UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

Ontologias e Consulta Semântica: Uma Aplicação ao Caso Lattes

por

AILTON SERGIO BONIFACIO

Dissertação submetida à avaliação,
como requisito parcial para a obtenção do grau de Mestre
em Ciência da Computação

Prof. Dr. Carlos Alberto Heuser
Orientador

Porto Alegre, setembro de 2002

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Bonifacio, Ailton Sergio

Ontologias e Consulta Semântica: Uma Aplicação ao Caso Lattes / por Ailton Sergio Bonifacio. – Porto Alegre: PPGC da UFRGS, 2002.
85 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2002. Orientador: Heuser, Carlos Alberto.

1. Ontologias. 2. DAML+OIL. 3. Lattes. 4. Metadados. I. Heuser, Carlos Alberto. II. Título.

“Filhos, tenham sempre sabedoria e compreensão e nunca deixem que elas se afastem de vocês.” Pv. 3.21

Aos meus filhos Caio Augusto e Amanda Higa.

Agradecimentos

Sobre tudo e todos, agradeço a Deus. Antes de mim, é por causa dEle que este trabalho foi realizado. Foi Ele quem me deu sabedoria, discernimento, forças e amparo necessário nos momentos que precisei.

Sou eternamente grato aos meus Pais, Alexandre e Setsuko, que se enchem de alegria e orgulho com cada conquista de seus filhos.

Agradeço a cada um da minha família, que torceu e partilhou de cada passo deste trabalho; em especial aos meus irmãos Alex, Yuri e Adilson.

À Rosângela, mãe dos meus filhos, primeira incentivadora deste projeto. Toda minha gratidão e respeito.

Aos amigos Robertson Todão e Aparecido Vilela Junior pelo companheirismo e amizade. Obrigado pelas orações.

Ao orientador Prof. Dr. Carlos Alberto Heuser, que mais do que orientar, soube conduzir e aconselhar. Obrigado Professor Heuser, pela confiança e paciência; e pelo tratamento cordial e amigo.

Ao Prof. Dr. Tiaraju Asmuz Diverio pelas palavras de incentivo e pelas dicas, nos rápidos encontros nos corredores do Instituto de Informática.

Aos amigos do mestrado, pela amizade constituída. A cada um deles que de uma forma ou de outra estiveram presentes e contribuíram para o sucesso deste trabalho. Em especial aos amigos Marcelo Costa Isolani e Yandre Maldonado e Gomes da Costa, este último, parceiro do início ao fim literalmente.

Aos amigos do Instituto de Informática da UFRGS, em especial ao pessoal da "sala 215". Obrigado pela ajuda neste período final. Abraços a todos.

À Universidade Estadual de Londrina pelo apoio e incentivo a esta capacitação.

Aos amigos da Coordenadoria de Recursos Humanos da UEL pelo incentivo e amizade, e por compreenderem minhas prioridades.

Sumário

Lista de Abreviaturas

ARP	Another RDF Parser
CNPq	Conselho Nacional de Desenvolvimento Científico e Tecnológico
DAML	DARPA Agent Markup Language
DC	Dublin Core
DCQ	Dublin Core Qualifiers
DL	Description Logic
DTD	Data Type Definition
ER	Entidade e Relacionamento
HTML	Hypertext Markup Language
IA	Inteligência Artificial
LMPL	Linguagem de Marcação para Plataforma Lattes
OIL	Ontology Inference Layer
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
RDQL	RDF Data Query Language
RQL	RDF Query Language
SiRPAC	Simple RDF Parser & Compiler
SQL	Structure Query Language
URI	Uniform Resource Identifiers
W3C	World Wide Web Consortium
XML	Extensible Markup Language

Lista de Figuras

Lista de Tabelas

Resumo

A quantidade e diversidade dos dados disponíveis na Web aumentam constantemente. Os motores de busca disponíveis, que usam palavras-chave, fornecem ao usuário resultados inexatos. Atualmente, os sistemas convencionais de consultas utilizam técnicas de base sintática. As pesquisas voltam-se para o estudo de abordagens de consultas através de conceitos, permitindo a recuperação semântica. Neste sentido, algumas propostas envolvem a criação de metadados que seguem modelos de ontologias.

O propósito deste trabalho é apresentar, avaliar e permitir uma melhor compreensão de um conjunto de conceitos, linguagens e ferramentas que são usadas na Web Semântica. Dentre elas, linguagens para construção de ontologias e linguagens para consultas; além das ferramentas associadas que objetivam o armazenamento, manutenção e anotação em ontologias.

Para atingir este propósito, estas linguagens e ferramentas são aplicadas a um caso de dimensão e complexidade realistas, o Currículo Lattes. O trabalho apresenta um modelo de metadados com semântica para o Currículo Lattes. Este modelo é baseado numa ontologia especificada na linguagem DAML+OIL. Além disso, é apresentada uma avaliação dos métodos de instanciação desta ontologia. Uma avaliação dos métodos e/ou linguagens de consulta mais adequadas para a consulta semântica das informações também é apresentada.

Palavras-chave: Ontologias, DAML+OIL, Lattes, Metadados e Web Semântica.

TITLE: "ONTOLOGY AND SEMANTIC QUERY: AN APLICATION AT LATTES-*CURRICULUM* CASE**"**

Abstract

The amount and diversity of the available data in the Web constantly increase. The available full text search engines, which are keyword-based, provide the user with inaccurate results. Actually, the conventional systems of queries use syntactic-based techniques. The focus of researches have been are addressed for studying the queries statements built from concepts and their relationships, allowing the semantic retrieval. In this sense, some proposals involve the metadata creation that follow ontologies models.

The purpose of this work is to present, to evaluate and to allow a better understanding of a group of concepts, languages and tools that are used in the Semantic Web. Among them, languages for ontology construction and quey languages; besides the associated tools that aims the storage, maintenance and annotation in ontologies.

To reach this purpose, these languages and tools will be applied to a case of dimension and complexity realists, the Lattes Curriculum. The work presents a metadata model with semantics for the Lattes Curriculum. This model is based on an ontology specified in the DAML+OIL language. Besides, an evaluation of the methods of instatiation of this ontology is presented. An evaluation of the methods and/or query languages more adapted for the semantic query of the information is presented too.

Keywords: Ontology, DAML+OIL, Lattes, Metadata and Semantic Web.

1 Introdução

A quantidade e diversidade dos dados disponíveis na Web aumentam constantemente. Para buscar informações, os usuários dispõem de mecanismos de busca que retornam resultados que não os satisfazem completamente. Da mesma forma, as Bibliotecas Digitais necessitam destes recursos de busca. Uma biblioteca digital permite que os usuários interajam com grandes quantidades de informações. Estes sistemas de informação suportam busca e exibição de componentes de coleções organizadas. Assim sendo, Recuperação de Informações tornou-se um tópico central em Bibliotecas Digitais. As pesquisas voltam-se para as consultas através de conceitos, permitindo a recuperação semântica.

Atualmente, os sistemas convencionais de consultas utilizam técnicas de base sintática sobre uma forma de adequação léxica, mais do que uma aplicação da base de conhecimento do campo de interesse. Em muitos domínios, o usuário está interessado em encontrar informação onde a relevância dos documentos não pode ser medida através do uso de sistemas de busca por palavras chaves (Keywords) **[BEK 98]**. A relevância deve ser preferencialmente estimada num nível de conhecimento profundo do domínio de problema em questão. Neste sentido, algumas propostas envolvem a criação de metadados que seguem modelos de ontologias. Uma ontologia **[GRU 93]** é um conjunto de termos hierarquicamente estruturado para a descrição de um domínio que pode ser usado como um esqueleto fundamental para uma base de conhecimento.

As ontologias, entre outras coisas, colaboram no sentido de se obter uma Web onde os recursos disponíveis são acessíveis não somente por seres humanos, mas também por processos automatizados. Esta automação depende da elevação do status da Web de “machine-readable” (lida automaticamente) para algo que é chamado de “machine-understandable” (entendida automaticamente). Isto denota a visão da Web Semântica **[HOR 2002, DEC 2000, BER 2001]**.

A necessidade de linguagens e ferramentas para construção, anotação, consulta e integração de ontologias é incontestável. Entretanto, somente a representação de conhecimento e informação não é suficiente. Pessoas e/ou agentes da Web que buscam informações necessitam usar e consultar ontologias e os recursos inerentes a elas. Com isso, fazem com que a necessidade de ferramentas de armazenamento e consultas em ontologias surjam. Novas linguagens para pesquisa de metadados baseados sobre padrões têm surgido para capacitar a aquisição de conhecimento das fontes dispersas de informações.

O propósito deste trabalho é apresentar, avaliar e permitir uma melhor compreensão de um conjunto de conceitos, linguagens e ferramentas que são usadas na Web Semântica. Dentre elas, linguagens para construção de ontologias e linguagens para consultas; além das ferramentas associadas que objetivam o armazenamento, manutenção e anotação em ontologias. Esta lista de linguagens e ferramentas obviamente não é completa. Portanto, são apenas indicativos da tendência para se fornecer suporte total ao armazenamento e consultas para os padrões de metadados/ontologias baseados na Web, tal como RDF **[W3C 99]**, RDFS **[W3C 2000a]** e DAML+OIL **[DAM 2001]**.

Para atingir este propósito, estas linguagens e ferramentas serão aplicadas a um caso de dimensão e complexidade realistas, o Currículo Lattes. Similarmente a uma Biblioteca Digital, porém uma biblioteca de currículos, a Plataforma Lattes **[LAT 2001]** do CNPq oferece uma grande quantidade de diferentes tipos de informações acerca dos pesquisadores, docentes e alunos ali cadastrados. Assim sendo, faz-se necessário prover seus usuários de métodos e formatos que disponibilizem metadados com características semânticas que permitam procedimentos de recuperação de informações cada vez mais eficientes; assim como prover mecanismos que habilitem e facilitem consultas às informações através de agentes da Web.

O trabalho apresenta um modelo de metadados com semântica, baseados em ontologias, para o Currículo Lattes. Além disso, é apresentada uma avaliação dos métodos de instanciação desta ontologia e dos métodos e/ou linguagens de consulta (*query languages*) mais adequadas para a consulta semântica das informações.

DAML+OIL deve ser visto como uma alternativa a mais para a representação do Sistema de Currículos Lattes. A adição de um modelo conceitual DAML+OIL para currículos, neste caso, o Lattes, pode enriquecer a relação de expressividade para aumentar o já existente padrão XML proporcionado pela comunidade LMPL.

A estrutura do trabalho é descrita a seguir.

No capítulo 2 são apresentados os conceitos e as relações entre metadados semânticos, ontologias e a Web Semântica **[HOR 2002, DEC 2000, BER 2001]**.

O terceiro capítulo apresenta uma explanação sobre a linguagem DAML+OIL. A relação de dependência entre XML, RDF e DAML+OIL, as ferramentas mais utilizadas para a construção de ontologias e as linguagens de consulta (*query languages*) utilizadas, também são apresentadas.

No capítulo 4, é apresentada uma ontologia em DAM+OIL para o Lattes, bem como os detalhes das etapas de desenvolvimento, instanciação e consultas sobre esta ontologia.

Finalmente, no capítulo 5, são apresentadas as conclusões deste trabalho e as indicações para trabalhos futuros.

2 Metadados Semânticos, Ontologias e a Web Semântica

2.1 Ontologias

O termo “ontologia” é emprestado da filosofia, onde uma Ontologia é uma descrição da Existência. Na Ciência da Computação, vem sendo aplicado desde o início da década de 90 na área de inteligência artificial para representação computacional de conhecimento em áreas como engenharia de conhecimento e processamento de linguagem natural. Para sistemas de Inteligência Artificial, o que “existe” é o que pode ser representado. Neste contexto, podem ser entendidas como uma especificação formal e explícita de uma conceitualização consensual **[GRU 93]**.

A noção de conceitualização exige, entretanto, uma adequada formalização, visto que ela pode gerar algumas confusões. Nesta definição de Tom Gruber **[GRU 93]** , a conceitualização é definida com uma estrutura <D,R>, onde D é um domínio e R é um conjunto de relações sobre D.

O corpo de um conhecimento representado formalmente é baseado numa conceitualização: os objetos, conceitos e outras entidades que são assumidas para existirem em alguma área de interesse e os relacionamentos que são organizados dentro delas **[GEN 87]**. Uma conceitualização é uma visão abstrata e simplificada do mundo que se quer representar para algum propósito. Toda base de conhecimento - sistema baseado em conhecimento que representa conhecimento - está comprometida com alguma conceitualização, explícita ou implícita. Uma ontologia é uma especificação explícita de uma conceitualização.

Usam-se ontologias comuns para descrever a representação de uma base de conhecimento para um conjunto de agentes. Desta forma, estes agentes podem passar informações sobre o domínio do discurso, sem necessariamente operar sobre uma teoria globalmente compartilhada. O Conhecimento é atribuído aos agentes através da observação de suas ações. Um agente “sabe” algo se ele age como se ele tivesse a informação e está agindo racionalmente para alcançar seus objetivos. As “ações” dos agentes - incluindo servidores de base de conhecimento e sistemas baseados em conhecimento - podem ser vistos através de uma ordem e perguntas da interface funcional. Nesta interface, um cliente interage com um agente fazendo afirmações lógicas e propondo perguntas. Pragmaticamente, um ontologia comum define o vocabulário com a qual as pesquisas e afirmações são trocadas entre os agentes.

Para representar uma conceitualização é necessário uma linguagem de representação. Muitas linguagens e sistemas de representação são definidos.

Entretanto, para aplicações sobre a Web, é importante ter uma linguagem com uma sintaxe padronizada (XML) **[MEL 2000]**. Este requerimento leva novamente às linguagens baseadas em XML, definindo-as sobre o topo do XML. Um exemplo é Resource Description Framework Schema Language (RDFS) **[W3C 2000a]**, entre outras. Todos eles usam a sintaxe do XML, mas com leves diferenças nas *tag* names.

Uma outra proposta que estende RDF e RDF Schema é o OIL (Ontology Inference Layer) **[FEN 2000]**. RDF e RDFS já estão em uso na comunidade bibliotecária e já estão sendo aceitos como um padrão. Um sucessor do OIL é o DAML+OIL, desenvolvido conjuntamente por um grupo de cientistas europeus e americanos.

2.2 Web Semântica

A visão da Web Semântica **[HOR 2002, DEC 2000, BER 2001]** é a de uma Web no qual os recursos disponíveis são acessíveis não somente por seres humanos, mas também por processos automatizados. Estes processos podem ser agentes que percorrem a Web e executam tarefas que objetivam a melhora das buscas em termos de precisão, a descoberta de recursos e a recuperação e filtragem de informações. A automação de tarefas depende da elevação do status da Web de "machine-readable" (lida automaticamente) para algo que é chamado de "machine-understandable" (entendida automaticamente). A idéia chave é ter dados sobre a Web. Estes dados devem ser definidos e ligados de tal forma que seus significados sejam, preferencialmente, interpretados explicitamente por processos de software ao invés de interpretados implicitamente por seres humanos.

Para se atingir este objetivo, torna-se necessário a anotação de recursos sobre a Web através de metadados. Na verdade, seria ideal, se houvesse a possibilidade de se anotar ou criar estes metadados com semântica, que proveria alguma indicação dos conteúdos de um recurso. Faz-se necessário, então, o uso de linguagens que suportem a representação de metadados semânticos. Muitas propostas de padronização de linguagens para metadados já são reconhecidas na W3C. Dentre elas, Resource Description Framework (RDF) **[W3C 99]** e RDF Schema (RDFS **[W3C 2000a]**). Entretanto, a menos que elas compartilhem um entendimento comum em relação a seu significado, estas linguagens possuem valor limitado para processos automatizados. As ontologias podem colaborar muito para o preenchimento deste requisito. Elas fornecem uma representação de uma conceitualização compartilhada de um domínio particular, além de um vocabulário controlado compartilhado que pode ser comunicado através das pessoas e aplicações **[BER 99]**.

RDF, com o poder de uma Web Semântica, será uma linguagem completa, capaz de expressar paradoxo e tautologia[1]. Desta forma, serão possíveis as perguntas de frases cujas respostas deveriam exigir de uma máquina uma busca por toda a Web e uma quantia inimaginável de tempo para solucioná-la.

Isto não impede que seja feita uma linguagem completa. Cada aplicação mecânica de RDF usará um *schema* para restringir seu uso de RDF para uma linguagem deliberadamente limitada. Porém, quando são feitos vínculos entre os nodos de RDF, o resultado é uma expressão de uma quantia enorme de informação. Está claro que, pelo fato da Web Semântica poder incluir todos os tipos de dados para representar o mundo, a própria linguagem deve ser completamente expressiva.

2.2.1 A Web Semântica e os Bancos de Dados Relacionais

O modelo de dados da Web Semântica é similar ao modelo de bancos de dados relacionais **[BER 98]**. Um banco de dados relacional consiste de tabelas, que consistem de linhas ou registros. Cada registro consiste de um conjunto de campos. O registro não é nada mais que o conteúdo de seus campos, da mesma maneira que um nodo de RDF não é nada mais que as conexões: os valores de propriedades. O mapeamento é muito direto:

- Um registro é um nodo RDF
- O nome do campo (coluna) é o tipo de propriedade RDF; e
- O campo do registro (célula da tabela) é o valor.

[1] Repetição das mesmas idéias desnecessariamente.

De fato, uma das principais forças de direção da Web Semântica tem sido a expressão de uma vasta quantia de informações de bancos de dados relacionais sobre a Web, de uma forma que possam ser processadas automaticamente.

O formato de serialização do RDF - sua sintaxe em XML - é muito adequado para expressar informações de bancos de dados relacionais.

Sistemas de banco de dados relacionais gerenciam dados RDF, mas de um modo especializado. Em uma tabela, existem muitos registros com o mesmo conjunto de propriedades.

A Web Semântica não foi projetada somente como um novo modelo de dados - ela é especificamente apropriada para a ligação de dados de diferentes modelos. Uma das mais importantes coisas que ela permite, é adicionar informações relacionando diferentes bancos de dados sobre a Web. Isto permite que sofisticadas operações sejam executadas através delas.

2.3 Metadados Semânticos

A mais breve definição de metadados é “dado sobre dado”. Neste contexto, metadado refere-se a alguma estrutura descritiva da informação sobre outro dado; que é usado para ajudar na identificação, descrição, localização e gerenciamento de recursos. Entretanto, eles podem ser aplicados em qualquer meio.

Metadados podem possuir a classificação de estrutural ou semântico. Metadado estrutural representa a informação que descreve a organização e estrutura dos dados gravados. Por exemplo, informações sobre o formato, os tipos de dados usados e os relacionamentos sintáticos entre eles. Em contraste, metadados semânticos fornecem informações sobre o significado dos dados disponíveis e seus relacionamentos semânticos. Por exemplo, dados que descrevem o conteúdo semântico de um valor de dado (como unidades de medida e escala), ou dados que fornecem informações adicionais sobre sua criação (algoritmo de cálculo ou derivação da fórmula usada), linhagem dos dados (fontes) e qualidade (atualidade e precisão). Neste sentido, é desejável uma conceitualização de um domínio específico de problema, ou ontologias que forneçam um acordo comum de vocabulários, para que os dados sejam referenciados. Assim, uma ontologia serve como uma base comum para a representação de dados e metadados.

Um objeto semântico representa um item de dado junto com sua base de contexto semântico que consiste de um conjunto flexível de meta-atributos que explicitamente descrevem a compreensão implícita sobre o significado do item de dado. Adicionalmente, cada objeto semântico possui um rótulo de conceito associado a ele, que especifica o relacionamento entre o objeto e os aspectos do mundo real que ele descreve. Estes rótulos são adquiridos de uma Ontologia.

Um domínio específico de ontologias pode ser utilizado para assegurar a correta interpretação dos metadados disponíveis. Uma ontologia fornece um entendimento sobre uma conceitualização compartilhada de um determinado domínio de aplicação. Os conceitos específicos numa ontologia fornecem um vocabulário comum para que nenhuma negociação adicional seja necessária.

Metadados semânticos representam um significante papel no contexto da Web Semântica (Semantic Web) **[HOR 2002, DEC 2000, BER 2001]** e, por consequência, em qualquer outro assunto relacionado a disponibilidade e acesso de dados e recursos na Web. Dentre outros, comércio eletrônico e bibliotecas digitais. Agentes necessitam de metadados que descrevem o contexto de recursos, a fim de executarem operações tais como consultas sobre estes recursos. Além disso, se uma rica semântica é fornecida,

estes agentes podem então aplicar “raciocínio” (reasoning) sobre os metadados, melhorando seu poder de processamento. Neste sentido, o uso da linguagem DAML+OIL é de grande importância. Assim, vários assuntos que dizem respeito a linguagem DAML+OIL serão analisados a fim de se atingir o objetivo de se obter metadados semânticos.

2.4 Consulta Semântica

As pessoas pelo mundo contribuem com informações de vários tipos e formatos sobre a Web. Embora esta contribuição de informações seja farta e simples, encontrar informações úteis é bem mais difícil.

A consulta semântica é uma parte da visão da Web Semântica. Ao contrário das consultas por palavras-chave, existentes atualmente na maioria dos serviços de busca, a consulta semântica ajuda o usuário encontrar informações de acordo com o conceito. Dois documentos podem descrever o mesmo conceito, mas não compartilham nenhuma palavra chave. Estes documentos tem proximidade conceitual, ou significado similar, mas não seriam incluídos no resultado de uma simples consulta por palavras-chave. Uma consulta semântica olha para o significado e poderia, então, incluir os dois documentos no resultado da busca.

No entanto, a construção semântica da base de conhecimento não pode ser forçada sem o uso de ferramentas correspondentes. Uma destas ferramentas é a aplicação das ontologias.

3 Linguagens e Ferramentas para Manipulação de Ontologias

A necessidade de linguagens e ferramentas para construção, anotação, consulta e integração de ontologias é incontestável. Entretanto, somente a representação de conhecimento e informação não é suficiente. Pessoas e/ou agentes da Web que buscam informações necessitam usar e consultar ontologias e os recursos inerentes a elas. Com isso, fazem com que a necessidade de ferramentas de armazenamento e consultas em ontologias surjam. Este contexto tem mudado devido a grande aceitação e uso da Web como plataforma de comunicação de conhecimento. Novas linguagens para pesquisa de metadados baseados sobre padrões têm surgido para capacitar a aquisição de conhecimento das fontes dispersas de informações. Enquanto isso, as técnicas tradicionais de armazenamento em banco de dados têm sido adaptadas para lidar com as peculiaridades de dados semi-estruturados sobre a Web.

Nesta seção, o propósito é apresentar e avaliar brevemente as linguagens e ferramentas que serão utilizadas neste trabalho. Dentre elas, linguagens para construção de ontologias e linguagens para consultas; além das ferramentas associadas que objetivam o armazenamento, manutenção e anotação em ontologias. Esta lista de linguagens e ferramentas obviamente não é completa. Portanto, são apenas indicativos da tendência para se fornecer suporte total ao armazenamento e consultas para os padrões de metadados/ontologias baseados na Web, tal como RDF **[W3C 99]**, RDFS **[W3C 2000a]** e DAML+OIL **[DAM 2001]**. As tecnologias de buscas habilitadas pelo RDF têm o potencial de prover uma melhora significante sobre os motores baseados em palavras-chave ou consultas por navegação de assuntos, especialmente quando se trata de consulta e navegação conceitual **[OSS 2002]**.

Para complementar, esta orientação facilita o entendimento das técnicas utilizadas no desenvolvimento e utilização do modelo conceitual para o Currículo Lattes.

3.1 Linguagens para modelos

3.1.1 XML e XML Schema

XML Schema é uma linguagem para modelos utilizada para dar estrutura a documentos XML. A especificação de XML Schema assume que pelo menos dois documentos XML são utilizados: um documento instância e pelo menos um documento esquema. O documento instância contém a informação propriamente dita e o documento esquema descreve a estrutura e tipo do documento instância. A distinção entre instância e esquema é semelhante à distinção entre objeto e classe em linguagens de programação orientadas a objeto. Uma classe descreve um objeto assim como um esquema descreve um documento instância **[BRV 2001]**.

Em XML Schema, o modelo de conteúdo de um elemento pode ser especificado a partir da declaração de um tipo. Um tipo pode ser Simples ou Complexo. Um tipo simples pode ser atribuído a um atributo ou a um elemento simples, que possui somente texto e não possui elementos filhos. Já um tipo complexo é utilizado para dizer quais são os subelementos permitidos para um determinado elemento. Tipos simples são declarados com simpleType, e tipos complexos com complexType. A declaração element liga um tipo a um nome de elemento. Elementos podem ser declarados dentro de um tipo complexo ou abaixo de schema. Neste último caso, são considerados elementos globais.

Um esquema pode ser definido por um prefixo associado a um *namespace*. Esse prefixo aparece quando um tipo definido neste esquema é utilizado para declarar um elemento ou atributo. O uso de *namespaces* aumenta a flexibilidade de XML Schema, permitindo a reutilização de definições feitas em outros esquemas. Pode-se também redefinir tipos declarados em um determinado namespace.

XML **[W3C 2000]**, além de incluir informação semântica, já é uma sintaxe comum que muitas ferramentas usam. Além disso, considera-se a efetiva adoção do XML como padrão para intercâmbio de informações na Web **[MEL 2000]**. Entretanto, com o advento da Web Semântica **[BER 99, BER 2001]**, XML mostra algumas limitações. Em **[DEC 2000]** é visto que a maior limitação do XML é que ele só descreve gramática. Tem-se a liberdade para definir e usar *tags*, atributos e outras primitivas da linguagem de um modo arbitrário, designando diferentes semânticas para descrever o modelo do domínio conceitual que se deseja. No entanto, visto que XML não impõe regras para tal descrição e que há muitos meios de denotar coisas semanticamente equivalentes, torna-se difícil a reconstrução do significado semântico de um documento XML.

RDF e RDFS **[DEC 2000, BRV 2001]** possuem recursos para descrever tanto metadados descritivos como metadados semânticos. Com estas linguagens, são possíveis as descrições de domínios de ontologias, através da identificação de hierarquias de conceitos e relações, juntamente com axiomas que podem ser usados para produzir novos fatos a partir de um já existente **[VDO 2001]**.

A linguagem DAML+OIL **[DAM 2001]** foi desenvolvida como uma extensão para XML e RDF. A última versão desta linguagem provê um rico conjunto de construções que objetivam a criação de ontologias e a marcação de informações de forma que sejam entendidas e lidas automaticamente. DAML+OIL provê uma infra-estrutura básica que permite às máquinas fazerem a mesma classificação de algumas simples inferências que os seres humanos fazem. É só o início, mas é uma base importante para uma Web de informações que as máquinas poderão extrair.

3.1.2 RDF e RDF Schema

RDF Schema fornece meios para a definição de vocabulários, estruturas e restrições (*constraints*) para expressar metadados sobre os recursos da Web. Entretanto a semântica formal para as primitivas definidas em RDF Schema não são fornecidas. Sendo assim, a expressividade destas primitivas não são suficientes para a completa modelagem e raciocínio ontológico. Para a execução destas tarefas, uma camada adicional no topo do RDF Schema é necessária. Esta arquitetura de extensão em camadas é chamada de Semantic Web **[BER 2001]**.

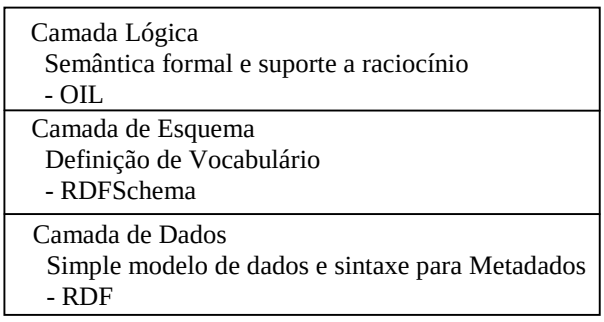

FIGURA 3.1 - A arquitetura de três camadas da Semantic Web

Um mecanismo genérico para expressar a semântica dos dados processáveis automaticamente é requerido até mesmo na camada de dados da Web Semântica. A

linguagem RDF é a base para este processamento de metadados, fornecendo um simples modelo de dados e uma sintaxe padronizada para metadados. Basicamente, ele provê a linguagem para a escrita das declarações fatuais. A próxima camada é a camada de Esquema, fornecida pela especificação RDF Schema. Uma linguagem de representação de conhecimento formal pode ser usada como a terceira camada, a camada Lógica. Esta linguagem pode ser o OIL, que será visto a seguir na seção 3.1.3.

