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

Raciocínio Plausível na Web Semântica através de Redes Bayesianas Multi-Entidades - MEBN

Rommel Novaes Carvalho

Monografia apresentada como requisito parcial
para conclusão do Mestrado em Informática

Orientador
Prof. Dr. Marcelo Ladeira

Coorientador
Prof. Dr. Paulo Cesar Guerreiro da Costa

Brasília
2008

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

Coordenadora: Prof.ª Dr.ª Alba Cristina Magalhães Alves de Melo

Banca examinadora composta por:

Prof. Dr. Marcelo Ladeira (Orientador) – CIC/UnB
Prof. Dr. Fabio Gagliardi Cozman – PMR/USP
Prof. Dr. Li Weigang – CIC/UnB

CIP – Catalogação Internacional na Publicação

Rommel Novaes Carvalho.

Raciocínio Plausível na Web Semântica através de Redes Bayesianas Multi-Entidades - MEBN/ Rommel Novaes Carvalho. Brasília : UnB, 2008.
118 p. : il. ; 29,5 cm.

Tese (Mestre) – Universidade de Brasília, Brasília, 2008.

1. raciocínio probabilístico, 2. Web Semântica, 3. MEBN, 4. rede bayesiana, 5. UnBBayes, 6. ontologia probabilística, 7. lógica de primeira ordem

CDU 004

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

Raciocínio Plausível na Web Semântica através de Redes Bayesianas Multi-Entidades - MEBN

Rommel Novaes Carvalho

Monografia apresentada como requisito parcial
para conclusão do Mestrado em Informática

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

Prof. Dr. Fabio Gagliardi Cozman
PMR/USP

Prof. Dr. Li Weigang
CIC/UnB

Profª. Drª. Alba Cristina Magalhães Alves de Melo
Coordenadora do Mestrado em Informática

Brasília, 29 de fevereiro de 2008

Dedicatória

Dedico este trabalho à minha família que foi sempre exemplo em todos campos de atuação, da seriedade e honestidade profissional ao respeito e afeto na convivência social.

Sem minha família, não teria conquistado o ambiente e os valores necessários para realizar essa jornada de muito esforço, dedicação e em alguns momentos até mesmo frustações.

Todo esse esforço realizado ao longo da jornada está começando a colher seus frutos não apenas com a conclusão deste trabalho, mas com a repercusão que tem alcançado, com oportunidades que têm surgido para continuação da pesquisa e com a contribuição científica que representa e conseqüentemente uma evolução que oportunamente será usufruído pela sociedade.

Esses resultados são motivos de alegria e satisfação para todos que participaram desse processo. Um momento vivido de forma muito especial por mim e por minha família.

Agradecimentos

Agradeço ao meu orientador Prof. Dr. Marcelo Ladeira por ter me orientado e aturado por mais de 7 anos. Muito mais que um professor, um amigo que educa tanto academicamente como socialmente, formando mestres que antes de tudo são cidadãos dignos de seus direitos e conscientes de suas responsabilidades sociais.

Agradeço também ao co-orientador Prof Dr. Paulo Cesar G. Costa por sua paciência e tolerância para ensinar um assunto tão complicado e com tanto entusiasmo e alegria. Um pessoa super prestativa que muitas vezes parecia ser o próprio mestrando pelas diversas noites passadas em claro terminando de corrigir artigos para submeter antes que o prazo se encerrasse. Como se não bastasse o excelente trabalho que fez como co-orientador, continua com um papel essencial no apoio que tem dado para a continuação de minha formação acadêmica, ajudando até mesmo a preencher meu resume e *application* para tentar um doutorado na *George Mason University* - GMU.

Gostaria também de agradecer a Prof. Dra. Kathryn B. Laskey que através do contato que possui com o Prof. Dr. Paulo Cesar G. Costa nos proporcionou um assunto muito interessante e estimulante para a pesquisa realizada e não mediu esforços para responder prontamente todas dúvidas (e não foram poucas) que foram levantadas sobre seu trabalho (MEBN).

Não poderia deixar de fazer um agradecimento especial às pessoas responsáveis pela concretização deste trabalho. Os graduandos em Ciência da Computação Shou Matsumoto e Laécio Lima foram essenciais para a implementação do UnBBayes com suporte à MEBN e PR-OWL. Sem eles tenho certeza que este trabalho não seria concluído a tempo e não teria a mesma qualidade. Pessoas de muita responsabilidade e de alta produtividade que recomendo para qualquer projeto e/ou pesquisa que envolva tecnologia da informação. Além deles, amigos como Michael Onishi e Danilo Custódio, se colocaram a disposição para ajudar a evoluir o UnBBayes, mesmo não participando diretamente da pesquisa.

Além disso, não seria possível realizar essa pesquisa e concluir o mestrado se não fosse o apoio que recebi da Controladoria-Geral da União - CGU, especialmente do meu diretor José Geraldo Rodrigues e de meu coordenador Oswaldo de Azeredo que estiveram sempre dispostos a me escutar e a me ajudar a tornar esse sonho uma realidade.

Por fim, agradeço a todos os amigos e familiares que de forma direta ou indireta contribuíram para a conclusão deste trabalho.

A todos vocês meu *MUITO OBRIGADO*!

Lista de Abreviaturas

BN	Rede Bayesiana (*Bayesian Network*)
CPT	Tabela de Probabilidades Condicionais (*Conditional Probability Tables*)
DM	Distúrbio magnético
DAG	Grafo Acíclico Orientado (*Directed Acyclic Graph*)
DBN	Rede Bayesiana Dinâmica (*Dynamic Bayesian Network*)
DL	Lógica Descritiva (*Description Logic*)
FOL	Lógica de primeira ordem (*First Order Logic*)
HOL	Lógica da mais alta ordem (*Higher Order Logic*)
HTML	*Hipertext Marktup Language*
IA	Inteligência Artificial
IO	Entrada/Saída (*Input/Output*)
JPD	Distribuição de probabilidade conjunta (*Joint Probabilistic Distribution*)
KB	Base de conhecimento (*Knowledge Base*)
KIF	Formato de intercâmbio de conhecimento (*Knowledge Interchange Format*)
KRR	Representação de conhecimento e raciocínio (*Knowledge Representation and Reasoning*)
MEBN	Rede Bayesiana Multi-Entidade (*Multi-Entity Bayesian Network*)
MSBN	Redes Bayesianas Múltiplas Secionadas (Multiply Sectioned Bayesian Network)
MN	Redes de Markov (*Markov Network*)
OWL	*Ontology Web Language*
PDBN	Rede Bayesiana Dinâmica Parcial (*Partially Dynamic Bayesian Network*)
RDF	*Resource Description Framework*

RV	Variável aleatória (*Random Variable*)
SSBN	Rede bayesiana de situação específica (*Situation-Specific Bayesian Network*)
STELLA	*Strongly Typed, Lisp-like LAnguage*
UML	Linguagem de Modelagem Unificada (*Unified Modeling Language*)
W3C	*WWW Consortium*
SW	Web Semântica (*Semantic Web*)
WWW	*World Wide Web*
XML	*Extensible Marktup Language*

Resumo

O objetivo geral deste trabalho é pesquisar formalismos para extensão de redes bayesianas (BN) e raciocínio plausível na Web Semântica. Dentre os formalismos para raciocínio plausível, um dos mais utilizados, dada sua flexibilidade, é rede bayesiana. No entanto, há diversas situações do mundo real e na Web que as BN são incapazes de representar. As duas principais restrições de redes bayesianas são a impossibilidade de representar recursão e situações onde o número de variáveis aleatórias envolvidas é desconhecido. Exatamente para superar essas limitações que utilizaremos o formalismo MEBN (*Multi-Entity Bayesian Network*), que agrega o poder de expressividade da lógica de primeira ordem (FOL) às redes bayesianas, possibilitando a representação de um número infinito de variáveis aleatórias e a representação de definições recursivas. Na Web Semântica, a linguagem OWL (*Ontology Web Language*) permite a definição de ontologias, mas por ser baseada na FOL não possui um suporte adequado para possibilitar o raciocínio plausível. Por ser uma implementação de lógica probabilística de primeira ordem, a MEBN é um dos formalismos mais indicados para tal expansão, tendo a linguagem PR-OWL (*Probabilistic OWL*) sido proposta como uma integração de MEBN e OWL [29, 28, 27]. O presente trabalho de pesquisa propõe alguns refinamentos do formalismo MEBN e da linguagem PR-OWL e apresenta a primeira implementação no mundo de MEBN com a possibilidade de representar e raciocinar em domínios com incerteza através de ontologias probabilísticas baseadas em PR-OWL. Além disso, um novo algoritmo para geração de SSBN (*Situation-Specific Bayesian Network*) foi proposto. Essa implementação foi feita no UnBBayes [26], ferramenta livre para raciocínio probabilístico, aproveitando o mecanismo de criação e inferência de BN que esta já possui. Para exemplificar uma aplicação de MEBN/PR-OWL, o UnBBayes-MEBN [6] foi utilizado para modelar e raciocinar no domínio fictício *Star Trek*. Como ontologias probabilísticas possuem um grande potencial de uso no campo da Web Semântica onde a incerteza é tratada de forma probabilística, essa pesquisa representa uma contribuição para os trabalhos que estão sendo realizados pelo URW3-XG (*W3C Uncertainty Reasoning for the World Wide Web Incubator Group*) [30], criado pelo Consórcio World Wide Web para melhor definir o desafio de representar e raciocinar com a informação incerta disponível na Web.

Palavras-chave: raciocínio probabilístico, Web Semântica, MEBN, rede bayesiana, UnBBayes, ontologia probabilística, lógica de primeira ordem

Abstract

The general objective of this work is to research formalisms for extending Bayesian networks (BN) and plausible reasoning in the Semantic Web. Among the formalisms for plausible reasoning, one of the most used, given its flexibility, is BN. However, there are several situations in the real world and in the Web that BN is unable to represent. The two main restrictions of BN are the impossibility of representing recursion and situations where the number of random variables is unknown. It is exactly to overcome those limitations that we will use the MEBN formalism (Multi-Entity Bayesian Network), which joins the expressiveness power of first order logic (FOL) to BN, making possible the representation of an infinite number of random variables and of recursive definitions. In Semantic Web, the OWL language (Ontology Web Language) allows the definition of ontologies, but because it is based in FOL it doesn't possess an appropriate support to make the plausible reasoning possible. MEBN is one of the most suitable formalisms for such expansion, as it is an implementation of first order probabilistic logic, having the PR-OWL language (Probabilistic OWL) as an integration of MEBN and OWL [29, 28, 27]. This research proposes some refinements for the MEBN formalism and PR-OWL language and it presents the first implementation in the world of MEBN with the possibility to represent and reason in domains with uncertainty through probabilistic ontologies based in PR-OWL. Besides that, a new algorithm for the SSBN generation was proposed. This implementation was made in UnBBayes [26], a free tool for probabilistic reasoning, taking advantage of the BN modeling and inference mechanism that it already has. To exemplify an application of MEBN/PR-OWL, UnBBayes-MEBN [6] was used to model and to reason in the Star Trek toy domain. As probabilistic ontologies have a great potential use in the field of Semantic Web where the uncertainty is treated in a probabilistic way, this research represents a contribution for the work that is being accomplished by the URW3-XG (W3C Uncertainty Reasoning for the World Wide Web Incubator Group) [30], created by the World Wide Web Consortium for best defining the challenge of representing and reasoning with the available uncertain information in the Web.

Keywords: probabilistic reasoning, Semantic Web, MEBN, Bayesian network, UnBBayes, probabilistic ontology, first order logic

Sumário

Lista de Figuras **13**

Lista de Tabelas **15**

Capítulo 1 Introdução **16**
1.1 Definição do Problema 16
1.2 Objetivos 18
1.2.1 Objetivo geral 18
1.2.2 Objetivos específicos 18
1.3 Áreas de Pesquisa Relacionadas 18
1.4 Estrutura da Dissertação 19

Capítulo 2 Conceitos Básicos **20**
2.1 Ontologia 21
2.2 OWL 21
2.3 Lógica 23
2.4 BN 26
2.4.1 Desvantagens 27

Capítulo 3 Fundamentos **30**
3.1 Extensão probabilística para lógica descritiva 30
3.2 Extensão probabilística para OWL 31
3.3 PR-OWL 31

Capítulo 4 Rede Bayesiana Multi-Entidade **33**
4.1 O que é MEBN? 33
4.2 MFrag 34
4.3 Recursividade em MEBN 36
4.4 MTeoria 39
4.5 Inferência em MEBN 41

Capítulo 5 Arquitetura do UnBBayes **43**
5.1 Arquitetura 43
5.2 Histórico do UnBBayes 44
5.3 Visão geral do UnBBayes 45
5.4 Modelagem do UnBBayes 45
5.4.1 BN e ID 46

5.4.2 MSBN 47

Capítulo 6 Bibliotecas de Terceiros 49
6.1 PowerLoom 49
6.1.1 Introdução ao PowerLoom 49
6.1.2 Lógica de Primeira Ordem no PowerLoom 50
6.2 Protégé 52

Capítulo 7 Modelagem e Implementação de MEBN 57
7.1 Modelagem da extensão para MEBN 57
7.1.1 MTeoria 57
7.1.2 Nós de MEBN 58
7.1.3 MFrag 60
7.2 Tabela de Probabilidades 61
7.3 Interface Gráfica da MEBN 63
7.3.1 Arquitetura da GUI 64
7.3.2 Tela principal 69
7.3.3 Painéis de Edição 71
7.3.4 Edição de Entidades e de Variáveis Ordinárias 77
7.3.5 Edição do Pseudocódigo para a definição da CPT 78
7.3.6 Entrada de evidências e de questionamento 78
7.4 Avaliação dos nós de contexto 78
7.4.1 Uso do PowerLoom 81
7.5 Regras de consistência 85
7.5.1 Definições de variáveis aleatórias 85
7.5.2 Global 86
7.5.3 Local 86
7.6 Tipo, tipo ordenável e recursão 88
7.7 Algoritmo de geração de SSBN 90
7.8 Entrada/Saída dos arquivos MEBN 92
7.8.1 Arquivo UBF - UnBBayes *File* 92
7.8.2 Implementação no UnBBayes 94

Capítulo 8 Estudo de Caso: *Star Trek* 97
8.1 MFrag `DangerToOthers` 101
8.2 MFrag `DangerToSelf` 102
8.3 MFrag `SensorReport` 103
8.4 MFrag `SRData` 104
8.5 MFrag `Starship` 104
8.6 MFrag `StarshipData` 106
8.7 MFrag `StarshipExistence` 107
8.8 MFrag `Zone` 108
8.9 *Query* `HarmPotential(!ST4,!T3)` 109

Capítulo 9 Conclusão 113
9.1 Limitações e trabalho futuro 114

Referências **116**

Lista de Figuras

2.1 Definição BNF da Lógica de Primeira Ordem. 24
2.2 Exemplo *family-out*. 27
2.3 Modelagem *Star Trek*. 28

3.1 Principais conceitos de PR-OWL. 32
3.2 Elementos de uma ontologia probabilística PR-OWL. 32

4.1 MFrag. 34
4.2 Pseudocódigo. 36
4.3 Instanciação de MFrag. 37
4.4 Exemplo de recursividade no modelo do *Star Trek*. 38
4.5 Exemplo de SSBN. 39
4.6 Exemplo de MTeoria no modelo do *Star Trek*. 40
4.7 Flexibilidade da MEBN. 41
4.8 Exemplo complexo de SSBN. 42

5.1 Arquitetura do UnBBayes. 43
5.2 Modelagem de classes para BN e ID. 46
5.3 Modelagem de classes para MSBN. 48

6.1 Tela de edição de classes/indivíduos no Protégé. 53
6.2 Tela de edição de propriedades no Protégé. 54
6.3 Esquema de acoplamento do Protégé-Owl com o Jena. 56

7.1 Modelagem de classes para MTeoria. 58
7.2 Modelagem de classes para os nós de uma MEBN. 59
7.3 MFrag `Starship Data`. 60
7.4 Modelagem de classes para MFrag. 61
7.5 Gramática para o pseudocódigo da tabela do nó residente. 62
7.6 Modelagem de classes do compilador do pseudocódigo. 63
7.7 Arquitetura de classes da GUI. 65
7.8 *Desktop* multi-telas do UnBBayes. 67
7.9 Diagrama de seqüência de criação dos objetos da GUI. 68
7.10 Componentes da tela de edição de MEBN. 69
7.11 Painel de edição de MEBN - `MEBNEditionPane`. 70
7.12 Classes utilizadas nos painéis de edição de MEBN. 72
7.13 Edição de MFrag. 73
7.14 Painel de edição de argumentos. 74

7.15 Edição de nó residente de uma MFrag. 75
7.16 Edição de nó de entrada de uma MFrag. 76
7.17 Edição de nó de contexto de uma MFrag. 76
7.18 Edição de entidades de uma MTeoria. 77
7.19 Edição de variável ordinária de uma MFrag. 77
7.20 Editor de CPT para nó residente. 78
7.21 Entrada de evidências das entidades. 79
7.22 Entrada de evidência - seleção do nó residente desejado. 79
7.23 Preenchendo informações do nó residente específico. 80
7.24 Questionamento - seleção do nó residente. 80
7.25 Questionamento - definição dos argumentos. 80
7.26 Modelagem do uso do PowerLoom. 81
7.27 Nós de contexto da MFrag `Starship`. 84
7.28 MFrags `Entity Type`, `IsA` e `TimeStep`. 88
7.29 Definição de entidade tipada no UnBBayes. 89
7.30 Projeto principal de entidade e tipo no UnBBayes. 89
7.31 Recursão no nó `DistFromOwn` na MFrag `Starship`. 90
7.32 Modelagem de IO de MEBN. 95

8.1 MTeoria *Star Trek* proposta em [13] adaptada. 99
8.2 MTeoria do *Star Trek* editada no UnBBayes. 100
8.3 MFrag `DangerToOthers` modelada no UnBBayes. 101
8.4 MFrag `DangerToSelf` modelada no UnBBayes. 102
8.5 MFrag `SensorReport` modelada no UnBBayes. 103
8.6 MFrag `SRData` modelada no UnBBayes. 104
8.7 MFrag `Starship` modelada no UnBBayes. 105
8.8 MFrag `StarshipData` modelada no UnBBayes. 106
8.9 MFrag `StarshipExistence` modelada no UnBBayes. 107
8.10 MFrag `Zone` modelada no UnBBayes. 108
8.11 SSBN gerada como resultado da *query* `HarmPotential(!ST4,!T3)`. 110
8.12 Entrando com evidência `StarshipClass(!ST4) = WarBird`. . . . 111
8.13 Resultado da propagação da evidência `StarshipClass(!ST4) = WarBird`. 112

Lista de Tabelas

2.1 CPT para o nó `Magnetic Disturbance Report`. 28

7.1 `BuiltInRV` implementadas no UnBBayes. 58

Capítulo 1

Introdução

Neste Capítulo, é apresentada a especificação geral do problema abordado, sua relevância e as áreas de pesquisa relacionadas.

1.1 Definição do Problema

Na atualidade, constata-se que a informação se tornou um recurso cada vez mais valorizado como viabilizador de decisões e de processos de conhecimento e inteligência nos mais diferentes campos. Um aspecto problemático da cultura de nosso tempo é o fenômeno da explosão informacional, onde a grande quantidade de informações produzidas e disponibilizadas por diferentes atividades sociais passa a dificultar sua identificação, acesso e utilização. Para que possa ser útil, a informação relevante para um dado problema precisa estar disponível no tempo certo. De nada adianta a informação existir, se quem dela necessita não sabe da sua existência, ou se ela não puder ser encontrada. A "revolução do conhecimento" será vista, no futuro, como a fase onde essa tarefa árdua e manual de identificação, acesso e utilização da informação for atribuída com sucesso aos computadores, permitindo que os seres humanos mudem seu foco das atividades dirigidas a dados para atividades dirigidas a conhecimento. Essa nova forma de atuação é chamada de "paradigma do conhecimento" [12]. Nesse contexto surge a Web Semântica através de um esforço de colaboração entre o Consórcio World Wild Web (W3C) e um grande número de pesquisadores e parceiros industriais para estender a Web atual com o intuito de prover um *framework* comum, permitindo que o dado seja compartilhado e reutilizado entre aplicações, corporações e comunidades. De acordo com o W3C [21], ontologia é vista como sendo a tecnologia que irá prover a base para a construção da Web Semântica. Ontologias contêm um conjunto comum de termos para descrever e representar o domínio de uma forma que permita ferramentas automáticas usarem dados armazenados de uma maneira mais contextualizada, agentes de software inteligentes atingirem um melhor gerenciamento de conhecimento e uma série de outras possibilidades trazidas pelo uso padronizado e mais intensivo de meta-dados. Enquanto a World Wide Web (WWW) atual se baseia em protocolos apenas sintáticos, a Web Semântica adiciona anotações de meta-dados como uma forma de consciência semântica para melhorar a interoperabilidade entre recursos Web [12]. O atual estágio da Web

Semântica é baseado na lógica clássica. Assim a linguagem de ontologia da Web (OWL), uma recomendação do W3C [31], não possui suporte para lidar com a incerteza. Essa característica constitui uma grande desvantagem para uma tecnologia projetada para a Web, um ambiente complexo, onde simplesmente não é possível ignorar a existência de informação incompleta. Por exemplo, do ponto de vista sintático, o sobrenome Carvalho é a mesma coisa que a árvore carvalho. Do ponto de vista semântico é necessário que esses dois termos possam ser diferenciados e é exatamente neste ponto que as ontologias têm um papel fundamental. No entanto, quando comparando duas ontologias com o termo "Carvalho", aplicativos que utilizam inferências determinísticas considerarão esse termo como uma árvore ou como uma entidade indefinida (que é diferente de não ser uma árvore), ou seja, não há como definir um grau intermédiario de pertinência, pois não possuem suporte para tratar a incerteza. Isso é perfeitamente aceitável quando a informação completa está disponível, que é geralmente o caso de premissas de mundo fechado, mas que é raro no caso de premissas de mundo aberto, onde a regra é haver informação incompleta. Essas premissas de mundo aberto são geralmente encontradas na maioria das situações reais onde a existência de informação incompleta é fonte de incerteza. Portanto, nesse contexto, para se representar conhecimento incerto e se raciocinar com base nele é necessário se dispor de um formalismo para tratar a incerteza. O problema estudado nessa pesquisa é o da representação de conhecimento e a realização de inferências em ontologias Web com incerteza, com o uso de redes probabilísticas. Redes probabilísticas - modelos probabilísticos baseados em representações gráficas das dependências probabilísticas - constituem uma alternativa bastante promissora para representar conhecimento e raciocinar com base nele em mundos abertos porque representam a incerteza através de probabilidade, o que permite manipulá-la com base em princípios bem fundamentados e com semântica conhecida. No entanto, embora redes probabilísticas como redes bayesianas (BN) sejam capazes de tratar incerteza, elas possuem algumas limitações que as tornam inadequadas para a Web Semântica, a qual requer uma capacidade representacional compatível com a obtida por linguagens baseadas em lógica clássica. Dentre essas limitações, são apresentadas duas que serão mais detalhadas no Capítulo 2:

1. o fato do número de variáveis ter de ser previamente conhecido;

2. a falta de suporte à recursividade.

Para resolver essas limitações, foi proposto um *framework* bayesiano para ontologias probabilísticas [12] que pode servir como base para serviços que realizem inferências sob incerteza na Web Semântica. Esse *framewok* é composto de uma linguagem de ontologia probabilística, PR-OWL, baseada em uma lógica probabilística de primeira ordem, *Multi-Entity Bayesian Network* (MEBN). A Linguagem PR-OWL (*Probabilistic OWL*) é uma extensão da linguagem OWL (*Ontology Web Language*) visando permitir a representação de ontologias probabilísticas na Web. Já MEBN é uma junção de redes bayesianas com lógica de primeira ordem - FOL (do inglês, *First Order Logic*) - que permite utilizar sentenças lógicas para definir contextos de validade para subredes bayesianas que juntas definem

uma única rede bayesiana. Com o uso de MEBN é possível realizar inferências probabilísticas com entidades representadas via PR-OWL.

1.2 Objetivos

O problema da representação de ontologias com incerteza tem despertado o interesse de diversos pesquisadores da área, em nível internacional. Recentemente foram realizadas três edições do workshop URSW (*Uncertainty Reasoning for the Semantic Web*) [17, 11, 3] dedicadas a esse tema, de 2005 a 2007, todas elas nas edições da conferência ISWC (*International Semantic Web Conference*). Além disso, como resultado do workshop URSW 2006, foi proposta a criação do grupo *W3C Uncertainty Reasoning for the World Wide Web Incubator Group* (URW3-XG) [30] que tem como escopo uma melhor definição do desafio de representar e efetuar inferências com informação incompleta no âmbito da Web Semântica. Esta proposta foi aprovada em 1º de março de 2007 e a partir desta data foi composto um grupo internacional (*W3C Incubator Group*) que deverá estudar o assunto e apresentar propostas no prazo de um ano. Como uma contribuição para os esforços que estão sendo desenvolvidos pelo URW3-XG, a presente pesquisa tem os objetivos abaixo.

1.2.1 Objetivo geral

Avaliar, de forma experimental, a utilização da linguagem PR-OWL para representar ontologias probabilísticas e da lógica probabilística MEBN para realizar inferências probabilísticas com base em entidades PR-OWL.

1.2.2 Objetivos específicos

1. Propor refinamentos à linguagem PR-OWL visando facilitar a implementação de uma ferramenta para construção de ontologias probabilísticas em PR-OWL.

2. Propor refinamentos à MEBN visando facilitar a implementação de uma ferramenta para realizar inferências probabilísticas com entidades PR-OWL.

3. Realizar a primeira implementação mundial de uma ferramenta visual que facilite criar e manipular entidades PR-OWL e realizar inferências com elas, com base na lógica MEBN.

4. Validar, experimentalmente, a utilização de PR-OWL e MEBN através da construção de, pelo menos, um modelo baseado nesses formalismos que possa ser utilizado para encontrar soluções para o problema enfocado.

1.3 Áreas de Pesquisa Relacionadas

Vários formalismos foram propostos com o objetivo de superar as limitações presentes em BN. No Capítulo 3 serão apresentadas dois grupos de pesquisas dife-

rentes, com objetivo de combinar lógica com probabilidade: extensão probabilística para lógica descritiva; e extensões probabilísticas para OWL. Além disso, é também apresentada a linguagem proposta em [12], a PR-OWL, que se baseia no uso de MEBN. Sendo que esta abordagem (PR-OWL e MEBN) será o foco deste trabalho.

1.4 Estrutura da Dissertação

O Capítulo 2 apresenta algumas definições básicas para o entendimento da solução proposta. O Capítulo 3 aborda o estado da arte das soluções propostas para o problema apresentado. O Capítulo 4 foca a lógica probabilística de primeira ordem MEBN. O Capítulo 5 apresenta uma visão da arquitetura do UnBBayes e maiores detalhes sobre a versão 2 do UnBBayes. O Capítulo 6 descreve as principais bibliotecas de terceiros utilizadas pelo UnBBayes na implementação de MEBN/PR-OWL. O Capítulo 7 introduz a modelagem e implementação de MEBN e PR-OWL no UnBBayes. O Capítulo 8 apresenta um estudo de caso. Por fim, o Capítulo 9 contém conclusões realçando as contribuições dessa pesquisa e sugestões de trabalhos futuros. Além dos capítulos apresentados nessa dissertação, os seguintes arquivos serão disponibilizados no CD anexo:

- dissertação em meio digital;
- modelo PR-OWL do *Star Trek* modelado no UnBBayes (arquivo .ubf e .owl);
- relatório de geração de uma SSBN;
- a linguagem PR-OWL (em formato .owl);
- principais artigos utilizados nessa pesquisa;
- documentação e código fonte do UnBBayes;
- executável do UnBBayes.

Capítulo 2

Conceitos Básicos

Este Capítulo apresenta os conceitos básicos da área de raciocínio probabilístico e Web Semântica e aborda os assuntos mais diretamente relacionados com o objetivo desta pesquisa. A seção 2.1 apresenta uma breve definição de ontologia; a seção 2.2 trata da linguagem OWL; a seção 2.3 introduz alguns conceitos gerais de lógica e, por fim, a seção 2.4 aborda as redes bayesianas e suas principais limitações para a SW. A Web Semântica é um projeto vislumbrado por Tim Berners-Lee, criador do padrão HTML e da WWW, sob os auspícios da W3C [1]. 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. Os documentos da Web tradicional são construídos em linguagens com pouco ou nenhum suporte para inferência semântica (pois os algoritmos podem apenas utilizar informações sintáticas), como o HTML e a 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. Apesar da estrutura de *tags* permitir que sejam providas regras sintáticas para extrair, transformar e trocar dados, estas informações são insuficientes para aplicações inteligentes por computadores, cabendo portanto ao homem tarefas que envolvam o conhecimento implicado pelos dados. Com a evolução dos computadores e o crescimento da quantidade de informações disponíveis na Web, tornou-se necessário que se começasse a planejar formas de fazer com que o próprio computador realizasse estes tipos de tarefas. Segundo Berners-Lee, Hendler e Lassila [2], os computadores necessitam ter acesso a coleções estruturadas de informações (dados e meta-dados) e de conjuntos de regras de inferência que ajudem no processo de dedução automática para que seja vializado o raciocínio automático. Agentes inteligentes são programas que realizam buscas e processamentos nas informações disponíveis de forma autônoma visando oferecer respostas para o 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 um horário para a consulta, interagindo diretamente com o agente inteligente do médico, dado um horário disponível em que o cliente possa comparecer ao consultório. A Web atual não permite a independência máquina-homem necessária para efetuar operações como esta. Tais agentes inteligentes devem ser capazes de identificar o significado exato de uma palavra e as relações lógicas entre várias palavras. Para

que os computadores entendam o conteúdo da Web é necessário que eles consigam ler dados estruturados e tenham acesso a conjuntos de regras que os ajudem a conduzir seus raciocínios. As novas páginas da Web terão que ser escritas em uma nova linguagem e serem entendidas por diversos sistemas.

2.1 Ontologia

De acordo com o W3C, as ontologias são a semente para a criação da Web Semântica. Ontologia é um termo emprestado da filosofia — que se refere à ciência de descrever os tipos de entidades no mundo e como elas estão relacionadas — e pode ser até mesmo encontrado nos estudos de Aristóteles. No entanto, seu uso no domínio de sistemas de informação é relativamente novo, sendo utilizado pela primeira vez nesse domínio em 1967 [35]. Uma ontologia é uma representação formal de um domínio, descrevendo as entidades presentes, as propriedades destas, os relacionamentos entre estas e os processos e eventos que acontecem entre estas entidades. O modelador deve montar a ontologia capturando do domínio todas as informações relevantes para a representação deste, tomando cuidado para inserir informações suficientes possibilitando aos computadores poderem processar corretamente (coisas óbvias para os homens podem requerer uma descrição mais detalhada para as máquinas). Estas ontologias devem ser compartilhadas para que haja uma uniformidade na descrição dos domínios (no exemplo citado, o agente inteligente do cliente deverá compartilhar a ontologia com os agentes inteligentes dos médicos, caso contrário, não será possível fazer a interação necessária). A melhor solução para evitar incompatibilidades seria definir uma ontologia única que cobrisse todas as áreas do conhecimento, porém, devido ao tamanho atual da Web, tal ontologia necessitaria de um conjunto enorme de descritores, tornando a catalogação exaustiva.

Definição 1 [12]: Uma ontologia é uma representação formal de conhecimento explícita sobre um domínio de aplicação. Isso inclui:

1. tipos de entidades que existem no domínio;
2. propriedades dessas entidades;
3. relacionamentos entre essas entidades;
4. processos e eventos que ocorrem com essas entidades.

Na Definição 1 o termo entidade se refere a qualquer conceito (real ou fictício, concreto ou abstrato) que pode ser descrito e usado para raciocínio em um domínio de aplicação.

2.2 OWL

As ontologias devem descrever a informação em um formato que possa ser entendido tanto por humanos quanto por computadores. O W3C propôs a linguagem

de ontologia para Web (OWL) como uma extensão da linguagem do *framework* de descrição de recursos (RDF). A OWL é baseada em lógica de primeira ordem, portanto tendo um grande poder de expressividade.

Segundo o W3C [21], a OWL é uma linguagem para definir e instanciar ontologias Web. Dado uma ontologia, a semântica formal de OWL especifica como derivar suas conseqüências lógicas, por exemplo, fatos que não estão presentes literalmente na ontologia, mas subentendido pela semântica por implicação. Essas implicações podem ser baseadas em um único documento ou em diversos documentos distribuídos combinados segundo mecanismos OWL definidos.

A Web Semântica deverá ser construída com base na habilidade do XML de definir esquemas para *tags* customizadas e na flexibilidade do RDF para representar dados. De acordo com o W3C [21], o primeiro nível acima do RDF necessário para Web Semântica é uma linguagem ontológica que pode descrever formalmente o significado de terminologias usadas em documentos Web. Para que as máquinas possam realizar tarefas de raciocínio mais úteis nesses documentos, a linguagem deve ir além da semântica básica do esquema do RDF.

A OWL foi projetada para atender a essa necessidade. OWL faz parte de um conjunto crescente de recomendações da W3C relacionado com a Web Semântica [21].

XML fornece uma camada de sintaxe para documentos estruturados, mas não impõe restrições semânticas no significado desses documentos.

Esquema XML é uma linguagem para restringir a estrutura de documentos XML e também estender XML com tipos de dados.

RDF é um modelo de dados para objetos ("recursos") e relacionamentos entre eles, fornecendo uma semântica simples. Esses modelos de dados podem ser representados em uma sintaxe XML.

Esquema RDF é um vocabulário para descrever propriedades e classes de recursos RDF com semântica para hierarquias generalizadas de tais propriedades e classes.

OWL acrescenta restrições para a definição mais precisa de propriedades e classes. Entre estas: relacionamentos entre classes (disjunta, por exemplo); cardinalidade (apenas uma, por exemplo); igualdade; e classes enumeradas.

OWL possui três versões [21] de sua linguagem com sua expressividade maior na medida que sua decidibilidade diminui. As três versões, começando pela menos expressiva e mais decidível até a mais expressiva e menos decidível, são:

OWL Lite atende as necessidades de usuários que precisam de classificação hierárquica e restrições simples.

OWL DL atende as necessidade de usuários que querem máxima expressividade, mas mantendo completude computacional (todas conclusões são garantidas serem computáveis) e decidibilidade (todas as computações finalizarão em um tempo finito). OWL DL é assim chamada devido a sua correspondência com lógica descritiva.

OWL Full atende as necessidades de usuários que desejam máxima expressividade e a liberdade sintática de RDF com nenhuma garantia computacional.

Cada versão de OWL é uma extensão da versão predecessora mais simples, tanto no que pode ser expressado legalmente, quanto no que pode ser validamente concluído. As seguintes relações são verdadeiras [21]:

- toda ontologia OWL Lite legal é uma ontologia OWL DL legal;
- toda ontologia OWL DL legal é uma ontologia OWL Full legal;
- toda conclusão OWL Lite válida é uma conclusão OWL DL válida;
- toda conclusão OWL DL válida é uma conclusão OWL Full válida.

