UNIVERSIDADE CANDIDO MENDES - UCAM
PROGRAMA DE PÓS-GRADUAÇÃO EM PESQUISA OPERACIONAL E INTELIGÊNCIA COMPUTACIONAL
CURSO DE MESTRADO EM PESQUISA OPERACIONAL E INTELIGÊNCIA COMPUTACIONAL

Rânielli Soares Neves de Azevedo

CATALOGAÇÃO E BUSCA SEMÂNTICA DE PROJETOS DE SOFTWARE UTILIZANDO OS PRINCÍPIOS LINKED DATA

CAMPOS DOS GOYTACAZES/RJ
Fevereiro de 2015

UNIVERSIDADE CANDIDO MENDES - UCAM
PROGRAMA DE PÓS-GRADUAÇÃO EM PESQUISA OPERACIONAL E INTELIGÊNCIA COMPUTACIONAL
CURSO DE MESTRADO EM PESQUISA OPERACIONAL E INTELIGÊNCIA COMPUTACIONAL

Rânielli Soares Neves de Azevedo

CATALOGAÇÃO E BUSCA SEMÂNTICA DE PROJETOS DE SOFTWARE UTILIZANDO OS PRINCÍPIOS *LINKED DATA*

Dissertação apresentada ao Programa de Pós-Graduação em Pesquisa Operacional e Inteligência Computacional da Universidade Candido Mendes – Campos /RJ, para obtenção do grau de MESTRE EM PESQUISA OPERACIONAL E INTELIGÊNCIA COMPUTACIONAL.

Orientador: Prof. Mark Douglas Azevedo Jacyntho D.Sc.

CAMPOS DOS GOYTACAZES/RJ
Fevereiro de 2015

RÂNIELLI SOARES NEVES DE AZEVEDO

CATÁLOGAÇÃO E BUSCA SEMÂNTICA DE PROJETOS DE SOFTWARE UTILIZANDO OS PRINCÍPIOS *LINKED DATA*

Dissertação apresentada ao Programa de Pós-Graduação em Pesquisa Operacional e Inteligência Computacional da Universidade Candido Mendes – Campos /RJ, para obtenção do grau de MESTRE EM PESQUISA OPERACIONAL E INTELIGÊNCIA COMPUTACIONAL.

Avaliada em ____/_____/____.

BANCA EXAMINADORA

Prof. Mark Douglas Azevedo Jacyntho D.Sc. - Orientador
Universidade Candido Mendes-Campos

Prof.ª Geórgia Regina Rodrigues Gomes. D.Sc.
Universidade Cândido Mendes-Campos

Prof. Breno Fabricio Terra Azevedo. D.Sc.
Instituto Federal Fluminense-Centro.

CAMPOS DOS GOYTACAZES, RJ
Fevereiro de 2015

AGRADECIMENTOS

Dedico este trabalho primeiramente a DEUS, por me conceder a vida e por sempre está presente ao meu lado me dando força para vencer toda e qualquer adversidade.

À minha Esposa, Leandra Souza, por toda compreensão, paciência, incentivo, amor e carinho, que sempre esteve ao meu lado nessa missão.

Á minha Mãe Alcimeia Soares e Meu Pai Sival Neves, eternos incentivadores, que sempre estão junto comigo em meus objetivos e pensamentos.

A minha Irmã, Rozana Soares e meu cunhado Ronivaldo, pelo incentivo e suporte.

Ao meu Orientador Mark Douglas, por ter acreditado e confiado em mim, por ter compartilhado grandes ensinamentos e pela amizade.

Aos meus familiares e amigos, em geral, pelo incentivo e apoio que sempre me deram.

Aos Amigos do grupo "*talento",* amigos de lutas onde sempre fomos suporte um para o outro nessa jornada.

Ao IFF pela oportunidade de realizar essa capacitação.

"*Enquanto houver vontade de lutar haverá esperança de vencer*"
(Santo Agostinho)

RESUMO

CATALOGAÇÃO E BUSCA SEMÂNTICA DE PROJETOS DE SOFTWARE UTILIZANDO OS PRINCÍPIOS *LINKED DATA*

A catalogação de projetos de *software* é essencial para o planejamento, registro e acompanhamento do ciclo de vida do *software* de forma organizada e precisa. Os catálogos/repositórios são fontes fundamentais de disseminação de conhecimento e reuso de *software* para a comunidade de engenharia de *software*. Não obstante, estes repositórios fazem parte da chamada *Web* convencional ou sintática, em que a informação é disponibilizada de forma textual, não estruturada, voltada apenas para exibição para humanos, via documentos HTML. Em outras palavras, uma vez que a informação não é estruturada, a máquina somente consegue exibi-la, mas não compreende o seu significado (semântica). Sendo assim, cabe aos humanos buscar as informações, integrá-las, consultá-las e interpretá-las para tomar decisões usando o conhecimento adquirido, um processo manual bastante demorado, tedioso e sujeito a erros. Esta pesquisa propõe construir um protótipo de aplicação *Web* que permita catalogar, de forma semiestruturada e compreensível por máquina, projetos de *software*, utilizando as tecnologias e padrões da *Web* Semântica em conformidade com os princípios *Linked Data*. Para isso, utiliza-se RDF como modelo de dados e ontologias consolidadas como DOAP, FOAF, SKOS para realizar a descrição dos termos dos projetos. Além disso, é realizada interligação automática (*mashup*) dos projetos com a *DBpedia,* fonte de dados central da *Web* de dados Interligados. Espera-se com este protótipo, que projetos de *software* sejam publicados e interligados na nova *Web* de dados, promovendo compartilhamento e reuso de conhecimento nesta área, impulsionados pelo poder de descoberta, integração e interpretação de informação que as ontologias proporcionam às máquinas.

Palavras-chave: *Web* Semântica, *Linked Data*, Ontologia, Catálogo de *Software*.

ABSTRACT

SEMANTIC CATALOGUING AND SEARCHING OF SOFTWARE PROJECTS USING THE LINKED DATA PRINCIPLES

Cataloging software projects is essential for planning, registration and monitoring of software lifecycle in an organized and precise way. Catalogs/repositories are fundamental sources of knowledge dissemination and software reuse for the software engineering community. Nevertheless, these repositories are part of the so-called conventional or syntactic Web, in which the information is made available in textual, unstructured format, just for display to humans, via HTML documents. In other words, once the information is unstructured, the machine can only display it, but doesn't understand its meaning (semantics). Therefore, it is up to humans to search for information, integrate, consult and interpret them to make decisions using the knowledge acquired, a manual process, fairly time consuming, tedious and error-prone. This research proposes to build a Web application prototype for cataloging software projects, in a semi-structured and comprehensible form by machine, using the technologies and Semantic Web standards in accordance with the Linked Data principles. To this end, we use RDF as data model and consolidated ontologies, like FOAF, DOAP, SKOS, for describing projects' terms. In addition, the projects are automatically mashed-up with the DBpedia - central dataset of Web of Linked Data. We hope with this application that software projects are published and interlinked in the new Web of Data, promoting sharing and reuse of knowledge in this area, driven by the power for discovering, integrating and interpreting information that ontologies provide to machines.

Keywords: Semantic Web, Linked Data, Ontology, Software Catalog.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ASF	*Apache Software Foundation*
API	*Application Programming Interface*
CPS	Catálogo de Projetos de *software*
CRUD	*Create Read Update Delete*
DOAP	*Description of a Project*
GRDDL	*Gleaning Resource Descriptions Dialects of Languages*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
IRI	*Internationalized Resource Identifier*
MIT	*Massachusetts Institute of Technology*
OWL	*Web Ontology Language*
POM	*Project Object Model*
RDF	*Resource Description Framework*
RDFS	RDF *Schema*
SQL	*Structured Query Language*
URI -	*Uniform Resource Identifier*
W3C	*World Wide Web Consortium*
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

São inúmeros os projetos de *softwares* produzidos por organizações públicas ou particulares e até mesmo por desenvolvedores independentes. São *softwares* proprietários, cujo uso limita-se às fronteiras da organização ou *softwares* livres, de código aberto (em inglês, *open source*) que precisam ser publicados na Internet para atrair usuários e colaboradores, *software* estes que são criados para atender aos mais variados domínios de conhecimento (SANTOS JR, 2010).

Sob o ponto de vista de engenharia de *software*, a busca incessante pela qualidade requer, sobretudo, que a organização possua um processo de desenvolvimento *software* sistemático claramente definido, que é continuamente aprimorado com base em um conjunto de projetos anteriores, conhecido por *baseline*. A catalogação organizada deste *baseline* é primordial para que se promova o efetivo reuso, quer seja em nível gerencial, quer seja em nível de especificação/desenvolvimento de *software* (PRESSMAN, 1995).

Já em termos de divulgação do *software*, obviamente a catalogação de projetos de *software* é essencial para permitir que estes projetos sejam encontrados e usados. Tal necessidade é bastante evidente dentre os *softwares* de código aberto. Cada *software* deste, em geral, possui em *Web site* descrevendo-o. Mas como atrair pessoas para estes *sites*? Enquanto muitos projetos contam com as máquinas de busca (p.ex.: Google) para serem encontrados, há um crescente número de catálogos ou repositórios de projetos, dos quais um projeto deve considerar fazer parte. Dentre outros, podemos destacar os seguintes catálogos: GIT-HUB[1], SourceForge[2], Google code[3], Portal do Software Público[4],

---

1 Git-hub - http://github.com/

2 SourceForge - http://www.sourceforge.net

3 Google code - https://code.google.com

4 Portal do Software Público - http://www.softwarepublico.gov.br/

FusionForge[5], Bitbucket[6], Freecode[7]. Tais catálogos são fonte fundamental de disseminação de conhecimento e reuso de software para a comunidade de engenheiros de *software*. Não obstante, estes catálogos fazem parte da chamada *Web* convencional ou sintática, em que a informação é disponibilizada de forma textual não estruturada, voltada apenas para exibição para humanos, via documentos HTML. Em outras palavras, uma vez que a informação não é estruturada, a máquina somente consegue exibi-la, mas não compreende o seu significado (semântica). Sendo assim, cabe aos humanos, buscar informações, integrá-las, consultá-las e interpretá-las para tomar decisões usando o conhecimento adquirido, um processo manual bastante demorado, tedioso e sujeito a erros.

Nos últimos anos, tem havido um crescente interesse por aplicações construídas segundo os padrões da *Web* Semântica. Trata-se de uma extensão da *Web* original denominada Web Semântica (também conhecida por *Web* 3.0) proposta por BERNES-LEE et al (2001). A ideia principal é utilizar a *Web* não apenas para compartilhar informação, mas também compartilhar o significado desta informação. Em outras palavras, associar a cada documento publicado na *Web* uma coleção de declarações estruturadas (metadados) inteligíveis por máquina, que permitam à máquina compreender o conteúdo do documento e, por conseguinte, tomar decisões inteligentes para nos auxiliar, realizando, em larga escala e de forma precisa, as tediosas tarefas que rotineiramente são executadas pelos usuários.

Como subconjunto mais pragmático da *Web* Semântica, em BERNES-LEE (2006) foi proposto o conceito de *Linked Data* (em português, *Dados Interligados*). Assim como a *Web*, a ideia de *Linked Data* é muito simples: por que não publicar e interligar dados diretamente na *Web?* Em outros termos, ao invés de apenas publicar metadados

---

5 FusionForge - http://fusionforge.org/

6 Bitbucket - https://bitbucket.org/

7 Freecode - http://freecode.com/

associados a documentos, publicar diretamente estes dados, independentemente de documentos, e interligá-los por meio de *links* semânticos (relacionamentos), criando a chamada *Web* de Dados (em inglês, *Web of Linked Data*). Seria uma *Web* de dados estruturados, totalmente compreensível por agentes de *software*, tornando, portanto, a busca por informações muito mais precisa e consistente, um grafo de dados global que não para de crescer (HEATH & BIZER, 2011).

Neste contexto, qualquer domínio de conhecimento pode (e deve) se beneficiar do movimento *Linked Data*, facilitando a descoberta, integração e reuso de informação. Em particular, no domínio de projetos de *software* (especialmente os projetos livres e de código aberto), catalogar semanticamente os projetos permite, sobremaneira, o uso e reuso de *software* completos, de bibliotecas e componentes, bem como boas práticas de *design* e desenvolvimento de *software*, sem mencionar que todos os projetos de *software* ficam devidamente sumarizados em um ponto central bem conhecido, processável de forma precisa pela máquina, permitindo-a responder perguntas como: Quais os projetos relacionados a certo tópico? Quais projetos foram escritos em uma linguagem particular? Quais projetos estão direta ou indiretamente relacionados a uma determinada pessoa?

1.1 MOTIVAÇÃO

A proposta de desenvolver um catálogo de projetos utilizando os princípios *Linked Data,* surge da dificuldade de encontrar ferramentas de catalogação de projetos de *software* que reúnam três importantes requisitos, a saber: publicação e reuso (*mashup*) de informação por meio do protocolo HTTP (ou seja, pela *Web*), codificação da informação por meio de um modelo semiestruturado inteligível por máquina que seja flexível o bastante para acomodar a heterogeneidade de esquemas e a necessidade de adaptação inerente ao volátil domínio de *software* e, por fim, inferências (deduções) automatizadas de novas informações, a partir dos esquemas utilizados. Em caráter complementar à descrição dos projetos de *software* em si, é bastante conveniente que o domínio ou área de conhecimento relacionado ao projeto também possa ser descrito, facilitando a busca por e/ou integração de projetos de um mesmo domínio ou de domínio correlatos. O uso de um modelo semiestruturado de dados, que facilmente acomode novos esquemas,

também visa este tipo de extensão.

Tanto para descrição dos projetos, quanto para a descrição do domínio de conhecimento relacionado aos projetos, torna-se necessário desenvolver um modelo ou esquema de domínio (*domain model*) e alimentar o sistema com dados que sejam instâncias deste modelo. Seria de grande valia se os usuários pudessem reusar/estender tanto modelos quanto dados pré-existentes na *Web*, sempre que possível. Dentre outras questões, os princípios *Linked Data* enfatizam justamente isso, impulsionando tanto o reuso de esquemas (aqui chamados de ontologias ou vocabulários), bem como dados disponibilizados na *Web* de Dados.

Outro fator que impulsiona este trabalho diz respeito aos catálogos/repositórios de projetos mencionados anteriormente (p.ex: GIT-HUB). Cadastrar um projeto nestes catálogos é uma tarefa bastante onerosa e pode consumir uma fatia considerável de tempo. Para complicar ainda mais, cada nova versão ou mudança significativa no projeto requer que alguém atualize, manualmente, todos os registros, visitando o *site* de cada catálogo/repositório do qual o projeto faça parte. Em muitos casos, isto significa que os projetos ou não são registrados ou seus registros se tornam obsoletos, dando a impressão de que o projeto foi descontinuado.

É preciso que todos estes catálogos sejam atualizados simultaneamente, de forma automática, em uma interação máquina-máquina, sem intervenção humana. Novamente, os princípios *Linked Data* proveem este mecanismo, uma vez que os dados são publicados em um modelo semiestruturado compreensível por máquina. Para tal, estes catálogos/repositórios de projetos de *software* precisam atuar como *sites* agregadores (em inglês, *pull model*) que, automaticamente, coletem registros de projetos de diferentes fontes na *Web*, integrando-os em seu banco de dados (*mashup*). Para um projeto ser incorporado nestes agregadores, é necessário criar uma descrição semiestruturada do projeto e publicá-la em um endereço HTTP (URI - *Uniform Resource Identifier*), o que usualmente significa publicar esta descrição via um *Web site* ou *Web service*. Com isso, basta registrar no *site* agregador o endereço HTTP (URI) da descrição do projeto e o *site* agregador pode monitorar este endereço à procura de mudanças e, portanto, se atualizar automaticamente. Dado que a descrição do projeto é armazenada localmente no próprio

Web site do projeto, não há a necessidade de alguém visitar o *site* agregador para atualizar seus registros. Basta atualizar o próprio *Web site* do projeto e esperar que o *site* agregador colete a nova informação conforme explicitado na Figura 1 .

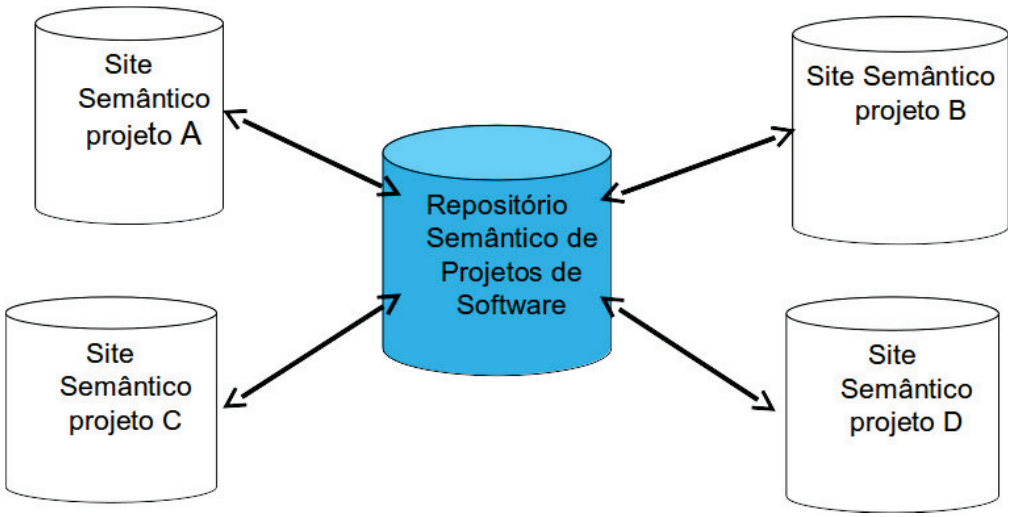

Figura 1- Repositório auto-autoalizável.

1.2 OBJETIVOS

O objetivo central deste trabalho é construir um protótipo de aplicação *Web* que permita catalogar, de forma semiestruturada compreensível por máquina, projetos de *software*, utilizando as tecnologias e padrões da *Web* Semântica em conformidade com os princípios *Linked Data*.

Um catálogo *Linked Data* de projetos de *software* que possa ser instalado, configurado e utilizado para registrar projetos de *software* por parte de qualquer agente desenvolvedor de *software*, desde iniciativas individuais até grandes corporações.

Juntamente com a aplicação em si, surgem também outros objetivos específicos, a saber: ratificar a eficiência da *Web* Semântica, especialmente o conceito *Linked Data*; propor um modelo ontológico extensível para descrição de projetos de *software* por meio

de seleção criteriosa de ontologias (esquemas) *Linked Data* de referência e, por fim, desenvolver um estudo de caso realista que corrobore a aplicabilidade da aplicação *Web* de catalogação de projetos de *software*.

1.3 CONTRIBUIÇÕES

Esta dissertação apresenta as seguintes contribuições para a área de catalogação de projetos de software:

- Aplicação *Linked Data* para catalogação semântica de projetos de *software* que pode ser instalada em qualquer ambiente de engenharia de *software*;
- Modelo ontológico extensível para descrição *Linked Data* de projetos de *software* que pode ser empregado por diferentes aplicações que necessitem registrar projetos de *software*;
- Uma abordagem de como publicar e consumir dados *Linked Data;*
- Uma abordagem semiautomática de realização de *mashup* semântico com a mais importante fonte de dados da *Web* de Dados: a *DBpedia*[8];

1.4 ESTRUTURA DO DOCUMENTO

Este trabalho está organizado em cinco capítulos conforme descrição abaixo.

No presente capítulo é apresentado o tema, questão de pesquisa, objetivos, justificativa proposta e contribuições. No capítulo 2 é apresentada a fundamentação teórica e conceitos utilizados neste trabalho. No capítulo 3 é abordado o detalhamento do

[8] DBpedia - http://dbpedia.org

desenvolvimento da aplicação. O capítulo 4 descreve o estudo de caso para validação do trabalho. O capítulo 5 apresenta os trabalhos relacionados e finalizando, o capítulo 6 expõe as considerações finais e alguns trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são abordados os principais conceitos utilizados neste trabalho.

2.1 CATÁLOGO DE PROJETOS DE SOFTWARE

Catálogo de Projetos de *software* (CPS) pode ser definido como um portal centralizador, no qual um usuário pode cadastrar e descrever um determinado projeto de *software*/aplicação, a fim de que o mesmo possa ser mais facilmente encontrado, seja em uma organização ou mesmo na internet. Fundamentalmente, um catálogo de projetos de *software* tem dois objetivos complementares: divulgação de projetos e *baseline* de projetos. O primeiro visa impulsionar o uso e contribuições para os projetos pré-existentes. Já o segundo objetiva o reuso de experiências passadas para serem utilizadas na construção de novos projetos e constante aprimoramento do processo de desenvolvimento de *software* utilizado (GARDLER, 2013). .

Basicamente, catálogos de projetosde *software* contêm: um tesauro, contendo uma taxonomia de temas e outros relacionamentos entre estes temas, usado para categorização dos projetos de *software*; descrição dos projetos de *software* em si, contendo propriedades como linguagem de programação, plataforma, versões e os agentes envolvidos nos projetos, ou seja, programadores, vendedores, testadores.

2.2 *WEB* SEMÂNTICA

A *Web* Semântica traz como objetivo dar significado às informações na *Web*, para que os computadores possam processar as mesmas, ou seja, a *Web* compreensível pelas máquinas. O conceito de *Web* Semântica, criado por Tim Bernes-Lee e a *World Wide Web*

Consortium W3C[9], foi apresentado de forma oficial ao mundo em Maio de 2001, por meio do artigo *"The Semantic Web: A New Form of Web Content That Is Meaningful to Computers Will Unleash a Revolution of New Possibilities"* (BERNERS-LEE et al, 2001).

Segundo Jacyntho (2012), a *Web* Semântica é definida como uma coleção de tecnologias e padrões que permitem que os computadores processem as informações da *Web*. Vale ressaltar que a *Web* Semântica não é uma *Web* separada, mas uma extensão da *Web* atual.

Em outubro de 1994, no *Massachusetts Institute of Technology* (MIT), foi fundado o W3C, um consórcio mundial liderado por Tim Berners-Lee que reúne empresas, instituições acadêmicas, profissionais e cientistas, com o objetivo de padronizar novas tecnologias que possibilitem estender gradativamente as funcionalidades do ambiente *Web*.

O W3C desempenha um papel fundamental no desenvolvimento e padronização de novas tecnologias baseadas no ambiente *Web*, de modo que desde sua criação o W3C tem se empenhado em desenvolver e padronizar tecnologias diretamente relacionadas ao projeto da *Web* Semântica.

Para que os computadores possam compreender o conteúdo de um documento *Web* é necessário associar ao mesmo metadados estruturados com representação semântica sobre o conteúdo. Metadados são dados que descrevem o conteúdo, ou seja, dados sobre dados. Por meio dos metadados é adicionada a semântica aos documentos, possibilitando que computadores possam processar e disponibilizar as informações de forma inteligente (RIOS, 2008).

Outro fator importante da *Web* Semântica é interoperabilidade, ou seja, competência que dois ou mais sistemas de computação distintos e distribuídos têm de

---

[9] W3C - *World Wide Web Consortium –https://www.w3.org*

atuar juntos, de forma autônoma, compartilhando informações, mas sempre com entendimento comum sobre o significado da informação que está sendo transmitida. Este entendimento comum é conseguido por meio do compartilhamento de esquemas ou modelos de representação do conhecimento chamados ontologias.

2.2.1 Arquitetura

A arquitetura da *Web* Semântica é composta de uma série de padrões organizados dentro de uma determinada estrutura que se relacionam entre si. Esta arquitetura é frequentemente representada usando um diagrama de camadas que foi primeiramente proposto por Tim Berners-Lee (2001). A Figura 2 apresenta a arquitetura da *Web* Semântica e suas tecnologias.

