UNIVERSIDADE DE CAXIAS DO SUL
CENTRO DE COMPUTAÇÃO E TECNOLOGIA DA INFORMAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

JOÃO TOSS MOLON

Armazenando registros de colaboração utilizando *triple store*

Elisa Boff
Orientadora

João Luis Tavares da Silva
Coorientador

Caxias do Sul, Dezembro de 2013

"A felicidade só é real quando compartilhada."
Christopher McCandless

AGRADECIMENTOS

Aos meus pais, Nestor Molon e Maristela Toss Molon, pela dedicação e sacrifício empregados na minha criação, pelo apoio moral e financeiro que me deram durante esses 5 anos de graduação, por tudo o que me ensinaram nesses meus 22 anos de vida e pela compreensão com as minhas escolhas de vida que distanciam um pouco daquilo que poderia ser o esperado.

Aos meus irmãos, Marcos e Mateus, pela amizade, parceria e companheirismo, além do conhecimento que me passaram sobre a vida.

À minha namorada, Mariana Libardi, pelo amor, carinho, compreensão com a minha necessidade de me dedicar a este trabalho, bem como a parceria para os momentos de descontração necessários durante este período.

Ao colega e amigo Luciano Camargo Cruz, pelas inúmeras cadeiras cursadas juntos, pelos trabalhos feitos em grupo, por me apresentar ao mundo do Python e pela indicação ao emprego que tenho hoje, sem o qual este trabalho jamais seria realizado.

Aos colegas e amigos, Matheus Pereira e Felipi Medeiros Macedo, pela amizade criada durante esses longos anos de graduação e, mais recentemente, de atividade profissional.

À Drª Elisa Boff, minha orientadora, por guiar o caminho durante a execução deste trabalho.

Ao chefe, professor, amigo e co-orientador deste trabalho, Dr. João Luis Tavares da Silva, pela imensa ajuda e orientação que proveu durante todo esse tempo, sendo sempre um pai para todos nós.

Ao orientador oculto deste trabalho, Dr. Tiago Thompsen Primo, que foi o "pai" das tecnologias semânticas para este trabalho.

Ao chefe Dr. Alexandre Moretto Ribeiro, pela oportunidade profissional que proporciona a todos os colegas na HaDi.Com e Instituto Communitas.

Aos amigos e irmãos do CDC Caxias, pelos tantos anos de amizade, companheirismo e eventos agregadores.

A todos aqueles que de alguma forma colaboraram para este trabalho e, por algum motivo que eu não saberia dizer qual é, foram esquecidos nesses

agradecimentos.

A todos vocês, minha sincera gratidão.

João Toss Molon

SUMÁRIO

LISTA DE ACRÔNIMOS

API	*Application Programming Interface*
HTTP	*Hypertext Transfer Protocol*
XML	*Extensible Markup Language*
RDF	*Resource Description Framework*
RDFS	*RDF Schema*
OWL	*Web Ontology Language*
SQL	*Structured Query Language*
HTML	*Hypertext Markup Language*
URI	*Uniform Resource Identifier*
URL	*Uniform Resource Locator*
ORM	*Object RDF Mapper*

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE TRECHOS DE CÓDIGO

RESUMO

Este trabalho tem como objetivo o desenvolvimento de um protótipo para o armazenamento de Registros de Colaboração em bases de dados *triple store*. No contexto de Comunidades de Prática, Registros de Colaboração podem ser vistos como conjuntos de propriedades que detalham as interações que podem ocorrer entre os membros de uma comunidade, comunidades distintas e os seus conteúdos. Tais registros são estruturados utilizando-se o modelo RDF. O desenvolvimento desse protótipo é efetuado com base no *Framework CoP*. O armazenamento dos Registros de Colaboração foi proposto através de uma *triple store* de maneira a proporcionar consultas mais ricas dentro do contexto de Comunidades de Prática.

Palavras-chave: Triple store, Comunidades de Prática, Web Semântica, Registros de Colaboração, Python, Zope, Plone.

English Abstract

ABSTRACT

This paper's goal is the development of a prototype for the storage of Collaboration Registers in triple store data bases. On Communities of Practice context, Collaboration Registers can be seen as properties sets which detail the interactions that happen between community's members, distinct communities a their contents. Such registers are structured using the RDF data model. This prototype development is based on the Framework CoP. The storage of Collaboration Registers was proposed through triple store in a way to provide richer querys in a Communities of Practice context.

Keywords: triple store, Communities of Practice, Semantic Web, Collaboration Registers, Python, Zope, Plone.

1 INTRODUÇÃO

Através do uso de Tecnologias de Informação e Comunicação (TIC) na educação, a aprendizagem vem ganhando novas dimensões. Desde a exploração do computador para reprodução de modelos conhecidos até o uso de sistemas inteligentes para fins de ensino (CARBONELL, 1970). Com o advento da Internet, de um lado, foram desenvolvidos no meio acadêmico os Ambientes Virtuais de Aprendizagem (AVA), enquanto no meio corporativo foram criados ambientes virtuais para o suporte a aprendizagem organizacional, as Comunidades de Prática (CoPs - *Communities of Practice*).

Com o surgimento de tecnologias para suporte à redes sociais, a aprendizagem social tem ganhado destaque nos desafios da educação. A aprendizagem social envolve grupos dinâmicos de pessoas que compartilham objetivos comuns e práticas sobre o conteúdo em questão. Este trabalho parte do pressuposto que aprendizagem colaborativa (BANDURA, 1977; HART, 2011) constitui a arquitetura base das CoP segundo (FIORIO; SILVA; RIBEIRO, 2011). Neste sentido, a aprendizagem social é construída a partir de conversas e participação em uma CoP, e tem mais relação com a forma como se aprende do que com o conteúdo aprendido (HEALY, 2009). Neste contexto, os "aprendizes sociais" assumem papéis dinâmicos, tanto como consumidores quanto de produtores de conhecimento, cujas contribuições estão relacionadas a suas habilidades, interesses e conhecimento.

Conforme WENGER (1998), mais que comunidades de aprendizes, a comunidade de prática é uma "comunidade que aprende", pois reúne pessoas que têm compromisso de agregar as melhores práticas. Uma comunidade de prática não é somente um agregado de pessoas definidas por algumas características, mas sim de pessoas que aprendem, constroem e fazem a gestão do conhecimento.

Em (FIORIO; SILVA; RIBEIRO, 2011) e (RIBEIRO et al., 2011) propõe-se um *Framework* de Comunidades de Prática. Esta ferramenta tem o intuito de proporcionar uma plataforma de construção de comunidades virtuais, visando o desenvolvimento de práticas e habilidades relacionadas a um domínio específico.

O *Framework CoP* é constituído por vários componentes que oferecem funcionalidades de cadastro, autenticação, criação e gestão de conteúdo, voltados para o contexto de Comunidades de Prática. Estes componentes interagem a partir de vários núcleos de conhecimento: Domínio de Interesses, Perfis e Registros de Colaboração. Os Domínios de Interesse tratam da construção coletiva de conhecimento, auxiliada por um grupo de editores/mediadores e ontologias de domínio predefinidas. As ontologias também são usadas na comunidade, construídas pelos próprios participantes (utilizando mediadores) e/ou geradas semi-automaticamente. Os Domínios de Interesse também integram um repositório de ontologias e abordam questões mais amplas, unificando conceitos em torno dos domínios da comunidade.

O principal foco deste trabalho encontra-se no núcleo de conhecimentos dos Registros de Colaboração, em que é preciso descrever em termos ontológicos as possíveis relações entre as características e interesses dos usuários através das suas interações em um ambiente dinâmico e distribuído. O Registro de Colaboração prevê conjuntos de propriedades que descrevem as formas de relacionamentos que podem ocorrer entre indivíduos, comunidades e seus conteúdos, através de um modelo ontológico (RIBEIRO et al., 2011), (PRIMO et al., 2012). Tais trabalhos constituem a base para definição dos requisitos para uma proposta de relacionamentos de usuários.

A questão de pesquisa norteadora deste trabalho é "Como implementar Registros de Colaboração de forma a aproveitar a característica do modelo ontológico?"

Uma hipótese a ser testada é representar os Registros de Colaboração usando *triple store* e aproveitar a representação em linguagem ontológica com consequente recuperação através de ferramentas de busca semântica. A proposta de implementação usando *triple store* ainda torna o *Framework CoP* aderente ao modelo de Web Semântica, padronizando a representação através do uso de RDF e OWL (BERNERS-LEE; HALL; SHADBOLT, 2006).

1.1 Objetivo e organização do texto

Este trabalho tem como principal objetivo o desenvolvimento de uma solução para armazenamento de dados de colaboração em bases de dados de triplas, aderente ao modelo de web semântica. O Capítulo 2 trata sobre Web Semântica, seus conceitos, os padrões e tecnologias que compõe suas bases estruturais. Em seguida, o Capítulo 3 aborda as questões relativas a *triple store*, como conceitos básicos, a linguagem de consulta SPARQL, *frameworks* existentes e trabalhos relacionados. O capítulo seguinte trata do *Framework* de Comunidades de Prática e as tecnologias envolvidas em seu desenvolvimento. No Capítulo 5 é apresentada a

solução desenvolvida para o problema já referido, assim como o processo de validação do mesmo. Por fim, são apresentadas as considerações finais do trabalho.

2 WEB SEMÂNTICA

A Web tradicional é composta basicamente por documentos que ligam-se uns aos outros por meio de links. A informação encontra-se em um formato muito mais compreensível por pessoas do que por computadores. É nesse cenário que surge a ideia da Web Semântica.

Segundo HOUAISS (2001), semântica é a "teoria abstrata da significação ou da relação entre os signos e seus referentes (em oposição à sintaxe e à pragmática), e constituindo com estas uma semiótica". Em outras palavras, semântica diz respeito ao significado de algo e às relações que o mesmo estabelece.

O termo Web Semântica refere-se a rede de informações que dá significado aos conteúdos na Internet. Para BERNERS-LEE; HENDLER; LASSILA (2001), a Web Semântica pode ser vista como "uma extensão da Web atual, onde a informação tem um significado bem definido, permitindo que computadores e pessoas trabalhem em cooperação". Ou seja, além do conteúdo em si, que pode ser interpretado pelo ser humano, existem mais informações que permitem que os conteúdos sejam reconhecidos e manipulados de maneira mais significativa pelas máquinas. Tais informações permitem que os computadores executem tarefas mais sofisticadas e retornem resultados mais precisos para o usuário (BERNERS-LEE; HENDLER; LASSILA, 2001; BERNERS-LEE; HALL; SHADBOLT, 2006).

Grande parte da utilização da Internet ocorre através de ferramentas de busca. Por mais que as mesmas tenham evoluído muito ao longo do tempo, a principal forma de buscar resultados continua sendo a pesquisa de termos e expressões dentro do texto do documento. Esse tipo de busca tem vários problemas. A pouca precisão, quando muitos resultados são retornados, mas poucos são relevantes. Outra questão importante é que a busca por uma palavra chave é sensível à terminologia empregada nas páginas. Por fim, quando o resultado esperado está distribuído entre vários documentos, mais de uma busca é necessária (ANTONIOU; HARMELEN, 2008).

Essa dificuldade em se localizar a informação na Web deve-se, em grande parte, a natureza pouco estruturada dos dados. A linguagem padrão utilizada no desenvolvimento das páginas, o HTML (*Hypertext Markup Language*), carece de

ferramentas para estruturação da informação. Esse fato tem como consequência a perda de conhecimento a respeito da estrutura e semântica dos dados (BIZER; HEATH; BERNERS-LEE, 2009).

O grande objetivo da Web semântica é trazer significado ao conteúdo da Internet. O primeiro passo para que isso ocorra é a estruturação dos dados. A utilização de linguagens e padrões apropriados para representação de um conteúdo permite que a sua estrutura, suas propriedades e suas relações com outros conteúdos tornem-se explícitas, facilitando e expandindo a gama de operações que os computadores podem processar sobre esses dados (BIZER; HEATH; BERNERS-LEE, 2009).

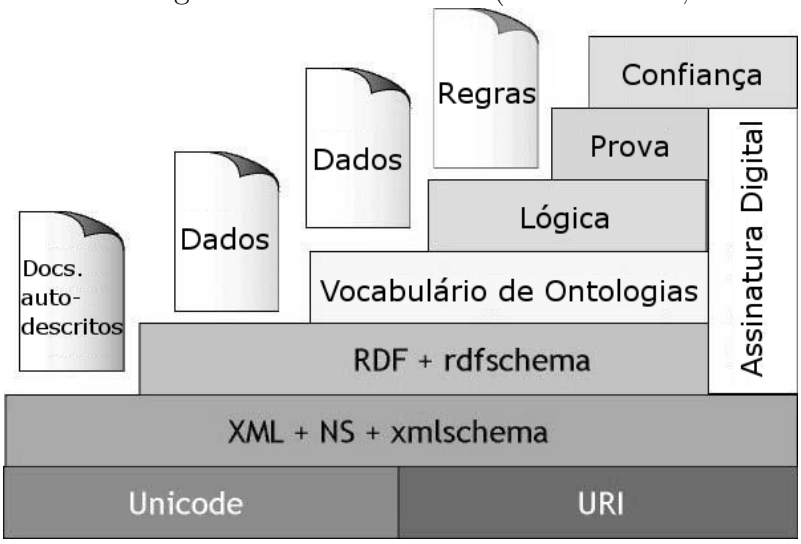
BERNERS-LEE (2006) propõe 4 regras para a disponibilização de dados na Web Semântica:

> "Utilizar URIs como nomes para as coisas; Utilizar URIs HTTP, para que as pessoas possam pesquisar esses nomes; Quando alguém pesquisar uma URI, prover informações úteis, utilizando padrões (RDF, SPARQL); Incluir links para outras URIs, para que eles possam descobrir mais coisas."

2.1 Bases Estruturais da Web Semântica

Os conceitos de Web Semântica são fundamentados sobre uma série de padrões e tecnologias com a finalidade de identificar, estruturar, relacionar e dar significado ao conteúdo da Web. A Figura 2.1 ilustra a arquitetura em camadas da Web Semântica conforme proposto por Berners-Lee (ANTONIOU; HARMELEN, 2008).

Figura 2.1: Pilha tecnológica da Web Semântica (ANTONIOU; HARMELEN, 2008)



Na base de tudo, estão uma codificação em comum (*Unicode*) e a identificação de todos os recursos com identificadores únicos URI (*Uniform Resource Identifier*). Na camada imediatamente superior está o XML (*Extensible Markup Language*),

que permite a construção de documentos estruturados. Acima, o RDF (*Resource Description Framework*) e o RDFS (*RDF Schema*) adicionam uma forma básica de semântica ao conteúdo. Na camada seguinte, estão as linguagens de ontologias, que permitem expressar relações mais ricas entre recursos na Web. A camada de Lógica permite que sejam feitas inferências a partir das ontologias descritas na camada inferior. Acima, a camada de Prova faz a validação de tais inferências. Por fim, a camada de Confiabilidade encarrega-se de garantir que as operações e os dados da Web sejam confiáveis, através de assinaturas digitais e outros recursos. Nesse trabalho, serão abordadas a camada de Ontologias e inferiores.

2.1.1 *Uniform Resource Identifier*

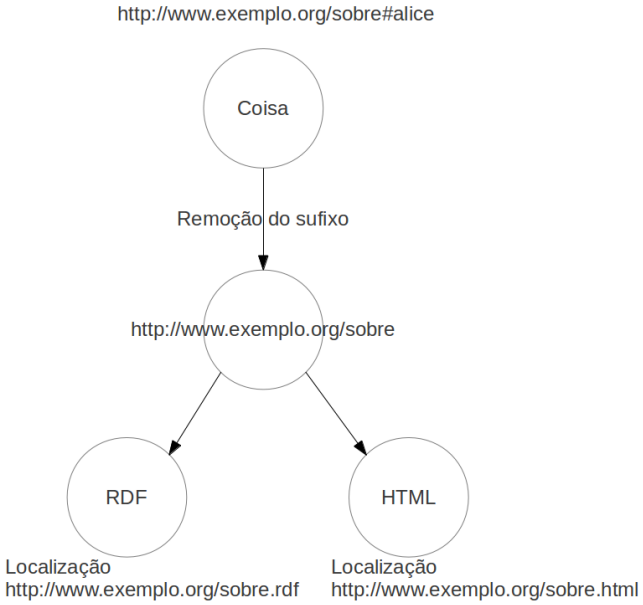
Em primeiro lugar, todo tipo de recurso utilizado em uma rede semântica deve possuir uma URI, a qual "é uma sequência compacta de caracteres que identifica um recurso abstrato ou físico" (BERNERS-LEE; FIELDING; MASINTER, 2005). Ao atribuir-se uma URI a um recurso, o mesmo torna-se globalmente acessível e referenciável (BERNERS-LEE; HALL; SHADBOLT, 2006).

Uma URI é um identificador único. O tipo mais comum de URI são as URLs (*Uniform Resource Locators*) (BERNERS-LEE; HENDLER; LASSILA, 2001), utilizadas para endereçar documentos e outros tipos de dados na Web. O conceito de URI, porém, é mais amplo, uma vez que uma URI identifica tanto recursos dentro da Web quanto entidades do mundo real (BIZER; HEATH; BERNERS-LEE, 2009). Sendo assim, uma URI pode ser, por exemplo, o ISBN de um livro, um número de telefone, entre outros (ANTONIOU; HARMELEN, 2008).

Os dois principais padrões para geração de URIs são o *Hash URI* e o *303 URI*. Ambos os métodos permitem a diferenciação entre entidades do mundo real e documentos da Web (BIZER; HEATH; BERNERS-LEE, 2009).

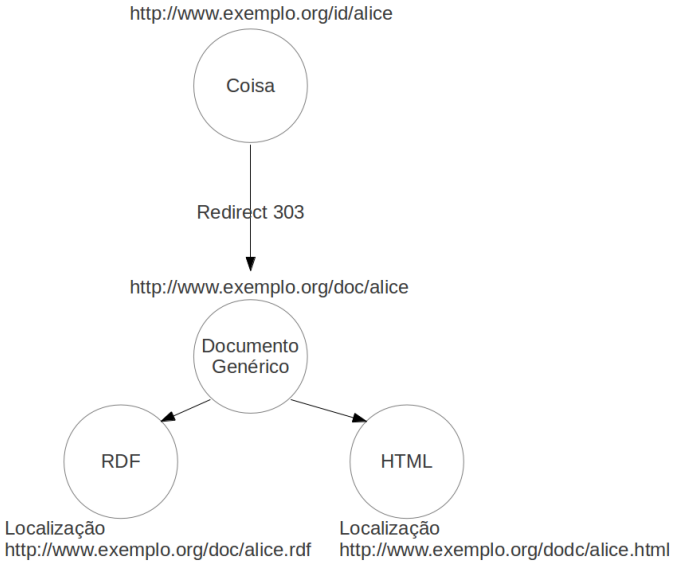
O método *Hash URI* adiciona um sufixo especial, que é separado do resto da URI pelo símbolo *hash* (#). Esse símbolo precisa ser tratado pelo servidor, o que indica que a URI pode não estar referenciando um simples documento web (SAUERMANN; CYGANIAK, 2008). A Figura 2.2 demonstra essa situação.