A OWL não possui, no entanto, uma maneira padronizada para expressar incertezas, implicando em limitações ao tentar representar as informações incompletas inerentes a diversos domínios do mundo real. Além disso, um agente terá dificuldades para inferir qual o significado de uma palavra quando esta puder ter significados diferentes. Por exemplo: a palavra "Grape" pode se referir ao sobrenome de uma pessoa (ex.: John Grape) ou à fruta (uva). Se torna necessário um mecanismo que permita a representação de incertezas e que os agentes façam inferências.

2.3 Lógica

A lógica possui vários formalismos que diferem no poder de expressividade. Apenas para justificar a escolha pela lógica de primeira ordem nessa pesquisa, serão avaliadas as outras lógicas que possuem elevado destaque na Inteligência Artificial (IA).

A *lógica proposicional* adota a premissa de que o mundo é composto por fatos, sendo o fato sua unidade básica. Conforme definido em [33], fatos são verdades em algum mundo relevante. Ora, apesar de resolver problemas particulares, a lógica proposicional falha por não poder representar generalizações, sendo portanto bastante limitada.

Uma *lógica de primeira ordem* (FOL), por outro lado, vê o mundo como um conjunto de objetos, possuindo estes, identidades individuais e propriedades que os distinguem de outros objetos.

As *lógicas de mais alta ordem* (HOL) se diferenciam por, além de teorizar sobre objetos, como a FOL, teorizar sobre relações e funções sobre estes objetos. Apesar da HOL ter mais poder expressivo, há ainda pouco conhecimento de como raciocinar efetivamente com sentenças HOL, além do problema geral ser indecidível [34].

Um quarto tipo importante de lógica é a *lógica modal* que, além de permitir raciocinar sobre a validade das expressões, também permite raciocinar sobre qualidades não relativas à validade destas, como por exemplo, necessidade e possibilidade. A lógica modal é baseada na semântica dos mundos possíveis, a qual permite que algumas expressões sejam verdade no mundo atual, mas não em

todos os mundos possíveis (verdade contingente) em oposição às sentenças necessariamente verdadeiras que são verdade em todos os casos aonde se verificam as condições de validade; e às sentenças necessariamente falsas que são sempre falsas.

Apesar das lógicas de alta ordem e da lógica modal permitirem descrições mais complexas, há um nível muito mais alto de dificuldade na modelagem do mundo em comparação com a FOL, além de ainda não haver sistemas de provas eficientes para estas, tornando a FOL o formalismo mais estudado e o mais utilizado.

A Figura 2.1 mostra, em notação BNF (do inglês, *Backus-Naur Form*), a definição dos elementos da FOL. Esta é uma adaptação da definição encontrada em [34], mesclada com algumas alterações para adaptá-la a definição feita em [27].

BNF da Lógica de Primeira Ordem

```
01. <Fórmula> ::= <Fórmula Atômica>
02.   | <Fórmula> <Conectivo> <Fórmula>
03.   | <Quantificador> <Variável>, ... <Fórmula>
04.   | ~ <Fórmula>
05.   | ( <Fórmula> )
06. <Fórmula Atômica> ::= <Predicado> ( <Termo>, ... )
07. <Termo> ::= <Função> ( <Termo>, ... )
08.   | <Constante>
09.   | <Variável>
10. <Conectivo> ::= → | ^ | v
11. <Quantificador> ::= ∀ | ∃
12. <Constante> ::= A | X | Maria | ....
13. <Variável> ::= a | x | s | ...
14. <Função> ::= PaiDe | RaizQuadradaDe | ...
15. <Predicado> ::= Antes | TemCor | Chovendo | ...
```

Figura 2.1: Definição BNF da Lógica de Primeira Ordem.

Constantes são símbolos primitivos (representam objetos do domínio). A interpretação deve especificar qual é a entidade especificada. São escritas iniciadas com letra maiúscula ou com um número. Ex.:

$$A$$

$$Maria$$

$$1888$$

Predicados referem-se a uma relação particular no modelo (especificado pela interpretação). Iniciam com letra maiúscula. Cada predicado é definido por um conjunto de tuplas de objetos que a satisfazem. Ex.:

$$Irmao(Eduardo, Pedro)$$

Funções são relações especiais, onde um dado objeto é relacionado exatamente com outro objeto. Ex.:

$$Pai$$

(cada pessoa tem exatamente uma pessoa que é seu pai). Funções iniciam com letra maiúscula e podem ser utilizadas para se referir a objetos particulares sem utilizar seus nomes, como por exemplo em

$$PaiDe(Maria)$$

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 estes últimos termos complexos, onde os argumentos da função serão dispostos em uma lista de termos parentetizada, com os termos separados por vírgulas.

Fórmulas Atômicas representam fatos. É formado por um predicado seguido por uma lista parentetizada de termos. Ex.:

$$Irmao(Eduardo, Pedro)$$

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.

Fórmulas Complexas são construídas a partir de sentenças atômicas utilizando os símbolos lógicos. Ex.:

$$Irmao(Eduardo, Pedro) \wedge Irmao(Maria, Joana)$$

Quantificadores são utilizados para expressar propriedades de objetos representados por variáveis (iniciam com letra minúscula), ao invés de ter que enumerar os objetos por nome, como proposto na lógica proposicional. A FOL contém dois quantificadores padrões:

Quantificador Universal é utilizado para especificar sentenças que são validas para todas os objetos. Ex.:

$$\forall x(Gato(x) \rightarrow Mamifero(x))$$

Quantificador Existencial é utilizado para especificar sentenças que são válidas para algum objeto, sem no entanto, nomear este. Ex:

$$\exists x(Irma(x, Spot)) \wedge (Gato(x))$$

Igualdade é utilizada para dizer que dois termos se referem ao mesmo objeto. A igualdade pode ser tratada como um predicado. Ex.:

$$Pai(John) = Henry$$

Para aplicar a lógica de primeira ordem 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 coisa segue de uma contradição.

Kurt Gödel provou que a FOL é um sistema lógico completo, ou seja, todas as sentenças válidas podem ser provadas. Foram definidos e estudados diversos sistemas de provas, entre eles, a resolução com Skolenização (uma descrição detalhada pode ser encontrada em [34]) e a dedução natural (um mecanismo de dedução natural pode ser encontrado em [18]).

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, tempo, etc, permitindo que o modelador possua uma base por onde começar a modelagem do domínio estudado.

2.4 BN

Uma rede bayesiana (BN) é um grafo acíclico direcionado (DAG) que representa uma função de distribuição de probabilidades conjunta (JPD) de variáveis que modelam certo domínio de conhecimento. Ela é constituída de um DAG, de variáveis aleatórias (também chamadas de nós da rede), arcos direcionados da variável pai para a variável filha e uma tabela de probabilidades condicionais (CPT) associada a cada variável.

A Figura 2.2 apresenta um exemplo de uma rede bayesiana bastante conhecida na literatura, extraída de Charniak [10], página 51 a 53.

Nesse exemplo, suponha que se queira determinar se a família está em casa ou se ela saiu. Pelo grafo, pode-se perceber que o fato da luz da varanda estar acesa e do cachorro estar fora de casa são indícios de que a família tenha saído. Como essa inferência está sendo feito com incertezas, o conhecimento dos valores dessas variáveis não é determinado, mas sim estimado. Quanto maior for a probabilidade da luz da varanda estar acesa, maior é a probabilidade da família ter saído. Por outro lado, caso se saiba com certeza que o cachorro não está fora de casa (ou seja, probabilidade 1), então a probabilidade da família ter saído é diminuída (o quanto essa probabilidade é diminuída é definido na CPT). As CPT podem ser construídas por um especialista do domínio ou estimadas a partir de uma base de dados históricos.

Uma das vantagens de se trabalhar com BN é o fato de não ser preciso especificar a distribuição de probabilidades de todo o domínio, envolvendo todas as variáveis. Isso é possível devido a existência de independências condicionais. De fato, Judea Pearl [32] provou que somente é necessário representar a CPT de uma variável, condicionada aos seus pais no DAG.

As redes bayesianas possuem diversas aplicações. Entre elas, estão: diagnósticos de defeitos em processadores (técnica utilizada pela Intel), exames patológicos, resolvedor de problemas do Microsoft Word, aplicações militares, filtragem

Figura 2.2: Exemplo *family-out*.

de *spam*, etc.

2.4.1 Desvantagens

Embora as redes bayesianas tenham muitas vantagens e sejam muito utilizadas atualmente, há dois problemas que limitam suas áreas de aplicação: o fato do número de variáveis ter de ser previamente conhecido e a falta de suporte à recursividade.

Para exemplificar as deficiências será utilizado o exemplo *Star Trek* apresentado em [12]. Esse exemplo é baseado na popular série de TV *Star Trek*. No entanto, os exemplos apresentados aqui foram construídos de tal forma que possam ser acessíveis para qualquer pessoa familiar com uma ficção científica relacionada ao espaço. Primeiramente, será feita uma exposição narrando uma versão simples de como identificar espaçonaves inimigas.

Nesse problema simplificado, a principal tarefa é modelar o problema de detectar espaçonaves Romulanas (consideradas aqui como hostis pela Federação Unida dos Planetas) e avaliar o nível de perigo que estas representam para a própria espaçonave, denominada Enterprise. Todas as demais espaçonaves são consideradas como amigas ou neutras. A detecção de espaçonave é feita através dos sensores da Enterprise, que podem determinar e discriminar espaçonaves com uma precisão de 95%. No entanto, espaçonaves Romulanas podem estar em *cloak mode*, que as tornam invisíveis aos sensores da Enterprise. Até mesmo para as tecnologias de

sensores mais atuais, a única pista de que há uma espaçonave em modo invisível nas proximidades é um pequeno distúrbio magnético causado pela enorme quantidade de energia necessária para que fique invisível aos sensores. A Enterprise tem um sensor de distúrbio magnético, mas é muito difícil distinguir um distúrbio magnético gerado pelo modo invisível, das variações magnéticas normais do ambiente.

A rede apresentada na Figura 2.3 modela o exemplo explicado acima, levando também em consideração as características do ambiente no espaço onde a ação acontece.

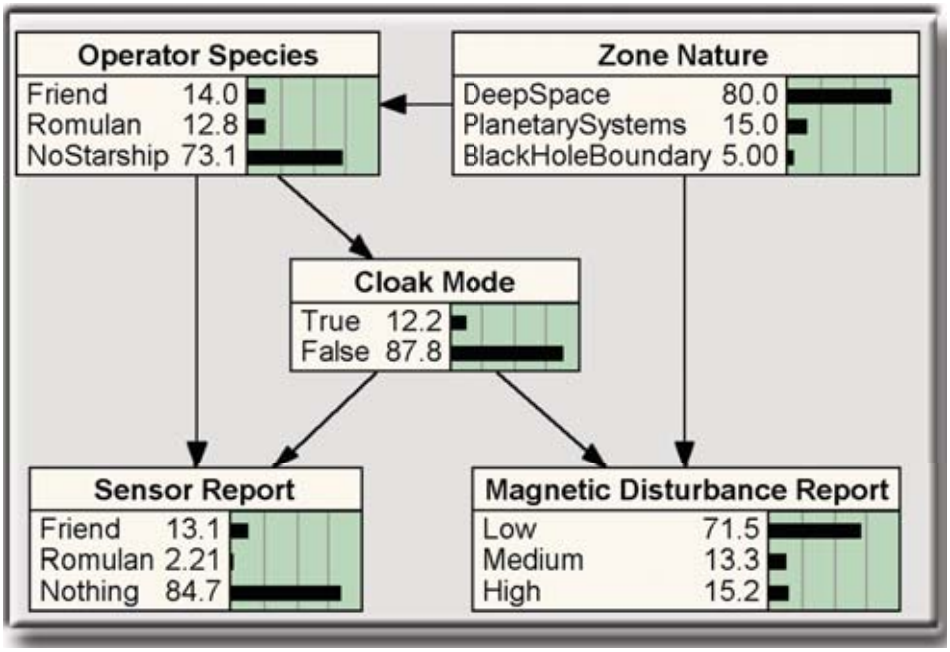

Figura 2.3: Modelagem *Star Trek*.

As variáveis aleatórias assumem estados discretos. O nó `Zone Nature` é o nó raiz (não possui nenhum nó pai). Os outros nós são condicionados pelos nós pais, conforme mostra o DAG, sendo esta influência quantificada por uma CPT. Por exemplo, a Tabela 2.1 extraída de Costa [12], exemplifica a probabilidade de cada estado do nó `Magnetic Disturbance Report` para cada combinação possível dos estados dos seus nós pais.

Tabela 2.1: CPT para o nó `Magnetic Disturbance Report`.

Zone Nature	**Cloak Mode**	**Mag. Disturb. Rep.**		
		Low	Medium	High
Deep Space	True	80.0	13.0	7.0
	False	85.0	10.0	5.0
Planetary Systems	True	20.0	32.0	48.0
	False	25.0	30.0	45.0
Black Hole Boundary	True	5.0	10.0	85.0
	False	6.9	10.6	82.5

Esta modelagem simples funciona apenas quando houver no máximo uma nave nas proximidades. Caso haja duas naves se aproximando simultaneamente, será necessário utilizar um novo modelo. Ora, em uma situação real não se sabe a priori quantas naves vão se aproximar, tornando inviável a modelagem utilizando redes bayesianas, pois seria necessário um modelo para cada possibilidade (note que neste caso o número de possibilidades é infinito), e não haveria como determinar qual modelo utilizar em um dado momento. As redes bayesianas 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 uma nave esteja nas proximidades da Enterprise em um determinado momento, validando o modelo proposto. Suponha que no momento *T1*, a Enterprise esteja viajando pelo espaço profundo (`Deep Space`) e que o seu sensor indique que não há nenhuma nave nas proximidades. Suponha que em um momento *T2* o sensor de distúrbio magnético reporte a ocorrência de um alto distúrbio. De acordo com a CPT mostrada na Tabela 2.1, as chances são de 7/5 de que este distúrbio tenha sido causado por uma nave em *cloak mode* na vizinhança. Porém, esta ocorrência não é garantida, pois os distúrbios podem ser apenas anomalias comumente encontradas no ambiente. Uma forma de confirmar a ocorrência de tal nave é comparar as medições seguintes do sensor, pois é natural que distúrbios anormais de ambiente mostrem flutuações aleatórias, enquanto, por outro lado, um distúrbio causado por uma espaçonave mostrará um padrão temporal característico: quando há uma espaçonave em *cloak mode* na vizinhança, o estado do `Magnetic Disturbance Report` em um dado momento dependerá de seu estado prévio. Ora, as redes bayesianas são estáticas, impedindo a recursão necessária para resolver problemas deste tipo.

Estes dois exemplos mostram dificuldades que um modelador poderá enfrentar ao tentar representar o mundo utilizando redes bayesianas. Estas limitações impedem que redes bayesianas sejam utilizadas, em sua forma pura, para representar problemas complexos, limitando o seu uso apenas a situações especificas e simples.

Capítulo 3

Fundamentos

Diversas linguagens surgiram para expandir a expressividade de BN. Duas vertentes diferentes de pesquisa, com objetivo de combinar lógica com probabilidade, serão tratadas nas próximas seções. Na seção 3.1 é apresentada uma discussão sobre extensão probabilística para lógica descritiva e na seção 3.2 extensões probabilísticas para OWL. Finalmente, na seção 3.3 é apresentada a expansão sugerida em [12] para a linguagem OWL, a PR-OWL, que se baseia no uso de MEBN.

3.1 Extensão probabilística para lógica descritiva

A maioria das extensões probabilísticas para as ontologias são baseadas em lógica descritiva (DL). DL é um subconjunto de FOL que provê uma boa combinação de decidibilidade e expressividade e é a base para OWL-DL. Algumas extensões de lógica descritiva são: lógica descritiva probabilística [22] e [23]; P-SHO(D) [20]; P-Classic [25].

Lógicas descritivas são muito eficazes e eficientes para problemas de classificação e assunção. No entanto, de acordo com [12] sua habilidade para representar e raciocinar sobre outros tipos comuns de conhecimento é limitada. Um aspecto restritivo de linguagens DL é sua limitação para representar restrições nas instâncias que podem participar em um relacionamento. Como exemplo, imagine a situação onde é necessário representar que para uma nave ser uma ameaça para outra nave em uma situação específica, é obrigatório que dois indivíduos da classe nave envolvidos na situação sejam diferentes. Garantir que essas naves são diferentes em uma situação específica só é possível na DL, se efetivamente forem criados/especificados os indivíduos tangíveis nessa situação. De fato, afirmar que dois "preenchimentos" (i.e. os indivíduos da classe nave que vão "preencher os espaços" do conceito nave na afirmativa) não são iguais sem especificar seus valores respectivos exigiria construções que não podem ser expressadas em DL, pois tornaria a lógica indecidível [4].

3.2 Extensão probabilística para OWL

Há algumas poucas pesquisas com o objetivo de estender OWL para representar incerteza. Dentre elas, tem-se duas abordagens que envolvem o uso de redes bayesianas. A pesquisa sendo feita por Zhongli Ding e Yung Peng (2004) [12, página 55] e o trabalho de Gu *et al.* (2004) [12, página 56].

Em ambos os casos, a limitação decorrente do uso de representação do domínio baseada em redes bayesianas limitam a habilidade de expressar modelos probabilísticos mais complexos, restringindo as soluções para classes de problemas muito específicas. Mais informações podem ser encontradas em [12].

3.3 PR-OWL

A linguagem PR-OWL foi proposta por Costa (2005) como uma extensão para a linguagem OWL. A PR-OWL tem por base a lógica MEBN, por causa da sua expressividade: MEBN pode expressar uma distribuição de probabilidade sobre modelos de qualquer teoria FOL axiomatizável. Como conseqüência, não há garantia de que o raciocínio exato com uma ontologia PR-OWL será eficiente ou mesmo decidível [16].

A PR-OWL foi construída de forma a ser interoperante com ontologias não probabilísticas. O modelador pode utilizar os recursos da PR-OWL apenas para as partes da ontologia que necessitarem de suporte probabilístico. Porém, as partes probabilísticas da ontologia devem formar uma MTeoria (do inglês, *MTheory*) válida.

Por ser baseada na FOL, a PR-OWL não garante que os questionamentos (do inglês, *queries*) efetuados serão tratáveis ou mesmo decidíveis. A OWL possui o mesmo problema e, para solucioná-lo foram criadas três versões diferentes da linguagem, com crescente poder expressivo: OWL Lite, OWL DL e OWL Full (seção 2.2). Seguindo a mesma abordagem, uma PR-OWL Lite poderia ser criada, como sugerido em [16], com algumas restrições.

A Figura 3.1 (extraída de [16], página 7) apresenta os principais conceitos envolvidos na definição de uma MTeoria em PR-OWL. No diagrama as elipses representam as classes gerais, enquanto os arcos representam os principais relacionamentos entre as classes. Uma ontologia probabilística deve ter ao menos um individuo da classe MTeoria, sendo que este é ligado com um grupo de MFrags que coletivamente formam uma MEBN Teoria válida. As MFrags são compostas por nós, sendo que estes podem ser residentes, de contexto, ou de entrada (seção 4.2). Cada indivíduo da classe `Node` é uma RV e esta possui um conjunto mutuamente exclusivo e coletivamente exaustivo de estados possíveis. Cada RV possui uma distribuição de probabilidade condicional ou incondicional [16].

A Figura 3.2 (extraída de [16], página 8) ilustra os elementos principais da PR-OWL, suas subclasses e os elementos secundários necessários para representar uma MTeoria e as relações que são necessárias para expressar a estrutura complexa dos modelos probabilísticos bayesianos, usando a sintaxe do OWL. Para uma descrição detalhada da PR-OWL veja [12].

De forma análoga ao caso de arquivos HTML, um arquivo PR-OWL também

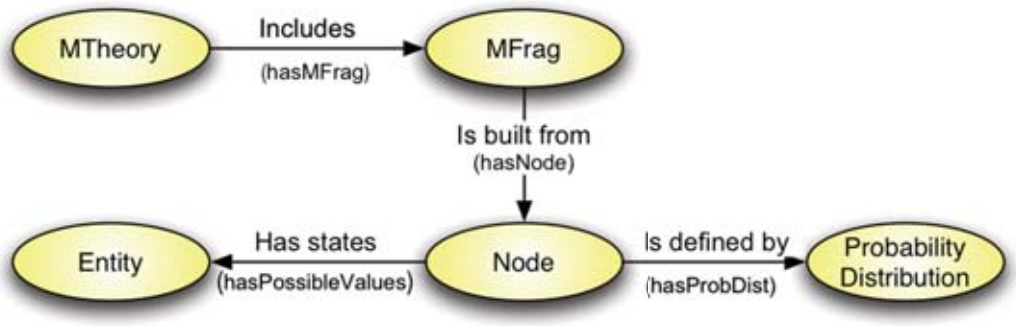

Figura 3.1: Principais conceitos de PR-OWL.

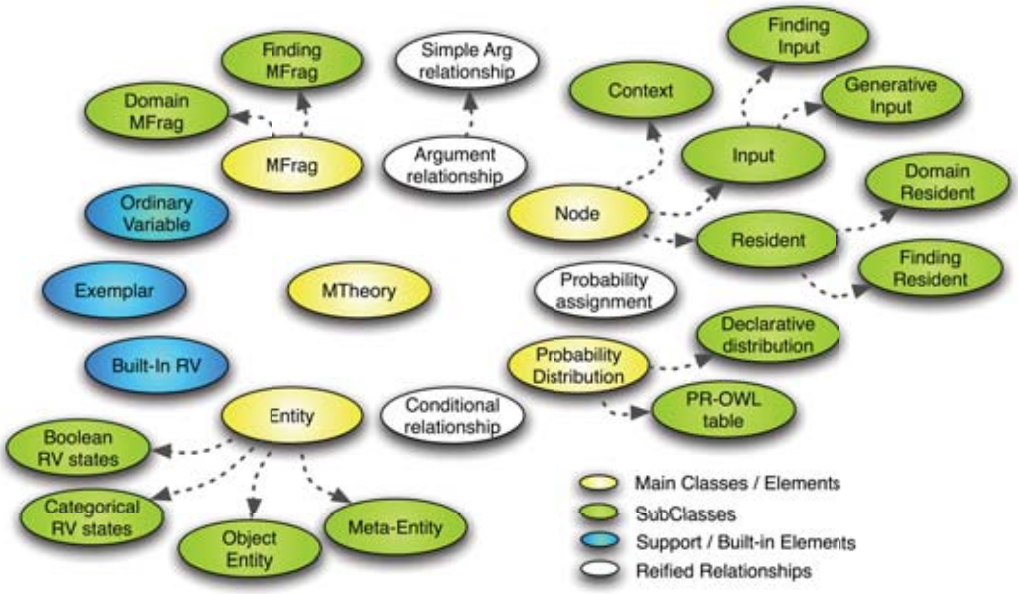

Figura 3.2: Elementos de uma ontologia probabilística PR-OWL.

é um arquivo texto. Portanto, construir MFrags e todos os seus elementos em uma ontologia probabilística, com um editor de textos ou mesmo um editor de ontologias como é o caso do software Protégé [19], é um processo cansativo e sujeito a erros, exigindo um conhecimento profundo da estrutura de dados do PR-OWL. Assim se torna necessário o desenvolvimento de um software que permite uma modelagem visual, ou um *plugin* para alguma das ferramentas bayesianas existentes. Um dos objetivos desta pesquisa é estender o UnBBayes [6] para permitir a edição de arquivos PR-OWL, permitindo que a MTeoria seja salva neste formato, fornecendo ao modelador um meio visual e intuitivo para a construção de ontologias probabilísticas.

Capítulo 4

Rede Bayesiana Multi-Entidade

Este Capítulo apresentará a definição dos principais elementos presentes nas redes bayesianas multi-entidades (MEBN). A seção 4.1 explica o que é uma MEBN. A seção 4.2, além de definir uma MFrag, apresenta os nós existentes em uma MEBN, que estão vinculados a uma MFrag. Já na seção 4.3 é apresentado a forma de se definir recursão em MEBN. Na seção 4.4 apresenta-se a definição de uma MTeoria. Por fim, a seção 4.5 apresenta a forma de se fazer inferência em MEBN e gerar uma SSBN.

Todas as figuras apresentadas nesse Capítulo foram extraídas de Costa (2005) e ilustram a aplicação de MEBN no domínio de ficção científica baseado na série de TV denominada *Star Trek*.

4.1 O que é MEBN?

As redes bayesianas multi-entidades são uma extensão das redes bayesianas, em que a lógica de primeira ordem é integrada. O domínio de conhecimento em MEBN é expresso por *MEBN Frament*s (MFrags), que são organizadas em *MEBN Theories* (MTeorias).

Uma MFrag representa uma distribuição de probabilidade condicional das instâncias de suas variáveis aleatórias residentes, dados os valores das instâncias de seus pais no grafo. Para que essa distribuição se aplique, as restrições de contexto do domínio devem ser satisfeitas.

Um conjunto de MFrags representa uma distribuição de probabilidade conjunta sobre um determinado número de instâncias de suas variáveis aleatórias. Uma das vantagens da utilização de MEBN em vez de redes bayesianas é que o número de instâncias de RV não tem um limite (podendo até tender ao infinito) e não precisa ser previamente conhecido já que as variáveis aleatórias são instanciadas dinamicamente, no instante em que for necessário.

Uma MTeoria é um conjunto de MFrags que satisfaz determinadas condições de consistência que garantem a existência de uma distribuição de probabilidade conjunta, única, sobre suas variáveis aleatórias.

4.2 MFrag

MEBN representa um domínio de conhecimento através das MFrags. Uma MFrag é organizada na forma de um DAG composto por nós (ou variáveis aleatórias) que representam as características das entidades do domínio e por arcos direcionados da variável pai para a variável filha, representando as relações entre as entidades.

Uma MFrag representa uma distribuição de probabilidade condicional das instâncias de suas variáveis aleatórias, dados os valores de seus pais, contanto que os nós de contexto sejam satisfeitos.

Para ilustrar os conceitos acima e para discutir o formalismo MEBN será utilizado o exemplo *Star Trek* apresentado em [12]. O exemplo é baseado na série de televisão homônima que mostra as diversas missões da nave U.S.S. Enterprise. A nave é equipada com diversos sensores que oferecem ao comandante as diversas informações que ele precisa para manter-se livre de perigos.

Um nó pode ter uma lista de argumentos entre parênteses. Esses argumentos são substituídos por identificadores únicos das entidades do domínio de conhecimento quando uma rede é instanciada. Por convenção, esses identificadores começam por ponto de exclamação e duas entidades distintas não podem ter o mesmo identificador único. Em MEBN, variáveis são iniciadas por letra minúscula e predicados e funções, por letra maiúscula. A MFrag apresentada na Figura 4.1 representa o nível de perigo a que uma determinada nave é exposta.

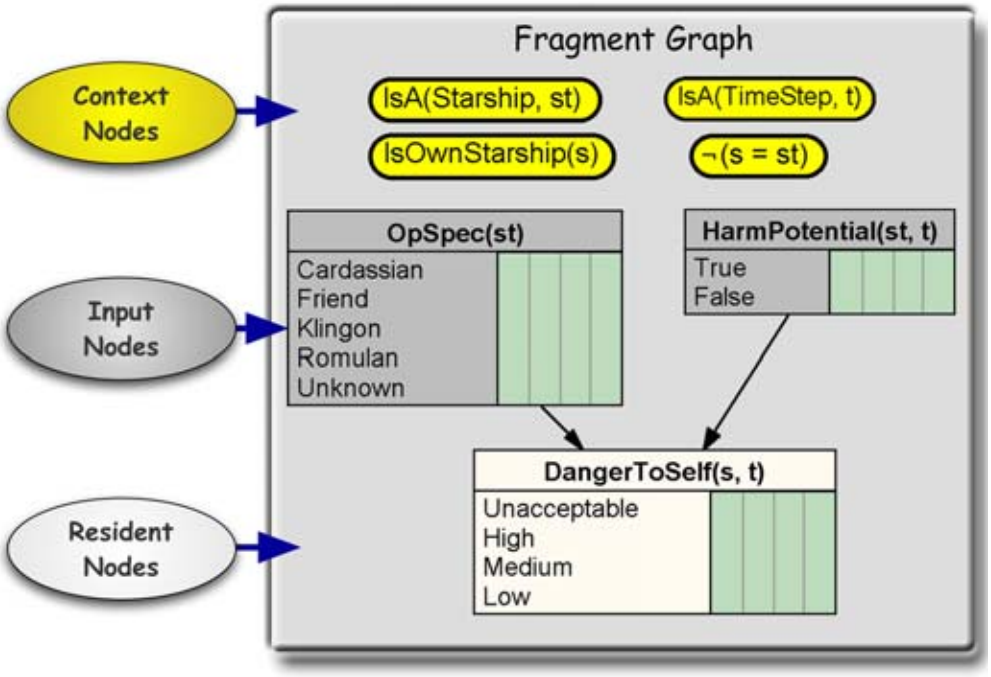

Figura 4.1: MFrag.

Essa MFrag possui sete nós: quatro de contexto; dois de entrada; e um residente.

Os nós de contexto (do inglês, *context nodes*) são variáveis booleanas que

representam condições que devem ser satisfeitas para que a distribuição de probabilidade dos nós residentes se aplique. Seus valores possíveis são:

- `True`, quando a condição é satisfeita;
- `False`, quando a condição não é satisfeita;
- `Absurd`, quando uma expressão não faz sentido.

Por exemplo, para o nó `IsOwnStarship(`*`s`*`)`, se *`s`* for substituído pelo identificador da Enterprise (por exemplo, *`!ST0`* ou *`!Enterprise`*), o valor do nó será `true`. Se o argumento for substituído pelo identificador de uma outra nave qualquer (por exemplo, *`!ST1`*), então o valor do nó será `false`. Mas se o argumento for substituído pelo identificador de qualquer outra coisa que não seja uma nave, o valor do nó será `absurd`, pois não faz sentido perguntar se o período de tempo é a sua própria nave, por exemplo.

Os nós de entrada (do inglês, *input nodes*) são variáveis que influenciam a distribuição de probabilidades dos nós residentes, mas as suas distribuições estão definidas nas suas próprias MFrags. Ou seja, em uma MTeoria completa, se em uma MFrag existe um nó de entrada, então é preciso que exista uma outra MFrag em que esse nó seja residente, onde sua distribuição de probabilidades é definida.

Os nós residentes (do inglês, *resident nodes*) possuem as distribuições locais de probabilidades. Nelas são definidas como as probabilidades dependem dos valores de seus pais (que podem ser tanto nós de entrada quanto outros nós residentes). Em uma MTeoria completa, cada variável aleatória possui apenas uma *home MFrag*, que é a MFrag onde sua distribuição local é definida. Portanto, a lógica MEBN padrão não suporta polimorfismo.

Na Figura 4.1, o nó `DangerToSelf(`*`s`*`,`*`t`*`)` possui dois argumentos: *`s`* e *`t`*. Nesse domínio de conhecimento (exemplo *Star Trek*), *`s`* deve ser uma espaçonave e *`t`*, um período de tempo. A instância `DangerToSelf(`*`!Enterprise`*`,`*`!T0`*`)` avalia o nível de perigo exposto à Enterprise no instante de tempo *`T0`*.

Note que os valores dos estados dos nós não foram mostrados no grafo da MFrag. Isso porque uma MFrag é apenas um *template*, ou seja, ela não representa variáveis aleatórias individuais, mas sim uma classe de variáveis aleatórias. Os valores dos estados só aparecem em uma instanciação da MFrag.

Por exemplo, para se encontrar a distribuição de probabilidades de uma instância da variável `DangerToSelf(`*`s`*`,`*`t`*`)`, primeiro é necessário encontrar todas as instâncias de `HarmPotential(`*`st`*`,`*`t`*`)` e de `OpSpec(`*`st`*`)` para as quais os nós de contexto são satisfeitos.

Um pseudocódigo para a distribuição da variável `DangerToSelf(`*`s`*`,`*`t`*`)` é mostrado na Figura 4.2.

Esse pseudocódigo define a distribuição local para o perigo a que uma nave é submetida devido a todas as naves que influenciam o seu nível de perigo. Essa distribuição leva em conta o número de naves, que não é previamente conhecido. Essa situação não seria possível em redes bayesianas.

As linhas 3 a 5 representam o caso de haver pelo menos uma nave operada pela espécie `Cardassian` e que pode representar uma situação de perigo para a Enterprise. A probabilidade de um perigo inaceitável é 90% mais o mínimo entre

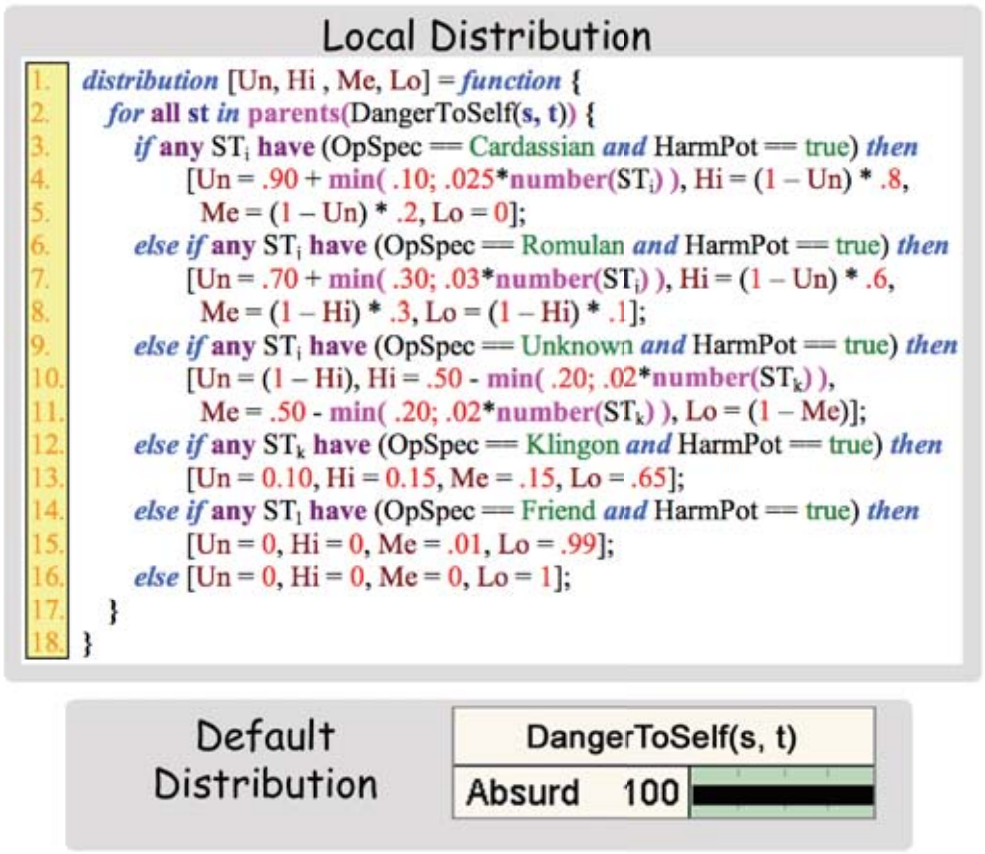

Figura 4.2: Pseudocódigo.

