

**Universidade Federal de Campina Grande  
Centro de Ciências e Tecnologia  
Curso de Mestrado em Informática  
Coordenação de Pós-Graduação em Informática**

**OntoEditor: Um editor para manipular  
ontologias na Web**

**Karine de Freitas Vasconcelos**

Campina Grande - PB  
Janeiro - 2003

**Universidade Federal de Campina Grande**  
**Centro de Ciências e Tecnologia**  
**Curso de Mestrado em Informática**  
**Coordenação de Pós-Graduação em Informática**

**OntoEditor: Um editor para manipular  
ontologias na Web**

**Karine de Freitas Vasconcelos**

Dissertação submetida à Coordenação de Pós-Graduação em Informática do Centro de Ciências e Tecnologia da Universidade Federal de Campina Grande como requisito parcial para a obtenção do grau de Mestre em Ciências (MSc)

**Prof. Dr. Cláudio de Souza Baptista**  
**(Orientador)**

**Campina Grande - PB**  
**Janeiro - 2003**

FICHA CATALOGRÁFICA

---

VASCONCELOS, Karine de Freitas

F2240

OntoEditor: Um Editor para Manipular Ontologias na Web

Dissertação (Mestrado), Universidade Federal de Campina Grande, Centro de Ciências e Tecnologia, Coordenação de Pós-Graduação em Informática, Campina Grande – Pb, Fevereiro de 2003.

124p. II.

Orientador: Cláudio de Souza Baptista, Dr.

Palavras Chave:

1. Banco de Dados
2. Representação de Conhecimento
3. Ontologias
4. RDF
5. XML

CDU – 681.3.07B

UFCG - BIBLIOTECA - CAMPUS I	
783	04-06-2003

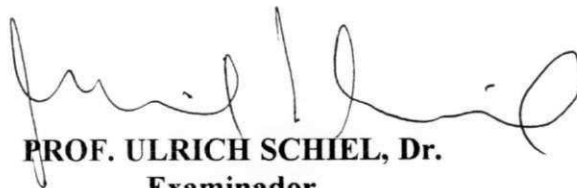
**“ONTOEDITOR: UM EDITOR PARA MANIPULAR ONTOLOGIAS NA  
WEB”**

**KARINE DE FREITAS VASCONCELOS**

**DISSERTAÇÃO APROVADA EM 24.02.2003**



**PROF. CLAUDIO DE SOUZA BAPTISTA , Ph.D  
Orientador**



**PROF. ULRICH SCHIEL, Dr.  
Examinador**



**PROF. ARTURO HERNÁNDEZ DOMÍNGUEZ, Dr.  
Examinador**

**CAMPINA GRANDE – PB**

*Para Gabriel.*

## Agradecimentos

Como “não há nada como um sonho para criar um futuro”, eu considero a conclusão desse trabalho a realização de um sonho que foi muito almejado e que, dentro das minhas expectativas, sempre teve uma importância e um significado muito especiais.

Definitivamente, por mais capazes que possamos ser, é muito complicado darmos passos tão significativos sozinhos e para mim foi muito gratificante poder contar com a experiência, o estímulo e a confiança de outras pessoas que me ajudaram a realizar essa dissertação. Gostaria de agradecer a todos de uma forma geral desde os que tiveram uma participação mais efetiva até os que, em algumas horas de cansaço e desânimo, confortaram-me com singelas e amigas palavras. De uma forma especial, agradeço:

- A Deus, sem o qual eu descredito que algo de bom possa ser alcançado;
- Ao meu amado filho, Gabriel, minha fonte alegria;
- Aos meus pais, Fernando e Gésia, e minha irmã, Fernanda, por toda dedicação e carinho;
- A minha querida tia-mãe, Marita, minha fonte de força e determinação;
- Ao meu amigo-irmão, André, pessoa indiscutivelmente especial, que considero um irmão com todo orgulho.
- As minhas grandes amigas, Nildinha e Jamile, por toda confiança, companheirismo e amizade;
- Ao meu orientador, professor Cláudio, a quem agradeço por toda colaboração e paciência;
- A todos os professores do curso de mestrado, com os quais eu tive a grata oportunidade de aprender muito. Vocês são show de bola!!!!

## Sumário

Capítulo I - INTRODUÇÃO.....	1
Capítulo II - FUNDAMENTAÇÃO TEÓRICA.....	8
2.1. Dados na Web .....	8
2.2. A Web Semântica (Semantic Web).....	12
2.2.1. O Conceito de Metadado.....	13
2.2.2. Dublin Core .....	14
2.3. O papel das Ontologias .....	15
2.4. Linguagens de Representação de Metadados.....	21
2.4.1. RDF (Resource Description Framework) .....	22
2.4.1.1. Tipos de Representação das Sentenças RDF .....	23
2.4.1.2 Sentenças sobre sentenças .....	26
2.4.1.3. Contêineres RDF .....	27
2.4.1.4. Esquemas RDF.....	30
2.5 Linguagens de Representação para Ontologias.....	33
2.5.1 OIL (Ontology Inference Layer ou Ontology Interchange Language) .....	34
2.5.2. DAML+OIL .....	36
2.5.3 OWL (Ontology Web Language) .....	37
2.6 Linguagem de Consulta – XMLQL.....	40
2.7 Trabalhos Relacionados .....	42
2.7.1 Protégé.....	42
2.7.2 KAON-SOEP (Simple Ontology Browser and Editor Plugin).....	45
2.7.3 OilEd .....	47
2.7.4 Quadro Comparativo com principais características das ferramentas .....	49
2.8 Conclusão.....	51
Capítulo III - PROJETO ONTOEDITOR .....	52
3.1 O Contexto do OntoEditor .....	52
3.2 Arquitetura do OntoEditor .....	54
3.2.1 Arquitetura Detalhada do OntoEditor.....	55

3.2.1.1 Interface OntoEditor.....	56
3.2.1.2 Processo de Tradução de Ontologias OntoEditor .....	58
3.2.1.3 Armazenamento e Recuperação de Ontologias no OntoEditor .....	59
3.2.1.4 Consultas OntoEditor .....	60
3.3 Modelo Conceitual .....	61
3.4 Levantamento de Requisitos .....	64
3.4.1 Requisitos Funcionais do OntoEditor .....	64
3.4.2 Requisitos Funcionais da Jena.....	67
3.4.3 Requisitos Funcionais do XML-QL.....	67
3.5 Planejamento dos Incrementos.....	68
3.6 Estudo de Caso .....	69
3.7 Conclusão .....	77
Capítulo IV - IMPLEMENTAÇÃO DO ONTOEDITOR.....	78
4.1 A Camada de Aplicação.....	78
4.1.1 Applets Java.....	79
4.1.2. Interface – Ontologia Gráfica .....	81
4.1.2.1. OpenJGraph - API Gráfica .....	81
4.1.3 Ontologia Lógica .....	82
4.2 A Camada de Dados .....	83
4.3 Processo de Tradução .....	84
4.3.1 Validação do Processo de Tradução .....	87
4.4 Consultas – Máquina de Busca .....	92
4.4.1 Utilização do JDBC .....	93
4.4.2 Consultas através das Propriedades da Ontologia .....	94
4.4.3 Consultas através do Conteúdo da Ontologia.....	95
4.5 Diagrama de Classes.....	100
4.6 Conclusão .....	103
Capítulo V - CONCLUSÃO .....	104
5.1 Trabalhos Futuros.....	106
Referências Bibliográficas.....	108



## Lista de Abreviaturas

API	Application-Programming Interface
DAML	DARPA Agent Markup Language
DARPA	Defense Advanced Research Projects Agency
DC	Dublin Core
DTD	Document Type Description
EBNF	Extended Backus-Naur Form
FaCT	Fast Classification of Terminologies
HTML	Hypertext Markup Language
IA	Inteligência Artificial
ISO	International Organization for Standardization
JDBC	Java Database Connectivity
KAON	Karlsruhe Ontology and Semantic Web Infrastructure
KAON-SOEP	Simple Ontology Browser and Editor Plugin
KRS	Sistemas de Representação de Conhecimento
OIL	Ontology Inference Layer ou Ontology Interchange Language
OKBC	Open Knowledge-Base Connectivity
OWL	Ontology Web Language
RDF	Resource Description Framework
SGML	Standard Generalized Markup
URIs	Uniform Resource Identifiers
W3C	World Wide Web Consortium
XML	Extensible Markup Language

## Lista de Figuras

FIGURA 1.1 Ilustração da Estrutura Semântica do campo RDF .....	6
FIGURA 2.1 Exemplo de Dados Semi-Estruturados.....	9
FIGURA 2.2 Exemplo ilustrando a utilização de elementos do Dublin Core em HTML 15	
FIGURA 2.3 Exemplo ilustrando a definição de uma ontologia para Vinhos. ....	19
FIGURA 2.4 Visão abstrata da estrutura das informações na <i>Web Semântica</i> .....	21
FIGURA 2.5 Sentença RDF representada num grafo .....	24
FIGURA 2.6 Exemplo de Sentença RDF escrita em sintaxe XML.....	25
FIGURA 2.7 Representação do Exemplo 2.2 em RDF.....	27
FIGURA 2.8 Especificação RDF para um Esquema que descreve Alunos .....	32
FIGURA 2.9 Exemplificando o uso dos atributos max-cardinality e min-cardinality.....	35
FIGURA 2.10 Exemplo de uma consulta na sintaxe XML-QL.....	41
FIGURA 2.11 Esquema utilizado na consulta do exemplo mostrado na FIGURA 2.10 ..	41
FIGURA 2.12 Interface do Protégé-2000.....	44
FIGURA 2.13 Interface do KAON-SOEP.....	46
FIGURA 2.14 Interface do OilEd .....	48
FIGURA 3.1 Visão Geral da Arquitetura do OntoEditor.....	55
FIGURA 3.2 Exemplo comparativo entre a representação de elementos de uma ontologia sob a forma de grafo e sob a forma de árvore.....	56
FIGURA 3.3 Visão geral do processo de Tradução.....	59
FIGURA 3.4 Modelo Conceitual do OntoEditor .....	63
FIGURA 3.5 Diagrama de Casos de Uso .....	64
FIGURA 3.6 Criando uma nova ontologia.....	70
FIGURA 3.7 Definição de termos .....	71
FIGURA 3.8 Definindo características do termo.....	71
FIGURA 3.9 Definindo o nome e descrição do termo.....	72
FIGURA 3.10 Interface para definir novas propriedades para o termo ou adicionar ao termo propriedades já existentes .....	72
FIGURA 3.11 Definição de relacionamentos entre termos.....	73
FIGURA 3.12 Definindo o nome do relacionamento ou excluindo relacionamento.....	73

FIGURA 3.13 Definição do nome do relacionamento.....	74
FIGURA 3.14 Representação dos termos após a definição de um relacionamento.....	74
FIGURA 3.15 Visualização do RDF gerado a partir da representação mostrada na FIGURA 3.14.....	75
FIGURA 3.16 Menu Ontologia do OntoEditor .....	75
FIGURA 3.17 Definindo as Propriedades da Ontologia .....	76
FIGURA 3.18 Ontologia Universidade.....	77
FIGURA 4.1 Visão geral das principais características de implementação da camada de aplicação .....	79
FIGURA 4.2 Mensagem de segurança do Java <i>Plug-in</i> .....	80
FIGURA 4.3 Implementação OntoEditor/Interface Gráfica .....	82
FIGURA 4.4 Visão específica do processo de Tradução .....	85
FIGURA 4.5 Implementação OntoEditor/Processo de Tradução .....	85
FIGURA 4.6 Parte da Ontologia Universidade .....	88
FIGURA 4.7 Documento RDF gerado a partir da ontologia mostrada na FIGURA 4.6 ..	89
FIGURA 4.8 Implementação do OntoEditor/Driver utilizado para acesso ao banco de dados.....	93
FIGURA 4.9 Implementação do OntoEditor/Conexão com o banco de dados.....	93
FIGURA 4.10 Implementação do OntoEditor/Parâmetros utilizados para conexão .....	94
FIGURA 4.11 Interface OntoEditor para Consulta por Propriedades da Ontologia.....	95
FIGURA 4.12 Implementação do OntoEditor/ Consulta por Propriedades da Ontologia	95
FIGURA 4.13 Interface Gráfica OntoEditor para Consulta por Conteúdo .....	96
FIGURA 4.14 Implementação do OntoEditor/ Consulta por Conteúdo 1 .....	97
FIGURA 4.15 Implementação do OntoEditor/ Consulta por Conteúdo 2 .....	97
FIGURA 4.16 Implementação do OntoEditor/ Consulta por Conteúdo 3 .....	98
FIGURA 4.17 Algoritmo Consulta_Por_Conteúdo.....	99
FIGURA 4.18 Algoritmo Consulta_XML_Processor.....	100
FIGURA 4.19 Diagrama de Classes .....	101

## Lista de Tabelas

TABELA 2.1 Representação de uma sentença RDF na forma de Triplas .....	25
TABELA 2.2 Quadro comparativo das principais ferramentas.....	50
TABELA 3.1 Requisitos Funcionais do OntoEditor.....	67
TABELA 3.2 Requisitos Funcionais da Jena .....	67
TABELA 3.3 Requisitos Funcionais do XML-QL .....	67
TABELA 4.1 Tabela de Campos que compõem as Ontologias.....	83
TABELA 4.2 Mapeamento entre elementos do Modelo de Dados OntoEditor e do Modelo RDF .....	87
TABELA 4.3 Triplas referentes ao documento da FIGURA 4.6, geradas pelo SiRPAC .	91

## Resumo

O contínuo crescimento da *Web* tem desencadeado o surgimento de novas necessidades de serviços, cada vez mais especializados e cada vez mais independentes da interferência humana. As pesquisas nos direcionam ao consenso de que, para alcançar esse nível de inteligência, as aplicações devem possuir um suporte de informações que ofereça um entendimento semântico das informações disponibilizadas na *Web*. Assim, aos dados disponibilizados na *Web*, seriam agregadas, além de informações que representam suas características sintáticas, outras informações com características semânticas, criando, assim, uma nova camada de informações. Para suportar essa nova estrutura, tem-se utilizado o conceito de ontologias, que de uma forma bem simplória, poderia ser considerada como um vocabulário que conteria as informações suficientes para dar suporte semântico a diversos serviços e aplicações. Nessa dissertação, apresentamos uma ferramenta para manipular ontologias na *Web* que oferece alguns diferenciais das demais existentes, tais como: acesso integrado a um banco de dados, com facilidades para consultar e manipular as ontologias armazenadas, bem como uma nova forma de definir e representar graficamente as ontologias criadas pelos seus usuários.

## **Abstract**

The continuous growth of the Web usage has given rise to new service needs even more specialized and independent from the human interference. Researches takes the consensus that, to reach this level of intelligence, the applications must have a support that can help in the semantic understanding of the available information on the Web. In this way, the available Web data, could incorporate information that represents it and other pieces of syntactic information that takes semantic characteristics, creating, in this way, a new layer of information. In order to support this new structure, the concept of ontologies has been used. In a very simple way, a ontology could be considered as a vocabulary that keep the necessary information to give semantic support to various services and applications. In our dissertation, we present a new tool which deals with ontologies in the Web and that varies a little from the ones we have nowadays, integrating access to a data base, with facilities to check and deal with the stored ontologies. We present also a new way to redefine and graphically represent the ontologies that have been created by their users.

# Capítulo I - INTRODUÇÃO

As facilidades e o vasto alcance trazidos pela geração WWW de comunicação têm tornado cada vez mais forte e crescente o uso da *Web* por diversas pessoas e sistemas de informação para publicação e intercâmbio de dados.

A *Web* possui uma diversidade de usuários e de serviços muito grande que disponibilizam e requisitam os mais variados tipos de fontes de informações. A integração dessas informações é indispensável para satisfazer qualquer tipo de comunicação e, na *Web*, ela é garantida através de alguns padrões. Dentro da arquitetura *Web*, esses padrões asseguram a interoperabilidade em vários níveis e viabilizam a existência de diversos serviços, disponibilizados e bastante requisitados, como consultas na *Web*, comércio eletrônico, etc. Entre os padrões utilizados, podemos destacar [Decker et al. – 2000]:

- O protocolo TCP/IP assegura o transporte de bits na rede;
- O protocolo HTTP utilizado para identificar endereços das fontes de informações na *Web*; e
- As linguagens HTML e XML definem a forma como as informações devem ser descritas dentro dos documentos, ou seja a estrutura sintática das informações.

Como qualquer processo evolutivo acaba por esgotar ao máximo as técnicas disponibilizadas, o contínuo crescimento da *Web* tem desencadeado novas necessidades de serviços e isso tem alimentado pesquisas científicas na busca de novas soluções que atendam as mesmas.

Graças ao imprescindível papel que a *Web* vem ocupando na comunicação mundial, vários pesquisadores têm concentrado seus esforços, ao longo dos anos, em definir a melhor forma de representar e posteriormente manipular as informações nesta rede para atender as contínuas necessidades dos seus usuários. Todavia, para viabilizar e acompanhar todo o

desenvolvimento da *Web*, algumas carências têm sido encontradas nas soluções utilizadas em sua arquitetura no nível de representação de informações.

Os problemas surgem uma vez que novos e melhores serviços sugerem aplicações mais robustas, inteligentes e independentes da interferência humana. Como a viabilidade de serviços mais especializados e completamente automatizados depende diretamente da riqueza de detalhes utilizada na descrição da informação manipulada, podemos considerar que a inteligência das aplicações depende do entendimento semântico da fonte de informações manipulada. Sendo assim, a priori, as limitações se concentram no nível de representação de informação, ou seja, nas linguagens utilizadas para representar os documentos na *Web*, HTML e XML, cujo objetivo principal é oferecer somente um entendimento sintático da informação descrita em cada documento, através, basicamente, da padronização da estrutura dessas informações.

Essas necessidades impulsionaram o surgimento de um novo nível de informação na *Web* para suportar o entendimento semântico das informações. Essa nova visão da *Web* tem sido chamada de *Web Semântica (Semantic Web)* e é defendida pelo *World Wide Web Consortium (W3C)* e por diversos pesquisadores, principalmente por Tim Berners-Lee, inventor da *Web Semântica*.

A idéia da *Web Semântica* é fazer com que sejam adicionadas, à descrição de informações, outras informações que sejam capazes de fornecer o entendimento do conteúdo das informações de um documento. Portanto, os documentos na *Web* possuiriam, além das informações que descrevem a estrutura sintática do seu conteúdo, outras informações que dariam o entendimento semântico desse conteúdo.

A descrição do conteúdo é feita através da utilização do conceito de metadados, que significa dados sobre dados, e se aplica à visão da *Web Semântica*: informações utilizadas para descrever o conteúdo de outras informações.

Com o aparecimento desse novo nível na arquitetura da *Web* e acompanhando todo o princípio de padronização para assegurar e viabilizar a comunicação, surgiram as seguintes questões: como esses metadados seriam representados? e será que as linguagens de representação de informações já utilizadas poderiam ser adaptadas para atender tal tarefa?



Apesar do conceito metadados já ser abordado e tratado nas linguagens HTML e XML, percebeu-se que o mesmo era feito de forma superficial e que não atenderia satisfatoriamente a descrição das informações com uma riqueza de detalhes suficiente para oferecer o seu entendimento semântico. Então, a W3C criou uma nova linguagem específica pra representação de metadados na *Web*: RDF (*Resource Framework Description*) [W3C – RDF].

Apesar de existirem outras propostas para formalizar a representação dos metadados, o RDF permite uma maneira mais flexível para representar os metadados quando comparada com as propostas que oferecem *tags* pré-definidas. Com ele, poderemos definir todos os metadados que forem necessários para descrever qualquer documento na *Web* de forma estruturada [Benjamins et al. 2000].

Como a sintaxe da linguagem RDF é feita em XML, que já é um padrão adotado por diversos desenvolvedores na *Web*, ele tem sido adotado por diversas aplicações utilizadas para gerar bases de informações com suporte semântico e que serão, posteriormente, manipuladas por outras aplicações. Devido a sua larga aceitação, o RDF é um forte candidato para torna-se um padrão na sua função.

Com o RDF, é possível padronizarmos a representação dos metadados na *Web*. Apesar de ser de suma importância que os metadados sejam representados de forma padronizada, somente isso não é suficiente para atender o objetivo principal da *Web* Semântica que é oferecer serviços completamente independentes da interferência humana através do entendimento semântico das informações disponíveis na *Web*.

Por exemplo, pensemos numa peça de automóvel que é disponibilizada por diversos serviços de fabricantes e utilizada por vários serviços de montadoras de veículos. Através da utilização do RDF, poderíamos padronizar que, ao descrever o seu produto, cada serviço de fabricantes dessa peça deveria associar ao seu nome a sua descrição, mas como ficaria o conteúdo dessa descrição? Cada fabricante descreveria de forma independente os seus produtos oferecidos? Certamente, não. Quando pensamos em integração de serviços de uma forma completamente automatizada, é indispensável que haja um entendimento semântico comum sobre as informações manipuladas pelos mesmos.

Portanto, não basta que tenhamos a padronização da descrição dos metadados, é necessário também que existam fontes comum de informações, funcionando como uma espécie de vocabulário de dados, para diferentes domínios de aplicações e serviços e que elas ofereçam o suporte semântico satisfatório para prover serviços cada vez mais inteligentes na *Web*. Para tanto, surgem as Ontologias.

Ontologias fornecem um entendimento semântico comum de tópicos que podem ser utilizados tanto na comunicação entre pessoas como entre aplicações de sistemas [Decker et al. – 2000]. Do ponto de vista de Banco de Dados, uma ontologia pode ser definida como: “uma especificação parcial de um domínio ou meta-domínio, descrevendo entidades, relações entre eles e regras de integridade” [Mello-Heuser 2000].

No contexto da *Web Semântica*, as ontologias são utilizadas para descrever conceitos-chave de um dado domínio de conhecimento, bem como propriedades que caracterizam esse conceito e possíveis relacionamentos semânticos entre os mesmos.

Cada ontologia agregará uma larga parte de domínio de conhecimento para cada área em particular e uma das principais motivações para sua construção é a possibilidade de compartilhar e reutilizar conhecimento, definidos de uma forma genérica, entre comunidades e aplicações. Para que as ontologias possam desempenhar o seu papel de integrar semântica à *Web*, faz-se necessário que cada documento disponibilizado na rede possua metadados descritos sob a padronização de alguma ontologia (uma ontologia pode ser representada como uma hierarquia de conceitos [Benjamins et al. 2000]).

É bastante importante que não esqueçamos o quão complicado é pensarmos em padronizar qualquer coisa dentro de um ambiente tão heterogêneo quanto é o ambiente *Web* e que quando falamos sobre um entendimento semântico comum não nos referimos a uma único domínio para atender todo esse vasto universo de informações. Por isso, a proposta das ontologias está associada ao conceito de representação de domínios de conhecimento, onde existiriam ontologias específicas para cada área de conhecimento, diminuindo a complexidade da representação, e elas englobariam as informações necessárias para descrever semanticamente cada domínio de forma independente, mas podendo ser integradas de acordo com a necessidade.

Os domínios de conhecimento especificados numa ontologia, quando aplicados à recuperação de informação, terão a finalidade básica de servir como esquemas conceituais que darão suporte semântico a consultas. Uma grande vantagem desse contexto é que a ontologia provê uma interpretação semântica unificada para diferentes representações de dados semi-estruturados referentes a um mesmo domínio [Mello et al. 2000].

Ao passo que a eficiência das ontologias têm sido fortemente defendida pela comunidade científica, existem várias ferramentas para representar e manipular ontologias na *Web*.

Essas ferramentas fornecem muitas funcionalidades, principalmente em suas interfaces que são ricas em flexibilidades para representar ontologias. Mesmo assim, no início do nosso projeto, percebemos que algumas funcionalidades e características importantes para representação e manipulação de ontologias ainda não eram abordadas por tais ferramentas. A partir dessas carências, o objetivo desta dissertação é a concepção do OntoEditor que consiste num editor para manipular ontologias na *Web*.

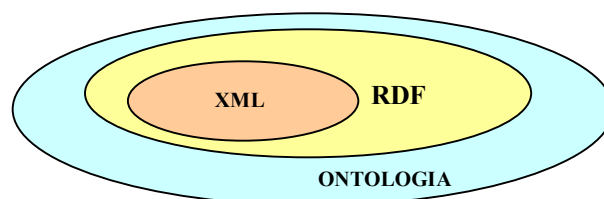
A principal característica do OntoEditor é ser um editor de ontologia para plataforma *Web* com integração a um banco de dados. Ao proporcionarmos essa integração, visamos atingir a viabilização da construção de um repositório de dados funcional e rico em informações semânticas. Através dessa ferramenta, os usuários podem descrever suas ontologias através de uma interface visual objetiva e de fácil utilização, armazená-las num banco de dados relacional e manipulá-las posteriormente de acordo com as suas necessidades.

Um outro diferencial do OntoEditor, em relação às ferramentas disponibilizadas na *Web* com o mesmo objetivo, é que na sua interface as ontologias são criadas na forma de grafos e não de árvores. Uma vez que a árvore é um tipo especial de grafo onde cada elemento possui um único ancestral, percebemos que essa característica limitava a representação semântica de conhecimento em várias situações do mundo real, onde necessitávamos descrever elementos que agregavam características de um ou vários ancestrais,. Logo, ao escolhermos a representação na forma de grafos simples para interface do OntoEditor, tais limitações foram suprimidas.

Após definir suas ontologias graficamente através do OntoEditor, o usuário poderá armazenar sua ontologia num banco de dados. Para tanto, o OntoEditor disponibiliza uma funcionalidade que traduz a ontologia representada graficamente para uma ontologia representada por um documento RDF, obedecendo toda padronização W3C. Posteriormente, quando o usuário necessitar manipular sua ontologia de forma gráfica novamente, o OntoEditor fornece o processo inverso e transforma o documento RDF em uma representação gráfica. Todo esse processo é transparente para o usuário que irá abstrair todas as complexidades para definir uma ontologia na linguagem RDF e ao mesmo tempo terá suas ontologias armazenadas numa linguagem que tem sido fortemente utilizada para representar metadados na *Web*, o RDF.

As ontologias são armazenadas num banco de dados relacional e além do documento RDF, que é a ontologia propriamente dita, o esquema possui informações da ontologia tipo nome, autor, domínio etc.

A estrutura semântica utilizada é ilustrada na FIGURA 1.1 onde ilustramos que para o OntoEditor, uma ontologia será descrita na linguagem RDF.. Portanto, temos uma especificação de alguma contextualização do mundo real (Ontologia), descrita sob uma forma padronizada de representação de informações/metadados (RDF), cuja sintaxe é descrita sob uma linguagem específica para definir esse tipo de informação no ambiente *Web* (XML).



**FIGURA 1.1 Ilustração da Estrutura Semântica do campo RDF**

O OntoEditor oferece um menu de consultas onde os seus usuários podem realizar alguns tipos de consultas:

- consultas por propriedades das ontologias – onde o usuário pode procurar por ontologias do autor X e do domínio Y; e
- consultas por conteúdo – onde o usuário poderá consultar por informações que estejam no conteúdo na ontologia, ou seja, dentro do documento RDF.

Com o OntoEditor, os usuários também podem exportar e importar suas ontologias de/para arquivos textos que podem ser armazenados de forma local na máquina que estiver acessando a ferramenta. O arquivo texto criado irá conter um documento RDF.

Ao decorrer dos próximos capítulos, serão enfatizadas e detalhadas todas as principais informações que nos induziram a idealização do OntoEditor, fundamentando a sua construção, bem como todas as características envolvidas desde o projeto à implementação dessa ferramenta.

O restante dessa dissertação está estruturado da seguinte forma: o capítulo II aborda vários temas que fundamentam teoricamente o nosso trabalho, o capítulo III apresenta todos os aspectos ligados ao projeto do OntoEditor, o capítulo IV descreve as características de implementação utilizadas para viabilizar o projeto e, finalmente, o capítulo V, discorre sobre qual foi a nossa proposta para esse trabalho e, dentro desse contexto, além de re-enfatizarmos as funcionalidades do OntoEditor, citamos direcionamentos para trabalhos futuros que possam ser realizados nessa ferramenta, buscando o seu aprimoramento.

## Capítulo II - FUNDAMENTAÇÃO TEÓRICA

Cada vez mais, a *WWW* vem se firmando como um paradigma na comunicação mundial e é inevitável que, para acompanhar toda a magnitude do seu crescimento, novas tecnologias tenham que ser adicionadas a ela constantemente. Novos e melhores serviços, completamente automatizados, têm que ser oferecidos e a nova geração da *Web* deve disponibilizar informações com características que expressem o significado das mesmas, permitindo, assim, o seu entendimento diretamente por máquinas (*machine-processable information*), independentemente da interferência humana [Decker et al. – 2000].

Essa é a visão da nova geração da *Web*: *A Web Semântica*, defendida pelo *World Wide Web Consortium* (W3C) e por Tim Berners-Lee, inventor da *WWW*, buscando um melhoramento das tecnologias oferecidas.

A *Web Semântica* induz uma nova arquitetura para a *Web*, onde serão introduzidas camadas para dar um suporte semântico às informações. Os principais aspectos envolvidos nesse contexto, como formas de representações das informações e padronizações, dentre outros, serão abordados nesse capítulo.

### 2.1. Dados na *Web*

Na *Web*, temos uma quantidade cada vez maior de dados disponíveis eletronicamente para os usuários, gerenciada através de uma diversidade crescente de modelos de dados e de mecanismos de acesso. Um tópico de estudo em crescimento nesta área diz respeito ao uso de dados do tipo semi-estruturados, cujos modelos podem ser aplicados para a representação e integração de dados na *Web* ou mesmo para proporcionar uma integração relativamente natural para dados oriundos de modelos distintos [Hull 1997], como o relacional e o de objetos [Silva-Heuser - 1999].

Como toda informação possui uma estrutura, na *Web*, essa estrutura é bastante peculiar: temos um esquema de dados onde as informações são descritas juntamente com o seu esquema. Esses dados são chamados dados semi-estruturados. Dados semi-estruturados são também chamados de dados sem esquema ("*schemaless*"), ou dados auto-descritivos ("*self-describing*") [Abiteboul et al. - 2000].

