CENTRO UNIVERSITÁRIO FEEVALE

DIEGO CIRINO KERN

INFERÊNCIA SOBRE ONTOLOGIAS

Novo Hamburgo, junho de 2007.

DIEGO CIRINO KERN

INFERÊNCIA SOBRE ONTOLOGIAS

Centro Universitário Feevale
Instituto de Ciências Exatas e Tecnológicas
Curso de Ciência da Computação
Trabalho de Conclusão de Curso

Professor Orientador: Rodrigo Rafael Villarreal Goulart

Novo Hamburgo, junho de 2007.

RESUMO

A World Wide Web, ou simplesmente Web, se tornou uma excelente fonte de pesquisa devido ao elevado número de informações que ela disponibiliza. Porém, a maioria desses dados não possui uma estrutura adequada. Frente a este cenário Tim Berners Lee propôs o termo Web Semântica. Essa seria uma extensão da Web atual com o objetivo de estruturar e dar significado a informação nela inserida, desta maneira, aumentar a capacidade de cooperação dos computadores com as pessoas. As ontologias vêm se destacando como forma de estruturar de forma organizada as informações de um determinado domínio de conhecimento, objetivando um entendimento semântico de situações do mundo real. Com este objetivo foram pesquisados e avaliados mecanismos para extração dessas informações e na verificação da estrutura de uma determinada base ontológica. Após as pesquisas e experimentos realizados, foi desenvolvido um quiz que possibilita a formulação de questionamentos sobre todo o conhecimento armazenado pela ontologia.

Palavras-chave: Ontologia, OWL, Lógica Descritiva e Inteligência Artificial.

ABSTRACT

The World Wide Web, or simply the Web, has become an excellent source for research, due to the high amount of information that is available through it. However, most of the data does not have a suitable structure. Facing this scenario, Tim Berners Lee proposed the term Semantic Web. It would be an extension of the present Web, with the aim of structuring and giving significance to the information it comprises, increasing, the capacity of cooperation among computers and people. Ontologies have proved to be an aoutsanding means of structuring information of a certain domain of knowlodge in an organized manner, aiming at a semantic understanding of the real world. The use of tools that are able to infer on these ontologies are necessary, extracting knowledge. With this objective they had been searched and evaluated mechanisms for extration of these information and in the verification of the structure of one determined ontology. After the carried through research and experiments, were developed a quiz that all makes possible the formularization of questionings on the knowledge stored for the ontology.

Key words: Ontology, OWL, Description Logic and Artifical Intelligence.

LISTA DE FIGURAS

Figura 2.1 - Exemplo de uma ontologia de vinhos (simplificada, sem axiomas e relações).... 24
Figura 2.2 - Tipos de Ontologias e suas relações 25
Figura 2.3 - Exemplo de uma ontologia no Protégé-2000..... 29
Figura 2.4 - Ambiente para construção de ontologias 30
Figura 3.1 - Estrutura de um sistema em Lógica Descritiva 33
Figura 3.2 - Fatos que formam os fundamentos básicos do método Tableaux 38
Figura 4.1 - Exemplo da declaração das namespaces em uma ontologia. 41
Figura 4.2 - Exemplo de headers em uma ontologia. 41
Figura 4.3 - Declaração de classes e subclasse em documento OWL..... 42
Figura 4.4 - Exemplo de um atributo Datatype Property 43
Figura 4.5 - Exemplo de um atributo Object Property 44
Figura 4.6 - Exemplo da utilização do elemento *minCardinality*..... 45
Figura 4.7 - Exemplo da utilização do elemento *hasValue* 46
Figura 4.8 Exemplo da utilização do elemento *equivalentClass*..... 47
Figura 4.9 Exemplo da utilização do elemento *equivalentProperty* 47
Figura 4.10 Exemplo da utilização do elemento *sameAs* 48
Figura 4.11 Exemplo da utilização do elemento *differentFrom*..... 48
Figura 4.12 Exemplo da utilização do elemento *allDifferent*..... 48
Figura 4.13 Exemplo da utilização do elemento *intersectionOf* 49
Figura 4.14 Exemplo da utilização do elemento *unionOf* 49
Figura 4.15 Exemplo da utilização do elemento *oneOf*..... 50
Figura 4.16 Exemplo da utilização do elemento *disjointWith*..... 50
Figura 5.1 - Pergunta do Quiz Ontomúsica com uma dica, após escolha da alternativa errada. 56
Figura 6.1 – Criação do modelo ontológico 58

Figura 6.2 – Criação do modelo ontológico com URI do *profile* 58
Figura 6.3 – Criação do modelo ontológico utilizando especificações 59
Figura 6.4 – Criação completa do modelo ontológico 60
Figura 6.5 – Leitura da ontologia 60
Figura 6.6 – Escrita do modelo ontológico 60
Figura 6.7 – Listagem de classes e propriedades 60
Figura 6.8 – Listagem dos indivíduos do modelo ontológico 61
Figura 6.9 – Listagem dos indivíduos de uma determinada classe do modelo ontológico 61
Figura 6.10 - Arquitetura geral da máquina de inferência do jena 62
Figura 6.11 – Visão da arquitetura do Jena, adaptado de (WILKINSON et al., 2003) 65
Figura 6.12– Criação do modelo para a ontologia ontomúsica. 66
Figura 6.13 – Listagem das classes da ontomusica. 66
Figura 6.14 – Listagens de classes e subclasses da ontomusica. 67
Figura 6.15 – Listagem de propriedades da ontomusica 68
Figura 6.16 – Listagem dos indivíduos de uma determinada classe da ontomusica 70
Figura 6.17 – Central de Estudos: listagem das classes da ontomúsica 73
Figura 6.18 – Central de Estudos: listagem das instâncias da classe Obra 73
Figura 6.19 – Central de Estudos: listagem das propriedades da instância, formando as triplas Instancia-Predicado-Objeto. 74
Figura 6.20 – Adicionando reasoner Pellet ao Jena 76
Figura 6.21 – Verificação da satisfabilidade de conceitos 77
Figura 6.22 – Verificação da equivalência entre conceitos 77
Figura 6.23 – Resultado do serviço de classificação sobre a ontomusica 79
Figura 6.24 – Verificação da realização da ontologia 79
Figura 6.25 – Resultado do serviço de realização da ontomusica 80
Figura 6.26 – Verificação da consistência da ontologia 81
Figura 6.27 - Declaração da propriedade 'nome' para a ontomusica 82
Figura 6.28 – Geração de inconsistência para uma propriedade funcional através do Pellet 82
Figura 6.29 – Declaração da classe ‘SomDeterminado’ para a ontomusica 82
Figura 6.30 – Geração de inconsistência para a propriedade *disjointWith* através do Pellet 82
Figura 7.1 – SPARQL: cláusula *Prefix* 84
Figura 7.2 – SPARQL: cláusula *Select* 84
Figura 7.3 – SPARQL: cláusula *From* 84
Figura 7.4 – SPARQL: cláusula *Where* 84

Figura 7.5 – SPARQL: cláusula *Filter* 84
Figura 7.6 – SPARQL: cláusula *Order By* 85
Figura 7.7 – SPARQL: cláusula *Limit* 85
Figura 7.8 – Exemplo de uma consulta SPARQL 85
Figura 7.9 – Execução de uma *query* SPARQL com a utilização do Jena 86
Figura 7.10 – Resultado de uma query SPARQL sobre a ontomusica 86
Figura 8.1 – OntoQuiz, área de exibição de dados 90
Figura 8.2 – OntoQuiz, classe sendo arrastada pelo usuário 91
Figura 8.3 – OntoQuiz, exemplo de pergunta montada pelo usuário 91
Figura 8.4 – OntoQuiz, alternativas para uma determinada pergunta 92
Figura 8.5 – OntoQuiz, exemplo de query para listar respostas corretas 93
Figura 8.6 – OntoQuiz, exemplo de *query* para uma *object property* 93
Figura 8.7 – OntoQuiz, exemplo de *query* para uma *datatype property* 93
Figura 8.8 – OntoQuiz, dica para uma *object property* 95
Figura 8.9 – OntoQuiz, dica para uma *datatype property* 96
Figura 8.10 – OntoQuiz , algoritmo para geração de dicas 97
Figura 8.11 – OntoQuiz, tela com ontologia sobre professores 98

LISTA DE TABELAS

Tabela 3.1 - Gramática da linguagem descritiva AL 34
Tabela 4.1 – Datatypes recomendados pela W3C para utilização em OWL 43
Tabela 6.1 - Constantes para os *profiles* das linguagens no Jena. 58
Tabela 6.2 – Métodos para listagem das propriedades de uma ontologia 69

LISTA DE ABREVIATURAS E SIGLAS

AC	Aquisição do conhecimento
API	Application Programming Interface
DAML	DARPA Agent MarkupLanguage
DL	Description Logic
EC	Engenharia do Conhecimento
FaCT	Fast Classification of Terminologies
GRR	Generic Rule Reasoner
HTML	HyperText Markup Language
IA	Inteligência Artificial
OIL	Ontology Inference Layer
OWL	Web Ontology Language
RC	Representação do Conhecimento
RDF	Resource Description Framework
SHOE	Simple HTML Ontology Extensions
URI	Unique Resource Identifier
XLS	EXtensible Stylesheet Language
XML	Extensible Markup Language
XOL	Ontology Exchange Language
W3C	World Wide Web Consortium
SPARQL	Sparql Protocol and RDF Query Language

SUMÁRIO

INTRODUÇÃO 12

1 ENGENHARIA DO CONHECIMENTO 15
1.1 Representação e aquisição do conhecimento 16
1.2 Validação do conhecimento 17
1.3 Inferência 17
1.4 Representação de conhecimento e ontologias 17
1.5 Metodologias para representação do conhecimento 18
1.5.1 Metodologia CommonKADS 18
1.5.2 Metodologia Protégé II 19

2 ONTOLOGIAS 21
2.1 Definição 21
2.2 Componentes de uma ontologia 23
2.3 Tipos de ontologias 24
2.4 Vantagens do uso de ontologias 25
2.5 Desvantagens do uso de ontologias 26
2.6 Construção de ontologias 26
2.7 Ferramentas para a construção de ontologias 28
2.8 Linguagens para construção de ontologias 30

3 LÓGICA DE DESCRIÇÃO 32
3.1 Estrutura de uma lógica descritiva 32
3.2 Inferência 34
3.2.1 Inferência em TBox 34
3.2.2 Inferência em ABox 36
3.2.3 Método Tableaux 37

4 WEB ONTOLOGY LANGUAGE 39
4.1 Estrutura das ontologias 41
4.2 Elementos básicos 42
4.2.1 Classes, subclasses e indivíduos 42
4.2.2 Propriedades e relacionamentos 43
4.3 Restrições em propriedades 44
4.3.1 Quantificadores 44
4.3.2 Cardinalidade 45
4.3.3 Informação do valor 46
4.4 Igualdade 46
4.4.1 Equivalência entre classes 46

4.4.2 Equivalência entre propriedades 47
4.4.3 Equivalência entre instâncias 47
4.5 Classes complexas 49
4.5.1 Conjunto de operadores booleanos 49
4.5.2 Classes enumeradas 50
4.5.3 Classes disjuntas 50
5 ONTOMÚSICA 52
5.1 Estrutura das classes 53
5.2 Definição dos atributos das classes 53
5.3 Estrutura dos atributos e base de conhecimento da ontomúsica 54
5.4 Quiz ontomúsica 55
6 SISTEMAS DE INFERÊNCIA 57
6.1 Jena 57
6.1.1 Estrutura geral 57
6.1.2 Motores de inferência 62
6.1.3 Generic Rule Reasoner 63
6.1.4 Jena e mecanismos de inferência adicionais 64
6.1.5 Experimentos com a utilização do jena e a ontomúsica 65
6.2 FaCT ++ (Fast Classification of Terminologies) 74
6.3 Pellet 75
6.3.1 Experimentos com o sistema de inferência Pellet 76
7 SPARQL PROTOCOL AND RDF QUERY LANGUAGE 83
7.1 Cláusulas principais da linguagem 83
7.2 Utilizando SPARQL com o Jena 85
8 ONTOQUIZ 87
8.1 Definição 87
8.2 Fluxo de funcionamento 88
8.3 Interface com o usuário e lógica das funcionalidades 89
8.3.1 Área de exibição dos dados da ontologia 89
8.3.2 Área de escolha de informações que forma a pergunta 91
8.3.3 Área de exibição das alternativas 92
8.3.4 Área de exibição de dicas 94
8.4 Utilização do Quiz com uma ontologia sobre professores 98
CONCLUSÃO 99
REFERÊNCIAS BIBLIOGRÁFICAS 101

INTRODUÇÃO

Há alguns anos ocorre uma notável expansão na quantidade de fontes de informação disponibilizadas através de meios digitais em grandes repositórios, assim como a web. Desta maneira, novas formas de representação e recuperação de informação são cada vez mais necessárias, fundamentando pesquisas em diversas áreas da ciência, como a ciência da computação, a ciência da informação e a lingüística.

Atualmente o termo ontologia vem se destacando no que tange as ciências anteriormente citadas. A sua utilização visa estruturar de forma organizada as informações de um determinado domínio de conhecimento e refletir um entendimento semântico de situações do mundo real.

Numa visão filosófica, o termo ontologia foi proposto por Aristóteles, onde tratava da natureza e da organização do ser. Desde a década de 90 esse termo também vem sendo adotado pela Inteligência Artificial (IA). Nessa área uma das principais definições sobre o tema, diz que uma ontologia é a especificação formal, explícita e compartilhada de uma conceitualização (GRUBER, 1993). Essa definição quer dizer que a ontologia deve ser entendida por uma máquina, onde os tipos de conceitos usados e suas restrições estejam explicitamente definidos e que o fenômeno do mundo que se esteja representado seja entendido por um grupo de indivíduos, ou seja, de modo consensual (STUDER et al., apud ARAUJO, 2003).

No primeiro e segundo capítulos deste trabalho são apresentados os conceitos básicos que definem a engenharia do conhecimento e as ontologias. Ainda sobre ontologias são apresentados os componentes básicos para essa estrutura de representação do conhecimento, seus tipos, os quais são classificados de acordo com o nível de generalidade da ontologia, também são descritas algumas das vantagens e desvantagens da sua utilização na ciência da

computação, assim como conceitos referentes à sua construção, tais como as ferramentas e as linguagens utilizadas.

Também são descritas informações sobre as metodologias para a representação do conhecimento, citando as metodologias CommonKADS e Protege-II.

A proposta deste trabalho é desenvolver um sistema para extração de informações armazenadas em bases ontológicas, utilizando uma ontologia sobre a história da música como estudo de caso. Essa ontologia foi estruturada sobre a linguagem OWL, a qual é baseada em lógica de descrição. Dessa maneira, no capítulo 3, são apresentados conceitos sobre as lógicas descritivas. As lógicas dessa família são utilizadas para a especificação de classes de dados e de relacionamentos entre estas, possuindo uma semântica formal, baseada em lógica, e mecanismos de inferência utilizados para extrair conhecimento que não esteja explicitado na base de conhecimento do domínio (NARDI et al., 2002).

A W3C (World Wide Web Consortium)[1], que é uma organização que trabalha para desenvolver padrões para a criação e interpretação para os conteúdos da web, recomenda que as pessoas utilizem a linguagem OWL para estruturar suas ontologias, esperando-se que esta linguagem torne-se um padrão. Desta forma, no capítulo 4 são tratadas com maiores detalhes as estruturas dessa linguagem, descrevendo a sua estrutura e seus principais elementos.

Logo em seguida, no capítulo 5, é feita uma análise sobre a ontologia referente à história da música, a Ontomusica (BOFF, 2005). Nesse capítulo são exibidos a estrutura da ontologia, as suas classes e os seus atributos. Também é descrito o Quiz Ontomúsica, o qual disponibiliza o conhecimento adquirido na forma de questionamentos e dicas baseadas na ontologia de música.

Logo em seguida são descritas as ferramentas de inferência FaCT++ e Pellet, as quais são baseadas em lógica de descrição. São apresentados dados sobre o funcionamento geral dessas ferramentas.

[1] www.w3.org

No capítulo de número 7, são apresentadas as principais cláusulas para a linguagem de *query* Sparql. Essa linguagem é projetada pela W3C com o objetivo de realizar consultas sobre bases RDF.

No último capítulo é apresentado o OntoQuiz, sistema desenvolvido para extrair o conhecimento armazenado em uma estrutura ontológica baseada na linguagem OWL. O sistema disponibiliza, em forma de questionamentos e de dicas, todo o conhecimento de uma determinada ontologia. Sobre o sistema, são apresentados o seu fluxo de funcionamento e os detalhes de implementação das suas funcionalidades. Por fim, são apresentadas as considerações finais deste trabalho.

1 ENGENHARIA DO CONHECIMENTO

As aplicações de Inteligência Artificial cada vez mais são utilizadas para dar suporte às atividades desenvolvidas no mundo real.

Diversas organizações usam a IA para produzir sistemas que apresentem solução para os problemas de forma inteligente, ou que organizem a informação de modo inteligente para facilitar o entendimento pelo tomador de decisão (BARRETO, 2002).

A Engenharia do Conhecimento (EC) tem sido uma parte fundamental da IA. Ela lida com aquisição e representação de conhecimento e validação, inferência, explicação e manutenção de bases de conhecimento (TURBAN, 1992, apud ZIULKOSKI, 2003).

Outra proposta que se apresenta é a de JARUFE que define a EC como uma associação entre teorias e técnicas que visam estruturar sistemas a base de conhecimentos susceptíveis de prolongar as capacidades humanas de percepção, aprendizagem, compreensão, resolução de problemas e execução de ações (JARUFE, 1999).

As principais idéias que atualmente envolvem a EC são apresentadas por Peter Drucker (1999):

- O conhecimento não é um bem estático a ser minerado, só existe quando se produz algo com ele ou a partir dele;
- O conhecimento não é individual, mas organizacional ou institucional;
- O conhecimento não é genérico, mas associado ou produzido pela solução de uma classe particular de problema ao qual está associado;

- Conhecimento não é um conjunto de regras de solução, mas uma experiência que foi sistematizada e pode ser transmitida.

De maneira geral pode-se afirmar que o objetivo geral da Engenharia de Conhecimento aproxima-se ao da Engenharia de Software: transformar o processo *ad hoc* de construir sistemas baseados em conhecimento em uma disciplina da Engenharia baseada em métodos, linguagens e ferramentas especializadas (Studer, Benjamins e Fensel, 1998, apud ABEL, 2001).

1.1 Representação e aquisição do conhecimento

A Representação do Conhecimento (RC) é, assim como o raciocínio automatizado, uma das questões centrais da Inteligência Artificial (VALENTE, 1995). Ela utiliza métodos para modelar com eficiência os conhecimentos de especialistas em uma determinada área, formando assim uma representação que possa compor um modelo de domínio e a codificação da informação adquirida dessa forma (TURBAN, 1992, apud ABEL, 1998). Esse conhecimento pode ser disponibilizado para que seja acessado pelos usuários de um sistema inteligente.

Segundo Rezende (2003), uma RC deve apresentar as seguintes características:

- Ser compreensível ao ser humano, pois, caso seja necessário avaliar o estado de conhecimento do sistema, a RC deve permitir a sua interpretação;

- Abstrair-se dos detalhes de como funciona internamente o processador de conhecimento que a interpretará;

- Ser robusta, isto é, permitir sua utilização mesmo que não aborde todas as situações possíveis;

- Ser generalizável, ao contrário do conhecimento em si que é individual. Uma representação necessita de vários pontos de vista do mesmo conhecimento, de modo que possa ser atribuída a diversas situações e interpretações.

A Aquisição do Conhecimento (AC) é o processo pelo qual acontece a extração do conhecimento. Essa extração pode ser de um ou mais especialistas ou de diversos outros meios, tais como livros e documentos. Esse processo de extração, geralmente é executado

pelo engenheiro de conhecimento, o qual é responsável por fazer a filtragem de todo a informação necessária. Nesse contexto insere-se uma série de metodologias para aquisição, modelagem e representação do conhecimento. Das propostas de metodológicas atuais, destacam-se a CommonKADS e a PROTÉGÉ, que serão tratadas com maiores detalhes ainda nesse capítulo.

1.2 Validação do conhecimento

A validação do conhecimento trata da consistência do conhecimento que é relevante para a formação da base que está sendo gerada.

Esse conhecimento deve ser submetido ao engenheiro do conhecimento para aprovação e, dependendo do resultado e do contexto no qual deve ser aplicado, será aceito ou não (JÚNIOR, 2003).

1.3 Inferência

Uma inferência descreve os passos de raciocínio primitivo no processo de solução de problemas. Uma inferência recebe uma parcela do conhecimento do domínio ou do problema como entrada e gera uma saída, a qual é uma transformação desse conhecimento. (SCHREIBER *et al.*, 2000, apud JÚNIOR, 2003).

No capítulo 3 será descrita com maiores detalhes a parte de inferência sobre lógicas de descrição.

1.4 Representação de conhecimento e ontologias

Como foi visto anteriormente, a representação de conhecimento é uma das questões centrais da Inteligência Artificial. Segundo Davis et al (1993) ela é um conjunto de comprometimentos epistemológicos, ou mais especificamente, comprometimentos ontológicos.

Seguindo a idéia do mesmo autor acima citado, muitas teorias da Inteligência Artificial que representam domínios específicos como Medicina, Física, ou conceitos básicos (tempo, ação) são atualmente ontologias ou representam de forma indireta o modelo

ontológico. As ontologias tratam as questões em nível de conhecimento e não em nível simbólico, o que é um dos pontos para dar atenção a essa forma de estruturar o conhecimento.

Uma das principais vantagens da modelagem no nível do conhecimento em relação ao nível simbólico é a possibilidade de reutilizar certos componentes do conhecimento em aplicações distintas. Para cada tipo de conhecimento é possível identificar uma série de descrições genéricas cujo comportamento não muda ainda que sejam transferidos a diferentes domínios (PENIN, 2000 apud AMORIM, 2002).

Os principais conceitos referentes ao termo ontologia serão tratados com maiores detalhes no capítulo seguinte.

1.5 Metodologias para representação do conhecimento

Segundo Rubenstein-Montano *et al.* (2001, apud JÚNIOR, 2003), uma metodologia consiste em uma descrição detalhada de passos para o desenvolvimento de um sistema dentro de um contexto de uma arquitetura específica.

Segundo Abel (2001), Allen Newell foi responsável por apresentar noções sobre a engenharia do conhecimento que viabilizaram uma evolução na área. Ele introduziu a noção de nível do conhecimento. Desta forma, a engenharia do conhecimento passou a modelar o conhecimento de forma independente de aspectos de implementação, permitindo identificar, representar e modelar explicitamente diferentes tipos de conhecimento. A ênfase no conhecimento, em vez da representação e implementação, constitui a base da idéia do nível de conhecimento.

A evolução das noções apresentadas por Newell levaram ao surgimento de uma série de metodologias de aquisição e representação de conhecimento que tornaram-se tecnologias de sucesso (ABEL, 2001).

Entre as metodologias mais representativas estão: CommonKADS e Protégé. Uma breve descrição sobre elas será feita a seguir.

1.5.1 Metodologia CommonKADS

A CommonKADS é uma metodologia de integração de metodologias orientadas a modelo, que abrange os diversos aspectos de um projeto de desenvolvimento de um sistema de conhecimento, incluindo: análise organizacional; gerenciamento de projetos; aquisição, representação e modelagem do conhecimento; integração e implementação de sistemas (OLSSON, 2003).

O objetivo básico da metodologia KADS é dar suporte ao desenvolvimento de sistemas baseados em conhecimento em todas as suas fases e aspectos. (SCHREIBER et al., 1999, apud ABEL, 2001). Isso quer dizer que ao invés da metodologia se preocupar unicamente com a aquisição e representação do conhecimento, ela também considera os aspectos organizacionais que definem onde esse conhecimento se insere e de que maneira ele é utilizado pelos usuários ou clientes.

Essa metodologia parte de um conjunto de princípios que norteiam a aplicação da metodologia, dentre os quais, os principais são (SCHREIBER, 1992, apud ABEL, 2001):

- Princípio da Modelagem: o desenvolvimento de um sistema baseado em conhecimento é visto como a construção de um conjunto de modelos de comportamento para a solução de problemas em uma organização. Um sistema baseado em conhecimento é a realização computacional associada a esses modelos.

- Princípio da Limitação de Papéis: um agente inteligente pode ser modelado por atribuir-lhe um conjunto de estruturas de conhecimento e os papéis que essas estruturas devem desempenhar no processo de solução de problemas.

- Princípio da Tipagem do Conhecimento: um modelo, no nível do conhecimento, pode ser visto como consistindo de três diferentes categorias de conhecimento (ou *tipos*): conhecimento do domínio, conhecimento da tarefa e conhecimento de inferência. Além desses, há também o conhecimento de solução de problemas, que compõe não uma quarta categoria, mas uma especificação de como as categorias acima são aplicadas para resolver problemas.

