas que implementam o protocolo SPARQL. Entretanto, os meta dados sobre os modelos da aplicação ficam todos na base de dados RDF local.

4.3.2. Jena

Jena[64] é um *framework* Java para o desenvolvimento de aplicações na Web Semântica que provê um ambiente programático para manipulação de RDF, RDFS e OWL, SPARQL e inclui uma máquina de inferências baseada em regras. O Jena é um projeto de código aberto nascido no HP Labs Semantic Web

[64] http://jena.sourceforge.net

Programme[65] e, recentemente, foi aceito pela incubadora[66] de projetos da Apache Foundation[67].

O *framework* Jena inclui:

- Uma API RDF;
- Leitura e escrita RDF nas notações RDF/XML, N3, N-Triples e TURTLE;
- Uma API OWL;
- Armazenamento em memória e persistente (arquivos, base de dados nativa e base de dados relacionais);
- Uma máquina de consulta SPARQL.

Na arquitetura conceitual apresentada na seção 4.2.1, o *framework* Jena cumpre o papel do ambiente RDF que compõe a camada de armazenamento, inferências e consultas.

A base de dados RDF local do Synth pode ser persistida em quaisquer das opções de armazenamento do Jena, porém a base de dados *default* usa o modo de armazenamento em sistema de arquivos.

Há outros *frameworks* RDF disponíveis no mercado, sendo que, entre os critérios de escolha, foi dada prioridade aos *frameworks* gratuitos que pudessem ser distribuídos junto com o Synth. Além do Jena, podemos citar como opções o Sesame e o RedLand[68]. A escolha pelo Jena se deu pelo fato do mesmo possuir funcionalidades que atendiam requisitos importantes para o método SHDM:

- Suporte às inferências básicas da linguagem OWL, possibilitando o uso de modelos de domínio mais expressivos. No caso do SHDM, a única inferência mandatória é por resultados decorrentes da aplicação da relação owl:inverseOf entre propriedades;
- Suporte à linguagem de consulta SPARQL 1.1, sucessor da SPARQL 1.0[69] que é recomendação do W3C, especificamente o suporte às consultas em bases de dados federadas, que permite a distribuição da

[65] http://www.hpl.hp.com/semweb
[66] http://incubator.apache.org/
[67] http://www.apache.org
[68] http://librdf.org
[69] http://www.w3.org/TR/rdf-sparql-query

consulta entre vários SPARQL *Endpoints* distribuídos na *web*, cumprindo o requisito de se poder utilizar instâncias de dados de domínio distribuídas.

4.3.3. ActiveRDF

O ActiveRDF[70] é um *framework* para acesso a dados RDF a partir de programas escritos em linguagem Ruby. Pode ser usado como a camada de dados de aplicações escritas sobre o *framework* Ruby on Rails, de modo similar à biblioteca ActiveRecord (que provê o mapeamento objeto-relacional para bases de dados relacionais). A combinação do ActiveRDF com o Ruby on Rails possibilita rápida criação de aplicações na Web Semântica.

O ActiveRDF oferece uma linguagem específica de domínio (DSL) para o modelo RDF dando acesso a recursos RDF, classes e propriedades de modo programático. O Quadro 12 mostra um exemplo de manipulação de recursos RDF através da DSL do ActiveRDF.

```
john = RDFS::Resource.new("http://example.com/john")

print john.foaf::name
John

print john.foaf::age
30

john.foaf::age = 31
31

print john.foaf::age
31
```

Quadro 12 – Exemplo de manipulação de recursos RDF com a DSL do ActiveRDF

O Quadro 13 mostra alguns exemplos das facilidades de consulta oferecidas pela DSL do ActiveRDF.

```
all_people = FOAF::Person.find_all #todos os recursos do tipo foaf:Person

john = FOAF::Person.find_by.foaf::name("john").execute

brazil = Resource.new("http://dbpedia.org/resource/Brazil")

people_from_brazil= FOAF::Person.find_by.foaf::based_near(brazil).execute
```

Quadro 13 – Exemplos de consultas com a DSL do ActiveRDF

[70] http://www.activerdf.org

O ActiveRDF pode ser usado com vários ambientes RDF do mercado como o Jena, Sesame e o Redland. Adaptadores para outros ambientes RDF podem ser adicionados facilmente. O ActiveRDF usa a mesma filosofia de convenção sobre configuração muito difundida pelo *framework* Ruby on Rails e é um projeto de código aberto e gratuito.

Na arquitetura conceitual apresentada na seção 4.2.1, o ActiveRDF cumpre o papel do módulo de persistência, provendo as funções das camadas de mapeamento RDF(S)/OWL e de armazenamento, inferências e consultas.

Os critérios utilizados para a escolha pelo ActiveRDF foram os mesmos utilizados para o *framework* Jena. Contudo, no inicio do desenvolvimento do Synth, o ActiveRDF era a única opção disponível para a linguagem Ruby que podia ser considerada madura o suficiente para ser utilizada. Além disso, havia a experiência de sucesso com a ferramenta Explorator [Araújo, 2009] como fator importante na decisão.

Hoje há pelo menos um projeto disponível que possui maturidade suficiente para ser considerado uma opção ao ActiveRDF, o RDF.rb[71]. Como o desenvolvimento do RDF.rb tem se mostrado mais ativo que o do ActiveRDF, é possível que no futuro ele seja incorporado ao Synth.

4.3.4. Ruby on Rails

O Rails é um *framework* de desenvolvimento *web* escrito na linguagem Ruby. Ele foi projetado para tornar a programação de aplicações web mais fácil, fazendo várias suposições sobre o que cada desenvolvedor precisa para começar. A filosofia Rails inclui diversos princípios guia:

• DRY – "Don't Repeat Yourself" – sugere que escrever o mesmo código várias vezes é uma coisa ruim;

• Convenção ao invés de Configuração – significa que o Rails faz suposições sobre o que o desenvolvedor quer fazer e como será feito;

• REST – organiza a aplicação em torno de recursos e verbos HTTP padrão.

[71] http://rdf.rubyforge.org

O Rails é organizado segundo o padrão de arquitetura modelo-visão-controlador, normalmente chamada apenas de MVC. Os benefícios do MVC incluem:

- Isolamento entre a lógica de negócios e a interface de usuário;
- Facilidade de manter o código DRY;
- Manter claro onde tipos de código diferentes devem ficar para facilitar a manutenção.

A escolha pelo Rails para o desenvolvimento do Synth teve as mesmas razões que motivaram a sua escolha para o HyperDE. Destaca-se o fato do Rails ser escrito em Ruby, uma linguagem dinâmica, que nos permitiu utilizar várias facilidades de meta programação não encontradas em outras linguagens.

O Rails é um projeto de código aberto com aproximadamente sete anos de idade, considerado bastante maduro entre as opções disponíveis no mercado. Além disso existe uma grande quantidade de bibliotecas disponíveis para uso em conjunto com o Rails. Isso nos proporcionou maior 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 infraestrutura do ambiente.

4.3.4.1. A linguagem Ruby

Conforme demonstrado por [Orena et al., 2008], dado a natureza dinâmica e aberta da Web Semântica, o uso linguagens de "scripting" dinâmicas parece ser mais adequado para o desenvolvimento sobre esse ambiente, graças aos seus objetos dinamicamente tipados, herança múltipla e relaxamento das restrições de conformidade das definições de classe que apresentam um bom mapeamento para a especificação de classes RDF(S). Linguagens como Smaltalk, Lua, Perl, Python, e Ruby são dinamicamente tipadas, geralmente suportam construções de alta ordem (por exemplo, passagem de métodos como argumento para métodos) e suporte completo a reflexão, introspeção e *intercession* (alteração dos objetos em tempo de execução), todas características desejáveis para o desenvolvimento de aplicações na Web Semântica.

> Ruby é uma linguagem de programação interpretada multiparadigma, de tipagem dinâmica e forte, com gerenciamento de memória automático, originalmente planejada e desenvolvida no Japão em 1995, por Yukihiro Matsumoto, para ser usada como linguagem de script. Ruby suporta programação funcional, orientada a objetos, imperativa e reflexiva.
> Uma série de características foram definidas para atender às propostas do Ruby:
> - Todas as variáveis são objetos, onde até os "tipos primitivos" (tais como inteiro, real, entre outros) são objetos;
> - Métodos de geração de código em tempo real, como os "attribute accessors";
> - Através do pacote *RubyGems*, é possível instalar e atualizar bibliotecas com uma linha de comando, de maneira similar ao APT do Debian Linux;
> - *Code blocks* (blocos de código) passados como parâmetros para métodos; permite a criação de *closures*;
> - *Mixins*, uma forma de emular a herança múltipla;
> - Tipagem dinâmica, mas forte. Isso significa que todas as variáveis devem ter um tipo, mas a classe pode ser alterada dinamicamente;
> - Ruby está disponível para diversas plataformas, como Microsoft Windows, Linux, Solaris e Mac OS X, além de também ser executável sobre a máquina virtual Java (através do JRuby) e haver um projeto para ser executável sobre a máquina virtual Microsoft .NET, o IronRuby.
>
> Wikipedia

Ruby foi escolhida como linguagem de programação para construção do Synth pelos mesmos motivos que foi escolhida para o HyperDE, em função dos seguintes aspectos técnicos:

- Por ser linguagem genuinamente orientada a objetos, Ruby permite um mapeamento de baixa impedância entre as classes RDF para as classes em linguagem de programação;
- 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 como 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 Synth a definição e redefinição de classes e métodos durante o uso do ambiente, com esforço mínimo de implementação, sem necessidade de fase de 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 Synth necessita utilizar linguagem de programação.

4.3.4.2. JRuby

JRuby é a implementação da linguagem Ruby para a plataforma Java[72]. Com JRuby é possível executar qualquer programa escrito em linguagem Ruby, ou uma aplicação Ruby on Rails a partir da JVM (*Java Virtual Machine*) e, dessa maneira, tirar proveito de todos recursos oferecidos pela plataforma Java como *multithreading* real, alto desempenho e uma vasta gama de bibliotecas e servidores de aplicação como JBoss[73], GlassFish[74] entre outros.

A principal motivação para a escolha do JRuby foi a necessidade de acesso às bibliotecas do *framework* Jena que é escrito em Java. O uso do JRuby tornou possível o acesso direto às bibliotecas do Jena a partir das classes Ruby do Synth. O HyperDE utiliza a biblioteca RJB[75] (*Ruby Java Bridge*) para o acesso às bibliotecas do Sesame, também escrita em Java, a partir do interpretador padrão do Ruby, escrito em linguagem C. Embora fosse possível utilizar o RJB no Synth, a escolha pelo JRuby se mostrou mais adequada devido ao melhor desempenho e a simplicidade do uso das bibliotecas Java, que para o programador Ruby, são acessadas como qualquer biblioteca Ruby.

4.3.5. Synth

A organização dos módulos do Synth segue a arquitetura conceitual descrita na seção 4.2. Sendo assim, cada módulo do Synth fornece funcionalidades de manipulação, persistência e semântica para os modelos que manipula.

No Synth, um modelo é um conjunto de recursos RDF descritos de acordo com as ontologias criadas para descrevê-lo. Para fornecer funcionalidades a estes modelos, o Synth possui, em cada módulo, uma classe Ruby para cada classe RDF

[72] http://www.java.com
[73] http://www.jboss.com
[74] https://glassfish.dev.java.net
[75] http://rjb.rubyforge.org

das ontologias que usa, responsável por dar comportamento e semântica aos recursos dessas classes. As classes Ruby possuem o mesmo nome das classes RDF para as quais dão comportamento. Por exemplo, para a classe RDF shdm:Context, o módulo navegacional possui a classe Ruby SHDM::Context.

Os módulos do Synth também podem possuir classes Ruby que não são relacionadas a nenhuma classe RDF. Essas são as classes que compõe os interpretadores dos modelos e complementam a semântica dos mesmos.

As seções a seguir apresentam em detalhes cada módulo do Synth.

4.3.5.1. Módulo de domínio

A Figura 17 apresenta o diagrama de classes Ruby que compõem o módulo de domínio.

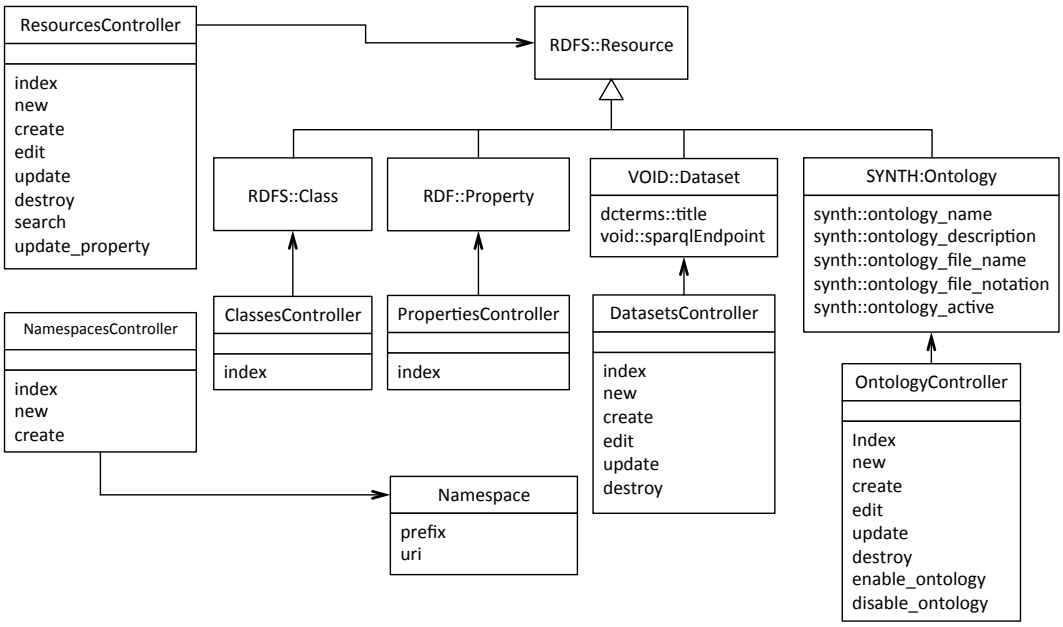


Figura 17 – Diagrama de classes do módulo de domínio

A classe RDFS::Resource representa qualquer recurso RDF. Esta classe é fornecida originalmente pela biblioteca ActiveRDF e possui as funcionalidades de persistência e consultas na base de dados RDF e manipulação dos valores das propriedades dos recursos. Todas as outras classes persistidas na base de dados RDF são subclasses de RDFS::Resource e herdam suas funcionalidades.

As classes RDFS::Class e RDF::Property representam, respectivamente, as classes e propriedades RDF e também são fornecidas pela biblioteca ActiveRDF.

Junto com a classe RDFS::Resource, as classes RDFS::Class e RDF::Property fornecem a semântica básica do modelo RDF e, consequentemente, a semântica para qualquer modelo de domínio descrito em RDFS e OWL.

A classe VOID::Dataset é parte do vocabulário voiD (Vocabulary of Interlinked Datasets)[76] e, segundo o voiD, representa um conjunto de triplas que são publicadas, mantidas ou agregadas por um único provedor. Para o Synth, são os repositórios de dados externos à sua base de dados local. Os valores da propriedade dcterms::title são utilizados nas expressões de consultas dos contextos para indicar que as consultas devem ser feitas nos SPARQL *endpoints* cujos endereços são indicados no valor de void::sparqlEndpoint. O uso dos recursos do tipo VOID::Dataset nas expressões de consultas dos contextos será visto na seção 4.4.3.1.1.

A classe SYNTH::Ontology representa as ontologias de domínio utilizadas pela aplicação. Seu principal atributo é o synth::ontology_file_name que indica um arquivo de texto local, em uma notação RDF válida, contendo as asserções, em RDFS ou OWL, sobre a ontologia em questão.

A classe Namespace é a única classe de modelo do módulo de domínio que não é subclasse de RDFS::Resource, pois não é persistida na base de dados como um recurso RDF. Esta classe é fornecida pelo ActiveRDF e representa os *namespaces* utilizados pela aplicação.

As classes cujos nomes são terminados em "Controller", são controladores do Ruby on Rails que compõem o ambiente de autoria do Synth. Cada módulo possui seus próprios controladores para o ambiente de autoria, que são responsáveis pelas ações de manipulação das instâncias (ou recursos) das classes relacionadas através de uma interface *web*. Em todos os controladores, os métodos "index", "new", "create", "edit", "update" e "destroy" representam as operações CRUD (Create, Read, Update e Delete) [Kilov, 1990] da classe relacionada ao controlador.

Na classe ResourcesController, o método "search" executa a busca por um recurso na base de dados por parte do valor da propriedade rdfs:label e o método "update_property" é responsável pela atualização do valor de uma propriedade de um recurso.

[76] http://vocab.deri.ie/void

As classes ClassesController e PropertiesController possuem apenas o método "index" que lista, respectivamente, as classes e as propriedades RDF. A manipulação dos recursos do tipo rdfs:Class e rdf:Property, subclasses de rdfs:Resource, são feitas através do controlador ResourcesController cujos métodos generalizam a manipulação de qualquer recurso RDF.

Os outros controladores possuem seus próprios métodos CRUD, pois fornecem formulários especializados para cada instância.

O controlador OntologyController ainda possui os métodos "enable_ontology" e "disable_ontology" responsáveis por habilitar e desabilitar as inferências das ontologias.

A seção 4.4.2 apresenta a interface gráfica do ambiente de autoria para a manipulação do modelo de domínio que faz uso dos controladores do módulo de domínio.

4.3.5.2. Módulo navegacional

A Figura 18 apresenta o diagrama de classes Ruby do módulo navegacional. Neste diagrama foram omitidas as classes de controladores que serão abordadas mais adiante.

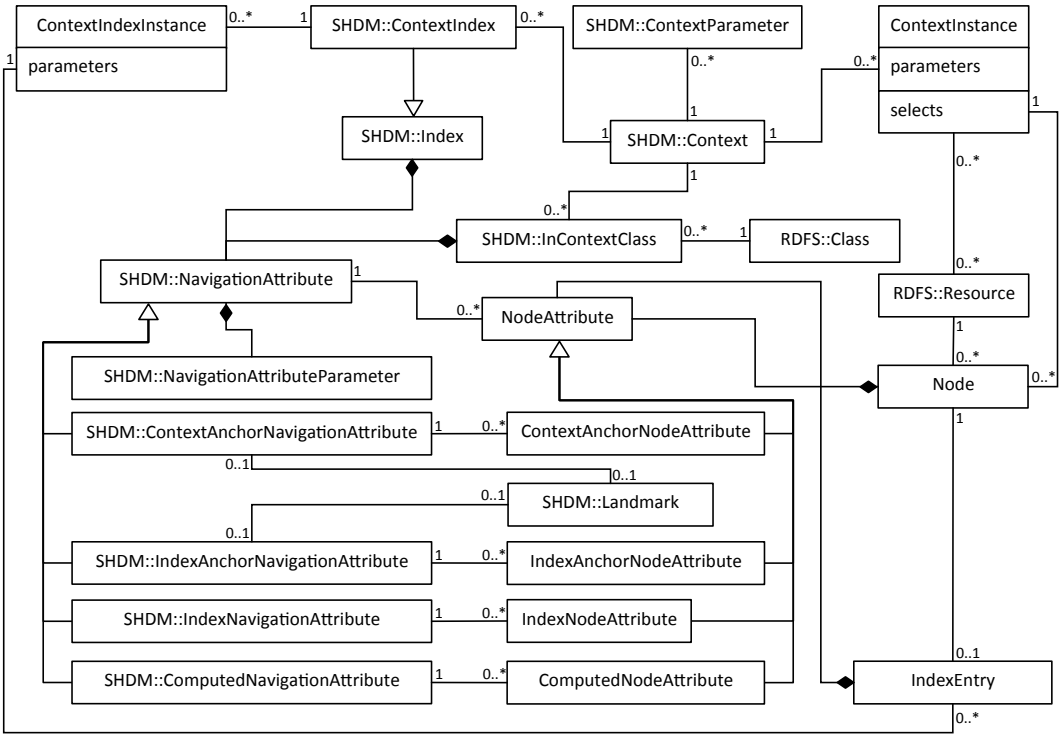


Figura 18 – Diagrama de classes do módulo navegacional

Para facilitar o entendimento, as classes apresentadas no diagrama acima serão descritas separadamente em dois grupos: classes de **tempo de modelagem** e classes de **tempo de execução**.

As classes de tempo de modelagem representam as primitivas do modelo navegacional persistidas na base de dados RDF. No diagrama acima, são as classes cujos nomes são iniciados pelo prefixo SHDM::. Essas classes conhecem apenas as informações estáticas do modelo navegacional e não têm acesso as informações obtidas durante a navegação.

As classes de tempo de execução dão semântica à navegação, instanciando os objetos navegacionais (contextos, índices, nós, entradas de índices e atributos) em tempo de execução de acordo com a interação do usuário com a aplicação. Estas classes compõem o **interpretador do modelo navegacional** descrito na seção 4.2.1.

A seguir, serão apresentadas em detalhes, as classes do módulo navegacional agrupadas por interesse.

4.3.5.2.1. Contextos

A Figura 19 apresenta as classes do módulo navegacional relacionadas com os contextos navegacionais.

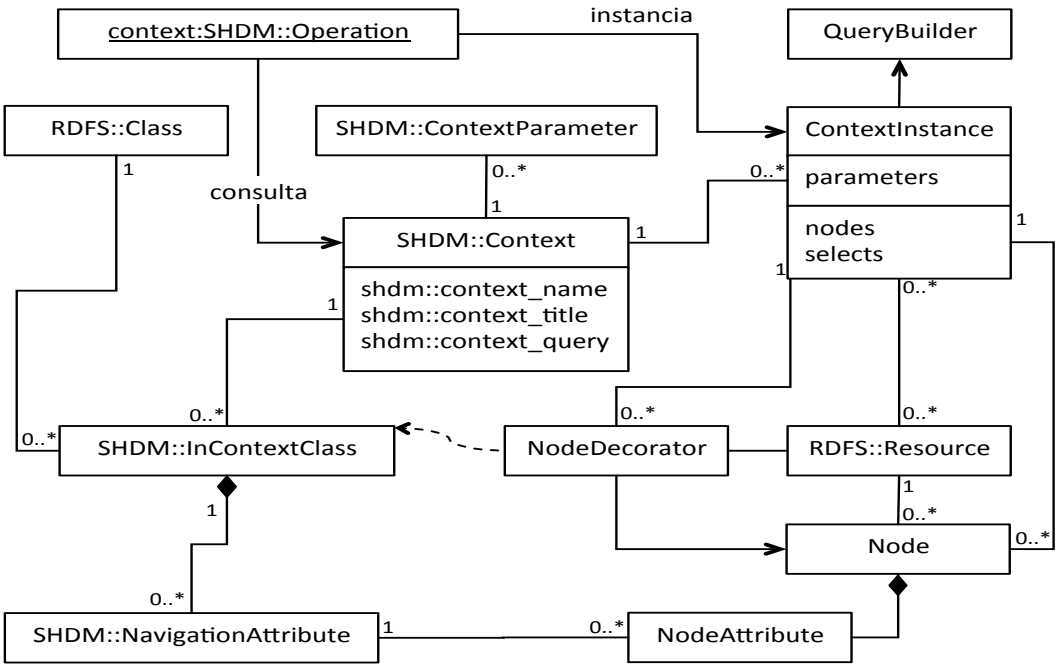


Figura 19 – Classes envolvidas na instanciação de um contexto navegacional

A classe SHDM::Context representa os contextos navegacionais em tempo de modelagem e guarda as informações básicas sobre um contexto, similares às informações descritas em um cartão de contexto [Rossi, 1996]. O principal atributo da classe SHDM::Context é o shdm::context_query, que mantém a expressão de consulta cuja execução gera o conjunto de nós navegacionais do contexto em questão.

A classe SHDM::ContextParameter representa os parâmetros esperados por um contexto.

A classe SHDM::InContextClass representa a relação de uma classe de domínio com um contexto. O objetivo desta classe é enriquecer os nós de uma classe acessados a partir de um determinado contexto com atributos navegacionais representados pelos recursos da classe SHDM::NavigationAttributes que participam de sua composição. A classe SHDM::NavigationAttributes será vista em detalhes mais adiante.

A classe ContextInstance representa um contexto em tempo de execução. Esta classe é necessária porque um mesmo contexto pode ser instanciado com parâmetros diferentes, gerando um conjunto de nós diferente em função dos valores dos parâmetros. Uma instância de ContextInstance está sempre relacionada a uma instância de SHDM::Context e possui um vetor associativo contendo um conjunto de pares parâmetro/valor no atributo “parameters”. Esses parâmetros são informados pelo usuário durante a navegação.

O conjunto de nós navegacionais do contexto são gerados pela instância de ContextInstance na execução do método “nodes”. O método “nodes” executa a expressão de consulta de SHDM::Context sobre a base de dados, obtém os recursos RDF e delega a criação de cada nó navegacional para NodeDecorator. A classe NodeDecorator gera uma instância da classe Node, que é o resultado a aplicação do padrão de projeto *Decorator* sobre uma instância de RDFS::Resource. As instâncias de Node representam recursos RDF decorados com os atributos navegacionais, em tempo de execução, descritos pelas instâncias de SHDM::NavigationAttribute que compõem as instâncias de SHDM::InContextClass que relacionam o contexto acessado com as classes RDF do recurso decorado.

A classe QueryBuilder transforma as expressões de consulta de contexto na linguagem SynthQL em expressões SPARQL. A linguagem SynthQL e as outras linguagens de consultas de contextos serão vistas em detalhes na seção 4.4.3.1.1.

A operação externa "context", instância da classe SHDM::Operation, possui toda a lógica de navegação no contexto. Esta operação é invocada pelo usuário e recebe como parâmetros o identificador do contexto a ser navegado, o identificador de um nó que participa do contexto e um conjunto de outros parâmetros esperados pelo contexto. A operação "context" deve encontrar a descrição do contexto, instância de SHDM::Context, na base de dados e instanciar ContextInstance com os parâmetros recebidos do usuário e depois invocar o método "nodes" de ContextInstance para obter os nós navegacionais.

O diagrama de sequência apresentado na Figura 20, ilustra a sequência de passos para a instanciação de um contexto e geração dos seus nós navegacionais.