Normalmente, quando desejamos armazenar uma fonte de dados, descrevemos primeiramente o esquema de dados. Por exemplo, no modelo relacional através de criação de tabelas no banco de dados, e posteriormente instâncias dessa estrutura de dados são atribuídas a essa tabela povoando-a. Já com os dados semi-estruturados acontece uma situação diferente: os dados são descritos associados a uma sintaxe simples que representa uma estrutura para esses dados. Nós temos pares de dados (*label-value pairs*) compostos por descrição e valor das informações [Abiteboul et al. - 2000]. Ilustramos um exemplo de dados semi-estruturados na FIGURA 2.1. Nesse exemplo, temos informações referentes ao esquema de dados (nome, idade e email) descritas ao mesmo tempo com instâncias desse esquema ( Carlos Santana, 42 e [Carlos@xxx.com.br](mailto:Carlos@xxx.com.br) , respectivamente).

```
{nome: "Carlos Santana", idade:42, email: "carlos@xxx.com.br"}
```

**FIGURA 2.1 Exemplo de Dados Semi-Estruturados**

A priori, qualquer informação na *Web* está representada sob a forma de texto sustentada sintaticamente por alguma linguagem de descrição. Essas linguagens são conhecidas como linguagens de marcação e elas descrevem a estrutura do texto dentro do documento. Regras explícitas determinam como será toda a estrutura do documento, além das suas características de formatação [Pitts – 2000]. Logo, podemos considerar que as linguagens de marcação de que dispomos garantem, aos documentos na *Web*, uma representação padronizada das características de sintaxe dos seus respectivos conteúdos.

Existem várias linguagens de marcação e dentre elas poderemos ressaltar as mais genéricas, como segue.

O SGML (*Standard Generalized Markup*) é uma linguagem de marcação que oferece um amplo conjunto de regras padronizado e genérico, que através das marcações descrevem a estrutura de um documento. A partir do SGML, podem ser especificadas outras linguagens de marcação e elas são chamadas de Aplicações SGML.

Apesar de oferecer uma enorme quantidade de recursos, o SGML é bastante complexo para ser interpretado pelos *browsers* na *Web* e isso restringe sua utilização por parte dos desenvolvedores, que preferem optar por alternativas mais simples. Mesmo assim, ele tem sido muito utilizado por diversas áreas durante vários anos, tornando-se, inclusive, um padrão ISO (*International Organization for Standardization*) [Moultis, Kirk – 2000].

O HTML (*Hypertext Markup Language*) é uma aplicação SGML e oferece uma quantidade de recursos consideravelmente menor que o SGML. Esses recursos se concentram em descrever o formato em que as informações na *Web* deverão ser exibidas, possibilitando a publicação de documentos na Internet na forma de páginas *Web*.

A linguagem HTML é simples de ser utilizada e rápida para ser implementada. Com isso, temos um conjunto limitado de recursos, de fácil utilização e interpretação, tornando o HTML uma linguagem de marcação de ampla aceitação, sendo bastante encontrada na descrição dos textos contido nos documentos na *Web*. As grandes limitações do HTML são conjunto de tags limitado e mistura de apresentação com conteúdo da informação.

O XML (*Extensible Markup Language*) é um subconjunto do SGML com recursos direcionados a publicação de informações no ambiente *Web*. Comparado ao HTML, o XML oferece uma gama de recursos muito mais ampla e expressiva para descrever o formato das informações na *Web*. Com XML o usuário pode criar suas próprias tags e detalhes de apresentação do conteúdo não são expressos juntos com a representação deste conteúdo.

O XML flexibiliza a descrição de informações na *Web*, uma vez que disponibiliza elementos que possibilitam descrever diversas situações do mundo real. Para isso, o XML permite um número ilimitado de *tags* que podem ser utilizado de forma indiscriminada, ao contrário do HTML que utiliza suas *tags* para associar somente características de formatação às informações. Logo, várias aplicações XML têm sido desenvolvidas e elas são sustentadas



em esquemas que definem a estrutura e conteúdo das informações descritas em XML, possibilitando a interpretação dessas informações.

Podemos encontrar esquemas XML descritos em sintaxe XML, ou não, como é o caso do DTD (*Document Type Description*), que é descrito na sintaxe EBNF (*Extended Backus-Naur Form*). O DTD descreve o conteúdo sintático e a estrutura de documentos XML, sendo que para cada documento existe um DTD respectivo. Portanto, através dele, temos informações que se concentram em definir a forma como as informações contidas nos documentos serão apresentadas, ou seja, características de sintaxe das informações definidas através de uma estrutura formal.

A origem da criação e utilização do DTD está associada a linguagens de marcação anteriores a *World Wide Web*, como podemos citar o SGML. Atualmente, o contexto da sua aplicação é bastante diferente do que tínhamos quando ele foi concebido, sendo que o DTD apresenta várias limitações para sustentar um enorme, e cada vez mais crescente, intercâmbio de informações escritas em XML através da *Web*. Como limitações do DTD podemos citar: tipos de dados, herança e integração com espaço de nomes XML [Ahmed et al. – 2001].

Para solucionar as limitações deixadas pelo DTD e oferecer maiores flexibilidades, outros esquemas XML têm sido definidos em sintaxe XML e, dentre eles, podemos citar o esquema XSD. Maiores informações podem ser encontradas em <http://www.w3.org/TR/xmlchema>.

Existem várias maneiras de se visualizar documentos XML. Se o *browser* entende o padrão XML, o documento pode ser enviado diretamente para ele. Caso contrário, pode-se usar uma folha de estilo para transformar o arquivo XML em algo que o *browser* entenda.

Com o XML, podemos criar dados estruturados e isso oferece flexibilidades aos desenvolvedores *Web*. Informações com uma estrutura formal na *Web* podem ser interpretadas pelo computador e com isso vários novos serviços podem ser disponibilizados. Com o HTML isso não seria possível, visto que os dados HTML não são estruturados.

O intercâmbio de dados na *Web* está sustentado sob vários padrões e eles garantem interoperabilidade em muitos níveis: o protocolo TCP/IP assegura o transporte de bits na rede, similarmente, o HTTP é um padrão para identificar endereços na *Web* e o HTML fornece uma

forma padronizada de guardar e representar documentos na forma de textos [Decker et al. – 2000].

Várias aplicações têm sido desenvolvidas para o ambiente *Web*, devendo estar de acordo e integrada com toda essa infraestrutura que é oferecida. Entretanto, até então, o entendimento das informações, disponibilizado por muitos serviços, é designado para as pessoas que as utilizam (*direct human processing*).

## 2.2. A *Web Semântica (Semantic Web)*

A *Web Semântica* se propõe a adicionar um entendimento semântico às informações disponibilizadas na *Web*. Como já foi dito, a idéia principal é disponibilizar informações associadas a outras informações que expressem os seus significados.

Uma vez que tenhamos o entendimento semântico das informações, as aplicações *Web* poderão acessar bases de dados ricas em características que lhes fornecerão a inteligência suficiente para automatizar vários serviços [Decker et al. – 2000]. Além disso, a *Web Semântica* possibilitará melhores resultados, para serviços que já são disponibilizados, como é o caso dos serviços de busca na *Web*.

Com a *Web Semântica*, poderemos pensar também em mapeamentos de dados. Esse mapeamento consiste em criar correlações semânticas entre os dados. Como a *Web* é uma enorme e heterogênea base de dados, encontraremos nela várias e diferentes descrições para as informações. Muitas vezes, encontraremos informações similares semanticamente, mas descritas sintaticamente de forma bastante diferente.

Por exemplo, numa página de uma universidade A poderíamos ter artigos e cada artigo traria, além do seu conteúdo, uma informação chamada autor, acompanhada do seu valor, nesse caso, consideremos como autor igual a Tereza Ramos. Pensemos agora numa universidade B que possuísse a mesma situação com uma simples diferença: a informação adicional seria chamada de criador, acompanhada do seu valor. Consideremos que as duas universidades possuem artigos da mesma pessoa, Tereza Ramos, então como uma aplicação,

sem a interferência humana, poderia entender que o valor da informação autor e criador, utilizadas em situações distintas, expressam na realidade o mesmo significado?

Uma outra situação semelhante acontece com os serviços disponibilizados no comércio eletrônico. Pensemos em duas lojas virtuais na *Web*, cada uma com o seu catálogo de produtos. Se descrições diferentes para o mesmo produto fossem utilizadas indistintamente pelas lojas, como poderíamos pensar numa intercomunicação eficiente entre essas lojas?

O mapeamento de informação ajudaria a resolver tais situações logo que poderíamos ter relacionamentos semânticos entre fontes de dados diferentes. Esta é uma característica bastante importante, pois a falta de padrões na descrição das informações na *Web* é uma séria e constante dificuldade para que tenhamos uma intercomunicação entre diferentes fontes de dados e o mapeamento pode permitir relacionamentos que flexibilizem satisfatoriamente essa comunicação [Ahmed et al. – 2001].

### **2.2.1. O Conceito de Metadado**

Utilizando uma definição bastante simplória poderíamos definir metadado como dado sobre dado, ou seja, uma informação sobre outra informação. A solução que vem sendo adotada para alcançarmos a *Web* Semântica é baseada na utilização de metadados para descrever os dados contidos na *Web*. Para que possamos expressar o significado de algum dado, precisamos adicionar ao mesmo outros dados que caracterizem-no. Assim, poderemos ter um entendimento semântico desse dado, visto que teremos outras informações que associam o significado àquele dado.

Logo, percebemos que na realidade, tudo pode ser considerado como dados, afinal um metadado é um dado sobre outro dado. O que realmente faz a diferença é o significado do conteúdo do metadado, sempre que ele estiver relacionado a algum outro dado.

Para exemplificar, pensemos na descrição de um artigo publicado na *Web*. Esse artigo possuirá uma URL, ou seja um endereço através do qual poderíamos acessá-lo. Tal artigo poderia também possuir outras informações tipo: quem o escreveu, o seu título, área de abrangência etc. Essas informações adicionais representariam os metadados e, à medida que,

temos fontes de informações mais completas, cada vez mais descritivas, elas poderiam ser utilizadas sob diversas formas para disponibilizar serviços cada vez mais expressivos na *Web*.

### 2.2.2. Dublin Core

O Dublin Core (DC) pode ser considerado basicamente um conjunto de termos que visa padronizar a representação das descrições de recursos na *Web*. Ele é essencialmente um esquema de dados compreendido por quinze elementos de metadados padronizados e criados pelos participantes do OCLC/NCSA Metadata Workshop, realizado em março de 1995 e que teve a participação de especialistas de campos como biblioteconomia, computação e codificação de texto [Moultis, Kirk – 2000].

A princípio o DC não foi definido sob uma sintaxe específica, mas já existe sua especificação completa em XML e, nesse caso, ele possui uma sintaxe específica. A sua padronização possibilita que tenhamos um comum entendimento semântico dos metadados associados aos documentos da *Web*. Consideremos vários documentos disponibilizados na Web. Uma característica comum a todos esses documentos é que tenham um título. Para descrever essa característica o Dublin Core criou a propriedade *Title*. Se cada vez que houvesse a necessidade de associar o metadado título ou qualquer outra palavra com o mesmo significado aos documentos na *Web*, utilizássemos a propriedade *Title*, conseguiríamos uma padronização que facilitaria o entendimento semântico dos serviços *Web* em relação aos metadados encontrados, evitando assim várias situações de ambigüidade.

O Dublin Core já é utilizado em vários documentos HTML, dentro das *tags* <META>, como mostrado no exemplo da FIGURA 2.2, mas não deve ser considerado exclusividade de documentos HTML [Ahmed et al. – 2001]. Nesse exemplo, o elemento DC.Creator do vocabulário Dublin Core é utilizado para descrever o criador ou autor, Craig Larman, do documento Utilizando UML e Padrões.

```
<HTML>
  <HEAD>
    <TITLE> Utilizando UML e Padrões </TITLE>
    <META NAME="DC.Creator" CONTENT="Craig Larman">
  </HEAD>
</HTML>
```

**FIGURA 2.2 Exemplo ilustrando a utilização de elementos do Dublin Core em HTML**

O Dublin Core nos proporcionou uma experiência de padronização de metadados. Tal padronização nos sugere uma idéia semântica bastante importante que é a utilização de vocabulários. À medida que tais informações são associadas aos documentos sob uma forma descritiva, porém padronizada, diversas aplicações podem utilizá-las, interagindo e oferecendo serviços integrados, porém cada vez mais independentes de interferências manuais, por estarem sustentados em informações com um comum entendimento semântico.

### 2.3. O papel das Ontologias

A *Web* Semântica sugere que sejam disponibilizadas informações na *Web* com características descritivas (metadados) que expressem um entendimento semântico de cada fonte de dados nela encontrada .

Já discutimos anteriormente algumas das utilidades decorrentes do uso dessa camada semântica. Todavia, a falta de padronização para definição dos metadados torna sua utilização, conseqüentemente, num sério problema para que tenhamos um eficiente intercâmbio de informações na *Web*.

Muitos desses problemas de definição podem ser resolvidos com a utilização de um vocabulário comum, englobando conjuntos de termos com suas definições. Como proposta para possibilitar um processamento, compartilhamento e reutilização dessas informações semânticas na *Web*, surgem as Ontologias [Decker et al. – 2000].

Na área de sistemas de informação, o conceito de ontologias foi inicialmente utilizado na área de Inteligência Artificial (IA) para descrever conceitos e relacionamentos utilizados

(conhecimento compartilhado). Uma das definições mais simples e bastante usada de ontologias aplicada ao contexto da Ciência da Computação é a de *Tom Grubber*, pesquisador em representação e compartilhamento de conhecimento. Para *Grubber*, uma ontologia é uma especificação explícita de uma conceitualização [Grubber – 1993].

Definido como uma conceitualização formal e comum de um particular domínio de conhecimento, as ontologias fornecem um entendimento semântico comum de tópicos que podem ser utilizados tanto na comunicação entre pessoas como entre aplicações de sistemas [Decker et al. – 2000]. Com as ontologias, poderemos especificar domínios de conhecimento, através da definição de conceitos e suas características. Essa fonte de informação, representada pelas ontologias, deverá ser cuidadosamente definida e, evidentemente, surge a necessidade da intervenção humana para alimentá-la, uma vez que ela representará uma interpretação semântica unificada de um domínio de conhecimento [Mello et al. – 2000].

Com o uso de ontologias, é possível representar informações que refletem um entendimento semântico de diversas situações do mundo real. Entretanto, temos que considerar que é uma tarefa bastante complicada representar tais situações com toda sua riqueza de detalhes. Para simplificar a definição de ontologias, tem sido utilizado o conceito de representação de domínios, onde é representada uma parte do mundo, restringindo, assim, a diversidade de informações e, ao mesmo tempo, concentrando a definição em cada domínio de conhecimento para permitir uma representação cada vez mais rica do mesmo.

As ontologias se propõem a capturar domínios de conhecimento de forma genérica, para fornecer um entendimento semântico desses domínios que poderá ser utilizado e compartilhado por diversas comunidades e aplicações [Staab et al. – 2000]. Logo, é importante que as ontologias sejam definidas preferencialmente por especialistas de um dado domínio, visto que não há ninguém mais indicado para definir os conceitos-chave de uma determinada área de conhecimento do que pesquisadores/estudiosos desta área.

Uma das principais motivações em se definir ontologias é a de que com isso poderemos ter uma base de informações que pode ser compartilhada e reutilizada por diversas aplicações [Guarino – 1996].

Ao definirmos ontologias poderemos definir acordos entre usuários de informação e os fornecedores de informação para padronização da utilização dos dados na Web, diminuindo os problemas de consistência no intercâmbio de dados nesse ambiente.

Esses acordos são chamados Compromissos Ontológicos (*Ontological Commitments*) e permitem que somente determinados significados de um domínio sejam capturados numa dada especificação. O intercâmbio de dados é padronizado e será baseado no modelo de especificação que foi criado.

Sendo assim, ao definir o compartilhamento da utilização de uma dada ontologia, usuários e aplicações associarão regras comuns de padronização aos dados e, conseqüentemente, assegurarão maior confiabilidade às informações disponibilizadas e recebidas.

Os compromissos ontológicos podem definir diferentes graus de detalhe e isso dependerá dos propósitos de cada ontologia, ou seja, a delimitação do escopo da representação de cada domínio de conhecimento dependerá dos propósitos definidos pelos usuários para especificar essa ontologia.

Quando falamos em definir ontologias para representar domínios de conhecimento, estamos considerando a possibilidade de termos várias ontologias representando diversas fontes de conhecimento, definidas, utilizadas e integradas de acordo com a necessidade da situação. Como não há um consenso que defina um escopo ao qual cada ontologia deverá atender, poderíamos pensar na definição de uma ontologia universal, proporcionando um vocabulário global. Entretanto, utilizando uma ontologia universal, seria mais complicado garantir um entendimento semântico unificado numa proporção tão ampla. É evidente que vários problemas decorreriam desde a definição desse vocabulário global, que deveria ser consensual para todos, até a imposição de sua utilização e conseqüentemente sua manutenção. Portanto, o uso de ontologias específicas para representar domínios de conhecimento é mais recomendado, dada a possibilidade de se estabelecer um consenso, entre os diversos membros que compõem a comunidade desta área, ser mais factível.

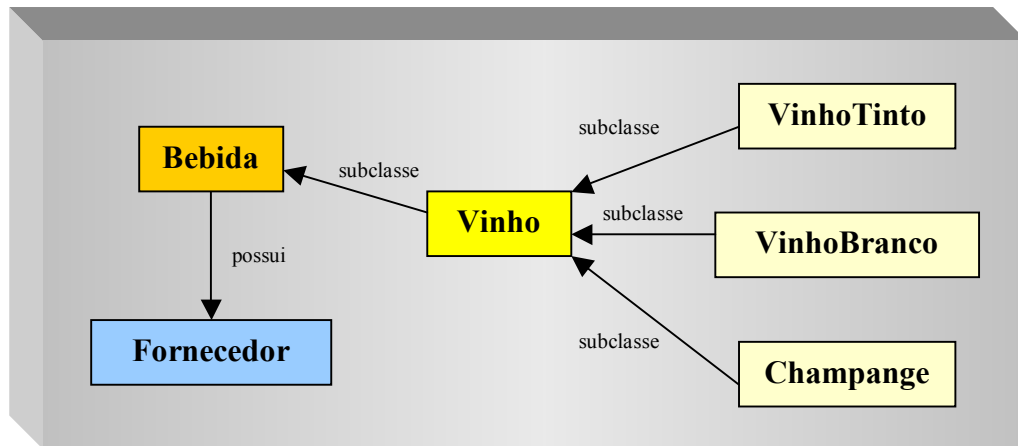
Em termos práticos, uma ontologia consiste basicamente na definição de alguns itens:

- Conceitos que representam tópicos importantes na definição de um dado domínio de conhecimento;
- Definição de algumas características relevantes para esses tópicos; e
- Definição de relacionamentos entre esses conceitos, além de possibilitar uma organização hierárquica desses conceitos, sempre que for necessário.

A FIGURA 2.3 ilustra uma possível definição para uma ontologia. Ressaltamos que tal exemplo é meramente ilustrativo, uma vez que ele mostra uma visão insuficiente para representação real e em detalhes de um domínio de conhecimento.

Nesse caso, o domínio de conhecimento abordado é Bebidas e, baseado nele, definimos alguns termos que representam conceitos-chave para criarmos um entendimento semântico para esse contexto. Foram definidos os termos: Bebida, Vinho, VinhoTinto, VinhoBranco, Chanpangne e Fornecedor. Através dos relacionamentos entre os termos, podemos perceber que Vinho é um tipo de Bebida e que ele possui algumas derivações como: VinhoTinto, VinhoBranco e Chanpangne. Vimos também que toda Bebida possui um Fornecedor.





**FIGURA 2.3** Exemplo ilustrando a definição de uma ontologia para Vinhos.

Segundo Guarino [Guarino – 199?], as ontologias podem ser classificadas da seguinte forma:

- Ontologias de nível superior – descrevem conceitos gerais como espaço, tempo, objeto, assunto, ação e/ou metadados como entidades, relações e atributos;
- Ontologias de domínio e de tarefa – podem ser especializações da ontologia de nível superior, descrevendo, respectivamente, um vocabulário para um universo de discurso (medicina ou automóveis) e para uma tarefa genérica (diagnóstico ou venda);
- Ontologias de aplicação – essas dependem tanto de um domínio quanto de uma tarefa particular, podendo ser uma especialização de ambas. Corresponde a regras impostas por conceitos de domínio quando executam uma certa tarefa, como por exemplo, substituição de uma unidade sobressalente de um automóvel.

As ontologias podem ser utilizadas em vários serviços na *Web*. Um exemplo da sua utilização é em *sites* de comércio eletrônico (*e-commerce*).

As ontologias são utilizadas para fornecer uma base de informações comum e padronizada, englobando os conceitos-chave utilizados pelos serviços requisitados por cada contexto. No caso do comércio eletrônico, essas informações são utilizadas para integrar e unificar as definições de produtos oferecidos por várias lojas de forma padronizada, além de reutilizar diferentes descrições para um mesmo produto oferecido por diferentes lojas virtuais

(isso é possível pois uma vez que as ontologias realizam relacionamentos entre conceitos, poderemos dispor de conceitos descritos de forma diferente sintaticamente mas similares semanticamente).

Assim vários serviços de compra e venda podem ser disponibilizados de forma completamente automatizada. Logo que os produtos oferecidos pelas lojas estejam descritos sob uma forma semanticamente comum entre elas, tornar-se-á possível uma comunicação de cliente vendedor entre máquinas diretamente, sem a direta interferência humana.

Uma outra aplicação, onde as ontologias vêm sendo bastante utilizadas, está ligada à área de banco de dados e recuperação de informação, como suporte à interoperabilidade de fontes de dados distribuídas e heterogêneas [Arruda et al. – 2002]. As ontologias têm sido aplicadas principalmente em banco de dados heterogêneos e *Data WareHouses* como modelos conceituais globais, resultantes de uma concordância na definição de entidades e relacionamentos.

Dentro da área de banco de dados, também podemos aplicar as ontologias a sistemas de recuperação de informação e esse é o nosso foco de estudo.

As ontologias podem ser utilizadas nas máquinas de busca (*search engines*) da *Web* servindo como um esquema conceitual que dá suporte semântico às consultas [Mello et al. – 2000]. Ao utilizar ontologias, as máquinas de busca podem fazer suas consultas através de suas palavras-chave e como resposta a essa consulta poderemos ter, além das páginas que contêm a palavra-chave informada, outras páginas contendo informação sintaticamente diferente da palavra-chave, ou seja, sinônimos que são escritos de outra maneira, mas que expressam o mesmo significado semântico [Decker et al. – 2000]. Por exemplo, numa consulta com palavra-chave “carro” poderia retornar páginas que contenham a palavra “automóvel”.

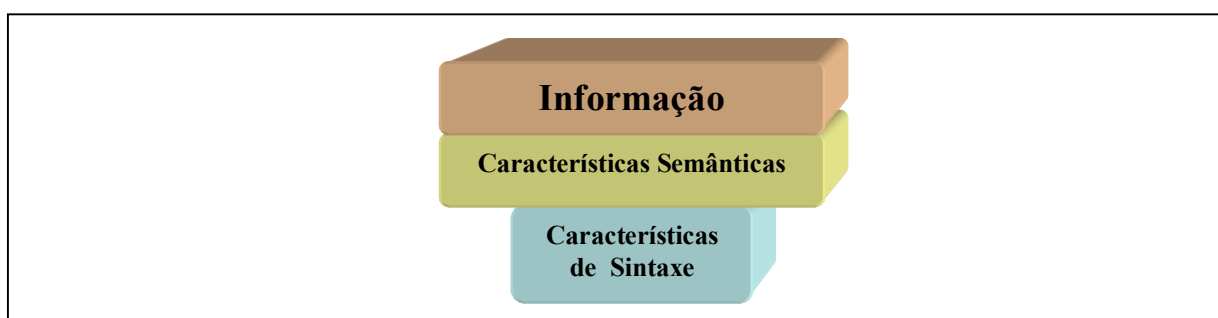
As ontologias também podem ser utilizadas para guiar processos de extração de dados semi-estruturados através da manutenção de informações que auxiliem na identificação de atributos de conceitos e relações. Como no gerenciamento de dados semi-estruturados, a extração de dados faz parte do processamento de uma consulta, algumas aplicações, que provêm essa facilidade, utilizam ontologias como esquemas conceituais a partir do qual

consultas com caráter semântico podem ser formuladas. Nesse contexto, podemos destacar algumas aplicações como: Observer, SHOE e a Proposta de Embley [Mello et al. – 2000].

## 2.4. Linguagens de Representação de Metadados

À medida que nos direcionamos para uma *Web Semântica*, surgem novas necessidades, principalmente para as linguagens de marcação que dispomos. Como os documentos na *Web* precisam ser associados a metadados que expressarão o seu valor semântico, deveremos utilizar uma linguagem que esteja apta a descrever tais informações, além das características de sintaxe já utilizadas. Essa linguagem deve oferecer uma representação padronizada dos metadados para que possamos sustentar o intercâmbio dessas informações dentre as mais diversas aplicações na *Web*.

Como mostra a FIGURA 2.4, para atender as necessidades da *Web Semântica* precisamos de uma linguagem que através de uma sintaxe especial descreva informações de forma padronizada, associadas a metadados que expressarão o valor semântico de cada informação.



**FIGURA 2.4** Visão abstrata da estrutura das informações na *Web Semântica*

Atualmente, temos algumas linguagens desenvolvidas especialmente para suprir tais necessidades e que já vêm sendo bastante utilizadas por pesquisadores e desenvolvedores envolvidos com as tecnologias de representação de conhecimento e de metadados. Descrevêmo-las a seguir.

### 2.4.1. RDF (Resource Description Framework)

O RDF foi criado pela *World Wide Web Consortium* (W3C) para definir uma padronização da representação e do uso de metadados na *Web*. O propósito da W3C era fornecer meios para possibilitar a interoperabilidade entre aplicações na *Web* que trocassem informações que fossem entendidas diretamente pelas máquinas (*machine-understandable*). Assim, o RDF define um mecanismo para descrever recursos na *Web* de uma forma neutra, ou seja, sem descrever uma particular área de aplicação ou de domínio de conhecimento. A priori, RDF não define a semântica de nenhum domínio [W3C – RDF].

O RDF pode ser utilizado em vários tipos de aplicações como: máquinas de busca, bibliotecas digitais, aplicações para compartilhar e fazer intercâmbio de bases de conhecimento, aplicações ligadas ao comércio eletrônico, dentre outras [W3C – RDF].

A sintaxe do RDF é descrita em XML. O RDF se integra ao XML, usufruindo todas as suas solidificadas vantagens, que já foram citadas de forma resumida anteriormente, ampliando o formato da sua estrutura para possibilitar a representação de informações semânticas.

O formato XML foi idealizado para representar informações associadas como pares, onde temos, sob uma visão simplificada, uma estrutura baseada em termo e valor. Por exemplo, um autor (termo) possui um nome (valor) [Moultis, Kirk – 2000]. Já o RDF representa as informações em forma de triplas, formadas por termo + valor (metadados) + documento ao qual se refere [Hjelm – 2001]. Graças às flexibilidades do XML, podemos expressar semântica através da representação padronizada das triplas, escritas sob a estrutura sintática do XML.

O modelo de dados RDF é definido formalmente como segue [W3C – RDF]:

1. Existe um conjunto chamado Recursos (*Resource*).
2. Existe um conjunto chamado Literais (*Literals*).
3. Existe um subconjunto de Recursos chamado Propriedades (*Properties*). As Propriedades são conjuntos de propriedades. Cada propriedade representa uma característica ou um atributo ou um relacionamento que descreve um recurso.

4. Existe um conjunto chamado Sentenças (*Statements*) – cada Sentença é formada por:

{pred, suj, obj}, onde: pred é um predicado, ou seja uma propriedade (membro de Propriedades); suj é um sujeito que representa um recurso (membro de Recursos) e obj é um objeto que representa um recurso ou uma literal, que é um dado do tipo string (membro de Literal). Resumindo, o relacionamento entre um recurso e uma literal é chamado de sentença e ele conecta um sujeito a um objeto via um predicado.

Exemplo:

Sentença: Larman é o autor do livro Utilizando UML e Padrões.

Sujeito: livro Utilizando UML e Padrões.

Predicado: autor.

Objeto: Larman.

Como foi citado acima, o RDF descreve recursos na *Web* (*resource Web*). Os recursos representam o sujeito das sentenças RDF e eles podem ser páginas, serviços ou qualquer coisa na *Web* que possua um identificador. Foi adotado um padrão para definir os identificadores que é o *Uniform Resource Identifier* ou, simplesmente, URI. Uma URI é um conjunto de caracteres que representa um endereço único para cada recurso na *Web*, oferecendo um identificador único para cada um deles. A URI é de fundamental importância para o RDF pois é através dela que poderemos determinar o sujeito a que se refere uma dada descrição [Hjelm – 2001].

#### **2.4.1.1. Tipos de Representação das Sentenças RDF**

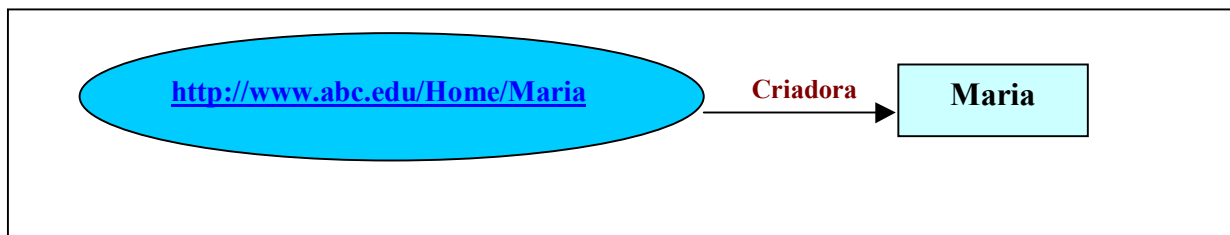
Os dados RDF são sentenças sobre os recursos e podem ser vistos sob três tipos de representação que se equivalem: como um grafo, como uma representação XML e como triplas.

