UNIVERSIDADE ESTADUAL DO NORTE DO PARANÁ

FACULDADES LUIZ MENEGHEL

PEDRO HENRIQUE ZAMMATARO YASUI

CONSTRUINDO UM MODELO ONTOLÓGICO PARA EVENTOS CIENTÍFICOS:

UMA ABORDAGEM EM DIREÇÃO A WEB SEMÂNTICA

Bandeirantes

2007

PEDRO HENRIQUE ZAMMATARO YASUI

CONSTRUINDO UM MODELO ONTOLÓGICO PARA EVENTOS CIENTÍFICOS:

UMA ABORDAGEM EM DIREÇÃO A WEB SEMÂNTICA

Monografia apresentada ao curso de Sistemas de Informação da Universidade Estadual do Norte do Paraná, campus Bandeirantes, para a obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Prof. Msc. Ailton S. Bonifacio

Bandeirantes

2007

PEDRO HENRIQUE ZAMMATARO YASUI

CONSTRUINDO UM MODELO ONTOLÓGICO PARA EVENTOS CIENTÍFICOS:

UMA ABORDAGEM EM DIREÇÃO A WEB SEMÂNTICA

Monografia apresentada ao curso de Sistemas de Informação da Universidade Estadual do Norte do Paraná, campus Bandeirantes, para a obtenção do grau de Bacharel em Sistemas de Informação.

COMISSÃO EXAMINADORA

_____

Prof. Msc. Ailton Sergio Bonifacio
Universidade Estadual do Norte do Paraná

_____

Prof. Msc. André Luis Andrade Menolli
Universidade Estadual do Norte do Paraná

_____

Prof. Msc Glauco Carlos Silva
Universidade Estadual do Norte do Paraná

Bandeirantes, __ de_____ de 2007

"O amor é um sentimento sublime; supera
os problemas e diferenças e resiste ao tempo.
E se fortalece com a distância.
Difícil fugir, impossível esquecer."

Barbosa Filho

AGRADECIMENTOS

Agradeço a Deus por me dar a oportunidade de viver.
Agradeço a minha namorada Simone por ser minha companheira fiel durante essa minha caminhada.
Agradeço a minha família, principalmente meus pais Valter e Marta e meu irmão Vitor por tudo que fazem por mim.
Agradeço ao meu orientador Ailton pela companhia durante o ano e por todo o conhecimento compartilhado.
Agradeço a todos os professores do Departamento de Informática pela convivência durante essa vida universitária.
Agradeço ao pessoal da VI Turma de Sistemas de Informação pela companhia. Vão fazer muita falta.
Agradeço aos meus amigos por estarem presentes em todos os momentos da minha vida. Sem vocês eu não seria ninguém.
E por fim agradeço aos moradores e animais da República Tcheca pela colaboração durante a realização deste trabalho.

A todos os citados acima, um sincero muito obrigado por tudo.

RESUMO

Devido ao aumento substancial das informações na web, as tarefas de recuperação da informação estão se tornando cada vez mais imprecisas e ineficientes. Atualmente, estas tarefas são realizadas, em sua maioria, através de consultas por palavras-chaves. Para essa tarefa, é necessário que se faça uma busca exaustiva sobre os documentos na web devido à heterogeneidade dos dados e a falta de estrutura dos mesmos. Estes conceitos vão contra a idéia da Web Semântica, que propõe estruturar a web de modo que os motores de busca realizem buscas sobre a semântica dos documentos. Atualmente, as pesquisas estão sendo direcionadas para que seja feita uma estruturação através de ontologias que visam definir restrições sobre conceitos de um determinado domínio. Este trabalho visa desenvolver uma ontologia para eventos científicos, levando em consideração a necessidade de se criar uma estrutura para definição de eventos e o interesse da comunidade em participar destes eventos. Para o desenvolvimento desta ontologia utilizou-se a linguagem OWL juntamente com a ferramenta *Protégé* e o raciocinador *RacerPro*. Essas tecnologias utilizadas em conjunto são suficientes para desenvolver uma ontologia capaz de suportar um grande número de eventos instanciados.

Palavras-chaves: Ontologias, Web Semântica, Eventos Científicos

ABSTRACT

In reason of the substantial increase of the information in web, the tasks of recovery of the information they are becoming more inexact and inefficient. Actually, the tasks is execute in the bigger part for research for key-words. For this tasks is necessary a exhaustive research on the web documents in reason of the data heterogeneity and its low structure. These concepts refute the Semantic Web idea that it considers to structuralize the web for the search motors look for the documents semantic. Actually, the searches are directed for data structure using ontologys to define restrictions for a determinate domain. This work look develops ontology for scientific events considering the necessity to create a structure for the definition for events and community interest in to participate of these events. For development this ontology was utilize the language OWL with Protégé instrument and the reasoner RacerPro. When this technologies was use in conjunct are sufficient for development a ontology for support a lot of created events.

Key-words: Ontologys, Semantic Web, Scientific Events

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

API:	Application Programming Interface
ASCII:	American Standard Code for Information Exchange
CAPES:	Coordenação de Aperfeiçoamento de Pessoas de Nível Superior
DAML:	Darpa Agente Markup Language
DAML+OIL:	Integração DAML+OIL
DTD:	Document Type Definition
FACT:	Fast Classification of Terminologies
HTML:	Hiper Text Markup Language
IRI:	Internationalized Resource Identifier
ISO:	International Standards Organization
MER:	Modelo Entidade Relacionamento
OIL:	Ontology Inference Layer
OML:	Ontology Markup Language
OWL:	Web Ontology Language
OWL-DL:	Web Ontology Language – Description Logic
RacerPro:	Renamed ABox and Concept Expression Reasoner Professional
RDF:	Resource Description Framework
RDF Schema:	Resource Description Framework Schema
SBC:	Sistemas Baseados em Conhecimento
SGBD:	Sistema Gerenciador de Banco de Dados
SWRL:	Semantic Web Rule Language
TCP/IP:	Transmition Control Protocol / Internet Protocol
UML:	Unified Modeling Language
UNA:	Unique Name Assumption
URI:	Universal Resource Identifier
W3C:	World Wide Web Consortium
XML:	eXtensible Markup Language
XML Schema:	eXtensible Markup Language Schema
Xmlns:	eXtensible Markup Language Namespaces
XOL:	Ontology Exchange Language

SUMÁRIO

1 INTRODUÇÃO

A web atualmente é formada por um volume de dados muito grande que não se encontram armazenados em sistemas caracterizados como banco de dados. Esse armazenamento pode ser feito em sistemas de arquivos como bibliotecas digitais ou informações acessíveis através da própria web através de sites.

Como esses dados não são mantidos em Sistemas Gerenciadores de Banco de Dados (SGBD), a tendência é que apresentem uma organização bastante heterogênea. Essa heterogeneidade torna complexa as atividades de pesquisa sobre eles, devido a sua falta de estrutura. Uma consulta sobre estes dados é realizada através de palavras-chaves. Com essas palavras-chaves, os agentes de busca verificam exaustivamente os documentos, retornando resultados nem sempre precisos (MELLO *et* al, 2000).

Essas buscas lentas e imprecisas vão contra a idéia de Tim Berners - Lee, um dos idealizadores da Web Semântica. A Web Semântica visa extrair a semântica dos documentos da web, de modo que as atividades de busca, por exemplo, se tornem mais precisas e mais rápidas.

O objetivo da Web Semântica é de fazer com que as aplicações se tornem capazes de compreender e raciocinar sobre a informação lida, ao invés de serem apenas leitoras de informação (SOUTO; WARPERCHOWSKI; OLIVEIRA, 2006).

Para o desenvolvimento da Web Semântica, há a necessidade de se utilizar ontologias. As ontologias agem em conjunto com o modelo RDF (*Resource Description Framework*) para estruturar hierarquicamente os documentos. Estes documentos se encontram em linguagem XML (*eXtensible Markup Language*), que os estrutura mas não possui nenhum tipo de semântica.

Como se sabe, as informações sobre congressos, conferências e outros eventos científicos são disponibilizadas aos usuários da internet através de sites de entidades relacionadas, páginas de pessoas ligadas ao evento ou pela página do próprio evento.

Embasado sobre essa visão da Web Semântica e analisando as informações sobre os eventos científicos, seria interessante estabelecer uma solução para organizar e reunir informações sobre eventos, de forma que as mesmas fiquem organizadas e sejam interpretadas por agentes computacionais.

Este trabalho visa propor e construir um modelo ontológico para estruturar os eventos científicos, modelando-os com suas particularidades,

estabelecendo seus principais pontos e criando os relacionamentos entre eles.

1.1 Objetivos

1.1.1 Objetivo Geral

Este trabalho tem como objetivo principal a construção de uma ontologia para eventos científicos em geral, tais como congressos, conferências, simpósios e similares, não necessariamente para uma área de atuação específica. Para o desenvolvimento desta ontologia utilizou-se a ferramenta Protégé-OWL, que provê suporte à linguagem OWL (*Web Ontology Language*), que é uma recomendação do W3C (*World Wide Web Consortium*) para o desenvolvimento de ontologias.

1.1.2 Objetivos Específicos

Antes de se atingir o objetivo de construir uma ontologia, alguns objetivos específicos devem ser alcançados. Os objetivos específicos deste trabalho são:

- Contextualizar a situação da web atualmente;
- Apresentar a definição de ontologias e como desenvolvê-las, e mostrar como as ontologias podem melhorar a web;
- Explicar a relação entre OWL, RDF Schema e XML;
- Apresentar a linguagem de ontologias OWL, bem como as suas sub-linguagens;
- Explicar o projeto da Web Semântica, expondo sua história, características, benefícios entre outros;
- Utilizar e demonstar a ferramenta *Protégé-OWL*, que será utilizada para o desenvolvimento da ontologia proposta.

1.2 Justificativa

A finalidade da Web Semântica é estruturar e organizar as informações de maneira que as buscas se tornem mais eficientes e inteligentes. Para isso, utiliza-se principalmente o conceito de ontologias (ARAÚJO, 2003).

Sobre essa idéia, propõe-se desenvolver uma ontologia para o domínio de eventos científicos, mas sem nenhuma área de atuação especifica. Esses eventos

englobam congressos, simpósios, workshops e similares. Essa ontologia iria adicionar semântica aos dados dos eventos, tornando mais fácil extrair as informações desejadas.

Esta ontologia se faz necessária, pois há um grande número de eventos, em todas as áreas de conhecimento. Outro motivo que leva ao desenvolvimento da ontologia é o interesse das pessoas nestes eventos, buscando informações importantes sobre datas de inscrição, de realização e submissão de artigos, local de realização, taxas, comissão organizadora entre outros.

Com esta ontologia desenvolvida, essas informações são estruturadas de modo que a recuperação de informação sobre estes eventos ficaria mais eficiente e rápida, seguindo o caminho proposto pela Web Semântica.

Para o desenvolvimento da ontologia, utilizou-se a ferramenta *Protégé-OWL* versão 3.2.1, desenvolvida pela Universidade de Stanford. Esta versão atualmente é a mais recente disponível. Esta ferramenta foi escolhida por oferecer suporte à linguagem OWL e às tecnologias RDF e XML. Estas tecnologias foram estabelecidas pelo W3C como padrão para o desenvolvimento de ontologias.

1.3 Organização do Trabalho

Este trabalho se encontra estruturado da seguinte maneira: no capítulo 2 será realizada uma abordagem sobre Web Semântica e as tecnologias utilizadas para a promoção da mesma atualmente. O capítulo 3 é referente a ontologias englobando tipos de ontologias e metodologias, entre outros conceitos. O capitulo 4 explica o desenvolvimento do trabalho. Finalmente, no capitulo 5 são apresentadas as considerações finais e trabalhos futuros relacionados.

2 WEB SEMÂNTICA

Atualmente, a web é considerada a maior fonte de disseminação da informação procedente de qualquer área do conhecimento. Devido a essa grande quantidade de informações existentes, existem problemas enfrentados tratando-se de recuperação de informações. A respeito desta afirmação, pode-se acrescentar que esta quantidade de informações está relacionada à grande utilização do protocolo TCP/IP (*Transmition Control Protocol/Internet Protocol*) e a invenção dos *browsers* (navegadores) (ARAÚJO, 2003).

Outro ponto a ser considerado ao falar sobre a web atualmente é o armazenamento e a estrutura dos dados. Ao contrário de um SGBD, que apresentam uma estrutura de representação ou esquema previamente definido, os dados para acesso eletrônico apresentam uma organização bastante heterogênea, que varia dependendo do tipo de dado. Na web, um dado pode ser uma imagem, vídeo, texto, entre outros (MELLO *et al*, 2000).

Ainda segundo Mello *et al* (2000), a heterogeneidade torna complexa qualquer atividade de pesquisa, pois não existe uma estrutura definida que possa gerar esquemas para que a consulta seja formulada.

Existem algumas técnicas para se procurar o conteúdo desejado na web atualmente. A primeira é a navegação exaustiva, percorrendo manualmente as páginas web até se encontrar o conteúdo desejado. A segunda técnica é utilizar ferramentas de busca disponíveis na internet atualmente.

Existem problemas nestes dois métodos. O primeiro exige muito tempo e paciência do usuário pois geralmente, dependendo do que se busca, não é uma tarefa simples procurar conteúdo na internet devido ao grande volume de informações existentes.

Sobre a segunda técnica, Cendon (2001) cita que existem milhares de ferramentas de busca. O problema destas ferramentas é o tempo gasto e os resultados geralmente são pouco precisos. Isso ocorre porque essas ferramentas realizam buscas por palavras-chaves (*full-text search*) e o processamento final dos resultados é feito por conta do usuário. Araújo (2003) explica essas imprecisões citando que as informações inseridas em meio eletrônico são facilmente interpretadas pelo homem, mas não são compreendidas por agentes computacionais.

Essa busca indexada fica clara ao se realizar uma pesquisa em qualquer site de busca atualmente. Experimente digitar a palavra “sol” em uma

ferramenta de busca. Certamente serão retornados resultados indesejáveis ao usuário porque a palavra “sol” pode representar um astro, uma marca, nome de pessoa, entre outros. O homem consegue interpretar o sentido da palavra “sol”, mas os computadores atualmente não conseguem trabalhar com a semântica das palavras, trabalhando apenas com a sua sintaxe.

Sobre essas dificuldades em relação a recuperação da informação na web atualmente, Bonifacio e Heuser (2002) concluem que a web apresenta sérios problemas de localização, acesso, apresentação e manutenção dos dados por parte dos usuários, que são os principais fatores envolvidos na web.

Para solucionar ou pelo menos amenizar esse problema de desorganização da informação na web, é necessário que se estabeleça algum padrão ou método para que os dados e informações contidos em páginas da web sejam estruturados de modo que os computadores possam trabalhar sobre eles extraindo informações de maneira ordenada.

Pensando nessa solução foi criada a Web Semântica. Esse termo foi idealizado por Tim Berners-Lee *et al* (2001) e possui o seguinte significado:

> “A Web Semântica não é uma web separada, mas uma extensão da atual, na qual a informação é utilizada com significado bem definido, aumentando a capacidade dos computadores para trabalharem em cooperação com as pessoas”.

Atualmente, esta idéia de Web Semântica é apenas um projeto. Sobre esta idéia denota-se a possibilidade de se possuir dados da web conectados e com significados bem definidos (ARAÚJO, 2003).

De acordo com Souto, Warpechowski e Oliveira (2006), o objetivo da Web Semântica é desenvolver aplicações capazes de compreender e raciocinar a informação. Com isso, elas deixariam de ser apenas leitoras de informação como são atualmente. Bonifacio (2002) complementa que esse processo de automação elevaria a web de “machine-readable”, para “machine-understandable”.

Ainda segundo os dois autores, para que as aplicações alcancem o status de raciocinar a informação, alguns passos devem ser seguidos. Os conteúdos e as funcionalidades dos recursos devem ser descritos de maneira formal. A descrição dos conteúdos é dada anotando-se os recursos com meta-dados. E para garantir o compartilhamento de informações entre as aplicações, existem alguns padrões de meta-dados já existentes como o Dublin Core, por exemplo.

Para que seja explorado ao máximo o potencial proporcionado pela Web Semântica, ao se desenvolver uma aplicação deve-se respeitar a arquitetura da Web Semântica proposta por Berners-Lee *et al* (2001). Esta arquitetura é demonstrada na figura 1. Cada camada descreve as tecnologias necessárias para que as páginas da web possam ser interpretadas por computadores.

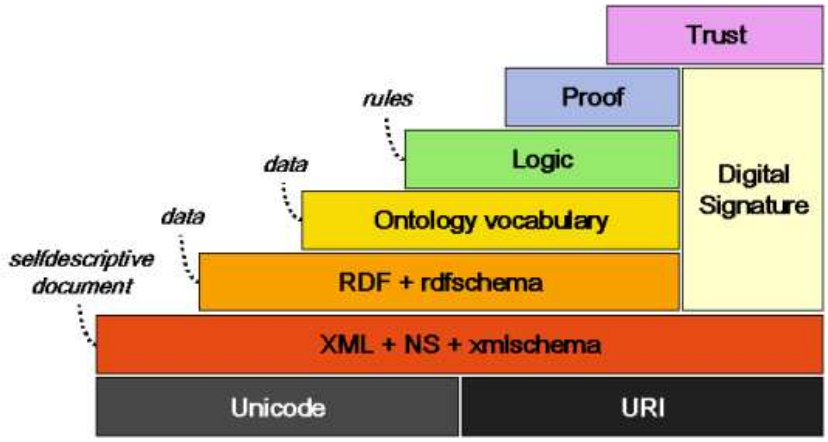

Figura 1 - Arquitetura da Web Semântica segundo Berners-Lee *et al* (2001)

De acordo com a figura acima, cada camada executa uma função diferente para a implementação de aplicações. Porém, essas camadas não são independentes pois as camadas superiores extendem a funcionalidade das camadas inferiores.

A seguir, cada camada será explicada para melhor entendimento da arquitetura da Web Semântica.

- *Camada Unicode e URI:* é a camada base da arquitetura. O *Unicode* permite que os elementos de uma página (texto, som, vídeos, etc) possam ser lidos por pelos computadores em qualquer lugar. O *URI* (*Uniform Resource Identifier*) determina que os recursos da WEB devem ser nomeados por identificadores únicos.
- *Camada XML + NS + xmlschema:* esta camada possibilita que as tecnologias XML, *Namespaces* e XML Schema estruturem os documentos da web em árvores utilizando as tags que são definidas pelos usuários.
- *Camada RDF + rdfschema:* depois de organizados em árvores, os dados já possuem significado e esta camada utiliza o RDF Schema para representar o conhecimento através relacionamentos entre conceitos. Nesta camada são descritos os meta-dados, pois pode se notar a descrição dos dados sobre dados.
- *Camada Ontology vocabulary:* esta camada representa o vocabulário da ontologia que é usado para descrever algum domínio do conhecimento. Essa

descrição permite que as máquinas entendam o significado dos dados e a partir deles possa inferir novas informações.

De acordo com Araújo (2003), as camadas superiores ainda se encontram em estudo pelo consórcio W3C. Com isso, pode-se apenas apresentar uma pequena introdução sobre elas.

- *Camada Logic (Lógica):* especifica linguagens mais poderosas que as atuais para melhorar a construção de inferências.
- *Camada Proof (Prova):* especifica a linguagem que atua testando se a informação trocada entre agentes e máquinas é verdadeira, uma vez que a troca pode ser verdadeira, mas a informação não.
- *Camada Trust (Confiança):* a camada de Confiança verifica, através de assinatura digital, se a pessoa que realiza as transações usando uma aplicação é ela mesma.

Para o desenvolvimento de aplicações para Web Semântica é necessário que se utilizem algumas tecnologias que são padronizadas pelo W3C para esse fim. De acordo com a figura, devem ser utilizada a linguagem XML, o modelo de dados RDF. Ainda deve ser usada alguma linguagem para expressar o vocabulário de ontologias. Atualmente, utiliza-se a linguagem OWL, que trabalha utilizando os recursos das duas tecnologias anteriores para representar o conhecimento.

No entanto, para o desenvolvimento de aplicações para Web Semântica, o XML possui algumas limitações. Segundo Decker *et al* (2000), a maior limitação do XML é o fato de apenas se descrever a gramática do documento. Para eliminar essa deficiência, utiliza-se o RDF que adiciona semântica aos documentos na web (ARAÚJO, 2003).

2.1 XML

A linguagem de marcação XML, definida pelo *World Wide Web Consortium* (W3C), está se tornando um padrão para manipulação, troca e representação de dados semi-estruturados atualmente na WEB (BRAGANHOLO; HEUSER, 2001).

Esta linguagem é utilizada para representar os dados como uma string de texto com marcas. Essas marcas são usadas para intercalar o texto com informações

sobre as próprias informações relacionadas a seu conteúdo ou forma.

A marcação, como em HTML (*HiperText Markup Language*), também é feita com *tags*, que também são destacados usando os caracteres “<” e “>”, sendo também usado o caractere “/” para indicar a *tag* de fechamento. Com isso, conclui-se que um documento XML é formado por *tags* e informações colocadas entre as *tags* (GRAVES, 2003).

Um documento escrito em XML pode possuir inúmeras *tags,* que são criadas conforme a necessidade e são definidas de acordo com a aplicação e o domínio de dados. Por esta razão o XML é chamado de extensível, justamente pelo fato de não possuir uma quantidade pré-definidas de *tags,* sendo assim uma metalinguagem para descrição de linguagens de marcação (DAUM; MERTER, 2002 *apud* OLIVEIRA, 2004).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<cadastro>
  <registro>
    <nome>Pedro</nome>
    <email>pedro@email.com</email>
  </registro>
  <registro>
    <nome>Ailton</nome>
    <email>ailton@email.com</email>
  </registro>
</cadastro>
```

Figura 2 - Exemplo de documento XML

De acordo com Graves (2003), o XML não tem por si só a finalidade de apresentar os dados ao usuário, como pode ser notado na figura acima. Esse conceito pode ser justificado pelo fato do XML ser desenvolvido para intercâmbio de informações na web. Para representar o conteúdo de um documento XML foram desenvolvidas folhas de estilos, que formatam o conteúdo do documento.

Segundo Anderson *et al*. (2001), os principais objetivos da linguagem XML são:

- Prover o intercâmbio de documentos por meio da web de forma independente de sistemas operacionais ou formatos de arquivos;
- Suportar uma grande gama de aplicações, permitindo a definição de elementos pelo usuário para estruturar o documento;
- Facilitar a análise de documentos XML por programas;
- Documentos XML devem ser legíveis por humanos;

- Economia de *tags* de marcação não é importante;
- Ter uma especificação normal para marcação de documentos.

Além de seu poder de extensão, o XML possui outra característica fundamental. A sua sintaxe pode ser entendida por qualquer aplicação. Essa característica contribuiu para que o XML se torne um padrão para o intercâmbio de informações na web (MELLO *et al*, 2000).

2.1.1 XML Schema

No desenvolvimento de documentos que utilizam a linguagem XML, um dos requisitos é definir o tipo de dados que podem ser inseridos nos elementos do documento. Para atender estes requisitos, o XML utiliza dois tipos de estrutura definição de dados: DTD (*Document Type Definition*) e XML Schema (GRAVES, 2003).

A escolha da estrutura a ser usada depende da aplicação. O uso de de utilizar XML Schema é motivado pelo fato desta estrutura ter sido definida em 2001 como recomendação do W3C para representação de documentos XML (W3C, 2001).

O XML Schema é um documento paralelo ao documento XML. Este documento paralelo é conhecido como esquema e serve como base a um documento de XML de instância. Essa relação pode ser comparada à relação entre objetos e classes em paradigmas orientados a objetos (BRAGANHOLO; HEUSER, 2001).

Com base nessa relação, Graves (2003) complementa que é o documento XML Schema que define os tipos de dados que serão aceitos para o documento de instância. Esses valores podem ser internos ou especificados como parte do esquema.

Esses valores internos são chamados de *simpleType* e deve receber atributos como textos, números, entre outros. Os valores especificadas são conhecidos como complexos e definidos como *complexType* e devem ser utilizados para especificar sub-elementos permitidos para um elemento (BRAGANHOLO; HEUSER, 2001).

Ainda segundo Braganholo e Heuser (2001), um *complexTime* impões restrições sobre elementos para definir a cardinalidade (*minOccurs* e *maxOccurs*) e os valores que um elemento pode assumir (*sequence*, *all* e *choice*). Os exemplos destas restrições podem ser conferidos nas próximas figuras.

```
(1)  <element name="Pessoa" type="pub:tPessoa" minOccurs="1" maxOccurs="unbounded"/>
```

Figura 3 - Restrição de cardinalidade utilizando XML Schema

```
(1)  <element name="PARAGRAFO" type="pub:tParagrafo"/>
(2)
(3)  <complexType name="tParagrafo" mixed="true">
(4)     <sequence>
(5)        <element name="CITACAO" type="string"/>
(6)        <element name="REFERENCIA" type="string/>
(7)     <sequence>
(8)  </complexType>
```

Figura 4 - Restrição de delimitação de valores

A figura 3 apresenta uma restrição de cardinalidade. Segundo esta restrição, o elemento Pessoa é obrigatório, pois o valor de *minOccurs* é igual a "1", que define o valor mínimo de ocorrências. No entanto, o número máximo de ocorrências é indeterminada devido ao valor "*unbounded*" recebido no elemento *maxOccurs*.

Caso a definição de cardinalidade não apareça na declaração, subentende-se que os valores mínimo e máximo de ocorrências é igual a "1". Esta definição aparece no segundo exemplo, que é mostrado na figura 4.

Neste exemplo, os elementos "CITACAO" e "REFERENCIA" são obrigatórios pela omissão das restrições de cardinalidade. Na linha 3, é notada a declaração do elemento *sequence*. Este elemento define a ordem que os sub-elementos devem aparecer na instância XML. Com isso, é possível concluir que o elemento "PARAGRAFO" deve conter um valor de "CITACAO" e "REFERENCIA" apenas e estes devem se apresentar nesta ordem.

2.2 RDF(S)

Embora um documento XML possa ser interpretado por um leitor humano, um computador não entende a semântica dos seus elementos. Para isso, foi criado o modelo RDF, que possui como objetivo estruturar os documentos de forma que se consiga extrair semanticamente a informação desejada (SANTOS, 2002).

Uma das maiores críticas ao XML é que sua especificação não define semântica para as marcas, permitindo que sejam adicionadas estruturas arbitrárias aos documentos, mas não diz nada a respeito do significado dessas estruturas.

Pensando nessas limitações, o W3C desenvolveu o RDF (*Resource Description Framework*), que oferece uma estrutura para adicionar semântica aos

documentos encontrados na WEB (ARAÚJO, 2003).

O RDF é uma aplicação da linguagem XML para o processamento de metadados na web. Seu principal objetivo é facilitar o intercâmbio de informações entre aplicativos na web. Esse modelo foi proposto pelo W3C em outubro de 1997, mas apenas em fevereiro de 1999 foi aprovado. O uso do modelo RDF pode ser conferido em páginas da web que apresentam o símbolo do modelo, apresentado na figura 5.

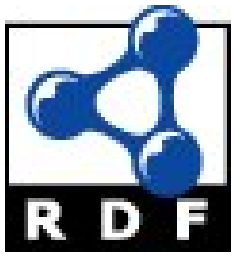

Figura 5 - Ícone do modelo RDF para identificação de páginas utilizando esta tecnologia.

Um objetivo do modelo RDF é realizar a especificação semântica dos dados de um documento XML. O W3C considera esse propósito um elemento-chave para a construção de uma “WEB confiável” (SANTOS, 2002).

2.2.1 Modelo de dados RDF

Segundo Santos (2002) e Araújo (2003), o modelo RDF concentra-se em três tipos de elementos:

- **Recursos**: um recurso é tudo que pode ser determinado por um URI, podendo ser um documento inteiro ou parte dele, um site, uma coleção de arquivos entre outros. Isso proporciona ao modelo RDF descrever quase tudo o que se encontra na web.
- **Propriedades**: são os atributos dos recursos. Possuem restrições para os tipos de recursos que podem ser aplicadas: tipos possíveis, faixa de valores, relacionamento com outras propriedades, entre outros.
- **Declarações**: é um conjunto formado por um recurso específico, a propriedade nomeada e o valor da propriedade. Essas três partes são chamadas respectivamente de sujeito, predicado e valor. O valor da propriedade pode ser um outro recurso, uma *string*, uma página da web ou um tipo de dado definido em XML.

Resumindo essa definição dos elementos, o RDF define triplas entre

objeto-propriedade-valor como primitivas básicas de modelagem e introduzem uma sintaxe padrão para elas (BONIFACIO, 2002).

Essa tripla é implementada em linguagem XML e pode ser representada através de grafos de arestas rotuladas, onde os nós são os recursos, as arestas são as propriedades e o elemento terminal se refere ao valor da propriedade (SANTOS, 2002).