10% e 2,5% vezes o número de naves operadas pelos `Cardassians`. O resto da probabilidade é dividido entre um nível de perigo alto (80% do que sobrou) e um nível médio (20% do que sobrou). A probabilidade de perigo baixo é zero. A *default distribution*, ou seja, a distribuição padrão é necessária para o caso de não existir nenhuma instância de `HarmPotential(st,t)` e de `OpSpec(st)` para os quais os nós de contexto são satisfeitos. Assim, é necessária essa distribuição padrão, com probabilidade 1 dado a `absurd`, ou seja, essa situação não faz sentido.

A Figura 4.3 apresenta um exemplo de uma instância da MFrag da Figura 4.1, no qual há quatro naves que representam perigo.

Observe que foram instanciadas as variáveis `HarmPotential(st,t)` e `OpSpec(st)` para cada nave. As distribuições de probabilidades dessas oito variáveis são calculadas em suas próprias MFrags, que não são mostradas aqui para simplificar o exemplo. Como citado anteriormente, as condições de contexto não são mostradas para evitar o excesso de variáveis no grafo.

4.3 Recursividade em MEBN

A falta de suporte a recursividade das redes bayesianas pode ser importante em casos reais. MEBN fornece suporte a definições de recursão bem gerais. A Figura 4.4 apresenta dois exemplos de recursão.

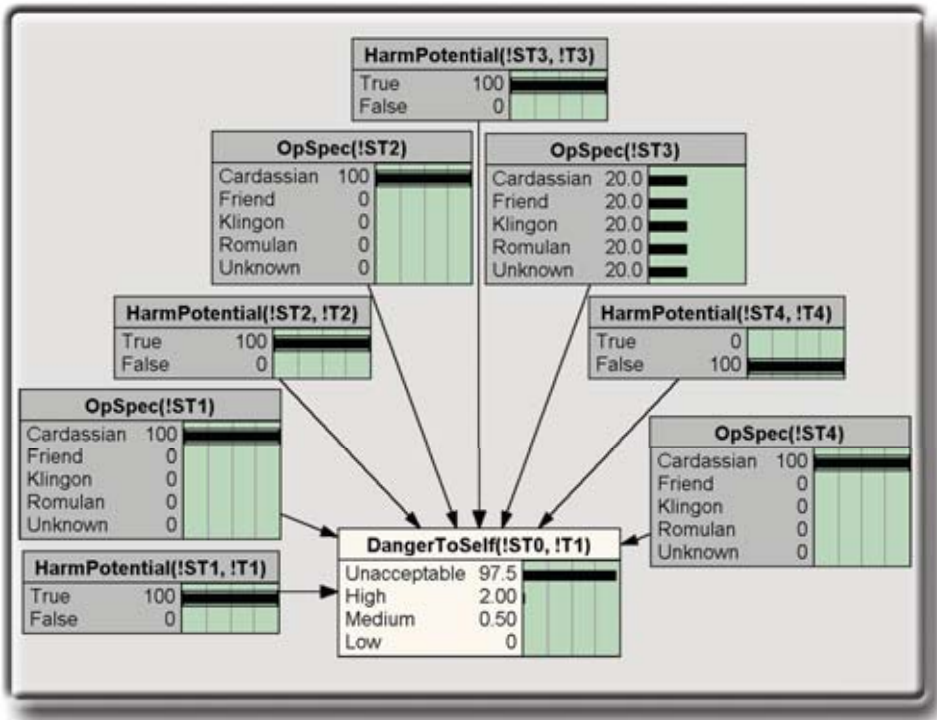

Figura 4.3: Instanciação de MFrag.

Observe que, para que as distribuições dos nós residentes se apliquem, é necessário que *z* seja uma zona e que *st* seja uma nave. Além disso, *st* precisa estar localizada na zona *z*. Esse é um tipo de recursão, já que uma variável depende de outra que, por sua vez, também depende de outra variável.

Outro tipo de recursão apresentado na Figura 4.4 é a temporal: 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*.

Outros tipos de recursão também são representados em MEBN por MFrags que permitem relações entre instâncias da mesma variável aleatória. É necessário que, ao se permitir definições de recursão, garanta-se que uma variável aleatória não pode influenciar a sua própria distribuição de probabilidade, pois senão teríase um caso de laço infinito. As condições de recursão de MFrags são apresentadas em [27].

Os nós de entrada de uma MFrag podem incluir instâncias de nós definidos recursivamente na própria MFrag. Por exemplo, o nó de entrada `ZoneMD(`*z*`,`*tprev*`)` representa o distúrbio magnético na zona *z* no período de tempo anterior ao *t*, e esse nó influencia o distúrbio magnético atual, dado pela variável `ZoneMD(`*z*`,`*t*`)`. A recursão precisa de uma distribuição inicial no período inicial (*T0*, por exemplo), que não dependa de um distúrbio anterior, para quebrar a cadeia de dependências.

A Figura 4.5 ilustra como as definições de recursividade podem ser aplicadas para construir uma SSBN (*Situation-Specific Bayesian Network* ou rede bayesiana de situação específica). Obviamente, uma SSBN é uma rede bayesiana. Nesse caso, deseja-se saber o distúrbio magnético no período de tempo *T3* na zona *Z0*.

O processo de construção da SSBN da Figura 4.5 começa criando a instân-

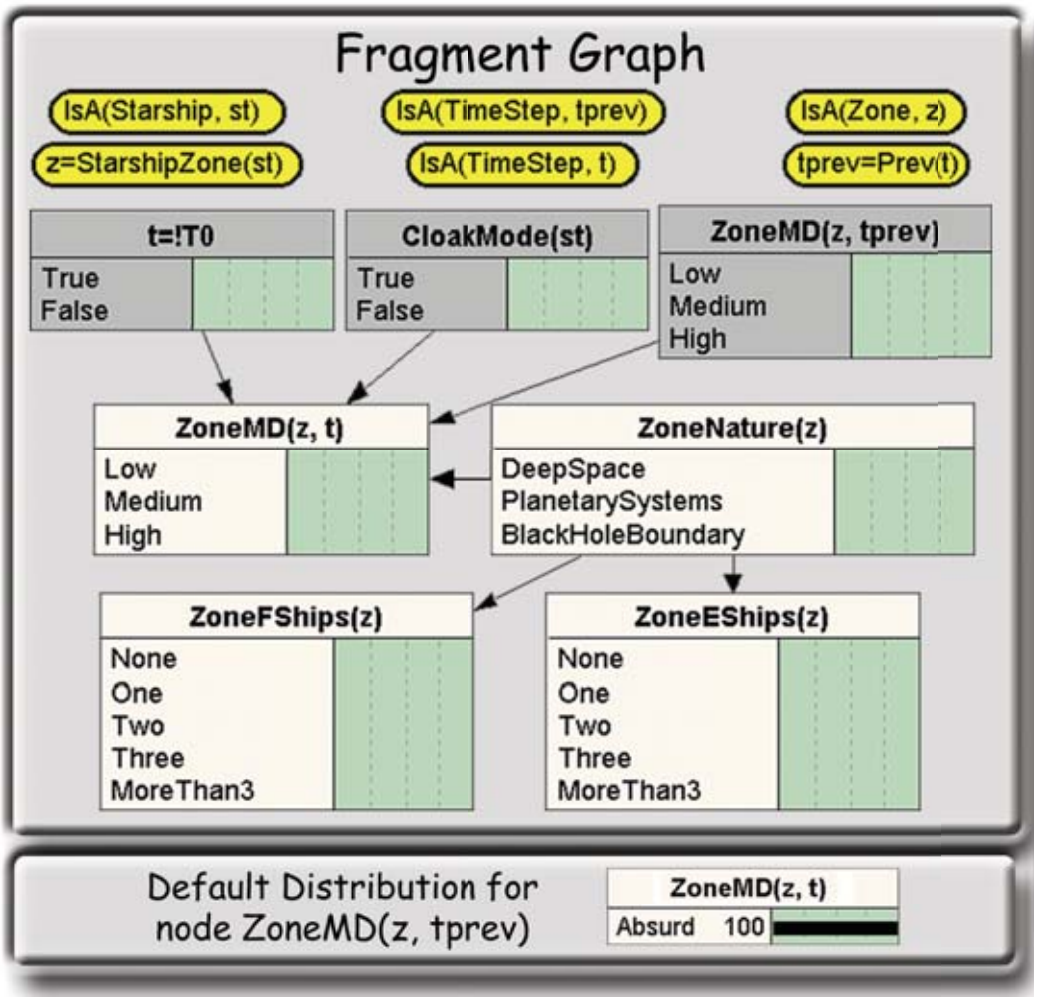

Figura 4.4: Exemplo de recursividade no modelo do *Star Trek*.

cia da MFrag no nó `ZoneMD(!Z0,!T3)`, nó cuja distribuição deseja-se conhecer. Instanciam-se, então, os outros nós residentes presentes na MFrag para os quais os nós de contexto são satisfeitos. A seguir, são construídas as CPT que podem ser construídas com os dados já disponíveis. As CPTs dos nós `ZoneNature(!Z0)`, `ZoneEShips(!Z0)` e `ZoneFShips(!Z0)` podem ser construídas.

No final desse processo, o nó `ZoneMD(!Z0,!T2)` permanece sem uma CPT. Assim, sua *home MFrag* (no caso, como é uma recursão, é a mesma MFrag, mas com outros parâmetros) é recuperada e as variáveis aleatórias que ainda não foram instanciadas precisam ser instanciadas. O processo continua até que o último nó, `ZoneMD(!Z0,!T0)` seja instanciado. Nesse ponto, a SSBN está completa, e já se conhece a distribuição de probabilidades do nó de interesse, `ZoneMD(!Z0,!T3)`.

É importante notar que seria fácil especificar um conjunto de MFrags com influências cíclicas, ou um conjunto tendo distribuições conflitantes para uma variável aleatória em diferentes MFrags. É necessário tomar cuidado para definir um modelo coerente como uma coleção de MFrags. No entanto, a interface gráfica criada no UnBBayes impede que o usuário cometa esse tipo de erro [6].

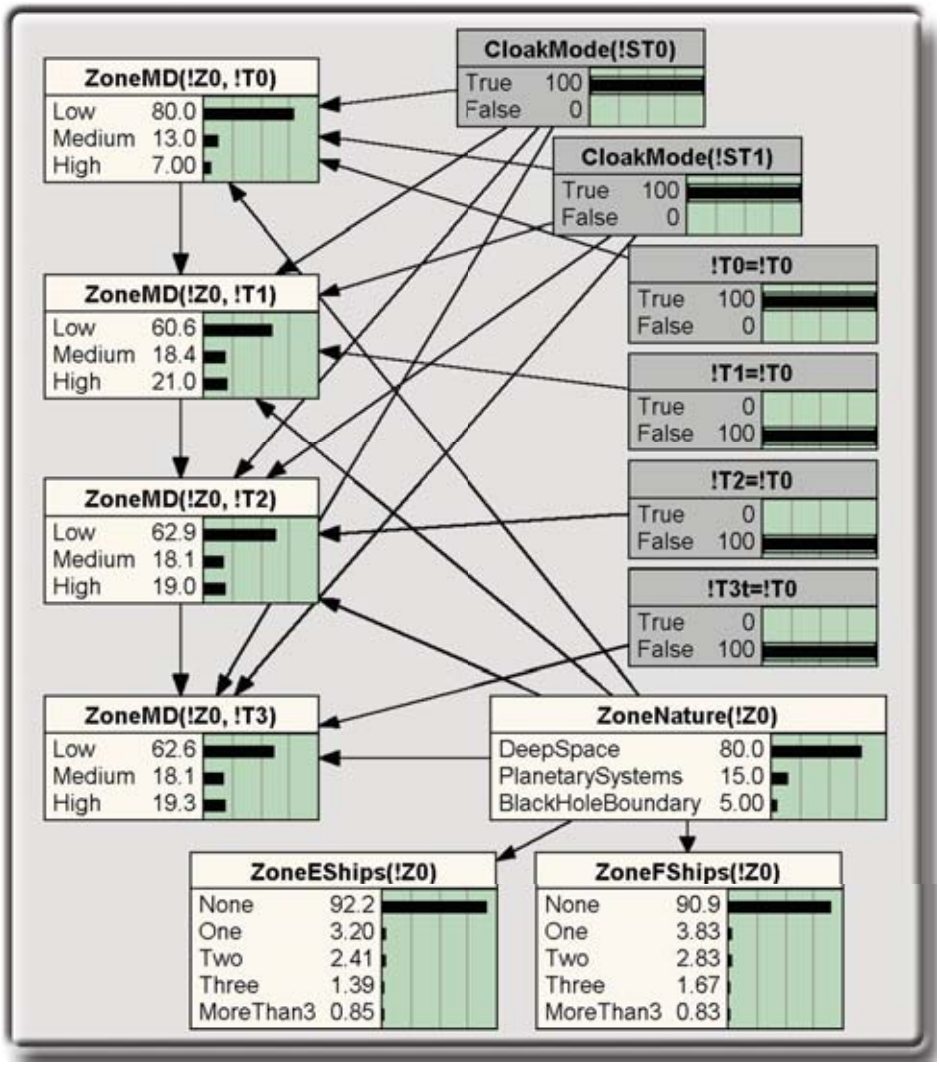

Figura 4.5: Exemplo de SSBN.

4.4 MTeoria

Para construir um modelo coerente é necessário que o conjunto de MFrags satisfaça coletivamente as condições de consistência, garantindo a existência de uma distribuição de probabilidades conjunta única sobre as instâncias das variáveis aleatórias das MFrags. Esse conjunto coerente é chamado de uma MTeoria.

Uma MTeoria 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 local e padrão de cada MFrag. Essas MFrags são parte de uma MTeoria geradora (do inglês, *generative MTheory*), que resume as regularidades estatísticas que caracterizam o domínio.

Para que uma MTeoria geradora possa ser utilizada em inferências sobre situações particulares, é necessário dar ao sistema informação específica das instâncias envolvidas na situação.

A inferência bayesiana pode ser usada tanto para responder as perguntas de in-

teresse (chamadas de *query*, ou questionamento), quanto para refinar uma MTeoria (através de *findings*, ou evidências).

A entrada de evidências é o mecanismo básico para incorporar novas informações em MTeorias. Elas são representadas 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, então, as distribuições de todas as variáveis envolvidas na situação específica. No entanto, na implementação presente no UnBBayes, é necessário que um novo questionamento seja realizado para que essas novas evidências sejam efetivamente incorporadas e processadas na geração da SSBN.

Cada variável aleatória deve ter uma única *home MFrag*. Além disso, uma MTeoria válida deve garantir que todas as definições de recursão terminam em passos finitos e não contenham influências circulares. Existem ainda outras condições necessárias para que uma MTeoria seja considerada válida. Essas condições foram apresentadas e provadas em [27].

A lógica MEBN contém um conjunto de *built-in MFrags*, ou seja, MFrags presentes em qualquer situação criada, que incluem quantificadores, referência indireta e MFrags booleanas que representam conectivos, como `and` e `or`. Estas agregam ao sistema a habilidade de representar qualquer sentença da lógica de primeira ordem.

A Figura 4.6 ilustra um exemplo de uma MTeoria geradora para o exemplo *Star Trek*.

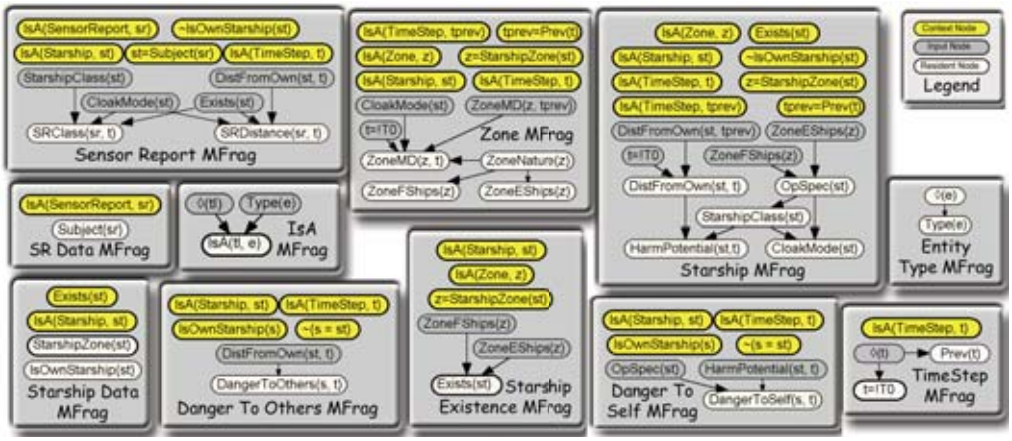

Figura 4.6: Exemplo de MTeoria no modelo do *Star Trek*.

A MFrag `Entity Type` é usada para declarar tipos de entidade que podem ser encontradas no modelo. Como MEBN não é um sistema tipado, é necessária a criação dessa MFrag auxiliar quando se deseja construir um sistema com suporte a tipos. MEBN pode ser estendida para acomodar qualquer tipo de sistema tipado, incluindo polimorfismo.

A lógica MEBN dá aos criadores de um determinado domínio um grande poder e flexibilidade porque permite diferentes modos equivalentes de se definir um mesmo domínio de conhecimento. Ou seja, a Figura 4.6 é apenas uma de diversas maneiras diferentes (embora todas consistentes) de se representar a distribuição conjunta da MTeoria.

Como exemplo dessa flexibilidade, a Figura 4.7 apresenta o mesmo domínio de conhecimento, em que a MFrag `Starship` foi decomposta em três MFrags distintas. Isso depende de quem modela o domínio. Um modelador pode escolher uma maior flexibilidade, com MFrags menores, mais específicas, enquanto outro pode querer um modelo mais conciso, em que todo o conhecimento de um determinado assunto é colocado em apenas uma MFrag.

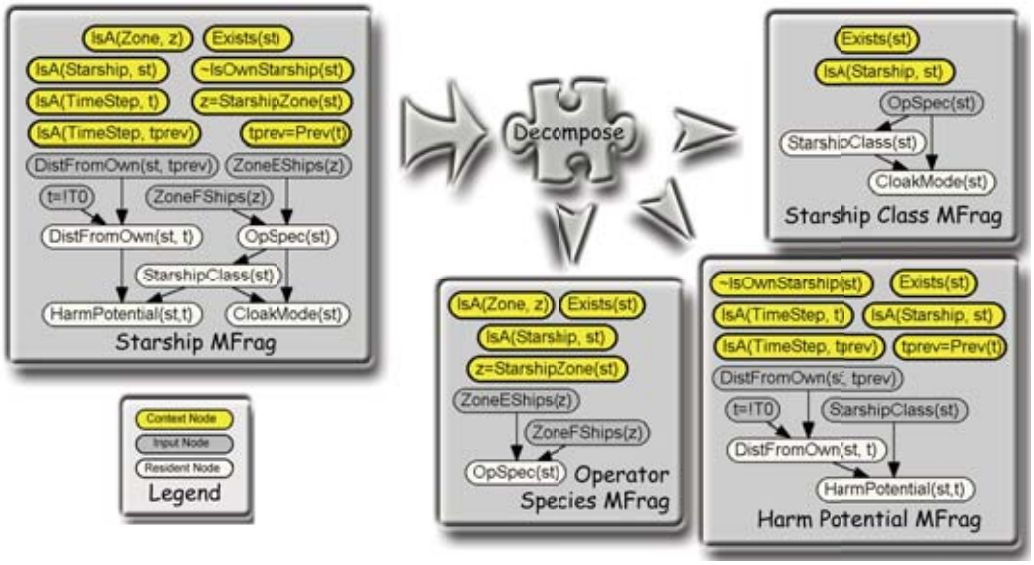

Figura 4.7: Flexibilidade da MEBN.

A FOL provê a base teórica para sistemas tipados usados em linguagens orientadas a objetos [12]. A lógica MEBN provê a base para estender a capacidade desses sistemas introduzindo uma base matemática para representar e raciocinar sobre incertezas.

4.5 Inferência em MEBN

O processo básico para se obter uma BN a partir de uma MTeoria e de evidências é a construção de uma SSBN.

Em redes bayesianas, uma *query* é o processo de entrar com evidências, propagar as crenças e observar os resultados nos nós de interesse.

Em MEBN, a situação é semelhante. Precisa-se de uma MTeoria válida, um conjunto de evidências (ou seja, as informações obtidas sobre a situação em questão) e um conjunto de objetivos (ou seja, os nós de interesse).

A SSBN começa a ser construída instanciando-se as *home MFrags* dos conjuntos de evidências e nós de interesse. Quando cada MFrag é instanciada, instâncias de todas as suas variáveis aleatórias são criadas. Se existir alguma variável com distribuição indefinida, então as suas *home MFrags* são instanciadas.

O processo de instanciação das MFrags continua até que não exista nenhuma variável com distribuição indefinida. O resultado, então, é uma SSBN. A Figura 4.8 ilustra um exemplo.

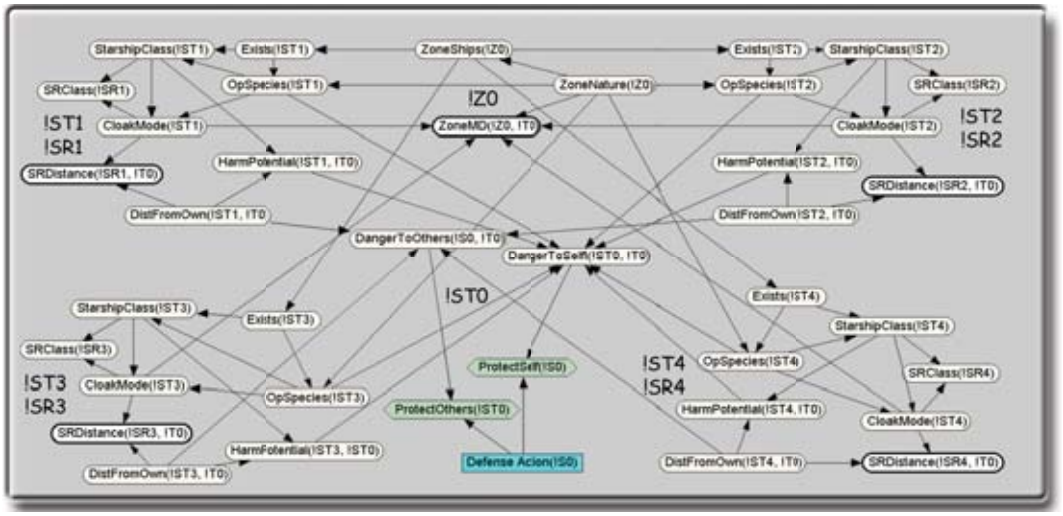

Figura 4.8: Exemplo complexo de SSBN.

Em alguns casos, a SSBN pode ser infinita. Entretanto, sob condições dadas por Laskey [27], o algoritmo produz uma seqüência aproximada de SSBN em que a distribuição dos nós de interesse converge para a distribuição correta ou aproximada. Uma SSBN pode conter qualquer número de instâncias de cada MFrag, dependendo do número de entidades no domínio e de suas relações.

Capítulo 5

Arquitetura do UnBBayes

Neste Capítulo, serão apresentadas uma visão da arquitetura do UnBBayes e maiores detalhes sobre a versão 2 do UnBBayes. A seção 5.1 apresenta a arquitetura do UnBBayes com seus principais módulos e as principais dependências com bibliotecas de terceiros. A seção 5.2 apresenta um histórico do UnBBayes. A seção 5.3 apresenta uma visão geral da ferramenta. Por fim, a seção 5.4 apresenta a modelagem do UnBBayes para as redes probabilísticas BN, ID e MSBN.

5.1 Arquitetura

A Figura 5.1 apresenta a arquitetura do UnBBayes. As configurações de dependência, do site, da versão entre outras é feita através do uso do Maven. A linguagem utilizada é o J2SE 5. O UnBBayes-MEBN, produto desta pesquisa, foi desenvolvido em cima da versão 2 do UnBBayes, disponível em www.sourceforge.net/projects/unbbayes. As caixas em verde representam as bibliotecas de terceiros utilizadas.

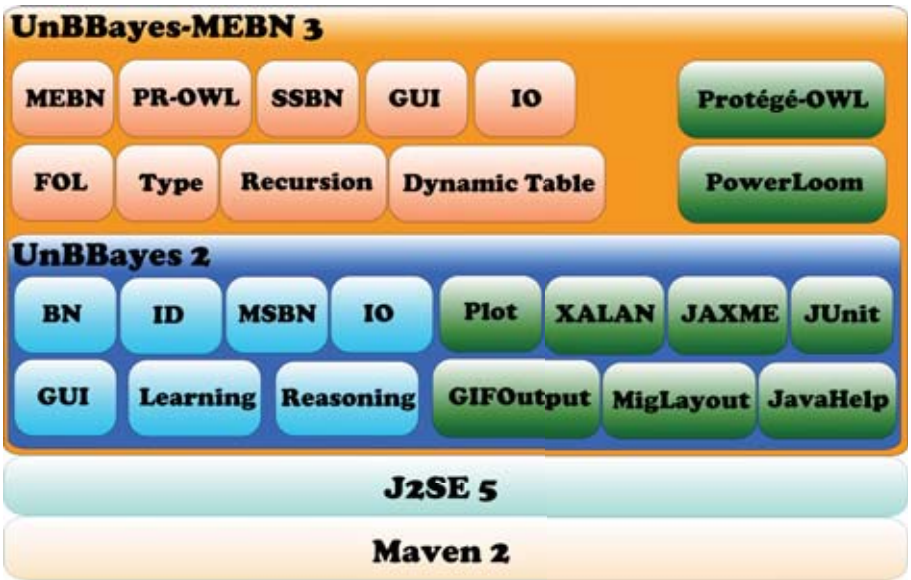

Figura 5.1: Arquitetura do UnBBayes.

A versão 2 do UnBBayes suporta redes bayesianas (BN), diagramas de influência (ID) e redes bayesianas múltiplas seccionadas (MSBN). Além disso, possui uma interface gráfica para modelar essas redes probabilísticas suportadas (GUI) e um mecanismo de entrada e saída para salvar e carregar os modelos (IO). Alguns mecanismos de aprendizagem também são suportados (Learning). Por fim, possui um mecanismo para fazer inferências nas redes probabilísticas suportadas. Nessa versão foram necessárias o uso de algumas bibliotecas de terceiros. O GIFOutput é utilizado para salvar as redes como um arquivo de imagem. O Plot é utilizado para desenhar gráficos. O MigLayout para fazer gerência de *layout* dos objetos de GUI. O JavaHelp para apresentar ajuda *online*. O JUnit para fazer os testes unitários. E o XALAN e JAXME para salvar e carregar BN e ID no formato XMLBIF.

A versão 3 do UnBBayes, também chamada de UnBBayes-MEBN, fruto desta pesquisa, é compatível com a versão 2. Dessa forma, faz uso de alguns módulos da versão anterior e extende outros. O UnBBayes-MEBN suporta redes bayesianas multi-entidades (MEBN) e possibilita a edição de ontologias probabilísticas (PR-OWL), ver seção 7.1. Para isso, estende a interface gráfica da versão 2 para possibilitar a edição de MEBN/PR-OWL (GUI), ver seção 7.3. Esta versão também implementa um algoritmo para geração de rede bayesiana de situação específica (SSBN), que é uma BN normal, ver seção 7.7. Dessa forma, faz uso do módulo de inferência (Reasoning) da versão 2, uma vez que a SSBN foi gerada. Possui um mecanismo para salvar e carregar arquivos PR-OWL (IO) compatíveis com o formato OWL, através do uso da biblioteca Protégé-OWL de terceiros, ver seção 7.8. Possibilita a representação e avaliação de nós de contexto em lógica de primeira ordem (FOL), através da biblioteca PowerLoom de terceiros, ver seção 7.4. Define e implementa mecanismo para tipagem (Type) com possibilidade de representar recursão (Recursion), ver seção 7.6. Por fim, possibilita uma definição da tabela de probabilidades condicionais de forma dinâmica através de pseudo-código (Dynamic Table), ver seção 7.2.

A API do Protégé-OWL e do PowerLoom podem ser substituídas por versões mais novas com uma simples configuração do arquivo pom.xml (arquivo de configuração do Maven) do projeto, atualizando apenas o campo versão de cada dependência. Isso é possível apenas se o fornecedor das bibliotecas garantirem que a versão mais nova é compatível com a utilizada pelo UnBBayes-MEBN. Caso se deseje utilizar outras API no lugar dessas, basta implementar uma classe que implemente a interface correspondente utilizada no UnBBayes, `KnowledgeBase` para a avaliação de FOL por exemplo, e atualizar a classe que faz uso dessa interface para utilizar a classe concreta correspondente. Além, é claro, de atualizar o arquivo pom.xml com a nova dependência da biblioteca de terceiros utilizada.

5.2 Histórico do UnBBayes

O UnBBayes é uma ferramenta em desenvolvimento pelo Grupo de Inteligência Artificial da Universidade de Brasília [26] para suportar o raciocínio probabilístico, desde 2000.

O UnBBayes possui um ambiente visual de fácil uso, permitindo que o usuá-

rio construa, de forma intuitiva, redes bayesianas, diagramas de influência e redes bayesianas múltiplas secionadas, além da entrada e propagação de evidências, realização de inferência probabilística e aprendizagem da topologia e/ou parâmetros de uma BN.

O UnBBayes foi desenvolvido utilizando-se a linguagem Java, sendo portanto independente de plataforma. O software é distribuído gratuitamente para uso não comercial, sob a licença GNU GPL, disponibilizando para os desenvolvedores uma API, juntamente com a documentação em JavaDoc, possibilitando que outros desenvolvedores que desejem utilizar raciocínio probabilístico em suas aplicações utilizem as classes disponibilizadas. Como ambiente de desenvolvimento foi utilizado o Eclipse (www.eclipse.org) com diversos *plugins* para facilitar a criação e testes das funcionalidades.

Atualmente o UnBBayes está sendo estendido para permitir a representação de ontologias probabilísticas e a realização de inferências em PR-OWL/MEBN.

5.3 Visão geral do UnBBayes

A modelagem do UnBBayes apresentada visa apenas oferecer uma noção da arquitetura geral do programa, não entrando em detalhes internos. O leitor interessado em estudar e utilizar a API encontrará a documentação JavaDoc e o código em http://unbbayes.sourceforge.net/.

Os pacotes que compõem o software são os seguintes:

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

pacote `controller` classes responsáveis por controlar o fluxo de operações realizadas pelo usuário, fazendo a ligação entre a GUI e as funcionalidades;

pacote `gui` contém as classes que implementam a interface com o usuário, permitindo a visualização e manuseio das redes bayesianas;

pacote `io` classes responsáveis por salvar as redes bayesianas 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 redes bayesianas dadas como parâmetro;

pacote `prs` classes que definem a estrutura das redes bayesianas e implementam os algoritmos de inferência sobre estas;

pacote `util` classes utilitárias.

5.4 Modelagem do UnBBayes

As modelagens apresentadas nesta seção visam apenas dar uma visão geral das classes do UnBBayes que serão reaproveitadas na implementação da MEBN. Para

tal descrição, serão apresentados apenas diagramas de classes na seguinte sequência: BN e ID na seção 5.4.1; e MSBN na seção 5.4.2.

5.4.1 BN e ID

A modelagem apresentada na Figura 5.2 apresenta as principais classes que implementam as redes bayesianas no UnBBayes.

Classe `Node` um nó genérico que possui estados (`String`), uma posição (x,y) e um tamanho (largura,altura). Estes dois últimos campos representados pela classe `SerializablePoint2D`.

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.

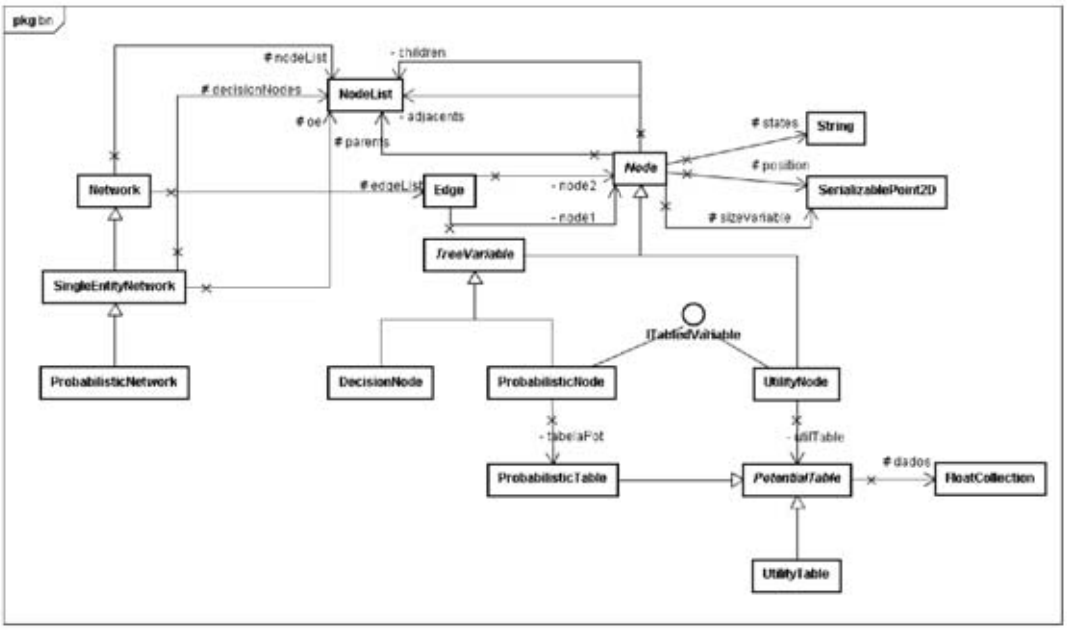

Figura 5.2: Modelagem de classes para BN e ID.

Os tipos de variáveis atualmente suportadas pelo UnBBayes são as variáveis probabilísticas (isto é, variáveis aleatórias), as variáveis de decisão e as funções utilidade, sendo estas duas últimas utilizadas para a modelagem de gráficos 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 de potencial que possui os valores das células como um número real entre 0 e 1 (propriedade `dados` que é uma coleção de números reais representados pela classe `FloatCollection`).

Classe `ProbabilisticTable` tabela de potencial de probabilidades.

Classe `UtilityTable` tabela de potencial de utilidade.

Interface `ITabledVariable` interface para variáveis com tabela para definição dos valores associados aos estados possíveis (as variáveis probabilísticas e as de utilidade).

Classe `NodeList` classe que representa um array dinâmico do tipo `Node`. É utilizado pelas classes: `Network` para armazenar seus nós; pela classe `SingleEntityNetwork` para armazenar a `oe` (ordem de eliminação dos nós) e os nós de decisão; pela classe `Node` para armazenar os nós pais, filhos e adjacentes; e pela classe `PotentialTable` para armazenar as variáveis que pertencem à tabela (não apresentados no modelo para facilitar o entendimento do modelo).

Classe `Network` representa um grafo genérico.

Classe `SingleEntityNetwork` representa uma rede que não possui multi-entidades.

Classe `ProbabilisticNetwork` representa uma rede probabilística.

5.4.2 MSBN

A Figura 5.3 apresenta a modelagem das classes que implementam MSBN. MSBN (rede bayesiana múltipla seccionada) é baseada na realização de inferência em uma rede bayesiana divida em sub-redes, preservando os subdomínios naturais existentes. O resultado da inferência é o mesmo que seria obtido com a rede bayesiana formado pela união das sub-redes.

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 as MSBN.

