MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
SECRETARIA DA CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA
CURSO DE MESTRADO EM SISTEMAS E COMPUTAÇÃO

FRANCISCO HANDRICK TOMAZ DA COSTA

ROSAI – UMA PROPOSTA DE REPRESENTAÇÃO DO MODELO ROSA EM LINGUAGEM LÓGICA

Rio de Janeiro
2005

INSTITUTO MILITAR DE ENGENHARIA

FRANCISCO HANDRICK TOMAZ DA COSTA

ROSAI – UMA PROPOSTA DE REPRESENTAÇÃO DO MODELO ROSA EM LINGUAGEM LÓGICA

Dissertação de Mestrado apresentada ao Curso de Mestrado em Sistemas e Computação do Instituto Militar de Engenharia, como requisito parcial para a obtenção do título de Mestre em Sistemas e Computação.

Orientador: Fábio André Machado Porto – D. Sc.

Rio de Janeiro
2005

C2005

INSTITUTO MILITAR DE ENGENHARIA
Praça General Tibúrcio, 80 – Praia Vermelha
Rio de Janeiro - RJ CEP: 22290-270

C837 Costa, Francisco Handrick Tomaz da
ROSAI – Uma Proposta de Representação do Modelo ROSA em Linguagem Lógica/ Francisco Handrick Tomaz da Costa. - Rio de Janeiro: Instituto Militar de Engenharia, 2005.
135 p. : il.

Dissertação (mestrado) - Instituto Militar de Engenharia – Rio de Janeiro, 2005.

1. Web Semântica. 2. Lógica.

I. Título. II. Instituto Militar de Engenharia

CDD 006.332

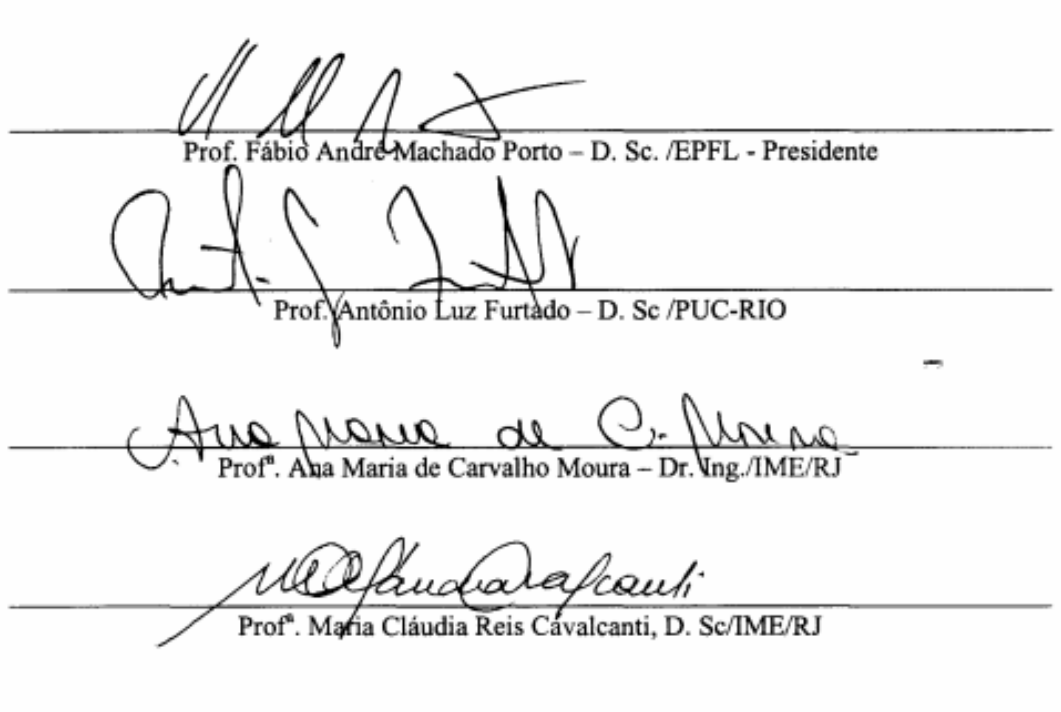
INSTITUTO MILITAR DE ENGENHARIA

FRANCISCO HANDRICK TOMAZ DA COSTA

ROSAI – UMA PROPOSTA DE REPRESENTAÇÃO DO MODELO ROSA EM LINGUAGEM LÓGICA

Dissertação de Mestrado apresentada ao Curso de Mestrado em Sistemas e Computação do Instituto Militar de Engenharia, como requisito parcial para a obtenção do título de Mestre em Ciências em Sistemas e Computação.
Orientador: Fábio André Machado Porto – D. Sc.

Aprovada em 21 de Fevereiro de 2005 pela seguinte Banca Examinadora:



Rio de Janeiro
2005

Dedico esse trabalho a todos que me apoiaram nessa caminhada, aos mestres, aos amigos, e principalmente aos meus pais e a minha noiva, que tanto me deram força nas horas difíceis dessa jornada.

AGRADECIMENTOS

Agradeço todos aqueles que me ajudaram direta ou indiretamente na realização desse trabalho. Aos meus pais por terem me proporcionado uma educação de qualidade na minha vida escolar, capaz de me fazer chegar até aqui, sem eles nada disso seria possível. Ao meu mestre e amigo Professor Francisco Vieira de Souza, da Universidade Federal do Piauí, por ter me ensinado o verdadeiro sentido da palavra aprender, e a todos os outros que efetivamente participaram de todo esse processo nesses últimos anos. A todos os amigos que passaram pelo nosso apartamento no Rio, Eduardo Albuquerque, Vladmir, Walter, Marcelo Loutfi, Fabrício Bissole, Fabrício Nogueira e Carlos André, obrigado pelo companheirismo e por todas os momentos de descontração que tivemos. Agradeço também a Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo apoio financeiro desses últimos dois anos. Ao Instituto Militar de Engenharia (IME/RJ) e a École Polytechnique Federal de Lausanne (EPFL), por ter acreditado em meu potencial e ter me oferecido total incentivo humano e material. Ao meu orientador e antes de tudo, um grande amigo, Fábio André Machado Porto, pessoa na qual serei eternamente grato por toda paciência e atenção que me dedicou, tanto no Brasil quanto na Suíça, verdadeiramente meu muito obrigado. A minha noiva Lucianna Oliveira, pelos momentos felizes que temos tido nesses últimos anos, obrigado por ter sido forte ao combater a saudade e a distância, e por toda força que me deste, você também foi de suma importância. Por fim, agradeço a todos os novos amigos que ganhei nesse período na cidade do Rio de Janeiro e em Lausanne, como também os mais antigos na minha cidade natal de Teresina e aos amigos virtuais de Internet, afinal são eles que fazem a nossa real riqueza e tornam possível que sonhos e desejos se tornem um dia realidade. A todos que torceram por mim, meu muito obrigado.

"**A descoberta consiste em ver o que todos viram e em pensar no que ninguém pensou**".

Edward Teller
Estados Unidos
[1908-2003]
Físico

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

RESUMO

O ROSA é um sistema para recuperação de objetos de aprendizado baseado na descrição semântica de seus conteúdos. Este trabalho tem por objetivo estender o modelo de dados ROSA, a partir de sua representação como uma base de conhecimento, na quais assertivas são definidas como fatos e restrições e regras são expressas através de fórmulas lógicas. As consultas presentes na estensão do modelo serão expressões lógicas com variáveis livres. Sua avaliação sobre a base de conhecimento permite a inferência de novos fatos, segundo a premissa do mundo aberto. A base de conhecimento é implementada em uma máquina Prolog que tratará fatos e regras, correspondendo ao domínio de aplicações suportadas pelo ROSA.

ABSTRACT

ROSA System is a system for Learning Objects retrieval, based on the semantic description of its contents. The purpose of this work is to extend the ROSA data model, starting from its knowledge base representation, where assertives are defined as facts and constraints, and rules are described as logic expressions with free variables. Evaluation in the knowledge base will enable new facts inference. The knowledge base is implemented in a Prolog machine that will evaluate facts and rules, corresponding to the application domain suppoted by ROSA.

1. INTRODUÇÃO

A *World-Wide-Web* faz parte de uma importante criação dos últimos tempos. Ela traz uma considerável quantidade de páginas e usuários que podem ser traduzidos como um enorme leque de informação direcionada para os mais diversos povos, culturas e nações.

Com todo o crescimento que a Web tem tido nos últimos anos, advindo do constante surgimento de novas páginas, a quantidade de informação presente na rede cresce exponencialmente, o que torna mais complexa a administração e organização das informações. Com isso, a busca por informações relevantes tem se tornado uma tarefa complexa e mal executada pelos mecanismos de busca. As ferramentas que existem hoje não possuem tecnologia suficiente para extrair informações seguindo uma certa linha de interpretação. Essa tarefa ainda se encontra a cargo dos humanos. Isso porque não existe uma estruturação clara na Web que possa facilitar o trabalho das ferramentas de busca.

Com base nessas premissas de dar semântica aos dados e tentar estruturá-los surgiu a idéia da Web Semântica, que tem como objetivo resolver esse problema de heterogeneidade da informação, extraindo suas diferenças sintáticas, estruturais e semânticas e com isso tentar facilitar a compreensão e a recuperação das informações contidas em um documento eletrônico. Assim, com a ajuda vinda da Web Semântica espera-se que o acesso às informações contidas nesses documentos eletrônicos seja mais preciso e eficiente e que estas informações possam agora ser interpretadas tanto por humanos quanto por máquinas, permitindo que estas trabalhem em cooperação com os seres humanos.

A forma como a Web Semântica se encontra hoje foi idealizada por Tim Berners – Lee, que afirma que ela deve ser vista como uma extensão da Web que vemos hoje, e que a informação deverá ter um significado bem definido, fazendo com que computadores e seres humanos possam trabalhar juntos [BERNNER-LEE, 2001]. Foi o mesmo Tim-Berners – Lee que também conceituou a WWW, URI's, HTTP e HTML e que atualmente faz parte de um grupo de pesquisa em um consórcio, o *World Wide Web Consortium* W3C [W3C, 2004].

Com o objetivo de iniciar o processo de criação da Web Semântica muitas tecnologias já foram criadas e já tiveram a sua importância reconhecida nesse processo. Como exemplo dessas tecnologias pode-se citar a *Extensible Markup Language* (XML) [XML, 2005] e o *Resourse Description Framework* (RDF) [RDF, 2004]. XML é uma linguagem que torna possível a representação de dados semi-estruturados, um formato bastante comum no mundo

real. Porém, o XML não é capaz de oferecer um certo significado a essas estruturas e sua função se restringe apenas a oferecer a capacidade de representar e transmitir dados semi-estruturados. A tarefa de expressar um significado a essa estrutura fica a cargo do RDF, que realiza a sua codificação atrás do uso de triplas, onde a mesma é composta por um sujeito, um predicado e por um objeto, utilizando-se da sintaxe do XML. Contudo, apesar de toda essa tecnologia existente, no campo da Web Semântica ainda existem muitos desafios a serem vencidos. Um dos principais pontos é a criação de uma linguagem que possa ser capaz de expressar o significado dos dados, e ao mesmo tempo capaz de definir regras para "raciocinar" sobre os mesmos, de uma maneira que possa produzir novos dados inferidos a partir dessas regras.

Vista essa idéia inicial sobre a Web Semântica, o que se percebe é que diversas aplicações podem se beneficiar de um modelo de dados no qual as representações semânticas de suas associações sejam computáveis. Tais associações são relevantes tanto na descrição dos objetos do domínio, quanto no suporte às consultas efetuadas pelos usuários e torna possível o desenvolvimento de sistemas que sejam capazes de efetuar consultas, que se valham de um modelo semântico de dados.

Um exemplo claro desse tipo de sistema é o ROSA (Repository of Objects with Semantic Access) [PORTO F., 2004] que foi desenvolvido como uma extensão do modelo de dados RDF e atende a aplicações voltadas para aprendizagem eletrônica, ou *e-learning.* O ROSA é responsável por gerenciar o conteúdo, armazenar e dar suporte ao acesso a certos tipos de objetos, conhecidos como objetos de aprendizagem, *Learning Object* (LO). Esses LOs são formados por um conjunto de características e propriedades denominados metadados. Com o auxílio dos LOs torna-se possível a criação de cursos eletrônicos à distância, oferecendo dessa maneira um suporte a uma área que muito tem crescido, a de Ensino a Distância (EAD). De posse de um determinado conjunto de LOs, um autor de um curso poderá estruturar todo o conteúdo didático necessário a ser utilizado em seu processo de aprendizado, educação ou treinamento.

No contexto da teoria proposta pela Web Semântica, o ROSA é um sistema que permite expressar associações entre os objetos LOs, de modo a prover uma contextualização entre os mesmos e, consequentemente, enriquecer semanticamente suas descrições. O sistema como descrito, se propõe a dar suporte a técnicas de processamento de consultas que irão explorar os metadados e os relacionamentos semânticos entre os LOs, auxiliando nas consultas efetuadas sobre o sistema. Um usuário poderá formular consultas mais expressivas em cima

de um modelo que ofereça uma forma eficiente de obter todos os LOs que julgar úteis no processo de elaboração de cursos ou treinamentos eletrônico.

1.1. MOTIVAÇÃO

O modelo ROSA, bem como todo o modelo de consultas definido, já se encontra em um estágio avançado de desenvolvimento. O sistema tem como suporte para consultas, uma álgebra baseada em outras já existentes, tais como a álgebra relacional, a orientada por objetos e a álgebra de consulta para RDF. Tomando como base essas álgebras foi elaborada a linguagem de consulta ROSAQL, que é capaz de exprimir consultas que exploram as características semânticas do modelo ROSA.

Entretanto, o modelo proposto ainda se encontra pobre do ponto de vista lógico. Nesse aspecto, a lógica poderia funcionar como importante ferramenta para o ROSA, já que a mesma é capaz de trazer significado e semântica a um determinado contexto, além de propor modelos que possam trazer conhecimento a uma determinada base de dados. Ao desenvolver esse tipo de modelo, a lógica cria representações parecidas com as desenvolvidas por pesquisadores da área de IA, que possuem um grande poder na execução de suas consultas.

Dessa maneira, a idéia é introduzir a lógica no modelo e assim estender a expressividade do ROSA, através de regras e do raciocínio sobre tais regras. Com isso espera-se ser possível expressar aspectos do modelo não representáveis, em um modelo fechado como o ROSA, como por exemplo, o tratamento de herança e de características implícitas de relacionamentos.

1.2. OBJETIVOS

O objetivo central deste trabalho de dissertação é a elaboração do novo modelo ROSA baseado em lógica. Tal representação deverá englobar tanto o modelo estrutural quanto as consultas baseadas nas operações algébricas. A execução de consultas se valerá da busca de regras no processo de inferência de novos fatos para o sistema, tornando-o bem mais poderoso. Com isso, esse novo modelo, que foi chamado de ROSAI (Repository of Objects with Semantic Access and Inference), será capaz de tomar decisões sobre fatos que já sejam do conhecimento do sistema, como também de novos fatos inferidos pelas regras elaboradas pelos usuários. Um pequeno exemplo de uma consulta possível a ser executada, uma vez realizada a extensão do modelo ROSA segue na FIG. 1.1. Nela aparece uma representação de

um esquema conhecido como mapa conceitual que será descrito no capítulo 3. Nesse exemplo a consulta é executada levando em conta não só os relacionamentos encontrados no mapa conceitual como também a regra presente. Segue o exemplo.

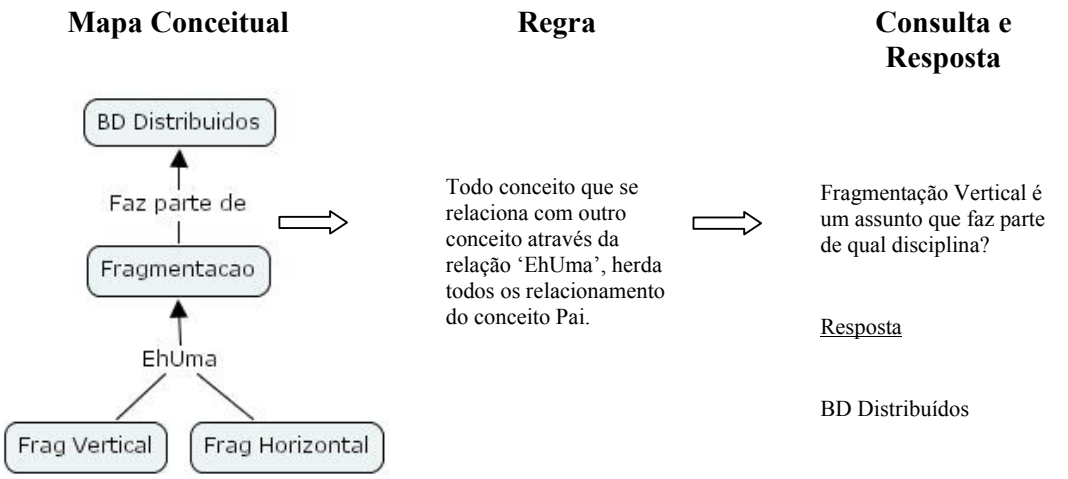


FIG. 1.1 – Exemplo de consulta com regra

1.3. ORGANIZAÇÃO DA DISSERTAÇÃO

O restante da dissertação está estruturado da seguinte forma. No capítulo seguinte serão descritos alguns trabalhos relacionados que utilizam a lógica como uma arma de apoio à consultas. No capítulo 3 será apresentado o estado da arte, onde será descrito o funcionamento das linguagens lógicas, dentre elas, o Prolog, que foi utilizada na implementação da proposta. Também nesse mesmo capítulo, discutir-se-á sobre a aplicação de lógica na Web Semântica e de que maneira a "lógica descritiva" se encaixa nesse contexto. O capítulo 4 apresenta a versão antiga do modelo ROSA permitindo uma comparação com a versão estendida baseada em uma linguagem lógica. A nova versão do ROSA, o ROSAI, é finalmente apresentada formalmente no capítulo 5. No capítulo 6 encontra-se o detalhamento de implementação do ROSAI e, finalmente, o capitulo 7 apresenta as conclusões finais com as contribuições e os trabalhos futuros.

2. TRABALHOS RELACIONADOS

Nessa seção serão apresentadas algumas iniciativas que utilizam a lógica como base para expressão e processamento de consultas, beneficiando-se principalmente do poderio da inferência lógica e do uso de regras em seus modelos. Dos trabalhos aqui apresentados será dado destaque basicamente a dois. O primeiro trata-se do TRIPLE [MICHAEL, 2002], um trabalho desenvolvido em conjunto pelo grupo de Banco de Dados da universidade de Stanford, através do pesquisador Stefan Decker [STEFAN, 2005], e do departamento de gerenciamento de conhecimento da mesma universidade citada, através do pesquisador Michael Sintek [MICHAEL, 2005]. Em seguida será descrito um outro trabalho, o FLORA [WEBCHANG ZHAOS, 2003], que se trata da sofisticação de uma linguagem lógica chamada F-Logic [MICHAEL KIFER, 1989]. Essa linguagem, juntamente com uma outra, denominada RDF, será apresentada em sessões seguintes, já que ambas servirão de base para o entendimento dos trabalhos aqui relacionados.

2.1. F-LOGIC

Aqui será descrita uma ligeira apresentação sobre a linguagem F-logic, que servirá de base para o entendimento dos trabalhos relacionados a essa dissertação.

De maneira semelhante a que acontece com outras linguagens dedutivas como o Prolog, o F-Logic possui suas declarações baseadas em fatos e regras. Os fatos formam uma base de dados que são sempre considerados verdadeiros e tudo que não estiver presente nessa base ou que não possa ser deduzido pelas regras é considerado um fato falso. Em cima dessa base de conhecimento e das regras, o F-logic realiza todas as suas operações de consultas. Essas consultas utilizam uma máquina de inferência dedutiva que pode inferir outros fatos não presentes na base de conhecimento.

A linguagem F-logic pode, através da declaração de fatos, definir classes, objetos, termos lógicos ou constantes. Quando se fala em termo lógico pretende-se expressar toda sentença que possui a sintaxe: p (a_1,a_2,a_3,...,a_n), onde n>0. Exemplo de fatos definidos como termos lógicos podem ser vistos a seguir através da definição de alguns pré-requisitos de disciplinas:

- Pré-requisito (banco dados, banco dados geográficos).

- Pré-requisito (banco dados, banco dados distribuídos).
- Pré-requisito (redes, banco dados distribuídos).

2.1.1. ORIENTAÇÃO A OBJETOS EM F-LOGIC

F-logic é uma linguagem que permite construções básicas de orientação a objetos. Essas construções são definidas através de 4 tipos de fórmulas atômicas, exibidas logo a seguir através da TAB. 2.1:

TAB 2.1 - Fórmulas Lógicas em F-Logic

	Formula atômica
1	O[M→V]
2	O[M→>{V1,...,Vn}]
3	O[M=>T]
4	O[M=>>{T1,..., Tn}]

A primeira fórmula atômica da tabela recebe o nome de átomo de dados e representa valores de métodos univalorados. Nessa mesma fórmula, o símbolo 'O' identifica a classe/objeto, enquanto o símbolo 'M' representa o nome do método e o 'V' o valor de retorno. Um exemplo desse tipo de representação pode ser visto a seguir: 'handrick [nacionalidade->brasileiro]'.

A segunda representação da tabela também possui a denominação de átomos de dados, porém nesse caso existe a representação de métodos multivalorados, ou seja, o método pode assumir mais de um valor. Isso não quer dizer que a semântica seja de igualdade como acontece com a primeira representação. Ela é de inclusão, isto é, o átomo: handrick [estudante→>{IME, EPFL}] não quer dizer que essas sejam as únicas instituições que o handrick estuda, mas que estão entre as instituições na qual ele estuda.

Ao contrário do que se pode esperar de uma linguagem voltada para orientação a objetos, o F-Logic não faz a distinção entre atributos e métodos. No F-Logic os métodos não possuem um código procedimental associado, as regras dedutivas é que fazem o papel dos valores requisitados por um certo método. No caso do 'M' visto na tabela acima, ele pode assumir um valor de qualquer termo, como uma função por exemplo. Nesse exemplo: handrick [bolsa (2003) → 700], o 'M' assumiu um valor de uma função, chamada bolsa, que tem como

parâmetro ‘2003’. Quando o ‘M’ representar uma constante, entende-se que o método torna-se um atributo.

O terceiro e o quarto tipos recebem o nome de átomos de assinatura e possuem em seu valor, o tipo de retorno do método. No caso do quarto tipo apresentado, o método deve possuir um valor de retorno igual a alguns dos valores declarados no conjunto. Sintaticamente, nos casos 2 e 4, quando existir um conjunto com apenas um elemento, as chaves poderão ser dispensadas.

A definição de hierarquia é representada na linguagem F-Logic da seguinte maneira, conforme a TAB. 2.2.

TAB. 2.2 - Instância e Sub-Classe em F-Logic

Instância	Sub-Classe
O:C	C::D

Na primeira coluna define-se que o ‘O’ é uma instância de ‘C’ e na outra que ‘C’ é uma sub-classe de ‘D’. O F-Logic permite que ocorra herança múltipla, ou seja, assim como um objeto pode ser instância de várias classes, uma classe pode herdar de várias classes.

Também existe uma definição eficiente de agrupamentos de átomos no F-Logic. Esse agrupamento recebe o nome de F-Moléculas. A sintaxe de F-Moléculas torna a apresentação dos fatos bem mais compreensiva. Um exemplo desse agrupamento pode ser visto logo a seguir na TAB. 2.3.

TAB. 2.3 - Transformação de átomos em molécula

Átomos	Molécula
handrick:pessoa handrick [idade→26] Handrick [estudante→>{IME,EPFL}	handrick:pessoa [idade→26, estudante→>{IME,EPFL}].

2.1.2. REGRAS EM F-LOGIC

Em F-Logic as regras apresentam-se na seguinte sintaxe: ‘conclusão :- corpo’. A conclusão pode ser um átomo ou molécula e o corpo pode ser uma conjunção de átomos ou

moléculas. Nas regras podem aparecer variáveis que são representadas por seqüências de caracteres iniciadas por uma letra maiúscula ou *underscore*. Elas aparecem em qualquer lugar de uma regra. A verificação de uma regra que têm variáveis consiste em verificar para quais valores das variáveis as premissas se tornam verdadeiras (de acordo com os fatos da base). Também são permitidas operações aritméticas através do operados "is", que atribui um valor a uma variável, e outros tipos de operadores como o de comparação e de igualdade. Seguem alguns exemplos: (X is 1+2), comparação (X>3), igualdade (X=Y).

O exemplo a seguir concatena um conjunto de conceitos vistos até o presente momento como o de instância e o de regras. Segue o exemplo:

```
X: poliglota:- X [línguas→Y], Y>3.
```

Nesse exemplo é visto uma regra que impõe que todo poliglota é um sujeito que possui conhecimento de mais de três línguas.

2.1.3. CONSULTAS EM F-LOGIC

Consultas em linguagens lógicas como o F-logic, são regras sem conclusões. No F-Logic elas são representadas com a sintaxe: '?- corpo'. O corpo é entendido como sendo uma conjunção de átomos e moléculas, negadas ou não. Uma consulta, portanto, será verdadeira se todas as condições contidas no corpo puderem ser provadas pela aplicação da base de regras sobre a base de fatos. As consultas que não possuem variáveis, ou com variáveis *don't care*, indicadas por *underscore*, apenas retornam verdadeiro ou falso. Consultas com variáveis normais, além disto, retornam todo o conjunto de valores que as mesmas podem assumir. O princípio é igual ao de regras, com a diferença que não existe uma conclusão a ser adicionada à base. É importante ressaltar que a principal diferença entre a máquina de inferência de F-logic e Prolog é a incorporação de herança.

Finalizando a breve introdução sobre F-Logic, será apresentado um pequeno exemplo que mostra todos os conceitos básicos aqui exibidos através de um pequeno programa em F-logic. Segue o exemplo:

TAB. 2.4 - Exemplo da descrição de veículos em F-Logic

<table>
<tr><th>Código</th><th>Consulta 1</th><th>Consulta 2</th></tr>
<tr><td rowspan="2">veiculo [].
veiculo_terrestre::veiculo.
carro [rodas⇒integer, rodas→4].
Gol:carro
Jumbo:avião
%definição de regra dinâmica de herança.
X::veiculo_terrestre:-X[rodas→Y], Y>0</td><td>%Relação de herança dinâmica.
?- X::Y.</td><td>%Expressão
de caminho
X.rodas > 0.</td></tr>
<tr><td>Resposta 1</td><td>Resposta 2</td></tr>
<tr><td></td><td>X = carro
Y = veiculo
X = carro
Y = veiculo_terrestre
Inferido pela regra dinâmica de herança.
X = veiculo_terrestre
Y = veículo
Presente na base.</td><td>X = Gol</td></tr>
</table>

2.2. RDF – RESOURCE DESCRIPTION FRAMEWORK

RDF é uma recomendação da W3C [W3C, 2004] para a definição e uso de dados que descrevem dados ou metadados na Web. Ele utiliza a sintaxe do XML para expressar o significado desses dados. É através do RDF que é possível haver a troca de informações compreensíveis pela Web e com isso, criar aplicações que possam interoperar entre si.

Enquanto o XML é uma linguagem que permite expressar a estrutura, o RDF surge como o responsável para especificar a semântica dos dados, realizando isso através das triplas do tipo <objeto,atributo,valor>, descritas na forma A(O,V), onde O representa o objeto que tem o atributo A com o valor V. Exemplos dessa representação podem ser vistos a seguir:

temNome(‘www.ime.eb.br/estudante/idSC03115’,’Handrick’).
estudanteDo (‘www.ime.eb.br/estudante/idSC03115’,’www.de9.ime.eb.br’).

Os exemplos anteriores também podem ter a sua representação descrita sob forma de grafos RDF. Assim sendo, a representação desse grafo é mostrada na FIG. 2.1.

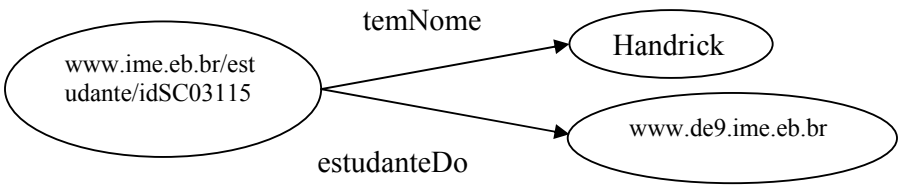


FIG. 2.1 – Representação do RDF através de Grafo

Como visto na FIG. 2.1, o exemplo pode ser descrito na forma de um grafo direcionado. Nesse grafo aparecem três tipos básicos de objetos, que são descritos como recursos, propriedades e sentenças.

Os recursos representam o que será descrito pelo RDF. No caso da FIG. 2.1, o recurso é visto como uma página Web. A propriedade é tudo aquilo que é usado para caracterizar, especificar, criar um atributo ou um relacionamento de um recurso. Por fim, a sentença representa a associação final entre o recurso especificado, a propriedade, e o valor dessa propriedade para esse recurso. Assim sendo, a TAB. 2.5 a seguir descreve todos os objetos presentes na FIG. 2.1.

TAB. 2.5 – Apresentação dos Recursos, Propriedades e Sentenças no RDF

Recurso	www.ime.eb.br/estudante/idSC03115
Propriedade 1	TemNome
Sentença 1	O estudante de identificação SC03115 tem nome Handrick
Propriedade 2	EstudanteDo
Sentença 2	O estudante www.ime.eb.br/estudante/idSC03115, estuda no www.de9.ime.eb.br

A identificação dos recursos, das propriedades e das sentenças pelo RDF são realizadas através de URI's [URIs, 2005], Uniform Resource Identifiers. Isso é feito para assegurar que aquelas palavras descritas tenham uma identificação única. É importante que se use URI's diferentes para diferentes conceitos, de maneira que se possa garantir a consistência das inferências realizadas. Um exemplo que torna isso claro pode ser visto na utilização do conceito 'lima', que pode ser entendido como uma fruta, ou como a capital do Peru, ou até mesmo como o nome de uma família. Assim, para assegurar a perfeita semântica, de forma

que cada conceito seja utilizado de forma distinta, é necessário utilizar diferentes URI's para cada um desses conceitos. A unicidade da relação entre conceito e URI's garante a unicidade do significado das triplas.

A seguir é apresentado uma pequena representação de como o RDF é utilizado no contexto do XML, tomando-se como base mais uma vez a FIG. 2.1.

```
<?xml version="1.0"?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:s="http://description.org/schema/">
<rdf:Descriptionabout=" www.ime.eb.br/estudante/idSC03115">
<s:temNome>Handrick</s:temNome>
</rdf:Description>
</rdf:RDF>
```

Além desse modelo, o RDF também possui um esquema que define um vocabulário particular que será usado pelos atributos, além de especificar os tipos de objetos que podem ser aplicados a esses atributos. Esse esquema e o modelo RDF irão trabalhar juntos para criar um mecanismo simples de representação do conhecimento na Web.

O modelo RDF , pode ser usado então para representar conhecimento, através de uma interoperabilidade sintática e estrutural. Esse esquema RDF é o responsável pela interoperabilidade sintática, já que ela provê regras de sintaxe através de uma gramática. Quanto a interoperabilidade estrutural ela é apresentada através dos dados e da especificação de tipos e possíveis valores para cada forma de representação <objeto,atributo,valor>. Maiores detalhes sobre RDF serão vistos no capítulo seguinte.

2.3. TRIPLE – VISÃO GERAL

TRIPLE [MICHAEL, 2004] é uma linguagem voltada para a Web Semântica que tem como objetivo realizar consultas, inferências e transformações sobre RDF. O núcleo da linguagem TRIPLE é baseado na lógica sintaticamente estendida para suportar primitivas do RDF, como namespace, recursos e declarações, daí o nome da linguagem. É uma linguagem que se baseia em regras para a Web Semântica, possuindo várias características herdadas do F-Logic.

Muitas das linguagens de consultas voltadas para RDF permitem expressões semânticas de consultas para RDF Schema, porém o TRIPLE permite expressar essas consultas utilizando regras em outras linguagens superiores como Topic Maps, UML e DAML+OIL.

Isso é bastante útil para linguagens na qual isso não é facilmente possível. O resultado é que o TRIPLE resulta em uma linguagem híbrida, que suporta tanto expressões semânticas quanto regras.

2.3.1. NAMESPACE E RECURSOS NO TRIPLE

Nas sessões que se seguem são apresentadas algumas das principais características presentes em TRIPLE. Como mencionado anteriormente, o TRIPLE possui algumas das primitivas presentes na linguagem RDF. Assim, esta linguagem possui um suporte especial para *namespace* e identificadores de recursos. N*amespaces* são declarações via construtores da forma: nsabrev:=namespace. Um breve exemplo a seguir demonstra a sintaxe:

```
rdf:= http://www.w3.org/1999/02/22-rdf-syntax-ns#
```

Os recursos são escritos na forma nsabrev:nome, onde nsabrev significa a abreviação do namespace e o nome significa o nome do local dos recursos.

Da mesma forma que acontece com os namespaces, os recursos também podem ser abreviados, como a seguir:

```
Rec_abrev:=nsabrev:nome
```

2.3.2. DECLARAÇÕES E MOLÉCULAS

As declarações em uma tripla em RDF são inspiradas no objeto de sintaxe do F-Logic, como apresentado anteriormente: sujeito [predicado→objeto].