O modelo básico de dados consiste de três tipos de objetos:

- Recursos (*Resources*): pode ser uma página inteira da Web ou uma parte dela; uma coleção toda de páginas; ou um objeto que não é diretamente acessível via Web, por exemplo, um livro impresso. Os recursos são também chamados de URIs.
- Propriedades (*Properties*): são aspectos específicos, características, atributos ou relações utilizadas para descreverem os Recursos.
- Declarações (*Statements*): um recurso específico junto com uma propriedade, mais o valor daquela propriedade para aquele recurso é uma declaração RDF.

Estas três partes individuais de uma declaração são chamados respectivamente de Sujeito, Predicado e Objeto. Em poucas palavras, RDF define triplas objeto-propriedade-valor como primitivas básicas de modelagem e introduzem uma sintaxe padrão para elas.

A modelagem das primitivas oferecidas por RDF são muito básicas. Entretanto, RDF Schema define além das primitivas de modelagem de RDF. Exemplos são classes, subclasses, restrições de domínio (domain) e alcance (range) para propriedades e subpropriedades. Com estas extensões, RDF Schema chega perto das linguagens para ontologias.

A despeito da similaridade nos seus nomes, RDF Schema representa um papel diferente daquele que XML Schema tem. XML Schema e DTDs prescrevem a ordem e combinação das tags num documento XML. Em contraste, RDF Schema somente provê informações sobre a interpretação das declarações dadas em um modelo de dados RDF. Entretanto, a definição de OIL em RDFS, não proverá restrições (*constraints*) sobre a estrutura de uma ontologia OIL real. Isto poderá ser visto com detalhes em **[BRO 2000]**.

3.1.2.1 Modelo de Dados do RDF Schema

As primitivas de modelagem do RDF Schema, sua hierarquia subclass-of e seus relacionamentos, são apresentados nas figuras 3.2 e 3.3 respectivamente. Na sequência, a apresentação das primitivas de modelagem do RDF Schema, onde:

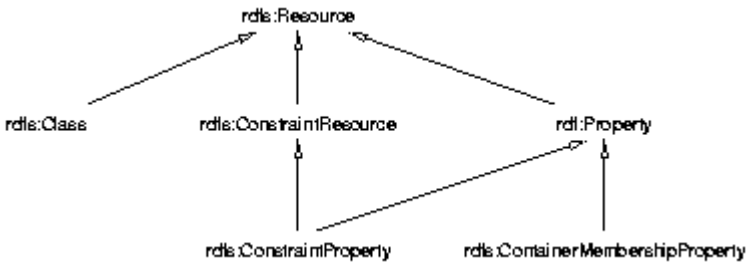

FIGURA 3.2 - A hierarquia subclass-of das primitivas de modelagem em RDFS

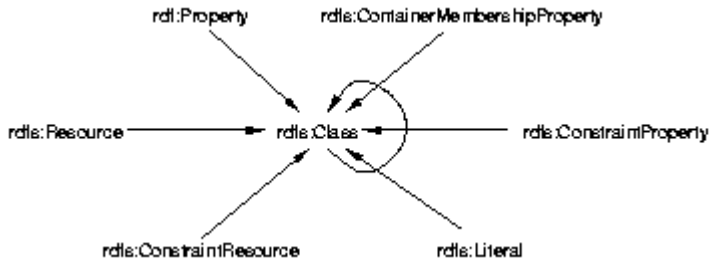

FIGURA 3.3 - Os relacionamentos das primitivas de modelagem em RDFS

- Classes Centrais (*Core Classes*) são: *rdfs:Resource*, *rdf:Property* e *rdfs:Class*. Tudo que é descrito pelas expressões RDF é visto como um instância da classe *rdfs:Resource*. A classe *rdf:Property* é a classe de todas as propriedades usadas para caracterizar instâncias de *rdfs:Resource*, por exemplo, cada *slot*/relação é uma instância de *rdf:Property*. Finalmente, *rdfs:Class* é usada para definir conceitos em RDFS, por exemplo, cada conceito deve ser uma instância de *rdfs:Class*.

- Propriedades Centrais (*Core properties*) são: *rdf:type*, *rdfs:subClassOf* e *rdfs:subPropertyOf*. A relação *rdf:type* modela instâncias de (*instance-of*) relacionamentos entre recursos e classes. Um recurso pode ser uma instância de mais do que uma classe. A relação *rdfs:subClassOf* modela a classificação hierárquica entre classes e é supostamente transitiva. Novamente, uma classe pode ser subclasse de muitas outras classes. Entretanto, uma classe não pode ser subclasse de sua própria classe e nem uma subclasse de sua própria subclasse. A relação *rdfs:subPropertyOf* modela a classificação hierárquica entre propriedades.

- Restrições Centrais (*Core constraints*) são: *rdfs:ConstraintResource*, *rdfs:ConstraintProperty*, *rdfs:range* e *rdfs:domain*. O primeiro define a classe de todas as restrições. A segunda é um subconjunto da primeira e de *rdf:Property* cobrindo todas as propriedades que são usadas para definir restrições. Até o momento ele tem duas instâncias; *rdfs:range* e *rdfs:domain* que são usadas para restringir alcance e domínio das propriedades. Não é permitido expressar duas ou mais restrições de alcance (*range*) sobre uma propriedade. Para domínios (*domains*) isto não é forçado e é interpretado como a união de domínios.

3.1.3 OIL

A linguagem OIL **[FEN 2000]** foi projetada de forma que:

- forneça a maioria das primitivas de modelagem comumente usadas em ontologias orientadas a *Frame* e Description Logic (DL);
- tenha uma semântica simples, clara e bem definida;
- proporcione automático suporte a raciocínio; por exemplo, consistência de classe e controle de classificação. O sistema FaCT (FaCT System) **[HOR 98]**, um raciocinador DL desenvolvido pela Universidade de Manchester, pode ser usado para este objetivo.

ontology-container
title "African Animals"
creator "Ian Horrocks"
subject "animal, food, vegetarians"
description "A didactic example ontology describing African animals"
description.release "1.01"
publisher "I. Horrocks"
type "ontology"
format "pseudo-xml"
format "rdf"
identifier "http://www.ontoknowledge.org/oil/rdfs-oil.pdf"
source "http://www.africa.com/nature/animals.html"
language "en-uk"
ontology-definitions
slot-def *eats*
inverse *is-eaten-by*
slot-def *has-part*
inverse *is-part-of*
properties transitive
class-def animal
class-def plant
subclass-of NOT animal
class-def tree
subclass-of plant
class-def plant
slot-constraint *is-part-of*
has-value tree
class-def leaf
slot-constraint *is-part-of*
has-value branch
class-def defined carnivore
subclass-of animal
slot-constraint *eats*
value-type animal
class-def defined herbivore
subclass-of animal, **NOT** carnivore
slot-constraint *eats*
value-type
plant **OR**
slot-constraint *is-part-of*
has-value plant
class-def giraffe
subclass-of herbivore
slot-contraint *eats*
value-type leaf
class-def lion
subclass-of animal
slot-constraint *eats*
value-type herbivore
class-def tasty-plant
subclass-of plant
slot-constraint *is-eaten-by*
has-value herbivore,carnivore

FIGURA 3.4 - Um exemplo de uma ontologia em OIL.

Uma ontologia em OIL é representada por um *ontology container* e uma *ontology definition*. Para o *container*, foram adotados os componentes definidos por *Dublin Core Metadata Element Set*, versão 1.1 **[DUB 2000]**.

A *ontology-definition* consiste de uma declaração *import* (opcional), uma *rule-base* (opcional) e definições de classes e *slots* .

Uma definição de classe (*class-def*) associa um nome de classe a sua descrição. Esta descrição de classe consiste do tipo da definição, de uma declaração *subclass-of* e zero ou mais restrições de *slots* (*slot constraint*). O tipo da definição pode ser, ou primitiva, quais meios que as condições declaradas para a classe são necessárias mas não suficientes; ou definidas, quais meios que estas condições são tanto necessárias quanto suficientes.

O valor da declaração *subclass-of* é uma expressão de classe (*class expression*). Isto pode ser um nome de classe, uma restrição de *slot* ou uma combinação booleana das expressões da classe usando os operadores AND, OR and NOT, com semântica padrão das DLs.

Uma restrição de *Slot* (*slot-constraint*) pode também ser chamada de uma função (*role*) ou de um atributo. Ela é uma lista de um ou mais restrições aplicadas ao *slot*. Algumas típicas restrições:

- *has-value (class-expr):* cada instância da classe definida pela restrição do *slot*, deve ser ligada, pela relação do *slot*, a instância de cada classe na lista.
- *Value-type (class-expr):* se uma instância da classe definida pela restrição do *slot* é ligada pela relação do *slot* a algum objeto *x*, então *x* deve ser uma instância de cada classe da lista.
- *Max-cardinality n (class-expr):* uma instância da classe definida pela restrição do *slot* pode ser ligada a no máximo *n* instâncias distintas da classe pela relação do *slot* (também para *min-cardinality*).

Uma definição de *slot* (*slot-def*) associa um nome de *slot* a uma definição de *slot*. Uma definição de *slot* especifica restrições globais que se aplicam a relação do *slot*. Um *slot-def* pode consistir de uma declaração subslot-of, de restrições “domain” e “range”, e características adicionais do *slot*, tal como *slot* inverso (*inverse slot*), transitivo (*transitive*) e simétrico (*symmetric*).

OIL é uma camada de representação e inferência para ontologias. Esta camada unifica três importantes aspectos fornecidos por diferentes comunidades: semântica formal e suporte a raciocínio, como os fornecidos pelas Description Logics **[HOR 99]**; ricas primitivas de modelagem, como as fornecidas pela comunidade *Frame*; e uma proposta padrão para troca sintática de notações como fornecidas pela comunidade Web. No entanto, para melhorar a representação de conhecimento formal da terceira camada da Web Semântica, surje a linguagem DAML+OIL.

Assim como OIL, a sintaxe da linguagem DAML+OIL é orientada em direção a XML e RDF.

3.1.4 DAML+OIL

DAML+OIL é uma linguagem ontológica, e como tal, foi definida para descrever a estrutura de um domínio. DAML+OIL assume uma abordagem orientada a objetos, com a estrutura do domínio sendo descrita em termos de classes e propriedades. Uma ontologia consiste de um conjunto de axiomas que são declarados; por exemplo, classificação de relacionamentos entre classes e propriedades.

De um ponto de vista formal, DAML+OIL pode ser visto como uma linguagem equivalente a expressiva descrição lógica SHIQ **[HOR 99, HOR 2001]**, com o acréscimo de classes definidas existencialmente (ou seja, o construtor *oneOf*) e tipos de dados (frequentemente chamados de domínios concretos em DLs). Tal como em uma DL, as classes DAML+OIL podem ser nomes (*URIs*) ou expressões e uma variedade de construtores são fornecidos para a construção de expressões de classes. O expressivo poder da linguagem é determinado pelos construtores de classes (e propriedades) suportados, e pelos tipos de axiomas suportados.

3.1.4.1 Histórico

Em **[HOR 2002a]**, é relatado que em 1999, o programa *DARPA Agent Markup Language* (DAML) foi iniciado com o objetivo de fornecer bases para uma nova geração: A Web Semântica. A adoção de uma linguagem ontológica comum facilitaria a interoperabilidade semântica através de vários projetos que complementavam o programa. RDFS foi um bom ponto de partida e já havia sido proposto como padrão pelo W3C. No entanto, ele não era expressivo o suficiente para adequar-se as necessidades do DAML. Uma nova linguagem chamada DAML-ONT foi então desenvolvida para que estendesse o RDF com construtores das linguagens de representação de conhecimento orientadas a objeto e baseadas em *Frame*. Como RDFS, DAML-ONT sofria de uma especificação semântica fraca e construída rapidamente. Isto levou a desacordos, no que dizia respeito a precisão do significado dos termos em uma ontologia DAML-ONT. Mais ou menos nesta mesma época, um grupo de pesquisadores (na maioria Europeus) com objetivos similares àqueles dos pesquisadores do DAML, haviam projetado uma outra linguagem para ontologias direcionada a Web, chamada OIL. Tal como DAML-ONT, OIL tinha uma sintaxe baseada em RDFS (e também uma alternativa sintaxe XML) e um conjunto de construtores de linguagem

baseados nas linguagens baseadas em *Frame*. Os desenvolvedores do OIL, entretanto, estabeleceram uma forte ênfase sobre o rigor formal. A linguagem foi explicitamente definida, tanto que sua semântica poderia ser especificada através de um mapeamento para uma descrição lógica bastante expressiva, SHIQ **[HOR 99, HOR 2001]**. Torna-se óbvio para ambos os grupos de pesquisa que seus objetivos poderiam ser melhor servidos se eles combinassem seus esforços, então o resultado seria a mesclagem entre DAML-ONT e OIL, produzindo assim, o DAML+OIL. A linguagem mesclada resultopu em uma semântica formal (modelo teórico), que fornecia entendimento automático e por seres humanos (e também uma axiomatização **[FIK 2001]**), e um acordo dos construtores das duas linguagens.

Até a pouco tempo atrás, o desenvolvimento de DAML+OIL era de responsabilidade do comitê *Joint EU/US Committee on Agent Markup Languages*, amplamente composto de membros das equipes de projeto das duas linguagens.

Mais recentemente, DAML+OIL está submetido ao W3C para formar a base da linguagem para ontologias da Web da W3C, na qual o *Web-Ontology Working Group* possui o mandato.

A última versão da linguagem DAML+OIL provê um rico conjunto de construções com o objetivo de criar ontologias e marcar informações de forma que seja entendida e lida automaticamente.

3.1.4.2 Motivação de uso

Quando se diz algo a uma pessoa, esta pessoa pode combinar o novo fato com um fato anterior e, então, dizer algo novo. Quando se diz algo a um computador em XML, ele é capaz de dizer algo novo em resposta, mas somente por causa de algum outro software que exista e que não é parte da especificação XML. Este software pode ser implementado diferentemente em sistemas que ainda obedecem a especificação XML. Pode-se obter diferentes respostas destes sistemas. Quando se diz algo novo ao computador em DAML+OIL, ele pode nos dar alguma nova informação, baseado inteiramente no padrão do próprio DAML+OIL. Um certo conjunto de conclusões são exigidas de qualquer sistema que obedeça ao DAML+OIL. Sistemas podem ser capazes de fornecer todos os tipos de serviços adicionais e respostas além dos requerimentos do padrão, mas um certo conjunto básico de conclusões serão sempre exigidos. DAML+OIL dá aos computadores um grau extra de autonomia que pode ajudá-los a fazer trabalhos mais úteis.

Um conjunto de declarações DAML+OIL por si só (e a especificação DAML+OIL) pode permitir a conclusão de uma outra declaração DAML+OIL, enquanto que um conjunto de declarações XML por si só (e a especificação XML) não permite a conclusão de qualquer outra declaração XML. Para empregar o XML para gerar novos dados, é necessário conhecimento embutido em algum código procedural em algum lugar, ao invés de explicitamente declarado, como em DAML+OIL.

Por exemplo, “Paternidade” é um relacionamento mais genérico que “Maternidade” e “Mary é a mãe de Bill” juntos, permitem um sistema de conformidade para DAML+OIL concluir que “Mary é um dos pais de Bill”. Desta forma, se um usuário propõe uma pesquisa em um sistema de buscas em DAML+OIL tal como “Quem são os pais de Bill?”, o sistema pode responder que “Mary é um dos pais de Bill”, mesmo que o fato não tenha sido declarado em qualquer lugar, mas pode somente ser derivado por uma aplicação DAML+OIL.

Declaração formal:

(motherOf subProperty parentOf)
(Mary motherOf Bill)

quando declarado em DAML+OIL, permite-se concluir:

(Mary parentOf Bill)

Baseado sobre a definição lógica de "subProperty" como determinado na especificação DAML+OIL. A mesma informação declarada em XML não permite afirmar o terceiro fato. XML por si só não provê nenhuma semântica em suas *tags*. Alguém pode criar um programa que determine semântica para uma *tag* "subProperty", mas posto que a semântica não é parte da especificação XML, as aplicações podem ser escritas em conformidade com a especificação XML, e mesmo assim não fazerem esta afirmação.

Outras linguagens tal com RDFS vão a um passo além do XML e podem suportar o exemplo dado, mas DAML+OIL oferece um conjunto de outras propriedades, tal como *equivalence* (*childOf*) ou que propriedades particulares são únicas (*uniqueProperty*).

3.1.4.3 DAML+OIL: uma linguagem para a Web Semântica

Um pré-requisito para difundir o uso de ontologias é a junção de padrões para sua descrição e troca **[BEC 2001]**. RDF Schema já é reconhecida como uma linguagem para ontologia e representação de conhecimento: ela trata de classes e propriedades (relações binárias), restrições de alcance (*range*) e domínio (*domain*) sobre propriedades, além de relações subclasses e subpropriedades. RDFS é uma linguagem relativamente primitiva, e mais poder de expressividade deveria ser necessário e desejável para a descrição de recursos em suficiente detalhamento. Além disso, estas descrições deveriam ser disciplinadas para raciocínio automático se elas forem utilizadas efetivamente para processos automáticos.

Como dito anteriormente, estas considerações levam ao desenvolvimento do OIL e, em seguida, DAML+OIL, que são linguagens para descrição de ontologias que estendem RDFS com um conjunto mais rico de primitivas de modelagem.

OIL é uma linguagem proposta como uma linguagem de representação de conhecimento para a Web e aplicações para a Web. OIL junta primitivas de modelagem frequentemente utilizadas em ontologias baseadas em *Frame* com uma semântica simples, clara e bem definida de uma Description Logic (DL). Uma DL facilita as condições de serviços de raciocínio automático, numa consistência particular e checagem de classificação.

Adicionalmente, existe o empenho da maximização de compatibilidade com os padrões da Web existentes e que estão surgindo. Estes padrões, tal como RDFS, tornam mais fáceis o uso consistente de ontologias (significados consistentes para os elementos da ontologia) sobre a Web.

De modo similar, o programa DAML **[DAM 2001]** investe na definição de uma linguagem para agentes na Web. DAML+OIL é a combinação destas duas iniciativas na qual possui as seguintes características:

- um *mapeamento base* para uma expressiva descrição lógica (SHIQ) **[HOR 99]**, fornecendo um semântica bem definida e um claro entendimento das propriedades formais da linguagem. A DL dá a DAML+OIL a habilidade e flexibilidade para compor classes e *slots* para formar novas expressões usando conectivas booleanas, aninhamento ilimitado de elementos de classes, *slots* inversos e transitivos, axiomas

gerais, etc. Por exemplo, é capaz de se definir condições suficientes para conceitos, assim como condições necessárias. Além disso, é capaz de utilizar suporte a raciocínio automático para se computar automaticamente relações de classificação entre os conceitos, ou checar a consistência e coerência da classificação e seus conceitos. Isto é muito útil quando as ontologias são reusadas ou fundidas. O mapeamento também fornece um mecanismo para provisão de serviços práticos de raciocínio através da utilização de sistemas DL, como por exemplo o sistema FaCT **[HOR 2000]**. Isto significa que uma ontologia expressada em DAML+OIL pode ser equilibrada pelo raciocionador FaCT.

- Uma *codificação sintática legível automaticamente (machine-readable) nas linguagens da Web*. RDFS é um mecanismo proposto para desenvolvimento de metadados. DAML+OIL é definido como uma extensão para RDFS, e, portanto, DAML+OIL é acessível por qualquer aplicação RDFS. Uma ontologia em DAML+OIL pode ser usada por um agente que não seja conhecedor de DAML+OIL, mas conhecedor de RDFS. Assim, DAML+OIL é um dispositivo de gerenciamento potencial para a Web Semântica.
- Uma arquitetura estendida em camadas, evitando a tentação de jogar tudo dentro do núcleo da linguagem, misturando características que não podem ser equilibradas (reasoned) com aquelas que podem. Assim, os limites são claros e explícitos.

DAML+OIL forma uma parte chave do trabalho da *W3C's Semantic Web Activity*.

DAML+OIL permite a definição e descrição de classes (conceitos), *slots*(relacionamentos), instâncias e axiomas dentro de uma ontologia.

3.1.4.3.1 Definições de Classes

Em DAML+OIL, as definições de classes são providas através do uso de afirmações (declarações) e axiomas. Afirmações como dizer que uma classe “gato” é uma subclasse de “animal”. Alternativamente, axiomas podem afirmar igualdade entre classes, dizendo que “animal” é equivalente a “bicho”. Um aspecto chave de DAML+OIL é que estes axiomas e afirmações precisam preocupar-se com conceitos atômicos (gato e animal), mas também com operadores de conceitos tal como os operadores booleanos **AND, OR** e **NOT**. Adicionalmente, podem ser usadas expressões de restrições (*constraints*) que representam o papel de restringir ou quantificar. Isto vem em contraste aos convencionais sistemas de representação, onde, em geral, restrições de *slots* e superclasses devem ser nomes de classes.

Lista de Classes:

- A seguir, as classes DAML (*daml:Classes*) que DAML+OIL define:
  - *Thing*
  - *Nothing*
- A seguir, as classes RDFS (*rdfs:Class*) que DAML+OIL define:
  - *Class*
  - *Datatype*
  - *Restriction*: algo está na classe R se ele satisfaz as restrições declaradas, e vice-versa.
  - *ObjectProperty*: se P é um *ObjectProperty*, e P(x, y), então y é um objeto.
  - *DatatypeProperty*: se P é um *DatatypeProperty*, e P(x, y), então y é um valor de dado.

- *TransitiveProperty*: se P é um *TransitiveProperty*, então se P(x, y) e P(y, z) então P(x, z). Cf. OIL *TransitiveProperty*.
- *UniqueProperty*: comparado a *maxCardinality*=1.
- *UnambiguousProperty*: se P é um *UnambiguousProperty*, então se P(x, y) e P(z, y) então x=z. Conhecido como injective. Ex. se firstBorne(m, Susan) e firstBorne(n, Susan) então m e n são o mesmo.
- *List*
- *Ontology*: Uma Ontologia é um documento que descreve um vocabulário de termos para comunicação entre agentes automatizados (e humanos).
- *Literal*: mesmo que *rdfs*:*Literal*
- *Property*: mesmo que *rdfs*:*Property*

3.1.4.3.2 Definições de Propriedades

As definições de propriedades dão nomes as propriedades e permitem propriedades adicionais do *slot* a ser declarado, por exemplo, *slot* inverso, ou se o *slot* tem propriedades transitivas, simétricas ou funcionais. Os *slots* também formam hierarquias, tanto que se pode especificar os nomes de qualquer superslot.

As restrições de domínio (domain) e alcance (range) podem ser especificadas sobre um *slot* e, assim como em descrições de classes, estas restrições podem ser expressões de classes arbitrárias tal como expressões de classes anônimas ou combinações booleanas de nomes de classes e expressões de classes, novamente estendendo a expressividade das linguagens tradicionais. Todas as declarações feitas sobre *slots* são usadas pelo raciocinador (reasoner) e podem induzir relacionamentos hierárquicos entre classes.

Lista de propriedades:

- A seguir, as propriedades DAML (*daml:Property*) que DAML+OIL define:
  - *equivalentTo*: para *equivalentTo***(X, Y)**, **X** é um termo equivalente a **Y**.
  - *sameClassAs*: para *sameClassAs***(X, Y)**, **X** é uma classe equivalente **Y**. *samePropertyAs*: para *samePropertyAs***(P, R)**, **P** é uma propriedade equivalente a **R**. Cf. OIL Equivalente.
  - *sameIndividualAs*: para *sameIndividualAs***(a, b)**, **a** é como **b**.
  - *differentIndividualFrom*: para *differentIndividualFrom***(a, b)**, **a** não é como **b**.
- A seguir, as propriedades RDF (*rdf:Property*) que DAML+OIL define:
  - *disjointWith*: para *disjointWith***(X, Y)**, **X** e **Y** não tem membros em comum.
  - *unionOf*: para *unionOf***(X, Y)**, **X** é a união das classes na lista **Y**; ex. se algo está em qualquer das classes em Y, também está em X, e vice-versa. Cf. OIL *Or*.
  - *disjointUnionOf*: para *disjointUnionOf***(X, Y)**, **X** é a união disjuntiva das classes na lista Y: (a) para qualquer **c1** e **c2** em **Y**, *disjointWith*(c1, c2), e (b) *unionOf*(X, Y). Cf. OIL *disjoint-covered*.
  - *intersectionOf*: para *intersectionOf***(X, Y)**, **X** é a interseção das classes na lista **Y**; Ex. se algo está em todas as classes em **Y**, então ele está em **X,** e vice-versa. Cf. OIL And.
  - *complementOf*: para *complementOf*(X, Y), X é o complemento de Y; se algo está em Y, então ele não está em X, e vice-versa. Cf. OIL *Not*.

- *oneOf*: para *oneOf*(C, L), tudo em C é uma das coisas em L; Isto permite-nos definir classes com enumeração dos membros. Cf. OIL *OneOf*
- *onProperty*: para *onProperty*(R, P), R está limitado a propriedade P.
- *toClass*: para *onProperty*(R, P) e *toClass*(R, X), i esta na classe R se e somente se para todo j, P(i, j) implicar *type*(j, X). Cf. OIL *ValueType*
- *hasValue*: para *onProperty*(R, P) e *hasValue*(R, V), i está na classe R se e somente se P(i, V). Cf. OIL *HasFiller*.
- *hasClass*: para *onProperty*(R, P) e *hasClass*(R, X), i está na classe R se e somente se para algum j, P(i, j) e *type*(j, X). Cf. OIL *HasValue*.
- *minCardinality*: para *onProperty*(R, P) e *minCardinality*(R, n), i esta na classe R se e somente se existem pelo menos n distintos de j com P(i, j). Cf. OIL *MinCardinality*
- *maxCardinality*: para *onProperty*(R, P) e *maxCardinality*(R, n), i esta na classe R se e somente se existem no máximo n distintos de j com P(i, j). Cf. OIL *MaxCardinality*
- *cardinality*: para *onProperty*(R, P) e *cardinality*(R, n), i esta na classe R se e somente se existem exatamente n distintos de j com P(i, j). Cf. OIL *Cardinality*
- *hasClassQ*: propriedade para especificação de restrição de classe com restrição *cardinalityQ* .
- *minCardinalityQ*: para *onProperty*(R, P), *minCardinalityQ*(R, n) e *hasClassQ*(R, X), i esta na classe R se e somente se existem pelo menos n distintos de j com P(i, j) e type(j, X). Cf. OIL *MinCardinality*
- *maxCardinalityQ*: para *onProperty*(R, P), *maxCardinalityQ*(R, n) e *hasClassQ*(R, X), i esta na classe R se e somente se existem no máximo n distintos de j com P(i, j) e *type*(j, X). Cf. OIL *MaxCardinality*
- cardinalityQ: para *onProperty*(R, P), *cardinalityQ*(R, n) e *hasClassQ*(R, X), i esta na classe R se e somente se existem exatamente n distintos de j com P(i, j) e type(j, X). Cf. OIL *Cardinality*
- *inverseOf*: para *inverseOf*(R, S), R é o inverso de S; Ex. se R(x, y) então S(y, x) e vice-versa. Cf. OIL *inverseRelationOf*
- *first*: car
- *rest*: cdr
- *item*: para item(L, I), I é um item em L; ou *first*(L, I) ou *item*(R, I) onde *rest*(L, R).
- *versionInfo*
- *imporsts*: para *imports*(X, Y), X importa Y.
- *subPropertyOf*: *rdfs*:*subPropertyOf*
- *type*: *rdfs:type*
- *value: rdfs:value*
- *subClassOf: rdfs:subClassOf*
- *domain: rdfs:domain*
- *range: rdfs:range*
- *label: rdfs:label*
- *comment: rdfs:comment*
- *seeAlso: rdfs:seeAlso*
- *isDefinedBy: rdfs:isDefinedBy*

3.1.4.3.3 Raciocínio e DAML+OIL

Os serviços de raciocínio oferecidos pelas descrições lógicas (DL) suportam o desenvolvimento e a manutenção incremental de uma ontologia. Implementações altamente otimizadas de verificação e completos algoritmos de classificação de quadros para muitas descrições lógicas expressivas tal como SHIQ pode ser usadas em contrário ao pior caso de complexidade. Assim, uma ontologia expressada utilizando-se DAML+OIL pode ser verificada usando-se o raciocinador FaCT **[HOR 98]**. A chave dos serviços de raciocínio são:

- Checagem de classificação entre duas descrições de conceitos, C e D; C inclui D quando o conjunto de objetos que são instâncias de D são sempre um subconjunto de objetos que são instâncias de C.
- A classificação organiza uma coleção de expressões de conceitos dentro de uma ordem parcial baseada na checagem de classificação. Isto provê uma rede simétrica de definições, dispondo-se do geral para o específico. As definições compostas têm sua posição implicitamente determinada automaticamente. Assim, a classificação é um processo dinâmico que pode ser adicionado a uma hierarquia existente.
- A satisfação (satisfability) dos conceitos checa se uma descrição de conceito nunca pode ter instâncias por causa das inconsistências ou contradições do modelo.

