UNIVERSIDADE DE SÃO PAULO ESCOLA POLITÉCNICA
DEPARTAMENTO DE ENGENHARIA MECATRÔNICA

nota final
9,2 (nove e dois)

ESPECIFICAÇÃO DE PRODUTOS E PARTES EM MANUFATURA UTILIZANDO ONTOLOGIAS BASEADAS EM LÓGICA E PROBABILIDADES

Monica Goes Eboli

São Paulo
2005

UNIVERSIDADE DE SÃO PAULO ESCOLA POLITÉCNICA
DEPARTAMENTO DE ENGENHARIA MECATRÔNICA

ESPECIFICAÇÃO DE PRODUTOS E PARTES EM MANUFATURA UTILIZANDO ONTOLOGIAS BASEADAS EM LÓGICA E PROBABILIDADES

Trabalho de formatura apresentado à Escola
Politécnica da Universidade de São Paulo para
Obtenção do título de Graduação em Engenharia

Monica Goes Eboli

Orientador: Fábio G. Cozman

Área de Concentração:
Engenharia Mecatrônica

São Paulo
2005

A um grande amigo e companheiro de
caminhada, Eduardo Barjud Bugelli.

RESUMO

O objetivo deste projeto é modelar peças de manufatura de maneira mais expressiva e aplicar este tipo de modelagem em tecnologia de grupo. Para obter maior expressividades, ontologias em lógica descritiva foram associadas a probabilidades e transformadas em uma rede bayesiana. Um programa livre chamado Protégé foi utilizado para modelar as ontologias, um programa em C++ foi desenvolvido para realizar a transformação através do método de Ding & Peng e o programa JavaBayes foi utilizado para tratar a rede bayesiana.

ABSTRACT

The goal of this Project was to model manufacturing parts in a more expressive way so as to apply to group technology. In order to obtain more expressivity, ontologies in descriptive logics were combined with probabilities and then translated into a Bayesian network. A software called Protégé was used to model the ontologies, a software in C++ was developed to turn the ontologies into a Bayesian network using Ding & Peng method and the software JavaBayes was used to manipulate the Bayesian network.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

1. INTRODUÇÃO

Desde a época de Henry Ford sabe-se que produzir em massa é mais eficiente e traz mais benefícios, mas com a evolução da indústria e da tecnologia aumentou-se a diversificação de produtos e conseqüentemente de peças que os compõem. Para aumentar a eficiência na produção de tantas peças variadas, teve-se a idéia de agrupá-las de acordo com suas características de projeto e manufatura. Assim foi criado o conceito de tecnologia de grupo [10 e 13], que no início era aplicado em células de manufatura, mas com o tempo expandiu-se para outras áreas como, por exemplo, a de planejamento com ferramentas CAPP que utilizam o método variante.

A tecnologia de grupo tráz diversos benefícios à manufatura, reduzindo o tempo de setup entre duas operações, evitando esforços desnecessários ao se focar apenas nas diferenças necessárias, armazenamento e recuperação de informações em menor tempo e com isso evitando-se resolver problemas já solucionados e a proliferação de novos itens desnecessários. Todas estas vantagens culminam em redução de custos que é muito importante para a competitividade do produto nos dias de hoje.

O modo tradicional de se fazer esta classificação de peças é através de códigos alfa-numéricos, porém para os sistemas de classificação utilizados na implementação da tecnologia de grupos é importante que sua estrutura atenda aos objetivos da aplicação e sejam flexíveis para suportar alterações de elementos e introdução de novos elementos, tecnologias e processos. Devido a estas necessidades, a tendência é que estes sistemas de classificação por códigos sejam substituídos por métodos mais expressivos que além de aumentar muito sua flexibilidade, facilitarão o uso por utilizar uma linguagem de mais alto nível e conseqüentemente mais fácil entendimento.

A proposta deste trabalho é a modelagem de componentes de manufatura criando-se uma ontologia para eles, utilizando-se uma linguagem conhecida por OWL, e em um estágio posterior sua associação a probabilidades para obter maior expressividade e com isso maior correspondência com o mundo real.

A palavra ontologia representa uma teoria específica sobre a natureza do ser ou existir. Ela representa o que existe no mundo; fatos, objetos e relações. No caso da manufatura representa classes de peças, suas características e relações.

OWL é Web Ontology Language, uma linguagem de representação de alto nível criada para a representação de dados que devem ser processados. Ela facilita a interpretação por máquinas, mas mantém-se facilmente compreensível para homens.

Como ferramenta de modelagem de ontologias em OWL utilizar-se-á o software Protégé [8] que é um software livre desenvolvido por Stanford Medical Informatics.

Este trabalho também trata de transformar esta ontologia construída em uma rede bayesiana, que é uma forte ferramenta da modelagem e inferência para dados incertos, através do método de Ding & Peng [4].

O arquivo em XML contendo a ontologia em OWL fornecido pelo Protégé [8] será transformado, através e um programa em C++, num arquivo no formato BIF contendo uma rede bayesiana que pode ser aberto pelo programa JavaBayes [11] que edita e trata redes bayesianas.

2. OBJETIVOS

Há dois objetivos principais neste projeto. O primeiro é conseguir associar ontologias em OWL a probabilidades condicionais e transformar este conjunto de informações em uma rede bayesiana.

O segundo é aplicar este processo a peças de manufatura, exemplificando assim sua utilização, e demonstrando sua aplicação no campo da manufatura.

3. LÓGICAS

Foram estudados diversos tipos de lógica das mais baixas para mais altas antes de suas associações com probabilidades. Os tipos de lógica estão sucintamente descritos a seguir.

3.1 Lógica Proposicional

A lógica proposicional de acordo coma descrição de [6] é o tipo mais simples das lógicas estudadas. Sua sintaxe define as sentenças atômicas (elementos sintáticos indivisíveis) que consistem em um único símbolo proposicional, representando uma proposição que pode ser verdadeira ou falsa (ou eventualmente desconhecida).

Seus conectivos de usos logísticos são:

- $\neg$ Negação
- $\wedge$ Conjunção ("e")
- v Disjunção ("ou")
- $\Rightarrow$ Implicação (Se ...., então....)
- $\Leftrightarrow$ Bicondicional

Sua ordem de precedência lógica, da mais alta para a mais baixar é:

$\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$

A seguinte tabela da verdade demonstra suas aplicações:

Tabela 3.1 Funções lógicas

A	B	$\neg A$	$A \wedge B$	$A \vee B$	$A \Rightarrow B$	$A \Leftrightarrow B$
Falso	Falso	Verdade	Falso	Falso	Verdadeiro	Verdadeiro
Falso	Verdade	Verdade	Falso	Verdadeiro	Verdadeiro	Falso
Verdade	Falso	Falso	Falso	Verdadeiro	Falso	Falso
Verdade	Verdade	Falso	Verdadeiro	Verdadeiro	Verdadeiro	Verdadeiro

3.2 Lógica de Primeira Ordem [6]

Os elementos sintáticos básicos são símbolos de constantes que representam objetos, símbolos de predicado que representam relações e símbolos de funções que representam funções.

Exemplo: suponha uma família com duas irmãs Ana e Maria, sendo Maria a mais velha.

Objetos: Ana, Maria

Relações: Irmã, Concunhado, Marido, MaisVelha.

As sentenças de lógica de primeira ordem podem ser atômicas ou complexas. Sentenças atômicas enunciam fatos e sentenças complexas utilizam conectivos lógicos.

Exemplo:

Atômica:

Irmã(Maria, Ana) enuncia que Maria é irmã de Ana

Concunhado(Marido(Maria),Marido(Ana)) enuncia que o marido de Maria é concunhado do marido de Ana.

Complexas:

Irmã(Maria, Ana) $\wedge$ Irmã (Ana, Maria)

$\neg$ MaisVelha(Ana) $\Rightarrow$ MaisVelha(Maria) enuncia que se Ana não é a mais velha então Maria é a mais velha.

Além de possuir os mesmos conectivos da lógica proposicional, a lógica de primeira ordem possui quantificadores, que servem para expressar propriedades de um grupo inteiro de objetos. Existem quantificadores universais e existenciais. Universais fazem declarações sobre todo o objeto enquanto os existenciais fazem sobre algum objeto (ao menos um).

Exemplo:

Universal: $\forall$ x Cachorro(x) $\Rightarrow$ Mamífero(x), ou seja, todo cachorro é mamífero.

Existencial: $\exists$ x Humano(x) $\Rightarrow$ Mulher(x), ou seja, alguns humanos são mulheres.

3.3 Lógica Descritiva

Lógica Descritiva [1 e 3] é um tipo especifico de lógica de primeira ordem. Ela é uma notação projetada para facilitar a descrição de definições e propriedades das categorias.

As sentenças da lógica descritiva podem ser separadas em dois grupos, o TBox e o ABox que se referem respectivamente a *terminology* e ao *assertional knowledge*. Deve-se notar que são assumidas as seguintes hipóteses para a terminologia da Lógica Descritiva:

- Apenas uma definição para cada nome de conceito é aceita
- Definições são acíclicas, no sentido de não serem definidas por si mesmas nem em outros conceitos que indiretamente se refiram a elas.

Exemplo de TBox:

$Mulher \equiv Pessoa \wedge Fe \min ino$

$Homem \equiv Pessoa \wedge \neg(Pessoa \wedge Fe \min ino)$

$Mãe \equiv (Pessoa \wedge Fe \min ino) \wedge \exists temFilho.Pessoa$

$Pai \equiv (Pessoa \wedge \neg(Pessoa \wedge Fe \min ino)) \wedge \exists temFilho.Pessoa$

Exemplo de ABox:

$Fe \min ino \wedge Pessoa(Ana)$

$temFilho(Ana, João)$

3.4 Lógica Probabilística

A crença do agente sobre os fatos em diversas lógicas em muitos casos é definida por verdadeira, falsa ou desconhecida. Na lógica com probabilidades estes "valores" são substituídos por valores probabilísticos entre zero e 1. Ao fazer esta substituição apesar de tornamos o problema a ser resolvido mais complicado, o tornamos mais coerente com o sistema sendo modelado, onde geralmente não se possuem dados exatos.

4. OWL

Para realizar a modelagem da ontologia em lógica descritiva, foram estudadas duas linguagens a fundo, OWL e OWL-S [7]. Abaixo se segue uma pequena descrição destas linguagens.

4.1 OWL

OWL é a Web Ontology Language, é uma linguagem desenvolvida para a criação de ontologias. Ontologia é um termo utilizado para descrever a representação de entidades de um mundo e suas relações. Ela pode ser utilizada entre outros, para os seguintes objetivos:

- Formalizar um domínio através de classes e suas propriedades
- Definir indivíduos e suas propriedades
- Racionalizar estas classes e indivíduos.