Várias declarações para o mesmo objeto são possíveis de serem feitas, sendo abreviados na forma de moléculas, como no F-logic. Exemplo:

```
Handrick [tem_idade→25;faz_mestrado→ime;...]
```

Seguindo esta mesma idéia, é possível aninhar declarações e moléculas em vários níveis, como segue:

```
Handrick [faz_mestrado→ime [endereco→praia_vermelha]]
```

2.3.3. EXPRESSÕES DE CAMINHO NO TRIPLE

Sabe-se que expressões de caminho são bem eficientes em linguagens orientadas a objetos, quando se deseja navegar através de uma determinada estrutura. O TRIPLE permite o uso desse tipo de expressão ao invés de usar sujeito, predicado e definições de objetos. Um exemplo prático de expressões de caminho pode ser visto logo a seguir.

```
Handrick.faz_mestrado.endereço
```

Nesse caso, a consulta deseja conhecer o endereço da instituição na qual o Handrick realiza o seu curso de mestrado. Seguindo o exemplo do item anterior a resposta viria como: 'praia_vermelha'.

2.3.4. PEQUENO EXEMPLO EM TRIPLE

Nessa seção será apresentado um simples exemplo da utilização do TRIPLE. A idéia aqui é demonstrar a sua utilização e não exibir um exemplo que apresente com profundidade todos os recursos e tipos de consultas possíveis de serem executadas nessa linguagem. Para tanto se recomenda uma pesquisa nas referências aqui citadas.

O exemplo aqui apresentado será demonstrado no contexto da linguagem DAML+OIL. Nele será demonstrada uma simples definição de uma ontologia sobre animais. Nessa definição, existe uma descrição do que são animais carnívoros e herbívoros. Na descrição afirma-se que esses tipos de animais pertencem a conjuntos disjuntos, o que é uma verdade. Nesse mesmo código também existe a definição de animais do tipo ovíparos, e nessa definição, estes são colocados como animais fazendo parte de uma sub-classe de carnívoros e herbívoros, o que é uma definição errada, já que não existem animais que sejam herbívoros e carnívoros ao mesmo tempo, devido a definição de disjunção. Nesse código existe uma consulta que tenta identificar esse tipo de incompatibilidade de conceitos que irá justamente tentar contestar essa ontologia, quando a mesma faz referencia aos ovíparos. Segue o código a seguir:

```
rdf := "http://www.w3.org/1999/02/22-rdf-syntax-ns#" .
rdfs := "http://www.w3.org/TR/1999/PR-rdf-schema-19990303#".
```

```
daml := "http://www.daml.org/2001/03/daml+oil#".
animais := "http://www.example.org/animals#".

// ontologia
@animais:ontology {

  animais:Animal[rdf:type -> daml:Class].
  //definição de herbívoros
  animais:Herbivoro[rdf:type -> daml:Class;
    daml:subClassOf -> animais:Animal].
  //definição de carnívoros
  animais:Carnivoro[rdf:type -> daml:Class;
    daml:subClassOf -> animais:Animal;
    daml:disjointWith -> animais:Herbivoro].
  //definição dos Ovíparos
  animais:Oviparo[rdf:type -> daml:Class;
    daml:subClassOf -> animais:Herbivoro;
    daml:subClassOf -> animais:Carnivoro].
}
// regras de insatisfação
FORALL Ont @check(Ont){
    FORALL C unsatisfiable(C) <-
        C[daml:subClassOf ->
        daml:Nothing]@daml_oil(Ont).
}
//consulta
    unsatisfiable(animais:Oviparo)@check(animais:ontology).
```

Na última linha, a consulta apresenta a função *unsatisfiable*, que faz a checagem sobre o que foi descrito sobre ovíparos na ontologia. Ele irá constatar que o mesmo está definido de forma errada, segundo a declaração de disjunção feita sobre os carnívoros e herbívoros:

```
daml:disjointWith -> animais:Herbivoro.
```

Esse exemplo mostra como é possível fazer a representação de uma ontologia através do TRIPLE, bem como a maneira como ele infere uma conclusão, através de uma regra pré-existente. No próximo item será apresentado mais um dos trabalhos aqui relacionados, que utilizam a lógica em suas execuções.

2.4. FLORA

FLORA [WEBCHANG, 2004] pode ser visto como uma sofisticação do interpretador lógico Prolog XSB [XSB, 2005]. A principal função do FLORA é realizar a tradução de um programa escrito em F-Logic para programas escritos em uma linguagem lógica Prolog, voltado para o interpretador XSB, que realiza a sua interpretação logo após a tradução ter sido realizada.

O FLORA pode ser encontrado como parte da distribuição oficial do interpretador XSB, fazendo parte de um pacote do mesmo. Atualmente o FLORA só é capaz de fazer a execução de suas traduções compatível com a plataforma XSB, tornado ambos bastante dependentes um do outro.

A linguagem suportada pelo FLORA é um dialeto do F-logic compatível com um sistema de F-logic baseado em C++ da Universidade de Freiburg, o FLORID [FLORID, 2004].

2.4.1. FUNCIONAMENTO DO FLORA

Visto que a principal função do FLORA é transformar um programa em F-Logic para um lógico XSB, ele necessita então de receber esse arquivo F-logic como argumento de entrada para logo em seguida realizar a sua tradução. Se nesse arquivo inicial existir algum tipo de consulta, a mesma será executada imediatamente pelo XSB.

Uma das principais funções propostas pelo FLORA, é que seja possível a execução de consultas em F-Logic mesmo depois de ter ocorrido a tradução. Um exemplo disso pode ser visto logo a seguir. Nesse exemplo uma consulta em F-logic poderá ser realizada sobre um arquivo que outrora estava escrito em F-logic.

```
Flora ?- X..empregado [      % Quais os empregados.
self → K;                    % Lista-los por nome.
salário →>                   % Que tem salário.
{S: >500}]                   % Acima de 500.
```

O FLORA irá analisar gramaticamente e fazer a avaliação dessa consulta da mesma maneira como se estivesse avaliando consultas feitas no arquivo fonte.

Para entender como funciona essa tradução realizada pelo FLORA, será apresentada uma molécula que representa fatos como idade, nome dos filhos e salário de um objeto Maria:

```
Maria: empregada [idade→29;criança →>{Tim, Leo};salário@(2003) →2000].
```

Qualquer molécula pode ser decomposta em conjunções menores e mais simples, denominados átomos, como visto anteriormente. Esses átomos podem ser representados diretamente usando a sintaxe Prolog. Para cada átomo do F-logic é usado um predicado

Prolog diferente. Como exemplo é exibido o código fonte lógico gerado pela tradução da molécula acima:

```
'_$_$_flora_isa'(Maria,empregada).
'_$_$_flora_fd'(Maria,@(idade),29).
'_$_$_flora_fd'(Maria,@(criancas),tim).
'_$_$_flora_fd'(Maria,@(criancas),leo).
'_$_$_flora_fd'(Maria,@(salário,2003),2000).
```

2.4.2. CONSIDERAÇÕES SOBRE O FLORA

FLORA é um projeto em desenvolvimento que já obteve grandes progressos. Muitas de suas características ainda se encontram em fase de teste, não se podendo afirmar que já se trate de um projeto maduro.

Uma dessas novas características seria o de abolir a restrição de uso do operador *not*, que atualmente só é usado em predicados. Também é proposto criar funções de agregação bem como criar operações do tipo *if-then-else*, com a finalidade de se criar programas mais legíveis. O FLORA também promete vir em suas próximas versões com suporte a sintaxe de lógica transitiva, além de aceitar declarações para Banco de Dados, a exemplo do que acontece com a interface de Banco de Dados do XSB.

2.5. CONSIDERAÇÕES FINAIS

A linguagem F-logic apresentada aqui possui sua base de conhecimento, formada por fatos representados por átomos e F-moléculas, bem como a representação de regras e consultas. O TRIPLE por possuir características herdadas do F-Logic, é um dos trabalhos relacionados que utiliza o poder de inferir novos fatos a sua base de conhecimento, realizando a análise das regras presentes. Essas caracterísiticas são as mesmas que estarão presentes quando se realizar a representação do modelo ROSA através de uma linguagem lógica.

A proposta apresentada pelo FLORA também leva em consideração a mesma linguagem F-Logic, sendo que dessa vez ela é responsável por realizar a tradução dessa linguagem e todas as suas características, para uma lingugem lógica interpretável por uma máquina Prolog, que por fim irá realizar a mesma tarefa de inferir novos fatos com as regras.

Assim, os trabalhos apresentados nesse capítulo são fundamentados em inferência lógica para realizar suas consultas. Ambos os trabalhos possuem aspectos semelhantes ao que será apresentado nos próximos capítulos 5 e 6, quando for apresentada a proposta formal do ROSAI, bem como seus detalhes de implementação.

3. LÓGICA E WEBSEMÂNTICA

Nessa sessão serão exibidos alguns aspectos do estudo da lógica, que demonstram que a mesma poderá ser de grande utilidade na tarefa atribuída a Web Semântica, a de trazer significado e semântica a um determinado conteúdo existente na Web.

Estudos nessa direção mostram que já existem modelos que se propõem a trazer conhecimento a uma determinada base de dados.

Como exemplo de um desses modelos de representação de conhecimento, temos uma representação conhecida como "rede semântica". Uma rede semântica é vista como um formalismo para representar fatos e relacionamentos entre estes por meio de relações binárias. A FIG. 3.1 a seguir mostra um exemplo dessa representação. Nela, BD, BD2 e TÓPICOS representam objetos. "Pré-requisito" representa a relação entre os objetos BD e BD2, enquanto "Envolve" representa uma relação entre BD e TÓPICOS.

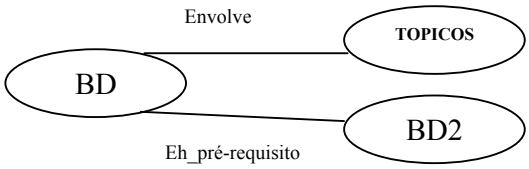


FIG. 3.1 - Uma rede semântica simples

Os relacionamentos individuais são conectados formando uma rede, onde os objetos como "BD", são representados uma única vez. Nesse caso, a rede semântica é uma forma muito boa de representar uma relação binária, através de uma notação gráfica bastante simplificada. As redes semânticas representam bem o raciocínio humano, já que o mesmo se baseia em associações lineares.

É possível fazer a representação das redes semânticas através de uma linguagem de programação baseada em lógica, como o Prolog. Nele, as relações binárias poderiam ser representadas através de fatos, implementando cada relação e repetindo os nomes dos objetos como segue a seguir.

```
prerequisito(bd,bd2).
envolve(bd,topicos).
```

3.1. ONTOLOGIAS

Como apresentado em capítulos anteriores, atualmente existe uma necessidade de se aprimorar as ferramentas de busca para Web. Nesse sentido, várias pesquisas têm sido desenvolvidas e com elas têm surgido algumas ferramentas, que já eram conhecidas no passado em outras áreas, mas que ainda não eram exploradas na Ciência da Computação. Ontologia é um desses termos e segundo o dicionário Merriam Webster [MERRIAM, 2005], trata-se de um termo já estudado e definido desde 1721, através de abstrações filosóficas, e também através de descrições formais da matemática. O termo ontologia denota filosoficamente, uma definição sobre a natureza das coisas ou a sua existência. No campo da Ciência da Computação, especificamente em Inteligência Artificial, ela pode ser interpretada como o conjunto de entidades com suas relações, restrições, vocabulários e axiomas. Segundo o pesquisador Tom Gruber [GRUBER, 1993], uma ontologia define um domínio, ou mais formalmente, especifica uma formalização a cerca desse domínio, organizada em hierarquia de conceitos ou taxonomias.

As pessoas para interagir com outras, ou entender certos processos devem possuir na maioria das vezes uma certa noção ou conceito do significado de termos. De maneira similar, os agentes computacionais precisam ter uma especificação de seus dados de entrada e de saída, ou seja, precisam também entender o significado dos termos que estão lhe dando. Seguindo essa idéia, a ontologia surge como uma solução para prover uma concreta especificação do nome dos termos e de seus significados para sistemas computáveis. Dentro dessa linha de pensamento de prover uma conceitualização de termos, surge um leque de potenciais interpretações para a mesma.

Especificamente para o caso da Web, a ontologia pode seguir por diferentes definições de acordo com o detalhamento de sua especificação. Uma das mais simples noções para uma possível ontologia pode ser a de um vocabulário controlado, onde o mesmo seria definido como uma lista finita de termos. Essa lista iria funcionar como um catálogo capaz de prover uma interpretação não ambígua de termos. Um exemplo da interpretação do catálogo pode ser vista quando se utiliza o termo “manga”, que deve denotar exatamente sempre um mesmo identificador interno, como por exemplo: “34”, sem nenhum significado semântico.

Seguindo por uma especificação mais detalhada, a ontologia também pode ser observada como um glossário, contendo uma lista de termos e seus significados. No glossário, os significados dos termos são vistos como especificações típicas de declarações feitas em linguagem natural, proporcionando uma semântica aos termos que pode ser compreendido e

interpretado por seres humanos. Entretanto, esses glossários não estão livres de interpretações ambíguas, e devido a isso não são adequados para serem usados por agentes computacionais e provavelmente não atende ao requisito de serem interpretáveis por máquinas.

Nesse contexto também surgem os tesauros, que podem prover uma semântica adicional no tratamento de relações entre termos. Eles constituem um dicionário de sinônimos que podem ajudar na tarefa de criar semântica. Essa tarefa é realizada informando quais os termos que possuem significado semelhante. Em muitos relacionamentos no qual eles atuam podem existir interpretações não ambíguas por parte de agentes computacionais. Porém os tesauros não provêem tipicamente uma hierarquia explícita de termos, embutida na noção de especialização e generalização. Um exemplo desse tipo de hierarquia de termos pode ser visto nos termos "transporte coletivo" e "metrô". Neles o termo "metro" é uma especialização de "transporte coletivo".

A formalização do que vem a ser uma ontologia nem sempre é uma tarefa simples. Muitas pessoas consideram que a categorização de catálogo, glossários e tesauros, são suficientes para serem ontologias. Porém outras acreditam que antes de qualquer coisa é preciso ter uma hierarquia explícita de classe incluída [CORCHO, 2002]. Observa-se que na maioria das vezes, as ontologias existentes são formadas por termos que permitem interpretações não ambíguas, onde esses termos atendem a um controle finito, organizado em uma hierarquia de classe. Na prática, ao se elaborar ontologias é possível criar um vocabulário controlado em um determinado domínio de aplicação. Este torna-se bastante útil para o entendimento desse domínio por partes de diferentes usuários e agentes, já que ambos podem compartilhar desse mesmo vocabulário, e usufruir de maior interação. Um benefício direto desse tipo de interação, por exemplo, pode surgir através da integração de diferentes grupos de pesquisas de áreas semelhantes ou afins, que mesmo em localizações geográficas distintas, podem trabalhar em cooperação utilizando uma mesma ontologia.

Assim sendo, a construção de ontologias constitui uma técnica eficiente de recuperação de informações em meios heterogêneos como a internet, representando uma maneira adequada de atender as necessidades de reuso de informações e conhecimento.

Ontologias também podem ser vistas como visão unificadora de conhecimento, capaz de trazer o consenso sobre determinados conceitos de domínios específicos. Assim, o seu papel de elemento estruturador das informações pode historicamente ser comparado ao armazenamento de conhecimento provido por enciclopédias e bibliotecas para seres humanos,

já que dessa vez o conhecimento pode trafegar por sistemas capazes de manipulá-los, pois os mesmos agora seguem uma determinada organização e contextualização.

3.2. MAPAS CONCEITUAIS

A apresentação de mapas conceituais nessa seção se deve ao fato desses serem adequados na elaboração de modelos capazes de reproduzir objetos ou conceitos e seus respectivos relacionamentos, gerando portanto, uma forma organizada de representar conhecimento.

Do ponto de vista gráfico, mapas conceituais são constutuidos de conceitos, representados por círculos, e relacionamentos, ligando estes conceitos, representados por arestas [NOVAK, 2003]. Um rótulo identifica o tipo de relação estabelecida entre dois conceitos. Esse rótulo pode ter a sua descrição especificada por palavras ou símbolos. Dessa maneira, os mapas conceituais podem ser entendidos como grafos, onde os nós representam os conceitos e os arcos relacionamentos.

Graficamente, os mapas conceituais se disponibilizam de forma hierárquica, onde os conceitos mais abrangentes encontram-se disponíveis no topo do mapa e os mais específicos são disponibilizados hierarquicamente abaixo [NOVAK, 1984]. Como exemplo de mapa conceitual, apresenta-se a FIG. 3.2, que reproduz um mapa conceitual de uma banco de dados geográfico.

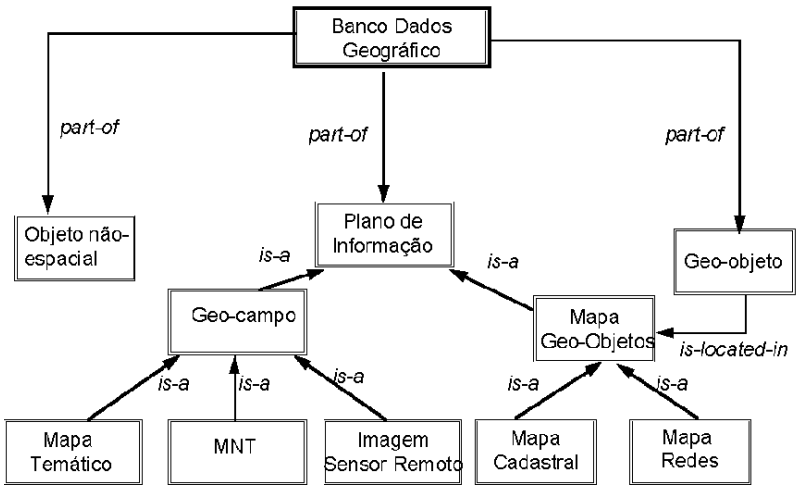


FIG. 3.2 – Mapa Conceitual de um Banco de Dados Geográfico

A construção de mapas conceituais é direcionada a um determinado domínio particular, e através de sua elaboração torna-se possível conhecer e realizar análises sobre um determinado domínio, ajudando com isso a solucionar questões particulares do mesmo.

No próximo item será feita uma abordagem sobre a Lógica Descritiva, que é uma linguagem de representação do conhecimento baseada em lógica, que muito tem contribuído para a área de raciocínio sobre modelos.

3.3. LÓGICA DESCRITIVA

A lógica tem como principal objetivo, construir representações do conhecimento. Essa área procura criar teorias e sistemas capazes de expressar estruturas de conhecimento bem como acessar e raciocinar em cima dessas estruturas. Nesse contexto, a Lógica Descritiva [DESCRIPTION LOGICS, 2004] surge como uma importante linguagem de representação do conhecimento, capaz de oferecer essa representação através da construção de ontologia.

A Lógica Descritiva oferece uma boa representação lógica para sistemas tradicionais baseado em Frames e redes semânticas [REDES SEMÂNTICAS, 2004], em representação de modelos baseados em orientação a objetos, e em modelos de dados semânticos e de sistemas de tipos.

A seguir é apresentado um breve histórico sobre a evolução da representação do conhecimento e nos itens que se seguem discute-se sobre a linguagem para representação do conhecimento.

3.3.1. HISTÓRICO

As pesquisas em representação do conhecimento e do raciocínio têm sido alvo de interesse crescente de pesquisadores, com a idéia de se aperfeiçoar em métodos de representação, que possam eventualmente ser utilizados para construir aplicações inteligentes. Quando se fala em inteligência, subtende-se a capacidade de um sistema poder encontrar características implícitas em cima de uma representação de conhecimento explícito. Tais sistemas são caracterizados como sistemas baseado em conhecimento.

Pesquisas no campo de representação do conhecimento começaram a crescer na década de 70, quando a área começou a ganhar grande popularidade [WOODS, 1975] [BRACHMAN, 1977] [BRACHMAN, 1979]. Essas pesquisas cresceram seguindo duas categorias distintas. Uma possuía um formalismo baseado na lógica, na qual a representação do mundo era feita através de predicados e a análise sobre esses predicados poderia capturar alguns aspectos sobre o mundo. Também existia a representação do conhecimento que não se baseava em lógica. Um exemplo desse sistema eram as estruturas de redes e as representações baseadas em regras. Como representante dessa categoria de representação temos as redes semânticas e os frames [MINSKY, 1981]. Porém, essas representações eram usadas para certas áreas específicas e precisavam ser tratadas como ferramentas de uso geral. Os sistemas que se baseavam em lógica, utilizavam a lógica de primeira ordem, e eram bem mais abrangentes e poderosos, o que deixava a proposta bem mais atrativa do que aqueles não baseados em lógica. Em um sistema baseado em lógica, a linguagem representativa geralmente era uma variação do cálculo de predicados da lógica de primeira ordem, cujo raciocínio nesses sistemas significava a verificação de seqüências lógicas.

Mas apesar dos sistemas baseados em lógica serem mais abrangentes, os sistemas baseados em rede eram originalmente considerados mais atrativos e mais eficientes do ponto de vista prático, do que os sistemas lógicos. Porém, eles não eram completamente satisfatórios, pelo fato de serem carentes de caracterização semântica. A questão que surgiu era como criar uma semântica para essas estruturas, particularmente para as redes semânticas e para os frames? Pensando em solucionar esse problema, surgiu a idéia de explorar a noção

de estruturas hierárquicas. Esse estudo ajudou a criar uma representação mais fácil e um mecanismo de raciocínio eficiente.

Um importante passo nessa direção foi reconhecer que era possível dar semânticas aos frames, desde que lhes fossem atribuidas uma lógica de primeira ordem. Os elementos básicos da representação eram caracterizados como predicados unários e binários, na qual o primeiro denotava o conjunto de indivíduos e o segundo a relação entre esses indivíduos. Mas a solução da semântica para os frames através da lógica de primeira ordem era bastante genérica. De fato, embora a lógica fosse a base natural para especificar um significado para essas estruturas, frames e redes semânticas não requerem todo os mecanismos da lógica de primeira ordem, mas apenas uma parte dela. Além disso, diferentes características das linguagens de representação, poderiam levar a diferentes fragmentos da lógica de primeira ordem.

Uma conseqüência importante desse fato foi o reconhecimento de que a forma típica de raciocínio usado em estruturas baseadas em representação, poderiam ser acopladas à técnica de raciocínio específico, sem necessariamente requerer provas de teoremas da lógica de primeira ordem. Além disso, chegou-se a conclusão de que raciocinando em diferentes fragmentos da lógica de primeira ordem, pode-se chegar a diferentes problemas computacionais de diferentes complexidades.

Logo após essa descoberta, foram iniciadas pesquisas na área da Lógica Descritiva com o título de sistemas terminológicos, com a idéia de deixar claro que a linguagem representativa foi usada para estabelecer a terminologia básica adotada no domínio do modelo. Mais tarde, a ênfase foi no conjunto de formalismo conceitual admitido na linguagem, surgindo portanto o nome de linguagem conceitual. Poucos anos atrás, quando as atenções se voltaram à propriedade dos sistemas lógicos subjacentes, o termo Lógica Descritiva tornou-se popular.

3.3.2. A Lógica Descritiva e Redes Semânticas

Como apresentado nos itens anteriores, as redes semânticas são vistas como uma forma de representar o conhecimento através de uma rede. Os elementos que compõem uma rede semântica são os seus nós e links, onde os nós caracterizam um conjunto ou classe de indivíduos e os links fazem a ligação entre eles. Algumas redes semânticas possuem relacionamentos mais complexos, que são representados por nós. Nesse caso, os nós

representam conceitos. Existem também redes semânticas mais avançadas que tratam tanto objetos quanto conceitos como nós.

Assim sendo, será aprofundado um pouco mais sobre as redes semânticas e o seu uso com a Lógica Descritiva. Para isso, apresentamos mais um pequeno exemplo que nos servirá de apoio no entendimento.

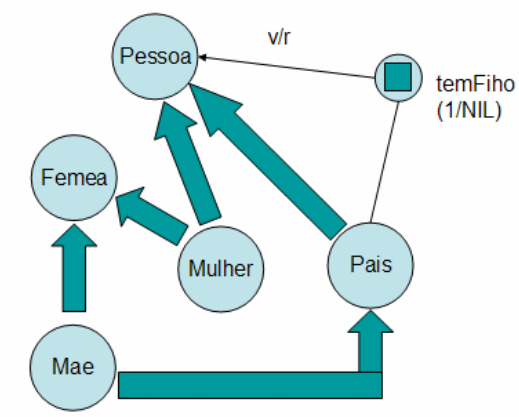


FIG. 3.3 - Uma rede semântica sobre parentes

Essa rede semântica representa os conceitos pessoas, pais, crianças, entre outros. A FIG. 3.3 representa o que muitas vezes é chamado de taxonomia. Na FIG., o link que existe entre *pessoa* e *mulher* mostra semanticamente que as mulheres são pessoas, o que muitas vezes é conhecido como uma relação "é um".

Essa relação "é um" é uma das mais importantes, pois cria a definição de herança sobre os conceitos e suas propriedades. Assim sendo, quando um conceito é mais específico que outro, ele herda as propriedades do mais genérico.

Nesse contexto, a Lógica Descritiva entra como uma ferramenta capaz de fazer a representação desse e outros tipos de relacionamentos. Ela permite interpretar a propriedade *temfilho* do conceito de *Pais*, como se fosse uma regra. Além disso, essa regra ainda possui o que se chama de restrição de valor, que pode ser observado pela legenda v/r. Essa legenda representa a limitação dos tipos de objetos que podem suprir essa regra. Possui também uma expressão de restrição (1,NIL), onde o primeiro número limita a quantidade mínima de filhos que a regra impõe, e a segunda representa a limitação máxima imposta pela regra, onde nesse caso, o NIL representa um número infinito. Dessa maneira, o que se pode observar é que a

regra expressa que alguém que seja ‘pais’, é uma pessoa que possui pelo menos um filho e todas as sua crianças também devem ser pessoas.

Na FIG. 3.3, *mae* é uma conceito que herda características de *pais*, desse modo, ela também irá herdar a propriedade *temFilho* da mesma forma. Devido a esse fato, o que se pode observar é que existem várias relações implícitas entre os conceitos. Os sistemas de representação do conhecimento devem ser capazes de encontrar esses relacionamentos implícitos no modelo, o que em certo momento pode ser uma tarefa simples, mas em outros uma tarefa um pouco mais complexa. Assim, os sistemas chegam ao que chamamos de inferência, sempre analisando as propriedades que envolvem a rede semântica.

Porém, podem acontecer casos em que existam relações bastante complexas entre os conceitos, e isso irá trazer uma certa dificuldade em precisar quais os tipos de relacionamentos podem ser computados, e de que maneira esses relacionamentos podem ser analisados de forma que não tragam resultados falhos.

3.3.3. CONSIDERAÇÕES SOBRE A LÓGICA DESCRITIVA

Com base no que foi visto até agora, vários sistemas de base de conhecimento foram propostos, dando origem à necessidade de criação de um modelo para tratamento de inferência. A linguagem aqui descrita foi introduzida para representar a sintaxe das estruturas propostas, que em muito se assemelhava a alguns formalismos lógicos.

Utilizando esse formalismo lógico, foi estipulado que todos os conceitos atômicos seriam representados através de predicados com um único termo, enquanto predicados binários iriam representar a relação entre esses conceitos.

Além desse formalismo, foram definidos vários símbolos para representação de diferentes tipos de construtores. Como exemplo de um desses símbolos temos o $\sqcap$, que denota a interseção de conceitos. A representação se apresenta da seguinte maneira: A $\sqcap$ B. Ele é usado para fazer a restrição entre indivíduos que pertencem tanto ao conceito de A, quanto ao conceito de B. Na Lógica Descritiva, a expressão de conceito denota todos os indivíduos que satisfazem as propriedades da expressão. Para a lógica de primeira ordem, a expressão acima pode ser vista como: $A(x) \wedge B(x)$, onde $A(x)$ corresponde a todos os indivíduos que satisfazem o conceito de A.

A caracterização básica da Lógica Descritiva se concentra na construção de relações entre os conceitos. Uma das relações mais básicas se chama restrição de valor, e pode ser vista

através da seguinte simbologia: ∀R.C, que denota que todos os indivíduos que estão na relação R são pertencentes ao conceito de C.

Semanticamente, um conceito pode ser visto como um conjunto de indivíduos, enquanto que as regras são vistas como um conjunto de pares de indivíduos. Os conceitos atômicos são interpretados como um subconjunto de um certo domínio, enquanto que a semântica da sua estrutura é especificada pelo conjunto de indivíduos denotados por cada estrutura. Um exemplo disso pode ser visto pela simbologia apresentada anteriormente: A ⊓ B, ∀R.C.

Em termos práticos, ao se observar a FIG. 3.3, interpreta-se como conceito atômico os termos *Femea*, *Pessoa* e *Mulher* e como regra o termo *temFilho*. Dessa maneira, usando alguns operadores vistos anteriormente, podemos formular alguns conceitos, como o conceito de pessoas que são do sexo feminino, usando a seguinte simbologia: Pessoa ⊓ Femea

Outros conceitos podem ser apresentados, como o de exibir todos os indivíduos que têm o conceito de 'mulher' que não são mães. Nesse caso entra outro operador lógico como a união, visto com o símbolo lógico: ⊔, e também o complemento, apresentado com o símbolo vindo da lógica: ¬ , que traz a idéia de negação. Usando esses símbolos, o conceito então ficaria: Mulher ⊓ ¬ Mae

Será dado destaque agora às regras de restrição, que foram apresentadas anteriormente. Muitas linguagens disponibilizam quantificação existencial e as restrições de valores que serão bastante úteis em certos conceitos. Observando-se mais uma vez a FIG. 3.3 digamos que se queira conhecer todos os indivíduos que possuem uma filha do sexo feminino. Esse conceito pode ser descrito usando um outro operador, o existencial, ∃. Assim, utilizando o operador, a sentença para a regra ficaria: ∃hasChild.Female. Da mesma maneira, outro operador existencial mais poderoso poderá dar o conceito dos indivíduos que possuem todos os seus filhos do sexo feminino, ao contrário do caso visto acima, onde os indivíduos que tivessem pelo menos uma filha atenderiam ao conceito. A representação do segundo caso apresentado ficaria da seguinte forma: ∀hasChild.Female.

Como pode ser visto, o qualificador existencial e a restrição de valor, fazem a caracterização dos conceitos. Podemos gerar vários conceitos através deles e se obter os mais diversos tipos de semânticas para uma estrutura. O conceito a seguir, também criado da FIG. 3.3, representa todos os indivíduos que possuem filhos pertencentes ao conceito de pessoa e pelo menos um deles é do sexo feminino: ∀temfilho.Pessoa ⊓ ∃temfilho.Femea

Outro tipo de restrição igualmente importante, muito usado em sistemas de conhecimento, são as restrições numéricas, que fazem a restrição da cardinalidade entre os

conceitos. Um exemplo desse tipo de restrição é mostrado na sentença: (>=2 hasChild) ⊔ (<=5 hasChild).

Esse tipo de restrição denota uma certa noção de cardinalidade, pois apresenta todos os indivíduos que possuem no mínimo 2 filhos e no máximo 5 filhos. Essas restrições numéricas são muitas vezes vistas como uma característica peculiar da Lógica Descritiva, a qual pode ser encontrada em algumas linguagens de modelagem para Banco de Dados.

Além do que foi apresentado até aqui, com relação a construções de expressões de conceitos, a Lógica Descritiva cria construções para regras, capazes de estabelecer regras hierárquicas. No entanto, a utilização dessas expressões de regras é geralmente limitada para expressar relação entre os conceitos.

A partir da interseção de várias regras, pode-se chegar à formulação de uma regra mais abrangente. O conceito de "temfilho ⊓ ∀temfilho.Femea", possui uma similaridade com a regra *temFilha*, já que qualquer que seja o filho de um indivíduo ele deve ser do sexo feminino. Assim é possível juntar essa regra a uma outra e formar um outro conceito bem mais completo, como o de todas as mulheres que possuem no máximo 2 filhas. Esse conceito pode ser observado da seguinte maneira: "Mulher ⊓ <=2 (temFilho ⊓ ∀temFilho.Femea)".

A seguir, nas TABs 3.1 e 3.2 serão apresentadas algumas das proposições presente na Lógica Descritiva, bem como algumas regras de equivalência respectivamente.

A	Conceitos Primitivos
R	Regras Primitivas
¬	Complemento
C⊓D	Conjunção
C⊔D	Disjunção
∀R.C	Quantitativo Universal
∃R.C	Quantitativo Existencial

TAB 3.1. Proposições mais simples da Lógica Descritiva

∃R.T		∃R
¬ (C⊓D)		¬C ⊔ ¬D
¬ (C⊔D)		¬C ⊓ ¬D
¬ (∀R.C)		∃R. ¬C
¬ (∃R.C)		∀R. ¬C

TAB 3.2. Proposições equivalentes

3.3.4. RACIOCINANDO ATRAVÉS DA LÓGICA DESCRITIVA

Existe um conceito básico de inferência presente na Lógica Descritiva (LD), que aparece quando se fala de subconjuntos. Assim sendo, a definição de que um conceito A é

subconjunto de um conjunto B é representado através da simbologia: A $\subseteq$ B. Verificar se um conceito é subconjunto de outro é verificar se o conceito denotado por B é mais genérico do que o denotado por A. Um exemplo pode ser visto no conceito de mamífero e animal, que simbolicamente é representado por: MAMIFERO $\subseteq$ ANIMAL. Assim, para fazer a verificação desse tipo de relação, é interessante que se esteja atento ao que acontece na terminologia. Algumas características presentes no conceito mais abrangente podem ser inferidas para os conceitos que são subconjuntos do mais genérico.