- Princípio da Interação Relativa: prevê diferentes níveis de interação entre as três categorias do conhecimento, que varia em função da aplicação modelada.

1.5.2 Metodologia Protégé II

A metodologia protégé-II está fundamentada em uma estrutura de decomposição de tarefa-métodos. Uma determinada tarefa é decomposta em sub-tarefas através da aplicação de métodos. Esse processo é feito até que se atinja um nível em que métodos primitivos são utilizados para resolver as sub-tarefas. As entradas e saídas de um método são especificadas em uma ontologia de método (CHANDRASEKARAN et al, 1999, apud SILVA, 2001). A estrutura disponibilizada por essa ontologia fornece definições dos conceitos e relações utilizados na especificação de um processo de raciocínio destinado a solucionar uma tarefa.

Grosso et al.(1999, apud SILVA, 2001) relata que o Protégé-II também se destaca por gerar ferramentas de aquisição de conhecimento personalizadas a partir de ontologias. O Protégé faz a especificação dos conceitos relacionados a uma ontologia, e sobre esses conceitos gera, de forma automática, uma ferramenta de aquisição de conhecimento, desta maneira, permite que especialistas de domínios de problemas acrescentem instâncias de conceitos modelados.

No próximo capítulo o termo ontologia é explorado em suas particularidades para fundamentar os métodos propostos neste capítulo.

2 ONTOLOGIAS

Devido à falta de estrutura adequada para organizar as fontes de informação disponibilizadas através de meios digitais, o surgimento de novas formas de representação e recuperação de informação é cada vez mais necessário.

Nos últimos tempos a palavra *ontologia* vem ganhando popularidade devido à possibilidade de sua aplicação em áreas como a busca inteligente na Web, gerência, compartilhamento, reuso de conhecimento e na elaboração de sistemas educacionais inteligentes. Através de uma estrutura organizada, as ontologias representam computacionalmente um determinado domínio de conhecimento, com isso, refletem um entendimento semântico de situações do mundo real, automatizando a comunicação entre pessoas e computadores.

2.1 Definição

Historicamente o termo ontologia tem origem no grego "*ontos",* ser, e "*logos*", palavra (ALMEIDA, BAX, 2003). Numa visão filosófica, o termo ontologia foi proposto por Aristóteles, onde tratava da natureza e da organização do ser, propriedades eram utilizadas para diferenciar espécies do mesmo gênero. Desde o início da década de 90 esse termo também vem sendo adotado pela área da Inteligência Artificial, e para ele são dadas diversas definições.

Para a Ciência da Computação o termo pode ser definido como:

> "Uma ontologia é um artefato de engenharia, constituído de um vocabulário de termos organizados em uma taxonomia, suas definições e um conjunto de axiomas formais usados para criar novas relações e para restringir as suas interpretações segundo um sentido pretendido" (NOY, HAFNER, apud GUIZZARD, 2000, p.38).

Outra definição diz que uma ontologia é um “catálogo de tipos de coisas” em que se supõe existir um domínio, na perspectiva de uma pessoa que usa uma determinada linguagem (SOWA, 1999, apud ALMEIDA, BAX, 2003).

Uma das principais conceitualizações sobre o tema diz que uma ontologia “é a especificação formal, explícita e compartilhada de uma conceitualização” (GRUBER, 1993). Com a definição de Gruber, fica claro que uma das principais utilidades da ontologia é o compartilhamento de informações e a possibilidade de reutilização das informações que um determinado domínio especifica.

Nessa definição, a palavra “formal” refere-se ao fato de que a ontologia deve ser compreendida por uma máquina. A palavra “explícita” significa que os tipos de conceitos usados, as restrições do uso desses conceitos e as suas relações devem ser explicitamente definidas. A palavra “conceitualização” refere-se a um modelo abstrato de algum fenômeno do mundo que se deseja representar, para identificar conceitos relevantes ao fenômeno em questão. Por fim, “compartilhada” reflete a noção de que a ontologia captura um conhecimento consensual, o que significa que esse conhecimento não deve ser restrito a alguns indivíduos, mas aceito por um grupo de pessoas (STUDER et al., apud ARAUJO, 2003).

Para Chandrasekaram (1999, apud ARAUJO 2003), uma ontologia é a representação de um vocabulário sobre um determinado domínio. Onde a qualificação da ontologia não é dada pelo vocabulário, mas sim pelos conceitos expostos por ele. Para o mesmo autor, o termo ontologia ainda teria um outro significado: uma ontologia é usada para referir-se a um conjunto de conhecimentos que, através de um vocabulário representativo, objetiva descrever um determinado domínio. Ou seja, a representação do vocabulário se dá por um conjunto de termos que descrevem os fatos de um domínio específico, e o conjunto de conhecimento utiliza o vocabulário como uma coleção de fatos a respeito do domínio.

Não há um consenso sobre a definição do termo ontologia. Há várias definições na literatura, algumas complementam a definição de Gruber outras sugerem novas, mas de forma geral, ontologias constituem uma ferramenta poderosa para suportar a especificação e a implementação de sistemas computacionais de qualquer complexidade (FONSECA, 1999).

2.2 Componentes de uma ontologia

As ontologias são desenvolvidas em distintas comunidades, sendo utilizadas para conceitualizar diferentes domínios de interesse. Desta forma, as ontologias apresentam estruturas que as distinguem umas das outras, porém existem componentes básicos que constituem a maioria das ontologias para representação de conhecimento (GRUBER, 1993; NOY & GUINNESS, 2001):

- *Classes*: as classes são coleções de elementos formados por um conjunto de atributos iguais. Formam a unidade básica de uma ontologia e formam conceitos que definem um determinado objeto. A ligação entre esses conceitos é dada através de relacionamentos.

- *Relações*: são responsáveis pelos relacionamentos semânticos entre os conceitos de um dado domínio, ou seja, representam um tipo de interação entre as classes do domínio. Elas irão formar a taxonomia do domínio;

- *Axiomas*: são as regras declaradas sobre as relações, essas regras devem ser cumpridas pelos elementos da ontologia, ou seja, são sempre verdadeiras. Elas possibilitam inferir conhecimento que não estão indicados nas taxonomias da ontologia.

- *Instâncias*: representam um determinado objeto de um conceito, ou seja, são os próprios dados da ontologia.

A figura 2.1 mostra uma ontologia de vinhos, onde uma classe de vinhos representa todos os vinhos. Exemplo de uma instância é uma garrafa de vinho *Bordeaux*, que é uma instância da classe *Bordeaux*. Essa classe pode possuir uma especificação, onde a subclasse possuirá conceitos mais específicos em relação a sua classe pai. No exemplo, as subclasses são *Branco*, *Rose* e *Tinto*. As subclasses herdam todos os atributos de suas super classes. Atributos descrevem propriedades de classes e instâncias, na ontologia da figura 2.1 os vinhos *Château*, *Lafite*, *Rothschild* e *Pauillac* são de sabor encorpado; são produzidos pela vinícola *Château*, *Lafite*, *Rothschild*. O atributo *sabor* com o valor *encorpado,* juntamente com o atributo *fabricante* com as vinícolas *Château*, *Lafite*, *Rothschild* descrevem os vinhos. (NOY & McGUINNESS, 2000, apud BOFF, 2005).

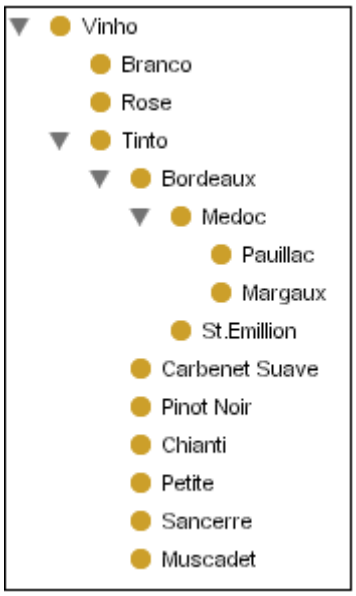

Figura 2.1 - Exemplo de uma ontologia de vinhos (simplificada, sem axiomas e relações).
Fonte: BOFF (2005)

2.3 Tipos de ontologias

De acordo com Guarino (1998) as ontologias são classificadas em quatro tipos, de acordo com o nível de generalidade necessária:

- *Ontologias de nível superior ou genéricas*: descrevem termos mais gerais, tais como espaço, tempo, matéria, objeto, evento e ação. São independentes de um problema ou domínio particular.

- *Ontologias de domínio e de tarefas*: essas ontologias, respectivamente, conceitualizam o vocabulário relacionado a um domínio genérico (como a medicina ou os automóveis) ou uma tarefa ou uma atividade genérica (como diagnósticos ou vendas), especializando os termos introduzidos na ontologia de nível superior.

- *Ontologias de aplicação:* descrevem conceitos dependentes do domínio e de tarefas particulares. Estes conceitos, freqüentemente, correspondem a papéis desempenhados por entidades de domínio, quando da realização de uma determinada atividade.

Na figura 2.2 é mostrada a relação entre estas ontologias. Os conceitos de uma ontologia de domínio ou de tarefa devem ser especializações dos termos introduzidos por uma ontologia genérica (alto nível). Enquanto que os conceitos de uma ontologia de aplicação

devem ser especializações dos termos das ontologias de domínio ou de tarefa correspondentes.

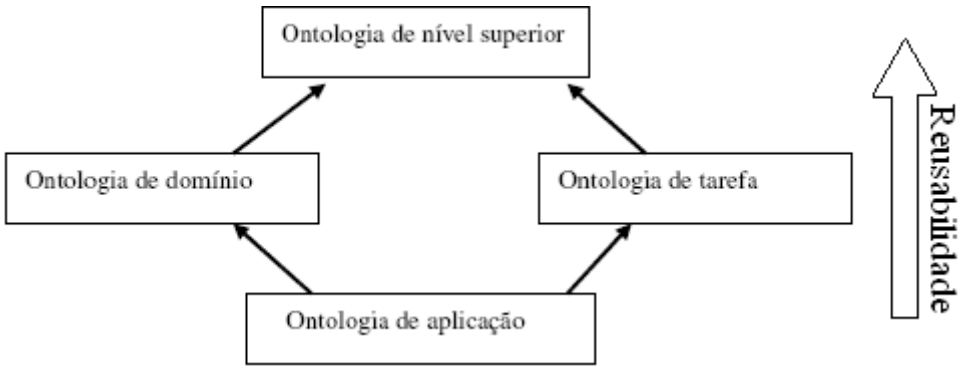

Figura 2.2 - Tipos de Ontologias e suas relações
Fonte: GUARINO (1998), adaptada

2.4 Vantagens do uso de ontologias

Após a descrição dos conceitos envolvidos com ontologias, é importante apresentar as principais vantagens do seu uso na ciência da computação. Três principais benefícios no emprego de ontologias são apresentados por Viera e Chaves (2001, apud CARLAN, 2006).

- Comunicação: são úteis por disponibilizarem várias formas para ajudar as pessoas a se comunicarem sobre um determinado conhecimento. Elas possibilitam que as pessoas raciocinem e entendam o domínio do conhecimento, atuando como referência para obter um consenso dentro de uma comunidade sobre o vocabulário técnico usado nas suas interações.

- Formalização: devido à natureza formal da notação utilizada nas ontologias, a especificação do domínio elimina contradições e inconsistências, envolvendo as restrições, resultando, em uma especificação não ambígua. Sendo assim, esta especificação formalizada pode ser automaticamente verificada e validada por um provador automático de teoremas. Isso também possibilita que um processo de inferência forme novos conhecimentos de forma automática, a partir da base de conhecimento já presente na ontologia.

- Representação do conhecimento e Reuso: as ontologias formam um vocabulário de consenso e representam o conhecimento de domínio de forma explícita no seu alto nível de abstração, possuindo um potencial enorme de reuso. O conhecimento

formalizado na camada de domínio pode ser especializado em diferentes aplicações, servindo diferentes propósitos, por diferentes equipes de desenvolvimento, em diferentes espaços do tempo.

2.5 Desvantagens do uso de ontologias

No subcapítulo anterior foram citadas algumas das vantagens do uso de ontologias. Apesar dessas vantagens, alguns problemas são identificados por O'Lery (1997, apud GUIZZARDI, 2000):

- A escolha de uma ontologia é um processo político, já que nenhuma ontologia pode ser totalmente adequada a todos os indivíduos ou grupos.

- As ontologias necessitam evoluir, não são estáticas. Poucos trabalhos têm enfocado a evolução de ontologias.

- Estender ontologias não é um processo direto. Ontologias são, geralmente, estruturadas de maneira precisa e, como resultado, são particularmente vulneráveis a questões de extensão, dado o forte relacionamento entre complexidade e precisão das definições.

- A interface entre elas constitui, portanto, um impedimento, especialmente porque cada uma delas é desenvolvida no contexto de um processo político. Ontologias desenvolvidas independentemente podem não se integrar efetivamente com outras por vários motivos, desde similaridade de vocabulário até visões conflitantes do mundo.

2.6 Construção de ontologias

De modo geral, para obter os benefícios proporcionados pelas ontologias, e para auxiliar o bom desenvolvimento das mesmas, deve ser considerado o seguinte conjunto de critérios (GRUBER, 1993):

- Clareza: uma ontologia deve, efetivamente, comunicar o significado pretendido dos termos definidos. Suas definições devem ser objetivas e independentes do contexto social ou computacional. Sempre que possível às descrições devem ser completas e documentadas em linguagem natural, com o objetivo de reforçar a clareza.

- Coerência: ser coerente significa dizer que as inferências sobre a ontologia devem ser consistentes com as definições axiomáticas. Esta coerência deve ser aplicada para os conceitos que são definidos de modo formal e também para os definidos de maneira informal, como aqueles descritos em documentos de linguagem natural e exemplos. Uma ontologia é denominada incoerente se uma de suas sentenças, que pode ser inferida através de seus axiomas, contradiz uma definição ou exemplo dado de maneira informal.

- Extensibilidade: ser extensível é a capacidade da ontologia definir novos termos para usos especiais, baseados em um vocabulário existente, sem haver a necessidade de rever as definições existentes.

- Compromisso mínimo com implementação: esse critério se refere à conceituação da ontologia, ela deve ser especificada no nível de conhecimento, sem se prender a uma determinada tecnologia de representação de conhecimento.

- Compromisso ontológico mínimo: uma ontologia requer o mínimo compromisso ontológico, suficiente para atender à intenção da atividade compartilhada do conhecimento. Uma ontologia deve fazer poucas imposições sobre o domínio que está sendo modelado, para assim permitir que, quando necessário, as partes comprometidas com a ontologia fiquem livres para a especializar e a instanciar.

Referente aos termos da ontologia, os seguintes critérios devem ser levados em consideração (USCHOLD, 1996):

- Definição dos termos em linguagem natural, da forma mais precisa possível;

- Garantia de consistências dos termos, através do uso de dicionários e glossários técnicos, quando for possível, deve-se evitar a introdução de novos termos;

- Identificar a relação do novo termo a ser inserido com os outros termos já existentes;

- Definição de cada termo, de forma a ser necessária e suficiente para especificar seu significado claramente;

- Apresentar exemplos quando for apropriado.

Anteriormente foram descritos os critérios para que as ontologias, de um modo geral, formem uma estrutura consistente, ajudando na reutilização do conhecimento conceitual para uma determinada área de conhecimento e facilitando ao mesmo tempo o seu compartilhamento e disseminação (VAN HEIJST et al., 1996, apud MACEDO, 2003).

2.7 Ferramentas para a construção de ontologias

No processo de construção de ontologias, qualquer auxílio pode resultar em ganhos significativos. Sendo assim, existem ferramentas gráficas com o objetivo de auxiliar no desenvolvimento de ontologias. A ontologia utiliza-se da representação gráfica como uma ferramenta para garantir um projeto lógico mais bem estruturado de um sistema (CAMPOS, 2004).

Almeida e Bax (2003) expõem uma lista de ferramentas para a construção, uso e edição de ontologias, dentre essas, são exibidas uma breve descrição de duas das mais citadas na literatura:

Protégé-2000[2]: é um ambiente interativo para projeto de ontologias, de código aberto, que oferece uma interface gráfica para edição de ontologias e uma arquitetura para a criação de ferramentas baseadas em conhecimento. A arquitetura é modulada e permite a inserção de novos recursos (NOY e MUSEN, 2000, apud ALMEIDA, BAX, 2003).

O desenvolvimento de uma ontologia no Protégé-2000 consiste basicamente nos seguintes passos: primeiramente é definido um esquema de classes *(class*s), subclasses *(subclas*s), propriedades e relações *(slot*s) referentes ao domínio que se deseja modelar (FELICÍSSIMO et al., 2003, apud BOFF, 2005).

Nessa ferramenta é possível visualizar a hierarquia de classes no formato de árvore, onde são exibidos seus relacionamentos e suas instâncias, como exemplifica a figura 2.3.

[2] http://protege.stanford.edu

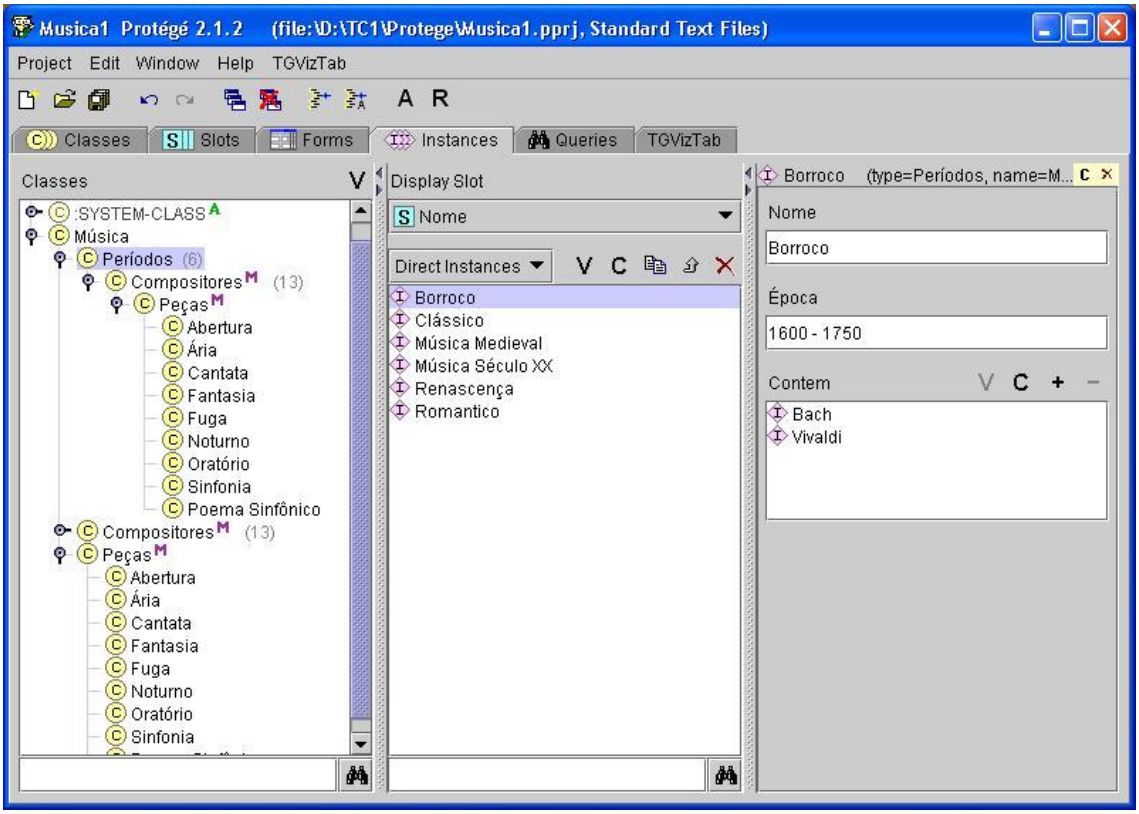

Figura 2.3 - Exemplo de uma ontologia no Protégé-2000
Fonte: BOFF (2005)

Uma outra característica dessa ferramenta é a possibilidade de criar uma pequena *query* para consultas futuras na ontologia e a existência de um *plugin* que converte a ontologia em OWL.

OntoEdit[3]: é uma ferramenta comercial, nela existe um ambiente gráfico para edição de ontologias que permite inspeção, navegação, codificação e alteração de ontologias. O modelo conceitual é armazenado usando um modelo de ontologia que pode ser mapeado em diferentes linguagens de representação. As ontologias são armazenadas em bancos relacionais e podem ser implementadas em XML, Flogic, RDF(S) e DAML+OIL (Maedche et al., 2000, apud ALMEIDA, BAX, 2003).

[3] http://www.ontoprise.de/products/ontoedit_en

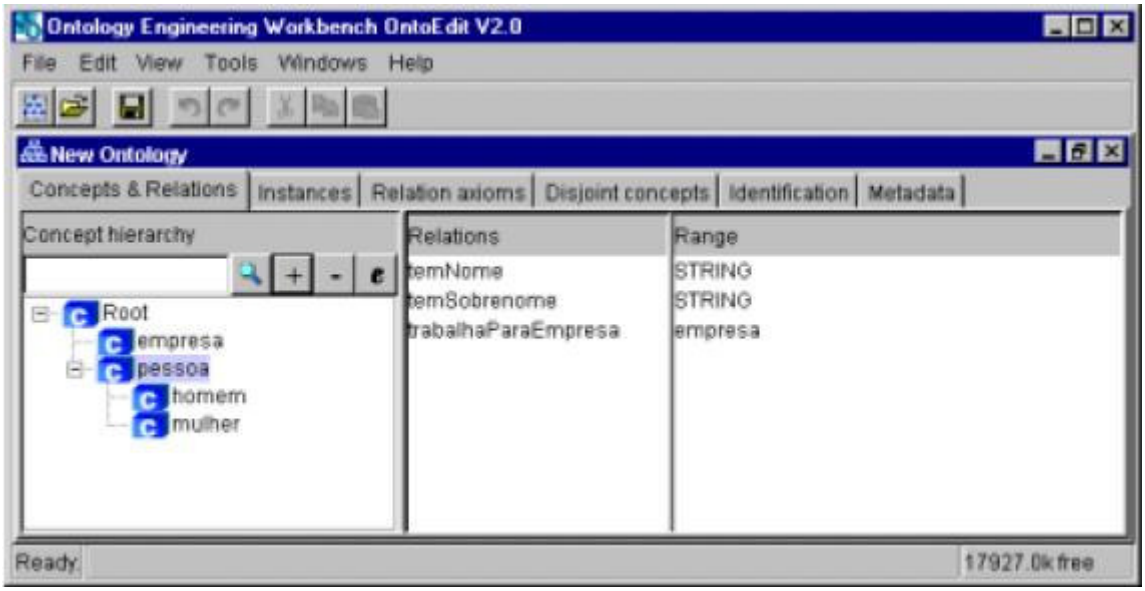

Figura 2.4 - Ambiente para construção de ontologias
Fonte: BONIFÁCIO (2002)

2.8 Linguagens para construção de ontologias

Existem várias linguagens para a construção de ontologias, cada uma disponibiliza diferentes facilidades. Dentre esse conjunto de linguagens pode-se citar: a SHOE[4] (*Simple* HTML *Ontology Extensions*), XOL[5] (*Ontology Exchange Language*), OIL[6] (*Ontology Inference Layer*) e DAML (DARPA *Agent MarkupLanguage*)[7]. A combinação entre estas duas últimas formou a DAML+OIL[8].

O objetivo deste trabalho não é tratar, nem fazer uma comparação entre as linguagens existentes para a construção de ontologias, e sim apenas citá-las como ferramentas importantes para a sua estruturação. Porém, será tratada com maiores detalhes a linguagem OWL (*Ontology Web Language*) (SMITH, WELTY & McGUINNESS, 2004). Essa linguagem é baseada em lógica de descrição e é recomendada pela W3C, a qual espera que ela se torne um padrão no desenvolvimento de ontologias. Desta forma, no próximo capítulo, são

[4] http://www.cs.umd.edu/projects/plus/SHOE/
[5] http://www.ai.sri.com/pkarp/xol/
[6] http://www.ontoknowledge.org/oil/
[7] http://www.daml.org/
[8] http://www.daml.org/

apresentados os conceitos que envolvem as lógicas de descrição e logo em seguida, no capítulo 4, é apresentada a linguagem OWL.

3 LÓGICA DE DESCRIÇÃO

As Lógicas de Descrição (DL – *Description Logic*) (NARDI et al., 2002; FRANCONI, 2002) são conjuntos de formalismos de representação do conhecimento que representam algum domínio, definindo os conceitos relevantes a este domínio, ou seja, a terminologia, com a utilização destes conceitos especificam as propriedades de objetos e indivíduos do mundo representado, formando a descrição desse domínio.

Dessa forma, as lógicas dessa família são utilizadas para a especificação de classes de dados e de relacionamentos entre estas, possuindo uma semântica formal, baseada em lógica, e mecanismos de inferência utilizados para extrair conhecimento que não esteja explicitado na base de conhecimento do domínio.