Sintaxe

Classes

Classes fornecem um mecanismo de abstração para agrupar elementos de características similares. Em OWL classes são associadas a conjuntos de indivíduos, chamado "class extension", e estes indivíduos são chamados "instances" desta classe.

Há seis modos de descrever classes usando OWL, pelo seu identificador, por enumeração exaustiva de indivíduos, restrições de propriedades e intersecção, união e complemento de classes.

O primeiro modo é diferente dos outros, pois o faz através do nome da classe enquanto os outros descrevem anonimamente. Abaixo, um exemplo de definição por identificador:

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

Por enumeração exaustiva de indivíduos, devem-se enumerar todos os indivíduos que são "instances" da classe.

```
<owl:Class>
<owl:oneOf rdf:parseType="Collection">
  <owl:Thing rdf:about="#Mindinho"/>
  <owl:Thing rdf:about="#SeuVizinho"/>
  <owl:Thing rdf:about="#PaiDeTodos"/>
  <owl:Thing rdf:about="#FuraBolos"/>
  <owl:Thing rdf:about="#MataPiolhos"/>
</owl:oneOf>
</owl:Class>
```

Por restrição de propriedades é um tipo especial de descrever classes, pois ela descreve anonimamente a classe, todas as classes onde todos os individuais obedeçam às restrições pertencem a classe descrita. Estas restrições podem ser de valores ou cardinalidade. O primeiro tipo restringe a extensão de valores possíveis enquanto o segundo tipo restringe o número de valores que uma propriedade pode receber.

Os últimos três tipos de construtores são muito utilizados em lógica descritiva, são equivalentes aos operadores AND, OR e NOT aplicados às classes.

Para realizar-se a operação lógica AND utiliza-se o operador "intersectionOf", como no exemplo abaixo:

```
<owl:Class>
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#Brie" />
        <owl:Thing rdf:about="#DoReino" />
      </owl:oneOf>
    </owl:Class>
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#DoReino" />
        <owl:Thing rdf:about="#Camembert" />
      </owl:oneOf>
    </owl:Class>
  </owl:intersectionOf>
</owl:Class>
```

Neste caso a classe resultante conteria o resultado desta operação lógica que no caso é "DoReino".

Para realizar a operação lógica OR utiliza-se o operador "unionOf", como no exemplo abaixo:

```
<owl:Class>
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#Brie" />
        <owl:Thing rdf:about="#DoReino" />
      </owl:oneOf>
    </owl:Class>
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#DoReino" />
        <owl:Thing rdf:about="#Camembert" />
      </owl:oneOf>
    </owl:Class>
  </owl:unionOf>
</owl:Class>
```

Neste caso a classe resultante conteria o resultado desta operação lógica que no caso é "DoReino", "Brie"e "Camembert".

Para realizar a operação lógica NOT utiliza-se o operador "complementOf", como no exemplo abaixo:

```
<owl:Class>
  <owl:complementOf>
    <owl:Class rdf:about="#Parafusos"/>
  </owl:complementOf>
</owl:Class>
```

No caso do nosso exemplo pertenceriam a esta classe todos os objetos que não fossem "Parafusos".

Em OWL também existem axiomas de classes que são utilizados para descrever as classes comparando-as com outras. Estes axiomas são subClassOf, equivalentClass e disjointWith.

subClassOf indica que uma classe C_1 é uma subclasse de outra classe C_2. Isto quer dizer que todos os membros de C_1 também são membros de C_2. O seu uso é exemplificado abaixo:

```
<owl:Class rdf:ID="Mulher">
      <rdfs:subClassOf rdf:resource="Humanos"/>
 </owl:Class>
```

No caso do exemplo, todas as mulheres são humanos e possuem as características de humanos.

equivalentClass indica que uma classe C_1 é equivalente a outra classe C_2. Isto quer dizer que todos C_1 e C_2 possuem os mesmos membros. O seu uso é exemplificado abaixo:

```
<owl:Class rdf:ID="Bolas_Vermelhas_Pequenas">
  <rdfs:equivalentClass rdf:resource="Bolas_Pequenas_Vermelhas"/>
 </owl:Class>
```

disjointWith indica que uma classe C_1 é desconexa a outra classe C_2. Isto quer dizer que todos C_1 e C_2 não possuem nenhum membros em comum. O seu uso é exemplificado abaixo:

```
<owl:Class rdf:ID="Homem">
      <rdfs:disjointWith rdf:resource="Mulher"/>
 </owl:Class>
```

Propriedades

Em OWL há dois tipos de propriedades, as de objeto que ligam indivíduos a indivíduos e as de DataType que ligam indivíduos a valores de dados que podem ser strings, inteiros, reais, etc. Por exemplo, PossuiFilho é do tipo objeto, pois une um indivíduo humano a um outro indivíduo humano, e PossuiTantosFilhos é do tipo DataType, pois associa um individuo a um número inteiro.

Um axioma de propriedade define as características da propriedade, em sua forma mais simples ele apenas define sua existência.

```
<owl:ObjectProperty rdf:ID="PaiDe"/>
```

Existem três construtores para propriedades: subPropertyOf, domain e range. subPropertyOf indica que uma propriedade P_1 é subpropriedade de outra propriedade P_2, seu uso é mais facilmente explicado pelo exemplo abaixo:

```
<owl:ObjectProperty rdf:ID="PaiDe">
      <rdfs:subPropertyOf rdf:resourse="ParenteDe">
</owl:ObjectProperty>
```

Ou seja, se o João é PaiDe José e "PaiDe" é subclasse de "ParenteDe" isto significa que João é ParenteDe José.

domain indica o domínio das classes que podem possuir esta propriedade e range limita estas propriedades para as classes. A aplicação de ambos pode ser vista no exemplo abaixo:

```
<owl:ObjectProperty rdf:ID="PaiDe">
      <rdfs:domain rdf:resource="Homem"/>
      <rdfs:range rdf:resource="Humano"/>
</owl:ObjectProperty>
```

Este exemplo diz que para ser PaiDe alguém se deve ser Homem e que o filho deve ser Humano.

Em OWL é possível descrever as relações entre propriedades utilizando-se "equivalentProperty" e "inverseOf" "equivalentProperty" significa que duas propriedades equivalentes então qualquer par que obedeça a uma elas obedecerá as duas. E inverseOf serve para mostrar relações inversas, como por exemplo "ÉPaiDe" é inversa a "ÉFilhoDe".

As propriedades em OWL também possuem algumas características lógicas, propriedades transitivas e de simetria. Veja exemplo de declaração de propriedade transitiva abaixo:

```
<owl:ObjectProperty rdf:ID="Prop">
  <rdf:type rdf:resource="&owl;TransitiveProperty" />
</owl:ObjectProperty>
```

Isto significa que se o par (x,y) é um instance de Prop e (y,z) também é instance de Prop podemos inferir que (x,z) também será instance de Prop.

A propriedade de simetria é declarada como mostrado abaixo:

```
<owl:SymmetricProperty rdf:ID="Prop2">
</owl:SymmetricProperty>
```

Isto significa que se o par (x,y) é instance de Prop2, o par simétrico (y,x) também será.

Indivíduos

Ao definir um indivíduo o declaramos na classe à que pertence e suas características, como é mostrado abaixo:

```
<Pessoa rdf:ID="Poliglota">
      <Linguas rdf:datatype=" float">5</Linguas>
</Pessoa>
```

Há algumas ferramentas em OWL que nos ajudam na identificação de indivíduos, pois nem sempre indivíduos com nomes diferentes referem-se a coisas diferentes.

sameAs indica indivíduos que possuem nomes diferentes, mas referem-se a mesma coisa, e é utilizado da seguinte forma:

```
<rdf:Description rdf:about="Clark_Kent">
```

```
  <owl:sameAs rdf:resource="Superman"/>
</rdf:Description>
```

differentFrom indica indivíduos que não se referem a mesma coisa e é utilizado as seguinte forma:

```
<owl:Class rdf:ID="Alhos">
  <owl:differentFrom rdf:resource="Bulgalhos"/>
</owl:Class>
```

E por fim AllDifferent lista diversos indivíduos todos diferentes entre si, da seguinte forma de forma similar ao differentOf.

4.2 OWL-S [12]

Entre os recursos mais importantes da web estão os fornecedores de serviços que não são necessariamente apenas fornecedores de informação estática. Sua descrição deve permitir usuários localizar, selecionar, empregar, compor e monitorar serviços web automaticamente. Para que isto seja possível, um agente de software necessita de uma descrição do serviço interpretável pelo computador e meios para acessá-la.

A ontologia dos serviços é motivada pela necessidade de responder três tipos essenciais de perguntas sobre o serviço:

- O que o serviço requer do usuário/agente e provê para eles? Esta resposta é dada pelo Profile
- Como funciona? Esta resposta é fornecida pelo modelo (Model)
- Como é usado? Esta resposta é fornecida pelo Grounding

OWL-S é uma linguagem criada com o objetivo de atender estas necessidades, pois já possui uma ontologia formada com uma classe de processos.

Esta ontologia de serviços possui uma classe muito importante que é a classe processo, e a ela estão associadas as seguintes propriedades: hasParameter, hasInput, hasOutput, hasPrecondition e hasEffect que abrangem as classes Parameter, Input, ConditionalOutput, Precondition e ConditionEffect.

Os inputs e outputs determinam a transformação dos dados causada pelo processo. Os inputs são a informação requerida para a execução do processo e são geralmente enviadas de outros processos ou de um cliente de webservice.Os outputs gerados são ou enviados para outro processo ou para webservices.

Um processo pode alterar o estado do mundo, estas alterações são chamadas de efeitos. Processos possuem precondições que são condições que precisam estar satisfeitas para que o processo possa ser executado corretamente, por esta necessidade existem as classes ConditionalOutput e ConditionalEffect que associam condições a outputs e efeitos.

Há três tipos de processos: Atômico, Simples e Composto. Processos atômicos e simples não possuem subprocessos e são realizados com apenas um passo na perspectiva de quem requisitou o serviço, não há visibilidade de sua execução. A única diferença é que o processo atômico precisa ter um grounding, ou seja, instruções de como acessar o serviço.

Processos compostos podem ser decompostos em outros processos. Esta decomposição pode ser realizada a partir de constructs de controle como SEQUENCE e IF-THEN-ELSE. Sua decomposição geralmente é responsável por mostrar como os inputs são distribuídos entre os subprocessos e como os outputs são retornados pelos subprocessos.