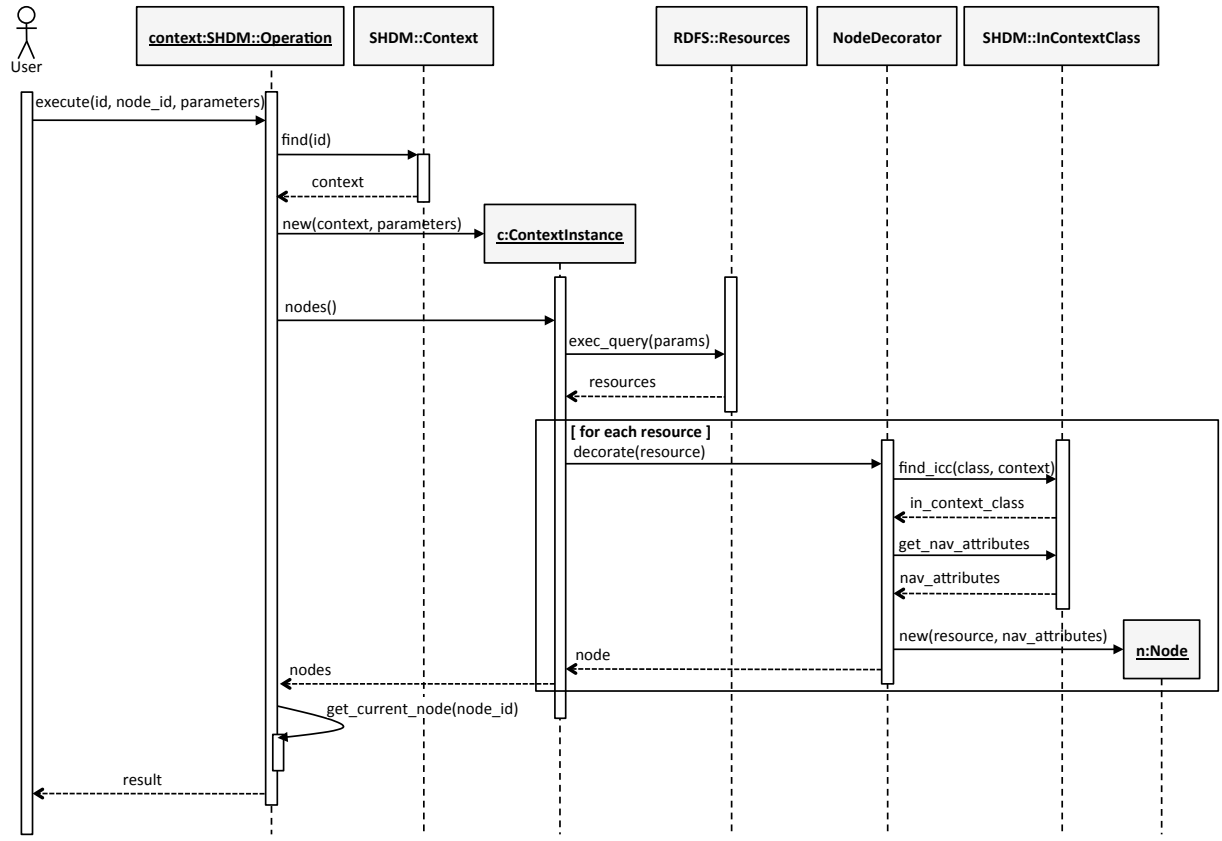


Figura 20 – Sequência de passos para a instanciação de um contexto

4.3.5.2.2. Índices

A Figura 21 apresenta as classes do módulo navegacional relacionadas com os índices navegacionais.

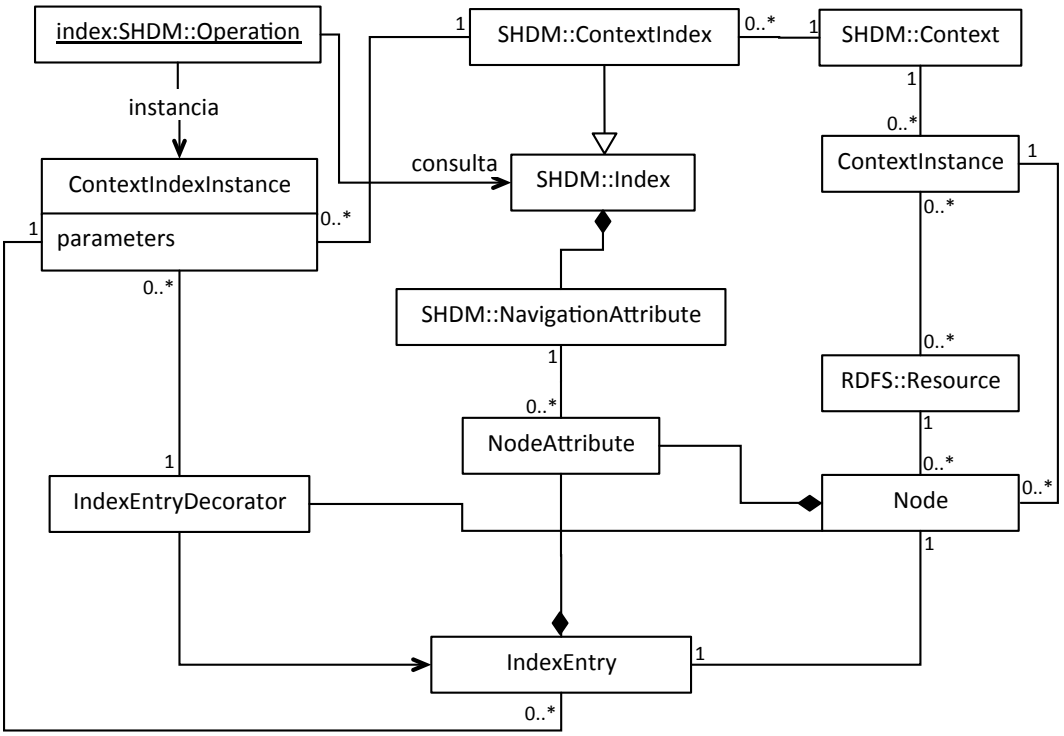


Figura 21 – Diagrama de classes envolvidas na instanciação de um contexto navegacional

A classe SHDM::Index representa os índices navegacionais em tempo de modelagem e guarda as informações básicas sobre um índice, similares às informações descritas em um cartão de índice [Rossi, 1996]. A classe SHDM::ContextIndex é uma especialização para índices baseados em contexto e possui um relacionamento com a classe SHDM::Context.

Um índice é composto por um conjunto de atributos navegacionais, representados pela classe SHDM::NavigationAttribute, e funcionam de forma similar aos atributos navegacionais das classes em contexto.

A classe ContextIndexInstance representa um índice baseado em contexto em tempo de execução e seu funcionamento é similar ao de ContextInstance. As instâncias de ContextIndexInstance possuem um conjunto de entradas de índices baseadas nos nós do contexto relacionado ao índice. Para a geração das entradas

do índice, instâncias de IndexEntry, a classe IndexEntryDecorator decora as instâncias de Node obtidas da instância do contexto relacionado ao índice com os atributos navegacionais, instâncias de SHDM::NavigationAttribute, que compõem o índice. Todo o processo é similar ao descrito para a classe NodeDecorator.

A operação externa "index", instância da classe SHDM::Operation, possui toda a lógica de navegação no índice. Seu funcionamento também é similar ao da operação externa "context".

4.3.5.2.3. Atributos navegacionais

A Figura 22 apresenta as classes do módulo navegacional relacionadas com os atributos navegacionais.

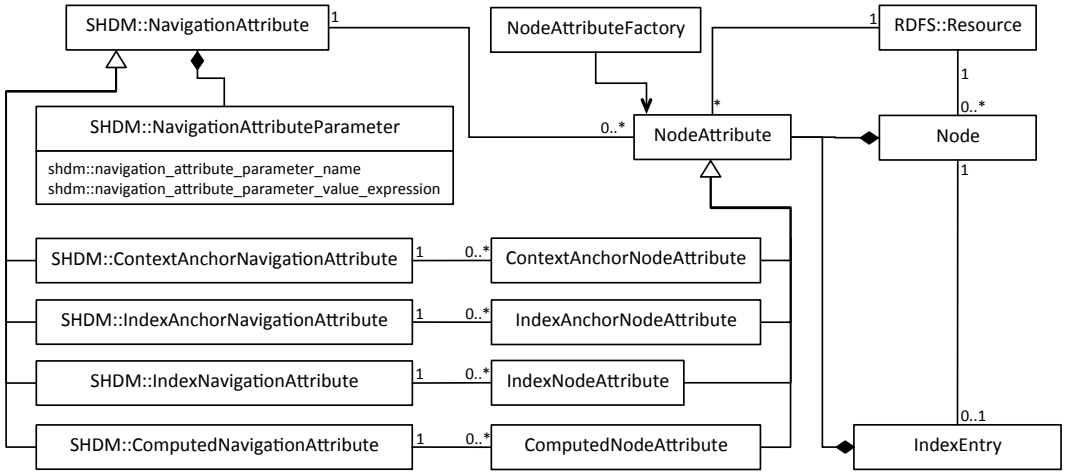


Figura 22 – Diagrama de classes envolvidas na instanciação de um atributo navegacional

A classe SHDM::NavigationAttribute e suas subclasses representam os atributos navegacionais em tempo de modelagem, ou seja, as descrições sobre a composição dos atributos navegacionais de um nó de uma classe em contexto ou uma entrada de índice.

A classe SHDM::NavigationAttributeParameter representa os parâmetros de um atributo navegacional.

A classe NodeAttribute representa o atributo navegacional em tempo de execução. As instâncias de Node e IndexEntry são compostas por um conjunto de instâncias de NodeAttribute, ou seja, atributos navegacionais obtidos em tempo de execução. A classe NodeAttributeFactory é a implementação do padrão de projeto *Abstract Factory* [Gamma et al., 1995] e é responsável por criar as instâncias das

subclasses de NodeAttribute de acordo com a subclasse de SHDM::NavigationAttribute do atributo a ser instanciado. Uma instância de NodeAttribute conhece o recurso RDF ao qual é associado e, dessa forma, acessa as propriedades do recurso para compor os valores do atributo navegacional.

A Figura 23 apresenta as relações entre as classes do módulo navegacional e os elementos que compõem um índice navegacional gerado em tempo de execução.

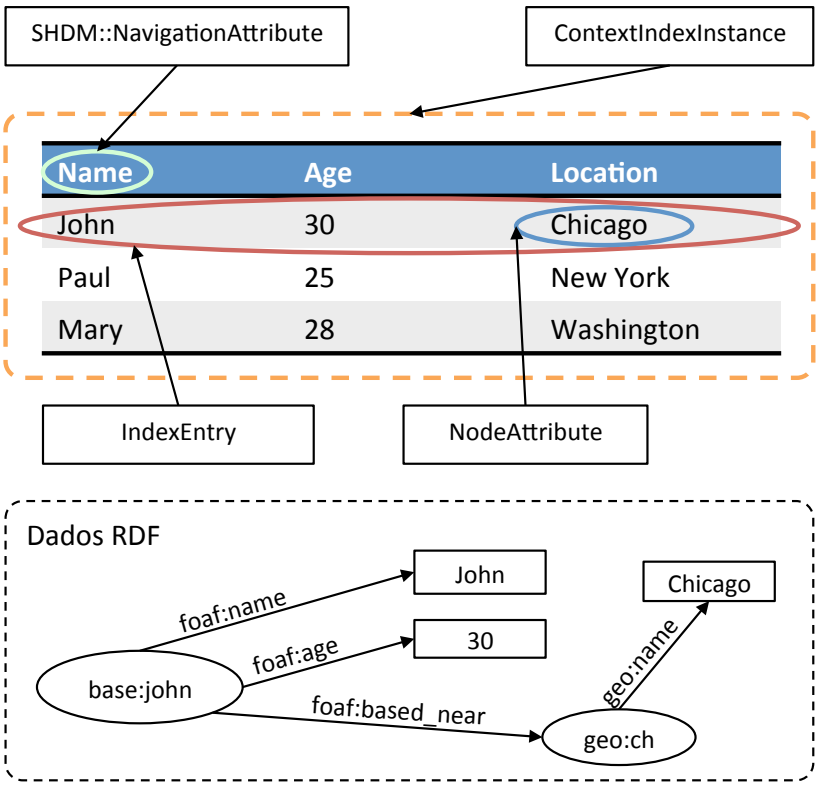


Figura 23 - Relações entre as classes do módulo navegacional e um índice

4.3.5.2.4. Controladores do módulo navegacional

A Figura 24 apresenta o diagrama de classes de controladores do módulo navegacional.

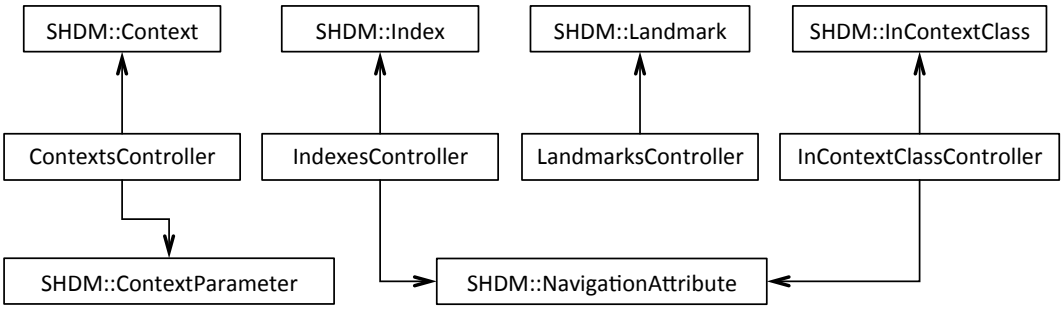


Figura 24 – Diagrama de classes de controladores do módulo navegacional

No módulo navegacional há uma classe de controlador para cada primitiva básica. A função desses controladores é fornecer uma interface web para o ambiente de autoria do Synth que permita a manipulação das primitivas do modelo navegacional. Embora tenha sido omitido no diagrama, todos os controladores possuem os métodos “index”, “new”, “create”, “edit”, “update” e “destroy” para as operações CRUD da classe relacionada ao controlador.

A seção 4.4.3 apresenta a interface gráfica do ambiente de autoria para a manipulação do modelo navegacional que faz uso dos controladores do módulo navegacional.

4.3.5.3. Módulo comportamental

A Figura 25 apresenta as classes do módulo comportamental.

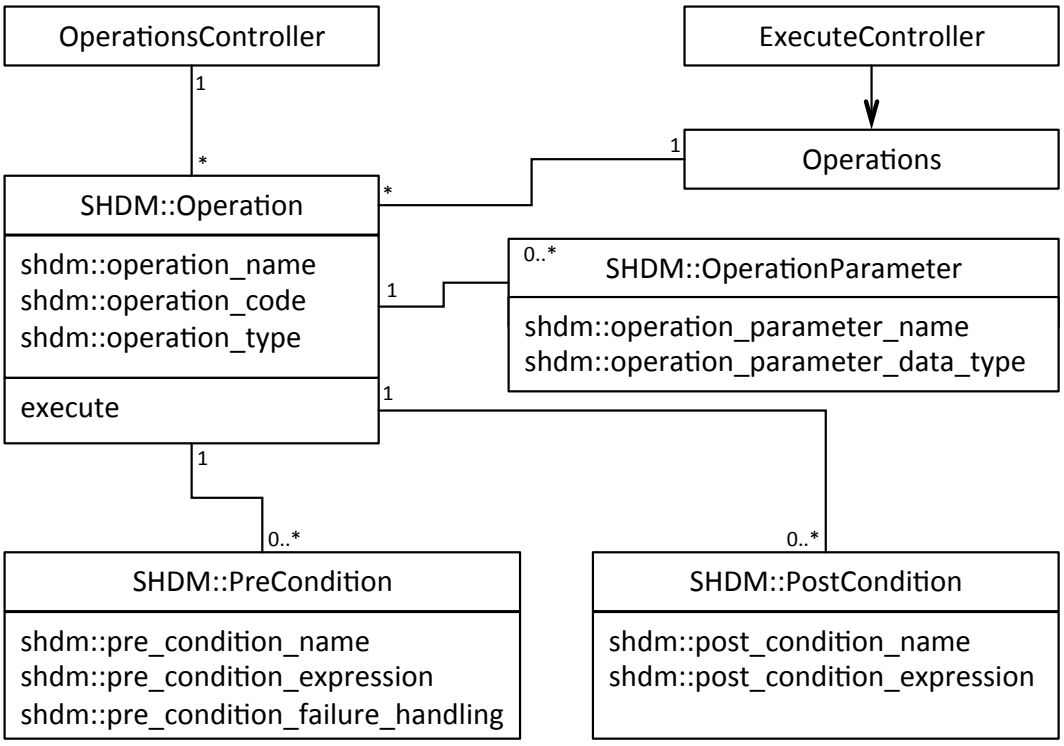


Figura 25 – Diagrama de classes do módulo comportamental

A implementação deste módulo segue a proposta de [Santos, 2010] para a modelagem de operações no HyperDE.

A classe SHDM::Operation representa as operações da aplicação. Suas instâncias devem conter um nome, um tipo e um código em linguagem Ruby.

Graças as funcionalidades de meta programação da linguagem Ruby, as instâncias de SHDM::Operation são capazes de efetuar a execução de seu próprio código a partir do método "execute". Quando o método "execute" é invocado o código contido no valor da propriedade shdm::operation_code é avaliado no escopo da instância da operação. O Quadro 14 ilustra um exemplo de execução de uma operação interna.

```
op = SHDM::Operation.new
op.shdm::operation_name = "sum"
op.shdm::operation_type = "internal"
op.shdm::operation_code = "1 + 1"
op.execute()
=> 2
```

Quadro 14 – Exemplo de execução de uma operação interna

A classe SHDM::OperationParameter representa os parâmetros esperados por uma operação. Devem ser informados um nome e um tipo de dados esperado. Os nomes dos parâmetros são usados no código da operação. A presença e o tipo dos parâmetros são conferidos durante a execução. Caso os tipos recebidos não combinem com os tipos esperados, uma mensagem de erro é retornada. O Quadro 15 ilustra execução de uma operação interna que possui parâmetros.

```
param1 = SHDM::OperationParameter.new
param1.shdm::operation_parameter_name = "a"
param1.shdm::operation_parameter_data_type = "Integer"

param2 = SHDM::OperationParameter.new
param2.shdm::operation_parameter_name = "b"
param2.shdm::operation_parameter_data_type = "Integer"

op = SHDM::Operation.new
op.shdm::operation_name = "sum"
op.shdm::operation_type = "internal"
op.shdm::operation_code = "a + b"
op.shdm::operation_parameters = [ param1, param2 ]

op.execute({:a => 2, :b => 3})
=> 5
```

Quadro 15 - Exemplo de execução de uma operação interna que possui parâmetros

A classe SHDM::PreCondition representa as pré-condições de uma operação. As pré-condições devem conter um nome, uma expressão em Ruby e um código de tratamento de falha. A expressão de uma pré-condição é avaliada antes do código da operação e deve retornar *true* ou *false*. Quando a avaliação retorna *true*, o código da operação é executado em seguida. Caso retorne *false*, o código do tratamento de falha é executado no lugar do código da operação. O

próprio código do tratamento de falha pode decidir se deve executar o código da operação após o tratamento da falha.

A classe SHDM::PostCondition representa as pós-condições de uma operação. Tem o funcionamento similar ao das pré-condições, porém é executado depois do código da operação e não possui tratamento de falha. Caso a avaliação da expressão da pós-condição retorne *false*, uma mensagem de erro é retornada.

Para facilitar a execução das operações no Synth, o módulo Ruby Operations fornece atalhos para a execução direta das operações pelos seus nomes. Considerando o exemplo do Quadro 15, a operação "sum" poderia ser invocada diretamente pelo código Operations.sum({:a => 2, :b => 3}). O módulo Operations instancia as operações externas como métodos de classe para serem usadas pelo controlador ExecutionController.

A classe ExecutionController possibilita a execução das operações externas por agentes externos por meio do envio de requisições do protocolo HTTP ao servidor no seguinte endereço:

http://[endereco_do_servidor]/execute/[nome_da_operação]

Pelo uso das funcionalidades de meta programação da linguagem Ruby, a classe ExecutionController, que é um controlador do Ruby on Rails, inclui entre seus métodos todos os métodos de classe do módulo Ruby Operations. Dessa maneira, esses métodos tomam forma de ações do controlador ExecutionController e podem ser invocados pelo envio de requisições HTTP.

A classe OperationsController fornece as funcionalidades de CRUD do modelo de operações para o ambiente de autoria do Synth.

A seção 4.4.4 apresenta a interface gráfica do ambiente de autoria para a manipulação do modelo comportamental.

4.3.5.4. Módulo de interface

O módulo de interface foi implementado segundo a proposta para um módulo de interfaces ricas do HyperDE apresentada por [Luna, 2010, p. 74]. A Figura 26 apresenta o diagrama de classes do módulo de interface.

As instâncias das classes SWUI::HTMLTag e SWUI::DHTMLEffect correspondem às instâncias dos conceitos de mesmo nome na Ontologia de

Descrição de Interfaces Ricas. As classes SWUI::ConcreteWidget, SWUI::RichControl e SWUI::Effect, por sua vez, correspondem aos conceitos ConcreteInterfaceElement, DHTMLRichControl e ConcreteEffect, da mesma ontologia.

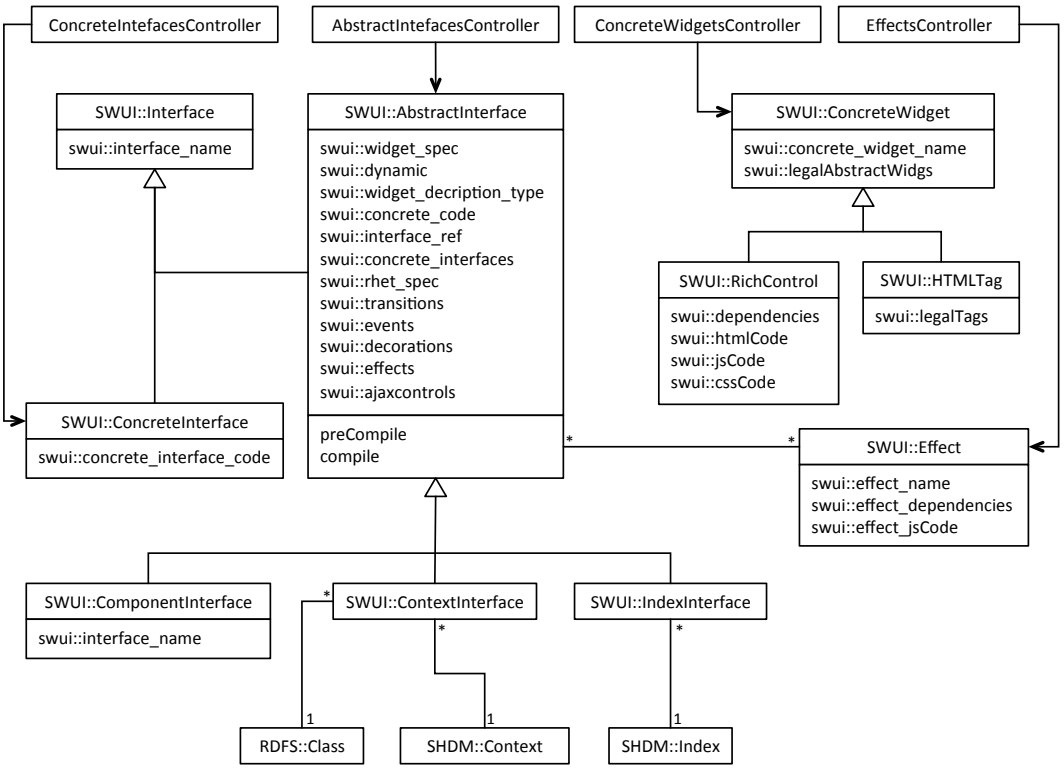


Figura 26 – Diagrama de classes do módulo de interface

SWUI::Interface é uma generalização que representa interfaces de uma aplicação.

A classe SWUI::AbstractInterface é a abstração para as descrições abstratas das interfaces da aplicação. O atributo swui::widget_desc armazena a descrição de instâncias dos conceitos AbstractInterface e AbstractInterfaceElement da ontologia de descrição de interfaces. O atributo swui::rhet_desc armazena a descrição de instâncias dos conceitos de Transition, RhetoricalStructure, Decoration e Event. As classes SWUI::ComponentInterface e SWUI::ConcreteInterface, embora não representem conceitos da ontologia de descrição de interfaces, agregam novos recursos ao ambiente de modelagem, como a criação de componentes reutilizáveis (SWUI::ComponentInterface) e a declaração de folhas de estilo CSS (SWUI::ConcreteInterface).

As classes de controladores fornecem as funcionalidades de CRUD do modelo de interface para o ambiente de autoria do Synth.

A seção 4.4.5 apresenta a interface gráfica do ambiente de autoria para a manipulação do modelo de interface.

4.4. Ambiente de autoria

O ambiente de autoria do Synth dispõe de uma interface gráfica de formulários que é acessada por meio de um navegador de internet e permite a execução, pelo projetista, das operações de visualização, criação, remoção e edição das primitivas dos modelos de uma aplicação construída segundo o método SHDM. Através desta interface é possível executar a aplicação enquanto ocorre a sua construção e, dessa forma, validá-la a cada passo do seu desenvolvimento.

Por ser uma aplicação *web*, o ambiente de autoria do Synth pode ser utilizado remotamente, bastando que se acesse a URL do serviço em um navegador de internet. As URLs de acesso ao ambiente de autoria do Synth seguem o padrão de URLs do *framework* Ruby on Rails, apresentado no Quadro 16.

```
http://[endereco_do_servidor]/[controlador_de_retaguarda]
```

Quadro 16 – Padrão de URLs do Synth

O Synth possui vários controladores de retaguarda para o ambiente de autoria, apresentados nas subseções da seção 4.3.5. Estes controladores são responsáveis pelas operações CRUD das primitivas que compõem os artefatos produzidos na execução de cada etapa do método SHDM. A Tabela 4 apresenta os endereços dos controladores de retaguarda do ambiente de autoria do Synth, organizados por etapa do método SHDM ao qual estão relacionados.

Etapa	Controladores
Modelagem do domínio	ontologies, namespaces, datasets, resources, classes, properties
Projeto navegacional	contexts, indexes, landmarks, in_context_classes
Projeto de interface	abstract_interfaces, concrete_interfaces, concrete_widgets, effects
Projeto comportamental	operations
“nenhuma”	applications

Tabela 4 – Controladores de retaguarda do Synth

A interface gráfica do ambiente de autoria foi organizada em dois níveis de navegação. No primeiro nível é exibido um menu horizontal de itens relacionados com cada etapa do método SHDM. Sendo assim, temos “Domain” para a modelagem de domínio, “Navigation” para o projeto navegacional, “Interface” para o projeto de interface e “Behavior” para o projeto comportamental. O item “Home” leva para a tela inicial do ambiente de autoria e não está relacionado a nenhuma etapa do método SHDM.