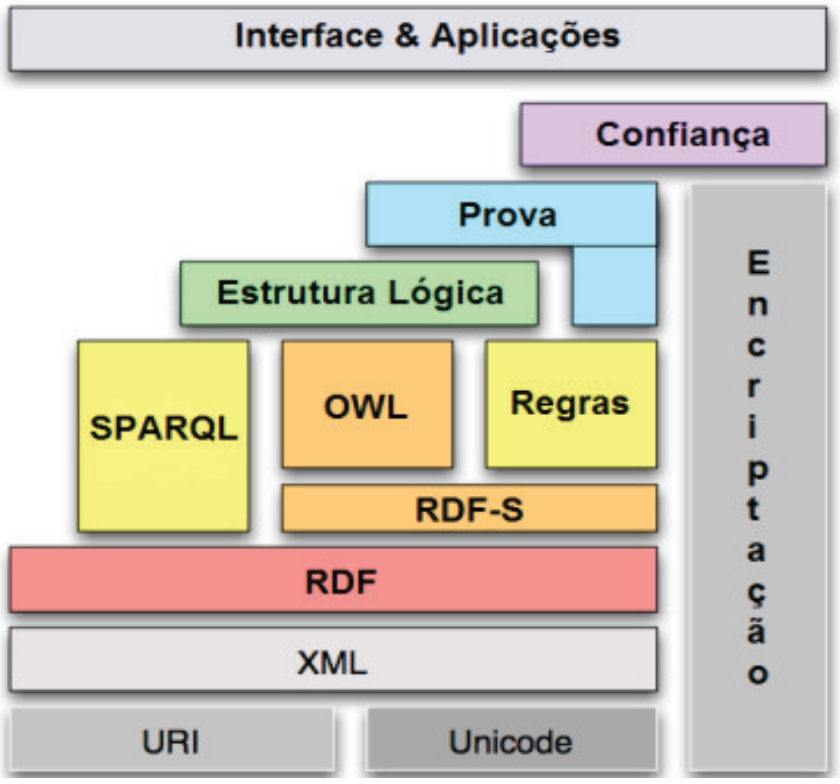

Figura 2: Arquitetura da *Web* Semântica atual (BRATT, 2006)

A arquitetura mencionada é organizada de forma que as camadas inferiores deem suporte às camadas superiores, em que cada uma possui a sua função específica.

Segundo SILVA (2012), as tecnologias mencionadas acima podem ser organizadas conforme a sua maturidade:

- Tecnologias Unicode, URI e XML: estas estão nas camadas de base da estrutura.
- Tecnologias recém-normatizadas: as tecnologias RDF, RDF-S e OWL estão no centro da estrutura.
- Tecnologias experimentais: as tecnologias SPARQL, Regras, Estrutura Lógica, Prova e Confiança, estão no topo da estrutura.

A seguir cada uma das tecnologias mencionadas será brevemente apresentada.

Unicode/URI

A primeira camada proporciona a interoperabilidade em relação à codificação de caracteres (*Unicode*) e uma única forma para identificação e localização de recursos (URI).

O *Uniform Resource Identifier* (URI) é um padrão para identificar um recurso físico ou abstrato de maneira única e global. Recurso pode ser absolutamente qualquer "coisa" (página Web, lei, pessoa, projeto, produto, evento, ideologia, etc.). É o elemento básico da estrutura a partir dos quais os demais componentes são construídos. Podemos citar como exemplo a URI http://www.ucam-campos.br/alunos/ranielli que é uma URI que identifica uma pessoa. *Unicode* é um padrão de codificação que permite aos computadores a representação e manipulação, de forma sólida, de qualquer texto independente de plataforma de *software* ou idioma. URIs somente podem conter um subconjunto dos caracteres ASCII. Sendo assim, para dar suporte a todos os caracteres e

idiomas *Unicode*, foi proposta uma extensão de URI chamada *Internationalized Resource Identifier*[10] (IRI) que nada mais é do que URIs que aceitam todos os caracteres *Unicode*. A ideia continua a mesma e o termo URI ainda continua sendo bastante utilizado na comunidade.

XML e XML Schema

Possibilita a criação de marcações para descrição das características de determinado dado (metadado) através da linguagem XML, possibilitando a integração de dados.

XML é uma linguagem para representação sintática de recursos de maneira independente de plataforma. Os documentos que têm sua estrutura e conteúdo representados na linguagem XML são denominados de documentos XML.

A XML *Schema*[11] é uma linguagem de definição para descrever uma gramática (ou esquema) para uma classe de documentos XML. A linguagem XML *Schema* fornece elementos para descrever a estrutura e restringir o conteúdo de documentos XML. Os *namespaces* fornecem um método para qualificar os nomes de elementos e atributos, utilizados nos documentos XML, através da associação destes nomes com os espaços de nomes identificados por referências de URI (FREITAS, 2011).

Modelo RDF

Segundo Manola e Miller (2004), *Resource Description Framework* (RDF) é um

---

[10] IRI -http://www.w3.org/International/O-URL-and-ident.html

[11] XML *Schema* - http://www.w3.org/XML/Schema.html

padrão para descrever semanticamente os recursos na *Web*, possibilitando a codificação, interação e o reuso de metadados, com o objetivo de promover a interoperabilidade entre aplicações que compartilham informações e que sejam compreendidas por sistemas na *Web*. Health e Bizer (2011) adicionam que o modelo RDF possibilita a implementação de aplicações genéricas com capacidade de operar no espaço de dados global.

Jacyntho (2012) relata que o modelo RDF tem como objetivo segmentar a informação (ou conhecimento) em pequenos pedaços semânticos (*Statement* ou declaração), para que ela seja compreendida pela máquina.

O modelo RDF é baseado no conceito de grafos, que são estruturados no formato (sujeito, predicado, objeto), também chamados de triplas. A Figura 3 apresenta de forma genérica uma notação gráfica de uma tripla.

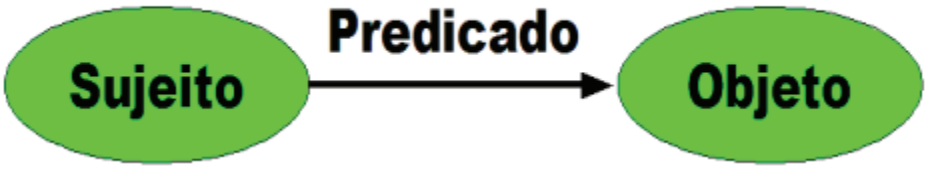

Figura 3: Exemplo genérico de notação gráfica de uma tripla.

Abaixo a descrição dos componentes de uma declaração/tripla:

- *Sujeito*: É o recurso ao qual uma sentença está se referindo;
- *Predicado*: Descreve uma característica, propriedade, ou relacionamento utilizado para descrever algo sobre este recurso;
- *Objeto*: É o valor de uma determinada característica do recurso referenciado, podendo ser um valor literal ou outro recurso.

A Figura 4 demonstra uma declaração RDF realista, em que todos os componentes são nomeados com URIs da seguinte forma:

O recurso `http://www.ucam-campos.br/projetos/projetoX` possui a propriedade `http://usefulinc.com/ns/doap#developer` cujo valor é o recurso `http://www.ucam-campos.br/alunos/ranielli`. Também possui uma declaração em que a propriedade http://usefulinc.com/ns/doap#name tem o valor literal `"ProjetoX"`.

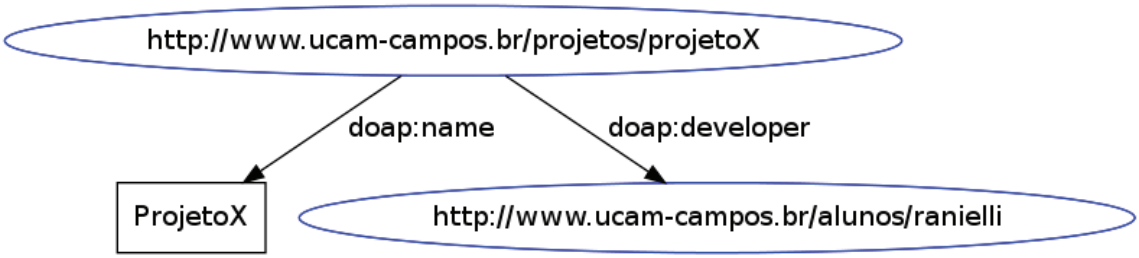

Figura 4: Uma declaração RDF realista de um recurso.

A Tabela 1 demonstra o mesmo grafo da Figura 4, mas utilizando notação em triplas em vez de notação gráfica.

Tabela 1: Notação em triplas.

Sujeito	Predicado	Objeto
<http://www.ucam-campos.br/projetos/projetoX>	<http://usefulinc.com/ns/doap#developer>	<http://www.ucam-ampos.br/aluno/ranielli>
<http://www.ucam-ampos.br/projetos/projetoX>	<http://usefulinc.com/ns/doap#name>	"ProjetoX" .

RDF *Schema* (RDFS)

Segundo Brickley e Guha (2004), RDF *Schema*[12] é um modelo de tipo de dados simplificado que permite a criação de classes e propriedades. Permite obter estruturas de informação sem ambiguidades. Disponibiliza um *framework* para representar informação

[12]RDF *Schema*- http://www.w3.org/TR/rdf-schema/

(metadados) sobre recursos. O RDF *Schema* é uma linguagem mínima para a representação de ontologias simples (taxonomias), denominada vocabulários RDF.

O conceito de classes em RDF representa um conjunto de recursos (indivíduos) com características semelhantes.

Ontologias (OWL)

Web Ontology Language[13] (OWL) é a linguagem definida como padrão pela W3C para o desenvolvimento de ontologias, de forma mais expressiva, possibilitando a representação do significado de termos em vocabulários e os relacionamentos entre estes termos.

A seção 2.3 apresentará de forma mais completa os conceitos Vocabulário e Ontologias.

SPARQL

Segundo Jardim (2010), SPARQL[14] é uma linguagem de consulta para os padrões (templates) de grafos RDF/RDFS. Está para o modelo RDF assim como SQL está para o modelo relacional. É formada por padrões (templates) de consultas, por um protocolo para uso na Web e formato XML para a saída dos resultados.

Como resultado da consulta SPARQL, obtém-se um subgrafo da consulta realizada sobre o grafo que representa o modelo.

A especificação SPARQL funciona com outras tecnologias de *Web* Semântica.

---

[13] Web Ontology Language (OWL) - http://www.w3.org/TR/owl-ref/

[14] SPARQL- http://www.w3.org/2009/sparql/wiki/Main_Page

Entre elas, está RDF, para representar dados; RDF *Schema* e OWL, para construir vocabulários; e a *Gleaning Resource Descriptions Dialects of Languages*[15] (GRDDL), para extração automática de dados semânticos de documentos.

Bem próximo à ideia do SQL, o SPARQL detém uma estrutura de seleção e projeção de dados S*elect-From-Where* conforme mencionado abaixo por Cunha et al (2012,pag 83):

- *Select*: Especifica uma projeção sobre os dados como a ordem e a quantidade de atributos e/ou instâncias que serão retornados.
- *From*: Declara as fontes que serão consultadas. Esta cláusula é opcional. Quando não especificada, assumimos que a busca será feita em um documento RDF/RDFS padrão (*default*).
- *Where*: Impõe restrições na consulta (seleção). Os registros retornados pela consulta deverão satisfazer as restrições impostas por esta cláusula.

Além das operações mencionadas acima, a linguagem oferece outras operações interessantes como o caso de *ASK* (para consultas booleanas cujo resultado é verdadeiro ou falso), *CONSTRUCT* (para construir um novo grafo RDF com base no resultado retornado da consulta) e *DESCRIBE* (retorna um grafo pré-definido representativo de um determinado recurso).

Regras

Rule Interchange Format [16] (RIF) é uma proposta de um formato padronizado para o compartilhamento e integração de regras lógicas de inferência entre diferentes

---

[15] *Gleaning Resource Descriptions Dialects of Languages* (GRDDL) - http://www.w3.org/TR/grddl/

[16] *Rule Interchange Format* (RIF) -http://www.w3.org/TR/rif-overview/

comunidades: acadêmicas, governamentais, empresariais, entre outros (JARDIM, 2010).

Há basicamente dois tipos de regras: declarativas e produtivas. Regras declarativas deduzem fatos. São sentenças do tipo "Se P então Q". Um exemplo é "Se uma mulher é irmã do pai de um indivíduo, então esta mulher é tia do referido indivíduo". Já regras produtiva deduzem ações. São sentenças do tipo "Se P então alguma ação precisa ser executada". Por exemplo, "Se Maria está com infecção, então ela precisa tomar antibiótico".

Camada Lógica Unificadora

Camada Lógica é uma representação unificada de expressões SPARQL, RIF e ontologias (descritas em OWL). O grande objetivo desta camada é oferecer um *framework* único para possibilitar a combinação dos elementos das camadas inferiores (JARDIM, 2010).

Camada de Prova

É camada que permite a validação dos recursos e regras da camada lógica, ou seja, garantir que os aspectos semânticos descritos estejam no formado adequado.

Camada de Criptografia

É a camada responsável por prover segurança e privacidade das informações. Ela é integrada às camadas e às aplicações (JARDIM, 2010).

Camada de Confiança

Nesta camada ocorre a verificação de autenticidade e evidência da confiabilidade de dados e serviços.

Camada de Aplicação

Camada na qual ocorre a interação entre o usuário e a aplicação *Web* Semântica (VASCONCELOS, 2012).

2.3 VOCABULÁRIO E ONTOLOGIAS

Para criar declarações RDF (triplas) de forma que possam se relacionar com diferentes fontes de dados na *Web* é necessário que existam termos e relacionamentos comuns, ou seja, um vocabulário compartilhado para um dado domínio de conhecimento.

As ontologias (ou vocabulários) são utilizadas para classificar os termos utilizados em uma determinada área de conhecimento e descrever seus relacionamentos. Antes de desenvolver uma aplicação semântica, se faz necessário definir a qual domínio de conhecimento ela se refere e escolher uma ou mais ontologias relacionadas a este domínio para usar na descrição das instâncias (recursos) RDF. Abaixo são apresentados alguns vocabulários (ontologias):

Friend-of-a-Friend[17] (FOAF) - Descrevem pessoas.

Description of a Project (DOAP) - Descrevem projetos de *Software*.

Simple Knowledge Organization System[18] (SKOS) - Representam taxonomias e tesaurus.

Creative Commons[19] (CC) - Descrevem termos de licença.

---

[17] *Friend-of-a-Friend*[17] (FOAF) - http://xmlns.com/foaf/0.1/

[18] *Simple Knowledge Organization System* (SKOS) - http://www.w3.org/2004/02/skos/core#

[19] *Creative Commons* (CC)- http://creativecommons.org/ns#

2.3.1 Inferências e Raciocinadores

Inferência é um conceito muito utilizado na *Web* Semântica, pois possibilita a descoberta de novas informações. Em outras palavras, as máquinas, de forma automática, deduzem novos conhecimentos por meio das ontologias. Em termos práticos, com base nos axiomas (regras) das ontologias, a máquina, a partir de um conjunto de triplas RDF de entrada, deduz outro conjunto de triplas RDF de saída automaticamente, trazendo à tona uma valiosa quantidade de conhecimento que estava implícito no conjunto original. Com isso, amplia a base de conhecimento e, por conseguinte, o poder de consulta(W3C BRASIL, 2013).

Raciocinador ou motor de regras pode ser definido como um *software* (biblioteca) que tem a capacidade de deduzir ou inferir consequências lógicas por meio de ontologias (LAGHI et al, 2011). Exemplos de Raciocinadores: RDF++ [20] (*Allegrograph*), JENA[21], Fact++[22], Pellet[23].

2.4 *LINKED DATA* E A *WEB* DE DADOS

A linguagem HTML é orientada para a estruturação de documentos textuais e não dados. Sendo assim se torna difícil para os aplicativos extrair dados estruturados a partir de páginas HTML.

Foram feitas algumas tentativas para reestruturar os dados na *Web*, uma delas

---

[20] RDF++ - http://franz.com/agraph/support/documentation/current/reasoner-tutorial.html

[21] *Jena* - https://jena.apache.org/

[22] Fact++ - http://owl.man.ac.uk/factplusplus/

[23] *Pellet* - http://clarkparsia.com/pellet/

foram os *Microformats*[24], criados para que os dados fossem publicados de forma estruturada dentro do código HTML. Sendo assim, os aplicativos conseguiriam extrair dados das páginas. No entanto, os *Microformats* possuem pontos fracos, já que a representação dos dados está restrita a um pequeno conjunto de entidades, e possivelmente não possibilita expressar as relações entre as entidades.

Outra tentativa de estruturar dados na *Web* foi o Uso de APIs, as quais simplificam o acesso de dados estruturados via HTTP. Nesse sentindo, surgiram as aplicações que integram vários serviços (*mashups*) que são acessados via APIs. A grande desvantagem é que não há um padrão e os desenvolvedores têm que conhecer as particularidades de cada API para poder acessar os dados.

Para interligar dados distribuídos na *Web* é necessário um padrão para especificar a existência e o significado das conexões entre os itens descritos nesses dados. Este padrão é o RDF. O RDF fornece uma maneira flexível para descrever as coisas, seja uma pessoa, um local, ou conceitos abstratos e como eles se relacionam com as outras coisas. Quando utilizamos RDF para publicar e ligar dados, estamos criando uma *Web* em que os dados se tornam mais visíveis e mais passíveis de serem utilizados (MANOLA E MILLER, 2004).

Como parte do desenvolvimento de *Web* Semântica, Tim Berners-Lee introduziu o conceito de *Linked Data* (dados linkados/ligados). *Linked Data* refere-se a um conjunto de melhores práticas para publicar e conectar dados de forma estruturada na *Web* (HEATH E BIZER, 2011).

2.4.1 Princípios Linked Data

A proposta de Tim Berners-Lee, feita em 2006, para que o *Linked Data* funcione,

---

[24] Microformats - http://microformats.org/

exige quatros princípios:

- Utilizar URIs como nome para coisas ou recursos.
- Utilizar URIs HTTP para que um cliente (seja humano ou máquina) possa procurar por estes nomes.
- Quando um cliente acessar (dereferenciar) uma URI, informações úteis deverão ser providas, representadas pelo formado RDF.
- Incluir *links* para outras URIs, para que outros recursos possam descobertos (navegação).

O primeiro princípio define o uso de referências URI para identificar, não apenas documentos *Web* e conteúdos digitais, mas também objetos do mundo real e conceitos abstratos (CUNHA et al, 2011).

O segundo princípio defende o uso de URIs HTTP para identificar os objetos e os conceitos abstratos definidos pelo princípio um, possibilitando que essas URIs sejam dereferenciáveis sobre o protocolo HTTP. Neste contexto, dereferenciar é o processo de recuperar uma representação de um recurso identificado por uma URI, em que um recurso pode ter várias representações como documentos HTML, RDF, XML entre outros (CUNHA et al, 2011).

Para que as aplicações Web possam processar dados disponíveis na Internet é importante que haja um padrão para disponibilizar os dados, nesse contexto é que o terceiro princípio define o uso do RDF, como o modelo padrão para publicação de dados estruturados na Web.

O quarto princípio define o uso de *links* para conectar recursos. Em *Linked Data*, os *links* são capazes de descrever a relação entres os recursos conectados, ou seja, são tipificados. São chamados de *links* RDF, para que sejam diferenciados dos *hiperlinks* da *Web* convencional (HEATH E BIZER, 2011). Os princípios *Linked Data* possibilitam que os dados sejam descobertos, acessados, integrados e utilizados com maior facilidade na *Web*.

Um dos exemplos efetivos da utilização dos princípios da *Linked Data* é o projeto *Linking Open Data*[25], fundado em janeiro de 2007 e apoiado pelo W3C. O objetivo principal deste projeto é identificar conjuntos de dados disponíveis sob licenças abertas e convertê-los para RDF de acordo com os princípios *Linked Data*.

Criação de URIs adequadas

Os recursos são nomeados via URIs, logo, ao publicar dados *Linked data* é interessante criar boas URIs para a identificação dos recursos. Para atingir esse objetivo devem-se seguir algumas considerações como:

- Usar URIs HTTP pra tudo, tornando-as habilitadas para serem dereferenciadas.
- Não é aconselhável URIs com detalhes de implementação.
- As URIS devem ser mantidas estáveis e persistentes, para que não haja quebra de links já realizados.

Usar URIs dereferenciáveis

Toda URI HTTP tem que ser apta a ser dereferenciada, ou seja, os clientes HTTP podem procurar a URI usando o protocolo HTTP e recuperar uma descrição do recurso que é identificado pela URI.

Um ponto importante a ser ressaltado na utilização de URIs para identificar objetos do mundo real é não confundir o próprio objeto com o documento da Web que a descreve. O correto é criar uma URI específica para descrever o objeto e outra para o documento que a descreve. As descrições destinadas para seres humanos são representadas como

---

[25] *Linking Open Data* -http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData

HTML, já as destinadas para as máquinas são representados em RDF. A ideia básica da negociação de conteúdo é que clientes HTTP enviem por meio de cabeçalhos HTTP o pedido para indicar que tipos de documentos desejam. Servidores recebem as requisições e analisam cabeçalhos e selecionam a resposta apropriada. Se os cabeçalhos indicam que o cliente solicita HTML, o servidor responderá enviando um documento HTML, já se o cliente solicitar RDF, o servidor enviará um documento RDF.

Formatos de serialização de RDF

RDF não é um formato de dados e sim um modelo abstrato de dados para descrever recursos por meio de triplas (sujeito, predicado e objeto). Para publicar um grafo RDF na *Web*, deve-se primeiramente ser serializado, usando uma sintaxe RDF. Os formatos de Serialização de RDF - RDF/XML[26] e RDFa[27] foram padronizados pela W3C sendo que também são utilizados: *Turtle*[28], N-*Triples*[29] e JSON[30].

Links RDF

Segundo Heath e Bizer (2011), para descrever os relacionamentos entre dois recursos no âmbito da *Linked Data*, utiliza-se o conceito de *links* RDF.

Um link RDF é composto por três referências URI. As URIs referentes ao sujeito e

---

[26] RDF/XML - http://www.w3.org/TR/REC-rdf-syntax/

[27] RDFa - http://www.w3.org/TR/xhtml-rdfa-primer/

[28] *Turtle*- http://www.w3.org/TR/turtle/

[29] N-*Triples* - http://www.w3.org/TR/n-triples/

[30] JSON- http://www.w3.org/TR/json-ld/

objeto identificam os recursos relacionados, enquanto que a URI referente ao predicado define o tipo de relacionamento entre os recursos. *Links* RDF internos realizam a conexão de recursos dentro de uma única fonte de dados. *Links* externos conectam recursos disponibilizados por diferentes fontes dados. Nos links externos, as URIs referentes ao sujeito e ao predicado do *link* pertencem a diferentes *namespaces*. *Links* externos são de extrema importância para a *Web* de Dados, já que proporcionam a interligação das fontes de dados em um espaço global de dados. Existem três tipos importantes de RDF *links*:

- *Links* de relacionamento – indicam "coisas" relacionadas em outras fontes de dados.
- *Links* de identidade – identificam as URIs que estão relacionadas ao mesmo recurso.
- *Links* de vocabulário - apontam para o vocabulário para definir dados.

2.4.2 A Web de Dados

Segundo o W3C, a *Web* de dados é a evolução da *Web* de documentos, e tem como objetivo possibilitar que os computadores possam processar e compreender os dados da *Web*, ou seja, utilizar toda capacidade computacional que normalmente fica ociosa. A *Web* de dados é um espaço global de dados utilizando os conceitos *Linked Data*, que contém um grafo de bilhões de RDFs, de inúmeras fontes de dados. As principais características da *Web* de dados são:

- É genérica e pode conter qualquer tipo de dado;
- Qualquer um pode publicar os dados;
- Dados são auto descritivos;

Entidades estão conectadas por *links* RDF, criando um grafo global de dados, possibilitando a descoberta de novas fontes de dados.

A *Web* de dados tem como um dos principais responsáveis pelo seu crescimento o projeto da W3C *Linking Open Data* (LOD), iniciado em 2007. Uma das principais ações do

projeto foi identificar dados disponíveis sobre licença aberta e convertê-la para RDF, de acordo com os princípios *Linked Data*.

As figuras 5, 6 e 7 abaixo demonstram o crescimento da Web de dados, a partir do início do projeto LOD.

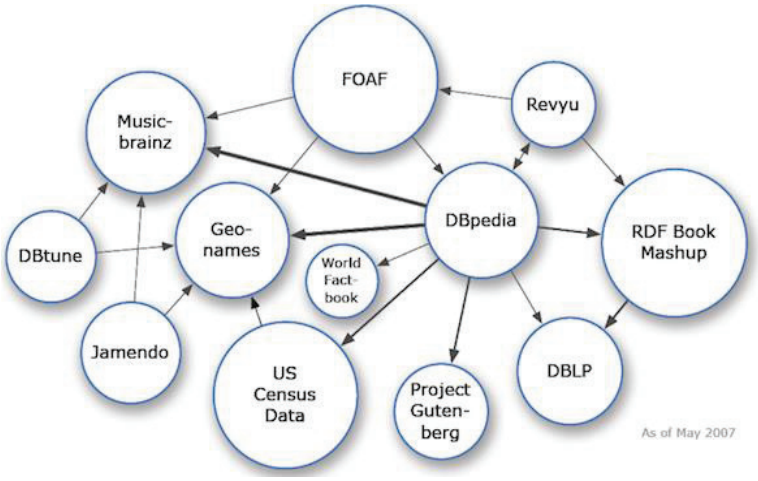

Figura 5: LOD em 2007(LOD, 2007).