Na lógica de descrição, um sistema de representação do conhecimento é formado com base em uma linguagem conceitual ou terminologia*,* essa linguagem consiste num grupo de construções que representam classes e suas relações em um determinado domínio do conhecimento. Os conceitos podem ser considerados como classes de indivíduos e os papéis definem as propriedades e ligações entre essas classes.

A fundamentação de uma linguagem conceitual está nas primitivas ou axiomas terminológicos, os quais são símbolos de um alfabeto. Os axiomas terminológicos que formam a linguagem são os conceitos atômicos, também chamados de nomes de conceitos, os papéis atômicos*,* também chamados de nomes de papéis, os nomes de atributos, os nomes de indivíduos e objetos.

3.1 Estrutura de uma lógica descritiva

Uma lógica descritiva é formada por uma linguagem descritiva, que é utilizada para definir como os conceitos e os papéis são formados, por um mecanismo para especificar

dados sobre os conceitos e papéis (TBox), por um mecanismo para especificar as propriedades dos indivíduos (ABox) e por maneiras de se inferir sobre uma base de conhecimento (CALVANESE, 2003).

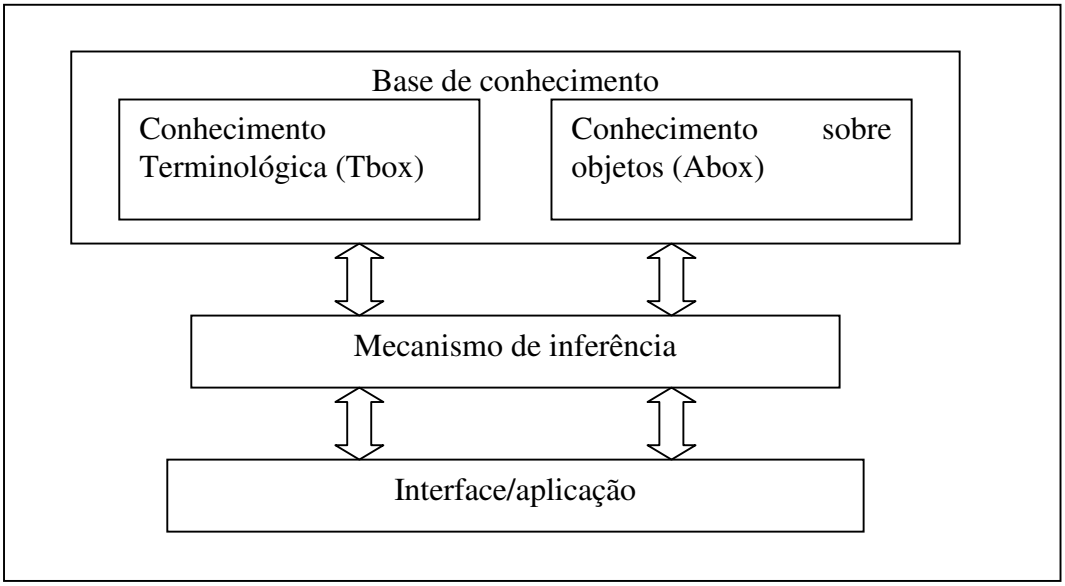

Figura 3.1 - Estrutura de um sistema em Lógica Descritiva
Fonte: CALVANESE (2003)

A base de conhecimento é formada por duas partes, uma parte que trata da terminologia, que possui um conjunto de declarações e axiomas que descrevem a estrutura do domínio (TBox), também chamada de conhecimento intensional, e por outra parte que trata das assertivas sobre os indivíduos, ou objetos (ABox), ou seja, a Abox contém o conhecimento extensional sobre o domínio de interesse (NARDI, 2002).

A forma básica de declaração em um Tbox é a definição de conceitos, sendo essa base constituída por um conjunto de afirmativas de inclusão entre os conceitos. Na declaração a baixo há a expressão do conceito ‘Mulher’, que é traduzido como um indivíduo que pertença ao conceito ‘Pessoa’ e ao conceito ‘Fêmea’:

$$\text{Mulher} \equiv \text{Pessoa} \prod \text{fêmea}$$

Nessa outra declaração, está descrito que para ser homem o indivíduo tem que pertencer ao conceito ‘Pessoa’ e não pertencer ao conceito ‘Mulher’:

$$\text{Homem} \equiv \text{Pessoa} \prod \neg\text{Mulher}$$

Para a Abox pode-se fazer as declarações de duas formas, onde a construção do conhecimento extensional é realizada quando conceitos e papéis formam assertivas sobre indivíduos:

Declaração de conceitos: Mulher(Maria), quer dizer que o individuo ‘Maria’ pertence ao conceito ‘Mulher’.

Declaração de papéis: temFilho(Maria, João), quer dizer que o indivíduo ‘Maria’ se relaciona com o indivíduo ‘João’ através do papel ‘temFilho’.

Existem diferentes lógicas descritivas, cada uma sendo adequada a determinada situação. Como exemplo são exibidos na tabela 3.1 os componentes da linguagem de descrição básica chamada AL, da qual as outras linguagens descritivas são extensões (MOREIRA, 2002). As letras C e D representam descrição de conceitos, a letra R representa papéis e a letra A representa conceitos atômicos.

Tabela 3.1 - Gramática da linguagem descritiva AL

C,D::=	A	Conceito atômico
	⊤	Conceito universal
	⊥	Conceito botton
	¬A	Negacão atômica
	C ⊓ D	Intersecão
	∀R.C	Restricão de valor
	∃R. ⊤ ∃R. ⊥	Quantificação existencial limitada

Fonte: MOREIRA (2002)

3.2 Inferência

Os sistemas de lógica descritiva não só armazenam terminologias e afirmativas, mas também possibilitam serviços que realizam inferência sobre o conhecimento representado. Nas seções seguintes são apresentados serviços típicos de inferência em TBox e ABox, como também uma descrição do método de inferência Tableaux.

3.2.1 Inferência em TBox

A baixo são apresentados alguns serviços típicos de inferência em TBox (VIEIRA, 2005; BONIFACIO, 2002):

3.2.1.1 Satisfabilidade

A satisfação dos conceitos checa se uma descrição de conceito nunca pode ter instâncias por causa das inconsistências ou contradições do modelo. Um exemplo de insatisfabilidade é o conceito Hermafrodita:

Mulher ≡ Pessoa ∏ Fêmea

Homem ≡ Pessoa ∏ ¬Mulher

Hermafrodita ≡ Homem ∏ Mulher

3.2.1.2 Subclassificação

Checagem de classificação entre duas descrições de conceitos, C e D; C inclui D quando o conjunto de objetos que são instâncias de D também são um subconjunto de objetos que são instâncias de C.

Um exemplo é o conceito Mãe, que é composto por instâncias do conceito Mulher e tem um relacionamento, através da propriedade temFilho, com o conceito Pessoa:

Mulher ≡ Pessoa ∏ Fêmea

Mãe ≡ Mulher ∏ ∃ temFilho.Pessoa

3.2.1.3 Equivalência

Dizer que os conceitos C e D são equivalentes, C ≡ D, é o mesmo que dizer que eles possuem as mesmas instâncias.

Uma exemplificação de equivalência é o conceito Homem e Masculino:

Mulher ≡ Pessoa ∏ Fêmea

Homem ≡ Pessoa ∏ ¬Mulher

Masculino ≡ Pessoa ∏ ¬Fêmea

3.2.2 Inferência em ABox

A baixo são apresentados alguns serviços típicos de inferência em ABox (VIEIRA, 2005):

3.2.2.1 Checagem de consistência

O ABox será consistente se existir uma instância que torne tanto o ABox quanto o TBox verdadeiros. Isso pode ser exemplificado quando se cria uma instância do conceito Mulher denominada Maria. Como o conceito Mulher é formado pelos conceitos Pessoa e Fêmea, Maria também fará parte desses conceitos. Sendo o conceito Mulher satisfatível, o ABox será consistente:

TBox:

Mulher ≡ Pessoa ∏ Fêmea

ABox:

Mulher(Maria)

Checagem da consistência:

Mulher(Maria) ≡ Pessoa(Maria) ∏ Fêmea(Maria)

3.2.2.2 Checagem de instância

Aqui é verificado se um determinado indivíduo é uma instância de um conceito específico. Um exemplo dessa inferência é dizer que o indivíduo Maria faz parte do conceito Mãe, sendo que o que se tem explicitado é que Maria é uma Mulher e possui um relacionamento, através da propriedade temFilho, com Pedro.

TBox:

Mulher ≡ Pessoa ∏ Fêmea

Mãe ≡ Mulher ∏ ∃ temFilho.Pessoa

ABox:

Mulher(Maria)

Homem(Pedro)

temFilho(Maria,Pedro)

A partir dessas informações pode-se dizer que o indivíduo Maria pertence ao conceito Mãe:

Mãe(Maria) ≡ Mulher(Maria) ∏ ∃ temFilho.Pessoa

3.2.2.3 Retorno

Encontra o conceito mais específico ao qual o indivíduo é uma instância. Aplicando-se esse serviço no indivíduo Maria tem como retorno o conceito mais baixo na hierarquia de conceito a qual o indivíduo pertence:

ABox:

Mulher(Maria)

Mãe(Maria)

Retorno:

Maria → Mãe

3.2.2.4 Realização

Esse serviço identifica na base de conhecimento os indivíduos que são instâncias de um determinado conceito.

ABox:

Mulher(Gisa)

Mulher(Maria)

Homem(Diego)

Resultado:

Mulher → Gisa, Maria

Homem → Diego

3.2.3 Método Tableaux

É um método de prova de refutação, no qual um teorema é provado pelo insucesso na tentativa de construção sistêmica de um modelo para a sua negação. O método possui fundamento na semântica: ao tentar provar que α é um teorema lógico, o que o método faz de fato é verificar a impossibilidade da insatisfação de $\sim\alpha$ (BUCHSBAUM E PEQUENO, 1990, apud BRITO, 2003)

A figura 3.2 mostra os oito fatos que formam os fundamentos básicos para a construção desse método:

1)	a) Se ~X é verdadeiro, então X é falso.
	b) Se ~X é falso, então X é verdadeiro.
2)	a) Se a conjunção X ∧ Y é verdadeira, então X, Y são ambos verdadeiros.
	b) Se a conjunção X ∧ Y é falsa, então X é falso, ou Y é falso ou ambos são falsos.
3)	Se a disjunção X ∨ Y é verdadeira, então X é verdadeiro, ou Y é verdadeiro, ou ambos são verdadeiros.
	b) Se a disjunção X ∨ Y for falsa, então X e Y são ambos falsos.
4)	a) Se X → Y é verdadeiro, então X é falso ou Y é verdadeiro.
	b) Se X → Y é falso, então X é verdadeiro e Y é falso.

Figura 3.2 - Fatos que formam os fundamentos básicos do método Tableaux
Fonte: SMULLYAN (1995, apud BRITO, 2003)

A prova de refutação é feita através da aplicação de regras específicas para a lógica de descrições que dependem dos construtores da lógica em questão (VIEIRA, 2005).

Seguindo exemplo do mesmo autor acima citado, utilizando Tableaux para verificar se um conceito D($C \subseteq D$) deve-se provar que $C \prod \neg D$ é insatisfatível. A prova que $C \prod \neg D$ é a negação de $C \subseteq D$ é demonstrada a seguir:

1. $C \subseteq D \equiv C \rightarrow D$

2. $C \rightarrow D \equiv \neg C \cup D$

3. $\neg(\neg C \cup D) \equiv C \prod \neg D$

Então a expressão geral da subclassificação é apresentada a baixo:

$C \subseteq D$ se somente se $C \prod \neg D$

4 WEB ONTOLOGY LANGUAGE[9]

Dentre as diversas linguagens de marcação para ontologias se destaca a OWL (Web Ontology Language) (SMITH, WELTY & McGUINNESS, 2004), que é uma revisão da linguagem DAML+OIL. A OWL é uma linguagem que faz parte da lista de recomendações do consórcio W3C (*World Wide Web Consortium*)[10]. Ela disponibiliza mecanismos para representar explicitamente o significado dos termos e os relacionamentos entre os mesmos, descrevendo características especiais sobre os conceitos e os relacionamentos através de axiomas lógicos.

É uma linguagem para ontologias Web que ao invés de apenas apresentar informações aos usuários, também viabiliza o processamento do conteúdo dessas informações, quando utilizada por aplicações. A OWL pode ser entendida como uma linguagem para definição e instanciação de ontologias Web, sendo que estas devem incluir descrições de classes, propriedades e suas instâncias.

A OWL foi projetada de modo a atender as necessidades das aplicações para a Web semântica. Esse termo foi proposto por Tim Berners Lee, também idealizador da World Wide Web. Segundo Berners a definição para a Web semântica é a seguinte:

"A Web semântica não é uma Web separada, mas uma extensão da atual, na qual a informação é utilizada com significado bem definido, aumentando a capacidade dos

---

[9] Apesar de não constituir um acrônimo, o nome completo é descrito pelos autores em http://www.w3.org/2004/OWL/

[10] http://www.w3c.org – Consorcio internacional de instituições e empresas para o desenvolvimento de tecnologias para web.

computadores para trabalharem em cooperação com as pessoas". (BERNERS, HENDLER & LASSILA, 2001)

As necessidades das aplicações para a Web Semântica podem ser resumidas em (BREITMAN, 2005):

- Construção de ontologias
  - Criar uma ontologia;
  - Explicar conceitos fornecendo informações sobre os mesmos;
  - Explicar propriedades fornecendo informações sobre as mesmas.
- Explicitar fatos sobre um determinado domínio
  - Fornecer informações sobre indivíduos que fazem parte do domínio em questão.
- Racionalizar sobre ontologias e fatos
  - Determinar as conseqüências do que foi construído e explicitado.

De modo similar a DAML+OIL, a intenção da OWL é representar conceitos e seus relacionamento na forma de uma ontologia. A OWL disponibiliza três sublinguagens incrementais, que foram criadas para serem utilizadas por diferentes comunidades de desenvolvedores e usuários (HARMELEN & MCGUINNESS, 2004) (HARMELEN & MCGUINNESS, 2004): OWL Lite, OWL DL e OWL Full.

Numa breve descrição, segundo os autores acima citados, a OWL Lite foi projetada para ser de fácil utilização, com um conjunto básico de características da OWL, é destinada para usuários que estão iniciando com a linguagem e que necessitam de uma classificação hierárquica e restrições simples. A OWL DL (*Description Language*) tem como finalidade prover um maior grau de expressividade, onde todas as conclusões são computáveis (completude) e todas as computações terminam em tempo finito (decidíveis). A OWL Full é destinada aos usuários que desejam expressividade máxima e liberdade de sintaxe do RDF (*Resource Description Framework*) (MANOLA & MILLER, 2003).

4.1 Estrutura das ontologias

De acordo com Smith, Welty e McGuinness (2004), nas primeiras linhas de um arquivo OWL são inseridas as *namespaces* da ontologia. Essas informações tem o objetivo de indicar qual vocabulário está sendo utilizado, possibilitando que os termos de uma ontologia possam ser interpretados sem ambigüidade**.** A figura 4.1 exemplifica a declaração das *namespaces* em uma ontologia.

```
<rdf:RDF
    xmlns     ="http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#"
    xmlns:vin ="http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#"
    xml:base  ="http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#"
    xmlns:food="http://www.w3.org/TR/2004/REC-owl-guide-20040210/food#"
    xmlns:owl ="http://www.w3.org/2002/07/owl#"
    xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:xsd ="http://www.w3.org/2001/XMLSchema#">
```

Figura 4.1 - Exemplo da declaração das namespaces em uma ontologia.
Fonte: SMITH, WELTY & MCGUINNESS (2004)

Normalmente um conjunto de informações a respeito da própria ontologia também é inserido na estrutura de um documento OWL. Os elementos que contém essas informações recebem o nome de *headers* e são agrupados na *tag* <owl:Ontology>.

Nos headers é possível inserir informações como a URI base que identifica a ontologia, comentários a seu respeito, sua versão, uma referência à outra ontologia que contém definições que são utilizadas como parte do significado da ontologia em questão, e a possibilidade de se inserir um rótulo, em linguagem natural, a respeito dessa ontologia. A utilização desses *headers* é exemplificada na figura 4.2.

```
<owl:Ontology rdf:about="">
  <rdfs:comment>An example OWL ontology</rdfs:comment>
  <owl:priorVersion rdf:resource="http://www.w3.org/TR/2003/PR-owl-guide-20031215/wine"/>
  <owl:imports rdf:resource="http://www.w3.org/TR/2004/REC-owl-guide-20040210/food"/>
  <rdfs:label>Wine Ontology</rdfs:label>
  ...
```

Figura 4.2 - Exemplo de headers em uma ontologia.
Fonte: SMITH, WELTY & MCGUINNESS (2004)

Após a declaração dos cabeçalhos que descrevem a ontologia, o agrupamento deve ser encerrado com a tag </owl:Ontology>.

4.2 Elementos básicos

Os elementos básicos para a construção de uma ontologia sobre a linguagem OWL são as classes, propriedades, as instâncias das classes e os relacionamentos entre essas instâncias (SMITH, WELTY & MCGUINNESS, 2004). A estrutura básica que envolve esses elementos é exemplificada nos próximos dois sub-capítulos. Os exemplos foram retirados da ontologia Ontomúsica (BOFF, 2005), a qual retrata o conhecimento sobre a história da música e está descrita no capítulo seguinte.

4.2.1 Classes, subclasses e indivíduos

De acordo com (BECHHOFER et al., 2004), as classes provêem um mecanismo de abstração para agrupar recursos com características similares. Esse conjunto de indivíduos que está associado à classe é chamado de extensão de classe. Os indivíduos dessa extensão são chamados de instâncias da classe.

Seguindo a idéia dos mesmos autores citados acima, outro construtor fundamental na taxonomia de classes é a representação de subclasses, desta forma pode-se construir uma hierarquia, onde a subclasse herda as propriedades da superclasse.

De acordo com o exemplo da figura 4.3, logo abaixo, são definidas as classes: Compositor, Obra e Vocal. Na definição da classe Vocal é declarada a sua subclasse Profana. Sendo assim, o conjunto de indivíduos da classe Profana será um subconjunto da classe Vocal.

```
<owl:Class rdf:ID="Compositor"/>
<owl:Class rdf:ID="Obra"/>
<owl:Class rdf:ID="Profana">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Vocal"/>
  </rdfs:subClassOf>
</owl:Class>
```

Figura 4.3 - Declaração de classes e subclasse em documento OWL
Fonte: BOFF (2005)

Os indivíduos da ontologia podem ser representados da seguinte maneira:

```
<Obra rdf:ID="A_bela_moleira">
```

Essa linha diz que o indivíduo “A_bela_moleira” é uma instância da classe Obra, ou seja, essa declaração diz que A_bela_moleira é uma obra musical.

4.2.2 Propriedades e relacionamentos

As propriedades podem ser utilizadas para definir relacionamentos entre indivíduos ou entre indivíduos e valores de dados. Sendo assim, ainda de acordo com (BECHHOFER et al., 2004), as propriedades se dividem em duas categorias: *Datatype property*, que associa indivíduos a valores de dados, e *Object Property*, que associa indivíduos a indivíduos. Nas figuras 4.4 e 4.5 essas duas categorias serão exemplificadas, respectivamente.

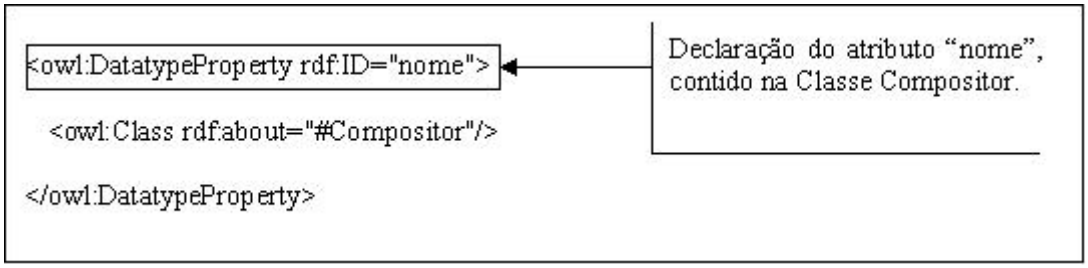

Figura 4.4 - Exemplo de um atributo Datatype Property
Fonte: BOFF (2005)

Os *datatypes* ilustrados na tabela a seguir são os recomendados pelo W3C para a utilização em OWL:

Tabela 4.1 – Datatypes recomendados pela W3C para utilização em OWL

xsd:string	xsd:normalizesString	xsd:boolean
xsd:decimal	xsd:float	xsd:double
xsd:integer	xsd:nonNegativeInteger	xsd:positiveInteger
xsd:nonPositiveInteger	xsd:negativeInteger	xsd:long
xsd:int	xsd:short	xsd:byte
xsd:unsignedLong	xsd:unsignedInt	xsd:unsignedShort
xsd:unsignedByte	xsd:hexBinary	xsd:base64Binary
xsd:dateTime	xsd:time	xsd:date
xsd:gYearMonth	xsd:gYear	xsd:gMonthDay
xsd:gDay	xsd:gMonth	xsd:anyURI
xsd:token	xsd:language	xsd:NMTOKEN
xsd:Name	xsd:NCName	

Fonte: BREITMAN (2005, pág. 63)

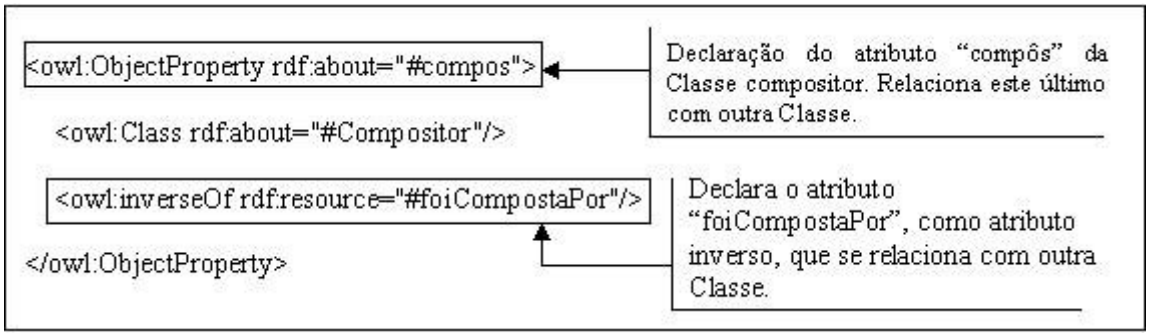

Figura 4.5 - Exemplo de um atributo Object Property
Fonte: BOFF (2005)

4.3 Restrições em propriedades

Na linguagem OWL as propriedades criam restrições, as quais são utilizadas para definir limites para os indivíduos de uma determinada classe. Essas restrições podem ser de três tipos (BREITMAN, 2005):

- Restrições que utilizam quantificadores;
- Restrições de cardinalidade;
- Restrições do tipo *hasValue* ("tem valor de").

4.3.1 Quantificadores

Os quantificadores utilizados nas restrições podem ser do tipo existencial (∃) ou universal (∀). O existencial indica a existência de pelo menos um elemento, e em OWL é representado pela expressão *someValuesFrom*. Exemplificando, ∃ Compositor compoe MusicaReligiosa – significa que o Compositor compõe músicas religiosas, mas também pode compor outros tipos de músicas.

O universal pode ser interpretado como 'apenas' e é representado em OWL pela expressão *allValuesFrom*. Exemplificando, ∀ Compositor compoe MusicaReligiosa – significa que Compositor compõe músicas religiosas ou não compõe nenhum outro tipo de música.

As restrições existenciais são as mais utilizadas em OWL. Para um conjunto de indivíduos, uma restrição existencial especifica a existência de pelo menos um

relacionamento que utiliza uma dada propriedade com um indivíduo de uma classe específica (BREITMAN, 2005).