Quando classifica-se uma ontologia, um número de novos relacionamentos de classificação pode ser descoberto (devido as definições da classe no modelo). Isto pode ser útil durante a fase de construção da ontologia.

DAML+OIL é equivalente a uma descrição lógica muito expressiva. Mais precisamente, DAML+OIL é equivalente a SHIQ **[HOR 99]** com o acréscimo de classes existencialmente definidas (ex. o construtor *oneOf*) e tipos de dados (frequentemente chamados de domínios completos em DLs **[BAA 91]**). Esta equivalência permite DAML+OIL explorar uma parte considerável das pesquisas em descrições lógicas, Ex:

- Para definir a semântica da linguagem e o entendimento formal das suas propriedades, particularmente a decidabilidade e complexidade dos problemas chaves de inferência **[DON 97]**;
- Como fonte de algoritmos sólidos e completos e técnicas de implementação otimizada para determinantes problemas chaves de inferência **[HOR 99]**;
- Para usar sistemas de descrições lógicas implementados para fornecer suporte (parcial) a raciocínio **[HOR 98, HAA 2001]**.

Uma consideração importante no objetivo de DAML+OIL foi que os problemas chaves de inferência na linguagem, particularmente consistência e classificação de classes, deveriam ter capacidade de decisão, a medida que isto facilitaria o fornecimento dos serviços de raciocínio. Além disso, a correspondência com as DLs facilitaria o uso de algoritmos DL que sabidamente são receptivos a implementação otimizada e comportam-se bem em aplicações realísticas apesar da sua complexidade **[HOR 98, HAA 2001a]**. Particularmente, DAML+OIL é capaz de explorar muitos serviços otimizados de raciocínio providos por sistemas DL tal como FaCT **[HOR 98]** e RACER **[HAA 2001]**, embora estes sistemas, por enquanto, não suportam totalmente a linguagem DAML+OIL (nenhum deles é capaz de entender classes definidas

existencialmente, ex. a construção *oneOf*, ou fornecem suporte para todos tipos de dados do XML Schema).

A manutenção da decidabilidade da linguagem requer certas restrições sobre seu poder de expressividade, que podem não ser aceitas por todas as aplicações. Entretanto, os projetistas da linguagem decidiram que raciocínio seria importante se o completo poder das ontologias fosse realizado, e que uma poderosa linguagem para ontologia, e com capacidade de decisão, seria um bom começo.

Raciocínio pode ser útil em muitos estágios durante o projeto, manutenção e desenvolvimento das ontologias.

- "Raciocínio" pode ser usado para dar suporte ao projeto da ontologia e para melhorar a qualidade dos resultados da ontologia. Por exemplo, raciocínio da consistência e classificação das classes pode ser usado para verificar classes logicamente inconsistentes e relacionamentos de classificação implícitas. Este tipo de suporte tem sido muito importante em grandes ontologias, nas quais são frequentemente construídas e mantidas por um longo período e por muitos autores.

- Como integração de informação **[CAL 98]**, a integração de ontologias pode também ser suportada pelo "raciocínio". Por exemplo, a integração pode ser executada usando afirmações inter-ontologias especificando relacionamentos entre classes e propriedades, com raciocínio sendo usado para computar a hierarquia integrada e destacando-se problemas ou inconsistências.

- "Raciocínio", no que diz respeito ao desenvolvimento de ontologias aumentará o poder dos "agentes inteligentes", permitindo-os determinar se um conjunto de fatos esta consistente em relação a ontologia, identificando objetos que são membros implícitos de uma dada classe, etc. Um adequado serviço de ontologia deve, por exemplo, permitir um agente procurar serviços seguros para identificar um serviço requerendo um usuário e senha como um possível candidato.

3.1.5 Linguagem para Modelos: Quadro Comparativo

Abaixo, uma tabela de comparação **[RAT 2001]** criada para entender as trocas e diferenças entre as linguagens para definição de modelos. São comparadas XML Schema , RDFS e DAML+OIL, mostrando uma descrição de como cada linguagem trata requerimentos comuns de representação de conhecimento.

Este comparativo serve como embasamento para se obeter uma visão geral dos melhores caminhos e métodos que permitam atingir o objetivo de se elaborar um modelo ontológico para o Currículo Lattes.

TABELA 3.1 - Quadro Comparativo - XML Schema, RDFS e DAML+OIL

Dimensão	**XML (Schema)**	**RDF (Schema)**	**DAML+OIL**	**Notas**
Contexto	Default Namespace: xmlns Declarando outros: xmlns:<label> XMLSchema Namespace (rotulado de xsd) xmlns:xsd="www.w3.org/2001/XMLSchema" targetNamespace refere-se ao namespace definido pelo arquivo atual. Nota: Namespaces não necessitam apontar para qualquer coisa na especificação do XML Namespaces. <xsi:schemaLocation..> fornece a localização do schema.	RDF usa XML Namespaces. RDF Syntax Namespace xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" RDF Schema Namespace xmlns:rdfs ="http://www.w3.org/2000/01/rdf-schema#" Nota: Em RDF, a URI do namespace identifica a localização do RDF Schema.	DAML também usa XML Namespaces. Ele usa elementos RDF & RDF Schema referenciando-se ao seus respectivos Namespaces. O último DAML Namespace é xmlns:daml="http://www.daml.org/2001/03/daml+oil#" . Em DAML, deve-se Importar as ontologias para ter a capacidade de usar as classes definidas na ontologia.	
Classes de Objetos e Propriedades	Sem conceito de classes e propriedades, somente Elementos de certos tipos. O tipo pode ser simpleType ou um complexType. (Uma classe pode ser qualquer elemento e sua propriedade pode ser seu elemento filho - SEM SEMÂNTICA DEFINIDA).	Resource é a classe de nível mais alto. (http://www.w3.org/2001/01/rdf-schema#Resource) Ciclos na hierarquia de classes não eram permitidos, mas as últimas revisões da especificação do RDF já permitiam.	DAML também possui Classes e Propriedades. Classes podem ser subclasses de uma classe anônima criado devido a uma Restrição (Restriction) sobre o conjunto de todas as 'Coisas' (Thing). Dois tipos de propriedades são definidos: ObjectProperties (Relaciona um objeto a um outro objeto - o valor da propriedade é também um objeto) e DatatypeProperties (Relaciona um objeto a um tipo de dado primitivo - o valor da propriedade é um tipo de dado primitivo) Ciclos na hierarquia de classes são permitidos.	

Dimensão	XML (Schema)	RDF (Schema)	DAML+OIL	Notas
Herança	Visto que não existem qualquer Classe, não há o conceito de herança. Entretanto, Tipos podem ser extended ou restricted , definindo-se assim, subtipos (subTypes).	Uma classe pode ser um subclasse de outras classes (subClassOf) - herança múltipla é permitida. Propriedades podem também ser subpropriedades de outras propriedades (subPropertyOf).	Igual ao RDF.	
Propriedade / Alcance do Elemento (Range)	Pode ser especificado globalmente. Para um range especificado localmente (elemento específico), o elemento deve ser declarado localmente.	Só pode ser especificado globalmente: <rdfs:range....> Declarações múltiplas implicam conjunção (todas devem ser satisfeitas).	Podem ser especificadas globalmente (globally) (<rdfs:range...>) quanto localmente (locally) <daml:Restriction> <daml:onProperty...> <daml:toClass....> </daml:Restriction> Declarações múltiplas implicam conjunção (todas devem ser satisfeitas).	Range especifica o tipo de valor (elementos / classes / datatypes) que as propriedades podem ter.
Propriedade / Domínio do Elemento (Domain)	Sem declaração explícita do domínio do elemento. O domínio é, implicitamente, o elemento na qual a definição aparece.	Pode ser especificado só globalmente <rdfs:domain....> Declarações múltiplas de domínio implicam conjunção (todas devem ser satisfeitas).	Pode ser especificado globalmente (<rdfs:domain...>). Declarações múltiplas de domínio implicam conjunção (todas devem ser satisfeitas).	Domain especifica em quais classes podem haver uma propriedade particular.
Propriedade / Cardinalidade do Elemento (Cardinality)	Pode ser especificado usando-se minOccurs e maxOccurs.	Não definido em RDF Schema. Por definição, não existem restrições de cardinalidade sobre propriedades. Entretanto, novas restrições como estas podem ser especificadas fazendo com que elas sejam subclasses da classe 'ConstraintProperty'.	Podem ser especificadas localmente minCardinality, maxCardinality, cardinality Também podem ser especificadas globalmente, embora somente como uma UniqueProperty (com valor simples de 1).	A cardinalidade (cardinality) de uma propriedade especifica o número de ocorrências da propriedade para uma certa classe.

Dimensão	XML (Schema)	RDF (Schema)	DAML+OIL	Notas
Tipos de dados básicos	Os Datatypes suportados pelo XMLSchema são, principalmente, variações de tipos de dados numéricos, temporais e strings.	O cerne do RDF Schema somente inclui 'Literais' na qual é o conjunto de todas strings.(deve-se ser incluídos os tipos de dados do XMLSchema)	Permite o uso dos tipos de dados do XMLSchema pelo simples referência a URI do XMLSchema.	
Enumeração dos valores de propriedades	Possível com a tag <enumeration>.	Não é possível.	Enumeração de tipos de propriedades é possível através da tag <oneOf rdf:ParseType="daml: collection"...> Também é possível por simplesmente apontar para um declarado tipo de dado enumerado usando o XMLSchema	Uma Enumeração restringe o valor de uma propriedade para um certo conjunto de valores.
Conjunto de dados ordenados	Data Sets mantêm ordem por definição Nota: Tipos de declarações possuem a tag <sequence>, só significa que os dados devem aparecer na ordem especificada.	Data Set ordenado com a tag <rdf:Seq...>.	Pode usar a tag <daml:list> .	
Listas restritas (Bounded Lists)	Não é possível especificar.	Não é possível especificar.	Possível com a tag <daml:collection>.	
Propriedades transitivas	Não é possível especificar.	Não podem ser declaradas.	Possível com a tag <daml:TransitiveProp erty>.	
Negação	Não é possível.	Não presente.	Possível com a tag <daml:complementOf. .>.	Negação implica a ausência de algum elemento (nenhum Carro é uma pessoa)
Classes disjuntivas (Disjoint)	Uma união de possíveis tipos para um elemento é possível com a tag <union>.	Pode usar um Bag para indicar coleções desordenadas (ou uniões de propriedades). Entretanto, não se pode ter uma classe como uma união de 2 classes.	Uma classe pode ser uma união de 2 outras classes. Possíveis com a tag <unionOf...>. Pode-se, ainda, representar uniões disjoint com a tag <disjointUnionOf...>.	

Dimensão	XML (Schema)	RDF (Schema)	DAML+OIL	Notas
Condições necessárias e suficientes para membros de uma classe	Não (embora <unique> possa ser interpretado como uma UnambigousProperty)	Não	Sim sameClassAs, equivalent, usando combinações booleanas de expressões de classes. UnambigousProperty especifica uma propriedade na qual identifica o recurso (como uma chave primária).	Condições necessárias e suficientes para membros de uma classe (Class Membership) especifica uma definição de classe que pode ser usada: 1) para determinar (ou identificar) se uma instância pertence àquela classe. 2) para determinar (ou classificar) se a classe é uma subclasse de uma outra classe.

3.2 Linguagens de consultas

3.2.1 TRIPLE

TRIPLE **[SIN 2002]** é uma linguagem de consulta, inferência e transformação para modelos RDF para a Web Semântica.

TRIPLE é baseada sobre a lógica Horn[2] **[AAB 99]**, na qual aborda muitas características básicas do F-Logic, mas é especialmente projetada para pesquisa e transformação de modelos RDF.

TRIPLE pode ser visto como um sucessor de SiLRI (Simple Logic-based RDF Interpreter **[DEC 98]**). Uma das mais importantes diferenças para F-Locic e SiLRI é que TRIPLE não tem uma semântica fixa para as características de orientação a objeto tal como classes e herança, ou seja, ao invés de ter uma semântica embutida para o RDF Schema (como outras linguagens de consulta tem), TRIPLE permite a semântica da linguagem no topo do RDF (tal como RDF Schema, etc) para serem definidas com regras. Sua arquitetura permite tais características serem facilmente definidas para diferentes objetos orientados e para extensões RDF (Schema) tal como OIL e DAML+OIL, que não podem se manipuladas diretamente pela lógica Horn. Além disso, são fornecidos acessos a programas externos, tal como classificadores de descrições lógicas como o FaCT **[HOR 98]** ou RACER **[HAA 2001]**, resultando em uma linguagem de regras híbridas.

[2] Cláusulas Horn são utilizadas na programação em Prolog.

Como resultado, TRIPLE permite raciocínio e transformação RDF sujeita a semânticas muito diferentes, que são necessárias no caso de acesso a muitos fontes de dados em uma aplicação (Ex. integração de dados).

TRIPLE é uma linguagem de regras em camadas. Dois diferentes tipos de camadas são suportadas:

- Extensões sintáticas da lógica Horn para suportar construções básicas do RDF, tal como recursos e declarações.
- Módulos para extensões semânticas de RDF, tal como RDF Schema, OIL e DAML+OIL, implementadas diretamente em TRIPLE ou através de interação com componentes externos de raciocínio.

DAML+OIL possui extensões de descrições lógicas do RDF Schema que não podem ser mapeadas diretamente pela lógica Horn. Por esta razão, um modelo daml_oil(Mdl) é fornecido através de acessos a um classificador de descrições lógicas (por exemplo, o FaCT) para que se realize a semântica do DAML+OIL.

3.2.2 Linguagens de Consulta para DAML+OIL

Na seção 3.1.4, foi visto que DAML+OIL é uma linguagem que acrescenta detalhes no topo do RDF/RDFS a fim de expressar mais classificações e propriedades de recursos do que RDFS. DAML+OIL fornece primitivas encontradas nas linguagens baseadas em *Frame*, enquanto que sua semântica formal é definida em descrições lógicas. Entretanto, uma linguagem de consulta para os metadados expressados em DAML+OIL é ainda um trabalho em andamento. Por exemplo, DAML-S (DAML Search Engine) **[DEN 2001]** é um motor que habilita pesquisa em uma ontologia DAML. A forma da linguagem de consulta usada é “FIND ... SUCH-THAT ...END”, que permite encontrar recursos que satisfaçam uma conjunção de declarações (triplas). Pode-se encontrar, também, uma discussão sobre a linguagem DQL (*DAML+OIL Query Language*) nos arquivos da lista de email do *joint-committee@daml.org*. Embora DQL ainda esteja em desenvolvimento, ele é definido em duas partes, um *Query Premise* e uma *Query Pattern*. A *Query Premise* é a condição básica de checagem sobre a base de conhecimento consultada. Uma *query premise* tem importância de permitir uma *query* “hipotetizar” um objeto (por exemplo, “Se Foo é uma pessoa com dois irmãos homens ...”) e então fazer questões sobre o objeto “hipotetizado”. A *Query Pattern* corresponde a cláusula *from*.

Arnold de Vos propôs em **[VOS 2002]**, uma linguagem de consulta baseada em RDF sobre DAML+OIL, que permite consultar tanto em (meta)dados RDFS quanto em DAML+OIL. Uma *query*, nesta linguagem, é formulada com uma expressão da forma “select ... from ...”, enquanto os resultados das consultas são somente as triplas. No entanto, a cláusula *from*, que é uma expressão que descreve uma classe DAML (onde as declarações são encontradas), pode ser usada para expressar propriedades complexas dos conceitos DAML+OIL.

Finalmente, uma linguagem de consulta para DAML+OIL proposta pela Universidade de Manchester **[HOR 2002]** também permite explorar sistemas com descrições lógicas (*DL Systems*) para fornecer serviços completos de raciocínio. Esta linguagem também conta com um modelo de dados em triplas. Enquanto se introduz tipos de dados concretos, uma *query*, nesta linguagem, representando uma conjunção booleana de declarações, pode retornar respostas verdadeiras ou falsas, ou um conjunto de tuplas arbitrárias.

3.2.3 JENA e RDQL

Jena **[JEN 2002]** é um ambiente para o desenvolvimento de aplicações dentro da Web Semântica.

Jena é uma Interface para Programação de Aplicativos (API) Java e um *toolkit* para manipulação de modelos RDF. Jena integra o processador RDFFilter **[MEG 2000]** de David Megginson e o *parser* RDF/XML baseado em Java, chamado Another RDF Parser (ARP) **[CAR 2001]**, que se adapta as últimas recomendações do RDF Core Working Group da W3C.

O conjunto de programas e rotinas do Jena fornece suporte para os modelos RDF e suas características incluem:

- Métodos de declarações centrais para manipulação de um modelo RDF como um conjunto de triplas RDF
- Métodos de recursos centrais para manipulação de um modelo RDF como um conjunto de recursos com propriedades
- Chamadas a métodos progressivos para programação mais conveniente
- Suporte embutido para *containers* RDF – *bag, alt e seq*
- Recurso melhorados – a aplicação pode estender o comportamentos dos recursos
- *Parsers* integrados (ARP (Another RDF Parser e RDFFilter de David Megginson)
- Integrada linguagem de consulta (RDQL)
- Suporte para armazenamento de ontologias em DAML+OIL em um modelo Jena
- Módulo de armazenamento persistente baseado sobre o banco de dados Berkeley
- Arquitetura aberta para suporte a outras implementações de armazenamento

3.2.3.1 RDQL

RDQL (*RDF Data Query Language*) é uma linguagem de consulta para RDF em modelos JENA. A idéia é prover um modelo de pesquisa orientado a dados de modo que exista uma abordagem mais declarativa para complementar a API do Jena. Esta linguagem é "orientado a dados" pois ela somente pesquisa as informações guardadas nos modelos; não existe inferência. No modelo Jena dá-se a impressão de que certas triplas existem pela criação das mesmas sobre demanda. Entretanto, o sistema RDQL simplesmente pega a descrição daquilo que a aplicação quer, na forma de uma *query*, e retorna aquela informação na forma de um conjunto de ligações.

RDQL é uma implementação da linguagem de consulta em RDF, chamada SquishQL **[MIL 2002]**, na qual é derivada do rdfDB **[GUH 2000]**. Esta classe de linguagens de consulta considera RDF como uma tripla de dados, sem esquema ou informações ontológicas; a menos que explicitamente incluídas no fonte RDF.

RDF fornece um grafo com arcos diretos – os nodos são recursos ou literais. RDQL fornece um caminho de especificação de um padrão gráfico que é combinado contra outro para render um conjunto de outras combinações. Ele retorna uma lista de ligações – cada ligação é um conjunto de pares nome-valor para os valores das variáveis. Todas as variáveis são limitadas (não existe disjunção na *query*).

3.2.3.1.1 Linha de Comando

O Jena vem com um programa de linha de comando para a execução das pesquisas em RDQL.

java -cp ... jena.rdfquery ...

Este programa executará uma consulta sobre os fontes de dados especificados na cláusula FROM da consulta, ou sobre a linha de comando. Ele pode pesquisar todas as formas do modelo Jena: XML, N-Triple, BerkeleyDB ou um banco de dados relacional.

Este programa tem um formatador embutido para o dados resultantes. Ele pode imprimir em texto como colunas alinhadas e em HTML, e também em formatos puros, mais adequados para favorecer o processamento.

Para tanto, ele requer alguns argumentos:

Uso: [--xml|--ntriple] [--data URL] [queryString | --query file]

TABELA 3.2 - Argumentos para linha de comando do RDQL

--query file	Arquivo de entrada da query
--xml	Fonte de dados é XML (default)
--ntriple	Fonte de dados é n-triple
--data URL	Fonte de dados (pode também ser parte da consulta)
--time	Imprime alguma informação de tempo
--test [file]	Executa um teste
--format FMT	Formato (texto, html, tuplas, dump ou nenhum)
--verbose	Mais mensagens
--quiet	Menos mensagens

3.2.3.1.2 Sintaxe RDQL

RDQL tem uma sintaxe como SQL para os modelos de *query* derivados do SquishQL e rdfDB.

Em SQL, um banco de dados é um mundo fechado; a cláusula FROM identifica as tabelas no banco de dados; a cláusula WHERE identifica as restrições e pode ser estendida com AND. Por analogia, a Web é o banco de dados e a cláusula FROM identifica os modelos RDF. As variáveis são introduzidas com uma instrução “?” e as URIs são mencionadas entre os sinais de menor e maior (< >); URIs sem sinais de delimitação podem ser usadas onde não há ambiguidade.

- Cláusula SELECT

Identifica quais as variáveis que serão retornadas pela consulta. Se for usado SELECT ?x, ?y, ?z, então retornará uma matriz de tuplas contendo valores para ?x, ?y e ?z. Podem ser usadas outras variáveis na consulta, tais como, ?a1, ?p, ?nome, entre outros. Entretanto, serão retornados valores para estas variáveis desde que elas sejam explicitadas na consulta. Se nem todas as variáveis são necessárias na consulta, então especifica-se apenas os resultados que são desejados. Assim, reduz-se a quantidade de memória necessária para os resultados, assim como o fornecimento de informações para o otimizador da consulta.

- Cláusula FROM

A cláusula FROM da consulta indica as fontes de dados RDF ou DAML+OIL a serem pesquisados. Cada fonte é especificada entre sinais de maior e menor (<&>) e indica a fonte pela URLs ou *paths* de documentos locais. Por exemplo:

```
FROM <doc.rdf>, <http://example.com/sample.rdf>, <rdfs/other.rdf>
```

Se forem indicados mais de uma fonte, estes devem ser separados por vírgulas.

- Cláusula WHERE

A cláusula WHERE é a mais importante parte da expressão RDQL. Nesta cláusula são indicadas as restrições que as triplas RDF (sujeito, predicado, objeto) devem satisfazer para os resultados serem retornados. A cláusula WHERE é expressada por uma lista de restrições separadas por vírgulas. Cada restrição adota a forma **(sujeito, predicado, objeto);** onde o sujeito, predicado e objeto podem ser um valor literal ou uma variável RDQL.

Para o predicado, pode-se expressar nomes de propriedades usando-se um *namespace* declarado na cláusula USING. Por exemplo, <dc:name>, onde indica que o predicado deve ser igual ao nome local do predicado "name" para o namespace declarado como "dc" na cláusula USING.

Exemplo:

```
(?x,<foo:has>,?y), (?y,<foo:color>,?z)
```

Isto retornará todas as declarações RDF onde algum sujeito "x" tem uma propriedade "has" apontando para um recurso e que este recurso tem uma propriedade "color". Para filtrar qual a cor desejada, deve-se usar a cláusula AND da consulta.

- Cláusula AND

A cláusula AND indica as restrições que as variáveis RDQL devem seguir. Para tanto, são especificadas expressões booleanas.

Exemplo: Selecione todos os sujeitos que tem um objeto azul.

```
SELECT ?x
FROM <doc.rdf>
WHERE (?x,<foo:has>,?y), (?y,<foo:color>,?z)
AND ?z=="blue"
USING foo for
```

- Cláusula USING

Esta cláusula serve para declarar todos os namespaces que serão usados para as propriedades RDF. Estas declarações devem ser separadas por vírgulas e usam a notação:

```
prefix for
```

Um meio para encurtar o tamanho das URIs. Como SquishQL é provavelmente escrito por pessoas, este mecanismo ajuda a contribuir para um melhor entendimento da sintaxe. Este não é um mecanismo de *namespace*; ao invés disso, ele é simplesmente um mecanismo de abreviação para URIs longos, através da definição de um prefixo.

Exemplo:

```
USING foo for <http://foo.org/properties#>,
      col for <http://props.com/catalog#>
```

3.3 Ferramentas

3.3.1 OntoEdit

Tipicamente, uma ontologia é construída e mantida por um esforço colaborativo dos domínios especialistas, usuários finais e especialistas em tecnologia da informação.

OntoEdit **[MAD 2000]** é um editor de ontologias desenvolvido pelo Knowledge Management Group da Universidade de Karlsruhe.

OntoEdit é uma ferramenta que habilita inspeção, navegação, codificação e modificação de ontologias; e suporta, desta forma, as tarefas de desenvolvimento e manutenção das ontologias.

Modelar ontologias utilizando OntoEdit significa modelar no nível conceitual enquanto for possível a independência de uma linguagem de representação concreta. Usando-se GUI's para visualizar representações de estruturas conceituais (conceitos, hierarquia de conceitos, relações e axiomas) ao invés de codificar estruturas conceituais em ASCII. O modelo conceitual de uma ontologia é armazenado internamente utilizando-se um modelo ontológico que pode ser mapeado sobre diferentes e concretas linguagens de representação.

OntoEdit é um ambiente de desenvolvimento para projetar, adaptar e importar modelos de conhecimento para sistemas de aplicações. Além disso, ele suporta desenvolvimento de ontologias em multi-idiomas e múltipla herança.

Dentre as funcionalidades da versão atual do OntoEdit, constam a definição de conceitos de uma forma hierárquica, a herança múltipla, a definição de relações locais e globais, definição de instâncias, desenvolvimento em multi-idiomas, a definição de axiomas básicos, tais como conceitos de disjunção (disjoint), relações simétricas e relações transitivas. Além disso, na última versão do OntoEdit, pode-se importar e exportar RDF, importar e exportar DAML+OIL e importar e exportar F-Logic.

O OntoEdit oferece ainda, a flexibilidade de uma interface para *plugin*. Esta interface oferece ao usuário a possibilidade do mesmo desenvolver seus próprios *plugins*, que automaticamente serão implementados no OntoEdit. OntoEdit também é capaz de carregar ou salvar as ontologias por um tipo de dados definido num formato de armazenamento baseado no XML, chamado OXML.

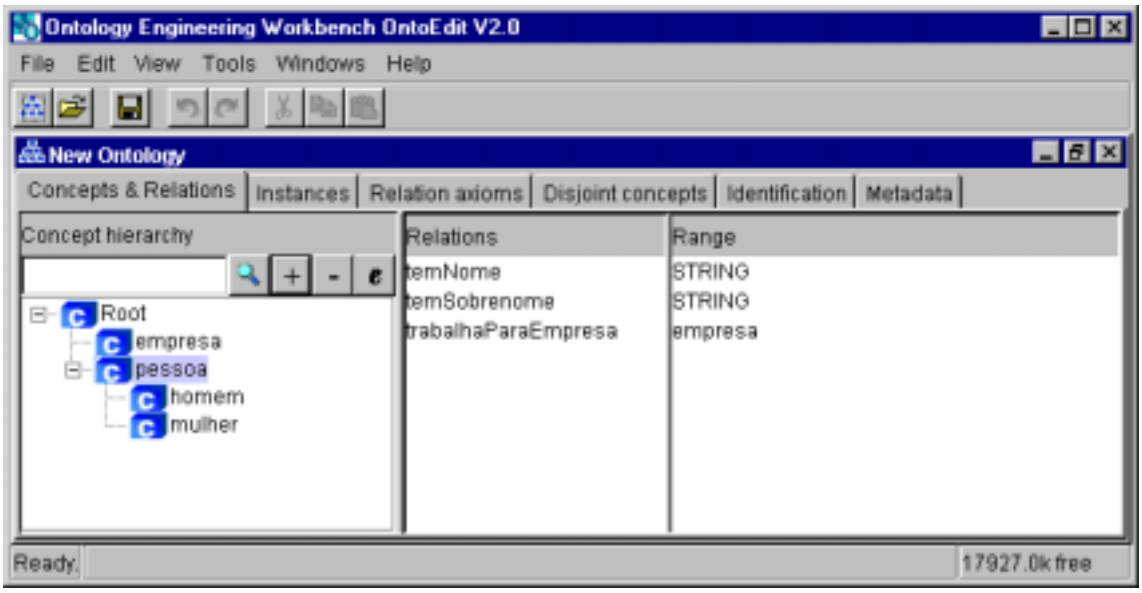

FIGURA 3.5 - OntoEdit - Ambiente para construção de Ontologias

As janelas de visualização do OntoEdit:

- Visão sobre conceitos e relações

O OntoEdit usa o paradigma da modelagem orientada a objetos para estruturar novos conceitos de um modo genérico.

O navegador de conceitos é mostrado na **FIGURA 3.**5. Usando este navegador de conceitos, os desenvolvedores podem editar a hierarquia dos conceitos através do clicar e arrastar. A introdução de um novo conceito significa a colocação do mesmo na taxonomia. Os conceitos são definidos usando-se identificadores abstratos, independentes da linguagem natural.