Esses procedimentos de inferência na Lógica Descritiva foram em grande parte influenciados pelas redes semânticas, onde como já foi destacado, os seus nós representavam os conceitos e as regras os links da rede. Muitos algoritmos foram criados para trabalharem com inferência nas redes semânticas, dentre eles os algoritmos de subconjuntos, que tinham como dado de entrada dois conceitos que eram transformados inicialmente em um grafo direcionado, e a idéia era descobrir se existia algum outro grafo que poderia ser embutido no primeiro. Caso fosse possível, significava que tal grafo correspondia ao conceito mais genérico. Esse método era conhecido como comparação estrutural [THOMAS A., 1982].

Uma vez apresentada uma representação básica de uma linguagem para a Lógica Descritiva e algumas técnicas chaves de raciocínio usando a LD, na próxima sessão será discutida a aplicação de LD para a representação de base de conhecimento, e ferramentas capazes de gerarem inferência em cima dessas bases.

3.3.5. A LÓGICA DESCRITIVA E A REPRESENTAÇÃO DO CONHECIMENTO

Alguns aspectos encontrados em sistemas baseados em conhecimento serão apresentados nessa seção. Serão destacados dois aspectos importantes que devem estar presentes neste tipo de sistema. O primeiro refere-se a base de conhecimento, que deve possuir uma especificação precisa capaz de ajudar o sistema a realizar suas operações, bem como uma definição de quais serviços de raciocínio serão criados, ou seja, que tipos de questões o sistema será capaz de responder. Um outro aspecto seria estabelecer um ambiente que ofereça uma boa interação do usuário com o sistema. Nessa seção, apenas o primeiro aspecto que trata da estruturação lógica para a base de conhecimento será destacado.

O desenvolvimento de aplicações sobre uma base de conhecimento surgiu a partir da especificação precisa para a representação do conhecimento e da elaboração de funcionalidades a serem providas. Esses sistemas possuíam uma máquina de raciocínio que

basicamente funcionava como uma interface de entrada e de saída, onde a primeira seria como uma entrada para a base de conhecimento, e o segundo seria responsável por trazer as respostas da base de conhecimento. A seguir é apresentada a definição do funcionamento de base de conhecimento para LD e os serviços de dedução por ela provida.

Uma base de conhecimento é composta de duas partes. A primeira é o conhecimento intencional ou conhecimento geral, que trata do domínio do problema como um todo; a segunda trata do conhecimento extensivo, correspondendo a problemas específicos. De forma análoga, a LD possui dois componentes, conhecidos como "TBox" e "ABox". O primeiro componente possui o conhecimento intensivo, que aparece na forma de terminologia, daí o nome "TBox". TBox pode ser usado para descrever as estruturas dos conceitos e também para trazer a idéia dos relacionamentos de subconjunto vindo da taxonomia, como visto no exemplo entre mamífero e animal. O segundo componente mencionado, o "ABox", é voltado ao conhecimento extensível, conhecido também como conhecimento sobre afirmações, daí o termo "ABox". Olhando esses dois tipos de conhecimento verifica-se que os "TBox" representam um tipo de conhecimento mais constante, que raramente muda, enquanto os "ABox" representam um tipo de conhecimento mais dinâmico, que depende de um conjunto de circunstâncias, sendo sujeito a mudanças constantes. Nos próximos dois subitens serão apresentados mais detalhes sobre ambos componentes da base de conhecimento da LD.

3.3.6. TBOX

Um dos elementos mais importantes da base de conhecimento na LD é dado pela operação de construção da terminologia. Seguindo essa idéia, a forma básica de declaração em um "TBox" é a definição de um conceito. Assim, como exemplo de um conceito, apresenta-se a declaração do conceito de homem. Ele pode ser visto como uma pessoa do sexo masculino, como segue:

$$\mathtt{HOMEM} \equiv \mathtt{PESSOA} \sqcap \mathtt{MASCULINO}$$

Em uma base de conhecimento para a LD, uma terminologia pode ser facilmente representada utilizando-se expressões como a anterior, porém, algumas regras de restrição sobre a criação de terminologias definidas usando-se a LD precisam ser seguidas, tais como:

- Só é permitida a definição de um nome para um dado conceito.
- As definições devem ser acíclicas, no sentido de que nenhum conceito é definido em termos dele mesmo, nem em termos de outros conceitos que diretamente se referem a ele.

Tais restrições são comuns em bases de conhecimento na LD, pois garantem que as definições de conceitos só poderão ser expandidas para uma expressão mais complexa, contendo apenas conceitos atômicos. Isso será possível substituindo toda definição de conceitos pelo lado direito da definição.

Observando então o aspecto da expansão, a complexidade de um processo de inferência vai surgir devido ao fato deste requerer a observação de toda a terminologia, tratando cada conceito como se fosse uma expressão totalmente expandida. A maioria dos métodos de raciocínio estudado na LD é voltada para análises feitas em expressões conceituais e mais profundamente nas análises feitas em subconjuntos, como visto no item anterior, podendo ser considerado um dos pontos fundamentais no serviço de raciocínio disponibilizado pelos "TBox".

Na construção de uma terminologia, uma das tarefas principais é situar uma nova expressão de conceito, seguindo-se de uma taxonomia com hierarquia de conceitos. Esse novo conceito entrará entre o conceito mais geral, que abrange esse novo conceito a ser classificado, e um conceito mais especifico, subconjunto desse novo conceito.

Basicamente, os serviços de dedução existentes em uma "TBox" podem ser vistos como implicações lógicas e se resumem a verificar se um relacionamento genérico, como por exemplo os encontrados em relacionamentos de conjunto e subconjunto de conceitos, é uma conseqüência lógica encontrada no "TBox". A seguir, serão apresentados alguns exemplos de membros de um "TBox".

Exemplo 1: Se existe alguém que ensina algum curso, este deve ser professor e não deve ser estudante universitário.

$$\exists \mathtt{ENSINA.Curso} \subseteq \neg\ \mathtt{Estudante\_universit} \cup \mathtt{Professor}$$

Exemplo 2: Mostra as regras que determinam o papel de cada membro de uma família.

$$\mathtt{Mãe} \equiv \mathtt{Mulher} \sqcap \exists \mathtt{temcrianca.Humano}$$

```
Pai ≡ Homem □ ∃temcrianca.Humano
Pais ≡ Mae □ Pai
Avó ≡ Mulher □ ∃temcrianca.Pais
```

3.3.7. ABOX

Como descrito anteriormente, os “ABox” são os responsáveis por disponibilizar o conhecimento extensível sobre o domínio de interesse. Um exemplo disso pode ser visto logo a seguir com a seguinte definição.

```
MaeSemFilha(MARY)
Pai(PETER)
temFilho(MARY, PETER)
temFilho(PETER, HARRY)
temFilho(MARY, PAUL)
```

Agora, segue o exemplo abaixo:

```
Femea □ Pessoa (MARY)
```

Nesse exemplo afirma-se que Mary é uma pessoa do sexo feminino. Esse tipo de afirmativa é conhecido como afirmativa de conceito, enquanto a primeira é chamada de afirmativas de regras. Dessa forma é visto que um “ABox” pode exprimir conhecimento através de dois tipos de afirmativas: Conceituais e regras.

As tarefas de raciocínio executadas em uma “ABox” consistem basicamente em checagem de instâncias, que simplesmente verificam se um indivíduo pertence ou não a um determinado conceito. Seguindo essa verificação de instância, outros tipos de raciocínio podem ser vistos, como a verificação da consistência de uma base de dados, o que significa verificar se todos conceitos são capazes de possuir pelo menos um indivíduo pertencente ao mesmo. Outro serviço de raciocínio seria o chamado “realização”, que identifica qual a classe mais específica da qual um determinado objeto será instância. Por fim, um dos serviços de raciocínio mais triviais, determina se um indivíduo pertence ou não a uma base de conhecimento. Esse raciocínio é chamado de recuperação.

Apesar do raciocínio imposto pelas “TBox” e pelas “ABox” terem suas particularidades, muitas vezes acontecem situações em que uma base de conhecimento possuindo sua engrenagem totalmente voltada para uma “TBox”, não é capaz de solucionar problemas mais

complexos. Nesse caso, o serviço de raciocínio, terá que levar em consideração, toda a base de conhecimento compreendida com “TBox” e “ABox” e ambas terão que trabalhar em conjunto. Quando um sistema usa toda a base de conhecimento, (“TBox” e “ABox”), é chamado de sistemas de raciocínio híbrido [RONALD J. BRACHMAN, 1985], e cria um poderoso mecanismo de raciocínio lógico, como pode ser visto na FIG. 3.4.

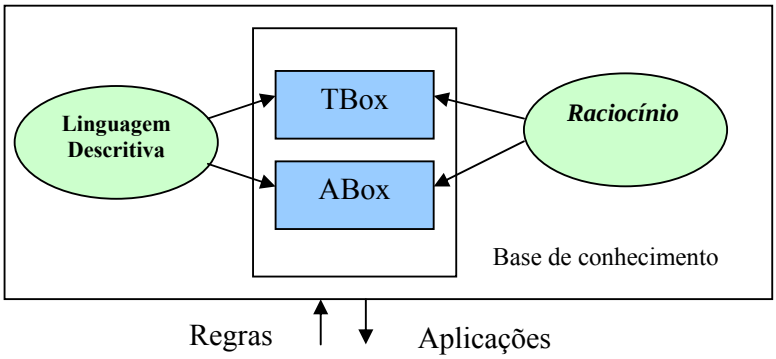


FIG. 3.4 – Arquitetura de sistema de representação do conhecimento baseado em Lógica Descritiva

A seguir será apresentado alguns exemplos de “ABox” que podem ser armazenados em uma base de conhecimento.

```
Estudante (handrick).
Curso (SC03115).
```

Define o indivíduo ‘Handrick’ como ‘Estudante’ e o código SC03115 como um elemento do curso.

```
MATRICULADO(handrick,SC03115).
```

Significa que o aluno ‘handrick’ está ‘MATRICULADO’ no curso ‘SC03115’.

3.3.8. UTILIZAÇÃO DA LÓGICA DESCRITIVA.

No início do desenvolvimento da LD foram propostas muitas linguagens de objetivo geral, capazes de representar conhecimento e raciocínio, podendo dessa maneira serem usadas em diferentes tipos de aplicações. Essas linguagens foram desenvolvidas visando

principalmente criar a representação de conhecimento sobre aplicações que utilizassem estrutura hierárquica sobre relacionamento do tipo "é um". Essa habilidade de representar estruturas hierárquicas e de raciocinar sobre taxonomias motivou a utilização da LD como linguagem de modelagem para desenvolvimento de estruturas de conhecimento, bem como uma linguagem para representar outros formalismos, como o formalismo de ontologia.

Na próxima sessão será apresentada uma linguagem voltada para ontologia, denominada OWL. Ela foi desenvolvida para facilitar a interpretação de conteúdos para web, disponibilizando um vocabulário adicional, juntamente com uma semântica formal. A OWL possui uma certa correspondência com a LD devido ao fato de toda sua fundamentação formal se basear na LD aqui apresentada.

3.4. OWL – ONTOLOGY WEB LANGUAGE

Como apresentado no primeiro item, a Web Semântica é a visão futurista da web, onde todas as informações possuem um significado explícito. Essa característica auxiliará na tarefa de processamento e de integração de informações ao longo da web, através de máquinas.

Atualmente, tem-se observado que grande parte dos aplicativos web considera o processamento de informações como voltado para apresentação a seres humanos. Entretanto, existem aplicações que precisam automaticamente processar informações contidas em um documento, sem intervenção humana. Para ajudar nessa tarefa surgiu a OWL [OWL, 2005] que permite representar o significado dos termos e a relação entre os mesmos. Como apresentado em itens anteriores, essas características são comuns à ontologias, daí o nome da linguagem OWL, "Ontology Web Language". Linguagens voltadas para Web Semântica como XML, RDF e RDF-S, não são capazes de apresentar o significado de expressões tão facilmente como a OWL, e com isso a OWL torna-se uma ferramenta muito mais poderosa na criação de aplicativos, capaz de interpretar conteúdos existentes na web, pois acrescenta maior vocabulário para descrever propriedades e classes em uma ontologia. Entre algumas propriedades presentes na OWL aparecem: a relação entre classes, como por exemplo, disjunção, cardinalidade de classes, caracterização de propriedade com a simetria, entre outros. Maiores detalhes de funcionalidades e sintática serão vistos nos próximos itens.

3.4.1 OWL E SUAS TRÊS SUB-LINGUAGENS.

Dependendo do tipo de usuário da aplicação, bem como da comunidade específica de implementação, uma das três sub-linguagens que a OWL pode oferecer poderá ser usada: OWL Lite, OWL DL e OWL FULL.

Primeiramente será dado destaque a OWL Lite, que é voltada para usuários que necessitam implementar pequenas aplicações, como por exemplo, uma simples classificação hierárquica, ou a especificação de algumas regras triviais, tais como regras de cardinalidade. O OWL Lite permite uma migração fácil e rápida para tesauros e taxonomias, além de possuir um formalismo bem menos complexo que a sub-linguagem OWL DL.

A OWL DL é mais poderosa que a OWL Lite e é manipulada por usuários exigentes que necessitam de uma maior expressividade, sem abrir mão da completeza computacional, ou seja, o usuário espera que todas as conclusões geradas sejam computáveis.

A OWL DL possui todas as construções presentes na linguagem OWL, porém com algumas restrições, como por exemplo, uma classe não poder ser instância de uma outra classe, apesar de permitir que uma classe seja uma subclasse de muitas outras classes.

Por último, será abordada a OWL Full, que é assim descrita devido ao fato da sub-linguagem possuir a máxima expressividade, além de toda a liberdade sintática oferecida pela RDF, porém, sem garantias computacionais. Um exemplo do uso da OWL Full é poder fazer com que uma classe seja tratada simultaneamente como um conjunto de indivíduos ou como um próprio indivíduo, aumentando com isso, o poder da ontologia com relação ao significado do vocabulário pré-estabelecido.

Como se pode observar, cada sub-linguagem possui menos características do que a sua sucessora. Assim, pode-se dizer que uma sub-linguagem pode ser legalmente expressa em sua linguagem sucessora, porém não o contrário. Assim uma ontologia presente em OWL Lite, pode ser perfeitamente representada em uma ontologia legal para a OWL DL, bem como todas as suas conclusões, e assim por diante. A representação a seguir mostra com detalhes esse aspecto.

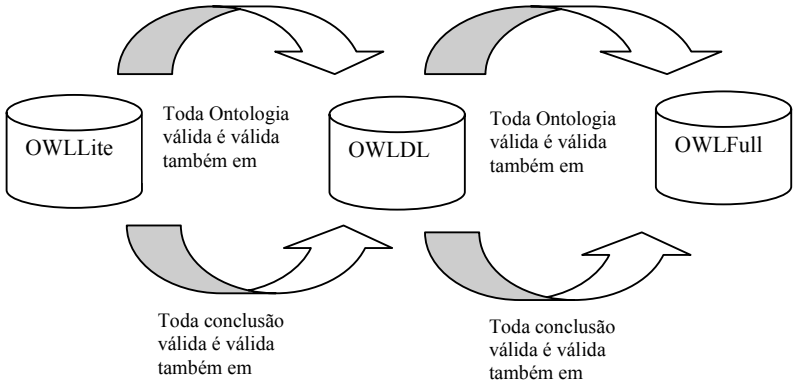


FIG. 3.5 – Sub-linguagens OWL

Ao se desenvolver uma ontologia usando OWL, deve-se considerar quais das sub-linguagens irá atender melhor às necessidades. Por exemplo, ao se definir o uso de OWL Lite ou OWL DL, deverá ser observado até que ponto será realmente necessário o alto poder de expressividade, que pode ser disponibilizado pelas construções feitas em OWL DL, a fim de que se não utilize uma linguagem bastante poderosa para solucionar problemas triviais, como o de herança.

Com relação ao uso do OWL Full, a sua melhor utilização seria em aplicações que requerem todas a facilidade do RDF-Schema. Isso porque, essa linguagem é vista como uma extensão do RDF, enquanto as outras duas linguagens, OWL Lite e DL, são vistas apenas como uma extensão de uma visão restrita do RDF. Dessa forma, pode-se afirmar que todas as linguagens OWL podem ser representadas por um documento RDF, porém nem todo documento RDF, pode ser representado por todas as linguagens OWL, a não ser a OWL Full. Só alguns documentos RDF podem ter as suas representações no OWL Lite e DL.

3.4.2. OWL LITE

Nessa sessão são apresentadas algumas características da OWL-Lite de uma forma bastante direta, sem aprofundar muito aos detalhes de sua sintaxe, lembrando mais uma vez que a OWL Lite possui mais limitações do que suas outras duas linguagens sucessoras. As construções presentes nessa sub-linguagem estão presentes na TAB. 3.3.

TAB. 3.3 – Sub-linguagens OWL Lite

RDF Schema – Características	Igualdade	Características de propriedade
• Class (Thing, Nothng) • rdfs: subClassOf • rdf: Property • rdfs: subPropertyOf • rdfs: domain • rdfs: range • Individual	• equivalentClass • equivalentProperty • sameAs • differentFrom • AllDifferent • distinctMembers	• ObjectProperty • DatatypeProperty • InverseOf • TransitiveProperty • Symmetric Property • Functional Property • InverseFunctionalProperty

A primeira coluna apresenta as características do RDF Schema, como a classe e sub-classe.

"Class" define um grupo de indivíduos que possuem as mesmas propriedades e características. Ela pode seguir uma especificação hierárquica que pode ser utilizada através da construção "subClass", que irá representar um sub-conjunto de uma classe mais geral. Assim como visto em itens anteriores, seguem exemplos de utilização.

```
<owl:Class rdf:ID="Pessoa"/>
<owl:Class rdf:ID="Curso"/>
<owl:Class rdf:ID="Disciplina"/>
```

O exemplo mostra a declaração de três classes, "Pessoa", "Curso" e "Disciplina", e em seguida um exemplo da segunda construção "subClass".

```
<owl:Class rdf:ID="Elefante">
      <rdfs:subClassOf rdf:resource="Animal"/>
      ...
</owl:Class>
```

Aqui o conceito de classe e sub-classe surge através dos termos, "Animal" e "Elefante". Pela construção OWL "subClassOf" é possível apresentar essa definição, onde fica claro que a classe "elefante" é uma sub-classe da classe dos "animais", ou seja, todas as características que estejam presentes na classe dos animais, devem também estar presentes na sub-classe elefante.

A seguir serão vistas algumas construções referentes à equivalência ou não entre algumas classes. Para isso, apresentam-se alguns exemplos do segundo grupo de construções apresentados na TAB. 3.3. Inicia-se com a construção de uma "equivalentClass", expressando

que duas classes são equivalentes e por conseqüência possuem as mesmas instâncias. Um exemplo dessa equivalência pode ser visto a seguir:

```
<owl:Class rdf:ID="Carro">
       <owl:equivalentClass rdf:resource="Automovel"/>
</owl:Class>
```

No terceiro conjunto de construções apresentado inicialmente, aparece uma das construções mais interessantes e bastante útil para construção de uma ontologia, que são as propriedades. Especial atenção será dada as propriedades transitivas, inversa e simétrica.

Começando pela transitividade, define-se que uma propriedade é transitiva quando dado um par (x,y), que é instância de uma propriedade transitiva P, e um par (y,z) que também é uma instância de P, conclui-se que o par (x,z) também é uma instância de P. Um exemplo prático disso pode ser visto quando observamos uma disciplina de um curso, que compreende um determinado assunto B, que igualmente compreende um outro assunto C. Por transitividade essa disciplina também envolverá esse assunto C. Segue a construção para esse caso:

```
<owl:ObjectProperty rdf:ID="compreende">
      <rdf:type rdf:resource="&owl;TransitiveProperty" />
      <rdfs:domain rdf:resource="&owl;Thing" />
      <rdfs:range rdf:resource="#Assunto" />
</owl:ObjectProperty>

<Assunto rdf:ID="Banco de Dados">
      <compreende rdf:resource="#Consulta" />
</Assunto>

<Assunto rdf:ID="Consulta">
      <compreende rdf:resource="#Sql" />
</Assunto>
```

Uma propriedade é dita simétrica, se existir um par (x,y) que é instância de uma propriedade simétrica P, e um par (y,x) que também é instância da mesma propriedade simétrica anterior. Um exemplo clássico desse tipo de propriedade pode ser visto no caso de dois amigos João e Pedro. A propriedade simétrica de amigo, tanto pode ser analisada do ponto de vista de João, quanto de Pedro, ou seja, se João é amigo de Pedro, Pedro também é amigo de João. Na forma OWL, tal propriedade pode ser vista da seguinte maneira:

```
<owl:ObjectProperty rdf:ID="amigo">
      <rdf:type rdf:resource="&owl;SymmetricProperty" />
      <rdfs:domain rdf:resource="#Amigos" />
```

```
      <rdfs:range rdf:resource="#Amigos" />
</owl:ObjectProperty>

<Amigos rdf:ID="Joao">
      <amigo rdf:resource="#Pedro" />
</Amigos>
```

Finalizando, apresenta-se a propriedade inversa. Quando uma propriedade é declarada como inversa de outra, diz-se que se um indivíduo é relacionado a outro através de uma propriedade P1, inversa a outra P2, então esse último indivíduo se relaciona ao primeiro através da propriedade P2. Nesse contexto, pode-se afirmar que as propriedades "é mãe" e "é filha" são inversas entre si, já que se Maria "é mãe" de Karina, Karina "é filha" de Maria. Segue a construção:

```
<owl:ObjectProperty rdf:ID="emae">
      <rdf:type rdf:resource="&owl;FunctionalProperty" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="efilha">
      <owl:inverseOf rdf:resource="#emae" />
</owl:ObjectProperty>
```

Não serão apresentados maiores detalhes sobre algumas propriedades de restrições que a OWL Lite permite realizar. A seguir, será apresentado um conjunto de construções previstas pela OWL Lite, onde entre elas está a restrição de cardinalidade. Apesar dessa restrição se encontrar presente, ela se encontra de uma forma limitada, visto que permite apenas declarações sobre cardinalidade de valores 0 ou 1. A restrição de cardinalidade de valor arbitrário só é permitida na OWL DL e OWL Full. Essas propriedades restritivas, bem como outras da OWL Lite estão na TAB. 3.4.

TAB. 3.4. – Outras Construções da OWL Lite

Propriedades de Restrição	**Restrição de cardinalidade**	**Informação do cabeçalho**
• Restriction • onProperty • allValueFrom • someValueFrom	• minCardinality (only 0 or 1) • maxCardinality (only 0 or 1) • cardinality (only 0 or 1)	• Ontology • Imports
Interseção de Classes	**Controle de Versão**	**Propriedade das anotações**

• IntersectionOf **Tipo de Dados** • xsd datatypes	• VersionInfo • priorVersion • backwardCompatibleWith • imcopatibleWith • DeprecateClass • DeprecateProperty	• rdfs: label • rdfs: comment • rdfs: seeAlso • rdfs: isDefinedBy • AnnotationProperty • OntologyProperty

3.4.3. OWL DL E FULL

Essas duas sub-linguagens que serão apresentadas a seguir são uma extensão da sub-linguagem apresentada no item anterior. Elas possuem as mesmas construções da OWL Lite acrescidas de algumas outras particularidades. Apesar da OWL DL e a OWL Full possuírem o mesmo vocabulário, ainda sim a OWL DL é um pouco mais limitada. Nela é preciso que exista uma separação de tipo, ou seja, uma classe não pode também ser uma propriedade ou um indivíduo, uma propriedade também não pode ser um indivíduo ou uma classe. Além disso, a OWL DL requer que uma propriedade seja, ou de um objeto ou de um tipo de dado. Na TAB. 3.5 apresentam-se as construções referentes a ambas.

TAB. 3.5 – Sub-linguagens OWL DL e Full

Classe de Axiomas	Combinação lógica de Classe
• oneOf, dataRange • disjoinWith • equivalentClass • rdfs: subClassOf	• unionOf • complementOf • intersectionOf

Primeiramente será descrita a construção do “oneOF”. Ela irá fazer a enumeração dos indivíduos que compõem uma determinada classe. Os membros das classes serão exatamente os indivíduos enumerados. Um exemplo desse tipo de classe é a classe “mesesdoAno”, que corresponde aos meses que compõem um ano. Esse conjunto pode ser perfeitamente enumerado e pode ter sua máxima cardinalidade facilmente inferida, através de qualquer propriedade que tenha “mesesdoAno” como restrição de “allValuesFrom”, cuja cardinalidade nesse caso seria doze. O exemplo a seguir enumera os tipos de vinhos que podem ser listados, especificando um conjunto formado pelos seguintes tipos de vinhos {Branco, Tinto, Rose}.

```
<owl:Class rdf:ID="TipoVinho">
  <rdfs:subClassOf rdf:resource="#TipoBebida"/>
  <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:about="#Branco"/>
    <owl:Thing rdf:about="#Tinto"/>
    <owl:Thing rdf:about="#Rose"/>
  </owl:oneOf>
</owl:Class>
```

Agora será mostrado o uso do "disjointWith", que representa a disjunção que pode ocorrer entre classes. Um exemplo de classes disjuntas são: HOMEM e MULHER. Assim, um indivíduo que pertence a uma classe disjunta a outra, não pode pertencer a ambas as classes. Chega-se a conclusão que um indivíduo irá pertencer a classe HOMEM, no momento em que se chegar a conclusão que ele não pertence a classe MULHER. Isso se for considerado que não existem outras classes envolvidas, conforme mostra o exemplo a seguir:

```
<owl:Class rdf:ID="Engenheiro">
      <rdfs:subClassOf rdf:resource="#Funcionarios"/>
      <owl:disjointWith rdf:resource="#Secretaria"/>
      <owl:disjointWith rdf:resource="#Motorista"/>
</owl:Class>
```

Finalizando essa primeira parte de construções, apresentam-se as combinações booleanas possíveis de serem realizadas na OWL DL e OWL Full. Elas são: "unionOf", "complementOf" e "intersectionOf". Essas duas sub-linguagem permitem essas combinações booleanas arbitrárias de classes e também de restrições. "unionOf" determina o particionamento total e uma classe por aquelas enumeradas no predicado. Seguem alguns exemplos dessas construções, iniciando por "UnionOf".

```
<owl:Class rdf:ID="Frutas">
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#FrutasDoce" />
        <owl:Class rdf:about="#FrutasAzedas" />
      </owl:unionOf>
</owl:Class>
```

A construção "complementOf" seleciona todos os indivíduos de um certo domínio de aplicação que não pertencem a uma determinada classe, servindo-lhe apenas de complemento. Um exemplo dessa construção pode ser visto a seguir, através da descrição de alimentos perecíveis e não perecíveis.

```
<owl:Class rdf:ID="AlimentosPereciveis" />
```

```
<owl:Class rdf:ID="AlimentosNaoPereciveis">
        <owl:complementOf rdf:resource="#AlimentosPereciveis" />
</owl:Class>
```

Finalizando essa primeira etapa, é apresentada a construção “intersectionOf” que traz uma classe de características comuns de uma ou mais classes. O exemplo abaixo descreve um aluno de mestrado do IME, que possui características comuns de um estudante do IME e de um estudante de mestrado.

```
<owl:Class rdf:ID="EstudanteMestradoIME">
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#EstudanteIME" />
        <owl:Class rdf:about="#EstudaneMestrado" />
      </owl:intersectionOf>
</owl:Class>
```

Outras construções, que dessa vez, tratam da cardinalidade são mostradas na TAB. 3.6. Nesse caso, as sub-linguagens OWL DL e Full são mais poderosas que a OWL Lite que só permite cardinalidade 0 e 1. Segue portanto o quadro com as construções finais a serem discutidas.

TAB. 3.6 – Outras construções das Sub-linguagens OWL DL e Full

Cardinalidade	Informação de conteúdo
• minCardinality • maxCardinality • cardinality	• hasValue

Diferentemente do que foi apresentado no OWL Lite, o caso aqui poderá ir bem mais além do que representar cardinalidade: “pelo menos um”, “não mais que um” e “exatamente um”. É possível disponibilizar números inteiros quaisquer, dependendo de sua necessidade de cardinalidade mínima ou máxima. É apresentado no Ex1 e no Ex2 o uso da cardinalidade mínima e máxima. No primeiro caso uma disciplina apresenta uma carga horária de no máximo 90 horas, e no segundo exemplo um curso requer no mínimo 4 alunos inscritos.

Ex 1:

```
<owl:Class rdf:ID="Disciplina">
      <rdfs:subClassOf>
        <owl:Restriction>
          <owl:onProperty rdf:resource="#temcargahoraria"/>
```

```
          <owl: maxCardinality
    rdf:datatype="&xsd;nonNegativeInteger"90</owl: maxCardinality>
        </owl:Restriction>
      </rdfs:subClassOf>
</owl:Class>
```

Ex 2:

```
<owl:Class rdf:ID="Curso">
      <rdfs:subClassOf>
        <owl:Restriction>
          <owl:onProperty rdf:resource="#temalunos"/>
          <owl: minCardinality
    rdf:datatype="&xsd;nonNegativeInteger">4</owl:minCardinality >
        </owl:Restriction>
      </rdfs:subClassOf>
</owl:Class>
```

Nessas últimas seções foram apresentadas algumas características e implementações da linguagem OWL. O objeto foi mostrar sua utilização no papel de representação de ontologias. Também foi apresentada uma ligeira descrição de todas as suas sub-linguagens, mostrando o poder de cada uma delas, bem como a sua utilidade para cada tipo de aplicação. Os exemplos aqui apresentados são uma pequena amostra de como é possível determinar propriedades de uma classe, bem como estabelecer o relacionamento dela com outras.

No próximo item será apresentada uma rápida apresentação sobre linguagem lógica, demonstrando que é possível representar algumas das construções apresentadas nesses últimos itens através desse tipo de linguagem.

3.5. LINGUAGEM LÓGICA

Nesse item são introduzidos os conceitos básicos de uma linguagem lógica. Para entender inicialmente como funciona esse tipo de programação, deve-se entender que a programação lógica é composta por dois elementos básicos e disjuntos: lógica e controle.

O componente lógico corresponde à definição criada que deverá ser solucionada, enquanto o controle irá estabelecer o caminho a ser seguido para se conseguir chegar a uma solução. A idéia é que um programador nesse tipo de linguagem deva apenas fazer a definição dos tipos lógicos, deixando a cargo do sistema de programação lógica a escolha da execução adequada. A tarefa de um programador nessa área é especificar o problema ou situação, através de um conjunto finito de um tipo especial de sentença lógica, denominada cláusula.

As linguagens procedimentais utilizam uma descrição de procedimentos para obtenção da solução de um problema, o que não acontece com as linguagens lógicas, onde o sistema responsável pelo processamento lógico do programa é responsável pelo procedimento a ser adotado em sua execução. Alternativamente, um programa em lógica poderia ser visto como uma base de dados, se as bases de dados convencionais só tratassem da manipulação de fatos. Porém ela vai mais além e possui um alcance mais abrangente, permitindo a implementação de regras como "Todo Mestrado em Computação é uma pós-graduação". A TAB. 3.7 a seguir apresenta uma breve representação das diferenças existentes entre uma programação convencional e lógica.

TAB. 3.7 - Programas Convencionais x Programas em Lógica

PROGRAMAS CONVENCIONAIS	PROGRAMAS EM LÓGICA
Processamento Numérico	Processamento Simbólico
Soluções Algorítmicas	Soluções Heurísticas
Estruturas de Controle e Conhecimento Integradas	Estruturas de Controle e Conhecimento Separadas
Difícil Modificação	Fácil Modificação
Somente Resposta Totalmente Correta	Incluem Respostas Parcialmente Corretas
Somente a Melhor Solução Possível	Incluem Todas as Soluções Possíveis

O principal paradigma da programação lógica é a programação declarativa, em oposição ao que se encontra nas linguagens convencionais, que é a programação procedimental. Esse mesmo tipo de programação declarativa também engloba outros tipos de programação, como a programação funcional, encontradas em linguagens como: Haskell [HASKELL, 2004] e Lisp [LISP, 2004].

O principal aspecto da programação lógica consiste em identificar a noção de computação com a noção de dedução. Dessa maneira, expressa-se conhecimento através dessas linguagens por meio de cláusulas do tipo: fatos e regras, onde um fato representa uma verdade incondicional e regras criam as definições necessárias para que uma declaração seja aceita como verdadeira. Considerando que ambas as cláusulas são utilizadas em conjunto, não se faz necessário a utilização de nenhum componente dedutível adicional. Além disso, como as regras podem ser implementadas de forma não determinística e utilizando recursão, pode-se obter representações não redundantes, concisas e bastante claras do que se deseja representar.

Com auxílio da demonstração da linguagem lógica, iremos utilizar a linguagem Prolog [PROLOG, 2004], porém deve-se tomar cuidado para não confundir a "programação Prolog"

com a “programação lógica”, pois esses dois termos devem ser empregados de forma distinta, pois a linguagem Prolog na realidade é apenas uma abordagem particular da programação em lógica.

3.5.1. A LINGUAGEM PROLOG

Prolog é uma linguagem que atua especificamente no domínio da programação simbólica. É especialmente adequado para solucionar problemas que envolvem objetos e seus relacionamentos. É uma linguagem que reforça a teoria de que a lógica é um mecanismo convincente para criar representações do conhecimento, bem como o processamento do mesmo. O programador Prolog não é aquele que realiza a descrição de procedimentos para solucionar um dado problema, e sim aquele que expressa apenas as estruturas lógicas, através de fatos, regras e consultas.