4.3.2 Cardinalidade

A OWL permite utilizar restrições de cardinalidade, uma vez que qualquer instância de uma classe pode ter um número arbitrário de valores para uma propriedade. Essas restrições são locais, pois são declaradas em propriedades com respeito a uma classe. A OWL disponibiliza três construtores para cardinalidade (BECHHOFER et al., 2004):

owl:*minCardinality*: esse elemento é utilizado para limitar a quantidade mínima de ocorrências para uma determinada propriedade, e deve ser representada por um valor inteiro positivo, como é exemplificado abaixo:

```
<owl:Class rdf:ID="Obra">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:minCardinality
                rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">
       3
      </owl:minCardinality>
      <owl:onProperty>
          <owl:ObjectProperty rdf:about="#caracteristicasDaObra"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Figura 4.6 - Exemplo da utilização do elemento *minCardinality*

Este exemplo apresenta uma classe que representa uma obra musical. Nele, o construtor *owl*:*minCardinality* é utilizado para especificar que um indivíduo da classe obra deve obrigatoriamente conter, no mínimo, três características.

owl:*maxCardinality*: esse elemento é utilizado para limitar a quantidade máxima de ocorrências para uma determinada propriedade, a qual deve ser representada por um valor inteiro positivo. Para implementar esse elemento basta substituir o termo *minCardinality* do trecho de código mostrado no exemplo da Figura 4.6 pelo termo *maxCardinality*. Então a instância da classe 'Obra' só poderá ter no máximo três características.

owl:*cardinality*: esse elemento obriga uma classe a conter o número exato de relacionamentos indicados pela restrição por meio de uma determinada propriedade. Para implementar esse elemento basta substituir o termo *minCardinality* do trecho de código

mostrado no exemplo da Figura 4.6 pelo termo c*ardinality*. Sendo assim, a instância da classe 'Obra' deverá ter exatamente três características.

4.3.3 Informação do valor

O elemento *hasValue* é utilizado para restringir o valor de uma propriedade a uma determinada classe. Este recurso determina que para ser uma instância de uma determinada classe é necessário que uma propriedade tenha um valor especifico pré-determinado. O elemento é exemplificado na figura abaixo:

```
<owl:Class rdf:ID="Religiosa">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#generoDaObra"/>
      <owl:hasValue rdf:resource="#Religioso"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Figura 4.7 - Exemplo da utilização do elemento *hasValue*

Este exemplo diz que qualquer obra Religiosa é considerada como sendo do gênero Religioso. Com isso, a classe "Religiosa" apresenta uma restrição, por meio do elemento *owl*:*hasValue*, em que as instâncias desta classe têm como valor da propriedade "generoDaObra" o objeto "Religioso".

4.4 Igualdade

A OWL apresenta características relacionadas à igualdade e desigualdade, para isso são utilizados os recursos descritos abaixo (BECHHOFER et al., 2004).

4.4.1 Equivalência entre classes

O elemento *equivalentClass* é utilizado quando duas classes apresentam as mesmas instâncias. Esta construção pode ser usada para criar classes sinônimas e para ligar duas classes na mesma ontologia ou em ontologias diferentes.

```
<owl:Class rdf:ID="Obra">
  <owl:equivalentClass rdf:resource="Musica"/>
</owl:Class>
```

Figura 4.8 Exemplo da utilização do elemento *equivalentClass*

No exemplo da figura 4.8, é feita a definição da classe ‘Obra’ e ao mesmo tempo a representação da classe ‘Música’ como classe equivalente. Sendo assim, elas podem ser consideradas equivalentes se uma instância de ‘Obra’ puder pertencer também a ‘Musica’. Desta maneira, se um sistema de inferência estiver procurando por ‘Obra’ e encontrar a classe ‘Musica’, nesse contexto, ele pode consultar a classe encontrada para apresentar os resultados da busca.

4.4.2 Equivalência entre propriedades

Para definir a equivalência entre propriedades é utilizado o elemento *equivalentProperty.* Esse construtor é utilizado para representar propriedades que apresentam o mesmo valor. A próxima figura exemplifica esse elemento.

```
<owl:DataTypeProperty rdf:ID="nome">
  <owl:equivalentProperty rdf:resource="titulo"/>
  <rdfs:domain rdf:resource="#Obra" />
</owl:DataTypeProperty>
```

Figura 4.9 Exemplo da utilização do elemento *equivalentProperty*

Nesse exemplo, a propriedade ‘nome’ da classe ‘Obra’ é equivalente a propriedade ‘titulo’.

4.4.3 Equivalência entre instâncias

O elemento *sameAs* possibilita fazer a declaração de equivalência entre duas instâncias da ontologia. Porém, esse atributo também pode ser utilizado para realizar a igualdade entre outros componentes, como por exemplo, entre classes (BECHHOFER et al., 2004). Esse construtor é comumente aplicado na importação de ontologias, com o objetivo de declarar a igualdade entre indivíduos das ontologias do contexto.

```
<Obra rdf:ID="O_rei_dos_Elfos">
  <owl:sameAs rdf:resource="#Rei_dos_elfos"/>
</Obra>
```

Figura 4.10 Exemplo da utilização do elemento *sameAs*

No trecho de código da figura acima, representa-se que a instância de obra ‘O_rei_dos_Elfos’ é igual ao indivíduo ‘Rei_dos_elfos’.

A OWL também disponibiliza construtores para se especificar que indivíduos são diferentes, isso é possível através dos elementos:

differentFrom: esse elemento explicita que a diferença entre indivíduos do contexto.

```
<Obra rdf:ID="O_rei_dos_Elfos">
  <owl:differentFrom rdf:resource="#O_canto_do_cisne"/>
  <owl:differentFrom rdf:resource="#O_caminhante"/>
</Obra>
```

Figura 4.11 Exemplo da utilização do elemento *differentFrom*

Nesse exemplo é explicitado que o indivíduo ‘O_rei_dos_Elfos’ é diferente dos indivíduos ‘O_canto_do_cisne’ e ‘O_caminhante’. Porém, essa declaração não diz que o indivíduo ‘O_canto_do_cisne’ é diferente do indivíduo ‘O_caminhante’.

allDifferent: através desse elemento pode-se especificar que os indivíduos de uma classe são diferentes uns dos outros.

```
<owl:allDifferent>
  <owl:distinctMembers rdf:parseType="Collection">
    <Obra rdf:about="#O_rei_dos_Elfos" />
    <Obra rdf:about="#O_canto_do_cisne" />
    <Obra rdf:about="#O_caminhante" />
  <owl:distinctMembers rdf:parseType="Collection"   >
<owl:allDifferent>
```

Figura 4.12 Exemplo da utilização do elemento *allDifferent*

Da maneira como foi especificado na figura 4.12, as três instâncias declaradas são diferentes entre si.

4.5 Classes complexas

A OWL oferece construtores adicionais que podem ser usados na construção de classes. A linguagem suporta operações como união, interseção e complemento. As classes podem ser enumeradas e também é possível afirmar que as extensões de classes são disjuntas.

4.5.1 Conjunto de operadores booleanos

A OWL permite combinações booleanas para manipulações das classes. Os membros das classes construídas são especificados por um conjunto de operadores, o qual pode ser visualizado como uma representação dos operadores AND, OR e NOT, usados na lógica descritiva. Abaixo, esses operadores são descritos.

intersectionOf: declara uma classe cuja extensão contém somente indivíduos que são membros da extensão de classe de todas as classes descritas em uma lista. Este operador é equivalente à conjunção lógica, AND.

```
<owl:Class rdf:ID="InstrumentoDedilhado">
    <owl:intersectionOf rdf:parseType="#Collection">
            <owl:disjointWith rdf:resource="#InstrumentoMusical" />
            <owl:disjointWith rdf:resource="#Dedilhada" />
    </owl:intersectionOf>
</owl:Class>
```

Figura 4.13 Exemplo da utilização do elemento *intersectionOf*

No exemplo acima, o conceito ‘InstrumentoDedilhado’ é formado pela intersecção das classes ‘InstrumentoMusical’ e ‘Dedilhada’.

unionOf: declara uma classe cuja extensão contém indivíduos que ocorrem em pelo menos uma extensão de classe das classes descritas em uma lista. Este operador é equivalente a disjunção lógica, OR.

```
<owl:Class rdf:ID="Percussao">
    <owl:unionOf rdf:parseType="#Collection">
            <owl:Class rdf:about="#SomIndeterminado" />
            <owl:Class rdf:about="#SomDeterminado" />
    </owl:unionOf>
</owl:Class>
```

Figura 4.14 Exemplo da utilização do elemento *unionOf*

No exemplo da figura 4.14 é especificado que a classe ‘Percussao’ é formada pela união de indivíduos das classes ‘SomIndeterminado’ e ‘SomDeterminado’.

complementOf: este construtor é definido em uma classe para especificar os tipos de instâncias que a classe não suporta. Sendo assim, as instâncias, com exceção daquelas definidas por *owl*:*complementOf*, podem ser reconhecidas como instância da classe definida por este elemento. Este operador é equivalente a negação lógica, NOT.

4.5.2 Classes enumeradas

oneOf: esse elemento é utilizado em uma classe para descrever uma enumeração de instâncias permitidas. Todas as instâncias que uma classe poderá ter são pré-definidas com o uso desse elemento.

```
<owl:Class rdf:ID="Corda">
  <owl:oneOf rdf:parseType="Collection">
    <owlThing rdf:about="#Dedilhada" />
    <owlThing rdf:about="#Friccionada" />
    <owlThing rdf:about="#Percutida" />
  </owl:oneOf>
</owl:Class>
```

Figura 4.15 Exemplo da utilização do elemento *oneOf*

Neste exemplo, está sendo representado que o instrumento musical ‘Corda’ deve ser um instrumento dedilhado, friccionado ou percutido. Nenhum outro indivíduo pode fazer parte da classe ‘Corda’ a não ser os declarados pela enumeração.

4.5.3 Classes disjuntas

disjointWith: esse elemento é utilizado para especificar a disjunção entre classes. Sendo assim, uma mesma instância não pode estar relacionada com duas classes declaradas como disjuntas.

```
<owl:Class rdf:ID="Religiosa">
    <rdfs:subClassOf rdf:resource="#Vocal" />
    <owl:disjointWith rdf:resource="#Profana" />
</owl:Class>
```

Figura 4.16 Exemplo da utilização do elemento *disjointWith*

Da maneira como foi declaro no exemplo acima, um sistema de inferência pode deduzir que se um indivíduo é instância de ‘Religiosa’ então ele não é instância de ‘Profana’.

O próximo capítulo apresenta a estrutura de uma ontologia sobre a história da música, a qual foi transformada em um arquivo OWL e serve como estudo de caso para o sistema de inferência a ser proposto.

5 ONTOMÚSICA

A Ontomúsica é uma ontologia desenvolvida por Rogério Eduardo Boff em seu trabalho de conclusão do curso de Ciência da Computação no Centro Universitário Feevale (BOFF, 2005).

O principal objetivo do trabalho é viabilizar conhecimento sobre a História da Música na Web, através da estrutura de ontologias. Deste modo, permitir que qualquer pessoa aprenda sobre história da música através da Internet. Informações, sobre o domínio proposto, são disponibilizadas para que, por exemplo, sistemas inteligentes apresentem questionamentos e dicas sobre o conteúdo.

A metodologia utilizada para a construção da ontologia é a metodologia descrita no guia *Ontology Development 101: A Guide to Creating Your First Ontology* (NOY & McGUINNESS, 2000).

Os termos definidos para serem utilizados na ontologia foram: período, compositor, instrumento musical, gênero musical, obra, tendência musical, vocal, instrumental, religiosa, profana, corda, friccionada, dedilhada, percutida, percussão, sopro, teclado.

A ontomúsica é composta por um relacionamento de classes, atributos e instâncias que descrevem os termos acima citados. As classes da ontologia descrevem determinados períodos da História da Música e a estes períodos relacionam-se compositores, que pertenceram a determinados períodos, aos quais ficam ligadas suas obras musicais, bem como os seus respectivos gêneros musicais. As obras possuem características conforme o período em que foram criadas.

5.1 Estrutura das classes

A hierarquia utilizada foi a de combinação, onde se definem primeiro os conceitos mais salientes e então é feita uma distribuição especifica dentro da hierarquia, tentando criar uma especialização aproximada de cada conceito (NOY & McGUINNESS, 2000).

A baixo é apresentada a estrutura das classes utilizada por Boff:

- Classe: Genero Musical;
    - Subclasse: vocal;
        - Subclasse: religiosa, profana;
    - Subclasse: instrumental;
- Classe: Periodo;
- Classe: Compositor;
- Classe: Instrumento Musical;
    - Subclasse: corda;
        - Subclasse: Friccionada, Dedilhada, Percutida;
    - Subclasse: Percussão;
        - Subclasse: Som Determinado, Som Indeterminado;
    - Subclasse: Sopro;
    - Subclasse: Teclado;
- Classe: Obra;
- Classe: Tendencia Musical.

A nomenclatura dada para essas classes, seguindo uma convenção, é iniciada com letras maiúsculas, e para os seus respectivos atributos as iniciais de seus nomes são escritas com letras minúsculas, ambos sem acentuação, espaços e caracteres especiais. Na seção seguinte são listados os atributos que fazem parte de cada classe.

5.2 Definição dos atributos das classes

Para cada classe utilizada na estrutura do ontomúsica foram selecionados os seguintes atributos:

- Compositor: nome, dataNascimento, dataFalecimento, cidadeNascimento, nomePai, nomeMae, vidaMusicaldoCompositor, pertenceaoPeriodo (atributo que se relaciona com a Classe Período) e o atributo inverso compos que se relaciona inversamente com a classe Obra;

- Periodo: nome, anoInicio, anoFim, característicasGerais, vidaMusical e o atributo inverso periodoContem, que se relaciona inversamente com a classe Genero Musical;

- Genero Musical: nome, descrição (descrição do gênero), pertenceAoPeriodo (que se relaciona com a classe Periodo) e o atributo inverso fazemParteDoGenero, que se relaciona inversamente com a classe Genero Musical. A classe Genero Musical tem como subclasse às classes: Vocal, que possui as subclasses Profana e Religiosa, e Instrumental. Como todas pertencem a superclasse Genero Musical, todas herdam os seus atributos.

- Obra: nome, caracteristicasDaObra, anoComposicao, foiCompostaPor (que se relaciona com a classe Compositor) e generoDaObra (que se relaciona com a classe Genero Musical);

- Instrumento Musical: nome e descricao. Contem as subclasses: Sopro, Teclado, Corda, que possui as subclasses Friccionada, Dedilhada e Percutida. Instrumento Musical contem ainda a subclasse Percussao que possui as subclasses Som Determinado e Som Indeterminado. Todas estas subclasses também herdam os atributos da classe Instrumento Musical;

- Tendencia Musical: nome, descricao e pertenceAoPeriodo, que se relaciona com a classe Periodo.

5.3 Estrutura dos atributos e base de conhecimento da ontomúsica

Nas duas seções anteriormente apresentadas, foram descritas as classes e citados seus respectivos atributos para a Ontomúsica. Após a conclusão dessa fase, foram feitas as restrições desses atributos, essas restrições são de cardinalidade e tipo. A cardinalidade do tipo múltipla foi utilizada, permitindo qualquer quantidade de valores em atributos do tipo *instance.* Os tipos utilizados foram *String* e *Instance*. Para exemplificar, Boff descreve a seguinte situação: uma classe chamada obra contendo um atributo chamado nome (nome de

uma música) do tipo String e um atributo chamado compositor (compôs as músicas) do tipo *Instance*.

O atributo Compositor pode ter vários valores relacionados à classe chamada compositor. Por causa disso é um atributo do tipo *Instance*.

Para as instâncias da ontologia foram utilizadas informações relacionadas com a história da música, a formação dessa base de conhecimento se deu através de dados retirados de páginas da web.

Após a construção da ontologia foi desenvolvido o Quiz Ontomúsica, com o objetivo de permitir que pessoas aprendam sobre a história da música através da internet, o qual é descrito no subcapítulo seguinte.

5.4 Quiz ontomúsica

O Quiz disponibiliza o conhecimento adquirido na forma de questionamentos e dicas baseadas na ontologia de música. São utilizadas folhas de estilo XSL para criar regras que possibilitem a inferência na busca de informações no OWL, essas informações são formatadas em HTML e juntamente com o Java Script geram uma interface web interativa.

A ontologia foi construída com o auxílio do software Protegé, onde são incluídas todas as classes, as subclasses, os atributos e as instâncias, bem como os relacionamentos entre atributos e classes. Nesta etapa também são populadas as instâncias com informações para compor a base de conhecimento da ontologia. Esse software possui um *plugin* que permite transformar a ontologia em um arquivo OWL.

Sobre esse arquivo OWL são utilizadas folhas de estilo XSL, que permitem a implementação de regras para inferir a busca de informações e formatá-las em HTML.

Juntamente com essa página HTML gerada, uma função Java Script que auxilia na criação das regras de inferência foi utilizada com a finalidade de empregar um caráter dinâmico ao conteúdo armazenado. São geradas perguntas do tipo <obra> x <autor> x <gênero>, disponibilizando dicas semânticas sobre o gênero da obra, como mostra a figura 5.1.

Figura 5.1 - Pergunta do Quiz Ontomúsica com uma dica, após escolha da alternativa errada.

Basicamente, existem comandos XSL[11] que procuram instâncias de obras no arquivo OWL, essas instâncias são armazenadas em *arrays* Java *script* da seguinte forma, Pergunta[InstA, Rel, InstB, DicaA] onde:

InstA: É o primeiro objeto do relacionamento, neste caso a obra;

Rel: Texto sobre o relacionamento;

InstB: Segundo objeto que se relaciona com InstA por meio de Rel, e este é inferido por meio do *template* Compositor;

DicaA: Dica ou informação textual sobre a InstA.

Todas as questões são criadas na função Java Script gerada, bem como as três opções de resposta. Para a exibição da questão e das alternativas é feito um sorteio e a pergunta escolhida é exibida ao usuário. Através da descrição feita sobre o ontomúsica, fica notável que os maiores esforços foram aplicados na parte de estruturação e construção da ontologia para a música e não em um sistema de inferência sobre a base ontológica. Desta maneira, nos próximos capítulos são apresentadas informações sobre lógica descritiva e inferência, assim como uma descrição sobre ferramentas utilizadas para o auxílio dessa descoberta de dados e checagem de consistência da estrutura da ontologia.

[11] http://www.w3.org/Style/XSL/. A XSL (Extensible Stylesheet Language) é uma linguagem para definir folhas de estilo personalizadas para o XML.

6 SISTEMAS DE INFERÊNCIA

A partir de informações representadas em uma estrutura formal, sistemas de inferência são capazes de extrair informações que não estejam explicitamente escritas na estrutura, da mesma forma que também testam a consistência de estruturas ontológica, desta forma, nesse capítulo serão descritas algumas dessas ferramentas.

6.1 Jena

Jena é uma API para a linguagem de programação Java, desenvolvida por Brian McBride da *Hewlett-Packard* (JENA, 2004; JENA, 2006b), ela permite o desenvolvimento de aplicações que manipulem ontologias.

Na sua primeira versão, a API Jena apenas possuía métodos para a manipulação de ontologias em DAML+OIL (Darpa Agent Markup Language + Ontology Inference Layer), porém na sua segunda versão foi adicionado um pacote específico para a manipulação de ontologias: API Jena 2 *Ontology*.

6.1.1 Estrutura geral

Nesta API uma determinada ontologia *é* transformada em um modelo abstrato de dados orientado a objetos, fazendo com que seus termos possam ser manipulados como objetos. Uma grande vantagem em se transformar ontologias em modelos orientados a objetos é que a programação orientada a objetos pode ser utilizada, fazendo com que as manipulações nestes modelos tornem-se comuns para os programadores Java, por exemplo.

Este modelo é baseado na linguagem em que a ontologia é escrita. Por exemplo, existem modelos específicos para *OWL*, *DAML+OIL*, *RDF*, entre outros. Isto porque o Jena dispõe de um *profile* para cada linguagem. Esses *profiles* contém os construtores permitidos e

as URIs das classes e propriedades das linguagens. No exemplo a baixo são mostrados exemplos de URIs para *object property*:

Linguagem DAML+OIL: *daml:ObjectProperty*

Linguagem OWL: *owl:ObjectProperty*

A arquitetura base da API Jena é composta por três módulos básicos para o processamento de ontologias. O primeiro módulo é o *Ontology Model*, este contém todas as classes necessárias para trabalhar com ontologias descritas em OWL, DAML, OIL ou RDFS. Neste módulo a classe mais relevante é a *OntModel* que representa um modelo ontológico, oferecendo suporte aos componentes de uma ontologia: classes, propriedades e instâncias.

Para a criação de modelos são utilizados os métodos disponíveis na classe *ModelFactory*:

createOntologyModel(): retorna um novo modelo para processar em memória as ontologias na linguaguem default (OWL Full).

```
01- OntModel modelo = ModelFactory.createOntologyModel();
```

Figura 6.1 – Criação do modelo ontológico

Esse método pode receber como parâmetro a URI a ser utilizada, esse parâmetro é utilizado para pesquisar os *profiles* das linguagens no *ProfileRegistry*, como demonstra o exemplo a baixo:

```
01- OntModel m = ModelFactory.createOntologyModel(ProfileRegistry.OWL_LANG);
```

Figura 6.2 – Criação do modelo ontológico com URI do *profile*

Na tabela 6.1 é exibida as constantes pré-definidas pelo Jena para cada *profile*.

Tabela 6.1 - Constantes para os *profiles* das linguagens no Jena.

Ontology Language	URI	Constant
OWL Lite	http://www.w3.org/TR/owl-features/#term_OWLLite	OWL_LITE_LANG
OWL DL	http://www.w3.org/TR/owl-features/#term_OWLDL	OWL_DL_LANG
OWL Full	http://www.w3.org/2002/07/owl#	OWL_LANG
DAML+ OIL	http://www.daml.org/2001/03/daml+oil#	DAML_LANG
RDFS	http://www.w3.org/2000/01/rdf-schema#	RDFS_LANG

A criação do modelo também pode ser iniciada com a utilização de uma especificação que permite um maior controle sobre o modelo:

createOntologyModel(*OntModelSpec* spec, *Model* base)

A especificação descreve os componentes do modelo, incluindo o esquema de armazenamento, a máquina de inferência e o *profile* da linguagem.

```
01- OntModel modelo=ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM,
    null);
```

Figura 6.3 – Criação do modelo ontológico utilizando especificações

As constantes para a configuração da especificação de um determinado modelo são as seguinte:

- Linguagem_MEM: especificação de um modelo que não utiliza nenhum *reasoner*;
- Linguagem_MEM_TRANS_INF: utiliza um *reasoner* transitivo, um componente de inferência simples que produz cláusulas transitivas a partir das hierarquias de classes e propriedades da ontologia;
- Linguagem_MEM_RULE_INF: utiliza um *reasoner* baseado em regras. Utiliza as regras do reasoner transitivo, com implicações adicionais do RDFS (Ex: domínio e imagem);
- Linguagem_MEM_RDFS_INF: utiliza um *reasoner* baseado em regras com um conjunto de regras semânticas adequadas à linguagem especificada. É mais completo para a linguagem OWL (porém restrito a um subconjunto de OWL próximo ao OWL Lite).

A palavra ‘Linguagem’ na descrição das constantes deve ser substituída por: DAML, OWL_LITE, OWL_DL, OWL ou RDFS.

No Jena *Ontology*, tem-se a classe *OntDocumentManager*, que provê serviços para a manipulação de documentos de ontologia, incluindo a importação de documentos. Nesse processo é feito um *cache* local dos documentos das URIs utilizadas, objetivando melhorar a performance da aplicação. O OntDocumentManager juntamente com o *OntModel* possibilitam o controle total do documento gerado. Segue sintaxe para criação de um modelo, utilizando essa classe:

```
01- OntDocumentManager gerenciador = new OntDocumentManager();
02- OntModelSpec spec = new OntModelSpec( OntModelSpec.OWL_MEM );
03- spec.setDocumentManager(gerenciador );
04- OntModel model = ModelFactory.createOntologyModel(spec, null );
```

Figura 6.4 – Criação completa do modelo ontológico

Para ler uma ontologia o Jena disponibiliza o método *read*(), que recebe como parâmetro a URI da ontologia:

```
01-model.read(http://www.rodrigo.goulart.nom.br/feevale/ontomusica/
              ontomusica.owl);
```

Figura 6.5 – Leitura da ontologia

Para escrever uma ontologia o Jena possui o método *write*(), que pode receber como parâmetro em qual formato o documento será escrito, caso não seja passado o formato, será assumido como *default* o RDF/XML.

```
01- model.write();
```

Figura 6.6 – Escrita do modelo ontológico

Listando as classes e suas propriedades em uma ontologia:

```
01- for ( ExtendedIterator i = m.listNamedClasses(); i.hasNext(); ) {
02-  OntClass classe = ( OntClass ) iterator.next();
03-  System.out.println( "Nome da Classe: "+ classe.getLocalName().toString() );
04-  for ( ExtendedIterator j=classe.listDeclaredProperties(); j.hasNext(); ) {
05-   OntProperty prop = (OntProperty) j.next();
06-   System.out.println("Nome Propriedade:"+prop.getLocalName().toString() );
07-  }
08- }
```

Figura 6.7 – Listagem de classes e propriedades

As classes são descritas na classe *OntClass*. Para obter uma classe deve-se usar simplesmente o método *getClass*(URI) do *OntModel*, ou usar o método *listClasses*() para obter uma lista de todas as classes da ontologia, como foi exemplificado no código acima.

A classe *OntClass* permite também obter todas as subclasses dessa classe através do método *listSubClasses*(). Como também foi mostrado no exemplo as propriedades são representadas pela classe *OntProperty*.

As instâncias das classes, também chamadas de indivíduos, são representadas pela classe *Instance*. A partir de uma classe, com a utilização do método *listInstances*(), é possível listar todas as suas instâncias. Existe um método parecido na Classe do modelo (*OntModel*) com o nome *listIndividuals*().