Ao clicar em um item de menu horizontal de primeiro nível, um menu vertical de segundo nível será exibido, apresentando itens que representam as primitivas que compõem os artefatos que serão manipulados na etapa relacionada ao item de primeiro nível selecionado. Por exemplo, ao clicar no item de primeiro nível “Navigation”, será exibido o menu de segundo nível apresentando os itens “Contexts”, “Indexes”, “Landmarks” e “In Context Classes”, todos relacionados às primitivas do projeto navegacional.

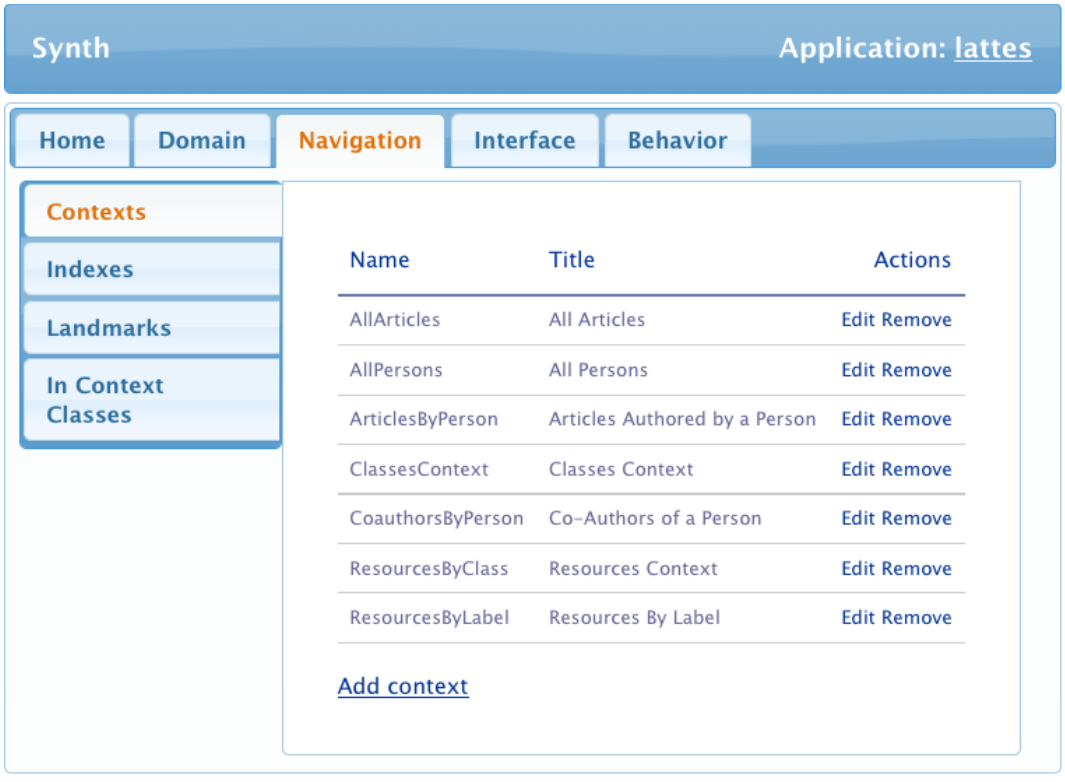


Figura 27 - Tela de listagem de contextos navegacionais

A Figura 27 apresenta a tela da modelagem de contextos navegacionais, onde é possível ver os itens do menu horizontal de primeiro nível e os itens do menu vertical de segundo nível.

As páginas que compõem a interface gráfica do ambiente de autoria 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 editada.

Todas as telas do ambiente de autoria possuem um cabeçalho padrão onde é apresentado o nome da aplicação ativa, ao lado do rótulo "Application". Uma aplicação ativa é aquela que está recebendo alterações no momento, sendo assim, todas as outras telas do ambiente de autoria darão acesso apenas às primitivas da aplicação ativa. Apenas uma aplicação pode estar ativa por vez. Na Figura 27, o nome da aplicação ativa é "lattes".

Serão apresentadas, a seguir, as descrições das tarefas que podem ser executadas na tela inicial e sobre cada uma das primitivas relacionadas com cada uma das etapas do SHDM através da interface gráfica do ambiente de autoria do Synth.

4.4.1. Tela inicial

A Figura 28 apresenta a tela inicial do ambiente de autoria do Synth. Esta tela é apresentada sempre que se acessa a URL de endereço do servidor que, em geral, é http://localhost:3000, ou quando se clica no item de menu "Home".

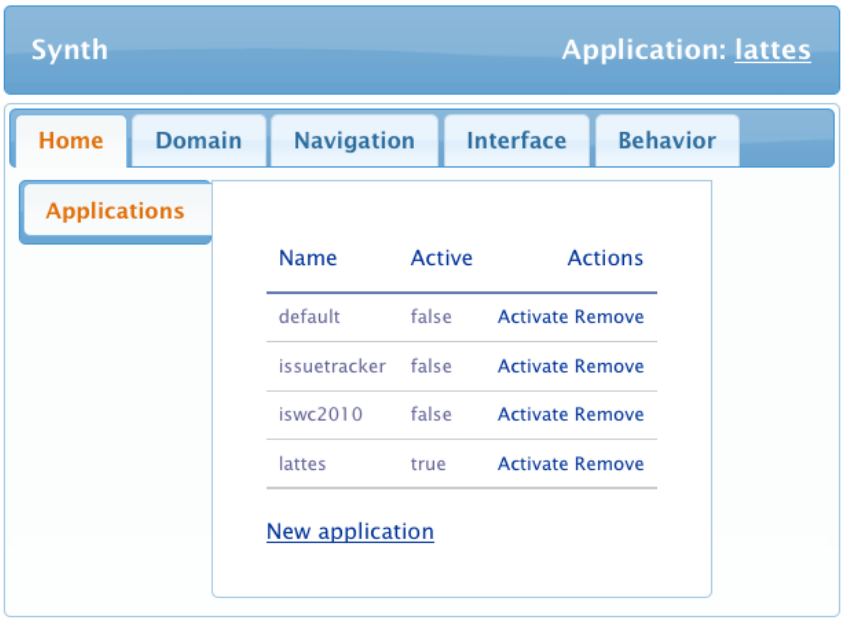


Figura 28 – Tela inicial do ambiente de autoria do Synth

A tela inicial apresenta apenas um único item no menu de segundo nível, “Applications”, que leva à listagem de aplicações cadastradas no Synth. Por meio desta tela é possível criar novas aplicações, remover aplicações e ativar uma aplicação.

Para criar uma nova aplicação, o projetista deve clicar na âncora “New application” e então a tela apresentada na Figura 29 será exibida. O nome da aplicação deve ser informado, e ao clicar no botão “create” o Synth criará a nova aplicação.

Uma aplicação Synth é um diretório com o mesmo nome da aplicação dentro do diretório “applications”, na raiz do diretório de instalação do Synth. Cada diretório de aplicação possui dois subdiretórios: “db” e “ontologies”. No diretório “db” fica o arquivo “main”, um arquivo texto na notação RDF/XML que é o banco de dados local da aplicação. No diretório “ontologies” devem ficar os arquivos que representam as ontologias de domínio utilizadas pela aplicação. Quando uma aplicação é criada através da interface de autoria, um novo diretório contendo os subdiretórios e arquivos básicos de uma aplicação é criado dentro do diretório “applications” do Synth.

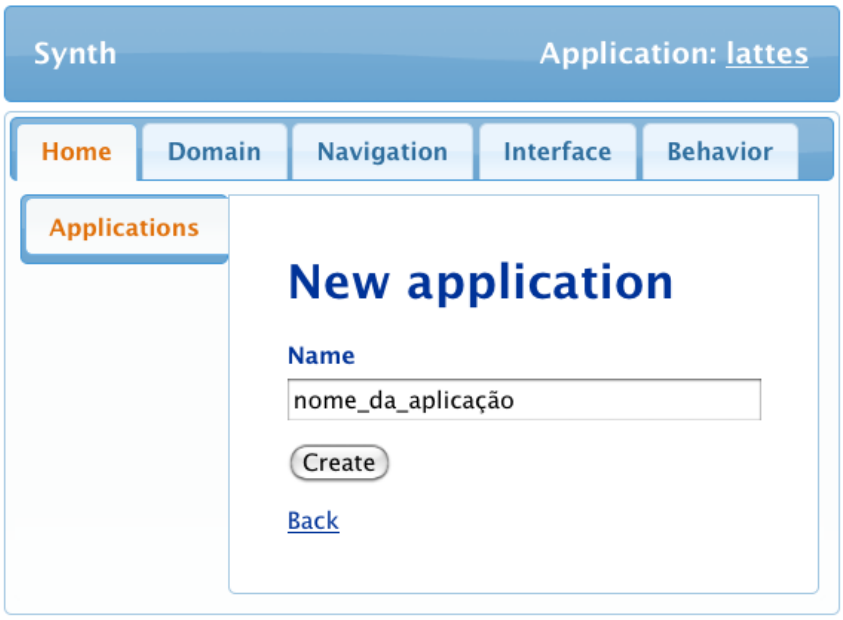


Figura 29 – Tela e criação de uma nova aplicação no Synth

4.4.2.
Modelagem de domínio

A tela de modelagem de domínio é acessada através do item de menu "Domain". Nesta tela, são manipuladas as primitivas do modelo de domínio acessadas por meio dos itens de menu de segundo nível "Ontologies", "Namespaces", "Datasets", "Resources", "Classes" e "Properties". O acesso ao item "Domain" leva diretamente ao primeiro item de menu de segundo nível "Ontologies" para a manipulação das ontologias de domínio da aplicação.

4.4.2.1.
Ontologias

O gerenciamento das ontologias de domínio da aplicação se inicia pela listagem das ontologias previamente cadastradas, conforme apresentado na Figura 30. Esta tela é apresentada ao clicar na âncora "Ontologies". A partir desta tela, é possível adicionar novas ontologias, editar ou remover ontologias previamente cadastradas.

Figura 30 – Tela de listagem de ontologias de domínio

Ao clicar na âncora "Add ontology", a tela de cadastro de uma nova ontologia será apresentada, conforme apresentado na Figura 31.

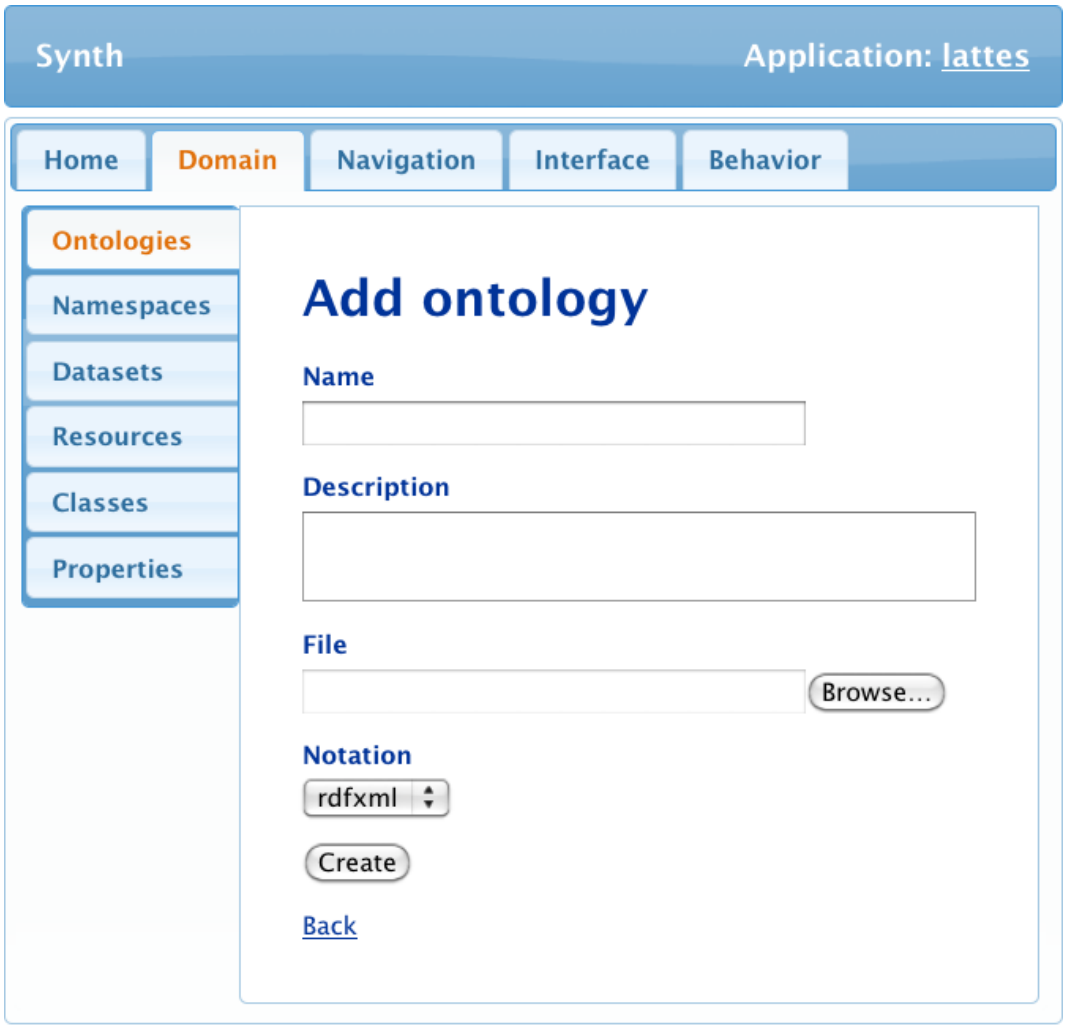


Figura 31 – Tela de cadastro de uma nova ontologia

No Synth, uma ontologia de domínio é um arquivo de texto contendo as asserções sobre uma ontologia em RDFS ou OWL representada em qualquer uma das seguintes notações: RDF/XML, N3, NTRIPLES, TURTLE.

No formulário apresentado na Figura 31 deve-se informar um nome para ontologia, uma descrição, um arquivo RDF e a notação do arquivo RDF. Ao clicar no botão “Create”, os dados do formulário e o arquivo RDF indicado serão enviados para o servidor.

Na tela de edição apresentada na Figura 32, é possível alterar a descrição da ontologia e ativar ou desativar as inferências sobre a ontologia cadastrada.

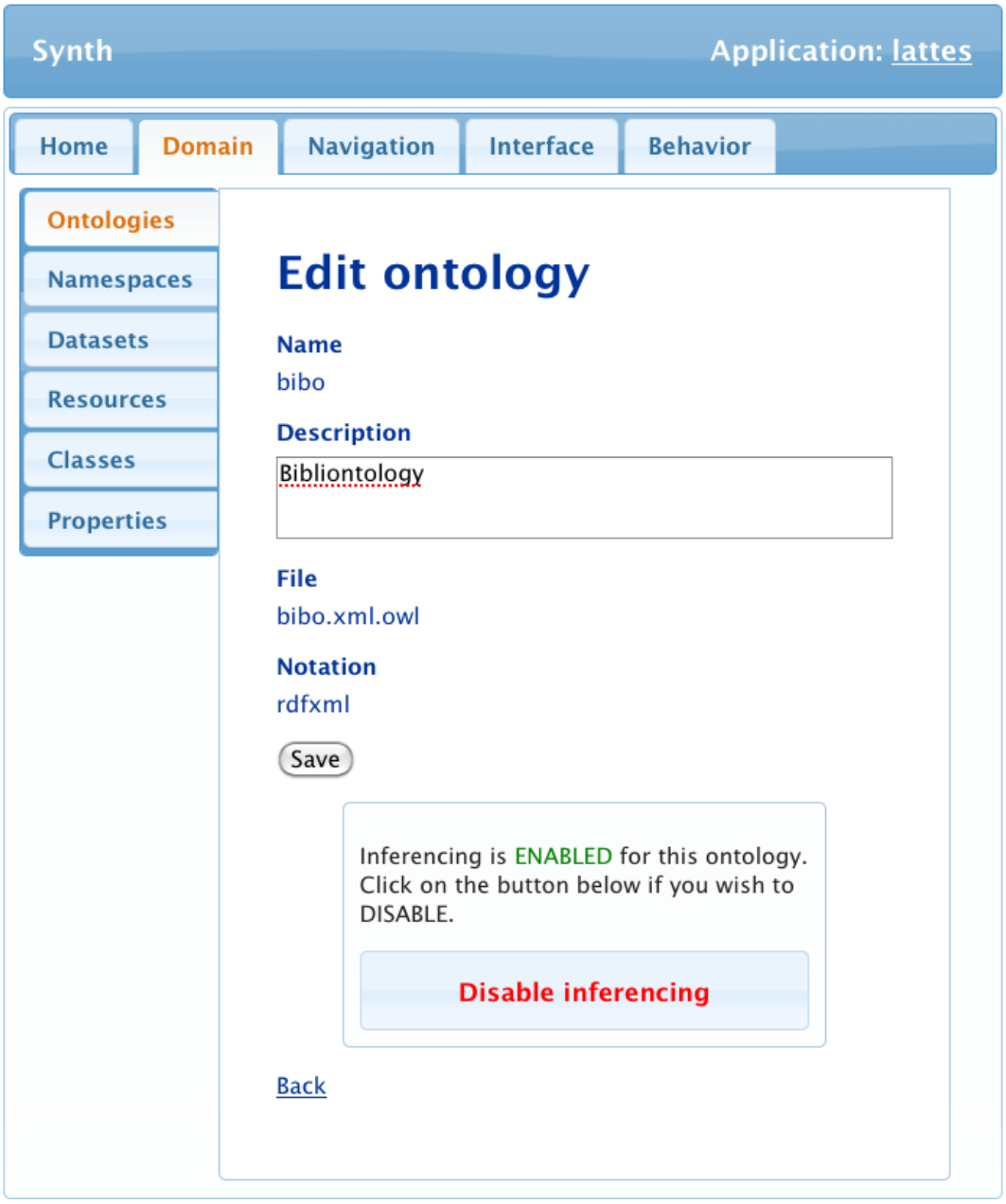


Figura 32 – Tela de edição de uma ontologia

Ativar as inferências sobre uma ontologia significa informar para o módulo de persistência que as asserções contidas no arquivo RDF da ontologia devem ser levadas em consideração nas consultas feitas à base de dados do Synth. Por exemplo, se na base de dados do Synth existe uma asserção que indica que o recurso A possui como valor de rdf:type a classe foaf:Person e uma das ontologias ativas possui uma asserção que indica que foaf:Person é rdfs:subClassOf de foaf:Agent, as consultas pelos recursos cujo rdf:type é foaf:Agent devem retornar o recurso A.

Desativar as inferências sobre uma ontologia significa apenas que o módulo de persistência deve ignorar as asserções contidas no arquivo RDF da ontologia.

4.4.2.2. Namespaces

O gerenciamento dos *namespaces* da aplicação se inicia pela listagem dos *namespaces* previamente cadastrados, conforme apresentado na Figura 33.

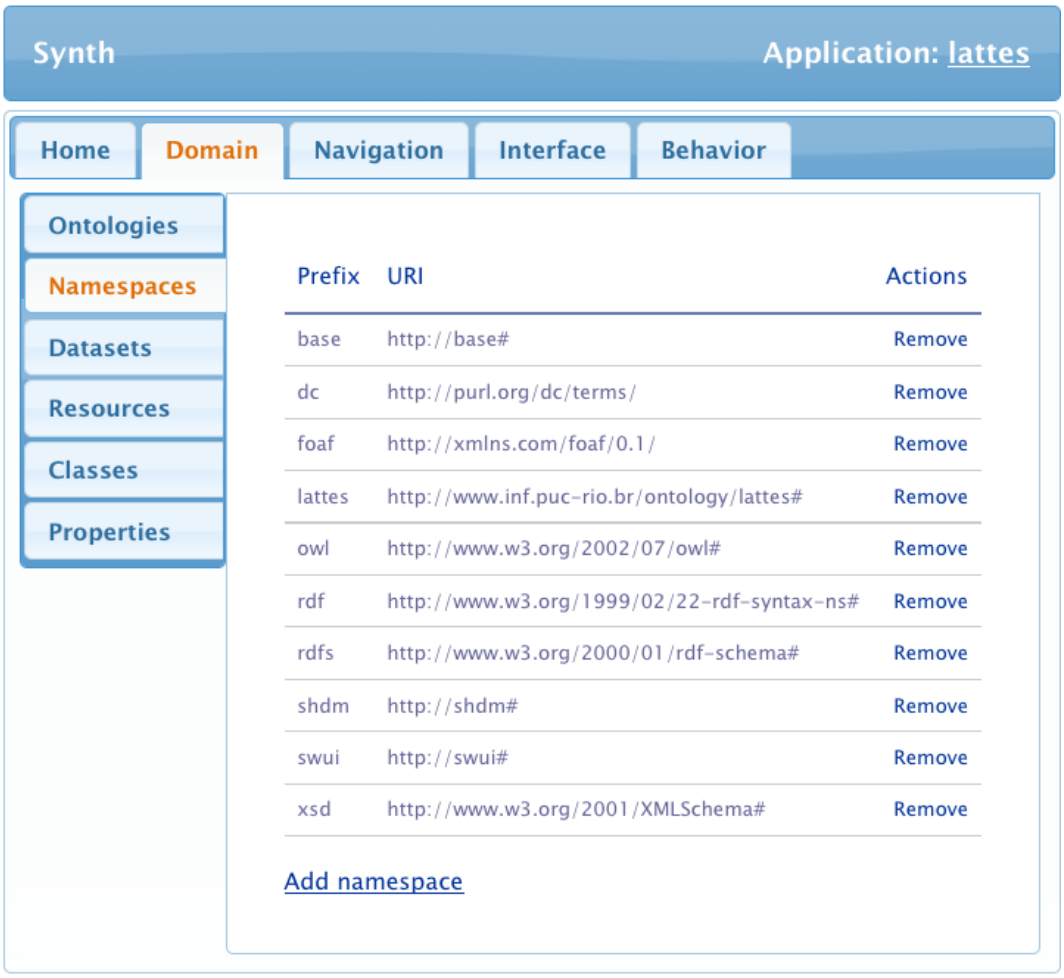


Figura 33 – Tela de listagem de namespaces

Esta tela é apresentada ao clicar na âncora "Namespaces". A partir desta tela, é possível adicionar novos *namespaces* e remover *namespaces* previamente cadastrados.

Ao clicar na âncora "Add namespace", a tela de cadastro de um novo *namespace* será apresentada, conforme apresentado na Figura 34.

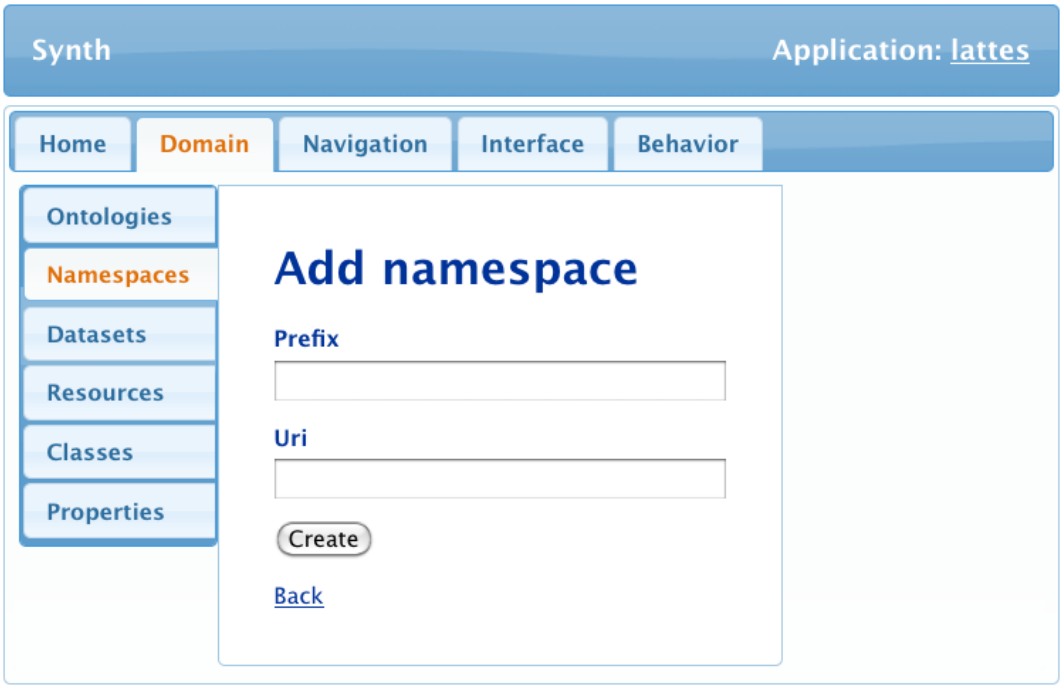


Figura 34 – Tela de cadastro de um novo *namespace*

Na tela de cadastro de um novo *namespace*, deve-se informar um texto para o prefixo e o URI correspondente.

Uma vez que o *namespace* esteja cadastrado, é possível referir-se ao mesmo nos outros modelos da aplicação. Por exemplo, para referir-se à classe RDF foaf:Person na linguagem de consulta dos contextos, deve-se usar o símbolo FOAF::Person. Neste caso, o símbolo FOAF aponta para um módulo da linguagem Ruby, que representa o *namespace* associado ao URI http://xmlns.com/foaf/0.1/. FOAF::Person é uma classe Ruby contida no módulo Ruby para o qual o símbolo FOAF aponta e representa a classe RDF foaf:Person.

Por se tratar de uma primitiva muito simples, que contém apenas dois atributos, a tela de listagem de *namespaces* não oferece a funcionalidade de edição de um *namespace*. Quando se deseja alterar o URI de um *namespace*, basta removê-lo e criar um novo.

4.4.2.3. Repositórios externos

O gerenciamento dos repositórios externos da aplicação se inicia pela listagem dos repositórios previamente cadastrados, conforme apresentado na Figura 35. Esta tela é apresentada ao clicar na âncora “Datasets”. A partir desta

tela, é possível adicionar novos repositórios externos, editar ou remover repositórios previamente cadastrados.