Em Prolog uma consulta pode ser entendida como uma seqüência formada por um ou vários objetivos. Para chegar a resposta da consulta, o Prolog varre todos os objetivos que a consulta exige e interpreta-os como uma conjunção. Ao satisfazer um objetivo está demonstrado que o mesmo é verdadeiro. Um consulta também pode possuir variáveis, e nesse caso, o sistema deverá encontrar também os objetos que, atribuídos a essas variáveis, iram satisfazer o objetivo proposto pela consulta. Caso nenhuma instanciação de uma variável consiga alcançar o objetivo proposto, a resposta a essa consulta será vazia.

Em termos matemáticos, o Prolog aceita os fatos e as regras com um conjunto de axiomas, e a consulta do usuário como um teorema a ser provado. Assim, a tarefa do sistema é demonstrar que o teorema pode ser provado com base nos axiomas representados pelo conjunto das cláusulas que formam o programa.

Para exemplificar essa idéia de consultas em cima de fatos e regras na linguagem Prolog, é apresentada a FIG. 3.6 a seguir. Nela é vista uma árvore genealógica de uma família, onde cada elemento da família pode ser entendido como um objeto e o parentesco de um membro da família com outro, a relação que pode existir entre esses dois objetos. Dessa maneira, a relação descrita como “progenitor” a seguir cria a associação de um determinado indivíduo a seus progenitores, como pode ser visto através do fato:

```
progenitor (tomaz, handrick).
```

Nesse exemplo, progenitor é o nome da relação, enquanto Tomaz e Handrick são os seus argumentos. Para completar o exemplo, são apresentados todos os fatos restantes que descrevem a relação do tipo "progenitores", que completam a árvore genealógica da FIG. 3.6.

progenitor (denise, tomaz).
progenitor (costa, tomaz).
progenitor (costa, alberto).
progenitor (tomaz, micaelle).
progenitor (tomaz, handrick).
progenitor (handrick, bruno).

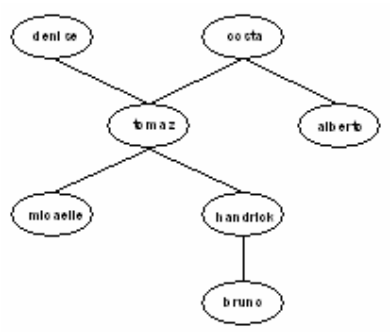


FIG. 3.6 - Árvore genealógica e representação de fato em Prolog

No Prolog, a idéia de se criar consultas em cima de fatos consiste em verificar a existência desse fato na base criada. Um exemplo que mostra esse tipo de consulta seria o de querer conhecer se o objeto 'Handrick' é progenitor de 'Bruno'. Como existe na base um fato que declara explicitamente que Handrick é realmente o progenitor de Bruno, então a máquina de interpretação Prolog irá trazer como resposta "yes". A consulta e a resposta da mesma pode ser vista logo a seguir.

```
progenitor(handrick,bruno).
Yes.
```

Consultas que são feitas em cima de fatos que não existem terão resposta negativa, "no", como no exemplo que segue.

```
progenitor(handrick,jose).
No.
```

Outro tipo de consulta que o Prolog efetua sobre a base de fatos consiste em averiguar a completude de um fato através de variáveis. Um exemplo dessa consulta é a que deseja conhecer o progenitor de Bruno. Para esse tipo de consulta, é necessária a introdução de uma variável X qualquer. Nesse caso, a máquina de interpretação Prolog não mais retornará uma resposta do tipo "yes" ou "no", e sim atribuições a essa variável X. Assim, o exemplo citado seria implementado da seguinte forma:

```
progenitor(X, bruno).
X=handrick;
```

Da mesma forma que foi feito com a consulta anterior, também é possível descobrir todos os filhos de uma pessoa. Assim a consulta que retorna todos os filhos de Tomaz seria estruturada da forma apresentada logo a seguir. A resposta final, 'no', significa que a busca por fatos na base foi finalizada.

```
progenitor(tomaz, X).
X=handrick;
X=micaelle;
no.
```

Assim, como pode ser visto pelo código fonte Prolog da árvore genealógica, a mesma pode ser ampliada acrescentando uma série de outros fatos, que mostram como a linguagem Prolog pode desempenhar bem a tarefa de representação do conhecimento. Um desses novos fatos que enriquece a definição da árvore genealógica é a definição do sexo das pessoas que a compõem. Para tanto, só se faz necessário acrescentar alguns fatos novos no código fonte original. A definição desses fatos é mostrada logo a seguir:

```
masculino (handrick).
masculino (tomaz).
masculino (costa).
masculino (bruno).

feminino (denise).
feminino (micaelle).
feminino (denise).
```

A primeira relação pode ser entendida semanticamente como sendo "Handrick é do sexo masculino". A mesma idéia pode ser seguida por todos os outros fatos.

Uma outra ampliação possível de ser feita no código seria o de introduzir a relação "filho", de forma semelhante como foi feita na relação "progenitor". Assim o novo fato lógico a ser acrescentado seria:

```
filho (handrick, tomaz)
```

Entretanto, esse modo de representação não é tão elegante, já que esse tipo de relação trata de uma relação inversa a uma relação já existente. Nesse caso o uso do fato de que a relação "filho" é o inverso da relação "progenitor" poderia ser vista de forma bem interessante. A declaração lógica que definiria essa relação seria, portanto, a de que para todo X e Y, Y é filho de X se X é progenitor de Y.

Essa definição encontra-se bem próximo do formalismo da linguagem Prolog e assim a sua tradução para o mesmo ficaria da seguinte maneira:

```
filho (Y,X) :- progenitor (X,Y).
```

Esse tipo de cláusula no Prolog é conhecida como regra e a mesma se apresenta de forma bastante semelhante em diferentes linguagens lógicas.

Como se pode observar, existe uma diferença grande entre fatos e regras. Um fato é aquilo que sempre é verdadeiro, o que nem sempre acontece com uma regra, que só será verdadeira se atender a certas condições.

Regras são formadas por duas partes: a primeira no lado esquerdo da cláusula é onde se encontra asconclusão, enquanto que no lado direito, encontram-se as condições. Ambas são separados pelo símbolo ":-", que significa "se".

Para exemplificar a nossa regra, digamos que se queria sabe se o objeto 'Handrick' se relaciona com o objeto 'Tomaz' através da relação 'filho. Em Prolog tal consulta se apresentaria na seguinte forma sintática:

```
filho (handrick, tomaz).
```

Nenhum fato que confirme a consulta existe no código fonte e a única forma de realizar a avaliação é através da interpretação da regra descrita. Com a interpretação da regra, a consulta transforma-se na avaliação do fato:

```
progenitor (tomaz, handrick).
```

Com essa idéia de regras, é possível criar novas definições de conceitos que podiam ser descritos anteriormente através de um fato. No caso da definição do conceito 'avô', ele poderia ser agora definido através de uma regra definida da seguinte maneira:

```
avo (X, S) :- progenitor (X, Y), progenitor (Y, S).
```

Como último exemplo de implementação de regra na linguagem Prolog, será mostrada uma última relação entre os objetos que formam a árvore genealógica da FIG. 3.6. Nesse exemplo é criada a definição da relação 'irmao'. Conceitualmente é dito que dois indivíduos X, Y são irmãos quando para todo X e Y, ambos possuem um progenitor em comum. Na

linguagem Prolog a regra que define essa relação encontra-se a seguir.

```
irmao (X,Y) :- progenitor (S,X), progenitor (S,Y).
```

Nesse item foram apresentadas algumas características da linguagem Prolog, destacando como se apresentam as regras e os fatos nesse tipo de linguagem, bem como são interpretadas as consultas na mesma. No item que se segue serão mostradas algumas aplicações possíveis de serem feitas utilizando uma linguagem lógica, como Prolog.

3.5.2. APLICAÇÕES DE PROGRAMAÇÃO LÓGICA

Os sistemas que se baseiam em conhecimento foram um dos que receberam grande parcela de ajuda das linguagens lógicas. Tais sistemas aplicam mecanismos automatizados de raciocínio para representar e inferir conhecimento. Eles costumam ser identificados como simplesmente "de inteligência artificial aplicada" e representam uma classe abrangente de aplicações, na qual todas as outras seriam sub-classes delas.

Uma outra área de utilização das linguagens lógicas seriam os sistemas de Banco de Dados, que teriam uma aplicação particular bem definida que utilizaria os sistemas baseados em conhecimento. Como se sabe, os bancos de dados armazenam coleções de relações que são guardadas em tabelas. Nesses bancos existe a necessidade de se realizar consultas sobre os modelos, que se fundamentam na álgebra relacional para realizar operações de junção e projeção. Foi observado, através de pesquisas, que tais operações de armazenamento e consultas a BD poderiam ser realizadas através de linguagens lógicas e que um dos principais problemas existentes em BD convencionais, relacionado à recuperação de dados, poderia ser resolvida através de mecanismos de inferência dos interpretadores lógicos. A partir dai, muitos sistemas têm sido criados com a representação de seu BD através da programação lógica.

Uma outra aplicação das linguagens lógicas pode ser encontrada nos sistemas especialistas, que são um tipo de sistema baseado em conhecimento projetados para emular a especialização humana em algum domínio especifico. Esse tipo de sistema possui tipicamente uma base de conhecimento formada por fatos, regras e heurísticas do domínio. Ele é capaz de

oferecer conselhos e sugestões aos usuários e também melhorar o seu desempenho a partir da sua experiência, ou seja, adquirir novos conhecimentos.

No item seguinte, será demonstrado de que maneira é possível representar algumas construções da linguagem OWL através da linguagem lógica Prolog. Com isso será mostrado que é possível implementar construções bem poderosas, como as presentes na OWL, através de uma implementação lógica.

3.5.3. PROLOG E OWL

Conforme visto anteriormente, OWL é uma linguagem de ontologias voltadas para a web. Também foram mostradas algumas construções que podem ser perfeitamente implementadas em linguagem lógica, que são úteis na construção de sistemas que se baseiam em Web Semântica e que necessitem desse tipo de implementação.

O objetivo dessa seção é demonstrar como essa implementação pode ser realizada através do Prolog. Inicialmente serão estudadas as primeiras construções apresentadas, tais como 'classe' e 'sub-classe'. A FIG. 3.7 apresenta a representação feita em OWL e em Prolog de um mapa conceitual.

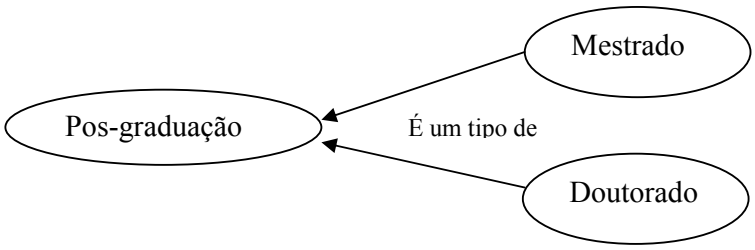


FIG. 3.7. - Mapa conceitual de uma Pós-graduação

Em linguagem OWL, esse tipo de mapa conceitual poderia ser implementado seguindo o código a seguir:

```
<owl:Class rdf:ID="Pos-graduacao"/>
    <owl:Class rdf:ID="Mestrado">
        <rdfs:subClassOf rdf:resource="Pos-graduacao"/>
    </owl:Class>
    <owl:Class rdf:ID="Doutorado">
        <rdfs:subClassOf rdf:resource="Pos-graduacao"/>
</owl:Class>
```

Na linguagem Prolog, esse mesmo mapa conceitual poderia ser implementado seguindo os seguintes fatos e regras.

```
Ehumtipode(mestrado, posgraduacao).// fatos que indicam a relação classe-subclasse
Ehumtipode(doutorado,posgraduacao).
```

Um outro caso de equivalências de implementações é da construção 'equivalentClass', que também pode ser representado no Prolog. Aqui segue um exemplo com as classes curso e matéria. Primeiramente é mostrada a representação de equivalência dessas classes no OWL, que já foi dado como exemplo na seção 3.4.2.

```
<owl:Class rdf:ID="Curso">
     <owl:equivalentClass rdf:resource="Materia"/>
</owl:Class>
```

No Prolog essa construção pode ser implementada através da regra "ehequivalente", que irá criar uma relação de equivalência entre duas classes. No código a seguir existe uma regra que dirá que, se um objeto é instância de uma classe A e essa classe é equivalente a uma outra B, esse mesmo objeto também será instância dessa classe B.

```
Ehamesmaclasse (curso, matéria).
Ehinstancia (bd,curso).
Ehinstancia (X,Y) :- Ehmesmaclasse ( Z, Y),  Ehinstancia (X,Z).
```

Assim, se for feita a consulta para conferir se BD é uma instância de matéria, a resposta será afirmativa devido a equivalência de classes, descrita pela regra. Agora o exemplo segue com a construção de transitividade, 'TransitiveProperty', tendo como base a FIG. 3.8.

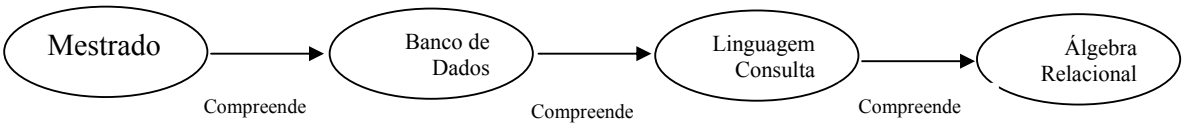


FIG. 3.8 - Mapa conceitual reduzido

Na OWL esse mapa conceitual pode ser implementado através do código apresentado:

```
<owl:ObjectProperty rdf:ID="compreende">
      <rdf:type rdf:resource="&owl;TransitiveProperty" />
      <rdfs:domain rdf:resource="&owl;Thing" />
      <rdfs:range rdf:resource="#LO" />
</owl:ObjectProperty>
<LO rdf:ID="Mestrado">
      <compreende rdf:resource="#Banco de Dados" />
```

```
</LO>
<LO rdf:ID="Banco de Dados">
      <compreende rdf:resource="#Linguagem Consulta" />
</LO>
<LO rdf:ID="Linguagem Consulta ">
      <compreende rdf:resource="#Álgebra Relacional" />
</LO>
```

A apresentação em Prolog equivalente ao código em linguagem OWL compreende as regras que definem a relação “compreende” entre LOs, além da regra de transitividade de fato, ou seja, se um LO (A) compreende um LO (B) e esse LO (B) compreende um outro LO (C), então o LO (A) irá compreender também o LO (C), conforme especificado pelo código:

```
Compreende (mestrado, banco_dados).
Compreende (banco_dados, linguagemConsulta).
Compreende (linguagemConsulta, algebraRelacional).
CompreendeTransiti(X,Y):-Compreende(X,Y);(Compreende(X,Z),Compreende (Z,Y).
```

Essa última regra deixa o código bem mais reduzido que o OWL, expressando a transitividade de forma bem mais clara. A seguir será apresentada uma outra construção, a de simetria. Neste exemplo será utilizado o mesmo caso de amigos aplicado anteriormente, isto é, se uma pessoa A é amiga de uma outra pessoa B, essa mesma pessoa também é simetricamente amiga de B. Assim, recordando o código:

```
<owl:ObjectProperty rdf:ID="amigo">
      <rdf:type rdf:resource="&owl;SymmetricProperty" />
      <rdfs:domain rdf:resource="#Amigos" />
      <rdfs:range rdf:resource="#Amigos" />
</owl:ObjectProperty>
<Amigos rdf:ID="Joao">
      <amigo rdf:resource="#Pedro" />
</Amigos>
```

No Prolog o mesmo exemplo pode ser represento pela regra: Amigo (X,Y):-Amigo (Y,X). Seguindo esta regra, ao se interrogar sobre a amizade de Pedro com relação a João, ela irá responder de forma afirmativa.

Finalizando essa pequena demonstração de similaridade entre a OWL e o Prolog, é apresentada a construção inversa, “inverseof”. Como exemplo, será utilizado o mesmo que referencia ‘ehmae’ e ‘ehfilha’ como cláusulas inversas.

```
<owl:ObjectProperty rdf:ID="ehmae">
      <rdf:type rdf:resource="&owl;FunctionalProperty" />
```

```
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="ehfilha">
      <owl:inverseOf rdf:resource="#emae" />
</owl:ObjectProperty>
```

Em Prolog, o mesmo código pode ser visto através da regra que demonstra o que é ser inverso do outro. Através dessa regra é possível inferir que Joana será a mãe de Maria devido a análise da regra e do fato apresentado. O código Prolog é o seguinte:

```
Ehfilha (Maria, Joana).
Ehfilha (X, Y) :- Ehmae (Y, X).
```

3.6. CONSIDERAÇÕES FINAIS

Visto que é possível fazer algumas das construções previstas pela OWL através de uma linguagem lógica, e que as mesmas ajudam na implementação de regras de inferência, fica claro que as essa linguagem lógica auxiliará na expansão de modelos que se baseiam em Web Semântica. Dessa maneira, no próximo capítulo será apresentado o sistema alvo principal desse trabalho de dissertação. O sistema ROSA, assim denominado, se baseia em fundamentos da Web Semântica. Nos próximos capítulos serão detalhados tanto o seu aspecto estrutural como a proposta de sua expansão utilizando-se de uma linguagem lógica.

4. MODELO DE DADOS ROSA

Nesse capítulo será apresentado o sistema ROSA. O ROSA é o centro das atenções dessa dissertação, já que a proposta da mesma gira em torno da criação de um modelo lógico para esse sistema, capaz de prover maior expressividade e poder de consulta. Na seção que se segue descreve-se o sistema e em seguida, um exemplo de mapa conceitual sobre a organização de um curso é apresentado. Esse exemplo servirá de base para o entendimento das possíveis consultas a serem feitas no sistema ROSA. Em seguida o modelo de dados ROSA será apresentado, e finalizando o capítulo, será apresentada a álgebra adotada no modelo, a álgebra ROSA. Essa álgebra servirá de base para a compreensão do capítulo seguinte, que mostra a implementação de consultas baseada nessa álgebra, tendo como base uma linguagem lógica.

4.1. DESCRIÇÃO DO ROSA

Existem sistemas responsáveis por suportar aplicações voltadas para aprendizagem eletrônica, ou *e-learning*. Esses sistemas gerenciam o conteúdo educacional através dos objetos chamados objetos de aprendizagem eletrônica, *e-learning object* (LO), que podem ser digitais ou não, e que auxiliam no processo de aprendizado, educação ou treinamento. Os sistemas de gerenciamento de LOs são conhecidos como *Learning Content Management System* (LCMS), e têm por finalidade o armazenamento e acesso à LOs. O ROSA [PORTO F, *et al*, 2004] é um sistema desse tipo, cujo modelo de dados é uma extensão do modelo de dados RDF [RDF, 2004]. Ele oferece uma linguagem de consulta, o ROSAQL, baseada na extensão da linguagem de consulta RQL, RDF Query Language [RDF, 2004].

LOs são formados por um conjunto de características e propriedades, também conhecidas como metadados. Esses metadados são padronizados por normas internacionais de metadados como o LOM (*Learning Object Metadado*) [IEEE, 2002] e por uma extensão do LOM, o SCORM (*Sharable Content Object Reference Model*) [SCORM OVERVIEW, 2004].

No contexto da teoria proposta pela Web Semântica, o ROSA permite expressar associações entre os LOs de modo a gerar uma contextualização para esses objetos. Adicionalmente, esse sistema propõe suportar técnicas de processamento de consultas, que irão explorar os metadados e os relacionamentos semânticos, auxiliando dessa maneira a

busca por LOs. Um usuário poderá formular consultas em cima do modelo, obtendo de forma mais eficiente os LOs que melhor lhe auxiliarão no processo de preparação de suas aulas ou conteúdos instrucionais.

4.2. MAPAS CONCEITUAIS E O ROSA

A breve introdução a mapas conceituais vista no capítulo anterior servirá de base para a compreensão do exemplo que será apresentado logo a seguir. Esse exemplo trata de um mapa conceitual relativo a um curso de mestrado, na área de ciência da computação, de uma instituição de ensino superior. Com base neste mapa conceitual serão sugeridas algumas consultas a respeito de sua organização, que mostrarão quais os tipos de consultas possíveis de serem elaboradas no domínio do modelo ROSA.

Como dito anteriormente, os LOs são formados por um conjunto de metadados que os descreve. Esses metadados juntamente com os relacionamentos oferecem um rico modelo semântico de buscas. Nos exemplos que se seguem, serão levados em consideração apenas dois metadados para o mapa conceitual em questão, o nível de agregação e título, uma vez que, a intenção aqui é explorar os relacionamentos que existem entre esses objetos e não o seu conjunto de metadados.

A FIG. 4.1. a seguir pode ser analisada mais uma vez como um grafo direcionado, onde os nós são os LOs e as arestas direcionadas os relacionamentos entre LOs. No mapa conceitual, LOs posicionados mais acima denotam níveis de agregação mais abrangentes, como curso.

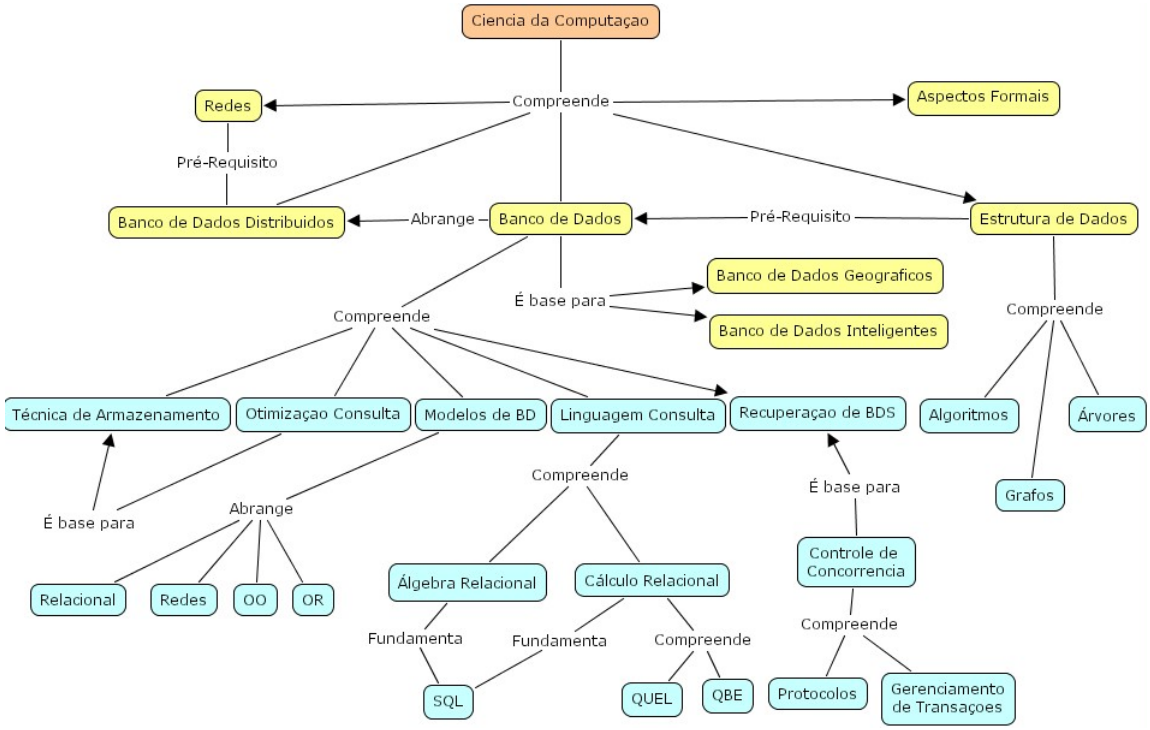


FIG. 4.1 – Mapa conceitual do curso de mestrado em Ciência da Computação

Observando o mapa conceitual da FIG. 4.1 percebe-se que maior detalhamento foi dado ao LO que representa o curso de Banco de Dados. As setas indicam a direção do relacionamento entre dois conceitos. Nos casos onde ela não aparece considera-se a direção do conceito seguindo do mais generalizado para o mais específico, ou seja, de cima pra baixo.

No mapa conceitual, a reposta a uma consulta sobre o conceito "curso de ciência da computação" pode ser obtida realizando-se uma simples navegação através do mapa. Questões como descobrir de quais disciplinas "Estrutura de Dados" é um pré-requisito, ou quais os tópicos que abrangem a disciplinas de "Banco de Dados", podem ser facilmente verificadas analisando-se o mapa conceitual.

As questões mencionadas anteriormente são simples de serem verificadas, já que uma simples navegação direta no mapa disponibiliza a resposta. Porém é possível se chegar a soluções de consultas bem mais complexas, que envolvem não só uma navegação direta, mas sim uma navegação através de dois ou mais relacionamentos, formando um caminho para a solução. Exemplo desse tipo de busca pode ser visto quando se deseja conhecer os tópicos que são abrangidos pelos tópicos que são compreendidos pela disciplina Banco de Dados.

Assim, percebe-se que uma grande variedade de indagações pode ser feita ao mapa, envolvendo consultas que acessam LOs diretamente ou indiretamente. Todas essas indagações

abriram espaço para o surgimento de uma álgebra capaz de expressar a manipulação desses dados, incluindo operações que realizam consultas tanto sobre o os metadados presentes nos LOs, quanto sobre seus relacionamentos diretos e indiretos. Sendo assim, surgiu a álgebra ROSA, que terá destaque nos próximos itens.

4.3. MODELO DE DADOS ROSA

Nessa seção será apresentado o modelo de dados ROSA. Basicamente, esse modelo é fundamentado por duas estruturas que são os LOs e o relacionamento entre os mesmos. Os LOs são formados por um conjunto de atributos, voltados para o domínio educacional chamados de atributosLOM, e por um atributo chamado CBN (Contexto de Busca Navegacional). Os atributosLOM são responsáveis por armazenar o subconjunto de atributos do padrão LOM do ROSA, inclusive o atributo responsável pela identificação única do LO, enquanto o CBN trata de um atributo atualizado em tempo de execução, responsável por armazenar as informações geradas por buscas navegacionais. Esse atributo corresponde a uma espécie de histórico navegacional, e apresenta todos os caminhos pelos quais é possível se chegar a um determinado LO.

A FIG. 4.2 a seguir mostra o modelo de dados ROSA expresso em um diagrama de classe UML. Na Figura, a classe raiz do diagrama é a classe 'Recurso Complexo'. Essa classe é especializada nas classes LO, descrita anteriormente, e na classe Relacionamento. A classe LO ainda se especializa em mais duas classes, a classe LO Lógico e LO Físico. Nessa especialização a classe LO Lógico é responsável pela parte conceitual do LO, como por exemplo criar o conceito de um tópico como o de fragmentação em banco de dados. Já o LO Físico é responsável pelo próprio documento de aprendizagem que pode ser uma apostila, um livro ou uma apresentação, tal como um pdf sobre o tópico fragmentação em banco de dados.

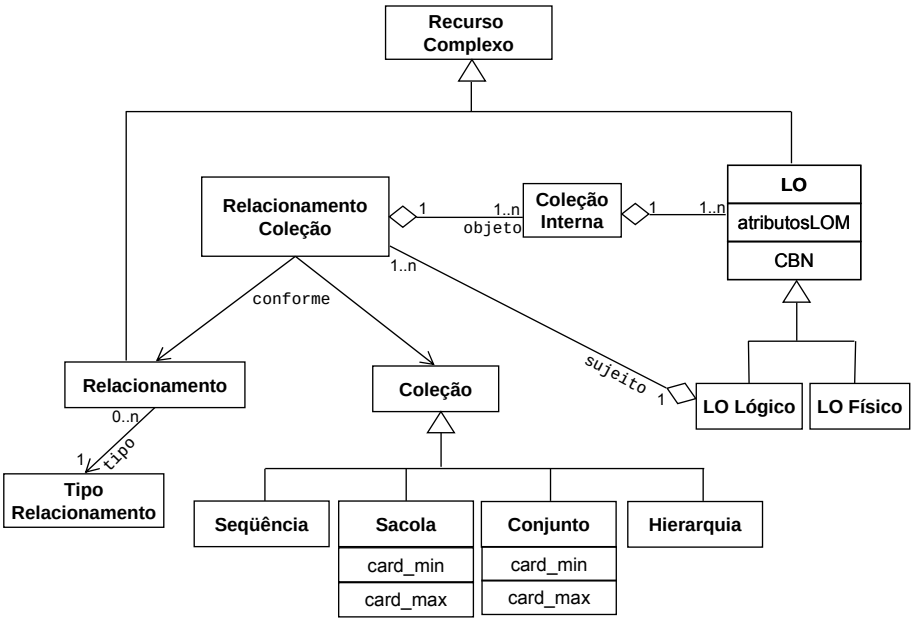


FIG. 4.2 – Modelo de Dados ROSA

A classe Relacionamento contém todos os predicados, expressos através de verbos, empregados em um mapa conceitual, como os que aparecem na FIG. 4.1. Cada instância é classificada de acordo com o tipo de relacionamento que define as propriedade de equivalência comuns a duas instâncias, que podem ser de transitividade, simetria ou reflexão. O sistema explora explicitamente cada propriedade quando avalia uma consulta sobre os relacionamentos. Dessa maneira é possível definir verbos, como o 'compreende' e 'fundamenta' como sendo relacionamentos do tipo transitivo, que devem atender a esse tipo de propriedade.

A classe Coleção representa as coleções físicas dos objetos e é especializada em outras classes que representam os possíveis tipos de coleção. Assim, o modelo especializa a classe Coleção nas classes Sequência, Sacola, Conjunto e Hierarquia. As classes Conjunto e Lista representam o conceito matemático correspondente, enquanto a Classe Sacola representa uma classe que pode conter duplicações em suas coleções. A classe Hierárquica representa LOs cujo conteúdo segue uma hierarquia de árvore. As classes Conjunto e Sacola incluem atributos que identificam suas cardinalidades máximas e mínimas. Esses atributos irão impor uma restrição sobre a coleção, que especifica o número máximo e mínimo de elementos que poderão ser extraídos da mesma. Um exemplo do uso desse tipo de atributo pode ser visto na FIG. 4.1, ao se modelar um relacionamento exclusivo entre os LO Linguagem Consulta e os LOs Álgebra Relacional e Cálculo Relacional. Estes podem ser modelados como uma coleção do tipo sacola, formada pelos LOs Álgebra Relacional e Cálculo Relacional, com

cardinalidade máxima igual a um e cardinalidade mínima igual a um. Isso significa que pelo menos um, e no máximo um dos LOs na coleção deverá ser explicitado quando for realizada uma consulta sobre o LO Linguagem Consulta através do relacionamento compreende.

Uma associação é formada por uma estrutura que possui como ponto inicial um único LO sujeito, que partirá através de um predicado a uma coleção de LOs descritos como LOs objetos. Dessa maneira, o sujeito da relação descrito será uma instância da classe LO Lógico e o restante da declaração será visto como instâncias da classe Relacionamento Coleção, que como visto pela FIG. 4.2, se relaciona com a classe Relacionamento e com a classe Coleção. Assim, uma instância da classe Relacionamento Coleção irá se relacionar a uma instância da classe Relacionamento, e irá representar o predicado da declaração ROSA. Essa mesma instância também irá se relacionar a uma instância da classe Coleção que descreverá o tipo de coleção a compor os LOs que participaram como objetos na associação. Finalmente, a instância da classe Relacionamento Coleção agrega uma ou mais instâncias da classe Coleção Interna, que irão representar os LOs objetos da declaração.

4.4. ÁLGEBRA ROSA

Para se tornar completo, um modelo de dados para os LOs precisa ter uma especificação completa de operações a serem realizadas. Essas operações são chamadas de modelos algébricos e definem um conjunto de expressões que possibilitam a criação de linguagens de consulta sobre o modelo. Em cima da álgebra é possível também fazer análises sobre as estratégias de consultas, com o objetivo de se chegar a consultas otimizadas, explorando regras de equivalência.

A álgebra proposta pelo modelo ROSA teria de se basear em outras já existentes, e uma das principais álgebras que serviu de referência na criação foi a álgebra relacional [CODD, 1972]. Outra também que serviu de base para essa álgebra foram as álgebras voltadas a modelos OO, como por exemplo a AQUA [LEUNG, 1993]. Apesar das álgebras OO serem bastante relacionadas ao modelo proposto pelo ROSA, algumas operações nelas são limitadas e não atendem a determinadas consultas, como consultas na qual o objetivo é conhecer determinado relacionamento existente entre dois LOs, ou seja, consultas que necessitam o processamento dos predicados.

Dessa maneira, para tratar dessas operações exigidas pelo modelo ROSA foi criada a álgebra ROSA. Essa álgebra é composta por operações que exploram os relacionamentos

entre os LOs, bem como os metadados ali presentes. As operações dessa álgebra que exploram os metadados dos LOs são as operações de projeção, seleção, e junção. Para as buscas navegacionais tem-se as operações de navegação e o fecho transitivo, e para as operações da teoria dos conjuntos tem-se a união, interseção e diferença. Por fim, tem-se as operações de seleção de predicado, que exploram o relacionamento dos LOs. Todas essas operações serão apresentadas a seguir com a sua descrição e alguns exemplos.

4.4.1. OPERAÇÃO DE PROJEÇÃO (Π)

A operação de projeção recebe um conjunto de LOs como entrada e produz um conjunto de LOs de saída, segundo uma determinada limitação de atributos, ou seja, nessa operação apenas os valores dos atributos listados pelo operador π constarão no resultado final, formando um resultado composto por LOs que projetam os atributos especificados. Na ausência desse operador limitador, os LOs de saída serão os mesmo de entrada. Seguem alguns exemplos que mostram a atuação da operação de projeção. Neles o símbolo 'C' representa uma coleção de LOs.