Uma lista de todos os indivíduos da ontologia pode ser obtida da seguinte maneira, por exemplo:

```
01- for(Iterator i = model.listIndividuals(); i.hasNext();) {
02-   System.out.println(((Individual)i.next()).toString());
03- }
```

Figura 6.8 – Listagem dos indivíduos do modelo ontológico

Listando todos os indivíduos da classe Obra:

```
01- OntClass Ontomusica;
02- Ontomusica =
             model.getOntClass("http://www.rodrigo.goulart.nom.br/feevale/
                                ontomusica/ontomusica.owl#Obra");
03- for(Iterator i= Ontomusica.listInstances();i.hasNext();) {
04-   System.out.println(((Individual)i.next()).toString());
05- }
```

Figura 6.9 – Listagem dos indivíduos de uma determinada classe do modelo ontológico

Outro módulo presente na arquitetura é o *Reasoner*. Este permite fazer inferências sobre o modelo. O uso das inferências sobre modelos semânticos é permitir obter informação adicional (inferida) sobre as ontologias.

Um exemplo de inferência é quando uma ontologia define *fatherOf* como uma *sub-property* de *parentOf*, desta maneira, quando se afirmar que 'John *fatherOf* Mary' é verdadeiro, também pode-se deduzir que 'John *parentOf* Mary' também é verdadeiro (JENA, 2004).

A figura 6.10 mostra a estrutura geral da máquina de inferência do Jena. A aplicação acessa a máquina de inferência usando o *ModelFactory*. Ele associa a base de conhecimento de uma ontologia com um *reasoner*, desta maneira, criando um novo modelo. Após a criação desse novo modelo, as consultas feitas retornarão como resultado os dados originais da ontologia e também as assertivas derivadas pelo *reasoner*, através das regras e das propriedades definidas no modelo ontológico.

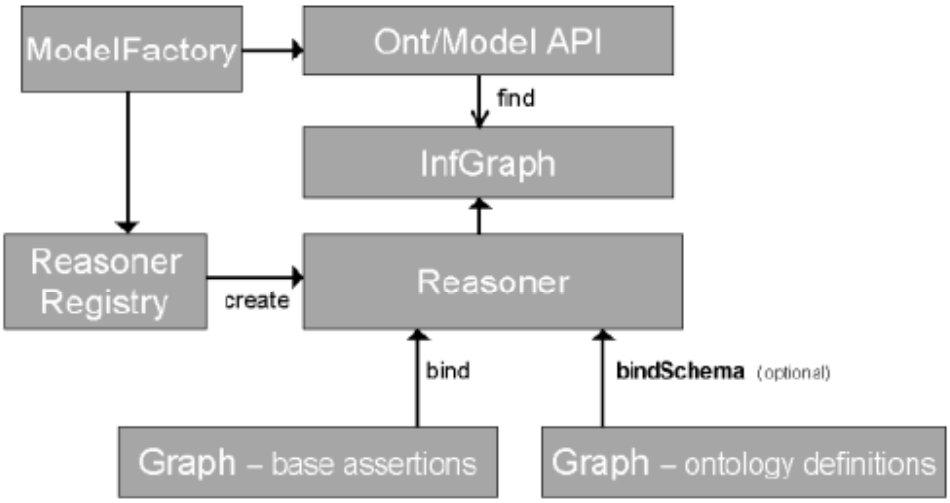

Figura 6.10 - Arquitetura geral da máquina de inferência do jena
Fonte: JENA (2006c)

O Jena também disponibiliza a interface *infModel,* que é uma extensão da interface *model*. A utilização dessa interface permite que sejam feitas operações de controle sobre o resultado da ligação entre o *reasoner* e os dados do modelo. O *reasoner* pode ser ligado a diferentes esquemas e instâncias, isso pode ser feito através dos métodos *bind* e *bindSchema*. Através desses métodos o *reasoner* pode ser ligado apenas a um conjunto de instâncias, através do método *bind*, ou somente ao esquema, utilizando-se o método *bindSchema.* A OWL não separa fortemente os dados e o esquema, então o Jena disponibiliza essa possibilidade de separação.

Os reasoners podem ser criados a partir do *ReasonerFactory* que por sua vez usa o *ReasonerRegistry* para encontrar o *Reasoner* apropriado. É no *ReasonerRegistry* que os motores de inferência do Jena podem ser escolhidos ou novos motores de inferência podem ser adicionados. A operação mais básica utilizada no *Reasoner* é o *validate*. Este método permite verificar se o modelo ontológico está de acordo com as regras de inferência especificadas no *Reasoner*.

6.1.2 Motores de inferência

O Jena fornece alguns motores de inferência pré-construídos e possibilita a criação de novos motores quando necessário, ou a possibilidade de estender os já existentes, abaixo é exibida uma breve descrição dos reasoners disponibilizados pelo Jena (JENA, 2006c).

Reasoners disponíveis:

- *Transitive reasoner*: Disponibiliza suporte para armazenar e percorrer classes e propriedades ligadas. Implementa apenas as propriedades transitivas e simétricas de rdfs:*subPropertyOf* e de rdfs:*subClassOf*.

- RDFS *rule reasoner*: Implementa um subconjunto configurável das implicações RDFS

- OWL, OWL *Mini*, OWL *Micro Reasoners*: Implementação incompleta da linguagem OWL-Lite

- DAML *micro reasoner* :Usada internamente para viabilizar o uso da API legada de DAML, fornece uma capacidade mínima de inferência.

- *Generic rule reasoner*: Raciocinador baseado em regras que suporta a criação de regras definidas pelo usuário. Suporta encadeamento para frente (*forward chaining*), encadeamento para trás (*tabled backward chaining*) e estratégias de execução híbridas.

6.1.3 Generic Rule Reasoner

O *Generic Rule Reasoner* (GRR) é o mais independente dos motores de inferência do Jena, por isso terá uma descrição mais detalhada. Ele foi utilizado tanto para implementar o *reasoner* RDFS quanto o OWL, mas também possibilita que o programador importe as regras dos outros *reasoners* existentes. Desta maneira ele se torna o *reasoner* mais abrangente do Jena.

Uma regra é definida como uma instância da classe *Rule* que contém uma lista de premissas e uma lista de conclusões sobre as mesmas, opcionalmente a regra definida possui um nome e um sentido. Uma premissa ou uma conclusão pode ser uma tripla, uma tripla estendida ou uma chamada a procedimento primitivo (chamado *builtin*).

O Jena também disponibiliza um *parser* para checar a legalidade das regras definidas seguindo a sintaxe original, mas também permite que um outro *parser* seja definido pelo usuário para se obter um melhor diagnóstico dos erros encontrados, já que o *parser* disponibilizado pelo Jena não se demonstra muito eficiente nesse diagnóstico (JENA, 2006c).

Como mencionado anteriormente, o GRR possui suporte a três tipos de mecanismos de ativação de regras:

Forward Chaining Engine (Para frente): No momento em que o modelo de inferência recebe a primeira consulta, todos os dados relevantes do modelo são enviados para o mecanismo de regras. Quando uma regra causa a criação de triplas extras, novas regras podem ser disparadas. Nesse momento, se as regras não forem bem definidas, pode acontecer um *loop* infinito (JENA, 2006c). Cada vez que são criadas ou removidas triplas do modelo pelos próprios métodos da API, as regras podem ser ativadas. A inferência acaba quando as regras pararem de ser ativadas. O algoritmo utilizado por este motor de inferência trabalha de forma incremental.

Backward Chaining Engine (Para trás): No modo para trás o *reasoner* usa uma estratégia de execução parecida com o mecanismo do Prolog. No momento em que o modelo de inferência recebe uma consulta, ele a transforma em um objetivo, e o motor de inferência aplica as regras de modo a tentar atingir esse objetivo, unindo as triplas armazenadas com as regras de *backward chaining*. Neste caso, o motor de inferência não trabalha incrementalmente, isto é, sempre que os dados originais forem alterados, todo o processamento realizado é perdido.

Hybrid (híbrido): Utiliza os dois mecanismos acima de forma conjunta. O mecanismo para frente executa primeiro e guarda um conjunto de deduções. Caso uma regra para frente crie novas regras para trás, ela vai instancia-la de acordo com as variáveis guardadas nas deduções e depois irá passar as regras instanciadas para o mecanismo LP para trás. Todas as consultas são resolvidas posteriormente pelo LP *engine* usando a mistura dos dados brutos e das deduções que vieram do mecanismo para frente.

6.1.4 Jena e mecanismos de inferência adicionais

Outros motores de inferência podem ser utilizados com o Jena, basta que esse mecanismo implemente o padrão DIG (*Description Logic Interface*), o qual visa padronizar a maneira de interação das ferramentas clientes com os diferentes reasoners existentes (DICKINSON, 2004). No próprio site de documentação do jena é aconselhado o uso de um

reasoner externo mais completo (JENA, 2006c), dentre esses sistemas de inferência, que seguem o padrão DIG, estão o FaCT++[12] e o Pellet[13].

6.1.5 Experimentos com a utilização do jena e a ontomúsica

Os dois principais objetivos da arquitetura do Jena são (WILKINSON et al., 2003):

- Possibilitar ao programador da aplicação representações flexíveis e múltiplas dos grafos RDF. Dessa forma, facilita o acesso e a manipulação dos dados nos grafos, possibilitando ao programador da aplicação navegar nas estruturas de triplas.

- Tornar simples a visão do grafo RDF ao programador do sistema que deseja expor seus dados como triplas.

Uma visão simplificada da arquitetura do Jena é exibida da figura a baixo.

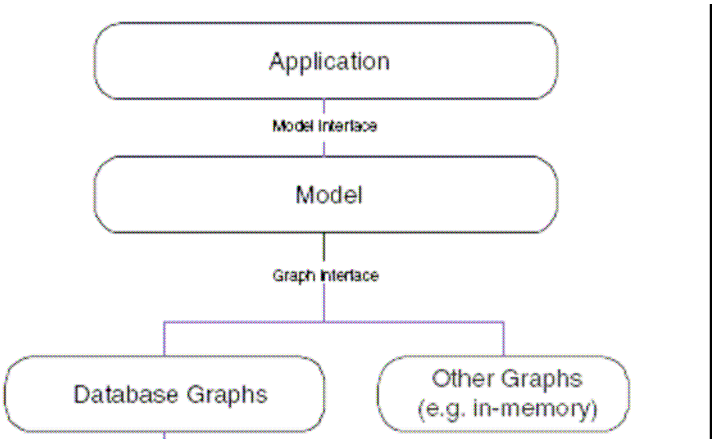

Figura 6.11 – Visão da arquitetura do Jena, adaptado de (WILKINSON et al., 2003)

As aplicações desenvolvidas com o Jena, tipicamente interagem com um modelo abstrato, que traduz operações de alto-nível em operações de baixo-nível sobre triplas armazenadas em um determinado grafo RDF.

[12] http://owl.man.ac.uk/factplusplus/
[13] http://www.mindswap.org/2003/pellet/index.shtml

Nas próximas figuras serão exibidos trechos de códigos utilizando o framework Jena para a manipulação das informações estruturadas na ontologia ontomúsica, a qual foi descrita no capítulo 5.

```
1 - String baseURI = "file:./owl/ontomusica.owl";
2 - String nameSpace =
            "http://www.rodrigo.goulart.nom.br/feevale/ontomusica/#";
3 - OntDocumentManager gerenciador = new OntDocumentManager();
4 - OntModelSpec spec = new OntModelSpec(OntModelSpec.OWL_MEM );
5 - spec.setDocumentManager(gerenciador );
6 - OntModel model = ModelFactory.createOntologyModel(spec, null );
7 - model.read(baseURI);
```

Figura 6.12– Criação do modelo para a ontologia ontomúsica.

Nas duas primeiras linhas foram definidas a localização do arquivo OWL e a definição do namespace da ontologia, respectivamente. Nas linhas 3, 4, 5 e 6, são criados os objetos que possibilitam o controle sobre o documento gerado. Na linha número 7 é feita a leitura do documento OWL. Dessa forma, o modelo orientado a objetos para a manipulação da estrutura OWL foi criado sem o uso de um sistema de inferência, definição que foi feita na linha 4, através do parâmetro *OWL_MEM*.

Após a criação do modelo que representa a estrutura da ontomúsica, na figura 6.13, serão listadas as classes que compõem a ontologia.

```
1 - for(ExtendedIterator i = model.listNamedClasses(); i.hasNext();) {
2 -    OntClass classe = ( OntClass ) i.next();
3 -   System.out.println( "Nome da Classe: " +
                  classe.getLocalName().toString() );
4 - }
```

Figura 6.13 – Listagem das classes da ontomusica.

Na primeira linha é iniciado o processo para percorrer todas as classes encontradas do ontomúsica, o resultado desse trecho de código é exibido abaixo:

```
Nome da Classe: Corda
Nome da Classe: Percussao
Nome da Classe: SomIndeterminado
Nome da Classe: Obra
Nome da Classe: Instrumental
Nome da Classe: Teclado
Nome da Classe: InstrumentoMusical
Nome da Classe: Dedilhada
Nome da Classe: Periodo
Nome da Classe: Percutida
```

```
Nome da Classe: Vocal
Nome da Classe: Profana
Nome da Classe: Compositor
Nome da Classe: GeneroMusical
Nome da Classe: Sopro
Nome da Classe: TendenciaMusical
Nome da Classe: SomDeterminado
Nome da Classe: Friccionada
Nome da Classe: Religiosa
```

Adaptando a listagem da figura número 6.13, pode-se listar uma estrutura das classes e de suas respectivas subclasses, conforme demonstra a figura 6.14.

```
 1 - for(ExtendedIterator i = model.listNamedClasses(); i.hasNext(); ) {
 2 -        OntClass superClasse = ( OntClass ) i.next();
 3 -        if (!superClasse.hasSuperClass()) {
 4 -              System.out.println( "Nome da Classe: " +
                                  superClasse.getLocalName().toString() );
 5 -              System.out.println(superClasse.getComment(null));
 6 -              imprimeSubClasses(superClasse);
 7 -        }
 8 - }

 9 - private void imprimeSubClasses (OntClass classe) {
10 -        if (classe.listSubClasses().toList().size() > 0) {
11 -              System.out.println("    SubClasses: ");
12 -        }
13 -        for (Iterator k = classe.listSubClasses(); k.hasNext(); ) {
14 -              OntClass subClasse = (OntClass) k.next();
15 -              System.out.print(subClasse.getLocalName() + " - " );
16 -              imprimeSubClasses(subClasse);
17 -        }
18 - }
```

Figura 6.14 – Listagens de classes e subclasses da ontomusica.

Na linha número 3 é verificado se a classe em questão possui alguma classe pai, caso não possua, a linha 6 chama o método ‘imprimeSubclasses’, declarado na linha 9 para a exibição das subclasses. Dessa forma são listadas todas as superclasses com suas respectivas subclasses, na linha 5 é exibido o comentário de cada uma das super classes. O resultado dessa listagem é exibido a baixo:

```
Nome da Classe: Obra
Nome da Classe: InstrumentoMusical
      SubClasses:
        Percussao -
          SubClasses:
           SomIndeterminado -
           SomDeterminado
        Teclado -
        Corda -
          SubClasses:
```

```
                Friccionada -
                Percutida -
                Dedilhada
            Sopro
Nome da Classe: Periodo
                Descreve o período em que se encontra
                determinada obra ou compositor.
Nome da Classe: Compositor
      Relação dos compositores.
Nome da Classe: GeneroMusical
      Descreve o tipo de música.
      SubClasses:
        Vocal -
          SubClasses:
            Religiosa -
            Profana
        Instrumental
Nome da Classe: TendenciaMusical
```

No Jena as propriedades são representadas pela classe *OntProperty*. Existem dois tipos básicos de propriedade: *Datatype Properties* e *Object Properties*. A ontomusica tem esses dois tipos de propriedades em sua estrutura.

```
1 - for (ExtendedIterator iProperty = model.listObjectProperties();
          iProperty.hasNext();)
    {
2 -         OntProperty prop = (OntProperty)iProperty.next();
3 -         System.out.println( "   Nome da Propriedade:" +
                                prop.getLocalName().toString() );
4 - }
```

Figura 6.15 – Listagem de propriedades da ontomusica

Na linha número 1 da figura 6.15, através do método *listObjectProperties()*, são listadas todas as *object properties* do modelo, as quais representam o relacionamento entre classes, ou seja, tem como valor a instância de outra classe. O resultado dessa listagem é mostrado a baixo:

```
Nome da Propriedade:generoMusical
Nome da Propriedade:compos
Nome da Propriedade:foiCompostaPor
Nome da Propriedade:fazemPartedoGenero
Nome da Propriedade:generoDaObra
Nome da Propriedade:periodoContem
Nome da Propriedade:pertenceAoPeriodo
```

Para listar as *datatype properties* declaradas no modelo basta substituir o método *listObjectProperties(),* da linha número 1 da figura 6.15, por *listDatatypeProperties()*, esse tipo restringe a propriedade a se relacionar com apenas um tipo de dado específico. Ela pode

assumir alguns valores como: *integer*, *string*, *float* e *boolean*. A listagem das *datatype properties* da ontomusica é mostrada a baixo:

```
Nome da Propriedade:nomePai
Nome da Propriedade:vidaMusical
Nome da Propriedade:dataFalecimento
Nome da Propriedade:anoInicio
Nome da Propriedade:descricao
Nome da Propriedade:cidadeNascimento
Nome da Propriedade:caracteristicasDaObra
Nome da Propriedade:vidaMusicalDoCompositor
Nome da Propriedade:dataNascimento
Nome da Propriedade:nomeMae
Nome da Propriedade:anoComposicao
Nome da Propriedade:anoFim
Nome da Propriedade:nome
Nome da Propriedade:caracteristicasGerais
```

Além desses dois tipos de propriedades que estão presentes na estrutura da ontomúsica, também se pode listar os seguintes outros tipos:

Tabela 6.2 – Métodos para listagem das propriedades de uma ontologia

Propriedade	**Método no Jena**	**Descrição**
Transitive Property	`listTransitiveProperties()`	Uma propriedade é transitiva quando se tem um objeto A que está contido em um objeto B e B está contido em C, então A também está contido em C.
`Symetric Property`	`listSymmetricProperties()`	Uma propriedade simétrica significa que quando um objeto denominado 'A' possui um relacionamento com um objeto denominado 'B', por meio de uma propriedade 'P', pode-se dizer então que o objeto 'B' também se relaciona com o objeto 'A' por meio desta mesma propriedade.
`FUNCTIONAL PROPERTY`	`listFunctionalProperties()`	Essa função determina e garante que uma propriedade possua apenas um valor. Se a ontologia ontomusica tem a propriedade 'tem obra', então um 'Autor A' pode ser associada somente com uma única obra.
`INVERSE FUNCTIONAL PROPERTY`	`listInverseFunctionalProperties()`	Define que uma propriedade representa somente um único indivíduo, ou seja, o valor

		determinado a essa propriedade representa apenas aquela instância. Por exemplo, considerando que existe a propriedade 'numeroIdentificador', e que para cada autor existe um número diferente, então essa propriedade seria uma InverseFunctionalProperty.

No próximo trecho de código, é exemplificado como detalhar os indivíduos de uma determinada classe da ontologia.

```
 1 - OntClass classe;
 2 - classe = model.getOntClass(nameSpace + "Teclado");
 3 - System.out.println("Classe: " + classe);
 4 - for(Iterator i = classe.listInstances(); i.hasNext();) {
 5 -        Individual individuo =  (Individual)i.next();
 6 -        individuo.listProperties();
 7 -        System.out.println("-------------------------------------");
 8 -        System.out.println("Individuo: " + (individuo.toString()));
 9 -        this.detalhar(individuo);
10 - }

11 - private void detalhar(Individual individuo) {
12 -        System.out.println("*********************************");
13 -        System.out.println("   INDIVIDUO-PREDICADO-OBJETO    ");
14 -        System.out.println("*********************************");
15 -        for(ExtendedIterator i = individuo.listProperties();
                i.hasNext(); )
            {
16 -              Object objPropriedade = i.next();
17 -              StatementImpl sentenca = (StatementImpl)
                                              objPropriedade;
18 -              Property predicado = sentenca.getPredicate();
19 -              Resource recurso = null;
20 -              try {
21 -                    recurso = sentenca.getResource();
22 -              } catch (Exception e) { }
23 -              System.out.println("predicado: " +
                                    predicado.getLocalName());
24 -              if(recurso != null) {
25 -                    System.out.println("Recurso: " +
                                        recurso.getLocalName());
26 -              } else {
27 -                    try {
28 -                       System.out.println("Objeto: " +
                          sentenca.getLiteral().getValue().toString());
29 -                    } catch (Exception e) { }
30 -              }
31 -        }
32 - }
```

Figura 6.16 – Listagem dos indivíduos de uma determinada classe da ontomusica

Na linha de número 2 do trecho de código exibido na figura 6.16 a cima, é criado um objeto para a classe 'Teclado', na linha 4 são listados os seus indivíduos, e para cada indivíduo é chamado o método *detalhar()*, na linha 9. Esse método extrai as triplas RDF da estrutura da ontomusica, no Jena as triplas RDF são representadas por objetos *Statement*. A partir do indivíduo, na linha 15, são listadas as suas propriedades, para cada propriedade é selecionado o seu predicado, conforme a linha 18. Para completar a tripla RDF é listado um recurso, para o caso de *object property*, ou é listado o valor literal do objeto, para o caso de uma *datatype property*. Abaixo é exibido o resultado para esse trecho de código:

```
Classe: http://www.rodrigo.goulart.nom.br/feevale/ontomusica/#Teclado
-----------------------------------------
Individuo:
http://www.rodrigo.goulart.nom.br/feevale/ontomusica/#MusicaOnto_Instanc
e_60003

**************************************************************************
                          INDIVIDUO-PREDICADO-OBJETO
**************************************************************************
predicado: nome
Objeto: Clavicordio
predicado: type
Recurso: Teclado
```

Nesse caso, foram listadas duas triplas RDF:

Tripla 1:
- Instancia: #MusicaOnto_Instance_60003
- Predicado: nome
- Objeto: Clavicordio

Tripla 2:
- Instancia: #MusicaOnto_Instance_60003
- Predicado: `type`
- Recurso: `Teclado`

A tripla número 1 é formada por uma *dataType property*, tendo um tipo *string* como valor para o Objeto. A tripla número 2 é formada por uma *object property*, que relacionada a instância com o recurso 'Teclado', o qual é detalhado abaixo:

```
predicado: subClassOf
Recurso: InstrumentoMusical
predicado: type
Recurso: Class
```

6.1.5.1 Sistema de Exemplo - Central de estudos

O objetivo da central de estudos é exemplificar a utilização do Jena para extração das informações da ontologia ontomusica (BOFF, 2005). Os trechos de códigos explicados na subseção anterior foram utilizados para a exibição das informações das próximas figuras.

A central de estudos foi desenvolvida com a utilização do framework Java Server Faces[14] sobre o ambiente de desenvolvimento Eclipse Exadel 4.0.3[15]. Esse sistema de exemplo possibilita a visualização de todo o conhecimento conceitualizado na ontomúsica, para isso exibe os dados na forma de triplas RDF. Toda a informação contida em um documento RDF é representada por coleções de triplas, onde cada tripla é formada por um sujeito, um predicado e um objeto (BERNERS-LEE, 2002).

A central de estudos tem um filtro para as classes, o qual exibe todas as classes da ontomúsica, após a escolha da classe são listadas as suas instâncias, e ao se escolher uma determinada instância, são listadas suas propriedades e valores, montando assim as triplas RDF. A seguir são exibidas as telas com essas informações.