Figura 2.2: Resolução de uma URI que segue o padrão *Hash URI* (Adaptado de (SAUERMANN; CYGANIAK, 2008))



O método *303 URI* modifica o código retornado pelo servidor após a resolução da URI. Ao invés de retornar o código 200, padrão para documentos da web, o servidor retorna o código 303, o que indica que a URI não se refere a um documento da Web padrão (SAUERMANN; CYGANIAK, 2008). A Figura 2.3 exemplifica o funcionamento de uma URI *303 URI*.

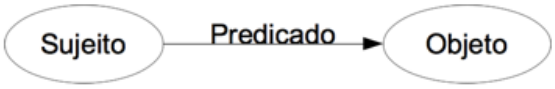
Figura 2.3: Resolução de uma URI que segue o padrão *303 URI* (Adaptado de (SAUERMANN; CYGANIAK, 2008))



2.1.2 RDF e *RDF Schema*

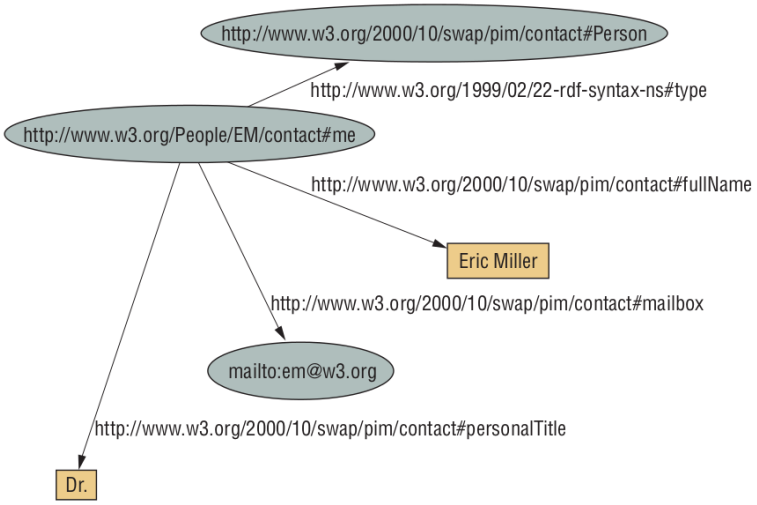
Como já referido anteriormente, a Web Semântica precisa que os dados tenham uma estrutura e um significado explícito. Para isso, existem tecnologias específicas que desempenham esse papel. O XML permite a criação de *tags* diversas, o que possibilita a estruturação da informação presente no documento. Porém, isso não é o bastante do ponto de vista semântico (BERNERS-LEE; HENDLER; LASSILA, 2001). Para a adição de significado aos conteúdos da Web é utilizado o padrão RDF. Essa tecnologia permite a representação de dados no modelo de um grafo. A Figura 2.4 ilustra este modelo, onde os registros são estruturados em triplas, cada uma contendo sujeito, predicado e objeto.

Figura 2.4: Exemplo de Grafo semântico



Sujeito e objeto são URIs que identificam recursos na Web. O predicado indica o tipo de relação presente entre o sujeito e o objeto (BIZER; HEATH; BERNERS-LEE, 2009). Por exemplo, uma tripla RDF pode representar a relação existente entre uma obra e o seu autor, atestando que *"Shakespeare"* (sujeito) *"é autor"* (predicado) de *"Romeu e Julieta"* (objeto). Aplicando-se essa visão ao modelo de um grafo, sujeito e objeto são os nodos, e o predicado é a aresta dirigida que liga o primeiro ao segundo (KLYNE; CARROLL, 2004). A Figura 2.5 representa as informações de contato de uma pessoa, chamada Eric Miller.

Figura 2.5: Exemplo de Grafo RDF representando uma Pessoa (MANOLA; MILLER, 2004)



Uma Pessoa pode ser identificada por uma URI (http://www.w3.org/People/EM

/contact#me), que possui um nome (*fullFame*) Eric Miller, um e-mail (*mailbox*) em@w3.org e um título (*personalTitle*) de Dr.

Sendo um modelo conceitual de dados, o RDF precisa ser representado de alguma forma que possa ser processada pelos computadores (ANTONIOU; HARMELEN, 2008). A representação mais usada (porém não a única) baseia-se em XML, conforme mostra o Trecho de Código 2.1.

Trecho de Código 2.1: Representação de um grafo RDF em XML. (MANOLA; MILLER, 2004)

```
 1 <?xml version="1.0"?>
 2 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 3              xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#">
 4
 5   <contact:Person rdf:about="http://www.w3.org/People/EM/contact#me">
 6     <contact:fullName>Eric Miller</contact:fullName>
 7     <contact:mailbox rdf:resource="mailto:em@w3.org"/>
 8     <contact:personalTitle>Dr.</contact:personalTitle>
 9   </contact:Person>
10
11 </rdf:RDF>
```

O RDF possui um escopo global. Ou seja, ele apenas permite a estruturação dos dados de forma que, a partir de algum vocabulário de um domínio específico, seja possível fazer inferências e relações semânticas entre os conteúdos. Para a definição desses vocabulários, existe o *RDF Schema* (RDFS) (ANTONIOU; HARMELEN, 2008).

O RDFS permite a criação de vocabulários para definição da nomenclatura específica para um conjunto de dados de um determinado domínio. É essa terminologia que será utilizada na construção dos arquivos RDF com o conteúdo do domínio (ANTONIOU; HARMELEN, 2008). Pode-se dizer que "o RDFS é uma extensão semântica do RDF" (BRICKLEY; GUHA, 2004).

Para representar um determinado domínio, antes de mais nada é necessário saber o que se quer falar sobre tal domínio (ANTONIOU; HARMELEN, 2008). Sendo assim, o RDFS permite a definição de Classes, que são os tipos de entidades que serão descritos, e Propriedades, que podem ser tanto atributos de uma entidade como tipos de relacionamento entre entidades (BRICKLEY; GUHA, 2004) (MANOLA; MILLER, 2004).

A representação de estrutura de um domínio com RDFS assemelha-se com o paradigma de Programação Orientada a Objetos em alguns pontos, e difere em outros. A semelhança existe pelo fato de as Classes poderem ser estruturadas de forma hierárquica (ANTONIOU; HARMELEN, 2008) (BRICKLEY; GUHA, 2004). Por exemplo, pode-se dizer que a classe Aluno é subclasse da classe Pessoa. Isso significa que todo recurso da classe Aluno terá, automaticamente, as mesmas

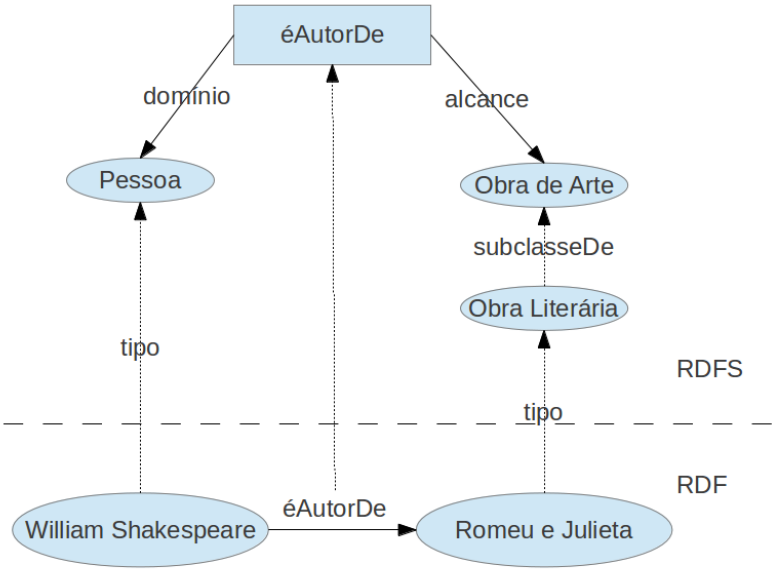
propriedades da classe Pessoa.

Porém, é justamente na forma de se definir as classes e propriedades que está a maior diferença entre o RDFS e as linguagens orientadas a objetos. Enquanto em orientação a objetos uma classe é definida a partir dos atributos que seus objetos vão possuir, em RDFS as propriedades são definidas levando-se em conta a quais classes de recursos elas podem ser aplicadas (BRICKLEY; GUHA, 2004).

Muitas vezes as classes são utilizadas para impor restrições ao domínio e alcance das propriedades (ANTONIOU; HARMELEN, 2008). Por exemplo, não é interessante que se possa declarar que "William Shakespeare é autor de Julio Verne". Espera-se que um escritor seja autor de uma obra, e não de outro autor. Essa é uma restrição de alcance da propriedade "é autor". Já na sentença "Brasil é autor de Romeu e Julieta", é necessário restringir que somente uma pessoa possa ser autora de uma obra. Dessa forma, se está restringindo o domínio da propriedade "é autor".

A Figura 2.6 demonstra a separação entre as camadas do RDF e do RDFS.

Figura 2.6: Divisão das camadas do RDF e do RDFS (Adaptado de (ANTONIOU; HARMELEN, 2008))



O Trecho de Código 2.2 demonstra a representação da camada RDFS acima na sintaxe RDF/XML.

Trecho de Código 2.2: Exemplo de RDFS na sintaxe RDF/XML.

```
1 <?xml version="1.0"?>
2   <!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
3   <rdf:RDF
4     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
5     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
6     xml:base="http://example.org/schemas/arte">
```

```
 7
 8   <rdfs:Class rdf:ID="ObraDeArte"/>
 9
10   <rdfs:Class rdf:ID="ObraLiteraria">
11     <rdfs:subClassOf rdf:resource="#ObraDeArte"/>
12   </rdfs:Class>
13
14   <rdfs:Class rdf:ID="Pessoa"/>
15
16   <rdf:Property rdf:ID="eAutorDe">
17     <rdfs:domain rdf:resource="#Pessoa"/>
18     <rdfs:range rdf:resource="#ObraDeArte"/>
19   </rdf:Property>
20
21 </rdf:RDF>
```

2.1.3 Ontologia e *Web Ontology Language*

Outro padrão para representação semântica na Web é uma linguagem para representação de ontologias, OWL (*Web Ontology Language*). A OWL é "uma linguagem de marcação semântica para publicação e compartilhamento de ontologias na World Wide Web" (DEAN; SCHREIBER, 2004). A OWL oferece maior poder de representação semântica que as linguagens XML, RDF e RDFS, incluindo a descrição de classes, propriedades e indivíduos, permitindo, com isto, utilizar inferências da lógica descritiva.

Uma ontologia é "uma especificação formal e explícita de uma conceitualização compartilhada" (GRUBER, 1995). Em Ciência da Computação, ontologias são uma forma de descrever uma entidade, as partes que a constituem e as relações presentes entre elas, de maneira que sirvam a um determinado propósito. Elas podem ser vistas como um conjunto de conceitos relativos a uma entidade e as relações existentes entre esses conceitos (STAAB; STUDER, 2009).

A principal motivação da OWL é a limitação semântica do RDF e do RDFS. Essas tecnologias permitem a definição de classes e propriedades, adicionando a estas restrições quanto ao alcance e ao domínio. Porém, muitas aplicações da Web Semântica demandam uma maior expressividade quanto ao significado do conteúdo (ANTONIOU; HARMELEN, 2008).

Nesse sentido, a OWL é uma linguagem mais rica. Algumas das funcionalidades que ela proporciona são (ANTONIOU; HARMELEN, 2008):

- definir o âmbito local de uma propriedade;
- fazer relações entre classes, como a disjunção;
- restringir a cardinalidade de uma propriedade;
- combinação booleana de classes;
- definir características especiais de uma propriedade, como a transitividade, a

unicidade, propriedade inversa, entre outras.

Os principais requisitos esperados para uma linguagem de ontologias, de acordo com ANTONIOU; HARMELEN (2008), são: sintaxe bem definida, suporte eficiente a inferência, semântica formal, poder de expressão suficiente e expressão conveniente. Nesse contexto, a OWL se divide em três sublinguagens, cada uma delas oferecendo um nível diferente em termos de eficiência de inferência e capacidade de expressão: *OWL Full*, *OWL DL* e *OWL Lite* (MCGUINNESS; HARMELEN, 2004).

A *OWL Full* configura-se não exatamente como uma sublinguagem, uma vez que permite a utilização de todos os construtores da OWL e é totalmente compatível com RDF. Na *OWL Full* existe a possibilidade de se representar uma classe e uma instância com o mesmo termo, por exemplo. Essa flexibilidade acaba tornando impraticável a realização de inferências sobre o conjunto de dados, uma vez que não há garantia de resposta em tempo hábil (ANTONIOU; HARMELEN, 2008) (MCGUINNESS; HARMELEN, 2004).

A sublinguagem que traz uma evolução em termos de eficiência de inferência é a chamada *OWL DL*. Ela permite a utilização de todas as funcionalidades idealizadas pela OWL, mas restringindo como elas podem ser aplicadas. Por exemplo, termo não pode definir ao mesmo tempo uma classe OWL e um recurso RDF. Com esse tipo de restrição, essa sublinguagem torna-se viável para inferências, garantindo que toda consulta terá um retorno em um tempo razoável (MCGUINNESS; HARMELEN, 2004).

A *OWL Lite* é a mais simples das sublinguagens da OWL. Permite a definição de classes com hierarquia e restrições simples de propriedades. Por exemplo, é possível restringir a cardinalidade de uma propriedade, mas somente com 0 ou 1 (MCGUINNESS; HARMELEN, 2004). Não estão inclusas no escopo da *OWL Lite* funções como a disjunção de classes, propriedades enumeráveis e cardinalidades mais complexas (ANTONIOU; HARMELEN, 2008). Apesar da pouca expressividade, essa sublinguagem tem como vantagem uma maior eficiência em termos de inferência.

A sintaxe utilizada pela OWL segue o mesmo padrão do RDF. O Trecho de Código 2.3 demonstra algumas funcionalidades da OWL representadas nessa sintaxe.

Trecho de Código 2.3: Exemplo de ontologia representada em OWL.

```
<owl:Ontology rdf:about="">
    <owl:imports rdf:resource="http://www.minhasontologias.org/pessoas"/>
    <rdfs:label>Ontologia Empresa de Software</rdfs:label>

    <owl:Class rdf:ID="programador">
        <rdfs:subClassOf rdf:resource="#funcionarios"/>
    </owl:Class>

    <owl:Class rdf:about="#programador">
        <owl:disjointWith rdf:resource="#programadorPython"/>
```

```
11          <owl:disjointWith rdf:resource="#programadorC"/>
12      </owl:Class>
13
14      <owl:Class rdf:ID="desenvolvedor">
15          <owl:equivalentClass rdf:resource="#programador"/>
16      </owl:Class>
17
18      <owl:ObjectProperty rdf:ID="programadaPor">
19          <rdfs:domain rdf:resource="#funcionalidade"/>
20          <rdfs:range rdf:resource="#programador"/>
21      </owl:ObjectProperty>
22
23      <owl:ObjectProperty rdf:ID="programou">
24          <rdfs:range rdf:resource="#funcionalidade"/>
25          <rdfs:domain rdf:resource="#programador"/>
26          <owl:inverseOf rdf:resource="#programadaPor"/>
27      </owl:ObjectProperty>
28
29      <owl:ObjectProperty rdf:ID="desenvolveu">
30          <owl:equivalentProperty rdf:resource="programou"/>
31      </owl:ObjectProperty>
32
33      <owl:Class rdf:about="#funcionalidade">
34          <owl:Restriction>
35              <owl:onProperty rdf:resource="#programadaPor"/>
36              <owl:Cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:
                    Cardinality>
37          </owl:Restriction>
38      </owl:Class>
39
40      <owl:Class rdf:about="#funcionalidadeEmC">
41          <rdfs:subClassOf rdf:resource="#funcionalidade">
42              <owl:Restriction>
43                  <owl:onProperty rdf:resource="#programadaPor"/>
44                  <owl:allValuesFrom rdf:resource="#programadorC"/>
45              </owl:Restriction>
46          </rdfs:subClassOf>
47      </owl:Class>
48
49      <owl:Class rdf:ID="equipeDesenvolvimento">
50          <owl:unionOf rdf:parseType="Collection">
51              <owl:Class rdf:about="#programador"/>
52              <owl:Class rdf:about="#analista"/>
53          </owl:unionOf>
54      </owl:Class>
55  </owl:Ontology>
```

A cláusula *owl:imports* permite que outras ontologias sejam utilizadas e estendidas na ontologia que está sendo definida. A cláusula *owl:disjointWith* permite fazer a disjunção de classes. No exemplo, a classe programador pode ser um *programadorPython* ou um *programadorC*. A equivalência entre classes é definida pela cláusula *owl:equivalentClass*. A união de duas ou mais classes é feita através da cláusula *owl:unionOf*. Propriedades inversas, como *programadaPor* e *programou* no exemplo, são descritas pela cláusula *owl:inverseOf*. Restrições de propriedades ficam aninhadas em *owl:Restriction*. A propriedade onde será

aplicada a restrição é definida em *owl:onProperty*. Restrições de cardinalidade são definidas pela cláusula *owl:Cardinality*. Para determinar o tipo de valor que pode ter uma propriedade de uma classe, utiliza-se a cláusula *owl:allValuesFrom*. No exemplo, é definido que a propriedade *programadaPor* da classe *funcionalidadeEmC* só pode ser ter como valor um *programadorC*.

2.2 Conclusões do Capítulo

Como visto ao longo deste capítulo, a Web Semântica busca estruturar as informações e incorporar significados a elas. A sintaxe XML encarrega-se da primeira parte, enquanto o RDF, RDFS e OWL incorporam, de forma crescente, sentido ao conteúdo da rede. É sobre esses pilares que serão especificados os Registros de Colaboração, objetivo maior deste trabalho. Através de tecnologias que agregam uma sintaxe bem definida e expressividade de significado, é que serão feitas as inferências aos conteúdos gerenciados através do *Framework CoP*.