A ontologia de processos presente em OWL-S pode ser visualizada no esquema a a seguir:

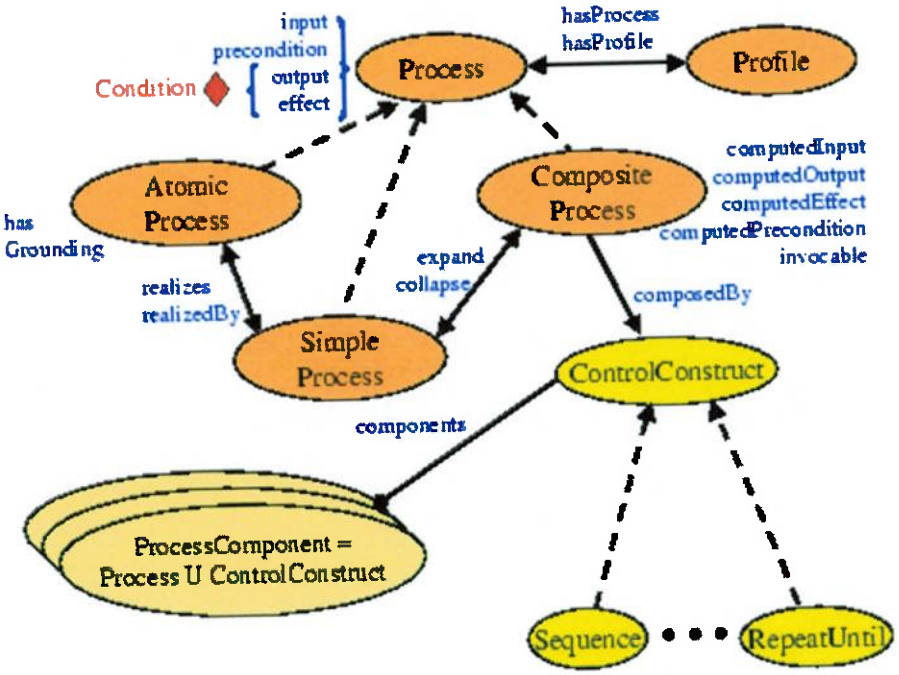

Figura 4.1 Esquema da Ontologia de processos de OWL-S

5. REDES BAYESYANAS

Redes Bayesianas são usadas para modelar conhecimento incerto. Uma rede bayesiana é um grafo direcional acíclico com nós representando variáveis e arcos representando relações de dependência entre as variáveis.

A rede bayesiana é a representação da distribuição conjunta de todas as variáveis representadas nela por nós. Nela cada nó depende de seus pais o que gera uma tabela de distribuição condicional mostrando esta dependência. No caso de nós sem pais sua tabela de probabilidades é incondicional.

A partir de seu estudo é possível responder questões sobre dependência entre as variáveis e realizar inferências. O objetivo das inferências é descobrir a distribuição de um subgrupo de variáveis a partir de outro subgrupo (evidência).

Na figura abaixo, se pode ver um exemplo de rede bayesiana.

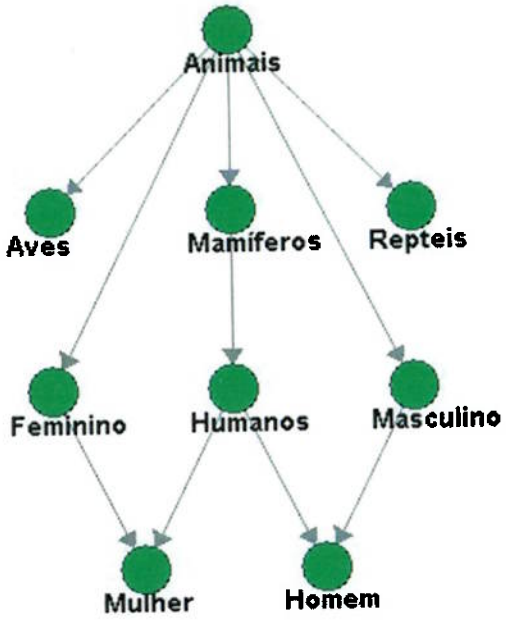

Figura 5.1 Exemplo de Rede Bayesiana

6. METODO DE DING & PENG [2]

Este método serve para o tratamento de probabilidades associadas a classes e propriedades, indivíduos não são considerados.

Para que a possa ser feito o mapeamento da ontologia em uma rede bayesiana é necessário que os dados contidos na ontologia sejam acíclicos, supõe-se isso de todas as ontologias a serem tratadas. Nas subseções seguintes e descrita a forma como adicionar os dados probabilísticos às ontologias, como realizar a construção da rede e das tabelas de probabilidade condicional de cada nó.

6.1 Modelamento da Ontologia

A aplicação do método começa na forma de modelar a ontologia de um modo que possibilite associar dados probabilísticos às classes, e para isso devem ser criadas as seguintes classes para armazenas os valores probabilísticos:

- *PriorProbObj* para representar probabilidades do tipo "P(A)", esta classe possui duas propriedades:
  - **hasVariable* que recebe o nome da variável
  - **hasProbValue* que recebe o valor da probabilidade

- *CondProbObjT* para representar probabilidades do tipo "P(A/B)", esta classe possui três propriedades:
  - **hasCondition* que recebe o nome das variáveis condicionantes.
  - **hasVariable* que recebe o nome da variável condicional.
  - **hasValeu* que recebe o valor da probabilidade

- *CondProbObjF* para representar probabilidades do tipo "P(A/¬B)", esta classe possui três propriedades:

  **hasCondition* que recebe o nome das variáveis condicionantes.

  **hasVariable* que recebe o nome da variável condicional.

  **hasValeu* que recebe o valor da probabilidade

6.2 Transformação

Uma vez modelada a ontologia, para transformá-la em uma rede bayesiana basta seguir as seguintes regras:

1. Classes "C" primitivas ou definidas e classes "Res" anônimas definidas por restrições serão mapeadas para um nó na Rede Bayesiana cuja variável poderá assumir dois estados, "true" "false".

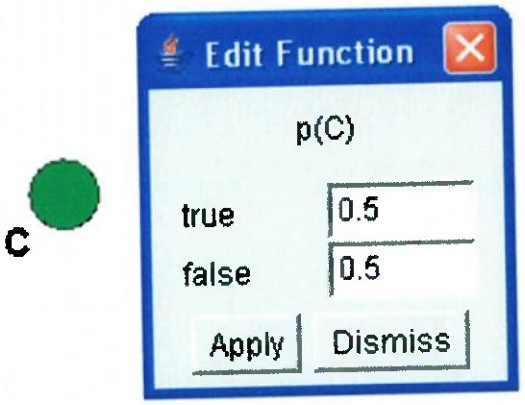

Figura 6.1 Exemplo de Classe C

2. Toda propriedade "P" de domínio "D" e imagem "R" será mapeada para um nó "Class_P" e sua variável terá dois estados ("true" e "false"). "Class_P" deve ser uma subclasse da classe do domínio "D".

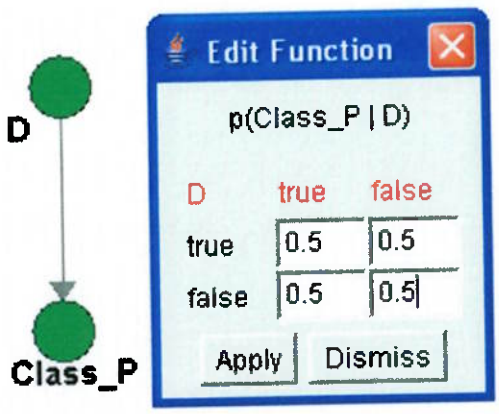

Figura 6.2 Exemplo Propriedade P

3. Toda classe "Class_P" terá um nó-filho "GR_P", quando a imagem é especificada pela ontologia, "GR_P" possui dois estados possíveis "R" ou "irrelevante". Caso não seja especificado ele possui o estado "Thing".

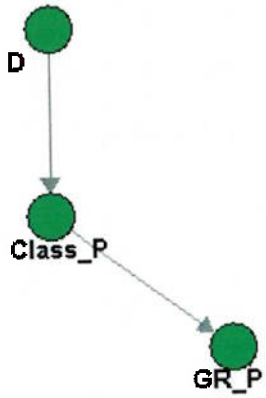

Figura 6.3 Exemplo Propriedade P2

4. Classes "Res" definidas pelas restrições podem ser de dois tipos, de restrição de cardinalidade e de imagem. Ela é mapeada como um nós "LC_P_Res" para restrição de cardinalidade local e "LR_P_Res" para restrição imagem local.

5. Subclasses se tornam nós-filhos dos nós de suas superclasses. O mesmo vale para propriedades.

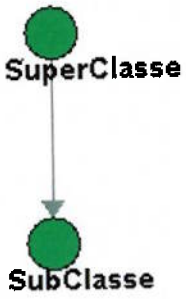

Figura 6.4 Exemplo de SubClasse

6. Há um arco ligando nós de classes que são mutuamente exclusivas ou disjoint, a direção não importa.

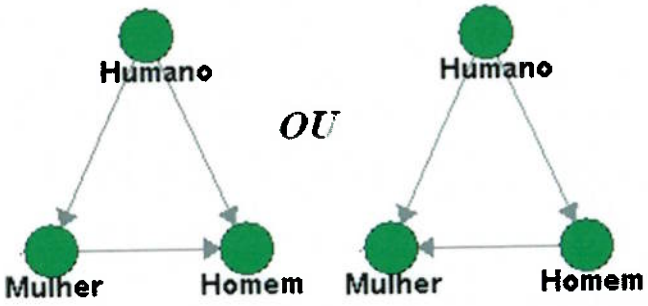

Figura 6.5 Exemplo de Classes que estão "disjoint"

7. Há arcos ligando arcos explicitamente dependentes, como por exemplo, quando é utilizada a função Collection e intersection0f.

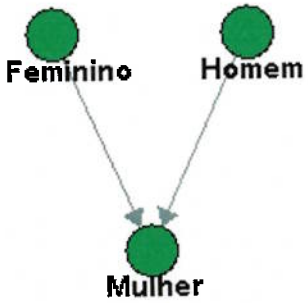

Figura 6.6 Exemplo de Intersecção

6.3 Construção das tabelas de probabilidade condicional

Para finalizar a construção da rede bayesiana é necessário associar a cada nó uma tabela de probabilidade condicional. As regras são as seguintes:

1. Para um nó "A" que não possui pais, colocamos apenas P(A) em sua tabela.
2. Caso "A" seja uma subclasse (nó-filho) de "B", sua tabela deve conter P(A/B). Notando que P(A/¬B)=0
3. Se a classe "A" for *disjoint* com a classe "B", sua tabela deve conter P(A/¬B). Notando que P(A/B)=0.
4. Algumas tabelas são determinadas apenas por relações lógicas, como será demonstrado no exemplo.
5. No caso de nós com diversos pais, são necessários cálculos para obter a tabela completa. Mas, para algumas classes, como por exemplo, uma classe "C" com diversos pais "S_1, S_2, ..., S_n" pode ser que seja impossível a computação destes valores. Para estes casos serão tomadas algumas medidas ainda não definidas.

7. EXEMPLOS DE TRANSFORMAÇÃO

Para demonstrar as regras de transformação será utilizado um exemplo fornecido pelos autores do método [9].

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

Regra 1

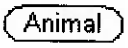

Figura 7.1 Passo 1

```
<owl:Class rdf:ID="Male">
  <rdfs:subClassOf rdf:resource="#Animal"/>
</owl:Class>
```

Regra 5

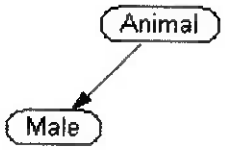

Figura 7.2 Passo 2

```
<owl:Class rdf:ID="Female">
  <rdfs:subClassOf rdf:resource="#Animal"/>
  <owl:disjointWith rdf:resource="#Male"/>
</owl:Class>
```

Regras 5 e 6

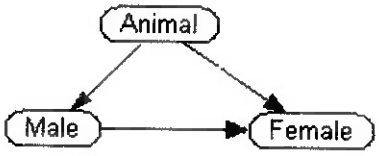

Figura 7.3 Passo 3

```
<owl:ObjectProperty rdf:ID="hasParent">
  <rdfs:domain rdf:resource="#Animal"/>
  <rdfs:range rdf:resource="#Animal"/>
</owl:ObjectProperty>
```

Regras 2 e 3

Figura 7.4 Passo 4

```
<owl:ObjectProperty rdf:ID="hasFather">
  <rdfs:subPropertyOf rdf:resource="#hasParent"/>
  <rdfs:range rdf:resource="#Male"/>
</owl:ObjectProperty>                          Regras 2 e 3
```

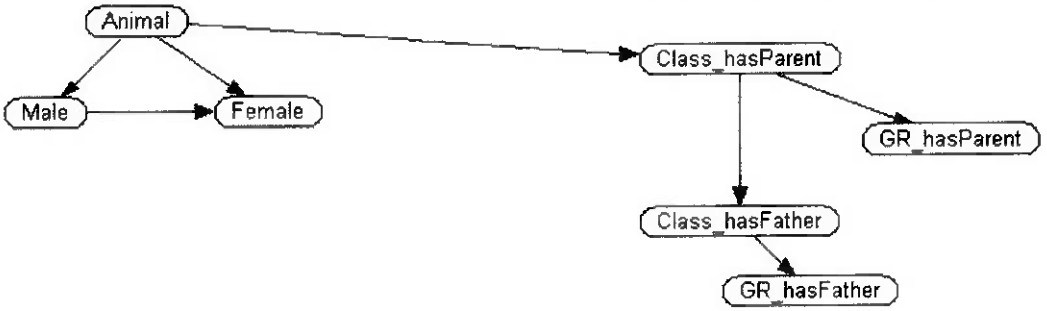

Figura 7.5 Passo 5

```
<owl:ObjectProperty rdf:ID="hasMother">
  <rdfs:subPropertyOf rdf:resource="#hasParent"/>
  <rdfs:range rdf:resource="#Female"/>
</owl:ObjectProperty>                          Regras 2 e 3
```

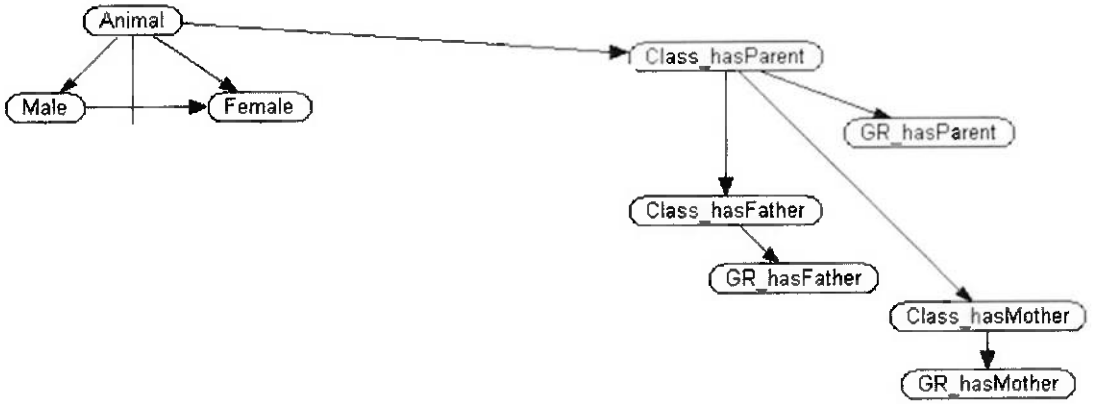

Figura 7.6 Passo 6

```
<owl:Class rdf:ID="Human">
  <rdfs:subClassOf rdf:resource="#Animal"/>
  <rdfs:subClassOf>
    <owl:Restriction rdf:ID="Res01">
      <owl:onProperty rdf:resource="#hasParent"/>
```

```
      <owl:allValuesFrom rdf:resource="#Human"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction rdf:ID="Res02">
      <owl:onProperty rdf:resource="#hasFather"/>
      <owl:cardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction rdf:ID="Res03">
      <owl:onProperty rdf:resource="#hasMother"/>
      <owl:cardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>                                        Regras 5 e 6
```

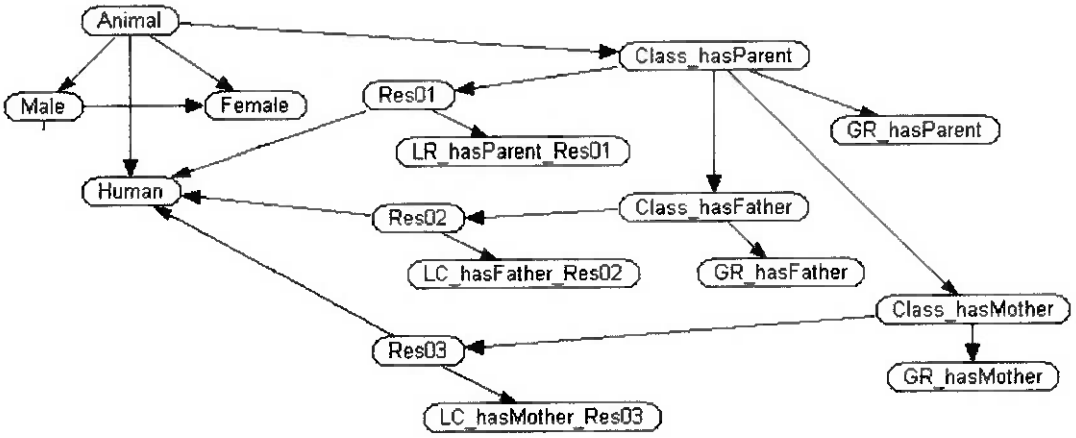

Figura 7.7 Passo 7

```
<owl:Class rdf:ID="Man">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Human"/>
    <owl:Class rdf:about="#Male"/>
  </owl:intersectionOf>
</owl:Class>                                        Regra 7
```

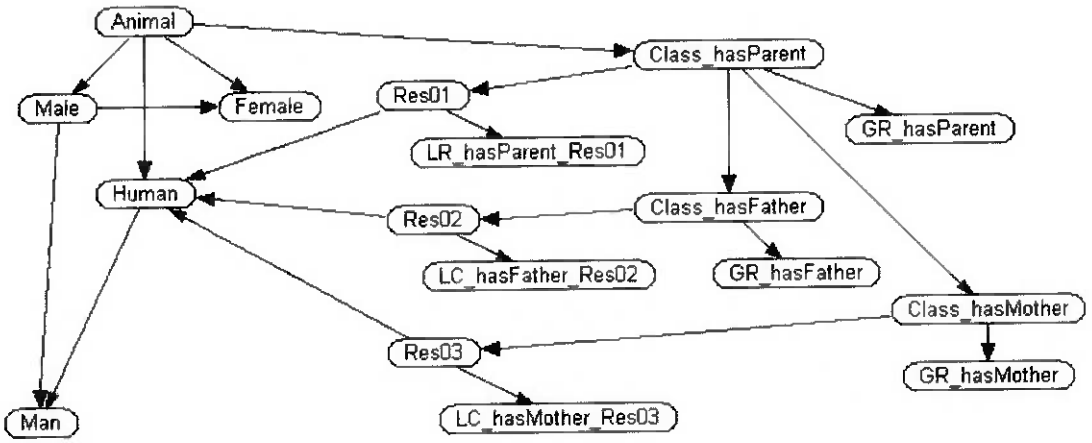

Figura 7.8 Passo 8

```
<owl:Class rdf:ID="Woman">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Human"/>
    <owl:Class rdf:about="#Female"/>
  </owl:intersectionOf>
</owl:Class>
```

Regra 7

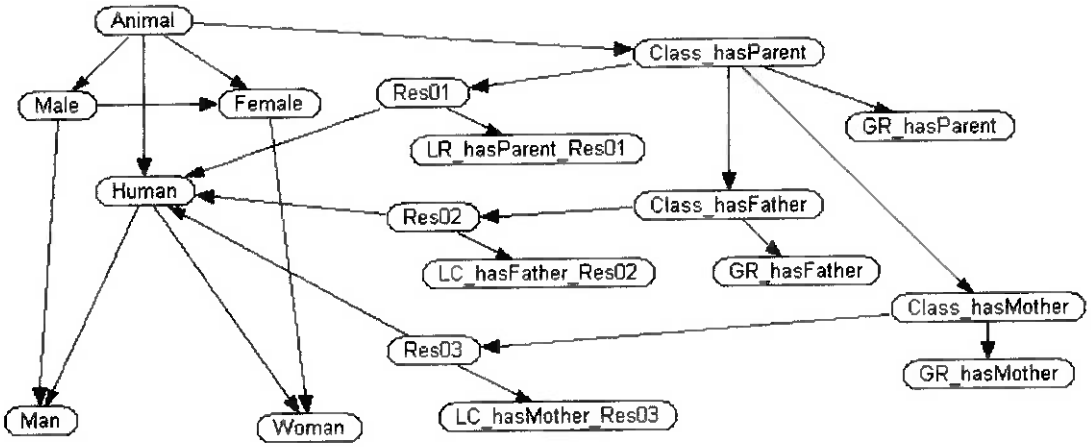

Figura 7.9 Passo 9

8. EXEMPLOS DE CONSTRUÇÃO DE TABELAS

Alguns dados para as tabelas não precisam ser declarados explicitamente, como nos exemplos abaixo (ainda no contexto do exemplo usado acima) que já se pode concluí-los a partir de suas definições:

Tabela 8.1 GR_hasParent

Class_hasParent	Animal	Irrelevant
True	1	0
False	0	1

Tabela 8.2 LR_hasParent_Res

Res	Human	Other
True	1	0
False	0	1

Tabela 8.3 LC_hasParent_Res

Res	N2	Nother
True	1	0
False	0	1

E em outros casos seus dados probabilísticos devem ser declarados como indivíduos das classes de dados probabilísticos, previamente citada no item 3.1.

```
<prob:PriorProbObj rdf:ID="P(Animal)">
<prob:hasVariable><rdf:value>&ont;Animal</rdf:value></prob:hasVariable>
  <prob:hasProbValue>0.5</prob:hasProbValue>
</prob:PriorProbObj>
```

Tabela 8.4 Animal

	Animal
True	0.5
False	0.5

```
<prob:CondProbObjT rdf:ID="P(Male|Animal)">
<prob:hasCondition><rdf:value>&ont;Animal</rdf:value></prob:hasCndtion>
  <prob:hasVariable><rdf:value>&ont;Male</rdf:value></prob:hasVariable>
  <prob:hasProbValue>0.5</prob:hasProbValue>
</prob:CondProbObjT>
```

```
<prob:CondProbObjT rdf:ID="P(Female|Animal)">
<prob:hasCondition><rdf:value>&ont;Animal</rdf:value></prob:hasCndtion>
<prob:hasVariable><rdf:value>&ont;Female</rdf:value></prob:hasVariable>
  <prob:hasProbValue>0.48</prob:hasProbValue>
</prob:CondProbObjT>
```

1. P(Female)=P(Female/Animal)* P(Animal)=0.48*0.5= 0.24
2. P(Male)=P(Male/Animal) * P(Animal)=0.5*0.5=0.25
3. P(Female/¬Male)=P(Female)/[1-P(Male)]=0.24/(1-0.25)=0.32
4. P(Male/¬Female)=P(Male)/[1-P(Female)]=0.25/(1-0.24)=0.33
5. P(Female/Animal,¬Male)=P(Female)/[P(¬Male)/Animal)*P(Animal)]= 0.24/(0.5*0.5)=0.96
6. P(Male/Animal,¬Female)=P(Male)/[P(¬Female)/Animal)*P(Animal)]= 0.25/(0.52*0.5)=0.96

Tabela para Female:

Tabela 8.5 Female

Animal	Male	True	False
true	true	0	1
true	false	0.96	0.04
false	true	0	1
false	false	0	1

Tabela Para Male:

Tabela 8.6 Male

Animal	Female	True	False
true	true	0	1
true	false	0.96	0.04
false	true	0	1
false	false	0	1

Como se pode notar acima, o cálculo da tabela de probabilidades para um nó com dois pais é complicado, e esta complicação tende a aumentar exponencialmente com o número de pais. Na seção seguinte será discutido um método para diminuir tal complicação.

9. NOISY OR

A criação das tabelas de probabilidades de variáveis com múltiplos pais é custosa e envolve muitos cálculos, afinal para uma variável com n pais, necessita-se calcular 2^n valores de probabilidades condicionais a partir das probabilidades condicionais simples (P(A/B) e P(A/¬B)). Para evitar estes cálculos, decidiu-se aplicar o método Noisy or descrito a seguir.

9.1 Alterações na Estrutura de Nós.

Toma-se, por exemplo, uma variável X com n pais (Y_1, Y_2,..., Y_n), como mostrado abaixo.

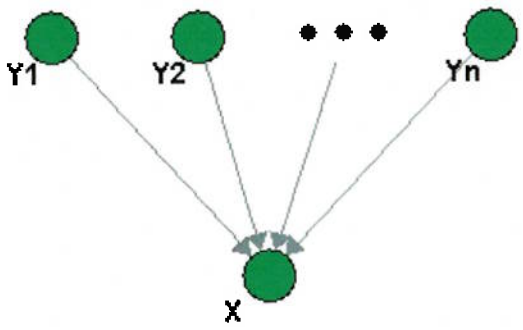

Figura 9.1 Nó com múltiplos pais

Para cada pai Y_i serão criados dois nós auxiliares, os nós $I\_Y_i\_X$ e $E\_X\_Y_i$, o arco que liga Y_i a X será retirado e serão criados arcos de $E\_X\_Y_i$ para X, de $I\_Y_i\_X$ para $E\_X\_Y_i$ e Yi para $E\_X\_Y_i$, gerando a seguinte configuração:

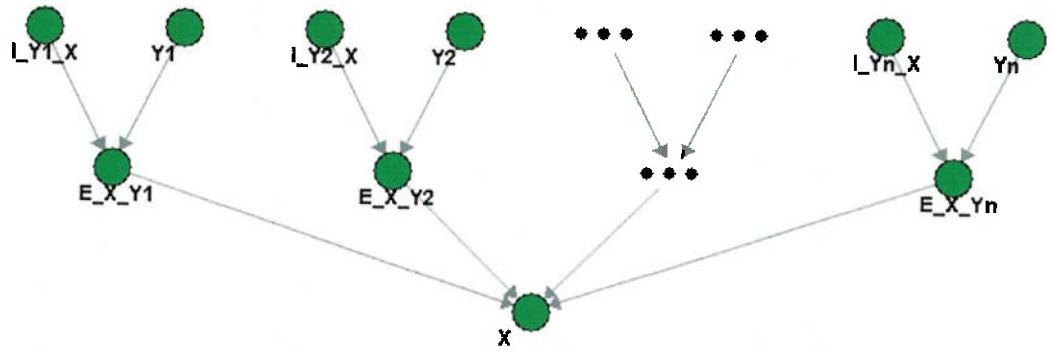

Figura 9.2 Rede após a aplicação de Noisy OR

O nó $E\_X\_Y_i$ possui a função de realizar a operação lógica "E" entre os nós Y_i e $I\_Y_i\_X$, logo sua tabela de probabilidades é bem simples como mostrada abaixo:

Tabela 9.1 Tabela operação lógica "E"

Yi=True		E_X_Yi	
		True	False
I_Yi_X	True	1	0
	False	0	1

Yi=False		E_X_Yi	
		True	False
I_Yi_X	True	0	1
	False	0	1

E o nó $I\_Y_i\_X$ é conhecido como nó da variável inibidora de Y_i, ou seja, em caso de $I\_Y_i\_X$ ser falso, $E\_X\_Y_i$ também será falso.

9.2 Probabilidades da Variável Inibidora

Necessita-se definir a tabela de probabilidades do nó da variável inibidora, para tal, segue-se o cálculo apresentado por [4]. Sabe-se que:

$$p_i = P(X / Y_i somente) = P(X / \bar{D}_1, \bar{D}_2, ..., \bar{D}_{i-1}, D_1, \bar{D}_{i+1}, ..., \bar{D}_n)$$

Considerando p_{all} a probabilidade de qualquer combinação dos fatores que resulte em F, obtemos o seguinte sistema de equações:

$$\begin{cases} P(F/D_i) = p_i + p_{all} - p_i \cdot p_{all} \\ P(F/\bar{D}_i) = p_{all} \end{cases}$$

Onde $P(F/D_i)$ e $P(F/\bar{D}_i)$ são probabilidades conhecidas, previamente fornecidas pelo modelamento proposto por Ding & Peng [2]. Substituindo-se o valor de p_{all} na primeira equação obtém-se:

$$P(F/D_i) = p_i + P(F/\bar{D}_i) - p_i \cdot P(F/\bar{D}_i)$$

Isolando-se o termo p_i:

$$p_i = \frac{P(F/D) - P(F/\bar{D}_i)}{1 - P(F/\bar{D}_i)}$$

10. CASO EXEMPLO DE PARAFUSOS

Na Figura 10.1, pode se ver a tela inicial de classes do software protege que foi utilizado ara o modelamento de parafusos. Este modelamento foi baseado na classificação apresentada por [5].

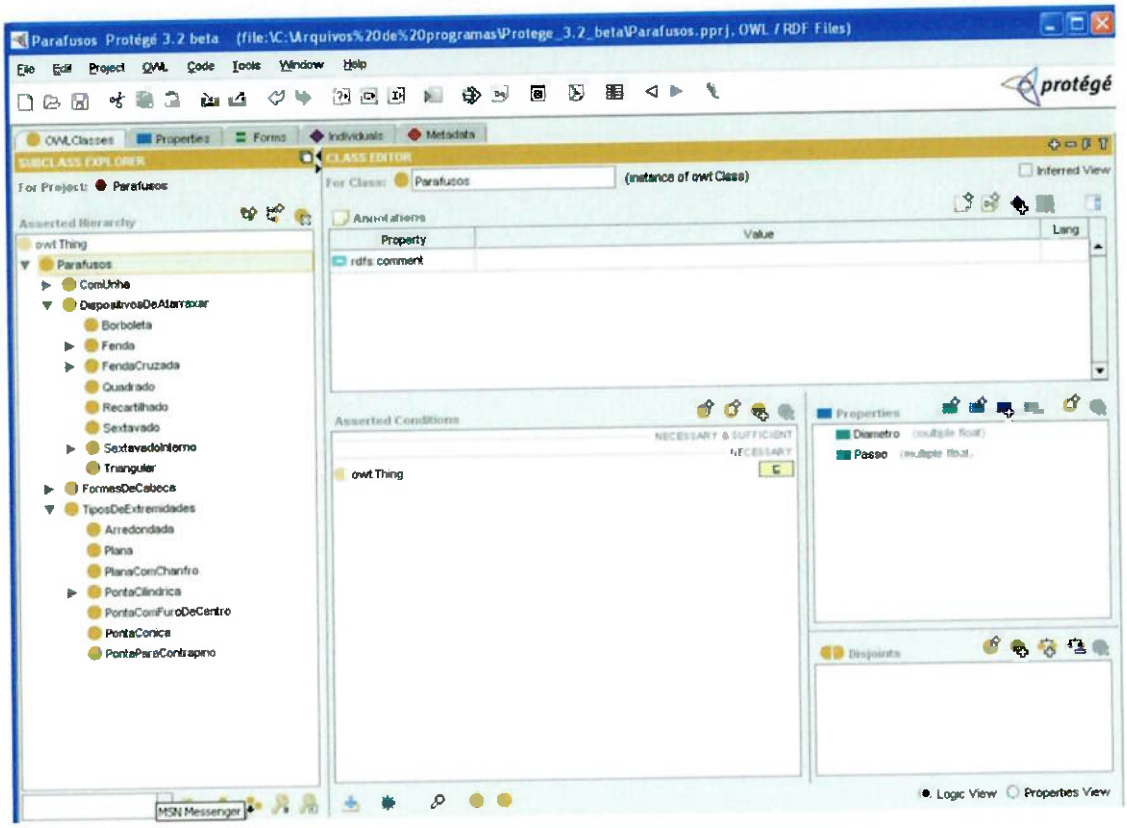

Figura 10.1 Interface Classes do Protégé com Parafusos

O modelo de classes parafusos foi baseado nas classificações fornecidas por [5]. Primeiro definiu-se classes de características de parafusos, separadas em três grupos (classes), representando dispositivos de atarraxar (DispositivosDeAtarraxar), formas de cabeça (FormaDeCabeça) e tipos de extremidade (TiposDeExtremidades).

A classe DispositivosDeAtarraxar está dividida nas seguintes subclasses:

- Borboleta;
- Fenda;
- FendaCruzada;
- Quadrado;
- Recartilhado;
- Sextavado;
- SextavadoInterno;
- Triangular.

A classe FormaDeCabeça possui as seguintes subclasses:

- Abaulada;
- AbauladaComPescoçoQuadrado;
- Cilíndrica;
- CilíndricoAbaulada;
- Escarreada;
- EscarreadoAbaulada;
- Quadrada;
- QuadradaComRessalto;
- Redonda;
- Retangular;
- Sextavada;
- SextavadaComRebaixo;
- SextavadaComRessalto;
- Triangular.

E por fim, a classe TipoDeExtremidades possui as seguintes divisões:

- Arredondada;
- Plana;
- PlanaComChanfro;
- PontaCilindrica;
- PontaComFuroDeCentro;

- PontaConica;
- PontaParaContrapino.

Pode-se notar esta estrutura de classes na figura abaixo:

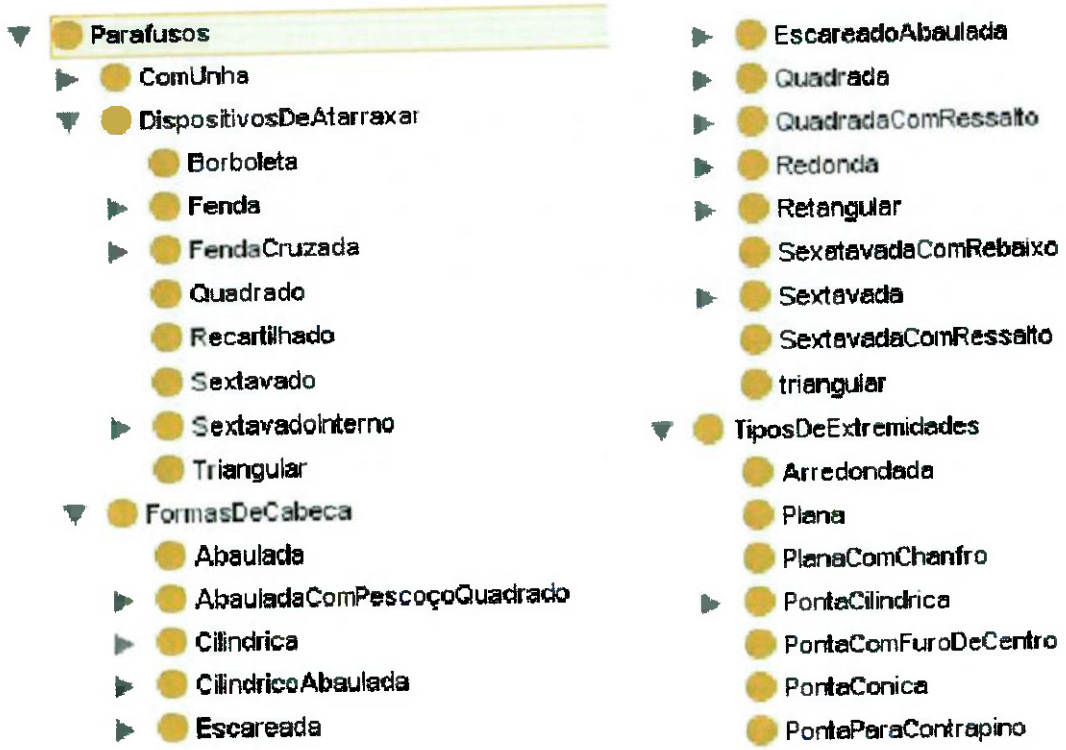

Figura 10.2 Classes de Características de Parafusos

Além dessas classes foram criadas classes que representam grupos de parafusos, que são fruto da intersecção de algumas das classes de classificação descritas acima. Um exemplo de grupo de parafusos é dado na figura a seguir, onde parafusos do tipo 72 são parafusos de cabeça escarreada e atarraxados por fenda cruzada.

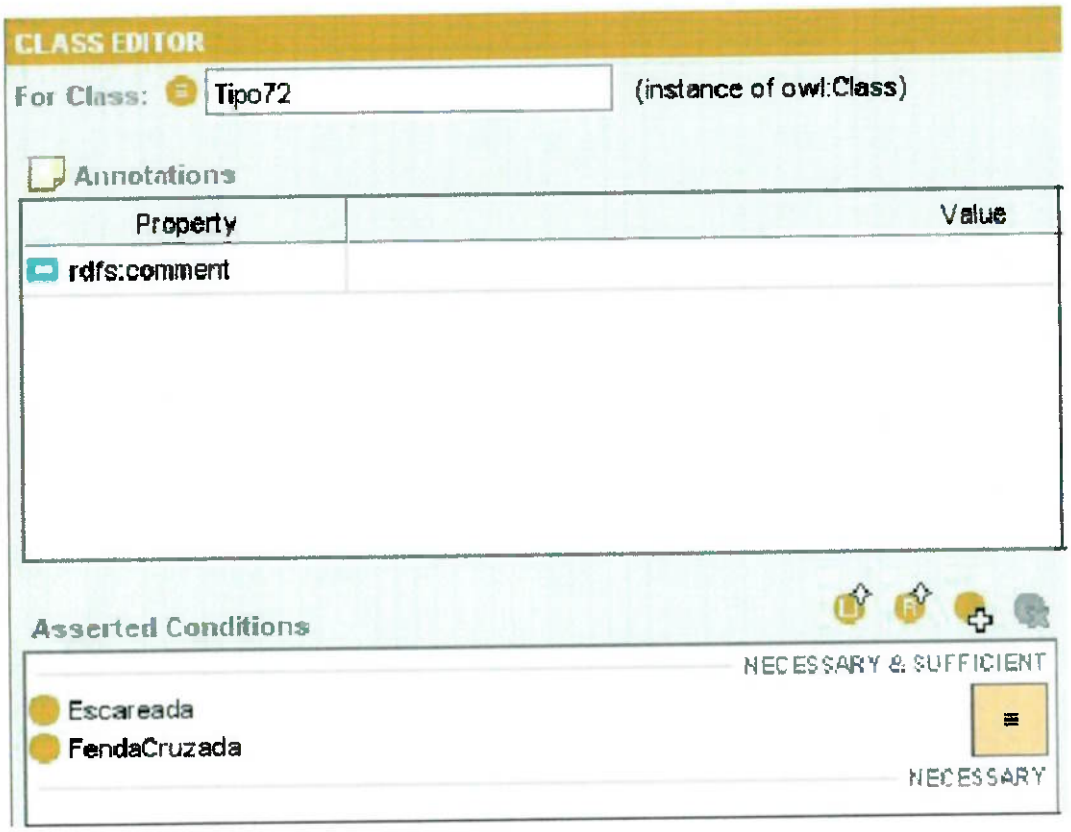

Figura 10.3 Classe de Tipos de Parafusos

Para a ontologia de parafusos foram criadas três propriedades; comprimento, diâmetro e passo. Pode-se notá-las na tela de propriedades do Protégé [8] na figura seguinte.

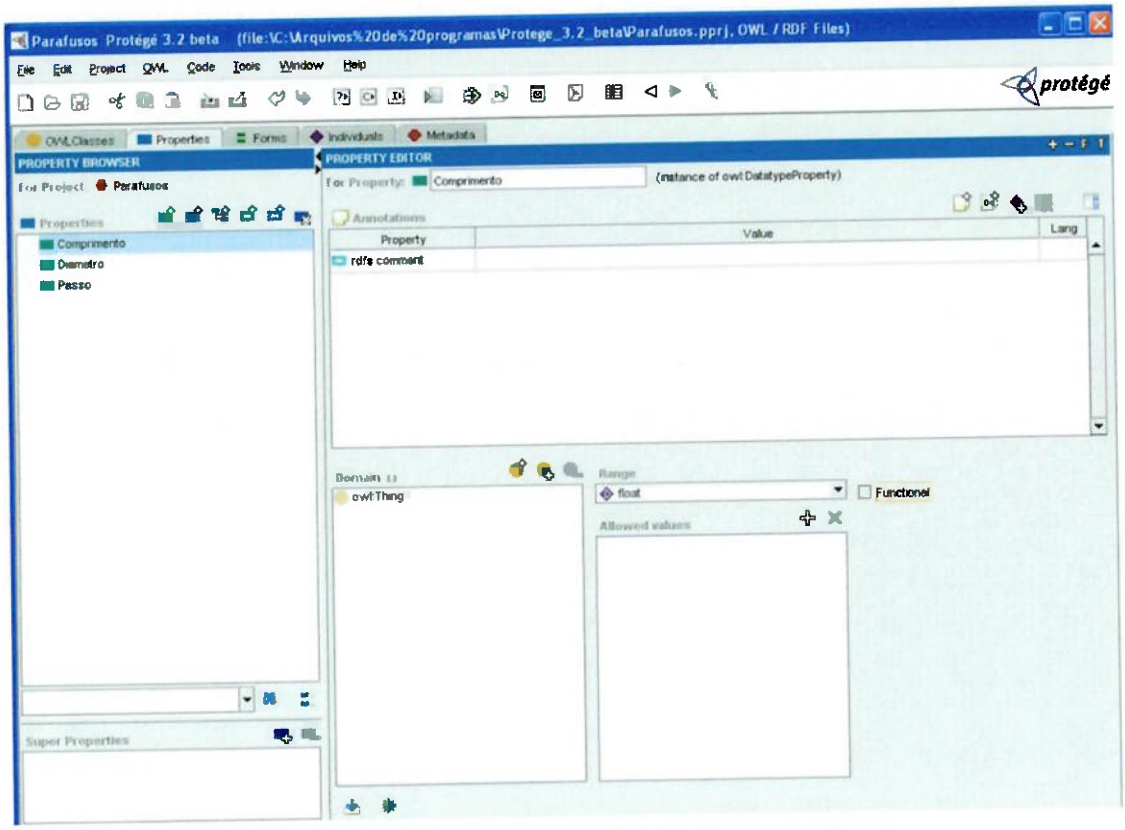

Figura 10.4 Interface de Propriedades do Protégé com Parafusos

11. CASO EXEMPLO DE ROLAMENTOS

Foi criado um caso exemplo de rolamentos para futuramente ser aplicado no programa de transformação de ontologia em uma rede bayesiana. Na Figura 11.1 pode-se notar a tela inicial de classes do Protégé [8] com a estrutura de classes criadas. Esta estrutura de classes foi baseada na classificação apresentada por [5].

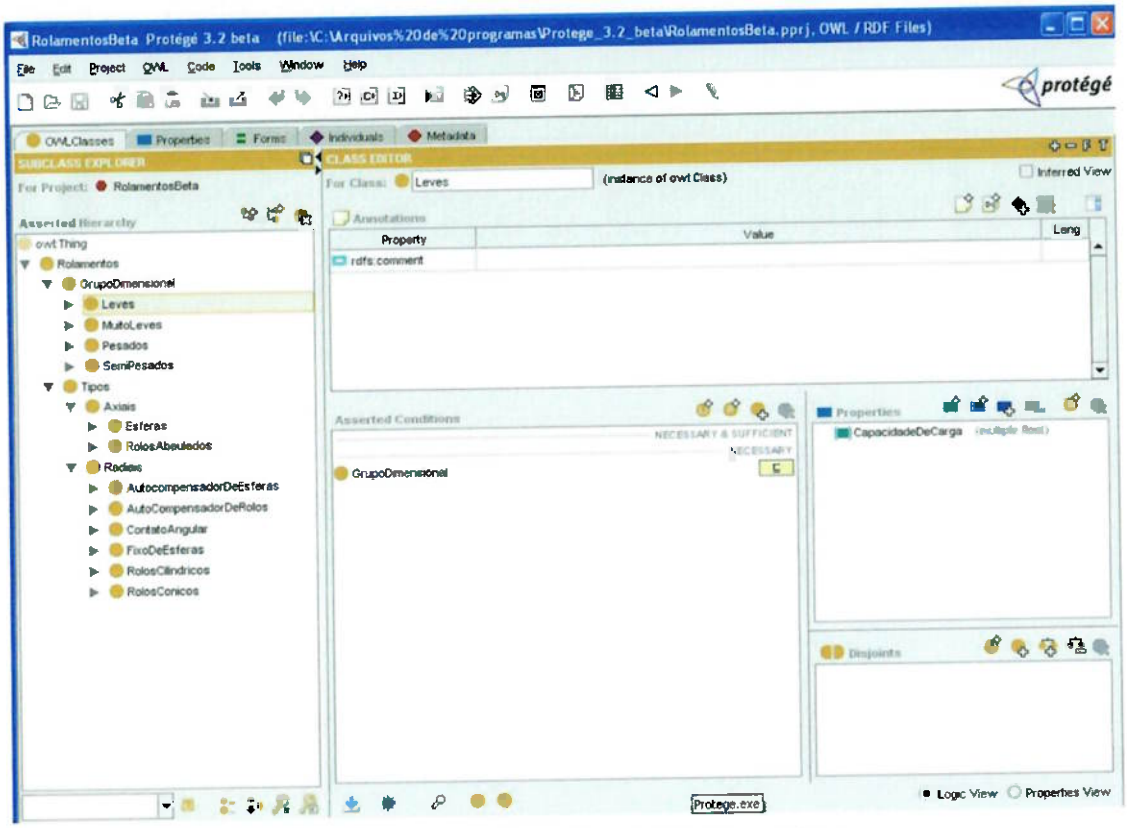

Figura 11.1 Interface de Classes Protégé do Caso de Rolamentos

Primeiro criaram se classes que representam características dos rolamentos. Há dois tipos de características, uma representada pela classe GrupoDimensional e outra chamada Tipos. Como existem quatro diferentes grupos dimensionais de parafusos, foram criadas subclasses para a classe GrupoDimensional as representando:

- Muito Leves;
- Leves;
- Semi Pesadas;
- Pesadas.

Os Tipos de rolamentos foram divididos em outras duas subclasses:

- Axiais;
- Radiais.

Que também possuem suas subclasses, no caso de Radiais as subclasses são as seguintes:

- Fixo de Esferas;
- Rolos Cilíndricos;
- Rolos Cônicos;
- Autocompensador de Esferas;
- Autocompensador de Rolos.

E para a classe Axiais:

- Esferas.
- Rolos Abaulados.

Pode-se notar esta estrutura de classes na figura seguinte.

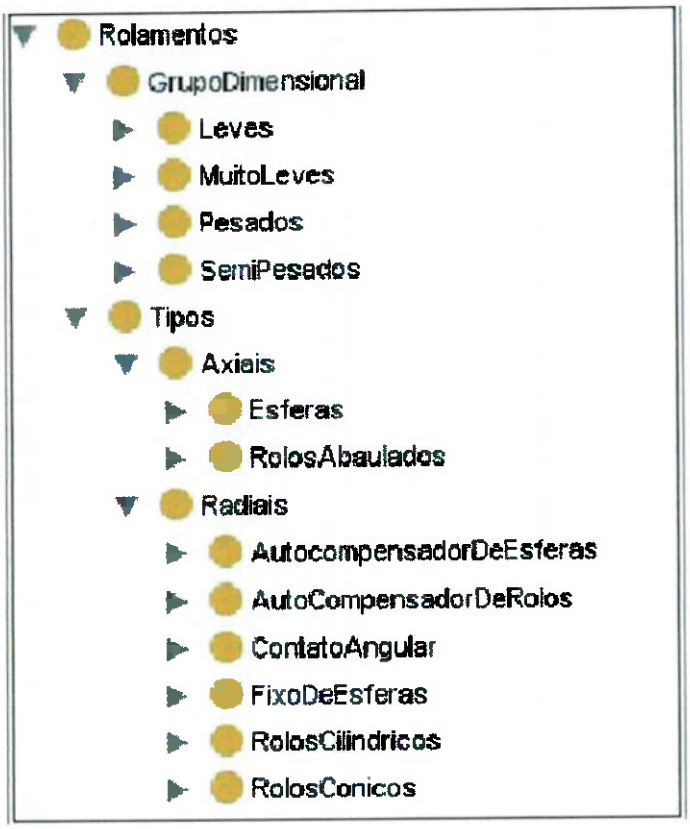

Figura 11.2 Estrutura de Classes de características de Rolamentos

Além destas classes, foram criadas classes que representam grupos de rolamentos, através da intersecção de características já definidas. Um exemplo dessas classes está na figura seguinte, onde Rol_14 representa rolamentos do grupo dimensional Leves e de tipo fixo de esferas.

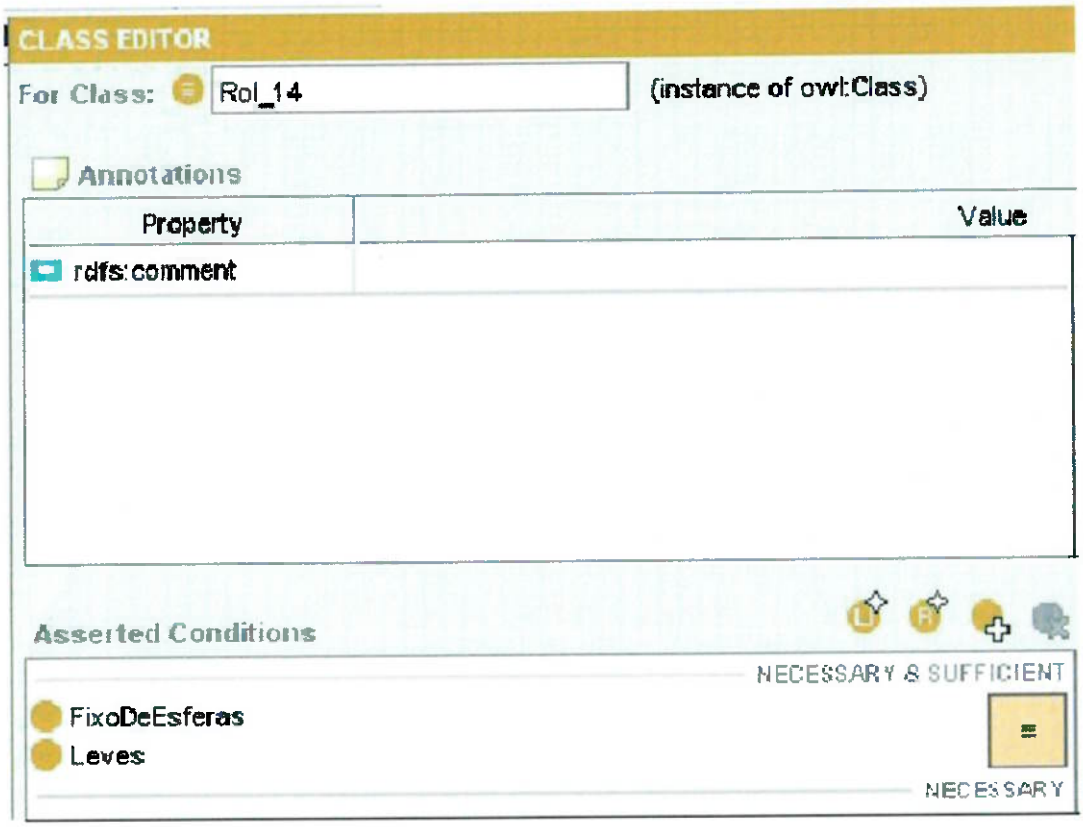

Figura 11.3 Classe representando um grupo de Rolamentos

E para esta ontologia foi criada apenas uma propriedade chamada de CapacidadeDeCarga, que pode ser vista na figura seguinte.

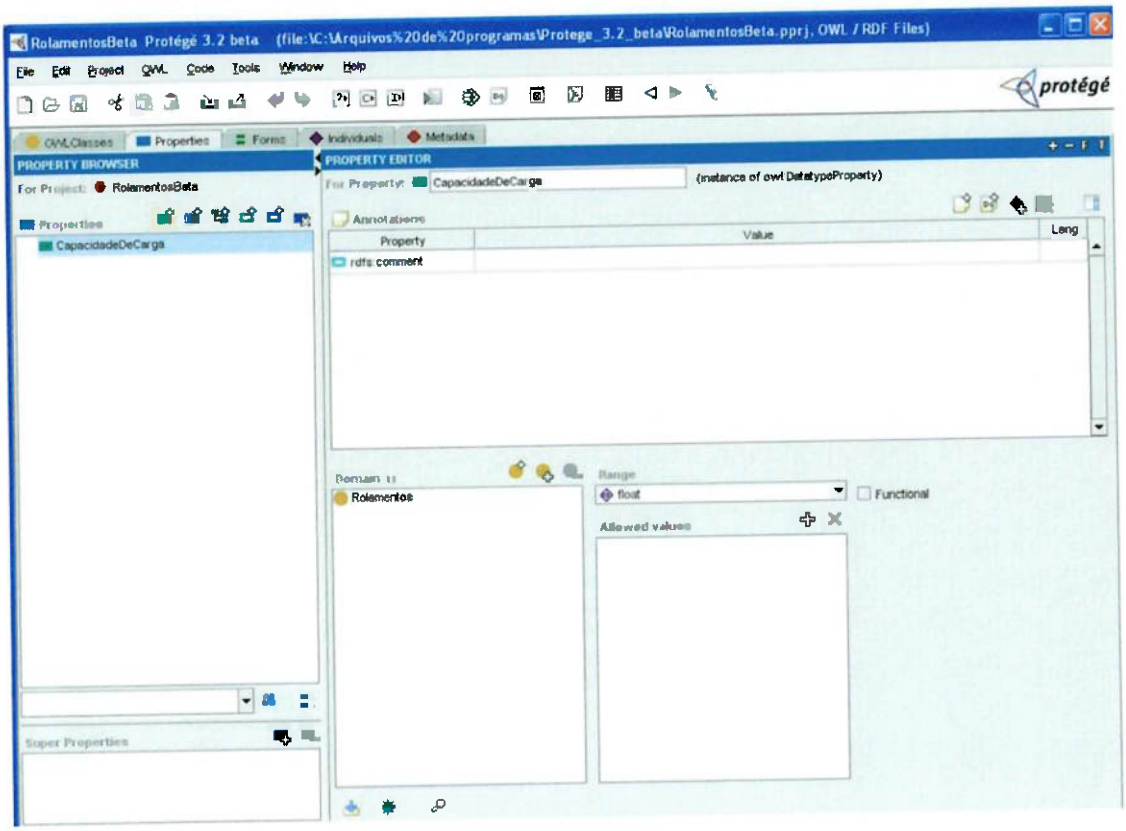

Figura 11.4 Propriedades Rolamentos

12. ESTRUTURA DO PROGRAMA

O programa que realiza a transformação de uma ontologia associada probabilidades condicionais em uma rede bayesiana foi escrito na linguagem C++.

Ele é composto de cinco partes principais, que são a leitura do arquivo com a ontologia, leitura do arquivo contendo as probabilidades, aplicação do método noisy or, associação das probabilidades as variáveis e no fim a criação de um arquivo e saída contendo a rede.

12.1 Leitura da Ontologia

O arquivo contendo a ontologia está no formato XML, então usarei como ferramenta para lê-los um pacote conhecido por TinyXml. Para armazenar o nome das variáveis lidas utilizaremos um vetor de tipo TIXML_STRING chamado Nome que armazenará o nome das variáveis, e uma matriz do tipo int chamada Parent que armazenará o número correspondente ao pai da variável.

12.2 Leitura das Probabilidades

As probabilidades também serão lidas a partir de um arquivo no formato XML. Utilizaremos três vetores de variáveis do tipo TIXML_STRING para armazenas os dados lidos, hasValue que armazenará o valor da probabilidade, hasVariable que armazenará o nome da variável em questão e hasCondition que armazenará o nome da variável condicional caso exista. Num passo seguinte, serão criados vetores do tipo int chamados VariableNo e ConditionNo que armazenaram o número correspondente ao nome destas variáveis, e também será criado um vetor do tipo double ValueNo que conterá o valor de hasValue convertido em double.

12.3 Aplicação do Método Noisy OR

Aplica-se o método Noisy OR para todas as variáveis com mais de um pai criando para cada nó pai os seguintes nós: I_NomePai_NomeFilho e E_NomeFilho_NomePai, usando o método explicado na seção 12.

12.4 Associação das Probabilidades às Variáveis

Neste estágio as probabilidades são associadas às suas respectivas variáveis e o calculo dos valores das variáveis inibidoras do método noisy or são calculados.

12.5 Criação da Saída

É criado um arquivo de saída do tipo BIF contendo a rede num formato a ser aberto pelo programa JavaBayes [11].

13. RESULTADOS

Abaixo é mostrada a rede resultante da transformação da ontologia de parafusos. Na parte superior notam-se as classes de características e na inferior as classes de tipos específicos de parafusos. E na Figura 13.2 pode-se ver a estrutura de classes de características ampliada.

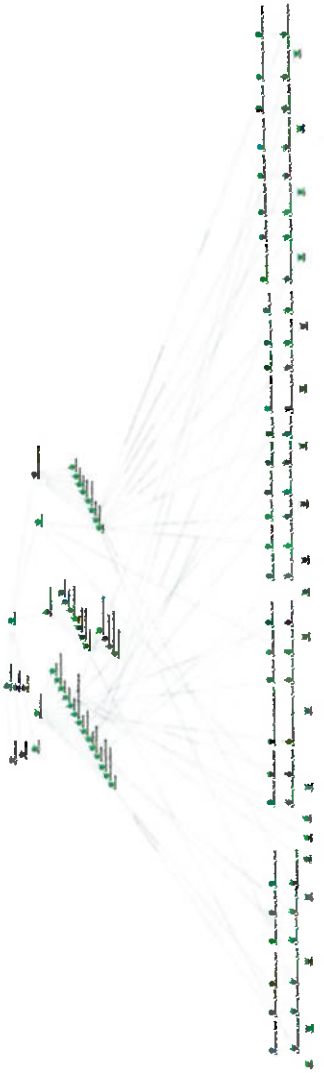

Figura 13.1Rede Bayesiana de Parafusos

Figura 13.2 Rede Bayesiana de características de parafusos

Abaixo é mostrada a rede resultante da transformação da ontologia de rolamentos. Na parte superior notam-se as classes de características e na inferior as classes de tipos específicos de rolamentos. E na Figura 13.4 pode-se ver a estrutura de classes de características ampliada.

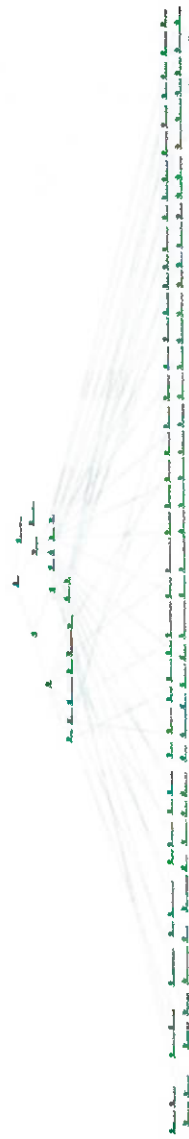

Figura 13.3 Rede Bayesiana de Rolamentos

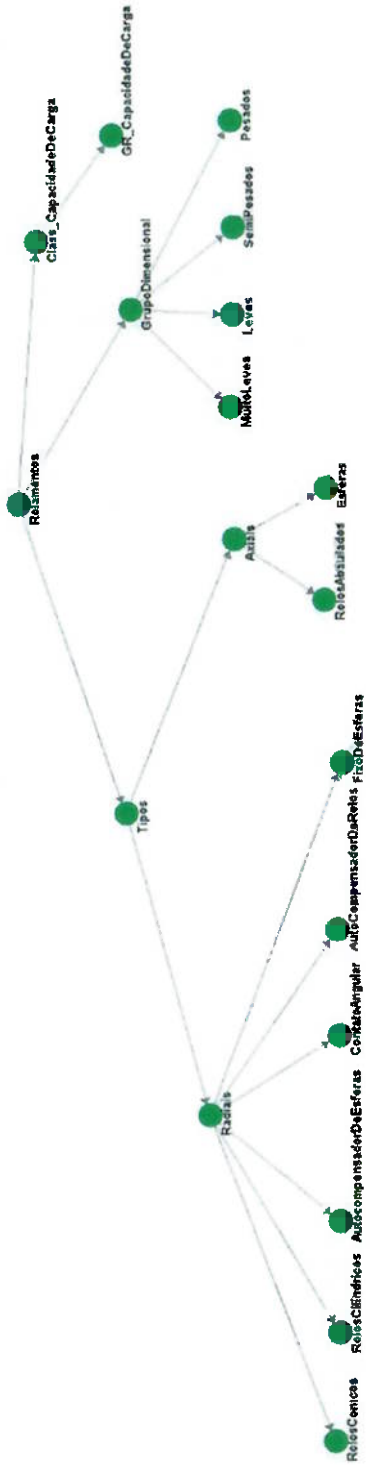

Figura 13.4 Rede Bayesiana de características de Rolamentos

14. CONCLUSÃO

Neste projeto foram desenvolvidas duas atividades principais. Uma delas foi a implementação do método de Ding & Peng [2] associada ao método Noisy Or e a outra foi a aplicação utilização destes métodos no ramo da manufatura.

A técnica de modelagem por ontologias e também em redes bayesianas já vem sendo usadas há muito tempo na área médica. Com este projeto, mostrou-se ser possível utilizá-lo também para classificação de peças na tecnologia de grupo [10 e 13], fornecendo mais expressividade e flexibilidade à classificação da tecnologia de grupo.

A implementação do método de Ding & Peng associada a Noisy OR obteve êxito, mas serviu para apontar uma falha no método. Por abordar apenas classes e propriedades, deixando os indivíduos de lado, o esforço para criar a ontologia é o mesmo que para criar a rede bayesiana, fazendo com que não haja nenhuma vantagem m utilizá-lo. Este método será de grande utilidade quando abranger indivíduos.

15. REFERÊNCIAS BIBLIOGRAFICAS

[1] F. Baader, W. Nutt. Basic Description Logics. In the Description Logic Handbook, edited by F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, P.F. Patel-Schneider, Cambridge University Press, 2002, pages 47-100.

[2] Ding, Z e Peng, Y. A probabilistic Extention to Ontology Language OWL, Proceedings of the 37th Hawaii International Conference on System Sciences-2004

[3] D. Nardi, R. J. Brachman. An Introduction to Description Logics. In the Description Logic Handbook, edited by F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, P.F. Patel-Schneider, Cambridge University Press, 2002, pages 5-44.

[4] Nikovski, D., Constructing Bayesian networks for medical diagnosis from incomplete and partially correct statistics. IEEE Transactions on Knowledge and Data Engineering, v.12, n.4, p.509–516, 2000

[5] Provenza, F. O projetista de máquinas- Editora f. Provenza

[6] Russel/Norvig, Inteligência Artificial –Editora Campus

[7] http://www.w3.org/TR/owl-features/ consultado em 11/04/2005

[8] http://protege.stanford.edu/plugins/owl/index.html consultado em 16/06/2005

[9]http://www.csee.umbc.edu/~zding1/HICSS37Supp/ consultado em 03/10/2005

[10]http://www.strategosinc.com/group_tecnology.htm consultado em 29/06/2005

[11] http://www.cs.cmu.edu/~javabayes/Home/ consultado em 19/03/1005

[12] http://www.daml.org/services/ consultado em 15/05/2005

[13]http://www.numa.org.br/conhecimentos/conhecimentos_port/pag_conhec/TG_Class_Produtos.htm consultado em 14/06/2005