```
(1)  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
(2)          xmlns:dc="http://purl.org/dc/elements/1.1/">
(3)    <rdf:Description
(4)      rdf:about="http://www.uel.br/pessoal/ailton/Trabalhos/Disserta%C3%A7ao%20de%20Mestrado-Ailton-Final.pdf">
(5)    <dc:title>Ontologias e Consulta Semântica:</dc:title>
(6)    <dc:subtitle>Uma Aplicação ao Caso Lattes</dc:subtitle>
(7)    <dc:creator>Ailton Sergio Bonifacio</dc:creator>
(8)    <dc:subject>Ontologias, DAML+OIL, Lattes, Metadados,Web Semântica</dc:subject>
(9)    </rdf:Description>
(10)  </rdf:RDF>
```

Figura 6 - Exemplo de recurso RDF em XML.

De acordo com a figura anterior, podem-se evidenciar os conceitos definidos anteriormente. Neste exemplo, o modelo RDF está sendo usado para descrever um documento digital com a ajuda do conjunto de metadados Dublin Core (dc). Ainda sobre a figura, é possível evidenciar os seguintes elementos.

A linha 1, com a *tag* <rdf:RDF>, declara que o bloco seguinte é uma expressão RDF, cujo formato e modelo se encontram na URI mencionada. Na linha 2 utiliza-se o atributo “xmlns” (XML *Namespaces*) para importar ao documento criado padrões já definidos, como o Dublin Core.

O próximo elemento <rdf:Description> indica a descrição de um recurso. Neste exemplo, o recurso a ser tratado é “http://www.uel.br/pessoal/ailton/Trabalhos/Disserta%C3%A7ao%20de%20Mestrado-Ailton-Final.pdf”. Esta propriedade indica que o elemento citado será descrito através das propriedades que serão definidas posteriormente.

As linhas seguintes (5 a 8) mostram essas propriedades para descrever o documento e seus valores. A linha 5 indica o título do documento representado pelo elemento *dc:title* e seu valor é “Ontologias e Consulta Semântica”. A linha 6 mostra o sub-título (*dc:subtitle*) do documento: “Uma aplicação ao Caso Lattes”.

As próximas linhas (7 e 8) mostram, respectivamente, o autor e o assunto do documento. A propriedade autor (*dc:*creator) possui como valor “Ailton Sergio Bonifacio” e a propriedade assunto (*dc:subject*) possui como valor “Ontologias, DAML+OIL, Lattes, Metadados, Web Semântica”.

Santos (2002) complementa que os documentos RDF podem ser representados em forma de grafo, o que tornaria mais fácil a compreensão por humanos. Com isso, o documento descrito anteriormente geraria o grafo rotulado

mostrado na figura 7.

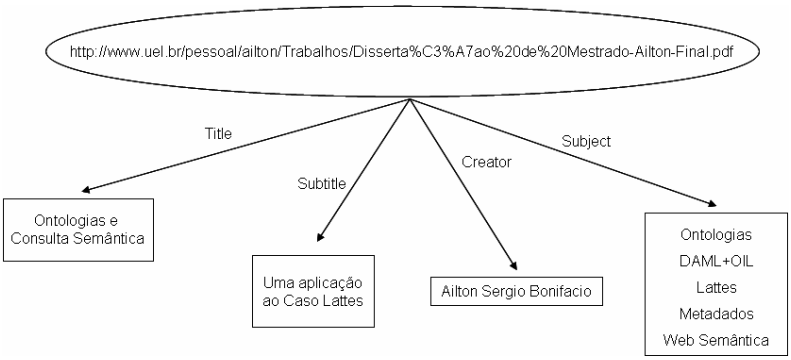

Figura 7 - Modelo de grafo rotulado gerado a partir do documento RDF anterior.

Ainda segundo Santos (2002), o grafo é formado por nós que representam os recursos, arestas que são as propriedades e elementos terminais (retângulos) que representam os valores de cada propriedade.

2.2.2 RDF Schema

O RDF Schema é uma extensão do modelo RDF e atua cobrindo as limitações que este modelo apresenta para representação das ligações das propriedades e dos outros recursos (BRITO; NOLETO, 2003).

Este modelo foi criado para complementar o modelo RDF básico com a finalidade de diminuir a interoperabilidade semântica no contexto da WEB. O RDF descreve de maneira simples os relacionamentos entre os recursos tratando-se de propriedades e valores.

As propriedades RDF são vistas como atributos de recurso, ou seja, são pares (atributo, valor) tradicionais e também representam relacionamentos entre recursos. Com isso, um modelo RDF pode ser considerado um Modelo Entidade-Relacionamento (MER).

O RDF Schema atua sobre o modelo RDF como um mecanismo que declara as propriedades do modelo RDF e fornece os relacionamentos entre essas propriedades e os recursos (ARAÚJO, 2003).

Além dessas funcionalidades, o RDF Schema propicia a criação de um vocabulário para o esquema representado através de classes e propriedades, o que possibilita reaproveitar o conteúdo em outros modelos (SILVA, 2004).

Para melhor compreensão do RDF Schema, serão apresentadas definições de classes, propriedades e restrições. As figuras abaixo representam,

respectivamente, um conceito de classes, subclasse e recursos e o diagrama de classes do modelo RDF Schema segundo Brickley e Guha (2000).

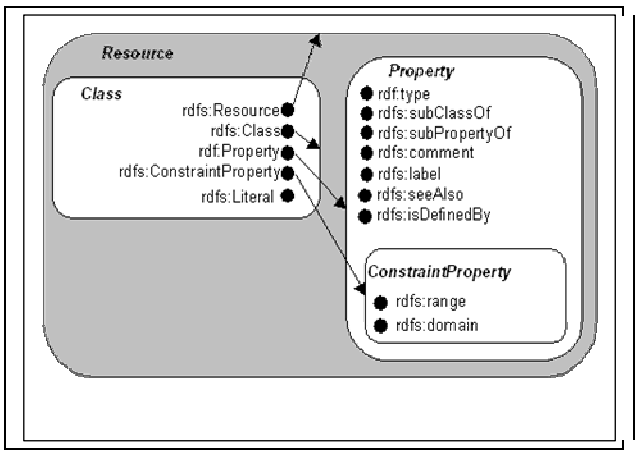

Figura 8 - Conceito de classe, subclasse e recurso (Brickley e Guha (2000)).

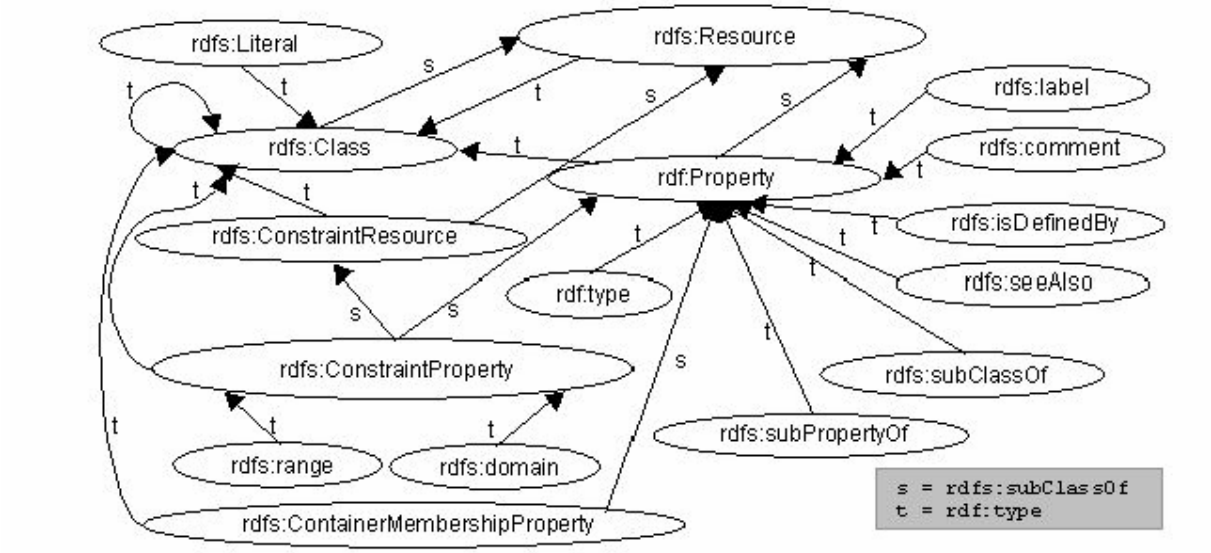

Figura 9 - Hierarquia de classes do modelo RDF Schema (Brickley e Guha (2000)).

2.2.2.1 Classes

Uma classe é um modelo criado a partir de uma descrição genérica de recursos aplicados sobre um determinado domínio. Após a sua definição, subclasses são geradas através do conceito de herança. Essa subclasse contém todos os métodos da superclasse e os seus métodos particulares. Esse conceito de herança possibilita o reuso de informações já existentes (SILVA, 2004).

Segundo Araújo (2003) e Santos (2002), as classes do modelo RDF Schema são:

- ***rdfs:Resource*:** classe genérica do modelo RDF Schema, pois todo objeto descrito é um recurso. Todas as demais classes são suas subclasses.
- ***rdfs:Class:*** representa o conceito de tipo ou categoria, semelhante às classes no paradigma orientado a objetos. Também é definido como um recurso, portanto, é uma subclasse de *rdfs:Resource*.
- ***rdfs:Property:*** este recurso define as propriedades de cada classe. Em orientação a objetos, pode ser comparado aos atributos de uma classe.

2.2.2.2 Propriedades

As propriedades são instâncias da classe *rdf:Property.* Essas propriedades utilizam mecanismos para estabelecer relacionamentos entre classes, instâncias e subclasses (ARAÚJO, 2003). Silva (2004) complementa que é possível verificar o relacionamento com outras propriedades gerando uma hierarquia, assim como são criadas hierarquias entre as classes.

As principais propriedades do RDF Schema são:

- ***rdfs:type*:** define que um recurso é membro de uma classe e, portanto, possui todas as suas características. Um recurso pode ser instância de mais de uma classe.
- ***rdfs:subClassOf:*** define a relação de subclasse/superclasse entre duas classes. Diferentemente do paradigma orientado a objetos, o RDF permite herança múltipla. Porém, existe uma restrição quanto à herança. Uma classe não pode ser subclasse de alguma subclasse sua e nem pode ser subclasse de si mesma, o que iria gerar um grafo de herança cíclica.
- ***rdfs:subPropertyOf:*** indica que uma propriedade é especialização de outra. Uma sub-propriedade pode ser especialização de várias propriedades.
- ***rdfs:seeAlso*** e ***rdf:isDefinedBy:*** a propriedade *rdfs:seeAlso* indica que um recurso pode fornecer informações adicionais sobre o recurso em questão. E a propriedade *rdf:isDefinedBy* é uma sub-propriedade de *rdfs:seeAlso* e indica o recurso que define o recurso em questão.
- ***rdfs:label*** e ***rdfs:comment:*** são utilizadas para inserir comentários no documento, para fins de documentação. A propriedade *rdfs:label* fornece um

nome ao recurso e a propriedade *rdfs:comment* é utilizada para inserir comentários.

2.2.2.3 Restrições

As restrições são mecanismos para representação dos recursos, permitindo associar as propriedades aos recursos criados. As principais restrições do modelo RDF Schema, segundo Araújo (2003), são:

- ***rdfs:ConstraintResource*** e ***rdfs:ConstraintProperty*****:** são as superclasses das restrições *rdfs:range* e *rdfs:domain*. São utilizadas para especificar restrições.
- ***rdfs:range:*** esta restrição define que o valor de uma propriedade deve possuir a uma determinada classe ou várias. Este valor deve ser sempre uma classe.
- ***rdfs:domain:*** define a qual classe a propriedade é aplicada. Várias classes podem fazer parte da mesma propriedade.

2.2.2.4 Desvantagens

Existem algumas desvantagens quando se utiliza o modelo RDF Schema. Como se sabe, o RDF Schema define um vocabulário que estrutura os metadados que descrevem os recursos na web. No entanto, é considerado um modelo simples se comparado com as linguagens de representação do conhecimento.

Segundo Broesksta *et al* (2001), o modelo não aplica uma semântica formal à web, não sendo possível desenvolver modelos ontológicos e de raciocínio. Heflin (2001), citado por Araújo (2003), complementa que falta ao RDF Schema um mecanismo para definição de axiomas, o que poderia proporcionar uma semântica mais robusta, restringindo o significado dos termos. Estes axiomas seriam usados para extrair informações implícitas nos dados. Essa extração é feita utilizando ontologias e linguagens de representação de ontologias.

2.3 OWL

Como apresentado anteriormente, a web atualmente apresenta problemas de localização, acesso, apresentação e manutenção da informação por parte dos usuários (BONIFACIO; HEUSER, 2002). Parte desse problema pode ser resolvido adotando algum método estruturando os dados de forma que os agente computacionais

compreendam os mesmos.

Uma maneira de representar estes dados é adotando o uso de ontologias. O uso de ontologias é motivado pelo fato de fornecer um vocabulário que permite estruturar e extrair a semântica dos documentos de modo que possam ser realizadas consultas semânticas sobre os mesmos (ARAÚJO, 2003). Silva (2004) complementa que o uso de ontologias estrutura o conhecimento de uma forma compartilhada e independente da aplicação utilizada.

Para o desenvolvimento de ontologias existem linguagens que trabalham em cima das tecnologias XML e/ou RDF Schema descritas anteriormente. Segundo Corcho e Pérez (2002), podemos citar as linguagens XOL (*Ontology Exchange Language*) e OML (*Ontology Markup Language*) que são baseadas na linguagem XML. Como exemplos de linguagens baseadas em RDF Schema podemos citar DAML (*Darpa Agent Markup Language*), OIL (*Ontology Inference Layer*) e DAML+OIL, que é uma junção das duas linguagens.

Para padronização dos métodos de desenvolvimento de ontologias, o W3C resolveu estabelecer como padrão a linguagem OWL, que é uma revisão da linguagem DAML+OIL. Essa padronização foi estabelecida visando o desenvolvimento da Web Semântica (BRITO; LUSTOSA; TEIXEIRA, 2004).

A OWL é uma linguagem para ontologias web designada para aplicações que necessitem processar o conteúdo das informações e não somente exibi-los. Por isso, a OWL é considerada uma linguagem de definição e instanciação de ontologias e é formada por classes, propriedades e suas instâncias (MCGUINNESS; SMITH; WELTY, 2004).

A sintaxe da OWL é baseada na linguagem XML, que é compatível com RDF e RDF Schema. A linguagem XML descreve os dados sintaticamente e o modelo RDF(S) estrutura esses dados de maneira semântica. Com isso, a OWL se apóia nestes recursos fazendo com que a máquina realize a inferência da semântica desses dados (SILVA, 2004).

Essa definição fica mais clara ao se analisar a Figura 10:

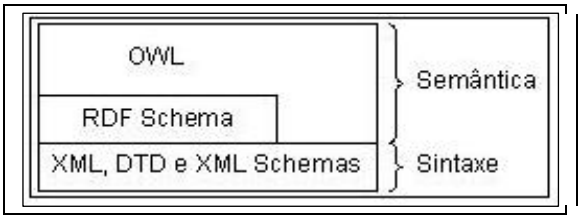

Figura 10 - Relação entre XML, RDF Schema e OWL

Esse exemplo mostra que ao se desenvolver uma ontologia, a OWL utiliza recursos definidos no RDF Schema e XML, além dos seus próprios recursos definidos. O uso destas tecnologias é justificado apresentando as suas funcionalidades. A linguagem é utilizada devido ao seu poder de extensibilidade, proporcionando a criação de inúmeras *tags* (GRAVES, 2003). Sobre ele, atua o modelo RDF Schema, que organiza um documento XML semanticamente, de modo que ele possa ser compreendido por máquinas (SANTOS, 2002). Ainda sobre a figura acima, pode-se incluir nesta definição o poder da linguagem OWL, que adiciona alguns recursos para a representação do conhecimento (SILVA, 2004).

2.3.1 Sub-linguagens OWL

Para facilitar o desenvolvimento de ontologias, o W3C dividiu a linguagem OWL em três sub-linguagens (MCGUINNESS; SMITH; WELTY, 2004):

- *OWL Lite*: é uma sub-linguagem da *OWL DL*. Possui algumas limitações que as outras duas sub-linguagens e usa algumas características da *OWL DL*.
- *OWL DL*: a sigla DL (*Description Logic*) significa, em português, Lógica Descritiva. Ela é utilizada para se extrair o máximo de expressividade, com completude e decidibilidade computacional. Ela possui todas as construções da linguagem OWL, que são usadas respeitando algumas restrições.
- *OWL Full*: não provê nenhuma garantia computacional, pois é utilizada por quem deseja o Maximo de expressividade e independência do modelo RDF. O diferencial entre a *OWL Full* e a *OWL DL* são as restrições, porque as duas suportam as mesmas construções de OWL. A *OWL Full* não requer a disjunção de classes, propriedades, indivíduos e valores e permite misturar OWL com RDF Schema. Por outro lado, a *OWL DL* requer disjunção entre os elementos e restringe o uso do modelo RDF.

Cada sub-linguagem apresenta é uma extensão da sua predecessora. Sendo assim, pode-se concluir que uma ontologia *OWL Lite* é válida como uma ontologia *OWL DL*, que por sua vez é válida como uma ontologia *OWL Full*. Em contrapartida, uma ontologia *OWL Full* não é uma ontologia *OWL DL* válida, e assim uma ontologia *OWL DL* não é uma ontologia *OWL Lite* válida. Essa relação é aplicada aos documentos das ontologias também, pois todo documento OWL é um documento RDF, e todo documento RDF é um documento *OWL Full* válido, e apenas alguns documentos RDF são documentos *OWL DL* ou *Lite* válidos.

2.3.2 Estrutura da linguagem OWL

Uma ontologia OWL é formada pelos seguintes componentes: Indivíduos, propriedades e classes. Esses conceitos na ferramenta Protégé são conhecidos como Instâncias, Slots e Classes (HORRIDGE *et al*, 2004). Indivíduos são instâncias das classes, Propriedades são os relacionamentos e Classes são conjuntos que contém os atributos para instanciar os Indivíduos (SOUTO; WARPECHOWSKI; OLIVEIRA, 2006).

Um documento OWL é composto necessariamente por uma declaração de *namespaces* e definição de classes, propriedades e indivíduos. Pode-se ainda acrescentar um cabeçalho contendo informações sobre a ontologia ou elementos que a compõem. A troca na ordem da apresentação dos elementos não influi em seu significado (LUSTOSA, 2003).

Os próximos itens explicam cada parte de um documento OWL com exemplos para demonstrar a sintaxe e a função de cada elemento.

2.3.2.1 Namespaces

Toda ontologia possui o seu *namespace*, que é conhecido como *default namespace*. Um *namespace* é um prefixo de uma classe, propriedade e indivíduos de uma ontologia (HORRIDGE *et al*, 2004). Outra função dos *namespaces* é evitar conflito entre vocabulários definidos em documentos diversos. Um *namespace* é identificado por um IRI (*Internationalized Resource Identifier*) que, diferentemente de uma URI, aceita a utilização de caracteres ASCII (*American Standard Code for Information Interchange*) (LUSTOSA, 2003).

Com isso, conclui-se que antes de utilizar algum termo de um vocabulário externo ao documento utilizado, é necessário que o vocabulário tenha sido declarado como um *namespace* do documento. Um exemplo de *namespaces* pode ser conferido na figura 11.

```
(1) <rdf:RDF xmlns="http://www.ffalm.br/ontologia.owl#"
(2)     xml:base="http://www.ffalm.br/ontologia.owl"
(3)     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
(4)     xmlns:p1="http://www.owl-ontologies.com/assert.owl#"
(5)     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
(6)     xmlns:owl="http://www.w3.org/2002/07/owl#"
(7)     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
```

Figura 11 - Exemplo de definição de *namespaces*

Como pode se observar, a figura acima é uma declaração de *namespaces* de um determinado documento OWL. As duas primeiras linhas identificam o *namespace* do documento citado. As demais linhas da declaração representam os vocabulários que serão utilizados no documento.

A linha 6 define que os elementos com o prefixo *owl:* façam referencia aos elementos da linguagem OWL, descrita no *namespace* indicado. Todas as classes de um documento OWL referenciam à classe *owl:Thing*, que está presente no vocabulário da linguagem (HORRIDGE *et al*, 2004).

Como a linguagem OWL é fundamentada sobre RDF, RDF Schema e XML Schema. Sendo assim, essas tecnologias precisam ser definidas na declaração dos *namespaces* do documento. Isso pode ser conferido nas linhas 3 (RDF Schema), 5 (XML Schema) e 7 (RDF).

Finalizando a definição dos *namespaces* do documento mencionado, restou a linha 4 da declaração. Este *namespace* faz referência ao documento *assert.owl* e possui o prefixo *p1*. Esse documento *assert.owl* é uma ontologia utilizada para definir afirmações sobre os recursos que é utilizada para testar ferramentas, como o *Protégé*.

2.3.2.2 Cabeçalho

O cabeçalho do documento contém informações relevantes sobre o mesmo. Essas informações devem ser incluídas dentro da *tag owl:Ontology*. Como informações do cabeçalho, pode-se colocar a versão do documento, nome dos desenvolvedores ou comentários para documentações (LUSTOSA, 2003).

```
(1)   <owl:Ontology rdf:about=""/>
```

Figura 12 - exemplo de cabeçalho sem informações sobre o documento

```
(1)   <owl:Ontology rdf:about="">
(2)         <rdfs:comment>Exemplo de cabeçalho de uma ontologia</rdfs:comment>
(3)         <owl:priorVersion rdf:resource="http://www.ffalm.br/versaoAntiga"/>
(4)         <owl:imports rdf:resource="http://www.ffalm.br/objetos"/>
(5)         <rdfs:label>Exemplo</rdfs:label>
(6)   </owl:Ontology>
```

Figura 13 - Exemplo de cabeçalho com informações sobre o documento

As figuras 12 e 13 mostram dois exemplos de cabeçalho. A figura 12 mostra um cabeçalho sem informações mas a *tag owl:Ontology* aparece no documento. As demais *tags* não são especificadas. Ao contrario da primeira figura, a figura 13 apresenta todos os elementos que fazem parte do cabeçalho.

As linhas 1 e 6 delimitam o tamanho do cabeçalho com as *tags* de abertura e fechamento *owl:Ontology*. A linha 2 define um comentário sobre o documento. Este valor deve ser colocado entre as *tags* de *rdfs:comment*. As linhas 3 e 4 utilizam elementos da linguagem OWL, que pode ser notado através do prefixo *owl:*. A linha 3 utiliza o elemento *owl:priorVersion*, que é utilizada para controlar as versões e a linha 4 estabelece importações de uma ontologia estabelecida por um recurso. Por fim, a linha 5 mostra o valor da propriedade *rdfs:label* que é utilizada para nomear elementos da linguagens como classes ou propriedades, por exemplo.

Analisando todas as propriedades envolvidas no cabeçalho de um documento, fica clara a importância dos *namespaces*. Observa-se que toda propriedade possui um prefixo e um nome. Esse prefixo são os *namespaces* definidos anteriormente e após ele aparece o nome do recurso em questão para identificação da propriedade.

2.3.2.3 Classes

A definição de uma classe em OWL é semelhante à declaração de classes de um sistema utilizando a linguagem UML (*Unified Modeling Language*) para modelagem. Segundo a UML, uma classe é utilizada para descrever objetos que compartilhem os mesmos atributos, operações e relacionamentos (BOOCH; RUMBAUGH; JACOBSON, 2005).

Essas classes serão úteis no processo de criação de instâncias. Fazendo uma analogia entre modelos, as instâncias podem ser comparadas a objetos criados na linguagem de programação Java. Em Java, deve-se definir os atributos, métodos e relacionamentos de uma classe e os objetos são instanciados a partir desses conceitos (DEITEL, 2004). Em OWL, esses conceitos são conhecidos como propriedades de uma classe.

Em OWL, toda classe criada é derivada da classe *owl:Thing*, sendo assim, a classe *owl:Thing* é superclasse de todas as classes de um documento. Com isso, as classes podem ser classificadas de maneira hierárquica possibilitando o relacionamento de herança entre elas. Existem também a classe *owl:Nothing* que não possui instâncias e é sub-classe de todas as classes OWL (CARVALHO; LIMA, 2005).

Para definir uma classe em OWL utiliza-se o elemento *owl:Class* e o nome da classe é indicado por *rdf:ID*, que contém o nome da classe. Essa definição pode ser conferida no exemplo da figura 14.

```
(1)    <owl:Class rdf:ID="Cidade">
```

Figura 14 - Exemplo de definição de classes

Na figura 14 pode-se notar os dois elementos: *owl:Class* e *rdf:ID*. Com isso pode se concluir que a declaração acima é de uma classe denominada de Cidade, indicado entre aspas. Esse nome da classe é utilizado para identificação da classe em questão.

Outro item importante é observar que a classe foi nomeada utilizando o modelo RDF através do elemento *rdf:ID*. Este elemento possibilita que a classe seja referenciada de maneira diferente (MCGUINNESS; SMITH; WELTY, 2004).

Essa referência é possível porque a classe é identificada como um recurso do documento criado definido como “*rdf:resource=#Cidade” (*SILVA, 2004*)*. Com isso, ele pode ser referenciado dentro do documento através da expressão “#Cidade”. Quando se tratar de um outro documento que importar o documento citado, essa importação será feita através da URI “http://www.ffalm.br/ontologia.owl#Cidade”. Com isso, é correto afirmar que o *namespace* do documento anterior é ”http://www.ffalm.br/ontologia.owl” e o recurso é “#Cidade”.

Ainda sobre classes, é possível se estabelecer subclasses para especializar algum conceito muito genérico. Esta definição de sub-classes organiza hierarquicamente um documento OWL, partindo da classe mais genérica até as suas

especializações (LUSTOSA, 2003).

A declaração de sub-classes é realizada através do elemento *rdfs:subClassOf* (CARVALHO; LIMA, 2005). Esse conceito de sub-classes em OWL é semelhante ao conceito de herança da linguagem Java, onde a classe mais genérica (classe-mãe) pode ser especializada em outras classes mais específicas (classes-filhas) (DEITEL, 2004).

```
(1)    <owl:Class rdf:ID="ComissaoOrganizadora">
(2)        <rdfs:subClassOf rdf:resource="#EventoCientifico"/>
(3)        ...
(4)    </owl:Class>
```

Figura 15 - Exemplo de declaração de subclasse

A definição de subclasses acima é exemplificada na figura 15. Na figura citada, há a criação da classe ComissaoOrganizadora (linhas 1 e 4), que é um recurso RDF identificado dentro do documento como “#ComissãoOrganizadora”. Segundo a modelagem estabelecida para a implementação do exemplo, uma Comissão organizadora faz parte de um evento, ou seja, pode ser considerada uma subclasse de EventoCientifico.

Esse conceito é definido na linha 2 do exemplo, onde mostra que a classe ComissaoOrganizadora é subclasse do recurso “#EventoCientifico”, que em alguma parte do documento, este recurso é definido como uma classe. Essa relação é definida através do elemento *rdfs:subClassOf*.

A linha 3 mostra itens que foram desconsiderados do documento. Esses itens não seriam interessantes para este exemplo, mas para constar poderiam estar definidos nestas linhas propriedades ou outras declarações de subclasses.

2.3.2.4 Propriedades

Segundo Horridge *et al* (2004), existem dois tipos principais de propriedades: *Object* e *Datatype properties*. Cada uma possui uma função diferente. O que há em comum entre elas é que ambas são usadas para representar o relacionamento entre duas instâncias.

A função de uma *object property* é relacionar duas classes como, por exemplo, as classes Cidade e Estado. Esse relacionamento pode ser estabelecido através da propriedade pertenceA.

O outro tipo de propriedade (*Datatype property*) é usado para ligação entre uma classe e um tipo de dado definido pelo XML Schema ou um literal RDF. Este tipo de propriedade é utilizado para criação dos campos de uma classe, que serão preenchidos com as características dos indivíduos.

A primeira restrição sobre as propriedades é sobre os valores que ela poderá assumir. Essa restrição fica por conta dos elementos *rdfs:domain* e *rdfs:range*. Com isso, o *domain* seria o domínio da propriedade e o *range* indica os o conjunto de valores que a propriedade pode assumir.

```
(1)  <owl:ObjectProperty rdf:ID="pertenceA">
(2)      <rdfs:domain rdf:resource="#Cidade"/>
(3)      <rdfs:range rdf:resource="#Estado"/>
(4)  </owl:ObjectProperty>
```

Figura 16 - Exemplo de *object property*

```
(1)  <owl:DatatypeProperty rdf:ID="nomeCidade">
(2)      <rdfs:domain rdf:resource="#Cidade"/>
(3)      <rdfs:range rdf:resource="&xsd;string"/>
(4)  </owl:ObjectProperty>
```

Figura 17 - Exemplo de *datatype property.*

Na figura 16 é possível perceber os elementos responsáveis pelas definições das propriedades. Na linha 1, é atribuído um nome à propriedade. No caso, o nome da propriedade é pertenceA.

Nas linhas 2 e 3 são especificados , respectivamente, *domain* e *range* da propriedade. O *domain* é a classe Cidade e o *range* é a classe Estado. Com isso, pode-se afirmar que os elementos da classe Cidade serão relacionados com a classe Estado através da propriedade pertenceA, fazendo com que uma cidade pertença a um determinado estado.

Para a definição de uma *datatype property*, é necessário definir os mesmos atributos de uma *object property*: nome (linha 1), *domain* (2) e *range* (3). A diferença na definição do *range* fica no valor da propriedade. Uma *datatype property* faz referência a um tipo de dado.

Essa definição pode ser notada na figura 17. Nesta figura, nota-se que valor de *range* é do tipo string definido em XML Schema. Sendo assim, uma classe cidade possui uma propriedade de instância definida como nomeCidade que receberá valores do tipo string.

Na definição das propriedades, devem ser observadas suas características. De acordo com Horridge *et al* (2004), as características das propriedades são usadas para enriquecer o significado das propriedades.

Lustosa (2003) organiza hierarquicamente as propriedades conforme a figura 18. Essa organização hierárquica possui o elemento *rdf:Property* como elemento raiz.

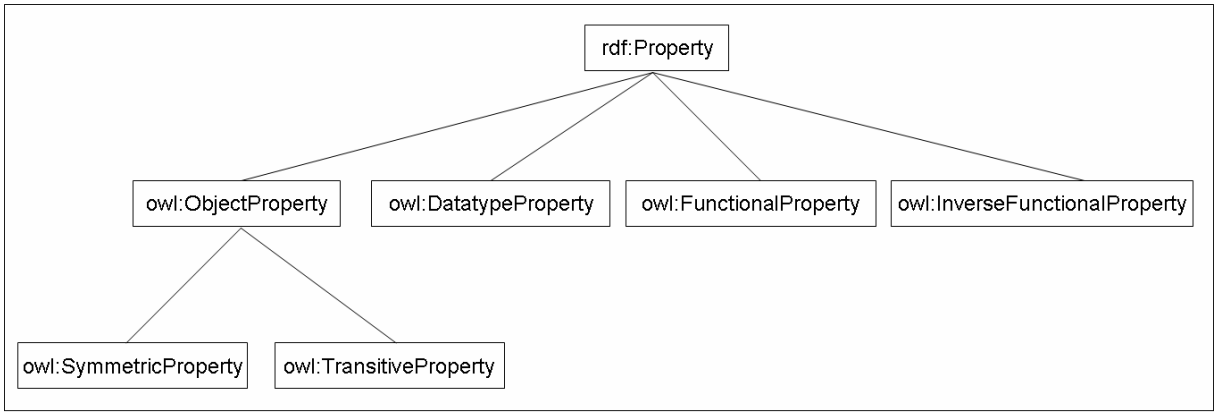

Figura 18 - Hierarquia das propriedades (LUSTOSA, 2003).

As características das propriedades de acordo com a figura 18 são:

- *owl:FunctionalProperty;*
- *owl:InverseFunctionalProperty;*
- *owl:SymmetricProperty;*
- *owl:TransitiveProperty.*

Ainda de acordo com a figura 18, conclui-se que as propriedades *owl:TransitiveProperty* e *owl:SymmetricProperties* são subclasses de *owl:ObjectProperty*, e com isso o *range* da propriedade é uma classe. Com isso, essas propriedades são utilizadas para relacionarem apenas classes.

No entanto, as propriedades *owl:FunctionalProperty* e *owl:InverseFunctionalProperty* podem relacionar tanto classes entre si ou classes com literais RDF ou algum tipo de dado XML Schema. Os exemplos e as definições destas características serão apresentados abaixo.