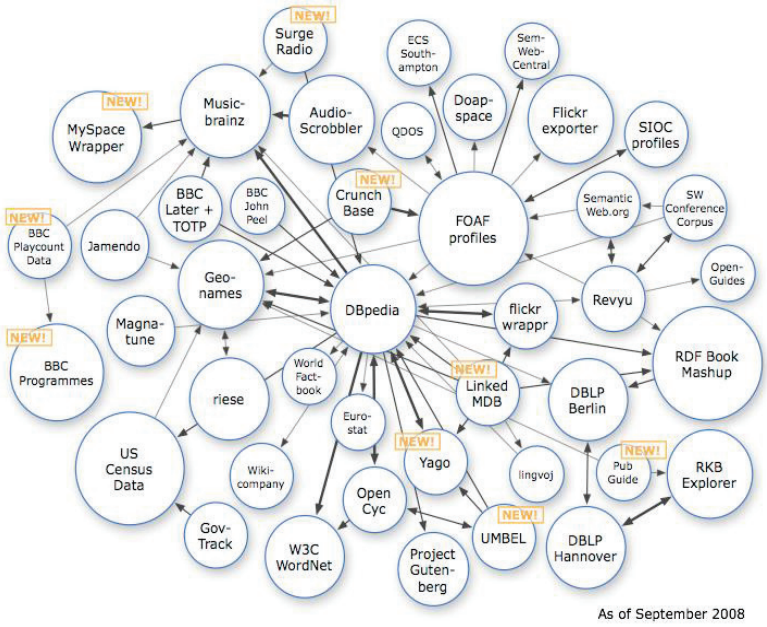

Figura 6: LOD em 2008(LOD, 2008).

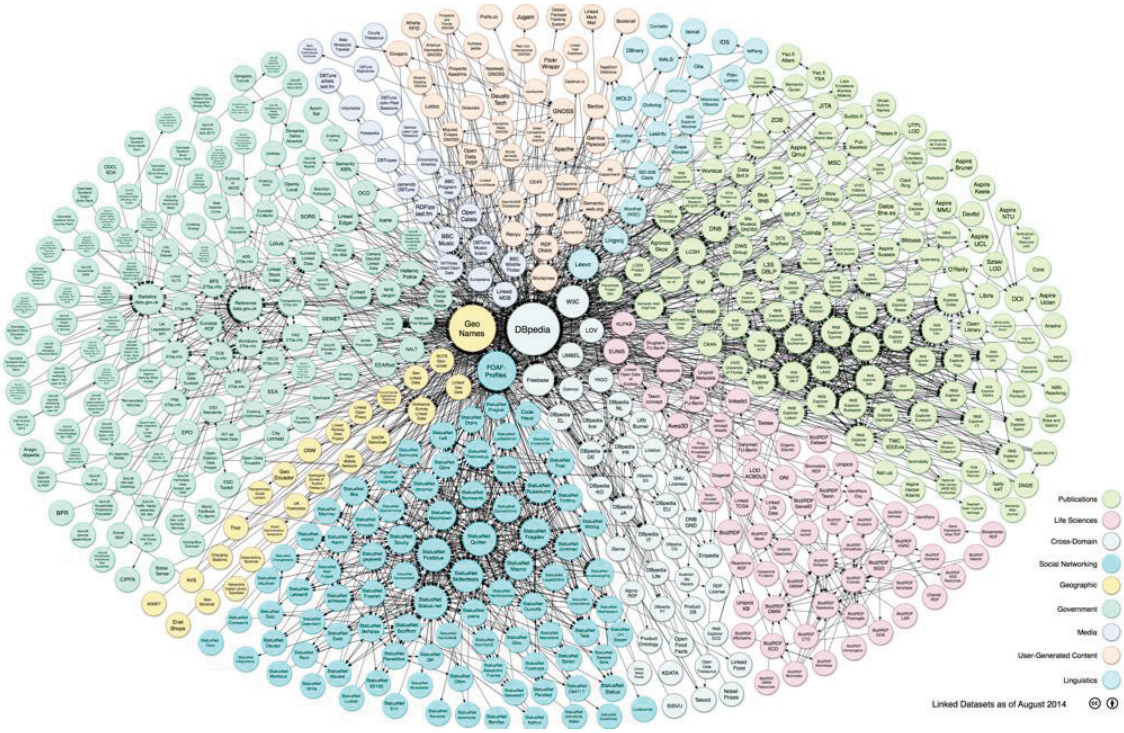

Figura 7: LOD em 2014(LOD, 2014).

Na nuvem LOD, os conjuntos de dados são classificados dentro dos seguintes domínios de conhecimento: geográfico, governo, mídia, bibliotecas, ciência, comércio, conteúdo gerado por usuário e conjunto de dados entre domínios.

2.5 TRIPLE STORES

Triple Stores ou banco de dados RDF são repositórios de dados que armazenam declarações RDF (triplas) e utilizam uma linguagem de consulta para recuperar as informações. A W3C recomenda o uso da linguagem SPARQL.

Na maioria dos bancos de dados RDF é disponibilizado uma interface para que as consultas possam ser realizadas, conhecida como SPARQL *endpoint*. Outro ponto de destaque nos bancos de dados RDF se refere à estrutura (esquema) de armazenamento

que deve ser padronizado no formato de tripla (sujeito, predicado, objeto), facilitando a otimização das consultas a serem realizadas (JACYNTHO, 2012). Como exemplo de *Triple Stores* podemos destacar o AllegroGraph[31] (que é utilizado neste trabalho), Virtuoso[32], 4store[33], Sesame[34], dentre outros.

[31]AllegroGraph- http://www.franz.com/agraph/allegroph

[32]Virtuoso- http://virtuoso.openlinksw.com/

[33]4Store- http://4store.org/

[34]Sesame- www.openrdf.org

3 DESENVOLVIMENTO DO PROTÓTIPO

Neste capítulo, são apresentadas as fases de desenvolvimento do protótipo proposto, detalhando-se as tecnologias utilizadas e já relacionadas no Capítulo 2.

3.1 DESCRIÇÃO GERAL

O sistema proposto neste trabalho consiste em uma aplicação *Web* para catalogação de projetos de *Software* utilizando as tecnologias e padrões da *Web* Semântica e seguindo os princípios *Linked Data*.

A aplicação apresenta como principal funcionalidade cadastrar os dados da área de engenharia de *software* utilizados em um projeto de *software*. Visualizando estas funcionalidades pode-se pensar que a aplicação se trata de um sistema CRUD convencional, mas não é. Internamente, ou seja, de forma transparente para o usuário, a aplicação é estruturada para realizar processamentos que vão se diferenciar das aplicações convencionais. Todos os dados de projetos, pessoas, organizações e *tags* (palavras-chave) entre outros conceitos utilizados para descrever um projeto de *Software*, são armazenados internamente em RDF e utilizando ontologias *Linked Data* consagradas. Sobre essas ontologias que foram cuidadosamente selecionadas podemos destacar o uso da DOAP, FOAF, VCARD[35] e SKOS.

Tendo compreendido que todo projeto tem sua representação em RDF, a seguir é descrito como funciona a aplicação e como os usuários submetem projetos.

Todas as entidades residentes na aplicação são identificadas por uma URI. Quando um projeto é submetido, o projeto, as *tags* (palavras-chave associadas aos projetos), as

[35] vCard - http://www.w3.org/2006/vcard/ns

organizações, as pessoas, todos estes itens recebem uma URI cada. O esquema de URIs utilizado segue os princípios *Linked Data*. Portanto, para cada recurso (quer seja um projeto, quer seja uma pessoa), temos três URIs relacionadas: a URI do próprio recurso, a URI de um documento RDF descrevendo o recurso e a URI de um documento HTML que exibe o recurso. Por exemplo, para um projeto XPTO teríamos as três URIs seguintes:

- URI que identifica o próprio projeto como um recurso não informacional (não residente na *Web*): `http://www.example.com/projects/xpto`.
- URI que identifica um documento *Web* que contém a representação RDF/XML do projeto. Esta URI é retornada quando o cliente requisita uma representação RDF do recurso: `http://www.example.com/projects/xpto.rdf`
- URI que identifica um documento *Web* que contém a representação HTML do projeto. Esta URI é retornada quando o cliente requisita uma representação HTML do recurso: `http://www.example.com/projects/xpto.html`

Com o esquema de URIs apresentado acima, a aplicação oferece a funcionalidade de negociação de conteúdo, de modo que se um agente de *software* requisitar a URI do projeto `http://www.example.com/projects/xpto` no formato HTML, ele receberá como resposta um `HTTP 303 See Other` contendo a URI `http://www.example.com/projects/xpto.html` que contém a representação HTML do projeto para que seja apresentado para visualização humana. Já se o agente de *software* requisitar a URI do projeto `http://www.example.com/projects/xpto` no formato RDF, ele receberá como resposta um `HTTP 303 See Other` contendo a URI `http://www.example.com/projects/xpto.rdf` que contém a representação RDF/XML do projeto, possibilitando que a máquina possa dereferenciar(obter) essa representação e interpretá-la. A Figura 8 apresenta o funcionamento da negociação de conteúdo.

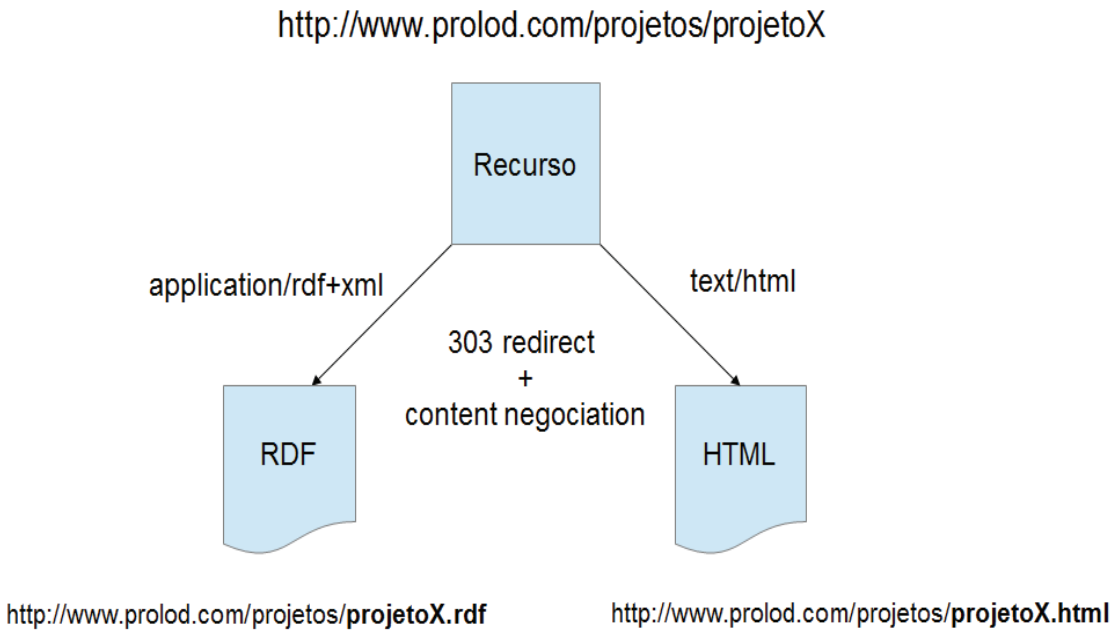

Figura 8: Negociação de Conteúdo.

Note que, diferentemente de *Web sites* convencionais, graças ao mecanismo de negociação de conteúdo, esta aplicação atende tanto clientes humanos (via navegadores *Web* tradicionais), gerando uma representação HTML do recurso, quanto clientes máquinas, gerando uma representação RDF do recurso. Desta forma, esta aplicação já está devidamente preparada para publicar dados na *Web* de Dados para serem reusados por outras aplicações. Um exemplo de reuso são os repositórios agregadores descritos na introdução desta dissertação.

No cadastro de um projeto de *software*, temos o conceito principal Projeto, com propriedades que estão diretamente relacionadas ao escopo de projetos de *software*, descritos com a ontologia DOAP (nome, descrição, linguagem de programação, versões, categorias, desenvolvedores, mantenedores, vendedores, repositórios, etc.). Entre os atributos dessa estrutura principal, duas propriedades têm um papel crucial: descrição e categorias. Como descrito a seguir, estas propriedades são usadas para realizar o *Linked Data mashup* com a *DBpedia*, que é a *Wikipédia* traduzida em RDF e considerada a fonte de dados central da *Web* de Dados.

Categorias são palavras-chave (*tags*) semânticas, conceitos que formam uma

folksonomia semântica. Estas categorias servem para classificar os projetos. Um projeto pode pertencer a várias categorias ao mesmo tempo. Na verdade, as categorias são organizadas como um tesauro, no qual temos uma taxonomia e outros relacionamentos previstos na ontologia *Simple Knowledge Organization System* (SKOS).

Esse consumo automático de informação da *Web* de Dados incrementa sobremaneira experiência do usuário e, por conseguinte, o valor do *site*, sem precisar que estes dados preciosos tenham que ser inseridos manualmente pelo usuário.

Outro diferencial notável que a *Web* semântica traz para a aplicação é que a partir do momento que os dados são descritos usando RDF e ontologias, a máquina pode inferir novos fatos (triplas RDF) importantes automaticamente, que seriam impossíveis de serem deduzidos manualmente por humanos, em larga escala.

Como todos os dados são estruturados em RDF, a aplicação trabalha com a linguagem de consulta SPARQL para a realização de consultas precisas e com inferências.

3.2 METODOLOGIA

Inicialmente foi feito um levantamento bibliográfico sobre *Web* Semântica em publicações nacionais e internacionais, realizado em fontes diversas (livros, periódicos, anais de congresso, dissertações, teses e documentos eletrônicos da Internet, entre outros).

Como a proposta foi desenvolver um protótipo de aplicação *Web* para catalogação de projetos de *Software* utilizando os princípios *Linked Data*, também foi feito um levantamento sobre os catálogos de *softwares* existentes (trabalhos relacionados) buscando identificar os requisitos principais necessários para o desenvolvimento do catálogo.

Após o levantamento dos requisitos, foi feita a modelagem conceitual e a criação do modelo de domínio para projetos de software. Em seguida, foi realizado o

mapeamento do modelo de domínio em ontologias comumente utilizadas na *Web* de Dados e, posteriormente, foi desenvolvida aplicação proposta.

E, por fim, a aplicação foi testada em um estudo de caso realista, verificando as suas funcionalidades de acordo com os objetivos propostos para o desenvolvimento deste projeto.

3.3 ESPECIFICAÇÃO

Nesta seção, é descrito todo o processo de especificação da aplicação, desde a elicitação e análise de requisitos até a construção do modelo de domínio conceitual.

3.3.1 Requisitos funcionais (casos de uso)

Esta seção aborda os casos de uso do projeto e uma breve descrição de cada caso de uso (historietas).

Durante o levantamentos de requisitos foram identificados oito casos de uso, a saber: "*Cadastrar Projeto*", "*Cadastrar Pessoa*", "*Cadastrar Tag (Categoria)*", "*Cadastrar Repositório*", "*Cadastrar Especificação*", "*Cadastrar Versão*", "*Cadastrar Organização*" e "*Buscar Projetos*". A Figura 9 exibe o diagrama de casos de uso da aplicação. A seguir são descritos cada caso de uso, sendo que o detalhamento de alguns atributos será apresentado na seção 3.4.2.

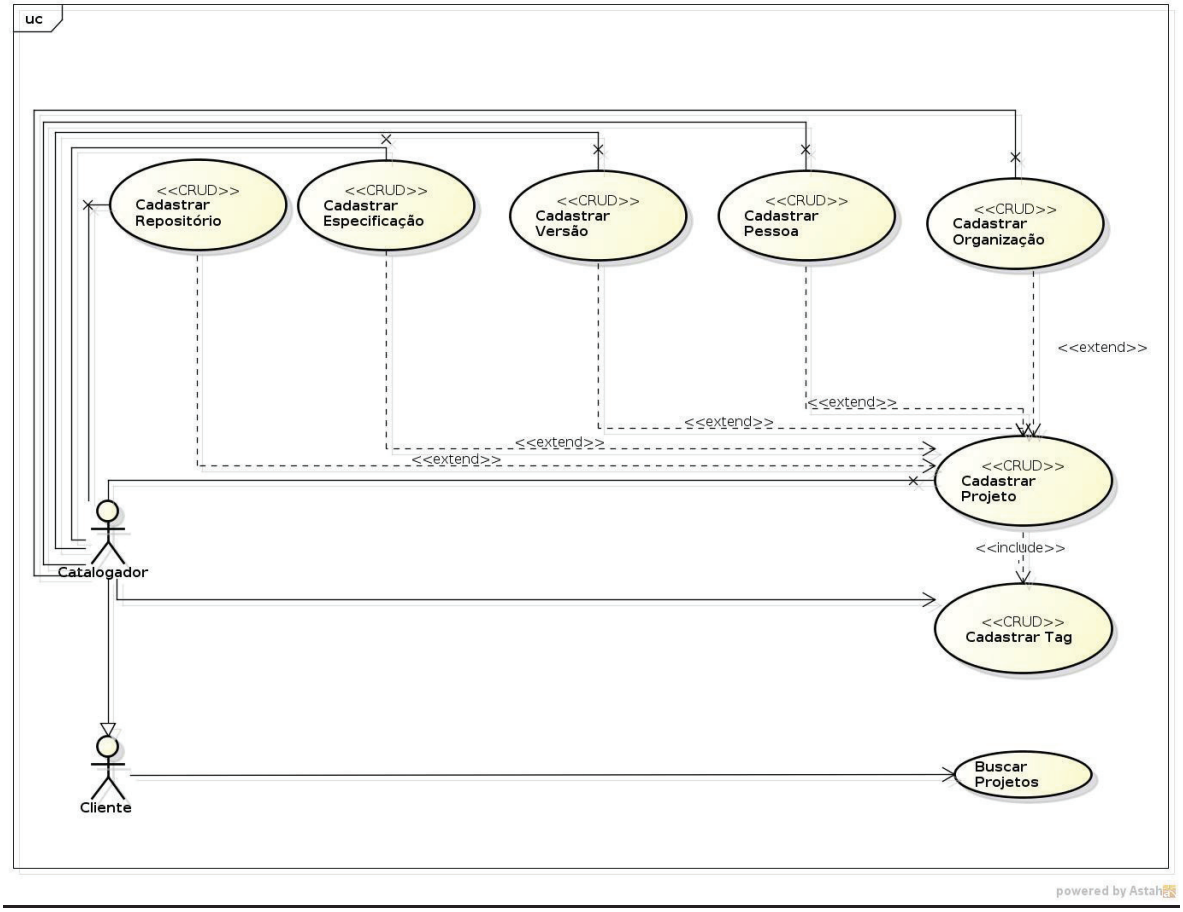

Figura 9: Diagrama de Caso de Uso da aplicação.

Cadastrar projeto (CRUD)

Este caso de uso permite ao catalogador cadastrar novos projetos, em que o mesmo informa vários dados, sendo que podemos destacar: nome do projeto, categorias, descrição do projeto, data de criação, desenvolvedores, mantenedores, entre outros. Podemos considerar esse cadastro sendo central da aplicação, visto que através dele é realizado o relacionamento com outro cadastro de outros casos de uso como é o caso de categorias, desenvolvedores, mantenedores, entre outros. Após confirmar o cadastro dos dados, os dados são descritos em RDF que contém uma URI para identificar o projeto. A URI é gerada com a junção de uma URI BASE da aplicação com o nome do projeto.

Para editar o projeto é utilizada a mesma estrutura do cadastro de um novo projeto, sendo que mesmo que mude o dado do atributo nome, a URI do projeto permanece a mesma, visto que outros relacionamentos podem ter sido criados para essa URI.

Para selecionar os projetos já cadastrados, é apresentada uma lista com o nome dos projetos e as opções de editar/visualizar/remover o projeto. Ao selecionar um determinado projeto, a aplicação apresenta somente os atributos que têm valores cadastrados.

Para remover um projeto, ao selecionar a opção excluir, a aplicação apresenta uma mensagem pedindo a confirmação de exclusão e, se confirmada, são excluídas todas as triplas que contém a URI do projeto como sujeito ou objeto.

Cadastrar categoria (CRUD)

Para criar uma categoria (*tag*), o catalogador deve inserir as seguintes propriedades: nome, nomes alternativos, nomes ocultos, escopo, definição, exemplo de uso, histórico de definições, relacionamento com categorias filhas e relacionamento com categorias mães. Após confirmar o cadastro dos dados, os dados são descritos em RDF que contém uma URI para identificar a categoria. A URI é gerada com a junção de uma URI BASE da aplicação com o nome da categoria.

Após confirmar o cadastro destes dados, o valor do atributo nome é utilizado para encontrar recursos provenientes da *Web* de Dados relacionados à categoria criada. Feito isto, é apresentada ao catalogador uma lista contendo o nome e um comentário resumido de cada recurso. O catalogador seleciona os recursos que estão relacionados à categoria em questão. O sistema cria RDF-*links* entre a categoria e os recursos selecionados, perfazendo o *mashup* com a *Web* de Dados.

Para editar uma categoria é utilizada a mesma estrutura do cadastro de uma nova categoria, sendo que mesmo que mude o nome, a URI da categoria permanece a mesma, visto que outros relacionamentos podem ter sido criados para essa URI.

De forma análoga aos projetos, para selecionar uma categoria já cadastrada é

apresentada uma lista com o nome da categoria e as opções de visualizar/editar/remover a categoria. No entanto, uma categoria somente pode ser removida se não houver projetos associados a ela.

Os outros casos de uso CRUD

Os outros casos de uso CRUD relacionados ao cadastro de Pessoa, Organização, Repositório, Versão, Especificação, possuem fluxos de execução similares, diferenciando-se apenas nas informações cadastradas. Para não ficar repetitivo, a descrição destes casos de uso foi omitida, mas as informações cadastradas estão detalhadamente apresentadas no modelo de domínio apresentado a seguir.

Buscar Projetos

Este caso de uso permite ao cliente realizar consultas na aplicação. São oferecidas consultas para buscar projetos por categoria e por palavra-chave, com ou sem inferência.

3.3.2 Modelo de domínio para projetos de software

Como base na descrição da aplicação e nos casos de uso, foi elaborado um modelo de domínio convencional para projetos de *software*, exposto pelo diagrama de classes UML da Figura 10. Para tal, foram analisados: características de vários *softwares open source* disponíveis na *Web*; processos de *softwares* bem conhecidos (ágeis e rigorosos); repositórios de código fonte (GitHub, Portal do *Software* Público, *Apache Foundation*, etc.) e experiências/pontos de vista de equipes de desenvolvimento.

Esta seção apresenta este modelo, descrevendo as classes e suas propriedades. Estas propriedades serão mais bem detalhadas na seção 3.4.2, na qual é descrito o mapeamento das mesmas em ontologias *Linked Data*.