Para pessoas, a Representação Gráfica é a mais fácil de ser entendida. O RDF fornece um modelo de dados baseado em grafos que consistem de nós (*nodes*) e arcos (*edges*). Os nós correspondem a objetos ou recursos (*resources*) e arcos a propriedades desses objetos. Os

*labels* nos nós e arcos são as **URIs** (*Uniform Resource Identifiers*). Quando um nó é uma *Literal*, ele é representado por retângulos. Exemplificando:

#### EXEMPLO 2.1

Consideremos que Maria é a criadora do recurso <http://www.abc.edu/Home/Maria/>. Graficamente poderíamos representar como segue:



**FIGURA 2.5 Sentença RDF representada num grafo**

A FIGURA 2.5 mostra como construir sentenças RDF graficamente. Em linhas gerais, cada grafo possui arcos direcionados e rotulados que associam duas informações: a primeira, representada pela elipse e rotulada por uma URI, indica o recurso que está sendo descrito (<http://www.abc.edu/Home/Maria/>); a segunda, representada por um retângulo, rotulada pelo nome Maria, indica o valor que a característica Criadora (metadado), rotulada no arco, deve assumir.

Apesar de termos utilizado, no EXEMPLO 2.1, um relacionamento que associa um recurso a um valor, o RDF também permite que façamos relacionamentos entre recursos.

Poderíamos representar essa mesma informação sob a forma de Triplas. As triplas são acessíveis para programas de aplicação que irão utilizá-las como entradas para suas operações. As triplas em RDF consistem de três partes do gráfico: o recurso (*resource*), a propriedade (*property*) e o valor da propriedade. Algumas constantes (*string values*) podem ser incluídas e elas são chamadas de literais. Logo, para a mesma sentença citada no EXEMPLO 2.1, teríamos:

<i>Sujeito (Resource) - suj</i>	<i>Predicado (Property)- pred</i>	<i>Objeto (Literal)- obj</i>
<a href="http://www.abc.edu/Home/Maria">Http://www.abc.edu/Home/Maria</a>	Criadora	Maria

**TABELA 2.1 Representação de uma sentença RDF na forma de Triplas**

Mesmo considerando que a representação gráfica é facilmente lida e entendida por pessoas, não podemos esquecer que ela é bastante complexa para ser entendida por computadores. Para que as sentenças RDF possam ser compreendidas pelas máquinas surge uma terceira forma de representação: poderíamos ter a mesma sentença escrita na sintaxe XML. Dessa forma, o RDF é utilizado para definir informações, possibilitando o seu intercâmbio entre máquinas, sem interferência humana direta, através de várias aplicações e serviços. Uma forma de representar o EXEMPLO 2.1 em RDF/XML seria:

```
<rdf:RDF
  xmlns:dc= "http://purl.org/metadata/dublin_core#">
  <rdf:Description about= "http://www.abc.edu/Home/Maria">
  <dc:Creator>Maria</dc:Creator>
  </rdf:Description>
</rdf:RDF>
```

**FIGURA 2.6 Exemplo de Sentença RDF escrita em sintaxe XML**

Na representação da FIGURA 2.6, utilizamos uma funcionalidade do XML bastante utilizada pelo RDF: o Namespace. O *namespace* é um apelido utilizado para identificar uma URI e, em XML, ele é representado através da propriedade xmlns. No nosso caso, utilizamos um *namespace* chamado dc que representa a URI da biblioteca que compreende os metadados do Dublin Core. Com isso, definimos características do recurso através de metadados padronizados, facilitando um comum entendimento e a intercomunicação dos mesmos [XML-  
Namespace].

Além disso, foram utilizados, nas tags XML, alguns elementos definidos pelo modelo RDF para permitir a representação das sentenças RDF em XML. Descreveremos, a seguir, elementos que integram o modelo RDF .

Todavia, vale lembrar que abordaremos os principais elementos do modelo e sintaxe RDF. Maiores detalhes da especificação RDF, cujo aprofundamento detalhado não faz parte do objetivo do nosso trabalho, poderão ser acessados no W3C, através do endereço <<http://www.w3.org/TR/REC-rdf-syntax/>>.

O elemento *Description* identifica os recursos que desejamos descrever nas sentenças RDF. Dentro desse elemento, poderemos descrever várias características (propriedades). Essas informações estarão associadas a um único objeto, ou seja, a um recurso identificado por uma URI e que esta sendo descrito. A URI é descrita como um atributo do elemento *Description*. Existem dois tipos de atributos: o *about* e o *ID* e eles são mutuamente exclusivos. Quando estamos descrevendo um outro recurso qualquer, deveremos utilizar o atributo *about*. Caso estejamos fazendo uma autodescrição de um recurso, deve ser utilizado o atributo *ID*.

#### 2.4.1.2 Sentenças sobre sentenças

Além de permitir a descrição de sentenças sobre recursos, o RDF possibilita que possamos definir sentenças sobre outras sentenças. A sentença é utilizada para descrever alguma informação sobre um dado recurso disponibilizado na *Web*. Utilizando esse mesmo raciocínio, poderíamos ter a necessidade de também descrever informações sobre sentenças e isso é tratado pelo RDF como sentenças sobre sentenças. Essas sentenças são consideradas sentenças de alto nível (*higher-order statements*).

#### EXEMPLO 2.2

Consideremos que Luiza disse que Maria é a criadora do recurso <http://www.abc.edu/Home/Maria/>.

No EXEMPLO 2.2, temos uma sentença que se refere a uma outra sentença. A sentença original é Maria é a criadora do recurso <http://www.abc.edu/Home/Maria/> e a sentença citada nesse exemplo descreve uma informação sobre ela.

Para representar a sentença do EXEMPLO 2.2 em RDF, é necessário:

1. Modelar a sentença origem, acrescentando um novo elemento, o TYPE, além dos já utilizados: SUBJECT, PREDICATE e OBJECT; e



2. Utilizar uma propriedade que represente o sentido da palavra “disse”, no contexto da sentença. Para tanto, podemos utilizar a propriedade *attributedTo*, cujo valor apropriado é “Luiza”.

Assim, temos:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:a="http://description.org/schema/">
  <rdf:Description>
    <rdf:subject resource="http://www.abc.edu/Home/Maria" />
    <rdf:predicate resource="http://description.org/schema/Creator" />
    <rdf:object>Maria</rdf:object>
    <rdf:type resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement" />
    <a:attributedTO>Luiza</a:attributedTO>
  </rdf:Description>
</rdf:RDF>
```

**FIGURA 2.7 Representação do Exemplo 2.2 em RDF**

O exemplo mostrado na FIGURA 2.7 foi copiado do site da W3C e adaptado para o EXEMPLO 2.2. O exemplo original pode ser encontrado em <http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/#higherorder>.

#### 2.4.1.3. Contêineres RDF

O RDF permite que possamos considerar um recurso como um conjunto de objetos e, para descrever sentenças que envolvem esses tipos de coleções de recursos, dispomos de três de três elementos: *Bag*, *Sequence* e *Alternative* [W3C – RDF].

O elemento *Bag* representa uma lista desordenada de recursos (URIs) ou de literais. São utilizados para declarar que uma propriedade tem múltiplo valor e que não importa a ordem em que os valores foram atribuídos. Valores duplicados são permitidos. O RDF ainda não define o tipo Set que seria um *Bag* que não aceita valores duplicados.

O elemento *Sequence* representa uma lista ordenada de recursos ou literais. Existe uma ordem dos valores que estão sendo atribuídos a propriedade. Valores duplicados são permitidos.

O elemento *Alternative* representa uma lista de recursos ou literais que representam os valores possíveis que podem ser atribuídos a uma certa propriedade. A propriedade poderá ser associada a um único valor da lista.

### EXEMPLO 2.3: Contêiner Bag

Nele descreveremos a seguinte sentença:

- Os alunos do curso 869 são Ana, Carla, Paulo e Gabriel.

Teremos o modelo RDF que segue:

```
<rdf:RDF>
  <rdf:Description about="http://universidade.edu/cursos/869">
    <a:alunos>
      <rdf:Bag>
        <rdf:li resource="http://universidade.edu/alunos/Ana"/>
        <rdf:li resource="http://universidade.edu/alunos/Carla"/>
        <rdf:li resource="http://universidade.edu/alunos/Paulo"/>
        <rdf:li resource="http://universidade.edu/alunos/Gabriel"/>
      </rdf:Bag>
    </a:alunos>
  </rdf:Description>
</rdf:RDF>
```

O curso é um recurso identificado como <http://universidade.edu/cursos/869>. As propriedades dos alunos são descritas dentro do elemento <a:alunos> </a:alunos>, utilizando o container Bag do RDF. Nesse tipo de container não importa a ordem das informações descritas.

### EXEMPLO 2.4: Contêiner Alternative

Nele descreveremos a seguinte sentença:

- Os artigos publicados pelo aluno Gabriel podem ser encontrados em [www.universidade.edu/alunos/Gabriel/artigos](http://www.universidade.edu/alunos/Gabriel/artigos) ou [www.universidade.edu/alunos/Gabriel/publicações](http://www.universidade.edu/alunos/Gabriel/publicações).

Teremos o modelo RDF que segue:

```
<rdf:RDF>
  <rdf:Description about=" http://universidade.edu/alunos/Gabriel ">
    <s:artigosPublicados>
      <rdf:Alt>
        <rdf:li resource=" www.universidade.edu/alunos/Gabriel/publicações"/>
        <rdf:li resource=" www.universidade.edu/alunos/Gabriel/artigos"/>
      </rdf:Alt>
    </s: artigosPublicados>
  </rdf:Description>
</rdf:RDF>
```

O aluno é um recurso identificado como <http://universidade.edu/alunos/Gabriel>. Os artigos publicados são considerados propriedades do aluno Gabriel e descritos dentro do elemento `<s:artigosPublicados>` `</s:artigosPublicados>`, utilizando o container Alt do RDF. Ao utilizar o container Alt, consideramos que os artigos publicados pelo aluno Gabriel podem ser encontrados em qualquer uma das alternativas descritas nesse exemplo.

#### EXEMPLO 2.5: Contêiner Sequence

Nele descreveremos a seguinte sentença:

- Alunos do curso 869 devem cursar as disciplinas Cálculo1, Cálculo2 e Cálculo3, nessa ordem.

Teremos o modelo RDF que segue:

```
<rdf:RDF>
  <rdf:Description about=" http://universidade.edu/cursos/869/grade">
    <s:disciplinas>
      <rdf:Seq>
        <rdf:li resource=" www.universidade.edu/disciplinas/calculo1"/>
        <rdf:li resource=" www.universidade.edu/disciplinas/calculo2"/>
        <rdf:li resource=" www.universidade.edu/disciplinas/calculo3"/>
      </rdf:Seq>
    </s:disciplinas>
```

```
</rdf:Description>
</rdf:RDF>
```

A grade do curso é um recurso identificado como <http://universidade.edu/cursos/869/grade>. As disciplinas que compõem essa grade são consideradas propriedades do curso 869 e são descritas dentro do elemento `<s:disciplinas>` `</s:disciplinas>`, utilizando o container Seq do RDF. Ao utilizar o container Seq, consideramos que os alunos que fazem parte do curso 869 irão ter uma grade de disciplina compostas por Calculo1, Calculo2 e Calculo3, nessa ordem.

#### 2.4.1.4. Esquemas RDF

Através do RDF, podemos definir de forma padronizada a representação de metadados na *Web*. O nosso trabalho está compreendido no escopo da Semântica na *Web* e, ao focar o RDF nesse contexto, percebemos que a sua especificação somente não é suficiente para descrever metadados de uma forma associada, representando semanticamente um dado domínio de conhecimento. Quando citamos a representação semântica, queremos nos referenciar principalmente à definição de ontologias e conseqüentemente de todas as características das informações por elas requeridas como definição de significado, definição de possíveis relacionamentos, propriedades, herança de propriedades e restrições de valores.

Assim, a W3C definiu o conceito de esquema RDF para expressar a estrutura de metadados de documentos na *Web* para um dado domínio e é utilizado para descrição de vocabulários [W3C – RDFSchema].

O Esquema RDF utiliza o modelo RDF associado a regras, formando um vocabulário do esquema, que define a estrutura dos dados, de forma similar a um esquema tradicional de banco de dados. Logo, os dados definidos de acordo com o esquema estarão sempre em conformidade com ele.

O vocabulário do Esquema RDF é definido em um espaço de nomes denominado '*rdfs*' e identificado pela URI <http://www.w3.org/2000/01/rdf-schema#>. A especificação também utiliza o prefixo '*rdf*' para referir-se ao espaço de nomes do modelo e sintaxe RDF, identificado pela URI <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

O esquema RDF utiliza um conceito bastante comum nas linguagens de programação orientadas a objetos: classes e, conseqüentemente, instâncias. Baseado no esquema, poderemos definir várias classes, com propriedades, relacionamentos e as instâncias dessas classes são os recursos. Um recurso pode ser instância de uma ou várias classes.

Dentro do vocabulário do esquema RDF, existem dois conceitos de classes: *rdfs:Resource* e *rdfs:Class*. O *rdfs:Resource* é a classe principal e pode ser utilizada para definir qualquer coisa. Todas as outras classes são definidas como subclasse dela. O *rdfs:Class* é utilizado para definir qualquer classe e é similar ao conceito de classes nas linguagens de programação orientadas a objeto. Quando desejamos definir um recurso qualquer como uma classe, devemos descrevê-lo como uma propriedade *rdf:type* que possuirá como valor o *rdfs:Class*. Isto quer dizer que aquele recurso que está sendo definido é do tipo classe. A propriedade *rdf:type* indica que um recurso é instância de uma classe, possuindo, conseqüentemente todas as características da classe mãe.

Ao citarmos o elemento *rdf:type*, descrevemos o mesmo como uma propriedade. As propriedades são utilizadas para indicar os possíveis relacionamentos entre as classes. O vocabulário do esquema RDF nos oferece outras propriedades além dessa e qualquer uma delas é uma instância da classe *rdf:Property*. Ressaltaremos, a seguir, algumas das principais propriedades:

- *rdfs:subClassOf* – indica que uma classe é uma subclasse de outra. Assim, a classe filha herda todas as características inerentes a classe mãe, sendo que uma subclasse pode pertencer a mais de uma classe.
- *rdfs:subPropertyOf* – essa propriedade possui a mesma idéia da propriedade *rdfs:subClassOf*, sendo que nesse caso nos referenciamos a propriedades. Logo, podemos considerar que ela é uma instância de *rdf:Property* e indica que uma propriedade é uma especialização de uma ou várias outras propriedades.
- *rdfs:seeAlso* – utilizada para fornecer informações adicionais sobre um determinado recurso.

Na FIGURA 2.8, temos um exemplo ilustrando a utilização dos elementos do vocabulário descrito acima. O exemplo mostra uma descrição de Alunos que são representados de acordo com o nível de graduação dos cursos, dos quais eles fazem parte. No

exemplo, temos uma hierarquia de classe e descrevemos que existe uma classe Aluno e que ela tem como subclasses as classes Aluno de Graduação e Aluno de Especialização. A classe Aluno de Especialização possui uma subclasse que é a classe Aluno de Doutorado.

O vocabulário do esquema RDF disponibiliza algumas classes para descrever restrições quanto ao uso das propriedades e classes em dados RDF. A classe *rdfs:ConstraintResource* é uma subclasse de *rdfs:Resource* em que todos os seus objetos são recursos para especificar restrições. A *rdfs:ConstraintProperty* é uma subclasse de *rdf:Property* e *rdfs:ConstraintResource* em que todos os seus objetos são propriedades para especificar restrições [W3C – RDFSHEMA].

```
<rdf:RDF xml:lang="en"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
<rdf:Description ID="Aluno">
  <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf
    rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
</rdf:Description>
<rdf:Description ID="Aluno de Graduacao">
  <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#Aluno"/>
</rdf:Description>
<rdf:Description ID="Aluno de Especializacao">
  <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#Aluno"/>
</rdf:Description>
<rdf:Description ID="Aluno de Doutorado">
  <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="# Aluno de Especializacao "/>
</rdf:Description>
</rdf:RDF>
```

**FIGURA 2.8 Especificação RDF para um Esquema que descreve Alunos**

Um caso muito comum de restrições encontrado no uso dos esquemas RDF é o uso das propriedades *rdfs:range* e *rdfs:domain*. Qualquer propriedade definida através do vocabulário possui duas informações associadas a ela que são obrigatórias para sua definição. Essas informações são descritas através das propriedades *rdfs:range* e *rdfs:domain*, sendo que qualquer propriedade que for descrita de acordo com o esquema deverá possuir um valor e um domínio associado. Com o *rdfs:range* dizemos os possíveis valores que uma propriedade poderá assumir. Tais valores devem estar representados por uma classe. O *rdfs:range* é uma instância de *rdfs:ConstraintProperty*. O *rdfs:domain* é utilizada para indicar a qual, ou quais, classes pertence um determinada propriedade.

Através das propriedades *rdfs:comment* e *rdfs:label*, podemos descrever informações mais detalhadas sobre recursos e criar assim uma espécie de documentação para ser entendida por pessoas.

## 2.5 Linguagens de Representação para Ontologias

Quando pensamos em definição de ontologias, não podemos esquecer que as ontologias devem conter termos associados às mais diversas características que podem representar fatos ou situações do mundo real, relacionados a cada um deles e que a partir de um simples fato, vários outros fatos podem ser inferidos ao termo de uma forma automática. Portanto, caminhando para um maior amadurecimento da definição de ontologias, linguagens específicas têm sido desenvolvidas e pesquisadas.

As linguagens para representação de ontologias se propõem a estender o poder de expressão semântica dos modelos já existentes e já utilizados para definição de ontologias. Evidentemente, elas oferecem algumas flexibilidades interessantes, porém seria bastante prematuro definirmo-las como um padrão para definição de ontologias uma vez que é muito difícil definir e padronizar todas as regras semânticas que poderiam estar envolvidas na descrição das ontologias. Mesmo assim, tal estudo é bastante relevante uma vez que as linguagens se propõem a oferecer representações semânticas mais detalhadas, para alcançar um estágio em que as ontologias possam assegurar um suporte semântico para acompanhar a evolução e automatização de serviços amplamente requisitados e disponibilizados pela *Web*.

A seguir, será feita uma breve apresentação das principais linguagens que se destinam a tal propósito.

### 2.5.1 OIL (Ontology Inference Layer ou Ontology Interchange Language)

A OIL (*Ontology Inference Layer* ou *Ontology Interchange Language*) é uma proposta para padronizar a forma de representar e expressar ontologias na *Web* estando concentrada na representação de modelos de domínios.

Para definir a sua linguagem, a OIL utiliza a sintaxe XML, estendendo o RDF e o esquema RDF ao adicionar primitivas que ainda não existiam no esquema RDF e criando uma linguagem designada especificamente para definir ontologias de uma forma cada vez mais expressiva [Klein et al. - 2000]. Como o OIL estende o RDF de forma compatível, uma ontologia escrita em OIL é um válido documento RDF [Horrocks et al. – 2000].

Além disso, a OIL define semântica baseada em lógica de descrição (*Description Logic*) e possibilita agregar ao processo de definição das ontologias uma espécie de raciocínio automático, ou seja, algumas regras que são utilizadas para fazer inferências e podem ser definidas para verificar automaticamente a consistência das informações geradas.

Tal como o esquema RDF, quando da representação das ontologias, a OIL permite que sejam definidas classes, propriedades e relacionamentos entre as classes. As classes são definidas através da expressão **class-def** e as propriedades através da expressão **slot-constraint**. Uma das características diferenciais da OIL em relação ao esquema RDF é que ela permite que instâncias de classes e propriedades sejam definidas utilizando combinações booleanas de outras classes e propriedades, respectivamente, através dos operadores **AND**, **OR** ou **NOT** [Horrocks et al. – 2000].

Com o operador **AND**, poderemos adicionar classes ou propriedades e definir instâncias que agregam as características dessas classes. Por exemplo, para uma ontologia do domínio automóvel, poderíamos definir uma classe para Carro e definir uma classe para Modelo Esportivo. Com o operador **AND**, poderíamos definir uma classe **Carro AND Modelo Esportivo**, onde todas as instâncias dessa classe seriam instâncias da classe Carro e da classe Modelo Esportivo.



Utilizando o mesmo exemplo para o operador OR, teríamos que todas as instâncias da classe **Carro OR Modelo Esportivo** seriam instâncias da classe Carro ou da classe Modelo Esportivo.

Com o operador NOT, a classe **NOT Modelo Esportivo** define que as instâncias dessa classe não são instâncias da classe Modelo Esportivo.

Esses operadores lógicos podem ser utilizados separadamente ou podem ser utilizados numa só expressão para definir uma classe, por exemplo **NOT (Carro OR Modelo Esportivo)** define que as instâncias dessa classe não são instâncias da classe Carro e não são também da classe Modelo Esportivo.

A OIL define também alguns atributos adicionais para as propriedades como: **min-cardinality** e **max-cardinality** [Horrocks et al. – 2000]. Esses atributos definem números inteiros não negativos que associa uma quantidade a propriedade que será atribuída a todas as instâncias da classe que a possua. No caso de uma classe Carro AND Modelo Esportivo, temos:

**slot-constraint** *porta*  
**max-cardinality** 4 porta com barras de aço  
**min-cardinality** 2 porta com barras de aço

**FIGURA 2.9 Exemplificando o uso dos atributos max-cardinality e min-cardinality**

Na FIGURA 2.9, a propriedade porta foi definida com dois atributos. O primeiro define a **quantidade máxima** de portas que as instâncias da classe Carro AND Modelo Esportivo podem ter, nesse caso 4 e elas são portas com barra de aço. Acompanhando o mesmo raciocínio, o segundo define que a **quantidade mínima** de portas é 2.

Outro atributo bastante interessante disponibilizado pela OIL para definição de propriedades é o **inverse**. Com esse atributo é possível definir uma propriedade e ao mesmo tempo definir a sua propriedade inversa, ou seja, definiremos uma propriedade **slot-def** é

fabricado por associada a **inverse** fabrica. Utilizando a mesma classe Carro AND Modelo Esportivo, poderíamos associar a propriedade definida, **slot-constraint é fabricado por** com o valor **value-type Indústria Automobilística**. Logo, se o Carro Esportivo é fabricado pela Indústria Automobilística, a Indústria Automobilística fabrica Carros Esportivos.

Podemos considerar a OIL como um importante passo para criação de um modelo para definir ontologias que deverá passar por algumas fases de amadurecimento, o que é muito comum na definição de qualquer modelo.

### 2.5.2. DAML+OIL

DAML (*DARPA Agent Markup Language*) é uma linguagem de marcação proposta pelo DARPA (*Defense Advanced Research Projects Agency*) para construir ontologias. Segundo Jim Hendler, pesquisador do DARPA e um dos criadores da DAML, a função do DAML é propiciar interoperabilidade semântica entre páginas na *Web*, bancos de dados e programas através da definição do significado das informações disponibilizadas na *Web*, fazendo com que essas informações possam ser lidas e entendidas por máquinas [Rapoza – 2000].

DAML fornece um conjunto de elementos específicos que possibilita a definição e criação de ontologias cada vez mais ricas em detalhes e conceitos semânticos, tal como a OIL. Por isso, em dezembro de 2000, a DAML mudou seu nome para DAML+OIL indicando a união das duas linguagens.

A DAML é escrita em sintaxe RDF. A DAML+OIL é um tipo específico de linguagem baseada em RDF, uma vez que ela parte da sintaxe RDF e esquemas RDF adicionando a eles novas características, através de novos elementos que oferecem maiores facilidades para descrever o significado das informações disponibilizadas na *Web*. Dentre as novas características propostas pela DAML+OIL podemos citar [DAML+OIL]:

- Representação de informações que compreendem conjuntos delimitados, por exemplo, João tem 3 filhos;
- Expressões de classes tipo união (**unionOf**) e intersecção (**intersectionOf**).

- Equivalências (**equivalentTo**) para propriedades, classes e instâncias;
- Propriedades qualificadas, tipo cardinalidade máxima (**maxCardinalityQ**) e cardinalidade mínima (**minCardinalityQ**) para qualificar a representação de uma propriedade; e
- Regras de inferências que fornecem informações adicionais tipo **inverseOf** e **disjointWith** para serem exploradas por máquinas de raciocínio.

### 2.5.3 OWL (Ontology Web Language)

A OWL (*Ontology Web Language*) é a linguagem de marcação semântica proposta recentemente (Julho de 2002) pela W3C para representar e compartilhar ontologias na *Web* [W3C – OWLref].

A OWL é derivada da DAML+OIL e é construída sob a sintaxe RDF. Além das facilidades já propostas pela DAML+OIL, a OWL disponibiliza outras características que se concentram basicamente em proporcionar maiores flexibilidades para propiciar que as ontologias possam ser definidas sob uma forma semântica cada vez mais rica em detalhes.

A OWL fornece representações semânticas mais expressivas para as informações na *Web*, através de elementos específicos para definição de ontologias, descrevendo semanticamente suas classes e relacionamentos de uma maneira cada vez mais rica e detalhada [W3C – OWLreq].

Uma ontologia definida em OWL pode ser considerada basicamente como uma seqüência de axiomas e fatos. A OWL considera como fatos informações sobre um particular indivíduo, descritas como classes que pertencem a ele, suas propriedades e valores.

A descrição de uma ontologia poderá incluir também referências para outras ontologias, desde que exista uma correlação entre os itens definidos pelas mesmas. Assim, a descrição de uma ontologia *Onto\_1* poderá incluir indiretamente a descrição de uma outra ontologia *Onto\_2* ao ser criada uma relação entre as ontologias através de referências, por exemplo, através da utilização de URIs para identificar as mesmas na *Web*.

Os axiomas são utilizados para associar as classes às propriedades, além de fornecer algumas características descritivas e lógicas para elas. Por exemplo, podemos definir que algumas propriedades são equivalentes ou que uma propriedade é sub-propriedade de uma outra etc. Basicamente, os axiomas são divididos em axiomas para: classes, propriedades, restrições e descrições.

Dentre os axiomas para classes, temos axiomas que definem que:

- Uma classe é equivalente a uma superclasse, incluindo suas propriedades, restrições e descrições;

Ex:

```
<axiom> ::=
    EquivalentClass( <classID> {super=<classID>}
                    {<restriction>}{<description>})
```

- Definem uma classe como conjunto de outros indivíduos;

Ex:

```
<axiom> ::=
    EnumeratedClass( <classID> {<individualID>} )
```

- Algumas descrições podem ser subclasses de outra, ou podem ser equivalentes ou não.

Ex:

```
<axiom> ::=
    DisjointClasses( <description> {<description>} )
<axiom> ::=
    EquivalentClasses( <description> {<description>} )
<axiom> ::=
    SubClassOf( <description> <description> )
```

As propriedades são definidas como dois grupos distintos: a propriedade propriamente dita e o seu valor. Assim, a OWL utiliza axiomas para definir cada grupo de propriedades, como podemos ver no exemplo a seguir:

Ex:

```
<axiom> ::=
    DataProperty ( <dataPropertyID>
                  {super=<dataPropertyID>}
                  {domain=<description>} {range=<dataRange>}
                  [Functional] )
```

```
<axiom> ::=
    IndividualProperty ( <PropertyID>
                        {super=<individualPropertyID>}
                        {domain=<description>}
                        {range=<description>}
                        [inverseOf=<individualPropertyID>]
                        [Symmetric]
```

```

[Functional | InverseFunctional |
OneToOne | Transitive] )

<axiom> ::=
  EquivalentProperties( <dataPropertyID>
                        {<dataPropertyID>} )

<axiom> ::=
  SubPropertyOf( <dataPropertyID> <dataPropertyID> )

<axiom> ::=
  EquivalentProperties( <individualPropertyID>
                        {<individualPropertyID>} )

<axiom> ::=
  SubPropertyOf( <individualPropertyID>
                 <individualPropertyID> )

```

Muitas das informações utilizadas para descrever as propriedades em OWL são definidas mais naturalmente sob a forma de restrições, bastante utilizadas para atribuir cardinalidade e “valores” (*range*) às propriedades.

Ex:

```

<restriction> ::=
  restriction( <dataPropertyID> [range=<dataRange>]
              {required=<dataRange>}
              {value=<dataValue>}

              [cardinality=<cardinality>] )

<restriction> ::=
  restriction( <individualPropertyID> [range=<description>]
              {required=<description>}
              {value=<individualID>}
              [cardinality=<cardinality>] )

<cardinality> ::=
  atleast( <positive-integer> )
  | atmost( <non-negative-integer> )
  | atleast( <positive-integer> ) atmost( <non-negative-integer> )
  | exactly( <non-negative-integer> )

```

A definição das descrições utiliza o identificador da classe (*classID*) e o construtor da restrição e podem ser formadas por combinações lógicas de outras descrições ou de conjuntos de indivíduos.

Ex:

```

<description> ::=
  <classID>
  | <restriction>
  | UnionOf( <description> {<description>} )
  | IntersectionOf( <description> {<description>} )
  | ComplementOf( <description> )
  | OneOf( {<individualID>} )

```

Todos os exemplo ilustrados acima foram extraídos de um documento disponibilizado pela W3C [W3C – OWLFeature].

## 2.6 Linguagem de Consulta – XMLQL

A sintaxe XML cria uma nova forma para representar informações. Essa forma faz com que fontes de informações sejam geradas num padrão novo, diferentemente do tradicional onde as informações são armazenadas em tabelas de bancos de dados relacionais.

Uma vez que tenhamos uma fonte de informação, pressupõe-se que queiramos manipulá-la posteriormente e, independentemente de onde ou como ela está armazenada, um dos principais serviços disponibilizado nesse contexto é a consulta aos dados.

A consulta às informações é uma das principais e mais utilizadas atividades efetuada na *Web*. Como já foi dito, as informações disponíveis na *Web* possuem um formato específico. Para que as consultas sejam feitas e tenham resultados satisfatórios, são utilizadas linguagens de consultas também específicas e apropriadas para recuperar tais informações.

Como a linguagem de representação de sintaxe que será utilizada em nosso projeto será a XML, foi escolhido o XML-QL como a linguagem utilizada para realizarmos as consultas a base de dados descritas em XML. Não é objetivo do nosso trabalho avaliar e escolher a melhor ou mais eficiente linguagem de consulta disponibilizada para dados XML. O XML-QL nos permitiu a recuperação das informações armazenadas na sintaxe XML de uma forma mais refinada e precisa. Para o nosso projeto, atender tal objetivo fora suficiente.

O XML-QL combina a sintaxe XML com técnicas de linguagens de consulta. É uma linguagem declarativa e não é rigidamente estruturada (uma vez que o esquema existe juntamente com os dados do documento). Ele organiza dados através do conceito de elementos do XML que mantém valores estruturados (através dos DTDs que funcionam como um esquema) e são representados por tags.

O XML-QL assume o modelo de dados semiestruturado representado por um grafo e é baseado na sintaxe *where/construct* ao invés da tradicional *select/from/where*, utilizadas nas linguagens convencionais [Abiteboul et al. - 2000].

Consideremos um exemplo onde desejamos consultar todos os artigos publicados pelo aluno Roberto Barros. A sintaxe da consulta na linguagem XML-QL será:

```
Where <artigo>
  <aluno><nome>Roberto Barros</nome></aluno>
  <titulo> $T </titulo>
  <datapublicacao> $D </datapublicacao>
  </artigo> in www.abc/exemplo.xml
construct $ T
```

**FIGURA 2.10 Exemplo de uma consulta na sintaxe XML-QL**

Baseado na FIGURA 2.10, podemos perceber que na consulta XML-QL existe uma estrutura padronizada que na realidade representa a estrutura da fonte de informação que está sendo consultada. Isto é, as informações armazenadas em [www.abc/exemplo.xml](http://www.abc/exemplo.xml), possuem um esquema utilizado para definir a informação artigo que é:

```
<artigo>
  <aluno><nome>Roberto Barros</nome></aluno>
  <titulo> $T </titulo>
  <datapublicacao> $D </datapublicacao>
</artigo>
```

**FIGURA 2.11 Esquema utilizado na consulta do exemplo mostrado na FIGURA 2.10**

A partir dessa estrutura, a consulta é definida. As variáveis \$T e \$D são utilizadas para representar de forma genérica todos os valores possíveis que podem ser recuperados para atender a consulta. O elemento construct define o que deverá ser retornado como resultado da consulta. Nesse caso, serão retornados somente os títulos dos artigos publicados por Roberto Barros.

Vale ressaltar que tal como XML-QL, existem outras linguagens de consulta para dados XML. A exploração dessas informações tem sido fortemente impactada pelas propostas da W3C que tem desenvolvido várias tecnologias baseadas em XML. Podemos citar, dentre elas, a Xquery que é uma linguagem de consulta para documentos XML ou coleções de documentos XML [W3C – Xquery].

## 2.7 Trabalhos Relacionados

Durante toda fase de planejamento e desenvolvimento do nosso projeto, analisamos as principais ferramentas destinadas à construção de ontologias, disponíveis na *Web*.

A principal motivação do nosso trabalho foi construir uma ferramenta para criar e manipular ontologias na *Web*. O nosso projeto contempla uma ferramenta que, além de possuir o mesmo propósito das ferramentas existentes, terá algumas características e funcionalidades não encontradas nas demais, mas que consideramos de importância significativa em tais ferramentas.

Dentre as ferramentas analisadas, ressaltaremos e descreveremos a seguir três projetos bastante relevantes. São eles: o Protégé da Universidade de Stanford, o KAON-SOEP da Universidade de *Karlsruhe* e o OILED.

### 2.7.1 Protégé

Desenvolvido por uma equipe de pesquisadores da Universidade de Stanford, há quinze anos, o **Projeto Protégé** é, atualmente, um grande referencial de ferramentas para modelar e construir bases de conhecimento, utilizadas para adicionar semântica a *Web*.

A atual versão do projeto disponibiliza o Protégé-2000 que é uma ferramenta extensível e de código aberto para editar bases de conhecimento na *Web*. Ele foi concebido para desenvolver ontologias e aquisição de conhecimento, podendo ser adaptado para editar modelos em diferentes linguagens da *Web Semântica* [Stanford – 2001].



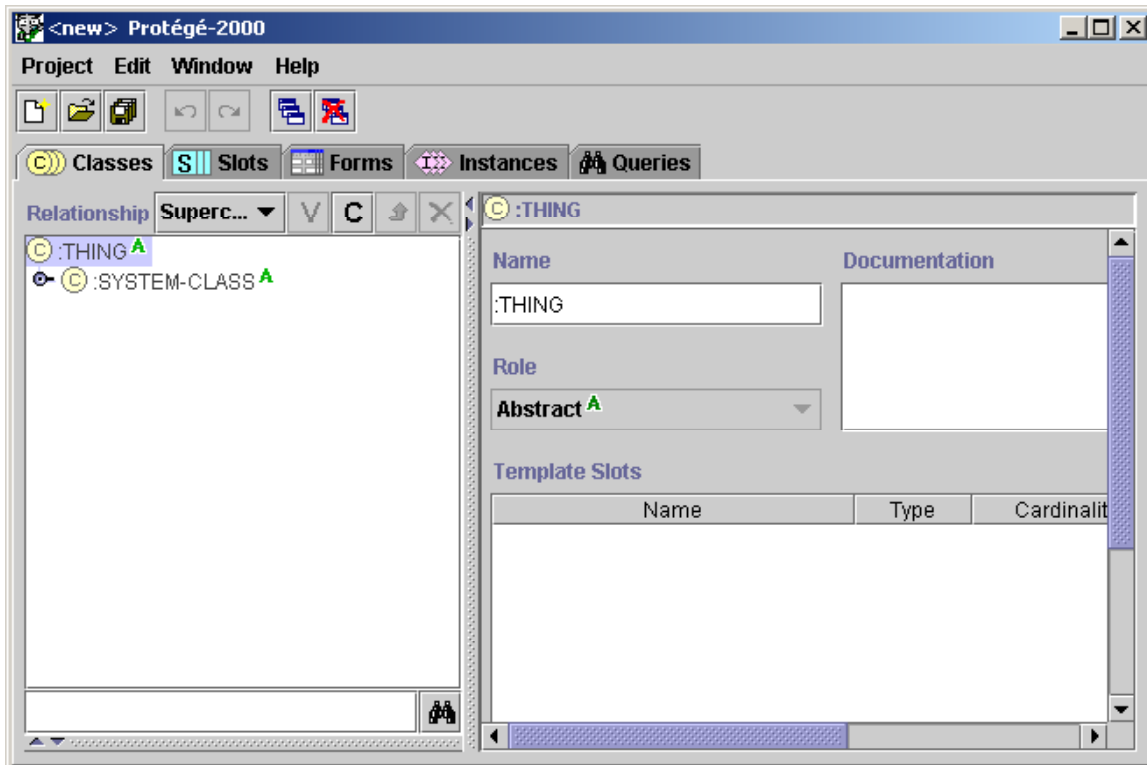
O modelo de aquisição de conhecimento da Protégé é um modelo abstrato e as ontologias são definidas através de uma interface gráfica, criando-se conceitos do domínio que se deseja representar sob uma estrutura de árvore, organizados numa hierarquia de subclasses.

A interface gráfica do Protégé é baseada no padrão OKBC (*Open Knowledge-Base Connectivity*), bastante utilizado em projetos da Universidade de Stanford, que é uma API (*application-programming interface*) para acessar bases de conhecimento armazenadas em sistemas de representação de conhecimento (KRS) [OKBC]. Além disso, o OKBC fornece uma interface gráfica genérica a ferramentas desenvolvidas para construção de sistemas de representação de conhecimento baseados em quadros.

Uma ontologia definida no Protégé consiste em:

- **Classes** que correspondem aos conceitos (termos) encontrados num domínio.
- **Slots** que são propriedades das classes e das instâncias.
- **Facets** que são propriedades dos *slots* [Noy et al. – 2000].

Segue abaixo, uma figura ilustrativa mostrando uma das janelas da interface do Protégé-2000.



**FIGURA 2.12 Interface do Protégé-2000**

Para exemplificar, consideraremos uma ontologia para o domínio Universidade e dentro dela deveremos representar os conceitos Aluno, Aluno de Graduação, Aluno de Mestrado e Aluno de Doutorado. Com o Protégé, poderíamos definir uma classe para Aluno e colocar todas as outras variações do conceito Aluno como subclasses distintas da classe Aluno, ou seja, num modelo de árvore, essas variações estariam hierarquicamente subordinadas à classe Aluno, que por sua vez poderia estar subordinada a outra classe e assim sucessivamente.

Continuando a definição, cada Aluno possui propriedades como nome e endereço. Cada propriedade é definida como um *slot*. Os *slots* no Protégé são objetos de primeira classe, ou seja, eles podem ser definidos independentemente de qualquer classe e nós podemos definir uma propriedade nome e ela pode ser associada a uma classe Aluno e depois essa mesma propriedade pode ser associada a uma classe Professor.

Tais propriedades sempre que definidas numa classe serão herdadas por todas as subclasses que possam ser subordinadas a mesma – Herança.

Nos casos de representação de conceitos como Herança Múltipla, a interface gráfica baseada em árvores dificulta uma representação direta. Por exemplo, um termo C é subclasse de A e de B. Esses casos são contornados, criando-os graficamente na ordem que se segue: serão criadas as classes A e B; logo depois será criada uma classe C, subordinada à classe A e outra classe C subordinada à classe B.

Além de ser um editor de ontologias, o Protégé destina-se a outras funções como:

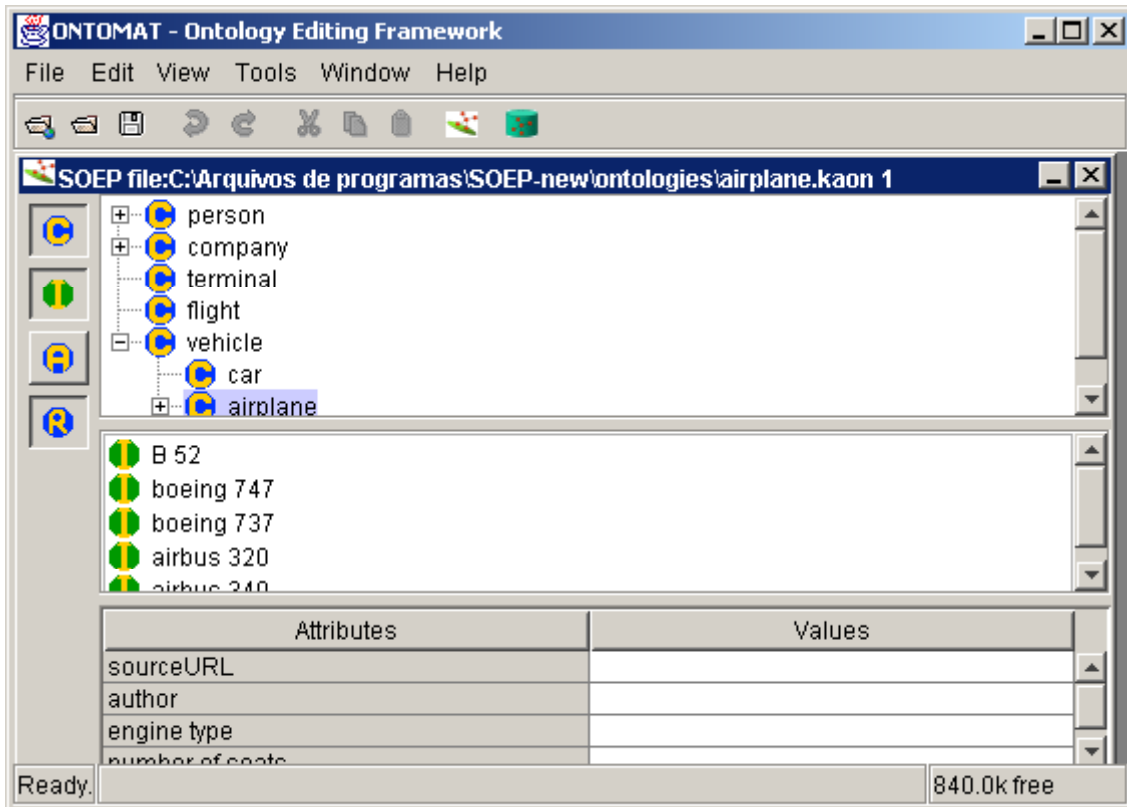
- Interoperabilidade entre sistemas, uma vez que ela se propõe a ser um mediador na comunicação entre diferentes bases e sistemas de representação do conhecimento. Como existem bases de dados representadas em diferentes linguagens, essa mediação flexibiliza o compartilhamento e a reutilização dessas informações [Noy et al. – 2000].
- Construção de ontologias colaborativas com migração, integração e controle de versão [Noy et al. – 2000].

Apesar de ser uma das mais expressivas ferramentas para representação de bases de conhecimento, o Protégé não disponibiliza uma integração a banco de dados. Ao criar uma ontologia no Protégé, é gerado um arquivo num formato próprio e desenvolvedores podem transformar esse formato em um outro formato, com uma outra sintaxe, através de *parsers* que são disponibilizados pelo Protégé. Assim, os desenvolvedores podem ler e gravar arquivos em qualquer sintaxe de linguagem, desde que já exista um *parser* disponível pelo Protégé para essa linguagem [Noy et al. – 2001].

### **2.7.2 KAON-SOEP (Simple Ontology Browser and Editor Plugin)**

O KAON (*Karlsruhe Ontology and Semantic Web Infrastructure*) é uma infraestrutura que provê serviços para definir, apresentar e manipular ontologias e metadados. A idéia dessa infraestrutura é fornecer serviços básicos para aplicações envolvidas na *Web Semântica*, através de uma família de ferramentas que vem crescendo cada vez mais [Handschuh et al. – 2001].

Dentro da infraestrutura KAON, temos o KAON-SOEP que é um editor de ontologias simples. Apesar de não possuir uma interface tão arrojada e flexível como o Protégé, sua interface, sob o modelo gráfico de árvore, assemelha-se a interface da Protégé (ver FIGURA 2.13).



**FIGURA 2.13 Interface do KAON-SOEP**

Basicamente, uma ontologia definida no KAON-SOEP consiste na representação dos seguintes elementos: classes, relacionamentos, atributos (similar a propriedades) e instâncias.

O KAON-SERVER tem como objetivo implementar um repositório central de ontologias. Ela possui as seguintes características: persistência dos dados; permitir atualização dos dados; gerenciar questões ligadas à concorrência no acesso aos dados, tipo atomicidade e integridade; e garantir segurança para essas informações.

A proposta da ferramenta KAON-REVERSE é a de permitir que bases de dados relacionais sejam mapeadas para ontologias, ou seja, extrair os dados contidos nos banco de dados, definir alguns mapeamentos para transformá-los em instâncias e em instâncias de

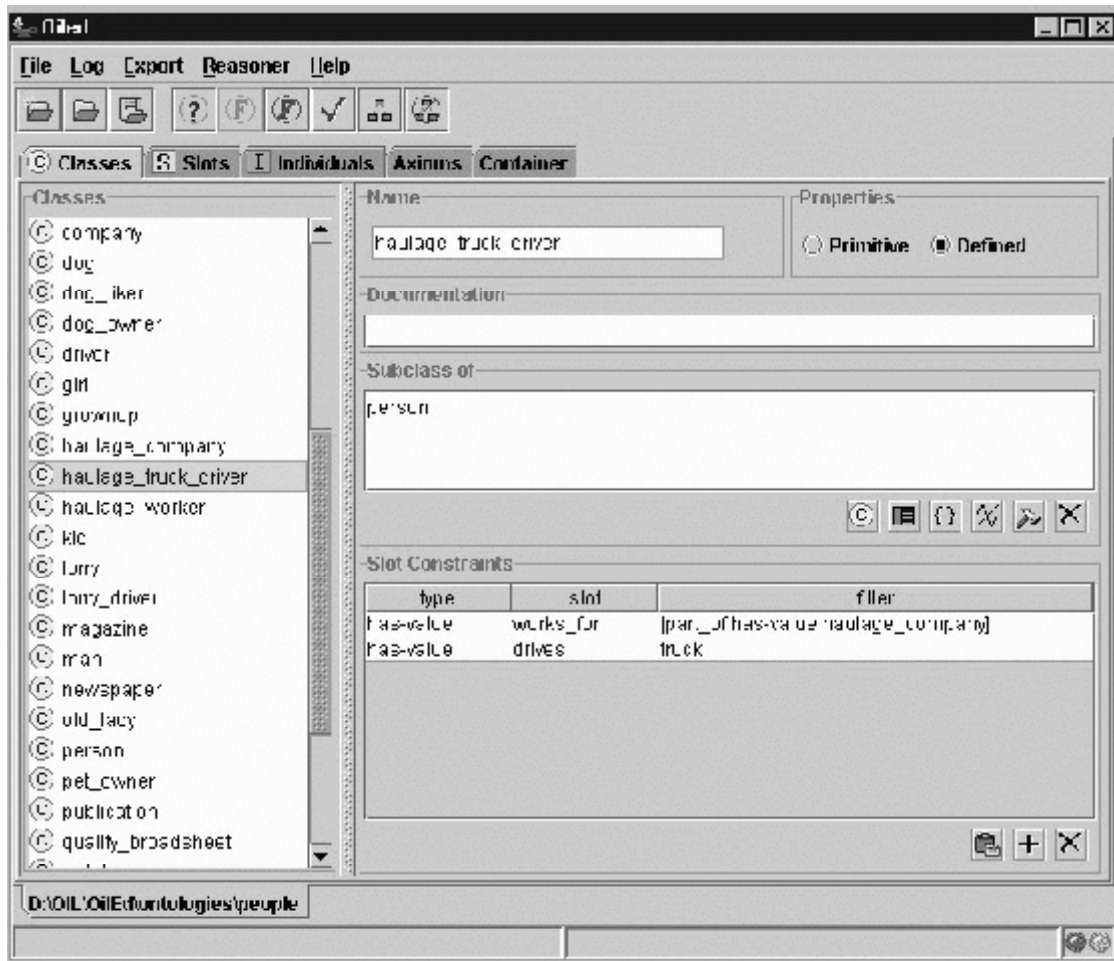
relacionamentos. Tais mapeamentos são propostos pela ferramenta e definidos manualmente via sua interface [Handschuh et al. – 2001].

### 2.7.3 OilEd

O OilEd é um editor de ontologias simples que suporta a construção de ontologias baseada na linguagem de representação OIL [Bechhofer et al. – 2001].

Sua interface gráfica foi fortemente influenciada por outras ferramentas com o mesmo propósito, como por exemplo, o Protégé. Entretanto, o design da ferramenta foi planejado e concebido para demonstrar como estender o poder de expressão dos modelos de conhecimentos baseados em quadros (*frames*) e como aplicar raciocínio para suportar a criação e manutenção de ontologias.

Uma ontologia definida no OilEd consiste na descrição de *classes*, *slots*, *individuals* e *axioms*. Segue abaixo, a FIGURA 2.13 mostrando a interface do editor OilEd.



**FIGURA 2.14** Interface do OilEd

Dentre esses elementos, daremos uma atenção especial aos *axioms*. Os *axioms* são utilizados por vários editores nas definições de ontologias para afirmar fatos sobre as classes e seus relacionamentos. Os *axioms* mais comuns são usados para classificar classes, numa relação de equivalência sob uma hierarquia de classes e subclasses (*subsumption/equivalence axiom*). Os *axioms* definidos pela OilEd suportam, além disso, a definição *disjointness classes*, onde para um dado conjunto de classes (*covering*) poderemos dizer que uma instância desse conjunto poderá ser exatamente instância de uma única classe desse conjunto [Bechhofer et al. – 2001].

Uma outra característica do OilEd é o uso de serviços de raciocínio para verificar semanticamente consistência de classes e inferir relacionamentos. O serviço de raciocínio utilizado é o sistema *reasoner* FaCT (*Fast Classification of Terminologies*) [FaCT], porém

qualquer *reasoner* com funcionalidades e conectividade poderá ser utilizado [Bechhofer et al. – 2001].

Ontologias editadas no OilEd podem ser exportadas para arquivos em alguns formatos: *OIL*, *OIL-RDFS*, *DAML-OIL* e *HTML*. Entretanto, o OilEd não possui nenhum tipo de integração a banco de dados.

Como já foi dito, a proposta principal do OilEd é ser um editor de ontologias simples, baseado em quadros, onde usuários poderão definir suas ontologias, usufruindo o poder de expressão semântica suportado pela linguagem OIL que a ferramenta integra aos seus *frames* (quadros).

#### **2.7.4 Quadro Comparativo com principais características das ferramentas**

Para maior clareza sobre o entendimento das funcionalidades propostas pelas ferramentas descritas nesse capítulo, segue a TABELA 2.2, um quadro comparativo com algumas das principais características disponibilizadas pelas mesmas.

Características	Editores de Ontologias		
	PROTEGÉ	KAON-SOEP	OILED
Interface gráfica	Rica em funcionalidades	Mais simples	Rica em funcionalidades
Formato gráfico utilizado na interface para representação das ontologias	Árvore	Árvore	Árvore
Herança múltipla entre os termos de uma ontologia	Não permite	Não permite	Não permite
Componentes básicos de uma ontologia	Classes, slots e facets	Classes, instâncias, atributos e relacionamentos	Classes, slots e axioms
Modelo abstrato para definição de ontologias	Modelo próprio, não é baseado em nenhuma linguagem específica.	Modelo próprio, não é baseado em nenhuma linguagem específica.	Baseado na linguagem OIL.
Integração com o modelo RDF	Disponibilizada	Disponibilizada	Disponibilizada
Integração direta a banco de dados e SGBDs	Inexistente	Inexistente	Inexistente
Utilização de regras de raciocínio e de inferências	Não disponibilizadas	Não disponibilizadas	Disponibilizadas

**TABELA 2.2** Quadro comparativo das principais ferramentas



## 2.8 Conclusão

Utilizamos esse capítulo para descrever a fundamentação teórica direta e indiretamente envolvida no nosso trabalho, dentro do escopo da *Web Semântica*.

Antes de chegarmos ao projeto e conseqüentemente conclusão da ferramenta OntoEditor, vários aspectos imprescindíveis ligados a *Web Semântica* foram analisados. O embasamento teórico, descrito de forma bem sucinta nesse capítulo, foi indispensável para que pudéssemos traçar os caminhos que deveríamos seguir nas etapas posteriores (projeto e implementação) até a conclusão do nosso objetivo com êxito.

## Capítulo III - PROJETO ONTOEDITOR

O nosso projeto contempla a concepção e o desenvolvimento de uma ferramenta para manipular ontologias na *Web*: o OntoEditor.

Outras ferramentas estão disponibilizadas na *Web* com o propósito de representar ontologias na *Web*, tal como o OntoEditor. Entretanto, a nossa idéia foi construir uma ferramenta com a qual os seus usuários poderiam, além de representar as ontologias graficamente de forma bem simples e objetiva, armazená-las, e posteriormente manipulá-las, num banco de dados relacional, num formato que já fosse aprovado e utilizado para representação de metadados na *Web* pela comunidade envolvida na *Web* Semântica.

Não é escopo do nosso trabalho oferecer uma ferramenta com uma interface rica em funcionalidades. Como já foi dito, outras ferramentas já são utilizadas para representar ontologias e elas foram definidas por equipes de pesquisadores que vem as aprimorando por vários anos, oferecendo interfaces, cada vez mais, arrojadas e eficientes. Logo, a nossa proposta é acrescentar algumas idéias pouco ou inexploradas por outros editores, sob a forma de funcionalidades que consideramos importantes quando da execução do processo de definir, representar e manipular ontologias no ambiente *Web*.

Descreveremos, nesse capítulo, todo o processo de planejamento, análise e projeto que foram utilizados no desenvolvimento do OntoEditor. Em seguida, mostraremos um estudo de caso utilizando o OntoEditor, enfatizando as funcionalidades e características da ferramenta.

### 3.1 O Contexto do OntoEditor

Como já foi mostrado, no capítulo II, existem outras ferramentas destinadas à representação e definição de ontologias na *Web*. Essas ferramentas concentram seus

propósitos na oferta de editores com interfaces gráficas ricas em funcionalidades para editar, integrar e traduzir ontologias.

Como essas ferramentas não possuem integração a banco de dados, as ontologias criadas não podem ser guardadas, gerenciadas e manipuladas diretamente pelo editor com todas as funcionalidades oferecidas e asseguradas por um SGBD. Ressaltamos também que, à medida que pensamos num editor integrado a um banco de dados, estamos considerando a viabilidade de construir, via o editor, um repositório de ontologias que possam ser manipuladas, armazenadas, gerenciadas e posteriormente até integradas, visando à construção de uma base de dados funcional e cada vez mais rica em informações semânticas.

Além disso, no início do nosso trabalho, o modelo de conhecimento das principais ferramentas, encontradas para esse propósito, estava baseado graficamente no padrão OKBC [OKBC]. O OKBC cria um tipo de padronização para as interfaces de editores utilizados na representação de ontologias na *Web*. Ele é formado por vários elementos que irão compor a representação das ontologias como interface para definição de classes, propriedades e relacionamentos, basicamente. Como o OKBC é um padrão que possui muitas funcionalidades, as ferramentas utilizam os seus elementos da forma que mais lhes convém para representar as ontologias. Assim, as ferramentas utilizam o padrão OKBC e geram suas ontologias sob um modelo especificado por elas.

Uma das características forte do modelo OKBC é que as ontologias são representadas graficamente sob a forma de árvore. A utilização de uma representação gráfica traz várias vantagens como simplicidade, flexibilidade e visibilidade. A estrutura sob a forma de árvore é a que mais se identifica para representação de dados já disponibilizados na *Web*, uma vez que essas informações são disponibilizadas através de linguagens que possuem um modelo de dados baseado graficamente em árvore, como HTML e XML.

Todavia, ao definirmos uma ontologia, temos o objetivo de especificar um dado domínio de conhecimento que deverá compreender os mais diversos tipos de informações para representar situações do mundo real. Nesse contexto, a forma de árvore, que é uma estrutura hierárquica, se utilizada para representação gráfica de ontologias, possui algumas limitações. Por exemplo, não podemos expressar diretamente um caso de herança múltipla, muito comum no mundo real, sob uma estrutura de árvore.

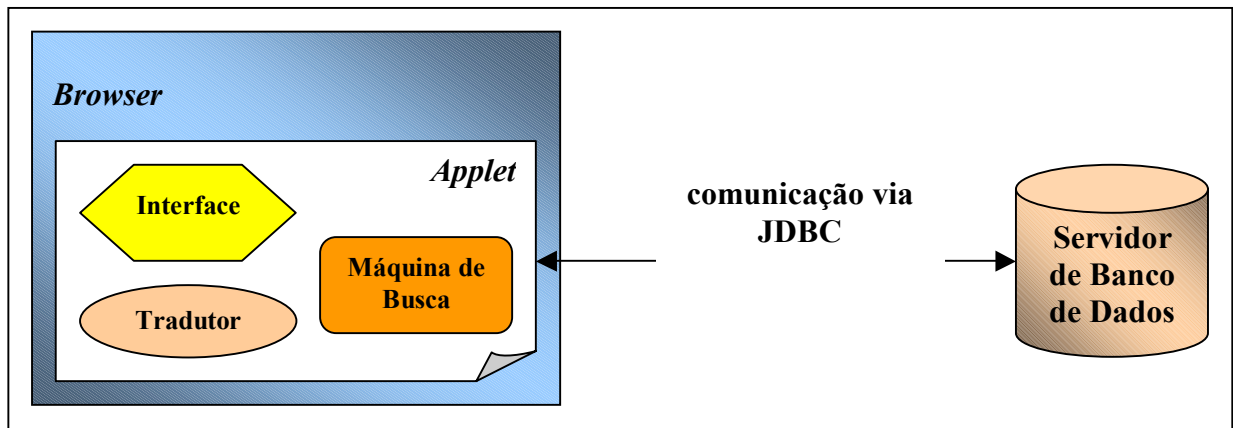
Logo, não foi utilizado em nosso projeto o padrão OKBC, visto que não queríamos utilizar a representação gráfica sob a forma de árvore. As ontologias serão definidas sob um modelo abstrato de conhecimento, baseado na linguagem de representação de metadados RDF, e que propõe a utilização de uma estrutura baseada em grafos direcionados, para tentar suprir as limitações encontradas nos modelos sob a forma de árvore utilizados nesse contexto.

Apesar do OntoEditor ser uma ferramenta disponibilizada para o ambiente Web, podendo, assim, ser executada a partir de qualquer *browser*, sugerimos sua utilização num escopo delimitado, *intranets*. Decidimos restringir a abrangência de atuação da ferramenta principalmente por questões de preservação da segurança das informações (base de dados) que são manipuladas através do OntoEditor.

As *intranets* são subconjuntos da *Web* formadas por usuários que compõem uma determinada comunidade. As ontologias especificadas com a ajuda do OntoEditor deverão expressar basicamente conhecimento e isso se dará através de informações que serão geradas por cada comunidade afim e que serão posteriormente disponibilizadas e manipuladas entre os seus usuários. Logo, com a intranet temos uma comunidade de usuários previamente conhecida que deverá estar comprometida com a segurança das informações disponibilizadas e mantidas por seus usuários.

### 3.2 Arquitetura do OntoEditor

O OntoEditor é uma ferramenta simples cuja arquitetura segue o modelo cliente-servidor, sob duas camadas: Aplicação e Dados. A FIGURA 3.1 ilustra uma visão geral da arquitetura do OntoEditor.



**FIGURA 3.1** Visão Geral da Arquitetura do OntoEditor

A **Camada de Aplicação** é a camada que compreende a ferramenta OntoEditor propriamente dita. Ela será utilizada pelo usuário final para manipular as ontologias na *Web*, compreendendo o modelo conceitual e a sua implementação lógica e física.

Nela, nós temos todo o modelo conceitual de dados, com os seus esquemas, acoplados a sua implementação lógica, que contém todos os objetos com suas regras de negócio e as operações necessárias para o seu funcionamento. Além disso, essa camada define as formas de acesso e manipulação à segunda camada.

Na **Camada de Dados** estão os dados gerados e manipulados pela ferramenta, gerenciados por um SGBD relacional.

A comunicação entre as duas camadas ocorre diretamente via chamadas JDBC (“*Java Database Connectivity*”) [Sun – B], possibilitando que a camada de dados possa estar num sistema computacional diferente da camada de aplicação, o que facilita resolver algumas questões de interoperabilidade.

### 3.2.1 Arquitetura Detalhada do OntoEditor

Como mostra a FIGURA 3.1, a camada de aplicação compreende basicamente as seguintes macro-funcionalidades:

- Interface;
- Tradução das ontologias;
- Armazenamento e Recuperação das ontologias; e

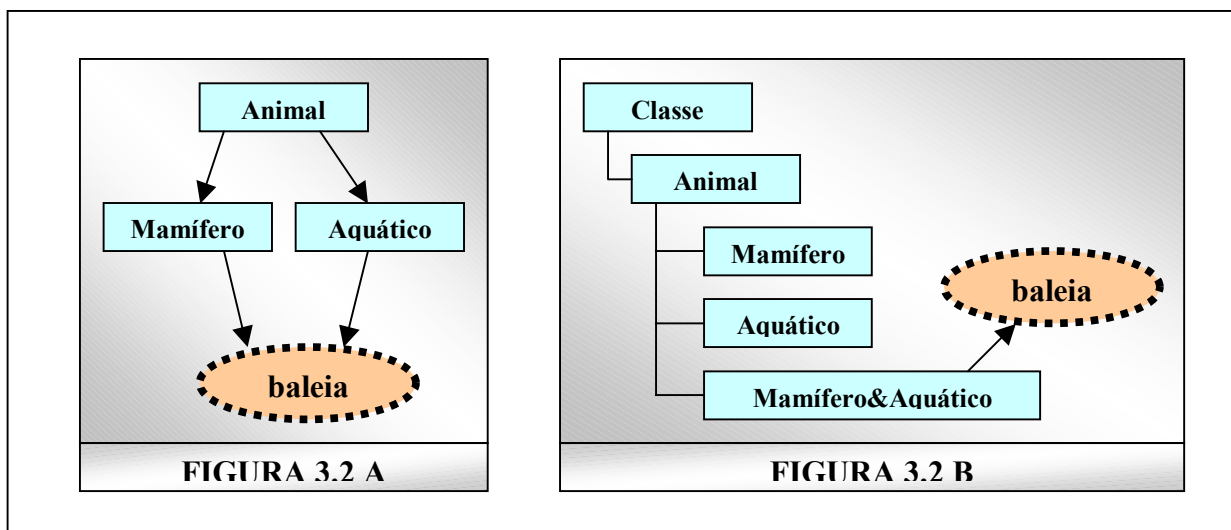
- Consultas.

### 3.2.1.1 Interface OntoEditor

A **Interface Gráfica** do OntoEditor é uma interface *Web* padrão que pode ser executada a partir de qualquer *browser*. Através dessa interface, o usuário da ferramenta pode manipular suas ontologias (criar, editar, salvar, abrir, consultar, etc).

Ao definir uma ontologia, o usuário estará utilizando um modelo abstrato que concebemos observando as funcionalidades básicas necessárias para atender o nosso propósito. Esse modelo foi baseado no modelo de representação de metadados RDF. Com ele, o usuário representará suas ontologias através da criação de grafos direcionados compostos por palavras-chaves que possuirão propriedades e poderão se relacionar de acordo com o contexto do domínio de conhecimento representado.

A FIGURA 3.2 mostra um esboço de como seriam definidos os elementos que irão compor uma ontologia, utilizando o OntoEditor, ao mesmo tempo, comparando com o padrão de representação (árvore) encontrado nas interfaces de outros editores.



**FIGURA 3.2** Exemplo comparativo entre a representação de elementos de uma ontologia sob a forma de grafo e sob a forma de árvore

Utilizamos a FIGURA 3.2, para ilustrar a definição de possíveis conceitos que poderiam compor uma ontologia para o domínio animais. Dentre esses conceitos, temos: Animal, Mamífero e Aquático.

Com a FIGURA 3.2 A, representamos esses conceitos de forma independente uns dos outros, através de um grafo. Essa é a proposta do OntoEditor e, para cada conceito definido, teremos uma classe específica e poderemos ter instâncias para cada uma delas ou para várias, como é mostrado no exemplo, onde a instância baleia é ao mesmo tempo instância da classe Mamífero e da classe Aquático.