```
(1)   <Estado rdf:ID="Rio Grande do Sul">
(2)     <temNaturalidade rdf:resource="#rio-grandense-do-sul"/>
(3)   </Estado>
(4)
(5)   <Estado rdf:ID="Rio Grande do Sul">
(6)     <temNaturalidade rdf:resource="#Gaucho"/>
(7)   </Estado>
(8)
(9)   <owl:ObjectProperty rdf:ID="temNaturalidade">
(10)    <rdf:type rdf:resource
(11)        "http://www.w3.org/2002/07/owl#FunctionalProperty"/>
(12)    <rdfs:domain rdf:resource="#Estado"/>
(13)    <rdfs:range rdf:resource="#Naturalidade"/>
(14)  </owl:ObjectProperty>
```

Figura 19 - Exemplo de *Functional Property*.

A primeira característica apresentada é denominada de *Functional Properties*. De acordo com Carvalho e Lima (2005), essa característica é caracterizada por apresentar cardinalidade mínima igual a 0 (zero) e cardinalidade máxima igual a 1 (um). Essa definição fica clara na figura 19 logo acima.

Pode-se observar na figura 19 que o documento está dividido em três partes para apresentação. Na primeira parte, entre as linhas 1 e 3, define-se uma instância da classe estado chamada Rio Grande do Sul. A propriedade instanciada em questão é temNaturalidade com o valor “rio-grandense-do-sul”. Na segunda parte, entre as linhas 5 e 7, aparece outra instância da classe Estado. Desta vez, o valor da propriedade tem Naturalidade é “Gaucho”. As linhas subseqüentes exibem a definição da propriedade temNaturalidade. A característica da propriedade é definida nas linhas 10 e 11.

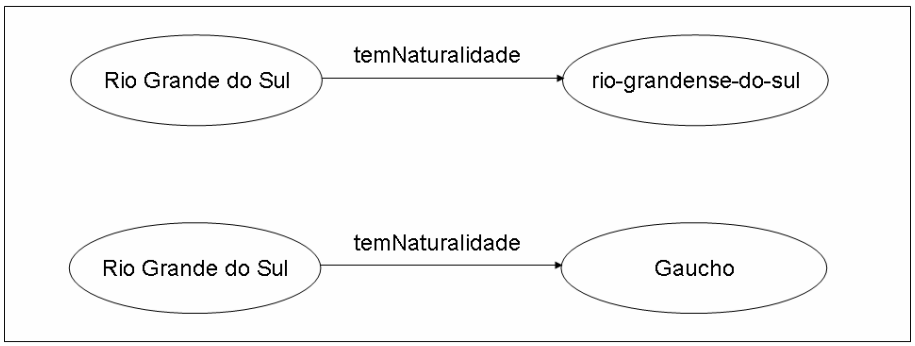

Figura 20 - Exemplo para representação de propriedade funcional.

Sobre essas informações, é possível concluir que as instâncias da classe Estado contendo “rio-grandense-do-sul” e “Gaucho” são iguais, como mostra a figura 20. Essa conclusão é fundamentada por se tratar de uma propriedade funcional. Com isso, é permitido uma relação entre as instâncias da classe *X* (Estado) e da classe

Y (Naturalidade).

Inversamente a esta propriedade, pode-se definir que uma propriedade é funcional inversa (*InverseFunctional Property*). Neste caso, pode-se afirmar que uma propriedade representa uma única instância (CARVALHO; LIMA, 2005). Silva (2004) exemplifica esta afirmação dizendo que se uma classe *X* possui relacionamento do tipo funcional inverso com a classe *Y*, pode-se garantir que uma instância de *X* pode-se relacionar com apenas uma instância de *Y*.

```
(1)  <owl:DatatypeProperty rdf:ID="quemNase">
(2)      <rdfs:domain rdf:resource="#Naturalidade"/>
(3)      <rdfs:range rdf:resource="#Estado"/>
(4)      <rdf:type rdf:resource
(5)           "http://www.w3.org/2002/07/owl#FunctionalProperty"/>
(6)      <rdf:type rdf:resource
(7)               "http://www.w3.org/2002/07/owl#InverseFunctionalProperty"/>
(8)  </owl:DatatypeProperty>
```

Figura 21 - Exemplo de *InverseFunctional Property*.

O exemplo da figura 21 apresenta a propriedade quemNasce, que é declarada como propriedade funcional inversa. Sendo assim, a instância "rio-grandense-do-sul" se encontra relacionada através da propriedade funcional inversa quemNasce com "Rio Grande do Sul". Logo, a instância "rio-grandense-do-sul" não pode se relacionar com nenhuma outra instância.

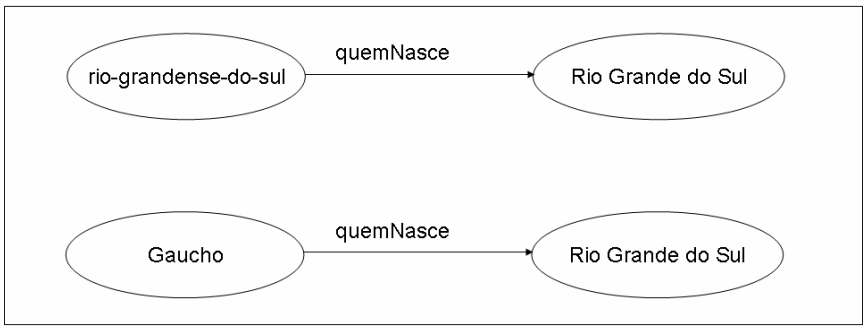

Figura 22 - Exemplo de representação de propriedade funcional inversa.

De acordo com a figura 22, caso a propriedade quemNasce tenha sido declarada como funcional inversa, pode-se concluir que as instâncias são iguais pelo fato de referenciaram a mesma instância (Rio Grande do Sul).

Para referenciar que uma propriedade é inversa de outra utilizamos o elemento *inverseOf*. De acordo com esta definição, pode-se afirmar que uma instância *X* se relaciona com uma instância *Y* através da propriedade *P*, que é inversa de *P1*. com

isso, pode-se afirmar que a propriedade *P1* relaciona as instâncias *Y* e *X*.

```
(1)   <owl:ObjectProperty rdf:ID="pertenceA">
(2)      <rdf:type rdf:resource="&owl;FunciontalProperty"/>
(3)   </owl:ObjectProperty>
(4)
(5)   <owl:ObjectProperty rdf:ID="formadoPor">
(6)      <owl:inverseOf rdf:resource="#pertenceA"/>
(7)   </owl:ObjectProperty>
```

Figura 23 - Exemplo de propriedade inversa.

A figura 23 acima mostra um exemplo de uma propriedade inversa. De acordo com a figura, a propriedade pertenceA é declarada como *object property* e *FunctionalProperty*. Logo abaixo, a propriedade formadoPor é definida como propriedade inversa de pertenceA. A representação gráfica da propriedade é mostrada abaixo na figura 24.

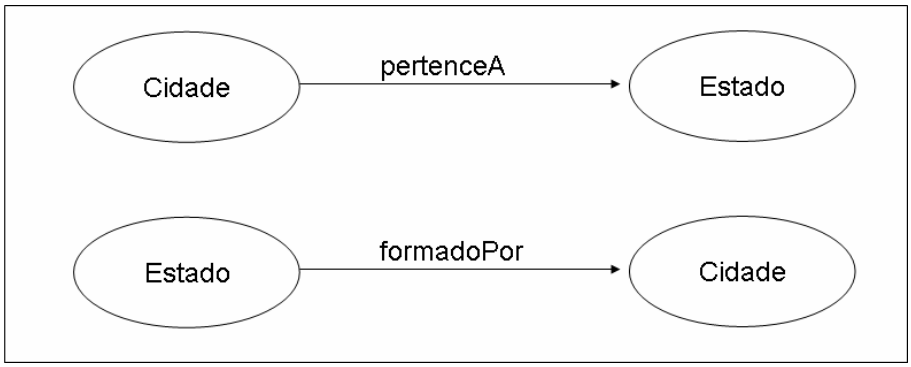

Figura 24 - Exemplo de representação de propriedade inversa.

Uma propriedade simétrica (*Symmetric Property*) é utilizada quando uma instância de *X* se relaciona com uma instância de *Y* através da propriedade *P*. Se essa propriedade for definida como simétrica, então a instância de *Y* também se relaciona com a instância de *X* através da propriedade *P* (HORRIDGE *et al*, 2004).

```
(1)   <owl:ObjectProperty rdf:ID="fazFronteira">
(2)     <rdf:type rdf:resource
(3)         "http://www.w3.org/2002/07/owl#SymmetricProperty"/>
(4)     <rdfs:domain rdf:resource="#Pais"/>
(5)     <rdfs:range rdf:resource="#Pais"/>
(6)   </owl:ObjectProperty>
(7)
(8)   <Pais rdf:ID="Brasil">
(9)      <locatedIn rdf:resource="#fazFronteira"/>
(10)     <adjacentRegion="#Uruguai"/>
(11)  </Pais>
```

Figura 25 - Exemplo de *Symmetric Property*.

Como apresentado na figura 25, a propriedade fazFronteira é declarada como simétrica e se auto-relaciona a classe País como pode ser conferido na definição das linhas 4 e 5. Com isso, ao instanciar a classe Brasil, a propriedade fazFronteira recebe como valor o recurso "#Uruguai", que é uma instância da classe País. Como a propriedade é declarada como simétrica, pode-se afirmar que a instância de Uruguai também relacionará com Brasil através da propriedade fazFronteira, como explica o grafo da figura 26.

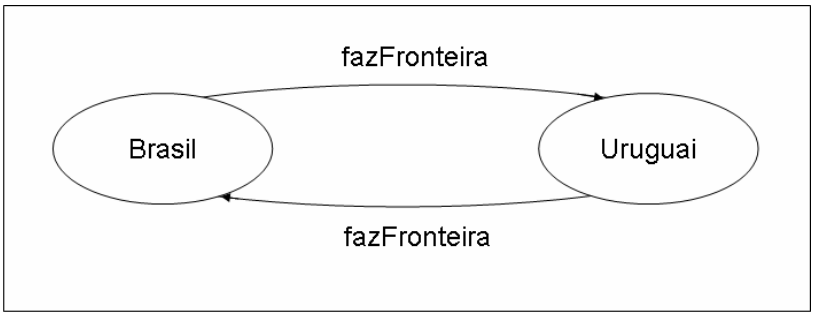

Figura 26 - Exemplo de representação de propriedade simétrica.

A última característica a ser definida a uma propriedade é conhecida como *Transitive Property*. Com uma propriedade definida como transitiva é possível relacionar duas instâncias sem que as mesmas estejam diretamente ligadas (HORRIDGE *et al*, 2004).

```
(1)   <owl:ObjectProperty rdf:ID="localizadoEm">
(2)      <rdf:type rdf:resource=
(3)         "http://www.w3.org/2002/07/owl#TransitiveProperty"/>
(4)      <rdfs:domain rdf:resource="http://www.w3c.org/2002/07/owl#Thing"/>
(5)      <rdfs:range   rdf:resource="#Regiao"/>
(6)   </owl:ObjectProperty>
(7)
(8)   <Regiao rdf:ID="Jalapao">
(9)      <localizadoEm rdf:resource="#Tocantins"/>
(10)  </Regiao>
(11)
(12)  <Regiao rdf:ID="Tocantins">
(13)     <localizadoEm rdf:resource="#Brasil"/>
(14)  </Regiao>
(15)
(16)  <owl:TransitiveProperty rdf:ID="localizadoEm">
(17)     <rdfs:domain rdf:resource="#http://www.w3c.org/2002/07/owl#Thing"/>
(18)     <rdfs:range rdf:resource="#Regiao"/>
(19)  </owl:TransitiveProperty>
```

Figura 27 - Exemplo de *Transitive Property.*

O exemplo da figura 27 pode ser definido em três partes. A primeira parte (linhas 1 a 6) mostra a definição da *object property* localizadoEm. A segunda parte contém a criação de duas instâncias. A primeira é entre as linhas 8 e 10 criando a instância "Jalapão" com a propriedade localizadoEm recebendo o recurso "*#Tocantins*". A segunda instância é de Tocantins e a propriedade localizadoEm recebe o recurso "*#Brasil*".

A terceira parte mostra a definição da propriedade localizadoEm com transitiva. Com isso, sobre o exemplo acima é possível inferir que a instância "Jalapão" está localizada em "Tocantins" e a instância "Tocantins" está localizada no "Brasil". Logo, através da transitividade de propriedade, pode-se concluir que "Jalapão" está localizado no "Brasil".

Esse conceito de transitividade pode ser conferido na figura 28 logo abaixo.

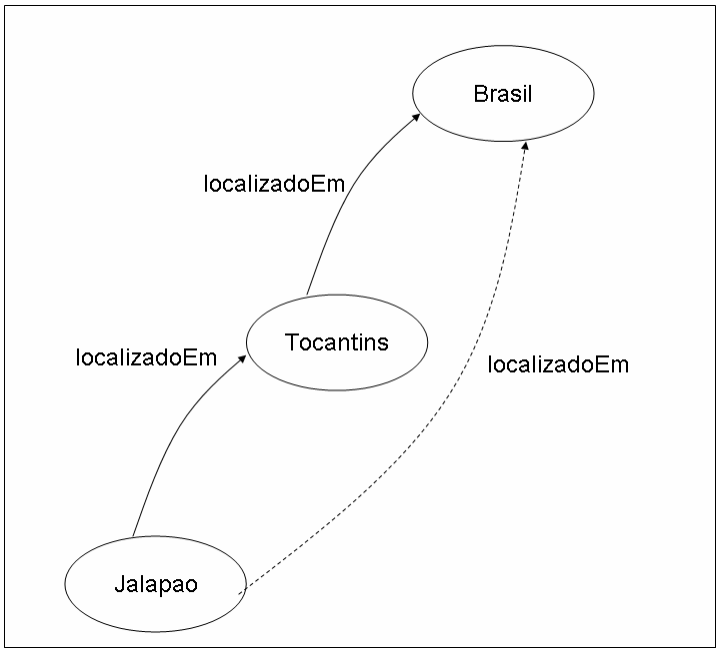

Figura 28 - Exemplo de representação de propriedade transitiva.

Para terminar o estudo sobre propriedades, o último item a ser abordado são as restrições que podem ser estabelecidas sobre elas. Essas restrições podem ser em relação ao valor que elas podem assumir ou quanto a cardinalidade (CARVALHO; LIMA, 2005).

Para delimitar os valores que uma propriedade pode assumir, temos as restrições *allValuesFrom*, *someValuesFrom* e *hasValue*. Para restrições de cardinalidade temos *maxCardinality*, *minCardinality* e *Cardinality*. Cada propriedade será apresentada logo abaixo.

A primeira restrição a ser apresentada é *allValuesFrom*. Ela indica que as instâncias das classes que possuam restrição definida são membros da classe referenciada por ela (HORRIDGE *et al*, 2004).

```
(1)   <owl:Class rdf:ID="Moveis">
(2)     <rdfs:subClassOf>
(3)       <owl:Restriction>
(4)         <owl:onProperty rdf:resource="#temFabricante"/>
(5)         <owl:allValuesFrom rdf:resource="#FabricaDeMoveis"/>
(6)       </owl:Restriction>
(7)     </rdfs:subClassOf>
(8)     ...
(9)   </owl:Class>
```

Figura 29 - Exemplo da restrição *allValuesFrom*.

Na figura 29, pode se notar que a restrição *allValuesFrom* indica para o recurso "*#FabricaDeMoveis*". Segundo essa restrição, fica evidente que todo fabricante de móveis deve ser uma "FabricaDeMoveis". Essa restrição é aplicada somente a esta classe. Caso a propriedade seja aplicada a outra classe, ela não é considerada válida.

A próxima restrição a ser apresentada é *someValuesFrom*. Esta restrição é semelhante a anterior. No entanto, o que as difere é o fato desta restringir que cada instância possua pelo menos um valor da classe descrita na restrição.

```
(1)   <owl:Class rdf:ID="Moveis">
(2)     <rdfs:subClassOf>
(3)       <owl:Restriction>
(4)         <owl:onProperty rdf:resource="#temFabricante"/>
(5)         <owl:someValuesFrom rdf:resource="#FabricaDeMoveis"/>
(6)       </owl:Restriction>
(7)     </rdfs:subClassOf>
(8)     ...
(9)   </owl:Class>
```

Figura 30 - Exemplo da restrição SomeValuesFrom.

Como exemplo desta restrição, podemos utilizar o exemplo anterior modificando apenas a restrição (linha 5). De acordo com este exemplo, conclui-se que cada instância da classe definida (Moveis) possua pelo um valor da classe definida pela restrição (FabricaDeMoveis). Com isso, é possível dizer que pelo menos um fabricante referencia uma instancia de FabricaDeMoveis. Mas podem aparecer fabricantes que não são.

A última restrição que pode ser aplicada para delimitar os valores de uma propriedade é *hasValue.* Carvalho e Lima (2005) define que uma propriedade tenha uma instância como valor. Sendo assim, uma instância será membro de tal classe quando pelo menos um dos valores das propriedades for igual ao valor da restrição *hasValue.*

```
(1)   <owl:Class rdf:ID="Brasileiro">
(2)     <rdfs:subClassOf>
(3)       <owl:Restriction>
(4)         <owl:onProperty rdf:resource="#produzidoEm"/>
(5)         <owl:hasValue rdf:resource="#Brasil"/>
(6)       </owl:Restriction>
(7)     </rdfs:subClassOf>
(8)   </owl:Class>
```

Figura 31 - Exemplo da restrição *hasValue.*

A figura 31 define que qualquer coisa que seja produzida no Brasil será considerada com Brasileira.

O outro tipo de restrição sobre as propriedades é em relação à cardinalidade. A definição da cardinalidade é importante para restringir a quantidade de valores de uma propriedade. (LUSTOSA, 2003). Em OWL, existem três tipos de restrições em respeito a cardinalidade:

- *owl:Cardinality;*
- *owl:maxCardinality;*
- *owl:minCardinality;*

A restrição *owl:Cardinality* define que as instâncias tenham um número pré-determinado de valores relacionados. A restrição *owl:maxCardinality* define o número máximo de instâncias que a propriedade poderá assumir. Finalizando, a restrição *owl:minCardinality* define a quantidade mínima de instâncias da propriedade.

Este conceito de cardinalidade mínima pode ser descrito como em um Banco de Dados. Em um Banco de Dados, quando a cardinalidade mínima é especificada como 0 (zero) indica que o preenchimento do campo é opcional. E quando é definida como 1 (um), indica que o preenchimento do campo é obrigatório com pelo menos um valor (HEUSER, 2004).

```
(1)  <owl:Class rdf:ID="Cidade">
(2)    <rdfs:subClassOf>
(3)      <owl:Restriction>
(4)        <owl:onProperty rdf:resource="#nomeCidade"/>
(5)        <owl:cardinality rdf:datatype="&xsd;int">1</owl:cardinality>
(6)      </owl:Restriction>
(7)    </rdfs:subClassOf>
(8)    ...
(9)  </owl:Class>
```

Figura 32 - Exemplo de *owl:Cardinality*.

A figura 32 acima explica a restrição de *owl:Cardinality*. Como aparece no exemplo, uma instância da classe Cidade pode conter apenas um valor para a propriedade nomeCidade. Esse fato ocorre porque uma cidade deve apresentar apenas um nome.

```
(1)   <owl:Class rdf:ID="TimeFutebol">
(2)     <rdfs:subClassOf>
(3)       <owl:Restriction>
(4)         <owl:onProperty rdf:resource="#possuiJogadores"/>
(5)         <owl:minCardinality rdf:datatype="&xsd;int">11</owl:minCardinality>
(6)         <owl:maxCardinality rdf:datatype="&xsd;int">18</owl:maxCardinality>
(7)       </owl:Restriction>
(8)     </rdfs:subClassOf>
(9)     ...
(10)  </owl:Class>
```

Figura 33 - Exemplo de *owl:minCardinality* e *owl:maxCardinality*.

A figura 33 mostra um exemplo das restrições de *owl:minCardinality* e *owl:maxCardinality*. Neste exemplo, essas restrições indicam que um time de futebol necessita de no mínimo 11 e no máximo 18 jogadores para realizar uma partida. Com isso, pode-se concluir que a propriedade possuiJogadores deve possuir entre 11 e 18 instâncias.

2.3.2.5 Instâncias

Uma instância é o nome recebido por indivíduos definidos a partir de uma classe. Essas instâncias são criadas através de axiomas de um determinado domínio. As instâncias se relacionam com outras instâncias através das propriedades (CARVALHO; LIMA, 2005).

Uma notificação deve ser feita em relação a instâncias em OWL e no *Protégé*. A linguagem OWL não utiliza *Unique Name Assumption* (UNA). Com isso, dois nomes diferentes podem se referir à mesma instância (HORRIDGE *et al*, 2004). Por exemplo, as instâncias de “Pelé” e “Edson Arantes do Nascimento” podem se referir a mesma pessoa. Com isso, em OWL deve-se especificar que os indivíduos são iguais ou diferentes uns dos outros.

Como pôde ser notado, durante a apresentação da linguagem OWL foram criadas algumas instâncias para representação das propriedades. As figuras 34 e 35 mostram criações de instâncias. Uma instância pode ser criada através da satisfação das propriedades relacionadas a ela.

```
(1)    <Cidade rdf:ID="Bandeirantes"/>
```

Figura 34 - Exemplo de criação de instância.

Na figura 34 acima, a instância “Bandeirantes” é criada a partir da classe Cidade. Com isso, fica claro que Bandeirantes é uma cidade. Neste caso, não há nenhuma propriedade. Se uma instância apresentar propriedades, a declaração será como apresentada na figura 35.