Tendo definido o centro da taxonomia, as relações dos conceitos e as relações entre os conceitos são modeladas em uma visão extra. A visão depende do conceito selecionado na taxonomia e é atualizado cada vez que um novo conceito é selecionado. Na parte central da **FIGURA 3.**5 pode-se identificar a janela de relações de um conceito. Similarmente aos conceitos, as relações são definidas utilizando-se identificadores abstratos. A representação e documentação externa dos atributos é atachada através de descrições de metadados.

- Visão das Instâncias

Na guia instâncias, pode-se definir, editar e remover instâncias (os fatos de acordo com o esquema consistem de conceitos e relações). Do lado esquerdo pode-se visualizar a hierarquia dos conceitos. Do lado direito todas as instâncias do conceito selecionado são listadas. Todas as instâncias de um classe são também instâncias da superclasse.

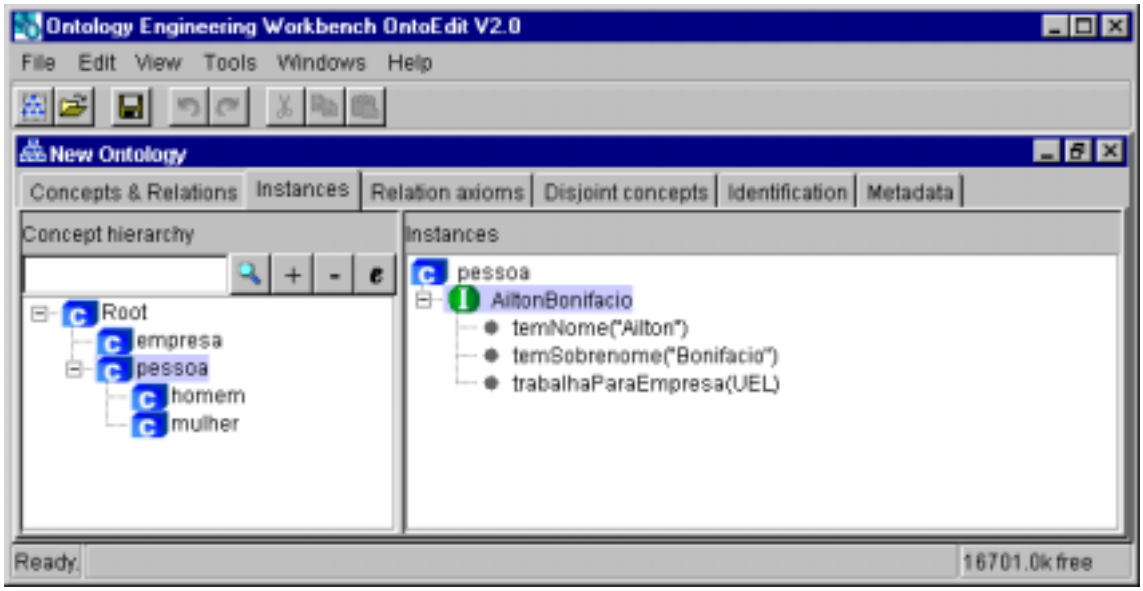

FIGURA 3.6 - OntoEdit - Guia Instâncias

- Visão dos axiomas

Consiste de múltiplas janelas que correspondem a classificação dos axiomas. Na **FIGURA 3.**7 e **FIGURA 3.**8, duas destas janelas podem ser observadas. São nestas janelas que se definem a hierarquia das relações. No OntoEdit, modelar relacionamentos

de subrelações dentro da hierarquia é tarefa simples; pois a janela da hierarquia das relações é muito similar a janela de classificação dos conceitos.

Pode-se ter axiomas simétricos, transitivos e inversos. Na **FIGURA 3.**7 pode-se observar um axioma de relação inversa.

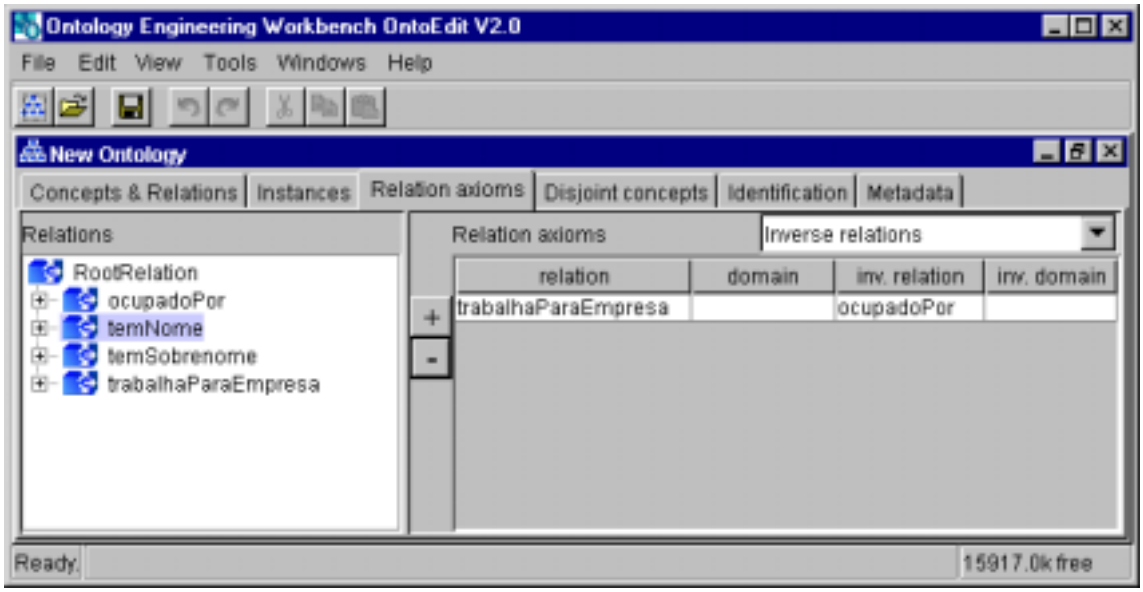

FIGURA 3.7 - OntoEdit - Guia da relação de axiomas

Na guia *Disjoint Concepts* podem ser definidos os conceitos disjuntivos. Na verdade, estes conceitos não são axiomas, mas sim restrições que podem ser utilizadas por novos *plugins* no fututo. Estas restrições podem ser utilizadas para a checagem da consistência das ontologias. Se dois conceitos são disjuntivos, não devem haver instâncias que instanciem ambos os conceitos. No momento, OntoEdit permite somente instâncias que instanciam apenas um conceito.

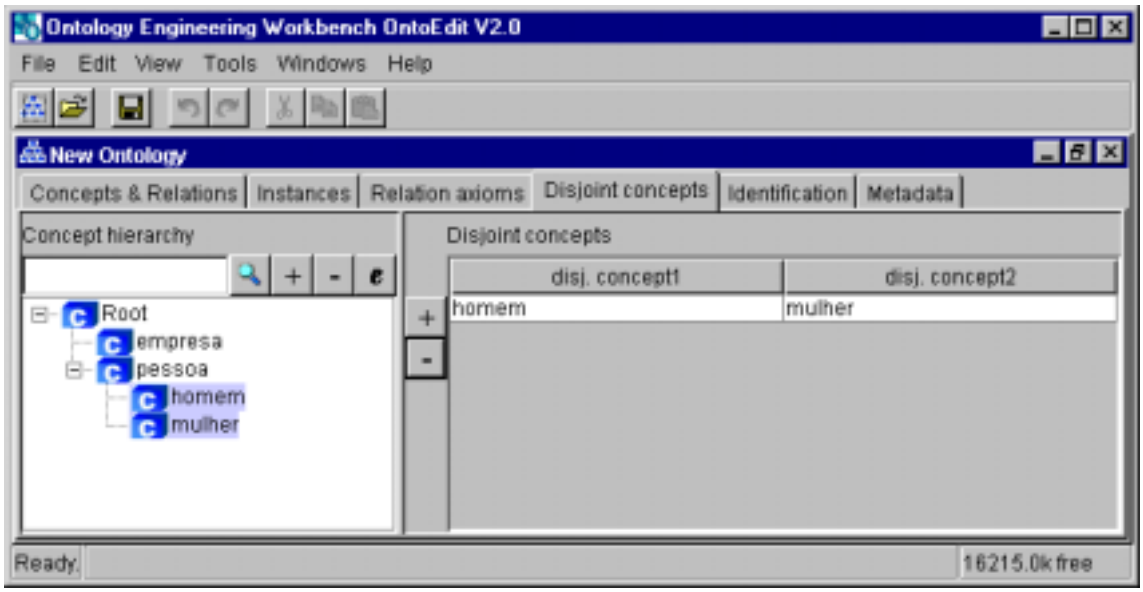

FIGURA 3.8 - OntoEdit - Guia de conceitos disjuntivos

- Visão sobre descrições gerais dos metadados

Metadados para a construção de ontologias têm sido subdivididos em metadados para a ontologia como um todo e metadados para as estruturas de conhecimento (conceitos, relações e axiomas) contidas na ontologia.

Metadados sobre a ontologia como um todo incluem características de identificação e características descritivas de uma ontologia. As características de identificação de uma ontologia incluem o nome da ontologia, a data de criação, a data da última modificação, a versão e as informações sobre os autores. Atributos descritivos armazenam informações e estatísticas sobre a ontologia. As informações gerais sobre a ontologia são decompostas em características como o tipo da ontologia, propósito, técnicas de projeto e metas da aplicação. A estatísticas sobre a ontologia são computadas automaticamente pelo OntoEdit (número de conceitos, número de relações, números de axiomas e a média e o nível mais alto de profundidade).

4 Uma Ontologia para o Currículo Lattes

O objetivo deste trabalho é apresentar, avaliar e permitir uma melhor compreensão de um conjunto de conceitos, linguagens e ferramentas que são usadas na Web Semântica. Dentre elas, linguagens para construção de ontologias e linguagens para consultas; além das ferramentas associadas que objetivam o armazenamento, manutenção e anotação em ontologias.

Para atingir este propósito, estas linguagens e ferramentas são aplicadas a um caso de dimensão e complexidade realistas, o Currículo Lattes. Então, neste capítulo, é apresentada uma ontologia em DAM+OIL para o Currículo Lattes, bem como os detalhes das etapas de desenvolvimento, métodos de instanciação e consultas sobre esta ontologia. Isto demonstrará a aplicação em um modelo real e possibilitará aos seus usuários a instanciação desta ontologia e a geração dos metadados no formato DAML+OIL. Visa demonstrar também, a classificação da ontologia através do TRIPLE e a obtenção de respostas para as consultas feitas através do RDQL. Outrossim, a simples disponibilização dos metadados para Agentes da Web é um dos objetivos a serem atingidos.

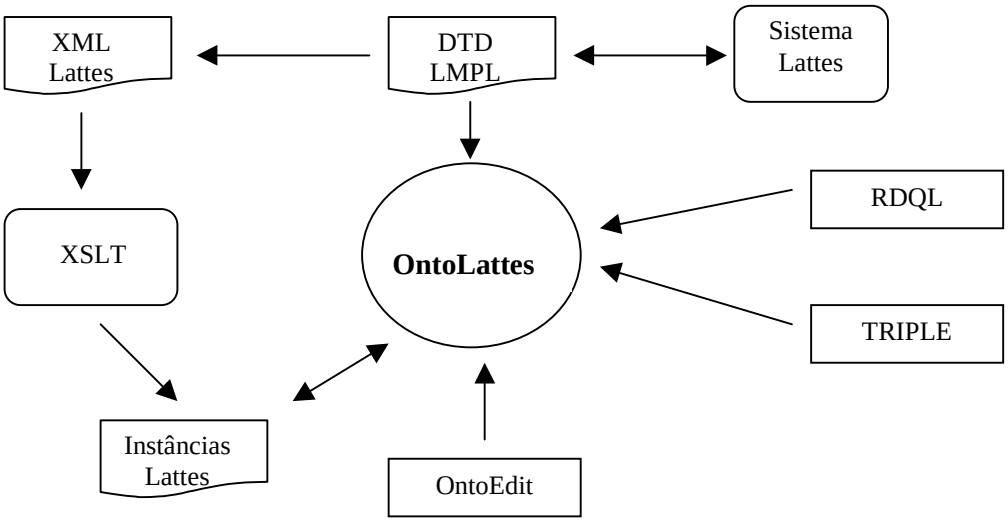

FIGURA 4.1 - Processo Global

Na **FIGURA 4.1**, pode-se verificar o fluxo do processo global da criação da ontologia OntoLattes, das formas de instanciação da ontologia, tanto manual (OntoEdit) quanto semi-automática (através do XSLT) e das linguagens de consulta e classificação que agem sobre ontologia gerada. Isto dá uma visão geral das técnicas, métodos e ferramentas utilizadas neste trabalho.

4.1 Currículo Lattes

4.1.1 Histórico

Em **[LAT 2001]**, pode-se obter um breve histórico sobre o Currículo Lattes. De 1993 a 1999, o CNPq utilizou formulários em papel, sistema em ambiente DOS (BCURR) e sistema de currículos específicos para credenciamento de orientadores (MiniCurrículo). Nesse período, a Agência acumulou cerca de 35 mil registros curriculares da atividade de C&T do País. Mediante a dificuldade de uma completa utilização em outros processos de gestão em C&T, o CNPq realizou um levantamento

junto à comunidade de consultores ad hoc visando estabelecer um modelo de currículo que atendesse tanto suas necessidades de operação de fomento como de planejamento e gestão em C&T. Em julho de 2000, a Coordenação Geral de Informática do CNPq iniciou um trabalho de intercâmbio com outras instituições ligadas à C&T no País. O resultado foi a ligação dinâmica dos currículos Lattes do CNPq com referência ao mesmo pesquisador em outras bases de dados. Ao mesmo tempo em que construiu o formulário off-line, a Coordenação Geral de Informática do CNPq também trabalhou na ferramenta on-line, que funciona sobre Plataforma Web e permite que os pesquisadores atualizem os seus currículos diretamente na base do CNPq.

Ainda no ano de 2000, as instituições federais de ensino superior reuniram suas equipes de informática no Workshop de Sistemas de Informações das IFES (UFOP - Ouro Preto) e convidaram as agências federais para construção de um modelo único de informação, visando racionalizar o processo de captura de dados no sistema federal de educação em ciência e tecnologia.

Em fevereiro de 2001, UFSC, UNICAMP, UFRJ, USP, UFRGS, UFBA e UFRN universidades que haviam procurado o CNPq solicitando abertura tecnológica de sua plataforma, participaram de workshop na Agência, visando à construção da Linguagem de Marcação da Plataforma Lattes (LMPL), sob coordenação da CGINF/CNPQ, sendo os trabalhos de desenvolvimento conduzidos pelo Grupo Stela do PPGEP/UFSC.

Desse encontro, resultou a formação da Comunidade Virtual LMPL, que definiu o modelo DTD (Data Type Definition) XML do Currículo Lattes, que faz parte da versão 1.4.0 do Sistema do Currículo Lattes, integrante da Plataforma Lattes, disponibilizado pelo CNPq. Com o mesmo, as universidades brasileiras podem agora extrair informações do Currículo Lattes e/ou gerar informações para o mesmo a partir dos seus sistemas corporativos. O projeto viabilizou a abertura da Plataforma Lattes, do ponto de vista de conteúdo dos dados, e manteve inalterado o acesso técnico às informações, preservando a segurança dos pesquisadores.

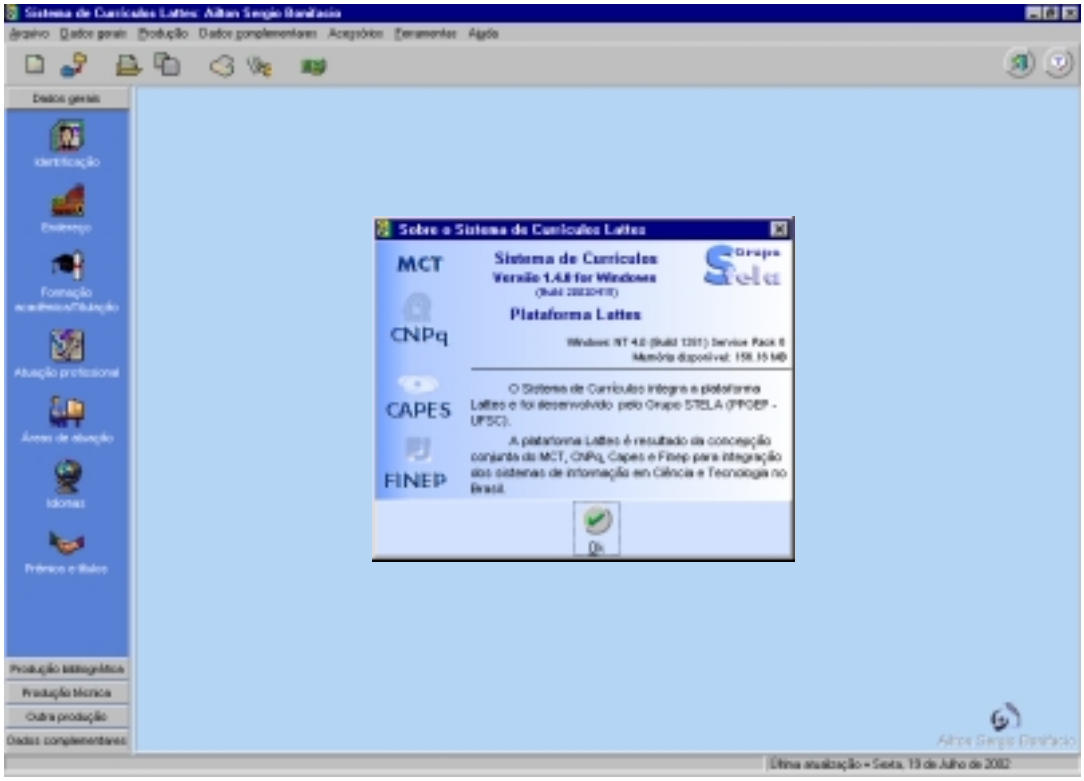

FIGURA 4.2 - Ambiente do Sistema do Currículo Lattes

4.1.2 DTD do Lattes

Os dados de um documento XML podem estar de acordo com um esquema definido através de uma DTD. Uma DTD (*Document Type Definition*) é uma gramática que restringe a forma como um conjunto de elementos pode ser organizado hierarquicamente. A cláusula <!ELEMENT ...> define um elemento e a cláusula <!ATTLIST ...> define um ou mais atributos de um elemento.

O primeiro passo dado em direção a construção da ontologia para o Currículo Lattes, foi a análise da DTD XML da Comunidade Virtual LMPL, disponibilizada juntamente com a distribuição do Sistema de Currículos do CNPq **[LAT 2001]**. Através desta análise, foi possível fazer o esboço inicial do modelo conceitual utilizado como base para a construção da ontologia OntoLattes.

Neste processo de análise da DTD do Lattes, verificou-se quais elementos desta DTD seriam definidos como conceitos (classes) e quais seriam definidos como propriedades dentro da ontologia OntoLattes. Neste sentido, definiu-se a hierarquia de classes e heranças de propriedades.

Outra tarefa de grande importância foi a verificação da cardinalidade dos elementos e dos tipos dos atributos da DTD. Isto foi feito para que a definição da ontologia siga o mais fiel quanto possível as terminologias apresentadas pela DTD.

Na **FIGURA 4.**3, que apresenta um trecho da DTD XML do Lattes, CURRICULO-VITAE é o elemento raiz da hierarquia. Este elemento é composto pelos elementos DADOS-GERAIS, PRODUCAO-BIBLIOGRAFICA, PRODUCAO-TECNICA, OUTRA-PRODUCAO e DADOS-COMPLEMENTARES. Cada elemento DADOS-GERAIS, por sua vez, possui outros elementos, cada qual com seus próprios atributos. Para cada um deles, leva-se em consideração a obrigatoriedade ou não dos elementos (cardinalidade) e os tipos de dados dos atributos.

Outra análise efetuada diz respeito a associação dos elementos a seus atributos. Por exemplo, no elemento DADOS-GERAIS há um atributo UF-NASCIMENTO associado.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- edited with XML Spy v3.5 NT (http://www.xmlspy.com) by  () -->
<!-- LMPLCurriculo.dtd                                                    -->
<!-- LMPL CNPq 2001                                                       -->
<!-- ##################################################################   -->
<!--  Este arquivo é parte integrante do DTD do Curriculo Lattes
-->
<!-- ##################################################################   -->
<!--  AVISO: Este documento e mantido pela comunidade virtual             -->
<!--         LMPL, e nAo deverá sofrer nenhuma alteracao. Havendo         -->
<!--         duvidas sobre seu conteudo, favor entrar em contato pelo     -->
<!--         e-mail lmpl@cnpq.br                                          -->
<!-- ##################################################################   -->
<!ELEMENT CURRICULO-VITAE (DADOS-GERAIS, PRODUCAO-BIBLIOGRAFICA?, PRODUCAO-
TECNICA?, OUTRA-PRODUCAO?, DADOS-COMPLEMENTARES?)>
<!ATTLIST CURRICULO-VITAE
      SISTEMA-ORIGEM-XML CDATA #REQUIRED
      xmlns:lattes CDATA #IMPLIED
>
<!ELEMENT DADOS-GERAIS (ENDERECO, FORMACAO-ACADEMICA-TITULACAO?, ATUACOES-
PROFISSIONAIS?, AREAS-DE-ATUACAO?, IDIOMAS?, PREMIOS-TITULOS?)>
<!ATTLIST DADOS-GERAIS
      NOME-COMPLETO NMTOKENS #REQUIRED
      NOME-EM-CITACOES-BIBLIOGRAFICAS CDATA #REQUIRED
      NACIONALIDADE (B | E) #REQUIRED
      CPF CDATA #IMPLIED
      NUMERO-DO-PASSAPORTE CDATA #IMPLIED
      PAIS-DE-NASCIMENTO CDATA #IMPLIED
```

```
      UF-NASCIMENTO CDATA #IMPLIED
      CIDADE-NASCIMENTO CDATA #IMPLIED
      FORMATO-DATA-DE-NASCIMENTO NMTOKEN #FIXED "DDMMAAAA"
      DATA-NASCIMENTO CDATA #IMPLIED
      SEXO (MASCULINO | FEMININO) #REQUIRED
      NUMERO-IDENTIDADE CDATA #IMPLIED
      ORGAO-EMISSOR CDATA #IMPLIED
      UF-ORGAO-EMISSOR CDATA #IMPLIED
      FORMATO-DATA-DE-EMISSAO NMTOKEN #FIXED "DDMMAAAA"
      DATA-DE-EMISSAO CDATA #IMPLIED
      NOME-DO-PAI CDATA #IMPLIED
      NOME-DA-MAE CDATA #IMPLIED
      PERMISSAO-DE-DIVULGACAO (SIM | NAO) #REQUIRED
      NOME-DO-ARQUIVO-DE-FOTO CDATA #IMPLIED
      OUTRAS-INFORMACOES-RELEVANTES CDATA #IMPLIED
>
<!ELEMENT ENDERECO (ENDERECO-PROFISSIONAL?, ENDERECO-RESIDENCIAL?)>
<!ATTLIST ENDERECO
      FLAG-DE-PREFERENCIA (ENDERECO_INSTITUCIONAL | ENDERECO_RESIDENCIAL)
#REQUIRED
>
<!ELEMENT ENDERECO-PROFISSIONAL EMPTY>
<!ATTLIST ENDERECO-PROFISSIONAL
      CODIGO-INSTITUICAO-EMPRESA CDATA #REQUIRED
      NOME-INSTITUICAO-EMPRESA CDATA #REQUIRED
      CODIGO-ORGAO CDATA #REQUIRED
      NOME-ORGAO CDATA #REQUIRED
      CODIGO-UNIDADE CDATA #REQUIRED
      NOME-UNIDADE CDATA #REQUIRED
      LOGRADOURO-COMPLEMENTO CDATA #IMPLIED
      PAIS CDATA #REQUIRED
      UF CDATA #IMPLIED
      CEP CDATA #IMPLIED
      CIDADE CDATA #IMPLIED
      BAIRRO CDATA #IMPLIED
      DDD CDATA #IMPLIED
      TELEFONE CDATA #IMPLIED
      RAMAL CDATA #IMPLIED
      FAX CDATA #IMPLIED
      CAIXA-POSTAL CDATA #IMPLIED
      E-MAIL CDATA #IMPLIED
      HOME-PAGE CDATA #IMPLIED
>
<!ELEMENT ENDERECO-RESIDENCIAL EMPTY>
<!ATTLIST ENDERECO-RESIDENCIAL
      LOGRADOURO CDATA #REQUIRED
      PAIS CDATA #REQUIRED
      UF CDATA #IMPLIED
      CEP CDATA #IMPLIED
      CIDADE CDATA #IMPLIED
      BAIRRO CDATA #IMPLIED
      DDD CDATA #IMPLIED
      TELEFONE CDATA #IMPLIED
      RAMAL CDATA #IMPLIED
      FAX CDATA #IMPLIED
      CAIXA-POSTAL CDATA #IMPLIED
      E-MAIL CDATA #IMPLIED
      HOME-PAGE CDATA #IMPLIED
>
...
```

FIGURA 4.3 - Parte da DTD XML do Lattes

Pode-se observar, por exemplo, que os elementos destacados (em negrito) na **FIGURA 4.**3 tornam-se conceitos em um modelo conceitual, ou seja, classes em uma ontologia.

Neste trabalho, a tradução desta DTD para uma ontologia (OntoLattes) foi executada manualmente. Isto deve-se ao fato do tratamento detalhado dos aspectos da hierarquia e da semântica na ontologia gerada. Outro motivo diz respeito ao fato de não existir atualmente uma ferramenta que faça esta tradução satisfatoriamente.

Os procedimentos de análise e criação das classes, propriedades, relações e axiomas para a ontologia do Currículo Lattes serão discutidos na seção 4.2 e o processo de instanciação da mesma, na seção 4.3. Para se compreender melhor o processo de definição da ontologia OntoLattes, serão apresentados trechos parciais desta ontologia, tanto na linguagem DAML+OIL quanto no editor de ontologias OntoEdit.

4.2 OntoLattes

A criação de uma ontologia para o Currículo Lattes nasceu de uma idéia de transformar a representação já consagrada do Sistema de Currículos Lattes do CNPq em uma representação conceitual. Com isto, pode-se fazer um repositório de Currículos que, disponibilizado numa linguagem com extensões semânticas, tal como o DAML+OIL, favoreça consultas aos conceitos. Outro utilidade seria a disponibilização na Web dos dados curriculares advindos do Sistema Lattes. Isto auxiliaria na extração de dados automaticamente por agentes da Web. Os estudos e pesquisas deste trabalho para a criação do OntoLattes, vêm de encontro com as tendências mundiais para o fornecimento de métodos e meios que habilitem cada vez mais a Web Semântica.

OntoLattes deve permitir aos usuários da Plataforma Lattes a utilização dos recursos disponíveis na linguagem DAML+OIL. Estes recursos enriquecem os documentos (no nosso caso, os dados dos currículos) com propriedades e conexões bem definidas. Usando estas conexões, o usuário do Sistemas de Currículos Lattes poderá conduzir consultas mais produtivas através da equalização dos termos e o uso dos conceitos da ontologia. Ou ainda, poderão criar os seus currículos e simplesmente traduzi-los para o formato DAML+OIL, disponibilizando-os na Web quando necessário.

Para a construção da ontologia para o Lattes, foi utilizado o editor de ontologias OntoEdit **[MAD 2000]**. Na verdade, OntoEdit não é o melhor e mais completo editor de ontologias. No entanto, foi escolhido por ter como grande vantagem em relação aos demais, a facilidade da exportação da ontologia para o formato DAML+OIL. O Protégé **[STA 2000]**, outro editor de ontologias, chegou a ser testado dentro deste trabalho. Uma das soluções testadas para a criação da ontologia do Currículo Lattes (OntoLattes), através do Protégé, foi a importação do documento XML gerado pelo Sistema Lattes. Isto foi testado na intenção de economizar etapas na construção da ontologia do Lattes. Este teste foi feito através de um *plugin* do Protégé chamado **XML Tab**. Entretanto, a importação não fora satisfatória no que diz respeito a hierarquia das classes e *slots* da ontologia. A importação se deu de forma linear, bem como só criou as classes que existiam no documento XML exportado do Sistema de Currículos Lattes. As demais classes que constam na DTD do Lattes não são contempladas. Outro problema encontrado com o Protégé, é que ele ainda não suporta a exportação da ontologia para o formato DAML+OIL e nem suporta as características semânticas do DAML+OIL em sua interface. Um plugin DAML+OIL (para o Protégé) está em desenvolvimento pelo grupo de pesquisa responsável pelo Protégé e DAML+OIL, mas infelizmente ainda não foi concluído. Portanto, esta abordagem foi descartada e não pode ser testada neste trabalho.

A partir da DTD do Lattes (LMPL), com os recursos do OntoEdit e a análise já citada na seção 4.1.2, foi definido o modelo ontológico para o Currículo Lattes

(OntoLattes). Nem todos os *slots*, relações, axiomas foram definidos na ontologia ainda. Isto deve-se a sua extensão e detalhamento. Outro motivo é que o propósito deste trabalho é fazer um estudo inicial e introduzir novos conceitos, técnicas e ferramentas, através da aplicação no caso Lattes, das tendências mundiais em relação as técnicas e abordagens que direcionam para a Web Semântica. A despeito de qualquer coisa, grande parte deste detalhamento já foi definida na ontologia; de maneira que seja o suficiente para os testes e avaliações que pretendem ser abordados por este trabalho, bem como os testes e avaliações para o processo de consulta semântica.

Apesar de todo trabalho disponibilizado, a ontologia gerada deveria passar, ainda, por refinamentos posteriores, por se tratar de um modelo bastante complexo e passível de muitas discussões. Entre outras coisas, este trabalho disponibiliza um estudo que permite fazer pensar na viabilidade de um padrão adicional para o Currículo Lattes.

Nesta seção, serão mostrados os principais métodos e exemplificações aplicados na construção da ontologia OntoLattes. Aliás, serão mostrados aqui, apenas alguns trechos da ontologia. A ontologia como um todo está disponível no endereço http://www.uel.br/pessoal/ailton/ontolattes.daml.

4.2.1 Nomenclatura

A grande maioria das classes, subclasses e propriedades definidas na ontologia OntoLattes, procuraram seguir a padronização dos nomes atribuídos pela DTD do Lattes. Contudo, algumas alterações tais como letras maiúsculas, *underscores* e hífens foram acrescentados ou tirados devido a restrição do editor de ontologias. Outras alterações, tais como nome de propriedades, obtiveram alterações de forma que tivessem maior semântica. Por exemplo, na DTD havia um elemento AREAS-DO-CONHECIMENTO e alguns atributos, tais como NOME-DA-SUB-AREA-DO-CONHECIMENTO;