[14] Java.sun.com/javaee/javaserverfaces/
[15] www.exadel.com/web/portal

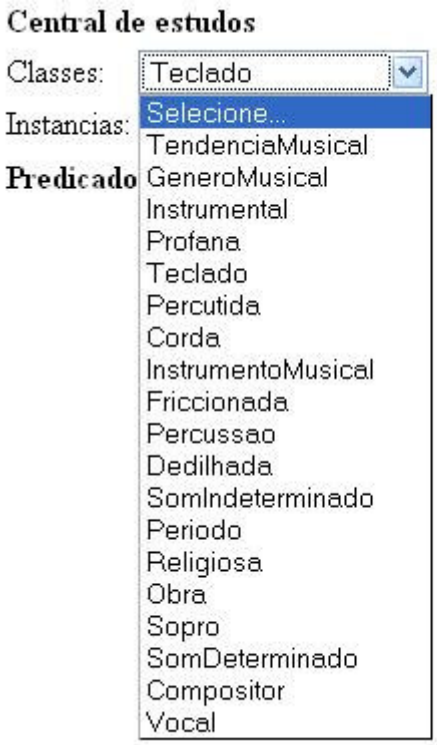

Figura 6.17 – Central de Estudos: listagem das classes da ontomúsica

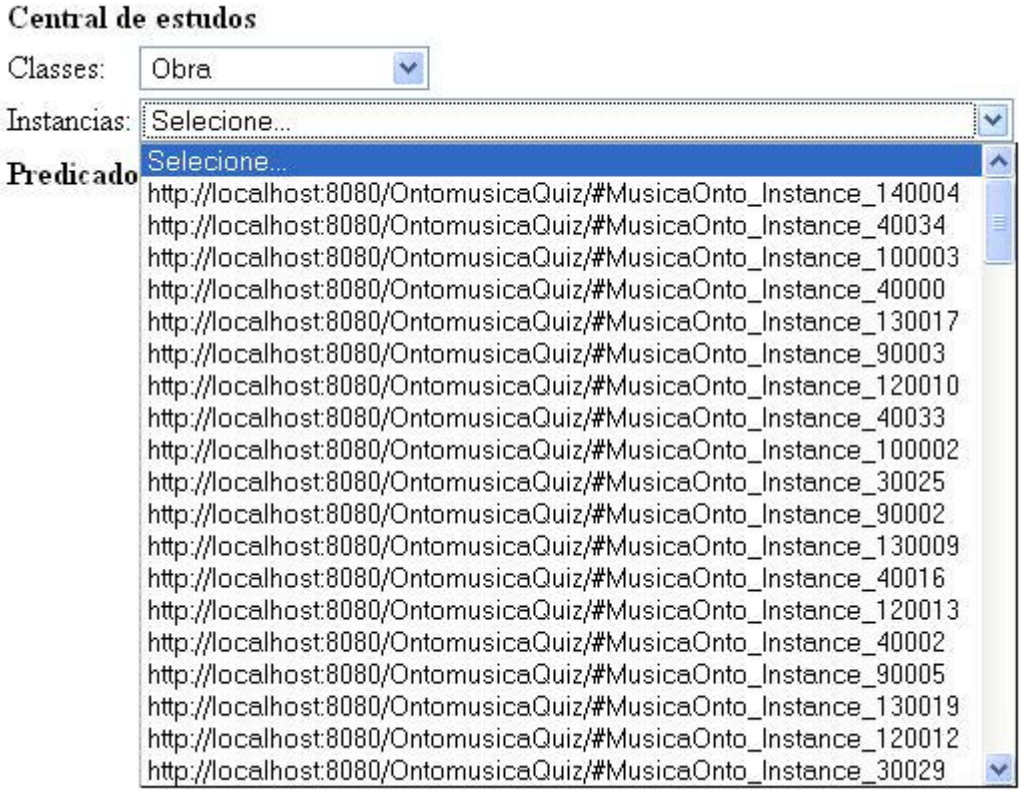

Figura 6.18 – Central de Estudos: listagem das instâncias da classe Obra

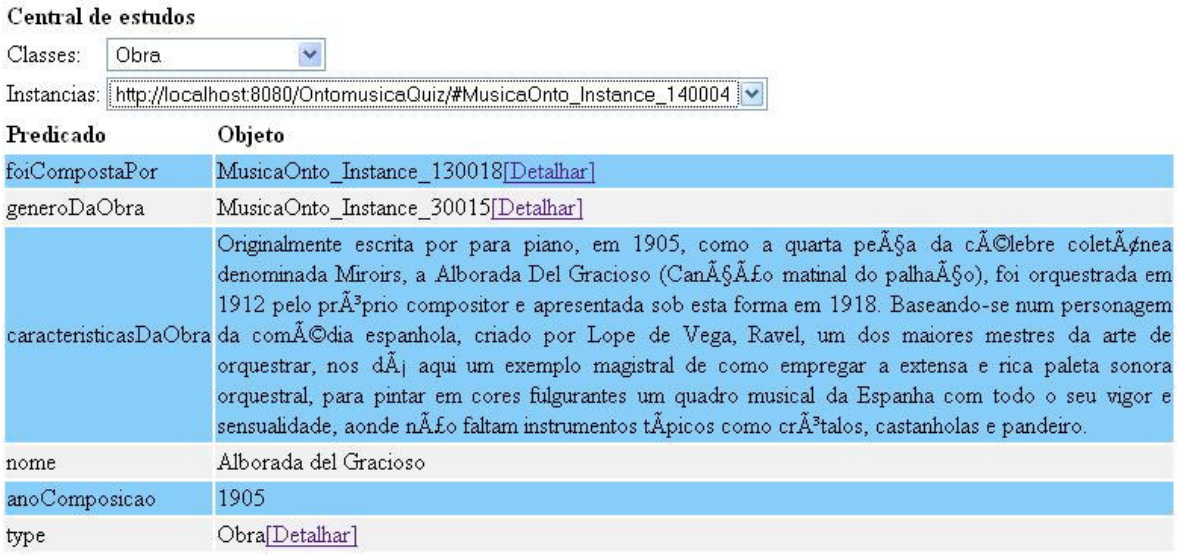

Figura 6.19 – Central de Estudos: listagem das propriedades da instância, formando as triplas Instancia-Predicado-Objeto.

6.2 FaCT ++ (Fast Classification of Terminologies)

FaCT++ (FaCT, 2003) é uma nova geração do *reasoner* FaCT OWL-DL, que é um classificador de lógica de descrição que também pode ser utilizado para testes de satisfação de modelos lógicos, desenvolvido em *Common Lisp*. Ele continua utilizando o algoritmo do FaCT, mas com uma nova estrutura. Foi implementado utilizando a linguagem C++, essa linguagem foi utilizada para deixar a ferramenta mais eficiente e para aumentar a portabilidade (HORROCKS, 2004). É um sistema cujo código fonte está disponível (licença pública geral GNU) e pode ser obtido gratuitamente a partir da URL http://wonderweb.semanticweb.org/deliverables/D14.shtml.

FaCT++ pode ser utilizado para as seguintes atividades(HORROCKS, 2004):

- Checar a consistência da ontologia;
- Checar a satisfabilidade dos conceitos de forma individual e também de grupos de conceitos;
- Checar a relação de submissão entre dois conceitos;
- Classificação da ontologia (criando uma taxonomia).

FaCT++ disponibiliza um bom controle de raciocínio para a lógica de descrição SHOIQ, que é uma extensão da linguagem descritiva AL, citada no capítulo 3. A linguagem de ontologia OWL DL é baseada na lógica descritiva SHOIQ (HORROCKS, 2005).

Abaixo é exibida uma lista com a avaliação das versões disponíveis da ferramenta:

- 05 de abril de 2006: FaCT++ v1.1.3 released. A versão do padrão DIG ficou mais estável.
- 08 de março de 2006: FaCT++ v1.1.2 released. Versão com correção de erros.
- 20 de fevereiro de 2006: FaCT++ v1.1.1 released. Versão cerca de 10% mais rápida que a anterior.
- 16 de janeiro de 2006: FaCT++ v1.1.0 released. Adicionado suporte *datatypes,* incluindo Inteiro e *String*.
- 17 de novembro de 2005: FaCT++ v1.0.0 released. Suporte a r*easoning* sobre a lógica descritiva SHOIQ.

6.3 Pellet

Pellet (PELLET, 2006) é um *reasoner open-source* baseado em código Java para a linguagem OWL-DL e disponibiliza controle de raciocínio para a lógica de descrição SHOIQ. Essa ferramenta também é baseada nos algoritmos Tableaux para lógicas de descrição expressivas e pode ser utilizada em conjunto com o Jena, descrito anteriormente.

Através dessa ferramenta pode-se utilizar os principais serviços de inferência para lógicas de descrição anteriormente citados, como: checagem de consistência, satisfabilidade, classificação e realização.

O Pellet também disponibiliza um mecanismo para execução de *queryes* sobre a base ABox do conhecimento. Para isso ele utiliza a linguagem de *query* SPARQL (SPARQL, 2006). Essa linguagem é utilizada para a consulta de dados para o padrão RDF, no qual a linguagem OWL é baseada.

Abaixo é apresentado um exemplo de uma *query* simples:

Dados:

```
<http://example.org/book/book1>  <http://purl.org/dc/elements/1.1/title>  “SPARQL Tutorial”  .
```

Query:

```
SELECT ?title
WHERE
{
<http://example.org/book/book1> <http://purl.org/dc/elements/1.1/title> ?title .
}
```

Esta *query* executada sobre os dados acima retorna o seguinte resultado:

title
"SPARQL Tutorial"

6.3.1 Experimentos com o sistema de inferência Pellet

Como descrito anteriormente, o Pellet pode ser utilizado em conjunto com o Jena, a figura 6.20 exibe o código que implementa essa funcionalidade.

```
OntModel       model       =       ModelFactory.createOntologyModel(
                    PelletReasonerFactory.THE_SPEC );

// cria o modelo para a ontologia
System.out.println( "Lendo..." );
model.read("http://localhost:8080/ontomusica/ontomusica.owl" );
System.out.println( "concluído" );

// prepara o modelo para o reasoner
System.out.println( "Preparando..." );
model.prepare();
System.out.println( "concluído" );

PelletInfGraph graph = (PelletInfGraph) model.getGraph();
KnowledgeBase base = graph.getKB();
```

Figura 6.20 – Adicionando reasoner Pellet ao Jena

No capítulo sobre lógica de descrição, foram descritos os serviços de inferência mais comumente utilizados. O pellet disponibiliza os mais tradicionais, os quais são detalhados a baixo:

Satisfabilidade (Satisfiability): serviço tem a finalidade de checar se uma descrição de conceito nunca pode ter instâncias por causa das inconsistências ou contradições do

modelo. No trecho de código a baixo é checada a satisfabilidade para todos os conceitos da ontomusica.

```
Set classes = base.getClasses();
Iterator i = classes.iterator();
StringBuffer satisfabilidade = new StringBuffer();
while (i.hasNext()) {
  ATermAppl classe = (ATermAppl) i.next();
       satisfabilidade.append("Classe: " + classe.getName());
       satisfabilidade.append("\n");
       satisfabilidade.append("     isSatisfiable: ");
       if (base.isSatisfiable(classe)) {
             satisfabilidade.append("Sim");
       } else {
             satisfabilidade.append("Não");
       }
       satisfabilidade.append("\n");
}
```

Figura 6.21 – Verificação da satisfabilidade de conceitos

Parte do resultado exibido pelo código acima é exibido a baixo:

```
Classe: http://localhost:8080/OntomusicaQuiz/#Percussao
      isSatisfiable: Sim
Classe: http://localhost:8080/OntomusicaQuiz/#GeneroMusical
      isSatisfiable: Sim
```

Equivalência: Dois conceitos são equivalentes quando possuem as mesmas instâncias. Na figura a seguir é exibido o código que verifica quais classes são equivalentes dentro da estrutura da ontologia.

```
Set classes = base.getClasses();
Iterator i = classes.iterator();
StringBuffer equivalente = new StringBuffer();
while (i.hasNext()) {
  ATermAppl classe = (ATermAppl) i.next();
       equivalente.append("Classe: " + classe.getName());
       equivalente.append("\n");
       equivalente.append("\n");
       equivalente.append("    Classes Equivalentes: \n");
       Iterator iEquivalente =
                  base.getAllEquivalentClasses(classe).iterator();
       while(iEquivalente.hasNext()) {
             ATermAppl classeEqui = (ATermAppl)
                                    iEquivalente.next();
             equivalente.append("    Classe: " +
                                    classeEqui.getName());
             equivalente.append("\n");
       }
       equivalente.append("\n");
  }
```

Figura 6.22 – Verificação da equivalência entre conceitos

Parte do resultado obtido com a execução do código a cima é exibido a baixo:

```
Classe: http://localhost:8080/OntomusicaQuiz/#Percussao
  Classes Equivalentes:
```

Classe: http://localhost:8080/OntomusicaQuiz/#Percussao

```
Classe: http://localhost:8080/OntomusicaQuiz/#GeneroMusical
  Classes Equivalentes:
  Classe: http://localhost:8080/OntomusicaQuiz/#GeneroMusical
```

Classificação (*Classification*): esse serviço computa as relações de subclasses entre cada classe, objetivando criar a hierarquia de classes da ontologia. Essa hierarquia pode ser utilizada para se obter as subclasses diretas de um outro determinado conceito.

A linha de código para a realização da classificação da ontologia é a seguinte:

```
((PelletInfGraph) model.getGraph()).getKB().classify();
```

O Pellet também pode ser utilizado *online*[16], isso é feito através da submissão da URI da ontologia ou do texto que a codifica. Ao submeter a URI da ontomusica para checar a classificação, foi obtido o seguinte resultado:

[16] http://www.mindswap.org/2003/pellet/demo.shtml

Input file: http://201.41.10.17:8080/ontomusicaJsf/owl/ontomusica.owl
OWL Species: DL
DL Expressivity: ALI(D)
Consistent: Yes
Time: 46056 ms (Loading: 44114 Species Validation: 1795 Consistency: 84 Classification: 62)

Classification:

- owl:Thing
  - ontomusicaQuiz:GeneroMusical
    - ontomusicaQuiz:Instrumental
    - ontomusicaQuiz:Vocal
      - ontomusicaQuiz:Religiosa
      - ontomusicaQuiz:Profana
  - ontomusicaQuiz:TendenciaMusical
  - ontomusicaQuiz:InstrumentoMusical
    - ontomusicaQuiz:Teclado
    - ontomusicaQuiz:Percussao
      - ontomusicaQuiz:SomIndeterminado
      - ontomusicaQuiz:SomDeterminado
    - ontomusicaQuiz:Corda
      - ontomusicaQuiz:Dedilhada
      - ontomusicaQuiz:Friccionada
      - ontomusicaQuiz:Percutida
    - ontomusicaQuiz:Sopro
  - ontomusicaQuiz:Compositor
  - ontomusicaQuiz:Obra
  - ontomusicaQuiz:Periodo

Figura 6.23 – Resultado do serviço de classificação sobre a ontomusica

Realização (*Realization*): esse serviço encontra as classes mais específicas que um indivíduo pertence, ou seja, computa os tipos diretos para cada um dos indivíduos. O serviço de realização deve ser executado após o de classificação, já que os tipos diretos são definidos de acordo com a uma hierarquia de classe. Através dessa hierarquia, também é possível obter todos os tipos para um determinado indivíduo.

No trecho de código abaixo é exemplificado como a realização é feita através do pellet:

```
((PelletInfGraph) model.getGraph()).getKB().classify();
 base.realize();
((PelletInfGraph) model.getGraph()).getKB().printClassTree();
```

Figura 6.24 – Verificação da realização da ontologia

A baixo é exibia a árvore da hierarquia de classes e seus indivíduos, a qual foi gerada através da execução do código da figura 6.24. O pellet mostra o resultado com todos os indivíduos de cada classe, porém, no exemplo abaixo é exibida apenas uma instância para as classes que possuem dois ou mais indivíduos:

```
owl:Thing
    ontomusicaQuiz:GeneroMusical
       ontomusicaQuiz:Instrumental - (MusicaOnto_Instance_50039...)
       ontomusicaQuiz:Vocal
          ontomusicaQuiz:Religiosa - (MusicaOnto_Instance_5...)
          ontomusicaQuiz:Profana - (MusicaOnto_Instance_20008...)
    ontomusicaQuiz:TendenciaMusical - (MusicaOnto_Instance_90028...)
    ontomusicaQuiz:InstrumentoMusical
       ontomusicaQuiz:Teclado - (MusicaOnto_Instance_60003)
       ontomusicaQuiz:Percussao
    ontomusicaQuiz:SomIndeterminado -(MusicaOnto_Instance_150014...)
          ontomusicaQuiz:SomDeterminado - (MusicaOnto_Instance_50025...)
       ontomusicaQuiz:Corda
          ontomusicaQuiz:Dedilhada - (MusicaOnto_Instance_50005...)
          ontomusicaQuiz:Friccionada - (MusicaOnto_Instance_10001...)
          ontomusicaQuiz:Percutida - (MusicaOnto_Instance_50031...)
       ontomusicaQuiz:Sopro - (MusicaOnto_Instance_50034...)
    ontomusicaQuiz:Compositor - (MusicaOnto_Instance_10015...)
    ontomusicaQuiz:Obra - (MusicaOnto_Instance_40001...)
    ontomusicaQuiz:Periodo - (MusicaOnto_Instance_70000...)
```

Figura 6.25 – Resultado do serviço de realização da ontomusica

Consistência (*Consistency*): verifica a ontologia para assegurar-se da inexistência de conceitos contraditórios. O documento que descreve a OWL fornece a definição formal da consistência de uma ontologia, a qual é utilizada pelo pellet (PELLET, 2006). Por exemplo, um construtor da OWL é o *disjointWith,* ele indica que duas classes são contrárias uma a outra, por exemplo as classes 'Homem' e 'Mulher'. Através desse construtor uma inconsistência sobre os indivíduos da ontologia pode ser indicada. Dessa maneira o Pellet poderá inferir que um indivíduo pertencente ao conceito 'Homem' não poderá pertencer ao conceito 'Mulher'.

No próximo trecho de código é exemplificado como o pellet checa se a base de conhecimento é consistente, e através do método *getExplanation* exibe as informações sobre alguma inconsistência encontrada.

```
System.out.println("A base é consistente?");
if (base.isConsistent()) {
      System.out.println("    Sim");
} else {
      System.out.println("    Não");
}
 System.out.println(((PelletInfGraph)
                   model.getGraph()).getKB().getExplanation());
```

Figura 6.26 – Verificação da consistência da ontologia

Resultado da execução desse trecho de código é o seguinte:

```
A base é consistente?
      Sim
No inconsistency was found! There is no explanation generated.
```

De acordo com (PATEL-SCHNEIDER, HAYES e HORROCKS, 2004), uma ontologia é consistente se houver uma interpretação que satisfaça a todos os fatos e axiomas da mesma. Seguindo a definição dos mesmos autores citados acima, fatos, em OWL, são as informações sobre um particular indivíduo, descritas através das classes que ele pertence, suas propriedades e valores. Os axiomas são utilizados para associar as classes às propriedades, além de fornecer algumas de suas características descritivas e lógicas. Os axiomas podem ser vistos com maiores detalhes no capítulo sobre OWL, como por exemplo, os axiomas *disjointWith* e *functionalProperty*.

Como visto no resultado da execução do código acima, a ontologia ontomusica não teve nenhuma inconsistência listada. Na estrutura da ontomusica está declarada a propriedade 'nome', como demonstra o código abaixo:

```
<owl:DatatypeProperty rdf:ID="nome">
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty" />
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <rdf:Description
rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
          <owl:Class rdf:about="#Periodo"/>
          <owl:Class rdf:about="#Compositor"/>
          <owl:Class rdf:about="#Obra"/>
          <owl:Class rdf:about="#TendenciaMusical"/>
          <owl:Class rdf:about="#GeneroMusical"/>
          <owl:Class rdf:about="#InstrumentoMusical"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
  </owl:DatatypeProperty>
```

Figura 6.27 - Declaração da propriedade 'nome' para a ontomusica

O tipo da propriedade foi editado para *FunctionalProperty*, o que significa que uma instância da classe "Obra", por exemplo, não pode ter mais de um valor para a propriedade 'nome'. Ao se adicionar mais uma instância para a classe 'Obra', e declarar dois valores para a propriedade funcional 'nome', obtém-se o seguinte resultado ao se checar a consistência da ontologia:

```
A base é consistente?
      Não
Individual
http://localhost:8080/OntomusicaQuiz/KillingYourSelfToLive    has
more    than    one    value    for    the    functional    property
http://localhost:8080/OntomusicaQuiz/#nome
```

Figura 6.28 – Geração de inconsistência para uma propriedade funcional através do Pellet

Para exemplificar a detecção de outra inconsistência, a classe 'SomIndeterminado' da ontomusica foi alterada para ser disjunta da classe 'SomDeterminado', conforme figura a baixo.

```
<owl:Class rdf:ID="SomIndeterminado">
<owl:disjointWith rdf:resource="#SomDeterminado" />
 <rdfs:subClassOf rdf:resource="#Percussao"/>
 <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        De altura indefinida, isto é, ruídos.
 </rdfs:comment>
</owl:Class>
```

Figura 6.29 – Declaração da classe 'SomDeterminado' para a ontomusica

No contexto da ontologia há o indivíduo '`MusicaOnto_Instance_50025`' ligado ao conceito 'SomDeterminado', direcionando esse mesmo indivíduo também a classe 'SomIndeterminado' é gerada uma nova inconsistência. O resultado pode ser visto na figura a baixo.

```
A base é consistente?
      Não

Individual http://localhost:8080/OntomusicaQuiz/#MusicaOnto_Instance_50025
is forced to belong to class
http://localhost:8080/OntomusicaQuiz/#SomDeterminado and its complement
```

Figura 6.30 – Geração de inconsistência para a propriedade *disjointWith* através do Pellet

7 SPARQL PROTOCOL AND RDF QUERY LANGUAGE

SPARQL (PRUD'HOMMEAUX e SEABORNE, 2006) é uma linguagem projetada pela *W3C RDF Data Access Working Group*[17], sendo utilizada para realizar consultas sobre estruturas RDF. A linguagem OWL é descrita sobre RDF, então a SPARQL também pode ser utilizada nos modelos descritos com essa linguagem.

A SPARQL é considerada uma linguagem 'orientada a dados', o que significa que ela permite apenas extrair dados dos documentos RDF disponíveis na Web ou armazenados em um meio físico qualquer, não possuindo mecanismos de inferência (MILLER; SEABORNE; REGGIORI, 2002).

7.1 Cláusulas principais da linguagem

A SPARQL tem uma grande semelhança com a linguagem SQL (*Structured Query Language*), e suas principais cláusulas são listadas a baixo (PRUD'HOMMEAUX e SEABORNE, 2006):

Prefix: pode-se associar uma descrição para uma determinada URI. Deve-se colocar a *string* que define o prefixo, seguida de dois pontos ':' e da URI que deve ser mapeada a partir no prefixo criado.

```
1 - PREFIX ontomusica:
              <http://www.rodrigo.goulart.nom.br/feevale/ontomusica/#>
```

[17] http://www.w3.org/2001/sw/DataAccess/

Figura 7.1 – SPARQL: cláusula *Prefix*

Select: nessa cláusula são colocadas as informações que se quer como retorno da consulta. Essas informações são guardadas em variáveis, as quais são identificadas com o sinal de interrogação '?' ou por cifrão '$'.

```
1 - SELECT ?nome
```

Figura 7.2 – SPARQL: cláusula *Select*

From: essa cláusula é implícita, pois a consulta é realizada sobre o modelo ontológico em questão. Porém, para identificar qual documento será pesquisado deve-se utilizar uma URI, conforme figura abaixo.

```
1 - FROM <http://www.rodrigo.goulart.nom.br/feevale/ontomusica/#>
```

Figura 7.3 – SPARQL: cláusula *From*

Where: com essa cláusula especifica-se as restrições que serão feitas para a realização da consulta. As restrições devem seguir o formato de tripla, que são formadas por um sujeito, um predicado e um objeto. Essas triplas podem ser formadas tanto por um objeto ou por um valor literal. Para os predicados, como forma de abreviação ou simplificação, pode ser utilizado o prefixo determinado na cláusula *Prefix.*

```
1 - WHERE
        { ?x ontomusica:nome ?nome }
```

Figura 7.4 – SPARQL: cláusula *Where*

Filter: possibilita a restrição do conjunto de soluções de acordo com a definição de expressões. As expressões podem ser funções e operações, sendo que os operandos dessas funções e operadores são um subconjunto dos tipos de dados do XML *Schema* (xsd:string, xsd:decimal, xsd:double, xsd:dateTime) e tipos derivados de xsd:*decimal*;

```
1 - FILTER (?nome = 'Concerto Solo')
```

Figura 7.5 – SPARQL: cláusula *Filter*

Order By: essa cláusula aplica condições de ordenação sobre o resultado da consulta. Uma condição de ordenação pode ser uma variável ou a chamada a uma função. Pode-se informar a direção da ordenação em ascendente ou decrescente, utilizando Asc e Desc.

```
1 - ORDER BY DESC (?nome)
```

Figura 7.6 – SPARQL: cláusula *Order By*

Limit: com essa cláusula pode-se limitar o número de soluções retornadas da consulta.

```
1 - LIMIT 1
```

Figura 7.7 – SPARQL: cláusula *Limit*

A figura 8 exibe o exemplo de uma *query* montada com as cláusulas descritas acima. Essa *query* lista o nome de uma obra musical, representada pela classe 'Obra' da ontomúsica, que tenha como nome a *string* 'As quatro estacoes'.

```
 1 -   PREFIX ontomusica:
                <http://www.rodrigo.goulart.nom.br/feevale/ontomusica/#>
 2 -   PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
 3 -   SELECT ?nome
 4 -     FROM <http://www.rodrigo.goulart.nom.br/feevale/ontomusica/#>
 5 -    WHERE
 6 -          { ?x ontomusica:nome ?nome .
 7 -            ?x rdf:type ontomusica:Obra .
 8 -   FILTER ( ?nome = 'As quatro estacoes') }
 9 - ORDER BY DESC (?nome)
10 -    LIMIT 1
```

Figura 7.8 – Exemplo de uma consulta SPARQL

7.2 Utilizando SPARQL com o Jena

O framework JENA suporta consultas utilizando a linguagem SPARQL, isso é possível através do módulo ARQ[18]. Esse módulo é capaz de fazer o *parse* nas *queryes* escritas com a linguagem SPARQL (MCCARTHY, 2005), ele já vem incorporado ao Jena desde a versão 2.3. O exemplo de código para execução da *query* mostrada na figura 7.8 da seção anterior, pode ser visto a baixo, na figura 7.9.

[18] http://jena.sourceforge.net/ARQ/

```
//Criando o modelo para a ontomusica
String baseURI = "file:./owl/ontomusica.owl";
String nameSpace = "http://www.rodrigo.goulart.nom.br/feevale/ontomusica/#";
OntDocumentManager gerenciador = new OntDocumentManager();
OntModelSpec spec = new OntModelSpec(OntModelSpec.OWL_MEM );
spec.setDocumentManager(gerenciador );
OntModel model = ModelFactory.createOntologyModel(spec, null );
model.read(baseURI);

String queryString =
    "PREFIX ontomusica: <http://www.rodrigo.goulart.nom.br/feevale/ontomusica/#> " +
    "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
    "SELECT ?nome " +
     "FROM <http://www.rodrigo.goulart.nom.br/feevale/ontomusica/#> " +
     "WHERE " +
          "{ ?x ontomusica:nome ?nome . " +
          "  ?x rdf:type ontomusica:Obra ." +
          "     FILTER ( ?nome = 'As quatro estacoes') } " +
"ORDER BY DESC (?nome) " +
     "LIMITI 1";

  Query query = QueryFactory.create(queryString);
 //Executa a query e obtém os resultados
  QueryExecution qe = QueryExecutionFactory.create(query, model);
  ResultSet results = qe.execSelect();
 //Imprime os resultados da query
  ResultSetFormatter.out(System.out, results, query);
 //Liberando os recursos utilizados ao rodar a query acima
  qe.close();
```

Figura 7.9 – Execução de uma *query* SPARQL com a utilização do Jena

Através da linha número 24 do código acima, o resultado obtido com a execução desse código é exibido na figura 7.10.

```
-------------------------------------------------------------------
| nome                                                            |
===================================================================
| "As quatro estacoes"^^<http://www.w3.org/2001/XMLSchema#string> |
-------------------------------------------------------------------
```

Figura 7.10 – Resultado de uma query SPARQL sobre a ontomusica

8 ONTOQUIZ

Com base no capítulo de análise sobre a estrutura da ontologia ontomusica e do sistema que disponibiliza perguntas construídas sobre uma pequena parcela das informações estruturadas por ela, nota-se que os maiores esforços foram aplicados na estrutura da ontologia, deixando um pouco de lado a exploração dos dados da estrutura. Com o objetivo de implementar um sistema dinâmico, capaz de montar questionamentos sobre todo o conhecimento armazenado em uma ontologia, é que foi implementado o OntoQuiz. O sistema utiliza como estudo de caso a ontologia ontomusica, mas não é limitado a sua estrutura, podendo ser aplicado a diversas bases de conhecimento estruturada com o uso da linguagem OWL.

8.1 Definição

Para a implementação, optou-se pela linguagem de programação Java e pela utilização do *framework* Java Server Faces (JSF). Um dos principais motivos que levou a escolha do Java foi o *framework* Jena e o sistema de inferência Pellet, utilizados para a manipulação dos dados e checagem da consistência da ontologia, também serem implementados com a linguagem. A utilização do *framework* JSF foi escolhida pelas facilidades que ele disponibiliza para a construção e estruturação do sistema. Juntamente com a utilização do JSF são disponibilizadas as bibliotecas *Ajax4JSF*[19] e *RichFaces*[20], as quais auxiliam no desenvolvimento de interfaces com a utilização da tecnologia Ajax. A utilização dessas bibliotecas facilita o desenvolvimento de aplicações com uma maior interação entre o usuário e o conteúdo disponibilizado, resultando em uma melhor dinamicidade para o sistema.

[19] https://ajax4jsf.dev.java.net/

Os capítulos sobre lógica descritiva, OWL, sistemas de inferência e linguagem de *query* Sparql detalharam e exemplificaram os principais trechos de código que foram utilizados para a implementação do OntoQuiz. O sistema é uma aplicação web que disponibiliza toda a informação inserida na ontologia através de uma tela que possibilita ao usuário arrastar e soltar os componentes que formam as perguntas. Esses componentes são os elementos que formam as triplas RDF da estrutura. Desta forma, o usuário tem a disposição os dados necessários para criar e responder as suas próprias perguntas.

8.2 Fluxo de funcionamento

Após ter sido feita a descrição geral da ferramenta, essa seção apresenta o fluxo de funcionamento do quiz:

- O sistema está disponível em um servidor Web com suporte a tecnologia Java/JSP para uso do JSF, o usuário acessa a aplicação através de uma URL digitada em um navegador.

- Ao acessar o sistema, o usuário tem disponível todos os conceitos armazenados na ontologia. Esses conceitos ficam disponíveis em uma coluna da página e são exibidos dentro de caixas que podem ser arrastadas dentro da interface.

- O usuário escolhe um conceito e o sistema exibe os indivíduos que fazem parte desse conceito. Ao escolher um determinado indivíduo, o quiz exibe todas as propriedades que o ligam a um outro determinado conceito ou literal. Dessa forma, são disponibilizados dados suficientes para uma pergunta ser montada.

- No momento em que o usuário montar a sua pergunta completa, o sistema gera automaticamente três alternativas. Caso o usuário escolha a alternativa incorreta, a aplicação pode mostrar dicas sobre os conceitos que envolvem a pergunta e a resposta.

---

[20] http://www.exadel.com/web/portal/products/VisualComponentPlatform

8.3 Interface com o usuário e lógica das funcionalidades

Todas as informações disponibilizadas pelo sistema são acessíveis através de uma única tela, a qual está dividida em áreas. Essas áreas disponibilizam os dados para a escolha dos usuários, locais para o usuário arrastar e soltar essas informações, para exibição das alternativas e escolha da alternativa correta e uma área para a exibição de dicas, com o objetivo de auxiliar o usuário em suas respostas. Nessa interface serão utilizados os recursos de *drag and drop* (arrastar e soltar) e atualização dinâmica dos dados apresentados sem a necessidade de atualização de toda a página. Esses recursos são utilizados para aumentar a dinamicidade da página e a interação com o usuário. Nas subseções abaixo são apresentadas as áreas que compõem a interface do quiz.

8.3.1 Área de exibição dos dados da ontologia

Conforme foi descrito no texto deste trabalho, toda a informação armazenada em um documento baseado em RDF, como documentos OWL, é estruturada através de triplas formadas por instâncias, propriedades e objetos. Levando essa informação em consideração, à parte que disponibiliza os dados para o usuário formar a sua própria pergunta foi estruturada em três colunas. Cada coluna é responsável por disponibilizar um dos elementos formadores da tripla, como pode ser visto na figura abaixo. Para a extração dos dados do documento OWL foi utilizada a linguagem Java em conjunto com o framework Jena.

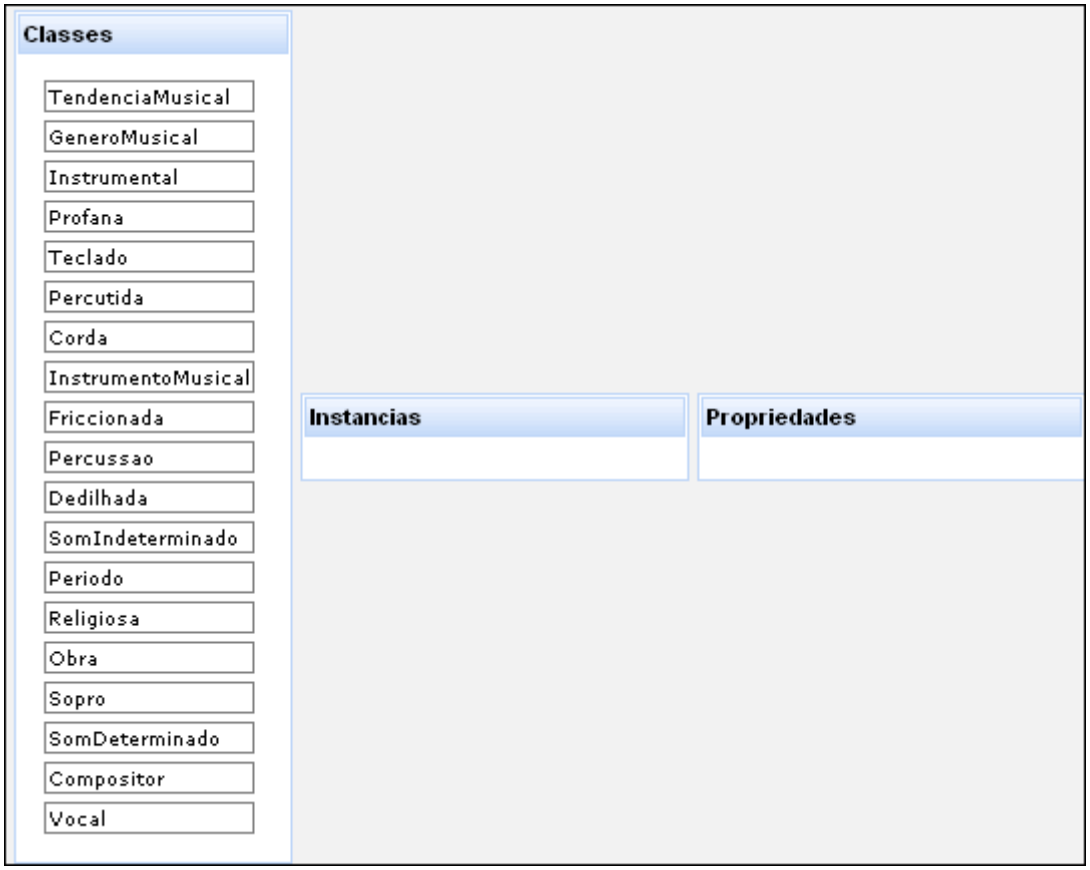

Figura 8.1 – OntoQuiz, área de exibição de dados

Com as classes da ontologia sendo disponibilizadas ao usuário, ele tem a possibilidade de escolher uma delas, para isso, é necessário que ele clique sobre a classe e a arraste até a área de seleção de informações do quiz, conforme exemplifica a figura número 8.2.

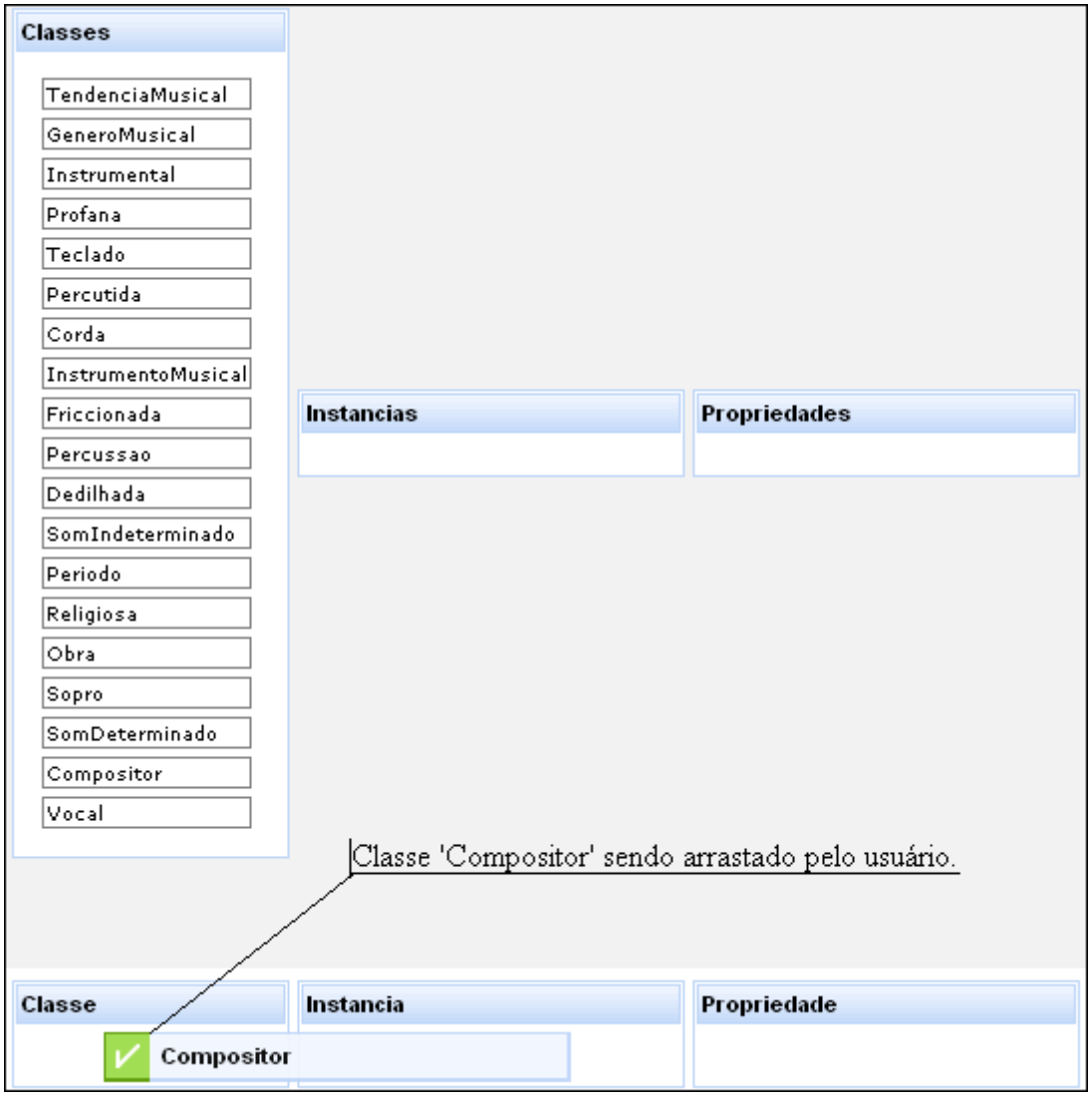

Figura 8.2 – OntoQuiz, classe sendo arrastada pelo usuário

8.3.2 Área de escolha de informações que forma a pergunta

Após o usuário ter selecionado uma classe e uma instância, o sistema popula objetos Java que guardam as triplas RDF, escondendo do usuário o objeto da tripla. O objeto de cada tripla é a resposta certa para a pergunta. No momento em que o usuário arrasta uma propriedade até o local destinado a ela, o sistema já tem uma pergunta formada, como pode ser visto na próxima figura.

Figura 8.3 – OntoQuiz, exemplo de pergunta montada pelo usuário

No momento em que a propriedade é solta pelo usuário, o sistema popula objetos para a exibição das alternativas para resposta da questão. A próxima seção descreve essa etapa.

8.3.3 Área de exibição das alternativas

No momento em que o usuário terminou de montar a sua pergunta, são geradas as alternativas para a pergunta, como exemplifica a figura 8.4, onde são listadas as alternativas para a pergunta montada na figura 8.3.

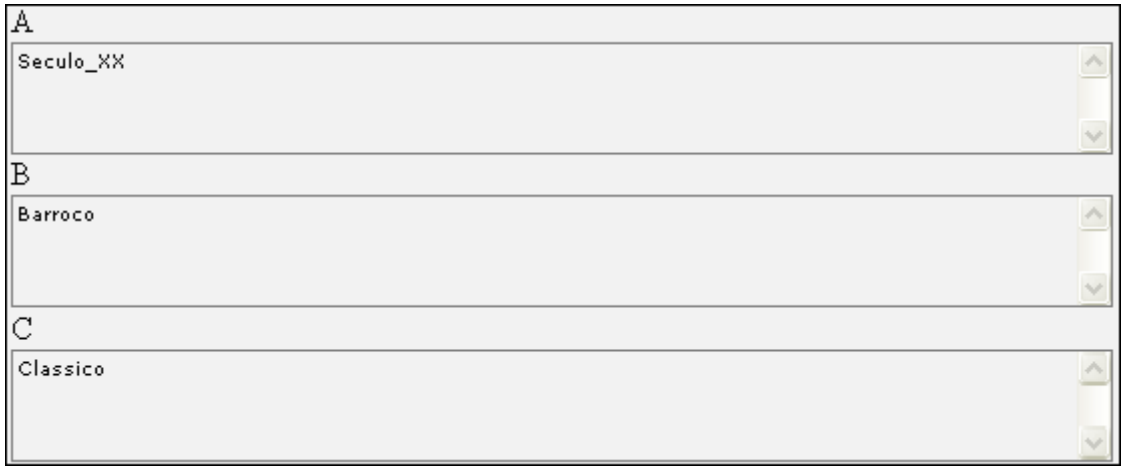

Figura 8.4 – OntoQuiz, alternativas para uma determinada pergunta

Através da linguagem de *query* Sparql são selecionados todos os objetos para completar a tripla que o usuário escolheu. Pois para a formação da questão o usuário seleciona apenas a instância e a propriedade. Esse processo seleciona todas as relações possíveis do indivíduo com uma outra instância relacionada a ele, isso no caso de uma *object property*. Isso é necessário para o caso de um indivíduo ter mais de uma tripla formada com a propriedade escolhida, como no caso de um compositor, que pode ter composto várias obras. Dessa forma é possível exibir apenas uma resposta correta.

O trecho de código seguinte exemplifica uma *query* montada por esse processo.

```
PREFIX ontomusica: <http://localhost:8080/OntomusicaQuiz/#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?Periodo
 WHERE {
        ?Periodo rdf:type ontomusica:Periodo .
        ontomusica:Ludwig_Van_Beethoven ontomusica: pertenceAoPeriodo
                  ?Periodo
       }
```

Figura 8.5 – OntoQuiz, exemplo de query para listar respostas corretas

Essa *query* foi montada quando o usuário escolheu a classe 'Compositor', a instância 'Ludwig_Van_Beethoven' e a propriedade 'pertenceAoPeriodo'. Essa *query* trará todos as períodos aos quais o compositor pertence, ou seja, todas as respostas verdadeiras possíveis.

Nesse mesmo momento, também é montada uma lista com todas as respostas possíveis para esse contexto, no exemplo, todos os períodos inseridos na ontologia. A próxima figura mostra um exemplo de *query* montada através desse processo, e quando a resposta é uma outra instância, ou seja, para o caso de uma *object property*.

```
PREFIX ontomusica: <http://localhost:8080/OntomusicaQuiz/#>
PREFIX rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
SELECT ?Periodo
 WHERE {
        ?Periodo rdf:type ontomusica:Periodo
       }
```

Figura 8.6 – OntoQuiz, exemplo de *query* para uma *object property*

Porém a resposta também pode ser um literal, ou seja, uma *datatype property*. Nesse caso o objetivo da *query* é trazer valores possíveis para um tipo de dado específico, como uma *string*. A próxima figura exibe um exemplo para essa particularidade, onde o usuário escolheu a classe 'Compositor', a instância 'Ludwig_Van_Beethoven' e a propriedade '`nomePai`'.

```
PREFIX ontomusica:http://localhost:8080/OntomusicaQuiz/#
PREFIX rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
SELECT ?nomePai
  FROM http://localhost:8080/OntomusicaQuiz/#
 WHERE {
         ?x ontomusica:nomePai ?nomePai .
         ?x rdf:type ontomusica:Compositor .
FILTER ( ?nomePai != 'Johann Van Beethoven')
        }
```

Figura 8.7 – OntoQuiz, exemplo de *query* para uma *datatype property*

Para esse caso, a *query* já exclui o literal da resposta correta, que no caso do exemplo acima é a *string* 'Johann Van Beethoven'.

A partir da lista gerada com as respostas corretas e da lista contendo todas as respostas possíveis, um outro processo seleciona duas alternativas erradas. Um número randômico é gerado para selecionar em qual posição a alternativa correta irá ser alocada, se na opção ‘A’, ‘B’ ou ‘C’. Desta forma a resposta correta não aparece sempre na mesma posição.

8.3.4 Área de exibição de dicas

Após o usuário ter montado a sua pergunta, e o sistema ter disponibilizado as três alternativas para a escolha, uma delas pode ser arrastada até a área destinada a resposta. Em caso de o usuário ter escolhido a resposta certa, o sistema o informa do sucesso, caso contrário é exibida uma dica, a qual está relacionada com o contexto da pergunta formada.

As dicas exibidas são escolhidas através da instância selecionada, com a utilização do framework Jena é possível listar todas as propriedades que a ligam com outros valores. Dessa forma, pode-se exibir os objetos da tripla que não sejam os da resposta correta. O módulo de dicas tem o objetivo de auxiliar o usuário que errou uma determinada pergunta.

Uma lista guarda todas as dicas possíveis, e na medida em que o usuário erra a alternativa de uma questão, o sistema informa uma dica diferente. É feita uma distinção entre propriedades que ligam a instância da pergunta a um objeto ou a um literal. Caso a resposta for uma outra instância da ontologia, as dicas são feitas sobre as propriedades desse indivíduo, porém, se a resposta for um literal, as dicas são feitas sobre as propriedades da própria instância escolhida para a pergunta.

Para exemplificar o caso de uma propriedade que liga uma instância à outra, é exibido uma dica abaixo para a propriedade ‘pertenceAoPeriodo’ da ontomusica, ela liga a instância de um compositor a uma outra instância da classe ‘Periodo’.

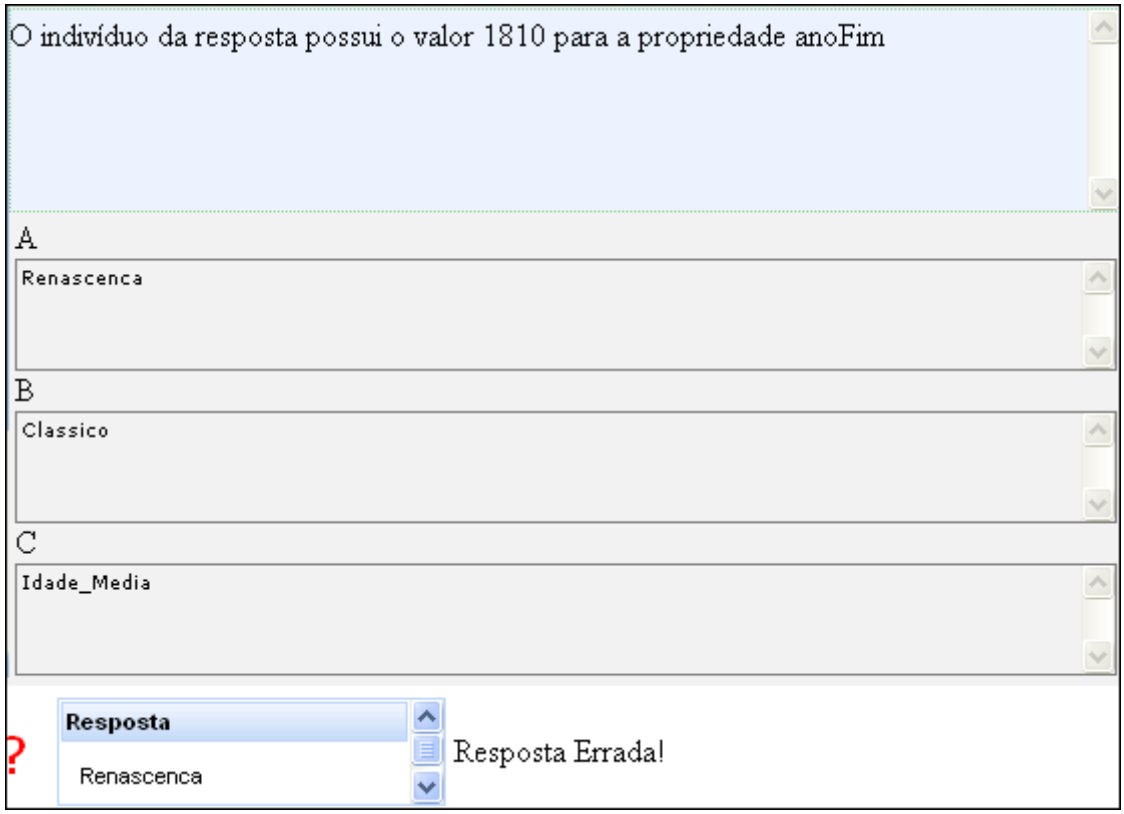

Figura 8.8 – OntoQuiz, dica para uma *object property*

No exemplo da figura acima, foi exibida uma dica para o indivíduo da resposta, que no caso é um período.

Como exemplo para uma propriedade que liga um indivíduo a um literal, é mostrado na próxima figura uma dica para a propriedade 'nomePai', que liga uma instância da classe compositor a uma *string*.