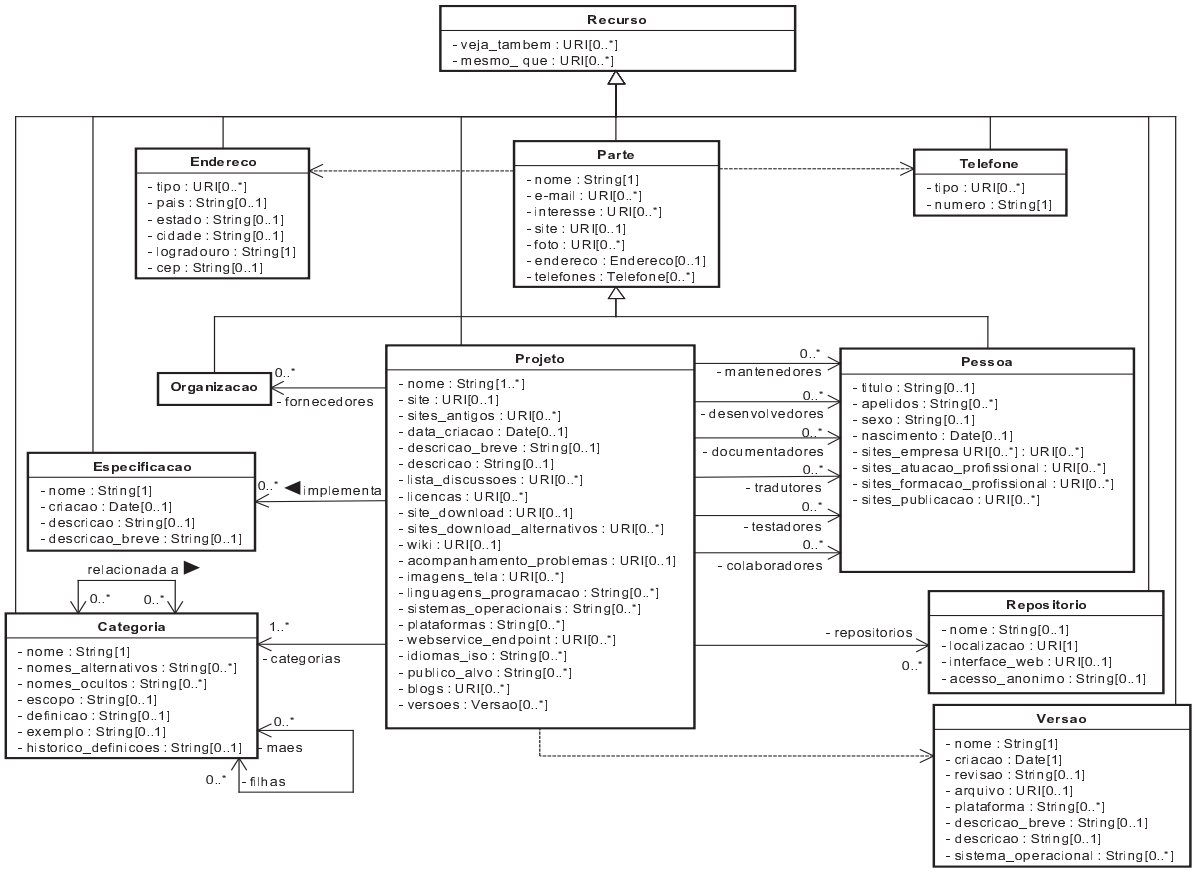

Figura 10: Modelo de Domínio Conceitual.

A classe *Recurso* é o supertipo comum a todos os outros tipos e representa qualquer objeto de domínio identificado por uma URI. Esta classe possui propriedades comuns, prevendo o *Linked Data mashup*: *veja_tambem* e *mesmo_que*. A propriedade *veja_também* relaciona um recurso com URIs de outros recursos relacionados a ele de alguma forma. Já a propriedade *mesmo_que* tem uma semântica mais forte e relaciona o recurso com outras URIs que representam o mesmo recurso, ou seja, deve ser usado quando encontramos outras URIs que são identificadores alternativos para o mesmo recurso em questão. Todas as outras classes herdam as propriedades desta classe.

Parte é a classe base para outras duas classes: *Pessoa* e *Organização*. *Parte* possui os atributos *nome*, *e-mail*, *interesse*, *site*, *foto* e atributos que contêm outros tipos de classe como *endereço*, que pode conter uma instância da classe *Endereço* e *telefones*

que pode conter um conjunto de instâncias da classe *Telefone*. A classe *Pessoa* além dos atributos herdados da classe *Parte* tem os seguintes atributos específicos: *título*, *apelidos*, *sexo*, *nascimento*, *sites_empresas*, *sites_atuação_profissional*, *sites_formação_profissional* e *site_publicação*.

A classe *Organização*, basicamente herda toda a classe parte, ou seja, contêm todas as propriedades da classe mencionada.

Projeto, obviamente, é a classe central do modelo, possuindo relacionamentos com a maioria das classes do modelo. Possui os atributos: *nome*, *sites*, *sites_antigos*, *data_criação*, *descrição_breve*, *descrição*, *lista_discussões*, *licenças*, *sites_download*, *sites_download_alternativos*, *wiki*, *acompanhamento_problemas*, *imagens_tela*, *linguagens_programação*, *sistemas_operacionais*, *plataformas*, *webservice_endpoint*, *idiomas_iso*, *publico_alvo*, *blogs*, *versões*. Além desses atributos, também possuem os seguintes relacionamentos: *fornecedores* com a classe *Organização*, *mantenedores* com a classe *Pessoa*, *desenvolvedores* com a classe *Pessoa*, *documentadores* com a classe *Pessoa*, *tradutores* com a classe *Pessoa*, *testadores* com a classe *Pessoa*, *colaboradores* com a classe *Pessoa*, *implementa* com a classe *Especificação*, *categorias* com a classe *Categoria*, *repositórios* com a classe *Repositório* e *versões* com a classe *Versão*.

A classe *Especificação* representa especificações que o projeto implementa ou está de acordo (p.ex.: especificação Java *Persistence* API - JPA). Esta classe possui os seguintes atributos: *nome*, *criação*, *descrição* e *descrição_breve*.

A classe *Categoria* representa categorias (*tags* ou *palavras-chave*) para classificação dos projetos. Esta classe possui os seguintes atributos: *nome, nomes_alternativos, nomes_ocultos, escopo, definição, exemplo, histórico_definições* e autorelacionamentos do tipo "mães" e "filhas" para construção de "hierarquias" (folksonomias).

A classe *Repositório* representa repositórios em que o projeto de *software* está hospedado. Esta classe possui os seguintes atributos: *nome*, *localização*, *interface_web*, *acesso_anônimo*.

A classe *Versão* representa as várias versões do projeto de software e possui os seguintes atributos: *nome*, *criação*, *revisão*, *arquivo*, *plataforma*, *descrição_breve*, *descrição* e *sistema_operacional*.

3.4 MODELO ONTOLÓGICO

Após a elaboração do modelo conceitual de domínio, várias ontologias *Linked Data* do catálogo *Linked Open Vocabularies* (LOV[36]) foram cuidadosamente analisadas e selecionadas. Por fim, o referido modelo de domínio foi mapeado nas classes e propriedades das ontologias selecionadas. As ontologias em questão serão apresentadas na seção 3.4.1. Ontologias Linked Data selecionadas

3.4.1 Ontologias Linked Data selecionadas

Description of a Project (DOAP)[37] é uma ontologia utilizada para descrever projetos de *software* (em especial, *open source*). Tem sido amplamente usada em catálogos de projetos de *softwares* de organizações como a fundação *Apache*, fundação *Mozzila* e o indexador de pacote *Python*. Obviamente, esta é a ontologia central do modelo. Para abreviar URIs, usamos o prefixo "*doap*" para referenciar o *namespace* da ontologia (*http://usefulinc.com/ns/doap#*).

Friend of a Friend (FOAF) é uma ontologia bastante popular, para descrever pessoas, suas atividades e relações com outras pessoas e entidades. Foi escolhida em função das pessoas (atores) envolvidas em um projeto de *software*. Para abreviar URIs, usamos o prefixo "*foaf*" para referenciar o *namespace* da ontologia

---

[36] *Linked Open Vocabularies* (LOV) - http://lov.okfn.org/dataset/lov/

(*http://xmlns.com/foaf/0.1/*).

vCard Ontology (vCARD) é uma ontologia que representa a especificação vCard[38] em RDF/OWL, sendo utilizada para descrever pessoas e organizações. Foi escolhida para complementar a ontologia FOAF, com algumas propriedades extras como telefone e endereço. Para abreviar URIs, usamos o prefixo "*vcard*" para referenciar o *namespace* da ontologia (*http://www.w3.org2006/vcard/ns#*).

Simple Knowledge Organization System (SKOS) é uma ontologia para representar sistema de organização do conhecimento (taxonomias, tesauros, folksonomias, entre outros). Foi escolhida para categorização dos projetos de *software*, para facilitar busca e navegação. Para abreviar URIs, usamos o prefixo "skos" para referenciar o *namespace* da ontologia (*http://www.w3.org/2004/02/skos/core#*).

Além das ontologias acima, também foram utilizadas algumas propriedades dos vocabulários RDF, RDFS e OWL amplamente usadas no contexto *Linked Data* para interligar dados, como, por exemplo, rdfs:seeAlso ("veja também") e owl:sameAs ("o mesmo que").

3.4.2 Mapeamento do modelo de domínio nas ontologias

As Tabelas de 2 a 13, a seguir, apresentam o mapeamento de cada classe/propriedade do modelo da Figura 10 nas ontologias, bem como o significado de cada elemento do modelo.

Tabela 2: Mapeamento da classe *Recurso*.

Classe *Recurso*		
Elemento do Modelo	Significado	Mapeamento

[38] Especificação vCard - http://tools.ietf.org/html/rfc6350

Classe *Recurso*	Superclasse geral que representa um recurso.	rdfs:Resource
veja_tambem	*Mashup* com recursos relacionados na *Web* de Dados.	rdfs:seeAlso
mesmo_ que	*Mashup* com recursos idênticos na Web de Dados.	owl:sameAs

Tabela 3: Mapeamento da classe *Parte*.

Classe *Parte*		
Elemento do Modelo	Significado	Mapeamento
Classe *Parte*	Parte (superclasse para pessoas e organizações).	foaf:Agent e vcard:Agent
nome	Nome da parte	foaf:name
e-mail	E-mail da parte.	foaf:mbox ou foaf:mbox_sha1sum
interesse	Tópicos de interesse da parte.	foaf:topic_interest
site	Site da parte.	foaf:homepage
foto	Foto (imagem) representativa da parte.	foaf:depiction
endereco	Endereço da parte.	vcard:adr
telefone	Telefones da parte.	vcard:Tel

Tabela 4: Mapeamento da classe *Telefone*.

Classe Telefone		
Elemento do Modelo	Significado	Mapeamento
Classe *Telefone*	Telefone de uma parte.	vcard:Tel
tipo	Tipo de telefone. Usar as seguintes classes da ontologia vcard: BBS, Car, Cell, Fax, Home, ISDN, Modem, Msg, PCS, Pager, Video, Voice, Work, Pref.	rdf:type
numero	Número completo com DDI e DDD do telefone.	rdf:value

Tabela 5: Mapeamento da classe *Endereço*.

Classe Endereco		
Elemento do Modelo	Significado	Mapeamento
Classe *Endereco*	Endereço de uma parte.	vcard:Address

tipo	Tipo de endereço. Usar as seguintes classes da ontologia vcard: Dom, Home, Intl, Parcel, Postal, Pref, Work.	rdf:type
pais	País do endereço.	vcard:country-name
estado	Unidade da federação (estado) do endereço.	vcard:region
cidade	Cidade do endereço.	vcard:locality
logradouro	Logradouro do endereço.	vcard:street-address
cep	Endereço Postal.	vcard:postal-code

Tabela 6: Mapeamento da classe *Organização.*

Classe *Organizacao*		
Elemento do Modelo	Significado	Mapeamento
Classe *Organizacao*	Organização (Empresa).	foaf:Organization e vcard:Organization

Tabela 7: Mapeamento da classe *Pessoa.*

Classe *Pessoa*		
Elemento do Modelo	Significado	Mapeamento
Classe *Pessoa*	Pessoa relacionada ao Projeto.	foaf:Person
titulo	Como tratar a pessoa: Mr. , Sr., Prof., etc.	foaf:title
apelidos	Apelidos (nick names) da pessoa;	foaf:nick
sexo	Sexo masculino/feminino da pessoa.	foaf:gender
nascimento	Data de nascimento da pessoa.	vcard:bday
sites_empresa	Sites da empresa onde a pessoa trabalha.	foaf:workplaceHomepage
sites_atuacao_profissional	Sites sobre a atuação (papel) profissional da pessoa.	foaf:workInfoHomepage
sites_formação_profissional	Sites de instituições de ensino onde da pessoa se formou.	foaf:schoolHomepage
sites_publicacao	Sites onde tem as publicações da pessoa.	foaf:publications

Tabela 8: Mapeamento da classe *Projeto.*

Classe *Projeto*

Elemento do Modelo	Significado	Mapeamento
Classe *Projeto*	Projeto de software.	doap:Project
nome	Nome do projeto.	doap:nome
site	Site atual do projeto.	doap:homepage
sites_antigos	Sites antigos do projeto.	doap:old-homepage
data_criacao	Data da criação do projeto.	doap:created
descricao_breve	Descrição breve do projeto.	doap:shortdesc
descricao	Descrição completa do projeto.	doap:description
lista_discussoes	Site de lista de discussões sobre o projeto.	doap:mailing-list
licencas	Tipo de licenças de uso do projeto.	doap:license
site_download	Site principal para download do projeto.	doap:download-page
sites_download_alternativos	Site de alternativos para download do projeto.	doap:download-mirror
wiki	Wiki sobre o projeto.	doap:wiki
acompanhamento_problemas	Servidor de issue tracking do projeto.	doap:bug-database
imagens_tela	Imagens (screenshots) da tela do projeto.	doap:screenshots
linguagens_programacao	Linguagens de programação usadas no projeto.	doap:programming-language
sistems_operacionais	Sistemas operacionais (SOs) compatíveis com o projeto.	doap:os
plataformas	Plataforma (não específica de SO, p.ex: Java) compatíveis com o projeto.	doap:platforma
webservice_endpoint	Acesso ao Projeto como um Web Service.	doap:service-endpoint
idiomas_iso	Códigos de idiomas (ISO) do projeto.	doap:language
publico_alvo	Público alvo (tipo de usuário) do projeto.	doap:audience
blogs	Blogs sobre o projeto.	doap:blog
versoes	Versões do projeto.	doap:release
repositorios	Repositório onde se encontra o projeto (p.ex: GitHub)	doap:repository
implementa	Especificações que o projeto implementa.	doap:implements
fornecedores	Fornecedor (vendedor) do projeto.	doap:vendor
mantenedores	Mantenedor (líderes) do projeto.	doap:maintainer
desenvolvedores	Desenvolvedor de software do projeto.	doap:developer
documentadores	Colaborador para a documentação do projeto.	doap:documenter
tradutores	Colaborador para tradução (localização) do projeto.	doap:translator
acesso_anonimo	Repositório para acesso anônimo (sem login) ao projeto.	doap:tester
colaboradores	Colaborador do projeto.	doap:helper

Tabela 9: Mapeamento da classe *Especificação*.

Classe *Especificacao*		
Elemento do Modelo	Significado	Mapeamento
Classe *Especificacao*	Especificação a ser implementada por um software.	doap:Specification
nome	Nome da especificação.	doap:name
data_criacao	Data da criação da especificação.	doap:created
descricao_breve	Descrição breve da especificação.	doap:shortdesc
descricao	Descrição completa da especificação.	doap:description

Tabela 10: Mapeamento da classe *Categoria*.

Classe *Categoria*		
Elemento do Modelo	Significado	Mapeamento
Classe *Categoria*	Categoria de projeto de software.	skos:Concept
nome	Nome da categoria.	skos:prefLabel
nomes_alternativos	Nomes alternativos (sinônimos) da categoria.	skos:altLabel
nome_ocultos	Nomes ocultos da categoria. Usados para buscas.	skos:hiddenLabel
escopo	Limita onde a categoria pode ser usada.	skos:scopeNote
definicao	Definição (significado) da categoria.	skos:definition
exemplo	Exemplo de uso da categoria.	skos:example
histórico_definicoes	Mudanças no significado da categoria.	skos:historyNote
filhas	Categoria filhas.	skos:narrower
maes	Categoria mães.	skos:broader
relacionada a	Categorias relacionadas (propriedade simétrica).	skos:related

Tabela 11: Mapeamento da classe *Repositório*.

Classe *Repositorio*		
Elemento do Modelo	Significado	Mapeamento
Classe *Repositorio*	Repositório (de código) do Projeto. Usar subclasses da ontologia doap: GitRepository, SVNRepository, etc.	doap:Repository
nome	Nome do repositório.	doap:name
localizacao	Localização do repositório.	doap:location
interface_web	Interface web do repositório.	doap:browse
acesso_anonimo	Repositório para acesso anônimo.	doap:anon-root

Tabela 12: Mapeamento da classe *Versão*.

Classe *Versao*		
Elemento do Modelo	Significado	Mapeamento
Classe *Versao*	Versão do Projeto.	doap:Version
nome	Nome da versão.	doap:name
criacao	Data de criação da versão.	doap:created
revisao	Identifica a revisão da versão.	doap:revision
arquivo	Arquivo da versão.	doap:file-release
plataforma	Plataforma (não específica de SO, p.ex: Java) compatíveis com o projeto.	doap:platform
descricao_breve	Descrição breve da versão.	doap:shortdesc
descricao	Descrição completa da versão.	doap:description
sistema_operacional	Sistemas operacionais (SOs) compatíveis com a versão.	doap:os

Tabela 13: Mapeamento tipos primitivos.

Tipos primitivos		
Elemento do Modelo	Significado	Mapeamento
String	String (sequência de caracteres).	xsd*:String
Date	Data no formato AAAA-MM-DD.	xsd:date

*xsd: prefixo para o *namespace* de XML *Schema* (http://www.w3.org/2001/XMLSchema#)

Após realizar o mapeamento chegou-se ao modelo de domínio ontológico apresentado na Figura 11.

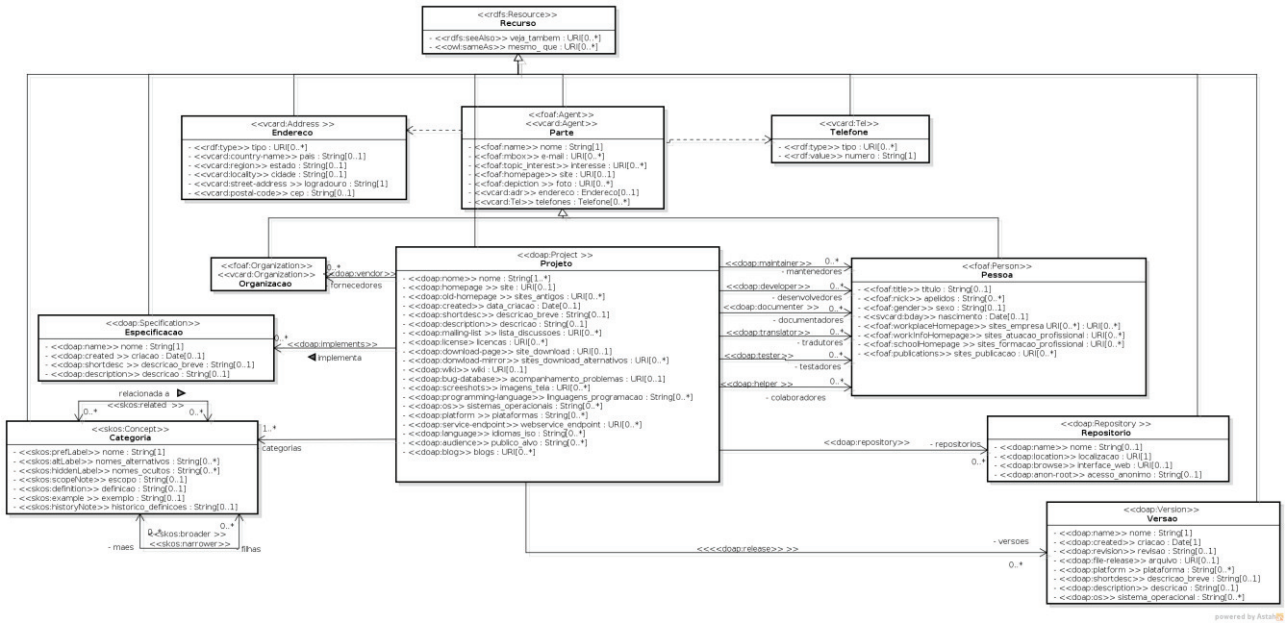

Figura 11 - Modelo de Domínio Ontológico.

3.4.3 Exemplo de instanciação do modelo

Para ilustrar o uso do modelo, a *Figura 12*, a Figura 13 e a Figura 14 descrevem, respectivamente, uma categoria, uma pessoa, e um projeto de *software* completo (que referencia a categoria e a pessoa descritas anteriormente). Vale destacar as propriedade *rdfs:SeeAlso*, *owl:sameAs* (representada pelo símbolo "=") e *foaf:topic_interest* que estabecelem o *mashup* semântico, interligando com recursos da *DBpedia*. A letra "a" representa a propriedade *rdf:type*, usada para classificar recursos em classes.

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://ucam-campos.br/projetos/category/Linked_data> a skos:Concept ;
    rdfs:seeAlso <http://dbpedia.org/page/Semantic_Web> ;
    = <http://dbpedia.org/resource/Linked_data> ;
    skos:altLabel "Web of Linked Data" ;
    skos:broader <http://ucam-campos.br/projetos/category/Semantic_Web> ;
    skos:definition "Conjunto de principios para publicar dados RDF diretamente na Web." ;
    skos:prefLabel "Linked Data" ;
    skos:related <http://ucam-campos.br/projetos/category/Database> ;
    skos:scopeNote "Linked Data conforme os padroes e tecnologias definidos pelo W3C." .
```

Figura 12: Instância de uma categoria em *Turtle.*

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix vcard: <http://www.w3.org/2006/vcard/ns#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://ucam-campos.br/professores/markjacyntho> a foaf:Person ;
    vcard:adr [ a vcard:Address,
                vcard:Work ;
            vcard:country-name "Brasil" ;
            vcard:locality "Campos dos Goytacazes" ;
            vcard:postal-code "28030-335" ;
            vcard:region "Rio de Janeiro" ;
            vcard:street-address "Rua Anita Peçanha, 100 - Pq. São Caetano" ] ;
    vcard:tel [ a vcard:Home,
                vcard:Tel,
                vcard:Voice ;
            vcard:hasValue <tel:+557555555555> ] ;
    foaf:mbox <mailto:markjacyntho@gmail.com> ;
    foaf:name "Mark Douglas de Azevedo Jacyntho" ;
    foaf:topic_interest <http://dbpedia.org/resource/Linked_data> .
```

Figura 13: Instância de uma pessoa em *Turtle*.

```
@prefix doap: <http://usefulinc.com/ns/doap#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://ucam-campos.br/projetos/xpto> a doap:Project ;
    doap:bug-database <http://issues.ucam-campos.br/jira/browse/XPTO> ;
    doap:category <http://ucam-campos.br/projetos/category/Linked_data> ;
    doap:created "2012-12-03" ;
    doap:description "O projeto UCAM Xpto implementa a especificação Linked Data Platform 1.0 do W3C." ;
    doap:developer <http://ucam-campos.br/alunos/raniellisoares>,
        <http://ucam-campos.br/professores/markjacyntho> ;
    doap:download-page <http://xpto.ucam-campos.br/download.html> ;
    doap:homepage <http://xpto.ucam-campos.br> ;
    doap:implements <http://www.w3.org/TR/ldp/#ldpr> ;
    doap:license <http://usefulinc.com/doap/licenses/asl20> ;
    doap:mailing-list <http://xpto.ucam-campos.br/mail-lists.html> ;
    doap:name "Ucam Xpto" ;
    doap:programming-language "Java" ;
    doap:release [ a doap:Version ;
            doap:created "2013-03-03" ;
            doap:name "Ucam Xpto Estavel" ;
            doap:revision "3.0.0" ],
        [ a doap:Version ;
            doap:created "2013-10-26" ;
            doap:name "Ucam Xpto Alpha" ;
            doap:revision "3.1.2-alpha-4" ] ;
    doap:repository [ a doap:SVNRepository ;
            doap:browse <http://svn.ucam-campos.br/viewvc/labs/xpto/> ;
            doap:location <http://svn.ucam-campos.br/repos/labs/xpto/> ] ;
    doap:shortdesc "Uma plataforma para Linked Data." ;
    rdfs:seeAlso <http://dbpedia.org/page/Semantic_Web>,
        <http://dbpedia.org/resource/Linked_data> .
```

Figura 14*:* Instância de um projeto *em* Turtle

3.5 ARQUITETURA PROPOSTA E TECNOLOGIAS UTILIZADAS

A arquitetura da aplicação é multicamadas, baseada na conhecida arquitetura *Model-View-Controller* (MVC)(KRASNER et al.,1988), na qual as camadas de interface com o usuário (apresentação), de controle de interação (controle) e de objetos de negócio (modelo) são devidamente desacopladas, visando facilidade de manutenção e reuso dos seus componentes. Além destas três camadas básicas, outras camadas também foram inseridas para dar suporte ao modelo de dados RDF. Para desenvolvimento da aplicação como um todo foi utilizado o *framework Web Ruby on Rails*[39] e, para cada camada em particular, foram utilizadas ferramentas (bibliotecas) específicas que serão descritas adiante, nesta seção. A Figura 15 apresenta visão geral da arquitetura da aplicação.