```
!ELEMENT AREAS-DO-CONHECIMENTO (AREA-DO-CONHECIMENTO-1?, AREA-DO-CONHECIMENTO-
2?, AREA-DO-CONHECIMENTO-3?)>
<!ELEMENT AREA-DO-CONHECIMENTO-1 EMPTY>
<!ATTLIST AREA-DO-CONHECIMENTO-1
      NOME-GRANDE-AREA-DO-CONHECIMENTO (OUTROS | LINGUISTICA_LETRAS_E_ARTES |
CIENCIAS_HUMANAS | CIENCIAS_SOCIAIS_APLICADAS | CIENCIAS_AGRARIAS |
CIENCIAS_DA_SAUDE | ENGENHARIAS | CIENCIAS_BIOLOGICAS |
CIENCIAS_EXATAS_E_DA_TERRA) #REQUIRED
      NOME-DA-AREA-DO-CONHECIMENTO CDATA #IMPLIED
      NOME-DA-SUB-AREA-DO-CONHECIMENTO CDATA #IMPLIED
      NOME-DA-ESPECIALIDADE CDATA #IMPLIED
```

Na ontologia, foram definidas as classes Grande_Area_de_Conhecimento, Area, Sub-Area e Especialidade, onde:

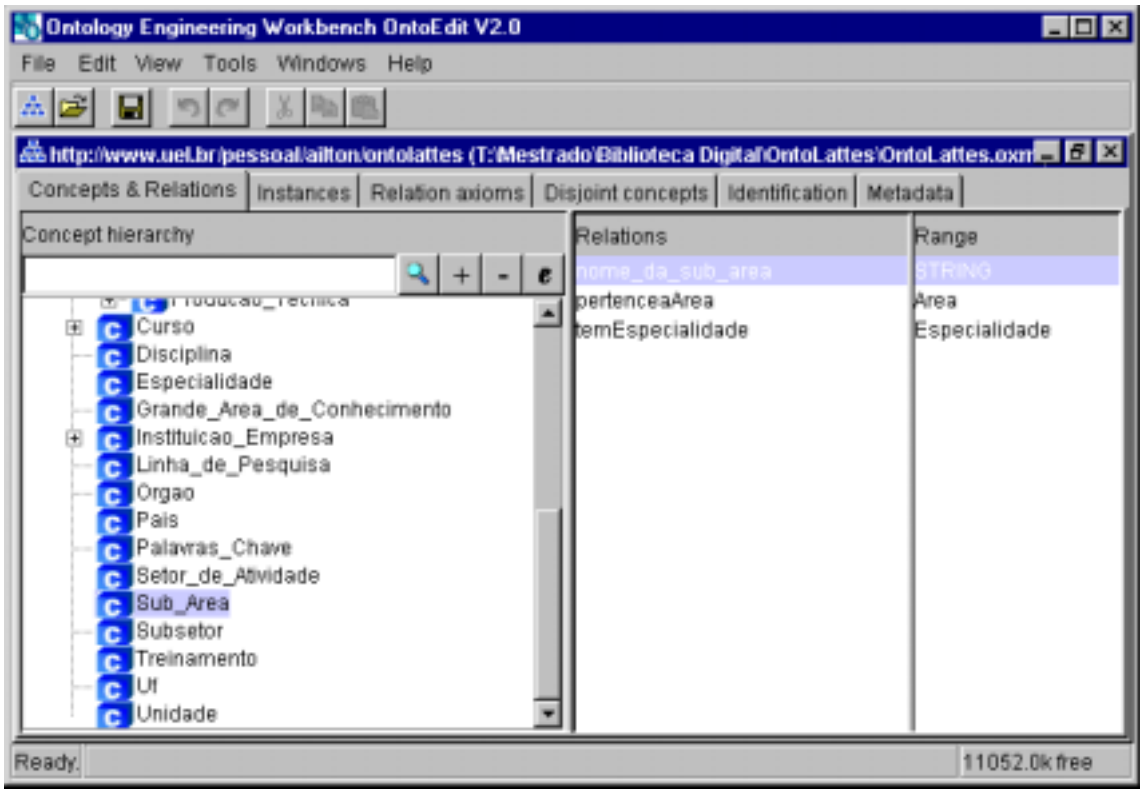

FIGURA 4.4 - Exemplo da definição da classe SubArea

a propriedade **pertenceaArea** tem como valor, uma instância da classe **Area**; e a propriedade **temEspecialidade** tem como valor, uma instância da classe **Especialidade**.

Em DAML+OIL a definição da classe **Sub_Area** seria:

```
<rdfs:Class rdf:ID="Sub_Area">
        <rdfs:subClassOf>
            <daml:Restriction daml:mincardinality="1" daml:maxcardinality="1">
                <daml:onProperty rdf:resource="#nome_da_sub_area"/>
                <daml:toClass rdf:resource="http://www.w3.org/TR/xmlschema-
2/#string"/>
            </daml:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <daml:Restriction daml:mincardinality="1" daml:maxcardinality="1">
                <daml:onProperty rdf:resource="#pertenceaArea"/>
                <daml:toClass rdf:resource="#Area"/>
            </daml:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <daml:Restriction daml:mincardinality="1" daml:maxcardinality="n">
                <daml:onProperty rdf:resource="#temEspecialidade"/>
                <daml:toClass rdf:resource="#Especialidade"/>
            </daml:Restriction>
        </rdfs:subClassOf>
</rdfs:Class>
```

FIGURA 4.5 - Exemplo da definição da classe Sub_Area em DAML+OIL

4.2.2 Definição das Classes

Para se explicar com detalhes a definição da ontologia, e principalmente a hierarquia de classes adotada, pode-se observar o trecho da DTD abaixo para uma explanação:

```
<!ELEMENT CURRICULO-VITAE (DADOS-GERAIS, PRODUCAO-BIBLIOGRAFICA?, PRODUCAO-
TECNICA?, OUTRA-PRODUCAO?, DADOS-COMPLEMENTARES?)>
<!ATTLIST CURRICULO-VITAE
      SISTEMA-ORIGEM-XML CDATA #REQUIRED
      xmlns:lattes CDATA #IMPLIED>
```

Observando este trecho da DTD e a **FIGURA 4.**6, vê-se que o elemento CURRICULO-VITAE é transformado em uma das classes que ficam somente abaixo do ROOT (classe raiz no OntoEdit), dentro da OntoLattes. Observa-se também, que os elementos que fazem parte de CURRICULO-VITAE são definidos como subclasses da classe Curriculo_Vitae na OntoLattes. São eles Dados_Gerais, Producao_Bibliografica, Producao_Tecnica, Outra_Producao e Dados_Complementares. No entanto, observa-se que outras subclasses para a classe Curriculo_Vitae foram definidas, mesmo sem possuírem esta hierarquia na DTD que as originou.

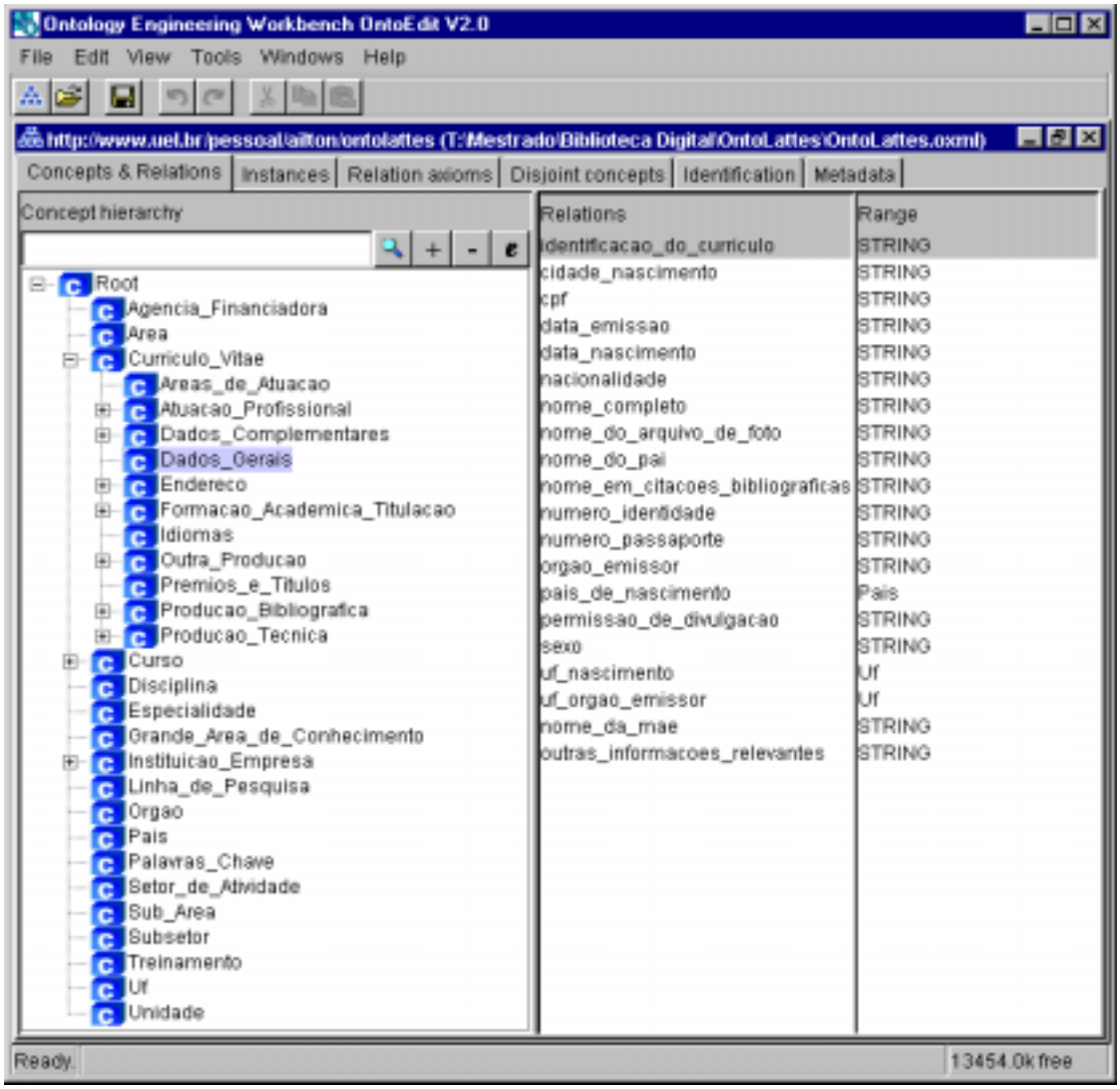

FIGURA 4.6 - Visão Geral do OntoLattes no ambiente OntoEdit

Observando os atributos do elemento DADOS-GERAIS (na DTD) pode-se explicar de forma mais clara a definição da hierarquia de classes.

```
<!ELEMENT DADOS-GERAIS (ENDERECO, FORMACAO-ACADEMICA-TITULACAO?, ATUACOES-
PROFISSIONAIS?, AREAS-DE-ATUACAO?, IDIOMAS?, PREMIOS-TITULOS?)>
<!ATTLIST DADOS-GERAIS
       NOME-COMPLETO NMTOKENS #REQUIRED
       NOME-EM-CITACOES-BIBLIOGRAFICAS CDATA #REQUIRED
       NACIONALIDADE (B | E) #REQUIRED
       CPF CDATA #IMPLIED
       NUMERO-DO-PASSAPORTE CDATA #IMPLIED
       PAIS-DE-NASCIMENTO CDATA #IMPLIED
       UF-NASCIMENTO CDATA #IMPLIED
       CIDADE-NASCIMENTO CDATA #IMPLIED
       FORMATO-DATA-DE-NASCIMENTO NMTOKEN #FIXED "DDMMAAAA"
       DATA-NASCIMENTO CDATA #IMPLIED
       SEXO (MASCULINO | FEMININO) #REQUIRED
       NUMERO-IDENTIDADE CDATA #IMPLIED
       ORGAO-EMISSOR CDATA #IMPLIED
       UF-ORGAO-EMISSOR CDATA #IMPLIED
       FORMATO-DATA-DE-EMISSAO NMTOKEN #FIXED "DDMMAAAA"
       DATA-DE-EMISSAO CDATA #IMPLIED
       NOME-DO-PAI CDATA #IMPLIED
       NOME-DA-MAE CDATA #IMPLIED
       PERMISSAO-DE-DIVULGACAO (SIM | NAO) #REQUIRED
       NOME-DO-ARQUIVO-DE-FOTO CDATA #IMPLIED
       OUTRAS-INFORMACOES-RELEVANTES CDATA #IMPLIED
>
```

FIGURA 4.7 - Definição do elemento DADOS-GERAIS na DTD

Ainda na **FIGURA 4.**6, pode-se verificar que todos os atributos do elemento DADOS-GERAIS foram definidos como propriedades da classe Dados_Gerais na ontologia. Entretanto, se os demais elementos de DADOS-GERAIS, tais como ENDERECO e FORMACAO-ACADEMICA-TITULACAO, fossem definidos como subclasses da classe **Dados_Gerais** na ontologia, estas subclasses herdariam as propriedades da classe Dados_Gerais. Isto tornaria esta definição inconsistente. Portanto, os elementos ENDERECO e FORMACAO-ACADEMICA-TITULACAO, entre outros, foram definidos como subclasses da classe Curriculo_Vitae. Desta forma, ficam no mesmo nível que a classe Dados_Gerais, na hierarquia de classes.

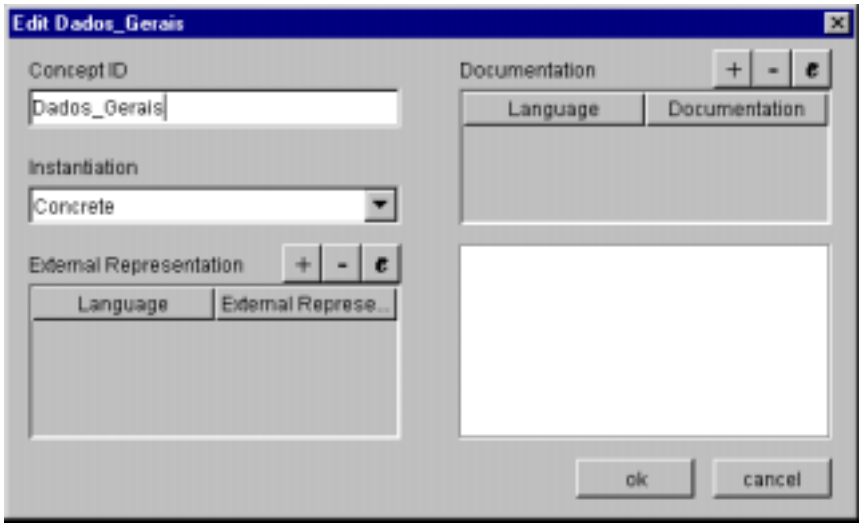

FIGURA 4.8 - Janela de edição da classe

4.2.3 Definição das propriedades

Para definir as propriedades de uma classe na ontologia, seguiu-se o padrão dos tipos de dados especificados nos atributos dos elementos da DTD do Lattes.

Entretanto, em determinados casos, a definição do tipo da propriedade (ou o *range*) provocou a definição de uma outra classe dentro da ontologia. Por exemplo, na **FIGURA 4.**7, pode-se verificar a definição dos atributos UF-NASCIMENTO e UF-ORGAO-EMISSOR, que neste caso, simplesmente são definidos como CDATA. Na ontologia OntoLattes, estes atributos são definidos como as propriedades **uf_nascimento** e **uf_orgao_emissor**. Porém, a definição do tipo de dado (ou *range*) para ambos, não é uma *string*, e sim, do tipo **Uf** (veja na janela de relações da **FIGURA 4.**6). Isto significa que as propriedades **uf_nascimento** e **uf_orgao_emissor** devem ter como valor, uma instância da classe **Uf**.

Em DAML+OIL seria:

```
<rdfs:Class rdf:ID="Dados_Gerais">
        <rdfs:subClassOf rdf:resource="#Curriculo_Vitae"/>
        <rdfs:subClassOf>
            <daml:Restriction daml:mincardinality="1" daml:maxcardinality="1">
                <daml:onProperty rdf:resource="#uf_nascimento"/>
                <daml:toClass rdf:resource="#Uf"/>
            </daml:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <daml:Restriction daml:mincardinality="1" daml:maxcardinality="1">
                <daml:onProperty rdf:resource="#uf_orgao_emissor"/>
                <daml:toClass rdf:resource="#Uf"/>
            </daml:Restriction>
    </rdfs:Class>

    <rdfs:Class rdf:ID="Uf">
        <rdfs:subClassOf>
            <daml:Restriction daml:mincardinality="1" daml:maxcardinality="1">
                <daml:onProperty rdf:resource="#pais"/>
                <daml:toClass rdf:resource="#Pais"/>
            </daml:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <daml:Restriction daml:mincardinality="1" daml:maxcardinality="1">
                <daml:onProperty rdf:resource="#nome_uf"/>
                <daml:toClass rdf:resource="http://www.w3.org/TR/xmlschema-
2/#string"/>
            </daml:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <daml:Restriction daml:mincardinality="1" daml:maxcardinality="1">
                <daml:onProperty rdf:resource="#sigla_uf"/>
                <daml:toClass rdf:resource="http://www.w3.org/TR/xmlschema-
2/#string"/>
            </daml:Restriction>
        </rdfs:subClassOf>
    </rdfs:Class>
```

FIGURA 4.9 - Exemplo de definição de classes relacionadas, em DAML+OIL

Outra coisa que devemos ressaltar na definição das propriedades de uma classe é a definição da sua cardinalidade.

Por exemplo, uma instância da classe **Palavras_Chave** pode possuir uma mínima cardinalidade de 1 e uma máxima cardinalidade de 6, conforme a **FIGURA 4.10** . Ou seja, Para cada instância na classe **Palavras_Chave**, podemos ter até 6 (seis) palavras chaves cadastradas.

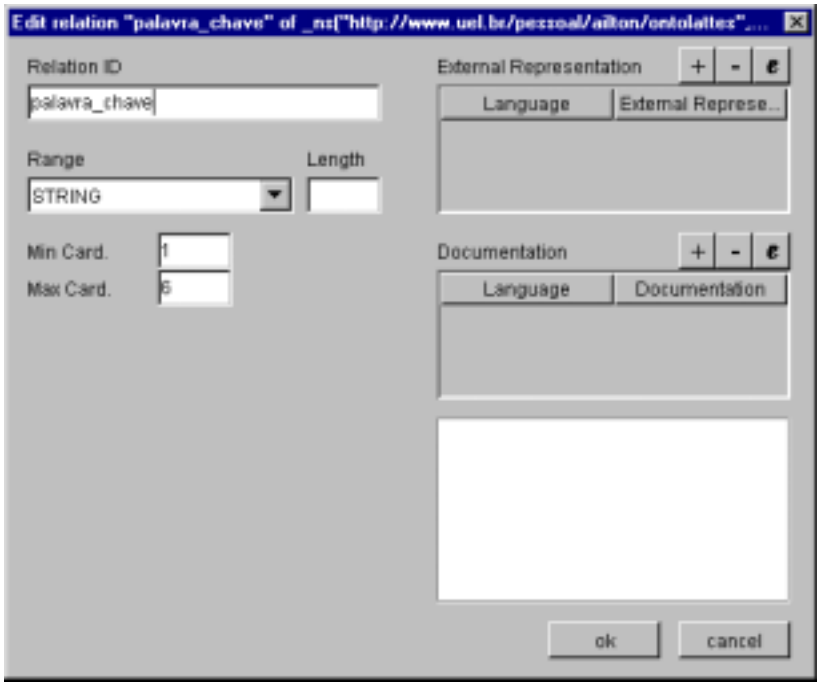

FIGURA 4.10 - Exemplo de edição de propriedade

Em DAML+OIL:

```
<rdfs:Class rdf:ID="Palavras_Chave">
    <rdfs:subClassOf>
        <daml:Restriction daml:mincardinality="1" daml:maxcardinality="6">
            <daml:onProperty rdf:resource="#palavra_chave"/>
            <daml:toClass rdf:resource=
                "http://www.w3.org/TR/xmlschema-2/#string"/>
        </daml:Restriction>
    </rdfs:subClassOf>
</rdfs:Class>
```

FIGURA 4.11 - Exemplo de definição de cardinalidade, em DAML+OIL

4.2.4 Definição dos axiomas

Na apresentação da seção 3.1.4, foi visto que DAML+OIL é, em sua essência, equivalente a uma expressiva Descrição Lógica (DL), ou seja, uma ontologia DAML+OIL correspondente a uma terminologia DL **[HOR 2002b]**. Tal como em uma DL, as classes de DAML+OIL podem ser nomes (URIs) ou expressões, além de uma variedade de construtores que são fornecidos para a construção de expressões de classes. O poder expressivo da linguagem é determinado pelos construtores de classes e propriedades que ela possui, e pelos tipos de axiomas permitidos (já apresentados na seção 3.1.4.3).

Foram declarados alguns axiomas para algumas classes e propriedades da ontologia OntoLattes. Isto se faz necessário para especificar tanto a intenção do significado das descrições, como também uma representação das descrições de quais inferências podem ser feitas automaticamente usando-se um raciocionador lógico.

Um exemplo de axioma simétrico é declarado para a propriedade **cooperaCom** que pertence a classe **Instituicao_Empresa.** O valor de uma propriedade desta instância é uma instância desta mesma classe (veja **FIGURA 4.**12).

```
<rdf:Property rdf:ID="cooperaCom">
        <rdf:type rdf:resource="daml:SymmetricProperty"/>
</rdf:Property>
```

FIGURA 4.12 - Exemplo de axioma transitivo

Alguns axiomas inversos definidos na ontologia são apresentados, na **FIGURA 4.**13. Observe que foram declaradas as propriedades **temEspecialidade** (da classe **SubArea**) inversa a propriedade **pertenceaSubArea** (da classe **Especialidade**), a propriedade **temSubArea** (da classe **Area**) inversa a propriedade **pertenceaArea** (da classe **SubArea**) e a propriedade **temArea** (da classe **Grande_Area_de_Conhecimento**) inversa a propriedade **pertenceaGrandeArea** (da classe **Area**), nesta ordem.

```
<rdf:Property rdf:ID="temEspecialidade">
        <daml:inverseOf rdf:resource="#pertenceaSubArea"/>
</rdf:Property>
<rdf:Property rdf:ID="temSubArea">
        <daml:inverseOf rdf:resource="#pertenceaArea"/>
    </rdf:Property>
<rdf:Property rdf:ID="temArea">
        <daml:inverseOf rdf:resource="#pertenceaGrandeArea"/>
</rdf:Property>
```

FIGURA 4.13 - Exemplo de axiomas inversos

No ambiente OntoEdit, pode-se, ainda, declarar os axiomas disjuntivos. No entanto, na exportação para a linguagem DAML+OIL, estes axiomas não são convertidos. Caso seja necessário, estes axiomas devem ser definidos manualmente no documento DAML+OIL gerado.

4.3 Metadados para a OntoLattes

A **FIGURA 4.**3 mostra a DTD da Comunidade LMPL para o Currículo Lattes de forma parcial. Já a **FIGURA 4.**14, mostra parcialmente o documento XML gerado pelo Sistema de Currículo Lattes válido para aquela DTD.

Estes dados do documento XML apresentado correspondem aos mesmos dados de um determinado usuário, cadastrados no Sistema do Currículo Lattes.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--edited with XML Spy v4.4 U (http://www.xmlspy.com) by Banco de Dados (UFRGS)-->
<!DOCTYPE CURRICULO-VITAE SYSTEM "..\xmltodaml\LMPLCurriculo.dtd">
<!-- A T E N Ç Ã O !!!  PARA VISUALIZAR ESTE DOCUMENTO XML NO FORMATO HTML, EXCLUA
OS CARACTERES '< ! - - ' E ' - - >' DA LINHA ABAIXO. -->
<!--<?xml-stylesheet type="text/xsl" href="..\XSL\LattesRelHtmlABNT.xsl"?>-->
<CURRICULO-VITAE SISTEMA-ORIGEM-XML="LATTES_OFFLINE" xmlns:lattes=
"http://www.cnpq.br/2001/XSL/Lattes">
```

```
<DADOS-GERAIS
       NOME-COMPLETO="Ailton Sergio Bonifacio"
       NOME-EM-CITACOES-BIBLIOGRAFICAS="BONIFACIO, A. S."
       NACIONALIDADE="B" CPF="09739477801"
       PAIS-DE-NASCIMENTO="Brasil" UF-NASCIMENTO="PR"
       CIDADE-NASCIMENTO="Londrina"
       DATA-NASCIMENTO="26111968" SEXO="MASCULINO"
       NUMERO-IDENTIDADE="85861506"
       ORGAO-EMISSOR="SSP"
       UF-ORGAO-EMISSOR="PR"
       DATA-DE-EMISSAO="30111998"
       NUMERO-DO-PASSAPORTE=""
       NOME-DO-PAI="Alexandre Bonifacio"
       NOME-DA-MAE="Setsuko Higa Bonifacio" PERMISSAO-DE-DIVULGACAO="SIM" OUTRAS-
       INFORMACOES-RELEVANTES="">
       <ENDERECO FLAG-DE-PREFERENCIA="ENDERECO_INSTITUCIONAL">
       <ENDERECO-PROFISSIONAL
              CODIGO-INSTITUICAO-EMPRESA="008000000006"
              NOME-INSTITUICAO-EMPRESA="Universidade Estadual de Londrina"
              CODIGO-UNIDADE="008024001991"
              NOME-UNIDADE="Serviço de Apoio Técnico"
              CODIGO-ORGAO="008024000006"
              NOME-ORGAO="Coordenadoria de Recursos Humanos"
              PAIS="Brasil" UF="PR"
              LOGRADOURO-COMPLEMENTO="Campus Universitário"
              BAIRRO=""
              CIDADE="Londrina"
              CAIXA-POSTAL="6001"
              CEP="86051990" DDD="43" TELEFONE=" 3714202" RAMAL="" FAX=" 3714101"
              E-MAIL="ailton@uel.br" HOME-PAGE="http://www.uel.br/pessoal/ailton"/>
       <ENDERECO-RESIDENCIAL PAIS="Brasil" UF="PR"
              LOGRADOURO="Rua Brasil, 562 Casa 21"
              BAIRRO="Jardim Brasil" CIDADE="Londrina"
              AIXA-POSTAL="" CEP="86063260" DDD="43" TELEFONE=" 3222254"
              RAMAL="" FAX="" E-MAIL="ailton@uel.br"
              HOME-PAGE="http://www.uel.br/pessoal/ailton"/>
       </ENDERECO>
...
```

FIGURA 4.14 - Dados exportados pelo Sistema Lattes em XML - Visão parcial

Para a instanciação da ontologia foi utilizada uma pequena massa de dados de alguns currículos advindos do Sistema Lattes.