Ao considerarmos os conceitos como classes e instâncias, queremos dar a eles a mesma conotação dessas definições compreendidas na orientação a objetos. Encontraremos na definição de ontologias também o conceito de herança e no caso do OntoEditor de Herança Múltipla, sendo que a instância baleia terá todas as propriedades da classe Mamífero e da classe Aquático. Conseqüentemente, as classes Mamífero e Aquático terão todas as propriedades da classe Animal.

No exemplo da FIGURA 3.2 foi utilizado apenas um esboço de como seria a definição de uma ontologia. Muitas outras informações são fornecidas quando estamos definindo uma ontologia, como descrição, propriedades, relacionamentos, etc. Sendo assim, a definição de uma ontologia para Animal, por exemplo:

- englobaria o máximo de informações importantes para caracterizar a natureza do seu domínio;
- associaria sentido e significado semântico a palavras-chaves que forneceriam uma base de informações rica e suficiente para ser utilizada por vários serviços, respondendo diretamente (automaticamente) a diversas questões, tipo: procure por animais que sejam mamíferos, ou mamíferos e aquáticos.

Já a FIGURA 3.2 B, mostra a mesma representação utilizando uma estrutura de árvore, comumente encontrada nas ferramentas utilizadas para definir ontologias na *Web*. Nesse caso, não conseguiríamos criar diretamente uma instância das classes Mamífero e Aquático. O que poderíamos fazer era definir uma terceira classe, Mamífero&Aquático e redefinir nela todas as características das classes Mamífero e Aquático.

Como já foi dito, a interface gráfica do OntoEditor utilizará grafos direcionados para definição de ontologias. Esses grafos são formados por elementos que permitem:

- Criar classes, com nome e descrição;
- Criar Propriedades, com nome e valor; e
- Criar Relacionamentos entre as classes, com seu respectivo nome.

Além disso, a interface disponibilizará opções para outras funcionalidades como:

- Organizar de forma visual o grafo na tela, para uma melhor a visibilidade da ontologia para o usuário;
- Traduzir a ontologia para RDF;
- Visualizar o documento RDF que será armazenado no banco de dados e gerado a partir do grafo;
- Definir as propriedades gerais da ontologia criada, requeridas pela ferramenta tipo nome da ontologia, autor da ontologia, etc Essas características ficarão armazenadas no banco de dados junto com a ontologia propriamente dita. Elas servirão como atributos independentes do conteúdo para identificar a ontologia criada;
- Salvar as ontologias criadas num banco de dados relacional;
- Abrir as ontologias que foram salvas no banco de dados relacional;
- Exportar as ontologias para um arquivo texto convencional;
- Importar as ontologias que foram exportadas; e
- Consultar as ontologias no banco de dados. Essas consultas poderão ser feitas utilizando as informações descritas nas propriedades gerais da ontologia ou através de alguma informação que esteja presente no conteúdo da ontologia.

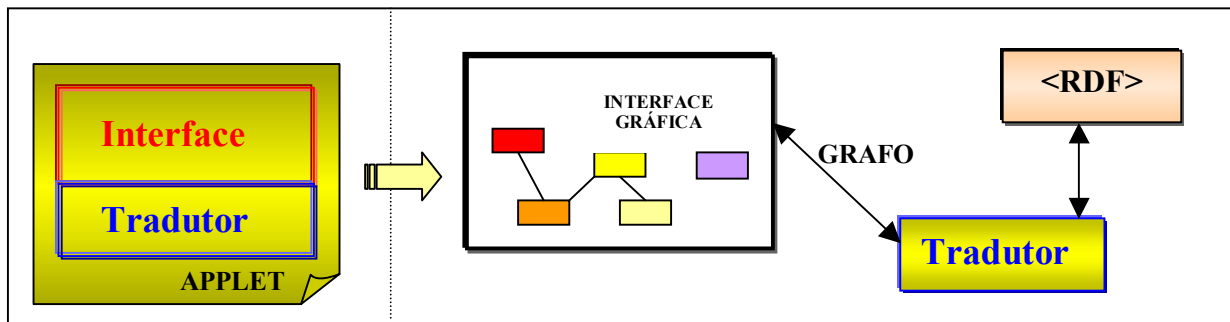
### **3.2.1.2 Processo de Tradução de Ontologias OntoEditor**

Para que as ontologias criadas no OntoEditor sejam armazenadas num banco de dados, o modelo de grafos será transformado num modelo escrito na linguagem RDF. Escolhemos o RDF pois, no início do nosso projeto, era grande e cada vez mais crescente a sua utilização para descrever informações semânticas na *Web*, através da padronização para definição e



utilização de metadados. Esse processo que transfere as ontologias representadas graficamente pelo OntoEditor para um documento RDF padrão é chamado **Tradução** em nosso projeto.

A Tradução das ontologias consiste num processo que tanto permite que os **grafos** sejam transformados para **documentos RDF** como permite o processo inverso, assim o usuário pode posteriormente manipular graficamente a ontologia armazenada no banco. A FIGURA 3.3 ilustra o processo da Tradução.



**FIGURA 3.3** Visão geral do processo de Tradução

Nesse processo temos a tradução de informações entre dois modelos: o modelo abstrato (detalhado na seção 3.3) que concebemos para representar graficamente as ontologias no OntoEditor e o modelo RDF. Cada elemento do grafo que representa uma ontologia será traduzido para um elemento correspondente do esquema e/ou linguagem RDF e vice-versa.

A correlação entre os dois modelos foi feita de forma bem direta uma vez que a própria concepção do modelo abstrato de dados do OntoEditor fora baseado no modelo de dados RDF associado ao esquema RDF. Para facilitar a geração dos documentos em RDF, após a correlação, fora utilizado uma *api* chamada *Jena*, disponibilizada pela HP, cuja função é gerar documentos no padrão RDF [HP - Jena]. Maiores detalhes sobre a forma como essa correlação foi implementada com a ajuda da *Jena* serão descritos no Capítulo IV, dentre as demais funcionalidades.

### 3.2.1.3 Armazenamento e Recuperação de Ontologias no OntoEditor

Após obter a ontologia especificada em RDF, a partir do grafo definido pelo usuário, o OntoEditor poderá armazenar essa ontologia num banco de dados (que pode estar em alguma máquina que sirva como servidor de dados na intranet) ou exportar essa ontologia para um

arquivo texto convencional (que poderia ser armazenado localmente na máquina que está executando a aplicação).

As ontologias são armazenadas juntamente com as informações descritas nas propriedades gerais da ontologia (nome, autor, domínio etc). Estas propriedades são metadados que serão responsáveis pela identificação da ontologia que será armazenada.

Uma vez que essas ontologias estejam armazenadas, poderemos recuperá-las, tanto do banco de dados como dos arquivos, e manipulá-las, reeditando-as, por exemplo, em nosso editor.

Além disso, com as ontologias armazenadas num banco de dados, podemos viabilizar a construção de num repositório com muitas ontologias que disporia dos serviços disponibilizados por SGBDs, tais como: concorrência, tolerância a falhas, segurança, dentre outros.

#### **3.2.1.4 Consultas OntoEditor**

Uma das atividades mais executadas e requisitadas quando dispomos de uma base de informação é a consulta. Por isso, o OntoEditor disponibiliza aos seus usuários um menu de consultas às ontologias armazenadas no banco de dados.

Dois tipos de consulta são disponibilizados e eles foram definidos e escolhidos com a intenção de permitir um razoável nível de flexibilidade aos seus usuários no momento em que os mesmos desejem manipular a sua base de informação. Os tipos são: Consulta por Propriedades das Ontologias e Consulta por Conteúdo.

Na **Consulta por Propriedades das Ontologias**, o usuário busca as suas ontologias através das informações disponibilizadas nas propriedades gerais das ontologias, ou seja seus metadados. Essas características da ontologia são informadas obrigatoriamente pelo usuário ao armazenar qualquer ontologia (ela já foi citada anteriormente, mas a detalharemos melhor a seguir). Logo, o usuário informa as características e a ferramenta busca todas as ontologias com tais características e, caso elas existam, a ferramenta as devolve como resposta a sua

consulta. Por exemplo, o usuário poderia fazer uma consulta por todas as ontologias cujo domínio fosse Universidade.

No caso da **Consulta por Conteúdo**, o usuário poderá consultar as ontologias através de palavras-chaves que poderão fazer parte do conteúdo de alguma ou de várias ontologias que estão armazenadas no banco de dados. Essa consulta é realizada no conteúdo das ontologias, ou seja, nas ontologias especificadas em RDF. Para exemplificar, poderíamos considerar uma consulta em que o usuário buscaria por todas as ontologias que possuíssem no seu conteúdo informações sobre o autor José de Alencar.

A primeira consulta é um tipo tradicional, realizado de forma bem simples com a ajuda de qualquer sistema gerenciador de banco de dados relacional. Já a segunda, age de uma forma diferente da convencional. A Consulta por Conteúdo irá buscar informações dentro de um documento RDF e isso a diferencia muito da primeira consulta citada. Para executar esse tipo de consulta, o OntoEditor teve que agregar, sob um segundo plano, uma funcionalidade específica que é a consulta a dados XML.

### 3.3 Modelo Conceitual

Descreveremos nessa seção como foi concebido o modelo abstrato de dados utilizado para representarmos as ontologias no OntoEditor. O entendimento desse modelo é de fundamental importância para que se torne evidente o uso de todas as outras funcionalidades propostas pelo OntoEditor.

Para o OntoEditor, uma ontologia é composta basicamente pelos seguintes elementos: Termos, Propriedades e Relacionamentos; e é identificada por metadados que são representados pela ferramenta como Propriedades da Ontologia (Nome, Autor, Domínio e Data de Criação).

Logo, para que os usuários do OntoEditor possam definir suas ontologias, além de informar as propriedades da ontologia, requisitadas pela ferramenta, devem informar, conforme suas necessidades:

- **Termos** que representarão os conceitos-chave extraídos de um dado domínio de conhecimento. Cada termo possui atributos, nome (obrigatório) e descrição (não-obrigatório), que identificam o termo. Além da definição desses atributos, o OntoEditor permite que para cada Termo sejam adicionadas propriedades e que tanto as propriedades quanto os termos possam também ser excluídos.
- **Propriedades** devem representar quaisquer outros atributos, definidos pelo usuário, que ajudarão a descrever e identificar, ainda mais, o termo que está sendo descrito. Cada propriedade possui os atributos nome e tipo do valor (obrigatórios). Por exemplo, poderíamos definir uma propriedade para um termo Pessoa da seguinte forma: nome = idade e tipo do valor = inteiro. Ao definir uma propriedade, ela deverá estar atribuída a um determinado termo. Uma vez feito isso, essa mesma propriedade poderá ser associada a outros termos de forma independente, tornando-se, assim, comum a mais de um termo. As propriedades que foram criadas podem ser alteradas e excluídas.
- **Relacionamentos** entre termos. Um termo poderá se relacionar ou não com um ou mais termos. Cada relacionamento possui apenas um atributo: o nome. A ferramenta disponibilizará alguns relacionamentos como sugestão para os usuários, mas ela será capaz de criar novos relacionamentos. Dentre os relacionamentos sugeridos pela ferramenta, existe o relacionamento **subclasse**. Ao definir que dois termos estão associados pelo relacionamento subclasse, dizemos que um termoA é subclasse do termoB e que o termo subclasse herda todas as propriedades da superclasse. Os relacionamentos também podem ser excluídos.

A FIGURA 3.4 mostra o modelo conceitual do OntoEditor. A partir dele, podemos extrair as informações descritas a seguir.

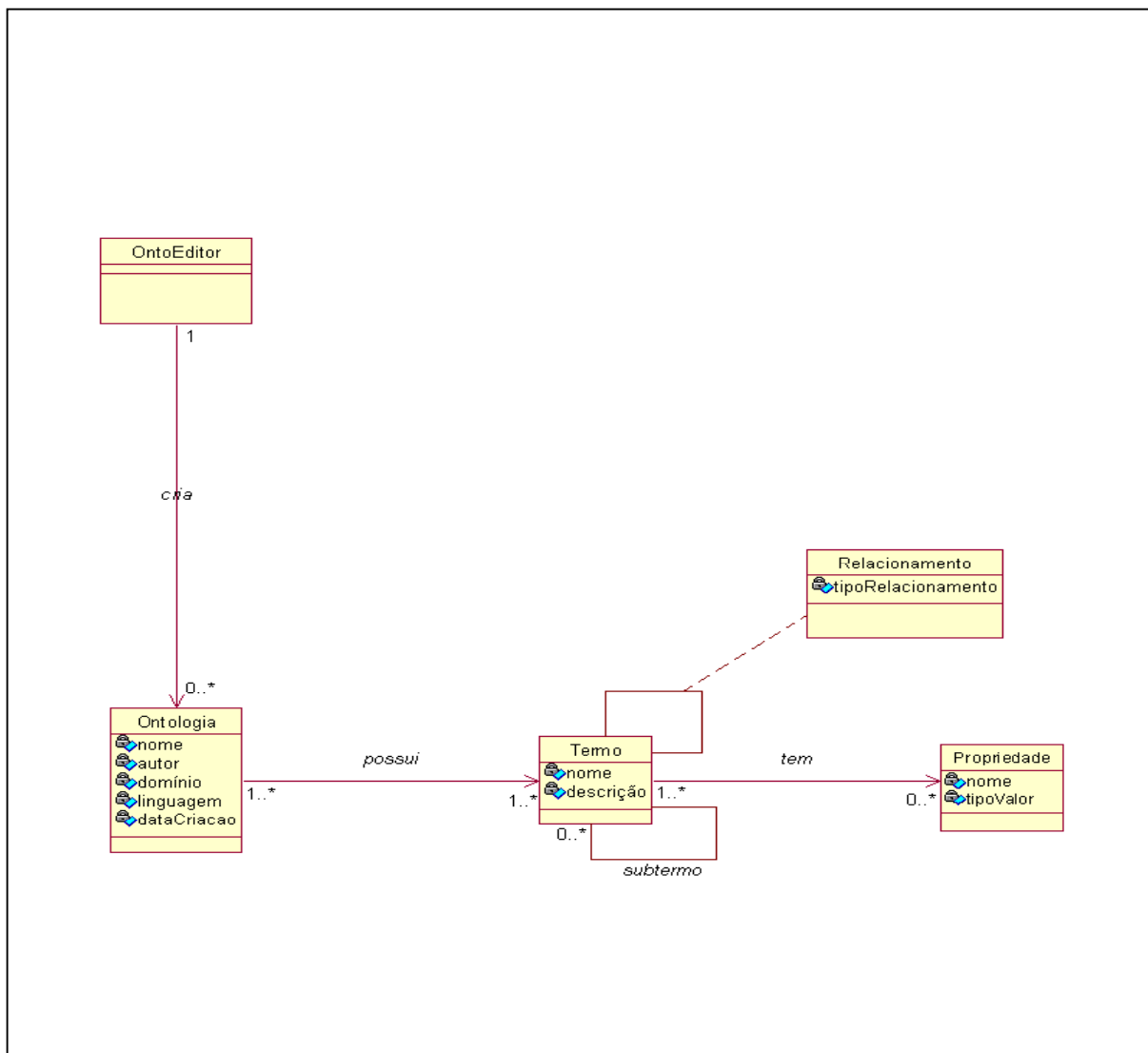
Uma ontologia possui os seguintes atributos: Código, Nome, Autor, Domínio e Data de Criação.

Uma ontologia é composta por um ou vários Termos. Cada Termo possui os atributos nome e descrição.

Um termo pode ou não ser formado por outros termos e nesse caso ele é considerado uma subclasse desse(s) termo(s). Quando um termo é subclasse de outro, ele herda todas as propriedades do termo-mãe.

Cada Termo pode possuir ou não propriedades. Cada Propriedade possui os atributos nome e tipo da propriedade.

Um Termo pode se relacionar com outros Termos. Para cada relacionamento é definido o seu nome como atributo.



**FIGURA 3.4 Modelo Conceitual do OntoEditor**

### 3.4 Levantamento de Requisitos

A análise e o projeto do OntoEditor avaliou um único tipo de usuário para ferramenta:

- Usuários que desejam descrever ontologias para representar semanticamente um domínio de conhecimento qualquer.

Por meio do diagrama de casos de uso mostrado abaixo, descreveremos como o usuário (**Ator**) poderá interagir com o OntoEditor.

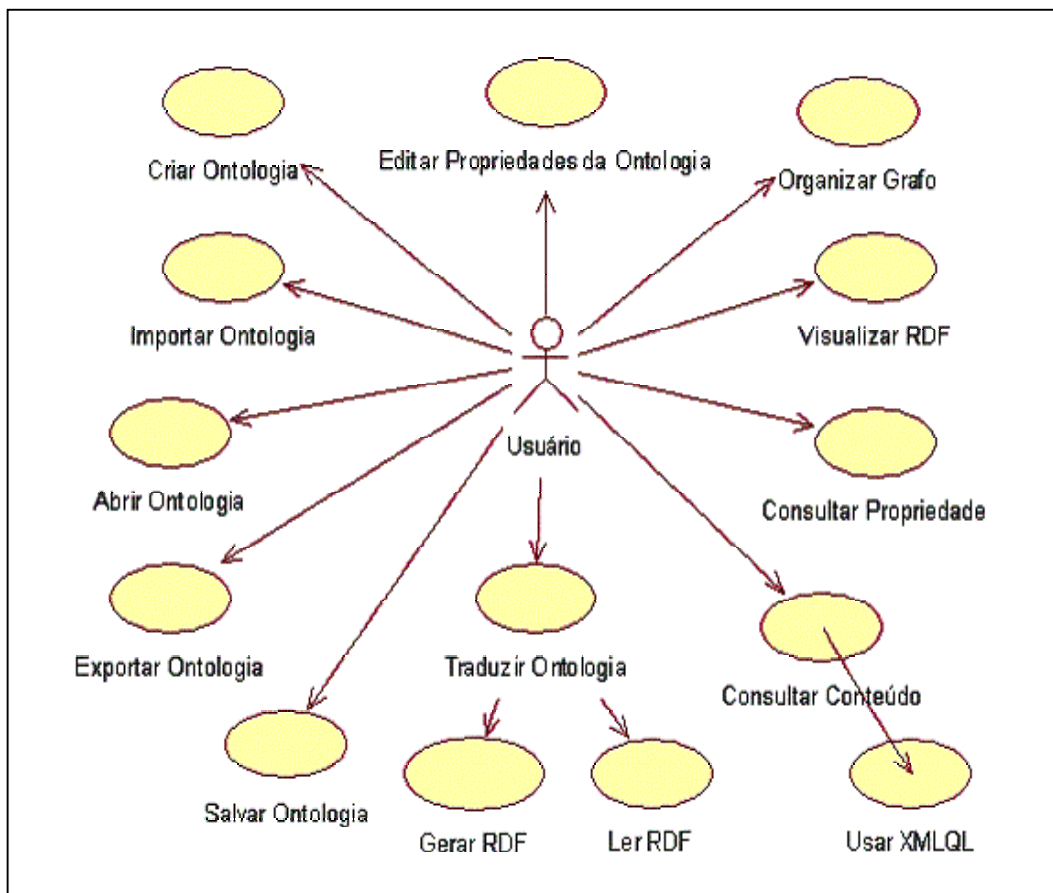


FIGURA 3.5 Diagrama de Casos de Uso

#### 3.4.1 Requisitos Funcionais do OntoEditor

Caso de Uso	Descrição
[OTE1] Criar Ontologia	Permite ao usuário criar ontologias, sob a forma de grafos, com os termos que irão

	compor o domínio da ontologia a ser especificada, bem como seus possíveis relacionamentos e propriedades.
[OTE2] Visualizar RDF	Permite que o usuário visualize o documento RDF que é gerado a partir da ontologia que está representada na interface da ferramenta.
[OTE3] Editar Propriedades da Ontologia	Permite que o usuário informe as propriedades requisitadas pela ferramenta para armazenamento no banco de dados.
[OTE4] Salvar Ontologia	Permite que o usuário armazene as ontologias criadas na ferramenta num banco de dados relacional.
[OTE5] Abrir Ontologia	Permite que o usuário abra qualquer ontologia que o OntoEditor armazenou no banco de dados. É disponibilizada, para o usuário, uma lista com todas as ontologias armazenadas no banco e a partir das informações dos metadados (propriedades da ontologia), a ontologia é escolhida e será representada na interface da ferramenta.
[OTE6] Exportar Ontologia	Permite que o usuário exporte a ontologia criada graficamente para um arquivo texto

	contendo a ontologia especificada em RDF.
[OTE7] Importar Ontologia	Permite que o usuário informe o nome de um arquivo, gerado pela ferramenta, e a partir da ontologia descrita no arquivo em RDF, a ferramenta disponibiliza a representação gráfica da mesma.
[OTE8] Consultar Propriedades da Ontologia	Permite que o usuário consulte as ontologias que estão armazenadas no banco de dados, através das propriedades da ontologias que foram requisitadas pela ferramenta quando a ontologia foi armazenada.
[OTE9] Consultar Conteúdo da Ontologia	Permite que o usuário consulte as ontologias que estão armazenadas no banco de dados, através de alguma palavra-chave que ele deverá informar. Essa palavra será procurada no conteúdo (documento RDF) de todas as ontologias que estão armazenadas no banco de dados. Utiliza o caso de uso [QL1].
[OTE10] Tradução das Ontologias	Permite que as ontologias geradas graficamente pelo usuário sejam traduzidas para linguagem RDF e vice-versa. Utiliza os casos de uso [J1] e [J2].



[OTE11] Organização do Grafo	Possibilita melhorar a visualização da ontologia, organizando a disposição gráfica dos elementos na interface do editor.
------------------------------	--

**TABELA 3.1 Requisitos Funcionais do OntoEditor**

### 3.4.2 Requisitos Funcionais da Jena

Caso de Uso	Descrição
[J1] Gerar RDF	Utilizado pelo caso de uso [OTE10], permite que seja gerado um documento nos padrões da linguagem RDF.
[J2] Ler RDF	Utilizado pelo caso de uso [OTE10], permite que um documento RDF seja lido, extraíndo as suas sentenças.

**TABELA 3.2 Requisitos Funcionais da Jena**

### 3.4.3 Requisitos Funcionais do XML-QL

Caso de Uso	Descrição
[QL1] Consultar Conteúdo	Permite consultas ao conteúdo de documentos XML. Utilizado pelo caso de uso [OTE9].

**TABELA 3.3 Requisitos Funcionais do XML-QL**

### 3.5 Planejamento dos Incrementos

Em nosso projeto, adotamos o desenvolvimento iterativo e incremental [Larman 2000].

Cada ciclo (iteração) trata um conjunto relativamente pequeno de requisitos, realizado através da análise, do projeto, da construção e do teste. A conclusão de cada ciclo permitirá que a ferramenta cresça incrementalmente [Larman – 2000].

Definimos dois incrementos para construção da ferramenta. O planejamento dos incrementos foi baseado na relevância das funcionalidades, às quais o OntoEditor se propunha.

O primeiro incremento, ou primeiro protótipo do OntoEditor, foi planejado para atender a principal finalidade do nosso projeto que seria a construção de um editor gráfico para criar ontologias. Consideramos tal finalidade primordial, visto que ela é o alicerce da arquitetura do projeto e, a partir do seu desenvolvimento, iremos nos deparar com diversas questões relacionadas às dificuldades de implementação que nos permitirá definir posteriormente todas as outras funcionalidades da ferramenta, abrangendo o que foi disposto na arquitetura.

Nesse incremento também foi desenvolvida a Tradução de ontologias que é o processo intermediário entre a representação gráfica das ontologias, definida pelo usuário ao utilizar o OntoEditor, e a representação da ontologia em RDF que é como as ontologias são armazenadas no banco de dados.

Concluimos esse incremento desenvolvendo as funcionalidades de exportar e importar as ontologias representadas em RDF para arquivos textos convencionais.

Como resultado desse incremento, temos um editor gráfico de ontologias com opção de exportá-las para arquivos texto, possibilitando que tais ontologias sejam recuperadas posteriormente desses arquivos e reeditadas na interface do editor.

O segundo incremento contemplou os requisitos de funcionalidade exigidos para a comunicação do OntoEditor com o banco de dados, englobando a definição de toda estrutura lógica de dados, bem como a forma de armazenamento, recuperação e manipulação das ontologias armazenadas.

Nesse incremento foi implementada a comunicação do Ontoeditor com o servidor de banco de dados, sendo, conseqüentemente, disponibilizadas as funcionalidades de armazenamento, recuperação, manipulação e consultas às ontologias criadas pelos usuários da ferramenta. Tal comunicação foi viabilizada via JDBC.

O esquema de banco de dados foi definido para atender o armazenamento tanto da ontologia propriamente dita, armazenada sob a descrição de um documento RDF, como das propriedades de cada ontologia, composta por: seu nome, seu domínio, seu autor e sua data de criação.

Como parte desse incremento, foi dada uma atenção especial a funcionalidade das consultas oferecidas pelo OntoEditor. O OntoEditor disponibiliza dois tipos de consulta: a consulta por propriedade, onde o usuário escolhe uma ou mais propriedades e busca as ontologias que possuam a(s) mesma(s), e a consulta por conteúdo, onde os usuários buscam ontologias que possuam uma determinada palavra-chave no seu conteúdo.

A conclusão do desenvolvimento do segundo incremento, tornou o OntoEditor apto para disponibilizar aos seus usuários mecanismos para recuperar ontologias, pesquisando e filtrando a sua base de dados de acordo com as suas necessidades.

### **3.6 Estudo de Caso**

Mostraremos a seguir um estudo de caso onde utilizaremos o OntoEditor para definir uma ontologia para o domínio Universidade.

A ontologia será composta por um conjunto de termos semanticamente coesos que, além de possuir atributos com propriedades específicas, podem se relacionar de diversas

formas, conforme as necessidades requeridas pelo domínio de conhecimento que está sendo especificado.

A ontologia deverá especificar que a Universidade:

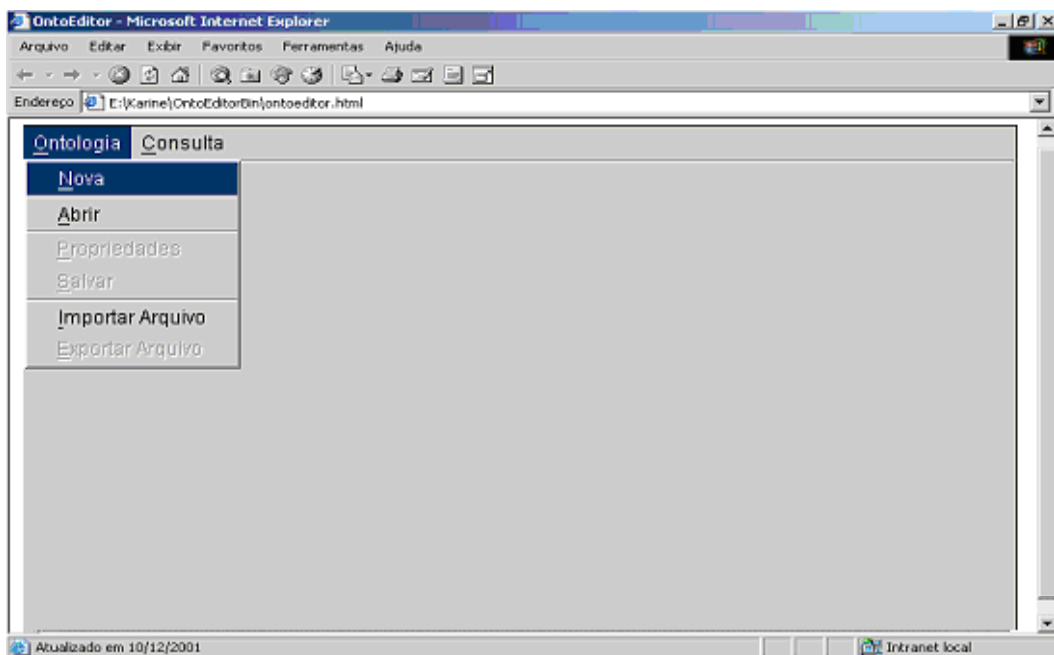
- Possui uma localização que pode ser indicada por uma cidade, um estado, e um país.
- Oferece cursos. Os cursos alocam várias pessoas que podem ser empregados da universidade ou alunos. Essas pessoas podem publicar teses, artigos e livros.
- Tem professores e assistentes como empregados.
- Possuem alunos de graduação, alunos de mestrado e alunos de doutorado. Os alunos de doutorado podem ser assistentes da universidade.

Abaixo são descritos os passos principais que foram realizados para construção da ontologia apresentada.

- Passo 1: Acessar o OntoEditor

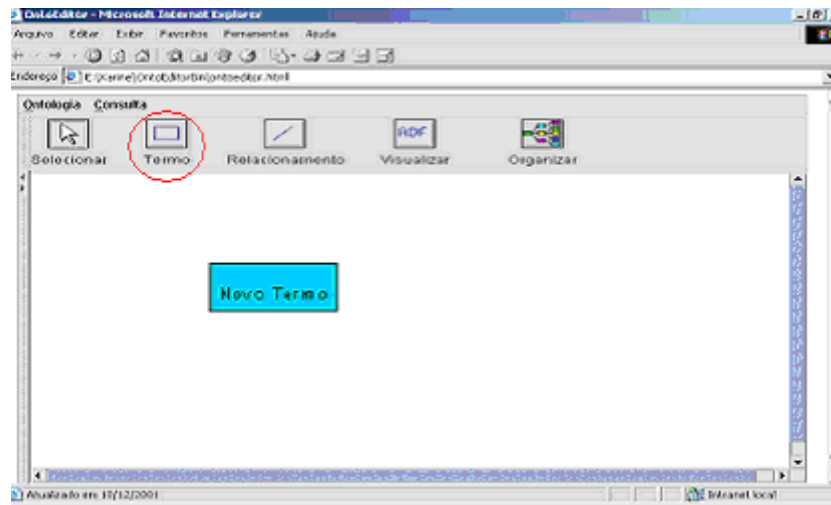
O usuário acessa a página que contém a ferramenta através de um link.

- Passo 2: O usuário deve escolher a opção Nova no menu Ontologia.



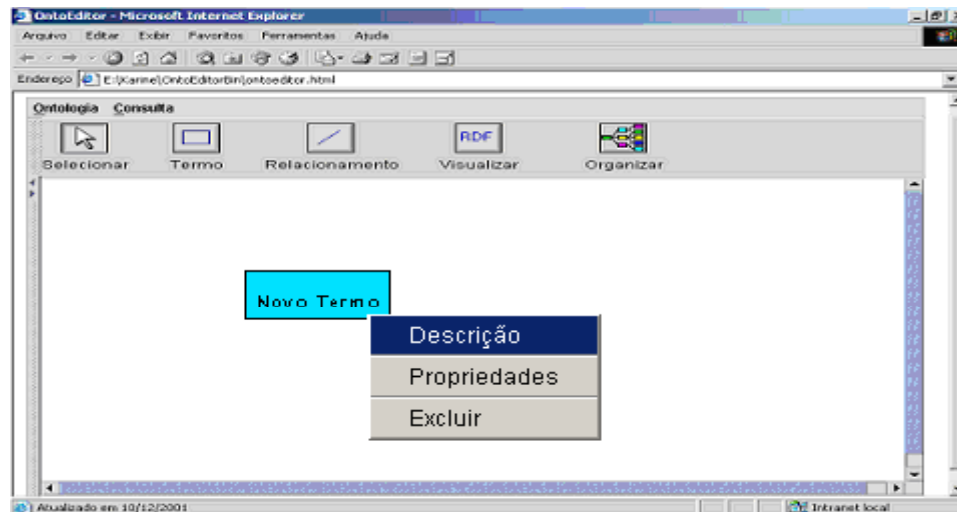
**FIGURA 3.6** Criando uma nova ontologia

- Passo 3: O usuário começa a definir a ontologia utilizando o botão Termo para iniciar a representação gráfica.

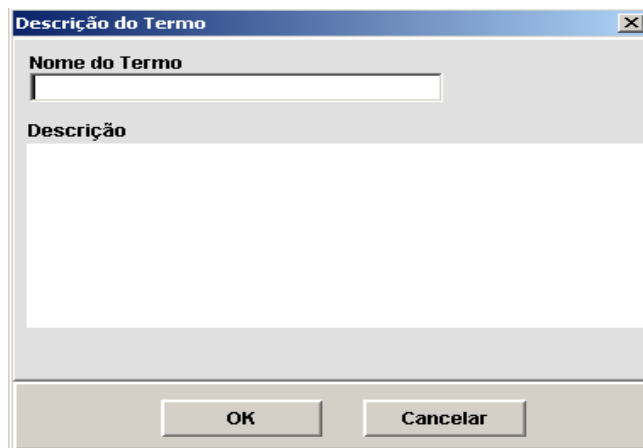


**FIGURA 3.7** Definição de termos

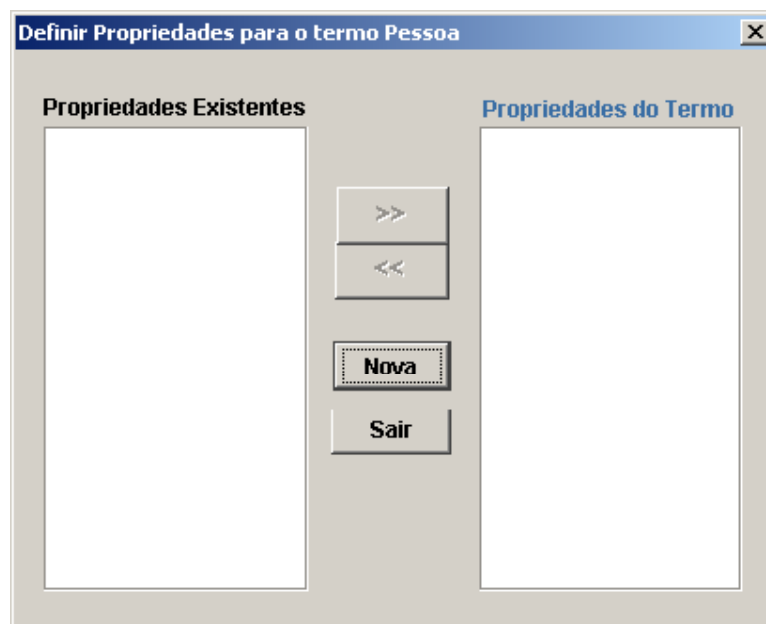
Após criar um Termo, o usuário deverá escolher o botão Selecionar e clicar com o botão direito do mouse em cima do Termo para definir algumas características dele, como: o nome (obrigatório), descrição e propriedades.



**FIGURA 3.8** Definindo características do termo

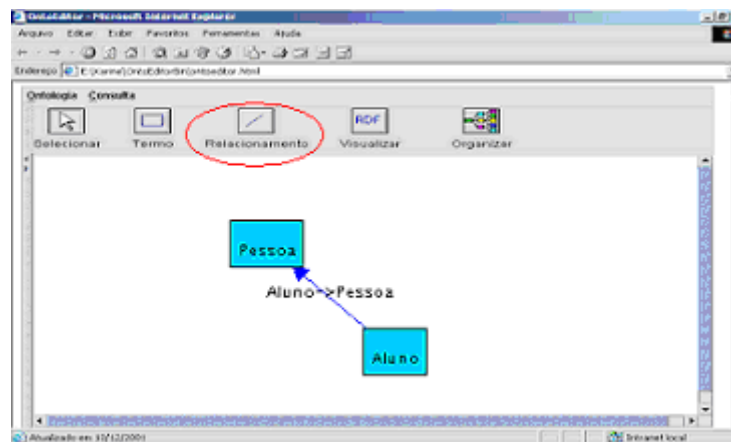


**FIGURA 3.9** Definindo o nome e descrição do termo



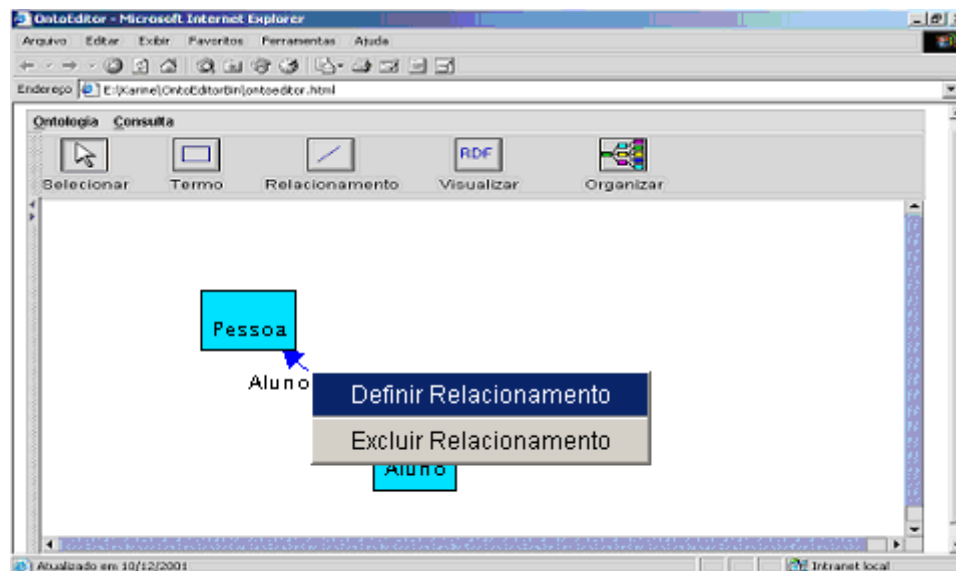
**FIGURA 3.10** Interface para definir novas propriedades para o termo ou adicionar ao termo propriedades já existentes

- Passo 4: Através do botão Relacionamento, o usuário define os possíveis relacionamentos entre os termos.

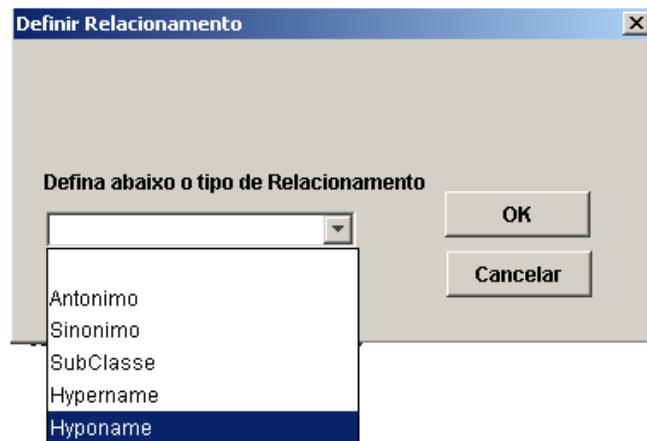


**FIGURA 3.11** Definição de relacionamentos entre termos

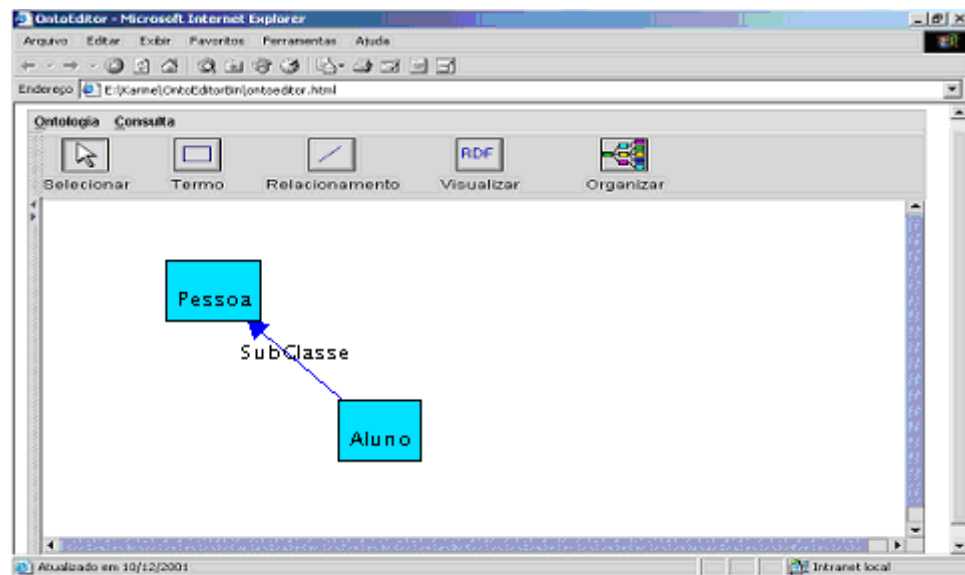
Para informar o nome do relacionamento, que é obrigatório, o usuário deve escolher a opção Selecionar e pressionar o botão direito do mouse em cima do relacionamento.



**FIGURA 3.12** Definindo o nome do relacionamento ou excluindo relacionamento



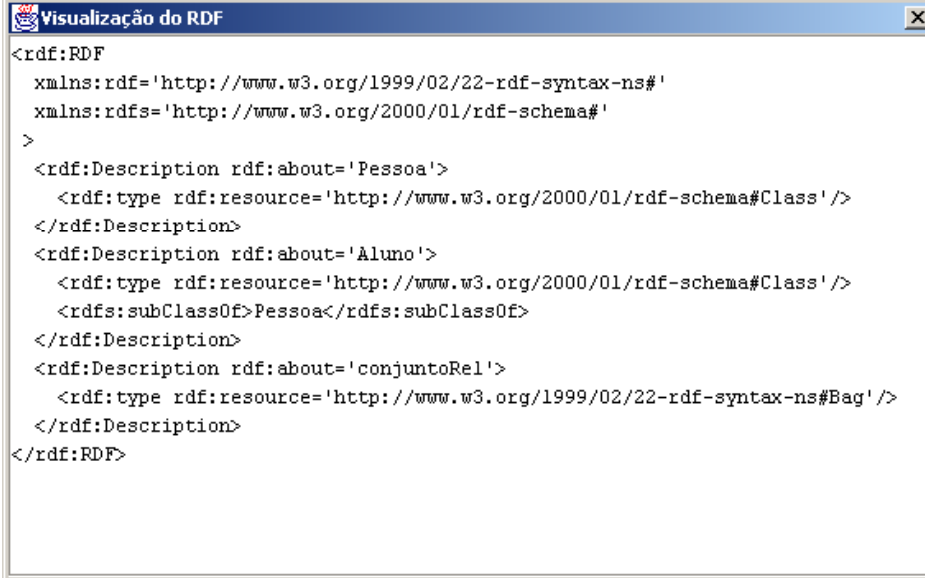
**FIGURA 3.13** Definição do nome do relacionamento



**FIGURA 3.14** Representação dos termos após a definição de um relacionamento



- Passo 5: Ao concluir a representação da ontologia, o usuário poderá visualizar o RDF gerado a partir do grafo.



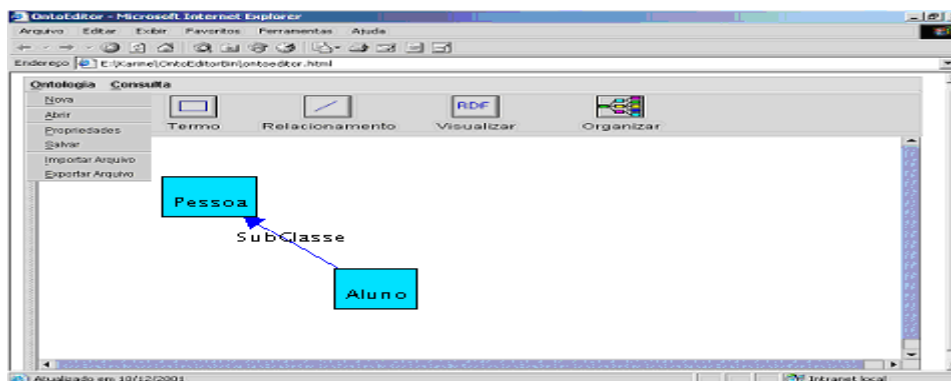
```