3 TRIPLE STORE

Com o aumento da utilização de tecnologias como o RDF e o RDFS, surgiu a necessidade de criação de repositórios capazes de fazer a persistência de tais tipos de dados. São os denominados *triple stores*. Essas ferramentas possuem funcionalidades que permitem o armazenamento e acesso das triplas RDF (BERNERS-LEE; HALL; SHADBOLT, 2006). A consulta a essas informações é feita através da linguagem SPARQL.

O principal aspecto de um *triple store* é a capacidade de fazer inferências a partir de consultas. Segundo HOUAISS (2001), inferência é a "operação intelectual por meio da qual se afirma a verdade de uma proposição em decorrência de sua ligação com outras já reconhecidas como verdadeiras". Ou seja, a característica fundamental de um *triple store* é a capacidade de "descobrir" fatos a partir de relações já existentes. Um exemplo de inferência é apresentado na sequência de Trechos de Código 3.1, 3.2, 3.3, 3.4 e 3.5.

Trecho de Código 3.1: Exemplo de ontologia.

```
1 ex1:ProgramadorC  rdf:subClassOf  ex1:Programador.
2 ex1:ProgramadorPython  rdf:subClassOf  ex1:Programador.
3 ex1:Programador  owl:equivalentClass  ex2:Desenvolvedor
```

A ontologia descrita no Trecho de Código 3.1 define que um "ProgramadorC" e um "ProgramadorPython" são ambos "Programadores", e que um "Programador" é equivalente a um "Desenvolvedor".

Trecho de Código 3.2: Exemplo de dados.

```
1 ex1:Paulo  rdf:type  ex1:ProgramadorC.
2 ex1:Alice  rdf:type  ex1:ProgramadorPython.
3 ex2:Maria  rdf:type  ex2:Desenvolvedor
```

O Trecho de Código 3.2 define que Paulo é um "ProgramadorC", Alice é uma "ProgramadoraPython" e Maria é uma "Desenvolvedora".

Trecho de Código 3.3: Exemplo de consulta.

```
1 SELECT ?x
2  WHERE {
3  ?x rdf:type ex1:Programador
4  }
```

Sem a capacidade de inferência, nenhum resultado seria retornado, pois não há a informação direta de que alguém seja "Programador". No entando, é possível inferir que Paulo, Alice e Maria são "Programadores", porque se sabe que "ProgramadorC", "ProgramadorPython" e "Desenvolvedor" são também "Programador".

Outro exemplo, Trecho de Código 3.4, utilizando a primitiva *owl:sameAs*

Trecho de Código 3.4: Exemplo de dados.

```
1 ex1:Paulo foaf:name "Paulo Silva" .
2 ex1:Paulo owl:sameAs ex2:Silva .
3 ex2:Silva foaf:phone "9999-9999" .
```

Dada a consulta SPARQL do Trecho de Código 3.5:

Trecho de Código 3.5: Exemplo de consulta.

```
1 SELECT ?name ?phone
2  WHERE {
3  ex1:Paulo foaf:name ?name .
4  ex1:Paulo foaf:phone ?phone .
5  }
```

Sem inferência, não haveria retorno para a consulta, pois não há tripla onde o sujeito é "Paulo" e o predicado é "phone". Entretando, através de inferência, descobre-se que o "phone" de "Paulo" é "9999-9999", pois "ex1:Paulo" é o mesmo que "ex2:Silva", e o "phone" de "ex2:Silva" é "9999-9999".

A grande diferença entre esses bancos de dados e os bancos de dados relacionais é o tipo de informação que é armazenada. Enquanto bancos relacionais armazenam tabelas de dados, os *triple stores* armazenam triplas RDF (SEQUEDA, 2013).

Outro ponto em que os *triple stores* diferem dos bancos relacionais é em relação à modelagem da base de dados. Enquanto bancos relacionais necessitam de um *schema* previamente definido, com as colunas das tabelas determinadas com antecipação, assim como as relações entre elas, os bancos de triplas são muito mais flexíveis nesse sentido. Os *triple stores* são dinâmicos em relação à sua estrutura. Novos predicados podem ser adicionados automaticamente, e utilizados em consultas. Também não é necessário estabelecer relações entre entidades previamente, uma vez que as essas ligações são feitas de forma direta (AASMAN, 2011).

Dependendo do tipo de implementação, os *triple stores* podem ser classificados em três categorias. Os *triple stores* em memória caracterizam-se por armazenarem as triplas do grafo RDF na memória principal da máquina. Os *triple stores* nativos

possuem uma base de dados própria onde são guardadas as informações. Já os não-nativos não em memória utilizam bancos de dados de terceiros para fazer a persistência dos dados (BHATIA, 2013).

Um *triple store* em memória, devido a sua natureza, não é utilizado para o armazenamento de grandes quantidades de dados. Por outro lado, ferramentas desse tipo podem ser utilizadas para outros propósitos, entre os quais estão o cacheamento de dados remotos e a execução de inferências. Em geral, as implementações que seguem esse padrão de *triple store* possuem raciocinadores eficientes (BHATIA, 2009), além de serem bastante aderentes ao modelo RDF.

Por sua vez, os *triple stores* não-nativos não em memória aproveitam-se do bom suporte a concorrência, segurança, recuperação de desastres entre outras funcionalidades que são disponibilizadas pelas implementações tradicionais de bancos relacionais. Por outro lado, esses *triple stores* afastam-se um pouco do modelo de dados proposto pela Web Semântica, uma vez que as triplas RDF precisam ser mapeadas para tabelas relacionais (FAYE; CURE; BLIN, 2012).

Já os *triple stores* nativos oferecem uma plataforma mais aderente ao modelo semântico, uma vez que são projetados com base no modelo de triplas. Um dos principais motivos para o uso desse tipo de *triple store* é justamente esse, o de aproveitar as vantagens do modelo RDF, como a flexibilidade de *schema* (FAYE; CURE; BLIN, 2012).

3.1 Linguagem de Consulta - SPARQL

A linguagem que foi idealizada para a realização de consultas em bases de dados *triple store* é a SPARQL. Segundo PEREZ; ARENAS; GUTIERREZ (2006), essa é "uma linguagem de correspondência de grafos". Através da SPARQL é possível fazer consultas por determinados padrões de grafos, suas conjunções e disjunções. A linguagem também permite que sejam feitas agregações, sub-consultas, negação, criação de valores através de expressões, entre outras operações nas consultas (HARRIS; SEABORNE, 2013).

Essa linguagem assemelha-se ao SQL (*Structured Query Language*), utilizado para consultar bancos de dados relacionais, e permite a combinação de dados de diferentes fontes (PINHATI et al., 2012). Sua sintaxe é bastante parecida com a sintaxe do SQL, o que torna a migração dos desenvolvedores acostumados às consultas em bancos relacionais menos traumática.

Uma consulta SPARQL consiste basicamente de um conjunto de triplas chamado de modelo básico de grafo. Essas triplas são similares a triplas RDF, com a diferença que tanto o sujeito quanto o predicado e o objeto podem ser variáveis. Esse modelo básico é então comparado a subgrafos do grafo principal. Caso algum subgrafo seja

equivalente ao modelo básico, esse subgrafo é retornado como resultado da consulta. Os trechos de código 3.6 e 3.7 ilustram algumas consultas básicas na linguagem SPARQL sobre o conjunto de dados representado na Tabela 3.1.

Tabela 3.1: Exemplo de conjunto de dados (adaptado de (HARRIS; SEABORNE, 2013))

Sujeito	**Predicado**	**Objeto**
<http://exemplo.org/livro/livro1>	<http://exemplo.org/predicados/titulo>	"Romeu e Julieta".
<http://exemplo.org/livro/livro1>	<http://exemplo.org/predicados/autor>	"William Shakespeare".
<http://exemplo.org/livro/livro2>	<http://exemplo.org/predicados/titulo>	"Viagem ao Centro da Terra".
<http://exemplo.org/livro/livro2>	<http://exemplo.org/predicados/autor>	"Julio Verne".
<http://exemplo.org/livro/livro3>	<http://exemplo.org/predicados/titulo>	"1984".
<http://exemplo.org/livro/livro3>	<http://exemplo.org/predicados/autor>	"George Orwell".

A consulta ilustrada no Trecho de Código 3.6 faz a busca pelo(s) objeto(s) da(s) tripla(s) que contém o predicado "título" (<http://exemplo.org/predicados/titulo>) do sujeito "livro" (<http://exemplo.org/livro/livro1>). O resultado dessa consulta é apresentado na Tabela 3.2.

Trecho de Código 3.6: Exemplo de consulta.

```
1 SELECT ?titulo
2 WHERE
3 {
4   <http://exemplo.org/livro/livro1> <http://exemplo.org/predicados/titulo> ?
        titulo .
5 }
```

Tabela 3.2: Retorno da consulta 3.6

titulo
"Romeu e Julieta"

Outra consulta, ilustrada no Trecho de Código 3.7, apresenta a cláusula "PREFIX". Através dela é possível definir um prefixo que será utilizado nas consultas. Por exemplo, ao invés de ser necessário referenciar o caminho completo do *namespace* "<http://exemplo.org/predicados/>" toda vez que o mesmo for utilizado, define-se um prefixo ("ex", neste caso), o que facilita a escrita das consultas. A consulta 3.7 efetua uma busca dos objetos relacionados aos predicados "ex:autor" e "ex:titulo". O resultado dessa consulta consta na Tabela 3.4

Trecho de Código 3.7: Exemplo de consulta.

```
1 PREFIX ex: <http://exemplo.org/predicados/>
2 SELECT ?autor ?titulo
3 WHERE
4   { ?x ex:autor ?autor .
5     ?x ex:titulo ?titulo }
```

Tabela 3.3: Retorno da consulta 3.7

autor	titulo
"William Shakespeare"	"Romeu e Julieta"
"Julio Verne"	"Viagem ao Centro da Terra"
"George Orwell"	"1984"

3.2 *Frameworks triple store*

Existem muitas ferramentas disponíveis para o desenvolvimento de aplicações utilizando *triple stores*. A seguir serão brevemente apresentadas algumas das ferramentas mais utilizadas pela comunidade de desenvolvimento: Jena, Sesame, Virtuoso e 4store. Estas ferramentas possuem características básicas referentes ao armazenamento, consulta e representação em RDF, além de capacidade de armazenamento e desempenho. A Tabela 3.4 apresenta as principais características de cada uma, que serão detalhadas nas subseções 3.2.1, 3.2.2, 3.2.3 e 3.2.4.

Tabela 3.4: Características dos *frameworks triple store*

Framework	Capacidade (triplas)	Nativo	Relacional	Licença
Jena	1.7B (TDB) / 650M (SDB)	X	X	Apache 2
Sesame	70M (Nativo)	X	X	BSD
Virtuoso	15.4B+	X	X	GPLv2 ou Comercial
4store	15B	X		GPLv3

3.2.1 Jena

O Apache Jena é um *framework* desenvolvido em linguagem Java para a criação de aplicações da Web Semântica. Esta ferramenta oferece uma série de funcionalidades, entre as quais estão uma API (*Application Programming Interface*) para o processamento de dados em formato RDF com suporte a diversas sintaxes, uma API para lidar com ontologias OWL e RDFS, um mecanismo de inferência para dados RDF e OWL, um banco de dados com capacidade para armazenar um grande número de triplas RDF de forma eficiente, um mecanismo de consulta compatível com SPARQL e a possibilidade de publicação de dados RDF para outras aplicações, utilizando-se diversos protocolos. Seu desenvolvimento é fomentado pela *Apache Software Foundation*, sob a licença Apache 2 (JENA, 2013).

No Jena, os dados podem ser armazenados através do TDB, um banco *triple store* nativo de alta performance, ou do SDB, que faz a persistência de dados em bancos de dados relacionais, como o PostgreSQL ou MySQL. A capacidade do Jena utilizando-se o TDB pode chegar a 1.7 bilhões de triplas, enquanto que com o uso do SDB a capacidade máxima é de cerca de 650 milhões de triplas (W3C, 2011).

3.2.2 Sesame

O Sesame é um *framework* para processamento de dados em formato RDF. Seu desenvolvimento é impulsionado pela empresa holandesa *Aduna*. Seu código é distribuído sob a licença BSD. Assim como o Jena, é desenvolvido em linguagem Java.

Esse *framework* possui funcionalidades para análise, persistência, inferência e consulta sobre triplas RDF. A persistência desses dados pode ser feita tanto em um banco nativo, como em bancos relacionais. O Sesame oferece ainda uma interface para consultas SPARQL e suporte para as principais sintaxes RDF (OPENRDF, 2013). Sua capacidade aproximada, utilizando-se o armazenamento nativo, é de 70 milhões de triplas (W3C, 2011). Um dos destaques do Sesame é a sua interface interativa e amigável, além de sua fácil instalação e configuração.

3.2.3 Virtuoso

O Virtuoso Universal Server é uma plataforma para gerenciamento de dados que pode ser aplicada para vários modelos de dados, entre eles o modelo RDF. Esta ferramenta possui funcionalidades que vão desde a geração e persistência das informações, que pode ser feita em um *triple store* nativo ou em bancos relacionais, até a disponibilização de um ponto de execução de consultas SPARQL e suporte a ontologias. A grande gama de possibilidades que a plataforma provê, apesar de facilitar a integração entre diferentes modelos de dados, torna seu uso mais complexo. A capacidade aproximada de armazenamento do *triple store* do Virtuoso é de de mais de 15.4 bilhões de triplas (VIRTUOSO, 2013) (W3C, 2011).

3.2.4 4store

O 4store é um mecanismo para armazenamento e consulta de dados RDF, desenvolvido pela empresa inglesa *Garlik*. É um software livre, distribuído sob a licença GPLv3. Apesar de não fornecer outras funcionalidades além da persistência e consultas SPARQL sobre os dados RDF, o 4store possui notoriedade em termos de performance, escalabilidade e estabilidade. O armazenamento dos dados é feito em uma base de dados nativa, sem suporte a bancos relacionais. Em certas aplicações, como o *DataPatrol* da *Garlik*, a capacidade deste *framework* chega a 15 bilhões de triplas (4STORE, 2013) (W3C, 2011).

3.3 Trabalhos Relacionados

Muitas aplicações voltadas para Web semântica fazem o uso de *triple stores* para a persistência dos seus dados. Estas aplicações possuem os mais diversos âmbitos, sendo aplicadas desde pesquisas científicas até órgãos de mídia. Algumas destas aplicações são apresentadas nas subseções 3.3.1, 3.3.2 e 3.3.3.

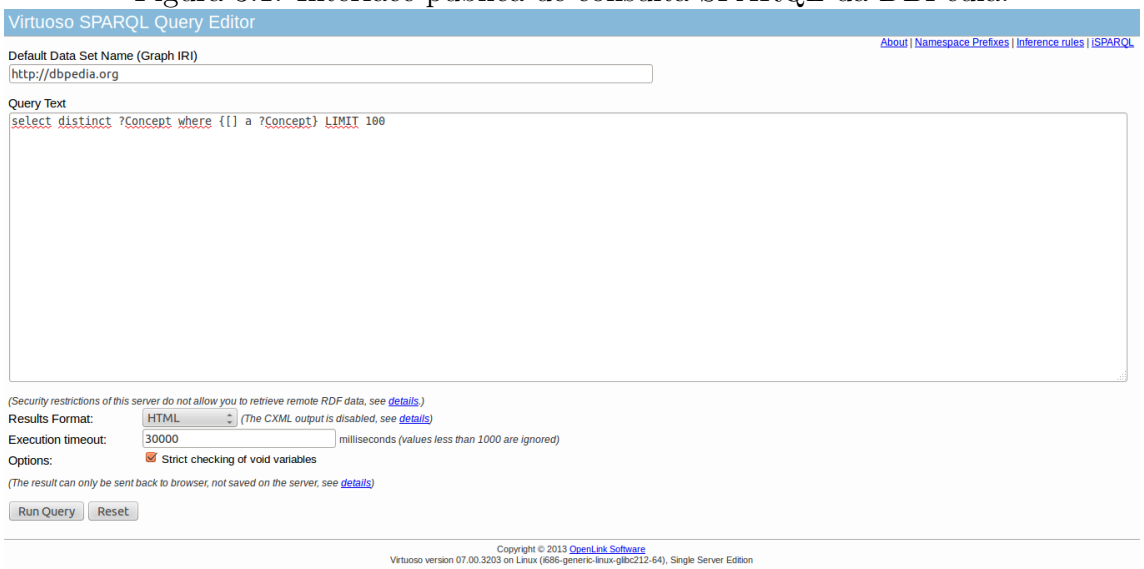
3.3.1 DBPedia

A DBPedia é um projeto que tem como objetivo extrair dados da Wikipedia e disponibilizar essas informações de forma estruturada. O propósito disso é permitir que outras aplicações possam fazer consultas mais refinadas sobre o conteúdo da Wikipedia, que vão além da simples busca textual por palavra chave.

Devido à grande diversidade de conteúdo disponibilizado pela DBPedia, ela vem se tornando uma central de informações semânticas da web. É grande o número de outras aplicações que se utilizam de informações estruturadas através da DBPedia, criando uma grande rede de dados em torno da mesma (BIZER et al., 2009). Atualmente a versão em inglês da DBPedia descreve cerca de 3.77 milhões de termos.

A Figura 3.1 ilustra a interface de consulta SPARQL disponibilizada pela DBPedia[1].

Figura 3.1: Interface pública de consulta SPARQL da DBPedia.