- Quais os níveis de agregação dos LOs ?
  $\Pi_{(aggregationlevel)}$ (C)

Nesse exemplo, o resultado será formado por um conjunto de LOs compostos apenas pelo atributo *aggregationlevel*. Outras projeções podem ter qualquer quantidade de atributos que se fizer necessária, como mostra o próximo exemplo.

- Qual o nível de dificuldade, título, linguagem, data de criação dos LOs ?
  $\Pi_{(difficult, title, language, creation\ date)}$ (C)

4.4.2. OPERAÇÃO DE SELEÇÃO (σ)

Na seleção, da mesma forma que na projeção, o parâmetro de entrada é um conjunto de LOs e a saída, outro conjunto de LOs que obedecem ao predicado de seleção estabelecida. As condições de seleção acompanham uma combinação de operadores lógicos (and, or, not) e

operadores comparativos como (<, >, $\leq$, $\geq$, =, $\neq$, in). A operação de seleção pode ser acompanhada ou não por uma projeção. Caso nenhum LO inicial satisfaça a condição de seleção, o conjunto final será vazio. Seguem alguns exemplos de seleção.

- Quais os LOs que possuem o nível de dificuldade fácil ?
  $\sigma_{(difficult = 'fácil')}$ (C)

- Qual a linguagem dos LOs que possuem o titulo banco de dados e o nível de agregação igual a disciplina?
  $\pi_{(language)}\ \sigma_{((title = 'banco\ de\ dados') \text{ and } (agregationlevel= 'disciplina'))}$ (C)

4.4.3. Operação de Junção ($\bowtie$)

As operações de junção envolvem dessa vez dois conjuntos de LOs. Cada elemento de cada conjunto deverá ser comparado um a um, acompanhando uma certa condição de junção, da mesma maneira que foi realizado com a seleção. Cada par de LOs correspondente a conjuntos distintos, que satisfazem a condição de junção, comporá o conjunto final do resultado. Da mesma forma como acontece com as operações mostradas anteriormente, outras operações podem ser realizadas após o resultado da junção, como uma seleção, uma projeção ou ambas. Segue um exemplo de junção.

- Recupere os pares de LOs pertencentes as Coleções C1 e C2 que representam os LOs do IME e da EPFL respectivamente, e que tenham o título em comum e que o mesmo seja 'banco_dados'.
  $\sigma_{(title = banco\_dados)}$ ((C1) $\bowtie_{(C1.title = C2.title)}$ (C2))

4.4.4. Operação teoria dos conjuntos – Interseção ($\cap$), União ($\cup$), Diferença (-)

As operações da teoria dos conjuntos representam operações similares ao que se vê em operações matemáticas simples sobre conjuntos. Da mesma forma como acontece com a junção, ela atua sobre conjuntos de LOs, as coleções. Esses conjuntos são ditos similares, se forem compostos pelos mesmos LOs. Para exemplificar o funcionamento dessas operações é exibido um exemplo da operação de União, que deseja conhecer o conjunto formado pela

união de dois outros conjuntos, sem repetições. As demais operações funcionam de forma análoga a essa exemplificada aqui.

- Quais as disciplinas que pertencem ao autor 'Handrick' nas coleções C1 e C2 livres de repetição?

  σ (agregationlevel='disciplina') and (acthor='Handrick') (C1) $\cup$ σ (agregationlevel='disciplina') and (author='Handrick') (C2)

4.4.5. NAVEGAÇÃO (λ)

Na navegação a entrada também é uma coleção ou conjunto de LOs a partir dos quais deseja-se seguir um certo caminho imposto pela regra de navegação, para se chegar aos LOs desejados. Assim, o operador λ irá estabelecer o caminho que deverá ser percorrido a partir dos LOs iniciais, presentes na coleção de entrada. Esses LOs também são conhecidos como sujeito ou objeto das associações, dependendo da direção em que se siga na navegação. A fórmula geral para esse operador é a seguinte: $\mathbf{C_R = \lambda^{i}_{(p)}\ (C_E)}$. Nessa fórmula, C_E e C_R representam respectivamente a coleções de LOs de entrada e a coleção resultante. O 'p' representa os predicados que irão participar da navegação. Caso em seu lugar exista um '*', significa que o predicado é irrelevante para a navegação. O 'i' na fórmula é responsável por indicar qual o papel desempenhado pelos LOs pertencentes a coleção inicial, se de sujeito(s) ou objeto(s). A sua ausência determina que os LOs resultantes, fazem parte dos objetos. Seguem alguns exemplos.

- Quais os LOs que são compreendidos pela disciplina 'Banco de Dados Distribuídos'?

  $\lambda_{(compreende)}$ (σ ((agregationlevel = 'disciplina') and (title = 'banco de dados distribuidos')) (C))

O próximo exemplo demonstra o uso da consulta no sentido reverso, seguindo o exemplo anterior.

- Quais os LOs que compreendem o LO 'distribuição horizontal' ?

$\lambda^{S}_{(compreende)}$ (σ (title = 'distribuição horizontal') (C))

Uma operação de navegação também pode incluir vários predicados distintos e combinados entre si, por operadores lógicos como ('∧', '∨', '.', '*'). Os próximos exemplos mostram um tipo de navegação com essa combinação de predicados.

- Quais os LOs que são compreendidos pela disciplina 'redes' e que também são pré-requisito de rede?

  $\lambda_{(\text{comprende} \wedge \text{pre\_requisito})} (\sigma_{(\text{(agregationlevel='Disciplina') } \textbf{and} \text{ (title='Redes'))}} (C))$

- O tópico 'distribuição' serve de base para LOs que compreendem quais LOs ?

  $\lambda_{(\text{eh\_base . compreende})} (\sigma_{(\text{(agregationlevel='tópico') } \textbf{and} \text{ (title='distribuição))}} (C))$

4.4.6. FECHO TRANSITIVO (φ)

O fecho transitivo é uma operação que obtém uma coleção resultante de LOs utilizando-se de operações de recursividade em cima dos relacionamentos dos LOs. Ele vasculha toda a profundidade dos relacionamentos até chegar a solução final. A iteração do fecho transitivo termina quando não existe mais nenhum caminho que ele possa seguir para obter mais LOs na coleção final. O caminho de navegação do fecho transitivo, segue por padrão a direção *down*, ou seja o LO inicial tem o papel de sujeito. Porém o caminho oposto, caminho *up*, também pode ser seguido e nesse caso o LO inicial segue como o objeto do relacionamento. O exemplo a seguir está baseado na FIG. 4.3.

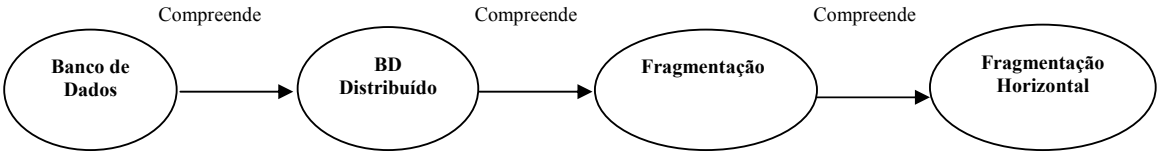


FIG. 4.3 – Exemplo de Fecho Transitivo

- Quais LOs são compreendidos pelo LO 'banco de dados' ?

  $\varphi_{(\text{compreende})} (\sigma_{(\text{agregationlevel='Disciplina') } \textbf{and} \text{ (title='Banco de Dados'))}} (C))$

4.4.7. SELEÇÃO DE PREDICADOS (σ)

Na seleção de predicados acontece algo bem diferente do que foi mostrado até agora nas consultas. De fato, essa operação possui como resultado, não um conjunto de LOs resultante, mas sim um predicado ou relacionamento como resultado final. Para realizar as consultas nessa operação é necessário apenas informar quais as coleções que irão funcionar como sujeito e quais irão se comportar como objetos, seguindo a seguinte fórmula: $\mathbf{C_R = (C_s)\ \mu\ (C_o)}$. Nessa fórmula aparecem as coleções que representam o sujeito e o objeto, bem como o operador binário **μ**. Porém esse operador também pode se comportar como unário, quando na consulta não interessar um dos lados. Seguem alguns exemplos desse tipo de consulta.

- Qual a relação que existe entre a disciplina 'banco de dados' e o tópico 'linguagem de consulta'?

  (σ((agregationlevel='Disciplina') **and** (title='Banco de Dados')) (C)) μ (σ((agregationlevel='Tópico') **and** (title='Linguagem de Consulta')) (C))

O exemplo a seguir mostra a ausência de um dos componentes da fórmula, demonstrando a utilização do operador μ de forma unária.

- Em quais associações a disciplina 'Estrutura de Dados' participa tendo o papel de sujeito do relacionamento?

  (σ((agregationlevel='Disciplina') **and** (title='Estrutura de Dados')) (C)) μ

4.5. CONSIDERAÇÕES FINAIS

Nesse capítulo foi apresentada uma noção geral do que é o sistema ROSA, mencionando aspectos relativos a sua forma estrutural, consultas viabilizadas pelo modelo, bem como uma rápida especificação da álgebra ROSA, que se apresenta como uma forma de atender as consultas requisitadas pela proposta do ROSA.

A noção dada nesse capítulo sobre a álgebra ROSA será bastante importante para o próximo capítulo, que tratará da representação do modelo ROSA em uma linguagem lógica. Além da representação lógica, esse capítulo discutirá ainda a implementação das mesmas operações de consultas algébricas vistas ao longo desse capítulo, em linguagem lógica, reproduzindo dessa maneira todo o modelo aqui apresentado.

5. MODELO ROSAI

Esse capítulo tem como objetivo mostrar como implementar consultas utilizando uma linguagem lógica. Para tanto foi definido um banco de dados lógico, representativo de um ambiente e-learning e especificado o tratamento dado a consultas sobre esse modelo. Os exemplos se baseiam no modelo de LOs referentes ao sistema ROSA, apresentado no capítulo anterior.

Aqui serão exploradas as consultas expressas na álgebra ROSA, através da sua representação em lógica. As operações algébricas exibidas no capítulo anterior serão aplicadas a exemplos que servirão de protótipo para validação dessas operações. Em cada exemplo, uma pequena noção funcional da operação algébrica analisada será apresentada. Adicionalmente será discutido cada código lógico produzido, bem como todos os resultados parciais que por ventura apareçam. Será igualmente discutida a representação de regras que, associadas às relações, definam a propriedade de transitividade, simetria e inversão, bem como a relação de herança.

Conforme mostrado no ROSA, algumas operações algébricas propostas no modelo permitem a realização de consultas características do domínio da Web Semântica, e portanto não são operações presentes em álgebras propostas anteriormente. Essas operações seriam capazes de atender a algumas consultas exigidas pelo modelo. Mesmo essas novas operações foram perfeitamente aceitas pela linguagem lógica e tiveram a sua implementação realizada com sucesso.

Na próxima seção será mostrado como pode ser feita a representação do modelo ROSA no Prolog e conseqüentemente como funciona a implementação de uma base de conhecimento para esse modelo. Nas seções finais esse modelo será posto em prática, através de exemplos que demonstram a utilização das operações propostas, através da linguagem lógica.

5.1. REPRESENTAÇÃO DO MODELO ROSA EM PROLOG

Nesta seção será apresentado o modelo de representação ROSA em linguagem lógica Prolog. A partir das premissas apresentadas pode-se estender o modelo para representação do modelo ROSA.

Inicialmente discute-se a representação de um ‘conceito’. O ‘conceito’, que é a base para os objetos de aprendizagem, é representado como um fato para a linguagem lógica, sendo expresso como a seguir:

conceito(a_1, a_2, a_3, $a_{4,}$..., a_n).

Onde “a1, a2, a3, a4,..., an” representam o conjunto de ‘n’ atributos, que realizam a descrição de um determinado ‘conceito’. Um atributo de um conceito também poderá ser multivalorado e assim ser representado da seguinte maneira.

conceito(a_1, a_2, (a_{31}, $a_{32,...,}a_{3n}$), $a_{4,}$..., a_n).

Na representação de relações a idéia segue um pouco parecida com a de ‘conceito’, já que relações também podem ser vistas como fatos na linguagem lógica.

Dessa maneira, uma relação pode ser expressa através de um fato composto por uma proposição ternária, expresso por:

relacao(a,relac,b).

Nessa representação o termo ‘a’ representa o primeiro conceito que irá se relacionar com o segundo, representado por ‘b’, através do relacionamento definido pelo termo ‘relac’. Assim, os fatos que irão representar os relacionamentos presentes no modelo serão representados dessa maneira. A idéia de representar relacionamentos através de regras pode ser verificada quando se cria a representação das propriedades que podem existir em um relacionamento. No caso da propriedade inversa, a representação segue entre os relacionamentos: “relac01” e “relac02”.

relacaoInver(a,relac01,b):-relacaoInver(b,relac02,a).

Essa regra define que os relacionamentos “relac01” e “relac02” são inversos e assim, se existir o fato verdadeiro de que o “b” se relaciona com “a” pelo relacionamento “relac02”, pela regras é também verdadeira a existência do fato de que “a” se relaciona com “b”, através do relacionamento “relac01”.

De forma semelhante acontece com a propriedade de simetria, que também é representada como uma regra, vista a seguir.

relacaoSimet(a,relacSim,b):-relacaoSimet(b,relacSim,a).

Admitindo-se que o conceito “b” se relaciona ao “a” pela relação simétrica “relacSim”, então o relacionamento de “a” com “b” através do mesmo relacionamento simétrico também é verdadeiro.

Por último, será apresentada a propriedade de transitividade. Da mesma forma que as outras propriedades, ela também é expressa através de uma regra pela linguagem lógica. A regra de transitividade é expressa por:

relacaoTrans(a,relac,b) :-relacao(a,relac,b);(relacao(a,relac,z_i),...,relacao(z_{n-1},relac,b)).

Nessa regra, a relação transitiva ‘relac’ entre os objetos “a” e “b” é atendida quando existir uma relação direta entre esses objetos, ou quando existir um conjunto de objetos formados por “Z_i, ..., Z_{n-1}” que serão objetos intermediários para se chegar no objeto “b” final, através de “relac”.

Um caso particular de relacionamento que será mostrado aqui é o que da a idéia de herança. Esse relacionamento, que receberá o nome de “Isa” é importante no modelo, na medida que o torna bem mais expressivo. Através desse tipo de relacionamento é possível criar uma regra através do qual todo conceito que seja filho de um determinado conceito pai irá herdar todos os relacionamentos que o ‘conceito’ pai possuir. A regra que impõe essa propriedade de herança é representada da seguinte maneira:

relacaoHeranc(a,relac,b):-relacao(a,relac,b);(relacao(z,relac,b), relacao(a,isa,z)).

Como pode ser visto por esta representação, dois conceitos “a” e “b” possuem um certo relacionamento “relac” se estes se relacionarem de forma direta, que é o primeiro caso, ou quando o conceito “z” se relacionar com um determinado conceito “b”, e o conceito “z” é um conceito pai do conceito “a”, ou seja, o conceito “a” herda todos os possíveis relacionamentos de possam existir com o conceito “z”, incluindo assim o relacionamento “relac”.

A partir dessa visão sobre representações lógicas para conceitos e regras, será destacada a representação de algumas consultas previstas pelo modelo já apresentadas em capítulos anteriores.

Todas as operações aqui destacadas podem ser entendidas como a verificação de um fato, sempre levando em conta as regras. Assim, a semântica das operações corresponde a validar sempre o que é falso ou verdadeiro, com base nos fatos. Inicialmente será apresentada a operação de seleção. Essa operação realiza a verificação de proposições, utilizando-se de constantes dentro do conjunto de elementos que formam um conceito. A seguir é dado um exemplo de como é feita a representação de uma operação de seleção em lógica:

conceito(x,y,...,n).

Nessa operação, deseja-se selecionar todos os conceitos, para os quais o primeiro e segundo elementos do conjunto de "n" elementos, é definido pelas constantes "x" e "y" respectivamente. Assim, essa operação irá percorrer todos os conceitos e verificar aqueles que trazem essa verificação como verdadeira, realizando a operação de seleção de maneira lógica.

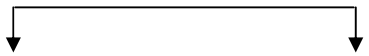
A projeção é outra operação prevista no modelo e que realiza esta tarefa através do uso de variáveis. Ela atua da mesma maneira que a seleção, porém agora são usadas variáveis ao invés de constates. Essas variáveis são responsáveis por projetar os valores de interesse, mesmo que esses valores se refiram a atributos multivalorados do conceito, conforme expressão a seguir:

conceito(A,B,...,n).

No caso desse exemplo, a idéia é a de projetar através das variáveis representadas por letras maiúsculas, "A" e "B", todos os valores do primeiro e segundo termo que formam um certo conceito. Essas variáveis tornam-se responsáveis por trazer a resposta da projeção dos termos escolhidos.

Outra operação que trabalha com conceitos é a operação de junção. Nessa operação são utilizados dois ou mais conceitos, representados da forma já mostrada, e associadas através de conjunções que possuem em comum uma variável de junção. A presença da mesma variável em termos de uma conjunção determina uma relação de igualdade de valores entre fatos de

conceitos relacionados. O exemplo a seguir mostra a junção de dois conceitos, formados por "n" termos, dentre os quais existe uma variável de junção "X".

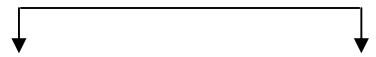


conceito1(a_1, a_2, ..., X,...,a_n), conceito2(a_1, a_2, ..., X,...,a_n).

Passaremos agora, às operações que envolvem teoria dos conjuntos, em particular, as operações de união e interseção. Inicialmente, será dado destaque às operações de união, que representam a disjunção de proposições. Assim, a união entre proposições é vista como uma operação com o operador "ou", representado por ";" na linguagem lógica. O exemplo a seguir mostra a disjunção entre dois conjuntos de 'n' termos formados respectivamente pelos "conceito1" e "conceito2".

conceito1(a_1, a_2, a_3, a_4,..., a_n); conceito2(b_1, b_2, b_3, b_4,..., b_n).

A operação de interseção sugere uma semântica parecida com a de junção mostrada anteriormente, já que também realiza a conjunção de proposições com uma variável em comum, só que dessa vez esta deverá ser a variável responsável pela identificação do conceito, ou seja, que assegure que os dois conceitos que se juntem sejam o mesmo, caracterizando a interseção. No exemplo que se segue, o primeiro termo presente no conceito designado por "X" foi o termo escolhido para ser o identificador de um conceito.

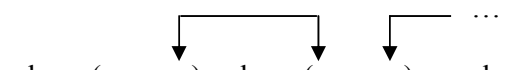


inter(X):-conceito1(X, a_2, a_3, a_4,..., a_n),conceito2(X, b_2, b_3, b_4,..., b_n).

O termo 'inter(X)' no lado esquerdo da regra irá garantir que a resposta final da interseção seja formada apenas por resultados sem repetições.

Na representação em lógica, as operações de relacionamento não são tratadas como conceitos. Conforme estudado anteriormente, uma relação é uma proposição ternária, onde: o primeiro termo representa o termo de origem; o segundo termo representa a relação; e o terceiro termo refere-se ao conceito relacionado.

Uma das operações que atua sobre relações é a de navegação. Nela podem existir duas ou mais relações representadas por proposições que se ligam por termos da conjunção ternária, como o apresentado no exemplo a seguir.



relacao(x_1,p_1,y_1), relacao(x_2,p_2,y_2),..., relacao(x_n,p_n,y_n),

De uma maneira formal, a navegação pode ser vista como uma conjunção de "n" proposições ternárias do tipo relacao(x_i,p_i,y_i), onde "n" representa não só o tamanho do caminho a ser seguido, como também o número de predicados tal que para todo $1 \leq i \leq n$, y_i é igual a x_{i+1} , até i=n-1. No exemplo anterior, p_i representa os predicados de navegação, tendo x_1 como ponto de partida da navegação.

Uma outra operação existente no modelo ROSA é a que trata da seleção de predicados. A operação sugere a busca de predicado através de seus objetos de partida e de destino. Assim, a representação para tal operação pode ser vista a seguir:

relacao(a,X,b);(relacao(A_i,X,B_i),...,relacao(A_n,X,B_n).

Essa representação sugere que tal operação é representada por uma proposição ternária do tipo relacao(a,X,b), onde "a" e "b" seriam constantes e "X" uma variável; ou uma operação formada por um conjunto de proposições de tamanho "n" arbitrário, do tipo relacao(A,X,B), onde "A", "B" e "X" são variáveis, tal que X é igual ao predicado transitivo "P_i", para todo $1 \leq i \leq n$, e que A="a" para i=1 e B="b" para i=n, e A=B para $1 < i \leq n$.

Essa seção, apresentou as operações do modelo ROSA segundo uma representação em linguagem lógica. Essa representação ficará mais clara quando forem exibidos exemplos práticos de aplicações do ROSA, na próxima seção.

5.2. EXEMPLIFICANDO OS COMPONENTES DO MODELO ROSAI

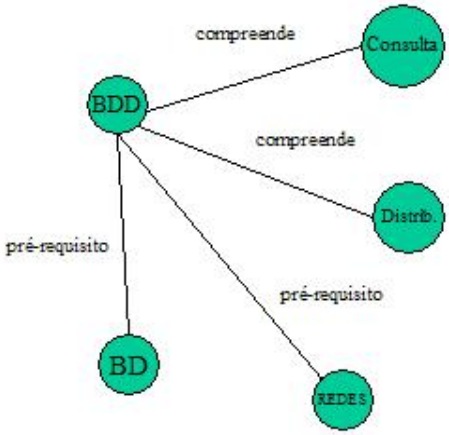
O modelo adotado para a representação de mapas conceituais baseia-se em duas representações básicas do conhecimento retratado em tais mapas: definição de "LO" e definição de "relacionamento".

A primeira especifica um LO através de suas propriedades LOM. Neste trabalho para efeito de exemplificação, será utilizado um subconjunto dos atributos definidos pelo padrão. Dessa maneira, segue a definição de um LO segundo o modelo para definição de conceito em lógica:

```
lo ( id, title, difficulty, keyword, aggregationLevel, Author).
```

Nessa definição, “id” representa o atributo de identificação do LO, “title” é o atributo do título do LO, “keyword” a sua palavra chave definidora, “aggregationLevel” o nível de agregação do LO e “Author” sugere o nome do criador do LO.

Assim sendo, observando o mapa conceitual da FIG. 5.1, é demonstrado como é realizada a representação dos LOs presentes nesse mapa, através dos fatos lógicos sugeridos.




```
lo(1,bdd,facil,banco_dados,disciplina,fabio_porto).
lo(2,bd,facil,armazanamento,disciplina,handrick).
lo(3,consulta,dificil,banco_dados,tópico,handrick).
lo(4,distribuicao,médio,arquitetura_bdd,tópico,fabio_porto).
lo(5,redes,facil,comunicacao,disciplina,handrick).
```

FIG. 5.1 – Tradução de LOs de mapas conceituais para fatos lógicos

A expressão de fatos em Prolog para o modelo ROSA é realizada definindo-se predicados que associam valores a uma constante. Assim, a constante que define os objetos de aprendizagem “lo” é associada a uma lista de valores na ordem dos atributos considerados anteriormente. Como exemplo, temos que “id” corresponde ao primeiro valor, “title” ao segundo e assim por diante. A expressão das operações algébricas que têm por base essa representação em Prolog sempre leva em consideração os atributos e valores nessa ordem pré-estabelecida.

A outra representação que também deverá estar presente na implementação diz respeito aos relacionamentos entre os LOs. Para isso, relacionamentos serão igualmente definidos como fatos conforme apresentado na FIG. 5.2.

```
relacao(LO1,compreende,LO2).
relacao(LO3,prerequisito,LO4).
```

FIG. 5.2 – Tradução de relacionamentos entre LOs para Fatos Lógicos

Esses dois fatos definem no sistema Prolog a existência de um relacionamento chamado de ‘compreende’ e outro chamado de ‘prerequisito’, que serão aplicados a dois objetos de aprendizagem.

O fato de que um determinado LO se relaciona a outro por uma relação, pode ser representado pelo predicado da FIG. 5.3, que na verdade é uma representação completa do que foi visto na figura anterior. Acompanhando o código abaixo, seguem as relações apresentadas de forma mais completa.

```
relacao(bdd,compreende,distribuicao).
    relacao(bd,prerequisito,bdd).
  relacao(bdd,compreende,consulta).
  relacao(redes,prerequisito,bdd).
```

FIG. 5.3 – Outras traduções de relacionamentos entre LOs para Fatos Lógicos

Como se pode observar, esses 4 fatos representam as relações presentes na FIG. 5.1 e constituem mais uma peça necessária para a formulação da base de conhecimento. Com a união de todos esses fragmentos de código, temos a representação de uma base de conhecimento que poderá ser entendida por qualquer interpretador Prolog. Todas as consultas presentes em nossos exemplos terão como LOs de entrada o conjunto formado pelos LOs presentes na representação da base de conhecimento.

5.3. EXEMPLIFICANDO AS OPERAÇÕES DO MODELO ROSAI

Os tipos de consultas exploradas pelo modelo ROSA são basicamente operações de busca a metadados, como se pode observar nas operações de projeção, seleção e junção. Um outro grupo de consultas presentes no modelo são aquelas que exploram a teoria dos conjuntos e é representada pelas operações de união, interseção e diferença. Por fim, existem também

consultas que exploram o aspecto navegacional presente no modelo. Essas últimas representam um aspecto relevante do sistema e se encontram representadas por operações de navegação simples e navegação complexa, bem como operações de fecho transitivo. As seções a seguir apresentam cada uma dessas operações.

5.3.1. PROJEÇÃO

As consultas envolvendo projeção determinam o subconjunto de metadados de LOs que deverão aparecer no conjunto resposta. Considera-se que tal operação determina os atributos LOM relevantes em uma dada consulta, conforme o exemplo da FIG. 5.4.

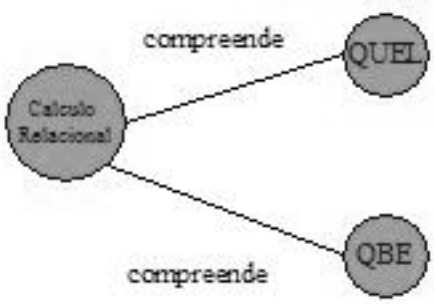


FIG. 5.4 - Mapa conceitual reduzido

Assumimos que os LOs apresentados nesta figura incluem metadados, conforme pode ser acompanhado pelo código a seguir:

```
lo(1,calculo_relacional,medio,consulta,topico,(fabio_porto,ana)).
          lo(2, quel, facil,consulta,topico,handrick).
          lo(3, qbe, medio,consulta,típico,handrick).
```

FIG. 5.5 – Fatos correspondentes aos LOs no mapa conceitual reduzido

Levando em conta que todas as referências aos LOs e aos relacionamentos já foram efetuadas anteriormente, é possível realizar consultas envolvendo projeção, a exemplo de:

1. Quais os autores desses LOs?
2. Quais os níveis de dificuldade e as palavras chaves dos LOs?

Para ter a resposta da primeira consulta o seguinte código em Prolog é proposto:

```
projecao_autor(X,A):-lo(_,X,_,_,_,A).
```

A resposta para a consulta ‘projecao_autor(X,A)’ pode ser vista logo a seguir:

```
X = calculo_relacional
A = fabio_porto, ana ;

X = qbe
A = handrick ;

X = quel
A = handrick ;
```

A resposta da consulta também realizou a projeção do atributo multivalorado referente a autores, presente no primeiro LO do exemplo.

Para resolver a segunda consulta, será elaborado um código que se assemelha bastante ao primeiro, a não ser pelo fato que dessa vez, se deseja obter o valor de dois atributos. Em uma projeção pode-se requisitar qualquer subconjunto dos atributos de LOs que se deseje. Para atender a esse requisito, basta que sejam acrescentadas, mais variáveis de atribuição e de resposta à consulta. A consulta ‘projecao_nive_pl_chave(X,D,P)’, a seguir projeta o nível de dificuldade e a palavra chave dos LOs.

```
projecao_nive_pl_chave(X,D,P):-lo(_,X,D,P,_,_).

X = calculo_relacional
D = medio
P = consulta ;

X = qbe
D = medio
P = consulta ;

X = quel
D = facil
P = consulta ;
```

5.3.2. SELEÇÃO

A seleção é outro caso de consulta que pode ser facilmente implementada em uma linguagem lógica. Nela, os LOs são selecionados seguindo uma expressão booleana, imposta sobre valores de atributos. Essa expressão de seleção pode incluir operadores lógicos,

encontrados na lógica booleana como “and”, “or”, “not”, e utiliza operações de comparações aritméticas (<, >, <, >, =, <>, ≠) . Assim como em Prolog, uma expressão de condição equivale a verificar na base de fatos a veracidade de uma afirmação. Neste caso, um predicado define uma expressão que pode se verificar verdadeira para um certo número de fatos da base que serão retornados como resposta. Para acompanhar o exemplo de seleção, segue o exemplo de um mapa conceitual da FIG. 5.6.

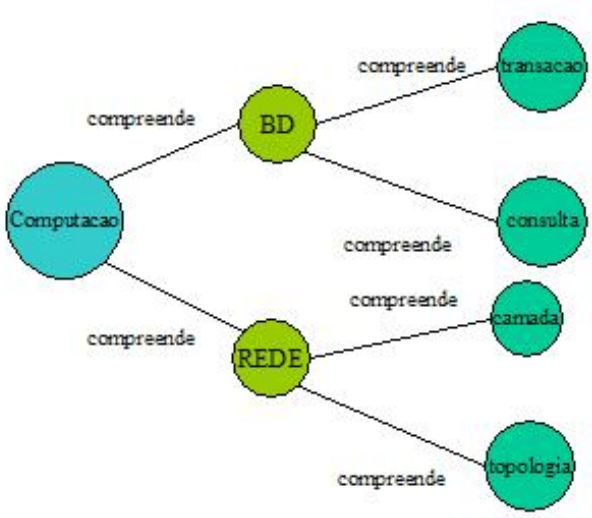


FIG. 5.6 - Sub-conjunto de LOs que formam um curso de computação

Na FIG. 5.6 estão presentes LOs de nível de agregação diferentes, cuja especificação de seus metadados pode fazer parte de um predicado de seleção. Uma consulta pode combinar operações de seleção com as de projeção em uma única expressão, conforme apresentado nas consultas que se seguem. A FIG. 5.7 apresenta a descrição dos LOs presentes da FIG. 5.6, em linguagem Prolog, com seus metadados.

```
lo(1,computação,dificil,ciências_exatas,curso,handrick).
  lo(2,bd,dificil,armazenamento,disciplina,ana_moura).
   lo(3,rede,dificil,comunicacao,disciplina,geovanne).
    lo(4,transacao,medio,execucao_bd,topico,handrick).
     lo(5,consulta,facil,resposta_bd,topico,handrick).
     lo(6,camada,dificil,arquitetura,topico,handrick).
   lo(7,topologia,medio,estruturacao,topico,handrick).
```

FIG. 5.7 – Fatos que representam o sub-conjunto de LOs

Em uma primeira consulta, deseja-se conhecer o grau de dificuldade das disciplinas que possuem como autor ‘ana_moura’. Essa consulta pode ser acompanhada pelo código a seguir e leva em conta mais uma vez a presença de todas as declarações de LOs e relacionamentos.

```
selecao_dif_disc_ana(D):-lo(_,_,D,_,disciplina,ana_moura).
```

Ao se executar a consulta ‘selecao_dif_disc_ana(R)’, a variável ‘D’ receberá o valor da dificuldade encontrada, no caso ‘dificil’, conforme a base de fatos exemplificada na FIG. 5.7. Essa consulta pode se tornar um pouco mais flexível, fazendo com que o próprio usuário apresente a granularidade e o autor desejado. Assim sendo, o código para se realizar essa consulta, deverá ser redefinido da seguinte forma.

```
selecao_dif_discq_autorq(N,A,D):-lo(_,_,D,_,N,A).
```

É possível repetir a mesma consulta anterior, fazendo-se referência à granularidade e ao autor que se deseja. No caso anterior, a consulta seria: selecao_dif_discq_autorq (disciplina,ana_moura,D).

Expressões de seleção mais complexas podem ser representadas em Prolog, conforme os exemplos abaixo:

1. Obter os LOs que representam disciplinas ou tópicos e que possuem o grau de dificuldade difícil.
2. Obter as palavras-chaves dos LOs cujo número de identificação seja maior que 3.

A primeira consulta possui como implementação o código presente logo a seguir. Nele, o símbolo ‘;’ representa a cláusula ‘ou’ solicitada pela consulta e prevista pela operação de seleção. O código e a sua resposta para ‘selecao_disc_top_dif(R)’ seguem:

```
lo(_,L,dificil,_,disciplina,_);lo(_,L,dificil,_,topico,_).
L = bd ;
L = rede ;
L = camada ;
```

Para a segunda consulta a implementação é bem mais simples e possui como novidade, a presença de uma operação de seleção que utiliza uma comparação aritmética.

```
selecao_ide_maior_3(P):- lo(I,_,_,P,_,_),I>3.
```

Respostas:

```
P = execucao_bd ;
P = resposta_bd ;
P = arquitetura ;
P = estruturacao ;
```

5.3.3. JUNÇÃO

Em seguida, será dado destaque a um tipo de consulta um pouco mais complexa. A junção corresponde a um tipo de operações que utiliza outras operações algébricas, como a operação de seleção descrita anteriormente. No exemplo seguinte será apresentada uma junção que atuará no conjunto de LOs presentes na FIG. 5.8.