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

Classe `SingleAgentMSBN` uma rede múltipla seccionada.

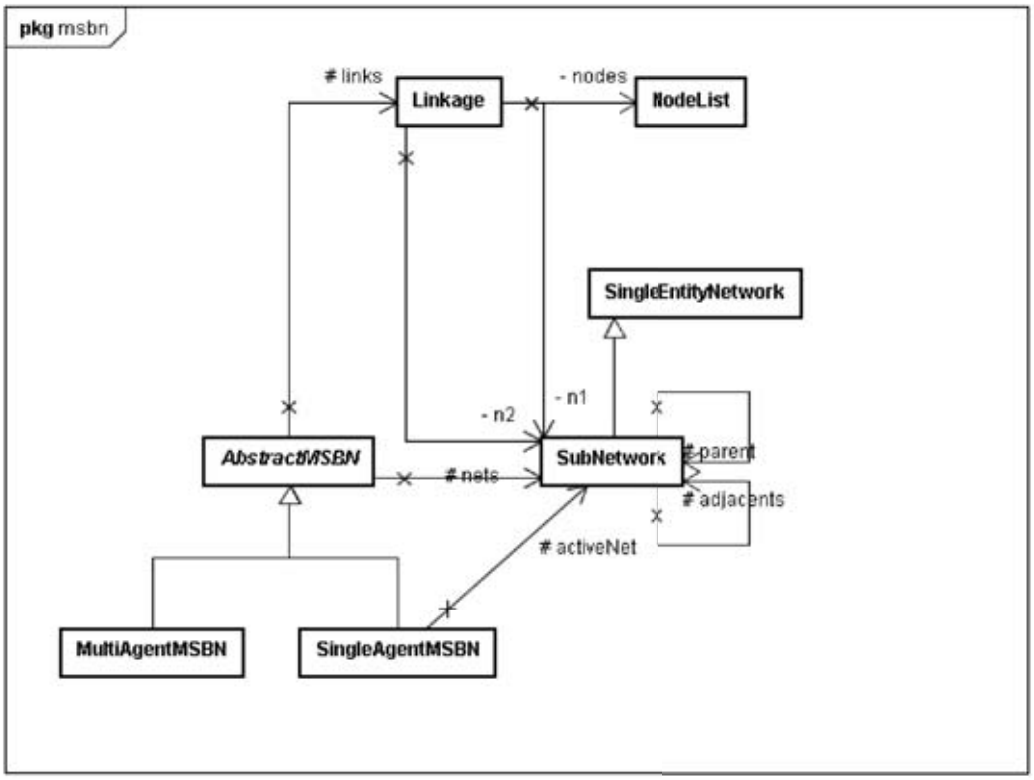

Figura 5.3: Modelagem de classes para MSBN.

Capítulo 6

Bibliotecas de Terceiros

Neste Capítulo, serão apresentadas as principais bibliotecas de terceiros utilizadas pelo UnBBayes na implementação de MEBN/PR-OWL. Primeiramente será apresentado o PowerLoom na seção 6.1. Em seguida, a seção 6.2 apresenta a API do Protégé.

6.1 PowerLoom

Os nós de contexto (`context nodes`) são variáveis booleanas que representam condições que devem ser satisfeitas para que a distribuição de probabilidade dos nós residentes se aplique. Avaliar o valor de um nó de contexto corresponde a avaliar uma fórmula da lógica de primeira ordem, tendo como premissas os fatos conhecidos. Na implementação do UnBBayes foi adotada a API PowerLoom para avaliação de fórmulas da FOL. Na seção 6.1.1 são apresentadas as principais características do PowerLoom. Na seção 6.1.2 é descrita a implementação da lógica de primeira ordem pelo PowerLoom, descrevendo os comandos utilizados para o seu uso. A seção não detalha a sintaxe dos comandos, devendo o leitor interessado, nesse tema, consultar [9].

6.1.1 Introdução ao PowerLoom

O PowerLoom é um sistema de representação de conhecimento e inferência (KR&R) baseado na lógica de primeira ordem, provendo 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).

Um sistema de KR&R permite modelar aspectos salientes de um mundo de interesse e fazer inferências a partir do modelo. Há diversos paradigmas de modelos para a implementação destes sistemas, sendo os baseados em lógica os mais utilizados por apresentarem vantagens tais como: a grande quantidade de pesquisas realizada, sintaxes e semânticas bem conhecidas e ao grande poder de representação.

O PowerLoom utiliza como linguagem de representação uma variante, com

expressividade total, do KIF (*Knowledge Interchange Format*). O KIF é uma linguagem baseada em lógica, desenvolvida para a troca de conhecimento entre sistemas de computadores diferentes.

O PowerLoom foi escrito em linguagem STELLA (*Strongly Typed, Lisp-like LAnguage*), que foi desenvolvida especificamente para este objetivo. Baseada em Lisp, a linguagem STELLA preserva as características desta que facilitam a programação simbólica e a rápida prototipagem, enquanto ainda permite a tradução em código Java, Lisp e C++ eficiente e legível. Por ser escrito em STELLA, o PowerLoom é distribuído nas três linguagens citadas. A API do PowerLoom para a linguagem Java é relativamente pequena, possuindo menos de 3 MB (juntamente com a STELLA).

O PowerLoom realiza inferência através da dedução natural, utilizando encadeamento *forward* e *backward* para derivar o que logicamente segue dos fatos e regras existentes na base de conhecimento. Além do mecanismo de dedução natural, o PowerLoom possui um grande número de procedimentos para *reasoners* especializados que eficientemente trabalham sobre hierarquias de conceitos e relacionamentos, conjuntos, etc. A arquitetura especialista é extensível para permitir que usuários acrescentem seus *reasoners* ou predicados computados. Há também suporte para raciocínio hipotético, raciocínio sobre igualdades, aritmética e raciocínio sobre desigualdades. Apesar de não ser uma lógica descritiva, possui classificadores que podem classificar hierarquias de conceitos e relacionamentos e instâncias definidas, utilizando o poder de expressividade da lógica de primeira ordem.

Como sistema de representação de conhecimento, o PowerLoom possui uma grande preocupação em oferecer o máximo de expressividade e de escalabilidade. A expressividade é vista com mais importância que a completude da inferência. O PowerLoom é bastante escalável, possuindo mecanismos para controlar a busca e oferecer suporte a grandes ontologias e bases de conhecimento.

Nessa pesquisa, o interesse no PowerLoom é no seu mecanismo de inferência sobre sentenças de lógica de primeira ordem, ficando em segundo plano as suas características de sistema de representação de conhecimento.

6.1.2 Lógica de Primeira Ordem no PowerLoom

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 estruturas de conhecimentos são organizadas em containers lógicos chamados módulos: os fatos não são válidos globalmente, mas apenas dentro de um contexto especifico. O PowerLoom permite a montagem de uma hierarquia de módulos, facilitando a modelagem de fatos que são válidos em diversos contextos e permitindo a importação de conhecimentos de outros módulos. Com essa característica, uma forma conveniente de se organizar o conhecimento é colocar as definições em módulos mais altos e os fatos em módulos mais baixos. A herança de fatos não é monotônica: um módulo filho pode retirar ou sobrescrever fatos herdados de módulos antecessores. O PowerLoom mantém um ponteiro apontando

para o módulo corrente, de forma que todas as assertivas, questionamentos, etc, são feitos relativamente a este módulo. O PowerLoom possui alguns módulos *built-in*, como por exemplo o PL-KERNEL que contém um conjunto de definições de conceitos e relacionamentos de propósito geral que coletivamente formam o suporte lógico para construir aplicações especificas baseadas em conhecimento.

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".

O PowerLoom requer que relacionamentos sejam definidos antes de serem utilizados em assertivas e questionamentos. Os comandos `defconcept`, `defrelation` e `deffunction` são utilizados para definir conceitos, relacionamentos e funções respectivamente. O uso de referência circular é permitido em definições, porém não em assertivas. Todos os axiomas que aparecem dentro dos limites de uma definição são resolvidos antes da próxima ocorrência de um questionamento.

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 o comando `assert` insere sentenças ou fatos na base de conhecimento.

- `(assert (not(Trabalha-para Jeronimo Megasoft)))`
- `(assert (Trabalha-para Maria Megasoft))`
- `(ask (Trabalha-para Jeronimo Megasoft))`
- `(retrieve all(Trabalha-para ?x Megasoft))`

Uma preposição PowerLoom é etiquetado com um valor verdade que pode ter um de cinco valores diferentes: `true`, `false`, `default-true`, `default-false` ou `unknow`. Os valores `true` e `false` podem ser definidos através do comando `assert`, enquanto os `defaults` são definidos através do comando `pressume`. Uma preposição previamente definida como `default` pode ser redefinida para qualquer um dos quatro outros valores. Preposições previamente definidas como `default-true` ou `default-false` podem ser redefinidas como `true` ou `false`, porém o inverso não é possível (apenas é possível subir o grau da força da definição do valor verdade, nunca descer para um grau anterior).

Um `clash` (contradição) ocorre quando uma preposição é redefinida de `true` para `false` ou vice-versa. Uma `clash-exception` é levantada, permitindo que as aplicações tomem a ação desejada (o comportamento padrão do PowerLoom é não dar tratamento nenhum, fazendo com que a assertiva que causou a exceção não seja executada). O comando `retract` permite retirar o atual valor-verdade de uma proposição, permitindo que seja definido qualquer outro valor verdade para esta.

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

6.2 Protégé

O formato adotado para representar PR-OWL é o apresentado por [12]. Esse formato é uma *upper-ontology* do formato OWL recomendado pela W3C, dessa forma, para que essa compatibilidade com o formato OWL fosse mantida, a API do Protégé foi utilizada para salvar e carregar as ontologias probabilísticas modeladas pelo UnBBayes. Ao carregar um ontologia probabilística, o UnBBayes lê o arquivo através da API do Protégé recuperando os elementos relacionados com a parte probabilística do domínio e constrói a estrutura de dados de MEBN utilizada pelo UnBBayes. Ao salvar, o processo inverso é feito, ou seja, a estrutura de dados de MEBN presente em memória no UnBBayes é adicionada aos elementos não probabilísticos presentes no arquivo carregado (se houve um arquivo carregado) e a API do Protégé é utilizada para salvar o modelo no formato definido. Dessa forma, esta seção descreve a API do Protégé utilizada nesse processo.

O Protégé é uma plataforma flexível e configurável para o desenvolvimento de aplicações dirigidas a modelos arbitrários e componentes. O Protégé permite a edição visual de ontologias por dois caminhos: via o editor Protégé-Frames e via o editor Protégé-Owl. As ontologias podem ser salvas em uma grande variedade de formatos, incluindo RDF, OWL e XML Schema. É uma ferramenta bastante flexível, permitindo a personalização da interface através da aba "forms" e configuração detalhada e diversificada.

O Protégé-OWL provê muitas facilidades de edição e navegação para modelos OWL. Provê diferentes classes de editores para uma ontologia em OWL. As mais

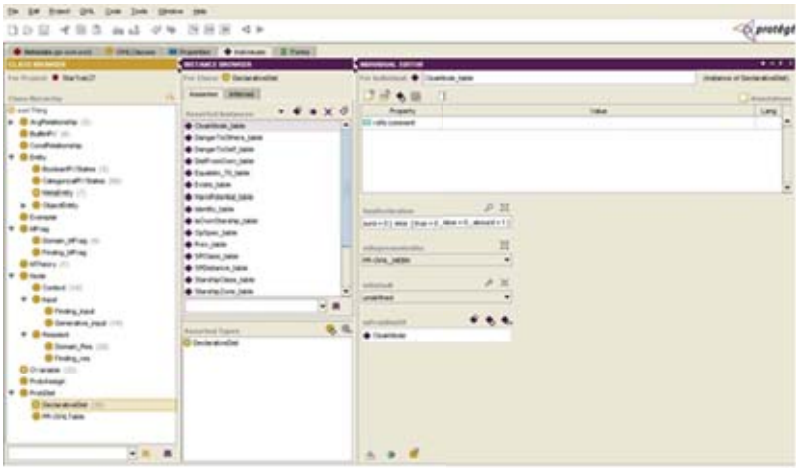

Figura 6.1: Tela de edição de classes/indivíduos no Protégé.

importantes são:

Editor de classes permite a navegação, criação, remoção e edição sobre classes OWL, incluindo edição avançada de restrições e atribuição de propriedades padrão do OWL e RDF.

Editor de indivíduos permite navegação sobre diferentes instâncias das classes presentes na ontologia e edição de seus atributos. A Figura 6.1 mostra uma captura de tela do modo de edição de indivíduos.

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, atribuição de propriedade inversa, comentários, etc. A Figura 6.2 mostra uma captura da tela de edição de propriedades.

Editor de formulários permite alterar o visual dos outros editores através da alteração do esquema de formulários. Todos os editores são formados por "blocos" de formulários com campos de entrada. Este editor permite modificar a organização desses formulários e a posição dos blocos.

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

O Protégé provê também uma série de API e possibilidade de extensão de sua funcionalidade através de desenvolvimento de *plugins*. Para mais informações, veja [19].

A API Protégé-OWL, parte do pacote Protégé-OWL, é uma biblioteca Java open source para OWL e RDF. Provê métodos para:

- abrir e salvar arquivos OWL;
- realizar *queries*;
- manipular modelos de dados OWL;

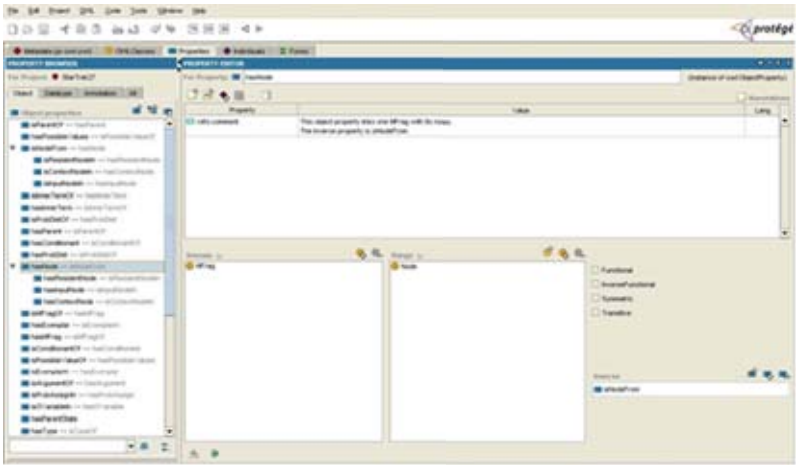

Figura 6.2: Tela de edição de propriedades no Protégé.

- realizar raciocínios.

A API Protégé OWL é centrada em uma coleção de interfaces Java para o pacote `model`. Essas interfaces fornecem acesso ao modelo OWL e aos seus diversos elementos como classes, propriedades e indivíduos. Desenvolvedores de aplicação não podem acessar a implementação destas interfaces diretamente, podendo apenas operar sobre elas, não sendo portanto necessário conhecer os detalhes internos sobre como o Protégé armazena suas ontologias.

O modelo de interface mais importante é o `OWLModel` que propicia acesso ao container de alto nível de recursos na ontologia. Assim pode-se utilizar o `OWLModel` para criar, fazer *queries*, e apagar recursos de vários tipos e utilizar os objetos retornados pelo `OWLModel` para operações especificas.

A API Protégé OWL pode ser utilizada em dois modos:

- modo de arquivos OWL (classe `JenaOWLModel`);
- modo de arquivos Database (classe `OWLDatabaseModel`).

Ambos os modos operam na mesma interface `OWLModel`. O programador não precisa se preocupar se a classe esta fisicamente criada em um banco de dados ou somente armazenada em memória. Há algumas diferenças entre os dois modos que trazem algumas funcionalidades extras.

As classes utilizadas pelo módulo de armazenamento de MEBN do UnBBayes estão listadas abaixo:

- `com.hp.hpl.jena.util.FileUtils`: utilizada para especificar o formato de armazenamento do arquivo OWL, através da constante `FileUtils.langXMLAbbrev`.
- `edu.stanford.smi.protegex.owl.ProtegeOWL`: é a classe núcleo da API. No UnBBayes, é utilizada como uma *builder* para a instanciação do `JenaOWLModel`.

- `edu.stanford.smi.protegex.owl.jena.JenaOWLModel`: `OWLModel` em modo de manipulação de arquivos OWL. Através desta classe, pode-se criar, alterar, remover e buscar classes, indivíduos e propriedades descritos na ontologia.

- `edu.stanford.smi.protegex.owl.model.OWLDatatypeProperty`: classe Java que representa uma propriedade em OWL que liga uma classe OWL (ou indivíduo) a um tipo simples de dado (como texto ou numerais). Foi utilizado, no UnBBayes, para ligar classes OWL de CPT à sua declaração em texto ou para ligar argumentos aos numerais indicando sua ordem.

- `edu.stanford.smi.protegex.owl.model.OWLIndividual`: classe Java que representa um indivíduo ou instância de uma classe OWL.

- `edu.stanford.smi.protegex.owl.model.OWLNamedClass`: classe Java que representa uma classe em OWL. Como são identificadas pelo nome, uma busca por nome pode ser feita pelo `JenaOWLModel`.

- `edu.stanford.smi.protegex.owl.model.OWLObjectProperty`: classe Java que representa uma propriedade OWL que liga uma classe a uma outra classe qualquer. Utilizada para declarar as propriedades da PR-OWL, que por sua vez são propriedades OWL.

- `edu.stanford.smi.protegex.owl.repository.impl.LocalFile Repository`: utilizada para criar um repositório local que armazena algumas informações pertinentes da *Dublin Core DL*. O arquivo gerado tem extensão ".repository". Esta classe ainda é uma *placeholder* para futuras adaptações.

Com a combinação das classes listadas acima, foi possível implementar o mecanismo de armazenamento do arquivo PR-OWL no UnBBayes. Para maiores informações sobre o uso da API Protégé-OWL, veja [24].

Jena é uma das APIs Java para RDF e OWL, fornecendo serviços para representação de modelos, *parsing*, persistência em banco de dados, *querying* e algumas ferramentas de visualização. O Protégé-OWL está intimamente acoplado ao Jena (Figura 6.3). Esta integração permite que os programadores usem certas funções do Jena em tempo real, sem precisar passar pelo processo de *build*.

A chave desta integração está no fato de ambos subsistemas operarem em representações de "triplas" em baixo-nível. O Protégé possui seu próprio mecanismo de armazenamento de *frames*, que foi encapsulado no conjunto de classes de "Grafos Protégé (*Wrapper* do Jena)"; permitindo que o acesso do Jena para a leitura passe sempre pelas "triplas" do Protégé. Esse esquema permite a edição com a API do Protégé e a *query* com a API do Jena.

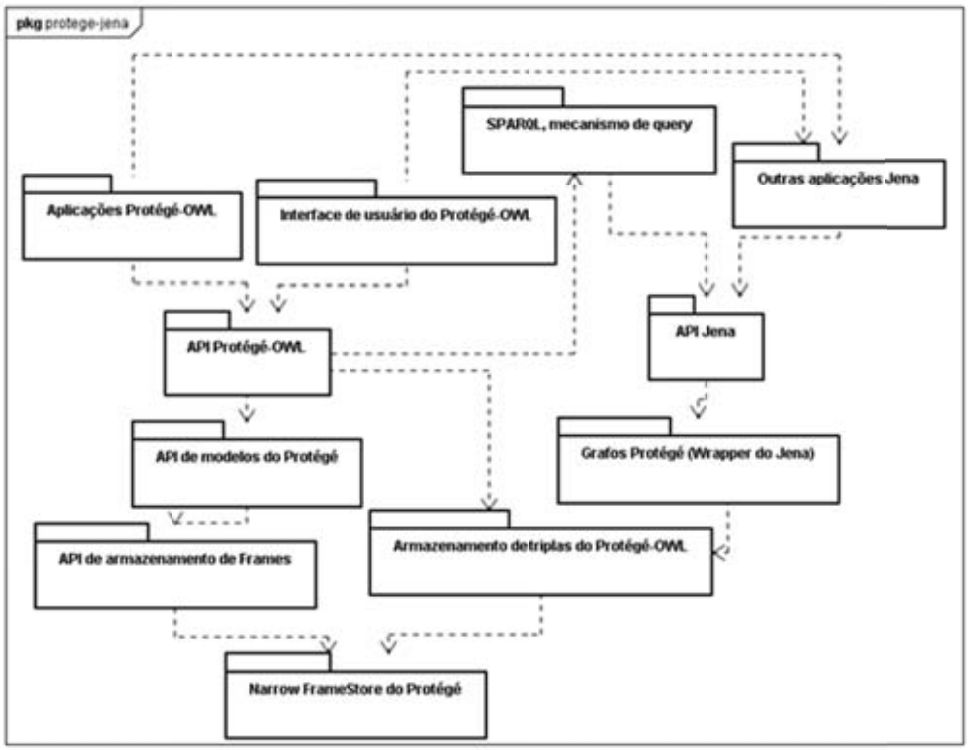

Figura 6.3: Esquema de acoplamento do Protégé-Owl com o Jena.

Capítulo 7

Modelagem e Implementação de MEBN

Neste Capítulo, tratar-se-á de questões de modelagem e de implementação relacionados ao UnBBayes já existente e principalmente da nova versão que possui funcionalidades para modelar uma MEBN e raciocinar em cima da mesma. Primeiramente será apresentada a modelagem e implementação de MEBN na seção 7.1. A seção 7.2 mostra como é definida uma tabela de um nó residente. Em seguida, a interface gráfica para MEBN será detalhada na seção 7.3. Para avaliar os nós de contexto, será explanado a API escolhida para FOL e como é utilizada na seção 7.4. Na seção 7.5 são destacadas as regras de consistência existentes em MEBN. Já na seção 7.6 será apresentada a implementação de tipo, tipo ordenável e recursão que foi criada como uma funcionalidade embutida no UnBBayes, ao invés de deixar a cargo do usuário. Na seção 7.7 é apresentado o algoritmo de geração de SSBN implementado no UnBBayes. Por fim, o mecanismo de I/O é detalhado na seção 7.8.

7.1 Modelagem da extensão para MEBN

Segue a descrição das principais classes construídas na extensão do UnBBayes. A modelagem será descrita utilizando-se diagramas de classes. Na seção 7.1.1 será apresentada as principais classes utilizadas para a representação de uma MTeoria. Na seção 7.1.2 serão apresentados os nós utilizados em MEBN e seus relacionamentos. Já na seção 7.1.3 será detalhada a estrutura de uma MFrag.

7.1.1 MTeoria

A modelagem da Figura 7.1 representa a estrutura geral de uma MTeoria.

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

Classe `MFrag` representa um fragmento de teoria MEBN. Uma MFrag é composta por variáveis ordinárias, nós de entrada e nós residentes, sendo que deve possuir ao menos um nó residente.

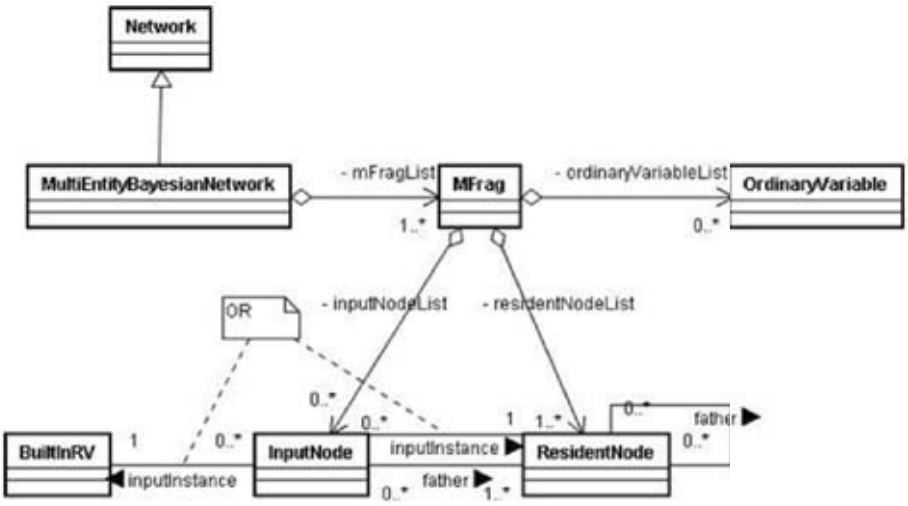

Figura 7.1: Modelagem de classes para MTeoria.

Classe `OrdinaryVariable` variáveis ordinárias são utilizadas para se referir a entidades não específicas como argumentos nas variáveis aleatórias (RV) das MFrags. Para cada argumento de uma RV na MFrag deve existir um objeto `OrdinaryVariable`, sendo que este define o tipo das entidades aceitas para preencher aquele parâmetro. Embora não esteja representado no modelo, ele estende `Node` para representar seu respectivo nó de contexto `isA(Type)`.

Classe `InputNode` classe que representa uma RV de entrada. É basicamente uma cópia de um ResidentNode que é usado como entrada em uma MFrag.

Classe `ResidentNode` classe que representa uma RV residente. Possui informações sobre seus argumentos (`OrdinaryVariable`).

Classe `BuiltInRV` classe que representa variáveis randômicas, previamente construídas pela implementação do UnBBayes (conectivos lógicos, quantificadores e igualdade). Permitem representar uma família de distribuições de probabilidade sobre interpretações da FOL. Estas variáveis são utilizadas para facilitar a modelagem, dando uma base onde o desenvolvedor pode começar a fazer a ontologia. Na Tabela 7.1 são apresentadas as `BuiltInRV` que estão implementadas no UnBBayes.

Tabela 7.1: `BuiltInRV` implementadas no UnBBayes.

`And`	`EqualTo`	`Exists`	`ForAll`
`Iff`	`Implies`	`Not`	`Or`

7.1.2 Nós de MEBN

A Figura 7.2 apresenta a hierarquia dos tipos de nós existentes na MEBN.

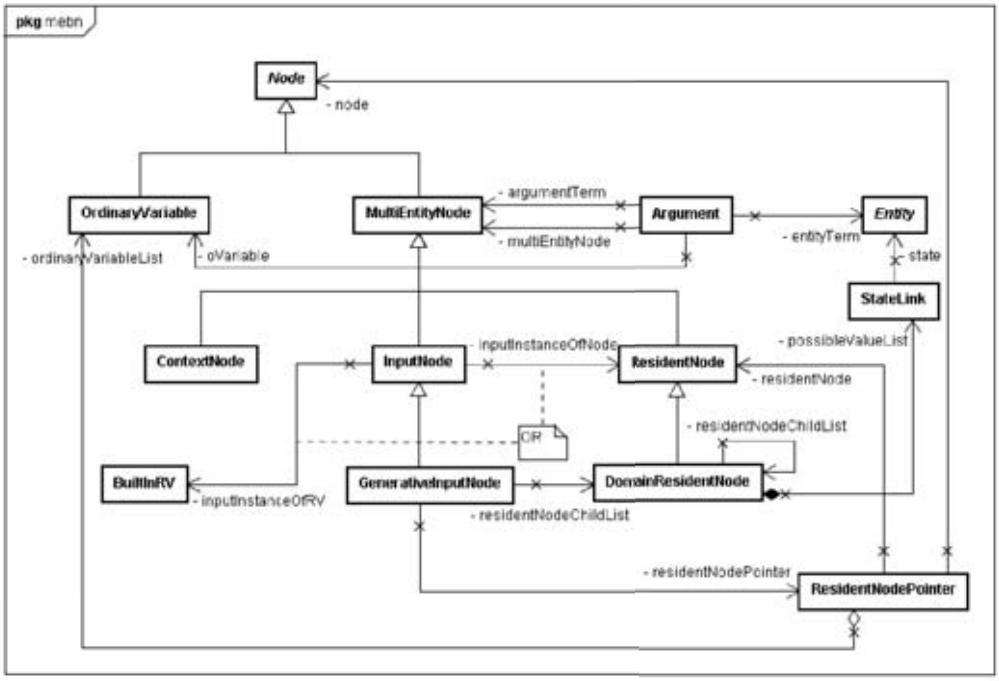

Figura 7.2: Modelagem de classes para os nós de uma MEBN.

Existe dois tipos de entidades `MultiEntityNode` e `SingleEntityNode` (seção 5.4.1), ambas herdando de `Node`. `OrdinaryVariable` herda de Node para permitir que este seja desenhado no grafo da MFrag como um nó de contexto `IsA(Type)`.

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

Classe `Entity` representa uma entidade. Explicada com mais detalhe na seção 7.6.

Classe `StateLink` representa uma vinculação entre o `DomainResidentNode` e `Entity`, possibilitando a definição de exclusividade global, explicado com mais detalhes mais abaixo nessa seção.

Classe `Argument` representa um argumento de um determinado `MultiEntityNode`. Esta classe é utilizada apenas no processo de salvar e carregar 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 da MFrag em questão.

Classe `GenerativeInputNode` representa os nós de entrada geradores. Estes nós são apenas uma referência para `DomainResidentNode` já existentes, que geralmente se encontra em outra MFrag, mas em algumas situações, são definidos na mesma MFrag, para representar recursão, por exemplo.

Classe `DomainResidentNode` representa os nós residentes geradores. Estes nós possuem um pseudocódigo, responsável por definir a regra de criação de uma CPT, independentemente do caso específico (entidades e evidências presentes). A seção 7.2 apresenta mais detalhes a respeito do pseudocódigo e geração de CPT. Os nós residentes podem ter como pais outros nós residentes (não pode ser pai dele mesmo) ou nós de entrada.

Uma contribuição para o PR-OWL, resultante dessa pesquisa, que foi aceita pela equipe responsável pela definição dessa linguagem, foi a inclusão de informação a respeito da exclusividade global de um determinado estado de um nó, na versão 1.05 (www.pr-owl.org). Essa propriedade é necessária em situações onde apenas uma evidência é permitada para um nó em um determinado estado. Por exemplo, na MFrag `Starship Data` apresentada na Figura 7.3, o estado `True` para a variável `IsOwnStarship(`*`st`*`)` só é permitido para uma espaçonave *st*. Ou seja, o estado `True` tem exclusividade global em relação à RV `IsOwnStarship(`*`st`*`)`.

Figura 7.3: MFrag `Starship Data`.

Outra contribuição importante para o PR-OWL foi a possibilidade de se definir as instâncias de uma entidade como estados possíveis de um nó. Na MFrag `Starship Data` (Figura 7.3) o nó `StarshipZone(`*`st`*`)` tem como estados possíveis todas as instâncias que existirem da entidade `Zone`. Essa possibilidade de definir uma entidade como estado possível de um nó foi aceita e incorporada na versão 1.03 do PR-OWL. Como conseqüência, foi necessário definir um padrão para a CPT associada a tal variável. Como o número de instâncias da entidade é desconhecido e não é possível utilizar o pseudocódigo neste caso, a solução adotada foi definir a CPT como uma distribuição uniforme, na ausência de informação específica mais detelhada.

7.1.3 MFrag

A Figura 7.4 ilustra os principais elementos de uma MFrag. As MFrags utilizadas no UnBBayes são as MFrags de domínio, que funcionam como um *template* para a

geração de uma SSBN, baseado nas instâncias de entidades e evidências presentes na situação específica.

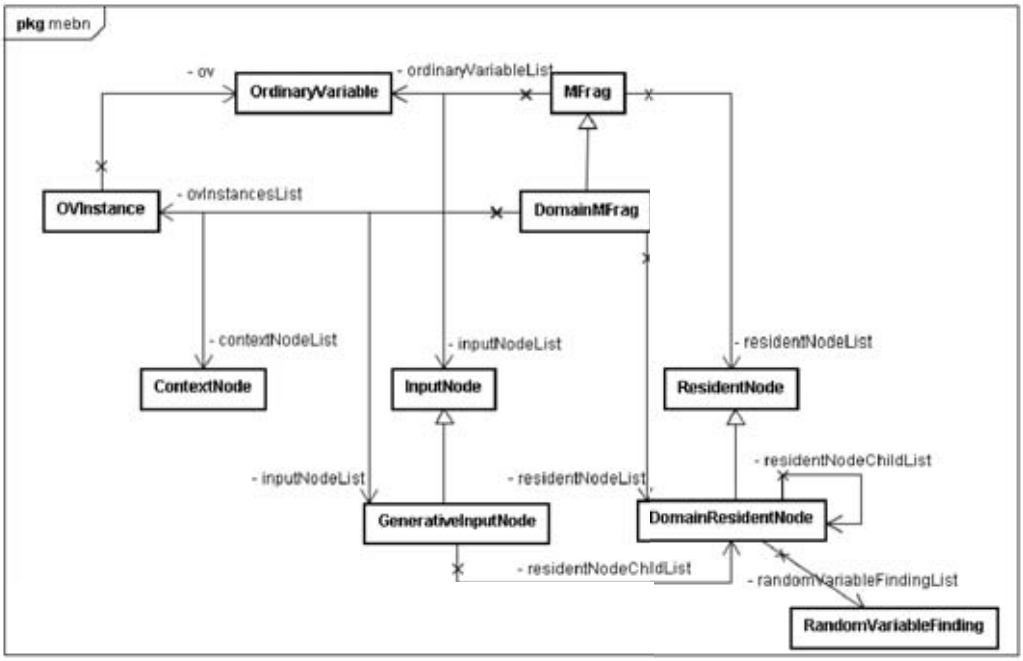

Figura 7.4: Modelagem de classes para MFrag.

Classe `MFrag` representa uma MFrag mais generalizada composta por uma lista de `InputNode` e uma de `ResidentNode`. Além disso, possui uma lista das suas variáveis ordinárias.

Classe `DomainMFrag` representa uma MFrag geradora: é a 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 (`GenerativeInputNode`) e pelos nós residentes (`DomainResidentNode`). Para possibilitar a geração da SSBN, essa MFrag possui uma referência para as instâncias (`OVInstance`) das entidades que estão relacionadas com suas respectivas variáveis ordinárias (`OrdinaryVariable`).

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

7.2 Tabela de Probabilidades

A proposta original da linguagem PR-OWL apresentada em [12] não contém uma especifição formal para fórmulas capazes de definir dinamicamente a CPT dos nós residentes instanciados durante a geração da SSBN. Essa funcionalidade é essencial quando o número de pais é desconhecido. No entanto, a proposta original

de PR-OWL apresenta um exemplo de pseudocódigo usado para definição de uma tabela de probabilidades para nós residentes de MEBN (Figura 4.2). Nessa pesquisa, com base na análise do pseudocódigo exemplo, foi proposta a gramática formal definida em notação BNF, apresentada na Figura 7.5.

Com base na especificação da gramática apresentada, foi projetado e desenvolvido um compilador com analisadores léxico, sintático e semântico. Além disso, é gerado um código intermediário usado para definir o pseudocódigo necessário para criar uma a CPT do nó na construção de uma SSBN para um dado contexto. Na implementação do UnBBayes, esse pseudocódigo é armazenado como objeto `String` no nó residente.