[39] *Ruby on Rails* - http://rubyonrails.org/

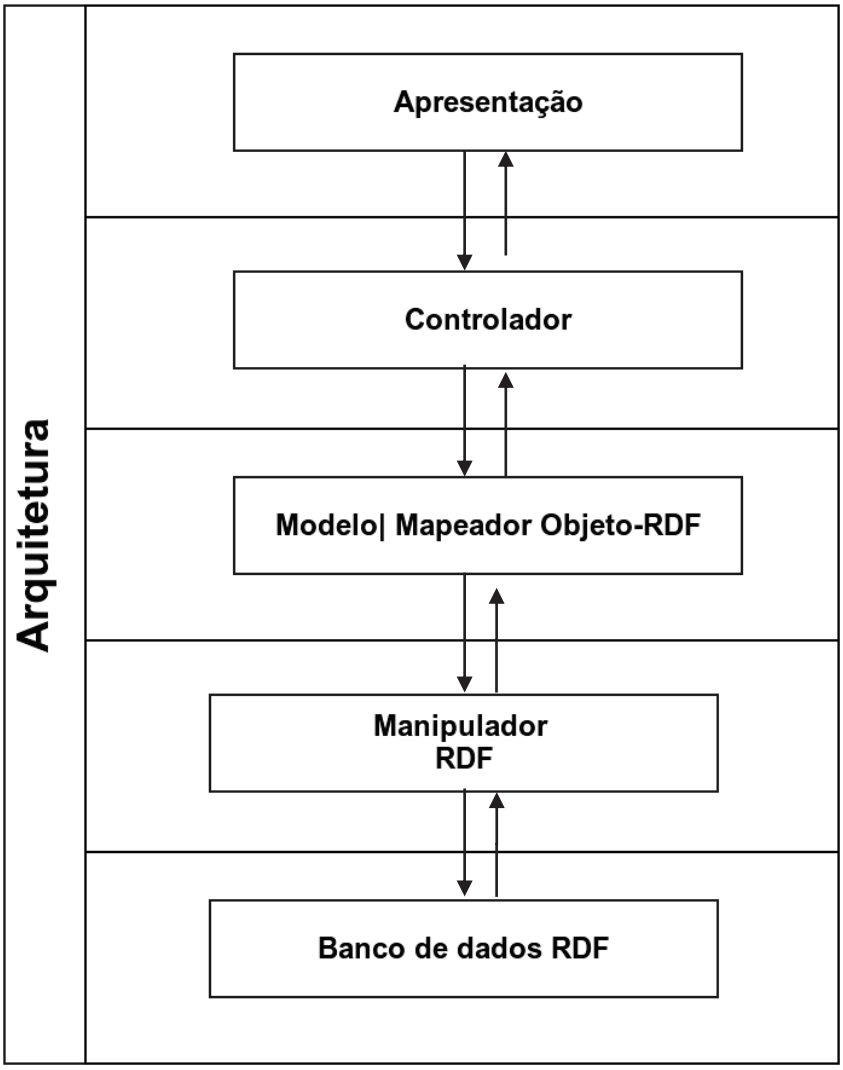

Figura 15: Arquitetura geral da aplicação.

3.5.1 Componentes da arquitetura

Camada de Apresentação

Esta camada é responsável pelos *scripts* das telas da aplicação *Web*. Telas de edição (formulários) e telas de apresentação (navegação). Temos também a página

principal (*home page*), contendo o menu principal para as telas dos casos de uso da aplicação. Toda esta camada foi implementada sobre a estrutura de *Views* do *Ruby on Rails.*

Camada de Controle

Para cada formulário da aplicação, temos um controlador que se relaciona com o respectivo modelo a fim de realizar as operações de cadastro (CRUD) e traduzir eventos da camada de apresentação em invocação de operação em objetos de negócio residentes na camada de modelo. Esta camada utiliza o controlador do *Ruby on Rails*, estendendo da classe *ApplicationController.*

Camada de Modelo e Mapeamento objeto RDF

Esta camada contém os objetos de negócio da aplicação. Para se trabalhar com os dados RDF de forma mais abstrata utilizou-se um mapeador objeto-RDF para mapear objetos de domínio paro o grafo RDF subjacente. A ideia é encapsular o grafo RDF, permitindo manipulá-lo como objetos convencionais. Esta abordagem nos habilita trabalhar com uma visão orientada a objetos (ou orientada a recursos), sem, no entanto, perder acesso à natureza orientada a triplas do modelo RDF.

Nesse contexto, é realizado o mapeamento das classes/propriedades do modelo domínio de *software* para classes/propriedades das ontologias *Linked Data* utilizadas. Para implementar esse modelo de ontologias de modo que possam trabalhar com dados RDF como classe *Ruby* convencionais, utilizou-se a biblioteca (gem) *Spira*[40], que basicamente se resume em um mapeador de objeto-RDF que oferece métodos para simplificar a manipulação de dados RDF em *Ruby* (LAVENDER, 2010), de modo que possa trabalhar com triplas (recurso-propriedade-valor) em que o recurso é o objeto que se deseja descrever, a propriedade se refere a alguma ontologia mapeada e o valor é o

[40] *Spira* - https://github.com/ruby-rdf/spira

dado que o usuário insere.

O ponto de partida para o uso da biblioteca *Spira* é a classe *Recurso*, esta herda da classe *Base* do *Spira.* A classe *Base* funciona de forma similar ao *ActiveRecord* nativo do *Rails* disponibilizando métodos para manipulação CRUD como: *save, destroy, create e update*.

A partir do momento que subclasses herdam da classe *Recurso*, automaticamente podem trabalhar com os atributos e operações desta superclasse e, consequentemente, com as funcionalidades da classe `Spira:Base`. Na Figura 16, é apresentado o mapeamento das classes *Recurso* e *Categorias* em ontologias e, na Figura 17, a implementação deste mapeamento utilizando os recursos da biblioteca *Spira*.

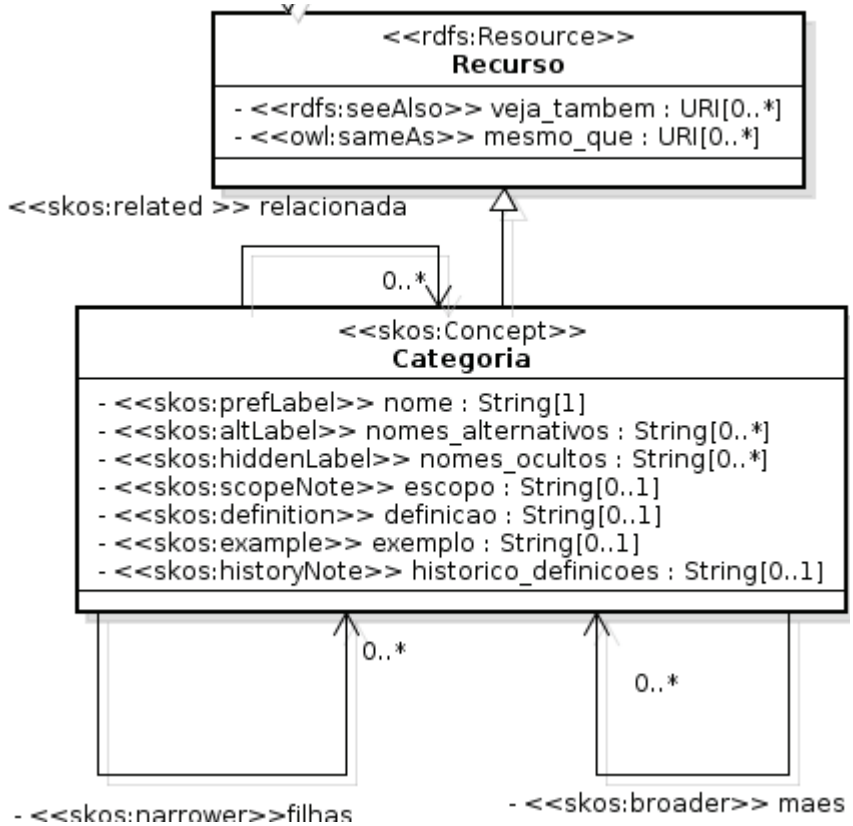

Figura 16: Mapeamento das classes *Recurso* e *Categoria* em ontologias.

```
class Recurso < Spira::Base
        has_many :veja_tambem, :predicate => RDFS.seeAlso, :type => URI
        has_many :mesmo_que, :predicate => OWL.sameAs, :type => URI
end

class Categoria < Recurso
        configure :base_uri => "http://www.prolod.com/tags"
        type RDF::URI.new(SKOS.Concept)
        property :nome, :predicate => SKOS.prefLabel, :type => XSD.string
        property :nomes_alternativos, :predicate => SKOS.altLabel, :type => XSD.string
        property :nomes_ocultos, :predicate => SKOS.hiddenLabel, :type => XSD.string
        property :escopo, :predicate => SKOS.scope, :type => XSD.string
        property :definicao, :predicate => SKOS.definition, :type => XSD.string
        property :exemplo, :predicate => SKOS.example, :type => XSD.string
        property :historico_definicoes, :predicate => SKOS.historyNote,  :type => XSD.string
        has_many :relacionado, :predicate => SKOS.related, :type => :Categoria
        has_many :filhas, :predicate => SKOS.narrower, :type => :Categoria
        has_many :maes, :predicate => SKOS.broader, :type => :Categoria
end
```

Figura 17: Exemplo de implementação da classe *Recurso* e *Categoria* com Spira.

A partir do momento que a classe *Recurso* herda da classe `Spira:Base`, a classe *Recurso* já pode, automaticamente, definir a estrutura que possibilite que os dados sejam mapeados em triplas (sujeito, predicado, objeto), em que o sujeito é a URI da instância da classe em questão, o objeto é o valor inserido na propriedade e o predicado é o mapeamento da propriedade no correspondente predicado da ontologia.

Com o uso do *Spira*, implicitamente temos disponíveis métodos para manipulação CRUD com o modelo RDF. Na Figura 18, é apresentado um exemplo do uso do método "`for`", onde o método recebe uma URI para instanciar um objeto (Recurso) identificado pela URI passada. Logo após, os dados são inseridos nos atributos e para persisti-lo no banco de dados RDF é utilizado o método "*save*". A Figura 19 apresenta a descrição em Turtle após utilizar o SPIRA. Logo após, a Figura 20 demonstra graficamente o mapeamento dos dados com as propriedades das ontologias.

```
categoria = Categoria.for("brasil")
categoria.nome = "brasil"
categoria.escopo = "pais"
categoria.save!
```

Figura 18: Exemplo de instanciação de uma classe do modelo.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://www.example.com/tags/brasil> a skos:Concept ;
    skos:prefLabel "brasil" ;
    skos:scope "pais" .
```

Figura 19: Exemplo da descrição em *Turtle* da instância da classe *Categoria* depois de ser persistido.

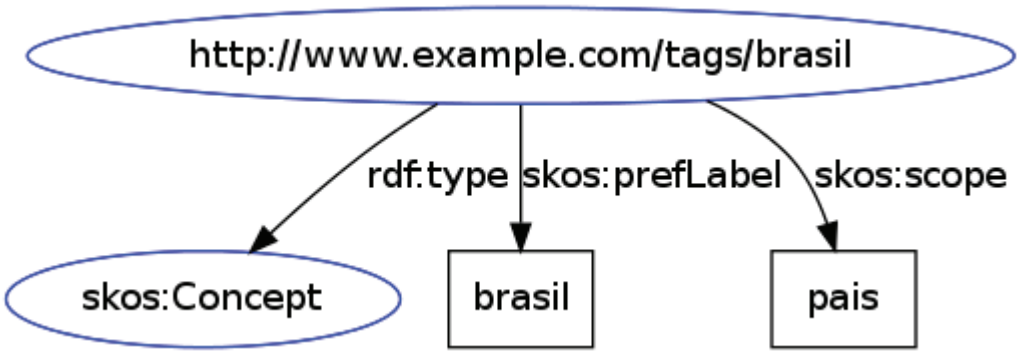

Figura 20: Exemplo do modelo instanciado em grafo.

Também no modelo foram criadas classes responsáveis por fazer o *Linked Data Mashup* dos objetos (recursos) do domínio com outros recursos provenientes da *Web* de Dados.

Manipulador RDF

De forma a manipular os dados em grafos RDF, esta camada fornece os recursos utilizados pelo *Spira* para mapear objeto em triplas do grafo RDF e persistir estas triplas no banco de dados RDF. Para tal, utilizou-se a biblioteca RDF.rb[41] que é a referência para se trabalhar com RDF e *Linked Data* na linguagem *Ruby*. Integrado ao RDF.rb foi utilizada

[41] RDF.rb - http://ruby-rdf.github.io/

também a biblioteca RDF-*Agraph*[42] que é um adaptador para se trabalhar com o banco de dados *AllegroGraph* e RDF.rb. Este adaptador permite a aplicação persistir o grafo RDF construído em memória, bem como realizar consultas SPARQL e inferências no banco de dados da aplicação.

Camada de Banco de Dados RDF

O banco de dados escolhido foi o *Allegrograph*[43], que é um banco de dados nativo RDF (*triple store*), com suporte a transações e alta escalabilidade, criado seguindo os padrões da W3C, oferecendo vários recursos como otimizador de consulta, consultas SPARQL e Inferências (FRANZ, 2005), funcionalidades que muito são utilizadas nesse projeto.

3.6 MASHUP LINKED DATA

Mashup Linked Data é uma forma de integrar dados da aplicação com dados oriundos da *Web* de Dados, reusando, portanto, informação (conhecimento). Para fins de *mashup* da aplicação, foi escolhida a fonte de dados *DBpedia*, por conter uma vasta quantidade de assuntos (*cross-domain*) e por ser a fonte de dados central da *Web* de Dados, estando interligada com diversas outras fontes de dados. Sendo assim, ao fazer o *mashup* diretamente com a *DBpedi*a, também o fazemos indiretamente com outras fontes de dados.

Para realizar o *mashup*, foi utilizada a propriedade `rdfs:seeAlso` da ontologia RDF *Schema*. Na Figura *21* é apresentada a descrição em *Turtle* de um recurso após realizar o *mashup* semântico, de modo que foi adicionada ao grafo a propriedade

---

[42] RDF-*Agraph* - https://github.com/emk/rdf-agraph

[43] Allegrograph - http://franz.com/agraph/allegrograph/

`rdfs:seeAlso` cujo valor é a URI do recurso externo http://dbpedia.org/resource/brazil, da fonte de dados *DBpedia*.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://www.example.com/tags/brasil> a skos:Concept ;
    rdfs:seeAlso <http://dbpedia.org/resource/brazil> ;
    skos:prefLabel "brasil" ;
    skos:scope "pais" .
```

Figura 21: Exemplo em *Turtle* de uma interligação com a Web de dados.

3.6.1 DBpedia Lookup

DBpedia Lookup é um *Web* service que permite realizar consultas na *DBpedia* através de palavras-chave relacionadas. Para isso, na aplicação desenvolvida foi utilizado à biblioteca *DbpediaConceptSearch*[44] onde na classe `CategoriasController` foi criado um método `lookup()`, que recebe o nome da categoria por parâmetro e através da biblioteca mencionada , envia para o *Web Service do DBpedia Lookup* a requisição de busca de palavras-chave relacionadas ao nome da categoria. Seguindo o exemplo da categoria "*brasil*", internamente a biblioteca *DbpediaConceptSearch* faz a seguinte requisição ao *Lookup:* http://lookup.dbpedia.org/api/search.asmx/KeywordSearch?QueryString=brasil".
Conseguindo obter algum resultado, é retornado para a aplicação, um arquivo XML com o *Label*, URI e a descrição dos recursos encontrados a partir das palavras-chave (*Figura 22*). Dentro da aplicação, essa estrutura é acessada e disponibilizada para que o usuário possa selecionar (aprovar) qual recurso está relacionado à categoria. Selecionando

[44] *DbpediaConceptSearch* -https://github.com/reggieb/DbpediaConceptSearch

alguma opção é estabelecido um RDF-*link* entre a categoria e o recurso encontrado, usando a propriedade `rdfs:seeAlso` (veja também) do vocabulário RDF *Schema*, perfazendo assim o *Linked Data mashup* entre a categoria e recursos da *DBpedia* (exemplo apresentado anteriormente na Figura 21).

.

Figura 22: Exemplo de resultado de consulta ao *DBpedia Lookup.*

3.6.2 DBpedia Spotlight

O *DBpedia Spotlight*[45] é uma ferramenta (*Web service*) que utiliza técnicas de processamento de linguagem natural para realizar anotação semântica automática de textos. A ferramenta é capaz de reconhecer e anotar entidades de acordo com o contexto, utilizando técnicas de mineração de texto como *Keyphrase Extraction, Tagging e Named Entity Recognition* (MENDES et al., 2011).

Na aplicação implementada foi criada a classe `Spot`, que ao instanciar a mesma

[45] *DBpedia Spotlight* - https://github.com/dbpedia-spotlight/dbpedia-spotlight/

deve-se passar como o parâmetro a descrição do projeto. Esta descrição é enviada para o *Spotlight* por meio do método `initialize()` via a URL "http://spotlight.dbpedia.org/rest/annotate/?text=DESCRICAO", na qual é realizada a anotação semântica e são extraídas entidades do texto, sendo retornada uma URI da *DBpedia* para cada entidade encontrada. Após obter as URIs das entidades encontradas, é chamado o método `dereferenc()` que dereferencia a URI e obtém a representação RDF da entidades. Por fim, é criada uma tabela *Hash* que mapeia a URI no valor da propriedade `rdfs:comment` obtida na dereferenciação, para que sejam apresentadas ao usuário as descrições dos recursos (entidades) para que o mesmo possa selecionar as que estejam relacionadas ao seu projeto. Caso o usuário selecione uma ou mais opções é estabelecido um RDF-*link* entre o projeto e o recurso selecioando, usando a propriedade `rdfs:seeAlso` (veja também) do vocabulário RDF *Schema*, estabelecendo assim o *Linked Data mashup* entre a o projeto e recursos da *DBpedia.*

3.7 CONSULTAS SPARQL E INFERÊNCIA

Conforme já mencionado anteriormente, um dos objetivos desse trabalho é publicar dados de projeto de *software* em RDF e ofertar recuperação desses dados de forma eficiente. Para tal, a aplicação faz-se uso de consultas SPARQL junto com inferências.

A escolha do banco de dados *Allegrograph* foi determinante para poder utilizar SPARQL e inferências, visto que mesmo oferece um *SPARQL endpoint* (*Web service* para consultas) e um raciocinador com suporte a alguns axiomas OWL.

Para trabalhar com a comunicação com o banco de dados RDF em *Ruby*, foi utilizada a biblioteca RDF-*agraph*[46] que é integrada com o RDF.rb. Através dessa biblioteca é possível inserir as consultas SPARQL dentro do código *Ruby*, de modo a se

[46] RDF-*agraph* - https://github.com/emk/rdf-agraph

comunicar com o banco de dados e obter os resultados das consultas. Para realizar uma consulta utiliza-se o método `sparql_query(consulta, inferência)` da classe *`AbstractRepository`* da biblioteca mencionada. Como parâmetro é passada a consulta em Sparql e se for utilizar inferência é passado o parâmetro "`infer`" com o valor *`true`* para que seja ativada a mesma.

3.7.1 Consultas SPARQL oferecidas.

Para obter resultados expressivos e eficientes na recuperação de dados na *Web* Semântica, a linguagem SPARQL é fundamental. Neste trabalho foram implementados dois tipos de consultas SPARQL.

A primeira consulta tem o objetivo buscar projetos de *Software* que têm alguma relação com uma determinada palavra-chave. Para isso foi criada a consulta em SPARQL apresentado na Figura 23.

```
prefix doap: <http://usefulinc.com/ns/doap#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>

select distinct ?project ?name ?desc
where
{
    ?project a doap:Project .
    ?project  doap:description ?desc .
    ?project  doap:name ?name .
    {
        {
            ?project rdfs:label ?projectLabel .
            filter regex(str(?projectLabel), "palavra_chave", "i" ).
        }
        union
        {
            ?project ?property ?object .
            ?object rdfs:label ?projectLabel .
            filter regex(str(?projectLabel), "palavra_chave", "i" ).
        }
        union
        {
           ?project rdfs:comment ?projectComment .
           filter regex(str(?projectComment), "palavra_chave", "i" ).
        }
        union
        {
           ?project ?property ?object .
           ?object rdfs:comment ?projectComment .
           filter regex(str(?projectComment), "palavra_chave", "i" ).
        }
        union
        {
           ?project doap:description ?desc .
           filter regex(str(?desc), "palavra_chave", "i" ).
        }
        union
        {
            ?project doap:name ?name .
            filter regex(str(?name), palavra_chave", "i" ).
        }
    }

}
```

Figura 23: Busca por projetos por determinada palavra-chave.

A consulta retorna a URI, o nome e descrição dos projetos que possuam a palavra-chave, ignorando-se diferença entre maiúscula e minúscula, no seu nome (`doap:name`) ou na sua descrição (`doap:description`) ou no seu rótulo (`rdfs:label`) ou no seu comentário (`rdfs:comment`) ou ainda que possuam algum outro recurso associado que, por sua vez, possua a palavra-chave no seu rótulo (`rdfs:label`) ou no seu comentário (`rdfs:comment`).

Ainda na mesma consulta se ativarmos o mecanismo de inferência do processador de consultas, teremos resultados mais abrangentes. Conforme apresentado no mapeamento do modelo em ontologias, o atributo *nome* da classe *Projeto* é mapeado na ontologia do DOAP pela propriedade `doap:name`, analisando de forma mais precisa a descrição dessa propriedade na ontologia DOAP, vemos que a mesma possui uma

relação de subpropriedade (`rdfs:subPropertyOf`) com a propriedade `rdfs:label`, conforme apresentado na Figura 24 . O mesmo ocorre para os atributos *nome* e *nomes_ocultos* da classe *Categoria* que são mapeados respectivamente com `skos:prefLabel` e `skos:hiddenLabel` e o atributo *nome* da classe *Pessoa* que é mapeado com `foaf:name`. Originalmente a propriedade `doap:description` não possui relacionamento com `rdfs:comment`, sendo que, neste trabalho foi criada uma extensão na qual `doap.description` passa a ser subpropriedade (`rdfs:subPropertyOf`) de `rdfs:comment`, enriquecendo as relações entre as ontologias DOAP e RDFS.

```
<rdf:Property rdf:about="http://usefulinc.com/ns/doap#name">
        <rdfs:isDefinedBy rdf:resource="http://usefulinc.com/ns/doap#" />
        <rdfs:label xml:lang="en">name</rdfs:label>
        <rdfs:label xml:lang="fr">nom</rdfs:label>
        <rdfs:label xml:lang="es">nombre</rdfs:label>
        <rdfs:label xml:lang="de">Name</rdfs:label>
        <rdfs:label xml:lang="cs">jméno</rdfs:label>
        <rdfs:label xml:lang="ja">名前</rdfs:label>
        <rdfs:comment xml:lang="en">A name of something.</rdfs:comment>
        <rdfs:comment xml:lang="fr">Le nom de quelque chose.</rdfs:comment>
        <rdfs:comment xml:lang="es">El nombre de algo.</rdfs:comment>
        <rdfs:comment xml:lang="de">Der Name von Irgendwas</rdfs:comment>
        <rdfs:comment xml:lang="cs">Jméno něčeho.</rdfs:comment>
        <rdfs:comment xml:lang="ja">何かの名前</rdfs:comment>
        <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal" />
        <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#label" />
</rdf:Property>
```

Figura 24: Exemplo em que o `doap:name` é uma subpropriedade de `rdfs:label`.

Tomemos como exemplo a propriedade `doap:name`. Em virtude de ser subpropriedade de rdfs:label, o raciocinador infere para cada tripla com predicado `doap:name` outra tripla com os mesmos sujeito e objeto, mas com predicado `rdfs:label`. De forma similar, ocorre para todas as outras subpropriedades de rdfs:label e `rdfs:comment`.Sendo assim, ao executar a consulta acima com inferência ativada, amplia-se o alcance da consulta, visto que todos os recursos associados ao projeto (*Pessoa*, *Categoria*, *Organização*, *Especificação*, *Repositório* e *Versão*) passarão a ter pelo menos uma das propriedades rdfs:label ou rdfs:comment por inferência. No capítulo 4 é apresentado um exemplo do que ocorre ao ativar o raciocinador.

