Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Um Framework Baseado em Plug-ins para Raciocínio em Ontologias PR-OWL 2

Shou Matsumoto

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

Orientador
Prof. Dr. Marcelo Ladeira

Brasília
2011

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Mestrado em Informática

Banca examinadora composta por:

Prof. Dr. Marcelo Ladeira (Orientador) — CIC/UnB
Prof.ª Dr.ª Kate Cerqueira Revoredo — UNIRIO
Prof.ª Dr.ª Célia Ghedini Ralha — CIC/UnB

CIP — Catalogação Internacional na Publicação

Matsumoto, Shou.

Um Framework Baseado em Plug-ins para Raciocínio em Ontologias PR-OWL 2 / Shou Matsumoto. Brasília : UnB, 2011.
179 p. : il. ; 29,5 cm.

Dissertação (Mestrado) — Universidade de Brasília, Brasília, 2011.

1. Ontologia, 2. Incerteza, 3. Reuso, 4. Linha de Produtos de Software,
5. Plug-in. I. Ladeira, Marcelo. II. Título.

CDU 004.8

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília

Instituto de Ciências Exatas
Departamento de Ciência da Computação

Um Framework Baseado em Plug-ins para Raciocínio em Ontologias PR-OWL 2

Shou Matsumoto

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

Prof. Dr. Marcelo Ladeira (Orientador)
CIC/UnB

Prof.ª Dr.ª Kate Cerqueira Revoredo
UNIRIO

Prof.ª Dr.ª Célia Ghedini Ralha
CIC/UnB

Prof. Dr. Maurício Ayala Rincón
Coordenador do Programa de Pós-Graduação em Informática

Brasília, 1 de julho de 2011

Dedicatória

Dedico este trabalho à minha família, que foi exemplo em todas as áreas imagináveis, da seriedade e honestidade profissional ao respeito, afeto na convivência social. Sem a família, não teria obtido o ambiente, esforço e os valores necessários para realizar essa jornada, com dedicação.

Todo esse esforço realizado ao longo da jornada está começando a colher frutos não apenas com a conclusão deste trabalho, mas com oportunidades que têm surgido para continuação da pesquisa e com a contribuição científica, uma evolução que eventualmente poderá ser usufruída pela sociedade.

Espero que os resultados tenham sido motivos de alegria e satisfação para todos que participaram deste projeto. Que considero um momento especial para mim e, desejavelmente, para as equipes que trabalhei junto.

Agradecimentos

Agradeço ao Prof. Orientador Dr. Marcelo Ladeira, por me orientar durante esse ponto crucial da longa jornada acadêmica; sempre educando conscientemente os alunos, formando cidadãos responsáveis e preparados para o exercício profissional.

Gostaría também de agradecer ao Dr. Paulo Cesar G. da Costa, pela grande disponibilidade que tem nos oferecido para esclarecimentos e apoios durante o projeto. Essa atenção e o entusiasmo foram uma grande ajuda para esta pesquisa. Agradeço também ao doutorando Rommel Novaes Carvalho. Uma parcela significativa do meu trabalho tem como base seu projeto de doutorado e informações da CGU obtidas sob seu intermédio; portanto, sem ele este trabalho nem teria nascido. Agradeço a todos os amigos e familiares que certamente contribuíram para a conclusão desta pesquisa.

Por fim, esta pesquisa foi financiada pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (Fundação CAPES), na forma de bolsa de estudo para alunos de mestrado. Aproveito esta oportunidade para agradecer a CAPES por esta grande ajuda, que também foi crucial para a continuação da pesquisa.

Resumo

O volume crescente de informações disponibilizadas na Internet dificulta a localização da informação desejada, visto que os mecanismos de busca usuais são fortemente baseados em aspectos sintáticos. A Web Semântica e a OWL (*Web Ontology Language*) são tecnologias promissoras para o desenvolvimento de aplicações que permitam realizar buscas complexas na Internet. No entanto, ambas, como atualmente proposto, não consideram a incerteza, a qual é inerente à problemas do mundo real. Uma linha de pesquisa atual bastante ativa é a busca por propostas que ofereçam princípios sólidos e consistentes para raciocínio plausível na Web. Nesse contexto, *Probabilistic Web Ontology Language* (PR-OWL) (31, 34) emergiu como uma candidata para representação de ontologias probabilísticas na Web. PR-OWL é baseada no formalismo de MEBN (*Multi-Entity Bayesian Network*) (67) que é uma linguagem probabilística de primeira ordem. Apesar de sua alta expressividade, PR-OWL falha ao integrar consistentemente o conhecimento probabilístico com conhecimento determinístico oriundo de OWL, dificultando o reuso de informações "não probabilísticas". Visando solucionar tal problema, PR-OWL 2, uma versão de PR-OWL que permite mapear propriedades OWL com variáveis aleatórias, foi proposta na Universidade George Mason (19). Esta dissertação propõe a primeira implementação mundial da especificação PR-OWL 2. Essa implementação foi feita no framework UnBBayes (17, 22, 79, 81, 100), composta de interface gráfica, API e máquina de inferência, todas elas programadas em Java. Para facilitar o desenvolvimento de aplicações, o UnBBayes foi refatorado, migrando para a arquitetura de plug-ins, com o objetivo de se tornar uma linha de produtos de software com variabilidade resolvida em tempo de execução. Portanto, este trabalho contribui também para a área de Engenharia de Software como um exemplo de linha de produtos de software dinâmico no domínio de Inteligência Artificial.

Palavras-chave: Ontologia, Incerteza, Reuso, Linha de Produtos de Software, Plug-in

Abstract

The growing volume of information available on the Internet makes it difficult to locate desired information, because search engines rely basically on syntactic aspects. The Semantic Web and OWL (Web Ontology Language) are promising technologies for developing applications which perform complex searches on the Internet. However, such technologies, as currently proposed, do not consider the uncertainty inherent to real world problems. A very active and up-to-date field of research is to look for proposals that offer principled, consistent, and plausible reasoning on the Web. In such context, Probabilistic Ontology Web Language (PR-OWL) (31, 34) has emerged as a candidate for representing probabilistic ontologies on the Web. PR-OWL is based on MEBN (Multi-Entity Bayesian Network) formalism (67), which is a first-order probabilistic language. Despite its expressiveness, PR-OWL fails to consistently integrate the probabilistic knowledge with deterministic knowledge coming from OWL, making the reuse of "non-probabilistic" information on probabilistic ontologies very difficult. In order to solve such problem, PR-OWL 2, a new version of PR-OWL which allows us to create a mapping from random variables to OWL properties, was proposed at George Mason University (19). This work is intended to offer the first world-wide implementation of PR-OWL 2. This implementation was built on UnBBayes framework (17, 22, 79, 81, 100), offering a GUI, API, and a reasoner, all developed in Java. Additionally, in order to facilitate the development of new applications, the UnBBayes framework was refactored to use a plug-in design, in order to become a Software Product Line with runtime variability. Therefore, this work also contributes to the area of Software Engineering as an example of a Dynamic Software Product Line in the field of Artificial Intelligence.

Keywords: Ontology, Uncertainty, Reuse, Software Product Line, Plug-in

Sumário

1 Introdução **1**
- 1.1 Contexto e Definição do Problema . . . 1
- 1.2 Objetivos . . . 4
- 1.3 Abordagem da Solução Proposta . . . 4
  - 1.3.1 Metodologia de Pesquisa . . . 5
- 1.4 Áreas de Pesquisas Relacionadas . . . 6
  - 1.4.1 Nicho Atacado . . . 6
- 1.5 Contribuições . . . 6
- 1.6 Critérios de Conclusão da Pesquisa . . . 7
- 1.7 Estrutura da Dissertação . . . 9

2 Base Teórica em IA **11**
- 2.1 Visão Geral sobre BN . . . 11
- 2.2 A Lógica de Primeira Ordem - FOL . . . 16
- 2.3 Web Semântica e Ontologias . . . 18
  - 2.3.1 Ontologia - Definição . . . 19
  - 2.3.2 A Linguagem OWL . . . 20
  - 2.3.3 A Linguagem OWL, Versão 2 . . . 21
  - 2.3.4 Lidando com Múltiplas Ontologias . . . 22
- 2.4 Compreendendo a Importância de Incertezas em Ontologias . . . 23
  - 2.4.1 A Abordagem Probabilística . . . 24
- 2.5 MEBN - Redes Bayesianas Multi-Entidade . . . 25
  - 2.5.1 MFrags . . . 26
  - 2.5.2 Recursividade em MEBN . . . 28
  - 2.5.3 MTheories . . . 29
- 2.6 Uma Ontologia Probabilística - PR-OWL . . . 30
- 2.7 PR-OWL 2 - Integrando Conhecimento Probabilístico e Determinístico . . . 32
- 2.8 Outras Abordagens . . . 34
  - 2.8.1 Exemplos de Implementações Relacionadas . . . 35

3 Arquitetura da Solução Proposta **44**
- 3.1 Base Teórica em Engenharia de Software - ES . . . 44
  - 3.1.1 Visão Geral sobre SPL e FM . . . 45
  - 3.1.2 Resolvendo Variabilidade em Tempo de Execução . . . 46
- 3.2 O UnBBayes . . . 47
  - 3.2.1 Histórico . . . 48

3.2.2 Arquitetura do UnBBayes - Antes da Refatoração 49
3.2.3 ID e BN 50
3.2.4 MSBN 54
3.3 Ferramentas Utilizadas 55
3.3.1 JPF 55
3.3.2 OWL API e HermiT 56
3.3.3 Protégé 4.1 58
3.4 Arquitetura Baseada em Plug-ins 59
3.4.1 Arquitetura Geral do UnBBayes - Após Refatoração 62
3.4.2 Ponto de Extensão para Novos Módulos 67
3.4.3 Plug-ins para Plug-ins - Hiearquia de Depenências 70
3.4.4 Pontos de Extensão para o Módulo de BN 71
3.4.5 Plug-ins de Recursos de Nacionalização 74

4 Extração de *Features* como Plug-ins **77**
4.1 Visão Geral da Extração de *Features* 77
4.2 Plug-ins de I/O 79
4.3 Plug-ins de Aprendizagem Bayesiana 81
4.4 Plug-ins de Metáfora 82
4.5 Plug-ins para Amostragem e Algoritmo de Inferência 85
4.6 Plug-in de OOBN 85
4.7 Plug-in de MSBN 88
4.8 Plug-in de PRM 90

5 UnBBayes-MEBN e Plug-in para Suporte a PR-OWL 2 **92**
5.1 UnBBayes-MEBN Antes da Refatoração 92
5.1.1 MTheory 93
5.1.2 MFrag 94
5.1.3 Tabela de Probabilidades 96
5.1.4 GUI 97
5.1.5 Edição de MFrags e Conteúdos 98
5.1.6 Edição de Entidades e Evidências 98
5.1.7 Geração de SSBN 100
5.1.8 FOL no PowerLoom 100
5.1.9 Avaliação dos Nós de Contexto 102
5.1.10 Arquivos UBF 102
5.1.11 Implementação de I/O no UnBBayes-MEBN 103
5.2 Refatorando o UnBBayes-MEBN como um Plug-in 104
5.3 Arquitetura do Plug-in para Suporte a PR-OWL 2 107
5.3.1 Plug-in de Entrada e Saída em PR-OWL 2 108
5.3.2 Um Novo Tratamento para a Exclusividade Global 112
5.3.3 Plug-in para Base de Conhecimento em PR-OWL 2 112
5.3.4 Mecanismo de Integração da GUI do Protégé 4.1 115
5.3.5 Painel de Mapeamento de Propriedades OWL 117
5.4 Análise de Desempenho - Profiling 119

6 Resultados **122**
6.1 Especificando FM em OWL . . . 123
6.2 Cenário de Uso . . . 125
6.2.1 Procedimento . . . 126
6.2.2 Ferramenta Utilizada . . . 127
6.3 Análise . . . 127
6.3.1 Uma Visão Futura para Extensão Probabilística . . . 130
6.4 Impactos da Refatoração do UnBBayes e Implementação da PR-OWL 2 no Mundo . . . 130
6.4.1 A Utilização do UnBBayes no Mundo . . . 131
6.4.2 Outros Indicadores de Utilidade do UnBBayes . . . 133
6.4.3 A Utilização da PR-OWL 2 no Mundo . . . 134

7 Estudo de Caso: Detecção de Fraudes em Licitações Públicas **136**
7.1 O Domínio de Licitação Pública . . . 137
7.2 Fusão de Conhecimento Via Ontologias Probabilísticas . . . 142
7.3 Resultados e Análise . . . 146

8 Conclusões e Trabalhos Futuros **150**
8.1 Limitações e Trabalhos Futuros . . . 151

Referências **152**

Lista de Figuras

2.1 Figura exemplo de BN, adaptada de (25) 12
2.2 Exemplo para ilustrar limitações de BN 14
2.3 BN para o caso de identificação de veículo, com recursividade 15
2.4 Ilustração da tripla forma-conceito-referente. 20
2.5 Figura exemplo de MFrag 27
2.6 Modelagem de uma situação recursiva 29
2.7 Conjunto de MFrags que formam uma MTheory (78) 39
2.8 Principais conceitos da PR-OWL (34) 40
2.9 Elementos de uma ontologia probabilística PR-OWL (34) 40
2.10 Ontologia de licitação pública, em UML 41
2.11 Exemplo de modelo probabilístico de licitação pública 42
2.12 Mapeamento do nó residente para uma propriedade binária. 42
2.13 Mapeamento do nó residente para uma propriedade não funcional. 43
2.14 Exemplo de mapeamento para propriedades n-árias 43

3.1 Pacotes do UnBBayes antes da refatoração. 49
3.2 Modelagem de classes para BN e ID. 51
3.3 Fluxo de uso do módulo de BN do UnBBayes 52
3.4 Classes de GUI do UnBBayes antes da refatoração. 53
3.5 Modelagem de classes para MSBN. 54
3.6 Comparação entre o uso de JPF como framework e como biblioteca 57
3.7 Diferença entre pura orientação a objeto e orientação a componentes. 60
3.8 Típica estrutura de pastas do UnBBayes. 62
3.9 Visão MVC do UnBBayes 63
3.10 Diagrama UML do ponto de extensão Module 65
3.11 Diagrama UML dos pontos de extensão InferenceAlgorithm e PNIO 65
3.12 UML das extensões PluginNode e ProbabilityFunctionPanel 66
3.13 Classes de Visão e Controle 66
3.14 Captura de tela de alguns plug-ins de módulos do UnBBayes 68
3.15 Diagrama do ponto de extensão para módulos 69
3.16 Visão abstrata da ligação de plug-ins para plug-ins. 70
3.17 Captura de tela de um plug-in para plug-in. 71
3.18 Pontos de extensão para funcionalidades de BN 72
3.19 Diagrama de classes de pontos de extensão para algoritmos de inferência e I/O . 73
3.20 Classes que devem ser estendidas para se criar um plug-in para novos tipos de nós 73

4.1 Famílias do UnBBayes antes da refatoração 78

4.2 Famílias do UnBBayes após refatoração 79
4.3 Projeto maven para plug-ins do UnBBayes 80
4.4 Diagrama de classes dos plug-ins dos módulos de aprendizagem 81
4.5 Sequência de instanciação dos plug-ins de aprendizagem. 82
4.6 Captura de tela da metáfora 83
4.7 Diagrama de classes da metáfora. 84
4.8 Diagrama de classes do plug-in da metáfora. 85
4.9 Diagrama de classes dos plug-ins de amostragem. 86
4.10 Diagrama de classes dos plug-ins de algoritmo de inferência. 86
4.11 Classes implementando a OOBN antes da refatoração 87
4.12 Refatoração de OOBN como plug-in 89
4.13 Refatoração de MSBN como plug-in. 89
4.14 Captura de tela do plug-in de PRM. 90
4.15 Arquitetura de camadas do plug-in de PRM. 91
4.16 Diagrama de classes da GUI do PRM. 91

5.1 Modelagem de classes para MTheory 93
5.2 Modelagem de classes para MFrag. 94
5.3 Modelagem de classes para os nós de uma MEBN. 95
5.4 Arquitetura da GUI 97
5.5 Janela de edição da CPT 98
5.6 Edição de entidades 99
5.7 Edição de evidências 99
5.8 Painel da rede compilada 100
5.9 MFrag `Speed`, com seus nós de contexto em formato de pentágono. 101
5.10 Modelagem de IO de MEBN 103
5.11 Resultado da criação da classe MEBNNetworkWindow. 105
5.12 Diagrama UML para classes de I/O 109
5.13 Bridge pattern para objetos de Protégé ou OWL API 111
5.14 Diagrama que relaciona classes de I/O com objetos do Protégé ou OWL API . . 111
5.15 Exemplo de exclusividade global 112
5.16 KB para OWL2 113
5.17 Diagrama UML do plug-in para base de conhecimento em OWL2 113
5.18 Diagrama UML do plug-in de edição da ontologia determinística 115
5.19 Organização de classloaders JPF e OSGi 116
5.20 Diagrama UML do painel de relacionamento 118
5.21 Laudo de tempo e espaço do plug-in de PR-OWL 2 121

6.1 Diagrama UML do FM 124
6.2 Esquema de tradução de FM a OWL. 125
6.3 Esquema de comunicação entre uma DSPL e um repositório OWL 2. 127
6.4 Diagrama de classes do plug-in de sugestão de *features* 128
6.5 Fácil extensão de ontologia OWL 2 para PR-OWL 2 131

7.1 Visão geral do ciclo de detecção/prevenção de fraudes em licitações públicas. . 138
7.2 Principais classes e propriedades OWL do domínio “global”. 140
7.3 MFrag de requisitos. 140

7.4 MFrag de direcionamento de licitações por índices contábeis. 141
7.5 MFrag de direcionamento de licitações. 141
7.6 Esquema de fusão de conhecimento oriundo de diferentes órgãos do governo. . 142
7.7 MFrag associado a informações sobre empresas. 144
7.8 MFrag associado a informações sobre participação em licitações. 144
7.9 MFrag associado a informações sobre esquemas de fachada. 144
7.10 MFrag sobre informações pessoais e renda. 145
7.11 MFrag sobre existência de fachada em uma empresa. 145
7.12 MFrag indicando se a licitação é suspeita. 146
7.13 SSBN gerada consultando-se `isDirected(procurement1)`. 147
7.14 SSBN gerada consultando-se `isSuspicious(procurement1)`. 148

Lista de Tabelas

2.1 CPT do nó “Cachorro fora de casa”. 12
2.2 CPT de *ObjectType*. 15

5.1 `BuiltInRV` implementadas no UnBBayes. 94

Glossário

API *Application Program Interface*.

BAN *Bayesian Network Augmented Naive Bayes*.

BN *Bayesian Networks*, ou redes bayesianas.

Booleano É um tipo de dado bivalorado (verdadeiro ou falso). No contexto de PR-OWL/MEBN, consideraremos também o valor "absurdo" (nem verdadeiro nem falso) como booleano.

Chain of responsibility Padrão de projeto (*design pattern*) que consiste de uma série de objetos de processamento implementando uma lógica que descreve os tipos de operações suportadas e como realizar o repasse de responsabilidades na cadeia (*chain*) quando uma requisição não é suportada (49).

Carregador de classes Java O carregador de classes Java (*Java class loader*) é um objeto responsável por localizar e gerar dados que constituem a definição de uma classe. O comportamento deste objeto determina diretamente a forma em que a máquina virtual Java carrega classes dinamicamente.

CBR *Case Based Reasoning*, ou raciocínio baseado em casos.

CGU Controladoria-Geral da União, órgão do Governo Federal responsável à defesa do patrimônio público e ao incremento da transparência da gestão.

CK *Configuration Knowledge* - conjunto de informações necessárias para se mapear uma funcionalidade abstrata (*e.g. feature*) a um artefato de computação. Associa um conjunto de requisitos a um componente de software.

CPT *Conditional Probability Table*, ou Tabela de Probabilidades Condicionais.

Design patterns Conjunto de abordagens genéricas que têm como objetivo evitar problemas comuns e frequentes no desenvolvimento de *softwares* (49).

DBN *Dynamic Bayesian Network*, ou Rede Bayesiana Dinâmica (86).

DENATRAN Departamento Nacional de Trânsito.

DL *Description Logic*, ou lógica descritiva.

DSPL *Dynamic Software Product Line*, ou linha de produtos de *software* dinâmico.

Eclipse IDE mantido pela fundação Eclipse - IBM (48).

Encapsulamento Mecanismo utilizado para se esconder detalhes de informações, permitindo que impactos causados por alterações em uma parte do programa sejam limitados a uma pequena porção do programa.

Entidade Refere-se a qualquer conceito (real ou fictício, concreto ou abstrato) que pode ser descrito e fundamentado em um domínio de aplicação.

ES Engenharia de *Software*.

Feature Característica. No âmbito de SPL, denota um conjunto de funcionalidades ou abstração de requisitos de *software*. O termo "característica" é evitado neste documento, a fim de distinguir o sentido em SPL do sentido geral (particularidade ou propriedade).

FM *Feature Model* (104), ou modelo de *features* - modelo que descreve as *features* (um conjunto de funcionalidades ou abstração de requisitos) em uma linha de produtos de *software*.

FOL *First Order Logic*, ou lógica de primeira ordem.

GIA/UnB **G**rupo de **I**nteligência **A**rtificial / UnB.

GMU A *George Mason University* (GMU) é uma universidade pública situada em *Virginia*, EUA.

GPL GNU *General Public License*.

GUI *Graphical User Interface*.

HTTP *Hyper Text Transfer Protocol*, um protocolo de transferência de dados via rede de computadores.

HUGIN Ferramenta comercial avançada para suporte a decisões, baseada em BN e ID. Site: <http://www.hugin.com/>).

IA Inteligência Artificial.

ID *Influence Diagram*, ou diagramas de influência.

IDE *Integrated Development Environment* ou Ambiente Integrado de Desenvolvimento são *softwares* que provêm um conjunto de funcionalidades de apoio ao desenvolvimento de sistemas (*softwares*).

Indivíduos No âmbito de ontologias OWL, instâncias de classes são chamadas de indivíduos.

jCOLIBRI Framework de código aberto para CBR, mantido pelo Grupo de Aplicações de Inteligência Artificial (*Group for Artificial Intelligence Applications* - GAIA), Dpt. de Engenharia de Softwares e Inteligência Artificial da *Universidad Complutense de Madrid*.

JPF *Java Plugin Framework*, um framework em Java para prover uma infra-estrutura de plug-ins. Página da JPF: <http://jpf.sourceforge.net/>

JPF *Java Plugin Framework*.

LGPL *GNU Lesser General Public License*.

Maven Ferramenta da Apache para auxílio de gerenciamento de projetos de *software* durante diversas etapas de desenvolvimento. Possui ferramentas para auxílio no carregamento de bibliotecas, geração de build, geração de javadoc, implantação, etc. Página principal: <http://maven.apache.org/>.

MDI *Multiple Document Interface* é tipicamente utilizada em ambientes Microsoft Windows™ e representa uma interface gráfica com várias janelas internas contidas em uma única janela pai.

MEBN *Multi-Entity Bayesian Network*, ou rede bayesiana multi-entidade.

MEC Ministério da Educação.

Metáfora No contexto do UnBBayes, a metáfora é uma interface gráfica mais simples que esconde do usuário alguns detalhes do modelo BN, mas permite inserção de evidências e verificação dos resultados da inferência.

MSBN *Multiply Sectioned Bayesian Network*, ou Redes Bayesianas Múltiplas Secionadas.

MTI *Moving Target Indicator*, ou Indicador de Alvo Móvel, é um tipo de sensor (normalmente baseado em radar) usado para detectar objetos em movimento.

MVC Uma arquitetura MVC separa a lógica e os dados da interface gráfica, criando-se então três categorias independentes: *Model* (dados e operações básicas sobre elas), *View* (interface com o usuário) e *Controller* (normalmente funciona como um mediator, escalonador, ou moderador de outras classes) (11).

Ontologia Especificação formal de conceitos e relacionamentos em um domínio.

OOBN *Object-Oriented Bayesian Network*, ou rede bayesiana orientada a objeto.

OSGi *Open Service Gateway Initiative*™, *framework* voltado à implementação e implantação de programas e bibliotecas em Java. Oferece também uma infra-estrutura de plug-ins. Página: <http://www.osgi.org/>.

OWL *Web Ontology Language*, linguagem recomendada pela W3C para ontologias.

Plug-in Nesta dissertação, usaremos o termo "plug-in" no mesmo sentido do apresentado em (90). Ou seja, plug-ins são artefatos de programação que são resolvidos em tempo de execução, implementam alguma funcionalidade esperada pelo programa principal e interage com uma aplicação hóspede (*core*) para prover uma funcionalidade específica.

Ponto de extensão Termo bastante utilizado na ES para indicar uma porção do programa que pode ser alterada (estendida) para se adicionar novas funcionalidades.

PR-OWL *Probabilistic Web Ontology Language* - extensão probabilística da OWL.

PRM *Probabilistic Relational Model* - formalismo para permitir o tratamento de incertezas em dados relacionais.

Propriedade funcional Uma propriedade é funcional se este representa uma função. Ou seja, cada elemento do domínio está associada a somente 1 elemento da imagem.

Protégé *Software* de código aberto com ferramentas para se modelar domínios de aplicações baseadas em ontologias (87), mantido pela *Stanford Center for Biomedical Informatics Research*, da Universidade de Stanford, Faculdade de Medicina. Site: <http://protege.stanford.edu/>).

RDF *Resource Description Framework*.

RDFS *Resource Description Framework Schema*.

Reusabilidade Qualidade de um sistema (ou do seu código fonte) de poder ser usado novamente, com modificações sutis ou sem modificação alguma.

RV *Random Variable* ou variável aleatória denota um conjunto de atributos, proposições ou hipóteses incertas.

SourceForge Sítio que hospeda softwares livres de código aberto. Página principal: <http://sourceforge.net/>

SPL *Software Product Line* (108), ou linhas de produtos de *software* - conjunto de sistemas de software que têm um determinado conjunto de funcionalidades em comum, e que satisfazem as necessidades de uma determinada missão, e que são desenvolvidos tendo a mesma base (*core*).

SSBN *Situation-Specific Bayesian Network*, ou rede bayesiana de situação específica.

Swing *Framework* gráfico no Java usado pela GUI do UnBBayes. Veja <http://download.oracle.com/javase/tutorial/uiswing/> para maiores detalhes.

Tag Metadado que consiste de um termo atribuído a informações.

TAN *Tree Augmented Naive Bayes*.

TI Tecnologia da Informação.

UFRGS Universidade Federal do Rio Grande do Sul.

UML *Unified Modeling Language* (89), ou linguagem de modelagem unificada.

UnB Universidade de Brasília.

UnBBayes Framework de código aberto, escrito em linguagem Java, que oferece um ferramental gráfico para modelagem de conhecimento probabilístico. Disponível em <http://unbbayes.sourceforge.net>.

Variabilidade Qualidade de um sistema (ou do seu código fonte) de poder ser alterado com menor esforço.

W3C *World Wide Web Consortium*, consórcio gestor da *World Wide Web*.

XML *Extensible Marktup Language*.

Capítulo 1

Introdução

Nos últimos anos, a forma em que os softwares são desenvolvidos no mundo acadêmico e nas pesquisas têm mudado drasticamente. Um dos principais causadores desse fenômeno é a crescente complexidade dos sistemas de IA, que enfrentam demandas cada vez mais exigentes do mundo real. Como requisitos de **usabilidade** (facilidade de uso), **produtividade** (eficiência e eficácia), **portabilidade** (compatibilidade multiplataforma) e **interoperabilidade** (capacidade de operar harmonicamente) tornaram-se uma preocupação central, a linguagem de programação Java™, com seu paradigma orientado a objeto, tornou uma opção natural em sistemas de IA.

Com a popularização da linguagem Java™, várias arquiteturas independentes de domínio, com suporte a raciocínio intensivo em bases de conhecimento (por exemplo, ontologias), foram desenvolvidas usando-se a referida linguagem (*e.g.* jCOLIBRI(50), Protégé. Outro exemplo é o **UnBBayes**.

Esta pesquisa se resume na criação de novas funcionalidades no UnBBayes para refletir o estado da arte em raciocínio plausível em ontologias na *Web Semântica*, através da implementação de PR-OWL versão 2 - uma linguagem que integra o conhecimento probabilístico e o determinístico.

Este capítulo convida então o leitor a entender a estrutura e especificação geral desta dissertação, sua relevância, as áreas de pesquisa relacionadas e uma visão da solução proposta. A Seção 1.1 introduzirá o problema a ser tratado e o contexto que o originou. A Seção 1.2 apresentará os objetivos desta pesquisa. A Seção 1.3 introduz uma visão geral da solução proposta aos problemas definidos na Seção 1.1. A Seção 1.4 exporá as áreas de pesquisa relacionadas; seguido pela Seção 1.5, que apresentará a importância do tema e a contribuição que a pesquisa oferece à comunidade. A Seção 1.6 apresenta um detalhamento dos critérios adotados para se considerar a pesquia como concluída, e finalmente a Seção 1.7 oferece referências aos demais capítulos desta dissertação.

Todos os exemplos de modelos probabilísticos apresentados nesta dissertação foram modelados na própria ferramenta UnBBayes. Adicionalmente, uma certa familiaridade com UML é desejável, para melhor compreender os diagramas apresentados no decorrer desta dissertação.

1.1 Contexto e Definição do Problema

A Web Semântica é definida pela W3C como a "visão para o futuro da Web, cujas informações possuem significados explícitos, facilitando as máquinas no processo de interpretação e integração de informações disponíveis na Web" (7). Alguns dos seus componentes previstos incluem *tags* especiais para a identificação semântica do documento (conteúdo), ontologias para a organização das *tags*, agentes de busca que utilizem máquinas de inferência (*reasoner*) para a captura de significados da ontologia, e entre outros componentes.

No âmbito de ontologias, a OWL (55, 91) é recomendada pela W3C para a especificação de ontologias a qual é dotada de um grande repositório de exemplos e aplicações. Esta linguagem provê um ferramental para a elaboração de uma taxonomia (i.e. hierarquia de classes e suas propriedades associadas) para a criação de um modelo complexo de conhecimento, e sua lógica é baseada em DL. Todavia, OWL (e seu sucessor OWL 2), não possui suporte natural à representação e tratamento de incerteza.

Não surpreendentemente, a Web Semântica e diversos outros domínios do mundo real estão estruturados de maneira em que informações não são sempre completas, consistentes ou imutáveis. Fatos podem ser desconhecidos (ignorância teórica), não revelados (ignorância prática, causada pela falta de evidências sugestivas) ou "grosseiras" (não refinadas - sem detalhes suficientes). Portanto, o tratamento de incertezas tornou-se um fator extremamente importante em tais sistemas.

Nesse contexto, modelos probabilísticos gráficos tornaram-se uma opção atraente para os engenheiros do conhecimento que procuram por formalismos coerentes para tratamento de informações incompletas. Graças à habilidade de expressar o conhecimento em componentes modulares inter-relacionados (*i.e.* nós em grafos), tais modelos permitem melhorar a tratabilidade das informações. A BN (93), um modelo gráfico flexível para se expressar uma distribuição de probabilidades conjunta em hipóteses interrelacionadas, destaca-se como uma das abordagens mais promissoras na reprentação de incertezas na Web Semântica (66).

Seguindo esse fluxo histórico, *Probabilistic OWL* (PR-OWL) (31, 34) foi formulado como uma extensão da OWL para o tratamento de incertezas, baseando-se em lógicas providas do formalismo MEBN - uma extensão de BN com expressividade também inerente da FOL (67). PR-OWL estende a OWL através da construção de estruturas MEBN utilizando elementos da OWL, portanto, uma ontologia probabilística em PR-OWL é, radicalmente falando, uma ontologia também em OWL, mas com atribuições especiais de significados em algumas classes e propriedades para que estas sejam interpretadas como elementos em uma MEBN (vide Seção 2.5). Como uma ontologia escrita em PR-OWL é, em sua taxonomia, uma ontologia escrita em OWL, um certo nível de compatibilidade morfossintática[1] é oferecida.

Durante meu trabalho de graduação, participei do desenvolvimento do UnBBayes-MEBN, um módulo computacional para edição e inferência de MEBN no framework UnBBayes (15, 79, 81, 100). O UnBBayes, disponibilizado como um software livre escrito em linguagem Java, oferece uma GUI e API para raciocínio probabilístico baseado em BN (17) e tem servido como infraestrutura para outras variações, como o UnBMiner (64). Com a adoção da PR-OWL como o formato de persistência de modelos MEBN, o UnBBayes tem contribuído para a comunidade também como uma implementação pioneira de um editor visual e máquina de inferência em ontologias probabilísticas.

[1] A compatibilidade morfossintática entre PR-OWL e OWL permite, por exemplo, que ontologias PR-OWL possam ser escritas em editores OWL (apesar de exigir um esforço considerável) e que máquinas de inferência em DL possam ser utilizadas para busca de indivíduos na taxonomia da PR-OWL.

De forma geral, pesquisas relacionadas à tecnologia de informação são reconhecidas por sua natureza evolutiva, cujo estado da arte é formulada por extensões e melhorias de trabalhos precedentes. Pesquisas em IA não diferem nesta natureza. Quando um sistema de IA passa por um longo ciclo de vida (contando os sucessivos progressos), é natural que este atraia uma quantidade significativa de pesquisadores, mentores, alunos, ou desenvolvedores; resultando em um ciclo de desenvolvimento com ritmo acelerado. Com o surgimento de novos princípios teóricos e com a crescente demanda, uma arquitetura computacional com mais reusabilidade, mais *variabilidade*, por um menor esforço de aprendizagem torna-se um grande diferencial na seleção de um ambiente (*e.g.* bibliotecas, frameworks, linguagens) de desenvolvimento de um sistema de IA.

No entanto, os seguintes problemas foram identificados no decorrer das experiências na graduação, com UnBBayes e ontologias PR-OWL:

1. por ser uma ferramenta herdada ao longo de gerações pelo GIA/UnB, e como desenvolvedores eram tipicamente compostos por estudantes de graduação, ainda em fase de treinamento e/ou aprendizagem, o UnBBayes possuía uma arquitetura bastante rígida[2], exigindo esforço considerável na extensão de suas funcionalidades;

2. adicionalmente, por falta de um mapeamento em nível metalinguístico, a PR-OWL falhava em tratar informações especificadas (e inferidas) na sua linguagem antecessora – OWL.

O primeiro problema também esteve intimamente ligado ao elevado índice de erros introduzidos à cada iteração evolutiva da ferramenta. Apesar do Java ser a linguagem de escolha, apenas o suporte natural ao reuso e modularização (graças ao paradigma orientado a objeto) e a portabilidade (graças à máquina virtual Java) não se mostraram suficientes para se alcançar a produtividade desejada.

Para a amenização desse problema, técnicas de SPL (108) foram utilizadas para que variações sejam realizadas com mínima alteração. Particularmente, abordagens que tratam a variabilidade em tempo de execução (e.g. plug-ins) tornaram-se bastante interessantes, devido sua flexibilidade e por tornar desnecessário as alterações em código fonte para o acréscimo de funcionalidades.

Para sanar o segundo problema, o doutorando Rommel Novaes Carvalho, integrante da George Mason University (GMU), trabalhou na especificação da linguagem PR-OWL versão 2, uma extensão da PR-OWL para permitir raciocínio probabilístico com reaproveitamento de informações especificadas e inferidas em OWL convencional.

Para uma melhor compreenção do relacionamento entre a pesquisa do atual mestrando e do referido doutorando Rommel Novaes Carvalho, deve-se também compreender suas diferenças. Seguem abaixo:

1. Enquanto que o foco do doutorando está em formular a sintaxe e a semântica da linguagem PR-OWL 2, o foco do mestrando está na implementação de uma ferramenta computacional para edição e inferência na linguagem PR-OWL 2. Em outras palavras, o doutorando está formulando uma linguagem, enquanto que o mestrando está paralelamente criando um software que a manipula[3].

[2]Detalhes técnicos sobre a referida rigidez arquitetural estão descritas nas Seções 3.2 e 4.1.

[3]Esse paralelismo obriga frequentes ajustes na implementação. Isso também foi um dos motivos na adoção de DSPL, pois sua flexibilidade minimiza o custo de tais ajustes.

2. O framework UnBBayes, selecionado para a implementação da PR-OWL 2, possuía variabilidade limitada, comprometendo o desenvolvimento do novo componente. Para sanar tal questão, o mestrando também foca na aplicação de técnicas de ES para refatoração do UnBBayes, tornando-o uma infraestrutura mais modular baseada em plug-ins. Naturalmente, essa refatoração não está no escopo do doutorado de Rommel.

Em suma, a presente pesquisa de mestrado é um fruto da cooperação entre o GIA/UnB e a GMU para o desenvolvimento de uma DSPL que ofereça, dentre outras funcionalidades, um ferramental para suporte à linguagem PR-OWL 2.

1.2 Objetivos

O objetivo principal desta pesquisa é implementar no framework UnBBayes o modelo de integração de ontologias probabilísticas e determinísticas baseadas na sintaxe e semântica PR-OWL 2.

Para se alcançar o objetivo principal, foram considerados alguns objetivos secundários. Seguem eles:

- aplicar técnicas de SPL para refatorar o framework UnBBayes a um ambiente baseado em plug-ins, para facilitar na criação do componente PR-OWL 2 e componentes futuros; isso inclui a elaboração de uma documentação centralizada que especifique dependências entre os artefatos de softwares (plug-ins) como um FM (vide Seção 3.1.1 para definição de FM);
- refatorar todas as funcionalidades preexistentes no UnBBayes, criadas por gerações de desenvolvedores do GIA/UnB, para se criar uma nova arquitetura baseada em plug-ins com um repositório inicial extensível.

Uma especificação mais completa dos critérios de conclusão da pesquisa estão apresentadas na Seção 1.6.

1.3 Abordagem da Solução Proposta

Normalmente, o processo de identificação e extração de *features* (vide Seção 3.1.1 para definição de *feature*) exige inspeções profundas no código ou uso de programas especiais para identificação de funcionalidades independentes e relacionamentos lógicos. A complexidade desse processo aumenta de acordo com a quantidade potencial de *features*, principalmente se algumas dessas não forem comportamentos fáceis de se observar no sistema.

Na nossa abordagem, uma *feature* representa basicamente uma alteração no UnBBayes realizada por algum trabalho anterior de um membro do GIA/UnB. Portanto, cada *feature* - ou um conjunto de *features* intimamente relacionadas - está mapeada a alguma publicação anterior do grupo (e.g. monografias, dissertações, teses, artigos).

Graças a isso, as *features* do UnBBayes estão ligadas a comportamentos imediatamente observáveis no sistema (*i.e* formalismos implementados no UnBBayes) e seus relacionamentos possuem fundamentos teóricos em IA, garantindo uma arquitetura intuitiva (pelo menos para os

familiarizados em IA) e poupando esforço considerável que seria gasto com inspeção de código. A Seção 4.1 descreve como foi realizada essa relação entre *features* e artefatos do GIA/UnB.

O UnBBayes utiliza o JPF versão 1.5.1 como uma biblioteca externa para prover um ambiente de plug-ins (vide Seção 3.3.1). Para manter a CK (vide Seção 3.1 para definições sobre CK e conceitos associados a SPL) simples, assume-se que um plug-in do UnBBayes representa logicamente uma única *feature*.

FMs são utilizados para descrever a nova arquitetura como uma SPL. Utilizou-se a linguagem OWL 2 para esse fim, seguindo uma adaptação da especificação descrita em (111). Como OWL 2 é um formato manipulável pelo UnBBayes, esta abordagem torna o FM editável e interpretável pela própria ferramenta, afixando a habilidade de se realizar inferências sobre sua própria arquitetura.

A implementação da PR-OWL 2 é realizada como um plug-in no padrão JPF, preenchendo pontos de extensão para GUI, persistência e algoritmos de inferência no módulo MEBN do UnBBayes (vide Capítulo 5). Essa implementação foi realizada de forma paralela à especificação da PR-OWL 2; ou seja, a implementação iniciou-se antes mesmo da especificação PR-OWL 2 estar completa. Felizmente, a adoção de técnicas de DSPL ofereceu um alto nível de flexibilidade à arquitetura do UnBBayes, fazendo com que impactos causados por alterações na especificação PR-OWL 2 fossem minimizados.

Versões de prova de conceito (PR-OWL 1 escrita em OWL 2 e com ligações entre RV e propriedades OWL) também foram implementadas inicialmente, a fim de identificar eventuais falhas na PR-OWL 2. Experiências adquiridas nessa prova de conceito foram utilizadas pela GMU no aprimoramento da PR-OWL 2.

1.3.1 Metodologia de Pesquisa

A metodologia adotada pela presente pesquisa se resume nos tópicos abaixo:

- Utilizou-se o controlador de versão SVN do SourceForge, sítio hospedeiro do UnBBayes, para o gerenciamento do código fonte. A linguagem utilizada foi o Java 5, para melhor compatibilidade com sistemas Windows, Linux e MacOS.
- Aderiu-se ao framework do UnBBayes para desenvolvimento, com uso de mecanismos de internacionalização, testes JUnit e outras práticas habituais do GIA/UnB.
- Realizou-se reuniões on-line semanais com o doutorando Rommel Carvalho, sob supervisão do professor orientador (do mestrando), para discussões sobre a extensão da PR-OWL, sincronização e acompanhamento de desempenho.
- Realizou-se um estudo histórico dos trabalhos de pesquisa que utilizaram o UnBBayes e resultaram em sua variação, para que se possa separar as funcionalidades observáveis no sistema em *features* (*i.e.* identificação de *features*).
- Realizou-se refatorações no UnBBayes para possibilitar a adoção do JPF como biblioteca. Criou-se pontos de extensões (de plug-ins) para o core e os plug-ins. Os plug-ins do core representam formalismos incorporados no UnBBayes, e plug-ins podem ser estendidos por "sub" plug-ins.
- Utilizou-se o software Protégé (<http://protege.stanford.edu/>) para o estudo inicial da sintaxe de PR-OWL 2, para permitir o levantamento dos tipos de buscas em DL necessárias para a navegação e tratamento da porção determinística da ontologia.

- Implementou-se o plug-in da PR-OWL 2 no módulo MEBN. Como foi projetado como um plug-in do módulo de MEBN, restrições inerentes da implementação do módulo MEBN afetaram diretamente na implementação de PR-OWL 2.
- Utilizou-se OWL API para a representação de elementos da OWL versão 2.
- Utilizou-se Protégé versão 4.1 como biblioteca gráfica da porção determinística.
- Utilizou-se uma máquina de inferência de OWL 2 (e.g. HermiT) para a inferência na porção determinística.
- Gerou-se um FM, escrito em OWL 2, no UnBBayes.
- Pesquisou-se domínios para exemplo de PR-OWL 2.

1.4 Áreas de Pesquisas Relacionadas

Por natureza, o framework UnBBayes está intimamente ligado à IA, particularmente na representação de conhecimentos com tratamento de incertezas, utilizando abordagens numéricas com modelos probabilísticos Bayesianos. Adicionalmente, por se tratar de ontologias - uma forma de representação de conhecimento prevista para a Web Semântica - áreas de pesquisa em Web Semântica também estão intimamente ligadas ao presente trabalho.

Como previamente mencionado, a refatoração do UnBBayes para suporte a plug-ins está relacionada à área de ES. Como o uso de plug-ins no UnBBayes resulta em um conjunto de variações de acordo com os plug-ins carregados (*i.e.* transforma as diferentes versões/edições do UnBBayes em uma família de *softwares*), esta pesquisa está relacionada SPL também.

1.4.1 Nicho Atacado

Dois grandes nichos são atacados com a presente pesquisa:

1. ES, na refatoração/reescrita de um sistema existente para adaptá-lo a uma arquitetura mais flexível, no modelo de plug-ins;
2. IA, ao trabalhar com representação de conhecimento e inferência lógica e probabilística via representação ontológica.

O nicho foi escolhido em função da potencial aplicabilidade em problemas reais. Refatorações arquiteturais, como suporte a plug-ins, exercitam um caso real de manutenção evolutiva de um sistema computacional, e o tratamento de incerteza torna-se uma necessidade em múltiplos casos: junção de bases de conhecimento de diferentes áreas (*i.e.* informações imprecisas ou com conflitos); modelagem de domínios cuja própria natureza é ligada à incerteza (ex. física quântica ou genética); tratamento de dados incompletos; simulação de raciocínio diagnóstico (*e.g.* dado a consequência, inferir a causa).

1.5 Contribuições

Considerando-se que uma das vantagens do uso de ontologias, no âmbito da Web Semântica, está na interoperabilidade entre diferentes sistemas, através da integração de conhecimento

de diferentes domínios, com mínima perda ou contaminação (31, 34), o problema da incompatibilidade da PR-OWL com o conhecimento previamente modelado em OWL era notavelmente grave, pois dificultava o reuso de conhecimento.

Como a nova especificação da PR-OWL - a PR-OWL 2 - oferece compatibilidade sintática e semântica com OWL (particularmente para OWL versão 2), a criação de uma GUI para a edição na nova linguagem oferece aos engenheiros de ontologias um meio simples e guiado para a modelagem de ontologias probabilísticas, com reuso de ontologias previamente modeladas em OWL. Consequentemente, o presente trabalho contribui para a comunidade com uma implementação pioneira na área de integração de ontologias probabilísticas e não probabilísticas com raciocínio concomitante, intercalado e miscigenado.

Adicionalmente, a refatoração arquitetural do UnBBayes para suporte a plug-ins resulta em uma ferramenta com maior variabilidade, possibilitando que o uso e adoção deste framework pelo GIA/UnB e por terceiros sejam facilitados. Esta alteração contribui com um exemplo de aplicação de técnicas de SPL com abordagem extrativa (i.e. geração de uma SPL a partir de um produto já existente), com funcionalidades adicionadas dinamicamente.

A Seção 6.4 apresenta indicadores que evidenciam o impacto da refatoração do UnBBayes e da implementação de um plug-in de PR-OWL 2 no mundo, como também algums trabalhos potencialmente beneficiados com o resultado desta pesquisa.

1.6 Critérios de Conclusão da Pesquisa

Considerou-se concluído o projeto com o cumprimento dos seguintes requisitos:

1. Refatoração do UnBBayes para fornecer uma infraestrutura baseada em plug-ins, provendo no mínimo os seguintes pontos de extensão:

    - módulos – funcionalidades relativamente independentes (não dependem diretamente de detalhes de implementação da BN no UnBBayes), carregam informações sobre quais extensões de arquivos podem ser tratados e se manifestam como janelas internas que são abertas em um dos seguintes eventos:
        (a) seleção de um item (associado ao módulo) na barra de menu;
        (b) seleção de um botão (associado ao módulo) na barra de ferramentas;
        (c) seleção de um arquivo que tenha uma extensão tratável pelo módulo.
    - algoritmos de inferência – funcionalidades para compilação de BNs, propagação de evidências e um painel para edição de preferências;
    - entrada e saída (Input/Output, ou I/O) – funcionalidades para se criar uma BN a partir de um arquivo (entrada) ou para se criar um arquivo a partir de uma BN (saída);
    - recursos para nacionalização – coleção de classes Java que serão automaticamente carregados dependendo da configuração regional do sistema operacional, para permitir que mensagens do sistema sejam customizados de acordo com o idioma;
    - novos tipos de nós - funcionalidades para se poder criar novos tipos de nós em uma BN, inclusive classes para renderização em tela e painéis de edição dos dados;
    - painéis para funções de probabilidade – funcionalidades para se criar novos painéis para a edição das distribuições de probabilidade dos nós existentes em uma BN.

2. Refatoração das seguintes variações e funcionalidades especiais do antigo UnBBayes como plug-ins na nova arquitetura:

    - I/O de BN (NET, XMLBIF e DNE) (63, 79),
    - UnBMiner (64),
    - aprendizagem bayesiana (parâmetros, estruturas, incremental, TAN e BAN) (36, 37, 46, 74),
    - metáfora médica (63),
    - metáfora de identificação humana (81),
    - OOBN e sua nacionalização para japonês (76),
    - PRM (77),
    - MSBN (16),
    - UnBBayes-MEBN (18, 81),
    - compilador de MEBN para MSBN (14, 80),
    - gerador de amostras (Monte Carlo, likelihood weighting, Gibbs) (13, 37),
    - inferência em BN por aproximação (likelihood weighting, Gibbs) (13, 79).

3. criação dos seguintes pontos de extensão no UnBBayes-MEBN (22):

    - entrada e saída para MEBN – funcionalidades de entrada e saída específicas para MEBN, mas reutilizando a mesma interface de plug-ins de entrada e saída das BNs do UnBBayes;
    - base de conhecimento (Knowledge Base ou KB) – um facilitador de acesso a base de conhecimento, que contém regras e fatos em FOL, e um painel para edição de preferências;
    - gerador de SSBN – algoritmo de inferência de MEBN e um painel para edição de preferências;
    - novos painéis de edição – funcionalidades para adicionar novas abas na GUI do UnBBayes-MEBN para editar modelos MEBN.

4. Implementação dos seguintes plug-ins para o UnBBayes-MEBN, para permitir manipulação de ontologias em PR-OWL 2:

    - painel (aba) para visualização e edição de ontologia não probabilística (usando-se o Protégé 4.1) (47);
    - base de conhecimento, para permitir no mínimo as seguintes operações:
        (a) verificar se restrições (nós de contexto na MEBN), especificadas como expressões em FOL, são satisfeitas na ontologia OWL;
        (b) indivíduos na ontologia OWL que não pertencem à taxonomia da PR-OWL2 sejam entendidos também como entidades (conceitos) MEBN;
        (c) fatos expressos na ontologia OWL sejam automaticamente entendidas como fatos em MEBN (seja, informações explícitas em OWL sejam consideradas como evidências em MEBN);

(d) permitir todas as operações anteriores sejam realizadas com reasoners de DL disponíveis no mercado/comunidade;

- I/O, para permitir que ontologias escritas em PR-OWL 2 possam ser traduzidas para estruturas de dados no UnBBayes-MEBN e vice-versa, com auxílio de um reasoner em DL;
- painel (aba) para edição do mapeamento entre um nó residente (porção probabilística) e propriedade OWL (porção não probabilística), que provê as seguintes funcionalidades:
  (a) criar nós residentes a partir de propriedades OWL - funcionalidade para se arrastar uma ou mais propriedades OWL ao painel de edição gráfico, gerando automaticamente nós residentes já mapeados a propriedades OWL;
  (b) mapear nós residentes e seus argumentos a propriedades OWL.

5. criação de um FM (vide Capítulo 6) na linguagem OWL, no mesmo padrão utilizado no artigo (111), que contenha no mínimo as seguintes informações:

   - conjunto de todos os plug-ins criados nesta pesquisa,
   - conjunto de axiomas que definam os relacionamentos (e.g dependência) entre os plug-ins, para que se possa usar inferência para classificar uma configuração (i.e. conjunto de plug-ins) em válida ou inválida.

Após concluído os requisitos acima, foi realizada a modelagem de um domínio (ontologia) visando o exercício das novas funcionalidades da PR-OWL 2 baseando-se no FM na linguagem OWL (Capítulo 6). Um estudo de caso sobre fusão de ontologias com uso de PR-OWL 2 também foi realizado (Capítulo 7).

1.7 Estrutura da Dissertação

Esta dissertação divide-se conceitualmente em duas grandes partes, de acordo com os nichos atacados pela presente pesquisa: capítulos que tratam sobre conceitos de IA e capítulos que tratam sobre conceitos de ES. Leitores mais interessados em IA poderão enfatizar a leitura dos capítulos sobre IA, e os leitores interessados em ES poderão focar mais nos capítulos de ES. Seguem abaixo descrição dos capítulos que tratam sobre IA:

- Capítulo 2 oferece uma introdução teórica aos conceitos básicos relacionados a IA, como BN, ontologias, FOL, MEBN, PR-OWL e PR-OWL 2, outras alternativas (formalismos ou implementações) para tratamento de incertezas e afins;
- Capítulo 5 é um capítulo que aborda tanto sobre IA quanto ES e apresenta os detalhes de implementação do plug-in da PR-OWL 2 na ferramenta UnBBayes;
- Capítulo 6 é um capítulo em comum entre IA e ES e apresenta os resultados da pesquisa. Foi gerado um FM do UnBBayes em OWL; seguido de impactos da utilização do UnBBayes e PR-OWL 2 no mundo (Seção 6.4.3), para ilustrar as contribuições reais e potenciais que esta pesquisa tem oferecido à comunidade;

- Capítulo 7 apresenta um estudo de caso da PR-OWL 2: fusão de conhecimentos oriundos de múltiplas fontes (*e.g.* órgãos do governo) para auxílio na detecção de fraudes em licitações públicas.

Similarmente, seguem os capítulos relacionados a ES:

- Capítulo 3 apresenta nas seções iniciais uma breve introdução teórica sobre DSPL, seguido de uma visão geral da solução proposta: arquitetura de plug-ins do UnBBayes;
- Capítulo 4 detalha cada plug-in resultante da refatoração do UnBBayes;
- Capítulos 5 e Capítulo 6 são uma interseção entre IA e ES e apresentam respectivamente a implementação de PR-OWL 2 como plug-in e criação do modelo de características como resultado da refatoração do UnBBayes a uma DSPL. A Seção 5.4 apresenta também alguns resultados de análise sistemática de performance do plug-in de PR-OWL 2.

Por fim, o Capítulo 8 apresenta as conclusões, limitações do trabalho e trabalhos futuros.

Capítulo 2

Base Teórica em IA

Este capítulo apresenta a fundamentação teórica necessária para o acompanhamento das ideias apresentadas neste documento, descrevendo de forma simplificada alguns dos aspectos, técnicas e formalismos que são ou serão utilizados durante a pesquisa. Conceitos referentes às BNs, FOL, ontologias, Web Semântica, formalismo MEBN, a linguagem PR-OWL e PR-OWL 2 são tratados aqui.

2.1 Visão Geral sobre BN

No âmbito de formalismos em IA, uma BN é um modelo compacto de raciocínio probabilístico para a representação e manipulação de um conhecimento com incertezas (92). As informações na BN são representadas como um grafo acíclico orientado e um conjunto de CPTs. Cada nó de uma BN representa uma RV, e os arcos representam dependências qualitativas entre as RV. Cada RV possui um conjunto de valores possíveis, em que somente um valor pode ser assumido em um determinado momento (mas não há a certeza de qual desses valores) (68).

O raciocínio em modelos bayesianos é geralmente baseado no **teorema de *Bayes***, um método utilizado para atualização de probabilidades de uma proposição a partir de novas informações. Portanto, na visão bayesiana, a inferência pode ser entendida como um problema de atualização de crenças, engatilhadas a partir do surgimento de novas evidências. Segue abaixo a fórmula padrão do teorema de Bayes:

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)} ,$$

$P(X|Y)$, é a probabilidade *a posteriori* de um evento X dado o evento Y. $P(Y)$ é a probabilidade *a priori* de Y e representa nossa crença sobre o evento Y antes de se obter uma evidência. De forma similar, $P(X)$ é a probabilidade *a priori* de X. $P(Y|X)$ é a probabilidade do evento Y dado como ocorrido o evento X. A Figura 2.1, mostra um exemplo de BN. A Tabela 2.1 especifica a distribuição de probabilidades para o nó "Cachorro fora de Casa" de acordo com os valores dos pais.

Nesse exemplo, suponhamos que se queira saber se o cachorro está passando mal. Pelo grafo (e pela CPT), podemos perceber que o fato do cachorro **não** estar passando mal é um forte causador da saída do cachorro (as probabilidades de saída do cachorro aumentam quando o cachorro não está passando mal, e vice-versa). O teorema de Bayes permite inferir o inverso

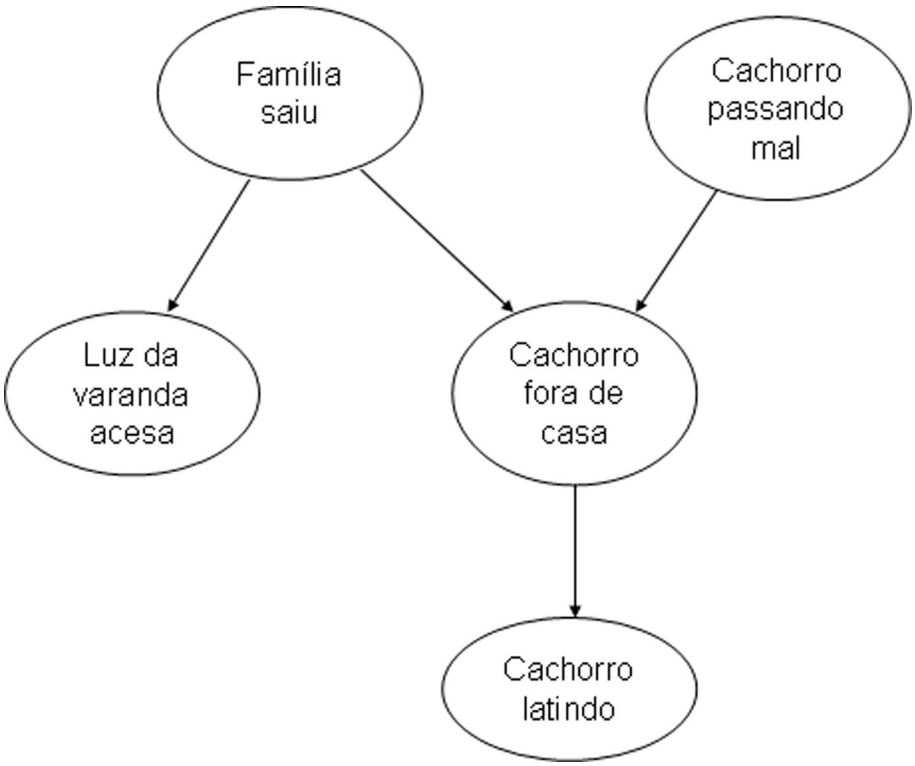

Figura 2.1: Figura exemplo de BN, adaptada de (25)

Cachorro passando mal	Família Saiu	**Verdadeiro**	**Falso**
Verdadeiro	Verdadeiro	**10**%	**90** %
Verdadeiro	Falso	**20**%	**80**%
Falso	Verdadeiro	**70**%	**30**%
Falso	Falso	**90**%	**10**%

Tabela 2.1: CPT do nó “Cachorro fora de casa”.

também. Quanto maior for a probabilidade do cachorro não ter saído, maior é a probabilidade do cachorro estar passando mal. Particularmente, se sabemos com certeza que o cachorro está fora de casa (ou seja, probabilidade 100%), então a probabilidade do cachorro estar passando mal é bem baixa. As CPTs, que especificam quantitativamente essas regras probabilísticas, podem ser construídas por um especialista do domínio ou por regularidades estatísticas (*e.g.* aprendizagem por máquina).

Uma das vantagens em se trabalhar com BN é o fato de não ser necessário especificar CPTs para todas as combinações possíveis de variáveis. Isso é possível por que algumas variáveis são condicionalmente independentes entre si, não influenciando então nas probabilidades. Variáveis não conectadas por arcos são as condicionalmente independentes, e variáveis conectadas por arcos são diretamente dependentes.

Ainda no exemplo anterior, suponhamos que não temos certeza se o cachorro está fora de casa. Se a probabilidade do cachorro estar passando mal é alterada, então a do cachorro estar fora de casa também será, e conseqüentemente a do cachorro estar latindo também será, por causa da dependência direta (portanto, existe uma dependência indireta entre o cachorro estar passando mal e o cachorro estar latindo). Porém, caso tenhamos certeza de que o cachorro está fora de casa (*i.e.* variável "cachorro fora de casa" é 100% verdadeira), então a variável "Cachorro passando mal" não influenciará mais a variável "Cachorro latindo", pois esta será influenciada diretamente pela variável "Cachorro fora de casa", cujo valor já é conhecido (e "fixado" em 100%)[4].

Assim, em vez de termos de especificar as relações de probabilidade entre todas as variáveis, precisamos especificar apenas as relações das variáveis diretamente dependentes. Isso facilita o uso de BN computacionalmente. É possível calcular a distribuição de probabilidade conjunta através da fórmula abaixo:

$$P(x_1,...,x_n) \;=\; \prod_{i=1}^{n} P(x_i|pais(x_i))$$

Sendo $P(x_i|pais(x_i))$ a probabilidade de ocorrência de x_i condicionada às variáveis pais. Portanto, é possível calcular a distribuição de probabilidade conjunta a partir de probabilidades condicionais locais, tornando a BN uma representação bastante compacta.

As BNs possuem diversas aplicações. Entre elas, podemos citar: diagnósticos de defeitos em processadores (técnica utilizada pela Intel), diagnósticos médicos (101), exames patológicos, *troublesolver* do Microsoft Word, aplicações militares, controle de spam, interpretação de linguagens (26), reconhecimento de imagens (9), entre outros. Inclusive, o GIA/UnB pesquisa a aplicação de modelos de inferência bayesiana na web (21).

Entretanto, as BNs carecem de expressividade forte o suficiente para representar diversas situações do mundo real. Esta incapacidade surge principalmente pelo fato das RVs em BN possuírem características proposicionais. Ou seja, um nó representa virtualmente a incerteza sobre uma única proposição e não consegue representar a incerteza sobre um conjunto potencialmente infinito de entidades (similares) em um domínio. Este problema é análogo ao problema da lógica proposicional, que motivou o surgimento da FOL (vide Seção 2.2).

Por exemplo, apesar dessas BNs convencionais serem úteis na modelagem de um domínio incerto, elas não conseguem representar corretamente um domínio em que se desconhece a quantidade de condicionantes à *priori* (*i.e.* incerteza sobre quantos pais um determinado nó

[4]O fornecimento de informações conhecidas à BN é chamada de entrada de evidências.

possui - isto deve estar pré-determinado[5]). Adicionalmente, como o grafo deve ser acíclico, não há como modelar recursividade de RVs (*i.e.* quando uma variável condiciona a ela mesma direta ou indiretamente), cujo o número de etapas que devem ser consideradas é indeterminado.

Para exemplificar as deficiências, será utilizada aqui uma pequena parte do domínio Identificação de Veículos (78). Nesse domínio, o objetivo é determinar automaticamente se um objeto captado por sensores é um tanque (*Tracked Vehicle*), ou um veículo comum em rodas (*Wheeled Vehicle*) ou se foi algum outro tipo de objeto qualquer (*NonVehicle*) para se determinar o nível de perigo. Algumas das informações disponibilizadas por sensores são: informações sobre o local (região) onde o objeto se encontra, tipo do terreno, relatórios de sensores baseados em imagens, relatório de sensores MTI.

A Figura 2.2 ilustra uma possível representação em BN de como os sensores captam informações para informar o tipo do objeto captado, para que então um agente possa determinar o seu nível de ameaça.

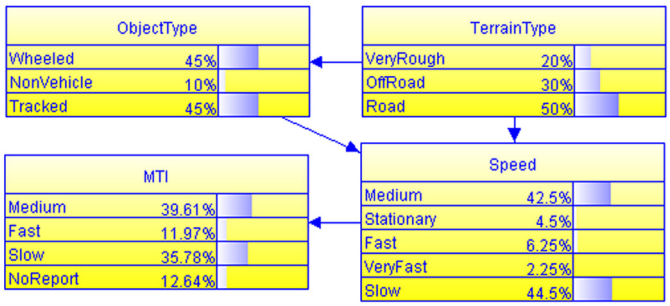

Figura 2.2: Um subdomínio de identificação de veículos através de informações de sensores, para ilustrar limitações na BN .

Os valores de *Speed* (velocidade real do objeto) dependem de *TerrainType* (tipo do terreno onde o objeto se encontra) e de *ObjectType* (tipo do objeto), pois um terreno acidentado diminuirá a velocidade do objeto e um objeto em rodas tenderá a ser mais veloz na estrada. *ObjectType* depende de *TerrainType* pelo fato dos veículos em rodas serem mais prováveis de circularem em estradas do que em terrenos acidentados. *MTI* representa a velocidade detectada por um sensor e foi incluída no modelo para indicar que a velocidade observada pode ser diferente da real, com um certo grau de incerteza.

Cada RV possui um conjunto de valores possíveis (*e.g.* a RV *TerrainType* possui *VeryRough*, *OffRoad* e *Road* como valores possíveis) e a probabilidade da RV assumir esses valores possíveis é especificada em uma CPT, onde estão declaradas probabilidades em função dos valores possíveis dos pais. A exemplo, a Tabela 2.2 declara as probabilidades de *ObjectType* em função dos valores possíveis de *TerrainType*. O nó *TerrainType*, por ser um nó raiz (*i.e.* não possui pais/dependências), pode ter suas probabilidades especificadas não sendo em função de outras RVs.

Esta modelagem simples funciona apenas quando houver no máximo um veículo nas proximidades. Caso haja dois veículos se aproximando simultaneamente, será necessário utilizar

[5]Muitos podem alegar que podemos usar aprendizagem estrutural de BN para resolver o problema de quantidade indefinida de RVs, mas nesse caso estamos tecnicamente dependendo de uma representação "fora" da BN (*i.e.* dados); portanto, a BN em si não está representando todo esse universo de RVs e dependências possíveis.

TerrainType	**Wheeled**	**NonVehicle**	**Tracked**
VeryRough	**10**%	**10**%	**80** %
OffRoad	**10**%	**10**%	**80**%
Road	**80**%	**10**%	**10**%

Tabela 2.2: CPT de *ObjectType*.

um novo modelo. Ora, em uma situação real não se sabe *a priori* quantos veículos vão se aproximar, tornando essa modelagem utilizando BN inviável, pois seria necessário um modelo para cada número de veículos (note que, neste caso, o número de possibilidade é infinito). As BNs carecem de poder expressivo para representar tipos de entidades que podem ser instanciadas tantas vezes quanto requeridas pela situação.

Considere agora uma segunda situação. Admita que apenas um veículo esteja nas proximidades em um determinado momento. Suponha que no momento anterior *T0*, o objeto esteve em repouso. Suponha que "agora" (momento *T1*) o sensor de velocidades detectou velocidade alta. É normal pensarmos então que o objeto em questão realizou uma aceleração, algo incomum em objetos que não são "veículos". No entanto, essa ocorrência poderia ser uma mera falha no sensor. Uma forma de confirmar essa ocorrência seria comparar as medições seguintes no sensor. Ora, as BNs (clássicas) são estáticas, impedindo a recursão necessária para resolver problemas deste tipo. Esta é outra restrição que impede a continuação da modelagem.

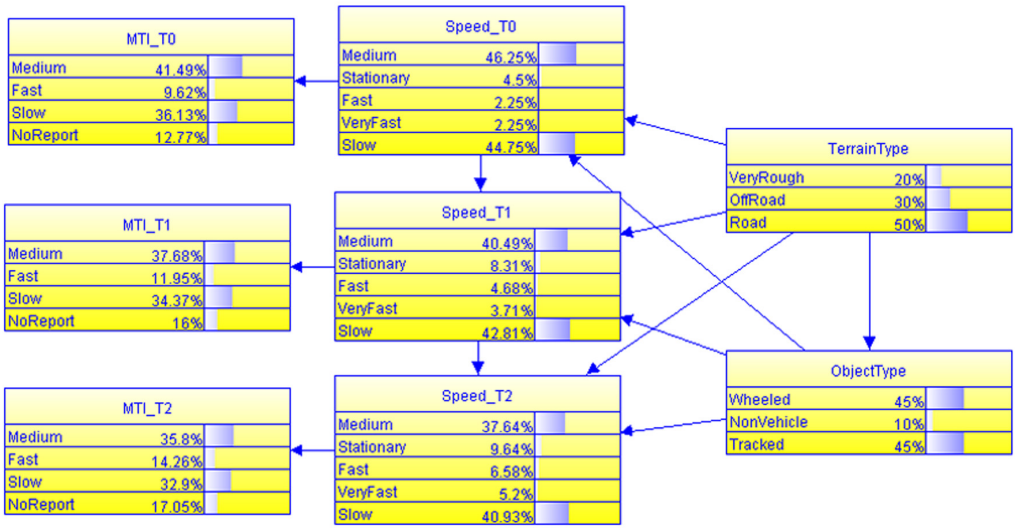

Figura 2.3: BN para o caso de identificação do tipo de um veículo reportado por um sensor de velocidade, com recursão

Estes dois exemplos mostram dificuldades que um modelador poderá enfrentar ao tentar representar o mundo utilizando BNs convencionais. Então, apesar da ampla utilidade, a BN falha na representação de diversos domínios complexos e reais, oriundas da falta de suporte a domínios com quantidade indefinida (e potencialmente infinita) de RV (30). Visando a solução de tal problema, a Dra. Laskey da GMU formulou a MEBN (vide Seção 2.5), extensão de BN que integra FOL (vide Seção 2.2) com a teoria de probabilidade Bayesiana (68). Como os

problemas da BN apresentados nesta seção são análogos aos presentes na lógica proposicional; a MEBN, por utilizar FOL, pode ser vista como uma evolução natural.

2.2 A Lógica de Primeira Ordem - FOL

Desde a antiguidade o homem inventa formalismos para descrever o mundo de uma forma a eliminar as ambigüidades da linguagem natural. O principal objetivo destas descrições é permitir mecanismos de inferência para se obter novos conhecimentos a partir de uma base formal de conhecimento. A lógica surgiu no intuito de explicar as várias noções da verdade, codificando os enunciados (vistos como verdades lógicas), avaliar a legitimidade (validade) de argumentos e realizar inferências plausíveis.

A lógica possui vários formalismos. O mais famoso seria a lógica proposicional, que assume o mundo como uma composição de fatos, sendo estes a sua unidade básica. Fatos são verdades em algum mundo relevante. Ora, apesar de resolver problemas particulares, a lógica proposicional falha por não poder representar generalizações, sendo portanto uma representação bastante limitada.

Uma lógica com poder expressivo de FOL, por outro lado, vê o mundo como um conjunto de objetos, estes possuindo identidades individuais e propriedades que os distinguem de outros objetos.

Antes de entrar em detalhes nas características da FOL, vamos definir alguns conceitos comuns a todos os formalismos lógicos:

- O vocabulário da lógica é composto por símbolos, podendo estes serem símbolos lógicos ou símbolos não-lógicos. Os símbolos não-lógicos variam de linguagem para linguagem, fornecendo um vocabulário específico para cada domínio de aplicação.
- A sintaxe fornece regras que permitem combinar os símbolos, formando expressões legais.
- As regras de inferência especificam caminhos pelos quais novas expressões podem ser derivadas a partir de expressões legais existentes.
- A semântica caracteriza o significado das expressões.
- Uma teoria é um conjunto que engloba as sentenças de uma linguagem e suas conseqüências lógicas, definidas graças à existência de uma semântica.

A lógica deve possuir funções de mapeamento que permitam transformar frases da linguagem natural em uma frase no formalismo lógico e vice-versa. As funções de mapeamento, via de regra, não são funções um para um.

A validade de um argumento independe da sua observação no mundo real. Depende somente das validades de suas premissas e a validade da conclusão. Esta relação é chamada de relação de conseqüência.

A FOL foi inventada por Frege e Peirce, independentemente, no final do século dezenove.

A gramática abaixo mostra, em formato BNF, a definição dos elementos da FOL. Esta é uma adaptação da definição encontrada em (68).

$< Formula > := < FormulaAtomica >$

$| < Formula >< Conectivo >< Formula >$
$| < Quantificador >< Variavel >, ... < Formula >$
$|\sim < Formula >$
$|(< Formula >)$

$< FormulaAtomica > := < Predicado > (< Termo >, ...)$

$| < Termo >=< Termo >$

$< Termo > := < Funcao > (< Termo >, ...)$

$| < Constante >$
$| < Variavel >$

$< Conectivo > := \rightarrow | \wedge | \vee$

$< Quantificador > := \forall | \exists$

$< Constante > := A|X|Maria| ....$

$< Variavel > := a|x|s| ...$

$< Funcao > := PaiDe|RaizQuadradaDe| ...$

$< Predicado > := Antes|TemCor|Chovendo|...$

Seguem abaixo algumas descrições:

- **Símbolos Constantes**: são símbolos que representam os objetos do domínio. A interpretação deve especificar qual é a entidade especificada. São escritas iniciadas com letra maiúscula ou com um número. Ex.: A, Maria, 1888.
- **Símbolos Predicados**: refere-se a uma relação particular no modelo (especificado pela interpretação). É definido por um conjunto de tuplas de objetos que as satisfazem. Ex.: Irmão(Eduardo, Pedro).
- **Símbolos Funcionais**: são relações especiais, cujo um dado objeto é relacionado exatamente com outro objeto. Ex.: Pai (cada pessoa tem exatamente uma pessoa, que é seu pai). Funções podem ser utilizadas para se referir a objetos particulares sem utilizar seus nomes diretamente, como por exemplo PaiDe(Maria).
- **Termos**: são utilizados para se referir a entidades no domínio, podendo servir como argumentos para funções e predicados. Pode ser uma variável, uma constante ou uma função, sendo estas últimas termos complexos; os argumentos da função serão dispostos em uma lista parentetizada de termos, com os termos separados por virgulas.
- **Sentenças Atômicas**: representam fatos. É formado por um predicado seguido por uma lista parentetizada de termos. Ex.: Irmão(Eduardo, Pedro), que pela interpretação significa que Eduardo e Pedro são irmãos. Uma sentença atômica é verdadeira se a relação referida pelo predicado é verdadeira para os objetos referidos como seus argumentos.
- **Sentenças Complexas**: são construídas a partir de sentenças atômicas utilizando-se os símbolos lógicos. Ex.: Irmão(Eduardo, Pedro) $\bigvee$ Irmão(Maria, Joana).

- **Quantificadores**: são utilizados para expressar propriedades de coleções de objetos, ao invés de ter que enumerar os objetos por nome, como proposto na lógica proposicional. A FOL contém dois quantificadores padrões:
  - Quantificador Universal ($\forall$): é utilizado para especificar sentenças que são validas para todas os objetos. Ex.: $\forall$x (Gato(x) $\rightarrow$ Mamífero(x)).
  - Quantificador Existencial ($\exists$): é utilizado para especificar sentenças que são válidas para algum objeto, sem no entanto, nomear este. Ex: $\exists$x(Irmã(x,Spot)) $\bigwedge$ (Gato(x))

- **Igualdade**: é utilizada para dizer que dois termos se referem ao mesmo objeto. Ex.: Pai(Joao) = Henrique. A igualdade pode ser tratada como um predicado.

Para aplicar a FOL, deve-se definir um conjunto de axiomas, ou sentenças, que definam os fatos relevantes a respeito do domínio. Juntamente com as conseqüências dos axiomas, este conjunto forma a teoria do domínio. O conjunto de conseqüências será um subconjunto próprio de todas as sentenças sintaticamente corretas, caso os axiomas sejam consistentes; ou todas as sentenças, caso os axiomas sejam inconsistentes; pois qualquer proposição segue de uma contradição.

Como forma de usar a FOL para descrever o mundo, as ontologias de domínio expressam conhecimento sobre os tipos de entidades no domínio de aplicação, bem como seus atributos, comportamentos e relacionamentos. Para tal, pode-se utilizar lógicas de propósito especial, construídas sobre a FOL, que provêem construções pré-definidas para raciocínio sobre tipos, espaço ou tempo; permitindo que o modelador possua uma base por onde começar a modelagem do domínio estudado.

Apesar de sua grande expressividade, a FOL não possui suporte natural ao tratamento de incertezas. A MEBN é um formalismo híbrido que integra o tratamento de incerteza das BNs com a expressividade da FOL, trazendo para a FOL o tratamento de incertezas. Maiores informações sobre a MEBN podem ser obtidas na Seção 2.5.

2.3 Web Semântica e Ontologias

A Web Semântica é um projeto idealizado por Tim Berners-Lee, criador da HTML e da *World Wide Web*, sob os auspícios do W3C (6). O objetivo deste projeto é melhorar as potencialidades da Web através da criação de ferramentas e de padrões que permitam atribuir significados claros aos conteúdos das páginas e facilitar a sua publicação e manutenção (7). O W3C definiu a web semântica como um esforço colaborativo entre o próprio W3C e um grande número de pesquisadores e parceiros industriais que querem estender a web como um framework comum que permita que dados sejam compartilhados entre aplicações, negócios e comunidades internacionais.

Os documentos da web tradicional são construídos em linguagens orientadas a sintaxe, como o HTML e o XML. A XML é uma linguagem de marcação que permite que os usuários criem *tags* personalizadas sobre o documento criado; diferentemente da HTML, que possui estrutura de *tags* fixas, impedindo a criação de um novo conjunto de descritores. Apesar da estrutura de *tags* permitir que sejam providas regras sintáticas e convenientes para extrair, transformar e trocar dados, estas informações são insuficientes para a manipulação pelos computadores,

cabendo portanto ao ser humano realizar tarefas árduas de extração de informação e formulação de conhecimento.

Com a evolução dos computadores e o crescimento da massa de informações disponíveis na web, tornou-se necessário um planejamento para permitir que o próprio computador possa realizar raciocínios a fim de formular conhecimento. Segundo (7), os computadores necessitam ter acesso a coleções estruturadas de informações (dados e metadados) e um conjunto de regras de inferência para permitir um raciocínio automatizado. Em outros termos, precisamos de uma representação "inteligente" dos dados disponíveis na web.

Os agentes inteligentes são programas que realizam buscas e processamentos das informações disponíveis, de forma a oferecer soluções "úteis" para determinados problemas do usuário. Um agente inteligente poderia, por exemplo, marcar uma consulta médica para o usuário, procurando o médico mais acessível - em termos de proximidade e preço - e agendando uma consulta a partir de negociações com o agente inteligente do médico. A web atual não permite tal independência máquina-homem, necessária para efetuar operações com esse nível de complexidade.

Os agentes inteligentes devem ser capazes de identificar o significado exato das palavras e as relações lógicas entre elas. Para que os computadores entendam o conteúdo da web, é necessário que eles consigam acessar dados estruturados e tenham conhecimento de conjuntos de regras que os ajudem a conduzir seus raciocínios. As novas páginas da web terão de ser ou escritas em uma nova linguagem ou estruturadas automaticamente (via técnicas de mineração) para que permita a compreenção uniforme por diversos sistemas. A especificação formal dos significados dos elementos da linguagem será feita através de ontologias.

2.3.1 Ontologia - Definição

A ontologia define conceitos; todavia, o próprio conceito sobre ontologias passou por ligeiras modificações ao ser estudado por diferentes ramos da ciência. Abaixo, mostraremos os diferentes conceitos sobre ontologia em diferentes ramos da ciência.

Ontologia na filosofia: segundo Aristóteles, em Metafísica, a ontologia é conhecida como a ciência da existência, uma disciplina da filosofia. Lida com a natureza e a organização da realidade. A ontologia filosófica aborda temas como "o que caracteriza um ser" e "o que é um ser".

Ontologia na lingüística: na lingüística, a ontologia é uma tripla entre a forma, o conceito e o referente. A forma é definida como o "canal" de representação da informação (ex. A seqüência de letras "Tanque de guerra"). O conceito é definido como um conjunto das idéias relacionadas. O referente é a entidade referenciada pela forma e conceito. Veja o esquema da Figura 2.4 para uma visualização abstrata.

Na visão do W3C, a ontologia é um artefato de engenharia, utilizada para a criação da Web Semântica. Pode ser definida como uma ferramenta para especificação formal de significados e associações entre *tags*, utilizadas em documentos web. Adicionalmente, provê um vocabulário de termos, cujas novas terminologias podem ser formuladas através das existentes. A semântica (significado) das entidades é especificada de modo formal e permite identificar relacionamentos entre termos pertencentes a diversas ontologias.

Sua estrutura possui tipicamente dois componentes distintos:

- nomes dos conceitos importantes ao domínio, também conhecidos como "classes";

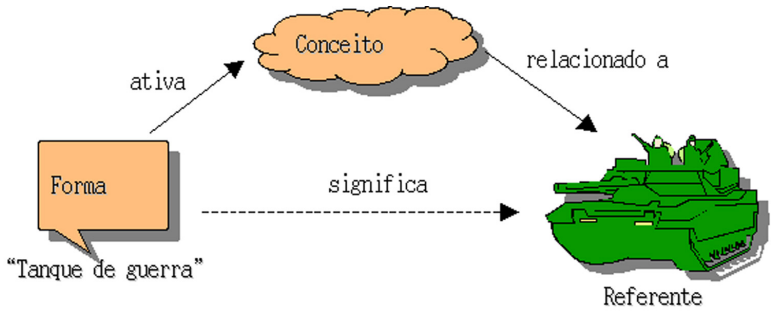

Figura 2.4: Ilustração da tripla forma-conceito-referente.

  - Ex. Humano: conceito sobre membros de uma categoria de animais. Adulto: conceito sobre seres vivos com idade maior do que um determinado valor que depende da categoria do ser vivo. Humano_Adulto: conceito sobre humanos com mais de 20 anos.

- conhecimentos e restrições sobre o domínio, também conhecido como "propriedades" .

  - Ex. Nenhum Indivíduo pode ser Adulto e Recém_Nascido. Todos os Humanos ou são Homens ou Mulheres. Um Humano_Adulto Homem tem em torno de 1,70m de altura.

A seguir, apesentaremos a linguagem OWL (91), recomendada pelo W3C como linguagem de descrição de ontologias na Web Semântica.

2.3.2 A Linguagem OWL

É uma extensão da linguagem XML e do vocabulário do RDF/RDFS (3). Acrescenta mais relações entre classes, como disjunção, cardinalidade, igualdade, tipos mais ricos, propriedades simétricas ou transitivas; sendo também basicamente composta pelas triplas (sujeito, predicado, objeto) do RDF.

A estrutura se divide em:

- **cabeçalho**, que importa ontologias e declara informações sobre a versão;
- **definição das classes**, usando propriedades como *equivalentClass*, *disjointWith* ou *oneOf* ;
- **definição das propriedades**, a partir de propriedades padrões como *TransitiveProperty* ou *inverseOf*.
- e a **declaração dos fatos dos indivíduos**, que utiliza definições como *type* para instanciação de classes.

A OWL define subconjuntos com restrições de acordo com a necessidade do projetista da ontologia. Veja abaixo os subconjuntos da OWL.

OWL Full

É definido como a junção do DAML+OIL(4) com RDF. Possui grande expressividade, podendo até considerar classes como indivíduos. Entretanto, não há garantia de computabilidade.

OWL DL

Inclui todas as construções da OWL, mas com algumas restrições que garantam computabilidade. É equivalente a uma subcategoria bastante conhecida da lógica - a DL - e essa equivalência traz grande entendimento na semântica, propriedades formais, algoritmos de verificação, implementações e otimizações; pois DL já foi intensamente estudado.

OWL Lite

É um subconjunto de OWL-DL. Possui suporte imediato aos problemas de hierarquia de classificação e cardinalidade binária (0 e 1). É uma ferramenta simples, sugerida para processos de migração de uma linguagem ontológica qualquer para a OWL.

2.3.3 A Linguagem OWL, Versão 2

OWL 2 Web Ontology Language (110), informalmente conhecida como OWL 2, é uma extensão e revisão da OWL e adiciona novas funcionalidades que facilitam na engenharia do conhecimento. Ontologias em OWL 2, como no seu predecessor, provê classes, propriedades, indivíduos, valores de dados e são armazenadas como documentos web - arquivos ou páginas web identificados por um IRI (*Internationalized Resource Identifier*). OWL 2 oferece facilidades adicionais (*e.g.* união disjunta de classes) e mais expressividade ao OWL 1, como:

- chaves;
- cadeia de propriedades;
- tipos e intervalos de dados mais avançados;
- restrições qualificada de cardinalidade;
- propriedades assimétricas, reflexivas, disjuntivas e
- mais funcionalidades relacionadas a *tags*.

Como OWL 2 é uma especificação estrutural e conceitural, não está limitada a uma sintaxe ou um formato de arquivo (no sentido de artefato computacional) em particular. No entanto, na prática uma ontologia em OWL 2 precisará estar concretizada em alguma sintaxe para que seja armazenada e trocada entre aplicações. A W3C sugere 5 tipos de sintaxes para ontologias em OWL 2:

- **RDF/XML**: representa grafos RDF. É uma implementação obrigatória e foca na interoperabilidade (softwares compatíveis com OWL 2 devem no mínimo implementar esta sintaxe).

- **OWL/XML**: representa uma serialização direta a XML e foca na facilidade de serialização em XML (facilita o processamento por ferramentas XML).
- **Funcional**: representa uma especificação estrutural, portanto facilita na visualização da estrutura formal da ontologia.
- **Manchester**: é uma sintaxe próxima da DL; portanto, facilita na leitura/escrita de ontologias em DL.
- **Turtle**: mapeia-se a grafos RDF e é um formato que facilita a leitura de triplas RDF

De forma similar ao OWL, OWL 2 também possui três sublinguagens com caracteristicas diferentes, para se adaptar às necessidades do usuário. São elas:

- **OWL 2 EL**: habilita algoritmos de tempo polinomial para todos os tipos de inferência padrão; são recomendadas em ontologias grandes em que a expressividade possui menor prioridade do que a performance.
- **OWL 2 QL**: permite que buscas conjuntivas sejam respondidas em espaço logaritmico usando-se tecnologias padrões para banco de dados relacionais; é útil quando dados devem ser acessados por consultas relacionais (*e.g.* SQL).
- **OWL 2 RL**: permite implementações de algoritmos com tempo polinomial utilizando tecnologias baseadas em regras, diretamente em triplas RDF; sendo então útil na manipulação direta de triplas RDF.

Uma ontologia OWL 1 é também uma ontologia OWL 2. Essa propriedade pode ser aproveitada para se reutilizar domínios modelados em OWL 1 em domínios baseados em OWL 2.

2.3.4 Lidando com Múltiplas Ontologias

Apesar de ontologias serem projetadas para oferecer uma especificação compartilhada, provendo uma terminologia comum entre diferentes sistemas ou fontes de informações, é difícil impor que todas as pessoas e organizações do mundo concordem a utilizar uma única ontologia global (106). Isso levantaria questões tanto técnicas quanto políticas.

Por exemplo, sistemas eventualmente precisarão trabalhar em domínios com escopos completamente diferentes (*i.e.* exigindo classes com extensões diferentes), coberturas diferentes (*i.e.* diferenças em porções do domínio cobrido pela ontologia), granularidades diferentes (*i.e.* nível de detalhe), paradigmas diferentes (*e.g.* tempo descrito como intervalo, frequência ou "pontos"), formas diferentes de classificação (*e.g.* categorização por subclasses ou por atributos), sinonímia sem consenso (*i.e.* sinônimos podem não valer em alguns domínios), presença de homonímias (*i.e.* grafia igual, diferentes significados), dificuldades na codificação ou unidades (*e.g.* quilômetros e milhas). Além disso (e mais importante), vários sistemas no mundo já utilizam suas próprias ontologias.

Visto essa dificuldade de se impor uma ontologia global comum e unificada; surge, na prática, a necessidade de se lidar com diferentes ontologias simultaneamente. Consequentemente, sistemas precisam considerar similaridades e diferenças para garantir interoperabilidade. A reconciliação dessas diferenças (e similaridades) entre ontologias podem ser realizadas em diferentes maneiras (38, 88):

- **Mapeamento**: cria-se uma representação das correspondências (e/ou diferenças) entre ontologias. Essa representação é normalmente armazenada separadamente como regras de mapeamento (especificadas geralmente em alguma linguagem declarativa). Essas correspondências podem também ser utilizadas para se converter informações de uma ontologia em outra (processo conhecido como **transformação**). A descoberta de tais correspondências com uso de ferramentas computacionais é conhecida como **alinhamento**).
- **Fusão (*merge*)**: uma nova ontologia é criada a partir da união das ontologias fonte. Um cuidado especial deve ser tomado para que a ontologia unificada realmente reflita as ontologias originais. Existem diferentes abordagens para se realizar tal fusão:
  - **Fusão completa**: une-se as informações contidas em um conjunto de ontologias, de maneira a gerar uma ontologia que substitui as originais (88).
  - **Ontologia de ponte**: cria-se uma nova ontologia com visão mais abstrata, conhecida como "ontologia ponte", contendo axiomas que especificam correspondências entre as ontologias originais (43). A ontologia ponte importa as ontologias originais e não as substituem; portanto, são mais adequadas para fusão incremental sem interrupção dos serviços que utilizam as ontologias originais.

O termo *knowledge fusion* (fusão de conhecimento) é muito utilizado em trabalhos que integram informações oriundas de múltiplas fontes para se gerar uma ontologia probabilística (vide Seção 2.6 para definição) com visão mais abstrata e global - conhecida como "*upper ontology*" (14, 28, 29, 65, 69, 71). Essas ontologias probabilísticas têm como objetivo ou resolver problemas que exijam a presença das informações de todas as fontes, ou aprimorar a qualidade das inferências com acréscimo de novas fontes. Por resultar em uma nova ontologia, a "fusão de conhecimento" também pode ser entendido como uma abordagem de *merge* (traduzido neste documento simplesmente como "fusão").

O Capítulo 7 apresenta um exemplo de fusão utilizando uma idéia similar a ontologia ponte. Contudo, diferente da ontologia ponte (cujo foco é especificar correspondências), o foco da ontologia probabilística do Capítulo 7 está em modelar regras que simulam o raciocínio de um especialista na detecção de fraudes em licitações públicas. Portanto, a preocupação está mais em se especificar a "maneira de se raciocinar com informações em várias fontes", do que especificar semelhanças ou diferenças entre tais informações.

A Seção 2.8.1 apresenta, dentre outras, ferramentas computacionais que podem auxiliar na união de ontologias por mapeamento ou fusão.

2.4 Compreendendo a Importância de Incertezas em Ontologias

Se não entrarmos muito em questões filósóficas ou religiosas, de fato a incerteza é onipresente no nosso cotidiano, e isso não é algo tão surpreendente.

Há quase um século, o Demônio de Laplace[6] foi considerado como morto por estudiosos da física quântica, que afirmam a impossibilidade de se determinar duas propriedades de uma partícula quântica simultaneamente com exatidão, inserindo-se então um certo grau de incerteza (princípio da incerteza). O raciocínio diagnóstico (amplamente utilizado por médicos, técnicos de oficinas de automóveis, auditores da Receita ou CGU, etc.) é incerto por natureza, pois utiliza as consequências (sintomas) para implicar causas - algo ilógico. Ora, a nossa própria linguagem utilizada na comunicação é repleta de incertezas; quem já conseguiu transmitir com exatidão a intensão real na primeira tentativa, sem se usar termos "redundantes"? Pelo menos, reconheço que esta dissertação está repleta de redundâncias.

Além da incerteza oriunda do próprio domínio de conhecimento, um caso previsto na Web Semântica é a união de ontologias (como mencionado na Seção 2.3.4), que é uma potencial fonte de contradições ou imprecisões na informação.

Outro problema que se imagina ocorrer em ontologias convencionais é o problema de informações incompletas. Esse tipo de situação ocorre quando instâncias (indivíduos) de classes são pesquisadas, mas algumas instâncias críticas para terminar a pesquisa não estão presentes.

Com a condição acima, uma busca que trata essas instâncias geraria resultados duvidosos, pois não haveria elementos que atenderiam a 100% do requisitado. O modelo convencional somente considera os casos de requisitos "completamente atendidos" ou "completamente falhos"; nesse caso, as instâncias que atendem "parcialmente" aos requisitos deveriam ser eliminadas ou consideradas? Note que se considerarmos as "parciais" como "completas", as "parciais" que atenderam a poucos requisitos serão tratadas com a mesma importância dos que atenderam mais requisitos; no caso contrário, se ignorarmos os resultados "parciais", informações preciosas serão desperdiçadas e a busca não seria eficaz.

Adicionalmente, problemas de confiabilidade da fonte de informação podem surgir. Surge também a necessidade de se "ponderar", de alguma forma, as informações quanto sua credibilidade na hora de realizar buscas (ou inferências). Reforça-se então a necessidade de representação e manipulação de ontologias com tratamento de incertezas, que é um dos nossos objetos de estudo.

A *Uncertainty Reasoning for the World Wide Web Incubator Group*(72), um grupo incubado pelo W3C, tem trabalhado na determinação de soluções para tratamento de incerteza em tecnologias *World Wide Web*. A linguagem PR-OWL (vide Seção 2.6), com sua abordagem probabilística, está sendo referenciada como uma das possíveis abordagens.

O Capítulo 7 apresenta um exemplo real que ilustra melhor um domínio com necessidade de tratamento de incertezas. Nesse problema, uma abordagem probabilística foi escolhida como solução. A leitura desse exemplo pode auxiliar melhor o leitor no entendimento dos detalhes sobre a necessidade de tratamento de incerteza na Web Semântica.

2.4.1 A Abordagem Probabilística

A abordagem probabilística baseia-se nas teorias da probabilidade para a representação de incertezas em uma ontologia. A probabilidade está intimamente relacionada com a freqüência,

[6]O Demônio de Laplace é uma figura hipotética criada pelo matemático Pierre-Simon Laplace na tentativa de se ilustrar uma situação em que se todas as partículas do universo e seus estados (momento) fossem conhecidos, seria possível revelar o passado e o futuro com exatidão.

crença, ocorrência ou relações de causa e efeito sobre um evento. Podemos aplicar diversos conceitos da teoria do conjunto para o tratamento de probabilidades.

A representação é feita através de numerais com valores dentro do intervalo fechado entre 0 e 1. O processo de validação e inferência é feita basicamente com o auxílio de modelos bayesianos. Um exemplo é a PR-OWL, linguagem tratada na Seção 2.6 como proposta de extensão da OWL para a incorporação de tratamento de incerteza por uso de probabilidades. A representação de incerteza da PR-OWL segue o formalismo MEBN.

As razões que podem justificar o uso da abordagem probabilística são muitas. Dentre elas, podemos citar as seguintes:

- A incerteza é representada por valores numéricos limitados no intervalo [0,1]. Operações são fechadas nesse intervalo (*i.e.* qualquer operação resultará em um valor no intervalo [0,1]).
- Possui propriedades bem definidas e predizíveis matematicamente.
- Possui restrições suficientes para manter o resultado computável.
- É semanticamente bem definida.
- Intimamente ligada à "freqüência" ou "chance" de ocorrência; seja, está ligada à estatística, a observação dos fatos; logo, abstrai-se a capacidade de atualização das ontologias através da observação dos fatos.
- Comparado ao modelo fuzzy, pode ser mais facilmente estendido a domínios maiores, pois sua semântica é definida e a lógica fuzzy representa um conceito relativamente restrito de incerteza (o "vago").
- Desde que uma informação represente somente um (1) "significado" em um determinado momento, a probabilidade representa melhor a realidade do que a lógica fuzzy. Ilustrando esse exemplo, informações ambíguas com 50% de validade indicam que uma das interpretações é válida com 50% de chance (mas que somente uma das interpretações da instância será válida em um determinado momento).
- Como comparação, o "0.5" na lógica fuzzy indica que uma instância está "meio" verdadeira e "meio" falsa simultaneamente, enquanto que na probabilística ele indica que a instância fará parte da verdade na metade das ocasiões observadas (ou com a metade da chance total) - no final, a instância pertencerá a somente um dos conjuntos, nunca em ambos.

Uma linguagem de representação de ontologias não possui valor prático sem um modelo computacional de edição e inferência. O UnBBayes-MEBN realiza uma implementação de editor e máquina de inferência para a PR-OWL, linguagem de ontologia probabilística.

2.5 MEBN - Redes Bayesianas Multi-Entidade

Redes bayesianas Multi-Entidades (MEBN) são uma extensão das BNs via incorporação da FOL. O domínio de conhecimento em MEBN é expresso por MFrags (vide Seção 2.5.1),

que são organizados em MEBN *Theories* (ou MTheories). Uma MTheory é um conjunto de MFrags que satisfaz determinadas condições de consistência, que garantem a existência de uma distribuição de probabilidade conjunta única sobre suas RVs (68).

De forma radical, pode-se dizer que a MEBN é um *template* para a geração automática de SSBN a partir da entrada de dados (*i.e.* evidências - normalmente expressões em FOL). As SSBNs são equivalentes às BNs convencionais e representam o domínio da MEBN em um determinado momento específico.

As seções seguintes utilizarão o domínio de Identificação de Veículos, descrito em (78), para ilustrar e exemplificar o formalismo MEBN.

2.5.1 MFrags

MEBN representa o domínio de conhecimento através de conjuntos de MFrags. Um MFrag pode ser entendido como um fragmento de BN que representa a distribuição de probabilidades condicionais sobre instâncias de suas RVs residentes. Para que essa distribuição se aplique, restrições de contexto do domínio devem ser satisfeitas. Um conjunto de MFrags representa uma distribuição de probabilidade conjunta sobre uma quantidade potencialmente infinita de instâncias de RVs (33).

Dentro de um MFrag, os atributos ou os relacionamentos entre as entidades são expressos como nós residentes, que possuem um formato similar a funções ou predicados da FOL (*i.e.* é composto por um nome, um conjunto de valores possíveis e um conjunto de argumentos). As dependências probabilísticas entre nós residentes são representadas por arcos que interligam tais nós, e uma função matemática (similar a CPT, mas que define probabilidades de um **conjunto** de RVs) define quantitativamente a intensidade das dependências.

Dita de forma simplificada, pode-se entender que um MFrag é uma classe de sub-redes BN com características similares. Consequentemente, um conjunto consistente de MFrags compõe um *template* para a geração automática de BN, com base em regras e axiomas expressas em FOL. A Figura 2.5 apresenta um MFrag e seus principais componentes.

Esse MFrag possui doze nós (sete de contexto, três de entrada e dois residentes) e cinco variáveis ordinárias (`rgn`, `rpt`, `obj`, `t` e `tPrev`). Seguem abaixo as descrições sobre os componentes de um MFrag.

Nós de contexto (*context nodes*) são variáveis booleanas que representam condições que devem ser satisfeitas para que a distribuição de probabilidade dos nós residentes na MFrag de aplique. Seus valores possíveis são:

- *True*, quando a condição é satisfeita;
- *False*, quando a condição não é satisfeita;
- *Absurd*, quando uma expressão não tem sentido algum. Ocorre geralmente quando um valor é avaliado com parâmetros não previstos ou inválidos.

Nós de entrada (*input nodes*) são variáveis que influenciam a distribuição de probabilidades dos nós residentes, mas as suas distribuições estão definidas nos seus próprios MFrags. Ou seja, em uma MTheory consistente, se em um MFrag existe um nó de entrada, então é necessário que exista algum MFrag em que esse nó seja residente, para que sua distribuição de probabilidades

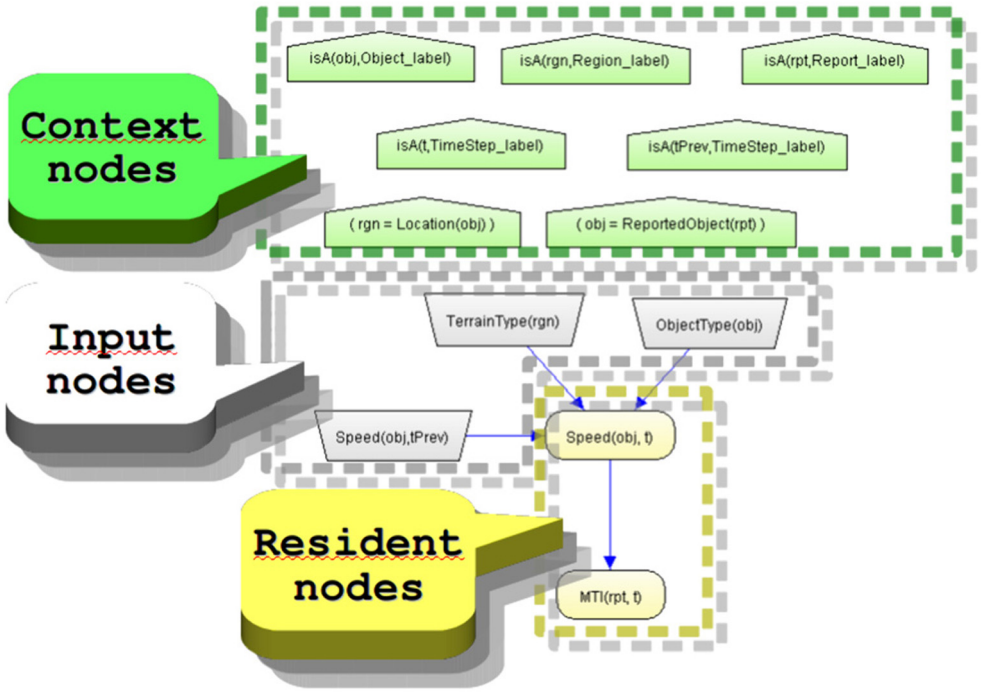

Figura 2.5: Figura exemplo de MFrag

seja bem definida.

Nós residentes (*resident nodes*) possuem as distribuições locais de probabilidades. Neles estão definidas as dependências probabilísticas dado os valores de seus pais (que podem ser tanto nós de entrada quanto outros nós residentes). Em uma MTheory completa, cada RV possui apenas um *home* MFrag, onde sua distribuição local é definida.

Variáveis ordinárias (*ordinary variables* - OV) são variáveis que se comportam de forma idêntica a variáveis da FOL e podem ser substituídos por instâncias de entidades. Em suma, são variáveis que não são RVs.

Para se realizar inferência, MFrags são instanciados tanto quanto necessários para se responder a uma determinada consulta. Quando um MFrag é instanciado, os argumentos de seus nós residentes (*i.e.* OVs) são substituídos por instâncias de entidades que satisfaçam os nós de contexto, resultando em uma RV da SSBN - que pode ser entendida como uma BN convencional.

Na Figura 2.5, observou-se que os valores dos estados dos nós não foram mostrados no grafo de MFrag. Isso porque um MFrag é apenas um *template*, ou seja, ele não representa RVs individuais, mas sim uma classe de RVs que podem existir na SSBN. Os valores reais dos estados só são concretizados ao se instanciar o MFrag.

A especificação da MEBN não define padrões para declarar a CPT em um nó de um MFrag (33). No UnBBayes-MEBN, utilizamos um pseudo-código para declarar as CPTs de uma classe de variáveis aleatórias (vide Seção 5.1.3).

Uma **distribuição padrão** deve ser declarada para o caso em que não exista, nos dados

fornecidos pelo usuário ou presentes na base de conhecimento, instâncias de entidades que satisfaçam a validade dos nós de contexto do MFrag. Nesse caso; a distribuição padrão, que não faz referência a nenhum estado dos pais, é utilizada. No UnBBayes-MEBN, essa distribuição padrão é especificada dentro do bloco "else" no fim do pseudocódigo (bloco obrigatório).

2.5.2 Recursividade em MEBN

A Figura 2.5 apresentou um exemplo de recursividade temporal, que é um caso bastante comum. Para que as distribuições se apliquem, ambos t e $tPrev$ precisam ser períodos de tempo (entidades *TimeStep*) e, além disso, $tPrev$ precisa ser um tempo anterior a t. Essa definição de ordem, indicando que t vem "depois" de $tPrev$, é implementada no UnBBayes-MEBN através da definição nativa de ordem de entidades. Se uma entidade (ou, por convenção, podemos chamá-lo de *tipo*) for declarada como ordenável, todas as suas instâncias (valores possíveis) possuirão ordem total. Com isso, torna-se desnecessária a criação de nós para a obtenção do tempo anterior (por exemplo, não é mais necessário criarmos um nó residente $previous(t)$, que retorna uma instância anterior a t).

Outros tipos de recursão também podem ser representados por MFrags com dependências entre instâncias (RVs) diferentes do mesmo nó residente (**OBS.** não são dependências entre mesmas instâncias do nó residente). É necessário que, ao se permitir definições de recursão, garanta-se que uma RV não possa depender, direta ou indiretamente, de sua própria distribuição de probabilidades, pois senão a BN representável por esse *template* poderia ser um grafo cíclico, desrespeitando um dos requisitos de uma BN (68).

Os nós de *input* de um MFrag podem incluir instâncias de nós definidos recursivamente no próprio MFrag. Por exemplo, o nó de entrada `Speed(obj, tprev)` representa a velocidade do objeto obj no período de tempo anterior ao t; e esse nó influencia a velocidade atual, dado pela variável `Speed(obj, t)`. A recursão precisa de uma distribuição de probabilidades no período incial (que, por convenção, chamamos de $T0$), que não dependa de alguma velocidade anterior, para que não tenhamos um laço infinito. No UnBBayes-MEBN, isso é garantido pelo fato de sempre existir um elemento mínimo em entidades ordenáveis[7].

A Figura 2.6 mostra como as definições de recursividade podem ser aplicadas para construir uma *SSBN*. Nesse caso, deseja-se saber qual foi o valor reportado por MTI no momento *T2* a um relatório *RPT1* sabendo-se que `OBJ1 = ReportedObject(RPT1)` e `RGN1 = Location(OBJ1)` (essas duas últimas expressões são consideradas evidências na MEBN).

O processo de construção do grafo mostrado na Figura 2.6 começa criando-se a instância de MFrag no nó `MTI(RPT1,T2)`, que é o nó cuja distribuição deseja-se conhecer. Instanciam-se, então, as variáveis aleatórias que sobraram, para as quais os nós de contexto estão satisfeitos. A seguir, são construídas as CPTs que podem ser obtidas com os dados já disponíveis (81).

Note que seria fácil especificar um conjunto de MFrags com dependências cíclicas, ou um conjunto tendo distribuições conflitantes para uma variável aleatória em diferentes MFrags. Na geração de SSBN, uma implementação deve detectar e impedir esses tipos de falhas (como faz o UnBBayes-MEBN[8]).

[7] A ordem total implica a presença de um elemento mínimo, garantindo que a recursividade pare

[8] O UnBBayes-MEBN realiza essa verificação em dois passos: estaticamente - garantindo que o editor não permita a inserção de variáveis inconsistentes - e dinamicamente - realizando o teste de consistência na etapa de instanciação

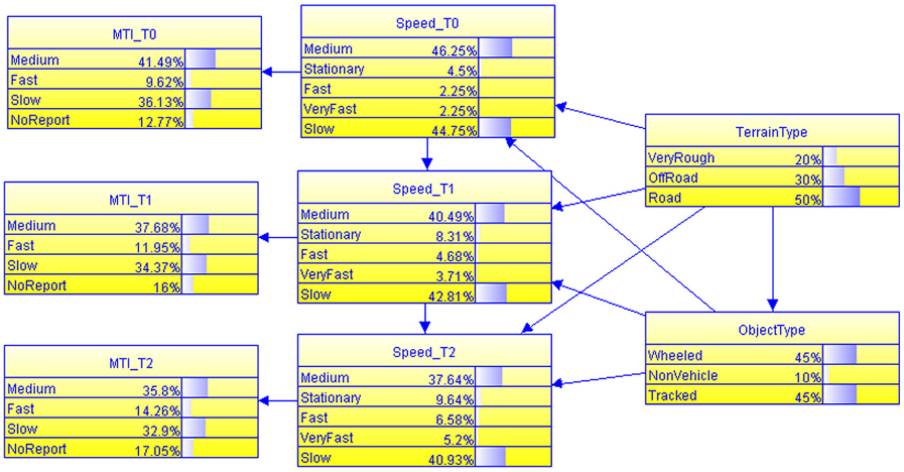

Figura 2.6: Modelagem de uma situação recursiva

Note também que em MEBN o MFrag inteiro é instanciado de acordo com a necessidade, garantindo assim um melhor controle e estruturação das variáveis "copiadas" (*i.e.* instanciadas). Como as instâncias são sub-redes similares com interface bem definida (*i.e.* as interfaces entre sub-redes são aquelas referenciadas por nós de entrada), gera-se uma estrutura de hiper-árvore (112) e a computação de probabilidades poderá teoricamente ser reaproveitada. Esse seria um diferencial comparado a DBN, por exemplo.

2.5.3 MTheories

Para construir um modelo coerente, é necessário que o conjunto de MFrags satisfaça coletivamente condições de consistência, garantindo a existência de uma distribuição de probabilidades conjunta única sobre as instâncias das variáveis aleatórias de MFrags. Esse conjunto coerente de MFrags é chamado de teoria MEBN (*MEBN Theory* - MTheory).

Uma MTheory representa uma distribuição de probabilidade conjunta única sobre um número (que pode tender ao infinito) de instâncias de suas variáveis aleatórias. Essa distribuição é especificada pelas distribuições locais e padrão de cada MFrag. Esses MFrags são parte de uma MTheory, que integra regras probabilísticas que caracterizam o domínio. Essas regras são capturadas e codificadas por um especialista ou via aprendizagem por máquina.

Cada variável aleatória deve ter um único *home* MFrag. Além disso, uma MTheory válida deve garantir que todas as definições de recursão terminem em passos finitos e não contenham influências circulares. Para que uma MTheory raciocine sobre situações particulares (*i.e.* gerar SSBN), é necessário informar o sistema sobre instâncias que estão envolvidas em uma determinada situação.

A inferência bayesiana pode ser usada tanto para responder a perguntas de interesse (chamadas de *query*) quanto para refinar uma MTheory (através de *findings*, ou evidências).

Findings é o mecanismo básico para incorporar novas informações em MTheories. Eles são representados por MFrags de apenas dois nós: um de entrada e um residente. Assim, MEBN tem a capacidade de incorporar novos axiomas quando novas evidências surgem, atualizando as

distribuições de todas as variáveis envolvidas na situação específica. São equivalentes à adição de um novo axioma na FOL.

A lógica MEBN contém um conjunto de *built-in MFrags* (ou seja, MFrags que devem ser embutidos na implementação, incluindo quantificadores, referência indireta e MFrags booleanos que representam conectivos, como "AND" e "OR"), que dão ao sistema a habilidade de representar qualquer sentença da FOL.

A Figura 2.7 mostra uma *MTheory* que representa o domínio de Identificação de Veículos.

O MFrag *Reference_MFrag* é usado para se declarar funções que associam duas entidades de tipos diferentes. Na verdade, como MEBN não é um sistema tipado, dever-se-ia criar MFrags contendo nós residentes definindo todos os tipos possíveis no domínio (*i.e.* definir nós residentes *isA* e suas distribuições). Contudo, como a implementação (UnBBayes-MEBN) assume tipagem forte por razões de eficiência, tais MFrags não precisaram ser criados. Todavia, a teoria pode acomodar qualquer mecanismo de sistema tipado, inclusive polimorfismo.

Vale notar que existem inúmeras maneiras de se representar uma MTheory de semântica equivalente. Por exemplo, podemos criar uma nova MTheory através da divisão de MFrags em MFrags menores (31).

2.6 Uma Ontologia Probabilística - PR-OWL

Uma ontologia probabilística é uma representação explícita e formal de conhecimento sobre um domínio de aplicação. Isso inclui:

- tipos de entidades presentes no domínio;
- propriedades dessas entidades;
- relações entre entidades;
- processos e eventos que ocorrem com essas entidades;
- regularidades estatísticas que caracterizam o domínio;
- conhecimento inconclusivo, ambíguo, incompleto, pouco confiável, ou dissonante sobre as entidades do domínio;
- incerteza sobre todas as formas do conhecimento citadas anteriormente (31).

Como previamente comentado na Seção 1.1, a Web Semântica é um modelo para permitir que máquinas integrem informações oriundas de múltiplas fontes (web), e ontologias comporão uma peça chave para a interpretação e operação de tais informações (6). A OWL e sua sucessora OWL 2 são linguagens recomendadas pela W3C para a representação de ontologias - uma especificação formal de significados e suas associações. Neste documento, iremos sempre nos referir à gramática da OWL (e OWL 2) como uma extensão da XML.

Apesar de popular, modelos representados em OWL não possuem suporte a tratamento de incertezas. A PR-OWL, proposta por Paulo Costa em 2005 (31, 34), estende a OWL para suporte à representação de incertezas. A lógica dessa linguagem baseia-se fielmente na MEBN, permitindo então a representação de modelos probabilísticos complexos. Por MEBN ser um

formalismo baseado na FOL, tornou-se um candidato natural na extensão de OWL, que se baseia em DL - subcategoria da FOL.

Em outras palavras, a PR-OWL é um formato para a aplicação da lógica MEBN em artefatos da Web Semântica (72), através da atribuição de significados especiais em classes e propriedades OWL para que estas passem a representar estruturas da MEBN (*e.g.* MFrags, nós residentes, dependências probabilísticas). Por tal motivo, necessita de máquina de inferência MEBN para o processamento da sintaxe adicional[9]. Esta linguagem foi projetada para representar diversos modelos bayesianos complexos. Como conseqüência, a PR-OWL pode ser considerada uma derivação da OWL-Full e na sua forma pura não há garantia de sua computabilidade (34).

Por ser sintaticamente compatível com OWL, o modelador pode utilizar os recursos novos da PR-OWL apenas para as partes da ontologia que necessitarem de suporte probabilístico. As partes probabilísticas da ontologia devem formar uma MTheory válida.

Por ser baseada na FOL, a PR-OWL não garante que consultas sejam tratáveis. Na OWL ocorre o mesmo problema e, no exemplo da OWL, foram criadas três versões diferentes da linguagem, com crescente poder expressivo, desenhadas para uso por comunidades específicas de desenvolvedores e usuários (OWL-Full, OWL-DL e OWL-Lite). Seguindo o mesmo princípio, uma restrição da PR-OWL poderia ser criada (*e.g.* PR-OWL Lite), mas isso ainda não foi especificada oficialmente (34).

A Figura 2.8 mostra os principais conceitos envolvidos na definição de uma MTheory em PR-OWL. No diagrama, as elipses representam as classes gerais, enquanto que os arcos representam os principais relacionamentos entre as classes. Uma ontologia probabilística deve ter ao menos um indivíduo da classe MTheory, sendo que este é ligado com um grupo de MFrags que coletivamente formam uma MTheory válida. MFrags são compostos por nós, sendo que estes podem ser residentes, de contexto, ou de entrada. Cada indivíduo da classe *Node* é uma RV e esta possui um conjunto mutuamente exclusivo e coletivamente exaustivo de possíveis estados. Cada variável aleatória possui uma distribuição de probabilidade condicional ou incondicional (34).

A Figura 2.9 mostra uma versão cobrindo os elementos principais da PR-OWL, suas subclasses e os elementos secundários necessários para representar MTheory e as relações que são necessárias para expressar a estrutura complexa dos modelos probabilísticos bayesianos usando a sintaxe da OWL.

CPTs na PR-OWL também podem ser declaradas respeitando-se a gramática adotada pelo UnBBayes-MEBN, descrita na Seção 5.1.3. No entanto, como em MEBN, não existe um único padrão para se especificar as distribuições de probabilidade condicional em PR-OWL.

Construir MFrags e todos os seus elementos em uma ontologia probabilística é um processo cansativo e sujeito a erros, exigindo um conhecimento profundo da estrutura de dados da PR-OWL. Tornou-se necessário o desenvolvimento de um software que permite uma modelagem visual, ou um *plug-in* para alguma das ferramentas bayesianas existentes. O módulo UnBBayes-MEBN possui funcionalidades de edição de arquivos PR-OWL, permitindo que a MTheory seja salva neste formato, fornecendo ao modelador um meio visual, intuitivo e guiado para a construção das ontologias probabilísticas.

[9] O UnBBayes-MEBN é pioneira no tratamento da PR-OWL.

2.7 PR-OWL 2 - Integrando Conhecimento Probabilístico e Determinístico

Apesar da PR-OWL ser uma linguagem com alta expressividade, a cooperação entre o raciocínio determinístico (oriundo de OWL) e probabilístico (oriundo dos novos elementos oferecidos por PR-OWL) era bastante limitada; seja, informações especificadas (e inferidas) em OWL convencional não podiam ser eficientemente reaproveitadas em aplicações PR-OWL, por causa da falta de um mapeamento em nível metalinguístico.

Claro, isso não seria um problema se todos os domínios fossem modelados em PR-OWL "a partir do zero". Todavia, muitas ontologias no mercado já foram modeladas em OWL, e como previamente comentado na Seção 2.3.4, o reuso seria uma opção mais natural.

Em teoria, qualquer domínio representado em OWL pode ser representado em MEBN (e consequentemente em PR-OWL), bastando utilizar somente 0% e 100% como valores das probabilidades. No entanto, essa "tradução" de conhecimento OWL para MEBN/PR-OWL exige um esforço considerável e presença de um especialista em ambos formatos. Para o contorno deste problema, foi formulado a PR-OWL 2 (19).

Na verdade, o problema da falha na integração do conhecimento probabilístico da PR-OWL com o determinístico da OWL também já fora mencionado em outros trabalhos na literatura:

> "PR-OWL does not provide a proper integration of the formalism of MEBN and the logical basis of OWL on the meta level. More specifically, as the connection between a statement in PR-OWL and a statement in OWL is not formalized, it is unclear how to perform the integration of ontologies that contain statements of both formalisms." (96)

Em outras palavras, os elementos da PR-OWL não se mapeavam explicitamente a elementos de OWL. Esta observação se resume principalmente, mas não integralmente, nos seguintes pontos:

1. meta entidades (ou simplesmente "entidades") da PR-OWL representam os componentes conceituais básicos de um domínio (seja, "tipos"), mas não eram compatíveis com os tipos preexistentes em OWL (i.e. classes);
2. as "propriedades" da OWL (que são elementos que representam um atributo de uma classe ou seus relacionamentos) não possuem ligações explícitas com nós residentes (que representam relações entre entidades MEBN). Essas questões dificultavam a inferência conjunta e o reuso de conhecimento expresso em OWL.

Mas afinal, para quê precisaríamos mapear um nó residente a algum elemento da OWL? As Figuras 2.10 e 2.11 ilustram respectivamente um modelo determinístico e probabilístico de um pequeno subdomínio de licitação pública CGU (vide Capítulo 7). Supondo que exista uma base determinística de conhecimento (e.g. ontologia OWL) que liste quais empresas e pessoas existem no domínio, é interessante que o modelo probabilístico (Figura 2.11) possa também utilizar essa informação para inferir o valor de `IsFrontFor` (que possui empresa e pessoa como argumentos).

Esse tipo de raciocínio somente é possível se existir um mapeamento explícito que indique que uma "pessoa" (ou "empresa") do modelo probabilístico é também uma "pessoa" (ou "empresa") do modelo determinístico. É mais interessante ainda se um nó residente - e.g. valueOf

(procurement) da Figura 2.11 - seja restrito somente a valores obteníveis de uma propriedade do modelo determinístico - e.g. propriedade procurement que conecta Procurement com Contract, na Figura 2.10. Este segundo é possível somente se existir um mapeamento entre um nó residente da PR-OWL com uma propriedade da OWL.

Portanto, para o contorno dos problemas de integração da PR-OWL, um dos princípios adotado em PR-OWL 2 segue basicamente a ideia de se criar um link do nó residente (PR-OWL) para uma propriedade (OWL) usando uma nova propriedade OWL com nome defineUncertaintyOf. As "entidades" da PR-OWL (que representam "entidades" da MEBN) serão eliminadas da linguagem, religando-se todas as referências a essas entidades para as classes OWL (ou seja, todos os elementos da PR-OWL que utilizavam as entidades passam agora a utilizar classes OWL diretamente), eliminando assim a necessidade de um mapeamento explícito entre uma entidade (PR-OWL) com classes OWL. Figuras 2.12, 2.13 e 2.14 ilustram o mapeamento de um nó residente para propriedades binárias funcionais, ternárias funcionais e binárias não funcionais, respectivamente.

As principais diferenças entre a PR-OWL (versão 1) e a PR-OWL 2 estão resumidas na listagem abaixo:

- PR-OWL 2 permite link entre RV a propriedades OWL para indicar que representam idéias iguais;
- PR-OWL exige que "entidades" (as conceitos descritos em MEBN) sejam subclasses de `Entity`, mas PR-OWL 2 permite qualquer classe;
- PR-OWL 2 separa RVs de nós da MEBN, aumentando a flexibilidade;
- PR-OWL 2 permite especificar a distribuição padrão (distribuição de probabilidades utilizada pelas RVs quando nós de contexto falham) explicitamente na RV[10];
- PR-OWL 2 permite especificar nós e RVs usando expressões FOL mais complexas;
- PR-OWL 2 é especificada em OWL 2, enquanto que PR-OWL 1 é em OWL 1.

Como comentado anteriormente, uma inferência MEBN consiste na criação de uma SSBN, mas para a tal é necessário realizar diversas consultas em uma base de conhecimento FOL para se retribuir instâncias de entidades que "casam" com as expressões presentes nos nós de contexto. Obviamente, se a base FOL não poder responder, essa expressão é considerada como "desconhecida" e é tratada probabilisticamente (*i.e.* convertida para nós na SSBN).

Por se basear em MEBN, a inferência em PR-OWL 2 segue o mesmo raciocínio. No entanto, graças a ligação entre RVs e propriedades OWL, torna-se possível traduzir as funções e predicados (*i.e.* RVs) dos nós de contexto para propriedades OWL. Entidades de MEBN e seus valores também podem ser traduzidas para classes OWL e seus indivíduos.

Consequentemente, consultas que antes eram realizadas na base de conhecimento FOL podem agora ser redirecionadas para bases de conhecimento em DL (*i.e.* tratadas por máquinas de inferência DL), desde que a expressão no nó de contexto seja tratável em DL[11]. Tal redireci-

[10]Com isso, a distribuição padrão não precisa ser especificada como um bloco "else" no final do pseudocódigo que especifica a distribuição de probabilidades no UnBBayes-MEBN.

[11]Como a expressividade de DL é menor do que a da FOL, nem sempre as expressões nos nós de contexto são tratáveis por máquinas de inferência DL. No entanto, implementações podem impor essa restrição para se garantir computabilidade e eficiência.

onamento permite integração da inferência probabilística (geração de SSBN) e determinístina (consultas DL) na PR-OWL 2.

Em suma, dado a importância do mapeamento formal entre OWL e PR-OWL, está formulado aqui uma das maiores alterações sintáticas da PR-OWL versão 2: a associação simbólica[12] entre nós residentes da PR-OWL às propriedades da OWL, permitindo uma navegação bidirecional entre os referidos elementos. Obviamente, outras alterações foram propostas em PR-OWL 2, como eventuais alterações em nomes de classes ou propriedades e criação de novas classes adaptadoras para permitir a representação de estruturas MEBN mais complexas que não eram possíveis na PR-OWL antiga. A ontologia base do PR-OWL 2 já está disponível como recurso em <http://www.pr-owl.org/pr-owl2.owl>; portanto, detalhes sobre classes, pripriedades e restrições já estão disponíveis em público.

Uma ontologia OWL 2 torna-se uma ontologia PR-OWL 2 ao se importar a ontologia de definição `pr-owl2.owl`, disponível também em <http://www.pr-owl.org/pr-owl2.owl>. Mecanismos de importação podem ser utilizados novamente para se integrar outros conhecimentos expressos em ontologias OWL 2 também. Essa capacidade de importação, inerente da linguagem OWL, pode facilitar em outras tarefas de integração de modelos também.

2.8 Outras Abordagens

Historicamente, abordagens numéricas[13] para a representação de incertezas têm recebido atenção graças à sua habilidade de classificar finamente a plausibilidade em contextos e hipóteses, com base em cálculo formal para inferência. Diversos modelos têm sido adotados por pesquisadores e engenheiros para representar incerteza, como funções de crença de *Dempster-Shafer* (39, 99). Aplicações de fatores de incerteza também existem (10), mas sabe-se que fatores de incerteza só sucedem quando utilizados com extrema cautela.

Modelos baseados em lógica *fuzzy* também têm se tornado popular na representação e processamento de grau de veracidade em informações linguísticas e têm se tornado um forte candidato no tratamento de incerteza oriunda de linguagens naturais: *Fuzzy RDF* (abordagem que incorpora lógica *fuzzy* em RDF) (82), f-$\mathcal{SHIN}$ (junção de DL com lógica *fuzzy*) (102) e *fuzzyDL* (máquina de inferência de DL e lógica *fuzzy*) (8). No entanto, a lógica *fuzzy* não possui uma semântica bem definida, exigindo mutas vezes que operadores sejam selecionados de forma semi-artesanal.

Finalmente, os modelos probabilísticos [14] (onde BN e consequentemente a MEBN se enquadram) são também fortes candidados para tratamento de proposições incertas, graças à sua representação plausível e cálculo formal. Diversas extensões de BN foram propostas para impulsionar sua expressividade, como DBN (86), *Object Oriented Bayesian Networks* (61), PRM

[12] O link entre um nó residente e uma propriedade OWL é simbólica por questões técnicas, para que uma ontologia consistente em PR-OWL 2 também a seja em OWL com DL.

[13] Uma abordagem numérica representa a incerteza usando numerais (valores pontuais ou intervalos) para se expressar graus de certeza (*i.e.* uma afirmação pode ser mais plausível ou não). Normalmente, esses valores se compreendem entre zero (0) e um (1) e operações especiais são aplicadas para se permitir inferência.

[14] A interpretação da "probabilidade" pode variar de grupos em grupos (*e.g.* os *frequentistas* interpretam a probabilidade como uma frequencia relativa entre uma grande quantidade de tentativas), mas nesta dissertação estamos usando o termo "probabilidade" na interpreção epistemológica bayesiana, que considera a probabilidade como uma medida do estado do conhecimento (*i.e.* grau de crença).

(51), sugerindo a BN como uma base sólida para raciocínio plausível em sistemas de conhecimento independente de domínio.

Naturalmente, existem modelos que incorporam probabilidades em ontologias, de maneira diferente da MEBN/PR-OWL: ACP (extensão probabilística de DL) (56), *pdl-program* (*Probabilistic Description Logic Programs*, uma abordagem que junta DL com lógica de escolha independente de Poole (95)) (73) e P-$\mathcal{SHOQ}$(D) (incorporação de probabilidades a DL $\mathcal{SHOQ}$(D)) (52).

Modelos baseados em redes de Markov também são fortes candidados à representação de modelos probabilísticos de alta expressividade, como ocorre com *Markov Logic Network* (MLN) (42, 98). Por ser uma rede sem orientação, redes de Markov, de forma geral, permitem a especificação de relacionamentos bi-direcionais de forma natural. Entretanto, essa característica também dificulta na especificação de probabilidades sob hipóteses, por não existir uma ordem que condiciona as variáveis. Adicionalmente, por se especificar as probabilidades indiretamente através de "pesos" em expressões FOL, a MLN é menos intuitiva para pessoas que costumam raciocinar sob contextos (no entanto, a MLN oferece um poderoso suporte à aprendizagem por máquinas).

Apesar de existirem diversas abordagens concorrentes, a MEBN/PR-OWL (e PR-OWL 2) foi selecionada nesta pesquisa por representar o conhecimento em blocos organizados (MFrags) em que o conhecimento é modelado de forma gráfica e intuitiva (nós conectados por arcos e especificando-se funções de probabilidade condicional - um modelo de raciocínio sob hipóteses) com restrições no domínio enfatizadas na forma de nós de contexto. A facilidade de visualização do conhecimento de forma gráfica (garantida pela própria lógica MEBN) é um grande diferencial.

2.8.1 Exemplos de Implementações Relacionadas

Alguns formalismos relacionados a MEBN e PR-OWL já foram apresentados anteriormente, e trabalhos que utilizam ou são diretamente influenciados pelo projeto UnBBayes são apresentados na Seção 6.4. Portanto, esta seção focará apenas na apresentação de aplicações concretas "similares" ao UnBBayes; ou seja, softwares que também implementam modelos probabilísticos ou união de ontologias.

Ferramentas para manipulação de modelos probabilísticos não são novos, e uma variedade de abordagens similares foram desenvolvidas nos últimos anos. Esses trabalhos podem ser normalmente classificados (mas não limitados) a API, ambiente de desenvolvimento com GUI, implementações de modelos com níveis mais altos de expressividade e ferramentas para data mining. Discutimos alguns nesta seção.

Free-BN[15] é uma API em linguagem Java para aprendizagem em BN, licenciada sob licença Apache 2.0.

Em contraste ao Free-BN, projetado para ser uma API, UnBBayes oferece adicionalmente uma GUI para ajudar os usuários na manipulação de BN com mínimo esforço. Esta abordagem pode levar a uma menor performance, ou aumento no tamanho do programa, mas não precisa confiar nas habilidades de programação do usuário para ser executada. Obviamente, existem outras abordagens que também provêm um ambiente com GUI, como as seguintes:

[15] http://vangjee.wordpress.com/2009/04/09/open-source-bayesian-network-structural-learning-api-free-bn/

GeNIe[16] é um ambiente de desenvolvimento voltado para a criação de modelos gráficos de decisão. Foi desenvolvida na *Decision Systems Laboratory*, da Universidade de Pittsburgh, e é implementada em Visual C++.

Comparada a implementações em Java, as implementações em C++ são mais rápidas, mas em contrapartida sua portabilidade é comprometida, pois não são executáveis em diferentes sistemas operacionais. No caso de GeNIe, sua dependência a bibliotecas da Microsoft o torna mais difícil ainda de ser executado em diferentes plataformas, pois até a compilação é difícil em sistemas não-Microsoft. Como o UnBBayes é escrito em Java, portabilidade não é um problema tão grave comparado a implementações em C++. Claro, existem outras alternativas em Java para modelos probabilísticos gráficos, como segue:

BNJ[17] ou "*Bayesian Network tools in Java*" é uma *suite* de código aberto para modelos gráficos probabilísticos. É publicado pelo "*Laboratory for Knowledge Discovery in Databases*" (KDD) da Universidade Estadual de Kansas.

Carmen[18] é um software livre de código aberto, escrito em Java, para modelos probabilísticos gráficos. Foi desenvolvido pela UNED (*Madrid*, Espanha) e oferece um ferramental para manipular BN e ID.

SamIam[19] é uma ferramenta para modelagem e raciocínio em BN, desenvolvida em linguagem Java por *Automated Reasoning Group* do professor Adnan Darwiche da Universidade de Califórnia, Los Angeles (UCLA).

Os três produtos acima são implementações de código aberto de BN (e ID) bem poderosas, que oferecem uma suite com GUI e máquina de inferência. Entretanto, eles não oferecem a flexibilidade no nível oferecido pela arquitetura de plug-ins do UnBBayes, nem suporte a ontologias e linguagens probabilísticas de primeira ordem (que é oferecido pelo UnBBayes graças ao repositório de plug-ins já existentes).

Existem implementações comerciais de BN também:

Netica[20] é um programa poderoso para se trabalhar com redes de crenças/probabilidades e ID. Uma versão *demo* está disponível também para *download*. Disponibiliza API para diversas plataformas, como Java, C, C#, Visual Basic, C++, Matlab, e CLisp.

HUGIN Este é um produto comercial, mas uma versão limitada está disponível também para *download*.

Implementações comerciais oferecem um rico repositório de funcionalidades que batem com a real demanda do mercado, uma vantagem que não pode ser oferecida pelo UnBBayes. No entanto, por questões de licença, o acesso ao código fonte é normalmente difícil. Como o UnBBayes é um *framework* de código aberto (licenciado sob GPL), o código fonte está disponível livremente.

Existem também ferramentas que focam em modelos probabilísticos de alta expressividade, como as seguintes:

[16] http://genie.sis.pitt.edu/
[17] http://sourceforge.net/projects/bnj/
[18] http://www.cisiad.uned.es/carmen/index.html
[19] http://reasoning.cs.ucla.edu/samiam/
[20] http://www.norsys.com/netica.html

Alchemy[21] é um pacote de software que provê uma série de algoritmos para aprendizagem estatística relacional e inferência lógica-probabilística. Baseia-se em *Markov Logic Networks* (MLN) (98). É implementado basicamente em C++.

Tuffy[22]é uma máquina de inferência de código aberto, escrita em Java, para o formalismo MLN. Faz parte do projeto *DARPA Machine Reading* e é gerenciada pela *Air Force Research Laboratory* (AFRL).

PROXIMITY[23] é uma platafora de código aberto em Java para dados relacionais e descoberta de conhecimento com modelos probabilísticos. Foi projetado e implementado por *Knowledge Discovery Laboratory* do departamento de ciência da computação da Universidade de Massachusetts Amherst.

Primula[24] é uma ferramenta Java, distribuída sob GPL, para modelagem probabilística de estruturas relacionais. Utiliza o SamIam internamente para permitir inferência exata. A versão 2.2 inclui implementação de MLN também.

Essas ferramentas foram projetadas para realizarem inferências probabilísticas em domínios que exigem expressividade extra. UnBBayes ainda não oferece suporte a MLN ou suporte completo a dados relacionais (são funcionalidades ainda em construção por membros do GIA/UnB). No entanto, ele oferece suporte a outros tipos de modelos probabilísticos de alta expressividade, como OOBN, MEBN, e um subconjunto de PRM com inferência limitada[25].

Vale observar que as ferramentas de edição de BN (ou extensões) apresentadas nesta seção oferecem pouca ou nenhuma funcionalidade para se especificar a distribuição de probabilidades condicionais de RVs de forma tão dinâmica como no UnBBayes-MEBN, que interpreta uma gramática especial para a geração dinâmica das distribuições (vide Seção 5.1.3). Definitivamente, esse é um diferencial no UnBBayes.

Por outro lado, existem abordagens que focam mais em mineração de dados. Apesar de algumas das abordagens anteriores também oferecerem suporte a construção de modelos a partir de dados, ferramentas especializadas em mineração de dados oferecem um suporte mais completo para manipulação de dados massivos.

Weka[26] é um software de código aberto, escrito em Java. Oferece uma coleção de algoritmos de aprendizagem e interface gráfica para tarefas de mineração de dados.

UnBBayes, por si, não é uma ferramenta para mineração de dados, mas o plug-in UnBMiner (64) consegue adicionar uma coleção de funcionalidades para manipulação de dados massivos e aprendizagem por máquina.

Por fim, existem ferramentas especializadas em mapeamento, alinhamento ou fusão de ontologias (nos sentidos apresentados na Seção 2.3.4). Seguem abaixo:

[21]http://alchemy.cs.washington.edu/

[22]http://research.cs.wisc.edu/hazy/tuffy/

[23]http://kdl.cs.umass.edu/proximity/proximity.html

[24]http://www.cs.aau.dk/ jaeger/Primula/

[25]A inferência de PRM no UnBBayes é realizada convertendo-se uma PRM em BN, de acordo com os dados presentes no momento.

[26]http://www.cs.waikato.ac.nz/ml/weka/

PROMPT[27] é um framework implementado como extensão (plug-in) do Protégé 3 e permite gerenciamento de múltiplas ontologias. Oferece ferramentas como IPROMPT (fusão interativa de ontologias), ANCHORPROMPT (alinhamento), PROMPTDIFF (versionador) (88).

FOAM(*Framework for Ontology Alignment and Mapping* é uma implementação Java para alinhamento de ontologias, basicamente em OWL-DL (44). É uma implementação que aproveita bastante as informações em rótulos ou identificadores para processamento.

GLUE é um sistema que emprega tecnologias de aprendizagem de máquina para criar mapeamentos entre ontologias de forma semi-automática, com base em dados de instância (41).

OntoMerge utiliza abordagem de ontologias ponte (vide Seção 2.3.4) para fusão de ontologias, mantendo as ontologias originais em uso (43).

Diferente dessas ferramentas, o UnBBayes (e seus plug-ins) não realiza mapeamento tampouco alinhamento de ontologias[28], nem é exatamente voltado à fusão.

Explicando de forma mais objetiva, um plug-in do UnBBayes (criado nesta pesquisa) implementa a PR-OWL 2, cuja funcionalidade é representar uma ontologia probabilística, que pode eventualmente especificar regras probabilísticas que reconciliam semelhanças/diferenças entre informações em ontologias OWL, permitindo então a fusão de ontologias.

Portanto, a habilidade do UnBBayes de fundir ontologias é somente uma consequência da PR-OWL 2. No entanto, a possibilidade de se especificar relacionamentos entre informações presentes em diferentes ontologias (não somente semelhanças/diferenças) de forma plausível e baseada em forte fundamentação teórica - MEBN - é uma grande vantagem da PR-OWL 2 (e do UnBBayes).

O UnBBayes também oferece um certo nível de suporte a tradução de linguagens de representações de conhecimento (no sentido amplo); como por exemplo, suporte a diferentes formatos de armazenamento de BN (Seção 4.2) e tradução limitada de expressões FOL em DL (Seção 5.3.3). Nesse sentido, **OntoMorph**, ferramenta que oferece funcionalidades para reescrita (transformação) de linguagens de representação de conhecimento (24), está relacionada ao UnBBayes. OntoMorph alega integração com PowerLoom (base de conhecimento utilizado pelo UnBBayes-MEBN antes da refatoração - vide Seção 5.1) e oferece API para LISP, C++ e Java; no entanto, por falta de documentação técnica, difícil acesso à API e por não ser um projeto ativo, não foi utilizado como biblioteca no UnBBayes.

Note que o UnBBayes é um *framework* e um ambiente de software, portanto não está limitado a um modelo em particular ou voltado a um único propósito. Neste sentido, ele pode inclusive ser comparado a softwares como o Protégé ou até Eclipse, se o ponto discutido for "um ambiente multi-uso com suporte a plug-ins". Obviamente não é a intensão do projeto UnBBayes em competir com produtos tão concebidos. O foco central do UnBBayes está em oferecer um framework (voltado a plug-ins) similar a Protégé ou Eclipse, mas em um nicho específico: edição de modelos probabilísticos (*e.g.* BN, ID, ontologias probabilísticas) e inferência.

[27] http://protegewiki.stanford.edu/wiki/PROMPT

[28] Teoricamente, como o UnBBayes possui a habilidade de carregar indiretamente plug-ins do Protegé 4.1 (vide Seção 5.3.4), a compatibilização do PROMPT para o Protégé 4.1 (que aparentemente ainda não foi realizada) automaticamente habilitaria o mapeamento/alinhamento de ontologias no UnBBayes também.

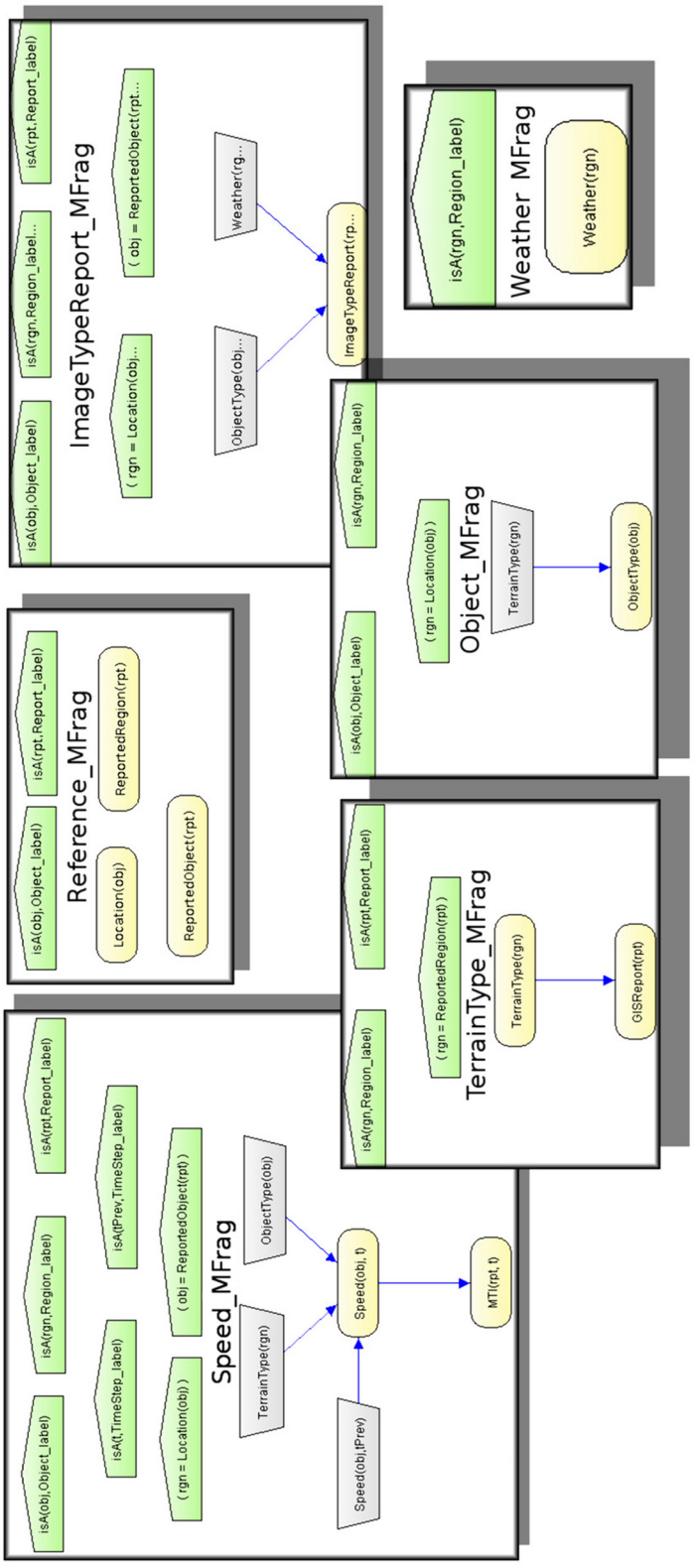

Figura 2.7: Conjunto de MFrags que formam uma MTheory (78)

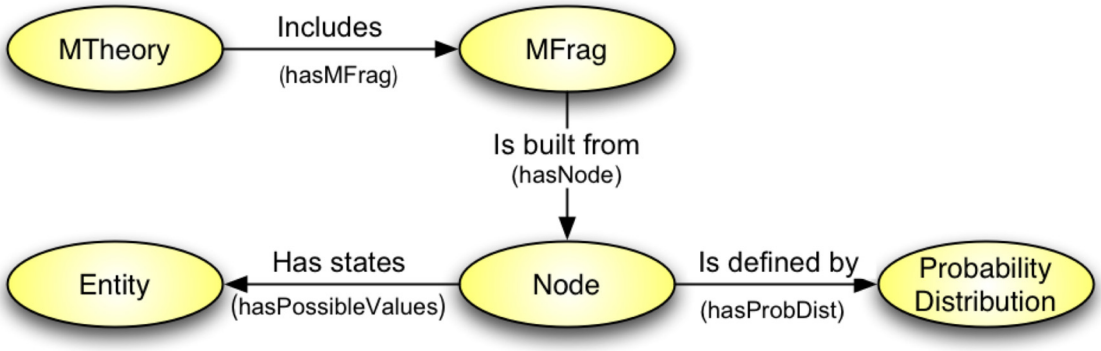

Figura 2.8: Principais conceitos da PR-OWL (34)

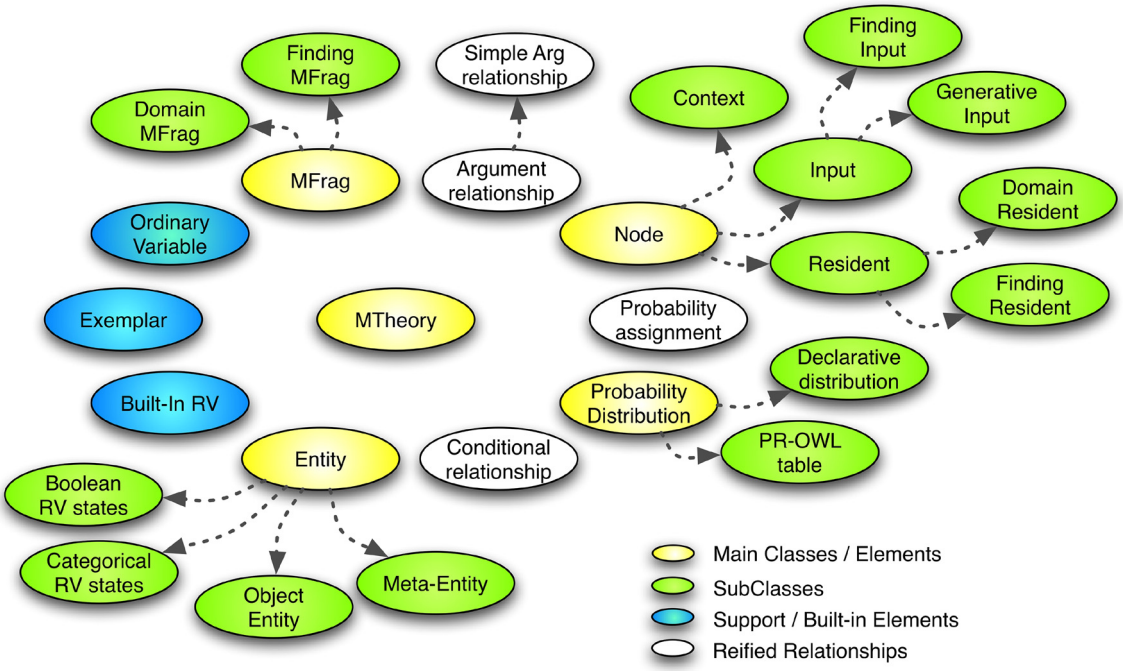

Figura 2.9: Elementos de uma ontologia probabilística PR-OWL (34)

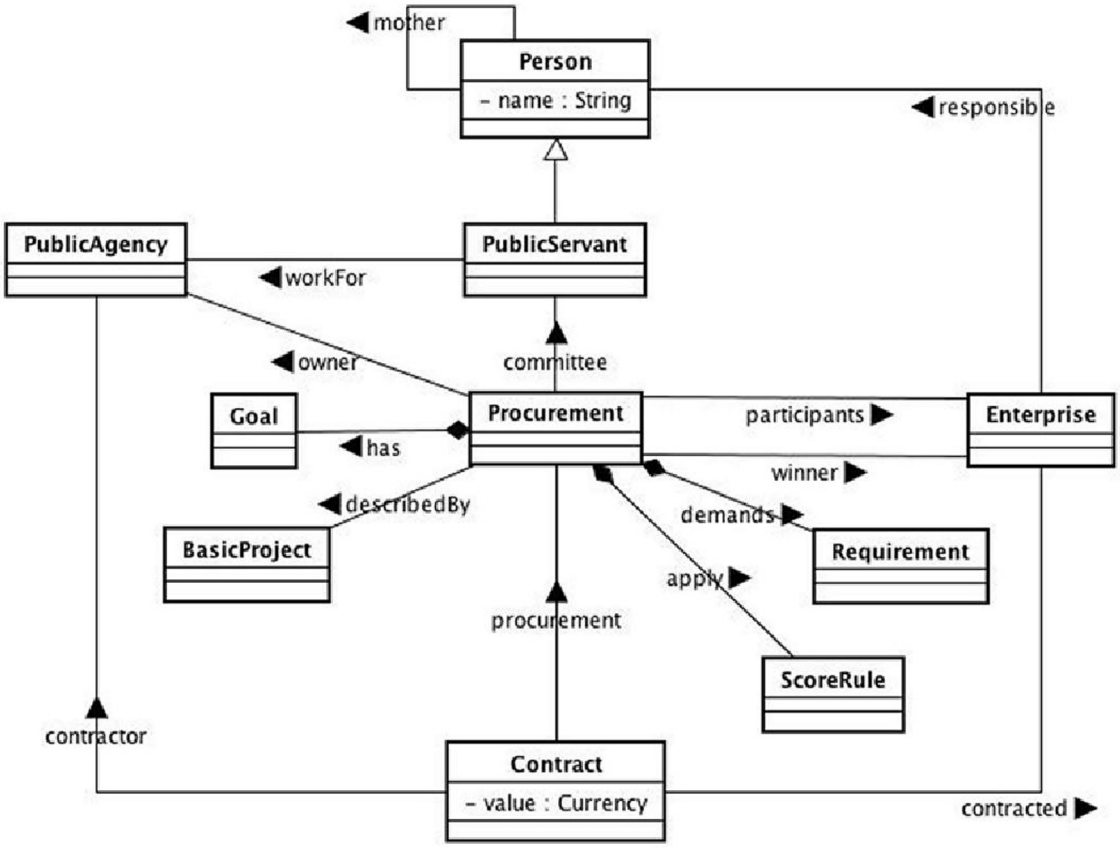

Figura 2.10: uma ontologia do domínio da licitação pública (sem incertezas), expressa em UML (extraída de (19)).

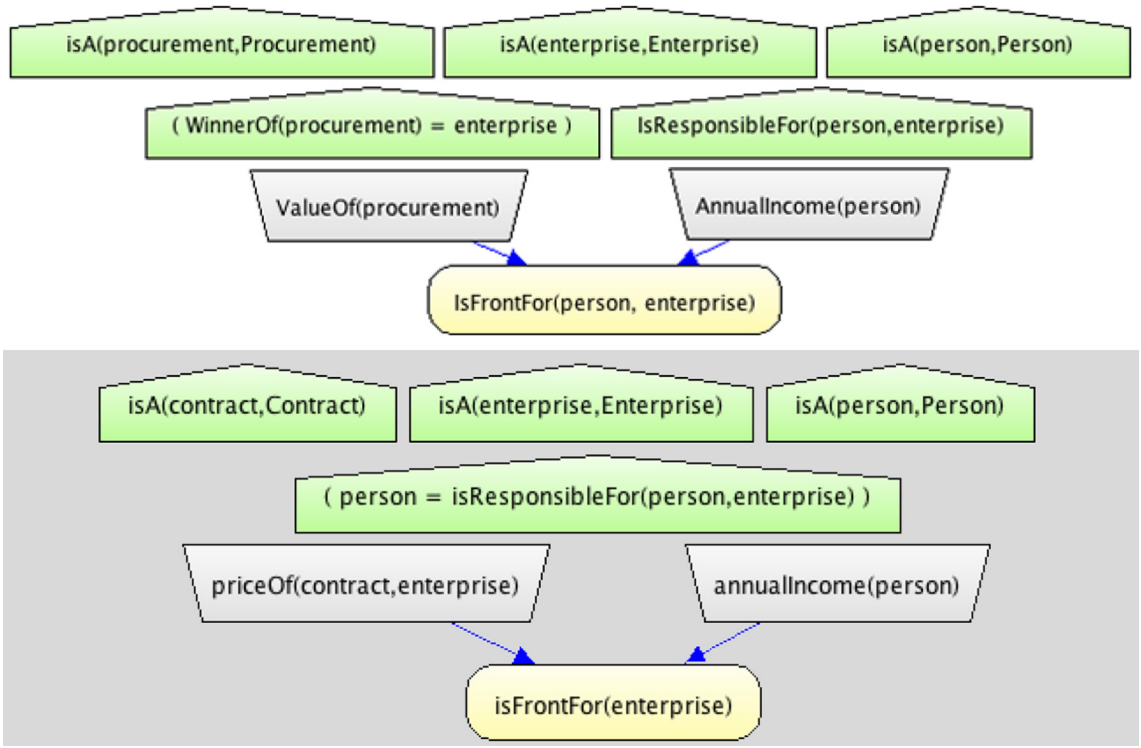

Figura 2.11: exemplo de modelagem probabilística do domínio de licitação pública, baseada em MEBN/PR-OWL, extraída de (19).

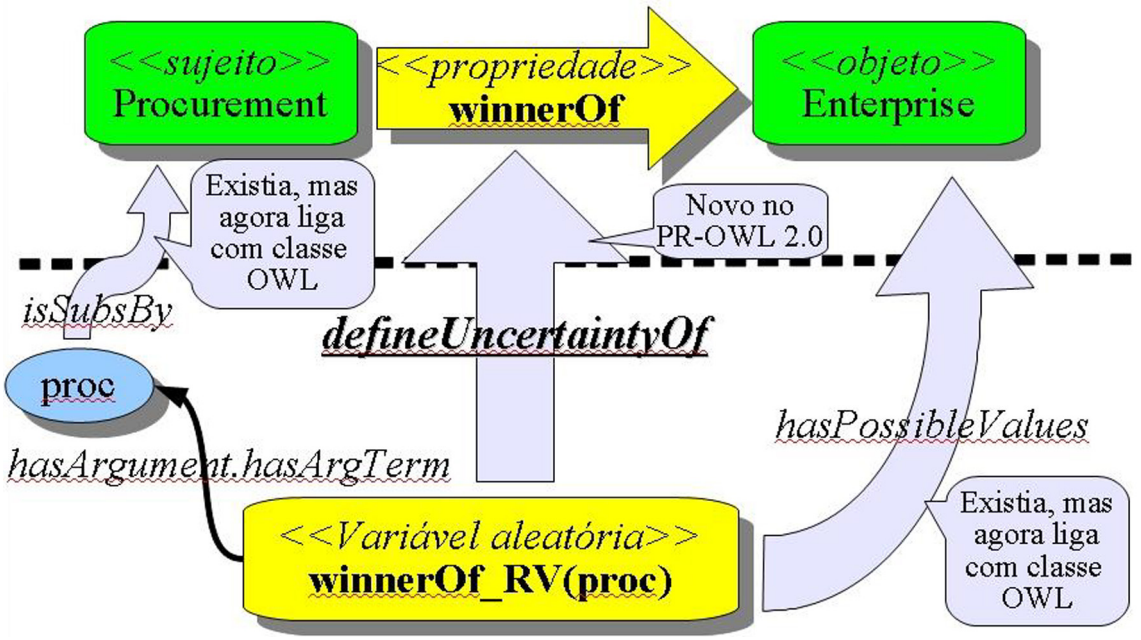

Figura 2.12: Mapeamento do nó residente para uma propriedade binária.

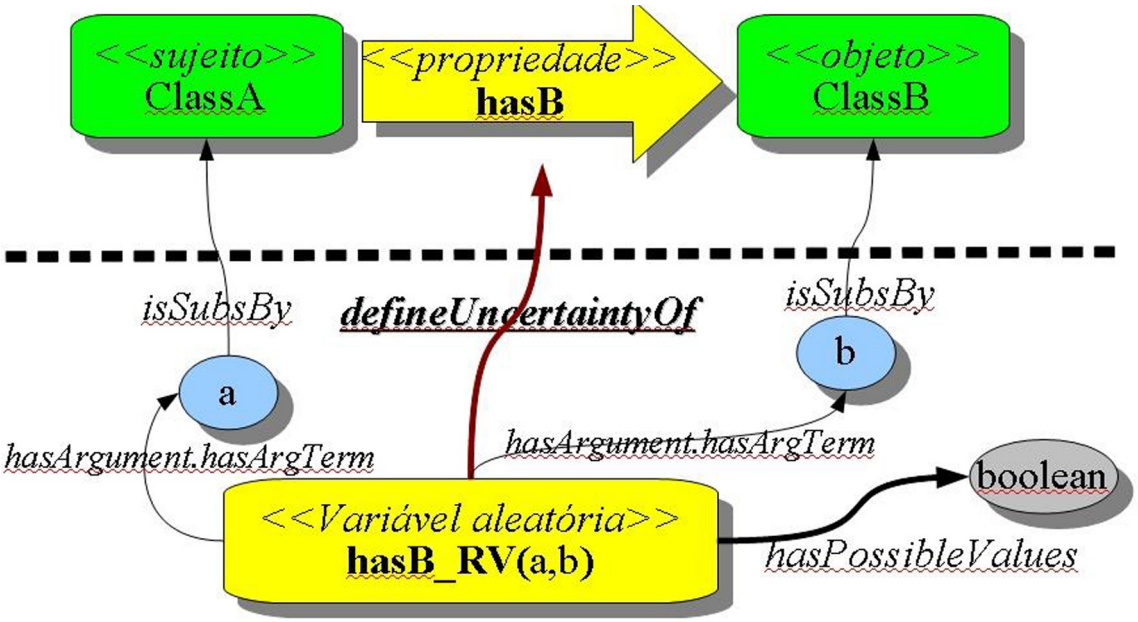

Figura 2.13: Mapeamento do nó residente para uma propriedade não funcional.

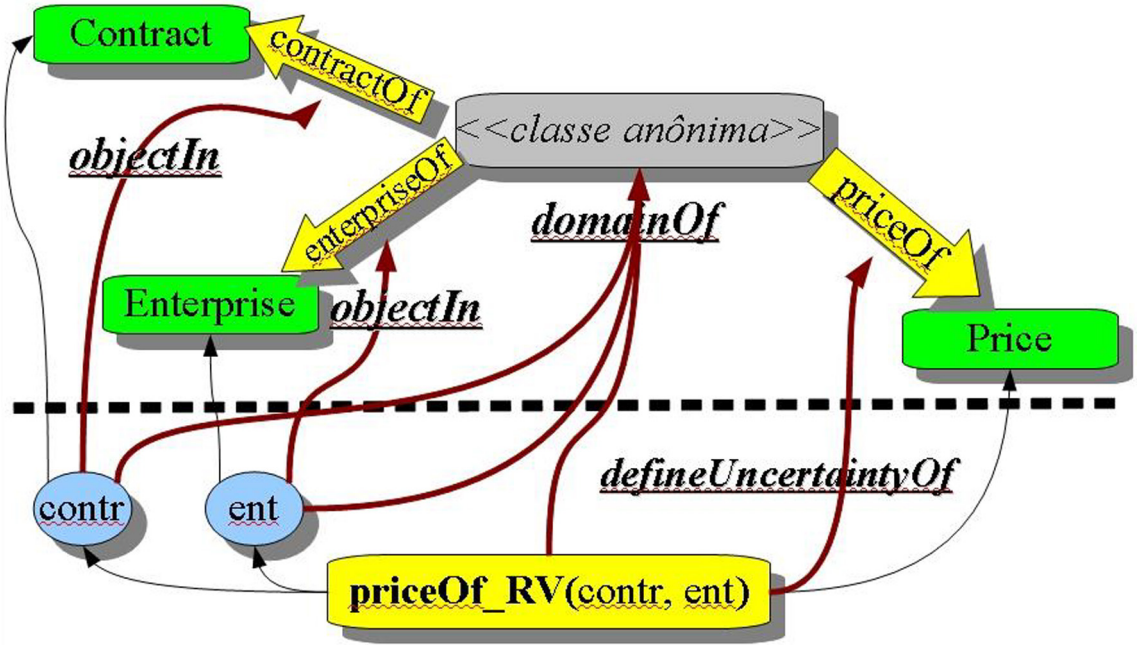

Figura 2.14: mapeamento do nó residente para uma propriedade ternária (exemplo de mapeamento para propriedades n-árias).

Capítulo 3

Arquitetura da Solução Proposta

A linguagem Java, com seu paradigma orientado a objeto e execução em uma máquina virtual, permitiu a criação de uma arquitetura flexível e independente de domínio em sistemas de IA com suporte a raciocínio intensivo em bases de conhecimento. No entanto, com o crescimento do tamanho e complexidade dos sistemas, a orientação a objeto e as outras facilidades oferecidas pela linguagem tornaram, por si só, insuficientes para se garantir a qualidade de softwares. Técnicas adicionais baseadas em ES tornaram-se imprescindíveis para se garantir um nível razoável de produtividade e manutenibilidade, principalmente quando o foco se expande a uma família de softwares.

Neste capítulo, as técnicas estudadas durante a pesquisa de mestrado são aplicadas para a modelagem da arquitetura básica de plug-ins. Naturalmente, para uma melhor compreensão da arquitetura da solução, é interessante conhecer primeiro alguns conceitos de ES, particularmente sobre SPL. Portanto, a Seção 3.1 deste capítulo realiza uma introdução teórica a SPL.

Por outro lado, as outras seções provêm uma visão geral da arquitetura da solução: a Seção 3.2 apresenta uma visão geral do framework UnBBayes, ferramenta utilizada como base da presente pesquisa para a criação de uma infra-estrutura de plug-ins. A Seção 3.3 apresenta as ferramentas adicionais utilizadas. A Seção 3.4 detalha a arquitetura proposta em um nível mais concreto.

Como o foco do capítulo está na apresentação da infra-estrutura do UnBBayes (ou seja, o núcleo da infra-estrutura de plug-ins), os plug-ins gerados durante a refatoração das funcionalidades preexistentes não estão abordados aqui. Informações sobre cada plug-in podem ser encontradas no Capítulo 4. Detalhes de implementação da especificação PR-OWL 2 no framework UnBBayes estão separados no Capítulo 5, por se tratar de uma nova funcionalidade.

3.1 Base Teórica em Engenharia de Software - ES

Esta seção provê uma visão geral sobre técnicas de SPL, o principal conceito de ES utilizado na presente pesquisa. A Seção 3.1.1 apresenta os conceitos básicos sobre a SPL e a Seção 3.1.2 apresenta alguns conceitos referentes à aplicação de tais técnicas em tempo de execução do programa.

3.1.1 Visão Geral sobre SPL e FM

Pesquisas em SPL têm como objetivo a minimização do custo de desenvolvimento e manutenção de produtos de software que fazem parte de uma mesma família de produtos (94). Uma **família de produtos de software** é basicamente um conjunto de softwares que compartilham similaridades (e.g. arquitetura, componentes, requisitos) (108).

No geral, uma SPL é especificada em termos de características (*i.e. features*) em comum entre produtos ou específicas de um produto. Uma *Feature* é uma unidade lógica de comportamento de um sistema, que pode ser especificada por um conjunto de requisitos. Em outras palavras, uma *feature* é um grupo lógico de requisitos que pode ou não ser mapeado a um conjunto de artefatos de software (*e.g.* uma *feature* pode ser implementada como um ou muitos plug-ins). A especificação formal do mapeamento entre *features* e artefatos de software é a ***configuration knowledge*** (CK). Uma **configuração** é o conjunto de *features* que compõe um (1) software em particular em uma SPL. Graças a CK, uma determinada configuração de *features* pode ser mapeada a um artefato final (*i.e. software* ou sua documentação).

A identificação das similaridades e diferenças (variações) possibilita que o desenvolvimento de um novo membro da família possa ser realizado focando-se nas variações, pois funcionalidades em comum poderão ser reusadas sistematicamente. Posições do software em que variações se encaixam são conhecidas como *pontos de variação* ou *pontos de extensão*. Em outras palavras, um ponto de variação (ou ponto de extensão) é um termo que designa uma porção do programa cuja funcionalidade pode ser alterada (estendida). No âmbito de *features*, é uma porção do sistema que pode ser implementada por alguma *feature* em particular. A implementação de um ponto de extensão, na terminologia mais geral, é chamada normalmente de *extensão*, mas usaremos também o termo *feature* como seu sinônimo. Uma *feature* pode ser classificada nos seguintes tipos (104):

- **Obrigatórias**: são *features* que caracterizam a família de *softwares* e geralmente abstraem requisitos ou funcionalidades mais comuns ou essenciais.
- **Variantes**: abstrai um conjunto de *features* que podem resolver um mesmo ponto de extensão ou um conjunto de pontos de extensão similares. Podem ser do tipo **OR** (permite resolução por zero ou mais *features* no mesmo momento) ou **XOR** (somente uma *feature* deve resolver o ponto de extensão em um determinado momento).
- **Opcionais**: são *features* que, quando habilitadas, adicionam um valor às funcionalidades principais do sistema ou família de sistemas. Não representam funcionalidades essenciais.
- **Externos**: é uma categoria especial de *features* oferecidas pelo ambiente ou plataforma em que o software executará. Representam tipicamente as funcionalidades oferecidas pelo Sistema Operacional ou por máquinas virtuais e não são de responsabilidade do projetista ou desenvolvedor da SPL.

Independente de suas classificações, duas *features* podem ser **dependentes** uma das outras. Uma *feature* `X` é dependente de `Y` quando a presença de `X` exige presença de `Y` para que a configuração seja consistente.

Uma *feature* em particular pode ser implementada por um ou mais artefatos de diversas categorias (e.g. classes, linhas de código, arquivos de recurso, métodos, funções, plug-ins). Os artefatos que estão ligados a alguma *feature* são chamados, na terminologia SPL, de ***assets***. Então, como já foi mencionado nos parágrafos anteriores, o conjunto de informações necessárias

para se mapear uma *feature* a um artefato de computação (*asset*) é a CK. Muitos sistemas que tratam a CK normalmente o representa como um mapa ou tabela que relaciona cada *feature* a algum arquivo (*e.g.* documento de texto, classe Java, planilha, programa executável, código objeto, arquivo XML, etc.).

Um Modelo de *Features* (*Feature Model* - FM) é uma descrição formal dos relacionamentos e dependências entre as *features* (111), e é útil na identificação de configurações (instâncias ou produtos em particular) consistentes de acordo com os tipos das *features*. Por exemplo, um FM pode especificar que duas *features* são obrigatórias (*i.e.* ambas precisam estar presentes em um produto), opcionais (*i.e.* podem ou não estar presentes em um produto), variantes XOR (*i.e.* somente uma deve estar presente no produto), dependentes (*i.e.* se uma estiver presente, a outra também). O procedimento para a criação de um FM pode seguir basicamente duas abordagens básicas (5):

- **Abordagem Construtiva**: é a criação de um FM "a partir do zero"; ou seja, consiste na modelagem de FM durante os processos iniciais de desenvolvimento de software (ou família de softwares), normalmente sendo executado paralelamente à análise de requisitos. É uma abordagem comum na inexistência de uma quantidade considerável de softwares preexistentes a avaliar. Esta abordagem acaba introduzindo um custo inicial maior, pela necessidade de se prever variabilidades e similaridades futuras; todavia, tal custo é muitas vezes compensado com o aumento na quantidade de membros na família de softwares.
- **Abordagem Extrativa**: consiste na criação de um FM a partir da análise de um conjunto de softwares já existentes. Nesta abordagem, um conjunto de softwares é analisado e similaridades/variabilidades são identificadas. Esse processo de análise pode ser feita por um especialista ou por algoritmos computacionais especializados para tal fim. Na prática, esta abordagem acaba sendo a que mais frequente no mercado. Após criado o FM, o conjunto de softwares é refatorado para que se tornem uma família de softwares (*i.e.* para que reflitam realmente o FM).

O uso sistemático das duas abordagens em conjunto é chamado de **abordagem mista**. O UnBBayes utiliza a abordagem extrativa na refatoração inicial e abordagem construtiva na criação de novos plug-ins, portanto, adere-se a abordagem mista de forma geral.

3.1.2 Resolvendo Variabilidade em Tempo de Execução

Uma SPL também especifica um conjunto de artefatos computacionais que, ao serem "combinadas" de acordo com assertivas de consistência especificadas no modelo de *features* e mapeamentos na CK, formam um software em particular que pode ser executada em alguma plataforma. Sem essa "ligação" consistente entre os artefatos, uma SPL não pode gerar um software propriamente dito. Dependendo das regras no modelo de *features*, diferentes *features* (e consequentemente diferentes artefatos computacionais) podem se associar a um mesmo ponto de extensão. A ligação entre uma *feature* em ponto de extensão em particular chama-se **resolução** do ponto de extensão por uma *feature*, e é um processo indispensável na geração de um software completo.

Segundo (104), a resolução pode ocorrer em diferentes momentos do ciclo de vida da SPL:

- **Derivação de arquitetura de software**: a própria arquitetura de uma SPL pode conter diversos pontos de extensão não resolvidas. Em alguns casos, a resolução dessas extensões podem gerar arquiteturas completamente diferentes. Nesse caso, a resolução está

ocorrendo durante o projeto de arquitetura. Um exemplo de resolução durante derivação de arquitetura ocorre quando usamos scripts para gerar diferentes diagramas de classes UML (*i.e.* a arquitetura de um outro software é um artefato final).

- **Compilação**: quando diferentes comportamentos são criados por mecanismos como diretivas de compilação, extensão de classes ou uso de aspectos (em programação orientada a aspectos) "não dinâmicos", a resolução está ocorrendo em tempo de compilação.
- **Ligação**: esta categoria difere significativamente de acordo com a linguagem de programação. Por exemplo, em programas em C, a ligação de códigos objeto tipicamente ocorre pouco depois da compilação e antes do carregamento do programa em mémória. No Java, a ligação entre classes é feita de forma "preguiçosa" em tempo de execução. No entanto, costumamos englobar a etapa de criação de links entre os artefatos de software como "tempo de ligação".
- **Execução**: é quando a resolução ocorre durante a execução do programa, ou no carregamento do programa em memória para a execução. Plug-ins se enquadram nesta categoria.

Linhas de Produto de Software Dinâmico, ou *Dynamic Software Product Lines* (DSPL) são SPL em que a resolução (*i.e.* ligação de *features* a pontos de extensão, resultando em uma configuração em particular) ocorre basicamente em tempo de execução do programa (54). Tipicamente, quanto mais postergada o momento da resolução, mais flexível torna-se a SPL. Apesar da DSPL maximizar a flexibilidade, possui alguns requisitos especiais que não se observam em SPL convencionais. Por exemplo:

- como a resolução ocorre sob responsabilidade do usuário (*e.g.* o usuário deve fazer o *download* dos artefatos e mover para os locais devidos) ou pelo próprio sistema (*e.g.* *download* automático de artefatos), a segurança e consistência torna-se um ponto crucial, pois não há mais uma etapa de testes por desenvolvedores internos;
- o tempo gasto para a transição entre configurações distintas torna-se importante, pois se a transição for demorada, causa um impacto imediato na aceitação do software.

3.2 O UnBBayes

Como já comentado, o framework UnBBayes é implementado pelo GIA/UnB (63), inicialmente desenvolvido em Delphi e posteriormente em Java, para suporte ao raciocínio probabilístico. Possui um editor visual (GUI) de fácil uso, permitindo que o usuário construa, de forma intuitiva, BN, ID e MSBN, além da entrada e propagação de evidências, realização de inferência probabilística e aprendizagem da topologia e/ou parâmetros de uma BN. O framework está em constante desenvolvimento, desde o ano 2000.

O principal ambiente de desenvolvimento do UnBBayes foi, e continua sendo, o Eclipse, dado seus diversos *plug-ins* para facilitar a criação e testes das funcionalidades. Ultimamente, o UnBBayes foi estendido para permitir a representação de ontologias probabilísticas e a realização de inferências em PR-OWL/MEBN, resultando o UnBBayes-MEBN. O trabalho atual realiza diversas otimizações e correções sobre esse novo módulo, adicionando funcionalidades para compatibilidade com a especificação PR-OWL 2.

Três objetivos formam a base para as últimas versões desenvolvidas do UnBBayes:

1. ser uma plataforma operacional para a disseminação dos conceitos e utilidades do raciocínio probabilístico,
2. ser uma ferramenta visual configurável e de fácil uso,
3. alcançar um alto grau de extensibilidade e variabilidade.

O primeiro objetivo pôde ser alcançado implementando-se um formalismo gráfico probabilístico em estado da arte - BN - e uma máquina de inferência padrão[29] baseada no algoritmo de árvores de junção (57–59). O segundo objetivo é alcançado ao se implementar uma GUI para permitir um meio intuitivo para a edição e inferência em modelos gráficos, oferecendo assim um ambiente integrado para engenharia do conhecimento. A nacionalização é facilitada com o uso de arquivos de recurso (*resources*), que são selecionados automaticamente pelo sistema de acordo com opções regionais do sistema operacional do usuário. Finalmente, o terceiro objetivo é alcançado com o uso sistemático das vantagens da linguagem Java™ e de seu paradigma orientado a objeto (permitindo então o encapsulamento e definindo pontos de extensão). A implementação de plug-ins então se enquadra como uma proposta para se alcançar o terceiro objetivo.

3.2.1 Histórico

Apesar da versão Java™ do UnBBayes ter origem em 2001, ele foi inicialmente escrito em linguagem Delphi™ em 2000, quando uma série de estudos foi conduzido pela GIA/UnB visando a criação de um sistema de suporte a diagnóstico médico. A lista cronológica a seguir resume os marcos deste projeto.

2000 O projeto *SARA* (Saúde Apoiada em Raciocínio Automatizado) foi iniciado. Seu objetivo principal foi a criação de um sistema para suporte a diagnóstico médico no hospital da UnB.

2000 A versão inicial do UnBBayes (100) foi implementado em Delphi™ 5.0 (*Imprise Corp*) para prover uma ferramenta computacional ao projeto *SARA*.

2001 A tradução do UnBBayes para a linguagem Java™ foi iniciada (17).

2003 Algoritmos para aprendizagem bayesiana foram incluídas ao UnBBayes (36).

2003 Suporte a ID e MSBN (112) foram incluídas (16).

2004 Suporte a algoritmo de aprendizagem bayesiana com dados incompletos (algoritmo EM) foi adicionado (74).

2005 UnBMiner (64) - uma variação do UnBBayes voltados para data mining - foi publicado.

2006 Metáfora Médica - uma interface projetada para facilitar diagnóstico médico - foi terminada.

2007 Suporte para MEBN (68) foi iniciada (20).

2008 Metáfora de Identificação Humana - uma interface projetada para o domínio de "Identificação Humana" (um domínio em que o programa estima o sexo de uma pessoa através das características do crânio) - foi incluída.

2008 Suporte a OOBN (61) foi incluída.

[29] Há outros algoritmos de inferência bayesiana implementadas no UnBBayes também.

2009 A versão 3.52.7 do UnBBayes recebe o prêmio *SOFTPEDIA "100% FREE" AWARD*[30], obtendo-se então uma atenção maior, como um software livre e imune (sem malwares).

2010 Suporte a Plug-ins (um dos temas desta dissertação) foi iniciada. Todas as variações anteriores do UnBBayes foram refatoradas como novos plug-ins.

2010 Iniciada a implementação de PRM (51) (vide Seção 4.8).

Como a cronologia acima sugere, a equipe do UnBBayes tem trabalhado ativamente na incorporação de novos formalismos baseados em BN ao UnBBayes. Atualmente, o software é distribuído gratuitamente para uso não comercial, sob a licença GPL versão 3. API e documentação em JavaDoc são disponibilizadas para desenvolvedores, possibilitando que os interessados em raciocínio probabilístico possam utilizar funcionalidades do UnBBayes em suas aplicações.

3.2.2 Arquitetura do UnBBayes - Antes da Refatoração

O UnBBayes, antes mesmo da refatoração para uma arquitetura de plug-ins (vide Seção 3.4 para detalhes sobre a refatoração), já era estruturado basicamente em um design pattern *Model-View-Controller* (MVC), o qual separa explicitamente os elementos dos programa em três papéis distintos, visando prover o que no inglês é conhecida como *SoC - Separation of Concerns*[31].

A Figura 3.1 mostra os pacotes básicos do UnBBayes antes da refatoração. São eles:

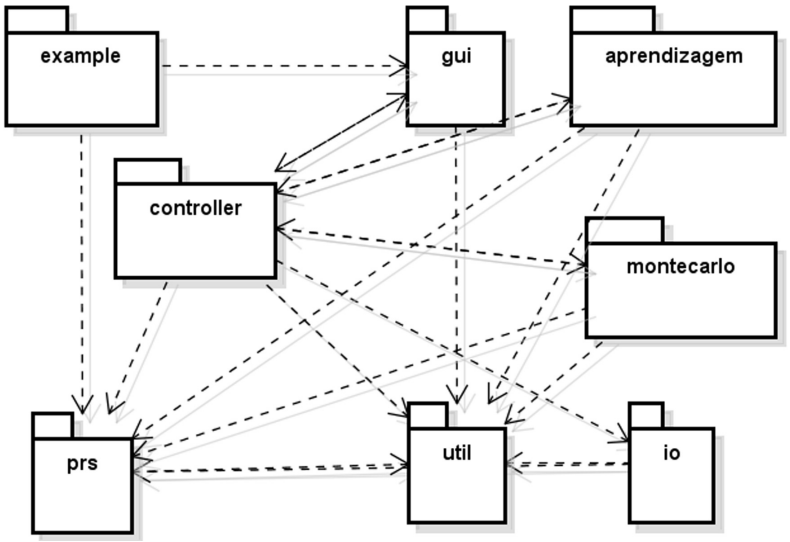

Figura 3.1: Pacotes do UnBBayes antes da refatoração.

pacote `aprendizagem`, reúne as classes relacionadas ao processo de aprendizagem em lote;

[30] http://www.softpedia.com/progClean/UnBBayes-Clean-144272.html

[31] No âmbito de softwares, *Separation of Concerns* (separação de preocupações ou "responsabilidade") ocorre quando o software é separado em diferentes elementos com mínima sobreposição de funcionalidades.

pacote `controller`, classes responsáveis por controlar o fluxo de operações realizadas pelo usuário, fazendo a ligação entre a GUI e as funcionalidades (papel "*Controller*" do MVC);

pacote `gui`, contém as classes que implementam a interface com o usuário (papel "*View*" do MVC), permitindo a visualização e manuseio de BN;

pacote `io`, classes responsáveis por salvar BN em arquivos, bem como por construí-las novamente a partir destes arquivos. Há atualmente dois formatos válidos: o formato "net" e o formato "xmlbif";

pacote `montecarlo`, conjunto de classes para gerar amostras aleatórias de BN dadas como parâmetro;

pacote `prs`, classes que definem a estrutura de BN (papel "*Model*" do MVC) e implementam os algoritmos de inferência sobre estas;

pacote `util`, classes utilitárias, utilizadas por pelo menos dois pacotes do projeto;

pacote `example`, classes com exemplos de uso do UnBBayes como uma API.

O projeto UnBBayes oferecia também algumas *branches* adicionais voltadas para objetivos específicos, permitindo o tratamento de necessidades não resolvidas pelo framework principal. Por exemplo:

branch `UnBMiner`, projeto voltado para data-mining. Fornece uma forte ferramenta para análise de dados no modelo CRISP-DM (1).

branch `Metaphor`, interface adicional para usuários comuns no tratamento de BN. Oferece somente funcionalidades de leitura, não oferecendo funcionalidades de edição de BN. Veja Seção 4.4 para a nova versão, voltada para o projeto de identificação humana.

Segue nas próximas seções algumas funcionalidades que o UnBBayes oferece.

3.2.3 ID e BN

A modelagem apresentada na Figura 3.2 apresenta as principais classes que implementam as BNs e IDs no UnBBayes.

Interface `INode`, um nó genérico que possui estados (`String`), uma posição (x,y) e um tamanho (largura, altura). Estes dois últimos campos são representados pela classe `SerializablePoint2D`. A classe `Node` é a implementação padrão de `INode`.

Classe abstrata `TreeVariable`, para variáveis que serão visualizadas na árvore de nós e estados com suas respectivas probabilidades no painel de compilação e inferência de redes probabilísticas.

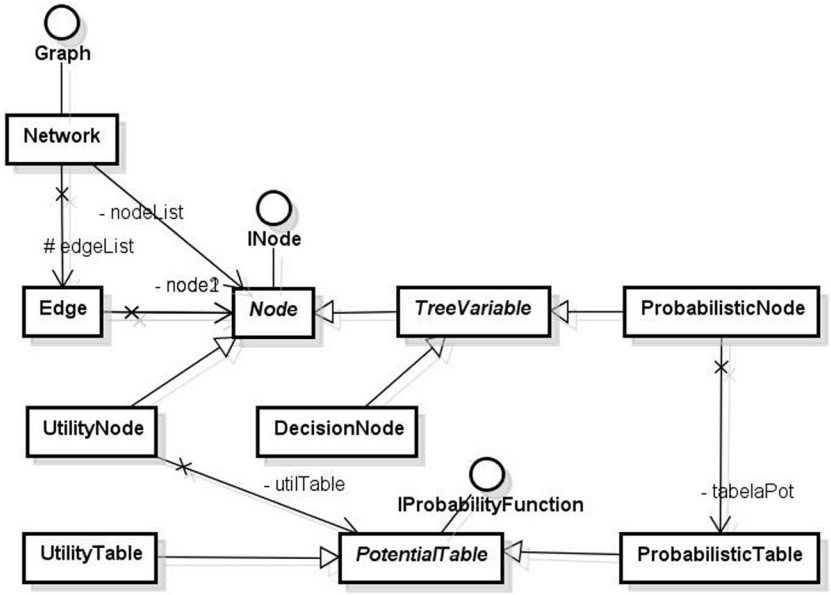

Figura 3.2: Modelagem de classes para BN e ID.

As classes que representam variáveis estão diretamente dependentes de classes de representação visual. Isso é um ponto fraco na modelagem atual, que compromete a separação de lógica com modelos visuais. Os tipos de variáveis atualmente suportados pelo UnBBayes são as RV, as variáveis de decisão e as funções utilidade, sendo estas duas últimas utilizadas para a modelagem de grafos de decisão.

Classe `DecisionNode`: variável de decisão.

Classe `ProbabilisticNode`: variável probabilística.

Classe `UtilityNode`: função utilidade.

Classe `Edge`: representa um arco ligando dois nós.

Classe `PotentialTable`: tabela que possui os valores das células como um número real entre 0 e 1 (coleção de números reais representados pela classe `FloatCollection`).

Classe `ProbabilisticTable`: CPT.

Classe `UtilityTable`: tabela representando uma função de utilidade.

Classe `Network`: representa um grafo genérico.

A Figura 3.3 ilustra um caso de uso comum do módulo de BN do UnBBayes e a Figura 3.4 mostra as classes principais de GUI e controle do UnBBayes (e seu relacionamento com algumas classes de modelo, como `ProbabilisticNetwork` - representação de uma rede probabilística em geral). Como pode ser observado, a GUI era completamente acoplada inclusive com classes de MEBN, antes da refatoração. A listagem abaixo oferece uma breve descrição

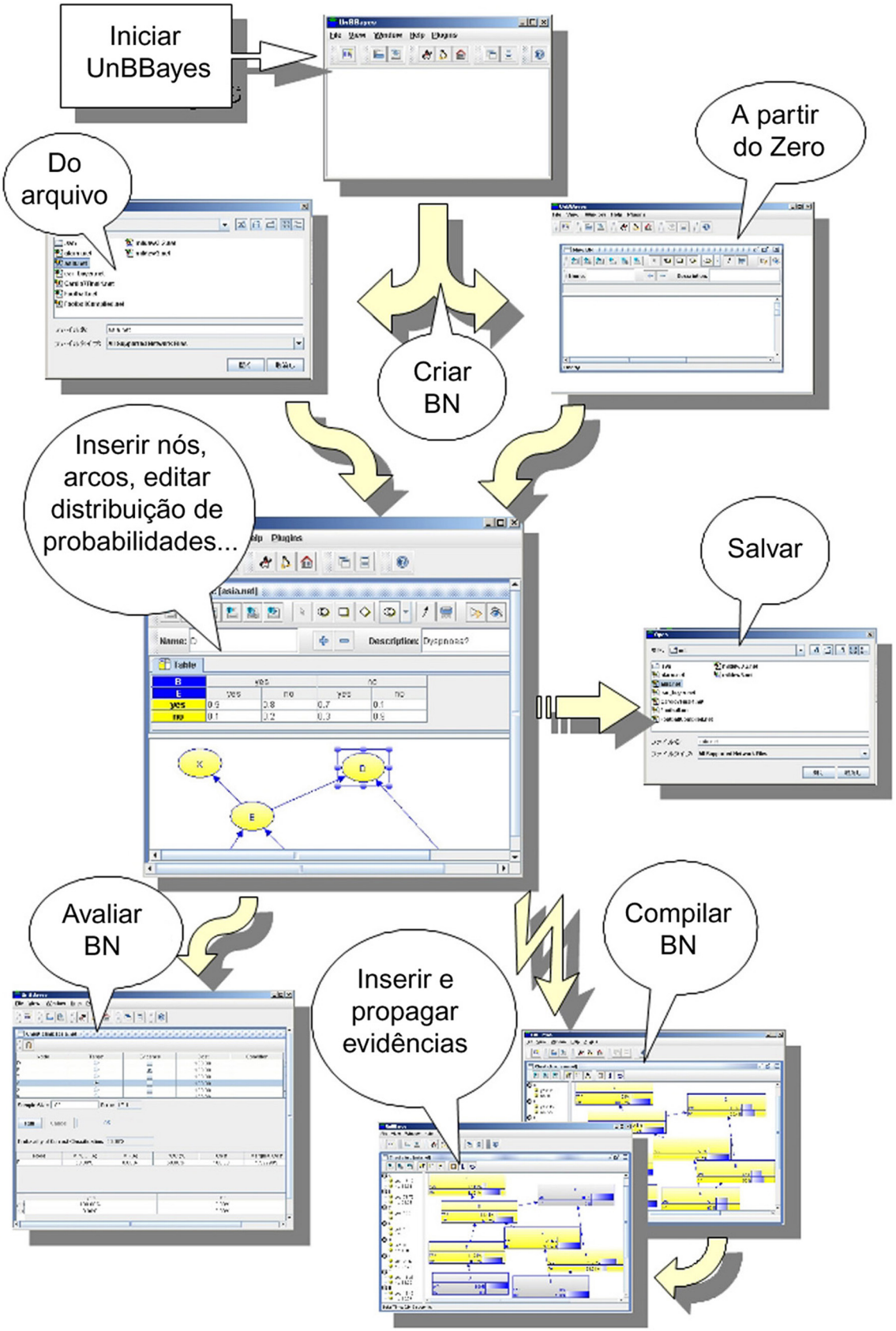

Figura 3.3: Fluxo de exemplo de uso do UnBBayes, módulo de BN e sua GUI. Cada "tela" representa meramente um estado possível do UnBBayes.

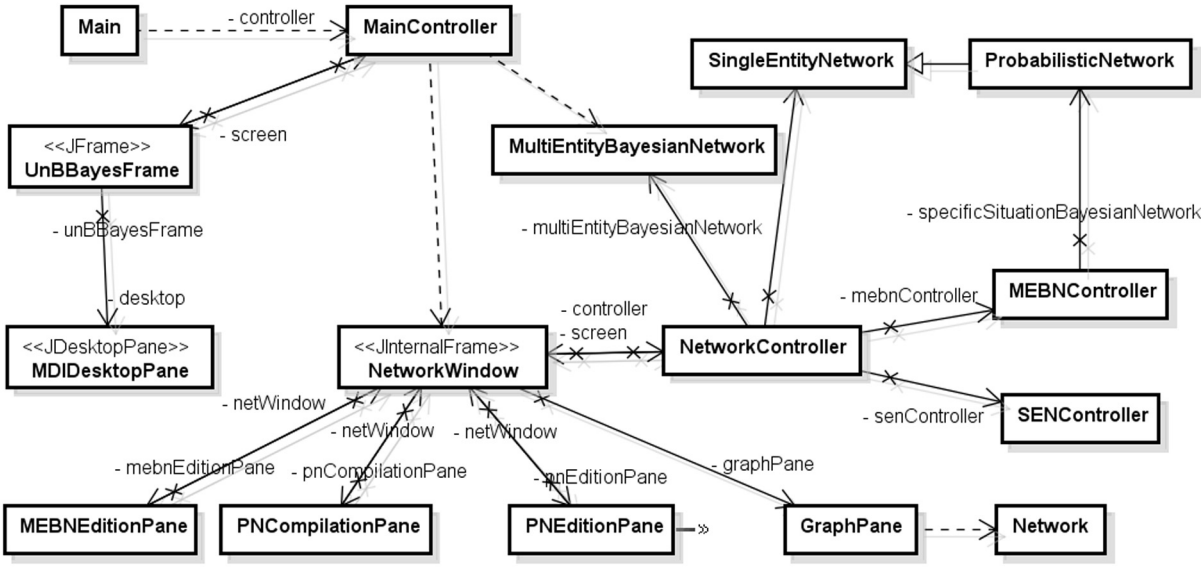

Figura 3.4: Classes de GUI do UnBBayes antes da refatoração.

das classes pertencentes a funcionalidades de BN, ID e dependências MEBN causadas por acoplamento indesejado na arquitetura antiga. Classes de MEBN estão mais detalhadas na Seção 5.1.4.

- Classe `Main`: responsável pela inicialização do UnBBayes. Cria o controlador principal (`MainController`).
- Classe `MainController`: responsável por criar, abrir e salvar redes suportadas pelo UnBBayes: Redes Probabilísticas, MEBN, e MSBN. Cria o controlador e o painel da rede selecionada.
- Classe `UnBBayesFrame`: tela principal do programa (exibida independentemente do tipo de rede).
- Classe `MDIDesktopPane`: *desktop* do `UnBBayesFrame`. Permite a abertura e gerenciamento de múltiplos painéis (podendo estes serem de tipos diferentes, de acordo com o tipo de rede tratado por cada um).
- Classe `NetworkWindow`: janela da rede, possuindo o painel de edição correspondente à rede e o painel que contém o grafo da rede. Cria a instância de `NetworkController`, controladora da rede.
- Classe `GraphPane`: responsável por desenhar o grafo da rede na tela. Este grafo é composto por nós e arcos, sendo que estes podem ser tanto a representação de uma BN normal, quanto a representação dos nós componentes de um MFrag. Trata os eventos do mouse, permitindo que o usuário o use para editar os objetos.
- Classe `NetworkController`: delega funções que serão executadas em uma BN normal ou MEBN, como por exemplo, inserir nós e propagar evidências. As funções são delegadas ao controlador associado à rede editada.

- Classe `SENController`: controlador das operações realizadas sobre uma rede probabilística normal.
- Classe `PNEdtionPane`: painel de edição de BN (***P**robabilistic **N**etworks*).
- Classe `MebnController`: controlador das operações realizadas sobre uma MEBN.
- Classe `MEBNEditionPane`: painel de edição de MEBN.

3.2.4 MSBN

A Figura 3.5 apresenta a modelagem das classes que implementam MSBN no UnBBayes. A MSBN divide uma BN em sub-redes que representam subdomínios naturais, formando uma estrutura conhecida como super árvore (112). Interfaces entre duas sub-redes são nós compartilhados que satisfazem algumas restrições para se garantir uma distribuição de probabilidade conjunta única e consistente. O resultado final da inferência deve ser o mesmo que seria obtido com a BN formada pela união das sub-redes.

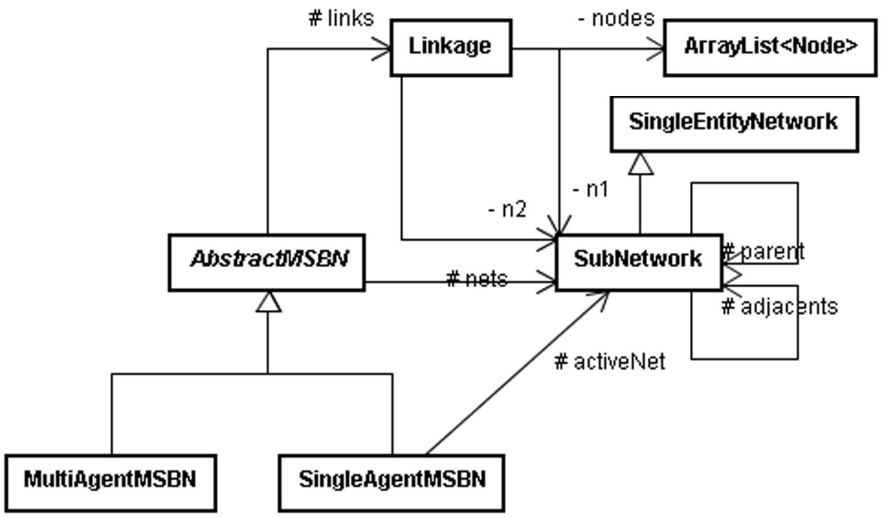

Figura 3.5: Modelagem de classes para MSBN.

Segue abaixo descrições das classes ilustradas no diagrama UML da Figura 3.5.

Classe `SubNetwork`, sub-rede de uma rede múltipla seccionada.

Classe `Linkage`, possui os nós que ligam duas sub-redes.

Classe `AbstractMSBN`, define os métodos para tratar MSBN.

Classe `MultiAgentMSBN`, uma rede múltipla seccionada multi-agente.

Classe `SingleAgentMSBN`, uma rede múltipla seccionada.

3.3 Ferramentas Utilizadas

Além das ferramentas já mencionadas, como a máquina virtual Java (*e.g.* Sun/Oracle JRE/JDK 5), foram utilizadas as seguintes ferramentas neste projeto.

- Ferramentas para edição, gerenciamento de artefatos (*e.g.* código fonte e diagramas) e comunicação:
  - Eclipse Helios (<http://www.eclipse.org/helios/>) - utilizado como um ambiente de desenvolvimento integrado para linguagem Java;
  - Eclipse TPTP (<http://www.eclipse.org/tptp/>) - plug-in do Eclipse que provê um conjunto de ferramentas para teste (JUnit), desempenho e outras funcionalidades afins (vide Seção 5.4 para seu uso);
  - Apache Maven (<http://maven.apache.org/>) - ferramenta para gerenciamento integrado de projetos de software.
  - ChangeVision astah community (<http://astah.change-vision.com/en/>) - ferramenta para edição de diagramas UML;
  - Subversion (<http://subversion.apache.org/>) - versionador de artefatos (o servidor Subversion do projeto UnBBayes está hospedado no repositório da SourceForge <http://sourceforge.net/projects/unbbayes/>);
  - Skype (<http://www.skype.com/>) - ferramenta para comunicação em voz e vídeo via Internet, usado meramente para manter contato com a equipe da George Mason University.
- Bibliotecas/frameworks a serem incorporadas pelo UnBBayes:
  - Java Plug-in Framework - framework para plug-ins (90).
  - OWL API (<http://owlapi.sourceforge.net/>) - API com suporte a OWL versão 2;
  - HermiT (<http://hermit-reasoner.com/>) - máquina de inferência (DL) de OWL versão 2;
  - Stanford Protégé 4.1 (<http://protege.stanford.edu/>) - software Java com GUI para edição de ontologias, com uso de OWL API e HermiT em seus componentes; utiliza OSGi como infra-estrutura de plug-ins;
  - Apache Felix (<http://felix.apache.org/>) - uma implementação da especificação OSGi^{TM}para carregamento dinâmico de classes Java como módulos independentes. O Felix será utilizado pelo UnBBayes para o carregamento de componentes do Protégé, uma vez que este se adere à especificação OSGi.

3.3.1 JPF

Como mencionado previamente, o UnBBayes utiliza o JPF versão 1.5.1 para prover um ambiente flexível baseado em plug-ins. O JPF é um framework para plug-ins licenciado em LGPL voltado para a construção de projetos Java escaláveis e com baixo custo de manutenção, provendo mecanismos para descoberta e carregamento dinâmico sob demanda. Como o processo

de ativação (*i.e.* o processo de carregamento de classes) é de forma "preguiçosa" (lazy-loading), as classes dos plug-ins são somente carregadas em memória quando necessárias.

O JPF é utilizada como biblioteca no UnBBayes. Esta forma de uso, comparada ao seu uso convencional como um framework, minimiza dependências do UnBBayes a decisões de projeto do JPF, tornando a própria arquitetura de plug-ins também em um potencial ponto de variação.

Sob o ponto de vista arquitetural, a maior vantagem do JPF está na **isolação do carregador de classes**. Cada plug-in do JPF possui seu própio carregador de classes (`ClassLoader`) que é responsável por gerenciar seus recursos (classes e arquivos correlatos). Dependências entre carregadores são organizadas de uma forma hierárquica, portanto requisições por recursos podem ser delegados para os carregadores ancestrais[32].

Apesar da JPF ser conceituralmente um *framework*, UnBBayes o usa basicamente como uma biblioteca[33] para se evitar dependência arquitetural excessiva ao JPF. Isso se justifica pelos seguintes:

1. UnBBayes deve permanecer uma aplicação/framework autônoma e portanto não deve ser dependente de decisões arquiteturais de terceiros, e
2. a infraestrutura de plug-ins por si é suscetível a mudanças (é um possível ponto de variação/extensão) e provavelmente será alterada em lançamentos futuros.

Figura 3.6 ilustra a diferença entre o uso comum do JPF como um *framework* e o uso no UnBBayes. O passo "*initialize main application*" (inicializa aplicação principal) serve virtualmente como uma interface entre a biblioteca de inicialização da JPF e a aplicação, e por esse motivo a linha tracejada passa por cima desse passo. De maneira similar, o passo "*load JPF (PluginManager)*" é considerado como uma interface também.

O JPF gerencia um registro de plug-ins disponíveis e seus pontos de extensão em uma classe chamada `org.java.plugin.PluginManager`. Existe uma classe adaptadora no UnBBayes (`unbbayes.util.extension.manager.UnBBayesPluginContextHolder`) para prover um acesso unificado ao `PluginManager`; portanto, desenvolvedores que necessitam de acesso a funcionalidades do JPF (*i.e.* pontos de extensão e suas extensões) via infraestrutura de plug-ins do UnBBayes devem primeiro procurar por `UnBBayesPluginContextHolder`.

3.3.2 OWL API e HermiT

OWL API é uma API em Java concebida como uma implementação referencial nas tarefas de criação, manipulação e serialização de ontologias em OWL 2. É distribuída sob licença LGPL e possui, dentre outros, os seguintes componentes:

- API para OWL 2;
- entrada e saída em RDF/XML;
- entrada e saída em OWL/XML;
- entrada e saída em sintaxe funcional de OWL;

[32] A ordem de busca por recursos pode ser invertida via configuração também.

[33] A principal diferença entre um *framework* e uma biblioteca está na direção das chamadas a rotinas. Se `X` é um *framework* e `Y` **adere** a ele, então `X` invoca rotinas de `Y` (`X` controla o fluxo geral do programa como todo). Se `X` é uma biblioteca e `Y` o **usa**, então `Y` invoca rotinas de `X` (`Y` controla o fluxo geral do programa).

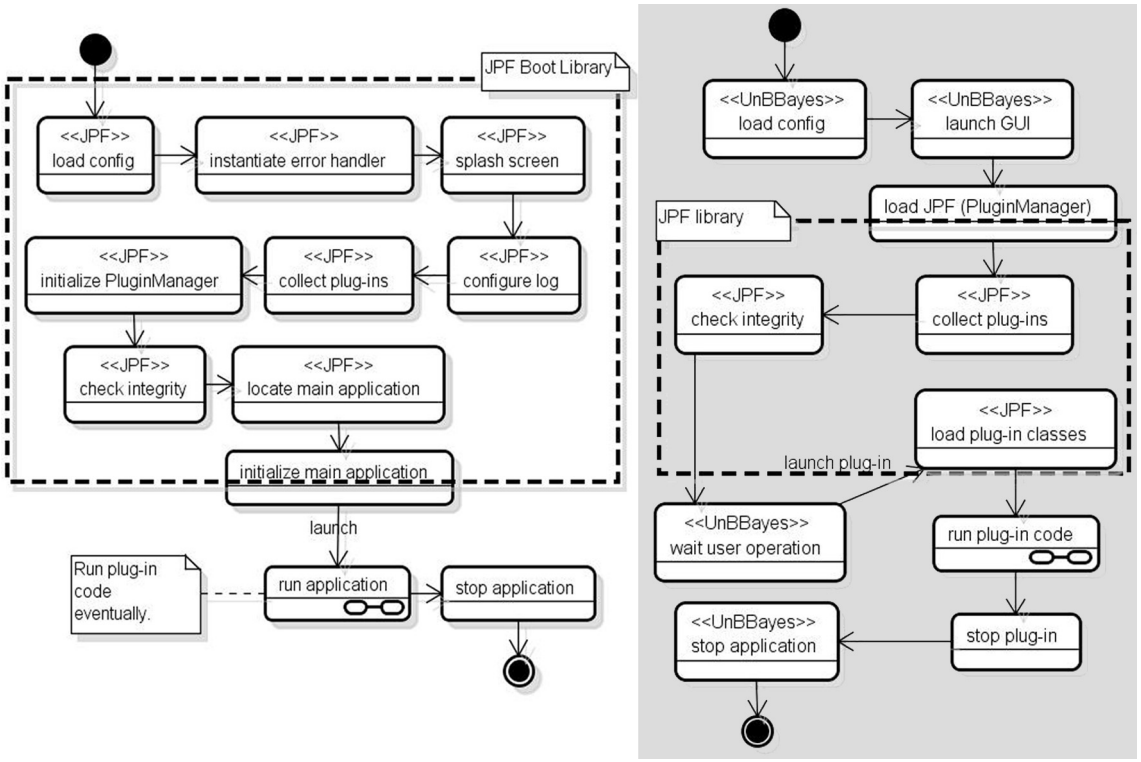

Figura 3.6: Comparação entre o uso de JPF como framework (esquerda) e como biblioteca (direita).

- entrada e saída em sintaxe Turtle;
- interfaces para máquinas de inferências como FaCT++, Pellet, Racer e HermiT.

O novo plug-in do UnBBayes para suporte a PR-OWL 2 utiliza os objetos da OWL API como tipos de dados primários na manipulação de ontologias OWL 2, em que PR-OWL 2 é escrito. Como o Protégé 4.1 (Seção 3.3.3) também utiliza classes da OWL API como base, esta API se tornou o pivô central para manipulação de ontologias OWL 2 e consequentemente de ontologias PR-OWL 2.

O HermiT é uma máquina de inferência padrão que vem junto ao Protégé 4.1 (Seção 3.3.3) para raciocínio em ontologias descritas em OWL 2. Como a OWL API oferece interface para essa máquina de inferência, foi uma opção natural. O plug-in de PR-OWL 2 do UnBBayes reutiliza a mesma instância do HermiT utilizado pelo Protégé 4.1 (que é nada mais do que a interface oferecida pela OWL API para o HermiT), portanto, máquinas de inferências disponibilizadas como plug-ins do Protégé 4.1 são supostamente reutilizáveis no UnBBayes. No Capítulo 5 são apresentados também alguns mecanismos adotados no UnBBayes para reuso de instâncias de máquinas de inferência da OWL API e Protégé 4.1.

Como o acesso ao HermiT é sempre via interface oferecida pela OWL API, detalhes de implementação do HermiT não são considerados pelo UnBBayes; logo, não serão considerados nesta dissertação também.

As principais interfaces utilizadas são:

OWLReasoner máquina de inferência (reasoner) para ontologias em OWL 2;

OWLOntology representa a ontologia como todo;

OWLOntologyManager principal ponto de acesso para facilidades da OWL API, como carregamento e armazenamento;

OWLDataFactory facilidades para criação de elementos da OWL 2, como classes ou indivíduos;

IRI *Internationalized Resource Identifier*;

PrefixManager gerencia os prefixos de IRI;

OWLClass representa uma classe;

OWLProperty representa uma propriedade;

OWLIndividual representa um indivíduo;

OWLLiteral representa um valor literal.

3.3.3 Protégé 4.1

O Protégé (47) é uma plataforma flexível e configurável para o desenvolvimento de aplicações dirigidas a modelos arbitrários e componentes, permitindo uma edição visual de ontologias. Ferramentas como o Protégé facilitam bastante a criação de ontologias (seja elas probabilísticas ou não). As ontologias podem ser armazenadas então em uma grande variedade de formatos, incluindo RDF, OWL e XML Schema.

O Protégé versão 4.1 introduziu algumas inovações em nível arquitetural, incluindo o uso de classes da OWL API para tratamento dos elementos contidos em uma ontologia. Provê muitas facilidades de edição e navegação para modelos OWL e diferentes classes de editores para uma ontologia em OWL, sendo os mais importantes:

Editor de classes permite a navegação, criação, remoção e edição sobre classes OWL 2, incluindo edição avançada de restrições e atribuição de propriedades padrões.

Editor de indivíduos permite navegação sobre diferentes instâncias das classes presentes na ontologia e edição de seus atributos.

Editor de propriedades exibe uma lista de todas as propriedades entre as classes presentes na ontologia e permite sua edição; como alteração de domínio e imagem, edição de comentários, atribuição de propriedade inversa, transitiva, simétrica ou cadeira de propriedades.

Editor de entidades é um painel que integra os três editores anteriores.

Editor de meta-dados permite gerenciar ontologias importadas. Ao editar uma ontologia PR-OWL 2 diretamente no Protégé, um passo inicial seria importar a biblioteca básica (prowl2.owl) através deste editor.

Queries em DL permite que consultas sejam feitas na ontologia usando-se sintaxe Manchester-OWL.

Plug-ins outras telas podem ser importadas via plug-ins.

O Protégé 4.1 oferece uma infra-estrutura de plug-ins que se adere ao framework OSGi, oferecendo um poderoso ambiente de softwares para desenvolvimento de novas funcionalidades. No entanto, diferente das suas versões anteriores, a versão 4.1 não oferece uma API propriamente dita. Por esse motivo, foi avaliado também a continuidade de uso do Protégé 3 (versão utilizada pelo UnBBayes-MEBN antes da refatoração) para a criação do plug-in de PR-OWL 2. No entanto, como a versão 3 do Protégé não oferecia suporte a OWL 2 (a linguagem em que PR-OWL 2 esteve baseada), a escolha pelo Protégé 4.1 foi mandatória. A forte dependência do Protégé 4.1 à infra-estrutura do OSGi e o uso de variáveis e métodos estáticos ou privados dificultou bastante o seu uso como API. No entanto, algumas técnicas voltadas a delegação de *classloaders* poderam ser utilizadas para sanar o problema no âmbito do UnBBayes. A Seção 5.3.4 apresenta quais foram os mecanismos utilizados para simular o uso do Protégé 4.1 como API.

O UnBBayes-MEBN, o módulo de MEBN criado antes da refatoração, utiliza a API do Protégé versão 3 para acesso a ontologias. Felizmente, não houve a necessidade de preocupação com conflitos de versões, pois as classes das versões 3 e 4.1 possuíam nomenclaturas diferentes e o mecanismo nativo de modularização de plug-ins do JPF facilitava no tratamento diferenciado de dependências. Maiores informações sobre o uso da API Protégé-OWL podem ser encontradas em (60).

As principais interfaces/classes utilizadas são:

ProtegeApplication representa o Protégé 4.1 em execução;

ProtegeManager ponto de acesso a funcionalidades da aplicação Protégé 4.1 como todo;

OWLModelManager ponto central de acesso objetos de edição de ontologias OWL 2 no Protégé 4.1.

EditorKit representa objetos da GUI;

3.4 Arquitetura Baseada em Plug-ins

Apesar do UnBBayes ser escrito em Java™, uma linguagem de programação tipicamente orientada a objeto[34], um modelo "puramente" orientado a objeto (seja, uma arquitetura em que os programas são compostos por conjuntos de elementos autônomos e relativamente completos, representando objetos reais ou imaginários, com comunicação via troca de mensagens) é raramente aplicado no UnBBayes. Ao invés disso, o UnBBayes segue uma adaptação de um paradigma orientado a "componentes", cujo programa (no sentido de conjunto de objetos/classes Java) é organizado em elementos destacáveis e substituíveis - *i.e.* "componentes" (tais elementos são similares a componentes eletrônicos - *e.g. chips* de placas de circuitos integrados).

Um componente é normalmente relacionado a um pequeno comportamento de um "pedaço" do software (logo, o componente não é necessariamente uma representação intuitiva de um elemento no domínio em questão) e diferentes componentes podem ser então agrupados para gerar uma entidade maior, representando uma característica observável de um sistema (*e.g.* o agrupamento entre um componente de visualização da BN, estrutura de dados da BN, formulário

[34] O conceito de orientação a objetos (OO) foi originalmente introduzido na linguagem *Smalltalk* (53).

de entrada de dados e outros componentes adicionais podem juntos gerar em um editor BN completo).

Arquiteturas baseadas em componentes substituíveis provêm um alto nível de variabilidade e reusabilidade, que são requisitos mencionados também no início da Seção 3.2. Customizações em modelos "orientados a componentes" podem ser realizados simplesmente substituindo os componentes (que são normalmente um conjunto de objetos ou classes). Por exemplo, se precisarmos alterar o formato (visual) de um nó da BN, bastaríamos substituir o componente responsável por desenhar o nó na tela; portanto, não haveria necessidade de "abrir" a classe e alterar linhas de código.

A Figura 3.7 ilustra uma diferença simbólica entre a "pura" orientação a objeto e a abordagem com "orientação a componentes" para se representar um editor simples de BN. O princípio que demanda a escrita de classes menos resilientes a alterações (de linhas de código), mas abertas a extensões (*i.e.* substituição sistemática sem a necessidade de acesso ao código fonte original) é conhecida como princípio *open-closed* (83) e é uma base fundamental na adoção da arquitetura baseada em componentes. Se tal princípio deve ser aplicado em tempo de execução, então uma arquitetura baseada em *plug-ins* pode ser adotada.

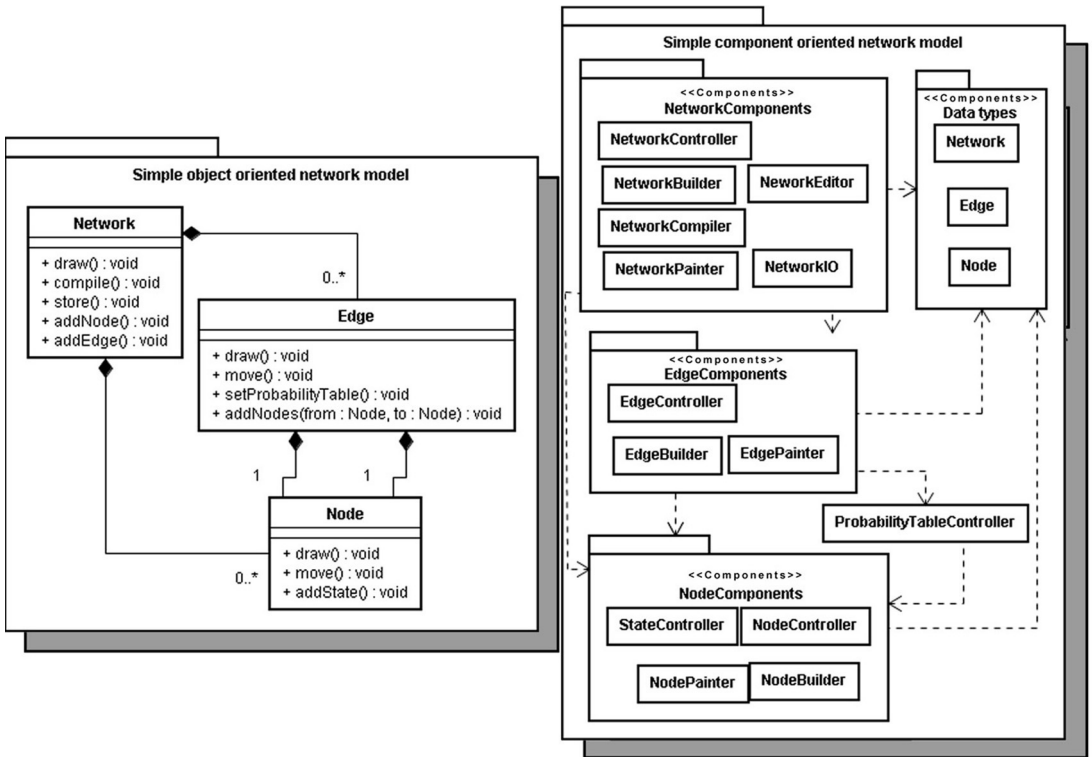

Figura 3.7: Diferenças entre "pura" orientação a objeto (esquerda) e orientação a componentes (direita). Como o foco de componentes está no reuso, os objetos/classes podem parecer contra-intuitivos.

Diferente da substituição bibliotecas ou alterações pontuais no código fonte, *plug-ins* oferecem um meio para se executar novas rotinas no próprio ambiente de execução do UnBBayes. Um plug-in é um programa que interage com a aplicação hóspede (core) para prover uma funcionalidade específica sob demanda. A ligação entre o plug-in e o core ocorre geralmente ou em tempo de carregamento (quando a aplicação inicia) ou em tempo de execução, portanto, uma

arquitetura orientada a plug-ins oferece um ambiente prático e flexível para a variabilidade de softwares, sem a necessidade de alteração da aplicação hóspede.

De forma geral, um sistema que provê pontos de extensão para plug-ins (*i.e.* porções do programa que podem ser estendidos por plug-ins) oferece as seguintes vantagens:

- permite que terceiros estendam a aplicação com mais facilidade;
- permite suporte a funcionalidades ainda não previstas;
- reduz o tamanho de um produto (pois partes indesejáveis podem ser simplesmente ignoradas pelo usuário);
- deixa o programa mais modular, pois pontos de extensão são interfaces bem definidas;
- em alguns casos, permite isolar licenças, possibilitando que plug-ins possam ser lançados em licenças diferentes do core[35].

Como comentado na Seção 1.3.1 e 1.3, o UnBBayes utiliza JPF para prover suporte a plug-ins. Como consequência, um plug-in no UnBBayes é armazenado como uma pasta, um arquivo *ZIP* ou *JAR* contendo os elementos na enumeração abaixo.

1. **Descritor de plug-ins**[36]: um arquivo XML (plugin.xml) contendo metadados sobre o próprio plug-in (exemplos de descritores podem ser encontrados nas Seções 3.4.4, 3.4.2 e 3.4.5). Como descritores também servem para declarar pontos de extensão, o core do UnBBayes oferece possui um descritor na pasta `unbbayes.core` somente para explicitar pontos de extensão, sem oferecer funcionalidades adicionais.
2. **Classes**: os programas em Java podem ser arquivos "*.class*" ou arquivos compactados no formato *JAR*.
3. **recursos**: *e.g.* imagens, ícones, arquivos de mensagens, textos com marcações.

Para que a infraestrutura de plug-ins do UnBBayes seja um ambiente de fácil uso, as seguintes funcionalidades básicas devem ser oferecidas na nova arquitetura:

1. gerenciamento de dependências - isso torna possível que um plug-in reuse classes oferecidas por outro plug-in, agilizando o desenvolvimento e produzindo artefatos menores e menos redundantes;
2. verificação de integridade - plug-ins devem ser ativados somente se todas as dependências forem resolvidas;
3. controle de versões - a aplicação deve permitir a coexistência de diferentes versões do mesmo plug-in no mesmo ambiente, para que não seja necessário a sobrescrita de arquivos para a sobrescrita de funcionalidades;
4. plug-ins de plug-ins - deve ser possível a definição de plug-ins em cascata, ou seja, plug-ins que adicionam funcionalidades a outros plug-ins;

[35] **OBS.** No caso do UnBBayes, a licença GPL do core exige que plug-ins do UnBBayes também sejam lançadas em licença compatível com GPL.

[36] O descritor de plug-in do JPF é o conteúdo minimal e principal de um plug-in. Portanto, é possível inclusive criar plug-ins compostos somente pelo descritor.

5. hot-plug - plug-ins deverão ser recarregáveis sem a necessidade de reiniciar a aplicação, para que a variabilidade seja realmente resolvida em tempo de execução;

6. gerenciador flexível de I/O - requisições de entrada e saída (*i.e.* requisições de leitura ou escrita de arquivos) deve ser multiplexada a plug-ins de acordo com a característica do arquivo, e se um conflito (situação em que mais de um plug-in declara ser compatível com um arquivo) ocorrer, a aplicação deve perguntar ao usuário qual plug-in usar.

As funcionalidades 1 até 4 são automaticamente resolvidas por JPF. O suporte à funcionalidade 5 é também oferecida pelo JPF, mas o UnBBayes precisou ser alterado para que eventos de GUI (*e.g.* seleção de um item de menu para "recarregar plug-ins") sejam traduzidos para requisições de recarregamento de plug-ins do JPF. *Listener design pattern* (49) foi utilizado para esse fim.

A funcionalidade 6 foi implementada alterando-se interfaces das classes de entrada e saída (Input/Output - I/O) do UnBBayes para que implementem *chain-of-responsibility design pattern* (49), que basicamente oferece um método que verifica se um determinado arquivo pode ser tratado pela classe cadastrada como plug-in de I/O. Por padrão, as classes de I/O existentes no momento somente testam a extensão do arquivo.

Na visão de implementação, a maior vantagem oferecida pelo JPF é a isolação dos carregadores de classes (*class loaders*). Cada plug-in do JPF possui seu próprio carregador associado, que por sua vez é responsável pelo carregamento de seus próprios recursos (*i.e.* classes e outros arquivos afins). Dependências entre plug-ins são resolvidas organizando-se os carregadores em uma hierarquia, delegando-se as requisições de carregamento em classes desconhecidas para o carregador pai.

No UnBBayes, somente plug-ins na pasta "plugins" serão carregados (como ilustrado na Figura 3.8). A classe `UnBBayesPluginContextHolder`, presente no pacote `unbbayes.util.extension.manager` foi implementada para poder encapsular o acesso ao JPF. Todas as variações foram refatoradas como plug-ins.

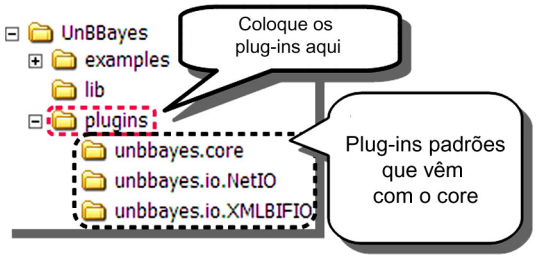

Figura 3.8: Típica estrutura de pastas do UnBBayes.

3.4.1 Arquitetura Geral do UnBBayes - Após Refatoração

Como resultado da refatoração, o core do UnBBayes tornou-se mais enxuto. Tecnicamente, o core do UnBBayes agora oferece duas funcionalidades primárias:

- uma infra-estrutura de base (*e.g.* suporte a janelas visuais, acesso ao armazenamento, gerência de preferências, suporte e gerenciamento de plug-ins) e

- funcionalidades para manipulação (*e.g.* edição, armazenamento, inferência, avaliação) de BN convencionais.

Os seguintes pacotes compõem o core do UnBBayes. As inter-dependências estão ilustradas na Figura 3.9.

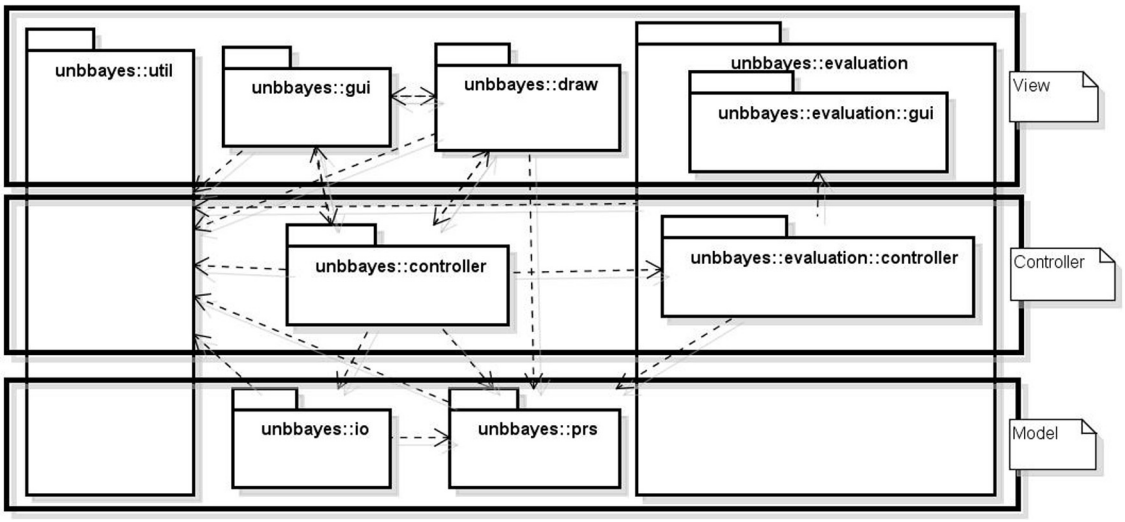

Figura 3.9: Diagrama de classes UML ilustrando a visão MVC do UnBBayes, em nível de pacotes. Esta figura já representa a arquitetura após a refatoração.

- `unbbayes.controller`: permanece com classes de controladora do modelo MVC.
- `unbbayes.draw`: este pacote foi criado pela equipe da George Mason University e contém classes adaptadoras que determinam como os elementos do grafo (*e.g.* nós, arestas) são desenhados na tela. Permite diversas funcionalidades de desenho, como redimensionamento, coloração, sombreamento, e movimentação. Classes deste pacote são normalmente utilizadas para desenhar grafos em componentes disponíveis no pacote `unbbayes.gui` .
- `unbbayes.evaluation`: contém classes para avaliação de BN.
- `unbbayes.gui`: contém classes gerais da GUI (*e.g.* painéis, janelas, formulários, tabelas).
- `unbbayes.io`: contém classes para armazenamento. Por exemplo, permite armazenamento de BN, configurações, preferências e logs em arquivos.
- `unbbayes.prs`: contém classes representando estruturas de dados e algoritmos (*i.e.* classes de modelo em uma arquitetura MVC). Elementos conceituais (*e.g.* `Node`, `Edge`, `Network`, `JunctionTree`, `PotentialTable`) e operações sobre as tais pertencem a este pacote. PRS significa ***P**robabilistic **R**easoning **S**ystem*. Classes em `unbbayes.prs.bn` representam uma BN, classes em texttttunbbayes.prs.id representam IDs e classes em `unbbayes.prs.hybridbn` representam uma BN híbrida, os quais possuem nós contínuos (uma RV com valores numéricos contínuos - não discretos).
- `unbbayes.example`: contém classes que exemplificam o uso do UnBBayes como uma API.

- `unbbayes.util`: agrega classes de utilidade, que são classes de uso geral utilizados pelo programa inteiro. Classes neste pacote provêm funcionalidades como: manipulação avançada de coleções, depuração, classes abstratas para se aderir a design patterns.
- `*.resources`: pacotes neste padrão de nomenclatura contém classes de "recurso", que provêem funcionalidades para nacionalização. O carregador de classes (*Java class loader*) do UnBBayes consegue selecionar automaticamente a classe, de recurso, mais apropriada para de acordo com a configuração do computador do usuário. Vide Seção 3.4.5 para maiores detalhes.
- `*.extension`: pacotes nesta nomenclatura contém classes implementando funcionalidades relacionadas à infra-estrutura de plug-ins.
- `*.exception`: pacotes nesta nomenclatura contém classes representando exceções (103).

As Seções 3.4.1 e 3.4.1 provêm diagramas detalhando a estrutura de classes no contexto MVC. Classes irrelevantes foram ocultadas, para facilitar a visualização.

Adicionalmente, foram também criados 6 pontos de extensão, para permitir extensão de funcionalidades do core.

- Module - um meio genérico para a criação de funcionalidades relativamente autossuficientes no UnBBayes, como janelas internas que são invocadas ao acionar menus ou botões da barra de ferramenta principal do UnBBayes (vide Figura 3.10 para um diagrama UML);
- InferenceAlgorithm - adiciona novos algoritmos de inferência Bayesiana (vide Figura 3.11);
- PNIO - adiciona novas classes de I/O para BN convencional (vide Figura 3.11);
- PluginNode - um meio para adicionar novos tipos de nós da BN (vide Figura 3.12);
- ProbabilityFunctionPanel - um meio para adicionar novos formulários de edição da distribuição de probabilidades de um nó da BN (vide Figura 3.12);
- ResourceBundle - plug-ins especiais que utilizam JPF e ResourceBundle do Java em conjunto para automaticamente resolver requisitos de nacionalização, de acordo com configurações de localidade do sistema operacional do usuário. Como o formato de uma classe ResourceBundle não é uma característica nova deste trabalho, não será detalhado neste documento.

Detalhes sobre o conteúdo do descritor (plugin.xml), que especifica as restrições impostas aos pontos de extensão do core, podem ser encontradas no repositório SVN do projeto UnBBayes (15).

Classes Controladoras e de Visão

A seguinte lista oferece uma breve descrição das classes ilustradas na Figura 3.13:

- `NamedWindowBuilder`: é uma *builder* que instancia uma janela interna. Cada janela instanciada por estes builders terão identificadores (nomes) diferentes e serão adicionadas em `MDIDesktopPane`. Esta classe elimina dependência direta de `MainController` para classes do módulo de BN (*e.g.* `NetworkWindow`).

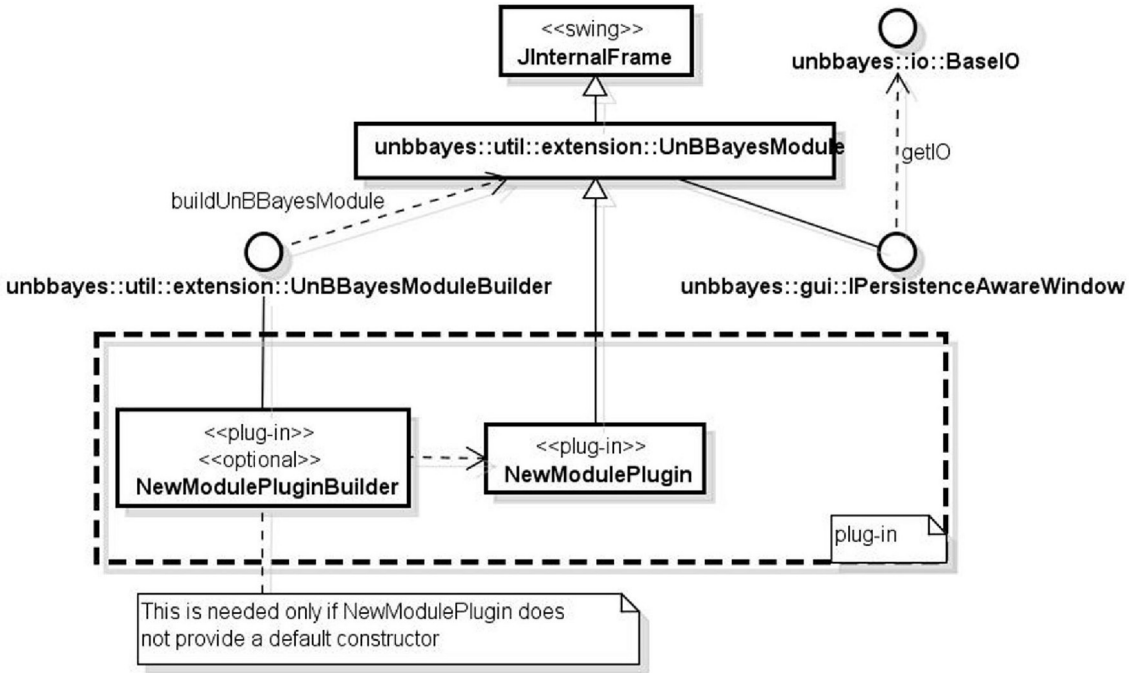

Figura 3.10: classes que devem ser criadas para se criar plug-ins para o ponto de extensão Module.

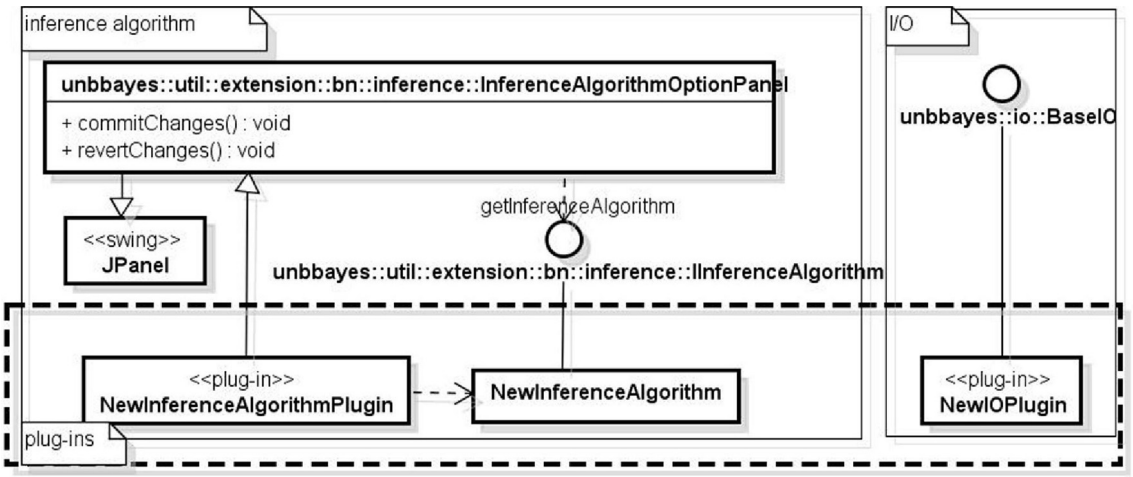

Figura 3.11: classes que devem ser criadas para se criar um plug-in para os pontos de extensão InferenceAlgorithm e PNIO.

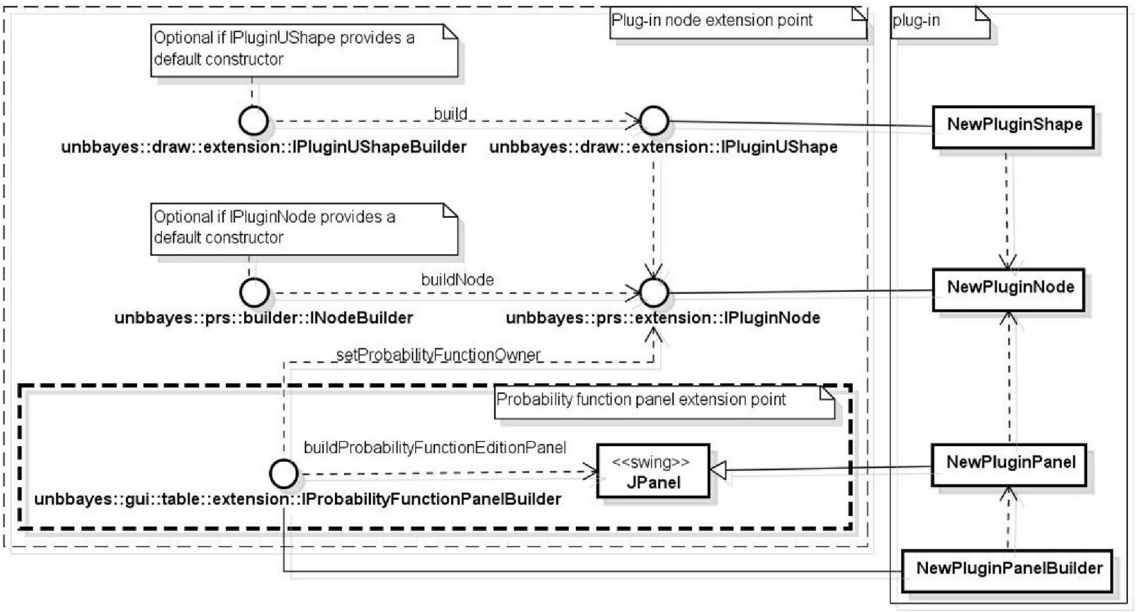

Figura 3.12: classes a serem criadas para os pontos de extensão PluginNode e ProbabilityFunctionPanel.

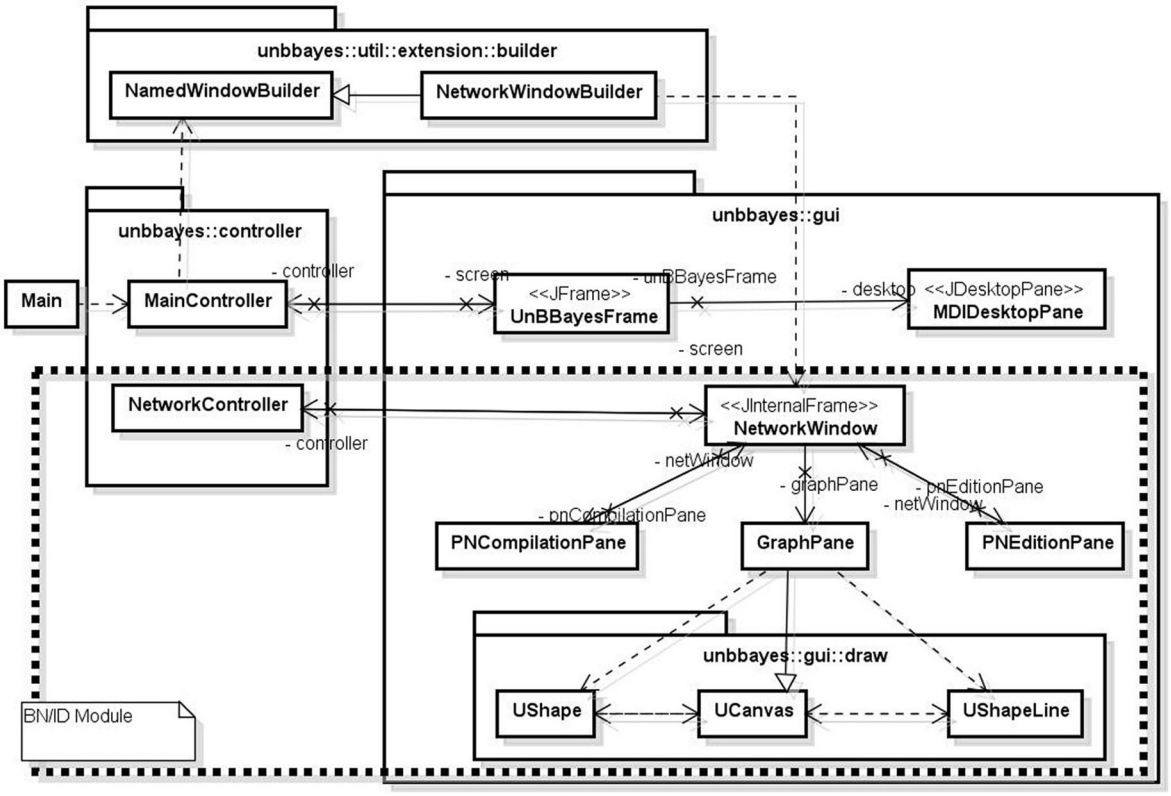

Figura 3.13: Classes do UnBBayes (*core*) implementando os papéis *View* e *Controller* do modelo MVC.

- `NetworkWindowBuilder`: um builder para `NetworkWindow`.
- `MainController`: esta é a controladora responsável por criar, carregar e armazenar redes genéricas suportadas pelo UnBBayes. Ele também é responsável pelas configurações do sistema.
- `UnBBayesFrame`: esta é uma extensão de `JFrame` do *swing* e representa a janela principal do UnBBayes.
- `MDIDesktopPane`: esta é uma extensão de `JDesktopPane` do *swing* para suporte a funcionalidades de MDI. Esta classe também é responsável por inicializar barras de rolagem quando janelas internas se deslocam para fora das dimensões visíveis da janela pai.
- `NetworkController`: esta é a controladora responsável por engatilhar operações do módulo de BN (*e.g.* inserção de nós, propagação de evidências).
- `NetworkWindow`: esta é uma janela interna onde a edição de BN é realizada.
- `PNCompilationPane`: este é um painel em `NetworkWindow` responsável por exibir uma BN compilada e por oferecer um meio para inserção de eviências e sua propagação.
- `GraphPane`: este é um painel em `NetworkWindow` responsável por exibir um grafo que representa a BN sendo editada.
- `PNEditionPane`: este é um painel em `NetworkWindow` responsável por oferecer ferramentas de edição de uma BN.
- `UShape`: esta classe é responsável pela renderização dos nós na tela, encapsulando então a classe de modelo `Node` (vide Seção 3.4.1).
- `UCanvas`: esta classe representa a tela onde os grafos são desenhados.
- `UShapeLine`: esta classe é responsável por renderizar um arco na tela, encapsulando então a classe de modelo `Edge` (vide Seção 3.4.1).

Classes de Modelo

Graças ao trabalho da equipe da GMU (responsável pela criação do pacote `draw`), classes nesta categoria não são mais responsáveis por atributos visuais, como posição ou cor; portanto, estas classes representam agora somente dados em memória. No entanto, como o relacionamento entre tais classes não sofreram alterações significativas comparadas a da Figura 3.2, serão omitidas.

3.4.2 Ponto de Extensão para Novos Módulos

A Figura 3.14 mostra algumas capturas de tela dos plug-ins de "**módulos**" do UnBBayes. Um plug-in de módulo provê um meio para se criar uma *feature* relativamente auto-suficiente no UnBBayes. Por exemplo, novos formalismos ou novas aplicações completas podem ser incorporadas implementando-se este ponto de extensão. No vocabulário do UnBBayes, um *módulo* é basicamente uma nova janela interna (*i.e.* instância de `JInternalFrame` do *swing*) que é inicializada quando eventos em botões ou barras de ferramentas são ativadas. Essas janelas internas não precisam ser sempre visíveis; portanto, podemos criar novos módulos sem

se criar uma “janela” completa de fato (*e.g.* assistentes - *wizards* - ou *pop-ups* podem ser emulados desta maneira). O código XML da listagem 3.1 define o ponto de extensão para módulos:

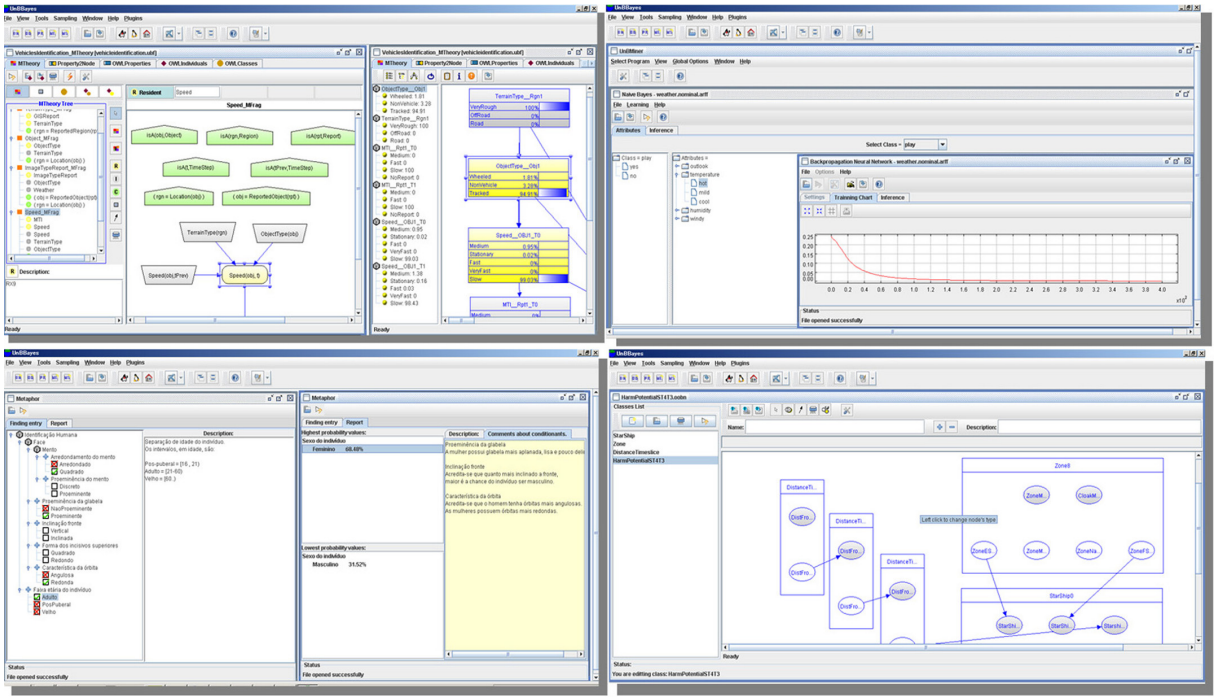

Figura 3.14: Captura de tela de alguns plug-ins de módulos do UnBBayes: MEBN (superior esquerdo), UnBMiner (superior direito), Metáfora de Identificação Humana (inferior esquerdo) e Redes Bayesianas Orientadas a Objeto (inferior direito).

Listagem 3.1: Especificação de um ponto de extensão de plug-ins de módulos.

```
1  <extension-point id="Module">
2      <!--subclasse de unbbayes.util.extension.UnBBayesModule-->
3      <parameter-def id="class"/>
4      <!-- subclass of unbbayes.util.extension.UnBBayesModuleBuilder -->
5      <parameter-def id="builder" multiplicity="none-or-one" />
6      <parameter-def id="name" multiplicity="none-or-one" />
7      <parameter-def id="description" multiplicity="none-or-one" />
8      <parameter-def id="icon" multiplicity="none-or-one" />
9      <!--Nome (string) do item de menu onde o modulo sera adicionado-->
10     <parameter-def id="category" multiplicity="none-or-one" />
11 </extension-point>
```

A Figura 3.15 ilustra as classes principais do ponto de extensão para módulos. A classe `UnBBayesModule` é a classe mais importante desse ponto de extensão e é uma subclasse de `JInternalFrame` do *swing* (portanto, representa uma janela interna no UnBBayes). Classes que implementam `IPersistenceAwareWindow` são classes da GUI contendo referências a classes de entrada e saída (I/O), e como `UnBBayesModule` a implementa, uma implementação de um módulo deve reconhecer os tipos de arquivos tratáveis, para que o *core* do UnBBayes possa consistentemente delegar requisições de I/O aos módulos que os conseguem tratar.

`NewModulePlugin` e `NewModulePluginBuilder` representam meramente as classes que devem ser de fato providas pelos plug-ins. Um *builder* só será necessário se `NewModulePlugin` não prover um construtor padrão sem parâmetros. O código XML na listagem 3.2 exemplifica um descritor[37] de um módulo:

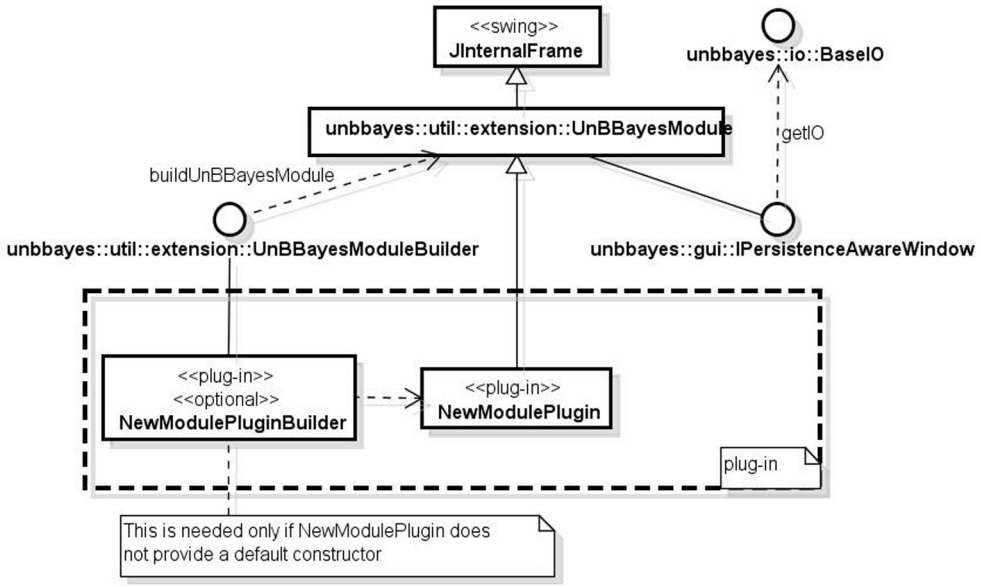

Figura 3.15: Este diagrama de classes indica as classes qeu devem ser estendidas para se criar um plug-in de módulo. Vide listagem 3.1.

Listagem 3.2: Um exemplo simplificado (mas completo sintaticamente) de um descritor de plug-in que implementa um módulo e especifica um outro ponto de extensão simultaneamente.

```
1  <?xml version="1.0" ?>
2  <!DOCTYPE plugin PUBLIC "-//JPF//Java Plug-in Manifest 1.0"
3        "http://jpf.sourceforge.net/plugin_1_0.dtd">
4  <plugin id="unbbayes.prs.mebn" version="1.7.0">
5     <requires><import plugin-id="unbbayes.util.extension.core"/></requires>
6     <runtime>
7        <library id="unbbayes.prs.mebn" path="unbbayes.prs.mebn-1.7.0.jar"
8        type="code" >
9           <export prefix="*"/>
10       </library>
11    </runtime>
12    <extension-point id="MEBNEditorPanel">
13       <parameter-def id="class"/>
14       <parameter-def multiplicity="none-or-one" id="name"/>
15       <parameter-def multiplicity="none-or-one" id="icon"/>
16       <parameter-def multiplicity="none-or-one" id="description"/>
17    </extension-point>
18    <extension plugin-id="unbbayes.util.extension.core"
19    point-id="Module" id="MEBN">
20       <parameter id="class" value="unbbayes.gui.mebn.MEBNNetworkWindow" />
```

[37]OBS. Os exemplos providos neste capítulo são simplificações. Descritores completos podem ser encontrados nos próprios plug-ins em <http://sourceforge.net/projects/unbbayes/>.

```
21          <parameter id="name" value="MEBN" />
22          <parameter id="builder"
23          value="unbbayes.gui.mebn.extension.MEBNWindowBuilder" />
24          <parameter id="description" value="Multi␣Entity␣Bayesian␣Network" />
25          <parameter id="icon" value="new-mebn.png" />
26          <parameter id="category" value="bn" />
27      </extension>
28  </plugin>
```

3.4.3 Plug-ins para Plug-ins - Hiearquia de Depenências

Como foi previamente mostrado pelo descritor na listagem 3.1 da Seção 3.4.2, um plug-in também pode declarar seus próprios pontos de extensão (vide ponto de extensão `MEBNEditorPanel` na listagem). Com isso, torna-se possível a criação de uma hierarquia de extensões. Essa hierarquia pode gerar um grafo de dependências entre plug-ins, que pode ser especificada com a *tag* `requires` no cabeçaho do descritor. A Figura 3.16 esquematiza a ligação dos componentes (plug-ins) em pontos de extensão. A Figura 3.17 contém uma captura de tela que serve como exemplo de um plug-in implementando o ponto de extensão `MEBNEditorPanel` (um ponto de extensão declarado por um plug-in).

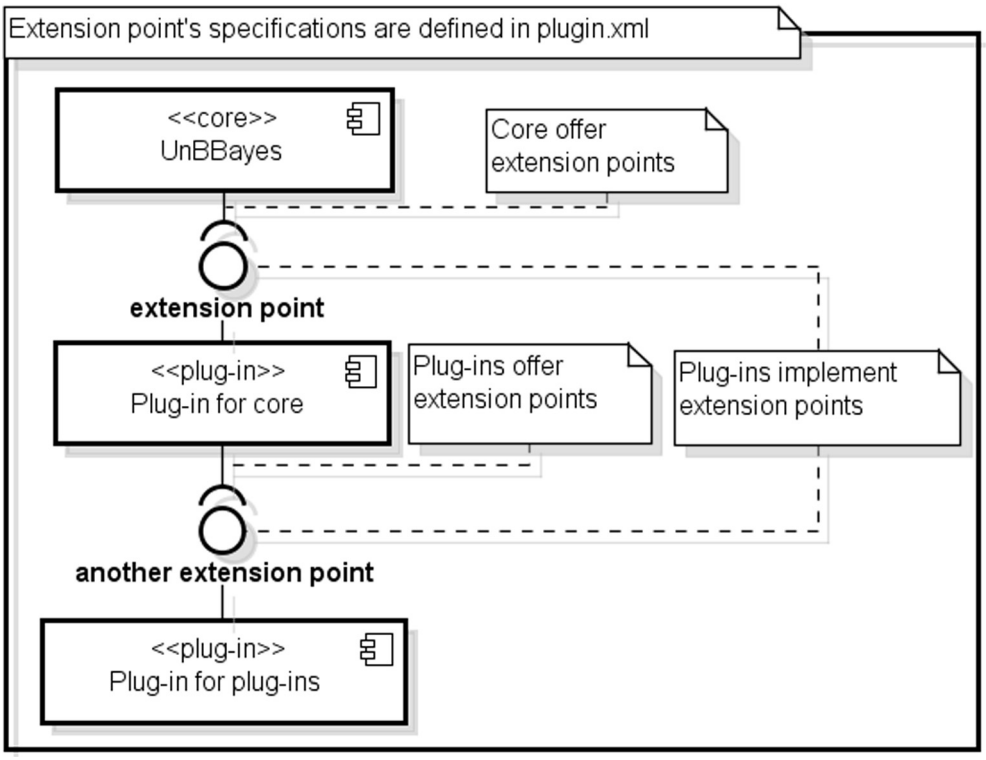

Figura 3.16: Diagrama de componentes ilustrando a ligação hierárquica de plug-ins em pontos de extensão.

A hierarquia de dependências não é diretamente gerenciada pelo UnBBayes, pois todo esse gerenciamento é automaticamente realizado pela JPF. Uma *feature* para resolução (download) automático de dependências de plug-ins está previsto como um trabalho futuro.

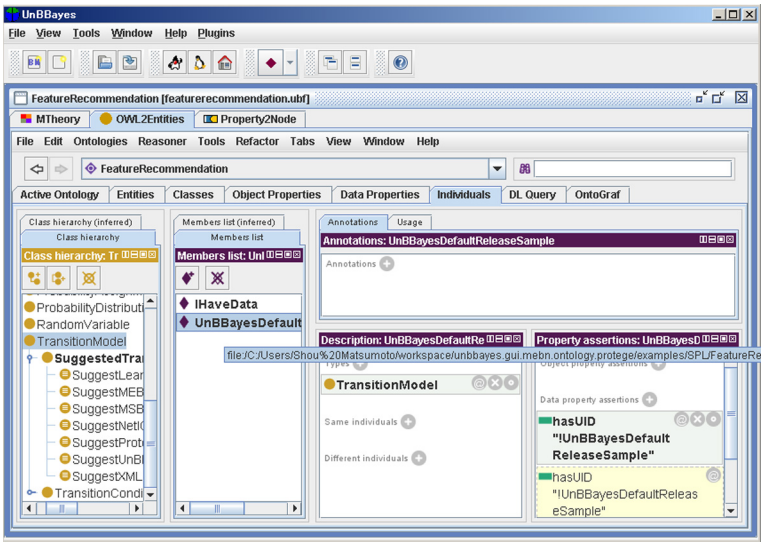

Figura 3.17: Captura de tela de um plug-in para plug-in (um plug-in oferecendo uma nova *feature* a um outro plug-in). Este provê um painel adicional para se editar ontologias usando a GUI do Protégé.

3.4.4 Pontos de Extensão para o Módulo de BN

A Figura 3.18 captura exemplos de plug-ins que estendem funcionalidades dos pontos de extensão *InferenceAlgorithm*, *PNIO*, *PluginNode* e *ProbabilityFunctionPanel*; plug-ins que oferecem funcionalidades adicionais ao módulo de BN do UnBBayes. As funcionalidades de BNs convencionais permanecem como componentes do core do UnBBayes.

Como mencionado nas seções anteriores, todos os metadados de um plug-in é especificado no seu descritor (*plugin.xml*). Esse metadado inclui especificações sobre quais pontos de extensão estão disponíveis para serem implementadas por plug-ins, e quais pontos de extensão o plug-in atual implementa. A *tag* `extension-point` indica os pontos de extensão disponíveis, enquanto que os pontos de extensão implementados são marcados pela *tag* `extension` tag. As listagens 3.3, 3.4, 3.5 e 3.6 ilustram exemplos de uso da *tag* `extension-point` para se especificar alguns pontos de extensão previamente apresentadas na seção anterior (3.4.1), respectivamente para novos algoritmos de inferência, novas classes de I/O (entrada e saída), novos tipos de nós e novos paineis/formulários para se editar a distribuição de probabilidade condicional.

Listagem 3.3: Especificação do ponto de extensão para plug-ins que implementam algoritmos de inferência.

```
1 <extension-point id="InferenceAlgorithm">
2 <!--deve implementar InferenceAlgorithmOptionPanel,-->
3 <!--pacote unbbayes.util.extension.bn.inference-->
4     <parameter-def id="class"/>
5 </extension-point>
```

Listagem 3.4: Especificação do ponto de extensão de plug-ins de I/O.

```
1 <extension-point id="PNIO">
2     <parameter-def id="class"/> <!-- implements unbbayes.io.BaseIO -->
```

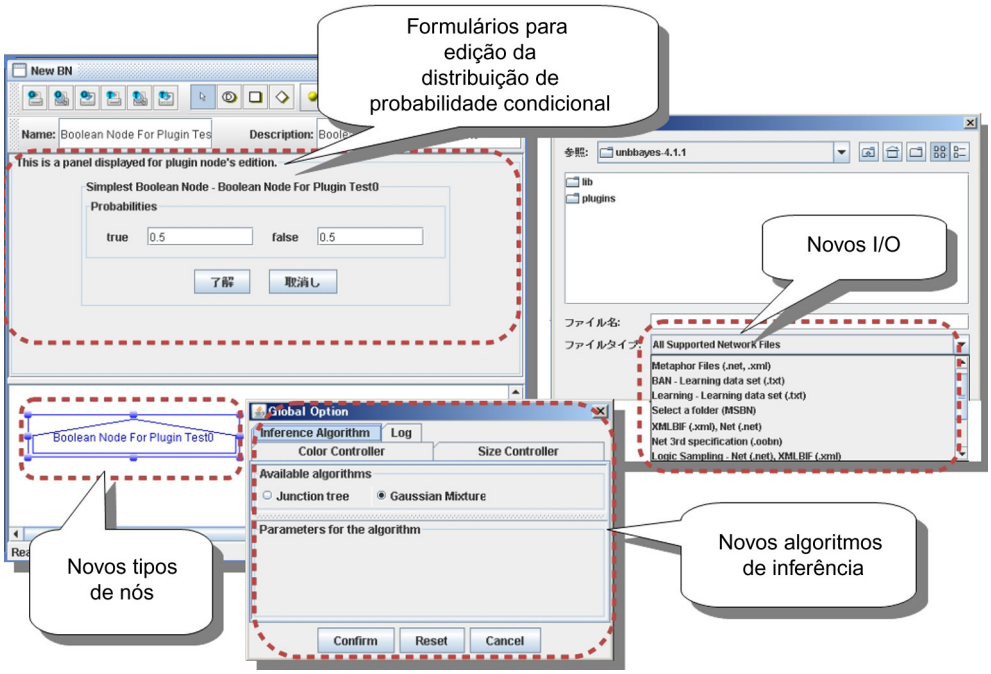

Figura 3.18: O módulo core (BN) declara alguns pontos de extensão para funcionalidades de BN.

```
3     <parameter-def id="name"/> <!-- String -->
4 </extension-point>
```

Listagem 3.5: Especificação de ponto de extensão para um novo tipo de nó.

```
1  <extension-point id="PluginNode">
2  <!--implementa unbbayes.prs.extension.IPluginNode-->
3     <parameter-def id="class"/>
4  <!-- implementa unbbayes.draw.extension.IPluginUShape -->
5     <parameter-def id="shapeClass"/>
6     <parameter-def id="name"/> <!--String-->
7  <!-- estende IProbabilityFunctionPanelBuilder -->
8  <!-- pacote unbbayes.gui.table.extension.-->
9     <parameter-def id="panelBuilder"/>
10    <parameter-def id="description" multiplicity="none-or-one" />
11 <!--Caminhos para imagens-->
12    <parameter-def id="icon" multiplicity="none-or-one" />
13    <parameter-def id="cursor" multiplicity="none-or-one" />
14 </extension-point>
```

Listagem 3.6: Especificação de ponto de extensão para novos paineis/formularios para se editar a distribuição de probabilidade condicional.

```
1  <extension-point id="ProbabilityFunctionPanel">
2  <!--subclasse existente de unbbayes.prs.Node-->
3     <parameter-def id="class"/>
4     <parameter-def id="name"/> <!--String-->
5  <!-- estende IProbabilityFunctionPanelBuilder -->
6  <!-- pacote unbbayes.gui.table.extension-->
7     <parameter-def id="panelBuilder"/>
```

```
8     <parameter-def id="description" multiplicity="none-or-one" />
9  <!--Caminho para imagem-->
10    <parameter-def id="icon" multiplicity="none-or-one" />
11 </extension-point>
```

A *tag* `parameter-def` define quais parâmetros devem ser oferecidos pelo plug-in que implementa o ponto de extensão(tais parâmetrosthose são obrigatórios se a quantidade não for explicitada). Figuras 3.19 e 3.20 ilustram a estrutura de classes dos pontos de extensão mencionadas anteriormente.

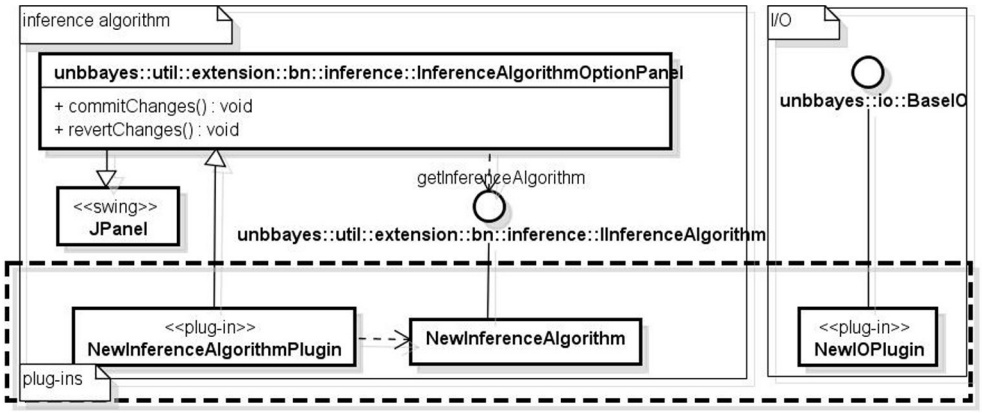

Figura 3.19: Este diagrama de classes indica as classes que devem ser estendidas para se criar plug-ins para algoritmos de inferência e I/O. Veja listagem 3.4.

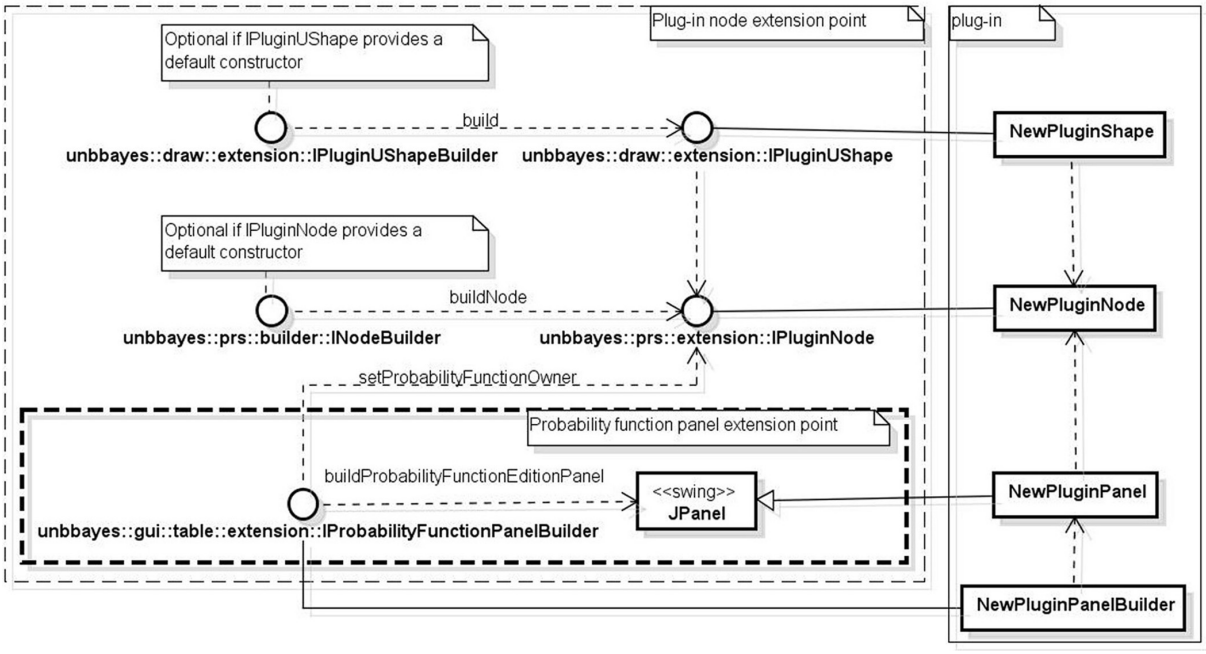

Figura 3.20: Este diagrama indica as classes que devem ser estendidas para se criar um plug-in para novos tipos de nós e novos formulários de edição da distribuição de probabilidade condicional. Vide listagem 3.5 e 3.6.

A lista abaixo provê uma breve explicação das classes principais apresentadas:

- `InferenceAlgorithmOptionPanel`: é um `JPanel` do *framework swing* para permitir a configuração de um algoritmo de inferência (*i.e.* um objeto da classe `IInferenceAlgorithm`).
- `IInferenceAlgorithm`: classes implementando esta inferface provê funcionalidades para se compilar e propagar evidências em BN.
- `BaseIO`: classes implementando esta interface oferece funcionalidades para se armazenar ou carregar modelos baseados em redes probabilísticas, de forma geral.
- `IPluginUShape`: classes implementando esta interface são adaptadoras que provêem funcionalidades para se desenhar objetos de `IPluginNode` na tela.
- `IPluginNode`: classes implementando esta interface representa nós na BN.
- `IProbabilityFunctionPanelBuilder`: classes implementando esta interface criam instâncias de `JPanel` que servem para se especificar a distribuição de probabilidades condicionais de um nó.
- `*Builder`: classes com esse padrão de nomenclatura são classes *builder* que criam instâncias de outras classes. Por causa de limitações na infraestrutura de plug-ins, classes especificadas na *tag* `parameter-def` devem fornecer um construtor padrão sem parâmetros. Como isso não é viável sempre (*e.g.* o construtor padrão pode estar marcado como *privado*, impedindo o acesso), um *builder* (oferecendo um construtor padrão sem parâmetro) pode ser utilizado como um inicializador de uma outra classe desejada.
- `New*`: classes com esse padrão não existem de fato. Representam classes externas e devem ser providas pelos plug-ins.

3.4.5 Plug-ins de Recursos de Nacionalização

Plug-ins de recursos (*resource*) podem ser utilizados para prover nacionalização. Este ponto de extensão difere substancialmente dos outros porque este combina a infra-estrutura de plug-ins com o mecanismo de *resoure bundle*[38] do Java para possibilitar o carregamento automático de classes de acordo com configurações regionais do sistema operacional do usuário.

No mecanismo de carregamento de *resoure bundle*, quando uma classe estende `java.util.ListResourceBundle`, o carregador de classes da máquina virtual Java automaticamente passa a gerenciar essa classe como um recurso de nacionalização. A maior vantagem de se combinar este mecanismo com o JPF é que não há a necessidade de se explicitar um ponto de extensão somente para habilitar variabilidade em classes de recursos, pois este passa a ser gerenciada pelo carregador da máquina virtual Java, não mais pela JPF. Em outras palavras, um plug-in de plug-ins pode ser criado sem se declarar um novo ponto de extensão, desde que seja no âmbito de plug-ins de recursos de nacionalização.

O UnBBayes utiliza o carregador de classes do Java para automaticamente selecionar a classe de recurso (`java.util.ListResourceBundle`) apropriada, dependendo da opção de nacionalização (*locale*) do usuário. Esta abilidade também está presente em carregadores de classes do JPF, pois reusam funcionalidades do carregador de classes do Java; portanto o mesmo

[38]Maiores informações sobre o *resoure bundle* do Java podem ser encontradas em <http://download.oracle.com/javase/1.5.0/docs/api/java/util/ResourceBundle.html>.

mecanismo pode ser aplicado para se carregar no *core* as classes de recursos presentes em plug-ins. A integração de ambos escopos (*i.e.* carregamento de plug-ins pelo JPF e carregamento de recursos pelo Java) é feita pela classe `unbbayes.util.ResourceController` do *core* do UnBBayes, que utiliza o padrão *chain of responsibility* para selecionar corretamente o carregador de classes com acesso à classe de recurso mais apropriada.

Apesar da grande flexibilidade, classes compondo os plug-ins de recurso possuem algumas restrições inerentes do mecanismo de *resource bundle* do Java. Com exceção das classes de recursos padrões, que geralmente representam recursos na língua inglesa e são carregadas na ausência de melhores opções, os nomes das classes de recursos devem seguir um padrão especial:

$$< \textit{Nome da classe} > \_ < \textit{codigo da regiao} > .$$

E.g. `MinhaClasse_en`, `MinhaClasse_jp`, `MinhaClasse_pt_br`. O conteúdo dessas classes são basicamente um vetor de textos indexados por chaves.

Dado a classe de recurso da listagem 3.7, o código da listagem 3.8 emitirá a mensagem "`Ola Mundo`" em um ambiente em português, e "`Hello World`" em outros ambientes (por falta de uma opção exata para essa linguagem). O código XML da listagem 3.9 exemplifica um descritor de plug-ins de recurso.

Listagem 3.7: Um recurso simples e sua versão em português.

```
1 class MyResources extends java.util.ListResourceBundle {
2     public static Object[][] contents = {{"myKey", "Hello World"}};
3     public Object[][] getContents() {return contents;}
4 }
5
6 class MyResources_pt extends java.util.ListResourceBundle {
7     public static Object[][] contents = {{"myKey", "Ola Mundo"}};
8     public Object[][] getContents() {return contents;}
9 }
```

Listagem 3.8: Um exemplo de uso de uma classe de recursos.

```
1     java.util.ResourceBundle resource =
2         unbbayes.util.ResourceController.newInstance().getBundle(
3             MyResources.class.getName()
4         );
5     System.out.println(resource.getString("myKey"));
```

Listagem 3.9: Exemmplo de descritor de plug-in de recursos para adicionar língua japonesa ao módulo de OOBN.

```
1  <?xml version="1.0" ?>
2  <!DOCTYPE plugin PUBLIC "-//JPF//Java Plug-in Manifest 1.0"
3  "http://jpf.sourceforge.net/plugin_1_0.dtd">
4  <plugin id="unbbayes.oobn.resources.ja" version="1.1.0">
5      <requires>
6          <import plugin-id="unbbayes.util.extension.core"/>
7          <import plugin-id="unbbayes.prs.oobn"/>
8      </requires>
9      <runtime>
10         <!-- JAR com classes de recurso -->
```

```
11          <library id="unbbayes.oobn.resources.ja"
12              path="unbbayes.oobn.resources.ja-1.1.0.jar" type="code" >
13            <export prefix="*"/>
14          </library>
15      </runtime>
16      <!--especifica qual ponto de extensao este plugin implementa-->
17      <!--O modulo OOBN nao prove este ponto, mas podemos o usar porque
18            plugins de recursos se comportam de maneira especial-->
19      <extension plugin-id="unbbayes.util.extension.core"
20                point-id="ResourceBundle" id="oobn_gui_ja">
21          <parameter id="class"
22                value="unbbayes.gui.oobn.resources.OOBNGuiResource_ja" />
23      </extension>
24  </plugin>
```

Capítulo 4

Extração de *Features* como Plug-ins

Neste capítulo são discutidos as arquiteturas dos plug-ins que implementam os pontos de extensão apresentados no Capítulo 3, apontando basicamente as diferenças entre a arquitetura antes e após da extração de funcionalidades e sua refatoração como plug-ins[39].

Como fruto da refatoração extrativa de diversas *features* pré-existentes e de atualizações recentes, o projeto UnBBayes já oferece um repositório que inclui plug-ins para BN, ID, MSBN, redes bayesianas híbridas (HBN), OOBN, PRM, MEBN e PR-OWL, aprendizagem de parâmetros de BN, aprendizagem estrutural e aprendizagem incremental de BN, amostragem estatística de dados, classificação e avaliação de performance de BN, data mining e muitos outros algoritmos de inferência bayesiana.

4.1 Visão Geral da Extração de *Features*

Muitas abordagens com SPL utilizam extração automática de *features* (*i.e.* execução de algoritmos para se extrair automaticamente as features e agrupando-as de acordo com similaridades e interdependências); no entanto, como previamente comentado na Seção 1.3, uma abordagem automática não se mostrou necessária no caso do UnBBayes pelos seguintes motivos:

- como o UnBBayes é basicamente um framework para aplicações em IA, cada funcionalidade observável estava intimamente ligada a algum formalismo conhecido em IA;
- cada iteração no desenvolvimento de novas funcionalidades no UnBBayes era fruto de pesquisa de graduandos ou pós-graduandos do GIA/UnB;
- monografias e dissertações descreviam uma *feature* em particular ou um conjunto de *features* relacionados, pois uma *feature* (ou um conjunto de *features* dependentes) estava sempre relacionado a um projeto de pesquisa.

Em outras palavras, o UnBBayes já podia ser considerado uma família de softwares com *features* bem distintas e com documentações separadas por *features*, antes mesmo da refeatoração. A Figura 4.1 ilustra o UnBBayes e sua família de softwares. No entanto, os membros da

[39]Não é o objetivo deste capítulo descrever todas as configurações possíves de plug-ins ou os detalhes sobre os formalismos implementados, mas somente descrever as alterações feitas para se refatorar as variações do UnBBayes como plug-ins na nova arquitetura.

família do UnBBayes eram um conjunto de softwares completamente independentes, com redundâncias e exigindo alteração de código fonte nos membros antigos para se permitir a criação de novos membros; exigindo então um esforço considerável na manutenção. A completa indepenência dificultava que correções de erros se propaguem a todos os membros, e a necessidade de alteração em nível de código fonte aumentava o risco de inserção de erros. A redundância causava o aumento desnecessário do tamanho de cada membro da família, dificultando a distribuição na Internet.

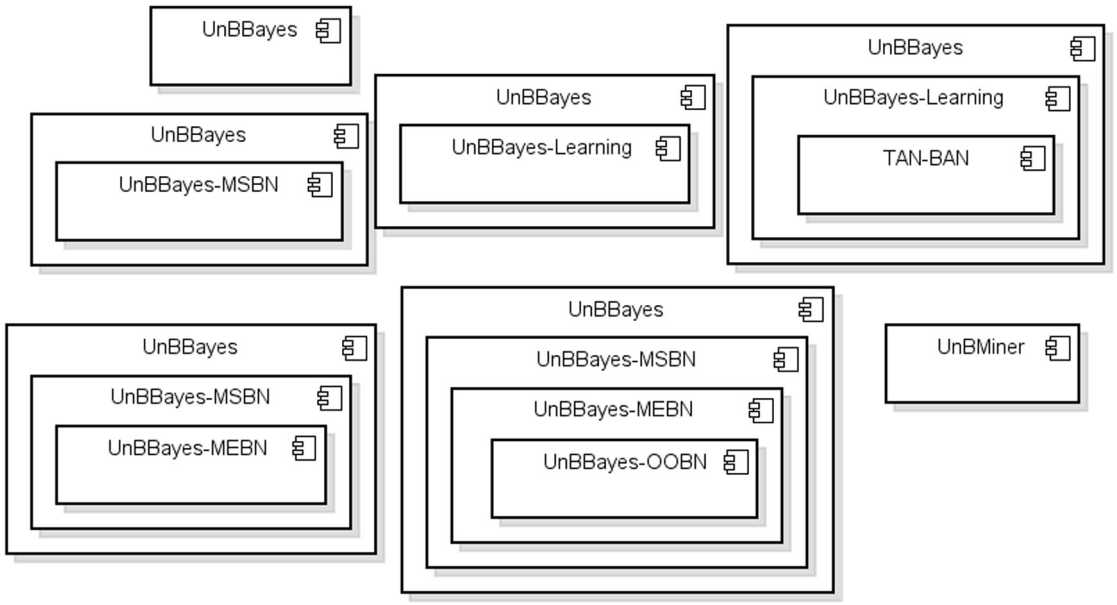

Figura 4.1: Famílias do UnBBayes antes da refatoração. Esta figura não apresenta todas as configurações possíveis, mas somente um subconjunto representativo.

A necessidade de alteração de código fonte das *features* antigas para se criar uma nova *feature* era particularmente um problema grave no GIA, pois os membros eram (e ainda são) tipicamente compostos por graduandos em etapa inicial de aprendizagem, eventualmente inserindo erros em rotinas que costumavam ser corretas.

A refatoração do UnBBayes (*core* - vide Capítulo 3) para uma arquitetura de plug-ins permitiu a extensão de *features* em tempo de execução, eliminando assim uma parcela significativa das necessidades de se realizar alterações em código fonte. A refatoração de *features* pré-existentes como plug-ins removerá redundância, pois os artefatos finais (plug-ins) somente conterão rotinas para um formalismo em particular e similaridades serão reusadas do *core* ou de outros plug-ins, diminuindo o tamanho de cada artefato e facilitando a distribuição via Internet. Adicionalmente, para que a correção em uma configuração se propague a outras configurações, bastaria substituir o plug-in errôneo pelo correto. A Figura 4.2 ilustra a refatoração dos softwares da Figura 4.1 a um ambiente de plug-ins.

O código fonte de um plug-in do UnBBayes é tratado como um novo projeto Maven e armazenado.

A Figura 4.3 apresenta a estrutura de pastas dos plug-ins do UnBBayes. Esta estrutura de pastas foi projetada de maneira a permitir execução a de códigos dos plug-ins, presentes em `src/main/java`, sem a necessidade de se mover os arquivos `.class` para a pasta `plugins`.

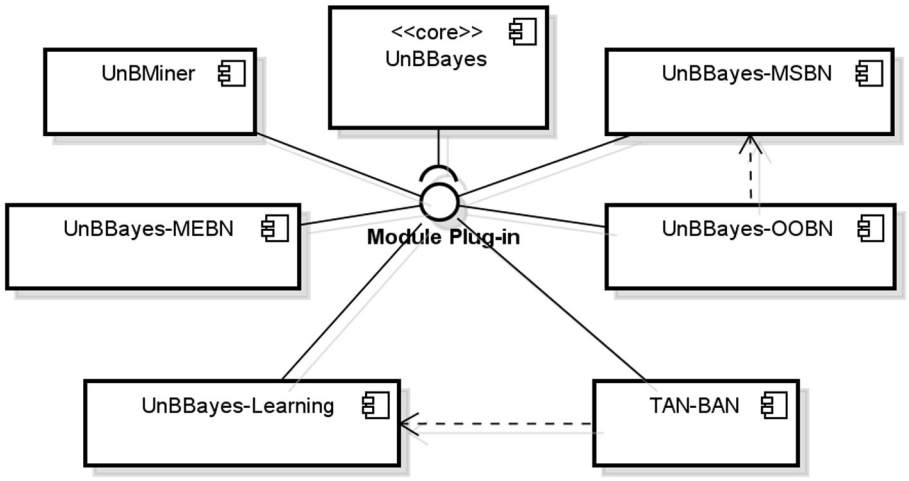

Figura 4.2: Famílias do UnBBayes após refatoração das configurações da Figura 4.1.

Nesta estrutura, ao se executar o método `main` da classe `UnBBayesMainDelegator`, a requisição será delegada ao *core* do UnBBayes (neste exemplo, contido em `unbbayes-4.0.0.jar`, carregada como dependência do Maven), que por sua vez executará o JPF (também carregada como dependência do Maven), que carregará os descritores presentes na pasta `plugins` da raiz. Os descritores eventualmente indicarão alguns nomes de classes localizadas em `src/main/java`. Como essa pasta está na *classpath* do projeto (por ser uma pasta padrão em projetos Maven), o carregamento de classes ocorrerá normalmente, de forma similar ao carregamento de plug-ins no ambiente de implantação. Essa estrutura de pastas facilita também na depuração, pois os depuradores do IDE permitirão sem configuração adicional a depuração de códigos fontes contidos em `src/main/java`.

4.2 Plug-ins de I/O

O UnBBayes oferecia previamente rotinas de I/O para 3 formatos diferentes de BNs convencionais: NET (extensão ".net" do HUGIN), XMLBIF e BNF. Classes que implementam esses três formatos permanecem nos pacotes que fazem parte do *core*; portanto, não foram refatorados como plug-ins. Entretanto, descritores de plug-ins que apontam para essas classees (no *core*) foram criados para simular plug-ins de I/O para os formatos NET, XMLBIF e BNF.

Esse tipo de construção (criação de descritores de plug-ins, mesmo não resultando em uma criação de um plug-in completo) é útil para permitir atualizações como *patches*. Novas classes de I/O podem substituir as classes que implementam NET, XMLBIF e BNF, sem sobreescrita de arquivos, utilizando-se o mesmo mecanismo de plug-ins oferecido pelo JPF (que permite com que classes antigas e novas coexistam no mesmo ambiente de execução, mas permitindo o acesso distinto ao se indicar o número da versão desejada). Aproveitando-se desse mecanismo de versionamento, podemos atualizar um I/O criando-se plug-ins de mesmo identificador, mas com versões maiores.

Um exemplo de atualização foi feito em XMLBIF versão 7, substituindo-se as rotinas do XMLBIF via versionador do JPF. O ponto de extensão `PNIO` foi implementado (vide listagem 4.1, que implementa a especificação da listagem 3.4). A classe `unbbayes.io.xmlbif.`

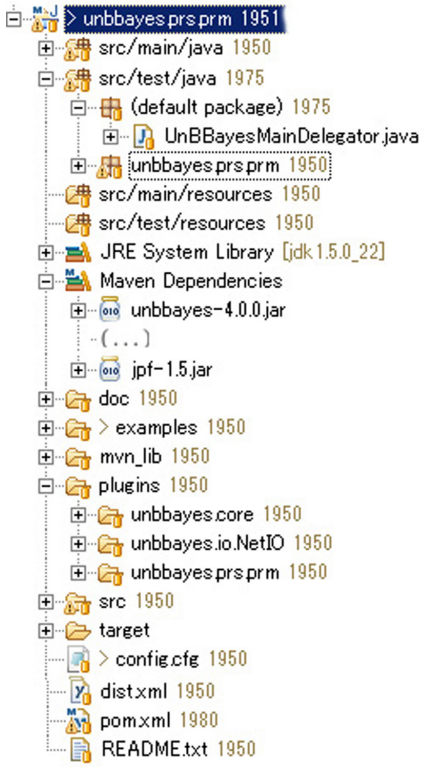

Figura 4.3: Estrutura de pastas do projeto maven para plug-ins do UnBBayes.

`version7.BaseXMLBIFIO`, contida no arquivo `edu.gmu.seor.prognos.unbbayesplugin.cps-1.0.0.jar` (ambos declarados no descritor) é a classe principal que substituirá as rotinas antigas do XMLBIF via versionador do JPF. Como foi mencionado na Figura 3.19, as classes que efetivamente oferem rotinas de I/O devem implementar a interface `BaseIO` (e `unbbayes.io.xmlbif.version7.BaseXMLBIFIO` o faz).

Listagem 4.1: Descritor do plug-in XMLBIF versão 7.

```
1  <?xml version="1.0" ?>
2  <!DOCTYPE plugin PUBLIC "-//JPF//Java Plug-in Manifest 1.0"
3      "http://jpf.sourceforge.net/plugin_1_0.dtd">
4  <plugin id="unbbayes.io.XMLBIFIO" version="2.0.0">
5      <requires>
6          <import plugin-id="unbbayes.util.extension.core"/>
7          <import plugin-id="edu.gmu.seor.prognos.unbbayesplugin.cps"/>
8      </requires>
9      <runtime>
10         <library id="edu.gmu.seor.prognos.unbbayesplugin.cps"
11         path="edu.gmu.seor.prognos.unbbayesplugin.cps-1.0.0.jar"
12         type="code" >
13           <export prefix="*"/>
14         </library>
15     </runtime>
16     <extension plugin-id="unbbayes.util.extension.core"
17     point-id="PNIO" id="XMLBIFIO">
18         <parameter id="class"
19         value="unbbayes.io.xmlbif.version7.BaseXMLBIFIO" />
```

```
20          <parameter id="name" value="XMLBIF Version 0.7" />
21      </extension>
22  </plugin>
```

Outros exemplos de descritor de plug-ins serão omitidos, para manter o capítulo menos extenso.

4.3 Plug-ins de Aprendizagem Bayesiana

O UnBBayes, antes de refatorado, oferecia quatro tipos de aprendizagem bayesiana: "aprendizagem" (parâmetros e estruturas via algoritmos como K2, B, CBL-A, CBL-B), TAN, BAN e aprendizagem incremental.

A principal preocupação na refatoração foi manter as classes de aprendizagem bayesiana como caixas pretas. Quatro novas subclasses de `UnBBayesModule` (classe que se deve estender para implementação do ponto de extensão `Module`) foram criadas para se representar respectivamente a "aprendizagem", TAN, BAN e aprendizagem incremental como plug-ins de módulos (vide Figura 4.4). São eles:

`unbbayes.learning.gui.extension.LearningModule`,

`unbbayes.learning.gui.extension.TANModule`,

`unbbayes.learning.gui.extension.BANModule` e

`IncrementalLearningModule`, do pacote `unbbayes.learning.incrementalLearning.gui.extension`.

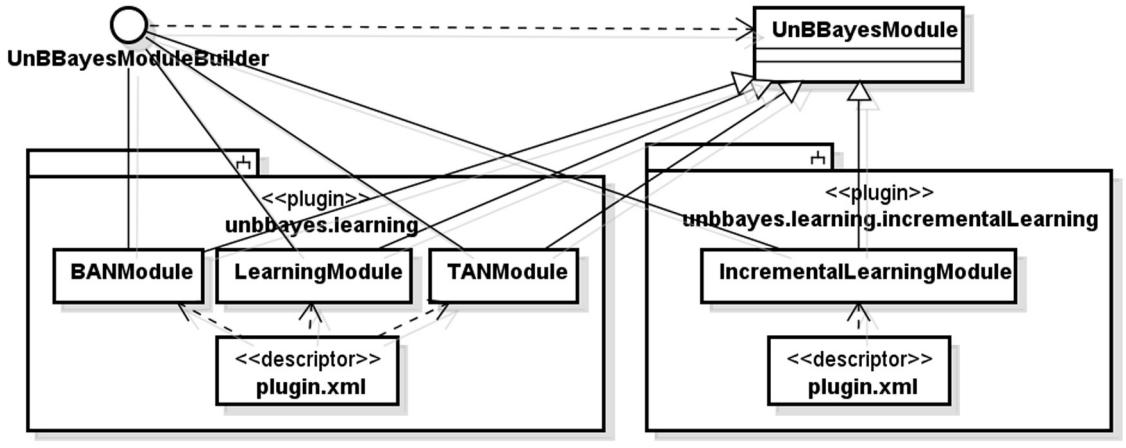

Figura 4.4: Diagrama de classes dos plug-ins dos módulos de aprendizagem. Quatro módulos foram agrupados em dois plug-ins como mostra a figura.

Os módulos de aprendizagem funcionavam na arquitetura antiga como "assistentes" (*wizards*) que eram engatilhadas automaticamente ao se instanciar a classe `unbbayes.learning`.

`ConstructionController`, portanto os novos plug-ins devem funcionar como a tal. Para esse fim, uma forma especial para se implementar os métodos em `UnBBayesModule` deve ser realizada. A Figura 4.5 ilustra como o módulo (subclasses de `UnBBayesModule`) de aprendizagem bayesina (o exemplo menciona somente `LearningModule`; mas são idênticos para `LearningModule`, `LearningModule` e `LearningModule`) é carregada para se simular um assistente.

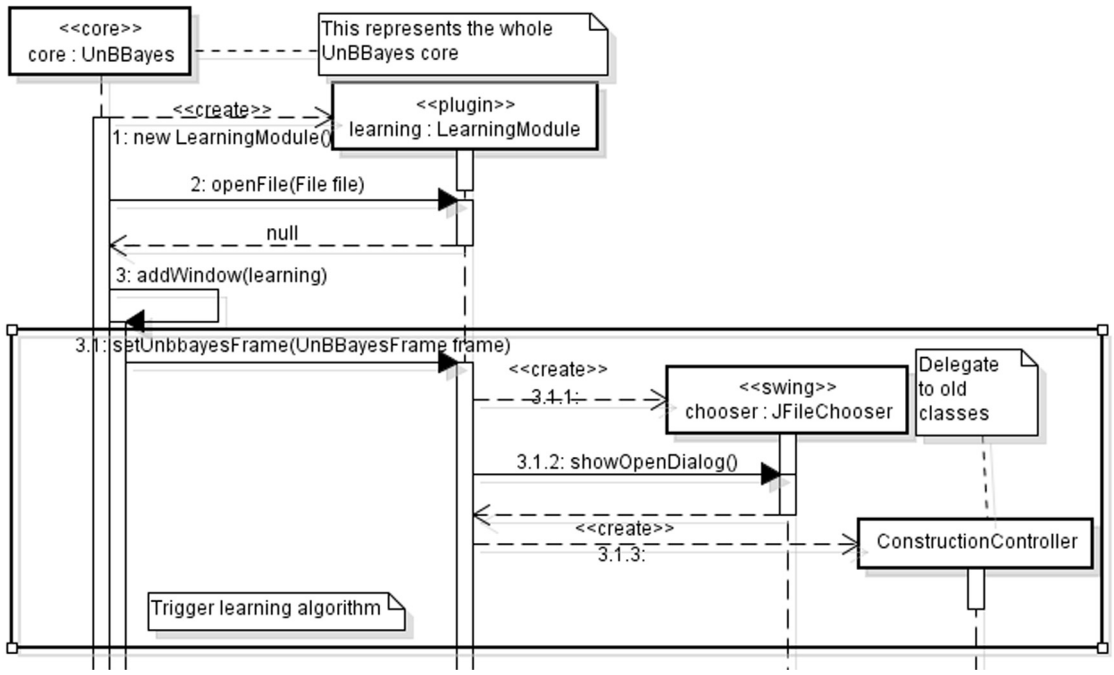

Figura 4.5: Sequência de instanciação dos plug-ins de aprendizagem.

Diferente da instanciação normal, na aprendizagem o método `openFile` funcionará somente para inicialização de atributos e retornará uma janela interna invisível vazia. A inicialização real ocorre somente ao ser chamado o `setUnBBayesFrame` (quando o módulo - janela interna oculta - é efetivamente adicionado à tela), que engatilha *popups* (como o `JFileChooser` do *swing*, diálogo que permite com que o usuário selecione um arquivo) e delega posteriormente ao `unbbayes.learning.ConstructionController`, que é tratado então como uma caixa preta. Esse tipo de sequência (retornar uma janela invisível inicialmente e engatilhar o processo somente depois do método `setUnBBayesFrame`) permite simular assistentes (*wizards*) em plug-ins de módulos no UnBBayes.

4.4 Plug-ins de Metáfora

A metáfora é uma *branch* do projeto UnBBayes que visa prover uma interface voltada para o usuário comum. Esconde as informações desnecessárias para pessoas que não vão modelar domínios, acessando somente domínios previamente modelados por outros. É basicamente composta por uma tela de entrada de evidências e leitura de resultado da inferência (Figura 4.6).

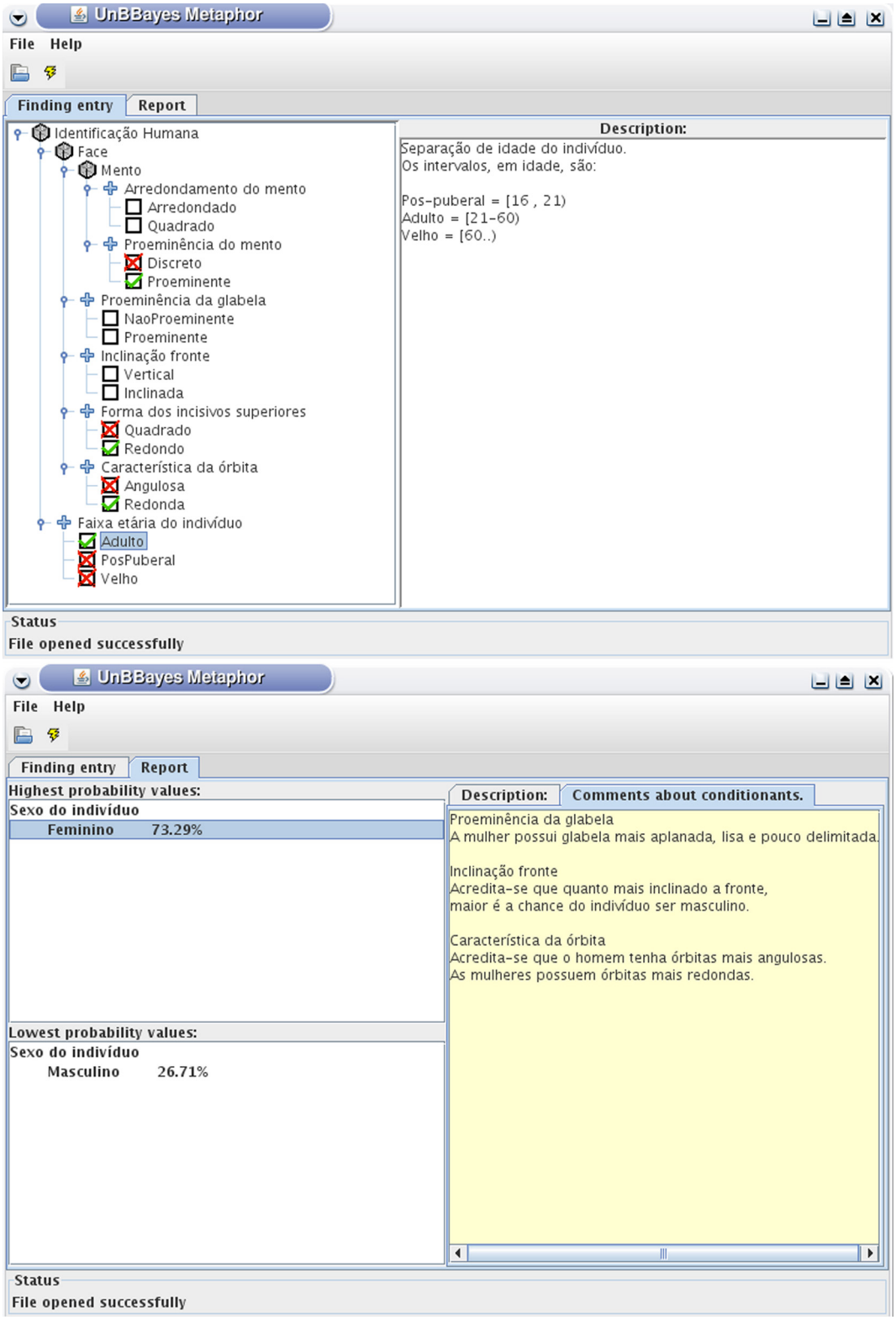

Figura 4.6: Entrada de evidências pela metáfora (acima) e laudo (abaixo).

No UnBBayes, estavam disponíveis duas variações da Metáfora: Metáfora Médica (especializada para o domínio de diagnóstico médico) e Metáfora de Identificação Humana (especializada para o domínio de identificação humana). Como ambas eram arquiteturalmente idênticas, somente a metáfora de identificação humana é apresentada nesta seção.

A arquitetura da metáfora antes da refatoração está ilustrada na Figura 4.7. Como é basicamente uma interface gráfica adicional para as funcionalidades já oferecidas pelo UnBBayes, não provê muitos serviços especiais. Segue também uma breve descrição das classes existentes.

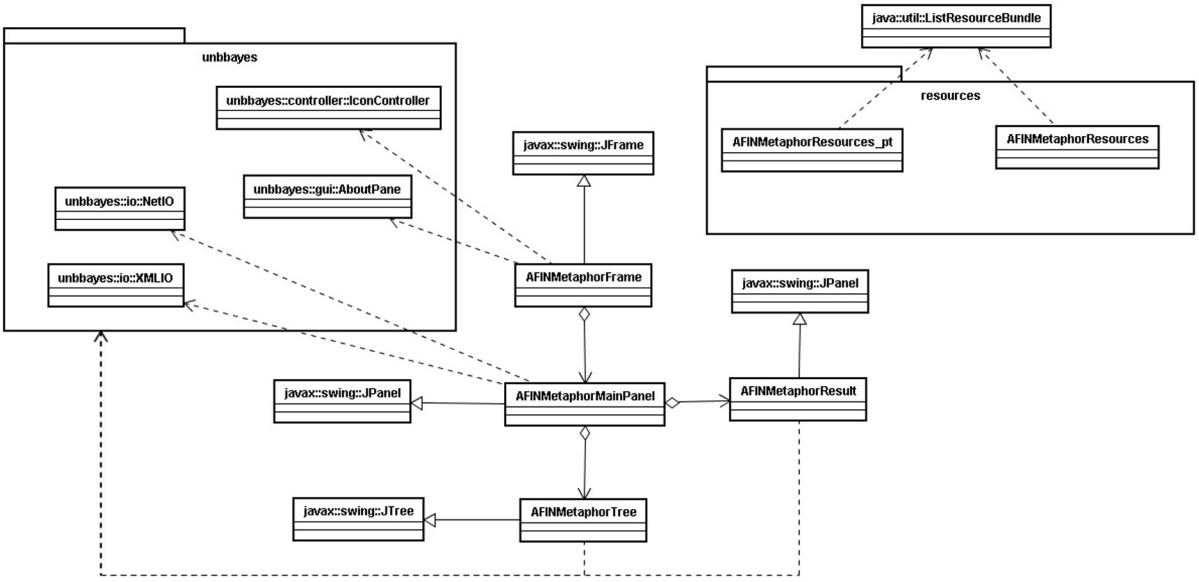

Figura 4.7: Diagrama de classes da metáfora.

`AFINMetaphorFrame` : `JFrame` principal do programa. Contém a barra de menu e a barra de *status*.

`AFINMetaphorMainPanel` : Integra os painéis de entrada de evidência e o painel de laudo em abas diferentes.

`AFINMetaphorTree` : Representa a árvore para entrada de evidências.

`AFINMetaphorResult` : Representa o painel para verificação de resultados (laudo) da inferência bayesiana.

Como pode ser visto na Figura 4.7, a metáfora era uma janela independente (*i.e.* estende uma `JFrame` do *swing*). Para tornar a metáfora como um novo plug-in, seria necessário o converter para uma janela interna. Isso pôde ser facilmente realizado criando-se uma classe `Afin` (que estende o `UnBBayesModule` - uma janela interna representando um plug-in de módulo) e incluindo nessa classe o `AFINMetaphorMainPanel`. Com isso, pôde-se converter a metáfora sem alteração alguma de código fonte da original. A Figura 4.8 ilustra a nova arquitetura: um plug-in que estende o ponto de extensão *Module*.

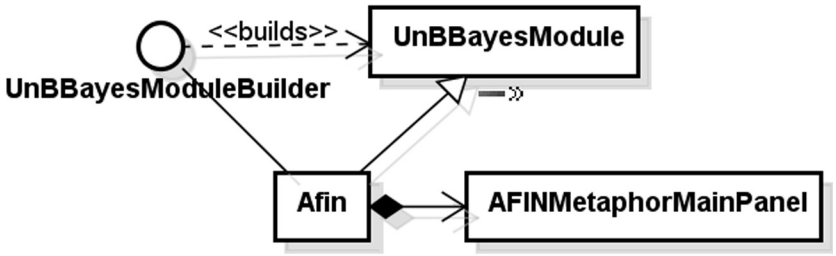

Figura 4.8: Diagrama de classes do plug-in da metáfora. As outras classes associadas a AFINMetaphorMainPanel não sofreram alterações.

4.5 Plug-ins para Amostragem e Algoritmo de Inferência

As *features* de amostragem permitem que seja gerado um arquivo de texto com conjuntos de dados que reflitam uma dada BN fornecida. Os dados gerados por estas *features* são no formato esperado pelas *features* de aprendizagem apresentadas na Seção 4.3. O UnBBayes, antes da refatoração, oferecia três algoritmos para esse fim: *montecarlo, Gibbs* e *likelihood weighting*. Dentre eles, *Gibbs* e *likelihood weighting* também são reutilizados para se implementar algoritmos de inferência bayesiana (funcionalidade ilustrada previamente na Figura 3.19).

A classe principal que inicializava as *features* de amostragem era a `MCMainController` do pacote `unbbayes.simulation.montecarlo.controller`. As rotinas de *monte carlo, Gibbs* e *likelihood weighting* eram chamadas respectivamente passando como parâmetro objetos de `unbbayes.simulation.montecarlo.sampling.MatrixMonteCarloSampling`, `unbbayes.simulation.sampling.GibbsSampling` ou `unbbayes.simulation.likelihoodweighting.sampling.LikelihoodWeightingSampling`.

De maneira similar aos algoritmos de aprendizagem bayesiana (vide Seção 4.3), as *features* de amostragem se manifestavam como assistentes (*wizards*) e foi utilizado uma janela interna vazia oculta para representar o módulo de amostragem (cujas funcionalidades são engatilhadas quando o método `setUnBBayesFrame` é chamado e consequentemente `MCMainController` é instanciado com os respectivos parâmetros). Portanto, o plug-in de amostragem também estede o ponto de extensão `Module`.

A Figura 4.9 ilustra as novas classes. A sequência de instanciação de `MCMainController` segue o mesmo padrão da Figura 4.5, trocando-se `ConstructionController` por `MCMainController`. Como as classes de amostragem de *Gibbs* e *likelihood weighting* dependiam de *monte carlo*, os plug-ins resultantes devem manter essa dependência em nível de plug-in.

A Figura 4.10 ilustra as classes dos plug-ins implementando algoritmos de inferência bayesiana, seguindo as especificações apresentadas na Seção 3.4.4.

4.6 Plug-in de OOBN

As BNs são ferramentas poderosas para representação de conhecimento e raciocínio em domínios com incerteza. Entretanto, em domínios grandes e complexos, elas mostram duas de-

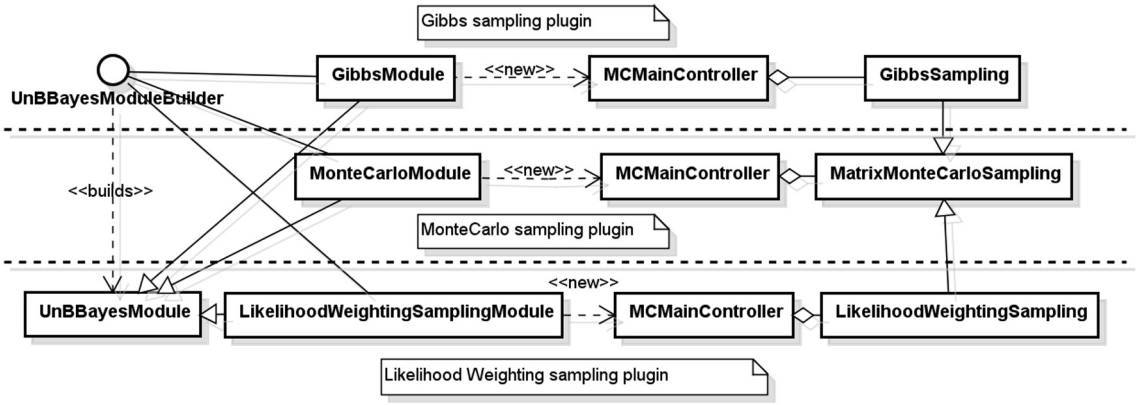

Figura 4.9: Diagrama de classes dos plug-ins de amostragem.

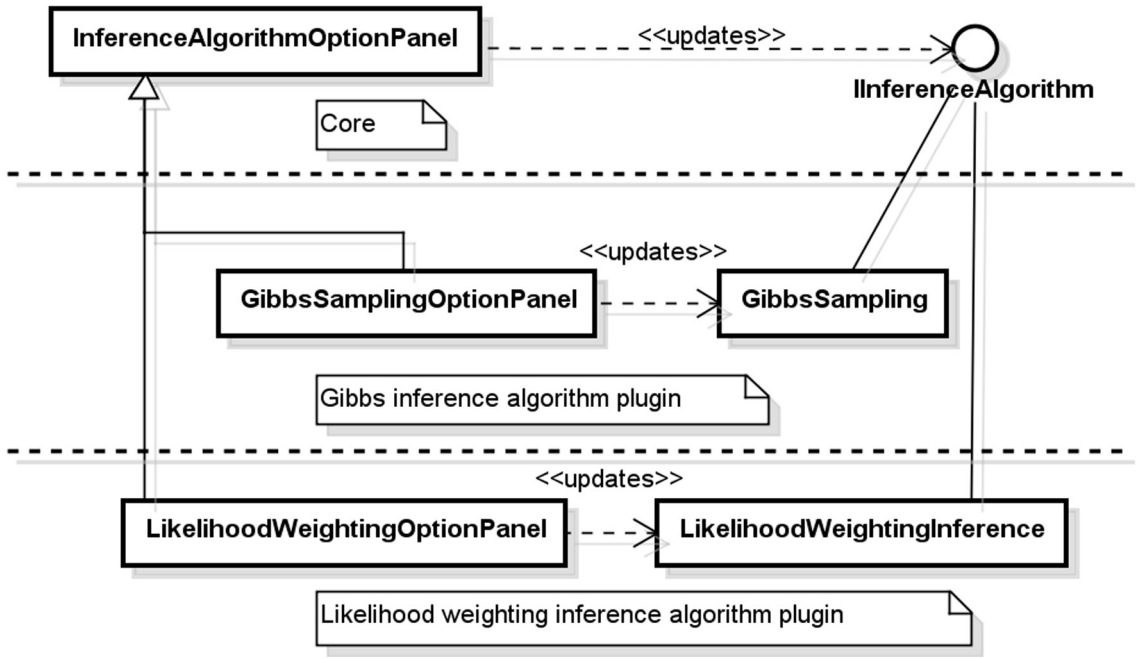

Figura 4.10: Diagrama de classes dos plug-ins de algoritmo de inferência.

ficiências: dificuldade de entendimento pelo decisor e/ou modelador, por se tornar uma enorme rede monolítica, e inexistência de um mecanismo eficiente para modelagem e inferência sobre estruturas repetitivas.

Visando uma modelagem eficiente, Koller propôs as OOBNs (61) . Em OOBN, um domínio é modelado como um conjunto de objetos inter-relacionados, com cada objeto representando um sub-domínio reutilizável. Cada sub-domínio é modelado como uma classe, que pode ser entendida como um fragmento de BN. Como os objetos somente associam-se por variáveis aleatórias de interface, os detalhes podem ser abstraídos, tornando-as um formalismo ideal para construção top-down de modelos probabilísticos para domínios grandes e complexos.

A Figura 4.11 ilustra a arquitetura da GUI do UnBBayes com o suporte a OOBN implementado, ilustrando suas relações com classes já existentes previamente no UnBBayes e suas interações com classes controladoras. O diagrama está simplificado: não são mostradas todas as relações de dependência. Somente estão mostradas as classes principais ao entendimento.

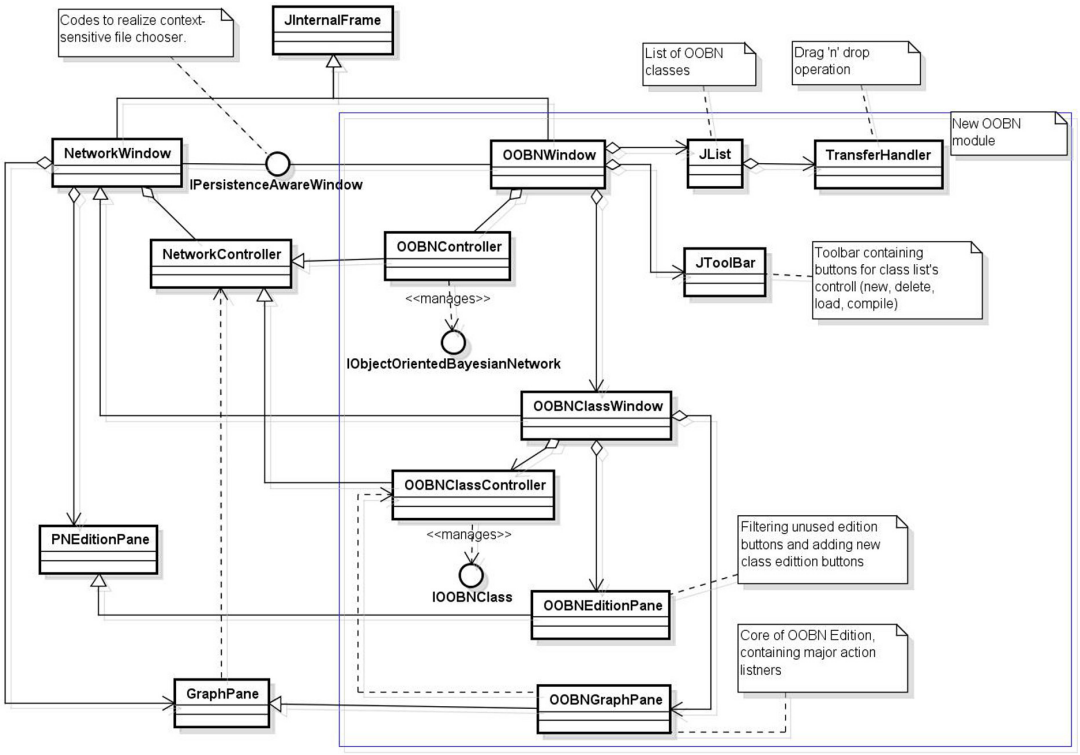

Figura 4.11: Classes de GUI implementando a OOBN antes da refatoração.

- `OOBNWindow`: container geral para a edição de OOBN (conjunto de classes). Contém internamente um painel de edição do conteúdo das classes também. É responsável por gerenciar visualmente uma lista de classes OOBN carregadas no projeto.
- `OOBNController`: classe responsável por rotinas de controle e interação entre classes de apresentação e as demais classes. Em particular, está responsável por delegar requisições da `OOBNWindow` à classes afora, como classes de persistência, classes de gerenciamento de classes OOBN, etc.

- `OOBNClassWindow`: classe que representa um painel (janela interna) à `OOBNWindow` para edição do conteúdo da classe atualmente selecionada para edição. Por estender `NetworkWindow`, funciona exatamente como uma janela de edição de BN convencional.
- `OOBNClassController`: controladora das classes da OOBN. Enquanto que `OOBNController` é uma controladora de um conjunto de classes OOBN, a `OOBNClassController` é responsável pelo controle do conteúdo de uma única classe OOBN.
- `OOBNEditionPane`: painel que contém uma barra de ferramentas com botões para edição do grafo contido no `OOBNGraphPane`. A maior parte de sua rotina já está implementada pela sua superclasse `PNEditionPane`. Esta classe basicamente adapta a `PNEditionPane` para ocultar botões de edição de BN convencional, mas não usadas na edição de OOBN.
- `OOBNGraphPane`: painel para edição do grafo que representa uma classe OOBN. Estende `GraphPane` somente para possibilitar que o tipo (*input, output, private*) de um nó OOBN possa ser alterado.

A compilação de uma OOBN no UnBBayes consiste na geração de uma MSBN (vide seção 4.7), e isso tinha sido implementado por simples repasse de controle a classes de MSBN do UnBBayes (por isso, o UnBBayes com OOBN fora implementado "reescrevendo-se" o UnBBayes com MSBN). Como a arquitetura geral de OOBN já estava relativamente modular, foi possível adotar uma abordagem de caixa preta para a refatoração do da *feature* de OOBN como um plug-in. Como a classe `OOBNWindow` representava uma janela interna (estendia `JInternalFrame`), bastou simplesmente que `OOBNWindow` passe a estender `UnBBayesModule`. Como `UnBBayesModule` já estende `JInternalFrame`, felizmente nenhuma classe da *feature* de OOBN foi impactada com essa alteração. Como `OOBNWindow` não oferecia um construtor sem parâmetros[40], foi utilizado um *builder* para sua construção como plug-in. A única alteração em nível de código fonte ocorreu em `OOBNWindow`, e o resto foi tratado como uma caixa preta. O resultado está esquematizado na Figura 4.12.

Por implementar um novo formalismo, naturalmente o ponto de extensão mais apropriado para o plug-in de OOBN é o `Module`. Pode-se ver que há uma dependência ao plug-in de MSBN (vide *tag* `requires`). Comparada a arquitetura anterior, em que se necessitava "reescrever" uma configuração antiga para se criar uma nova configuração, o plug-in de OOBN não precisou mais reescrever códigos da MSBN. Bastou agora declarar dependência ao plug-in de MSBN, pois a compilação da OOBN realizava repasse de controle a esse módulo.

4.7 Plug-in de MSBN

Como comentado na Seção 3.2.4, uma MSBN divide uma BN em sub-redes, de forma em que inferências possam ser limitadas nessa sub-rede (para diminuir a carga de computação). São úteis para representar BNs grandes. Da mesma forma que na OOBN (Seção 4.6), esta *feature* já estava relativamente modular, em uma arquitetura bem similar ao da OOBN. Por esse motivo, a mesma técnica de refatoração pôde ser utilizada. A Figura 4.13 ilustra o ponto principal da refatoração. Naturalmente, o ponto de extensão implementado é o `Module`.

[40] A infraestrutura de plug-ins do UnBBayes exige que a classe especificada em plugin.xml tenha construtor padrão sem parâmetros.

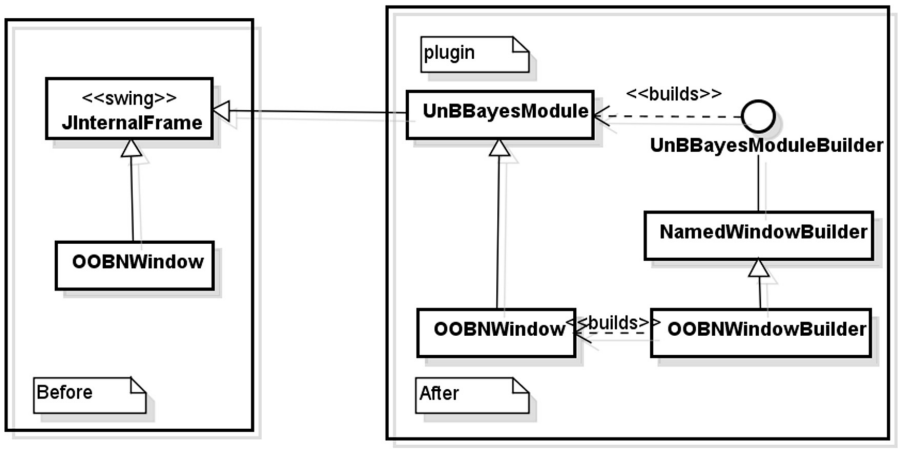

Figura 4.12: Refatoração de OOBN como plug-in. O lado esquerdo indica a arquitetura antes da refatoração. O lado direito ilustra a arquitetura depois da refatoração.

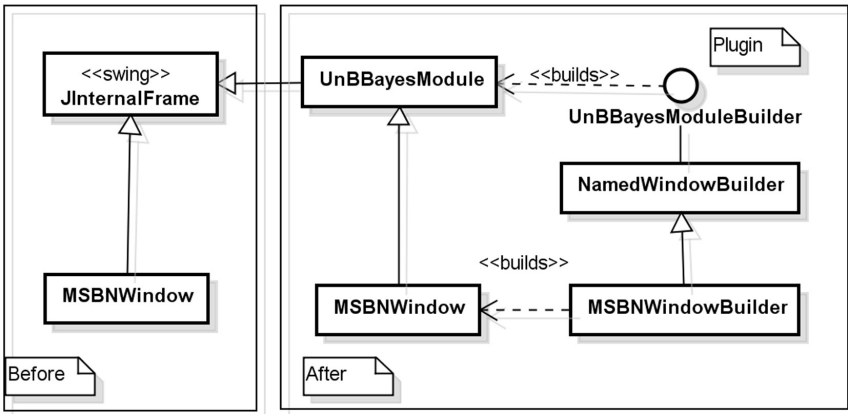

Figura 4.13: Refatoração de MSBN como plug-in.

4.8 Plug-in de PRM

O plug-in de PRM foi uma funcionalidade criada pelo presente mestrando após a refatoração do UnBBayes, como um requisito parcial para conclusão da disciplina IA 1, mestrado da UnB. Portanto, merece atenção[41] por não fazer parte do grupo de funcionalidades antigas "refatoradas" à nova arquitetura. É o primeiro plug-in "novo" criado após a refatoração da arquitetura do UnBBayes. A Figura 4.14 mostra a captura de tela dessa funcionalidade.

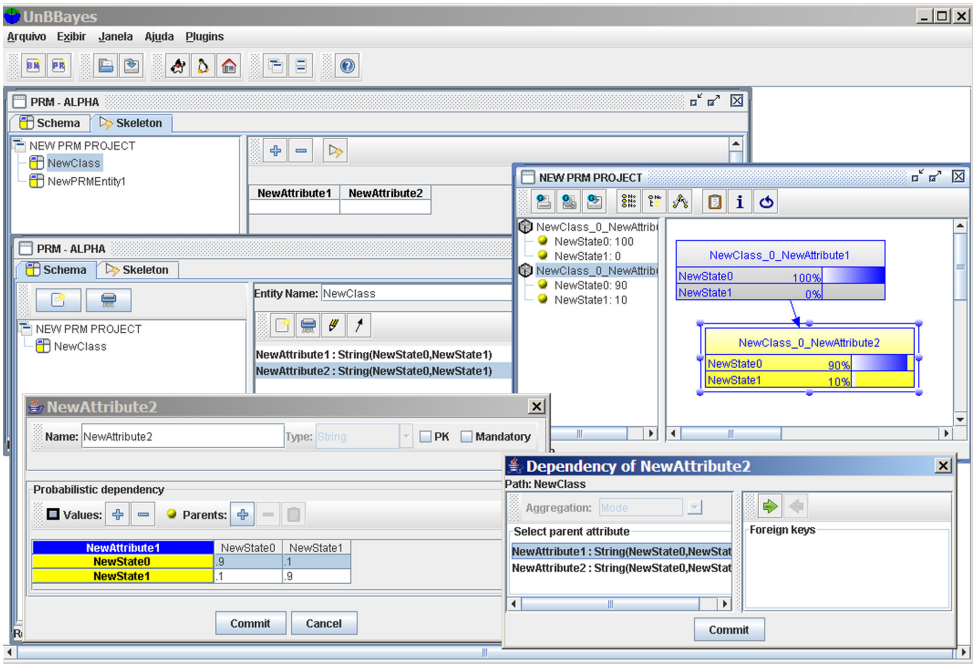

Figura 4.14: Captura de tela do plug-in de PRM.

Em PRM, dados são organizados como entidades e relacionamentos (E/R); cujas entidades podem ser entendidas como um composto de atributos, e os relacionamentos como uma associação entre atributos, geralmente interligando diferentes entidades. Neste modelo, atributos que "hospedam" valores incertos são modelados como variáveis aleatórias e são tratadas por uma lógica probabilística (*i.e.* BN). Por tal motivo, define-se PRM como uma abordagem intercessora entre a lógica bayesiana e a lógica relacional. Nota-se que banco de dados E/R são massivamente utilizados no mercado (*e.g.* banco de dados SQL), tornando o PRM um candidato promissor de retorno imediato.

As informações em PRM são estruturadas ou em *schema* (analogia a tabelas em bancos de dados relacionais) ou *scheleton* (analogia a dados em bancos de dados relacionais), cujos atributos de *schema* (*i.e.* colunas das tabelas) podem possuir dependências probabilísticas especificadas como uma "estrutura de dependências PRM", com distribuições de probabilidade condicional associadas. A Figura 4.15 mostra as interfaces mais básicas que implementam a lógica de negócio, e a A Figura 4.16 ilustra as classes da GUI. Observa-se que as classes `PRMWindow` e `PRMWindowBuilder` são as que se aderem à infra-estrutura de plug-ins.

[41] No entanto, por não fazer parte dos critérios de conclusão de pesquisa de mestrado, o PRM não é detalhado nessa dissertação. Somente as informações do plug-in de PRM relacionadas à infra-estrutura geral de plug-ins estão apresentadas.

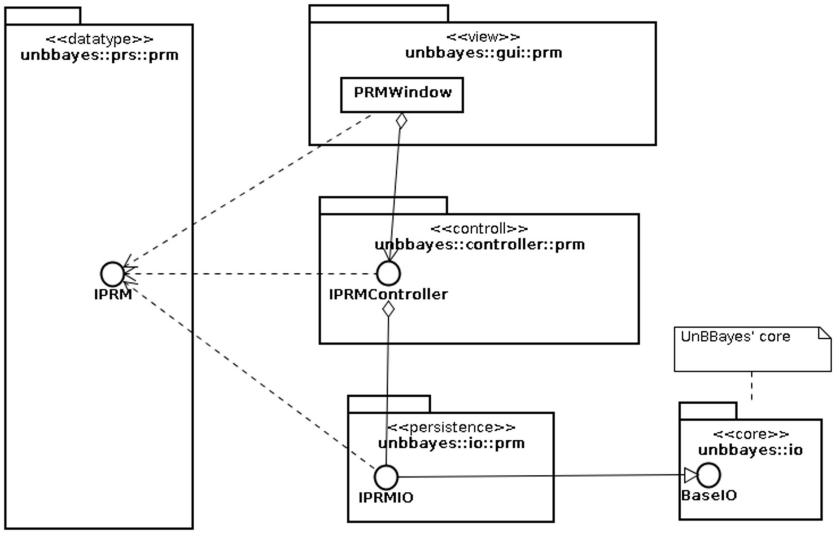

Figura 4.15: Arquitetura de camadas do plug-in de PRM.

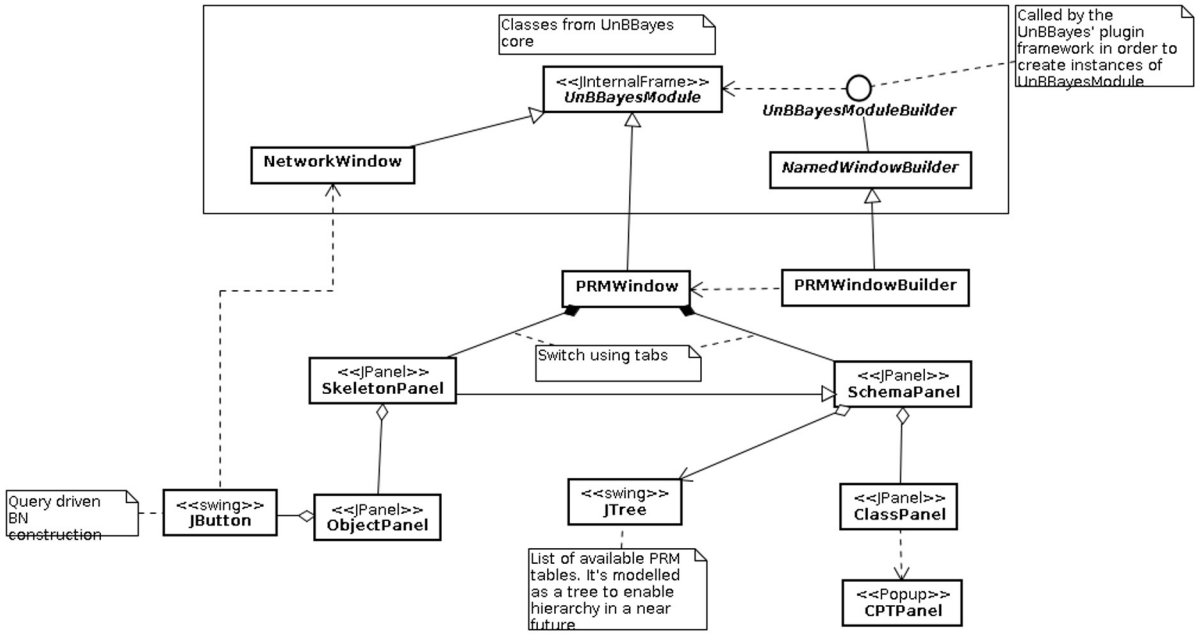

Figura 4.16: Diagrama de classes da GUI do PRM.

Capítulo 5

UnBBayes-MEBN e Plug-in para Suporte a PR-OWL 2

As funcionalidades para se realizar a linguagem PR-OWL 2 no UnBBayes são implementadas basicamente reusando-se as funcionalidades previamente criadas no UnBBayes-MEBN. Como o UnBBayes-MEBN foi também refatorado como um plug-in de módulo do *core*, naturalmente o plug-in de PR-OWL 2 foi projetado como um plug-ins do UnBBayes-MEBN. Ou seja, aproveitando-se o do mecanismo de dependências entre plug-ins, foi-se criado um plug-in para plug-in.

Este capítulo descreve as alterações realizadas no UnBBayes-MEBN para se tornar um plug-in de módulo do UnBBayes. Novos pontos de extensão também foram criados durante a refatoração. A descrição do conjunto de classes que compõem o plug-in para suporte a PR-OWL 2 no UnBBayes também é oferecida neste capítulo.

A Seção 5.1 apresenta o estado anterior do UnBBayes-MEBN, seguido pela Seção 5.2, que descreve as alterações feitas ao UnBBayes-MEBN para se aderir a nova arquitetura de plug-ins. A Seção 5.3 descreve a arquitetura dos componentes criados para a implementação da especificação PR-OWL 2 no UnBBayes e finalmente a Seção 5.4 apresenta algumas observações de performance que foram obtidas com o uso da ferramenta *Test and Performance Tools Platform* (TPTP) do Eclipse.

5.1 UnBBayes-MEBN Antes da Refatoração

O UnBBayes é o primeiro software a oferecer suporte a edição de MEBN e de arquivos PR-OWL de forma gráfica, e é um ferramental que facilita a modelagem ao omitir muitos termos técnicos cujo interesse está vinculado apenas ao processamento interno durante as fases de consistência e inferência da ontologia, os preenchendo automaticamente. Foi modelado desde o início pensando-se no armazenamento dos modelos editados como uma ontologia PR-OWL.

Segue a descrição das principais classes construídas na extensão inicial do UnBBayes para suporte à MEBN. A Seção **MTheory** seguinte apresentará as principais classes utilizadas para a representação de uma MTheory. Na Seção **MFrag**, serão detalhadas as estruturas de um MFrag e os relacionamentos dos nós. As classes apresentadas nessas seções foram utilizadas sem grandes alterações no plug-in de PR-OWL 2 também.

5.1.1 MTheory

O diagrama de classes em UML da Figura 5.1 representa a estrutura geral de uma MTeoria tratada pelo UnBBayes-MEBN.

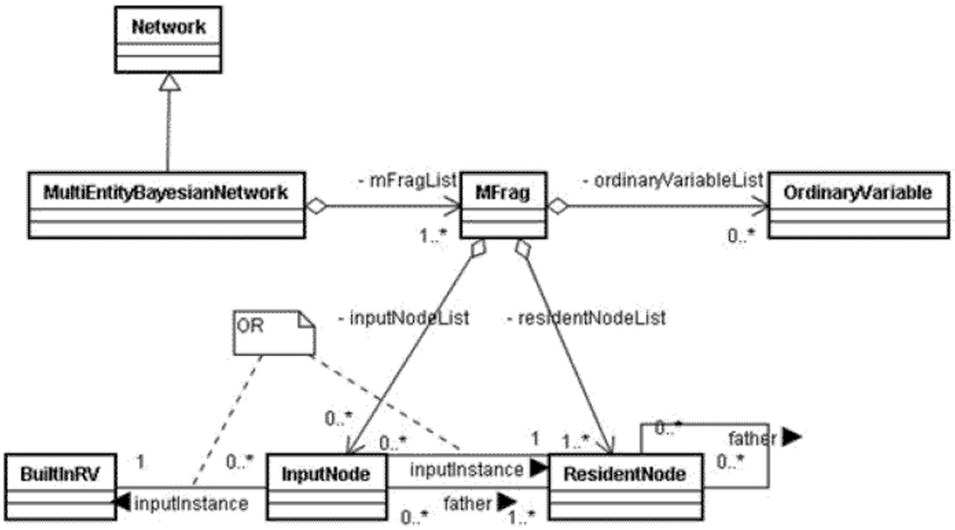

Figura 5.1: Modelagem de classes para MTheory

Classe `MultiEntityBayesianNetwork`: representa uma MTheory, a qual define a ontologia do domínio. A teoria é composta por um ou mais MFrags.

Classe `MFrag`: representa um MFrag. MFrag é composta virtualmente por variáveis ordinárias, nós de entrada, nós de contexto e nós residentes.

Classe `OrdinaryVariable`: variáveis ordinárias são utilizadas para se referir a entidades que entrarão como argumentos nas variáveis aleatórias (*Random Variable* - RV) de MFrags. Cada argumento de uma RV em MFrag exige um objeto da classe `OrdinaryVariable`, sendo que este está associado com o tipo das entidades aceitas para preencher aquele parâmetro. Por questão de conveniência, o UnBBayes representa `OrdinaryVariable` como um nó de contexto *IsA(Type)*[42], especialização de `Node`. Possui escopo interno o MFrag.

Classe `InputNode`: classe que representa uma RV de entrada. É basicamente uma referência para um `ResidentNode`, que é usado como entrada em um MFrag. Apesar de não estar mostrada no diagrama UML, utiliza uma classe `ResidentNodePointer` para desacoplar com `ResidentNode`.

Classe `ResidentNode`: classe que representa uma RV residente. Possui informações sobre seus argumentos (`OrdinaryVariable`) e a declaração do pseudo-código que define uma CPT. Possui escopo global à MTheory[43].

[42] Já que no UnBBayes todas as entidades devem ser tipadas, e essa associação de tipos deve ser declarada como um nó de contexto IsA, resolveu-se juntar esses dois conceitos em um só.

[43] Apesar de não ser mostrada no diagrama UML, a MultiEntityBayesianNetwork referencia os conjuntos de ResidentNode que existem na MTheory.

Classe `BuiltInRV`: classe que representa variáveis aleatórias nativas ao UnBBayes[44], previamente construídas para facilitar a modelagem de MEBN. Permitem representar uma família de distribuições de probabilidade sobre interpretações da FOL. A Tabela 5.1 apresenta as `BuiltInRV` que estão implementadas no UnBBayes.

And	EqualTo	Exists	ForAll
Iff	Implies	Not	Or

Tabela 5.1: `BuiltInRV` implementadas no UnBBayes.

5.1.2 MFrag

O diagrama da Figura 5.2 ilustra os principais elementos de um MFrag. O UnBBayes somente lida com MFrags de domínio[45], que funcionam como um *template* para a geração de uma SSBN, baseado nas instâncias de entidades e evidências presentes na base de conhecimento que ilustra uma situação específica.

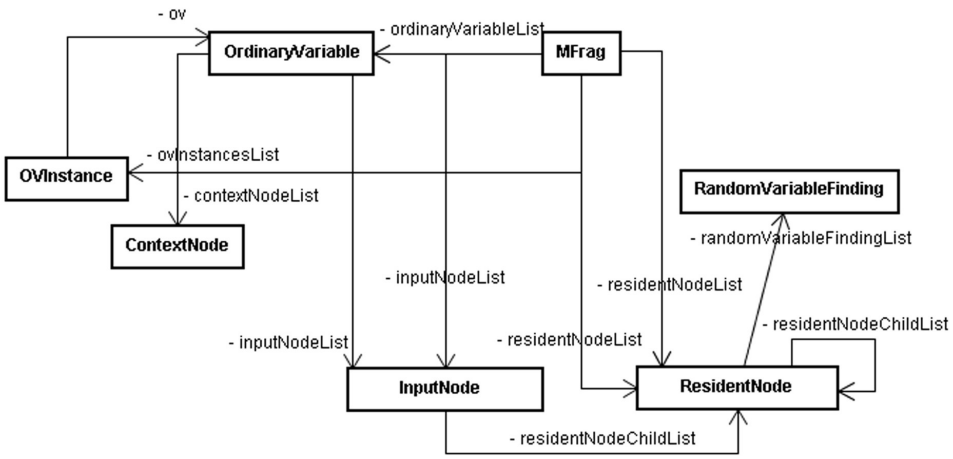

Figura 5.2: Modelagem de classes para MFrag.

Classe `MFrag`: representa um MFrag, utilizada como *template* para a instanciação da SSBN de acordo com as evidências, instâncias de entidades presentes e os questionamentos do usuário. Representa a modelagem do domínio (através de ontologia probabilística). É composta pelos nós de contexto (`ContextNode`), pelos nós de entrada (`InputNode`) e pelos nós residentes (`ResidentNode`). Para possibilitar a geração da SSBN, esse MFrag possui uma referência para as instâncias `OVInstance`[46].

[44] Conectivos lógicos, quantificadores e igualdade.

[45] MEBN possui dois tipo de MFrags: de domínio e de *findings*. Como o UnBBayes-MEBN representa findings como declarações na base de conhecimento, o UnBBayes-MEBN não oferece edição explícita de MFrags de *findings*.

[46] Composta por um par de uma variável ordinária e uma instância de entidade, representando um valor específico que OrdinaryVariable está assumindo em um determinado momento.

Classe `RandomVariableFinding`: representa as evidências existentes para um determinado nó. Essa informação é utilizada para a geração da SSBN.

A Figura 5.3 apresenta a hierarquia dos tipos de nós existentes na MEBN.

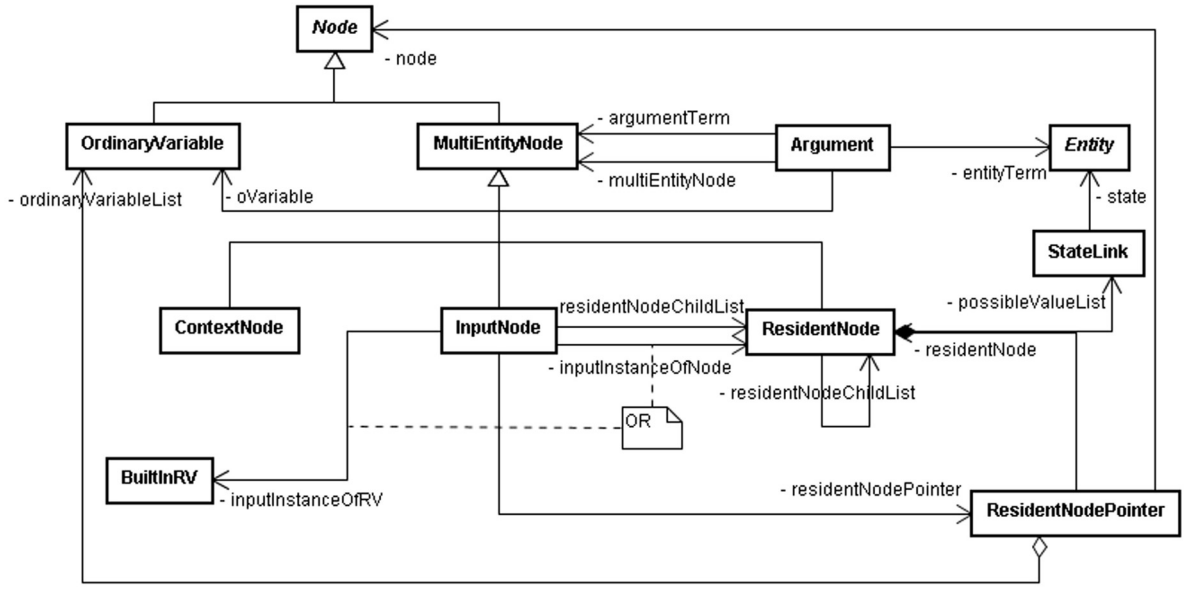

Figura 5.3: Modelagem de classes para os nós de uma MEBN.

Classe `MultiEntityNode` representa um nó de uma MEBN, podendo ser residente, de entrada ou de contexto, conforme sua subclasse.

Classe `Entity` representa uma entidade.

Classe `StateLink` representa uma vinculação entre o `DomainResidentNode` e `Entity`, possibilitando a definição de exclusividade global, explicado com mais detalhes na Seção 5.3.2.

Classe `Argument` representa um argumento de um determinado `MultiEntityNode`. Esta classe é utilizada apenas no processo de armazenamento e carregamento de arquivos PR-OWL. O argumento pode ser complexo, através do uso das propriedades `entityTerm` e `argumentTerm`, ou simples, através da propriedade `oVariable`.

Classe `ResidentNodePointer` representa uma ligação do nó de entrada ou do nó de contexto, através da propriedade `node`, ao nó residente correspondente, através da propriedade `residentNode`.

Classe `ContextNode` representa os nós de contexto, que fazem restrições às entidades utilizadas como argumentos nas RV de entrada e residentes do MFrag em questão.

Classe `InputNode` representa os nós de entrada. Estes nós são apenas uma referência para `DomainResidentNode` já existente, que geralmente se encontra em outro MFrag, mas em algumas situações, é definido no mesmo MFrag, para representar recursão, por exemplo.

Classe `ResidentNode` representa os nós residentes geradores. Estes nós possuem um pseudo-código, responsável por definir a regra de criação de uma CPT, independentemente do caso específico (entidades e evidências presentes). A Seção 5.1.3 apresenta mais detalhes a respeito do pseudo-código e geração de CPT. Os nós residentes podem ter como pais ou outros nós residentes[47] ou nós de entrada.

5.1.3 Tabela de Probabilidades

A proposta original da linguagem PR-OWL apresentada em (31) não contém uma especificação formal para fórmulas capazes de definir dinamicamente a CPT dos nós residentes instanciados durante a geração da SSBN. Essa funcionalidade seria essencial quando o número de pais é desconhecido. O UnBBayes-MEBN utiliza um compilador de pseudo-código próprio para a geração dinâmica de tabelas. A gramática formal, definida em notação BNF, é apresentada abaixo:

```
table := statement | if_statement
 if_statement  ::=
 "if" allop varsetname "have" "(" b_expression ")" statement
  "else" else_statement
 allop ::= "any" | "all"
 varsetname ::= ident ["." ident]*
 b_expression ::= b_term [ "|" b_term ]*
 b_term ::= not_factor [ "&" not_factor ]*
 not_factor ::= [ "~" ] b_factor
 b_factor ::= ident "=" ident
 else_statement ::= statement | if_statement
 statement ::= "[" assignment "]"
 assignment ::= ident "=" expression [ "," assignment ]*
 expression ::= term [ addop term ]*
 term ::= signed_factor [ mulop signed_factor ]*
 signed_factor ::= [ addop ] factor
 factor ::= number | function | "(" expression ")"
 function ::= possibleVal
  | "CARDINALITY" "(" varsetname ")"
  | "MIN" "(" expression ";" expression ")"
  | "MAX" "(" expression ";" expression ")"
 possibleVal ::= ident
 addop ::= "+" | "-"
 mulop ::= "*" | "/"
 ident ::= letter [ letter | digit ]*
 number ::= [digit]+
```

Internamente, um compilador traduz o pseudo-código para uma CPT de uma BN normal (a SSBN é tratada pelo UnBBayes-MEBN como uma BN normal). As expressões matemáticas e booleanas são resolvidas usando-se classes internas que representam uma árvore de avaliação de expressões, satisfazendo o *design-pattern Composite*. Esse tipo de abordagem é bastante comum em parsers orientado-a-objeto. O pseudo-código e seu mecanismo básico de compilação não sofreram grandes alterações na refatoração.

A gramática incorpora três funções para manipulação de expressões com valores diâmicos: CARDINALITY, MIN e MAX. Abaixo, sua semântica está descrita.

CARDINALITY Retorna o número de combinações de nós pais que possuem como argumento predominante a indicada por "varsetname". Se o valor "varsetname" for diferente

[47] Instâncias iguais de nós residentes - com argumentos iguais - não podem estar ligadas.

da indicada na expressão “if”, obviamente será retornado o valor 0. A função é útil para se ter uma idéia de quantos pais foram conectados àquele nó.

MIN Obtém duas expressões como argumento e retorna o menor valor entre o resultado dessas duas expressões. É útil para limitar superiormente uma expressão.

MAX Obtém duas expressões como argumento e retorna o maior valor entre o resultado dessas duas expressões. É útil para limitar inferiormente uma expressão.

Quando um pseudo-código não é declarado para um nó residente, será atribuído uma interpretação especial a esse nó. Nesse caso, a distribuição de probabilidades será uniforme para todos os valores possíveis daquele nó. Isso é particularmente útil quando o estado possível de um nó é uma entidade, cujo número de instâncias não é conhecido *a priori*, seja, não se conhece quais estados o nó pode apresentar. Essa convenção foi mantida na refatoração também.

5.1.4 GUI

A Figura 5.4 apresenta a arquitetura da GUI do UnBBayes-MEBN antes da refatoração, para ilustrar que painel de edição da MEBN (classe MEBNEditionPane) foi criado para a edição de MTheories e de seus elementos. Todos os outros elementos gráficos estão contidos dentro deste.

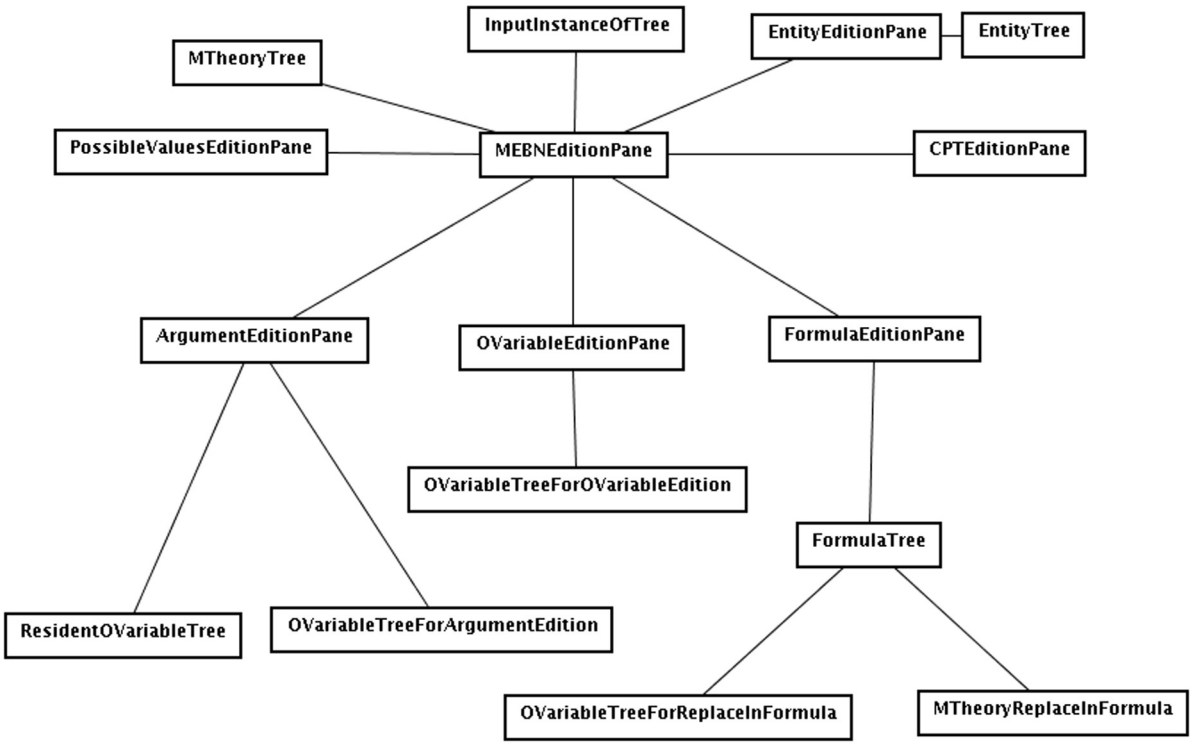

Figura 5.4: Arquitetura da GUI

5.1.5 Edição de MFrags e Conteúdos

A edição de MFrag consiste em adicionar, remover, visualizar e renomear MFrags. A seleção de MFrags é realizada através da árvore da MTheory, implementada pela classe MTheoryTree, que mostra os elementos da MTheory de forma hierárquica, apresentando os nós como filhos de seus respectivos MFrags. O MFrag ativo terá o seu conteúdo exibido no grafo em cena.

Similarmente, a edição de um nó residente permite adicionar, renomear e excluir nós (através do grafo e através da árvore da MTheory), editar os estados através do painel de edição de possíveis valores, editar os argumentos inserindo e retirando variáveis ordinárias através do painel de edição de argumentos ou editar a distribuição de probabilidades do nó.

A edição da tabela probabilística é feita através de uma *popup*, conforme ilustrada na Fig. 5.5. Neste painel, o usuário tem opções de auto-texto que o auxilia na edição do pseudo-código: além de botões para inserir as estruturas sintáticas, há listas para a seleção dos objetos utilizados no código (nós pais, estados e argumentos). O código da tabela é colorido, facilitando a visualização dos diversos *tokens*.

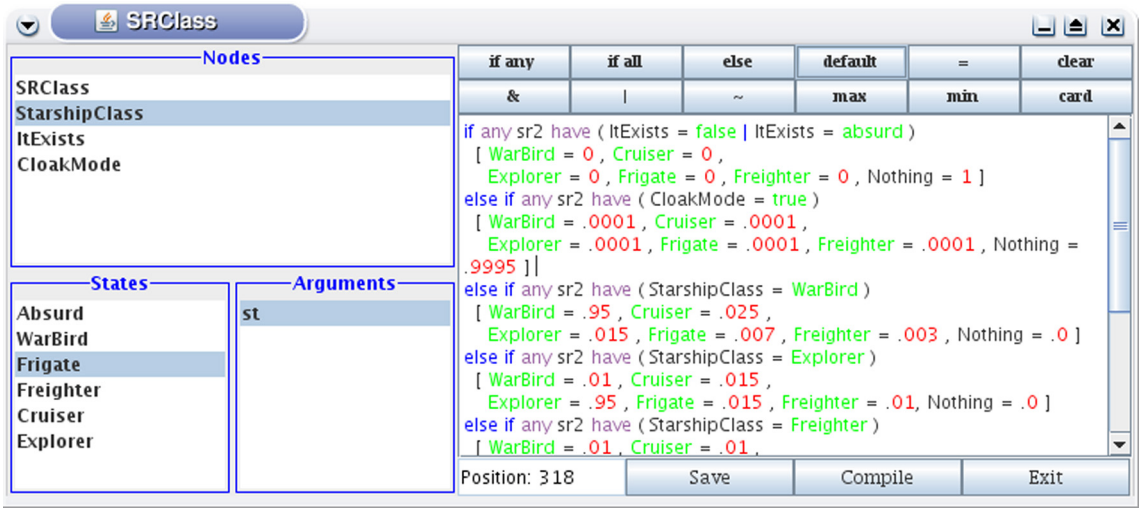

Figura 5.5: Janela de edição da CPT

5.1.6 Edição de Entidades e Evidências

A edição de entidades (Fig. 5.6) permite a criacão, eliminacão e alteração destas. Na edição, o usuário pode alterar o nome e definir a propriedade `É Ordenável`, que indica se as instâncias da entidade possuem ordem (podendo, portanto, serem utilizadas em construções que envolvam recursividade).

O usuário entra com as informações a respeito das situações específicas através do painel de instâncias de entidades e do painel de edição de evidências (Fig. 5.7). Estes painéis também mostram as informações contidas na base de conhecimento e permite que evidências ou instâncias retiradas sejam excluídas.

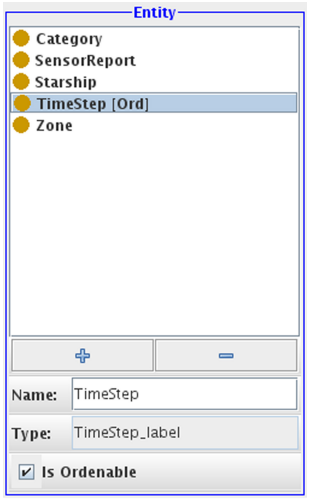

Figura 5.6: Edição de entidades

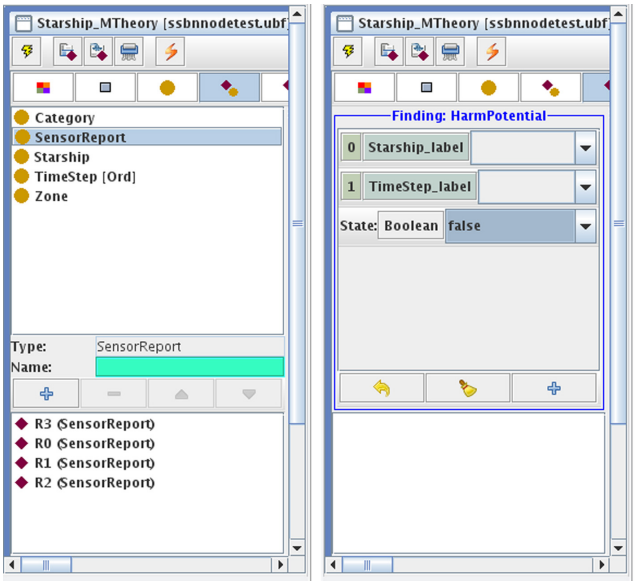

Figura 5.7: Edição de evidências

5.1.7 Geração de SSBN

A janela de questionamentos permite que o usuário selecione qual o nó (são listados todos os nós residentes) ele deseja realizar a inferência e sobre quais argumentos. Após a inferência, a SSBN gerada será mostrada no Painel de rede compilada (Fig 5.8), painel previamente implementada no UnBBayes. Através dele, o usuário poderá visualizar a probabilidade de cada estado, além de atribuir outras evidências e propagá-las (na SSBN gerada).

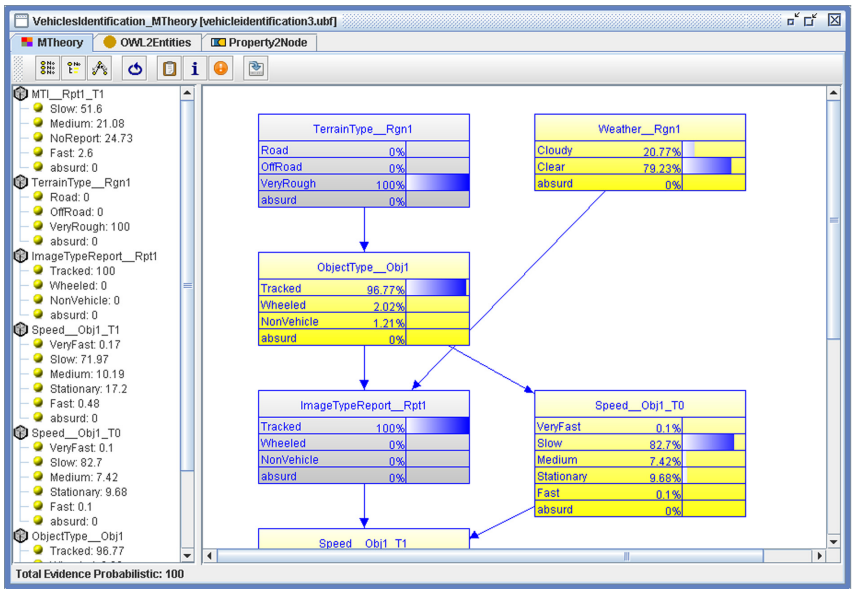

Figura 5.8: Painel da rede compilada

5.1.8 FOL no PowerLoom

Os nós de contexto (vide exemplos na Figura 5.9) são variáveis booleanas que representam condições que devem ser satisfeitas para que a distribuição de probabilidade dos nós residentes se aplique. Avaliar um nó de contexto corresponde a avaliar uma fórmula da FOL, tendo como premissas os fatos conhecidos. Na implementação do UnBBayes, antes da refatoração, foi adotada a API PowerLoom para avaliação de fórmulas da FOL.

O PowerLoom é um sistema de representação de conhecimento e inferência baseado na FOL (*First-Order Logic* - FOL) que provê linguagem e ambiente para a construção de aplicações baseadas em conhecimento. Foi criado por desenvolvedores da *University of Southern California*, a partir do sistema Loom, sob a liderança de Hans Chalupsky. Atualmente é distribuído em licença *open-source* (o usuário pode escolher entre as licenças GPL, LGPL ou Mozilla).

O PowerLoom permite modelar aspectos salientes de um mundo de interesse e fazer inferências a partir do modelo. Entre os diversos paradigmas de modelos para a implementação destes sistemas, os baseados em lógica são os mais utilizados por apresentarem vantagens tais como: a grande quantidade de pesquisas realizadas, sintaxes e semânticas bem conhecidas e ao grande poder de representação. Ele utiliza como linguagem de representação uma variante, com expressividade total, do KIF (*Knowledge Interchange Format*). O KIF é uma linguagem

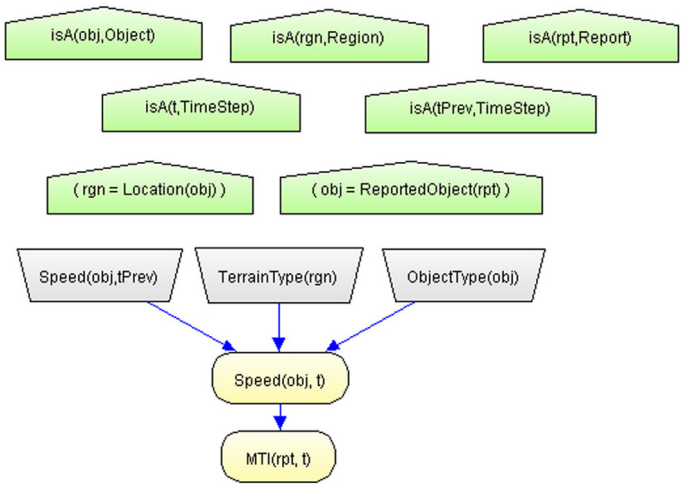

Figura 5.9: MFrag `Speed`, com seus nós de contexto em formato de pentágono.

baseada em lógica, desenvolvida para a troca de conhecimento entre sistemas de computadores diferentes.

No UnBBayes-MEBN (antes da refatoração), o interesse no PowerLoom esteve no seu mecanismo de inferência sobre sentenças de FOL, ficando em segundo plano as suas características como sistema de representação de conhecimento. Uma base de conhecimento PowerLoom é construída com a definição da terminologia para o domínio e pelas regras e fatos sobre este. Como os fatos podem ser inseridos e retirados, as respostas para os questionamentos podem mudar no tempo.

As entidades do mundo são capturadas como termos (ex.: `Brasília`, `3`, `Pessoa`). Os termos no PowerLoom devem ter nomes distintos. São categorizados ou relacionados com outros por objetos chamados relacionamentos (ex.: `temIdade`, `maiorQue`). Conceitos como `Pessoa`, `Estado`, `Compania` e `Número` são considerados uma subcategoria de relacionamento. Uma proposição é uma sentença lógica que tem um valor verdade associado, como por exemplo: "Benjamin é uma pessoa". A notação KIF seguida pelo PowerLoom utiliza a forma pré-fixada para representar as preposições, como ilustrado no exemplo a seguir, utilizando os relacionamentos `Pessoa`, `Casados`, `+` e `=`:

- `(Pessoa Joao)`
- `(Casados Joao Maria)`
- `(= (+ 2 3) 6)`

O cálculo predicativo constrói sentenças complexas a partir de sentenças simples usando os conectivos lógicos `and`, `or`, `not`, as implicações `<=`, `=>`, `<=>` e os quantificadores `forall` e exists. Exemplos:

- `(not (Homem Maria))`
- `(forall ?p (=>(Pessoa ?p) (exists ?m (Mae ?m ?p))))`

A primeira sentença representa o fato "Maria não é homem", enquanto segunda, mais complexa, representa o fato "Toda pessoa tem uma mãe". Para fazer inferência a partir das sentenças

e regras da base de conhecimento, podem ser utilizados os comandos `ask` e `retrieve`. O comando `ask` retorna o valor verdade de uma assertiva, enquanto o comando `retrieve` retorna uma listagem dos objetos que corretamente preenchem as variáveis. No exemplo abaixo, o `ask` retornará `false`, enquanto o `retrieve` retornará `?x = Maria`. Note que nesse modelo, uma consulta a diversas variáveis retornará tuplas que satisfazem condições, o que difere de *reasoners* (máquinas de inferência) em DL, cuja consulta é para uma variável somente.

O PowerLoom, como em OWL, assume premissas de mundo aberto por padrão. Em uma visão de mundo aberto é assumido que não há um modelo completo do mundo tratado: caso não haja informações para provar a verdade ou a falsidade de um questionamento, o valor será dado como "desconhecido". No entanto, a visão de premissa de mundo fechado também é implementada no PowerLoom, permitindo que se opte por um dos dois modos.

5.1.9 Avaliação dos Nós de Contexto

O processo descrito nesta seção é comum tanto para KB em PowerLoom quanto para KB em reasoners OWL. Os passos gerais para a avaliação dos nós de contexto são os seguintes:

- mapeamento da MTheory geradora na base de conhecimento;
- mapeamento da situação específica para a base de conhecimento;
- avaliação dos nós de contexto.

O mapeamento da MTheory geradora para a KB corresponde a inserir na base de conhecimento as informações de uma determinada situação da MTheory. Essa inserção na KB é precedida de uma tradução da modelagem em definições de entidades e de variáveis aleatórias.

O UnBBayes implementa uma MTheory tipada, de forma que estes tipos devem ser mapeados na KB. Ao se utilizar a KB, no entanto, o UnBBayes está assumindo que já fez todas as verificações quanto a corretude dos tipos definidos na MTheory, informando ao usuário possíveis erros. Portanto, **a arquitetura geral de KB não assume que a checagem de tipos seja feita pela KB**. Isso é particularmente problemático quando tal verificação, fora da KB, é custosa (e portanto, valendo mais a pena repassar o teste de tipagem à KB).

Com a avaliação de um nó de contexto, as variáveis ordinárias da fórmula deverão estar corretamente substituídas pelos valores reais da situação específica. Não pode haver variáveis ordinárias não substituídas, a não ser aquela que se deseja obter como resposta. Novamente cabe ao UnBBayes fazer estas verificações, passando para a KB as fórmulas corretas.

A avaliação dos nós de contexto de um MFrag é feita em cadeia até que a primeira avaliação retorne falso. Caso isto ocorra, será utilizada a distribuição padrão. Caso contrário, as condições estão corretamente verificadas e as variáveis aleatórias do MFrag serão instanciadas utilizando-se as suas distribuições locais.

5.1.10 Arquivos UBF

O arquivo UBF foi elaborado como um arquivo de projeto do UnBBayes, armazenando informações específicas utilizadas pelo UnBBayes para a representação da MEBN. Discutiu-se se estas informações específicas deveriam ser acopladas a uma nova versão da PR-OWL, porém

chegou-se a conclusão que a PR-OWL não deveria conter informações específicas de softwares utilizados para a sua edição/visualização, pois estas não fazem parte da ontologia.

O UBF é um arquivo texto com extensão ".ubf" contendo uma lista de declarações no formato: `ATRIBUTO=VALOR1,VALOR2,...`, onde `ATRIBUTO` é um nome interno para a informação utilizada pelo UnBBayes e `VALOR` é o seu valor. Quando múltiplos valores são declarados, eles são separados por vírgulas. Os comentários são iniciados com o símbolo `%` e se estendem até o final da linha.

O UBF está intimamente acoplado a um arquivo PR-OWL, pois suas declarações indicam parâmetros especiais dos componentes da MTheory, armazenados no arquivo PR-OWL. Para manter controle deste acoplamento, o arquivo UBF contém a referência explicita de à qual arquivo PR-OWL ele se refere.

Aquelas declarações no arquivo UBF que não forem identificadas na especificação ou que não tenham representantes no arquivo PR-OWL serão simplesmente ignoradas. Isso permite que uma distribuição do UnBBayes que leia uma versão antiga do UBF (ou PR-OWL) possa ler uma versão recente sem maiores danos. O formato UBF é ainda o formato utilizado pelo UnBBayes após a adoção de PR-OWL 2 e não sofreu alterações quanto sua especificação. Portanto, não é mostrado com detalhes neste documento.

5.1.11 Implementação de I/O no UnBBayes-MEBN

Para a implementação do suporte a IO de MEBN foi utilizado o modo de arquivos OWL do Protégé 3. Este modo é baseado na classe `JenaOWLModel`. O Protégé usa a API Jena para várias tarefas, em particular para fazer o *parsing* de arquivos OWL/RDF.

A Figura 5.10 mostra o diagrama das classes que compõem o suporte à abertura e armazenamento de MTheories.

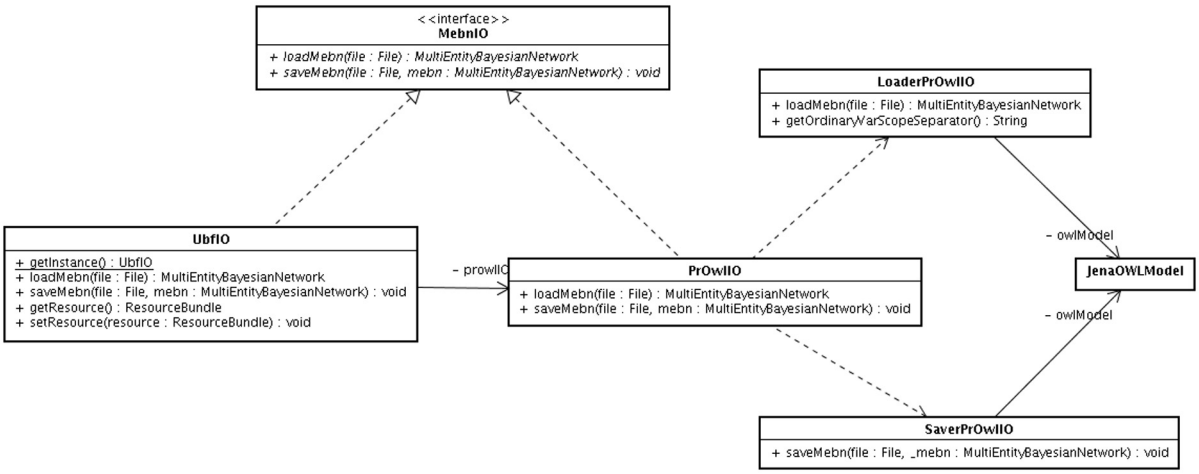

Figura 5.10: Modelagem de IO de MEBN

Pacote `unbbayes.io.mebn` possui as classes necessárias para abrir e salvar as MEBN nos formatos suportados, além de sub-pacotes contendo classes de exceção e classes *resource* para nacionalização.

Interface `MebnIO` interface para classes que implementam funcionalidades de entrada e saída de MEBN no UnBBayes. As classes que a implementam devem prover os seguintes métodos:

`loadMebn` reconstrói a MEBN a partir de um arquivo válido.

`saveMebn` salva uma MEBN em um arquivo válido.

Classe `PrOwlIO` classe que implementa a interface de `MebnIO` para abrir e salvar arquivos no formato PR-OWL. Para tal, utiliza as classes `LoaderPrOwlIO` e `SaverPrOwlIO`.

Classe `LoaderPrOwlIO` classe que concentra rotinas de carregamento da MTheory descrita em PR-OWL. Utiliza a API Protege-OWL para esse fim.

Classe `SaverPrOwlIO` classe que concentra rotinas de armazenamento da MTheory descrita em PR-OWL. Utiliza a API Protege-OWL para esse fim.

Classe `UbfIO` classe que implementa a interface `MebnIO`, adicionando funcionalidades de manipulação de arquivos UBF. É uma extensão da classe `PrOwlIO` através da agregação no lugar de heranças.

`getInstance` como a visibilidade do construtor da classe é privada, exige-se que se use este método de instanciação para a geração de objetos desta classe. Padrão de projeto *Singleton* foi utilizado.

`loadMebn` utiliza o método `loadMebn` da classe `PrOwlIO` para a leitura do arquivo PR-OWL e posteriormente realiza a leitura do arquivo UBF para o ajuste de parâmetros internos do UnBBayes.

`saveMebn` utiliza o método `saveMebn` da classe `PrOwlIO` para o armazenamento do arquivo PR-OWL e posteriormente realiza a escrita de parâmetros internos do UnBBayes no arquivo UBF.

A API do Protégé é utilizada no UnBBayes antigo apenas para fazer o carregamento e o salvamento das ontologias em PR-OWL. As outras funcionalidades da API não foram aproveitadas. Após ser feito o carregamento, a ontologia é armazenada no modelo de dados do UnBBayes. Para salvar, a ontologia é transferida do modelo de dados do UnBBayes para o modelo de dados do Protégé e então salva. Esta independência entre o UnBBayes e o Protégé possibilitou que novos formatos para a persistência de modelos MEBN sejam implementados sem grandes dificuldades.

5.2 Refatorando o UnBBayes-MEBN como um Plug-in

Como foi comentado no Capítulo anterior, Seção 3.2.3, as classes de GUI de BN estavam completamente acopladas a classes de GUI de MEBN (vide Figura 3.4). A principal causa era o fato da GUI do MEBN estar utilizando a classe que representa uma janela interna (`NetworkWindow`) e sua controladora (`NetworkController`) e realizando repasses a painéis de acordo com o tipo de rede sendo editada no momento. Se a rede editada era a

`MultiEntityBayesianNetwork`, instanciava-se os painéis de MEBN; caso contrário, instanciava-se os painéis de BN.

Caso essa interdependência entre painéis (e controladoras) BN e MEBN fossem mantidas em plug-ins, criaria-se uma dependência circular entre o *core* (que contém as rotinas de BN) e o plug-in de MEBN. Como o desejado seria que a dependência fosse unidirecional (o plug-in dependendo de códigos do *core*, nunca o contrário[48]), foi necessário a remoção de todos os códigos em `NetworkWindow` e `NetworkController` relacionados a MEBN.

Os códigos removidos de `NetworkController` não implementavam funcionalidades, pois eram simplesmente rotinas de repasse. No entanto, os códigos antes presentes em `NetworkWindow` implementavam papéis importantes na GUI do MEBN. Então, foi criado a classe `MEBNNetworkController` e todo código referente a MEBN foi migrado para essa nova classe. Como a nova classe estende indiretamente um `UnBBayesModule`, ele pode ser usado como classe central de plug-in de módulos (mecanismo apresentado na Seção 3.4.2). Essa refatoração está ilustrada na Figura 5.11.

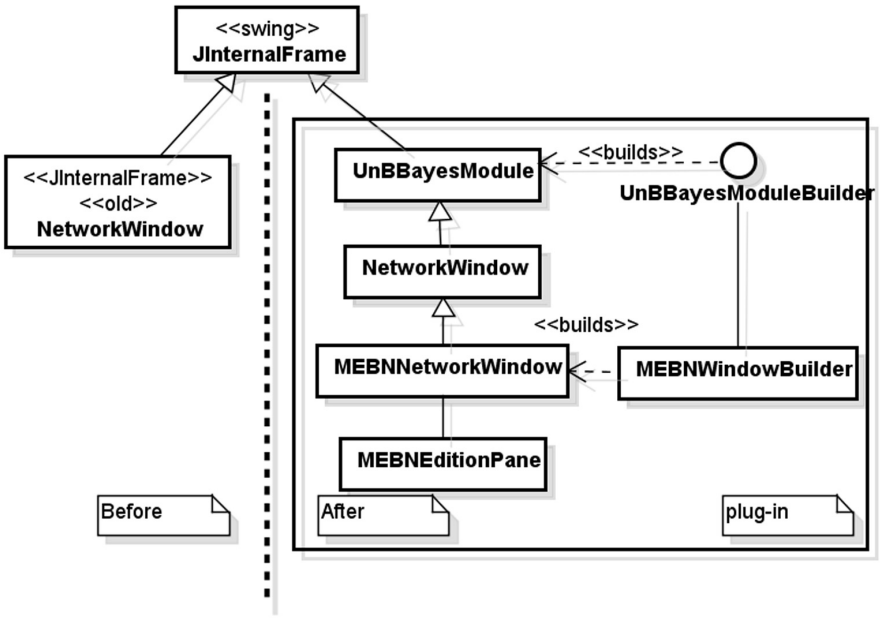

Figura 5.11: Resultado da criação da classe MEBNNetworkWindow.

A lista a seguir descreve os pontos de extensão oferecidos pelo plug-in de MEBN:

- **MEBNIO**: ponto de extensão para I/O em MEBN. Consiste em um conjunto de classes implementando a interface `unbbayes.io.BaseIO` para se adicionar funcionalidades de entrada e saída. Como exigido pela infra-estrutura de plug-ins, esta classe deve fornecer um construtor padrão sem parâmetros. Os parâmetros desse ponto de extensão são:
  - **class**: classe implementando `unbbayes.io.BaseIO`
  - **name**: nome mostrado ao usuário em caso de conflito (quando mais de uma classe pode tratar um tipo de arquivo)

[48] Uma dependência do *core* a um plug-in tornaria o plug-in mandatório.

- **KnowledgeBase**: classes implementando `IKnowledgeBaseBuilder` do pacote `unbbayes.prs.mebn.kb.extension`, para adicionar uma nova base de conhecimento na MEBN. Os parâmetros desse ponto de extensão são:
  - **class**: classe implementando `IKnowledgeBaseBuilder` (que constói instâncias de `KnowledgeBase`).
  - **name**: identificador da KB.
  - **optionPanel**: painel utilizado para preencher atributos da KB. Deve implementar `unbbayes.gui.mebn.extension.kb.IKBOptionPanelBuilder`.
- **SSBN**: classes que adicionam novos algoritmos de geração de SSBN. Os parâmetros desse ponto de extensão são:
  - **class**: classe que implementa `unbbayes.prs.mebn.ssbn.extension.ISSBNGeneratorBuilder`.
  - **name**: identificador do algoritmo de SSBN.
  - **optionPanel**: formulário para se preencher atributos do algoritmo de SSBN. Deve implementar `unbbayes.gui.mebn.extension.ssbn.ISSBNOptionPanelBuilder`.
- **MEBNEditorPanel**: paineis adicionais para se editar uma MEBN e objetos relacionados. Os parâmetros desse ponto de extensão são:
  - **class**: classe que implementa `unbbayes.gui.mebn.extension.editor.IMEBNEditionPanelBuilder`.
  - **name**: identificador de um painel em particular.
  - **icon**: ícone mostrado na aba onde o painel será inserido.
  - **description**: dicas *tooltip text*.
- **MEBNPluginNode**: declaração de novos tipos de nós para MEBN e um conjunto de informações adicionais para se mostrar esse nó na GUI. I/O e/ou algoritmos de inferência (geradores de SSBN) precisarão ser implementadas para suportar armazenamento e compilação desses novos nós. Esse ponto de extensão pode ser utilizado, por exemplo, para simular nós de input com expressões complexas em FOL. Os parâmetros desse ponto de extensão são:
  - **class**: classe do nó, implementando `unbbayes.prs.extension.IMEBNPluginNode`. Este parâmetro pode também ser um *builder* que estende `unbbayes.prs.builder.extension.PluginNodeBuilder` ou implementa `unbbayes.prs.builder.INodeBuilder`. Nesses casos, o nó criado deve ser instância de `unbbayes.prs.extension.IPluginNode`.
  - **shapeClass**: classe implementando `unbbayes.draw.extension.IPluginUShape`, usado para se desenhar o nó na tela. Este argumento pode também ser um *builder* que implementa `unbbayes.draw.extension.IPluginUShapeBuilder` e gere objetos de `unbbayes.draw.extension.IPluginUShape`.
  - **name**: nome do novo tipo de nó. Será usado como rótulo principal.

- **panelBuilder**: classe implementando `unbbayes.gui.table.extension.IProbabilityFunctionPanelBuilder`. É basicamente um `JPanel` do *swing* usado para se editar as informações de um nó.
- **description**: descrição para o nó. Será usado como dicas *tool tip*.
- **icon**: uma imagem (*e.g.* gif, png) para ser usado como ícone nos botões para criação dos nós.
- **cursor**: uma imagem (*e.g.* gif, png) para ser usado como cursor durante a criação do nó pelo usuário. O canto superior esquerdo (ponto x = 0, y = 0) será usado como o ponto de ativação (a "ponta" do cursor).

Como alterações secundárias, as classes que tratam a script de probabilidades condicionais do UnBBayes-MEBN (Seção 5.1.3) sofreram alterações pequenas voltadas na facilitação de uso no contexto de PR-OWL 2. Na especificação antiga, todos os estados deviam estar explicitados no pseudo-código. Isso era aceitável naquele momento, pois todas as informações relacionadas a variáveis aleatórias estavam unicamente definidas pela porção probabilística da PR-OWL.

Adicionalmente, na nova especificação - PR-OWL 2 - informações sobre os estados possíveis das variáveis aleatórias podem conter valores inferidos pela porção determinística também, com uso de máquinas de inferência em DL. Isso insere uma certa dinâmica nos estados possíveis de um nó, pois agora seus valores possíveis não necessariamente são "imediatos", já que precisam ser inferidos. Estados implícitos (como o valor "*absurd*", que tornou-se agora um estado possível para qualquer variável aleatória na PR-OWL 2) podem também ocorrer. Visto essas condições, a restrição que forçava com que as scripts de probabilidades condicionais contenham todos os estados possíveis (explícitos e inferíveis) passou a dificultar muito na modelagem de ontologias probabilísticas.

Por tais motivos, a restrição que forçava com que a script de probabilidades condicionais do UnBBayes-MEBN contenha todos os estados possíveis foi removida. No novo esquema, valores não declarados pela script são interpretados como probabilidade zero.

5.3 Arquitetura do Plug-in para Suporte a PR-OWL 2

Para exercitar o subconjunto da PR-OWL 2 que foi abordada na Seção 2.7, o plug-in de PR-OWL 2 permitirá as seguintes operações em um modelo MEBN:

1. carregamento de classes e indivíduos OWL que não fazem parte da taxonomia PR-OWL como "entidades" MEBN;
2. mapeamento dos nós residentes a propriedades OWL;
3. edição da parte determinística da MEBN como uma ontologia OWL;
4. delegação do raciocínio determinístico para máquinas de inferência OWL;

Para a implementação das quatro funcionalidades acima mencionadas, os seguintes pontos de extensão foram implementados (a Seção 5.2 descreveu as funcionalidades esperadas nesses pontos de extensão):

- **MEBNEditorPanel**: foi utilizado para realizar as operações 2 e 3, adicionando-se novas abas no painel de edição. Figuras 5.18 e 5.20 esquematizam a implementação realizada. Na Figura 5.18, como o Protégé adere a OSGi, este plug-in implementa um adaptador entre JPF e OSGi através da delegação de ClassLoader do OSGi para o JPF. Elementos não referenciados são elementos preexistentes (não implementados no presente trabalho) e fazem parte ou do UnBBayes, UnBBayes-MEBN ou de bibliotecas externas. Na Figura 5.20, a tela do canto superior direito é implementada pela classe OWL2PropertyViewerPanel e mostra a criação de um nó residente a partir de uma propriedade existente (via drag and drop). A tela do canto inferior direito é implementada pela classe DefinesUncertaintyOfPanel e edita a associação entre um nó residente e uma propriedade.
- **KnowledgeBase**: foi utilizado para realizar a operação 4 e uma pequena parcela da 1 (para que classes e indivíduos "não PR-OWL" sejam interpretados como entidades MEBN), 2 (para que fatos na propriedade OWL sejam também fatos no nó residente) e 3 (para editar configurações do reasoner OWL). A Figura 5.17 ilustra esta implementação. A classe `OWL2KnowledgeBaseBuilder` gera uma instância de `OWL2KnowledgeBase`, que por sua vez delega a inferência da porção determinística da MEBN a uma máquina de inferência OWL. O painel do canto direito é implementado pela classe `OWL2KnowledgeBaseOptionPanelBuilder` e oferece formulários para a configuração da `OWL2KnowledgeBase`.
- **MEBNIO**: foi utilizado para realizar a operação 1, 2 e parte da 3 (inicialização da máquina de inferência, que também é usado pelo editor gráfico). A Figura 5.13 ilustra o diagrama UML dessas classes. Para que os plug-ins implementados neste trabalho possam operar harmonicamente, objetos de uso comum foram ligados a objetos da classe MultiEntityBayesianNetwork (classe de modelo que representa uma MEBN e trafega por todos os componentes do UnBBayes-MEBN) utilizando bridge design pattern (49) (vide Figura 5.12). Como o UnBBayes-MEBN já oferecia essa funcionalidade de ligação, esta alteração pôde ser realizada com esforço mínimo. Detalhes estão apresentados na Seção 5.3.1.

Em suma, o "plug-in de PR-OWL 2" é constituído virtualmente de 4 plug-ins (duas que estendem o ponto de extensão MEBNEditorPanel, uma que estende KnowledgeBase e outra que estende MEBNIO) que operam em conjunto para uma finalidade geral: prover funcionalidades para manipulação (edição, armazenamento/carregamento e inferência) de ontologias PR-OWL 2.

5.3.1 Plug-in de Entrada e Saída em PR-OWL 2

Como previamente apresentado na Seção 5.1.10, informações peculiarmente tratadas pelo UnBBayes para representar a MEBN não são armazenadas diretamente em arquivos PR-OWL (ou PR-OWL 2). Para o armazenamento dessas informações internas do UnBBayes, utilizamos ainda o formato UBF. A Figura 5.12 apresenta as classes criadas para I/O em OWL 2 (e consequentemente para PR-OWL 2).

O formato UBF não sofreu alterações após a refatoração e criação de plug-ins, pois continua com os mesmos atributos e apontando para arquivos ".owl". As rotinas de I/O responsáveis por

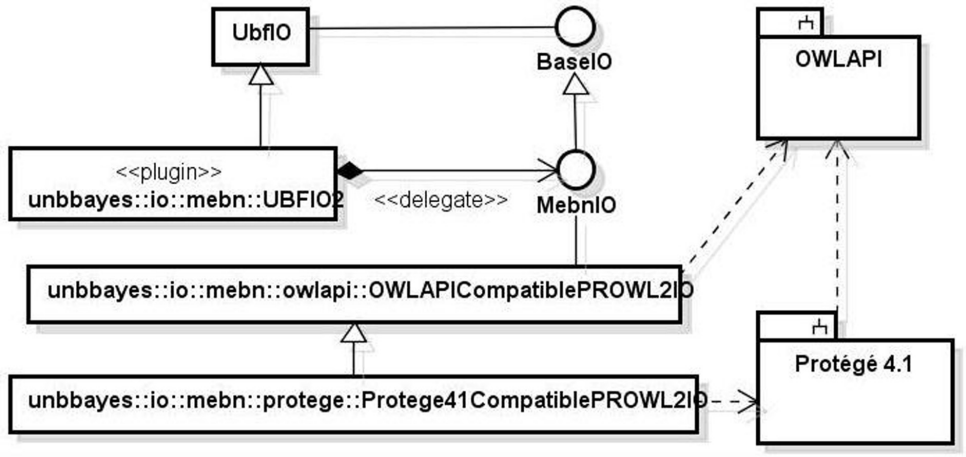

Figura 5.12: diagrama UML para classes de entrada e saída (I/O). UBFIO2 delega o carregamento de ontologias à classe OWLAPICompatiblePROWL2IO, que utiliza a biblioteca OWL API para o carregamento da ontologia OWL. Protege41CompatiblePROWL2IO é uma extensão que resolve requisitos específicos do Protégé 4.1.

tratar arquivos OWL (ou OWL 2) é responsável por verificar se o arquivo considerado é uma ontologia PR-OWL ou PR-OWL 2. Foi criado uma nova classe `UbfIO2`, extendendo a `UbfIO`, para implementar carregamento e armazenamento de arquivos UBF associados a arquivos OWL 2 (PR-OWL 2). No entanto, o tratamento dos próprios arquivos OWL 2 (PR-OWL 2) é feito pelas seguintes classes:

OWLAPICompatiblePROWLIO responsável pelo carregamento de ontologias PR-OWL 1 escritas sobre OWL 2 utilizando-se OWL API.

OWLAPICompatiblePROWL2IO responsável pelo carregamento de ontologias PR-OWL 2 (escritas sobre OWL 2, por natureza) utilizando-se OWL API.

Protege41CompatiblePROWLIO carrega algumas classes adicionais específicas de Protégé 4.1 e delega posteriormente para as rotinas de `OWLAPICompatiblePROWLIO`. Esse repasse foi somente possível porque Protégé 4.1 utiliza OWL API internamente.

Protege41CompatiblePROWL2IO carrega algumas classes adicionais específicas de Protégé 4.1 e delega posteriormente para as rotinas de `OWLAPICompatiblePROWL2IO`. Essas classes adicionais do Protégé 4.1 inclui objetos necessários para se instanciar a GUI do Protégé 4.1 (*e.g.* objetos de `EditorKit`).

A listagem 5.1 apresenta um descritor para I/O de PR-OWL 2 (UbfIO2, que aponta para o `Protege41CompatiblePROWL2IO`)

Listagem 5.1: Descritor do plug-in de I/O de PR-OWL 2.

```
1 <?xml version="1.0" ?>
2 <!DOCTYPE plugin PUBLIC "-//JPF//Java Plug-in Manifest 1.0"
3 "http://jpf.sourceforge.net/plugin_1_0.dtd">
4 <plugin id="unbbayes.io.mebn.UbfIO2" version="0.0.1">
5     <requires>
```

```
6           <import plugin-id="unbbayes.prs.mebn"/>
7           <import plugin-id="unbbayes.gui.mebn.ontology.protege"/>
8       </requires>
9       <extension plugin-id="unbbayes.prs.mebn"
10      point-id="MEBNIO" id="UbfIO2">
11          <parameter id="class"
12          value="unbbayes.io.mebn.UbfIO2" />
13          <parameter id="name"
14          value="UnBBayes File with PR-OWL 2.0" />
15      </extension>
16  </plugin>
```

Vínculo Entre Classes de Modelo do UnBBayes-MEBN com Classes de Modelo da OWL API

Para não se perder o conteúdo da parte não-probabilística da ontologia, a versão anterior do UnBBayes-MEBN armazenava em memória objetos de `OWLModel`[49], oferecidos pela API do Protégé 3. Tais objetos eram ligados à classe que representa uma MTheory[50]. Para se evitar o forte acoplamento entre classes de representação com classes de persistência, era utilizada uma combinação parcial das *design-patterns Bridge* e *Decorator*.

Esse mecanismo foi reaproveitado para que objetos específicos de OWL API ou Protégé 4.1 estejam ligadas à classe `MultiEntityBayesianNetwork` sem acoplamento demasiado. Com isso, tornou-se possível que classes de I/O e classes de KB utilizem os mesmos objetos de OWL API ou Protégé 4.1, de forma sincronizada, sem a criação de novas interfaces somente para esse propósito (já que o objeto `MultiEntityBayesianNetwork` é compartilhado pelo módulo inteiro).

O diagrama de classes que implementam essa nova funcionalidade está apresentada na Figura 5.13, e como classes de I/O se associam a elas está apresentada na Figura 5.14. A classe `ProtegeStorageImplementorDecorator` também contém uma instância da máquina de inferência utilizada pelo Protégé 4.1 (que é nada mais do que a interface oferecida pela OWL API), portanto, máquinas de inferências disponibilizadas como plug-ins do Protégé 4.1 são supostamente reutilizáveis no UnBBayes. Por padrão, o UnBBayes utiliza o HermiT como máquina de inferência para a parte determinística, por este ser o padrão que acompanha o Protégé 4.1.

`IBridgeImplementor` : interface de utilidade que deve ser implementada pelas classes que utilizam *Bridge design-pattern* em algum nível de abstração.

`IOWLAPIStorageImplementorDecorator` : classe responsável por implementar mecanismos de persistência de dados da MTheory como ontologia OWL. Ela está responsável por manter a `OWLOntology` (classe que representa uma ontologia em OWL 2 na OWL API) acessível pela MTheory de forma indireta.

`IOWLAPIStorageImplementorDecorator` : classe responsável por implementar mecanismos de persistência de dados da MTheory como ontologia OWL. Ela está responsável por manter a `OWLOntology` (classe que representa uma ontologia em OWL 2 na OWL API) acessível pela MTheory de forma indireta.

[49] O OWLModel é a classe que representa uma ontologia em OWL e permite acesso a todas as funcionalidades para sua manipulação.

[50] Seja, a classe MultiEntityBayesianNetwork

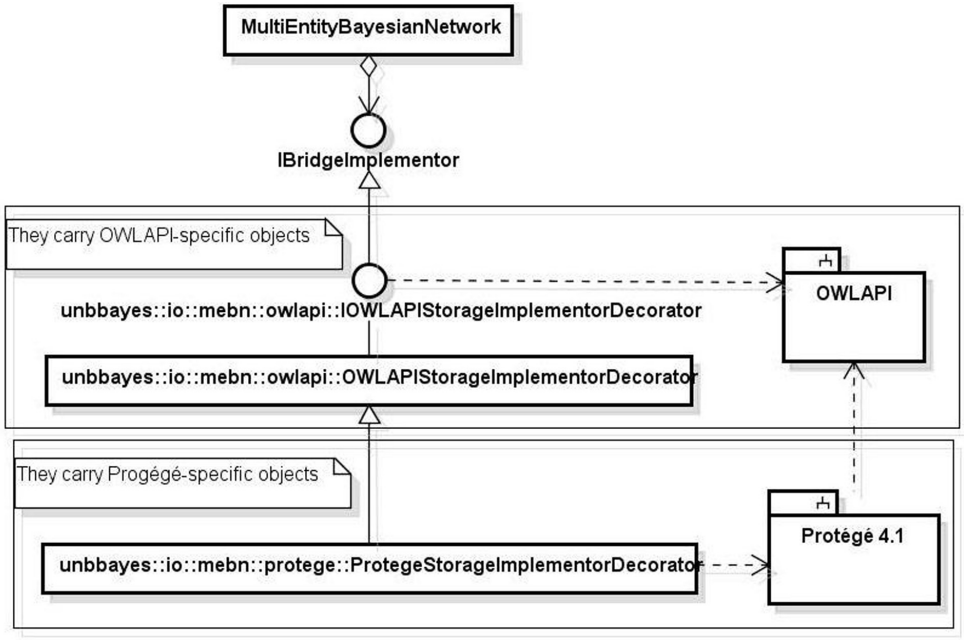

Figura 5.13: `MultiEntityBayesianNetwork` utiliza *bridge design pattern* (`IBridgeImplementor`) para criar um link para objetos específicos de implementação (*e.g.* objetos da OWL API ou Protégé 4.1).

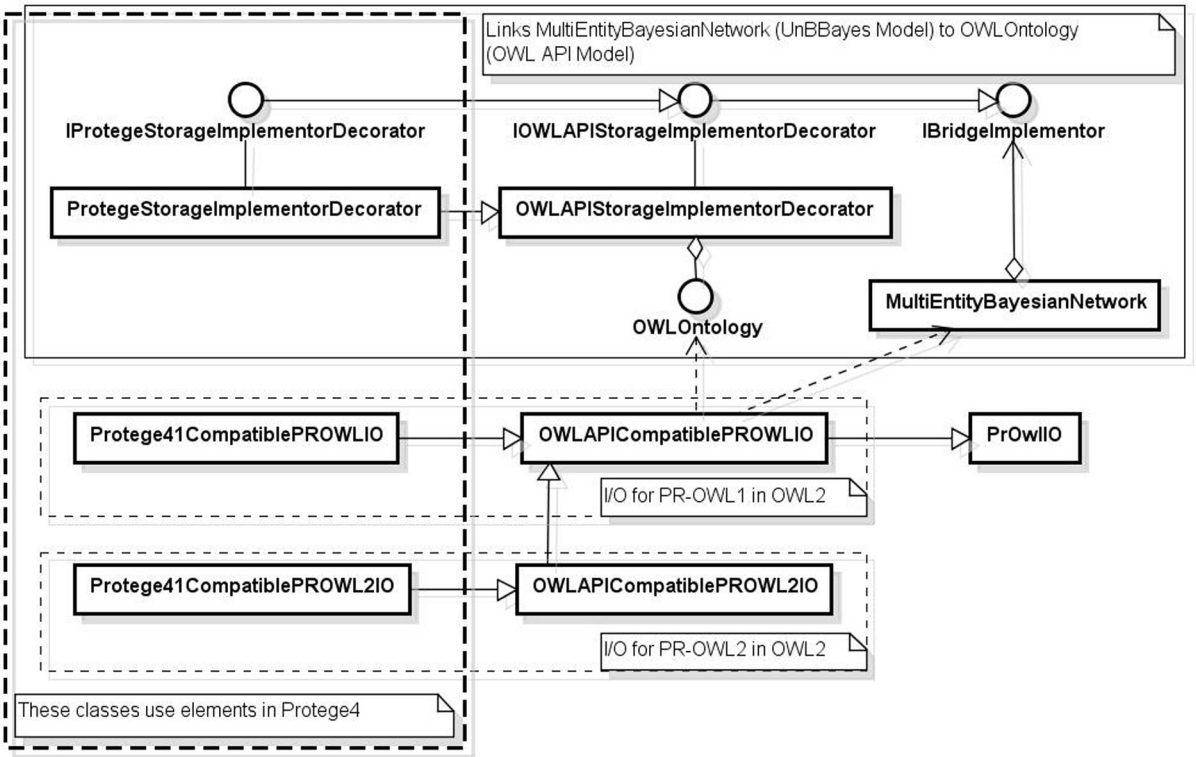

Figura 5.14: Diagrama de classes que relaciona classes de I/O com os objetos do Protégé ou OWL API. Classes de I/O utilizam OWLOntology para preencher o conteúdo de MultiEntityBayesianNetwork posteriormente adiciona um link por *bridge design pattern*.

5.3.2 Um Novo Tratamento para a Exclusividade Global

Na PR-OWL 1, existia um conceito conhecido como *exclusividade global* de um determinado estado de um nó. Essa propriedade era necessária em situações em que apenas uma evidência é permitida para uma variável em um determinado estado. Por exemplo, no MFrag `Starship Data` apresentada na Figura 5.15, o estado `True` para a variável `IsOwnStarship(st)` só é permitido para uma única espaçonave `st`. Ou seja, o estado `True` é *exclusivo globalmente* em relação à RV `IsOwnStarship(st)`.

Como tal conceito era sempre tratada como uma propriedade determinística (*i.e.* ou a propriedade sempre existe ou nunca existe); a partir da PR-OWL 2 essa propriedade passou a não ser explicitada pela parte probabilística da ontologia (*i.e.* passou a inexistir na especificação PR-OWL). Na nova especificação, a garantia de exclusividade global (*i.e.* somente uma única entidade possuir uma determinada propriedade na ontologia inteira) precisou ser responsabilidade de restrições em OWL. A exemplo, o uso de propriedades OWL inversas e funcionais permite tal restrição.

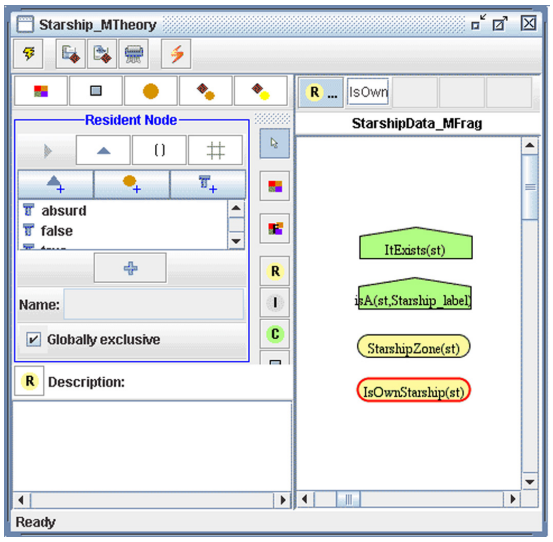

Figura 5.15: Exemplo de uso da exclusividade global - MFrag `Starship Data`.

5.3.3 Plug-in para Base de Conhecimento em PR-OWL 2

As classes presentes na implementação da base de conhecimento (KB) são apresentadas na Figura 5.16.

Interface `KnowledgeBase` contém os métodos que a base de conhecimento utilizada pelo UnBBayes deve conter. Os métodos correspondem as operações de inserir e retirar informações a base de conhecimento (a partir dos objetos MEBN) e fazer questionamentos sobre informações na base de conhecimento (a partir de nós de contexto). Esta interface não foi alterada com a refatoração do UnBBayes-MEBN.

Classe `OWL2KnowledgeBase` implementação de `KnowledgeBase` utilizando interfaces da OWL API. Como a MEBN (`MultiEntityBayesianNetwork`) carrega um

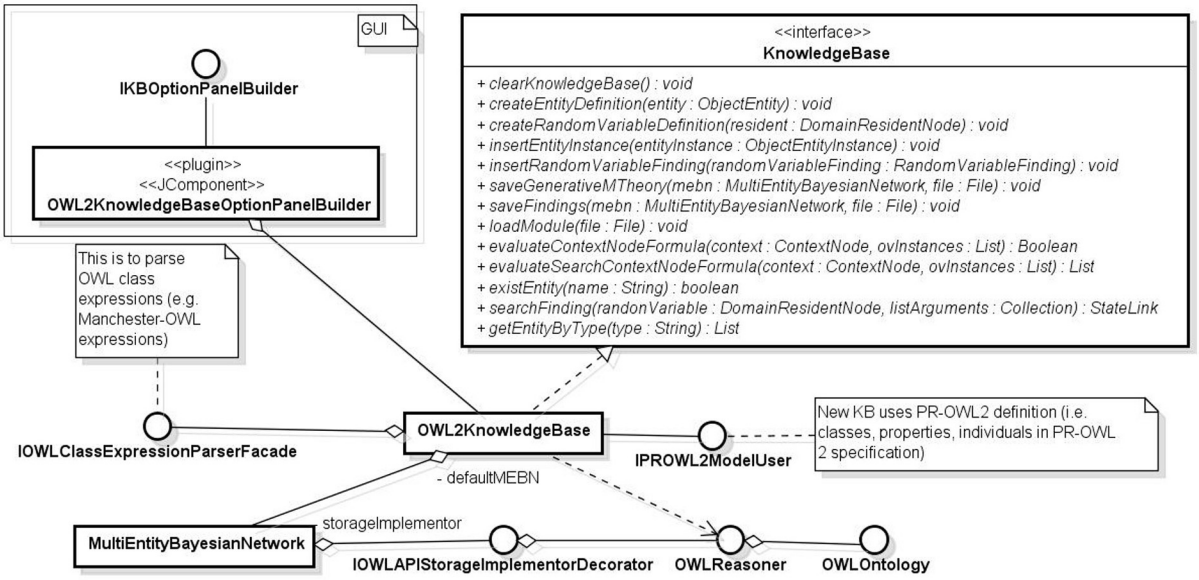

Figura 5.16: Modelagem do uso de máquinas de inferência de OWL 2 (DL) para implementar KB.

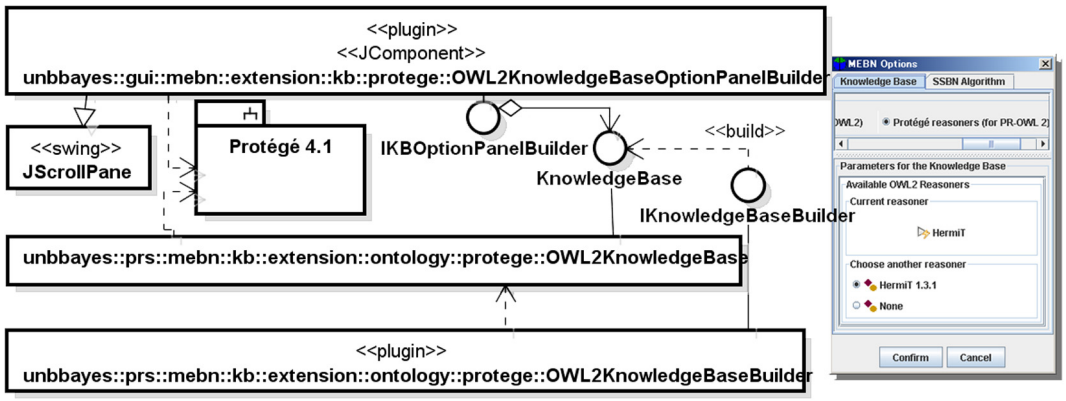

Figura 5.17: diagrama UML do plug-in que representa uma base de conhecimento expresso em OWL2. A KB é ativada selecionando-o na tela de opções da MEBN.

`IOWLAPIStorageImplementorDecorator`, que por sua vez carrega um reasoner, sempre a mesma instância do reasoner é reutilizado na inferência integrada.

Classe `OWL2KnowledgeBaseBuilder` builder de `OWL2KnowledgeBase`.

Classe `OWL2KnowledgeBaseOptionPanelBuilder` classe esperada pelos pontos de extensão do novo UnBBayes-MEBN para representar um painél de opções para esta KB. Esse painel aparecerá nos painéis de configuração (preferências) na barra de edição da MEBN.

Interface `IOWLClassExpressionParserFacade` realiza parsing de expressões em sintaxe Manchester-OWL. É usado para facilitar consultas em DL.

Interface `IPROWL2ModelUser` interface implementada por todas as classes que utilizam o modelo PR-OWL 2.

Para a avaliação dos nós de contexto são utilizados os dois métodos declarados na interface `KnowledgeBase`:

- `evaluateContextNodeFormula(context: ContextNode, ovInstances: List<OVInstance>)`: avalia um nó de contexto de fórmula simples para verdadeiro ou falso.
- `evaluateSearchContextNodeFormula(context: ContextNode, ovInstances: List<OVInstance>)`: avalia um nó de contexto de fórmula complexa, retornando uma lista com indivíduos OWL que satisfazem a restrição definida no nó de contexto, de acordo com os mapeamentos de propriedades definidos na Seção 2.7.

Como visto na Seção 2.7, em PR-OWL 2, os nós residentes podem ser ser traduzidos para propriedade de dados (quando o nó residente for booleano e com 1 argumento), relacionamentos (propriedade ligando 1 ou mais indivíduos OWL) ou para funções (propriedades funcionais). No UnBBayes, os nós residentes booleanos (com mais de 1 argumento) são traduzidos para relacionamentos e todos os outros são traduzidos para funções cuja imagem é um conjunto de indivíduos (podendo ser então uma classe OWL).

A DL (utilizada na KB de PR-OWL 2) implementa nativamente as *builtInRV* (*i.e.* operações como *and, or, not, forAll*, ou *exists* são implementadas nativamente). No entanto, por causa de diferenças de expressividade entre FOL (utilizada em expressões das fórmulas dos nós de contexto) e DL, as fórmulas dos nós de contexto não podem ser diretamente mapeadas para consultas na ontologia PR-OWL 2, principalmente por causa das consultas DL não poderem ser feitas para diversas variáveis ordinárias simultaneamente.

Uma abordagem adotada para se sanar esse problema é o particionamento da consulta em diversas sub-consultas, com posterior integração dos resultados por programação Java. Por exemplo, na Figura 5.9, dado como desconhecidos os valores das variáveis ordinárias obj e rpt, a consulta (*i.e.* chamada ao método `evaluateSearchContextNodeFormula`) à KB pela expressão `obj = ReportedObject(rpt)`[51] exige o seguinte procedimento:

[51] Assumindo-se que o argumento de `ReportedObject` está mapeado como sujeito da propriedade OWL de mesmo nome e o seu valor como a imagem da propriedade OWL com mesmo nome.

1. como foram identificados 2 variáveis ordinárias desconhecidas na consulta (obj e rpt), primeiramente faremos uma consulta de quais valores possíveis de rpt possuem algum vínculo com algum obj via propriedade OWL `ReportedObject` (assumindo-se que essa seja a propriedade OWL mapeada com o nó residente `ReportedObject`);
2. extraia os tipos de obj e rpt (no UnBBayes-MEBN, a tipagem é forte, portanto, a extração desses tipos é feita de forma imediata, sem consultas na KB);
3. seja Object o tipo de obj e Report o tipo de rpt, a nova consulta em sintaxe Manchester-OWL será "Report that ReportedObject some Object";
4. para cada indivíduo "<ind>" resultante da consulta anterior, realize a consulta "Object that (inverse ReportedObject value <ind>)";
5. combine os resultados (criação de um conjunto de pares, tal que cada par seja constituído por um indivíduo de Report e outro de Object) e retorne.

Requisições com mais de duas variáveis ordinárias desconhecidas podem ser realizadas de forma similar. Infelizmente, esta sequência adicional de consultas gera um gargalo, identificado na Seção 5.4.

5.3.4 Mecanismo de Integração da GUI do Protégé 4.1

A GUI do Protégé 4.1 foi utilizada como painel principal para edição da porção não-probabilística da ontologia PR-OWL 2. Sua implementação segue o diagrama da Figura 5.18.

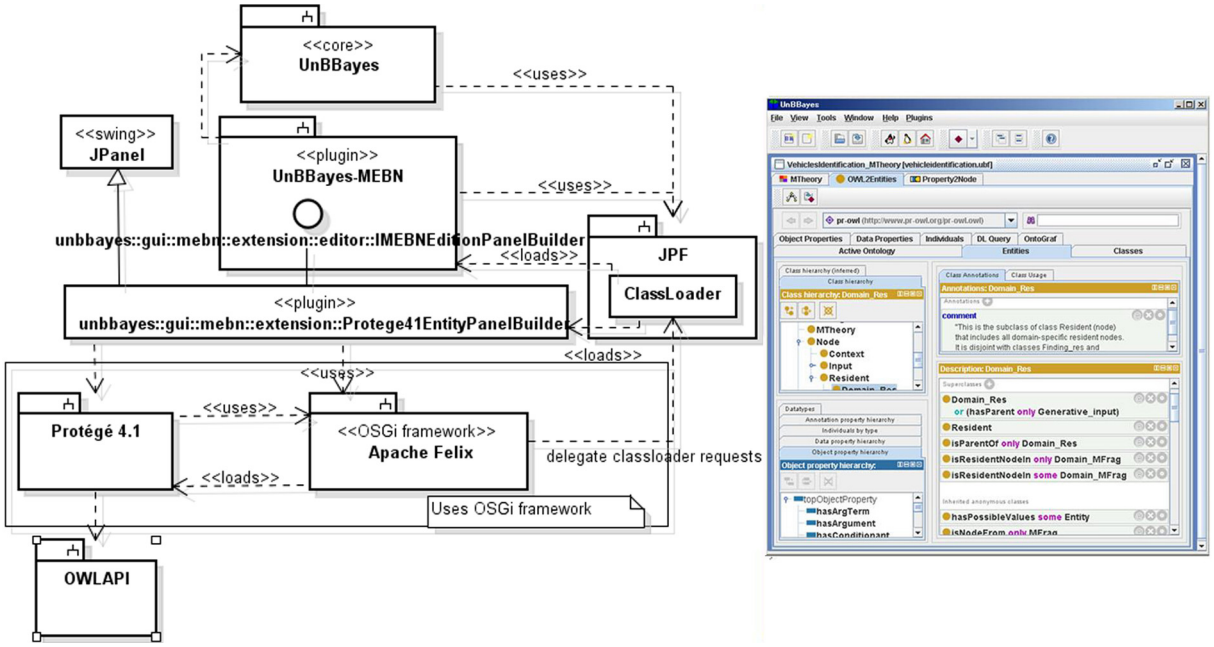

Figura 5.18: diagrama UML do plug-in de edição da ontologia determinística usando classes do Protégé 4.1 (esquerda) e captura de tela dessa funcionalidade (direita).

Resumindo-se os conceitos apresentados na Figura 5.18, o Protégé 4.1 (não somente sua GUI) está encapsulada como um plug-in (representada pela classe

`Protege41EntityPanelBuilder`) que estende o ponto de extensão MEBNEditorPanel; portanto, por implicação desse ponto de extensão, implementa a interface `IMEBNEditionPanelBuilder`. O UnBBayes-MEBN carrega esse plug-in via JPF. O plug-in, por sua vez, utiliza o Apache Felix (uma implementação do framework OSGi) para o carregamento dos components do Protégé 4.1, que estão estruturados como plug-ins no padrão OSGi.

O Protégé 4.1, apesar de oferecer um ambiente de plug-ins no padrão OSGi, não provê uma API para acesso a suas funcionalidades por aplicações que não se aderem à infra-estrutura particular de plug-ins do Protégé. O acesso a classes principais do Protégé 4.1 (vide Seção 3.3.3), como `OWLOntologyModel`, ou `EditorKit` eram impossibilitados, pois seu único caminho de acesso era via `ProtegeManager`, mas tal classe só se podia acessar de forma estática. Esse acesso estático é impossível quando estamos executando em contextos diferentes de carregadores de classes (*i.e.* classloaders incompatíveis).

A Figura 5.19 esquematiza a solução utilizada para se resolver o problema de contextos de classloaders. De forma geral, propriedades do OSGi (nesse caso, Apache Felix) foram alteradas primeiro para que os pais de cada classloaders de cada plug-in do OSGi seja o classloader do próprio framework (alterando-se a propriedade "org.osgi.framework.bundle.parent" ao valor "framework"). Como o framework (Felix) é carregado pelo plug-in de PR-OWL 2 (ou seja, pelo classloader do JPF), gera-se automaticamente uma hierarquia de classloaders com o JPF acima de OSGi. Após criada essa hierarquia, adiciona-se os pacotes do Protégé 4.1 na propriedade "org.osgi.framework.bootdelegation". Com isso, requisições de carregamento de classes pertencentes a pacotes do Protégé 4.1 serão repassados a classloaders pais (que eventualmente chegarão ao JPF, que carregará as classes). Nesse esquema, classloaders do JPF e OSGi estarão executando no mesmo contexto, resolvendo-se o problema que impedia o acesso a objetos estáticos.

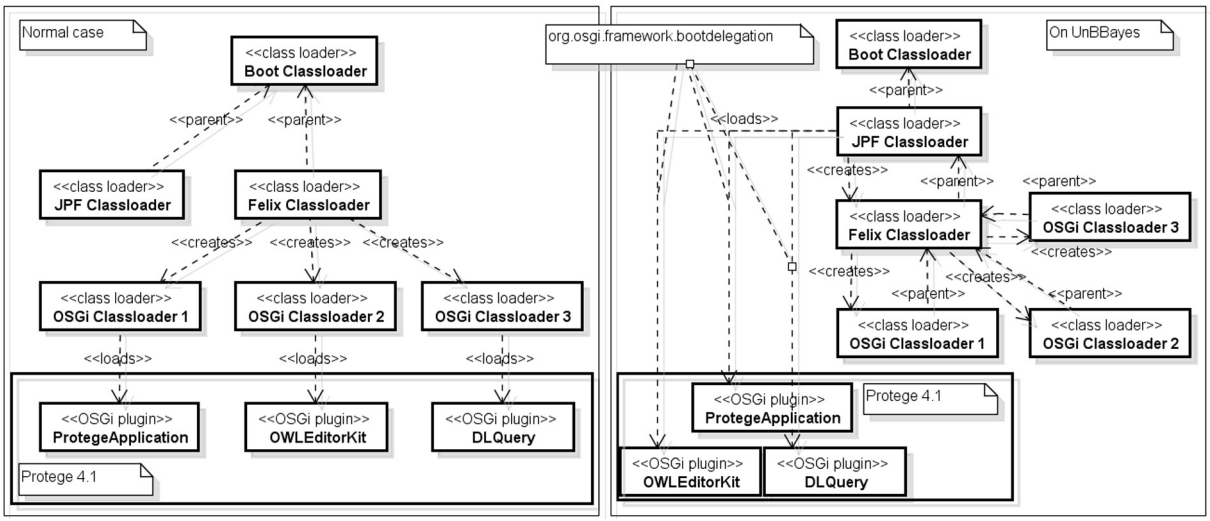

Figura 5.19: Organização de classloaders e esquema de repasse de carregamento de classes de OSGi a JPF. O uso convencional encontra-se à esquerda, e o esquema utilizado no UnBBayes encontra-se à direita.

Como a GUI do Protégé agora está sendo utilizado como editor primário para a porção determinística de PR-OWL 2 (e consequentemente de entidades e evidências), os painéis apresen-

tados na Seção 5.1.6 são ocultados, para se evitar um conflito causado por dois meios distintos de inserção de eviências e entidades.

5.3.5 Painel de Mapeamento de Propriedades OWL

Painéis foram criados para permitir que o usuário mapeie uma propriedade a uma variável aleatória (e consequentemente a um nó residente) e mapear os argumentos a domínios ou imagens de propriedades de forma intuitiva (via *drag-and-drop* ou seleção de propriedades em uma lista). A Figura 5.20 ilustra as alterações realizadas. A listagem 5.2 apresenta o descritor do plug-in de PR-OWL 2, relacionada a todas as funcionalidades previamente descritas, exceto I/O, que foi migrado a um descritor separado (listagem 5.1).

Listagem 5.2: Descritor do plug-in PR-OWL 2

```
1  <?xml version="1.0" ?>
2  <!DOCTYPE plugin PUBLIC "-//JPF//Java Plug-in Manifest 1.0"
3  "http://jpf.sourceforge.net/plugin_1_0.dtd">
4      <requires>
5          <import plugin-id="unbbayes.util.extension.core"
6          plugin-version="4.2.3" />
7          <import plugin-id="unbbayes.prs.mebn"
8          plugin-version="1.8.7"/>
9      </requires>
10     <extension plugin-id="unbbayes.prs.mebn"
11     point-id="MEBNEditorPanel" id="Protege41Entities">
12         <parameter id="class"
13         value="unbbayes.gui.mebn.extension.Protege41EntityPanelBuilder" />
14         <parameter id="name" value="OWL2Entities" />
15         <parameter id="description"
16         value="Edit OWL2 entities using Protege 4.1 editor" />
17         <parameter id="icon" value="class.gif" />
18     </extension>
19     <extension plugin-id="unbbayes.prs.mebn"
20     point-id="MEBNEditorPanel" id="OWL2Properties2Node">
21         <parameter id="class"
22         value="unbbayes.gui.mebn.extension.OWL2PropertyImportPanelBuilder"
               />
23         <parameter id="name" value="Property2Node" />
24         <parameter id="icon" value="properties2node.png" />
25         <parameter id="description"
26         value="Convert OWL2 properties to MEBN resident nodes" />
27     </extension>
28     <extension plugin-id="unbbayes.prs.mebn"
29     point-id="KnowledgeBase" id="Protege41Reasoners4PROWL2">
30         <parameter id="class"
31 value="unbbayes.prs.mebn.kb.extension.ontology.protege.
       PROWL2KnowledgeBaseBuilder"/>
32         <parameter id="name" value="Protege reasoners (for PR-OWL 2)"/>
33         <parameter id="optionPanel"
34 value="unbbayes.gui.mebn.extension.kb.protege.
       OWL2KnowledgeBaseOptionPanelBuilder"/>
35     </extension>
36 </plugin>
```

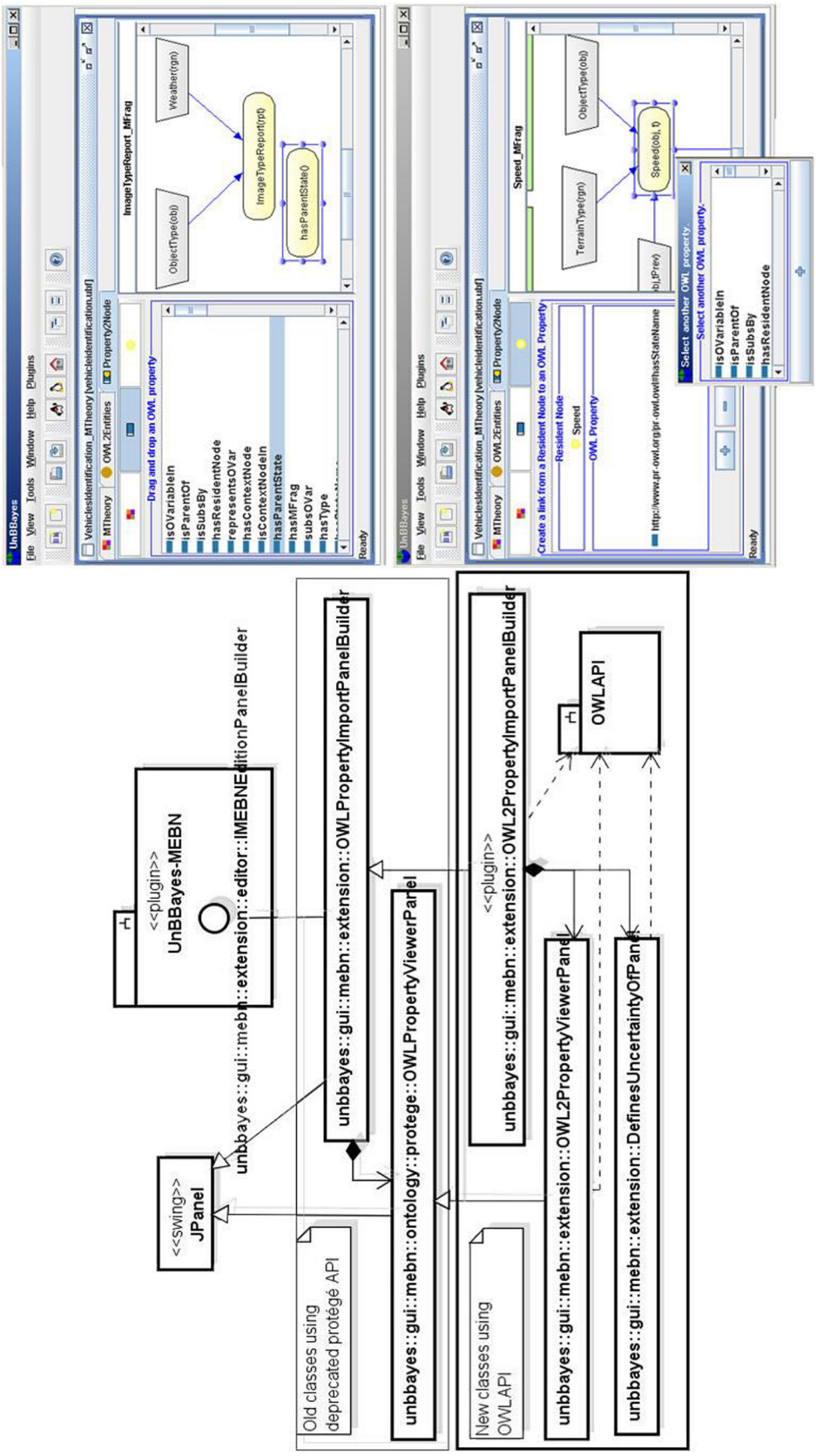

Figura 5.20: diagrama UML do painel de relacionamento entre um nó residente (MEBN) e propriedade (OWL).

5.4 Análise de Desempenho - Profiling

O plug-in TPTP (*Test & Performance Tools Platform Project*)(2) do ambiente de desenvolvimento Eclipse foi utilizado como ferramenta para a análise de tempo de execução e uso de memória do plug-in de PR-OWL 2 (e seu funcionalmento em conjunto com o plug-in UnBBayes-MEBN) na geração de SSBN na ontologia de identificação de veículos (78). A ontologia de identificação de veículos está disponível para download junto com o plug-in UnBBayes-MEBN na pasta de exemplos.

O Eclipse TPTP, isolado, é um framework para a construção de ferramentas de teste e análise de performance. Sua distribuição padrão é acompanhada por um ambiente integrado de teste JUnit, analisador de desempenho (tanto local ou remoto), mecanismo de inserção de código em *bytecode* e um analisador gráfico de logs de sistemas; úteis durante todo o ciclo de vida de desenvolvimento de sistemas.

Para a coleta de informações de desempenho, uma ferramenta adicional é utilizada pelo TPTP. O *AgentController*. Essa ferramenta gerencia um processo *daemon* que interliga dois processos clientes. O TPTP, rodando no Eclipse, acessará o aplicativo testado com o intermédio do *AgentController*. Tanto o TPTP quanto o *AgentController* podem ser obtidos no site principal (2).

Basicamente, os seguinte passos foram testados na avaliação de performance do PR-OWL 2 (executando sobre o UnBBayes e no plug-in UnBBayes-MEBN):

1. inicialização do UnBBayes-MEBN;
2. abertura do arquivo UBF de identificação de veículo[52] (e conseqüentemente de um arquivo PR-OWL 2 associado);
3. configuração das opções de MEBN para que passe a usar a base de conhecimento em OWL (OWL 2);
4. sincronização da base de conhecimento (pressionando-se o botão "carregar base de conhecimento");
5. insersão das seguintes evidências (assertivas de propriedades OWL 2 em indivíduos) pelo painél de edição da porção determinística (painel do Protégé 4.1):
   - RPT1 → REPORTEDOBJECT → OBJ1,
   - RPT2 → REPORTEDOBJECT → OBJ2,
   - RPT1 → REPORTEDREGION → RGN1,
   - RPT2 → REPORTEDREGION → RGN2,
   - OBJ1 → LOCATION → RGN1,
   - OBJ2 → LOCATION → RGN2,
   - RGN1 → TERRAINTYPE → VERYROUGH,
   - RGN2 → TERRAINTYPE → ROAD;
6. execução da *query* de `ObjectType(OBJ1)`;

[52] O nome do arquivo presente na pasta de exemplos de MEBN é `VehicleIdentification.ubf`.

O filtro usado eliminava classes não pertencentes ao UnBBayes. O laudo foi armazenado em arquivos `trcxml` do TPTP. Os laudos da avaliação dos dez maiores tempos de execução e dez maiores usos de memória para o UnBBayes executando o plug-in de PR-OWL 2 estão apresentados na Figura 5.21.

A análise de memória e tempo de execução resultou nas seguintes observações:

1. dentre os 10 métodos mais demorados, classes da PR-OWL 2 aparecem 5 vezes;
2. o primeiro e o terceiro método mais demorado são de rotinas para consulta da base de conhecimento OWL 2;
3. o plug-in de PR-OWL 2 não possui problema aparente quanto ao uso de memória;
4. o quinto método mais demorado ocorre carregando-se plug-ins do Protégé.

Por ser uma responsabilidade do OSGi, o a última observação não é algo tangível por nossa equipe. No entanto, estima-se que não será um problema grave, pois o Protégé é carregada uma vez como GUI e não é mais recarregada durante o ciclo da ontologia probabilística. Portanto, é um custo a pagar somente uma vez.

Por outro lado, a demora na execução da base de conhecimento é um problema; mas é algo esperado, pois ocorre um grande gargalo com divisões de consulta em FOL em múltiplas consultas em DL. Uma solução baseada em caching amenizaria o problema.

Hiehest 10 base time

Method	Class	<Base Time (se...	Average Base Time (seconds)	Cumulative Tim...	Calls
searchFinding(unbbayes.prs.mebn.ResidentNode, java.util.Collection) unbbayes.prs.mebn.entity.StateLink	OWL2KnowledgeBase	51.244025	2.846890	51.266453	18
actionPerformed(java.awt.event.ActionEvent) void	MEBNEditionPane$ToolBarGlobal...	30.775085	15.387542	91.187495	2
fillFindings(unbbayes.prs.mebn.ResidentNode) void	OWL2KnowledgeBase	19.848100	1.984810	19.941406	10
doLoadKnowledgeBase() void	MEBNEditionPane$ToolBarGlobal...	15.467586	15.467586	37.440919	1
startProtegeBundles() java.util.Collection	ProtegeBundleLauncher	5.461746	5.461746	5.468211	1
initialize() void	PowerLoomKB	5.253105	5.253105	5.258258	1
showFileChooser() java.io.File	UnBBayesFrame$OpenFileUsing...	4.591926	4.591926	6.667801	1
loadMebn(java.io.File) unbbayes.prs.mebn.MultiEntityBayesianNetwork	Protege41CompatiblePROWL2IO	4.365648	4.365648	15.999482	1
actionPerformed(java.awt.event.ActionEvent) void	MEBNEditionPane$ToolBarGlobal...	2.910734	2.910734	11.188609	1
solveFormulaTreeOVEqualsToNonBooleanNode1Argument(unbbayes.prs.mebn.context.NodeFormulaTree, unbbay	OWL2KnowledgeBase	2.726639	0.340830	2.741908	8

Class	Total Instances	Live Instances	Collected	<Total Size (bytes)	Active Size (bytes)
Compiler$EntityAndArguments	1,167,732	0	1167732	46,709,280	0
UShapeSizeBtn	4,088	4,064	24	3,303,104	3,283,712
UShapeState	1,992	1,491	501	1,657,344	1,240,512
UShapeLine	1,009	1,009	0	879,848	879,848
UShapeProbabilisticNode	497	497	0	421,456	421,456
ProbabilisticTable	2,030	1,091	939	146,160	78,552
FloatCollection	3,651	744	2907	116,832	23,808
MEBNEditionPane$EmptyPanel	60	60	0	48,000	48,000
Compiler$SimpleProbabilityValue	664	0	664	31,872	0
OVInstance	994	251	743	31,808	8,032

Figura 5.21: Laudo de tempo-base da execução do plug-in de PR-OWL 2 (esquerda) e espaço total (direita).

Capítulo 6

Resultados

Como mencionado na Seção 3.1, a modelagem de *features* é uma técnica amplamente utilizada na comunidade de SPL para a especificação de domínios. Nesse contexto, FM podem ser aplicados para capturar comunalidades e variabilidades, definir o escopo e especificar configurações e derivações possíveis em uma SPL.

Por outro lado, uma outra técnica bastante utilizada na engenharia de conhecimento (e por sua vez aplicada na ES) é a modelagem de ontologias, como aquelas que utilizam a OWL (55) ou diagramas UML (89) como linguagens para sua representação. Como ontologias são especificações formais de domínios, FM podem ser considerados como visões particulares de ontologias (35). Portanto, o uso de linguagens para especificação de ontologias torna-se um candidato natural na especificação de FM.

Neste capítulo, o relacionamento conceitual entre FM e ontologias foi explorado para que técnicas e formalismos associados a ontologias fossem aplicados na especificação de um FM. Particularmente, nosso foco está no uso da semântica OWL 2 para expressar FMs no contexto de DSPL, para que ontologias de FM possam ser interpretadas em tempo de execução e oferecer um meio consistente para a representação de configurações válidas e transições entre elas. Um plug-in no UnBBayes foi desenvolvido baseando-se nesse cenário de uso.

Portanto, este capítulo apresenta um uso da OWL 2 (110) como linguagem de especificação do FM do UnBBayes, resultante da pesquisa desta dissertação. Adicionalmente, é apresentado um cenário que utiliza inferências OWL 2 no âmbito de DSPL. A seção seguinte (Seção 6.1) descreve com mais detalhes o contexto desse problema e sua solução. A Seção 6.2 detalha um cenário de uso e como o FM foi estendido para se especificar as condições necessárias para a transição entre configurações. Em seguida, a Seção 6.3 apresenta uma breve análise do uso da OWL 2 no FM do UnBBayes. Por fim, a Seção 6.4 apresenta os impactos da presente pesquisa (UnBBayes e sua implementação de PR-OWL 2) no mundo.

Vale notar que o uso de OWL para descrever o FM permitiu que a arquitetura do UnBBayes esteja descrita como um metamodelo, em um formato padronizado que permita a intercomunicação entre diferentes sistemas. Essa abordagem também possibilitou que o próprio UnBBayes realize inferência - seja determinística ou probabilística - sobre sua própria arquitetura. Esta capacidade de autoconhecimento possibilitará futuramente a implementação de diversas funcionalidades avançadas, como a auto reconfiguração (e.g. download de plug-ins) ao identificar

padrões de uso[53].

A ontologia UnBBayes-FM, apresentada neste capítulo, foi criada para duas finalidades:

- para documentar a arquitetura geral do sistema, em nível de *features* e
- para ilustrar uma possível aplicação de PR-OWL 2 na área de ES.

Em suma, a ontologia UnBBayes-FM, apresentada neste capítulo, consiste em um modelo criado para se documentar a arquitetura do UnBBayes como um modelo de *features*, em um formato tratável pelo próprio. A linguagem OWL 2 foi selecionada por ser um formato imediatamente prorrogável a uma ontologia probabilística em PR-OWL 2, objeto de estudo do presente mestrando. As seções seguintes descrevem a ontologia.

6.1 Especificando FM em OWL

No âmbito de DSPL (Seção 3.1.2), em que variabilidades são resolvidas em tempo de execução, o sistema precisará lidar com transições de configurações. Para isso o sistema precisa conhecer, no mínimo, as seguintes informações:

- **consistência da configuração** - se a configuração atual ou a futura (após a transição de configuração) é válida;
- **conjunto de *features* impactadas** - um espaço amostral que contenha todas as *features* que devem ser removidas, acrescentadas ou mantidas intactas durante a transição.

Em outras palavras, uma implementação de DSPL necessita de um certo conhecimento sobre o FM da própria. Obviamente, é inaceitável que essa informação (FM) esteja *hard-coded* no software, principalmente caso a SPL esteja sujeta a alterações em nível de modelagem de domínio, ou caso o espaço de *features* seja desconhecido à priori (e.g. as *features* poderão ser implementadas como plug-ins, que por sua vez podem ser implementadas por terceiros). Nesses últimos casos, torna-se importante que FM seja "interpretado" pelo software; portanto, esteja especificado em uma linguagem "interpretável" por uma máquina.

Durante o processo de especificação de uma SPL, dinâmica ou não, sempre nos deparamos com a tarefa de criação de um FM. A tarefa de criação de FM está associada à tarefa de seleção de uma linguagem para a especificar. Naturalmente, se selecionarmos uma linguagem que possa ser interpretada pelo próprio software, estaremos automaticamente resolvendo o problema acima descrito sobre FM em DSPL.

Como previamente comentado, uma ontologia é uma descrição formal de um domínio, e um FM pode ser considerado como uma visão de uma ontologia. Nesse âmbito, a OWL versão 2 (110), linguagem recomendada pela W3C como linguagem para especificação de ontologias, oferece um meio expressivo para especificar uma taxonomia de classes e propriedades que represente um conjunto de *features* e restrições sobre elas, tornando-se então em um forte candidato como linguagem para especificação de FM. A Figura 6.2 ilustra o esquema de mapeamento dos elementos comuns em FM para entidades OWL 2, adaptada de (111). Elementos

[53] Apesar da inferência já estar implementada, por questões técnicas a auto reconfiguração e identificação de padrões de uso não foram implementadas.

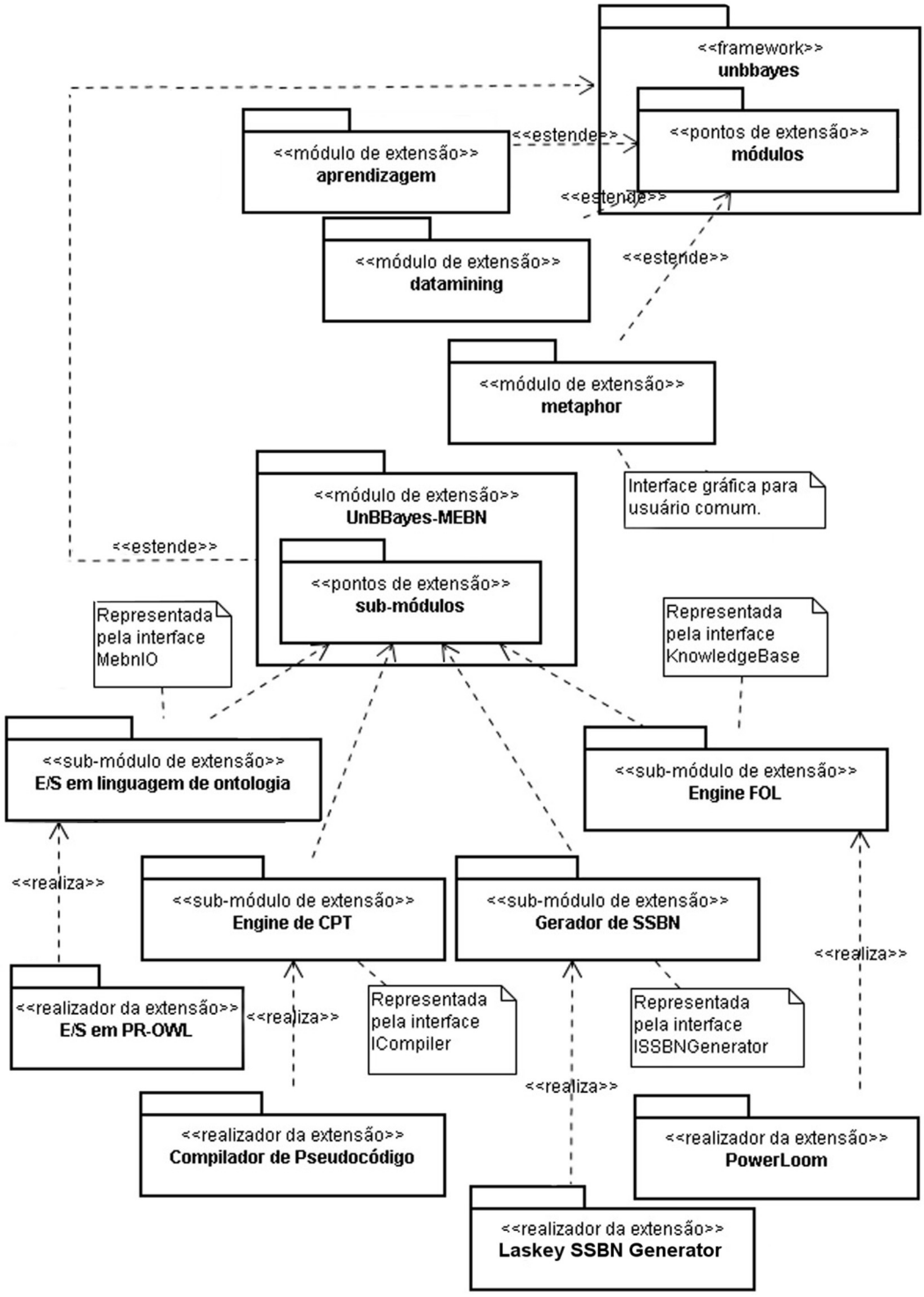

Figura 6.1: Diagrama UML ilustrando alguns pontos de extensão existentes no UnBBayes.

com formato quadrilátero representam *features*, as elipses representam classes OWL, e as arestas representam seus relacionamentos. Maiores informações sobre as regras de tradução podem ser obtidas no referido artigo original.

De maneira geral, cada *feature* é traduzida para duas classes OWL: uma classe que representa a *feature* e outra classe que contém regras impostas à *features*. As caixas em cinza são *features*, elipses em amarelo são classes OWL. Arcos ligando classes OWL são propriedades OWL e caixas de texto em branco incidindo em classes OWL são restrições expressas em DL. Dependências entre *features* podem ser traduzidas em OWL de maneira idêntica a *features* obrigatórias.

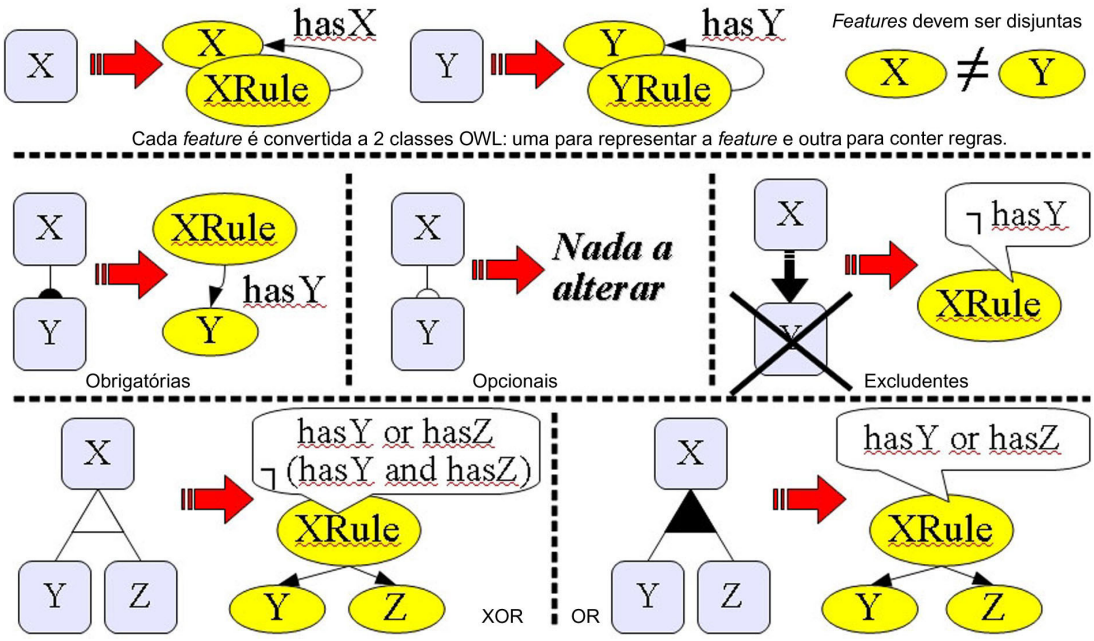

Figura 6.2: Esquema de tradução dos elementos comuns em um FM para classes e propriedades da OWL 2.

O uso de mecanismos de extensão de ontologias OWL 2 (e.g. *import*, subclasses, subpropriedades) permite que regras e fatos sejam adicionadas a FM, possibilitando inclusive que informações como o CK. Adicionalmente, por se tratar de DSPL, podemos integrar a ontologia de FM a uma outra ontologia que descreva eventos necessários para engatilhar transições de configuração. Com isso, um modelo que permite uma representação integrada do FM, CK, eventos de transição e configurações alvo da transição (configuração esperada após observada um conjunto de eventos) pode ser criado como uma ontologia única. A Seção 6.2 oferece um exemplo de cenário que aplica este conceito.

6.2 Cenário de Uso

Como as principais vantagens previstas em se especificar FM em OWL 2 está na sua alta expressividade (que implica em extensibilidade) e na possibilidade de interpretação pelo próprio software, um cenário de uso foi modelado e implementado em uma aplicação Java para permitir

o exercício da abordagem em questão. Os objetivos no exercício deste cenário de uso se resume nos seguintes dois pontos:

1. exercitar a extensibilidade de FM, quando especificadas como ontologias OWL2, a partir de adição incremental de informações na ontologia;
2. realizar uma transição de configuração de uma DSPL a partir de uma interpretação computacional do FM.

6.2.1 Procedimento

O procedimento adotado para a realização deste cenário de uso se resume nos seguintes quatro passos:

- criação de FM;
- associação de FM com URLs para download de artefatos (e.g. plug-ins) que os implementam (isso, no nosso contexto, se equivale a adicionar CK na ontologia);
- criar a ontologia categorizadora de eventos (de transição) a partir do FM, utilizando mecanismo de *import* da OWL2;
- incorporação do modelo de transição (de configuração) para associar o categorizador de eventos com um conjunto de *features* que a configuração futura deve possuir;

Todos esses passos serão realizados no ambiente da ferramenta UnBBayes, que está apresentada na Seção 6.2.2. A Figura 6.3 apresenta o esquema de comunicação entre a DSPL e a máquina de inferências OWL2. No caso do UnBBayes, ambos nós (DSPL e máquina de inferência) residem no mesmo processo; ou seja, o UnBBayes fará o papel de ambos. Segue abaixo a descrição dos passos apresentados na figura:

(i) a DSPL envia informações sobre a configuração atual (e.g. identificadores de um conjunto de *features* ativos) e um conjunto de eventos observados para o repositório OWL2;

(ii) o repositório OWL 2 só permitirá que as informações enviadas em (i) sejam inseridas caso elas estejam consistentes com as regras em DL especificadas na ontologia, se consistente, a máquina de inferência classificará o conjunto de eventos[54];

(iii) a ontologia OWL 2 possuirá uma associação entre os eventos classficados e um conjunto de *features*, portanto, a máquina de inferência poderá retornar a DSPL informações sobre o conjunto de *features* (essa será a configuração futura), incluindo endereços dos artefatos que implementam as tais;

(iv) baseada no conjunto retornado, a DSPL realizará um download dos artefatos tangíveis (e.g. plug-ins) a partir dos endereços associados.

[54]Por exemplo, um conjunto de eventos de botões pode ser classificado como "padrões típicos de um usuário iniciante". Com isso, pode-se selecionar um conjunto de *features* apropriadas para um usuário iniciante.

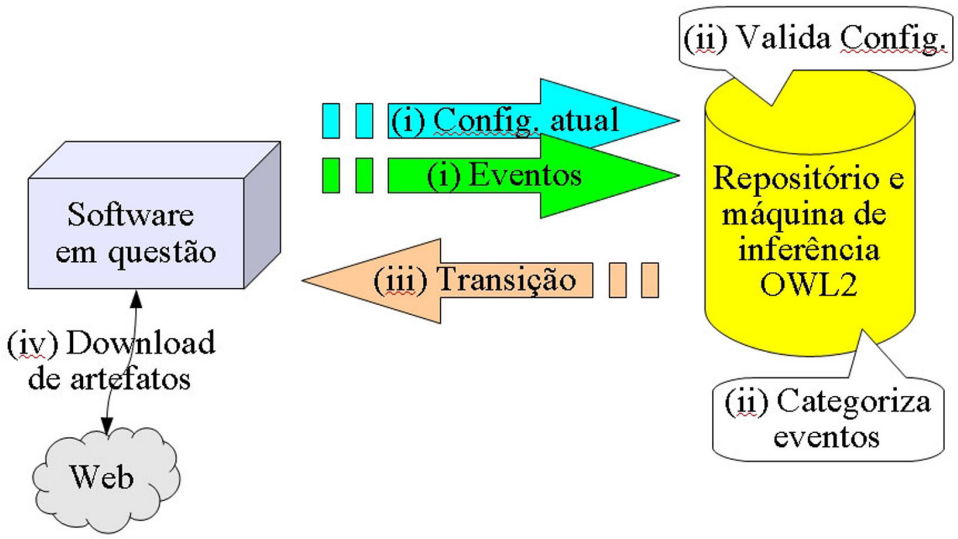

Figura 6.3: Esquema de comunicação entre uma DSPL e um repositório OWL 2 acompanhado de uma máquina de inferências.

6.2.2 Ferramenta Utilizada

Como ferramenta principal para edição e inferência em ontologias OWL2, foi utilizado o UnBBayes (22). O UnBBayes, disponibilizado como um software livre escrito em linguagem Java, oferece uma GUI e API para raciocínio probabilístico e foi ultimamente refatorado pelo autor do presente documento para servir como um DSPL via plug-ins no formato JPF (90).

Para a realização do procedimento da Seção 6.2.1 no UnBBayes, três plug-ins foram utilizados para permitir respectivamente as seguintes operações:

1. realizar a edição da ontologia ;
2. realizar a inferência (atuar como a máquina de inferência);
3. automatizar a comunicação entre a DSPL e a máquina de inferência OWL2. Este último foi implementado como um novo plug-in na família de softwares.

Na a implementação desses três plug-ins, foi utilizado a OWL API versão 3 (105) e Protégé versão 4.1 (47) na manipulação dos elementos OWL2. A seção seguinte (Seção 6.3) também apresenta um diagrama do novo plug-in gerado.

6.3 Análise

Os artefatos gerados neste trabalho (i.e. componentes computacionais e ontologia FM) podem ser encontrados no repositório Subversion do projeto UnBBayes (15) como um projeto com identificador "`unbbayes.gui.mebn.ontology.protege`". O plug-in de PR-OWL 2, apresentado no Capítulo 5 foi utilizado para edição e inferência da ontologia FM no UnBBayes. A Figura 6.4 ilustra o plug-in que automatiza o envio de informações de configuração (e.g. plug-ins atualmente instalados) e eventos. Esse último plug-in também é responsável por obter o retorno e mostrar ao usuário um diálogo para confirmação de download.

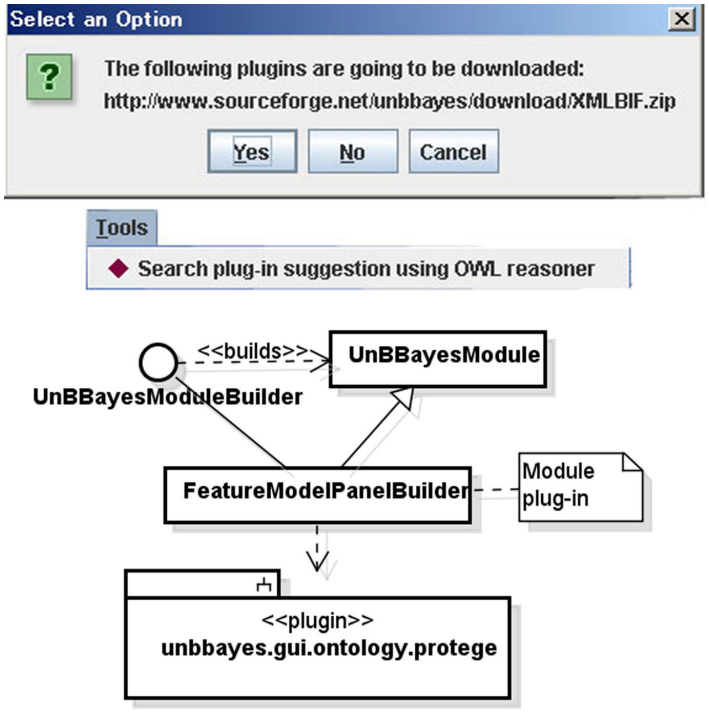

Figura 6.4: diagrama UML do plug-in que captura e envia a configuração e eventos à máquina de inferência (inferior) e captura de tela dessa funcionalidade (centro e superior).

Algumas das vantagens que poderam ser percebidas com a implementação do cenário de uso foram:

- por ser voltada à tecnologia da Web Semântica, OWL 2 oferece interoperabilidade (por ser um formato universal, interpretável por diversas ferramentas) e escalabilidade (complexidade polinomial na verificação de consistência);
- por ser uma tecnologia madura, poderam ser encontrados diferentes exemplos e implementações (a exemplo, (23) representa contextos que engatilham transição de configurações usando OWL);
- a grande expressividade implicou diretamente na sua grande extensibilidade, permitindo reuso de ontologias (i.e. *import*), modelagem do CK na mesma ontologia e modelagem de transição de configurações da DSPL na própria ontologia;
- a hipótese do mundo aberto permite tratamento natural de *features* imprevistas (e.g. plug-ins desenvolvidos por terceiros).

No entanto, as seguintes desvantagens também foram observadas no decorrer da modelagem de FM em OWL2:

- a alta expressividade mostrou dificuldade no entendimento correto das ontologias (pelo ser humano), causando dificuldades na depuração em eventuais erros;

- o uso da expressividade OWL 2 sem controle algum quebra requisitos de escalabilidade (causando inclusive que queries em OWL 2 tornem um problema 2NEXPTIME[55] em alguns casos);
- a hipótese do mundo aberto não é uma hipótese tão intuitiva, pois *features* inexistentes precisam ser explicitamente informadas como "inexistentes".

Pelo visto, OWL 2 oferece sintaxe e semântica formal para expressar FM, provendo também uma infraestrutura para fácil reuso e extensão. No âmbito de DSPL, OWL 2 mostrou que softwares podem manipular e raciocinar sobre seu próprio modelo/arquitetura, permitindo que eventos e transições de configuração possam ser tratados dinamicamente.

Um exemplo de implementação foi criado usando-se a ferramenta UnBBayes, um DSPL de código aberto que oferece uma arquitetura com plug-ins. Por prover um editor e máquina de inferência OWL2, um FM e transições especificadas em OWL 2 poderam ser consistentemente avaliados e interpretados pela própria DSPL, mostrando a grande utilidade em se especificar FM em OWL2.

A funcionalidade para efetivamente realizar a mudança de configuração no UnBBayes (i.e. download de plug-ins) não foi implementada ainda, unicamente por falta de uma infra-estrutura sólida para tal realização (e.g. presença de um repositório de plug-ins disponível na Internet). Essa funcionalidade poderia ser destacada como um possível trabalho futuro. A listagem 6.1 apresenta o descritor do novo plug-in gerado.

Listagem 6.1: Descritor do plug-in que preenche a ontologia de FM com configurações atuais do UnBBayes e retorna as sugestões inferidas pela máquina de inferência.

```
1  <?xml version="1.0" ?>
2  <!DOCTYPE plugin PUBLIC "-//JPF//Java Plug-in Manifest 1.0"
3  "http://jpf.sourceforge.net/plugin_1_0.dtd">
4  <plugin id="unbbayes.gui.featuremodel" version="0.0.1">
5      <requires>
6          <import plugin-id="unbbayes.util.extension.core"/>
7          <import plugin-id="unbbayes.gui.mebn.ontology.protege"
8          plugin-version="1.0.1"/>
9      </requires>
10
11      <extension plugin-id="unbbayes.util.extension.core"
12      point-id="Module" id="FM">
13          <parameter id="class"
14      value="unbbayes.gui.featuremodel.extension.FeatureModelPanelBuilder"/>
15          <parameter id="name"
16          value="Search plug-in suggestion using OWL reasoner" />
17      <parameter id="description"
18      value="Test of Feature Model using OWL" />
19      <parameter id="icon" value="individual.gif" />
20      <parameter id="builder"
21       value="unbbayes.gui.featuremodel.extension.FeatureModelPanelBuilder"/>
22      <parameter id="category" value="tool" />
23      </extension>
24 </plugin>
```

[55] Um problema 2NEXPTIME possui complexidade de tempo na ordem de 2^{2^n} em algoritmos não determinísticos. Portanto, são problemas mais complexos do que os NP-Completos.

6.3.1 Uma Visão Futura para Extensão Probabilística

Particularmente, como o UnBBayes é um framework para raciocínio probabilístico, a incorporação de regras probabilísticas em FM para realizar um mecanismo avançado de sugestão de *features* (plug-ins) poderia também ser destacado como um trabalho futuro.

Como existem incertezas oriundas da linguagem e do desconhecimento do usuário, o raciocínio probabilístico permitiria um mapeamento fino entre necessidades do usuário e artefatos computacionais. A exemplo, um conjunto de informações obtidas por um formulário podem se mapear probabilisticamente a um conjunto de requisitos, que por sua vez se mapeiam a um conjunto de *features*, que se mapeiam a um conjunto de artefatos.

Como o reuso de ontologias OWL 2 em ontologias PR-OWL 2 pode ser feita de forma imediata, bastando utilizar os mecanismos de importação oferecida pela própria linguagem OWL 2 para incluir as definições da PR-OWL 2 na ontologia modelada (vide Figura 6.5), o FM criado nesta pesquisa poderá ser facilmente estendida para um modelo probabilístico que vise, por exemplo, sugestão de *features* via padrão de uso também. Para uma ferramenta com resolução dinâmica de *features*, a possibilidade de se raciocinar sobre suas próprias configurações e realizar a transição de forma inteligente seria um grande diferencial.

Na Figura 6.5, foi modelado que a propriedade `hasSuggestion`, que sugere uma nova *feature* "`feat`" a uma configuração "`config`", depende probabilisticamente da propriedade `equals`[56] da *feature* `feat`. Em outras palavras, a probabilidade de uma *feature* ser sugerida para uma determinada configuração depende de quem é essa *feature* (isso pode parecer algo óbvio, mas precisa estar explícito em algum ponto). Adicionalmente, foram arrastadas as propriedades `hasSubdomain` e `hasDataSet` para que posteriormente se tornem dependências de `hasSuggestion` também. Graças ao mapeamento que o plug-in de PR-OWL 2 cria automaticamente entre as propriedades OWL 2 arrastadas e os nós gerados na tela, os fatos expressos (ou inferidos) em OWL 2 são automaticamente interpretados pela porção probabilística também.

6.4 Impactos da Refatoração do UnBBayes e Implementação da PR-OWL 2 no Mundo

Atualmente, reconhece-se que o UnBBayes, resultante da refatoração, é utilizado pela GMU e pela UnB em aulas e tutoriais relacionados a BN e suas extensões. No entanto, claramente a utilidade do novo UnBBayes não se restringe à área didática.

Esta seção apresenta então alguns trabalhos ou indicadores que exemplificam os resultados obtidos com a refatoração do UnBBayes e implementação da PR-OWL 2, no âmbito de reuso de componentes de software (via arquitetura de plug-ins) e de conhecimento (via ontologias PR-OWL 2), no Brasil e no mundo. Alguns trabalhos anteriores à refatoração são apresentados também para ilustrar situações reais no qual PR-OWL 2 ou plug-ins do UnBBayes são sugeridos.

[56] A propriedade `equals` foi modelada em OWL 2 para ser simétrica, transitiva e reflexiva; ou seja, representa literalmente a relação de "igualdade".

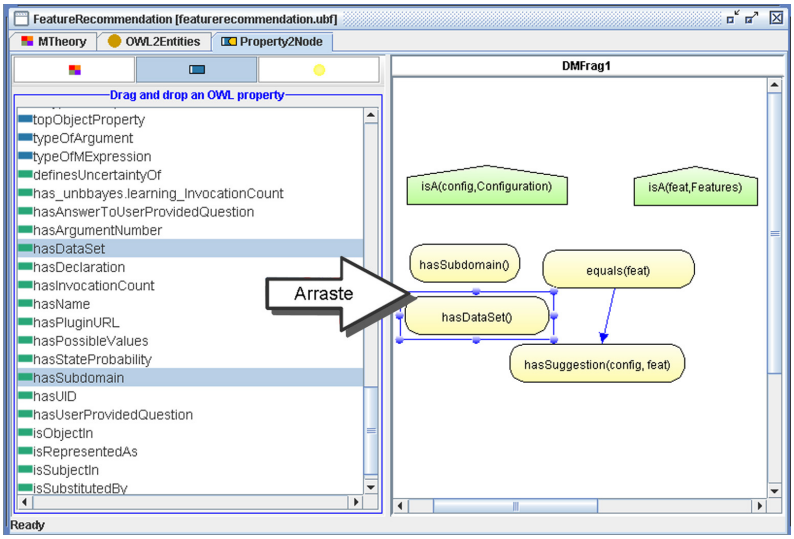

Figura 6.5: Com o UnBBayes e o plug-in para PR-OWL 2, converter uma propriedade OWL 2 a uma variável aleatória da PR-OWL 2 é simples - basta arrastar propriedades OWL 2 para um MFrag.

6.4.1 A Utilização do UnBBayes no Mundo

Uma das primeiras manifestações de uso do UnBBayes por terceiros ocorreu já em 2002 com o sistema FutSim, um jogo on-line de administração de futebol. Freire Valadares e Jeferson Luiz, estudantes da Universidade Federal de Pernambuco, adicionaram ao FutSim um suporte a raciocínio probabilístico baseado em BN (107) e para esse fim utilizaram classes do UnBBayes. É notável que o UnBBayes já não é um projeto de uso restrito à UnB, e podemos naturalmente imaginar que outros casos surgiram (ou surgirão) de forma similar. Alguns desses casos estão apresentados nesta seção.

Implementações que utilizavam o UnBBayes como API podem ser reproduzidas na nova arquitetura sem alterações em nível de código (exceto códigos que utilizavam algumas classes do UnBBayes que sofreram traduções de seus nomes para inglês). O trabalho de Charitha Dias, Marc Miska e Masao Kuwahara (40) ilustra um cenário que utiliza o algoritmo de aprendizagem K2 do UnBBayes (antes da refatoração) como API para atualizar distribuições de probabilidades condicionais em um modelo voltado para previsão de atrasos de veículos em estradas urbanas. Uma implementação similar pode ser perfeitamente reproduzida na arquitetura atual do UnBBayes, pois classes no *core* ou em plug-ins podem ser reutilizadas, como API, via mecanismos nativos do Java (*e.g.* instanciação de objetos, extensão de classes, sobreescrita de métodos). Todavia, diferente da arquitetura antiga, funcionalidades modeladas como plug-ins são optativas e podem ser descartadas caso desnecessárias. Essa possibilidade de podagem de funcionalidades simplifica o projeto e permite a criação de produtos similares com requisitos menores de espaço.

Um outro exemplo de ferramenta computacional que utiliza o UnBBayes como API é a versão Java do sistema AMPLIA[57] (**A**mbiente **M**ultiagente **P**robabi**L**ístico **I**nteligente de **A**prendizagem) (109) da UFRGS. O AMPLIA foi desenvolvido para auxílio na educação médica, no apoio ao desenvolvimento do raciocínio diagnóstico e na modelagem das hipóteses

[57] A versão Java do AMPLIA está hospedada em <http://code.google.com/p/amplia-java/>.

diagnósticas. Sua tradução em Java está utilizando classes do UnBBayes-MEBN (antes da refatoração) para implementar algumas funcionalidades de BN convencionais. Notavelmente, funcionalidades e bibliotecas do módulo de MEBN não são necessárias na a manipulação de BN convencional; portanto, há um pequeno desperdício de espaço.

Com a refatoração do UnBBayes a uma arquitetura de plug-ins, funcionalidades básicas de BN (utilizadas pelo AMPLIA) estão encapsuladas no *core* e funcionalidades específicas de MEBN foram migradas para plug-ins, que não precisam ser carregadas na classpath do AMPLIA por não serem *features* obrigatórias. Estima-se que aproximadamente 10 MB[58] de espaço seria liberado do projeto AMPLIA como resultado da migração para a nova versão do UnBBayes.

Pesquisadores como Miska, Muller e Van Zuylen sugeriram em (85) o uso de modelos macroscópicos (comportamento de motoristas em veículos) para se calibrar modelos de simulação microscópia on-line. Nesse trabalho, o módulo de aprendizagem do UnBBayes foi utilizado para a criação de um modelo probabilístico para auxílio na tomada de decisões sobre a velocidade de um veículo. Uma série de transformações foram aplicadas a esse modelo probabilístico para que sirva, no final, como dados de entrada ao simulador microscópico *MiOS* (84). Estima-se que tais transformações e o repasse de dados ao *MiOS* também possam ser realizados de forma automatizada e em lotes, implementando-se novos plug-ins no UnBBayes. Tal implementação (do repasse automatizado) seria difícil e propensa a erros no UnBBayes antigo, por exigir abertura da fonte e alterações em nível de linhas de código.

A arquitetura de plug-ins também permite que o usuário execute duas variações (customizações) do UnBBayes em um ambiente mais integrado, pois todas as variações são agora implementadas como plug-ins, que executam sobre uma mesma instância do framework.

A exemplo, o sistema "Lupa Digital" (62), fruto da cooperação entre a academia e o Departamento de Polícia Federal (DPF/INI) para aplicar mineração de dados em impressões digitais para agilizar a identificação criminal, utilizou tanto o UnBBayes (funcionalidades de aprendizagem bayesiana) quanto o UnBMiner (variação do UnBBayes para mineração de dados) na extração de conhecimento e avaliação comparativa. Nessa época (*i.e.* antes da refatoração), tais variações executavam como duas aplicações Java diferentes; portanto, sua execução simultânea exigia memória adicional. A execução alternada ou sequencial (*e.g.* iniciar o UnBBayes, terminar o UnBBayes e iniciar o UnBMiner) também exigia um tempo de carregamento ligeiramente maior, pois objetos ou classes que eram virtualmente "iguais" precisavam ser carregadas duas vezes. A nova arquitetura facilita o usuário na execução de diferentes variações do UnBBayes, pois provê uma interface unificada com reuso de classes e objetos. Isso resulta na redução do requisito de memória e tempo de carregamento, pois evita o carregamento duplicado.

No âmbito de MEBN, Saleha Raza e Sajjad Haider publicaram em 2010 um comparativo entre extensões bayesianas com expressividade de FOL (97) e utilizaram o plug-in UnBBayes-MEBN para um teste com MEBN. Apesar do foco estar mais no formalismo do que em implementações, este trabalho exemplifica o reconhecimento do UnBBayes-MEBN como um representante da lógica MEBN e revela sua crescente responsabilidade na a aceitação do formalismo MEBN.

Daniela Ballari, da Universidade de *Wageningen* (Holanda) está trabalhando em seu PhD no projeto *Dynamic interoperability of Wireless Sensor Network: context modeling approach based on metadata elements* e manifestou interesse particular no UnBBayes-MEBN para modelagem

[58] O valor de 10 MB foi obtido somando-se o tamanho dos artefatos referentes a MEBN, Protégé 3 e PowerLoom e subtraindo-se o tamanho dos artefatos referentes ao JPF.

de ontologias probabilísticas voltadas à interoperabilidade em rede de sensores. Integração de informações oriundas de sensores são casos de uso bastante comuns em MEBN (32, 71) e demandas por implementações nunca faltarão. A nova arquitetura do UnBBayes e o plug-in UnBBayes-MEBN poderão oferecer uma infra-estrutura adequada para customizações rápidas e simples.

Por oferecer um ambiente dinâmico baseado em plug-ins, o UnBBayes foi também adotado com sucesso pelo projeto ***PR**obabilistic **O**ntolo**G**ies for **N**etcentric **O**perational **S**ystems* (*PROGNOS*[59]) (14, 32) na implementação de *scripts* de probabilidades condicionais (*Conditional Probability Scripts* - CPS), um meio para se especificar a distribuição de probabilidades condicionais em uma BN utilizando-se uma *script* especial. Observou-se então um avanço substancial na reusabilidade e variabilidade do UnBBayes, comprovada pela equipe do projeto *PROGNOS*.

Não surpreendentemente, diversos doutorandos da *George Mason University* também estão de pouco em pouco adotando o UnBBayes-MEBN e sua arquitetura de plug-ins para avançarem nas suas pesquisas. De forma similar, alguns projetos de pesquisa da *George Mason* também estão passando a manifestar interesses especiais pelo UnBBayes. Por exemplo, o projeto "*ACE-Decomposition Based Information Elicitation & Aggregation*" recebeu recentemente investimentos da *Intelligence Advanced Research Projects Activity* (IARPA) e tem demonstrado interesse em usar o UnBBayes-MEBN e sua arquitetura de plug-ins. Estes exemplos sugerem um futuro aumento na demanda pelo UnBBayes.

Corporações também têm mostrado interesse no UnBBayes e nas implementações de PR-OWL. A *EADS INNOVATION WORK*[60], por exemplo, convidou o Dr. Paulo Costa da *George Mason* para apresentar um curso sobre UnBBayes e MEBN/PR-OWL na França. Isso é um forte indício da crescente atenção que o UnBBayes tem recebido no mercado corporativo.

6.4.2 Outros Indicadores de Utilidade do UnBBayes

Como comentado nos capítulos anteriores (ex. Seção 1.3.1), o UnBBayes é uma ferramenta disponibilizada no sítio do SourceForge. Por conveniência, esse sítio oferece um serviço que estima a posição (*ranking*) de um projeto em relação a outros. Surpreendentemente, após atualizada a arquitetura de plug-ins[61], o UnBBayes obteve a quadragésima terceira posição, uma posição certamente excepcional para uma ferramenta de IA - um nicho relativamente estreito - hospedada em um sítio junto a milhares de softwares de uso geral.

Esse fato ilustra a enorme atenção que o UnBBayes tem recebido no mundo e a crescente importância da manutenibilidade, que pode ser oferecida pela arquitetura de plug-ins. Com a crescente atenção na comunidade, ocorrerá um aumento proporcional na demanda por novas extensões, que poderão frequentemente originar de terceiros. Uma arquitetura que oferece a possibilidade de extensões sem a necessidade de acesso ao código fonte possibilitará que terceiros customizem a ferramenta de acordo com suas necessidades, de forma prática, modular e segura, até mesmo sem a necessidade de contato com a equipe do UnBBayes[62].

[59] Página do projeto *PROGNOS*: <http://sourceforge.net/projects/prognos>.

[60] http://www.eads.com/

[61] Vide *ranking* em <http://sourceforge.net/project/stats/?group_id=47519&ugn=unbbayes> na data 16/06/2011.

[62] A extensão do UnBBayes sem uma notificação qualquer pode ser algo lamentável para nossa equipe (por estarmos deixando de obter informações preciosas sobre casos de uso da ferramenta), mas também reconhecemos que é um fator que agiliza bastante a evolução de softwares livres na comunidade.

Conforme comentado anteriormente, o UnBBayes tem recebido atenção de corporações norte-americanas também. No final de 2007, a Harris[63], companhia de TI voltado ao mercado público (governo americano) e comercial, presente em mais de 150 países com mais de 16 mil funcionários e com uma receita de mais de 6 bilhões de dólares anuais, contatou um dos membros da equipe do UnBBayes solicitando informações e sugerindo possíveis parcerias. Este fato evidencia a grande atenção que o UnBBayes recebe de entidades estrangeiras, mostrando que melhorias na manutenibilidade do UnBBayes, oferecidas pela refatoração, é um interesse internacional.

Um outro indicativo que sugere o crescente valor que o mundo tem dado ao UnBBayes é o custo total que o sítio *ohloh*[64] estimou para o projeto UnBBayes[65]. Esse sítio avaliou nosso projeto com um valor total de $17.633.194 (em dólar americano), esforço estimado de 321 pessoas/ano e com qualidade de código "bem estabelecido e maduro" (dados acessados em 17/06/2011). Portanto, pode-se dizer que o impacto financeiro potencial com a melhoria na manutenibilidade, oferecida pela infra-estrutura de plug-ins, é também na ordem de milhões de dólares!

6.4.3 A Utilização da PR-OWL 2 no Mundo

As utilizações pioneiras de PR-OWL/MEBN no UnBBayes por terceiros tiveram origem nos próprios criadores de PR-OWL/MEBN, em (69) e posteriormente em (66). Nesses trabalhos, o UnBBayes-MEBN (ainda em versão primordial) foi referido como GUI e máquina de inferência que facilita na manipulação de ontologias probabilísticas baseadas no framework PR-OWL/MEBN. Após tal publicação, diversas pesquisas foram realizadas e novas necessidades e limitações foram descobertas, motivando o surgimento da PR-OWL 2 e sua implementação no UnBBayes. Esta seção apresenta então algumas utilizações da linguagem PR-OWL 2 e/ou do plug-in de PR-OWL 2 na comunidade, para servir de indicador da utilidade do trabalho realizado nesta pesquisa de mestrado.

Pesquisadores da MITRE[66], uma corporação que trabalha em interesses públicos (governo americano) e possui mais de 7 mil pesquisadores para oferecer soluções em áreas de TI e modernização de empresas, também têm entrado em contato ativamente com Rommel Novaes Carvalho, criador da linguagem PR-OWL 2, para discutir sobre soluções em Web Semântica e ontologias probabilísticas e têm utilizado bastante o novo UnBBayes-MEBN (e eventualmente alguns conceitos de PR-OWL 2) em seus projetos de doutorado.

Em 2010, a *Institute of Business Administration* de Karachi (Paquistão) e a *George Mason University* têm realizado uma videoconferência; levantando, dentre outros, assuntos como: o processo genérico de modelagem de ontologias probabilísticas, o uso das funcionalidades de UnBBayes-MEBN (após refatoração como plug-in) para a modelagem, a necessidade de mapeamento de propriedades OWL a variáveis aleatórias (uma novidade da PR-OWL 2), futuras implementações computacionais (plug-in do UnBBayes para PR-OWL 2) e alguns outros formalismos alternativos suportados pela ferramenta (*e.g.* OOBN). Essa videoconferência foi

[63] www.harris.com

[64] O ohloh é um serviço que oferece um diretório público gratúito sobre softwares de código aberto e seus desenvolvedores. Sua página principal é <http://www.ohloh.net/>.

[65] Vide sumário de avaliação do UnBBayes pela ohloh em <http://www.ohloh.net/p/unbbayes>.

[66] Página da MITRE: <www.mitre.org>.

gravada e já recebeu mais de 200 acessos desde sua publicação no YouTube[67], serviço de compartilhamento de vídeos. Considerando-se que o assunto está em estado da arte, essa quantidade de reproduções é algo surpreendente. E-mails de pesquisadores e estudantes que ficaram interessados no assunto devido a esse vídeo têm chegado com bastante frequência.

O Instituto Tecnológico de Aeronáutica também está presente na lista de usuários do UnBBayes e PR-OWL. O trabalho em (75) utiliza o novo UnBBayes-MEBN para a modelagem de ontologias probabilísticas voltadas ao auxílio na tomada de decisões (curso de ação) da força aérea (75), e para isso levanta alguns conceitos da PR-OWL 2 também. É um exemplo interessante de aplicação de ontologias probabilísticas em domínios realistas.

Robert Schrag da *Global InfoTek, Inc.* (GITI[68]), integrante de um programa da DARPA (*Defense Advanced Research Projects Agency*), também manifestou interesse em PR-OWL 2 e tem entrado frequentemente em contato com o Rommel Novaes Carvalho da *George Mason University* a respeito de novidades sobre representações e raciocínios em PR-OWL 2.

Um exemplo relativamente mais completo que utiliza a PR-OWL 2 e sua implementação no UnBBayes está para ser publicado em (12). A PR-OWL 2 é utilizada basicamente na integração probabilística de informações coletadas de bases/ontologias distintas oriundas de entidades públicas do Brasil (*e.g.* Receita, Polícia Federal, DENATRAN, CGU) para detecção de fraudes na licitação pública. Este é o exemplo mais claro e atual do uso da PR-OWL 2 e sua implementação no UnBBayes.

[67] Sítio da videoconferência no YouTube: <http://www.youtube.com/watch?v=e8NabmtbFNc>.
[68] http://www.globalinfotek.com/

Capítulo 7

Estudo de Caso: Detecção de Fraudes em Licitações Públicas

Como já comentado nos capítulos anteriores, a crescente massa de informações exigidas em aplicações reais torna impeditiva a fusão de "dados" como uma proposta tangível. Uma fusão em nível de "conhecimento" precisa ser realizada.

Linhas de pesquisas que trabalham com fusão de conhecimentos locais, oriundos de múltiplas fontes, para a geração de um conhecimento global compartilhado vêm obtendo crescente atenção. Por exemplo, domínios militares buscam trabalhar com compartilhamento de informações para a disseminação - de preferência em tempo-real - de informação relativa à situação operacional. Nesses contextos, ontologias probabilísticas podem ser úteis na representação de conhecimento "global", resultante da fusão de conhecimentos locais (27, 69).

Por outro lado, um outro exemplo de domínio que necessita dessa visão mais "global", voltada na tomada de decisões a partir da fusão de informações em múltiplas fontes, é o domínio de detecção de fraudes em licitações públicas da CGU. Nesse domínio, os auditores precisam rotineiramente lidar com uma quantidade enorme de informações (oriunda de fontes diversas), identificar os aspectos mais críticos e então reportar aos responsáveis pela tomada de decisão, para avaliar se medidas adicionais - como investigação - precisam ser adotadas. É fácil de perceber que esta tarefa está longe de ser algo trivial.

Como a principal responsabilidade da CGU é prevenir e detectar corrupção do governo, o órgão tem adotado uma série de ações relacionadas a tal missão, que por sua vez resultaram em um grande repositório de dados. Tais ações incluem: campanhas de conscientização destinadas ao setor privado; campanhas para a educação do público; iniciativas de pesquisa; inspeções e auditorias regulares de municípios e estados.

Apesar da CGU ter recolhido uma massa significativa de dados oriundos de centenas de fontes (*e.g.* Receita, MEC, DENATRAN e Polícia Federal), o processo de fusão desses dados não fora tão eficiente e nem suficiente para atender às necessidades reais na tomada de decisões. Portanto, o foco que antes era na "fusão de dados" precisava ser deslocado para a "fusão de conhecimento(s)". Como conseqüência, métodos tradicionais - baseadas em sintaxe - precisavam ser melhoradas via técnicas de representação e raciocínio com semânticas.

Como pode se imaginar, a incerteza é onipresente na fusão do conhecimento. Ainda; a incerteza é especialmente importante no domínio de fraudes, pois autores de fraudes procuram ocultar atividades e intenções ilícitas, tornando difícil realizar afirmações nítidas sobre qualquer

hipótese. Em outras palavras, as informações parciais (incompletas) ou aproximadas (inexatas) são mais regras do que exceções nesse domínio.

Em suma, este capítulo apresenta um exemplo de uso da PR-OWL 2 na fusão de conhecimentos determinísticos locais (*i.e.* ontologias que descrevem dados oriundos de órgãos como a CGU, Receita e Polícia Federal) para a representação de uma ontologia probabilística que oferece uma visão global necessária para a detecção de fraudes em licitações públicas, considerando-se o reuso das porções determinísticas. Esta ontologia é fruto de trabalho em equipe do mestrando com o Rommel Novaes Carvalho (12), que gentilmente permitiu a incorporação nesta dissertação.

Vale notar que trabalhos em ontologias probabilísticas PR-OWL antes da formulação da PR-OWL 2 focavam apenas na porção probabilística, e nunca explicavam a forma em que a porção determinista - semântica OWL - era utilizada (28, 29, 65, 69–71). Naturalmente, isso não seria um problema caso a ontologia seja criada "a partir do zero" e/ou composta integralmente por conhecimento probabilístico, que pode perfeitamente representar conhecimentos determinísticos atribuindo-se somente probabilidades 0% ou 100% em variáveis aleatórias. Contudo, em caso de existência de ontologias determinísticas "a se basear" (o que se espera ser algo comum, visto a grande aceitação da OWL pela comunidade e a quantidade de ontologias OWL já criadas), as abordagens antes da PR-OWL 2 perderiam em produtividade, pois demonstrariam dificuldades no reuso sistemático das ontologias de base.

A Seção 7.1 apresenta a visão geral do domínio de licitação pública, o processo adotado pela CGU na detecção de fraudes e uma modelagem inicial da ontologia probabilística para auxílio a tal processo. A Seção 7.2 descreve como foi modelada a fusão de conhecimento nesse domínio. Por fim, a Seção 7.3 apresenta e analisa os resultados obtidos pela inferência na ontologia gerada.

Observação: as ontologias apresentadas neste capítulo foram criadas pelos autores (12), inclusive pelo presente mestrando. Entretanto, a fim de ilustrar a idéia de fusão de informações, estamos apresentando como sendo ontologias criadas e distribuídas por diferentes agências do Governo Brasileiro.

7.1 O Domínio de Licitação Pública

Sabe-se que uma das principais fontes de corrupção no governo brasileiro está no processo de licitação pública. Embora leis tentem garantir um processo competitivo justo, autores de fraudes conseguem burlar o processo em vantagem própria, de maneira legítima em primeira vista. Apesar da diversidade, fraudes podem ser muitas vezes categorizados por critérios conhecidos. Esse é um dos motivos que impulsionam os especialistas - auditores - a utilizarem modelos que descrevam tais critérios, com base nas ocorrências já tratadas pela CGU.

Alguns critérios fortes já conhecidos na identificação de tipos de fraudes são: presença de pessoas que trabalham como uma fachada para uma empresa (esquema conhecido como funcionários/proprietários "laranjas"), presença de índices contábeis incomuns, e outros. Indicadores foram criados para auxiliar na identificação desses casos. Por exemplo, um princípio que deve ser estritamente seguido nos contratos públicos é o da competição justa. Todos os contratos públicos devem estabelecer requisitos mínimos necessários para se garantir a execução do contrato, maximizando o número de licitantes. No entanto, é comum a existência de competições falsas, quando diferentes licitantes são, na verdade, propriedades de um mesmo indivíduo. Isso

é possível nominando-se alguém como um falso proprietário (“laranja”), que frequentemente são pessoas com pouca ou nenhuma educação.

O principal objetivo deste estudo de caso foi estruturar o conhecimento de um especialista no domínio, possibilitando que um sistema automatizado possa, dado um conjunto de evidências, inferir conclusões de forma similar a do especialista. É importante observar que estes tipos de sistemas (sistemas especialistas) se destinam ao apoio ou treinamento de novos especialistas, nunca para a substituição de pessoas por programas.

Inicialmente, alguns critérios simples foram selecionados como prova de conceito. No entanto, pode-se mostrar (vide Seção 7.2) que o modelo permite incorporação progressiva de novos critérios. No decorrer da modelagem, tornou-se evidente que diferentes fontes precisavam ser consultadas para se obter indicadores necessários para a criação de um conhecimento atualizado e realmente útil para os tomadores de decisão.

A Figura 7.1 apresenta uma visão geral do processo previsto para a detecção de fraudes em licitações. “Dados” representam os vários pedidos de proposta e leilões que são emitidos pela Secretaria Federais, Estadual e Municipal.

Figura 7.1: Visão geral do ciclo de detecção/prevenção de fraudes em licitações públicas.

Como o foco deste estudo de caso não foi o tratamento de dados brutos, o passo de “coleta de informação” prevê auxílio de analistas. A idéia é que os analistas da CGU, já trabalhando em auditorias e inspeções, contribuam para a tal coleta respondendo questionários especialmente criados para a coleta de indicadores dos critérios selecionados. Tais questionários podem ser gerados por um sistema já em produção na CGU. Respondidos os questionários, as informações necessárias precisam ser disponibilizadas (passo “Banco de dados/informações”).

Posteriormente, a ferramenta UnBBayes, usando-se uma ontologia probabilística modelada por peritos nessa tarefa (passo “Design - UnBBayes”), será capaz de integrar milhões de itens de informação e transformá-los em dezenas ou centenas de itens de conhecimento, graças a lógica e inferência probabilística. Por exemplo, informações como anúncios de compras, contratos e relatórios podem ser analisadas para se identificar relações e propriedades relevantes, que

podem ser utilizadas para se tirar algumas conclusões sobre possíveis irregularidades (passo "Inferência / conhecimento").

Conclusões inferidas podem ser filtradas para que apenas as licitações que mostrem probabilidades superiores a um limiar - *e.g.* 50% - sejam automaticamente encaminhadas para o departamento responsável, juntamente com relatórios sobre a inferência realizada, tipo de fraudes e suas evidências (passo "Relatórios").

Os critérios selecionados pelo especialista foram o uso de índices contábeis e a exigência de experiência em apenas um contrato. Segundo o especialista, existem quatro tipos comuns de índices que são geralmente utilizados como requisitos nas compras: ILC, ILG, ISG, e IE. Qualquer outro tipo pode ser um índice criado especialmente para direcionar uma licitação a alguma empresa em particular. Quanto maior o número de índices contábeis incomuns na licitação, mais suspeito é. Além disso, licitações normalmente especificam um valor mínimo para tais índices. O valor mínimo exigido é normalmente 1,0; mas quanto maior o valor desse mínimo, mais estreito é a competição e maior é a chance da licitação estar sendo direcionada para alguma empresa.

O outro critério que exige prova de experiência em apenas um contrato torna a licitação suspeitosa, porque é raro os casos em que experiências são obtidas apenas por um contrato em particular, pois pode-se também fazê-los repetidas vezes em diferentes contratos.

Como base para a implementação das regras probabilísticas descritas acima, utilizamos definições presentes na ontologia OWL ilustrada na Figura 7.2, que assumimos estar disponível na URI <http:// www.cgu.gov.br/ontologies/ProcurementDomain.owl> para representar conhecimento oriundo da fonte CGU, antes da fusão. Como a modelagem de ontologias determinísticas OWL não faz parte do nosso foco, detalhes sobre sua criação foi omitida neste documento.

Usando-se o plug-in para PR-OWL 2, previamente apresentado na Seção 5.3, foi possível criar RV que representam as regras probabilísticas descritas nos parágrafos anteriores simplesmente arrastando as propriedades OWL da Figura 7.2 para a tela de edição de MFrags do UnBBayes. As regras foram implementadas em três diferentes MFrags.

O primeiro MFrag, ilustrado na Figura 7.3, apresenta critérios exigidos a uma empresa para participar em uma licitação. Este MFrag contém informações sobre os tipos de índice - definida pela RV `hasIndexType(index)`, cujos valores possíveis são indivíduos da classe `IndexValueRange` - e seus valores mínimos - definida pela RV `hasMinIndexValue(index)`, cujos valores possíveis são indivíduos da classe `AccountingIndexType`. Este MFrag também contém informações sobre o tipo de requisito exigido pela licitação - `demandsRequirement (procurement, requirement)`, RV de tipo booleano - e se a licitação exige experiência em apenas um contrato - RV `acquiredInOneContract(experience)`, também de tipo booleano.

Ambas classes `IndexValueRange` e `AccountingIndexType` são nominais definidas na OWL. A primeira possui `ILC`, `ILG`, `ISG`, `IE` e `other` como indivíduos possíveis. A segunda possui `between0And1`, `between1And2`, `between2And3` e `greaterThan3` como indivíduos possíveis.

O segundo Mrag, ilustrado na Figura 7.4, representa se a licitação está sendo direcionada a uma empresa específica via índices incomuns - RV `isDirectedByIndexes(procurement)`, de tipo booleano. Como explicado anteriormente, esta análise é baseada no tipo de índice e seu valor mínimo exigido - representados respectivamente pelos nós de entrada `hasIndexType(index)` e `hasMinIndexValue(index)`.

Este MFrag considera apenas os índices exigidos como requisitos em uma licitação específica, graças ao nó de contexto `demandsRequirement(procurement, index)`. Observe que essa RV é definida no MFrag da Figura 7.3 como `demandsRequirement(procurement,`

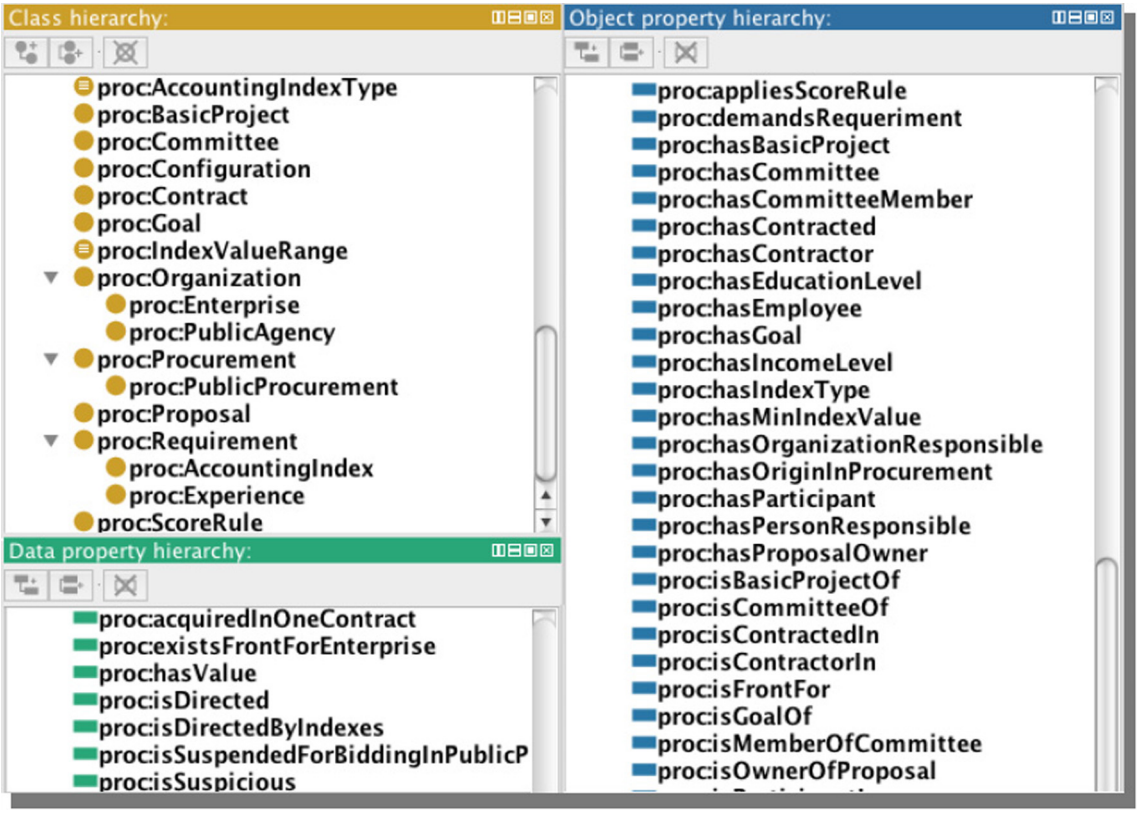

Figura 7.2: Principais classes e propriedades OWL do domínio “global”.

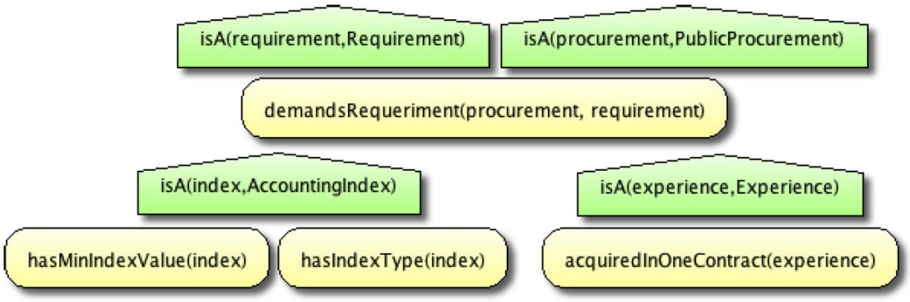

Figura 7.3: MFrag de requisitos.

requirement), cujo segundo argumento é do tipo Requirement. No entanto, na Figura 7.4 o segundo argumento é um AccountingIndex. Este é uma nova funcionalidade oferecida pelo plug-in de PR-OWL 2, que permite o uso de subtipos na ontologia probabilística. Na Figura 7.2, a classe AccountingIndex é definida como subclasse de Requirement, e esta semântica é herdada pela PR-OWL 2.

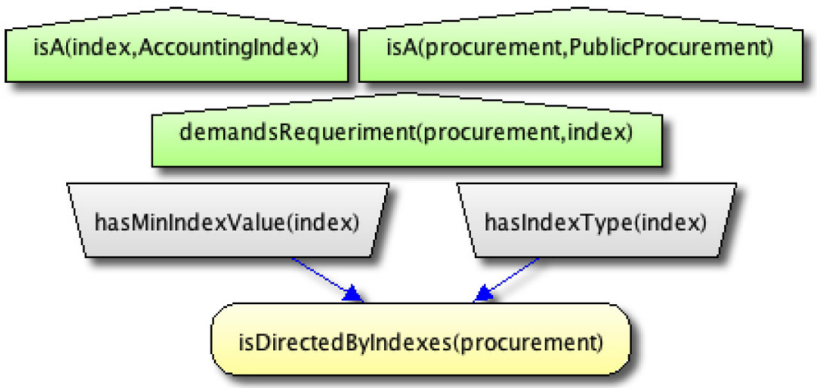

Figura 7.4: MFrag de direcionamento de licitações por índices contábeis.

O último MFrag, Figura 7.5, representa a possibilidade geral da licitação estar sendo encaminhada a uma empresa específica - definida pela RV isDirected(procurement), de tipo booleano - com base nos valores obtidos pelo uso de índices incomuns - nó de entrada isDirectedByIndexes(procurement) - e exigência de experiência em apenas um contrato - nó de entrada acquiredInOneContract(experience). Observe que estes nós também fazem uso de subtipagem, ao considerar apenas as experiências exigidas como requisitos para esta licitação específica (na ontologia OWL da Figura 7.2, a classe Experience é definida como subclasse de Requirement).

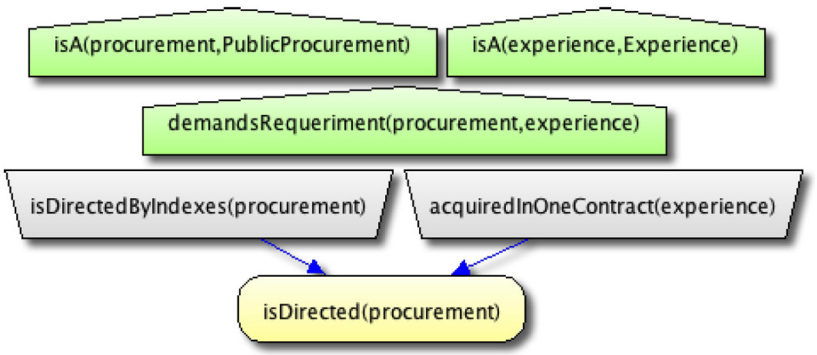

Figura 7.5: MFrag de direcionamento de licitações.

7.2 Fusão de Conhecimento Via Ontologias Probabilísticas

Pelos critérios apresentados e modelados na seção anterior, percebe-se claramente a necessidade de princípios para lidar com a incerteza. Todavia, qual seria o papel da Web Semântica nesse domínio?

A CGU utiliza dados oriundos não só de suas auditorias ou inspeções. De fato, muitas informações complementares podem ser recuperadas de outras agências federais, como a Receita Federal, Polícia Federal, MEC, DENATRAN e outros. Imagine uma situação em que informações pessoais (*e.g.* renda anual) do(s) proprietário(s) de uma empresa vencedora de uma licitação são requisitadas. Esses tipos de informações geralmente não estão disponíveis no banco de dados da CGU, portanto devem ser recuperadas a partir da Receita Federal. Ao se verificar as informações sobre o(s) proprietário(s), pode ser interessante também verificar o histórico criminal. Para esse fim (vide Figura 7.6), informações da Polícia Federal precisam ser usadas.

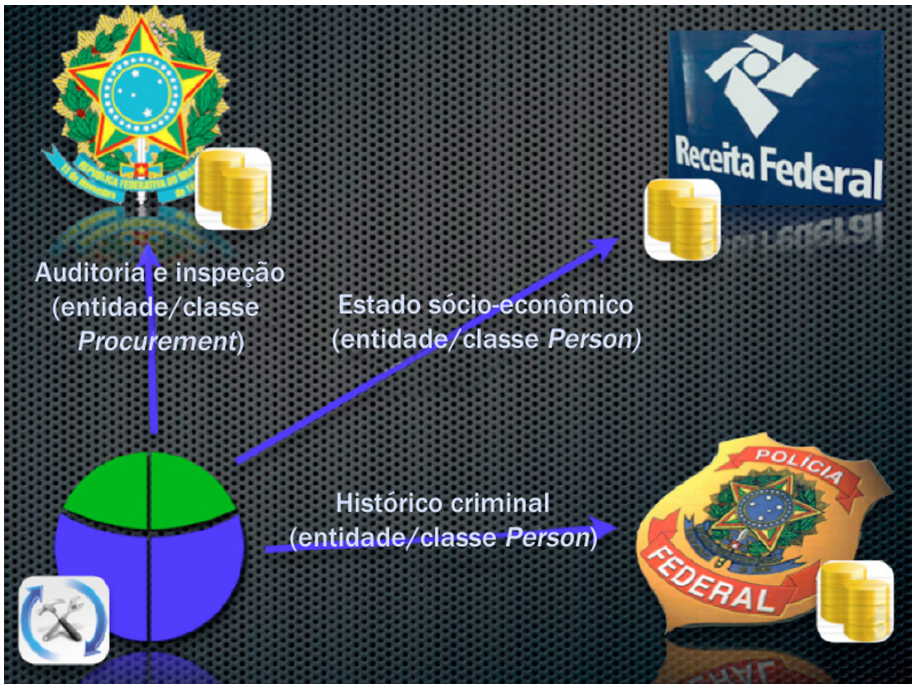

Figura 7.6: Esquema de fusão de conhecimento oriundo de diferentes órgãos do governo.

Note que neste cenário temos diferentes fontes realizando afirmações diferentes sobre um mesmo indivíduo. Adicionalmente, podem existir outras agências com informações cruciais relacionadas com o indivíduo de interesse. Estamos portanto operando em um mundo aberto. Finalmente, a fim de possiblitar o compartilhamento dessas informações de maneira consistente, precisamos ter a certeza de que estamos tratando sobre um mesmo indivíduo; que pode, especialmente em caso de fraudes, utilizar nomes diferentes em contextos variados.

Essa visão principal - fusão de conhecimento oriundo de diferentes fontes - foi considerada como a base para se incluir uma nova regra na inferência probabilística: verificar se uma pessoa é fachada de alguma empresa. Neste país, é comum o uso de pessoas com pouca (ou nenhuma) educação como fachadas (sócios "laranjas") da empresa. Normalmente, tais pessoas não pos-

suem propriedades muito valiosas (como automóveis ou casas) e apresentam baixa renda anual. Frequentemente são jardineiros ou empregadas domésticas que trabalham para a pessoa que realmente toma as decisões para a empresa.

Então, visto o nível de educação, renda anual e posse de propriedades (*e.g.* se a pessoa possui um automóvel), podemos identificar se essa pessoa é mais provável de ser fachada ou não. Infelizmente, a CGU não possui informações sobre escolaridade, renda anual ou propriedades de uma pessoa. Tais informações estão disponíveis, mas em outras agências federais: a escolaridade pode ser obtida do MEC; a renda anual pode ser obtida da Receita e a posse de veículos (automóveis) pode ser obtida do DENATRAN.

A CGU tem se envolvido em colaborações com diferentes órgãos por alguns anos, a fim de recolher mais informações que ajudem a identificar e previnir fraudes em licitações públicas. Esta seção apresenta então a maneira em que a CGU pode se beneficiar das tecnologias da Web Semântica para se adicionar novas regras na ontologia probabilística criada na Seção 7.1 e realizar inferências a partir de informações fornecidas por outrem.

No modelo aqui apresentado, assume-se que cada órgão possui sua própria ontologia, de visão local e focada no seu próprio domínio de aplicação. Assume-se também que todos os órgãos governamentais utilizam uma ontologia comum com os conceitos básicos sobre pessoas (nome, endereço, relacionamento, etc.), uma ontologia criada pelo Governo Federal disponível em <http:// www.brasil.gov.br/ ontologies/ People.owl>. Assumimos também que o MEC fornece uma ontologia em <http:// www.mec.gov.br/ ontologies/ Education.owl> sobre educação; o DENATRAN fornece uma ontologia sobre veículos (*e.g.* posse e licença) em <http:// www.denatran.gov.br/ ontologias/ MotorVehicle.owl> e a Receita fornece uma ontologia sobre seus serviços em <http:// receita.fazenda.gov.br / ontologias / InternalRevenue.owl>. Como os conhecimentos estão representados em OWL, é possível incorporar todos eles na ontologia probabilística usando-se mecanismos nativos de importação da OWL (*i.e. import*).

Após a importação, pôde-se modelar novos MFrags representando regras probabilísticas sobre a identificação de "laranjas". Foram criados quatro novos MFrags e dois MFrags adicionais para melhor integração com as regras probabilísticas anteriores, que identificavam se uma licitação era direcionada a alguma empresa em particular (vide Seção 7.1).

O MFrag da Figura 7.7 possui informações associadas a uma empresa. Neste MFrag, só foi especificada a RV `isResponsibleForOrganization(person, enterprise)`, que indica se um pessoa é responsável por uma empresa. Apesar de que na ontologia determinística importada a propriedade `isResponsibleForOrganization` - propriedade ligada à RV `isResponsibleForOrganization(person, enterprise)` via mapeamento PR-OWL 2[69] - tenha a classe `Organization` como imagem, no MFrag a subclasse `Enterprise` foi utilizada, pois a identificação de fraudes somente precisará lidar com empresas. Isto é um novo recurso oferecido pelo plugin de PR-OWL 2.

A Figura 7.8 modela a associação entre uma licitação e uma empresa através da RV `hasParticipant(procurement, enterprise)`, que indica se a empresa particupa de uma licitação. A subclasse `Enterprise` também é usada no lugar do tipo geral especificado na ontologia OWL.

A Figura 7.9 define a RV `isFrontFor(person, enterprise)`, que indica se uma pessoa é uma fachada para uma empresa. Aqui também é utilizada a subclasse `Enterprise` no lugar da classe `Organization`.

[69]O termo "mapeamento PR-OWL 2" neste capítulo denota o uso da propriedade `definesUncertaintyOf`, introduzida em PR-OWL 2, para ligar uma RV a propriedades OWL.

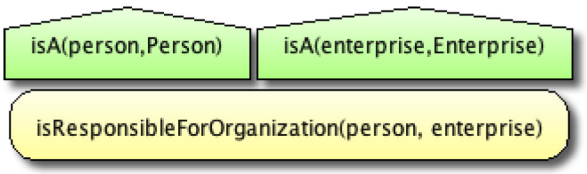

Figura 7.7: MFrag associado a informações sobre empresas.

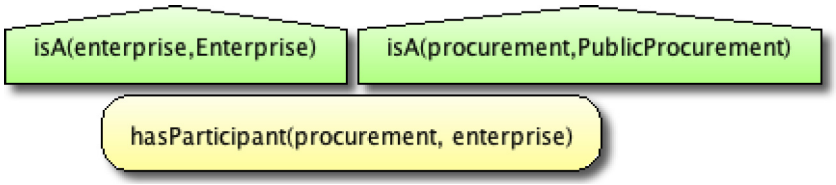

Figura 7.8: MFrag associado a informações sobre participação em licitações.

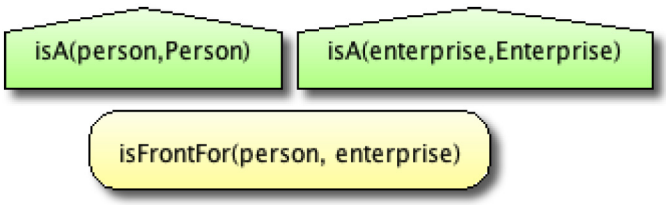

Figura 7.9: MFrag associado a informações sobre esquemas de fachada.

A Figura 7.10 apresenta a regra principal que determina se uma pessoa é "laranja": é mais provável que a renda anual seja baixa (vide RV `hasIncomeLevel(person)`), não possui automóvel (vide RV `hasMotorVehicle(person)`) e apresenta níveis baixos em educação (vide RV `hasEducationLevel(person)`). Estas RV estão ligadas (também via mapeamento PR-OWL 2) a propriedades OWL das ontologias importadas respectivamente da Receita, DENATRAN e MEC. Observe que o nó de contexto `isResponsibleForOrganization(person, enterprise)` indica que esta regra principal somente se aplica aos responsáveis pela empresa em questão.

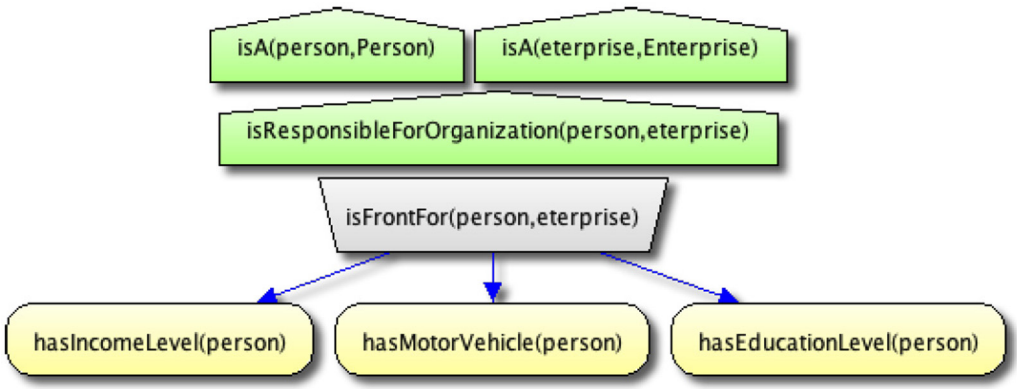

Figura 7.10: MFrag sobre informações pessoais e renda.

A Figura 7.11 apresenta uma RV criada para se simular um quantificador existencial (da FOL) em uma RV. Essa funcionalidade já está embutida na PR-OWL 2, mas sua semântica ainda não está implementada no UnBBayes. Por causa dessa limitação, a RV `existsFrontForEnterprise(enterprise)` precisou ser definida manualmente. De forma resumida, essa RV indica que se existir pelo menos uma pessoa atuando como "laranja" em uma empresa, então essa empresa possui "laranjas". Observe que as únicas pessoas analisadas são os responsáveis por essa empresa - vide nó de contexto `isResponsibleForOrganization(person, enterprise)`.

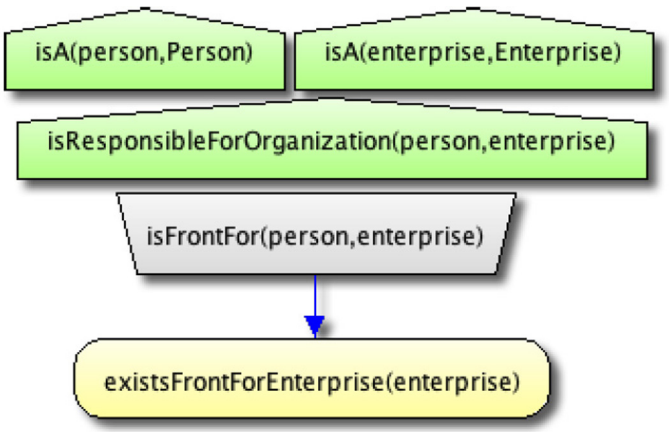

Figura 7.11: MFrag sobre existência de fachada em uma empresa.

Por fim, a Figura 7.12 representa uma regra que integra as duas principais regras probabilísticas deste domínio: uma licitação é suspeita - `isSuspicious(procurement)` - se esta foi direcionada a uma empresa específica - nó de entrada `isDirected(procurement)` - e se a empresa possui "laranjas" - nó de entrada `existsFrontForEnterprise(enterprise)`. Note que o nó de contexto `hasParticipant(procurement, enterprise)` indica que somente as empresas participantes da licitação são consideradas nessa regra.

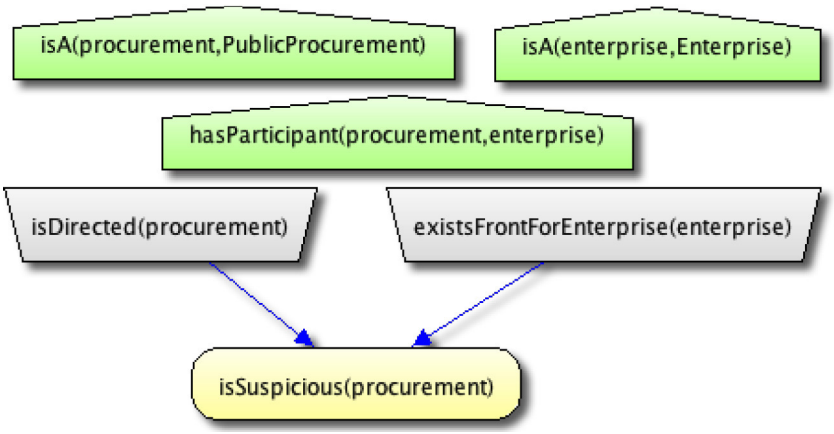

Figura 7.12: MFrag indicando se a licitação é suspeita.

7.3 Resultados e Análise

Para se testar a ontologia inicial apresentada na Seção 7.1, dois cenários foram selecionados a partir de um conjunto de casos reais representando licitações suspeitas e não suspeitas respectivamente:

- Licitação suspeita (`procurement1`):
  - `index1` = `ILC` >= 2:0;
  - `index2` = `ILG` >= 1:5;
  - `index3` = `other` >= 3:0.
  - Exige experiência em somente um contrato.
- Licitação não suspeita (`procurement2`):
  - `index4` = `IE` >= 1:0;
  - `index5` = `ILG` >= 1:0;
  - `index6` = `ILC` >= 1:0;
  - Não exige experiência em somente um contrato.

As evidências acima foram introduzidas em nosso modelo como indivíduos de classes OWL e afirmações sobre suas propriedades (*i.e.*, triplas RDF na ontologia OWL). Consultando-se

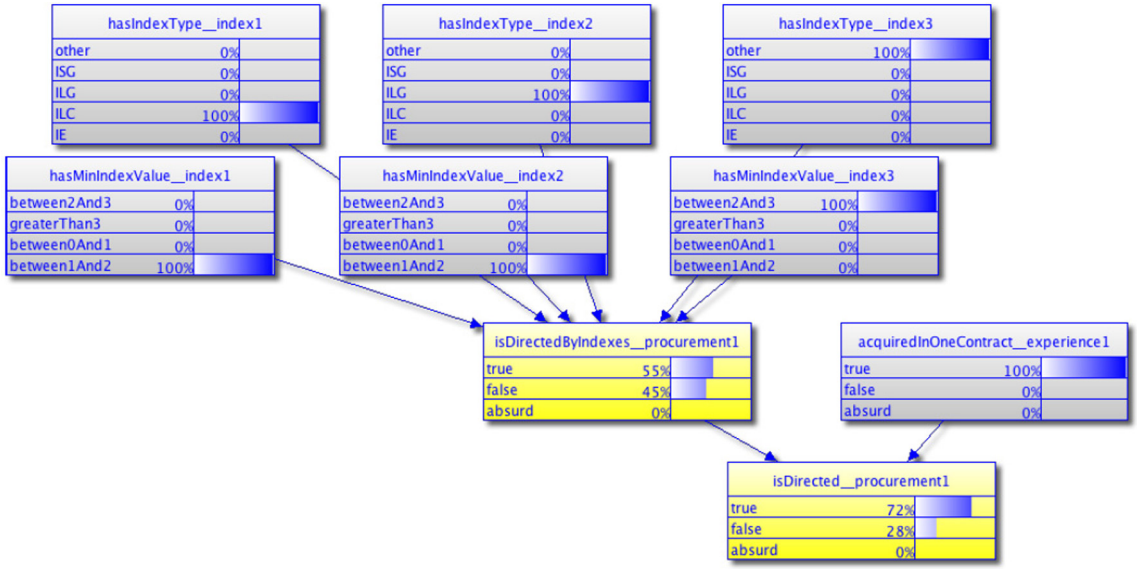

Figura 7.13: SSBN gerada consultando-se `isDirected(procurement1)`.

os valores de `isDirected(procurement1)` e `isDirected(procurement2)`, o algoritmo de geração de SSBN do plugin de PR-OWL 2 gerou a estrutura mostrada na Figura 7.13.

Apesar da Figura 7.13 mostrar somente a SSBN gerada na consulta de `isDirected (procurement1)`, a mesma estrutura vale também para `isDirected(procurement2)`, pois ambas licitações possuem evidências sobre três índices contábeis e alguma evidência sobre a experiência exigida em apenas um contrato, diferindo somente nos seus valores. Obviamente, como esperado, as probabilidades inferidas também diferem, como apresentado abaixo:

- Licitação não suspeita:
  - 0,01% que a licitação foi dirigida a uma empresa específica, usando índices incomuns de contabilidade;
  - 0,10% que a licitação foi dirigida a uma empresa específica (no sentido geral).
- Licitação suspeita:
  - 55,00% que a licitação foi dirigida a uma empresa específica via índices;
  - 72,00% que a licitação foi dirigida a uma empresa específica. Ao se omitir evidências sobre a experiência exigida em apenas um contrato, este valor cai a 29,77%.

Este resultado foi analisado por um especialista da CGU, que concordou com a inferência realizada pela referida ontologia probabilística. O mesmo afirmou que as probabilidades representam semanticamente (*i.e.* em alta, média ou baixa chance) o que ele pensaria ao analisar os mesmos indivíduos e provas. Este resultado é encorajador, pois sugere que sistemas especialistas baseados em ontologias probabilísticas são promissores.

Embora essas SSBN geradas apresentem as mesmas estruturas, é comum que as estruturas variem de licitação por licitação. Por exemplo, deparamo-nos com vários casos apresentando todos os quatro índices contábeis comuns e mais alguns outros diferentes. Nestes casos, a presença de dois índices adicionais (`index5` e `index6`) faria com que a SSBN resultante contenha

duas cópias adicionais dos nós `hasIndexType(index)` e `hasMinIndexValue(index)` (com valor de `index` substituído pelos índices adicionais). Isto torna a BN, na sua forma clássica, não aplicável neste domínio, pois a capacidade de se criar várias "cópias" de RV com base em um contexto exige BN de maior expressividade, como a oferecida por MEBN.

Notou-se que em ontologias PR-OWL 2 é fácil de se incorporar novos critérios e regras de forma incremental. No caso particular desta ontologia, indicadores de fachadas (sócios "laranjas") puderam ser adicionados sem retrabalho.

Para se avaliar a implantação da arquitetura de fusão de conhecimentos apresentada na Seção 7.2, cada ontologia foi publicada em servidores HTTP hospedados em diferentes computadores conectados por rede. Com isso, quando um usuário executar a ferramenta (*i.e.* UnBBayes com o plug-in de PR-OWL 2) e realizar uma consulta na ontologia probabilística de detecção e prevenção de fraudes (a ontologia de visão global), as ontologias locais serão importadas pela ontologia global via rede e uma inferência probabilística integrada será realizada. Este esquema simula então uma situação em que conhecimentos locais, distribuídos em diferentes fontes, mas conectados por rede, são "fundidos" a fim de se realizar uma inferência de visão mais global. Diferente do usado na Figura 7.13, as evidências utilizadas nesta avaliação foi fictícia.

A Figura 7.14 apresenta a SSBN gerada inserindo-se evidências sobre *John Doe* (fulano, em inglês), responsável pela empresa `ITBusiness` e *Jane Doe* (fulana, em inglês), responsável pela empresa `TechBusiness`. Ambas empresas também participam de `procurement1`, usada anteriormente na consulta que gerou a Figura 7.13.

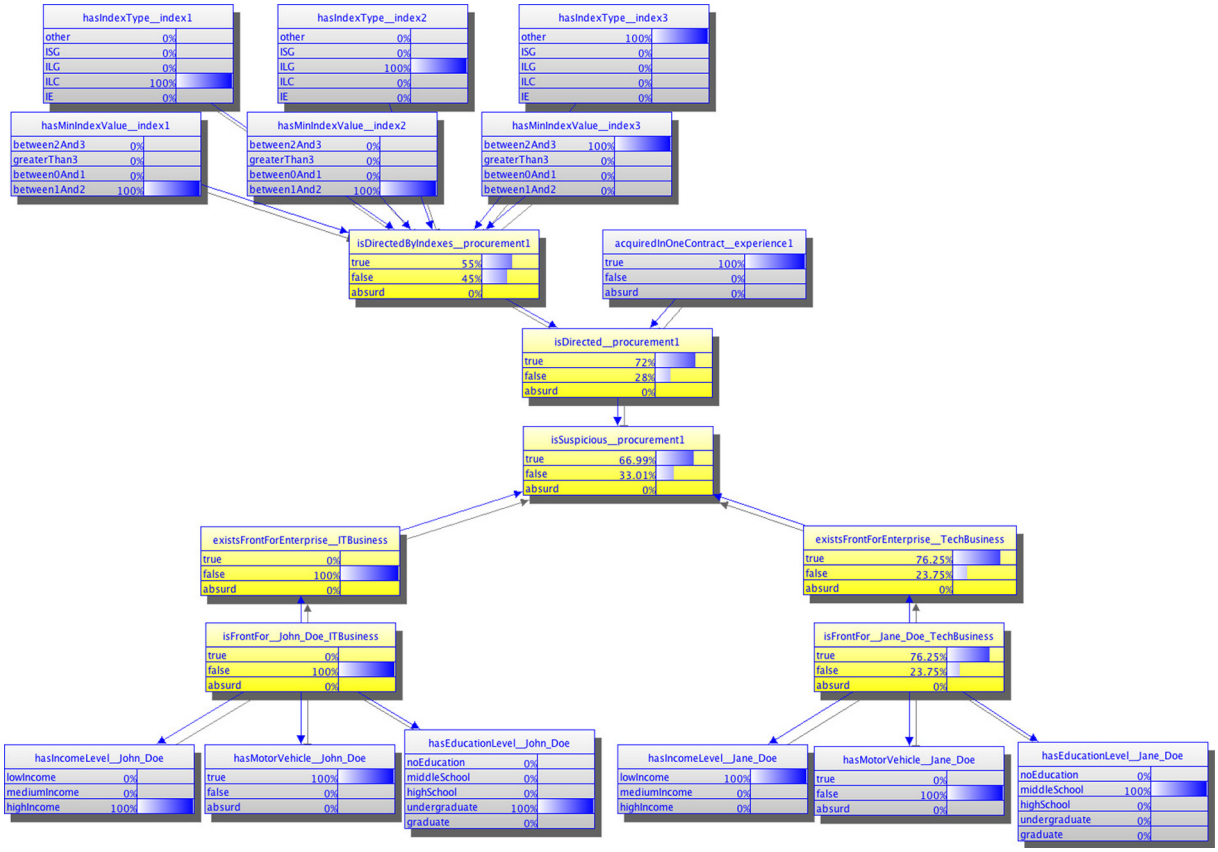

Figura 7.14: SSBN gerada consultando-se `isSuspicious(procurement1)`.

Pôde ser observado que as informações sobre *Jane Doe* favorece a hipótese de `procurement1` ser suspeita de fraudes, já que aparenta ser sócio "laranja" de `TechBusiness`. As informações sobre *John Doe*, por outro lado, não favorece essa hipótese, pois não oferece evidências que o apontam como um sócio "laranja" de `ITBusiness`.

A importância da fusão do conhecimento é melhor percebida quando consideramos as evidências de cada fonte isoladamente e posteriormente comparamos seus resultados com os obtidos na presença de todas as fontes. A informação sobre a renda anual de *Jane Doe* resulta, sozinha, em uma probabilidade *posteriori* de 8,26% de *Jane Doe* ser "laranja". A informação indicando que *Jane Doe* não possui veículos nos resulta em uma probabilidade de 0,05% de ser "laranja". Finalmente, considerando-se apenas as informações sobre escolaridade, sua probabilidade de ser "laranja" é 0,07%. É fácil notar que separadamente essas informações não fornecem fortes evidêncis de que *Jane Doe* é "laranja". No entanto, ao juntar as fontes (e consequentemente considerar as informações em conjunto), o sistema mostra fortes indícios (76,25%) de que *Jane Doe* é sócio "laranja" de `TechBusiness`, como mostra a Figura 7.14.

Espera-se que com o uso da PR-OWL 2 e do UnBBayes, vantagens como a imparcialidade no julgamento (visto que sob mesmas condições o sistema retornará sempre os mesmos resultados), escalabilidade (no sentido de capacidade de analisar milhares de licitações em um tempo curto) e análise conjunta de volume massivo de indicadores (quanto maior o número de indicadores a analisar, mais difícil se torna uma análise objetiva e coerente por um especialista) possam ser exploradas.

Capítulo 8

Conclusões e Trabalhos Futuros

Acreditamos que o presente trabalho tem realizado avanços consideráveis na ferramenta UnBBayes, reparando inclusive falhas inerentes de classes antigas como módulo MEBN. Adicionalmente, sendo a usabilidade um fator decisório na aceitação de uma ferramenta computacional, as alterações que resultaram em adição de funcionalidades na GUI são de natureza importante na aceitação do produto. A análise de performance também tem nos apontado problemas inerentes da "emulação" de um modelo baseado em FOL (MEBN) em um modelo baseado em DL (OWL 2).

No entanto, dado que a PR-OWL 2 foi formulado para resolver o problema de integração da porção determinística e probabilística de uma ontologia PR-OWL, percebe-se que foi incluída uma praticidade substancial, pois muitos engenheiros de ontologias poderão incorporar probabilidades a modelos existentes, ao invés de criar modelos probabilísticos equivalentes a partir do zero.

Uma versão inicial de PR-OWL 2 foi modelado e implementado no framework UnBBayes, que agora possui uma arquitetura de DSPL - resolvida em tempo de execução - baseada em plug-ins. O FM dessa arquitetura está expressa em OWL 2, permitindo inclusive que testes de consistência e raciocínio sejam feitas pela própria ferramenta descrita pelo modelo, e de forma bastante escalável. O plug-in de PR-OWL 2 criado para o UnBBayes permite agora que o "mapeamento" (no sentido de ligação) entre a porção probabilística e determinística seja realizado por um simples *drag-and-drop* ou seleção de elementos em uma lista, o que facilita bastante no processo de fusão (e/ou extensão) de ontologias.

No âmbito de raciocínio integrado, por falta da especificação da semântica PR-OWL 2, uma parcela significativa da implementação precisou ser realizada de forma *ad-hoc*. No entanto, a tradução programática (em Java) de uma consulta em FOL para um conjunto de consultas em DL permitiu que verdades expressas em OWL 2 sejam também interpretadas como verdades pela máquina de inferência probabilística. Com essa arquitetura, tornou-se possível a integração de ontologias probabilísticas e determinísticas não só em nível sintático, mas também em nível de raciocínio (seja, semântico).

O poder de reuso da PR-OWL 2 e as facilidades oferecidas pela ferramenta (plug-in do UnBBayes) puderam ser demonstrados com a modelagem da ontologia de detecção de fraudes em licitações públicas (Capítulo 7), que evidenciou a facilidade que a ferramenta e a linguagem oferece no reuso e fusão incremental de sub-domínios que representam novas fontes de conhecimento.

O próximo passo natural na evolução da ontologia probabilística do Capítulo 7 seria selecionar novos critérios a serem incorporados ao modelo. Espera-se que os novos critérios exigirão informações disponibilizadas somente por diferentes órgãos do governo, conforme mostrado na Seção 7.2. Portanto, o poder semântico e a capacidade de reuso da PR-OWL 2 será extremamente útil na fusão de conhecimento disponibilizado por essas novas fontes.

Vale observar que PR-OWL 2 é um conceito em estado da arte, e é natural que necessite de estudos aprofundados, eventuais ajustes e testes para se atingir a maturidade. Espera-se que a presente pesquisa contribua para o processo de amadurecimento da PR-OWL 2.

8.1 Limitações e Trabalhos Futuros

Por ser uma funcionalidade que se adere ao UnBBayes-MEBN, o plug-in de PR-OWL 2 possui diversas limitações inerentes dessa ferramenta. Por exemplo, não há ainda um suporte nativo a polimorfismo, expressões complexas em nós de input, incerteza de existência e incerteza de tipos. As últimas três poderão eventualmente ser adicionadas pelo mecanismo de plug-ins incorporado ao UnBBayes-MEBN; todavia, o polimorfismo talvez exija uma refatoração mais completa, por não ser uma funcionalidade tão pontual. Consideram-se estas tarefas um trabalho futuro.

Adicionalmente, suspeita-se que as limitações de desempenho causadas pela tradução de expressões em FOL para DL sejam um sério limitador na aceitação do produto. Soluções baseadas em cache estão previstas também como trabalho futuro.

Por ser um framework mais completo e recente, talvez a substituição do JPF pela OSGi seja interessante como um próximo passo evolutivo na infra-estrutura de plug-ins do UnBBayes. Apesar de ser um framework incomparavelmente mais complexo, o OSGi permitirá resoluções de plug-ins de maneira não suportada nativamente pelo JPF, como plug-ins em ambientes distribuídos.

Como outra extensão do UnBBayes, podemos também pensar no uso de interfaces ou serviços web (*e.g.* modelo *Service Oriented Arquitecture* - SOA (45)), para que haja armazenamento e processamento centralizado de ontologias, acessíveis ao público por navegadores convencionais e incorporáveis a outros sitemas através de requisições na web. A descoberta de serviços poderá ser realizada por inferência em ontologias, que eventualmente podem ser probabilísticas.

Adicionalmente, uma outra linha de pesquisa seria a aprendizagem em MEBN, o que permitira a utilizção do UnBBayes em domínios em que a modelagem seja guiada por dados.

Um modelo que represente uma ontologia PR-OWL 2 (ou seu subconjunto) em bancos de dados relacionais pode se tornar um grande atrativo no futuro, quando se tornar necessário a integração probabilística de modelos grandes oriundos de bancos de dados relacionais. Nesse caso, inferências baseadas em PRM (*i.e.* mapeamento entre formalismos e linguagens probabilísticas de primeira ordem) também pode se tornar interessante.

Referências

[1] CRISP-DM. Disponível em <http://www.crisp-dm.org/>. Acesso em 20/06/2011.

[2] Eclipse TPTP. Disponível em <http://www.eclipse.org/tptp/>. Acesso em 20/06/2011.

[3] Resource Description Framework. Disponível em <http://www.w3.org/RDF/>. Acesso em 20/06/2011.

[4] W3C DAML+OIL. Disponível em <http://www.w3.org/TR/daml+oil-reference>. Acesso em 20/06/2011.

[5] V. Alves, R. Gheyi, T. Massoni, U. Kulesza, P. Borba, and C. Lucena. Refactoring product lines. Em *Proceedings of the 5th international conference on Generative programming and component engineering*, GPCE '06, páginas 201–210, NY, EUA, 2006. ACM.

[6] T. Berners-Lee, D. Connolly, S. Hawke, I. Herman, E. Prudhommeaux, and R. Swick. W3C Semantic Web Activity. Disponível em <www.w3.org/2001/sw/>. Acesso em 20/06/2011.

[7] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American Magazine*, Maio 2001.

[8] F. Bobillo and U. Straccia. fuzzyDL: An expressive fuzzy description logic reasoner. *IEEE International Conference on Fuzzy Systems IEEE World Congress on Computational Intelligence*, 2008.

[9] L. B. Booker and N Hota. Probabilistic reasoning about ship images. Em *Second Annual Conference on Uncertainty in Artificial Intelligence*, Philadelphia, PA, 1986.

[10] B. G. Buchanan and E. H. Shortliffe. Rule-based expert systems: The MYCIN. Em *Experiments of the Stanford Heuristic Programming Project*, MA, USA, 1984. Addison-Wesley.

[11] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M Stal. *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*. John Wiley & Sons, Chichester, England, 1996.

[12] R. Carvalho, S. Matsumoto, K. Laskey, P. C. G. Costa, M. Ladeira, and L. Santos. Probabilistic Ontology and Knowledge Fusion for Procurement Fraud Detection in Brazil. Em *Uncertainty Reasoning for the Semantic Web 2 (to appear)*, URSW 2, 2011.

[13] R. N. Carvalho and K. C. Chang. A performance evaluation tool for multi-sensor classification systems. Em *Proceedings of the 12th International Conference on Information Fusion (FUSION 2009)*, páginas 6–9, Washington, USA, 2009.

[14] R. N. Carvalho, P. C. G. Costa, K. B. Laskey, and Kuo-Chu Chang. PROGNOS: predictive situational awareness with probabilistic ontologies. Em *Proceedings of the 13th International Conference on Information Fusion*, Edinburgh, UK, Jul. 2010.

[15] R. N. Carvalho, M. Ladeira, and M. H. P. Vieira. UnBBayes. Disponível em <http://unbbayes.sourceforge.net/>. Acesso em 20/06/2011.

[16] R. N. Carvalho and M. S. Onishi. Framework e API para construção de sistemas inteligentes baseados em diagrama de influências e rede bayesiana múltipla secionada. Monografia de Graduação. Departamento de Ciência da Computação, Universidade de Brasília, 2003.

[17] R. N. Carvalho, M. S. Onishi, and M. Ladeira. Desenvolvimento da versão Java do UnBBayes framework para raciocínio probabilístico. Em *8º Congresso de Iniciação Científica da UnB*, Universidade de Brasília: Brasilia, DF, Brazil, 2002.

[18] R.N. Carvalho. Raciocínio plausível na web semântica através de Redes Bayesianas Multi-Entidades - MEBN. Dissertação de Mestrado, Mestrado em Informática, Universidade de Brasília, 2008.

[19] R.N. Carvalho, K.B. Laskey, and Paulo C. G. Costa. PR-OWL 2.0 - Bridging the gap to OWL semantics. Em *Proceedings of the 6th Uncertainty Reasoning for the Semantic Web (URSW 2010) on the 9th International Semantic Web Conference (ISWC 2010)*, Shanghai, China, Nov. 2010.

[20] R.N. Carvalho, L.L. Santos, M. Ladeira, and P.C.G. Costa. A GUI Tool for Plausible Reasoning in the Semantic Web using MEBN. Em *Proceedings of the 7th International Conference on Intelligent Systems Design and Applications*, Rio de Janeiro, Brazil, 2007. IEEE Press.

[21] R.N. Carvalho, L.L. Santos, S. Matsumoto, M. Ladeira, and P.C.G. Costa. A GUI tool for plausible reasoning in the semantic web using MEBN. Em *Innovative Applications in Data Mining*, volume 169, série *Studies in Computational Science*, páginas 17–45. Springer-Verlag, Berlin/Heidelberg, 2008.

[22] R.N. Carvalho, L.L. Santos, S. Matsumoto, M. Ladeira, P.C.G. Costa, and K.B. Laskey. UnBBayes: Modeling uncertainty for plausible reasoning in the semantic web. Em *Semantic Web*, páginas 1–28. IntechWeb, Croatia, 2010.

[23] C. Cetina, P. Giner, J. Fons, and V. Pelechano. Designing and prototyping dynamic software product lines: techniques and guidelines. Em *Proceedings of the 14th international conference on Software product lines: going beyond*, SPLC'10, páginas 331–345, Berlin, Heidelberg, 2010. Springer-Verlag.

[24] H. Chalupsky. OntoMorph: A translation system for symbolic knowledge. Em *In Principles of Knowledge Representation and Reasoning*, páginas 471–482. Morgan Kaufmann, 2000.

[25] E. Charniak. Bayesian network without tears. *AI Magazine, American Association for Artificial Intelligence (AAAI)*, 1991. Disponível em <http://www.aaai.org/ojs/index.php/aimagazine/article/view/918/836>. Acesso em 20/06/2011.

[26] E. Charniak and R. P. Goldman. A semantics for probabilistic quantifier-free first-order languages with particular application to story understanding. Em *Eleventh International Joint Conference on Artificial Intelligence*, Detroit, Michigan, USA, 1989.

[27] P. Costa and Zeigler B. Modeling framework for synchronizing global and local situation awareness. Relatório técnico, C4I Center at George Mason University, 2011.

[28] P. C. G. Costa, K. Chang, K. B. Laskey, and R. N. Carvalho. High level fusion and predictive situational awareness with probabilistic ontologies. Em *In Proocedings of the AFCEA-GMU C4I Center Symposium*, Fairfax, VA, USA, 2010. George Mason University.

[29] P. C. G. Costa, K. B. Laskey, and K. J. Laskey. Probabilistic ontologies for efficient resource sharing in semantic web services. Em *In Proceedings of the Second Workshop on Uncertainty Reasoning for the Semantic Web (URSW 2006)*, Athens, GA, USA, 2006.

[30] P. C. G. Costa, K. B. Laskey, M. Takikawa, M. Pool, F. Fung, and E. J. Wright. MEBN logic: A Key Enabler for Network Centric Warfare. Em *10th International Command and Control Research and Technology Symposium*, 10th ICCRTS, McLean, Virginia, USA, 2005. CCRP publications.

[31] P.C.G. Costa. *Bayesian Semantics for the Semantic Web*. Tese de Doutorado, Department of Systems Engineering and Operational Research, George Mason University, 2005.

[32] P.C.G. Costa, Kuo-Chu Chang, K.B. Laskey, and Rommel Novaes Carvalho. A multidisciplinary approach to high level fusion in predictive situational awareness. Em *Proceedings of the 12th International Conference on Information Fusion*, páginas 248–255, Seattle, Washington, USA, Jul. 2009.

[33] P.C.G. Costa and K.B. Laskey. Multi-Entity Bayesian Networks without Multi-Tears. Em *C4I Papers*, C4I-05-07. Department of Systems Engineering and Operations Research, George Mason University, Fairfax, VA, USA, 2006. Disponível em <http://hdl.handle.net/1920/456>. Acesso em 20/06/2011.

[34] P.C.G. Costa, K.B. Laskey, and K.J. Laskey. PR-OWL: A Bayesian Ontology Language for the Semantic Web. Em *Proceedings of the ISWC Workshop on Uncertainty Reasoning for the Semantic Web*, Galway, Ireland, 2005.

[35] K. Czarnecki, C. H. Peter Kim, and K. T. Kalleberg. Feature Models are Views on Ontologies. Em *Proceedings of the 10th International on Software Product Line Conference*, páginas 41–51, Washington, DC, USA, 2006. IEEE Computer Society.

[36] D. C. da Silva. Aprendizagem de máquinas para redes bayesianas. Monografia de Graduação. Departamento de Ciência da Computação, Universidade de Brasília, 2002.

[37] D. C. da Silva. Aprendizagem estrutural de redes bayesianas com dados massivos. Dissertação de Mestrado, Mestrado em Informática, Universidade de Brasília, 2005.

[38] J. de Bruijn, M. Ehrig, C. Feier, F. Martíns-Recuerda, F. Scharffe, and M. Weiten. Ontology Mediation, Merging, and Aligning. Edit. John Davies, Rudi Studer, and Paul Warren, *Semantic Web Technologies*. Wiley, Jul. 2006.

[39] A. P. Dempster. Upper and lower probabilities induced by a multivalued mapping. *Annals of Mathematical Statistics*, 1967.

[40] C. Dias, M. Miska, and M. Kuwahara. Self Learning Tool for Travel Time Estimation in Signalized Urban Networks Based on Probe Data. *6th ITS symposium*, 2007.

[41] A. Doan, J. Madhaven, P. Domingos, and A. Halevy. Ontology matching: A machine learning approach. Edit. S. Staab and R. Studer, *Handbook on Ontologies in Information Systems*, páginas 397—416. Springer-Verlag, 2004.

[42] P. Domingos and D. Lowd. *Markov Logic: An Interface Layer for AI*. Morgan & Claypool, San Rafael, CA, 2009.

[43] D. Dou, D. McDermott, and P. Qi. Ontology translation by ontology merging and automated reasoning. Em *Proc. EKAW2002 Workshop on Ontologies for Multi-Agent Systems*, páginas 3—18, 2002.

[44] M. Ehrig. FOAM - framework for ontology alignment and mapping; results of the ontology alignment initiative. Em *Proceedings of the Workshop on Integrating Ontologies*, volume 156, páginas 72–76. CEUR-WS.org, 2005.

[45] T. Erl. *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. Prentice Hall, 2005. ISBN: 978-0131858589.

[46] C. M. Fernandes. Aprendizagem estrutural de redes bayesianas usando conjunto de dados reduzido. Dissertação de Mestrado, Mestrado em Informática, Universidade de Brasília, 2003.

[47] Stanford Center for Biomedical Informatics Research. Protégé. Disponível em <http://protege.stanford.edu/>. Acesso em 20/06/2011.

[48] IBM Eclipse Foundation. Eclipse. Disponível em <http://www.eclipse.org>. Acesso em 20/06/2011.

[49] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, USA, 1994.

[50] J. A. R. Garcia. *jCOLIBRI: A multi-level platform for building and generating CBR systems*. Tese de Doutorado, Department of Software Engineering and Artificial Intelligence Facultad de Informatica Universidad Complutense de Madrid, 2008.

[51] L. Getoor, N. Friedman, D. Koller, and A. Pfeffer. Learning probabilistic relational models. Edit. Saso Dzeroski and Nada Lavrac, *Relational Data Mining*, New York, USA, 2001. Springer-Verlag.

[52] R. Giurno and T. Lukasiewicz. P-$\mathcal{SHOQ}$(D): A Probabilistic Extension of $\mathcal{SHOQ}$(D) for Probabilistic Ontologies in the Semantic Web. Em *European Conference on Logics in Artificial Intelligence (JELIA 2002)*, Cosenza, Italy, 2002.

[53] A. Goldberg and A. Kay. Smalltalk-72 instruction manual. ssl 76-6. Relatório técnico, Learning Research Group, Xerox Palo, Alto Research Center, 1976.

[54] S. Hallsteinsen, M. Hinchey, S. Park, and K. Schmid. Dynamic Software Product Lines. Em *Computer*, volume 41(4), páginas 93–95. IEEE Computer Society Press, Los Alamitos, CA, USA, 2008.

[55] J. Heflin. OWL Web Ontology Language - Use Cases and Requirements (W3C Recommendation). Disponível em: <www.w3.org/TR/2004/REC-webont-req-20040210>. Acesso em 20/06/2011.

[56] J. HEINSOHN. Probabilistic Description Logics. Em *Tenth Conference on Uncertainty in Artificial Intelligence (UAI-94)*, Seattle, WA, USA, 1994.

[57] F. V. Jensen and F. Jensen. Optimal junction trees. Em *In: Proceedings of the Conference on Uncertainty in Artificial Intelligence*, páginas 360–366, San Francisco, USA, 1994.

[58] F. V. Jensen, K. G. Olsen, and S. K. Andersen. An algebra of bayesian belief universes for knowledge-based systems. *Networks*, 1990.

[59] F.V. Jensen, S.L. Lauritzen, and K.G. Olsen. Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly*, 1990.

[60] H. Knublauch. Protégé-owl api programmer's guide. Disponível em <http://protege.stanford.edu/plugins/owl/api/guide.html>. Acesso em 20/06/2011.

[61] D. Koller and A. Pfeffer. Object-Oriented Bayesian Networks. Em *Thirteenth Conference on Uncertainty in Artificial Intelligence (UAI-97)*, San Francisco, CA, USA, 1997.

[62] M. Ladeira, M. G. de Oliveira, and M. E. C. de Araújo. Lupa Digital: Agilização da Busca Decadactilar na Identificação Criminal Através de Mineração de Dados. *SEMISH XXXII*, Jul. 2005.

[63] M. Ladeira, D.C. Silva, M.H.P. Vieira, M.S. Onishi, R.N. Carvalho, and W.T. Silva. Ferramenta Aberta e Independente de Plataforma para Redes Probabilísticas. Em *Encontro Nacional de Inteligência Artificial - ENIA*, Campinas, Brazil, 2003.

[64] M Ladeira, M.H.P Vieira, H.A Prado, R.M Noivo, and D.B.S Castanheira. UnBMiner - ferramenta aberta para mineração de dados. *Revista Tecnologia da Informação*, 2005.

[65] B. K. Laskey, P. C. G. Costa, and T. Janssen. Probabilistic ontologies for Multi-INT fusion. Relatório técnico, George Mason University, C4I Center, 2008.

[66] J.K. Laskey, B.K Laskey, P.C.G. Costa, M. M. Kokar, T. Martin, and T. Lukasiewicz. W3C Incubator Group Report. Em *Uncertainty Reasoning for the World Wide Web*, Busan, Korea, 2007-2008. Disponível em <http://www.w3.org/2005/Incubator/urw3/XGR-urw3-20080331>. Acesso em 20/06/2011.

[67] K. B. Laskey. First-Order Bayesian Logic. Relatório técnico, George Mason University SEOR Dept., Fairfax, VA, 2005. Disponível em <http://ite.gmu.edu/ klaskey/publications.html>. Acesso em 20/06/2011.

[68] K. B. Laskey. First-order bayesian logic. Relatório técnico, George Mason University, 2006.

[69] K.B. Laskey, P. Costa, and T. Janssen. Probabilistic ontologies for knowledge fusion. Em *11th International Conference on Information Fusion*, páginas 1–8, Cologne, 2008. ISBN: 978-3-8007-3092-6.

[70] K.B. Laskey and P.C.G. Costa. Of Starships and Klingons: Bayesian Logic for the 23rd Century. Em *Proceedings of the Twenty-first Conference Uncertainty in Artificial Intelligence (UAI -05)*, Arlington, Virginia, USA, 2005. AUAI Press.

[71] K.B. Laskey, P.C.G. Costa, E.J. Wright, and K.J. Laskey. Probabilistic Ontology for Net-Centric Fusion. Em *Proceedings of the Tenth International Conference on Information Fusion*, páginas 1–8, Quebec, Canada, 2007.

[72] K.J. Laskey, K.B. Laskey, and P.C.G Costa. Uncertainty Reasoning for the World Wide Web Incubator Group Charter (W3C Incubator Activity). Disponível em <www.w3.org/2005/Incubator/urw3/charter>. Acesso em 20/06/2011.

[73] T. Lukasiewicz. Stratified probabilistic description logic programs. Relatório técnico, Dipartimento di Informatica e Sistemistica, Università di Roma La Sapienza, Italy, Nov. 2005.

[74] J. E. O. Luna. Algoritmos EM para Aprendizagem de Redes Bayesianas a partir de Dados Incompletos. Dissertação de Mestrado, Universidade Federal de Mato Grosso do Sul, 2004.

[75] H. C. Marques, J. M. P. de Oliveira, and P. C. G. Costa. Representing COA with probabilistic ontologies. Em *16th International Command and Control Research and Technology Symposium (ICCRTS)*, Collective C2 in Multinational Civil-Military Operations, 2010.

[76] S. Matsumoto. Extensão do UnBBayes para suporte a OOBN – Object Oriented Bayesian Network -. Relatório Técnico. Departamento de Ciência da Computação, Universidade de Brasília, 2008.

[77] S. Matsumoto. UnBBayes-PRM: Framework para edição e inferência em PRM. Relatório Técnico. Departamento de Ciência da Computação, Universidade de Brasília, 2010.

[78] S. Matsumoto, R. N. Carvalho, P. C. G. Costa, K.B. Laskey, L. L. Santos, Silva, and M. Ladeira. There's no more need to be a night OWL: on the PR-OWL for a MEBN tool before nightfall. Edit. Gabriel Fung, *Introduction to the Semantic Web: Concepts, Technologies and Applications*. iConcept Press, 2010. ISBN: 9780980733013.

[79] S. Matsumoto, R. N. Carvalho, M. Ladeira, P. C. G. Costa, L. L. Santos, D. Silva, M. Onishi, and E. Machado. UnBBayes: a java framework for probabilistic models in AI. Em *Java in Academic and Research*. iConcept Press, 2010.

[80] S. Matsumoto, R. Mezzomo, and E. Aguiar. MSSSBN - programa implementado no curso de treinamento do GIA/UnB sobre UnBBayes. Departamento de Ciência da Computação, Universidade de Brasília, 2010.

[81] S. Matsumoto and L. L. Santos. Framework para Redes Bayesianas Multi-Entidades e Ontologias Probabilísticas. Monografia de Graduação. Departamento de Ciência da Computação, Universidade de Brasília, 2008.

[82] M. Mazzieri and A. F. Dragoni. A fuzzy semantics for semantic web languages. Relatório técnico, Université Politecnica delle Marche Dipartimento di Elettronica, Intelligenza Artificiale e Telecomunicazioni (DEIT), Mar. 2006.

[83] B. Meyer. *Object-oriented software construction*. Prentice-hall International Series in Computer Science. Prentice Hall PTR, 1997.

[84] M. Miska, T. H. J. Muller, and H. van Zuylen. MiOS – a microscopic online simulator. Em *ITS World Conference*, Nagoya, Japan, 2004.

[85] M. Miska, T. H. J. Muller, and H. van Zuylen. A self-learning driving behavior model for microscopic online simulation based on remote sensing and equipped vehicle data. Em *The 16th Mini - EURO Conference and 10th Meeting of EWGT*, 2005.

[86] K. P. Murphy. Dynamic bayesian networks: Representation, inference and learning, 2002.

[87] N. F. Noy, R. Fergerson, and M. A. Musen. The knowledge model of protege-2000: Combining interoperability and flexibility. Em *2th International Conference on Knowledge Engineering and Knowledge Management (EKAW'2000)*, Juan-les-Pins, France, 2000.

[88] N. F. Noy and M. A. Musen. The PROMPT suite: Interactive tools for ontology merging and mapping. *International Journal of Human-Computer Studies*, 2003.

[89] Object Management Group. Unified Modeling Language 2.0. Disponível em <http://www.uml.org/>. Acesso em 20/06/2011.

[90] D. Olshansky. Java Plug-in Framework (JPF). Disponível em <http://jpf.sourceforge.net/>. Acesso em 20/06/2011.

[91] P.F. Patel-Schneider, P. Hayes, and I. Horrocks. OWL Web Ontology Language - Semantics and Abstract Syntax (W3C Recommendation). Disponível em <http://www.w3.org/TR/owl-semantics/>. Acesso em 20/06/2011.

[92] J. Pearl. *Artificial Intelligence*, volume 29, capítulo Fusion, Propagation, and Structuring in Belief Networks, páginas 241–288. Amsterdam, 1986.

[93] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, San Mateo, CA, USA, 1988.

[94] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 1 edition, 2005.

[95] D. Poole. The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence*, 1997.

[96] L. Predoiu and H. Stuckenschmidt. Probabilistic extensions of semantic web languages - a survey. Em *The Semantic Web for Knowledge and Data Management. Technologies and Practices*. Idea Group Inc, 2008.

[97] S. Raza and S. Haider. Modeling first-order bayesian networks (FOBN) - a comparative study of BLOG, BLP and MEBN. Em *3rd International Conference on Advanced Computer Theory and Engineering*, Chendgu, China, 2010.

[98] M. Richardson and P. Domingos. Markov logic networks. Relatório técnico, University of Washington, Seattle, WA, 2004.

[99] G. Shafer. *A Mathematical Theory of Evidence*. University Press, Princeton, NJ, USA, 1976.

[100] A. M. M. Soares. UnBBayes: Framework para redes probabilísticas. Monografia de Graduação. Departamento de Ciência da Computação, Universidade de Brasília, 2000.

[101] D. J. Spiegelhalter, R. Franklin, and K. Bull. Assessment, criticism, and improvement of imprecise probabilities for a medical expert system. Em *Fifth Conference on Uncertainty in Artificial Intelligence*, páginas 257–285, Mountain View, CA, 1989.

[102] G. Stoilos, G. Stamou, V. Tzouvaras, J. Z. Pan, and I. Horrocks. The fuzzy description logic f-$\mathcal{SHIN}$. Relatório técnico, Department of Electrical and Computer Engineering, National Technical University of Athens, Greece. School of Computer Science, The University of Manchester, UK, Nov. 2004.

[103] Sun Microsystems, Inc. Java language specification, second edition, Chapter 11 - Exceptions. Disponível em <http://java.sun.com/docs/books/jls/second_edition/html/exceptions.doc.html>. Acesso em 20/06/2011.

[104] M. Svahnberg1, J. Van Gurp, and J. Bosch. A taxonomy of variability realization techniques. Em *Software—practice and experience*, volume 35, páginas 705—-754. Wiley InterScience, 2005. DOI: 10.1002/spe.652.

[105] University of Manchester. OWLAPI. Disponível em <http://owlapi.sourceforge.net/>, 2010. Acesso em 20/06/2011.

[106] M. Uschold. Creating, integrating, and maintaining local and global ontologies. Em *Proceedings of the First Workshop on Ontology Learning (OL-2000), in conjunction with the 14th European Conference on Artificial Intelligence (ECAI-2000)*, Berlin, Germany, 2000.

[107] F. Valadares and J. Luiz. Simulação de sistemas complexos para fins de entretenimento usando redes bayesianas: o caso do FutSim, 2002. Universidade Federal de Pernambuco.

[108] J. Van Gurp, J. Bosch, and M. Svahnberg. On the notion of variability in software product lines. Em *Working IEEE/IFIP Conference on Software Architecture*, WICSA'01. IEEE, 2001.

[109] R. M. Vicari, C. D. Flores, L. Seixas, J. C. Gluz, and H. Coelho. AMPLIA: A Probabilistic Learning Environment. *International Journal of Artificial Intelligence in Education*, 2008.

[110] W3C OWL Working Group. OWL 2 Web Ontology Language Document Overview. Disponível em <http://www.w3.org/TR/owl2-overview/>. Acesso em 20/06/2011.

[111] H. H. Wang, Y. F. Li, J. Sun, H. Zhang, and J. Pan. Verifying feature models using OWL. Em *Journal of Web Semantics*, volume 5(2), páginas 117–129, Amsterdam, Jun. 2007. Elsevier Science Publishers B. V.

[112] Y. Xiang, D. Poole, and M. P. Beddoes. Multiply sectioned bayesian networks and junction forests for large knowledge based systems. Em *Computational Intelligence*, volume 9(2), páginas 171–220, Cambridge, 1993.