MEBN Table BNF

```
01. table ::= statement | if_statement
02. if_statement ::=
03.   "if" allop varsetname "have" "(" b_expression ")" statement
04.   "else" else_statement
05. allop ::= "any" | "all"
06. varsetname ::= ident ["." ident]*
07. b_expression ::= b_term [ "|" b_term ]*
08. b_term ::= not_factor [ "&" not_factor ]*
09. not_factor ::= [ "~" ] b_factor
10. b_factor ::= ident "=" ident
11. else_statement ::= statement | if_statement
12. statement ::= "[" assignment "]"
13. assignment ::= ident "=" expression [ "," assignment ]*
14. expression ::= term [ addop term ]*
15. term ::= signed_factor [ mulop signed_factor ]*
16. signed_factor ::= [ addop ] factor
17. factor ::= number | function | "(" expression ")"
18. function ::= possibleVal
19.   | "CARDINALITY" "(" varsetname ")"
20.   | "MIN" "(" expression ";" expression ")"
21.   | "MAX" "(" expression ";" expression ")"
22. possibleVal ::= ident
23. addop ::= "+" | "-"
24. mulop ::= "*" | "/"
25. ident ::= letter [ letter | digit ]*
26. number ::= [digit]+
```

Figura 7.5: Gramática para o pseudocódigo da tabela do nó residente.

A estrutura de classes responsável por compilar o pseudocódigo apresentado está na Figura 7.6.

Interface `ICompiler` responsável por definir o método para compilar o pseudocódigo e para gerar a CPT.

Classe `Compiler` implementa a interface `ICompiler` para compilar o pseudocódigo através da técnica descendente recursivo e gera a CPT (`PotentialTable`) para o nó a que se refere (`SSBNNode`).

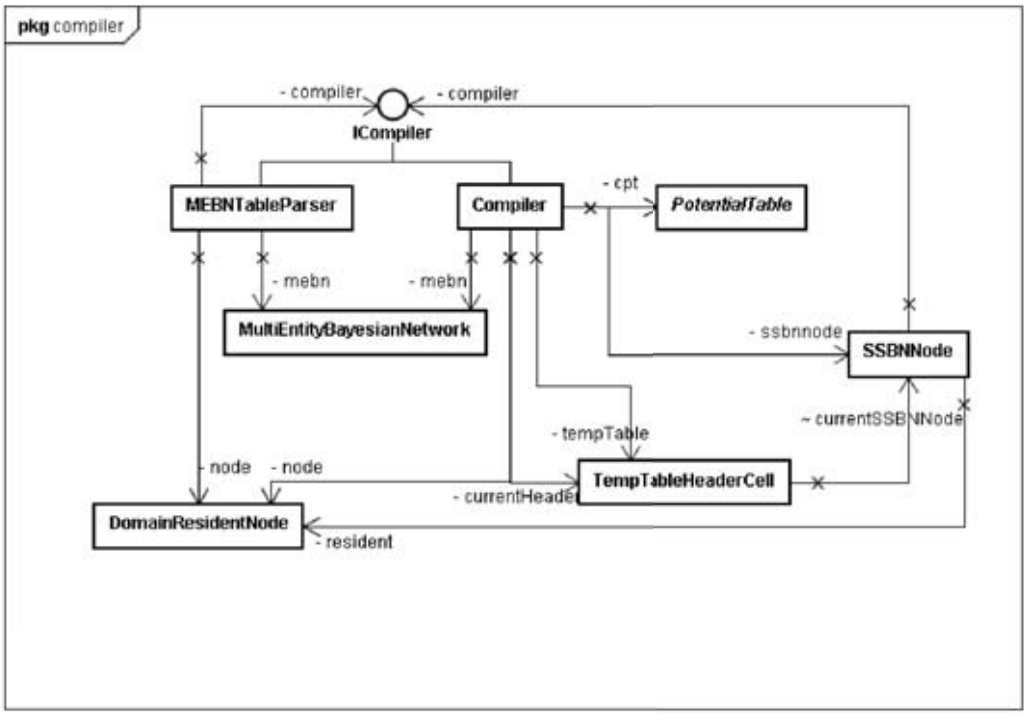

Figura 7.6: Modelagem de classes do compilador do pseudocódigo.

Classe `TempTableHeaderCell` classe interna de `Compiler` utilizada como código intermediário para calcular a CPT (`PotentialTable`).

Classe `SSBNNode` classe utilizada para representar um nó residente com informações específicas para gerar um nó de uma rede bayesiana comum.

Classe `MEBNTableParser` classe utilizada para compilar as tabelas dos nós residentes. Encapsula a classe `Compiler` incorporando alguns detalhes de consistência. Além de possibilitar o uso de outras implementações de `ICompiler`.

7.3 Interface Gráfica da MEBN

Nesta seção é apresentada a interface gráfica implementada no UnBBayes para oferecer o suporte à MEBN. A interface gráfica pode ser dividida em duas partes:

Edição e visualização da MTeoria permite que o usuário edite as MFrags e os diversos nós da MTeoria.

Geração da SSBN permite que o usuário entre com as evidências e o questionamento e gere a SSBN apresentando os resultados na tela de inferência probabilística de BN.

Nas seções seguintes, serão apresentadas a arquitetura da GUI e as principais classes utilizadas. Além disso, os principais painéis para a modelagem de uma MTeoria serão apresentados e explicados.

7.3.1 Arquitetura da GUI

A Figura 7.7 apresenta a arquitetura da GUI do UnBBayes com suporte à MEBN. O diagrama está simplificado: não são mostradas todas as relações de dependência e são mostradas apenas as classes principais ao entendimento. Já a seqüência de criação dos objetos relacionados à GUI da MEBN está detalhada na Figura 7.9.

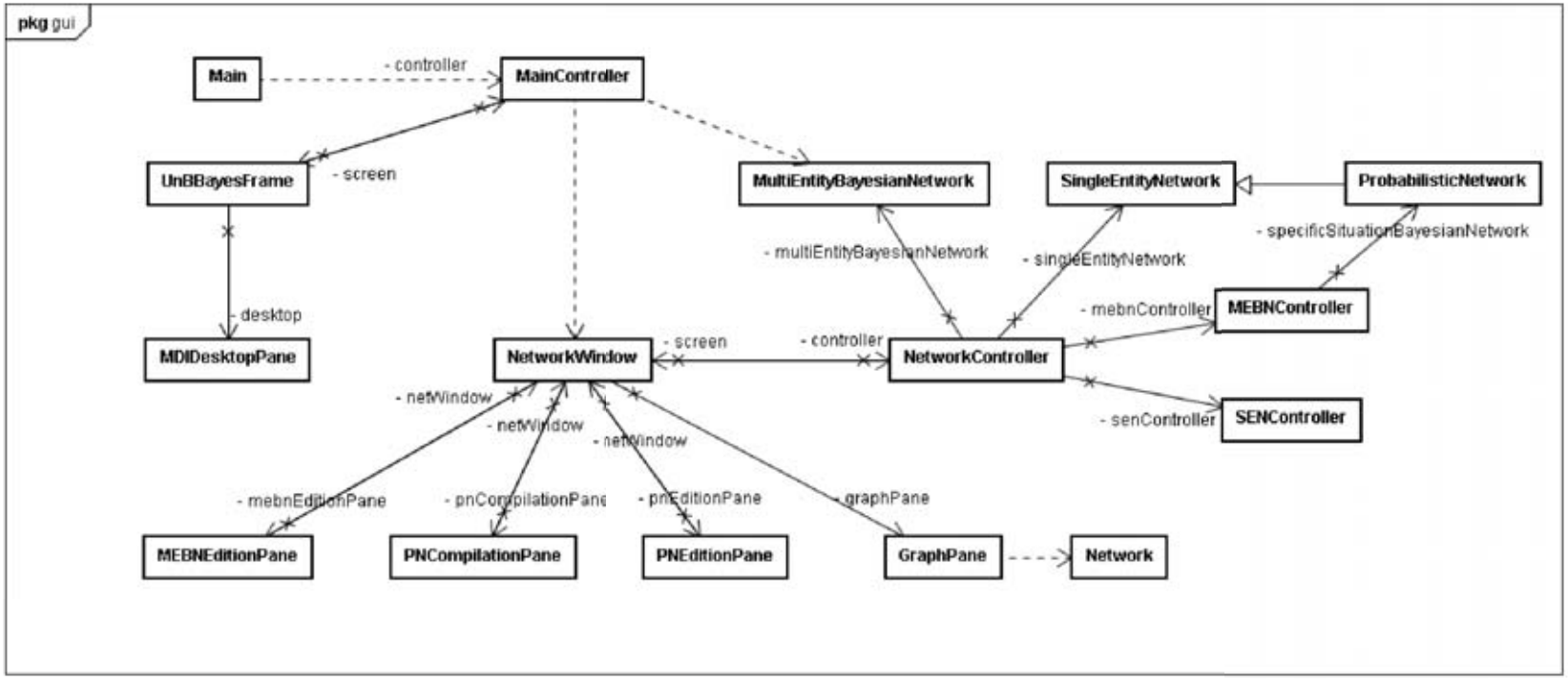

Figura 7.7: Arquitetura de classes da GUI.

Classe `Main` inicializa o UnBBayes criando o `MainController`.

Classe `MainController` classe responsável por criar, abrir e salvar MEBN e redes probabilísticas (PN) suportadas pelo UnBBayes, ou seja, rede bayesiana (BN), diagrama de influências (ID) e MSBN, sendo que esta última não foi mostrada na modelagem para simplificar o modelo.

Classe `UnBBayesFrame` responsável pela interface da tela principal do programa. Esta tela é independente do tipo de rede sendo exibida.

Classe `MDIDesktopPane` *desktop* do `UnBBayesFrame`. Permite a abertura dos *frames* relativos ao tipo de rede sendo tratada, além de permitir a abertura de múltiplas telas (conforme ilustrado na Figura 7.8).

Classe `NetworkWindow` representa a janela da rede. Responsável por criar o controlador para a rede, o `NetworkController`. Possui ainda o painel de edição correspondente a rede (MEBN ou PN) e o `GraphPane`.

Classe `GraphPane` responsável por desenhar o grafo na tela. Este grafo é composto por nós e arcos e pode ser tanto a representação da rede bayesiana quanto o desenho dos nós de uma MFrag. Implementa as interfaces `MouseListener` e `MouseMotionListener` para poder tratar os eventos de *mouse*.

Classe `NetworkController` classe responsável por delegar funções para execução em `SingleEntityNetwork` ou em `MultiEntityNetwork`. Inserir nós e propagar evidências, por exemplo. Delega as funções para um dos dois controladores de acordo com o tipo de rede sendo tratada.

Classe `SENController` controlador das operações realizadas sobre uma rede probabilística normal.

Classe `MEBNController` controlador das operações realizadas sobre uma rede bayesiana multi-entidade. Além disso, é responsável por criar a SSBN.

Classe `MEBNEditionPane` painel de edição de redes bayesianas multi-entidades.

Classe `PNEdtionPane` painel de edição de redes probabilísticas.

Classe `PNCompilationPane` painel para entrada de evidências e realização de inferências em redes probabilísticas.

Classe `MultiEntityBayesianNetwork` representa as MEBN.

Classe `SingleEntityNetwork` representa as redes probabilísticas que não possuem multi-entidades.

Classe `ProbabilisticNetwork` representa as redes probabilísticas, como BN e ID.

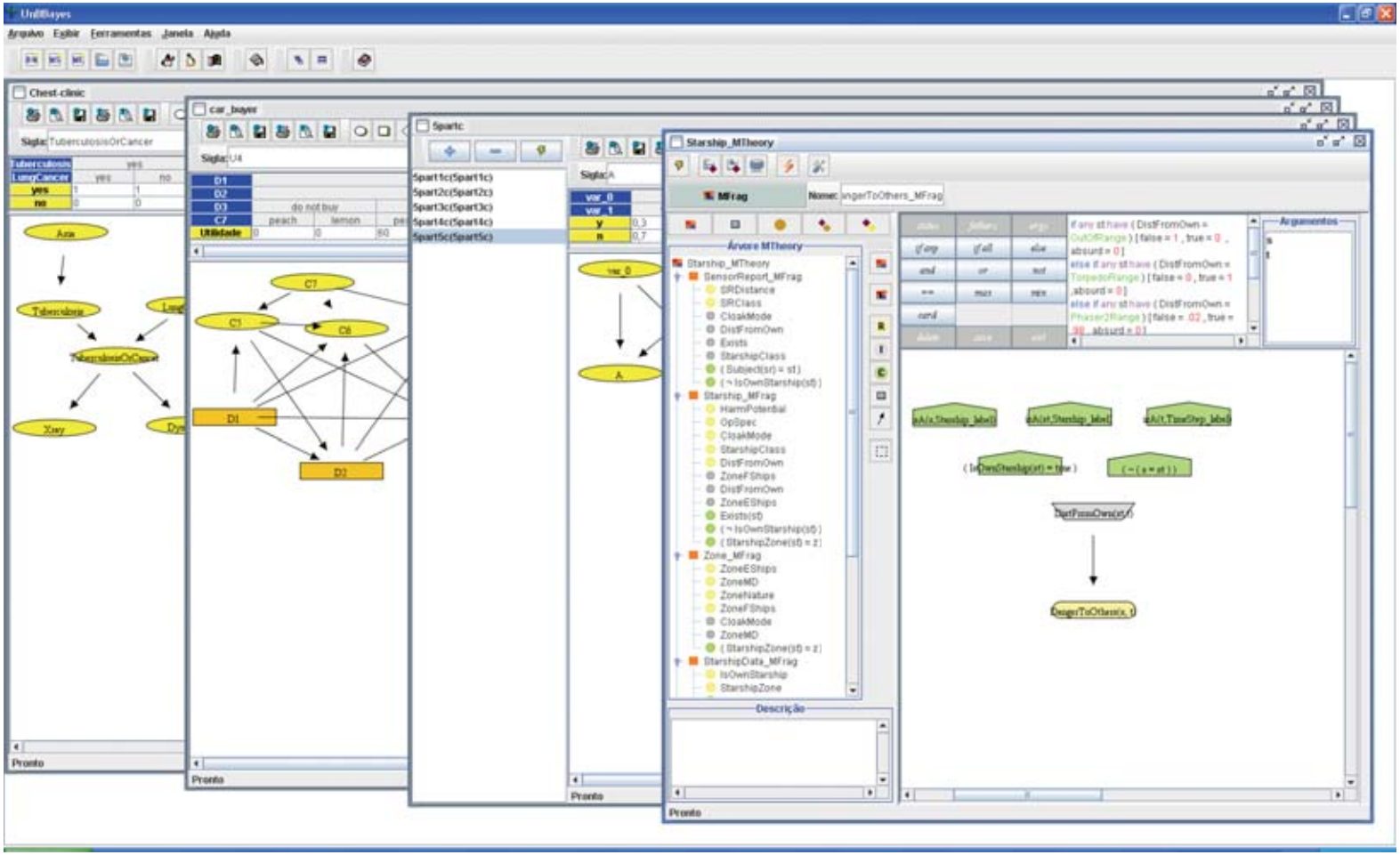

Figura 7.8: *Desktop* multi-telas do UnBBayes.

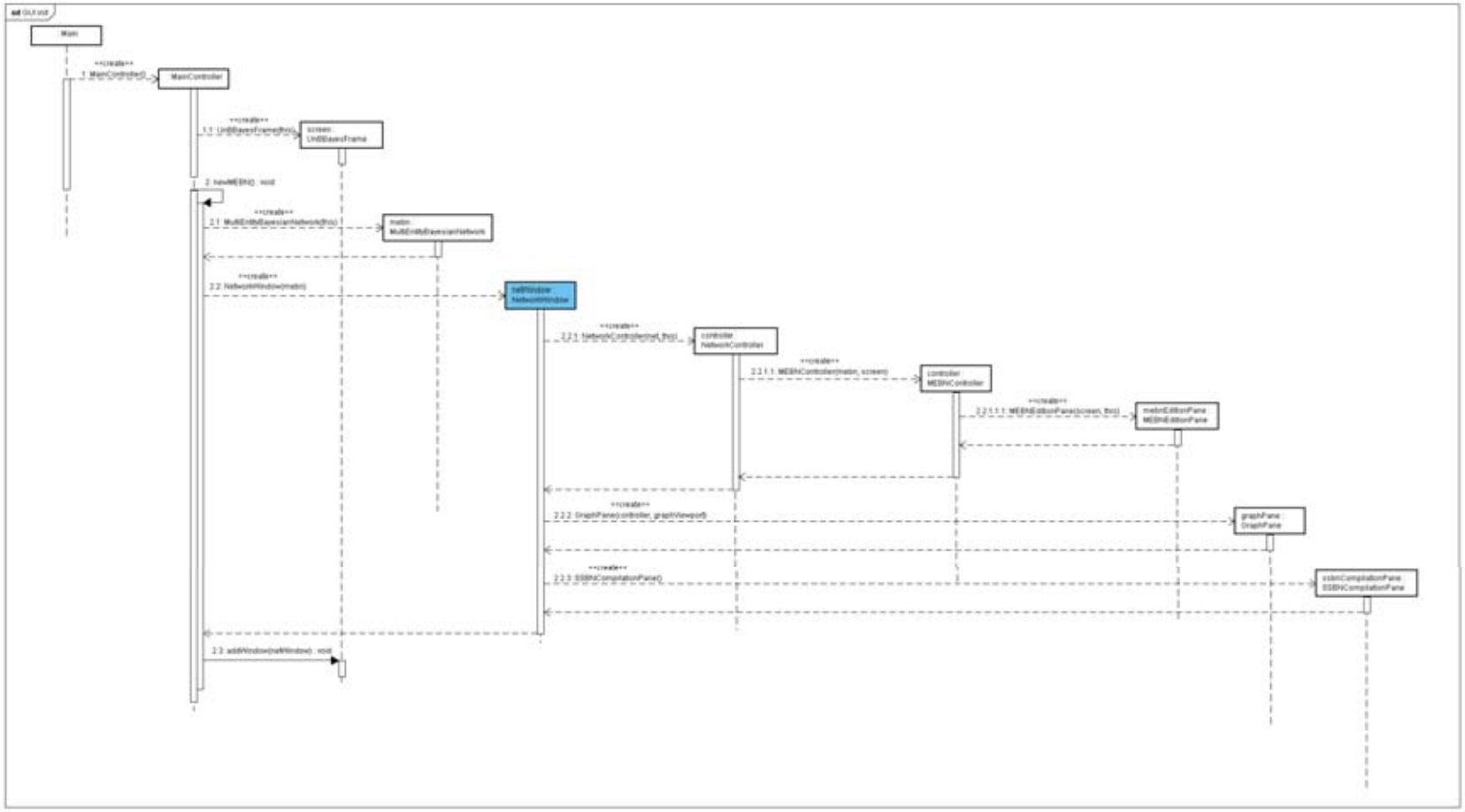

Figura 7.9: Diagrama de seqüência de criação dos objetos da GUI.

7.3.2 Tela principal

Figura 7.10 ilustra os componentes da tela de edição da MEBN.

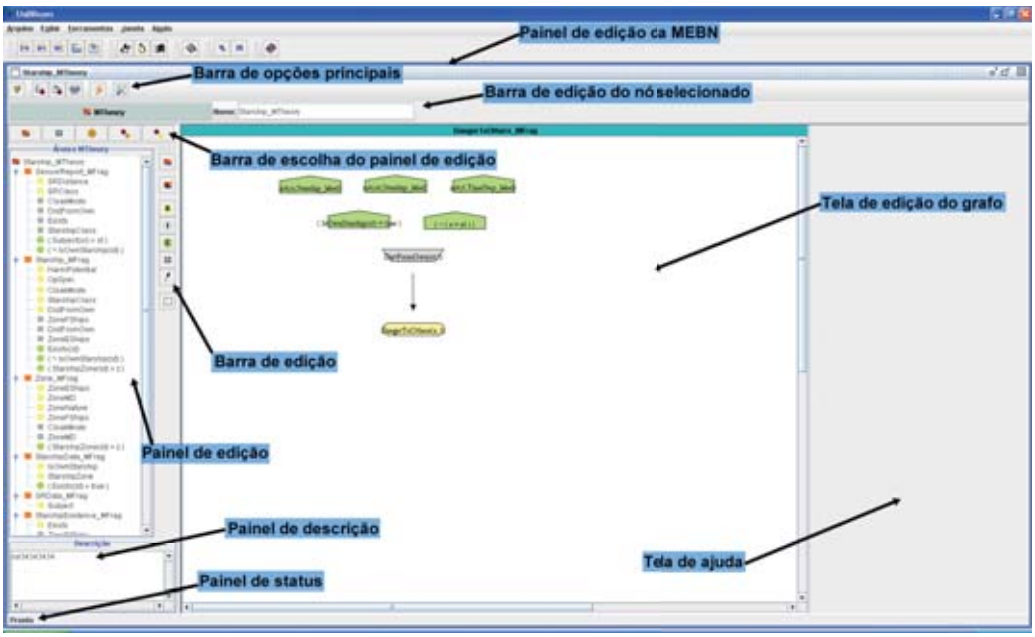

Figura 7.10: Componentes da tela de edição de MEBN.

O painel de edição da MEBN (classe `MEBNEditionPane`) é o novo painel construído para o UnBBayes a fim de oferecer todo o ambiente gráfico necessário para que o usuário construa MTeorias de forma simples e intuitiva. Todos os outros componentes gráficos da MEBN estão contidos dentro deste painel conforme mostrado na Figura 7.11.

A barra de escolha do painel de edição (`jtbTabSelection`) permite que o usuário escolha qual dos possíveis painéis de edição ele quer utilizar:

- visualizar a ávore com os elementos da MTeoria;
- visualizar as variáveis ordinárias existentes na MFrag que está sendo editada;
- visualizar o painel de edição de entidades;
- visualizar o painel de edição de evidências para entidades;
- visualizar o painel de edição de evidências para nós residentes.

O painel de edição (`jpTabSelected`) muda de acordo com o componente da MTeoria sendo editado. Cada nó editado tem um painel de edição padrão que surge sempre que ele é selecionado. Dependendo da característica sendo editada, alguns painéis não estarão disponíveis.

A barra de edição (classe `ToolBarEdition`) contém os botões para inserir os elementos principais de uma MTeoria: MFrags, nós residentes, nós de entrada, nós de contexto, arcos e seleção de vários elementos.

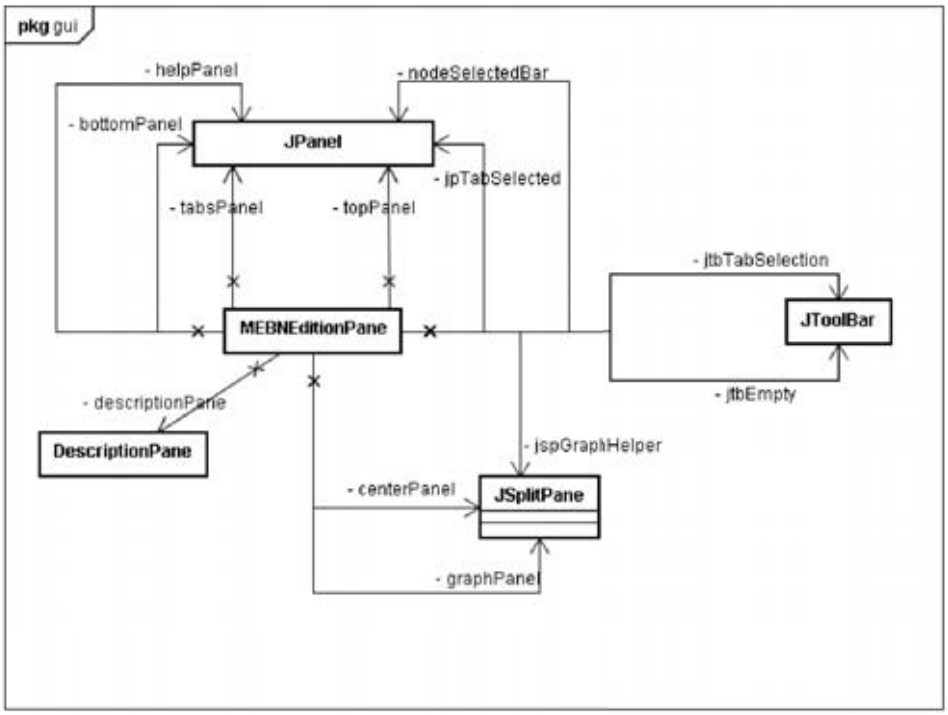

Figura 7.11: Painel de edição de MEBN - `MEBNEditionPane`.

O painel de descrição (`descriptionPane`) contém a descrição do componente selecionado. Permite que o usuário descreva os objetos da MTeoria para documentação. Os últimos quatro elementos descritos fazem parte do painel `tabsPanel`, sendo que o `jtbTabSelection` é acrescentado ao norte, o `jpTabSelected` é acrescentado ao centro, a barra de edição é acrescentado ao leste e o `descriptionPane` é acrescentado ao sul.

A tela de edição de grafo (`graphPanel`) apresenta o conteúdo da MFrag ativa e é onde devem ser inseridos os nós e arcos.

A tela de ajuda (`helpPanel`) ainda não está sendo utilizada. No entanto, ela será utilizada para mostrar:

- textos de ajuda para o usuário;
- relatórios;
- descrições comlexas de erros.

Os últimos dois elementos descritos são acrescentados no `jspGraphHelper`, sendo que o `graphPanel` é acrescentado à esquerda e o `helpPanel` é acrescentado à direita.

A barra de edição do nó selecionado (`nodeSelectedBar`) contém as informações sobre o nó selecionado e os botões para editar as diversas características do componente ativo (MTeoria, MFrag, nó residente, nó de entrada, nó de contexto ou variável ordinária). Se nenhum objeto estiver selecionado a barra é preenchida com um painel vazio (`jtbEmpty`).

A barra de opções principais (classe `ToolBarGlobalOptions`) possui botões para:

- entrar com um questionamento;
- carregar, salvar e limpar as evidências da base de conhecimento;
- voltar para a última SSBN gerada ou para o módulo de edição de MEBN;
- alterar as opções globais. O botão de opções globais já está presente na GUI, mas a funcionalidade ainda não foi implementada.

Tanto essa barra de opções principais quanto a `nodeSelectedBar` são acrescentadas ao `topPanel`.

No painel de status (`bottomPanel`) são apresentadas as informações sobre algumas operações efetuadas.

7.3.3 Painéis de Edição

Para a edição de todos os objetos da MEBN foi necessário a criação de vários painéis diferentes para as diversas características editáveis. O diagrama na Figura 7.12 mostra as classes que implementam os novos painéis criados.

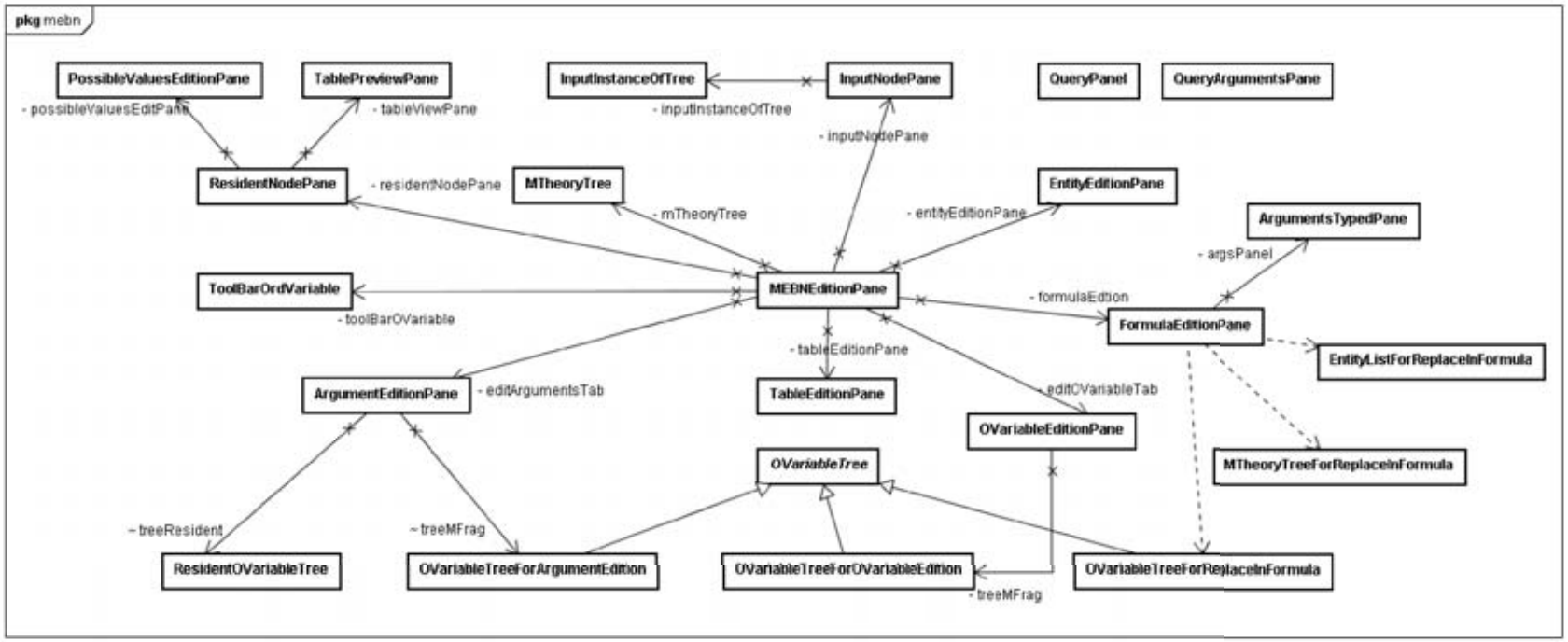

Figura 7.12: Classes utilizadas nos painéis de edição de MEBN.

A edição de MFrag consiste em adicionar, remover e renomear MFrags. A MFrag ativa terá o seu conteúdo exibido na tela de edição do grafo conforme ilustrado na Figura 7.13.

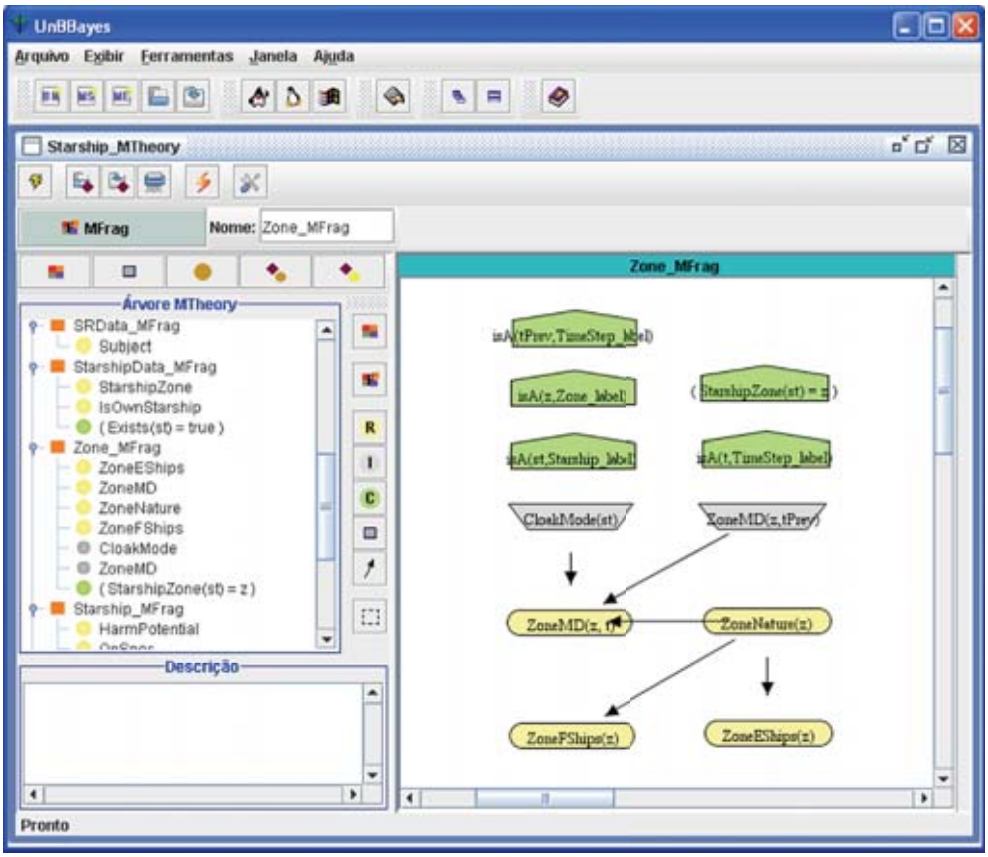

Figura 7.13: Edição de MFrag.

A seleção de MFrags é realizada através da árvore da MTeoria que é implementada pela classe `MTheoryTree`. Essa classe mostra as MFrags de uma MTeoria e os seus respectivos nós (nós de contexto, nós residentes e nós de entrada). Implementa *listeners* para as opções básicas de edição destes: inserir, remover e selecionar.

A edição de um nó residente permite:

- adicionar, renomear e excluir nós, tanto através do grafo quanto pela árvore da MTeoria;
- editar os estados através do painel de edição de possíveis valores (`PossibleValuesEditionPane`). Este painel contém uma lista com os possíveis valores atuais e botões para adicionar um novo valor ou excluir o valor selecionado.
- editar os argumentos, inserindo e retirando variáveis ordinárias através do painel de edição de argumentos (`ArgumentEditionPane`). Este painel, ilustrado na Figura 7.14, contém duas árvores: uma com as variáveis ordinárias existentes na MFrag (`OVariableTreeForArgumentEdition`) e outra com

as variáveis ordinárias existentes no nó (`ResidentOVariableTree`). A seta para baixo permite que a variável selecionada na árvore da MFrag seja inserida no nó (o mesmo pode ser feito clicando-se duas vezes nesta). O botão "+" permite que se adicione uma nova variável ordinária na MFrag ou como argumentos do nó residente, enquanto o botão "-" permite que se exclua a variável ordinária selecionada na árvore inferior da lista de argumentos do nó.

- editar a tabela de distribuição probabilística (`EditionPane`). A edição é feita através de um editor de texto com painéis contendo listas com os diversos objetos que podem ser utilizados para a definição da distribuição (listagem dos nós de entrada que influenciam o nó residente, listagem dos estados dos nós de entrada, listagem dos estados do nó residente, listagem dos argumentos, etc). Há botões que permitem a inserção de auto-texto. O pseudocódigo da tabela é colorido, facilitando a visualização do diversos *tokens*.

Figura 7.14: Painel de edição de argumentos.

A Figura 7.15 ilustra a tela de edição de nós residentes com o painel de edição de possíveis valores selecionado. É apresentado também um painel com o pseudocódigo da distribuição de probabilidades definida para o nó.

A edição de um nó de entrada permite:

- adicionar, renomear e excluir nós de entrada tanto através do grafo quanto pela árvore da MTeoria.

- editar de qual nó residente o nó de entrada é instância. Para tanto, a classe `InputInstanceOfTree` cria uma árvore com todos os nós residentes da MTeoria, separados por MFrag. Basta clicar duas vezes no nó residente para que ele seja setado como a referência para o nó de entrada (Figura 7.16).