Neste trabalho, o processo de instanciação da OntoLattes seguiu dois caminhos. No primeiro, o processo de instanciação foi manual. No segundo, o processo de instanciação foi através de um processo semi-automatizado de tradução do documento XML para a ontologia OntoLattes em DAML+OIL, utilizando-se dos recursos do XSLT **[KAY 2000]**.

Do ponto de vista da instanciação, tanto um como outro procedimento são bastante manuais e passíveis de substituição por um método mais automatizado. Porém não há, atualmente, ferramentas que executem este processo automaticamente. Este método automatizado deveria instanciar a ontologia através de um *plugin* para um editor de ontologias ou *interface*. Este *plugin* ou *interface* faria, a partir do documento XML gerado pelo sistema Lattes, as equivalências quanto a nomenclatura e a estrutura, e convertesse automaticamente todos os dados disponíveis para o modelo OntoLattes, ou para qualquer outra ontologia em DAML+OIL. Porém, um processo genérico de tradução requer muita pesquisa e estudo para se chegar a resultados satisfatórios na execução da análise e conversão destas equivalências. Infelizmente, até o momento, não fora desenvolvido nada de efetivo neste sentido.

4.3.1 Instanciação através do OntoEdit

Uma das formas de instanciação da ontologia foi através da inserção das instâncias uma a uma, manualmente. Para tanto, foi utilizado o próprio OntoEdit, o mesmo editor utilizado para a edição da ontologia.

Na guia de instâncias do editor OntoEdit, podem ser definidas, editadas e removidas as instâncias, que são os fatos que estão de acordo com a consistência dos conceitos e relações da ontologia. Observando-se a **FIGURA 4.**15 pode-se verificar a hierarquia dos conceitos da OntoLattes na janela da esquerda e todas as instâncias relativas ao conceito selecionado na janela a direita. Neste caso em particular, todas as instâncias da ontologia estão sendo exibidas, já que o Root é quem está selecionado. Isto é devido a hierarquia dos conceitos, onde todas as instâncias de uma classe, são também instâncias da sua superclasse.

Para definir uma nova instância no OntoEdit, basta selecionar a classe na qual deseja-se instanciar, adiconá-la e dar um nome para seu identificador.

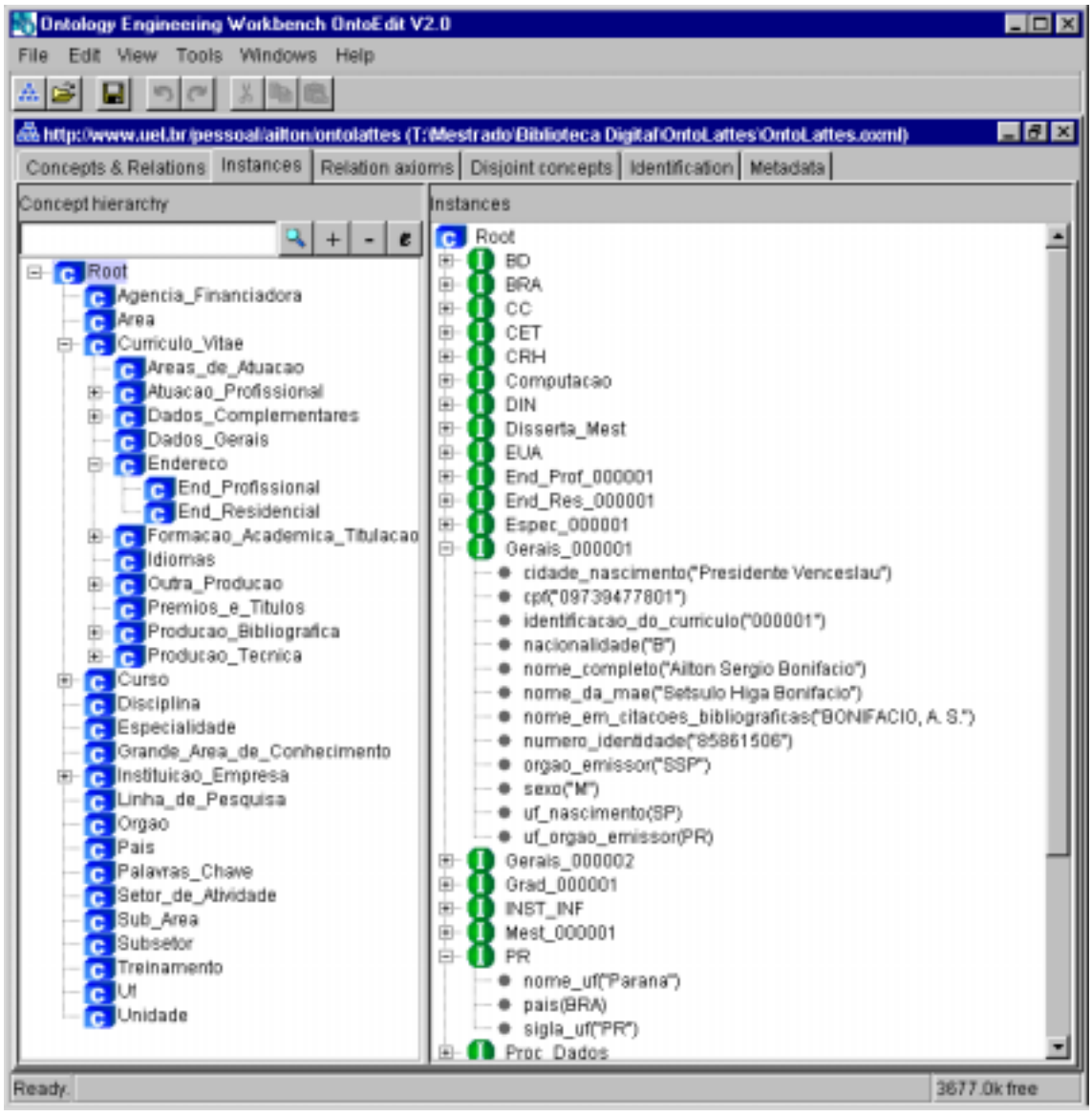

FIGURA 4.15 - Janela de Instâncias do OntoEdit - OntoLattes

A edição das instâncias segue o mesmo padrão. Após adicionar uma instância, edita-se a mesma através da caixa de diálogo apresentada na **FIGURA 4.**16. Nesta caixa de diálogo pode-se modificar o identificador da instância e o conjunto de valores das suas relações (propriedades). Na coluna da esquerda da tabela, são apresentadas as relações daquela instância e, para cada relação escolhida, na coluna da direita deve ser preenchido o valor da mesma.

Se a propriedade a ser preenchida for do tipo “STRING”, “NUMBER” ou outro literal, simplesmente preenche-se o valor. No entanto, se o tipo, ou *range* da propriedade for uma outra classe (por exemplo, Uf), será disponibilizado uma lista com todas as instâncias desta classe (conforme **FIGURA 4.**16).

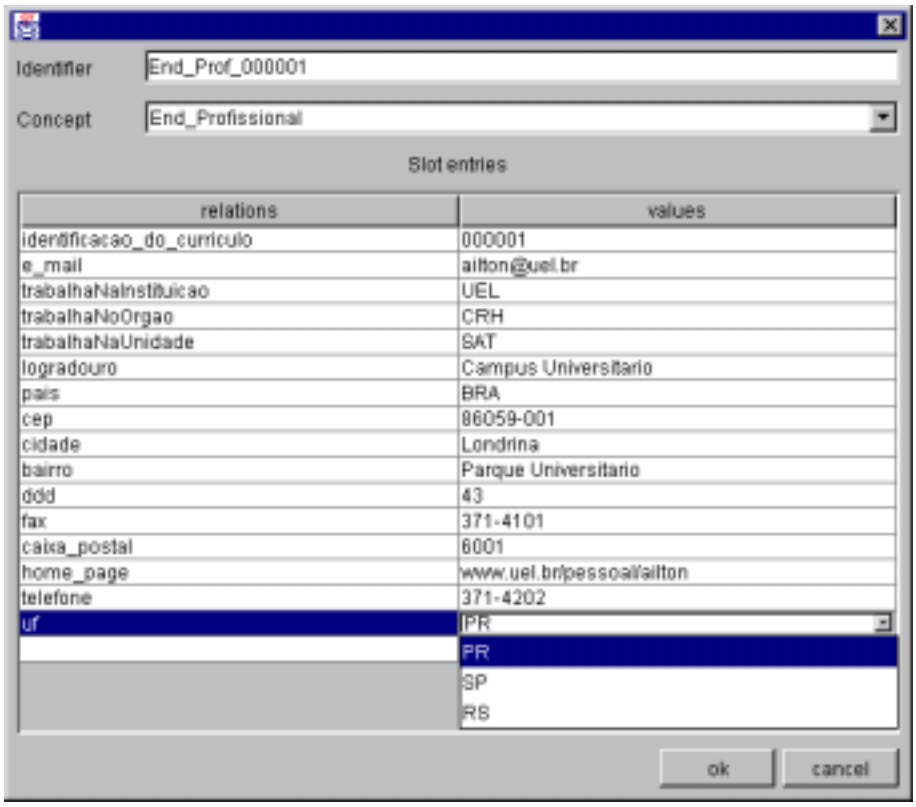

FIGURA 4.16 - Janela de edição de instâncias - OntoEdit

Para efetuar a remoção de uma instância basta selecioná-la e removê-la na sequência. Ou pode-se apenas remover um par propriedade/valor.

Apesar de ser um processo muito simples, fácil e intuitivo de operar, a instanciação manual através do OntoEdit se mostra muito lenta e trabalhosa. Para grandes bases ontológicas, com muitas classes, propriedades e relações definidas, tal como o OntoLattes, este processo não se torna viável.

4.3.2 Instanciação através do XSLT

O processo de instanciação semi-automático por meio do XSLT visa otimizar a tradução dos modelos XML gerados pelo Sistema Lattes para o modelo em DAML+OIL utilizado pelo OntoLattes. Esta otimização é de grande valia, visto que

tem-se a perspectiva de uma possível utilização desta ontologia como um grande repositório de currículos. Sendo assim, se o uso da ontologia não for apenas para a disponibilização na Web, de um currículo em particular, o processo manual de instanciação via OntoEdit, perde totalmente o sentido. O processo semi-automático torna-se, então, a opção mais adequada.

Para tanto, um modelo de instanciação semi-automático é descrito abaixo. Vale notar que isto não esgota os esforços para buscas de processos genéricos de conversão e/ou tradução de documentos XML para DAML+OIL.

Este trabalho visa prover um modelo de tradução particular para o caso Lattes.

4.3.2.1 Lógica da Transformação

XSLT é uma linguagem de transformação de documentos XML em outros documentos XML. Em XSLT, um vocabulário específico em XML é incluído para se especificar as transformações.

A medida em que DAML+OIL é uma extensão do modelo RDFS, que utiliza a sintaxe XML, é perfeitamente possível utilizar o XSLT para fazer a transformação de um documento XML para DAML+OIL.

Nesta subseção será apresentada a lógica da transformação aplicado particularmente sobre uma parte do documento XML do Lattes. Presume-se que uma vez apresentada esta lógica para uma parte do documento XML, esta lógica valeria para o restante deste documento também.

Para entender como foi especificada a transformação dos documentos, deve-se observar, em primeiro lugar, o documento XML gerado, tal como apresentado na **FIGURA 4.**14.

A transformação XSLT gera as instâncias para as classes “Dados_Gerais” e “Endereco_Profissional”, juntamente com as respectivas instâncias relacionadas a elas (classes “Uf” e “Pais”). Para esclarecer este ponto deve-se recordar, por exemplo, que o valor da propriedade “uf_nascimento” é uma instância da classe “Uf”. Este é um dos pontos críticos da transformação, e será explicada a contento mais adiante. Estas instâncias são geradas a partir de um documento XML gerado pelo Sistema de Curriculos Lattes e traduzidas para instâncias dentro do modelo ontológico em DAML+OIL (OntoLattes).

Para este trecho do documento XML, a transformação é realizada através de dois *templates* básicos (<xsl:template match="DADOS-GERAIS"> e <xsl:template match="ENDERECO-PROFISSIONAL">). Estes *templates* são associados aos elementos DADOS-GERAIS e ENDERECO-PROFISSIONAL do documento XML, respectivamente. Outros *templates* adicionais, que fazem o processamento das instâncias associadas, são necessários também. Dentre outros, o **monta-lista-uf**, **monta-lista-paises**, **gera-uf** e **gera-pais**, exemplificados na sequencia desta seção.

A dificuldade nesta transformação está em criar apenas uma instância de cada Uf (ou País), mesmo que apareçam mais de uma referência aos mesmos ao longo do documento. Por exemplo, se em um determinado currículo existe referência à propriedade UF 'RS', tanto no atributo UF-NASCIMENTO do elemento DADOS_GERAIS, como no atributo UF do elemento ENDERECO-PROFISSIONAL, apenas uma instância da UF 'RS' deve ser criada na classe “Uf” da ontologia OntoLattes, e não duas. Isto é contornado através da criação de listas de Ufs e Países, de tal forma que elementos repetidos não apareçam duas vezes na lista.

```
<xsl:call-template name="monta-lista-uf">
   <xsl:with-param name="lista-uf"/>
   <xsl:with-param name="contador" select="1"/>
</xsl:call-template>

<xsl:call-template name="monta-lista-paises">
   <xsl:with-param name="lista-uf" select="$lista-uf"/>
   <xsl:with-param name="lista-pais"/>
   <xsl:with-param name="contador" select="1"/>
</xsl:call-template>
```

FIGURA 4.17 - Templates para montar as listas - XSLT

A criação dessas listas é feita da seguinte maneira:

Existem dois templates, **monta-lista-uf** e **monta-lista-pais**. A função desses templates é gerar as listas de Ufs e Países, sem que haja redundância no conteúdo dos mesmos. Eles funcionam recursivamente, controlando as iterações através de um contador. A cada iteração, um determinado campo do documento (referente a uma UF ou país) é tratado. Caso a UF ou país já exista na lista, nada é feito; o contador é incrementado e a função é chamada recursivamente. Caso contrário, a nova UF ou país é concatenada ao final da lista.

A partir dessas listas, são geradas as instâncias correspondentes. Dois templates são responsáveis por isso: **gera-uf** e **gera-pais**. Seu funcionamento também é recursivo: a cada iteração, um elemento da lista é retirado e tratado. Ao final, o template é chamado recursivamente, reiniciando o ciclo, até que todos os elementos da lista tenham sido gerados.

```
<xsl:call-template name="gera-uf">
   <xsl:with-param name="lista-uf" select="$lista-uf"/>
   <xsl:with-param name="lista-pais" select="$lista-pais"/>
</xsl:call-template>

<xsl:call-template name="gera-pais">
   <xsl:with-param name="lista-pais" select="$lista-pais"/>
</xsl:call-template>
```

FIGURA 4.18 - Templates que geram listas - XSLT

A separação entre um elemento e outro da lista é feito através de um ponto-e-vírgula. Na criação das listas, toda vez que um elemento é concatenado, ao final é concatenado também um *string* contendo um ponto-e-vírgula. Assim, a criação das instâncias torna-se bastante simplificada: o elemento atualmente sendo tratado é o

conteúdo da lista à esquerda do primeiro ponto-e-vírgula (`Xpath: substring-before($lista-pais,';') `). Da mesma forma, a remoção do primeiro elemento da lista é feito selecionando-se o conteúdo da mesma à direita do primeiro ponto-e-vírgula (`Xpath: substring-after($lista-pais,';') `).

Como dito anteriormente, este trabalho visa prover um modelo que demonstra parcialmente como seria o modelo completo de tradução para o caso Lattes em particular. O arquivo XSLT de tranformação parcial aplicado tem o seu fonte disponível no anexo 2 deste trabalho.

4.4 Consultas na OntoLattes

Dentre as linguagens de consultas para os modelos DAML+OIL analisadas, foram utilizadas, neste trabalho, TRIPLE e RDQL. A escolha deve-se ao fato destas duas linguagens possuírem características específicas para modelo RDFS. Por extensão, aplicam-se a modelos DAML+OIL; já que DAML+OIL é um meta modelo do RDFS. Como visto na seção 3.1.4, não se pode esquecer que este modelo é uma linguagem de ontologias que usa a sintaxe XML, o modelo RDF e mecanismos de inferência baseados em Description Logics.

4.4.1 TRIPLE sobre OntoLattes

Como já mencionado anteriormente, TRIPLE, é uma linguagem de regras que objetiva dar suporte as aplicações que tenham necessidade de tranformação e raciocínio em RDF. TRIPLE é definida em camadas. Uma suporta as extensões da lógica Horn para sustentar as construções básicas em RDF, tais como, recursos e declarações, e a outra suporta módulos para as extensões semânticas do RDF, tais como, RDF Schema, OIL e DAML+OIL, que são implementadas diretamente em TRIPLE ou por meio de interação com componentes de raciocínio externos.

Nesta subseção, será submetida uma parte da ontologia OntoLattes para se demonstrar como TRIPLE pode classificar uma ontologia em DAML+OIL e como pode ser traduzida a ontologia para que haja a integração de modelos via TRIPLE. Para tanto, um modelo daml_oil (Mdl) em TRIPLE irá utilizar do classificador de descrições lógicas a fim de se compreender a semântica do DAML+OIL.

No exemplo abaixo, foram aplicadas algumas regras para que o TRIPLE verifique a integridade e classificação das classes/conceitos da ontologia desejada através do FaCT, analisando a satisfabilidade da ontologia. Neste exemplo foi utilizado apenas uma parte da ontologia OntoLattes.
Linha de comando:

```
daml -lisp ontolattes-parcial.daml
```

onde:

o parâmetro *-lisp* faz a transformação da ontologia para SHIQ em sintaxe LISP, e verifica a satisfabilidade das classes/conceitos.

Exemplo de trecho da ontologia OntoLattes em DAML+OIL, onde são declaradas algumas classes, subclasses, propriedades e dois conceitos disjuntivos:

```
<rdfs:Class rdf:ID="Doutorado">
       <rdfs:subClassOf rdf:resource="#Mestrados_Doutorado"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Mestrado">
       <rdfs:subClassOf rdf:resource="#Mestrados_Doutorado"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Mestrado_Profissionalizante">
       <rdfs:subClassOf rdf:resource="#Mestrados_Doutorado"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Mestrados_Doutorado">
       <rdfs:subClassOf rdf:resource="#Formacao_Academica_Titulacao"/>
       <daml:disjointUnionOf rdf:parseType="daml:collection">
              <daml:Class
              rdf:about="http://www.uel.br/pessoal/ailton/ontolattes#
              Mestrado"/>
              <daml:Class
              rdf:about="http://www.uel.br/pessoal/ailton/ontolattes#
              Doutorado"/>
              <daml:Class
              rdf:about="http://www.uel.br/pessoal/ailton/ontolattes#
              Mestrado_Profissionalizante"/>
       </daml:disjointUnionOf>
       <rdfs:subClassOf>
              <daml:Restriction daml:mincardinality="1"
        daml:maxcardinality="1">
              <daml:onProperty rdf:resource="#codigo_agencia_financiadora"/>
              <daml:toClass rdf:resource="#Agencia_Financiadora"/>
              </daml:Restriction>
       </rdfs:subClassOf>
       ...
</rdfs:Class>
```

FIGURA 4.19 - Trecho da OntoLattes para transformação em SHIQ via TRIPLE

Após a transformação para *SHIQ*:

```
(defconcept ontolattes+Mestrados_Doutorado)
(defconcept ontolattes+Doutorado)
(defconcept ontolattes+Mestrado_Profissionalizante)
(defconcept ontolattes+Mestrado)
(defrole ontolattes+codigo_agencia_financiadora)
...
(implies ontolattes+Mestrado (not ontolattes+Doutorado))
(implies ontolattes+Mestrado (not ontolattes+Mestrado_Profissionalizante))
(implies ontolattes+Doutorado (not ontolattes+Mestrado_Profissionalizante))
(equal ontolattes+Mestrados_Doutorado (or ontolattes+Mestrado
ontolattes+Doutorado ontolattes+Mestrado_Profissionalizante))
;;; namespaces:
;;; ontolattes = http://www.uel.br/pessoal/ailton/ontolattes#
```

FIGURA 4.20 - OntoLattes parcial transformado em SHIQ - sintaxe LISP

Onde *defconcept* indica uma classe em DAML+OIL, *defrole* indica uma propriedade e *implies* mostra o significado de regras, neste caso, a regra de disjunção dos conceitos Mestrado, Doutorado e Mestrado_Profissionalizante.

Linha de comando:

```
daml -xml ontolattes-parcial.daml
```

onde:

o parâmetro *-xml* faz a transformação da ontologia, também para SHIQ, porém em sintaxe XML, tal como é aceita para os raciocinadores FaCT e RACER. Também verifica a satisfabilidade das classes/conceitos.

Após a transformação do modelo parcial da ontologia OntoLattes apresentada na **FIGURA 4.**21 para *SHIQ* em sintaxe XML:

```
<KNOWLEDGEBASE>
<DEFCONCEPT NAME="ontolattes+Mestrados_Doutorado"/>
<DEFCONCEPT NAME="ontolattes+Doutorado"/>
<DEFCONCEPT NAME="ontolattes+Mestrado_Profissionalizante"/>
<DEFCONCEPT NAME="ontolattes+Mestrado"/>
<DEFROLE NAME="ontolattes+codigo_agencia_financiadora"/>
...
<IMPLIESC><CONCEPT><PRIMITIVE
NAME="ontolattes+Mestrado"/></CONCEPT><CONCEPT><NOT><PRIMITIVE
NAME="ontolattes+Doutorado"/></NOT></CONCEPT>
</IMPLIESC>
<IMPLIESC><CONCEPT><PRIMITIVE
NAME="ontolattes+Mestrado"/></CONCEPT><CONCEPT><NOT><PRIMITIVE
NAME="ontolattes+Mestrado_Profissionalizante"/></NOT></CONCEPT>
</IMPLIESC>
<IMPLIESC><CONCEPT><PRIMITIVE
NAME="ontolattes+Doutorado"/></CONCEPT><CONCEPT><NOT><PRIMITIVE
NAME="ontolattes+Mestrado_Profissionalizante"/></NOT></CONCEPT>
</IMPLIESC>
<EQUALC><CONCEPT><PRIMITIVE
NAME="ontolattes+Mestrados_Doutorado"/></CONCEPT><CONCEPT><OR><PRIMITIVE
NAME="ontolattes+Mestrado"/><PRIMITIVE NAME="ontolattes+Doutorado"/><PRIMITIVE
NAME="ontolattes+Mestrado_Profissionalizante"/></OR></CONCEPT></EQUALC>
<!-- namespaces:
  ontolattes = http://www.uel.br/pessoal/ailton/ontolattes#
-->
</KNOWLEDGEBASE>
```

FIGURA 4.21 - OntoLattes parcial transformado em SHIQ - sintaxe XML

Para se fazer a classificação da ontologia através do raciocinador FaCT, submete-se ao mesmo, a fonte em *SHIQ* gerada pelo TRIPLE. Desta forma, tem-se a saída com as mensagens da análise efetuada:

```
135 concepts
153 roles
135 impliesc
Compilation completed.
T:\OntoLattes\OntoLattes classified in 2 seconds.
Checking Unsatisfiable Classes
Done Checking Classes

Checking Subsumptions
Done Checking Subsumptions

Checking Instances
Done Checking Instances

T:\OntoLattes\OntoLattes checked in 5 seconds.
```

4.4.2 OntoLattes sob o DAML Viewer

DAML Viewer **[VIE 2001]** provê um meio de visualizar as instâncias encontradas em um documento DAML+OIL. Pode-se visualizar uma ontologia em um arquivo local ou uma URI de um documento contendo DAML+OIL.

A janela de nodos fornece uma lista das URIs para todos os recursos referenciados no documento. Em cada nodo que se clica, uma janela de visualização de nodos é aberta mostrando todas as declarações que envolvem aquele nodo.

Declarações marcadas com uma seta para frente ("→") são aquelas onde a URI que está sendo mostrada na barra de títulos da janela é o sujeito, e o Valor é o objeto. Já as declarações marcadas com um seta para trás ("←") são aquelas onde a URI que está sendo mostrada na barra de títulos é o objeto, e o Valor é o sujeito.

Algumas interações podem ser executadas na janela de visualização dos nodos:

- Pode-se clicar nas entradas na coluna "Value" para se abrir uma nova janela de visualização de nodo para o nodo indicado, sendo que valores literais não respondem ao clique do mouse;
- Pode-se clicar nas entradas na coluna "Property" para abrir uma nova janela de visualização de nodo para a propriedade indicada (caso as definições da ontologia estejam incluídas no arquivo carregado);

Para exemplicar, a **FIGURA 4.22** mostra um visão da janela de nodos para algumas instâncias da ontologia OntoLattes.

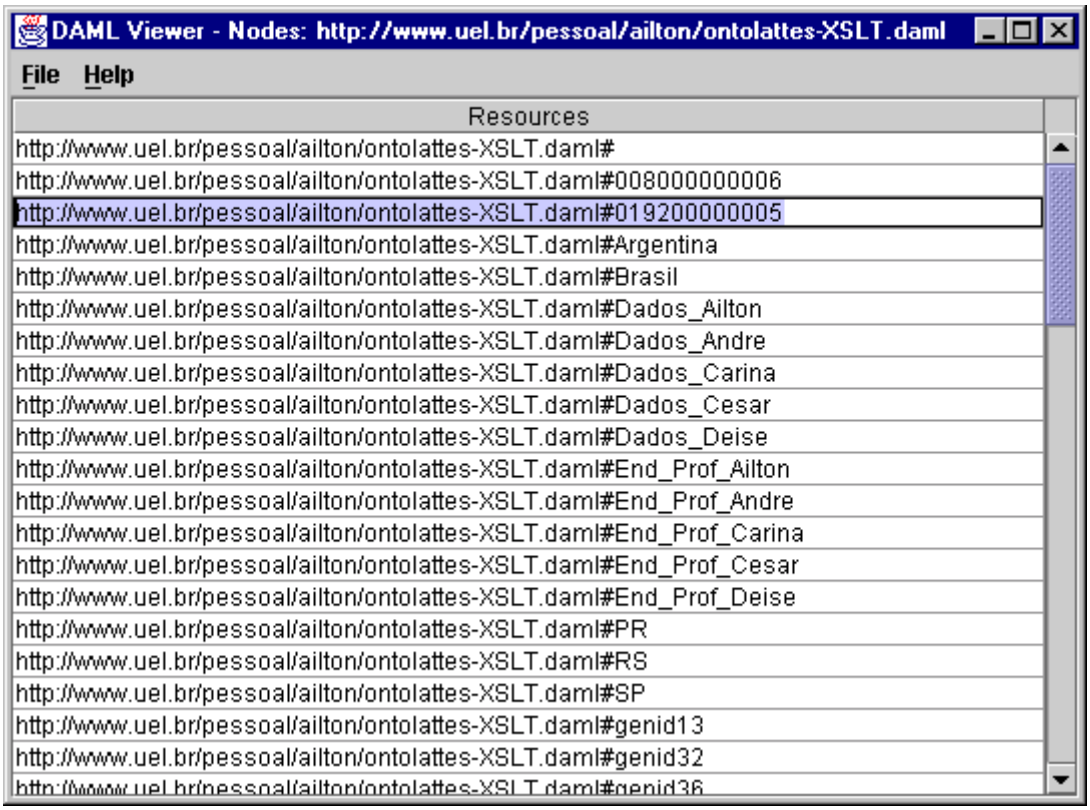

FIGURA 4.22 - DAML Viewer – Janela de nodos (OntoLattes)

Observa-se que ao clicar no nodo em destaque (código da Instituição_Empresa advindo do XML gerado pelo sistema Lattes), são apresentados, em outra Janela de visualização de nodos, todos os nodos relacionados a ele.

Observe que na **FIGURA 4.**23, a relação de nodos estabelecida pela propriedade **trabalhaNaInstituicao**, da classe **End_Profissional**. Esta relação se dá pelo fato da mesma ter como valor, uma instância da classe **Instituicao_Empresa**.