<rdf:RDF
  xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:rdfs='http://www.w3.org/2000/01/rdf-schema#'
>
  <rdf:Description rdf:about='Pessoa'>
    <rdf:type rdf:resource='http://www.w3.org/2000/01/rdf-schema#Class' />
  </rdf:Description>
  <rdf:Description rdf:about='Aluno'>
    <rdf:type rdf:resource='http://www.w3.org/2000/01/rdf-schema#Class' />
    <rdfs:subClassOf>Pessoa</rdfs:subClassOf>
  </rdf:Description>
  <rdf:Description rdf:about='conjuntoRel'>
    <rdf:type rdf:resource='http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag' />
  </rdf:Description>
</rdf:RDF>

```

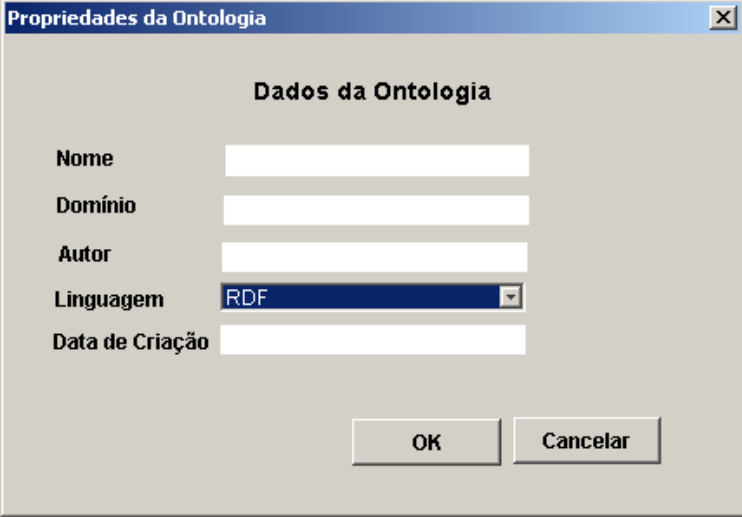
**FIGURA 3.15** Visualização do RDF gerado a partir da representação mostrada na **FIGURA 3.14**

O usuário também pode salvar essa ontologia no banco de dados e/ou exportá-la para um arquivo texto. Essas duas últimas opções estão disponibilizadas no menu Ontologia.



**FIGURA 3.16** Menu Ontologia do OntoEditor

Vale lembrar que para utilizar a opção Salvar, o usuário deverá informar, em Propriedades do menu Ontologia, as informações que são requisitadas pelo OntoEditor para efetivar o armazenamento da ontologia.



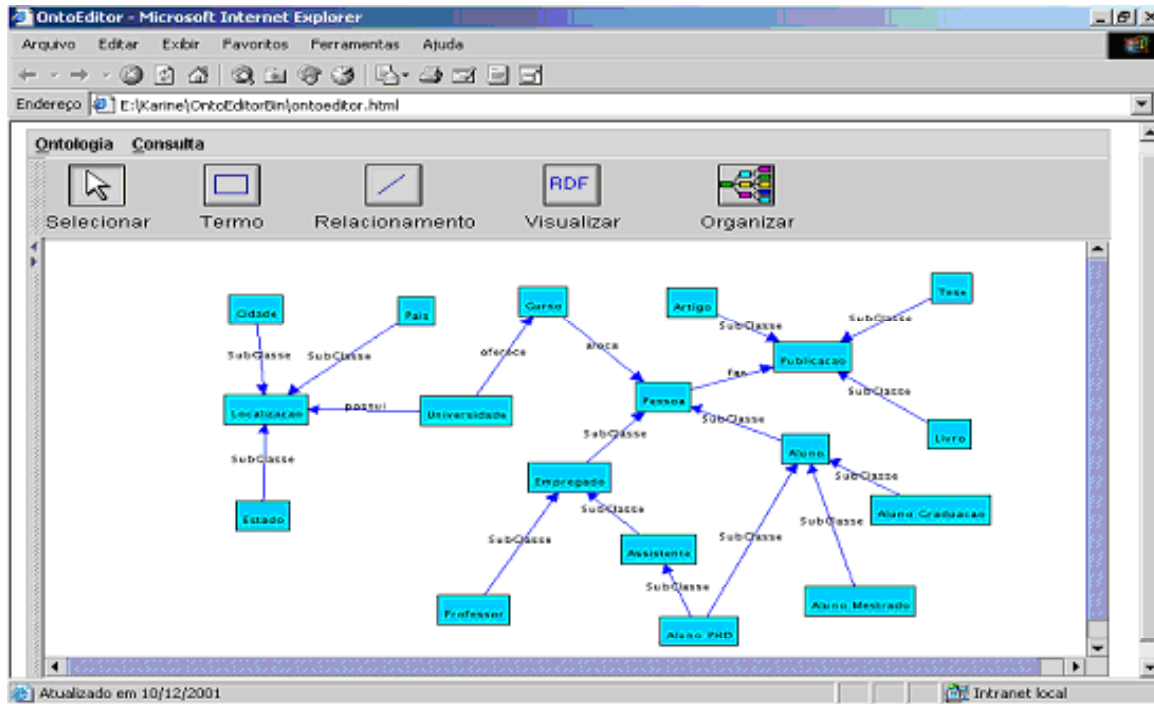
The image shows a dialog box titled "Propriedades da Ontologia" with a close button in the top right corner. The main content area is titled "Dados da Ontologia" and contains the following fields:

- Nome**: A text input field.
- Domínio**: A text input field.
- Autor**: A text input field.
- Linguagem**: A dropdown menu with "RDF" selected.
- Data de Criação**: A text input field.

At the bottom right of the dialog are two buttons: "OK" and "Cancelar".

**FIGURA 3.17** Definindo as Propriedades da Ontologia

A FIGURA 3.18 ilustra uma ontologia definida através do OntoEditor para o domínio universidade.



**FIGURA 3.18 Ontologia Universidade**

### 3.7 Conclusão

Foram descritos nesse capítulo todos os aspectos envolvidos com o projeto da ferramenta OntoEditor. Mostramos, basicamente, o que é o OntoEditor, como ele foi idealizado e projetado e a quais funcionalidades ele se propõe atender.

Fizemos uma abordagem geral da ferramenta, através da ilustração e descrição da sua arquitetura, detalhando-a, posteriormente, para descrever a forma como a mesma foi concebida, quais os seus propósitos e quais as funcionalidades que ela se propunha a disponibilizar. Concluímos com uma descrição de como as funcionalidades oferecidas pelo OntoEditor seriam disponibilizadas aos seus usuários, ou seja, como elas seriam apresentadas. Utilizamos para tanto um estudo de caso, onde o OntoEditor foi usado para definir uma ontologia para o domínio Universidade.

## Capítulo IV - IMPLEMENTAÇÃO DO ONTOEDITOR

De acordo com o projeto arquitetural do OntoEditor, descrito no capítulo III, apresentaremos nesse capítulo os aspectos de implementação utilizados na construção dessa ferramenta.

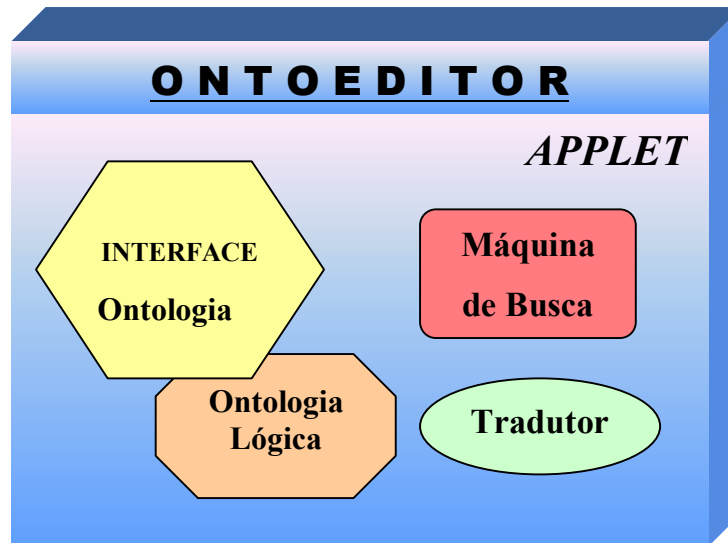
Iniciaremos a descrição da implementação do OntoEditor a partir da Camada de Aplicação. Descreveremos suas principais características de implementação, associadas as suas respectivas funcionalidades (tradução e validação, consultas, dentre outras), bem como enfatizando todas as *api's* e *parsers* usados no nosso projeto.

Em seguida, descreveremos o esquema lógico de dados adotado e concluiremos mostrando como o OntoEditor acessa e manipula sua base de dados. Além disso, discorreremos sobre como foram desenvolvidas as funcionalidades de consulta oferecidas pela ferramenta.

### 4.1 A Camada de Aplicação

A arquitetura do OntoEditor proporciona uma visão da ferramenta em alto nível. Através dele, entendemos que a ferramenta OntoEditor, ou seja, toda a camada de aplicação, está representada por um *Applet Java*.

A FIGURA 4.1 ressalta as principais características de implementação referentes às funcionalidades disponibilizadas pelo OntoEditor. Todas essas funcionalidades (interface gráfica, tradução, consultas etc) foram implementadas a partir de um *Applet Java*.



**FIGURA 4.1** Visão geral das principais características de implementação da camada de aplicação

#### 4.1.1 Applets Java

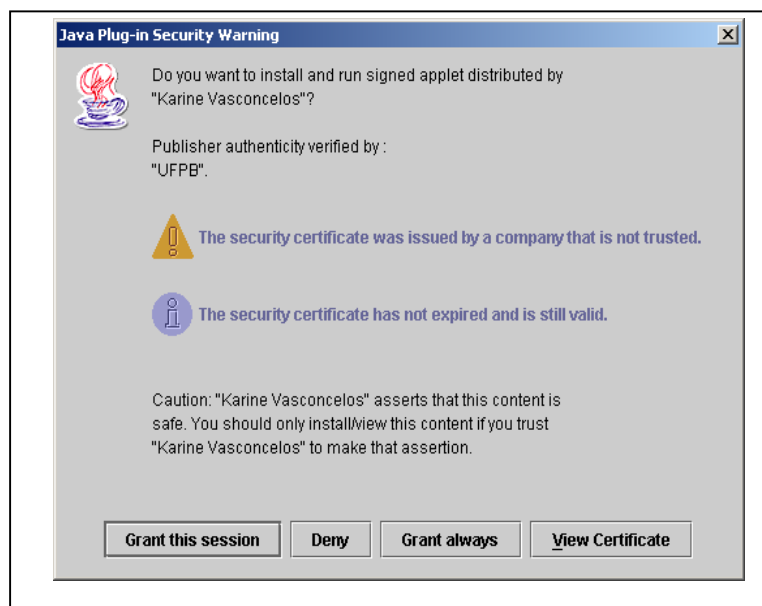
O *applet* é uma solução computacional, implementada na linguagem de programação Java, para construção de aplicações *Web*. Os *Applets Java* ficam hospedados para que possam ser obtidos e executados pelo navegador *Web* (*browser*) através de uma URL. Caso o *applet* esteja implementado numa versão do Java diferente da máquina virtual do *browser*, é necessário utilizar o Java *plug-in* no navegador *Web*. Este *plug-in* irá oferecer o ambiente apropriado que é requerido pelo *applet*. A versão do Java utilizada no nosso trabalho foi a 1.3.1\_02 [Sun - A].

Com o objetivo de proteger a estação do usuário, a máquina virtual Java restringe as operações que um *applet* pode executar (*sandbox*). As principais restrições de segurança são:

- Um *applet* não pode ler ou gravar arquivos na máquina em que está sendo executado.
- Um *applet* não pode fazer conexões de rede, exceto com o servidor *Web* onde ele está armazenado.
- Um *applet* não pode executar outras aplicações da máquina em que está sendo executado.

Apesar de possuir as restrições citadas acima, o *applet* Java possui um artifício para permitir que o seu desenvolvedor possa contorná-las, conforme a necessidade. Esse artifício é conhecido como *applet com assinatura*.

Quando um *applet* com assinatura é executado, surgem na tela, informações (FIGURA 4.2) sobre ele tais como: quem o implementou, alertas dizendo que ele poderá ler ou gravar arquivos na máquina . Logo, fica a critério do usuário autorizar ou não a execução da aplicação (*applet*).



**FIGURA 4.2 Mensagem de segurança do Java *Plug-in***

Adotamos essa idéia para construir o OntoEditor por considerarmos que o *applet* satisfaz todas as necessidades requisitadas pelas funcionalidades da ferramenta. Portanto, ao utilizamos o *applet com assinatura*, no nosso projeto, possibilitamos que os usuários do OntoEditor possam, após a criação das ontologias, salvá-las num banco de dados relacional ou exportá-la para um arquivo texto.

Não foi desconsiderado que, ao usar um *applet* com assinatura, o OntoEditor estará oferecendo flexibilidades que podem ameaçar a segurança das informações por ele manipuladas. Por isso, ressaltamos que o OntoEditor é uma ferramenta para manipular ontologias na *Web* dentro de um escopo delimitado, ou seja, uma intranet, onde existe uma comunidade de usuários. Logo, as ontologias irão ser compartilhadas por usuários com

interesses em comum e os mesmos serão responsáveis pelas informações manipuladas pela aplicação (*applet*).

#### **4.1.2. Interface – Ontologia Gráfica**

Na definição do nosso projeto, percebemos que todas as ferramentas que existiam (descritas no Capítulo 2), com o mesmo propósito do OntoEditor, disponibilizavam, através das suas interfaces, a criação de ontologias sob a forma de **Árvore**.

A árvore é um tipo especial de grafo onde cada descendente possui um único ancestral. Numa interface gráfica, esse tipo de estrutura simplifica a criação e visualização das ontologias, sendo que a definição dos termos, relacionamentos e propriedades se tornam atividades bastante simples de serem executadas.

Entretanto, ao aplicar esse tipo de estrutura a algumas representações de conhecimento, refletindo situações reais, percebemos que esse tipo de grafo (árvore) oferecia algumas limitações como podemos citar o caso clássico da representação da herança múltipla – muito comum no mundo real e, conseqüentemente, na grande maioria das representações.

Então, em nosso projeto, decidimos implementar a interface gráfica do OntoEditor com suporte para criar ontologias sob a forma de grafos direcionados. Com isso, nós conservamos a simplicidade, objetividade e as facilidades de visualização de um editor gráfico baseado em grafos, ampliando significativamente as possibilidades para o usuário criar suas ontologias espelhando um conhecimento cada vez mais próximo da realidade.

##### **4.1.2.1. OpenJGraph - API Gráfica**

Para implementação da interface gráfica do OntoEditor, foi utilizada, além da *API Swing Java*, uma biblioteca específica de código aberto para componentes gráficos. Essa biblioteca é chamada de *OpenJGraph* e é disponibilizada gratuitamente na *Web* [OpenJGraph – 2000].

A *OpenJGraph* foi utilizada como suporte para auxiliar na construção de alguns componentes da interface do OntoEditor, como Termos, Relacionamentos . Ela é composta por vários pacotes e a FIGURA 4.3 mostra os pacotes que foram utilizados para construção da interface gráfica do OntoEditor.

<b><u>OntoEditor - Implementação</u></b>
<pre>import salvo.jesus.graph.*; import salvo.jesus.graph.visual.*; import salvo.jesus.graph.visual.layout.*; import salvo.jesus.graph.visual.drawing.*;</pre>

**FIGURA 4.3 Implementação OntoEditor/Interface Gráfica**

### 4.1.3 Ontologia Lógica

Para o usuário, uma ontologia é criada através de uma representação gráfica de termos, suas propriedades e seus relacionamentos. Para o OntoEditor, logicamente, uma ontologia é composta por:

- Um grafo – que irá conter toda a ontologia representada graficamente; e
- Suas propriedades (Nome, Domínio, Autor e Data de Criação).

Assim, definimos o esquema lógico do nosso projeto, visando a melhor forma para que as ontologias fossem armazenadas numa base de dados e pudessem ser posteriormente manipuladas e compartilhadas. A partir dessas premissas, mostraremos, a seguir, como o esquema de dados foi definido.

Lembramos que o OntoEditor também permite que as ontologias criadas possam ser exportadas para um arquivo, como também importar as ontologias do arquivo para o editor. Esses arquivos criados com o OntoEditor serão arquivo de texto padrão, onde teremos ontologias especificadas em RDF, com uma extensão .ote, particular da ferramenta.



## 4.2 A Camada de Dados

A estrutura do banco de dados é formada por uma tabela de Ontologias com as seguintes informações: Código, Nome, Autor, Domínio, Data de Criação e RDF.

O primeiro item (Código) é um número seqüencial que representa a chave da tabela.

Do segundo ao quinto item (Nome, Autor, Domínio e Data de Criação) teremos características da ontologia que está sendo definida e essas informações serão requisitadas pela ferramenta todas as vezes que uma ontologia for salva pela primeira vez. Esses itens identificarão as Propriedades da Ontologia.

O sexto item, RDF, é a ontologia criada graficamente por um usuário da ferramenta, representada na linguagem RDF (Processo de Tradução – ver a seguir).

Os tipos dos campos estão descritos abaixo:

<b>Campo</b>	<b>Tipo</b>	<b>Tamanho</b>
Código	Int	-
Nome	Varchar	30
Autor	Varchar	40
Domínio	Varchar	30
Data de Criação	Datetime	-
RDF	Image	-

**TABELA 4.1 Tabela de Campos que compõem as Ontologias**

O campo RDF é um campo especial. Como já foi dito, ele irá conter um texto em RDF que representará a ontologia criada graficamente pelo usuário. Ele possuirá informações armazenadas sob uma estrutura semântica e isso possibilitará que tais informações sejam manipuladas e compartilhadas não só por um grupo de pessoas, mas, também, por aplicações, disponibilizando recursos para que as máquinas possam entendê-las e processá-las.

Para armazenar os dados gerados pelo OntoEditor, foi utilizado o servidor de banco de dados relacional *SQLServer 7.0* da *Microsoft* [Microsoft - SQLServer].

### 4.3 Processo de Tradução

Com o OntoEditor, os usuários podem criar, gravar, recuperar e editar ontologias.

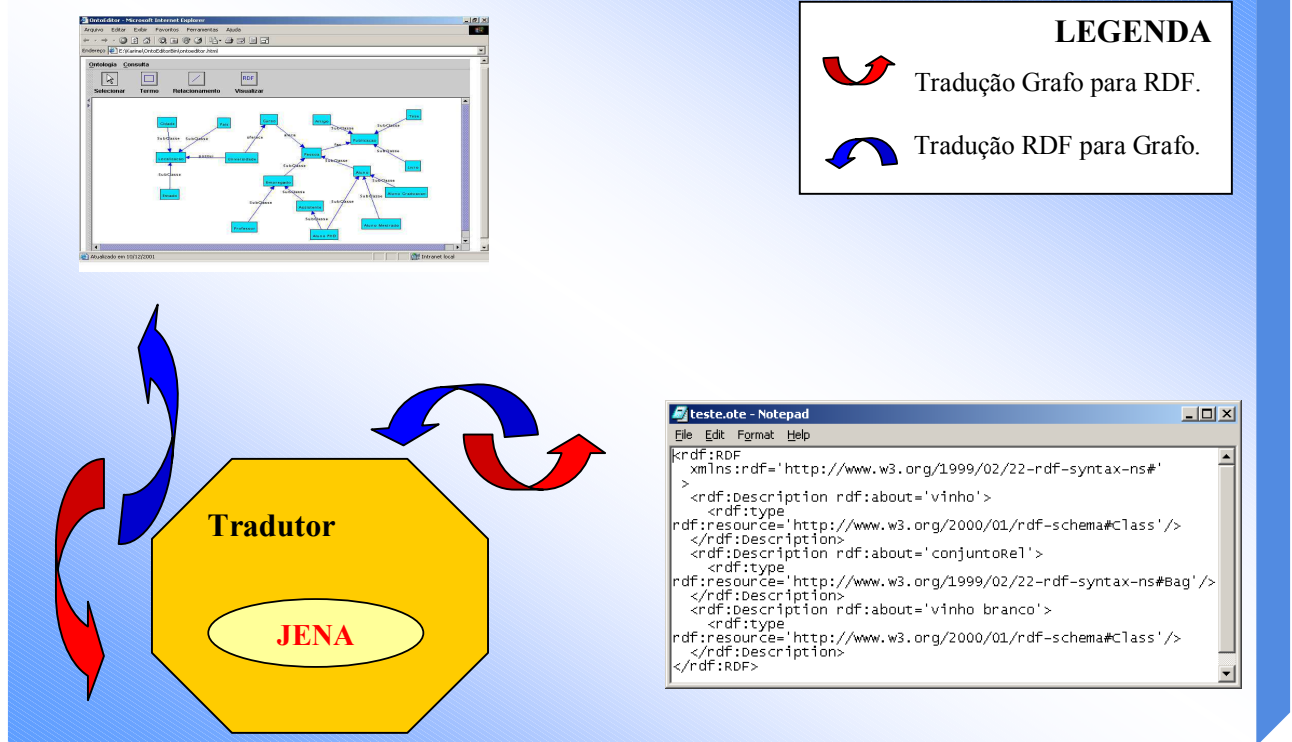
Como já foi dito, os usuários do OntoEditor criam e editam suas ontologias graficamente e elas são armazenadas num banco de dados, descritas na linguagem RDF. Entretanto, para que as ontologias pudessem ser transformada de uma descrição sob a forma gráfica para uma descrição em RDF, foi necessário definirmos e implementarmos um processo intermediário que tornou possível uma correlação direta entre os componentes gráficos do modelo de dados do OntoEditor (termos, relacionamentos e propriedades) e o vocabulário da especificação RDF e esquema RDF. Esse processo foi definido como *processo de tradução*.

Nesse processo, o OntoEditor faz a tradução das ontologias representadas graficamente para ontologias representadas em RDF. Isso acontece quando a ontologia é armazenada/salva no banco de dados, ou exportada para um arquivo, e vice-versa, ou seja, quando uma ontologia é recuperada do banco de dados, ou importada de um arquivo, e re-exibida na interface do OntoEditor.

Para gerar e extrair ontologias descritas em RDF, abstraindo os detalhes de sintaxe requeridos pela especificação, o OntoEditor utiliza a *api Jena* [HP]. A Jena pode ser descrita basicamente como uma biblioteca de componentes utilizada para gerar documentos RDF.

A FIGURA 4.4 ilustra a utilização da Jena no processo de Tradução de ontologias oferecido pelo OntoEditor. As setas em vermelho ilustram o processo de Tradução feito a partir da ontologia representada no grafo para a ontologia representada em RDF. As setas em azul ilustram o processo contrário, onde a tradução ocorrerá a partir de um documento RDF e gerará uma representação gráfica para aquela ontologia.

## APPLET



**FIGURA 4.4** Visão específica do processo de Tradução

De forma similar à utilizada com a biblioteca OpenJGraph, foram utilizados em nosso projeto alguns dos pacotes disponibilizados pela Jena. Foram utilizados somente os pacotes necessários para transformar cada elemento do modelo de dados do OntoEditor em um documento RDF de conteúdo equivalente. A FIGURA 4.5 mostra os pacotes que foram utilizados pelo OntoEditor.

<b>OntoEditor - Implementação</b>
<pre> import com.hp.hpl.mesa.rdf.jena.mem.ModelMem; import com.hp.hpl.mesa.rdf.jena.model.*; import com.hp.hpl.mesa.rdf.jena.vocabulary.*; </pre>

**FIGURA 4.5** Implementação OntoEditor/Processo de Tradução

Como o modelo de aquisição do conhecimento do OntoEditor é baseado no modelo da linguagem RDF e esquema RDF, as correlações necessárias entre grafo e RDF, para que a

tradução seja implementada assegurando a equivalência de conteúdo, foram feitas da forma mais transparente e direta possível. Mostraremos a seguir uma correlação entre os elementos do modelo de dados do OntoEditor e o modelo RDF, enfatizando os métodos da *api* Jena utilizados na implementação.

Nº	MODELO ONTOEDITOR	MODELO RDF	IMPLEMENTAÇÃO COM API JENA
1	TERMO	RDFS.CLASS	RESOURCE TERMORDF = MODEL.CREATERESOURCE (NOMETERMO, RDFS.CLASS);  // Criando um recurso TERMORDF do tipo RDFS.Class cujo nome é igual ao NOME DO TERMO.
2	DESCRIÇÃO DO TERMO	RDFS.SEEALSO	TERMORDF.ADDPROPERTY ( RDFS.SEEALSO, DESCRICAO );  // Adicionando ao TERMORDF uma propriedade do tipo RDFS.seeAlso que irá conter a descrição do termo, caso ela tenha sido informada.
3	PROPRIEDADE DO TERMO	RDF.PROPERTY	RESOURCE PROPRIEDADERDF = MODEL.CREATERESOURCE (NOMEPROPRIIDADE, RDF.PROPERTY);  // Criando um recurso novo para cada propriedade definida pelo usuário que será adicionada ao termo.  PROPRIEDADERDF.ADDPROPERTY (RDFS.DOMAIN, NOMETERMO);  // Informando o termo ao qual essa propriedade se refere. Associando a propriedade ao termo.
4	TIPO DO VALOR DA PROPRIEDADE	RDFS.RANGE	PROPRIEDADERDF.ADDPROPERTY

			(RDFS.RANGE, TIPOVALOR);  // Informando um escopo para o tipo do valor da propriedade.
5	RELACIONAMENTO	RDFS.SUBCLASSOF	TERMORDF.ADDPROPERTY ( RDFS.SUBCLASSOF, NOMETERMO);  // Adiciona uma propriedade tipo RDFS.subClassOf a um termo A, informando no campo NOMETERMO o nome da superclasse.
6	RELACIONAMENTO	OUTROS	BAG CONJUNTOREL = MODEL.CREATEBAG ("CONJUNTOREL"); PROPERTY RELACIONAMENTO = MODEL.CREATEPROPERTY( "HTTP://WWW.KARINE.COM#",NOMERELACIONAMENTO);  // Cria um Bag de propriedades com todos os relacionamentos definidos pelo usuário.  CONJUNTOREL.ADDPROPERTY ( RELACIONAMENTO, NOMERELACIONAMENTO );  // Informa o nome do relacionamento.

**TABELA 4.2 Mapeamento entre elementos do Modelo de Dados OntoEditor e do Modelo RDF**

### 4.3.1 Validação do Processo de Tradução

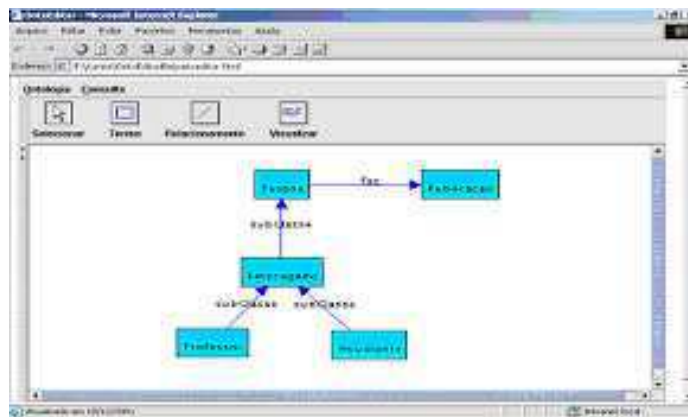
Após chegarmos ao ponto em que o OntoEditor gerava um documento RDF a partir de um grafo representado na sua interface, observamos a necessidade de verificar a validade do documento gerado, ou seja, validar se o documento gerado estava descrito obedecendo aos padrões do modelo RDF e esquema RDF de forma correta.

Para realizarmos a validação foi utilizado um *parser* RDF, chamado SiRPAC [SiRPAC], disponibilizado pela W3C. O SiRPAC gera, a partir de um documento escrito em RDF/XML, as tuplas correspondentes ao modelo de dados RDF [Hjelm – 2001].

O SiRPAC foi utilizado para validar todos os elementos do modelo de dados do OntoEditor, assegurando que os documentos gerados estavam de acordo com a especificação RDF e esquema RDF.

Mostraremos a seguir algumas das validações efetuadas durante o processo de Tradução de ontologias. Para simplificarmos a demonstração da validação, utilizamos uma parte da ontologia Universidade, reduzindo, assim, a repetência da apresentação da validação para elementos similares. A ontologia foi mostrada no capítulo III e a sua parte que fora utilizada é ilustrada na FIGURA 4.6.

A partir da representação da FIGURA 4.6, foi gerado documento RDF mostrado na FIGURA 4.7.



**FIGURA 4.6 Parte da Ontologia Universidade**