A segunda consulta oferecida é a busca de projetos por uma determinada categoria (*tag*) relacionada (Figura 25). Essa consulta é realizada utilizando a URI da categoria especificada e verificando por meio da propriedade `doap:category` qual projeto está relacionado. Na mesma consulta, se ativarmos a inferência, podemos obter mais projetos, utilizando o relacionamento *filhas* (`skos:narrower`) da classe *Categoria*. Nesse caso, é possível obter um projeto que esteja relacionado indiretamente a uma categoria, por meio de suas categorias descendentes. Seja o exemplo de hierarquia de categorias da Figura 26. A categoria *SemanticWeb* possui como filha (`skos:narrower`) a categoria *Linked Data*, que, por sua vez, possui como filhas (skos:narrower) a categoria *Ontology*. Se executarmos a consulta sem inferência para categoria *SemanticWeb*, obteremos apenas o projeto *Alpha* que está diretamente associado a categoria especificada. No entanto, se for ativada a inferência e realizarmos a mesma consulta para a categoria *SemanticWeb*, obteremos os projetos Alpha, bem como os projetos Beta e Gamma, associados as suas categorias filha e neta, respectivamente. Isto ocorre porque na especificação da ontologia SKOS, a propriedade `skos:narrower` é subpropriedade (`rdfs:subPropertyOf`) de `skos:narrowTransitive` e esta, por sua vez, é uma propriedade transitiva (`owl:TransitiveProperty`). Sendo assim, a partir da propriedade filha (`skos:narrower`), a máquina infere, por conta própria, os relacionamentos `skos:narrowTransitive` entre a categoria *SemanticWeb* e todas as suas descendentes, conforme ilustram as arestas pontilhadas em vermelho da Figura 26, e, por conseguinte, a consulta passa a retornar todos os projetos que estejam direta ou indiretamente ligados a categoria *SemanticWeb*.

```
prefix doap: <http://usefulinc.com/ns/doap#>
prefix skos: <http://www.w3.org/2004/02/skos/core#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>


select distinct ?project ?projectName ?projectComment
where
{

    {
      <URI_da_Categoriaa> a skos:Concept.
      ?project doap:category  <URI_da_Categoria>.
      ?project a doap:Project
    }
       union
    {
      <URI_da_Categoria> skos:narrowerTransitive ?desc .
      ?project doap:category     ?desc.
      ?project a doap:Project
    }
    optional {?project doap:description ?projectComment}
    optional {?project doap:name ?projectName}


}
```

Figura 25: Consulta SPARQL de projetos por categoria.

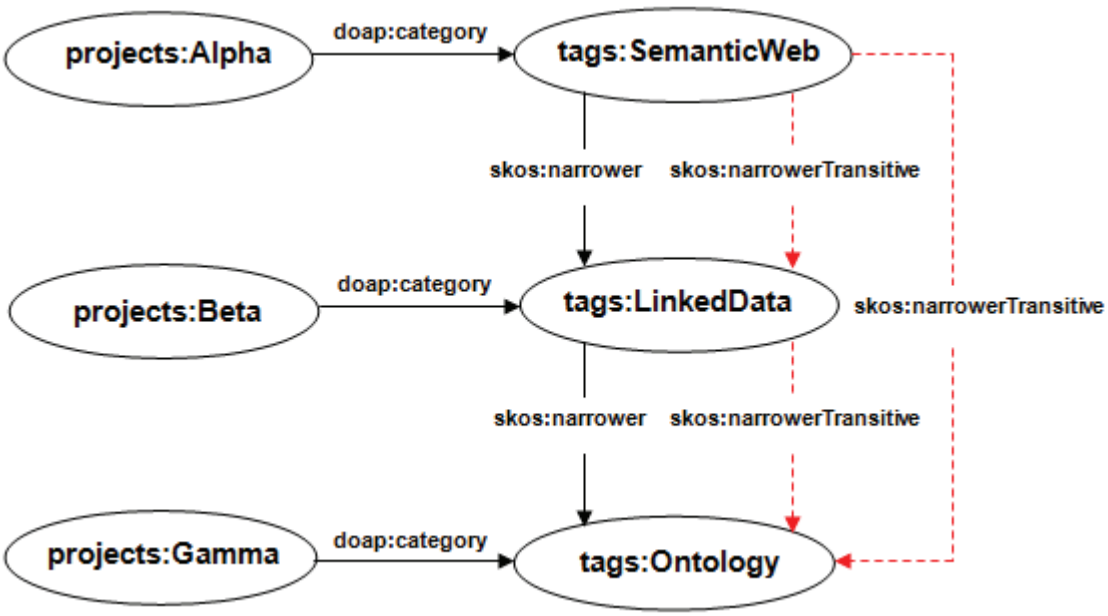

Figura 26: Exemplo de hierarquia de categorias e seus projetos.

3.7.2 Inferências utilizadas

O banco de dados *Allegrograph* permite trabalhar com inferências com os seguintes predicados:

- `rdf:type` que tem a função de declarar que determinado recurso pertence a determinado tipo.
- `rdfs:domain` é usada para indicar que uma propriedade em particular se aplica a uma determinada classe.
- `rdfs:range` é usada para o valor de uma certa propriedade é de uma determinada classe.
- `rdfs:subClassOf` é utilizado para definir a relação em que uma classe é subclasse de outra, e em que as subclasses são transitivas. Exemplo: `(motocicleta, rdfs:subClassOf, veículo)` e `(off-road, rdfs:subClassOf, motocicleta)`, com transitividade chega-se a `(off-road, rdfs:subClassOf, Veículo)`.
- `rdfs:subPropertyOf` possui o conceito similar ao da subclasse, onde é uma especialização de uma propriedade. Exemplo: se propriedade P1 é subpropriedade de P2 e A está relacionado com B por P1 então A também está relacionado com B por P2.
- `owl:inverseOf` é utilizada para definir que uma propriedade é inversa de outra propriedade. Exemplo: se filho é o inverso de pai , então, dado que X é filho de Y, implica que Y é Pai de X.
- `owl:sameAs`_é utilizado para definir que duas URIS distintas identificam a mesma entidade.
- `owl:TransitiveProperty` permite realizar deduções transitivas. Exemplo: se propriedade P1 é transitiva e A está relacionado a B por P1, B está relacionado a C por P1, então A também está relacionado a C por P1.

Neste trabalho, as consultas com inferência trabalham com os seguintes predicados: `rdfs:subPropertyof`, `owl:TransitiveProperty`, `owl:sameAs`.

Para trabalhar com Inferências, foi necessário importar os arquivos fonte das ontologias utilizadas na base de dados da aplicação. Portanto, foram importadas as ontologias DOAP, FOAF, OWL, RDF-SCHEMA, VCARD, SKOS E RDF. O banco de dados *AllegroGraph* possui em sua inferface de administração a opção "*from an upload file"* que permite importar um arquivo em RDF. A Figura 27 ilustra a importação do arquivo da ontologia DOAP.

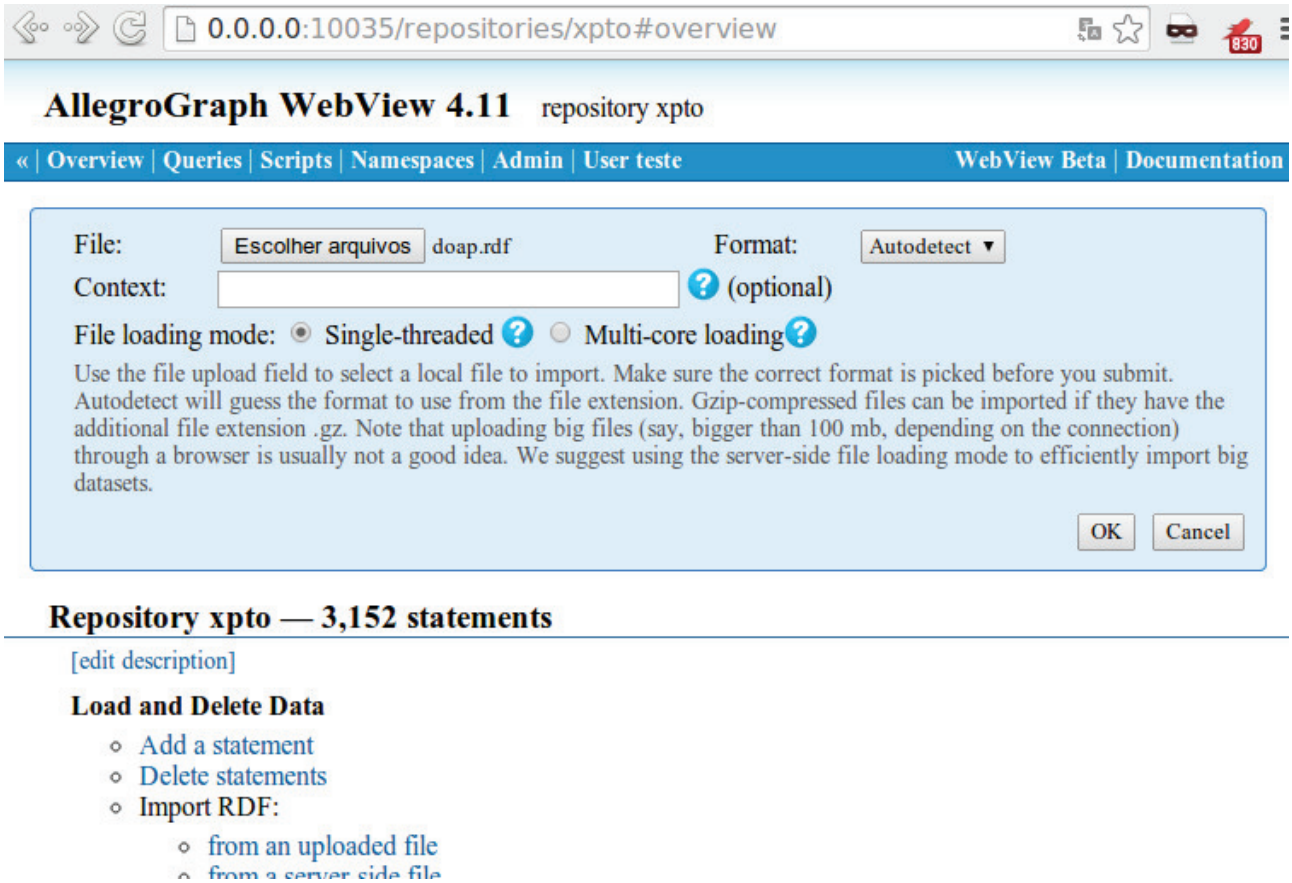

Figura 27: Importação da ontologia DOAP no banco de dados da aplicação.

4 ESTUDO DE CASO

Neste capítulo, é apresentado um estudo de caso realista, para exemplificar o uso do protótipo. Os recursos de exemplo foram baseados projetos reais extraídos do repositório *Freecode.*

4.1 CASO DE USO

Para realização deste estudo de caso, foi realizado o cadastro de projetos de *software* a fim de apresentar o funcionamento do protótipo e, especialmente, o *mashup Linked Data.*

De forma a apresentar os casos de uso do protótipo, as figuras a seguir apresentam o cadastro de projeto, organização, categoria e pessoa.

Vale ressaltar que para o usuário final é transparente o uso da *Web* Semântica, visto que todas as idiossincrasias desta nova tecnologia estão encapsuladas no interior da aplicação, contribuindo para o enriquecimento da mesma, sem que o usuário perceba.

Os cadastros foram feitos basicamente com a estrutura padrão do *Ruby on Rails*. Para cadastrar dados dos principais conceitos como projeto, pessoa, categoria, organização foram utilizados formulários independentes.

A tela inicial do protótipo disponibiliza as opções de cadastro de projetos e entidades relacionadas, como também a busca de projetos por palavra-chave e por categoria. A Figura 28 apresenta o menu principal do protótipo. Para este exemplo foi utilizado o domínio fictício http://www.prolod.com, que serve com base (*namespace*) para todas as URIs geradas.

Figura 28: Página inicial da aplicação.

CADASTRO DE PESSOA

Primeiro são listadas as *Pessoas* cadastradas e as opções de cadastrar nova *Pessoa*, editar *Pessoa*, excluir *Pessoa* e exibir *Pessoa* (Figura 29). O formulário de criação e edição de *Pessoa* contém os campos para cadastrar os atributos de *Pessoa* e as relações com *Telefone* e *Endereço*. Na Figura 30 é apresentado o cadastro de uma Pessoa.

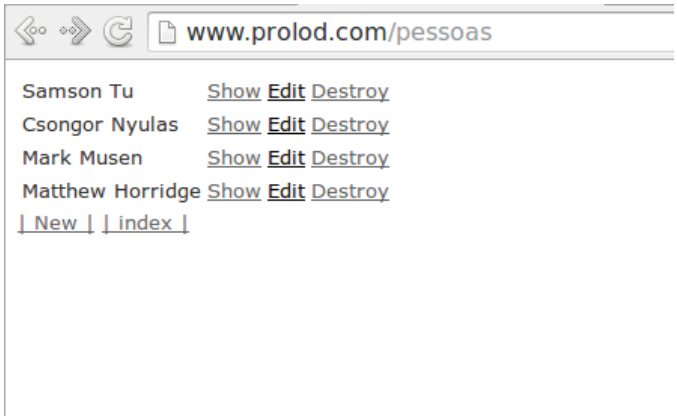

Figura 29: Índice do CRUD de pessoa.

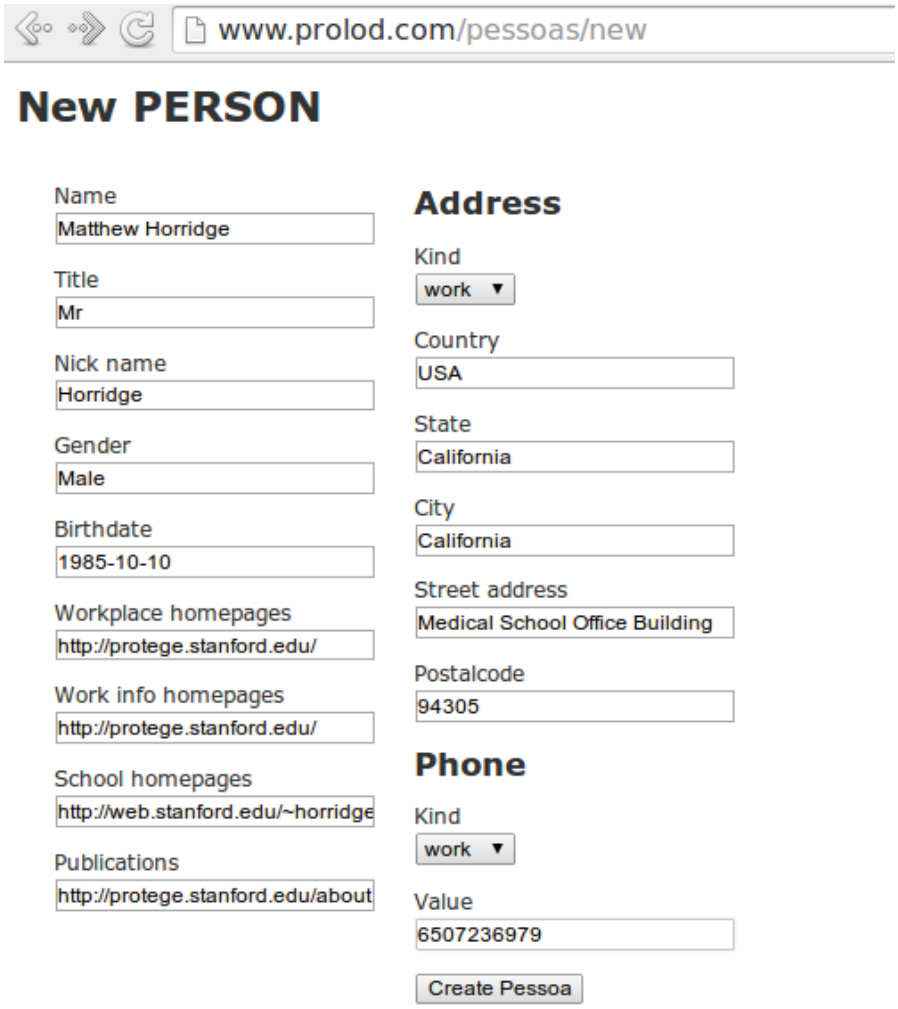

Figura 30: Formulário de cadastro de uma pessoa.

Após inserir os dados e confirmar a criação, direciona-se para a confirmação do cadastro e informando os dados cadastrados (Figura 31).

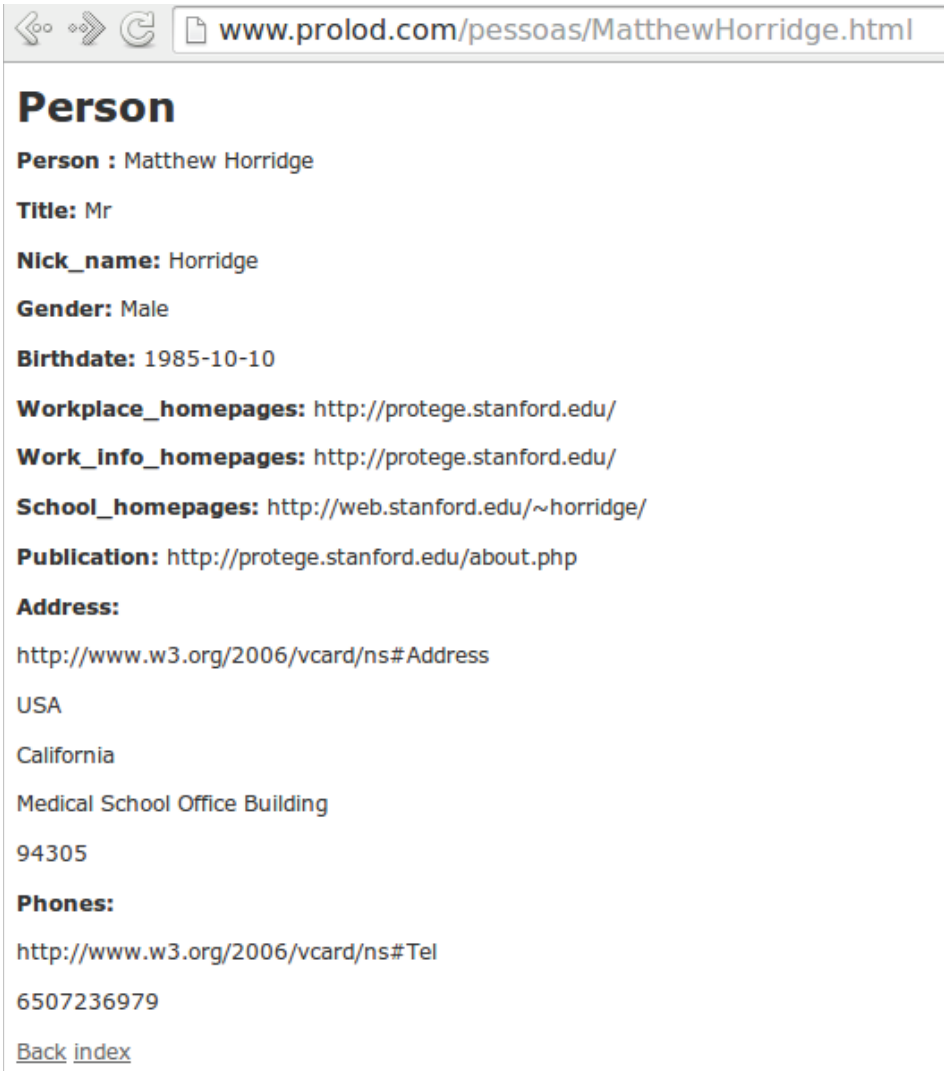

Figura 31: Dados cadastrados de uma pessoa.

Até o momento, o cadastro e exibição aparentam ser iguais ao de uma aplicação convencional, mas não o são. Quando for requisitada via URI uma representação do recurso no formato RDF, a aplicação executará a negociação de conteúdo e devolverá a descrição do recurso em RDF/XML. Para gerar a declaração RDF/XML, a aplicação executa no método `show()` da classe *PessoasController*, uma consulta *SPARQL*

Describe, por meio de interpolação de *string* de *Ruby,* retornando a representação *RDF* do Recurso. Segue abaixo o trecho de código Ruby da consulta mencionada.

```
("describe  #{"<"+ @pessoa.to_uri+">"}  ?x
WHERE
{
      {
             #{"<"+ @pessoa.to_uri+">"} #{"<"+VCARD.adr+">"} ?x
      }
      union
      {        #{"<"+ @pessoa.to_uri+">"} #{"<"+VCARD.tel+">"} ?x
      }
}")
```

Através da biblioteca *Agraph*, responsável por fazer a comunicação com o banco de dados *AllegroGrpah*, a consulta anterior é submetida ao *SPARQL endpoint* do banco de dados onde é executada. Na Figura 32 é apresentada a representação RDF da pessoa cadastrada em *Turtle.*

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix vcard: <http://www.w3.org/2006/vcard/ns#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://www.prolod.com/pessoas/MatthewHorridge> a foaf:Person ;
    vcard:adr <http://www.prolod.com/address/g70320798445800> ;
    vcard:bday "1985-10-10"^^xsd:date ;
    vcard:tel <http://www.prolod.com/phones/g70320660689540> ;
    foaf:gender "Male" ;
    foaf:name "Matthew Horridge" ;
    foaf:nick "Horridge" ;
    foaf:publications <http://protege.stanford.edu/about.php> ;
    foaf:schoolHomepage <http://web.stanford.edu/~horridge/> ;
    foaf:title "Mr" ;
    foaf:workInfoHomepage <http://protege.stanford.edu/> ;
    foaf:workplaceHomepage <http://protege.stanford.edu/> .

<http://www.prolod.com/address/g70320798445800> a <http://www.w3.org/2006/vcard/Work>,
        vcard:Address ;
    vcard:country_name "USA" ;
    vcard:locality "Medical School Office Building" ;
    vcard:postal_code "94305" ;
    vcard:region "California" .