[1] http://dbpedia.org/sparql

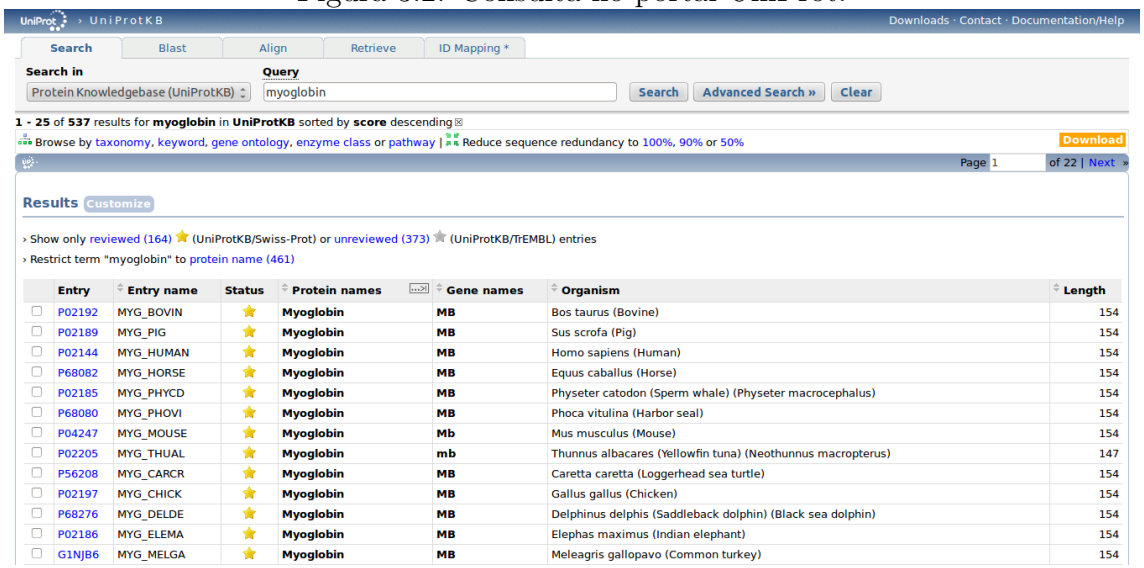
3.3.2 UniProt

O *Universal Protein Resource* (UniProt) é um projeto que disponibiliza um abrangente recurso sobre sequenciamento de proteínas e dados anotados, formado por um consórcio Europeu que mantém mais de 120 bancos de dados e coleções de dados de Bioinformática (APWEILER; BAIROCH; WU, 2004). A principal missão do Consórcio UniProt é apoiar a investigação biológica através da manutenção de uma sociedade estável, abrangente e totalmente classificada, ricamente anotada e precisa de uma base de conhecimento sobre sequenciamento de proteínas, mantendo extensas referências cruzadas e consulta de interfaces de livre acesso para a comunidade científica.

Alguns dos requisitos dessa iniciativa precisam lidar com desempenho em relação à recuperação de dados a partir de consultas. Para atingir este objetivo, o consórcio UniProt utiliza representação de ontologias em triplas RDF armazenadas em um banco de *triple store*, que precisam tratar cerca de 13 milhões de entradas nas bases de entrada integradas dos países participantes do consórcio (MAGRANE; CONSORTIUM, 2011).

A Figura 3.2 ilustra uma consulta pela proteína mioglobina (em inglês, *myoglobin*), através do portal do UniProt[2].

Figura 3.2: Consulta no portal UniProt.



Entry	Entry name	Status	Protein names	Gene names	Organism	Length
P02192	MYG_BOVIN	★	Myoglobin	MB	Bos taurus (Bovine)	154
P02189	MYG_PIG	★	Myoglobin	MB	Sus scrofa (Pig)	154
P02144	MYG_HUMAN	★	Myoglobin	MB	Homo sapiens (Human)	154
P68082	MYG_HORSE	★	Myoglobin	MB	Equus caballus (Horse)	154
P02185	MYG_PHYCD	★	Myoglobin	MB	Physeter catodon (Sperm whale) (Physeter macrocephalus)	154
P68080	MYG_PHOVI	★	Myoglobin	MB	Phoca vitulina (Harbor seal)	154
P04247	MYG_MOUSE	★	Myoglobin	Mb	Mus musculus (Mouse)	154
P02205	MYG_THUAL	★	Myoglobin	mb	Thunnus albacares (Yellowfin tuna) (Neothunnus macropterus)	147
P56208	MYG_CARCR	★	Myoglobin	MB	Caretta caretta (Loggerhead sea turtle)	154
P02197	MYG_CHICK	★	Myoglobin	MB	Gallus gallus (Chicken)	154
P68276	MYG_DELDE	★	Myoglobin	MB	Delphinus delphis (Saddleback dolphin) (Black sea dolphin)	154
P02186	MYG_ELEMA	★	Myoglobin	MB	Elephas maximus (Indian elephant)	154
G1NJB6	MYG_MELGA	★	Myoglobin	MB	Meleagris gallopavo (Common turkey)	154

3.3.3 BioPortal

O BioPortal é um repositório de ontologias biomédicas. Este portal oferece uma interface através da qual é possível navegar, pesquisar e visualizar estas ontologias. Esta interface ainda facilita a participação da comunidade na construção das

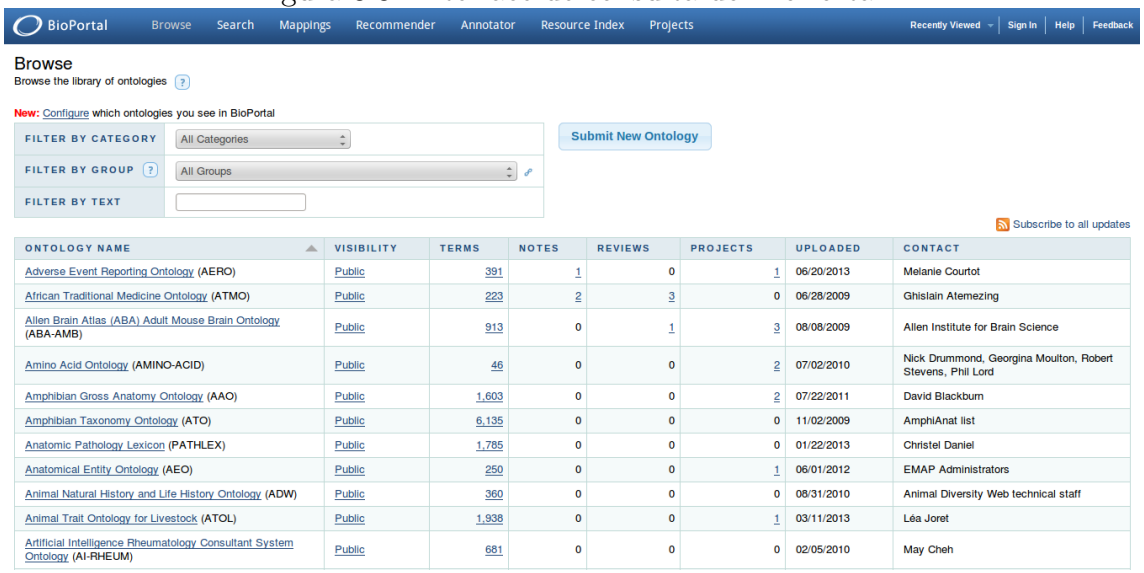
[2]http://www.uniprot.org/

ontologias, oferecendo ferramentas para anotação e avaliação das ontologias, bem como a relação destas com outras fontes de dados (NOY et al., 2009).

Este repositório contém mais de 300 ontologias, totalizando mais de 203 milhões de triplas representando conteúdo e metadados e 9 milhões de triplas com relações entre termos (SALVADORES et al., 2012). Estas ontologias estão descritas em diversos formatos, sendo persistidas em bancos de dados *triple store*.

A Figura 3.3 ilustra a interface de consultas de ontologias do BioPortal[3].

Figura 3.3: Interface de consulta do BioPortal.



3.4 Conclusões do capítulo

Como visto nas seções anteriores, os *triple stores* são bancos de dados projetados para o armazenamento de dados em formato RDF. Sendo assim, esses bancos seguem o paradigma da Web Semântica, mantendo uma estrutura padrão compatível com os outros componentes existentes nesse contexto.

Como o objetivo dessa proposta é a implementação de Registros de Colaboração em um *framework* de Comunidades de Prática, de forma que essa implementação seja aderente ao modelo da Web Semântica, define-se que será feita a utilização de uma base de dados *triple store* nativa. Esse modelo é o que segue mais à risca o conceito de dados linkados, adequando-se à proposta deste trabalho.

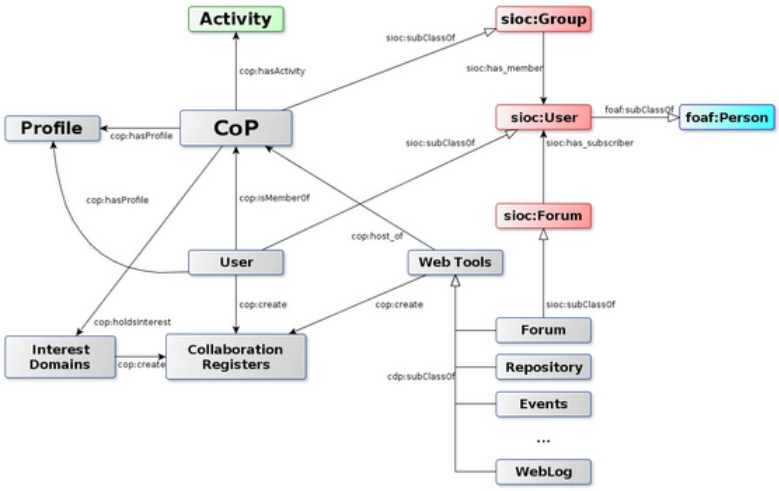
[3]http://bioportal.bioontology.org/ontologies

4 FRAMEWORK COP E TECNOLOGIAS

Conforme descrito no Capítulo 1, a implementação da proposta de solução se dará no *Framework* de Comunidades de Prática proposto por FIORIO; SILVA; RIBEIRO (2011) e RIBEIRO et al. (2011).

Esse *framework* disponibiliza um conjunto de ferramentas e componentes com funções de cadastro, autenticação, criação e gestão de conteúdo, todos voltados para o cenário de Comunidades de Prática. A Figura 4.1 representa o diagrama do *framework*, com base em uma ontologia de referência, representando seus componentes e as relações entre eles.

Figura 4.1: Ontologia de referência representando o *Framework CoP*

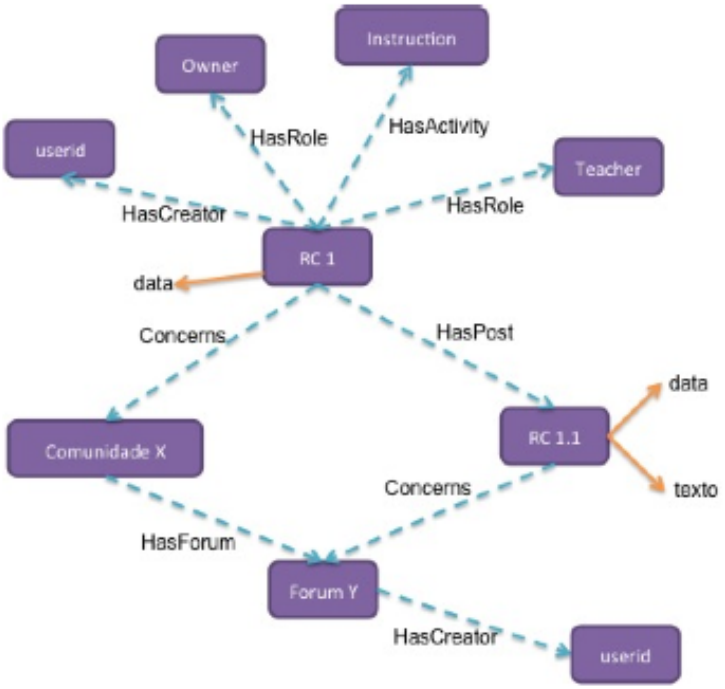


O foco desta proposta está na classe *Collaboration Registers* (Registros de Colaboração). Essa classe descreve as relações que podem ser estabelecidas entre aspectos e interesses de usuários através das interações destes em um ambiente dinâmico e distribuído. É através de conjuntos de propriedades que são descritas formas de relacionamentos que ocorrem entre sujeitos, comunidades e seus conteúdos, através de um modelo ontológico (RIBEIRO et al., 2011), (PRIMO et al., 2012).

A Figura 4.2 ilustra um exemplo da classe *Collaboration Registers* e vários

indivíduos que representam “colaborações”, ou seja, conjuntos de ações realizadas entre usuários ou entre usuários e conteúdos. Desta forma cada colaboração é descrita através de um indivíduo da ontologia. Os retângulos representam indivíduos, linhas pontilhadas propriedades de objetos e as planas propriedades de dados. O indivíduo RC 1 representa um registro de colaboração relacionado através da propriedades de objeto *HasCreator* ao indivíduo *userid* como forma de representar o usuário que o criou; a propriedade *HasRole* associada ao indivíduo *Owner* e ao indivíduo *Teacher* representando o papel assumido por este usuário neste Registro de Colaboração; a propriedade *Concerns* descreve que este registro de colaboração está relacionado a alguma comunidade de prática representada pelo indivíduo *Comunidade X*.

Figura 4.2: Composição de indivíduos e propriedades de um Registro de Colaboração



4.1 Implementação do Framework CoP

O *Framework* CoP é uma plataforma de construção de comunidades de prática virtuais. Seu desenvolvimento é feito em linguagem Python, tendo o Plone como gerenciador de conteúdos e o Zope como o servidor de aplicação. As seções seguintes dão uma visão geral sobre as tecnologias envolvidas na confecção desse ambiente e das ferramentas disponibilizadas por ele.

4.1.1 Python

Python é uma linguagem de programação de computadores de código aberto, multi-plataforma, orientada a objetos, dinâmica e fortemente tipada. A linguagem foi criada pelo holandês Guido van Rossum em 1990, a partir de outra linguagem já existente, chamada ABC (BORGES, 2010). Atualmente é utilizada por diversas empresas, agências e órgãos governamentais, como o Google e a NASA (LUTZ, 2011).

São muitos os fatores que incentivam o uso desta linguagem pela comunidade de desenvolvedores de software. Os principais que podem ser destacados são a síntaxe simples e legível, a qualidade do software produzido, a grande produtividade, a fácil portabilidade dos programas desenvolvidos, a grande quantidade de bibliotecas embutidas e a fácil integração com outras linguagens (LUTZ, 2009).

O Trecho de Código 4.1 mostra um exemplo de código em Python.

Trecho de Código 4.1: "Olá mundo" em Python

```
1 print "Ola, mundo"
```

Dentre as bibliotecas disponibilizadas pelo Python, algumas são de grande utilidade para a construção de aplicações voltadas para a Web Semântica. A *httplib*[1] e a *urllib*[2] são bibliotecas que fazem interface com o protocolo HTTP (*Hypertext Transfer Protocol*). A primeira é uma implementação de um cliente HTTP, enquanto a segunda permite o acesso a recursos por meio de URLs. Outra biblioteca que pode ser destacada é a *rdflib*[3]. Esta biblioteca possui diversas funcionalidades para trabalhar com dados no formato RDF.

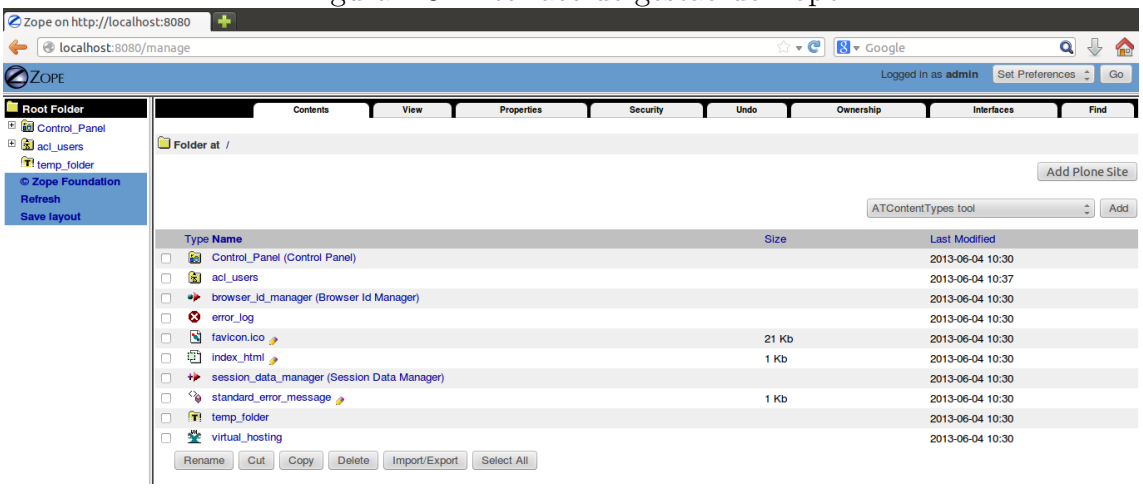
4.1.2 Zope

O *Z Object Publishing Environment* (Zope) é um servidor de aplicações web, desenvolvido majoritariamente em linguagem Python, com alguns trechos em linguagem C. Assim como a linguagem na qual foi desenvolvido, o Zope possui código aberto. Seu desenvolvimento foi iniciado pelo professor Jim Fulton, em 1996. Atualmente, o Zope é administrado pela Zope Corporation, e fomentado pela Zope Foundation (ZOPE, 2010).

O Zope possui uma interface de gestão chamada ZMI (Zope Management Interface, Figura 4.3), através da qual podem ser gerenciados produtos, objetos, configurações, entre outros. Por padrão, os objetos criados no Zope são armazenados em um banco de dados próprio, orientado a objetos, denominado *Zope Object Data Base* (ZODB).

[1] http://docs.python.org/2/library/httplib.html
[2] http://docs.python.org/2/library/urllib.html
[3] https://rdflib.readthedocs.org/

Figura 4.3: Interface de gestão do Zope



O ZODB é um banco de dados orientado a objetos desenvolvido, também, na linguagem Python. Algumas características marcantes desse banco são a fácil manipulação dos objetos através de scripts Python, sendo que não existe necessidade de mapeamento de objetos para tabelas de um banco relacional. Controle de transações, escalabilidade, suporte a conteúdos binários (BLOB) são outras funcionalidades presentes no ZODB.

A consulta de dados no ZODB é feita através de uma ferramenta chamada *portal_catalog*. Os objetos contidos dentro de uma base do ZODB possuem índices, pelos quais podem ser filtrados os resultados das buscas. Esses índices podem ser, por exemplo, o tipo do objeto, seu título ou descrição, o criador, entre outros (ZOPE, 2010).