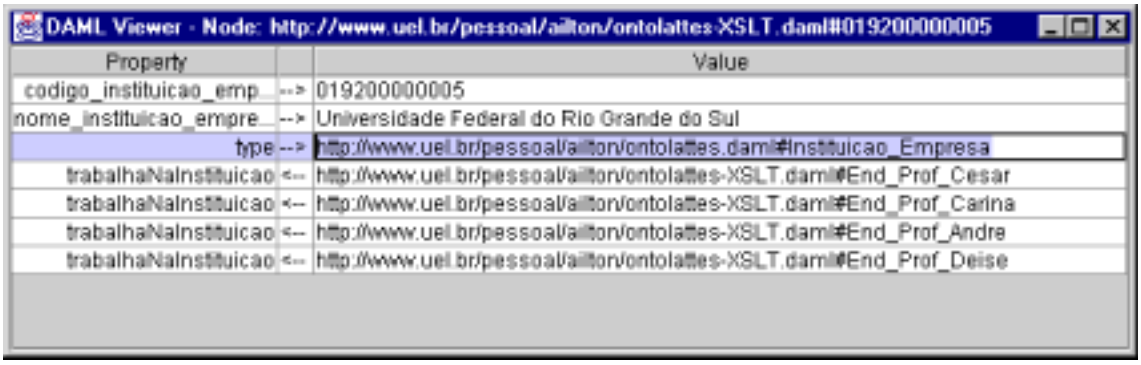

FIGURA 4.23 - DAML Viewer – Nodos relacionados

A relação inversa é estabelecida. Para se observá-la, basta clicar no nodo selecionado na **FIGURA 4.**23. Neste caso, tem-se a propriedade **Type** e o valor **Instituicao_Empresa**, e observando-se o resultado na **FIGURA 4.**24, temos a visualização dos nodos de todas as empresas que fazem parte da base ontológica.

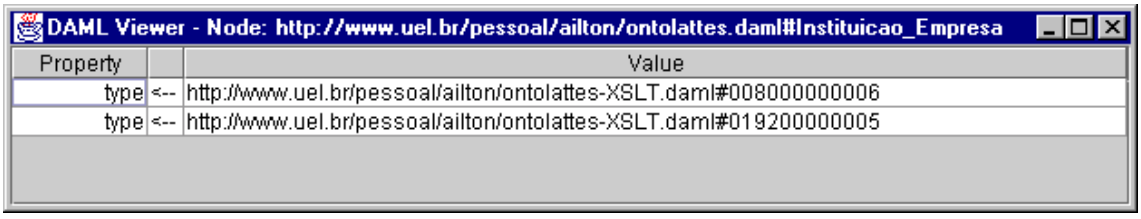

FIGURA 4.24 - DAML Viewer - Nodo Instituicao_Empresa

Da mesma forma, se a intenção é observar o Endereço Profissional de um dos nodos da **FIGURA 4.**23, basta efetuar a navegação através de mais um clique, e assim, sucessivamente. Onde se obtém:

FIGURA 4.25 - DAML Viewer - Nodo End_Profissional

4.4.3 RDQL sobre o OntoLattes

Nesta seção, são apresentadas algumas consultas efetuadas sobre a base do Currículo Lattes, dentro da ontologia OntoLattes. As consultas na OntoLattes são guiadas através da importação da ontologia em DAML+OIL para o ambiente Jena, que serve como um *wrapper* para RDF pela leitura e escrita das *tags* em RDF.

Ressalva-se o fato de que a base ontológica não está completa. Nestes exemplos foram utilizados apenas alguns poucos dados demonstrativos para as efetivas consultas serem executadas. Não é objetivo deste trabalho a avaliação de tempo de recuperação em uma grande base de dados.

Para se obter um completo aproveitamento das extensões da linguagem DAML+OIL, deveria haver uma linguagem de consulta definida especificamente para ela. No entanto, como foi visto na seção 3.2.2, uma linguagem de consulta para modelos DAML+OIL ainda está em processo de desenvolvimento e sob pesquisas.

Entretanto, visto que DAML+OIL é uma extensão do modelo RDFS, pode-se utilizar linguagens de consultas desenvolvidas para o modelo RDFS.

Na ontologia OntoLattes, que utiliza o modelo DAML+OIL, seria necessária uma linguagem de consulta muita mais complexa, mas, mesmo assim, utilizando-se dos recursos da linguagem de consulta RDQL, pode-se obter respostas sob os aspectos do modelo e das instâncias da ontologia.

Observe a linha de comando:

```
java jena.rdfquery
–data http://www.uel.br/pessoal/ailton/ontolattes.daml
--query onto-q1
```

Está sendo invocado o método **rdfquery** do Jena para a execução de uma consulta **onto-q1**, na qual estão os comandos SELECT, WHERE, etc, sobre uma ontologia chamada **ontolattes.daml** na URI especificada.

Todas as linhas de comando que serão executadas deverão conter estes parâmetros: URI ou arquivo local (ontologia) e arquivo local contendo os comandos para a consulta.

4.4.3.1 Visualizando as triplas

Aqui é apresentada uma consulta para gerar todas as triplas da ontologia.

Executando a seguinte consulta sobre as instâncias da OntoLattes:

Consulta *q1*: Para retornar todas as triplas da ontologia.

```
SELECT ?x ,?w,?y
WHERE (?x, ?w, ?y)
```

Tem-se para a URI:

```
<http://www.uel.br/pessoal/ailton/inst-ontolattes.daml
```

A resposta das triplas, onde:

X é o sujeito;
W é o predicado e,
Y é o objeto

TABELA 4.1 - RDQL - Consulta 1

x	w	y
<http://www.uel.br/pessoal/ailton/inst-ontolattes.daml#Dados_Carina>	<http://www.uel.br/pessoal/ailton/ontolattes.daml#nacionalidade>	"B"
<http://www.uel.br/pessoal/ailton/inst-ontolattes.daml#019200000005>	<http://www.uel.br/pessoal/ailton/ontolattes.daml#codigo_instituicao_emp_cur>	"019200000005"
<http://www.uel.br/pessoal/ailton/inst-ontolattes.daml#End_Prof_Cesar>	<http://www.uel.br/pessoal/ailton/ontolattes.daml#trabalhaNaInstituicao>	<http://www.uel.br/pessoal/ailton/019200000005>
<http://www.uel.br/pessoal/ailton/inst-ontolattes.daml#End_Prof_Ailton>	<http://www.uel.br/pessoal/ailton/ontolattes.daml#ddd>	"43"
<http://www.uel.br/pessoal/ailton/inst-ontolattes.daml#End_Prof_Carina>	<http://www.uel.br/pessoal/ailton/ontolattes.daml#bairro>	"Agronomia"
<http://www.uel.br/pessoal/ailton/inst-ontolattes.daml#Dados_Deise>	<http://www.uel.br/pessoal/ailton/ontolattes.daml#sexo>	"FEMININO"
...		

4.4.3.2 Visualizando as instâncias

Ao se olhar para a consulta **q2**, tem-se um padrão (?x, <lattes:sexo>, “FEMININO”) para as triplas no fonte que contém as instâncias da ontologia OntoLattes em DAML+OIL. Este padrão é verificado em cada uma das triplas da ontologia e os resultados recuperados.

As URIs são delimitadas usando o sinal de maior e menor (< >), as variáveis são introduzidas por um sinal de “?” e a constante *string* especificada entre aspas (“ “). Caso a constante fosse um número, poderia ser especificada sem as aspas.

Consulta *q2*: Esta consulta retorna uma variável, que mostra todos os sujeitos que possuem a propriedade **#sexo** igual a “FEMININO”.

```
SELECT ?x
WHERE (?x, <http://www.uel.br/pessoal/ailton/ontolattes.daml#sexo>,
"FEMININO")
```

Tem-se o resultado, onde:

X é o recurso que satisfaz a cláusula WHERE da consulta.

TABELA 4.2 - RDQL - Consulta 2

x
<http://www.uel.br/pessoal/ailton/inst-ontolattes.daml#Dados_Carina>
<http://www.uel.br/pessoal/ailton/inst-ontolattes.daml#Dados_Deise>

No caso da consulta **q3**, o resultado para uma consulta a ontologia retorna duas variáveis. Diferentemente da consulta **q2**, esta consulta não especifica um valor constante como restrição, retornando como resultado, todos os sujeitos das triplas que possuem a propriedade **#sexo** e seu respectivos valores.

Consulta *q3*: Esta consulta retorna duas variáveis. Uma das váriáveis é o recurso e a outra é o sexo.

```
SELECT ?x, ?sexo
WHERE (?x, <http://www.uel.br/pessoal/ailton/ontolattes.daml#sexo>, ?sexo)
```

Tem-se o resultado, onde:

X é o recurso e
sexo é o valor do recurso

TABELA 4.3 - RDQL - Consulta 3

x	**sexo**
<http://www.uel.br/pessoal/ailton/inst-ontolattes.daml#Dados_Ailton>	"MASCULINO"
<http://www.uel.br/pessoal/ailton/inst-ontolattes.daml#Dados_Carina>	"FEMININO"
<http://www.uel.br/pessoal/ailton/inst-ontolattes.daml#Dados_Deise>	"FEMININO"
<http://www.uel.br/pessoal/ailton/inst-ontolattes.daml#Dados_Andre>	"MASCULINO"
<http://www.uel.br/pessoal/ailton/inst-ontolattes.daml#Dados_Cesar>	"MASCULINO"

4.4.3.3 Utilizando vários padrões de triplas

Nas *queries* anteriores, foi utilizado um único padrão para as triplas. No entanto, a cláusula WHERE faz o emparceiramento das descrições em forma de grafo das triplas, dando uma lista de padrões de triplas (arcos).

Consulta *q4*: Nesta próxima consulta, quer-se encontrar um nodo no gráfico das triplas, **?y**, que tem uma propriedade **#sexo** com o valor "**MASCULINO**". E **?y** tem ainda uma segunda propriedade chamada **#nome_completo**, que é o resultado que deve retornar para a variável **?Nome**.

```
SELECT ?Nome
WHERE
(?y, <http://www.uel.br/pessoal/ailton/ontolattes.daml#sexo>,"MASCULINO"),
(?y, <http://www.uel.br/pessoal/ailton/ontolattes.daml#nome_completo>,?Nome)
```

Tem-se o resultado, onde:

Nome é o valor que satisfaz a cláusula WHERE da consulta.

TABELA 4.4 - RDQL - Consulta 4

Nome
"Ailton Sergio Bonifacio"
"Andre Rauber Du Bois"
"Cesar Roberto Mariano Vittori"

Foram encontrados três valores. Na consulta q4, a variável **?y** é a mesma para cada padrão de tripla. Para o sucesso da comparação e busca, o valor da variável deve ser o mesmo para o padrão da tripla, pois significa que estão sendo pesquisados os sujeitos de uma mesma classe ou conceito dentro da ontologia. Neste caso, a classe **Dados_Gerais**. O valor retornado é o bNode (intersecção) da chegagem das informações das triplas da OntoLattes para cada propriedade especificada.

Quando se deseja filtrar o valor de uma propriedade encontrada, pode-se utilizar a cláusula AND na definição da consulta. Esta mesma consulta poderia ser escrita da seguinte forma:

```
SELECT ?Nome
WHERE
(?y, <http://www.uel.br/pessoal/ailton/ontolattes.daml#sexo>,?sexo),
(?y, <http://www.uel.br/pessoal/ailton/ontolattes.daml#nome_completo>,?Nome)
AND   ?sexo eq "MASCULINO"
```

4.4.3.4 Utilizando Paths *para buscas*

Uma estrutura bastante comum é conhecer os *paths* de um grafo de triplas, conhecendo-se o ponto de partida ou através de uma variável. *Paths* são os caminhos que um número de arcos, ligados por um nodo, seguem. Supondo que toda a base de descrições pode ser visualizada como uma coleção de nodos e arcos, as expressões de caminho (*paths*) podem ser utilizadas nos filtros RDQL para a recuperação de nodos em profundidade. Desta forma, RDQL captura a semântica existencial da navegação nos grafos semi-estruturados do modelo RDFS/DAML+OIL.

Consulta *q5*: Para uma dada variável **?z**, que é interna para a consulta, ou seja, serve para a consulta ligar os recursos ao nome, através da composição do *path* das propriedades **#nome_completo**, **#onProperty** e **#subClassOf**:

```
SELECT ?Conceito, ?Instancia, ?Nome
WHERE  (?Conceito, <http://www.w3.org/2000/01/rdf-schema#subClassOf>, ?z),
       (?z, <http://www.daml.org/2001/03/daml+oil#onProperty>,
       <http://www.uel.br/pessoal/ailton/ontolattes.daml#nome_completo>),
       (?instancia,
       <http://www.uel.br/pessoal/ailton/ontolattes.daml#nome_completo>,
       ?Nome)
```

Tem-se o resultado, onde:

Conceito é a classe/subclasse da ontologia,
Instancia é a instancia da classe/subclasse e,
Nome é o valor da propriedade #nome_completo

TABELA 4.5 - RDQL - Consulta 5

Conceito	Instancia	Nome
<http://www.uel.br/pessoal/ailton/ontolattes.daml#Dados_Gerais>	<http://www.uel.br/pessoal/ailton/ontolattes.daml#Dados_Ailton>	"Ailton Sergio Bonifacio"
<http://www.uel.br/pessoal/ailton/ontolattes.daml#Dados_Gerais>	<http://www.uel.br/pessoal/ailton/ontolattes.daml#Dados_Carina>	"Carina Friedrich Dorneles"
<http://www.uel.br/pessoal/ailton/ontolattes.daml#Dados_Gerais>	<http://www.uel.br/pessoal/ailton/ontolattes.daml#Dados_Deise>	"Deise de Brum Saccol"
<http://www.uel.br/pessoal/ailton/ontolattes.daml#Dados_Gerais>	<http://www.uel.br/pessoal/ailton/ontolattes.daml#Dados_Andre>	"Andre Rauber Du Bois"
<http://www.uel.br/pessoal/ailton/ontolattes.daml#Dados_Gerais>	<http://www.uel.br/pessoal/ailton/ontolattes.daml#Dados_Cesar>	"Cesar Roberto Mariano Vittori"

Nesta consulta *q5b*, a variável **?z** é chamada de bNode (nodos anônimos). Esta consulta encontra intencionalmente o bNode para a estrutura da informação sobre o predicado <http://www.w3.org/2000/01/rdf-schema#subClassOf>.

Consulta *q5b*: Para encontrar o bNode do predicado **#subClassOf**.

TABELA 4.6 - RDQL - Consulta 5b

z
<anon:453dce:ef5d44f9c9:-7d6e>

Este resultado, aparentemente estranho, é um identificador interno do Jena para bNodes. bNodes não possuem rótulos (onde o “b” representa “blank”). Isto é apenas uma representação dos detalhes da implementação do Jena. Sendo assim, não é util a apresentação do resultado de um bNode.

4.4.3.5 Utilizando prefixos para as URIs

Como foi observado na seção anterior, as URIs tendem a ser muito longas. Um esquema RDFS, e consequentemente um esquema DAML+OIL, tem uma URI que define um espaço para os identificadores, e cada identificador é um nome concatenado àquela URI.

RDQL tem uma conveniência sintática que permite especificar prefixos para as URIs utilizadas em uma consulta, através do uso da cláusula USING. Portanto, se a consulta q5 for escrita, conforme abaixo, usando-se a cláusula USING, o resultado da consulta seria exatamente o mesmo.

Consulta *q6*: Consulta q5 sendo executada com a cláusula USING:

```
SELECT ?conceito, ?instancia, ?Nome
WHERE  (?conceito, <rdfs:subClassOf>, ?z),
       (?z, <daml:onProperty>, <lattes:nome_completo>),
       (?instancia, <lattes:nome_completo>,?Nome)
USING  rdfs   FOR <http://www.w3.org/2000/01/rdf-schema#>,
       daml   FOR <http://www.daml.org/2001/03/daml+oil#>,
       lattes FOR <http://www.uel.br/pessoal/ailton/ontolattes.daml#>
```

A medida em que mais propriedades aparecem em uma consulta, este mecanismo ajuda a manter uma consulta muito mais legível e compreensível, sem longas URIs que podem comprometer o entendimento da estrutura de um padrão de triplas.

4.4.3.6 Consulta a propriedades

As variáveis podem aparecer no sujeito, no predicado ou no objeto (valor) de um padrão de tripla. Sem qualquer restrição, pode-se efetuar consultas para que retornem as propriedades de determinados valores.

Consulta q7: Esta consulta tem como resultado a descrição da propriedade que possua o valor "BONIFACIO, A. S.", na qual procura demonstrar como o recurso ?y, ou seja, qualquer recurso, se relaciona com o valor dado.

```
SELECT ?Propriedade
WHERE ( ?y , ?Propriedade, "BONIFACIO, A. S.")
USING rdfs   FOR <http://www.w3.org/2000/01/rdf-schema#>,
      daml   FOR <http://www.daml.org/2001/03/daml+oil#>,
      lattes FOR <http://www.uel.br/pessoal/ailton/ontolattes-xslt.daml#>
```

Tem-se o resultado, onde:

Propriedade é o predicado da tripla, ou seja, a propriedade que possui o valor da constante declarada na consulta.

TABELA 4.7 - RDQL - Consulta 7

Propriedade
<http://www.uel.br/pessoal/ailton/ontolattes.daml#nome_em_citacoes_bibliograficas>

Diante do que foi apresentado, RDQL pode recuperar informações sobre as coleções de dados RDFS e consequentemente sobre dados DAML+OIL (por exemplo, classes, propriedades, valores, etc) e pode-se introduzir variáveis para isso. Além disso, RDQL fornece um bom filtro de informações através das cláusulas SELECT-FROM-WHERE.

Em outras palavras, RDQL estende a noção de generalizadas expressões de caminhos **[CER 99]** para completa herança de *paths* de classes (ou propriedades). Isto é totalmente útil visto que os recursos podem se multiplicar e muitas propriedades podem vir de diferentes hierarquias de classes que podem ser usadas para uma mesma descrição de recurso. RDQL permite, ainda, consultar propriedades advindas de recursos que estejam de acordo com uma específica hierarquia de classes, ou de acordo com um específico valor.

5 Conclusões

Este trabalho apresentou, além dos conceitos básicos, uma introdução a novos paradigmas de linguagens e ferramentas que estão dando os primeiros passos em direção a Web Semântica. Dentre elas, linguagens para construção de ontologias e linguagens para consultas; além das ferramentas associadas que objetivam o armazenamento, manutenção e anotação em ontologias. Para atingir este propósito, estas linguagens e ferramentas foram aplicadas a um caso de dimensão e complexidade realistas, o Currículo Lattes. Por isso, foi proposta uma ontologia na linguagem DAML+OIL. Foi apresentada, também, uma descrição dos métodos de instanciação da ontologia proposta e dos métodos e/ou linguagens de consulta mais adequadas para a consulta semântica das informações geradas por esta ontologia.

A maior contribuição deste trabalho foi permitir uma melhor compreensão deste conjunto de conceitos, linguagens e ferramentas apresentadas, através da aplicação ao caso Lattes. Estes estudos serão de grande utilidade para os trabalhos futuros. Além disso, os estudos envolveram pesquisas sobre Ontologias, Metadados, Bibliotecas Digitais, Plataforma Lattes, XML, RDFS e DAML+OIL, entre outros. Outros estudos envolveram a avaliação e aplicação de ferramentas para construção e edição de ontologias (tais como OilEd, OntoEdit, Protégé), ferramentas de instanciação e anotação (ex: OntoMat, XSLT, etc) e linguagens de consultas (*query languages*) para os modelos RDFS (ex: RDQL, TRIPLE, etc) e DAML+OIL (ex:, DQL).

Obviamente, as ferramentas e linguagens apresentadas neste trabalho não esgotam os estudos referentes a Web Semântica. Melhor dizendo, estão apenas começando. Portanto, são apenas indicativos da tendência para se fornecer suporte total ao armazenamento e consultas para os padrões de metadados/ontologias baseados na Web. Muito do que se espera de uma Web Semântica ainda está em processo de pesquisas, estudos e desenvolvimento.

Por se tratar de um modelo bastante extenso e passível de muitas discussões, a ontologia gerada (OntoLattes) deve passar, ainda, por refinamentos posteriores. A definição desta ontologia, neste trabalho, serve apenas como um ponto inicial de estudos para um padrão adicional para o Sistema de Currículos Lattes da comunidade LMPL. Este trabalho, faz pensar na viabilidade da definição concreta de um modelo DAML+OIL para o Sistema de Currículos Lattes.

O plano original deste trabalho visava a importação da DTD do Currículo Lattes diretamente para um editor de ontologias chamado Protégé (um execlente editor de ontologias), através de plugins de importação. O primeiro problema encontrado foi que não havia plugin para o Protégé **[STA 2000]** que satisfatoriamente suprisse esta intenção. Outro fator que fez abandonar a idéia de se utilizar o Protégé, é que o mesmo não exportava o modelo conceitual construído em seu ambiente para a linguagem DAML+OIL. Pelas pesquisas efetuadas e relacionamentos mantidos através da lista de mail do Protégé com os seus pesquisadores, verificou-se que o plugin de exportação de ontologia do Protégé para a linguagem DAML+OIL está em desenvolvimento.

Outra ferramenta para conversão de uma DTD para a linguagem DAML+OIL foi o XMLtoDAML **[NEI 2001]** que, apesar de fazer a tradução, não demonstrou ser flexível ao ponto de se especificar as hierarquias de classes mais adequadas ao contexto. A tradução efetuada via XMLtoDAML era de forma linear. Por exemplo, mantêm-se a hierarquia da DTD indiscriminadamente, sem analisar os atributos que terão como conteúdo, instâncias de uma outra classe.

Atualmente, os processos automáticos de transformação de uma DTD em uma ontologia na linguagem DAML+OIL, tais como, XMLtoDAML **[NEI 2001]** estão

apenas no início. Estes pocessos ainda não são eficazes o suficiente para gerarem uma ontologia dentro dos parâmetros desejáveis. Além disso, a ferramenta XMLtoDAML só faz a tradução do modelo ontológico, ou seja, não efetua a tradução de conteúdo.

Para a instanciação, um processo genérico de tradução seria ideal. Um processo que faça a análise das equivalências entre um documento XML, ou sua DTD e uma ontologia em DAML+OIL é desejável.

Na verdade, não foi encontrada nenhuma ferramenta que consiga ler e interpretar um documento XML (tal como o documento XML que o Sistema de Currículos Lattes exporta) e fazer a sua tradução para um modelo conceitual em DAML+OIL. Como dito anteriormente, a ferramenta XMLtoDAML ainda não faz a tradução de conteúdo XML.

Neste trabalho, um processo de tradução semi-automática dos dados do documento XML gerado pela exportação do Sistema Lattes para o modelo ontológico em DAML+OIL foi apresentado.

Dentre outras coisas, um efetivo modelo de interface de usuário deveria ser projetado, desenvolvido e utilizado. Com ele, seriam providos meios para que se combinem e efetuem as traduções das instâncias do documento XML do Sistema Lattes para sua respectiva ontologia em DAML+OIL, combinando e relacionando padrões e nomenclaturas.

Quanto a linguagens de consultas, verificou-se uma grande área de pesquisa em evolução. Existem muitas linguagens de consultas e motores de inferência para RDF (por exemplo, SiLRI **[DEC 98]**, RQL **[KAR 2001]**, etc). Porém, não há, ainda, uma linguagem de consulta específica para a linguagem DAML+OIL que tenha sido concluída. As soluções existentes nesta direção são linguagens de consulta que foram desenvolvidas para os modelos RDFS e que podem, com algumas restrições, serem estendidas para modelos em DAML+OIL. É o caso de RQL, TRIPLE e RDQL, entre outras. Uma linguagem de consulta que está direcionando seus esforços para os modelos em DAML+OIL é a DQL (DAML+OIL Query Language). No entanto, esta linguagem também se encontra em processo de desenvolvimento.

Outra observação quanto as linguagens de consulta pesquisadas e utilizadas diz respeito a carência de documentação. Tanto TRIPLE (este com uma carência ainda maior, pois só havia um artigo a respeito) quanto RDQL contemplam esta lacuna. Embora o ambiente Jena seja relativamente fácil de se usar, ele necessita de uma melhor documentação e de tutoriais. Algumas características destas linguagens não são fáceis de serem utilizadas e entendidas sem uma documentação mínima.

Com certeza, há muito o que ser discutido, mas este trabalho procurou abordar e aplicar as tendências das pesquisas para se atender as exigências para uma Web Semântica. A aplicação foi sobre o Currículo Lattes. Para isto, foram utilizadas algumas ferramentas e novas linguagens, tal como DAML+OIL, que têm surgido para capacitar a aquisição de conhecimento de fontes dispersas de informações.

DAML+OIL deve ser visto como uma alternativa a mais para a representação do Sistema de Curriculos Lattes. A adição de um modelo conceitual DAML+OIL para currículos, neste caso, o Lattes, pode enriquecer a relação de expressividade para aumentar o já existente padrão XML proporcionado pela comunidade LMPL.

Anexo 1 BNF para RDQL

A tabela abaixo apresenta a BNF para o RDQL.