```
(1)    <Cidade rdf:ID="Bandeirantes">
(2)        <nomeCidade rdf:resource="#Bandeirantes"/>
(3)        <pertenceA rdf:resource="#Paraná"/>
(4)    </Cidade>
```

Figura 35 - Exemplo de criação de instância e propriedades.

Complementando o a figura anterior, pode-se notar que foram adicionadas duas propriedades à instância: nomeCidade e pertenceA. Cada propriedade recebe um valor, de acordo com as restrições impostas na sua definição. Na linha 2 há uma propriedade do tipo *Datatype Property* que recebe como valor o recurso

"*#Bandeirantes*". Na próxima linha pode-se notar a existência de uma propriedade do tipo *object property*. neste caso, esta propriedade se relaciona com a instância de outra classe definida pelo recurso "*#Paraná*".

2.4 Protégé-OWL 3.2.1

A ferramenta *Protégé* é um projeto desenvolvido pela Universidade de Stanford. A principio, o projeto do *Protégé* era uma ferramenta de aquisição de conhecimento limitada ao *Oncocin,* um sistema especialista para oncologia. Essa ferramenta, aos poucos, foi se modernizando para acompanhar a evolução dos sistemas baseados em conhecimento (SBC).

Para se desenvolver uma ontologia usando a linguagem OWL, é necessário que se faça uso de alguma ferramenta que provenha suporte para linguagem OWL. Uma ferramenta que atende a essas necessidades é o *Protégé*. A versão mais recente desta ferramenta é 3.3 que ainda se encontra em período de testes (STANFORD).

Para o desenvolvimento deste trabalho, optou-se por utilizar a versão 3.2.1 do *Protégé*. Optou-se por utilizar esta ferramenta devido à recomendação do W3C, que a utiliza para promoção da Web Semântica (W3C, 2001). Esta versão foi adotada em declínio da mais recente devido ao fato que esta ainda se encontra em versão de testes.

Para o desenvolvimento de ontologias OWL utilizando o *Protégé*, é necessário que se utilize o *plugin* OWL. Para as versões anteriores do *Protégé* era necessário instalar este *plugin* a parte. Este *plugin* é necessário para criação e edição de ontologias e é baseado em projetos anteriores como o RDF, OilTab e DAML+OIL (LUSTOSA, 2003). A versão 3.2.1 já oferece este recurso agregado, não sendo necessário que se faça a instalação complementar.

Para utilização do *Protégé* é necessário que se tenha instalada alguma versão da JVM (*Java Virtual Machine*). Esta máquina virtual se encontra atualmente na versão 1.6.0, mas versões mais antigas podem ser utilizadas para execução do *Protégé.*

As principais vantagens de se utilizar esta ferramenta são (STANFORD):

- Criar, carregar e salvar ontologias OWL utilizando RDF, RDF Schema, XML Schema entre outros.

- Editar e visualizar classes, propriedades e regras SWRL (*Semantic Web Rule Language*)
- Definir classes lógicas usando expressões OWL.
- Executar raciocinadores como classificadores de lógica descritiva.
- Editar instâncias OWL para a marcação da Web Semântica.

Além dessas características, o *Protégé* possui uma arquitetura flexível, sendo possível configurar e extender as suas funcionalidades. O *Protégé* ainda é integrado com *Jena*, que é uma API (*Application Programming Interface*) Java para aplicações para Web Semântica, e uma API Java para o desenvolvimento de interface com o usuário para serviços da Web Semântica (STANFORD).

Como já definido e brevemente justificado, a versão utilizada para o desenvolvimento da OntoEvento é a 3.2.1. Para melhor compreensão das funcionalidades da ferramenta, serão apresentadas algumas telas do *Protégé* 3.2.1 e será criado um pequeno exemplo.

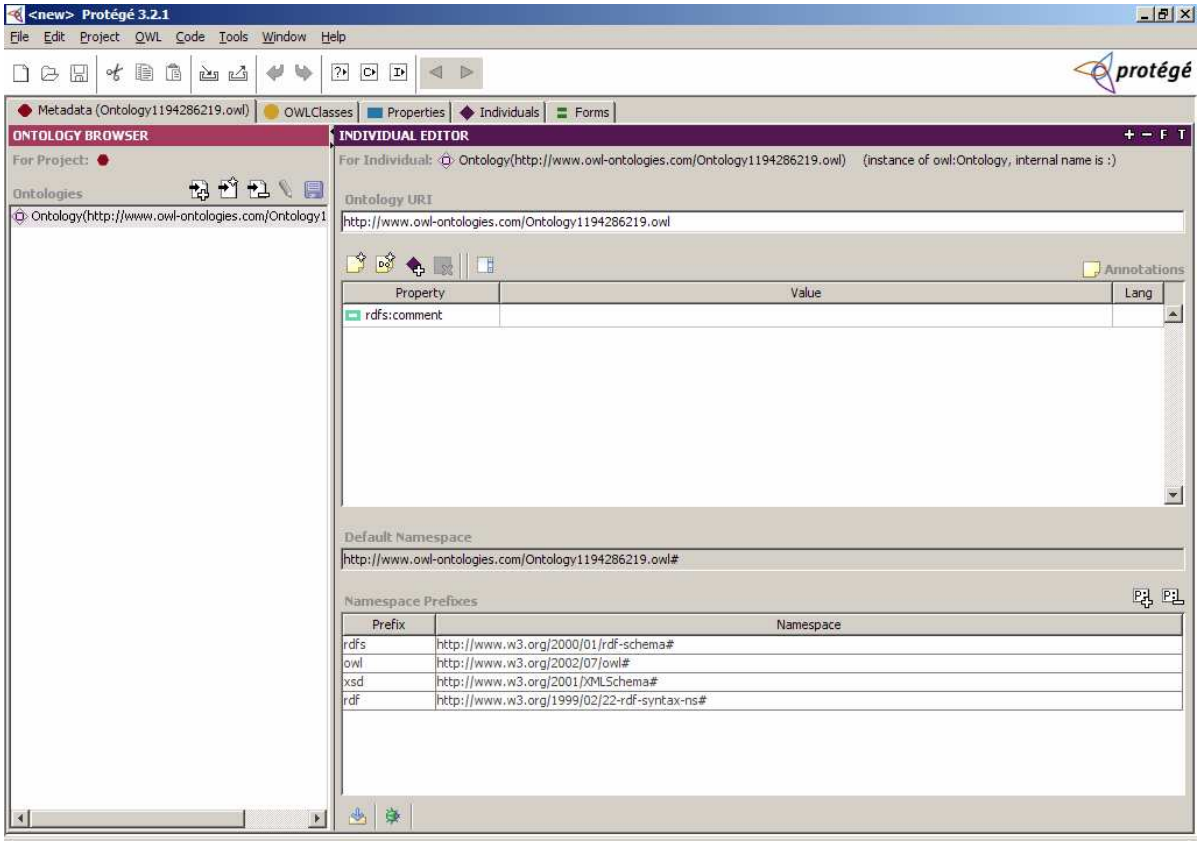

Figura 36 - Tela Inicial do *Protégé.*

A figura 36 mostra a tela inicial do *Protégé*. Ela é mostrada ao iniciar o uso da ferramenta. Nesta situação, é mostrada a aba *Metadata*, onde podem ser encontradas as guias *Ontology Browser* e *Individual Editor*. Na primeira guia são encontradas as ontologias importadas para dentro do projeto, para facilitar a navegação.

Na segunda guia (*Individual Editor*), é possível definir alguns recursos de uma ontologia. No campo *Ontology URI* é definido a URI do projeto. logo abaixo existem locais para inserção de comentários sobre a ontologias e definição de *namespaces*.

Na segunda aba (*OWLClasses*) são definidas as classes que farão parte do documento. Esta aba, representada na figura 37, possui as guias *Subclass Explorer* e *Class Editor*. Na guia *Subclass Explorer* é possível criar ou editar as classes e subclasses do documento, formando a hierarquia de classes. A guia *Class Editor* é usada para definição das restrições sobre as classes e propriedades.

Por ela, é possível se definir o nome das classes, as propriedades que fazem parte da classe, bem como as suas restrições, quais classes são disjuntas umas das outras o algum comentário sobre a classe.

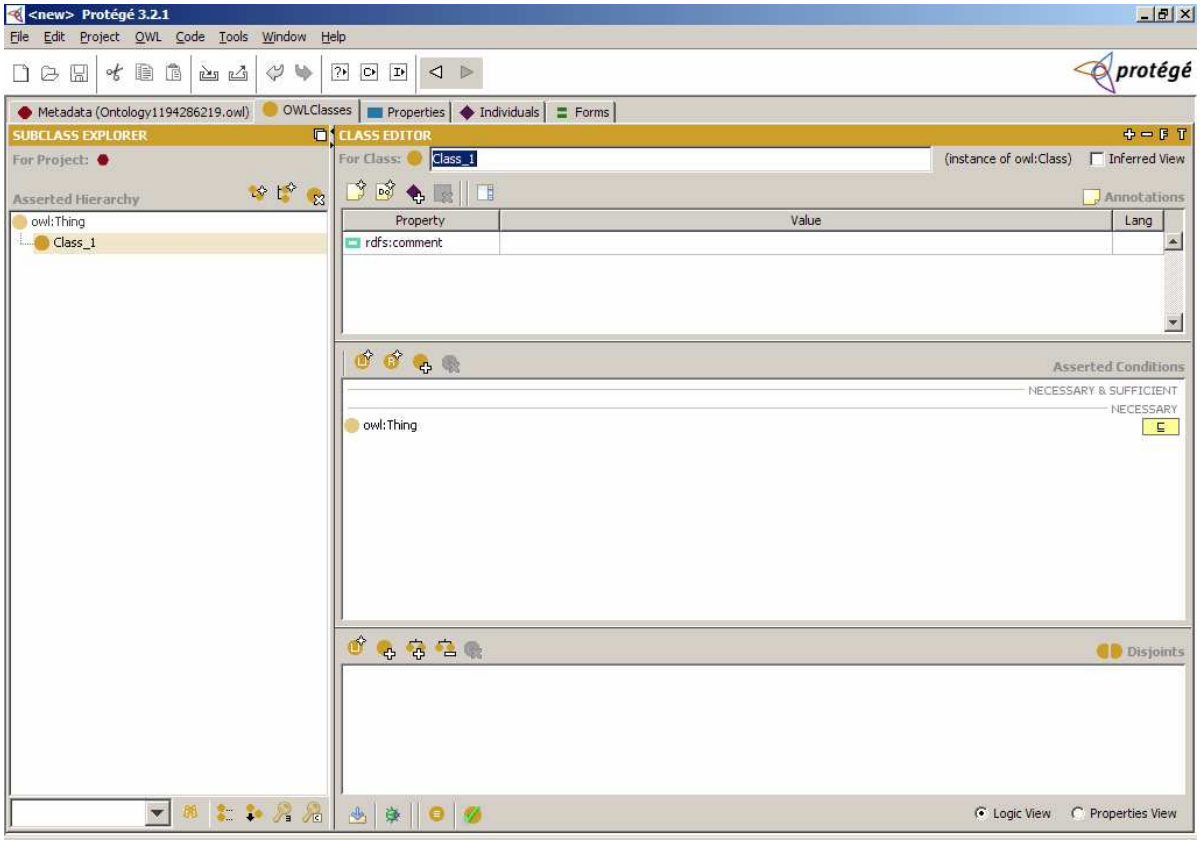

Figura 37 - Tela de definição e edição de classes.

Nesta tela, é possível observar que se encontram definidas duas classes: Pai e Filho. Elas serão utilizadas para melhor explicação da ferramenta *Protégé*. A próxima aba a ser descrita é *Properties* (Propriedades). Nesta tela são definidas as propriedades que farão parte do projeto.

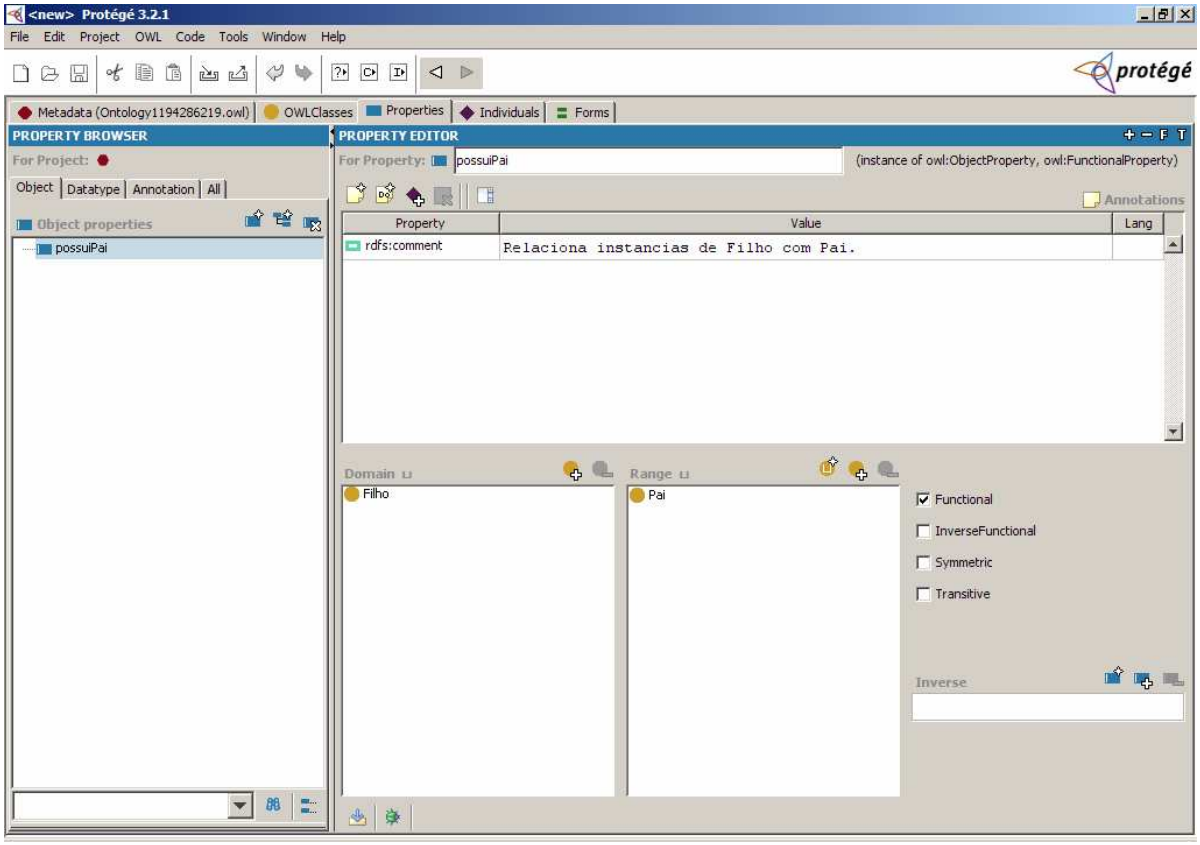

Figura 38 - Tela de definição das propriedades (*object properties*).

A figura 38 mostra a definição das propriedades de objeto, utilizadas para relacionamento entre classes. Como é observado, também existem duas guias nesta tela: *Property Browser* e *Property Editor*. Na primeira guia aparecem listadas as propriedades. Na segunda guia é possível estabelecer os atributos da propriedade selecionada.

No exemplo da figura 38, é criada a propriedade possuiPai para relacionar as classes Pai e Filho. Nesta propriedade, devem ser definidos o nome, os valores de *domain* (Filho) e *range* (Pai) e podem ser acrescentadas características à propriedade. Ainda é possível criar uma propriedade inversa.

Na aba *Property Browser* ainda há uma pequena aba contendo outros tipos de propriedades. Como as mais utilizadas são *object* e *datatype properties*, os exemplos sobre as demais no *Protégé* não serão apresentados.

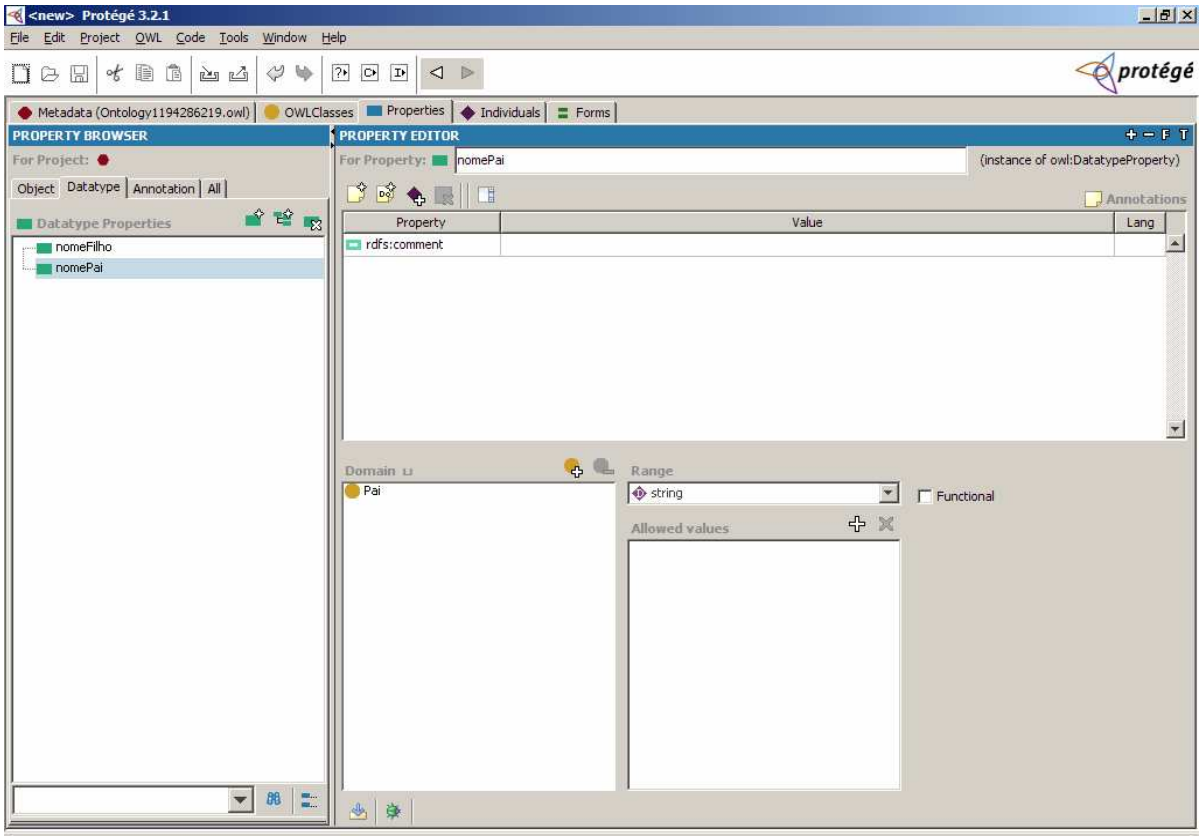

Figura 39 - Tela de definição das propriedades (*datatype properties*).

A tela de definição das propriedades de tipo de dado é praticamente idêntica á anterior. Nota-se que as diferenças ficam por conta da definição do *range* e da retirada de algumas características, que são exclusivas para as propriedades de objeto.

Neste caso, o *range* é um tipo de dado XML Schema ou um literal RDF. Essa definição é evidenciada na figura 39, que apresenta a definição da propriedade nomePai, que apresentará o nome do Pai quando a instância for criada. Pode-se notar que o *range* deverá assumir qualquer valor *string*, pois no campo *Allowed values* o valor é nulo.

Depois de definidas as propriedades e as classes, é necessário que sejam definidas as restrições sobre elas. Essas restrições são feitas na aba *OWLClasses*, dentro da guia *Class Editor,* apresentada na figura 37. Nesta guia existe uma *grid* chamada de *Asserted Conditions*. Nesta guia devem ser colocadas as restrições. Passada este processo, o último passo é definir as instâncias da ontologia, como mostra a figura 40.

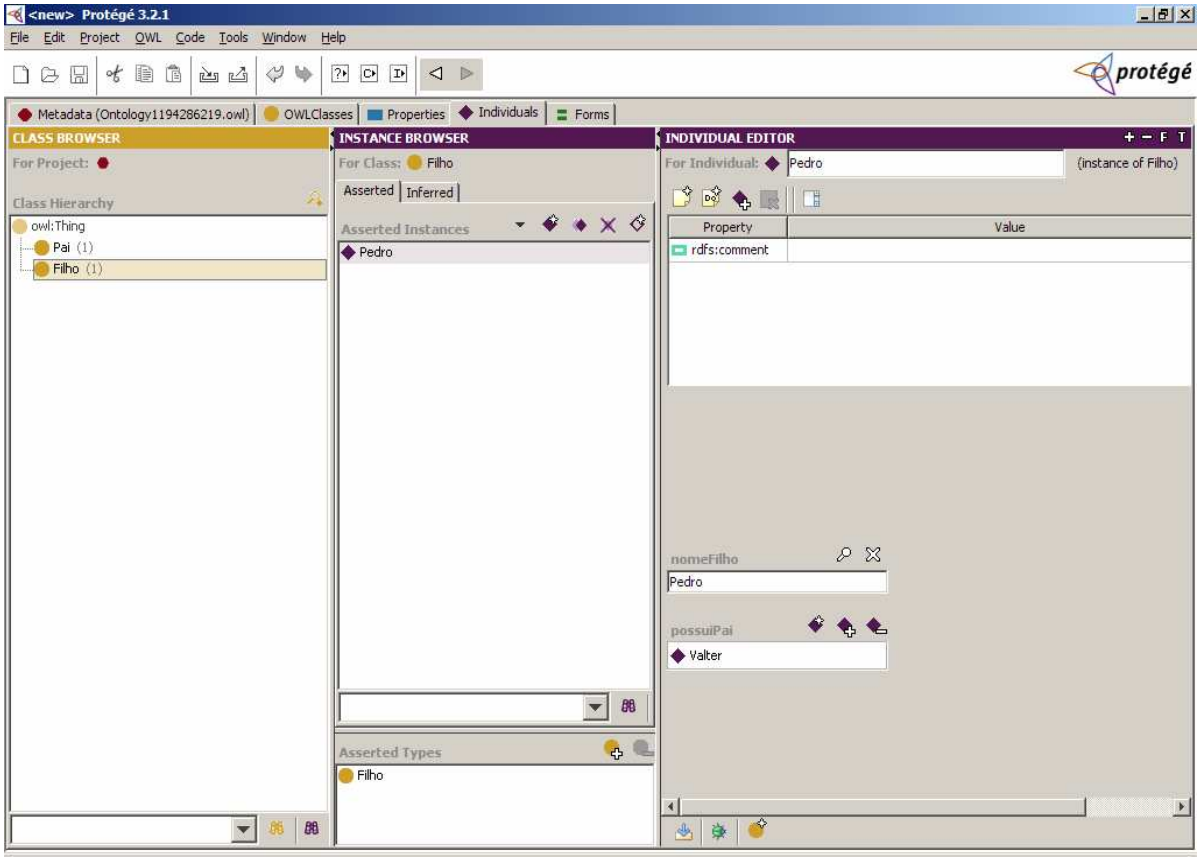

Figura 40 - Tela de criação de instâncias.

As instâncias do documento devem ser definidas na aba *Individuals*. Nesta aba, aparecem a hierarquia de classes definida (*Class Explorer*), as instâncias mostradas separadamente por classe (*Instance Browser*) e o editor de instâncias (*Individual Editor*), como mostra a figura 40.

Nesta figura, foram criadas duas instâncias: “Pedro”, a partir da classe Filho e “Valter”, a partir da classe Pai. As duas classes possuem propriedades para serem instanciadas. A classe Pai possui a propriedade nomePai e a classe Filho possui as propriedades nomeFilho e possuiPai, que aponta para a instância “Valter”. Neste caso, pode-se afirmar que Pedro é filho de Valter pois eles se encontram relacionados pela propriedade possuiPai.

A aparência e a disposição dos itens da guia *Individual Editor* pode ser modificada. Esta modificação é realizada na aba *Forms.* Nesta aba, os campos podem ser arrastados para melhor apresentação.

2.5 RacerPro

Como uma das principais características das ontologias implementadas utilizando a linguagem OWL-DL é a possibilidade de se utilizar raciocinadores (*reasoners*) para testar a hierarquia de classes e a consistências das propriedades e instâncias. Um raciocinador testa a taxonomia verificando o relacionamento entre classes e subclasses e realiza teste verificando a consistência das propriedades e das instâncias, verificando se ela atenda as necessidades da classe especificada (HORRIDGE *et al*, 2004).

O raciocinador mais conhecido atualmente é o *RacerPro*[1], que se encontra na versão 1.9.2 *Beta.* A versão mais recente completa do *RacerPro* é a 1.9.0, que será apresentada nesta seção. A sigla *RacerPro* é um acrônimo para Renamed ABox and Concept Expression Reasoner Professional (RACER, 2007). Outros exemplos de raciocinadores podem ser citados como o *FaCT* (*Fast Classification of Terminologies)* [2], *FaCT*++ [3], *Pellet* [4], entre outros. Todos estes raciocinadores possuem suporte para a linguagem OWL-DL (STANFORD).

Os seguintes serviços são fornecidos para documentos OWL e dados RDF (RACER, 2007):

- Checar a consistência da ontologia OWL e a descrição dos dados;
- Encontrar relacionamentos implícitos nas subclasses declarados na ontologia;
- Encontrar sinônimos entre recursos (nomes de classes e instâncias);
- Recuperar recursos importados da web via TCP/IP;
- Desenvolvimento de respostas às perguntas para tarefas de recuperação de informação. A isso, adiciona que o *RacerPro* se adapta aos recursos computacionais.

Em relação a adaptação aos recursos computacionais, deve-se incrementar que essa adaptação é feita através da escolha das tarefas que serão realizadas. O *RacerPro* escalona as tarefas de modo que sejam executadas primeiramente as que consomem mais recursos computacionais (RACER, 2007).

Ao iniciar o uso do *RacerPro*, é apresentada uma tela contendo informações da ferramenta conforme apresentada na figura 41 abaixo. As informações

---

1 http://www.sts.tu-harburg.de/~r.f.moeller/racer/
2 http://www.cs.man.ac.uk/~horrocks/FaCT/
3 http://owl.man.ac.uk/factplusplus/
4 http://pellet.owldl.com/

contidas nesta tela são referentes ao programa como versão, desenvolvedor, tecnologias suportadas, entre outros.

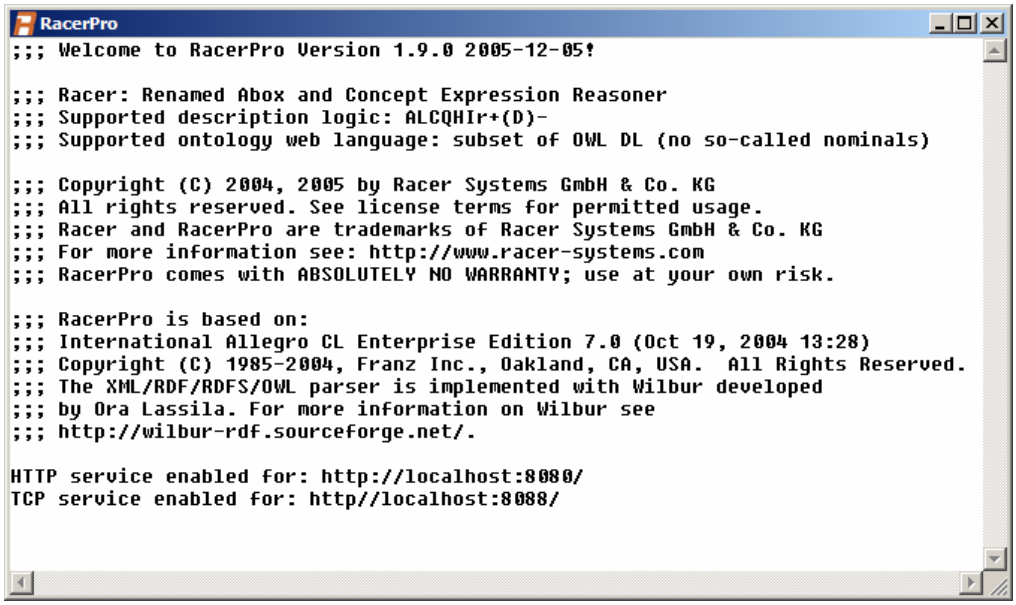

Figura 41 - Tela inicial do *RacerPro 1.9.0.*

O *Racer* é implementado a linguagem *Common Lisp* e está disponível como um programa multi-plataforma. As licenças não são específicas e os clientes podem se conectar a um usuário através do protocolo TCP/IP baseado em *sockets*. Lira (2006) complementa que o *RacerPro* é um servidor para descrição lógica ou serviços de inferência utilizando OWL.

Para realizar os testes de consistência da ontologia, é necessário que as ferramentas *Protégé* e *RacerPro* estejam sendo executadas simultaneamente. Os testes devem ser chamados através do *Protégé*, através do menu *OWL.*

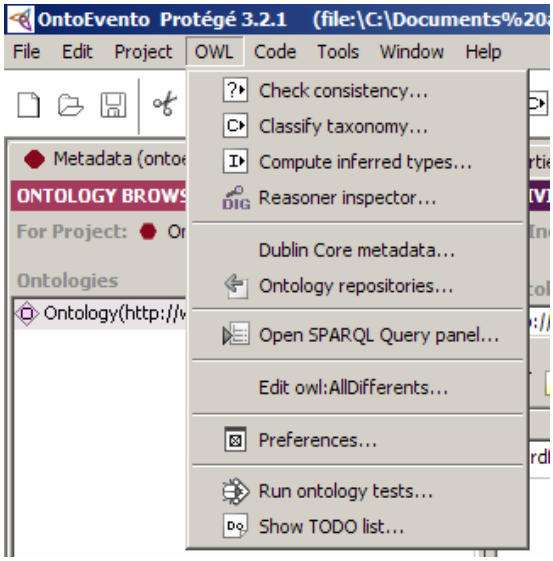

Figura 42 - Menu para escolha dos testes de consistência.

A figura 42 mostra os testes que são realizados que utilizando para validação da ontologia. O *RacerPro* é utilizando quando são realizados os testes *Check consistency*, *Classify Taxonomy* e *Compute inferred types.*

Caso se tente realizar estes testes sem o uso do *RacerPro*, ocorrerá um erro, pois não há nenhuma máquina de inferência em execução. Nota-se mais um tipo de teste denominado *Run ontology tests*. Este teste é realizado pela ferramenta *Protégé*.

Cada teste citado anteriormente verifica um item da ontologia. o teste *Check Consistency* verifica a consistência das propriedades da ontologia. A hierarquia de classes da ontologia é testada através do teste de classificação da taxonomia (*Classify taxonomy*). O terceiro tipo de teste realizado pelo *RacerPro* é *Compute inferred types*, que testa os tipos inferidos na ontologia. por último, existe a opção *Run ontology tests...*, que realiza testes sobre as instâncias da ontologia.

3 ONTOLOGIAS

O termo ontologia é oriundo da filosofia grega e, nesta área, trata do ser enquanto ser, a descrição da existência (ARAÚJO, 2003). No campo da computação, o termo vem sendo empregado desde o início da década de 90 na área de Inteligência Artificial e significa representar o conhecimento em áreas de engenharia do conhecimento e processamento de linguagem natural (BONIFACIO, 2002). Para simplificar e sintetizar esse conceito de ontologia, Gruber (1993) define ontologia como uma representação formal, explícita e compartilhada de algum contexto.

Analisando os termos desta definição, conclui-se que a ontologia precisa ser formal porque pode ser entendida por máquinas; explícita pois seus conceitos, relacionamentos e propriedades estão bem definidos, explícitos; compartilhada pois o conteúdo pode ser desenvolvido e aceito por um grupo de pessoas; e contexto porque representa alguma área do conhecimento real, um domínio específico.

O termo ontologia ganhou destaque na área da computação juntamente com o surgimento do projeto da Web Semântica, que é uma extensão da web atual. Segundo o seu criador, Tim Berners-Lee (2001), essa extensão da web possibilitará a manipulação dos dados semanticamente, independente do formato em que se encontram. Essa manipulação semântica será possível com o uso de ontologias, pois as mesmas são usadas para estruturar hierarquicamente o conteúdo de algum domínio, possibilitando aos agentes computacionais a realização de buscas semânticas sobre os dados.

Sobre essa definição de ontologia, pode-se dizer que as ontologias permitem às pessoas e às máquinas entenderem o conteúdo de um determinado domínio que está sendo analisado. Desta forma, Marietto (2002) cita alguns benefícios do uso de ontologias:

- Permite que o desenvolvedor compreenda o domínio que está modelando;
- Possibilita o compartilhamento e o reuso do conhecimento e troca de informações;
- Suporta a interoperabilidade de sistemas operacionais;
- Auxilia no processo de manutenção, documentação e verificação de um sistema computacional, comparando o modelo conceitual com o computacional;

Para o desenvolvimento de ontologias não existe um padrão definido para ser seguido. Existem algumas metodologias para o seu desenvolvimento que estão sendo testadas e discutidas atualmente. Como exemplo de metodologias pode citar as propostas de Guizzardi (2000) e Noy e McGuinness (2001).

Como a construção de uma ontologia é uma tarefa complexa e requer alguns critérios para que sejam bem desenvolvidas, se torna necessária a compreensão de alguns termos básicos da ontologia.

3.1 Componentes

De acordo com Corcho e Perez (2002), uma ontologia é formada por cinco componentes:

- **Conceitos**: podem ser definidos como classes e são organizados em hierarquias, sendo aplicados sobre eles relações de herança. Esses conceitos isolados não expressam muito significado. Para que os conceitos mostrem o seu real valor é necessário que ele se relacione com outros através de relações.
- **Relações**: são os relacionamentos que os conceitos possuem com seus atributos. As relações podem ser classificadas de duas maneiras distintas: relacionamento hierárquico (é-um) e não-hierárquico (realizado-por, sediado-por, feito-por,etc).
- **Funções**: é um tipo especial de relacionamento. Por exemplo, Exponencial(x), MediaFinal(p1,p2).
- **Axiomas**: são usados para modelar sentenças que são sempre verdadeiras. Como são sempre verdadeiros, os axiomas são utilizados para impor restrições, verificar correções e deduzir novas informações.
- **Instâncias**: representam elementos de um domínio associados a um conceito específico.

Sobre esses componentes, é possível estabelecer relacionamentos entre os conceitos de uma ontologia. Com isso, os motores de busca possuem acesso ao significado dos termos podendo realizar um processo de recuperação da informação de maneira mais eficiente e com tempo de resposta menor.

3.2 Critérios

De acordo com Araújo (2003), existem alguns critérios que devem ser observados e cumpridos durante o desenvolvimento de ontologias. Estes critérios visam atingir os benefícios proporcionados por ontologias:

- **Clareza**: deve comunicar o significado pretendido na definição dos termos. Essas definições devem ser objetivas e independentes do contexto social. Devem ainda ser declaradas em axiomas lógicos, quando possível, com documentação em linguagem natural.
- **Coerência**: as inferências devem ser coerentes com as definições axiomáticas. Quando uma sentença é passível de ser inferida a partir de axiomas da ontologia contradiz uma definição, então a ontologia é considerada inconsistente.
- **Extensibilidade**: deve proporcionar a criação de novos termos, sem que haja novas definições. Esses termos são criados a partir do vocabulário existente da ontologia.
- **Compromissos de codificação mínimos**: deve ser aplicada a nível de conhecimento, independente da tecnologia para representação de conhecimento.
- **Compromissos ontológicos mínimos**: os compromissos ontológicos de uma ontologia devem ser suficientes para suportar o compartilhamento do conhecimento. As partes envolvidas em uma ontologia devem ficar livres para se especializar e instanciar a ontologia. Por isso, uma ontologia deve fazer poucas imposições a respeito do domínio modelado.

3.3 Tipos de ontologia

As ontologias são divididas em quatro grupos segundo Guarino (1998), como pode ser observado posteriormente na figura 43.

- **Ontologias de nível superior ou genéricas**

São compartilhadas por um grande grupo e não dependem de um problema ou domínio particular. Definem apenas termos muito genéricos como espaço, tempo, matéria, evento, entre outros.

- **Ontologias de domínio**

São desenvolvidas para expressar conceitos de um domínio particular, descrevendo

o seu vocabulário relacionando a um domínio genérico, como medicina, informática, indústria farmacêutica, etc.

- **Ontologias de tarefas**

São ontologias que expressam conceitos para a resolução de tarefas independente do domínio que ocorram. Descrevem o vocabulário relacionado a uma tarefa genérica, como um diagnose ou vendas.

- **Ontologias de aplicação**

São ontologias que expressam conceitos dependentes de domínio e de tarefas especificas. Esses conceitos correspondem à atividade realizadas por entidades do domínio, quando há realização de certa atividade.

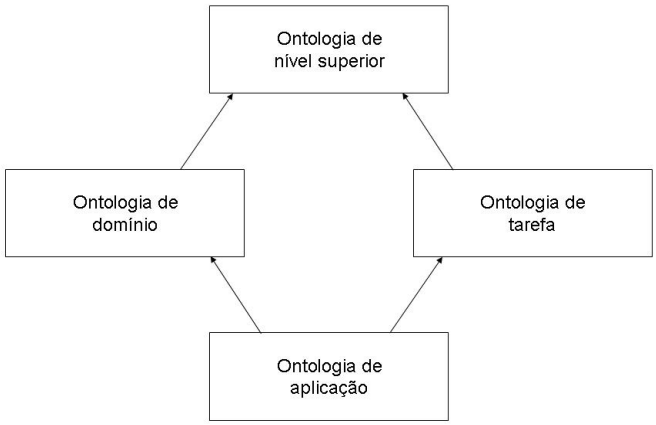

Figura 43 - Tipos de ontologias definidas por Guarino (1998).

A figura 43 mostra os quatro tipos de ontologia e sua “cardinalidade”. Segundo Guarino (1998), ontologias de domínio e de tarefa descrevem conceitos que devem ser especializações dos termos de uma ontologia de nível superior e também que os termos de uma ontologia de aplicação devem ser especializações dos termos das ontologias de domínio e tarefas.

3.4 Metodologia