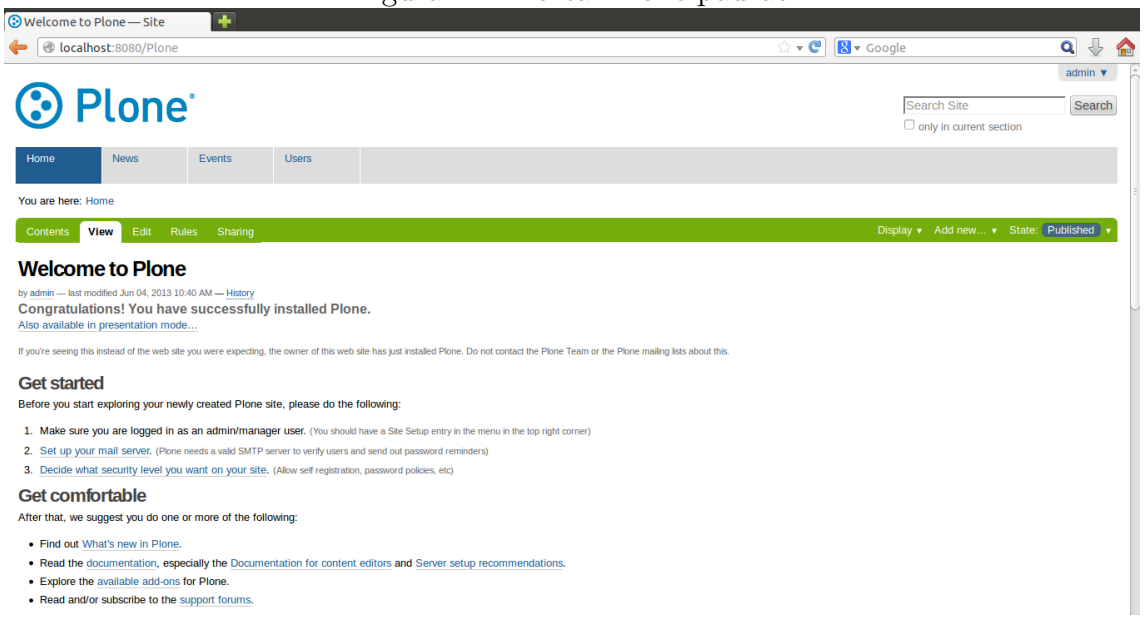
4.1.3 Plone

O Plone é um Gerenciador de Conteúdo (*Content Management System* - CMS) desenvolvido em Python, que roda como um produto do servidor de aplicações Zope. Assim como este, o Plone é um software livre, de código aberto. Seu desenvolvimento é promovido pela *Plone Foundation*, uma organização sem fins lucrativos (ASPELI, 2011).

Esse sistema oferece uma interface amigável para a criação e gestão de conteúdos através da Web. Para isso, o Plone possui tipos de conteúdo nativos, como Páginas, Imagens, Eventos e Notícias, além de permitir a criação de novos tipos. A flexibilidade de customização é um dos pontos fortes, sendo possível a criação de novas funcionalidades que são acopladas ao portal através de produtos. O maior destaque, porém, é na questão de segurança. O Plone possui um sistema bastante avançado de *workflow*, e é considerado um dos CMS mais seguros existentes no mercado (DELMONTE et al., 2009).

A Figura 4.4 ilustra a interface de um portal Plone padrão.

Figura 4.4: Portal Plone padrão



4.1.4 Ferramentas do *Framework CoP*

A implementação do *Framework CoP* disponibiliza uma plataforma para construção de comunidades de prática virtuais. Nessa plataforma está presente um conjunto de ferramentas que tem o intuito de promover a aprendizagem coletiva e compartilhada entre os membros dessas comunidades.

A plataforma oferece funcionalidades que vão desde a criação das comunidades em si até a criação, compartilhamento e gestão de conteúdos. É possível também o gerenciamento dos participantes, definindo seu papel na comunidade. Todas essas atividades são executadas por meio de uma interface amigável e intuitiva.

As ferramentas disponibilizadas no *Framework CoP* são:

- Acervo: é um repositório dos documentos da comunidade. Nele podem ser criados e armazenados documentos de texto, arquivos, imagens e links para outras páginas;
- Calendário: espaço destinado aos eventos da comunidade;
- Portfólio: local destinado ao armazenamento da produção individual dos membros. Os tipos de conteúdo que podem ser criados neste repositório são documento de texto, imagem e arquivo;
- Fórum: neste espaço ocorrem as discussões entre os membros da comunidade.
- Tarefas: local destinado a postagem de trabalhos e tarefas concluídas pelos membros;

- Atividades: resumo das atividades ocorridas na comunidade. Através dessa funcionalidade é possível monitorar o que foi criado e modificado na comunidade, e por qual membro essas ações foram executadas;
- Notificações: funcionalidade que permite o envio de um e-mail de notificação para todos os membros da comunidade. Tal ação só está disponível para os moderadores da comunidade. Além desse envio manual, é possível configurar o envio de notificações diárias das atividades da comunidade. Nesse caso, cada membro escolhe se quer ou não receber as notificações.
- Configurações: local onde os membros definem suas configurações pessoais. Neste espaço os moderadores também fazem a gestão dos participantes

A Figura 4.5 ilustra a interface do *Framework CoP*.

Figura 4.5: Interface do *Framework CoP*

4.2 Conclusões do Capítulo

Ao longo deste Capítulo foram descritas a implementação do *Framework CoP*, as ferramentas que esse *framework* disponibiliza e as tecnologias envolvidas no seu desenvolvimento. É dentro deste contexto específico que será implementado o protótipo proposto neste trabalho.

O *framework* ainda não tem implementados todos os seus componentes previstos na especificação. Em função disso, somente alguns campos já disponíveis no *framework* serão considerados na estruturação dos Registros de Colaboração da solução desenvolvida no Capítulo 5. Tais campos são:

- ID da Comunidade X (URL)

- ID do Criador
- Tipo do Conteúdo
- ID do Conteúdo (URL)
- Data de criação
- Tipo do Registro
- Contexto na Comunidade
- Domínio do Registro

O conjunto destas informações representa os registros de colaboração e interação ocorridos na comunidade. Através destes campos é possível estabelecer relações entre comunidades, entre membros de uma mesma comunidade e entre membros de comunidades distintas, além de oferecer uma base de metadados para consultas contextuais se armazenada como triplas.

5 O SISTEMA COP.SEMANTIC

Com base nos estudos apresentados nos Capítulos anteriores, elaborou-se uma solução para o problema apresentado no Capítulo 1, sobre a implementação da classe Registros de Colaboração do *Framework CoP* de forma aderente ao modelo de Web Semântica.

A plataforma de Comunidades de Prática que implementa os módulos básicos do CMS Plone foi desenvolvida em módulos. O módulo principal envolve a criação, descrição e desenvolvimento da comunidade de prática com as ferramentas colaborativas, que encontra-se em produção. Para a validação da proposta de representação semântica dos Registros de colaboração, optou-se pelo desenvolvimento de um protótipo que faz um mapeamento das ações colaborativas na plataforma para compor os Registros de Colaboração.

Durante o projeto do protótipo, definiu-se que dois tipos específicos de colaborações serão considerados para a geração dos Registros e persistência na *triple store*: a criação de conteúdos e a participação em uma comunidade. Estes dois tipos foram escolhidos por já ser possível mapear seus campos fundamentais a partir da implementação já disponível do *Framework CoP*. Suas estruturas serão detalhadas nas seções seguintes.

Na prototipação foi criado um produto Plone para que as novas funções semânticas possam ser integradas à implementação do *Framework CoP*. Tal produto foi denominado *cop.semantic*[1]. A opção pelo desenvolvimento do protótipo em um produto separado foi motivada pela necessidade de não se criar uma dependência entre o *framework* e o *cop.semantic*. Como a plataforma de Comunidades de Prática já está em produção e o *cop.semantic* encontra-se em fase experimental, não é recomendável que o primeiro dependa do segundo. A inexistência de dependências possibilita que o *framework* continue sendo funcional.

[1] https://bitbucket.org/jtmolon/cop.semantic

5.1 Projeto do Protótipo

O objetivo fundamental do *cop.semantic* é a geração dos Registros de Colaboração em formato semântico. Os Registros foram estruturados seguindo o modelo de dados RDF, utilizando-se a sintaxe RDF/XML. Para a geração e manipulação dos dados neste formato foi utilizada a biblioteca Python *rdflib*, detalhada na subseção 4.1.1. Essa biblioteca possui diversas funcionalidades que visam o gerenciamento de grafos RDF.

Outro aspecto importante dentro do escopo deste trabalho, como sugere o próprio título, diz respeito à persistência dos dados em RDF em um banco de triplas. Para isso foi utilizado o *triple store* nativo do *framework* Sesame. O Sesame foi escolhido pela sua fácil instalação e configuração, além de oferecer uma interface amigável e intuitiva para a consulta dos dados. O *framework* será hospedado sobre o servidor web *Apache Tomcat*[2]. Esta escolha se deve ao fato deste servidor ser específico para aplicações Java, que é o caso do Sesame, tornando a configuração do ambiente menos complexa.

O produto *cop.semantic* funciona no *background* do *Framework CoP*. A partir das interações dos usuários com a interface do *framework*, o protótipo faz a geração dos Registros de Colaboração. Como não há uma ferramenta nativa para a comunicação direta entre o Plone e o Sesame, é necessária a utilização de mecanismo que faça este intercâmbio de dados entre a aplicação e o *triple store*. Para esta função foi escolhido o protocolo HTTP.

Devido à implementação da plataforma sobre a pilha Python/Zope/Plone, escolheram-se ferramentas dentro deste escopo para a comunicação via HTTP. Foram utilizadas as bibliotecas *httplib* e a *urllib*, padrão da linguagem Python. Estas bibliotecas possuem funcionalidades de alto nível que permitem a comunicação entre serviços que utilizam o protocolo HTTP. A implementação da interface entre *triple store* e produto através de HTTP ainda torna a solução facilmente portável para outro *framework triple store*, uma vez que o padrão para comunicação utilizado pela maioria dos *frameworks* é justamente o protocolo HTTP.

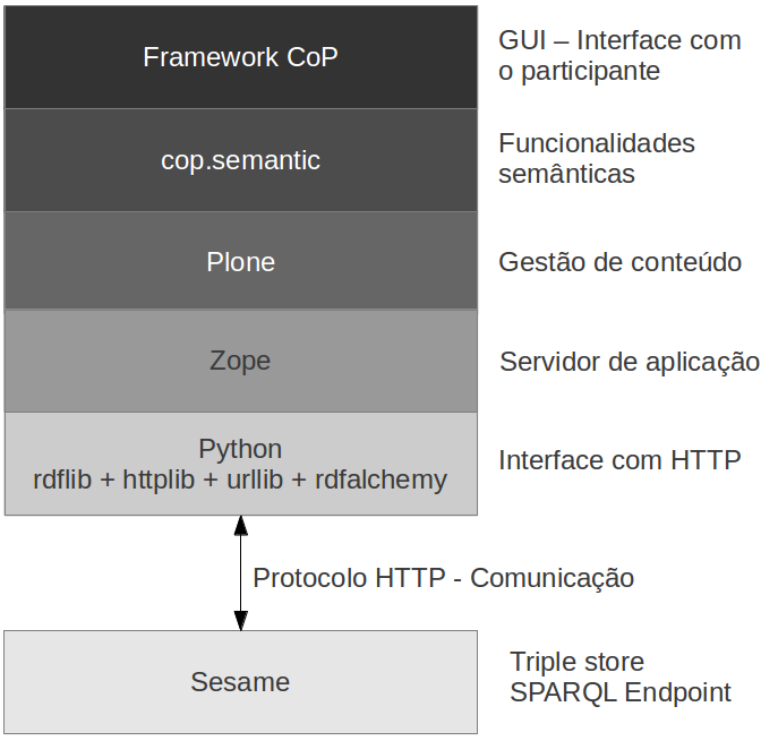
A última ferramenta a compor esta solução é o *RDFAlchemy*[3]. Trata-se de um ORM (*Object RDF Mapper*), que é um componente que possibilita fazer o mapeamento dos tipos de recursos RDF em classes Python. A partir desta ferramenta, será modelada a representação RDF para as classes e recursos dos Registros de Colaboração descritos no Capítulo 4. O uso deste componente torna a geração dos dados RDF mais transparente, pois ela é feita diretamente com código Python, sem que seja necessário utilizar a sintaxe do RDF/XML de forma

[2]http://tomcat.apache.org/
[3]http://www.openvest.com/trac/wiki/RDFAlchemy

explícita.

A Figura 5.1 ilustra a arquitetura proposta na solução.

Figura 5.1: Diagrama da arquitetura da solução



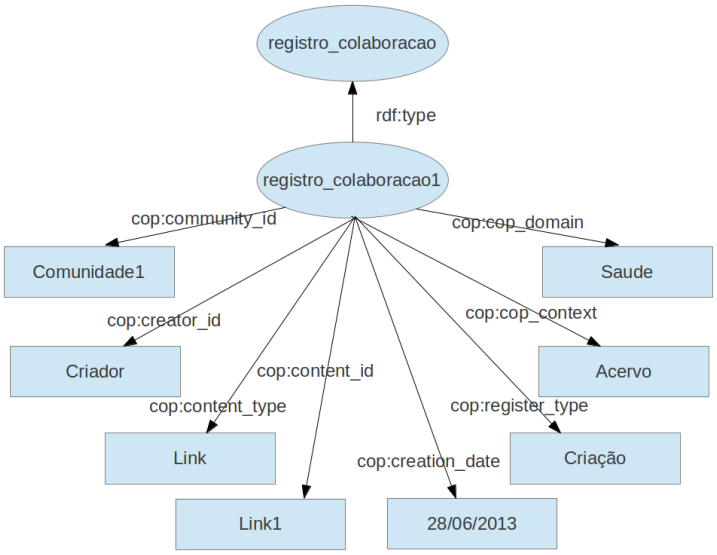
Conforme os estudos realizados nesse trabalho, foram definidos os campos que serão utilizados para estruturar os Registros de Colaboração. Cabe ressaltar que neste trabalho não será descrita a informação completa definida para o *Framework* de Comunidades de Prática. O escopo de representação será limitado aos campos ilustrados na Tabela 5.1.

Tabela 5.1: Campos do Registro de Colaboração

Campo	**Conteúdo armazenado**
community_id	URL da Comunidade onde foi criado o conteúdo
creator_id	ID do participante que criou o conteúdo
content_type	Tipo do conteúdo criado (Página, Link, Imagem, etc.)
content_id	URL do conteúdo criado
creation_date	Data de criação do conteúdo
register_type	Tipo do Registro (criação ou participação)
cop_context	Contexto na Comunidade (acervo, portfólio, etc.)
domain	Domínio do Registro

A Figura 5.2 ilustra um grafo representando um exemplo de Registro de Colaboração que segue o modelo proposto. No Trecho de Código 5.1 é ilustrada a representação deste mesmo exemplo na sintaxe RDF/XML.

Figura 5.2: Grafo de um exemplo de Registro de Colaboração



Trecho de Código 5.1: Exemplo de Registro de Colaboração em sintaxe RDF/XML

```
 1 <rdf:Description rdf:about="registro_colaboracao1">
 2     <rdf:type rdf:resource="CollaborationRegister"/>
 3     <cop:community_id>Comunidade1</cop:community_id>
 4     <cop:creator_id>Criador</cop:creator_id>
 5     <cop:content_type>Link</cop:content_type>
 6     <cop:content_id>Link1</cop:content_id>
 7     <cop:creation_date>28-06-2013</cop:creation_date>
 8     <cop:register_type>Criacao</cop:register_type>
 9     <cop:cop_context>Acervo</cop:cop_context>
10     <cop:domain>Saude</cop:domain>
11 </rdf:Description>
```

5.2 Desenvolvimento

A primeira etapa do desenvolvimento do *cop.semantic* foi justamente a criação do produto Plone. Este produto é a base para todas as funcionalidades implementadas no protótipo, respeitando a necessidade de não ser criada uma dependência entre o *framework CoP* e as novas funções semânticas, confome dito anteriormente.

A etapa seguinte no desenvolvimento foi a definição das propriedades referentes aos campos do Registro de Colaboração, em formato RDFS, utilizando-se a sintaxe RDF/XML. A representação das propriedades nesse formato adiciona semântica aos dados que serão gerados em RDF, criando um vocabulário para a criação desses dados.

Conforme o Trecho de Código 5.2, a classe *CollaborationRegister* representa um Registro de Colaboração propriamente dito, e a classe *CommunityOfPractice* representa uma Comunidade de Prática. Já as propriedades *community_id*,

creator_id, *content_type*, *content_id*, *creation_date*, *register_type*, *cop_context* e *domain* representam, respectivamente, a Comunidade de Prática onde foi gerado o registro, o criador do registro, a que tipo de conteúdo o registro se refere, qual o identificador deste registro, a data em que o mesmo foi criado, o tipo do Registro de Colaboração (criação ou participação), o contexto da comunidade onde ocorreu a colaboração e o domínio do registro.