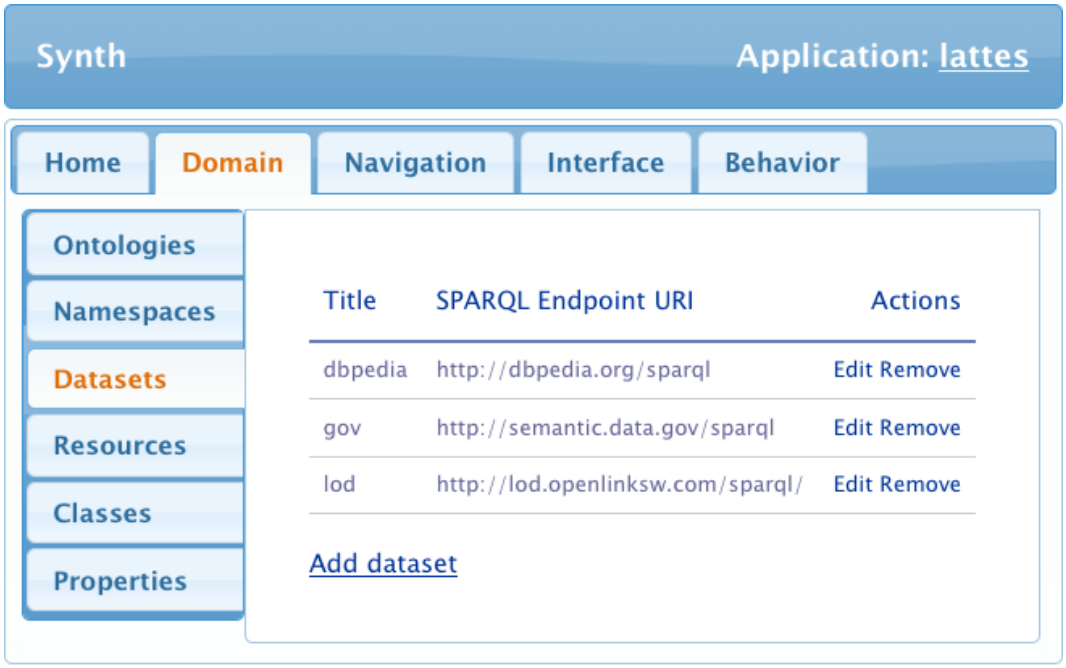


Figura 35 – Tela de listagem dos repositórios

Ao clicar na âncora "Add dataset" a tela de cadastro de um novo repositório será apresentada, conforme apresentado na Figura 36.

Na tela de cadastro de um novo repositório deve-se informar um título para o repositório e o URI do SPARQL *endpoint* correspondente.

Uma vez que o repositório esteja cadastrado, é possível referir-se ao mesmo nas consultas dos contextos da aplicação pelo seu título prefixado pelo símbolo : (dois pontos). Por exemplo, o Quadro 17 apresenta uma consulta por todos os recursos do tipo foaf:Person que será feita apenas no SPARQL *endpoint* do repositório cujo título é 'dbpedia'.

```
selects {
  type FOAF::Person
  datasets :dbpedia
}
```

Quadro 17 – Exemplo de consulta de contexto em um repositório externo

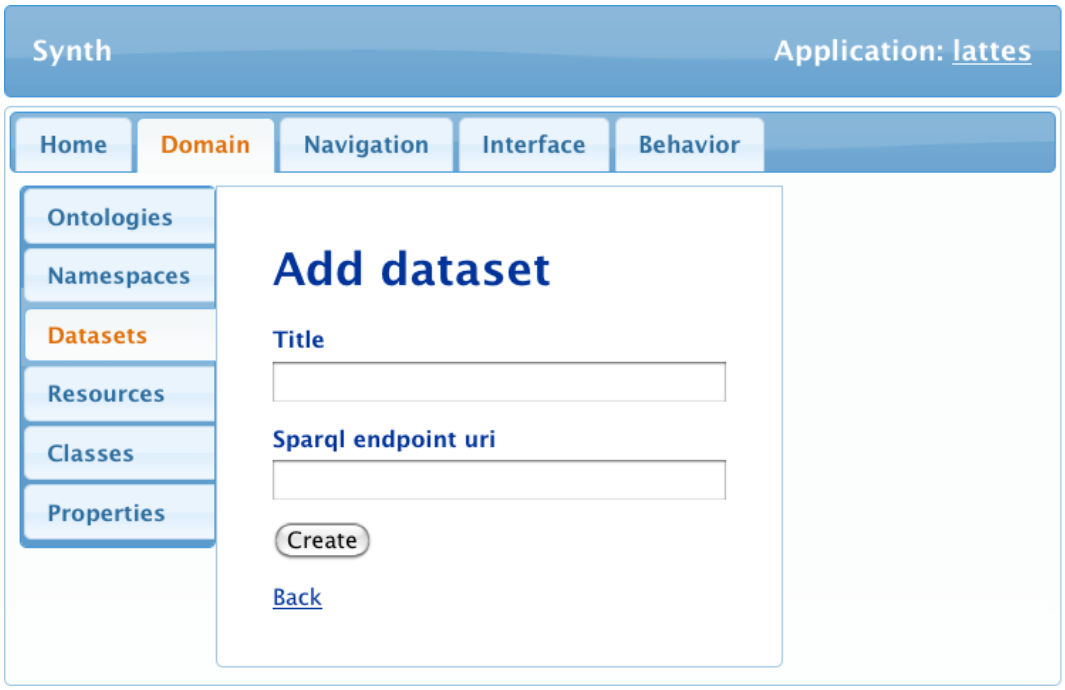


Figura 36 – Tela de cadastro de um novo repositório

Na tela de edição de um repositório, apresentada Figura 37, é possível alterar o título ou o URI do SPARQL *endpoint* do repositório.

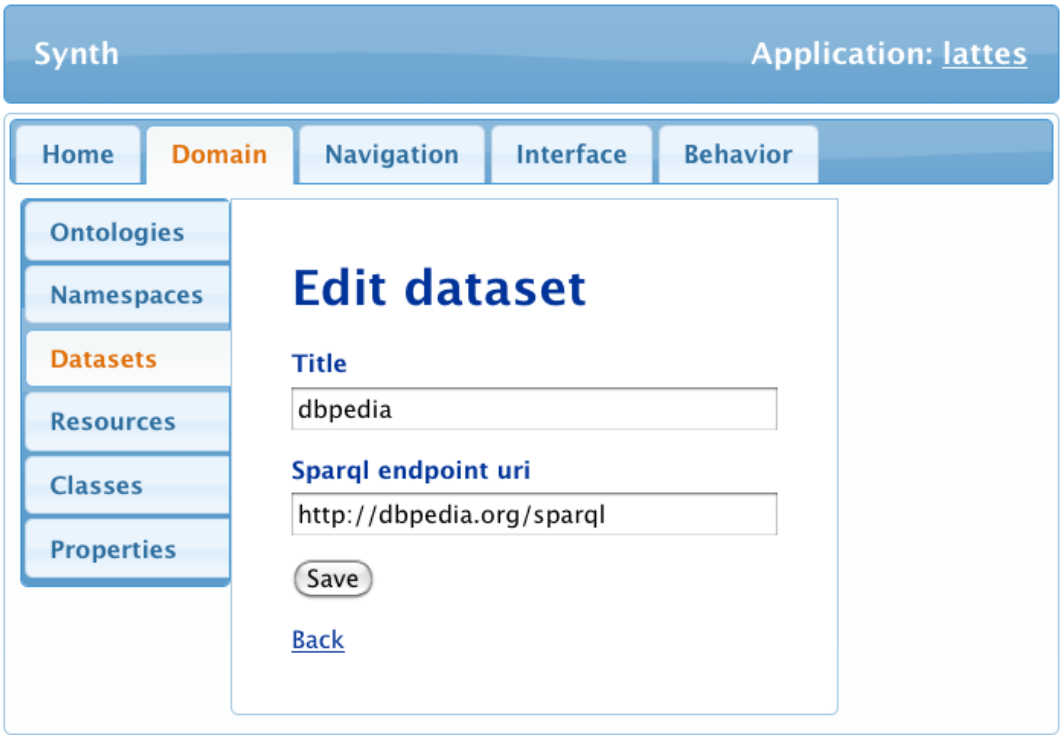


Figura 37 – Tela de edição de um repositório

4.4.2.4. Editor RDF

Os recursos RDF são as primitivas mais básicas da modelagem de domínio. Por isso, o ambiente de autoria do Synth dispõe de um editor RDF, que é uma interface gráfica de visualização e edição, adequada às características do modelo de dados RDF. Por exemplo, como não é possível prever quais são as propriedades de um recurso RDF, o editor RDF possui uma interface gráfica capaz de ajustar-se de acordo com as propriedades do recurso a ser editado. O editor RDF que acompanha o ambiente de autoria do Synth foi inspirado na ferramenta OntoWiki[77], que é um ambiente para visualização e edição de bases de dados RDF.

A Figura 38 apresenta uma das telas do editor RDF acessada a partir do ambiente de autoria do Synth.

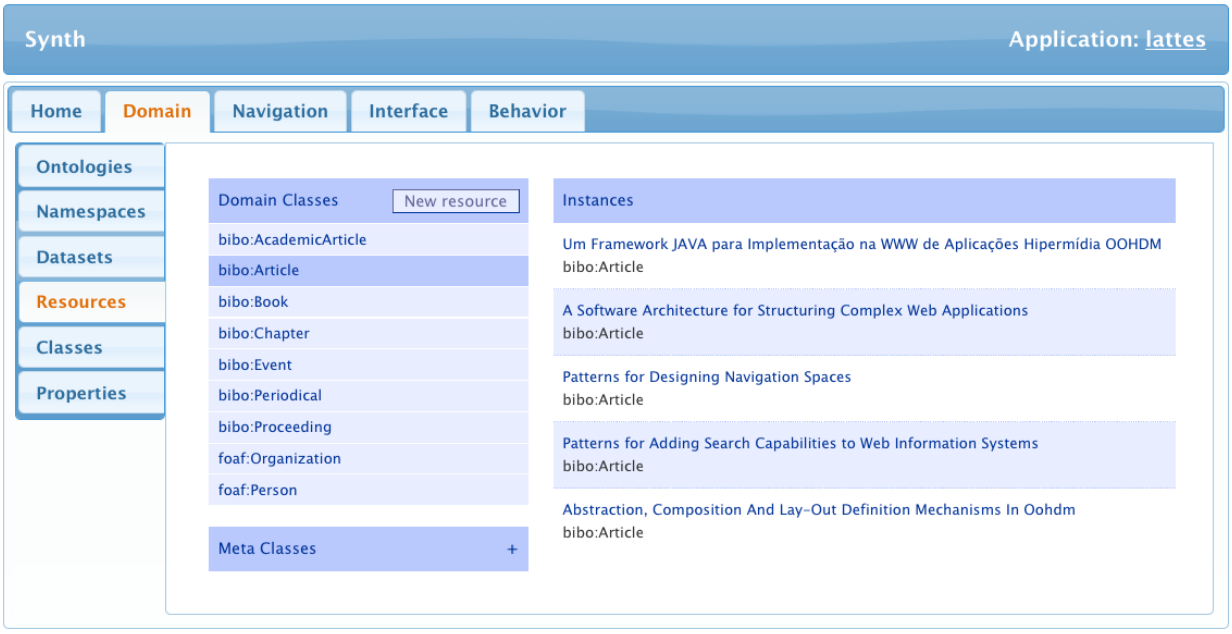


Figura 38 – Editor RDF

O editor RDF pode ser acessado tanto através do ambiente de autoria como pela própria aplicação resultante da modelagem no Synth. Isso torna possível que se forneça, junto com a aplicação, um editor para os dados que estão sendo manipulados pela mesma.

[77] http://ontowiki.net

As âncoras “Resources”, “Classes” e “Properties” dão acesso ao editor RDF. O clicar em uma dessas âncoras, o editor RDF será exibido com o foco na primitiva correspondente à âncora clicada.

4.4.2.4.1. Recursos

Ao clicar na âncora “Resources”, o editor RDF é exibido, apresentando um menu navegacional com as classes do domínio da aplicação presentes tanto na base de dados local do Synth como nas ontologias de domínio ativas. A Figura 39 apresenta a tela inicial de recursos do editor RDF.

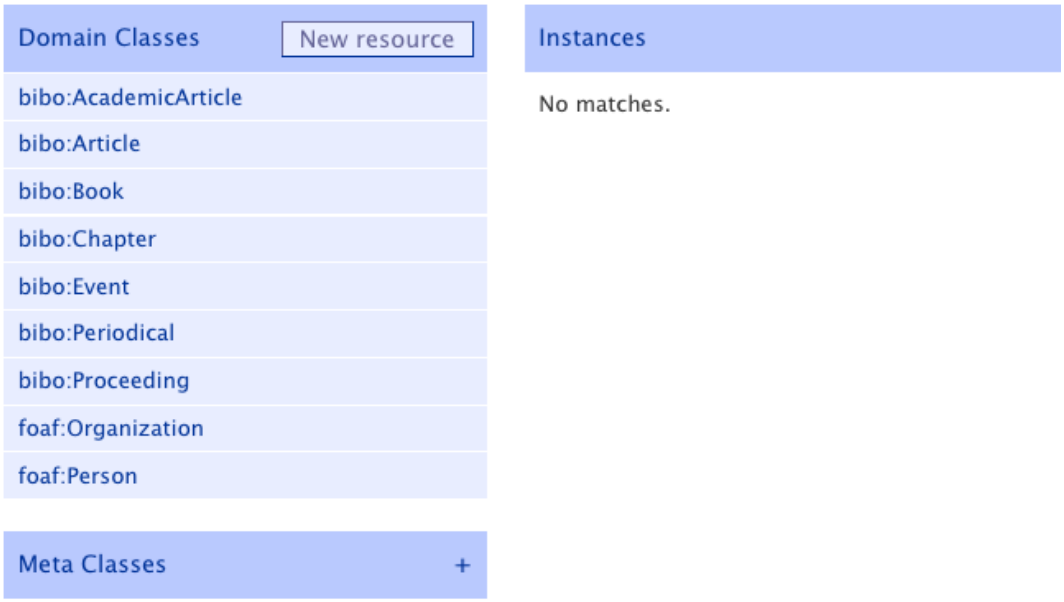


Figura 39 – Tela inicial de recursos do editor RDF

Além classes de domínio é possível listar as meta classes do Synth clicando no sinal “+” ao lado do rótulo “Meta Classes”. As meta classes são todas as classes que descrevem os modelos manipulados pelo Synth além das classes básicas das linguagens RDF(S) e OWL. A grosso modo, são as classes agrupadas nos *namespaces* owl, rdf, rdfs, shdm, swui, synth e void. O funcionamento do menu de meta classes é idêntico ao do menu de classes de domínio.

Como é possível notar na Figura 39, o lado direito da tela apresenta a mensagem “No matches”. Esta mensagem é mostrada sempre que não é encontrado nenhum recurso da classe selecionada no menu da esquerda ou quando

não há uma classe selecionada. Na tela inicial de recursos, como nenhuma classe foi selecionada, a mensagem “No matches” é exibida.

Ao clicar em uma das classes do menu esquerdo, a listagem de recursos da classe selecionada será apresentada no lado direito da tela. A

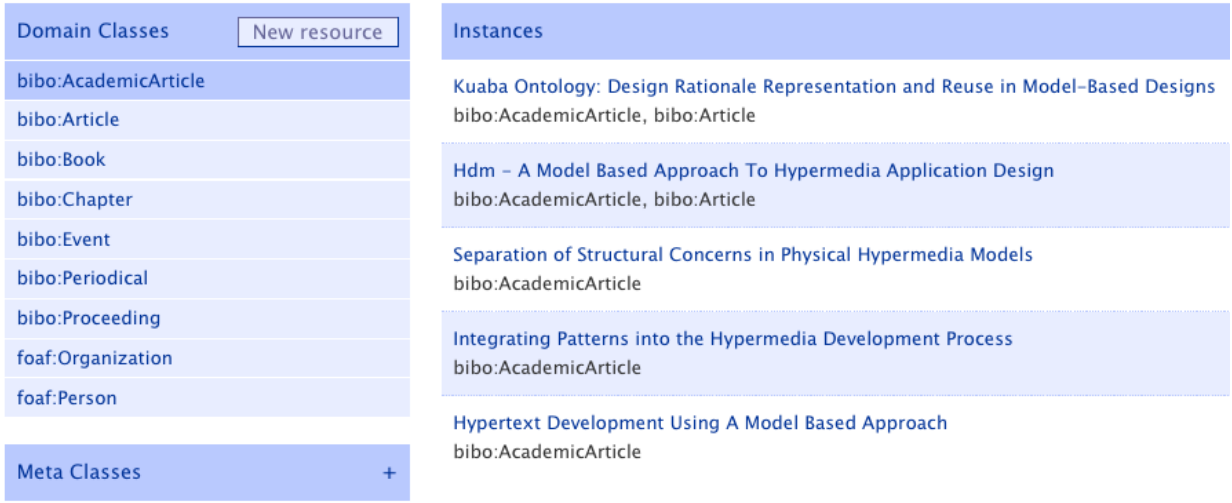


Figura 40 apresenta a listagem de recursos da classe bibo:AcademicArticle.

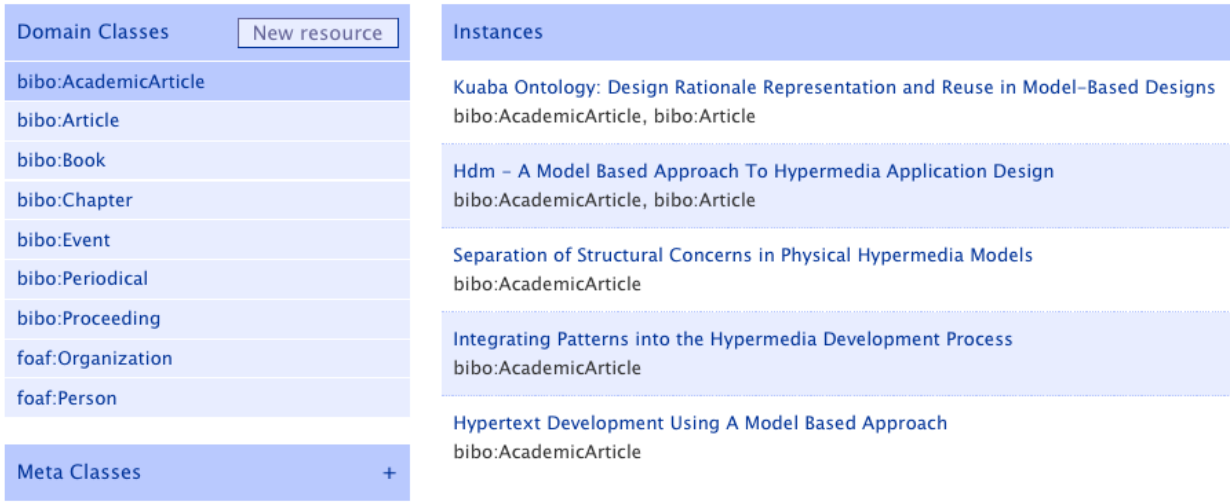


Figura 40 – Listagem de recursos da classe bibo:AcademicArticle

A listagem de recursos apresentada no lado direito da tela exibe o valor da propriedade rdfs:label do recurso com uma âncora para a sua edição e, abaixo, as classes do mesmo. Quando o recurso não possui nenhum valor na propriedade rdfs:label, o URI do mesmo é exibida. Ao clicar na âncora do recurso, a tela de edição de propriedades do recursos é exibida. A

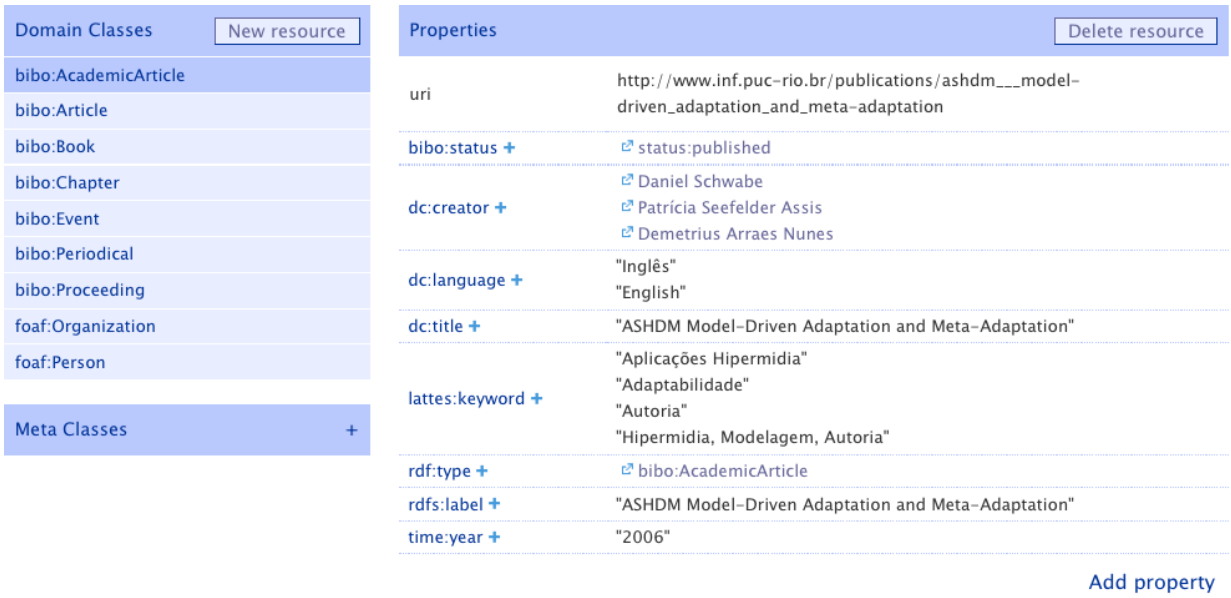


Figura 41 apresenta a tela de edição das propriedades de um recurso.

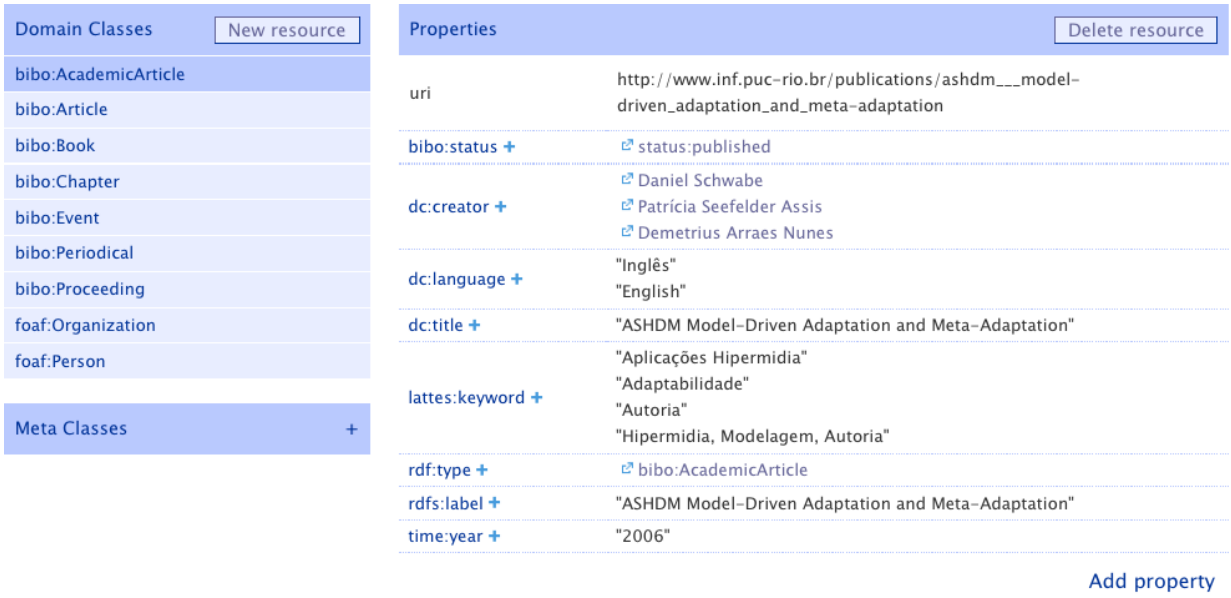


Figura 41 – Tela de edição das propriedades de um recurso

Esta tela apresenta a lista de propriedades do recurso e, para cada propriedade, os seus valores. Os valores literais são exibidos na cor preta entre aspas duplas. Os valores que são ligações para outros recursos são exibidos na cor azul, com uma imagem à esquerda que possui uma âncora para o recurso associado.

Por meio desta tela é possível alterar e remover o valor de qualquer propriedade do recurso, adicionar valores às propriedades ou adicionar novas propriedades ao recurso.

Com auxílio de campos de edição *in loco*, popularmente conhecidos como *edit-in-place*, é possível alterar e remover os valores das propriedades do recurso previamente cadastrados. A Figura 42 apresenta o detalhe da edição do valor de uma propriedade literal de um recurso, neste caso, a propriedade que será editada é dc:language e o valor é o texto "Inglês".

Figura 42 – Detalhe da edição do valor de uma propriedade

A edição de um valor de propriedade que é uma ligação para outro recurso é similar à edição de um literal, sendo que neste caso deve-se informar o URI do recurso a ser relacionado. A alteração do valor se completa quando a tecla "Enter" é pressionada.

Para remover o valor de uma propriedade, basta apagar o seu valor e pressionar a tecla "Enter". Antes da remoção do valor, um diálogo de confirmação será apresentado.

Para adicionar um novo valor à uma propriedade já cadastrada, deve-se clicar no sinal "+" ao lado do nome da propriedade que se deseja adicionar o novo valor e o campo de edição *in loco* será exibido abaixo da último valor da propriedade.

Para adicionar uma nova propriedade ao recurso, deve-se clicar na âncora "Add property". Neste caso, um novo campo será exibido no fim da lista de propriedades para a seleção da propriedade desejada. Este campo possui a funcionalidade de auto completar, ou seja, ao se digitar parte do nome de uma propriedade RDF, o campo deverá sugerir as propriedades que possuem parte do nome digitado. A Figura 43 apresenta o detalhe da adição de uma nova propriedade a um recurso, neste caso, ao informar o texto "volume", as propriedades bibo:numVolumes e bibo:volume são sugeridas.

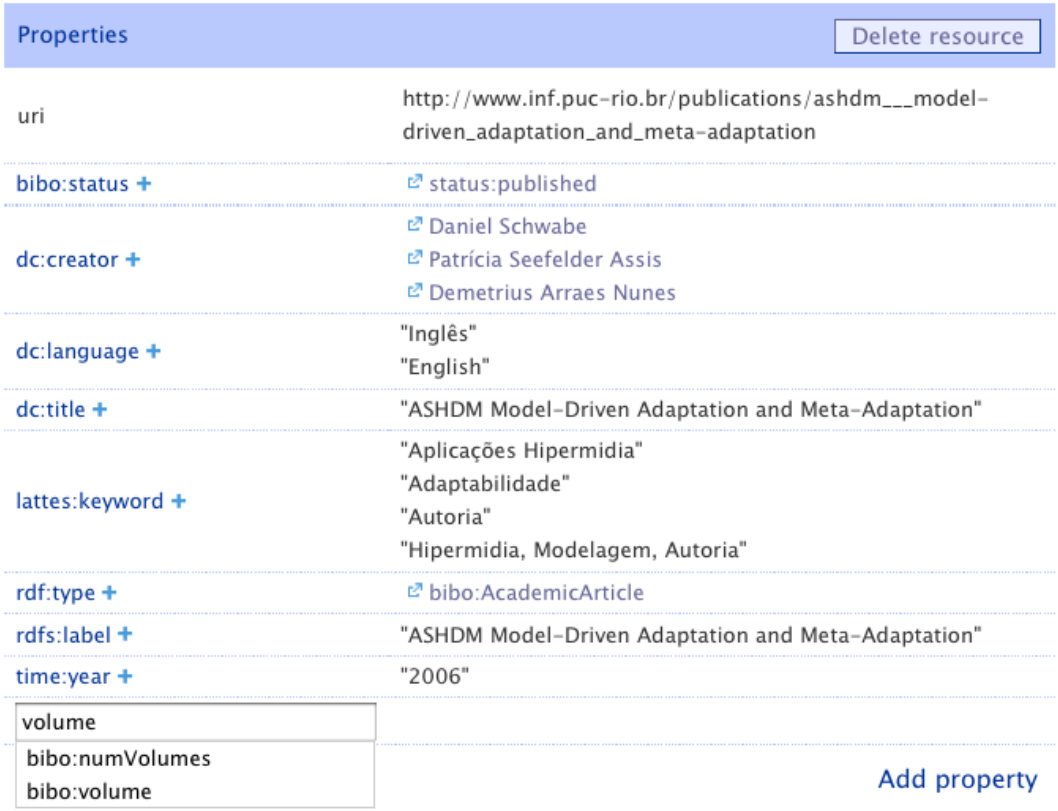


Figura 43 – Detalhe da adição de uma nova propriedade a um recurso

Após a seleção da propriedade desejada, o campo de edição *in loco* para a inserção do valor da propriedade é exibido ao lado do nome da propriedade.

Para criar um novo recurso deve-se clicar no botão "New resource" que fica sobre o menu de classes de domínio ao lado do rótulo "Domain classes" e um formulário para a criação do recurso será exibido. A Figura 44 apresenta a tela de cadastro de um novo recurso.

Figura 44 – Tela de cadastro de um novo recurso

Deve-se informar apenas o URI do novo recurso a ser criado. O campo "Namespace" presente no formulário é apenas um facilitador para sugerir os prefixos de URI associados aos *namespaces* cadastrados no Synth. Por padrão, é sugerido o *namespace* "base", porém pode-se utilizar qualquer *namespace* ou mesmo um URI cujo *namespace* não esteja presente na base de dados. Após clicar no botão "Create", o novo recurso é criado e a tela de edição de propriedades do mesmo é exibida.

Finalmente, é possível remover todo o recurso, clicando no botão "Delete resource". Um diálogo de confirmação será apresentado. Quando um recurso é removido, todas as triplas que possuem o URI do recurso no sujeito ou no objeto são removidas da base de dados.

4.4.2.4.2. Classes

Ao clicar na âncora "Classes", o editor RDF é exibido, apresentando diretamente a tela de edição de propriedades da primeira classe da lista de classes de domínio do menu da esquerda. A Figura 45 apresenta a tela de edição de propriedades de uma classe.

Como é possível observar, a tela de edição de propriedades da classe é a mesma tela de edição de propriedades de qualquer recurso, uma vez que uma classe RDF também é um recurso RDF. Desta maneira, todo o funcionamento da tela de edição de propriedades da classe se mantém consistente com a tela de

edição de propriedades de um recurso e é exatamente o mesmo apresentado na seção anterior.

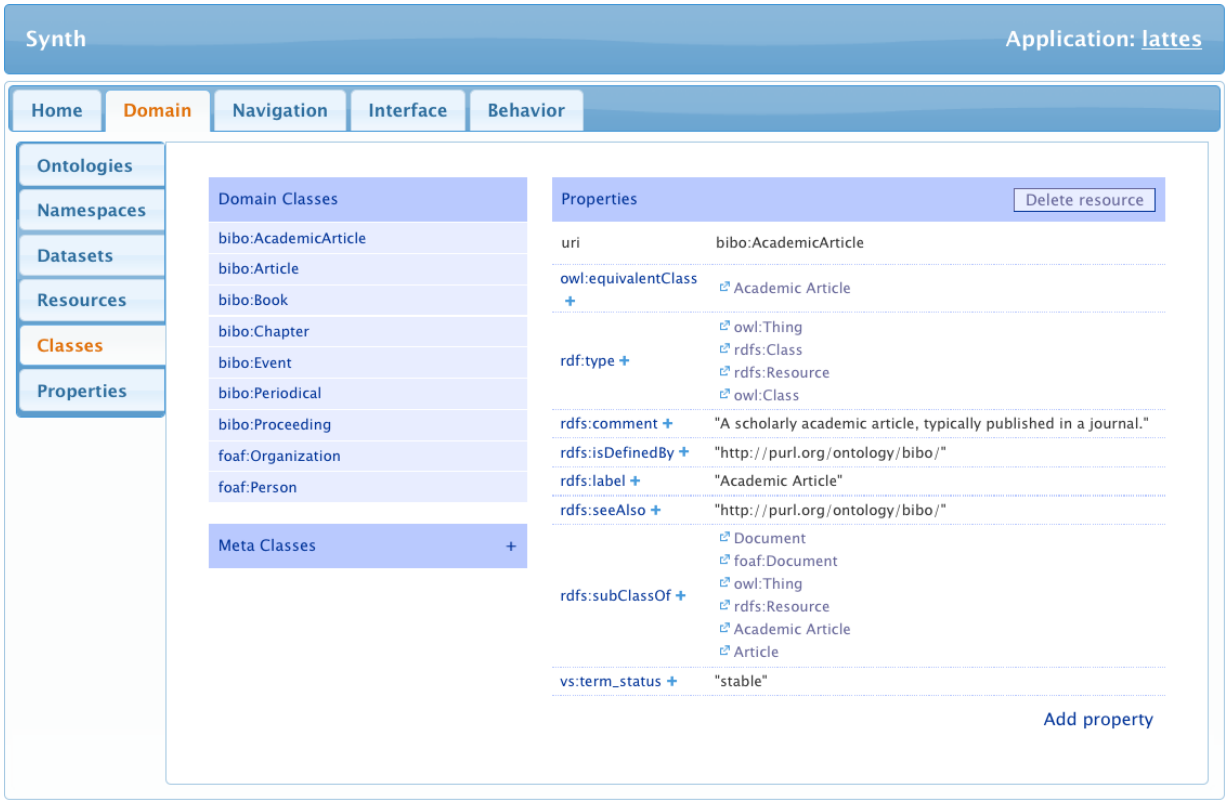


Figura 45 – Tela de edição de propriedades de uma classe

A única diferença da tela de edição de propriedades acessada a partir da âncora “Classes” em relação a mesma tela acessada a partir da âncora “Recursos”, é que as âncoras do menu de navegação de classes à esquerda sempre apresentam a tela de edição de propriedades da classe selecionada, em vez da listagem de recursos daquela classe.

4.4.2.4.3. Propriedades

Ao clicar na âncora “Properties” será exibido, no menu da esquerda, a lista de propriedades de domínio da aplicação abaixo do rótulo “Domain Properties”. À direita, é exibida a tela de edição de propriedades da propriedade selecionada.

Como uma propriedade RDF também é um recurso RDF, a tela de edição das propriedades da propriedade possui o mesmo funcionamento da tela de edição das propriedades dos recursos, assim como demonstrado com a tela de edição de

propriedades das classes na seção anterior. A Figura 46 mostra a tela de edição das propriedades de uma propriedade.

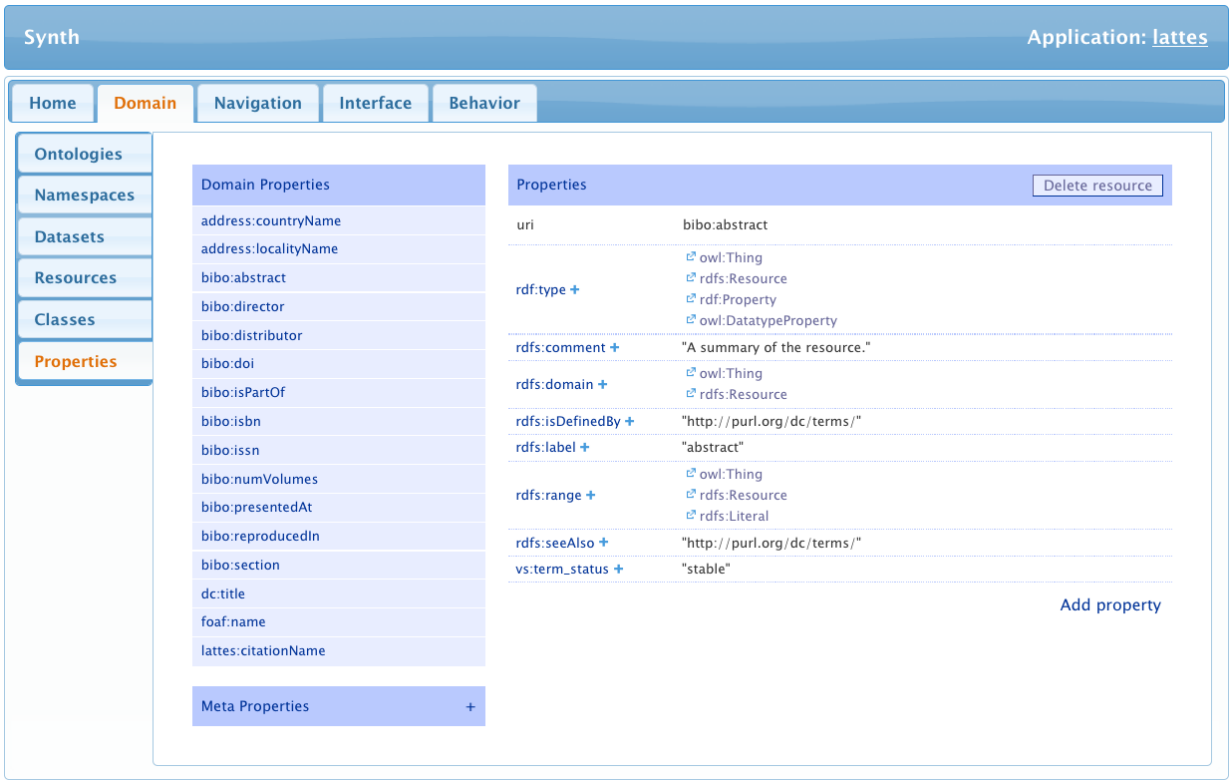


Figura 46 – Tela de edição de propriedades de uma propriedade

Além propriedades de domínio é possível listar as meta propriedades do Synth clicando no sinal "+" ao lado do rótulo "Meta Properties". As meta propriedades são todas as propriedades usadas pelos modelos manipulados pelo Synth além das propriedades básicas das linguagens RDF(S) e OWL. A grosso modo, são as propriedades agrupadas nos *namespaces* owl, rdf, rdfs, shdm, swui, synth e void. Assim como o no caso das meta classes, o funcionamento do menu de meta propriedades é idêntico ao do menu das propriedades de domínio.

4.4.3. Projeto navegacional

A tela de projeto navegacional é acessada através do item de menu "Navigation". Nesta tela, são manipuladas as primitivas do modelo de navegacional acessadas por meio dos itens de menu de segundo nível "Contexts", "Indexes", "Landmarks" e "In Context Classes". O acesso ao item "Navigation"

leva diretamente ao primeiro item de menu de segundo nível “Contexts” para a manipulação dos contextos navegacionais da aplicação.

4.4.3.1. Contextos

O gerenciamento dos contextos navegacionais da aplicação se inicia pela listagem dos contextos previamente cadastrados, conforme apresentado na Figura 47. Esta tela é apresentada ao clicar na âncora “Contexts”. A partir desta tela, é possível adicionar novos contextos, editar ou remover contextos previamente cadastrados.

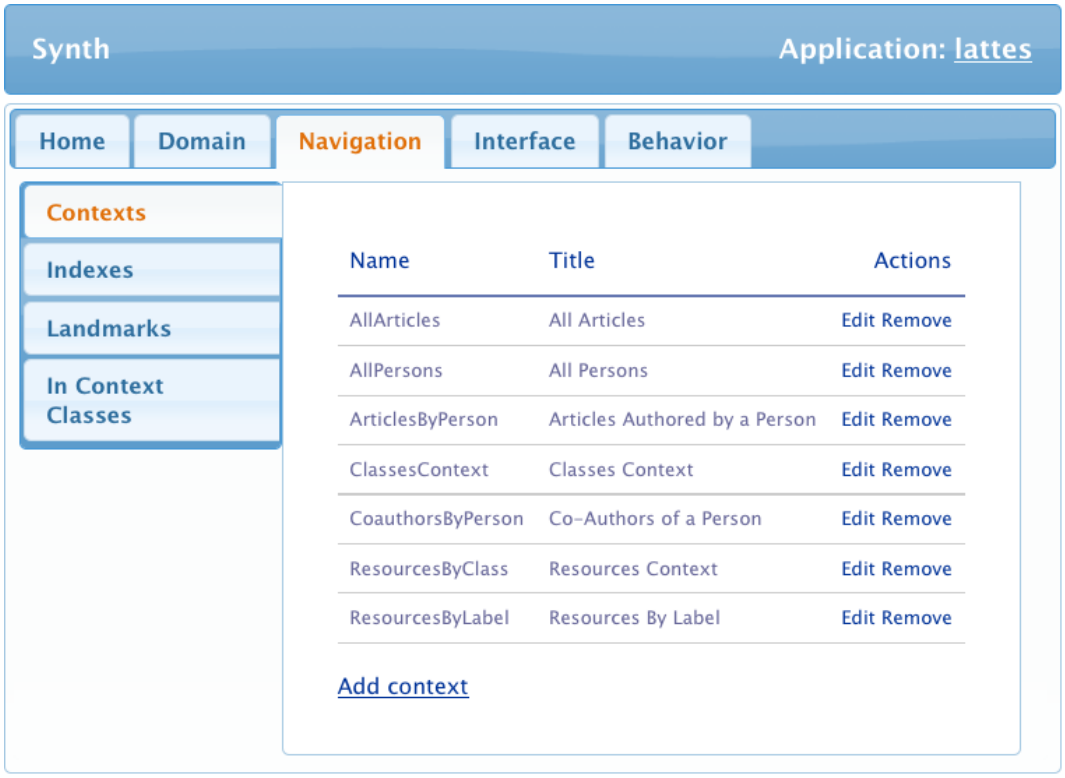


Figura 47 – Tela de listagem de contextos navegacionais

Ao clicar na âncora “Add context”, a tela de cadastro de um novo contexto navegacional será apresentada. Após preencher os campos e clicar no botão “Create”, o novo contexto será cadastrado e a tela de edição do contexto recém cadastrado será exibida.

Na tela de edição de um contexto, apresentada na Figura 48, deve-se informar um nome para o contexto, um título e uma expressão de consulta cujo

resultado deve ser um conjunto de recursos RDF que servirão de base para os nós navegacionais que participarão do contexto. A expressão de consulta deve ser escrita em uma das linguagens de consultas de contexto que serão vistas mais adiante na seção 4.4.3.1.1.

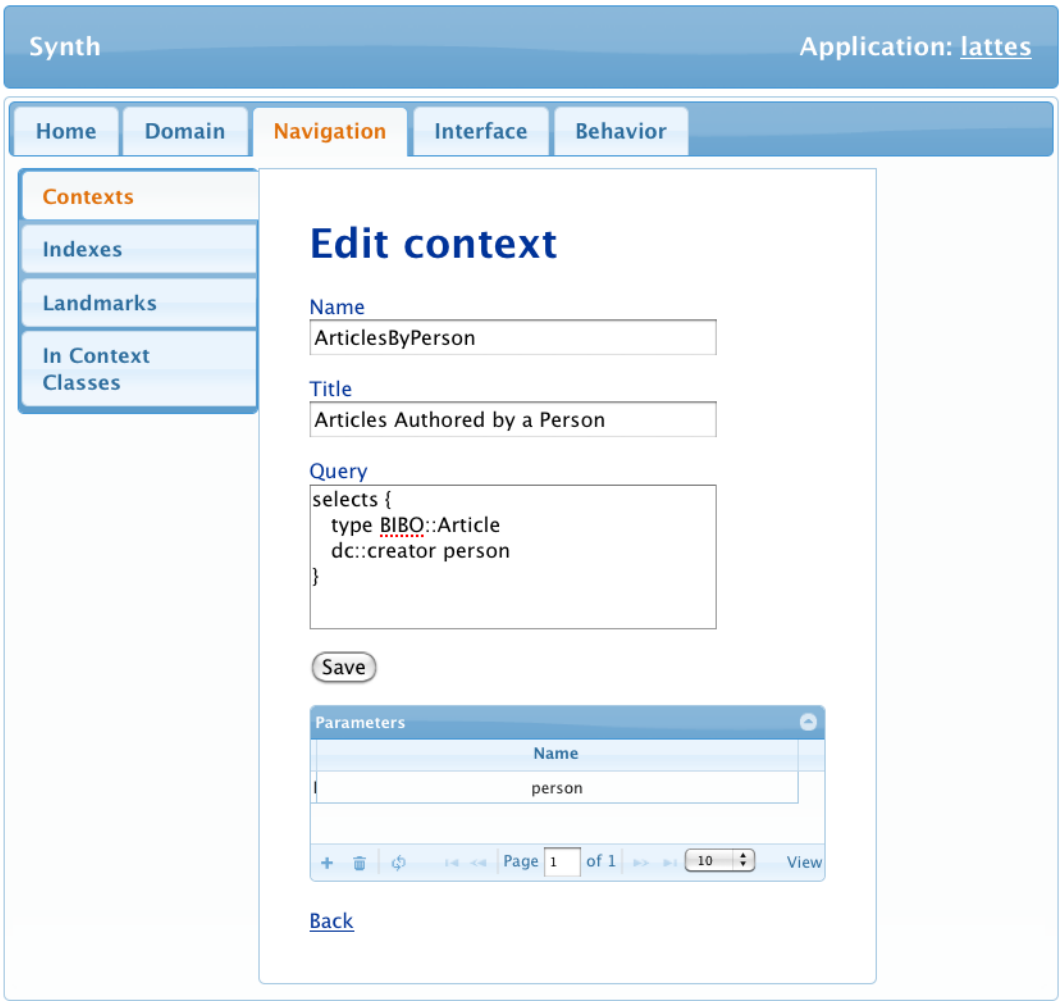


Figura 48 – Tela de edição de um contexto navegacional

Se a expressão de consulta contiver parâmetros cujos valores serão informados em tempo de execução do contexto, estes devem ser declarados, adicionando o nome do parâmetro na tabela “Parameters” abaixo do botão “Save”.

O contexto navegacional apresentado na Figura 48 é um exemplo de contexto parametrizado, onde o valor do parâmetro “person” é um recurso RDF que será usado como filtro no valor da propriedade dc:creator dos recursos do tipo bibo:Article que serão obtidos como resultado da consulta. O valor do parâmetro “person” será passado em tempo de execução na operação de navegação no contexto navegacional. Outro exemplo de contexto parametrizado seria

"PessoasPorNome", onde o valor do parâmetro "nome", passado em tempo de execução, é um texto que será usado como filtro no valor da propriedade nome dos recursos do tipo Pessoa.

Um contexto navegacional pode conter zero ou mais parâmetros, sem limite superior de parâmetros. Para adicionar um nome de parâmetro na tabela "Parameters", deve-se clicar no botão "+" presente no rodapé da tabela e um formulário para preenchimento do campo "Name" será exibido. Para editar o nome de um parâmetro, basta clicar sobre o nome e um campo de edição *in loco* será exibido. Para remover um parâmetro declarado, deve-se clicar sobre o nome e depois clicar no botão com a figura de uma lixeira presente no rodapé da tabela "Parameters".

A tabela "Parameters" é um componente *javascript* de comunicação assíncrona com o servidor. Dessa forma, ao adicionar, remover, ou editar o nome de um parâmetro, o comando é enviado para o servidor sem a necessidade de que toda a tela de edição do contexto seja atualizada.

4.4.3.1.1. Linguagens de consultas de contextos

As expressões de consultas dos contextos podem ser escritas nas linguagens SPARQL, Ruby ou SynthQL e devem sempre retornar um conjunto de recursos RDF que servirão de base para os nós navegacionais que participarão do contexto. A seguir serão abordadas cada uma das opções de sintaxe de expressões de consultas de contextos do Synth.

4.4.3.1.1.1. Expressões de consultas em SPARQL

O uso de expressões de consultas em linguagem SPARQL devem seguir a sintaxe aceita pela máquina de consultas ARQ[78], parte do *framework* Jena, que é umas das implementações de maior conformidade com a especificação[79] da linguagem SPARQL definida pelo W3C. A máquina de consultas ARQ também

[78] http://jena.sourceforge.net/ARQ
[79] http://www.w3.org/TR/rdf-sparql-query

implementa algumas das novas funcionalidades da versão 1.1[80] da linguagem SPARQL que ainda estão em fase de desenvolvimento, em particular, a consulta em bases de dados SPARQL federadas[81].

Na especificação do contexto, a expressão de consulta em linguagem SPARQL deve ser uma cadeia de caracteres válida na linguagem Ruby, podendo estar presente entre aspas simples, aspas duplas ou entre os delimitadores %{}. Para que os caracteres aspas simples e aspas duplas fiquem livres para uso na própria expressão de consulta, recomenda-se o uso dos delimitadores %{}. O Quadro 18 ilustra um exemplo de consulta de contexto em linguagem SPARQL.

```
%{
   PREFIX  rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
   PREFIX  foaf: <http://xmlns.com/foaf/0.1/>
   SELECT DISTINCT ?s WHERE { ?s rdf:type foaf:Person . }
}
```

Quadro 18 – Exemplo de expressão de consulta de contexto em linguagem SPARQL

Esta expressão de consulta deve retornar todos os recursos do tipo foaf:Person.

O trecho SELECT DISTINCT ?s no início da expressão garante que o resultado será um conjunto de recursos RDF e é um padrão comum entre as expressões de consultas de contexto em linguagem SPARQL.

Quando é necessário combinar a expressão de consulta em SPARQL com o valor de um parâmetro do contexto, deve-se interpolar a cadeia de caracteres da expressão com a variável que carrega o valor do parâmetro. Esta variável tem o mesmo nome do parâmetro.

```
%{
   PREFIX  rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
   PREFIX  bibo: <http://purl.org/ontology/bibo/>
   PREFIX  dc:   <http://purl.org/dc/terms/>

   SELECT DISTINCT ?s WHERE { ?s rdf:type bibo:Article .
                              ?s dc:creator <%s>  .
                              ?s dc:language "%s" .}
} % [ person, language ]
```

Quadro 19 – Exemplo de expressão de consulta SPARQL com um parâmetro de contexto

[80] http://www.w3.org/TR/sparql11-query
[81] http://jena.sourceforge.net/ARQ/service.html

O Quadro 19 ilustra um exemplo de expressão de consulta em SPARQL por artigos de uma pessoa em inglês, onde a pessoa e o idioma são, respectivamente, um recurso RDF e um texto passados como valores para os parâmetros “person” e “language”. Os símbolos %s são trocados pelos valores das variáveis “person” e “language” na cadeia de caracteres resultante da interpolação.

4.4.3.1.1.2. Expressões de consultas em Ruby

As expressões de consultas de contextos na linguagem Ruby são, na verdade, expressões da DSL de manipulação de recursos RDF fornecida pela biblioteca ActiveRDF. O Quadro 20 ilustra alguns exemplos de expressões de consultas feitas com a DSL do ActiveRDF.

```
FOAF::Person.find_all

FOAF::Person.alpha(FOAF::name)

FOAF::Person.find_by.foaf::name("john").execute

brazil = Resource.new("http://dbpedia.org/resource/Brazil")
FOAF::Person.find_by.foaf::based_near(brazil).execute

FOAF::Person.find_by.foaf::name("john").foaf::based_near(brazil).execute

BIBO::Article.find_by.dc::creator(person).execute
```

Quadro 20 – Exemplos de consultas de contexto com a DSL do ActiveRDF

A lista a seguir detalha o significado de cada uma das expressões do Quadro 20 na ordem em que aparecem no quadro:

1. Obter todos os recursos do tipo foaf:Person;
2. Obter todos os recursos do tipo foaf:Person em ordem alfabética crescente do valor da propriedade foaf:name.
3. Obter todos os recursos do tipo foaf:Person que contenham, no valor da propriedade foaf:name, o texto “john";
4. Obter todos os recursos do tipo foaf:Person que contenham, no valor da propriedade foaf:based_near, o recurso descrito pelo URI http://dbpedia.org/resource/Brazil;
5. Obter todos os recursos do tipo foaf:Person que contenham, no valor da propriedade foaf:name, o texto “john" e, na propriedade

foaf:based_near, o recurso descrito pelo URI http://dbpedia.org/resource/Brazil.

6. Obter todos os recursos do tipo bibo:Article que contenham no valor da propriedade dc:creator o recurso passado como valor do parâmetro do contexto “person”.

A única restrição para o uso de uma expressão de consulta na linguagem Ruby é que o resultado de sua avaliação retorne um vetor de recursos RDF. Ou seja, um objeto Ruby do tipo Array contendo instâncias da classe RDFS::Resource ou suas subclasses.

É possível tirar proveito da expressividade da linguagem Ruby e de suas bibliotecas padrões para obter conjuntos de recursos RDF que não estariam disponíveis apenas pelo uso da DSL padrão do ActiveRDF.

O exemplo do Quadro 21 ilustra a combinação da DSL do ActiveRDF com vários métodos da biblioteca Array da linguagem Ruby. Neste exemplo, deseja-se obter os coautores de uma pessoa. Como o modelo de domínio utilizado não possui o conceito de coautor, optou-se por obter este resultado pelo conjunto de autores das mesmas publicações da pessoa em questão.

```
articles = BIBO::Article.find_by.dc::creator(person).execute
authors = articles.map{|a| a.dc::creator}
authors = authors.flatten.uniq
co_authors = authors.delete_if{|p| p == person}
co_authors = co_authors.sort_by{|co_author| co_author.foaf:name}
```

Quadro 21 – Exemplo de combinação da DSL do ActiveRDF com métodos Ruby

A lista a seguir explica de cada uma das expressões do Quadro 21 na ordem em que aparecem no quadro:

1. Obtém todos os artigos de uma pessoa. Mais especificamente, um vetor de todos os recursos do tipo bibo:Article que possuem como valor da propriedade dc:creator o recurso passado como valor para o parâmetro “person”;
2. Obtém todos os autores dos artigos obtidos no passo anterior. Mais especificamente, gera um novo vetor de vetores onde, para cada recurso

do vetor “articles” se obtém o vetor resultante da avaliação da propriedade dc:creator;

3. Planifica o vetor de vetores em um vetor de autores e elimina os autores duplicados;
4. Remove a pessoa em questão do vetor de autores;
5. Ordena o vetor pela propriedade foaf:name;

O ActiveRDF também dispõe do módulo ActiveRDF::Query para consultas na base de dados RDF com expressões similares à linguagem SPARQL. O Quadro 22 é um exemplo de consulta com o módulo AcitveRDF::Query. Neste exemplo, a expressão retorna todos os recursos do tipo foaf:Person, que possuem o texto ‘joao’ como parte do valor da propriedade foaf:name, ordenados pela propriedade foaf:name, limitados em 100 resultados e a consulta deve ser feita no repositório local e naquele cujo título é “dbpedia”.

```
q = ActiveRDF::Query.new.select(:s)
q = q.where(:s, RDF::type, FOAF::Person)
q = q.regexp(FOAF::name, /joao/)
q = q.sort(FOAF::name)
q = q.limit(100)
q = q.datasets(:local, :dbpedia)
q.execute
```

Quadro 22 – Exemplo de expressão de consulta com o módulo AcitveRDF::Query

As expressões de consultas geradas a partir do módulo ActiveRDF::Query são feitas no nível da tripla RDF, assim como na linguagem SPARQL. Esta opção de consulta oferece recursos que não estão disponíveis na DSL padrão do ActiveRDF como filtros, ordenação, limite entre outros que também estão disponíveis na linguagem SPARQL. A cláusula *datasets* foi adicionada no Synth e não está presente na biblioteca ActiveRDF padrão. Mais informações sobre a sintaxe de consultas do ActiveRDF, estão disponíveis na documentação da biblioteca no site www.activerdf.org.

4.4.3.1.1.3. Expressões de consultas em SynthQL

A linguagem de consultas para contextos SynthQL foi criada com o objetivo de abstrair a sintaxe da linguagem SPARQL para as expressões mais comuns.

Essa abordagem procura minimizar a necessidade de conhecimento da linguagem SPARQL, cobrindo a grande maioria das expressões que são necessárias a uma aplicação SHDM.

O Quadro 23 ilustra um exemplo de expressão de consulta de contexto com a linguagem SynthQL. Neste exemplo, a expressão retorna todos os recursos do tipo foaf:Person, que possuem o texto 'joao' como parte do valor da propriedade foaf:name, ordenados pela propriedade foaf:name, limitados em 100 resultados e a consulta deve ser feita no repositório local e naquele cujo título é "dbpedia".

```
selects {
  type FOAF::Person
  foaf::name like "joao"
  order FOAF::name
  limit 100
  datasets :local, :dbpedia
}
```

Quadro 23 – Exemplo de expressão de consulta na linguagem SynthQL

A linguagem SynthQL manipula os elementos da DSL da biblioteca ActiveRDF para fornecer uma sintaxe mais simples, porém com resultados equivalentes aos obtidos com o uso do módulo ActiveRDF::Query. Por isso, os símbolos usados para representar os recursos RDF na expressão de consulta em SynthQL são os mesmos usados na DSL do ActiveRDF.

A sintaxe da linguagem SynthQL é bem simples. A expressão sempre começa com *selects*, seguido de um par chaves. Dentro das chaves devem ser informadas, em qualquer ordem, as cláusulas reservadas e os pares propriedade/valor que serão utilizados como condições da consulta. A Tabela 5 lista as cláusulas reservadas e suas descrições.

Cláusula reservada	Descrição
type	Indica o tipo do recurso a ser retornado pela consulta. Deve ser informada uma classe RDF. Esta cláusula é única obrigatória. Exemplo: type FOAF::Person
a	Equivalente a type. Pode ser utilizado opcionalmente.
order	Indica as propriedades utilizadas para a ordenação do conjunto de recursos em ordem crescente. Deve ser informada uma ou mais propriedades RDF, separadas por vírgula.

	Exemplo: order FOAF::name, FOAF::age
reverse_order	Similar a order, porém retorna os resultados em ordem decrescente.
limit	Limita o resultado da consulta. Deve ser informado um número inteiro. Exemplo: limit 100
datasets	Indica em quais repositórios a consulta deve ser feita. Deve receber um ou mais títulos de repositórios precedidos de dois pontos. Exemplo: datasets :local, :dbpedia

Tabela 5 – Cláusulas reservadas da linguagem SynthQL

Na expressão de consulta em SynthQL, tudo que não é uma cláusula reservada, é um par propriedade/valor. Os pares propriedade/valor formam a cláusula *where* da consulta e o uso de dois ou mais pares propriedade/valor na consulta formam a conjunção lógica desses pares na condição.

A propriedade do par propriedade/valor deve ser informada na sintaxe de propriedades do método find_by do ActiveRDF, ou seja, namespace::localname. Por exemplo foaf::name. O valor pode ser um literal (cadeia de caracteres ou número), uma variável ou uma constante Ruby. Os recursos RDF na sintaxe do ActiveRDF são constantes Ruby. Por exemplo FOAF::Person e FOAF::name.

Os pares propriedade/valor também aceitam a cláusula *like*, presente no valor do par, indicando que a condição é pela parte do texto no valor da propriedade, de modo similar a cláusula LIKE da linguagem de consulta SQL para bancos de dados relacionais. A cláusula *like* só pode ser usada quando o valor indicado no par propriedade/valor é uma cadeia de caracteres. O exemplo do Quadro 23 ilustra o uso da cláusula *like*.

Um par propriedade/valor pode ter o símbolo :o no lugar do valor, indicando que a propriedade do par deve estar presente nos recursos resultantes, não importando o seu valor.

O Quadro 24 ilustra um exemplo de consulta pela presença da propriedade foaf:name, nos recursos do tipo foaf:Person, não importando o seu valor.

```
selects {
  type FOAF::Person
  foaf::name :o
}
```

Quadro 24 – Exemplo de consulta pela presença da propriedade foaf:name

Também é possível usar valores de parâmetros dos contextos nas condições da expressão de consulta, pois esses valores são variáveis Ruby. O Quadro 25 ilustra um exemplo de expressão de consulta em SynthQL que usa um valor de parâmetro como parte da condição. Neste caso, a consulta retorna todos os recursos do tipo FOAF::Person cujo valor da propriedade foaf:age é igual ao valor do parâmetro do contexto "age".

```
selects {
 type FOAF::Person
 foaf::age age
}
```

Quadro 25 – Exemplo de consulta SynthQL usando um valor de parâmetro de contexto

4.4.3.2. Índices

O gerenciamento dos índices navegacionais da aplicação se inicia pela listagem dos índices previamente cadastrados, conforme apresentado na Figura 49. Esta tela é apresentada ao clicar na âncora "Indexes" e funciona de maneira similar à tela de listagem de contextos navegacionais.

Figura 49 – Tela de listagem de índices navegacionais

Ao clicar na âncora “Add index”, a tela de cadastro de um novo índice navegacional será apresentada. Após preencher os campos e clicar no botão “Create”, o novo contexto será cadastrado e a tela de edição do contexto recém cadastrado será exibida.

Na tela de edição de um índice, apresentada na Figura 50, deve-se informar um nome para o índice, um título e o contexto no qual o índice é baseado.

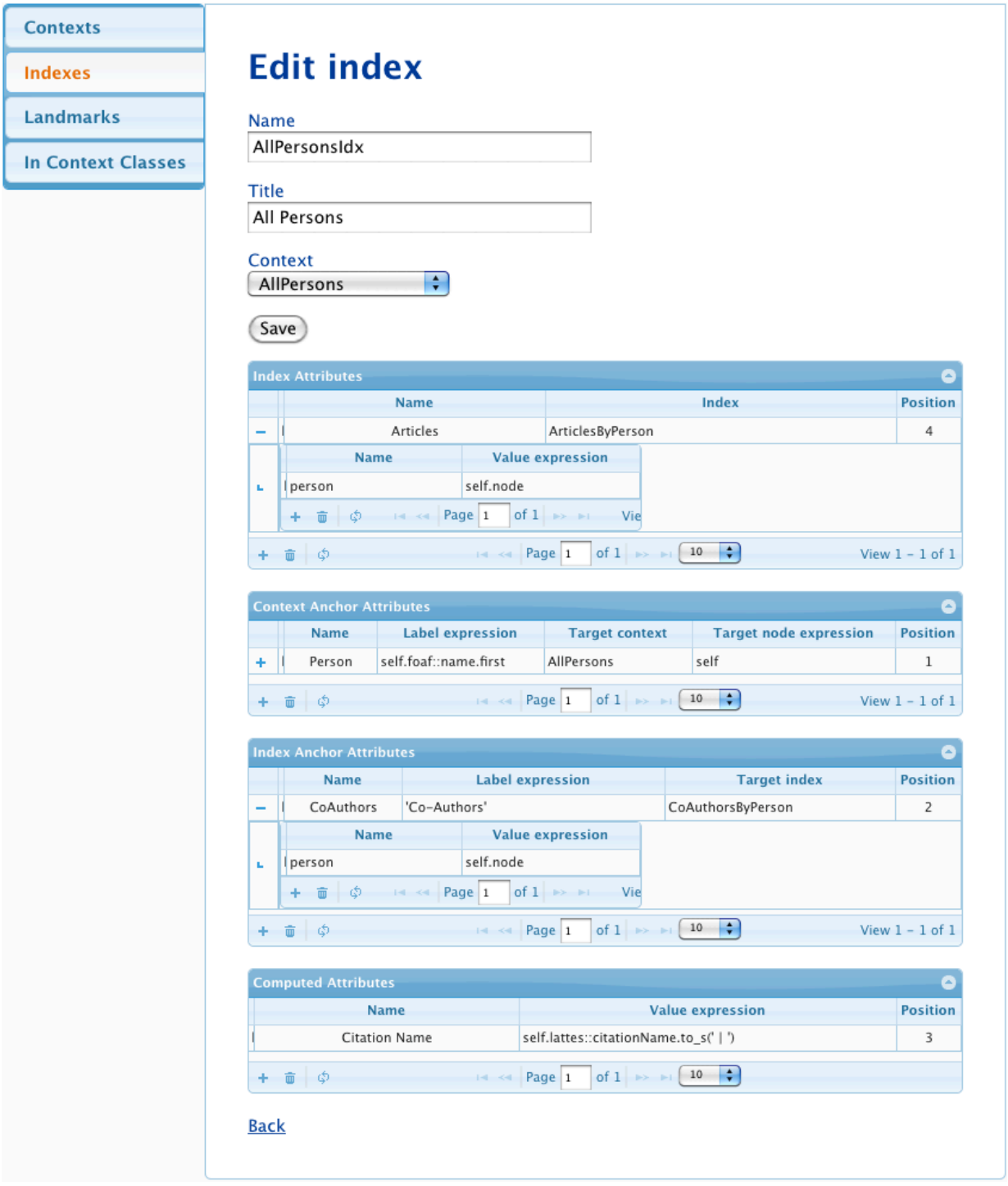


Figura 50 – Tela de edição de um índice navegacional

Para completar a edição do índice, deve-se informar os seus atributos navegacionais nas tabelas “Index Attributes”, “Context Anchor Attributes”, “Index Anchor Attributes” e “Computed Attributes”. Essas tabelas usam o mesmo componente *javascript* da tabela “Parameters” da tela de edição de contextos e funcionam de forma similar.

Em cada uma das tabelas da tela de edição do índice, deve-se informar os atributos navegacionais pelo seu tipo, que é o mesmo descrito pelo título da tabela. Um índice navegacional deve ter pelo menos um atributo navegacional do tipo âncora para contexto ou âncora para índice.

A Figura 51 é um detalhe do modo de edição da tabela de atributos do tipo índice.

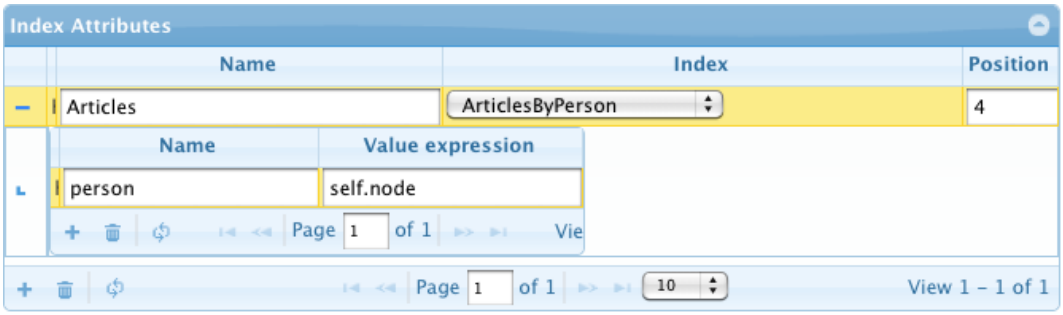


Figura 51 – Detalhe da tabela de atributos do tipo índice

A lista a seguir detalha o significado de cada campo a ser preenchido na tabela de atributos do tipo índice:

1. *Name*: nome do atributo navegacional. Este nome poderá ser usado no cabeçalho da interface do índice navegacional;
2. *Index*: um índice navegacional. Este índice será renderizado na interface como um atributo da entrada do índice que está sendo editado. Na Figura 54, o item “d” ilustra o efeito do uso de um atributo do tipo índice;
3. *Position*: posição em que o atributo navegacional deve ser exibido no índice. Os atributos são exibidos no índice pela ordem crescente de sua posição em relação aos outros atributos.

A Figura 51 também apresenta o detalhe do sub-formulário para edição dos parâmetros a serem enviados para o índice selecionado, para os casos em que este

é parametrizado. Neste sub-formulário, deve-se informar o nome do parâmetro no campo *Name* e uma expressão de valor no campo *Value Expression*. Essas expressões são descritas em linguagem Ruby e devem retornar um valor compatível com o parâmetro esperado pelo índice selecionado, podendo-se utilizar a DSL do ActiveRDF. A expressão de valor é avaliada no escopo de execução de cada entrada do índice. No exemplo da Figura 51, *self.node* é o nó navegacional no qual se baseia a entrada do índice.

A Figura 52 é um detalhe do modo de edição da tabela de atributos do tipo âncora para contexto.

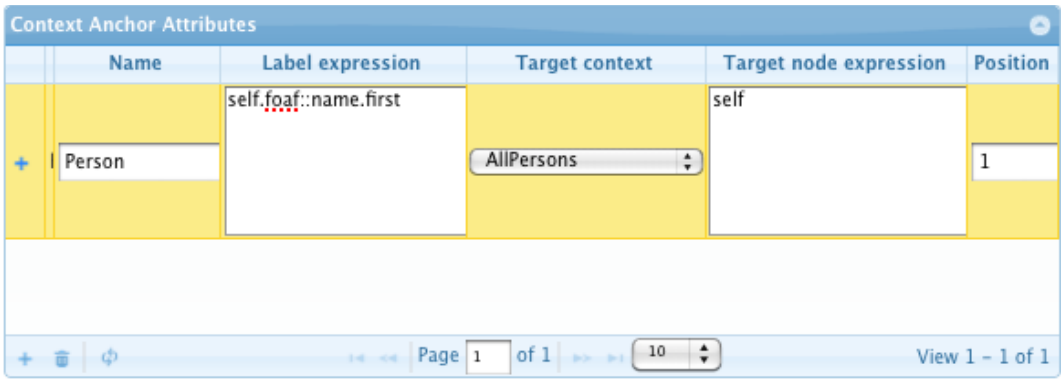


Figura 52 – Detalhe da tabela de atributos do tipo âncora para contexto

A lista a seguir detalha o significado de cada campo a ser preenchido na tabela de atributos do tipo âncora para contexto. Os campos omitidos na lista tem o mesmo significado dos campos de mesmo nome na tabela "Index Attributes":

1. *Label expression*: deve ser informada uma expressão em linguagem Ruby que deverá retornar uma cadeia de caracteres para o rótulo da âncora para o contexto. A expressão é avaliada no escopo de execução de cada entrada do índice. Sendo assim, o símbolo *self* refere-se à própria entrada do índice. A expressão ainda pode usar a DSL do ActiveRDF para obter os valores das propriedades das entradas do índice. No exemplo da Figura 52, *self.foaf::name.first* retorna o primeiro valor do atributo foaf:name do nó navegacional no qual a entrada de índice é baseado;
2. *Target context*: contexto alvo da âncora;
3. *Target node expression*: expressão em Ruby que deve retornar um recurso RDF que representa o nó que será acessado na âncora para o contexto. Sua avaliação é idêntica a de *Label expression*.

Quando o contexto destino é parametrizado, deve-se informar os nomes dos parâmetros e suas expressões de valores para a formação da âncora para o contexto. Para isso deve-se clicar no sinal de "+" ao lado do nome do atributo e o sub-formulário de parâmetros será exibido. Este sub-formulário tem o mesmo funcionamento do sub-formulário apresentado tabela "Index Attributes".

A Figura 53 é um detalhe do modo de edição da tabela de atributos do tipo âncora para índice.

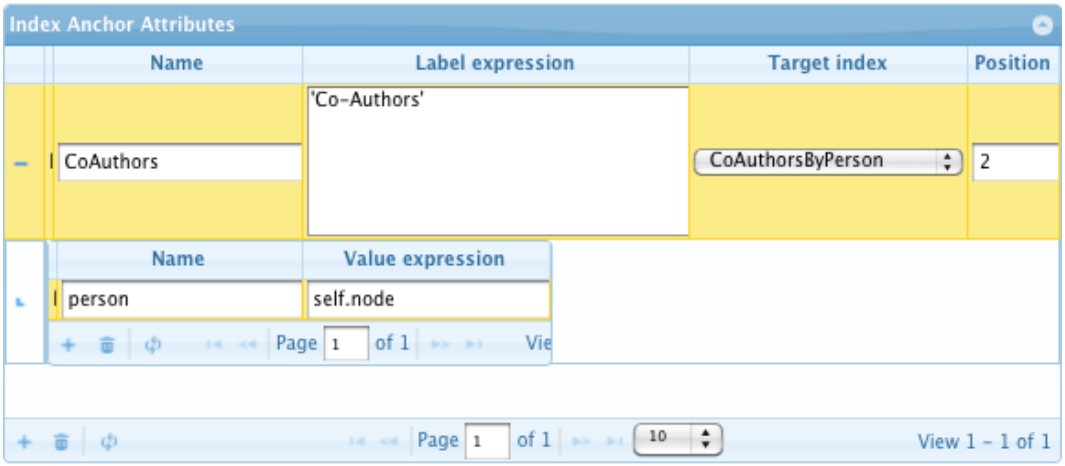


Figura 53 – Detalhe da tabela de atributos do tipo âncora para índice

Exceto pelo campo *Target index*, todos os campos da tabela "Index Anchor Attributes" têm significado idêntico aos de mesmo nome descritos nas tabelas apresentadas anteriormente. O campo *Target index* é o índice alvo usado na formação da âncora para o índice.

Quando o index destino é baseado em um contexto parametrizado, deve-se informar os nomes dos parâmetros e suas expressões de valores para a formação da âncora para o índice. Assim como na tabela "Context Anchor Attribute" o sub-formulário para edição dos parâmetros tem o mesmo funcionamento do sub-formulário apresentado na tabela "Index Attributes".

Nos sub-formulários de edição de parâmetros das três tabelas apresentadas até agora é possível reaproveitar os parâmetros que foram passados para o índice editado através do símbolo parameters[:nome_do_parametro]. Exemplo: considerando que o índice editado é "Idx Artigos por pessoa" baseado no contexto "Artigos por pessoa" que requer o parâmetro "pessoa". Se este índice possui um atributo do tipo âncora para o contexto no qual é baseado, este atributo deve conter a descrição do parâmetro "pessoa" e uma expressão de valor. A expressão

de valor pode ser parameters[:pessoa], que significa que a âncora resultante será formada, também, pelo valor do parâmetro "pessoa" que foi recebido originalmente pelo índice "Idx Artigos por pessoa".

Os atributos computados são descritos na tabela "Computed atributes", onde devem ser descritos apenas o nome, uma expressão de valor e a posição. Os campos *Name* e *Position* têm o mesmo significado dos campos de mesmo nome nas tabelas apresentadas anteriormente. A expressão de valor (*Value expression*) deve conter uma expressão Ruby, avaliada no escopo de execução de cada entrada de índice, que formará o valor do atributo computado e pode usar a DSL do ActiveRDF.

A Figura 54 é a tela resultante da execução do índice AllPersonsIdx, apresentado nos exemplos vistos até agora.

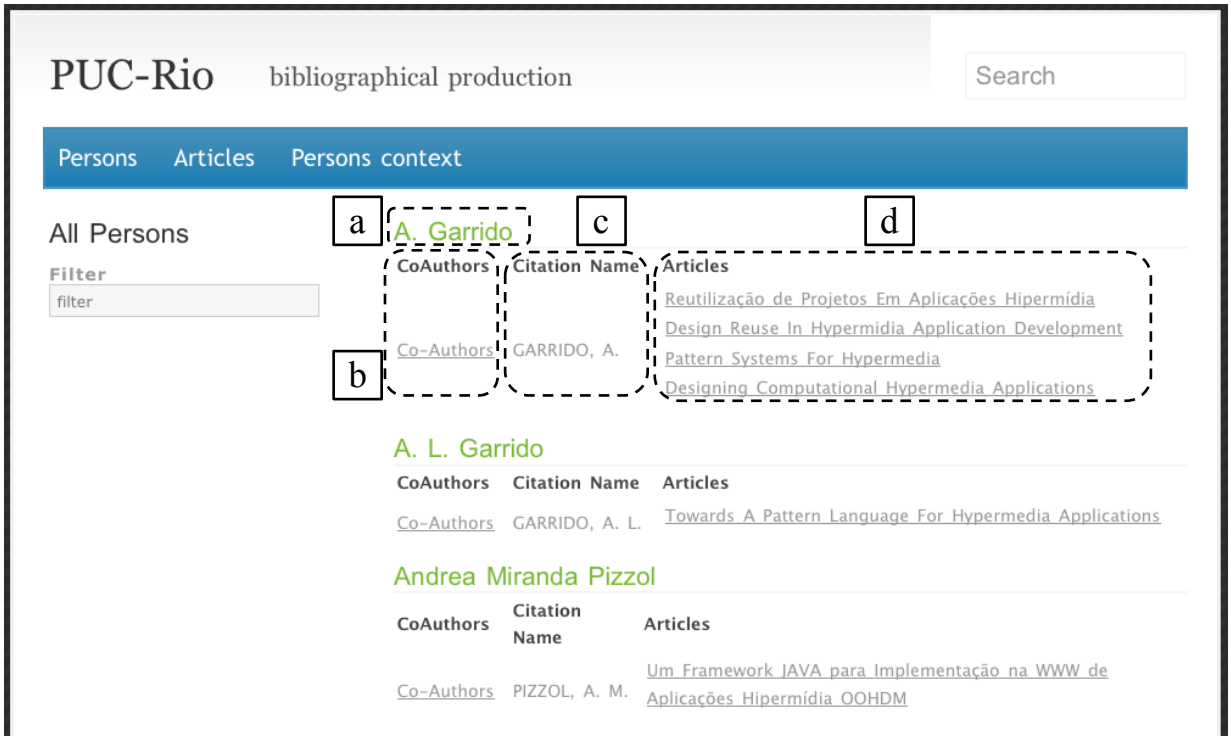


Figura 54 – Resultado do índice AllPersonsIdx

A lista a seguir descreve cada atributo navegacional destacado na Figura 54:

a. Atributo âncora para contexto "Person", apresentado na Figura 52;
b. Atributo âncora para índice "CoAuthors", apresentado na Figura 53;
c. Atributo computado "Citation Name";
d. Atributo do tipo índice "Articles", apresentado na Figura 51.

4.4.3.3. Landmarks

O gerenciamento dos *landmarks* da aplicação se inicia pela listagem dos *landmarks* previamente cadastrados, conforme apresentado na Figura 55.

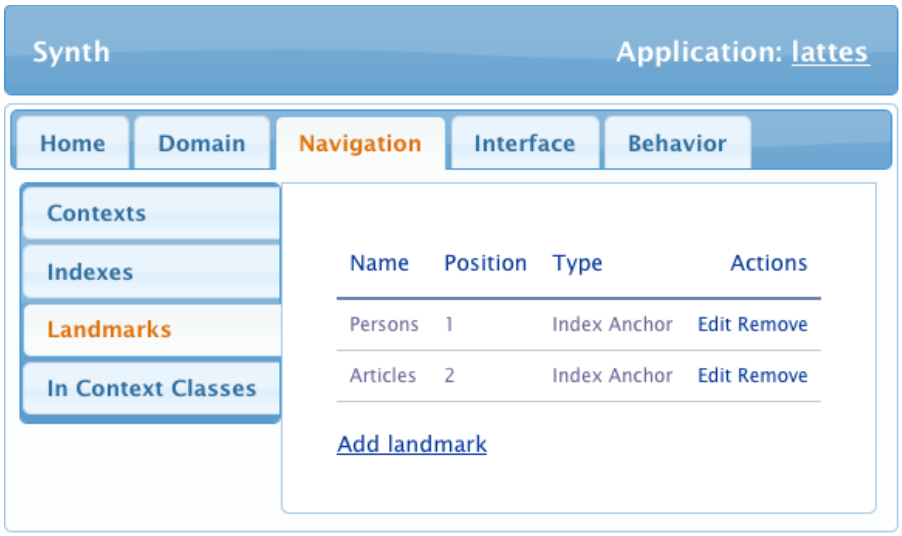


Figura 55 – Tela de listagem de *landmarks*

Esta tela é apresentada ao clicar na âncora "Landmarks" e funciona de maneira similar à tela de listagem de contextos navegacionais.

Existem 2 tipos de *landmarks* suportados pelo Synth: âncora para índice e âncora para contexto. No formulário de criação de um *landmark*, deve-se primeiro selecionar o tipo de *landmark* e a interface apropriada será apresentada.

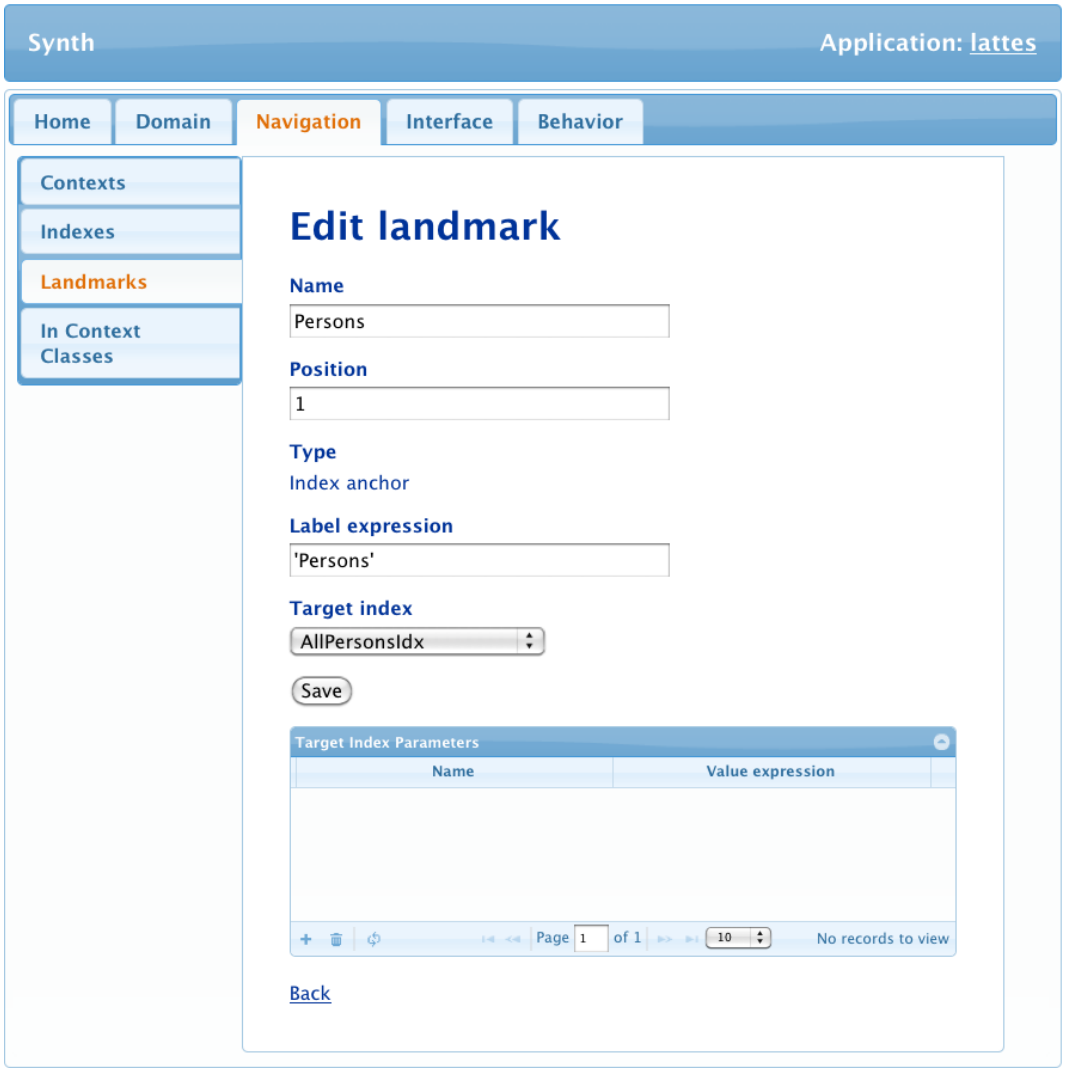


Figura 56 – Tela de edição de *landmark* do tipo âncora para índice

Para editar um *landmark* do tipo âncora para índice, deve-se informar o seu nome, posição em relação aos outros *landmarks* da aplicação, a expressão de rótulo e o índice alvo da âncora. Opcionalmente podemos fornecer um conjunto de parâmetros na tabela "Target Index Parameters" que serão usados na geração da âncora para o índice parametrizado. Tanto a expressão de rótulo como a expressão de valor do parâmetro são avaliadas no escopo de execução do *landmark*.

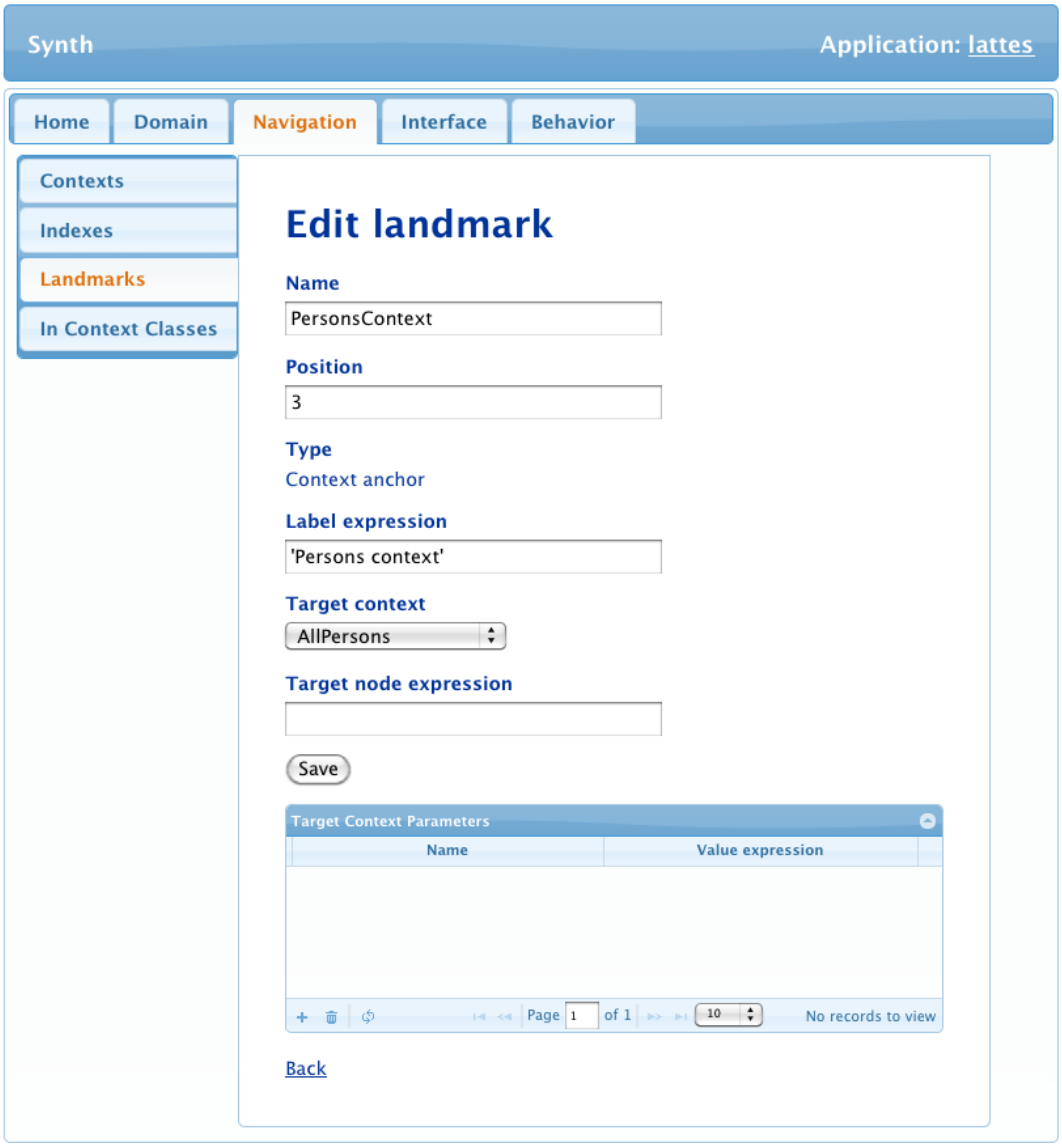


Figura 57 – Tela de edição de *landmark* do tipo âncora para contexto

A edição do *landmark* do tipo âncora para contexto é similar à edição do *landmark* do tipo âncora para índice com a diferença de que deve-se informar um contexto alvo e uma expressão para o nó alvo do contexto. Essa expressão também é avaliada no escopo de execução do *landmark*. Quando a expressão do no alvo não está presente, o primeiro nó do contexto é selecionado automaticamente.

4.4.3.4. Classes em contexto

O gerenciamento das classes em contexto da aplicação se inicia pela listagem das classes em contexto previamente cadastradas, conforme apresentado na Figura 58.

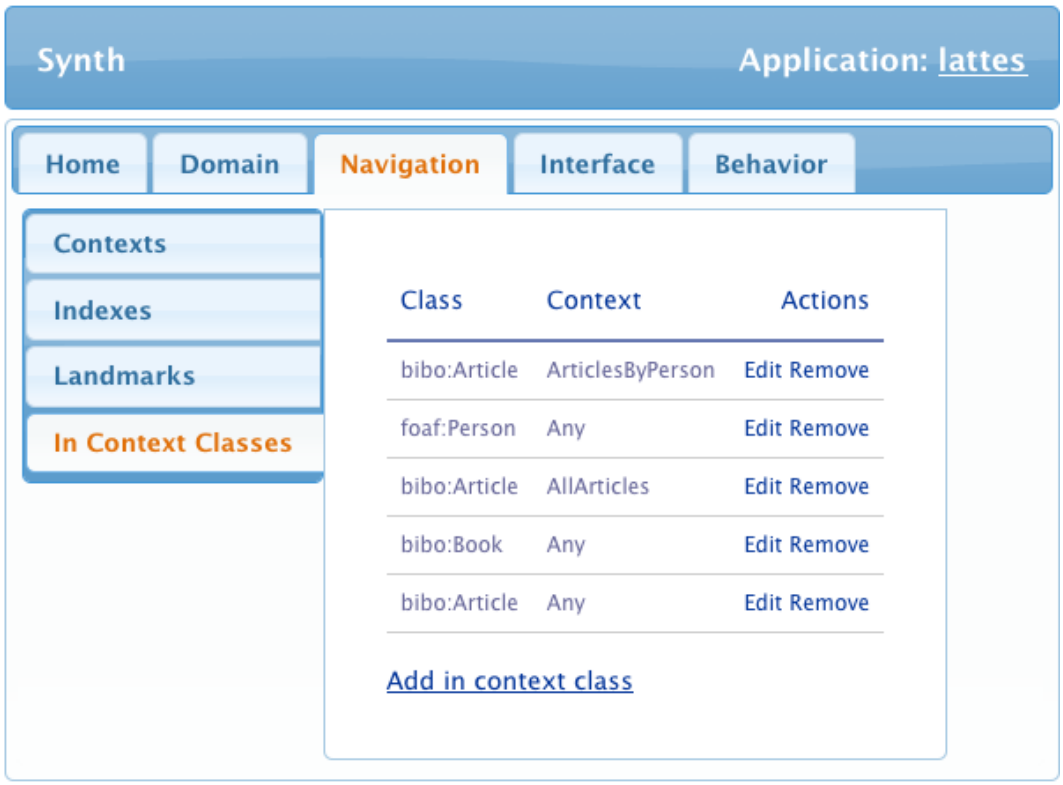


Figura 58 – Tela de listagem de classes em contexto

Esta tela é apresentada ao clicar na âncora “In Context Classes” e funciona de maneira similar as telas de listagem de listagem vistas até agora.

Ao clicar na âncora “Add in context class”, a tela de cadastro de uma nova classe em contexto será apresentada. Após preencher os campos e clicar no botão “Create”, a nova classe em contexto contexto será cadastrada e a tela de edição será exibida.

Na tela de edição de uma classe em contexto, apresentada na Figura 59, deve-se informar uma classe e um contexto, de modo que os nós da classe selecionada apresentarão os atributos navegacionais especificados nesta mesma tela quando forem acessados a partir do contexto selecionado. Caso se deseje especificar atributos navegacionais para os nós de uma classe independentemente do contexto acessado, basta selecionar a opção “Any”, na caixa de seleção do

contexto. Nesta mesma tela, o link “Show meta classes” disponibiliza, na caixa de seleção de classes, as meta classes do Synth que são as mesmas descritas na seção 4.4.2.4.1, possibilitando que se adicione atributos navegacionais às meta classes do ambiente.

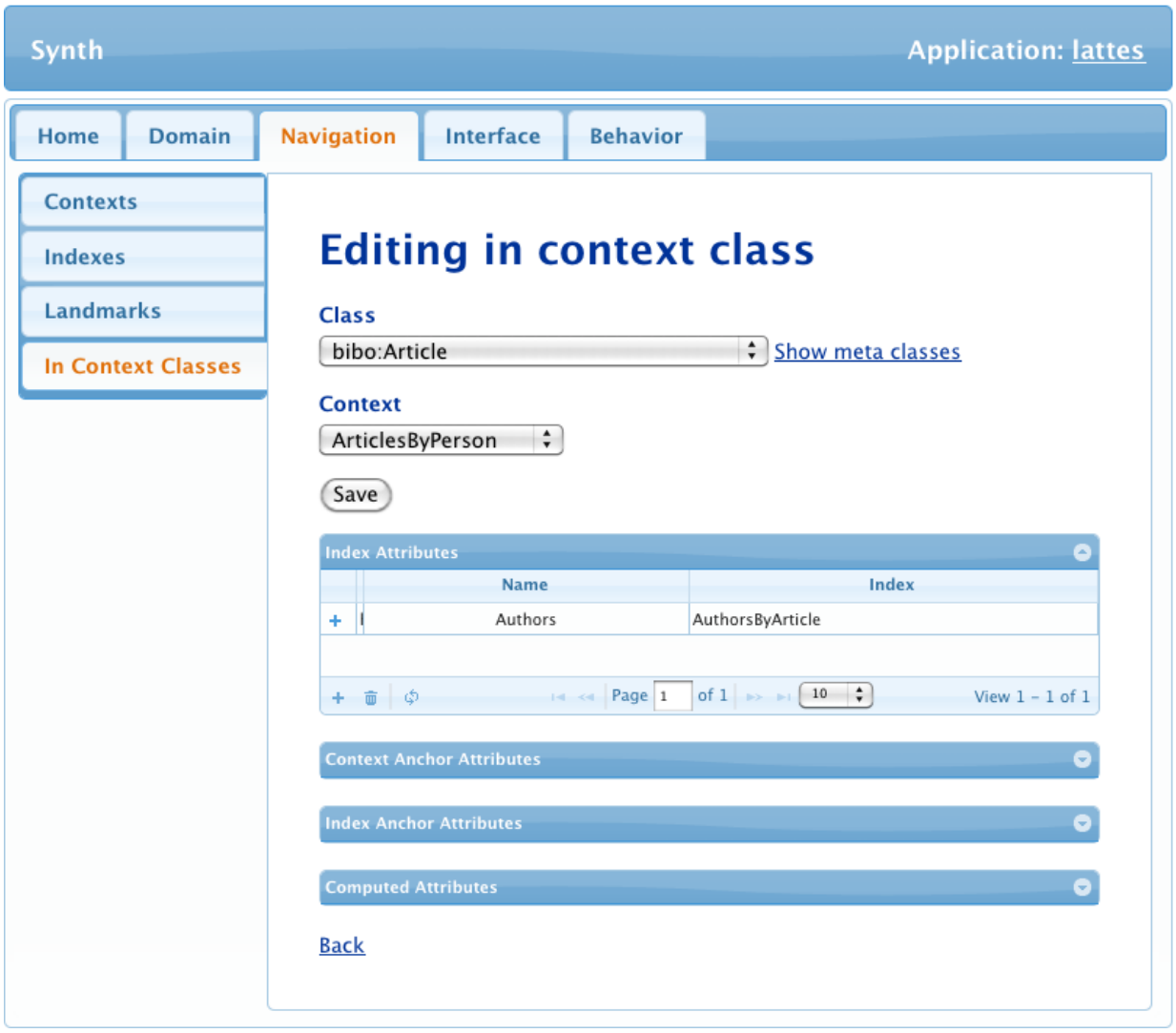


Figura 59 – Tela de edição de uma classe em contexto

Para completar a edição da classe em contexto, deve-se informar os seus atributos navegacionais nas tabelas “Index Attributes”, “Context Anchor Attributes”, “Index Anchor Attributes” e “Computed Attributes”. Essas tabelas têm o mesmo funcionamento das tabelas de mesmo nome na tela de edição de índices, apresentadas na seção 4.4.3.2, por isso não serão descritas em detalhes aqui. Essas tabelas possuem duas diferenças em relação as tabelas de mesmo nome na tela de edição índices: não há o campo *Position* e todos os campos que recebem expressões em Ruby são avaliados no escopo de execução do nó

navegacional e não no escopo de execução da entrada do índice, como é o caso da anterior.

Uma funcionalidade dessas tabelas apresentada na Figura 59, é que quando a tabela não possui nenhuma entrada, esta se apresenta inicialmente minimizada. Para adicionar um novo atributo na tabela, basta clicar no ícone no canto direito e então os outros controles da tabela serão apresentados.

A Figura 60 é a tela resultante do acesso a um nó da classe bibo:Article no contexto ArticlesByPerson.

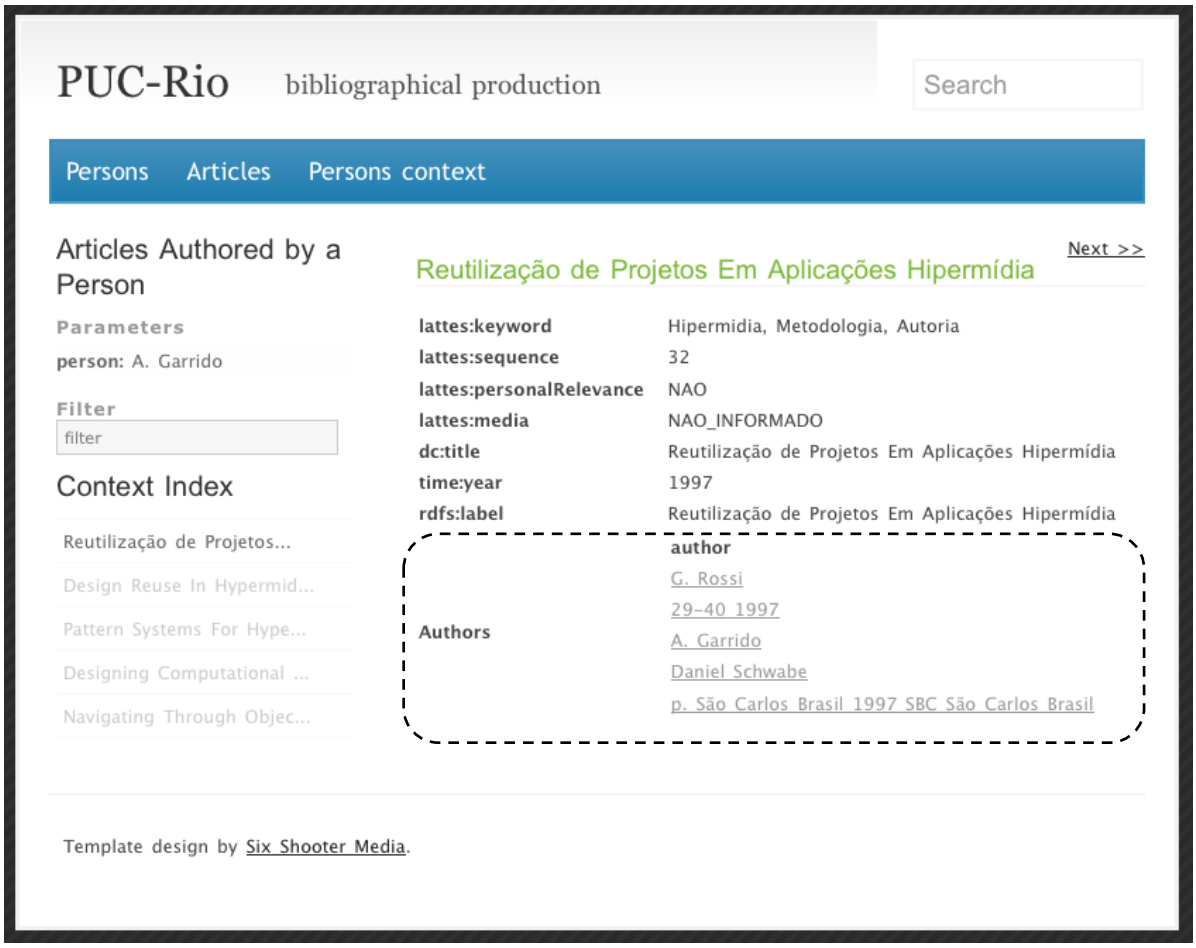


Figura 60 – Tela resultante do acesso a um nó da classe bibo:Article no contexto ArticlesByPerson

O trecho destacado com o contorno tracejado é o atributo navegacional do tipo índice, adicionado ao nó navegacional em tempo de execução em virtude da definição presente na tabela “Index Attributes” apresentada na Figura 59.

4.4.4. Projeto comportamental

A tela de projeto comportamental é acessada através do item de menu "Behavior". Nesta tela, são manipuladas as operações da aplicação.

O gerenciamento das operações da aplicação se inicia pela listagem das operações previamente cadastradas, conforme apresentado na Figura 61. A partir desta tela, é possível adicionar novas operações, editar ou remover operações previamente cadastradas.

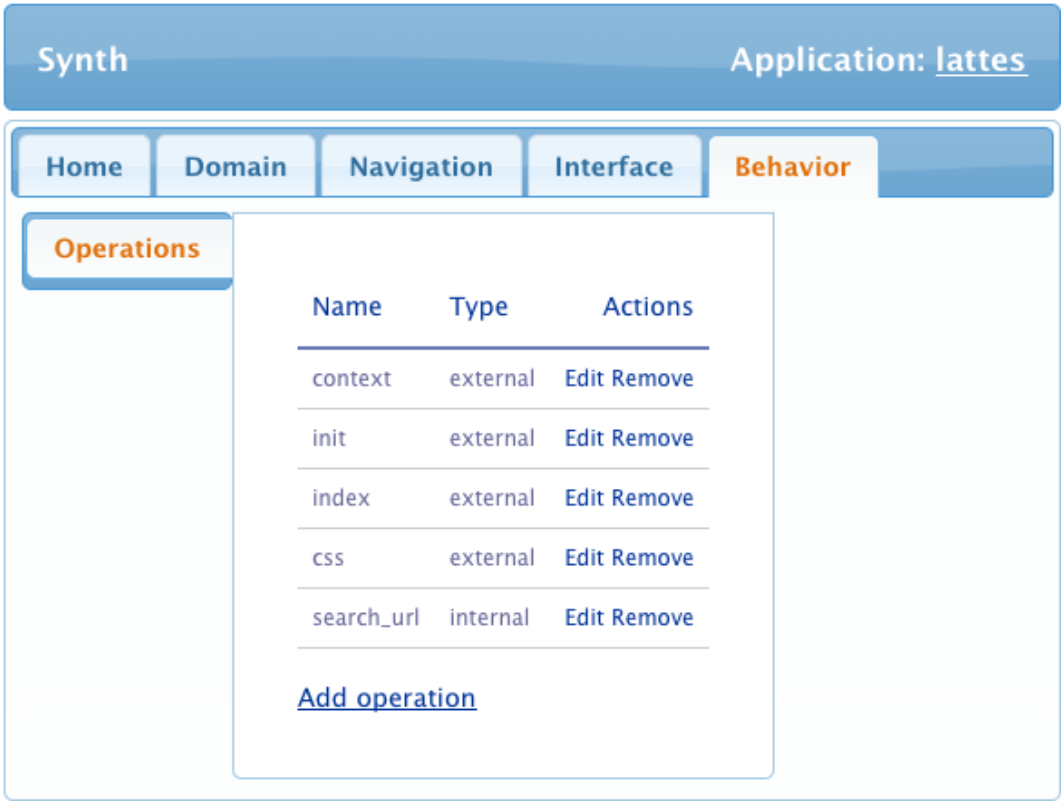


Figura 61 – Tela listagem de operações

Ao clicar na âncora "Add operation", a tela de cadastro de uma nova operação será apresentada. Após preencher os campos e clicar no botão "Create", a nova operação será cadastrada e a tela de edição da operação recém cadastrada será exibida.

Na tela de edição de uma operação, apresentada na Figura 62, deve-se informar o nome da operação, o seu tipo e o seu código. As operações podem ser internas ou externas. O código de uma operação deve ser descrito em linguagem Ruby.

Nas operações internas, o código é avaliado no escopo de execução local da operação, ou seja, este código pode manipular apenas os parâmetros que recebe ou variáveis globais do ambiente. Entre as variáveis globais estão as classes e módulos da biblioteca ActiveRDF, possibilitando que as operações acessem diretamente todos os recursos RDF da base de dados local e, consequentemente, todos os outros modelos da aplicação.

As operações internas são invocadas apenas a partir do próprio ambiente, ou seja, a partir das operações externas, descrições concretas das interfaces ou outras expressões em Ruby usadas nos modelos como, por exemplo, expressões de consultas dos contextos em Ruby ou expressões de valor dos atributos navegacionais. Para invocar uma operação interna deve-se usar a seguinte chamada:

Operations.nome_da_operacao(:parametro1 => valor1, :parametro2 => valor 2).