<http://www.prolod.com/phones/g70320660689540> a <http://www.w3.org/2006/vcard/Work>,
        vcard:Tel ;
    rdf:value "6507236979" .
```

Figura 32: Resultado da representação RDF de uma pessoa cadastrada em *Turtle.*

Cadastro de Organização

O cadastro de *Organização* segue o mesmo padrão do cadastro de *Pessoa*, sendo que o tipo e atributos são diferentes. Figura 33, Figura 34, Figura 35 apresentam, respectivamente, o formulário de cadastro, a página HTML de exibição e a descrição do recurso em *Turtle*.

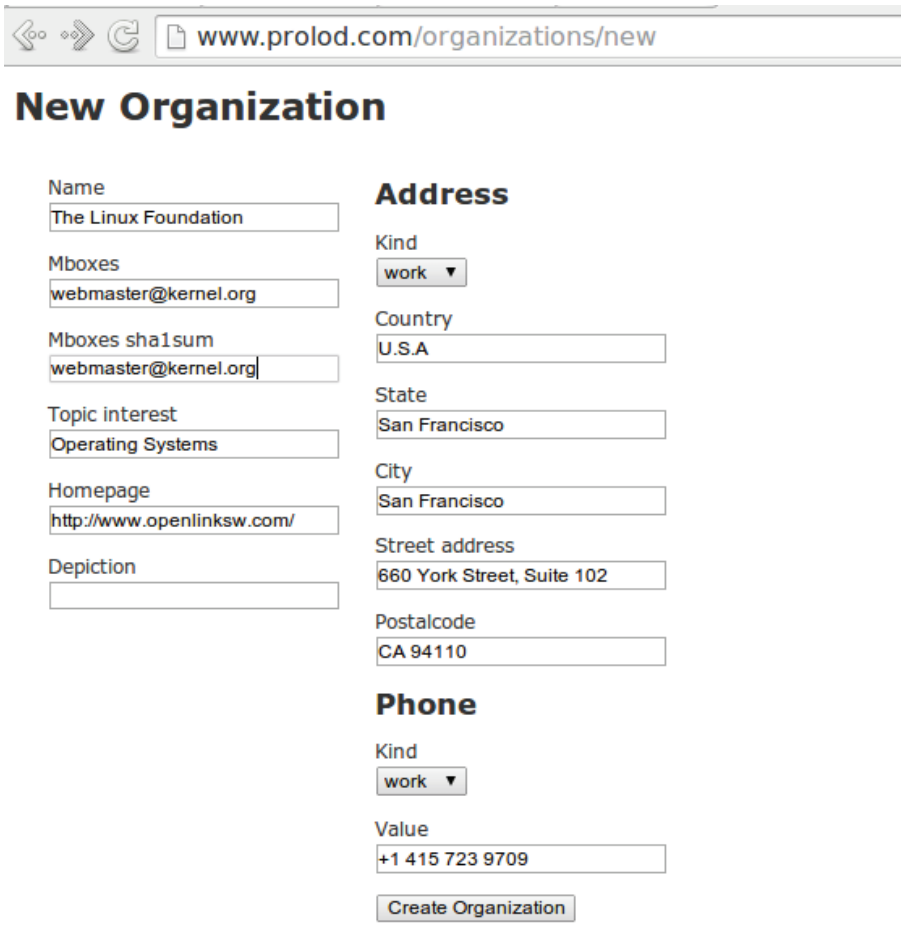

Figura 33: Formulário de cadastro de uma organização.

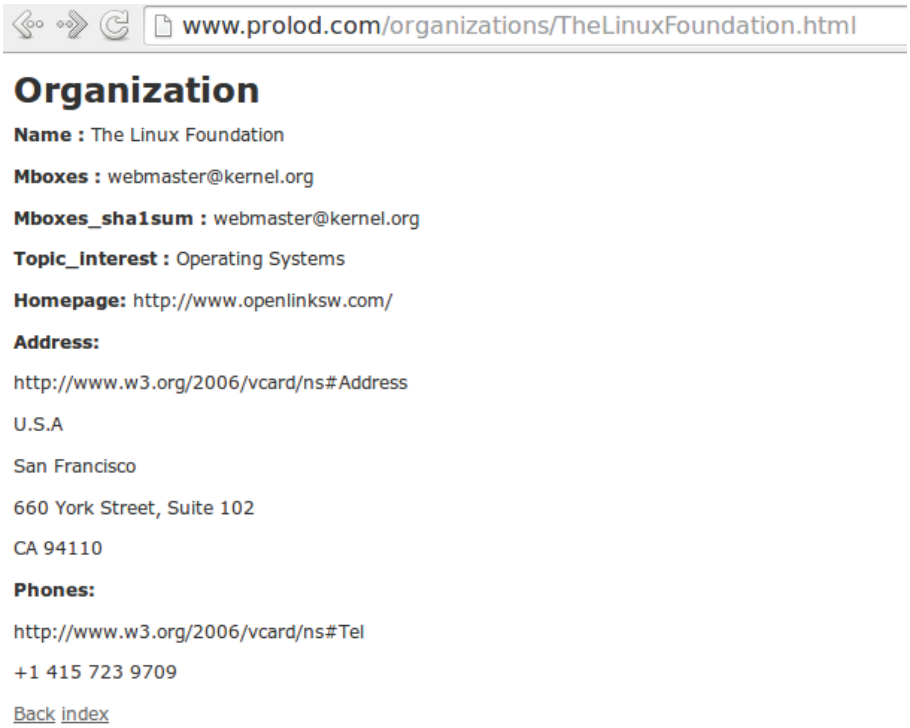

Figura 34: Dados cadastrados de uma pessoa.

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix vcard: <http://www.w3.org/2006/vcard/ns#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://www.prolod.com/organizations/TheLinuxFoundation> a foaf:Organization ;
    vcard:adr <http://www.prolod.com/address/g70320698605540> ;
    vcard:tel <http://www.prolod.com/phones/g70320694274640> ;
    foaf:homepage <http://www.openlinksw.com/> ;
    foaf:mbox <webmaster@kernel.org> ;
    foaf:mbox_shalsum "webmaster@kernel.org" ;
    foaf:name "The Linux Foundation" ;
    foaf:topic_insterest "Operating Systems" .

<http://www.prolod.com/address/g70320698605540> a <http://www.w3.org/2006/vcard/Work>,
        vcard:Address ;
    vcard:country_name "U.S.A" ;
    vcard:locality "660 York Street, Suite 102" ;
    vcard:postal_code "CA 94110" ;
    vcard:region "San Francisco" .

<http://www.prolod.com/phones/g70320694274640> a <http://www.w3.org/2006/vcard/Work>,
        vcard:Tel ;
    rdf:value "+1 415 723 9709" .
```

Figura 35: Resultado da representação RDF de uma pessoa cadastrada em *Turtle*.

Cadastro de Categoria

No cadastro de *Categoria* foi implementado uma parte do *mashup Linked Data* com a *DBpedia*. A página de cadastro de Categoria possui funcionalidade similar aos cadastro anteriores, porém ao submeter é realizado o *mashup*.

A página do cadastro de uma *Categoria* permite inserir dados, bem como estabelecer relacionamentos hierárquicos (verticais) e de outra natureza (horizontais) entre *Categorias*, como o exemplo apresentado na Figura 36, onde foi realizado o cadastrado da categoria *Linked Data*, criando uma relação de filha em relação à categoria *Semantic Web*.

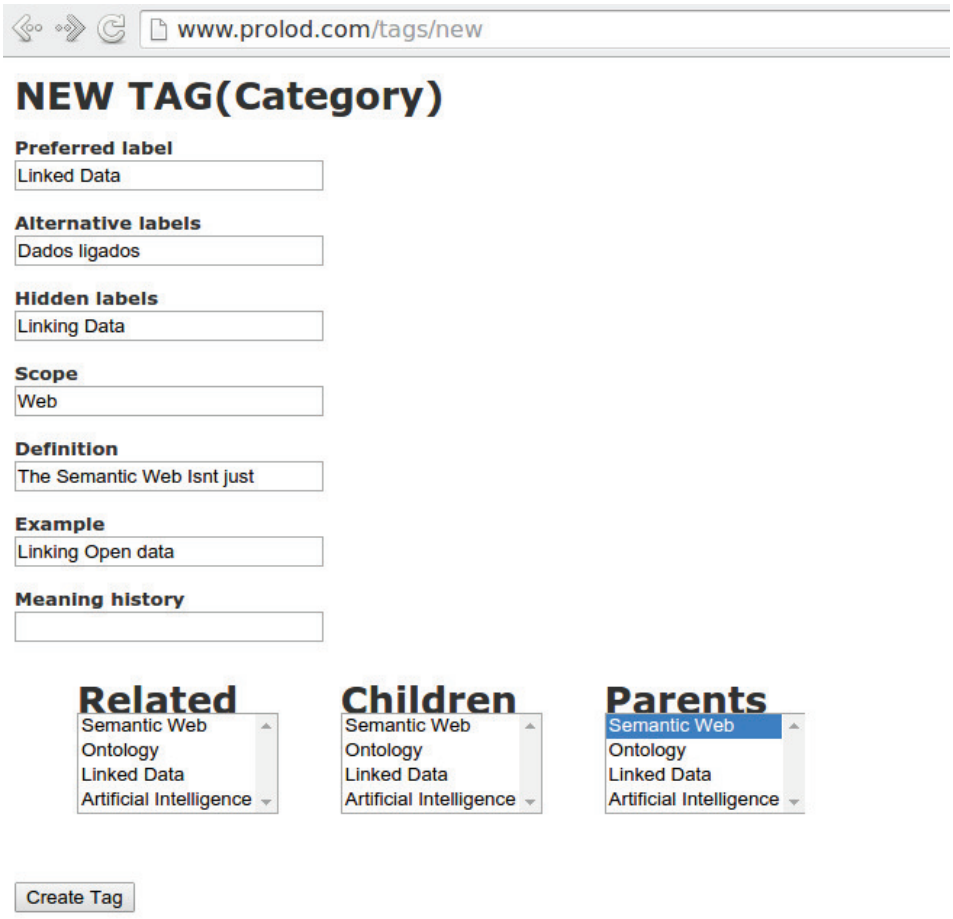

Figura 36: Cadastro de uma categoria.

.

Após confirmar a criação da *Categoria*, a aplicação realiza um processamento em cima do valor cadastrado no atributo nome, submetendo-o para o *Web service* do *DBpedia Lookup,* onde é realizada uma busca por palavra-chave na *DBpedia* e retorna para a aplicação os recursos (URIs), seus rótulos (rdfs:label) e suas descrições (rdfs:comment). Feito isto, a aplicação trata esse resultado e apresenta para o usuário avaliar se algum destes recursos têm algum tipo de relação com a *Categoria* criada. Os recursos selecionados são associados à *Categoria* criada por meio da propriedade `rdfs:seeAlso`, perfazendo o *mashup*. A Figura 37 demonstra a página que é apresentada ao usuário propondo sugestões de recursos da *DBpedia* que podem estar relacionados à *Categoria* em questão.

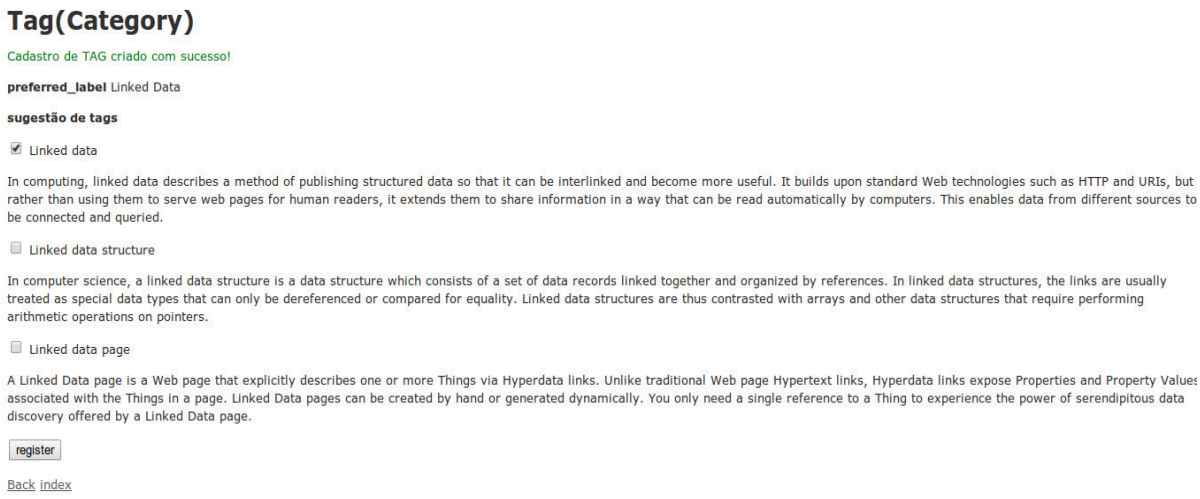

Figura 37: Seleção de recursos a *DBpedia* relacionada a uma categoria.

Após relacionar e confirmar os recursos que possuem alguma relação com a categoria, é apresentada a página com os dados cadastrados. Figura 38 e Figura 39 apresentam os resultados do cadastro em HTML e em *Turtle*.

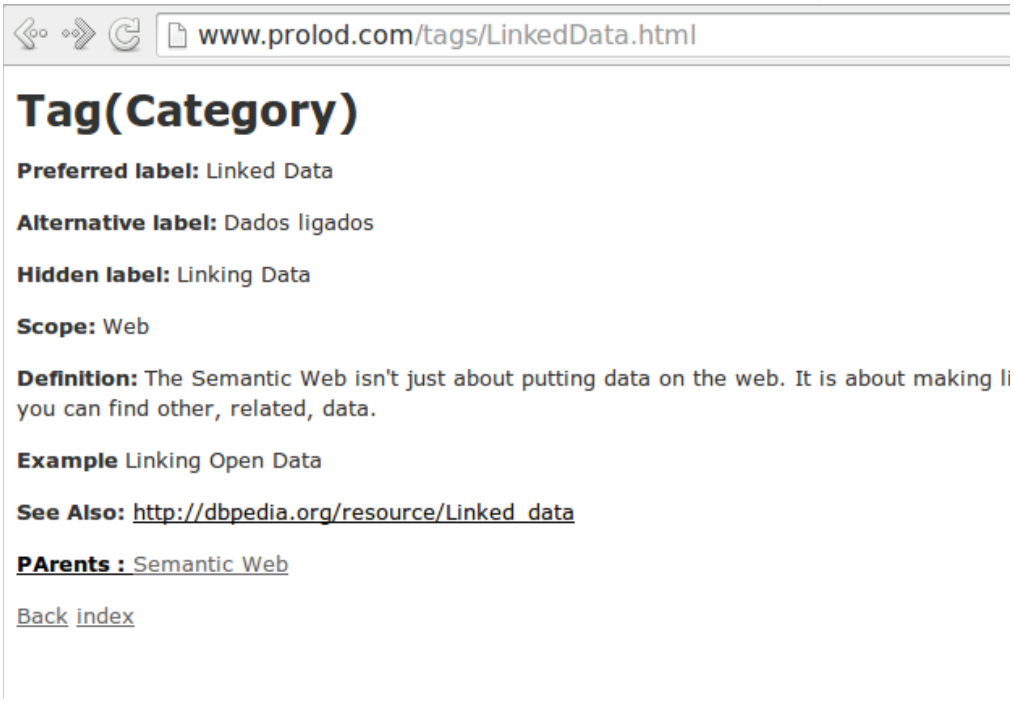

Figura 38: Página de confirmação dos dados cadastrados da categoria.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://www.prolod.com/tags/LinkedData> a skos:Concept ;
    rdfs:seeAlso <http://dbpedia.org/resource/Linked_data> ;
    skos:altLabel "Dados ligados" ;
    skos:broader <http://www.prolod.com/tags/SemanticWeb> ;
    skos:definition "The Semantic Web isn't just about putting data
    skos:example "Linking Open Data" ;
    skos:hiddenLabel "Linking Data" ;
    skos:prefLabel "Linked Data" ;
    skos:scope "Web" .
```

Figura 39: Descrição em *Turtle* de uma categoria.

Cadastro de Projeto

O cadastro de projeto, que é um dos objetivos centrais do trabalho possui a página apresentada na *Figura 40*.

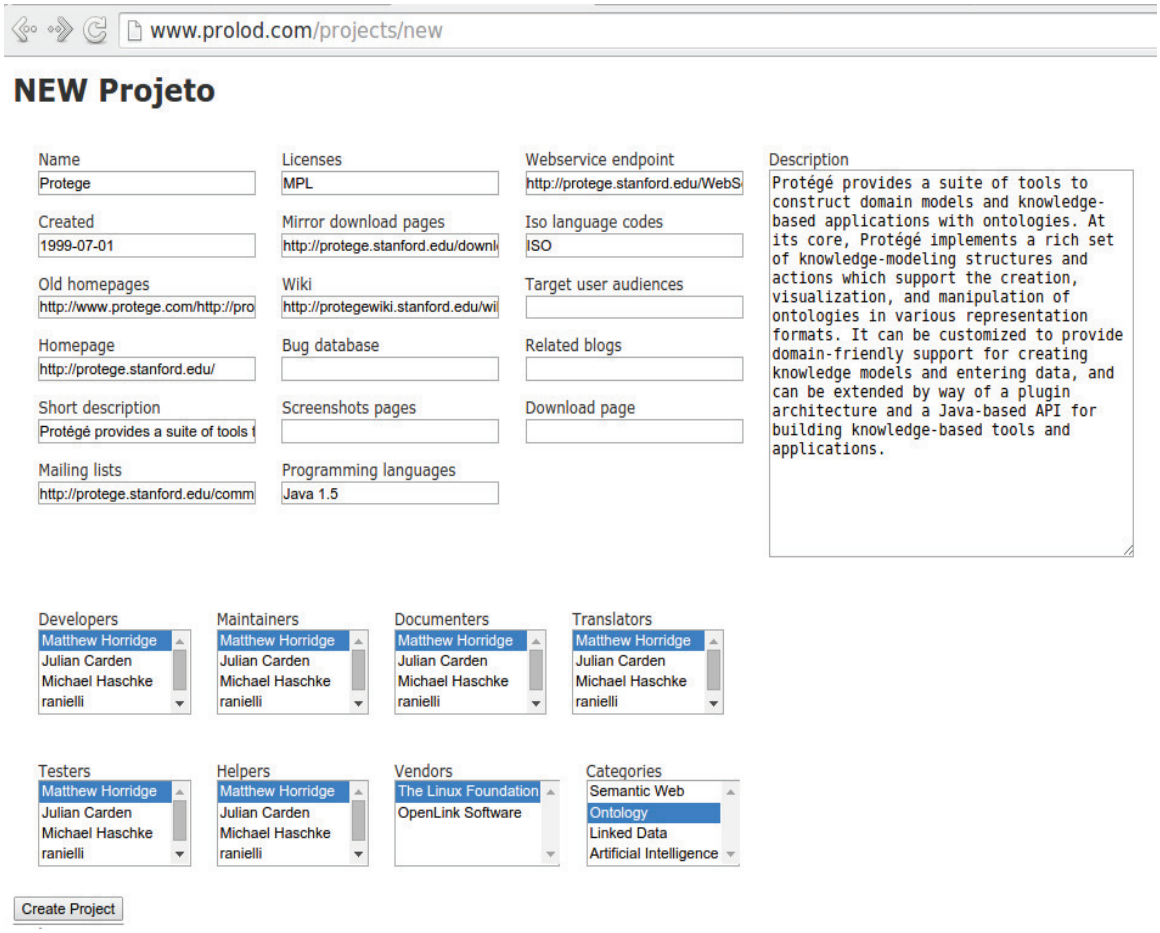

Figura 40: Página de cadastro de um projeto.

O após confirmar os dados e criar o projeto, a aplicação realiza um processamento interno utilizando o valor da propriedade descrição. Este procedimento consiste em submeter o texto para o *DBpedia Spotlight*, de modo a realizar um processamento por meio de técnicas de linguagem natural e extrair automaticamente entidades (recursos) relacionadas à *DBpedia* com as devidas URIs. Após receber as URIs das entidades encontradas, a aplicação dereferencia cada recurso via URI e obtém a descrição RDF do mesmos. Outro processamento é feito internamente, em que é selecionado o valor da propriedade `rdfs:comment` para que sejam apresentadas para o usuário as descrições

das entidades extraídas. Na Figura 41 é apresentada a página com as descrições dos recursos para que o usuário possa aprovar qual deles está relacionada ao projeto em questão. Cada recurso aprovado é associado ao projeto por meio da propriedade `rdfs:seeAlso`, realizando assim outro tipo de *mashup* com a DBpedia. A Figura 42 de confirmação de cadastro do *Projeto* e a Figura 43 apresenta a descrição do mesmo em *Turtle*.

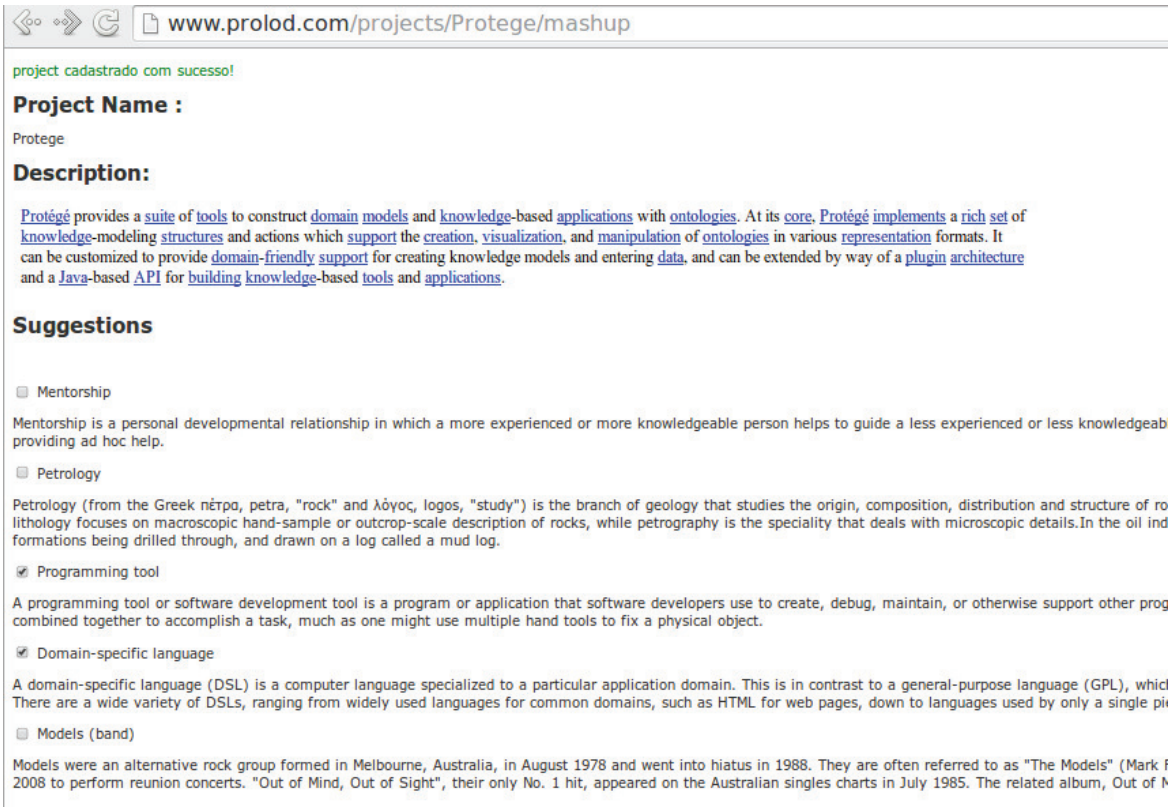

Figura 41: *Seleção de recursos da DBpedia relacionados a um projeto.*

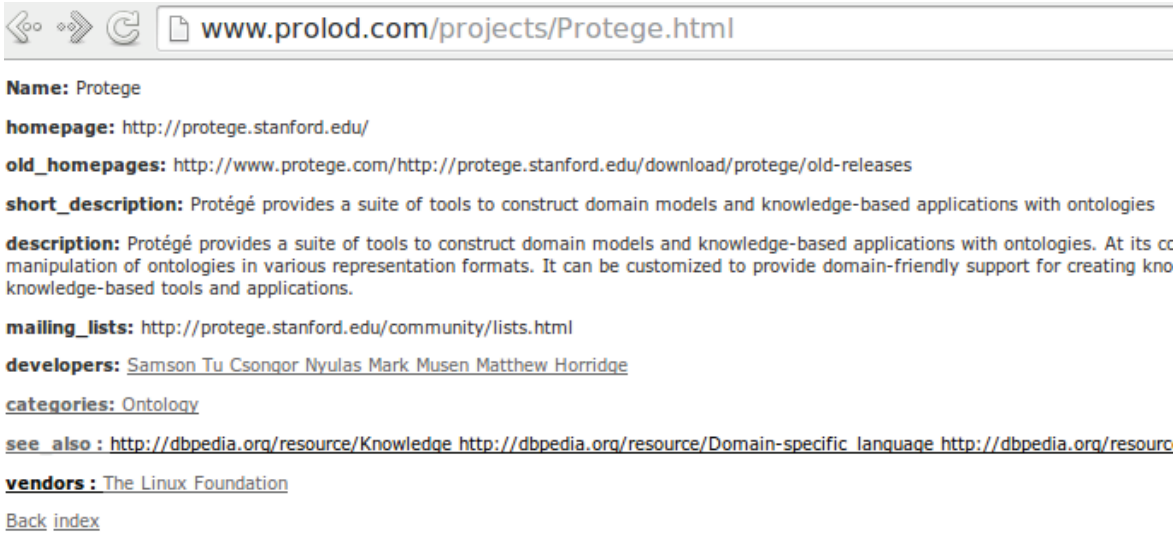

Figura 42: Cadastro final de um projeto.

```
@prefix doap: <http://usefulinc.com/ns/doap#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://www.prolod.com/projects/Protege> a doap:Project ;
    doap:category <http://www.prolod.com/tags/Ontology> ;
    doap:created "1999-07-01"^^xsd:date ;
    doap:description "Protégé provides a suite of tools to construct domain models and knowledge-based applications with
    doap:developer <http://www.prolod.com/pessoas/CsongorNyulas>,
        <http://www.prolod.com/pessoas/MarkMusen>,
        <http://www.prolod.com/pessoas/MatthewHorridge>,
        <http://www.prolod.com/pessoas/SamsonTu> ;
    doap:download_mirror "http://protege.stanford.edu/download/download.html" ;
    doap:download_wiki "http://protegewiki.stanford.edu/wiki/ProtegeProject_FAQ" ;
    doap:homepage <http://protege.stanford.edu/> ;
    doap:language "ISO " ;
    doap:license "MPL" ;
    doap:mailing_list "http://protege.stanford.edu/community/lists.html" ;
    doap:name "Protege" ;
    doap:old_homepage <http://www.protege.com/http://protege.stanford.edu/download/protege/old-releases> ;
    doap:programminglanguage "Java 1.5" ;
    doap:service_endpoint "http://protege.stanford.edu/WebService" ;
    doap:shortdesc "Protégé provides a suite of tools to construct domain models and knowledge-based applications with on
    doap:vendor <http://www.prolod.com/organizations/TheLinuxFoundation> ;
    rdfs:seeAlso <http://dbpedia.org/resource/Domain-specific_language>,
        <http://dbpedia.org/resource/Knowledge>,
        <http://dbpedia.org/resource/Programming_tool> .
```

Figura 43: Cadastro final de um projeto em *Turtle*.

Consultas e Inferências

Como resultado e benefício de aplicar *Web* Semântica neste projeto, temos consultas com resultados bem expressivos. Foram criadas duas opções de busca de *projetos*, sendo uma por meio de palavras-chave e outra com projetos relacionados a uma determinada categoria.

Para buscar um projeto via palavras-chave temos o formulário apresentando na Figura 44. Internamente utiliza-se a linguagem de consulta SPARQL. Ao realizar uma busca por palavra-chave, basicamente a busca é feita em cima das propriedades `doap:name` e `doap:description` dos projetos, o que limita um pouco a possibilidade de resultados, ficando restritas somente a estas duas propriedades diretamente relacionadas ao tipo (classe) *Projeto*. No mesmo formulário, é disponibilizada a opção de utilizar inferência. Selecionando a opção *true* esse recurso é ativado e a busca passa a ter um maior alcance. Com inferência ativada, o raciocinador entende os relacionamentos que existem entre as propriedades, deduzindo novas triplas RDF, de modo a possibilitar resultados mais abrangentes.

Figura 44: Página para realizar busca por palavra chave.

No exemplo da *Figura* 44, é realizada uma busca, sem inferência, com a palavra "Semantic", para buscar projetos que tenham esta palavra na propriedade `doap:name` ou `doap:description`. Nesse caso a consulta é feita de forma direta somente na classe *Projeto* e obtém dois projetos como resultados, ou seja, encontrou a palavra-chave em uma das propriedades mencionadas anteriormente. Sendo assim, é exibido para o usuário o nome do projeto, que é um *hiperlink* possibilitando ao usuário navegar para ver a página de exibição do projeto, como pode ser visto na figura anterior apresentada.

No momento em que é ativada a inferência, a busca é feita utilizando-se as propriedades `rdfs.label` e `rdfs.comment` de todos os recursos, independente de tipo (classe), conforme explicado no capítulo anterior. Neste caso, o raciocinador entende que `doap.name`, `foaf.name` e `skos.prefLabel` são subpropriedades de `rdfs.label` e `doap:description` é subpropriedade de `rdfs.comment`, ampliando a busca para outras entidades. Na Figura 45 é apresentado o resultado da consulta com inferência. Perceba o aumento no número de resultados.

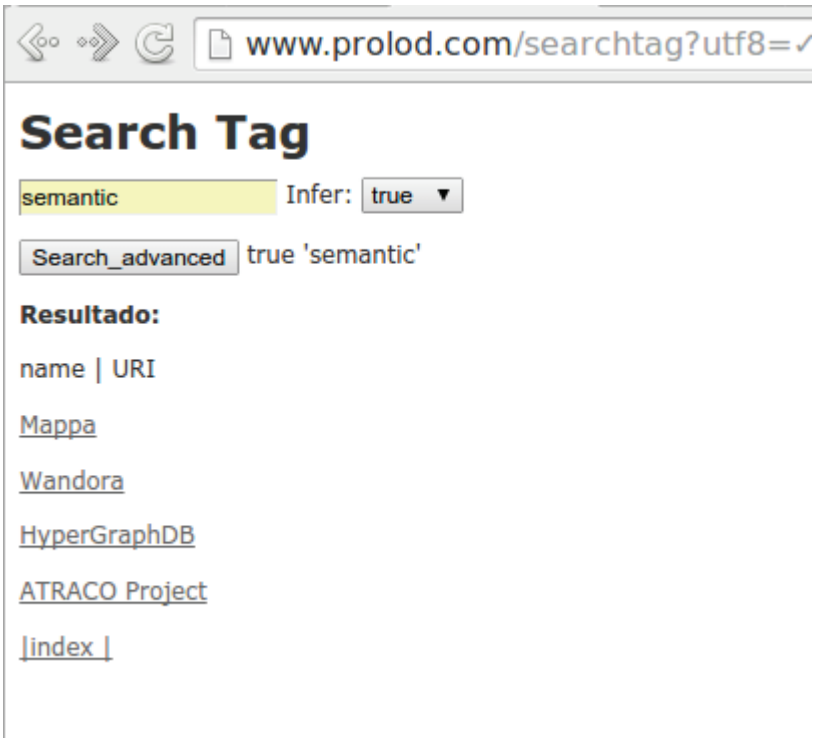

Figura 45: Exemplo de resultado da consulta com inferência.

A segunda consulta é a busca de projetos por categoria. Conforme apresentado na seção 3.7.2, a consulta SPARQL retorna os projetos relacionados com uma determinada especificada. Essa busca sem inferência somente consegue trazer projetos que tenham relação direta com a categoria selecionada por meio da propriedade `doap:category` conforme apresentada na Figura 46.

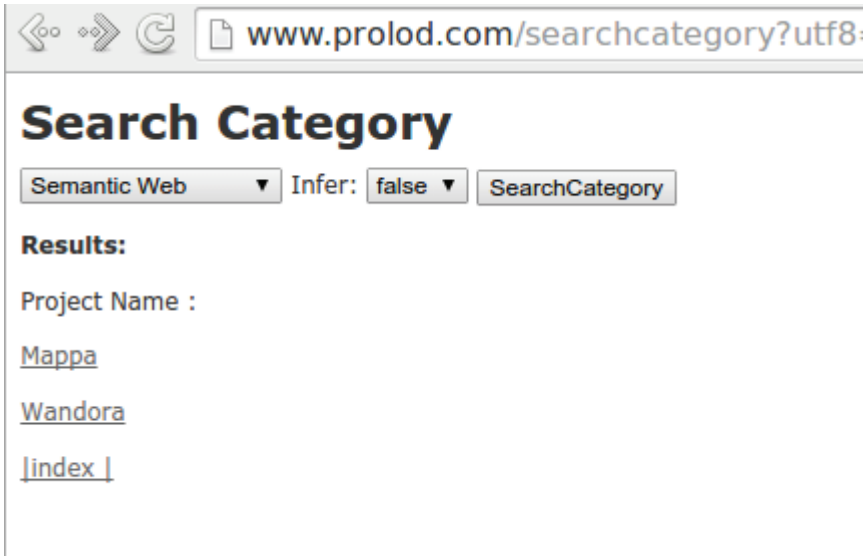

Figura 46: Resultado da busca por categoria sem inferência.

No entanto, ao ativar a inferência, esta mesma busca é ampliada retornando os Projetos da Categoria relacionada, bem como os Projetos relacionados às subcategorias diretas e indiretas (descendentes). Isto ocorre porque o raciocinador consegue entender que a propriedade `skos:norrower` é subpropriedade da propriedade `skos:norrowerTransitive` e esta propriedade é transitiva. Na Figura 48 temos o exemplo da descrição de uma categoria `http://www.prolod.com/tags/SemanticWeb`, sem o uso da inferência, persistida no *Allegrograph*. Já na Figura 49, é apresentada a mesma categoria com triplas inferidas. Perceba que sem inferência, a categoria tem apenas relacionamentos `skos:norrower` com sua categoria filha `http://www.prolod.com/tags/Linkeddata`. Já com inferência, a mesma categoria passa a ter relacionamento `skos:norrowerTransitive` com a categoria filha `http://www.prolod.com/tags/LinkedData` e com a categoria neta `http://www.prolod.com/tags/Ontology`.

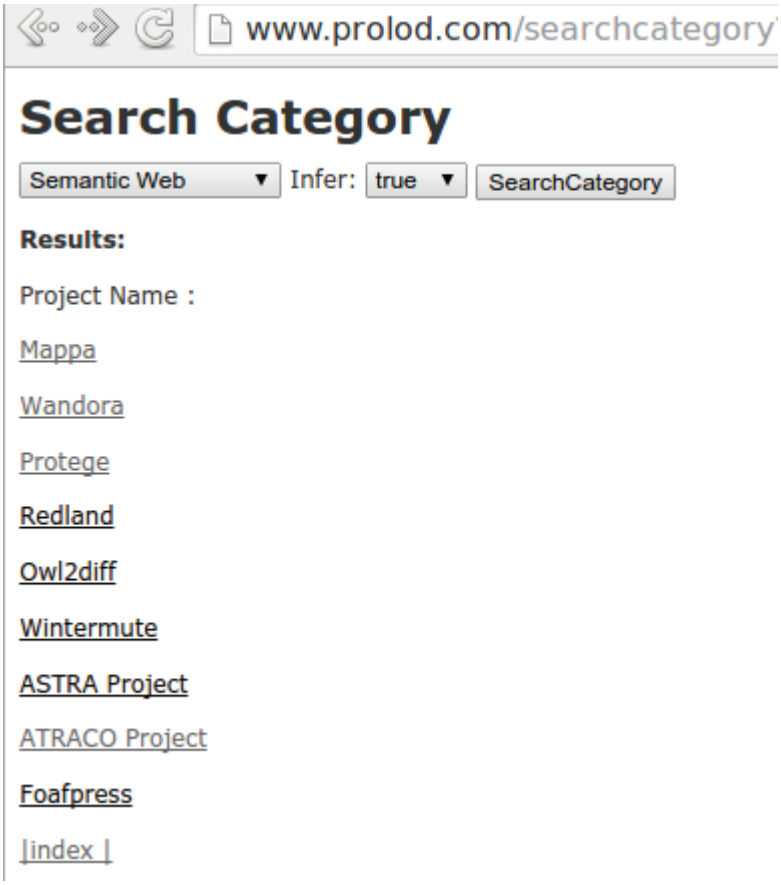

Figura 47: Resultado da busca por categoria com inferência.

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix err: <http://www.w3.org/2005/xqt-errors#> .
@prefix fn: <http://www.w3.org/2005/xpath-functions#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix fti: <http://franz.com/ns/allegrograph/2.2/textindex/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://www.prolod.com/tags/SemanticWeb> a skos:Concept ;
    rdfs:seeAlso <http://dbpedia.org/resource/Semantic_Web> ;
    skos:definition "The Semantic Web isn't just about putting data on the web.
    skos:narrower <http://www.prolod.com/tags/LinkedData> ;
    skos:prefLabel "Semantic Web" .
```

Figura 48: Declaração de uma categoria em *Turtle*.

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix err: <http://www.w3.org/2005/xqt-errors#> .
@prefix fn: <http://www.w3.org/2005/xpath-functions#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix fti: <http://franz.com/ns/allegrograph/2.2/textindex/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://www.prolod.com/tags/SemanticWeb> a rdfs:Resource,
        skos:Concept ;
    rdfs:label "Semantic Web" ;
    rdfs:seeAlso <http://dbpedia.org/resource/Semantic_Web> ;
    skos:definition "The Semantic Web isn't just about putting data on the web. I
    skos:narrower <http://www.prolod.com/tags/LinkedData> ;
    skos:narrowerTransitive <http://www.prolod.com/tags/LinkedData>,
        <http://www.prolod.com/tags/Ontology> ;
    skos:note "The Semantic Web isn't just about putting data on the web. It is a
    skos:prefLabel "Semantic Web" ;
    skos:related <http://www.prolod.com/tags/ArtificialIntelligence> ;
    skos:semanticRelation <http://www.prolod.com/tags/LinkedData>,
        <http://www.prolod.com/tags/Ontology> .
```

Figura 49: Declaração de uma categoria utilizando inferência em *Turtle*.

5 TRABALHOS RELACIONADOS

Todos os catálogos/repositórios de projetos de *software* mencionados anteriormente são de certa forma, trabalhos relacionados, com a diferença de que são aplicações *Web* convencionais, não semânticas. Dentre eles, vale destacar o Portal do *Software* Público Brasileiro que é um repositório de *software* livre que permite que sejam cadastrados *softwares* para que seja compartilhado e disseminado este conhecimento. Esse portal, muito divulgado no Brasil, pode ser considerado uma das grandes referências de catalogação de *software* do Brasil, sendo que seu formato segue a estrutura da *Web* Convencional, não disponibilizando dados de forma estruturada (BRASILEIRO, 2007).

Com relação à ontologia mais importante utilizada neste trabalho, à ontologia *Description of a Project*[47] (DOAP) para descrever projetos de software, temos:

O catálogo de projetos da *Apache Software Foundation*[48] (ASF), no qual foi criado o catálogo de projetos *softwares* desenvolvidos pela fundação *Apache*, com objetivo de ter um ponto central para buscar esses projetos. Internamente a ASF utiliza DOAP para cadastrar os *softwares* (GARDLER, 2013).

Maven Doap Plugin[49] - *plugin* para gerar descrição do projeto usando a ontologia DOAP a partir de um *Project Object Model* (POM) da ferramenta *Apache Maven* de gerenciamento de *building* de projetos baseados na linguagem de programação Java.

DOAP A Matic[50] - ferramenta Web simples cujo objetivo é criar uma descrição

---

[47] DOAP - https://github.com/edumbill/doap

[48] Apache Software Foundation (ASF) - http://www.apache.org/

[49] Maven DOAP Plugin - http://maven.apache.org/plugins/maven-doap-plugin/index.html

[50] DOAP a Matic - http://crschmidt.net/semweb/doapamatic/

DOAP inicial de um projeto que pode ser estendida depois. Trata-se de um simples formulário a ser preenchido com algumas propriedades da ontologia DOAP.

Apache Labs DOAPizer[51] - aplicação Javascript que ilustra a criação de um arquivo DOAP.

Alguns repositórios de projetos de *software*, como SourceForge[52], proveem APIs para expor informações sobre projetos usando a ontologia DOAP. Neste caso, a informação que o usuário provê ao cadastrar um projeto é usada para gerar a descrição DOAP[53] do projeto.

Nenhum destes trabalhos tem a completude oferecida pela aplicação descrita nesta dissertação. Todos são geradores de arquivos DOAP. Não oferecem suporte para combinação com outras ontologias pertinentes, nem consultas semânticas, também não fazem *mashup* com outras fontes de dados, tampouco proveem inferência.

---

[51] Apache Labs DOAPizer - http://labs.apache.org/doapizer.html

[52] DOAP REST API SourceForge - http://sourceforge.net/p/forge/documentation/Allura%20API/#project-doap-rdfxml

[53]Description of a Project (DOAP) - https://github.com/edumbill/doap/wiki

6 CONCLUSÃO

Este trabalho apresentou a construção de um protótipo para catalogação de projetos de *software* utilizando as tecnologias e padrões da *Web* Semântica em conformidade com os princípios *Linked Data*. No estudo de caso, foi demonstrado, na prática, como o emprego da *Web* Semântica pode ser benéfico para os usuários, no momento que proporciona uma redução de ações manuais no processo de inserção e recuperação da informação. Além disso, uma vez que os dados são armazenados de forma estruturada em RDF, consultas mais sofisticadas podem ser feitas com auxílio da máquina, por meio de inferências e emprego das ontologias, obtendo assim resultados que seriam difíceis ou impossíveis de serem obtidos na *Web* convencional, em larga escala.

Na próxima sessão são reiteradas as contribuições obtidas por este trabalho. Em seguida, são elencados alguns trabalhos futuros com objetivo de dar prosseguimento ao trabalho desenvolvido.

6.1 CONTRIBUIÇÕES

A contribuição central desta pesquisa é o protótipo *Linked Data* para catalogação de projetos de *software*, aplicação esta que pode ser utilizada por qualquer agente desenvolvedor de *software* para registrar estruturada e semanticamente seu *baseline* histórico de projetos de *software*, *baseline* este que pode ser facilmente publicado quer seja na intranet da corporação, quer seja na *Web* global, contribuindo para o crescimento da *Web* de Dados Ligados ou *Web* 3.0.

Como parte fundamental da aplicação mencionada, destaca-se o modelo ontológico extensível para descrição *Linked Data* de projetos de *software* que pode ser empregado por diferentes aplicações que necessitem registrar projetos de *software*, como por exemplo, repositórios de projetos *software* agregadores como discutidos na introdução. Vale destacar que este modelo foi publicado no artigo “Um modelo baseado

em ontologias *linked data* para Catalogação de Projetos de *Software*", na 12º Conferência Ibero-Americana WWW/Internet 2014.

Outro ponto considerável é a abordagem semiautomática, com aprovação de usuário e as ferramentas utilizadas para fazer o *mashup Linked Data* da aplicação com o *hub* central da *Web* de Dados, a *DBpedia*. Esta abordagem pode ser reusada pelas mais diversas aplicações *Linked Data*. Por fim, aplicação e o estudo de caso deixam um exemplo de arquitetura e diretrizes de criação de uma aplicação *Linked Data*, demonstrando a eficácia e como podemos usar a *Web* Semântica em um determinado domínio de conhecimento.

6.2 TRABALHOS FUTUROS

Como principal trabalho futuro ou segundo passo deste projeto, enfatizamos a criação de um *framework* que viabilize o desenvolvimento de repositórios agregadores de projetos *software*, usando o modelo ontológico aqui proposto. Estes repositórios serão o casamento perfeito com a aplicação proposta. De um lado teremos a aplicação *Linked Data* para catalogação de projetos de *software*, utilizada pelas diversas empresas produtoras de *software* ao redor do mundo (ou pelos diversos departamentos de uma empresa, se estivermos falando de um ambiente fechado de intranet). Do outro lado, teremos os servidores agregadores que, sem intervenção humana, integram, *on the fly*, a descrição de inúmeros projetos de *software*, publicados por *Web sites* ou *Web services* por este mundo afora. Este agregador é uma base de conhecimento que não para de crescer e isso tudo de forma automática, máquina-máquina. Depois basta fazer consultas (SPARQL) nestes agregadores para encontrar precisamente o que se procura.

Podemos ainda sugerir a criação de novas consultas SPARQL na aplicação proposta que explore ainda mais o universo de inferência trazido pelas ontologias, enriquecendo a experiência do usuário e a usabilidade da aplicação. Outro trabalho seria a otimização de desempenho da aplicação proposta, no que diz respeito ao *mashup*

semântico com fontes de dados geograficamente distribuídas. Neste caso, o banco de dados deve ser incrementado com técnicas de cache, dereferenciando os recursos ligados e armazenando suas representações localmente.

Outra sugestão seria melhorar a navegação/interface da aplicação, incorporando as informações oriundas do *mashup* semântico nas fronteiras da própria aplicação, de forma que o usuário não precise sair da aplicação para acessá-las. Em outras palavras, tornar indiferente para o usuário quando o mesmo está navegando por informações intrínsecas ou extrínsecas à aplicação.

Ainda com relação à interface com o usuário da aplicação, esta pode ser mais rica e amigável, por exemplo, com uso de bibliotecas AJAX apropriadas. Mas, sobretudo, uma questão bastante interessante é explorar a noção de interface orientada por ontologias, ou seja, *widgets* para visualização/edição customizados, dinamicamente, de acordo com as construções ontológicas que descrevem os dados.

7 REFERÊNCIAS BIBLIOGRÁFICAS

BRASILEIRO, PORTAL DO SOFTWARE PÚBLICO. **Portal do Software Público Brasileiro.** 2007. Disponível em: <www.softwarepublico.gov.br/>. Acesso em: 20 ago. 2014.

BRATT, STEVE. ***Semantic Web, and Other W3C Technologies to Watch***. 2006. Disponível em: <http://www.w3.org/2006/Talks/1023-sb-W3CTechSemWeb/>. Acesso em: 10 maio 2014.

BERNERS-LEE, T.; ***Information Management: A Proposal,***CERN, 1989. Disponível em: <http://www.w3.org/History/1989/proposal.html>. Acesso em: Out. 2012.

BERNERS-LEE, T.; FISCHETTI, M. ***Weaving the web - the original design and ultimate destiny of the world wide web****, by its inventor.* ISBN 006251587X, 1997. <http://googleblog.blogspot.com.br/2008/07/we-knew-web-was-big.html>. Acesso em: 10 out 2012.

BERNERS-LEE, T., J. HENDLER, AND O. LASSILA. ***The Semantic Web. Scientific American***, Maio 2001.

BERNERS-LEE, T., CHEN, Y., CHILTON, L., CONNOLLY, D., DHANARAJ, R., HOLLENBACH, J., LERER, A., AND SHEETS, D. ***Tabulator: Exploring and Analyzing Linked Data on the Semantic Web. In In Procedings of the 3rd International Semantic Web User Interaction Workshop*** (SWUI06, *page* 06). 2006.

BRICKLEY, Dan; GUHA, R.V. **RDF *Schema*** 1.1. 2004. Disponível em: <http://www.w3.org/TR/rdf-schema/>. Acesso em: 30 jul. 2014.

CUNHA, D. R. ; SOUZA, D. ; LÓSCIO, B. F. ***Linked Data*****: da Web de Documentos para a Web de Dados. In: III Escola Regional de Computação** - Ceará Maranhão e Piauí - ERCEMAPI 2011. : SBC, 2011.

FRANZ INCORPORATED. ***AllegroGraph* 4.14 Introduction**. 2005. Disponível em: <http://franz.com/agraph/support/documentation/current/agraph-introduction.html>. Acesso em: 12 jul. 2014.

FREITAS, G.C.DE . **Perspectiva e Análise de Viabilidade Sobre a Web Semântica. Trabalho de conclusão de graduação**. Universidade São Francisco. Campinas - São Paulo 2011. http://www.scribd.com/doc/75824460/37/Logica-Prova-e-Confianca

GARDLER, Ross. ***Project Catalogues And Project Descriptors Using* DOAP**. 2013. Disponível em: <http://oss-watch.ac.uk/resources/doap>. Acesso em: 01 de maio de 2014

HEATH, T.; BIZER, C. ***Linked Data: Evolving the Web into a Global Data Space*** *(1st edition). Synthesis Lectures on the Semantic Web: Theory and Technology*, 1:1, 1-136. Morgan & Claypool, 2011.

JACYNTHO, MARK DOUGLAS DE AZEVEDO. **Um modelo de bloqueio multigranular para RDF**. 2012. 90 f. Tese (Doutor) - Departamento de Informática, PUC-RIO, Rio de Janeiro, 2012.

JARDIM, ANDRÉ DESESSARDS. **Aplicações de Modelos Semânticos em Redes Sociais**. 2010. 106 f. Dissertação (Mestre) - Curso de Mestre em Ciência da Computação, Universidade Católica De Pelotas, Pelotas, 2010.

LAGHI, Murilo Soares; LENGLER, Roberto Daniel. **USO DE TECNOLOGIAS SEMÂNTICAS EM GOVERNANÇA DE TI**. 2011. 145 f. TCC (Graduação) - Curso de Curso de Sistemas de Informação, Departamento de Departamento de InformÁtica e EstatÍstica, Universidade Federal de Santa Catarina, Florianópolis -sc, 2011

KRASNER, G. E.; POPE, S. T**.** ***A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80***. 1988.

LAVENDER, Ben. *Spira:* ***A Linked Data ORM for Ruby***. 2010. Disponível em: <http://blog.datagraph.org/2010/03/rdf-for-ruby>. Acesso em: 20 de novembro de 2013.

LOD. *Cloud diagram.*Disponível em : http://richard.cyganiak.de/2007/10/lod/ acessado em

05 de janeiro de 2013.

MANOLA, F.; MILLER, E. ***RDF Primer*. W3C *Recommendation,*** 10 February 2004. Disponível em <http://www.w3.org/TR/rdf-primer/>. Acessado em 12 de novembro de 2012.

MENDES, PABLO N.; JAKOB, MAX; GARCÍA-SILVA, A.; BIZER, CHRISTIAN. ***DBpedia spotlight: shedding light on the web of documents***. ISSN 978-1-4503-0621-8, 2011.

PEREIRA, Henrique C. (Ed**.). Folksonomia e a maneira com que nós colocamos ordem nas coisas**. 2016. Disponível em: <http://revolucao.etc.br/archives/folksonomia-e-a-maneira-com-que-nos-colocamos-ordem-nas-coisas/>. Acesso em: 30 de maio de 2013.

PRESSMAN, Roger S. ENGENHARIA DE SOFTWARE - MAKRON Books, 1995.

RIOS, L. Uma Arquitetura para Integração de Gestão do Conhecimento e e-Learningvia Web Semântica. Dissertação (Mestrado Integrado Profissional emComputação Aplicada - MPCOMP) 93 f. Universidade Estadual do Ceará,Fortaleza, 2008.

SANTOS JR., C. **Atratividade de Projetos de Software Livre: Importância Teórica e Estratégias para Administração**, Revista de Administração e Economia (RAE), v. 50 (4), 2010.

SILVA, FERNANDA CARDINALY DE ARAÚJO. **Web semântica, desvendando a evolução da web e dos softwares de suporte a tecnologia semântica**. Disponível em: <http://pt.scribd.com/doc/93442754/Web-Semantica>. Acesso em: 05 de julho de 2012.

VASCONCELOS, GABRIELA FERNANDA SILVA DE. **Publicando dados dos docentes do CIn- UFPE na Web de dados**. 2012. 97 f. Monografia (Graduado) - Departamento deCentro de Informática, Universidade Federal de Pernambuco, Recife, 2012.

W3C BRASIL. WEB Semântica.. Disponível em:

<http://www.w3c.br/Padroes/WebSemantica>. Acesso em: 11 out. 2014.