```

<rdf:RDF
  xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:NS0='http://www.karine.com#'
  xmlns:rdfs='http://www.w3.org/2000/01/rdf-schema#'
>
  <rdf:Description rdf:about='nome'>
    <rdf:type rdf:resource='http://www.w3.org/1999/02/22-rdf-syntax-ns#Property'/>
    <rdfs:domain>Pessoa</rdfs:domain>
    <rdfs:range>string</rdfs:range>
  </rdf:Description>
  <rdf:Description rdf:about='Professor'>
    <rdf:type rdf:resource='http://www.w3.org/2000/01/rdf-schema#Class'/>
    <rdfs:subClassOf>Empregado</rdfs:subClassOf>
  </rdf:Description>
  <rdf:Description rdf:about='data'>
    <rdf:type rdf:resource='http://www.w3.org/1999/02/22-rdf-syntax-ns#Property'/>
    <rdfs:domain>Publicacao</rdfs:domain>
    <rdfs:range>string</rdfs:range>
  </rdf:Description>
  <rdf:Description rdf:about='conjuntoRel'>
    <rdf:type rdf:resource='http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag'/>
    <NS0:faz>Publicacao</NS0:faz>
  </rdf:Description>
  <rdf:Description rdf:about='Publicacao'>
    <rdf:type rdf:resource='http://www.w3.org/2000/01/rdf-schema#Class'/>
  </rdf:Description>
  <rdf:Description rdf:about='idade'>
    <rdf:type rdf:resource='http://www.w3.org/1999/02/22-rdf-syntax-ns#Property'/>
    <rdfs:domain>Pessoa</rdfs:domain>
    <rdfs:range>inteiro</rdfs:range>
  </rdf:Description>
  <rdf:Description rdf:about='Assistente'>
    <rdf:type rdf:resource='http://www.w3.org/2000/01/rdf-schema#Class'/>
    <rdfs:subClassOf>Empregado</rdfs:subClassOf>
  </rdf:Description>
  <rdf:Description rdf:about='Empregado'>
    <rdf:type rdf:resource='http://www.w3.org/2000/01/rdf-schema#Class'/>
    <rdfs:subClassOf>Pessoa</rdfs:subClassOf>
  </rdf:Description>
  <rdf:Description rdf:about='Pessoa'>
    <rdf:type rdf:resource='http://www.w3.org/2000/01/rdf-schema#Class'/>
    <NS0:faz>Publicacao</NS0:faz>
  </rdf:Description>
</rdf:RDF>

```

**FIGURA 4.7 Documento RDF gerado a partir da ontologia mostrada na FIGURA 4.6**

O SiRPAC gerou as triplas (ver TABELA 4.3) a partir do documento mostrado na FIGURA 4.7. Através das triplas podemos formar sentenças e avaliar se a sentença está usando os elementos do modelo RDF e esquema RDF corretamente para expressar o conteúdo semântico desejado.

Number	Subject	Predicate	Object
1	<a href="#">nome</a>	<a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">http://www.w3.org/1999/02/22-rdf-syntax-ns#type</a>	<a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property">http://www.w3.org/1999/02/22-rdf-syntax-ns#Property</a>
2	<a href="#">nome</a>	<a href="http://www.w3.org/2000/01/rdf-schema#domain">http://www.w3.org/2000/01/rdf-schema#domain</a>	Pessoa
3	<a href="#">nome</a>	<a href="http://www.w3.org/2000/01/rdf-schema#range">http://www.w3.org/2000/01/rdf-schema#range</a>	string
4	<a href="#">Professor</a>	<a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">http://www.w3.org/1999/02/22-rdf-syntax-ns#type</a>	<a href="http://www.w3.org/2000/01/rdf-schema#Class">http://www.w3.org/2000/01/rdf-schema#Class</a>
5	<a href="#">Professor</a>	<a href="http://www.w3.org/2000/01/rdf-schema#subClassOf">http://www.w3.org/2000/01/rdf-schema#subClassOf</a>	Empregado
6	<a href="#">data</a>	<a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">http://www.w3.org/1999/02/22-rdf-syntax-ns#type</a>	<a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property">http://www.w3.org/1999/02/22-rdf-syntax-ns#Property</a>
7	<a href="#">data</a>	<a href="http://www.w3.org/2000/01/rdf-schema#domain">http://www.w3.org/2000/01/rdf-schema#domain</a>	Publicacao
8	<a href="#">data</a>	<a href="http://www.w3.org/2000/01/rdf-schema#range">http://www.w3.org/2000/01/rdf-schema#range</a>	string
9	<a href="#">conjuntoRel</a>	<a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">http://www.w3.org/1999/02/22-rdf-syntax-ns#type</a>	<a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag">http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag</a>
10	<a href="#">conjuntoRel</a>	<a href="http://www.karine.com#faz">http://www.karine.com#faz</a>	Publicacao
11	<a href="#">Publicacao</a>	<a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">http://www.w3.org/1999/02/22-rdf-syntax-ns#type</a>	<a href="http://www.w3.org/2000/01/rdf-schema#Class">http://www.w3.org/2000/01/rdf-schema#Class</a>
12	<a href="#">idade</a>	<a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">http://www.w3.org/1999/02/22-rdf-syntax-ns#type</a>	<a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property">http://www.w3.org/1999/02/22-rdf-syntax-ns#Property</a>
13	<a href="#">idade</a>	<a href="http://www.w3.org/2000/01/rdf-schema#domain">http://www.w3.org/2000/01/rdf-schema#domain</a>	Pessoa
14	<a href="#">idade</a>	<a href="http://www.w3.org/2000/01/rdf-schema#range">http://www.w3.org/2000/01/rdf-schema#range</a>	inteiro



15	<a href="#">Assistente</a>	<a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">http://www.w3.org/1999/02/22-rdf-syntax-ns#type</a>	<a href="http://www.w3.org/2000/01/rdf-schema#Class">http://www.w3.org/2000/01/rdf-schema#Class</a>
16	<a href="#">Assistente</a>	<a href="http://www.w3.org/2000/01/rdf-schema#subClassOf">http://www.w3.org/2000/01/rdf-schema#subClassOf</a>	Empregado
17	<a href="#">Empregado</a>	<a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">http://www.w3.org/1999/02/22-rdf-syntax-ns#type</a>	<a href="http://www.w3.org/2000/01/rdf-schema#Class">http://www.w3.org/2000/01/rdf-schema#Class</a>
18	<a href="#">Empregado</a>	<a href="http://www.w3.org/2000/01/rdf-schema#subClassOf">http://www.w3.org/2000/01/rdf-schema#subClassOf</a>	Pessoa
19	<a href="#">Pessoa</a>	<a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">http://www.w3.org/1999/02/22-rdf-syntax-ns#type</a>	<a href="http://www.w3.org/2000/01/rdf-schema#Class">http://www.w3.org/2000/01/rdf-schema#Class</a>
20	<a href="#">Pessoa</a>	<a href="http://www.karine.com#faz">http://www.karine.com#faz</a>	Publicacao

**TABELA 4.3** Triplas referentes ao documento da FIGURA 4.6, geradas pelo SiRPAC

Podemos ler as triplas formando sentenças da seguinte maneira:

- A coluna *Subject* representa o sujeito da sentença;
- A coluna *Predicate* representa o predicado da sentença; e
- A coluna *Object* representa o objeto ao qual a sentença se refere.

Logo, temos para cada tripla a sentença que segue:

1. nome é do tipo propriedade, ou seja, o elemento `rdf_type` indica que o conteúdo do objeto representa o tipo do conceito descrito. Nesse caso, o sujeito é um objeto do tipo propriedade.
2. nome está no domínio de Pessoa. Através do elemento `rdfs_domain`, podemos associar uma propriedade a uma classe. Nesse caso, a classe Pessoa tem uma propriedade chamada nome.
3. O range de nome é string, ou seja o tipo do valor que a propriedade pode assumir é uma string. Exemplificando, a propriedade nome de uma instância da classe Pessoa possui o valor Maria.
4. Professor é do tipo classe, ou seja, o elemento `rdf_type` indica que o conteúdo do objeto representa o tipo do conceito descrito. Nesse caso, o sujeito é um objeto do tipo classe.
5. Professor é subclasse de Empregado.

6. data é uma propriedade.
7. data faz parte de Publicação. O item mostra que Publicação é uma classe, logo data é uma propriedade da classe Publicação.
8. O tipo do valor de data é string.
9. conjuntoRel é do tipo bag. O elemento rdfs\_bag é utilizado para agrupar outros elementos e em nosso projeto ele foi utilizado para agrupar todos os relacionamentos (os definidos pelo usuário e os oferecidos pela ferramenta como sinônimo, antônimo etc, com exceção do relacionamento subclasse que é tratado separadamente por questões das implicações trazidas com a herança de classes).
10. conjuntoRel possui um relacionamento chamado faz cujo objeto é Publicação. O OntoEditor entende que cada informação contida no bag conjuntoRel representa um relacionamento e isso será utilizado para auxiliar o processo de tradução no momento em que a ontologia armazenada em RDF é transformada para representação gráfica.
11. Publicação é do tipo classe.
12. idade é do tipo propriedade.
13. idade é uma propriedade de Pessoa.
14. O tipo do valor de idade é inteiro.
15. Assistente é do tipo classe.
16. Assistente é subclasse de Empregado.
17. Empregado é do tipo classe.
18. Empregado é subclasse de Pessoa.
19. Pessoa é do tipo classe.
20. Pessoa faz Publicação.

#### 4.4 Consultas – Máquina de Busca

Com o OntoEditor, o usuário pode realizar consultas às ontologias armazenadas através das propriedades ou do conteúdo delas. As consultas são realizadas com base nas informações fornecidas pelo usuário e descreveremos, a seguir, como elas foram implementadas.

#### 4.4.1 Utilização do JDBC

A comunicação feita entre o OntoEditor (aplicação) e a base de dados ocorre via JDBC.

Com o JDBC, o OntoEditor poderá realizar conexões ao banco de dados de forma direta sem ter que se preocupar com as características de implementação dos SGBDs. Para tanto, o JDBC requisita que a ferramenta esteja configurada com o *driver* específico para o SGBD que será utilizado. Mostramos na FIGURA 4.8 a configuração do JDBC para utilização do *driver* para o *SQLServer*, que foi o SGBD usado em nosso projeto [ThinWeb].

<b><u>OntoEditor - Implementação</u></b>
<pre>Class.forName("com.thinweb.tds.Driver");</pre>

**FIGURA 4.8 Implementação do OntoEditor/Driver utilizado para acesso ao banco de dados**

Para que o OntoEditor possa efetivamente acessar sua base de dados, além de realizar a configuração mostrada na FIGURA 4.8, ela deverá criar uma conexão JDBC. A FIGURA 4.9 ilustra a forma como foi implementada essa conexão.

<b><u>OntoEditor - Implementação</u></b>
<pre>Connection conn; conn = DriverManager.getConnection(OntoEditor.dbURL, OntoEditor.dbUser, OntoEditor.dbPassword);</pre>

**FIGURA 4.9 Implementação do OntoEditor/Conexão com o banco de dados**

De acordo com a FIGURA 4.9, podemos perceber que a conexão necessita de alguns parâmetros para que ela seja efetivada. Esses parâmetros são *OntoEditor.dbURL*, *OntoEditor.dbUser*, *OntoEditor.dbPassword* e eles indicarão, respectivamente, onde está a

base de dados (em que servidor na *Web*), qual o usuário e qual a senha requeridos pelo SGBD para que a base possa ser acessada.

No OntoEditor, esses parâmetros são informados no código HTML contido na página que hospeda o *applet* (aplicação), como mostra a FIGURA 4.10.

<u>OntoEditor - Implementação</u>
<pre> &lt;PARAM NAME="DB_URL" VALUE="jdbc:twtds:sqlserver://desenv_autom01/ProBd_iW orkplaceTeste"&gt; &lt;PARAM NAME="DB_USER" VALUE="dono_iwpteste"&gt; &lt;PARAM NAME="DB_PASSWORD" VALUE="dono_iwpteste"&gt; </pre>

**FIGURA 4.10 Implementação do OntoEditor/Parâmetros utilizados para conexão**

#### 4.4.2 Consultas através das Propriedades da Ontologia

A finalidade dessa funcionalidade é permitir que, através das propriedades da ontologia, os usuários do OntoEditor possam realizar consultas às ontologias armazenadas no banco de dados, tais como: quais as ontologias criadas pelo autor Carlos Barreto, ou quais as ontologias com o domínio Universidade, ou quais as ontologias criadas pelo autor Carlos Barreto cujo domínio é Universidade.

Esse tipo de consulta é bastante simples, quanto ao aspecto de implementação. Quando essa consulta é executada pelo usuário, o OntoEditor captura as informações fornecidas (propriedades da ontologia) e faz um acesso a sua base de dados, à procura de ontologias que possuam tais características.

Através de uma conexão JDBC ao banco de dados, o OntoEditor faz uma consulta tipo *Select* à tabela de ontologias armazenada. Essa consulta não é CaseSensitive e todos os campos são tratados como string.

Na FIGURA 4.11, temos um exemplo de uma consulta à tabela de ontologias, buscando todas as ontologias que possuam o campo Domínio igual a palavra Universidade e o campo Autor igual a palavra Francisco Barros. A FIGURA 4.12 mostra como essa consulta fora implementada.

**FIGURA 4.11 Interface OntoEditor para Consulta por Propriedades da Ontologia**

<b><u>OntoEditor - Implementação</u></b>
<pre> Connection conn; conn.createStatement().executeQuery("select Codigo, Nome, Dominio, Autor, Linguagem from TB_Ontologia Ontologia where Domínio='Universidade' and Autor='Francisco Barros' "); </pre>

**FIGURA 4.12 Implementação do OntoEditor/ Consulta por Propriedades da Ontologia**

#### **4.4.3 Consultas através do Conteúdo da Ontologia**

Com esse tipo de consulta, o OntoEditor possibilita aos seus usuários uma procura no seu banco de dados por um determinado termo ou palavra-chave que possa fazer parte do conteúdo de alguma, ou várias, ontologia(s).

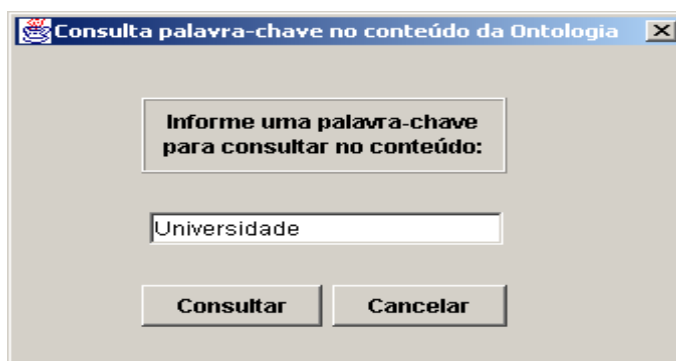
Como já foi dito, as ontologias estão armazenadas no campo RDF, tipo *Image*, da tabela de ontologias para cada registro do banco de dados. Para cada ontologia gravada no banco de dados, esse campo será preenchido com a ontologia propriamente dita, ou seja, um texto, descrevendo a ontologia, escrito na linguagem RDF, cuja sintaxe é o XML.

Apesar da nossa base de dados ser relacional, teremos um campo especial na tabela. Esse campo representa uma base de dados especial (Dados XML), cuja sintaxe é bastante diferenciada do modelo relacional e por isso, para que pudéssemos recuperar e manipular tais informações, houve a necessidade de usarmos uma linguagem de consulta específica para sintaxe XML. Nós utilizamos a linguagem de consulta XML-QL que é baseada na sintaxe *where/construct* ao invés da familiar *select/from/where* do SQL [Abiteboul et al. - 2000].

Para executar a consulta utilizando a linguagem XML-QL, foi necessário utilizar um processador de consultas específico e nós utilizamos o XML-QL Processor [XML-QL].

Ao realizar uma consulta por conteúdo, o usuário simplesmente informa a palavra-chave que deseja procurar. A partir dessa informação, o OntoEditor cria uma conexão JDBC, acessa a tabela de ontologias, varrendo-a linha a linha à procura das ontologias que possuam essa palavra-chave no seu conteúdo.

A FIGURA 4.13 mostra um exemplo de consulta no OntoEditor , que busca todas as ontologias no seu banco de dados que contenham em seu conteúdo,descrito em RDF, pelo menos uma vez , a palavra Universidade.



**FIGURA 4.13 Interface Gráfica OntoEditor para Consulta por Conteúdo**

O OntoEditor realiza a implementação das consultas por conteúdo da seguinte forma:

- É executada uma consulta simples na tabela de ontologias, onde são retornados os campos Codigo e RDF para cada ontologia que está armazenada no banco de dados. A FIGURA 4.14 ilustra essa a implementação dessa consulta.

<u>OntoEditor – Implementação</u>
<pre>conn.createStatement().executeQuery("select Codigo, RDF from TB_Ontologia ");</pre>

**FIGURA 4.14 Implementação do OntoEditor/ Consulta por Conteúdo 1**

À medida que possuímos todos os campos RDF registrados na tabela de ontologias, será realizada uma consulta específica dentro de cada documento RDF, um a um, à procura de uma determinada palavra-chave. Como um documento RDF é baseado na sintaxe XML, foi utilizado um processador específico para realizar consultas em documentos XML: XML-QL Processor.

- Cada documento RDF, retornado na consulta mostrada na FIGURA 4.14, será colocado num arquivo que servirá de entrada para o processador de consultas XML-QL. Ou seja, para cada ontologia armazenada, o OntoEditor vai capturar o campo RDF, extraíndo o seu conteúdo e colocando-o num arquivo temporário que ficará armazenado localmente no diretório C:/temp da máquina que está executando o *applet Java*. A implementação é mostrada através da FIGURA 4.15.

<u>OntoEditor - Implementação</u>
<pre>File arq = null; File.createTempFile ("arq", ".ote", new File("c:/temp")); FileWriter arquivo; arquivo = new FileWriter ( arq ); arquivo.write( rdf );</pre>

**FIGURA 4.15 Implementação do OntoEditor/ Consulta por Conteúdo 2**

- Um segundo arquivo, `rdfQuery.xmlql`, contendo a consulta por conteúdo na linguagem XML-QL propriamente dita é utilizado. O conteúdo desse arquivo é fixo, ou seja, independe da ontologia que está armazenada, e foi definido na linguagem XML-QL para atender as necessidades da funcionalidade consulta por conteúdo disponibilizada pelo OntoEditor. O código mostrado na FIGURA 4.16 mostra a implementação de uma consulta XML-QL que recebe dois parâmetros: o primeiro é o arquivo temporário com o documento RDF e o segundo é a palavra-chave que está sendo procurada. Baseado nesses parâmetros, é procurado no arquivo temporário as *tags* XML que possuam o conteúdo igual a `<rdf:Description rdf:about=$param>`, onde `$param` é igual à palavra-chave. Cada vez que a palavra-chave for identificada em alguma das *tags* do documento, um valor `true` será atribuído a uma tag ontologia e gravado no arquivo de saída da consulta. Independentemente do resultado encontrado em cada *tag*, `true` ou `false`, todo o documento é lido na consulta.

<b><u>OntoEditor - Implementação</u></b>
<pre>function QueryAbout (\$input, \$param) { where     &lt;rdf:RDF&gt;     &lt;rdf:Description rdf:about=\$param&gt;&lt;/&gt;     &lt;/&gt;     IN source(\$input), \$valor = "true" construct &lt;ontologia&gt;\$valor&lt;/&gt;</pre>

**FIGURA 4.16 Implementação do OntoEditor/ Consulta por Conteúdo 3**

- Um terceiro arquivo é utilizado como saída da consulta e nele teremos um documento XML. Nesse documento, poderemos ter uma ou várias *tags* `<ontologia>` com valor igual a `true`. Sempre que a palavra-chave for encontrada na ontologia que está sendo pesquisada, essa *tag* será gravada no arquivo de saída. O arquivo de saída só possuirá conteúdo se a palavra-chave for encontrada pelo menos uma vez dentro da ontologia, caso contrário, o arquivo será gerado sem conteúdo algum.

Após consultar cada documento RDF da tabela, o OntoEditor verifica se há pelo menos um valor `true` para *tag* `<ontologia>` no arquivo de saída. Caso exista, o código que identifica essa ontologia na tabela do banco será adicionado a um vetor de códigos. Esse procedimento é repetido até o último registro da tabela de ontologias. O vetor de códigos será



utilizado depois para identificar as ontologias que possuem a palavra-chave procurada e, baseado nele, o OntoEditor exibirá para o usuário uma lista com tais ontologias.

Utilizamos dois algoritmos para melhor enfatizar a descrição do processo consulta por conteúdo. Eles são apresentados a seguir nas FIGURAS 4.17 e 4.18.