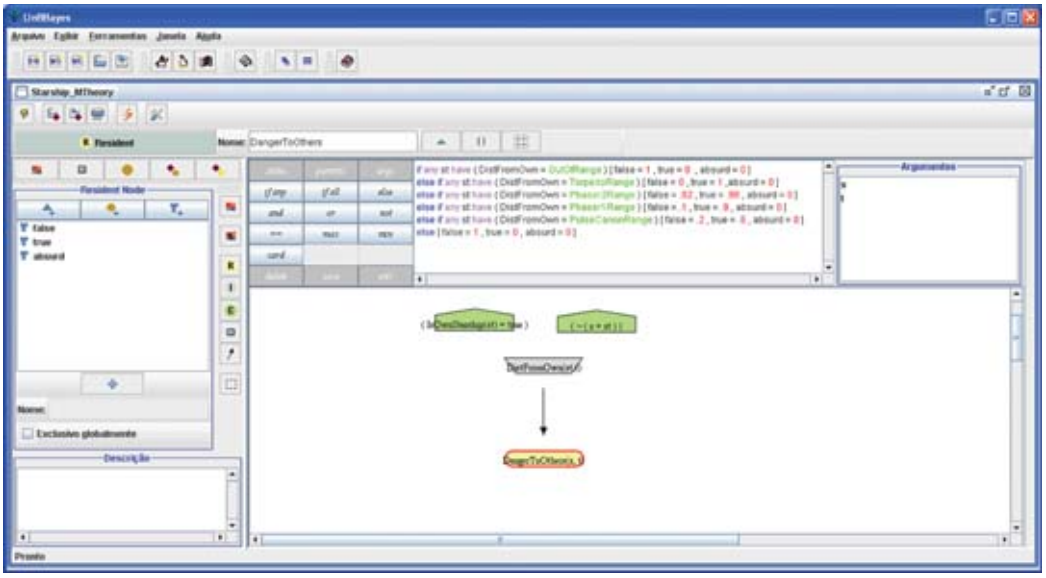

Figura 7.15: Edição de nó residente de uma MFrag.

A edição de um nó de contexto permite:

- "adicionar, renomear e excluir nós de contexto tanto através do grafo quanto pela árvore da MTeoria. "editar a fórmula da FOL que o nó representa. O painel para edição é implementado pela classe FormulaEditionPane. A fórmula é editada como uma árvore onde os operandos de um nó operador são representados como nós filhos (Figura 5.19).

A árvore representa a fórmula `= (StarshipZone(st), z)`, ou em forma infixada: `(StarshipZone(st) = z)`. O objetivo deste modo de criar fórmulas é diminuir a liberdade para o modelador de forma a evitar construções irregulares. Quando um operador é criado, a quantidade de operandos definidos para aquele operador define quantos nós filhos serão criados, de forma que a única coisa que o modelador poderá fazer é definir quais argumentos possuem estes nós filhos. A árvore é implementada pela classe `FormulaTree`.

Conforme ilustrado, os operandos podem ser nós, variáveis ordinárias, entidades ou variáveis (utilizadas nos quantificadores). Ao escolher uma destas opções, será exibido para o usuário uma árvore com todos os objetos do tipo escolhido presentes na MTeoria para seleção (classe `MTheoryReplaceInFormula` para seleção de nós, classe `OVariableTreeForReplaceInFormula` para seleção de variável ordinária ou classe `EntityListForReplaceInFormula` para seleção de uma entidade). Os operadores são os `BuiltInRv` suportados: `and`, `or`, `not`, `equalTo`, `implies`, `iff`, `exists` e `forAll`.

A padronização dos elementos gráficos para representar nós residentes, de entrada e de contexto foi definida a partir de uma interação da equipe responsável pelo desenvolvimento do UnBBayes com a equipe responsável pela linguagem PR-OWL e publicada em [15].

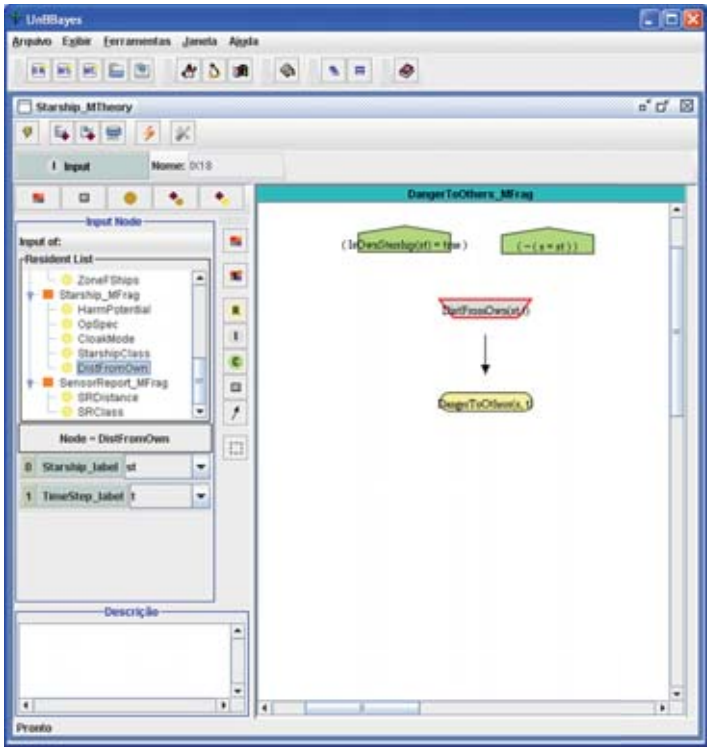

Figura 7.16: Edição de nó de entrada de uma MFrag.

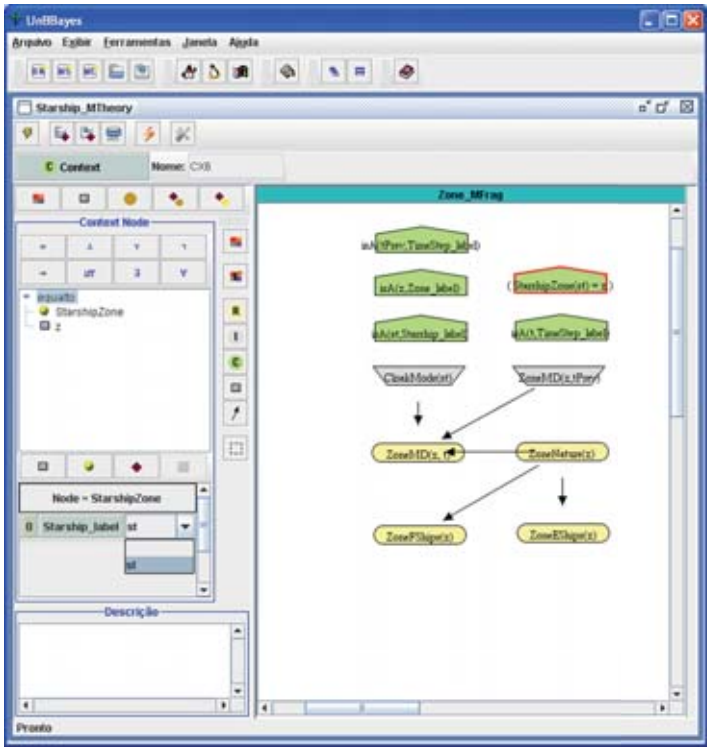

Figura 7.17: Edição de nó de contexto de uma MFrag.

7.3.4 Edição de Entidades e de Variáveis Ordinárias

A edição de entidades e de variáveis ordinárias permite a criação, eliminação e renomeação. A edição das entidades é realizada através do painel `EntityEditionPane` que apresenta uma árvore contendo todas as entidades definidas na MTeoria (`EntityTree`), conforme exemplo apresentado na Figura 7.18.

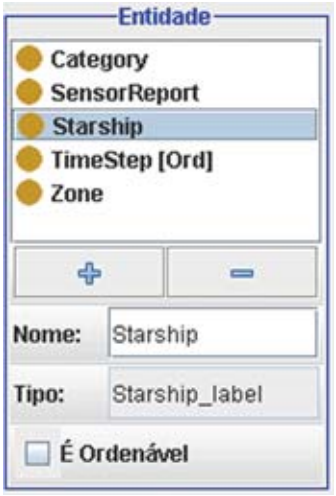

Figura 7.18: Edição de entidades de uma MTeoria.

A edição das variáveis ordinárias é realizada através do painel `OVariableEditionPane` que apresenta uma árvore contendo todas as variáveis ordinárias definidas na MFrag ativa (`OVariableTreeForOVariableEdition`). Para inserir uma variável ordinária, é preciso selecionar o botão correspondente da barra de edição e clicar em alguma área da tela de edição do grafo (Figura 7.19)

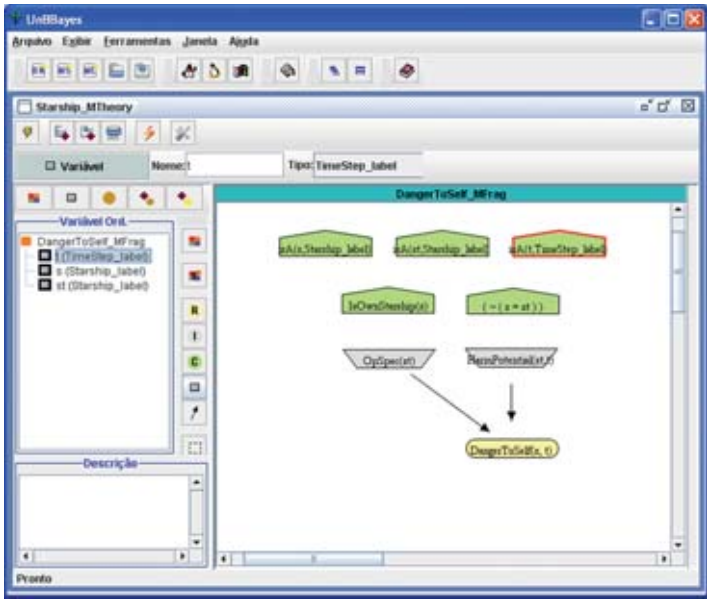

Figura 7.19: Edição de variável ordinária de uma MFrag.

7.3.5 Edição do Pseudocódigo para a definição da CPT

O editor de pseudocódigo (Figura 7.20) visa facilitar a edição deste pelo modelador.

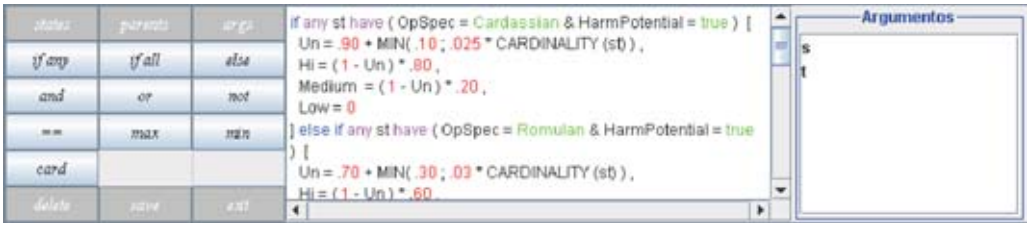

Figura 7.20: Editor de CPT para nó residente.

Os botões permitem que se insira auto-texto. O painel direito pode ser alternado entre um painel que lista os argumentos do nó editado, os nós pais e seus respectivos estados, e os possíveis estados do nó. Com este suporte, é possível que o usuário edite a tabela utilizando o teclado apenas para entrar com as fórmulas numéricas que definem o valor de cada um dos estados.

7.3.6 Entrada de evidências e de questionamento

Antes de gerar uma SSBN, é preciso entrar com as evidências que definem o contexto da situação específica. Para isso, foram desenvolvidos dois painéis para entrada de evidências: um para entidades (Figura 7.21) e outro para os nós residentes (Figura 7.22).

Para entrar com uma evidência de uma determinada entidade, basta selecionar a tela de entrada de evidências para entidades, preencher o nome da entidade que representa a evidência conhecida e clicar na seta para baixo de forma a adicionar essa evidência na base de conhecimento.

Para entrar com uma evidência de um determinado nó residente, basta selecionar a tela de entrada de evidências para nós residentes, selecionar o nó residente que deseja informar a evidência conhecida, clicar no botão de editar (Figura 7.22), selecionar os argumentos do nó residente específico, selecionar o estado conhecido e clicar no botão "+" (Figura 7.23).

Uma vez finalizada a modelagem da MTeoria e informadas todas evidências da situação específica, é possível fazer um questionamento que significa selecionar um nó residente de interesse (Figura 7.24) e informar os valores de seus argumentos (Figura 7.25).

7.4 Avaliação dos nós de contexto

Como apresentado na seção 6.1 o UnBBayes adotou a API PowerLoom para avaliação de fórmulas da FOL. Dessa forma, a seção 7.4.1 descreve a integração da API PowerLoom ao UnBBayes e o mecanismo utilizado para a avaliação dos nós de contexto.

Figura 7.21: Entrada de evidências das entidades.

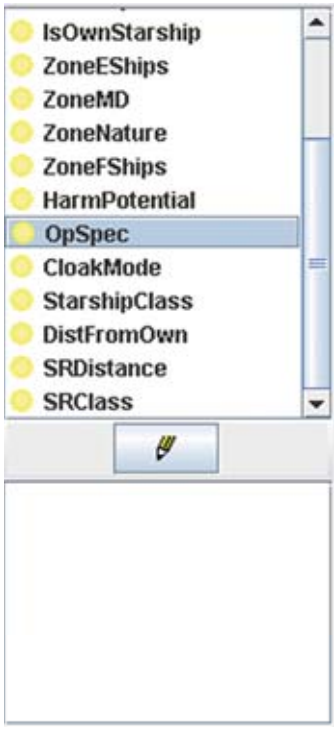

Figura 7.22: Entrada de evidência - seleção do nó residente desejado.

Figura 7.23: Preenchendo informações do nó residente específico.

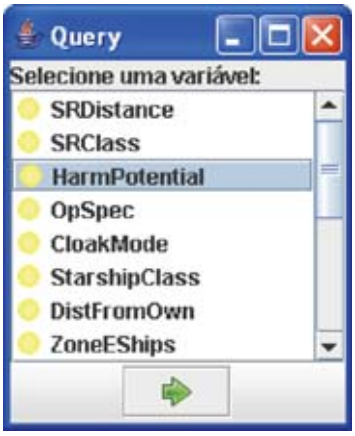

Figura 7.24: Questionamento - seleção do nó residente.

Figura 7.25: Questionamento - definição dos argumentos.

7.4.1 Uso do PowerLoom

Uma MFrag representa uma distribuição de probabilidade de instâncias de seus nós residentes, dados os valores das instâncias dos seus pais no grafo. Para que estas distribuições se apliquem, as restrições de contexto devem ser satisfeitas. A API PowerLoom foi utilizada para a avaliação dos nós de contexto.

O trabalho com a API é facilitado através de uma interface especial (PLI), que evita que o usuário utilize os objetos STELLA diretamente (o que exigiria um conhecimento profundo do processo seguido para se fazer as operações). A maioria dos métodos presentes na interface PLI possuem duas versões: uma que recebe como parâmetros objetos STELLA e outra que recebe como parâmetro uma `string` contendo o comando PowerLoom. Ambas retornam objetos STELLA.

As classes presentes na implementação da base de conhecimento (KB) são apresentadas na Figura 7.26.

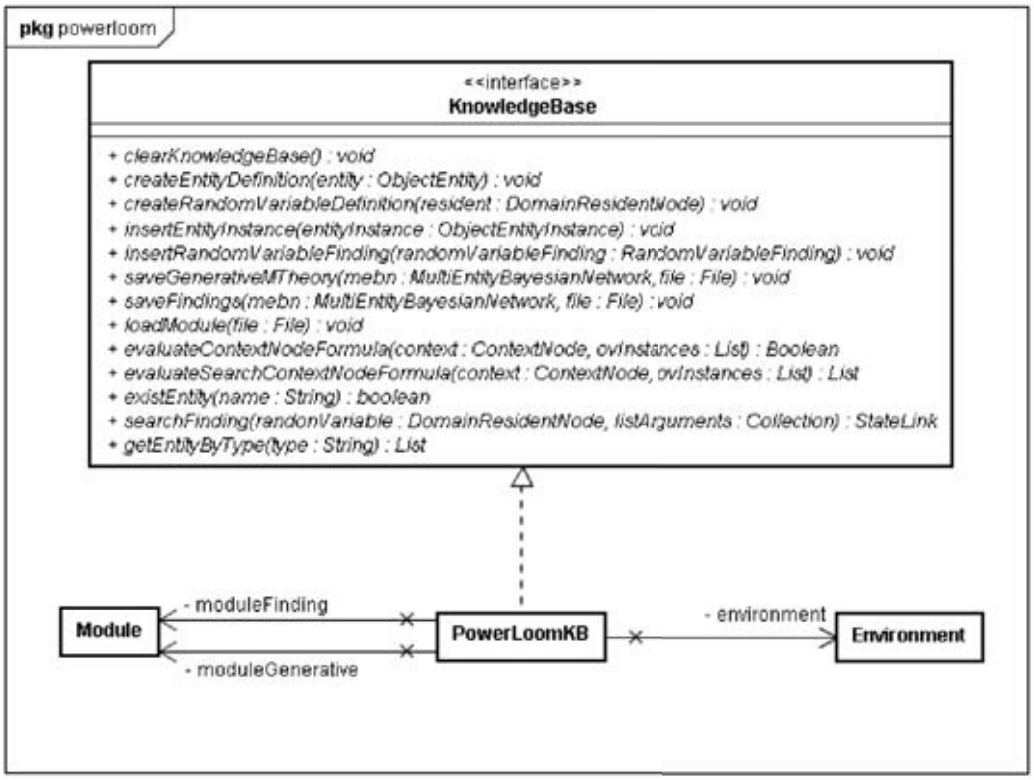

Figura 7.26: Modelagem do uso do PowerLoom.

Interface `KnowledgeBase` contém os métodos que a base de conhecimento utilizada pelo UnBBayes deve conter, afim de permitir a avaliação dos nós de contexto.

Classe `PowerLoomKB` implementação da interface `KnowledgeBase` utilizando o PowerLoom.

Os passos para a avaliação dos nós de contexto são os seguintes:

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

A seguir é analisado como o PowerLoom é utilizado em cada um dos passos e como funcionam os métodos da classe `PowerLoomKB`.

O mapeamento da MTeoria geradora para a KB corresponde a inserir na base de conhecimento a modelagem da MTeoria geradora. Essa inserção na KB é precedida de uma tradução da modelagem em definições de entidades e de variáveis aleatórias no PowerLoom.

Como a MTeoria geradora é reaproveitada para a geração de diversas SSBN, é possível salvar um módulo PowerLoom com estas definições, de forma que ao se gerar outra SSBN se aproveite este super-módulo e a partir deste se acrescente apenas as assertivas relativas a SSBN atual. Este super-módulo precisa ser modificado apenas se o usuário fizer modificações na modelagem da MTeoria geradora.

O UnBBayes implementa uma MTeoria tipada, de forma que estes tipos devem ser mapeados no PowerLoom. O PowerLoom possui suporte completo a definições de tipos e subtipos, oferecendo mecanismos de inferência para trabalhar com eles. Ao utilizar o PowerLoom, no entanto, assumimos que o UnBBayes já fez todas as verificações quanto a corretude dos tipos definidos na MTeoria, informando ao usuário possíveis erros. Foi assumida essa opção porque o suporte a tipos do PowerLoom exige que se avalie toda a base de conhecimento após a inserção de cada assertiva, para verificar se a nova assertiva inserida possui algum erro de tipo. A verificação de tipos na montagem das MTeorias geradoras é garantida pelo UnBBayes através da própria GUI que não permite inconsistências. O UnBBayes também faz as verificações necessárias ao receber as informações do usuário para a montagem da SSBN, reportando os erros.

As funções utilizadas para a tradução da MTeoria geradora para a base de conhecimento são descritas a seguir (Figura 7.26):

- `createEntityDefinition(entity: ObjectEntity)`: faz a tradução de um objeto de uma entidade para a sintaxe do PowerLoom e grava a definição na KB;
- `createRandomVariableDefinition(resident: DomainResidentNod)`: faz a tradução de um nó residente para a sintaxe do PowerLoom e grava a definição na KB.

Os objetos `ObjectEntity` são traduzidos como conceitos na KB. A entidade `Starship`, por exemplo, seria traduzida para: `(defconcept Starship ?st)`.

Os nós residentes poderiam ser traduzidos para relacionamentos ou para funções. Como na implementação um nó residente é mapeado para apenas um de seus possíveis valores após a entrada das evidências, optamos por traduzir o nó residente para funções. O nó `StarshipZone`, por exemplo, seria traduzido para: `(deffunction starshipZone ((?st Starship )) :-> (?z StarshipZoneState))`

Os possíveis estados foram mapeados utilizando-se o conceito de `collections` no PowerLoom, onde um conceito é definido em termo de um conjunto enumerado utilizando a `built-in setof`:

- `(defconcept StarshipZoneState (?z) :<=> (member-of ?z setof ZNBlackHoleBoundary, ZNDeepSpace, ZNPlanetarySystem))`

Um exemplo completo de um super-módulo representando a MTeoria geradora *Star Trek* é apresentado no CD.

Para a inserção da situação especifica na base de conhecimento são utilizados os dois métodos:

- `insertEntityInstance(entityInstance: ObjectEntityInstance)`: traduz uma instância de uma entidade para a sintaxe do PowerLoom e a grava na KB.
- `insertRandomVariableFinding(randomVariableFinding: RandomVariableFinding)`: traduz uma evidência de uma variável aleatória e a grava na KB.

As evidências são mapeados para a base de conhecimento via o comando `assert`. Ao inserir uma instância da entidade `Starship` com o nome `Enterprise` na KB, a tradução feita é a seguinte:

- `(assert (Starship Enterprise))`

Como já visto, em PR-OWL o nome de uma variável começa com ponto de exclamação. Assim, nos exemplos a seguir o ponto de exclamação representa um identificador e não a negação lógica. Em PowerLoom variáveis não precisam iniciar com exclamação. Ao inserir uma evidência sobre uma determinada variável aleatória, temos duas possibilidades. Se os estados da variável aleatória forem booleanos, por exemplo, afirmar que *`!Enterprise`* é sua própria nave, então temos:

- `(assert (IsOwnStarship Enterprise))`

Caso contrário, ou seja, se os estados da variável aleatória não forem booleanos, por exemplo, afirmar que a zona de *`!Enterprise`* é *`!Z2`*, então temos:

- `(assert (= (StarshipZone Enterprise) Z2))`

Um exemplo completo de evidências gravadas em um arquivo do .plm através do UnBBayes é apresentado no CD.

As fórmulas dos nós de contexto são diretamente mapeadas para o PowerLoom. Os *built-ins* implementados no UnBBayes estão presentes no PowerLoom: `and`, `or`, `not`, `implies`, `iff`, `equalTo`, `exists` e `forall`. As variáveis ordinárias são mapeadas para variáveis. Cada nó de contexto é traduzido para um comando `ask`. Além disso, eles podem ser divididos em duas categorias: os de fórmula simples e os de fórmula complexa. Uma fórmula simples é aquela que não possui variáveis e que o resultado é um valor booleano. Uma fórmula complexa é aquela que retorna uma lista de entidades que satisfazem uma determinada restrição. Na MFrag `Starship` (Figura 7.27), temos os dois casos.

Para a avaliação dos nós de contexto são utilizados os dois métodos:

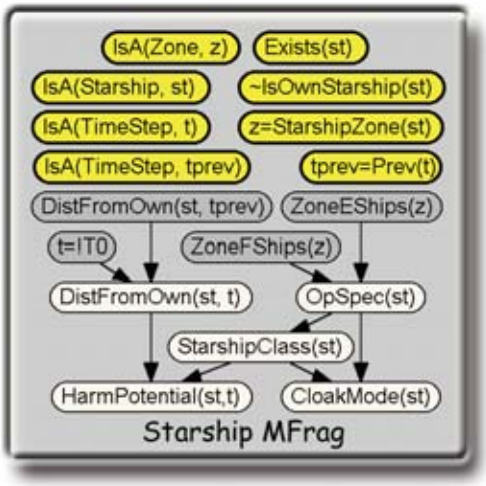

Figura 7.27: Nós de contexto da MFrag `Starship`.

- `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 as entidades que satisfazem a restrição definida no nó de contexto.

Uma fórmula simples, apresentada na Figura 7.27, é `Exists(st)` que no UnBBayes é representada como:

- `( Exists(st) )`

Ao ser avaliada para uma `Starship` *`!Enterprise`*, por exemplo, é traduzida para o PowerLoom como:

- `(ask (not( IsOwnStarship Enterprise ) ))`

Uma fórmula complexa, apresentada na Figura 7.27, é `z=StarshipZone(st)` que no UnBBayes é representada como:

- `( StarshipZone(st) = z )`

Ao ser avaliada para uma `Starship` *`!Enterprise`*, por exemplo, é traduzida para o PowerLoom como:

- `(retrieve ( = ( StarshipZone Enterprise ) ?z))`

Observe que ao fazer a avaliação, as variáveis ordinárias da fórmula deverão estar corretamente substituídas pelos valores reais da situação especifica. 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 o PowerLoom as fórmulas corretas.

A avaliação dos nós de contexto de uma 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 da MFrag serão instanciadas utilizando-se as suas distribuições locais.

7.5 Regras de consistência

Como resposta a uma solicitação feita durante o desenvolvimento do UnBBayes, a equipe da linguagem PR-OWL apresenta uma lista das regras de consistência para MEBN em [14]. A seguir serão apresentadas as regras de consistência implementadas no UnBBayes. As que não foram implementadas se deve ao fato do UnBBayes não possuir polimorfismo e não suportar árvore de tipos nem composição de funções.

7.5.1 Definições de variáveis aleatórias

1. Escopo da variável aleatória
    - Nomes de variáveis aleatórias são globais à MTeoria, portanto essa verificação se resume na garantia de unicidade das variáveis aleatórias na MTeoria.
2. Argumentos das variáveis aleatórias
    - Cada variável aleatória possui um número fixo de argumentos e cada argumento possui um tipo associado.
3. Tipo das variáveis aleatórias
    - Cada variável aleatória possui um tipo associado a seu valor possível. Por padrão, seu valor possível é o identificador da entidade que se situa na folha da árvore de tipos.
        - Deve ser possível restringir os valores possíveis a um subconjunto de um tipo pré-definido. Por exemplo, restringir que n-1 seja o único valor possível para Prev(n). A definição de tipos pode requerer uma restrição estrutural que garanta uma distribuição computável.
        - Muitas variáveis podem ter valores categóricos (como alto, médio e baixo). Nesse caso, seria conveniente definir um rótulo genérico, o `CategoricalLabel`. Claro, não seria desejável definir os valores possíveis como todo e qualquer valor categórico possível, se é conhecido que devem ser somente alto, médio ou baixo.
4. RV *built-in*
    - `isA`: é uma variável aleatória de 3 estados lógicos (true, false e absurd). Um dos argumentos deve ser um rótulo do tipo e o outro pode ser de qualquer tipo (pois é uma entidade).

- `Entity`: é uma constante de tipo `TypeLabel` e valor `Entity`.
- `TypeLabel`: é uma constante de tipo `TypeLabel` e valor `TypeLabel`.
- Valores lógicos: são constantes representando `true`, `false` e `absurd`.

7.5.2 Global

1. Acíclico

   - Para implementações que não incluem recursividade, a verificação de ciclos na definição de tipos é relativamente mais simples. Quando existe recursividade, é preciso dividir os tipos em recursivos ou não-recursivos. Todos os nós na subárvore de tipos recursivos são também recursivos, da mesma forma com os não-recursivos. Uma variável aleatória é considerada como recursiva quando qualquer um dos seus argumentos tem tipo recursivo. Pode-se definir uma ordem parcial entre instâncias de variáveis aleatórias recursivas. Uma instância está "antes" de outra quando todos os seus argumentos estão "antes" dela na sua árvore de tipos. Nesse caso, um arco só pode ser criado entre dois nós caso seja respeitada essa ordem parcial.

7.5.3 Local

1. Definição de nós

   - Cada nó em uma MFrag aponta para uma variável aleatória na lista global da MTeoria. O número de argumentos entre esses nós e a variável aleatória deve ser equivalente.

2. Variáveis ordinárias

   - Para cada variável ordinária da MFrag deve haver exatamente uma variável aleatória de contexto que especifica o seu tipo.
   - Somente instâncias desse tipo podem ser substituídas para a variável ordinária.
   - A variável ordinária tem escopo local à MFrag, ou seja, se uma mesma variável ordinária aparece em diferentes pontos da MFrag, deve ser substituída pela mesma entidade. Essa condição não se aplica a variáveis ordinárias que aparecem em diferentes MFrags.

3. Conectividade de nós de entrada

   - Cada nó de contexto deve ser raiz no grafo do fragmento.

4. Conectividade de nós de contexto

   - Cada nó de contexto é raiz no grafo do fragmento. Esses nós não devem mostrar arcos para nenhum nó; entretanto, está implícito que os nós de contexto são pai dos nós que compartilham uma mesma

variável ordinária. Na construção da SSBN, a distribuição local de um nó se aplica somente quando o contexto é verdadeiro. Caso contrário, a distribuição padrão é aplicada.

- Portanto, quando há incerteza sobre o valor do nó de contexto, ele deve ser explicitamente representado como um pai do nó que ele afeta. A distribuição de probabilidade deve ser, nesse caso, definida como o abaixo:
  - Se o nó de contexto é verdadeiro com probabilidade 1, não há a necessidade de colocá-lo explicitamente no SSBN. Basta usar a distribuição declarada.
  - Se o nó de contexto é falso com probabilidade 1, não há a necessidade de colocá-lo na SSBN. Basta usar a distribuição padrão.
  - Senão, o nó de contexto deve ser considerado como pai de todos os nós residentes. Sobre a distribuição de probabilidades das variáveis aleatórias, quando o contexto tiver valor verdadeiro, os filhos devem ter a distribuição que reflita a declarada; enquanto que ao ter o contexto falso a distribuição dos filhos deve refletir a distribuição padrão.

5. Acíclico

   - 1. O grafo do fragmento não deve conter ciclos.

6. Tipo das variáveis ordinárias

   - Cada variável aleatória tem um rótulo associado indicando o tipo, que descreve seus possíveis valores.

7. Distribuição local

   - Uma verificação local de consistência da distribuição de probabilidades deve ser feita para na MFrag. Especificamente:
     - Cada variável aleatória precisa ter definido uma probabilidade para cada estado em particular, deve estar em um intervalo válido de probabilidade (entre 0 e 1) e sua soma precisa dar 1. Dentro da MFrag, deve ser verificado se todos os estados dos parentes foram considerados pelos seus filhos.
     - Este tipo de verificação é imediato quando tratamos de redes bayesianas (seja, SSBN já criada). Entretanto, ao verificar uma MFrag, que seria um *template* de informações para a instanciação de variáveis aleatórias na construção de SSBN, essa verificação não é trivial. Resumindo, deve haver uma maneira de se gerar uma CPT antes do processo de instanciação, para que seja possível a verificação local da distribuição. Dependendo da implementação, uma linguagem *script* pode ser utilizada para a declaração de tabelas, mas que devem ser devidamente consideradas na consistência.

7.6 Tipo, tipo ordenável e recursão

Desenvolvida como uma lógica de propósito abrangente, MEBN não garante a tipagem das entidades por padrão, embora seja facilmente implementada com as definições das MFrags `Entity Type` e `IsA`, como pode ser visto na Figura 7.28. No entanto, durante a implementação de MEBN, ficou claro a necessidade de se implementar essa funcionalidade.

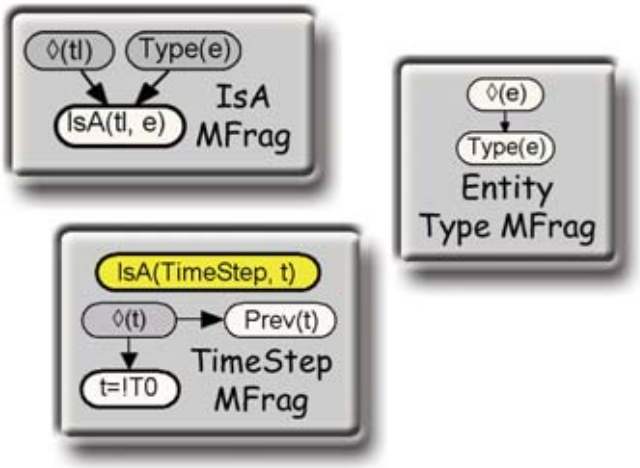

Figura 7.28: MFrags `Entity Type`, `IsA` e `TimeStep`.

Outra funcionalidade que não vem definida na lógica MEBN é o uso de tipos ordenáveis de tal forma que possa ser usada para recursão. Para que isso seja possível, no uso de MEBN puro, o usuário tem que definir uma MFrag como a apresentada na MFrag `TimeStep` definida na Figura 7.28 para toda entidade que fosse utilizada em recursões. Essa pode ser uma tarefa árdua e cansativa para o usuário, além de, possivelmente, levá-lo a cometer erros.

Dessa forma, foi implementada uma solução para garantir que todas entidades sejam tipadas e possibilitar a configuração de um tipo ordenável. Assim, é possível a implementação de recursão. Na Figura 7.30, pode-se verificar que toda entidade tem um tipo, o que garante uma política de tipagem. Já na Figura 7.29 é apresentado como criar uma entidade e seu tipo, assim como definí-la como ordenável na GUI do UnBBayes.

No que tange recursão, o projeto da solução envolve a classe `ObjectEntityInstanceOrdereable`, também na Figura 7.30, que garante que um certo tipo, `TimeStep`, por exemplo, tenha uma ordem total para toda instância criada. Assim, se permite o acesso à instância anterior e posterior.

Com essa solução, a implementação de recursão é basicamente limitada a definir que um determinado tipo é ordenável e definir um nó de entrada como pai do nó residente ao qual ele se refere, mas tendo como argumento um tipo ordenável. Dessa forma, a ferramenta interpreta que o nó de entrada se refere a uma instância anterior da que o nó residente se refere. Para exemplificar, veja a Figura 7.31. Nesse caso, o nó de entrada `DistFromOwn(`*`st,tPrev`*`)` é pai e ao mesmo tempo é entrada do nó residente `DistFromOwn(`*`st,t`*`)`. Logo, a ferramenta

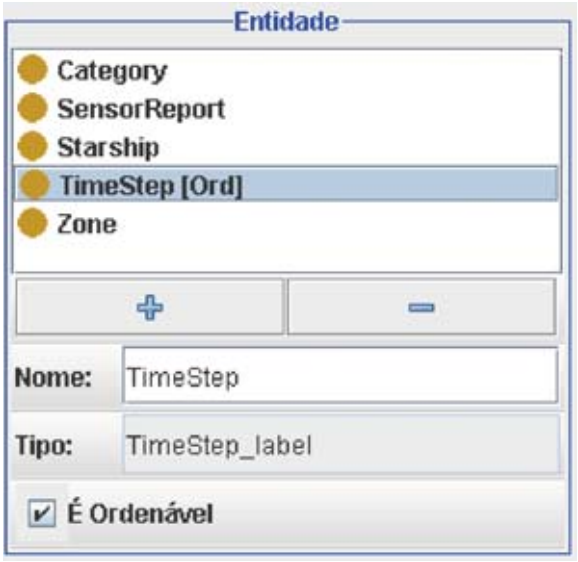

Figura 7.29: Definição de entidade tipada no UnBBayes.

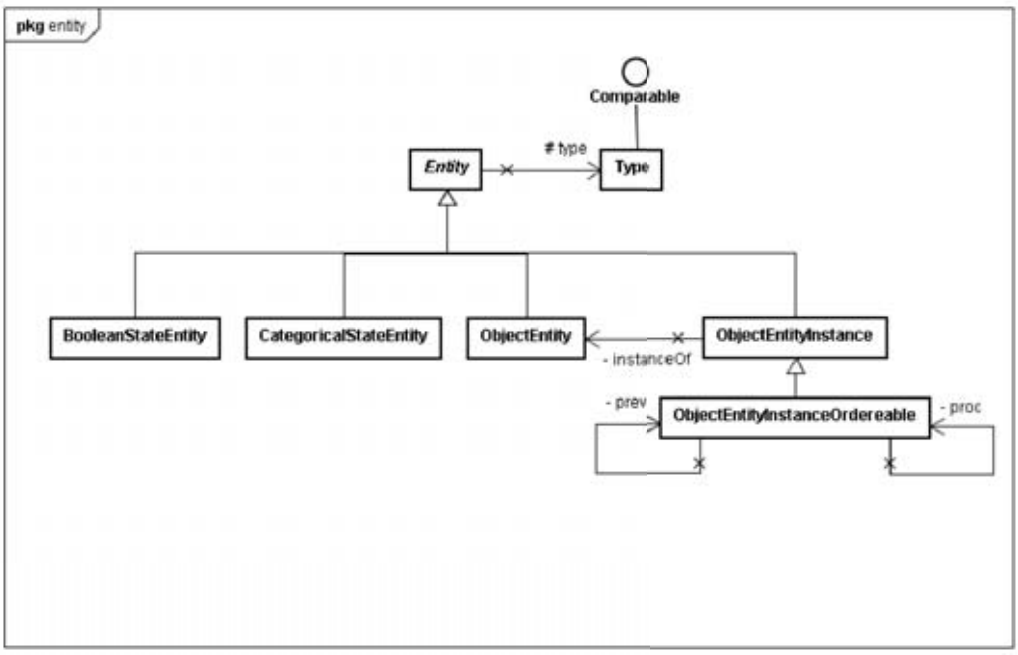

Figura 7.30: Projeto principal de entidade e tipo no UnBBayes.

infere que o nó de entrada se referencia à distância da nave *st* no tempo *tPrev*, ou seja, no tempo imediatamente anterior a *t*. Atualmente, a condição de parada é a primeira instância da lista. Futuramente, pode ser necessário a definição explícita da condição de parada, como por exemplo, as cinco entidades anteriores, ou, como no exemplo dado, nos 5 últimos `TimeStep`.

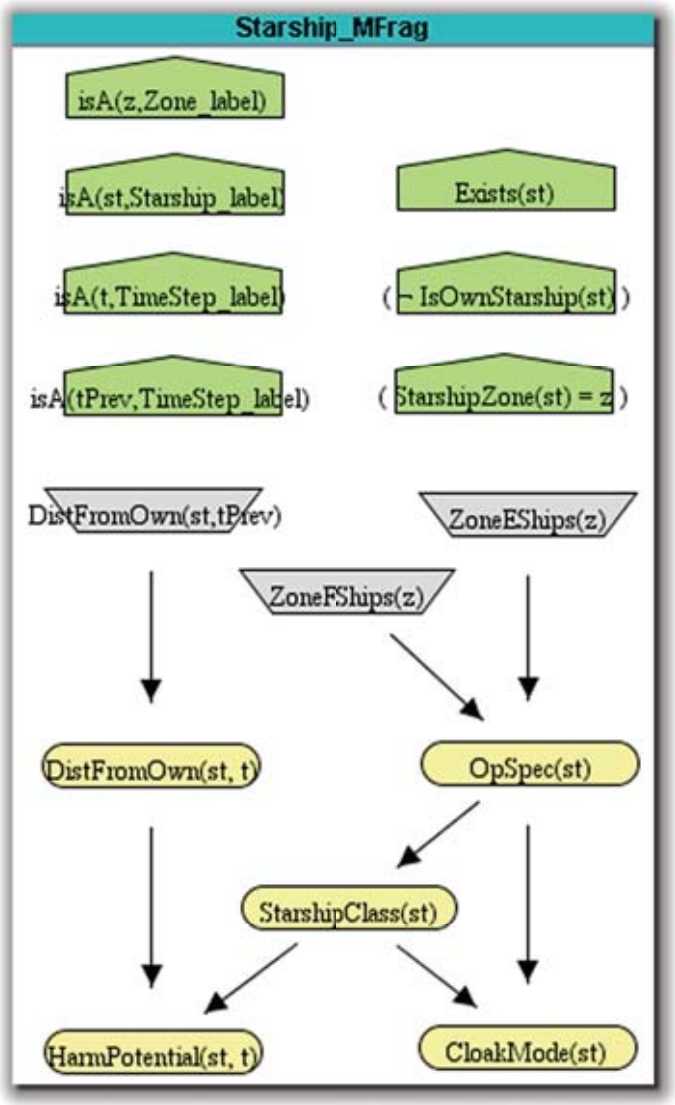

Figura 7.31: Recursão no nó `DistFromOwn` na MFrag `Starship`.

Essa solução está sendo avaliada pela equipe do PR-OWL para verificar se estará na próxima versão do PR-OWL.

7.7 Algoritmo de geração de SSBN

Além de possibilitar a modelagem de uma MTeoria, o UnBBayes permite a geração da SSBN dentro de um contexto específico que é informado pelas evidências. Para isso, é utilizada uma base de conhecimento (KB) para armazenar as informações de domínio. Em [27] é apresentado um algoritmo para geração de SSBN,

no entanto, para uso no UnBBayes, foi proposto um novo algoritmo para geração de SSBN. Quando um questionamento é submetido ao sistema, o primeiro passo é procurar a informação que responde esse questionamento na KB. Uma vez que é determinado que as informações existentes não são suficientes, então a KB é utilizada junto da definição da MTeoria geradora para instanciar a variável aleatória que pode ajudar a responder o questionamento. Ao fazer a geração da rede bayesiana específica, o UnBBayes gera um relatório com todos os passos realizados.

Na implementação atual, o UnBBayes apenas permite a seleção de um nó residente como questionamento. Uma vez selecionada o nó residente e informado os valores específicos de seu argumento, o algoritmo realiza os seguintes passos, onde OV é uma variável ordinária, que é representada em MEBN como um parâmetro de um nó:

i. Para uma dada entrada na forma `NODE(ENTITY-LIST)`, procure pela evidência na KB. Se existir uma evidência para a entrada fornecida, termine. Se não, vá para o próximo passo.

ii. Procure pelo nó residente que tem o nome `NODE` e recupere sua MFrag. Uma vez encontrado `NODE(OV-LIST)`, verifique se o tipo de `ENTITY-LIST` é o mesmo que `OV-LIST`. Se for o mesmo tipo, vá para o próximo passo.

iii. Verifique na KB quais nós de contexto fazem referência para `OV-LIST`, substituindo seus valores por `ENTITY-LIST`. Se pelo menos um nó de contexto for falso, marque a MFrag para utilizar a distribuição padrão. Vá para o próximo passo.

iv. Se o nó de contexto em (iii) não tiver uma solução, crie-o como pai de `NODE`.

v. Para cada pai de `NODE`, vá para o passo (i), substituindo as RV pelas entidades conhecidas (presentes no questionamento ou na KB).

vi. Crie a CPT de `NODE`.

vii. Fim.

Durante a implementação, alguns ajustes em relação a performance foram realizados. Dependendo da complexidade do domínio, o algoritmo pode encontrar um nó de contexto que não pode ser resolvido imediatamente. Isso acontece quando todas variáveis ordinárias no conjunto de pais da variável aleatória residente não aparecem no próprio termo residente. Nesse caso, deve haver um número arbitrário, possivelmente infinito, de instâncias de um pai para cada instância dada do filho. Por exemplo, na MFrag `Starship`, apresentada na Figura 7.27, se a zona onde a espaçonave está localizada é desconhecida, o número de inimigos e amigos (`ZoneEStarships(`z`)` e `ZoneFStarships(`z`))` em cada zona onde possa estar localizada é relevante para a distribuição da variável aleatória `OpSpec(`st`)`. Se o tempo t tiver tempos anteriores, então, mais de uma distância (`DistanceFromOwn(`st`,`$tprev$`)`) deve ser avaliada, fazendo com que distâncias medida em todos os tempos sejam relevantes para a distribuição da variável aleatória `DistFromOwn(`st`,`$tprev$`)` no tempo t. Logo, qualquer número de

instâncias das variáveis aleatórias `ZoneEShips(z)`, `ZoneFShips(z)` e `DistFromOwn(st,tprev)` podem ser relevantes para a distribuição das variáveis aleatórias `OpSpec(st)` e `DistFromOwn(st,tprev)` no tempo t. Nesse caso, a distribuição local para a variável aleatória deve especificar como combinar as influências de todas instâncias relevanes de seus pais.

No entanto, especialmente em fórmulas complexas, isso pode ter um forte impacto na performance do algoritmo. Dessa forma, a solução projetada envolve solicitar ao usuário mais informação. Na atual implementação, se o usuário não apresentar a informação necessária, o algoritmo apenas pára. Outra decisão de projeto foi restringir o uso de memória de tal forma que um possível estouro de memória gera um aviso para o usuário e para o algoritmo. No passo (iii) do algoritmo, uma otimização no projeto do novo algoritmo foi realizada quando comparado com o algoritmo de geração de SSBN apresentado em [27]. Apenas os nós de contextos necessários, baseado nas entidades instanciadas no momento, são avaliados, ao contrário do algoritmo proposto por Laskey que avalia todos nós de contexto da respectiva MFrag.

7.8 Entrada/Saída dos arquivos MEBN

A linguagem formal escolhida para o formato de armazenamento da MTeoria construída no UnBBayes é a PR-OWL (veja seção 3.3). Por estender a OWL, a MTeoria representada nessa linguagem reside em um arquivo texto, em esquema XML, com extensão ".owl".

Para incorporar no UnBBayes o mecanismo de leitura e armazenamento de arquivos escritos em PR-OWL, foi escolhida o uso da API do Protege-OWL, descrita na seção 6.2.

Certas informações peculiarmente tratadas pelo UnBBayes para representar a MEBN não são armazenadas diretamente nos arquivos PR-OWL. Para o armazenamento dessas informações internas do UnBBayes, é utilizado um arquivo UBF, com extensão ".ubf", apresentada na seção 7.8.1.

7.8.1 Arquivo UBF - UnBBayes *File*

O arquivo UBF foi elaborado como um arquivo de projeto do UnBBayes, armazenando informações específicas utilizadas pelo UnBBayes para a representação da MEBN. Como são informações que não fazem parte de uma ontologia PR-OWL, utiliza-se um arquivo separado.

Basicamente, 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. Pode-se também ter comentários no arquivo. Os comentários são iniciados com o símbolo `%` e se estende até o final da linha.

Atualmente na versão 0.02, um UBF está intimamente acoplado a um arquivo PR-OWL. Por suas declarações indicarem parâmetros especiais dos componentes

da MTeoria, e como o próprio conteúdo da MTeoria é armazenado em um arquivo PR-OWL, automaticamente o UBF torna-se acoplado a um arquivo PR-OWL.

No CD há um exemplo de conteúdo de uma UBF. Um UBF completo é dividido em três partes:

1. Cabeçalho contendo meta dados do UBF.

   Version indica a versão do arquivo UBF.

   - `Version=0.02`

   PrOwl indica a URI relativa do arquivo PR-OWL que o arquivo UBF descreve. Seu valor deve estar entre aspas.

   - `PrOwl="StarTrek27.owl"`

2. Declarações da MTeoria: na versão atual, o UnBBayes presume que um único arquivo PR-OWL tenha somente uma MTeoria declarada, entretanto, a estrutura do UBF já permite a declaração de diversas MTeorias em um único arquivo PR-OWL.

   `MTheory` nome da MTeoria tratada pelo UBF.

   - `MTheory=StarshipMTheory`

   `NextMFrag` numeral usado como sufixo para o nome da próxima MFrag que será criada pelo UnBBayes. Este número é tipicamente importante para evitar que os nomes automaticamente gerados choquem com nomes existentes.

   - `NextMFrag=10`

   `NextResident` numeral usado como sufixo para o nome do próximo nó residente que será criado pelo UnBBayes. Este número é tipicamente importante para evitar que os nomes automaticamente gerados choquem com nomes existentes.

   - `NextResident=24`

   `NextInput` numeral usado como sufixo para o nome do próximo nó de entrada que será criado pelo UnBBayes. Este número é tipicamente importante para evitar que os nomes automaticamente gerados choquem com nomes existentes.

   - `NextInput=19`

   `NextContext` numeral usado como sufixo para o nome do próximo nó de contexto que será criado pelo UnBBayes. Este número é tipicamente importante para evitar que os nomes automaticamente gerados choquem com nomes existentes.

   - `NextContext=15`

   `NextEntity` numeral usado como sufixo para o nome da entidade que será criada pelo UnBBayes. Este número é tipicamente importante para evitar que os nomes automaticamente gerados choquem com nomes existentes.

- `NextEntity=1`

3. Lista de declaração de MFrags: declara atributos referentes a conteúdos da MFrag. São diversos blocos de declaração que iniciam com a declaração de nome da MFrag e terminam ou com o início do próximo bloco ou com o fim do arquivo.

   `MFrag` nome da MFrag tratada naquele bloco.

   - `MFrag=DangerToOthersMFrag`

   `NextOrdinaryVar` numeral usado como sufixo para o nome da variável ordinária que será criada pelo UnBBayes para aquela MFrag. Este número é tipicamente importante para evitar que os nomes automaticamente gerados choquem com nomes existentes. Esta declaração está neste escopo, pois uma variável ordinária é declarada localmente na MFrag.

   - `NextOrdinaryVar=4`

   Lista de declaração de nós são diversos blocos que declaram atributos referentes a nós em particular, que podem ser residentes, de entrada, contexto ou variáveis ordinárias.

   `Node` nome do nó.

   - `Node=DangerToOthers`

   `Type` tipo do nó. Seu valor pode ser `OrdinalVar`, `ContextNode`, `GenerativeInputNode` ou `DomainResidentNode`.

   - `Type=DomainResidentNode`

   `Position` dois numerais que representam a coordenada X,Y da posição do nó referente no painel de visualização, separada por vírgulas.

   - `Position=165.0,280.0`

   `Size` largura e altura do nó referente no painel de visualização. Apesar de no momento não fazer sentido para uma variável ordinária, futuramente pode ser usada para especificar um nó de contexto `IsA`.

   - `Size=100,20`

Aquelas declarações no arquivo UBF que não forem identificadas na especificação ou que não tenham representantes no arquivo PR-OWL serão simplesmente ignoradas. Isso permitiria que uma classe que leia uma versão antiga do UBF (ou PR-OWL) possa ler uma versão recente sem maiores danos.

7.8.2 Implementação no UnBBayes

Para a implementação do suporte a IO de MEBN foi utilizado o modo de arquivos OWL. 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 7.32 mostra o diagrama das classes que compõem o suporte a abertura e armazenamento de MTeorias. Logo abaixo será apresentada a especificação dos componentes da implementação.

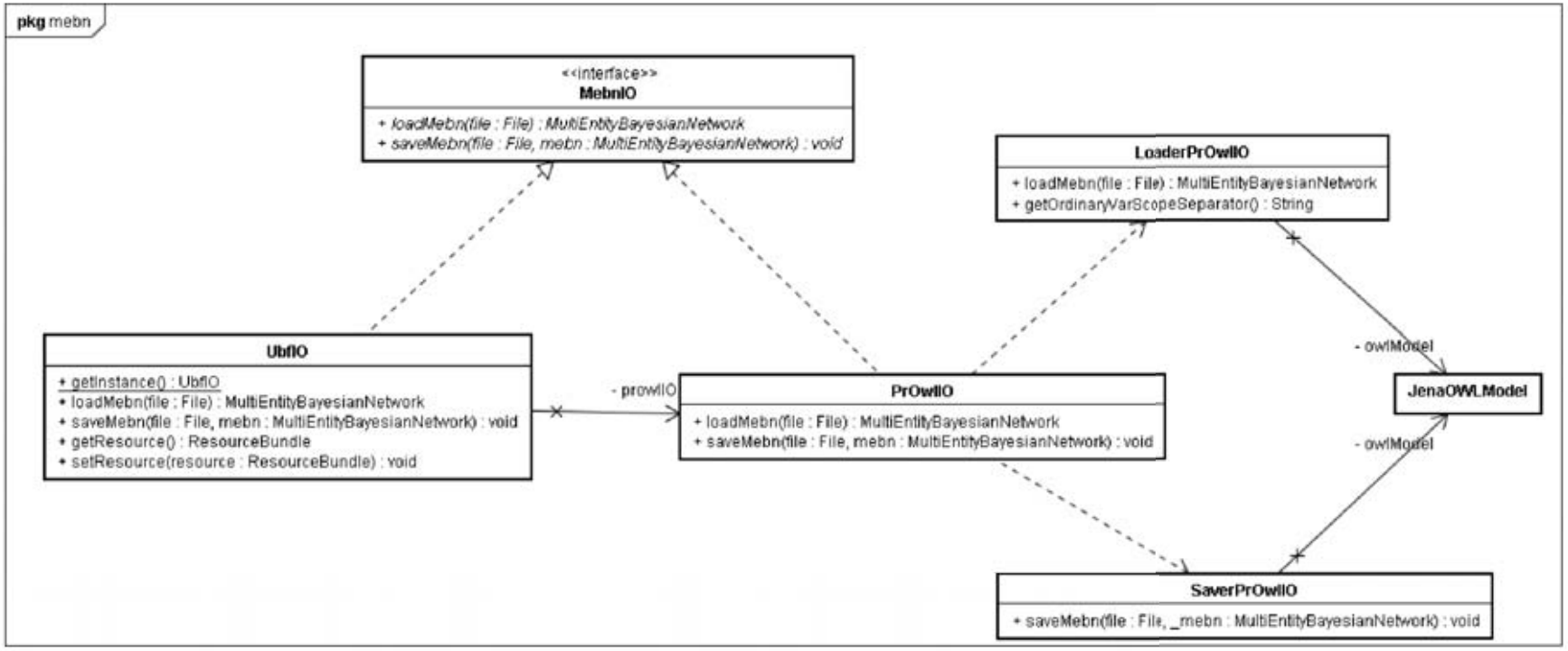

Figura 7.32: Modelagem de IO de MEBN.

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 regionalização.

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 MTeoria descrita em PR-OWL. Utiliza a API Protege-OWL para esse fim.

Classe `SaverPrOwlIO` classe que concentra rotinas de armazenamento da MTeoria 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.

Capítulo 8

Estudo de Caso: *Star Trek*

Para ilustrar um caso de uso de ontologia probabilística em PR-OWL no UnB-Bayes, foi feita uma migração da ontologia *Star Trek* [13] para o editor visual.

As Figuras 8.1 e 8.2 detalham a MTeoria utilizada por Costa e Laskey em [12] e [13]. De uma maneira simplificada, o principal objetivo dessa MTeoria é modelar o problema de detectar espaçonaves *Romulan* (consideradas comos hostis para Federação Unida dos Planetas) a analisar o nível de perigo que representam para a própria espaçonave, a *Enterprise*. Todas outras espaçonaves são consideradas amigáveis ou neutras. A detecção de espaçonaves é realizada através de um conjunto de sensores, que podem detectar e discriminar com uma acurácia de 95%. No entanto, espaçonaves *Romulan* podem estar em "*cloack mode*", que as tornam invisíveis para os sensores da *Enterprise*. Até mesmo para as tecnologias mais recentes de sensores, a única dica de que há uma espaçonave próxima em *cloack mode* é um leve distúrbio magnético causado pela enorme quantidade de energia exigida para permanecer nesse estado. A *Enterprise* possui um sensor de distúrbio magnético, mas é muito difícil distinguir o distúrbio magnético natural de um causado pela espaçonave no *cloack mode*.

Existem quatro entidades básicas nesse modelo: *Starship*, *Zone*, *TimeStep* e *SensorReport*.

De acordo com a enciclopédia Trecknológica[1], *Starship* é a designação para uma espaçonave com uma dobra espacial. Ela tipicamente possui mais de um convés e tem compartimentos separados, como ponte, oficina, efermaria, etc. Nesse modelo, essa palavra é utilizada para designar qualquer espaçonave.

Uma `Zone`, ou zona, pode ser `Deep Space`, `Planetary System` ou `Boundary of a Black Hole`. É assumido que `OwnStarship`, ou `Enterprise`, quando em operação, tem 80% de chance de estar na zona `Deep Space Zone`, 15% de estar na `Planetary System` e 5% de estar na `Boundary of a Black Hole`. Nesse modelo, fronteiras de buracos negros são as zonas preferidas para armadilhas preparadas por espaçonaves que usam *cloak device*, já que o alto distúrbio magnético gerado nessas zonas dificulta até mesmo para os sensores mais avançados a distinção dos distúrbios causados pela zona dos causados por espaçonaves em *cloack mode*.

`TimeStep` é uma entidade especial utilizada para modelar nós dinâmicos e há vários nesse domínio.

[1] http://www.ex-astris-scientia.org/treknology1.htm

Finalmente, os relatórios gerados pelos sensores da `Enterprise` são indivíduos da entidade `SensorReport`.

De forma a possibilitar uma comparação com a MTeoria modelada no UnBBayes com a proposta em [13], esta é apresentada novamente na Figura 8.1, juntamente com a MTeoria modelada no UnBBayes na Figura 8.2. No UnBBayes, as variáveis ordinárias são declaradas com nós de contexto `isA`.

As CPT para os nós residentes apresentados nas seções seguintes e o arquivo .ubf e .owl que representam o modelo *Star Trek* editado no UnBBayes estão disponíveis no CD.

As próximas seções apresentam as MFrags editadas no UnBBayes. As figuras das MFrags estarão seguidas por uma breve descrição de seus componentes.

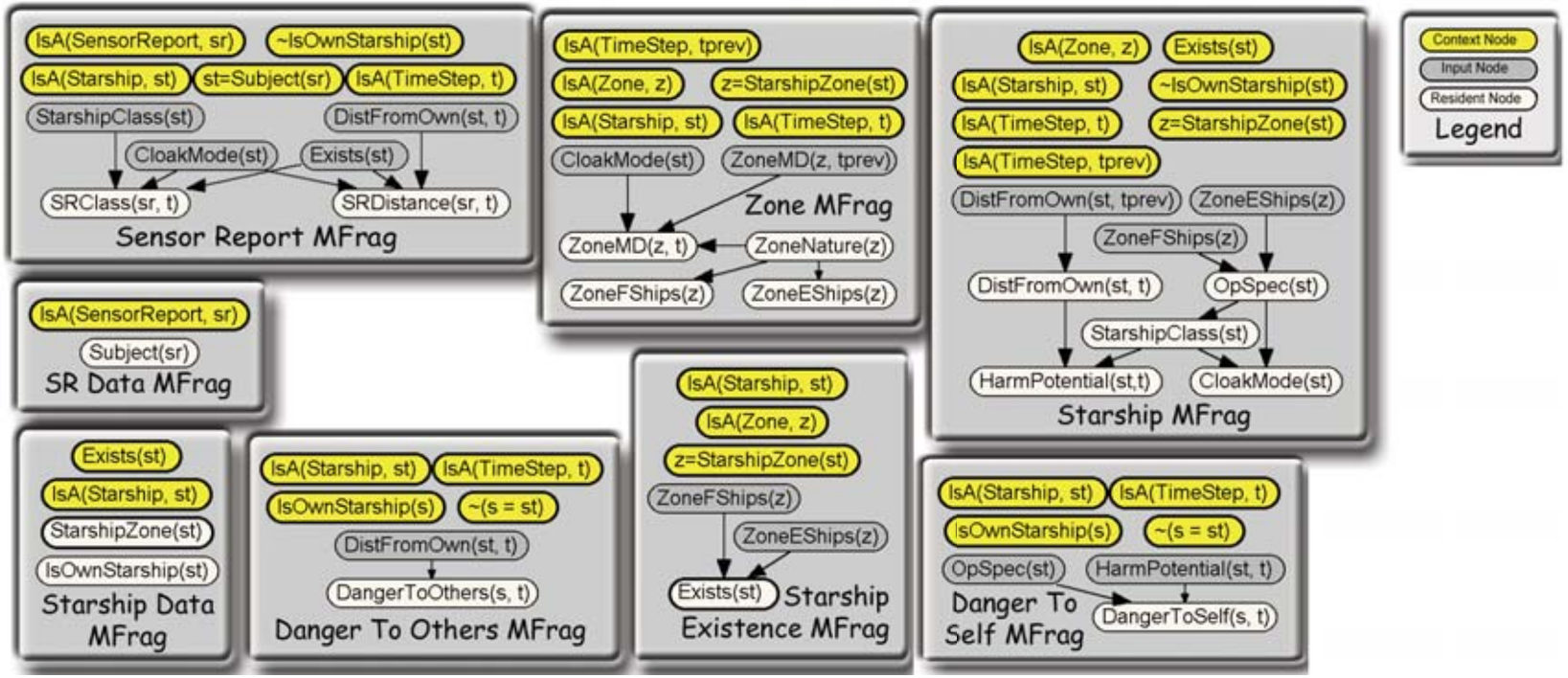

Figura 8.1: MTeoria *Star Trek* proposta em [13] adaptada.

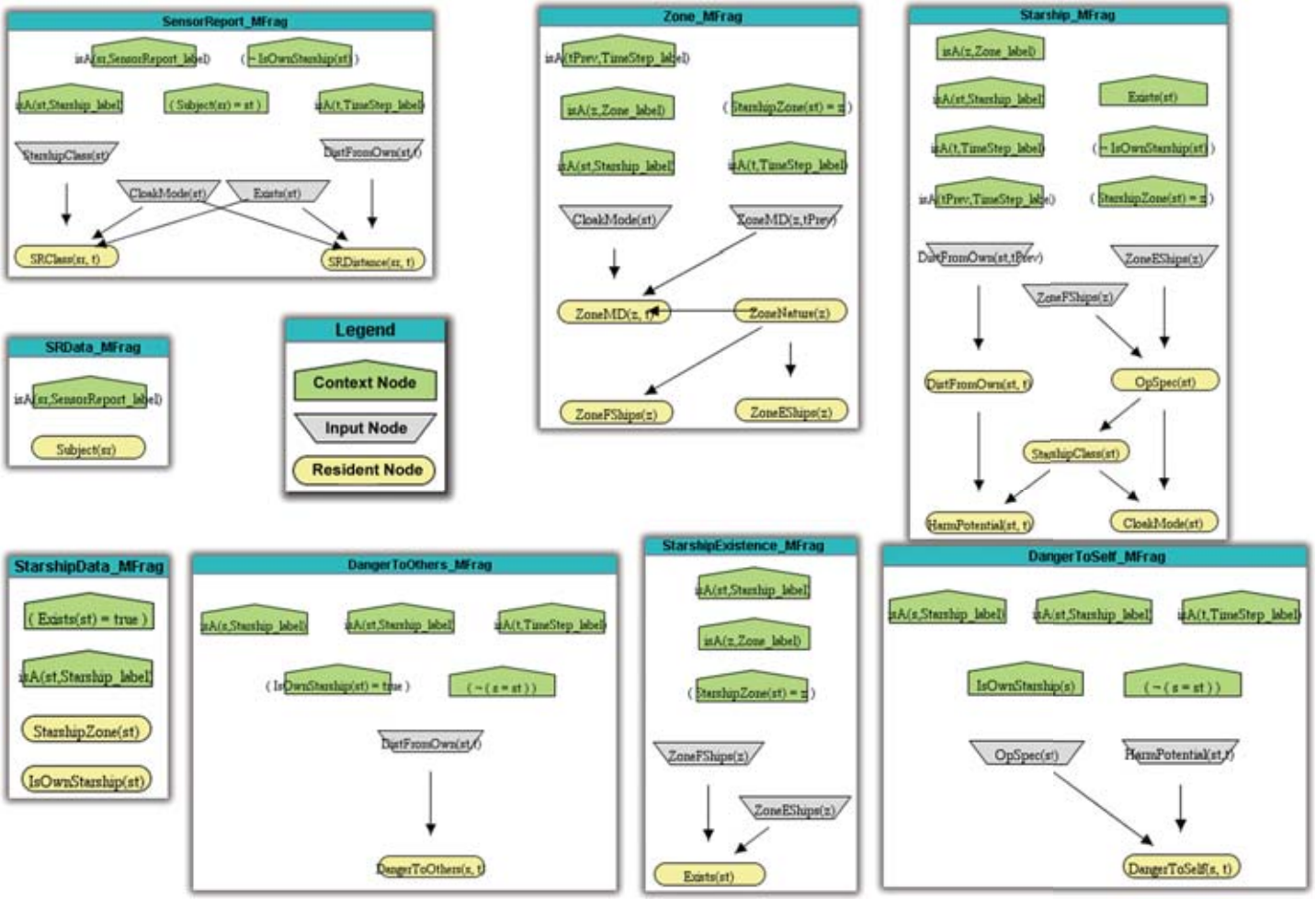

Figura 8.2: MTeoria do *Star Trek* editada no UnBBayes.

8.1 MFrag `DangerToOthers`

A Figura 8.3 mostra a MFrag `DangerToOthers` modelada com a GUI do UnBBayes.

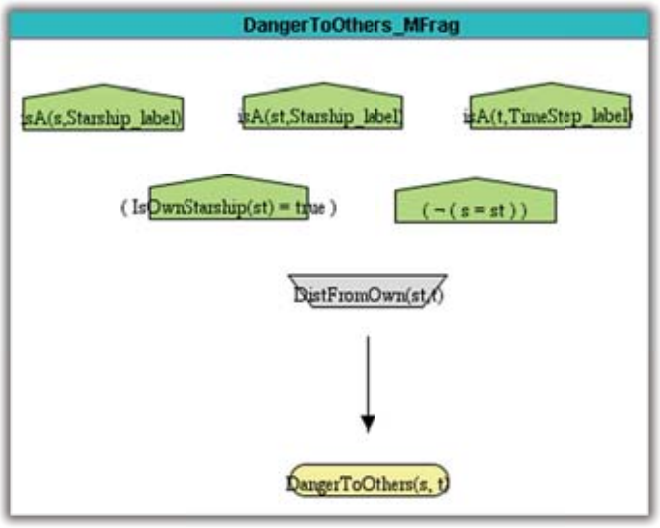

Figura 8.3: MFrag `DangerToOthers` modelada no UnBBayes.

- Variáveis ordinárias:

  `t` tipo `TimeStep`. Utilizado para representar o tempo atual.

  `s` tipo `Starship`. É a própria espaçonave testada.

  `st` tipo `Starship`. É a espaçonave que a distância com o `Starship` ***s*** é testada.

- Nós de entrada:

  `DistFromOwn` representa o nó residente de mesmo nome, presente na MFrag `Starship`.

- Nós residentes:

  `DangerToOthers` representa a capacidade da espaçonave ***s*** de inferir sobre o perigo de outras espaçonaves ***st***.

- Nós de contexto:

  `IsOwnStarship` indica que a espaçonave ***s*** é a própria espaçonave.

  `(not(s = st))` garante que as duas espaçonaves ***s*** e ***st*** são distintas.

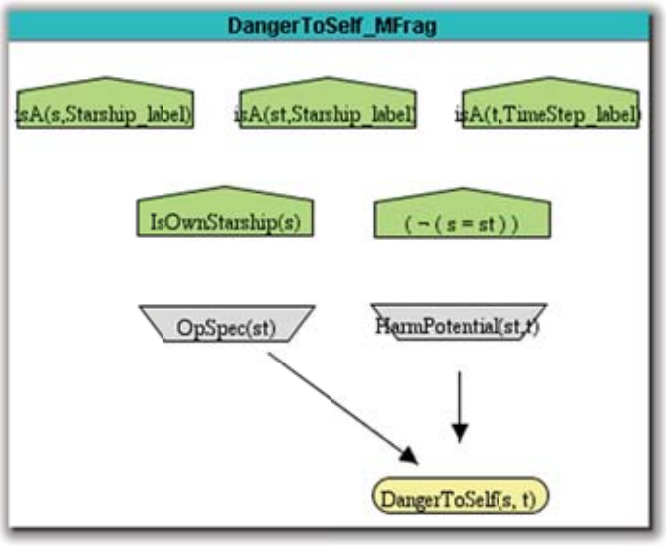

Figura 8.4: MFrag `DangerToSelf` modelada no UnBBayes.

8.2 MFrag `DangerToSelf`

A Figura 8.4 mostra a MFrag `DangerToSelf` modelada com a GUI do UnBBayes.

- Variáveis ordinárias:

  `t` tipo `TimeStep`. Utilizado para representar o tempo atual.

  `s` tipo `Starship`. É a própria espaçonave testada.

  `st` tipo `Starship`. É a espaçonave que representa um perigo potencial para a própria espaçonave.

- Nós de entrada:

  `HarmPotential` representa o nó residente de mesmo nome, presente na MFrag `Starship`.

  `OpSpec` representa o nó residente de mesmo nome, presente na MFrag `Starship`.

- Nós residentes:

  `DangerToSelf` representa o perigo que a própria espaçonave *`s`* está exposta no tempo *`t`*.

- Nós de contexto:

  `IsOwnStarship` indica que a espaçonave *`s`* é a própria espaçonave.

  `(not(s = st))` garante que as duas espaçonaves *`s`* e *`st`* são distintas.

8.3 MFrag `SensorReport`

A Figura 8.5 mostra a MFrag `SensorReport` modelada com a GUI do UnBBayes.

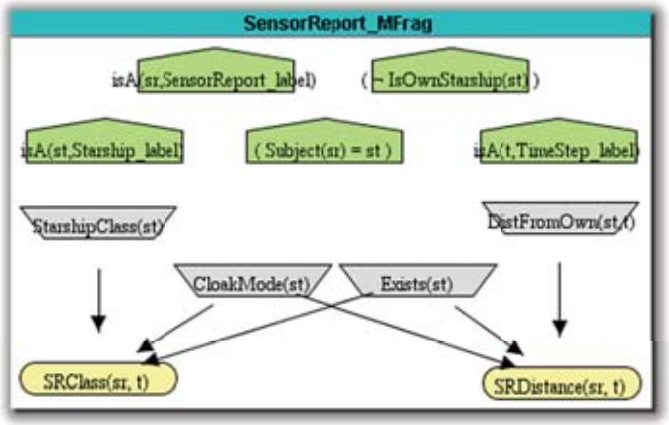

Figura 8.5: MFrag `SensorReport` modelada no UnBBayes.

- Variáveis ordinárias:

  `t` tipo `TimeStep`. Utilizado para representar o tempo atual.

  `sr` tipo `SensorReport`. Indica o relatório obtido pelo sensor.

  `st` tipo `Starship`. É a espaçonave que a distância com a espaçonave ***`s`*** é testada.

- Nós de entrada:

  `Exists` representa o nó residente de mesmo nome, presente na MFrag `StarshipExistance`.

  `CloakMode` representa o nó residente de mesmo nome, presente na MFrag `Starship`.

  `StarshipClass` representa o nó residente de mesmo nome, presente na MFrag `Starship`.

  `DistFromOwn` representa o nó residente de mesmo nome, presente na MFrag `Starship`.

- Nós residentes:

  `SRClass` representa que tipo de classe de espaçonave foi reportada por ***`sr`*** em um tempo ***`t`***.

  `SRDistance` representa a distância reportada entre a própria espaçonave e a espaçonave detectada pelo relatório ***`sr`*** no tempo ***`t`***.

- Nós de contexto:

  `not IsOwnStarship` indica que a espaçonave s não é a própria espaçonave.

  `(Subject(sr) = st)` garante que a espaçonave ***`st`*** é a reportada pelo sensor em ***`sr`***.

8.4 MFrag SRData

A Figura 8.6 mostra a MFrag SRData modelada com a GUI do UnBBayes.

Figura 8.6: MFrag SRData modelada no UnBBayes.

- Variáveis ordinárias:

  sr tipo SensorReport. Indica o relatório obtido pelo sensor.

- Nós residentes:

  Subject tem como seu valor possível todas as entidades que podem ser objeto do relatório *sr* do sensor. No caso, qualquer instância da entidade Starship como o conjunto dos seus valores possíveis, representando todas as espaçonaves do domínio. Dessa forma, a distribuição de probabilidades é uniforme.

8.5 MFrag Starship

A Figura 8.7 mostra a MFrag Starship modelada com a GUI do UnBBayes.

- Variáveis ordinárias:

  t tipo TimeStep. Utilizado para representar o tempo atual. É uma entidade ordenável que será utilizada nessa MFrag para recursividade temporal.

  tPrev tipo TimeStep. Indica o tempo imediatamente anterior a *t*.

  st tipo Starship. É a espaçonave que a distância com a espaçonave *s* é testada.

  z tipo Zone. É a zona em que *st* está contida.

- Nós de entrada:

  DistFromOwn representa o nó residente de mesmo nome, presente na mesma MFrag. É um caso de recursividade.

  ZoneEShip representa o nó residente de mesmo nome, presente na MFrag Zone.

  ZoneFShip representa o nó residente de mesmo nome, presente na MFrag Zone.

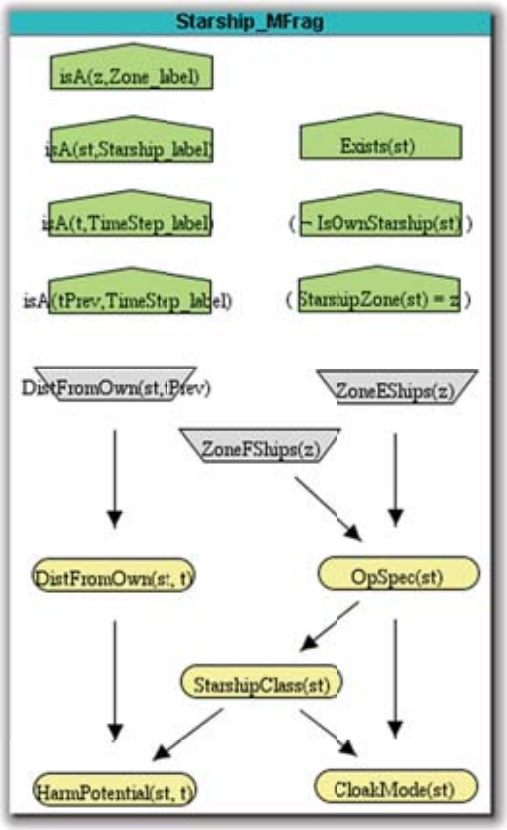

Figura 8.7: MFrag `Starship` modelada no UnBBayes.

- Nós residentes:

  `DistFromOwn` representa a distância entre a própria espaçonave e a espaçonave st. Essa distância é medida de acordo com o alcance das armas disponíveis. Seu valor pode ser: OutOfRange, TorpedoRange, Phaser2Range, Phaser1Range, PulseCanonRange e absurd.

  `HarmPotential` representa o potencial de ***st*** em danificar a própria espaçonave no tempo ***t***. Não representa a intenção, e sim a sua habilidade.

  `OpSpec` representa a espécie ou raça que está pilotando a espaçonave ***st***. Pode ser `Cardassian`, `Friend`, `Klingon`, `Romulan`, `Unknown` e `absurd`.

  `StarshipClass` representa a classe da espaçonave ***st***. Pode ser `WarBird`, `Cruiser`, `Explorer`, `Frigate`, `Freighter` e `absurd`.

  `CloakMode` variável booleana que indica se a espaçonave em questão está em modo invisível.

- Nós de contexto:

  `not IsOwnStarship` indica que a espaçonave ***st*** não é a própria espaçonave.

  `Exists` garante que a espaçonave ***st*** está presente no sistema.

  `(StarshipZone(st) = z)` indica que a zona ***z*** é onde a espaçonave ***st*** está contida.

8.6 MFrag `StarshipData`

A Figura 8.8 mostra a MFrag `StarshipData` modelada com a GUI do UnBBayes.

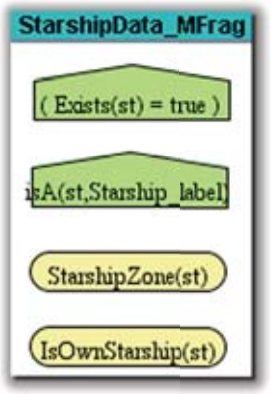

Figura 8.8: MFrag `StarshipData` modelada no UnBBayes.

- Variáveis ordinárias:

  `st` tipo `Starship`. É a espaçonave em questão.

- Nós residentes:

`StarshipZone` indica qual é a zona que a espaçonave *st* se situa. No caso, qualquer instância da entidade `Zone` faz parte do conjunto dos seus valores possíveis, representando todas as espaçonaves do domínio. Dessa forma, a distribuição de probabilidades é uniforme.

`IsOwnStarship` variável booleana que indica se *st* é a própria espaçonave. A princípio, seu condicionante deveria ser um nó de identidade [12], mas como o UnBBayes ainda não implementa esses nós, `IsOwnStarship` foi mantido sem pai.

- Nós de contexto:

`Exists` garante que a espaçonave *st* está presente no sistema.

8.7 MFrag `StarshipExistence`

A Figura 8.9 mostra a MFrag `StarshipExistence` modelada com a GUI do UnBBayes.

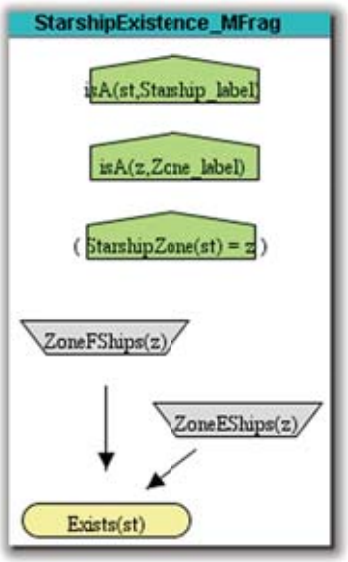

Figura 8.9: MFrag `StarshipExistence` modelada no UnBBayes.

- Variáveis ordinárias:

`z` tipo `Zone`. É a zona onde se encontra a espaçonave *st* no momento *t*.

`st` tipo `Starship`. É a espaçonave cujo `CloakMode` é testado.

- Nós de entrada:

`ZoneFShips` representa o nó residente de mesmo nome, presente na MFrag `Zone`.

`ZoneEShips` representa o nó residente de mesmo nome, presente na MFrag `Zone`.

- Nós residentes:

  `Exists` representa se uma espaçonave está presente em uma determinada zona dado as chances de existência de inimigos (ou amigos) na zona. Possui valor booleano.

- Nós de contexto:

  `(StarshipZone(st) = z)` indica que a zona ***z*** é a zona onde a espaçonave ***st*** se encontra.

8.8 MFrag `Zone`

A Figura 8.10 mostra a MFrag `Zone` modelada com a GUI do UnBBayes.

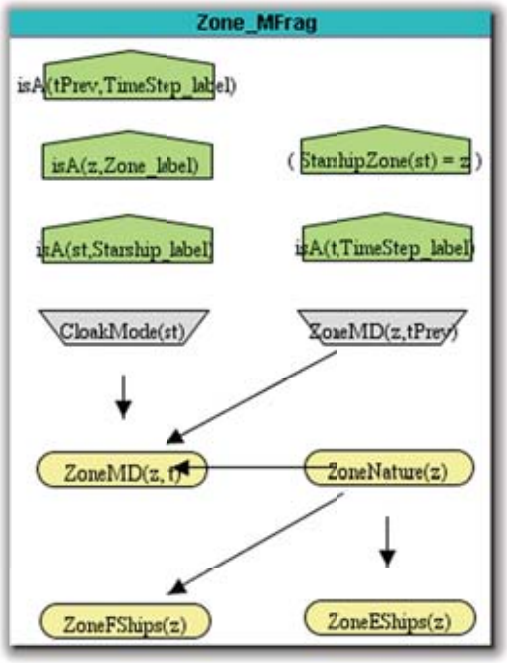

Figura 8.10: MFrag `Zone` modelada no UnBBayes.

- Variáveis ordinárias:

  `t` tipo `TimeStep`. Utilizado para representar o tempo atual. É uma entidade ordenável que será utilizada nessa MFrag para recursividade temporal.

  `tPrev` do tipo `TimeStep`. É o tempo imediatamente anterior ao ***t***.

  `z` tipo `Zone`. É a zona onde se encontra a espaçonave ***st*** no momento ***t***.

  `st` tipo `Starship`. É a espaçonave cujo `CloakMode` é testado.

- Nós de entrada:

  `ZoneMD` representa o nó residente de mesmo nome, presente na mesma MFrag. É um caso de recursividade.

`CloakMode` representa o nó residente de mesmo nome, presente na MFrag `Starship`.

- Nós residentes:

  `ZoneNature` é um nó sem condicionante. Pode ter o valor `BlackHoleBoundary`, `DeepSpace` ou `PlanetarySystem`.

  `ZoneMD` representa o valor de distúrbio magnética na zona ***z*** no tempo ***t***.

  `ZoneFShips` relaciona a possibilidade de uma zona conter naves amigas ao redor.

  `ZoneEShips` relaciona a possibilidade de uma zona conter naves inimigas ao redor.

- Nós de contexto:

  `(StarshipZone(st) = z)` indica que a zona ***z*** é a zona onde a espaçonave ***st*** se encontra.

8.9 *Query* HarmPotential(!ST4,!T3)

Ao fazer a geração da rede bayesiana específica, o UnBBayes gera um relatório com todos os passos realizados. Para exemplificar e deixar mais claro as informações contidas nesse relatório, este será intercalado com a rede bayesiana existente em cada alteração realizada à BN que está sendo gerada. O questionamento utilizado foi o `HarmPotential(!ST4,!T3)`. O modelo e a KB utilizados, assim como, o relatório se encontram no CD.

Uma vez finalizado o algoritmo de geração de SSBN, o UnBBayes escaminha a rede gerada para a tela de inferência em BN como pode ser visto na Figura 8.11. O algoritmo de inferência em BN foi implementado na versão anterior do UnBBayes conforme apresentado em [26].

O resultado do questionamento feito indica que a probabilidade de `HarmPotential(!ST4,!T3)` ser `false` é de 91,46%, de ser `true` é de 8,54% e de ser `absurd` é de 0,00%.

No entanto, dificilmente esse resultado será sastisfatório. Na verdade, o que se busca é a BN específica para que seja possível realizar inferências probabilísticas para o contexto existente. Uma nova evidência poderia ser encontrada com a informação de que a `StarshipClass(!ST4)` é `WarBird`. Dessa forma, essa informação deve ser alimentada no modelo como apresentado na Figura 8.12 e propagada para o resto das variáveis.

Uma vez feita a propagação o usuário pode avaliar novamente a informação disponível para a variável de interesse. Com a nova evidência, a variável `HarmPotential(!ST4,!T3)` apresenta agora uma probabilidade de 16,31% para `false`, de 83,69% para `true` e de 0,00% para `absurd`, como apresentado na Figura 8.13. Uma variação significativa que certamente será útil para o usuário.

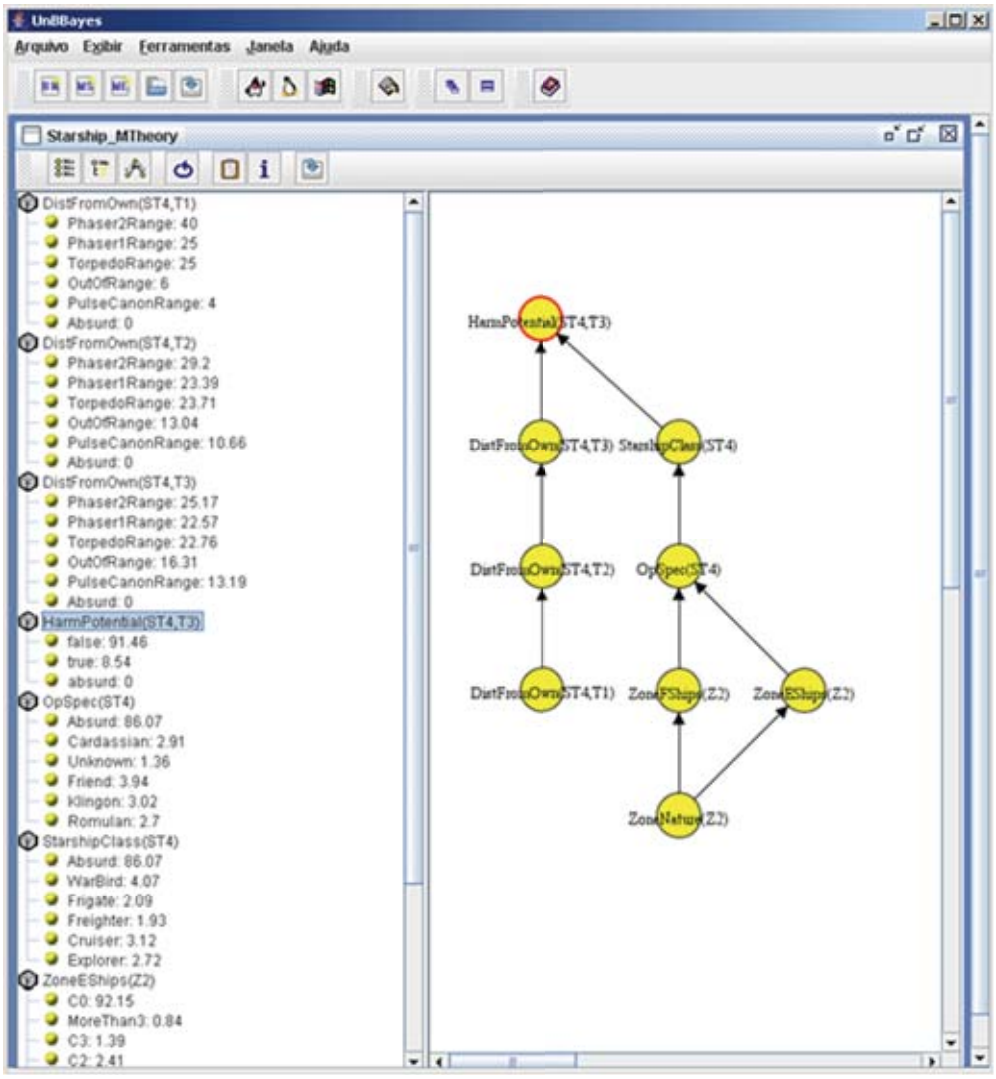

Figura 8.11: SSBN gerada como resultado da *query* `HarmPotential(!ST4,!T3)`.

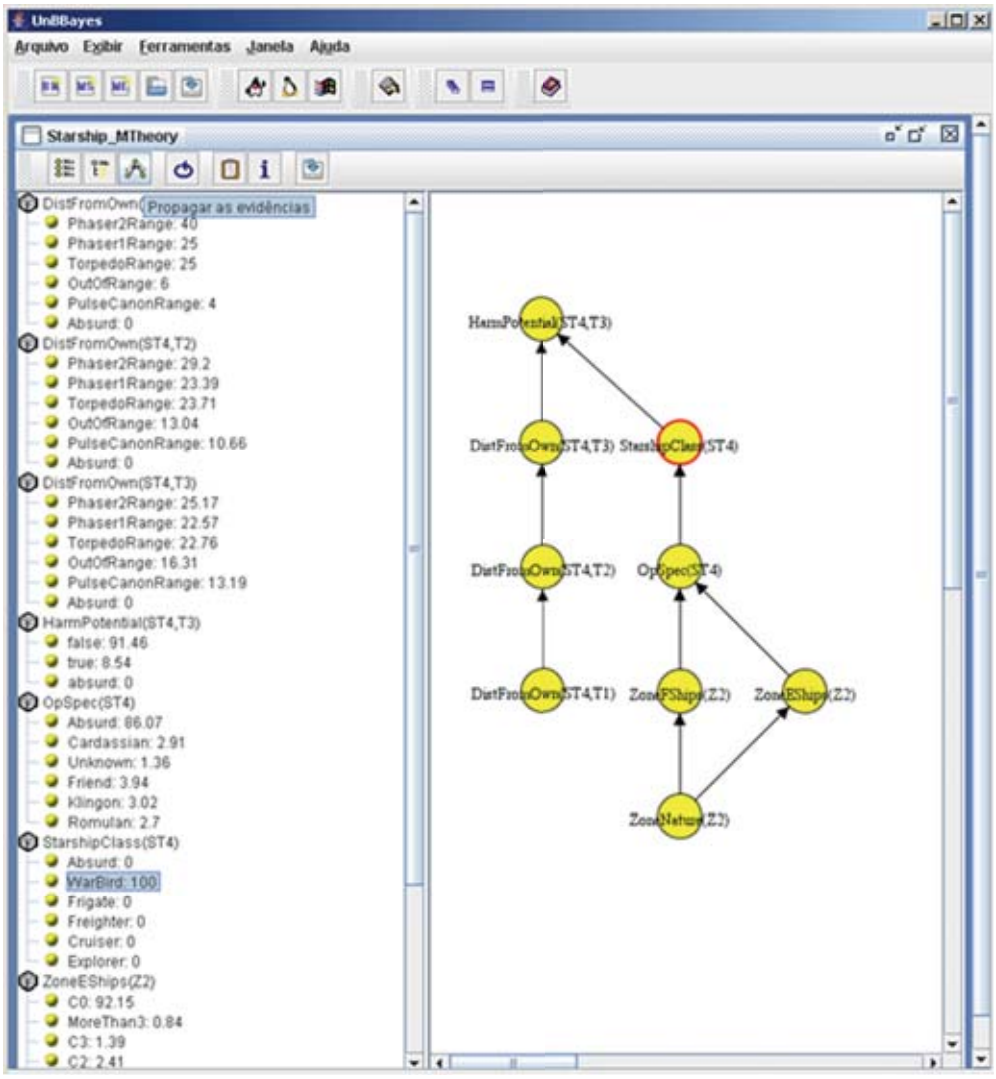

Figura 8.12: Entrando com evidência `StarshipClass(!ST4) = WarBird`.

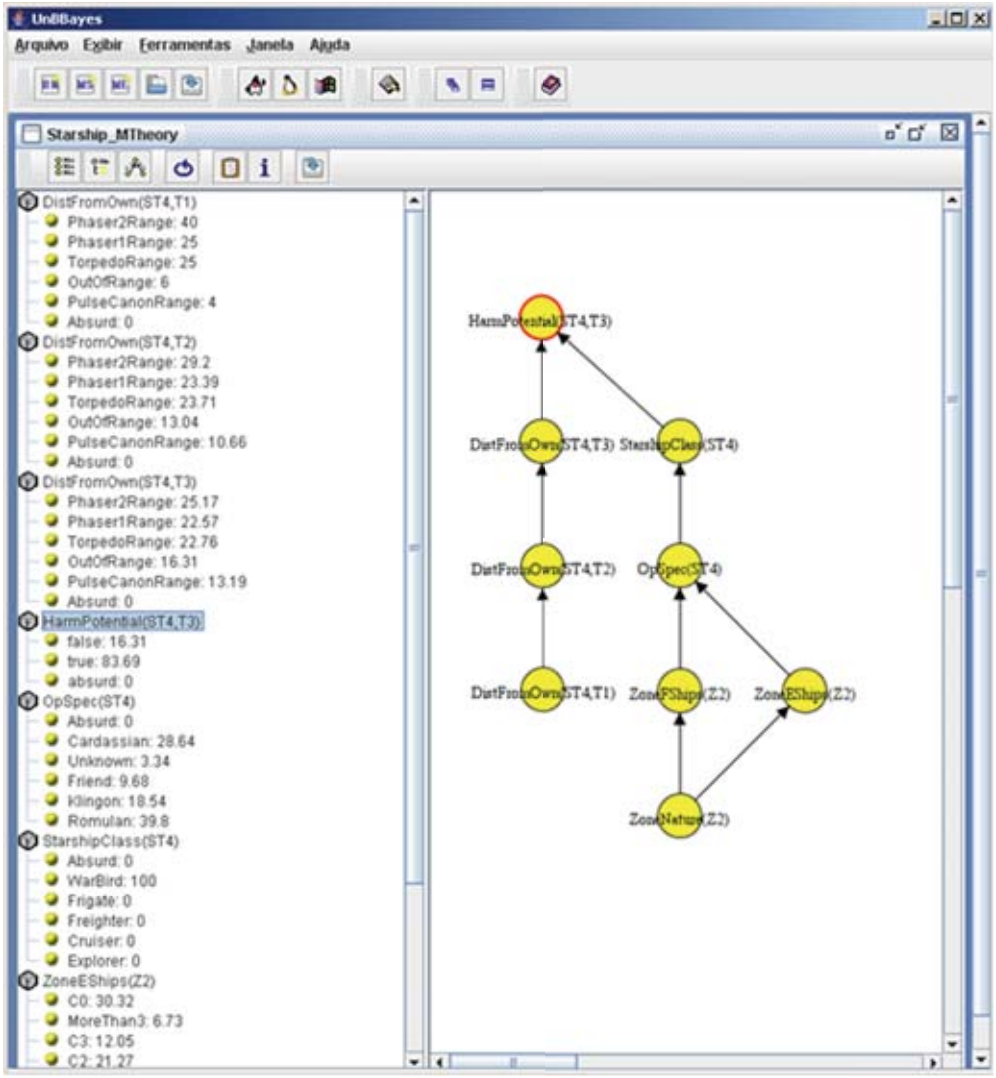

Figura 8.13: Resultado da propagação da evidência `StarshipClass(!ST4) = WarBird`.

Capítulo 9

Conclusão

O grupo URW3-XG foi criado pelo consórcio W3C com o objetivo de estudar formalismos para representar incerteza na Web semântica e raciocinar com base em ontologias. Essa pesquisa constitui uma contribuição para os trabalhos desse grupo visto que foca a primeira implementação mundial de uma ferramenta aberta capaz de representar e raciocinar com ontologias probabilísticas, baseadas na linguagem PR-OWL e na lógica MEBN. A ferramenta implementada é o UnBBayes, versão MEBN, disponível via licença GNU GPL.

A descrição da lógica MEBN utilizada na implementação do UnBBayes é a apresentada em [27]. Já a descrição da linguagem PR-OWL utilizada é a apresentada em [12], com as atualizações disponibilizadas no site www.pr-owl.org.

As principais contribuições científicas obtidas com a implementação do UnBBayes-MEBN são:

1. extensão da linguagem PR-OWL com a inclusão da propriedade de exclusividade global (seção 7.1.2), possibilidade de definir uma entidade como sendo um estado possível de um nó (seção 7.1.2) e facilidade de recursão embutida (seção 7.6).

2. proposição de um novo algoritmo para geração de SSBN (seção 7.7 e [7, 8, 5]).

3. proposição de uma gramática BNF para geração de pseudocódigo para construção de CPT de forma dinâmica (seção 7.2 e [6]).

As principais contribuições tecnológicas resultantes dessa pesquisa são:

1. implementação do algoritmo para geração de SSBN proposto.

2. primeira implementação mundial de uma ferramenta PR-OWL/MEBN. Essa ferramenta permite salvar modelos baseados em ontologias probabilísticas. O formato do arquivo gravado é denominado PR-OWL o qual é compatível com o formato OWL. O UnBBayes-MEBN também permite realizar inferências probabilísticas com o modelo PR-OWL/MEBN criado. O *software* implementado é aberto, gratuíto, desenvolvido em Java com utilização de padrões de projeto e suporte para internalicionalização (atualmente em inglês e português). O UnBBayes-MEBN está disponível no site do Source Forge (www.sourceforge.net/projects/UnBBayes).

3. desenvolvimento de uma GUI para permitir a construção de modelos baseados em PR-OWL e MEBN de forma visual (Capítulo 7 e [6]), minimizando o esforço requerido para o modelador, pois construir MFrags e todos os seus elementos em uma ontologia probabilística, com um editor de textos ou mesmo um editor de ontologias como é o caso do *software* Protégé [19], é um processo cansativo e sujeito a erros, exigindo um conhecimento profundo da estrutura de dados do PR-OWL.

4. implementação de um compilador das regras de produção da gramática BNF para a construção de CPT de forma dinâmica (seção 7.2).

5. proposição e implementação do formato .ubf (seção 7.8.1) para representar propriedades gráficas de modelos PR-OWL/MEBN, variáveis de trabalho e representação de recursão.

6. padronização da representação gráfica dos nós residentes, de entrada e de contexto (seção 7.3 e [15]).

9.1 Limitações e trabalho futuro

O UnBBayes-MEBN disponibilizado é uma versão alfa visto que não foi possível testá-la de forma exaustiva com a modelagem de um domínio real. Os testes realizados foram baseados no domínio fictício *Star Trek* [12].

Atualmente é possível entrar com diversas evidências mas a inferência é apenas sobre um nó de interesse.

No que se refere ao mecanismo de recursão, a implementação atual se limita a informar que um tipo é ordenável e que um nó de entrada usa esse tipo. Dessa forma, o algoritmo, para gerar a SSBN, faz a recursão até chegar ao menor elemento (segundo a ordenação) desse tipo ordenável. Uma extensão interessante seria permitir informar a condição de parada para esse tipo ou até mesmo o número de passos que se deseja caminhar na recursão. Por exemplo, o nó `DistFromOwn(st,tPrev)` pode ser relevante apenas para os 5 tempos anteriores, logo, se `DistFromOwn(st,t)` possui o parâmetro `t` como `!T10`, a condição de parada para `DistFromOwn(st,tPrev)` seria quando `tPrev` fosse `!T5`. Na implementação corrente, a recursão pararia quando `tPrev` fosse `!T0`.

Em relação à GUI, o desenho dos nós poderia ser melhorado da seguinte forma:

- o tamanho dos nós ser individual e aumentar automaticamente de acordo com o rótulo que aparece dentro dele;

- fazer com que as setas saiam do contorno do nó de origem e cheguem no contorno do nó de destino. Na implementação atual, isso não ocorre quando o nó não é um círculo;

- implementar uma visualização de toda MTeoria, similar ao apresentado na Figura 8.2.

Uma funcionalidade importante para tornar possível a reutilização de MFrags é a de se salvar uma MFrag sem necessariamente salvar toda a MTeoria. Na hora de salvar uma MFrag, seriam apresentadas duas possibilidades:

1. salvar apenas a MFrag corrente, podendo causar inconsistências caso o nó de entrada não exista como residente em outra MFrag da MTeoria onde aquela MFrag for carregada;

2. salvar a MFrag corrente e todas as dependentes direta ou indiretamente. As MFrags referenciadas pelos nós de entrada da MFrag corrente são dependentes direto. Da mesma forma, as MFrags referenciadas devem salvar todas MFrags, ainda não salvas, que seus nós de entrada fazem referência (dependente indireto). Isso garante uma consistência quando essas MFrags forem carregadas em outra MTeoria.

No entanto, em ambos os casos, poderia acontecer de existir MFrags ou até mesmo nós com mesmo nome na MTeoria carregada e esse conflito teria que ser resolvido pelo usuário.

Uma melhoria útil para ser feita, no atual algoritmo de geração da SSBN implementado, é a possibilidade do usuário indicar mais de um nó como de interesse para o algoritmo, ou seja, possibilitar que o questionamento seja um conjunto de nós de interesse e não apenas um nó.

Por fim, funcionalidades previstas em [12] e [27] como função identidade, composição de funções, árvore de tipos, polimorfismo e variáveis exemplar devem ser consideradas para implementações futuras.

Referências

[1] T. Berners-Lee, D. Connolly, S. Hawke, I. Herman, E. Prudhommeaux e R. Swick. W3C Semantic Web Activity. www.w3.org/2001/sw/, 2001.

[2] T. Berners-Lee, J. Hendler e O. Lassila. The Semantic Web. *Scientific American Magazine*, 2001.

[3] F. Bobillo, P.C.G. Costa, C. d'Amato, N. Fanizzi, F. Fung, T. Lukasiewicz, T. Martin, M. Nickles, Y. Peng, M. Pool, P. Smrz e P. Vojtas, editores. *Proceedings of the Third ISWC Workshop on Uncertainty Reasoning for the Semantic Web*, Busan, Korea, 2007.

[4] D. Calvanese e G. De Giacomo. *The Description Logics Handbook: Theory, Implementation and Applications*, páginas 184–225. Cambridge University Press, 2003.

[5] R.N. Carvalho, P.C.G. Costa, L.L. Santos, S. Matsumoto, M. Ladeira e K.B. Laskey. A First-Order Bayesian Tool for Probabilistic Ontologies. In *13th Uncertain Reasoning Special Track (UR'2008) at the 21st International Florida Artificial Intelligence Research Society Conference (FLAIRS-21)*, Coconut Grove, Florida, EUA, 2008.

[6] R.N. Carvalho, L.L. Santos, M. Ladeira e P.C.G. Costa. A GUI Tool for Plausible Reasoning in the Semantic Web using MEBN. In *Proceedings of the Seventh International Conference on Intelligent Systems Design and Applications*, páginas 381–386, Rio de Janeiro, Brasil, Outubro 2007. IEEE Press.

[7] R.N. Carvalho, L.L. Santos, S. Matsumoto, M. Ladeira e P.C.G. Costa. *A chapter in Computational Intelligence in Information Assurance and Security.* A ser publicado por Springer-Verlag Berlin Heidelberg New York, EUA, 2008.

[8] R.N. Carvalho, L.L. Santos, S. Matsumoto, M. Ladeira e P.C.G. Costa. UnBBayes-MEBN: Comments on Implementing a Probabilistic Ontology Tool. In *IADIS Applied Computing 2008 Conference*, Algarve, Portugal, 2008.

[9] H. Chalupsky, R.M. Mac-Gregor e T.A. Russ. Powerloom manual. http://www.isi.edu/isd/LOOM/Power-Loom/documentation/manual.pdf, 2006. University of Southern California.

[10] E. Charniak. Bayesian Networks without Tears. *AI Magazine*, 1991.

[11] P. C. G. Costa, F. Fung, K. B. Laskey, K. J. Laskey e M. Pool, editores. *Proceedings of the Second ISWC Workshop on Uncertainty Reasoning for the Semantic Web*, Athens, GA, EUA, 2006.

[12] P.C.G. Costa. *Bayesian Semantics for the Semantic Web*. PhD thesis, Department of Systems Engineering and Operational Research, George Mason University, 2005.

[13] P.C.G. Costa e K.B. Laskey. PR-OWL: A Framework for Probabilistic Ontologies. In *Proceedings of the Fourth International Conference on Formal Ontology in Information Systems*, Baltimore, EUA, 2006.

[14] P.C.G. Costa e K.B. Laskey. Consistency rules for building MTheories. http://www.pr-owl.org/mebn/consistencyrules.php, 2007.

[15] P.C.G. Costa e K.B. Laskey. Graphical conventions for representing MEBN nodes. http://www.pr-owl.org/mebn/graphrules.php, 2007.

[16] P.C.G. Costa, K.B. Laskey e K.J. Laskey. PR-OWL: A Bayesian Ontology Language for the Semantic Web. In *Proceedings of the ISWC Workshop on Uncertainty Reasoning for the Semantic Web*, Galway, Irlanda, 2005.

[17] P.C.G. Costa, K.B. Laskey, K.J. Laskey e M. Pool, editores. *Proceedings of the First ISWC Workshop on Uncertainty Reasoning for the Semantic Web*, Galway, Ireland, 2005.

[18] E. Davis. *Representation of Commonsense Knowledge*. Morgan Kaufmann, San Mateo, California, EUA, 1990.

[19] Stanford Center for Biomedical Informatics Research. Protégé-OWL. http://protege.stanford.edu/, 2008.

[20] R. Giugno e T. Lukasiewicz. P-SHOQ(D): A Probabilistic Extension of SHOQ(D) for Probabilistic Ontologies in the Semantic Web. In *European Conference on Logics in Artificial Intelligence (JELIA 2002)*, Cosenza, Itália, 2002.

[21] J. Heflin. OWL Web Ontology Language - Use Cases and Requirements (W3C Recommendation). www.w3.org/TR/2004/REC-webont-req-20040210, 2004.

[22] J. Heinsohn. Probabilistic Description Logics. In *Tenth Conference on Uncertainty in Artificial Intelligence (UAI-94)*, Seattle, WA, EUA, 1994.

[23] M. Jaeger. Probabilistic Reasoning in Terminological Logics. In *Fourth International Conference on Principles of Knowledge Representation and Reasoning (KR94)*, Bonn, Alemanha, 1994.

[24] H. Knublauch. Protégé-OWL API Programmer's Guide. http://protege.stanford.edu/plugins/owl/api/guide.html, 2006.

[25] D. Koller e A. Pfeffer. Object-Oriented Bayesian Networks. In *Thirteenth Conference on Uncertainty in Artificial Intelligence (UAI-97)*, San Francisco, CA, EUA, 1997.

[26] M. Ladeira, D.C. Silva, M.H.P. Vieira, M.S. Onishi, R.N. Carvalho e W.T. Silva. Ferramenta Aberta e Independente de Plataforma para Redes Probabilísticas. In *Encontro Nacional de Inteligência Artificial - ENIA*, Campinas, Brasil, 2003.

[27] K.B. Laskey. MEBN: A Language for First-Order Bayesian Knowledge Bases. *Artificial Intelligence*, 172(2–3):172–225, 2007.

[28] K.B. Laskey e P.C.G. Costa. Of Klingons and Starships: Bayesian Logic for the 23rd Century. In *Proceedings of the Twenty-first Conference Uncertainty in Artificial Intelligence (UAI-05)*, páginas 346–353, Edinburgh, Escócia, 2005.

[29] K.B. Laskey e S.M. Mahoney. Network Fragments: Representing Knowledge for Construction Probabilistic Models. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence (UAI-97)*, Brown University, Providence, Rhode Island, EUA, Agosto 1997.

[30] K.J. Laskey, K.B. Laskey e P.C.G Costa. Uncertainty Reasoning for the World Wide Web Incubator Group Charter (W3C Incubator Activity). www.w3.org/2005/Incubator/urw3/charter, 2007.

[31] P.F. Patel-Schneider, P. Hayes e I. Horrocks. OWL Web Ontology Language - Semantics and Abstract Syntax (W3C Recommendation). www.w3.org/TR/owl-semantics/, 2004.

[32] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, San Mateo, CA, EUA, 1988.

[33] E. Rich. *Inteligência Artificial*. McGraw-Hill, São Paulo, Brasil, 1998.

[34] S. Russell e P. Norvig. *Artificial Intelligence: a Modern Approach*. Prentice-Hall, New Jersey, EUA, 2002.

[35] B. Smith. Ontology and Information Systems. http://ontology.buffalo.edu/smith//articles/ontologies.htm, 2004. Draft.