Trecho de Código 5.2: Propriedades do Registro de Colaboracao

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
]>
<rdf:RDF

  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

  <rdfs:Class rdf:ID="CollaborationRegister"/>
  <rdfs:Class rdf:ID="CommunityOfPractice"/>

  <rdf:Property rdf:ID="community_id"/>
  <rdf:Property rdf:ID="creator_id"/>
  <rdf:Property rdf:ID="content_type"/>
  <rdf:Property rdf:ID="content_id"/>
  <rdf:Property rdf:ID="creation_date" rdf:datatype="&xsd;date">
  <rdf:Property rdf:ID="register_type">
  <rdf:Property rdf:ID="cop_context">
  <rdf:Property rdf:ID="domain">

</rdf:RDF>
```

Com as propriedades devidamente definidas e mapeadas, o passo seguinte foi o mapeamento dos campos RDF dos Registros de Colaboração e Comunidades de Prática para classes Python. Para isso foi utilizado o RDFAlchemy. No Trecho de Código 5.3 foram criadas duas classes: uma para os Registros de Colaboração (*CollaborationRegister*) e outra para as Comunidades de Prática (*CommunityOfPractice*). Ambas as classes têm herança da classe *rdfSubject*, o que as torna matrizes de recursos RDF. No mapeamento, cada uma das propriedades desses recursos RDF torna-se um atributo da respectiva classe, representado como um *rdfSingle*. Para especificar os tipos das propriedades foram instanciados *namespaces* referentes aos vocabulários utilizados, do próprio *cop.semantic* e do SIOC (*Semantically-Interlinked Online Communities*)[4].

[4]http://sioc-project.org/

Trecho de Código 5.3: Mapeamento dos campos RDF para classes Python

```
 1 from rdfalchemy import rdfSubject
 2 from rdfalchemy import rdfSingle
 3 from rdflib import Namespace
 4 #...
 5 COPSEMANTIC = Namespace("http://localhost:8080/copsemantic/rdfs-copsemantic#")
 6 SIOC = Namespace("http://rdfs.org/sioc/ns#")
 7 #...
 8 class CollaborationRegister(rdfSubject):
 9     """Classe do registro de colaboracao
10     """
11     rdf_type = COPSEMANTIC.CollaborationRegister
12     community_id = rdfSingle(COPSEMANTIC.community_id, 'community_id')
13     creator_id = rdfSingle(COPSEMANTIC.creator_id, 'creator_id')
14     content_type = rdfSingle(COPSEMANTIC.content_type, 'content_type')
15     content_id = rdfSingle(COPSEMANTIC.content_id, 'content_id')
16     creation_date = rdfSingle(COPSEMANTIC.creation_date, 'creation_date')
17     register_type = rdfSingle(COPSEMANTIC.register_type, 'register_type')
18     cop_context = rdfSingle(COPSEMANTIC.cop_context, 'cop_context')
19     domain = rdfSingle(COPSEMANTIC.domain, 'domain')
20
21 class CommunityOfPractice(rdfSubject):
22     """Classe da comunidade de pratica
23     """
24     rdf_type = COPSEMANTIC.CommunityOfPractice
25     community_id = rdfSingle(COPSEMANTIC.community_id, 'community_id')
26     name = rdfSingle(SIOC.title, 'title')
27     creator_id = rdfSingle(COPSEMANTIC.creator_id, 'creator_id')
28     creation_date = rdfSingle(COPSEMANTIC.creation_date, 'creation_date')
29     domain = rdfSingle(COPSEMANTIC.domain, 'domain')
```

Após a definição das propriedades dos Registros de Colaboração, foram desenvolvidos os *scripts* para a geração de tais registros e a persistência dos mesmos na base de dados *triple store* do Sesame. A ideia central do modelo proposto é explorar uma das facetas inerentes a definição de Registros de Colaboração. Para isto, propõe-se a descrição dos mesmos através de triplas em RDF compostas dos conteúdos criados pelos usuários em uma Comunidade de Prática, bem como do registro da própria participação de um usuário em uma comunidade. No protótipo desenvolvido, a geração dos registros pode ser feita de maneira síncrona ou assíncrona.

A geração dos registros de forma síncrona é feita utilizando-se o mecanismo de *subscribers* do Plone. Este mecanismo permite que funções sejam associadas a determinados eventos. Um evento pode ser uma ação do usuário, uma mudança de estado de um objeto, a troca do papel de um membro em uma comunidade, entre outros. Sendo assim, é perfeitamente possível associar uma função que gera um Registro de Colaboração ao evento lançado pela criação de um tipo de conteúdo qualquer e à participação do usuário em uma comunidade. Esta associação é feita através de um arquivo de configuração, chamado *subscribers.zcml*, parcialmente apresentado no Trecho de Código 5.4.

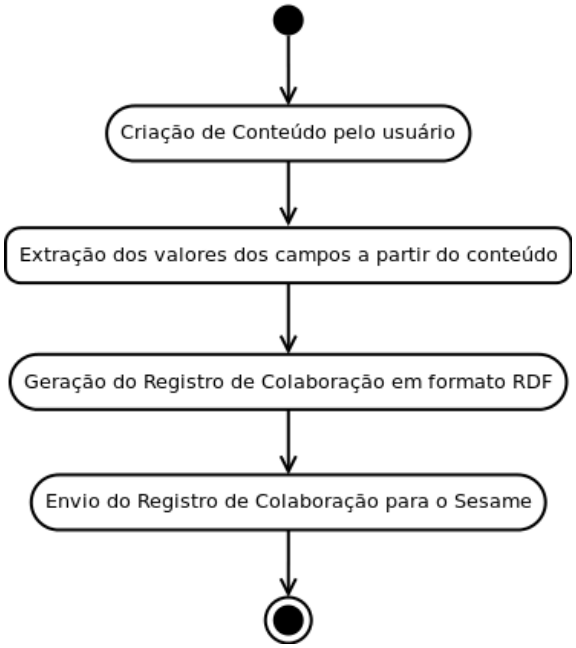
Trecho de Código 5.4: Configuração de um *subscriber* no Plone

```
1     ...
2     <subscriber
3         for="communities.practice.interfaces.ICoPDocument
4              Products.Archetypes.interfaces.IObjectInitializedEvent"
5         handler=".subscribers.insertContentCollaborationRegister"
6     />
7     ...
```

Neste exemplo é feita a associação da função *insertContentCollaborationRegister* com o evento *IObjectInitializedEvent* para o tipo *ICoPDocument*. Ou seja, cada vez que um conteúdo do tipo Página (*ICoPDocument*) for criada (*IObjectInitializedEvent*), a função especificada será executada.

Após a captura do evento é disparado o *script* que faz a geração do Registro de Colaboração. Este *script*, apresentado no Trecho de Código 5.5, extrai do conteúdo criado os valores das propriedades que serão atribuídas aos campos do registro, chama a função que gera os dados no formato RDF e persiste os mesmos na *triple store*. O diagrama de atividades ilustrado na Figura 5.3 apresenta a sequência de ações realizadas pelo *script*

Figura 5.3: Diagrama de atividades da geração síncrona



Trecho de Código 5.5: *Script* de geração síncrono

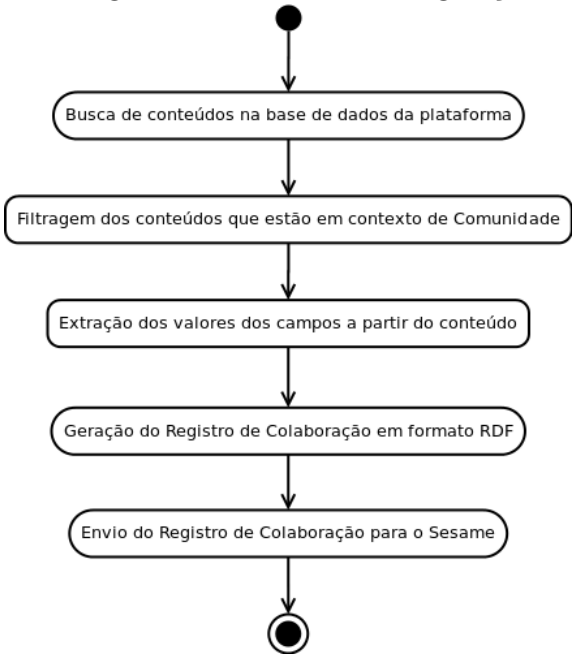
```
def insertContentCollaborationRegister(obj, event):
    """Insere o Registro de Colaboracao na criacao de conteudos
    """
    cop = getCoPContext(obj)
    #cria o registro somente se o conteudo for criado no contexto de uma CoP
    if cop:
        register_uri = obj.absolute_url()
        community_uri = cop.absolute_url()
        creator_uri = obj.Creator()
        content_type = obj.portal_type
        if content_type == "Discussion Item":
            prefix = obj.absolute_url()[:obj.absolute_url().find("++")]
            suffix = obj.absolute_url()[obj.absolute_url().rfind("/")+1:]
            content_uri = "%sview#%s" % (prefix, suffix)
        else:
            content_uri = obj.absolute_url()
        creation_date = datetime.now()
        register_type = "CreateContent"
        copmenu = getCoPMenuContext(obj)
        community_context = copmenu.titlemenu
        if content_type not in ["Discussion Item", "PloneboardComment",]:
            domain = obj.Title()
        elif content_type == "Discussion Item":
            domain = obj.text
        else:
            domain = obj.getText()
        insertCollaborationRegister(register_uri, community_uri, creator_uri,
            content_type, content_uri,
                                    creation_date, register_type,
                                        community_context, domain)

    return True
```

A geração síncrona dos registros funciona bem em um cenário onde a plataforma de Comunidades de Prática e o *cop.semantic* começam a ser utilizados simultaneamente. Neste cenário todos os Registros de Colaboração são gerados conforme o ambiente das Comunidades é configurado, fazendo com que a plataforma e o *triple store* mantenham-se sincronizados. Porém, isto não ocorre em casos onde as Comunidades já encontram-se em produção. Nesse caso, a geração síncrona dos registros só satisfaz os dados incorporados à plataforma após a adição do *cop.semantic* àquele contexto.

Buscando prover uma solução para um cenário como esse, foi desenvolvido um *script* assíncrono para a geração dos Registros de Colaboração. Este script permite que sejam gerados, a partir da base já existente da plataforma de Comunidades, todos os Registros referentes às colaborações que já ocorreram até então. Além disso, esta forma de geração é uma solução para os casos onde a criação automática dos registros se mostre muito custosa em relação ao desempenho. Nesse caso o *script* pode ser executado a cada determinado intervalo de tempo, de modo a não comprometer a performance do *framework*.

Para a geração dos Registros de Colaboração a partir de uma base de dados já existente, o *script* faz uma busca dentro desta base. Num primeiro momento, são buscadas as Comunidades de Prática, sendo gerados os registros das mesmas e dos seus participantes. Após, são buscados todos os conteúdos daquela base, e gerados os Registros de Colaboração referentes a eles. O processo de extração das propriedades a partir dos campos dos conteúdos é semelhante ao do *script* assíncrono. Como podem existir conteúdos dentro desta base que não fazem parte do contexto da plataforma de Comunidades de Prática, é feita uma filtragem, para que apenas registros de conteúdos pertinentes sejam persistidos na base de triplas. O diagrama de atividades ilustrado na Figura 5.4 apresenta a sequência de ações realizadas pelo *script*.

Figura 5.4: Diagrama de atividades da geração assíncrona



O *script* descrito é apresentado no Trecho de Código 5.6. A execução do *script* é disparada através do acesso a uma página do portal que hospeda uma comunidade, disponível em qualquer contexto, adicionando-se à URL o sufixo “import-rdf-view”.

Trecho de Código 5.6: *Script* de geração assíncrono

```
1     def import_rdf(self):
2         portal = getSite()
3         catalog = portal.portal_catalog
4         #busca das Comunidades de Pratica
5         communities = catalog(portal_type="CoP")
6         for community in communities:
7             #
8             # extracao dos valores das propriedades
9             #
10
11            #geracao do registro da comunidade
12            insertCommunityOfPractice(community_uri, name, creator_uri,
                  creation_date, domain)
13
14            #inserir local roles das comunidades no triple store
15            local_roles = cop_object.get_local_roles()
16                #
17                # extracao dos valores das propriedades
18                #
19
20                insertCollaborationRegister(register_uri, community_uri,
                      creator_uri, content_type, content_uri,
21                                            creation_date, register_type,
                                                  community_context, domain)
22
23        all_content = catalog(portal_type=["CoPDocument", "CoPImage", "CoPFile",
               "CoPLink", "CoPEvent", "CoPATA",
24                                           "CoPPortfolio", "CoPUpload", "
                                                 PloneboardForum", "
                                                 PloneboardConversation",
25                                           "PloneboardComment", "Discussion Item
                                                 "])
26        for content in all_content:
27            content_object = content.getObject()
28            community = getCoPContext(content_object)
29            if community:
30                #
31                # extracao dos valores das propriedades
32                #
33
34                insertCollaborationRegister(register_uri, community_uri,
                      creator_uri, content_type, content_uri,
35                                            creation_date, register_type,
                                                  community_context, domain)
```

Tanto a geração síncrona quanto assíncrona compartilham um último passo, que é a chamada da função que efetivamente faz a geração e persistência do Registro de Colaboração. Esta função cria um grafo, implementado pela biblioteca *rdflib*, que funciona como um banco de dados temporário para as triplas geradas. Este grafo é vinculado à classe dos Registros de Colaboração, detalhada anteriormente. Cada atributo da classe recebe o seu respectivo valor, recebido como parâmetro dos *scripts* síncrono e assíncrono, que realizaram a extração dos dados do conteúdo. Após a construção do Registro os dados são serializados em formato XML. A função então

envia uma requisição HTTP para Sesame, contendo os dados do registro, além de informações referentes ao local onde o registro deve ser persistido e o formato das informação que estão sendo enviadas. O Trecho de Código 5.7 apresenta esta função.

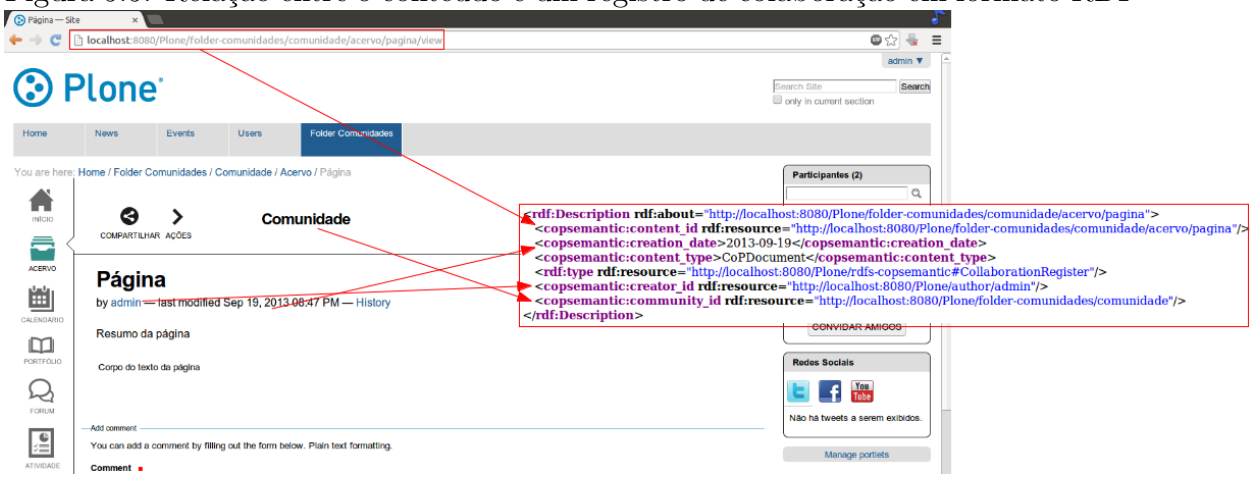
Trecho de Código 5.7: *Script* de geração e persistência do Registro

```
from rdflib import ConjunctiveGraph

def insertCollaborationRegister(register_uri, community_uri, creator_uri,
    portal_type, content_uri,
                                creation_date, register_type, community_context,
                                    domain):
    """Insere o Registro de Colaboracao
    """
    portal = getSite()
    graph = ConjunctiveGraph()
    graph.bind("foaf", FOAF)
    graph.bind("xsd", XSD)
    graph.bind("copsemantic", COPSEMANTIC)
    graph.bind("copontology", COPONTOLOGY)

    CollaborationRegister.db = graph
    collaboration_register = CollaborationRegister(URIRef(register_uri))
    collaboration_register.community_id = URIRef(community_uri)
    collaboration_register.creator_id = URIRef("%s/author/%s" % (portal.
        absolute_url(), creator_uri))
    if portal_type:
        collaboration_register.content_type = Literal(portal_type)
    if content_uri:
        collaboration_register.content_id = URIRef(content_uri)
    collaboration_register.creation_date = Literal(creation_date.strftime("%Y-%m
        -%d"))
    collaboration_register.register_type = Literal(register_type)
    if community_context:
        collaboration_register.cop_context = Literal(community_context)
    if domain:
        collaboration_register.domain = Literal(domain)

    context = portal.absolute_url()
    params = {}
    params["context"] = "<%s>" % context
    endpoint = "%s/%s/statements?%s" % (TRIPLESTORE_URL, REPOSITORY, urllib.
        urlencode(params))
    data = graph.serialize(format="xml")
    dict_headers = {}
    dict_headers["content-type"] = "application/rdf+xml"
    (response, content) = httplib2.Http().request(endpoint, "POST", body=data,
        headers=dict_headers)

    return True
```

Conforme definido anteriormente, serão gerados Registros de Colaboração referentes a criação de conteúdos em Comunidades e a participação de usuários em Comunidades. A Figura 5.5 apresenta a relação entre a criação de qualquer tipo de conteúdo e a geração de um registro. Este registro é referente à criação de um conteúdo do tipo "Página" dentro de uma comunidade "Comunidade" por um membro "admin". A URL do conteúdo é armazenada na propriedade *content_id*; sua data de criação é armazenada na propriedade *creation_date*; a propriedade *content_type* refere-se ao tipo de conteúdo, neste caso *CoPDocument* (página); a

URL referente ao criador no portal é armazenada na propriedade *creator_id*; e a URL da comunidade onde foi criado o conteúdo é armazenada na propriedade *community_id*.

Figura 5.5: Relação entre o conteúdo e um registro de colaboração em formato RDF



5.3 Cenários de Uso

Para a realização dos testes do protótipo foi utilizada uma base de dados real, de um portal com 12 Comunidades de Prática abordando diferentes temas. Este portal possui mais de 13.000 participantes, e aproximadamente 6.600 conteúdos dos mais diversos tipos criados em sua base de dados.

Dentre todas as ferramentas de interação oferecidas pela plataforma do *Framework CoP* a mais utilizada pelos participantes deste portal são os comentários. Este tipo de colaboração pode ser feita tanto nos fóruns de discussão quanto nos próprios conteúdos de outros tipos.

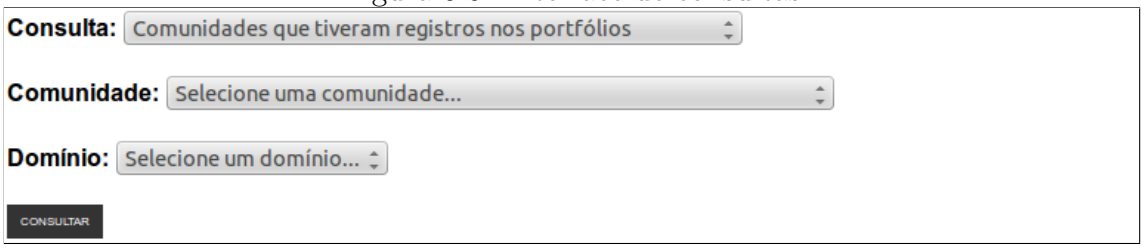
É neste cenário que se desenvolveram experimentos sobre um conjunto de consultas SPARQL. O objetivo destas consultas é a validação da estrutura proposta dos Registros de Colaboração, através da busca de determinados padrões gerados pelo protótipo.

Foram definidos quatro cenários principais para testes, abordando diferentes aspectos referentes aos conteúdos, comunidade e participantes, levando em conta o tema, assunto ou área do conhecimento envolvido. Com isto podemos descobrir possíveis intersecções entre as comunidades que estão desenvolvendo mais o domínio a que se proporam. Dessa forma, futuros algoritmos de recomendação podem direcionar usuários que comentam assuntos cujo domínio pertence a outras comunidades as quais eles não pertencem.

As consultas especificadas podem ser executadas diretamente através da

interface de gerenciamento do Sesame, ou por meio de uma interface desenvolvida no *cop.semantic*. Esta interface permite que a consulta desejada seja escolhida e que determinados filtros sejam informados. A Figura 5.6 ilustra esta interface.

Figura 5.6: Interface de consultas



5.3.1 Comunidades que compartilham o mesmo domínio

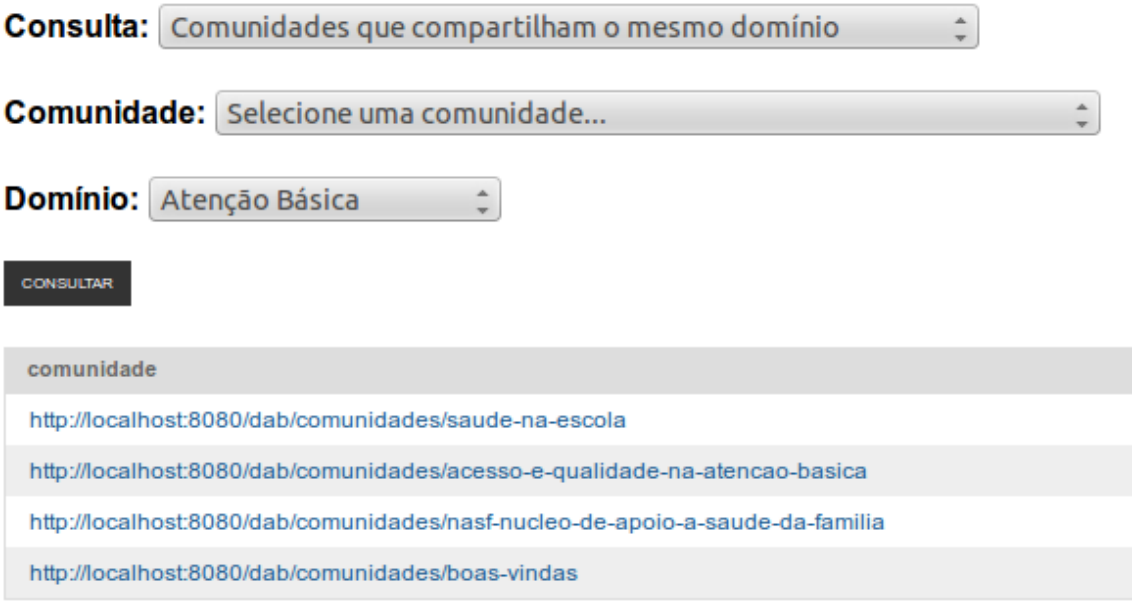
Esta consulta visa mostrar as Comunidades de Prática que possuem o mesmo domínio. Ou seja, através dela é possível identificar comunidades que compartilham o mesmo tema, assunto ou área do conhecimento. O Trecho de Código 5.8 apresenta esta consulta.

Trecho de Código 5.8: Consulta de comunidades que compartilham o mesmo domínio