```

consulta_Conteudo_Ontologia (palavra_chave = string);

// palavra_chave é um nome qualquer que servirá como parâmetro de entrada para
consulta

// Declaração de variáveis

variáveis:

    Arq_DocumentoXML, Arq_Saída    : Tipo Arquivo;
    verifica                       : Tipo lógico;

// A variável verifica assume valor igual a verdadeiro ou falso;
//

// Inicialização das variáveis;

Arq_DocumentoXML = branco;
Arq_Saída         = branco;
//

ler_Tabela;

Até Fim_Tabela faça:
    Arq_DocumentoXML = Tabela.DocumentoRDF;
    gravar_Arq_DocumentoXML;
    verifica = efetivar_Consulta (Arq_DocumentoXML, palavra_chave);
    se verifica = verdadeiro
        então Arq_saída = '<ontologia>>true</ontologia>';
    Arq_DocumentoXML = branco;
    ler_Prox_Tabela;
Fim_Até;

Fim_consulta_Conteudo_Ontologia;

```

**FIGURA 4.17 Algoritmo Consulta\_Por\_Conteúdo**

```
efetivar_Consulta (Arq_DocumentoXML, palavra_chave) : Tipo_Logico;

// efetivar_Consulta é a consulta propriamente dita realizada pelo processador XML. Ela
// retorna um valor verdadeiro ou falso para sua execução.

// Declaração de variáveis

variáveis:

valor_ontologia : Tipo_Logico;
//

valor_ontologia = falso;

Até fim_ Arq_DocumentoXML faça:

// verifica em cada sentença RDF o valor da propriedade rdf:Description
// se Arq_DocumentoXML .rdf:Description = palavra_chave
// então valor_ontologia = verdadeiro;

Fim_ate;

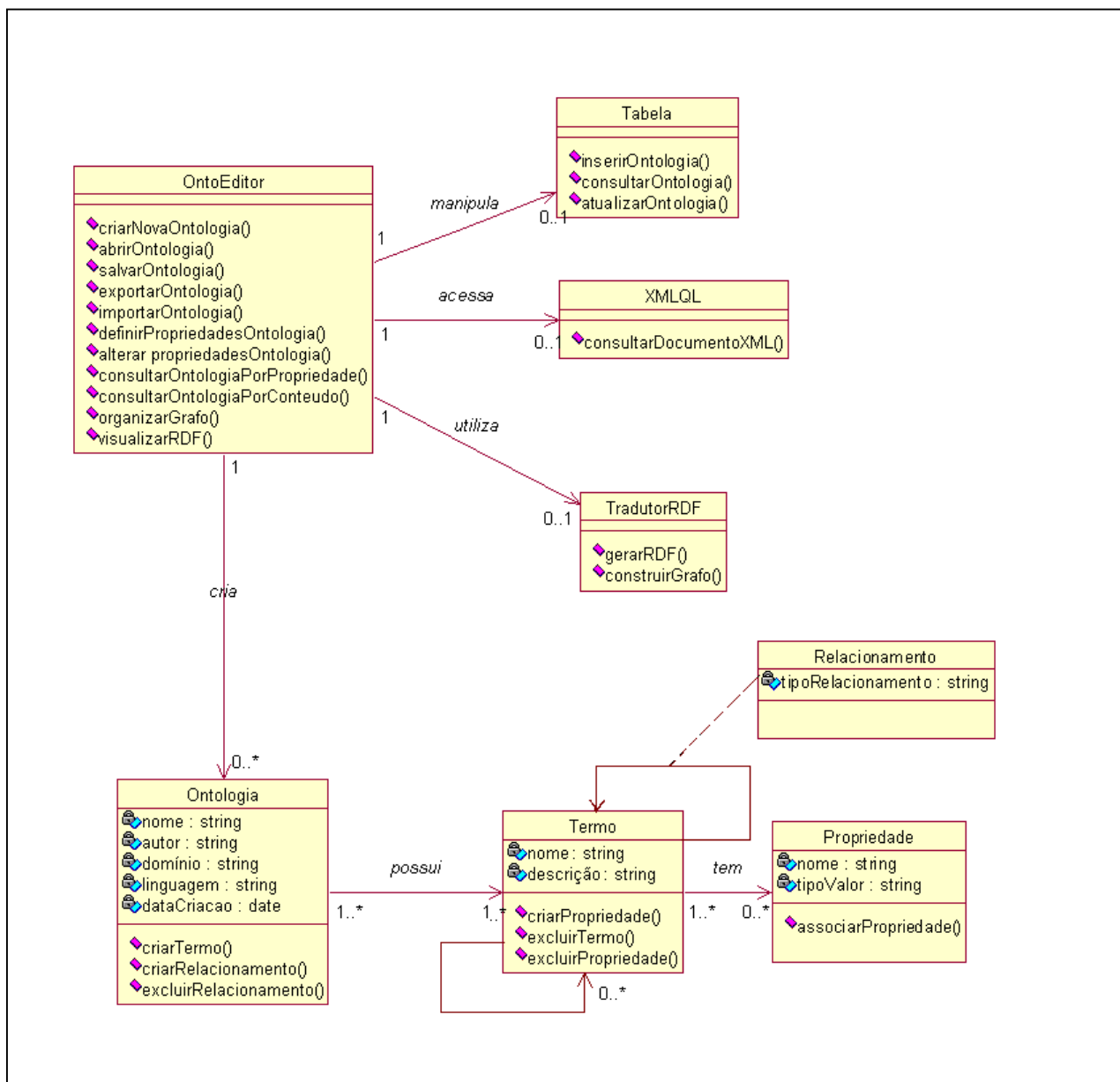
Fim_efetivar_Consulta;
```

**FIGURA 4.18 Algoritmo Consulta\_XML\_Processor**

## 4.5 Diagrama de Classes

Um diagrama de classes ilustra as especificações para as classes de software e de interfaces, por exemplo, interfaces Java de uma aplicação [Larman – 2000].

Mostraremos, na FIGURA 4.19, o diagrama de classes utilizado no desenvolvimento do OntoEditor.



**FIGURA 4.19 Diagrama de Classes**

A partir do diagrama de classes, podemos extrair as informações que seguem:

A classe *OntoEditor* possui os seguintes métodos:

1. *criarNovaOntologia* – cria uma nova ontologia;
2. *abrirOntologia()* – abre uma ontologia que foi criada pelo *OntoEditor* e armazenada no banco de dados;
3. *salvarOntologia()* – salva uma ontologia no banco de dados;
4. *exportarOntologia()* – armazena a ontologia representada em um arquivo texto convencional;

5. importarOntologia() – recupera ontologias representando-as graficamente na interface do editor a partir de arquivos textos convencionais criados com a opção citada no item anterior (5);
6. definirPropriedadesOntologia() – define as propriedades gerais da ontologia (Nome, Autor, Domínio e Data de Criação);
7. alterarpropriedadesOntologia() – edita as propriedades de uma ontologia;
8. consultarOntologiaPorPropriedade() – consulta ontologias a partir das propriedades da ontologia informada pelo usuário;
9. consultarOntologiaPorConteúdo() – consulta a existência de ontologias que possuam em seu conteúdo a palavra-chave informada pelo usuário.
10. organizarGrafo() - organiza a forma como o grafo está sendo exibido na interface do editor; e
11. visualizarRDF() – mostra o documento RDF gerado a partir do grafo que está disposto na interface.

A classe Ontologia possui os seguintes métodos:

- a. criarTermo() – cria um termo graficamente;
- b. criarRelacionamento() – cria um relacionamento entre dois termos graficamente; e
- c. excluirRelacionamento() – exclui o termo do grafo.

A classe Termo possui os seguintes métodos:

1. criarPropriedade() – cria uma nova propriedade para o termo;
2. excluirTermo() – exclui o termo do grafo; e
3. excluirPropriedade() – exclui uma dada propriedade para um certo termo.

A classe Propriedade possui os seguintes métodos:

1. associarPropriedade() – associa ao termo selecionado uma propriedade que já foi definida para um outro termo.

A classe Tabela possui os seguintes métodos:

1. inserirOntologia() – insere um registro na tabela de ontologia do banco de dados;
2. consultarOntologia() – consulta a tabela de ontologia no banco de dados; e

3. atualizarOntologia() - atualiza os registros na tabela de ontologia do banco de dados.

A classe XMLQL possui os seguintes métodos:

1. consultarDocumentoXML() – consulta alguma palavra-chave em documentos XML.

A classe TradutorRDF possui os seguintes métodos:

1. gerarRDF() – gera um documento RDF respectivo à ontologia representada graficamente; e
2. construirGrafo() – constrói a partir de um documento RDF um grafo representando a respectiva ontologia.

## 4.6 Conclusão

Descrevemos nesse capítulo, as principais características de implementação envolvidas para construir o OntoEditor: uma ferramenta para manipular ontologias na *Web*.

Para facilitar a visualização da nossa solução ao nível de implementação, apresentamos esse capítulo dividido basicamente entre características de implementação envolvidas com a ferramenta propriamente dita, ou seja, a camada de aplicação do nosso projeto e a camada de dados, bem como toda intercomunicação entre as camadas.

Foram descritas de forma individualizada as soluções computacionais utilizadas para atender as funcionalidades do OntoEditor como: interface gráfica, integração ao banco de dados, tradução de ontologias e consultas. Além disso, foi apresentado como essas funcionalidades interagem para formar o OntoEditor, através da descrição do diagrama de classes do nosso projeto.

## Capítulo V - CONCLUSÃO

O objetivo do nosso trabalho foi construir uma ferramenta para manipular ontologias na *Web*, chamada de OntoEditor. Nessa dissertação, mostramos a fundamentação teórica que nos deu suporte para idealizar e construir a ferramenta e descrevemos as características relevantes relacionadas ao projeto e implementação da ferramenta.

No início do nosso trabalho, observamos que já existiam outras ferramentas disponibilizadas na *Web* com o mesmo propósito. A partir daí, começamos a analisar algumas delas, optamos pelas mais utilizadas, e, apesar de considerarmos-as ferramentas ricas em funcionalidades e flexibilidades, percebemos que havia algumas limitações.

Dentro desse contexto, a nossa proposta foi oferecer uma ferramenta que disponibilizasse funcionalidades que até então não existiam em outras ferramentas, mas que facilitariam a representação e manipulação de ontologias na *Web*.

Tal como as demais ferramentas, o OntoEditor oferece um editor onde os usuários podem criar suas ontologias, definindo seus termos, propriedades e relacionamentos. O OntoEditor é uma ferramenta que pode ser acessada a partir de qualquer browser e uma vez que não era o propósito do nosso trabalho oferecer um editor rico em funcionalidades gráficas, até porque as outras ferramentas já atendem de forma muito satisfatória a tais características, ele possui uma interface gráfica bastante simples de ser utilizada, porém muito objetiva.

Além de poder criar suas ontologias através do OntoEditor, os usuários podem armazenar suas ontologias num banco de dados e manipulá-las posteriormente conforme suas necessidades. Tudo isso é possível porque o OntoEditor possui integração direta a um banco de dados. O usuário cria suas ontologias graficamente e elas são armazenadas no banco de dados no formato que vem sendo mais utilizado dentro da *Web* para representação de metadados na *Web*: o RDF da W3C, cuja sintaxe é feita em XML. Portanto, ao final, o

usuário terá sua ontologia descrita como um documento RDF e toda a operação de tradução da ontologia, representada graficamente para representação RDF, fica transparente para o usuário que se concentra somente em alimentar a sua ontologia com as informações necessárias para suportar o entendimento semântico do seu domínio de conhecimento.

O OntoEditor também realiza o processo inverso da tradução, assim, todas as vezes que os usuários necessitarem reeditar suas ontologias, já armazenadas no banco de dados, ele transforma o documento RDF em ontologia gráfica.

A ferramenta também oferece um menu de consultas, onde os usuários podem consultar sua base de dados e buscar tanto por meta-informações da ontologia, tipo nome da ontologia, autor etc, como por informações que estejam no conteúdo da ontologia, ou seja dentro do documento RDF propriamente dito. Por exemplo, o usuário deseja consultar todas as ontologias que estejam tratando a informação: peças de automóveis. A partir daí, o OntoEditor consulta toda sua base de informação e oferece como respostas toda(s) a(s) ontologia(s) que tratem do termo peças de automóveis, caso elas existam, evidentemente.

Outro importante diferencial do OntoEditor está na forma como a ontologia é representada graficamente. Nos demais editores, as ontologias são criadas na forma de árvores. Percebemos que para representar o mundo real, com toda a sua riqueza de detalhes, esse tipo de grafo apresenta limitações, uma vez que cada elemento dessa estrutura possui somente um ancestral. Muitas das representações que encontramos apresentam elementos que possuem mais de um ancestral, caso típico de herança múltipla, e por isso, no OntoEditor, as ontologias são representadas sob uma estrutura de grafo simples, suprimindo tal limitação.

Analisando sob uma forma mais generalizada e num primeiro plano, podemos considerar que as funcionalidades propostas pelo OntoEditor buscam auxiliar diretamente a viabilização da Semântica na *Web*, Isso acontece através da criação de ontologias padronizadas e integração da ferramenta a um banco de dados, visando atingir a construção de um repositório de ontologias.

Todavia, como o OntoEditor lida basicamente com ontologias e as mesmas podem ser utilizadas para auxiliar na automatização de diversos, como já foi mostrado no capítulo II, podemos ressaltar a importância da ferramenta, identificando, indiretamente, outras

aplicações que poderiam ser beneficiadas com o auxílio do OntoEditor, tais como: consultas na *Web* e o comércio eletrônico.

## 5.1 Trabalhos Futuros

Consideramos que para o foco teórico do nosso projeto, a viabilização e incorporação da Semântica na *Web*, um assunto de pesquisa relativamente novo, onde várias idéias têm surgido freqüentemente e onde, apesar de termos algumas definições muito fundamentadas e defendidas por vários pesquisadores, como é o caso do uso de ontologias, ainda não temos padrões que facilitem a representação dessas informações.

Como foram mostradas no capítulo II, outras novidades têm surgido para enriquecer ainda mais em detalhes a representação das ontologias, como é o caso do DAML+OIL e o mais recente OWL. Quanto mais rica em detalhes for a ontologia, maior expressão semântica ela terá e conseqüentemente serviços cada vez mais especializados poderão ser disponibilizados.

O campo de pesquisa de representação de ontologias continua oferecendo novidades freqüentes à busca de uma representação de conhecimento ideal. Mesmo assim, percebemos que o RDF tem se firmado cada vez mais como forte candidato a linguagem padrão para representação de metadados e podemos observar isso nas propostas DAML+OIL e OWL que têm sintaxe RDF.

Assim, acreditamos que o OntoEditor está no caminho certo, ou seja, gerando suas ontologias num forte candidato a padrão de representação. Entretanto, não podemos desconsiderar essas novas propostas que oferecem numeras propriedades para auxiliar na descrição das ontologias, enriquecendo-a de forma significativa. Por isso, concordamos que outras características poderão ser incorporadas ao OntoEditor, como gerar ontologias em OWL, além do RDF, para que ele seja melhorado, acompanhando o andamento evolutivo das pesquisas para representações das ontologias.



Além disso, a interface do OntoEditor poderia ser melhorada. Editores com o mesmo propósito do OntoEditor, como o Protégé, primam por oferecer interfaces gráficas ricas em funcionalidades que ajudam a flexibilizar cada vez mais a representação das ontologias para os usuários. Tal como esses editores e uma vez que estamos oferecendo uma nova forma visual de representação gráfica, na forma de grafos, novas funcionalidades poderiam ser integradas à interface do OntoEditor para que os seus usuários possam explorar ainda mais essa nova forma de representação. Como exemplo, podemos sugerir a implementação de níveis de visualização para possibilitar, aos usuários, a visualização de ontologias grandes sob vários subníveis que poderiam ser expandidos conforme fosse necessário.

Relacionado as consultas realizadas pelo OntoEditor, apesar de termos utilizado a linguagem XML-QL para consultar termos em documentos RDF e da mesma ter atendido de forma satisfatória os nossos propósitos, poderíamos adaptar a ferramenta para utilizar novas linguagens. Seguindo um quadro evolutivo para ferramenta, novas linguagens, com maior poder de expressão, como a Xquery, poderiam ser integradas ao OntoEditor para acompanhar necessidades de consultas ainda mais detalhadas e mais complexas, tornando, assim, a ferramenta cada vez mais robusta para os seus usuários.

## Referências Bibliográficas

- [Abiteboul et al. - 2000] Serge Abiteboul, Peter Buneman, Dan Suciu. Data on the Web: From Relations to Semistructured Data and XML. Morgan Kaufmann Publishers – San Francisco, California – 2000.
- [Ahmed et al. – 2001] Kal Ahmed, Danny Ayers, Mark Birbeck, Jay Cousins, David Dodds, Josh Lubell, Miloslav Nic, Daniel Rivers-Moore, Andrew Watt, Robert Worden, Ann Wrightson. Professional XML Meta Data. Wrox Press Ltd. EUA – 2001.
- [Arruda et al. – 2002] Ladjane Arruda, Cláudio de Souza Baptista, Carlos A. Lima. MEDIWeb: a Mediator-based Environment for Data Integration on the Web. In Proceedings of the Fourth International Conference on Enterprise Information Systems, Ciudad-Real, Spain, April 2002.
- [Bechhofer et al. – 2001] Sean Bechhofer, Ian Horrocks, Carole Goble e Robert Stevens. OilEd: a Reason-able Ontology Editor for the Semantic Web – 2001.
- [Benjamins et al. - 2000] V. Benjamins, D. Fensel, S. Decker, A. Pérez. (KA)2: Building Ontologies for the Internet: a Mid Term Report – 2000.
- [DAML+OIL] <http://www.daml.org>
- [Decker et al. – 2000] Stefan Decker, Sergey Melnik, Frank van Harmelen, Dieter Fensel, Michel Klein, Jeen Broekstra, Michael Erdmann, Ian Horrocks. The Semantic Web: The Roles of XML and RDF. IEEE Internet Computing. September-October 2000.
- [FaCT] The FaCT System – <http://www.cs.man.ac.uk/~horrocks/FaCT>.
- [Grubber – 1993] Tom Grubber. A Translation Approach to Portable Ontologies. 1993.
- [Guarino – 1996] Nicola Guarino. Undersatanding, Building and Using

- Ontologies. LADSEB-CNR, National Research Council – Padova, Italy – 1996.
- [Guarino – 199?] Nicola Guarino. Semantic Matching: Formal Ontological Distinctions for Information Organization, Extraction and Integration. Berlin: Springer Verlag, p. 139-170.
- [Handschuh et al. – 2001] Siegfried Handschuh, Alexander Maedche, Ljiljana Stojanovic and Raphael Volz. KAON — The Karlsruhe ONtology and Semantic Web Infrastructure – 2001.
- [Hjelm – 2001] Johan Hjelm. Creating the Semantic Web with RDF. John Wiley & Sons, Inc. EUA – 2001.
- [Horrocks et al. – 2000] I. Horrocks, D. Fensel, J. Broekstra, S. Decker, M. Erdmann, C. Goble, F. van Harmelen, M. Klein, S. Staab, R. Studer and E. Motta. The Ontology Inference Layer OIL. <http://www.ontoknowledge.org/oil>. - 2000.
- [HP - Jena] The Jena Semantic Web Toolkit - <http://www.hpl.hp.com/semweb/jena-top.html>.
- [Klein et al. - 2000] Michel Klein, Dieter Fensel, Frank van Harmelen, and Ian Horrocks. The Relation between Ontologies and Schema-languages: Translating OIL-specifications in XML-Schema.
- [Larman – 2000] Craig Larman. Utilizando UML e Padrões. Tradução: Luiz Augusto Meirelles Salgado. Bookman Companhia Editora – Porto Alegre, Rio Grande do Sul – 2000.
- [Mello-Heuser - 2000] R. Mello, C. Heuser. Aplicação de Ontologias a Dados Semi-Estruturados. Exame de Qualificação UFRGS - 2000.
- [Mello et al. – 2000] Ronaldo dos Santos Mello, Carina Friedrich Dorneles, Adrovane Kade, Vanessa de Paula Braganholo, Carlos Alberto Heuser. Dados Semi-Estruturados. XV Simpósio Brasileiro de Banco de Dados. João Pessoa – Paraíba – 2000.
- [Microsoft - SQLServer] Microsoft SQLServer - <http://www.microsoft.com/sql/default.asp>.
- [Moultis, Kirk – 2000] Natanya Pitts-Moultis, Cheryl Kirk. XML Black Book. Tradução: Ariosvaldo Griesi. Makron Books do Brasil – 2000.
- [Noy et al. – 2000] Natalya Noy Fridman, Ray Ferguson, Mark A. Musen.

- The knowledge model of Protégé-2000: combining interoperability and flexibility. International Conference on Knowledge Engineering and Knowledge Management (EKAW'2000), Juan-les-Pins, France. 2000.
- [Noy et al. – 2001] Natalya F. Noy, Michael Sintek, Stefan Decker, Mônica Crubézy, Ray W. Ferguson e Mark A. Musen. Creating Semantic Web Contents with Protege-2000. Stanford University – 2001.
- [OKBC] Open Knowledge Base Connectivity - <http://www.ai.sri.com/~okbc>.
- [OpenJGraph – 2000] Jesus Salvo. OpenJGraph - Java Graph and Graph Drawing Project. - <http://openjgraph.sourceforge.net>
- [Rapoza – 2000] Jim Rapoza – DAML could take search to a new level. New query language being developed for Defense Department helps agents understand content. PC Week Labs - 2000.
- [Silva-Heuser - 1999] SILVA, A.; HEUSER, C.; Materialização de Visões sobre Dados Semi-Estruturados - 1999. Disponibilizado por: <http://www.inf.ufrgs.br/pos/SemanaAcademica/Semana99/achilles/achilles.html>
- [Staab et al. - 2000] S. Stabb, J. Angele, S. Decker, M.Erdmann, A. Hotho, A. Maedche, H-P. Schnurr, R. Studer, Y. Sure. Semantic Community Web Portals. University of Karlsruhe. Germany.
- [Sun - A] Applets Java - <http://java.sun.com/applets>
- [Sun – B] JDBC Data Access API - <http://java.sun.com/products/jdbc>.
- [ThinWeb] ThinWeb Technologies Inc. - <http://www.thinweb.com/index1.html>.
- [W3C – OWLFeature] <http://lists.w3.org/Archives/Public/www-webont-wg/2002May/att-0173/01-owl.html#1.1>
- [W3C – OWLreq] <http://www.w3.org/TR/webont-req/>
- [W3C – OWLref] <http://www.w3.org/TR/owl-ref/>
- [W3C – OWLabs] <http://www.w3.org/TR/owl-absyn/>
- W3C – RDF] <http://www.w3.org/RDF/>

[W3C – RDFSHEMA]	<a href="http://www.w3.org/TR/rdf-schema/">http://www.w3.org/TR/rdf-schema/</a>
[W3C – XQUERY]	<a href="http://www.w3.org/XML/Query">http://www.w3.org/XML/Query</a>
[XML-Namespaces]	<a href="http://www.w3.org/TR/REC-xml-names/">http://www.w3.org/TR/REC-xml-names/</a>
[XML-QL]	<a href="http://www.research.att.com/~mff/xmlql/doc/sitegraph.genoid_1.html">http://www.research.att.com/~mff/xmlql/doc/sitegraph.genoid_1.html</a>
[SiRPAC]	SIRPAC: Simple RDF Parser and Compiler. <a href="http://www.w3.org/RDF/Implementations/SiRPAC">http://www.w3.org/RDF/Implementations/SiRPAC</a>