CompilationUnit	::=	Query <EOF>
Query	::=	SelectClause (SourceClause)? TriplePatternClause (ConstraintClause)? (PrefixesClause)?
SelectClause	::=	(<SELECT> Var ("," Var)* | <SELECT> "*")
SourceClause	::=	(<SOURCE> | <FROM>) SourceSelector
SourceSelector	::=	URL
TriplePatternClause	::=	<WHERE> TriplePattern ("," TriplePattern)*
ConstraintClause	::=	<SUCHTHAT> Expression (("," | <SUCHTHAT>) Expression)*
TriplePattern	::=	<LPAREN> VarOrURI "," VarOrURI "," VarOrLiteral <RPAREN>
VarOrURI	::=	Var
	|	URI
VarOrLiteral	::=	Var
	|	Literal
Var	::=	"?" Identifier
PrefixesClause	::=	<PREFIXES> PrefixDecl ("," PrefixDecl)*
PrefixDecl	::=	Identifier <FOR> URI
Expression	::=	ConditionalOrExpression
ConditionalOrExpression	::=	ConditionalXorExpression (<SC_OR> ConditionalXorExpression)*
ConditionalXorExpression	::=	ConditionalAndExpression
ConditionalAndExpression	::=	ValueLogical (<SC_AND> ValueLogical)*
ValueLogical	::=	StringEqualityExpression
StringEqualityExpression	::=	NumericalLogical (<STR_EQ> NumericalLogical | <STR_NE> NumericalLogical)*
NumericalLogical	::=	InclusiveOrExpression
InclusiveOrExpression	::=	ExclusiveOrExpression (<BIT_OR> ExclusiveOrExpression)*
ExclusiveOrExpression	::=	AndExpression (<BIT_XOR> AndExpression)*
AndExpression	::=	ArithmeticCondition (<BIT_AND> ArithmeticCondition)*
ArithmeticCondition	::=	EqualityExpression
EqualityExpression	::=	RelationalExpression (<EQ> RelationalExpression | <NEQ> RelationalExpression)?
RelationalExpression	::=	NumericExpression (<LT> NumericExpression | <GT> NumericExpression | <LE> NumericExpression | <GE> NumericExpression)?
NumericExpression	::=	ShiftExpression
ShiftExpression	::=	AdditiveExpression (<LSHIFT> AdditiveExpression | <RSIGNEDSHIFT> AdditiveExpression | <RUNSIGNEDSHIFT> AdditiveExpression)*
AdditiveExpression	::=	MultiplicativeExpression (<PLUS> MultiplicativeExpression | <MINUS> MultiplicativeExpression)*
MultiplicativeExpression	::=	UnaryExpression (<STAR> UnaryExpression | <SLASH> UnaryExpression | <REM> UnaryExpression)*
UnaryExpression	::=	UnaryExpressionNotPlusMinus
	|	(<PLUS> UnaryExpression | <MINUS> UnaryExpression)
UnaryExpressionNotPlusMinus	::=	(<TILDE> | <BANG>) UnaryExpression
	|	PrimaryExpression
PrimaryExpression	::=	Var
	|	Literal
	|	FunctionCall
	|	<LPAREN> Expression <RPAREN>
FunctionCall	::=	Identifier <LPAREN> ArgList <RPAREN>
ArgList	::=	VarOrLiteral ("," VarOrLiteral)*
Literal	::=	URI
	|	NumericLiteral
	|	TextLiteral
	|	BooleanLiteral
	|	NullLiteral

```
NumericLiteral ::= ( <INTEGER_LITERAL> | <FLOATING_POINT_LITERAL> )
   TextLiteral ::= <STRING_LITERAL>
BooleanLiteral ::= <BOOLEAN_LITERAL>
   NullLiteral ::= <NULL_LITERAL>
           URL ::= URI
           URI ::= "<" <URI> ">"
    Identifier ::= <IDENTIFIER>
```

Anexo 2 Programa Fonte em XSLT

Este anexo fornece o programa fonte em XSLT utilizado para a transformação das instâncias em XML para DAML+OIL.

```
<?xml version="1.0" encoding = "ISO-8859-1" ?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<xsl:output method="xml" indent="yes" encoding="ISO-8859-1"/>

<xsl:template match="/">
   <xsl:element name="rdf&#58;RDF">
      <xsl:apply-templates select="/CURRICULO-VITAE/DADOS-GERAIS"/>
      <xsl:apply-templates select="/CURRICULO-VITAE/DADOS-
    GERAIS/ENDERECO/ENDERECO-PROFISSIONAL"/>
      <xsl:call-template name="monta-lista-uf">
         <xsl:with-param name="lista-uf"/>
         <xsl:with-param name="contador" select="1"/>
      </xsl:call-template>
   </xsl:element>
</xsl:template>

<xsl:template match="DADOS-GERAIS">
   <xsl:element name="Dados_Gerais">
      <xsl:attribute name="rdf&#58;ID">
         <xsl:value-of select="concat('Dados_',substring-before(@NOME-
    COMPLETO,' '),'_',@NUMERO-IDENTIDADE)"/>
      </xsl:attribute>
      <nome_completo>
         <xsl:value-of select="@NOME-COMPLETO"/>
      </nome_completo>
      <nome_em_citacoes_bibliograficas>
         <xsl:value-of select="@NOME-EM-CITACOES-BIBLIOGRAFICAS"/>
      </nome_em_citacoes_bibliograficas>
      <nacionalidade>
         <xsl:value-of select="@NACIONALIDADE"/>
      </nacionalidade>
       <cpf>
         <xsl:value-of select="@CPF"/>
      </cpf>
       <xsl:element name="uf_nascimento">
         <xsl:attribute name="rdf&#58;resource">
            <xsl:value-of select="@UF-NASCIMENTO"/>
         </xsl:attribute>
      </xsl:element>
      <cidade_nascimento>
         <xsl:value-of select="@CIDADE-NASCIMENTO"/>
      </cidade_nascimento>
      <sexo>
         <xsl:value-of select="@SEXO"/>
      </sexo>
      <numero_identidade>
         <xsl:value-of select="@NUMERO-IDENTIDADE"/>
      </numero_identidade>
      <orgao_emissor>
         <xsl:value-of select="@ORGAO-EMISSOR"/>
      </orgao_emissor>
      <uf_orgao_emissor>
         <xsl:attribute name="rdf&#58;resource">
            <xsl:value-of select="@UF-ORGAO-EMISSOR"/>
```

```
         </xsl:attribute>
      </uf_orgao_emissor>
      <nome_da_mae>
         <xsl:value-of select="@NOME-DA-MAE"/>
      </nome_da_mae>
   </xsl:element>
</xsl:template>

<xsl:template match="ENDERECO-PROFISSIONAL">
   <xsl:element name="End_Profissional">
      <xsl:attribute name="rdf&#58;ID">
         <xsl:value-of select="concat('Dados_',substring-
    before(/CURRICULO-VITAE/DADOS-GERAIS/@NOME-COMPLETO,'
    '),'_',/CURRICULO-VITAE/DADOS-GERAIS/@NUMERO-IDENTIDADE)"/>
      </xsl:attribute>
      <xsl:element name="uf">
         <xsl:attribute name="rdf&#58;resource">
            <xsl:value-of select="@UF"/>
         </xsl:attribute>
      </xsl:element>
      <xsl:element name="trabalhaNaInstituicao">
         <xsl:attribute name="rdf&#58;resource">
            <xsl:value-of select="@CODIGO-INSTITUICAO-EMPRESA"/>
         </xsl:attribute>
      </xsl:element>
      <logradouro>
         <xsl:value-of select="@LOGRADOURO-COMPLEMENTO"/>
      </logradouro>
      <xsl:element name="pais">
         <xsl:attribute name="rdf&#58;resource">
            <xsl:value-of select="@PAIS"/>
         </xsl:attribute>
      </xsl:element>
      <cep>
         <xsl:value-of select="@CEP"/>
      </cep>
      <cidade>
         <xsl:value-of select="@CIDADE"/>
      </cidade>
      <bairro>
         <xsl:value-of select="@BAIRRO"/>
      </bairro>
      <ddd>
         <xsl:value-of select="@DDD"/>
      </ddd>
      <fax>
         <xsl:value-of select="translate(@FAX,' ','')"/>
      </fax>
      <caixa_postal>
         <xsl:value-of select="@CAIXA-POSTAL"/>
      </caixa_postal>
      <home_page>
         <xsl:value-of select="@HOME-PAGE"/>
      </home_page>
      <telefone>
         <xsl:value-of select="translate(@TELEFONE,' ','')"/>
      </telefone>
      <e_mail>
         <xsl:value-of select="@E-MAIL"/>
      </e_mail>
   </xsl:element>
   <xsl:element name="Instituicao_Empresa">
      <xsl:attribute name="rdf&#58;ID">
         <xsl:value-of select="@CODIGO-INSTITUICAO-EMPRESA"/>
      </xsl:attribute>
```

```
      <nome_instituicao_empresa>
         <xsl:value-of select="@NOME-INSTITUICAO-EMPRESA"/>
      </nome_instituicao_empresa>
      <codigo_instituicao_emp_cur>
         <xsl:value-of select="@CODIGO-INSTITUICAO-EMPRESA"/>
      </codigo_instituicao_emp_cur>
   </xsl:element>
</xsl:template>

<xsl:template name="monta-lista-uf">
   <xsl:param name="lista-uf"/>
   <xsl:param name="contador"/>
   <xsl:choose>
      <xsl:when test="$contador=1">
         <xsl:variable name="lista-uf" select="concat(/CURRICULO-
    VITAE/DADOS-GERAIS/@UF-NASCIMENTO,';')"/>
         <xsl:call-template name="monta-lista-uf">
            <xsl:with-param name="lista-uf" select="$lista-uf"/>
            <xsl:with-param name="contador" select="2"/>
         </xsl:call-template>
      </xsl:when>
      <xsl:when test="$contador=2">
         <xsl:variable name="uf-atual" select="/CURRICULO-VITAE/DADOS-
    GERAIS/@UF-ORGAO-EMISSOR"/>
         <xsl:if test="contains($lista-uf,$uf-atual)">
            <xsl:call-template name="monta-lista-uf">
               <xsl:with-param name="lista-uf" select="$lista-uf"/>
               <xsl:with-param name="contador" select="3"/>
            </xsl:call-template>
         </xsl:if>
         <xsl:if test="not(contains($lista-uf,$uf-atual))">
            <xsl:variable name="lista-uf" select="concat($lista-uf,$uf-
    atual,';')"/>
            <xsl:call-template name="monta-lista-uf">
               <xsl:with-param name="lista-uf" select="$lista-uf"/>
               <xsl:with-param name="contador" select="3"/>
            </xsl:call-template>
         </xsl:if>
      </xsl:when>

      <xsl:when test="$contador=3">
         <xsl:variable name="uf-atual" select="/CURRICULO-VITAE/DADOS-
    GERAIS/ENDERECO/ENDERECO-PROFISSIONAL/@UF"/>
         <xsl:if test="contains($lista-uf,$uf-atual)">
            <xsl:call-template name="monta-lista-uf">
               <xsl:with-param name="lista-uf" select="$lista-uf"/>
               <xsl:with-param name="contador" select="4"/>
            </xsl:call-template>
         </xsl:if>
         <xsl:if test="not(contains($lista-uf,$uf-atual))">
            <xsl:variable name="lista-uf" select="concat($lista-uf,$uf-
    atual,';')"/>
            <xsl:call-template name="monta-lista-uf">
               <xsl:with-param name="lista-uf" select="$lista-uf"/>
               <xsl:with-param name="contador" select="4"/>
            </xsl:call-template>
         </xsl:if>
      </xsl:when>
      <xsl:otherwise>
         <xsl:element name="ufs">
            <xsl:value-of select="$lista-uf"/>
         </xsl:element>
         <xsl:call-template name="monta-lista-paises">
            <xsl:with-param name="lista-uf" select="$lista-uf"/>
            <xsl:with-param name="lista-pais"/>
```

```
            <xsl:with-param name="contador" select="1"/>
         </xsl:call-template>
      </xsl:otherwise>
   </xsl:choose>
</xsl:template>

<xsl:template name="monta-lista-paises">
<xsl:param name="lista-uf"/>
<xsl:param name="lista-pais"/>
   <xsl:param name="contador"/>
   <xsl:choose>
      <xsl:when test="$contador=1">
         <xsl:variable name="lista-pais" select="concat(/CURRICULO-
    VITAE/DADOS-GERAIS/@PAIS-DE-NASCIMENTO,';')"/>
         <xsl:call-template name="monta-lista-paises">
            <xsl:with-param name="lista-uf" select="$lista-uf"/>
            <xsl:with-param name="lista-pais" select="$lista-pais"/>
            <xsl:with-param name="contador" select="2"/>
         </xsl:call-template>
      </xsl:when>
      <xsl:when test="$contador=2">
         <xsl:variable name="pais-atual" select="/CURRICULO-VITAE/DADOS-
    GERAIS/ENDERECO/ENDERECO-PROFISSIONAL/@PAIS"/>
         <xsl:if test="contains($lista-pais,$pais-atual)">
            <xsl:call-template name="monta-lista-paises">
               <xsl:with-param name="lista-uf" select="$lista-uf"/>
               <xsl:with-param name="lista-pais" select="$lista-pais"/>
               <xsl:with-param name="contador" select="3"/>
            </xsl:call-template>
         </xsl:if>

         <xsl:if test="not(contains($lista-pais,$pais-atual))">
            <xsl:variable name="lista-pais" select="concat($lista-
    pais,$pais-atual,';')"/>
            <xsl:call-template name="monta-lista-paises">
               <xsl:with-param name="lista-uf" select="$lista-uf"/>
               <xsl:with-param name="lista-pais" select="$lista-pais"/>
               <xsl:with-param name="contador" select="3"/>
            </xsl:call-template>
         </xsl:if>
      </xsl:when>
      <xsl:otherwise>
         <xsl:element name="paises">
            <xsl:value-of select="$lista-pais"/>
         </xsl:element>
         <xsl:call-template name="gera-uf">
            <xsl:with-param name="lista-uf" select="$lista-uf"/>
            <xsl:with-param name="lista-pais" select="$lista-pais"/>
         </xsl:call-template>
      </xsl:otherwise>
   </xsl:choose>
</xsl:template>
<xsl:template name="gera-uf">
   <xsl:param name="lista-uf"/>
   <xsl:param name="lista-pais"/>
   <xsl:choose>
      <xsl:when test="string-length($lista-uf)>0">
         <xsl:variable name="uf-atual" select="substring-before($lista-
    uf,';')"/>
         <xsl:variable name="lista-uf" select="substring-after($lista-
    uf,';')"/>

         <xsl:element name="Uf">
            <xsl:attribute name="rdf&#58;ID">
               <xsl:value-of select="$uf-atual"/>
```

```
            </xsl:attribute>
            <xsl:element name="pais">
               <xsl:attribute name="rdf&#58;resource">
                  <xsl:value-of select="'Brasil'"/>
               </xsl:attribute>
            </xsl:element>
            <nome_uf>
               <xsl:value-of select="$uf-atual"/>
            </nome_uf>
            <sigla_uf>
               <xsl:value-of select="$uf-atual"/>
            </sigla_uf>
            </xsl:element>
         <xsl:call-template name="gera-uf">
            <xsl:with-param name="lista-uf" select="$lista-uf"/>
            <xsl:with-param name="lista-pais" select="$lista-pais"/>
         </xsl:call-template>
      </xsl:when>
      <xsl:when test="string-length($lista-uf)=0">
         <xsl:call-template name="gera-pais">
            <xsl:with-param name="lista-pais" select="$lista-pais"/>
         </xsl:call-template>
      </xsl:when>
   </xsl:choose>
</xsl:template>

<xsl:template name="gera-pais">
   <xsl:param name="lista-pais"/>
   <xsl:if test="string-length($lista-pais)>0">
      <xsl:variable name="pais-atual" select="substring-before($lista-
    pais,';')"/>
      <xsl:variable name="lista-pais" select="substring-after($lista-
    pais,';')"/>
      <xsl:element name="Pais">
         <xsl:attribute name="rdf&#58;ID">
            <xsl:value-of select="$pais-atual"/>
         </xsl:attribute>
         <nome_pais>
            <xsl:value-of select="$pais-atual"/>
         </nome_pais>
         <sigla_pais>
            <xsl:value-of select="$pais-atual"/>
         </sigla_pais>
      </xsl:element>
      <xsl:call-template name="gera-pais">
         <xsl:with-param name="lista-pais" select="$lista-pais"/>
      </xsl:call-template>
   </xsl:if>
</xsl:template>

</xsl:stylesheet>
```

Bibliografia

[AAB 99] AABY, Anthony. **The Logical Foundations of Computer Science and Mathematics.** [August, 1999]. Disponível em: < http://cs.wwc.edu/~aabyan/Logic/>. Acesso em: 15 jul. 2002.

[BAA 91] BAADER, F.; HANSCHKE, P. A schema for integrating concrete domains into concept languages. In: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, IJCAI, 1991, Sidney. **Proceedings...** Sidney, Australia: [s.n.], 1991. p. 452-457.

[BEC 2001] BECHHOFER, Sean; GOBLE, Carole. Towards Annotation using DAML+OIL. In: INTERNATIONAL CONFERENCE ON KNOWLEDGE CAPTURE, K-CAP, 2001, Victoria. **Proceedings...** Victoria B.C, Canada: [s.n.], 2001.

[BEK 98] BECKS, Andreas; SKLORZ, Stefan; TRESP, Christopher. Semantic Struturing and Visual Querying of Document Abstracts in Digital Libraries. In: EUROPEAN CONFERENCE ON DIGITAL LIBRARIES, ECDL, 2., 1998, Heraklion. **Research and advanced technology for digital libraries**: Proceedings. Berlin: Springer-Verlag, 1998. p. 443-458. (Lecture Notes in Computer Science, v. 1513).

[BER 2001] BERNERS-LEE, Tim; HENDLER, James; LASSILA, Ora. **The Semantic Web.** Scientific American. [May, 2001]. Disponível em: <http://www.sciam.com/2001/0501issue/0501berners-lee.html> . Acesso em: 15 maio 2002.

[BER 98] BERNERS-LEE, Tim. **What the Semantic Web can represent**. [September, 1998]. Disponível em: < http://www.w3.org/DesignIssues/RDFnot.html>. Acesso em: 15 maio 2002.

[BER 99] BERNERS-LEE, Tim; FISCHETTI, Mark. **Weaving the Web**. San Francisco:Harper, 1999. p. 177-198.

[BRA 2000] BRADLEY, N. **The XML Companion.** 2nd ed. Harlow, USA: Addison-Wesley Longman, 2000. 435p.

[BRO 2000] BROEKSTRA, Jeen et al. Adding formal semantics to the Web: building on top of RDF Schema. In: EUROPEAN CONFERENCE ON DIGITAL LIBRARIES, ECDL, 4., 2000, Lisbon, Portugal. **Workshop on the Semantic Web**: Proceedings. Berlin: Springer-Verlag, 2000. (Lecture Notes in Computer Science, v. 1923).

[BRV 2001] BRAGANHOLO, Vanessa; HEUSER, Carlos. XML Schema, RDFS e UML: uma comparaçao. In: JORNADAS IBEROAMERICANAS DE INGENIERIA DE REQUISITOS Y AMBIENTES DE SOFTWARE, IDEAS, 4., 2001, Santo Domingo. **Memórias**. Costa Rica:CIT, 2001. p. 78-90.

[CAL 98] CALVANESE, D. et al. Information integration: Conceptual modeling and reasoning support. In: INTERNATIONAL CONFERENCE ON COOPERATIVE INFORMATION SYSTEMS, CoopIS, 3., 1998, New York City. **Proceedings...** Los Alamitos:IEEE Computer Society, 1998. p. 280-291.

[CAR 2001] CARROLL, Jeremy. **ARP: Another RDF Parser.** The Jena RDF/XML Parser. 2001. Disponível em: <http://www.hpl.hp.co.uk/people/jjc/arp/>. Acesso em: 10 jul. 2002.

[CER 99] CERI, Stefano et al. XML-GL: a Graphical Language for Querying and Restructuring XML Documents. In: INTERNATIONAL WORLD WIDE WEB CONFERENCE, 8., 1999, Toronto. **Proceedings...** Toronto, Canada: [s.n.], 1999. p. 151-165.

[DAM 2000] DAML. **Darpa Agent Markup Language Program**. 2000. Disponível em: <http://www.daml.org>. Acesso em: 12 jul. 2002.

[DAM 2001] DAML+OIL Reference. **Reference description of the DAML+OIL (March 2001) ontology markup language.** [March 2001]. Disponível em: <http://www.daml.org/2001/03/reference>. Acesso em: 07 jul. 2002.

[NEI 2001] NEIGHBORS, Mark. **XML To DAML Translator.** 2001. Disponível em: <http://www.davincinetbook.com:8080/daml/xmltodaml/xmltodaml.html>. Acesso em: 20 jul 2002.

[DEC 98] DECKER, Stefan et al. A query and inference service for RDF. In: THE QUERY LANGUAGES WORKSHOP, QL, 1998, Boston. **Proceedings...** [S.l.]:WorldWideWeb Consortium (W3C), 1998. Disponível em: <http://www.ilrt.bris.ac.uk/discovery/rdf-dev/purls/papers/QL98-queryservice/>. Acesso em: 20 jun. 2002.

[DEC 2000] DECKER, Stefan et al. The semantic Web: The roles of xml and rdf. **IEEE Internet Computing**, Los Alamitos, v. 4, n. 5, p. 63-74, Oct. 2000.

[DEN 2001] DENKER, Grit et. al. Accessing Information and Services on the DAML-Enabled Web. In: INTERNATIONAL WORKSHOP ON THE SEMANTIC WEB, SEMWEB, 2., 2001, Hongkong. **Proceedings...** [S.l.:s.n.], 2001. Disponível em: <http://www.csl.sri.com/papers/denkeretal01/>. Acesso em: 10 jul. 2002.

[DON 97] DONINI, F. M. et al. The complexity of concept languages. **Information and Computation**, [S.l.], v.134, n. 1, p. 1-58, Apr. 1997.

[DUB 2000] DUBLIN CORE METADATA INITIATIVE. **About the Dublin Core Metadata Initiative.** [October 18, 2000]. Disponível em: <http://dublincore.org/about/>. Acesso em: 15 out. 2001.

[FEN 2000] FENSEL, D. et al. OIL in a nutshell. In: EUROPEAN WORKSHOP ON KNOWLEDGE ACQUISITION, MODELING, AND MANAGEMENT,

EKAW, 12., 2000, Juan-les-Pins. **Proceedings...** Berlin:Springer-Verlag, 2000. p. 1-16. (Lecture Notes in Artificial Intelligence, v. 1937).

[FIK 2001] FIKES, R.; MCGUINNESS, D. L. **An axiomatic semantics for rdf, rdf schema, and daml+oil.** Stanford University KSL Technical Report KSL-01-01. Disponível em: <http://www.ksl.stanford.edu/people/dlm/daml-semantics/abstract-axiomatic-semantics.html, 2001>. Acesso em: 10 jun. 2002.

[GEN 87] GENESERETH, M. R.; NILSSON, N. J. **Logical Foundation of Artificial Intelligence.** Los Altos, CA:Morgan Kaufmann, 1987.

[GRU 93] GRUBER, T. R. **Towards principles for the design of ontologies used for knowledge sharing**. Stanford University, Knowledge Systems Laboratory Technical Report KSL93-04. Disponível em: <http://gicl.mcs.drexel.edu/people/regli/Classes/KBA/Readings/KSL-93-04.pdf>. Acesso em: 10 jul. 2002.

[GUH 2000] GUHA, R.V. **rdfDB : An RDF Database.** 2000. Disponível em: <http://guha.com/rdfdb/>. Acesso em: 15 jul. 2002.

[HAA 2001] HAARSLEV, V.; MOLLER, R. RACER system description. In: INTERNATIONAL JOINT CONFERENCE ON AUTOMATED REASONING, IJCAR, 2001, Siena. **Proceedings...** Berlin: Springer-Verlag, 2001. p. 701-705.

[HAA 2001a] HAARSLEV, V.; MOLLER, R. High performance reasoning with very large knowledge bases: A practical case study. In: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, IJCAI, 17., 2001, Seattle. **Proceedings...** [S.l.:s.n], 2001. p. 161-166.

[HOR 2000] HORROCKS, I. Benchmark analysis with fact. In: INTERNATIONAL CONFERENCE AUTOMATED REASONING WITH ANALYTIC TABLEAUX AND RELATED METHODS, TABLEAUX, 2000, St Andrews, Scotland, UK. **Proceedings...** Berlin:Springer-Verlag, 2000. p. 62-66. (Lecture Notes in Computer Science, v. 1847).

[HOR 2001] HORROCKS, I.; SATTLER, U.; TOBIES, S. Reasoning with individuals for the description logic SHIQ. In: INTERNATIONAL CONFERENCE ON AUTOMATED DEDUCTION, CADE, 17., 2000, Pittsburgh. **Proceedings...** Berlin:Springer-Verlag, 2000. p. 482–496. (Lecture Notes in Artificial Intelligence, v. 1831).

[HOR 2002] HORROCKS, Ian; TESSARIS, Sergio. **Querying the Semantic Web: a Formal Approach.** Presented at 1st. International Semantic Web Conference (ISWC~2002). [Online]. Disponível em: < http://www.cs.man.ac.uk/~horrocks/Publications/download/2002/iswc2002.pdf>. Acesso em: 20 jul. 2002.

[HOR 2002a] HORROCKS, I. **DAML+OIL: a Reasonable Web Ontology Language**. Presented at EDBT 2002. [Online]. Disponível em:

<http://www.cs.man.ac.uk/~horrocks/Publications/download/2002/edbt02.pdf>. Acesso em: 20 jul. 2002.

[HOR 2002b] HORROCKS, I.; PATEL-SCHNEIDER, P. F.; HARMELEN, F. van. **Reviewing the Design of {DAML+OIL}: An Ontology Language for the Semantic Web.** Presented at 18th Nat. Conf. on Artificial Intelligence (AAAI~2002). [Online]. Disponível em: <http://www.cs.man.ac.uk/~horrocks/Publications/download/2002/AAAI02IHorrocks.pdf>. Acesso em: 20 jul 2002.

[HOR 98] HORROCKS, I. The FaCT system. In: INTERNATIONAL CONFERENCE AUTOMATED REASONING WITH ANALYTIC TABLEAUX AND RELATED METHODS, TABLEAUX, 1998, Oisterwijk. **Proceedings...** Berlin:Springer-Verlag, 1998. p. 307-312. (Lecture Notes in Artificial Intelligence, v. 1397).

[HOR 99] HORROCKS, I.; U. SATTLER; TOBIES, S. Practical reasoning for expressive description logics. In: INTERNATIONAL CONFERENCE ON LOGIC FOR PROGRAMMING AND AUTOMATED REASONING, LPAR, 6., 1999, Tbilisi. **Logic for programming and automated reasoning**: Proceedings. Berlin:Springer-Verlag, 1999. p. 161-180. (Lecture Notes in Artificial Intelligence, v. 1705).

[JEN 2002] HEWLETT-PACKARD. HPL Semantic Web activity. **The jena semantic web toolkit**. Jena 1.5.0 [July 2002]. Disponível em: <http://www.hpl.hp.com/semweb/jena-top.html>. Acesso em: 10 jul. 2002.

[KAR 2001] KARVOUNARAKIS, G. et al. **Querying Community Web Portals**. 2001. Disponível em: <http://www.ics.forth.gr/proj/isst/RDF/RQL/>. Acesso em: 30 jun. 2002.

[KAY 2000] KAY, Michael. **XSLT**: Programmer's Reference. [S.l.]:Wrox Press Inc.: 2000. ISBN: 1-861003-12-9.

[LAT 2001] CNPq. Grupo Stela. **Plataforma Lattes.** Disponível em: <http://lattes.cnpq.br:8888/plataformalattes/>. Acesso em: 10 jul 2002.

[MAD 2000] MÄDCHE, A. et al. **Representation Language-Neutral Modeling of Ontologies**. Presented at German Workshop “Modellierung” 2000. [Online]. Disponível em: <http://www.ontoprise.de/download/ontoedit_paper.pdf>. Acesso em: 15 jul. 2002.

[MEG 2000] MEGGINSON, David. **RDF Filter (1.0alpha)** [Mar. 27, 2000]. Disponível em : <http://www.megginson.com/Software/>. Acesso em: 20 jul. 2002.

[MEL 2000] MELLO, Ronaldo; DORNELES, Carina; KADE, Adrovane; BRAGANHOLO, Vanessa; HEUSER, Carlos. Dados Semi-Estruturados. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, 15., 2000, João Pessoa, PB. **Anais**... João Pessoa: [s.n.], 2000.

[MIL 2002] MILLER, Libby; BRICKLEY, Dan; DODDS, Leigh. **Inkling**: RDF query using SquishQL**.** Harmony project. IMesh Project. [July 26, 2002]. Latest Version (0.70). Disponível em: <http://swordfish.rdfweb.org/rdfquery/>. Acesso em: 15 jul. 2002.

[OSS 2002] OSSENBRUGGEN, J. van; HARDMAN, L.; RUTLEDGE, L. **Hypermedia and the Semantic Web**: A Research agenda. [May 17, 2002]. Disponível em: < http://jodi.ecs.soton.ac.uk/Articles/v03/i01/VanOssenbruggen/>. Acesso em: 10 jul. 2002.

[RAT 2001] RATNAKAR, V.; GIL, Y. **A Comparison of (Semantic) Markup Languages**. USC/Information Sciences Institute. TRELLIS project. 2001. Disponível em: <http://trellis.semanticweb.org/expect/web/semanticweb/comparison.html>. Acesso em: 10 maio 2002.

[SIN 2002] SINTEK, Michael; DECKER, Stefan. **TRIPLE - A Query, Inference, and Transformation Language for the Semantic Web.** Presented at 1st International Semantic Web Conference (ISWC~2002). [Online]. Disponível em: <http://triple.semanticweb.org/>. Acesso em: 21 jul. 2002.

[STA 2000] STANFORD MEDICAL INFORMATICS. **Protégé-2000**: Home Page. [1998-2000]. Stanford University. Disponível em: <http://protege.stanford.edu>. Acesso em: 10 jun. 2002.

[VDO 2001] VDOVJAK, Richard; HOUBEN, Geert-Jan. **RDF Based Architecture for Semantic Integration of Heterogeneous Information Sources**. Presented at International Workshop on Information Integration on the Web - Technologies and Applications (WIIW~2001). [Online]. Disponível em: < http://wwwis.win.tue.nl/~houben/respub/wiiw01.pdf>. Acesso em: 10 jul. 2002.

[VIE 2001] DEAN, Mike; BARBER, Kelly. **DAML Viewer.** 2001. Disponível em: <http://www.daml.org/viewer/>. Acesso em: 18 jul. 2002.

[VOS 2002] VOS, Arnold de. **An RDF query language based on DAML.** Revision 1.0. [February 15, 2002]. Disponível em: <http://www.langdale.com.au/RDF/DAML-Query.html>. Acesso em: 16 jul. 2002.

[W3C 2000] W3C (WORLD WIDE WEB CONSORTIUM). **Extensible Markup Language (XML).** [October 6, 2000]. Disponível em: <http://www.w3.org/TR/REC-xml>. Acesso em: 27 mar. 2001.

[W3C 2000a] W3C (WORLD WIDE WEB CONSORTIUM). **Resource Description Framework (RDF) Schema Specification 1.0**. [March 27, 2000]. Disponível em: <http://www.w3.org/TR/2000/CR-rdf-schema-20000327/>. Acesso em: 14 maio 2002.

[W3C 98] W3C (WORLD WIDE WEB CONSORTIUM). **Namespaces in XML.**[January 19, 1998]. Disponível em: <http://www.w3.org/TR/1998/NOTE-xml-names-0119.html>. Acesso em: 04 mar. 2001.

[W3C 99] W3C (WORLD WIDE WEB CONSORTIUM). **Resource Description Framework (RDF) Model and Syntax Specification.** [February 22, 1999]. Disponível em: <http://www.w3.org/TR/REC-rdf-syntax/>. Acesso em: 23 maio 2001.