```
1 PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX copsemantic:<http://localhost:8080/copsemantic/rdfs-copsemantic#>
3
4 SELECT DISTINCT ?comunidade
5 WHERE {
6
7 ?comunidade rdf:type copsemantic:CommunityOfPractice .
8 ?comunidade copsemantic:domain <http://localhost:8080/copsemantic/owl-
      copontology/COPOntology#atencaobasica> .
9
10 }
```

A consulta apresentada faz uma busca das Comunidades de Prática (*copsemantic:CommunityOfPractice*) que possuem um mesmo domínio (*copsemantic:domain*). Neste exemplo o domínio pesquisado refere-se a "atenção básica" (<http://localhost:8080/copsemantic/owl-copontology/COPOntology#atencaobasica>). Os resultados desta consulta são ilustradados na Figura 5.7.

Figura 5.7: Resultados da consulta de comunidades que compartilham o mesmo domínio



A partir desta consulta, podem ser extraídos conjuntos de triplas como o apresentado no Trecho de Código 5.9. Tal conjunto é a representação completa do primeiro registro da tabela de resultados da Figura 5.7.

Trecho de Código 5.9: Exemplo de Registro da Consulta de comunidades com o mesmo domínio

```
1 <rdf:Description rdf:about="http://localhost:8080/dab/comunidades/saude-na-
      escola">
2     <rdf:type rdf:resource="http://localhost:8080/copsemantic/rdfs-copsemantic#
          CommunityOfPractice"/>
3     <copsemantic:community_id rdf:resource="http://localhost:8080/dab/
          comunidades/saude-na-escola"/>
4     <sioc:title>Saude na Escola</sioc:title>
5     <copsemantic:creator_id rdf:resource="http://localhost:8080/dab/author/
          paulonm@atencabasica.org.br"/>
6     <copsemantic:creation_date>2013-10-29</copsemantic:creation_date>
7     <copsemantic:domain rdf:resource="http://localhost:8080/copsemantic/owl-
          copontology/COPOntology#atencaobasica"/>
8 </rdf:Description>
```

5.3.2 Usuários que fazem comentários sobre um mesmo domínio

Esta consulta visa mostrar quais usuários colaboram através de comentários em Comunidades de Prática que compartilham um mesmo domínio. Estes comentários podem ter sido feitos tanto em conteúdos como em fóruns de discussão. O Trecho de Código 5.10 mostra esta consulta.

Trecho de Código 5.10: Consulta de usuários que fazem comentários sobre um mesmo domínio

```
1 PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX copsemantic:<http://localhost:8080/copsemantic/rdfs-copsemantic#>
3
4 SELECT DISTINCT ?usuario ?comunidade
5 WHERE {
6
7 ?comunidade rdf:type copsemantic:CommunityOfPractice .
8 ?comunidade copsemantic:domain <http://localhost:8080/copsemantic/owl-
      copontology/COPOntology#atencaobasica> .
9 ?comentario copsemantic:community_id ?comunidade .
10 ?comentario copsemantic:creator_id ?usuario .
11 ?comentario copsemantic:content_type ?conteudo .
12 FILTER(?conteudo = "Discussion Item" || ?conteudo = "PloneboardComment") .
13
14 }
15 ORDER BY ?usuario
```

A consulta apresentada faz uma busca dos criadores (*copsemantic:creator_id*) de conteúdos dos tipos (*copsemantic:content_type*) *"Discussion Item"* e *"PloneboardComment"* (comentários) em todas as Comunidades de Prática (*rdf:type copsemantic:CommunityOfPractice*) de um determinado domínio (*copsemantic:domain*), que neste exemplo é "atenção básica". Parte dos resultados desta consulta é ilustrada na Figura 5.8.

Figura 5.8: Resultados da consulta de usuários que comentam sobre um mesmo domínio

comunidade	usuario
http://localhost:8080/dab/comunidades/acesso-e-qualidade-na-atencao-basica	http://localhost:8080/dab/author/Rodrigofelberg@gmail.com
http://localhost:8080/dab/comunidades/saude-na-escola	http://localhost:8080/dab/author/TATY_IZ@YAHOO.COM.BR
http://localhost:8080/dab/comunidades/acesso-e-qualidade-na-atencao-basica	http://localhost:8080/dab/author/acaciosms@gmail.com
http://localhost:8080/dab/comunidades/acesso-e-qualidade-na-atencao-basica	http://localhost:8080/dab/author/acarolns@hotmail.com
http://localhost:8080/dab/comunidades/acesso-e-qualidade-na-atencao-basica	http://localhost:8080/dab/author/addenilda@hotmail.com
http://localhost:8080/dab/comunidades/acesso-e-qualidade-na-atencao-basica	http://localhost:8080/dab/author/adeiltonmendonca@hotmail.com
http://localhost:8080/dab/comunidades/acesso-e-qualidade-na-atencao-basica	http://localhost:8080/dab/author/adriana_fw@hotmail.com
http://localhost:8080/dab/comunidades/nasf-nucleo-de-apoio-a-saude-da-familia	http://localhost:8080/dab/author/adrianawalherifi@yahoo.com.br
http://localhost:8080/dab/comunidades/acesso-e-qualidade-na-atencao-basica	http://localhost:8080/dab/author/adrianoviskpinheiro@hotmail.com
http://localhost:8080/dab/comunidades/saude-na-escola	http://localhost:8080/dab/author/adrifeso@bol.com.br
http://localhost:8080/dab/comunidades/acesso-e-qualidade-na-atencao-basica	http://localhost:8080/dab/author/adrifeso@bol.com.br
http://localhost:8080/dab/comunidades/acesso-e-qualidade-na-atencao-basica	http://localhost:8080/dab/author/ageu1987@hotmail.com

5.3.3 Usuários que participam de uma mesma comunidade

Esta consulta visa mostrar quais usuários participam de uma determinada Comunidade de Prática. O Trecho de Código 5.11 mostra esta consulta.

Trecho de Código 5.11: Consulta de usuários que participam da mesma comunidade

```
 1 PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
 2 PREFIX copsemantic:<http://localhost:8080/copsemantic/rdfs-copsemantic#>
 3
 4 SELECT DISTINCT ?membro
 5 WHERE {
 6
 7 ?participante copsemantic:community_id <http://localhost:8080/dab/comunidades/e-
       sus> .
 8 ?participante copsemantic:register_type "Participate" .
 9 ?participante copsemantic:creator_id ?membro .
10
11 }
```

A consulta apresentada faz uma busca dos membros (*copsemantic:creator_id*) que participam (*copsemantic:register_type* “Participate”) de uma determinada comunidade (*copsemantic:community_id*). No exemplo, a comunidade do “e-SUS” (<http://localhost:8080/dab/comunidades/e-sus>) . Parte dos resultados desta consulta é ilustrada na Figura 5.9.

Figura 5.9: Resultados da consulta de usuários que participam de uma mesma comunidade

5.3.4 Comunidades que tiveram registros nos portfólios

Esta consulta visa mostrar em quais comunidades ocorreram colaborações que tiveram como contexto o portfólio, dentro das Comunidades de Prática. O Portfólio é usado para a produção individual dos membros e pode ser uma fonte rica de informações sobre interesses e especialidades de um indivíduo. O Trecho de Código 5.12 mostra esta consulta.

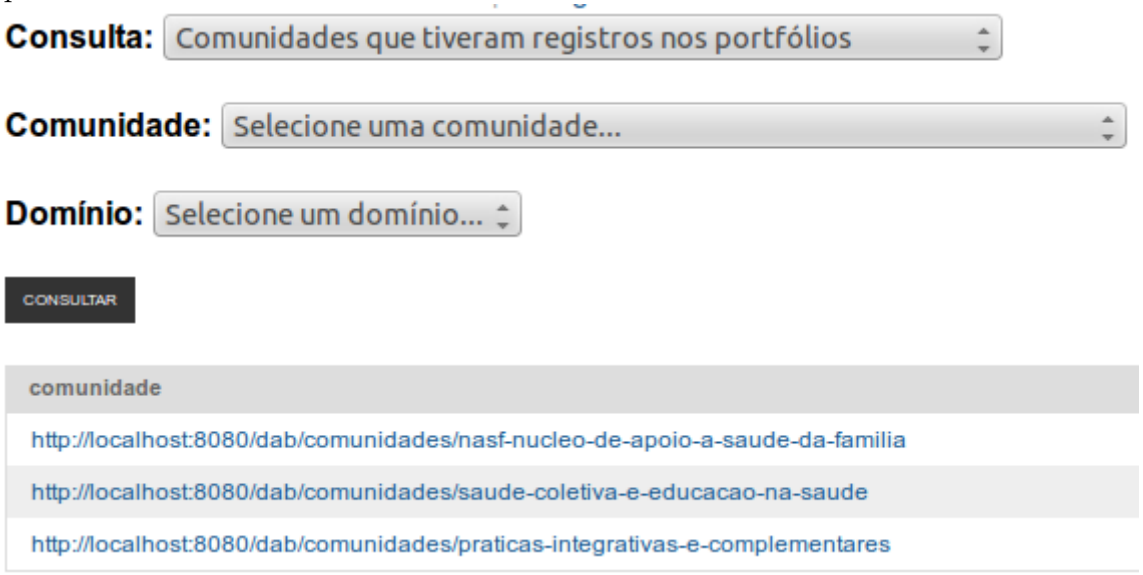
Trecho de Código 5.12: Consulta comunidades com registros nos portfólios

```
1 PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX copsemantic:<http://localhost:8080/copsemantic/rdfs-copsemantic#>
3
4 SELECT DISTINCT ?comunidade
5 WHERE {
6
7 ?registro copsemantic:cop_context "portfolio" .
8 ?registro copsemantic:content_type ?conteudo .
9 FILTER(?conteudo != "CoPPortfolio") .
```

```
10 ?registro copsemantic:community_id ?comunidade .
11
12 }
```

A consulta apresentada faz uma busca das comunidades (*copsemantic:community_id*) que tiveram conteúdos criados no contexto (*copsemantic:cop_context*) de "portfolio". Além disso, os tipos de conteúdo (*copsemantic:content_type*) são filtrados, para que não sejam trazidos os portfólios vazios (tipo "CoPPortfolio"). Parte dos resultados desta consulta é ilustrada na Figura 5.10.

Figura 5.10: Resultados da consulta de comunidades que tiveram registros nos portfólios



Um dos conjuntos de triplas que pode ser obtido a partir desta consulta é apresentado no Trecho de Código 5.13, na sintaxe RDF/XML. Tal conjunto é a representação completa do primeiro registro da tabela de resultados da Figura 5.10.

Trecho de Código 5.13: Exemplo de RDF de Comunidade

```
1 <rdf:Description rdf:about="http://localhost:8080/dab/comunidades/nasf-nucleo-de
      -apoio-a-saude-da-familia">
2     <rdf:type rdf:resource="http://localhost:8080/copsemantic/rdfs-copsemantic#
          CommunityOfPractice"/>
3     <copsemantic:community_id rdf:resource="http://localhost:8080/dab/
          comunidades/nasf-nucleo-de-apoio-a-saude-da-familia"/>
4     <sioc:title>NASF - Nucleo de Apoio a Saude da Familia</sioc:title>
5     <copsemantic:creator_id rdf:resource="http://localhost:8080/dab/author/
          karensathie@atencaobasica.org.br"/>
6     <copsemantic:creation_date>2013-10-29</copsemantic:creation_date>
7     <copsemantic:domain rdf:resource="http://localhost:8080/copsemantic/owl-
          copontology/COPOntology#atencaobasica"/>
8 </rdf:Description>
```

5.4 Interoperabilidade Semântica

Um dos vários objetivos da Web Semântica é promover a uniformidade e interoperabilidade com outras plataformas na Web. A propriedade *copsemantic:domain* permite que seja associada uma ontologia que descreva o domínio relacionado ao registro de colaboração, da mesma forma que o SIOC pode ser associado garantindo interoperabilidade de dados com outros portais. As propriedades e classes que descrevem os registros de colaboração, preferencialmente, devem estar presentes em uma ontologia, descrita especificamente, ou, estendendo outras ontologias de domínio existentes

Para os experimentos, foi escrita uma ontologia onde estão definidos alguns indivíduos relacionados a áreas da saúde, tema principal do portal utilizado nos testes. Nesta construção foi utilizada a linguagem OWL, que dá um passo além do RDFS na questão da representação do conhecimento e expressividade semântica. Esta ontologia é apresentada no Trecho de Código 5.14.

Trecho de Código 5.14: Ontologia utilizada como referência na base de testes

```
 1 <?xml version="1.0"?>
 2 <!DOCTYPE rdf:RDF [
 3     <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
 4     <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
 5     <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
 6     <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
 7 ]>
 8
 9 <rdf:RDF xmlns="http://localhost:8080/copsemantic/owl-copontology#"
10      xml:base="http://localhost:8080/copsemantic/owl-copontology"
11      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
12      xmlns:owl="http://www.w3.org/2002/07/owl#"
13      xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
14      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
15
16     <owl:Ontology rdf:about="http://localhost:8080/copsemantic/owl-copontology"
           />
```

```
17
18      <owl:Class rdf:about="http://localhost:8080/copsemantic/owl-copontology/
            COPOntology#Educacao_Na_Saude">
19          <rdfs:subClassOf rdf:resource="http://localhost:8080/copsemantic/owl-
                copontology/COPOntology#AreaMedicina"/>
20      </owl:Class>
21      <owl:Class rdf:about="http://localhost:8080/copsemantic/owl-copontology/
            COPOntology#Doencas_Cronicas">
22          <rdfs:subClassOf rdf:resource="http://localhost:8080/copsemantic/owl-
                copontology/COPOntology#AreaMedicina"/>
23      </owl:Class>
24      <owl:Class rdf:about="http://localhost:8080/copsemantic/owl-copontology/
            COPOntology#Coordenacao_Na_Saude">
25          <rdfs:subClassOf rdf:resource="http://localhost:8080/copsemantic/owl-
                copontology/COPOntology#AreaMedicina"/>
26      </owl:Class>
27      <owl:Class rdf:about="http://localhost:8080/copsemantic/owl-copontology/
            COPOntology#TI_Na_Saude">
28          <rdfs:subClassOf rdf:resource="http://localhost:8080/copsemantic/owl-
                copontology/COPOntology#AreaMedicina"/>
29      </owl:Class>
30      <owl:Class rdf:about="http://localhost:8080/copsemantic/owl-copontology/
            COPOntology#Atencao_Psicossocial">
31          <rdfs:subClassOf rdf:resource="http://localhost:8080/copsemantic/owl-
                copontology/COPOntology#AreaMedicina"/>
32      </owl:Class>
33      <owl:Class rdf:about="http://localhost:8080/copsemantic/owl-copontology/
            COPOntology#Praticas_Integrativas">
34          <rdfs:subClassOf rdf:resource="http://localhost:8080/copsemantic/owl-
                copontology/COPOntology#AreaMedicina"/>
35      </owl:Class>
36      <owl:Class rdf:about="http://localhost:8080/copsemantic/owl-copontology/
            COPOntology#Atencao_Basica">
37          <rdfs:subClassOf rdf:resource="http://localhost:8080/copsemantic/owl-
                copontology/COPOntology#AreaMedicina"/>
38      </owl:Class>
39      <owl:Class rdf:about="http://localhost:8080/copsemantic/owl-copontology/
            COPOntology#Saude_Mental">
40          <rdfs:subClassOf rdf:resource="http://localhost:8080/copsemantic/owl-
                copontology/COPOntology#AreaMedicina"/>
41      </owl:Class>
42      <owl:Class rdf:about="http://localhost:8080/copsemantic/owl-copontology/
            COPOntology#Doencas_Cronicas">
43          <rdfs:subClassOf rdf:resource="http://localhost:8080/copsemantic/owl-
                copontology/COPOntology#AreaMedicina"/>
44      </owl:Class>
45
46      <owl:NamedIndividual rdf:about="http://localhost:8080/copsemantic/owl-
            copontology/COPOntology#educacaonasaude">
47          <rdf:type rdf:resource="http://localhost:8080/copsemantic/owl-
                copontology/COPOntology#Educacao_Na_Saude"/>
48      </owl:NamedIndividual>
49      <owl:NamedIndividual rdf:about="http://localhost:8080/copsemantic/owl-
            copontology/COPOntology#doencascronicas">
50          <rdf:type rdf:resource="http://localhost:8080/copsemantic/owl-
                copontology/COPOntology#Doencas_Cronicas"/>
51      </owl:NamedIndividual>
52      <owl:NamedIndividual rdf:about="http://localhost:8080/copsemantic/owl-
            copontology/COPOntology#coordenacaonasaude">
```