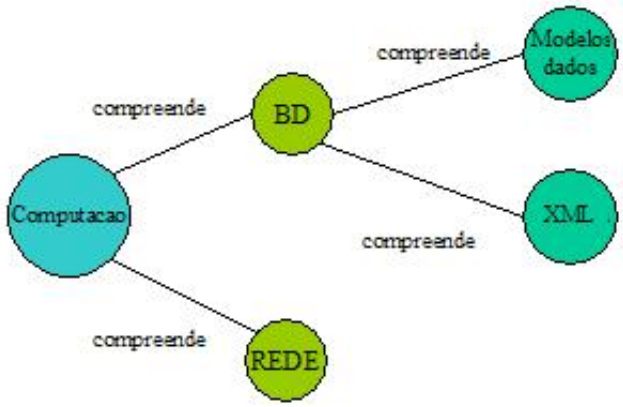


FIG. 5.8 – Mapa Conceitual como exemplo para Junção

Na FIG. 5.8 existem representados cinco LOs que mais uma vez possuem granularidades diferentes. A idéia é mostrar que é possível criar operações de junções entre tais LOs, ou entre aqueles que possuam um outro tipo de metadado que possa ser utilizado como base para a expressão de um predicado de junção. Será apresentada inicialmente uma consulta que visa mostrar uma operação de junção entre LOs, seguindo a idéia de distintas granularidades. Desse modo, a consulta visa saber qual o conjunto de LOs que representa os "tópicos" e "disciplinas" que possuem o grau de dificuldade "difícil". Os metadados do LOs podem ser vistos através do código:

```
lo(1,computacao,dificil,ciências_exatas,curso,handrick).
lo(2,bd,dificil,armazenamento,disciplina,ana_moura).
lo(3,rede,facil,comunicacao,disciplina,geovanne).
lo(4,modelos_dados,dificil,banco_dados,topico,ana_moura).
lo(5,xml,facil,web_semantica,topico,maria_luiza).
```

A consulta requer a seleção de LOs classificados segundo cada uma dessas granularidades e o estabelecimento do ponto em comum entre esses dois conjuntos. Isso seria o ponto de junção entre esses dois conjuntos finais formados. No caso da consulta aqui descrita seria o grau de dificuldade 'difícil'. Assim, o código necessário para se realizar essa operação seria o seguinte.

```
lo(I1,N1,DIF,_,disciplina,_),lo(I2,N2,DIF,_,topico,_),I1=\=I2,DIF=difícil,R
=(N1,N2,DIF2).
```

A execução dessa junção irá lançar na variável DIF, o valor referente ao grau de dificuldade. A cláusula 'I1=\I2' existe para evitar que ocorra uma relação de um LO com ele mesmo. A variável DIF irá realizar a junção do atributo escolhido, no caso o nível de dificuldade sendo "difícil". A resposta final será atribuída a variável R, que receberá os valores dos dois nomes dos LOs que se juntam, bem como o seu nível de dificuldade, que realiza a junção. Na TAB. 5.1 a seguir serão apresentados os resultados dessa operação, quando se aplica a junção explicitando um valor à variável de junção, e quando não se declara esse valor.

TAB 5.1 – Junção com atributo de Junção especificado e não especificado

Valor da dificuldade declarado. **lo(I1,N1,DIF,_,disciplina,_), lo(I2,N2,DIF,_,tópico,_),I1=\=I2, DIF=difícil,R=(N1,N2,DIF2).**	Valor da Dificuldade não declarado. **lo(I1,N1,DIF,_,disciplina,_), lo(I2,N2,DIF,_,tópico,_),I1=\=I2, R=(N1,N2,DIF2).**
R = bd, modelos_dados, dificil ;	R = bd, modelos_dados, dificil ; R = rede, xml, facil ;

5.3.4. OPERAÇÕES DA TEORIA DOS CONJUNTOS

Nesse item será discutido a implementação em linguagem lógica Prolog de operações de consultas do tipo teoria dos conjuntos. Serão apresentadas as operações de: União ($\cup$), interseção ($\cap$) e diferença (-). Na maioria dos interpretadores Prolog, já existem bibliotecas prontas que realizam essas operações. Aqui apresentaremos exemplos utilizando essas bibliotecas e o fundamento da implementação dessas funções nessas bibliotecas. O exemplo toma como referência a base de conhecimento presente a seguir:

```
lo(1,bd,facil,armazenamento,disciplina,ana_moura).
lo(2,redefacil,comunicacao,disciplina,geovanne).
lo(3,estrutura_dados,facil,fundamento_comp,disciplina,justel).
lo(4,sistema_computação,dificil,computacao,curso,justel).
lo(5,modelos_dados,medio,banco_dados,topico,ana_moura).
lo(6,xml,facil,web_semantica,topico,fabio_porto).
lo(7,algoritmos,facil,codigo,topico,justel).
lo(8,arvores,dificil,estrutura,topico,justel).
lo(9,grafos,dificil,estrutura,topico,justel).
```

Inicialmente será demonstrada a operação de "união". Para exemplificar essa operação é solicitada uma consulta que visa conhecer todos os LOs que possuem como autoria 'fabio_porto' ou 'ana_moura'. A resposta a essa consulta será obtida pela união de todos os LOs cujo autor seja 'fabio_porto' com todos os LOs cujo autor seja 'ana_maria'. Assim sendo, o código responsável por elaborar essa consulta é dado por.

```
[lo,_,_,_,_,_,ana_moura];[lo,_,_,_,_,_,fabio_porto].
```

Essa operação também pode ser elaborada através da manipulação de listas no Prolog. Isso é realizado quando se utiliza a função 'union' presente na maioria dos interpretadores. Nessa manipulação, a função recebe duas listas iniciais e devolve como resultado a união das mesmas, sem que ocorram repetições. Assim, o exemplo mostrado anteriormente pode ser elaborado novamente da seguinte forma:

```
lo_ana_moura(V):-R=..[lo,_,_,_,_,_,ana_moura],R,V=R.
lo_fabio_porto(V):-R=..[lo,_,_,_,_,_,fabio_porto],R,V=R.
lista_lo_ana_maria(C):-findall(F,lo_ana_moura(F),C).
lista_lo_fabio_porto(C):-findall(F,lo_fabio_porto(F),C).
```

As duas listas irão conter em suas variáveis de saída, 'C', a lista com as devidas seleções. Por fim só resta elaborar a união das duas listas, como pode ser vista pela resposta da execução.

```
execucao_u(U):-
lista_lo_ana_maria(L1),lista_lo_fabio_porto(L2),union(L1,L2,U).

L1 = [lo(1, bd, facil, armazenamento, disciplina, ana_moura), lo(5,
modelos_dados, medio, banco_dados, topico, ana_moura)];

L2 = [lo(6, xml, facil, web_semantica, topico, fabio_porto)];

U= [lo(1, bd, facil, armazenamento, disciplina, ana_moura), lo(5,
modelos_dados, medio, banco_dados, topico, ana_moura), lo(6, xml, facil,
web_semantica, topico, fabio_porto)] ;
```

A outra operação destacada aqui é a operação de interseção. Essa operação também possui uma função pronta, chamada de 'intersection' que se encontra disponível na maioria dos interpretadores lógicos, e que também realiza a manipulação em listas. Será apresentado as duas maneiras de realizar essa operação. Fundamentalmente ela é vista como uma conjunção de proposições, que representam os LOs selecionados, e que possuem uma variável de identificação em comum entre elas. Este exemplo visa conhecer a interseção entre o conjunto de LOs que possuem como autor 'justel' e outro conjunto de LOs que possuem como palavra-chave 'estrutura'. O código lógico para essa consulta é apresentado a seguir.

```
inter(R):-R=..[lo,I,_,_,_,_,justel],R,V=..[lo,I,_,_,estrutura,_,_],V,R.
```

Na outra elaboração, seguindo a mesma idéia vista no exemplo de união, é feita uma consulta que necessita de duas seleções e por fim é feita a interseção dos conjuntos selecionados. Seguindo a mesma consulta passada, apresenta-se o código que manipula a função 'intersection'.

```
lo_justel(V):-R=..[lo,_,_,_,_,_,justel],R,V=R.
lo_pc_estrutura(V):-R=..[lo,_,_,_,estrutura,_,_],R,V=R.
lista_lo_justel(C):-findall(F,lo_justel(F),C).
lista_lo_estrutura(C):-findall(F,lo_pc_estrutura(F),C).
```

Em seguida executa-se a interseção das listas obtidas.

```
execucao_i(I):-
lista_lo_justel(L1),lista_lo_estrutura(L2),intersection(L1,L2,I).

L1 = [lo(3, estrutura_dados, facil, fundamento_comp, disciplina, justel),
lo(4, sistema_computação, dificil, computacao, curso, justel), lo(7,
algoritmos, facil, codigo, topico, justel), lo(8, arvores, dificil,
estrutura, topico, justel), lo(9, grafos, dificil, estrutura, topico,
justel)] ;

L2 = [lo(8, arvores, dificil, estrutura, topico, justel), lo(9, grafos,
dificil, estrutura, topico, justel)] ;
```

```
I = [lo(8, arvores, dificil, estrutura, topico, justel), lo(9, grafos,
dificil, estrutura, topico, justel)] ;
```

5.3.5. NAVEGAÇÃO

A navegação representa uma operação bastante importante do modelo ROSA e permite a travessia do mapa conceitual de LOs através de caminhos estabelecidos por LOs e seus relacionamentos. Assim, consultas em cima de uma determinada navegação são consultas que possuem LOs iniciais como entrada e caminhos a serem seguidos a partir destes, ou que partam de LOs iniciais e que identifiquem os predicados a ele associados, assim como os LOs destino. Para deixar mais claro o que foi explicitado, é mostrado o exemplo da FIG. 5.9.

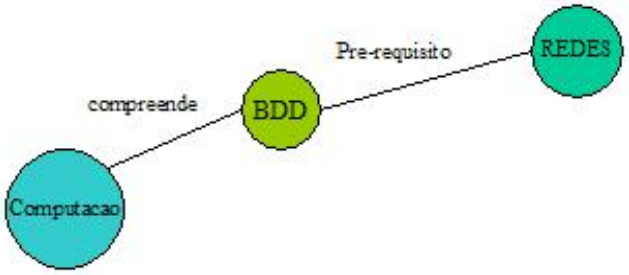


FIG. 5.9 – Sub-conjunto de LOs para processamento navegacional.

No exemplo da FIG. 5.9, pode-se elaborar consultas que desejem obter os LOs que "compreendem" o LO 'computacao', ou ainda os LOs que representam os pré-requisitos do LO 'bdd' e assim por diante. Estes exemplos de consultas seguem a idéia de navegação entre os LOs e seus predicados.

5.3.5.1. RELACIONAMENTO IMPLÍCITO

No modelo é considerado o relacionamento de herança entre LOs. Assim sendo, um LO_1 pode herdar de um LO_2 todos os relacionamentos definidos para o primeiro. O relacionamento de herança e expresso pelo termo "Ehum". Essa relação declara que um determinado LO é do tipo de outro LO. Dessa maneira, um LO filho irá herdar certas propriedades do LO pai. A

idéia de herança nesse caso, só será aplicada aos relacionamentos, ou seja, um LO que é filho de um LO pai, herdará dele os seus relacionamentos, não seus atributos. O exemplo apresentado a seguir demonstra como funciona esse tipo de relacionamento.

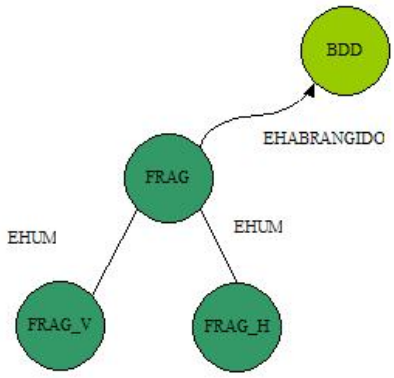


FIG. 5.10 – Herança de LOs

Na FIG. 5.10, devido a existência do relacionamento ‘Ehum’, os LOs que representam a fragmentação horizontal e vertical são especialização do LO que representa a fragmentação. Esse LO pai possui um relacionamento ‘EHABRANGIDO’ com o LO que representa o banco de dados distribuídos. Devido a existência da herança, os outros dois LOs também herdarão esse relacionamento, estabelecendo um relacionamento implícito, ‘EHABRANGIDO’, com o LO BDD. A implementação do comportamento de herança de relacionamento é realizada pela definição de regras. Assim, tal definição estabelece que: Um determinado LO X se relaciona com outro LO Y, através de seu predicado ‘P’, quando ele já possui esse fato declarado explicitamente, ou quando existe um LO Z, com o qual esse LO se relaciona a ‘Y’ por ‘P’, e cujo LO X seja do tipo de Z. Isso pode ser codificado da seguinte maneira.

```
(relacao(X,P,Y));(relacao(Z,P,Y),relacao(X,isa,Z)).
```

Visto isso, observa-se que utilizando conceitos lógicos, se torna possível elaborar consultas interessantes que, utilizando herança, podem trazer resultados inesperados.

5.3.5.2. NAVEGAÇÃO SIMPLES

Nesse sub-item serão apresentadas algumas navegações simples, onde se possui um LO inicial de navegação e um caminho de apenas um predicado, transitivo ou não, que seguirá em

direção ao conjunto de LOs de destino. Nessas navegações aparecerão algumas regras que serão responsáveis por tratar dos relacionamentos que possuem a propriedade de simetria e de inversão. O código para elaboração dessas regras será apresentado, bem como o seu funcionamento. Para ver a forma como essa consultas funcionam na prática será examinado um conjunto mais abrangente de LOs que estarão presentes na próxima FIG. 5.11.

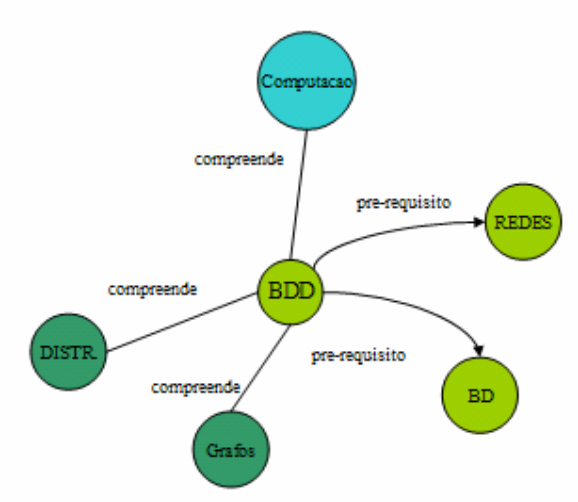


FIG. 5.11 - LOs para processamento navegacional.

Seguindo a mesma idéia apresentada nas consultas até aqui, é exibida a seguir a base de conhecimento que representa a FIG. 5.11. Será considerado os LOs descritos com apenas quatro metadados, para deixar o código um pouco mais reduzido, e a relação entre eles expressa pelas proposições sobre “relacao”.

```
relacao(sistema_computacao,compreende,bdd).
relacao(bdd,compreende,grafos).
relacao(bdd,compreende,distrib).
relacao(bdd,prerequisito,redes).
relacao(bdd,prerequisito,bd).

lo(bd,facil,armazenamento,disciplina).
lo(redes,facil,comunicacao,disciplina).
lo(sistema_computação,dificil,computacao,curso).
lo(distrib,facil,codigo,topico).
lo(bdd,dificil,estrutura,disciplina).
lo(grafos,dificil,estrutura,topico).
```

Quanto aos relacionamentos nota-se, pelo código apresentado, que apenas dois tipos distintos são considerados, pré-requisito e compreende, cujas representações serão exibidas de forma explícita. Um terceiro relacionamento será alcançado através de uma regra mostrada no código a seguir. Esse código permitirá inferir, a partir do relacionamento ‘compreende’, o

relacionamento ‘ehcompreendido’, que segue uma propriedade de simetria em referência ao primeiro. Segue o código lógico que torna isso possível.

```
inverso(ehcompreendido,compreende). %Propriedade Simétrica.
```

Esse fato declara que a relação ‘ehcompreendido’ é o inverso da relação ‘compreende’. Qualquer outra relação que se queira declarar como oposta a outra, fará uso deste tipo de definição.

Tendo apresentado os detalhes de codificação, resta agora apresentar as consultas navegacionais simples, submetidas à base de conhecimento. Inicialmente, suponha que se deseja usar o sistema ROSA para se conhecer melhor uma determinada disciplina. Assim, considerando a disciplina representada através de um LO, deseja-se saber todos os LOs que se associam à mesma, independente do tipo de relacionamento. Na consulta exemplo será usado o LO ‘bdd’. Será inicialmente mostrada a implementação da regra ‘pesq_relacao’, que irá assegurar a pesquisa dos relacionamentos levando em conta a questão da herança e do relacionamento inverso. O código apresentado mostra a pesquisa com atenção na herança.

```
pesq_relac(X,P,Y):-(relacao(X,P,Y));(relacao(Z,P,Y),relacao(X,isa,Z)).
```

Em seguida, a mesma regra é complementada para também assegurar a pesquisa dos relacionamentos inversos que podem surgir. A complementação é vista a seguir.

```
pesq_relac(X,A,Y):-relacao(Y,B,X),inverso(A,B).
```

Para iniciar as consultas dos relacionamentos a partir de um determinado LO é criada uma regra de execução, que irá simplesmente verificar os relacionamentos existentes, utilizando-se da regra ‘pesq_relac’, como pode ser visto pelo código.

```
navegacao(X,S):-pesq_relac(X,_,X1),S=..[lo,X1].
```

Essa consulta possui um parâmetro de entrada que será associado a variável ‘X’ e representará o LO a qual a consulta irá fazer referência. Na variável S será associada a resposta para a consulta que irá varrer toda a base de conhecimento a procura de LOs associados ao LO de entrada por algum caminho. Assim a resposta para a consulta ‘navegacao(bdd,S)’ pode ser vista a seguir:

```
S = lo(grafo) ;
S = lo(distrib) ;
S = lo(sistema_computacao) ;
S = lo(redes) ;
S = lo(bd) ;
```

Existem outros tipos de situações em que se faz necessário explicitar o caminho de navegação, ou seja, dado um LO e a declaração de um predicado, obter os LOs assim associados. Em termos práticos e usando a base anterior, pode-se elaborar consultas como, quais LOs são pré-requisito pra 'bdd'. Nesse caso, o predicado explicitado é o 'pre-requisito' e deixa claro que esse é o caminho que se deseja navegar. O código lógico pra consultas desse tipo é o seguinte:

```
naveg_expl(X,Y,R):-pesq_relac(X,Y,X1),R=..[lo,X1].
```

Nesta consulta, a variável 'X' da mesma forma que na consulta anterior, receberá o LO a ser pesquisado. Porém, agora surge uma outra variável, 'Y', que receberá o predicado sobre o qual a navegação deverá seguir para se chegar aos LOs desejados. Assim, uma consulta que possui a forma, 'naveg_expl(bdd, prerequisito,R)' terá como resposta o conjunto formado por:

```
R = lo(redes) ;
R = lo(bd);
```

Ainda explorando expressões de consulta, pode-se conhecer não só os LOs que possuam algum tipo de associação com outros, mas também metadados que formam esses LOs. Assim, um usuário do ROSA pode querer conhecer quais os níveis de dificuldade que encontrará nos LOs compreendidos por um determinado LO de sua escolha. Será utilizado novamente o LO 'bdd' como exemplo para esse tipo de consulta, conforme o código a seguir:

```
naveg_expl_dif(X,Y,R):-pesq_relac(X,Y,X1),lo(X1,X2,_,_),R=(X1,X2).
```

A execução da consulta é descrita com a especificação da variável X e Y, que receberão o LO e o predicado respectivamente. A variável X1 irá receber a resposta de qual LO compreende e a variável X2 irá receber o grau de dificuldade desse LO. A execução da consulta 'naveg_expl_dif(bdd,compreende,R) terá como reposta:

```
R = grafos, dificil ;
R = distrib, facil ;
```

Da mesma maneira que é possível exprimir a navegação no sentido explícito, também é permitido elaborar de maneira implícita. Em particular, pode-se utilizar expressões de forma inversa ao apresentado pelo modelo, desde que exista uma regra que defina a associação contrária. No caso apresentado, existe o relacionamento 'compreende', que segundo a regra presente no código lógico anterior, é inverso, ou contrário, ao relacionamento 'ehcompreendido'. Dessa maneira, a lógica nos permite exprimir consultas do tipo: quais os LOs que são compreendidos pelo LO 'bdd'? A resposta a consulta é facilmente solucionada, fazendo a inferência da regra implementada, '`inverso(ehcompreendido,compreende)`'. A consulta é elaborada através da clausula 'naveg_expl(bdd,ehcompreendido,R)' já exibida anteriormente e possui como resposta o LO 'sistemas_computacao'.

```
R = lo(sistema_computacao);
```

5.3.5.3. NAVEGAÇÃO COMPLEXA

Até aqui, foram exploradas consultas baseadas em um único predicado, porém nem sempre as consultas navegacionais se apresentam de uma forma tão simples assim. Muitas vezes é necessário a associações de predicados através de operadores lógicos, como o 'ou' e o 'e'. Associações desse tipo deixam as consultas navegacionais ainda bem mais poderosas e com grande expressividade. Nessa seção serão exporadas consultas que apresentam esse tipo de comportamento. Para isso, será apresentado um exemplo estendido ao que foi visto até agora, mas que se encontra um pouco mais complexo, com alguns LOs e relacionamentos adicionais. A FIG. 5.12 mostra o exemplo que será elaborado para as consultas. A mesma também será utilizada nos próximos exemplos que se seguem.

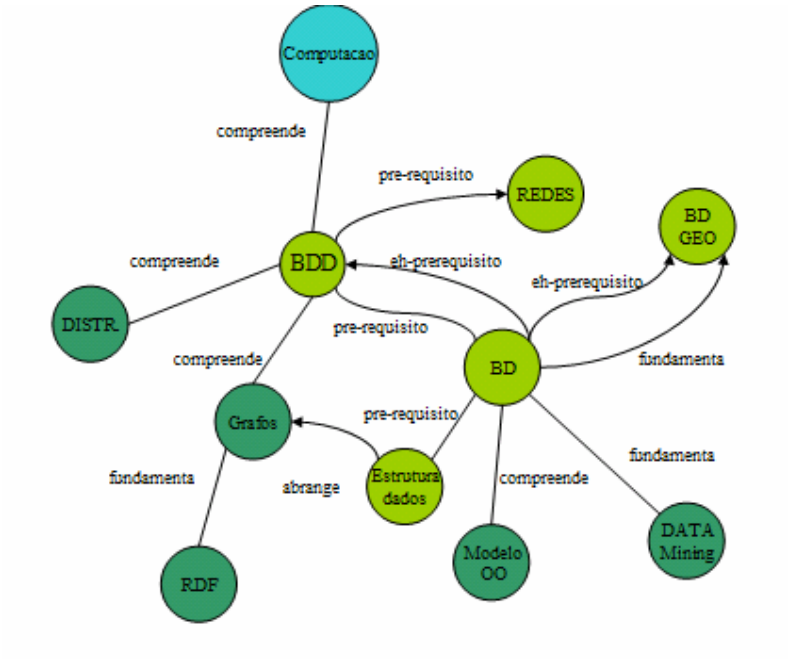


FIG. 5.12 - LOs em um mapa conceitual mais completo

No mapa conceitual da FIG. 5.12 foi adicionado um outro relacionamento inverso, 'eh-prerequisito'. Este funciona de forma oposta ao predicado 'pre-requisito'. O código lógico presente na TAB. 5.2 servirá de base para os exemplos que seguirão.

TAB 5.2 – Novos fatos acrescentados a base de conhecimento

Novos Los	Novos fatos e relacionamentos

`lo(bd_geo,facil,armazenamento,disciplina).`
`lo(modelo_oo,facil,modelos,topico).`
`lo(data_mining,facil,mineracao,topico).`
`lo(estrutura_dados,facil,estrutura,disciplina).`
`lo(rdf,facil,estrutura,topico).`
`lo(grafos,facil,estrutura,topico).`
	`relacao(bd,fundamenta,datamining).`
	`relacao(grafos,fundamenta,rdf).`
	`relacao(bd,fundamenta,bdgeo).`
	`relacao(estrutura_dados,abrange,grafos).`
	`relacao(bd,compreende,modelo_oo).`
	`relacao(bd,ehprerequisito,bdgeo).`
	`relacao(bd,prerequisito,estrutura_dados).`
	`inverso(ehpreresquisito,prerrequisito).`

De posse dessa nova base de conhecimento, podemos elaborar consultas mais complexas. Assim sendo, digamos que um usuário do sistema ROSA pretenda conhecer os LOs que possuam dois tipos de relacionamentos com um determinado LO. Considere, por exemplo, uma consulta que pretende conhecer os LOs associados tanto pelo relacionamento 'ehprerequisito', quanto pelo relacionamento 'fundamenta' ao LO 'bd'. Em um português claro, a consulta deseja encontrar os LOs para os quais o LO 'bd' é "pré-requisito" e que "fundamenta" ao mesmo tempo. Sendo assim, para elaborar essa consulta foi criado o seguinte código lógico bem parecido aos já elaborados.

```
naveg_predicados(X,Y,Z,R):-pesq_relac(X,Y,X1),pesq_relac(X,Z,X1),R=X1.
```

Na expressão acima, às variáveis Y e Z podem-se atribuir os dois relacionamentos associados à X, e à X1. A chamada para a consulta descrita anteriormente é elaborada da seguinte forma 'naveg_predicados(bd,prerequisito,fundamenta,S)'. A variável S receberá uma resposta para a consulta, que no caso é "banco de dados geográficos".

```
S = bd_geo ;
```

Existem outras consultas que se fazem necessárias, em particular, a navegação através dos predicados. Para exemplificar, considere a consulta: "obter os LOs que são fundamentados por aqueles que têm estrutura de dados como pré-requisito". Olhando cuidadosamente o que se deseja, percebe-se que de início é preciso conhecer os LOs que tem 'estrutura_dados' como pré-requisito e em seguida, de posse desse conjunto, encontrar aqueles que são fundamentados por esse conjunto resultante e intermediário.

```
naveg_pred_inter(X,Y,Z,R):-pesq_relac(X,Y,J),pesq_relac(J,Z,X1),R=X1.
```

Inicialmente, a pesquisa que possui "X" como LO inicial e "Y" como relacionamento inicial, irá chegar ao LO de destino e fará a atribuição do mesmo para a variável J. Ela servirá de variável de transição para a outra associação que possui a variável Z representando o relacionamento da segunda associação. No caso apresentado, o conjunto intermediário é formado por apenas um LO, 'bd', e a consulta seguinte que utiliza esse LO intermediário possui como resultado os LOs 'bd_geo' e 'data_mining'. A consulta é chamada através da cláusula: 'naveg_pred_inter(estrutura_dados,ehprerequisito,fundamenta,S).', onde S receberá o resultado final.

```
S = bd_geo ;
S = data_mining ;
```

Finalizando as consultas de navegação, serão explorados operadores lógicos citados anteriormente, que estenderão os predicados de navegação com operadores lógicos como "ou" e "e". Observando a FIG. 5.12, deseja-se conhecer, quais os LOs abrangidos ou fundamentados por aqueles que são abrangidos pelo LO 'estrutura de dados'. Da mesma

maneira que aconteceu com o exemplo anterior, se faz necessário ter em mãos um conjunto intermediário de LOs, e de posse desses LOs a consulta restante avaliará a navegação seguinte. Porém, no caso presente, a navegação seguinte pode ser atendida por dois tipos de relacionamento: 'fundamenta' ou 'abrange', como demonstrada na consulta lógica:

```
naveg_tres_predic(X,Y,Z,W,R):-
pesq_relac(X,Y,J),(pesq_relac(J,Z,X1);pesq_relac(J,W,X1)),R=X1.
```

A consulta possui cinco parâmetros que seguem os padrões anteriormente exemplificados, salvo o terceiro e o quarto parâmetro que representam os caminhos que podem ser seguidos pela consulta intermediária a serem navegados a partir do conjunto de LOs guardados pela variável 'J'. O símbolo ';' representa a idéia da função lógica 'ou'. A chamada à consulta seria expressa conforme a expressão composta de cinco parâmetros: 'naveg_tres_predic(estrutura_dados,abrange,abrange,fundamenta,S)'. Observando-se a FIG. 5.12, o conjunto de LOs intermediário na verdade é formado por apenas um LO, 'grafos', que é o único que "estrutura de dados" "abrange". A partir desse LO a consulta segue através dos relacionamentos "abrange" ou "fundamenta", terceiro e quarto parâmetro, respectivamente. O LO 'grafo' não abrange ninguém, porém ele fundamenta algo, o que também satisfaz a consulta. Assim a resposta para a consulta final vem de "grafos fundamentar RDF".

```
S = rdf;
```

Também é possível realizar seleções em expressões de caminho quando existir a necessidade de conhecer um determinado atributo do LO pesquisado. No exemplo anterior, se for necessário conhecer o nível de dificuldade do LO, a consulta poderia ser reformulada da seguinte forma:

```
naveg_tres_predic_dif(X,Y,Z,W,R):-
pesq_relac(X,Y,J),(pesq_relac(J,Z,X1);pesq_relac(J,W,X1)),lo(X1,R,_,_).
```

Dessa forma, ao se realizar a chamada para a seguinte cláusula: "naveg_tres_predic_dif(estrutura_dados,abrange,abrange,fundamenta,S)", a resposta final será atribuída à variável 'S'.

```
S = facil ;
```

5.3.6. OPERAÇÃO DE FECHO TRANSITIVO

O fecho transitivo representa uma forma poderosa de consulta no modelo ROSA e se aplica à navegação entre LOs e seus relacionamentos. Ela atua sobre os caminhos do modelo e a sua execução é realizada utilizando técnicas de recursividade. Nas linguagens lógicas esse tipo de operação também pode ser expressa.

Para exemplificar um tipo de consulta de fecho transitivo, considere que se deseja conhecer os pré-requisitos de determinada disciplina. A princípio a resposta seria imediata ao se observar o modelo, sem levar em consideração a questão da recursividade, ou seja, uma disciplina pode ter um pré-requisito, que possui outro, que possui um outro e assim por diante.

A mesma FIG. 5.12 será aproveitada para o exemplo de fecho transitivo, por esta se apresentar de uma forma completa, a ponto de poder nos trazer um caso dessa natureza. De posse desse modelo, se faz necessário conhecer todas as disciplinas que são pré-requisito da disciplina banco de dados distribuídos, 'bdd'. A princípio, pensando de uma forma não recursiva, o seguinte código lógico poderia ser elaborado:

```
conhecer_prereq(X,S):-relacao(X,prerequisito,Y),S=Y.
```

A execução da cláusula 'conhecer_prereq(bdd,X)', recupera os pré-requisitos diretos de banco de dados distribuídos, porém ainda existem pré-requisitos seguintes a esses LOs encontrados e que também precisam ser declarados.

```
X = redes ;
X = bd ;
```

Para corrigir esse problema, a solução é elaborar consultas recursivas, capazes de explorar todos os relacionamentos até a última profundidade. Assim sendo, uma melhor elaboração da consulta exemplificada anteriormente começa com a criação de regras que assegurarão a realização de uma pesquisa transitiva usando recursão. Tal regra recursiva é descrita a seguir pelo complemento da regra do 'pesq_relac'.

```
pesq_relac(X,A,Y):-relacao(X,A,Z),pesq_relac(Z,A,Y),transitivo(A).
```

Depois de elaborada essa regra, um fato deve estar presente na base de conhecimento, para determinar os relacionamentos transitivos. Para tanto, basta o fato 'transitivo(prerequisito)' seja acrescentado. Depois dele presente na base, o relacionamento 'prerequisito' já é entendido como sendo de transitividade. Assim, a consulta pode ser realizada atingindo toda a profundidade, obtendo um resultado mais completo.

```
X = redes ;
X = bd ;
X = estrutura_dados ;
```

Vale lembrar novamente que é possível fazer a mistura de operações algébricas e de seleção em um consulta como essa, a exemplo dos os níveis de dificuldade e autores desses LOs.

5.3.7. Operação de Seleção de Predicados

Em todas as consultas aqui apresentadas, a idéia era que fossem recuperados LOs ou os seus metadados. Todavia, muitas vezes se faz necessário no modelo ROSA, elaborar consultas que tornem possível conhecer os predicados que envolvem LOs que se comportam ora como sujeito, ora como objeto. Também é necessário se conhecer os predicados que envolvam um determinado LO, sendo ele sujeito ou objeto, ou ambos. Para se entender a seleção de predicados, é necessário observar mais uma vez a FIG. 5.12. Nela pode-se interrogar quais os relacionamentos que envolvem os LOs 'bd' e 'bd_geo'. Essa consulta teria como resposta os predicados 'eh-prerequisito' e o predicado 'fundamenta'. Esse tipo de consulta pode ser implementado na linguagem lógica da seguinte maneira:

```
sele_pred(X,Y,R):-pesq_relac(X,R,Y).
```

As variáveis X e Y irão receber os LOs sujeito e objeto respectivamente. Como resposta, a variável R será retornada como a relação que existe entre esses dois LOs. A chamada 'sele_pred(bd,bd_geo,S)', terá a resposta aguardada.

```
S = ehprerequisito ;
S = fundamenta ;
```

Uma consulta bem mais poderosa poderia solicitar que sejam obtidos todos os relacionamentos que envolvem um determinado LO como sujeito, ou como objeto, como descrito anteriormente. Esse tipo de consulta pode ajudar bastante a usuários do sistema ROSA a terem acesso a várias informações que dizem respeito a predicados de um LO. Seguindo essa nova consulta, digamos que já se conheça que o LO 'bd' se relaciona ao LO 'bdd' através dos relacionamentos acima, porém se queira conhecer quais os outros relacionamentos existentes nesse mesmo LO, na qual ele atue como sujeito. O código para essa consulta seria simplesmente 'sele_pred (bd,Y,R)'. Nele as variáveis Y e R contém o LO objeto e o relacionamento respectivamente.

```
Y = datamining
R = fundamenta ;

Y = bdgeo
R = fundamenta ;

Y = modelo_oo
R = compreende ;

Y = estrutura_dados
R = prerequisito ;

Y = bdd
R = ehprerequisito ;

Y = bdgeo
R = ehprerequisito ;
```

Dessa mesma maneira é possível conhecer todos os relacionamentos que um LO possui no papel de objeto, bastando para isso realizar uma pequena alteração no código acima trocando apenas o local da variável X, como segue a cláusula 'sele_pred (X,redes,R)'. Nesse exemplo, 'redes' comporta-se como o objeto da relação e a consulta tem como resposta:

```
X = bdd
R = prerequisito
```

5.4. CONSIDERAÇÕES FINAIS

Como visto através dos exemplos apresentados, é perfeitamente possível realizar a representação de LOs e relacionamentos do modelo ROSA utilizando uma linguagem lógica, bem como através da elaboração desse modelo, implementar qualquer tipo de consulta prevista pela álgebra ROSA. Com isso, foi possível realizar o aperfeiçoamento de consultas

para esse modelo. Um aperfeiçoamento claro foi a elaboração de consultas que, utilizando regras de inferência, a exemplo da regra de herança, possibilita alcançar resultados inesperados em consulta. Com o auxílio da lógica, o caminho ficou aberto para o aperfeiçoamento do modelo existente através de uma extensão expressa em linguagem lógica, possibilitando o surgimento do novo modelo ROSAI.