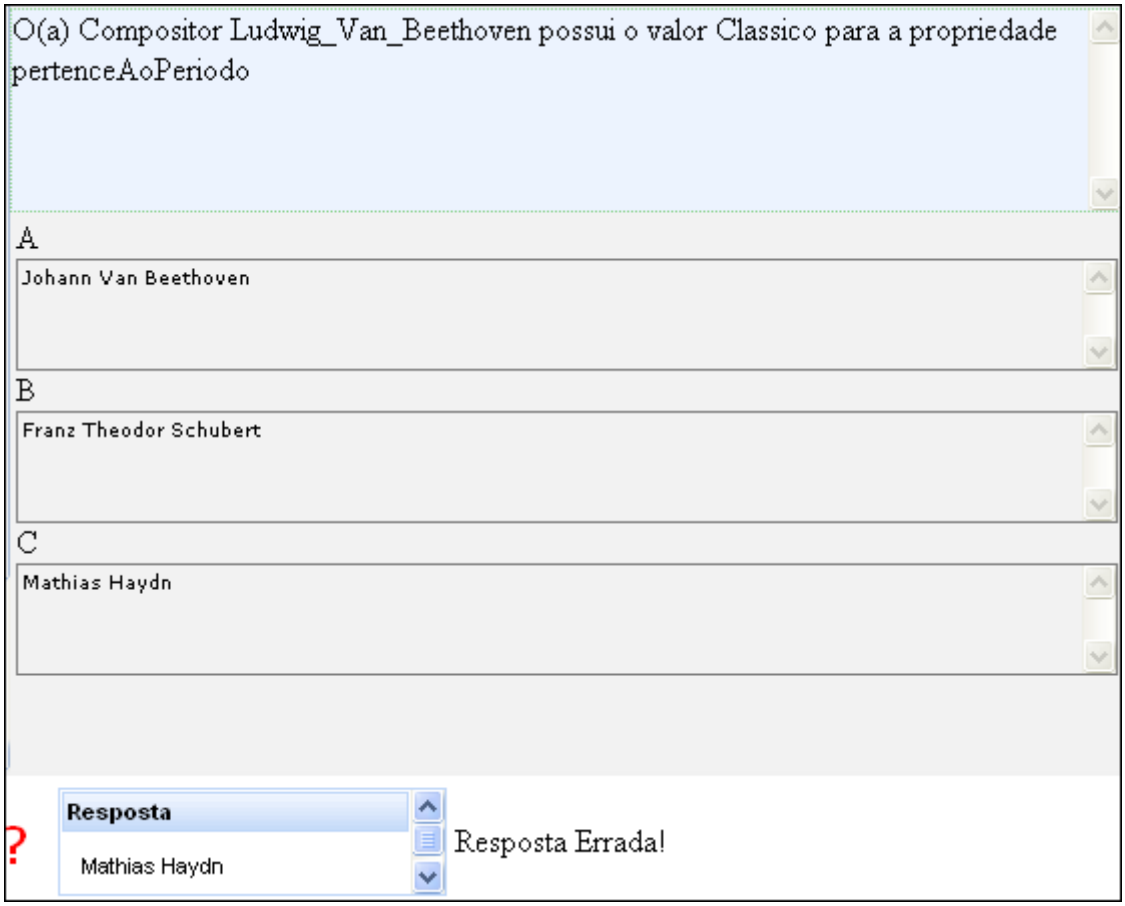

Figura 8.9 – OntoQuiz, dica para uma *datatype property*

No exemplo acima, a resposta é um literal, não tendo nenhuma estrutura para ele dentro da ontologia, então as dicas são feitas sobre o próprio indivíduo selecionado para a pergunta, no caso o compositor 'Ludwig_Van_Beethoven'.

A próxima figura exibe o algoritmo utilizado para a formação da estrutura utilizada para a exibição das dicas exemplificadas nas figuras 8.8 e 8.9.

```
01- public void montaDicas(String dsInstancia, String tipo) {
02-  String dsUri = "http://localhost:8080/OntomusicaQuiz/#" +
03-                 dsInstancia;
04-  String dicaDesc = null;
05-  Individual individuo = model.getIndividual(dsUri);
06-  nrNumeroDica = 0;
07-  try
08-  {
09-    int count = 0;
10-    //estrutura que guarda as dicas que são exibidas na interface
11-    dicas = new HashMap();
12-    for(Iterator i = individuo.listProperties(); i.hasNext();)
13-    {
14-      dicaDesc = null;
15-       Statement stmt = (Statement) i.next();
16-       Property prop = stmt.getPredicate();
17-       RDFNode node = stmt.getObject();
18-       String dsValor = null;
19-       if (node.isResource())
20-      {
21-        try
22-       {
23-          dsValor = stmt.getResource().getLocalName();
24-        } catch (ResourceRequiredException e) {e.printStackTrace();}
25-       } else if (node.isLiteral())
26-      {
27-        try
28-        {
29-           dsValor = stmt.getLiteral().getValue().toString();
30-         } catch (LiteralRequiredException e) {e.printStackTrace();}
31-       }
32-       if (tipo.equalsIgnoreCase("instancia"))
33-      {
34-       dicaDesc = new String("O indivíduo da resposta possui o valor
"
35-            +   dsValor   +   "   para   a   propriedade   "   +
prop.getLocalName());
36-       } else if (tipo.equalsIgnoreCase("literal") &&
37-             !dsValor.equalsIgnoreCase(this.getDsRespostaCerta()))
38-      {
39-      dicaDesc = new String ("O(a) " + dsClasse + " " +
40-             dsInstancia + " possui o valor " + dsValor + " para a
41-                propriedade " + prop.getLocalName());
42-      }
43-    //monta uma lista de dicas sobre o indivíduo pertinente
44-     if (dicaDesc != null)
45-      {
46-         dicas.put(count, dicaDesc);
47-         count++;
48-         System.out.println(dicaDesc);
49-       }
50-    }
51-  } catch (Exception e) {
52-    e.printStackTrace();
```

Figura 8.10 – OntoQuiz , algoritmo para geração de dicas

O método ‘montaDicas’ popula um objeto que armazena as possíveis dicas para auxiliar o usuário em sua resposta. A estrutura recebe como parâmetro a string da instância

escolhida pelo usuário na interface e uma outra string que indica o tipo da propriedade que relaciona essa instância com a resposta, se uma *object property* ou uma *datatype property*.

Na linha número 5 cria-se o objeto para a instância escolhida pelo usuário, em seguida são listadas todas as suas propriedades. Para cada uma das propriedades verifica-se o objeto que ela relaciona, se a uma instância ou a um literal, esse tratamento é feito a partir da linha 19 até a 31.

Entre as linhas 31 e 42 verifica-se o tipo da propriedade que o usuário escolheu na pergunta, possibilitando formar a frase correta a ser exibida na tela. Nessas linhas é concluída a descrição da dica. Por fim, na linha 46, a frase com a dica é inserida na estrutura que será usada para exibi-la na interface.

8.4 Utilização do Quiz com uma ontologia sobre professores

Apenas para exemplificar a utilização do OntoQuiz com uma outra estrutura ontológica, a figura 8.10 exibe a tela do sistema utilizando uma ontologia que guarda informações sobre professores.

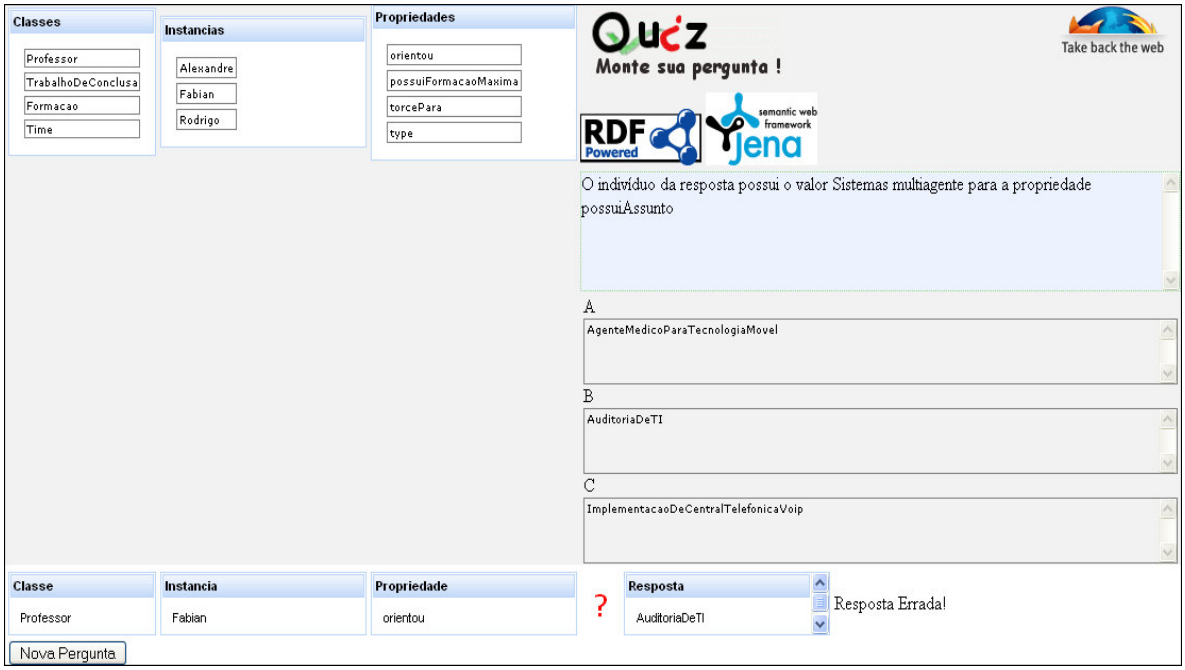

Figura 8.11 – OntoQuiz, tela com ontologia sobre professores

CONCLUSÃO

O objetivo inicial deste trabalho foi o estudo dos conceitos referentes à organização do conhecimento através da estrutura de ontologias e sobre a inferência na informação armazenada sobre ela.

Com base na descrição dessas informações foi mostrada a importância e a crescente utilização de ontologias para a estruturação de domínios de conhecimento. Ainda referente a ontologias, é apresentado um exemplo na conceitualização de informações referentes à história da música por meio de ontologias, a Ontomúsica, a qual foi estruturada com a utilização da linguagem OWL. Desta mesma forma também foi exposto o uso de lógica de descrição, visando disponibilizar estrutura e semântica para o domínio em questão. Essa estrutura formal das lógicas de descrição possibilita a extração de dados que não estejam diretamente explicitados na base de conhecimento, e é utilizada por várias ferramentas de edição de ontologias.

No capítulo sobre a linguagem OWL foram apresentados os seus principais axiomas, através dos quais foram forçadas inconsistências na estrutura ontológica sobre a história da música. Essas inconsistências geradas foram detectadas pelos serviços de inferência da ferramenta Pellet.

O framework Pellet possibilitou a exibição dessas inconsistências geradas, as apresentando através de uma descrição textual. Dessa forma, o Pellet auxilia na solução para os problemas encontrados. Com a base ontológica livre de inconsistências, um sistema capaz de exibir todo o conhecimento de uma determinada ontologia foi desenvolvido, o OntoQuiz.

Para a extração das triplas RDF da ontologia, foi utilizado o framework Jena, o qual facilitou a listagem e a manipulação dessas triplas, pois as disponibiliza através de um modelo orientado a objetos, facilmente tratado pela linguagem Java. Através do Jena é possível a

utilização da linguagem de *query* Sparql. Essa linguagem facilitou a listagem de informações que envolvem mais de um conceito ou propriedade da estrutura ontológica, como no caso da listagem das alternativas e das dicas para as perguntas do quiz. Através dessas dicas geradas com a utilização da linguagem Sparql, o usuário pode deduzir as respostas corretas.

Com a utilização dessas ferramentas e tecnologias possibilitou-se a melhoria do quiz originalmente desenvolvido no trabalho de conclusão do aluno Rogério Boff (BOFF, 2005). O OntoQuiz disponibiliza um sistema de perguntas dinâmico capaz de agir sobre todo o conhecimento do mundo contextualizado pela ontologia.

REFERÊNCIAS BIBLIOGRÁFICAS

ABEL, Mara. **Sistemas Especialistas**, 1998. Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre. Disponível em: http://www.ppgia.pucpr.br/~scalabrin/SE_MILTON/SistEspec%20MaraAbel%20mar2002.pdf. Acesso em 19 de nov. de 2006.

ABEL, Mara. **Estudo da perícia em petrografia Sedimentar e sua importância para a Engenharia de Conhecimento**, 2001. Tese de doutorado - Universidade Federal do Rio Grande do Sul, Porto Alegre.

ALMEIDA, M.B. ; BAX, M.P. **Uma visão geral sobre ontologias: pesquisa sobre definições, tipos, aplicações, métodos de avaliação e de construção.** Ciência da Informação, Brasília, v.32, n.3, p.7-20, 2003.

AMORIM, Ricardo José Rocha. 2002. **Desenho de um Sistema Gerenciador Inteligente de Recursos em um ambiente de Aprendizagem Cooperativa.** Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Produção da Universidade Federal de Santa Catarina, Florianópolis.

ARAUJO, Moysés de. **Educação à distância e a web semântica: Modelagem ontológica de materiais e objetos de aprendizagem para a plataforma COL**. São Paulo, 2003. 173 f. Tese para obtenção do título de doutor em engenharia – Escola Politécnica da Universidade de São Paulo.

BARRETO, J. M. **Inteligência artificial no limiar do século XXI**. 3ª edição, Ed. – Florianópolis, 2002. 392 p.

BECHHOFER, S; HARMELEN, F. V; HENDLER, J; HORROCKS, I; CGUINNESS, D. L; PATEL-SCHNEIDER, P. F; STEIN, L. A. **OWL Web Ontology Language Reference.** http://www.w3.org/TR/2004/REC-owl-ref-20040210/, 2004. Acesso em: 12 out. 2006.

BERNERS-LEE, T.; HENDLER, J. and LASSILA, O. (2001) The semantic web. Scienti_c American, May 2001. Disponível em: http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&pageNumber=2&catID=2

BERNERS-LEE, Tim**. Primer: Getting into RDF & Semantic Web using N3**. The World

Wide Web Consortium (W3C). Abril, 2002. Disponível em <www.w3.org/2000/10/swap/Primer.html>. Acesso em 24/03/2007.

BOFF, Rogério Eduardo. **Educação Musical à Distância Utilizando Ontologias**. 2005. Projeto de Diplomação (Bacharelado em Ciências da Computação) – Instituto de Ciências Exatas e Tecnológicas (ICET), Centro Universitário FEEVALE, Novo Hamburgo.

BONIFACIO, Ailton Sergio. **Ontologias e Consulta Semântica: Uma Aplicação ao Caso Lattes.** 2002. Dissertação parcial para o grau de mestre – Instituto de informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

BREITMAN, Karin Koogan. **Web semântica: a internet do futuro.** Rio de Janeiro, 2005.

BRITO, Parcilene Fernandes. **Dedução automática por Tableaux estruturada em XML,** 2003. Dissertação de mestrado submetida à Universidade Federal do Rio Grande do Sul, Porto Alegre.

BUENO, Tânia Cristina D'Agostini, 2005. **Engenharia da Mente: Uma Metodologia de Representação do Conhecimento para Construção de Ontologias em Sistemas Baseados em Conhecimento.** Tese de doutorado apresentada a Universidade Federal de Santa Catarina – UFSC, Florianópolis.

CALVANESE, D., De Giacomo, G. **Description Logics for Conceptual Data Modeling in UML**.ESSLLI 2003. Disponível em http://ftp.dis.uniroma1.it/~degiacom/didattica/esslli03/. Acesso em 12 de nov.

CAMPOS, M.L. DE A. **Modelização de domínios de conhecimento: uma investigação de princípios fundamentais.** 2004**.** Ciência da Informação**,** Brasília, v.33, n.1, p.22-32.

CARLAN, Eliana. **Ontologia e Web Semântica**. 2006. Projeto de Diplomação (Bacharelado em Biblioteconomia) – Departamento de Ciência da Informação e Documentação, Universidade de Brasília - UnB, Brasília.

DAVIS, R., Shrobe, H. e Szolovits, P. **What is knowledge representation?** AI Magazine, pages 17-33. 1993.

DICKINSON, Ian. 2004. **Implementation experience with the DIG 1.1 specification**. Digital Media Systems Laboratory, Bristol.

DRUCKER, Peter. **Sociedade Pós-Capitalista**. 7ª edição. São Paulo:Pioneira, 1999.

FaCT. "**The FaCT System**", 2003. http://www.cs.man.ac.uk/~horrocks/FaCT/. Acesso em 10 nov. 2006.

FALBO, Ricardo de A. **Interação de conhecimento em um ambiente de desenvolvimento de software.** 1998. f.188. Tese (Doutorado). Universidade Federal do Rio de Janeiro. Rio de Janeiro.

FONSECA, Frederico T. e Egenhofer Max J. **Sistemas de informação geográficos baseados em ontologias.** Disponível em: <www.spatial.maine.edu/~fred/fonseca_IP.pdf> Acessado em 25 de Nov 2006.

FRANCONI, Enrico, "**Structural Description Logics: FL-** ",in Description Logics Course, 2002. Disponível em: http://www.inf.unibz.it/~franconi/dl/course/slides/struct-DL/flminus.pdf. Acesso em 12 de nov. 2006.

GUARINO, N. **Formal Ontology and Information Systems**, 1998. In: Proceesings of the fist International Conference on Formal Ontology in Information Sustems. FOIS,98. Trento. Disponível em : http://www.loa-cnr.it/Papers/FOIS98.pdf. Acesso em: 25 de nov. 2006.

GUIZZARD, Giancarlo. **Uma Abordagem Metodológica de Desenvolvimento para e com Reuso, Baseada em Ontologias Formais de Domínio**. 2000. 148 f. Dissertação (Mestrado em Informática) – Universidade Federal do Espírito Santo, UFES, Vitória.

GRUBER, Thomas R. **Toward principles for the Design of Ontologies Used for Knowledge Sharing**, 1993. In: FORMAL ONTOLOGY IN CONCEPTUAL ANALYSIS AND KNOWLEDGE REPRESENTATION. Padova. Italy. Available as Technical Report KSL 93-04, Stanford University. Disponível em: http://www.cise.ufl.edu/~jhammer/classes/6930/XML-FA02/papers/gruber93ontology.pdf. Acesso em 18 de nov. 2006.

HARMELEN, F. V; MCGUINNESS, D. L. **OWL Web Ontology Language Overview,** 2004. http://www.w3.org/TR/2004/REC-owl-features-20040210/. Acesso em: out. 2006.

HORROCKS, Ian. TSARKOV, Dmitry. **Reasoner Demonstration. Implementing new reasoner with datatypes support**, 2004. Disponível em: http://wonderweb.semanticweb.org/deliverables/documents/D14.pdf. Acesso em 20 de nov. de 2006.

HORROCKS, Ian. SATTLER, Ulrike. **A Tableaux Decision Procedure for SHOIQ**, 2005. School of computer Sience, Univercity of Manchester, UK. Disponível em: http://www.cs.man.ac.uk/~horrocks/Publications/download/2005/HoSa05a.pdf. Acesso em: 23 de nov. de 2006.

JARUFE, Manuel Salomon Salazar. **Concepção de Sistema de Informação de Apoia à Operação de Sistemas Complexos: Uma Abordagem da Engenharia do Conhecimento**, 1999. Tese para doutorado em Engenharia de Produção, Universidade Federal de Santa Catarina, Florianópolis.

JENA. "**Jena 2 Ontology API**", 2004. http://jena.sourceforge.net/ontology/, acesso em 18/06/2006.

JENA. "**A Semantic WEB Framework for Java**", 2006b. http://jena.sourceforge.net/. Acesso em 10 nov. 2006.

JENA. "**Jena 2 Inference support**", 2006c. http://jena.sourceforge.net/inference/index.html, acesso em 09 de nov. 2006.

JENA. "**HOWTO use Jena with an external DIG reasoner**", 2006d. http://jena.sourceforge.net/how-to/dig-reasoner.html, acesso em 10 de nov. 2006.

JÚNIOR, Olival de Gusmão Freitas. **Um Modelo de Sistema de Gestão do Conhecimento para Grupos de Pesquisa e Desenvolvimento**, 2003. Tese de doutorado apresentada ao Programa de Pós-Graduação em Engenharia da Produção da Universidade Federal de Santa Catarina. Florianópolis.

MACEDO, Nestor Adolfo Mamani. **Criando uma arquitetura de memória corporativa baseada em um modelo de negócio.** Rio de Janeiro, 2003. Tese para obtenção do título de doutor em informática – Pontifícia Unidade Católica do Rio de Janeiro.

MANOLA, F. and MILLER, E. (eds.) (2003) "**RDF Primer – W3C Working Draft 23 January 2003**". Disponível em: http://www.w3.org/TR/rdf-primer/. Acesso em: 14 abr. 2006.

MCCARTHY, PHILIP. **Search RDF data with SPARQL**, 2005. Disponível em: www-128.ibm.com/developerworks/library/j-sparql/. Acesso em: 26/03/2007.

MILLER, Libby; SEABORNE, Andy; REGGIORI, Alberto. **Three implementations ofsquishql, a simple rdf query language**. In: Proceedings of the First International Semantic Web Conference on The Semantic Web. London, 2002. Disponível em http://portal.acm.org/citation.cfm?id=711274. Acesso em 25/03/2007.

MOREIRA, Álvaro Freitas. **Introdução as Lógicas de Descrição**, 2002. Universidade Federal do Rio Grande do Sul, Porto Alegre. Disponível em: http://www.inf.unisinos.br/~renata/cursos/iam/aula4-dl2.pdf. Acesso em 18 de nov. 2006.

NARDI, D., R. J. Brachman. **"An Introduction to Description Logics"**. In the Description Logic Handbook, edited by F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, P.F. Patel-Schneider, Cambridge University Press, 2002, pages 5-44. Disponível em: http://www.inf.unibz.it/~franconi/dl/course/dlhb/dlhb-01.pdf. Acesso em 12 de nov. 2006.

NOY, Natalya F.; MCGUINNESS, Deborah L. **Ontology Development 101: A Guide to Creating Your First Ontology**. [S.l.: s.n.], 2000. Disponível em: http://www.ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinness-abstract.html. Acesso em: 18 Ago. 2006.

OLSSON, O. **CommonKADS and the KADS-II Project**. Disponível em:http://www.sics.se/ktm/projects/kads.html. Acesso em: 25 nov 2006.

PATEL-SCHNEIDER Peter F., HAYES Patrick, HORROCKS Ian, 2004. **OWL web ontology language Abstract Syntax and Semantics, W3C Recommendation**. Disponível em: http://www.w3.org/TR/owl-semantics/. Acesso em: 15/05/07

PELLET. "Pellet: An OWL DL Reasoner", 2006. Disponível em: http://pellet.owldl.com. Acesso em 25 de nov. 2006.

PRUD'HOMMEAUX, E.; SEABORNE, A. **SPARQL query language for RDF**. W3C working draft, 4 October 2006. Disponível em: <http://www.w3.org/TR/rdf-sparql-query/>. Acesso em: 20 mar. 2007.

REZENDE, Solange O. (org.). **Sistemas Inteligentes: Fundamentos e Aplicações**. Barueri-SP: Manole, 2003.

SILVA, Luís Alvaro de Lima. **Aplicando Métodos de Solução de Problemas em Tarefas de Interpretação de Rochas**, 2001. Dissertação de Mestrado, Universidade Federal do Rio Grande do Sul, Porto Alegre.

SMITH, M. K., WELTY, C. and McGUINNESS, D. L. (eds.) (2004) "**OWL Web Ontology Language Guide – W3C Recommendation 10 February 2004**". Disponível em: http://www.w3.org/TR/owl-guide/. Acesso em: 08 out . 2006.

SPARQL. "**SPARQL Query Language for RDF**", 2006. Disponível em: http://www.w3.org/TR/rdf-sparql-query. Acesso em: 25 de nov. 2006.

USCHOLD, M; GRUNINGER, M. **Ontologies: Principles, Methods and Applications. The Knowledge Engineering Review**, 1996**.** Disponível em: http://citeseer.ist.psu.edu/uschold96ontologie.html. Acesso em 18 de nov. 2006.

VALENTE, Andre. **Legal Knowledge Engineering: A Modelling Approach,** 1995. IOS Press, (Amsterdam) and Omsha (Tokyo).

VIEIRA, Renata ; SANTOS, Débora Abdalla dos ; SILVA, Douglas Michaelsen da ; SANTANA, Menandro Ribeiro . **Web semântica: ontologias, lógica de descrição e inferências,** 2005**.** In: Cesar Teixeira; Eduardo Barrere; Iran Abraão. (Org.). Web e Multimidia: Desafios e Soluções (WebMedia 2005 - Minicursos). 1 ed. Porto Alegre: SBC, 2005, v. 1, p. 127-167.

WILKINSON, Kevin; SAYERS, Craig; KUNO, Harumi A.; REYNOLDS, Dave. **Efficient rdf storage and retrieval in jena2**, 2003. Disponível em http://www.hpl.hp.com/techreports/2003/HPL-2003-266.pdf. Último acesso: 24/03/2007.

ZIULKOSKI, Luís Cláudio Chaves. **Coleta de Requisitos e Modelagem de Dados para Data Warehouse: um Estudo de Caso utilizando Técnicas de Aquisição de Conhecimento,** 2003**.** Universidade Federal do Rio Grande do Sul, Porto Aletre. Disponível em: http://www.inf.ufrgs.br/gpesquisa/bdi/publicacoes/files/ColetaRequisDWH.pdf. Acesso em 19 de nov. de 2006.