Para o desenvolvimento de ontologias, não existe uma metodologia definida como padrão. Existem algumas propostas muito difundidas e aceitas atualmente para o desenvolvimento. As propostas por Noy e McGuinness (2001**),** Guizzardi (2000) e Uschold e Gruninger (1996) são as mais citadas na literatura atualmente.

Estas metodologias são semelhantes, apresentando uma seqüência coerente para o desenvolvimento. Estas metodologias seguem a linha do desenvolvimento de software no Modelo Cascata definido por Pressman (2002). Segundo este autor, para desenvolver um software é necessário que sejam estabelecidas etapas seqüenciais. Para avançar à próxima etapa é necessário que a anterior esteja finalizada.

Embora o resultado esperado do processo de desenvolvimento de ontologias seja o mesmo para cada metodologia, cada uma possui suas particularidades. Para o desenvolvimento deste trabalho foi utilizada a metodologia proposta por Noy e McGuinness (2001), futuramente intitulada de “Método 101”. Essa metodologia é composta basicamente por sete etapas que serão apresentadas:

1. Identificação do domínio e do escopo;
2. Verificação de ontologias existentes;
3. Enumeração de termos importantes do domínio;
4. Definição de classes e sua hierarquia (taxonomia);
5. Definição das propriedades;
6. Definição dos valores das propriedades;
7. Criação de instâncias.

3.4.1 Identificação do domínio e escopo

O primeiro passo para o desenvolvimento de ontologias é a definição da finalidade e do escopo da mesma. Nesta etapa, devem ser analisados os requisitos para a construção e responder as chamadas questões de competência. Essas questões englobam a motivação para o desenvolvimento, quem serão os usuários, quem irá manter a ontologia, entre outros (NOY, MCGUINNESS, 2000).

Essa etapa pode ser comparada a análise de requisitos da Engenharia de Software Orientada a Objetos, quando são criados os casos de usos, que representam os atores (usuários) do sistema e como os mesmos interagem com o sistema (GUIZZARDI, 2000).

3.4.2 Verificação de ontologias existentes

Antes de se iniciar o processo de implementação da ontologia, é necessário pesquisar se existem ontologias já prontas para o domínio escolhido. Essa

pesquisa é importante pois caso haja algum modelo pronto, não há a necessidade de se começar a desenvolver um novo projeto desde o início. Pode-se aplicar técnicas de refinamento ao modelo existente para melhor representação do conhecimento.

Nesta busca sobre ontologias, podem ser também encontrados modelos prontos que serão úteis em algumas partes do projeto em desenvolvimento. Esses modelos podem ser importados para o projeto, proporcionando ao desenvolvedor um tempo de implementação menor.

Essa importação pode aumentar o grau de confiança sobre o trabalho, pois existem modelos disponíveis que são reconhecidos pela comunidade.

3.4.3 Enumeração de termos importantes do domínio

Depois de definidos o domínio e o escopo e de se verificar que não existem modelos ontológicos desenvolvidos começa o processo de implementação da ontologia. Nesta etapa, Noy e McGuinness (2001) recomendam que se faça uma lista com os termos que serão utilizados. A partir desta lista com os termos importantes do domínio, começa a se ter uma visão mais específica do domínio.

Essa lista de termos é importante porque a partir dela o desenvolvedor começa a elaborar as classes, os relacionamentos e as propriedades de cada classe. Com isso, a lista elaborada servirá como base para as próximas etapas.

3.4.4 Definição de classes e sua hierarquia

Este item aborda o quarto processo do método proposto por Noy e McGuinness (2001). Após escolher o domínio, verificar a existência de modelos prontos ou que possam ser reutilizados e definir os principais termos, é necessário que se estruture esse conhecimento em forma de classes.

Para estruturação do conhecimento, estas classes serão definidas como uma hierarquia, onde podem apresentar subclasses para especificar alguns conceitos. Esta hierarquia também pode ser chamada de taxonomia. Os nomes e as funções de cada classe são definidos de acordo com os termos analisados na etapa anterior.

Segundo Uschold e Gruninger (1996), existem três métodos para o desenvolvimento da hierarquia de classes, sendo os dois primeiros distintos e o terceiro uma união dos dois primeiros:

- **top-down**: processo de desenvolvimento que é iniciado do conceito mais genérico. A partir dele são definidas as outras classes do sistema através de especializações.
- **bottom-up**: processo inverso ao anterior, pois o desenvolvimento começa das classes mais específicas e a partir dela são criadas as classes mais genéricas.
- **Combination**: processo que engloba os dois anteriores. Primeiramente são definidos os conceitos mais relevantes e com isso são feitas generalizações/especializações, para melhor definição da hierarquia.

Nenhum destes métodos é considerado melhor que os demais para desenvolvimento de ontologias. A escolha da melhor técnica é determinada através da visão que o desenvolvedor tem do projeto, ou seja, se o desenvolvedor possui uma visão sistemática *top-down* do domínio, é mais fácil ele utilizar a primeira proposta.

Entretanto, recomenda-se usar a *combination* para o desenvolvimento, porque podem surgir outras formas de representar e relacionar os conceitos durante o processo. Outra vantagem deste método é que o mesmo possibilita ao desenvolvedor descrever melhor o domínio da ontologia.

3.4.5 Definição das propriedades

Segundo Booch, Rumbaugh e Jacobson (2005), ao modelar um sistema percebe-se a necessidade de fazer as classes se comunicarem, de modo que elas colaborem umas com as outras. Com isso, fica claro que na modelagem é imprescindível identificar os itens que formam o vocabulário, mas também é importante modelar como estes itens se relacionam entre si.

Esta definição sobre a modelagem de sistemas orientados a objetos se encaixa perfeitamente no processo de estabelecimento das propriedades de ontologias. Em uma modelagem ontológica também é necessário fazer com que as classes se comuniquem, para que elas possam fornecer informações para as demais.

Depois de definido as classes e a sua hierarquia, é necessário estabelecer as propriedades que farão parte de cada classe. Este processo de definição das propriedades é necessário, pois ao criar uma classe ela se encontra vazia e não possui relacionamentos com outras classes.

Nessa situação não é possível responder as questões de competência elaboradas no primeiro passo da metodologia. Essas propriedades atuarão tanto no

processo de instanciação das classes da ontologia como no relacionamento entre elas (NOY; MCGUINNESS, 2001). Esse conceito de relacionamento entre as classes recorre a idéia de um banco de dados relacional (HEUSER, 2004).

Segundo Heuser (2004), em um sistema de banco de dados existem tabelas que são preenchidas com informações. Essas informações podem ser acessadas por outras tabelas através da criação de índices. Essa indexação de tabelas é uma técnica muito utilizada pois diminui consideravelmente o tempo de recuperação da informação.

Em uma ontologia, os conceitos podem ser interligados para que se possa extrair a semântica das informações, de modo que o computador consiga inferir sobre esses dados. Esses relacionamentos, em OWL, são criados utilizando propriedades de objeto.

Como exemplo de propriedade de objeto, considere a afirmação “Paulo é formado em Moda”. Para uma pessoa, essa afirmação é obvia. Para um agente computacional, não passa de uma string. Se essa afirmação for dividida em partes, poderá fazer mais sentido para um computador. Pode-se dividir esta frase em duas classes: Pessoa e Curso. Ainda assim, não possui muito sentido para um agente computacional. É necessário então que se relacionem os conceitos. Como exemplo para este relacionamento, pode-se criar o conceito ehFormadoEm. Com isso, poderia se afirmar que “Pessoa” ehFormadoEm “Curso”. a partir deste axioma, é possível extrair informações relacionadas de forma semântica.

Além das propriedades de objeto, devem ser definidas nesta etapa também as propriedades que devem ser preenchidas para realizar a instância das classes. Essas propriedades (*datatype properties*) são utilizadas para prover informações que não se encontram em outras classes. Esse tipo de propriedade foi criado para preenchimento de informações relevantes do contexto que não precisam ser criados a partir de uma classe específica a eles. Essas propriedades são utilizadas também para apresentar informações sobre as instâncias

Fazendo uma analogia novamente entre Banco de dados e ontologias, essas propriedades seriam os campos que devem ser preenchidos em uma tabela como nome, endereço, telefone, entre outros (HEUSER, 2004).

.Para a criação das propriedades, também deve ser usada à tabela de termos importantes do domínio gerada durante a terceira etapa desta metodologia. Como as classes foram definidas com alguns termos, entende-se que os termos não utilizados darão nome às propriedades. Cada propriedade criada deve fazer parte de

alguma classe, assim como toda classe deve possuir propriedades de modo que não fique vazia e sem relacionamentos.

3.4.6 Definição dos valores das propriedades

Esta etapa visa estabelecer as restrições da ontologia. Estas restrições englobam, por exemplo, cardinalidade, tipo de valor e domínio da propriedade (NOY; MCGUINNESS, 2001). Sobre estas restrições poderão ser estabelecidas quais instâncias farão parte de cada classe. Essa afirmação será possível pois, para uma instância pertencer a uma classe, ela deve atender a todos os requisitos estabelecidos pela classe (SOUTO; WARPECHOWSKI; OLIVEIRA, 2006).

Um cuidado a parte deve ser tomado na definição das propriedades. Durante este processo serão definidos os axiomas sobre os quais o raciocinador, que testa a consistência da ontologia, irá inferir. Caso algum axioma seja definido de forma semanticamente errada, o raciocinador não acusará nenhum erro de consistência (HORRIDGE *et al*, 2004). Tal erro não afetará na consistência da ontologia, pois não apresenta falha de definição ou de restrições. Um exemplo de erro que pode se ocorrer na definição dos axiomas é definir que a propriedade “possuiPalestrante” relacione a classe “Palestra” a uma instância da classe “Cidades”. Isto não acarretará em erro de consistência, mas o raciocinador poderá inferir que uma instância de cidade poderá ser palestrante de determinado evento.

A primeira etapa para definir as propriedades é estabelecer quais classes serão relacionadas ou quais classes possuirão campos de instância dentro da ontologia. Essa definição é realizada estabelecendo o domínio (*domain*) e a escala (*range*) de cada propriedade.

O domínio da propriedade são os elementos em que a propriedade citada pode ser aplicada e devem ser sempre referenciados a alguma classe. Em contrapartida, deve-se estabelecer a escala de cada propriedade, que é a imagem da propriedade. Essa imagem são os valores que a ela pode assumir. Estes valores podem estar associados a uma classe ou a um tipo de dado, dependendo da propriedade escolhida.

Feita a definição do domínio e da imagem da propriedade, a próxima etapa é definir as suas características. Em OWL, uma propriedade pode assumir diversas características.

3.4.7 Criação de instâncias

A etapa de criação de instâncias é a última parte do "*Método 101"* de Noy e McGuinness (2001). como se sabe, uma instância é o nome dado a indivíduos definidos a partir de uma classe. Esses indivíduos são definidos através da instanciação das propriedades da classe (CARVALHO; LIMA, 2005).

De acordo com Noy e McGuinness (2001), esta etapa é realizada seguindo uma seqüência. A primeira tarefa a ser realizada é selecionar a classe que será instanciada. Logo após, é necessário que se crie um exemplo e que sejam preenchidos os valores das propriedades necessárias.

4 DESENVOLVIMENTO

O objetivo deste trabalho é definir um modelo ontológico para o domínio de eventos científicos. Esta ontologia representara o conhecimento através de regras impostas a classes e propriedades modeladas a partir da análise do domínio citado. Para melhor identificar o modelo ontológico desenvolvido, foi denominado o nome de OntoEvento, que será utilizado durante o trabalho.

O desenvolvimento deste trabalho é motivado pela promoção da Web Semântica, que é uma extensão da web atual e uma área de pesquisa que se encontra em ascensão no momento. Essa promoção é realizada através de ontologias. Outro item que deve ser levado em conta é que não existem modelos ontológicos direcionados a este domínio e a necessidade de se estruturar o domínio.

Para o desenvolvimento da OntoEvento utilizou-se a metodologia proposta por Noy e McGuinness (2001) pois esta apresenta uma seqüência lógica de etapas, sendo possível explicar, separadamente, cada uma delas. Este processo de desenvolvimento será explanado durante as próximas seções deste trabalho.

A OntoEvento foi desenvolvida utilizando a ferramenta *Protégé* versão 3.2.1 para a representação do conhecimento e estabelecimento de regras, classes, propriedades e instâncias. Esta ferramenta foi desenvolvida pela Universidade de Stanford e provê suporte à linguagem OWL-DL, que por sua vez é sustentada pelas tecnologias XML e RDF.

O uso da linguagem OWL é justificado por ser definido como um padrão para o desenvolvimento de ontologias segundo o W3C (MCGUINNESS; SMITH; WELTY, 2004). Para desenvolver o trabalho, foi usada a sub-linguagem OWL-DL (Lógica Descritiva) pois esta linguagem suporta construções complexas para representar o conhecimento, funcionalidade não encontrada na versão *Lite.* A sub-linguagem OWL *Full* poderia ser utilizada para a construção da ontologia, pois possibilita realizar as mesmas tarefas da linguagem OWL DL. No entanto, a segunda sub-linguagem foi adotada devido a possibilidade de se utilizar raciocinadores para testar a taxonomia e a consistência dos documentos.

Para os testes de consistência da ontologia foi empregado o *reasoner* (raciocinador) *RacerPro 1.9.0*. Este raciocinador age sobre a ontologia usando um classificador *SHIQ*. Essa escolha pode ser fundamentada sobre o fato de este ser um raciocinador que atende as necessidades exigidas no trabalho.

A ferramenta *Protégé* é utilizada porque se encontra em constante

desenvolvimento, se aprimorando cada vez mais para oferecer suporte à Web Semântica. Sobre isso, ainda pode-se afirmar que a interface gráfica amigável contribui para a escolha da ferramenta. Além de possuir um robusto banco de dados de bases de conhecimentos.

A seguir, serão apresentadas as etapas do processo de desenvolvimento da OntoEvento com base na metodologia proposta por Noy e McGuinness (2001).

4.1 Identificação do domínio e do escopo

Segundo Guizzardi (2000), a identificação do domínio e do escopo é primordial para o desenvolvimento da ontologia, pois a partir dela será desenvolvido todo o trabalho proposto. A partir desta análise é possível especificar os requisitos e as questões de competência da ontologia. Para se definir o domínio foram acessadas inúmeras páginas de eventos. A primeira observação feita é sobre os tipos de eventos encontrados. Na OntoEvento foram categorizados cinco tipos de eventos: Congresso, Simpósio, Semana, Workshop e Conferência.

Cada tipo de evento possui características próprias. Como exemplo disso, podemos apresentar os eventos classificados como Congresso, que podem ser subdivididos em duas categorias: Técnico e Científico.

Outro item que foi constatado na análise do domínio são os itens essenciais para a divulgação do evento. Como exemplos destes itens, podem-se citar datas de realização e inscrição, local de realização, programação, entre outros. Estas afirmações podem ser conferidas nas figuras abaixo.

Figura 44 - Página do evento Latinoware 2007 (www.latinoware.org).

Figura 45 - Página do evento 10º CBCENF (http://www.cbcenf.com.br/10cbcenf/default.asp).

Como se pode notar nas figuras 44 e 45, é possível obter informações a respeito de eventos pela internet. A primeira figura mostra a página IV Conferência Latino-Americana de Software Livre chamada de LatinoWare, enquanto a segunda mostra a página do 10º Congresso Brasileiro dos Conselhos de Enfermagem. É evidente que a estrutura da página dos eventos se encontra diferente em razão de serem desenvolvidos por autores diferentes. Mas essa apresentação não altera o conteúdo da página.

A respeito dos itens essenciais, de inicio é possível conferir nas duas páginas informações como o nome e edição do evento, a cidade e o estado de realização, notícias ou apresentação do evento, entre outros. Ao acessar os links da página do evento, é possível abstrair informações importantes como público-alvo, atrações, eventos similares, entre outros.

Alguns destes itens não são encontrados em todos os eventos, com isso eles são definidos como opcionais na criação de instâncias. Através disso, pode-se satisfazer um objetivo da OntoEvento, abrangendo o maior número possível de eventos científicos, independente das suas características.

Além das páginas mostradas nas 44 e 45, foram visitadas as páginas de diversos eventos como o 13º Congresso da Abraves (Medicina Veterinária), XXII Simpósio Brasileiro de Banco de Dados/XXI Simpósio Brasileiro de Engenharia de Software (SBBD/SBES), II Seminário de Informática e Tecnologia (SiTE), I Simpósio de Engenharia Rural (SER), entre outros.

Com base na análise realizada, foram definidos os principais conceitos relacionados a eventos científicos. Estes conceitos tornam possível a elaboração das questões de competência da ontologia, como sugere a metodologia utilizada.

Ao final do processo de definição da OntoEvento será possível responder, de forma mais precisa, as questões de competência abaixo:

- Quais os eventos realizados no estado do Paraíba?
- Em quais eventos acontecem palestras sobre Web Semântica? E quem será o palestrante?
- Onde serão realizados eventos na área de enfermagem em 2007?
- Onde se encontram eventos que possuem público alvo professores e estudantes de Agronomia?
- Em quais eventos tal pessoa terá participação como palestrante ou como membro da comissão organizadora?

- Quais os eventos realizados no ano de 2007?

4.2 Verificação de ontologias existentes

Após a definição do domínio e do escopo da OntoEvento, a próxima etapa da metodologia de Noy e McGuinness (2001) é a verificação de modelos ontológicos existentes para a ontologia criada.

Como verificado na análise do domínio e escopo da ontologia, não foram encontradas ontologias desenvolvidas para estruturação do domínio de eventos científicos.

Ainda nesta pesquisa para desenvolvimento da ontologia, foram encontrados alguns modelos implementados que podem ser importados para a OntoEvento devido a necessidade. Essa importação se faz útil devido ao fato de uma ontologia representar um conhecimento compartilhado. Assim, o processo de desenvolvimento pode ser simplificado importando modelos prontos e reutilizando o conteúdo. Para a OntoEvento, existem alguns modelos que satisfazem essa afirmação:

- Dublin Core (http://dublincore.org/): metadados semânticos utilizados para modelagem de documentos digitais.
- countries.owl (http://www.bpiresearch.com/BPMO/2004/03/03/cdl/Countries): ontologia para países, mostrando a estrutura territorial e o código de cada um de acordo com a ISO 3166.
- OntoQualis (SOUTO; WARPECHOWSKI; OLIVEIRA, 2006): modelo ontológico criado para qualificar eventos científicos na área da Ciência da Computação.

A principio, estes modelos citados não serão importados para a OntoEvento. Esta opção pode ser justificada devido ao fato destes modelos estarem passando por uma análise de suas funcionalidades e a contribuição para a OntoEvento.

4.3 Enumeração de termos importantes do domínio

Após o cumprimento das duas primeiras etapas, começa a implementação da ontologia. Como o próprio nome da etapa sugere, agora serão enumerados os itens importantes do domínio. Sobre eventos científicos, podemos citar os seguintes termos importantes do domínio:

Apresentador	Área	Associação	Cidade
Comissão Organizadora	Conferência	Congresso	Cursos
Datas	Edição	Empregado	Empresa
Estado	Evento Científico	Formação Acadêmica	Grande Área
Histórico	Inscrições	Instituição	Local de realização
Ministrado	Objetivo	Órgão Governo	País
Palestrante	Patrocinador	Pessoa	Programação
Realização	Sala	Sede	Semana
Sigla	Simpósio	Valor	Workshop

Quadro 1 - Lista de termos importantes do domínio.

O quadro 1 definida anteriormente apresenta os principais termos definidos para a realização das próximas etapas da construção da OntoEvento. Este quadro sofreu alterações durante o desenvolvimento do trabalho devido à descoberta de novas funcionalidades que foram adicionadas posteriormente.

Ainda assim, este quadro deve ser seguido como um guia para representação dos conceitos e propriedades de ontologias. Finalizando esta etapa, pode-se considerar que está encerrada a análise do domínio, pois foram definidas as funcionalidades e os termos que farão parte do domínio.

4.4 Implementação da ontologia

Esta seção representa as etapas de implantação da ontologia que correspondem às etapas de Definição de Classes e Hierarquia, Definição das Propriedades e Definição dos Valores das Propriedades do método de Noy e McGuinness (2001). Estas etapas foram realizadas em conjunto porque não há necessidade de se realizar sequencialmente estas tarefas, pois as mesmas se encontram interligadas e são extremamente dependentes umas das outras.

Durante o processo de implementação da OntoEvento são identificados alguns conceitos que foram apresentados no capítulo anterior. O primeiro item identificado são os componentes de uma ontologia definidos por Corcho e Perez (2002). Segundo estes autores, uma ontologia é composta por cinco componentes: conceitos, relações, funções, axiomas e instâncias.

Ao iniciar o uso da ferramenta Protégé para desenvolvimento da OntoEvento, estes componentes são facilmente definidos. Os conceitos são apresentados através das classes da ontologia. As relações são as propriedades definidas para relacionar uma classe à outra (*object properties*) ou a algum tipo de dado (*datatype proerties*). Os axiomas são as restrições aplicadas sobre as propriedades, que atuam delimitando o valor que pode ser aceito pelas propriedades. E as instâncias são os indivíduos criados a partir de uma classe. As funções, que é um tipo especial de relacionamento, não apresentaram nenhum exemplo durante o trabalho.

Para nomear os componentes que compõem a OntoEvento, foi adotado o critério de criar nomes auto-descritivos, que sejam fáceis de identificar a função de cada elemento. Essa nomenclatura foi adotada seguindo o critério de clareza estabelecido para o desenvolvimento de ontologias citado por Araújo (2003). Segundo este critério, os termos de uma ontologia devem possuir nomes objetivos e independentes do contexto em que estão aplicados.

Outros critérios são utilizados para o desenvolvimento da OntoEvento. Sobre o critério de coerência da ontologia, todas as inferências respeitam os axiomas definidos. Esta coerência será testada ao final do processo de desenvolvimento, quando são realizados os testes de validação da ontologia.

Em relação à codificação, qualquer ferramenta ou linguagem de programação com suporte a linguagem OWL pode interpretar este documento. Os critérios de extensibilidade e compromissos ontológicos garantem que a OntoEvento possa receber novos itens a partir do vocabulário existente e que apenas o conhecimento essencial seja incluído, de modo que seja maximizado o reuso.

A última consideração a ser feita antes de se iniciar a descrição do processo de implementação da ontologia é em relação ao seu tipo. A OntoEvento pode ser considerada uma ontologia de alto-nível, pois descreve termos de um domínio mais genérico. Sobre este tipo de ontologia podem ser geradas outras para representação de domínios relacionados ao de eventos científicos.

Para iniciar a implementação da OntoEvento, foram definidas as classes que fariam parte do documento. A escolha dos elementos que se tornaram classes foi baseada na necessidade de se agrupar informações semelhantes de uma determinada especificação do domínio. Analisando o quadro elaborado na etapa anterior, foram definidas as classes que são mostradas na figura abaixo.

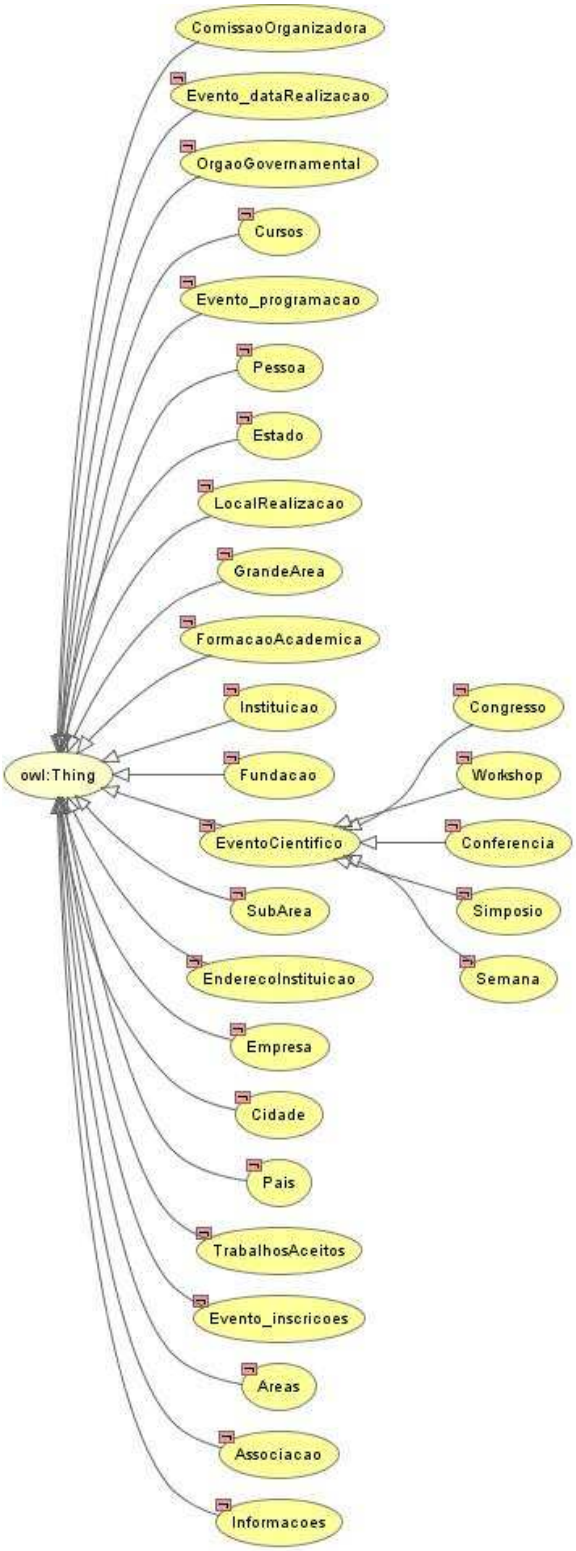

Figura 46 - Representação gráfica da hierarquia de classes da OntoEvento.

Analisando o grafo da figura 46, que contém as classes definidas para a OntoEvento, pode-se concluir que todas as classes derivam da classe *owl:Thing*. Pode-

se observar que o elemento Evento Cientifico do quadro 1 foi transformado na classe EventoCientifico.

A partir desta classe, foram criadas especificações de acordo com o tipo de evento. Essas especificações foram definidas na análise do domínio. As especificações da classe EventoCientifico são Congresso, Conferencia, Simposio, Workshop ou Semana, que pode ser caracterizado como Seminario também.

Para o desenvolvimento da hierarquia de classes utilizou-se o método *combination*. A utilização deste método foi necessária pois houve a necessidade de se realizar generalizações/especializações em determinadas classes. Este método pode ser conferido ao se definir a classe Pessoa. A princípio, para se criar alguma pessoa relacionada a um evento científico, deveria ser criada uma instância a partir do nível de participação da pessoa no evento. Este fato ocorria, por exemplo, pois existiam classes distintas de Palestrante e Instrutor.

Com isso, seriam criadas inúmeras instâncias referentes a mesma pessoa. E estas instâncias poderiam estar relacionadas ao mesmo evento, caso esta pessoa exercesse mais de uma função em um evento. Para solucionar este impasse, todas as pessoas que fazem parte de um evento foram agrupadas na classe Pessoa, e dentro da classe Pessoa existem propriedades para definir o nível de participação no evento, evitando que sejam criadas mais de uma instância para uma mesma pessoa.

Como exemplo de especialização é possível citar as classes referentes às datas do evento. Em um primeiro momento, haveria a classe DatasEvento, que seria subclasse de EventoCientifico. Com isso, não era possível dividir em etapas as datas de inscrições que variam de acordo com a classificação da pessoa ou data da realização da inscrição.

Este problema foi solucionado ao definir a classe Evento_inscricoes. Nesta classe são armazenadas a data de início e final das inscrições, bem como o tipo de inscrição e o valor da inscrição ou de programas adicionais.

Depois de definidas as classes que farão parte da ontologia, é necessário implementar essas classes em OWL através da ferramenta *Protégé*. Após essa implementação, foi gerado a hierarquia de classes definida na figura 47.

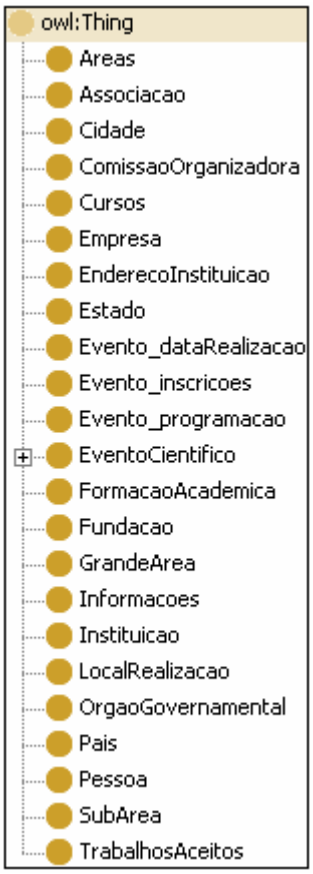

Figura 47 - Hierarquia de classes da OntoEvento.

Como pode ser observado, a classe EventoCientifico se apresenta com maior destaque. Esse destaque ocorre pois a partir desta classe serão criadas as instâncias dos eventos científicos presentes na ontologia. Como já mencionado anteriormente, foi identificado que eventos possuem características essenciais para sua realização.

A classe EventoCientifico agrupa as informações principais de um evento. Devido ao fato desta classe não trabalhar sozinha e haver a necessidade de se relacionar os conceitos para extrair a semântica do contexto, as demais classes são criadas para definir as características do evento. Essas classes trabalham alimentando a classe EventoCientifico com informações através de relacionamentos.

Como a estrutura das classes já foi implementada, a próxima tarefa é definir as propriedades de cada classe e as restrições para a formação dos axiomas, tornando possível a extração semântica das informações depois de criadas as instâncias.

A definição das propriedades segue a mesma lógica de nomenclatura adotada para as classes, seguindo uma seqüência lógica de raciocínio e obedecendo o critério de clareza, o que torna mais simples a definição dos valores das propriedades.

Para demonstrar as propriedades da OntoEvento, serão mostradas as propriedades da classe EventoCientifico.

As propriedades comuns para todos os eventos foram definidas na classe mais genérica. Deste modo, todo evento criado a partir desta ontologia deverá instanciar essas propriedades. Uma exceção é feita em relação às propriedades funcionais ou não necessárias, que são opcionais.

Figura 48 - Definição das propriedades da classe EventoCientifico.

Como pode ser conferido na figura 48, uma instância da classe EventoCientifico deve satisfazer as condições descritas pelas propriedades. Como são muitas as propriedades desta classe, serão apresentadas apenas alguns exemplos de definição de propriedades e restrições. A primeira propriedade a ser explicada é chamada de realizadoPor.

Segundo esta propriedade, um evento cientifico pode ser realizado por um conjunto formado pela união das classes do tipo Associacao, Empresa, Instituicao, Fundacao ou OrgaoGovernamental. Esta união foi estabelecida para que não seja necessário criar uma propriedade específica para cada classe. Assim, a propriedade é definida conforme a figura 49.

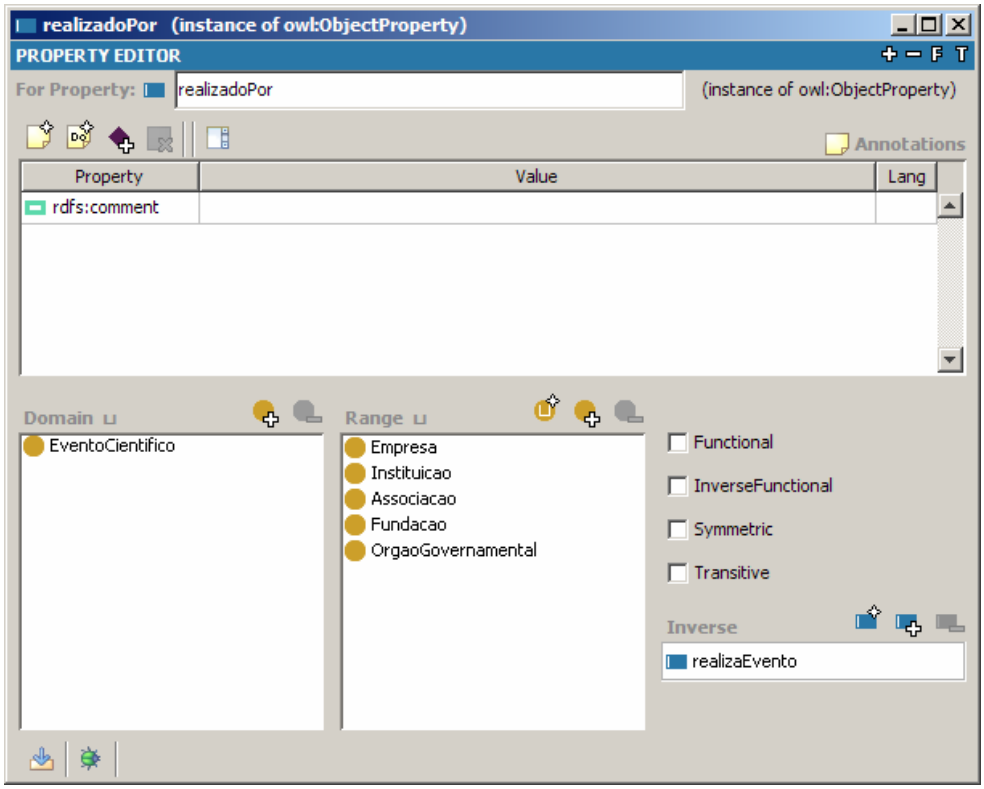

Figura 49 - Definição da propriedade realizadoPor.

O domínio (*rdfs:domain*) da propriedade é a classe EventoCientifico, pois a referida propriedade se trata de um evento. A união entre as classes citada anteriormente pode ser notada na declaração do valor (*rdfs:range*). Essa construção foi elaborada com base na análise do domínio. Segundo esta análise, um evento pode ser realizado por diferentes tipos de entidades, dependendo do caráter do evento.

Outro item desta definição é que entidades que possuem características diferentes e que forem enquadradas em classes diferentes podem organizar o mesmo evento, sem nenhum problema. Como exemplo desta afirmação, um evento do tipo Congresso pode ser realizado por diversas empresas em parceria com algum órgão do governo.

Ainda segundo a propriedade realizadoPor, nota-se a propriedade inversa realizaEvento. Esta propriedade tem um valor significativo em termos de semântica. Segundo esta propriedade, a união mencionada entre as classes é mantida, sendo que as entidades podem realizar determinado evento.

A partir destas duas propriedades, são definidos dois axiomas da OntoEvento. O primeiro axioma indica que um evento cientifico é realizado pela união de algumas entidades. O outro axioma representa que várias entidades podem realizar um evento.

A próxima propriedade a ser explicada é possuiComissaoOrganizadora. Segundo o nome da propriedade, esta propriedade estabelece um relacionamento entre EventoCientifico e ComissaoOrganizadora. Esse relacionamento ocorre devido a necessidade de se ter uma comissão organizadora para eventos científicos. Essa comissão é formada por pessoas e tem por objetivo organizar o evento.

De acordo com essa definição, é possível definir que um evento deve possuir uma comissão organizadora. E essa comissão deve ser formada por pessoas. Para representar essa afirmação na OntoEvento, serão necessárias três classes e duas propriedades.

A primeira relação é entre EventoCientifico e ComissaoOrganizadora através da propriedade possuiComissaoOrganizadora. Essa propriedade deve ser definida conforme a figura 50.

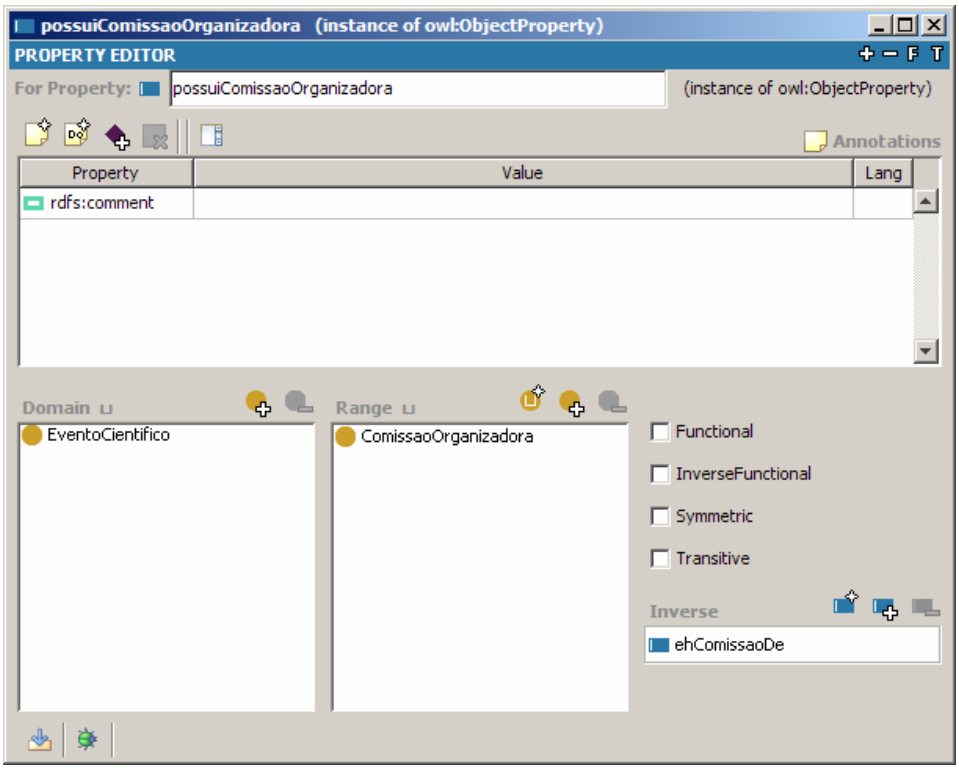

Figura 50 - Definição da propriedade possuiComissaoOrganizadora.

A propriedade possuiComissaoOrganizadora define como domínio a classe EventoCientifico e como valor a classe ComissaoOrganizadora. Para esta propriedade deve ser criada a restrição de cardinalidade para que haja exatamente uma comissão organizadora por evento. Essa restrição é definida conforme a figura abaixo.

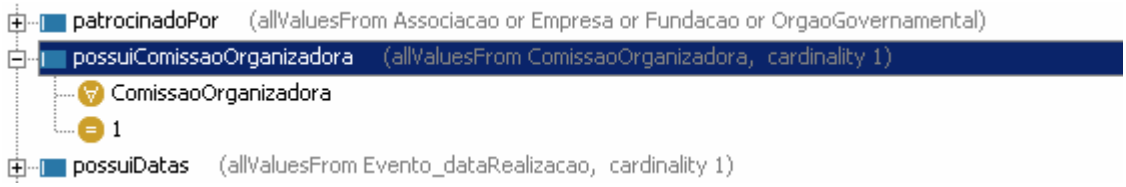

Figura 51 - Restrições da propriedade possuiComissaoOrganizadora.

De acordo com a figura 51, a propriedade em questão só poderá assumir valores que sejam uma instância de ComissaoOrganizadora. Essa restrição é imposta através do elemento *owl:allValuesFrom*. A restrição de cardinalidade fica por conta do elemento *owl:cardinality*. Na figura acima, esse elemento é referenciado com o valor 1.

Essa propriedade possui como a propriedade ehComissaoDe como propriedade inversa. Segundo essa propriedade, toda instância de ComissaoOrganizadora deve ser relacionada com a classe EventoCientifico. essa propriedade pode ser conferida na figura abaixo.

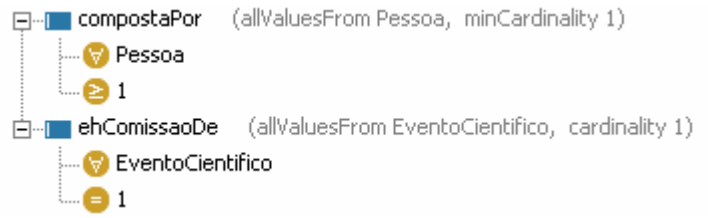

Figura 52 - Restrições da classe ComissaoOrganizadora.

Como mencionado, a classe ComissaoOrganizadora possui a propriedade ehCompostaDe, que deve assumir instâncias da classe EventoCientifico e deve se relacionar a apenas uma instância. Outra regra da formação da comissão organizadora é que a mesma deve ser composta por pessoas. Essa funcionalidade é descrita através da propriedade compostaPor.

Para limitar os valores assumidos pela propriedade é criada a restrição que indica que uma comissão deve ser formada por pessoas. A outra restrição é que haja pelo menos uma pessoa na comissão, porque não há um número mínimo de integrantes para compor uma comissão. A propriedade inversa à compostaPor é compoeComissao. Essa regra define que uma pessoa pode compor comissão e não há limite para participação. Uma instância de pessoa pode aparecer em quantas comissões forem definidas.

As demais classes e propriedades da OntoEvento podem ser conferidas nas figuras posteriores, onde são apresentados alguns diagramas de classes referentes

a OntoEvento. O primeiro diagrama foi gerado através da ferramenta Protégé e apresenta uma visão global da ontologia.

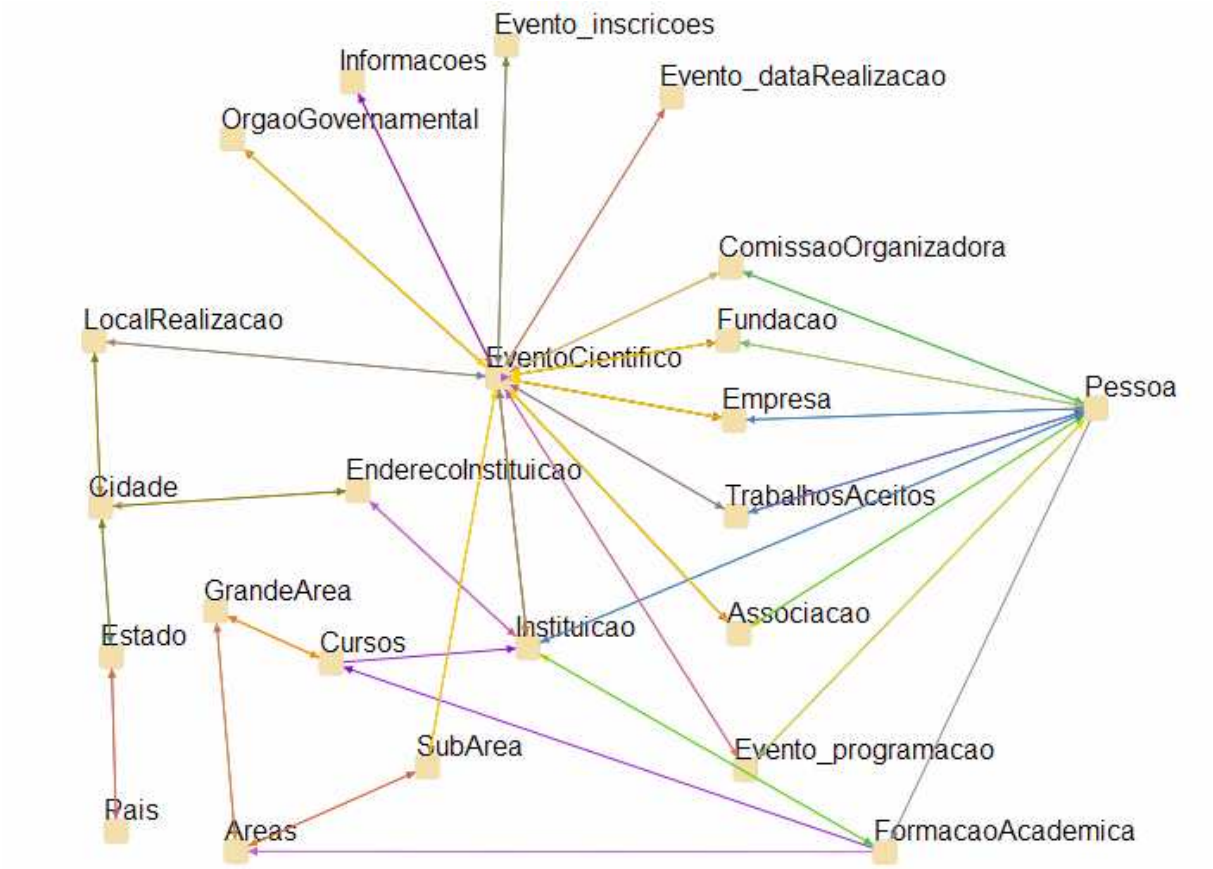

Figura 53 - Classes e relacionamentos da OntoEvento.

Pode ser notado no diagrama acima as classes e as propriedades que atuam sobre elas. Estas propriedades são utilizadas para estabelecer o relacionamento entre as classes e são representadas por arestas coloridas. Nos próximos diagramas serão mostradas as classes bem como as propriedades que atuam sobre cada uma.

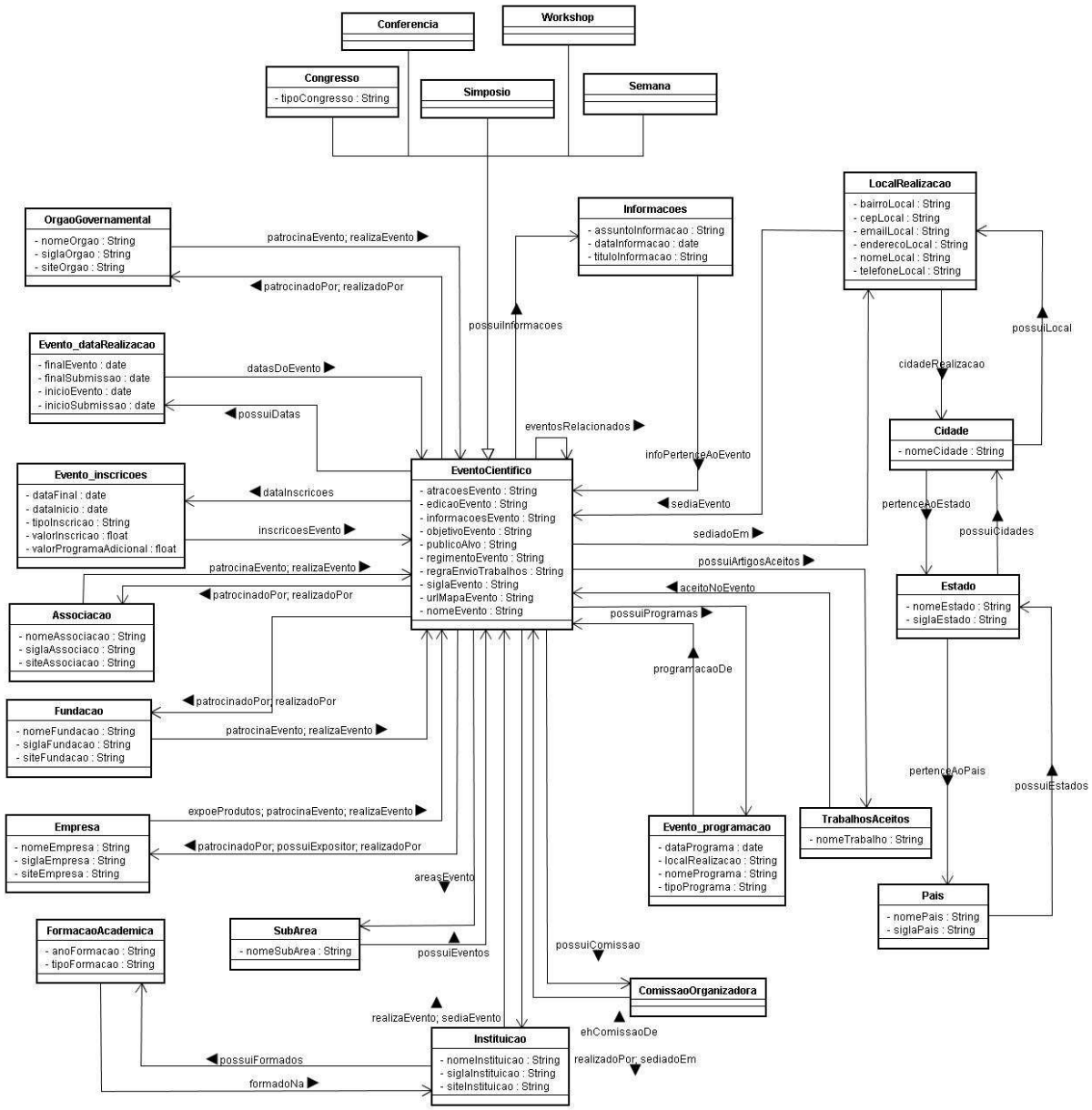

Figura 54 - Representação gráfica das classes e propriedades da OntoEvento.

O diagrama de classes apresentado pela figura acima foi desenvolvido utilizando a linguagem UML. Esta escolha é justificada pela semelhança entre as classes de uma ontologia e de um sistema desenvolvido utilizando o paradigma orientado a objetos. Com isso, as classes da ontologia foram representadas utilizando o modelo de classes UML.

As propriedades foram representadas de acordo com o seu tipo. As propriedades do tipo *object properties* associam duas classes de modo que seja definido o relacionamento existentes. E as propriedades do tipo *datatype properties* foram representadas como atributos de uma classe, pois a sua funcionalidade é semelhante a dos atributos de uma classe definida em UML.

Para melhor visualização dos conceitos, o diagrama apresentado na figura anterior foi dividido em dois sub-diagramas, que serão apresentados nas próximas figuras. a primeira figura mostra as propriedades relacionadas a classe EventoCientifico e a segunda mostra a classe Pessoa e suas propriedades. Estas classes foram escolhidas pois agrupam os principais itens representados na OntoEvento.

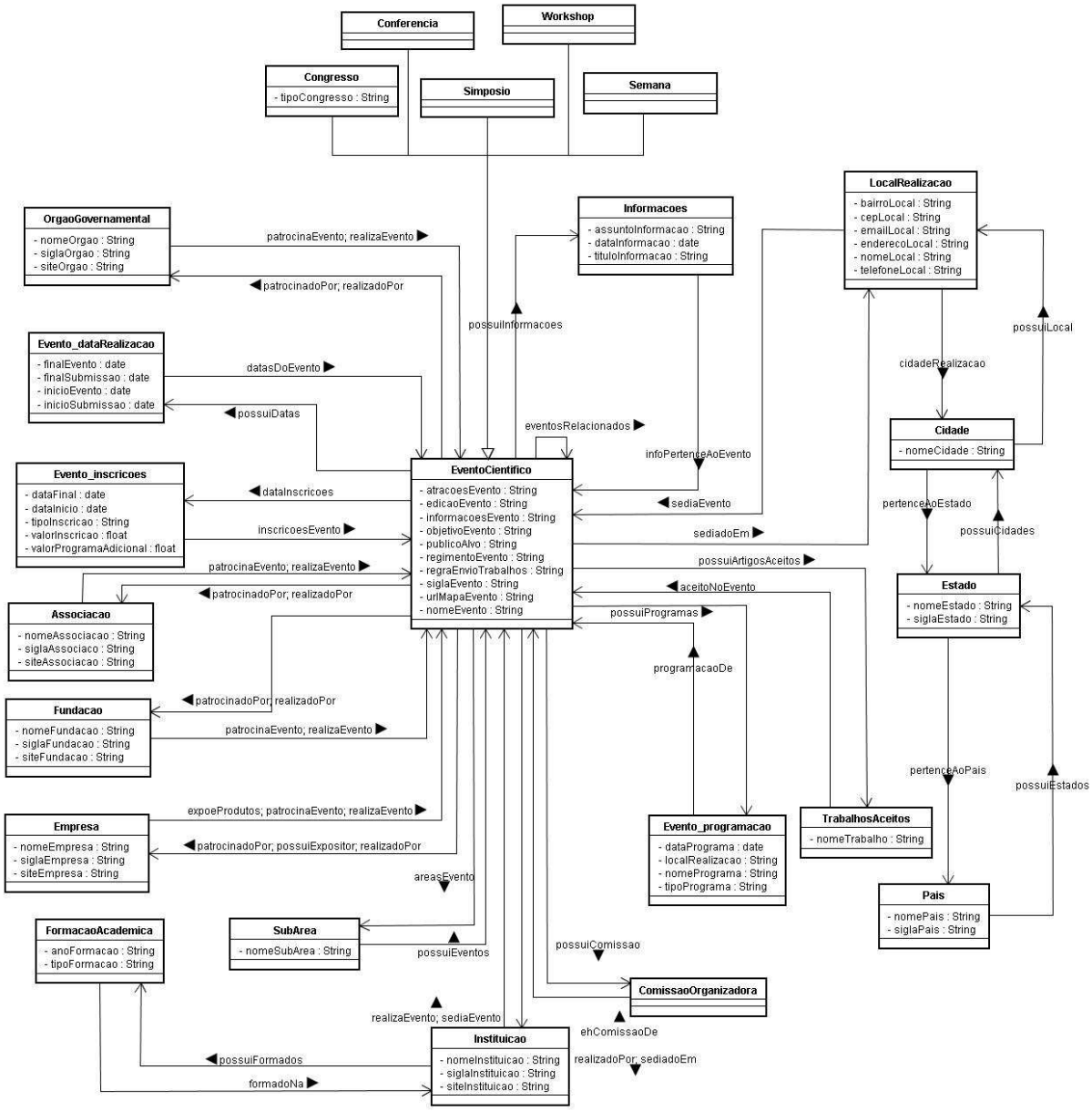

Figura 55 - Representação gráfica da classe EventoCientifico.

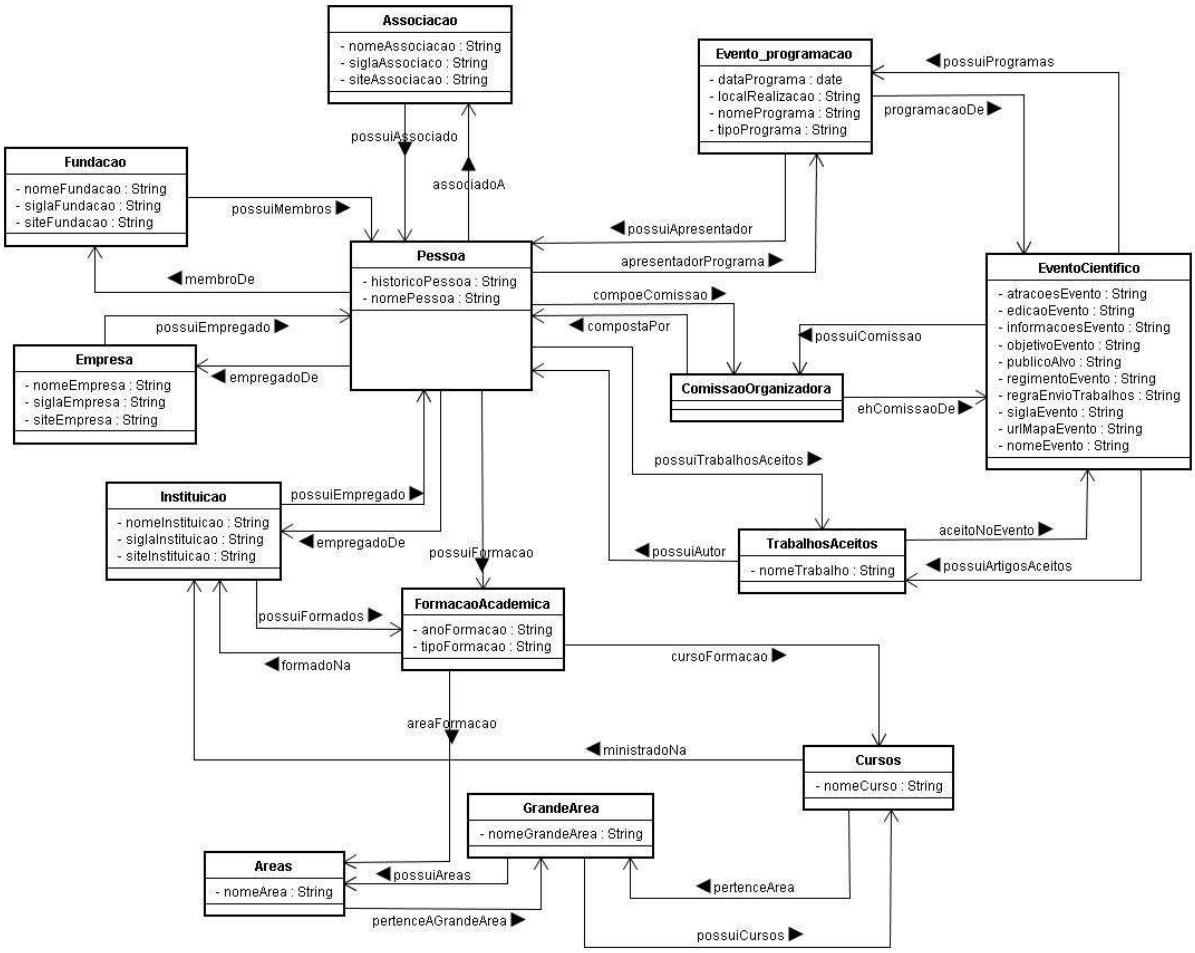

Figura 56 - Representação gráfica da classe Pessoa.

Os dois diagramas são representados utilizando o diagrama de classes da linguagem UML. Com o uso destes diagramas é possível obter uma visão mais ampla sobre a OntoEvento pois os conceitos se apresentam representados e relacionados graficamente. A lista com as propriedades da OntoEvento, bem como suas restrições de domínio e valor pode ser conferida no Apêndice A deste trabalho. A OntoEvento se encontra disponível através do endereço eletrônico http://www.uel.br/pessoal/ailton/ontoevento.owl.

4.5 Criação de instâncias

A etapa de criação de instâncias é o último passo para a definição da ontologia segundo a metodologia de Noy e McGuinness (2001). A partir destas instâncias são realizadas as consultas utilizando a semântica dos documentos. No *Protégé*, as instâncias são conhecidas como indivíduos e devem ser criadas na aba *Individuals* (HORRIDGE *et al*, 2004).

Como foi definido anteriormente, o domínio atendido pela OntoEvento é o de eventos científicos. Com isso, os valores inseridos serão necessariamente referente a algum tipo de evento. Para demonstrar a criação de instâncias na OntoEvento, serão criadas instâncias de dois eventos científicos consultados: XXI Simpósio Brasileiro de Banco de Dados (XXII SBBD) [1] e X Congresso Brasileiro de Enfermagem (X CBCENF) [2]. Este processo pode ser conferido durante o desenvolver desta seção.

O primeiro evento a ser criado é o XXII Simpósio Brasileiro de Banco de Dados. Este evento ocorreu simultaneamente ao Simpósio Brasileiro de Engenharia de Software (SBES). Estes eventos são divulgados na mesma página e são realizados no mesmo local de realização. Mas como se trata de eventos totalmente diferentes, deve ser criada uma instância para cada evento.

A instância do XXII SBBD será chamada de “XXII_SBBD”. Como o *Protégé* não aceita espaços na declaração das instâncias, utiliza-se o caractere *underline* para a separação das palavras. Para criar a instância “XXII_SBBD”, é preciso selecionar a classe Simpósio e definir seu nome, como mostra a figura abaixo. Não há uma recomendação quanto a nomenclatura das instâncias, porém, para a OntoEvento, as instâncias serão definidas em caixa alta para melhor identificação.

Para a criação da instância citada, devem ser criadas algumas instâncias que serão relacionadas a ela. Estas instâncias podem ser definidas anteriormente ou conforme apareça a necessidade de defini-las. Com isso, evita-se criar instâncias que não serão utilizadas na ontologia.

[1] http://www.sbbd-sbes2007.ufpb.br/
[2] http://www.cbcenf.com.br/10cbcenf/evento.asp?sectionID=203

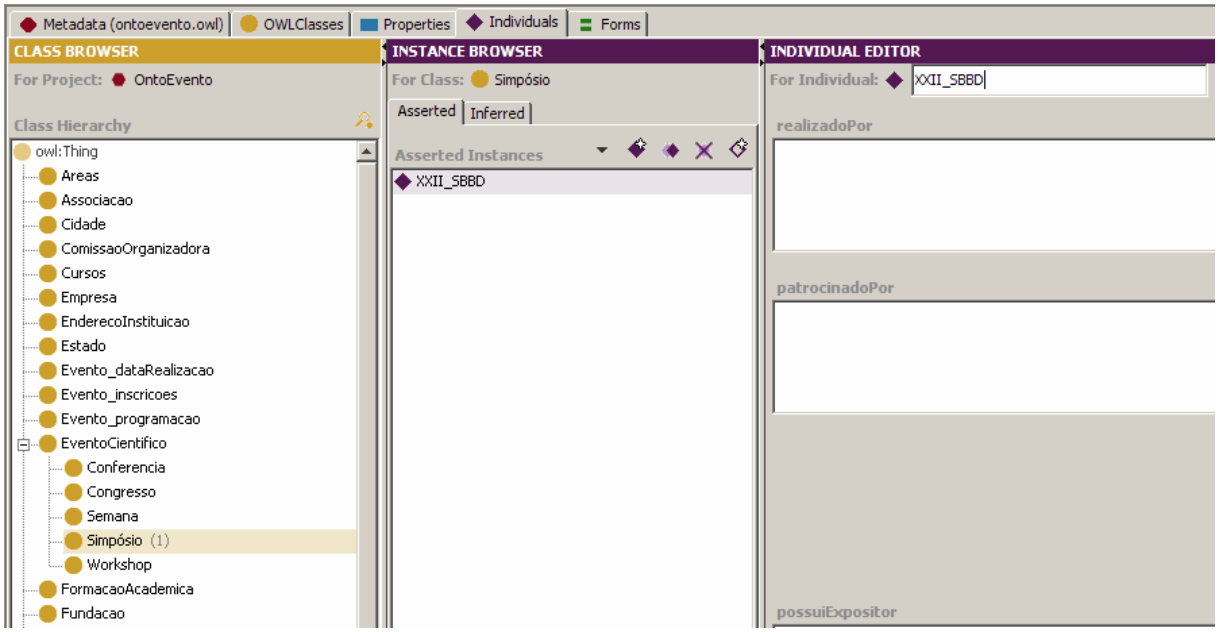

Figura 57 - Criação da instância "XXII_SBBD".

Em OWL, essa criação é definida da seguinte maneira:

```
(1)    <Simposio rdf:ID="XXII_SBBD"/>
```

Figura 58 - Criação da instância "XXII_SBBD" em OWL.

Ao clicar sobre o nome da instância criada, é aberta uma janela para edição das propriedades. O uso desta janela é recomendado pois apresenta mais espaços para visualização dos campos. Deve-se observar que o *Protégé* indica os campos que são necessários. Estes campos podem ser conferidos na figura 59.

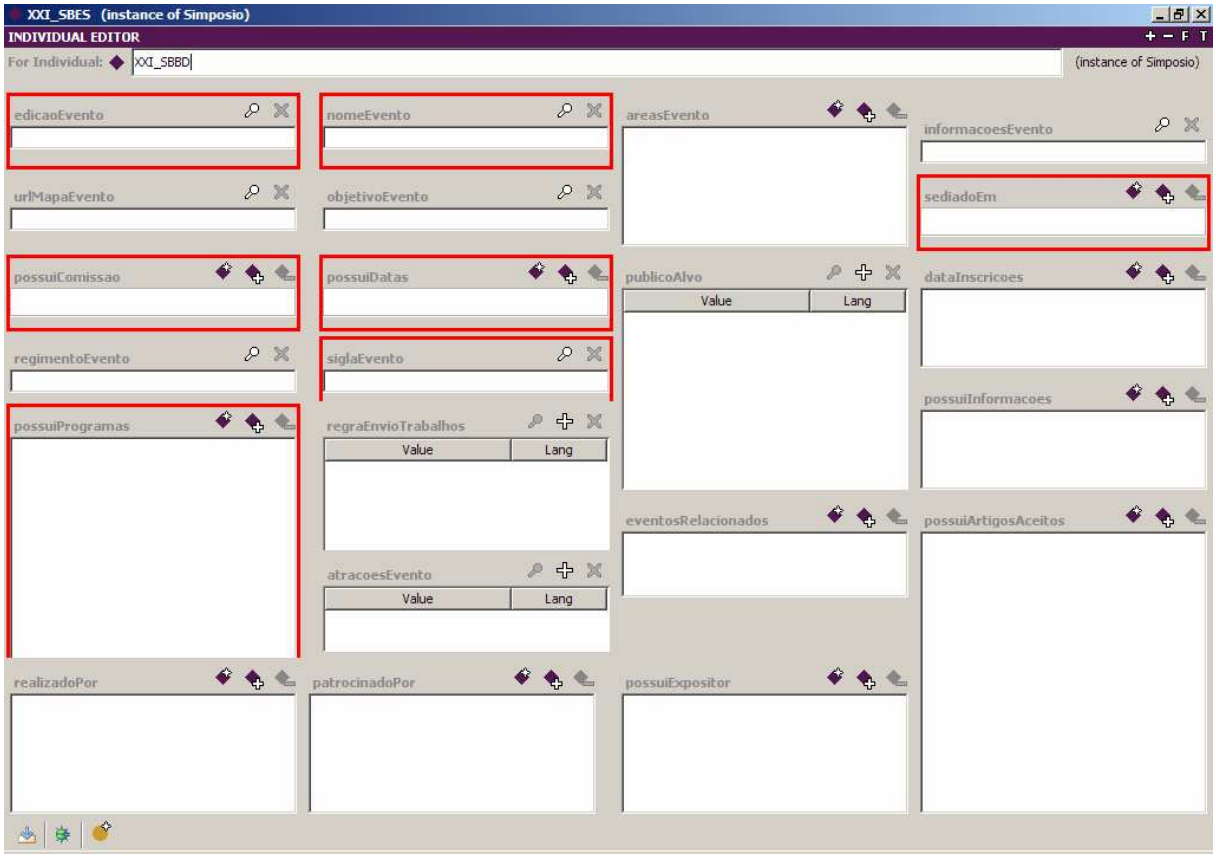

Figura 59 - Tela indicando os campos obrigatórios.

Após a criação da instância, é necessário que sejam definidas as propriedades que estão relacionadas a ela. Estas propriedades são especificadas de acordo com o conteúdo da página do evento analisado. Existem atributos que não serão preenchidos devido a ausência de informação da página. No código da ontologia, estes atributos não são criados e no trabalho estes atributos serão declarados como *null* (nulo).

Nome da propriedade	**Valor**
areasEvento	
Tipo:	*Object*
Instâncias:	BANCO_DE_DADOS COMERCIO_ELETRONICO APLICACOES_WEB entre outras.
atracoesEvento	
Tipo:	*Datatype (xsd:string)*
Valores:	**null**
dataInscricoes	
Tipo:	*Object*
Instâncias:	XXII_SBBD_INSC_FASE1_CATE1 XXII_SBBD_INSC_FASE1_CATE2 entre outras.

edicaoEvento	
Tipo:	*Datatype (xsd:string)*
Valor:	XXII
eventosRelacionados	
Tipo:	*Object*
Instâncias:	XXI_SBES II_WOMSDE III_SEAS entre outros.
informacoesEvento	
Tipo:	*Datatype (xsd:string)*
Valores:	Promovido anualmente pelas Comissões Especiais de Banco de Dados ..
nomeEvento	
Tipo:	*Datatype (xsd:string)*
Valores:	Simpósio Brasileiro de Banco de Dados
objetivoEvento	
Tipo:	*Datatype (xsd:string)*
Valores:	Promover e discutir tecnologias referentes à área do evento e áreas relacionadas.
patrocinadoPor	
Tipo:	*Object*
Instâncias:	GOOGLE, CNPQ, SEBRAE, entre outros
possuiArtigosAceitos	
Tipo:	*Object*
Instâncias:	**null**
possuiComissaoOrganizadora	
Tipo:	*Object*
Instância:	XXII_SBBD_COMISSAO
possuiDatas	
Tipo:	*Object*
Instância:	XXII_SBBD_DATA_REALIZACAO
possuiExpositor	
Tipo:	*Object*
Instâncias:	**null**
possuiInformacao	
Tipo:	*Object*
Instâncias:	XXII_SBBD_INFO1, XXII_SBBD_INFO2, entre outras.
possuiProgramas	
Tipo:	*Object*
Instâncias:	XXII_SBBD_PAL1, XXII_SBBD_PAL2, XXII_SBBD_MC1, XXII_SBBD_TUTO1, entre outros.
publicoAlvo	
Tipo:	*Datatype (xsd:string)*
Valores:	Estudantes de informática Profissionais de informática Público em geral.

realizadoPor	
Tipo:	*Object*
Instâncias:	UFPB, SBC, ACM e VLDB
regimentoEvento	
Tipo:	*Datatype (xsd:string)*
Valor:	**null**
regraEnvioTrabalhos	
Tipo:	*Datatype*
Valor:	Regras da SBC
sediadoEm	
Tipo:	*Object*
Instância:	UFPB
siglaEvento	
Tipo:	*Datatype (xsd:string)*
Valor	SBBD
urlMapaEvento	
Tipo:	*Datatype (xsd:string)*
Valor:	**null**

A inserção destes valores na ontologia é realizadas através da janela contendo as informações da instância “XXII_SBBD”. Esta janela é apresentada na figura 60.

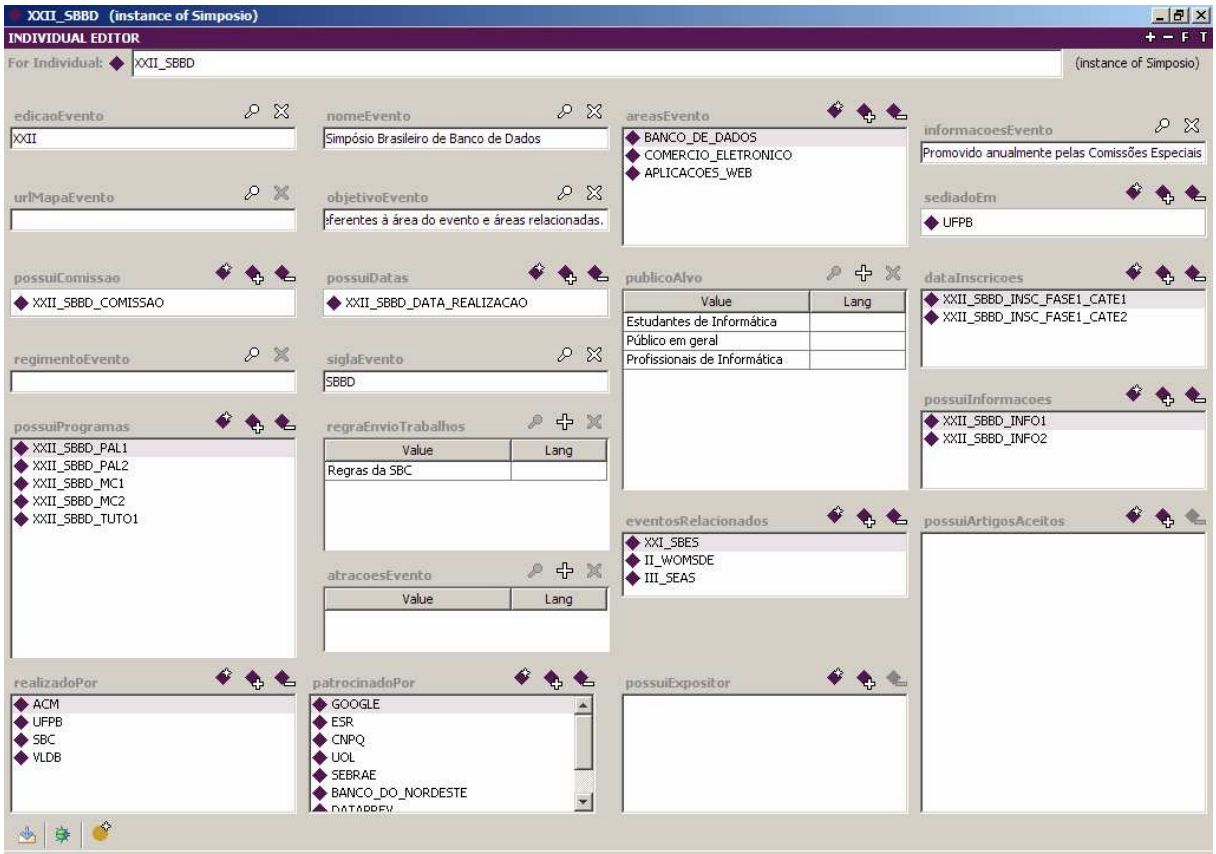

Figura 60 - Tela para inserir valores na instância “XXII_SBBD”.

Nesta tela são exibidas todas as propriedades definidas para a classe EventoCientifico. A partir desta tela são definidas todas as características do evento apresentado. Como pode ser notado, existem campos que não foram preenchidos. Estes campos fazem referência às propriedades como regimentoEvento, possuiExpositor, possuiArtigosAceitos, ente outros.

A opção por deixar estes valores em branco é justificada pelo fato de não existir informações referentes a eles na página do evento, não sendo possível extrair as informações necessárias. Uma exceção é feita sobre a propriedade possuiArtigosAceitos. Este campo não foi preenchido por se tratar de um evento recente e não constar essas informações na página, mas que normalmente são inseridas após um certo tempo de conclusão do evento.

Outro item a ser observado na nomenclatura das propriedades é em relação ao prefixo adotado. Em propriedades especificas do evento como programação, informações ou datas do evento, por exemplo, utilizou-se como prefixo o nome da instância. Essa nomenclatura ajuda na identificação dos recursos referentes ao evento.

Em relação às propriedades que podem ser referenciadas a outros eventos ,como patrocinadores, instituições ou áreas do evento, adotou-se por utilizar um nome praticamente auto-descritivo em relação a natureza da propriedade. Como exemplo, é possível citar a instância BANCO_DE_DADOS. Esta instância faz referência à sub-área Banco de Dados e contém informações para identificação da sub-área. A definição dessa propriedade é definida na figura abaixo.

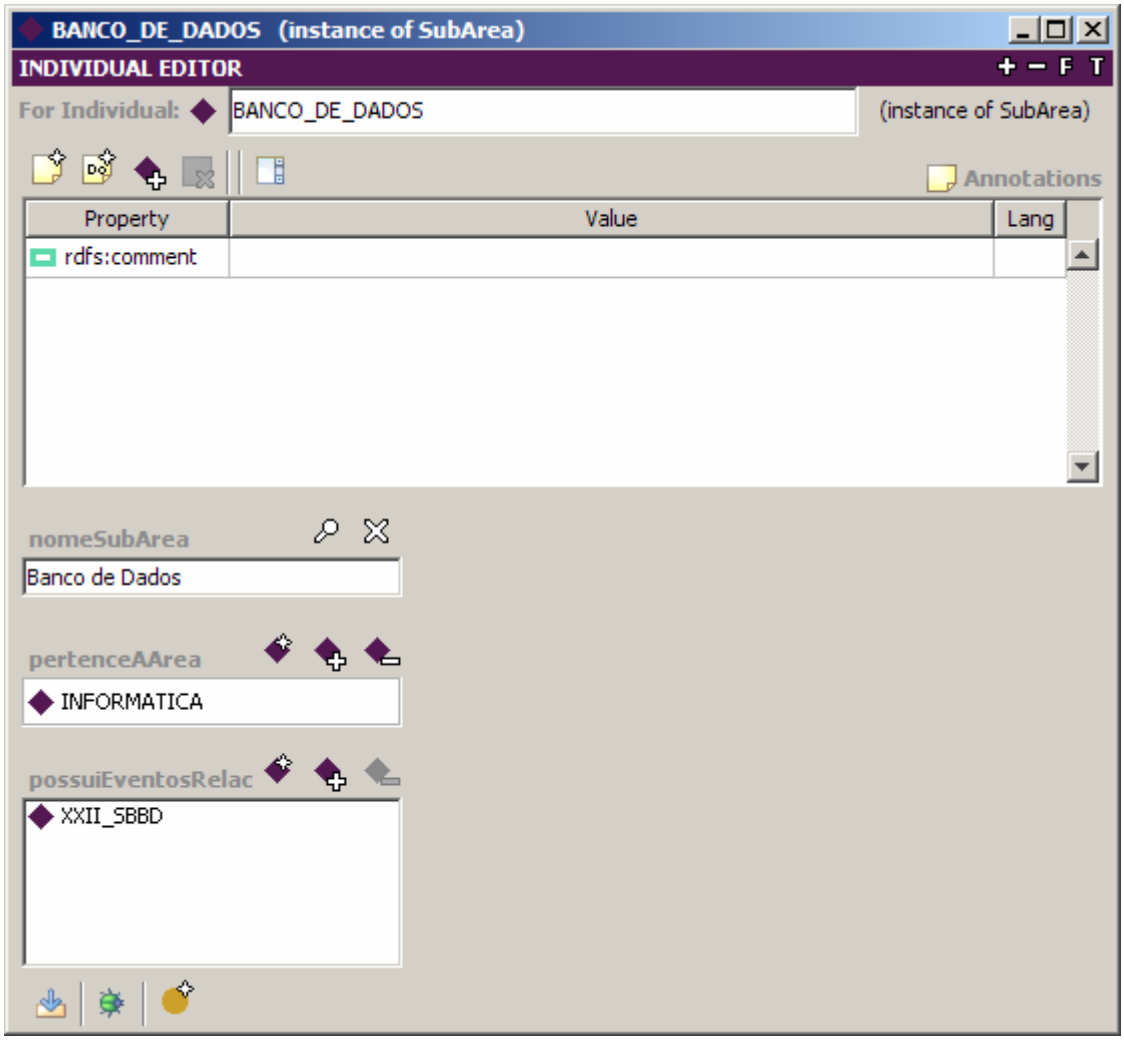

Figura 61 - Criação da instância “BANCO_DE_DADOS”.

A outra instância criada na ontologia é referente ao X CBCENF. Esta instância será nomeada como “X_CBCENF”. As propriedades relacionadas a este evento serão definidas na figura abaixo.

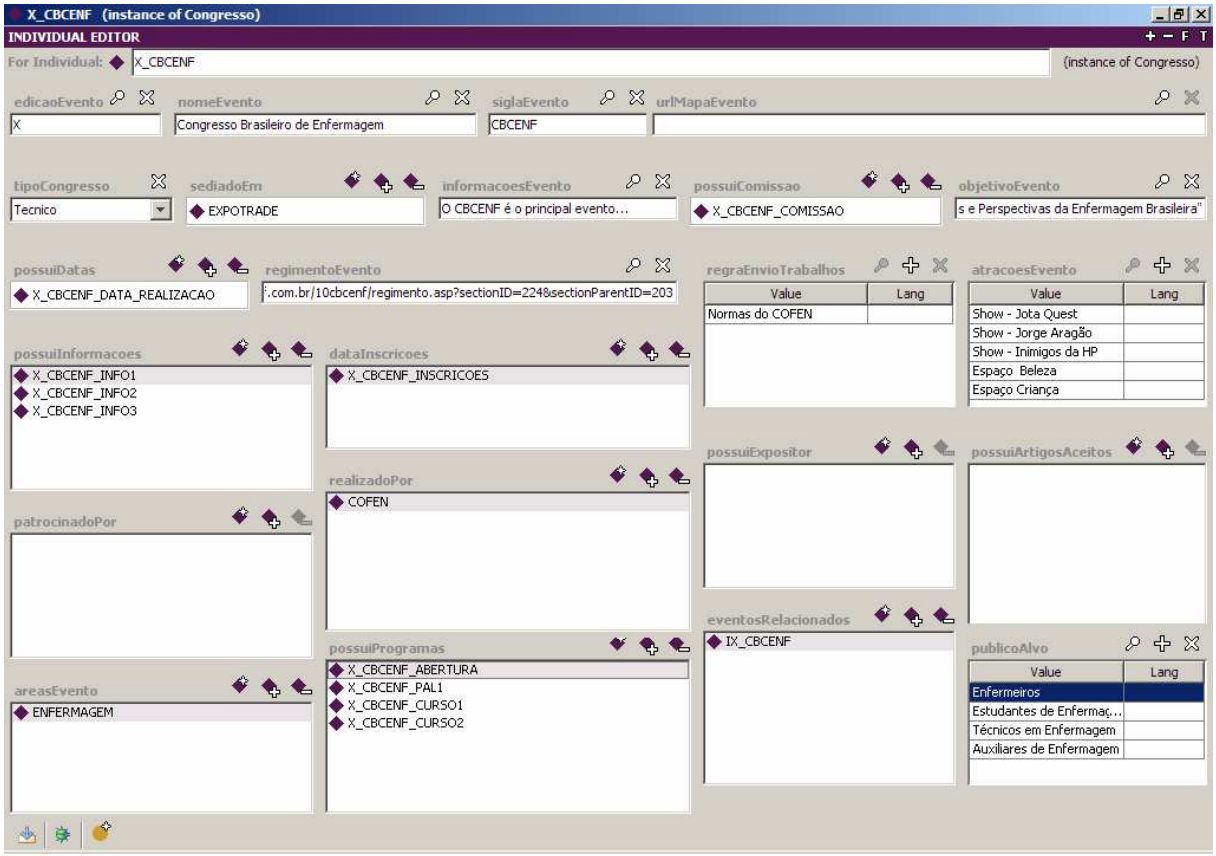

Figura 62 - Definição das propriedades da instância “X_CBCENF”.

A figura 62 mostra a criação da instância “X_CBCENF”. Esta instância é criada a partir da classe Congresso, que é sub-classe de EventoCientifico. como se percebe, as propriedades referentes a simpósios e congressos são praticamente as mesmas. O diferencial entre as classes, além do tipo de evento, é a adição da propriedade tipoCongresso, que obrigatoriamente assume valores “Técnico” ou “Cientifico”.

Com a realização desta etapa de instanciação da ontologia, é encerrado o processo de criação da ontologia segundo a metodologia de Noy e McGuinness (2001). Após o processo de criação da ontologia, é necessário realizar testes para comprovar a consistência da mesma. Caso este teste não apresente nenhuma incoerência, a ontologia é considerada consistente e pode ser manipulada.

4.6 Validando a OntoEvento

O processo de validação da ontologia pode ser realizado em duas etapas. A primeira validação é realizada após a definição da ontologia e a segunda é realizada após a criação das instâncias (LIRA, 2006). Para validar a OntoEvento é utilizado o raciocinador *RacerPro*, que deve ser utilizado juntamente com o *Protégé*.

Para a primeira validação serão realizados os testes utilizando o *RacerPro*. O primeiro teste realizado irá checar a consistência da ontologia. Este teste é realizado através do menu *OWL/Check consistency*.

Esta verificação é feita sobre as restrições definidas para classes e propriedades para checar a semântica da ontologia. Uma classe denominada inconsistente não pode apresentar instâncias e são representadas na hierarquia através de um círculo vermelho. Durante o teste de consistência da OntoEvento não foram apresentados erros de consistência. Esse teste pode ser conferido na figura 63.

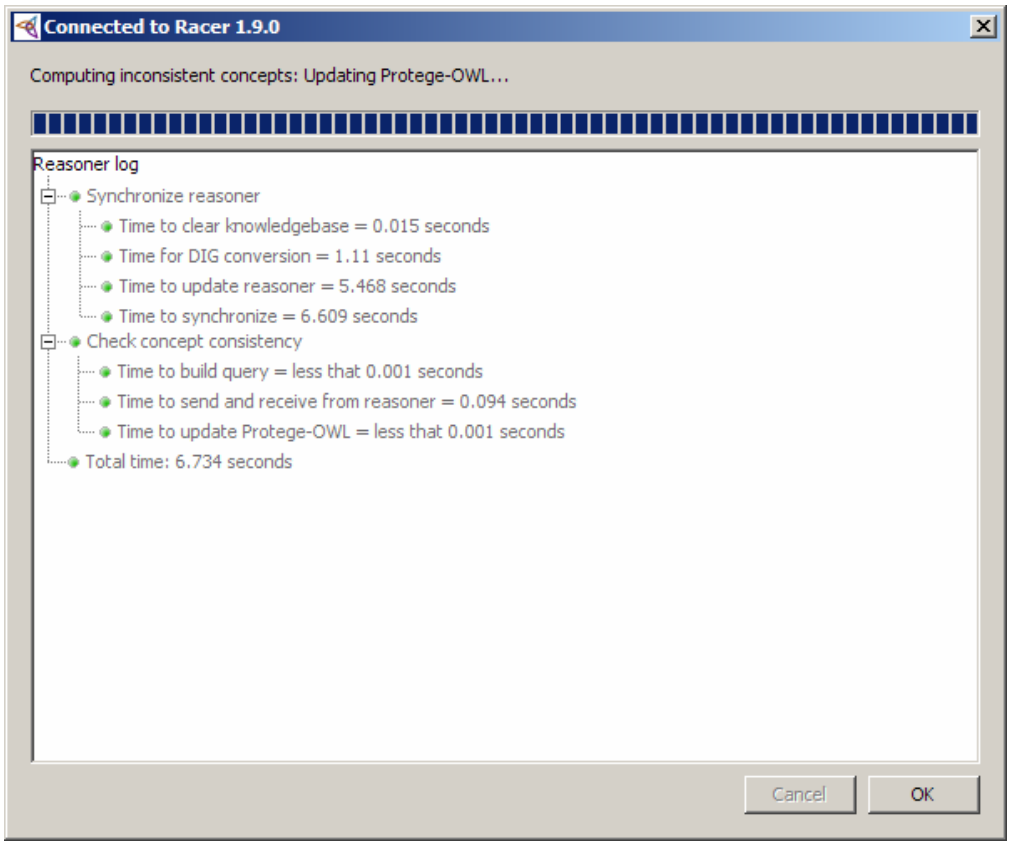

Figura 63 - Tela exibindo o resultado da verificação de consistência da ontologia.

Ainda na primeira validação, a próxima tarefa é testar a hierarquia de classes através do teste denominado *Classify taxonomy.* Este verifica se uma classe é ou não sub-classe de outra. Este teste possibilita ao *RacerPro* inferir sobre a hierarquia

de classes da ontologia. o resultado deste teste pode ser conferido na figura 64.

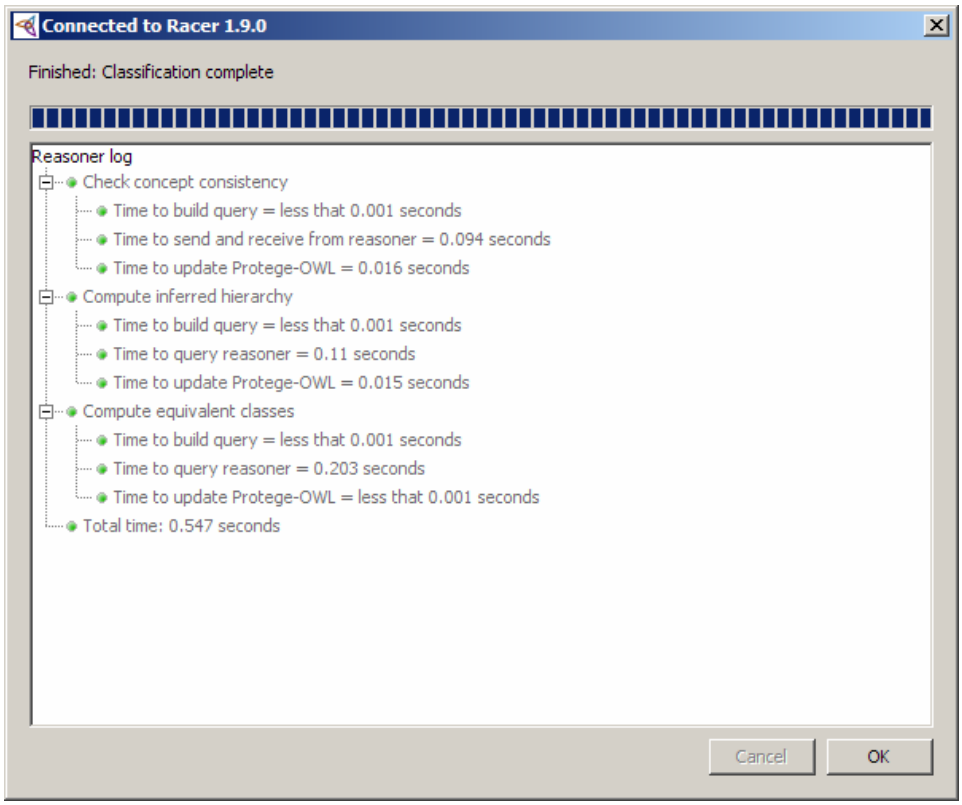

Figura 64 - Tela exibindo o resultado da classificação da hierarquia de classes.

A hierarquia de classes definida manualmente no *Protégé* é denominada hierarquia definida (*asserted hierarchy*). Com a utilização do *RacerPro*, é gerada a hierarquia inferida (*inferred hierarchy*), que é automaticamente validada pelo raciocinador.

O ultimo teste a ser realizado pelo *RacerPro* é definido como *Compute inferred types.* Este verifica se os tipos inferidos estão de acordo com os tipos definidos na ontologia. A figura 65 mostra o resultado deste processo.

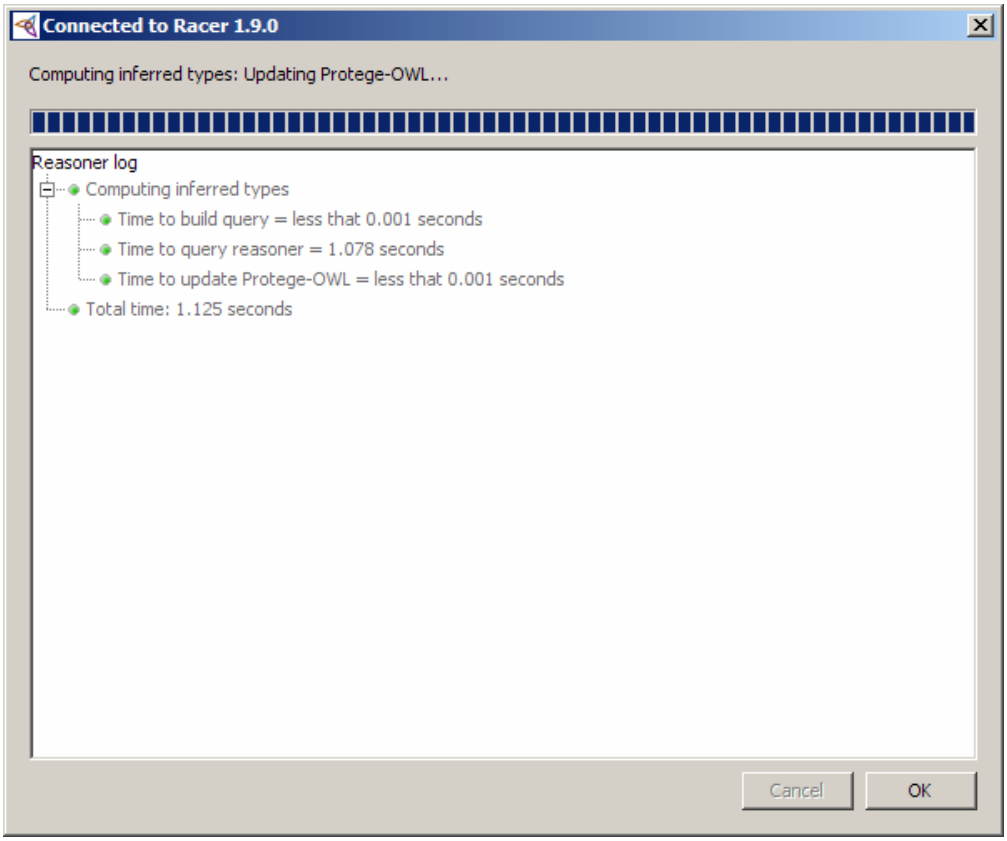

Figura 65 - Tela exibindo o resultado da verificação dos tipos inferidos.

Terminada a validação da ontologia e a criação de instâncias, é iniciado o teste para verificar a consistência das instâncias. Segundo Lira (2006), o primeiro teste verifica as regras de semântica da ontologia e o segundo verifica se as instâncias atendem a estas regras.

Para a realização deste teste sobre as instâncias não é necessário o uso do raciocinador, pois ele é realizado pela ferramenta *Protégé*. Para iniciar o teste, deve ser escolhida a opção *Run ontology tests...* que é encontrada no menu OWL.

Este teste testará as instâncias definidas no processo de criação da OntoEvento. O resultado é apresentado na parte inferior da ferramenta *Protégé*, como é apresentado na figura 66.

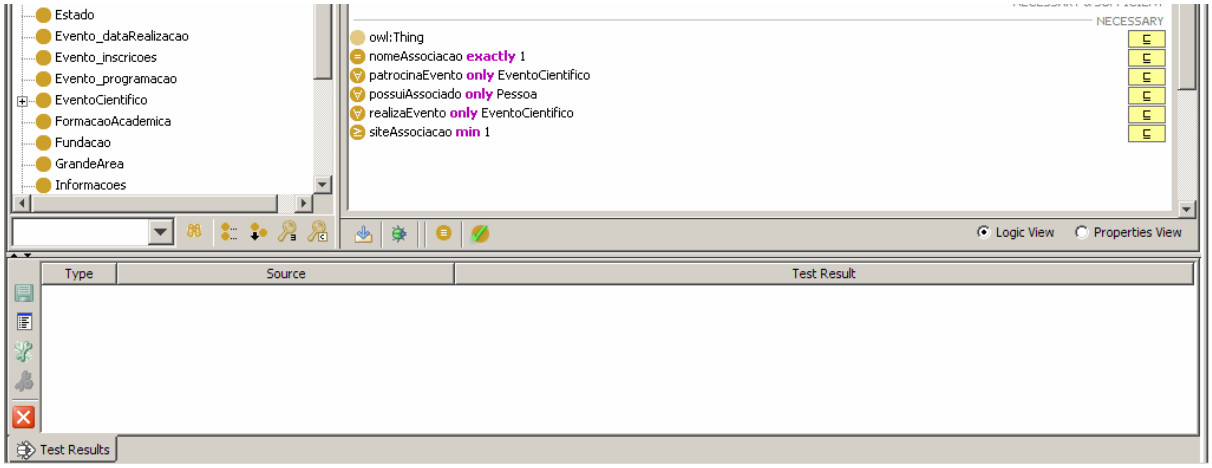

Figura 66 - Tela exibindo o resultado da verificação dos testes da ontologia.

Durante a realização dos testes, não foram encontrados erros e incoerências na OntoEvento. Com isso, esta ontologia é considerada consistente e atende a todos os requisitos especificados durante o seu desenvolvimento.

5 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Ao final do trabalho realizado, conclui-se que as ontologias podem melhorar os sistemas de buscas realizadas na web. Com isso, o uso de ontologias pode ser o primeiro passo para a implantação efetiva dos ideais propostos pelas Web Semântica. Essa conclusão é embasada sobre o fato de que uma ontologia adiciona semântica aos documentos da web, fazendo que ela seja facilmente entendida por agentes humanos e computacionais.

Outro item a favor do uso de ontologias é que aplicações podem compartilhar o conhecimento através da interoperabilidade dos dados. Estas aplicações apenas acessariam os dados da ontologia, deixando a parte de restrições isolada pela ontologia. Ainda podem ser adicionadas ontologias prontas e testadas para um modelo em desenvolvimento, de modo que seja realizadas em menos tempo e com menos esforços para o desenvolvimento.

Sobre os eventos científicos, é necessário citar a quantidade de páginas de eventos que existem na web atualmente. Essas páginas não possuem uma estrutura básica para os dados e se apresentam de forma desordenada e em alguns casos incompletos.

A respeito das tecnologias envolvidas, é necessário salientar a flexibilidade da linguagem OWL e dos modelos RDF e RDF Schema. Essa flexibilidade se deve principalmente devido ao uso da linguagem XML, que permite criações de *tags* em forma de árvore para serem usadas conforme a necessidade. Com essa flexibilidade, é possível descrever o domínio com inúmeras soluções, sem que haja uma solução ideal.

Essa flexibilidade incide sobre o desenvolvimento da OntoEvento. É evidente que esse modelo será alterado devido a inclusão de funcionalidades em eventos científicos. Outro item a ser considerado sobre a OntoEvento é a provável aplicação de técnicas de refinamento para melhoria constante do modelo. Outro ponto sobre a flexibilidade é que devido a essa característica foi possível definir um modelo para todos os eventos pesquisados.

Sobre a OntoEvento, podem ser citados como trabalhos relacionados a inclusão de funcionalidades que não estão descritas neste modelo. Outro item a ser considerado é a legibilidade da ontologia para agentes humanos. Como uma ontologia é um documento com *tags* para representação do conhecimento, ele não é legível para o

humano e o processo de instanciação requer o uso da ferramenta utilizada para edição da ontologia.

Para sanar este problema, sugere-se criar uma aplicação utilizando, por exemplo, a linguagem Java para manipulação da ontologia por parte dos usuários. Este trabalho pode ser realizado através da API Jena, que pode ser manipulada pela linguagem citada.

REFERENCIAS

ANDERSON, Richard et al. **Professional XML**. 1. ed. Rio de Janeiro: Ciência Moderna, 2001. 1266 pg.

ARAÚJO, Moysés de. **Educação a Distância e a Web Semântica**: Modelagem Ontológica de Materiais e Objetos de Aprendizagem para a Plataforma CoL. 2003. 191 p. Tese (Doutorado em Engenharia) - Escola Politécnica da Universisade de São Paulo, São Paulo.

BERNERS-LEE, Tim. *et al.* **The Semantic Web**. Scientific American, 2001. Disponível em: <http://www.sciam.com/2001/0501issue/0501berners-lee.HTML>. Último acesso em: 01 novembro 2007.

BRICKLEY, Dan; GUHA, R.v.. **Resource Description Framework (RDF) Schema Specification 1.0**: W3C Candidate Recommendation 27 March 2000. Disponível em: <http://www.w3.org/TR/2000/CR-rdf-schema-20000327/>. Último acesso em: 04 novembro 2007.

BRITO, Parcilene Fernandes de; LUSTOSA, Pollyane de Almeida; TEIXEIRA, Darlene. **OWL no Desenvolvimento de uma Ontologia para um Sistema de Inteligência Competitiva.** 2004. In: ENCOINFO 2004 – CEULP/ULBRA – Curso de Sistemas de Informação – Palmas – TO. Disponível em: <http://www.ulbra-to.br/ensino/43020/artigos/anais2004/anais/pollyaneLustosaInteligenciaCompetitivaEncoinfo2004.pdf>. Último acesso em: 29 outubro 2007.

BRITO, Parcilene Fernandes de; NOLETO, Danilo de Abreu. **RDF e RDF Schema na representação de páginas do portal CEULP/ULBRA**. In Anais do V Encontro de Estudantes de Informática do Tocantins. Palmas, TO. outubro, 2003. pp. 113-122. Disponível em: <http://www.ulbra-to.br/ensino/43020/artigos/anais2003/anais/rdfschema-encoinfo2003.pdf >. Último acesso em: 15 outubro 2007.

BROESKSTA, Jeen et al. **Enabling Knowledge Representation on the Web by Extending RDF Schema**. 2001. Disponível em: <http://www.cs.vu.nl/~frankh/postscript/WWW01.pdf>. Último acesso em: 18 outubro 2007.

BONIFACIO, Ailton Sergio. **Ontologias e Consulta Semântica:** Uma Aplicação ao Caso Lattes. 2002. 85 p. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal do Rio Grande do Sul, Porto Alegre.

BONIFACIO, Ailton Sergio; HEUSER, Carlos Alberto. **Ontologias e consultas semânticas:** uma aplicação ao caso Lattes. Disponível em: <http://www.uel.br/pessoal/ailton/Trabalhos/SemanaAcad-2002%20-%20Ailton%20-%20Final.htm>. Acesso em: 02 novembro 2007.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML:** Guia do Usuário. 2. ed. Rio de Janeiro: Elsevier, 2005. 474 p.

BRAGANHOLO, Vanessa; HEUSER, Carlos. **XML Schema, RDF(S) e UML**: uma comparaçao. In: IDEAS 2001 - 4th Iberoamerican Workshop on Requirements Engineering and Software Environments, Santo Domingo, Heredia, Costa Rica, 2001. Proceedings... page 78-90. Disponível em: <http://www.inf.ufrgs.br/~vanessa//artigos/ideas2001.pdf>. Último acesso em 04 novembro 2007.

CARVALHO, Cedric L. de; LIMA, Júnio César de. **Ontologias**: OWL (Web Ontology Language). Goiânia: Universidade Federal de Goiás, 2005. 24 p. Disponível em: <http://www.inf.ufg.br/virtualbib/RT-INF_004-05.pdf>. Último acesso em: 29 outubro 2007.

CENDÓN, Beatriz Valadares. **Ferramentas de busca na web**. In: Ci. Inf., Brasília, v. 30, n. 1, p. 39-49, jan./abr. 2001.

CORCHO, Oscar; PERÉZ, Asunción Gómez. **Ontology Languages for Semantic Web. IEEE Intelligent Systems**. p. 54-60, 2002. Disponivel em <www.cs.umbc.edu/771/papers/ieeeIntelligentSystems/webservices/ontologyLanguages.pdf>. Ultimo acesso em: 29 outubro 2007.

DECKER, S. et al. **Ontobroker:** Ontology base Acess to Distributed and Semi-Structured Information. In: Semantic Issues in Multimedia Systems. Proceding oi DS-8, Boston: Academic Publisher, 1999. p. 351-369.

DEITEL, H. M.. **Java:** Como Programar. 6. ed. Porto Alegre: Bookman, 2004. 2129 p.

GRAVES, Mark. **Projeto de Banco de Dados com XML.** São Paulo: Pearson Education do Brasil, 2003. 518 p.

GRUBER, Thomas.R. **A Translation Approach to Portable Ontologies. Knowledge Acquisition**, v.5, n.2, p.199-200, 1993. Disponível em: <http://tomgruber.org/writing/ontolingua-kaj-1993.pdf>. Último acesso em: 19 outubro 2007.

GUARINO, Nicola. **Formal Ontology and Information Systems.** In: First International Conference (FOIS), Trento, Itália. Anais: Trento: IOS Press, 1998.

GUIZZARDI, Giancarlo. (2000), **Análise de Domínio e Ontologias**. Vitória, 2000. Programa de Mestrado em Informática – Universidade Federal do Espírito Santo, Vitória, 2000. Disponível em: <http://wwwhome.cs.utwente.nl/~guizzard/MSc/> Último acesso em 21 outubro 2007.

HEUSER, Carlos Alberto. **Projeto de Banco de Dados.** 4. ed. Porto Alegre: Sagra Luzzato, 2004. 236 p.

HORRIDGE, M. et. al. **A Practical Guide to Building OWL Ontologies Using the Protégé-OWL Plugin and CO-ODE Tools Edition 1.0. Tutorial**. The University of Manchester,2004.

LIRA, Heremita Brasileiro. **Uma Ontologia para a Descrição da Semântica da IMML**. 2006. 195 p. Dissertação (Mestrado em Sistemas e Computação) – Universidade Federal do Rio Grande do Norte, Natal.

LUSTOSA, Pollyane de Almeida. **OWL e Protégé**: estudo e aplicação de conceitos para definição de ontologias Monografia – curso de Sistemas de Informação, Centro Universitário Luterano de Palmas. Disponível em: <http://www.ulbra-to.br/ensino/43020/artigos/TCC2003/> Último acesso: 18 outubro 2007.

MARIETTO, M. B. et al. Requeriments analysis of multi-agent-based simulation platforms. Technical report. São Paulo University, 2002

MCGUINNESS, D. L; SMITH, M. K; WELTY, C. **OWL Web Ontology Language Guide**. Disponível em <http://www.w3.org/TR/2004/REC-owl-guide-20040210/>, Último acesso em 03 novembro 2007.

MELLO, Ronaldo. *et al.* **Dados Semi-Estruturados**. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, 15., 2000, João Pessoa, PB. Anais... João Pessoa: [s.n.], 2000.

NOY, Natalya F, MCGUINNESS, Deborah L. **Ontology Development 101**: A Guide to Creating Your First Ontology. Disponível em <http://protege.stanford.edu/publications/ontology_development/ontology101.html>. Último acesso: 02 novembro 2007.

OLIVEIRA, Lucas Gonçalves de. **Construção de um Sistema de Blackboard para a Gestão de Documentos Usando XML** 2004. 81 p. Monografia (Bacharel em Sistemas de Informação) – Pontífice Universidade Católica, São Paulo.

PRESSMAN, Roger. **Engenharia de Software.** 6. ed. São Paulo: Mcgraw-hill, 2006. 720 p.

RACER. **RacerPro:** User´s Guide Version 1.9.2. Racer Systems - Gmbh & Co. Kg, 2007. Disponível em: <http://www.racer-systems.com>. Último acesso em 04 novembro 2007.

SANTOS, Domingos Sávio Apolônio. **RDF na Interoperabilidade entre os** Domínios na WEB. 2002. 121 p. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Campina Grande, Campina Grande. Disponível em: <http://www.dsc.ufcg.edu.br/~copin/pesquisa/bancodissertacoes/2002/domingossavio.pdf>. Último acesso em: 20 outubro 2007.

SILVA, Nalva Neila da. **A Utilização da Linguagem OWL na Definição de uma Ontologia para o Currículo Lattes.** 2004a. 112 p. Monografia (Bacharel em Sistemas de Informação) – Centro Universitário Luterano de Palmas, Palmas. Disponível em <http://www.ulbra-to.br/ensino/43020/artigos/relatorios2004-1/TCC/Estagio_Nalva.pdf >. Último acesso em 23 outubro 2007.

SOUTO, Maria Aparecida Martins; WARPECHOWSKI, Mariusa; OLIVEIRA, José Palazzo Moreira de . **Modelo de Avaliação da Qualidade de Conferências Científicas na Área da Ciência da Computação:** uma Abordagem Ontológica. In: WOMSDE - SBBD/SBES, 2006, Florianópolis. Florianópolis: Sociedade Brasileira de Computação, 2006, p. 93-102.

STANFORD. **What is Protégé OWL?.** Home page. Disponível em: <http://protege.stanford.edu/overview/protege-owl.html>. Último acesso em: 22 outubro 2007.

USCHOLD, M., GRUNINGER M. **Ontologies**: principles, methods and applications. The Knowledge Engineering Review, v. 11, n. 2, p. 93-136, 1996.

W3C. **XML SCHEMA.** Home page. Disponível em: <http://www.w3.org/TR/xmlschelma-0/>. Último acesso em: 22 outubro 2007.

APÊNDICES

APÊNDICE A – PROPRIEDADES UTILIZADAS NA ONTOEVENTO

Neste apêndice serão mostradas as propriedades que fazem parte da OntoEvento. Serão especificados os elementos aceitos como domínio e valor, suas características e propriedades inversas.

Object Properties

cidadeInstituicao
Domínio: *EnderecoInstituicao*
Valores aceitos: *Cidade*

realizaEvento
Domínio: *Associacao, Empresa, Fundacao, Instituicao ou OrgaoGovernamental*
Valores aceitos: *EventoCientifico*
Propriedade inversa: *realizadoPor*

patrocinaEvento
Domínio: *Associacao, Empresa, OrgaoGovernamental ou Fundacao*
Valores aceitos: *EventoCientifico*
Propriedade inversa: *patrocinadoPor*

possuiAssociado
Domínio: *Associacao*
ValoresAceitos: *Pessoa*
Propriedade inversa: *associadoA*

pertenceAoEstado
Domínio: *Cidade*
Valores aceitos: *Estado*
Propriedade inversa: *possuiCidades*

possuiSubAreas
Domínio: *Areas*
Valores aceitos: *SubArea*
Propriedade inversa: *pertenceAArea*

pertenceAGrandeArea
Domínio: *Areas*
Valores aceitos: *GrandeArea*
Propriedade inversa: *possuiAreas*

ehComissaoDe
Domínio: *ComissaoOrganizadora*
Valores aceitos: *EventoCientifico*
Propriedade inversa: *possuiComissaoOrganizadora*

compostaPor
Domínio: *ComissaoOrganizadora*
Valores aceitos: *Pessoa*
Propriedade inversa: *compoeComissao*

possuiAreas
Domínio: *GrandeArea*
Valores aceitos: *Areas*
Propriedade inversa: *pertenceAGrandeArea*

compoeComissao
Domínio: *Pessoa*
Valores aceitos: *ComissaoOrganizadora*
Propriedade inversa: *compostaPor*

possuiFormacao
Domínio: *Pessoa*
Valores aceitos: *FormacaoAcademica*

apresentadorPrograma
Domínio: *Pessoa*
Valores aceitos: *Evento_programacao*
Propriedade inversa: *possuiApresentador*

associadoA
Domínio: *Pessoa*
Valores aceitos: *Associacao*
Propriedade inversa: *possuiAssociado*

possuiTrabalhosAceitos
Domínio: *Pessoa*
Valores aceitos: *TrabalhosAceitos*
Propriedade inversa: *possuiAutor*

empregadoDe
Domínio: *Pessoa*
Valores aceitos: *Instituicao ou Empresa*
Propriedade inversa: *possuiEmpregado*

membroDe
Domínio: *Pessoa*
Valores aceitos: *Fundacao*
Propriedade inversa: *possuiMembros*

ministradoNa
Domínio: *Cursos*
Valores aceitos: *Instituicao*

pertenceArea
Domínio: *Cursos*
Valores aceitos: *GrandeArea*
Propriedade inversa: *possuiCursos*

possuiFormados
Domínio: *Instituicao*
Valores aceitos: *FormacaoAcademica*
Propriedade inversa: *formadoNa*

sediaEvento
Domínio: *Instituicao ou LocalRealizacao*
Valores aceitos: *EventoCientifico*
Propriedade inversa: *sediadoEm*

possuiProgramas
Domínio: *EventoCientifico*
Valores aceitos: *Evento_programacao*
Propriedade inversa: *programacaoDe*

sediadoEm
Domínio: *EventoCientifico*
Valores aceitos: *Instituicao ou LocalRealizacao*
Propriedade inversa: *sediaEvento*

possuiDatas
Domínio: *EventoCientifico*
Valores aceitos: *Evento_dataRealizacao*
Propriedade inversa: *datasDoEvento*

possuiExpositor
Domínio: *EventoCientifico*
Valores aceitos: *Empresas*
Propriedade inversa: *expoeProdutos*

possuiComissaoOrganizadora
Domínio: *EventoCientifico*
Valores aceitos: *ComissaoOrganizadora*
Propriedade inversa: *ehComissaoDe*

realizadoPor
Domínio: *EventoCientifico*
Valores aceitos: *Associacao, Empresa, Instituicao, Fundacao ou OrgaoGovernamental*
Propriedade inversa: *realizaEvento*

areasEvento
Domínio: *EventoCientifico*
Valores aceitos: *SubArea*
Propriedade inversa: *possuiEventosRelacionados*

eventosRelacionados
Domínio: *EventoCientifico*
Valores aceitos: *EventoCientifico*

dataInscricoes
Domínio: *EventoCientifico*
Valores aceitos: *Evento_inscricoes*
Propriedade inversa: *inscricoesEvento*

possuiInformacoes
Domínio: *EventoCientifico*
Valores aceitos: *Informacoes*
Propriedade inversa: *infoPertenceAoEvento*

patrocinadoPor
Domínio: *EventoCientifico*
Valores aceitos: *Empresa, Associacao, OrgaoGovernamental ou Fundacao*
Propriedade inversa: *patrocinaEvento*

possuiArtigosAceitos
Domínio: *EventoCientifico*
Valores aceitos: *TrabalhosAceitos*
Propriedade inversa: *aceitosNoEvento*

possuiEmpregado
Domínio: *Empresa ou Instituicao*
Valores aceitos: *Pessoa*
Propriedade inversa: *empregadoDe*

expoeProdutos
Domínio: *Empresa*
Valores aceitos: *EventoCientifico*
Propriedade inversa: *possuiExpositor*

possuiMembros
Domínio: *Fundacao*
Valores aceitos: *Pessoa*
Propriedade inversa: *membroDe*

infoPertenceAoEvento
Domínio: *Informacoes*
Valores aceitos: *EventoCientifico*
Propriedade inversa: *possuiInformacoes*

possuiEstados
Domínio: *Pais*
Valores aceitos: *Estado*
Propriedade inversa: *pertenceAoPais*

possuiAutor
Domínio: *TrabalhosAceitos*
Valores aceitos: *Pessoa*
Propriedade inversa: *possuiTrabalhosAceitos*

aceitoNoEvento
Domínio: *TrabalhosAceitos*
Valores aceitos: *EventoCientifico*
Propriedade inversa: *possuiArtigosAceitos*

possuiCidades
Domínio: *Estado*
Valores aceitos: *Cidades*
Propriedade inversa: *pertenceAoEstado*

pertenceAoPais
Domínio: *Estado*
Valores aceitos: *Pais*
Propriedade inversa: *possuiEstados*

possuiEventosRelacionados
Domínio: *SubArea*
Valores aceitos: *EventoCientifico*
Propriedade inversa: *areasEvento*

pertenceAArea
Domínio: *SubArea*
Valores aceitos: *Areas*
Propriedade inversa: *possuiSubAreas*

areaFormacao
Domínio: *FormacaoAcademica*
Valores aceitos: *Areas*

formadoNa
Domínio: *FormacaoAcademica*
Valores aceitos: *Instituicao*
Propriedade inversa: *possuiFormados*

programacaoDe
Domínio: *Evento_programacao*
Valores aceitos: *EventoCientifico*
Propriedade inversa: *possuiProgramas*

possuiApresentador
Domínio: *Evento_programacao*
Valores aceitos: *Pessoa*
Propriedade inversa: *apresentadorPrograma*

datasDoEvento
Domínio: *Evento_dataRealizacao*
Valores aceitos: *EventoCientifico*
Propriedade inversa: *possuiDatas*

inscricoesEvento
Domínio: *Evento_inscricoes*
Valores aceitos: *EventoCientifico*
Propriedade inversa: *dataInscricoes*

possuiCursos
Domínio: *GrandeArea*
Valores aceitos: *Cursos*
Propriedade inversa: *pertenceArea*

cidadeRealizacao
Domínio: *LocalRealizacao ou Instituicao*
Valores aceitos: *Cidade*
Propriedade inversa: *possuiLocal*
Característica: *Propriedade funcional*

possuiEndereco
Domínio: *Instituicao*
Valores aceitos: *EnderecoInstituicao*
Propriedade inversa: *ehEnderecoDaInstituicao*
Característica: *Propriedade funcional*

cursoFormacao
Domínio: *FormacaoAcademica*
Valores aceitos: *Cursos*
Característica: *Propriedade funcional*

ehEnderecoDaInstituicao
Domínio: *EnderecoInstituicao*
Valores aceitos: *Instituicao*
Propriedade inversa: *possuiEndereco*
Característica: *Propriedade funcional inversa*

possuiLocal
Domínio: *Cidade*
Valores aceitos: *LocalRealizacao ou Instituicao*
Propriedade inversa: *cidadeRealizacao*
Característica: *Propriedade funcional inversa*

Datatype Properties

telefoneInstituicao
Domínio: *EnderecoInstituicao*
Valores aceitos: *xsd:string*

enderecoInstituicao
Domínio: *EnderecoInstituicao*
Valores aceitos: *xsd:string*

bairroInstituicao
Domínio: *EnderecoInstituicao*
Valores aceitos: *xsd:string*

emailInstituicao
Domínio: *EnderecoInstituicao*
Valores aceitos: *xsd:string*

cepInstituicao
Domínio: *EnderecoInstituicao*
Valores aceitos: *xsd:string*

siteAssociacao
Domínio: *Associacao*
Valores aceitos: *xsd:string*

nomeAssociacao
Domínio: *Associacao*
Valores aceitos: *xsd:string*

nomeCidade
Domínio: *Cidade*
Valores aceitos: *xsd:string*

nomeArea
Domínio: *Areas*
Valores aceitos: *xsd:string*

nomeGrandeArea
Domínio: *GrandeArea*
Valores aceitos: *xsd:string*

nomePessoa
Domínio: *Pessoa*
Valores aceitos: *xsd:string*

nomeCurso
Domínio: *Cursos*
Valores aceitos: *xsd:string*

siteInstituicao
Domínio: *Instituicao*
Valores aceitos: *xsd:string*

nomeInstituicao
Domínio: *Instituicao*
Valores aceitos: *xsd:string*

nomeEvento
Domínio: *EventoCientifico*
Valores aceitos: *xsd:string*

edicaoEvento
Domínio: *EventoCientifico*
Valores aceitos: *xsd:string*

siglaEvento
Domínio: *EventoCientifico*
Valores aceitos: *xsd:string*

siteEmpresa
Domínio: *Empresa*
Valores aceitos: *xsd:string*

nomeEmpresa
Domínio: *Empresa*
Valores aceitos: *xsd:string*

siteFundacao
Domínio: *Fundacao*
Valores aceitos: *xsd:string*

nomeFundacao
Domínio: *Fundacao*
Valores aceitos: *xsd:string*

tituloInformacao
Domínio: *Informacoes*
Valores aceitos: *xsd:string*

assuntoInformacao
Domínio: *Informacoes*
Valores aceitos: *xsd:string*

dataInformacao
Domínio: *Informacoes*
Valores aceitos: *xsd:date*

siglaPais
Domínio: *Pais*
Valores aceitos: *xsd:string*

nomePais
Domínio: *Pais*
Valores aceitos: *xsd:string*

enderecoLocal
Domínio: *LocalRealizacao*
Valores aceitos: *xsd:string*

emailLocal
Domínio: *LocalRealizacao*
Valores aceitos: *xsd:string*

cepLocal
Domínio: *LocalRealizacao*
Valores aceitos: *xsd:string*

nomeTrabalho
Domínio: *TrabalhosAceitos*
Valores aceitos: *xsd:string*

siglaEstado
Domínio: *Estado*
Valores aceitos: *xsd:string*

nomeEstado
Domínio: *Estado*
Valores aceitos: *xsd:string*

nomeOrgao
Domínio: *OrgaoGovernamental*
Valores aceitos: *xsd:string*

nomeSubArea
Domínio: *SubArea*
Valores aceitos: *xsd:string*

anoFormacao
Domínio: *FormacaoAcademica*
Valores aceitos: *xsd:string*

tipoFormacao
Domínio: *LocalRealizacao*
Valores aceitos: *xsd:string*
{Graduacao, Mestrado, Doutorado, Pos-doutorado ou Especializacao}

tipoPrograma
Domínio: *Evento_programacao*
Valores aceitos: *xsd:string*
{Palestra, Minicurso ou Tutorial}

dataPrograma
Domínio: *Evento_programacao*
Valores aceitos: *xsd:date*

nomePrograma
Domínio: *Evento_programacao*
Valores aceitos: *xsd:string*

finalEvento
Domínio: *Evento_dataRealizacao*
Valores aceitos: *xsd:date*

inicioEvento
Domínio: *Evento_dataRealizacao*
Valores aceitos: *xsd:date*

dataFinal
Domínio: *Evento_inscricoes*
Valores aceitos: *xsd:date*

nomeLocal
Domínio: *LocalRealizacao*
Valores aceitos: *xsd:string*

finalSubmissao
Domínio: *Evento_dataRealizacao*
Valores aceitos: *xsd:date*

tipoCongresso
Domínio: *Congresso*
Valores aceitos: *xsd:string*
{Técnico ou Cientifico}

regraEnvioTrabalhos
Domínio: *EventoCientifico*
Valores aceitos: *xsd:string*

telefoneLocal
Domínio: *LocalRealizacao*
Valores aceitos: *xsd:string*

inicioSubmissao
Domínio: *Evento_dataRealizacao*
Valores aceitos: *xsd:date*

publicoAlvo
Domínio: *EventoCientifico*
Valores aceitos: *xsd:string*

atracoesEvento
Domínio: *EventoCientifico*
Valores aceitos: *xsd:string*

urlMapaEvento
Domínio: *EventoCientifico*
Valores aceitos: *xsd:string*
Característica: *Propriedade Funcional*

siteOrgao
Domínio: *OrgaoGovernamental*
Valores aceitos: *xsd:string*
Característica: *Propriedade Funcional*

siglaInstituicao
Domínio: *Instituicao*
Valores aceitos: *xsd:string*
Característica: *Propriedade Funcional*

dataInicio
Domínio: *Evento_inscricoes*
Valores aceitos: *xsd:date*
Característica: *Propriedade Funcional*

historicoPessoa
Domínio: *Pessoa*
Valores aceitos: *xsd:string*
Característica: *Propriedade Funcional*

siglaOrgao
Domínio: *OrgaoGovernamental*
Valores aceitos: *xsd:string*
Característica: *Propriedade Funcional*

valorProgramaAdicional
Domínio: *Evento_inscricoes*
Valores aceitos: *xsd:float*
Característica: *Propriedade Funcional*

siglaEmpresa
Domínio: *Empresa*
Valores aceitos: *xsd:string*
Característica: *Propriedade Funcional*

regimentoEvento
Domínio: *EventoCientifico*
Valores aceitos: *xsd:string*
Característica: *Propriedade Funcional*

tipoInscricao
Domínio: *Evento_inscricoes*
Valores aceitos: *xsd:string*
Característica: *Propriedade Funcional*

localRealizacao
Domínio: *Evento_programacao*
Valores aceitos: *xsd:string*
Característica: *Propriedade Funcional*

informacoesEvento
Domínio: *EventoCientifico*
Valores aceitos: *xsd:string*
Característica: *Propriedade Funcional*

objetivoEvento
Domínio: *EventoCientifico*
Valores aceitos: *xsd:string*
Característica: *Propriedade Funcional*

siglaAssociacao
Domínio: *Associacao*
Valores aceitos: *xsd:string*
Característica: *Propriedade Funcional*

siglaFundacao
Domínio: *Fundacao*
Valores aceitos: *xsd:string*
Característica: *Propriedade Funcional*

valorInscricao
Domínio: *Evento_inscricoes*
Valores aceitos: *xsd:float*
Característica: *Propriedade Funcional*