A FIG. 5.13 a seguir apresenta a arquitetura do modelo ROSAI. No próximo capítulo será apresentada toda a especificação técnica do ROSAI, que possibilitou a validação de toda a representação lógica apresentada nesse capítulo.

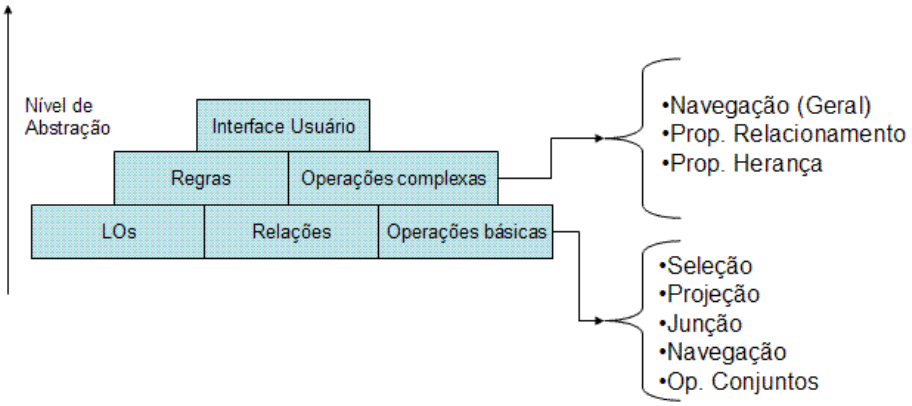


FIG. 5.13 – Arquitetura do modelo ROSAI

6. IMPLEMENTAÇÃO DO SISTEMA ROSAI

Nesse capítulo será discutida a especificação técnica da implementação adotada para a extensão do modelo de dados ROSA. O sistema foi batizado de ROSAI, por se tratar da expansão da proposta do modelo ROSA, acrescido do poder de inferência em suas consultas. Será detalhada a implementação, indo desde a elaboração da arquitetura, até o seu funcionamento.

6.1. ARQUITETURA DO ROSAI

A arquitetura apresentada aqui na FIG. 6.1 é constituída por três camadas. Essas camadas trabalham em harmonia uma com a outra, constituindo um conjunto que faz com que o sistema desempenhe o seu papel.

A primeira camada, encontrada na base da arquitetura do ROSAI, é a camada de persistência. Ela é responsável por manter guardados os LOs que serão usados nas consultas pelo ROSAI.

No topo da arquitetura encontra-se a camada de interface. Essa camada é a que permite a apresentação do sistema ao usuário final, permitindo que o mesmo interaja com o ROSAI utilizando-se de uma interface baseada em formulários. Através dessa camada, torna-se possível a execução de consultas que exprimem navegação ou que acessam metadados de um LO, sem que o usuário conheça detalhes da sua implementação lógica.

A camada intermediária refere-se à máquina de execução lógica. Ela é responsável por executar as consultas vindas do usuário, através da sua interface. É nessa camada que as regras e os fatos são interpretados e as respostas à questões elaboradas são produzidas.

As camadas descritas podem ser vistas na FIG. 6.1.

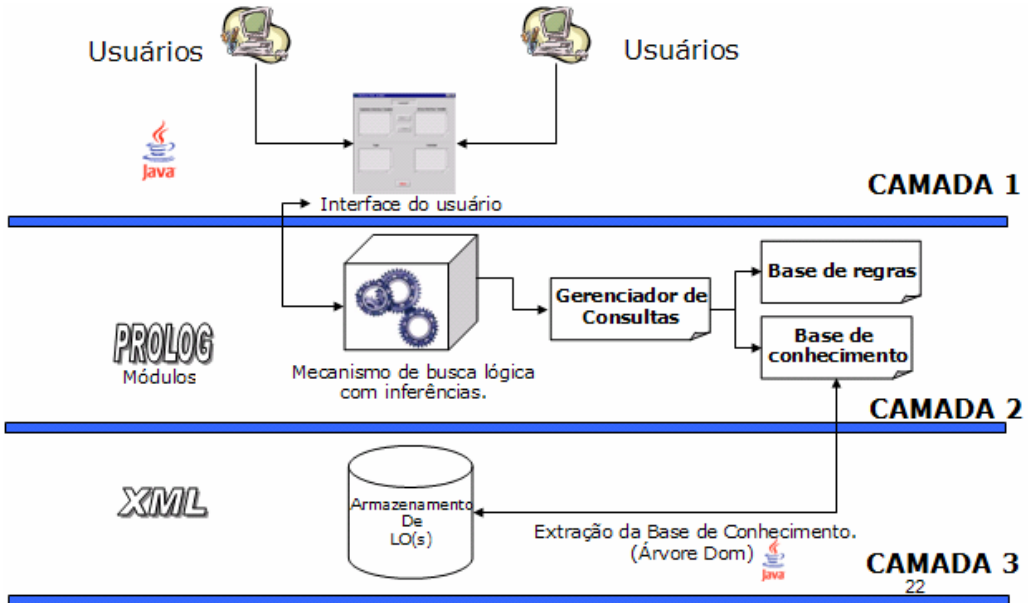


FIG. 6.1 – Arquitetura do sistema ROSAI

Pela figura é visto que na camada de persistência existe um documento XML [XML, 2005]. Esse documento possui a especificação de todos os LOs e relacionamentos presentes na base. Desse documento são extraídos todos os fatos que representam os LOs e os relacionamentos através de uma aplicação que será descrita nas seções seguintes, e escritos em um documento lógico na camada acima, seguindo o modelo de representação apresentado no capítulo 5. Esses fatos comporão a base de conhecimento do ROSAI.

Nessa mesma camada estarão presentes as regras do ROSAI que farão parte da base de regras. Em qualquer momento a base de regras poderá ser acrescida de novas regras, na medida em que o usuário sinta essa necessidade. Essas duas bases citadas irão trabalhar em conjunto para resolver as consultas solicitadas.

Para que essas duas bases trabalhem em cooperação, se faz necessário a existência de um outro arquivo lógico, capaz de atender as consultas solicitadas pela máquina de inferência. Esse arquivo, chamado de gerenciador de consultas, realiza a mediação entre as duas bases. Para ele as duas bases são vistas como módulos do sistema, acessando-as quando necessário.

Uma vez apresentadas as camadas da arquitetura, o item seguinte trata de aspectos associados às linguagens de programação, bem como das bibliotecas utilizadas na implementação da arquitetura proposta.

A principal linguagem de programação utilizada nesta implementação foi a linguagem Java [JAVA, 2005], em sua versão 1.4.2. A linguagem Java permitiu implementar: as rotinas de extração dos fatos do documento XML; as rotinas de manipulação do usuário final do ROSAI via Web; bem como a execução da consultas através da linguagem lógica Prolog. Para a rotina de extração dos fatos do documento XML se fez necessário a utilização de uma biblioteca já presente no pacote Java, que será descrita mais adiante. Esse pacote permitiu o entendimento de todas as partes que compunham o documento XML, bem como a nevegação através da estrutura do documento.

Existem vários interpretadores lógicos para o Prolog, porém para a implementação aqui citada foi utilizado o SWI-Prolog [SWI-PROLOG, 2004]. Este foi escolhido por se tratar de um software livre, que possui várias bibliotecas responsáveis por atuar em vários ramos da ciência da computação. Entre esses pacotes, um foi desenvolvido para funcionar como uma interface para a linguagem Java. Esse pacote será detalhado mais a seguir, e funcionou de ponte entre as requisições vindas da camada de interface do usuário para a camada de interpretação lógica.

O framework utilizado para realizar a integração de diferentes tipos de aplicações foi o Eclipse, em sua versão 3.0. O eclipse pode ser utilizado para integrar diferentes tipos de aplicações, porém ele possui como padrão as aplicações JDT (Java Development Tooling), que já vem com o Eclipse. No desenvolvimento da aplicação, ele serviu de apoio para a implementação na linguagem Java, bem como a implementação de consultas lógicas feitas no gerenciador de consultas. O eclipse possui um eficiente mecanismo de gerenciamento dos arquivos implementados. Nesse gerenciamento, os fontes são armazenados em uma workspace, que é uma pasta especial localizada no sistema de arquivos. Através da workspace, todos as aplicações instaladas comunicam-se entre si, e com isso, quando uma aplicação altera um recurso qualquer, todas as outras aplicações instaladas são notificadas sobre a mudança, garantindo com isso uma consistência e integração de todo o ambiente de desenvolvimento. Maiores detalhes de como foi organizada o workspace de desenvolvimento do ROSAI será exibido nas seções seguintes.

Como exemplo de bibliotecas que auxiliaram na implementação do ROSAI, podemos citar algumas que estão presentes no próprio pacote padrão do Java. O primeiro pacote utilizado foi a “javax.xml.parsers”, onde basicamente foram utilizados as classes

'DocumentBuilder' e 'DocumentBuilderFactory'. Essas classes serviram para criar um *Parser*, capaz de transformar um documento XML em um documento DOM (Document Object Model) [DOM, 2004]. Essa operação foi utilizada na implementação da extração de um documento XML, para um arquivo lógico final.

Na manipulação dos objetos presentes no documento DOM criado, foi utilizado um outro pacote padrão do Java, o "org.w3c.dom". Nesse pacote foi utilizada a interface 'Document', que conceitualmente é a raiz do documento árvore gerado. Permite o acesso inicial aos dados do documento, viabilizando o inicio das manipulações sobre o documento. Nesse mesmo pacote, uma outra interface que foi bastante útil nesse processo de transformação foi a interface 'Node', que é a representação de um simples nó em uma representação de árvore. Através dessa interface é possível obter informações importantes através de certos métodos, como o nome de um determinado nó, 'nodeName', o seu valor, 'nodeValue' e atributos, 'attibutes'. A especificação dessa interface contém mecanismos convenientes para se obter informações relevantes a respeito de um determinado nó na árvore DOM com que se está trabalhando. Outra interface importante nesse mesmo pacote foi a que permitiu o conhecimento de todos os filhos de um determinado nó na árvore. A interface 'nodeList' cria uma abstração para uma coleção ordenada de nós. Essa interface, através do método 'getChildNodes', tornou possível obter todos os filhos de um determinado nó que podiam ser acessados, obedecendo um índice ordenado, iniciado por 0. Com isso, operações utilizando uma estrutura de repetição que percorresse todos esses índices, seria capaz de analisar um por um, todos os nós filhos de um dado nó pai.

Por último, um importante pacote utilizado nessa implementação foi a biblioteca JPL [JPL, 2005]. Ela trata de um conjunto de classes Java e funções escritas em C que criam uma interface entre Java e o Prolog. O JPL usa a interface nativa do Java para se conectar ao mecanismo Prolog através de sua interface. O JPL não é uma implementação Java pura do Prolog. Ele faz uso da implementação nativa do Prolog que da suporte à plataforma. A versão corrente do JPL utilizada no ROSAI foi implementada exclusivamente para o Prolog, o SWI-Prolog. Com o pacote JPL foi possível fazer com que a implementação em Java acessasse qualquer biblioteca padrão do Prolog, predicados e qualquer outro recurso sem necessitar de qualquer tipo de *wrapper* ou metadado. O pacote permitiu, portanto, que fosse desenvolvida uma aplicação híbrida, capaz de usufruir o melhor que cada tipo de linguagem tem para oferecer. No item seguinte será mostrado com maiores detalhes como foi desenvolvida a extração do documento XML para o documento lógico.

Para entender a tradução do documento XML responsável por armazenar todos os LOs, se faz necessário entender inicialmente a sua estrutura. A estrutura desse documento possui uma série de metadados padronizados que podem ser vistos através da referência [IEEE, 2002]. No caso dessa dissertação foi utilizado um subconjunto desse padrão que possui uma representação mais reduzida, mas que não deixa de validar a nossa proposta.

Com isso o documento será constituído de uma *tag* inicial chamada de LOs, que irá possuir o conjunto de todos os LOMs (Learning Object Metadata) que formam o documento. Cada LOM é constituído de 3 partes: uma composta por um atributo único e responsável pela identificação única do LOM; uma segunda que corresponde à descrição dos atributos do LOM, sendo formada pelo sub-conjunto de atributos: Título, Linguagem, Palavra Chave, Nível de Dificuldade, Nível de Agregação e Descrição do LOM; e a última parte, onde encontra-se a descrição de todas as associações existentes entre os LOMs. Essa parte é formada por um conjunto de predicados, já que um LOM pode se relacionar a vários outros LOMs, e cada predicado possui 2 *tags*, uma que dita o nome do relacionamento e outra que lista a identificação de todos os LOMs que participam do relacionamento como objeto. Para exemplificar, a FIG. 6.2 e 6.3 mostram respectivamente a representação em forma de um mapa conceitual de uma estrutura ROSA e a representação no formato de um documento XML.

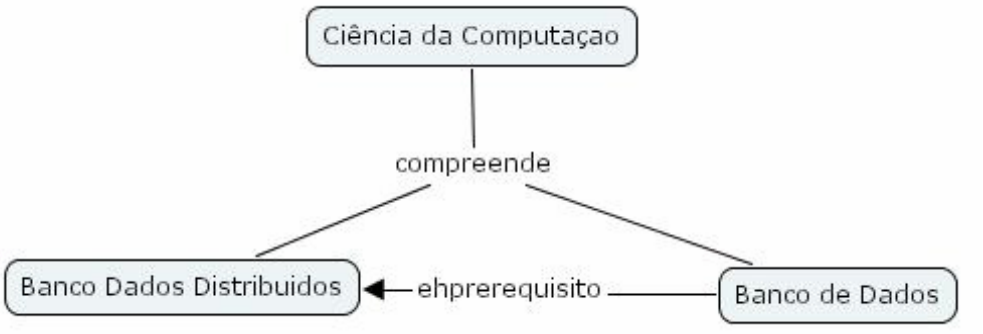


FIG. 6.2 – Mapa conceitual reduzido do curso de computação

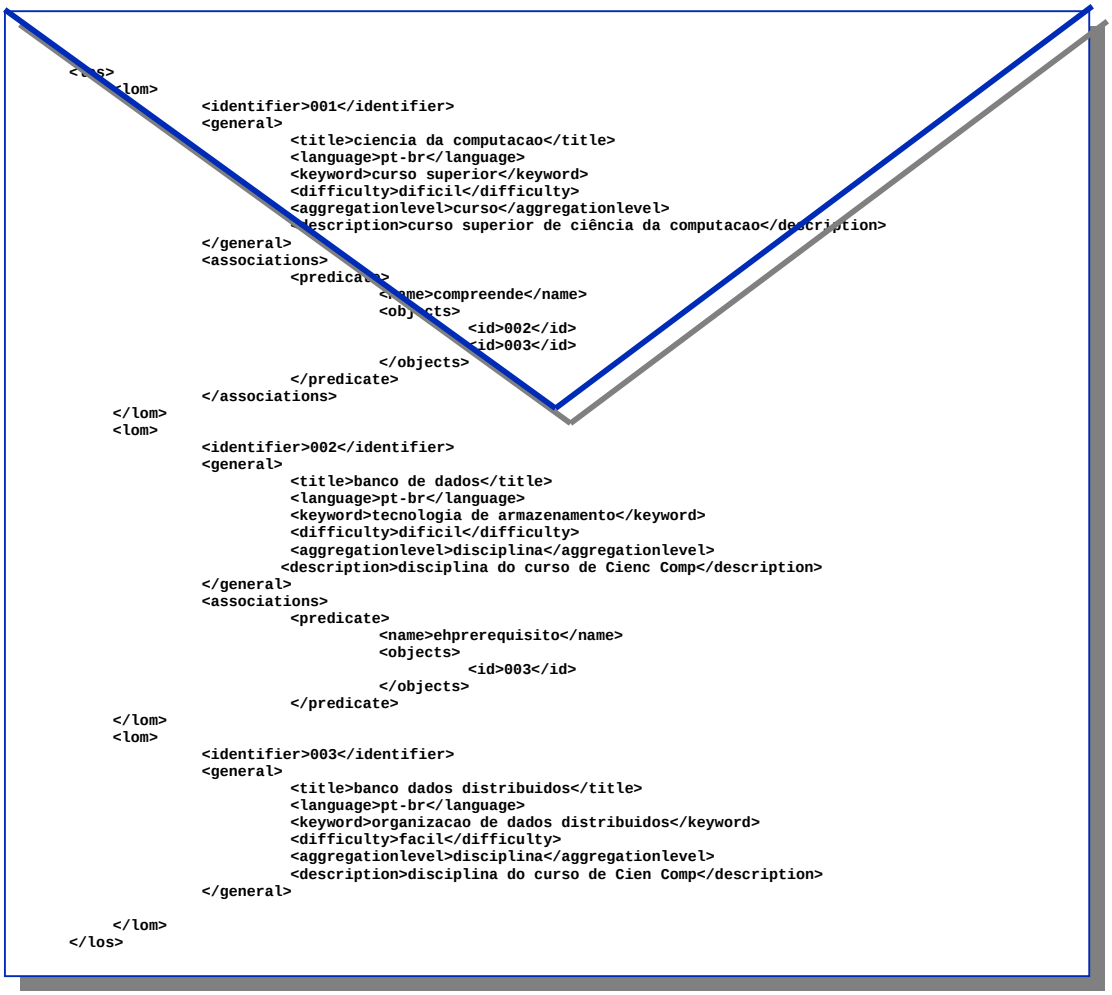


FIG. 6.3 – Representação em XML

Dessa maneira, a FIG. 6.3 mostra a especificação do código que representa o mapa conceitual do modelo ROSA através de um arquivo XML. De posse de um documento nesse formato é que foi possível realizar toda a operação de extração do documento lógico final.

Primeiramente o documento XML foi transformado em um documento DOM, através do conjunto de classes Java mencionadas anteriormente. Esse documento DOM criado pelo *Parser* pode ser visto como uma árvore, conforme ilustrado na FIG. 6.4.

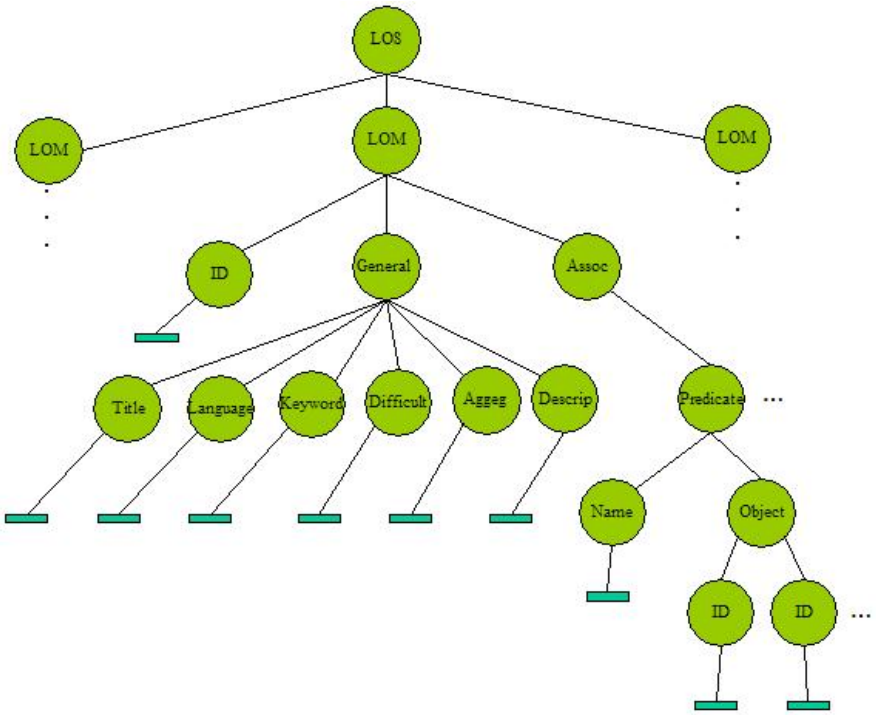
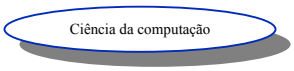
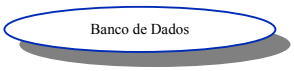
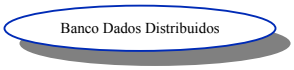


FIG. 6.4. Árvore DOM do documento XML

Uma vez criada em memória, a árvore DOM é armazenada em uma estrutura de dados do tipo *HashMap*. Essa estrutura organiza todo os LOMs presentes na árvore para que em seguida possam ser processados. Nesse processamento, a chave de acesso aos LOs será o nome do nó e seus metadados.

O próximo passo a ser seguido é varrer a estrutura HashMap criada, e a partir daí gerar os fatos lógicos. A cada LOM analisado novos fatos são gerados e os mesmos são armazenados em um vetor de fatos. A TAB. 6.1 mostra os passos seguidos nessa varredura.

TAB. 6.1 - Criação do Vetor de Fatos através dos LOMs na HashMap

Iteração	LOM	Vetor de Fatos
1		[lo (ciencia da computação,pt-br,curso superior,difícil,curso,curso superior em C. Comp), relacao(ciencia da computacao,compreende,banco dados), relação (ciência da computação,compreende,banco dados distribuídos)]
2		[lo (ciencia da computação,pt-br,curso superior,difícil,curso,curso superior em C. Comp), relacao(ciencia da computacao,compreende,banco dados), relação (ciencia da computação,compreende,banco dados distribuídos), lo (banco de dados,pt-br,tecnologia de armazenamento,difícil,disciplina,disciplina do curso de C Comp), relacao (banco de dados, ehprerequisito, banco dados distribuídos)]
3		[lo (ciencia da computação,pt-br,curso superior,difícil,curso,curso superior em C. Comp), relacao(ciencia da computacao,compreende,banco dados), relação (ciencia da computação,compreende,banco dados distribuídos),lo (banco de dados,pt-br,tecnologia de armazenamento,difícil,disciplina,disciplina do curso de C Comp), relacao (banco de dados, ehprerequisito, banco dados distribuídos), lo (banco dados distribuidos,pt-br,organizacao de dados,difícil,disciplina,disciplina do curso de C Comp)].

A base de conhecimento deverá ser um arquivo lógico que interpretado pelo Prolog. Assim, para criar esse arquivo lógico, só resta varrer o vetor de fatos criado, capturando cada membro do mesmo e escrevendo-os em linhas diferentes em um arquivo. O formato do arquivo final pode ser visto pela FIG. 6.5.

```
lo (ciencia da computação,pt-br,curso superior,difícil,curso,curso superior em C. Comp).
relacao (ciencia da computacao,compreende,banco dados).
relacao (ciencia da computação,compreende,banco dados distribuídos).
lo (banco de dados,pt-br,tecnologia de armazenamento,difícil,disciplina,disciplina do curso de C Comp).
relacao (banco de dados, ehprerequisito, banco dados distribuídos).
lo (banco dados distribuidos,pt-br,organizacao de dados,difícil,disciplina,disciplina do curso de C Comp).
```

FIG. 6.5 – Base de Conhecimento Final

A próxima seção discutirá de que maneira esse arquivo gerado trabalha em conjunto com mais outros dois arquivos lógicos: a base de regras e o gerenciador de consultas.

6.4. OS ARQUIVOS LÓGICOS DO ROSAI

Nessa seção será discutida a integração dos três arquivos lógicos presentes na implementação do ROSAI. Como visto na seção anterior, um desses arquivos é a base de conhecimento, cujo formato foi apresentado tanto na seção passada, quanto no capítulo 5.

Maior atenção será dada à base de regras, por se tratar do arquivo lógico responsável por armazenar as regras definidas no sistema. Nessa implementação são fornecidas algumas regras básicas aplicadas a relacionamentos, tais como transitividade, simetria e inversão que, conforme mostrado no capítulo 5, asseguram a herança de relacionamentos entre LOs. Adicionalmente às regras pré-estabelecidas, usuários mais especialistas podem formular suas próprias regras e armazená-las nesse arquivo.

O terceiro arquivo presente é o gerenciador de consulta (GC). É dele a responsabilidade de receber as consultas dos usuários em um formato lógico, já transformado, e avaliá-los em cima da base de fatos e regras. O GC irá observar as duas bases e inferir conclusões lógicas sobre o que não atende e sobre o que atende ao que lhe foi solicitado. É no GC onde se encontra toda a implementação lógica das consultas com base nas operações algébricas, tais como seleções, projeções, e navegações.

Para o GC, a base de conhecimento e a base de regras são entendidas como módulos que podem ser acessados através de uma implementação.

Uma vez apresentada a funcionalidade básica de cada um desses arquivos no ROSAI, será discutido um pequeno exemplo do funcionamento dos mesmos, mostrando o papel de cada um nessa operação. Será utilizado como exemplo uma operação de seleção sobre uma base de conhecimento, gerada a partir do processo de transformação apresentado na seção anterior, sobre o mapa conceitual da FIG. 6.6.

FIG. 6.6 – Mapa conceitual Reduzido com herança e transitividade

No mapa conceitual da FIG. 6.6 apresentam-se quatro LOs e dois tipos de relacionamentos. O primeiro desses relacionamentos representa a herança do LO “Mestrado no DE9/IME”, em relação a “Mestrado Computacao”, enquanto o segundo representa o relacionamento transitivo ‘compreende’. Em cima desse mapa conceitual é formulado a consulta que objetiva conhecer os LOs compreendidos pelo LO ‘Mestrado no DE9/IME’. Essa consulta é submetida ao GC que se responsabilizará por obter a resposta.

Para tanto, no GC deverá estar presente um código que permita executar a consultas de navegação, assim como acessar os outros dois arquivos lógicos que o auxiliam. A especificação resumida da implementação lógica do GC com apenas essa consulta exemplo pode ser vista a seguir:

```
:-use_module(knowbase).           %/acesso a base de conhecimento
:-use_module(rulebase).           %/acesso a base de fatos
navegacaopred(X,PRED,RESP):-pesq_relac(X,PRED,X1),RESP=..[lo,X1],RESP.
```

Nesse código, as primeiras duas linhas fazem a ligação entre os dois módulos anteriormente descritos. A função 'navegacaopred' que possui três parâmetros: o primeiro recebe o LO a partir do qual se deseja realizar a navegação, o segundo recebe o predicado a ser navegado, e o último a resposta final da consulta.

Dessa forma, utilizando esse código, a consulta lógica responsável por fazer a verificação nas bases seria: 'navegacaopred (mestrado no de9/ime, compreende, RESP), onde a variável X recebe '*mestrado no de9/ime'* e PRED recebe *compreende*.

Se nessa consulta o GC utilizar apenas a base de conhecimento, a resposta retornará um conjunto vazio, já que a base de conhecimento só guarda fatos explícitos, não existindo nenhum fato afirmando que o LO citado compreenda alguma coisa. Porém, como o GC também utiliza a base de regras, ele poderá inferir outros fatos implícitos, gerados através de herança, transitividade, etc.

O exemplo permite ainda que sejam explorados dois aspectos importantes introduzidos por regras associadas a relacionamentos: herança e transitividade. O código a seguir define as regras necessárias.

```
pesq_relac(X,P,Y):-(relacao(X,P,Y));(relacao(Z,P,Y),relacao(X,isa,Z)).%//heranca
pesq_relac(X,P,Y):-relacao(X,P,Z),pesq_relac(Z,P,Y),transitivo(P).%//transitividade
```

A primeira linha especifica a relação de herança e a segunda de transitividade. Nesse código, a segunda linha exige que o predicado envolvido na pesquisa possua um fato explícito que diga que ele é de transitividade. Assim o usuário do ROSAI deverá acrescentar esse fato na base de fatos para que a consulta funcione perfeitamente, já que na extração do arquivo XML, o arquivo lógico resultante não possui ainda essa informação de quais relacionamentos são transitivos, ou que possuem algum outro tipo de característica como a inversão com outro relacionamento ou se o mesmo é simétrico. Assim o seguinte fato a respeito do

relacionamento ‘compreende’ deverá ser acrescentado pelo usuário do ROSAI, através de sua interface, na base de fato: ‘transitivo (compreende)’.

A aplicação da consulta, a forma como ela foi executada e as respostas obtidas sobre o mapa conceitual da FIG. 6.6, podem ser vistos através da FIG. 6.7 logo a seguir.

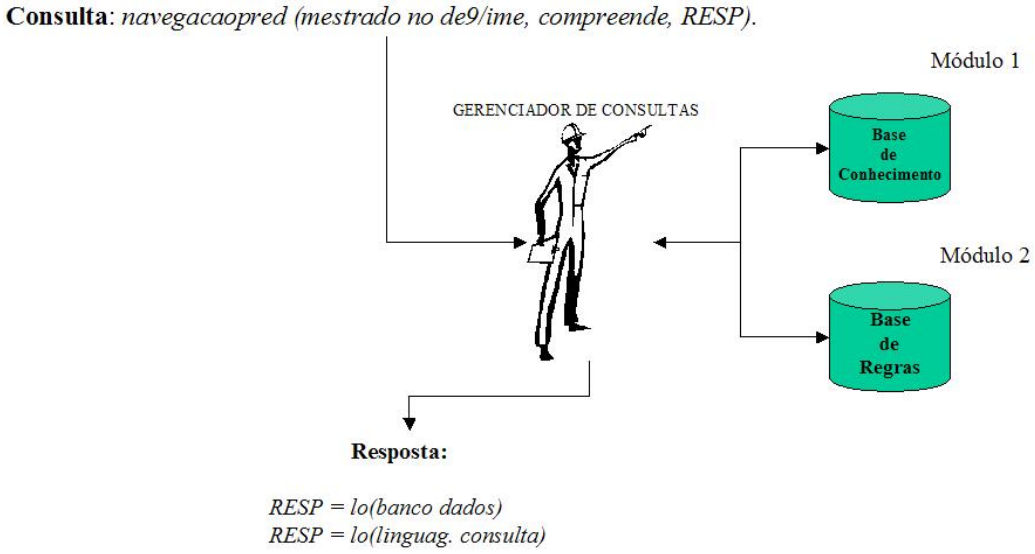


FIG. 6.7 – Atuação do gerenciador de consultas

Nessa seção foi discutido o funcionamento dos arquivos lógico do ROSAI. No próximo item será apresentado o processo de integração entre a interface java e a camada de inferência Prolog, considerando a diferença de paradigma entre as duas linguagens.

6.5. A INTERFACE JAVA - PROLOG

Nesse item será feita uma rápida explicação de como atua o funcionamento dessas duas interfaces no ROSAI. Para tanto será exibido um exemplo baseado numa navegação simples, a exemplo do que foi visto na seção anterior.

Inicialmente será mostrado como tudo começa, através de uma pequena tela existente no ROSAI, onde o usuário pode iniciar a sua solicitação de navegação. Maiores detalhes sobre as interfaces de acesso ao sistema, estão presentes no final dessa dissertação (Ver Apêndice 1). A tela exemplo a ser usada para iniciar todo o processo pode ser vista na FIG. 6.8.

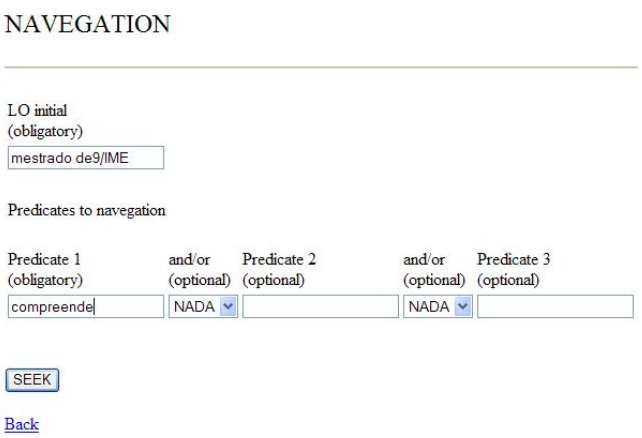


FIG. 6.8 – Layout de navegação

Nessa tela o usuário irá escolher por qual LO será iniciada a navegação e através de qual predicado. No caso, a navegação escolhida é a mesma vista no item anterior. Todo o processo a ser seguido por essa consulta de navegação após acionado o botão SEEK, pode ser visto na FIG. 6.9.