```
53          <rdf:type rdf:resource="http://localhost:8080/copsemantic/owl-
                copontology/COPOntology#Coordenacao_Na_Saude"/>
54      </owl:NamedIndividual>
55      <owl:NamedIndividual rdf:about="http://localhost:8080/copsemantic/owl-
            copontology/COPOntology#tinasaude">
56          <rdf:type rdf:resource="http://localhost:8080/copsemantic/owl-
                copontology/COPOntology#TI_Na_Saude"/>
57      </owl:NamedIndividual>
58      <owl:NamedIndividual rdf:about="http://localhost:8080/copsemantic/owl-
            copontology/COPOntology#atencaopsicossocial">
59          <rdf:type rdf:resource="http://localhost:8080/copsemantic/owl-
                copontology/COPOntology#Atencao_Psicossocial"/>
60      </owl:NamedIndividual>
61      <owl:NamedIndividual rdf:about="http://localhost:8080/copsemantic/owl-
            copontology/COPOntology#praticasintegrativas">
62          <rdf:type rdf:resource="http://localhost:8080/copsemantic/owl-
                copontology/COPOntology#Praticas_Integrativas"/>
63      </owl:NamedIndividual>
64      <owl:NamedIndividual rdf:about="http://localhost:8080/copsemantic/owl-
            copontology/COPOntology#atencaobasica">
65          <rdf:type rdf:resource="http://localhost:8080/copsemantic/owl-
                copontology/COPOntology#Atencao_Basica"/>
66      </owl:NamedIndividual>
67      <owl:NamedIndividual rdf:about="http://localhost:8080/copsemantic/owl-
            copontology/COPOntology#saudemental">
68          <rdf:type rdf:resource="http://localhost:8080/copsemantic/owl-
                copontology/COPOntology#Saude_Mental"/>
69      </owl:NamedIndividual>
70
71  </rdf:RDF>
```

Apesar de uma nova ontologia ter sido escrita para os testes do protótipo, nada impede que ontologias externas já existentes possam ser referenciadas pela propriedade *copsemantic:domain*. Esta característica da propriedade permite a integração da solução desenvolvida com outros sistemas já existentes.

5.5 Conclusões do Capítulo

Ao longo deste Capítulo foi apresentada a solução desenvolvida para a resolução do problema proposto para este trabalho. Foram descritas as suas principais características, desde a arquitetura do protótipo criado, até os cenários de uso utilizados para testes e validação do mesmo através de consultas semânticas à base de dados de triplas.

As Seções 5.1 e 5.2 abordaram questões referentes ao projeto e desenvolvimento do produto *cop.semantic*. Após a apresentação da arquitetura que compõe a solução, foram mostrados o processo de mapeamento dos campos do Registro de Colaboração para classes Python, a captura de eventos lançados pelo *Framework CoP* para a geração síncrona dos registros, a consulta à base da plataforma de Comunidades de Prática para a geração assíncrona de registros, a extração dos valores das propriedades dos registros a partir dos objetos criados e a geração e

persistência dos registros no *triple store* Sesame.

Após o detalhamento do desenvolvimento da solução, foram apresentados os cenários de uso do protótipo. Nestes cenários foram executadas consultas sobre as triplas geradas pelo *cop.semantic*. A partir destas, foi possível observar a recuperação de diversos tipos de informações, como comunidades com o mesmo assunto, participantes que colaboram sobre os mesmos domínios, entre outros. Além disso foram mostrados alguns dos conjuntos de triplas que podem ser acessados através deste tipo de consulta.

Por fim foi apresentada em maiores detalhes a propriedade *copsemantic:domain*. É nesta propriedade que se encontra a maior possibilidade de integração da plataforma com outras aplicações, uma vez que ela permite que sejam referenciados individuos de ontologias externas, enriquecendo a gama de resultados que podem ser obtidos a partir de futuras consultas.

6 CONSIDERAÇÕES FINAIS

Ao longo deste trabalho foram abordadas questões relativas à Web Semântica, a bases de dados *triple store* e ao *Framework CoP*. Estes estudos culminaram no projeto e desenvolvimento de uma solução para a implementação de Registros de Colaboração do referido *framework*.

No Capítulo 2 tratou-se da Web Semântica, a Web dos significados. Seus conceitos são estruturados por meio de diversas tecnologias e padrões, como o XML, o RDF, o RDFS e a OWL, através dos quais atribui-se estrutura e sentido aos dados disponíveis na Web.

No Capítulo 3 foram abordados conceitos, *frameworks* e trabalhos relacionados a bancos de dados de triplas, os *triple stores*. Tais bancos oferecem um meio de persistência compatível ao modelo dados RDF, padrão da Web Semântica.

No Capítulo 4 foi exposto o *Framework CoP*, uma plataforma para o desenvolvimento de Comunidades de Prática virtuais. Esta ferramenta oferece diversas funcionalidades para o fomento da aprendizagem coletiva e colaborativa.

Por fim, no Capítulo 5, foi descrita a solução para o problema do trabalho. Esta solução teve como objetivo adicionar mais semântica às Comunidades de Prática do *framework*. Nesta mesma seção foram apresentados cenários de uso do protótipo desenvolvido. Como resultado deste desenvolvimento obteve-se resultados mais ricos em significado para as consultas realizadas dentro do *Framework CoP*.

6.1 Limitações e Dificuldades

Devido ao fato de o *Framework CoP* ainda não ter sido completamente implementado, nem todas as interações ocorridas dentro da Plataforma de Comunidades puderam ser mapeadas. Nesse ponto encontra-se uma das limitações do protótipo, uma vez que apenas dois tipos de colaboração são mapeadas, a criação de conteúdos e a participação em comunidades.

Algumas dificuldades foram sentidas durante a realização deste trabalho. Foram muito poucos os trabalhos correlatos encontrados para a determinação de

um ponto de partida para a solução. A quase inexistência de ferramentas nativas para o desenvolvimento de aplicações semânticas com *triple store* sobre a pilha Python/Zope/Plone também se tornou um obstáculo a ser superado na implementação.

6.2 Contribuições

Este trabalho propôs o desenvolvimento da classe Registros de Colaboração do *Framework CoP* seguindo seu modelo ontológico conceitual, conforme apresentado no Capítulo 4. Ele representa um primeiro passo em direção ao desenvolvimento dos outros componentes do *framework* respeitando este modelo.

A concretização deste componente, seguindo as bases teóricas referenciadas ao longo do trabalho, representa também um avanço significativo no sentido da integração do *framework* com outras aplicações. A plataforma poderá então estar em conformidade Web Semântica, o que abrirá um novo leque de possibilidades referentes ao compartilhamento de informações entre o *framework* e outras plataformas que seguem modelos semelhantes.

O desenvolvimento deste protótipo traz, além das já citadas contribuições à própria Plataforma de Comunidades de Prática, uma colaboração com a comunidade de desenvolvimento Python/Zope/Plone. Por se tratar de um assunto pouco abordado dentro desta comunidade, este trabalho torna-se uma rica fonte de consultas e assunto para discussões a respeito de tecnologias, metodologias e ferramentas a serem utilizadas em outros produtos semelhantes.

6.3 Trabalhos Futuros

Este trabalho constitui-se do desenvolvimento de uma pequena parte de um grande conjunto que é o *Framework CoP*. A partir deste ponto surge como um caminho natural a subsequente adequação dos outros componentes do *framework* ao modelo da Web Semântica. Além disso, a expansão daquilo que já foi desenvolvido através do mapeamento de novos Registros de Colaboração também é fundamental.

A iminente integração do *Framework CoP* no mundo da Web Semântica faz emergir uma necessidade que ainda é pouco atendida pelo protótipo já desenvolvido: a utilização de ontologias externas já existentes dentro dos Registros de Colaboração. Conforme apresentado na Seção 5.4, a propriedade *copsemantic:domain* já prevê este cenário. Propõe-se, dessa forma, como trabalho futuro, a utilização de propriedades semelhantes para a referência de outras características dos registros. A concretização deste passo aumentará significativamente a integração semântica do *framework* com o contexto na qual ele está inserido.

REFERÊNCIAS

4STORE. **About 4store**. Disponível em: <http://4store.org/about>. Acessado em 18/06/2013.

AASMAN, J. Will Triple Stores Replace Relational Databases? **Information Management**, [S.l.], 2011. Disponível em: <http://www.information-management.com/newsletters/database_metadata_unstructured_data_triple_store-10020158-1.html>. Acessado em 20/05/2013.

ANTONIOU, G.; HARMELEN, F. van. **A Semantic Web Primer**. 2.ed. Cambridge, Massachusetts: The MIT Press, 2008. ISBN 978-0-262-01242-3.

APWEILER, R.; BAIROCH, A.; WU, C. H. Protein sequence databases. **Current Opinion in Chemical Biology**, [S.l.], v.8, n.1, p.76–80, 2004. Disponível em: <http://www.sciencedirect.com/science/article/pii/S136759310300173X>. Acessado em 24/06/2013.

ASPELI, M. **Professional Plone 4 Development**. 1.ed. Birmingham: Packt Publishing, 2011. ISBN: 978-1-849514-42-2.

BANDURA, A. **Social Learning Theory**. [S.l.]: Prentice Hall, 1977. (Prentice-Hall series in social learning theory). ISBN 0138167516.

BERNERS-LEE, T. Linked Data - Design Issues. **W3C**, [S.l.], 2006. Disponível em: <http://www.w3.org/DesignIssues/LinkedData.html>. Acessado em 23/04/2013.

BERNERS-LEE, T.; FIELDING, R.; MASINTER, L. **Uniform Resource Identifier (URI)**: generic syntax. [S.l.]: IETF, 2005. Disponível em: <http://www.ietf.org/rfc/rfc3986.txt>. Acessado em 18/03/2013.

BERNERS-LEE, T.; HALL, W.; SHADBOLT, N. The Semantic Web Revisited. **IEEE Inteligent Systems**, [S.l.], May/June 2006.

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. The Semantic Web. **Scientific American Magazine**, [S.l.], v.284, n.5, p.28–37, May 2001.

BHATIA, N. **Comparison of Triple Stores**. [S.l.]: Department of Computer Science. Stanford University, 2009. Disponível em: <http://www.bioontology.org/wiki/images/6/6a/Triple_Stores.pdf>. Acessado em 03/06/2013.

BHATIA, N. **RDF Triple Stores**. [S.l.]: Department of Computer Science. Stanford University, 2013. Disponível em: <http://www.bioontology.org/wiki/images/6/6f/RDFTripleStore.ppt>. Acessado em 22/05/2013.

BIZER, C. et al. DBpedia – A Crystallization Point for the Web of Data. **Journal of Web Semantics: Science, Services and Agents on the World Wide Web**, [S.l.], v.7, p.154–165, 2009. Disponível em: <http://intranet.inria.fr/international/arima/015/pdf/Vol.15.pp.11-35.pdf>. Acessado em 03/06/2013.

BIZER, C.; HEATH, T.; BERNERS-LEE, T. **Linked Data - The Story So Far**. [S.l.: s.n.], 2009.

BORGES, L. E. **Python para Desenvolvedores**. 2.ed. Rio de Janeiro: Edição do Autor, 2010.

BRICKLEY, D.; GUHA, R. V. **RDF Vocabulary Description Language 1.0 - RDF Schema**. [S.l.]: W3C, 2004. Disponível em: <http://www.w3.org/TR/2004/REC-rdf-schema-20040210/>. Acessado em 03/05/2013.

CARBONELL, J. G. **AI in CAI**: an artificial intelligence approach to computer assisted instruction. [S.l.]: IEEE, 1970. 190–202p. v.11, n.4. IEEE Transactions on Man Machine Systems.

DEAN, M.; SCHREIBER, G. **OWL Web Ontology Language Reference**. [S.l.]: W3C, 2004. Disponível em: <http://www.w3.org/TR/2004/REC-owl-ref-20040210/>. Acessado em 04/05/2013.

DELMONTE, M. et al. **The Definitive Guide to Plone**. 2.ed. [S.l.]: Apress, 2009. ISBN: 978-1-4302-1894-4.

FAYE, D. C.; CURE, O.; BLIN, G. A survey of RDF storage approaches. **ARIMA Journal**, [S.l.], v.15, p.11–35, 2012. Disponível em: <http://intranet.inria.fr/international/arima/015/pdf/Vol.15.pp.11-35.pdf>. Acessado em 03/06/2013.

FIORIO, M.; SILVA, J. L. T. da; RIBEIRO, A. M. Um framework de comunidades de prática em ambientes virtuais de aprendizagem. **Revista Novas Tecnologias na Educação**, [S.l.], 2011.

GRUBER, T. R. Toward principles for the design of ontologies used for knowledge sharing. **International Journals of Human-Computer Studies**, [S.l.], v.43, p.907–928, November 1995. Issues 5-6.

HARRIS, S.; SEABORNE, A. **SPARQL 1.1 Query Language**. [S.l.]: W3C, 2013. Disponível em: <http://www.w3.org/TR/2013/REC-sparql11-query-20130321/>. Acessado em 25/05/2013.

HART, J. **Social Learning Handbook**. [S.l.]: Lulu.com, 2011. ISBN 1843101866.

HEALY, A. **Communities of Practice as a Support Function for Social Learning in Distance Learning Programs**. [S.l.: s.n.], 2009. 49-56p. In M.D. Lytras et al. (Eds.): WSKS 2009, CCIS 49.

HOUAISS, A. **Dicionário Houaiss da Língua Portuguesa**. 1.ed. Rio de Janeiro: Objetiva, 2001. ISBN 85-7302-383-X.

JENA. **Apache Jena**. Disponível em: <http://jena.apache.org/index.html>. Acessado em 17/06/2013.

KLYNE, G.; CARROLL, J. J. **Resource Description Framework (RDF)**: concepts and abstract syntax. [S.l.]: W3C, 2004. Disponível em: <http://www.w3.org/TR/2004/REC-rdf-concepts-20040210>. Acessado em 01/05/2013.

LUTZ, M. **Learning Python**. 4.ed. 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly Media, 2009.

LUTZ, M. **Programming Python**. 4.ed. 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly Media, 2011.

MAGRANE, M.; CONSORTIUM, U. UniProt Knowledgebase: a hub of integrated protein data. **Database**, [S.l.], v.2011, 2011.

MANOLA, F.; MILLER, E. **RDF Primer**. [S.l.]: W3C, 2004. Disponível em: <http://www.w3.org/TR/2004/REC-rdf-primer-20040210/>. Acessado em 01/05/2013.

MCGUINNESS, D. L.; HARMELEN, F. van. **OWL Web Ontology Language Overview**. [S.l.]: W3C, 2004. Disponível em:

<http://www.w3.org/TR/2004/REC-owl-features-20040210/>. Acessado em 04/05/2013.

NOY, N. F. et al. BioPortal: ontologies and integrated data resources at the click of a mouse. **Nucleic Acids Research**, [S.l.], v.37, p.170 – 173, 2009. Disponível em: <http://nar.oxfordjournals.org/content/37/suppl_2/W170.full.pdf+html>. Acessado em 24/06/2013.

OPENRDF. **openRDF.org...home of Sesame**. Disponível em: <http://www.openrdf.org/about.jsp>. Acessado em 16/06/2013.

PEREZ, J.; ARENAS, M.; GUTIERREZ, C. **Semantics and Complexity of SPARQL**. [S.l.: s.n.], 2006. Disponível em: <http://arxiv.org/abs/cs/0605124v1.pdf>. Acessado em 21/03/2013.

PINHATI, F. A. et al. **Semantic Recommendation of Bibliographical References**: an experience with master science's degree students. [S.l.: s.n.], 2012. Disponível em: <http://www.ewh.ieee.org/soc/e/sac/itee/index.php/meem/article/viewFile/229/236>. Acessado em 22/03/2013.

PRIMO, T. T. et al. Towards Ontological Profiles in Communities of Practice. **IEEE Technology and Engineering Education (ITEE)**, [S.l.], v.7, n.3, p.13–22, 2012. Disponível em: <http://www.ewh.ieee.org/soc/e/sac/itee/index.php/meem/article/viewFile/223/237>. Acessado em 24/03/2013.

RIBEIRO, A. M. et al. Dos Ambientes de Aprendizagem às Comunidades de Prática. **22º Simpósio Brasileiro de Informática na Educação**, Aracaju, p.690–699, 2011.

SALVADORES, M. et al. Using SPARQL to Query BioPortal Ontologies and Metadata. In: **The Semantic Web – ISWC 2012**. [S.l.]: Springer Berlin Heidelberg, 2012. p.180–195. (Lecture Notes in Computer Science, v.7650). Disponível em: <http://iswc2012.semanticweb.org/sites/default/files/76500177.pdf>. Acessado em 25/06/2013.

SAUERMANN, L.; CYGANIAK, R. **Cool URIs for the Semantic Web**. [S.l.]: W3C, 2008. Disponível em: <http://www.w3.org/TR/cooluris/>. Acessado em 29/04/2013.

SEQUEDA, J. Introduction to: triplestores. **SemanticWeb.com**, [S.l.], 2013. Disponível em: <http://semanticweb.com/introduction-to-triplestores_b34996>. Acessado em 19/05/2013.

STAAB, S.; STUDER, R. **Handbook On Ontologies**. 2.ed. Berlin: Springer, 2009. ISBN 978-3-540-70999-2.

VIRTUOSO. **Virtuoso Universal Server**. Disponível em: <http://virtuoso.openlinksw.com/>. Acessado em 19/06/2013.

W3C. **Large Triple Stores**. Disponível em: <http://www.w3.org/wiki/LargeTripleStores>. Acessado em 15/06/2013.

WENGER, E. **Communities of practice**: learning, meaning, and identity. [S.l.]: Cambridge University Press, 1998. ISBN 978-0-521-66363-2.

ZOPE. **The Zope 2 Book**. [S.l.: s.n.], 2010. Disponível em <http://docs.zope.org/zope2/zope2book/>. Acessado em 02/06/2013.