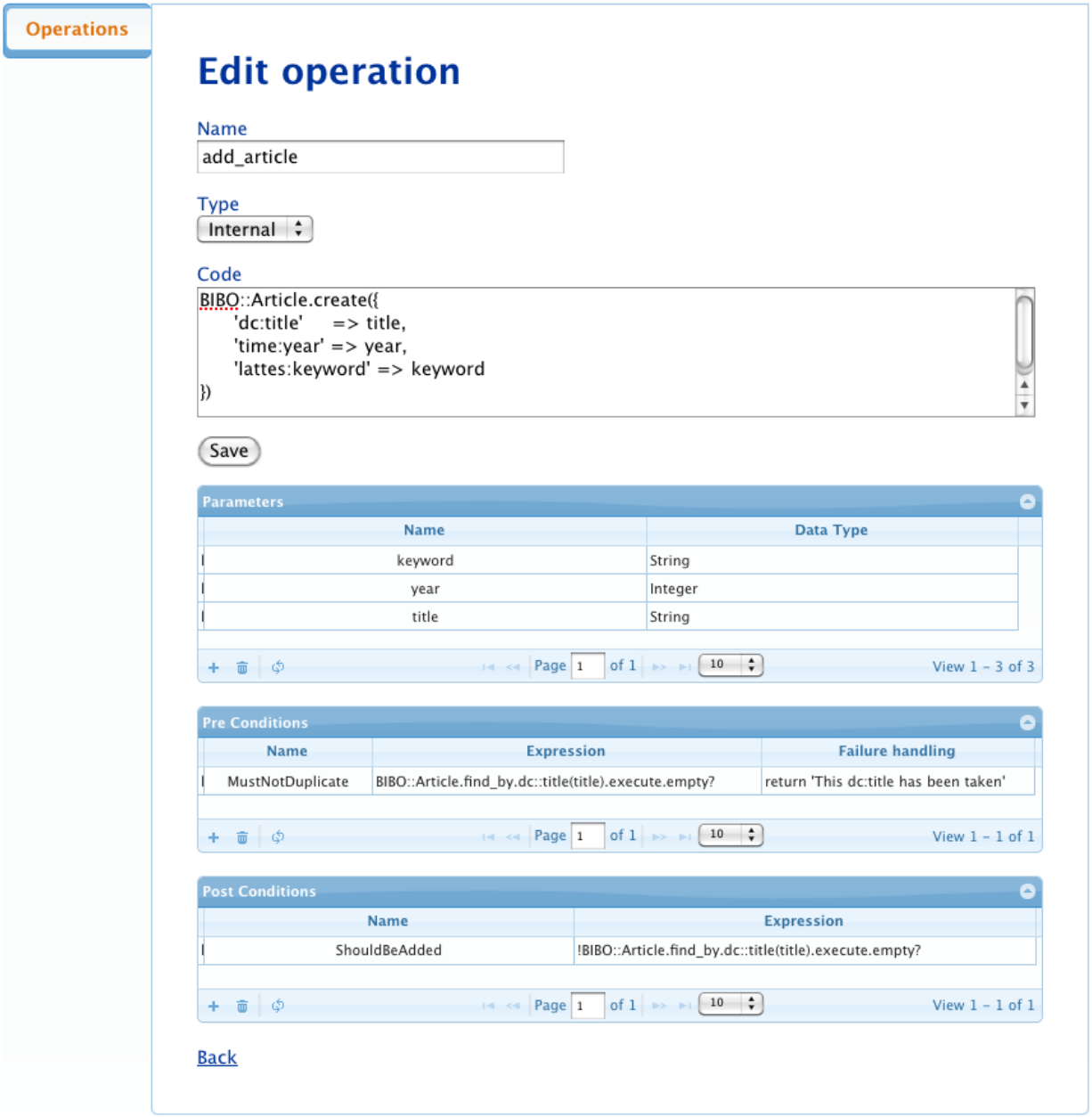


Figura 62 – Tela edição de uma operação

As operações externas têm o seu código avaliado no escopo de execução de um controlador do Ruby on Rails. Sendo assim, as operações externas podem manipular, além dos seus parâmetros e variáveis globais, todos os métodos e variáveis do módulo ActionController do Ruby on Rails, o que na prática, permite que se use os métodos de renderização e redirecionamento, a manipulação de cabeçalhos e parâmetros das requisições HTTP e outras facilidades presentes nos controladores do Ruby on Rails.

A lista a seguir apresenta os métodos e variáveis mais utilizados nos códigos das operações externas:

1. *redirect_to*: executa o redirecionamento da requisição para um controlador ou URL qualquer. Exemplo: redirect_to “http://www.google.com”;
2. *render*: renderiza um template ou string retornando ao cliente uma página. Exemplo: render :text => “hello world”;
3. *params*: variável que carrega os valores dos parâmetros enviados pelo cliente na requisição HTTP. Exemplo: params[:nome_do_parametro];.

As operações externas são invocadas por agentes externos à aplicação por meio do envio de requisições HTTP ao servidor no seguinte endereço:

http://[endereco_do_servidor]/execute/[nome_da_operação]

Para enviar parâmetros para a operação externa, deve-se adicioná-los como parâmetros da requisição HTTP. Para capturar os valores dos parâmetros no código da função, deve-se usar a variável params.

Para completar a edição da operação, deve-se informar os seus parâmetros na tabela “Parameters”, as suas pré-condições na tabela “Pre Conditions” e as suas pós-condições na tabela “Post Conditions”. As tabelas da tela de edição de operações funcionam de forma similar as tabelas vistas nas outras telas de edição do projeto navegacional.

A Figura 63 é um detalhe do modo de edição da tabela de parâmetros de uma opearação. Nesta tabela deve-se informar o nome do parâmetro e o seu tipo.

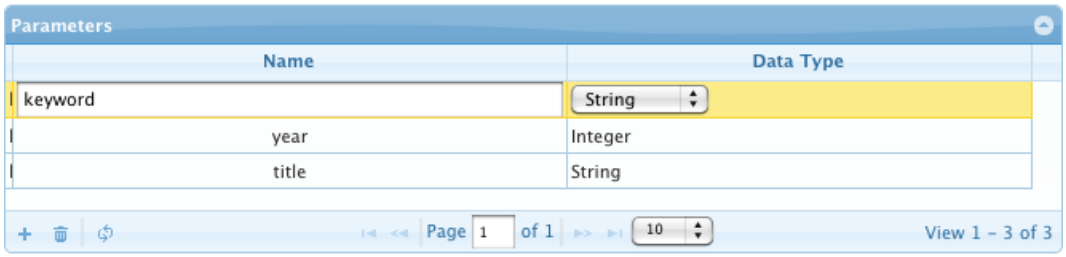


Figura 63 – Detalhe da tabela de parâmetros de uma operação

A Figura 64 é um detalhe do modo de edição da tabela de pré-condições de uma operação.

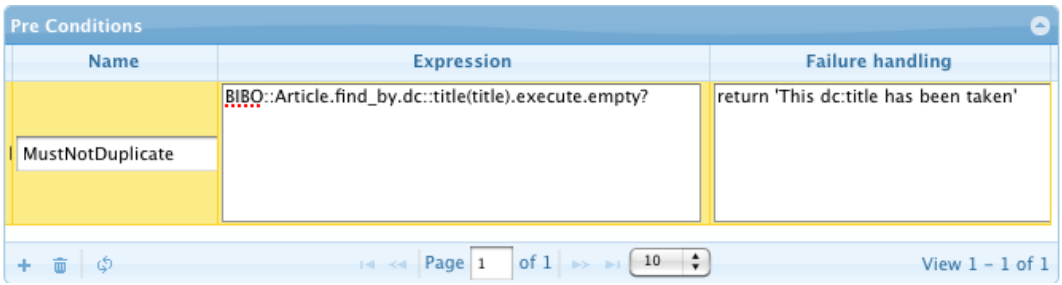


Figura 64 – Detalhe da tabela pré-condições

Nesta tabela deve-se informar o nome da pré-condição, uma expressão a ser avaliada que deve retornar *true* ou *false* e um código de tratamento de falha. A expressão e o código de tratamento de falha devem ser descritos em linguagem Ruby e são avaliados no escopo de execução da operação. A expressão da pré-condição é avaliada antes da execução da operação. Caso a avaliação retorne *true*, o código da operação é executado normalmente. Caso contrário, o código do tratamento de falha é executado no lugar do código da operação.

A Figura 65 é um detalhe do modo de edição da tabela de pós-condições de uma operação.

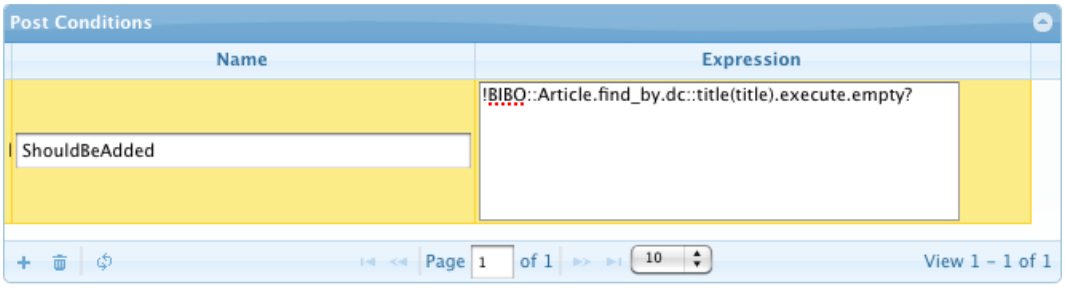


Figura 65 – Detalhe da tabela pós-condições

Nesta tabela deve-se informar o nome da pós-condição e uma expressão em linguagem Ruby avaliada no escopo de execução da operação e que deve retornar *true* ou *false*. Caso a avaliação da expressão retorne *false* a operação retornará, junto com o retorno do seu código, uma mensagem de erro padrão que deve ser tratada pelo código que invocou a operação.

4.4.5.
Projeto de interface

O Synth fornece um ambiente para a modelagem das primitivas do projeto de interfaces da aplicação cuja implementação, salvo pelo novo projeto visual, é praticamente a mesma do ambiente de modelagem proposto por [Luna, 2009, p. 76] para o HyperDE. Por esse motivo, a descrição desta parte do ambiente de autoria não será vista em detalhes nesta dissertação, podendo-se utilizar o texto original como referência.

A tela de projeto de interface é acessada através do item de menu "Interface". Nesta tela, são manipuladas as primitivas do projeto de interface acessadas por meio dos itens de menu de segundo nível "Interfaces", "Style Sheets", "Concrete Widgets" e "Effects". O acesso ao item "Interface" do menu de primeiro nível leva diretamente ao primeiro item de menu de segundo nível "Interfaces" para a manipulação das interfaces da aplicação.

As diferenças do ambiente de autoria do projeto de interfaces do Synth em relação ao ambiente original proposto por [Luna, 2009] são as seguintes:

1. A descrição do *widget* da interface pode ser abstrata ou concreta. A descrição abstrata permanece a mesma proposta originalmente por [Luna, 2009]. A descrição concreta é feita na linguagem ERB (Ruby Templating)[82], que combina código Ruby com HTML para preencher a interface e funciona de forma similar as visões do HyperDE [Nunes, 2005];
2. Os rótulos dos itens do menu de segundo nível eram "Abstract Interfaces" e "Concrete Interfaces" e mudaram para, respectivamente, "Interfaces" e "Style Sheets". As telas de destino permaneceram as mesmas.

Na descrição concreta do *widget* de uma interface é possível utilizar variáveis que tornam possível a renderização dos nós navegacionais e seus atributos. Essas variáveis são listadas a seguir:

- @current_node: o nó navegacional corrente;

[82] http://www.ruby-doc.org/stdlib/libdoc/erb/rdoc

- @context: instância de contexto navegacional corrente;
- @index: instância de índice navegacional corrente.

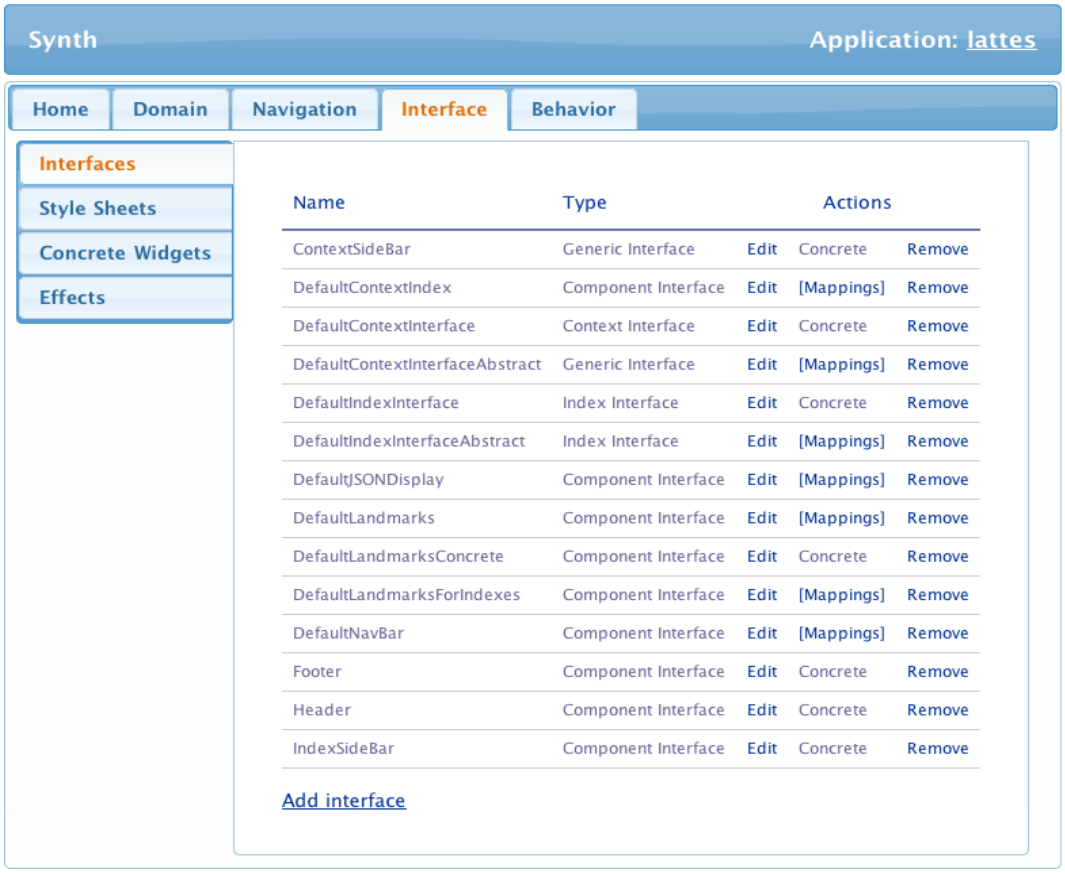
Além das variáveis descritas acima, a descrição concreta da interface tem acesso a qualquer variável ou constante global do ambiente e operações internas. Como as classes da biblioteca ActiveRDF são acessadas a partir de suas constantes globais, é possível manipular qualquer objeto dos modelos manipulados pelo Synth. Por exemplo, a chamada SHDM::Landmark.all retorna um array com todos os *landmarks* da aplicação e pode ser usado para renderizar o menu de *landmarks* na interface.

O Quadro 26 ilustra a descrição concreta de uma interface de contexto que manipula a maioria dos elementos necessários para compor uma interface.

```
<%= include 'Header' %>
<%= include 'ContextSideBar' %>

<h2><%= @current_node.rdfs::label.first</h2>

<table>
<% for property in @current_node.direct_properties do %>
  <tr>
  <th><%= property.label.first %></th><td><%= h property.to_s %></td>
  </tr>
<% end %>
</table>
<table>
  <% for name, attribute in @current_node.attributes_hash do %>
  <tr><th><%= name %></th>
     <td>
     <% if attribute.respond_to?(:target_url) %>
       <%= link_to attribute.label, attribute.target_url %></td>
     <% else %>
       <%= h attribute.label %>
     <% end %>
  </td></tr>
  <% end %>
</table>

<%= include 'Footer' %>
```

Quadro 26 – Código de descrição concreta de uma interface

As figuras a seguir apresentam as telas de modelagem da interface no Synth.



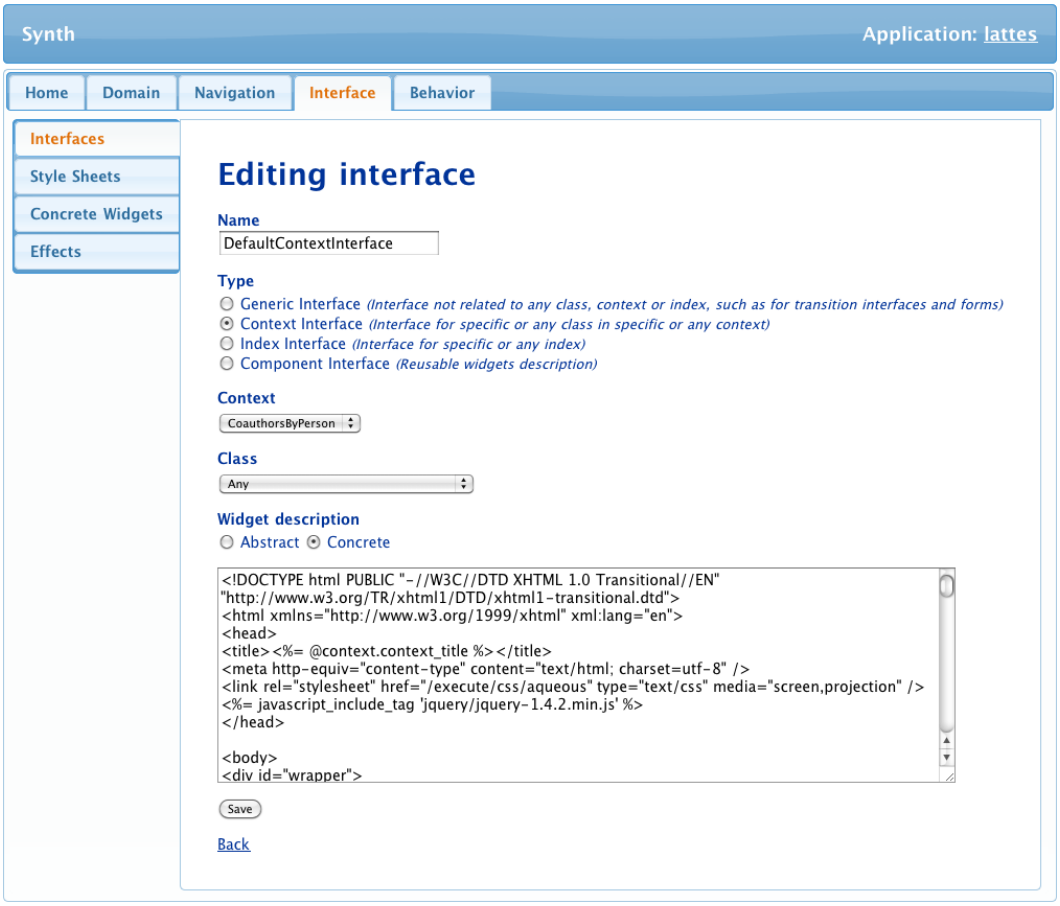


Figura 66 – Telas de listagem e edição de interfaces

Figura 67 – Telas de listagem e edição de folhas de estilo

Figura 68 – Telas de listagem e edição de *widgets* concretos

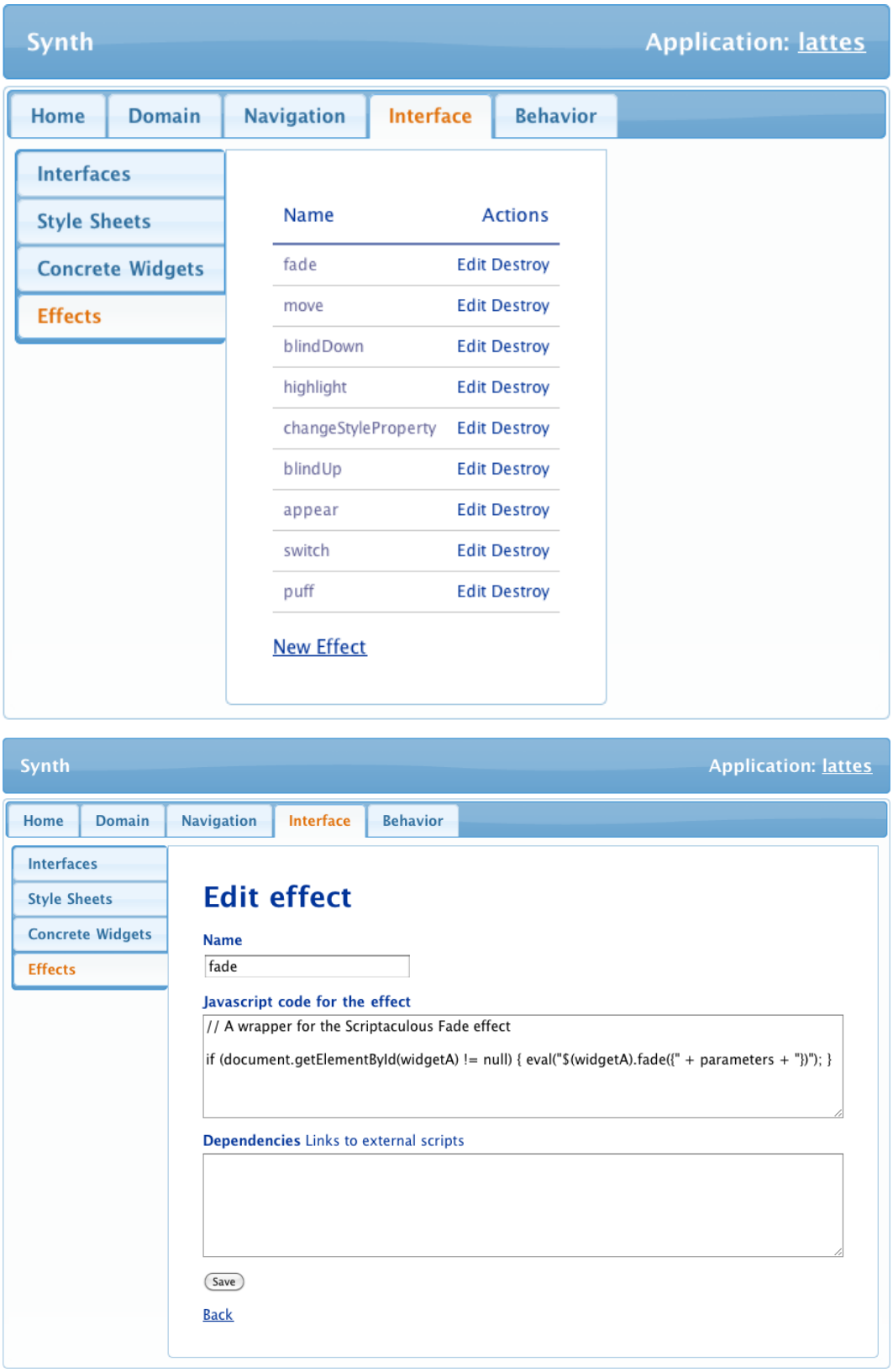


Figura 69 – Telas de listagem e edição de efeitos

4.5. Execução da aplicação

A execução de uma aplicação modelada com o Synth começa pela execução da operação externa "init", ou seja, o envio de uma requisição HTTP ao endereço http://[endereco_do_servidor]/execute/init.

A tarefa operação "init" é identificar o primeiro *landmark* da aplicação e redirecionar a página para a âncora desse *landmark*. Este é o ponto inicial de navegação na aplicação. O Quadro 27 apresenta o código da operação "init".

```
landmark = SHDM::Landmark.alpha(SHDM::landmark_position).first

unless landmark.nil?
   redirect_to landmark.target_url
else
   render :text => "There is no initial navigation"
end
```

Quadro 27 – Código da operação externa "init"

Por se tratar de uma operação externa, primitiva do modelo comportamental, a semântica da inicialização da aplicação pode ser modificada pela alteração do código da operação "init". O mesmo é verdade para as operações externas "context" e "index" que contêm a semântica de navegação nos contextos e índices. A Figura 70 apresenta a tela inicial de uma aplicação, resultado da execução da operação externa "init".

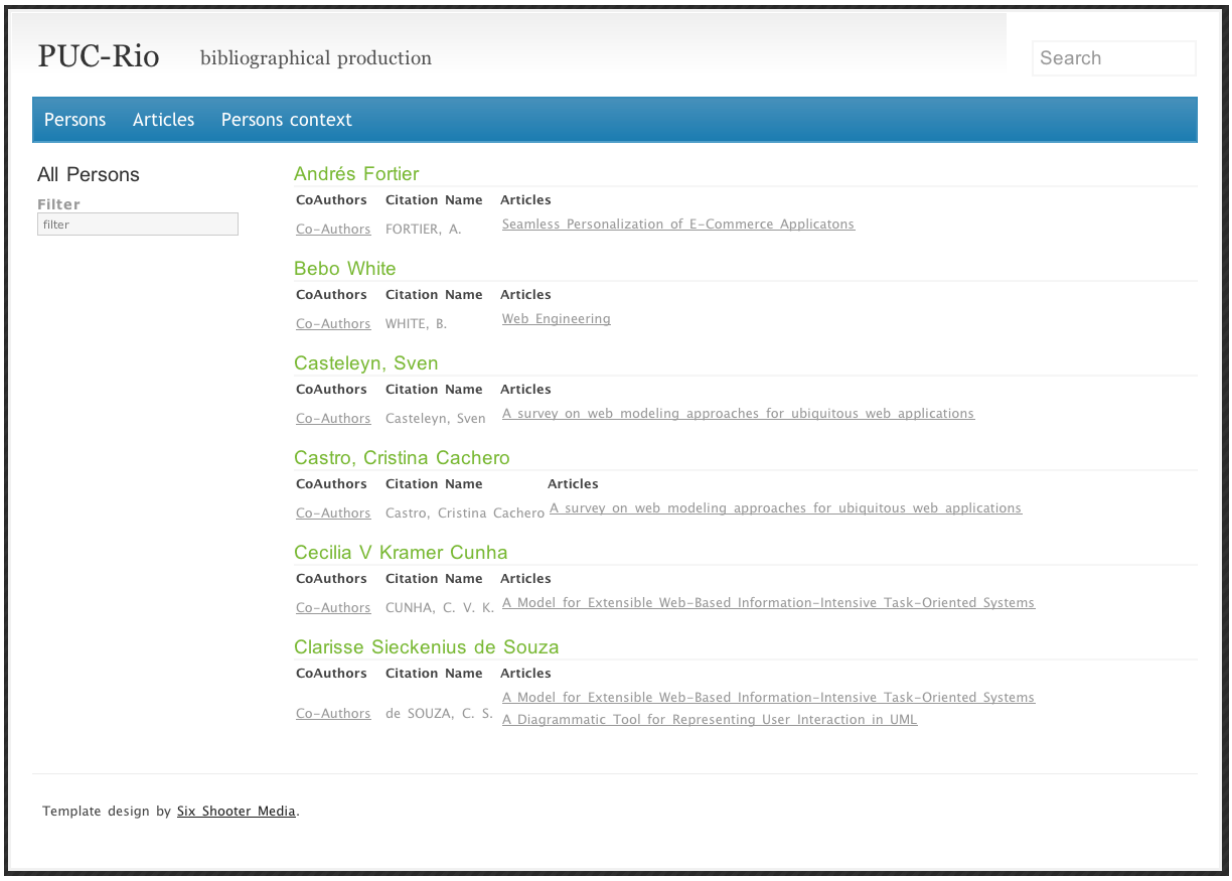


Figura 70 – Tela inicial de uma aplicação construída com o Synth

4.6. Ferramenta de importação de dados RDF

O Synth oferece uma ferramenta de importação de dados RDF para facilitar a inclusão de dados na sua base de dados local. Esta ferramenta é um comando executado a partir do terminal ou *prompt* de comandos do sistema operacional e aceita como entrada qualquer arquivo RDF.

O Quadro 28 apresenta um exemplo de uso do comando de importação.

```
jruby -S script/import origem.rdf model=nome_da_aplicacao format=rdfxml
```

Quadro 28 – Exemplo de comando de importação

A lista a seguir descreve cada parâmetro do comando:

- "origem.rdf" é o nome do arquivo RDF que será importado para a base de dados. Deve ser informado o caminho completo para o arquivo;
- nome_da_aplicação é o nome da aplicação que receberá os dados do arquivo;

- “format” recebe o nome da notação utilizada e aceita os seguintes valores: rdfxml, n3, ntriples e turtle.