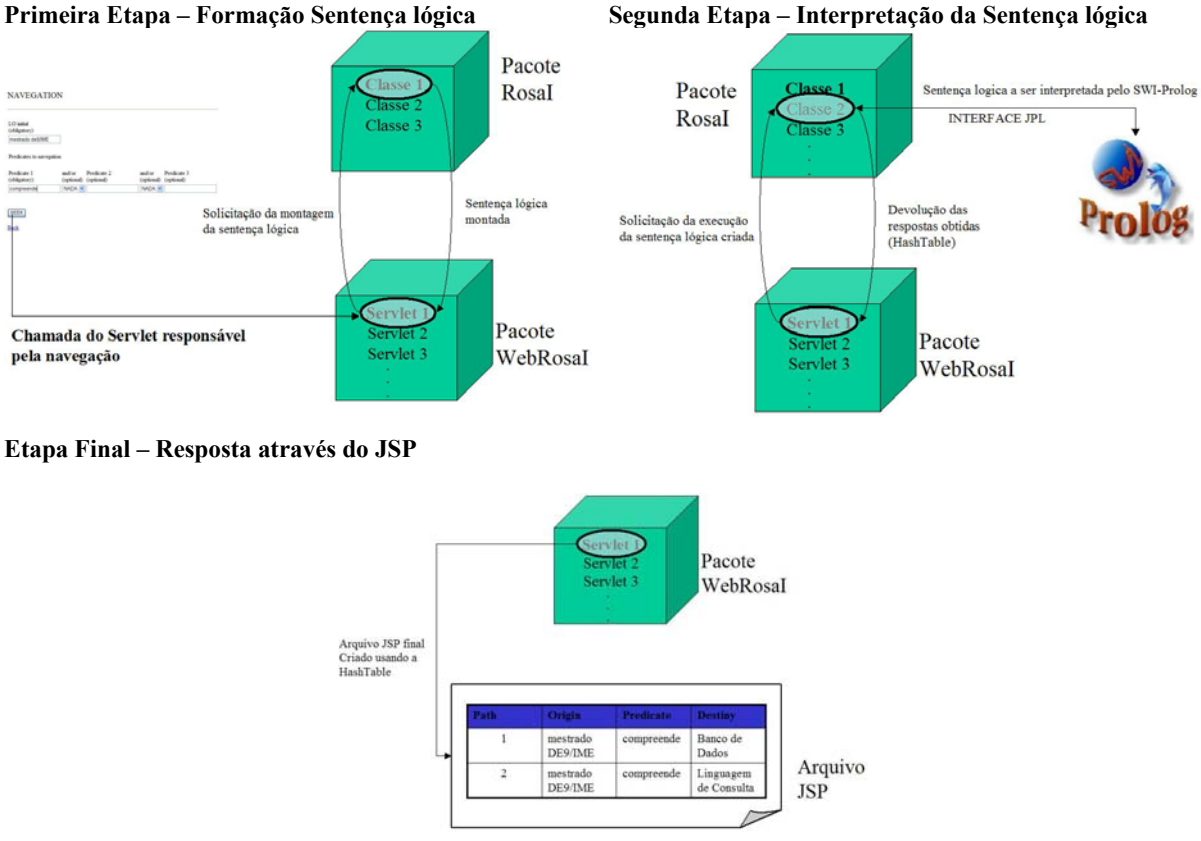


FIG. 6.9 – Operação de execução de uma consulta

Pela FIG. 6.9 o primeiro passo a ser seguido depois de composta a consulta é passá-la a uma classe *Servlet* [SERVLET, 2005] Java, pertencente ao pacote de um projeto chamado WebRosaI. Esta última chamará um método de uma classe pertencente a um pacote de um outro projeto, o RosaI, responsável por montar a sentença lógica para uma consulta de navegação com apenas um predicado. Maiores detalhes sobre esses projetos e seus pacotes serão dados na próxima seção, quando for exibida a estrutura desse sistema na *workspace* do Eclipse. O método de classe *criar()* irá produzir a sentença lógica a ser submetida ao GC, dando inicio ao processo de avaliação já discutido. Depois de instanciado o objeto da classe, tendo como parâmetros os predicados escolhidos pelo usuário e os dois termos (and/or) do formulário, o método identifica quais parâmetros estão vazios e quais estão preenchidos, determinando o tipo de navegação, se com um, dois ou três predicados. Feita essa análise, o método do objeto devolve ao *Servlet* a sentença lógica que ele criou a partir dos parâmetros a ele entregues.

Com essa sentença, o *Servlet* prossegue a sua operação solicitando mais uma vez a ajuda de uma outra classe do projeto RosaI. Dessa vez ele irá solicitar a essa classe as respostas da consulta lógica criada na etapa anterior. Sendo assim, a classe chamada terá a função de executar a interpretação lógica da sua consulta. Para tanto, o *Servlet* entrega a essa classe dois parâmetros: o *path,* onde se encontra o arquivo lógico responsável pela interpretação da consulta, no caso, o arquivo onde se encontra o GC; a sentença lógica entregue a este pela operação anterior. Nessa classe chamada ocorre a comunicação entre as duas linguagens. É nela que está presente a biblioteca JPL, e será utilizada para tornar isso possível. A sua invocação é feita a partir da criação do objeto, através dos dois parâmetros passados pelo *Servlet*. Nela existe um método, chamado *executar()*, que irá interpretar a consulta no arquivo indicado, através do interpretador SWI-Prolog, seguindo o processo mostrado no item anterior, de integração dos arquivos lógicos. As respostas obtidas pelo SWI-Prolog serão devolvidas à Classe Java executora, que então guarda essas respostas em uma estrutura do Java chamada *HashTable*, permitindo acesso posterior.

A *HashTable* obtida é entregue então ao Servlet solicitante, que exibe as respostas obtidas em um arquivo JSP (Java Server Page) final, conforme mostrado na FIG. 6.9, manipulando toda a HashTable da maneira que melhor achar necessário.

Assim, o processo de execução de consultas no ROSAI, envolvendo as três camadas da arquitetura foi apresentado. No exemplo mostrado, a consulta era responsável por uma

operação de navegação, com apenas um predicado. Porém consultas envolvendo mais predicados ou outras operações, como seleção e projeção, seguem o mesmo processo, acessando as classes responsáveis por sua execução específica.

6.6. ORGANIZAÇÃO DA WORKSPACE NO ECLIPSE

A workspace do Eclipse é uma pasta localizada no sistema de arquivo que organiza os arquivos fonte do sistema implementado. É através da workspace que as aplicações conseguem se enxergar e criar um ambiente de desenvolvimento integrado e sempre consistente.

No desenvolvimento do ROSAI, a workspace foi criada contendo dois conjuntos de pacotes já apresentados em seções anteriores: RosaI, e WebRosaI. Esses dois projetos são dependentes um do outro, sendo que o principal deles é o RosaI, funcionando como o núcleo do sistema. É nele que estão presentes todos os procedimentos para as principais operações. Dessa maneira, o WebRosaI funciona apenas como uma ponte de acesso entre a camada de interface do usuário e o funcionamento efetivo das operações, realizando chamadas ao projeto RosaI sempre que assim precisar.

Da maneira que foi elaborada essa implementação, outros tipos de interface podem ser elaborados utilizando Java. Essas interfaces podem ser elaboradas para o ROSAI utilizando-se de outras tecnologias Java, como Swing e SWT, reaproveitando o projeto núcleo responsável por todas as operações, o RosaI. Nesse caso é necessário apenas que essas interfaces façam acesso à método de classes, presentes no pacote RosaI da mesma maneira como que é feito atualmente pelo WebRosaI.

Nas seções seguintes serão apresentados, a maneira como estão organizados os dois projetos implementados aqui, exibindo todos os pacotes, bem como quais classes compõem esses pacotes, e a função de cada uma dessas classes.

6.6.1. PROJETO WEBROSAI

Esse projeto foi implementado visando criar uma ponte de acesso dos usuários para as consultas lógicas, de forma transparente. Assim sendo, os principais arquivos presentes nesse projeto estão em um pacote chamado ‘com’, que basicamente realiza esse papel de

comunicação entre a camada de interface e o acesso às funções presentes no projeto RosaI, que serão mostrados no próximo sub-item. Cada classe apresentada nesse pacote é responsável por tratar um tipo diferente de consulta prevista pelas operações da álgebra ROSA, realizando chamadas às classes responsáveis para tal operação no projeto RosaI. Uma outra classe que aparecerá nesse pacote será a que fará a chamada para a operação de extração do arquivo XML para o arquivo lógico. A TAB. 6.2 apresentada a seguir exibe as classes presentes nesse pacote do projeto WebRosaI.

TAB 6.2 – Classes presentes no projeto WebRosaI

Projeto: **WebRosaI** Pacote: **com –** Viabiliza a comunicação dos pedidos via Web para efetuar as consultas.	
Classe	Descrição
Extract	Classe responsável por fazer a chamada à classe que realiza a extração do arquivo XML para o arquivo lógico.
Join	Utiliza as classes no projeto RosaI responsável pela montagem das junções e pela execução da consulta lógica montada.
NavegPredServ	Chama a classe responsável pela montagem da consulta de navegação em apenas um predicado no RosaI e a classe executora dessa consulta.
NavegPredTrans	Chama a classe no RosaI responsável por realizar a montagem da consultas em vários predicados e a classe que executa a consulta.
Projection	Faz a chamada para as classes responsáveis pela montagem das projeções e execução da consulta no projeto RosaI.
Select	Faz a chamada para as classes responsáveis pela montagem das seleções e execução da consulta no projeto RosaI.
SelectPredServ	Realiza a chamada para a classe que executa a montagem da consulta lógica de seleção de predicado e executando depois a consulta.

Como pode ser visto, esse projeto serviu apenas de ponte entre as chamadas via Web para a execução efetiva das operações de consulta e extração. Detalhes de implementação de todas essas classes poderão ser encontrados junto ao material gravado no "cd" em anexo a essa dissertação. Na próxima sub-seção serão detalhadas as classes que compõem o projeto núcleo dessa implementação, o projeto RosaI.

6.6.2. PROJETO ROSAI

Serão apresentadas aqui as classes que compõem cada um dos três pacotes presentes no projeto RosaI, com sua respectiva descrição.

O primeiro pacote apresentado é o pacote responsável por realizar as consultas que serão analisadas. Nesse pacote estão as classes responsáveis por analisar cada tipo de operação presente na álgebra ROSA, bem como a classe responsável por efetuar a interpretação lógica, realizando a chamada ao SWI-Prolog. É nessa mesma classe que é utilizada a interface de comunicação entre o Java e o Prolog, a interface JPL, já descrita anteriormente. Esse pacote é identificado como 'consultas' e terá sua descrição exibida na TAB. 6.3.

O segundo pacote, se encarrega de realizar a transformação do arquivo XML contendo os LOs e seus relacionamentos para o arquivo lógico, que será entendido pelo Prolog como a base de conhecimento. Nesse pacote estão presentes três classes que atuam em conjunto para tornar essa operação possível. Esse pacote recebe o nome de 'util' e maiores detalhes sobre ele são apresentados na TAB. 6.5.

O terceiro e último pacote, que forma o conjunto desse projeto, é o que faz a descrição das partes que compõem o documento que representa a árvore DOM. Assim, nesse pacote estão presentes as classes que descrevem a formação das estruturas, como por exemplo, a de um LOM. A TAB. 6.5 traz maiores detalhes desse pacote.

TAB 6.3 – Classes presentes no projeto RosaI no pacote 'consultas'

Projeto: **RosaI** Pacote: **consultas –** Montar as consultas lógicas para cada operação da álgebra e executa-as.	
Classe	Descrição
Juncao	Executa a montagem da operação lógica de junção.
NavegPredExpli	Monta a consulta lógica de navegação por um predicado.
NavegPredTran	Monta a consulta lógica de navegação por mais de um predicado.
Predicado_geral	Classe responsável por receber a consulta lógica montada e o arquivo lógico a ser pesquisado pela consulta lógica.
Projeção	Monta as consultas lógicas para executar a projeções.
Selecao	Monta as consultas lógicas de seleção.
SelecPredicado	Monta as consultas lógicas que executam a seleção de predicado.
NavegPred	Monta a consulta lógica para se obter todos os relacionamentos que se ligam a um determinado LO.

TAB. 6.4 – Classes presentes no projeto RosaI no pacote 'estrutura'

Projeto: **RosaI** Pacote: **estrutura –** Cria a estrutura para guardar a árvore DOM obtida.	
Classe	Descrição
Lom	Descreve uma Classe que possui três atributos, dentre eles um de identificação do tipo string, outro que é um objeto do tipo 'general' e outro do tipo 'association'.
General	Classe que descreve seis atributos que irão caracterizar um nó da Classe Lom.
Association	Classe que descreve as associações como uma coleção de vários objetos do tipo 'predicate'.
Predicate	Classe que descreve a estrutura dos predicados como dois atributos. Um do tipo string, que recebe o nome do predicado o outro que um objeto do tipo 'objects'.
Objects	Classe que descreve os objects como uma coleção de identificadores, 'ids'.

TAB. 6.5 – Classes presentes no projeto RosaI no pacote 'util'

Projeto: **RosaI** Pacote: **util –** Pacote que trás a funcionalidade de extração do arquivo XML.	
Classe	Descrição
ExtractXml	Classe que recebe um arquivo XML e extrai todos os seus LOMs para um HashMap chamada de Mapa de LOMs.
ExtractFacts	Classe que recebe a HashMap criada e extrai todos os fatos lógicos, colocando-os em um vetor de fatos.
Extrator	Classe que recebe dois arquivos, um XML e outro lógico, que serão os arquivos de entrada XML e o arquivo lógico de saída a ser criado respectivamente.

6.7. CONSIDERAÇÕES FINAIS

Nesse capítulo foram apresentados detalhes de implementação do sistema protótipo nomeado de ROSAI, que na prática serviu para validar o modelo de expansão do ROSA proposto. Foram apresentados os desafios da implementação e as soluções encontradas, mostrando aspectos da especificação técnica que tornaram a solução possível. No capítulo seguinte, serão apresentadas as considerações finais dessa dissertação, contendo as contribuições desse trabalho de mestrado, bem como algumas sugestões para trabalhos futuros, que poderão engrandecer essa proposta.

7. CONCLUSÕES

O crescimento exponencial da quantidade de páginas presentes na Web fez com que o grande volume de informações contido nas mesmas se apresentassem de maneira heterogênea e desordenada, trazendo por conseqüência problemas de administração desse conteúdo. Nesse contexto, a busca por conteúdos relevantes tornou-se uma operação custosa, e cujo resultado, na maioria dos casos, não atende precisamente às necessidades de consultas feitas por usuários que, por exemplo, utilizam robôs de busca. Uma das razões para esse problema está na ausência de semântica associada aos recursos publicados na Web.

Sendo assim, uma das soluções encontradas para tentar organizar esse conteúdo foi o de trazer maior significado às páginas na Web. Dessa maneira surgiu a idéia da Web Semântica, que tem como objetivo viabilizar uma estruturação semântica dos dados na web, para que os mesmos possam ter suas informações processadas por máquinas [GRUBER, 1993]. Atualmente as consultas elaboradas na web têm a sua semântica dependente da interpretação e precisão daqueles que a elaboram.

Baseado nos princípios da Web Semântica, novos sistemas foram concebidos, valendo-se de ontologias para apoiar o processo de busca. Um desses sistemas é o ROSA, cujo objetivo é apoiar o processo de descoberta de objetos de aprendizagem. O ROSA provê conhecimento através da contextualização de conceitos, utilizando relacionamentos e metadados

O modelo ROSA, no entanto, tem expressividade limitada, uma vez que não considera o uso de regras, a partir dos quais novos fatos possam ser verificados. Este fato motivou esse trabalho de pesquisa, tendo como hipótese que o modelo poderia ser representado em linguagem lógica, permitindo a expressão de regras e o processamento de inferência.

De modo a investigar essa hipótese, decidiu-se adotar a sub-parte da linguagem lógica correspondente a Datalog para tentar representar o modelo estrutural, como também para expressar consultas.

A conclusão obtida é que não só é possível mapear inteiramente o modelo usando primitivas Prolog, como também foi possível estendê-lo incluindo a representação de herança e de propriedade de relacionamentos.

Um aspecto negativo a observar está na ausência do tratamento de esquema e tipagem dos dados, tarefas básicas em um sistema de banco de dados, mas que iriam requerer uma base de conhecimento bem mais complexa.

7.1. CONTRIBUIÇÕES

Nessa dissertação foram estudados o modelo ROSA e suas operações de consultas, bem como elaborada sua tradução para um modelo lógico capaz de garantir a expansão do modelo atual. As seguintes extensões foram implemetadas: suporte a regras de usuários; herança; propriedade de relacionament1o; suporte a inferência. Assim, o modelo aqui criado é mais expressivo do que o atual presente no sistema ROSA original [FABIO F. *et al*, 2004].

7.2. TRABALHOS FUTUROS

Uma das primeiras iniciativas sugeridas é a implementação da proposta do ROSAI, mostrado no capítulo anterior, porém utilizando-se unicamente da programação com regras em Java. Esse tipo de programação é possível de ser realizado, através do uso de algumas APIs (Application Programming Interface) da linguagem, que possibilitam uma programação capaz de inferir regras no próprio ambiente Java, sem a necessidade de realizar chamada a um interpretador Prolog para tal. Dessa maneira, o protótipo proposto no capítulo anterior ganharia um melhor desempenho, já que não mais existiria o acesso à plataforma Prolog externa, e tudo seria executado pela própria máquina virtual Java.

Um outro trabalho é em cima da álgebra ROSA, que poderia ser estendida, para levar em consideração a questão das regras. Nessa reavaliação, a álgebra iria conter operações algébricas atuando juntamente com as regras e com isso, poderiam ser criados planos de execução de consultas com as operações já existentes mais a interpretação das regras. Dessa forma, essa nova versão da álgebra poderia ter suas consultas e interpretações das regras comportamentais, analisadas através de algumas regras de equivalência, que abrem caminho para estudos sobre consultas otimizadas, já que a interpretação das regras no sistema atual fica a cargo do interpretador lógico Prolog e assim, representa uma caixa preta.

Outro trabalho seria dar continuidade ao ambiente de consulta lógico do sistema ROSAI, para que ele agora fosse capaz de realizar consultas inferindo regras em cima de um ambiente P2P. Assim, seria utilizada uma base de regras única, que iria atuar na inferência de novos fatos sobre diversas bases de conhecimento espalhadas geograficamente em lugares distintos. De fato, este trabalho poderá dar continuidade à dissertação de [BRITO, 2005], que já estendeu o ROSA para o ambiente P2P.

8. REFERÊNCIAS

BERNERS-LEE, T., HENDER, J., LASSILA, O., **The Semantic Web,** 4.ed. D Disponível em : http://www.scientificamerican.com/2001/0501issue/0501berners-lee. Último acesso : 12 Junho 2004.

BRACHMAN, R. **On the Epistemological Status of Semantic Networks.** Associative Networks: Representation and Use of Knowledge by Computers. Academic Press. Pp. 3-50. 1979.

BRACHMAN, R. **What's in a concept: structural foundations for semantic networks.** Int. Journal of Man-Machine Studies. 9. Pp. 127-152. 1977.

BRITO G. A. D. DUBOIS
Integração de ObjetoAprendizagem no sistema ROSA em ambiente P2P. Dissertação (mestrado) – Instituto Militar de Engenharia – Rio de Janeiro, 2005.

CERI, S., G. Gottlob, L. Tanca. **Logic Programming and Databases**. , Springer Verlag 1990.

CODD, E. **Relational Completeness of Data Base Languages.** Data Base Systems, Courant Computer Symposia Series, Prentice-Hall, Communications of the ACM, 1972. vol. 6, p. 65-98.

CORCHO, Oscar; GÓMEZ-PÉREZ, Assunción. **A Roadmap to Ontology Specification Language.** Disponível em : http://babage.dia.fi.upm.es/ontoweb/wp1/OntoRoadMap/ekaw00.pdf. Último acesso : Out. 2002.

COSTA F HANDRICK, PORTO F., Relatório Técnico n◦ 104/DE9/SET2004, **Lógica como suporte para Web Semântica**, IME/RJ.

COSTA F HANDRICK, PORTO F., Relatório Técnico n◦ 105/DE9/SET2004, **Processamento de consultas em ambiente Lógico**, IME/RJ.

DECKER, S., Frank, H., Broekstra, J., Erdmann, M., Fensel, D., Horrocks, I., Klein, M., Melnick, S. **The semantic Web – on the respective Roles of XML and RDF,** Disponível em : http://www.ontoknowledge.org/oil/downl/IEEE00.pdf
Último acesso : 06 Janeiro 2005.

DESCRIPTION LOGICS web Site Disponível em : http://www.dl.kr.org.
Último acesso: 10 novembro 2004.

DOM – **Official Web Site for DOM in W3C** Disponível em : http://www.w3.org/DOM/
Último acesso : 04 novembro 2004.

FLORID – **The FLORID/FloXML Project** Disponível em : www.informatik.uni-freiburg.de/~dbis/florid
Último acesso : 02 setembro 2004.

FREITAS, F. **Ontologias e a Web Semântica** In: Anais do XXIII Congresso da Sociedade Brasileira de Computação. Volume 8: Jornada de Mini Cursos em Inteligência Artificial ed.Campinas : Sociedade Brasileira de Computação (SBC), v.8, p. 1-52 (2003).

GRUBER, T. R. 1993. **A Translation Approach to Portable Ontologies.** Knowledge Acquisition 5 (2): 199-220.

HASKELL – **Official Web Site for Haskell** Disponível em : http://www.haskell.org/ Último acesso : 03 julho 2004.

HENDLER, James. **Agents and the Semantic Web.** Disponível em : http://www.cs.umd.edu/users/hendler/AgentWeb.html Último acesso : 10 janeiro 2005.

IEEE: **"Draft Standard for Learning Object Metadata"**; 15 July 2002. Disponível em: http://ltsc.ieee.org/wg12/files/LOM_1484_12_1_v1_Final_Draft.pdf Último acesso : 13 dezembro 2004.

JAVA – **Official Web Site for Java** Disponível em : http://java.sun.com/ Último acesso : 20 janeiro 2005.

JPL – **Official Web Site for Interface Java-Prolog** Disponível em : http://www.swi-prolog.org/packages/jpl/ Último acesso : 11 janeiro 2005.

LEUNG, T. W. et al. **The AQUA Data Model and Algebra.** Technical Report Nº CS-93-09, Brown University, 1993.

LISP - **Official Web Site for Lisp.** Disponível em : http://www.lisp.org/ Último acesso : 3 julho 2004.

MERRIAM - **Merriam Webster Online**. Disponível em : http://www.mw.com/home.htm Último acesso : 18 janeiro 2005.

MICHAEL KIFER, GEORG LAUSEN. **F-Logic: A high Order Language for Reasoning about Objects, Inheritance and Scheme.** ACM SIGMOD Int. Conference on Management of Data 1989, Portland, Oregon: pp. 134-146.

MICHAEL Sintek – **Home Page Pessoal.** Disponível em : http://www.dfki.uni-kl.de/~sintek/ Último acesso : 20 janeiro 2005.

MICHAEL Sintek, Stefan Decker ISWC, Sardinia. **TRIPLE: A Query, Inference, and Transformation Language for the Semantic Web,** June 2002. Disponível em : http://www.dfki.uni-kl.de/frodo/triple/iswc2002/TripleReport.pdf Último acesso : 9 outubro 2004.

NOVAK, J. D. (2003) **The Theory Underlaying Concept Maps and How to Construct Them.** Disponível em : http://cmap.coginst.uwf.edu/info/printer.html Último acesso : 03 de junho de 2003.

NOVAK, J. D. and GOWIN, D. B. (1984) **Learning How to Learn.** New York, Cambridge University Press.

OWL – **Official Web Site for OWL in W3C** Disponível em : http://www.w3.org/TR/owl-features. Último acesso : 07 janeiro 2005.

Porto F., Moura A, Coutinho, F. **ROSA: a Repository of Objects with Semantic Access for e-Learning.** 8 th International DataBase Enginnering & Applications Symposium – IDEAS'04. Portugal, Julho, 2004.

PROLOG – **Apostila 'A linguagem lógica'** Disponível em : http://www2.dem.inpe.br/gadelha/Prolog02.doc. Último acesso : 2 abril 2004.

RDF – **Official Web Site for RDF in W3C.** Disponível em : http://www.w3.org./RDF/ Último acesso : 23 outubro 2004.

REDES SEMANTICAS e Sistemas de Frame – Transparências Disponível em : http://www.di.ufpe.br/~compint/aulas-IAS/frames.ppt Último acesso : 4 outubro 2004.

Ronald J. Brachman, Victoria Pigman Gilbert, and Hector J. Levesque. **An essencial hybrid reasoning system: Knowledge and symbol level accounts in KRYPTON.** In Proc of the 9th Int. Joint Conf. on Artificial Intelligence (IJCAI'85), pages 532-539, 1985.

SCORM OVERVIEW - **Advanced Distributed Learning Sharable ontent Object Reference Model Version 1.2.** Disponível em : http://www.adlnet.org/index.cfm?fuseaction=scormabt**.** Último acesso : 18 Março 2004.

SERVLET – **Official Web Site for Java Servlet Technology** Disponível em : http://java.sun.com/products/servlet/. Último acesso : 20 janeiro 2005.

SILVA, Fábio José Coutinho da. **Processamento de Consultas sobre o Modelo de Dados ROSA** / Fábio José Coutinho da Silva. - Rio de Janeiro : Instituto Militar de Engenharia, 2004. 124 p. : il.
Dissertação (mestrado) – Instituto Militar de Engenharia – Rio de Janeiro, 2004.

STEFAN Decker – **Home Page Pessoal** Disponível em : http://www.isi.edu/~stefan/ Último acesso : 6 janeiro 2005.

SWI-Prolog – **Official Web Site for SWI-Prolog** Disponível em : http://www.swi-prolog.org/ Último acesso : 3 novembro 2004.

Thomas A. Lipkis. **A KL-ONE classifier.** In [Schomolze and Brachman1982], pag 128-145. Published as BBN Research Report 4842, Bolt Beranek and Newman Inc., June 1982.

URIs - **Naming and Addressing: URIs, URLs.**
Disponível em : http://www.w3.org/Addressing/. Último acesso: 13 janeiro 2005.

W3C – **Official Web Site for World Wide Web Consortium**
Disponível em : http://www.w3.org
Último acesso : 2 dezembro 2004.

WebChang Zhaos, Guizhen Yang, and Michael Kifer.
FLORA-2: A Rule-Based Knowledge Representation and Inference Infrastructure for the Semantic Disponível em :
http://www.cse.buffalo.edu/faculty/gzyang/papers/odbase2003.pdf
Último acesso : 02 setembro 2004.

WOODS, W. **What's in a Link: Foundations for Semantic Networks. Representation and Understanding:**
Studies in Cognitive Science. Academic Press. Pp. 35-82. 1975.

XML – **Official Web Site for XML in W3C** Disponível em : http://www.w3c.org/XML/
Último acesso : 16 janeiro 2005.

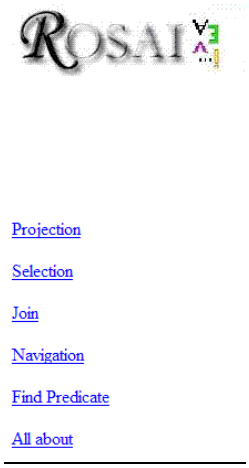
XSB – **Official Web Site for Logic Programming XSB** Disponível em :
http://xsb.sourceforge.net
Último acesso : 12 janeiro 2005.

9. APÊNDICES

9.1. APÊNDICE 1: PRINCIPAIS LAYOUTS DO PROTÓTIPO ROSAI

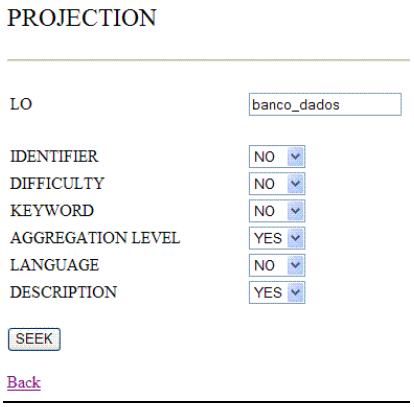
Tela de abertura no Browser:

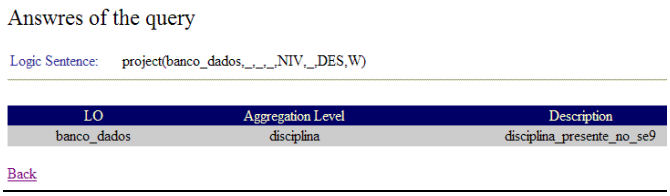
Tela de inicial do ROSAI com os Menus de consulta



Tela da consulta de uma Projeção e sua reposta

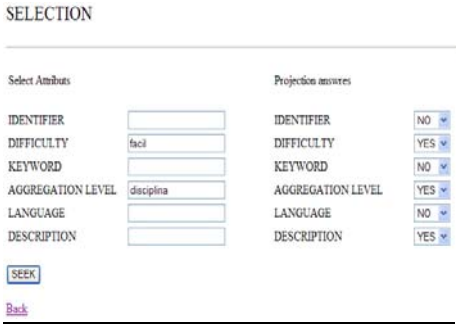
Qual o nível de agregação e a descrição do LO “banco_dados” ?

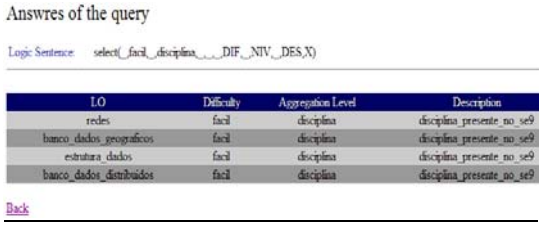




Tela da consulta de uma Seleção e sua reposta

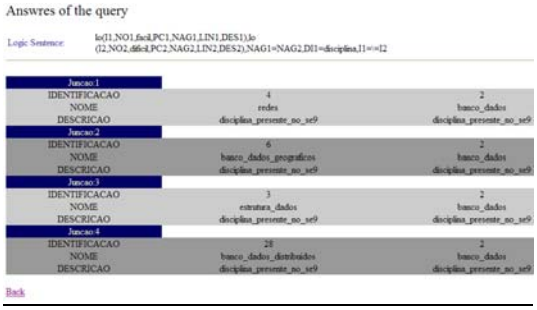
Quais os LOs no qual o dificuldade seja fácil e seu nível de agregação seja disciplina ?





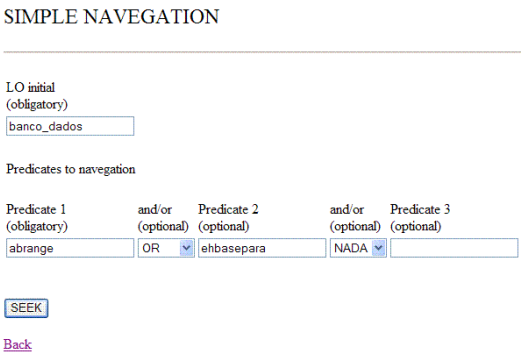
Tela da consulta de Junção e sua reposta

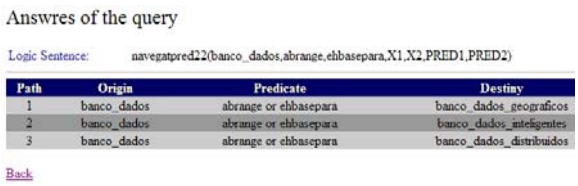
Dado uma coleção de LOs formada por LOs de nível de dificuldade fácil e uma outra coleção formada por LOs de nível de dificuldade difícil, quais os pares formado cujo nível de agregação seja igual a disciplina ?



Tela da consulta de Navegação Simples e sua reposta

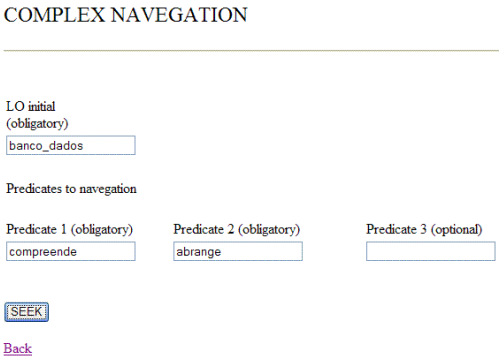
Quais os LOs que o LO ‘banco de dados’ abrange ou é base ?

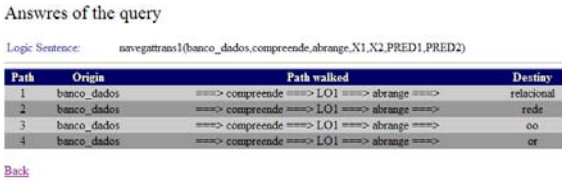




Tela da consulta de Navegação Complexa e sua reposta

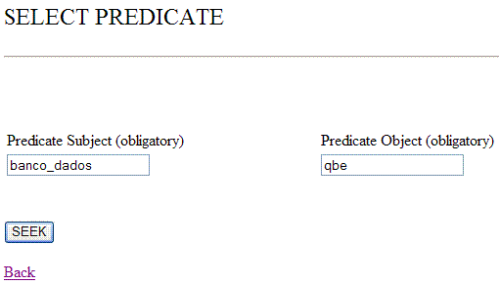
Quais os LOs abrangidos por LOs , no qual o LO ‘banco de dados’ compreende ?





Tela da consulta de Seleção de Predicados e sua reposta

Qual a relação existente entre o LO ‘banco de dados’ e o LO ‘qbe’ ?



Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo